JDI - JTAG Debug Interface

User Manual

Revision: 1.4
Date: Feb. 2009

北京君正集成电路有限公司
Ingenic Semiconductor Co. Ltd

JDI – JTAG Debug Interface

User Manual

Release history

Date	Revision	Change
Dec. 2008	1.4	- Modified configuration file NAND and SDRAM related
Jan. 2007	1.3	- Added new commands: nquery, nreadraw, nreadoob, memtest, gpios, gpioc. - Modified some commands usage: nread, nprog. - Added comments to scale PLL etc.
Aug. 2006	1.2	- Added new commands (showcfg/reset). - Removed command (config). - Modified command (debug).
July 2006	1.1	- Added description for new commands.
Apr. 2006	1.0	- First release

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co. Ltd

Room 108, Unit A, Building 1, Zhongguancun Software Park,
Haidian District, Beijing 100193, China
Tel: 86-10-82826661
Fax: 86-10-82825845
Http: //www.ingenic.cn

Content

1 Overview

JDI is a powerful and flexible JTAG debug tool provided by Ingenic for product development and debugging. JDI can be used for Flash/EEPROM programming on target board, and device driver and OS low level debugging.

Features

- Connected to HOST PC through Ethernet connection which provides high data throughput
- Powered through USB power, no extra power supply needed
- Configuration of JDI is through RS-232C
- Use Linux as it's OS
- Providing a Command Line Interface (CLI) through either Telnet or serial port connection.
- JDI access target board's configuration file through TFTP
- Low level debugging of target board though EJTAG protocol.

Debugging function of JDI is designed or OS low level programs, not application programs. OS provides mechanism (For example gdbserver) for application program's debugging.

JDI can be adapted to other CPU architecture through updating JDI firmware.

2 Tutorial

The major function of JDI is Flash/EEPROM programming and program debugging. This chapter gives a tutorial of using JDI for these functions. Detail operations are list in later chapters.

JDI needs a host machine which provides TFTP service. In this chapter, we use a LINUX PC whose IP address is 192.168.1.20 to serve this purpose.

When starting, JDI will access target board's configuration file though TFTP protocol.

In this tutorial, NOR flash chip on the demo's target board is Intel E28F128(J3A150), and NAND flash chip is SAMSUNG K9F280.

2.1 Preparation

Only USB and RS232 cables are needed to connect to JDI when doing initialization for JDI. After initialization, 3 cables (USB, Ethernet, JTAG） are required for the normal work. RS232 will not affect JDI's further operation after initialization is finished.

2.1.1 Connect JDI

Connect JDI to target board according the following method:

- Connect JDI to PC or target board though USB cable. This provides JDI's power supply.
- Connect JDI to PC through RS232 cable. This is used to configure JDI or use JDI's command line interface.
- Connect JDI to networks through Ethernet cable.
- Connect JDI to target board through JTAG cable.

If the connection is correct, there will be output message on serial port console.（The serial port's configuration is ： 115200bps, 8N1）。

2.1.2 Start TFTP service

TFTP service needs to be configured and started on host machine.

- For Linux，start TFTP service according to the following

First, edit /etc/xinetd.d/tftp as following (Use /tftpboot as the directory to provide TFTP service)：

```
service tftp
{
   disable                 = no
   socket_type           = dgram
   protocol                = udp
   wait                     =   yes
   user                    =    root
   server                  =    /usr/sbin/in.tftp.d
   server_args           =   -s /tftpboot
   … …
}
```

Then, run the following command:

/etc/init.d/xinetd restart

- For Windows，install a third party TFTP server and configure it to work properly.

2.1.3 Prepare configuration file

Configuration file is used to describe target board's features and specify some predefined operation on target board. These predefined operations can perform some essential configuration for target board which eases later development. For example, some predefined operations perform PLL initialization and SDRAM configuration. These SDRAM can be used by GDB for user application downloading directly.

Ingenic supports a development board for each CPU, pmp board for JZ4730, pavo board for JZ4740, and apus board for JZ4750. There are 6 configuration files for the 3 boards: jz4730_pmp_boot.cfg, jz4730_pmp_yaffs2.cfg, jz4740_pavo_boot.cfg, jz4740_pavo_yaffs2.cfg, jz4750_apus_boot.cfg, jz4750_apus_yaffs2.cfg.

jz47xx_xxx_boot.cfg is used for programming bootloader or kernel image, jz47xx_xxx_yaffs2.cfg is used for programming YAFFS2 image file.

The following jz4730_pmp_uboot.cfg lists an example of configuration file:

```
;-------------------------------------------
; JDI configuration file for Jz4730 PMP board
;-------------------------------------------
;
```

```
[INIT]
; Init INTC
WM32    0xB0001010   0xFFFFFFFF   ;clear all intrs
WM32    0xB0001008   0xFFFFFFFF   ;mask all intrs
; Init PLL
WM32    0xB0000000   0x10422220   ;PLL frequency division register
WM32    0xB0000010   0x50800520   ;PLL control register
; Init memory controller
WM32    0xB0010070   0x40000000   ;GPALR2, GPIO as emc
WM32    0xB0010074   0x00005555   ;GPAUR2, GPIO as emc
;
[HOST]
IP      192.168.1.20         ;The Host IP
;
[FLASH]
CHIPTYPE    AM29             ;FLash type (AUTO|AM29|I28F)
CHIPSIZE    0x00800000       ;The size of the flash in bytes
;; Parameters for chiptype I28F and AM29
CHIPWIDTH   16               ;The width of the flash chip in bits (8|16|32)
BUSWIDTH    16               ;The width of the flash memory bus in bits (8|16|32)
;; Parameters for chiptype AM29
SETUPADDR1  0x555            ;The first setup address of chiptype AM29
SETUPADDR2  0x2AA            ;The second setup address of chiptype AM29
;
[NAND]
BUSWIDTH        8       ;The width of the NAND flash chip in bits (8|16|32)
ROWCYCLES       3       ;The row address cycles (2|3)
PAGESIZE        2048    ;The page size of the NAND chip in bytes(512|2048)
PAGEPERBLOCK    64      ;The page number per block
OOB             0       ;Whether the binary contains oob data? (0|1)
OOBECC          0       ;Whether the oob data of the binary contains ECC data? (0|1)
ECCPOS          4       ;ECC offset in oob (0-[oobsize-1]), it should be 4*N for jz4730
BADBLOCKPOS     0       ;The badblock flag offset inside the oob (0-[oobsize-1])
BADBLOCKPAGE    63      ;The page number of badblock flag inside a block(0-[PAGEPERBLOCK-1])
FORCEERASE      0       ;The force to erase flag (0|1). When set, all blocks including that
                        ; were marked as bad blocks will be erased. Don't set it during normal
                        ; operation.
;WP_PIN                 ;Specify the write protect pin number, e.g. GPD23=32*3+23=119
;
[EEPROM]
ADDRESS     7       ;The device address of EEPROM
;
[SDRAM]
BUSWIDTH    32      ;The bus width of the SDRAM in bits (16|32)
```

```
BANKS        4        ;The bank number (2|4)
ROWADDR      13       ;Row address width in bits (11-13)
COLADDR      9        ;Column address width in bits (8-12)
CASLATENCY   2        ;CAS latency (2|3)
;
[GDB]
INITFILE     gdbinit.bin  ;Platform-dependent init code, required for gdb debugging
;
;; -- END --
```

Detailed description of the configuration file is in section <Configuration File>.

Copy the configuration file to the directory where HOST machine provides TFTP service. This finishes the preparation of configuration file.

2.1.4 Configure JDI

JDI needs the following information before working: network information, JDI's IP address, TFTP host machine's IP address, configuration file's path and name in TFTP's directory. The configuration of JDI is performed through RS232 port.

The configuration process needs to be performed in two situations: The first time when using JDI, or the configuration parameters need to be changed.

Before configuration, do the following step:

- Disconnect Ethernet cable
- Connect JDI to PC though RS232 cable. The serial port is configured as 115200bps, 8N1
- Power JDI though USB connection. This starts JDI.

After JDI is started, the following output is displayed in serial port console:

JDI>

Next step, type the following commands and parameters to configure JDI and network:

JDI> *ipconfig 192.168.1.201*
JDI> *fconfig 192.168.1.20 jz4730_pmp.cfg*
JDI> *netmask 255.255.255.0*
JDI> *gateway 255.255.255.255*
JDI> *bcast 192.168.1.255*

In above command, 192.168.1.201 is JDI's IP address, 192.168.1.20 is TFTP Host's IP address, jz4730_pmp.cfg is the configuration file's name and data path information.

The above commands will store the information into JDI's internal flash. JDI configuration is then finished.

2.1.5 Entering into JDI's Command Line Interface

First, connect JDI to target board correctly and connect JDI to PC though serial port. Configure the serial port on PC with the following data:

Baud rate	:	115200 bps
Data bits	:	8
Even/odd check	:	none
Stop bits	:	1
Flow control	:	none

Then restart JDI through re-plugging USB cable. The output of JDI's command line interface is displayed on serial port's console. When following JDI's command prompt "JDI>" is displayed, users can type commands for execution.

```
JDI >
```

Users can also enter into JDI's command line interface through Telnet logon which is more powerful:

```
$ telnet 192.168.1.201
JDI>
```

Operations of JDI can be performed in this interface through typing command.

2.1.6 Exiting from JDI

Disconnect USB cable and shut down the power on target machine. Disconnect all the other cables on JDI.

2.2 EEPROM programming

JDI can be used to program EEPROM on target board. EEPROM information needs to be specified in configuration file for JDI to access EEPROM correctly.

2.2.1 Edit configuration file

Add the following information to configuration file

```
[EEPROM]
ADDRESS          7
```

[EEPROM] indicated the start of EEPROM segment which describes EEPROM parameters required

by JDI. Number ‘7’ indicates EEPROM’s address，which is target board specific. Refer to hardware specification for different platform.

JDI will re-read the configuration file before executing commands, so JDI need not to be restarted after configuration file is modified.

2.2.2 Program EEPROM

Suppose we want to update EEPROM’s 8-15 bytes with the following data:0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88。

First, read the content of the 8 bytes through the following command:

```
JDI> eread 8 8
0x008: 00 00 00 00 00 00 00 00
JDI>
```

Then update the 8 bytes with data “0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88” through the following command:

```
JDI> eprogv 8 1122334455667788
JDI>
```

At last, read the data again to check if the contents are update correctly:

```
JDI> eread 8 8
0x008: 11 22 33 44 55 66 77 88
JDI>
```

Arbitrary length of data can be programmed to EEPROM according to above procedure. If the amount of data programmed to EEPROM is too big, users can put the data into a data file and put the data file to TFTP directory. Users then program the data file to EEPROM. The detailed procedure is illustrated in chapter 4.

2.3 NOR FLASH programming.

Before programming, users need to study the hardware specifications of the NOR flash on target board and supply the information in JDI configuration file. In this chapter, we take a popular NOR flash chip on Jz4730 development board to illustrate the NOR flash programming procedural.

2.3.1 Edit configuration file

In this example, we add the following information to the configuration file jz4730.cfg：

```
[FLASH]
CHIPTYPE          I28F
CHIPSIZE          0x01000000
CHIPWIDTH         16
BUSWIDTH          16
```

‘[FLASH]’ indicates the start of FLASH segment. This segment describes NOR flash chip related parameters. See chapter 4 for detailed description of this segment.

Flash of similar type can use the same algorithm during NOR flash programming process. In this example, the CHIPTYPE field is I28F.

CHIPSIZE specifies the size of the Flash. In this example, the flash we used is 128M bits，so CHIPSIZE is designated to 0x01000000（16M bytes）.

‘CHIPWIDTH 16’ specifies the data width of the flash on target board is 16 bits.

‘BUSWIDTH 16’ specifies the data width on the data bus when accessing the FLASH.（Some target system can accesses two 16 bits FLASH simultaneously through its 32 bits data bus. Through this way, two 16 bits FLASH can be combined to form 32 bits data, while for each single FLASH, the data width is still 16 bits）.

JDI will re-read the configuration file before executing commands, so JDI need not to be re-start after configuration file is modified.

2.3.2 Program NOR FLASH

Put the content data file that will be programmed to FLASH into a directory provided by TFTP service. In this example, we use a 256K data file named “test256k.bin” and copy the file to sub-directory jdi. We assume the host PC’s IP address is 192.168.1.20 in this example.

2.3.2.1 Read NOR FLASH data

Before programming NOR FLASH，we first read the data in NOR flash and check its content. Type the following command:

```
JDI> readf 0xbfc00000   128

0xbfc00000:   00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0xbfc00010:   00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0
0xbfc00020:    …   … … … …
… … … …        …   … … … …
0xbfc00070:    …   … … … …
```

```
JDI>
```

Parameter “0xbfc00000” is the starting address of NOR flash and parameter 128 is the number of bytes that this operation will read.

2.3.2.2 Erase NOR FLASH

The blocks that will be programmed need to be erased before programming for NOR flash. In this example, the 256K size data file will be programmed to the space from starting of the NOR flash. The block size of the NOR flash we used is 128K bytes, so 2 blocks from the beginning of the NOR flash need to be erased. The command is as following:

```
JDI> erase 0xbfc00000   0x20000   2
JDI>
```

The first parameter 0xbfc00000 is the starting address of NOR flash, the second parameter 0x20000 is the block size of the NOR flash (128K), and the third parameter 2 is the number of blocks that will be erased.

After erase command is executed, we use readf to read the contents of the NOR FLASH’s first 2 blocks and find its data is 0xFF. This means the 2 block are erased correctly.

```
JDI> readf 0xbfc00000    128
0xbfc00000:    ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff
0xbfc00010:    ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff
…   … …              … … … …
0xbfc00070:    ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff  ff

JDI>
```

2.3.2.3 Programming NOR FLASH

Programming operation can be performed after the blocks are properly erased. The following command program a binary data file into NOR flash:

```
JDI> prog 0xbfc00000 192.168.1.20 jdi/test256k.bin
JDI>
```

The first parameter 0xbfc00000 is the starting address of NOR flash; the second parameter 192.168.1.20 is the IP address of the host PC which provides TFTP service; the third parameter jdi/test256k.bin is the data file that will be programmed to NOR FLASH.

After the above command is finished, type the following command to verify that the data are programmed correctly:

```
JDI> readf 0xbfc00000    128
0xbfc00000:   12  34  56  78  23  45  67  89  ab  cd  ef  34  56  78  9a  bc
0xbfc00010:   aa  bb  cc  dd  ee  ff  24  68  ac  e0  13  57  9b  df  11  22
…   … …            … … … …
0xbfc00070:   11  33  55  77  99  00  22  44  66  88  ab  cd  ef  10  22  30

JDI>
```

Compare the above data with data file jdi/test256k.bin，make sure data are programmed to NOR flash.

2.4 NAND Flash Programming

Before programming, users need to study the hardware specifications of the NAND flash on target board and supply the information in JDI configuration file. In this section, the NAND flash we used has the following features: Data width is 8 bits; 4096 blocks in total; each block contains 128 pages; the page number of bad block flag inside a block is 127; the bad block flag offset inside the oob is 0; the size of each page is 2048 bytes; write row address to needs 3 cycles.

2.4.1 Prepare Configuration File

Add the following information into the configuration file according to the NAND flash specification in the above example:

```
[NAND]
BUSWIDTH        8       ;The width of the NAND flash chip in bits (8|16|32)
ROWCYCLES       3       ;The row address cycles (2|3)
PAGESIZE        2048    ;The page size of the NAND chip in bytes(512|2048)
PAGEPERBLOCK    128     ;The page number per block
OOB             1       ;Whether the binary contains oob data? (0|1)
OOBECC          0       ;Whether the oob data of the binary contains ECC data? (0|1)
ECCPOS          28      ;ECC offset in oob (0-[oobsize-1])
BADBLOCKPOS     0       ;The badblock flag offset inside the oob (0-[oobsize-1])
BCHBIT          4       ;Specify the hardware BCH algorithm for JZ4750 (4|8)
BADBLOCKPAGE    127     ;The page number of badblock flag inside a
                        ;block(0-[PAGEPERBLOCK-1])
FORCEERASE      0       ;The force to erase flag (0|1). When set, all blocks including that
                        ;were marked as bad blocks will be erased. Don't set it during
                        ;normal operation.
WP_PIN          119     ;Specify the write protect pin number, e.g. GPD23=32*3+23=119
;
```

In this configuration file:

‘[NAND]’ indicates the start of NAND flash segment which describes NAND flash specifications.

‘OOB 1’: Indicates the binary contains oob data. It should be 1 when programming YAFFS2 image file.

‘OOBECC 0’: Indicates the oob data of the binary doesn’t contain ECC data, and the ECC will be generated by JZ CPU.

‘BCHBIT 4’: Specify the hardware BCH algorithm for JZ4750 is 4 bit ECC, which is careless for JZ4730 and JZ4740.

‘WP_PIN 119’: Specify the write protect pin number is 119 (GPD23), and the pin will be set to high to disable write-protect when programming.

JDI doesn’t need to be restarted after configuration file is changed as JDI will re-read the configuration file before executing command.

2.4.2 Program NAND Flash

Put the content data file that will be programmed to NAND flash into a directory provided by TFTP service. In this example, we use a 256K data file named “test256k.bin” and copy this file to sub-directory jdi. We assume the host PC’s IP address is 192.168.1.20.

2.4.2.1 Read NAND Flash Data

Read the contents of NAND FLASH before programming for comparison after data is programmed to NAND flash. The command is as following:

```
JDI> nread   0   512

0x00000000:   00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0x00000010:   00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0
0x00000020:    …   … … …
… … … …       …   … … …
0x000001f0:    …   … … …

JDI>
```

The first parameter of ‘0’ means read NAND flash from the first data page. The second parameter ‘512’ means read 512 bytes data.

2.4.2.2 Erase NAND Flash

The blocks that will be programmed need to be erased before programming for NAND flash. In this example, the 256K size data file will be programmed to the space from starting of the NAND flash. The block size of the NAND flash we used is 16K（32 x 512bytes）bytes, so the first 16 blocks at the

beginning of the NAND flash need to be erased. The command is as following:

```
JDI> nerase    0   16
JDI>
```

Parameter ‘0’ means to erase NAND FLASH from block 0, and parameter ‘16’ means the number of blocks that will be erased is 16

After erase operation is finished, use nread command to read the contents of the NAND flash’s first 2 blocks and find its data is 0xFF. This means the 2 block are erased correctly.

```
JDI> nread   0   512

0x00000000:   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x00000010:   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x00000020:     …   … … …
… … … …        …   … … …
0x000001f0:     …   … … …

JDI>
```

2.4.2.3 Program NAND Flash

NAND Flash can be programmed after the erase operation is performed properly. The following command program the data file into NAND flash:

```
JDI> nprog    0    192.168.1.20   jdi/test256k.bin
JDI>
```

Parameter ‘0’ indicates to program NAND flash from page 0, parameter ‘192.168.1.20’ is the IP address of the host PC that provides TFTP service, parameter ‘jdi/test256k.bin’ is the file containing data that will be programmed to NAND flash.

If the blocks that will be programmed are not erased properly, data couldn’t be programmed to NAND flash correctly.

Read the contents of NAND flash to check the data are programmed into correctly:

```
JDI> nread   0   512
0x00000000:   12  34  56  78  23  45  67  89  ab  cd  ef  34  56  78  9a  bc
0x00000010:   aa  bb  cc  dd  ee  ff  24  68  ac  e0  13  57  9b  df  11  22
…   … …            … … … …
0x000001f0:   11  33  55  77  99  00  22  44  66  88  ab  cd  ef  10  22  30
```

JDI>

Comparing the data read from NAND flash with the data in file jdi/test256k.bin，make sure data is programmed to NAND flash correctly.

2.5 Debugging with GDB

In addition to the above functionalities, JDI provides GDB a debugging interface through Ethernet connection. Users can debug low level program through connect Ethernet to JDI. In this section, we illustrate the debug processing through a small program.

Programs need to be downloaded to target board's memory before debugging. The configuration of memory on target board is different and therefore initialization also varies significantly. JDI solves this problem by supply a INITFILE in the JDI configuration file, this will tell GDB to run the init code specified by INITFILE to init the target board. The init code is a binary, its program entry is 0xff2f,0000, and must be linked to addresses inside 0xff2f,0000 to 0xff2f,1000.

Before running GDB, you need to run a JDI command to enable remote debugging:

JDI> *debug 0x80001000*
******** pid = 0x3e8 ********
Listening on port 6666

0x80001000 is the program entry address of the debuggee. The parameter is optional. On the host, run mipsel-linux-gdb to debug the program. In this example, we generate a ELF file t.elf from a simple assembly file, and the start address is specified at address 0x80001000：

$ *./mipsel-linux-gdb ./t.elf*
(gdb) *target remote 192.168.1.201:6666*
Remote debugging using 192.168.1.201:6666
0x00000000 in ?? ()
(gdb)

In above command‘target remote 192.168.1.201:6666’，‘192.168.1.201’is the IP address of JDI，‘:6666’is the TCP port number that JDI provides to GDB. After the command is executed, GDB made connection with JDI. Next step, program can be downloaded to target through the following command:

(gdb) *load*

Program is downloaded to target's memory after load command is finished. Debugging the program with the following GDB commands:

```
(gdb) disp/i    $pc
1: x/i $pc    0x800010b0    <_ftext>:   lui     a3, 0xb600
(gdb) si
1: x/i $pc    0x800010b0    <_ftext>:   lui     a3, 0xb600
(gdb) si
0xffffffff800010b4    in    _ftext ()
1: x/i $pc    0x800010b4    <_ftext + 4>:   ori     a3, a3, 0x58
(gdb) p/x $a3
$1 = 0xb6000000
(gdb) si
0xffffffff800010b8    in    _ftext ()
1: x/i $pc    0x800010b8    <_ftext + 8>:   lui     t0, 0x30
(gdb) si
0xffffffff800010bc    in    _ftext ()
1: x/i $pc    0x800010bc    <_ftext + 12>:   sb     t0, 0(a3)
(gdb) p/x $t0
$2 = 0x30
(gdb) q
```

The commands in the above debugging process are GDB standard command. Refer GDB document for details of these command. All these commands are executed in GDB, the connection between JDI and GDB is transparent to users.

3 JDI Command Line Interface

JDI provides a command line interface (CLI) supporting the following 3 kinds of commands: configure and query operations, FLASH and EEPROM programming operations, test and upgrade operations.

Make sure GDB is not performing debugging operation when performing operations on CLI, otherwise the operations couldn't complete correctly.

3.1 Enter and Exit CLI

There are 2 ways to enter/exit JDI's CLI: through serial port console, or though Telnet remote logon.

3.1.1 Access CLI though Serial Port

First, connect JDI to target board correctly and connect JDI to PC though serial port. Configure the serial port on PC with the following data:

Baud rate	:	115200 bps
Data bits	:	8
Even/odd check	:	none
Stop bits	:	1
Flow control	:	none

Then restart JDI through re-plugging USB cable. The output of JDI's command line interface is displayed on serial port's console. When JDI's command prompt "JDI>" is displayed, users can type commands for execution.

```
JDI> help
JDI Commands:

help                print this help
ipconfig            set local ip
… …                 … … …

JDI>
```

To exit the CLI interface, just close the serial port console on PC.

3.1.2 Access CLI though Telnet Logon

Another method to access JDI is through Telnet remote logon. Users can logon to JDI from any

computer on the network through telnet command as following:

$ *telnet 192.168.1.201*
JDI>

After command prompt “JDI>”is displayed，JDI is ready for accepting command. ‘192.168.1.201’ is JDI’s IP address。Users can enter command at this time, for example:

JDI> *help*

EXIT command will quit JDI:

JDI> *exit*
$

3.2 Configure and Query Commands

Configure and query commands are used for the following functions:

- Modify JDI configuration information
- Query NOR flash information on target board
- Check all the commands supported by JDI’s Command Line Interface

3.2.1 ‘help’ Command

‘help’ is used to query the commands supported by JDI.

JDI> *help*
JDI Commands:

help	print this help
ipconfig	set local ip
… …	… … …

JDI>

‘help’ lists all the commands and explanations of the commands. This command helps users to find the usage of a command quickly.

3.2.2 ‘ipconfig’ command

This command is used to configure the JDI IP address. The usage of this command is as following:

ipconfig <JDI IP ADDRESS>

JDI> *ipconfig 192.168.1.201*
… … …
JDI>

3.2.3 'fconfig' command

This command is used to configure the JDI configuration file information. The usage of this command is as following:

fconfig <HOST IP ADDRESS> <CONFIGURATION FILE>

JDI> *fconfig 192.168.1.20 jdicfg/jz4730.cfg*
… … …
JDI>

3.2.4 'netmask' command

This command is used to configure the netmask of the network. The usage of this command is as following:

netmask <NETMASK>

JDI> *netmask 255.255.255.0*
… … …
JDI>

3.2.5 'gateway' command

This command is used to configure the gateway IP address of the network. The usage of this command is as following:

gateway <GATEWAY IP ADDRESS>

JDI> *gateway 255.255.255.255*
… … …
JDI>

3.2.6 'bcast' command

This command is used to configure the broadcast IP address of the network. The usage of this command is as following:

bcast <BROADCAST IP ADDRESS>

JDI> *bcast 192.168.1.255*
… … …
JDI>

3.2.7 Display configuration command ‘showcfg’

This command is used to query and display current JDI configurations. The usage of this command is as following:

showcfg

JDI> *showcfg*

LOCAL IP: 192.168.1.201
HOST IP : 192.168.1.20
CFGFILE: jdi.cfg
…

3.3 GDB debug command ‘debug’

This command is used to enable remote GDB debugging. The usage of this command is as following:

debug [program_entry]

[program_entry] is the program entry of the debuggee, it tells GDB the program entry. This parameter is optional.

For example:

JDI> *debug*
******** pid=0x3e8 ********
Listening on port 6666

3.4 Flash and EEPROM Manipulation Commands

These commands perform the following operation on FLASH and EEPROM:

- EEPROM read/write operation
- NOR flash read/erase/program operation
- NAND flash read/erase/program

Before executing of these commands, the information in JDI configuration file needs to be edited properly to reflect target board's memory configurations.

3.4.1 'eread' EEPROM read command

'eread' is used to read the content of EEPROM. The syntax of 'eread' has the following types:

'eread start count'

This command reads EEPROM from address offset **start**, the amount of data is **count** byte. The command display the data read on sceen.

'eread start count file-name'

This command reads **count** bytes from EEPROM beginning from address offset **start**, and writes the data into a file '**file-name'.** This command requires '**file-name**' is an existent file locating at TFTP directory. This command couldn't create a new file on TFTP host, but can override the existent file. '**file-name**' can contain directory path which is the relative path from TFTP directory. The IP address of TFTP host is specified in the [HOST] segment of the configuration file.

'eread start count host file-name'

This command is similar with the above one. The different is: this command uses **host** as the IP address of the TFTP host machine; the IP address in configuration file's [HOST] segment is ignored.

For example, the following command read Byte 12 to Byte 19 from EEPROM:

```
JDI> eread    12     8
0x00c:    11 22 33 44 55 66 77 88
JDI>
```

The following command writes the first 256 bytes of EEPROM into data file jdi/eeprom.bin:

```
JDI>eread   0    256    192.168.1.20    jdi/eeprom.bin
```

3.4.2 'eprogv' program EEPROM from string

This command is used to program EEPROM from a number of consecutive data. The syntax of this command is as following:

'eprogv start hex'

'hex' is hex data representing with a string, **'start'** is the start offset in EEPROM. For example:

```
JDI> eprogv    12    1020304050607080
```

This command update EEPROM from Byte 12 to Byte 19 with '0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80'. Type the following command to make sure the 'eprogv' command is finished correctly.

```
JDI> eread  12  8
0x00c:  10 20 30 40 50 60 70 80
JDI>
```

3.4.3 'eprog' Program EEPROM from data file

This command is used to program EEPROM from the content of data file in TFTP directory. This command has two types of syntax:

'eprog start file-name'

'**file-name**' is the file name in TFTP directory. The content of this file will be programmed to EEPROM. The TFTP host PC's IP address is specified in JDI configuration file's [HOST] segment. '**start**' is the address offset of EEPROM

For example, the following command program the content in file *jdi/eeprom.bin* to EEPROM starting from offset 0.

```
JDI> eprog  0  jdi/eeprom.bin
```

'eprog start host file-name'

This command differs from the previous one with **host** parameter. **'host'** is the IP address that provides TFTP service. The IP address specified in JDI configuration file's [HOST] segment is ignored.

For example, the following command program the content in file *jdi/eeprom.bin* located at computer 192.168.1.30 to EEPROM starting from offset 0.

```
JDI> eprog  0  192.168.1.30  jdi/eeprom.bin
```

3.4.4 'query' NOR Flash query command

This command is used to query NOR FLASH information on target board. NOR flash information is not available if the type of NOR flash in configuration file is not AUTO. The usage of this command is as following:

'query address'

'**address**' is the start address of NOR flash on JZ CPU. It's 0xbfc0000 for JZ4730 pmp board, and 0xa8000000 for JZ4750 apus board.

JDI> *query* 0xa8000000
FLASH querying ... completed.
Flash type is AM29.
Vendor ID is 0x1, Product ID is 0x227e.
JDI>

3.4.5 'querya' NOR Flash and its sector query command

This command is used to query the information of target board's NOR flash and the address of each sector of NOR FLASH. NOR flash information and sector address is not available if the type of NOR flash in configuration file is not AUTO. The usage of this command is as following:

JDI> *querya*
… … …
JDI>

3.4.6 'readf' NOR flash read command

'**readf**' command read NOR flash data from a start address. The address in the command must be in the boundary of NOR flash address and the amount of data couldn't exceed NOR flash's boundary. This command has the following two types:

'readf address count'

This command reads '**count**' consecutive bytes starting from NOR flash address '**address**'. The data read from NOR flash is displayed on screen.

'readf address count file-name'

This command reads '**count**' consecutive bytes starting from NOR flash address '**address**'. The data read from NOR flash is written into data file '**file-name**'. This command requires '**file-name**' is an existent file locating at TFTP directory. This command couldn't create a new file on TFTP host, but can override the existent file. '**file-name**' can contain directory path which is the relative path from TFTP directory. The IP address of TFTP host is specified in the [HOST] segment of the configuration file.

'readf address count host file-name'

This command differs from the previous one with **host** parameter. **'host'** is the IP address that provides TFTP service. The IP address specified in JDI configuration file's [HOST] segment is ignored. Other operations are same.

For example, the following command reads 256 bytes from NOR flash address 0xbfc00000 and display the result on screen:

JDI> *readf 0xbfc00000 256*

```
0xbfc00000:   00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0xbfc00010:   00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0
0xbfc00020:    …   … … …
… … … …       …   … … …
0xbfc00100:    …   … … …

JDI>
```

For example, the following command reads 256 bytes from NOR flash address 0xbfc00000 and write the result into file name jdi/norflash.bin:

```
JDI> readf      0xbfc00000    256     jdi/norflash.bin
JDI>
```

3.4.7 'erase' NOR flash erase command

'erase' command is used to erase the data on NOR flash. NOR flash content must be erased before being programmed, otherwise NOR flash couldn't be programmed correctly. 'erase' command has the following syntax:

'erase address chip'

This command performs erase operation on the whole chip. This command is applicable only to NOR flash chip that support such kind of operation. **'address'** is an arbitrary address on NOR flash, **'chip'** is string "chip"。

'erase address step count'

This command erases one or more consecutive blocks on NOR flash. **'address'** is the address of the first data block; **'step'** is the size of each data block in byte; **'count'** is the number of data blocks. This command requires users to know NOR flash's address and block size.

For example，the following command erases all the data of the NOR flash on target board (Assume there is only one NOR flash chip on the target board, and the start address for NOR flash is 0xbfc00000) .

```
JDI> erase      0xbfc00000    chip
JDI>
```

The following command erases the first 256K data on the NOR flash of target board (Assume the start address for NOR flash is 0xbfc00000, and each data block size is 0x20000).

```
JDI> erase 0xbfc00000    0x20000   2
JDI>
```

3.4.8 'prog' NOR flash programming command

This command is used to program NOR flash from the content of a data file in TFTP directory. Erase operation must be performed before programming. This command has two types of syntax:

'prog address file-name'

'**file-name**' is the file name whose contents will be programmed to NOR flash. If the file name contains no path information, the file is located under the directory where TFTP provides service. If the file name contains path information, the file is located under sub-directory of the directory where TFTP provides service. In this command, the IP address's of the host who provides TFTP service is specified in [HOST] segment of JDI's configuration file.

After the command is executed, the content of '**file-name**' is programmed into NOR flash starting from '**address**'.

'prog address host file-name'

In this command, '**host**' is the IP address of the host who provides TFTP service. The processing of this command is similar to the previous one. The difference is: '**file-name**' is located on the machine specified by the '**host**' who provides TFTP service.

For example, the following command programs the content of a 256K size data file **jdi/test256k.bin**（This file is located under the directory provides by TFTP service） into the first 256K bytes of NOR flash (Assume the start address of NOR flash on target board is 0xbfc00000)：

JDI> *prog 0xbfc00000 jdi/test256k.bin*
JDI>

If the data file is located on another computer (Assume the IP address of that machine is 192.168.1.30), the command is as following:

JDI> *prog 0xbfc00000 192.168.1.30 jdi/test256k.bin*

3.4.9 'nquery' NAND flash query command

'**nquery**' is used to query for NAND flash information, such as manufacture ID, chip ID and chip size etc.:

JDI> *nquery*
NAND querying … completed.
NAND device: Vendor ID 0xec, Chip ID 0xf1 (Samsung NAND 128MiB 3,3V 8-bit)

3.4.10 'nread' NAND flash read command

'nread' reads a number of consecutive bytes from NAND flash and display the content on screen or write into a data file. It will skip bad blocks, and during read operation, hardware ECC will be performed. This command has the following 3 kinds of syntax:

'nread start count'

This command reads **'count'** bytes from NAND flash starting from page address **'start'** and displays the content on screen.

'nread start count file-name'

This command reads **'count'** bytes from NAND flash starting from page address **'start'** and writes the content into data file **'file-name'**. This command requires "**file-name**" is an existent file locating at TFTP directory. This command couldn't create a new file on TFTP host, but can override the existent file. '**file-name**' can contain directory path which is the relative path from TFTP directory. The IP address of TFTP host is specified in the [HOST] segment of the configuration file.

'nread start count host file-name'

This command is similar to the previous one. The difference is: **host** rather than the [HOST] segment in the configuration file is used to specify the TFTP host.

For example, the following command reads 512 bytes data from page 0 from NAND flash and display the result on screen:

```
JDI> nread   0   512

0x00000000:   00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
0x00000010:   00 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0
0x00000020:    …   … … …
… … … …       …   … … …
0x000001f0:    …   … … …

JDI>
```

The following command reads the first page of block 0 from NAND flash and writes the result into data file *jdi/nandflash.bin:*

```
JDI> nread    0    512    jdi/nandflash.bin
JDI>
```

3.4.11 'nreadraw' NAND flash read command

'nreadraw' reads a number of consecutive bytes from NAND flash and display the content on screen or write into a data file. It will not skip bad blocks, and during read operation, no hardware

ECC will be performed. This command usage is similar to 'nread'.

3.4.12 'nreadoob' NAND flash read command

'nreadoob' reads a number of consecutive bytes of OOB data from NAND flash and display the content on screen or write into a data file. It will not skip bad blocks, and during read operation, no hardware ECC will be performed. This command usage is similar to 'nread'.

3.4.13 'nerase' NAND flash erase command

'nerase' is used to erase the data in NAND flash. NAND flash can only be programmed after the content is erased properly , otherwise data file couldn't be programmed into NAND flash correctly. The usage of this command is as following:

'nerase start count'

This command erases a number of consecutive blocks on NAND flash. **'start'** is the address of the first block that will be erased, **'count'** is the number of data blocks that will be erased.

For example, the following command erases the first 16 data blocks on NAND flash:

JDI> *nerase 0 16*
JDI>

3.4.14 'nprog ' NAND flash programming command

This command is used to program NAND flash from the content of a data file in TFTP directory. Page data and OOB data including hardware ECC will be written to NAND flash at the same time. Hamming hardware ECC used for JZ4730 is 3 bytes length generated from 256 bytes data, and could be written to ECCPOS bytes offset to the OOB area. The value of ECCPOS is defined in JDI configuration file, and should be 4*N for JZ4730 (N = 1, 2, 3...), for example (ECCPOS = 4):

Pagesize=512

Page Data (512Bytes):

Data0(256)	Data1(256)
0 255	256 511

OOB (16Bytes):

	ECC0(3)	0	ECC1(3)	0	
0 3	4 6	7	8 10	11	12 15

Where ECC0 is the ECC value of Data0, and ECC1 is the ECC value of Data1. The 8th and 12th byte is stored with zero. It is similar for 2KB and 4KB page size NAND flash.

Reed-solomn hardware ECC used for JZ4740 is 9 bytes length generated from 512 bytes data, and could be written to ECCPOS bytes offset to the OOB area. The value of ECCPOS is defined in JDI configuration file, for example (ECCPOS = 6):

Pagesize=2048

Page Data (2048Bytes):

Data0(512)	Data1(512)	Data2(512)	Data3(512)
0 511	512 1023	1024 1535	1536 2047

OOB (36Bytes):

	ECC0(9)	ECC1(9)	ECC2(9)	ECC3(9)	
0 5	6 14	15 23	24 32	33 41	63

Where ECC0 is the ECC value of Data0, ECC1 for Data1, ECC2 for Data2, and ECC3 for Data3.

BCH hardware ECC used for JZ4750 could be 4 bit ECC and 8 bit ECC. 4-bit BCH ECC which could correct 4 bit error every 512 bytes data, is 7 bytes for 512 bytes data, while 8-bit BCH ECC which could correct 8 bit error every 512 bytes data, is 13 bytes for 512 bytes data. Whether 4-bit or 8-bit BCH ECC is used is specified by BCHBIT in JDI configuration file. And the ECC could be written to ECCPOS bytes offset to the OOB area, for example (ECCPOS = 3, BCHBIT = 4):

Pagesize=2048

Page Data (2048Bytes):

Data0(512)	Data1(512)	Data2(512)	Data3(512)
0 511	512 1023	1024 1535	1536 2047

OOB (36Bytes):

	ECC0(13)	ECC1(13)	ECC2(13)	ECC3(13)	
0 2	3 15	16 28	29 41	42 54	63

Where ECC0 is the ECC value of Data0, ECC1 for Data1, ECC2 for Data2, and ECC3 for Data3.

Erase operation must be performed properly before programming. This command has two syntaxes:

‘nprog start file-name’

‘**file-name**’ is the file name whose contents will be programmed to NAND flash. If the file name contains no path information, the file is located under the directory where TFTP provides service. If the file name contains path information, the file is located under sub-directory of the directory where TFTP provides service. In this command, the IP address’s of the host who provides TFTP service is specified in [HOST] segment of JDI’s configuration file.

After the command is executed, the content of ‘**file-name**’ is programmed into NAND flash starting from ‘**address**’.

‘nprog start host file-name’

In this command, ‘**host**’ is the IP address of the host who provides TFTP service. The processing of this command is similar to the previous one. The difference is: ‘**file-name**’ is located on the machine specified by the ‘**host**’ who provides TFTP service.

For example, the following command programs the content of a 256K size data file **jdi/test256k.bin**（This file is located under the directory provides by TFTP service） into the first 256K bytes of NAND flash :

JDI> *nprog 0 jdi/test256k.bin*
JDI>

If the data file is located on another computer (Assume the IP address of that machine is 192.168.1.30), the command is as following:

JDI> *nprog 0 192.168.1.30 jdi/test256k.bin*

3.5 Test, upgrade and exit command

These commands are used for the following function:

- Test the connectivity between JDI and target board
- Upgrade JDI firmware
- Exit JDI’s CLI command

3.5.1 ‘readids’ JDI test command

This command reads IDCODE and IMPCODE of EJTAG TAP on target board. This command can be used to test the connectivity between JDI and target board or check the type of CPU.

JDI> *readids*
target idcode is 0x0000024f, impcode is 0x20404000
JDI>

3.5.2 ‘cputest’ JDI test command

This command tests the CPU access on the EJTAG port.

JDI> *cputest*
JDI>

3.5.3 'memtest' JDI test command

This command is used to test the on-board SDRAM. The SDRAM parameters are defined in the JDI configuration file. The command syntax is:

'memtest address size'

Where 'address' is the starting address, 'size' is the testing memory size.

JDI> *memtest 0x80000000 0x1000000*
Checking memory from 0x80000000 to 0x80ffffff.
Checking memory passed

3.5.4 'gpios' and 'gpioc' JDI test commands

These two commands are used to set and clear the GPIO pin on the target board. The command syntax is:

'gpios pin'
'gpioc pin'
Where 'pin' is the pin number from 0 to 127…

3.5.5 'hwtest1' and 'hwtest2' JDI test commands

This command generates pulses on the address and data buses of the target CPU. This can help to find out the hardware problem of the target board.

JDI> *hwtest1*
JDI> *hwtest2*
JDI>

3.5.6 'reset' hardware reset command

This command generates hardware reset pulse to reset the target board.

JDI> *reset*
Resetting target … done.
JDI>

3.5.7 'version' JDI version command

This command shows the version of the JDI firmware.

JDI> *version*
The current jdi version is 01.02.03.

JDI>

3.5.8 'upgrade' JDI firmware upgrade command

This command is used to query or upgrade JDI's firmware. Before upgrade operation is performed, users need to open JDI box, turn jumper JP1 to on. Turn jumper 1 JP1 after the upgrade operation is finished. Refer chapter 5 for JDI hardware description.
The syntax of upgrade command is as following:

'upgrade'
This is used to query the version number of JDI's firmware.

'upgrade <hostip> <binary file>'
'<binary file>' is the file name that used to upgrade the firmware. It is located under the directory provided by TFTP service. **'<hostip>'** is the IP address of the host who provides TFTP service.

For example, the following command query JDI firmware's version number.

JDI> *upgrade*
The current firmware version is 01.02.00
JDI>

The following command upgrades JDI firmware:

JDI> *upgrade 192.168.1.20 JDI-FW-01.02.03*
JDI>

3.5.9 'run' batch command

This command can be used to run the batch command file on the host side. You can add one or more JDI commands in this file, and run it on the JDI by one command.

JDI> *run 192.168.1.20 test.sh*
JDI>

3.5.10 'exit' exit from Telnet

This command exit Command Line Interface login into through Telnet:

JDI> *exit*
Connection closed by foreign host.
$ >

4 Configuration File

JDI configuration file is used to describe target board's resource status and specify some predefined operation related to target board's initialization process. These predefined operations will be executed once JDI is powered on, thus make proper preparation for later operations. In the configuration file, SEGMENT is introduced to describe the various kinds of operation and configuration. A complete JDI configuration file consists of the following SEGMENTS:

[INIT] segment: This segment defines some initialization operation for target board.
[HOST] segment: This segment defines the IP address of the host who provides TFTP service.
[FLASH] segment: This segment defines NOR FLASH configuration on target board.
[NAND] segment: This segment defines NAND FLASH configuration on target board.
[EEPROM] segment: This segment defines EEPROM configuration on target board.
[SDRAM] segment: This segment defines SDRAM configuration on target board.
[GDB] segment: This segment defines GDB configuration on target board.

Some segments can be omitted according to target board's status, or none of the segments is defined. This means JDI configuration file can be an empty file.

In some circumstances, it is possible to add or modify some item in configuration file on the fly. It doesn't need to restart JDI after configuration file is modified because JDI will access the configuration through TFTP each time a command is executed.

JDI will report error message on serial port at the starting time when configuration file is not existed or contains errors.

The configuration file's format is as following:

[segment name]
Operation or Parameter 1
Operation or Parameter 2
… … … …
[segment name]
Operation or Parameter 1
Operation or Parameter 2
… … … …

… … … …

A sample configuration file is as following:

```
[INIT]
WM32        0xb3010010        0x0fff7700

[HOST]
IP                  192.168.1.20

[FLASH]
… … … …

… … … …
```

4.1 [INIT] segment

This segment defines a list of command line to specify the predefined operations. The syntax is as following:

〈command〉 〈parameter 1〉 〈parameter 2〉

These commands will be executed once JDI is powered on.

For the time being, two types of command is defined in this segment: Write CP0 register command and store a value into a specific address.

4.1.1 'WCP0' CP0 register write command

This command writes a value into a specific CP0 register:

WCP0 <Reg.select> <Value>

For example, the following command write 0x into CP0 register Config7（register number is 16，select number is 7）：

WCP0 16.7 0x1

After execution of this command, BTB optimization of Jz4730 CPU core is closed.

4.1.2 Write operation to a specific address

These operations write data into a specific address on target board. Users can initialize and configure resources on target CPU through this way. There are 3 kinds of commands according to the data width:

'WM32 <ADDRESS> <VALUE>'

This command write 32 bits data **<Value>** into address specified by **<Address>**.

'WM16 <ADDRESS> <VALUE>'
This command write 16 bits data **<Value>** into address specified by **<Address>**.

'WM8 <ADDRESS> <VALUE>'
This command write 8 bits data **<Value>** into address specified by **<Address>**.

For example, the following command configures Jz4730 CPU'e EMC register:

```
WM32    0xb3010010    0x0fff7700
```

Depending on the hardware features, it is necessary to perform a number of similar commands to complete EMC configuration on a real target board.

4.2 [HOST] segment

This segment defines the IP address for a host machine who provides TFTP service. Some command needs to access a specific file located on a host machine who provides TFTP service. If users didn't explicitly specify the host's IP address in the command line, the command will use the IP address specified in [HOST] segment. The definition of the IP address is defined as following:

```
IP          <IP address of the host machine>
```

The definition of this segment doesn't affect JDI's configuration file itself. JDI's configuration file is accessed through the IP address obtained by *'config'* command when initializing JDI.

An example of this segment is as following:

```
[HOST]
IP          192.168.1.20
```

4.3 [FLASH] segment

This segment specifies NOR FLASH parameters on target board. Before performing program or erase operation, a specific algorithm defined in [FLASH] segment needs to be selected. Users need to choose a specific algorithm according the hardware feature of NOR FLASH on target, and fill the configuration items according to the algorithm.

The current version of JDI supports 3 kinds of algorithm for program or erase operation: AUTO, I28F and AM29. The following sections will describe [FLASH] segment's definition for each algorithm. For some specific NOA FLASH, more than one algorithm is applicable. For example, either I28F or AUTO is applicable to some INTEL flash. In this case, just pick one of them.

4.3.1 [FLASH] segment definition for AUTO algorithm

The following types of NOR FLASH are of auto algorithm:

SST28SF020，SST28SF040
SST39VF010，SST39VF020，SST39VF040
SST39SF010，SST39SF020，SST39SF040
SST39VF1601，SST39VF1602
SST39VF3201，SST39VF3202
SST39VF6401，SST39VF6402
And all NOR FLASH that supports CFI interface.

Two parameters need to be defined for this kind of NOR FLASH:

‘CHIPTYPE type’

This parameter defines the algorithm for the NOR FLASH. The value of **‘type’** is **‘AUTO’** in this case.

‘CHIPSIZE Value’

This parameter defines the size of NOR FLASH on target board. ‘**Value**’ is the number of bytes. For example, 0x00400000 means the size of NOR FLASH is 4M bytes.

The following is an example for a [FLASH] segment definition on a target board for this kind of NOR FLASH:

```
[FLASH]
CHIPTYPE        AUTO
CHIPSIZE        0X00400000
```

4.3.2 [FLASH] segment definition for I28F algorithm

NOR FLASH chips covered by this algorithm have the same program and erase process with Intel’s 28F series NOR FLASH chip. Users need to judge that the NOR FLASH on the target board follows this algorithm according to the FLASH chip’s hardware specification.

Four parameters need to be defined for this kind of NOR FLASH:

‘CHIPTYPE type’

This parameter defines the algorithm for the NOR FLASH. The value of **‘type’** is **‘I28F’** in this case.

‘CHIPSIZE Value’

This parameter defines the size of NOR FLASH on target board. ‘**Value**’ is the number of bytes. For example, 0x00400000 means the size of NOR FLASH is 4M bytes.

‘CHIPWIDTH Value’

This parameter defines the data width of NOR FLASH on target board. ‘**Value**’ can be 8, 16 or 32bits for a specific NOR FLASH. Users need to determine this number according to the NOR FLASH’s hardware specification and its connection on target board.

‘BUSWIDTH Value’

This parameter defines data bus width when accessing the NOR FLASH on target board. On some target board, it is possible to access two 16 bits FLASH simultaneously by connecting them into a 32 bits width data bus. 32 bits data can be accessed each time, while each FLASH’s data width is 16 bits. The value of ‘**CHIPWIDTH’** and ‘**BUSWIDTH’** maybe same or different for a specific target, so they need to be defined separately.

‘**Value**’ can be 8, 16 or 32bits for a specific NOR FLASH. Users need to determine this number according to the NOR FLASH’s hardware specification and its connection on target board.

The following is an example for a [FLASH] segment definition on a target board for this kind of NOR FLASH:

```
[FLASH]
CHIPTYPE        I28F
CHIPSIZE        0X00400000
CHIPWIDTH       16
BUSWIDTH        16
```

4.3.3 [FLASH] segment definition for AM29 algorithm

NOR FLASH chips covered by this algorithm have similar program and erase process with AMD’s AM29 series NOR FLASH chip. The difference is: when issuing a command to FLASH, its command address maybe different and the address value need to be specified in [FLASH] segment. Users need to judge that the NOR FLASH on the target board follows this algorithm according to the FLASH chip’s hardware specification.

Six parameters need to be defined for this kind of NOR FLASH:

‘CHIPTYPE type’

This parameter defines the algorithm for the NOR FLASH. The value of **‘type’** is **‘AM29’** in this case.

‘CHIPSIZE Value’

This parameter is same with I28F FLASH.

‘CHIPWIDTH Value’

This parameter is same with I28F FLASH.

‘BUSWIDTH Value’

This parameter is same with I28F FLASH.

‘SETUPADDR1 Value’

This parameter defines the first address when issuing a command to NOR flash. The address value is different for different type of NOR flash. Users need to determine this value according to NOR flash’s hardware specification. For example, **‘Value’** for AMD’s AM29F040B FLASH is 0x555.

‘SETUPADDR2 Value’

This parameter defines the second address when issuing a command to NOR flash. The address value is different for different type of NOR flash. Users need to determine this value according to NOR flash’s hardware specification. For example, **‘Value’** for AMD’s AM29F040B FLASH is 0x2AA.

For example, [FLASH] segment definition for AMD’s AM29F040B flash is as following:

```
[FLASH]
CHIPTYPE        AM29
CHIPSIZE        0X00400000
CHIPWIDTH       8
BUSWIDTH        8
SETUPADDR1      0x555
SETUPADDR2      0x2AA
```

4.4 [NAND] segment

This segment specifies NAND flash parameters on target board. JDI will use these parameters when operating NAND flash. Operations couldn’t be finished properly if these parameters are not defined correctly. This segment has the following items:

‘BUSWIDTH Value’

This parameter defines the data width of NAND flash on target board. ‘**Value**’ can be 8 or 16 bits according to NAND flash width.

‘ROWCYCLES Value’

NAND flash needs its row address to be written 2 or more times when performing access operations. This parameter defines the number of times that row address of NAND flash that need to be written. Usually, **‘Value’** is 2 or 3.

‘PAGEPERBLOCK Value’

This parameter defines the page number per block.

'PAGESIZE Value'

This parameter defines the page size of NAND flash on target board. **'Value'** is the number in bytes, for example, NAND flash page size can be 512 bytes or 2048 bytes.

'OOB Value'

This parameter indicates whether the binary contains oob data. **'Value'** could be 0(doesn't contain) or 1(contain).

'OOBECC Value'

This parameter indicates whether the oob data of the binary contains ECC data. **'Value'** could be 0(doesn't contain) or 1(contain).

'ECCPOS Value'

Specify the ECC offset inside the oob data. **'Value'** could be 0-[oobsize-1].

'BCHBIT Value'

Specify the hardware BCH algorithm for 4750. **'Value'** could be 4 or 8.

'BADBLOCKPOS Value'

Specify the badblock flag offset inside the oob. **'Value'** could be 0-[oobsize-1].

'BADBLOCKPAGE Value'

Specify the page number of badblock flag inside a block. **'Value'** could be 0 - [PAGEPERBLOCK-1].

'FORCEERASE Value'

This parameter indicates whether the erase operation is applied to bad block when performing such operation. If **'Value'** is 1, erase operation is performed to bad block; if **'Value'** is 0, erase operation is not performed.

'WP_PIN Value'

Specify the write protect pin number, e.g. GPD23=32*3+23=119.

The following example illustrates a [NAND] segment definition:

```
[NAND]
BUSWIDTH        8
ROWCYCLES       3
PAGESIZE        2048
PAGEPERBLOCK    128
OOB             1
OOBECC          0
ECCPOS          28
BCHBIT          4
```

```
BADBLOCKPOS          0
BADBLOCKPAGE         127
FORCEERASE           0
WP_PIN               119
```

4.5 [EEPROM] segment

Access to EEPROM is generally performed through I2C bus on actual target board. A device address value is required for this operation. This section contains only one item for defining this value as following:

‘ADDRESS Value’

‘Value’ is the I2C device address for EEPROM.

An example of this segment definition is as following:

```
[EEPROM]
ADDRESS              7
```

4.6 [SDRAM] segment

This segment defines parameters of SDRAM for use by ‘memtest’ command. This segment has the following items:

‘BUSWIDTH Value’

This parameter defines the data width of SDRAM on target board. ‘**Value**’ can be 16 or 32 bits according to SDRAM width.

‘BANKS Value’

This parameter defines the bank number of SDRAM on target board. ‘**Value**’ can be 2 or 4 bits according to SDRAM specification.

‘ROWADDR Value’

This parameter defines the row address width of SDRAM on target board. ‘**Value**’ can be 11 to 13 bits according to SDRAM specification.

‘COLADDR Value’

This parameter defines the column address width of SDRAM on target board. ‘**Value**’ can be 8 to 12 bits according to SDRAM specification.

‘CASLATENCY Value’

This parameter defines the CAS latency of SDRAM on target board. ‘**Value**’ can be 2 or 3 according to SDRAM specification.

‘ISBUSSHARE Value ’

Whether bus is shared by sdram and nand.

4.7 [GDB] segment

This segment is related to GDB debugging. You must specify the INITFILE field when you want to start a GDB debug session. GDB will execute the init code specified by the INITFILE prior to loading the debuggee to the target memory.

‘INITFILE FILENAME’

‘FILENAME’ is init binary file name. This file must be located at the TFTP directory on the host.

5 Hardware Description

This section describes JDI hardware and covers the following aspects:

- Port description
- LED description
- Jumper for firmware protection
- JTAG signal description.

5.1 JDI Port

There are 5 ports on JDI:

USB port
This port serves no other purpose than power supply.

Ethernet Port
This port provides network service. It is used for data transfer between JDI and the host machine who provides TFTP service.

RS-232Port (PS2)
This port provides a serial interface for JDI operations.

JTAG port
This port provides interface between JDI and target board. Operation of JDI to target board is performed through this port.

5V power supply
This port is used to provide JDI 5V power supply when USB power is not used.

5.2 JDI LED indicator

Red LED on: menas JDI power is on
Green LED (The middle one) on: JDI is ready for accepting command.
Green LED (The side one): JDI is performing operations to JTAG

5.3 JDI Jumper

A jumper J1 is designed to provide protection mechanism for JDI firmware. This jumper is off in general case and firmware couldn't be updated.

Before upgrade operation is performed, users need to open JDI box, turn jumper JP1 on and then

upgrading the firmware. Turn jumper 1 JP1 off after the upgrade operation is finished.

5.4 JDI Signals

JDI ‘s JTAG port signal is compatible with MIPS™ EJTAG standard. The following picture shows the signals: