HiDdrTraining

User Guide

Issue	02
Date	2014-09-05

HiSilicon Technologies Co., Ltd.

Address:	Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China
Website:	http://www.hisilicon.com
Email:	support@hisilicon.com

About This Document

Purpose

This document describes the GUI and usage of the HiDdrTraining.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3521	V100
Hi3531	V100
Hi3520D	V100
Hi3716M	V3XX
Hi3716C	V2XX

Intended Audience

This document is intended for:

- Technical support engineers
- Hardware development engineers

Change History

Updates between document issues are cumulative. Therefore, the latest document issue contains all updates made in previous issues.

Issue 02 (2014-09-05)

This issue is the second official release, which incorporates the following changes:

Chapter 2 GUI and Functions

Description of the reg.bin file is added in section 2.1.3.

Issue 01 (2014-05-23)

This issue is the first official release.

The information of Hi3521 is added.

Issue 00B01 (2014-05-15)

This issue is the first draft release.

Contents

Figures

1 Overview

1.1 Introduction to the HiDdrTraining

Currently the DDR frequency is becoming higher, and the DDR timing window is becoming smaller. The skew change on signal traces is more sensitive, and it is susceptible to the PHY-to-IO trace skew, chip package skew, and board trace skew. The DDR training software is designed to improve the anti-interference capability of the chip DDR, DDR running environment, DDR rate, and product competitive strength.

There are two DDR training modes:

- Boot training

  Boot training automatically starts when the board is powered on. The DDR parameters are dynamically configured so that the DQS/DQ timing parameters are configured to the medium values. The **ddr training** command provided in the software development kit (SDK) is used to view the DQS/DQ parameters. Typically boot training is implemented during the boot process. The training program runs on the flash memory or on-chip static random-access memory (SRAM). The training time period is within 100 ms.

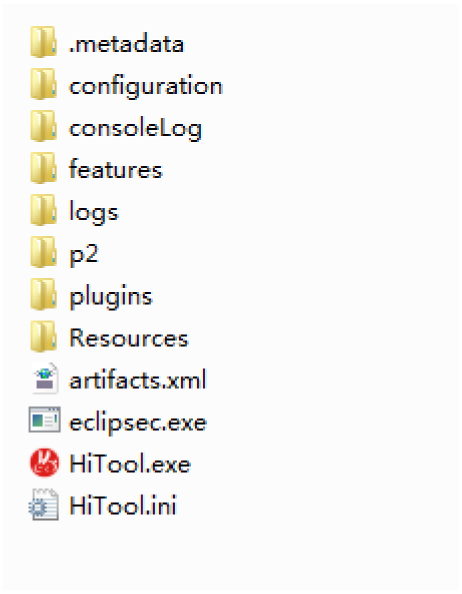
- Service training

  The service training and boot training results are automatically compared. Boot training is not implemented for some chips, therefore the service training results are used as default values of the fastboot table; or the customer does not completely copy the reference design, the default values in the fastboot table need to be reconfigured. The HiDdrTraining provides related auxiliary functions. The following sections describe the functions and GUI of the HiDdrTraining.

1.2 Main GUI

Decompress **HiTool XXX.zip** in the release package (**jre-6u1-windows-i586-p-s.rar** must have been installed). Double-click **HiTool.exe**, as shown in Figure 1-1. Select a required chip, and click **HiDdrTraining**, as shown in Figure 1-2. The main GUI of the HiDdrTraining is displayed, as shown in Figure 1-3. The main GUI contains the menu bar, toolbar, perspective view, HiDdrTraining configuration view, console view, and Trivial File Transfer Protocol (TFTP) command view.

Figure 1-1 HiTool package

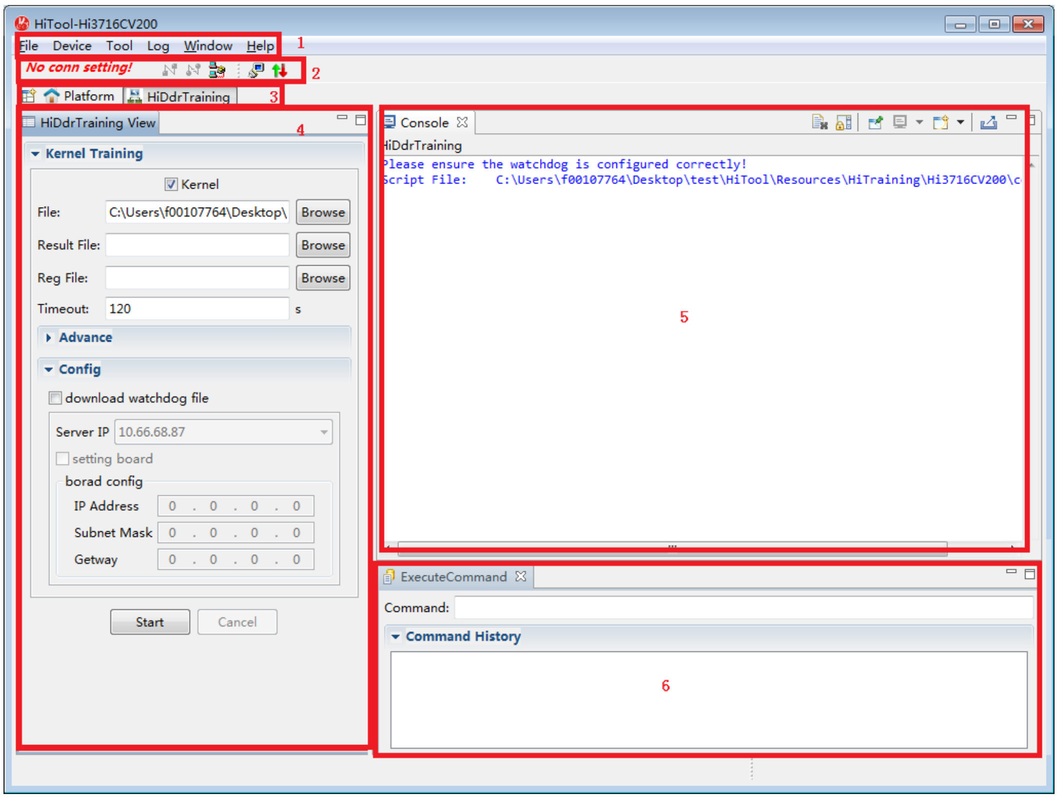


```
.metadata
configuration
consoleLog
features
logs
p2
plugins
Resources
artifacts.xml
eclipsec.exe
HiTool.exe
HiTool.ini
```

Figure 1-2 HiDdrTraining

HiDdrTraining
Support ddr business training etc.

Figure 1-3 Main GUI of the HiDdrTraining



File: C:\Users\f00107764\Desktop\
Result File:
Reg File:
Timeout: 120 s
Server IP 10.66.68.87
IP Address 0 . 0 . 0 . 0
Subnet Mask 0 . 0 . 0 . 0
Getway 0 . 0 . 0 . 0
Command:

```
HiDdrTraining
Please ensure the watchdog is configured correctly!
Script File:    C:\Users\f00107764\Desktop\test\HiTool\Resources\HiTraining\Hi3716CV200\c
```

- Menu bar (area 1)
- Toolbar (area 2)
- Perspective view (area 3)
- HiDdrTraining configuration view (area 4)
- Console (area 5)
- ExecuteCommand view (area 6)

TIP

The **ExecuteCommand** view is not displayed by default. If you need to display it, press **Ctrl+R** or choose **Window** > **Show View** > **ExecuteCommand**.

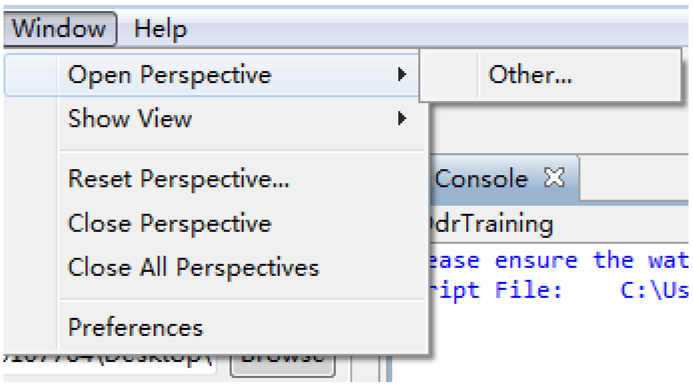
2 GUI and Functions

2.1.1 Menu Bar

To start the HiDdrTraining, you can click the **HiDdrTraining** icon on the startup GUI, or choose the HiDdrTraining perspective view on the menu bar as follows:

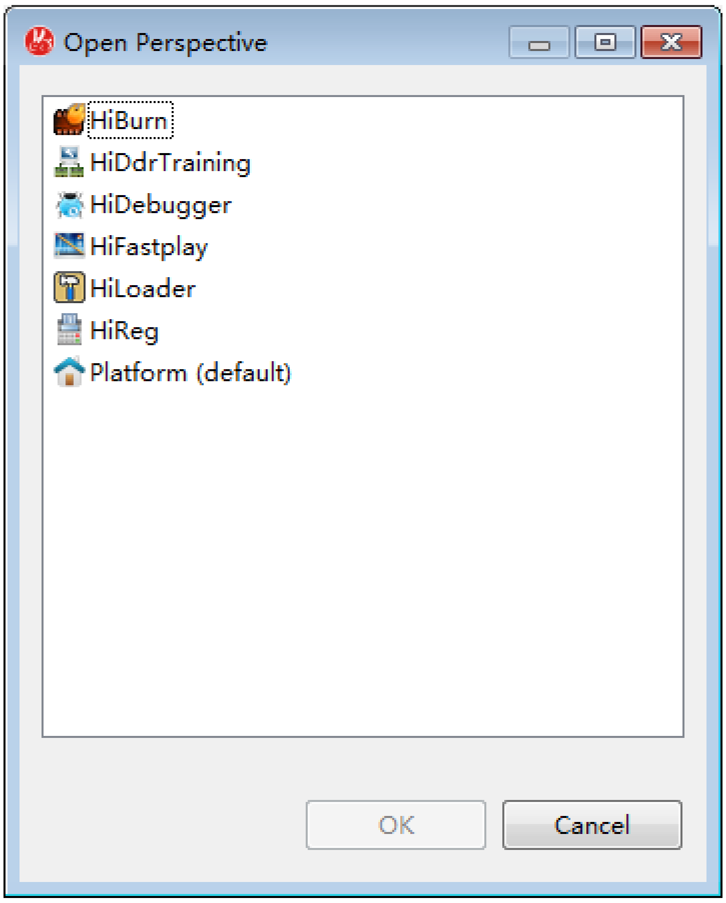
Step 1 Choose **Window** > **Open Perspective** > **Other**, as shown in Figure 2-1.

Figure 2-1 Window menu



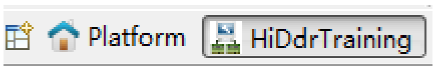
Step 2 Select **HiDdrTraining**, and click **OK**, as shown in Figure 2-2.

Figure 2-2 Open Perspective



Step 3 Click the **HiDdrTraining** icon on the toolbar, as shown in Figure 2-3.

Figure 2-3 HiDdrTraining icon



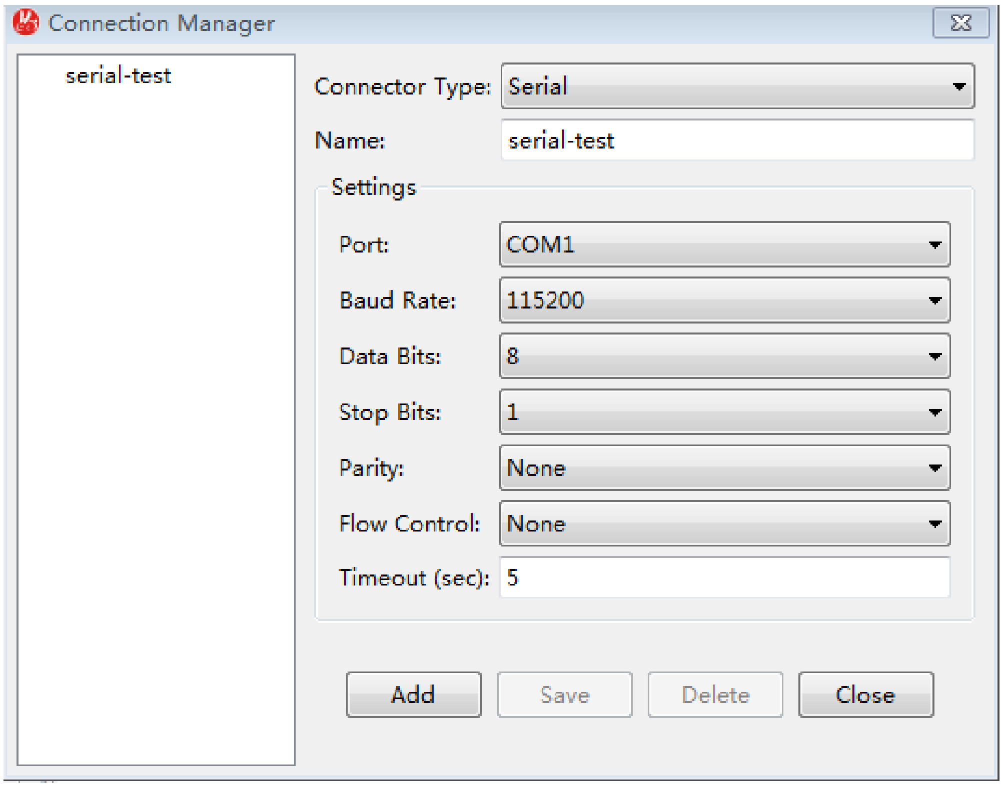
----End

2.1.2 Toolbar

This section describes operations related to connection configuration and management. Currently the HiDdrTraining supports only serial port connections. To create a serial port connection, perform the following steps:

Step 1 Click on the toolbar to open the **Connection Manager** dialog box, as shown in Figure 2-4, in which you can add, save, and delete serial port connections and network connections. The HiDdrTraining supports only the serial port connection.

Figure 2-4 Connection Manager



Connector Type: Serial
Name: serial-test
Settings
Port: COM1
Baud Rate: 115200
Data Bits: 8
Stop Bits: 1
Parity: None
Flow Control: None
Timeout (sec): 5

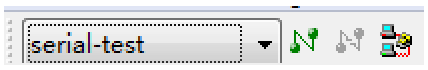
Step 2 Select **Serial** from the **Connector Type** drop-down list, and click **Add**. The added serial connection is listed in the left pane. Then modify the following configurations:

- **Port**: COM1
- **Baud Rate**: 115200
- **Data Bits**: 8
- **Stop Bits**: 1
- **Parity**: None
- **Flow Control**: None
- **Timeout (sec)**: 5

Step 3 Click **Save** to save the configurations.

The saved connection is displayed on the toolbar of the main GUI, as shown in Figure 2-5.

Figure 2-5 Connection displayed on the toolbar



Step 4 Select a serial port connection on the toolbar, and click to connect to the board.

----End

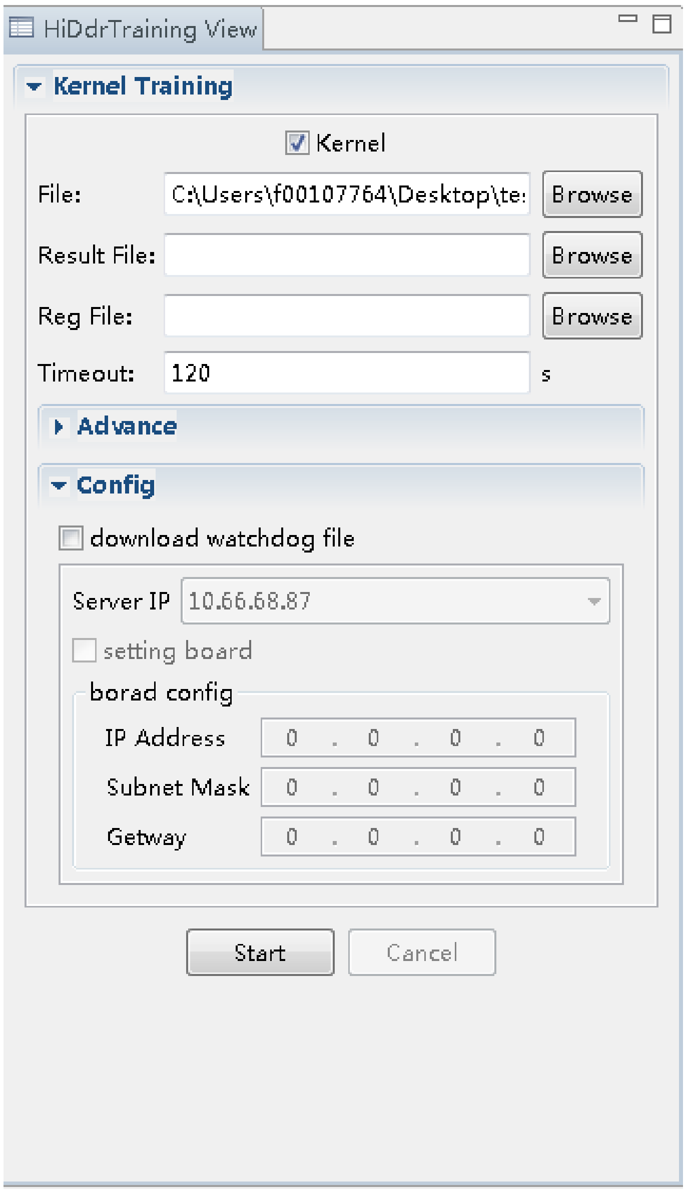
CAUTION

- You are advised to run only one service case on the board. The service case cannot occupy the serial port exclusively; otherwise, data transmission failures may occur.
- The service test cases must maximize the DDR load by enabling multiple modules to access the DDR concurrently.
- You can obtain the recommended DDR test cases from HiSilicon field application engineers (FAEs).

2.1.3 HiDdrTraining View

The HiDdrTraining can automatically execute the DdrTraining script, calculate the optimal values, and integrate the results to the reg file. Figure 2-6 shows the configuration view.

Figure 2-6 HiDdrTraining View


HiDdrTraining View

Kernel Training

☑ Kernel

File: C:\Users\f00107764\Desktop\te:

Result File:

Reg File:

Timeout: 120 s

Advance

Config

☐ download watchdog file

Server IP 10.66.68.87

☐ setting board

borad config

IP Address 0 . 0 . 0 . 0

Subnet Mask 0 . 0 . 0 . 0

Getway 0 . 0 . 0 . 0

Kernel Configuration

After the HiDdrTraining is started, the default configuration files are selected based on the chip. These default files are stored in the local directory of the HiTool (*HiTool root directory*/**Resources/HiTraining**/*current chip name*).

- The script file is **config.xml**.
- The reg file is **reg_info.bin**.
- The result file is **result.txt**.
- The watchdog file is **watchdog.bin**.

Note the following when customizing these files:

- The script file text box is mandatory, and the script file must exist.

- If the **Reg File** text box is not blank, the specified reg file must exist.
- The **Reg File** and **Result File** text boxes can be left blank. If the **Reg File** text box is empty, the optimal values are not saved after kernel training.
- If the **Result File** text box is empty, the HiDdrTraining automatically creates a **result.text** file in the HiTool local directory (*HiTool root directory*/**Resources/HiTraining**/*current chip name*). If the specified file in the **Result File** text box does not exist, the HiDdrTraining automatically creates the file.

NOTE

The reg file is a DDR parameter file in the boot. Take the HiSTBLinux V100R002C0XSPCXXX release package as an example. The **reg.bin** file can be generated in **\source\boot\sysreg** by selecting the corresponding hardware configurations.

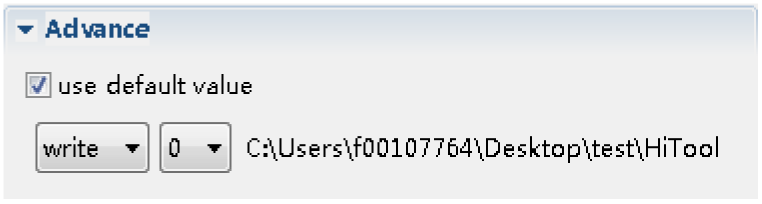
Advanced Settings

If **use default value** in the advanced options is not selected, you can select the script file, result file, and reg file, and perform kernel training of specified default configurations. Naming rules for scripts in advanced options are as follows:

- Read script: **config_read[*digit*].xml**, for example, **config_read0.xml**, **config_read1.xml**
- Write script: **config_write[*digit*].xml**, for example, **config_write0.xml**, **config_write1.xml**

If **use default value** in the advanced options is selected, the specified script files must exist in the HiTool local directory (*HiTool root directory*/**Resources/HiTraining**/*current chip name*). The register optimal values and result file are not saved.

Figure 2-7 Advanced settings


Advance

☑ use default value

write 0 C:\Users\f00107764\Desktop\test\HiTool

Board Configuration

To download the watchdog file, you need to configure the network. If the **watchdog.bin** file exists on the board and is normal, you do not need to reconfigure the network. If the file does not exist on the board, you need to download it over TFTP. Otherwise, the HiDdrTraining test is interrupted.

Exclusive Serial Port Mode

If a large amount of data is displayed over the service serial port, data reception may be affected. In this case, you are advised to use the KO mode for using the serial port exclusively. To use this mode, set the HiTool as follows:

Step 1 Compile the .ko file (**mono_uart.ko**) for using the serial port exclusively. It is compiled by default as a module. The corresponding kernel configurations are as follows:

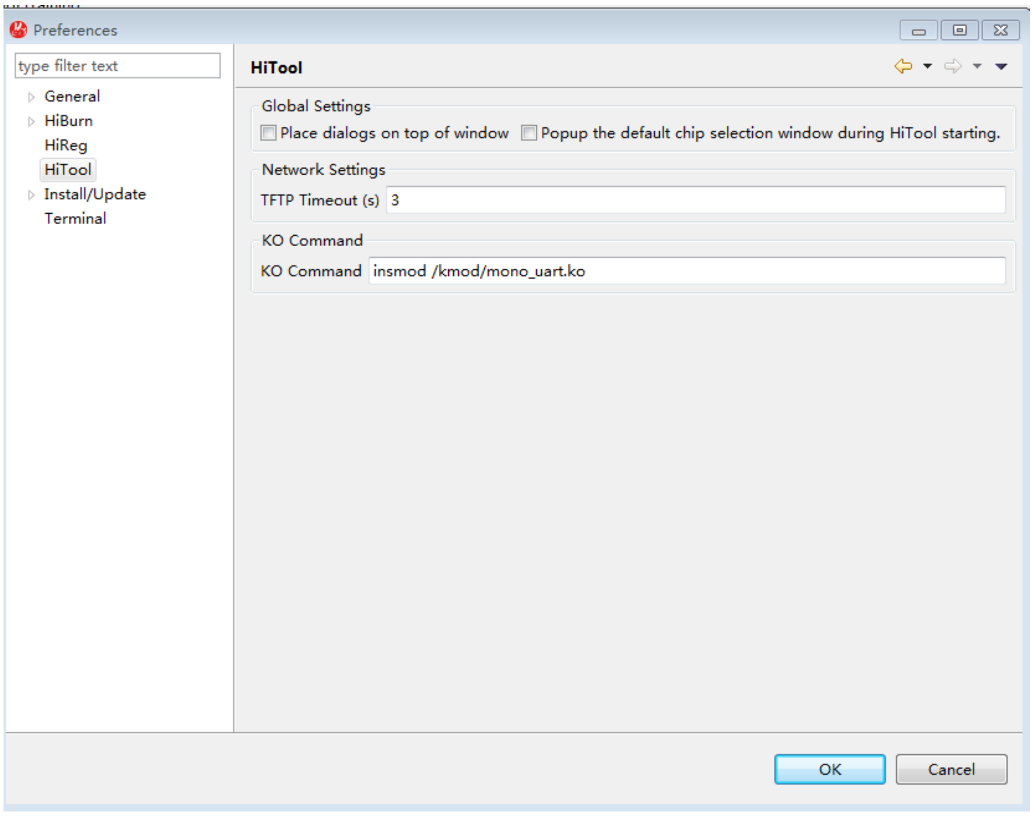
Device Drivers --->

```
Character devices   --->
    Serial drivers   --->
        <M> Mono serial support
```

The compiled file is stored in the **drivers/serial** directory of the kernel source code.

Step 2 Choose **Window** > **Preferences**, and click **HiTool**, as shown in Figure 2-8. Enter the command for loading the .ko module correctly. Note that the file name and path are case sensitive. The HiTool then automatically mounts the configured .ko module before DDR training.

Figure 2-8 Ko command configuration



Global Settings
☐ Place dialogs on top of window ☐ Popup the default chip selection window during HiTool starting.

Network Settings
TFTP Timeout (s) 3

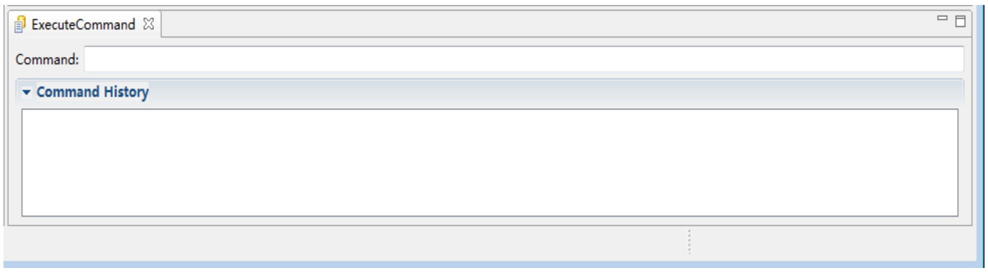
KO Command
KO Command insmod /kmod/mono_uart.ko

----End

2.1.4 ExecuteCommand View

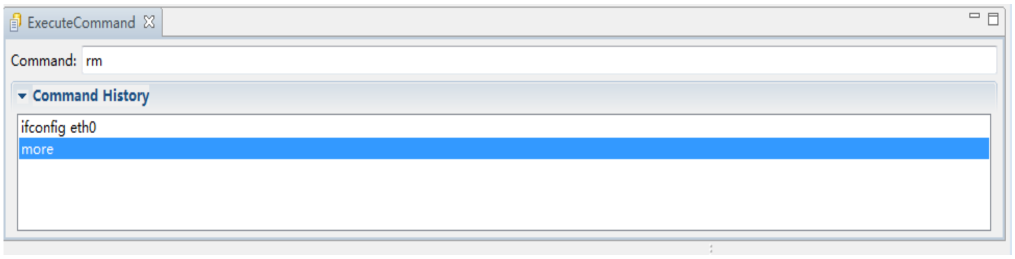
To display the **ExecuteCommand** view, press **Ctrl+R** or choose **Window** > **Show View** > **ExecuteCommand**. Then you can enter an executable Linux command in the **Command** text box, and press **Enter** to execute it. See Figure 2-9.

Figure 2-9 Entering an executable command



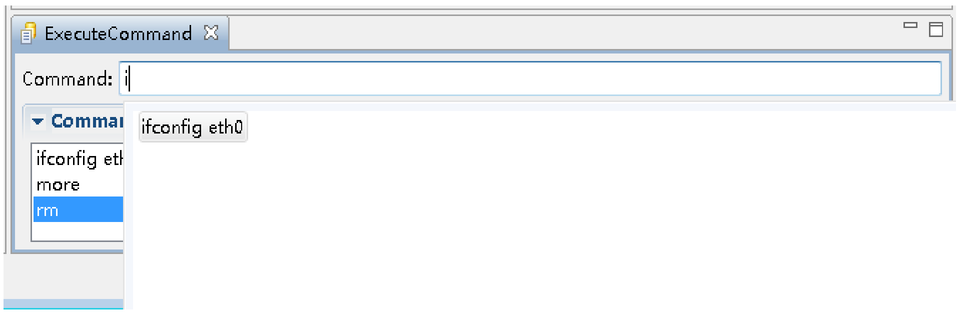
Command History displays entered historical commands, as shown in Figure 2-10.

Figure 2-10 Command History



When you enter a command in the **Command** text box, the system automatically searches for similar commands in the **Command History** list based on the currently entered characters and displays the results (note that the same command in the command history record is displayed only once) in the AutoComplete box, as shown in Figure 2-11.

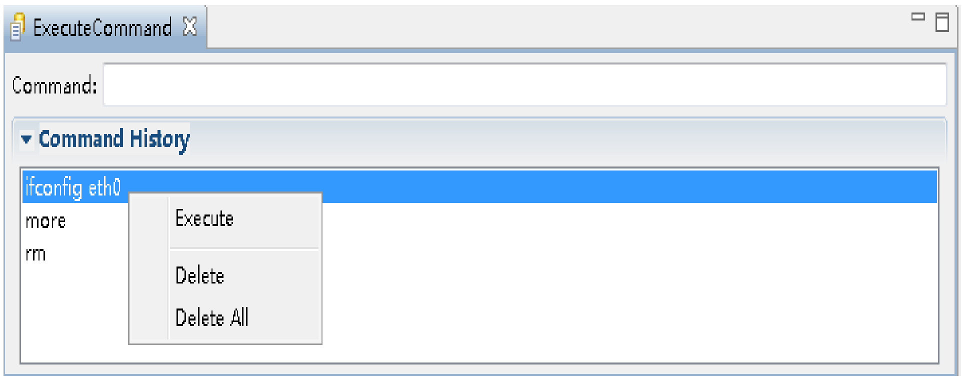
Figure 2-11 AutoComplete box



You can execute and delete commands by using the shortcut menu in **Command History**, as shown in Figure 2-12. You can also double-click a command to execute it. If a command in **Command History** is selected by using the up or down arrow key, the selected command is displayed in the **Command** text box.

- **Execute**: Executes the selected command in the current command history list.
- **Delete**: Deletes the selected command in the current command history list.
- **Delete All**: Deletes all commands in the current command history list.

Figure 2-12 Shortcut menu in the command history list



2.1.5 Console

After the board, network, serial port, and files are configured by following the preceding sections, click **Start**. Then interactions with the board and results are displayed in the console.

3 HiDdrTraining Running Principles and Script

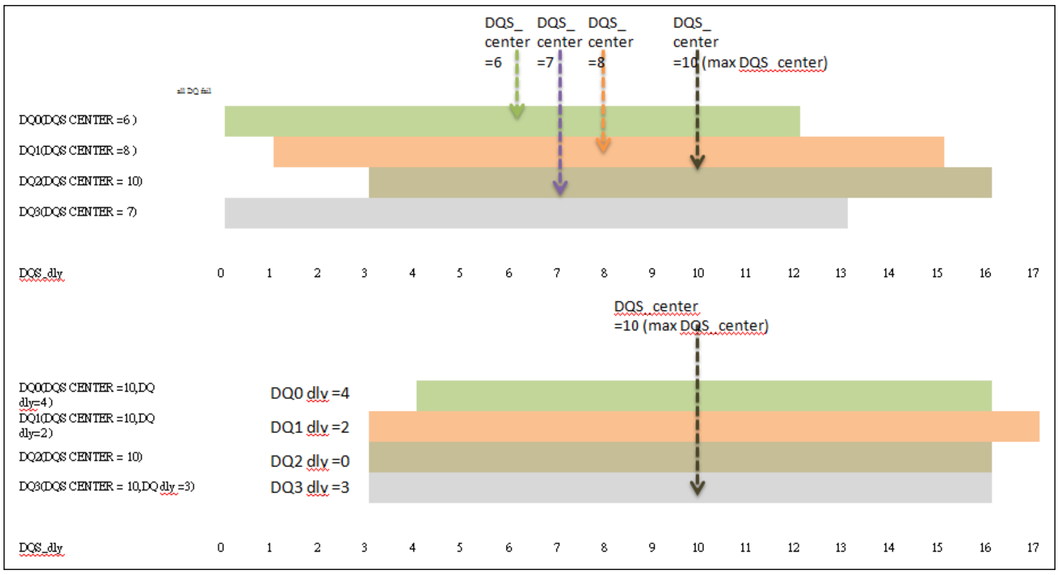
3.1 HiDdrTraining Running Principles

3.1.1 Principles

All the currently used DDR PHYs provide the function of adjusting the DQ or DQS signal delay. This delay is used to compensate unequal DDR traces. Especially on the 2-layer printed circuit board (PCB), the length difference of DQ signal traces is large to facilitate routing. If compensation is not implemented within the chip, the window will be very small.

As shown in Figure 3-1, the DDR timing window is the intersection part of various DQ signals. The intersection range is small if DDR training is not implemented. After DDR training, the timing window is enlarged.

Figure 3-1 Timings before and after DDR training



The principles of adjusting DQ/DQS signals by using the HiDdrTraining are similar to those under fastboot. The difference is that the DDRT or other service module (VEDU) used for adjustment under fastboot is replaced with actual services. In this way, there are much more interferences within the chip and pressure on the DDR, and the obtained results are more accurate.

Take Hi3716C V200 as an example. Training is implemented by byte. Adjustment on DQS signals affects all windows in the byte, and the adjustment range of DQ signals is broader (32-level bit delay line, 25 ps a level). Therefore, the DQS signals are not adjusted, and the DQS signal configurations in the boot table are directly used. Only the bit delay line (BDL) value of the DQ signal is adjusted.

To adjust the BDL value of a DQ signal, perform the following steps:

Step 1 Configure test services (PIP+TDE+GPU+DDRT) on Linux, and enable the services to be automatically started upon power-on. Enable the watchdog, and set the interval of feeding the watchdog to 5s. If the watchdog is not fed, it restarts the board.

Step 2 Read the default BDL value of a DQ signal.

Step 3 Increase the BDL value from the default value, and wait 30s (adjustable).

- Record the BDL as OK if the board is not restarted in 30s.
- Record the BDL as fail if the board is restarted in 30s.

Step 4 Repeat step 3 until the maximum BDL value is obtained.

Step 5 Decrease the BDL value from the default value, and wait 30s.

- Record the BDL as OK if the board is not restarted in 30s.
- Record the BDL as fail if the board is restarted in 30s.

Step 6 Repeat step 5 until the minimum BDL value is obtained.

Step 7 Use the number of BDL levels recorded as OK as the timing window size of the DQ signal, and use the middlemost BDL value as the optimal configuration.

- For Hi3716C V200, the DQ signals support the 32-level BDL, and the DQS BDL does not need to be adjusted.
- For Hi3716M V300, the DQ and DQS signals support the 8-level BDL, and therefore the 15-level BDL of the DQS and DQ combination is used for adjustment.
- For the Hi3531 and Hi3716C V100, the supported BDL levels in the read direction are different from those in the write direction. For Hi3716C V100, the 29-level BDL is supported in the write direction, and the 15-level BDL is supported in the read direction.

Table 3-1 DQ/DQS BDL levels for Hi3716C V100

Write Window Level	Read Window Level	DQS Level	DQ Level
0	-	Disabled	7
1	-	Disabled	6
2	-	Disabled	5
3	-	Disabled	4
4	-	Disabled	3
5	-	Disabled	2

Write Window Level	Read Window Level	DQS Level	DQ Level
6	-	Disabled	1
7	0	0	7
8	1	0	6
9	2	0	5
10	3	0	4
11	4	0	3
12	5	0	2
13	6	0	1
14	7	0	0
15	8	1	0
16	9	2	0
17	10	3	0
18	11	4	0
19	12	5	0
20	13	6	0
21	14	7	0
22	-	1	Disabled
23	-	2	Disabled
24	-	3	Disabled
25	-	4	Disabled
26	-	5	Disabled
27	-	6	Disabled
28	-	7	Disabled

Table 3-2 DQ/DQS BDL levels for Hi3716M V300

Write Window Level	Read Window Level	DQS Level	DQ Level
0	0	7	0
1	1	6	0
2	2	5	0
3	3	4	0

Write Window Level	Read Window Level	DQS Level	DQ Level
4	4	3	0
5	5	2	0
6	6	1	0
7	7	0	1
8	8	0	2
9	9	0	3
10	10	0	4
11	11	0	5
12	12	0	6
13	13	0	7

Table 3-3 DQ BDL levels for Hi3716C V200

Write Window Level	Read Window Level	DQ Level
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16

Write Window Level	Read Window Level	DQ Level
17	17	17
18	18	18
19	19	19
20	20	20
21	21	21
22	22	22
23	23	23
24	24	24
25	25	25
26	26	26
27	27	27
28	28	28
29	29	29
30	30	30
31	31	31

----End

3.1.2 Operation Process

Currently all chips contain scripts that can be directly used. If you need to update the script, you can develop the script based on the detailed process. The operation process is as follows:

NOTE

- The register group that defines only one register is called Single.
- The register group that defines multiple registers is called Multiple.

Step 1 Read the configuration file.

Step 2 Enable the watchdog.

Step 3 Load and run the **regtool** command in **KO Command** in the **Preferences** dialog box.

Step 4 Read registers to be traversed, and save the read values as initial values.

Step 5 Process loops (corresponding to a byte), and then process each loop (corresponding to the DQ signal) in the loops.

Step 6 Process loop_level_begin.

Check whether **initvalue** of the current register group is **true**. If yes, display and save the initial values of all registers in the current loop.

- Process Single.

1. When the default value is **−1**:

   If **minvalue** (minimum DQ BDL level) is configured:

   - Update the register value (DQ BDL level) circularly, write the value back to the device, and check whether the device is restarted.
   - Update rule: Value after update = Value before update – 1 (the initial value of the current value is determined based on the configured bit range and the value read from the board by the register)
   - Condition for updating register values circularly: Current value ≥ **minvalue**
   - Update the register values circularly, and check whether the device is restarted. The wait time can be specified in the **Timeout** text box of the GUI, and the current execution result is output.
   - After a cycle, all registers in the current registergroup are reset.

   If **maxvalue** (maximum DQ BDL level) is configured:

   - Update the register value circularly, write the value back to the device, and check whether the device is restarted.
   - Update rule: Value after update = Value before update + 1 (the initial value of the current value is determined based on the configured bit range and the value read from the board by the register)
   - Condition for updating register values circularly: Current value ≤ **maxvalue**
   - Update the register values circularly, and check whether the device is restarted. The wait time can be specified in the **Timeout** text box of the GUI, and the current execution result is output.
   - After a cycle, all registers in the current registergroup are reset.

2. When the default value is not **−1**:

   Write the default value to the device, and check whether the device is restarted. The wait time can be specified in the **Timeout** text box of the GUI, and the current execution result is output. After the default values are successfully written to the device, the register in the current register group is reset after the values of registers in all register groups of the same group are executed.

- Process Multiple.

1. If **minvalue** is configured:

   - Update the register value circularly, write the value back to the device, and check whether the device is restarted.
   - Update rule: Value after update = Value before update – 1 (The initial value of the current value is determined based on the configured bit range and the value read from the board by the register. The current value of each register is independent.)
   - Condition for updating register values circularly: Minimum value of the current register value > **minvalue**
   - Update the register values circularly, and check whether the device is restarted. The wait time can be specified in the **Timeout** text box of the GUI, and the current execution result is output.
   - After a cycle, all registers in the current registergroup are reset.

2. If **maxvalue** is configured:

   - Update the register value circularly, write the value back to the device, and check whether the device is restarted.

- Update rule: Value after update = Value before update + 1 (The initial value of the current value is determined based on the configured bit range and the value read from the board by the register. The current value of each register is independent.)
- Condition for updating register values circularly: Maximum value of the current register value < **maxvalue**
- Update the register values circularly, and check whether the device is restarted. The wait time can be specified in the **Timeout** text box of the GUI, and the current execution result is output.
- After a cycle, all registers in the current registergroup are reset.

NOTE

If the boot does not restart automatically during execution, a message is displayed asking you to restart it manually.

Step 7 After loop_level in all loops is executed, search for reg files and combine the files, and process, display, and save the optimal values. For details, see the following:

- Rules for obtaining the optimal values

  NOTE

  You need to obtain one optimal value for each loop based on the execution results of each loop.

  - If there is no fail in the execution results, use the middlemost value in the result set as the optimal value.
  - If there is a fail in the execution results, compare the number of results between the fail result and the start with that between the fail result and the end. Obtain the middlemost value in the part with more results as the optimal value.
  - If there are multiple fail results, compare the number of results between the start and the nearest fail result, that between the end and the nearest fail result, and that between every two adjacent fail results. Obtain the middlemost value in the part with the most results as the optimal value.
- Processing results
  - If the training type is **byteTraining**, the execution result and optimal value of each loop are displayed on the console.
  - If the training type is **bitTraining**, the optimal value of each loop is processed based on the script configuration. The processing procedure is as follows:

    If the maximum value register is specified in the loops:

    - ➢ Obtain the maximum optimal value of the specified register in all loops.
    - ➢ Obtain the key value of the register for each loop.
    - ➢ Calculate the optimal value portfolio of the loops. The optimal value portfolio contains the key optimal value of each loop. The key optimal value is calculated as follows: Key optimal value of each loop = Maximum value – Optimal value of the register specified by loops in each loop + Key value of each loop.
    - ➢ Calculate the DM value. Add up the key optimal value of each loop and obtain the round-off average value (integer).

    If the maximum value register is not specified in the loops:

    - ➢ Obtain the key value of the register for each loop.
    - ➢ Calculate the optimal value portfolio of the loops. The optimal value portfolio contains the key optimal value of each loop. The key optimal value of each loop equals the key value of each loop.
    - ➢ Calculate the DM value. Add up the key optimal value of each loop and obtain the round-off average value (integer).

Then the bit training execution results are displayed, including the optimal value portfolio of each loops node and the DM values.

- Result format

NOTE

The following are only references for the result format. The executed data varies according to the actual scenario. dqs0, dq0, dq1, dq2, dq3, dq4, dq5, dq6, and dq7 are register names, which must be defined in the **description** attribute of each loop.

The **description** information is displayed after each loop is executed. Take the single loop (the default dqs is dqs3) as an example.

Table 3-4 Single loop, dqs3

dqs0	dq0	dq1	dq2	dq3	dq4	dq5	dq6	dq7	Result
7	0	0	0	0	2	2	2	1	Fail
6	0	0	0	0	2	2	2	1	OK
5	0	0	0	0	2	2	2	1	OK
4	0	0	0	0	2	2	2	1	OK

Table 3-5 Multiple loop

dqs0	dq0	dq1	dq2	dq3	dq4	dq5	dq6	dq7	Result
3	1	1	1	1	3	3	3	2	OK
3	2	2	2	2	4	4	4	3	OK
3	3	3	3	3	5	5	5	4	Fail

The loop ends at the first fail result. The middle row (row 4) is used as the optimal values (0, 0, 0, 0, 2, 2, 2, 1), and the optimal and default values are displayed, as shown in Table 3-6.

Table 3-6 Output default and optimal values

Value Type	dqs0	dq0	dq1	dq2	dq3	dq4	dq5	dq6	dq7
Default	3	0	0	0	0	2	2	2	1
Optimal	4	0	0	0	0	2	2	2	1

When the training type is bit training, the following content is also displayed. The maximum values are not displayed if there is no maximum value.

- **MaxRegister**: name of the register specified in the loops
- **MaxValue:** maximum value of the register specified in the loops

Table 3-7 Valid value portfolio (maximum value and key register values of each loop)

Loop Nme	Max	Key Value 1	Key Value 2
Loop1	0	0	
Loop2	0		0

Table 3-8 Optimal value portfolio of the loops

Value Type	Max	Key Value 1	Key Value 2
Optimal	0	0	0

DM: 0

- Search and combination

  Search for the same addresses in the reg file, combine data in the same address, and update the data after combination.

  Save the results.

CAUTION

If the training operation fails or is canceled, the results are not saved to the result file.

After the training operation is executed successfully, the optimal results are saved to **reg.bin**. This file contains the following structures. If both **rddqs0** and **rddqs1** need to be set to **4** (address="0x10100b4c", start="0", end="2", name="rdqs1", address="0x10100b4c", start="3", end="5"), the value of the register with the address 0x10100b4c must be changed to **0x24**.

```
struct regentry {
unsigned int reg_addr;  ---- register address
unsigned int value; -------register value
unsigned int delay;
unsigned int attr;
};
```

Table 3-9 Reg file format

Struct	Struct	Struct	...	Struct
regentry	regentry	regentry	...	Struct

----End

3.2 Script File Format

3.2.1 Relationship Between Nodes

Nodes are described as follows:

- There are only one training node and one commands node.
- The training node is the root node, and it contains the commands node and multiple loops nodes.
- The commands node can contain multiple command nodes.
- A loops node contains multiple loop nodes, and a loop node contains multiple group nodes.
- A group node contains multiple registergroup nodes.
- A registergroup node contains multiple register nodes.

3.2.2 Node Attributes

The node attributes are described as follows:

- The training node contains the **mode** attribute, which defines the training mode (byte or bit).

  If the training node does not contain the **mode** attribute, the default mode type is byte.
- The command node contains the **value** attribute, which defines the command character string to be executed. This attribute cannot be empty. If a command contains a symbol, the HTML escape character of the symbol must be used.
- The loops node contains the **max** attribute, which defines the name of the maximum register for the loops node. Note the following:
  - When **mode** is **bit**, the loops node can contain the **max** attribute or not. If the node contains the **max** attribute, the value cannot be empty. The **max** attribute affects the processing on bit training results.

CAUTION

The maximum register defined in the **max** attribute must be defined in each loop in the current loops node.

  - When **mode** is **byte**, the **max** attribute can be defined or not defined because the attribute works only in bit mode.
- The loop node contains the **key** and **description** attributes, which cannot be empty.
  - The **key** attribute specifies the key register for the current loop.
  - The **description** attribute specifies all registers in the current loop node. The names of all registers in each loop node must be defined in the **description** attribute of the loop node; otherwise, the registers are invalid.
  - The register names in the **description** attribute are separated by commas. The first name is ignored, and the name cannot be duplicate. For example, the first one **read** is ignored, and the actual registers are **rdqs0**, **rdq0**, **rdq1**, **rdq2**, **rdq3**, **rdq4**, **rdq5**, **rdq6**, and **rdq7**.

Note the following:

- When **mode** is **bit**, the **key** attribute must be defined, and the **key** attribute value must be defined in the register list in any registergroup node of the current loop.
- When **mode** is **byte**, the **key** attribute can be defined or not defined because the attribute works only in bit mode.

- The group node contains no attribute value.
- The registergroup node contains the **inivalue**, **sequence**, **maxvalue**, **minvalue**, and **defaultbvalue** attributes.
  - The **inivalue** attribute defines whether the current registergroup node outputs only the initial value. The attribute value can be **true** or **false** (default). If it is **true**, other attributes of the registergroup node are invalid and not verified during loading; otherwise, other attributes are loaded and verified.
  - The **sequence** attribute defines the output sequence of the registergroup results. The attribute value can be **true** or **false**, and it cannot be empty. If it is **true**, results are output in order; otherwise, results are output in reversed order.
  - **maxvalue** specifies the maximum value that can be configured for all registers in the current node.
  - **minvalue** specifies the minimum value that can be configured for all registers in the current node.
  - The **defaultvalue** attribute defines the default values of all registers in the current node.

  Note the following:

  - The **maxvalue** and **minvalue** attributes of a registergroup node cannot be specified at the same time.
  - If **inivalue** is **false**, one of **maxvalue**, **minvalue**, and **defaultvalue** must be defined.
  - The register node contains the **name**, **address**, **start**, and **end** attributes. The **name** attribute defines the current register name, the **address** attribute defines the register address, the **start** attribute defines the start position of the register, and the **end** attribute defines the end position of the register.

3.2.3 Notes

Note the following when writing the script:

- The script must contain the XML declarations, and the encoding format must be UTF-8.
- All nodes in the script appear in pairs, and the node names are lowercase characters. The script must comply with the XML standard.
- In a loop node, if the **address**, **start**, and **end** attributes of two registers are the same, their **name** attributes must also be the same. If one of the **address**, **start**, and **end** attributes of two registers is different, the **name** attributes must be different.
- The suffix .xml is recommended for the script file. You can use the provided XML file as reference.

4 FAQs

4.1 What Should I Pay Attention to During Kernel Training?

Problem Description

What should I pay attention to during kernel training?

Solution

Note the following:

- The file **watchdog.bin** is required to automatically restart the board. Therefore, ensure that the file exists on the board, it is normal, and services can run automatically.
- The HiTool interacts with the board over the serial port. Therefore, ensure that the serial port is not exclusively occupied by a service. Load **mono_uart.ko**. This file is compiled as a module by default. The corresponding kernel configurations are as follows:

```
Device Drivers    --->
     Character devices    --->
          Serial drivers    --->
               <M> Mono serial support
```

  The compiled file is stored in the **drivers/serial** directory of the kernel source code.

4.2 What Do I Do If the Watchdog Script Fails to Be Downloaded?

Problem Description

A message similar to "Nothing received from the device for the command: tftp -r watchdog.bin X.X.X.X –g Remote no response, stop executing training." is displayed, indicating that the watchdog script fails to be downloaded. What are the possible causes?

Solution

- The file **watchdog.bin** does not exist in the HiTool local directory (**HiTool/Resources/HiTraining/***current chip*), or the board IP address, subnet mask, and network gateway conflict.
- If "Failed to reset init value Remote no response" is displayed, the watchdog hardware is not started.

4.3 What Do I Do If "Address xx not found, the value will be xx" Is Displayed?

Problem Description

The error information "Address xx not found, the value will be xx" is displayed.

Solution

This issue occurs because there is no register in **reg_info.bin** that matches the register address in the script file. The combination fails. Ensure that the **reg_info.bin** file matches the script file.

4.4 What Do I Do If the File System Is Damaged During Kernel Training and Cannot Be Restored After the System Is Restarted?

Problem Description

The file system is damaged during kernel training, and it cannot be restored after the system is restarted.

Cause Analysis

This issue usually occurs in the yaffs file system of the NAND flash, and the occurrence probability is high if there are dual cores. The yaffs file system periodically updates data in the DDR to the flash memory, and the DDR is unstable during kernel training. Therefore, when the dual-core concurrent system is used, the file system is easily damaged after being written.

Solution

Remount the file system as read-only by running commands, for example, **mount / -o remount**, **ro -o noatime**. You can also use the SPI/jffs2 file system.