

Official
Release

Linux Development User Manual

Document
Number

UM218-22

Date Issued

2023-12-04

Copyright © 2020-2023 Andes Technology Corporation.
All rights reserved.



Copyright Notice

Copyright © 2020–2023 Andes Technology Corporation. All rights reserved.

AndesCore™, AndeSight™, AndeShape™, AndESLive™, AndeSoft™, AndeStar™, Andes Custom Extension™, CoDense™, StackSafe™, QuickNap™, AndesClarity™, AndeSim™, AndeSysC™, AndesAIRE™, AnDLA™, NNPilot™, Andes-Embedded™ and Driving Innovations™ are trademarks owned by Andes Technology Corporation. All other trademarks used herein are the property of their respective owners.

This document contains confidential information pertaining to Andes Technology Corporation. Use of this copyright notice is precautionary and does not imply publication or disclosure. Neither the whole nor part of the information contained herein may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the written permission of Andes Technology Corporation.

The product described herein is subject to continuous development and improvement. Thus, all information herein is provided by Andes in good faith but without warranties. This document is intended only to assist the reader in the use of the product. Andes Technology Corporation shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

Contact Information

Should you have any problems with the information contained herein, you may contact Andes Technology Corporation through

- email – support@andestech.com
- Website – <https://es.andestech.com/eservice/>

Please include the following information in your inquiries:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem

General suggestions for improvements are welcome.

Revision History

Rev.	Revision Date	Revised Content
2.2	2023/12/04	<ol style="list-style-type: none"> Updated the verified Ubuntu version of the Linux package. (Chapter 1) Updated the procedure to initialize the linux_devel_pack project from the Andes GitLab repository. (Section 2.2) Explained the --toolchain_path option for using Prepare_rootfs.sh; changed to specify --cpu=45 for 60-series cores when using Prepare_rootfs.sh. (Section 3.2.1) Modified descriptions about how to copy libraries and binary tools for a manual setup of root file system. (Section 3.2.2) Updated the code snippet in the build script build_features.sh and the patch file force_color.patch; noted to use -mtune=andes-45-series to build perf for 60-series cores. (Section 3.3.3) Updated the menuconfig path to enable SPI feature in the Linux kernel. (Section 3.3.4) Updated the path to the DMA driver. (Section 4.2) Updated the path to the DTS files in the Linux package. (Section 4.3) Clarified the menuconfig path to the PMA configuration for Andes cores. (Section 9.1.2.1) Revised descriptions as cache is now enabled in U-Boot by default. (Section 9.1.2.2) Updated the reference function for applying the API "pmp_set" to OpenSBI. (Section 9.1.4) Updated the menuconfig path to enable the SMU support in the Linux kernel; updated the location of sleep.s and that of atcsmu.c; added the clock frequency setting to the wdt node. (Section 9.2.1) Added a section about GPIO. (Section 9.2.4)
2.1	2023/07/28	<ol style="list-style-type: none"> Updated the document template to V15. Upgraded the kernel version to v6.1 and updated the paths in this document.

Rev.	Revision Date	Revised Content
		<p>3. Noted to specify --cpu=25 in Prepare_rootfs.sh if using 27-series cores. Modified the rootfs build script to support 60-series cores (Section 3.2.1)</p> <p>4. Added descriptions about manual rootfs setup for 27 or 60-series cores. (Section 3.2.2)</p> <p>5. Modified the build procedure of Perf and deleted the description of using Perf to count machine mode events. (Section 3.3.3)</p> <p>6. Added how to have a modified DTB with a legacy DMA driver to run MPlayer. (Section 3.4.2)</p> <p>7. Modified the description of how to generate a kernel configuration file. (Section 4.1)</p> <p>8. Updated DTS naming rules. (Section 4.3)</p> <p>9. Changed the build path of OpenSBI images (Section 5.1, 5.2.1, 5.2.2 and 9.3.2)</p> <p>10. Changed the names of U-Boot SPL default configuration files for Normal Boot. (Section 5.2.1)</p> <p>11. Changed "SPL/TPL" in U-Boot menuconfig to "SPL configuration options" and added how to hide OpenSBI firmware information during U-Boot SPL compilation; added a note that ITB files need to be copied to the root of SD card for being booted from MMC (Section 5.2.1 and 5.2.2)</p> <p>12. Updated the address where u-boot.itb will be loaded for Normal Boot (Section 6.1.1)</p> <p>13. Changed the names of options associated with SPL memory layouts in U-Boot SPL defconfig files for Fast Boot. (Section 6.2)</p> <p>14. Updated Andes GitHub link to obtain Andes Buildroot default configuration file and OpenEmbedded layer. (Section 8.2)</p> <p>15. Changed the required option for HPM support in Linux. (Section 9.1.1.1)</p>

Rev.	Revision Date	Revised Content
		<p>16. Changed the file containing HPM configs. Removed the description about editing ae350_early_init for OpenSBI when HPM feature is not configured in the target system. (Section 9.1.1.2)</p> <p>17. Changed the DTS file for the platform AE350_AX25MP_AX25 (Section 9.1.7)</p> <p>18. Removed the note that suspend/resume functions are not implemented in Andes PWM driver. (Section 9.2.1)</p>
2.0	2023/03/27	<p>1. Added the export of OpenSSL installation path to LIBRARY_PATH for installing OpenSSL on CentOS 7. (Section 5.2)</p>
1.9	2022/12/28	<p>1. Added that users need to use absolute paths during rootfs setup. (Section 3.2)</p> <p>2. Adjusted commands to copy library and binary tools during manual root file system setup (Section 3.2.2)</p> <p>3. Updated the destination to which a strace binary will be generated. (Section 3.3.2)</p> <p>4. Added wildcard character * for filenames of shared objects in \$TOP/sysroot_tmp/lib; added -march option to build_features.sh. (Section 3.3.3)</p> <p>5. Added how to enable the SPI support for a kernel being built. (Section 3.3.4)</p> <p>6. Explained the representation of environment variables “XCFLAGS” and “XLFLAGS” for Dhrystone, Whetstone and CoreMark applications. (Section 3.4.4~3.4.6)</p> <p>7. Noted that DTC binary is only available after Linux kernel is built. (Section 4.3)</p> <p>8. Added that a running GDB must be halted first before it is used to load a kernel image; noted that users must terminate the ICEman and GDB used to load Linux kernel if they want to debug with gdbserver afterwards. (Section 6.1.2.1 and 6.2.1)</p> <p>9. Added that ICEman connected to the target board must be terminated first before execution of gdbserver. (Section 7.1)</p>

Rev.	Revision Date	Revised Content
		10. Updated the download link of the prebuilt Ubuntu disk image for Linux distribution. (Section 8.1.3)
1.8	2022/09/16	<p>1. Removed the support of OpenSBI in payload mode and jump mode. (Chapter 1, Section 5.1, Chapter 6, Section 10.2)</p> <p>2. Provided the link to Andes Gitlab Repository. (Section 2.2)</p> <p>3. Modified some commands for building gdbserver. (Section 3.3.1)</p> <p>4. Modified the description of MPlayer application due to the change of default enabled DMA driver. (Section 3.4.2)</p> <p>5. Added explanations for the core in the DTS naming and noted how to retrieve a built-in DTB being used. (Section 4.3)</p> <p>6. Added how to install OpenSSL 1.1.1q and set environment variable for building U-Boot on CentOS 7. (Section 5.2)</p> <p>7. Added copying u-boot.itb and DTB to an SD card for booting U-Boot SPL from flash using sf command. (Section 6.1.1.2)</p> <p>8. Supported a Fast Boot process. (Section 5.2, 6.2 and 9.3)</p> <p>9. Added descriptions of Linux distributions and customization with Buildroot or Yocto. (Chapter 8)</p> <p>10. Added a section about how to build and boot an AMP Linux kernel. (Section 9.1.7)</p> <p>11. Added how to ensure the DTB in-use contains a DMA node with DMAEngine and updated the procedure to test the DMA driver with DMAEngine for its memcpy capability. (Section 9.2.3)</p>
1.7	2022/02/14	<p>1. Added descriptions of PMP, DMA coherence and MSB mechanism. (Section 9.1.4, 9.1.5 and 9.1.6)</p> <p>2. Added how to resolve unusual physical addresses on peripheral device bring-up. (Section 10.3)</p>
1.6	2022/01/14	<p>1. Added a note about shell script execution and changed to use the Bash shell to execute build scripts for root file system and MTD utility. (Chapter 1, Section 3.2.1 and 3.3.4)</p> <p>2. Added a cpu option for running Prepare_rootfs.sh. (Section 3.2.1)</p> <p>3. Updated the sysroot path. (Section 3.2.2 and 7.2)</p>

Rev.	Revision Date	Revised Content
		<p>4. Modified S-lang descriptions by updating the download link, code snippet for force_color.patch and adding CFLAGS and LDFLAGS to its build commands. (Section 3.3.3)</p> <p>5. Noted that Mplayer is only supported when the default DMA driver is enabled. (Section 3.4.2)</p> <p>6. Added DMA and SD 2.0 device drivers that support DMAEngine. (Section 4.2)</p> <p>7. Removed the ICEman option “-Z v5” for target connection. (Chapter 6)</p> <p>8. Added how to boot Linux kernel via GDB with OpenSBI in Dynamic mode. (Section 6.1.2.1)</p> <p>9. Added application notes for Hardware Performance Monitor. (Section 9.1.1)</p> <p>10. Added a section about huge pages. (Section 9.1.3)</p> <p>11. Updated SMU suspend descriptions by supporting the feature on Andes 45-series CPUs and Mac driver, providing solutions for the breakpoint problem after wakeup, noting that local interrupt must be disabled before the sleep and a switch implementation of CSR_xIE is required for having the HPM support in sleep mode. Added preconditions for SMU reboot feature. Updated CPU hotplug descriptions for supporting suspend on SMU. (Section 9.2.1)</p> <p>12. Added a section about dmatest for testing the DMA driver with DMAEngine. (Section 9.2.3)</p> <p>13. Detailed the boot process. (Section 9.3)</p> <p>14. Listed information required for diagnosing kernel boot or crash problems. (Chapter 10)</p> <p>15. Removed descriptions of Linux distributions.</p>
1.5	2021/06/30	<p>1. Added a paragraph about Andes prebuilt Linux image for direct evaluation of Andes platforms. (Chapter 1)</p> <p>2. Explained that a script “fixup.sh” is needed to resolve errors during the build of elfutils. (Section 3.3.3)</p>

Rev.	Revision Date	Revised Content
		<ol style="list-style-type: none"> 3. Added three Linux demo applications “Dhrystone”, “Whetstone” and “CoreMark”. (Section 3.4.4 to 3.4.6) 4. Simplified the Linux kernel boot process with the built-in DTB. (Chapter 4, Section 4.3 and Chapter 6) 5. Added a step to copy u-boot.itb to the root directory of an SD card. (Section 6.1.1.2) 6. Added that U-Boot SPL can also be burnt using SPI_burn. (Section 6.1 and 6.1.1.2) 7. Noted that SMU is not supported on AndesCore 45 series and there is a compatibility issue between SMU and SD card on VCU118 board. Explained why Andes MAC and PWM drivers don't work properly after waking up from light/deep sleep. Added how to set wakeup events if using a built-in DTB. (Section 9.2.1) 8. Removed descriptions of HIGHMEM support on RV32 kernels.
1.4	2020/12/31	<ol style="list-style-type: none"> 1. Changed the document template to V14. 2. Upgraded the kernel version to v5.4 and updated the package paths in this document. 3. Added how to load binaries to the kernel at runtime (Section 3.1.2) 4. Replaced the build script Prepare_rootfs_multilib.sh with Prepare_rootfs.sh. (Section 3.2.1) 5. Updated the process of building perf and added perf usage notes (Section 3.3.3) 6. Changed BSP User Manual to AndeSight BSP User Manual. (Section 3.4) 7. Added descriptions of frame buffer test (Section 3.4.3) 8. Added a step to generate required files for kernel modules during the kernel build process. (Section 4.1) 9. Added SPI and PWM drivers. (Section 4.2) 10. Replaced BBL with the newly-supported OpenSBI and enhanced U-Boot with U-Boot SPL. (Chapter 1 and 5) 11. Updated kernel boot procedures with the use of OpenSBI and U-Boot

Rev.	Revision Date	Revised Content
		12. Added how to run Fedora or Debian distribution. 13. Added descriptions of the CPU hotplug feature (Section 9.2.1) 14. Removed the section about ELF attribute checking mechanism
1.3	2020/08/04	1. Added a note about Linux host environment. (Chapter 1)
1.2	2020/07/23	1. Removed descriptions of built-in DTB on Andes boards and added a note that users can see the boot message for the DTB file suggested for their target systems.
1.1	2020/07/10	1. Changed the U-Boot package name “u-boot-riscv” to “u-boot” 2. Added how to build and use the MTD utility. (Section 3.3.4) 3. Added that the option “ <code>-with-arch=rv[32 64]v5</code> ” is required when configuring BBL for platforms that don’t support hardware floating-point instructions. 4. Added built-in DTB on Andes target boards and incorporated its use scenario. (Chapter 6) 5. Added the Pulse Width Modulation (PWM) modules in Andes Linux package and described how to configure them. (Section 9.2.2) 6. Added the Physical Memory Attribute (PMA) feature and described how it is supported. (Section 9.1.2)
1.0	2020/03/16	Initial release

Table of Contents

COPYRIGHT NOTICE	I
CONTACT INFORMATION	I
REVISION HISTORY	II
LIST OF TABLES	XII
LIST OF FIGURES	XIII
1. ABOUT THIS DOCUMENT	1
2. ANDES LINUX SOURCE FILES	3
2.1. OBTAINING SOURCES FROM ANDESIGHT PACKAGE	3
2.2. DOWNLOADING SOURCES FROM ONLINE GIT REPOSITORIES	3
3. ROOT FILE SYSTEM (ROOTFS)	5
3.1. SETTING UP ENVIRONMENT	5
3.1.1. <i>Setting up host PC environment</i>	5
3.1.2. <i>Loading binaries to the kernel file system at runtime</i>	5
3.2. PREPARING ROOT FILE SYSTEM	7
3.2.1. <i>Setup using a build script</i>	7
3.2.2. <i>Manual setup</i>	8
3.3. LINUX UTILITIES	11
3.3.1. <i>Gdbserver</i>	11
3.3.2. <i>Strace</i>	12
3.3.3. <i>Perf</i>	13
3.3.4. <i>MTD-Utills</i>	25
3.4. LINUX DEMO APPLICATIONS	27
3.4.1. <i>iPerf</i>	27
3.4.2. <i>MPlayer</i>	29
3.4.3. <i>Frame Buffer Test (FBtest)</i>	32
3.4.4. <i>Dhrystone</i>	34
3.4.5. <i>Whetstone</i>	36
3.4.6. <i>CoreMark</i>	38
4. LINUX KERNEL, DEVICE DRIVERS AND DEVICE TREE BLOB	40
4.1. BUILDING A LINUX KERNEL	40
4.2. BUILDING DEVICE DRIVERS.....	42
4.3. BUILDING A DEVICE TREE BLOB (DTB)	44

5.	FIRMWARE AND BOOTLOADER.....	46
5.1.	OPENSBI (SUPERVISOR BINARY INTERFACE)	46
5.2.	DAS U-BOOT	48
5.2.1.	<i>Building U-Boot SPL and ITB file for Normal Boot.....</i>	<i>50</i>
5.2.2.	<i>Building U-Boot SPL and ITB file for Fast Boot.....</i>	<i>53</i>
6.	BOOTING LINUX KERNEL.....	56
6.1.	NORMAL BOOT.....	58
6.1.1.	<i>Booting U-Boot.....</i>	<i>58</i>
6.1.2.	<i>Booting Linux kernel with U-Boot.....</i>	<i>61</i>
6.2.	FAST BOOT	66
6.2.1.	<i>Loading U-Boot SPL, kernel to RAM and booting kernel.....</i>	<i>68</i>
6.2.2.	<i>Burning U-Boor SPL, kernel to flash and booting kernel.....</i>	<i>70</i>
7.	BUILDING CONNECTION BETWEEN A LINUX TARGET AND A DEBUG HOST	72
7.1.	SETTING UP A LINUX TARGET	72
7.2.	SETTING UP A DEBUG HOST	75
8.	LINUX DISTRIBUTIONS AND CUSTOMIZATIONS	76
8.1.	COMMUNITY DISTRIBUTIONS	76
8.1.1.	<i>Fedora.....</i>	<i>76</i>
8.1.2.	<i>Debian.....</i>	<i>79</i>
8.1.3.	<i>Ubuntu.....</i>	<i>80</i>
8.2.	BUILD SYSTEM/SDK.....	80
9.	APPLICATION NOTES	81
9.1.	ADVANCED SETTINGS	81
9.1.1.	<i>HPM (Hardware Performance Monitor).....</i>	<i>81</i>
9.1.2.	<i>PMA (Physical Memory Attribute).....</i>	<i>82</i>
9.1.3.	<i>Huge pages.....</i>	<i>83</i>
9.1.4.	<i>PMP (Physical Memory Protection).....</i>	<i>85</i>
9.1.5.	<i>DMA coherence.....</i>	<i>86</i>
9.1.6.	<i>MSB mechanism (legacy non-cacheability mechanism)</i>	<i>87</i>
9.1.7.	<i>Asymmetric multi-processing (AMP) Linux kernel.....</i>	<i>87</i>
9.2.	SPECIAL PERIPHERALS	88
9.2.1.	<i>SMU (Power management).....</i>	<i>88</i>
9.2.2.	<i>PWM (Pulse Width Modulation).....</i>	<i>96</i>
9.2.3.	<i>dmatest (DMA test for memcpy).....</i>	<i>97</i>
9.2.4.	<i>GPIO (General-purpose input/output).....</i>	<i>99</i>

9.3.	BOOT FLOW UNVEILED	100
9.3.1.	<i>U-Boot SPL</i>	100
9.3.2.	<i>OpenSBI</i>	100
9.3.3.	<i>U-Boot proper (in Normal Boot process only)</i>	101
9.3.4.	<i>Linux</i>	101
9.3.5.	<i>User space</i>	101
10.	TROUBLESHOOTING	102
10.1.	BOOT PROCESS TROUBLESHOOTING	102
10.1.1.	<i>Debugging DTB</i>	102
10.1.2.	<i>Boot hangs in early stage due to lottery</i>	103
10.2.	INFORMATION FOR DIAGNOSING KERNEL BOOT FAILURE OR CRASHES	104
10.3.	ACCESS FAULT AT UNUSUAL PHYSICAL ADDRESS WHEN BRINGING UP PERIPHERAL DEVICES	105

List of Tables

TABLE 1. TARGET_LINK FILE FOR RESPECTIVE CPU AND ARCH	9
TABLE 2. LINK FILE FOR RESPECTIVE ARCH.....	10
TABLE 3. ANDES LINUX DRIVERS, DRIVER PATHS, AND CORRESPONDING IPs AND KERNEL CONFIGURATION NAMES	42



List of Figures

FIGURE 1. MEMORY LAYOUT FOR THE DEFAULT U-BOOT SPL 67

FIGURE 2. NETWORK BETWEEN HOST AND LINUX TARGET..... 72



Typographical Convention Index

Document Element	Font	Font Style	Size	Color
Normal text	Georgia	Normal	12	Black
Command line, source code or file paths	Lucida Console	Normal	11	Indigo
VARIABLES OR PARAMETERS IN COMMAND LINE, SOURCE CODE OR FILE PATHS	LUCIDA CONSOLE	BOLD + ALL-CAPS	11	INDIGO
Hyperlink	Georgia	<u>Underlined</u>	12	Blue

1. About this document

This document aims to provide useful information for Linux developers using Andes RISC-V target systems. It outlines methods to obtain sources and software components from the Andes Linux package, gives instructions on how to enable the software on AE350 platforms, and provides operational guidelines. For experienced users, it further introduces miscellaneous advanced usages in Chapter 9. For those who want to report kernel boot or crash problems to Andes, Chapter 10 describes the information they need to collect for the Andes technical support staff to diagnose and troubleshoot problems.

This document is structured in accordance with the development flow from the user's perspective. The following gives a top-down view of the typical development process on RISC-V systems:

1. A developer first obtains Linux sources, either from the Andes online repository or from the tarballs enclosed in AndeSight release packages.
2. With the sources, the developer
 - collects source files for user space (U-mode in RISC-V terminology) and compiles them into a root file system ("rootfs"),
 - builds the Linux kernel combined with the rootfs to serve as the S-mode program.
3. The developer proceeds to boot the kernel image either through GDB or U-Boot, debug the system with GDB, and kick-start the development cycle.

Andes also provides prebuilt Linux images in a GitLab repository for direct evaluation of Andes target platforms. If you are a user with an Andes evaluation board on hand and the permission to access the Andes-hosted GitLab server, please link to the [prebuilt linux image project](#) on the server to obtain the Linux image and DTB file required for your evaluation. For quick-start instructions on booting the system, just see the file [README.md](#) in the project.

NOTE

1. All Linux packages provided have been verified and tested on Ubuntu 20.04. However, other environments cannot be guaranteed due to build dependency issues.

2. All the shell scripts mentioned in this document are assumed to be executed using the Bash shell (i.e., `/bin/bash`). Please avoid using `sh` to execute the scripts in case that your `/bin/sh` does not link to `/bin/bash`.
-



2. Andes Linux source files

Andes Linux-related source files are released in two forms: one is as tarballs in the AndeSight package and the other is in Git format in Git repositories. The local directory that stores these Linux-related files from either source serves as the base directory for Linux development with Andes V5 targets and is represented as `<LINUX_ROOT>` in this document.

2.1. Obtaining sources from AndeSight package

The device drivers, utilities, and Linux kernel files for Linux development with Andes V5 targets are provided as tarballs in `<ANDESIGHT_ROOT>/Linux` of AndeSight package. All of these tarball files must be decompressed in their existing directories before you perform any tasks in the following chapters. The directory `<ANDESIGHT_ROOT>/Linux` in this approach can be taken as `<LINUX_ROOT>`.

2.2. Downloading sources from online Git repositories

You can also obtain the packages for Linux development in Git format online. The public repository at https://github.com/andestech/linux_devel_pack.git provides free Linux-related files from the second-to-last AndeSight releases. The latest Linux packages are available in the `linux_devel_pack` project of the restricted Andes GitLab repository hosted at <https://git.andestech.com>. If you are a licensed user of the most recent releases, you will get a credential e-mail that permits you to access Andes FTP and the `linux_devel_pack` project on the Andes GitLab server.

The `linux_devel_pack` project, either in the public or restricted repository, is a super-repository that hosts all packages for Linux development. After the project is retrieved from the GitHub or GitLab server, its local directory is ready for use as `<LINUX_ROOT>`. The steps to initialize the project are as follows:

- From the Andes GitLab repository

Step 1 Create a personal access token following the steps in

https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html.

Step 2 Add the SSL PEM file.

```
$ openssl s_client -showcerts -servername git.andestech.com \  
-connect git.andestech.com:443 </dev/null 2>/dev/null \  
| sed -n -e '/BEGIN\ CERTIFICATE/,/END\ CERTIFICATE/ p' \  
> ~/git-andestech-com.pem  
$ git config --global \  
http."https://git.andestech.com/".sslCAInfo \  
~/git-andestech-com.pem
```

Step 3 Download the linux_devel_pack project.

```
$ git clone  
https://oauth2:<ACCESS_TOKEN>@git.andestech.com/andes/linux_deve  
l_pack.git
```

Step 4 Modify the submodule URLs.

```
$ cd linux_devel_pack/  
$ sed -i 's|https://git.andestech.com/andes|..|g' .gitmodules  
$ git submodule sync
```

Step 5 Download the submodules.

```
$ git submodule update --init --recursive
```

- **From the public GitHub repository**

```
$ git clone https://github.com/andestech/linux_devel_pack.git  
$ cd linux_devel_pack/  
$ git submodule update --init --recursive
```

3. Root file system (Rootfs)

3.1. Setting up environment

3.1.1. Setting up host PC environment

Step 1 On the host terminal, set environment variables.

```
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
```

Step 2 Open the .bashrc file.

```
$ vi ~/.bashrc
```

Step 3 Append the following line to the bottom of the .bashrc file for including an appropriate Linux toolchain in PATH.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

Step 4 Reload the bash configuration file.

```
$ source ~/.bashrc
```

3.1.2. Loading binaries to the kernel file system at runtime

Program binaries can be copied to the kernel root file system at runtime through an SD card or via NFS.

● Through an SD card

Store the desired binary on an SD card and issue the following commands to mount the SD card on the kernel root file system:

```
# mount /dev/mmcblk0p1 /mnt
```

● Using NFS

To upload a program binary via NFS, follow below to configure the network interface first:

Step 1 Issue `udhcpc` to obtain the IP address of the DHCP server.

Step 2 Input the following to mount the file system exported via the NFS server:

```
# mount -t nfs -o nolock <NFS_SERVER_IP>:/local
```

<DESTINATION_PATH>



3.2. Preparing root file system

You can set up a root file system (rootfs) either automatically using the build script `Prepare_rootfs.sh` provided in `<LINUX_ROOT>` or manually. Please see the subsections below for instructions on the two methods and use absolute paths during the setup process. The build script method is strongly recommended as it simplifies the process and prevents possible errors caused by the complexity of Linux multilib toolchains.

3.2.1. Setup using a build script

Step 1 Create a new directory `<RAMDISK_ROOT>`.

Step 2 Run the build script with appropriate arguments.

```
$ <LINUX_ROOT>/Prepare_rootfs.sh \  
--toolchain_path=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN> \  
--CROSS_COMPILE=riscv[32|64]-linux- \  
--ramdisk_root_path=<RAMDISK_ROOT> \  
--tar_file_path=<LINUX_ROOT> \  
--arch=rv[32|64]v5[d] \  
--cpu=[25|45]
```

Where

- `--toolchain_path` specifies the absolute path to the top-level directory of the desired toolchain.
- `--tar_file_path=<LINUX_ROOT>` specifies the directory that contains busybox and rootfs
- `--arch=rv[32|64]v5[d]` specifies whether to use soft-float ABI (v5) or hard-float ABI (v5d)
- `--cpu=[25|45]` specifies which series core to use. Apply `--cpu=25` for a 25/27-series core and `--cpu=45` for a 45/60-series core. It is default set as `--cpu=25` when not explicitly specified.

3.2.2. Manual setup

Step 1 Create a new directory `<RAMDISK_ROOT>` and copy `rootfs` and `busybox` packages from `<LINUX_ROOT>`.

```
$ mkdir <RAMDISK_ROOT>
$ cd <RAMDISK_ROOT>
$ cp -r <LINUX_ROOT>/rootfs ./
$ cp -r <LINUX_ROOT>/busybox ./
```

Step 2 Follow below to copy libraries and binary tools for your toolchain from the AndeSight package to the root file system.

```
$ cp -arf <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/lib/*
    <RAMDISK_ROOT>/rootfs/disk/lib/

$ cp -arf <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/riscv[32|64]-
    linux/lib/* <RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>/

$ cp -arf <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/sbin/*
    <RAMDISK_ROOT>/rootfs/disk/sbin/

$ cp -arf
    <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/usr/sbin/*
    <RAMDISK_ROOT>/rootfs/disk/usr/sbin

$ cp -arf
    <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/usr/bin/*
    <RAMDISK_ROOT>/rootfs/disk/usr/bin
```

Add the following commands for the 25/27 series and 45/60 series respectively. Note that `<ABI_DIR>` in the commands refers to “lib64/lp64” or “lib32/ilp32” for soft-float ABI and “lib64/lp64d” or “lib32/ilp32d” for hard-float ABI.

- For 25 or 27 series

```
$ cp -arf
    <ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/<ABI_DIR>/*
    <RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>/

$ cp -arf
```

```

<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/usr/<ABI_DIR>/
* <RAMDISK_ROOT>/rootfs/disk/usr/<ABI_DIR>/
$ rm -rf <RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>/mtune-andes-45-
series
$ rm -rf <RAMDISK_ROOT>/rootfs/disk/usr/<ABI_DIR>/mtune-andes-
45-series
● For 45/60 series
$ cp -arf
<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/<ABI_DIR>/mtun
e-andes-45-series/* <RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>/
$ cp -arf
<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot/usr/<ABI_DIR>/mt
une-andes-45-series/* <RAMDISK_ROOT>/rootfs/disk/usr/<ABI_DIR>/
$ ln -fs . <RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>/mtune-andes-45-
series
$ ln -fs <TARGET_LINK> <RAMDISK_ROOT>/rootfs/disk/lib/<LINK>

```

For **<TARGET_LINK>** and **<LINK>** in the above command, see Table 1 and Table 2 below to specify pertinent files for your CPU and architecture (i.e., ABI).

Table 1. TARGET_LINK file for respective CPU and Arch

CPU	Arch	File for TARGET_LINK
45	rv32v5	ld-linux-riscv32-ilp32_andes-45-series.so.1
	rv32v5d	ld-linux-riscv32-ilp32d_andes-45-series.so.1
	rv64v5	ld-linux-riscv64-lp64_andes-45-series.so.1
	rv64v5d	ld-linux-riscv64-lp64d_andes-45-series.so.1
60	rv64v5	ld-linux-riscv64-lp64_andes-45-series.so.1
	rv64v5d	ld-linux-riscv64-lp64d_andes-45-series.so.1

Table 2. LINK file for respective Arch

Arch	File for LINK
rv32v5	ld-linux-riscv32-ilp32.so.1
rv32v5d	ld-linux-riscv32-ilp32d.so.1
rv64v5	ld-linux-riscv64-lp64.so.1
rv64v5d	ld-linux-riscv64-lp64d.so.1

Step 3 Make sure the strip program is included in `PATH` and remove unwanted debug information to reduce the code size. For example, strip debug information for nds64le-linux-glibc-v5d as follows:

```
$ riscv64-linux-strip --strip-unneeded
  <RAMDISK_ROOT>/rootfs/disk/lib/*
$ riscv64-linux-strip --strip-unneeded
  <RAMDISK_ROOT>/rootfs/disk/lib64/lp64/*
```

Step 4 Build BusyBox with the pre-written build script. Make sure that you define environment variables with ABI information (i.e., rv64v5/rv64v5d/rv32v5/rv32v5d) for the build. For example,

```
$ cd <RAMDISK_ROOT>/busybox
$ CFLAGS="-march=rv64v5" LDFLAGS="-march=rv64v5" \
  ./build_busybox.sh -build riscv64-linux-
$ ./build_busybox.sh -install <RAMDISK_ROOT>/rootfs/disk
```

3.3. Linux utilities

You may include additional utilities, such as dropbear or gdbserver, by using Andes toolchains to cross-compile them and putting them into `<RAMDISK_ROOT>/rootfs/disk/usr/bin`.

3.3.1. Gdbserver

Gdbserver is a control program that allows you to run GDB on a remote system for debugging. You can use the GDB source code in the AndeSight package to build gdbserver following below:

Step 1 Create a directory to build the gdbserver binary.

```
$ mkdir build-gdbserver
```

Step 2 Export the toolchain directory to the environment variable `$PATH`.

```
$ export PATH=/<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

Step 3 Switch to the directory that was previously created.

```
$ cd build-gdbserver
```

Step 4 Configure gdbserver under the GDB source directory (`<GDB_SRC>`) to run on the Andes Linux system.

```
$ <GDB_SRC>/configure CFLAGS="-march=rv[32|64]v5[d]" CPPFLAGS="-march=rv[32|64]v5[d]" CXXFLAGS="-march=rv[32|64]v5[d]" LDFLAGS="-march=rv[32|64]v5[d] -static" --host=riscv[32|64]-linux --enable-gdbserver
```

Note that `<GDB_SRC>` is in `<LINUX_ROOT>/gdb` by default.

Step 5 Build gdbserver and find the binary built under the `gdbserver/` directory.

```
$ make all-gdbserver
```

For instructions on running the gdbserver, see Section 7.1.

3.3.2. Strace

Strace is a useful tool for diagnostics, instructional tasks, and debugging. Strace intercepts and records system calls generated by a process and the signals received by a process. The following instructions describe how to build strace in the AndeSight package.

Step 1 Export the relevant toolchain directory to \$PATH.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

Step 2 Change the current directory to the strace source, which is under <LINUX_ROOT> by default.

```
$ cd <LINUX_ROOT>/strace
```

Step 3 Bootstrap the build script.

```
$ ./bootstrap
```

Step 4 Configure the build environment.

```
$ ./configure CFLAGS="-march=rv[32|64]v5[d]" LDFLAGS="-  
march=rv[32|64]v5[d] -static -lrt -pthread" --  
host=riscv[32|64]-unknown-linux
```

Step 5 Build strace and find its binary under <LINUX_ROOT>/strace/src.

```
$ make
```

For more details about strace options and usages, see [its online manual](#).

3.3.3. Perf

Building perf

Perf requires an auxiliary library, libelf, to resolve symbols of a user space program and report; without libelf, perf events can only be recorded in the raw address format. Currently, libelf can only be enabled manually. Please follow the step-by-step instructions below to build the libelf dependency and perf.

- Step 1** Issue the following commands to fetch the perf source code from the Andes Linux kernel source directory.
- ```
$ cp -a /<LINUX_ROOT>/linux-6.1/tools/ $PWD
```
- Step 2** Issue the following commands to fetch sources of elfutils, the parent project of libelf, and zlib from upstream releases. The highlighted parts below show the preferred versions of zlib and elfutils.
- ```
$ git clone https://sourceware.org/git/elfutils.git -b elfutils-0.178
$ git clone https://github.com/madler/zlib -b v1.2.11
```
- Step 3** Set the necessary environment variables and prepare a temporary sysroot.
- ```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export CROSS_COMPILE=riscv[32|64]-linux-
$ export TOP=$PWD
$ mkdir sysroot_tmp
$ mkdir scripts
$ cp -arf /<LINUX_ROOT>/linux-6.1/scripts/bpf_doc.py ./scripts/
```
- Step 4** Build zlib.
- ```
$ cd zlib
$ CC=${CROSS_COMPILE}gcc CFLAGS+='-
    march=rv[32|64]v5[d]' ./configure --prefix=$TOP/sysroot_tmp
$ make install
$ cd $TOP
```

Step 5 Prepare the script `fixup.sh` to resolve errors during the build of elfutils (Step 6). elfutils isn't designed for cross-compilation. Parts of its dependencies (i.e., shared objects `libeu` and `lib`) expect `--host` and `--target` to be aligned with your build machine at the first stage of the building and yet with the cross-machine architecture (i.e., Andes V5 ISA) in the second stage. Such inconsistency will result in a build error during the build process. To resolve the issue, follow below to prepare the script `fixup.s` in advance.

1. Open a code editor, paste the following code snippet, and save it as `fixup.sh`.

```
#!/bin/sh
# The following is fixup.sh.
patch_makefile() {
    sed -ri 's/riscv(32|64)(-unknown)?-linux-//' $1
    sed -ri 's/-wnull-dereference//' $1
    sed -ri 's/-march=rv(32|64)v5d?//' $1
    sed -ri 's/-wimplicit-fallthrough=5//' $1
    sed -ri 's/-wduplicated-cond//' $1
}

cp -a ./lib ./lib.riscv
cd lib
patch_makefile ./Makefile
make clean all
cd ..
cp -a ./libcpu/ ./libcpu.failed
cd libcpu
patch_makefile ./Makefile
make clean i386_gendis
cp ../libcpu.failed/Makefile .
make
cd ..
make
mv ./lib ./lib.x86
cp -a ./lib.riscv/ ./lib
```

`make`

2. Give execute permission to the script.

```
$ chmod +x fixup.sh
```

Step 6 Build elfutils to facilitate perf's functionalities:

1. Change to the elfutils directory.

```
$ cd elfutils
```

2. Edit `libeb1.h`. Search for `eb1_syscall_abi` and change its last parameter `"int args[6]"` to `"int *args"`.

```
$ vim libeb1/libeb1.h
```

3. Edit `libdw.h`. Search for `dwarf_frame_register` and change its parameter `"Dwarf_Op ops_mem[3]"` to `"Dwarf_Op *ops_mem"`.

```
$ vim libdw/libdw.h
```

4. Proceed with the following commands.

```
$ aclocal
$ autoheader
$ autoconf
$ autoreconf -f -i
$ automake --add-missing
$ ./configure CFLAGS="-O2 -I${TOP}/sysroot_tmp/include -fPIC
-march=rv[32|64]v5[d] " LDFLAGS="-L${TOP}/sysroot_tmp/lib
-lz " --host=riscv[32|64]-unknown-linux --
target=riscv[32|64]-unknown-linux --enable-maintainer-mode
--disable-debuginfod --prefix=${TOP}/sysroot_tmp
$ make
```

5. A build error will occur during the `make` process. To fix the problem, put the script `fixup.sh` created in Step 5 to the build directory and execute it.

```
$ cp ${TOP}/fixup.sh .
$ ./fixup.sh
```

```
$ make install
$ cd $TOP
```

Step 7 Copy shared objects (i.e., `libelf*.so*`, `libdw*.so*` and `libz*.so*`) from `$TOP/sysroot_tmp/lib` to your library search path(s). For Andes targets, the search path is `<RAMDISK_ROOT>/rootfs/disk/<ABI_DIR>` in which `<ABI_DIR>` refers to `/lib32/rlp32[d]` for 32-bit platforms and `/lib64/lp64[d]` for 64-bit platforms.

Step 8 Prepare a build script `build_features.sh` to enable perf features.

1. Paste the following shell script snippet to a code editor and save it as `build_features.sh`.

```
#!/bin/sh
CC_AND_FLAGS="CC=${CROSS_COMPILE}gcc \
CFLAGS='-I${TOP}/sysroot_tmp/include \
-I${TOP}/sysroot_tmp/usr/local/include -march=rv[32|64]v5[d]' \
LDFLAGS='-L${TOP}/sysroot_tmp/lib/ \
-L${TOP}/sysroot_tmp/usr/local/lib/ -lz -lelf -ldw'"

eval $CC_AND_FLAGS make test-libelf.bin
eval $CC_AND_FLAGS make test-glibc.bin
eval $CC_AND_FLAGS make test-pthread-attr-setaffinity-np.bin
eval $CC_AND_FLAGS make test-dwarf.bin
eval $CC_AND_FLAGS make test-dwarf_getlocations.bin
eval $CC_AND_FLAGS make test-libdw-dwarf-unwind.bin
eval $CC_AND_FLAGS make test-libelf-getphdrnum.bin
eval $CC_AND_FLAGS make test-libelf-gelf_getnote.bin
eval $CC_AND_FLAGS make test-libelf-getshdrstrndx.bin
```

2. Give execute permission to the script.

```
$ chmod +x build_features.sh
```

Step 9 Run the script `build_features.sh` to enable libelf and libdwarf features for perf.

```
$ cd tools/build/feature/
```

```
$ cp $TOP/build_features.sh .
$ ./build_features.sh
$ cd $TOP
```

Step 10 Build perf. Note that for 60-series cores, use `-mtune=andes-45-series` in the following.

```
$ cd tools/perf/
$ ARCH=riscv CROSS_COMPILE=$CROSS_COMPILE EXTRA_CFLAGS='-
  I${TOP}/sysroot_tmp/include -L${TOP}/sysroot_tmp/lib -
  march=rv[32|64]v5[d] -mtune=andes-[25|45]-series' NO_LIBPERL=1
  NO_LIBPYTHON=1 NO_SLANG=1 NO_GTK2=1 NO_LIBNUMA=1 NO_LIBAUDIT=1
  NO_STRLCOPY=1 NO_LIBCRYPTO=1 NO_JVMTI=1 WERROR=0 make perf
```

Now you will have a working perf.

NOTE

The default version of GNU Make on CentOS 7 is v3.82, which is incompatible with current Makefiles shipped with perf. If you're a CentOS user, please download GNU Make v4.2.1 from the Red Hat Software Collection (SCLo) repository

(http://mirror.centos.org/centos/7/sclo/x86_64/rh/Packages/d/devtoolset-7-make-4.2.1-2.el7.x86_64.rpm) before building perf.

If you have concerns about using SCLo packages, just follow below to download and extract the GNU Make v4.2.1 from the repository and export it to the environment variable `$PATH` for a temporary use:

```
$ cd $TOP
$ wget
  http://mirror.centos.org/centos/7/sclo/x86_64/rh/Packages/d/de
  vtoolset-7-make-4.2.1-2.el7.x86_64.rpm -qO- | rpm2cpio | cpio
  -id
$ export PATH=$PWD/opt/rh/devtoolset-7/root/usr/bin:$PATH
```

Using perf

The following lists some frequent usages of perf.

- Listing the events supported by perf

```
# perf list
```

A list of pre-defined events will be shown.

List of pre-defined events (to be used in -e or -M):

branch-instructions OR branches	[Hardware event]
branch-misses	[Hardware event]
bus-cycles	[Hardware event]
cache-misses	[Hardware event]
cache-references	[Hardware event]
cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
ref-cycles	[Hardware event]
stalled-cycles-backend OR idle-cycles-backend	[Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend	[Hardware event]
alignment-faults	[Software event]
bpf-output	[Software event]
cgroup-switches	[Software event]
context-switches OR cs	[Software event]
cpu-clock	[Software event]
cpu-migrations OR migrations	[Software event]
dummy	[Software event]
emulation-faults	[Software event]
major-faults	[Software event]
minor-faults	[Software event]
page-faults OR faults	[Software event]
task-clock	[Software event]
duration_time	[Tool event]
user_time	[Tool event]
system_time	[Tool event]
L1-dcache-load-misses	[Hardware cache event]
L1-dcache-loads	[Hardware cache event]
L1-dcache-prefetch-misses	[Hardware cache event]
L1-dcache-prefetches	[Hardware cache event]
L1-dcache-store-misses	[Hardware cache event]
L1-dcache-stores	[Hardware cache event]
L1-icache-load-misses	[Hardware cache event]
L1-icache-loads	[Hardware cache event]
L1-icache-prefetch-misses	[Hardware cache event]
L1-icache-prefetches	[Hardware cache event]
LLC-load-misses	[Hardware cache event]
LLC-loads	[Hardware cache event]
LLC-prefetch-misses	[Hardware cache event]
LLC-prefetches	[Hardware cache event]

```

LLC-store-misses      [Hardware cache event]
LLC-stores             [Hardware cache event]
branch-load-misses    [Hardware cache event]
branch-loads          [Hardware cache event]
dTLB-load-misses      [Hardware cache event]
dTLB-loads            [Hardware cache event]
dTLB-prefetch-misses  [Hardware cache event]
dTLB-prefetches       [Hardware cache event]
dTLB-store-misses     [Hardware cache event]
dTLB-stores           [Hardware cache event]
iTLB-load-misses      [Hardware cache event]
iTLB-loads            [Hardware cache event]
node-load-misses      [Hardware cache event]
node-loads            [Hardware cache event]
node-prefetch-misses  [Hardware cache event]
node-prefetches       [Hardware cache event]
node-store-misses     [Hardware cache event]
node-stores           [Hardware cache event]

rNNN                  [Raw hardware event descriptor]
cpu/t1=v1[,t2=v2,t3 ...]/modifier [Raw hardware event descriptor]
(see 'man perf-list' on how to encode it)

mem:<addr>[/len][:access] [Hardware breakpoint]

```

■ Counting

- Counting user mode events

```
# perf stat -e cycles:u ls
```

- Counting kernel mode events

```
# perf stat -e cycles:k ls
```

- Counting all events

```
# perf stat -e cycles ls
```

```

~ # perf stat -e cycles ls
root

Performance counter stats for 'ls':

      95127949      cycles

    0.060416067 seconds time elapsed

    0.000000000 seconds user
    0.054915000 seconds sys

```

■ Sampling

- Start sampling kernel instructions

```
# perf record -e cycles:k ls
```

- Show the result

```
# perf report
```

The sample analysis will be displayed as depicted below.

```
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 93 of event 'cycles:k'
# Event count (approx.): 5944546
#
# Overhead Command Shared Object Symbol
# .....
#
3.58% ls [kernel.kallsyms] [k] do_raw_spin_lock
3.49% ls [kernel.kallsyms] [k] mntput_no_expire
2.91% ls [kernel.kallsyms] [k] __asm_copy_to_user
2.76% ls [kernel.kallsyms] [k] next_uptodate_page
2.69% ls [kernel.kallsyms] [k] do_raw_spin_unlock
2.47% ls [kernel.kallsyms] [k] generic_fillattr
1.92% ls [kernel.kallsyms] [k] generic_permission
1.60% ls [kernel.kallsyms] [k] unlink_file_vma
1.60% ls [kernel.kallsyms] [k] memcpy
1.59% ls libc.so.6 [.] strlen
1.59% ls libc.so.6 [.] 0x0000000000053df4
1.59% ls [kernel.kallsyms] [k] __d_lookup_rcu
1.57% ls [kernel.kallsyms] [k] path_put
1.57% ls [kernel.kallsyms] [k] mas_next_entry
1.56% ls [kernel.kallsyms] [k] _save_context
1.56% ls libc.so.6 [.] printf
1.56% ls libc.so.6 [.] 0x0000000000065a12
1.54% ls [kernel.kallsyms] [k] __lock_text_start
1.53% ls [kernel.kallsyms] [k] PageHuge
1.52% ls busybox [.] 0x000000000007d3e2
1.52% ls [kernel.kallsyms] [k] should_failslab
1.52% ls libc.so.6 [.] 0x0000000000053ca6
1.52% ls libc.so.6 [.] gnu_dev_major
1.50% ls libc.so.6 [.] 0x00000000000659f0
1.49% ls [kernel.kallsyms] [k] _raw_spin_unlock_irqrestore
1.49% ls ld-linux-riscv64-lp64d.so.1 [.] 0x000000000007c50
1.48% ls [kernel.kallsyms] [k] uart_start
1.48% ls libc.so.6 [.] memset
1.48% ls [kernel.kallsyms] [k] dput
1.47% ls [kernel.kallsyms] [k] kmem_cache_alloc
1.47% ls [kernel.kallsyms] [k] lock_mm_and_find_vma
1.46% ls [kernel.kallsyms] [k] step_into
1.45% ls [kernel.kallsyms] [k] __fput
1.45% ls libc.so.6 [.] strchrnul
1.44% ls [kernel.kallsyms] [k] vfs_write
1.43% ls busybox [.] 0x000000000007cd0e
1.43% ls busybox [.] 0x000000000007cd70
1.42% ls [kernel.kallsyms] [k] __anon_vma_prepare
1.41% ls libc.so.6 [.] 0x000000000007370e
1.41% ls [kernel.kallsyms] [k] release_pages
1.40% ls [kernel.kallsyms] [k] __rcu_read_unlock
```

```

1.39% ls      libc.so.6      [.] memcpy
1.38% ls      [kernel.kallsyms] [k] walk_component
1.37% ls      [kernel.kallsyms] [k] strncpy_from_user
1.37% ls      [kernel.kallsyms] [k] folio_flags.constprop.0
1.36% ls      libc.so.6      [.] 0x0000000000054638
1.35% ls      ld-linux-riscv64-lp64d.so.1 [.] 0x00000000000b778
1.31% ls      libc.so.6      [.] readdir64
1.30% ls      [kernel.kallsyms] [k] memset
1.29% ls      [kernel.kallsyms] [k] check_syscall_nr
1.24% ls      [kernel.kallsyms] [k] _raw_spin_unlock
1.22% ls      ld-linux-riscv64-lp64d.so.1 [.] 0x000000000007dc0
1.22% ls      [kernel.kallsyms] [k] link_path_walk
1.16% ls      [kernel.kallsyms] [k] page_add_new_anon_rmap
1.14% ls      [kernel.kallsyms] [k] may_expand_vm
1.07% ls      ld-linux-riscv64-lp64d.so.1 [.] 0x00000000000d508
1.05% ls      [kernel.kallsyms] [k] call_rcu
0.98% ls      ld-linux-riscv64-lp64d.so.1 [.] 0x000000000001695c
0.96% ls      [kernel.kallsyms] [k] getname_flags
0.89% ls      [kernel.kallsyms] [k] folio_unlock
0.86% ls      [kernel.kallsyms] [k] lockref_mark_dead
0.79% ls      [kernel.kallsyms] [k] d_add
0.79% ls      [kernel.kallsyms] [k] __sbi_tlb_flush_range.constprop.0
0.72% ls      [kernel.kallsyms] [k] kmem_cache_free
0.70% ls      [kernel.kallsyms] [k] __softirqentry_text_start
0.52% perf-ex [kernel.kallsyms] [k] preempt_schedule_irq
0.49% perf-ex [kernel.kallsyms] [k] finish_task_switch.isra.0
0.47% ls      [kernel.kallsyms] [k] __d_drop
0.46% ls      [kernel.kallsyms] [k] _find_next_bit
0.27% perf-ex [kernel.kallsyms] [k] __softirqentry_text_start
0.00% perf-ex [kernel.kallsyms] [k] perf_event_exec

```

For more details about performance events, see *AndeStar V5 System Privileged Architecture and CSR Specification* and *AndesCore data sheets*.

NOTE

1. The libelf support for perf makes it possible to record and report user space programs compiled with debug symbols and the libdwf support allows you to track down inlined function calls with the option “`--call-graph dwarf`” during the recording.
2. If you’re recording user space programs without root privileges, set the following sysfs options to “0” so that perf can be permitted to access `/proc/kallsyms` and get correct information of kernel symbols:
 - `/proc/sys/kernel/perf_event_paranoid`
 - `/proc/sys/kernel/kptr_restrict`
3. Call graph tracking based on libdwf is very resource-demanding in terms of disk storage and CPU computing power. If you encounter a drastic slowdown or use up disk/RAM space, it is suggested to tune down the sampling rate from the default 4,000 Hz to around 1,000 Hz. To do so, just add “`-F 1000`” to the `record` command, as shown below:


```
perf record --call-graph dwarf -F 1000 [PROGRAM_AND_ARGUMENTS]
```
4. Call graphs recorded with libdwf support can be too exhaustive to read sometimes. To resolve the problem, you may use S-lang to bring up the TUI mode for a clearer display like below.

Samples: 846 of event 'cycles', Event count (approx.): 20874513

Children	Self	Command	Shared Object	Symbol
+ 73.40%	70.52%	test_dwarf_perf	test_dwarf_perf	[.] fib
- 28.24%	0.00%	test_dwarf_perf	test_dwarf_perf	[.] fib inlined at ./test_dwarf_perf.c:7:17 in fib (inlined)
- fib inlined at ./test_dwarf_perf.c:7:17 in fib (inlined)				
- 16.61%				fib
+ 8.39%				fib
0.87%				ret_from_exception
0.67%				fib inlined at ./test_dwarf_perf.c:7:17 in fib (inlined)
- 1.30%				ret_from_exception
- 1.16%				__lock_text_end
				irq_exit
				__softirqentry_text_start
+ 25.85%	0.00%	test_dwarf_perf	test_dwarf_perf	[.] main
+ 25.82%	0.14%	test_dwarf_perf	libc-2.29.9000.so	[.] __libc_start_main
+ 24.97%	0.00%	test_dwarf_perf	test_dwarf_perf	[.] fib (inlined)
+ 24.97%	0.00%	test_dwarf_perf	test_dwarf_perf	[.] fib inlined at ./test_dwarf_perf.c:7:17 in main (inlined)
+ 20.97%	0.00%	test_dwarf_perf	test_dwarf_perf	[.] _start
+ 14.19%	0.00%	test_dwarf_perf	[kernel.kallsyms]	[k] ret_from_exception
+ 13.09%	0.00%	test_dwarf_perf	ld-linux-riscv32-ilp32d.so.1	[.] _dl_map_object
+ 12.24%	0.15%	test_dwarf_perf	ld-linux-riscv32-ilp32d.so.1	[.] open_path
+ 10.88%	0.00%	test_dwarf_perf	ld-linux-riscv32-ilp32d.so.1	[.] _dl_catch_exception
+ 10.88%	0.00%	test_dwarf_perf	ld-linux-riscv32-ilp32d.so.1	[.] openaux
+ 10.29%	0.16%	test_dwarf_perf	[kernel.kallsyms]	[k] ret_from_syscall

This will require you to change the building procedure of perf as follows:

Step 1 Follow instructions in Step 1~Step 6 of Building perf

Step 2 Download S-lang v2.3.2 from its [official website](#) and extract the compressed archive.

Step 3 Patch S-lang to force colored outputs on AE350 platforms:

1. Open an editor, paste the following code snippet and save it as `force_color.patch`.

```
diff --git a/src/sldisply.c b/src/sldisply.c
index 2664aad..67b44fa 100644
--- a/src/sldisply.c
+++ b/src/sldisply.c
@@ -160,7 +160,7 @@ int SLtt_Screen_Cols = DEFAULT_SCREEN_COLS;
int SLtt_Screen_Rows = DEFAULT_SCREEN_ROWS;
int SLtt_Term_Cannot_Insert = 0;
int SLtt_Term_Cannot_Scroll = 0;
-int SLtt_Use_Ansi_Colors = 0;
+int SLtt_Use_Ansi_Colors = 1;
int SLtt_Blink_Mode = 0;
int SLtt_Use_Blink_For_ACS = 0;
int SLtt_Newline_Ok = 0;
@@ -3238,7 +3238,7 @@ int _pSLtt_init_cmdline_mode (void)
return 0;

SLtt_Term_Cannot_Scroll = 1;
- SLtt_Use_Ansi_Colors = 0;
+ SLtt_Use_Ansi_Colors = 1;
Use_Relative_Cursor_Addresssing = 1;
return 1;
}
diff --git a/src/slvideo.c b/src/slvideo.c
index f43ac2e..d3dc3f8 100644
--- a/src/slvideo.c
+++ b/src/slvideo.c
@@ -34,7 +34,7 @@ USA.
int SLtt_Term_Cannot_Insert = 0;
int SLtt_Term_Cannot_Scroll = 0;
int SLtt_Ignore_Beep = 3;
-int SLtt_Use_Ansi_Colors = 0;
+int SLtt_Use_Ansi_Colors = 1;
int SLtt_Has_Status_Line = 0;
int SLtt_Screen_Rows = 25;
int SLtt_Screen_Cols = 80;
```

2. Put `force_color.patch` in the source directory of S-lang and issue the following command to patch S-lang:

```
patch -p1 < force_color.patch
```

Step 4 Build S-lang with the following commands:

```
$ CFLAGS="-I${TOP}/sysroot_tmp/include" LDFLAGS="-  
L${TOP}/sysroot_tmp/lib" ./configure --host=riscv[32|64]-linux  
--target=riscv[32|64]-linux --without-pcre --without-png  
$ make  
$ make DESTDIR=${TOP}/sysroot_tmp install
```

Step 5 Follow Step 7 of the perf building process to copy shared objects to your library search paths. Make sure you copy additional `libslang*.so` into the `rootfs` directory.

Step 6 Follow Step 8 and Step 9 of the perf building process to enable `libelf` and `libdwarf` support for perf. In Step 8, append the line `"$CC_AND_FLAGS make test-libelf-mmap.bin"` to the script `build_features.sh` for feature detection.

Step 7 Follow Step 10 of the perf building process to build perf but remove the option `"NO_SLANG=1"`.

3.3.4. MTD-Utills

The mtd-utils are used to perform read/write/erase operations to the SPI NOR flash devices. The SPI feature is not supported by default in the Linux kernel. To enable the feature for a kernel being built, just run “make menuconfig” and configure it as follows:

```
Device Drivers --->
<*> Memory Technology Device (MTD) support --->
    <*> SPI NOR device support --->

[*] SPI support --->
    <*> Andes ATCSPI200 SPI controller
```

To build the mtd-utils, proceed as follows:

Step 1 Export the relevant toolchain directory to \$PATH.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

Step 2 Change the current directory to the source of the mtd-utils, which is under <LINUX_ROOT> by default.

```
$ cd <LINUX_ROOT>/mtd
```

Step 3 Set the environment variables.

```
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export MARCH=rv[32|64]v5[d]
```

Step 4 Run the build script to build the mtd-utils.

```
$ ./build.sh
```

Using the mtd-utils

■ Checking the information of a flash device

```
mtd_debug info <DEVICE>
```

For example, to query the information of the device “mtd0”, issue as follows:

```
$ ./mtd_debug info /dev/mtd0
```

■ Erasing a flash device

```
mtd_debug erase <DEVICE> <OFFSET> <LENGTH>
```


For example, to erase 0x1000 bytes from the address 0x0 of the flash device “`mtd0`”, issue as follows:

```
$ ./mtd_debug erase /dev/mtd0 0x0 0x1000
```

NOTE

If the content of the flash is critical, make sure you back it up before performing the erase operation.



■ Reading a flash device

```
mtd_debug read <DEVICE> <OFFSET> <LENGTH> <DESTINATION_FILE>
```

For example,

```
$ ./mtd_debug read /dev/mtd0 0x0 16 data
```

This is to read 16 bytes from the address 0x0 of the flash device “`mtd0`” and copy them into a file named “`data`”.

■ Writing a flash device

```
mtd_debug write <DEVICE> <OFFSET> <LENGTH> <SOURCE_FILE>
```

For example,

```
$ ./mtd_debug write /dev/mtd0 0x0 16 data
```

This is to copy 16 bytes from a file named “`data`” and write them into the address 0x0 of the flash device “`mtd0`”.

3.4. Linux demo applications

3.4.1. iPerf

“iPerf” is a Linux demo application using iperf3 to test network performance. The application is placed under `<ANDESIGHT_ROOT>/demo/linux`. It measures TCP or UDP throughput between the client and server and outputs the transfer rate and bandwidth performance over time.

Loading iPerf to the kernel file system

See Section 3.1.2 to copy the iperf3 binary to the Linux root file system.

Building iPerf

Step 1 Export the appropriate toolchain directory to `$PATH` and set the environment variables for the ABI to use.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export CFLAGS="-march=rv[32|64]v5[d]" LDFLAGS="-march=rv[32|64]v5[d] -static"
```

Step 2 Extract the source code of the iPerf demo application, which is located under `<ANDESIGHT_ROOT>/demo/linux`.

```
$ tar -zxf iperf.tgz
```

Step 3 Change the current directory to the directory containing the source code of the iPerf demo application.

```
$ cd iperf
```

Step 4 Build the iperf3 binary.

```
$ ./bootstrap.sh
$ ./configure --host=riscv[32|64]-linux --enable-static --disable-shared
$ make
```

The binary will be generated in `<IPERF_ROOT>/src`.

Running iPerf

- Step 1** Copy the iPerf demo application from `<ANDESIGHT_ROOT>/demo/linux` to an SD card.

Note that iperf3 requires two systems, which respectively act as a server and a client. In this case, the host PC serves as one system, whereas the target system using the SD card with the iperf3 binary serves as the other. For the host PC, download and install iperf3 following the instructions on <https://iperf.fr/iperf-download.php>.

- Step 2** See *AndeSight v5.3 BSP User Manual* to boot up an AndeShape platform.

- Step 3** See Chapter 6 to boot up the Linux kernel.

- Step 4** Mount the SD card containing the iperf3 application on the Linux root file system.

```
$ mount /dev/mmcblk0p1 /mnt
```

- Step 5** Change the current directory to the Linux root file system.

```
$ cd /mnt
```

- Step 6** Run the demo application:

Launch the application from the SD card on the target system (as shown below), and from the host PC (Linux, Windows ...).

```
$ ./iperf3 -s #run in a server mode
```

```
$ ./iperf3 -c <SERVER_IP> #run in a client mode
```

To view the complete list of options for iperf3, just issue `./iperf3 -h`. You may also see <https://iperf.fr/> for more information about iPerf.

3.4.2. MPlayer

The demo application “MPlayer” is used to play WAV or MP3 audio files. It is placed under `<ANDESIGHT_ROOT>/demo/linux` and requires the support of the legacy DMA driver (ATCDMAC300). This application is not supported if the driver for the DMA engine (ATCDMAC300G), rather than the legacy DMA driver, is enabled.

Loading MPlayer to the kernel file system

See Section 3.1.2 to copy the MPlayer binary to the Linux root file system.

Building MPlayer

- Step 1** Export the appropriate toolchain directory to `$PATH` and set the environment variables.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export MARCH=rv[32|64]v5[d]
```

- Step 2** Extract the source code of the MPlayer demo application, which is located under `<ANDESIGHT_ROOT>/demo/linux`.

```
$ tar -zxf mplayer.tgz
```

- Step 3** Change the current directory to the directory containing the source code of the MPlayer demo application.

```
$ cd mplayer
```

- Step 4** Build the demo application using the build script.

```
$ ./build_mplayer.sh
```

The MPlayer binary “`mplayer`” is generated under the `demo_mplayer` directory along with a test file “`testwav.wav`”.

Running MPlayer

- Step 1** Copy the MPlayer binary “`mplayer`” and the test file “`testwav.wav`” to an SD card.

Step 2 See *AndeSight v5.3 BSP User Manual* to boot up an AndeShape platform.

Step 3 See Chapter 6 to boot up the Linux kernel.

Step 4 Mount the SD card containing the demo application on the Linux root file system.

```
$ mount /dev/mmcblk0p1 /mnt
```

Step 5 Change the current directory to the Linux root file system.

```
$ cd /mnt
```

Step 6 Run the demo application:

Plug your headset into the LINE_OUT1 port on your target board and launch the application from the SD card to play the test WAV file on the target system.

```
$ ./mplayer testwav.wav
```

If you use an audio file with a sampling rate other than 48 KHz, you can resample the file with the following command to enhance the sound quality:

```
$ ./mplayer -af resample=48000 <AUDIO>.wav
```

NOTE

Running the MPlayer application requires the legacy DMA driver. If you encounter problems running the application, follow below to examine the built-in DTB of your FPGA bitmap and modify the DTS to enable the legacy DMA driver if the DTB only supports the driver of the DMA engine.

Step 1 Follow the note instructions in Section 4.3 to retrieve the built-in DTB of your FPGA bitmap and inspect the device tree for a DMA node with legacy DMA.

Step 2 If the DTS file doesn't contain a DMA node with legacy DMA, modify it as follows:

■ For the mmc node

- Modify the compatible string as “andestech,atfsdc010”
 - Remove the following lines
- ```
dma0 = <&dma0 9>;
dma-names = "rxtx";
```

■ For the dma node

- Remove the “dma0:” label before the node name “dma@f0c00000”
- Change the compatible string to “andestech,atcdmac300”
- Change the interrupts to “<0xa 0x4 0x40 0x4 0x41 0x4 0x42 0x4 0x43 0x4 0x44 0x4 0x45 0x4 0x46 0x4 0x47 0x4>;”
- Remove the line “#dma-cells = <1>;”
- Check if the attribute “dma-coherent” is specified for the mmc node of the DTS file. If yes, add it to the dma node for the sake of consistency.

**Step 3** Follow the instructions in Section 4.3 to create a DTB file that supports the legacy DMA driver for your kernel.

---

### 3.4.3. Frame Buffer Test (FBtest)

The demo application “FBtest” is used to test the Linux frame buffer on LCD monitors. It is placed under `<ANDESIGHT_ROOT>/demo/linux`.

#### Configuring Linux kernel to support the frame buffer device

The FBtest demo program requires the support of the frame buffer device in the Linux kernel. For example, to run the application on an LCD display of FTLCDC100, you will need to enable the LCD driver support before the kernel build process (see Chapter 4) by running “`make menuconfig`” and specifying the option `CONFIG_FB_FTLCDC100=y` through the following path:

```
Prompt: Faraday FTLCDC100 driver
Location:
 -> Device Drivers
 -> Graphics support
 -> Frame buffer Devices
```

#### Loading FBtest to the kernel file system

See Section 3.1.2 to copy the FBtest binary to the Linux root file system.

#### Building FBtest

**Step 1** Export the appropriate toolchain directory to `$PATH` and set the environment variables.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export MARCH=rv[32|64]v5[d]
```

**Step 2** Extract the source code of the FBtest demo application, which is located under `<ANDESIGHT_ROOT>/demo/linux`.

```
$ tar -zxf fbtest.tgz
```

**Step 3** Change the current directory to the directory containing the source code of the FBtest demo application.

```
$ cd fbtest
```

**Step 4** Build the demo application using the build script.

```
$ make
```

The FBtest binary “`riscv[32|64]-linux-fbtest`” is generated under

```
<FBTEST_ROOT>/ .
```

### Running FBtest

**Step 1** Copy the FBtest binary “`riscv[32|64]-linux-fbtest`” to an SD card.

**Step 2** See *AndeSight v5.3 BSP User Manual* to set up an AndeShape platform.

**Step 3** See Chapter 6 to boot up the Linux kernel.

**Step 4** Mount the SD card containing the demo application on the Linux root file system.

```
$ mount /dev/mmcb1k0p1 /mnt
```

**Step 5** Change the current directory to the Linux root file system.

```
$ cd /mnt
```

**Step 6** Run the demo application:

```
$./riscv[32|64]-linux-fbtest
```

Issue the following command to specify a test case:

```
$./riscv[32|64]-linux-fbtest test[001~012]
```



### 3.4.4. Dhrystone

The demo application “Dhrystone” is a synthetic computing benchmark program. It is placed under `<ANDESIGHT_ROOT>/demo/linux`.

#### Loading Dhrystone to the kernel file system

See Section 3.1.2 to copy the Dhrystone binary to the Linux root file system.

#### Building Dhrystone

- Step 1** Export the appropriate toolchain directory to `$PATH` and set the environment variables.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export XCFLAGS="-march=rv[32|64]v5[d]"
$ export XLFLAGS="-march=rv[32|64]v5[d]"
```

In the above commands, variables “XCFLAGS” and “XLFLAGS” represent extra cflags and ldflags apart from preconfigured ones for the application.

- Step 2** Extract the source code of the Dhrystone demo application, which is located under `<ANDESIGHT_ROOT>/demo/linux`.

```
$ tar -zxf Dhrystone.tgz
```

- Step 3** Change the current directory to the directory containing the source code of the Dhrystone demo application.

```
$ cd Dhrystone
```

- Step 4** Build the demo application using the make command.

```
$ make CPU_FREQ=60
```

The Dhrystone binary “`dhry-linux.out`” will be generated in `<DHRYSTONE_ROOT>/`.

## Running Dhrystone

**Step 1** Copy the Dhrystone binary “`dhry-linux.out`” to an SD card.

**Step 2** See *AndeSight v5.3 BSP User Manual* to boot up an AndeShape platform.

**Step 3** See Chapter 6 to boot up the Linux kernel.

**Step 4** Mount the SD card containing the demo application on the Linux root file system.

```
$ mount /dev/mmcblk0p1 /mnt
```

**Step 5** Change the current directory to the Linux root file system.

```
$ cd /mnt
```

**Step 6** Run the demo application.

```
$ echo 10000000 | ./dhry-linux.out
```

### 3.4.5. Whetstone

The demo application “Whetstone” is a synthetic benchmark for evaluating the performance of computers. It is placed under `<ANDESIGHT_ROOT>/demo/linux`.

#### Loading Whetstone to the kernel file system

See Section 3.1.2 to copy the Whetstone binary to the Linux root file system.

#### Building Whetstone

- Step 1** Export the appropriate toolchain directory to `$PATH` and set the environment variables.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export XCFLAGS="-march=rv[32|64]v5[d]"
$ export XLFLAGS="-march=rv[32|64]v5[d]"
```

In the above commands, variables “XCFLAGS” and “XLFLAGS” represent extra cflags and ldflags apart from preconfigured ones for the application.

- Step 2** Extract the source code of the Whetstone demo application, which is located under `<ANDESIGHT_ROOT>/demo/linux`.

```
$ tar -zxvf whetstone.tgz
```

- Step 3** Change the current directory to the directory containing the source code of the Whetstone demo application.

```
$ cd whetstone
```

- Step 4** Build the demo application using the make command.

```
$ make
```

The Whetstone binary “whetstone” will be generated in `<WHETSTONE_ROOT>/`.

## Running Whetstone

- Step 1** Copy the Whetstone binary “whetstone” to an SD card.
- Step 2** See *AndeSight v5.3 BSP User Manual* to boot up an AndeShape platform.
- Step 3** See Chapter 6 to boot up the Linux kernel.
- Step 4** Mount the SD card containing the demo application on the Linux root file system.  
`$ mount /dev/mmcblk0p1 /mnt`
- Step 5** Change the current directory to the Linux root file system.  
`$ cd /mnt`
- Step 6** Run the demo application.  
`$ ./whetstone 100000`

### 3.4.6. CoreMark

The demo application “CoreMark” is a benchmark that measures the performance of central processing units (CPU) used in embedded systems. It is placed under

<ANDESIGHT\_ROOT>/demo/linux.

#### Loading CoreMark to the kernel file system

See Section 3.1.2 to copy the CoreMark binary to the Linux root file system.

#### Building CoreMark

- Step 1** Export the appropriate toolchain directory to `$PATH` and set the environment variables.

```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
$ export XCFLAGS="-march=rv[32|64]v5[d]"
$ export XLFLAGS="-march=rv[32|64]v5[d]"
$ export PORT_DIR=[linux|linux64]
```

In the above commands, variables “XCFLAGS” and “XLFLAGS” represent extra cflags and ldflags apart from preconfigured ones for the application.

- Step 2** Extract the source code of the CoreMark demo application, which is located under <ANDESIGHT\_ROOT>/demo/linux.

```
$ tar -zxf CoreMark.tgz
```

- Step 3** Change the current directory to the directory containing the source code of the CoreMark demo application.


```
$ cd CoreMark
```

- Step 4** Build the demo application using the make command.

```
$ make compile
```

The CoreMark binary “`coremark.exe`” will be generated in  
<COREMARK\_ROOT>/.

## Running CoreMark

- 
- Step 1** Copy the CoreMark binary “`coremark.exe`” to an SD card.
- Step 2** See *AndeSight v5.3 BSP User Manual* to boot up an AndeShape platform.
- Step 3** See Chapter 6 to boot up the Linux kernel.
- Step 4** Mount the SD card containing the demo application on the Linux root file system.  
`$ mount /dev/mmcblk0p1 /mnt`
- Step 5** Change the current directory to the Linux root file system.  
`$ cd /mnt`
- Step 6** Run the demo application.  
`$ ./coremark.exe 0x0 0x0 0x66 0 7 1 2000`

## 4. Linux kernel, device drivers and device tree blob

The Linux kernel source is placed in `<LINUX_BOOT>/linux-6.1/` by default. Based on the source, you can build the Linux kernel image including the ELF file for the Linux kernel, `vmlinux`. You will also be able to build a device tree blob (DTB) file for the kernel in case the DTB integrated with your platform bitmap does not work.

---

### NOTE

The building procedure in this section is workable only on a Linux console. It is not possible to build a Linux kernel on a Windows console.

---

### 4.1. Building a Linux kernel

**Step 1** Follow the methods below to set up the operating environment.

1. On the host terminal, set the environment variables.

```
$ export ARCH=riscv CROSS_COMPILE=riscv[32|64]-linux-
```

2. Open the `bashrc` file.

```
$ vi ~/.bashrc
```

3. Append the following line to the bottom of the `bashrc` file to include the appropriate Linux toolchain in the `PATH`.

```
$ export
PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

4. Reload the bash configuration file.

```
$ source ~/.bashrc
```

**Step 2** Change the current directory to where the Linux kernel source resides.

```
$ cd <LINUX_ROOT>/linux-6.1
```

**Step 3** Issue as follows to configure the kernel with an appropriate

configuration file. A kernel configuration file (`.config`) will be generated in `<LINUX_ROOT>/linux-6.1/` following rules of Makefile in `<LINUX_ROOT>/linux-6.1/arch/riscv/`.

```
$ make ae350_rv[32|64]_[up|smp]_defconfig
```

- Step 4** Open the kernel configuration file (`.config`). Modify it by pointing the `CONFIG_INITRAMFS_SOURCE` parameters to the rootfs and `initramfs.devnodes`. For example,

```
CONFIG_INITRAMFS_SOURCE="<RAMDISK_ROOT>/rootfs/disk
<RAMDISK_ROOT>/rootfs/disk/dev/initramfs.devnodes"
```

- Step 5** Build the Linux kernel for the first time to generate files needed for kernel modules:

```
$ make
```

To speed up the build, use a parallel make to leverage all the CPU cores:

```
$ make -j$(nproc)
```

- Step 6** Build the kernel modules as follows:

```
$ make modules
$ make modules_install
INSTALL_MOD_PATH=<RAMDISK_ROOT>/rootfs/disk
```

- Step 7** Build the Linux kernel again to use the updated rootfs with the modules:

```
$ make
```

The kernel image in ELF format, `vmlinux`, is generated under `<LINUX_ROOT>/linux-6.1/` and its raw binary file `Image` is generated under `<LINUX_ROOT>/linux-6.1/arch/riscv/boot/`.



## 4.2. Building device drivers

The following table lists the Andes Linux drivers for AE350, the driver paths, and their corresponding IPs.

Table 3. Andes Linux drivers, driver paths, and corresponding IPs and kernel configuration names

| Device              | Device Driver Path                                                                                                    | IP Name    | Kernel Configuration Name(s)                                                                                                  |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------|
| DMA                 | <code>drivers/soc/andes/atcdmac300.c</code>                                                                           | ATCDMAC300 | CONFIG_PLATFORM_AHBDMA<br>CONFIG_PLAT_AE350<br>CONFIG_ATCDMAC300                                                              |
| DMA<br>(DMA engine) | <code>drivers/dma/atcdmac300g.c</code><br><code>drivers/dma/atcdmac300g.h</code>                                      | ATCDMAC300 | CONFIG_ATCDMAC300G                                                                                                            |
| GPIO                | <code>drivers/gpio/gpio-atcgpio100.c</code>                                                                           | ATCGPIO100 | CONFIG_GPIO_ATCGPIO100                                                                                                        |
| I2C                 | <code>drivers/i2c/busses/i2c-atciic100.c</code>                                                                       | ATCIIC100  | CONFIG_I2C_ATCIIC100                                                                                                          |
| LCD                 | <code>drivers/video/fbdev/ftlcdc100/faradayfb-main.c</code><br><code>drivers/video/fbdev/ftlcdc100/faradayfb.h</code> | ATFLCDC100 | CONFIG_FB_FTLCDC100<br>CONFIG_PANEL_AUA036QN01<br>CONFIG_PANEL_AUA070VW04<br>CONFIG_PANEL_CH7013A<br>CONFIG_PANEL_LW500AC9601 |
| MAC                 | <code>drivers/net/ethernet/faraday/ftmac100.c</code><br><code>drivers/net/ethernet/faraday/ftmac100.h</code>          | ATFMAC100  | CONFIG_NET_VENDOR_FARADAY<br>CONFIG_FTMAC100                                                                                  |
| PWM                 | <code>drivers/pwm/pwm-atcpit100.c</code>                                                                              | ATCPIT100  | CONFIG_PWM_ATCPIT100                                                                                                          |
| RTC                 | <code>drivers/rtc/rtc-atcrtc100.c</code>                                                                              | ATCRTC100  | CONFIG_RTC_DRV_ATCRTC100                                                                                                      |
| SD 2.0              | <code>drivers/mmc/host/ftsdc010.c</code><br><code>drivers/mmc/host/ftsdc010.h</code>                                  | ATFSDC010  | CONFIG_MMC_FTSDC                                                                                                              |

| Device                 | Device Driver Path                                         | IP Name    | Kernel Configuration Name(s)                                                                          |
|------------------------|------------------------------------------------------------|------------|-------------------------------------------------------------------------------------------------------|
| SD 2.0<br>(DMA engine) | drivers/mmc/host/ftsd010g.c<br>drivers/mmc/host/ftsd010g.h | ATFSDC010  | CONFIG_MMC_FTSDCG                                                                                     |
| SOUND                  | sound/v5/FTSSP010_ALSA.c<br>sound/v5/FTSSP010_UDA1345TS.h  | ATFSSP010  | CONFIG_SND_FTSSP010<br>CONFIG_SND_FTSSP010_AC97<br>CONFIG_SND_FTSSP010_I2S<br>CONFIG_SND_FTSSP010_HDA |
| SPI                    | drivers/spi/spi-atcspi200.c                                | ATCSPI200  | CONFIG_SPI_ATCSPI200                                                                                  |
| UART                   | drivers/tty/serial/8250/                                   | ATCUART100 | CONFIG_SERIAL_8250                                                                                    |
| WDT                    | drivers/watchdog/atcwdt200_wdt.c                           | ATCWDT200  | CONFIG_ATCWDT200_WATCHDOG                                                                             |

### 4.3. Building a device tree blob (DTB)

Andes platform bitmaps from AndeSight v5.0.0 and later releases are integrated with a ready-to-use DTB at 0xF2000000 for evaluation purposes. You can skip this section if using the built-in DTB. In case the ready-made DTB doesn't work, follow the instructions in this section to build a DTB for the kernel boot process.

**Step 1** Change the current directory to where the Linux kernel source resides.

```
$ cd <LINUX_ROOT>/linux-6.1/
```

**Step 2** Use the Device Tree Compiler (DTC) to compile the Device Tree Source (DTS) file and generate the DTB file. The DTC binary is available in `<LINUX_ROOT>/linux-6.1/scripts/dtc/dtc` after you build the kernel following Section 4.1. You may also obtain the tool from the Linux distribution repository you use.

The DTS files are placed in `<LINUX_ROOT>/linux-6.1/arch/riscv/boot/dts/andes/` and named as `[CORE]_c[N]_[_d]_[_dsp]_[_noncoherent]_[_PLATFORM].dts` where

- **CORE** represents a specific Andes CPU. Currently, the valid values for **CORE** include `a[x]25[mp]`, `a[x]2712`, `a[x]45[mp|mpv]` and `ax65mp`.
- `_c[N]` represents the number of available cores.
- `_d` is only present for a target with FPU support.
- `_dsp` is only present for a target with DSP support.
- `_noncoherent` is only present for a UP system without a coherent unit.
- `_PLATFORM` represents a specific Andes platform.

Make sure you select a DTS file that matches your target system and follow the same naming rules to build your DTB file.

The following shows building a DTB file for an AE350 target that

is pre-integrated with an AX25 core, FPU and DSP support, and no coherent unit.

```
$./scripts/dtc/dtc -I dts -o
ax25_c1_d_dsp_noncoherent_ae350.dtb arch/riscv/boot/dts/andes/ax
25_c1_d_dsp_noncoherent_ae350.dts
```

Official  
Release

---

## NOTE

If the built-in DTB is used by the kernel and you want to retrieve it for experimental use or inspect the device tree, proceed as follows:

**Step 1** Initiate GDB.

```
$ riscv[32|64]-linux-gdb
```

**Step 2** Enter the IP address of the target system and the port number assigned for ICEman to build the connection between the target and ICE. Then, reset the target system and hold the processor.

```
(gdb) target remote <TARGET_IP>:<PORT_NUMBER>
```

```
(gdb) reset-and-hold
```

**Step 3** Use the `dump` command to save the DTB content to a file (e.g., `andes.dtb` in this example).

```
(gdb) dump memory andes.dtb 0xf2000000 0xf2004000
```

**Step 4** Decompile the newly saved file to a DTS file.

```
$ dtc -I dtb -O dts andes.dtb -o andes.dts
```

---

## 5. Firmware and Bootloader

### 5.1. OpenSBI (Supervisor Binary Interface)

The RISC-V Open Source Supervisor Binary Interface (OpenSBI) is an implementation of the RISC-V SBI specifications. It provides a supervisor execution environment in M-mode. It serves as a firmware that provides interfaces for S-mode software to communicate with the hardware. For more information about OpenSBI, please see <https://github.com/riscv/opensbi>.

To build OpenSBI images, proceed as follows:

**Step 1** Change the current directory to where the OpenSBI source resides.

```
$ cd <LINUX_ROOT>/opensbi/
```

**Step 2** Build an OpenSBI binary for the AE350 target system. Make sure that you follow below to specify the target architecture and ABI with flags “PLATFORM\_RISCV\_ISA” and “PLATFORM\_RISCV\_ABI” for the build.

- Build for 32-bit systems with soft-float ABI:  
`PLATFORM_RISCV_ISA=rv32v5 PLATFORM_RISCV_ABI=ilp32`
- Build for 32-bit systems with hard-float ABI:  
`PLATFORM_RISCV_ISA=rv32v5d PLATFORM_RISCV_ABI=ilp32d`
- Build for 64-bit systems with soft-float ABI:  
`PLATFORM_RISCV_ISA=rv64v5 PLATFORM_RISCV_ABI=lp64`
- Build for 64-bit systems with hard-float ABI:  
`PLATFORM_RISCV_ISA=rv64v5d PLATFORM_RISCV_ABI=lp64d`

The following illustrates a soft-float build of OpenSBI that can be booted from RAM on a 64-bit target system with four cores and generates firmware images under the directory

```
<LINUX_ROOT>/opensbi/build/platform/generic/firmware/.
```

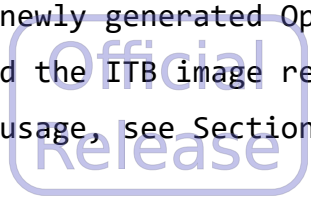
```
$ export PATH=<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/bin:$PATH
```

```
$ export CROSS_COMPILE=riscv64-linux-
```

```
$ make distclean
```

```
$ make PLATFORM=generic PLATFORM_RISCV_ISA=rv64v5
 PLATFORM_RISCV_ABI=lp64
```

The newly generated OpenSBI image `fw_dynamic.bin` will be used to build the ITB image required for the kernel boot. For more about its usage, see Section 5.2.1 and Section 5.2.2.



### 5.2. Das U-Boot

U-Boot is an open-source, cross-platform boot loader applicable to several embedded boards and CPU architectures. Its source is placed under the directory `<LINUX_ROOT>/u-boot` by default and the build process requires a host GCC 4.8.1 or later version. For more information about U-Boot, see <http://www.denx.de/wiki/U-Boot>.



U-Boot SPL (secondary program loader) and U-Boot proper are two compiled products of the U-Boot bootloader. U-Boot SPL is the first stage of the bootstrapping. With a minimum set of code, it works with limited initial storage by only initializing the necessary device (e.g., DRAM).

In a regular booting process (referred to as Normal Boot hereafter), U-Boot SPL copies the whole U-Boot proper image to the initialized RAM and then jumps to the entry of U-Boot proper before booting the kernel.

A faster booting process (referred to as Fast Boot afterward) is also supported. Using a flow like the U-Boot Falcon Mode, it reduces the time spent in the bootloader by skipping the loading and initialization of U-Boot proper completely. U-Boot SPL in this process copies the whole Linux image to the initialized RAM and then jumps to start the Linux kernel directly. With a shorter booting time, the Fast Boot process is best suited for developments that concern the booting performance.

Please follow the instructions in Section 5.2.1 to build the U-Boot SPL and ITB file for the Normal Boot process or Section 5.2.2 to build files for Fast Boot. If you encounter any problems when developing with U-Boot, contact Andes technical support staff for assistance and/or an up-to-date patch.

---

#### NOTE

The U-Boot in AndeSight v5.3 has a critical dependency on OpenSSL. Building the U-Boot on CentOS 7 will cause dependency trouble or linker errors as the OpenSSL version offered by the OS, 1.0.2k, is too old. To resolve the problems, please follow below to install the verified OpenSSL version on CentOS 7, OpenSSL 1.1.1q, first.

**Step 1** Obtain the OpenSSL 1.1.1q tarball.

```
$ curl https://www.openssl.org/source/openssl-1.1.1q.tar.gz >
openssl-1.1.1q.tar.gz
```

**Step 2** Decompress the tarball and change the current directory to the extracted folder.

```
$ tar xvf openssl-1.1.1q.tar.gz
$ cd openssl-1.1.1q
```

**Step 3** Configure, build and install the OpenSSL package on CentOS 7.

```
$./Configure linux-x86_[32|64]
$ make && make install
```

**Step 4** Export the OpenSSL installation path to environment variables `LIBRARY_PATH` and `LD_LIBRARY_PATH` before you proceed with Section 5.2.1 or 5.2.2.

```
$ export LIBRARY_PATH=/usr/local/lib[64]
$ export LD_LIBRARY_PATH=/usr/local/lib[64]
```

---



## 5.2.1. Building U-Boot SPL and ITB file for Normal Boot

The Normal Boot process requires a U-Boot SPL image and a U-Boot proper image `u-boot.itb`. Depending on how you boot the U-Boot SPL, different U-Boot SPL images will be used. To boot it from RAM, use `u-boot-spl`; to boot it from flash, use `u-boot-spl.bin` instead. The following steps explain how to compile and obtain these files.

**Step 1** Change the current directory to where the U-Boot source resides.  
`$ cd <LINUX_ROOT>/u-boot/`

**Step 2** Build a U-Boot SPL binary and `u-boot.itb` for your AE350 target system. This involves the following:

1. Copy the dynamic mode OpenSBI binary (i.e., `fw_dynamic.bin` in Section 5.1) to the U-Boot directory.

```
$ cp
 <LINUX_ROOT>/opensbi/build/platform/generic/firmware/fw_dyn
 amic.bin <LINUX_ROOT>/u-boot/.
```

2. Follow below to select a default configuration file for Normal Boot.

|                 | Normal mode:<br>Booting U-Boot SPL from RAM | XIP (eXecution In Place) mode:<br>Booting U-Boot SPL from flash |
|-----------------|---------------------------------------------|-----------------------------------------------------------------|
| 32-bit<br>cores | <code>ae350_rv32_spl_defconfig</code>       | <code>ae350_rv32_spl_xip_defconfig</code>                       |
| 64-bit<br>cores | <code>ae350_rv64_spl_defconfig</code>       | <code>ae350_rv64_spl_xip_defconfig</code>                       |

For example, to boot U-Boot SPL from RAM on a 64-bit CPU target system for Normal Boot, issue as follows to select the corresponding defconfig file:

```
$ make ae350_rv64_spl_defconfig
```

3. If you are going to boot the kernel from a TFTP server (see

Section 6.1.2.3) for a bitmap of 25-series core with the non-coherent system (i.e., UP system without a coherent unit), please disable d-cache as follows to ensure data transaction integrity for MAC driver. Otherwise, jump to the next.

`$ make menuconfig`  
     RISC-V architecture --->  
     [\*] Do not enable dcache  
     [\*] Do not enable dcache in SPL

4. If you are going to boot the U-Boot SPL in XIP mode and hoping to boot the U-Boot proper from MMC, configure the features to be built as follows. Otherwise, jump to the next.

`$ make menuconfig`  
     SPL configuration options --->  
     [\*] Support FAT filesystems  
     [\*] Support MMC  
     [ ] Support booting from RAM  
     (0x1) Scratch options passed to OpenSBI

OpenSBI firmware information will be provided for inspection when the configuration option “Scratch options passed to OpenSBI” is set to its default value 0x0. To hide and not print out the information, just change the value to 0x1.

5. Use the flag “ARCH\_FLAGS” to specify the target architecture for the build. If a v5 toolchain is used, apply the option “-march=rv[32|64]v5”; for a v5d toolchain, apply “-march=rv[32|64]v5d”. The following example is to use a v5 toolchain for a 64-bit system.

`$ make ARCH_FLAGS="-march=rv64v5"`

6. Confirm that the ELF format images of U-Boot SPL “u-boot-spl” and “u-boot-spl.bin” are generated in <LINUX\_ROOT>/u-boot/spl/ and the ITB file of U-Boot “u-boot.itb” is in <LINUX\_ROOT>/u-boot.

```
$ ls <LINUX_ROOT>/u-boot/spl/u-boot-spl \
 <LINUX_ROOT>/u-boot/spl/u-boot-spl.bin \
 <LINUX_ROOT>/u-boot/u-boot.itb
```

The following presents the entire building procedure for Normal Boot. The example uses a v5 toolchain to build `u-boot.itb` and a U-Boot SPL binary that will be booted from RAM on a 64-bit 45 CPU target system with an SMP core.

```
$ make distclean
$ cp
 <LINUX_ROOT>/opensbi/build/platform/generic/firmware/fw_dynami
 c.bin <LINUX_ROOT>/u-boot/.
$ make ae350_rv64_spl_defconfig
$ make ARCH_FLAGS="-march=rv64v5"
$ ls <LINUX_ROOT>/u-boot/spl/u-boot-spl \
 <LINUX_ROOT>/u-boot/spl/u-boot-spl.bin \
 <LINUX_ROOT>/u-boot/u-boot.itb
```

---

## NOTE

To boot the U-Boot proper from MMC, be sure to copy the generated `u-boot.itb` to the root directory of an SD card for the booting process.

---

## 5.2.2. Building U-Boot SPL and ITB file for Fast Boot

The Fast Boot process is designed to speed up the booting procedure by skipping the execution of U-Boot proper. This process requires the presence of `linux.itb` and a U-Boot SPL image.

The specific U-Boot SPL image required can be either `u-boot-spl` or `u-boot-spl.bin`, depending on whether the U-Boot SPL is booted from RAM or flash, respectively. The following steps explain how to compile and obtain these files.

**Step 1** Change the current directory to where the U-Boot source resides.

```
$ cd <LINUX_ROOT>/u-boot/
```

**Step 2** Build a U-Boot SPL binary and `linux.itb` for your AE350 target system. This involves the following:

1. Copy the dynamic mode OpenSBI binary (i.e., `fw_dynamic.bin` in Section 5.1) and Linux kernel image (i.e., `Image` in Section 4.1) to the U-Boot directory.

```
$ cp
 <LINUX_ROOT>/opensbi/build/platform/generic/firmware/fw_dynamic.bin <LINUX_ROOT>/u-boot/.
$ cp <LINUX_ROOT>/linux-6.1/arch/riscv/boot/Image
 <LINUX_ROOT>/u-boot/.
```

2. Follow below to select a default configuration file for Fast Boot.

|                 | Normal mode:<br>Booting U-Boot SPL from RAM | XIP (eXecution In Place) mode:<br>Booting U-Boot SPL from flash |
|-----------------|---------------------------------------------|-----------------------------------------------------------------|
| 32-bit<br>cores | <code>ae350_rv32_fastboot_defconfig</code>  | <code>ae350_rv32_fastboot_xip_defconfig</code>                  |
| 64-bit<br>cores | <code>ae350_rv64_fastboot_defconfig</code>  | <code>ae350_rv64_fastboot_xip_defconfig</code>                  |

For example, to boot U-Boot SPL from RAM on a 64-bit CPU

target system for Fast Boot, issue as follows to select the corresponding defconfig file:

```
$ make ae350_rv64_fastboot_defconfig
```

3. If you are going to boot the U-Boot SPL in XIP mode and hoping to boot Linux from MMC, configure the features to be built as follows. Otherwise, jump to the next.

```
$ make menuconfig
```

```
SPL configuration options --->
```

```
[*] Support FAT filesystems
```

```
[*] Support MMC
```

```
[] Support booting from RAM
```

```
(0x1) Scratch options passed to OpenSBI
```

OpenSBI firmware information will be provided for inspection when the configuration option “Scratch options passed to OpenSBI” is set to its default value 0x0. To hide and not print out the information, just change the value to 0x1.

4. Use the flag “ARCH\_FLAGS” to specify the target architecture for the build. If a v5 toolchain is used, apply the option “-march=rv[32|64]v5”; for a v5d toolchain, apply “-march=rv[32|64]v5d”. The following example is to use a v5 toolchain for a 64-bit system.

```
$ make ARCH_FLAGS="-march=rv64v5"
```

5. Confirm that the ELF format images of U-Boot SPL “u-boot-spl” and “u-boot-spl.bin” are generated in <LINUX\_ROOT>/u-boot/spl/ and the ITB file of kernel “linux.itb” is in <LINUX\_ROOT>/u-boot.

```
$ ls <LINUX_ROOT>/u-boot/spl/u-boot-spl \
 <LINUX_ROOT>/u-boot/spl/u-boot-spl.bin \
 <LINUX_ROOT>/u-boot/linux.itb
```

The following presents the entire building procedure for Fast

Boot. The example uses a v5 toolchain to build `linux.itb` and a U-Boot SPL binary that will be booted from RAM on a 64-bit target system with SMP core.

```
$ make distclean
$ cp <LINUX_ROOT>/opensbi/build/platform/generic/firmware/fw_dynami
c.bin <LINUX_ROOT>/u-boot/.
$ cp <LINUX_ROOT>/linux-6.1/arch/riscv/boot/Image
<LINUX_ROOT>/u-boot/.
$ make ae350_rv64_fastboot_defconfig
$ make ARCH_FLAGS="-march=rv64v5"
$ ls <LINUX_ROOT>/u-boot/spl/u-boot-spl \
<LINUX_ROOT>/u-boot/spl/u-boot-spl.bin \
<LINUX_ROOT>/u-boot/linux.itb
```

---

### NOTE

To boot the kernel from MMC, be sure to copy the generated `linux.itb` to the root directory of an SD card for the booting process.

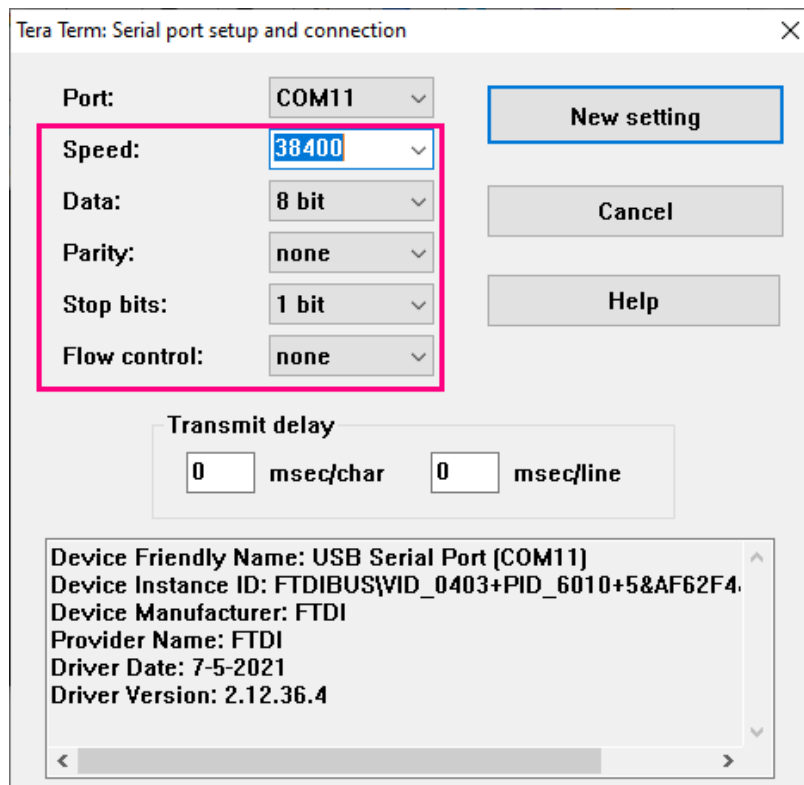
---

## 6. Booting Linux kernel

A successful Linux kernel booting involves several processes. It first requires a bootloader to correctly initialize the necessary devices and then load the kernel into the memory so that the kernel can be properly executed on the CPU. The following Sections 6.1 and Section 6.2 describe detailed steps to boot up a kernel in a regular and faster process. Before that, you will need to follow the steps below to set up an ICE target first .

**Step 1** Prepare a real evaluation platform (REP) and connect it to an ICE device.

**Step 2** Open a serial terminal program, such as Tera Term, select the COM port to be connected and configure COM port settings as follows:



**Step 3** Toggle on the power switch of the EVB. Ensure that the on-board LED is illuminated and press the “PWR ON” button.

**Step 4** Initiate the stand-alone Andes ICE controller program ICEman and specify an available port from 0 to 65535 for GDB connection.

```
$./ICEman.exe --port <PORT_NUMBER>
```

If your target is integrated with a Symmetric Multiprocessing (SMP) core, make sure you also apply the `--smp` option as follows:

```
$./ICEman.exe --port <PORT_NUMBER> --smp
```

---

### NOTE

The values presented in the following subsections (e.g., file addresses, file sizes, and the start address for flash) are for reference purposes only and will vary according to the target and development environment.

---



## 6.1. Normal Boot

OpenSBI in dynamic mode is suitable for product-oriented services and can be burnt onto permanent storage on targets. It works with U-Boot SPL to boot the kernel. During the kernel boot process, U-Boot SPL starts first to set up all necessary devices and then transfers the control over to OpenSBI in dynamic mode to register M-mode interfaces. In a Normal Boot process, the OpenSBI will jump back to U-Boot waiting to execute commands.

U-Boot has to be booted before the kernel boot from a Flash, a TFTP server, or an SD card with OpenSBI in dynamic mode. There are two U-Boot boot approaches to cater to different scenarios. The first is to load U-Boot SPL to RAM and boot U-Boot via GDB for development or debugging purposes. The other is for the production environment and it requires burning U-Boot SPL to flash first and booting U-Boot there after resetting the target board. The two approaches are described in Section 6.1.1.1 and 6.1.1.2 respectively.

To burn U-Boot SPL, you can either use the U-Boot `sf` command or the Andes flash burner “SPI\_burn” with ICEman. Compared with the other method, the burning using the Andes flash burner can be performed on a target without U-Boot running, but it takes more time as the image needs to be transferred via the USB interface.

As to the kernel boot, make sure to use the command `bootm` (rather than `go`) to boot the kernel, because the `go` command is used only for legacy boot and does not pass the device tree blob (DTB).

### 6.1.1. Booting U-Boot

#### 6.1.1.1 Loading U-Boot SPL to RAM and booting U-Boot via GDB

**Step 1** Open a terminal program. Initiate GDB and specify the ELF format image of U-Boot SPL “`u-boot-spl`” under `<LINUX_ROOT>/u-boot/spl/`.

```
$ riscv[32|64]-linux-gdb u-boot-spl
```

**Step 2** Enter the IP address of the target system and the port number assigned for ICEman to build the connection between the target

and ICE. Then, reset the target system and hold the processor.  
For example,

```
(gdb) target remote <TARGET_IP>:<PORT_NUMBER>
```

```
(gdb) reset-and-hold
```

**Step 3** Load the U-Boot SPL program.

```
(gdb) load
```

**Step 4** Load the U-Boot ITB file `u-boot.itb` from `<LINUX_ROOT>/u-boot/`.  
For example,

```
(gdb) restore u-boot.itb binary 0x10000000
```

**Step 5** Jump to the next step if the built-in DTB is in use. Otherwise, load your manually built DTB file (see Section 4.3) to RAM via GDB. For example,

```
(gdb) restore <DTB_FILE> binary 0x20000000
```

**Step 6** When the built-in DTB is in use, issue GDB commands to set the value of the registers `$pc`, `$a0` and `$a1` respectively to the entry, hart ID and the default DTB address `0xf2000000` and apply them to all the threads.

```
(gdb) thread apply all set $pc=&_start
```

```
(gdb) thread apply all set $a0=$mhartid
```

```
(gdb) thread apply all set $a1=0xf2000000
```

If you are using a manually built DTB, modify the last GDB command by setting `$a1` to the DTB address you specified in Step 5.

```
(gdb) thread apply all set $a1=0x20000000
```

**Step 7** Execute the U-Boot program.

```
(gdb) continue
```

## 6.1.1.2 Burning U-Boot SPL to flash and booting U-Boot

### Burning U-Boot SPL via U-Boot's sf command

**Step 1** Copy the U-Boot SPL image `u-boot-spl.bin` from `<LINUX_ROOT>/u-boot/spl/`, `u-boot.itb` from `<LINUX_ROOT>/u-boot/` and DTB file to an SD card.

**Step 2** Follow the instructions in Section 6.1.1.1.

**Step 3** Load the U-Boot SPL image `u-boot-spl.bin` from the SD card to RAM on the target system. Then use a series of `sf` commands to write the image to the SPI flash on the target system.

```
RISC-V # fatload mmc 0:1 0x600000 u-boot-spl.bin
RISC-V # sf probe 0:0 50000000 0
RISC-V # sf erase 0x0 0x10000
RISC-V # sf write 0x600000 0x0 0x10000
```

For details about the uses of U-Boot commands to program SPI flash, type `sf` in the U-Boot prompt.

**Step 4** Skip this step if you have configured the U-Boot proper to be booted from MMC (see Section 5.2.1, Step 2-4). Otherwise, use the same commands in Step 3 to burn the U-Boot image `u-boot.itb` from RAM to the flash of the target.

```
RISC-V # fatload mmc 0:1 0x600000 u-boot.itb
RISC-V # sf probe 0:0 50000000 0
RISC-V # sf erase 0x10000 0xa0000
RISC-V # sf write 0x600000 0x10000 0xa0000
```

**Step 5** Use the same commands in Step 3 to burn the corresponding DTB file from RAM to the flash of the target.

```
RISC-V # fatload mmc 0:1 0x20000000 <DTB_FILE>
RISC-V # sf probe 0:0 50000000 0
```

```
RISC-V # sf erase 0xf0000 0x10000
RISC-V # sf write 0x20000000 0xf0000 0x10000
```

**Step 6** Restart the target system by toggling the power switch off and on. Press the PWR ON button and check the U-Boot messages in the console. U-Boot is now ready for booting the Linux kernel.

Official  
Release

## Burning U-Boot SPL using Andes flash burner “SPI\_burn”

See the chapter “In-system programming” in *AndeSight v5.3 BSP User Manual* for SPI\_burn usages and use the utility to burn the U-Boot SPL image `u-boot-spl.bin`, the U-Boot image `u-boot.itb` and its corresponding DTB file to the flash of the target.

## 6.1.2. Booting Linux kernel with U-Boot

### 6.1.2.1 Booting via GDB

**Step 1** Flow instructions in Section 5.2.1 and Section 6.1.1 to build U-Boot SPL and boot U-Boot.

**Step 2** Issue as follows to load the Linux kernel file “`Image`” using GDB. If GDB is busy running at the moment, press Ctrl-C to halt the current action first.

```
(gdb) restore <LINUX_ROOT>/linux-6.1/arch/riscv/boot/Image
binary 0x02000000
(gdb) continue
```

**Step 3** Boot the Linux Kernel from RAM.

```
RISC-V # booti 0x02000000 - $fdtcontroladdr
```

---

#### NOTE

If you plan to debug with gdbserver afterwards, be sure to terminate the ICEman and GDB used to load the kernel after the boot process is completed.

---

### 6.1.2.2 Booting from flash

**Step 1** Follow instructions in Section 5.2.1 and Section 6.1.1 to build U-Boot SPL and boot U-Boot.

**Step 2** Remove the meta data in `vmlinux` and generate a raw binary file of the Linux kernel.

```
$ riscv[32|64]-linux-objcopy -O binary vmlinux vmlinux.bin
```

**Step 3** Change the current directory to the “tools” directory of the U-Boot package “u-boot” and use the `mkimage` utility to convert the raw binary kernel `vmlinux.bin` into a format compatible with U-Boot. For example, create an image `bootm-vmlinux.bin` as follows:

- To convert a 64-bit `vmlinux` binary

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x200000 -e 0x200000 -d vmlinux.bin bootm-vmlinux.bin
```

- To convert a 32-bit `vmlinux` binary

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x400000 -e 0x400000 -d vmlinux.bin bootm-vmlinux.bin
```

**Step 4** Copy the newly-generated Linux kernel `bootm-vmlinux.bin` to an SD card.

**Step 5** Reset your target system and check the U-Boot messages in the console.

**Step 6** Load the kernel image from the SD card to the parallel flash of your target. For example,

```
RISC-V # fatload mmc 0:1 0x600000 bootm-vmlinux.bin
RISC-V # protect off all; erase 0x88000000 0x8bffffff;
RISC-V # cp.b 0x600000 0x88600000 0x2000000
```

Note that the values marked in red above indicate the length (byte count) and must be modified according to the size of your kernel image.

**Step 7** Set environment variables and boot the kernel image from the parallel flash.

```
RISC-V # setenv bootm_size 0x20000000;setenv fdt_high
0x1f000000;
RISC-V # bootm 0x88600000 - $fdtcontroladdr
```

### 6.1.2.3 Booting from a TFTP server

**Step 1** Follow instructions in Section 5.2.1 and Section 6.1.1 to build U-Boot SPL and boot U-Boot.

**Step 2** Remove the meta data in `vmlinux` and generate a raw binary file of the Linux kernel.

```
$ riscv[32|64]-linux-objcopy -O binary vmlinux vmlinux.bin
```

**Step 3** Change the current directory to the “tools” directory of the U-Boot package “u-boot” and use the `mkimage` utility to convert the raw binary kernel `vmlinux.bin` into a format compatible with U-Boot. For example, create an image `bootm-vmlinux.bin` as follows:

- To convert a 64-bit `vmlinux` binary

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x200000 -e
0x200000 -d vmlinux.bin bootm-vmlinux.bin
```

- To convert a 32-bit `vmlinux` binary

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x400000 -e
0x400000 -d vmlinux.bin bootm-vmlinux.bin
```

**Step 4** Copy the newly generated Linux kernel `bootm-vmlinux.bin` to a TFTP server.

**Step 5** Load the Linux kernel image and DTB file from the TFTP server to RAM on the target system. For example,

```
RISC-V # setenv bootm_size 0x20000000;setenv fdt_high 0x1f000000;
RISC-V # dhcp 0x600000 10.0.12.145:bootm-vmlinux.bin
```

**Step 6** Boot the Linux kernel from RAM.

```
RISC-V # bootm 0x00600000 - $fdtcontroladdr
```

## 6.1.2.4 Booting from an SD card

**Step 1** Follow instructions in Section 5.2.1 and Section 6.1.1 to build U-Boot SPL and boot U-Boot.

**Step 2** Remove the meta data in vmlinux and generate a raw binary file of the Linux kernel.

```
$ riscv[32|64]-linux-objcopy -O binary vmlinux vmlinux.bin
```

**Step 3** Change the current directory to the “tools” directory of the U-Boot package “u-boot” and use the mkimage utility to convert the raw binary kernel vmlinux.bin to a format compatible with U-Boot. For example, create an image bootm-vmlinux.bin as follows:

- To convert a 64-bit vmlinux binary
 

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x200000 -e 0x200000 -d vmlinux.bin bootm-vmlinux.bin
```
- To convert a 32-bit vmlinux binary
 

```
$./mkimage -A riscv -O linux -T kernel -C none -a 0x400000 -e 0x400000 -d vmlinux.bin bootm-vmlinux.bin
```

**Step 4** Copy the newly-generated Linux kernel bootm-vmlinux.bin to an SD card.

**Step 5** Scan the SD card.

```
RISC-V # mmc rescan
```

**Step 6** List the partition(s) on the SD card.

```
RISC-V # mmc part
```

**Step 7** List the files on the SD card.

```
RISC-V # fatls mmc 0:1
```

**Step 8** Load the Linux kernel image and DTB file from the SD card to RAM on the target system. For example,

```
RISC-V # setenv bootm_size 0x20000000;setenv fdt_high
0x1f000000;
```

```
RISC-V # fatload mmc 0:1 0x600000 bootm-vmlinux.bin
```

**Step 9** Boot the Linux kernel from RAM.

```
RISC-V # bootm 0x00600000 - $fdtcontroladdr
```



## 6.2. Fast Boot

In the hope of reducing the boot time, Andes U-Boot is implemented with a feature similar to U-Boot Falcon Mode. The boot flow using this feature, called Fast Boot, initializes memory with the U-Boot SPL at the first stage, just like what a regular booting process (i.e. Normal Boot) does in the beginning. Instead of jumping to the U-Boot proper from OpenSBI before booting the kernel, the Fast Boot process jumps directly to the kernel to enable a shorter boot time.

There are two different methods to perform Fast Boot. The first is to load U-Boot SPL and kernel to RAM and boot kernel via GDB for development or debugging purposes. The other is for the production environment and it requires burning U-Boot SPL and kernel to flash first and booting kernel there after resetting the target board. The two methods are described in Section 6.2.1 and 6.2.2 respectively.

---

### NOTE

The maximum kernel size for Fast Boot is default set to 32MB. This is determined by the U-Boot parameter `CONFIG_SPL_SYS_MALLOC_F_LEN`, which configures the SPL memory allocation and has a default value of `0x2000000`. If you need to load a larger kernel, find this parameter and associated configurations `CONFIG_TEXT_BASE`, which defines the start address of kernel `Image`, and `CONFIG_CUSTOM_SYS_INIT_SP_ADDR`, which defines the initial stack pointer address, in `<LINUX_ROOT>/u-boot/configs/ae350_rv[32|64]_fastboot[_xip]_defconfig` and modify them for appropriate memory layouts following below:

1. `CONFIG_SPL_SYS_MALLOC_F_LEN` must be larger than the size of your kernel `Image`.
2. “`CONFIG_TEXT_BASE` + size of your kernel `Image`” must not exceed  
“`CONFIG_CUSTOM_SYS_INIT_SP_ADDR` - `CONFIG_SPL_SYS_MALLOC_F_LEN` - 1MB”

You may also reference memory layouts for the default U-Boot SPL, illustrated below, to make the modifications.

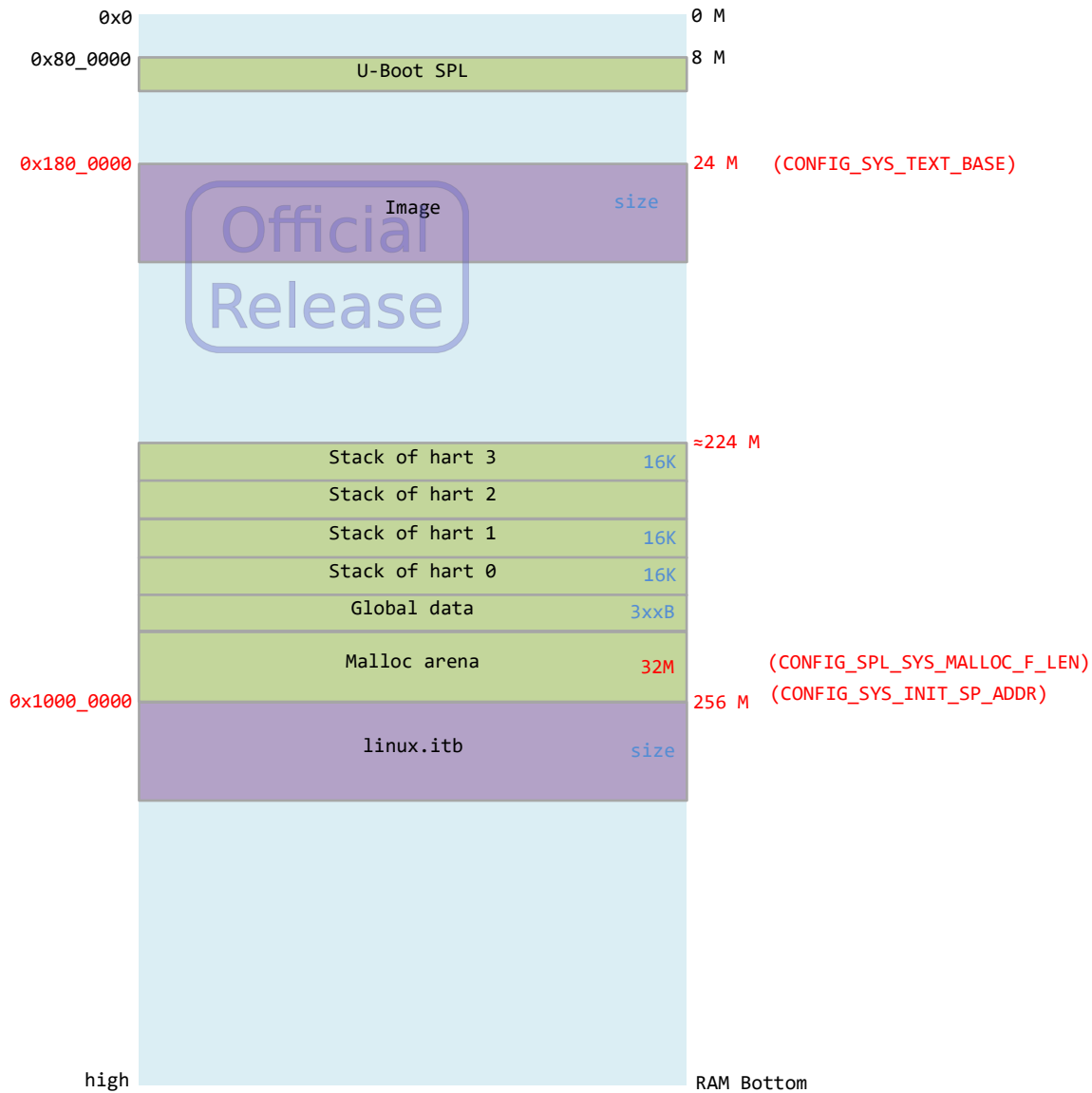


Figure 1. Memory layout for the default U-Boot SPL

### 6.2.1. Loading U-Boot SPL, kernel to RAM and booting kernel

**Step 1** Open a terminal program. Initiate GDB and specify the ELF format image of U-Boot SPL “u-boot-spl” under `<LINUX_ROOT>/u-boot/spl/`.

```
$ riscv[32|64]-linux-gdb u-boot-spl
```

**Step 2** Enter the IP address of the target system and the port number assigned for ICEman to build the connection between the target and ICE. Then, reset the target system and hold the processor. For example,

```
(gdb) target remote <TARGET_IP>:<PORT_NUMBER>
```

```
(gdb) reset-and-hold
```

**Step 3** Load the U-Boot SPL program.

```
(gdb) load
```

**Step 4** Load the kernel ITB file `linux.itb` from `<LINUX_ROOT>/u-boot/`. For example,

```
(gdb) restore linux.itb binary 0x10000000
```

**Step 5** Jump to the next step if the built-in DTB is in use. Otherwise, load your manually built DTB file (see Section 4.3) to RAM via GDB. For example,

```
(gdb) restore <DTB_FILE> binary 0x20000000
```

**Step 6** When the built-in DTB is in use, issue GDB commands to set the value of the registers `$pc`, `$a0` and `$a1` respectively to the entry, hart ID and the default DTB address `0xf2000000` and apply them to all the threads.

```
(gdb) thread apply all set $pc=&_start
```

```
(gdb) thread apply all set $a0=$mhartid
```

```
(gdb) thread apply all set $a1=0xf2000000
```

If you are using a manually built DTB, modify the last GDB command by setting `$a1` to the DTB address you specified in Step 5.

```
(gdb) thread apply all set $a1=0x20000000
```

**Step 7** Execute the kernel.

```
(gdb) continue
```

---

### NOTE

If you plan to debug with gdbserver afterward, be sure to terminate the ICEman and GDB used to load the kernel after the boot process is completed.

---

### 6.2.2. Burning U-Boot SPL, kernel to flash and booting kernel

**Step 1** Copy the U-Boot SPL image `u-boot-spl.bin` from `<LINUX_ROOT>/u-boot/spl/`, `linux.itb` from `<LINUX_ROOT>/u-boot/` and DTB file to an SD card.

**Step 2** Follow the instructions in Section 6.1.1.1.

**Step 3** Load the U-Boot SPL image `u-boot-spl.bin` from the SD card to RAM on the target system. Then use a series of `sf` commands to write the image to the SPI flash on the target system.

```
RISC-V # fatload mmc 0:1 0x600000 u-boot-spl.bin
RISC-V # sf probe 0:0 50000000 0
RISC-V # sf erase 0x0 0x10000
RISC-V # sf write 0x600000 0x0 0x10000
```

For details about the uses of U-Boot commands to program SPI flash, type `sf` in the U-Boot prompt.

**Step 4** Use the same commands in Step 3 to burn the corresponding DTB file from RAM to the flash of the target.

```
RISC-V # fatload mmc 0:1 0x20000000 <DTB_FILE>
RISC-V # sf probe 0:0 50000000 0
RISC-V # sf erase 0xf0000 0x10000
RISC-V # sf write 0x20000000 0xf0000 0x10000
```

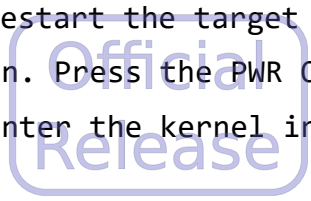
**Step 5** Burn the kernel ITB file `linux.itb` from the SD card to the parallel flash of your target. For example,

```
RISC-V # fatload mmc 0:1 0x600000 linux.itb
RISC-V # protect off all; erase 0x88000000 0x8bffffff
RISC-V # cp.b 0x600000 0x88600000 0x1800000
```

Note that the values marked in red above indicate the length

(byte count) and must be modified according to the size of your kernel image.

- Step 6** Restart the target system by toggling the power switch off and on. Press the PWR ON button and check the console. You will enter the kernel in a short while.



## 7. Building connection between a Linux target and a debug host

The illustration below shows a network between the host and a Linux target system.

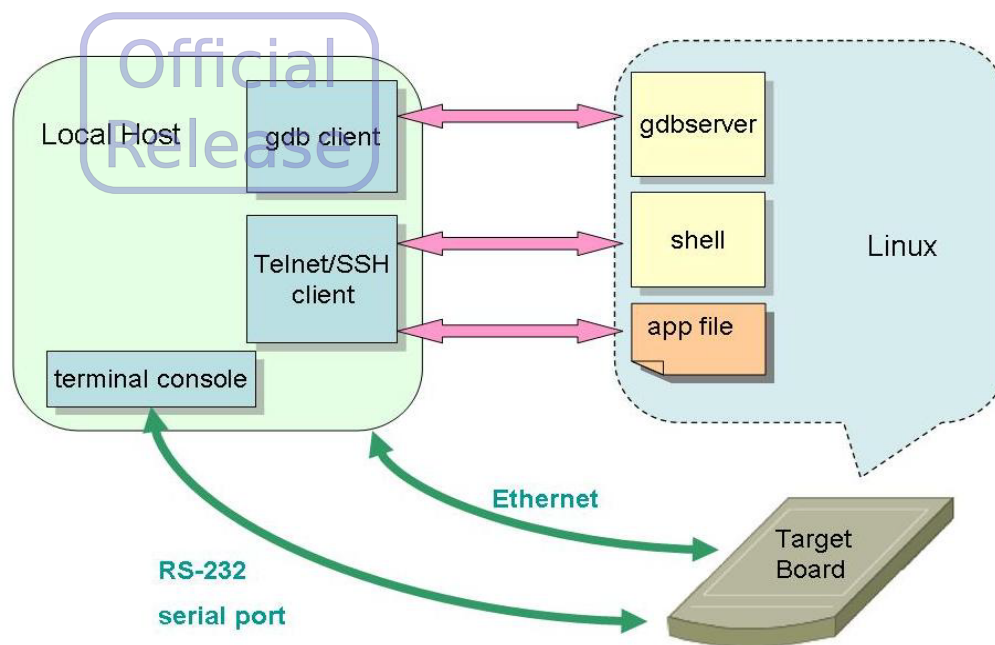


Figure 2. Network between host and Linux target

### 7.1. Setting up a Linux target

The following is a step-by-step guide to the preparation of a Linux-based target for debugging via GDB. If you already set up the target, know its IP address and port number, and initiate the gdbserver, you can jump to Section 7.2 to set up the host for debugging.

---

#### NOTE

The AndeSight IDE can detect whether there is an existing Linux-based target over LAN. If one exists, then a target connection can be easily built using the AndeSight GUI. For details, please consult the *AndeSight User Manual*.

---

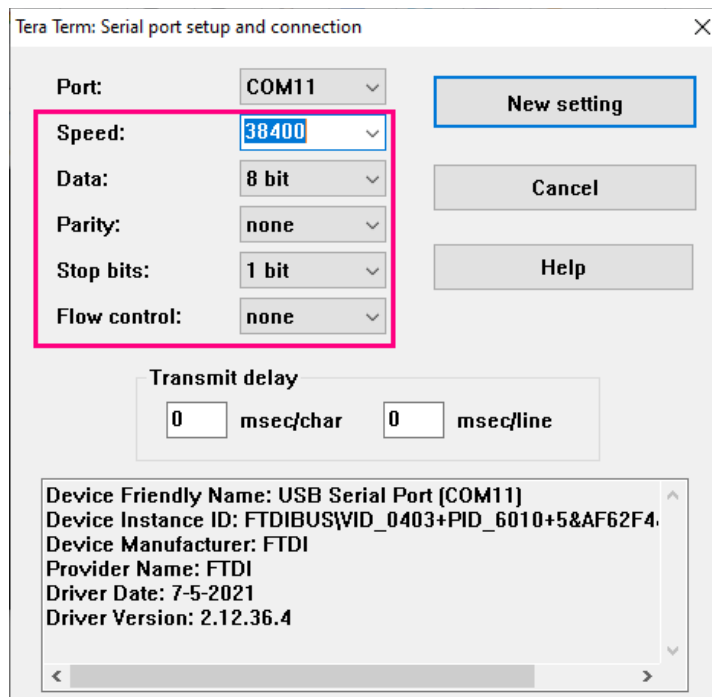
- Step 1** Prepare the program executable (denoted as `<PROJECT_EXECUTABLE>` in the following steps) and follow the methods outlined in Section 3.3.1 to build the gdbserver.

**Step 2** Copy the program executable and gdbserver binary to an SD card or include the two files on the RAM disk following the instructions in Chapter 3.

**Step 3** Follow the methods outlined in Chapter 4 to build the Linux kernel image.

**Step 4** Open a serial terminal program, such as Tera Term, and specify the COM port for connection.

Configure the COM port settings as follows:



**Step 5** Toggle on the power switch of the target board. Ensure that the on-board LED is illuminated and press the “PWR ON” button.

**Step 6** Boot up the Linux kernel following the instructions in Chapter 6.

**Step 7** Follow below to obtain the IP address of the Linux application



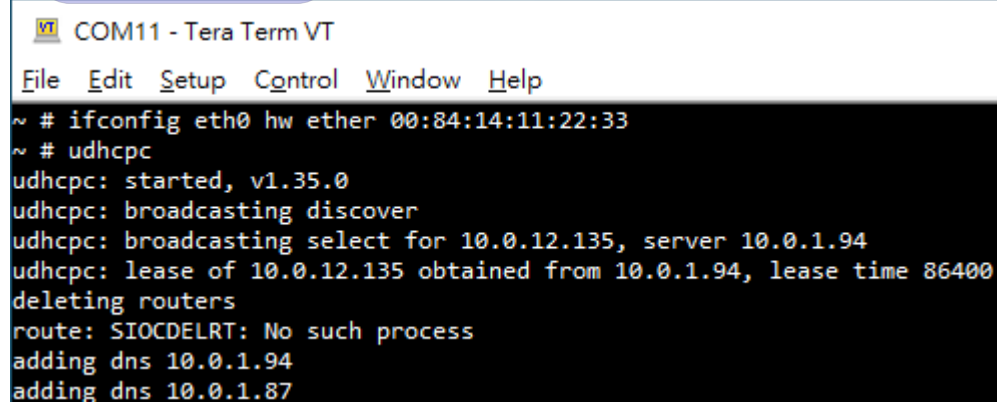
from the DHCP server on the Linux target. In the commands,

<MAC\_ADDR> is the unique MAC address on your LAN.

```
$ ifconfig eth0 hw ether <MAC_ADDR>
```

```
$ udhcpc
```

In the following example, “00:84:14:11:22:33” is used as the MAC address.



```
COM11 - Tera Term VT
File Edit Setup Control Window Help
~ # ifconfig eth0 hw ether 00:84:14:11:22:33
~ # udhcpc
udhcpc: started, v1.35.0
udhcpc: broadcasting discover
udhcpc: broadcasting select for 10.0.12.135, server 10.0.1.94
udhcpc: lease of 10.0.12.135 obtained from 10.0.1.94, lease time 86400
deleting routers
route: SIOCDELRT: No such process
adding dns 10.0.1.94
adding dns 10.0.1.87
```

The IP address of the Linux application is shown to be “10.0.12.135” in this case.

- Step 8** If the program executable and gdbserver binary are loaded onto a RAM disk, then jump to the next step. If the two files are copied to an SD card, then mount the SD card to the Linux target.

```
$ mount /dev/mmcblk0p1 /mnt
```

- Step 9** Please terminate ICEman first if it is connected to the target board. Run gdbserver and specify an available port for the host and the program executable on the Linux target.

```
$ <GDBSERVER_BINARY> :<PORT_NUMBER> <PROJECT_EXECUTABLE>
```

For example:

```
$ gdbserver :1111 ./Hello.out
```

### 7.2. Setting up a debug host

**Step 1** Open a console and initiate GDB on the local host.

```
$ riscv[32|64]-linux-gdb <PROJECT_EXECUTABLE>
```

**Step 2** Locate the target library by setting the system root directory to “sysroot” under the directory of the toolchain currently in use on the host.

```
$ (gdb) set sysroot
/<ANDESIGHT_ROOT>/toolchains/<TOOLCHAIN>/sysroot
```

**Step 3** Enter the IP address of the Linux target and the port number specified in Section 7.1, Step 9 for the host-target connection.

```
$ (gdb) target remote <LINUX_IP>:<PORT_NUMBER>
```

For example,

```
$ (gdb) target remote 10.0.4.86:1111
```

## 8. Linux distributions and customizations

### 8.1. Community distributions

Fedora, Debian, and Ubuntu are the verified Linux distributions for the Andes Linux kernel. To switch to a different Linux distribution, simply unpack the root file system from the prebuilt disk image and run applications in a chroot environment. Currently, the switch between the distributions is only supported on 64-bit systems.

#### 8.1.1. Fedora

**Step 1** Log in to a Linux machine with root permission.

**Step 2** Download a prebuilt Fedora disk image.

```
$ wget https://dl.fedoraproject.org/pub/alt/risc-v/repo/virt-
builder-images/images/Fedora-Developer-Rawhide-20200108.n.0-
sda.raw.xz
```

**Step 3** Use the `unxz` command to decompress the raw disk image.

```
$ unxz Fedora-Developer-Rawhide-*.raw.xz
```

**Step 4** Search for a free loop device, scan the partition table, and mount it.

1. Use the `losetup` command to find the first unused loopback device, scan the partition table, and make the compressed raw disk image associated with the device.

```
$ sudo losetup --show -f -P Fedora-Developer-Rawhide-
20200108.n.0-sda.raw
```

The name of the loopback device that covers the raw disk image will be printed. For example,

```
/dev/loop7
```

2. Use the following command to create a mount point for the partition, where `<DISTRO_ROOT>` is the absolute path to store the rootfs for the distribution.

```
$ sudo mkdir -p <DISTRO_ROOT>
```

3. Use the `fdisk` command to list partition tables on the loopback device to which the disk image is attached and identify the partition that stores the root filesystem. Mount the partition on the mount point. For example,

```
$ sudo fdisk -l /dev/loop7
```

```
$ sudo mount /dev/loop7p4 <DISTRO_ROOT>
```

- Step 5** Change the root directory to a temporary root filesystem, set the password for it, and return to the original environment.

```
$ sudo chroot <DISTRO_ROOT> /bin/bash
```

```
$ echo "root:linux" | chpasswd
```

```
$ exit
```

- Step 6** Follow below to share new rootfs files with an NFS file system or copy the files to an SD card.

- Via NFS

1. Edit the NFS export file “`/etc/exports`” as follows.

```
<DISTRO_ROOT> *(rw,root_squash,no_subtree_check,insecure)
```

2. Re-export the NFS file system to apply the change on the export file.

```
$ sudo exportfs -arv
```

- Via SD card

1. Create a disk image with the `dd` command and format it as an ext4 filesystem.

```
$ dd if=/dev/zero of=rootfs.img bs=1M count=8192
$ mkfs.ext4 -F rootfs.img
```

2. Mount the disk image as a loop device and copy the new rootfs to the disk image.

```
$ sudo mount -o loop rootfs.img /mnt
$ sudo rsync -auH <DISTRO_ROOT>/ * /mnt
$ sudo umount /mnt
```

3. Copy the disk image to an SD card with the `dd` command. The following example assumes that the SD card device is recognized as `"/dev/sdb"` by the system.

```
$ sudo dd if=rootfs.img of=/dev/sdb bs=4M status=progress
```

**Step 7** Remove the original init script that has a symbolic link to `/bin/busybox` and create an init script for the new rootfs using an appropriate code snippet below where `<NFS_SERVER_IP>` is the NFS server you are using and `<SDCARD_DEVICE>` is the name of your SD card device. Both of the code snippets use the `switch_root` command to switch to a different distribution and invoke the command via the `exec` command so that the new init program can inherit PID 1.

- Via NFS

```
#!/bin/sh
Obtain the IP address of the DHCP server
udhcpc
Mount the NFS
NEW_ROOTFS=<NFS_SERVER_IP>:<DISTRO_ROOT>
mount -t nfs -o nolock $NEW_ROOTFS /mnt

Switch to the new rootfs
exec switch_root /mnt /sbin/init
```

- Via SD card

```
#!/bin/sh
mount -t proc proc /proc
mount -t devtmpfs devtmpfs /dev
mount -t sysfs sysfs /sys

Mount the SD card
mkdir -p ./new_root
mount /dev/<SDCARD_DEVICE> ./new_root

Switch to the new rootfs
exec switch_root ./new_root /sbin/init
```

## 8.1.2. Debian

**Step 1** Log in to a Linux machine with root permission.

**Step 2** Download the prebuilt Debian disk image “Images for riscv64-virt” from the webpage <https://people.debian.org/~gio/dqib/>.

**Step 3** Decompress the image

1. Use the `unzip` command to extract the qcow2 format disk image.  
\$ `unzip artifacts.zip`

2. Convert the image to a raw disk image.

```
$ cd artifacts; qemu-img convert -f qcow2 -O raw image.qcow2
debian.img
```

**Step 4** Follow instructions in Section 8.1.1, Step 4~Step 7 to put the Debian rootfs to an NFS server or SD card and switch to the distribution.

### 8.1.3. Ubuntu

**Step 1** Log in to a Linux machine with root permission.

**Step 2** Download a prebuilt Ubuntu disk image.

```
$ wget https://old-releases.ubuntu.com/releases/focal/ubuntu-20.04.4-preinstalled-server-riscv64+unmatched.img.xz
```

**Step 3** Use the `unxz` command to decompress the raw disk image.

```
$ unxz ubuntu-20.04.4-preinstalled-server-*.img.xz
```

**Step 4** Follow instructions in Section 8.1.1, Step 4~Step 7 to put the Ubuntu rootfs to an NFS server or SD card and switch to the distribution.

## 8.2. Build System/SDK

For customization of the Linux distributions, you can use Buildroot or Yocto. The two systems allow you to configure the bootloader, kernel, busybox and user space packages. Andes provides a Buildroot default configuration file and an OpenEmbedded layer in Andes-hosted GitHub [buildroot](https://github.com/andestech/buildroot) and [meta-andes](https://github.com/andestech/meta-andes) repositories at <https://github.com/andestech/>. Following the [README.md](#) on the repositories, you will be able to generate a bootable image that can be copied to an SD card directly without dealing with the partitioning.

## 9. Application notes

### 9.1. Advanced settings

#### 9.1.1. HPM (Hardware Performance Monitor)

HPM (also named PMU, Performance Monitoring Unit) allows you to measure performance metrics of programs. Andes provides an HPM total solution that includes a set of custom CSRs and hart-local interrupt control for the hardware and the porting of Linux perf tool for the software. The following sections highlight software-related configurations. For details of the hardware design, see *AndeStar V5 System Privileged Architecture and CSR Specification*.

##### 9.1.1.1 HPM support in Linux

The HPM support in the Andes Linux kernel requires the specification of the option `CONFIG_RISCV_PMU`, which has been enabled by default in the configuration.

##### 9.1.1.2 HPM support in OpenSBI

OpenSBI has no switch of HPM-related features, which are default included in and initialized in the function `ae350_early_init()` of the file `<LINUX_ROOT>/opensbi/platform/generic/andes/ae350.c`.



## 9.1.2. PMA (Physical Memory Attribute)

The Physical Memory Attribute feature is available for AndesCore processors 27 and later series. It allows device drivers to set memory regions as non-cacheable if necessary. The bit “`mmio_cfg[PPMA]`” in Andes RTL is to indicate if the bitmap supports the PMA feature. System software will check this bit at runtime to determine the corresponding behavior.

### 9.1.2.1 PMA support by Linux kernel and the OpenSBI

The OpenSBI detects the setting in the bit “`mmio_cfg[PPMA]`” and passes information to the kernel. Though the Andes Linux kernel is configured to enable the PMA support by default, the PMA functionality will not be turned on until the kernel is notified by the OpenSBI.

For Andes cores with their bitmaps supporting the PMA feature, such as the 45-series, you can use the PMA functionality directly. However, for Andes cores without the PMA support, such as 25-series, the Linux kernel will still work but with a slight increase in code size. In this case, you can reduce the marginal size of the kernel image by turning off the PMA configuration in the kernel configuration through the path “`Device Drivers ---> / SOC (System On Chip) specific Drivers ---> / Andes AE350 platform drivers ---> / Andes PPMA Support`”.

### 9.1.2.2 PMA support by U-Boot

By default, cache is enabled in U-Boot. If you obtain a bitmap without the PMA support (e.g., a bitmap of an Andes core from the 25-series), you will need to disable the cache in the U-Boot configuration.

```
$ make menuconfig
RISC-V Architecture --->
[*] Do not enable icache
[*] Do not enable icache in SPL
[*] Do not enable dcache
[*] Do not enable dcache in SPL
```

## 9.1.3. Huge pages

Proceed as follows to configure the kernel to enable the huge page support and allocate and use huge pages. For detailed instructions, see the Linux kernel official document at

<https://www.kernel.org/doc/Documentation/admin-guide/mm/hugetlbpage.rst>.

**Step 1** Configure the Linux kernel as follows to enable the huge page support and build the kernel:

1. Run the `menuconfig` utility.

```
CROSS_COMPILE=riscv[32|64]-linux- ARCH=riscv make menuconfig
```

2. Follow below to enable the option `CONFIG_HUGETLBFS`:

```
File systems --->
```

```
Pseudo filesystems --->
```

```
[*] HugeTLB file system support
```

**Step 2** Open a shell prompt and issue as follows to confirm that the huge page has been enabled through the `/proc/meminfo` file.

```
$ cat /proc/meminfo | grep Huge
```

The output will be presented in the following format:

```
HugePages_Total: UUU
HugePages_Free: VVV
HugePages_Rsvd: WWW
HugePages_Surp: XXX
Hugepagesize: YYY kB
HugeTLB: ZZZ kB
```

where

- `HugePages_Total` is the size of the pool of huge pages.
- `HugePages_Free` is the number of huge pages in the pool that are not yet allocated.
- `HugePages_Rsvd` is short for “reserved” and is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. Reserved huge pages guarantee that an application will be able to allocate a

huge page from the pool of huge pages at fault time.

- `HugePages_Surp` is short for "surplus," and is the number of huge pages in the pool above the value in `"/proc/sys/vm/nr_hugepages"`. The maximum number of surplus huge pages is controlled by `"/proc/sys/vm/nr_overcommit_hugepages"`.
- `Hugepagesize` is the default size of the huge page (in kB).
- `Hugetlb` is the total amount of memory (in kB), consumed by huge pages of all sizes.

**Step 3** Use either of the following commands to allocate a specific number of huge pages:

- `$ echo <HUGEPAGE_NUMBER> > /proc/sys/vm/nr_hugepages`
- `$ sysctl -w vm.nr_hugepages=<HUGEPAGE_NUMBER>`

**Step 4** Request huge pages using either the `mmap` system call or a shared memory system call.

- Via the `mmap` system call

Set the parameter `FLAGS` to `MAP_HUGETLB` in the following function:

```
mmap(void *ADDR, size_t LENGTH, int PROT, int FLAGS,
 int FD, off_t OFFSET)
```

For example,

```
mmap(NULL, LENGTH, PROTECTION, MAP_HUGETLB, 0, 0);
```

- Via the shared memory system call "`shmget`"

Set the parameter `SHMFLG` to `SHM_HUGETLB` in the following function:

```
shmget(key_t KEY, size_t SIZE, int SHMFLG);
```

For example,

```
shmget(key, size, SHM_HUGETLB);
```

### 9.1.4. PMP (Physical Memory Protection)

As stated in *AndeStar V5 System Privileged Architecture and CSR Specification*, PMP can secure the system by limiting access from and to physical addresses. The accesses include Read, Write and eXecute operations.

Considering the speculative nature of advanced CPU functionality, you should protect your system with the following rules of thumb:

1. Do not allow execution requests or instructions fetches to device regions other than DRAM/SRAM/SPI Flash/PLDM as they may be handled incorrectly and cause the system to hang.
2. Disable all Read, Write and eXecute accesses to empty regions, especially when the bus controller doesn't respond appropriately. Additionally, make sure to disable the R/W/X accesses across all three modes (M-/S-/U-mode).

You can reference the OpenSBI source code to set up appropriate PMP settings for your platform. The setting of a PMP entry consists of two writes to a pair of CSRs, `pmpaddrX` and `pmpcfgX`, where `X` is an available number between 0 and the number of PMP entries minus 1. In practice, you can simply apply the existing API "`pmp_set`" to OpenSBI (`lib/sbi/riscv_asm.c`) referencing usages of the function `sbi_hart_pmp_configure()` in `lib/sbi/sbi_hart.c`.

### 9.1.5. DMA coherence

A DMA master device is considered coherent if its read/write to memory is coherent to the CPU's cache; otherwise, it is non-coherent for requiring a non-cacheable region to sync with the CPU while the OS/CPU side also has to perform cache operations before and after DMA data read/write operations. The information on whether a DMA master is coherent is critical and should be noted in the device tree blob passed to U-Boot and Linux kernel.

The device tree that represents your platform must follow the rules below to indicate the cache coherency of DMA master devices.

- For a coherent DMA master

Its device tree node must contain a dma-coherent attribute or be capable of recognizing a dma-coherent attribute in a higher level of the device tree (e.g. a global dma-coherent attribute not specific to any device). The attribute will be recognized by the device driver, thereby ensuring communication between the system software and the device without further settings.

- For a non-coherent DMA master

The dma-coherent attribute must not be contained in its device tree node or any higher level of the device tree. Once the device driver realizes that the device is not coherent, the system software can properly initialize the common memory region and handle each DMA transaction data.

Such use of the dma-coherent attribute with the Andes Linux package is to conform to the [Devicetree Specification](#). Some other systems, however, may not strictly require the dma-coherent attribute in their device trees for the coherent DMA masters as their default behaviors are different.

### 9.1.6. MSB mechanism (legacy non-cacheability mechanism)

Before the page-based memory type extension (Svpbmt) was ratified, there was no standard method to specify the cacheability for every page in Linux. Some drivers which rely on non-cacheable memory regions fail to function properly.

To work around the problems, Andes provides the MSB mechanism which treats the most significant bit in a physical address as a hardware signal to bypass cache hierarchy. For example, `0x8000A000` will be a non-cacheable page alias to that at `0x0000A000`, given that the `BIU_WIDTH` is 32.

The MSB mechanism is only applied when a system has no PMA (Physical Memory Attribute; see Section 9.1.2) and some drivers still require the Linux kernel to provide non-cacheable regions.

### 9.1.7. Asymmetric multi-processing (AMP) Linux kernel

To build the Linux kernel and DTB for an AMP system, see instructions in Sections 4.1 and 4.3. Make sure that you use a kernel default configuration file and Device Tree Source (DTS) file corresponding to the AMP system for the builds. For example, given the `AE350_AX25MP_AX25` AMP platform, you can use “`make ae350_rv64_ax25_amp_defconfig`” to generate the `defconfig` and the DTS file `ax25mp_amp_c2_64_d_ae350.dts` for the associated DTB.

With the built kernel and DTB, you can follow Chapter 6 to boot the Linux kernel.

If you want to run the AMP kernel with OpenAMP, please see *Andes AMP System with OpenAMP Application Note*. This document provides instructions on how to build and run OpenAMP bare-metal binaries and Linux AMP applications together on Andes AMP platforms.

## 9.2. Special peripherals

### 9.2.1. SMU (Power management)

#### Suspend feature

Andes targets are integrated with a System Management Unit (SMU) to save power by suspending the system or specific core(s). The suspend feature offers two configurations for system-wide suspension, i.e., a light sleep mode to suspend the system to standby and a deep sleep mode to suspend it to memory. In addition, the hart-wide suspension is also possible when the feature works with the CPU hotplug in deep sleep.

To perform the suspend feature, proceed as follows:

**Step 1** Enable the SMU support in the Linux kernel:

1. Run the `menuconfig` utility.

```
CROSS_COMPILE=riscv[32|64]-linux- ARCH=riscv make menuconfig
```

2. Navigate through the submenus “Device Drivers → SOC (System On Chip) specific Drivers → Andes AE350 platform drivers → Andes ATCSMU Support” to access and enable the `CONFIG_ATCSMU` option.

**Step 2** Add the smu node to the device tree source (DTS) file.

```
smu@f0100000 {
 compatible = "andestech,atcsmu";
 reg = <0x0 0xf0100000 0x0 0x1000>;
};
```

**Step 3** For devices featuring the wakeup capability, you may set them to be the wakeup events. If you are using the built-in DTB, issue as follows:

```
echo enabled > /sys/devices/platform/soc/.../power/wakeup
```

With a manually built DTB, issue like below as there is no “soc”

directory layer in the path.

```
echo enabled > /sys/devices/platform/.../power/wakeup
```

The following example is to set UART as the wakeup event when a manually built DTB is used.

```
echo enabled >
/sys/devices/platform/f0300000.serial/tty/ttyS0/power/wakeup
```

**Step 4** Specify a power-saving configuration of system-wide suspension or hart-wide suspension. The two suspension types cannot be used at the same time.

■ System-wide suspension

Make sure no hart is in sleep mode and proceed with the following:

- For a light sleep mode, issue as follows to suspend your system to standby:

```
echo standby > /sys/power/state
```

- For a deep sleep mode, issue as follows to suspend the system into memory:

```
echo mem > /sys/power/state
```

■ Hart-wide suspension

To suspend a specific CPU with CPU hotplug in deep sleep mode, apply the unplug command for CPU hotplug. For details, see the CPU hotplug section later.

Note that all backup and restore processes associated with the suspend feature are programmed in [opensbi/platform/generic/andes/sleep.S](#). You can edit the file and specify desired registers to back up and restore the system state for the suspension and wake-up.

**Step 5** During debugging, you may find that GDB is not able to stop at a



designated breakpoint after the system or core wakes up from deep sleep. To solve the problem, make sure only hardware breakpoints are set and use either of the following methods to ensure the stop:

- Launching ICEman with the `--halt-on-reset` option

```
./ICEman --halt-on-reset 1
```

- Issuing GDB commands as follows

```
(gdb) monitor nds configure halt_on_reset 1
```

---

## NOTE

1. To ensure that targets with SMU operate properly after waking up from light/deep sleep, all the peripheral device drivers must implement the suspend and resume functions.
  2. At present, there is a compatibility problem between the SD card and SMU on the VCU118 FPGA board. As this problem can hinder the functionality of SMU, it is recommended to remove the SD card when operating SMU on the VCU118 FPGA board.
  3. Local interrupts may block CPU cores from entering the sleep mode. As a result, please disable the local interrupts beforehand. In addition, the current implementation of the Hardware Performance Monitor (HPM) does not support the sleep mode. If the monitoring is required during the sleep mode, you will need to specify when to disable/enable CSR\_SLIE in your drivers. For instructions on enabling/disabling CSR\_xIE, please reference the function `smu_suspend_prepare()` in `opensbi/lib/utis/sys/atcsmu.c`.
- 

## Reboot feature

Andes' reboot feature for the Linux kernel is supported only when harts are not in deep sleep mode or unplugged from the system. It requires an SMU that supports reset commands or a watchdog driver. U-Boot is also engaged in the reboot process.

**Step 1** Make sure you have U-Boot installed at the memory address `0x80000000` of the target board.

**Step 2** Configure the kernel to enable the SMU support by running the `menuconfig` utility and selecting the option `CONFIG_ATCSMU` from

the submenus “Device Drivers → SOC (System On Chip) specific Drivers → Andes AE350 platform drivers → Andes ATCSMU Support.” Then, add the SMU node to the device tree source file (DTS) as follows:

```
smu@f0100000 {
 compatible = "andestech,atcsmu";
 reg = <0x0 0xf0100000 0x0 0x1000>;
};
```

If the SMU does not support reset commands, you will need to reboot the system via a watchdog timer. In this case, additionally activate the watchdog driver in the kernel by selecting the option `CONFIG_ATCWDT200_WATCHDOG` in the `menuconfig` path “Device drivers → Watchdog Timer Support → ATCWDT200\_WATCHDOG” and add the wdt node to the DTS like below.

```
wdt@f0500000 {
 compatible = "andestech,atcwdt200";
 reg = <0x0 0xf0500000 0x0 0x1000>;
 interrupts = <0x03 0x04>;
 interrupt-parent = <0x06>;
 clock-frequency = <15000000>;
};
```

**Step 3** Type the command “`reboot`” in the shell and then the CPU will jump to U-Boot at `0x80000000`. Follow the instructions in Section 6.1.2 to re-boot the kernel with U-Boot.

### PowerBrake feature

Andes’ PowerBrake feature allows you to scale the CPU performance with the throttling level defined by the register `mpft_ctl`. For details about the performance throttling control register, see *AndeStar V5 System Privileged Architecture and CSR Specification*.

The PowerBrake usages include the following.

- View all available governors

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

```
~ # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
performance powersave
```

- View the current governor

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

```
~ # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
powersave
```

- Set a governor

```
echo GOVERNOR > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

The following example is to set the governor to performance.

```
~ # echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
~ # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance
```

- View the current CPU frequency (in KHz)

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

```
~ # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
600000
```

---

## NOTE

The displayed CPU frequency is not entirely accurate because it just reflects the throttling level set by the `mpft_ctl` register.

---

- Specify a CPU frequency range (in KHz)

```
echo MIN_FREQ > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

```
echo MAX_FREQ > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

The following is to set the CPU to a frequency between 10 and 100 KHz.

```
echo 10 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

```
echo 100 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

Based on the frequency range, the `mpft_ctl` register will be modified to switch to an

appropriate throttling level. The following table explains the conversion between the frequency ranges and throttling levels.

| CPU frequency range                                                                 | Throttling level |
|-------------------------------------------------------------------------------------|------------------|
| $1 \leq \text{MIN\_FREQ} < \text{MAX\_FREQ} \leq 3750$                              | 15               |
| $3751 \leq \text{MIN\_FREQ} < \text{MAX\_FREQ} \leq 3750 * 2$                       | 14               |
| $3750 * 2 + 1 \leq \text{MIN\_FREQ} < \text{MAX\_FREQ} \leq 3750 * 3$               | 13               |
| $3750 * (15 - N) + 1 \leq \text{MIN\_FREQ} < \text{MAX\_FREQ} \leq 3750 * (16 - N)$ | N (N=0~15)       |

## CPU hotplug

The CPU hotplug feature allows you to specify whether a CPU is available at runtime. When the kernel option `CONFIG_ATCSMU` is enabled, the CPU hotplug will work with the deep sleep function of SMU to achieve hart-wide suspension. Otherwise, a core only executes the CPU hotplug function of the software layer. For detailed instructions on how to enable the suspend support of SMU, see the previous section “Suspend feature”, Step 1~Step 3.

For the 25-series, please note that the suspend function of SMU does not apply to Core 0. This is because Core 0 is bound with the L2 cache and limited to software-level CPU hotplugging in any case.

The following lists some usage examples of the CPU hotplug feature –

- View the information about the processors in use:

```
cat /proc/cpuinfo
```

```
~ # cat /proc/cpuinfo
processor : 0
hart : 0
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100

processor : 1
hart : 1
isa : rv64imafdc
```

```
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100

processor : 2
hart : 2
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100

processor : 3
hart : 3
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

The above output shows there are four cores used by the system.

#### ■ Unplug Core 1:

```
echo 0 > /sys/devices/system/cpu/cpu1/online
```

```
~ # echo 0 > /sys/devices/system/cpu/cpu1/online
[6254.830710] CPU1: off
ae350_enter_suspend_mode(): CPU[1] Cpu Hotplug DeepSleepMode
~ # cat /proc/cpuinfo
processor : 0
hart : 0
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100

processor : 2
hart : 2
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

```
processor : 3
hart : 3
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

Official  
Release

## ■ Plug Core 1:

`echo 1 > /sys/devices/system/cpu/cpu1/online`

```
~ # echo 1 > /sys/devices/system/cpu/cpu1/online
```

```
~ # cat /proc/cpuinfo
```

```
processor : 0
hart : 0
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

```
processor : 1
hart : 1
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

```
processor : 2
hart : 2
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

```
processor : 3
hart : 3
isa : rv64imafdc
mmu : sv39
mvendorid : 0x31e
marchid : 0x8a45
mimpid : 0x70100
```

### 9.2.2. PWM (Pulse Width Modulation)

Two PWM modules, pwm0 and pwm1, are provided under `/sys/class/pwm/pwmchip0/` for PWM devices on Andes target boards. The pwm0 has an available duty cycle range of 30.5us~1.998s and a period of 61us~3.997s. The pwm1, on the other hand, has an available duty cycle range of 16.6ns~1.078ms and a period of 33.2ns~2.175ms. In either module, the duty cycle and period values must be multiples of their fundamental units, and the period value must always be larger than the duty cycle value.

The following instructions are useful for configuring PWM modules for LED blinking with a noticeable frequency,

- Export a PWM module (e.g., pwm0) to user space

```
/sys/class/pwm/pwmchip0 # echo 0 > export && cd pwm0
/sys/class/pwm/pwmchip0/pwm0 #
```

- Set the period of the PWM module (e.g., 2s)

```
/sys/class/pwm/pwmchip0/pwm0 # echo 2000000000 > period
```

- Set the duty cycle for a PWM module (e.g., 1s)

```
/sys/class/pwm/pwmchip0/pwm0 # echo 1000000000 > duty_cycle
```

- Enable the PWM signal

```
/sys/class/pwm/pwmchip0/pwm0 # echo 1 > enable
```

## 9.2.3. dmatest (DMA test for memcpy)

The dmatest is a test suite capable of testing the DMA driver via the DMA engine inside the Linux kernel. Among the available two DMA drivers (see Section 4.2), only the ATCDMAC300G DMA driver provides a channel capability for DMA\_MEMCPY (memory-to-memory) transactions and can be verified by this test suite.

To adopt the DMA driver with the DMA engine and test its memcpy capability with dmatest, proceed as follows:

- Step 1** Make sure that the DTB in use contains a DMA node with a DMA engine and jump to the next step.

DMA engine has been enabled by default for Andes software since AndeSight v5.2.0. For the hardware part, however, some evaluation bitmaps still contain a built-in DTB supporting only the DMA driver without the DMA engine. Therefore, in the case the built-in DTB is used, be sure to retrieve it and inspect the device tree following the note instructions in Section 4.3. If the DTS file doesn't contain a DMA node with a DMA engine, follow below to modify it and see Section 4.3 to build a supportive DTB file for your kernel.

- For the mmc node
  - Modify the compatible string as “andestech,atfsdc010g”
  - Add the following lines
 

```
dmac = <&dma0 9>;
dma-names = "rxtx";
```
- For the dma node
  - Add a “dma0: ” label before the node name “dma@f0c00000”
  - Modify the compatible string as “andestech,atcdmac300g”
  - Change the interrupts to “<10 4>”
  - Add a line “#dma-cells = <1>;”



- Check if the attribute “[dma-coherent](#)” is specified for the mmc node of the DTS file. If yes, add “[dma-coherent](#)” to the dma node too for the sake of consistency.

**Step 2** Use `dmatest` to test the DMA driver with the DMA engine:

1. Change the current directory to the [sys](#) directory for `dmatest`.

```
cd /sys/module/dmatest/parameters/
```

2. Specify the test buffer size. For example,

```
echo 4000032 > test_buf_size
```

3. Specify the transfer size for `memcpy` and make sure that it is less than the buffer size you just set. For example,

```
echo 4000000 > transfer_size
```

4. Specify the loop count for the transfer operation. For example,

```
echo 1 > iterations
```

5. Designate Channel 1 for the `DMA_MEMCPY` transaction.

```
echo dma0chan1 > channel
```

6. Enable the DMA for the designated channel and start the test.

```
echo 1 > run
```

```
/sys/module/dmatest/parameters # echo 4000032 > test_buf_size
/sys/module/dmatest/parameters # echo 4000000 > transfer_size
/sys/module/dmatest/parameters # echo 1 > iterations
/sys/module/dmatest/parameters # echo dma0chan1 > channel
[4066.361351] dmatest: Added 1 threads using dma0chan1
/sys/module/dmatest/parameters # echo 1 > run
[4066.371220] dmatest: Started 1 threads using dma0chan1
/sys/module/dmatest/parameters # [4066.630683] dmatest: dma0chan1-copy0:
summary 1 tests, 0 failures 4.44 iops 17380 KB/s (0)
```

### 9.2.4. GPIO (General-purpose input/output)

When the ATCGPIO100 driver is successfully loaded into the Andes Linux kernel, controlling files are created in `/sys/class/gpio` and its subdirectories to operate the GPIOs. You can use these files to configure the supported GPIO number or individual GPIO as follows, where **ID** represents the index of the GPIO you want to control.

- Specify the number of supported GPIOs on your platform

The ATCGPIO100 driver supports 16 GPIOs by default. If this default number of GPIOs does not match the GPIO amount on your target board, modify your DTS file and change the `ngpios` value in the gpio-controller node accordingly. For example,

```
gpio-controller {
 ...
 ngpios = <32>;
}
```

- Export a GPIO to user space

```
echo ID > /sys/class/gpio/export
```

- Set the direction of a GPIO

- Set the GPIO as an output

```
echo "out" > /sys/class/gpio/gpioID/direction
```

- Set the GPIO as an input

```
echo "in" > /sys/class/gpio/gpioID/direction
```

- Get the input value of a GPIO

```
cat /sys/class/gpio/gpioID/value
```

- Unexport a GPIO

```
echo ID > /sys/class/gpio/unexport
```

## 9.3. Boot flow unveiled

This section reveals the details of the Normal Boot and Fast Boot process in Chapter 6. It gives an overview of the boot flow and provides information that helps to resolve problems on platform bring-up. The subsections are ordered according to the boot sequence for ease of reference.



### 9.3.1. U-Boot SPL

After being executed, the SPL (Secondary Program Loader) (i.e., `<LINUX_ROOT>/u-boot/spl/u-boot-spl`) goes through the function `call_board_init_f` to initialize basic peripherals and the function `call_board_init_r` for parsing the ITB image under `<LINUX_ROOT>/u-boot/` later. For a Normal Boot process, `call_board_init_r` will parse the U-Boot ITB image, `u-boot.itb`, which contains both OpenSBI and U-Boot proper; for a Fast Boot process, the function will parse the Linux ITB image, `linux.itb`, which contains OpenSBI and Linux kernel. The boot flow jumps to OpenSBI in M-mode next and never comes back to the U-Boot SPL.

### 9.3.2. OpenSBI

The firmware used at this stage is `fw_dynamic.bin` and `fw_dynamic.elf` in `<LINUX_ROOT>/opensbi/build/platform/generic/firmware`. – The former is the binary file used to build the U-Boot or Linux ITB image; the latter is in ELF format and used for debugging.

This phase involves a “lottery” procedure that determines a main hart to perform one-shot initializations or global changes to the system. One hart is chosen to do the `cold_boot` function and the others go to the `warm_boot` function. The cold-boot hart initializes M-mode runtime structures (e.g., scratch space) and function pointers (e.g., SBI call entries) first. In a Normal Boot process, the cold-boot hart then goes into the U-Boot proper and next Linux kernel with a privilege-level transition from M-mode to S-mode; in a Fast Boot process, the hart skips the U-Boot proper and goes directly to the kernel. Other harts just wait in the function `sbi_hsm_hart_wait` until they are waked up by an SBI call, as described in Section 9.3.4 later.

### 9.3.3. U-Boot proper (in Normal Boot process only)

The cold-boot hart in a Normal Boot process goes to the U-Boot proper, which relocates itself to the top of the memory and resumes from there. If everything goes as expected, the U-Boot proper will provide a prompt for you to specify and manipulate the boot device. After that, the boot flow moves to the Linux and never returns to the U-Boot proper.



### 9.3.4. Linux

The cold-boot hart alone executes the architecture-dependent initialization code. At this stage, interrupts are masked and the trap vector is set to catch early errors. In addition, a few sets of page tables are set up and the MMU (Memory Management Unit) is enabled.

The boot flow now goes to the generic boot function `start_kernel`, which is written in C and contains many function calls named `*_init`. While in `setup_smp`, this hart issues an SBI call to wake up other harts, which wait in `sbi_hsm_hart_wait` at the moment. These harts thus resume executions, arrive at Linux afterwards and enter the idle state waiting for tasks.

### 9.3.5. User space

Busybox with sysinit is delivered for system initialization. You may check the scripts in `/etc/init.d` for details of the startup work.

## 10. Troubleshooting

### 10.1. Boot process troubleshooting

#### 10.1.1. Debugging DTB

If you obtain an experimental bitmap from Andes and fail to bring up the system following the instructions in Chapter 6, it is recommended to check the DTB configuration first. While the cause of the problem varies, the following provides a general guide to diagnose the possible cause and troubleshoot the problem by modifying the DTS file and building a new DTB.

**Step 1** Make sure you have the DTC (device tree compiler) installed. For where to obtain the tool, see Section 4.3, Step 2.

**Step 2** Disassemble your existing DTB (e.g., `orig.dtb`) into a DTS file.

```
$ dtc -I dtb -O dts orig.dtb -o orig.dts
```

**Step 3** Use GDB to check the value of the `mcause` or `scause` register and diagnose the exception that just occurred. The two most likely cases are as follows:

- When the value of the `mcause/scause` register is `0x2`:  
An illegal instruction exception is indicated. You may have applied an FPU-capable DTB (i.e., with “`f2p0d2p0`” specified in the “`riscv,isa`” attribute of each CPU node) to a no-FPU bitmap or a DSP-capable DTB (i.e., with “`xdsp0p0`” specified in the “`riscv,isa`” attribute) to a no-DSP bitmap. In this case, remove the sub-strings mentioned above from your DTS file or reference the DTS templates Andes provides (see Section 4.3, Step 2) to modify your DTS file.
- When the value of the `mcause/scause` register is `0x5` or `0x7`:  
A load/write access fault is indicated. The system may have attempted to access the MMIO region of a device that the bitmap does not have. In such a case, use GDB to check the `mepc/sepc`

register for the kernel function and device that resulted in the fault and then remove the device node from the DTS file. For example, if the fault occurs at the `i2c_probe` function, just delete the `i2c` device node from your DTS file.

**Step 4** Compile the modified DTS file (e.g., `fixed.dts`) into a DTB (e.g., `fixed.dtb`).

```
$ dtc -I dts -O dtb fixed.dts -o fixed.dtb
```

Then, use the newly generated DTB to reboot the Linux kernel.

### 10.1.2. Boot hangs in early stage due to lottery

A lottery mechanism is used in the early stage of the boot process to determine the main hart that controls most of the platform. It requires concurrent access from all the harts, so the use of atomic instructions is necessary.

However, the atomic instructions used for the lottery sometimes may cause problems on the platforms in scenarios like

- The AXI Exclusive Access protocol is not properly integrated into the platform. This leads to the failure of atomic instructions directly.
- The AXI Burst Transfer is not properly integrated into the platform. This makes atomic instructions unlikely to succeed once the cache/coherency of the CPUs is enabled and burst transfers go wrong on the bus.

If you want to eliminate the atomic instructions used for the lottery mechanism in the early stage, just slightly rewrite the boot code related to the lottery. For example, the following part in the file to boot the U-Boot-SPL (i.e., `<LINUX_ROOT>/u-boot/arch/riscv/cpu/start.S`)

```
la t0, hart_lottery
li t1, 1
amoswap.w s2, t1, 0(t0)
bnez s2, wait_for_gd_init
```

can be re-written to

```
csrr s2, mhartid
bnez s2, wait_for_gd_init
```

## 10.2. Information for diagnosing kernel boot failure or crashes

If you run into a reproducible kernel boot or crash problem and want to raise a support request, please gather and provide the following information. The information will help Andes technical support staff to identify the cause of your problem and expedite the troubleshooting process.

- Information about your boot process

Please describe how you booted the kernel or provide your shell script for the booting.

- Binary files used for the boot process

These include all the binaries in ELF format, such as kernel image (`vmlinux`), OpenSBI image (`fw_dynamic.elf`), U-Boot image (`u-boot`), and your DTS/DTB file. If you were using the built-in DTB file, please let us know too.

- Information about your hardware architecture

Please provide the configuration file of your hardware architecture (`.cfg`) and, if possible, the memory map information as well.

- Information about the state of the system when the problem occurred

- For the case of kernel hangs, check if the console interrupt is available. If not, issue commands as follows to identify the stuck thread at the moment.

```
(gdb) Ctrl+C
(gdb) info threads
```

Also, collect the register information for the hang using the following command:

```
(gdb) info all-register
```

- For the case of kernel panics, collect the kernel panic messages and issue as follows to get the register information upon the kernel failure.

```
(gdb) info all-register
```

### 10.3. Access fault at unusual physical address when bringing up peripheral devices

For the 25-series CPUs, check if the physical address is reasonable after ignoring the MSB bit. If it is, the problem is associated with the MSB mechanism described in Section 9.1.6. To resolve it, follow the instructions in Section 9.1.5 to specify the DMA-coherent attribute in the device tree.

In the case that the physical address looks unreasonable but like a normal virtual address after ignoring the MSB bit, see Section 9.1.4 to add appropriate PMP settings for your platform.