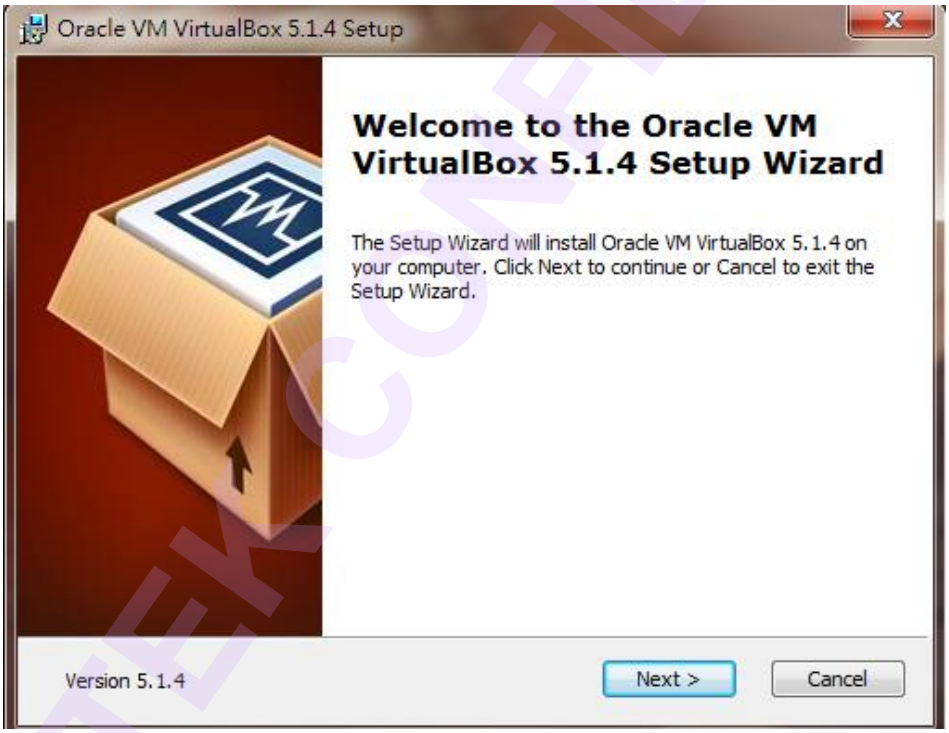
NT96680 SDK Pure Linux Programing Guide

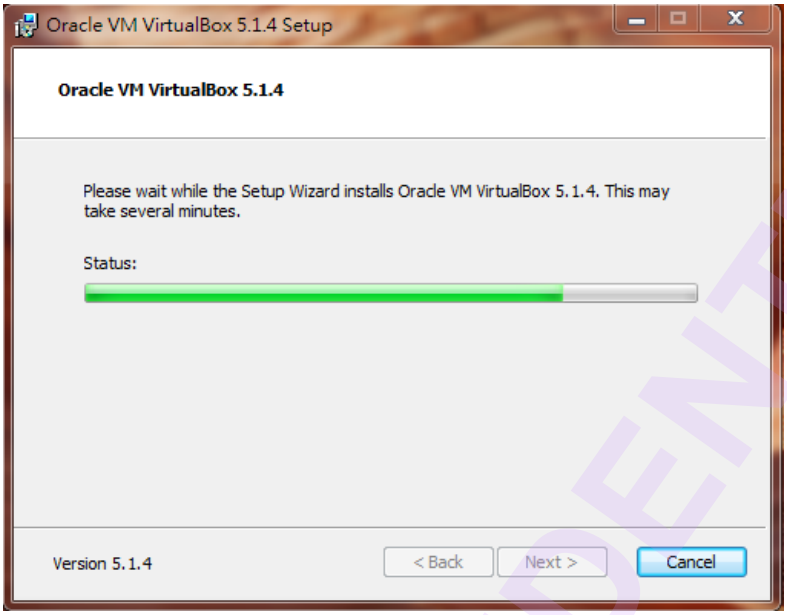
Table of Content

1 Development Environment

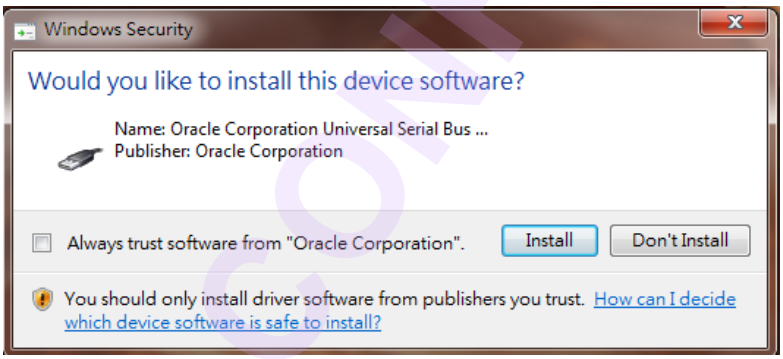
1.1 VirtualBox/Ubuntu OS installation

This section will introduce how to install Ubuntu on Windows OS, ignore this section if your environment is under single OS. Please download from VirtualBox official website (https://www.virtualbox.org/wiki/Downloads) and follow below instructions to install.

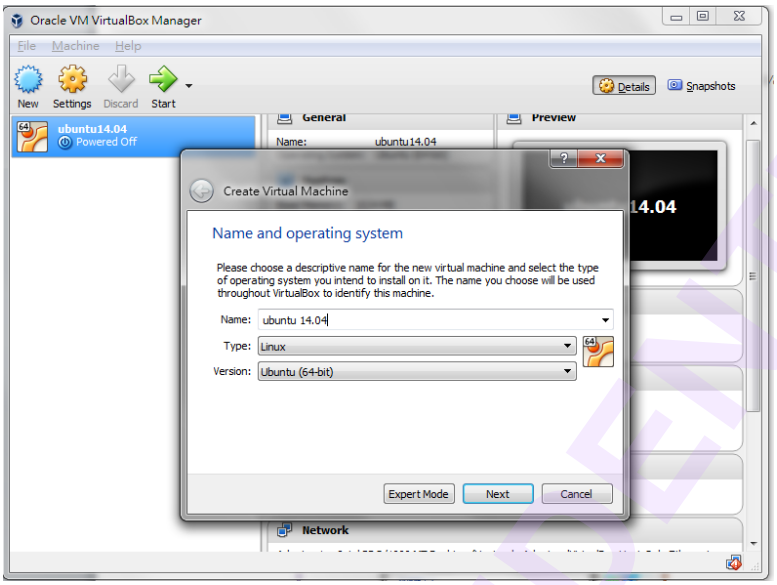




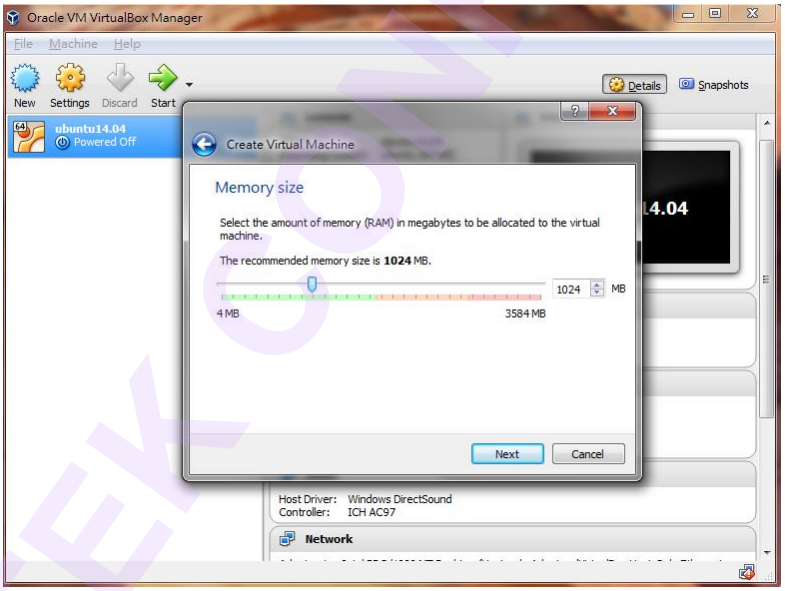
This step will install device driver, please select “Install”:



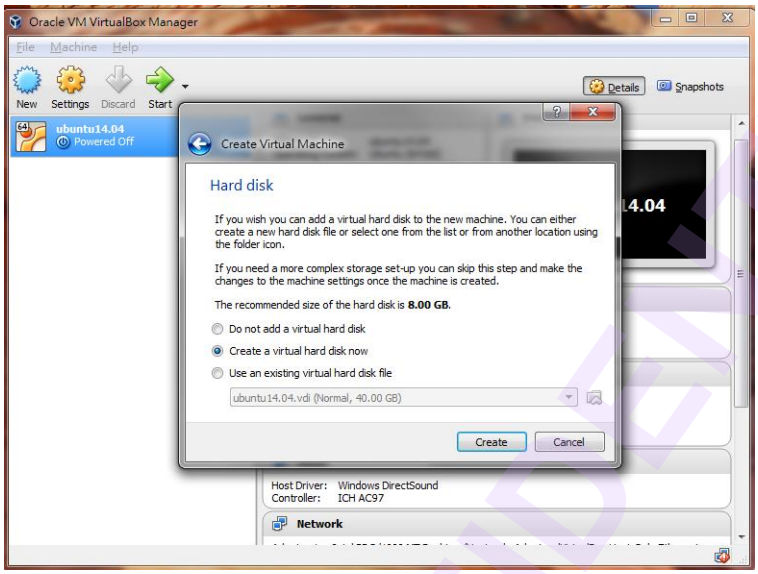
Create virtual machine (Version: Ubuntu 64-Bit):



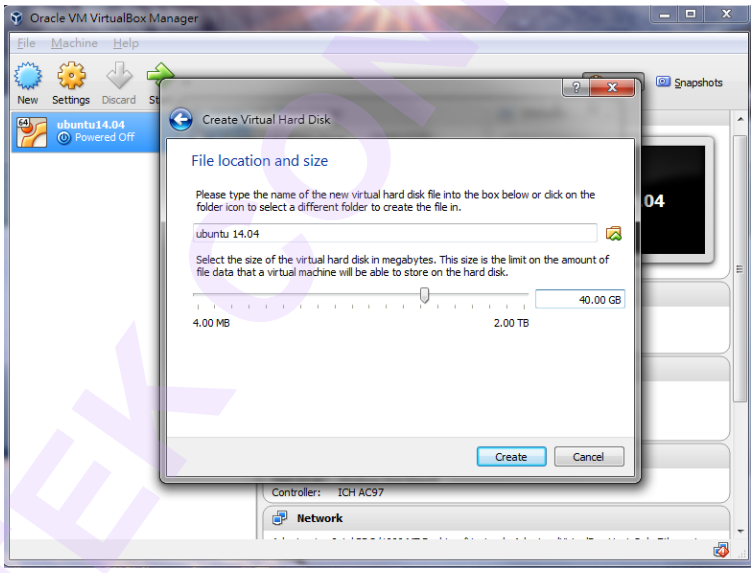
Select virtual machine memory size:



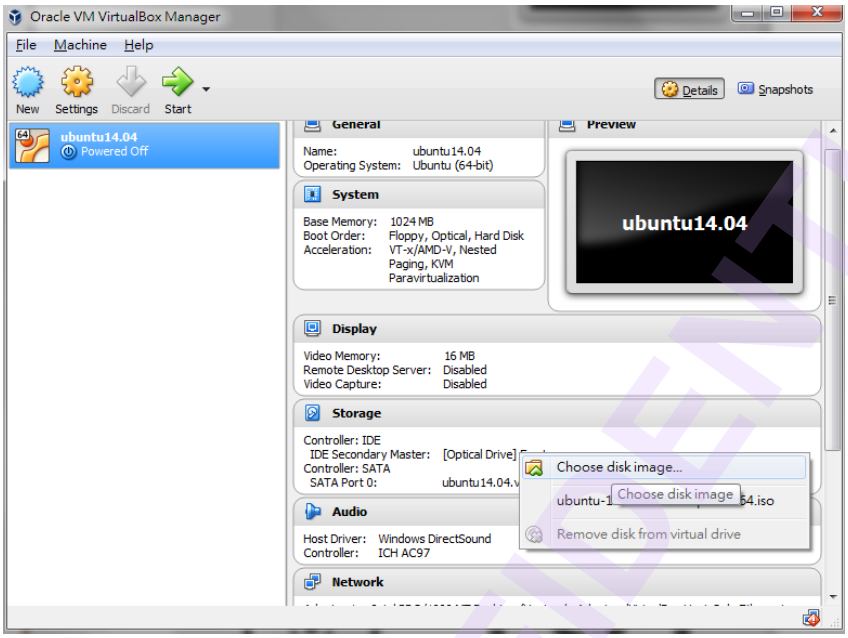
Create virtual machine (If you have an existed Ubuntu image, you can select “Use an existing virtual hard disk file”):

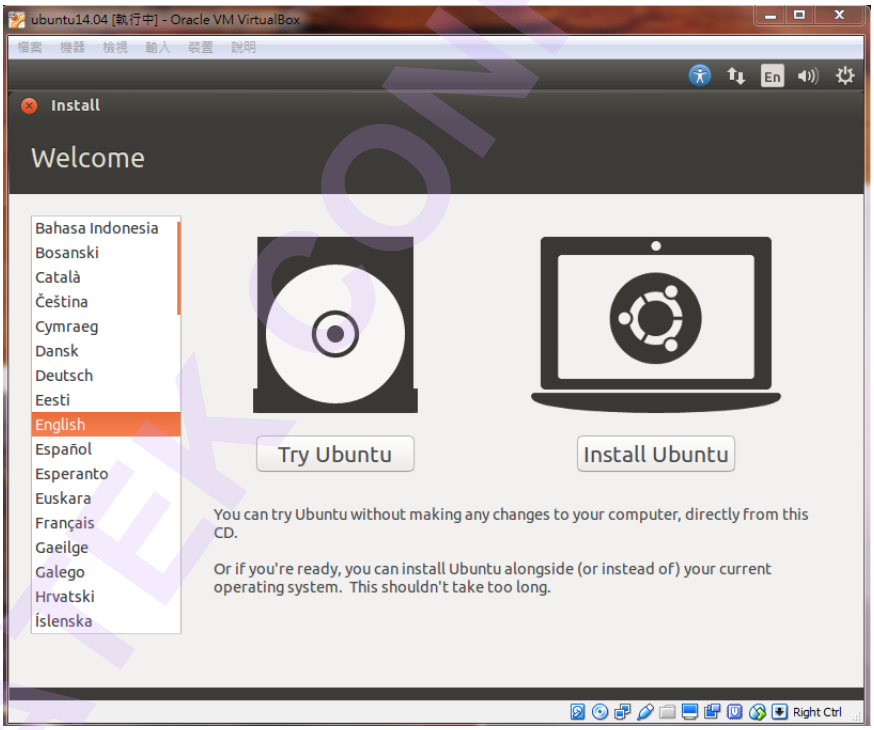


To configure virtual disk space (At least 40GB size):



Select Optical drive and using Ubuntu ISO (Ubuntu 12.04/14.04 64-bit) startup:





1.2 Linux environment setup

In order to prevent some complaints in the SDK compilation of the 32 Bits OS, we will use 64 Bits Ubuntu OS as our development environment. The first of all, you should install an Ubuntu based server or Ubuntu on VirtualBox which is introduced in previous section, please download the image from (http://releases.ubuntu.com/) to get Ubuntu 12.04/14.04/16.04 Desktop AMD64 version ISO image and use the following instructions to install necessary Ubuntu packages.

Ubuntu 12.04:

$ sudo apt-get install build-essential libc6-dev libncurses5-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos ia32-libs uboot-mkimage zlib1g-dev mtd-utils vim squashfs-tools gawk cmake cmake-data libstdc++6 device-tree-compiler android-tools-fsutils texinfo

$ sudo add-apt-repository ppa:nathan-renniewaldock/ppa
$ sudo apt-get update
$ sudo apt-get install liblz4-tool

Ubuntu 14.04:

$ sudo apt-get install build-essential libc6-dev lib32ncurses5 libncurses5-dev libncurses5:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos lib32z1 lib32bz2-1.0 u-boot-tools zlib1g-dev bison libbison-dev flex mtd-utils vim squashfs-tools gawk cmake cmake-data liblz4-tool libmpc3 libstdc++6 device-tree-compiler android-tools-fsutils texinfo

Ubuntu 16.04:

$ sudo apt-get install build-essential libc6-dev lib32ncurses5 libncurses5-dev libncurses5:i386 libgl1-mesa-dev g++-multilib mingw-w64 tofrodos lib32z1 u-boot-tools zlib1g-dev bison libbison-dev flex mtd-utils vim squashfs-tools gawk cmake cmake-data liblz4-tool libmpc3 libstdc++6 device-tree-compiler android-tools-fsutils texinfo

Ubuntu default shell is dash, please reconfigure the default shell with bash:

$ sudo dpkg-reconfigure dash, and choose “No” in the window

Or

$ sudo rm /bin/sh && sudo ln -s /bin/bash /bin/sh

Besides, the openssh-server is used to provide Windows Host PC connected to Linux server and remote building the Linux SDK, and the Samba server is to provide client get Linux SDK image from Linux server.

- $ apt-get install openssh-server
- $ apt-get install samba
- $ vim /etc/samba/smb.conf
  - Please reference to related Samba configuration as below: https://help.ubuntu.com/12.04/serverguide/samba-fileserver.html

Windows Host PC will also need Teraterm or putty to connect to Target board UART2 port with 115200/8/1/n configuration.

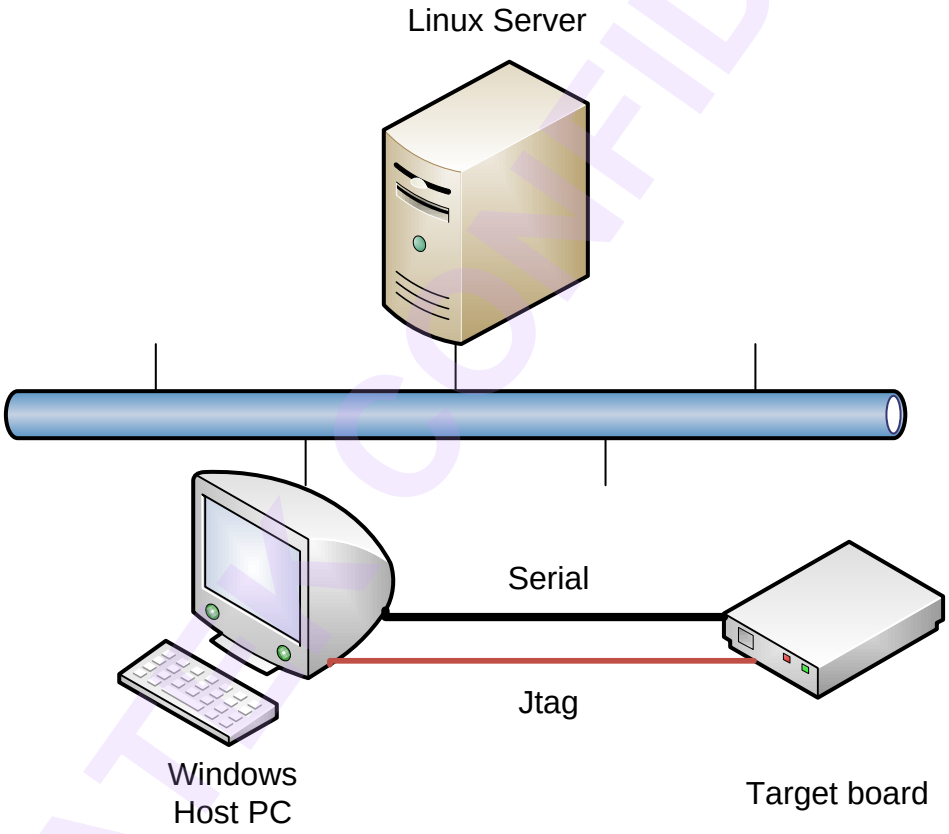


Figure 1-1 Linux Development Environment

1.3 How to install NA51000 Linux SDK

Using the following instructions to decompress SDK pack under Linux:

```
$ tar -jxvf na51000_linux_sdk_{version}.tar.bz2
```

You will get the folder tree as below:

```
├─── na51000_linux_sdk          Used to put unpacked SDK source code
    ├─── BSP                   Including linux, busybox, uboot and rootfs source
    ├─── build                 scripts for the environment setup
    ├─── code                  HDAL, linux drivers and sample code
    ├─── configs               Model settings
    ├─── Makefile              Top level Makefile
    └─── tools                 target board tool
├─── toolchains                Toolchain folder
    ├─── arm-ca53-linux-uclibcgnueabihf-6.4.tar.bz2
    └─── arm-ca53-linux-gnueabihf-6.4.tar.bz2
```

1.4 How to install Cross compiler

We support both glibc and uclibc cross compiler toolchain, please choose and install it by using below instructions:

```
$ cd toolchain
$ sudo tar -jxvf arm-ca53-linux-gnueabi-6.4-{DATE}.tar.bz2  -C /opt/arm
or
$ sudo tar -jxvf arm-ca53-linux-uclibcgnueabihf-6.4-{DATE}.tar.bz2  -C /opt/arm
```

2 Introduction to Compilation

2.1 Environment setup

Before each opened a new Terminal window needs to do compiler environment setting, the relevant variables set up, please follow the below instructions to finish it.

```
$ cd na51000_linux_sdk\
$ source build/envsetup.sh
```

2.2 Compilation

Please do a complete compilation for first time.

Select your model:

```
$ lunch
```

List your nvt build setting:

```
$ get_stuff_for_environment
```

Build overall system:

```
$ make all
```

It will generate the images under na51000_linux_sdk/output after the compilation. The details are listed as below.

```
|-- na51000_linux_sdk                  Put unpacked source code and image
|    |-- Makefile                      Top level Makefile
|    |-- output                        Compiled output images
|         |-- packed
|              |-- FW(SOC)A.bin        nvtpack image (All-in-one image)
|         |-- FW(SOC)A.bin             itron image
|         |-- uImage.bin               Linux uImage
|         |-- uboot.bin                uboot image with NVT checksum
|         |-- rootfs.ramdisk.bin       ramdisk image
|         |-- rootfs.ubifs.bin         UBIFS image with NVT checksum
|         |-- rootfs.ubifs.bin.raw     UBIFS image without NVT checksum
|         |-- rootfs.squash.bin        Squashfs image with NVT checksum
|         |-- rootfs.squash.bin.raw    Squashfs image without NVT checksum
|         |-- rootfs.jffs2.bin         jffs2 image with NVT checksum
|         `-- rootfs.jffs2.bin.raw     jffs2 image without NVT checksum
```

2.3 Top level Makefile

na51000_linux_sdk folder has a top level Makefile, it supports many of the make command, such as “make linux” is to compile linux-kernel, “make uboot” can compile u-boot, “make rootfs” can compile root-fs ... and so on, you can use “make help” to find what its commands are supported. Please use top level Makefile to do SDK compilation to avoid some link error occurred. Its help description is as follows:

```
$ make help
==================================================
make help                -> show make command info
make all                 -> build all
make linux               -> build linux-kernel
make linuxram            -> build linux-kernel with ramdisk support
make modules             -> build built-in kernel modules
make linux_driver        -> build NVT linux driver modules
make uboot               -> build loader(uboot)
make library             -> build library
make busybox             -> build busybox
make rootfs              -> build rootfs
make app                 -> build applications
make tools               -> build tools
make sample              -> build sample code
make pack                -> Generate nvtpack image
==================================================
make linux_config        -> config linux-kernel
make uboot_config        -> config uboot
make busybox_config      -> config busybox
make linux_header        -> generate linux-kernel out of tree headers
==================================================
make clean               -> clean all
make linux_clean         -> clean linux-kernel & built-in kernel modules
make linux_driver_clean  -> clean NVT linux driver modules
make uboot_clean         -> clean loader(uboot)
make library_clean       -> clean library
make busybox_clean       -> clean busybox
make rootfs_clean        -> clean rootfs
```

```
make app_clean          -> clean applications
make tools_clean        -> clean tools
make sample_clean       -> clean sample code
make pack_clean         -> Remove nvtpack image
================================================
```

2.4 Project configuration

We provide the following file to control functionalities enable or disable, please refer to the below procedures to configure.

Check model type:

$ get_stuff_for_environment

```
======================================== NVT Setting ========================================
NVT_PRJCFG_MODEL_CFG = /home/user1/na51000_linux_sdk/configs/cfg_IPCAM1_EVB/ModelConfig.mk
NVT_PRJCFG_CFG = Linux
NVT_PRJCFG_MODEL_MK =
```

The ModelConfig.mk will be generated from configs/cfg_gen/nvt-na51000-info.dtsi

Enable/disable function:

$ cd na51000_linux_sdk/

$ vi configs/cfg_gen/ nvt-na51000-info.dtsi

To find Linux related options:

```
    nvt_info {                          /* Get from ModelConfig.txt */
        BIN_NAME = "FW96687A";
        BIN_NAME_T = "FW96687T";
        /* EMBMEM_BLK_SIZE, Normally, 2KPageNand=0x20000, 512PageNand=0x4000, SPI=0x10000 */
        EMBMEM_BLK_SIZE = "0x20000";


        /*
         * [EMBMEM]
         * EMBMEM_NONE
         * EMBMEM_NAND
         * EMBMEM_SPI_NOR
         * EMBMEM_SPI_NAND
         * EMBMEM_EMMC
         */
        EMBMEM = "EMBMEM_SPI_NAND";
        /*
```

```
            * ======= Linux common =========
            * application/external
            */
           NVT_CFG_APP_EXTERNAL = "dhd_priv";
           /* application include list */
           NVT_CFG_APP = "hfs lviewd nvtrtspd nvtipcd nvteventd uctrl fslinux ugxstrg nvtsystem
camctrlcgi nvtweb nvtrecordManagerd msdcnvt arm_neon_perf mem ucmd isp_demon";
           /* rootfs etc folder */
           NVT_ROOTFS_ETC = "";
           /* strip executable binary and library files: yes/no */
           NVT_BINARY_FILE_STRIP = "no";
           /* Using customized kernel config */
           NVT_CFG_KERNEL_CFG = "";

           /*
            * ======= Linux for different code setting =========
            * [NVT_LINUX_SMP]
            * NVT_LINUX_SMP_ON
            * NVT_LINUX_SMP_OFF
            */
           NVT_LINUX_SMP = "NVT_LINUX_SMP_ON";

           /*
            * [NVT_DEFAULT_NETWORK_BOOT_PROTOCOL]
            * NVT_DEFAULT_NETWORK_BOOT_PROTOCOL_DHCP_SERVER
            * NVT_DEFAULT_NETWORK_BOOT_PROTOCOL_DHCP_CLIENT
            * NVT_DEFAULT_NETWORK_BOOT_PROTOCOL_STATIC_IP
            */
           NVT_DEFAULT_NETWORK_BOOT_PROTOCOL = "NVT_DEFAULT_NETWORK_BOOT_PROTOCOL_STATIC_IP";

           /*
            * [NVT_ROOTFS_TYPE]
            * NVT_ROOTFS_TYPE_NAND_UBI
            * NVT_ROOTFS_TYPE_NAND_SQUASH
            * NVT_ROOTFS_TYPE_NAND_JFFS2
            * NVT_ROOTFS_TYPE_NOR_SQUASH
```

```
 * NVT_ROOTFS_TYPE_NOR_JFFS2
 * NVT_ROOTFS_TYPE_RAMDISK
 * NVT_ROOTFS_TYPE_EMMC
 */
NVT_ROOTFS_TYPE = "NVT_ROOTFS_TYPE_NAND_UBI";


/*
 * [NVT_ETHERNET]
 * NVT_ETHERNET_NONE
 * NVT_ETHERNET_EQOS
 */
NVT_ETHERNET = "NVT_ETHERNET_EQOS";


/*
 * [NVT_SDIO_WIFI]: Remember to update root-fs/rootfs/etc/init.d/S05_Net
 * NVT_SDIO_WIFI_NONE
 * NVT_SDIO_WIFI_RTK
 * NVT_SDIO_WIFI_BRCM
 * NVT_SDIO_WIFI_NVT
 */
NVT_SDIO_WIFI = "NVT_SDIO_WIFI_NONE";


/*
 * [NVT_USB_WIFI]
 * NVT_USB_WIFI_NONE
 */
NVT_USB_WIFI = "NVT_USB_WIFI_NONE";


/*
 * [NVT_USB_4G]
 * NVT_USB_4G_NONE
 */
NVT_USB_4G = "NVT_USB_4G_NONE";


/*
 * [WIFI_RTK_MDL]  : sub item for NVT_SDIO_WIFI_RTK
```

```
         * WIFI_RTK_MDL_NONE
         * WIFI_RTK_MDL_8189
         */
        WIFI_RTK_MDL = "WIFI_RTK_MDL_8189";

        /*
         * [WIFI_BRCM_MDL]  : sub item for NVT_SDIO_WIFI_BRCM
         * WIFI_BRCM_MDL_43438a1_ampk6212axtal26
         * WIFI_BRCM_MDL_43455c0_ampk6255c0
         */
        WIFI_BRCM_MDL = "WIFI_BRCM_MDL_43438a1_ampk6212axtal26";

        /*
         * [WIFI_NVT_MDL]  : sub item for NVT_SDIO_WIFI_NVT
         * WIFI_NVT_MDL_18202
         * WIFI_NVT_MDL_18211
         */
        WIFI_NVT_MDL = "WIFI_NVT_MDL_18211";

        /*
         * [NVT_CURL_SSL]
         * NVT_CURL_SSL_OPENSSL
         * NVT_CURL_SSL_WOLFSSL
         */
        NVT_CURL_SSL = "NVT_CURL_SSL_OPENSSL";

        /*
         * [NVT_UBOOT_ENV_IN_STORG_SUPPORT]
         * NVT_UBOOT_ENV_IN_STORG_SUPPORT_NAND
         * NVT_UBOOT_ENV_IN_STORG_SUPPORT_NOR
         * NVT_UBOOT_ENV_IN_STORG_SUPPORT_MMC
         * NVT_UBOOT_ENV_IN_STORG_SUPPORT_OFF
         */
        NVT_UBOOT_ENV_IN_STORG_SUPPORT = "NVT_UBOOT_ENV_IN_STORG_SUPPORT_OFF";
    };
```

3 Build U-boot

3.1 Compilation

The Uboot source code is placed on “na51000_linux_sdk/u-boot”, typing “make uboot” can be used to compile Uboot, and we provide two images are u-boot.bin(non-compressed) and u-boot.lz.bin(compressed) under na51000_linux_sdk/output.

U-boot build:

```
$ cd na51000_linux_sdk/
$ make uboot
```

U-boot config:

```
$ cd na51000_linux_sdk/
$ make uboot_config
```

U-boot clean build:

```
$ make uboot_clean
```

3.2 User customization

The uboot have two config files will have different configured properties. One is for common function, the other is for the low level setting.

The low level setting can modify this file “include/configs/nvt-na51000-evb.h” directly when you have request for the Uboot related configuration.
e.g. Uboot passed to Linux’s cmdline can be changed with this variable

```
#define CONFIG_BOOTDELAY            1
#define CONFIG_BOOTARGS_COMMON      "earlyprintk console=ttyS0,115200 rootwait "
#define CONFIG_BOOTARGS             CONFIG_BOOTARGS_COMMON "root=/dev/ram0 rootfstype=ramfs
rdinit=/linuxrc "
```

Please reference to “U-Boot_User_Guide” for more details.

4 Build Kernel Code

4.1 Compilation

The Linux kernel source code is placed on “na51000_linux_sdk/linux-kernel”, typing “make linux” can be used to compile Linux kernel, and the image name is uImage.bin under na51000_linux_sdk, this is an Uboot format Linux kernel image (uImage).

```
$ cd na51000_linux_sdk/
$ make linux
```

Linux clean build:

```
$ make linux_clean
```

4.2 System configuration

4.2.1 Menu configuration

Top Makefile is already integrated the formal Linux menuconfig, to use the following instruction can do the function selection. Please avoid using formal Linux “make menuconfig” under linux-kernel directly; it will cause error because the important variables are not set correctly.

```
$ make linux_config
```

Choose “Save” after finished function selection; it can generate a new *.config* for the Linux compilation usage.

NA51000 SDK provides two Kernel configuration file under “NA51000_BSP/linux-kernel/arch/arm/configs/”, one is for the debug mode, and the other is for the release mode.

- na51000_XXX_defconfig_debug, the debug mode will enable most of the functions for the development stage

- na51000_XXX_defconfig_release, the release mode will only enable boot necessary parts.

Edit Top Makefile to switch the configuration file:

$ cd na51000_linux_sdk/

$ vi Makefile

```
BOARDCONFIG := $(shell if [ ! -z $(CUSTBOARDCONFIG) ]; then echo $(CUSTBOARDCONFIG)_debug; else
echo na51000_evb_defconfig_debug; fi)
#BOARDCONFIG := $(shell if [ ! -z $(CUSTBOARDCONFIG) ]; then echo $(CUSTBOARDCONFIG)_release;
else echo na51000_evb_defconfig_release; fi)
```

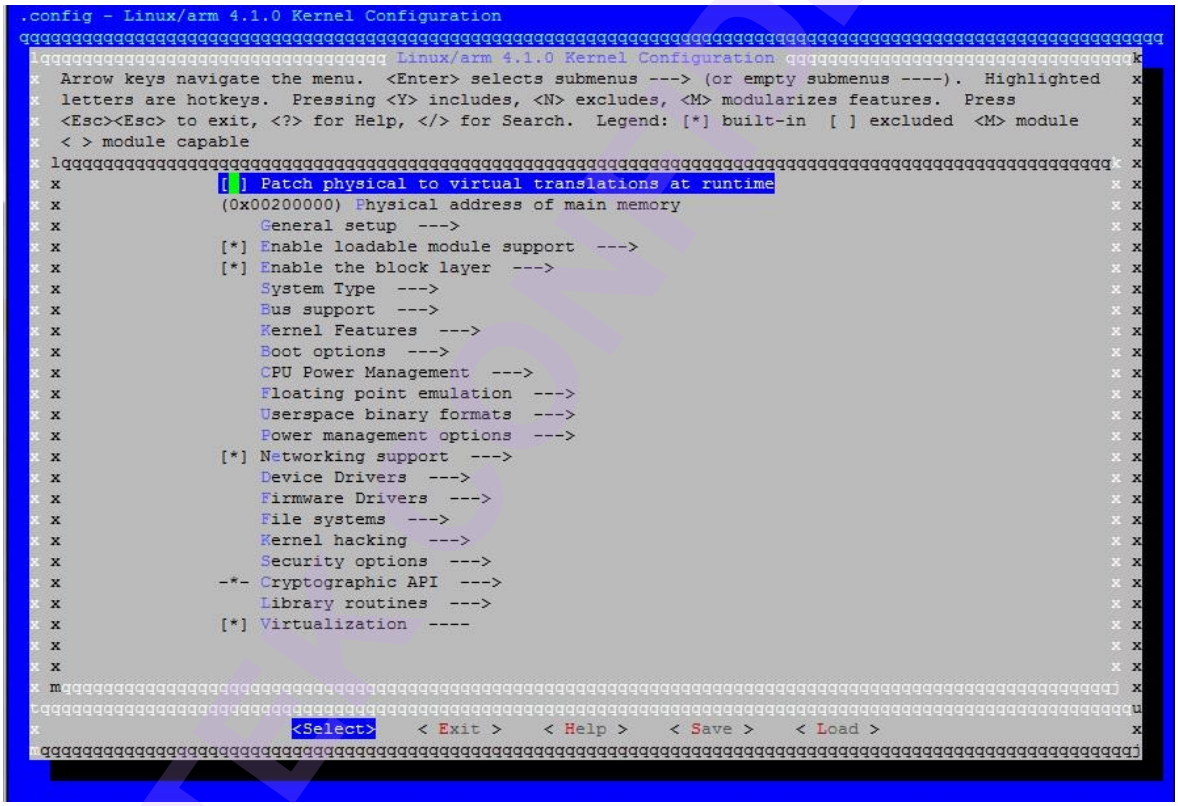
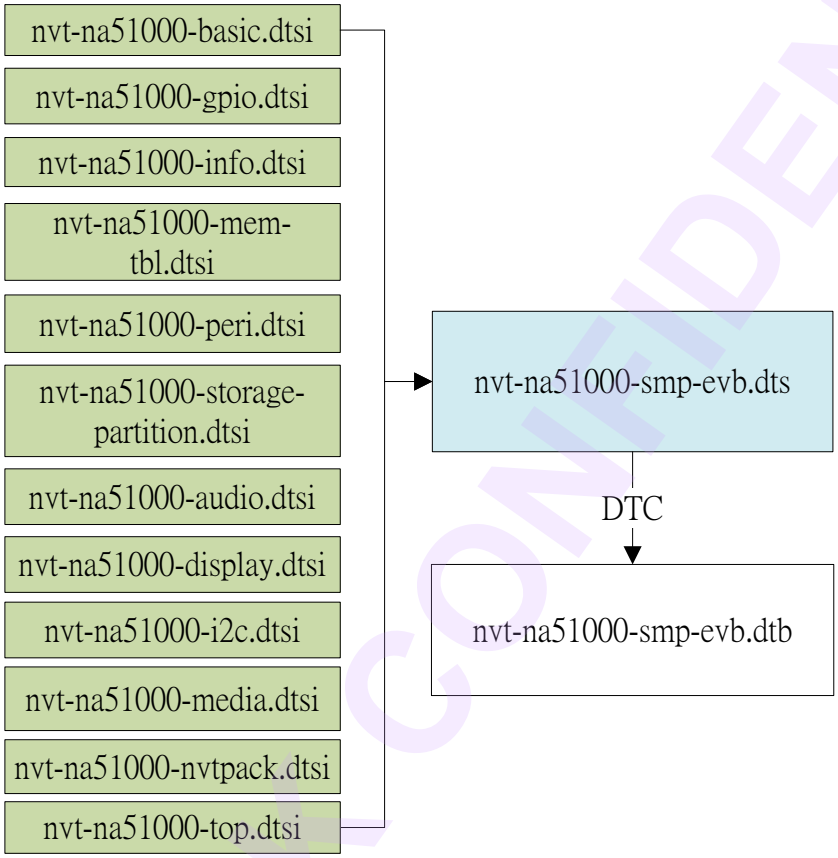


Figure 4-1 Menu configuration

4.2.2 Device tree

The device tree can be generated by dtc(device tree compiler) tool as below, you can find the *.dtsi in “na51000_linux_sdk/configs/cfg_xxx” and users can set command “make cfg” at “na51000_linux_sdk/”. It can be used to compile device tree, and we will output image nvt-na51000-smp-evb.dtb under na51000_linux_sdk/output.



The merge method is Union, for example:

configs/cfg_gen/nvt-na51000-peri.dtsi for commandline

```
#include <dt-bindings/gpio/nvt-gpio.h>
#include "nvt-na51000-basic.dtsi"

/ {
	chosen {
		bootargs = "quiet ";
	};

	aliases {
		mmc0 = <&mmc0>; /* Fixed to mmcblk0 for sdio1 */
		mmc1 = <&mmc1>; /* Fixed to mmcblk1 for sdio2 */
	};

	uart@f0290000 {
		compatible = "ns16550a";
		reg = <0xf0290000 0x1000>;
		interrupts = <GIC_SPI 43 IRQ_TYPE_LEVEL_HIGH>;
		baud = <115200>;
		reg-shift = <2>;
		reg-io-width = <4>;
		no-loopback-test = <1>;
		clock-frequency = <24000000>;
	};
```

configs/cfg_gen/nvt-na51000-peri.dtsi

```
            uart@f0290000 {
            compatible = "ns16550a";
            reg = <0xf0290000 0x1000>;
            interrupts = <GIC_SPI 43 IRQ_TYPE_LEVEL_HIGH>;
            baud = <115200>;
            reg-shift = <2>;
            reg-io-width = <4>;
            no-loopback-test = <1>;
            clock-frequency = <24000000>;
        };
```

4.2.3 Partition table

The Nand/EMMC driver of kernel will do for the set partition initialization process as shown below, Uboot will read the partition info(nvt-na51000-storage-partition.dtsi) from dts, and then the Kernel also read it from dts to do basic initialization.

If you want to change flash partition size or add a partition, please configure device tree nvt-na51000-storage-partition.dtsi and build dtb image, burn into flash or load to dram again(default address at 0x1f00000).

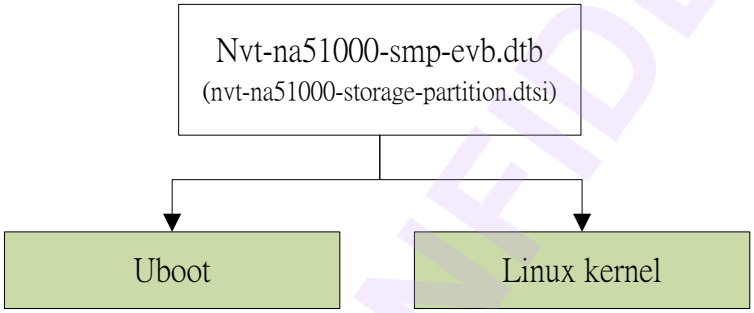


Figure 4-2 Partition initialization

SPI NAND partition example:

nvt-na51000-storage-partition.dtsi for flash partition

```
&nand {
	partition_loader {	label = "loader";	reg = <0x0 0x0000000 0x0 0x40000>; }; /*
Fixed */
	partition_fdt {	label = "fdt";	reg = <0x0 0x40000	0x0 0x40000>; }; /* Fixed
*/
	partition_fdt.restore { label = "fdt.restore";  reg = <0x0 0x80000	0x0 0x40000>; }; /*
Fixed */
	partition_uboot {	label = "uboot";	reg = <0x0 0xC0000	0x0 0x200000>; };
	partition_uenv {	label = "uenv";	reg = <0x0 0x2C0000	0x0 0x40000>; };
	partition_kernel {	label = "kernel";	reg = <0x0 0x300000	0x0 0x00400000>; };
	partition_rootfs {	label = "rootfs";	reg = <0x0 0x700000	0x0 0x3200000>; };
	partition_rootfs1 {	label = "rootfs1";	reg = <0x0 0x3900000 0x0 0x4700000>; };
};
```

If you want to change the partition size or add a partition, please configure it from the dts info.

4.3 Debug

To debug the Kernel, you will need System.map or objects for the debug symbol loading, please get them from “linux-kernel/” as shown below:

```
|-- linux-kernel
|       |-- Makefile
|       |-- Module.symvers
|       |-- System.map
|       |-- arch
|       |-- block
|       |-- crypto
|       |-- drivers
|       |-- firmware
|       |-- fs
|       |-- include
|       |-- init
|       |-- ipc
|       |-- kernel
|       |-- lib
|       |-- mm
|       |-- modules.builtin
|       |-- modules.order
|       |-- net
|       |-- scripts
|       |-- security
|       |-- sound
|       |-- source
|       |-- usr
|       |-- vmlinux
|       `-- vmlinux.o
```

To add more debug information, we can turn on CONFIG_DEBUG_INFO option before compiling Linux kernel as below.

"Kernel Hacking > Compile-time checks and compiler options > Compile the kernel with debug info"

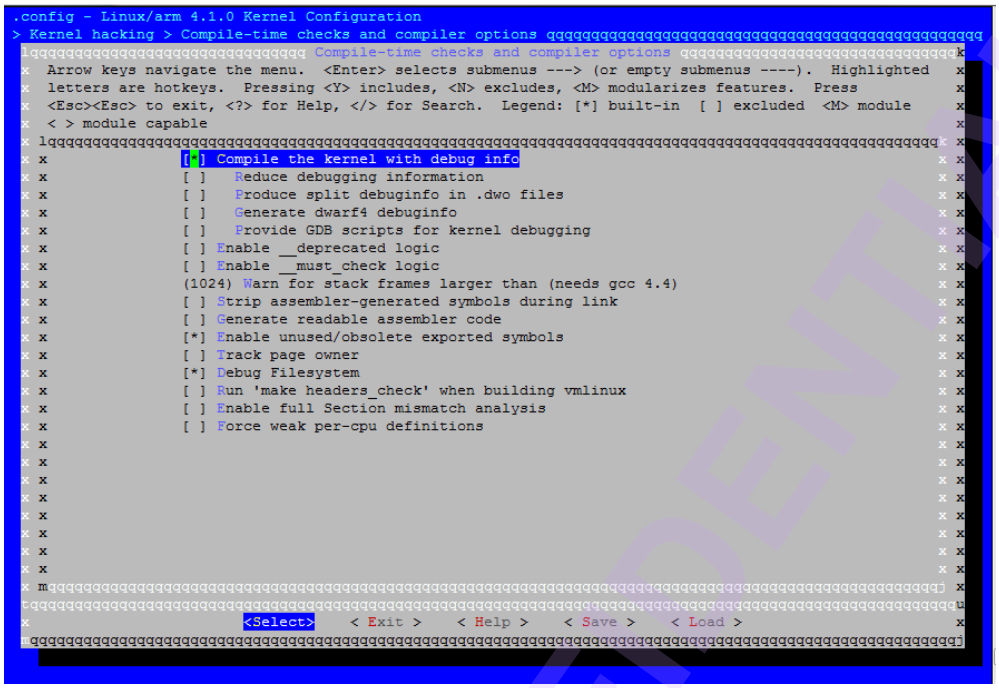
$ cd na51000_linux_sdk/

$ make linux_config

Rebuild all; this binary will contain debug symbol information.

|-- linux-kernel

| |-- vmlinux



5 Build Kernel Modules

In addition to build-in drivers, Linux had also provided an external load mode called Kernel module, can be loaded by “modprobe” or “insmod”. The below will introduce to build-in modules and the out-of-tree modules.

5.1 Build-in module compilation

Typing “make modules” under na51000_linux_sdk can do build-in modules compilation and install path is root-fs/rootfs/lib/modules/4.1.0/kernel/.

$ cd na51000_linux_sdk/

$ make modules

5.2 Out-of-Tree module compilation

This folder under “na51000_linux_sdk/code” provides NVT platform drivers, typing “make supplement” to do the Out-of Tree modules compilation as below. And the modules will be installed on na51000_linux_sdk/BSP/root-fs/rootfs/lib/modules/4.1.0/extra/.

```
$ cd na51000_linux_sdk/
$ make linux_driver
```

Linux Out-of-Tree driver module clean build:

```
$ make linux_driver_clean
```

5.3 HDAL

HDAL is the hardware abstraction layer driver and sample code, this one can be used to build proprietary driver modules, such as video process, video capture and video codec…etc. We provide the following instructions to build.

```
$ make hdal
```

Clean build:

```
$ make hdal_clean
```

Please refer to the other documents for details.

5.4 Installation

The modules can be installed by “modprobe” or “insmod/rmmod” to install or uninstall, besides the modprobe will also install related modules automatically.
Example:

1. modprobe (Only needs the module name)
   modprobe ehci-hcd

2. insmod/rmmod (This method needs a full path)

```
insmod /lib/modules/4.1.0/extra/net/synopsys/ntkimethmac.ko
```

6 Build tools

This folder will integrate some Linux open source tools as the following description:

htop	- Linux process monitoring, to provide more than top information
gdb	- gdb server for the application debug
ethtool	- Utility for controlling network drivers and hardware
bonnie++	- A benchmark suite that is aimed at performing a number of simple tests of hard drive and file system performance
memtester	- A userspace utility for testing the memory subsystem for faults
mtd-utils	- MTD device utilities
procps	- A tool set to provide system analysis tools (vmstat, slabtop,…etc)
stress	- System performance testing tool
stress-ng	- Advanced system performance testing tool
sdcard_test.sh	- To do sd driver r/w testing

6.1 Compilation

To select the tools what you want: mtd-utils, memtester, bonnie, ethtool, gdb, htop, netperf, iperf and procps and running below instructions:

e.g.

```
$ cd na51000_linux_sdk/
$ cd tools/
$ make stress
```

Tools clean build:

```
$ make clean
```

6.2 Installation

```
$ make install
```

The tools will be installed on “na51000_linux_sdk/root-fs”.

7 Build root-fs

7.1 Introduction

The root file system will be mounted by Linux kernel, the first process of the kernel is /sbin/init (PID=1). We provide several root file systems support for selection, the following are the summary features:

- UBIFS: readable/writable file system, support Nor and Nand flash, fast mounting speed, best bad block management and better IO performance. Suitable for low memory size and large flash size use condition.
- Squashfs: read-only file system, high compression rate. Suitable for small size flash and readonly use condition.
- JFFS2: It is a log-structured file system which can support Nand and Nor flash devices. Providing zlib, lzo and rtime compression methods. Suitable for small flash size and readable/writable use condition.

7.2 Configuration

The root file system will generate Nand flash type image format, we support squashfs and ubifs, please follow the Nand flash specification to modify the parameters:

$ cd na51000_linux_sdk/BSP/root-fs/

$ vi ubi_max_leb.py

To find below lines:

```
W = 1000
SP = 128 * 1024
SL = 124 * 1024
```

According the Nand flash to modify the parameters:

- W is the entire flash chip Physical eraseblocks numbers

- SP is the Size of block page
- SL= (Size of block -2) * Size of page.

e.g.

128MB Nand flash = 1000 eraseblocks = 1000 * 128KB

W = 1000

SP = 64 * 2048 = 128 * 1024

SL = (64 – 2) * 2048 = 124 * 1024

$ vi mtd_cfg.txt

To find below lines:

```
ROOTFS_UBI_SUB_PAGE_SIZE=2048                # Same as page size
ROOTFS_UBI_PAGE_SIZE=2048                    # Nand page size
ROOTFS_UBI_ERASE_BLK_SIZE=126976             # (64-2) * Page size=126976
ROOTFS_UBI_MAX_LEB_COUNT=608                 # Size = UBI_MAX_LEB_COUNT * UBI_BLK_SIZE; It's
calculated by "python ubi_max_leb.py Bytes"
ROOTFS_UBI_RW_MAX_LEB_COUNT=271              # Size = UBI_MAX_LEB_COUNT * UBI_BLK_SIZE; It's
calculated by "python ubi_max_leb.py Bytes"
ROOTFS_UBI_BLK_SIZE="128KiB"                 # UBIFS Nand flash block size (KiB)
ROOTFS_UBI_COMPRESS_MODE="lzo"               # UBIFS compression type: "lzo", "favor_lzo",
"zlib" "none"

ROOTFS_SQ_COMPRESS_MODE="xz"                 # Squashfs compression type: "gzip", "lzo" and "xz"
ROOTFS_SQ_BLK_SIZE="128K"                    # Squashfs Nand flash block size (KiB): e.g.
spinand: 128K, spinor: 64K

ROOTFS_JFFS2_COMPRESS_MODE="lzo"             # jffs2 compression type: "lzo" "zlib" "rtime"
ROOTFS_JFFS2_SIZE=0x03200000                 # jffs2 partition size: get from /proc/mtd
ROOTFS_JFFS2_RW_SIZE=0x4700000               # jffs2 partition size: get from /proc/mtd
ROOTFS_JFFS2_BLK_SIZE="128KiB"               # jffs2 block size (KiB): spinand: 128KiB, spinor:
64KiB
ROOTFS_JFFS2_PAGE_SIZE="2048"                # jffs2 page size (Bytes): only used by nand, nor
flash can be ignored.

ROOTFS_EXT4_SIZE=$(shell printf "%d\n" 0x04000000)
ROOTFS_FAT_CACHE_SIZE=$(shell printf "%d\n" 0x05000000)
```

The necessary parameters need to be modified as below description:

- ROOTFS_UBI_SUB_PAGE_SIZE: The sub-page size of the Nand flash
- ROOTFS_UBI_PAGE_SIZE: The page size of the Nand flash
- ROOTFS_UBI_ERASE_BLK_SIZE: (Nand flash block size – 2) * Page size
- ROOTFS_UBI_MAX_LEB_COUNT: Use ubi_max_leb.py to calculate it
  $ Usage: ubi_max_leb.py PartitionSize (Bytes)
- ROOTFS_UBI_RW_MAX_LEB_COUNT: Use ubi_max_leb.py to calculate it
  $ Usage: ubi_max_leb.py PartitionSize (Bytes)
- ROOTFS_UBI_BLK_SIZE: Nand flash block size
- ROOTFS_UBI_COMPRESS_MODE: Compression method = LZO
- ROOTFS_SQ_COMPRESS_MODE: Squashfs compression mode
- ROOTFS_SQ_BLK_SIZE: Squashfs nand flsh block size
- ROOTFS_JFFS2_COMPRESS_MODE: jffs2 compression type: "lzo" "zlib" "rtime"
- ROOTFS_JFFS2_SIZE: Rootfs partition size

7.3 Compilation

Using “make rootfs” instruction to generate rootfs bin, the image type can be selected by nvt-na51000-info.dtsi in configs folder. They can be produced into “na51000_linux_sdk/output/rootfs.ubifs.bin”, “na51000_linux_sdk/output/rootfs.squash.bin” and “na51000_linux_sdk/output/rootfs.jffs2.bin” separately. The command “mr” also can be used to compile rootfs if you are not in na51000_linux_sdk root folder.

```
$ cd na51000_linux_sdk/
$ source build/envsetup.sh
$ make rootfs
```

Rootfs clean build:

```
$ make rootfs_clean
```

This command will remove busybox tools, kernel modules…etc., please follow below procedure to generate rootfs image:

```
$ make busybox
$ make app ($ make library if necessary)
```

```
$ make hdal
$ make rootfs
```

7.4 Folder description

- **Architecture**

Folder	Description
bin	User binaries
dev	Device files
etc	System management configuration files
home	User home directories
init -> bin/busybox	It is used to kernel boot necessary init process, for the initial environment setup.
lib	Standard system libraries
linuxrc -> bin/busybox	
mnt	External storage device mount folder (/mnt/sd, /mnt/sd2)
proc	RAM based FS to provide process information
root	Root's folder (The default shell will login here)
sbin	System management binaries
srv	Service data
sys	RAM based FS to provide user space and kernel space attribute/properties link.
tmp	RAM based temp folder
usr	User libraries, binaries
var	Service log message, including kernel, application, web server default folder(/var/www) and service…etc

- **/etc/passwd**

This file can setup user account environment, below is to introduce how to enable login password.

$ vi /etc/inittab

```
::respawn:-/bin/login
```

Replace "::respawn:-/bin/login -f root" with "::respawn:-/bin/login" as below

Fill in the red part with the encryption password which can be generated by openssl tool:

$ vi /etc/passwd

```
root:EncryptionCode:0:0:root:/root:/bin/sh
```

Openssl generation:

$ openssl passwd -crypt YourPWD

- **/etc/init.d**

  System will execute the following shell scripts according sequence.

  rcS -> S00_PreReady -> S05_Net -> S10_SysInit -> S60_LogDaemon -> S90_Nvtwatchdog -> S99_Sysctl

  Moreover, power off will execute deinitialization process as below.

  rcK -> K00_Sys -> K20_LogDaemon -> K99_Sys

- **/etc/sysctl.conf**

  This file is handle sysctl parameters setup.

7.5 UBIFS

The UBIFS is our default rootfs format, UBIFS (Unsorted Block Image File System) was originally called JFFS3, is JFFS2 next generation version. The main capabilities are faster mounting, quicker access to large files, and improved write speeds. UBIFS also preserves or improves upon JFFS2's on-the-fly compression, recoverability and power fail tolerance, and data compression allows zlib or LZO. The filename is UBI.IMG after compilation.

7.6 Squashfs

Squashfs is a read-only file system which can support gzip, lzo and xz compression modes. The main features are high compression rate, stores full 32bits uid/gids and creation time, support block size up to 1Mbytes. The filename is SQ_ROOTFS.IMG after compilation.

7.7 Jffs2

JFFS2 was developed by Red Hat, based on the work started in the original JFFS by Axis Communications, AB, it is a readable and writable file system. JFFS2 will scan rootfs partition during mounting; the mount time depends on the rootfs size. The main features are listed as below:

- Support compression mode
- Mounting time will be affected by flash size
- Not support all Nand flash devices with HW ecc, please refer to Linux driver application note

The filename is JFFS2_ROOTFS.IMG after compilation.

For the kernel configuration to add jffs2 support, you must add below configurations:
File systems -> Miscellaneous filesystems -> [*] Journalling Flash File System v2 (JFFS2) support -> [*] Advanced compression options for JFFS2 -> [*] JFFS2 LZO compression support

For the uboot configuration to choose root file system type, please refer to the UBoot_Programing_Guide.

8 Build App

8.1 Compilation

NVT platform needs the necessary applications to perform the requested actions, please using below instructions to compile (This part doesn't provide source code).

```
$ cd NA51000_BSP/application/
$ make install
```

Please execute "source build/envsetup.sh" firstly when you start to build it. Please reference to Application Note for the other details.

We have fine-tuned memcpy and memset functions, please add $(PLATFORM_CFLAGS) to your Makefile if you want to use it.

9 Build Libraries

9.1 Compilation

na51000_linux_sdk provides some proprietary libraries and header files for the product customization (this part doesn't involve source code, we only provide you *.so), please according the following instruction to compile it.

```
$ cd na51000_linux_sdk/
$ make library
```

Libraries clean build:

```
$ make library_clean
```

Please execute "source build/envsetup.sh" firstly when you start to build it. Please reference to Application Note for the other details.

10 Build busybox

10.1 Compilation

Busybox can provide rootfs necessary tools, using below instruction can compile it. And the tools will be installed to na51000_linux_sdk/BSP/root-fs.

$ cd na51000_linux_sdk/
$ make busybox

mybusybox clean build:

$ make busybox_clean

10.2 Menu configuration

SDK will provide two busybox configuration files, one is normal version (busybox_cfg_normal), and the other is minimized version (busybox_cfg_small). Edit Top Makefile can change the busybox configuration, the default is normal version.

Edit Top Makefile to switch the configuration file:

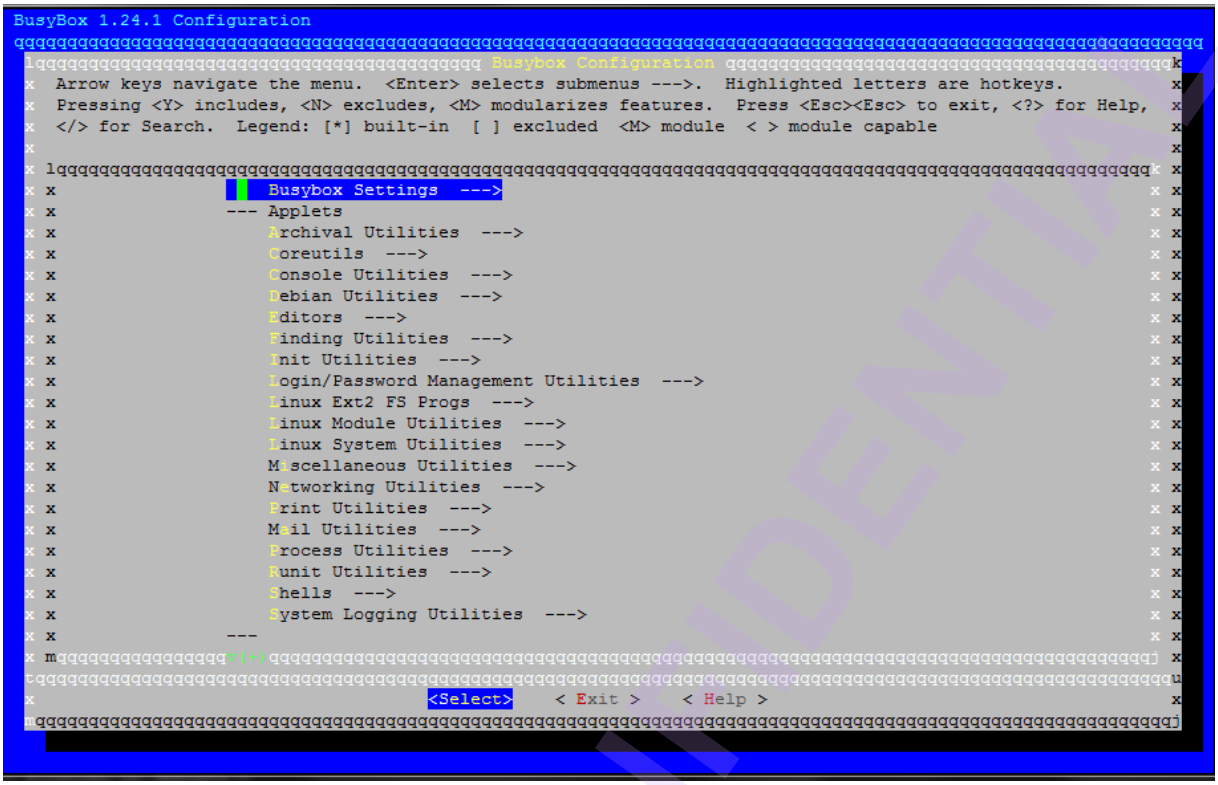
$ cd na51000_linux_sdk/
$ vi Makefile

```
BUSYBOX_CFG:=busybox_cfg_normal
#BUSYBOX_CFG:=busybox_cfg_small
```

Below instruction can handle busybox features selection:

$ make busybox_config

Choose “Exit/Save” after you finished function selection, it can generate new .config for the Busybox compilation usage.



11 Build sample code

11.1 Compilation

The device driver testing applications will be used to test NVT peripheral devices; the following instructions can compile it. And it will be installed to na51000_linux_sdk/BSP/root-fs.

```
$ cd na51000_linux_sdk/
$ make sample
```

Driver test clean build:

```
$ make sample_clean
```

12 Update Firmware

We provide two OS version nvtpack image could be used, one is Linux version will be generate by Linux version nvtpack tool, the images will be generated under the output folder. Another is Windows vesion nvtpack tool, the following section can get more clearly description.

12.1 Linux version nvtpack

The output/FW(SOC).ini is used to control what kinds of images want to be involved.

```
[NVTPACK_FW_INI_16072017]
GEN packed/FW96687A.bin
CHIP_SELECT 1
ITEM00 0
ITEM01 1 nvt-na51000-smp-evb.dtb
ITEM02 0
ITEM03 1 u-boot.bin
ITEM04 0
ITEM05 1 uImage.bin
ITEM06 1 rootfs.ubifs.bin
ITEM07 1 rootfs_1.rw.ubifs.bin
ITEM08 0
ITEM09 0
```

Using “make all” or “make pack” can generate the packed image is under output/packed/.

12.2 Windows version

Partition sequence depends on emb_partition_info.c (is reffered to *Setup embedded flash partition* in NT96680 SDK Introduction) . The BSP defulat partition is fixed as below:

[00] Loader
[01] dtb
[02]
[03] Uboot
[05] Linux kernel
[06] Rootfs image (RO): ubifs/jffs2/squashfs…
[07] Rootfs image (RW)

12.3 Using NvtPack.exe to pack All-in-One bin

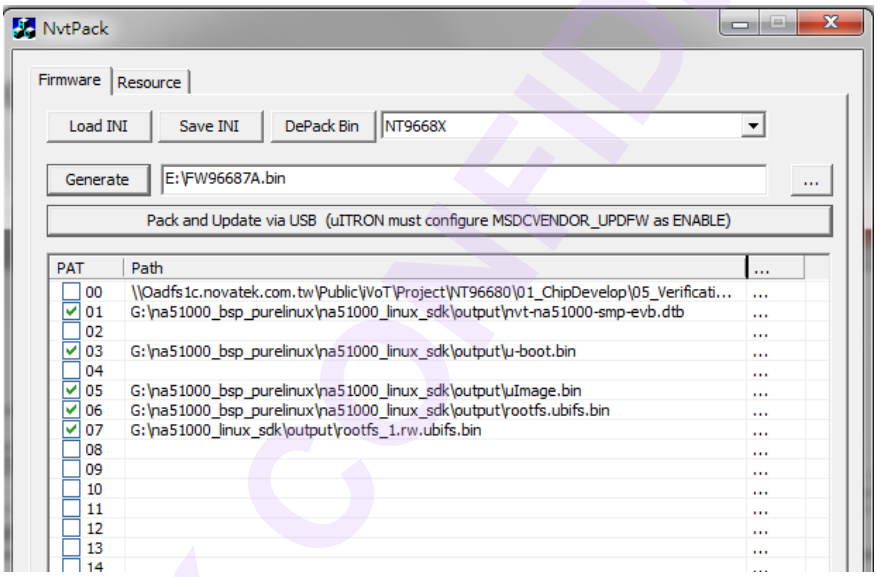


Figure 11-1 NvtPack PC tool

Each Partition using the above figure [...] setting the path, then “Save INI” will store the environment configuration. The Generate field is the output image path, usually set to SD card root. Update all after the first time, you can choose the field what you want to update, tick this check Box only, and then press Generate.

12.4 Update Firmware

Insert SD card including All-in-One bin to the target board and power on can update firmware.

12.5 Update Loader

For a blank Nand flash, you need to burn loader (LD96687A.bin), by first SD card format (be sure to format), and then immediately put LD96687A.bin, then placed all in one bin (FW96687A.bin).

13 Power on

13.1 How to power on

To follow update steps to burn the desired Image, remove SD card can boot up directly.

14 Debug

The following list provides comparison and classification of the debugging tools.

Name	Classification	Description
Coredump	AP debug	Generated file for the further analysis when the program has terminated abnormally
Messages	AP/Kernel debug	To record Linux kernel and AP booting log
GDB	AP debug	To debug target board application from remote server
printk	Kernel debug	Basic kernel/module debug usage
kmemleak	Kernel debug	To analyze if Linux kernel has memory leak issue.
OPENOCD	Kernel debug	To debug/ trace kernel or uboot

14.1 Coredump

Provide analytical application error log, the application does not properly terminated, it generates a file in /var/log. It can record the program name, PID and time, can be loaded for analysis through a cross compiler. You should build it with debug mode when you start application analysis.

The following is related setting:

$ cd na51000_linux_sdk/root-fs/rootfs/

$ vi etc/profile

```
# coredump setting
echo 1 > /proc/sys/kernel/core_uses_pid
ulimit -c unlimited
echo "/var/log/core-%e-%p-%t" > /proc/sys/kernel/core_pattern
```

14.2 Messages

The boot log of the Linux will be stored in /var/log/messages, this file can involve Kernel

and user space app. If the kernel crash occurred, please provide this file for the further analysis.

14.3 GDB

GDB (GNU Project Debugger) can support Remote and Target mode to debug AP.

Target mode gdb/gdbserver can be generated by the following command:

```
$ cd na51000_linux_sdk/tools
$ make gdb
```

The connection diagram as shown below, Figure 13-1:

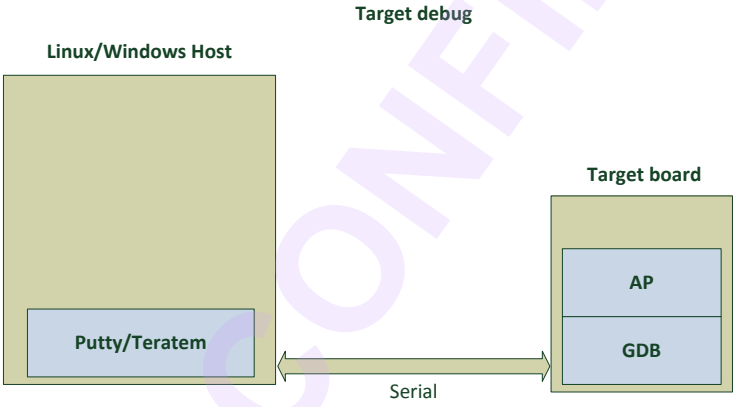


Figure 14-1 Target debug connection

Remote mode can debug user space application through GDB server, and the connection architecture as below. Linux server is x86_64 compile server, target board is the EVB, they can be connected by serial or TCP/IP. Linux server will use cross compiler toolchain GDB to debug target board AP, this AP must be enabled debug symbol, and target board also needs to execute gdbserver which can be find in toolchain.

The serial connection can connect USB-to-Serial cable to target board USB port, and check if there is /dev/ttyUSB0 existed.

TCP/IP connection can use Wi-Fi or Ethernet, install necessary drivers and confirm whether it can ping to server.

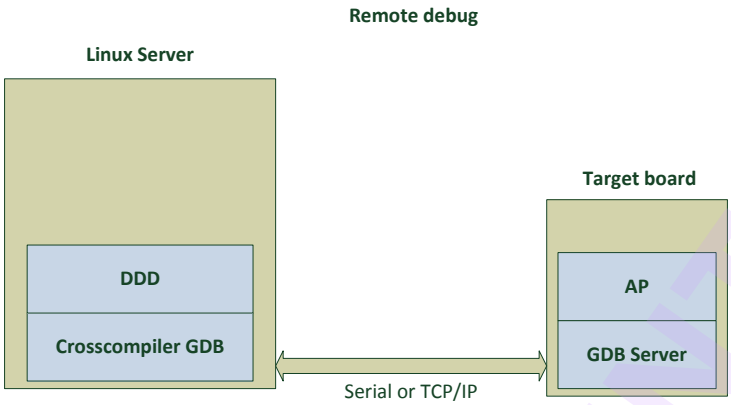


Figure 14-2 Remote debug connection

GDB server needs to be executed on EVB.

```
$ cp tools/__install/bin/gdbserver  root-fs/rootfs/bin/
$ cp tools/__install/bin/gdb  root-fs/rootfs/bin/
```

And then, running below procedures can debug your AP.

1. Target Board

   target > gdbserver comm prog [args...]

   The gdbserver doesn't loading debug symbol, all of the symbols will be loaded by the Linux server cross compiler gdb. It can reduce memory space in this way.

   Serial:

   ```
   target > gdbserver /dev/ttyUSB0  hello_world
   ```

   Net:

   ```
   target > gdbserver Host _IP:1234  hello_world
   ```

2. Linux Server

   Serial:

   ```
   Server > {CROSS_COMPILE}-gdb  hello_world
   Server > set remotebaud  115200
   Server > target remote /dev/ttyUSB0
   ```

   Net:

   ```
   Server > {CROSS_COMPILE}-gdb  hello_world
   Server > target remote localhost:1234
   ```

Reference to below link can show you how to use command debug your AP:
http://sourceware.org/gdb/current/onlinedocs/gdb/index.html

In addition to command mode debug you also can use DDD, it is a framework on top of GDB debug visualization software, you can install and use by below command:

```
$ sudo apt-get install ddd
$ sudo ddd --debugger arm-ca53-linux-uclibcgnueabihf-gdb
```

14.4 Printk

Linux provides seven levels of Log printk available in the following table:

Level	Description	Usage
(0) KERN_EMERG	system is unusable	pr_emerg
(1) KERN_ALERT	action must be taken immediately	pr_alert
(2) KERN_CRIT	critical conditions	pr_crit
(3) KERN_ERR	error conditions	pr_err
(4) KERN_WARNING	warning conditions	pr_warning
(5) KERN_NOTICE	normal but significant condition	pr_notice
(6) KERN_INFO	Informational	pr_info
(7) KERN_DEBUG	debug-level messages	pr_debug

Above the printk level is used to decide whether or not to print the message console, the below instruction can show you the printk level, current representative of the level of the boot to be printed, default is the default level, minimum is the lowest possible print level, boot-time-default is boot stage log:

```
root@NVTEVM:~$ cat /proc/sys/kernel/printk
                7       4       1       7
                current default minimum boot-time-default
```

Kernel will compare the printed message log level, if the value is less than the current will be printed out. Therefore, to change the output level so that all messages are printed out can use this command:

```
root@NVTEVM:~$ echo 8 > /proc/sys/kernel/printk
```

14.5 Kmemleak

Linux Kmemleak is provided for detecting a memory leak tool, it will record detect report in /sys/kernel/debug/kmemleak, to use this function as long as enable the "Kernel Hacking", "Kernel Memory Leak Detector" (CONFIG_DEBUG_KMEMLEAK) in the kernel option, and configure the "Maximum kmemleak early log entires" with 1200.

Clear current record:

```
root@NVTEVM:~$ echo clear > /sys/kernel/debug/kmemleak
```

Testing your driver:

```
root@NVTEVM:~$ insert YourModule.ko
```

Scan:

```
root@NVTEVM:~$ echo scan > /sys/kernel/debug/kmemleak
```

Check the resules:

```
root@NVTEVM:~$ cat /sys/kernel/debug/kmemleak
```

15 FAQ

This section will list frequently problems.

15.1 Toolchain can’t be found

We have two toolchains to build overall SDK, one is for itron, another is for Linux. First, please use below command to check your environment setting.
$ get_stuff_for_environment

15.2 Operation not permitted

The SDK installation path should be under your home folder, you should use the same owner to decompressing and building, otherwise you will get “*operation not permitted*” related message.

Please use below command to check permission and owner.
$ ls -al YOUR_FOLDER

15.3 Linux kernel uImage can’t be generated

This is because our default setting is lz4 compression format, you should follow section 1.2 to check lz4 tool is installed.

Try to use lz4 command to check your compiling environment.
$ lz4