NT96680 SDK Introduction

Table of Content

1 Introduction

1.1 SDK Overview

This Novatek NT96680 SDK provides a platform solution to shorten the development cycle of image-processing products. NT96680 has four cores, 2 ARM-CPUs and 2 DSPs. Core-1 always runs uITRON OS (codename is GS); Core-2 always runs on Linux. Core-3 and Core-4 run the freeRTOS OS.

To install the development tools, please refer to the document: ***NT96680 SDK Toolchain Setup.doc***

1.2 SDK Software Architecture (Block Diagram)

The four cores relationship is illustrated as Figure 1-2.

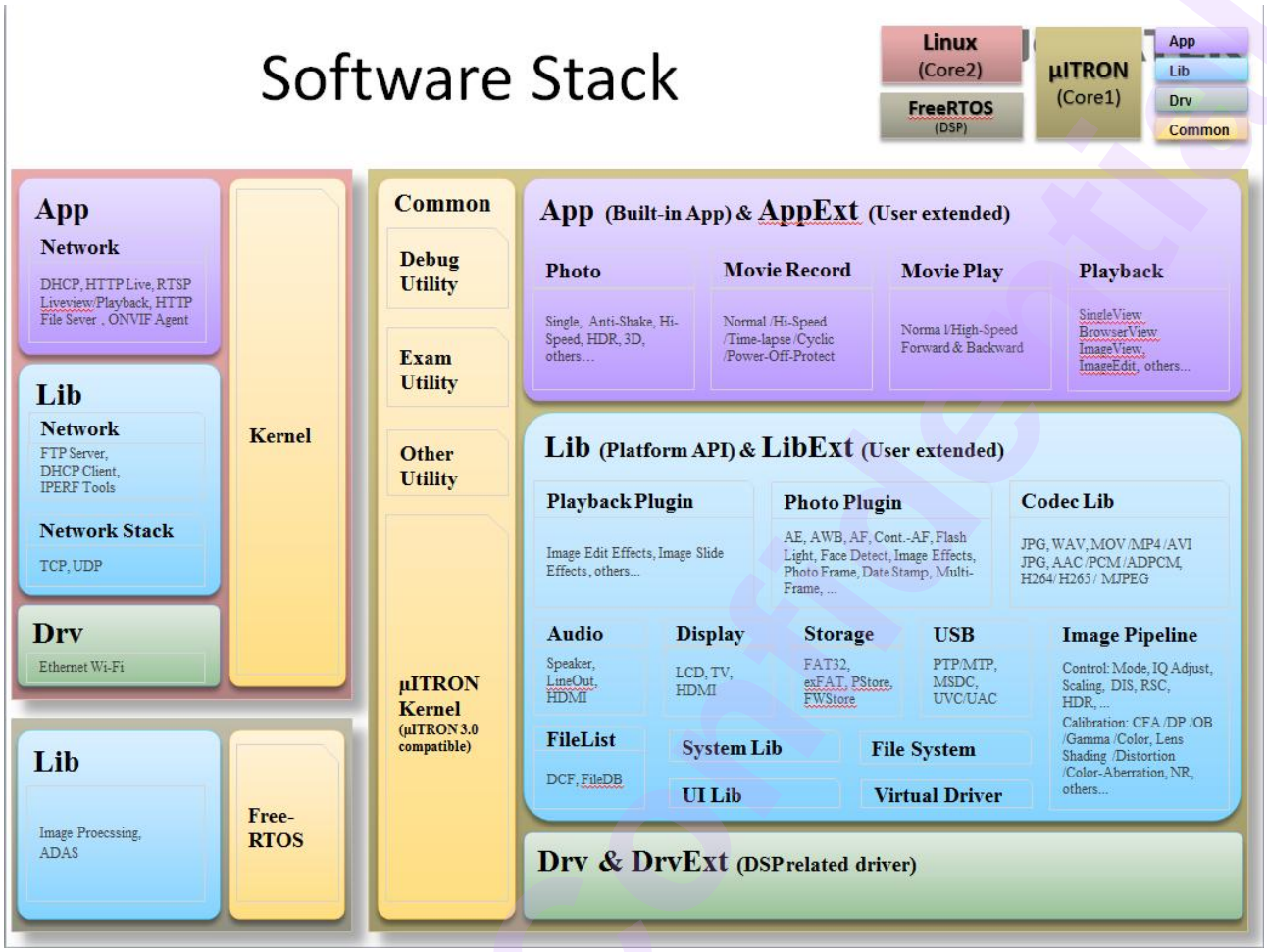


Figure 1-2 Software Stack of 4 cores

1.3 SDK Code Tree

The NT96680 project number is NA51000, so the SDK naming is NVT_NA51000_BSP. NVT_NA51000_BSP includes the uITRON & Linux code.

Using the following instructions to decompress SDK pack under Linux:

```
$ tar -jxvf NVT_NA51000_BSP_{DATE}.tar.bz2
```

You will get the folder tree as below:

```
|-- NA51000_BSP                    Used to put unpacked SDK source code
|    |-- Makefile                  Top level Makefile
|    `-- packages                  Source code packages
```

```
|-- bin                                                    Prebuild EVB image in this version
|     |-- FW(SOC)A.bin                                     uITRON image
|     |-- uImage.bin                                       Linux kernel image
|     |-- u-boot.bin                                       Uboot image
|     |-- rootfs.jffs2.bin                                 jffs2 rootfs image
|     |-- rootfs.squash.bin                                Squashfs rootfs image
|     `-- rootfs.ubifs.bin                                 UBIFS rootfs image (Default rootfs)
|-- toolchain                                              Cross Compiler Toolchain
|     |-- arm-ca53-linux-gnueabi-4.9-{DATE}.tar.bz2        ARM Linux glibc gnu 4.9 toolchain
|     `-- arm-ca53-linux-uclibcgnueabihf-4.9-{DATE}.tar.bz2 ARM Linux uclibc gnu 4.9 toolchain
|-- version                                                SDK version
|-- sdk.unpack                                             Unpack script
`-- nvt.md5                                                Source code packages md5 sum
```

The following instruction can install SDK source code:

```
$ sh sdk.unpack
```

After the installation process finished, the NA51000_BSP tree is listed as below:

```
|-- Makefile                Top level Makefile
|-- application             NVT platform application
|-- build                   scripts for the environment setup
|-- busybox                 it's used to generate rootfs basic instruction set
|-- sample                  Testing sample code
|-- include                 Proprietary header file for the customized AP
|-- lib                     Proprietary libraries for the customized AP
|-- linux-kernel            Support NVT platform Linux kernel (Version: 4.1.0)
|-- linux-supplement        Out-of-tree Linux kernel driver module
|-- root-fs                 Linux root file system
|-- tools                   Linux useful tools
|-- u-boot                  Bootloader
|-- uitron                  uITRON codebase
```

1.4 Commutation between Cores

The picture shown in Figure 1-3 is the way to commutate message between cores, named NVT IPC Framework. The general commutation flows is a situation that core-1 sends a message to core-2 which does something for core-1. Then after core-2 finish its job, notifies a message to core-1.

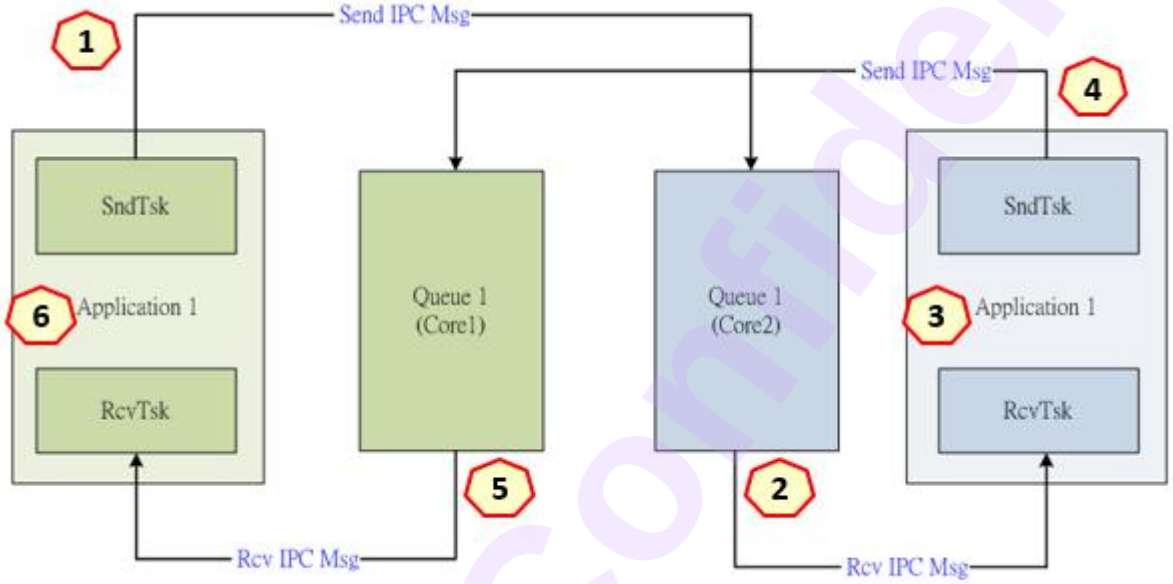


Figure 1-3 NVT IPC Framework

Each NVT-IPC unit has a sender to trigger the target ISR, a receiver that is an ISR handler, a working queue to dispatch the received message. It is shown in Figure 1-4.

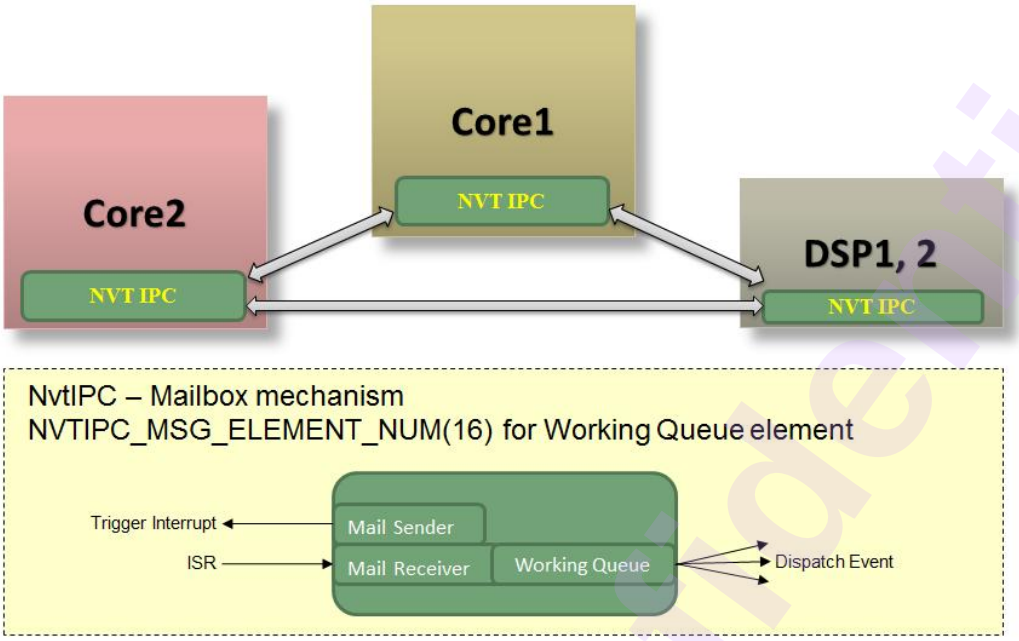
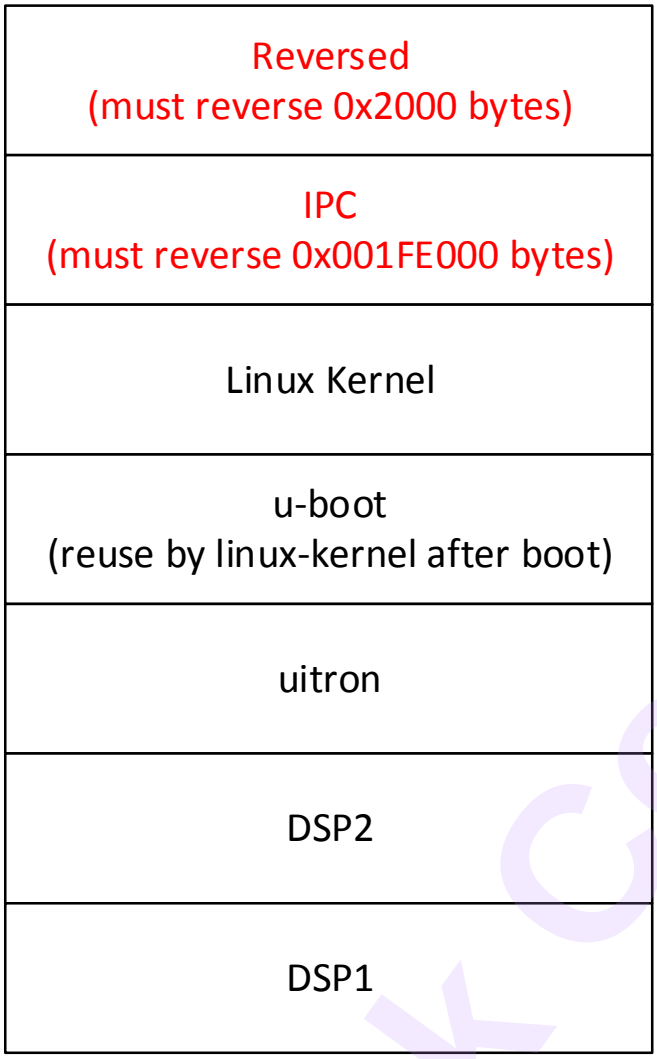


Figure 1-4 NVT IPC Mechanism

1.5 Dram partition



1.6 Flash partition

loader (must be 1st partition and reserve 0x4000 bytes for spi-nand)
modelext (must be 2nd partition)
uitron
u-boot
uenv (optional)
linux-kernel
DSP1 (optional)
DSP2 (optional)
PStore (optional)
root-fs
fat (optional)

2 Setup ModelExt

Some global information located in uitron/DrvExt/DrvExt_src/ModelExt/{MODEL_NAME} /independent. FW*.ext.bin is made from those c files. Those information are divided into eight parts listed below:

1. modelext_info.c: DO NOT EDIT IT. It reversed for a tag to indicate modelext.
2. bin_info.c: only “version (8)” and “releasedate (8)” for vendor to modify. Others are reversed to exchange data between loader, uitron, uboot.
3. pinmux_cfg.c: pinmux table to configure NT96680’s pinmux.
4. intdir_cfg.c: interrupt direction table to decide where engine’s interrupt send to.
5. emb_partition_info.c: flash partition table indicate how to divide the used flash.
6. gpio_info.c: set gpio’s direction (input / ouput) or initial state (pull up / pull down)
7. dram_partition_info.c: DRAM partition table, indicate where loader, uitron, u-boot, linux located.
8. model_cfg.c: DO NOT EDIT IT. It reversed to store MODELCONFIG_CFLAGS for debug.

Configure DRAM and flash are more complicated, that describe as following section,

2.1 Setup DRAM partition

Don’t edit dram_partition_info.c directly. Configure ModelConfig_$MODEL.txt instead to layout the DRAM partition. For example, like below,

```
# [BOARD DRAM PARTITION]
# 1. DO NOT EDIT the partitions of DRAM, REV, IPC, LOADER.
# 2. u-boot must be in the bottom of linux memory.
# 3. rebuild uITRON, u-boot, linux-kernel after changing partition.
# 4. iTron will reserve a region to the Linux system memory usage. (BOARD_UITRON_RESV_SIZE)
# 5. linux automatically use the memory from the bottom of uitron memory to the dram end if following
condition:
#       5.1 there is no dsp
```

```
#         5.2 there is a gap between the bottom of uitron memory and dsp starting address.
BOARD_DRAM_ADDR =   0x00000000
BOARD_DRAM_SIZE =   0x20000000
BOARD_REV_ADDR =    0x00000000
BOARD_REV_SIZE =    0x00002000
BOARD_IPC_ADDR =    0x00002000
BOARD_IPC_SIZE =    0x001FE000
BOARD_LINUX_ADDR =  0x00200000
BOARD_LINUX_SIZE =  0x01000000
BOARD_UBOOT_ADDR =  0x01200000
BOARD_UBOOT_SIZE =  0x00600000
BOARD_UITRON_ADDR = 0x01800000
BOARD_UITRON_SIZE = 0x19800000
BOARD_UITRON_RESV_SIZE = 0x03000000
BOARD_RAMDISK_ADDR = 0x1B000000
BOARD_RAMDISK_SIZE = 0x03000000
BOARD_DSP2_ADDR =   0x1F000000
BOARD_DSP2_SIZE =   0x00800000
BOARD_DSP1_ADDR =   0x1F800000
BOARD_DSP1_SIZE =   0x00800000
BOARD_LOADER_ADDR = 0xF07F0000
BOARD_LOADER_SIZE = 0x0000A000
BOARD_EXTDRAM_ADDR = 0x40000000
BOARD_EXTDRAM_SIZE = 0x20000000
```

Note:

1. uboot memory space will reuse by Linux after uboot finished.
2. loader may need modifying after uitron space is adjusted.
3. Normally, for increate linux memory space, just adjust BOARD_UITRON_RESV_SIZE.

The suggestion of dram partition sequence as below,

Reversed (must reverse 0x2000 bytes)
IPC (must reverse 0x001FE000 bytes)
Linux Kernel
u-boot (reuse by linux-kernel after boot)
uitron
DSP2
DSP1

2.2 Setup embedded flash partition

Partition table is defined at emb_partition_info.c. There can see the sample. The valid partition number is from 0 to 15. User can create his own partitions by using EMBTYPE_USER0~ EMBTYPE_USER5, and modify u-boot to handle ones.
If many partitions with the same type are require, use OrderIdx to indicate them (e.g. DSP1 and DSP2).

EMBTYPE_DSP, EMBTYPE_PSTORE, EMBTYPE_FAT can be removed if not used.
NvtPack's partition order also refer to emb_partition_info.c.

2.2.1 Partition Structure

```
typedef struct _EMB_PARTITION{

    unsigned short  EmbType;        /* EMBTYPE_ */
```

```
    unsigned short  OrderIdx;       /* Order index of the same EmbType based on '0' */
    unsigned int    PartitionOffset; /* Phyical offset of partition */
    unsigned int    PartitionSize;   /* Size of this partition */
    unsigned int    ReversedSize;    /* Reserved size for bad block */
}EMB_PARTITION, *PEMB_PARTITION;
```

- *EmbType*: partition type.
- *OrderIdx*: a certain type has more the same partitions. For example, the 2nd PStore has to set it to 1, the 3rd PStore has to set it to 2, and so on.
- *PartitionOffset*: Physical offset in partition.
- *PartitionSize*: Partition size.
- *ReversedSize*: This field is only for FAT that needs the space to exchange bad block. The size will be reversed from your assigned PartitionSize.
- *Offset and Size* is stored by block unit in modext bin file.

2.2.2 Partition Adjustment

There some rules on partition adjustment.

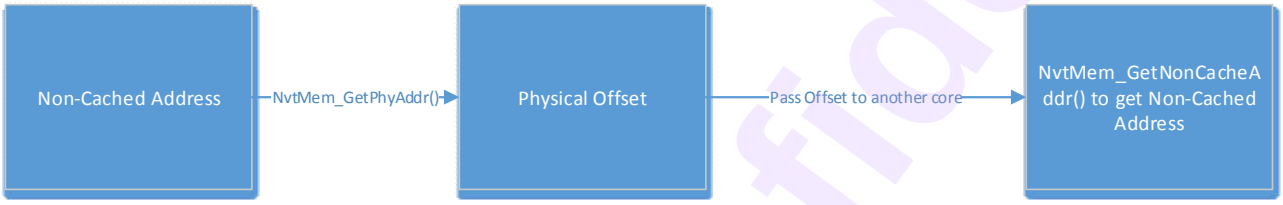
1. Partition[0] is only for load, Partition[1] is only for modelext.
2. Each partition region cannot be overlapped each other.
3. Partition offset most be ascending order.
4. After adjusting partition, better to rebuild whole BSP.
5. Partition size should be block size alignment.
6. For calculating each partition size, the bad block should be considered in it. The more partition size you set, more bad blocks can be accepted.

The suggestion of flash partition sequence as below,

loader (must be 1st partition and reserve 0x4000 bytes for spi-nand)
modelext (must be 2nd partition)
uitron
u-boot
uenv (optional)
linux-kernel
DSP1 (optional)
DSP2 (optional)
PStore (optional)
root-fs
fat (optional)

3 Addressing between Multi-Core

We recommend using non-cached memory (Memory Pool on uITRON) to exchange data. It is difficult to use cached memory because MIPS-1, MIPS-2, DPS have their own independent cache. There write-back or invalid need to take care. Besides, both the starting address and the cached size must be 4-word alignment, if not there will be incorrect data occurred on itself or other variables. The basic strategy is shown below.



That ways to getting physical offset and inverting to non-cached is described below.

- uITRON:
  - Non-Cache to Physical: use NvtMem_GetPhyAddr()
  - Physical to Non-Cache: use NvtMem_GetNonCacheAddr()

- Linux – Kernel Space:
  - Non-Cache to Physical: use dma_map_single()
  - Physical to Non-Cache: use ioremap()

- Linux – User Space:
  - Include nvtipc.h and link nvtipc library
  - Need to use NvtIPC_mmap() map the memory range firstly
  - Non-Cache to Physical: use NvtIPC_GetPhyAddr()
  - Physical to Non-Cache: use NvtIPC_GetNonCacheAddr()
  - Physical to Cache: use NvtIPC_GetCacheAddr()

For example, a non-cached memory needs to be transferred from uITRON to Linux User Space. There must follow the steps.

1. Invoke NvtMem_GetPhyAddr() to get the physical address on uITRON side.
2. Send the physical address to Linux User Application via NvtIPC.

3. Invoke NvtIPC_GetNonCacheAddr () and use the API returned address to access memory.

NOTE:

1. Use the same physical address to invoke uITRON's NvtMem_GetNonCacheAddr() and Linux User Space's one, whose returned Non-Cached Address are not the same. So do not pass the Non-Cached Address directly between different spaces and OS.
2. NvtMem_GetPhyAddr() and NvtMem_GetNonCacheAddr() check the input address whether is in region of shared memory. In case of the returned address adds or subtracts an offset by users, that causes addressing memory is out of shared memory region, it still cause memory overwrite on non-shared-memory. So accessing memory region needs to be taken care of, after getting non-cached address.

4 First Boot

For empty flash-nand or flash-nor, boot-up ROM try to get loader on 1st sector of SD card that must be FAT32. For 64GB or higher SD, try to use ‘GUIFormat’ which can be found on internet.

Because Win8 / Win10 formating sd causes that we cannot boot our device with bootstrap on sd, we need to modify registry can disable this issue. Please save following to an sd.reg and run it.

Windows Registry Editor Version 5.00

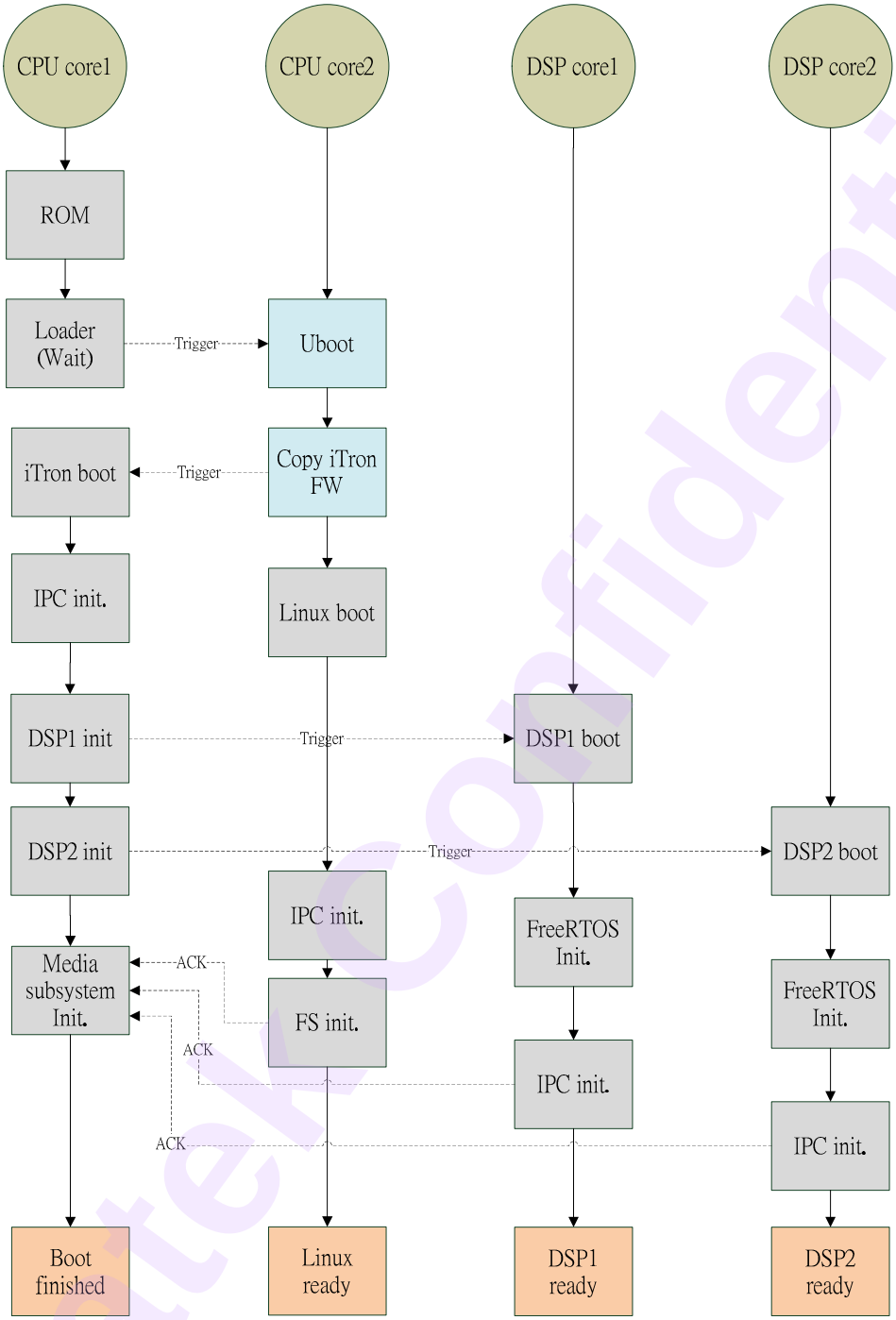
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Windows Search]
"DisableRemovableDriveIndexing"=dword:00000001

5 Boot flow

This chapter will introduce Multi-OS boot/power-off/reboot flow.

5.1 System boot flow

The boot flowchart is listed as below, we have four cores are ARM Cortex A53 core1, ARM Cortex A53 core2, CEVA DSP core1 and CEVA DSP core2 individually to handle overall boot flow. This boot flowchart can boot each core parallelly to minimize boot time.



Step 1: CPU1 ROM code will read Loader image from flash into memory runs it. Loader will initialize basic I/O and DRAM.

Step 2: CPU1 Loader reads uboot image from flash into memory, and trigger CPU2 jump to uboot starting address. And then the CPU1 is waiting for CPU2 signal to continue running. CPU1 can execute iTron OS boot when it get CPU2 signal.

CPU2 uboot will execute basic HW initialization and copy itron/Linux/DSP1/DSP2 (if your SOC can support DSP core) images individually, each image will be triggered according to init sequence.

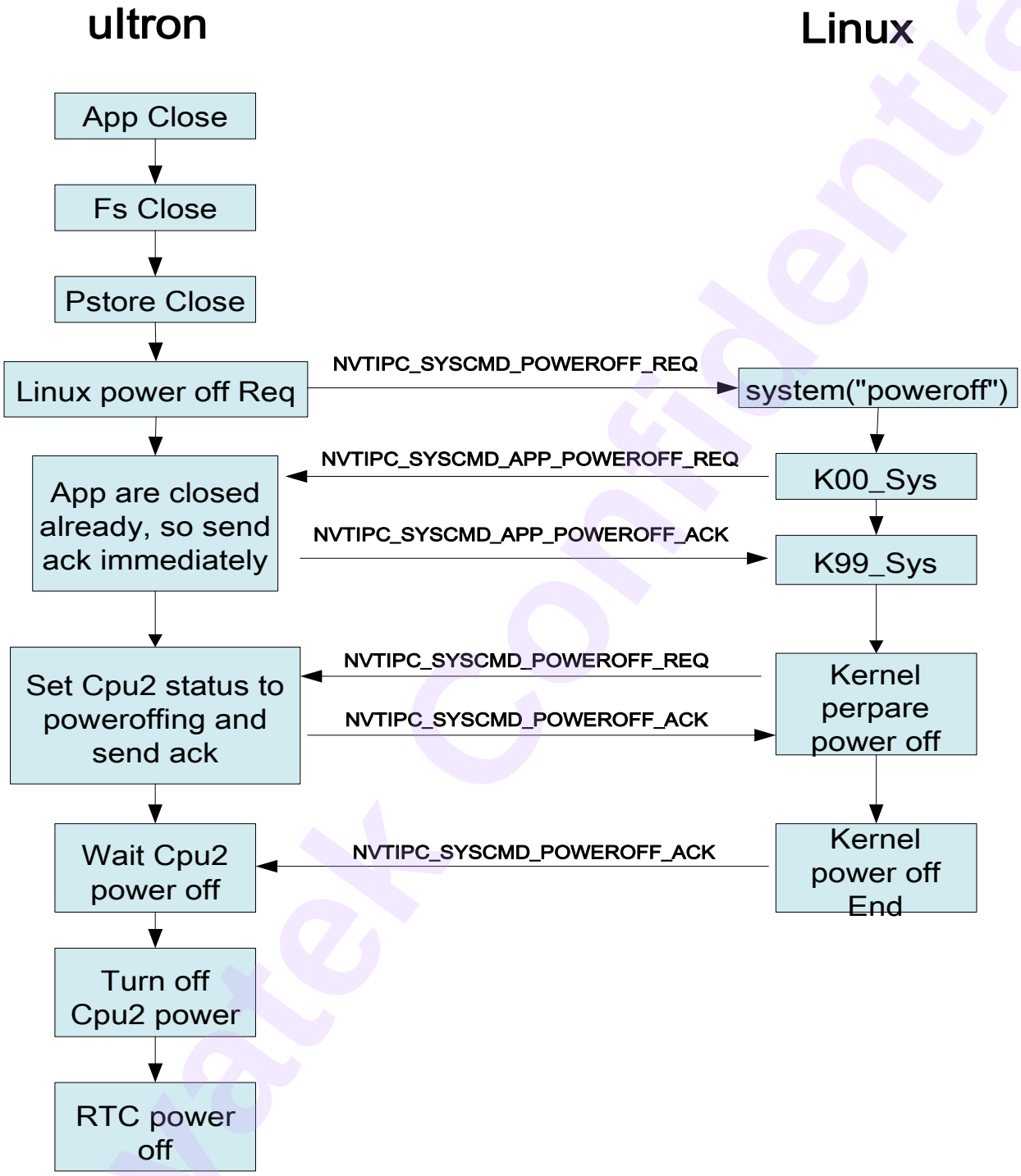
Step 3: DSP1/DSP2 bootloader can launch freeRTOS, the freeRTOS will execute basic init and NVT IPC init to send ack to CPU1.

Step 4: Linux finished initialization and notifies uITRON to start image capture, ISP and encoder. uITRON begins to send video/audio encoded data to Linux

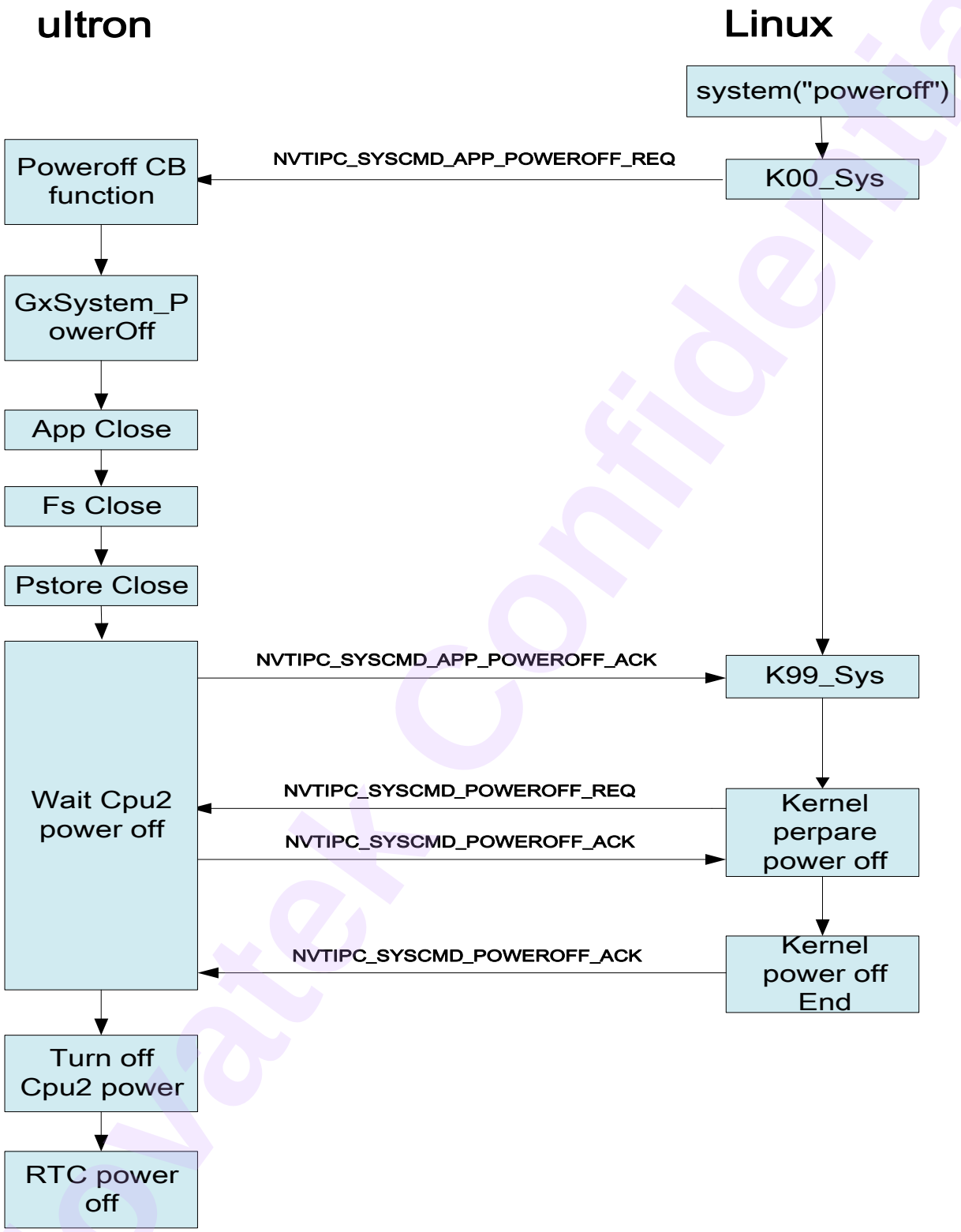
5.2 System poweroff flow

The power off flow can be separated to two cases. One is uITRON launch the power off by press power key. Second is Linux launch the power off by type the command “poweroff” on Linux console. The power off handshake flows have a little different in these two cases. The detail handshake flows are described in below sections.

5.2.1 uITRON launch the power off



5.2.2 Linux launch the power off



5.3 System reboot flow

The reboot flow is very similar to the case that Linux launch the power off. The only different is that reboot will set the power alarm to 5 seconds later and then power off. So the device will auto power on again after 5 seconds.