地 平 线
Horizon Robotics

X3J3

MU-2520-3-X3J3 Platform

System Software Development Manual

Rev. 1.0.0

2021-02

Important Notice and Disclaimer

Information in this document is provided solely to enable system and software implementers to use Horizon products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

All statements, information and recommendations in this document are provided "AS IS". Horizon makes no warranty, representation or guarantee of any kind, express or implied, regarding the merchantability, fitness or suitability of its products for any particular purpose, and non-infringement of any third party intellectual property rights, nor does Horizon assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

"Typical" parameters that may be provided in Horizon datasheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Buyers and others who are developing systems that incorporate Horizon products (collectively, "Users") understand and agree that Users shall remain responsible for using independent analysis, evaluation and judgment in designing their applications and that Users have full and exclusive responsibility to assure the safety of Users' applications and compliance of their applications (and of all Horizon products used in or for Users' applications) with all applicable regulations, laws and other applicable requirements.

User agrees to fully indemnify Horizon and its representatives against any claims, damages, costs, losses and/or liabilities arising out of User's unauthorized application of Horizon products and non-compliance with any terms of this notice.

Horizon Robotics, Inc.

https://www.horizon.ai

Revision History

Time	Version	Revision Detail
2020.06	V0.5	Document created
2020.08	V0.5.2	Updated System Software Overview; Corrected typos
2020.09	V0.5.2a	Add BPU reserve memory configuration
2020.09	V0.5.2b	Add SDIO Manual
2020.09	V0.5.3	Add I2C, SPI module debug guideline; Add GPIO, Pinctrl, IO-Domain, ADC module debug guideline; Add DDR Malfunction Debug guideline Update Camera Sensor Debug guideline.
2020.12	V0.5.4	Add single module programable image compilation explanation; Add DTB Mapping modification manual;
2021-02	V.0.0	Release 1.0

Contents

Cautions

None.

1 X3J3 Platform System Software Overview

1.1 Overview

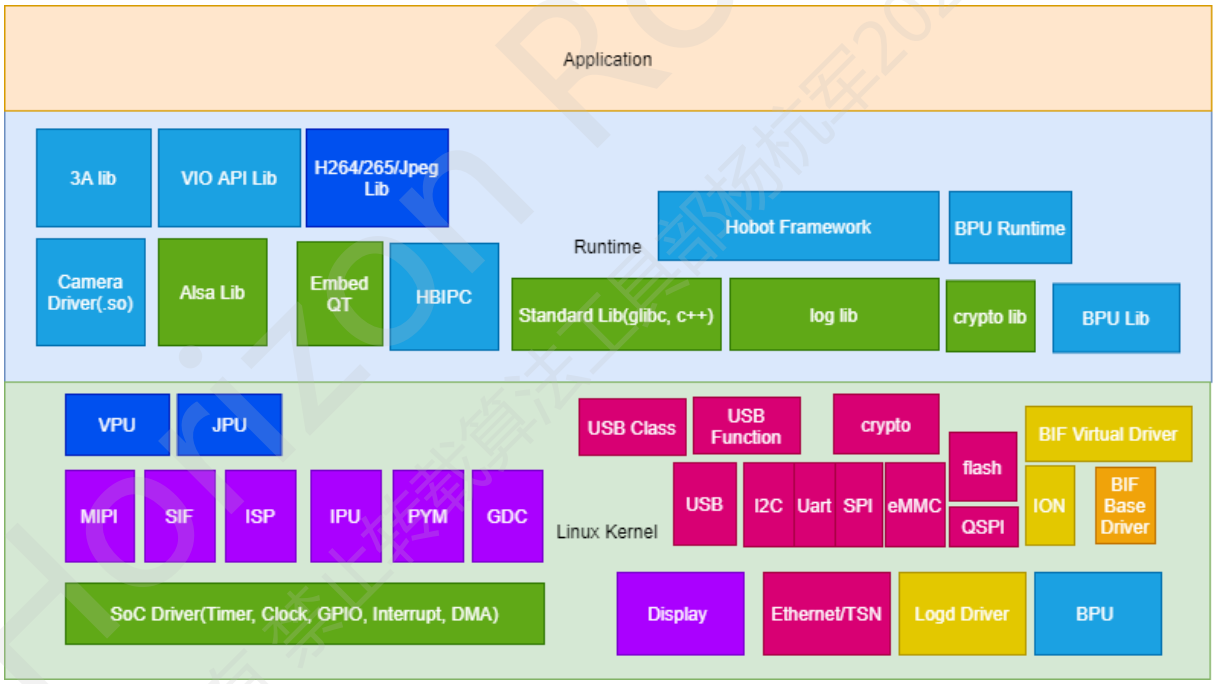
X3J3 system software is based on Linux OS. Kernel adapts Linaro Linux Kernel Arm Branch. The Kernel version used is 4.14.

X3J3 system provides application with basic C/C++ libraries, hardware firmware library, C++ algorithm application framework. The system is also integrated with MiniGUI and Qt graphic library.

X3J3 system will be customized for each individual product to provide sufficient software modules.

X3J3 System inherits the basic solution from X2J2 platform with extra software support for new hardware modules.

1.2 System Topology



2 SDK Directory Structure

This chapter will introduce the SDK directory structure to the developer.

Note: this is only a general introduction to the most commonly used files, please refer to the actual directory for details.

```
build  hbre  kernel  out  prebuilts  README  uboot  unittest
```

2.1 Platform SDK

2.1.1 build

Images build related files.

- build.sh: the script used to compile and build the images;
- envsetup.sh: environment setup script for compile and build.

2.1.1.1 ota_tools

OTA related tools.

2.1.1.2 tools

cramfs and squashfs related tools.

2.1.2 hbre

Horzon Customized user library source code. E.g. Camera, Diagnose, etc.

2.1.3 kernel

Kernel related files.

2.1.3.1 arch

This directory will include a Kconfig file which is used to setup the compile of the Kernel source code.

2.1.3.2 block

This directory consists of **block device** source codes. The block device framework and block device I/O scheduling is implemented here. Block devices are the type of device who sends and received data in blocks. All data is transmitted in blocks instead of streams.

2.1.3.3 crypto

This directory consists of the source codes of multiple encryption algorithm. For example, "sha1_generic.c" implements sha1 algorithm.

2.1.3.4 drivers

This directory consists of device drivers' source codes. Device drivers are the software used to control hardware. For example, if the system is to recognize and utilize keyboard, a driver for the type of keyboard is required. This directory includes multiple subdirectories, among which, each of them are named after the type or model of the hardware.

2.1.3.5 firmware

This directory consists of the source code assists the system in understanding the signals sent from devices. For example, Linux system utilized vicam firmware to understand the communication from sensors. Linux is otherwise not capable of processing signals sent from sensors.

2.1.3.6 fs

This directory consists of source codes implementing file systems. Every file system will have its own subdirectory in this directory. For example, "ext4/" directory consists of all the source codes implementing ext4 file system. General framework codes are in the fs directory itself, for example, mount.h consists codes related to mounting file systems.

2.1.3.7 Makefile

This is a file used to compile the Linux Kernel. Makefile consists of necessary compile parameters which will be passed to compiler.

2.1.4 out

The output directory of the compiled images

2.1.5 prebuilts

The precompiled files for rootfs.

2.1.6 uboot

Source code for booting.

2.1.6.1 api

APIs for UBoot.

2.1.6.2 arch

Source codes handling CPU architecture related functions.

2.1.6.3 board

Source codes implementing customized functions for individual boards.

2.1.6.4 common

Common codes, mostly related to UBoot commands.

2.1.6.5 disk

Hard drive partitioning related codes.

2.1.6.6 doc

Codes related to README.txt.

2.1.6.7 drivers

Device drivers source codes.

2.1.6.8 examples

Example applications.

2.1.6.9 fs

File systems compatible with most of the development boards.

2.1.6.10 include

Header files.

2.1.6.11 lib

Libraries.

2.1.6.12 net

Ethernet related source codes, including a simple protocol stack.

2.1.6.13 post

Power On Self Test.

2.1.6.14 tools

Miscellaneous applications used to compile and check UBoot related files.

2.1.7 unittest

Simple unittest applications, such as VIO access verification.

2.2 APP SDK

2.2.1 appuser

2.2.1.1 include

Header files for user compiled applications.

2.2.1.2 lib

Dynamic libraries for user compiled application.

2.2.1.3 etc

VIO; Camera; IAR related configuration files.

2.2.1.4 share

Configuration files needed for ALSA Module compilation.

2.2.2 Toolchain

Cross Compiler.

3 Development Environment Setup

3.1 Configure Cross Compiler

For executable binaries to be executed on development boards (X2/J2/X3/J3), the binaries have to be compiled via cross compilers. This section will introduce the steps for configuring cross compilers.

3.1.1 Cross Compiler Version

There are numerous cross compilers available, in our product, we require cross compiler: gcc-linaro-6.5.0-2018.12-x86_64_aarch64-linux-gnu. One such compiler is provided in the build system as shown below:

```
========================================
ARCH="arm64"
CROSS_COMPILE="/home/weitao.li/code/prebuilts/host/gcc-linaro-6.5.0-2018.12-x86_64_aarch64-linux-gnu/bin/aarch64-linux
-gnu-"
TARGET_VENDOR="horizon"
TARGET_PROJECT="x2"
TARGET_MODE="debug"
TARGET_BIT="64"
ENABLE_BUILD_DEPENDENCY="false"
N="4"
```

3.2 Ubuntu 18.04

For Ubuntu18.04, execute the following command to setup development environment:

```
sudo apt install build-essential
sudo apt-get install make
sudo apt install cmake
sudo apt install bison
sudo apt install flex
sudo apt-get install python-numpy
sudo apt install android-tools-fsutils
sudo apt install mtd-utils
sudo apt install zlib1g-dev
```

If any errors are reported, install those too.

3.3 CentOS

```
yum install glib cmake gcc bison flex minicom python-pip
pip install numpy
yum install make_ext4fs-1.0.0-1.el7.x86_64.rpm
yum install zlib-devel
yum install mtd-utils-1.5.0-2.el6.nux.x86_64.rpm
```

4 Build and Image Generation

4.1 Setup Build Environment

```
mkdir x3j3
```

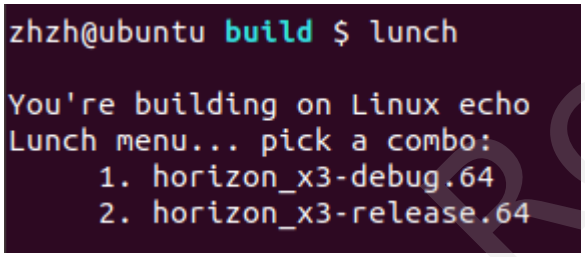
Unzip the SDK zip file into the directory created, and execute the following command:

```
cd x3j3
source ./build/envsetup.sh
```

After the environment is set up, use lunch command to check options available:

```
lunch
```

The following options should appear, choose the option as needed:



4.2 Image Partition Configuration

The partitioning of the image is configured by the gpt.conf files. The following chapter will introduce how to modify the gpt.conf files.

4.2.1 Configuration file

The following example is based on easy-gpt.conf. For actual design of image partition, please refer to the files under the path /build/device/horizon/x3 from the Platform SDK.

"gpt.conf"(partition configuration) use ":" as separator of different segments. There are two forms of configuration:

```
Part_img_en:part_name/part_content:part_fs:start:end:disk_en
Part_img_en:part_name/part_content:part_fs:part_size:disk_en
```

The meaning of the segments is listed below:

- part_img_en: indicates whether creates a standalone image for the current partition, when "0" is put in, the partition content under the path will be combined into the previous partition image under the name of part_name with part_img_en enabled;
- part_name/par_content: separated with "/", the first part indicates the partition

name, all contents of the current partition will be placed under the folder named "part_name" under deploy directory;

- part_fs: indicates the filesystem used to manage the content of the current partition;
- start: the start location of the current partition, can be measured in bytes(no unit), kibibytes(k), mebibytes(m), or sectors – 512bytes(s). for example, 30, 30k, 30m, 30s etc.
- stop: the end location of the current partition, calculated as "start + part_size - 1"
- part_size: the size of the partition
- disk_en: whether the current partition should be included in the final disk.img output, if "1" then 0 (1 when using NAND) paddings will be created in the disk.img to be filled with actual content.

Here is an actual example:

```
1:veeprom/veeprom.bin:none:34s:37s:1
1:sbl/spl_storage.bin:none:38s:1061s:1
0:sbl/spl_warm_storage.bin:none:550s:1061s:1
1:ddr/ddr_storage.bin:none:1062s:2085s:1
0:ddr/efuse.bin:none:2080s:2085s:1
1:bl31/bl31.img:none:2086s:3109s:1
1:uboot/uboot.img:none:3110s:7205s:1
0:ubootbak/uboot.img:none:5158s:7205s:1
1:vbmeta/vbmeta.img:none:7206s:7461s:1
1:boot/boot.img:none:7462s:27941s:1
1:recovery/recovery.img:none:27942s:58661s:1
1:system:ext4:58662s:365861s:1
1:bpu/bpu_image.bin:none:365862s:570661s:1
1:app:ext4:570662s:1094949s:0
1:userdata:none:6600m:0
```

4.2.2 Adding Content to Partition

If the content of the partition is in the path listed in the gpt.conf relative to deploy folder, the content is automatically copied to the disk.img.

For example:

- In the above gpt.conf, the part_img_en of partition sbl is 1, thus a file named sbl.img will be created. If "spl_storage.bin" and "spl_warm_storage.bin" is available in /**/deploy/sbl/ directory, both files will be copied into sbl.img at the offset specified in gpt.conf. And eventually, sbl.img will be combined in disk.img since disk_en is 1.
- If there is no directory in the deploy folder under the name of "system", when

disk.img is created, there will be only paddings in the offset of the partition "system".

- If partition filesystem is listed as "none", when creating disk.img, paddings will be added first then build system will look for "part_name.img" and use "dd" command to add it to disk.img. If filesystem is specified, a file system based on /**/deploy/<part_name>/ directory will be creaded and copied to disk.img.

4.3 Rootfs Content Modification

4.3.1 Add/Remove Rootfs source files

The precompiled source files of rootfs is stored under /prebuilts/root/ directory, it will also be modified according to /prebuilts/root_hijack/. Eventually, all source files will be copied to /out/horizon_x3-[debug/releas].[64/32]/target/rootfs/ for further filtering.

There are four ways to add new files to the rootfs. (each method will overwrite changes made by above methods):

- Add new files to <path to your folder>/prebuilts/root and execute build.sh
- Add new files to <path to your folder>/out/horizon_x3-[debug/release].[64/32]/patchrootfs/ and execute build.sh. (the directory needs to be created)
- Add new fiels to <path to your folder>/prebuilts/root_hijack and update do_hijack.sh script under the same directory, then execute build.sh
- **AFTER** a full compile, add new files to <path to your folder>/ out/horizon_x3-[debug/release].[64/32]/target/rootfs/ and use "-c none" in the build command:

```
./build.sh -c none -b xxx -i xxx -e xxx
```

Note: the file under /target/rootfs is deleted when executing full compile, thus the red part of the above command must be used as is while blue part can be modified to suit the purpose of the current build.

4.3.2 Use Manifest to define rootfs

After modifying the source file of root filesystem, use manifest file to modify the actual rootfs before executing ./build.sh. Use "-" to delete source files and "+" to add files. The targeted files should already be in /prebuilts/root，/prebuilts/root_hijack，/out/<version>/patchrootfs/ directories or be compiled and stroed to /out/../tmprootfs

For example:

```
-<absolute path to file from root>
+<absolute path to file from root>
```

For example, to remove a file on board with absolute path "/app/bin/bin_to_rm", add the following line to manifest file:

```
-/app/bin/bin_to_rm
```

Manifest can also be defined with "+" to specify each and every file required in Rootfs.

After editing the manifest file, execute ./build.sh again to have the rootfs updated.

4.4 Boardid and DTS Mapping

4.4.1 Overview

When new DTS and Boardid mapping is required, there are changes necessary in both Build and Kernel.

4.4.2 Modification Procedure

During boot, system software can auto detect the hardware and select the corresponding DTB file. The auto detection process is customizable. The DTB selection can also be hardcoded by assigning boardid during image compilation. Boardid definition has a higher priority than auto detection.

4.4.2.1 DTB Selection Modification

a) Build System Assigning Boardid

Boardid has the highest priority, boardid can be used to hardcode DTB used. The required fields in boardid includes SOM_TYPE and BASE_BOARD_TYPE, for details, please refer to build script directly.

b) UBoot Boot Process Modification

During boot, the auto detection of hardware is customizable. The related files and functions are listed below:

```
uboot/board/hobot/common/board.c: hb_board_type_get()
uboot/board/hobot/common/board.c: hb_base_board_type_get()
uboot/board/hobot/common/board.c: hb_som_type_get()
uboot/board/hobot/x3/x3.c: hb_gpio_to_borad_id()
```

4.4.2.2 Procedure to add DTB Mapping

a) Add DTS File and Compilation Options

Files involved:

Add the DTS file to the path below, take adding hobot-x3-sample.dts as an example:

```
kernel/arch/arm64/boot/dts/hobot/hobot-x3-sample.dts
```

After the DTS file is created according to the hardware targeted, add the compilation option in the following file:

```
kernel/arch/arm64/boot/dts/hobot/Makefile
```

Take hobot-x3-sample.dts as an example:

```
dtb-$(CONFIG_HOBOT_XJ3)  += hobot-x3-sample.dtb
```

The preceding " CONFIG_HOBOT_XJ3" can be modified to the Kernel config preferred.

b) Adding DTB Mapping

Still using hobot-x3-sample.dts as example:

Files involved:

```
Kernel/tools/dtbmapping/dtb_mapping_xj3.json
```

Add the following entry in the file including <index + dtb file name + gpioid>:

```
    "105": {
        "dtb_name"  :   "hobot-x3-sample.dtb",
        "gpio_id"   :   "0"
},
```

In the above example,

"105" is the index, must correspond to Build system modification and auto detection;

"dtb_name" is the actual DTB file name;

"gpio_id" is not implemented now, can be customized based on auto detection modification.

4.5 Build System Synopsis

The default parameter used in Build is listed below, pass in the desired parameters as needed

```
-e all -b xj3_emmc -t LPDDR4 -f 2666 -v db
```

Except for nor/nand boot Images, all other boot mode Images disabled Flash detection by default. If Flash detection is needed, please use FLASH_ENABLE=nor/nand before build command:

```
FLASH_ENABLE=nor ./build.sh
```

```
FLASH_ENABLE=nand ./build.sh
```

Other detailed options description can be accessed by the following command:

```
./build.sh -h
```

4.5.1 X3-dvb/J3-dvb Universal Image

```
./build.sh
```

Image created without specifying DDR parameters will auto-detect DDR parameters from eFuse.

4.5.2 X3-dvb/J3-dvb eMMC Boot Image, Hynix DDR

```
./build.sh -m hynix
```

4.5.3 X3-dvb/J3-dvb eMMC Boot Image, Micron DDR

```
./build.sh -m micron
```

4.6 Modular Build

First, enter the build directory, and set up the compile environment.

4.6.1 Uboot

Enter uboot directory

```
cd uboot
export BOARD_TYPE=xj3dvb
./build.sh
```

To build standalone programable image, use the command below:

```
./build.sh -p
```

The standalone programable "uboot.img" is saved in:

```
x3j3/out/horizon_x[i]-[debug/release].[64/32]/target/deploy/uboot/uboot.img
```

OTA package which can be used in Kernel OTA upgrade is saved in:

```
x3j3/out/horizon_x[i]-[debug/release].[64/32]/target/deploy/ota/uboot.zip
```

4.6.2 Kernel

Enter kernel directory

```
cd kernel
export KERNEL_WITH_CPIO=false
./build.sh
```

To build standalone programable image, use the command below:

```
./build.sh -p
```

The standalone programable "kernel.img" is saved in:

```
x3j3/out/horizon_x[i]-[debug/release].[64/32]/target/deploy/kernel.img
```

"kernel.img" includes both vbmeta and boot.img.

OTA package which can be used in Kernel OTA upgrade is saved in:

```
x3j3/out/horizon_x[i]-[debug/release].[64/32]/target/deploy/ota/boot.zip
```

4.7 Build Output

The Image will be generated to the directory:

```
x3j3/out/horizon_x[i]-[debug/release].[6432]/target/deploy/xxxx.img
```

In the above path, "i" is the current product number, "debug/release" indicates system status version, "64/32" indicates the if the system is running on a 64 bit or 32 bit machine. For example, when we are building a "x3 debug 64 bit" image, the output path will be:

```
x3j3/out/horizon_x3-debug.64/target/deploy/xxxx.img
```

5 Burn Image

5.1 Horizon Upgrade Tool

5.1.1 Windows

Unzip hbupdate_win64_v*.zip, execute hbupdate.exe in the unzipped directory. Logs will be saved to the log directory in the unzipped directory.

Upgrade Process:

1. Choose product type;
2. Choose upgrade mode
3. Configure options for PC and Product connection: serial port, IP address etc.
4. Choose Upgrade file
5. Start Upgrade

Please refer to "Horizon Upgrade Tool User Manual" for details

5.1.2 MAC

Unzip hbupdate_MAC_v*.zip, drag the app file to "Application" (Might require the user to change the settings in "Settings-> Security and Privacy" to allow execution). If execute failed or "Permission Denied" prompts, please use "sudo" in console to execute the tool:

```
sudo /Applications/hbupdate.app/Contents/MacOS/hbupdate
```

Logs are stored in directory:
/Applications/hbupdate.app/Contents/MacOS/hbupdate/log

5.1.3 FAQ

1) Please go over the hbupgrade tool user manual before upgrading any boards
2) Before upgrade start, please make sure tftp service is properly setup and serial port to the board is NOT in use
3) File name of the upgrade image should contains only English characters and underscore(without any spaces or tabs)
4) To achieve optimal upgrade speed and stability, please connect your board with PC via cable instead of wireless connection
5) If upgrade/boot failed, please retry using "uart" mode
6) Please make sure PC network card connecting the board is setup properly under

the same subnet with the board

7) If network connection failed, please turn off firewall and try again

5.2 Manual Image Burn

5.2.1 Environment setup (tftp server and nfs setup)

5.2.1.1 tftp Environment

Windows: use executable tftpd64

Linux tftp server setup:

a) Linux tftp Server setup

```
sudo apt-get install tftpd-hpa
sudo mkdir  /tftpboot
sudo vi  /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="[::]:69"
TFTP_OPTIONS="-l -c -s"
```

Configure IP address: 192.168.1.1:

```
Ifconfig 192.168.1.1
```

Restart tftp service:

```
sudo service tftpd-hpa restart
```

Put image into /tftpboot directory

b) Download through tftp in UBoot:

```
setenv serverip 192.168.1.1
setenv ipaddr 192.168.1.10
# File address in DDR
setenv target_addr 0x6000000
tftp ${target_addr} <file path from tftp server folder>
mmc write ${target_addr} <File length in sectors(512B/sectors)>
reset
```

5.2.1.2 Mount nfs directory from PC

Recommended for running applications on board without transferring the application to the board, preferable for large applications. (Any additional libs should be copied to /lib directory onboard)

a) PC Setup - Linux

Install nfs

```
sudo apt-get install nfs-kernel-server
```

Setup

```
cd ~
mkdir nfs
sudo vim /etc/exports
/home/gs/nfs 192.168.*.*(sync,rw,no_root_squash)
sudo exportfs -a
```

Reboot rpcbind service

```
sudo service rpcbind restart
```

Reboot nfs service

```
sudo service nfs-kernel-server restart
```

b) PC Setup - Windows

Download and install nfs1169--.zip.

c) Board Setup

Configure IP address

```
ifconfig eth0 192.168.1.10 netmask 255.255.255.0
```

Mount

```
mkdir -p /mnt/nfs
mount -t nfs -o nolock 192.168.1.1:/home/gs/nfs /mnt/nfs
```

5.2.2 UART Programming

When the board is powered up, the default boot mode is eMMC boot (please check DIP switch or Bootsel Register to make sure). If this is an empty SOM, boot will automatically switch wo UART mode. Use SecureCRT/MobaXterm on Windows, minicom on Linux is recommended.

5.2.2.1 SecureCRT

Choose the correct transfer protocol in menu or drag the file into SecureCRT window and select the correct protocol.

5.2.2.2 Minicom

Use Ctrl-A+S or customized hotkey to select the correct transfer protocol.

Go to the input director, use "space" to select the files, press "OK" button to confirm and execute the transfer.

5.2.2.3 Programming Procedure

Note: Start with XModem protocol in bootrom phase to transfer SPL. Use YModem totransfer the remaining images.

1) After board powered up and prompts for UART boot, use XModem to download SPL file.
2) Following the prompt to download DDR image and boot.pkg using YModem and enter UBoot console. The remaining program procedure, please refer to *UBoot Image Programming*

Note: After UART boot to UBoot Console, if there is no image programmed to storage, all progress will be lost after powered down/reboot. The UART Boot process will have to be performed again to enter UBoot.

5.2.3 UBoot Image Programming

5.2.3.1 Program Full Image

Execute the following command in UBoot console:

Set Board ip address to 192.168.1.1 or IP preferred

```
setenv ipaddr 192.168.1.1
```

Set the server id to 192.168.1.10 or IP preferred

```
setenv serverip 192.168.1.10
```

Use relative file path in tftp server folder

```
tftp 0x3e30000 disk.img
```

Write the image to emmc

```
otawrite all 0x6000000 ${filesize} emmc
```

- TFTP download target address should avoid all reserved memory, but smaller than (memory_size – disk.img_size). Here, 0x3e30000 is recommended, modify as needed
- The length passed to "otawrite" is the bytes transferred by tftp, ${filesize} means using the environment variable "filesize" set by tftp.

5.2.3.2 Partition-wise Programming

Take X3 DVB as an example, to change Kernel, both vbmeta.img and boot.img need to be programmed.

【In UBoot】

1) vbmeta.img

```
tftp 0x3e30000 vbmeta.img
otawrite vbmeta 0x6000000 ${filesize} emmc
```

"mmc write" command can also be used:

```
mmc write 0x6000000 0x384c00 100
```

(0x384c00 is the starting address of vbmeta partition, which can be calculated by (start_lba*512), start_lba can be accessed by command "part list"; 0x20000 is the size of vbmeta.img denoted in hexadecimal form, which will be printed after tftp download)

2) boot.img

```
tftp 0x3e30000 boot.img
otawrite vbmeta 0x6000000 ${filesize} emmc
```

"mmc write" command can also be used:

```
mmc write 0x36000000 0x3a4c00 5000
```

(0x3a4c00 is the starting address of boot partition, which can be calculated by (start_lba*512), start_lba can be accessed by command "part list"; 0xa00000 is the size of boot.img denoted in hexadecimal form, which will be printed after tftp download)

【In Kernel】

1) Download vbmeta.img and boot.img through tftp:

```
ifconfig eth0 192.168.1.10
tftp -g -r vbmeta.img 192.168.1.1
tftp -g -r boot.img 192.168.1.1
```

2) Write the Images to the corresponding partitions. Partition Number can be accessed by command "fdisk -l". (Partition 6: vbmeta; Partition 7: boot)

6 USB Manual

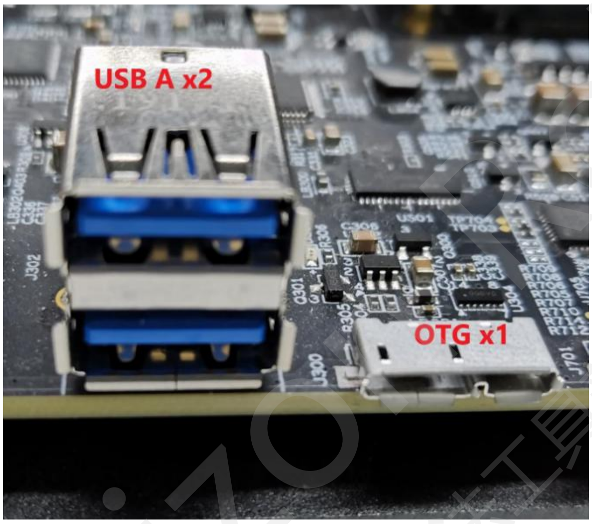
6.1 USB3.0 Port configuration and Usage

6.1.1 Configuration

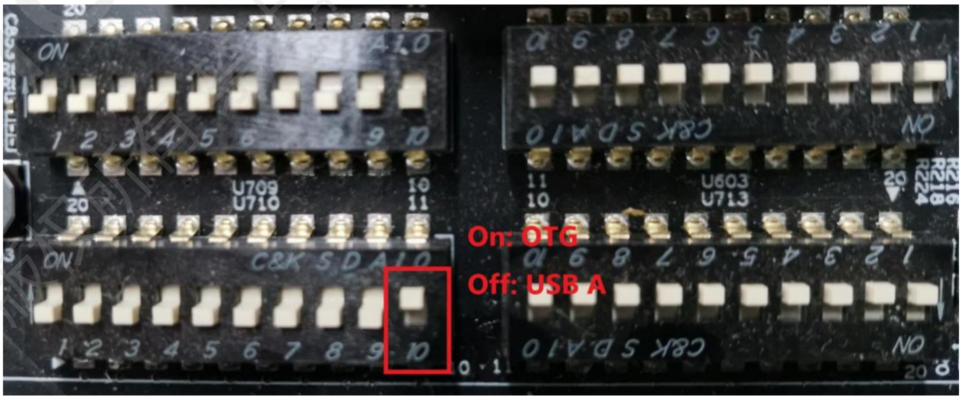
6.1.1.1 X3DVB

There are 3 USB3.0 ports, listed below:

- 2 USB Standard A Port
- 1 USB Micro-B Port(OTG Compatible)



OTG Port and USB Standard A ports are mutually exclusive. U710DIP Switch 10 is used to configure which port is enabled. When U710DIP switch 10 is on, OTG port is enabled; otherwise standard A port is enabled.



6.1.1.2 J3DVB

J3DVB is equipped with only 1 USB port, no configuration needed

6.1.2 Usage

OTG port will configure master/slave mode automatically on connect, defaulted to slave mode. On DVBs, OTG port is configured to ADB mode by default. OTG port can also be used to connect to peripherals like mouse and keyboard.

The OTG cable/converter need to have usb_id line pulled down. If usb_id line is not pulled down, software configuration is required for slaves to be recognizable.

Use sysfs gpio interface to configure usb_id line.

```
#!/bin/sh

# Stop adb deamon
service adbd stop

# Switch USB controller to master mode
echo host > /sys/kernel/debug/b2000000.usb/mode

# Unbind extcon-usb driver, release gpio pin
echo "soc:usb-id" > /sys/bus/platform/drivers/extcon-usb-gpio/unbind

# echo gpio 65 to sysfs
echo 65 > /sys/class/gpio/export

# Configure gpio65 to output
```

```
echo "out" > /sys/class/gpio/gpio65/direction

# Configure pulldown to gpio65
echo 0 > /sys/class/gpio/gpio65/value
```

6.2 ADB and Fastboot Usage

6.2.1 ADB

6.2.1.1 PC

PC Connected to the board must be equipped with ADB tools. Please refer to LineageOS and other websites to install ADB tools. Commands like adb shell, adb push, adb pull etc are supported.

6.2.1.2 Device

After ADB tools are installed on PC, usd micro-USB cable to connect the board to PC via OTG port. All boards are configured to have ADB Debug enabled, PC should recognize the device automatically.

ADB related configuration can be accessed through adbd:

```
service adbd start
service adbd stop
service adbd restart
service adbd status
```

6.2.2 Fastboot

6.2.2.1 How to enter Fastboot

a) In Kernel

```
reboot fastboot
```

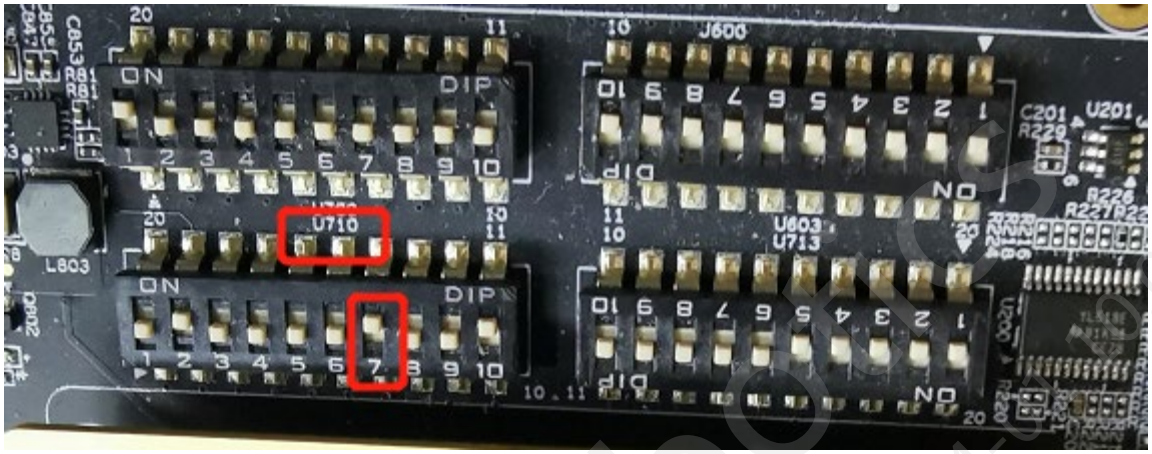
b) In UBoot

```
fastboot 0
```

c) Using DIP Switch

When Bootsel[15] is turned on, the board will boot into Fastboot mode:

X3DVB DIP Switch: U710 DIP7:



J3DVB DIP Switch: U38 DIP7:

Use Micro-USB cable to connect the DVB to PC through micro-USB port, PC should recognize the device automatically as Android Phone.

6.2.2.2 Usage

The board needs to boot into Fastboot mode using either one of the methods above and connect to PC via USB cables.

PC must install Fastboot tools.

a) Use Fastboot to program all partitions:

When the board is being programmed for the first time or there is a change in gpt, please update gpt.img, Windows Bat script for reference:

```
:: fastboot flash bat
@echo off
set ANDROID_PRODUCT_OUT=%cd%
type nul > android-info.txt


fastboot flash mbr mbr.bin
```

```
fastboot flash gpt gpt_main.img
fastboot flash sbl sbl.img
fastboot flash bl31 bl31.img
fastboot flash uboot uboot.img
fastboot flash boot boot.img
fastboot flash system system.img

fastboot reboot
```

Put the script and the images for each partition in the same folder and execute the script to perform Image programming.

b) Program each partition separately:

Enter the folder containing the Images for partition to be programmed, and use the following commands to upgrade corresponding partition:

Boot Partition:

```
fastboot flash boot boot.img
```

System Partition:

```
fastboot flash system system.img
```

etc.

7 ALSA Manual

7.1 Overview

This chapter will focus on the principles of Audio development, including a simple guide to add new audio cards and debug new audio cards.

Relevant acronyms:

CPU DAI: CPU side digital audio interface, usually in I2S protocol, controls bus transfer;

CODEC DAI: codec, control codec workflow, provides interface to core level;

DAI LINK: link between CPU DAI and CODEC DAI, here indicates machine driver;

PLATFORM: indicates CPU side platform driver, usually DMA driver for data transfer;

DAPM: Dynamic Audio Power Management

7.2 Audio Development

A complete Audio card consists of cpu_dai,codec_dai, platform and dai_link.

Each corresponding to: driver of cpu_dai – usually i2s, driver of codec_dai – ac108 codec driver for example, driver of platform – usually DMA drivers, used for audio data processing, driver of dai_link – used to connect cpu_dai and codec_dai, for example: sound/soc/hobot/hobot-snd-96.c.

This chapter use ac108 as an example to demonstrate how to add and debug sound cards.

7.2.1 I2S Parameters

Regarding X3J3 I2S capabilities, there are certain restrictions:

- Lane Support: Audio in supports 1/2/4/8/16 lanes; Audio out supports 1/2 lanes
- Sample Rate: 8k/16k/32k/48k/44.1k/64k
- Sample bit-width: 8bit/16bit
- Transmission Protocol: I2S/dsp
- Master mode default clock: mclk – 12.288MHz; bclk – 2.048MHz. When there is no change to mclk, bclk supports 6.144MHz, 4.096MHz, 3.072MHz, 2.048MHz, 1.536MHz. Please configure to fit the need of application. Frequency Modulation strategy is implemented in function "hobot_i2s_sample_rate_set" in file sound/soc/hobot/hobot-cpudai.c. For support of 44.1K sample rate, without changing PLL, the closest frequency one can get is 44.11764KHz;
- Slave mode: bclk must be generated and transferred to I2S before any R/W to I2S register occurs, otherwise I2S will malfunction.

7.2.2 Guide to Add New Codec

7.2.2.1 Adding codec driver。

For example, to adding ac108_driver.c to sound/soc/codecs directory.

7.2.2.2 Adding Compiling Options

Modify sound/soc/codec/Kconfig and Makefile to add the targeted codec for compilation.

For Kconfig, add:

```
        depends on I2C
config SND_SOC_AC108_DRIVER
        tristate "AC108 Audio Codec"
        depends on I2C
config SND_SOC_AC101_DRIVER
```

For Makefile, add:

```
obj-$(CONFIG_SND_SOC_AC108_DRIVER)   += ac108_driver.o
```

Enable codec in Menuconfig:

```
Device Drivers  --->
<*> Sound card support  --->
<*>   Advanced Linux Sound Architecture  --->
<*>   ALSA for SoC audio support  --->
<M>   hobot audio use dts
        CODEC drivers  -->
            [M]  AC108 Audio Codec
```

7.2.2.3 Modify dts file

Add the codec information to the corresponding I2C, for example:

```
ac108_0: ac108_0@0x3b {
        compatible = "ac108_ic_0";
        reg = <0x3b>;
        #sound-dai-cells = <1>;
};
```

Configure dts file to add the corresponding sound card, for example:

```
&snd0 {
        model = "hobotsnd0";  // Name of Sound Card
        dai-link@0 { //Corresponding ADC
                dai-format = "dsp_a"; // Corresponding SND_SOC_DAIFMT_DSP_A
```

```
                bitclock-master; // Corresponding SND_SOC_DAIFMT_CBM_CFM
                frame-master;
                frame-inversion; // Corresponding SND_SOC_DAIFMT_NB_IF
                link-name = "hobotdailink0"; //Name of dailink
                cpu {
                        sound-dai = <&i2s0>;
                        // CPU Side digital audio interface, controlling bus transfer
                };
                codec {
                        sound-dai = <&ac108_0 0>; //codec Chip interface
                }
                platform {
                        sound-dai = <&i2sidma0>; //dma driver, used for data transfer
                };
        };
        dai-link@1 { // Corresponding DAC
                dai-format = "i2s"; // Corresponding SND_SOC_DAIFMT_I2S(Transfer Mode)
                bitclock-master; // Corresponding SND_SOC_DAIFMT_CBM_CFM(Master/Slave Mode)
                frame-master;
                frame-inversion; // Corresponding SND_SOC_DAIFMT_NB_IF(clock polarity)
                link-name = "hobotdailink1"; //Name of dailink
                cpu {
                        sound-dai = <&i2s1>;
                };
                codec {
                        sound-dai = <&ac101>;
                };
                platform {
                        sound-dai = <&i2sidma1>;
                };
        };
};
```

7.2.3 Application API

For detailed description of Application API description and usage, please refer to API Manual.

7.3 Debug Guide

7.3.1 Audio Card Debug

1. Make sure audio card driver module is correctly loaded, if not, load audio card

driver module first;

- 96board audio board half duplicate (ac108 Record/ac101 Play)

```
modprobe ac101
modprobe ac108_driver
modprobe hobot-dma
modprobe hobot-cpudai
modprobe hobot-snd-96 snd_card=0
```

- 96board audio board full duplicate (ac101 Record/Play)

```
modprobe ac101
modprobe hobot-dma
modprobe hobot-cpudai i2s_ms=4
modprobe hobot-snd-96 snd_card=2
```

- 1977 Audio board

```
modprobe snd-soc-adau1977
modprobe snd-soc-adau1977-i2c
modprobe hobot-dma
modprobe hobot-cpudai
modprobe hobot-snd-96 snd_card=1
```

- ac108 modules

```
Modprobe ac108_driver
Modprobe ac101
Modprobe hobot-dma
Modprobe hobot-cpudai
Modprobe hobot-snd-96 snd_card=1
```

2. Make sure audio card driver is successfully registered

```
root@x3dvbx3-hynix1G-2666:~# cat /proc/asound/cards
0 [hobotsnd0      ]: hobotsnd0 - hobotsnd0
                     hobotsnd0
root@x3dvbx3-hynix1G-2666:/userdata# ls -l /dev/snd
drwxr-xr-x    2 root     root           60 Jan  1 08:00 by-path
crw-rw----    1 root     audio     116,   2 Jan  1 08:00 controlC0
crw-rw----    1 root     audio     116,   3 Jan  1 08:00 pcmC0D0c
crw-rw----    1 root     audio     116,   4 Jan  1 08:00 pcmC0D1p
crw-rw----    1 root     audio     116,  33 Jan  1 08:00 timer120 (range 0->255)
```

3. Use the following command to test audio card functionality:

Record: tinycap parameters:

```
root@x3dvbx3-hynix1G-2666:/userdata# tinycap
```

```
Usage: tinycap {file.wav | --} [-D card] [-d device] [-c channels] [-r rate] [-b bits] [-
period_size] [-n n_periods] [-t time_in_seconds]
Use -- for filename to send raw PCM to stdout
```

Command:

```
tinycap /userdata/test.wav -D 0 -d 0 -t 5
```

Play: Play 1kHz sinusoidal wave, listen to the tune or use codec to output waveform and use digital oscilloscope to check signal waveform.

```
root@x3dvbx3-hynix1G-2666:/userdata# tinyplay
usage: tinyplay file.wav [options]
options:
-D | --card   <card number>    The device to receive the audio
-d | --device <device number>  The card to receive the audio
-p | --period-size <size>      The size of the PCM's period
-n | --period-count <count>    The number of PCM periods
-i | --file-type <file-type >  The type of file to read (raw or wav)
-c | --channels <count>        The amount of channels per frame
-r | --rate <rate>            The amount of frames per second
-b | --bits <bit-count>        The number of bits in one sample
-M | --mmap                   Use memory mapped IO to play audio
```

Comannd:

```
tinyplay /userdata/1khz.wav -D 0 -d 1
```

4. Tinymix Debug:

```
root@x3dvbx3-hynix1G-2666:/userdata# tinymix contents
Number of controls: 79
ctl     type    num     name                            value
0       INT     1       ADC1 PGA gain                    0 (range 0->31)
1       INT     1       ADC2 PGA gain                    0 (range 0->31)
2       INT     1       ADC3 PGA gain                    0 (range 0->31)
3       INT     1       ADC4 PGA gain                    0 (range 0->31)
4       INT     1       CH1 digital volume               160 (range 0->255)
5       INT     1       CH2 digital volume               160 (range 0->255)
6       INT     1       CH3 digital volume               160 (range 0->255)
7       INT     1       CH4 digital volume               160 (range 0->255)
8       BOOL    1       CH1 ch1 mixer gain               Off
9       BOOL    1       CH1 ch2 mixer gain               Off
10      BOOL    1       CH1 ch3 mixer gain               Off
11      BOOL    1       CH1 ch4 mixer gain               Off
```

Use ctr or name to control, set or get parameters, for example:

Using ctr:

```
root@x3dvbx3-hynix1G-2666:/userdata# tinymix set 6 120
root@x3dvbx3-hynix1G-2666:/userdata# tinymix get 6
120 (range 0->255)
```

Using Name:

```
root@x3dvbx3-hynix1G-2666:/userdata# tinymix set "DAC MIXER RECD_DAT switch" 1
root@x3dvbx3-hynix1G-2666:/userdata#
root@x3dvbx3-hynix1G-2666:/userdata# tinymix get "DAC MIXER RECD_DAT switch" On
```

Using the previous command to do audio card creation and simple debug, the "tiny*" utilities source code is listed under: hbre/libalsa/tinyalsa/utils

7.3.2 Common Debug Methods

1. Check codec or I2S registers. When recording or playing malfunctions, make sure if registers are configured correctly:

Check I2S regitsters:

```
cat /sys/devices/platform/soc/a5008000.i2s/reg_dump
```

Check codec registers: for example, check ac108 codec registers, which is connected to I2C0, with the address of 0x3b/0x35:

```
cat /sys/devices/platform/soc/a5009000.i2c/i2c-0/0-003b/ac108_debug/ac108
```

2. Use digital Oscilloscope to measure the audio signals, including mclk/bclk/lrck/data, make sure clk rate bclk/lrck comply with the configured sampling rate adjust if required;

8 SDIO Porting Guide

8.1 About SDIO

8.1.1 SDIO Version

The highest SDIO protocol version supported is Ver3.0, including eSDIO. The detailed description of SDIO protocol, please refer to the protocol manual.

8.1.2 Clock Frequency

SDIO input clock frequencies available on X3J3 can be calculated by: 1536Mhz/(8 * [1-16]). If any other frequency is needed, please contact Horizon for clock tree modification.

8.1.3 Transfer Mode Support

SDIO on X3J3 support 4 bit SDR mode, default mode supported includes:

- SD High Speed: 52MHz, actual clock provided is 48MHz
- SDR12
- SDR25
- SDR50
- SDR104

Detailed mode configuration is explained in "DTS modification"

Currently X3J3 does not support DDR transfer mode.

8.2 Porting Guide

This chapter will introduce the basic procedure of porting SDIO devices.

8.2.1 Porting Procedure

8.2.1.1 Device Driver

Please prepare the device driver according to the product requirement. Use in-tree or out-of-tree module compile to suit the need of the product.

- If in-tree module compile is used, please refer to "defconfig modification" to add compile configuration;

- If out-of-tree module compile is used, please compile the device driver **AFTER** a complete compile of the kernel is completed. And transfer the module file to X3J3.

For module compile Makefile, please refer to the official document from Linux Kernel. Below is an example:

The main interactive function between device driver and SDIO:

- Driver registration:

```
int sdio_register_driver(struct sdio_driver *drv)
```

- Driver unregistration:

```
void sdio_unregister_driver(struct sdio_driver *drv)
```

sdio_driver struct is defined below:

```
struct sdio_driver {
    char *name;
    const struct sdio_device_id *id_table;

    int (*probe)(struct sdio_func *, const struct sdio_device_id *);
    void (*remove)(struct sdio_func *);

    struct device_driver drv;
};
```

8.2.1.2 DTS Modification

Currently, X3J3 disabled SDIO port by default in DTS. Please enable it manually by adding the following line in the DTS:

```
&sdio2 {
    status = "okay";
}
```

If any extra transfer modes are required for the device, please add the mode in the product DTS. For specific description of mode, please refer to SDIO protocol manual.

If input clock needs to be customized, please add "clock-frequency" to "sdio2" in DTS, measured in Hz. Input clock is determined by clock tree configuration, please contact Horizon for details:

```
&sdio2 {
    status = "okay";
    clock-frequency = <52000000>;
}
```

If bus speed needs to be customized, please add "clock-freq-min-max" to the product

DTS for "sdio2" measured in Hz. Bus speed is determined by controller clock divider. for example:

```
&sdio2 {
    status = “okay”;
    clock-freq-min-max = <100000 52000000>;
}
```

Other SDIO configuration is done in "arch/arm64/boot/dts/hobot/hobot-xj3.dtsi" and " arch/arm64/boot/dts/hobot/hobot-xj3-xvb.dtsi ". Detailed meaning of the configurations can be found in mmc framework code.

8.2.1.3 Kernel defconfig Modification

SDIO itself does not require any modification to defconfig. Please modify Kernel defconfig to suit the need of the ported device.

8.3 Verify SDIO

This chapter takes Marvell SD8801 WiFi module as an example to demonstrate how to port and verify SDIO

8.3.1 In-Tree Compile

After in-tree compile is completed, the dependency of the module is generated automatically. After SDIO2 is enabled in DTS, Kernel will automatically probe WiFi module and load driver. During boot, the following message should print in dmesg:

```
[    2.126471] mlan: module is from the staging directory, the quality is unknown, you have been warned.
[    2.127677] mlan: module license 'Marvell Proprietary' taints kernel.
[    2.128605] Disabling lock debugging due to kernel taint
[    2.137005] sd8xxx: module is from the staging directory, the quality is unknown, you have been warned.
```

WiFi module can also be checked by:

```
ifconfig mlan0
```

Use the following command to load SD8801 with nl80211:

```
ifconfig mlan0 down
rmmod sd8xxx
insmod /lib/modules/4.14.74/sd8xxx.ko drv_mode=1 cfg80211_wext=0x0f
fw_name=mrvl/sd8801_uapsta.bin
```

Configure WiFi connection information: in the below example, "<WiFi Name>" and "<WiFi Password>" should be substitute with the actual WiFi connection configuration.

```
mount -o remount,rw /
wpa_passphrase <WiFi Name><WiFi Password> >> /userdata/wpa_supplicant.conf
```

Start WiFi and request ip:

```
wpa_supplicant -D nl80211 -i mlan0 -c /userdata/wpa_supplicant.conf -B
udhcpc -i mlan0
```

Test WiFi connection

```
ping www.baidu.com
```

8.3.2 Out-of-Tree Compile

After Out-of-Tree compile is completed, load all dependent modules manually:

```
insmod /lib/modules/4.14.74/cfg80211.ko
insmod /lib/modules/4.14.74/mac80211.ko
insmod /lib/modules/4.14.74/mlan.ko
```

The rest is the same as in-tree compile.

9 I2C Debug Guide

9.1 Introduction

Horizon Robotics X3J3 provides standard I2C bus. I2C bus controller utilizes SDA and SCL to transfer data between devices connected to the I2C bus. Every single device attached has a unique address (regardless of MCU, LCD Controller, Storage or keyboard). Every device is potentially a transmitter or a receiver (depending on the device functionality). I2C controller supports:

- SMBus and I2C compatibility
- Software programmable clock frequency upto 400 kbps
- 7/10-bit bus addressing

9.2 Kernel Driver

```
drivers/i2c/busses/i2c-hobot.c   # I2C Driver Source Code
inlcude/linux/i2c-hobot.h        # I2C Driver Header
```

9.2.1 Kernel DTS Node Configuration

```
/* arch/arm64/boot/dts/hobot/hobot-xj3.dtsi */
i2c0: i2c@0xA5009000 {
    compatible = "hobot,hobot-i2c";
    reg = <0 0xA5009000 0 0x100>;
    interrupt-parent = <&gic>;
    interrupts = <0 38 4>;
    clocks = <&i2c0_mclk>;
    clock-names = "i2c_mclk";
    resets = <&rst 0x50 10>;
    reset-names = "i2c0";
    status = "disabled";
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_func>;
};
```

For detailed I2C manual, please refer to Linux Kernel Documentation/i2c. This chapter focuses only on X3J3 I2C driver modifications unique to X3J3。

9.2.2 Driver Code Configuration 驱动代码配置

Default frequency of I2C can be specified in i2c-hobot.c:

```
/* drivers/i2c/busses/i2c-hobot.c */
#define I2C_SCL_DEFAULT_FREQ    400000 /*I2c SCL default frequency is 400KHZ*/
```

9.3 I2C Usage

The detailed explanation of I2C can be found under Linux Kernel source: Documentation/i2c. This chapter will focus on the changes done specifically for X3J3

9.3.1 Kernel Space

X3J3 I2C driver provides function for configuring I2C transmission frequency in Kernel Space:

```
#include <linux/i2c-hobot.h>
...
{
    struct client_request *client_data = (struct client_request *)(client->adapter->algo_data);
    ...
    client_data->client_req_freq = 100000;  //Configure I2C Transmission frequency to 100kHz
    ret = i2c_transfer(client->adapter, request, ARRAY_SIZE(request));
    ...
}
```

Note: If the targeted transmission frequency is not the same as the default frequency, it has to be configured before every transmission. In other words, frequency configuration is valid for the period of every transmission only to prevent cross interference between different I2C devices. The code is listed below:

```
/* Check is re-configuration of frequency is needed, if so, configure I2C frequency as needed */
static void recal_clk_div(struct hobot_i2c_dev *dev)
{
    u32 clk_freq = 0;
    int temp_div = 0;
    struct client_request *client_req;

    client_req = (struct client_request *)dev->adapter.algo_data;
    clk_freq = clk_get_rate(dev->clk);
    if (client_req->client_req_freq != 0) {
        temp_div = DIV_ROUND_UP(clk_freq, client_req->client_req_freq) - 1;
    } else {
        temp_div = DIV_ROUND_UP(clk_freq, dev->default_trans_freq) - 1;
    }
    dev->clkdiv = DIV_ROUND_UP(temp_div, 8) - 1;
    if (dev->clkdiv > I2C_MAX_DIV) {
        dev_dbg(dev->dev, "clkdiv too large, set to 255");
```

```
        dev->clkdiv = I2C_MAX_DIV;
    }
}

/* reset I2C frequency to default */
static void reset_client_freq(struct hobot_i2c_dev *dev)
{
    struct client_request *client_req;

    client_req = (struct client_request *)dev->adapter.algo_data;
    client_req->client_req_freq = 0;
}

/* I2C master_xfer function*/
static int hobot_i2c_xfer(struct i2c_adapter *adap, struct i2c_msg msgs[], int num)
{
    ...
    hobot_i2c_reset(dev);
    ... /* I2C transfer */
    reset_client_freq(dev);
    ...
}
```

9.3.2 User Space

Usually, I2C device is controlled by Kernel I2C driver. But userspace access to devices on I2C bus is also provided by /dev/i2c-%d sysfs nodes. The document under: Documentation/i2c/dev-interface has detailed usage explained

9.3.2.1 Frequency Configuration

Check the frequency of I2C-N, take i2c-0 as an example:

```
cat /sys/bus/i2c/devices/i2c-0/speed
```

And the following message should print on console:

```
root@x3dvbx3-hynix1G-2666:~# cat /sys/bus/i2c/devices/i2c-0/speed
400000
```

Set he frequency of I2C-N, take i2c-0 as an example:

```
echo 100000 > /sys/bus/i2c/devices/i2c-0/speed
cat /sys/bus/i2c/devices/i2c-0/speed
```

And the following message should print on console:

```
root@x3dvbx3-hynix1G-2666:~# echo 100000 > /sys/bus/i2c/devices/i2c-0/speed
root@x3dvbx3-hynix1G-2666:~# cat /sys/bus/i2c/devices/i2c-0/speed
100000
```

On contrary to the Kernel Space configuration, userspace configuration is effective permanently, please proceed with cautions.

9.3.2.2 I2c-tools

"i2c-tools" is a set of open-source tools integrated into X3J3 rootfs:

- i2cdetect – list all I2C buses and all devices on each I2C buses
- i2cdump – dump the register values of the selected I2C device
- i2cget – read the register value of a specific register from an I2C device
- i2cset – write a value to a register to an I2C device

10 GPIO Debug Guide

10.1 Kernel Drivers

Path to GPIO driver source code relative to Kernel source folder:

```
drivers/gpio/gpio-hobot-x3.c
```

10.1.1 Kernel defconfig

Path to Kernel defconfig files:

```
arch/arm64/configs/xj3_debug_defconfig   # debug
arch/arm64/configs/xj3_perf_defconfig    # release
```

GPIO Driver defconfig option:

```
CONFIG_GPIO_HOBOT_X3=y
```

10.1.2 Kernel DTS Configuration

```
/* arch/arm64/boot/dts/hobot/hobot-xj3.dtsi */
gpios: gpio@0xA6003000 {
    compatible = "hobot,x3-gpio";
    reg = <0 0xA6003000 0 0x100>;
    gpio-controller;
    #gpio-cells = <2>;
    gpio-ranges = <&pinctrl 0 0 121>;
    interrupts = <0 54 4>;
    interrupt-parent = <&gic>;
    interrupt-controller;
    #interrupt-cells = <2>;
};
```

Note:

"hobot-xj3.dtsi" declared general resources for I2C registers, interrupts defined by SoC, regardless of the exact product, should generally be left unchanged.

10.2 GPIO Usage

10.2.1 Kernel Space

10.2.1.1 DTS Congifguration

GPIO device tree node is named in convention of "names-gpios" or "names-gpio", for example:

```
/* arch/arm64/boot/dts/hobot/hobot/hobot-xj3-xvb.dtsi */
&usb_id {
    status = "okay";
    /* vbus-gpio = <>; */
    id-gpio = <&gpios 65 GPIO_ACTIVE_HIGH>;
};
```

10.2.1.2 Kernel Driver Interfaces

Use the header below:

```
include/linux/gpio.h
```

Request certain GPIO:

```
int gpio_request(unsigned gpio, const char *label);
```

Initialize GPIO to output and set output value:

```
int gpio_direction_output(unsigned gpio, int value);
```

Initialize GPIO to input:

```
int gpio_direction_input(unsigned gpio);
```

Get GPIO value:

```
int gpio_get_value(unsigned int gpio);
```

Set GPIO Value:

```
void gpio_set_value(unsigned int gpio, int value);
```

Free GPIO:

```
void gpio_free(unsigned gpio)
```

Request GPIO interrupt, the return value can be used for "request_irq" and "free_irq":

```
int gpio_to_irq(unsigned int gpio);
```

10.2.1.3 X3J3 GPIO IRQ

X3J3 has 121 GPIO pins but only 4 hardware gpio interrupt in the GPIO module. During operation, a maximum of 4 pins out of 121 can be mapped to the 4 IRQ interrupt. The mapping process is managed by the GPIO driver, other device drivers need to use "gpio_to_irq" function to request IRQ. Once all 4 hardware IRQs are mapped, all the following request will fail. The IRQ mapping code is shown below:

```
/* drivers/gpio/gpio-hobot-x3.c */
/* initialize irq mapping table */
void init_irqbank(struct x3_gpio *gpo) {
    int i = 0;

    for (i = 0; i < GPIO_IRQ_BANK_NUM; i++) {
        gpo->irqbind[i] = GPIO_IRQ_NO_BIND;
    }
}

/* request irq and update mapping table */
int request_irqbank(struct x3_gpio *gpo, unsigned long gpio) {
    int i = 0, index = GPIO_IRQ_NO_BIND;

    index = find_irqbank(gpo, gpio);
    if (index == GPIO_IRQ_NO_BIND) {
        for (i = 0; i < GPIO_IRQ_BANK_NUM; i++) {
            if (gpo->irqbind[i] == GPIO_IRQ_NO_BIND) {
                gpo->irqbind[i] = gpio;
                index = i;
                break;
            }
        }
    } else {
        dev_err(gpo->dev, "gpio(%ld) has be binded\n", gpio);
        return GPIO_IRQ_NO_BIND;
    }
    return index;
}

/* free gpio and update mapping table */
void release_irqbank(struct x3_gpio *gpo, unsigned long gpio) {
    int index = GPIO_IRQ_NO_BIND;
```

```
    index = find_irqbank(gpo, gpio);
    if (index != GPIO_IRQ_NO_BIND) {
        gpo->irqbind[index] = GPIO_IRQ_NO_BIND;
    }
}
```

Note:

X3J3 GPIO utilizes the standard Linux interface in Kernel space. For more details, please refer to Documentation/gpio/consumer.txt

10.2.2 User Space

10.2.2.1 Control Interface

Userspace application can write GPIO number to /sys/class/gpio/export to export the control of certain GPIO to usersapce, for example:

```
echo 42 > export
```

To release control of gpio from userspace:

```
echo 42 > unexport
```

The path to GPIO controller:

```
/sys/class/gpio/gpiochip0
```

10.2.2.2 Usage

After export the control of GPIO 42 to userspace as shown above, the control path "/sys/class/gpio/gpio42/" should appear and the below sysfs node should be listed:

- direction: indicates the direction of GPIO, write value in/out to it to set direction and read from it to get the current direction
- value: indicates the logical voltage level of the GPIO pin, "0" indicating low, and "1" indicating high. If the direction of GPIO is set to "output", then this value is r/w
- edge: indicates the trigger mode of GPIO, 4 total modes:
  - "none": not an interrupt GPIO
  - "rising": interrupt GPIO triggered by rising edge
  - "falling": interrupt GPIO triggered by falling edge
  - "both": interrupt GPIO triggered by both rising and falling edge

10.2.2.3 Debug Interface

If the "DEBUGFS" option is enabled in arch/arm64/configs/xj3_debug_defconfig or arch/arm64/configs/xj3_perf_defconfig, " sys/kernel/debug/gpio " can be used to read the current GPIO in use:

```
root@x3dvbx3-hynix1G-2666:~# cat /sys/kernel/debug/gpio
gpiochip0: GPIOs 0-120, parent: platform/a6003000.gpio:
 gpio-42  (                    |sysfs               ) in  hi
 gpio-64  (                    |cd                  ) in  lo IRQ
 gpio-65  (                    |id                  ) in  hi IRQ
 gpio-100 (                    |?                   ) out lo
 gpio-120 (                    |?                   ) out hi
root@x3dvbx3-hynix1G-2666:~#
```

Note:

X3J3 GPIO utilizes the standard Linux interface in user space. For more details, please refer to Documentation/gpio/sysfs.txt

11 Pinctrl Debug Guide

11.1 Introduction

X3J3 pinctrl uses Linux Kernel mainline code pintrl-single as the device driver. Mainly use DTS to implement pinctrl functionality.

11.2 Kernel Driver

11.2.1 Path to Kernel Driver

The path to pinctrl device driver source code:

```
drivers/pinctrl/pinctrl-single.c
```

The path to pinctrl device driver header file:

```
include/linux/platform_data/pinctrl-single.h
```

11.2.2 Pinctrl DTS Configuration

Path to pinctrl DTS: arch/arm64/boot/dts/hobot/hobot-pinctrl-xj3.dtsi

```
pinctrl: pinctrl@0xA6004000 {
    compatible = "pinctrl-single";
    reg = <0x0 0xA6004000 0x0 0x200>;
    #pinctrl-cells = <1>;
    #gpio-range-cells = <0x3>;
    pinctrl-single,register-width = <32>;
    pinctrl-single,function-mask = <0x3FF>;
    /* pin base, nr pins & gpio function */
    pinctrl-single,gpio-range = <&range 0 120 3>;

    i2c0_func: i2c0_func {
        pinctrl-single,pins = <
            0x020   (MUX_F0 | DRIVE2_09MA | SCHMITT2_DIS | PULL2_UP)
            0x024   (MUX_F0 | DRIVE2_09MA | SCHMITT2_DIS | PULL2_UP)
            >;
    };
    ...
}
```

11.3 Pinctrl Usage

11.3.1 Driver DTS Configuration

Before any device drivers call Pinctrl interfaces, there must be corresponding pinctrl configuration group in the DTS. When the device drives are probed, "default" group will be written to the register. The other groups of configurations need to be read and processed for further usage, take IAR as an example:

```
/* arch/arm64/boot/dts/hobot/hobot-xj3.dtsi */
iar: iar@0xA4001000 {
    compatible = "hobot,hobot-iar";
    reg = <0 0xA4301000 0 0x400>, <0 0xA4355000 0 0x1000>;
    clocks = <&iar_pix_clk>, <&iar_ipi_clk>, <&sif_mclk>;
    clock-names = "iar_pix_clk","iar_ipi_clk", "sif_mclk";
    interrupt-parent = <&gic>;
    interrupts = <0 69 4>;
    resets = <&rst 0x40 12>;
    pinctrl-names = "bt_func", "rgb_func", "rgb_gpio_func", "bt1120_voltage_func";
    pinctrl-0 = <&btout_func>;
    pinctrl-1 = <&rgb_func>;
    pinctrl-2 = <&rgb_gpio_func>;
    pinctrl-3 = <&bt1120_1_8v_func>;
    disp_panel_reset_pin = <28>;
    reset-names = "iar";
    status = "disabled";
};
```

The pinctrl of IAR used is listed under: arch/arm64/boot/dts/hobot/hobot-pinctrl-xj3.dtsi

```
btout_func: btout_func {
    pinctrl-single,pins = <
    0x138   (MUX_F0 | DRIVE1_12MA | SCHMITT2_ENA | PULL2_DOWN) /*BT1120_OUT_CLK*/
    0x13c   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN) /*BT1120_OUT_DAT0*/
    0x140   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x144   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x148   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x14c   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x150   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x154   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x158   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN) /*BT1120_OUT_DAT7*/
    0x15c   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN) /*BT1120_OUT_DAT8*/
    0x160   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x164   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
```

```
    0x168   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x16c   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x170   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x174   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN)
    0x178   (MUX_F0 | DRIVE1_12MA | SCHMITT2_DIS | PULL2_DOWN) /*BT1120_OUT_DAT15*/
    >;
};
```

The configuration group of pinctrl combines multiple pin configurations together, each pin configuration consists 2 column:

the first column indicates the offset of the pin configuration register, which can be calculated by <pin num> * 4. For example, BT1120_OUT_CLK has pin number of 78, then the offset is 78 * 4 = 312, denoted as 0x138 in hex

the second column indicates the configuration of the pin, the available configurations is introduced below.

11.3.1.1 Pin-mux Configuration

Every pin of X3J3 supports a maximum of 4 functions. When configuring which function to use, please check the Spec of X3J3 to determine the value of pinmux corresponding to the targeted function. For example, pin 78 has the pinmux configured to 0, which is the MUX_F0 of pin, BT1120_OUT_CLK.

```
/* include/dt-bindings/pinctrl/hobot-xj3.h */
/* MUX functions for pins */
#define MUX_F0      0
#define MUX_F1      1
#define MUX_F2      2
#define MUX_F3      3
```

11.3.1.2 Pin Drive Strength Configuration

Every pin of X3J3 supports configuration of the output current. Drive current configuration of each pin is divided into two categories. Same register value on different pins yields different output current. For example, on "DRIVE1" category, register value 0 indicates 3mA, but on "DRIVE2", 0 indicates 6mA. The Category of pin drive is listed under "arch/arm64/boot/dts/hobot/hobot-pinctrl-xj3.dtsi".

```
/* include/dt-bindings/pinctrl/hobot-xj3.h */
/*
 * DRIVE1 > bit5:2(0000b~1111b==3mA~45mA
          { 3, 6,  9, 12, 17, 20, 22, 25, 33, 35, 37, 39, 41, 42.5, 44, 45})
 * DRIVE2 -> bit5:2(0000b~1111b==6mA~45mA
```

```
            { 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 41, 42.5, 44, 45})
 *
 */
/* drive strength definition */
#define DRIVE_MASK  (4 << 2)
#define DRIVE1_03MA (0 << 2)
#define DRIVE2_06MA (0 << 2)
#define DRIVE1_06MA (1 << 2)
#define DRIVE2_09MA (1 << 2)
#define DRIVE1_09MA (2 << 2)
#define DRIVE2_12MA (2 << 2)
#define DRIVE1_12MA (3 << 2)
#define DRIVE2_15MA (3 << 2)
#define DRIVE1_17MA (4 << 2)
#define DRIVE2_18MA (4 << 2)
#define DRIVE1_20MA (5 << 2)
#define DRIVE2_21MA (5 << 2)
#define DRIVE1_22MA (6 << 2)
#define DRIVE2_24MA (6 << 2)
#define DRIVE1_25MA (7 << 2)
#define DRIVE2_27MA (7 << 2)
#define DRIVE1_33MA (8 << 2)
#define DRIVE2_30MA (8 << 2)
#define DRIVE1_35MA (9 << 2)
#define DRIVE2_33MA (9 << 2)
#define DRIVE1_37MA (10 << 2)
#define DRIVE2_36MA (10 << 2)
#define DRIVE1_39MA (11 << 2)
#define DRIVE2_39MA (11 << 2)
#define DRIVE1_41MA (12 << 2)
#define DRIVE2_41MA (12 << 2)
#define DRIVE1_42_5MA   (13 << 2)
#define DRIVE2_42_5MA   (13 << 2)
#define DRIVE1_44MA (14 << 2)
#define DRIVE2_44MA (14 << 2)
#define DRIVE1_45MA (15 << 2)
#define DRIVE2_45MA (15 << 2)
```

11.3.1.3 Pullup Pulldown Configuration

Every pin of X3J3 also supports pullup/pulldown configuration, similar to drive strength, pullup/pulldown is also divided into 2 categories. The bit of configuration of different category is different. The exact category of each pins is listed under "hobot-pinctrl-

xj3.dtsi"

```
/* include/dt-bindings/pinctrl/hobot-xj3.h */
/*
 * PULL1 ->  bit7(0==pulldown, 1==pullup)
 *           bit6(0==pull disable, 1==pull enable)
 *
 * PULL2 ->  bit8(0==pullup enable, 1==pullup enable)
 *           bit7(0==pulldown diable, 1==pulldown enable)
 */
/* pin states bits */
#define PULL1_MASK  (3 << 6)
#define PULL2_MASK  (3 << 7)
#define PULL1_EN     (1 << 6)
#define PULL1_DIS   (0)
#define PULL2_DIS   (0)
#define PULL1_UP    (PULL1_EN | (1 << 7))
#define PULL2_UP    (1 << 8)
#define PULL1_DOWN  (PULL1_EN | (0 << 7))
#define PULL2_DOWN  (1 << 7)
```

11.3.1.4 Schmitt Trigger Configuration

Every pin of X3J3 also supports configuration of Schmitt trigger, similar to the previous configurations, Schmitt trigger configuration is divided into 2 categories, different category requires changes to different bits in register. The exact category of each pins is listed under "hobot-pinctrl-xj3.dtsi"

```
/* include/dt-bindings/pinctrl/hobot-xj3.h */
/*
 * SCHMITT1 -> bit8(0==diable, 1==enable)
 *
 * SCHMITT2 -> bit9(0==diable, 1==enable)
 */
/* pin schmitt */
#define SCHMITT1_ENA    (1 << 8)
#define SCHMITT1_DIS    (0 << 8)

#define SCHMITT2_ENA    (1 << 9)
#define SCHMITT2_DIS    (0 << 9)
```

11.3.2 Device Driver Sample Code

Drivers needs to first find the pinctrl state corresponding to pinctrl-names then switch

to the target state:

```
static int hobot_xxx_probe(struct platform_device *pdev)
{
    ...
     g_xxx_dev->pinctrl = devm_pinctrl_get(&pdev->dev);
    if (IS_ERR(g_xxx_dev->pinctrl)) {
        dev_warn(&pdev->dev, "pinctrl get none\n");
        g_xxx_dev->pins_xxxx = NULL;
    }
    ...
    /* lookup state according to pinctrl-names */
    g_xxx_dev->pins_xxxx = pinctrl_lookup_state(g_xxx_dev->pinctrl, "xxx_func");
        if (IS_ERR(g_xxx_dev->pins_xxxx)) {
            dev_info(&pdev->dev, "xxx_func get error %ld\n", PTR_ERR(g_xxx_dev->pins_xxxx));
            g_xxx_dev->pins_xxxx = NULL;
        }
    ...
}

int xxxx_pinmux_select(void)
{
    if (!g_xxx_dev->pins_xxxx)
        return -ENODEV;
    /* Switch to corresponding state */
    return pinctrl_select_state(g_xxx_dev->pinctrl, g_xxx_dev->pins_xxxx);
}
```

11.3.3 Userspace Debug Guide

If "DEBUG_FS" option is enabled in

```
arch/arm64/configs/xj3_debug_defconfig
arch/arm64/configs/xj3_perf_defconfig
```

during Kernel compile, then sysfs node /sys/kernel/debug/pinctrl/a6004000.pinctrl/ can be used to debug in userspace.

```
cat /sys/kernel/debug/pinctrl/a6004000.pinctrl/pinmux-pins
```

12 IO-DOMAIN Debug Guide

12.1 Introduction

IO-Domain can be used to configure the voltage domain of part of the modules. Take RGMII as an example, if the power supply to the module is 3.3V, then the IO-Domain of the RGMII module needs to be configured to be 3.3V and vice versa.

If the power supply is 3.3v but the IO-Domain is configured to 1.8V, the Chip can be damaged;

If the power supply is 1.8v but the IO-Domain is configured to 3.3V, the Chip might malfunction.

12.2 Kernel Driver

12.2.1 Path to Source Files

Path to the source code of pinctrl driver:

```
drivers/pinctrl/pinctrl-single.c
```

Path to the header of pinctrl driver:

```
include/linux/platform_data/pinctrl-single.h
```

12.2.2 IO-DOMAIN DTS

```
/* arch/arm64/boot/dts/hobot/hobot-pinctrl-xj3.dtsi */
/*  pinctrl_voltage used to config X/J3 pin mode, for example,
 *  when SD2 external power supply is 3.3v, we need config pin-mode to
 *  3.3v, otherwise X/J3 chip will be damaged.
 *  when SD2 external power supply is 1.8v, we need config pin-mode to
 *  1.8v, otherwise SD2 will not work.
 */
pinctrl_voltage: pinctrl_voltag@0xA6003000 {
    compatible = "pinctrl-single";
    reg = <0x0 0xA6003170 0x0 0x8>;
    #pinctrl-cells = <2>;
    #gpio-range-cells = <0x3>;
    pinctrl-single,bit-per-mux;
    pinctrl-single,register-width = <32>;
    pinctrl-single,function-mask = <0x1>;
    status = "okay";
    /* rgmii 1.8v func */
```

```
    rgmii_1_8v_func: rgmii_1_8v_func {
        pinctrl-single,bits = <
            0x4 MODE_1_8V RGMII_MODE_P1
            0x4 MODE_1_8V RGMII_MODE_P0
            >;
    };
    /*rgmii 3.3v func */
    rgmii_3_3v_func: rgmii_3_3v_func {
        pinctrl-single,bits = <
            0x4 MODE_3_3V RGMII_MODE_P1
            0x4 MODE_3_3V RGMII_MODE_P0
            >;
    };
    ...
};
```

IO-Domain is implemented under the framework of pinctrl-single from Linux Kernel, thus the DTS configuration of IO-Domain is similar to pinctrl. In the DTS of IO-Domain, all 1.8V and 3.3V configuration group is already listed and should generally be left unchanged.

12.2.3 Device Driver Client DTS Configuration

Similar to Pinctrl, the client device driver need to include the IO-Domain targeted, take BT1120 driver as an example:

```
xxx: xxx@0xA6000000 {
    ...
    pinctrl-names = "default", "xxx_voltage_func", ;
    pinctrl-0 = <&xxx_func>;
    pinctrl-1 = <&xxx_1_8v_func>;  // pinctrl-3 is the IO-Domain of 1.8v
    ...
};
```

12.2.4 Client Device Driver Sample Code

Similar to Pinctrl, client drivers needs to first find the pinctrl state corresponding to pinctrl-names then switch to the target state:

```
static int hobot_xxx_probe(struct platform_device *pdev)
{
    ...
    g_xxx_dev->pinctrl = devm_pinctrl_get(&pdev->dev);
    if (IS_ERR(g_xxx_dev->pinctrl)) {
        dev_warn(&pdev->dev, "pinctrl get none\n");
```

```
        g_xxx_dev->pins_voltage = NULL;
    }
    ...
    /* According to pinctrl-names lookup state */
    g_xxx_dev->pins_voltage = pinctrl_lookup_state(g_xxx_dev->pinctrl, "xxx_voltage_func");
    if (IS_ERR(g_xxx_dev->pins_voltage)) {
        dev_info(&pdev->dev, "xxx_voltage_func get error %ld\n", PTR_ERR(g_xxx_dev->pins_voltage
));
        g_xxx_dev->pins_voltage = NULL;
    }
    ...
    /* select state */
    if (g_xxx_dev->pins_voltage) {
        ret = pinctrl_select_state(g_xxx_dev->pinctrl, g_xxx_dev->pins_voltage);
        if (ret) {
            dev_info(&pdev->dev, "xxx_voltage_func set error %d\n", ret);
        }
    }
    ...
}
```

13 ADC Debug Guide

13.1 Introduction

The PVT module (Process Voltage Temperature) on X3J3 has 3 functions:

- Process Detector: no software driver
- Voltage Monitor: 16 channel volage detector, used to monitor the voltage of part of the modules on-chip
- Temperature Sensor: 4 temperature sensors, used to monitor the temperature of part of the modules on-chip

Among the 16 channels of voltage monitor, one of them is connected to a pin on chip, thus can be used to implement a lightweight ADC. This channel can be used to detect 0-1V voltage, sampling at 14bit.

13.2 Kernel Drivers

Path to pvt-vm driver source code, also implements the ADC module driver:

```
drivers/hwmon/hobot-pvt-vm.c
```

Path to pvt-vm driver and ADC module driver header:

```
drivers/hwmon/hobot-pvt.h
```

13.2.1 Kernel Configuration

Path to Kernel defconfig files:

```
arch/arm64/configs/xj3_debug_defconfig   # debug
arch/arm64/configs/xj3_perf_defconfig    # release
```

Kernel defconfig options:

```
CONFIG_IIO=y                                # kernel iio device support
CONFIG_SENSORS_HOBOT_PVT=y                   # X3J3 I2C driver support
```

13.2.2 Kernel DTS Configuration

```
/* arch/arm64/boot/dts/hobot/hobot-xj3.dtsi */
pvt: pvt@0xA1009000 {
    compatible = "hobot,hobot-pvt";
    reg = <0 0xA1009000 0 0x1000>,
          <0 0xA6008000 0 0x100>;
    interrupt-parent = <&gic>;
```

```
    interrupts = <0 64 4>;
    clocks = <&sys_div_pclk>;
    clock-names = "sys_pclk";
    #thermal-sensor-cells = <0>;
    voltage_monitor: voltage_monitor {
        #io-channel-cells = <1>;
    };
};
```

Note:

PVT related DTS does not contain board level configurations, should generally be left unchanged. If PVT VM module needs to be disabled, delete voltage_monitor nodes in the Board level DTS.

13.3 ADC Usage

13.3.1 Kernel Space

13.3.1.1 DTS Configuration

Clients in Kernel Space need to first configure ADC node and channel in the DTS before using the ADC channel:

```
/* arch/arm64/boot/dts/hobot/hobot-xj3.dtsi */
adc_key: adc_key {
    compatible = "hobot,hobot-adc-key";
    status = "disabled";
    io-channels = <&voltage_monitor 0>;
    demo-key {
        linux,code = <108>;
        label = "demo key";
        hobot,adc_value = <130>;
    };
};
```

Then call "iio_channel_get" function to acquire iio channgel:

```
/* drivers/input/misc/hobot-key.c */
static int hobot_keys_parse_dt (...)
{
    struct iio_channel *chan;
    ...
    chan = iio_channel_get(&pdev->dev, NULL);
```

```
    if (IS_ERR(chan)) {
        dev_info(&pdev->dev, "no io-channels defined\n");
        chan = NULL;
        return -ENAVAIL;
    }
    ...
}
```

After the iio channel is successfully acquired, use "iio_read_channel_raw" to read the value of ADC channel measured in μV.

```
/* drivers/input/misc/hobot-key.c */

static int hobot_key_adc_iio_read (struct hobot_keys_drvdata *data)
{
    struct iio_channel *channel = data->chan;
    int val, ret;

if (!channel)
        return INVALID_ADVALUE;
    ret = iio_read_channel_raw(channel, &val);
    if (ret < 0) {
        pr_err("read channel() error: %d\n", ret);
        return ret;
    }
    return val / 1000;
}
```

13.3.2 User Space

Each ADC channel can use sysfs node to read its value, use channel 0 as an example:

```
# in_voltageN_raw meand channel N
cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
```

13.4 Appendix

"drivers/input/misc/hobot-key.c" is a sample ADC keyboard driver, please modify to fit the need of product.

14 SPI Debug Guide

14.1 Kernel Driver

14.1.1 Path to Source Code

Char device sourcecode:

```
drivers/spi/spidev.c
```

SPI Framework source code:

```
drivers/spi/spi.c
```

SPI Driver source code:

```
drivers/spi/spi-hobot.c
```

14.1.2 Kernel Configuration

The path to defconfig:

- Debug version: arch/arm64/configs/xj3_debug_defconfig
- Release version: arch/arm64/configs/xj3_perf_defconfig

Options for SPI debugging:

```
CONFIG_SPI_SPIDEV=y
CONFIG_SPI_SLAVE=y
CONFIG_SPI_HOBOT=y
```

14.1.3 DTS Device Node Configuration

Add the corresponding device nodes in the following DTS files and compile Kernel:

Path: arch/arm64/boot/dts/hobot/hobot-xj3.dtsi

```
spi0: spi@0xA5004000 {
    compatible = "hobot,hobot-spi";
    reg = <0 0xA5004000 0 0x1000>;
    clocks = <&spi0_mclk>;
    clock-names = "spi_mclk";
    interrupt-parent = <&gic>;
    interrupts = <0 33 4>;
    resets = <&rst 0x50 4>;
    reset-names = "spi0";
    pinctrl-names = "default";
```

```
    pinctrl-0 = <&spi0_func>;
    status = "disabled";
    #address-cells = <1>;
    #size-cells = <0>;
};
```

Path: arch/arm64/boot/dts/hobot/hobot-j3-dvb.dts

```
&spi0 {
    status = "okay";
    spidev@0x00 {
    compatible = "rohm,dh2228fv";
    spi-max-frequency = <20000000>;
    reg = <0>;
    };
};
&spi2 {
    status = "okay";
    isslave = <1>;
    slave@0x00 {
    compatible = "rohm,dh2228fv";
    spi-max-frequency = <20000000>;
    reg = <0>;
    };
};
```

Take spi0 and spi2 configuration as an example:

- hobot-xj3.dtsi contains public nodes should generally left unchanged. Modify the corresponding board level DTS file to suit the need of product;
- Configure spi0 as spi master and spi2 as spi slave. Use " isslave = <1>" to indicate the current (spi2) device is slave;
- The spidev@0x00, slave@0x00 node will be export to the sysfs as /dev/spidev0.0 and /dev/spidev2.0 for userspace operations;

14.2 SPI Device Driver

Path: drivers/spi/spi-hobot.c

14.2.1 SPI master/slave Configuration

```
static int hb_spi_probe(struct platform_device *pdev)
{
    ...
```

```
    /* master or slave mode select */
    isslave = of_property_read_bool(pdev->dev.of_node, "slave");
    if (isslave == MASTER_MODE) {
         ctlr = spi_alloc_master(&pdev->dev, sizeof(*hbspi));
         if (!ctlr) {
              dev_err(&pdev->dev, "failed to alloc spi master\n");
              return -ENOMEM;
         }
    } else if (isslave == SLAVE_MODE) {
         ctlr = spi_alloc_slave(&pdev->dev, sizeof(*hbspi));
         if (!ctlr) {
              dev_err(&pdev->dev, "failed to alloc spi slave, try master\n");
              return -ENOMEM;
         }
    }
    ...
}
```

14.2.2 SPI Registration

Register the SPI controller to the Linux Kernel:

```
static int hb_spi_probe(struct platform_device *pdev)
{
    ...

    if (isslave == MASTER_MODE) {
        hbspi->isslave = MASTER_MODE;
        snprintf(ctrl_mode, sizeof(ctrl_mode), "%s", "master");
        ctlr->bus_num = pdev->id;
        // ctlr->num_chipselect = HB_SPI_MAX_CS;
        ctlr->mode_bits = SPI_CPOL | SPI_CPHA | SPI_LSB_FIRST | SPI_CS_HIGH | SPI_NO_CS;
        ctlr->setup = hb_spi_setup;
        ctlr->prepare_transfer_hardware = hb_spi_prepare_xfer_hardware;
        ctlr->transfer_one = hb_spi_transfer_one;
        ctlr->unprepare_transfer_hardware = hb_spi_unprepare_xfer_hardware;
        ctlr->set_cs = hb_spi_chipselect;
        ctlr->dev.of_node = pdev->dev.of_node;
    } else if (isslave == SLAVE_MODE) {
        hbspi->isslave = SLAVE_MODE;
        snprintf(ctrl_mode, sizeof(ctrl_mode), "%s", "slave");
        ctlr->mode_bits = SPI_CPOL | SPI_CPHA | SPI_LSB_FIRST;
        ctlr->setup = hb_spi_slave_setup;
        ctlr->prepare_message = hb_spi_slave_prepare_message;
        ctlr->transfer_one = hb_spi_slave_transfer_one;
```

```
        ctlr->slave_abort = hb_spi_slave_abort;
    }

    /* register spi controller */
    ret = devm_spi_register_controller(&pdev->dev, ctlr);
    if (ret) {
        dev_err(&pdev->dev, "failed to register %s controller(%d)\n",
            ctrl_mode, ret);
        goto clk_dis_mclk;
    }

    ...
}
```

14.2.3 Hardware Initialization

Hardware initialization function is listed below, please contact Horizon for detailed register definition.

```
/* spi hw init */
static void hb_spi_init_hw(struct hb_spi *hbspi)
{
    u32 val = 0;

    /* First, should reset the whole controller */
    hb_spi_reset(hbspi);

    hb_spi_en_ctrl(hbspi, HB_SPI_OP_CORE_DIS, HB_SPI_OP_NONE,
              HB_SPI_OP_NONE);
    hb_spi_wr(hbspi, HB_SPI_INTSETMASK_REG, HB_SPI_INT_ALL);
    /* clear all interrupt pending */
    hb_spi_wr(hbspi, HB_SPI_SRCPND_REG, HB_SPI_INT_ALL);
    /* init rfto */
    hb_spi_wr(hbspi, HB_SPI_RFTO_REG, 0x27F);
    /* no instruction */
    hb_spi_wr(hbspi, HB_SPI_INST_REG, 0x0);
    hb_spi_wr(hbspi, HB_SPI_INST_MASK_REG, 0xFFFFFFFF);
    /* spi master mode */
    val = hb_spi_rd(hbspi, HB_SPI_CTRL_REG);
    if (hbspi->isslave == SLAVE_MODE)
        val |= HB_SPI_SLAVE_MODE;
    else
        val &= (~HB_SPI_SLAVE_MODE);
    if (hbspi->isslave == MASTER_MODE)
        val &= (~HB_SPI_SAMP_SEL);
```

```
    hb_spi_wr(hbspi, HB_SPI_CTRL_REG, val);

    if (debug)
        dev_err(hbspi->dev, "%s CTRL=%08X\n",
            __func__, hb_spi_rd(hbspi, HB_SPI_CTRL_REG));

    hb_spi_config(hbspi);
    hb_spi_en_ctrl(hbspi, HB_SPI_OP_CORE_EN, 0, 0);
}
```

14.2.4 Debug Parameters

Listed below is the exported SPI driver debug parameters:

```
static int debug;
static int slave_tout = 2000;
static int master_tout = 1000;
module_param(debug, int, 0644);
MODULE_PARM_DESC(debug, "spi: 0 close debug, other open debug");
module_param(slave_tout, int, 0644);
MODULE_PARM_DESC(slave_tout, "spi: slave timeout(sec), default 10 s");
module_param(master_tout, int, 0644);
MODULE_PARM_DESC(master_tout, "spi: master timeout(sec), default 2 s");
```

- Debug level can be configured to 0, 1, 2. (Default: 0)
- Slave Timeout is default to 2s, maximum 100s.
- Master Timeout is default to 1s, maximum 10s.

Read from sysfs node for the debug parameters, write to the same nodes to change the debug parameters. All units are ms:

```
ls /sys/module/spi_hobot/parameters/
```

The following message should appear on console.

```
root@x3dvbj3-hynix2G-2666:~# ls /sys/module/spi_hobot/parameters/
debug       master_tout  slave_tout
```

Read current debug level, default is 0, which means no debug message

```
cat /sys/module/spi_hobot/parameters/debug
```

The following message should appear on console.

```
root@x3dvbj3-hynix2G-2666:~# cat /sys/module/spi_hobot/parameters/debug
0
```

Set debug level to 1, and read from it to confirm:

```
echo 1 > /sys/module/spi_hobot/parameters/debug
cat /sys/module/spi_hobot/parameters/debug
```

The following message should appear on console.

```
root@x3dvbj3-hynix2G-2666:~# echo 1 > /sys/module/spi_hobot/parameters/debug
root@x3dvbj3-hynix2G-2666:~# cat /sys/module/spi_hobot/parameters/debug
1
```

Read current master_tout value, i.e. master timeout, default is 2s:

```
cat /sys/module/spi_hobot/parameters/master_tout
```

The following message should appear on console.

```
root@x3dvbj3-hynix2G-2666:~# cat /sys/module/spi_hobot/parameters/master_tout
1000
```

Read current slave_tout value, i.e. slave timeout, default is 1s:

```
cat /sys/module/spi_hobot/parameters/slave_tout
```

The following message should appear on console.

```
root@x3dvbj3-hynix2G-2666:~# cat /sys/module/spi_hobot/parameters/slave_tout
2000
```

14.3 SPI Verification

14.3.1 SPI Verification Patch

Copy the code below into a file named "spi_test.patch" and use "git apply" to apply the patch file, compile and enter test:

```
diff --git a/arch/arm64/boot/dts/hobot/hobot-j3-dvb.dts b/arch/arm64/boot/dts/hobot/hobot-j3-
dvb.dts
index 298e5933f..88fc44a15 100644
--- a/arch/arm64/boot/dts/hobot/hobot-j3-dvb.dts
+++ b/arch/arm64/boot/dts/hobot/hobot-j3-dvb.dts
@@ -164,3 +164,22 @@
    pinctrl-1 = <&sensor1_gpio_func>;
    snrclk-idx = <1>;
 };
+
+&spi2 {                                    # configure spi2 as spi slave
+   status = "okay";
+   isslave = <1>;
+   slave@0x00 {
```

```
+       compatible = "rohm,dh2228fv";
+       spi-max-frequency = <20000000>;
+       reg = <0>;
+   };
+};
+
+&spi0 {                                     # configure spi0 as spi master
+   status = "okay";
+   spidev@0x00 {
+       compatible = "rohm,dh2228fv";
+       spi-max-frequency = <20000000>;
+       reg = <0>;
+   };
+};
diff --git a/arch/arm64/boot/dts/hobot/hobot-xj3-xvb.dtsi b/arch/arm64/boot/dts/hobot/hobot-xj3-
xvb.dtsi
index 2c356a42a..d03051a6d 100644
--- a/arch/arm64/boot/dts/hobot/hobot-xj3-xvb.dtsi
+++ b/arch/arm64/boot/dts/hobot/hobot-xj3-xvb.dtsi
@@ -175,10 +175,10 @@
 };

 &i2c4 {                    # due to spi0 multiplexes i2c4/i2c5 pins, i2c4/5 needs to be disabled
-   status = "okay";
+   status = "disable";
 };
 &i2c5 {
-   status = "okay";
+   status = "disable";
 };

 &pinctrl {
diff --git a/arch/arm64/configs/xj3_debug_defconfig b/arch/arm64/configs/xj3_debug_defconfig
index b1e0e568e..6fb50ec7c 100755
--- a/arch/arm64/configs/xj3_debug_defconfig
+++ b/arch/arm64/configs/xj3_debug_defconfig
@@ -1641,7 +1641,7 @@ CONFIG_SPI_HOBOT_QSPI=y
 CONFIG_SPI_SPIDEV=y
 # CONFIG_SPI_LOOPBACK_TEST is not set
 # CONFIG_SPI_TLE62X0 is not set
-# CONFIG_SPI_SLAVE is not set
+CONFIG_SPI_SLAVE=y                  # Enable CONFIG_SPI_SLAVE to create spidevX.X sysfs
nodes
 # CONFIG_SPMI is not set
```

```
# CONFIG_HSI is not set
CONFIG_PPS=y
```

14.3.2 Check spidev sysfs nodes

Check SPI master nodes

```
root@x3dvbj3-hynix2G-2666:~# ls /sys/class/spi_master/  # spi0 as master
spi0
root@x3dvbj3-hynix2G-2666:~#
```

Check SPI slave nodes

```
root@x3dvbj3-hynix2G-2666:~# ls /sys/class/spi_slave/   # spi2 as slave
spi2
root@x3dvbj3-hynix2G-2666:~#
```

Check SPI dev nodes:

Method 1:

```
root@x3dvbj3-hynix2G-2666:~# ls /dev/spidev*
/dev/spidev0.0  /dev/spidev2.0
root@x3dvbj3-hynix2G-2666:~#
```

Method 2:

```
root@x3dvbj3-hynix2G-2666:~# ls /sys/class/spidev/
spidev0.0  spidev2.0
root@x3dvbj3-hynix2G-2666:~#
```

14.3.3 Test Procedure

Use spidev_tc to loop test spi0 and spi2

slave_write --> master_read

Sample command is listed below:

```
/* slave write */
root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3# ./spidev_tc -D /dev/spidev2.0 -v -s 1000000 -m 2
-e 64 -t 1
spi mode: 0x0
bits per word: 8
max speed: 1000000 Hz (1000 KHz)
userspace spi write test, len=64 times=1
test, times=0
TX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
```

```
TX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87

root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3#

/* master read */
root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3# ./spidev_tc -D /dev/spidev0.0 -v -s 1000000 -m 1
-e 64 -t 1
spi mode: 0x0
bits per word: 8
max speed: 1000000 Hz (1000 KHz)
userspace spi read test, len=64 times=1
test, times=0
RX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
RX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87

root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3#
```

Slave sends 64 byts

```
TX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
TX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87
```

Master receives 64 bytes

```
RX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
RX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87
```

slave_read --> master_write

```
/* slave read */
root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3# ./spidev_tc -D /dev/spidev2.0 -v -s 1000000 -m 1
-e 64 -t 1
spi mode: 0x0
bits per word: 8
max speed: 1000000 Hz (1000 KHz)
userspace spi read test, len=64 times=1
test, times=0
RX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
```

```
RX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87

root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3#

/* master write */
root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3# ./spidev_tc -D /dev/spidev0.0 -v -s 1000000 -m 2
-e 64 -t 1
spi mode: 0x0
bits per word: 8
max speed: 1000000 Hz (1000 KHz)
userspace spi write test, len=64 times=1
test, times=0
TX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
TX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87

root@x3dvbj3-hynix2G-2666:/var/volatile/tmp/x3#
```

Slave receives 64 bytes:

```
RX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
RX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87
```

Master sends 64 bytes

```
TX | 67 C6 69 73 51 FF 4A EC 29 CD BA AB F2 FB E3 46 7C C2 54 F8 1B E8 E7 8D 76 5A 2E 63 33 9F
C9 9A
TX | 66 32 0D B7 31 58 A3 5A 25 5D 05 17 58 E9 5E D4 AB B2 CD C6 9B B4 54 11 0E 82 74 41 21 3D
DC 87
```

- Note 1: Test case code is attached in Appendix
- Note2: Execute slave program first, then execute master program(See SPI Timing)

14.4 Appendix

14.4.1 Appendix A: spidev_tc.c Source code:

```
/*
 * SPI testing utility (using spidev driver)
 *
 * Copyright (c) 2007  MontaVista Software, Inc.
```

```
 * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * Cross-compile with cross-gcc -I/path/to/cross-kernel/include
 */

#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <fcntl.h>
#include <time.h>
#include <sys/ioctl.h>
#include <linux/ioctl.h>
#include <sys/stat.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

static void pabort(const char *s)
{
    perror(s);
    abort();
}

static const char *device = "/dev/spidev1.1";
static uint32_t mode;
static uint8_t bits = 8;
static char *input_file;
static char *output_file;
static uint32_t speed = 500000;
static uint16_t delay;
static int verbose;
static int transfer_size;
static int iterations;
static int interval = 5; /* interval in seconds for showing transfer rate */
static int rw_mode = 0; //1: read, 2: write, 3: write and read
static int rw_len = 4;
```

```
static int rw_times = 5;

uint8_t default_tx[] = {
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xF0, 0x0D,
};

uint8_t default_rx[ARRAY_SIZE(default_tx)] = {0, };
char *input_tx;

static void hex_dump(const void *src, size_t length, size_t line_size,
            char *prefix)
{
    int i = 0;
    const unsigned char *address = src;
    const unsigned char *line = address;
    unsigned char c;

    printf("%s | ", prefix);
    while (length-- > 0) {
        printf("%02X ", *address++);
        if (!(++i % line_size) || (length == 0 && i % line_size)) {
            if (length == 0) {
                while (i++ % line_size)
                    printf("__ ");
            }
            printf(" | ");  /* right close */
            while (line < address) {
                c = *line++;
                printf("%c", (c < 33 || c == 255) ? 0x2E : c);
            }
            printf("\n");
            if (length > 0)
                printf("%s | ", prefix);
        }
    }
}

static void hex_dump2(const void *src, size_t length, size_t line_size,
         char *prefix)
```

```
{
    int i = 0;
    const unsigned char *address = src;
    const unsigned char *line = address;
    unsigned char c;

    printf("%s | ", prefix);
    while (length-- > 0) {
        printf("%02X ", *address++);
        if (!(++i % line_size) || (length == 0 && i % line_size)) {
            if (length == 0) {
                while (i++ % line_size)
                    printf("__ ");
            }
            printf("\n");
            if (length > 0)
                printf("%s | ", prefix);
        }
    }
    printf("\n");
}

/*
 *  Unescape - process hexadecimal escape character
 *      converts shell input "\x23" -> 0x23
 */
static int unescape(char *_dst, char *_src, size_t len)
{
    int ret = 0;
    int match;
    char *src = _src;
    char *dst = _dst;
    unsigned int ch;

    while (*src) {
        if (*src == '\\' && *(src+1) == 'x') {
            match = sscanf(src + 2, "%2x", &ch);
            if (!match)
                pabort("malformed input string");

            src += 4;
            *dst++ = (unsigned char)ch;
        } else {
            *dst++ = *src++;
```

```
        }
        ret++;
    }
    return ret;
}

static void transfer(int fd, uint8_t const *tx, uint8_t const *rx, size_t len)
{
    int ret;
    int out_fd;
    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = len,
        .delay_usecs = delay,
        .speed_hz = speed,
        .bits_per_word = bits,
    };

    if (mode & SPI_TX_QUAD)
        tr.tx_nbits = 4;
    else if (mode & SPI_TX_DUAL)
        tr.tx_nbits = 2;
    if (mode & SPI_RX_QUAD)
        tr.rx_nbits = 4;
    else if (mode & SPI_RX_DUAL)
        tr.rx_nbits = 2;
    if (!(mode & SPI_LOOP)) {
        if (mode & (SPI_TX_QUAD | SPI_TX_DUAL))
            tr.rx_buf = 0;
        else if (mode & (SPI_RX_QUAD | SPI_RX_DUAL))
            tr.tx_buf = 0;
    }

    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
    if (ret < 1)
        pabort("can't send spi message");

    if (verbose)
        hex_dump(tx, len, 32, "TX");

    if (output_file) {
        out_fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
        if (out_fd < 0)
```

```
            pabort("could not open output file");

        ret = write(out_fd, rx, len);
        if (ret != len)
            pabort("not all bytes written to output file");

        close(out_fd);
    }

    if (verbose)
        hex_dump(rx, len, 32, "RX");
}

static void transfer2(int fd, uint8_t const *tx, uint8_t const *rx, size_t len)
{
    int ret;
    int out_fd;
    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = len,
        .delay_usecs = delay,
        .speed_hz = speed,
        .bits_per_word = bits,
    };

    if (mode & SPI_TX_QUAD)
        tr.tx_nbits = 4;
    else if (mode & SPI_TX_DUAL)
        tr.tx_nbits = 2;
    if (mode & SPI_RX_QUAD)
        tr.rx_nbits = 4;
    else if (mode & SPI_RX_DUAL)
        tr.rx_nbits = 2;
    if (!(mode & SPI_LOOP)) {
        if (mode & (SPI_TX_QUAD | SPI_TX_DUAL))
            tr.rx_buf = 0;
        else if (mode & (SPI_RX_QUAD | SPI_RX_DUAL))
            tr.tx_buf = 0;
    }

    if (verbose && rw_mode >> 1)
        hex_dump2(tx, len, 32, "TX");
```

```
    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
    if (ret < 1) {
        //pabort("can't send spi message");
        printf("can't send spi message");
    } else {
        if (output_file) {
            out_fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
            if (out_fd < 0)
                pabort("could not open output file");

            ret = write(out_fd, rx, len);
            if (ret != len)
                pabort("not all bytes written to output file");

            close(out_fd);
        }

        if (verbose && rw_mode&0x01)
            hex_dump2(rx, len, 32, "RX");
    }

}

static void print_usage(const char *prog)
{
    printf("Usage: %s [-DsbdlHOLC3vpNR24SImet]\n", prog);
    puts("  -D --device   device to use (default /dev/spidev1.1)\n"
         "  -s --speed    max speed (Hz)\n"
         "  -d --delay    delay (usec)\n"
         "  -b --bpw      bits per word\n"
         "  -i --input    input data from a file (e.g. \"test.bin\")\n"
         "  -o --output   output data to a file (e.g. \"results.bin\")\n"
         "  -l --loop     loopback\n"
         "  -H --cpha     clock phase\n"
         "  -O --cpol     clock polarity\n"
         "  -L --lsb      least significant bit first\n"
         "  -C --cs-high  chip select active high\n"
         "  -3 --3wire    SI/SO signals shared\n"
         "  -v --verbose  Verbose (show tx buffer)\n"
         "  -p            Send data (e.g. \"1234\\xde\\xad\")\n"
         "  -N --no-cs    no chip select\n"
         "  -R --ready    slave pulls low to pause\n"
         "  -2 --dual     dual transfer\n"
         "  -4 --quad     quad transfer\n"
```

```
        "  -S --size     transfer size\n"
        "  -I --iter     iterations\n"
        "  -m --rw-mode  1 read, 2 write, 3 write and read\n"
        "  -e --rw-len   read or write len\n"
        "  -t --rw-times read or write times\n");
    exit(1);
}

static void parse_opts(int argc, char *argv[])
{
    while (1) {
        static const struct option lopts[] = {
            { "device",  1, 0, 'D' },
            { "speed",   1, 0, 's' },
            { "delay",   1, 0, 'd' },
            { "bpw",     1, 0, 'b' },
            { "input",   1, 0, 'i' },
            { "output",  1, 0, 'o' },
            { "loop",    0, 0, 'l' },
            { "cpha",    0, 0, 'H' },
            { "cpol",    0, 0, 'O' },
            { "lsb",     0, 0, 'L' },
            { "cs-high", 0, 0, 'C' },
            { "3wire",   0, 0, '3' },
            { "no-cs",   0, 0, 'N' },
            { "ready",   0, 0, 'R' },
            { "dual",    0, 0, '2' },
            { "verbose", 0, 0, 'v' },
            { "quad",    0, 0, '4' },
            { "size",    1, 0, 'S' },
            { "iter",    1, 0, 'I' },
            { "rw-mode",   1, 0, 'm' },
            { "rw-len",    1, 0, 'e' },
            { "rw-times",    1, 0, 't' },
            { NULL, 0, 0, 0 },
        };
        int c;

        c = getopt_long(argc, argv, "D:s:d:b:i:o:lHOLC3NR24p:vS:I:m:e:t:",
                lopts, NULL);
        //printf("optind: %d\n", optind);
        //printf("optarg: %s\n", optarg);
        //printf("option: %c\n", c);
```

```
    if (c == -1)
        break;

    switch (c) {
    case 'D':
        device = optarg;
        break;
    case 's':
        speed = atoi(optarg);
        break;
    case 'd':
        delay = atoi(optarg);
        break;
    case 'b':
        bits = atoi(optarg);
        break;
    case 'i':
        input_file = optarg;
        break;
    case 'o':
        output_file = optarg;
        break;
    case 'l':
        mode |= SPI_LOOP;
        break;
    case 'H':
        mode |= SPI_CPHA;
        break;
    case 'O':
        mode |= SPI_CPOL;
        break;
    case 'L':
        mode |= SPI_LSB_FIRST;
        break;
    case 'C':
        mode |= SPI_CS_HIGH;
        break;
    case '3':
        mode |= SPI_3WIRE;
        break;
    case 'N':
        mode |= SPI_NO_CS;
        break;
    case 'v':
```

```
            verbose = 1;
            break;
        case 'R':
            mode |= SPI_READY;
            break;
        case 'p':
            input_tx = optarg;
            break;
        case '2':
            mode |= SPI_TX_DUAL;
            break;
        case '4':
            mode |= SPI_TX_QUAD;
            break;
        case 'S':
            transfer_size = atoi(optarg);
            break;
        case 'I':
            iterations = atoi(optarg);
            break;
        case 'm':
            rw_mode = atoi(optarg);
            break;
        case 'e':
            rw_len = atoi(optarg);
            break;
        case 't':
            rw_times = atoi(optarg);
            break;
        default:
            print_usage(argv[0]);
            break;
        }
    }
    if (mode & SPI_LOOP) {
        if (mode & SPI_TX_DUAL)
            mode |= SPI_RX_DUAL;
        if (mode & SPI_TX_QUAD)
            mode |= SPI_RX_QUAD;
    }
}

static void transfer_escaped_string(int fd, char *str)
{
```

```
    size_t size = strlen(str);
    uint8_t *tx;
    uint8_t *rx;

    tx = malloc(size);
    if (!tx)
        pabort("can't allocate tx buffer");

    rx = malloc(size);
    if (!rx)
        pabort("can't allocate rx buffer");

    size = unescape((char *)tx, str, size);
    transfer(fd, tx, rx, size);
    free(rx);
    free(tx);
}

static void transfer_file(int fd, char *filename)
{
    ssize_t bytes;
    struct stat sb;
    int tx_fd;
    uint8_t *tx;
    uint8_t *rx;

    if (stat(filename, &sb) == -1)
        pabort("can't stat input file");

    tx_fd = open(filename, O_RDONLY);
    if (tx_fd < 0)
        pabort("can't open input file");

    tx = malloc(sb.st_size);
    if (!tx)
        pabort("can't allocate tx buffer");

    rx = malloc(sb.st_size);
    if (!rx)
        pabort("can't allocate rx buffer");

    bytes = read(tx_fd, tx, sb.st_size);
    if (bytes != sb.st_size)
        pabort("failed to read input file");
```

```
    transfer(fd, tx, rx, sb.st_size);
    free(rx);
    free(tx);
    close(tx_fd);
}

static uint64_t _read_count;
static uint64_t _write_count;

static void show_transfer_rate(void)
{
    static uint64_t prev_read_count, prev_write_count;
    double rx_rate, tx_rate;

    rx_rate = ((_read_count - prev_read_count) * 8) / (interval*1000.0);
    tx_rate = ((_write_count - prev_write_count) * 8) / (interval*1000.0);

    printf("rate: tx %.1fkbps, rx %.1fkbps\n", rx_rate, tx_rate);

    prev_read_count = _read_count;
    prev_write_count = _write_count;
}

static void transfer_buf(int fd, int len)
{
    uint8_t *tx;
    uint8_t *rx;
    int i;

    tx = malloc(len);
    if (!tx)
        pabort("can't allocate tx buffer");
    for (i = 0; i < len; i++)
        tx[i] = random();

    rx = malloc(len);
    if (!rx)
        pabort("can't allocate rx buffer");

    transfer(fd, tx, rx, len);

    _write_count += len;
    _read_count += len;
```

```
    if (mode & SPI_LOOP) {
        if (memcmp(tx, rx, len)) {
            fprintf(stderr, "transfer error !\n");
            hex_dump(tx, len, 32, "TX");
            hex_dump(rx, len, 32, "RX");
            exit(1);
        }
    }

    free(rx);
    free(tx);
}

static void transfer_read_write(int fd)
{
    uint8_t *tx;
    uint8_t *rx;
    int i, j;
    int len, times;
    char str[64] = {0};

    len = rw_len > 0 ? rw_len : 4;
    times = rw_times > 0 ? rw_times : 4;
    if (rw_mode == 2)
        sprintf(str, "write");
    else if (rw_mode == 3)
        sprintf(str, "read and write");
    else {
        rw_mode = 1;
        sprintf(str, "read");
    }

    printf("userspace spi %s test, len=%d times=%d\n", str, len, times);

    tx = malloc(len + 4);
    if (!tx)
        pabort("can't allocate tx buffer");
    rx = malloc(len + 4);
    if (!rx)
        pabort("can't allocate rx buffer");

    for (j = 0; j < rw_times; j++) {
        memset(tx, 0 ,len);
```

```
        memset(rx, 0, len);

        if (rw_mode >> 1) {
            for (i = 0; i < len; i++)
                tx[i] = random();
        } else {
            for (i = 0; i < len; i++)
                tx[i] = i;
        }
        printf("test, times=%d\n", j);
        transfer2(fd, tx, rx, len);
        //sleep(2);
    }
}

int main(int argc, char *argv[])
{
    int ret = 0;
    int fd;

    parse_opts(argc, argv);

    fd = open(device, O_RDWR);
    if (fd < 0)
        pabort("can't open device");

    /*
     * spi mode
     */
    ret = ioctl(fd, SPI_IOC_WR_MODE32, &mode);
    if (ret == -1)
        pabort("can't set spi mode");

    ret = ioctl(fd, SPI_IOC_RD_MODE32, &mode);
    if (ret == -1)
        pabort("can't get spi mode");

    /*
     * bits per word
     */
    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
    if (ret == -1)
        pabort("can't set bits per word");
```

```
ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
if (ret == -1)
    pabort("can't get bits per word");

/*
 * max speed hz
 */
ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
    pabort("can't set max speed hz");

ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
if (ret == -1)
    pabort("can't get max speed hz");

printf("spi mode: 0x%x\n", mode);
printf("bits per word: %d\n", bits);
printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);

if (input_tx && input_file)
    pabort("only one of -p and --input may be selected");

if (input_tx)
    transfer_escaped_string(fd, input_tx);
else if (input_file)
    transfer_file(fd, input_file);
else if (transfer_size) {
    struct timespec last_stat;

    clock_gettime(CLOCK_MONOTONIC, &last_stat);

    while (iterations-- > 0) {
        struct timespec current;

        transfer_buf(fd, transfer_size);

        clock_gettime(CLOCK_MONOTONIC, ¤t);
        if (current.tv_sec - last_stat.tv_sec > interval) {
            show_transfer_rate();
            last_stat = current;
        }
    }
    printf("total: tx %.1fKB, rx %.1fKB\n",
           _write_count/1024.0, _read_count/1024.0);
```

```
    } else if (rw_mode) {
        transfer_read_write(fd);
    } else
        transfer(fd, default_tx, default_rx, sizeof(default_tx));

    close(fd);

    return ret;
```

14.4.2 Appendix B: run_master.sh Test Script

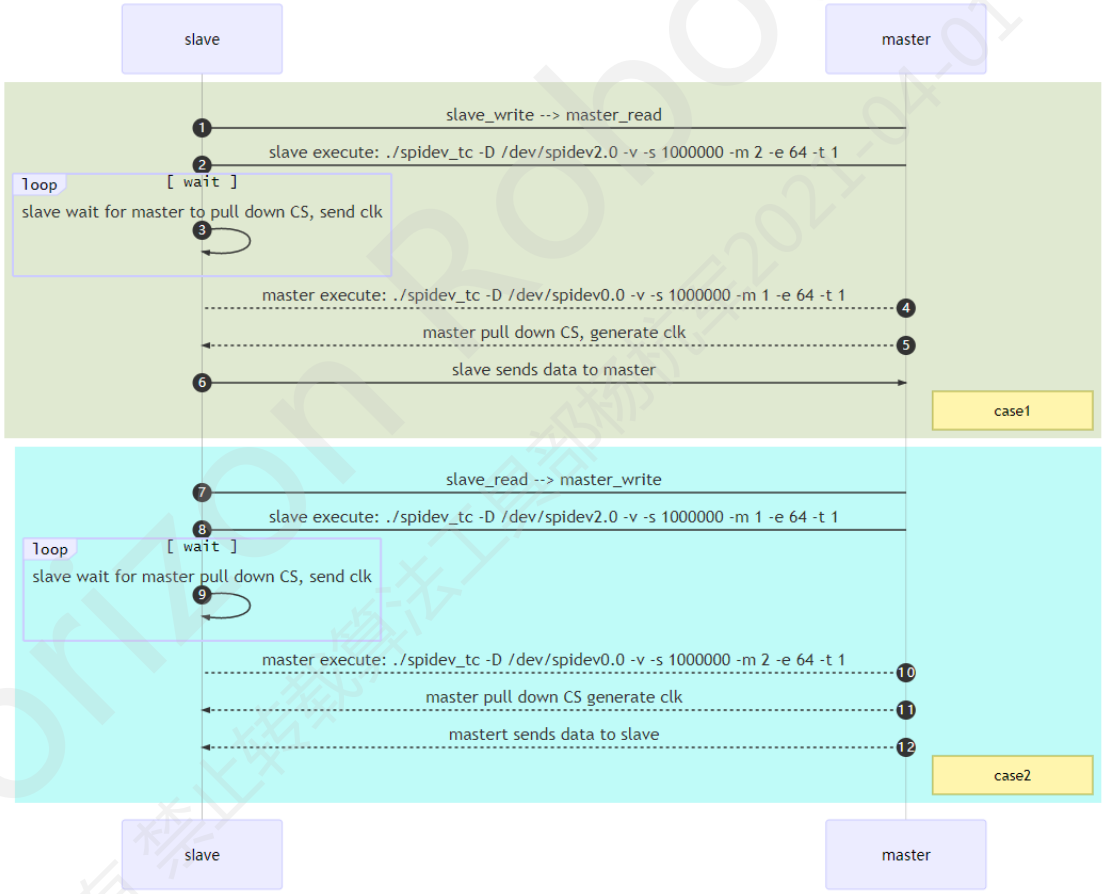
```
#!/bin/sh

i=0
t=10000
size=4096
while [ $i -le $t ]
do
    ./spidev_tc -D /dev/spidev2.0 -v -s 27000000 -m 2 -e $size -t 1
    let i=$i+1
    echo "i = "$i
done

i=0
while [ $i -le $t ]
do
    ./spidev_tc -D /dev/spidev2.0 -v -s 27000000 -m 1 -e $size -t 1
    let i=$i+1
    echo "i = "$i
done
```

14.4.3 Appendix C: run_slave.sh Test script

```
#!/bin/sh

i=0
t=10000
size=4096
while [ $i -le $t ]
do
    ./spidev_tc -D /dev/spidev0.0 -v -s 27000000 -m 1 -e $size -t 1
    let i=$i+1
    echo "i = "$i
    sleep 1
```

```
done

i=0
while [ $i -le $t ]
do
    ./spidev_tc -D /dev/spidev0.0 -v -s 27000000 -m 2 -e $size -t 1
    let i=$i+1
    echo "i = "$i
    sleep 1
done
```

14.4.4 Appendix D: SPI Timing



15 Camera Sensor Debug Manual

15.1 MIPI Protocol

This manual aims at analyzing some of the frequently encountered problems, and provides some insight combining MIPI configuration and MIP registers/functions. This manual strives to provide some starting point for debugging MIPI module, demonstrates the influence of user space MIPI configuration over MIPI lane. The last part of this chapter will introduce the basic method for creating new sensor configuration for the current board.

15.1.1 Abbreviates

- PPI: PHY-Protocol Interface
- IPI: Image Pixel Interface
- HS: High-Speed
- LP: Low-Power
- RX: Receiver
- TX: Transmitter
- SoT: Start of Transmission
- SoF: Start of Frame
- Vsync: Vertical Synchronism
- Hsync: Horizontal Synchronism
- Mbps: Megabits per second
- FPS: Frames Per Second
- HSA: Hsync Active time
- HSD: Hsync Delay time
- HBP: Horizontal Back Porch time
- CLK: clock
- ULPM: Ultra Low Power Mode
- VSA: Vsync Active Time
- VBP: Virtical Back Porch
- VFP: Virtical Front Porch
- LP-11: Dp-Line LP High / Dn-line LP High

15.1.2 Parameters

15.1.2.1 Universal Parameters

- lane: Physical Wire pairs used in MIPI transfer
- datatype: The type of data
- mclk: The reference clk passed in from Camera, reserved for ASIC camera driver
- mipiclk: Total Bit Rate in MIPI transfer, Unit: Mbps
- fps: Camera transfer frame per second
- width: Camera output actual valid width, Unit: Pixels
- height: Camera output actual valid height, Unit: Lines
- linelenth: Camera output line length, inclcuding valid width and blanking zone width, Unit: Pixels
- framelenth: Camera output line number, including valid height and blanking zone lines, Unit: Lines
- settle: The delay between LP to HS during Camera output

15.1.2.2 MIPI Host Unique Parameters

- enable: if MIPI Host module is enabled
- hsa: Time Hsync (IPI generated) last
- hbp: Time between Hsync generated (byIPI) until data_en is enabled
- hsd: The delay between IPI receiving valid data and Hsync generated (Ready for data collection)
- vc_num: Total number of virtual channel of Camera input, maximum: 2
- vc0_index: The IPI index of virtual channel 0 of Camera input
- vc1_index: The IPI index of virtual channel 1 of Camera input
- vc2_index: The IPI index of virtual channel 2 of Camera input
- vc3_index: The IPI index of virtual channel 3 of Camera input

15.1.2.3 MIPI Device Unique Parameters

- enable: whether enable MIPI device module
- format: MIPI Device output CSI2 format. Used when device output alone, e.g. IAR/VPG

- vpg: VPG mode enable
- ipi_lines: MIPI Device detect the total lines of IPI input, used primarily with IAR

15.1.3 Parameter Configuration

This section is a detailed description of how to configure the parameters listed in *15.1.2* This section also focuses on several key parameters in MIPI driver configuration. The "Databook" referred in the following contents is the MIPI Protocol Databook, please refer to the MIPI alliance official site for details. All source files referred to in this chapter is listed under: drivers/media/platform/hobot/mipi/.

15.1.3.1 MIPI Host Configuration

Among MIPI Host configurations, those relate to parameter are: D-PHY clock, IPI clock, IPI and HSD.

a) D-PHY clock Configuration

Key Function: hobot_mipi_dphy.c: mipi_dphy_clk_range

Parameters Involved: phy, mipiclk, osc_freq

Configuration Details:

1. Based on mipiclk/lane, calculate the single lane bit rate (laneclk)
2. Based on laneclk, refer to data book, select the proper HS clock range
3. Write range value to VIO registers

b) IPI clock Configuration

Key Function: hobot_mipi_host.c: mipi_host_pixel_clk_select

Parameters Involved: cfg(width, height, fps, linelenth, framelenth)

Configuration Details:

1. Calculate total pixel clock: pixel clock = linelenth*framelenth*fps

   Note, new formula is used:

```
pixclk = (width + 128) * (height + 96) * fps
```

Note: Formula for calculating pixclk is changed to avoid HSD overload (exceeding 4095) when Camera Blanking is too large. Theoretically, as long as pixel clk is greater than actual transfer speed, requirement would be met. However, this requires high FPS measurement accuracy. Users should adjust pixclk calculation according to their own need.

2. Based on Databook, 48-bit IPI transfer raw at a rate of 3 pixel/clock; transfer yuv at

a rate of 1 pixel/clock.

```
ipiclk = pixel clock (YUV), ipiclk = pixel clock / 3 (RAW)
```

3. Based on ipiclk, referring to current clock tree, and round up the clock to actual clock rate.

c) HSD Configuration

Key Function: hobot_mipi_host.c: mipi_host_get_hsd

Parameters Involved: cfg(width, fps, linelenth, framelenth, format,HSA, HBP), pixclk

Configuration Details:

1. In user space, use format, pix_len to convert to CSI2 data type
2. Based on data type, calculate bits_per_piexel and cycles_to_trans
3. Calculate rx_bit_clk:

```
rx_bit_clk = linelenth*framelenth*fps*bits_per_pixel, line_size = width
```

4. HSD > (bits_per_pixel * line_size * pixclk / rx_bit_clk)-(hsa + hbp + cycles_to_trans)

d) IPI Configuration

Key Function: hobot_mipi_host.c: mipi_host_configure_ipi

Parameters Involved: format, pix_len , vc_num, vc0_index , vc1_index, HSA, HBP, HSD

Configuration Details:

1. In user space, use format, pix_len to convert to CSI2 data type, write to DATA_TYPE register
2. If vc_num==2, add new config IPI2
3. Write vc0_index, vc1_index to IPI_VCID and IPI2_VCID register. Assign IPI1 to collect the data from virtual channel id = vc0_index; Assign IPI2 to collect the data from virtual channel id = vc1_index
4. Default: HSA=4, HBP=4. If json configures otherwise, use value from json. HSD is calculated as described in *HSD Configuration*, used json config if configured.

e) Advanced Features

1. Default off, controller will automatically detect the right time for generating Vsync, Hsync signal
2. Different camera require its own line event selection, used for control controller generated Vsync, Hsync timing. For example: if video data comes long after Vsync/Hsync signal, might need to enable en_video; Camera confirm Line Start/End short package sent, might enable en_line_start etc.

15.1.3.2 MIPI Device Configuration

Among MIPI Device configurations, those relate to parameters are: D-PHY clock, PLL, IPI, VPG mode. Note: The IPI clock of MIPI Device comes from MIPI Host which needs not configure separately.

a) D-PHY clock Configuration

Key function: hobot_mipi_dphy.c: mipi_dphy_clk_range

Parameters Involved: phy, mipiclk, osc_freq

Configuration Details:

1. Based on mipiclk/lane, calculate single lane bit rate(laneclk)
2. Based on laneclk, refer to Databook, select proper HS clock range
3. Write range value to VIO register

b) PLL Configuration

Key function: hobot_mipi_dphy.c: mipi_tx_pll_div

Parameters involved: lane, mipiclk

Configuration Details:

1. Based on mipiclk/lane, calculate single lane bit rate(laneclk)
2. Calculate output frequency :

```
fout = PLL Fout(GHz) = data rate(Gbps) / 2
```

3. Based on fout, refer to Databook, calculate proper frequency division factor between fvco and fout
4. Calculate fvco:

```
fvco = fout << vco_div
```

5. Based on TX default reference clock TX_REFSCLK_DEFAULT=24M, calculate proper frequency divider(n) and frequency multiplier(m)

Note: To improve arithmetic accuracy, currently n is fixed at 12, the output of frequency divider would be integer 2MHz

6. Due to arithmetic accuracy of non-floating point calculation, recalculate fout:

```
fout = fvco >> vco_div = ((refsclk * (*m + 2)) / (*n + 1)) >> vco_div
```

7. Refer to Databook, check vco range and select proper vco range and the uppoer limit of current range.

Note: Combine actual usage and formula, VCO range from Databook should be fout

range

8. Based on the fvco_max from step 7, recalculate the frequency multiplier(m) of current frequency

Note: the recalculated m aims at maximizing final output frequency within the limit of vco range. Maximizing output frequency can avoid device overflow for some camera using normal m value derived from normal fvco.

c) IPI Configuration

Key function: hobot_mipi_dev.c: mipi_dev_configure_ipi

Parameters Involved: cfg(format, pix_len, width, height/ipi_lines)

Configuration Details:

1. In user space, use format, pix_len to convert to CSI2 data type. Write to PKT_CFG register
2. PKT_CFG can control if Device should send line sync and scan every other line

Note: if Arbitrary Value Mode is configured, START_LINE_NUM and STEP_LINE_NUM register should also be configured.

3. Write width to IPI_PIXELS register, write height + 1(default) or ipi_lines to IPI_LINES register

Note: To meet MIPI Device clock requirement, SIF would add an extra Hsync after each frame when bypassing IPI data. Otherwise, MIPI Device would prompt underflow error. (Refer to Device Controller data boot Figure 2-24, Device IPI need an extra Hsync after the last data_en signal to collect the last line data, or Frame End packet would be generated prematurely.) Meanwhile, IPI_LINES need to be configured as height + 1 (Accommodate extra Hsync at frame end by SIF), otherwise error lines would occur

Additionally, if legacy mode is configured in advanced feature, Host would generate an extra Hsync along with Vsync. Thus, in this case, IPI_LINES need to be configured as height+2 (1 Hsync at frame start from Host; 1 Hsync at frame end from SIF)

d) VPG Configuration

Key function: hobot_mipi_dev.c: mipi_dev_configure_vpg

Parameters Involved: cfg(format, pix_len, width, height, linelenth, framelenth)

Configuration details:

1. In userspace, use format, pix_len to convert to CSI2 data type, write to PKT_CFG register
2. Write width to PKT_SIZE register; write height to ACT_LINES register
3. Under VPG mode, HSA/HBP/HLINE/VSA/VBP/VFP need to be configured manually. Currently HSA/HBP/VSA/VBP are fixed.
4. Calculate HLINE, based on data type and linelenth to calculate single line time (in

lane byte clk)

5. Calculate VFP based on framelength
6. Currently VPG mode is configured as Vertical Color Bar, adjust according to your own need

15.1.4 Debug Tips

This section aims at revealing some of the critical point that is often not properly configured. Primarily focusing on Host/Camera, Device/AP Rx flow, and potential bug introduced by not properly configured parameters.

15.1.4.1 Configuration Process

a) RX Configuration Process

The configuration process of MIPI Host consists of 3 major step:

1. Initialization: Start up -> Initialize -> Configure IPI
   - Host would check online Stop status
   - Camera should be in LP11 status, i.e. Camera is in Stop status/Stream off status
2. Start: Start HS Reception -> Detect Errors
   - Host would check online HS status, register system interrupt, if any interrupts received, check interrupt status register
   - Camera should be in HS mode, i.e. Camera start sending/in Stream On status
3. Stop: Stop HS Reception -> Reset Controller/PHY
   - Camera should be in LP11 status, i.e. Camera is in Stop status/Stream Off status. Power off directly is acceptable

b) TX Configuration Process

The configuration process of MIPI Device consists of 3 major steps:

1. Initialization: Initialize PHY -> Initialize Controller -> Wake Up
   - Device would check PLL lock status and make sure online entering LP11 status, i.e. in stop status, waiting IPI data input.
   - AP side RX should have finished initialization, waiting for data input
2. Start:
   - When IPI data arrived at Device, Device would enter HS mode automatically and start output data
3. End: Reset Controller/PHY

15.1.4.2 Process Errors

a) RX Process Error

1. Camera start sending before MIPI Host initialization complete
   - Result: MIPI Host receive Camera data during initialization or in the unstable state just after initialization. This would cause PHY or IPI to encounter irrecoverable error
   - Advice: Configure Camera/Host comply the process as detailed in *15.1.4.1a)* do not skip any status check to prevent unknown errors
2. When MIPI Host start working, Camera Keeps rebooting
   - Result: While Camera reboot/power on/off, there is no guarantee the integrity and completeness of tx data of camera, might affect MIPI Host working status.
   - Advice: When starting and stopping camera, comply the process as detailed in *15.1.4.1* to do the same starting/stopping o MIPI Host

b) TX Process Error

1. Before AP side RX ready, Camera->Host->SIF->Device starts transmitting prematurely
2. Result: AP RX encounter the same error as MIPI Host as detailed in *15.1.4.2a)*.

Also note: After measurement, if AP side RX not ready or AP not connected, MIPI Device output signal would malfunction. E.g. no signal on clock lane, data lane signal abnormal.

15.1.4.3 Parameter Errors

a) Clock Configuration Error

Parameters Involved: mipiclk, lane, settle

1. lane speed = mipiclk/lane, this parameter is used to configure Host/Device PHY basic configuration. Affects the HS clock range of Host/Device PHY and the PLL calculation of Device PHY. Meanwhile, lane number would be written to Host/Device PHY registers. Please make sure lane speed and lane number are correctly configured
2. Settle, normally automatically matched by PHY. But in some extreme circumstances, PHY cannot match settle automatically, thus configured settle in driver. Typically, camera supplier need to supply value for settle measured in PPI clock. If not provided, trial until pass. (The range of the value of settle is small, the robustness of it is quite good)

Error: if the above parameters are not configured properly, would result in PHY error while receive/resolve CSI2 data, issuing various PHY/PKT/FRAME Fatal error interrupts.

b) Frame rate Configuration Error

Parameters Involved: width, height, linelenth, framelenth, fps

1. For MIPI Host, these parameters affect IPI Clock and HSD calculation, if not configured properly, would result in IPI overflow/underflow error. Ultimately undermine the correctness of data transmitted to SIF;
2. For MIPI Device, width/height are written directly to IPI registers, affect the size of final output. Linelenth and frame length are sued in VPG mode, simulating normal camera output blanking.

15.1.5 Error Analysis

This section focuses on frequently encountered debug error LOG providing a detailed explanation and analysis. Also summarizes some of the solution for your information.

15.1.5.1 MIPI Host Error

a) INT_ST_PHY_FATAL

Error Description: Detect irrecoverable error on datalane occurs: ErrSotSyncHS, SOT is required packet for LP-HS, affects data receiving

Error Analysis:

1. Check PHY Clock and Settle configuration, please refer to *15.1.4.3a)*
2. If configured properly, refer to *15.1.4.1a)* for proper Camera/Host initialization process
3. If the previous checks passed, use oscilloscope to measure camera output data and signal quality

b) INT_ST_PKT_FATAL

Error Description: Packet related error during data transmission

1. Packet ECC error: ErrEccDouble
2. Packet CRC error: ErrCrc

Error Analysis:

1. Check PHY Clock and Settle configuration, please refer to *15.1.4.3a)*
2. If configured properly, refer to *15.1.4.1a)* for proper Camera/Host initialization process

3. If the previous checks passed, use oscilloscope to measure camera output data and signal quality

c) INT_ST_FRAME_FATAL

Error Description: Frame related error during data transmission

1. CRC error in frame data: ErrFrameData
2. SOF/EOF match failed: ErrFrameSync e.g. two consecutive SOF from the same VC without EOF
3. Frame Number sequence error: e.g. SOF frame number of previous frame is 1, current frame number becomes 4

Error Analysis:

1. Check PHY Clock and Settle configuration, please refer to *15.1.4.3a)*
2. If configured properly, refer to *15.1.4.1a)* for proper Camera/Host initialization process
3. If the previous checks passed, use oscilloscope to measure camera output data and signal quality

d) INT_ST_PHY

Error Description: data lane Warning

1. SOT check error, but recoverable by PHY. (Data unreliable): ErrSotHs
2. Escape enter error, Escape cmd only functions under ULPM: ErrEsc

Error Analysis:

1. Check for proper Camera/Host initialization sequence
2. If previous checks passed, use oscilloscope to measure camera output data and signal quality

e) INT_ST_PKT

Error Description: Packet related Warning during data transmission

1. Data type not recognized or unsupported: ErrID
2. ECC error detected and fixed: ErrEccCorrected

Error Analysis:

1. Make sure pixlen, format is configured properly, e.g. Camera sending RAW8, Host expecting RAW12
2. Check for proper Camera/Host initialization sequence
3. If previous checks passed, use oscilloscope to measure camera output data and signal quality

f) INT_ST_IPI

Error Description: IPI related Warning

1. FIFO underflow: Not enough pixels in FIFO for IPI

Error Analysis: IPI Clock too fast, or HSD too small, result in IPI emptied FIFO before PPI data transmission finished. Refer to *IPI clock Configuration* and *HSD Configuration* and check if {height, linelenth, framelenth, fps} are configured properly.

2. FIFO overflow: FIFO overflow

Error Analysis: IPI Clock too slow, or IPI data collect delayed: e.g. HSD too large or wait too long after receiving Hsync (exceeds HSD time calculated by blanking too much), resulting in FIFO full before IPI start collecting or IPI collect speed is slower than PPI filling FIFO, causing FIFO to be filled. Refer to *IPI clock Configuration* and *HSD Configuration* and check if {height, linelenth, framelenth, fps} are configured properly.

Additionally, refer to *Advanced Features* to change value in ADV_FEATURE register to adjust timing of generating Hsync signal.

3. Frame Sync Error: New Vsync arrive before previous frame end

Error Analysis: Might appear along side ErrFrameSync

4. FIFO nempty: New Vsync arrive when FIFO is not cleared

Error Analysis: Usually appear alongside FIFO Overflow or Underflow, especially overflow. When FIFO overflows, FIFO is always full, thus inducing nempty error. When FIFO underflows, since IPI reads before FIFO is properly filled, when new Vsync arrives, FIFO is left with partial data from last frame. Thus inducing nempty error.

Refer to *RX Process Error* for proper RX process. Or enable auto flush function in MEM_FLUSH register.

15.1.5.2 MIPI Device Error

If MIPI Host prompts errors, there is no need for debugging MIPI Device since Host error would induce IPI data abnormity, which in turn causing device error.

a) INT_ST_PHY

Error Description: TX D-PHY Error

1. TX tries to enter LP1, i.e. trying to pull up the line, detecting RX side control contention: errcontentionlp1
2. TX tries to enter LP0, i.e. trying to pull down the line, detecting RX side control contention: errcontentionlp0
3. TX tries to enter HS, detecting RX side control contention, causing HS timeout

Error Analysis:

AP side RX in error state and force MIPI lane (High/Low) resulting in TX failed to control Lane behavior when trying to start transmission. Refer to *TX Configuration Process* for proper configuration.

b) INT_ST_IPI

Error Description: IPI related Warning

1. errpixel: IPI received pixels length not matching configured line length

Error Analysis: Check width parameters, width both too large and small will induce such error. If width configuration differs from actual image by too much, overflow error would also be induced

2. fifo_overflow: FIFO overflow

Error Analysis: Check width parameter configuration, if width is much larger than actual size, both errpixel and overflow error would occur. Furthermore, when operating in low frequency (400Mbps/Lane), if the m value in PLL configuration is too small, overflow error would occur. This is caused by data collection slower than data sent. The current driver updated PLL calculation model to suit multiple projects. If any projects requires change of Device Clock and overflow error occurred, please refer to this section to resolve.

3. errline: IPI receiving line number not matching configured line number

Error Analysis: Check height parameter configuration, both too large and small height parameter would induce such error, if height is too small, underflow error would also occur.

4. fifo_underflow: IPI reads when FIFO is empty

Error Analysis: Check height parameter configuration, if height is too small, errline and underflow error would occur, if only underflow error occurred, refer to *IPI Configuration* and make sure Hsync is added to frame end.

15.2 Guide for Adding New Sensors

Please refer to Image Media Module Debug guide for details.

16 BPU Driver sysfs Interface

16.1 sysfs Interface Overview

BPU sysfs Interface Path:

```
/sys/devices/system/bpu
```

```
root@j3dvbj3-hynix2G-2666:/sys/devices/system/bpu# ls
bpu0      core_num  power     uevent
bpu1      group     ratio     users
```

Each interface can be accessed by cat to get information and echo to configure, the detailed description is listed below:

- bpu*: Directory, bpu0, bpu1 corresponds to the 2 cores of BPU, under each subdirectory, the structure is listed below:

```
root@j3dvbj3-hynix2G-2666:/sys/devices/system/bpu# cd bpu0
root@j3dvbj3-hynix2G-2666:/sys/devices/platform/soc/a3000000.cnn# ls
burst_len        hotplug          power_enable     uevent
devfreq          limit            power_level      users
driver           modalias         queue
driver_override  of_node          ratio
fc_time          power            subsystem
```

  - burst_len: r/w file, specifies the burst_len of the corresponding BPU core
  - hotplug: r/w file, specifies whether hotplug of the corresponding BPU core is enabled.
    - 0: disabled
    - 1: enabled
  - power_enable: r/w file, specifies whether the BPU core is powered on., used to turn on/off the power supply to the corresponding BPU core.
  - devfreq: r/w file, used to get/set the frequency of the corresponding BPU core.
    - Take BPU0 as an example:
      - Change the frequency modulation strategy to userspaces:

```
echo userspace > /sys/devices/system/bpu/bpu0/devfreq/devfreq*/governor
```

      - Check frequency supported by BPU:

```
cat /sys/devices/system/bpu/bpu0/devfreq/devfreq*/available_frequencies
```

      - Set BPU frequency (Target Frequency must be supported):

```
echo 200000000 > /sys/devices/system/bpu/bpu0/devfreq/devfreq*/userspace/set_freq
```

- Confirm frequency configured:

```
cat /sys/devices/system/bpu/bpu0/devfreq/devfreq*/available_frequencies
```

- limit: r/w file, used to configure the buffer number of the corresponding BPU Core, default value 0. Any value greater than 0 is the actual buffer number. This parameter is related to priority level. The smaller the priority value, the higher the priority. The higher the priority, the earlier the task is dispatched. Using higher priority will cause the dispatch efficiency to decrease, please configure to fit the specific tasks.
- power_level: r/w file, used to configure the power level of the corresponding BPU core (Configures the working power and frequency):
  - 1: dynamically configured by linux dvf
  - 0: performance first, highest power consumption
  - < 0: within valid range, the smaller the value, the lower the power consumption.
- users: r/o file, used to access the user info of the corresponding BPU core, detailed description is listed below by "users"
- queue: r/o file, used to access the number of configurable FunctionCall.

```
root@j3dvbj3-hynix2G-2666:/sys/devices/platform/soc/a3000000.cnn# cat queue
1024
```

- ratio: r/o file, used to access the utilization rate of the corresponding BPU Core.

```
root@j3dvbj3-hynix2G-2666:/sys/devices/platform/soc/a3000000.cnn# cat ratio
0
```

- fc_time: used to access the information of the processed tasks on the corresponding BPU core, for each tasks, detailed info is listed below:

```
root@x3dvbj3-hynix2G-3200:~# cat /sys/devices/platform/soc/a3000000.cnn/fc_time
index   id:hwid         group   prio    s_time  e_time  r_time
```

  - index: the position of the task in the BPU FIFO
  - id: User defined interrupt id
  - hwid: device driver maintained interrupt id
  - group: User defined group id, user pid
  - prio: priority level of the task
  - s_time: the timestamp when the processing of the task starts
  - e_time: the timestamp when the processing of the task has ended
  - r_time: total time consumed processing the task
- core_num: r/o file, used to access the total number of cores in BPU
- group: r/o file, used to access the group information of tasks running on the

corresponding BPU core, use "cat group" to access:

```
root@j3dvbj3-hynix2G-2666:/sys/devices/system/bpu# cat group
group           prop    ratio
0(1884)         100     0
16(1884)                100     97
```

- group: User defined group id and pid
- prop: User defined ratio
- ratio: the actual ratio of the current group on BPU

- ratio: r/o file, used to access the current utilization rate of the BPU
- users: r/o file, used to access the information of users using BPU. Users are categorized: a. BPU framework assigned tasks; b. Core assigned tasks, use "cat users" to get:

```
root@j3dvbj3-hynix2G-2666:/sys/devices/system/bpu# cat users
*User via BPU Bus*
user            ratio
1884            0
*User via BPU Core(0)*
1884            49
*User via BPU Core(1)*
1884            49
```

- user: user pid
- ratio: the utilization rate taken by the current user

16.2 Examples

The following examples are all targeting BPU0, all commands can be executed after there is a model running on the BPU.

16.2.1 Power off BPU Core

Execute the following command:

```
echo 0 > /sys/devices/system/bpu/bpu0/power_enable
```

16.2.2 Hotplug BPU Core

Hotplug will not affect models running on single core, dual core model does not support hotplug. When hotpug is enabled, it will not be disabled automatically. If hotplug is not needed, please manually disable hotplug (echo 0 to the corresponding sysfs interface)

Execute the following command:

```
echo 1 > /sys/devices/system/bpu/bpu0/hotplug
```

```
echo 0 > /sys/devices/system/bpu/bpu0/power_enable
```

16.2.3 Decrease BPU Core Power Consumption

The following command will NOT power off the BPU core, it only decreases the frequency and power consumption of the corresponding BPU core, the detailed description of the sysfs interface is listed above.

```
echo -2 > /sys/devices/system/bpu/bpu0/power_level
```

16.2.4 Priority Model Execution

Please compile and use the priority model according to the manual of HBDK compiler.

Use hb_bpu_core_set_fc_prio or hb_bpu_core_set_fc_group with specific group_id to configure the priority of the tasks:

```
echo 2 > /sys/devices/system/bpu/bpu0/limit
```

limit can be used in debugging. User can also use the following command to dynamically configure limit before executing the application:

```
export BPLAT_CORELIMIT=2
```

17 Temperature Sensor Usage

Currently there are 3 temperature sensors: 1 on DVB, 1 on Ethernet, and the last 1 on SOM.

Under /sys/class/hwmon/, there are 3 hwmonX directory, each directory contains sysfs interface for each temperature sensor.

For example, under /sys/class/hwmon/hwmon2, there are 2 important files: name and temp1_input:

- name: file specifying the name of the temperature sensor
- temp1_input: file specifying the value of the reading of the temperature sensor, measured in mili-degree Celsius.

18 Memory Management

18.1 UBoot

In UBoot, it is possible to modify the size of total memory available to system and the size of reserved memory.

e.g. for physical memory of size 1G, can be restricted to 512M.

```
setenv mem_size 0x20000000
```

Modify the reserved memory to be 128M, below, region ion is the physically consecutive memory region reserved for IPU, BPU and ISP

```
setenv ion_size  '128'
boot
```

18.2 IPU Memory Reservation Configuration

Every IPU channel has a parameter to control the number of buffers, HAL will reserve memory through ion interface according to the number of buffers. The parameter is listed in VIO JSON config file. VIO config files are under directory:

```
/etc/vio/
```

The parameter is under the name of:

```
"us_buf_num"
```

Detailed description of VIO config file, please refer to Graphic System API Manual.

18.3 BPU Memory Reservation Configuration

When ion choose CMA as memory pool, use CMA area to reserve space for BPU mem. In this implementation, BPU memory reservation is more convenient and efficient. The size of reserved space can be changed after boot using sysfs node:

```
echo 100 >  /sys/class/misc/ion/cma_carveout_size
```

Using the command above to change reserved memory size (measured in MBytes). Please modify according to the actual requirement of different scenarios. For example, if VIO report ion_alloc error while running multiple lanes of cameras, reduce memory reserved by BPU. When "0" is specified, BPU will only use dynamically allocated memory from CMA. (If the above sysfs node is not present, the System software version does NOT support dynamic configuration)

Note: Only when user is NOT using BPU_MEM, can the above configuration success. Since the reserved space will be contiguous physical memory, the maximum reserved

BPU memory cannot exceed CMA size. When BPU_MEM cannot reserve enough space from CMA, system will try to dynamically request memory from outside of CMA.

Since the reservation is contiguous memory space from CMA, there are possibilities that reservation cannot succeed. After setting desired size to the above sysfs node, read from it to check if configuration is success, 0 means reservation has failed.

19 Using kgdb to debug Kernel

19.1 Debug Booting Sequence

To debug booting sequence (after kgdb and serial driver install), compile kgdb module to Kernel. Add following option to kernel boot option, please configure baudrate to suit your need, default is 921600:

```
kgdboc=ttyS0,921600 kgdbwait
```

After kernel boot, the following should appear:

```
kgdb: Registered I/O driver kgdboc.
kgdb: Waiting for connection from remote gdb...
```

Enter "kgdb" in kdb command line,and board would be waiting for PC gdb connection:

```
kdb> kgdb
Entering please attach debugger or use $D#44+ or $3#33
```

19.2 PC gdb connection

While prompting "waiting for connection", close serial port (to prevent port contention). Connect through proper gdb tool on PC (aarch64-linux-gnu-gdb) to load kernel vmlinux file. Connect through PC serial port to board (/dev/ttyUSBx). Use "sudo" if necessary:

```
aarch64-linux-gnu-gdb ./vmlinux
(gdb) set serial baud 921600
(gdb) target remote /dev/ttyUSB0
```

If connection established, use gdb command to debug kernel.

20 Memory dump Function Manual

More functions are introduced in: *swinfo Manual (mem-dump function synopsis)*

The dumped img can be evaluated by steps listed in: *Using crash to Analysis ramdump*

20.1 Preparation

Based on your Scenerio, select different dump method:

20.1.1 Through TFTP Network

PC internet port connect to board under examination, setup tftp:

Manually setup TFT, please refer to *Environment setup (tftp server and nfs setup)*

Horizon Upgrade tool hbupdate comes with tftp service.

20.1.2 Dump to TF/SD

Format TF/SD card using gpt partition with as FAT File system. Use mkfs.fat onboard if preferred.

```
parted /dev/mmcblk1 mktable gpt
parted /dev/mmcblk1 mkpart primary fat32 1Mib 100%
mkfs.fat /dev/mmcblk1p1
```

20.1.3 Dump to EMMC

After entering recovery mode, use the following command to format "userdata"

```
umount /userdata
mkfs.ext4 -O 64bit -L userdata -F /dev/mmcblk0p12
mount -t ext4 /dev/mmcblk0p12 /userdata
```

In kernel, use the following command to enter Recovery mode:

```
hrut_resetreason recovery
reboot
```

20.1.4 Dump to USB Drives

USB drives must be formatted to FAT32.

20.2 Manual dump

Manual dump of memory is done in UBoot.

Linux provides option stopping boot sequence at UBoot after reboot. Use this option to manually dump memory:

```
echo 4 >/sys/kernel/hobot-swinfo/boot
```

Note: If set to stop at UBoot after reboot, use UBoot command to clear the flag to return to normal booting sequence:

```
swinfo boot 0
```

Previous setting is not required for dumping memory, enter UBoot in any way.

20.2.1 Manual Dump through TFTP

Setup Local IP address and tftp server. Use tput command to dump memory to tftp server root directory. Below, 0x200000 to 0x3f200000is the non-mpu protected area for 1G DDR. Please adjust to your own board.

```
setenv ipaddr 192.168.1.10;setenv serverip 192.168.1.1; tput 0x200000 0x3fe00000
dump_ddr_40000000.img
```

20.2.2 Manual Dump to TF/SD Card

Use fatwrite to dump memory to TF/SD card root directory. Below, 0x200000 to 0x3f200000is the non-mpu protected area for 1G DDR. Please adjust to your own board.

```
mmc rescan; part list mmc 1; fatwrite mmc 1:1 0x200000 /dump_ddr_40000000.img 0x3fe00000
```

20.2.3 Manual Dump to EMMC

Use ext4write to dump to /userdata directory in eMMC. Below, Below, 0x200000 to 0x3f200000is the non-mpu protected area for 1G DDR. Please adjust to your own board.

```
mmc rescan; part list mmc 0; ext4write mmc 0:c 0x200000 /dump_ddr_40000000.img 0x3fe00000
```

If "hash tree directory Error ext4fs_write()" occurred, please write to a folder under /userdata, for example, /userdata/log:

```
ext4write mmc 0:c 0x200000 /log/dump_ddr_40000000.img 0x3fe00000
```

20.3 Automatic Dump

Linux provides option to enter UBoot and finish dump automatically after crash. (After configuration, system will boot normally, when system crashes and reboot, Linux will dump dump_ddr_xxx.img for future analysis)

20.3.1 Dump through TFTP

Configure force dump through tftp automatically after reboot: (assuming tftp server IP address: 192.168.1.64)

```
echo 192.168.1.64 > /sys/kernel/hobot-swinfo/dump
```

Previous command operates on flag, will force next boot into UBoot to perform dump. If dump only after panic, use the following command:

```
echo dump=192.168.1.64 > /sys/kernel/hobot-swinfo/panic
```

20.3.2 Dump to TF/SD Card

Configure force dump to TF/SD Card automatically after reboot:

```
echo 5 > /sys/kernel/hobot-swinfo/boot
```

Previous command operates on flag, will force next boot into UBoot to perform dump. If dump only after panic, use the following command:

```
echo boot=5 > /sys/kernel/hobot-swinfo/panic
```

20.3.3 Dump to EMMC

Configure force dump to /userdata directory on eMMC automatically after reboot:

```
echo  6 > /sys/kernel/hobot-swinfo/boot
```

Previous command operates on flag, will force next boot into UBoot to perform dump. If dump only after panic, use the following command:

```
echo boot=6 > /sys/kernel/hobot-swinfo/panic
```

Note: After configuring the previous dump configures, after reboot (or panic reboot), system will boot into UBoot and execute dump command and stays in UBoot regardless of execution result of dump. If dump failed, use autodump command to dump again.

```
run dumpcmd
```

More detailed explanation, please refer to: *swinfo Manual (mem-dump function synopsis)*

To analysis dump file, please refer to: *Using crash to Analysis ramdump*

21 swinfo Manual (mem-dump function synopsis)

21.1 mem-dump Overview

21.1.1 Function

During system debug or BUG reproduction, configure mem-dump environment (including PC and board). After system crash or panic, setup board to enter different boot or automatically dump memory image for crash analysis.

21.1.2 Implementation

Reserved a memory region accessible by both SPL, UBoot and Kernel, to store flags and other configs, also provide interface accordingly.

Currently, SWREG of PMU and sw_reserved are used, this is configurable.

Index(4B)	SWREG	SWMEM	Function	Description
0	SWREG0	MEM+00	NC/MAGIC	SWREG0 is used in SPL for sleep and resume, reserved; MEM+00 stores MAGIC Flag, when valid, use SWMEM, otherwise SWREG
1	SWREG1	MEM+04	boot	[bit3:0] boot Mode: 0-regular boot; detailed below
2	SWREG2	MEM+08	dump	PC IP address for tftp dump: 0-do not dump; Non zero: IP address of PC, e.g. 192.168.1.64 <-> 0xC0A80140

All field mentioned previously in SWREG and SWMEM can be used as flag are. Either SWREG or SWMEM is used is decided when executed, marked by MAGIC in SWMEM[0].

Reboot will have different effect on data in SWREG and SWMEM as shown below:

Operation	Software reboot/reset	Hardware RESET	Unplugged
SWREG	Keep	Clear	Clear
SWMEM	Keep	Keep	Clear

Currently 9 type of boot is supported (including 2 dump operation):

Boot Index	Type	Description	Note
0	normal	Boot kernel	Default

1	splonce	Wait in SPL and reset flags	Wait only once, hardware reset will return to normal boot
2	ubootonce	Wait in UBoot, reset flags	Wait only once, hardware reset will return to normal boot
3	splwait	Wait in SPL without resetting flags	If using SWMEM, only unplugging can return to normal boot
4	ubootwait	Wait in UBoot without resetting flags	If using SWMEM, without manually resetting flags, only unplugging can return to normal boot
5	udumptf	UBoot dump memory to tf card	Automatically dump memory image to TF/SD card in FAT32 format
6	udumpemmc	UBoot dump memory to eMMC	Automatically dump memory image to /userdata in eMMC in EXT4 format
7	udumpusb	UBoot: dump memory to USB Drives	Automatically dump the memory image to the first partition of the USB drive formatted to FAT32
8	udumpfastboot	UBoot: dump memory through fastboot	On PC, use fastboot, executing command: Fastboot oem ramdump

21.2 UBoot Operation Interface

21.2.1 swinfo Command

UBoot is integrated with swinfo command to operate on SWREG and SWMEM:

```
swinfo command
swinfo - swinfo sub system

Usage:
swinfo info [reg/mem] - display info of the swinfo # Check current SWINFO region(Automatically
# decide SWREG/SWMEM)
swinfo sel [reg/mem] - select reg/mem                    # Check/select SWREG/SWMEM for
SWINFO
swinfo reg [index [value]] - get/set reg              # Check/set SWREG value stored in index
swinfo mem [index [value]] - get/set mem              # Check/set SWMEM value stored in index
swinfo boot [type] - get/set boot info                    # Check/set boot type in SWINFO
swinfo dump [iphex] - get/set dump ip in hex          # Check/set dump IP address in SWINFO
```

21.2.2 Example

Check all configure value in SWINFO:

```
### swinfo info ###
Hobot>swinfo info
swinfo: reg -- 00000000a6000200
00: 00000000 00000000 00000000 00000000
10: 00000000 00000000 00000000 00000000
20: 00000000 00000000 00000000 00000000
30: 00000000 00000000 00000000 00000000
40: 00000000 00000000 00000000 00000000
50: 00000000 00000000 00000000 00000000
60: 00000000 00000000 00000000 00000000
70: 00000000 00000000 00000000 00000000
swinfo boot: 0 normal
swinfo dump: none
```

Changed to use SWMEM:

```
swinfo sel mem
```

The following prompts should appear

```
swinfo sel: mem
```

Change boot type to 4(ubootwait):

```
Hobot>swinfo boot 4
```

The following prompts should appear

```
swinfo boot: 4 ubootwait
```

Enable dump, configure PC IP address to: 192.168.1.64

```
swinfo dump 0xc0a80140
```

The following prompts should appear

```
swinfo dump: 192.168.1.64
```

Check dump flag:

```
swinfo dump
```

The following prompts should appear

```
swinfo dump: 192.168.1.64
```

Directly write to SWREG2: 0xAA55AA55 (writing to SWMEM is similar)

```
swinfo reg 2 0xAA55AA55
```

The following prompts should appear

```
swinfo reg: 2[14]: 00000000 -> AA55AA55
```

Directly read SWREG2. (Read SWMEM is similar)

```
swinfo reg 2
```

The following prompts should appear

```
swinfo reg: 2[14]: AA55AA55
```

21.3 Kernel Interface

21.3.1 Sysfs Interface

Linux Kernel in integrated with similar operation to SWINFO. Under /sys/hobot-swinfo:

```
ls /sys/hobot-swinfo
boot    # boot type get/set
dump    # dump type get/set
panic   # panic action get/set
sel     # SWREG/SWMEM check/set
swinfo  # SWINFO info check(automatically select SWREG/SWMEM)
swmem   # SWMEM check/set
swreg   # SWREG check/set
```

21.3.2 Example

Check all current value in SWINFO:

```
cat /sys/kernel/hobot-swinfo/swinfo
```

The following should appear:

```
~# cat /sys/kernel/hobot-swinfo/swinfo
 0[00-ro]: 0x00000000 [MAGIC]
 1[04-rw]: 0x00000000
 2[08-rw]: 0x00000000
 3[0C-rw]: 0x00000000
```

Change to using SWMEM:

```
echo mem > /sys/kernel/hobot-swinfo/sel
```

The following should appear:

```
~# echo mem > /sys/kernel/hobot-swinfo/sel
```

Change boot type to 4(ubootwait):

```
echo 4 > /sys/kernel/hobot-swinfo/boot
```

Enable dump, set PC IP address to: 192.168.1.64:

```
echo 192.168.1.64 > /sys/kernel/hobot-swinfo/dump
```

Check dump type:

```
cat /sys/kernel/hobot-swinfo/dump
```

The following should appear:

```
~# cat /sys/kernel/hobot-swinfo/dump
dump: 0xC0A80140 192.168.1.64
```

Directly write to SWREG2: 0xAA55AA55 (writing to SWMEM is similar):

```
echo 2=0xAA55AA55 > /sys/kernel/hobot-swinfo/swreg
```

Directly read SWREG2 (Reading SWMEM is similar):

```
cat /sys/kernel/hobot-swinfo/swreg |grep " 2["
```

The following should appear:

```
2[08-rw]: 0xAA55AA55
```

Previous echo will affect SWMEM/SWREG directly. Effective the next boot. If need to boot normally:

```
echo 0 to reset flags.
```

Additionally, panic interface is added, enables dumping only when system panic (Note: does not work when system hang). This way, boot/dump debug of reboot is not affected.

Set boot type to 4(ubootwait, all boot type as defined above) when panic:

```
echo boot=4 >/sys/kernel/hobot-swinfo/panic
~# echo boot=4 >/sys/kernel/hobot-swinfo/panic
```

Set dump PC IP address to 192.168.1.64 when panic:

```
echo dump=192.168.1.64 >/sys/kernel/hobot-swinfo/panic
~# echo dump=192.168.1.64 >/sys/kernel/hobot-swinfo/panic
```

Check panic preset operation:

```
cat /sys/kernel/hobot-swinfo/panic
~# cat /sys/kernel/hobot-swinfo/panic
```

```
panic: dump=0xC0A80140 192.168.1.64
```

21.4 dump Usage

21.4.1 Scenario: Automatically dump memory Image through TFTP

For typical bug reproduction, setup test and dump environment. After system crash and reboot, board will automatically boot into UBoot and dump memory image through tput for future analysis.

21.4.1.1 Setup tftp Server on PC

Manually, please refer to: *Environment setup (tftp server and nfs setup)*

Or, use tftp service come with hbupdate tool.

Configure PC IP address properly, e.g.: 192.168.1.64(For configuration below)

21.4.1.2 Configure dump flags

First, configure before debugging: Two scenarios:

Kernel boots normally: Execute following command

```
echo 192.168.1.64 > /sys/kernel/hobot-swinfo/dump
```

Next Boot (even normal reboot and reset) will enter dump immediately, if dump only after panic (without affecting reboot or hardware reset):

```
echo dump=192.168.1.64 > /sys/kernel/hobot-swinfo/panic
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo dump 0xc0a80140
```

21.4.1.3 Test automatic dump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

```
echo c > /proc/sysrq-trigger
```

After reboot, board will enter UBoot, according to dump configuration, setup serverip and ipaddr. Use tput to write memory image to dump_ddr_${ddr_size}.img file.

Note: IP address should comply following restrictions:

Board and PC should be in the same subnet,

If serverip is not 1, then board ipaddr is set to 1; if serverip is set to 1, then board ipaddr will be 10 please see below for example:

```
swinfo dump ddr 0x3fe00000 1/0.85.1/0.1 -> 85
Phy name:Marvell 88E1518
Phy uid:1410dd0
ethernet@A5014000 Waiting for PHY auto negotiation to complete...... done
set mac_div_clk = 125000000Using ethernet@A5014000 device
TFTP to server 170.85.170.85; our IP address is 170.85.170.1
Filename 'dump_ddr_3fe00000.img'.
Save address: 0x200000
Save size:    0x3fe00000
Saving: *
```

After dump success, dump flags will be reset, system stay in UBoot until further command.

Note: After successfully configure dump flags, if normal reboot is needed, Manfully reset flags (Not necessary for panic dump options) as below:

In Linux Kernel, before reboot:

```
echo 0 > /sys/kernel/hobot-swinfo/dump
```

In UBoot, before reset:

```
swinfo dump 0
```

21.4.2 Scenario: Automatically dump memory Image to TF/SD Card

For circumstances where tftp is not available (or too slow), when the board is equipped with TF/SD card, memory image can be dumped directly into TF/SD card.

21.4.2.1 TF/SD card Format

To prepare TF/SD card for UBoot to detect and mount correctly, partition and format TF/SD card

In Linux, use mkfs.fat, in case onboard system not integrated with mkfs.fat, please cross-compile ARM64 parted and mkfs.fat. The following example assumes targeted SD card is mounted to mmcblk1, before formatting, please umount SD card:

Formatting TF/SD Card:

```
~# parted /dev/mmcblk1 mktable gpt
~# parted /dev/mmcblk1 mkpart primary fat32 1Mib 100%
~# mkfs.fat /dev/mmcblk1p1
```

Alternatively, Format TF/SD card on PC with tools you prefer. (Please change "mmcblk1" to the device appear on PC when SD card is inserted)

21.4.2.2 Configure boot flags

Two scenarios:

Kernel boots normally: Execute following command

```
echo 5 > /sys/kernel/hobot-swinfo/boot
```

Next Boot (even normal reboot and reset) will enter dump immediately, if dump only after panic (without affecting reboot or hardware reset):

```
echo boot=5 > /sys/kernel/hobot-swinfo/panic
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 5
```

21.4.2.3 Test autodump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

```
echo c > /proc/sysrq-trigger
```

After reboot into UBoot, based on boot configuration, the following prompt should apper:

```
swinfo dump ddr 0x40000000 -> tfcard:1
```

Use fatwrite to dump memory image to partition 1 of the TF/SD card. After dump success, reset dump flags and stays in UBoot until further command.

In UBoot, use fatls to check the dumped dump_ddr_xxxx.img file. Afterwards, take TF/SD card and use the dumpfile for further analysis.

```
fatls mmc 1:1
```

21.4.3 Scenario: Automatically dump memory Image to eMMC

For scenarios where tftp is not available (or too slow) and the board is not equipped with TF/SD card, dump memory image directly into partition userdata of eMMC.

21.4.3.1 Configure boot flags

Two scenarios:

Kernel boots normally: Execute following command

```
echo 6 > /sys/kernel/hobot-swinfo/boot
```

Next Boot (even normal reboot and reset) will enter dump immediately, if dump only after panic (without affecting reboot or hardware reset):

```
echo boot=6 > /sys/kernel/hobot-swinfo/panic
```

kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 6
```

21.4.3.2 Test autodump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

```
echo c > /proc/sysrq-trigger
```

After reboot into UBoot, based on boot configuration, the following prompt should paper:

```
swinfo dump ddr 0x40000000 -> emmc:12
```

Use fatwrite to dump memory image to partition 12(userdata) of the EMMC. After dump success, reset dump flags and stays in UBoot until further command.

In UBoot, use fatls to check the dumped dump_ddr_xxxx.img file. Afterwards, take dump file out through sftp or tftp and use the dumpfile for further analysis.

```
ext4ls mmc 0:c
```

21.4.4 Scenario: Automatically dump memory Image to USB Drive

21.4.4.1 Configure boot flags

Two scenarios:

Kernel boots normally: Execute the following command:

```
echo udumpusb > /sys/kernel/hobot-swinfo/boot
```

Next Boot (even normal reboot and reset) will enter dump immediately, if dump only after panic (without affecting reboot or hardware reset):

```
echo boot=7 > /sys/kernel/hobot-swinfo/panic
```

kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 7
```

21.4.4.2 Test autodump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

```
echo c > /proc/sysrq-trigger
```

After reboot into UBoot, based on boot configuration, the following prompt should paper:

```
swinfo dump ddr 0x40000000 -> usb
```

The memory Image will be dumped to USB drive under the name of "dump_ddr_xxxx.img".

21.4.5 Scenario: Automatically dump memory Image via fastboot

21.4.5.1 Configure boot flags

Two scenarios:

Kernel boots normally: Execute the following command:

```
echo udumpfastboot > /sys/kernel/hobot-swinfo/boot
```

Next Boot (even normal reboot and reset) will enter dump immediately, if dump only after panic (without affecting reboot or hardware reset):

```
echo boot=8 > /sys/kernel/hobot-swinfo/panic
```

kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 8
```

21.4.5.2 Test autodump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

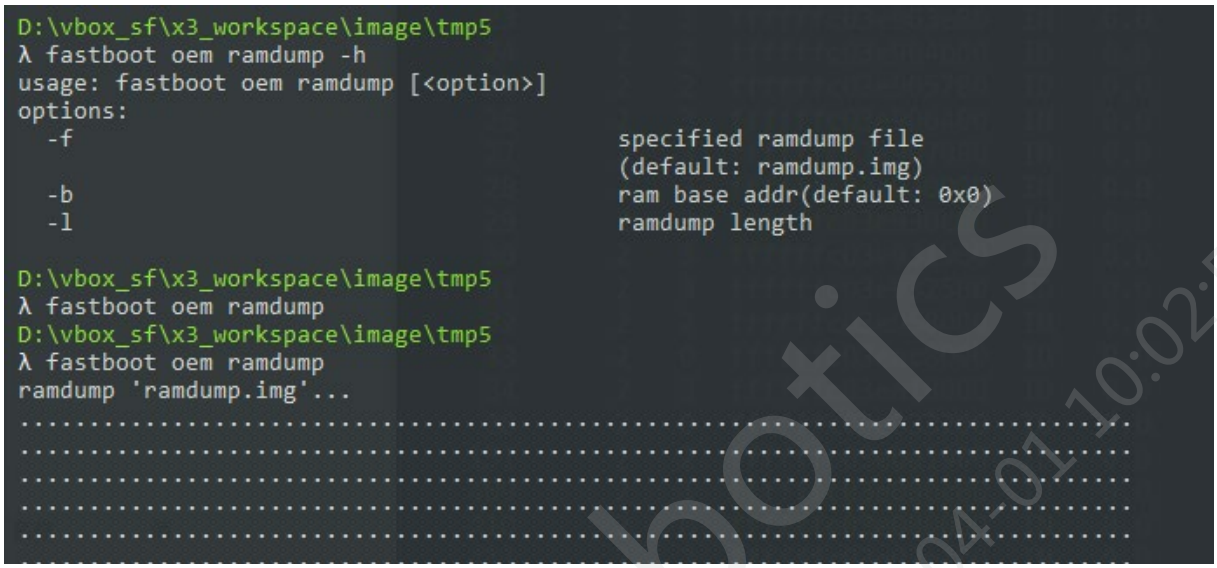
```
echo c > /proc/sysrq-trigger
```

After reboot into UBoot, based on boot configuration, the following prompt should appear:

```
enter fastboot ramdump mode
please use "fastboot oem ramdump" command in pc
```

Use fastboot tools on PC to execute the following command:

```
fastboot oem ramdump
```



Note:

If sw_info is not properly configured, after panic, the board will enter UBoot automatically, execute the following command:

```
fastboot 0
```

to enter Fastboot.

21.4.6 Scenerio: Reboot and stop at SPL

For boards equipped with BIF interface and can access memory on board through AP or pc, stop at SPL to maintain original status of memory and dump memory image.

First configure before application is executed, the method is listed below:

Configure boot flags: Two scenarios:

Kernel boots normally: Execute following command

```
echo 1 > /sys/kernel/hobot-swinfo/boot
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 1
```

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic. After panic reboot, enters SPL and wait in SPL as shown below: (Press RESET to reboot normally):

```
hang for swinfo boot: splonce
### ERROR ### Please RESET the board ###
```

If need to preserve very important scenarios, configure the board to reboot normally only after unplug:

Kernel boots normally: Execute following command

```
echo mem > /sys/kernel/hobot-swinfo/sel
echo 3 > /sys/kernel/hobot-swinfo/boot
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo sel mem
swinfo boot 3
```

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic. After panic reboot, enters SPL and wait in SPL as shown below: (Press RESET will not reboot, only unplug will):

```
hang for swinfo boot: splonce
### ERROR ### Please RESET the board ###
```

Note: After successfully configure dump flags, if normal reboot is needed, Manfully reset flags as below:

In Linux Kernel, before reboot:

```
echo 0 > /sys/kernel/hobot-swinfo/boot
```

In UBoot, before reset:

```
swinfo boot 0
```

21.4.7 Scenerio: Reboot and stop at UBoot

For scenarios require stopping at UBoot, configure as below:

Configure boot flags: Two scenarios:

Hardware RESET button resets the board normally

Two Scenarios:

Kernel boots normally: Execute following command

```
echo 2 > /sys/kernel/hobot-swinfo/boot
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo boot 2
```

Test autodump

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic. After panic reboot, enters UBoot and wait in UBoot as shown below: (Press RESET or use reset command to reboot normally):

```
uboot: normal boot
wait for swinfo boot: ubootonce
Hobot>
```

If need to preserve very important scenarios, configure the board to reboot normally only after unplug:

Kernel boots normally: Execute following command

```
echo mem > /sys/kernel/hobot-swinfo/sel
echo 4 > /sys/kernel/hobot-swinfo/boot
```

Kernel boot abnormal, Execute following command in UBoot console:

```
swinfo sel mem
swinfo boot 4
```

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic. After panic reboot, enters SPL and wait in SPL as shown below: (Press RESET will not reboot, only unplug or resetting flags will):

```
uboot: normal boot
wait for swinfo boot: ubootwait
Hobot>
```

Note:

Board in ubootwait status, can unplug to reboot normally or use following command:

```
### In UBoot, before reset: ###
swinfo boot 0
```

21.5 Function Tests

21.5.1 Select SWREG/SWMEM

In UBoot: Check the current Default to be SWREG (reg)

```
swinfo sel
```

Switch to SWMEM from SWREG:

```
swinfo sel mem
```

Reset:

```
reset
```

Enter UBoot after reboot, check current valid swinfo location:

```
swinfo sel
```

Now, UBoot should prompt: mem for using SWMEM

21.5.2 Set boot type

In Linux Shell: (take boot type 1 as example)

```
echo 1 > /sys/kernel/hobot-swinfo/boot
reboot
```

Or in UBoot: (take boot type 1 as example)

```
swinfo boot 1
reset
```

Reboot will enter different status:

- 0 – normal, Normal Linux boot sequence
- 1 – splonce, Stop at SPL, Hardware RESET possible
- 2 – UBootonce, Stop at UBoot, Hardware RESET possible
- 3 – splwait, Stop at SPL, using SWREG can Hardware RESET, otherwise(SWMEM), must unplug
- 4 – UBootwai, Stop at UBoot, using SWREG can Hardware RESET, otherwise(SWMEM), must unplug
- 5 – udumptf, After panic, automatically dump memory image to TF/SD card
- 6 – udumpemmc, After panic, automatically dump memory image to partition 8(userdata) of emmc
- 7 – udumpusb, After panic, Automatically dump memory image to the first partition of USB drive formatted to FAT32
- 8 – udumpfastboot, After panic, automatically dump memory image via Fastoboot.

21.5.3 dump function

Refer to: *Environment setup (tftp server and nfs setup)* to setup tftp server properly.

```
echo 192.168.1.64 > /sys/kernel/hobot-swinfo/dump
```

After configuration, run tests and wait for panic reboot. Alternatively, use sysrq to create panic:

```
echo c > /proc/sysrq-trigger
```

After panic reboot, enter UBoot and create a dump_ddr_xxx.img file with a size of 1G

21.5.4 panic preset function

Please refer to *21.3.2*, configure only after panic (normal reboot not affected):

```
echo boot=2 > /sys/kernel/hobot-swinfo/panic
echo c > /proc/sysrq-trigger
```

After panic reboot, enter UBoot and wait. When preset function is executed, execute reboot under UBoot to enter Kernel:

```
echo boot=2 > /sys/kernel/hobot-swinfo/panic
reboot
```

For non-panic reboot, the previous command will have no effect on normal reboot sequence.

22 Using crash to Analysis ramdump

22.1 Preparation

To run crash on PC, compile from source code. Source code of crash can be obtained from:

<https://people.redhat.com/anderson/crash-7.2.6.tar.gz>

Unzip and enter directory, execute commands below:

```
make target=arm64
make install
```

And crash should be properly installed.

22.2 crash Introduction

Crash is used primarily for offline Linux kernel memory dump analysis. Crash integrated gdb tool, and can check stack, dmesg logs, kernel data structures disassemble etc. It also supports multiple format of memory dump, including:

- Live Linux System
- kdump produced normal and compressed memory dump
- makedumpfile produced compressed memory dump
- netdump produced memory dump
- diskdump produced memory dump.
- kdump produced Xen memory dump
- LKCD produced memory dump
- Mcore produced memory dump
- rawmemory dump in format ramdump

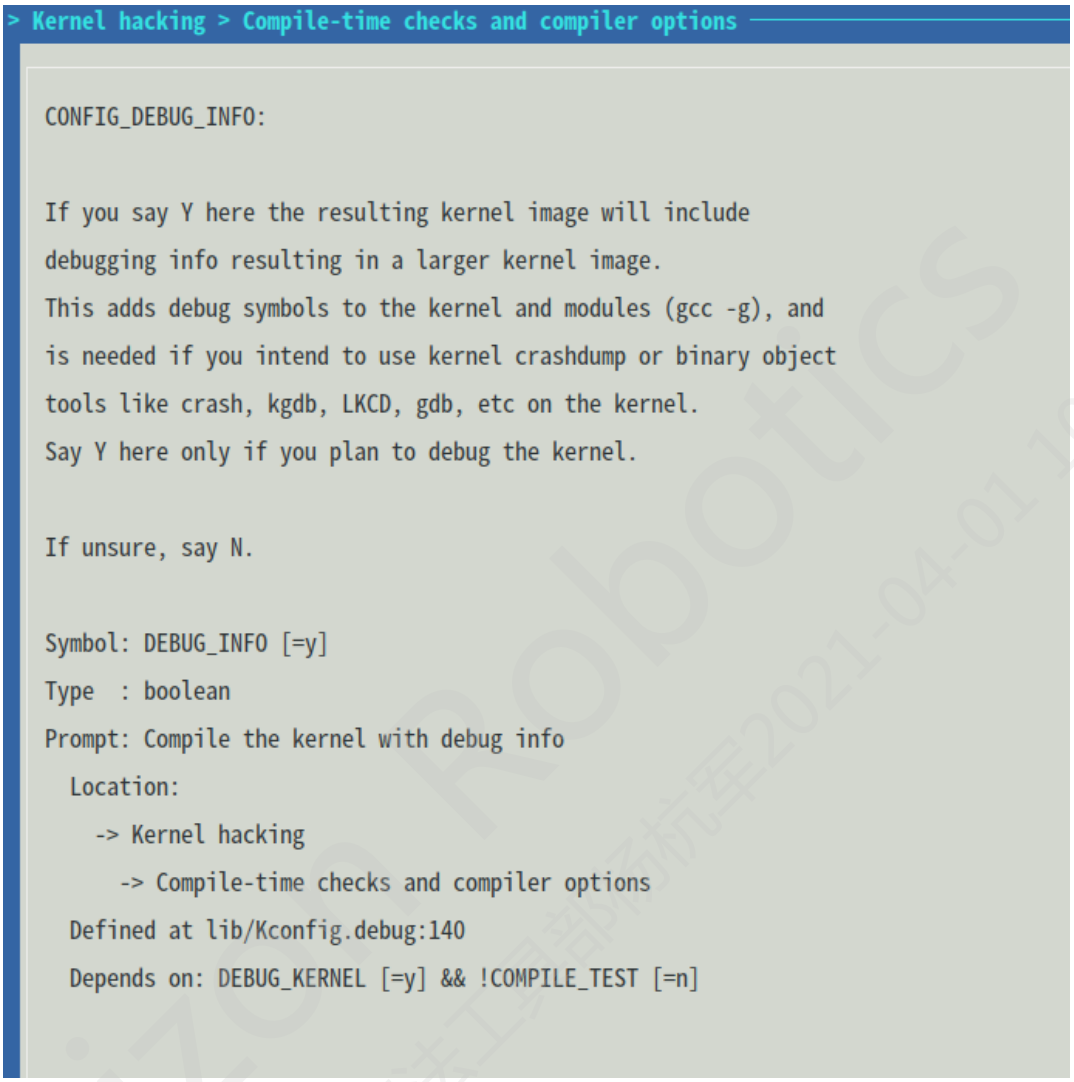
22.3 crash Usage

This section primarily focus on using crash with ramdump files. ramdump file is a mirror image of almost entire memory except for memory of security type. Some BUG is rarely reproduce, some of these BUGs are induced by memory corruption, which can hardly be exposed by simple logs. Using ramdump files provides a mean for debugging such problems.

In Command Prompt:

```
#crash <vmlinux> <ram_dump_img@dram_addr_phy_start>
```

Note: vmlinux and ramdump must use the same kernel. The kernel used must also be configured with CONFIG_DEBUG_INFO option:



In the command, physical address of physical memory is attached after "@". If run successfully, the following message should prompt in command prompt:

```
      KERNEL: kernel-4.14/vmlinux
    DUMPFILE: myfile
        CPUS: 2 [OFFLINE: 1]
        DATE: Thu Jan  1 08:00:20 1970
      UPTIME: 00:00:20
LOAD AVERAGE: 0.31, 0.07, 0.02
       TASKS: 46
    NODENAME: dhw
     RELEASE: 4.14.74-g2acb89ada-dirty
     VERSION: #164 SMP PREEMPT Fri Jul 19 12:25:04 CST 2019
     MACHINE: aarch64  (unknown Mhz)
      MEMORY: 1 GB
       PANIC: "sysrq: SysRq : Trigger a crash"
         PID: 836
     COMMAND: "sh"
        TASK: ffffffc05cb16c00  [THREAD_INFO: ffffffc05cb16c00]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)

crash>
```

From here on, detailed system info is shown and crash command can be used for further analysis.

22.4 Crash commands

22.4.1 ps

Output process info.

```
   PID    PPID  CPU       TASK        ST  %MEM     VSZ    RSS  COMM
>     0      0   0  ffffff800888d480  RU   0.0       0      0  [swapper/0]
      0      0   1  ffffffc05c890c00  RU   0.0       0      0  [swapper/1]
      1      0   1  ffffffc05c848000  IN   0.0    2132   1172  init
      2      0   1  ffffffc05c848c00  IN   0.0       0      0  [kthreadd]
      3      2   0  ffffffc05c849800  ID   0.0       0      0  [kworker/0:0]
      4      2   0  ffffffc05c84a400  ID   0.0       0      0  [kworker/0:0H]
      5      2   0  ffffffc05c84b000  ID   0.0       0      0  [kworker/u4:0]
      6      2   0  ffffffc05c84bc00  ID   0.0       0      0  [mm_percpu_wq]
      7      2   0  ffffffc05c84c800  IN   0.0       0      0  [ksoftirqd/0]
      8      2   0  ffffffc05c84d400  ID   0.0       0      0  [rcu_preempt]
      9      2   0  ffffffc05c84e000  ID   0.0       0      0  [rcu_sched]
     10      2   0  ffffffc05c84ec00  ID   0.0       0      0  [rcu_bh]
     11      2   0  ffffffc05c890000  IN   0.0       0      0  [migration/0]
     12      2   0  ffffffc05c891800  IN   0.0       0      0  [cpuhp/0]
     13      2   1  ffffffc05c892400  IN   0.0       0      0  [cpuhp/1]
     14      2   1  ffffffc05c893000  IN   0.0       0      0  [migration/1]
     15      2   1  ffffffc05c893c00  IN   0.0       0      0  [ksoftirqd/1]
     16      2   1  ffffffc05c894800  ID   0.0       0      0  [kworker/1:0]
     17      2   1  ffffffc05c895400  ID   0.0       0      0  [kworker/1:0H]
     18      2   1  ffffffc05c896000  IN   0.0       0      0  [kdevtmpfs]
     19      2   1  ffffffc05c896c00  ID   0.0       0      0  [netns]
     20      2   0  ffffffc05c930000  ID   0.0       0      0  [kworker/u4:1]
     24      2   0  ffffffc05c933000  ID   0.0       0      0  [kworker/u4:2]
     37      2   0  ffffffc05c936c00  ID   0.0       0      0  [kworker/0:1]
     43      2   0  ffffffc05c930c00  ID   0.0       0      0  [kworker/u4:3]
     50      2   1  ffffffc05c9b4800  RU   0.0       0      0  [kworker/1:1]
    114      2   1  ffffffc05c935400  ID   0.0       0      0  [kworker/u4:4]
```

Process status indicating:

- RU: Status R, process is under the status TASK_RUNNING.
- IN: Status S, process is under the status TASK_INTERRUPTIBLE.
- UN: Status D, process is under the status TASK_UNINTERRUPTIBLE.

22.4.2 bt <pid>

Check stack of process(thread) <pid>.

```
bt -a: categorized by tasks, display stack for each task
bt -t: print all text identifier in the stack of current task
bt -f: print all data in the stack of current task, usually used to check parameters passing
between functions
```

22.4.3 dis <function name>

Disassemble <function name>.

```
crash> dis schedule
0xffffff8008602450 <schedule>:  stp     x29, x30, [sp,#-32]!
0xffffff8008602454 <schedule+4>:        mrs     x0, sp_el0
0xffffff8008602458 <schedule+8>:        mov     x29, sp
0xffffff800860245c <schedule+12>:       str     x19, [sp,#16]
0xffffff8008602460 <schedule+16>:       ldr     x1, [x0,#24]
0xffffff8008602464 <schedule+20>:       cbz     x1, 0xffffff8008602470 <schedule+32>
0xffffff8008602468 <schedule+24>:       ldr     x1, [x0,#1752]
0xffffff800860246c <schedule+28>:       cbz     x1, 0xffffff80086024a8 <schedule+88>
0xffffff8008602470 <schedule+32>:       mrs     x19, sp_el0
0xffffff8008602474 <schedule+36>:       ldr     w0, [x19,#16]
0xffffff8008602478 <schedule+40>:       add     w0, w0, #0x1
0xffffff800860247c <schedule+44>:       str     w0, [x19,#16]
0xffffff8008602480 <schedule+48>:       mov     w0, #0x0                        // #0
```

Extra dis options:

```
 -l | -u | -b num | address| symbol| expression| count
```

22.4.4 whatis -o <struct>

Used to check structs, can also be used for union, Macro, or kernel symbols

```
crash> whatis -o mm_struct
struct mm_struct {
    [0] struct vm_area_struct *mmap;
    [8] struct rb_root mm_rb;
   [16] u64 vmacache_seqnum;
   [24] unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
   [32] unsigned long mmap_base;
   [40] unsigned long mmap_legacy_base;
   [48] unsigned long task_size;
   [56] unsigned long highest_vm_end;
   [64] pgd_t *pgd;
   [72] atomic_t mm_users;
   [76] atomic_t mm_count;
   [80] atomic_long_t nr_ptes;
```

22.4.5 rd <virtual address>

Check value stored in <virtual address>, extra options:

```
-dDsSupxmf | -8 | -16| -32| -64| -o offset| -e addr| address| symbol |count
```

22.4.6 struct <struct> <virtual address of <struct>>

Check the value stored in the struct located at the virtual address. Use this to check arbitrary struct of certain process.

```
crash> struct mutex ffffffc00f387b98
struct mutex {
  count = {
    counter = -1
  },
  wait_lock = {
    {
      rlock = {
        raw_lock = {
          owner = 1,
          next = 1
        }
      }
    }
  },
  wait_list = {
    next = 0xffffffc095e43ac8,
    prev = 0xffffffc095e43ac8
  },
```

22.4.7 struct <struct>.member, member<struct virtual address>

used to check the value of specific member in a struct.

22.4.8 sym

Convert certain symbol to corresponding virtual address and vice versa

```
sym -l                    ### list all symbols and corresponding virtual address
```

```
sym -M                    ### list union of all module identifier
sym -m module <name>      ### print the virtual address of module <name>
sym  vaddr                ### print symbol corresponds to vaddr
sym -q <string>           ### search for symbols and virtual address containing <string>
```

```
crash> sym -l
ffffff8008080000 (t) .head.text
ffffff8008080000 (t) _head
ffffff8008080000 (T) _text
ffffff8008080800 (t) .text
ffffff8008080800 (T) __exception_text_start
ffffff8008080800 (T) _stext
ffffff8008080800 (T) do_undefinstr
ffffff8008080a50 (T) do_sysinstr
ffffff8008080ac0 (T) do_mem_abort
ffffff8008080b60 (T) do_el0_irq_bp_hardening
ffffff8008080b68 (T) do_el0_ia_bp_hardening
ffffff8008080b88 (T) do_sp_pc_abort
ffffff8008080c68 (T) do_debug_exception
ffffff8008080d28 (t) gic_handle_irq
ffffff8008080dd0 (t) gic_handle_irq
```

22.4.9 task [-R member][, member] [pid]

Print the content of task_struct of process [pid]

22.4.10 timer

Prints timer info

22.4.11 kmem -i

Check memory usage info

22.4.12 l *<virtual address>

Check the code at specified address

22.5 Attachment: frequently used crash command

Command	Description	Example
*	Shortcut for pointers, used to replace struct/union	*page 0xc02943c0: print the page struct located at 0xc02943c0
files	Print all opened files	files 462: print info of all opened files of process 462
mach	Display machine related parameters	mach: prints CPU model, number of cores, size of memory etc.
sys	Prints special system parameter	sys config: prints status of CONFIG_xxx
timer	No options. Prints timer queue info according to time	timer: prints detailed timer info
mod	Prints details of loaded modules	mod: Prints info of all loaded modules
runq	Prints runqueue info	runq: prints all tasks in runqueue
tree	Display radix tree/RB tree struct	tree -t rbtree -o vmap_area.rb_node vmap_area_root: Prints all RBtree vmap_area.rb_node node address
fuser	Prints tasks using specified file/socket	fuser /usr/lib/libkfm.so.2.0.0: prints all process using /usr/lib/libkfm.so.2.0.0
mount	Prints all mounted file systems	mount: prints all mounted file systems
ipcs	Prints System V IPC messages	ipcs: Prints System V IPC messges
ps	Prints process status	ps: similar to ps in Linux Kernel
struct	Prints content of the sturct	struct vm_area_struct c1e44f10: prints contents of cle44f10
union	Prints content of union, similar to struct	union bdflush_param: prints content of bdflush_param

waitq	Prints all tasks in wait queue. Use options to specify the name of queue, address etc.	waitq buffer_wait: prints buffer_wait
irq	Prints info of interrupt <num>	irq 18: prints info of interrupt 18
list	Prints the content of linked list	list task_struct.p_pptr c169a000: prints the p_pptr linked list in the task located at c169a000
log	Prints kernel log in ascending time	log -m: prints kernel log
dev	Prints device related resources, including bus usage, memory usage and PCI device data	dev: prints char/block device related info
sig	prints one or more tasks related signal-handling info	sig 8970: prints process 8970 signal-handling info
task	prints specified content or the content of task_struct of the specified process	task -x: prints content of task_struct of current process
swap	No options, prints configured swap device info	swap: swap device info
search	search value within specified range of Users, kernel virtual address, or physical memory	search -u deadbeef: search 0xdeadbeef within user memory space
bt	Prints stack info	bt: prints current stack
net	Prints network related info	net: prints list of network device
vm	Prints the basic virtual address info of given task	vm: similar to /proc/self/maps in Linux kernel
btop	Convert a hexadecimal address to its page number	N/A
ptob	Reverse btop, convert page number to its hexadecimal address	N/A
vtop	prints the physical memory corresponds to the specified user/kernel virtual address	N/A
ptov	Reverse vtop, converts physical memory to virtual address	N/A

pte	Convert hexadecimal page table entry to physical page address and offset	N/A
alias	Prints or establish an alias for a command	alias kp kmem -p: kp is equivalent to kmem -p
foreach	Command iteration	foreach bt: prints stacks for each process
repeat	Loop command	repeat -1 p jiffies: execute p jiffies for every 1s
ascii	Convert hexadecimal string convert to ascii string	ascii 62696c2f7273752f results in: /usr/lib
set	Set the content for prints. Content is ustally grouped by process. Also used to set internel variables in crash	set -p: switch to the context of crashed process
p	Abbreviate for print. Print the value of the following expression. Expression can be variable or structs	N/A
dis	Abbreviate of disassemble. Convert a command or function to assembly code	dis sys_signal: disassemble function sys_signal
whatis	Search info of data/type	whatis linux_binfmt: prints struct linux_binfmt
eval	Evaluate the value of expression, and prints the result in hexadecimal, decimal, octal, decimal or binary format	N/A
kmem	Prints the memory status of current kernel	kmem -i: prints kernel memory usage
sym	Prints the virtual address of the symbol or the symbol at the virtual address	sym jiffies: prints the address of jiffies
rd	Prints the content at the specified location of memory in hexadecimal	rd -a linux_banner: prints the content of linux_banner
wr	Write to memory based on options. When trying to locate the position of system fault, wr is usually avoided	my_debug_flag 1: Change the value of my_debug_flag to 1
gdb	Execute gdb command	gdb help: execute help from gdb
extend	Dynamically load/unload extra dynamically linked libraries for crash	N/A
q	terminate	N/A

exit	same as q, terminate	N/A
help	help command	N/A

23 DDR Debug Guide

23.1 Boot Log DDR Message Details

X3J3 system booting is divided into 5 stages:

```
BootROM –> SPL -> Uboot -> Kernel -> Userpsace,
```

DDR detection and parameter configuration is done in SPL phase. SPL image is precompiled and close source.

If Boot failed and hung at SPL stage, please check the SPL boot log for the exact hung location. For example, if DDR autodetection failed, the last print will be "DDR auto detect". Furthermore, the frequency and capacity detection also needs to be check for integrity, below is a sample boot log:

```
U-Boot SPL 2018.09-00641-g1a2d98dbb5 (Sep 01 2020 - 12:48:23 +0800)
SPACC Init - ID: (00000061)
SPL emmc boot mode (from strap)
emmc: width = 4, mclk = 48000000, sclk = 47500000
SPL cold boot flow
check_reset_state (0): wakeup address = 0x0, hw wakeup status = 0x0,
wakeup src=0x100
***** using efuse ddr type ******
ddr_type = 1 (LPDDR4)        ← Indicates DDR Type is LPDDR4
ddr_hdr.ecc_gran: 0x0
ddr_hdr.ecc_map: 0x0         ← Indicates ecc enable, 0: disable
***** using efuse vendor type *****
***************** ddr auto detect **************  ← DDR auto detection
......
board id = 00
ddr daterate: 2666           ← DDR frequency 2666 detected
### HYNIX DDR ###
....
executing 2D fw
hobot_snps_scrub_test: 464
Set QoS.
SPL cold boot flow
check_reset_state (0): wakeup address = 0x0, hw wakeup status = 0x0,
wakeup src=0x100
total_sz = 2048              ← DDR capacity 2GB detected
load u-boot: src_addr=0x184c00, dest_addr=0x4000000, len=876032
```

23.2 DDR Memory Malfunction Detection and Analysis

23.2.1 Hung at Boot

When DDR memory parameter is critically mismatching with the board, boot might hung at SPL:

```
p_dmem_lpddr4_2d_16b->MR11_A0:63
p_dmem_lpddr4_2d_16b->MR14_A0:2e
p_dmem_lpddr4_2d_16b->MR12_A0:2e
p_dmem_lpddr4_2d_16b->MR22_A0:4

p_dmem_lpddr4_2d_16b->MR3_A0:b3
p_dmem_lpddr4_2d_16b->Reserved00:40
p_dmem_lpddr4_2d_16b->Reserved0E:4
p_dmem_lpddr4_2d_16b->Delay_Weight2D:20
p_dmem_lpddr4_2d_16b->Voltage_Weight2D:80
Controller:
DRAMTMG2 :9121219
RFSHTMG  :82090095
RFSHTMG1 :610000
**************************************************

executing 1D fw
DWC_DDRPHYA_DBYTE0__DFIMRL_p0=6
DWC_DDRPHYA_DBYTE1__DFIMRL_p0=6
DWC_DDRPHYA_DBYTE2__DFIMRL_p0=6
DWC_DDRPHYA_DBYTE3__DFIMRL_p0=6
executing 2D fw
Set QoS.
SPL cold boot flow
check_reset_state (0): wakeup address = 0x0, hw wakeup status = 0x0, wakeup src=0x100
scomp_dram_detect ddr : 0
```

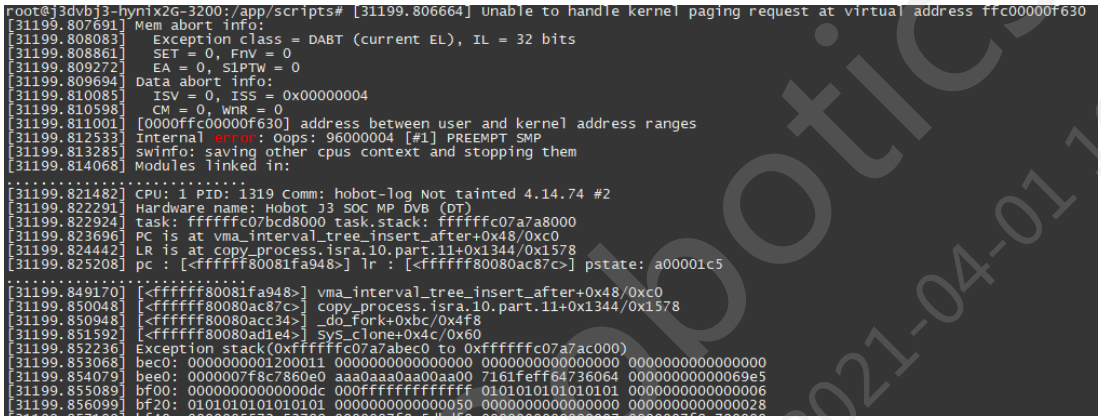
Or SPL stage might pass but random reset happens in UBoot might also be the result of DDR memory malfunction, as show below:

```
U-Boot 2018.09 (Jul 06 2020 - 08:10:52 +0800), Build: jenkins-xj3_autobuild-408

Model: Hobot XJ3 Soc Board
DRAM:  system DDR size: 0x37e00000
894 MiB
MMC:   "Synchronous Abort" handler, esr 0x96000210
elr: 0000000040876e4 lr : 0000000004032ad0 (reloc)
elr: 0000000040876e4 lr : 0000000004032ad0
x0 : 0000000004fe35b0 x1 : 0000000004fdb180
x2 : 0000000000000200 x3 : 0000000000000000
x4 : 0000000000000200 x5 : 0000000000000040
x6 : 0000000004fdb180 x7 : 0000000080000010
x8 : 0000000080000014 x9 : 0000000000000004
x10: 00000000000000ec x11: 0000000004fdb15c
x12: 00000000000000d4 x13: 0000000000000000
x14: 0000000040b6fd0 x15: 0000000004fdb5ac
x16: 0000000000000000 x17: 0000000000000000
x18: 0000000004fdb670 x19: 0000000004fdb180
x20: 0000000004fe0ac0 x21: 0000000000000000
x22: 0000000004fe09f0 x23: 0000000004fe0aa0
x24: 0000000004fe3590 x25: 0000000004fe34f0
x26: 0000000000000000 x27: 0000000000000000
x28: 0000000000000000 x29: 0000000004fdb140

Resetting CPU ...

resetting ...
NOTICE:  fast_boot:0
```

If hung or abnormities happens in the early stages of boot, check if the DDR parameter is properly selected and configured, or check if DDR memory hardware is functioning normally.

23.2.2 Kernel Random Panic

Even after DDR memory is operating relatively steadily under room temperature, during stress test/high and low temperature test, there is possibility that random Kernel panic might happen. Each time the panic code location would be different and usually resides in general Kernel codes. In these scenarios, the root cause is most likely DDR memory instability causing software memory corruption. For example, in the below picture, the panic address is "ffc00000f630" but normal ARM64 Kernel Code address starts with "ffffff":

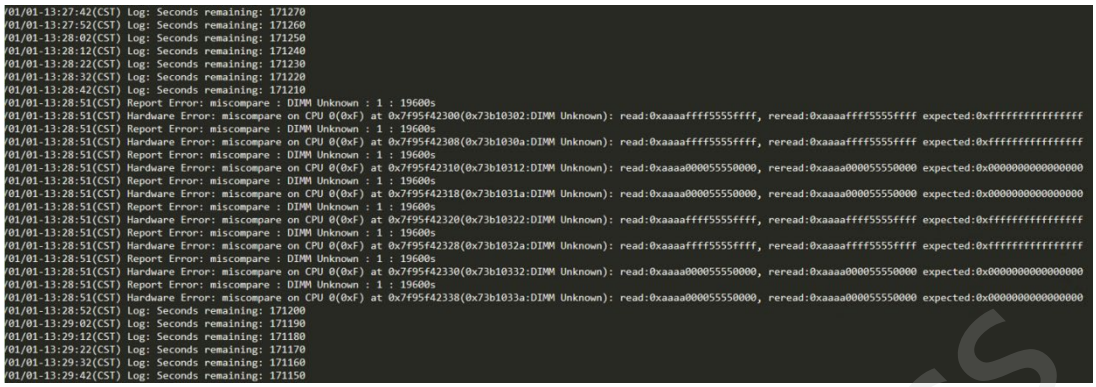


Another example, below, the highest 24 bit changed from "ffffff" to "fff7ffb" indicating bitflips at bit52 and bit59, which is not rare during high/low temperature tests.

```
[  458.685616] Unable to handle kernel paging request at virtual address fff7ffbf00041ba0
[  458.752133] PC is at find_get_entries+0x6c/0x218
[  458.756776] LR is at find_get_entries+0x178/0x218
[  458.761503] pc : [<ffffff8008155124>] lr : [<ffffff8008155230>] pstate: 40000145
[  458.768937] sp : ffffffc03ca1fac0
[  458.772265] x29: ffffffc03ca1fac0 x28: 0000000000000000
[  458.777606] x27: 000000000000000d x26: ffffffc03ca8c1c8
[  458.782947] x25: ffffffc03ca1fba0 x24: ffffffc03ca1fba0
[  458.788288] x23: ffffffc03ca1fc20 x22: 000000000000000e
[  458.793627] x21: 0000000000000003 x20: ffffffc03d342140
[  458.798967] x19: ffffffc03ca8c1d0 x18: 00000000103929a0
[  458.804307] x17: 0000007f8ea01e88 x16: 0000000044e488
[  458.809647] x15: 0000000000000016 x14: 0000000000000000
[  458.814988] x13: 0000000000000228 x12: 0000000000000040
[  458.820327] x11: 0000000000000000 x10: 0000000000000000
[  458.825667] x9 : fff7ffbf00041b80 x8 : 0000000000000460
[  458.831007] x7 : 0000000000000002 x6 : 0000000000000002
[  458.836347] x5 : ffffffbf00041b5c x4 : 0000000000000020
[  458.841686] x3 : 0000000000000000 x2 : fff7ffbf00041b80
[  458.847025] x1 : 0000000000000463 x0 : 0000000000000000
```

23.2.3 Stressapptest Memory Stress Test Errors

DDR memory Stability Stress test often make use of "stressapptest" for memory stress tests. When there is a board highly suspected DDR memory malfunctioning, using "stressapptest" to stress test DDR memory is often a good idea. Take the below example, "stressapptest" detected memory miscompare error, which is usually a result of unstable DDR memory:



"stressapptest" is an open source tool which can be used to do memory, CPU and Storage stress tests.

X3J3 comes with a precompiled "stressapptest" located in UT image /app/bin/stressapptest. If the image is not UT enabled, it can also be found in source code: unittest/prebuilts/stressapptest

"stressapptest" can also be downloaded and cross-compiled from github: https://github.com/stressapptest/stressapptest

Commonly used test command:

```
/app/bin/stressapptest -M 128M -s 3600 -m 8 -i 8 -C 8 -l /userdata/x3_stressapptest.log --
cc_test
```

Messages similar to below should appear on console:

```
root@x3dvbx3-hynix1G-2666:~# /app/bin/stressapptest -M 128M -s 3600 -m 8
-i 8 -C 8 -l /userdata/x3_stressapptest.log --cc_test


…
1970/01/01-10:49:42(CST) Log: Seconds remaining: 3550
1970/01/01-10:49:52(CST) Log: Seconds remaining: 3540
1970/01/01-10:50:02(CST) Log: Seconds remaining: 3530
…
1970/01/01-10:50:28(CST) Stats: Completed: 318004.00M in 96.29s 3302.60MB/s, with 0
hardware incidents, 0 errors
1970/01/01-10:50:28(CST) Stats: Memory Copy: 231252.00M at 2471.39MB/s
1970/01/01-10:50:28(CST) Stats: Invert Data: 86752.00M at 927.11MB/s
1970/01/01-10:50:28(CST)
1970/01/01-10:50:28(CST) Status: PASS - please verify no corrected errors
```

23.2.4 ECC Error Detection

X3J3 also supports DDR memory ECC error detection and correction. After ECC is enabled, if ECC errors are detected, DDR memory can be considered unstable as well.

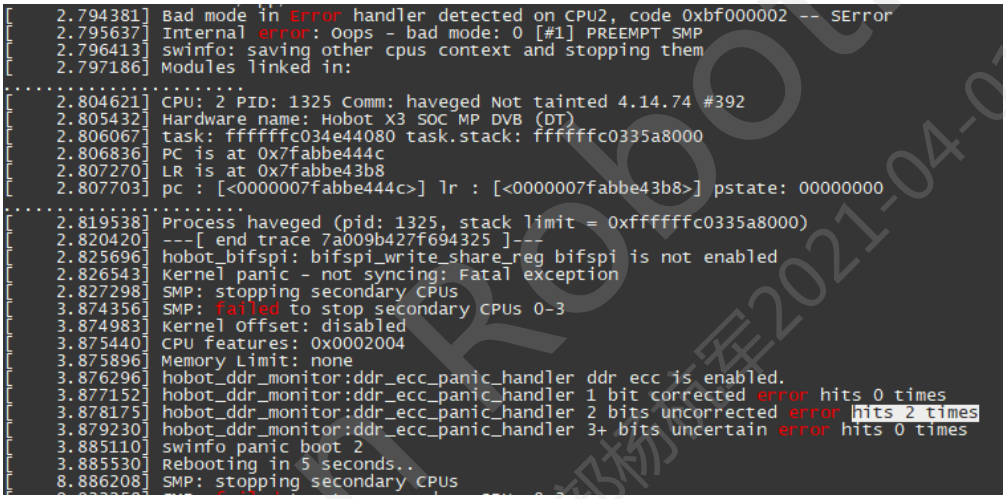
Use the following sysfs node to check if there are 1bit, 2bits or >=3bits flips happened:

```
cat /sys/devices/platform/soc/a2d10000.ddr_monitor/ddr_ecc_stat
```

The following message should appear on console:

```
cat /sys/devices/platform/soc/a2d10000.ddr_monitor/ddr_ecc_stat
ddr ecc is enabled
1 bit corrected error hits 512 times
2 bits uncorrected error hits 16392 times
3+ bits uncertain error hits 272 times
```

After ECC protection is enabled, if there is a Kernel Panic, panic log will also prints ecc stats as shown in the below example, 2 times 2bits flip happened:



ECC protection can be enabled through the below command, effective after reboot:

```
hrut_ddr_ecc s on

reboot
```