Document Number: ANnnnn
Rev. 4, 2017.10

i.MX6UL/ULL EVK 扩展串口板移植手册

by i.MX CAS

GSM CAS
NXP Semiconductor, Inc.
Shanghai

本文为 NXP i.MX CAS Team 设计的 i.MX6UL/ULL EVK 板的扩展串口板的 BSP 移植说明，以此展示如何将 NXP 标准发布的基于 i.MX6UL/ULL EVK 板的 BSP 移植到某一目标板上。

本文增加串口，GPIO_LED，GPIO_KEY，PWM，ADC 驱动是 i.MX6UL/ULL 本身支持的，PCF8591 ADC 和 CH438 是外部连接设备驱动，以展示如何增加和修改各种驱动。

version	history	owner
V1	• 创建此文档	JohnLi
V2	• 增加 i.MX6ULL 支持	JohnLi
V3	• 增加 PCA9555 支持	JohnLi
V4	• 增加 PCT2075/PCF8563/PCA9632 支持(rework)	JohnLi

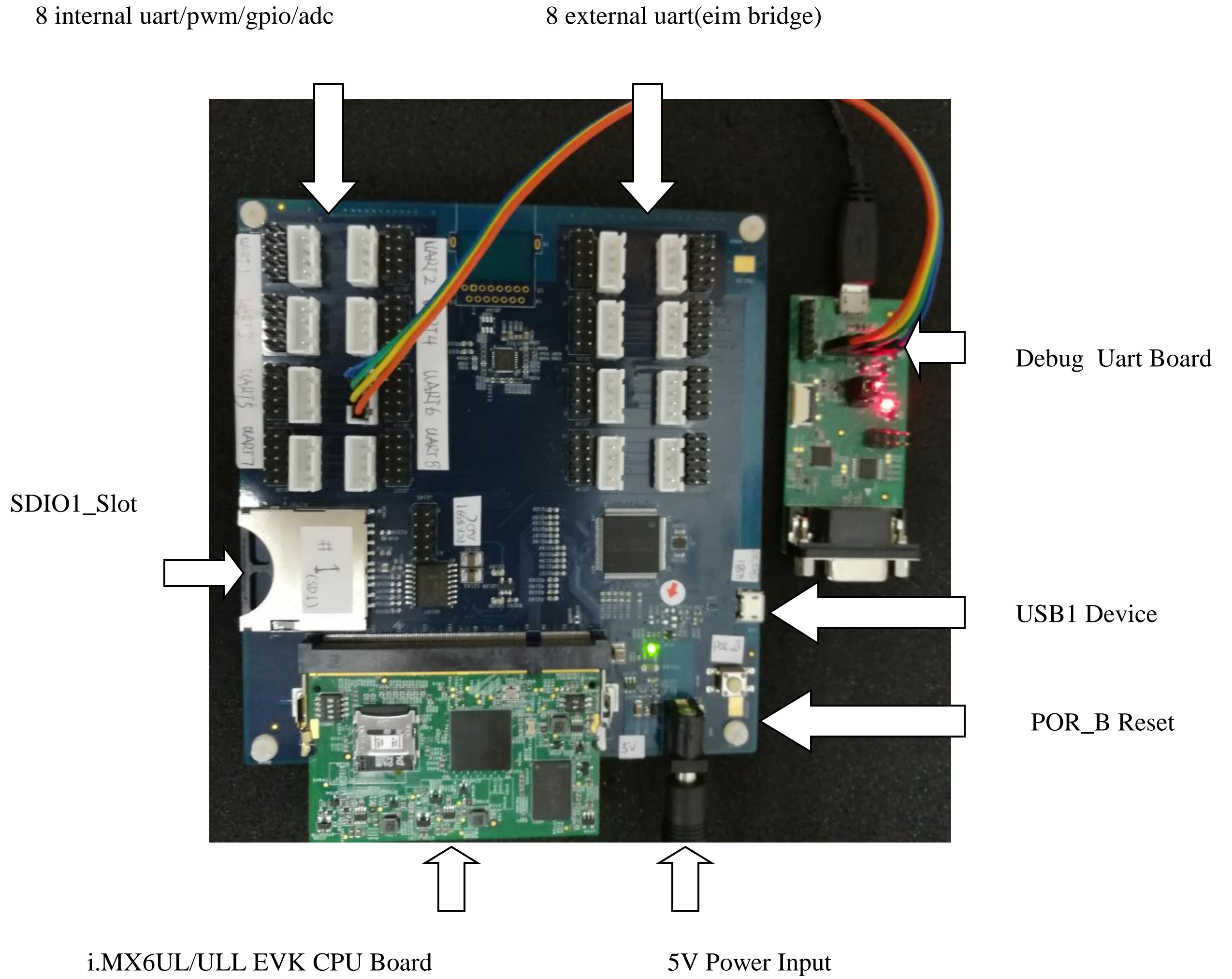
Contents

1 硬件设计说明 2
硬件框图 2
硬件模块设计 4
IOMUX 表 8
2 编译环境搭建 8
编译环境文档及镜像下载。 8
编译环境搭建 11
3 移植 BSP 到扩展串口板 15
Uboot 中支持新的 DTB 15
Uboot 中调试串口改成 UART6 16
去除掉无用的驱动及其 IOMUX 18
增加 i.MX6UL/ULL 本身串口支持 18
增加 GPIO 输出支持（GPIO_LED） 26
增加 GPIO 输入支持（GPIO_KEY） 30
增加 PWM 支持 34
增加 i.MX6UL 本身 ADC 支持 38
修改网口驱动仅支持一个网口 41
增加 NXP PCF8591 I2C 转 ADC 芯片支持 44
增加 NXP PCA9555A I2C 转 GPIO 芯片支持(rework 支持) 47
增加 NXP PCT2075 I2C 温度传感器芯片支持(rework 支持) 55
增加 NXP PCF8563 I2C RTC 支持(rework 支持) 58
增加 NXP PCA9632 I2C LED 控制器芯片支持(rework 支持) 65
增加 CH438 EIM 转串口芯片支持(delay) 70

1 硬件设计说明

硬件框图

i.MX6UL/ULL EVK 板的扩展串口板实际硬件框图如下：



Debug 串口板的配置如下：

Connection	Jumper Setting	
UART4-USB UART1-DB9	J110:1-3,2-4 J109:1-3,2-4	Default
USB-DB9	J110:3-5,4-6 J109:3-5,4-6	

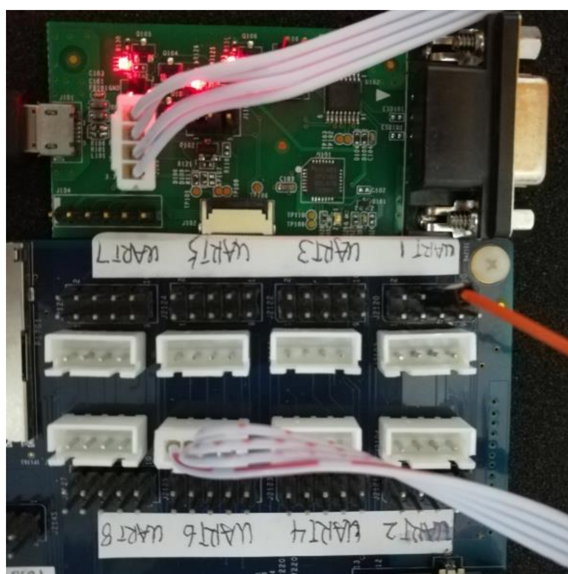
Debug 串口板默认连接方式是从 UART4 到 USB，所以我们从 i.MX6UL/ULL EVK 板的扩展串口板的 UART6，（UART6 只设计 TX/RX，所以做为 Debug 串口），连接到串口板的 J105 上的 UART4 上。

连接方式为：

- i.MX6UL/ULL EVK 板的扩展串口板.J2109.VPERI_3V3(PIN 1)<->串口板.J105.3.3V(PIN6)
- i.MX6UL/ULL EVK 板的扩展串口板.J2109.TXD_UART6(PIN 2)<->串口板.J105.TX4(PIN5)
- i.MX6UL/ULL EVK 板的扩展串口板.J2109. RXD_UART6 (PIN 3)<->串口板.J105.RX4V(PIN4)
- i.MX6UL/ULL EVK 板的扩展串口板.J2109.GND(PIN 4)<->串口板.J105.GND(PIN1)

如果是使用排线连接，则

- i.MX6UL/ULL EVK 板的扩展串口板.J2109.GND(PIN 4)<->串口板.J105.TX1(PIN3)，也可以使用，电源可以作为信号参考：



硬件模块设计

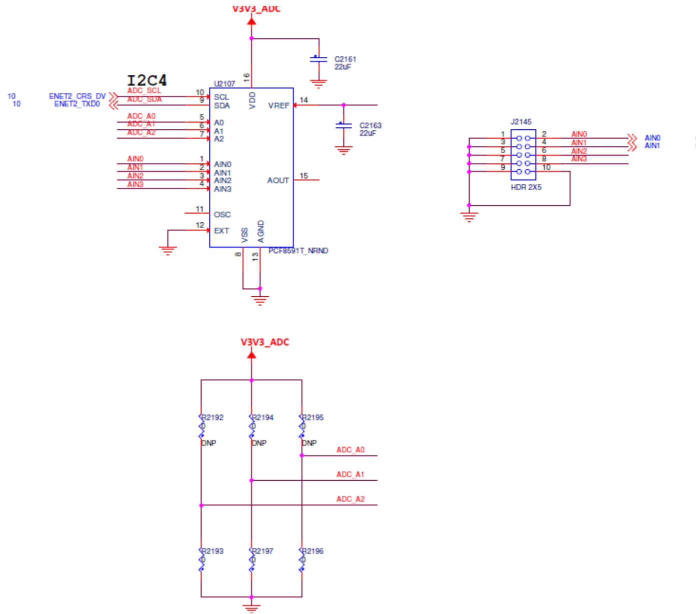
1：PCF8591 I2C to ADC

PCF8591 为 NXP 的 8-bit A/D, D/A 转换器，芯片说明如下：

http://www.nxp.com/products/interfaces/ic-bus-portfolio/ic-dacs-and-adcs/8-bit-a-d-and-d-a-converter:PCF8591

i.MX6UL/ULL EVK 扩展串口板设计中使用 ADC_A0~3 配置 I2C 地址 pcf i2c slave address is [1][0][0][1][a2=0][a1=0][a0=0][rw]=0x48，使用 AIN0~1 两个脚的模拟输入。

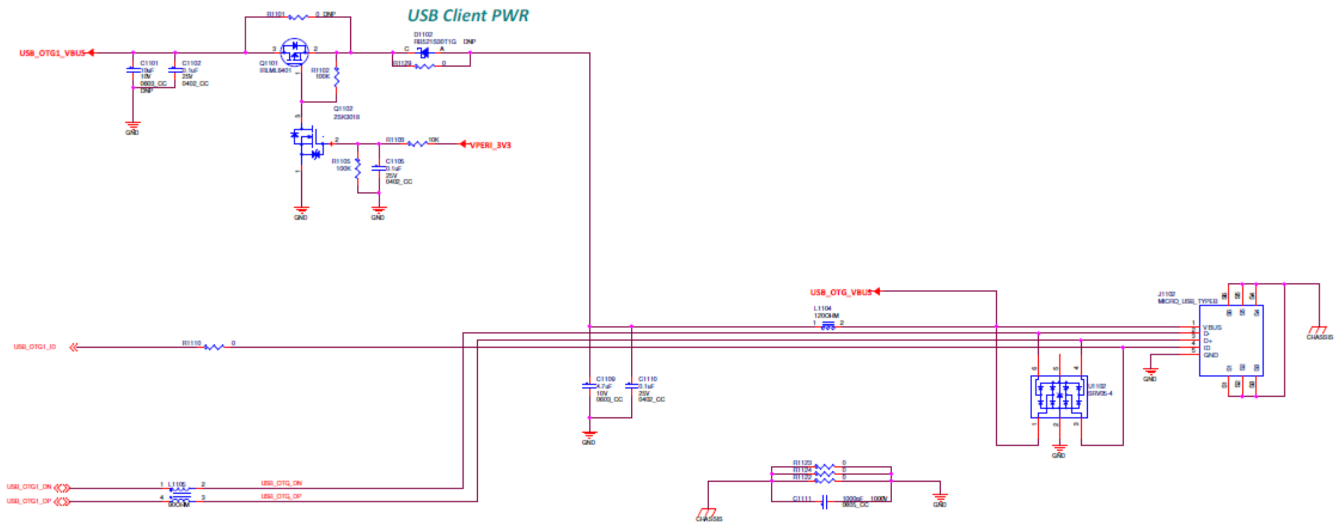
I2C 连接到 I2C4 上。



设计中遗忘了 I2C 上拉电阻和 VREF 连接 3.3V 电压，rework 如下：

2：USB device

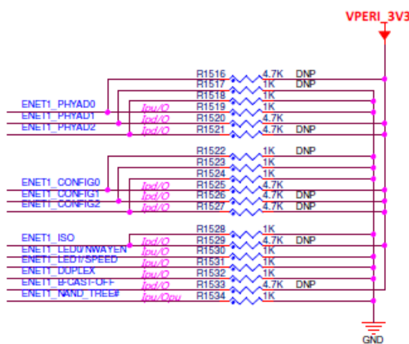
i.MX6UL/ULL EVK 扩展串口板上的 USB OTG 口仅设计为 device 使用，用于烧写镜像，所以 Vbus 5V 设计为从 USB OTG Vbus 输入给 i.MX6UL/ULL，中间设计了 MOS 管隔离来防止拔插时的电源 glitch：



3： 1 100M Ethernet

多数工业串口网关应用使用串口来读取相关控制数据，然后通过网络上传到上位机，所以 i.MX6UL/ULL EVK 扩展串口板设计了一个 100M 以太网口，i.MX6UL/ULL 本身支持两个 100M 网口，这儿只使用 ENET1。

MDIO 接口使用 GPIO_6/7，RMII CFG 如下：



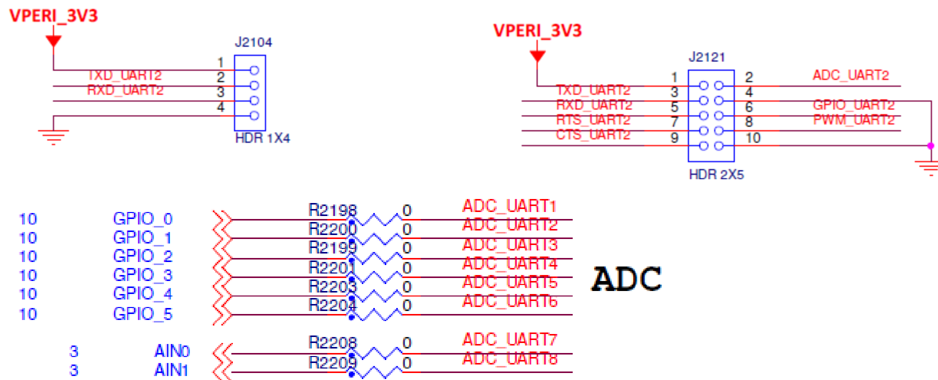
# CFG	Description
PHYAD[2:0]	PHY ADDR 00-XXX (00001 DEFAULT)
CONFIG[2:0]	IF MODE 001 RMII 101 RMII Back-to-Back xxx Reserved-not used
ISO	ISOLATE mode Pull-up = Enable Pull-down (defaule) = Disable
SPEED	SPEED mode Pull-up (defaule) = 100Mbps Pull-down = 10Mbps

# CFG	Description
DUPLEX	DUPLEX mode Pull-up (defaule) = Half Duplex Pull-down = Full Duplex
NWAYEN	Nway Auto-Negotiation Pull-up (defaule) = Enable Pull-down = Disable
B_CAST_OFF	Broadcast Off - for PHY Address 0 Pull-up = PHY Address 0 set as unique PHY addr Pull-down (default) = PHY Address 0 set as broadcast PHY addr
NAND_TREE#	NAND Tree Mode Pull-up (defaule) = Disable Pull-down = Enable

PHY 地址=2。

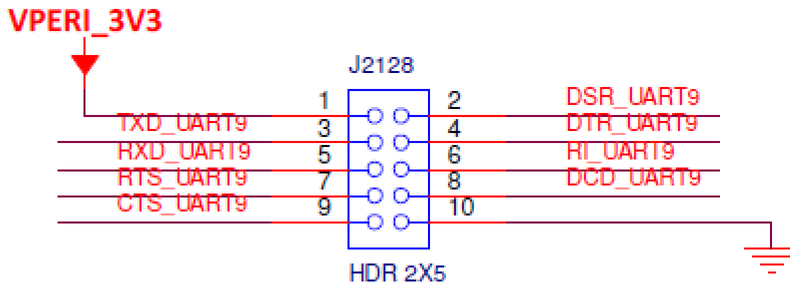
4: i.MX6UL/ULL UART/PWM/GPIO/ADC

i.MX6UL/ULL 本身支持 8 个串口，8 个 PWM 和 2X10 个 ADC pin。i.MX6UL/ULL EVK 扩展串口板设计了两类 8 个串口接口，一种标准 UART TX/RX 接口，和包含了一个四线 UART(增加 RTS/CTS)，一个 PWM，一个 GPIO，一个 ADC 的接口，其中 6 个串口接口使用 i.MX6UL/ULL ADC channel 1 input0~5。另两个使用 PCF8591ADC：



5：EIM to UART bridge

i.MX6UL/ULL EVK 扩展串口板设计中，为了扩展更多串口，使用了 CH438 EIM to UART bridge 扩展出 8 个标准 9 线串口，以便以后更多的扩展：

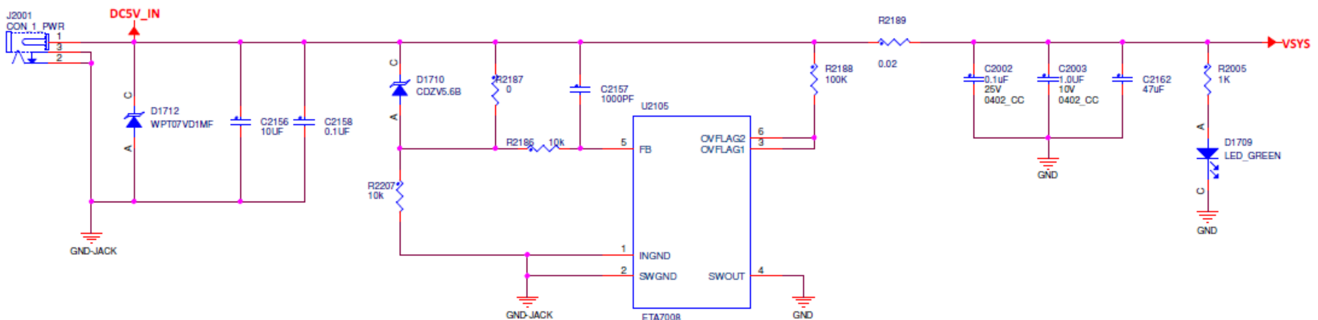


6:　SD 插槽

i.MX6UL/ULL EVK 扩展串口板设计时考虑有可能使用 SDIO WiFi 连接网络，设计了 SDIO1 插槽可连接 WiFi 模块。

7：System Power

系统电源设计上设计了 ETA7008 OVP 保护器件以及电源灯：

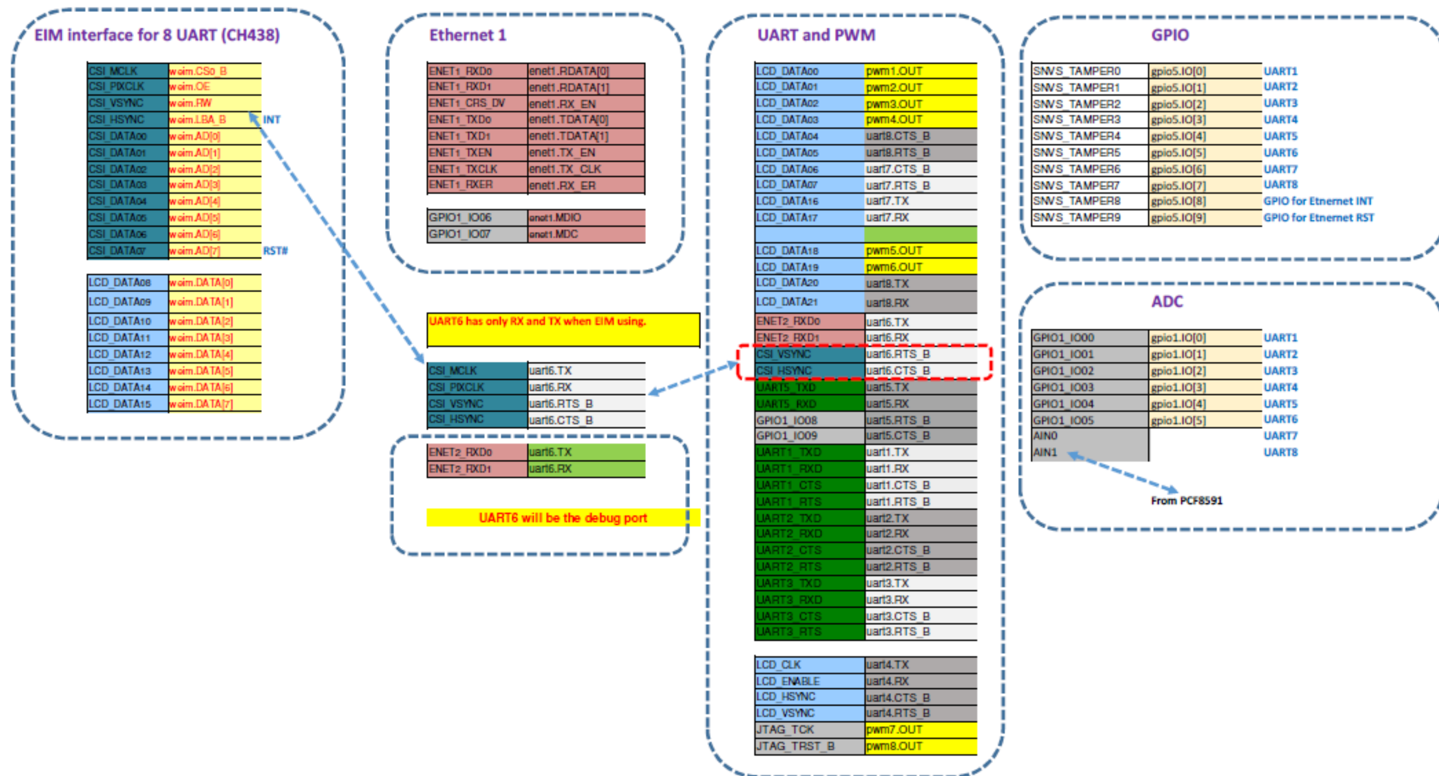


8：Connector to i.MX6UL/ULL EVK CPU board

与 i.MX6UL/ULL EVK 底板类似，也设计了连接器连接 i.MX6UL/ULL EVK CPU 板。

IOMUX 表

IOMUX 表修改过的地方 如下：



2 编译环境搭建

编译环境文档及镜像下载。

本文使用 Yocto Linux L4.1.15_2.0.3 BSP。这是一个基于 L4.1.15_2.0.0 的 Patch release。2.0.3 与 2.0.0 相比的区别如下：

Linux L4.1.15_2.0.1 Patch release: fixed the i.MX 6UL/ULL new Ethernet PHY ID issue and i.MX 6UL/6ULL/7D Audio Codec WM8960 issue.

Linux L4.1.15_2.0.2 Patch release: beside 2.0.1, add i.MX 6ULL@800MHz support.

Linux L4.1.15_2.0.3 Patch release: beside 2.0.1, 2.0.2 add i.MX 6ULL@900MHz support.

所以，需要注意的是，根据：
http://git.freescale.com/git/cgit.cgi/imx/fsl-arm-yocto-bsp.git/tree/ChangeLog?h=imx-4.1-krogoth
Date 5/16/2017

4.1.15-2.1.0 - manifest imx-4.1.15-2.1.0.xml
- GA release for i.MX 6SLL
- Changes in Kernel and U-Boot
- Multimedia upgrade to 4.1.6

Date 3/29/2017
4.1.15-2.0.3 - manifest imx-4.1.15-2.0.3.xml
- Patch release for i.MX 6ULL 900MHz
- Patch release only kernel changes on the meta-fsl-bps-release/imx/meta-bsp/recipes-kernel/linux
- This patch release is just a Yocto Project release layer and manifest update

Date 2/16/2017
4.1.15-2.0.2 - manifest imx-4.1.15-2.0.2.xml
- Patch release for i.MX 6ULL 800MHz
- This patch release is just a Yocto Project release layer and manifest update

…

Date 11/16/2016
4.1.15-2.0.1 - manifest imx-4.1.15-2.0.1.xml
- Patch release only kernel changes on the meta-fsl-bps-release/imx/meta-bsp/recipes-kernel/linux
- Patches are fox WM8960 and KSZ8081
- Note the kernel patches are not provided on linux-imx repo but will be provided in next release
- This patch release is just a Yocto Project release layer and manifest update

Date 10/1/2016
4.1.15-2.0.0 - manifest imx-4.1.15-2.0.0.xml
- GA release for i.MX 6ULL, i.MX 6Quad/Dual/Plus/Solo/SoloX/SoloLite/Ultralite and i.MX 7Dual
- Upgrade U-Boot to 2016.03
- Upgrade Gstreamer to 1.8.1 and multimedia upgrade to 4.1.4
- Upgrade to Krogoth Yocto Project release with GCC 5.3
- Kernel and BSP library fixes

- 如果在 2016_11_16 之前，使用命令 repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth 。将会拿到 2.0.0 版本，需要打上 2.0.3 的 patch。
- 在 2016_11_16~2017_3_29 号之间使用命令 repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth。将会拿到 2.0.1 或 2.0.2 版本，需要打上 2.0.3 的 patch。
- 在 2017_3_29 号之后使用命令 repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth。将会直接拿到大于 2.0.3 的版本。
- 根据 http://git.freescale.com/git/cgit.cgi/imx/fsl-arm-yocto-bsp.git/tree/README?h=imx-4.1-krogoth 文档说明，可以使用以下命令拿到需要的准确的版本：

```
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth [-m <manifest>]
```

To download the 4.1.15-2.0.0_ga release

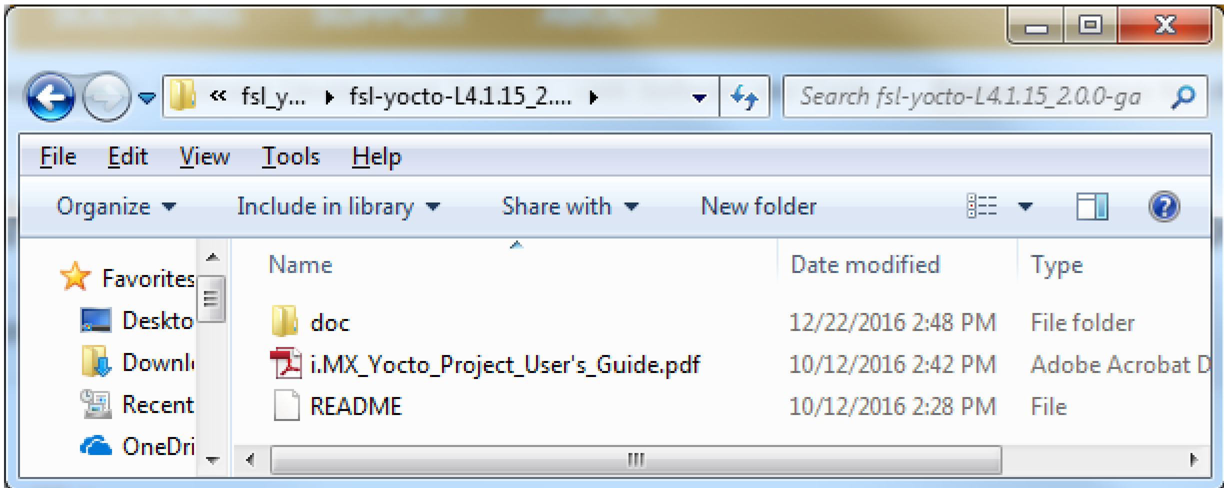
```
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth -m imx-4.1.15-2.0.0.xml
```

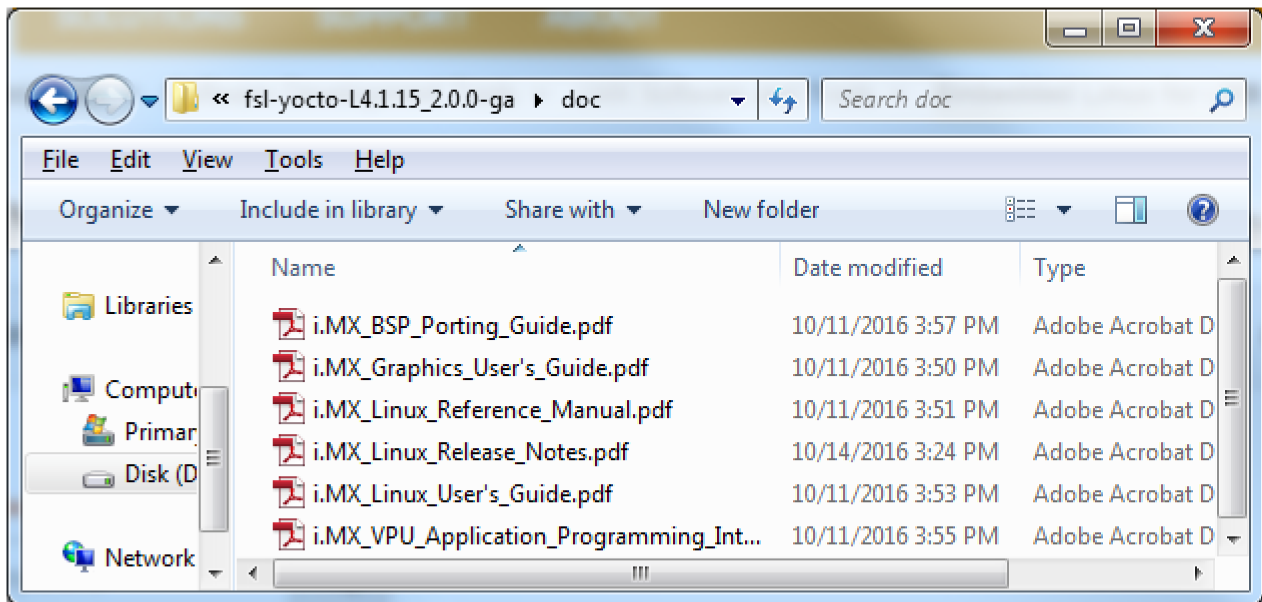
To download the 4.1.15-2.0.3 patch release

```
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth -m imx-4.1.15-2.0.3.xml
```

编译环境搭建文档下载地址为：www.nxp.com/imx -> Software and Tools -> Linux -> DOCUMENTATION -> L4.1.15_2.0.0_LINUX_DOCS。

下载下来的文档包目录如下：





搭建编译环境需要的文档包括<<i.MX_Yocto_Project_User's_Guide.pdf>>和<<i.MX_Linux_User's_Guide.pdf>>。

Linux L4.1.15_2.0.3 Demo Image 下载地址为：www.nxp.com/imx -> Software and Tools -> Linux -> DOWNLOADS-> Board Support Packages-> More-> L4.1.15_2.0.3_iMX6UL

编译环境搭建

根据前一章文档说明，如下搭建 Linux L4.1.15_2.0.3 的编译环境，在移植 BSP 的过程中，我们需要获得

包含编译器的 SDK

Uboot 源代码

Kernel 源代码

包含 rootfs 的镜像

然后我们在修改了 uboot 及 kernel 源代码后就可以直接 standalone 的编译，替换镜像。

1: host build ubuntu 64bit 14.04

```
sudo apt-get update
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libsdl1.2-dev
sudo apt-get install libsdl1.2-dev xterm sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 help2man make gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl lzop asciidoc
sudo apt-get install u-boot-tools
```

2: got bsp

```
mkdir bin
//repo can not download from google website, change another link
curl http://raw.githubusercontent.com/android/tools_repo/stable/repo > ./bin/repo
chmod a+x ./bin/repo
export PHTH=~/mx6ul/bin/:$PATH
git config --global user.name "johnli"
git config --global user.name "r64710@freeescale.com"
git config --global --list
    user.name=r64710@freeescale.com
    user.email=vmuser@vmuser.com
mkdir fsl-release-bsp
cd fsl-release-bsp/
//repo can not use the google website, change another link
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1-krogoth
--repo-url=http://github.com/android/tools_repo
repo sync
```

前文已经说明了此命令会获得基于 L4.1.15 的最新的 BSP，2017_3_29 号之后获得的为大于 2.0.3 的版本，本文也会说明如何基于 2.0.0 打上 2.0.3 的 Patch。

3:compiling

```
DISTRO=fsl-imx-fb MACHINE=imx6ulevk source fsl-setup-release.sh -b imx6ulevk_fb
bitbake fsl-image-machine-test
//need a long time
```

4:compiling and install sdk

```
bitbake fsl-image-machine-test -c populate_sdk
//need a long time
cd tmp/deploy/sdk/
./fsl-imx-fb-glibc-x86_64-fsl-image-machine-test-cortexa7hf-neon-toolchain-4.1.15-2.0.0.sh
//press enter and Y, and install to your own folder
```

(比如安装到/home/vmuser/imx6ulullexternaluart/sdk)

5:打上 2.0.3 patch，如果我们只有基于 L4.1.15_2.0.0 的 Yocto 环境，需要重新下载下基于 L4.1.15_2.0.3 的 Yocto 环境，然后根据之中的
sources/meta-fsl-bsp-release/imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2016.03.bb
sources/meta-fsl-bsp-release/imx/meta-bsp/recipes-kernel/linux/linux-imx_4.1.15.bb 说明

从同级目录下可以拿到 patch，我们这儿假定已经有 patch 的情况下如何在 L4.1.15_2.0.0 的源代码上打上。(如果已经是 L4.1.15_2.0.3 的 Yocto 环境，则不需要执行此步)。

Uboot:

```
pwd
~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/u-boot-imx/2016.03-r0/git
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/uboot/0001-MLK-13602-1-mx6-Add-i.MX6ULL-fused-modules-checking-.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/uboot/0002-MLK-13602-2-epdc-Add-disable-fuse-checking.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/uboot/0003-MLK-13602-3-mx6ullevk-arm2-Enable-module-fuse-checki.patch
```

Kernel:

```
pwd
/home/ysli/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/linux-imx/4.1.15-r0/git
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0001-ARM-imx-imx6ul-add-PHY-KSZ8081-new-silicon-revision-.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0001-MLK-13418-ASoC-wm8960-workaround-no-sound-issue-in-m.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0001-MLK-13748-ARM-dts-imx6ull-9x9-evk-ldo-add-ldo-enable.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0002-MLK-13422-ASoC-wm8960-fix-the-pitch-shift-issue-afte.patch
patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0002-MLK-13766-ARM-dts-imx6ull-9x9-evk-correct-the-xnur-p.patch
```

```
patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0003-MLK-13774-ARM-imx-fix-lpddr2-busfreq-support-on-i.mx.patch

patch -p1 < ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0004-MLK-13724-ARM-dts-fix-audio-error-log-in-kernel-boot.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0005-MLK-13601-01-ARM-dts-imx-update-the-setpoint-for-imx.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0006-MLK-13601-02-ARM-imx-Add-fuse-check-support-for-imx6.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0007-MLK-13616-ARM-imx-Add-low-power-run-voltage-change-s.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0008-MLK-14409-01-ARM-imx-Add-speed-grading-fuse-check-fo.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0009-MLK-14409-02-ARM-dts-imx-Add-900MHz-setpoint-on-i.mx.patch

patch -p1
< ../2.0.3_patch/L4.1.15_2.0.3_patch/kernel/0010-MLK-14409-03-ARM-dts-imx-Correct-the-setpoint-on-imx.patch
```

6: compiling the uboot

可以使用 bitbake 命令编译

```
pwd
~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb$
bitbake u-boot-imx -c listtasks
bitbake u-boot-imx -c compile -f
```

或者使用交叉编译器独立编译

```
source /"sdk install folder"/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
```

(比如：vmuser@ubuntu:~/imx6ulullexternaluart/sdk$ pwd

/home/vmuser/imx6ulullexternaluart/sdk

vmuser@ubuntu:~/imx6ulullexternaluart/sdk$ source environment-setup-cortexa7hf-neon-poky-linux-gnueabi)

```
pwd
~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/u-boot-imx/2016.03-r0/git
```

(如果已经拷贝出源文件目录如下：pwd

/home/vmuser/imx6ulullexternaluart/sources/u-boot-2.0.3)

```
make clean
make distclean
make mx6ul_14x14_evk_defconfig
(make mx6ull_14x14_evk_defconfig /*i.mx6ull evk config*/)
make
```

7: compiling the bsp&dtb

可以使用 bitbake 命令编译

```
pwd
~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb$
 bitbake linux-imx -c listtasks
bitbake linux-imx -c menuconfig
bitbake linux-imx -c compile -f
```

或者使用交叉编译器独立编译

```
pwd
/home/ysli/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/linux-imx/4.1.15-r0/git
 cd kernel
```

(如果已经拷贝出源文件目录如下：pwd

/home/vmuser/imx6ulullexternaluart/sources/kernal-2.0.3)

```
  make clean
  make distclean
  make imx_v7_defconfig
  make zImage
  make imx6ul-14x14-evk.dtb
  (make imx6ull-14x14-evk.dtb /*i.mx6ull evk dtb*/)
```

8: burn image

- 烧写整个 SDcard 镜像

  使用命令 cat /proc/partition 检查目前挂接盘符，然后插入 TFcard 读卡器，再次检查盘符

```
vmuser@ubuntu:~$ cat /proc/partitions
major minor  #blocks  name
…
   8       32    3872256 sdc
   8       33       8192 sdc1
   8       34     618496 sdc2
 sudo dd if=<sdcard image folder>/fsl-image-machine-test-imx6ulevk.sdcard of=/dev/sdc bs=1M &&sync.
```

(比如：sudo dd if=/mnt/hgfs/share_folder/external_uart/image/fsl-image-machine-test-imx6ulevk.sdcard of=/dev/sdc bs=1M && sync)

- 烧写 uboot 镜像(注意 uboot 镜像是 u-boot.imx 而不是 u-boot.bin)

```
sudo dd if=<uboot image folder>/u-boot.imx of=/dev/sdc bs=512 seek=2 conv=fsync
```

(比如：pwd

/home/vmuser/imx6ulullexternaluart/sources/u-boot-2.0.3

vmuser@ubuntu:~/imx6ulullexternaluart/sources/u-boot-2.0.3$ sudo dd if=u-boot.imx of=/dev/sdc bs=512 seek=2 conv=fsync)

sync

- 更新 zImage

  重新 umount/mount 一下 TFcard

  pwd

  /media/vmuser/Boot imx6ul

  cp <zImage folder>/zImage .

  (比如：cp ~/imx6ulullexternaluart/sources/kernel-2.0.3/arch/arm/boot/zImage .)

- 更新 DTB

  pwd

  /media/vmuser/Boot imx6ul

  cp <dtb folder>/imx6ul-14x14-evk.dtb .

  (cp <dtb folder>/imx6ull-14x14-evk.dtb .)

  (比如：cp ~/imx6ulullexternaluart/sources/kernel-2.0.3/arch/arm/boot/dts/imx6ull-14x14-evk.dtb .)

3 移植 BSP 到扩展串口板

Uboot 中支持新的 DTB

我们将为 i.MX6UL/ULL EVK 扩展串口板创建一个新的 DTS/DTB 文件，所以在 uboot 中需要指定加载此 DTB 文件：

pwd

~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/u-boot-imx/2016.03-r0/git

(如果已经拷贝出源文件目录如下：pwd

/home/vmuser/imx6ulullexternaluart/sources/u-boot-2.0.3)

vi include/configs/mx6ul_14x14_evk.h

(vi include/configs/mx6ullevk.h /*i.mx6ull evk config file*/)

修改 "imx6ul-14x14-evk.dtb" 为 "imx6ul-14x14-evk-externaluart.dtb"

(修改 "imx6ull-14x14-evk.dtb" 为 "imx6ull-14x14-evk-externaluart.dtb" /*i.mx6ull evk config file*/)

如下：

```
"if test $board_name = EVK && test $board_rev = 14X14; then " \
                                    "setenv fdt_file imx6ul-14x14-evk-externaluart.dtb /*imx6ul-14x14-evk.dtb*/; fi; " \
```

然后，再次编译 uboot。

之后，为 i.MX6UL/ULL EVK 扩展串口板创建一个新的 DTS 文件

```
pwd
/home/ysli/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/linux-imx/4.1.15-r0/git
(如果已经拷贝出源文件目录如下：pwd
/home/vmuser/imx6ulullexternaluart/sources/kernal-2.0.3)
```

创建一个新的 dts 文件来支持 i.MX6UL/ULL EVK 扩展串口板，之后大部分的移植工作主要是修改此文件：

```
cp arch/arm/boot/dts/imx6ul-14x14-evk.dts arch/arm/boot/dts/imx6ul-14x14-evk-externaluart.dts
(cp arch/arm/boot/dts/imx6ull-14x14-evk.dts arch/arm/boot/dts/imx6ull-14x14-evk-externaluart.dts  /*i.mx6ull evk dtb*/)
```

编译此 DTB

```
make imx6ul-14x14-evk-externaluart.dtb
(make imx6ull-14x14-evk-externaluart.dtb /*i.mx6ull evk dtb*/))
```

Uboot 中调试串口改成 UART6

在 uboot 配置头文件中将初始化 UART1 改成 UART6

```
Include\configs\mx6ul_14x14_evk.h
(Include\configs\mx6ullevk.h /*i.mx6ull evk config file*/)
#define CONFIG_MXC_UART_BASE            MX6UL_UART6_BASE_ADDR /*change to uart6 UART6_BASE */
```

MX6UL_UART6_BASE_ADD 定义在：

```
arch\arm\include\asm\arch-mx6\imx-regs.h
#define MX6UL_UART6_BASE_ADDR           (AIPS2_OFF_BASE_ADDR + 0x7C000)
```

在 uboot 传递给内核的 console hard coding 的参数将 UART1 改成 UART6。

```
#if defined(CONFIG_SYS_BOOT_NAND)
#define CONFIG_EXTRA_ENV_SETTINGS \
        …
          "console=ttymxc5\0" \
          "bootargs=console=ttymxc5,115200 ubi.mtd=3 "  \
         …
        #else
         #define CONFIG_EXTRA_ENV_SETTINGS \
        …
         "console=ttymxc5\0" \
```

```
    …
    #endif
```

C 代码中将 DEBUG UART 的 IOMUX 从 UART1 改成 UART6:

```
board\freescale\mx6ul_14x14_evk\mx6ul_14x14_evk.c
(board\freescale\mx6ullevk\mx6ullevk.c /*i.mx6ull evk source file*/)
static iomux_v3_cfg_t const uart6_pads[] = {
        MX6_PAD_ENET2_RX_DATA0__UART6_DCE_TX | MUX_PAD_CTRL(UART_PAD_CTRL),
        MX6_PAD_ENET2_RX_DATA1__UART6_DCE_RX | MUX_PAD_CTRL(UART_PAD_CTRL),
};
static void setup_iomux_uart(void)
{
    imx_iomux_v3_setup_multiple_pads(uart6_pads, ARRAY_SIZE(uart6_pads));
}
```

移除掉 IOMUX 冲突：

```
static iomux_v3_cfg_t const fec2_pads[] = {
        …
        /*
        MX6_PAD_ENET2_RX_DATA0__ENET2_RDATA00 | MUX_PAD_CTRL(ENET_PAD_CTRL),
        MX6_PAD_ENET2_RX_DATA1__ENET2_RDATA01 | MUX_PAD_CTRL(ENET_PAD_CTRL),
        */
        …
};
```

特别注意在 i.MX6UL/ULL Rootfs 中使用了 getty 命令强制终端为 ttymxc0,所以这儿也需要修改：

如果使用 ubuntu 主机，挂接 TFcard 后：

```
/media/vmuser/ef457e66-9d82-425c-b336-cd0f758e4f01/etc$ sudo gedit inittab
mxc5:12345:respawn:/bin/start_getty 115200 ttymxc5 /* ttymxc0 改为 ttymxc5*/
```

当然在使用 uart6 启动过 uboot 和 kernel 后，再进入 console 时硬件改为连接到 UART1，也可以在线修改：

```
root@imx6ulevk:~# vi /etc/inittab
```

修改后的 console 打印如下：

```
Freescale i.MX Release Distro 4.1.15-2.0.0 imx6ulevk /dev/ttymxc5
imx6ulevk login:
```

去除掉无用的驱动及其 IOMUX

在设计 i.MX6UL/ULL EVK 扩展串口板时，设计出了 8 个串口以及其它管脚，很多都与其它外设模块冲突，所以我们首先是要把本板不支持的功能及其 IOMUX 去掉

```
pwd
/home/ysli/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/linux-imx/4.1.15-r0/git
(如果已经拷贝出源文件目录如下：pwd
/home/vmuser/imx6ulullexternaluart/sources/kernal-2.0.3)
vi arch/arm/boot/dts/imx6ul-14x14-evk-externaluart.dts
(vi arch/arm/boot/dts/imx6ull-14x14-evk-externaluart.dts /*i.mx6ull evk dts file*/)
```

需要去掉的驱动如下：将其状态设置为 disabled 就可以取消息其驱动及设备注册。

```
backlight { status = "disabled";}
pxp_v4l2 { status = "disabled";}ex
reg_can_3v3: regulator@0 { status = "disabled";}
sound{ status = "disabled";}
spi4{ status = "disabled";}
fec2{ status = "disabled";}
flexcan1{ status = "disabled";}
flexcan2{ status = "disabled";}
mag3110@0e{ status = "disabled";}
fxls8471@1e{ status = "disabled";}
codec: wm8960@1a{ status = "disabled";}
&lcdif{ status = "disabled";}
&pxp{ status = "disabled";}
&qspi{ status = "disabled";}
&sim2{ status = "disabled";}
&tsc{ status = "disabled";}
```

驱动去掉后，相应的 IOMUX 设置就不会被调用了：

增加 i.MX6UL/ULL 本身串口支持

i.MX6UL/ULL EVK 板的原生设计中仅有两个串口，其中串口 1 为调试串口，串口 2 为标准 4 线串口。i.MX6UL/ULL EVK 扩展串口板设计为 8 个串口，所以要增加 6 个，然后将串口 6 改成调试串口，串口 1 是支持 RTS/CTS 的标准串口：

```
&uart1 {
	pinctrl-names = "default";
```

```
        pinctrl-0 = <&pinctrl_uart1>;
        fsl,uart-has-rtscts;
        /* for DTE mode, add below change */
        /* fsl,dte-mode; */
        /* pinctrl-0 = <&pinctrl_uart1dte>; */
        status = "okay";
};
    &uart2 {
…
};
&uart3 {
              pinctrl-names = "default";
              pinctrl-0 = <&pinctrl_uart3>;
              fsl,uart-has-rtscts;
              /* for DTE mode, add below change */
              /* fsl,dte-mode; */
              /* pinctrl-0 = <&pinctrl_uart3dte>; */
              status = "okay";
};
&uart4 {
              pinctrl-names = "default";
              pinctrl-0 = <&pinctrl_uart4>;
              fsl,uart-has-rtscts;
              /* for DTE mode, add below change */
              /* fsl,dte-mode; */
              /* pinctrl-0 = <&pinctrl_uart4dte>; */
              status = "okay";
};
&uart5 {
              pinctrl-names = "default";
              pinctrl-0 = <&pinctrl_uart5>;
              fsl,uart-has-rtscts;
              /* for DTE mode, add below change */
              /* fsl,dte-mode; */
              /* pinctrl-0 = <&pinctrl_uart5dte>; */
              status = "okay";
```

```
};
```

/*注意 uart6 因为 IOMUX 冲突原因，没有设计 RTS/CTS 管脚，所以用作 DEBUG 串口*/

```
&uart6 {
	pinctrl-names = "default";
	pinctrl-0 = <&pinctrl_uart6>;
	status = "okay";
};
&uart7 {
	pinctrl-names = "default";
	pinctrl-0 = <&pinctrl_uart7>;
	fsl,uart-has-rtscts;
	/* for DTE mode, add below change */
	/* fsl,dte-mode; */
	/* pinctrl-0 = <&pinctrl_uart7dte>; */
	status = "okay";
};
&uart8 {
	pinctrl-names = "default";
	pinctrl-0 = <&pinctrl_uart8>;
	fsl,uart-has-rtscts;
	/* for DTE mode, add below change */
	/* fsl,dte-mode; */
	/* pinctrl-0 = <&pinctrl_uart8dte>; */
	status = "okay";
};
```

然后根据 IOMUX 表，配置其 IOMUX:

/*uart1 as debug port first, and do not configure its rts/cts*/

```
		pinctrl_uart1: uart1grp {
			fsl,pins = <
				MX6UL_PAD_UART1_TX_DATA__UART1_DCE_TX 0x1b0b1
				MX6UL_PAD_UART1_RX_DATA__UART1_DCE_RX 0x1b0b1
				MX6UL_PAD_UART1_RTS_B__UART1_DCE_RTS 0x1b0b1
				MX6UL_PAD_UART1_CTS_B__UART1_DCE_CTS 0x1b0b1
			>;
		};
```

注意 i.MX6UL/ULL EVK 板原生 BSP 中使用 MX6UL_PAD_UART1_RTS_B 作为 SD1 的 CD GPIO,在 i.MX6UL/ULL EVK 扩展串口板中我们修改为：

```
                /*MX6UL_PAD_UART1_RTS_B__GPIO1_IO19     0x17059*/ /* SD1 CD */
                MX6UL_PAD_ENET2_TX_EN__GPIO2_IO13  0x17059 /* SD1 CD */

&usdhc1 {
…
   /*       cd-gpios = <&gpio1 19 GPIO_ACTIVE_LOW>; */
    cd-gpios = <&gpio2 13 GPIO_ACTIVE_LOW>;
…
}

              pinctrl_uart2: uart2grp {
                     fsl,pins = <
                            MX6UL_PAD_UART2_TX_DATA__UART2_DCE_TX       0x1b0b1
                            MX6UL_PAD_UART2_RX_DATA__UART2_DCE_RX       0x1b0b1
/*uart2 RTS/CTS pin need re-configura it*/
                            MX6UL_PAD_UART2_RTS_B__UART2_DCE_RTS   0x1b0b1
                            MX6UL_PAD_UART2_CTS_B__UART2_DCE_CTS   0x1b0b1
                            /*
                            MX6UL_PAD_UART3_RX_DATA__UART2_DCE_RTS      0x1b0b1
                            MX6UL_PAD_UART3_TX_DATA__UART2_DCE_CTS      0x1b0b1
                            */
                     >;
              };
…
              pinctrl_uart3: uart3grp {
                     fsl,pins = <
                            MX6UL_PAD_UART3_TX_DATA__UART3_DCE_TX   0x1b0b1
                            MX6UL_PAD_UART3_RX_DATA__UART3_DCE_RX       0x1b0b1
                            MX6UL_PAD_UART3_RTS_B__UART3_DCE_RTS          0x1b0b1
                            MX6UL_PAD_UART3_CTS_B__UART3_DCE_CTS          0x1b0b1
                     >;
              };
              pinctrl_uart4: uart4grp {
```

```
                fsl,pins = <
                    MX6UL_PAD_LCD_CLK__UART4_DCE_TX     0x1b0b1
                        MX6UL_PAD_LCD_ENABLE__UART4_DCE_RX 0x1b0b1
                        MX6UL_PAD_LCD_VSYNC__UART4_DCE_RTS 0x1b0b1
                        MX6UL_PAD_LCD_HSYNC__UART4_DCE_CTS      0x1b0b1
                >;
            };
            pinctrl_uart5: uart5grp {
                fsl,pins = <
                        MX6UL_PAD_UART5_TX_DATA__UART5_DCE_TX     0x1b0b1
                        MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX     0x1b0b1
                        MX6UL_PAD_GPIO1_IO08__UART5_DCE_RTS  0x1b0b1
                        MX6UL_PAD_GPIO1_IO09__UART5_DCE_CTS  0x1b0b1
                >;
            };
```

UART5 与 I2C2 IOMUX 冲突，所以需要把 I2C2 的驱动移掉，或是将其 IOMUX 注掉

```
        pinctrl_i2c2: i2c2grp {
            fsl,pins = <
/*johnli remove for pin conflict with uart5*/
/*
                MX6UL_PAD_UART5_TX_DATA__I2C2_SCL 0x4001b8b0
                MX6UL_PAD_UART5_RX_DATA__I2C2_SDA 0x4001b8b0
*/
            >;
        };
```

UART5 与 SD1 RESET 冲突，需要注掉：

```
/*johnliremove it to configlict with uart MX6UL_PAD_GPIO1_IO09__GPIO1_IO09     0x17059*/  /* SD1 RESET */

            pinctrl_uart6: uart6grp {
                fsl,pins = <
                        MX6UL_PAD_ENET2_RX_DATA0__UART6_DCE_TX     0x1b0b1
                        MX6UL_PAD_ENET2_RX_DATA1__UART6_DCE_RX     0x1b0b1
```

/*注意对 i.MX6ULL,其 UART5_RX_DATA 的 dasiy chain 值与 i.MX6UL 不同，如下：

IOMUXC_UART5_RX_DATA_SELECT_INPUT field descriptions (continued)

Field	Description
	Instance: uart5, In Pin: uart_RX_DATA 000 **CSI_DATA00_ALT8** — Selecting Pad: CSI_DATA00 for Mode: ALT8 001 **CSI_DATA01_ALT8** — Selecting Pad: CSI_DATA01 for Mode: ALT8 010 **GPIO1_IO04_ALT8** — Selecting Pad: GPIO1_IO04 for Mode: ALT8 011 **GPIO1_IO05_ALT8** — Selecting Pad: GPIO1_IO05 for Mode: ALT8 100 **UART1_TX_DATA_ALT9** — Selecting Pad: UART1_TX_DATA for Mode: ALT9 101 **UART1_RX_DATA_ALT9** — Selecting Pad: UART1_RX_DATA for Mode: ALT9 110 **UART5_TX_DATA_ALT0** — Selecting Pad: UART5_TX_DATA for Mode: ALT0 111 **UART5_RX_DATA_ALT0** — Selecting Pad: UART5_RX_DATA for Mode: ALT0

而 i.MX6UL 的值如下：

IOMUXC_UART5_RX_DATA_SELECT_INPUT field descriptions (continued)

Field	Description
	000 **CSI_DATA00_ALT8** — Selecting Pad: CSI_DATA00 for Mode: ALT8 001 **CSI_DATA01_ALT8** — Selecting Pad: CSI_DATA01 for Mode: ALT8 010 **GPIO1_IO04_ALT8** — Selecting Pad: GPIO1_IO04 for Mode: ALT8 011 **GPIO1_IO05_ALT8** — Selecting Pad: GPIO1_IO05 for Mode: ALT8 100 **UART5_TX_DATA_ALT0** — Selecting Pad: UART5_TX_DATA for Mode: ALT0 101 **UART5_RX_DATA_ALT0** — Selecting Pad: UART5_RX_DATA for Mode: ALT0

所以 i.MX6ULL 对 UART5_RX_DATA 的 IOMUX 要重新定义，此处为 BSP bug:

```
vi arch/arm/boot/dts/imx6ull-pinfunc.h
#undef MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX
#define MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX          0x00C0 0x034C 0x0644 0x0 0x7 /*i.mx6ul 的 dasiy chain 值是 0x5*/
```

此种 bug 还涉及 UART5_RTS 相关输入 pin，我们在最新的 BSP 中会 fix 掉。

*/

/*注意 uart6 因为 IOMUX 冲突原因，没有设计 RTS/CTS 管脚*/

```
                >;
            };
            pinctrl_uart7: uart7grp {
                fsl,pins = <
                    MX6UL_PAD_LCD_DATA16__UART7_DCE_TX 0x1b0b1
                    MX6UL_PAD_LCD_DATA17__UART7_DCE_RX 0x1b0b1
                    MX6UL_PAD_LCD_DATA07__UART7_DCE_RTS      0x1b0b1
                    MX6UL_PAD_LCD_DATA06__UART7_DCE_CTS      0x1b0b1
                >;
            };
            pinctrl_uart8: uart8grp {
                fsl,pins = <
                    MX6UL_PAD_LCD_DATA20__UART8_DCE_TX 0x1b0b1
                    MX6UL_PAD_LCD_DATA21__UART8_DCE_RX 0x1b0b1
```

```
MX6UL_PAD_LCD_DATA05__UART8_DCE_RTS          0x1b0b1
MX6UL_PAD_LCD_DATA04__UART8_DCE_CTS          0x1b0b1
>;
};
```

最后，因为我们使用 UART6 为调试串口，而 UART1 为普通串口，所以我们把 UART6 的 DMA 禁掉，把 UART1 的 DMA 打开，UART1 的 DMA 号如下：

25	UART1	UART1 Rx FIFO
26	UART1	UART1 Tx FIFO
27	UART2	UART2 Rx FIFO
28	UART2	UART2 Tx FIFO

```
vi arch/arm/boot/dts/imx6ul.dtsi (i.MX6ULL 为 imx6ull.dtsi)
uart1: serial@02020000 {
  …
    /*johnli add dma*/
    dmas = <&sdma 25 4 0>, <&sdma 26 4 0>;
        dma-names = "rx", "tx";
  …
};
uart6: serial@021fc000 {
  …
    /* johnli disable uart6 dma
        dmas = <&sdma 0 4 0>, <&sdma 47 4 0>;
        dma-names = "rx", "tx";
    */
…
};
```

驱动测试：

- 启动信息：

```
2018000.serial: ttymxc6 at MMIO 0x2018000 (irq = 19, base_baud = 5000000) is a IMX
2020000.serial: ttymxc0 at MMIO 0x2020000 (irq = 20, base_baud = 5000000) is a IMX
2024000.serial: ttymxc7 at MMIO 0x2024000 (irq = 21, base_baud = 5000000) is a IMX
21e8000.serial: ttymxc1 at MMIO 0x21e8000 (irq = 227, base_baud = 5000000) is a IMX
21ec000.serial: ttymxc2 at MMIO 0x21ec000 (irq = 228, base_baud = 5000000) is a IMX
21f0000.serial: ttymxc3 at MMIO 0x21f0000 (irq = 229, base_baud = 5000000) is a IMX
21f4000.serial: ttymxc4 at MMIO 0x21f4000 (irq = 230, base_baud = 5000000) is a IMX
21fc000.serial: ttymxc5 at MMIO 0x21fc000 (irq = 232, base_baud = 5000000) is a IMX
```

console [ttymxc5] enabled

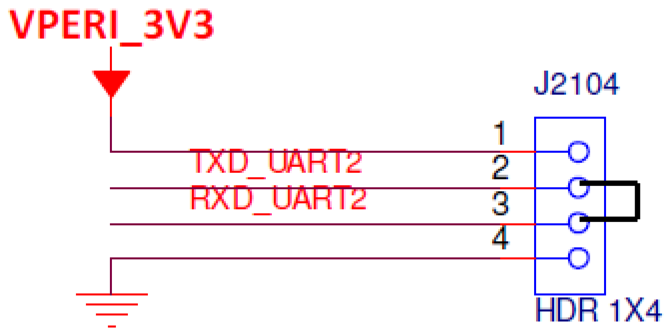
- 注册的驱动：

root@imx6ulevk:/dev# ls ttymxc*

ttymxc0 ttymxc1 ttymxc2 ttymxc3 ttymxc4? ttymxc5 ttymxc6 ttymxc7

- 测试的硬件连接

测试时我们使用外部回环连接，如下连接：



我们使用杜邦线将 TXD/RXD 直接连接：（如下 UART2 的连接实图）

- 测试命令

简单的回环测试可以如下：

root@imx6ulevk:/dev# cat ttymxc1 &

[1] 529

root@imx6ulevk:/dev# echo 1234 > ttymxc1

root@imx6ulevk:/dev# 1234

1234

另外，我们的/unit_test 下也提供 uart 测试代码，可以做内部回环测试：

root@imx6ulevk:/unit_tests# ./mxc_uart_test.out /dev/ttymxc1

```
Test: MXC UART!
Usage: mxc_uart_test <UART device name, opens UART2 if no dev name is specified>
/dev/ttymxc1 opened
Attributes set
Test: IOCTL Set
Data Written= Test
Data Read back= Test
```

依次测试 ttymxc0~7，会发现除了 ttymxc5 是调试串口外，其它内部回环测试都可以通过。

增加 GPIO 输出支持（GPIO_LED）

i.MX6UL/ULL EVK 扩展串口板为每个原生串口接口设计一个 GPIO，一般只要我们设计相关管脚为 GPIO，就可以使用 sys 文件系统在用户空间直接使用其 GPIO，此处我们将其中 四个设计为输出，并定义其为 GPIO_LED 驱动。

GPIO_LED 驱动默认已经编译到内核中了，如下内核配置文件：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_LEDS_GPIO=y
```

GPIO_LED 的驱动 Makefile 文件

```
vi drivers/leds/Makefile
obj-$(CONFIG_LEDS_GPIO)		+= leds-gpio.o
```

GPIO_LED 的驱动源代码文件

```
vi drivers/leds/leds-gpio.c
static const struct of_device_id of_gpio_leds_match[] = {
	{ .compatible = "gpio-leds", },
	{},
};
```

GPIO_LED 的驱动 binding 文件说明了如何在 DTS 中加上 GPIO_LED 支持

```
vi Documentation/devicetree/bindings/leds/leds-gpio.txt
```

i.MX6UL/ULL EVK 扩展串口板的支持如下：

```
leds {
		compatible = "gpio-leds";
		pinctrl-names = "default";
		pinctrl-0 = <&pinctrl_gpio_leds>;
		led0: user1 {
			label = "user1";
```

```
                gpios = <&gpio5 0 GPIO_ACTIVE_HIGH>;
                default-state = "on";
                linux,default-trigger = "heartbeat";
            };
            led1: user2 {
                label = "user2";
                gpios = <&gpio5 1 GPIO_ACTIVE_HIGH>;
                default-state i2= "on";
                linux,default-trigger = "heartbeat";
            };
            led2: user3 {
                label = "user3";
                gpios = <&gpio5 2 GPIO_ACTIVE_HIGH>;
                default-state = "on";
                linux,default-trigger = "heartbeat";
            };
            led3: user4 {
                label = "user4";
                gpios = <&gpio5 3 GPIO_ACTIVE_HIGH>;
                default-state = "on";
                linux,default-trigger = "heartbeat";
            };
        };
```

由于 GPIO5_3 用于了 reg_gpio_dvfs 驱动，只能将此驱动注掉：

```
        reg_gpio_dvfs: regulator-gpio {
            status = "disabled";
            };
```

IOMUX 配置在 i.MX6UL 上如下：

```
pinctrl_gpio_leds: gpioledsgrp {
                fsl,pins = <
                    MX6UL_PAD_SNVS_TAMPER0__GPIO5_IO00      0x1b0b0
                    MX6UL_PAD_SNVS_TAMPER1__GPIO5_IO01      0x1b0b0
                    MX6UL_PAD_SNVS_TAMPER2__GPIO5_IO02      0x1b0b0
                    MX6UL_PAD_SNVS_TAMPER3__GPIO5_IO03      0x1b0b0
                >;
```

```
        };
```

然后将冲突的 IOMUX 注掉：

```
                pinctrl_hog_1: hoggrp-1 {
                fsl,pins = <
/*                      MX6UL_PAD_SNVS_TAMPER0__GPIO5_IO00     0x80000000*/
                >;
        };
```

IOMUX 配置在 i.MX6ULL 上,由于 TAMPER 管脚是在 SNVS 域，所以配置如下：并且 PIN 脚名使用 i.MX6ULL 的专用名，参考文件 arch\arm\boot\dts\imx6ull-pinfunc-snvs.h

```
&iomuxc_snvs {
…
/*johnli add for gpio led*/

    pinctrl_gpio_leds: gpioledsgrp {
        fsl,pins = <
            MX6ULL_PAD_SNVS_TAMPER0__GPIO5_IO00         0x1b0b0
            MX6ULL_PAD_SNVS_TAMPER1__GPIO5_IO01         0x1b0b0
            MX6ULL_PAD_SNVS_TAMPER2__GPIO5_IO02         0x1b0b0
            MX6ULL_PAD_SNVS_TAMPER3__GPIO5_IO03         0x1b0b0
        >;
    };

/*end*/
…
}
```

然后将冲突的 IOMUX 注掉：

```
  pinctrl_dvfs: dvfsgrp {
        fsl,pins = <
            /*johnli remove  MX6ULL_PAD_SNVS_TAMPER3__GPIO5_IO03      0x79 */
        >;
    };
```

驱动测试：

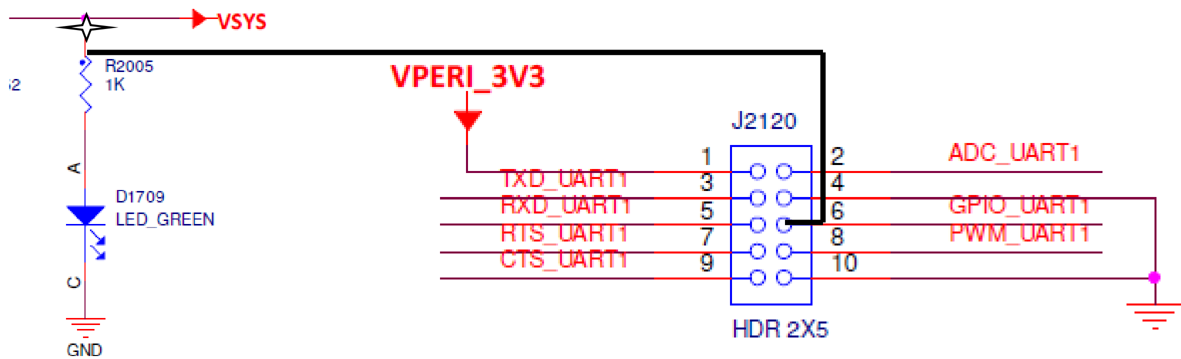
- SYS 文件系统节点

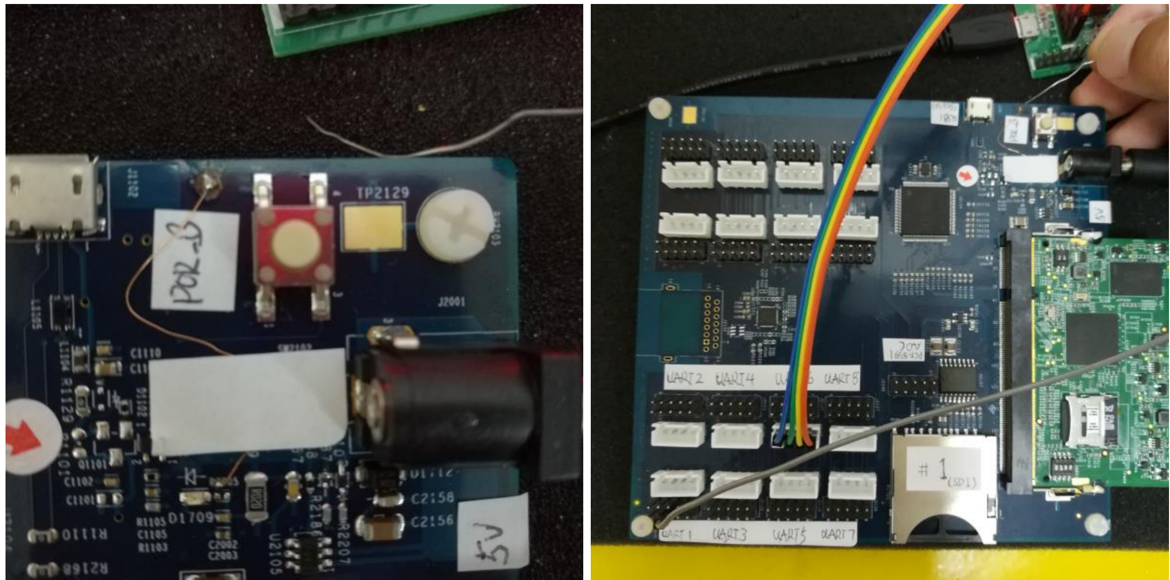
```
root@imx6ulevk:/sys/class/leds# ls
mmc0::  mmc1::  user1  user2  user3  user4
```

```
root@imx6ulevk:/sys/class/leds# cd user1
root@imx6ulevk:/sys/class/leds/user1# ls
brightness  device  max_brightness  power  subsystem  trigger  uevent
root@imx6ulevk:/sys/class/leds/user1#
```

- 硬件连接

i.MX6UL/ULL EVK 扩展串口板上本身设计有一个电源灯，我们将连接到 VSYS 上限流电阻 R2005 断开，再连接到一个定位点做测试点，然后从 LED 管脚跳线过来就可以测试了，rework 原理图和实际连接如下：UART1~4 的 GPIO 设计会 GPIO_LED。





- 测试命令

1. 测试开关

LED 我们默认设置为 linux,default-trigger = "heartbeat"; 所以连接好后，LED 会使用 heatbeat 模式闪烁

如果要测试开关，我们需要先将 trigger 设置为 gpio:

```
root@imx6ulevk:/sys/class/leds/user1# cat trigger
none rc-feedback nand-disk mmc0 mmc1 timer oneshot [heartbeat] backlight gpio
root@imx6ulevk:/sys/class/leds/user1# echo gpio > trigger
root@imx6ulevk:/sys/class/leds/user1# cat trigger
none rc-feedback nand-disk mmc0 mmc1 timer oneshot heartbeat backlight [gpio]
```

然后设置其 brightess

```
root@imx6ulevk:/sys/class/leds/user1# cat max_brightness
255
root@imx6ulevk:/sys/class/leds/user1# cat brightness
0
root@imx6ulevk:/sys/class/leds/user1# echo 128 > brightness
root@imx6ulevk:/sys/class/leds/user1# cat brightness
128
root@imx6ulevk:/sys/class/leds/user1#
```

2. 测试闪烁

```
root@imx6ulevk:/sys/class/leds/user1# cat trigger
none rc-feedback nand-disk mmc0 mmc1 timer oneshot [heartbeat] backlight gpio
echo timer > trigger
```

增加 GPIO 输入支持（GPIO_KEY）

另外四个 GPIO，我们配置为输入，GPIO_KEY 驱动

GPIO_KEY 驱动默认已经编译到内核中了，如下内核配置文件：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_KEYBOARD_GPIO=y
```

GPIO_KEY 的驱动 Makefile 文件

```
vi drivers/input/keyboard/Makefile
obj-$(CONFIG_KEYBOARD_GPIO)        += gpio_keys.o
```

GPIO_LED 的驱动源代码文件

```
vi drivers/leds/gpio-keys.c
static const struct of_device_id gpio_keys_of_match[] = {
```

```
{ .compatible = "gpio-keys", },
{ },
};
```

GPIO_KEY 的驱动 binding 文件说明了如何在 DTS 中加上 GPIO_KEY 支持

```
vi Documentation/devicetree/bindings/input/gpio-keys.txt
```

i.MX6UL/ULL EVK 扩展串口板的支持如下：其中 linux,code 项为实际上报的 key_event 的键值。

```
gpio-keys {
    compatible = "gpio-keys";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpio_keys>;
    power {
        label = "Power Button";
        gpios = <&gpio5 4 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_POWER>;
    };
    volume-up {
        label = "Volume Up";
        gpios = <&gpio5 5 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_VOLUMEUP>;
    };
    volume-down {
        label = "Volume Down";
        gpios = <&gpio5 6 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_VOLUMEDOWN>;
    };
    home {
        label = "Home Button";
        gpios = <&gpio5 7 GPIO_ACTIVE_LOW>;
        gpio-key,wakeup;
        linux,code = <KEY_HOME>;
    };
```

```
};
```

IOMUX 配置在 i.MX6UL 上如下：

```
pinctrl_gpio_keys: gpio_keysgrp {
                fsl,pins = <
                        MX6UL_PAD_SNVS_TAMPER4__GPIO5_IO04 0x1b0b0
                        MX6UL_PAD_SNVS_TAMPER5__GPIO5_IO05  0x1b0b0
                        MX6UL_PAD_SNVS_TAMPER6__GPIO5_IO06  0x1b0b0
                        MX6UL_PAD_SNVS_TAMPER7__GPIO5_IO07  0x1b0b0
                >;
        };
```

IOMUX 配置在 i.MX6ULL 上,由于 TAMPER 管脚是在 SNVS 域，所以配置如下：

并且 PIN 脚名使用 i.MX6ULL 的专用名，参考文件 arch\arm\boot\dts\imx6ull-pinfunc-snvs.h

```
&iomuxc_snvs {
…
/*johnli add for gpio key*/
        pinctrl_gpio_keys: gpio_keysgrp {
                fsl,pins = <
                        MX6ULL_PAD_SNVS_TAMPER4__GPIO5_IO04 0x1b0b0
                        MX6ULL_PAD_SNVS_TAMPER5__GPIO5_IO05  0x1b0b0
                        MX6ULL_PAD_SNVS_TAMPER6__GPIO5_IO06  0x1b0b0
                        MX6ULL_PAD_SNVS_TAMPER7__GPIO5_IO07  0x1b0b0
                >;
        };
/*end*/…
}
```

然后将冲突的 IOMUX 注掉：

```
        pinctrl_spi4: spi4grp {
                fsl,pins = <
                        MX6ULL_PAD_BOOT_MODE0__GPIO5_IO10       0x70a1
                        MX6ULL_PAD_BOOT_MODE1__GPIO5_IO11       0x70a1
                        /*johnli remove
                        MX6ULL_PAD_SNVS_TAMPER7__GPIO5_IO07      0x70a1
                         */
                        MX6ULL_PAD_SNVS_TAMPER8__GPIO5_IO08      0x80000000
```

```
                >;
        };
        pinctrl_sai2_hp_det_b: sai2_hp_det_grp {
                fsl,pins = <
                        /*johnli remove  MX6ULL_PAD_SNVS_TAMPER4__GPIO5_IO04 */  0x17059
                >;
        };
```

驱动测试：

- 启动信息

```
input: gpio-keys as /devices/platform/gpio-keys/input/input1
```

- 测试的硬件连接

由于 GPIO_KEY 管脚是默认上拉，然后下拉触发，所以我们只要从相应 GPIO_KEY 管脚跳线到地，就可以测试 GPIO_KEY。UART5~8 的 GPIO 设计为 GPIO_KEY

- 测试命令

```
root@imx6ulevk:/dev# evtest
No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      20cc000.snvs:snvs-powerkey
/dev/input/event1:      gpio-keys
Select the device event number [0-1]: 1
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 102 (KEY_HOME)
    Event code 114 (KEY_VOLUMEDOWN)
    Event code 115 (KEY_VOLUMEUP)
    Event code 116 (KEY_POWER)
Properties:
Testing ... (interrupt to exit)
Event: time 1499054766.630752, type 1 (EV_KEY), code 116 (KEY_POWER), value 1
```

```
Event: time 1499054766.630752, -------------- SYN_REPORT ------------
Event: time 1499054766.740596, type 1 (EV_KEY), code 116 (KEY_POWER), value 0
…
```

增加 PWM 支持

在 i.MX6UL/ULL EVK 板级 DTS 中仅使用了一个来作为背光。在 i.MX6UL/ULL EVK 扩展串口板为每个原生串口接口设计一个 PWM，所以我们要分别增加到 8 个 PWM：

在板级 DTS 中增加到 8 个 PWM 并配置其 IOMUX

```
 vi arch/arm/boot/dts/imx6ul-14x14-evk-externaluart.dts
    &pwm1 {
        …
};
/*johnli add for pwm*/
&pwm2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_pwm2>;
        status = "ok";
};
&pwm3 {
        pinctrl-names = "default";
        pinctrl-0 = <& pinctrl_pwm3>;
        status = "ok";
};

&pwm4 {
        pinctrl-names = "default";
        pinctrl-0 = <& pinctrl_pwm4>;
        status = "ok";
};
&pwm5 {
        pinctrl-names = "default";
        pinctrl-0 = <& pinctrl_pwm5>;
        status = "ok";
};
&pwm6 {
        pinctrl-names = "default";
```

```
		pinctrl-0 = <& pinctrl_pwm6>;
		status = "ok";
};
&pwm7 {
		pinctrl-names = "default";
		pinctrl-0 = <& pinctrl_pwm7>;
		status = "ok";
};
&pwm8 {
		pinctrl-names = "default";
		pinctrl-0 = <& pinctrl_pwm8>;
		status = "ok";
};
```

IOMUX 配置为：

```
		pinctrl_pwm1: pwm1grp {
			fsl,pins = <
				MX6UL_PAD_LCD_DATA00__PWM1_OUT 0x110b0
				/*
				MX6UL_PAD_GPIO1_IO08__PWM1_OUT   0x110b0
				*/
			>;
		};
		pinctrl_pwm2: pwm2grp {
				fsl,pins = <
					MX6UL_PAD_LCD_DATA01__PWM2_OUT   0x110b0
				>;
		};
		pinctrl_pwm3: pwm3grp {
				fsl,pins = <
					MX6UL_PAD_LCD_DATA02__PWM3_OUT   0x110b0
				>;
		};

		pinctrl_pwm4: pwm4grp {
				fsl,pins = <
```

```
					MX6UL_PAD_LCD_DATA03__PWM4_OUT   0x110b0
				>;
			};
			pinctrl_pwm5: pwm5grp {
				fsl,pins = <
					MX6UL_PAD_LCD_DATA18__PWM5_OUT   0x110b0
				>;
			};
			pinctrl_pwm6: pwm6grp {
				fsl,pins = <
					MX6UL_PAD_LCD_DATA19__PWM6_OUT   0x110b0
				>;
			};
			pinctrl_pwm7: pwm7grp {
				fsl,pins = <
					MX6UL_PAD_JTAG_TCK__PWM7_OUT   0x110b0
				>;
			};
			pinctrl_pwm8: pwm8grp {
				fsl,pins = <
					MX6UL_PAD_JTAG_TRST_B__PWM8_OUT   0x110b0
				>;
			};
```

原生的芯片级 dts 文件：arch/arm/boot/dts/imx6ul.dtsi(imx6ull.dtsi 是注掉的，放开再同样修改)中，pwm5~7 的 clocks 源设置为空 IMX6UL_CLK_DUMMY，所以需要修改为正确的值：

```
vi arch/arm/boot/dts/imx6ul.dtsi,
			pwm5: pwm@020f0000 {
…
				clocks = <&clks IMX6UL_CLK_PWM5>,
					 <&clks IMX6UL_CLK_PWM5>;
…
			};
			pwm6: pwm@020f4000 {
…
				clocks = <&clks IMX6UL_CLK_PWM6>,
					 <&clks IMX6UL_CLK_PWM6>;
```

```
…
		};
		pwm7: pwm@020f8000 {
…
			clocks = <&clks IMX6UL_CLK_PWM7>,
				 <&clks IMX6UL_CLK_PWM7>;
…
		};
		pwm8: pwm@020fc000 {
…
			clocks = <&clks IMX6UL_CLK_PWM8>,
				 <&clks IMX6UL_CLK_PWM8>;
…
	};
```

i.MX6ULL 需要多修改：

```
vi arch/arm/boot/dts/imx6ull.dtsi,
	pwm2: pwm@02084000 {
…
			clocks = <&clks IMX6UL_CLK_PWM2>,
				 <&clks IMX6UL_CLK_PWM2>;
…
		};
		pwm4: pwm@0208c000 {
…
			clocks = <&clks IMX6UL_CLK_PWM4>,
				 <&clks IMX6UL_CLK_PWM4>;
…
		};
```

驱动测试：

•SYS 文件系统节点

```
root@imx6ulevk:/sys/class/pwm# ls
pwmchip0  pwmchip1  pwmchip2  pwmchip3  pwmchip4  pwmchip5  pwmchip6  pwmchip7
root@imx6ulevk:/sys/class/pwm# cd pwmchip0
root@imx6ulevk:/sys/class/pwm/pwmchip0# ls
device  export  npwm  power  subsystem  uevent  unexport
```

```
root@imx6ulevk:/sys/class/pwm/pwmchip0#
```

- 硬件连接

与测试 GPIO_LED 类似，我们可以用 LED 灯来测试 PWM。PWM 管脚为 UART 插座的第 8 脚。

- 测试命令

```
root@imx6ulevk:/sys/class/pwm/pwmchip0# echo 0 > export
root@imx6ulevk:/sys/class/pwm/pwmchip0# ls
device  export  npwm  power  pwm0  subsystem  uevent  unexport
root@imx6ulevk:/sys/class/pwm/pwmchip0# cd pwm0
root@imx6ulevk:/sys/class/pwm/pwmchip0/pwm0# ls
duty_cycle  enable  period  polarity  power  uevent
root@imx6ulevk:/sys/class/pwm/pwmchip0/pwm0# echo 1 > enable
root@imx6ulevk:/sys/class/pwm/pwmchip0/pwm0# echo 1000000 > period
root@imx6ulevk:/sys/class/pwm/pwmchip0/pwm0# echo 100000 > duty_cycle
```

period 和 duty_cycle 的单位是 ns。所以以上设置为 PWM1 1KHZ 50% duty cycle。减小 duty cycle 可以看到 LED 灯的亮度越来越淡。

增加 i.MX6UL 本身 ADC 支持

i.MX6UL/ULL EVK 扩展串口板为每个原生串口接口设计一个 ADC，其中有 6 个是使用 i.MX6UL 本身的 ADC 的。i.MX6UL/ULL EVK 默认没有增加 ADC 驱动，需要增加：

i.MX6UL/ULL ADC 驱动默认已经编译到内核中了，如下内核配置文件：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_VF610_ADC=y
```

i.MX6UL/ULLADC 的驱动 Makefile 文件

```
vi drivers/iio/adc/Makefile
obj-$(CONFIG_VF610_ADC) += vf610_adc.o
```

ADC 的驱动源代码文件

```
vi drivers/iio/adc/vf610_adc.c
static const struct of_device_id vf610_adc_match[] = {
	{ .compatible = "fsl,vf610-adc", },
	{ /* sentinel */ }
};
```

ADC 的驱动 binding 文件说明了如何在 DTS 中加上 ADC 支持

```
vi Documentation/devicetree/bindings/iio/adc/vf610-adc.txt
```

i.MX6UL/ULL EVK 扩展串口板的支持如下：

```
&adc1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_adc1>;
        vref-supply = <&reg_vref_3v3>;
        status = "okay";
    };
```

注意 ADC 使用 3V3 reference regulator, i.MX6UL/ULL EVK 板原生 BSP 未定义，需要加进去：

```
/*johnli add for adc ref-reg*/
    reg_vref_3v3: regulator@2 {
            compatible = "regulator-fixed";
            regulator-name = "vref-3v3";
            regulator-min-microvolt = <3300000>;
            regulator-max-microvolt = <3300000>;
        };
```

IOMUX 配置如下：

```
pinctrl_adc1: adc1grp {
            fsl,pins = <
                MX6UL_PAD_GPIO1_IO00__GPIO1_IO00     0xb0
                MX6UL_PAD_GPIO1_IO01__GPIO1_IO01     0xb0
                MX6UL_PAD_GPIO1_IO02__GPIO1_IO02     0xb0
                MX6UL_PAD_GPIO1_IO03__GPIO1_IO03     0xb0
                MX6UL_PAD_GPIO1_IO04__GPIO1_IO04     0xb0
                MX6UL_PAD_GPIO1_IO05__GPIO1_IO05     0xb0
            >;
        };
```

再去掉冲突的 IOMUX:

```
/*johnli conflict with adc MX6UL_PAD_GPIO1_IO05__USDHC1_VSELECT 0x17059*/ /* SD1 VSELECT */
```

最后 i.MX6UL/ULL EVK BSP 默认只打开了两个输入通道，我们扩展到六个：

```
vi arch/arm/boot/dts/imx6ul.dtsi(imx6ull.dtsi)
adc1: adc@02198000 {
 ….
        num-channels = <6/*johnli add more2*/>;
```

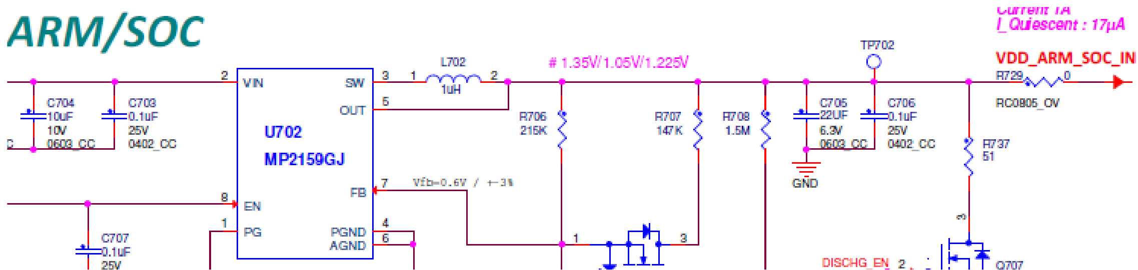
驱动测试：

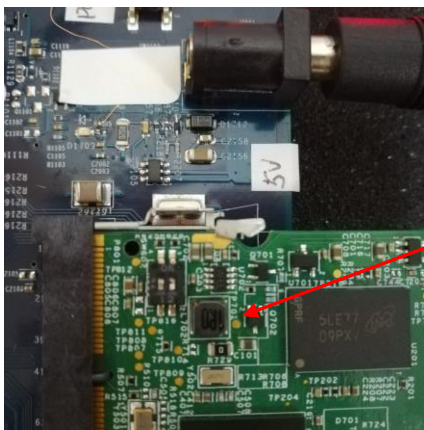
- SYS 文件节点

```
root@imx6ulevk:/sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0# ls
dev             in_voltage3_raw                 in_voltage_scale  sampling_frequency_available
in_voltage0_raw  in_voltage4_raw                name              subsystem
in_voltage1_raw  in_voltage5_raw                of_node           uevent
in_voltage2_raw  in_voltage_sampling_frequency  power
cat in_voltage0_raw
```

- 硬件连接

  UART1~6 插座使用了 i.MX6UL/ULL 自己的 ADC。为管脚 2，ADC 的参考电平是 3.3V。插座的管脚 1 的 3.3V 电源，管脚 4 是地。所以测量最小值可以连接到地，测量最大值可以连接到 3.3V。测量中间值时我们跳线连接到 i.MX6UL/ULL EVK CPU 板上的 TP702 上，如下图：





- 测试命令(连接 UART1 SLOT 的 ADC：管脚 2)

  接地：

```
root@imx6ulevk:/sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0# cat in_voltage0_raw
0
```

  接 3.3V：

```
root@imx6ulevk:/sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0# cat in_voltage0_raw
4041
```

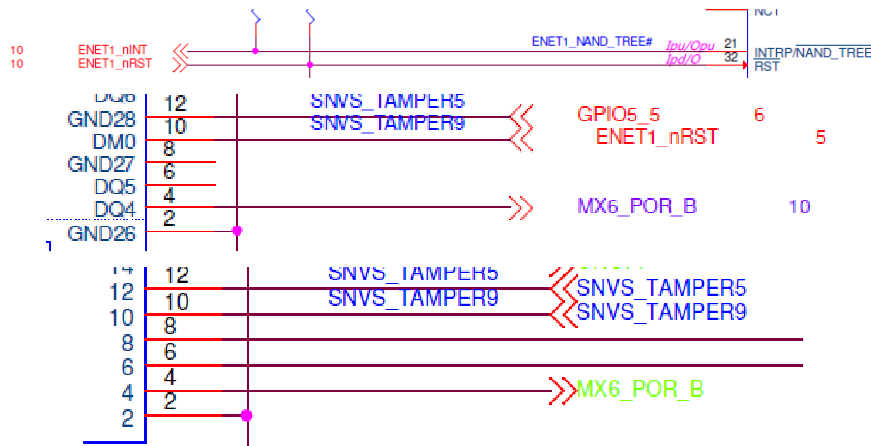
接中间电压：

```
root@imx6ulevk:/sys/devices/platform/soc/2100000.aips-bus/2198000.adc/iio:device0# cat in_voltage0_raw
1327
```

修改网口驱动仅支持一个网口

i.MX6UL/ULL EVK 板本身支持两个 网口，所以驱动将 MDIO 接口设置在网口 2 上，而 i.MX6UL/ULL EVK 扩展串口板上仅用了网口 1，所以驱动需要将 MDIO 接口设置在网口 1 上并调整 IOMUX：

另外，i.MX6UL/ULL EVK 板的 Ethernet PHY reset 管脚是硬件拉动的，而 i.MX6UL/ULL EVK 扩展串口板则使用一 GPIO 来软件拉动，如下：



所以要增加拉动 GPIO reset 的标准代码：

```
&fec1 {
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_enet1>;
  phy-mode = "rmii";
  phy-reset-gpios = <&gpio5 9 0>; /*SNVS_TAMPER9 IOMUX to GPIO5_9*/
  phy-handle = <&ethphy0>;
  status = "okay";
  /*johnli add mdio support*/
  /*MDIO 默认在 fec2 中配置，由于只使用了 fec1,所以要增加对 mdio 的支持*/
            mdio {
            #address-cells = <1>;
            #size-cells = <0>;
```

```
        ethphy0: ethernet-phy@2 { /*2 为 phy 配置的硬件配置的 PHY ID reg : The ID number for the phy, usually a small integer*/
                compatible = "ethernet-phy-ieee802.3-c22";
                reg = <2>;
        };
  };
  /*end*/
};
```

需要将 fec2 原本定义的 ethphy0 去掉，以防止编译冲突：

```
&fec2 {
/* johnli remove it phy-handle = <&ethphy1>;*/
/*
        ethphy0: ethernet-phy@2 {
                compatible = "ethernet-phy-ieee802.3-c22";
                reg = <2>;
*/
        };
};
```

i.MX6UL IOMUX 需要增加 MDIO 支持：

```
  pinctrl_enet1: enet1grp {
                fsl,pins = <
                        MX6UL_PAD_ENET1_RX_EN__ENET1_RX_EN  0x1b0b0
…
                        MX6UL_PAD_ENET1_TX_CLK__ENET1_REF_CLK1      0x4001b031
                        /*johnli add mdio support*/
/*注意 MII 接口，需要配置成连接到 ENET1 上*/
                        MX6UL_PAD_GPIO1_IO07__ENET1_MDC             0x1b0b0
                        MX6UL_PAD_GPIO1_IO06__ENET1_MDIO      0x1b0b0
                        MX6UL_PAD_SNVS_TAMPER9__GPIO5_IO09    0x17059 /*phy reset*/
                        /*end*/
                >;
        };
```

i.MX6ULL 应该注意到 reset pin PAD_SNVS_TAMPER9 是属于 snvs 域，所以修改方式如下：

首先，在 iomux_snvs 域中增加 TAMPER9 GPIO IOMUX，注意是使用 i.MX6ULL PAD。

```
&iomuxc_snvs {
…
pinctrl_ent1_reset_b: enet1_reset_grp {
                fsl,pins = <
                    MX6ULL_PAD_SNVS_TAMPER9__GPIO5_IO09  0x17059 /*johnli add for enent reset*/
                >;
            };
…
}
```

其次，移除掉原本 iomux 定义：

```
pinctrl_enet1: enet1grp {
                fsl,pins = <
                    …
                        /*  MX6UL_PAD_SNVS_TAMPER9__GPIO5_IO09    0x17059 */ /*phy reset*/

                >;
            };
```

最后，增加 pinctrl:

```
&fec1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet1
            &pinctrl_ent1_reset_b>;
```

驱动测试：

- 启动信息

```
Configuring network interfaces... fec 2188000.ethernet eth0: Freescale FEC PHY driver [Micrel KSZ808
1 or KSZ8091] (mii_bus:phy_addr=2188000.ethernet:02, irq=-1)
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
```

- 硬件连接

网线一端连接 i.MX6UL/ULL EVK 扩展串口板的 J2144 RJ45 接口，另一端连接路由器。

- 测试命令

未插网线：

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:04:9F:03:FD:7D
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

插上网线：

```
root@imx6ulevk:~# fec 2188000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
```

重敲 ifconfig 命令：

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:04:9F:03:FD:7D
          inet addr:10.192.245.63  Bcast:10.192.245.255  Mask:255.255.255.0
          inet6 addr: fe80::204:9fff:fe03:fd7d%1995830992/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:203 errors:0 dropped:0 overruns:0 frame:0
          TX packets:81 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21987 (21.4 KiB)  TX bytes:11456 (11.1 KiB)
```

增加 NXP PCF8591 I2C 转 ADC 芯片支持

i.MX6UL/ULL EVK 扩展串口板增加了 I2C 转 ADC 的 PCF8591 芯片，以展示如何支持 NXP 的 I2C 芯片，

PCF8591 的相关资料在 www.nxp.com/i2c -> I2C DACs and ADCs ->PCF8591->Documentation。

所以要为此开发相应驱动，Linux L4.1.15_2.0.3 已经有标准驱动：

```
drivers\hwmon\pcf8591.c
```

此驱动说明文件在\Documentation\hwmon\pcf8591

所以驱动需要增加进去：

PCF8591 ADC 驱动需要编译到内核中了，修改如下内核配置文件增加：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_SENSORS_PCF8591=y
```

PCF8591ADC 的驱动 Makefile 文件

```
vi drivers\hwmon\Makefile
obj-$(CONFIG_SENSORS_PCF8591)   += pcf8591.o
```

PCF8591 ADC 的驱动源代码文件是使用比较老的内核的 i2c 驱动格式，修改为符合 L4.1.15 要求的格式，如下：

//使用 module_i2c_driver 标准宏来替换掉以前的 module_init/exit

```
#if 0
static int __init pcf8591_init(void)
{
        if (input_mode < 0 || input_mode > 3) {
                pr_warn("invalid input_mode (%d)\n", input_mode);
                input_mode = 0;
        }
        return i2c_add_driver(&pcf8591_driver);
}
static void __exit pcf8591_exit(void)
{
        i2c_del_driver(&pcf8591_driver);
}
#else
 module_i2c_driver(pcf8591_driver);
#endif
#if 0
module_init(pcf8591_init);
module_exit(pcf8591_exit);
#endif
```

//修改后将 pcf8591_init 中的 input_mode 初始化代码改到 probe 函数中：

```
static int pcf8591_probe(struct i2c_client *client,
                        const struct i2c_device_id *id)
{
     …
#if 1
    if (input_mode < 0 || input_mode > 3) {
                pr_warn("invalid input_mode (%d)\n", input_mode);
                input_mode = 0;
        }
#endif
```

//最后，为驱动增加 of_match 表：

```
    static struct i2c_driver pcf8591_driver = {
            .driver = {
                    .name     = "pcf8591",
             .owner  = THIS_MODULE,
             .of_match_table = pcf8591_of_match,
            },
            …
            .id_table = pcf8591_id,
};
static const struct of_device_id pcf8591_of_match[] = {
//compatible 必须和 DTS 中的 compatible 一致：
      { .compatible = "nxp,pcf8591", },
      { }
};
MODULE_DEVICE_TABLE(of, pcf8591_of_match);
```

i.MX6UL/ULL EVK 扩展串口板的支持如下：

PCF8591 连接到 i.MX6UL/ULL 的 I2C4，默认 i.MX6UL/ULL EVK 板的 BSP 并没有支持 I2C4，需要增加上去：

```
     &i2c4 {
            clock_frequency = <100000>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_i2c4>;
            status = "okay";
/*pcf i2c slave address is [1][0][0][1][a2=0][a1=0][a0=0][rw] 根据硬件配置得到 i2c slave adress*/
            pfc8591@48 {
                    compatible = "nxp,pcf8591";
                    reg = <0x48>;
                    status = "okey";
            };
};
```

I2C4 的 IOMUX 配置为：

```
pinctrl_i2c4: i2c4grp {
                        fsl,pins = <
                                MX6UL_PAD_ENET2_RX_EN__I2C4_SCL 0x4001b8b0
```

```
                    MX6UL_PAD_ENET2_TX_DATA0__I2C4_SDA 0x4001b8b0
            >;
        };
```

驱动测试：

- SYS 文件节点

  PCF8591 被注册成一个 hwmon 驱动，所以其 SYS 节点如下：

```
root@imx6ulevk:/sys/class/hwmon/hwmon0/device# ls
driver  in0_input  in2_input  modalias  of_node     out0_output  subsystem
hwmon   in1_input  in3_input  name      out0_enable  power       uevent
```

- 硬件连接

  UART7~8 插座使用了 PCF8591 的 ADC。为管脚 2，ADC 的参考电平是 3.3V。插座的管脚 1 的 3.3V 电源，管脚 4 是地。所以测量最小值可以连接到地，测量最大值可以连接到 3.3V。测量中间值时我们跳线连接到 i.MX6UL/ULL EVK CPU 板上的 TP702 上。

- 测试命令(连接 UART7 SLOT 的 ADC：管脚 2)

  接地：

```
root@imx6ulevk:/sys/class/hwmon/hwmon0/device# cat in0_input
0
```

  接 3.3V：

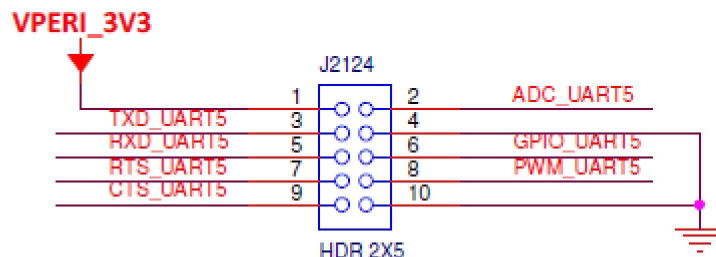
```
root@imx6ulevk:/sys/class/hwmon/hwmon0/device# cat in0_input
2540 /*连接 3.3V 是最高电压了，所以理论上读出来的值是 0xFF=255，所以 254 表示是接近最大值*/
```

  接中间电压：

```
root@imx6ulevk:/sys/class/hwmon/hwmon0/device# cat in0_input
1100  /*(110/255)*3.3V=1.424V*/
```

增加 NXP PCA9555A I2C 转 GPIO 芯片支持(rework 支持)

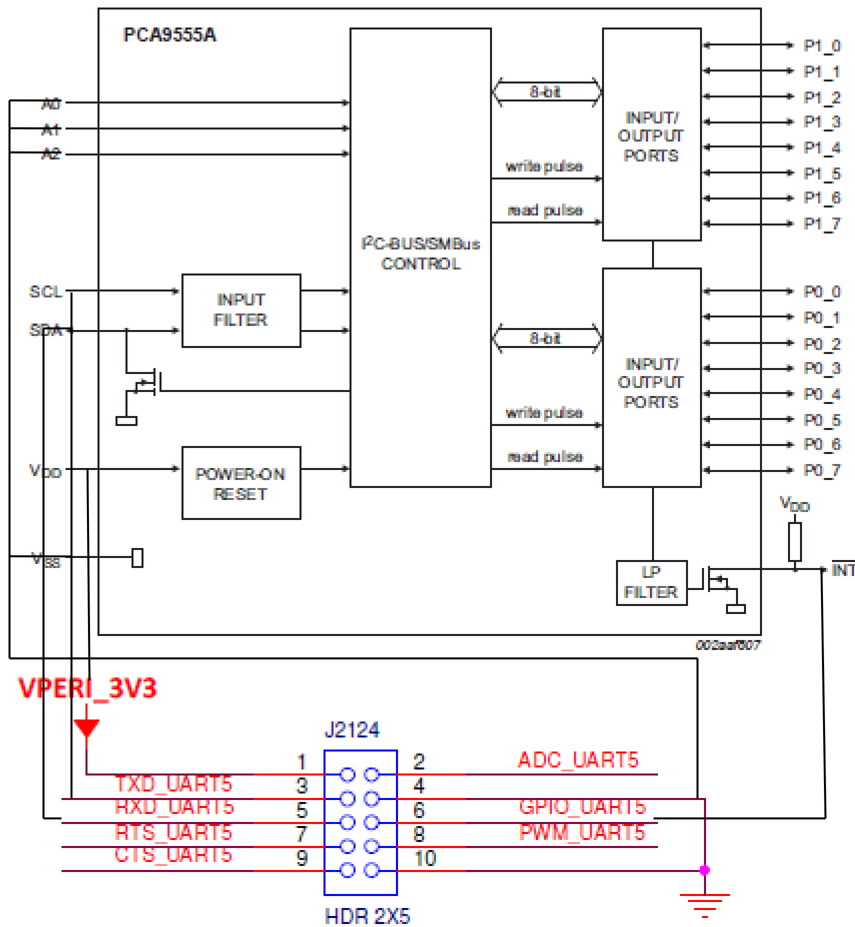
i.MX6UL/ULL EVK 扩展串口板的 UART5 接口，如下所示：



本身包含了一个 3.3V 电源，一个地，5 个 可配的 GPIOs(ADC/PWM/RTS/CTS 管脚也可以配置成 GPIO)，和一个 I2C 接口(TXD_UART5/RXD_UART5 也可以配置成 I2C2)。所以我们利用这一个接口来 rework 跳线连接来测试 NXP 的 SIP I2C 设备芯片，本章说明如何测试 PCA9555A，I2C 转 GPIO 桥芯片。

PCA9555A 的相关资料在 www.nxp.com/i2c -> I2C General Purpose I/O ->PCA9555A->Documentation。

我们选择 PCA9555APW，TSSOP24 封装，硬件 rework 如下(I2C 需要电阻上拉)：



实际 rework 照片如下：

Linux L4.1.15_2.0.3 已经有标准驱动：

Drivers\gpio\gpio-pca953x.c

此驱动 binding 说明文件在\Documentation\devicetree\bindings\gpio\gpio-pca953x.c。

PCA9555A GPIO 扩展桥芯片的驱动 Makefile 文件：

```
vi drivers\gpio\Makefile
obj-$(CONFIG_GPIO_PCA953X) += gpio-pca953x.o
```

PCA9555A GPIO 扩展桥芯片的驱动已经默认编译到内核中了，如下：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_GPIO_PCA953X=y
```

原因是 i.MX6X sabreauto 板所使用的 max7310 I2C 扩展 GPIO 桥芯片就是使用的此驱动，其 dts 配置如下：

```
arch\arm\boot\dts\imx6qdl-sabreauto.dtsi
    max7310_a: gpio@30 {
            compatible = "maxim,max7310";
            reg = <0x30>;
            gpio-controller;
            #gpio-cells = <2>;
    };
    max7310_b: gpio@32 {
```

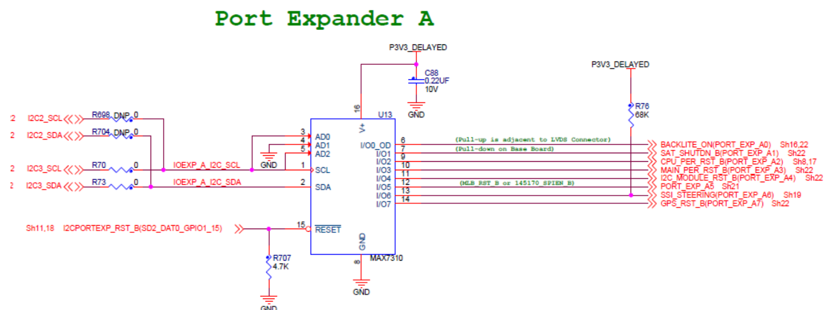
```
                    …
        };
        max7310_c: gpio@34 {
                    ….
        };
```

其 compatible 对应的驱动正好是：

```
drivers\gpio\gpio-pca953x.c
    static const struct of_device_id pca953x_dt_ids[] = {
        { .compatible = "nxp,pca9505", },
        { .compatible = "nxp,pca9534", },
        { .compatible = "nxp,pca9535", },
        { .compatible = "nxp,pca9536", },
        { .compatible = "nxp,pca9537", },
        { .compatible = "nxp,pca9538", },
        { .compatible = "nxp,pca9539", },
        { .compatible = "nxp,pca9554", },
        { .compatible = "nxp,pca9555", },
        { .compatible = "nxp,pca9556", },
        { .compatible = "nxp,pca9557", },
        { .compatible = "nxp,pca9574", },
        { .compatible = "nxp,pca9575", },
        { .compatible = "nxp,pca9698", },

        { .compatible = "maxim,max7310", },
        ...
        { }
};
```

其硬件设计如下：



所以，我们可以参考 i.mx6qdl-sabreauto.dtsi 的配置来为 i.MX6UL/ULL EVK 扩展串口板，增加 PCA9555A 的支持：

首先，我们为 rework 的 I2C 设备验证平台定义一个新的 DTS 文件：

```
pwd
~/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/u-boot-imx/2016.03-r0/git
(如果已经拷贝出源文件目录如下：pwd
/home/vmuser/imx6ulullexternaluart/sources/u-boot-2.0.3)
vi include/configs/mx6ul_14x14_evk.h
(vi include/configs/mx6ullevk.h /*i.mx6ull evk config file*/)
```

修改 " imx6ul-14x14-evk-externaluart.dtb " 为 "imx6ul-14x14-evk-externaluart-i2c.dtb"

(修改 "imx6ull-14x14-evk-externaluart.dtb" 为 "imx6ull-14x14-evk-externaluart-i2c.dtb" /*i.mx6ull evk config file*/)如下：

```
"if test $board_name = EVK && test $board_rev = 14X14; then " \
                    "setenv fdt_file imx6ul-14x14-evk-externaluart-i2c.dtb /*imx6ul-14x14-evk.dtb*/; fi; " \
```

然后，再次编译 uboot。

之后，为 i.MX6UL/ULL EVK 扩展串口板 I2C 验证创建一个新的 DTS 文件

```
pwd
/home/ysli/L4.1.15_2.0.0_ga/fsl-release-bsp/imx6ulevk_fb/tmp/work/imx6ulevk-poky-linux-gnueabi/linux-imx/4.1.15-r0/git
(如果已经拷贝出源文件目录如下：pwd
/home/vmuser/imx6ulullexternaluart/sources/kernal-2.0.3)
```

创建一个新的 dts 文件来支持 i.MX6UL/ULL EVK 扩展串口板 I2C 验证：

```
cp arch/arm/boot/dts/imx6ul-14x14-evk-externaluart.dts arch/arm/boot/dts/imx6ul-14x14-evk-externaluart-i2c.dts
(cp arch/arm/boot/dts/imx6ull-14x14-evk-externaluart.dts arch/arm/boot/dts/imx6ull-14x14-evk-externaluart-i2c.dts
/*i.mx6ull evk dtb*/)
```

编译此 DTB

```
make imx6ul-14x14-evk-externaluart-i2c.dtb
(make imx6ull-14x14-evk-externaluart-i2c.dtb /*i.mx6ull evk dtb*/))
```

其次：PCA9555A 连接到 i.MX6UL/ULL EVK 扩展串口板的 I2C2 上，是由 UART5 IOMUX 出来的，所以我们首先要去掉 UART5 的支持和 IOMUX，回复到 I2C2 的支持：

```
            pinctrl_uart5: uart5grp {
             fsl,pins = <
                 /*
                 MX6UL_PAD_UART5_TX_DATA__UART5_DCE_TX 0x1b0b1
                 MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX 0x1b0b1
```

```
                    MX6UL_PAD_GPIO1_IO08__UART5_DCE_RTS 0x1b0b1
                    MX6UL_PAD_GPIO1_IO09__UART5_DCE_CTS 0x1b0b1
                    */
                      >;
          };
     &uart5 {
        ….
         status = "disabled";
      };
                  pinctrl_i2c2: i2c2grp {
                           fsl,pins = <
                                    MX6UL_PAD_UART5_TX_DATA__I2C2_SCL 0x4001b8b0
                                    MX6UL_PAD_UART5_RX_DATA__I2C2_SDA 0x4001b8b0
                           >;
                  };
&i2c2 {
         clock_frequency = <100000>;
         pinctrl-names = "default";
         pinctrl-0 = <&pinctrl_i2c2>;
         status = "okay";
        …
}
```

UART5 串口座上的 GPIO KEY 驱动也去掉：

```
    /*
         power {
                           label = "Power Button";
                           gpios = <&gpio5 4 GPIO_ACTIVE_LOW>;
                           gpio-key,wakeup;
                           linux,code = <KEY_POWER>;
                  };
     */
          pinctrl_gpio_keys: gpio_keysgrp {
                           fsl,pins = <
                                    /*MX6ULL_PAD_SNVS_TAMPER4__GPIO5_IO04 0x1b0b0*/
                                    …
                           >;
                  };
```

最后，在 DTS 中增加 PCA9555A 的支持：

```
     &i2c2 {
```

```
        clock_frequency = <100000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c2>;
        status = "okay";
```

/*pca9555A i2c slave address is [0][1][0][0][a2=0][a1=0][a0=0][rw] 根据硬件配置得到 i2c slave address=20*/

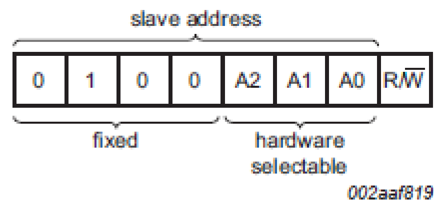


Fig 4. PCA9555A device address

A2, A1 and A0 are the hardware address package pins and are held to either HIGH (logic 1) or LOW (logic 0) to assign one of the eight possible slave addresses. The last bit of the slave address (R/W) defines the operation (read or write) to be performed. A HIGH (logic 1) selects a read operation, while a LOW (logic 0) selects a write operation.

```
                pca9555: gpio@20 {
                compatible = "nxp,pca9555";
                reg = <0x20>;
                gpio-controller;
                #gpio-cells = <2>;
        };
};
```

驱动测试：

- SYS 文件节点

PCA9555 被注册成一个 GPIO 驱动，所以其 SYS 节点如下：

```
/sys/class/gpio# ls
export  gpiochip0  gpiochip128  gpiochip32  gpiochip496  gpiochip64  gpiochip96  unexport
```

其中 gpiochip0~128 是 i.MX6ULL/ULL 本身有的 5X32=160 个 GPIOs,如下：

```
root@imx6ulevk:/sys/class/gpio# cd gpiochip128
root@imx6ulevk:/sys/class/gpio/gpiochip128# ls
base  device  label  ngpio  power  subsystem  uevent
root@imx6ulevk:/sys/class/gpio/gpiochip128# cat label
20ac000.gpio
```

```
root@imx6ulevk:/sys/class/gpio/gpiochip128# cat ngpio
32
root@imx6ulevk:/sys/class/gpio/gpiochip128# cat base
128
```

而 gpiochip496 是 pca9555 的 GPIOs,如下：

```
root@imx6ull14x14evk:/sys/class/gpio# cd gpiochip496
root@imx6ull14x14evk:/sys/class/gpio/gpiochip496# ls
base  device  label  ngpio  power  subsystem  uevent
root@imx6ull14x14evk:/sys/class/gpio/gpiochip496# cat label
pca9555
root@imx6ull14x14evk:/sys/class/gpio/gpiochip496# cat ngpio
16
root@imx6ull14x14evk:/sys/class/gpio/gpiochip496# cat base
496
root@imx6ull14x14evk:/sys/class/gpio/gpiochip496#
```

- 测试命令

Gpiochip496 是 pca9555 的 GPIO 控制器。如下：

从 base=496 开始，共有 16 个。

所以我们的操作命令应该是 echo (496+pca9555 gpio numbers) > /sys/class/gpio/export，如下为操作第二个 gpio：

```
    root@imx6ulevk:/sys/class/gpio/gpiochip496# echo 497 > /sys/class/gpio/export
root@imx6ull14x14evk:/sys/class/gpio/gpio497# cat direction
in
```

默认的 GPIO 方向是输入的，所以先改为输出：

```
root@imx6ull14x14evk:/sys/class/gpio/gpio497# echo out > direction
    cat /sys/calss/gpio/gpio497/value
1
    echo 0 > /sys/calss/gpio/gpio497/value
    echo 1 > /sys/calss/gpio/gpio497/value
```

然后，使用万用表量测相应管脚电平，可以看到电平会是 3.3V(value=1)，或是 0V(value=0)。

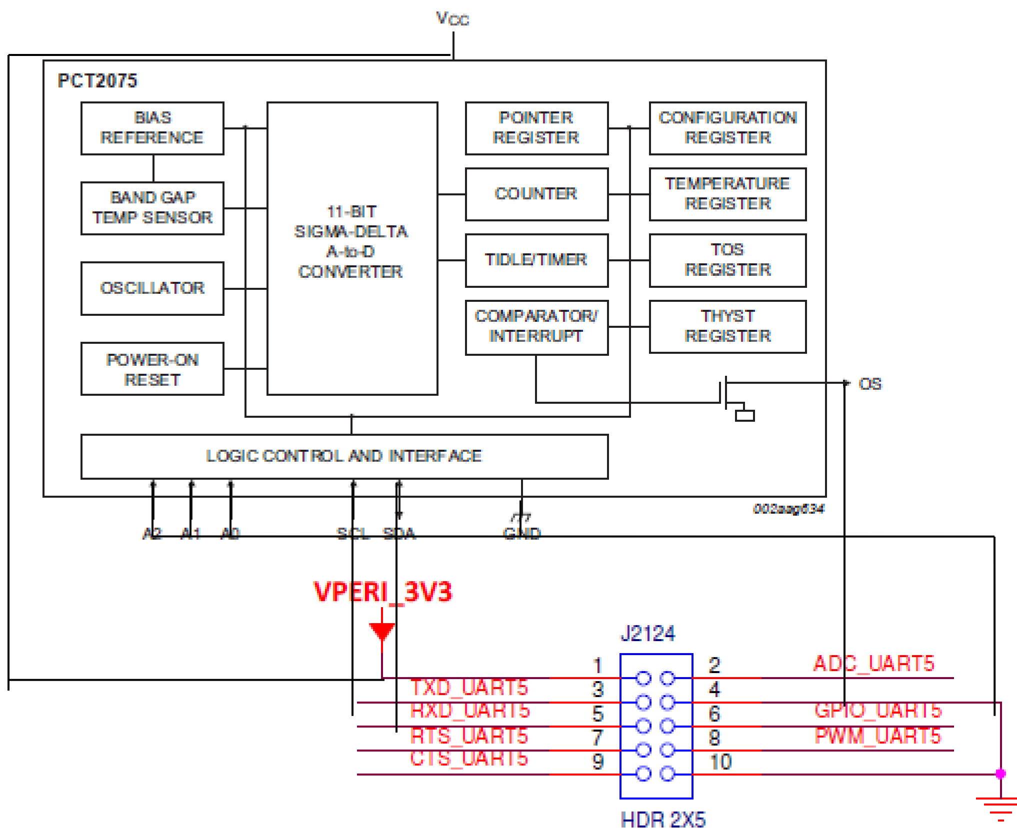
或是参考 GPIO_LED 的测试方式，使用 LED 灯来判断(由于 PCA9555 有弱上拉，可以输入 25mA，可以驱动 LED)。

增加 NXP PCT2075 I2C 温度传感器芯片支持(rework 支持)

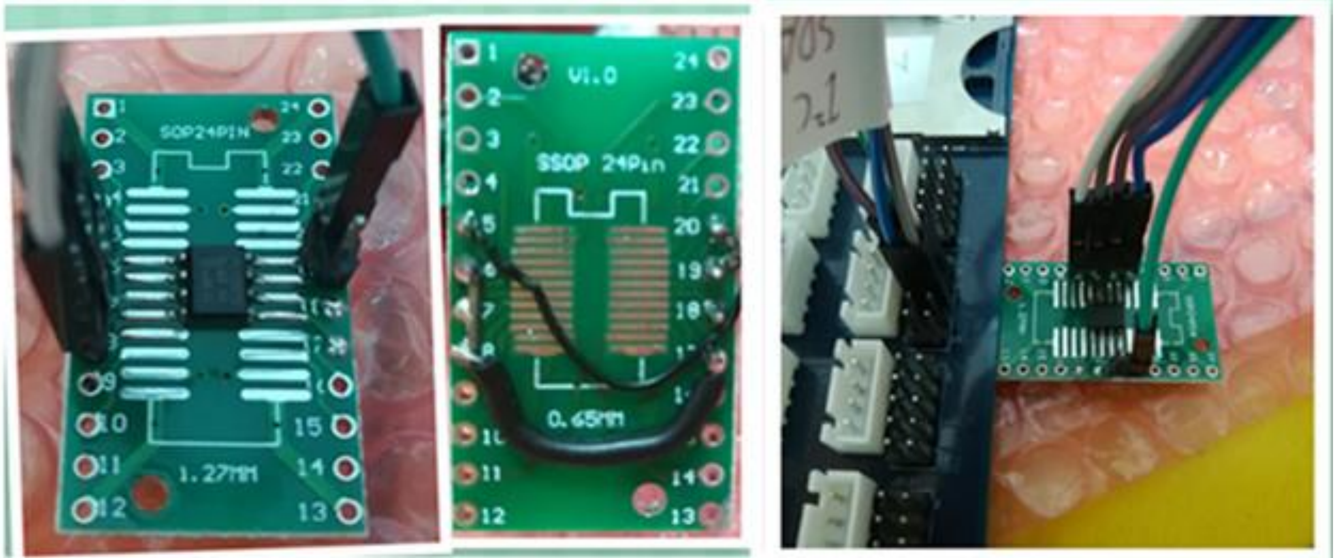
本章说明如何测试 PCT2075，I2C 接口温度传感器，PCT2075 为 LM75B 的升级版，大部分功能一致，寄存器地址，定义一致，只是多了一个寄存器，所以我们可以重用 LM75B 的驱动。

PCT2075 的相关资料在 www.nxp.com/i2c -> I2C Temperature Sensors->Filter I2C Temperature Sensors by Parameter ->PCT2075D->Documentation。

我们选择 PCT2075D，SO8 封装，硬件 rework 如下(I2C 需要电阻上拉)：



实际 rework 照片如下：



Linux L4.1.15_2.0.3 已经有标准驱动：

```
Drivers\hwmon\lm75.c
```

PCT2075(LM75B)温度传感器的驱动 Makefile 文件：

```
vi drivers\hwmon\Makefile
obj-$(CONFIG_SENSORS_LM75)      += lm75.o
```

PCT2075(LM75B)温度传感器芯片的驱动需要编译到内核中了，如下增加：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_SENSORS_LM75=y
```

然后重新编译内核。

如何创建一个新的 DTS 及如何改回到 I2C2 请参考上一章。

PCT2075(LM75B)温度传感器的驱动源代码文件是使用比较老的内核的 i2c 驱动格式，修改为符合 L4.1.15 内核要求的格式，如下

为驱动增加 of_match 表：

```
static struct i2c_driver lm75_driver = {
        .class          = I2C_CLASS_HWMON,
        .driver = {
                .name   = "lm75",
                . of_match_table =lm75b_of_match,
                .pm     = LM75_DEV_PM_OPS,
        },
        .probe          = lm75_probe,
```

```
        .remove          = lm75_remove,
        .id_table= lm75_ids,
        .detect          = lm75_detect,
        .address_list    = normal_i2c,
};
static const struct of_device_id lm75b_of_match [] = {
```

//compatible 必须和 DTS 中的 compatible 一致：

```
    { .compatible = "nxp,lm75b", },
    { }
};
MODULE_DEVICE_TABLE(of, lm75b_of_match);
```

最后，在 DTS 中增加 PCT2075(LM75B)的支持：

```
    &i2c2 {
        …
```

/* PCT2075(LM75B) i2c slave address is [1][0][0][1][a2=0][a1=0][a0=0][rw] 根据硬件配置得到 i2c slave address=48*/

Table 5. PCT2075 address table (SO8, TSSOP8, HWSON8 packages)

No.	Address pin coding			Slave address
	A2	A1	A0	
1	0	0	0	1001 000

```
        lm75b@48 {
                compatible = "nxp,lm75b";
                reg = <0x48>;
                status = "okey";
        };
};
```

驱动测试：

- 启动信息

```
lm75 1-0048: hwmon0: sensor 'lm75b'
```

- SYS 文件节点

  PCT2075(LM75B)被注册成一个 hwmon 驱动，所以其 SYS 节点如下：

```
root@imx6ull14x14evk:/sys/class/hwmon# ls
hwmon0  hwmon1  hwmon2
root@imx6ull14x14evk:/sys/class/hwmon# cd hwmon0
```

```
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# ls
device  name  of_node  power  subsystem  temp1_input  temp1_max  temp1_max_hyst  uevent
```

- 测试命令

```
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_input
24625
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_max
80000
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_max_hyst
75000
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0#
```

用手触摸芯片逐渐升温时测试结果如下：

```
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_input
28250
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_input
28875
root@imx6ull14x14evk:/sys/class/hwmon/hwmon0# cat temp1_input
29375
```

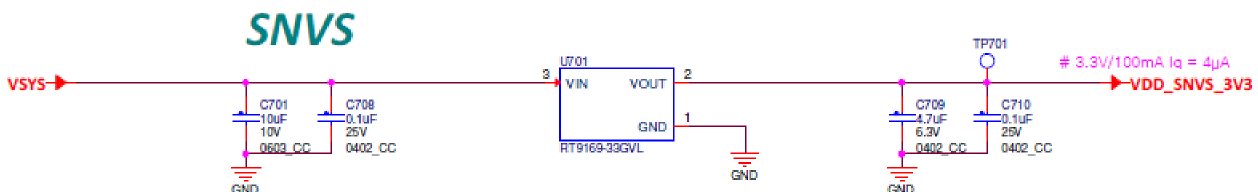
增加 NXP PCF8563 I2C RTC 支持(rework 支持)

本章说明如何测试 PCF8563，I2C 接口 RTC。我们选择 PCF8563T，SO8 封装。

PCF8563 的相关资料在 www.nxp.com/i2c -> I2C Real-Time Clocks(RTC)->Filter I2C Real-Time Clocks(RTC)by Parameter ->PCF8563T->Documentation。

硬件 rework 如下 Real-Time Clocks(RTC)需要电阻上拉)：

与其它 I2C 设备芯片电源 rework 方案不同，RTC 一般需要常备电源，所以我们将其 3.3V 电源单独跳线到 CPU 板的 VDD_SNVS_3V3 上，如下 TP701:

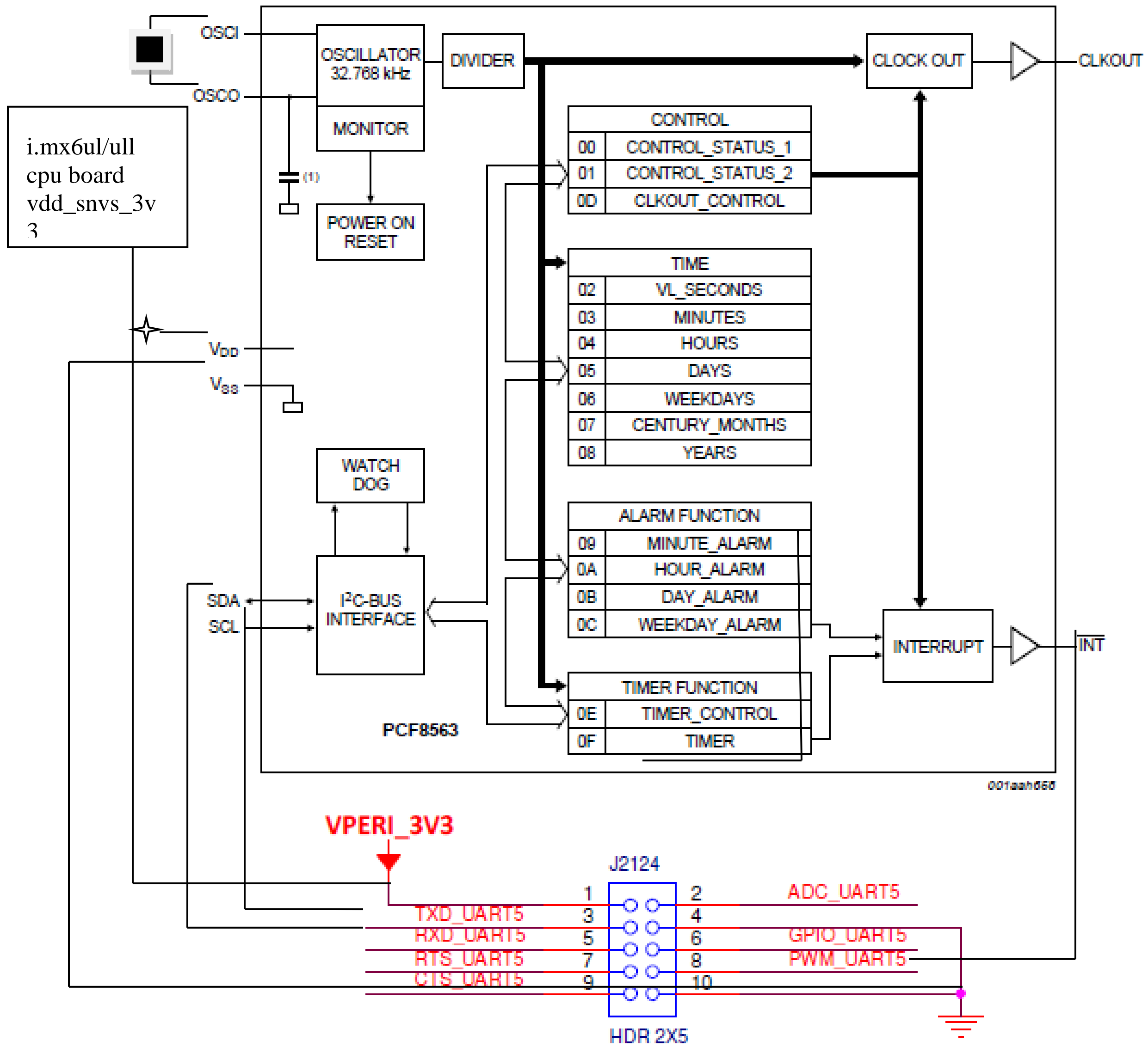


实际测试中也发现在电源不稳定的情况(刚开机，或是电源有纹波跌落)时，此 RTC 芯片会报有电源跌落，RTC 值可能不正确，或是干脆报 RTC 值无效：

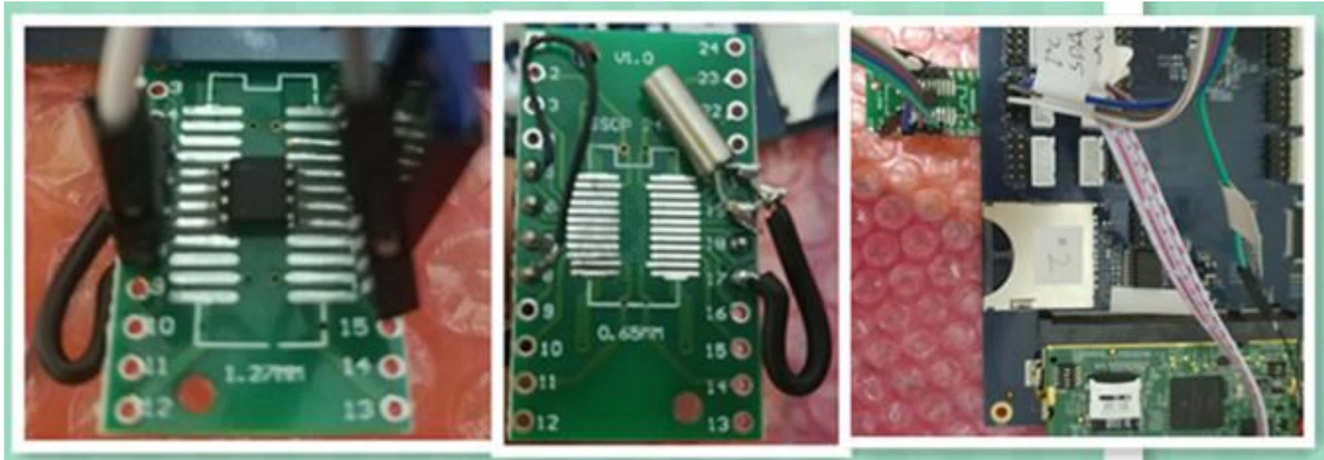
```
rtc-pcf8563 1-0051: low voltage detected, date/time is not reliable.
```

或：

rtc-pcf8563 1-0051: retrieved date/time is not valid.



实际 rework 照片如下：



Linux L4.1.15_2.0.3 已经有标准驱动：

Drivers\rtc\rtc-pcf8563.c

PCF8563 RTC 芯片的驱动 Makefile 文件：

```
vi drivers\rtc\Makefile
obj-$(CONFIG_RTC_DRV_PCF8563)  += rtc-pcf8563.o
```

PCF8563 RTC 芯片的驱动需要添加编译到内核中了，如下：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_RTC_DRV_PCF8563=y
```

然后重新编译内核。

如何创建一个新的 DTS 及如何改回到 I2C2 请参考上上章。

R4.1.15 内核中已经有支持 PCF8563 的 DTS 格式驱动：

```
vi drivers\rtc\rtc-pcf8563.c
MODULE_DEVICE_TABLE(i2c, pcf8563_id);
#ifdef CONFIG_OF
static const struct of_device_id pcf8563_of_match[] = {
	{ .compatible = "nxp,pcf8563" },
	{}
};
MODULE_DEVICE_TABLE(of, pcf8563_of_match);
#endif
static struct i2c_driver pcf8563_driver = {
	.driver		= {
		.name	= "rtc-pcf8563",
		.owner	= THIS_MODULE,
		.of_match_table = of_match_ptr(pcf8563_of_match),
```

```
        },
        .probe              = pcf8563_probe,
        .id_table= pcf8563_id,
};
```

所以，只需要在 DTS 中增加 PCF8563 的支持：

```
    &i2c2 {
        …
```

/*pcf8563 i2c slave address 如下：=0x51*/

Two slave addresses are reserved for the PCF8563:

Read: A3h (10100011)

Write: A2h (10100010)

The PCF8563 slave address is illustrated in Figure 17.

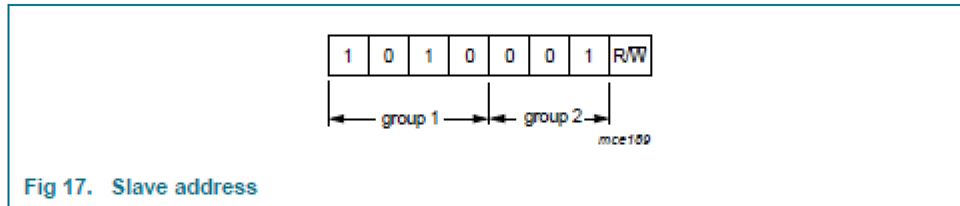


Fig 17. Slave address

```
        pcf8563@51 {
                compatible = "nxp,pcf8563";
                reg = <0x51>;
        };
};
```

最后，i.MX6UL/ULL 默认使用的内部 SNVS RTC，如果不需要，可以去掉，如果需要，此 RTC 会被注册为 rtc1(/dev/rtc1)，而 pcf8563 注册为 rtc0(/dev/rtc0)：

```
vi arch\arm\boot\dts\imx6ull.dtsi(imx6ul.dtsi)
/*
snvs_rtc: snvs-rtc-lp {
                    compatible = "fsl,sec-v4.0-mon-rtc-lp";
                    regmap = <&snvs>;
                    offset = <0x34>;
                    interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL_HIGH>, <GIC_SPI 20 IRQ_TYPE_LEVEL_HIGH>;
                };
*/
```

驱动测试：

- 启动信息

```
rtc-pcf8563 1-0051: chip found, driver version 0.4.3
rtc-pcf8563 1-0051: rtc core: registered rtc-pcf8563 as rtc0
snvs_rtc 20cc000.snvs:snvs-rtc-lp: rtc core: registered 20cc000.snvs:snvs-r as rtc1
Fri Mar 14 08:39:19 UTC 2014
```

- 设备文件

```
root@imx6ull14x14evk:/dev# ls -l rtc*
lrwxrwxrwx 1 root root      4 Mar 14  2010 rtc -> rtc0
crw------- 1 root root 254, 0 Mar 14  2010 rtc0
crw------- 1 root root 254, 1 Mar 14  2010 rtc1
```

- Sys 文件系统：

```
root@imx6ull14x14evk:/sys/class/rtc# ls
rtc0  rtc1
root@imx6ull14x14evk:/sys/class/rtc# cd rtc0/
root@imx6ull14x14evk:/sys/class/rtc/rtc0# ls
date  device   max_user_freq  power       subsystem  uevent
dev   hctosys  name           since_epoch  time      wakealarm
root@imx6ull14x14evk:/sys/class/rtc/rtc0# cat name
rtc-pcf8563
root@imx6ull14x14evk:/sys/class/rtc/rtc0# cd ..
root@imx6ull14x14evk:/sys/class/rtc# ls
rtc0  rtc1
root@imx6ull14x14evk:/sys/class/rtc# cd rtc1
root@imx6ull14x14evk:/sys/class/rtc/rtc1# ls
date  device   max_user_freq  power       subsystem  uevent
dev   hctosys  name           since_epoch  time      wakealarm
root@imx6ull14x14evk:/sys/class/rtc/rtc1# cat name
20cc000.snvs:snvs-r
```

- 配置硬件 RTC 时钟

```
root@imx6ull14x14evk:/dev# date/* 显示系统时间 */
Wed Mar 15 09:59:10 UTC 2017
root@imx6ull14x14evk:/dev# date -s "2007-08-03 14:15:00"
Fri Aug  3 14:15:00 UTC 2007
root@imx6ull14x14evk:/dev# hwclock -w -f /dev/rtc0/* 把系统时间写入 RTC */
```

```
root@imx6ull14x14evk:/dev# hwclock -f /dev/rtc0/*再次查看硬件 rtc 时间是否和系统中的一致*/
Fri Aug  3 14:15:24 2007  0.000000 seconds
```

- 外部 RTC 加内部 RTC 的应用场景：

由于 i.MX6UL/ULL 内部的 RTC 的电源功耗比较高， 如下：

SUSPEND (DSM)	• LDO_SOC is in bypass mode, LDO_ARM is in PG mode • LDO_2P5 and LDO_1P1 are shut off • CPU in power gate mode • DDR is in self refresh • All PLLs are power down • 24 MHz XTAL is off, 24 MHz RCOSC is off • All clocks are shut off, except 32 kHz RTC • High-speed peripheral are powered off	VDD_SOC_IN (0.9 V)	0.44	mA
		VDD_HIGH_IN (3.0 V)	0.03	
		VDD_SNVS_IN (3.0 V)	0.03	
		Total	0.58	mW

而 PCF8563 为：

- Low backup current; typical 0.25 μA at V_{DD} = 3.0 V and T_{amb} = 25 °C

所以在某些手持设备的情况下，需要使用钮扣电池，或是超级电容来长期保持本机 RTC 值的情况下，使用 i.MX6UL/ULL 本身的 RTC 功耗太高，会考虑使用外部 RTC。所以，在开机 时，脚本 /etc/init.d/hwclock.sh 会把外部 PCF8563 的 RTC 值 同步到内核中，如果还打开了 i.MX6UL/ULL 的内部 SNVS RTC，有可能需要同步到内部 RTC。

嵌入式Linux 内核(ARM)在系统启动时执行/etc/init.d/hwclock.sh 脚本,这个脚本会调用 hwclock 小程序读取 RTC 的值并设置系统时钟。

```
    [ "$UTC" = "yes" ] && tz="--utc" || tz="--localtime"
case "$1" in
        start)
…
                        if [ "$HWCLOCKACCESS" != no ]
                        then
                                if [ -z "$TZ" ]
                                then
                                    hwclock $tz --hctosys
                                else
                                  TZ="$TZ" hwclock $tz --hctosys
                                fi
                        fi
…
```

如果没有去掉内部 SNVS_RTC，可以修改为：

```
      if [ -z "$TZ" ]
                                then
                                    hwclock $tz –hctosys -f /dev/rtc0
```

```
                                hwclock $tz –systohc -f /dev/rtc1
                        else
```

修改之前系统启动后验证：

```
root@imx6ull14x14evk:~# date
Sat Mar 15 02:05:51 UTC 2014 /* 显示系统时间 */
root@imx6ull14x14evk:~# hwclock -f /dev/rtc0
Sat Mar 15 02:06:08 2014  0.000000 seconds 读外部 PCF8563 RTC 来对比*/
root@imx6ull14x14evk:~# hwclock -f /dev/rtc1
Thu Jan  1 00:02:13 1970  0.000000 seconds /*读 i.MX6UL/ULL 自身的 SNVS_RTC*/
```

修改之后系统启动后验证：

```
root@imx6ull14x14evk:~# date
Sat Mar 15 02:10:18 UTC 2014
root@imx6ull14x14evk:~# hwclock -f /dev/rtc0
Sat Mar 15 02:10:24 2014  0.000000 seconds
root@imx6ull14x14evk:~# hwclock -f /dev/rtc1
Sat Mar 15 02:10:26 2014  0.000000 seconds
```

- 外部 RTC 的主板掉电保持测试

由于 PCF8563 的 3.3V 电源是接在 VDD_SNVS_3V3 上，在 i.MX6UL/ULL EVK 的 CPU 板上，VDD_SNVS_3V3 是主电源直供的， 所以我们使用长按 reset 按键来关闭 CPU 电源，但保持 RTC 电源来模拟 RTC 的主板掉电保持，正式的设计需要为外部 RTC 设计独门的电源，电池(超级电容)电源，和电池(超级电容)充电电路。

测试结果如下：

第一次整板上电启动：

```
Reset cause: POR
rtc-pcf8563 1-0051: setting system clock to 2014-03-14 08:56:10 UTC (1394787370)
root@imx6ulevk:~# date
Fri Mar 14 08:58:22 UTC 2014
```

长按 reset 按键一段时间，然后释放 reboot 重启：

```
Reset cause: POR
rtc-pcf8563 1-0051: setting system clock to 2014-03-14 08:59:41 UTC (1394787581)
root@imx6ull14x14evk:~# date
Fri Mar 14 08:59:53 UTC 2014
```

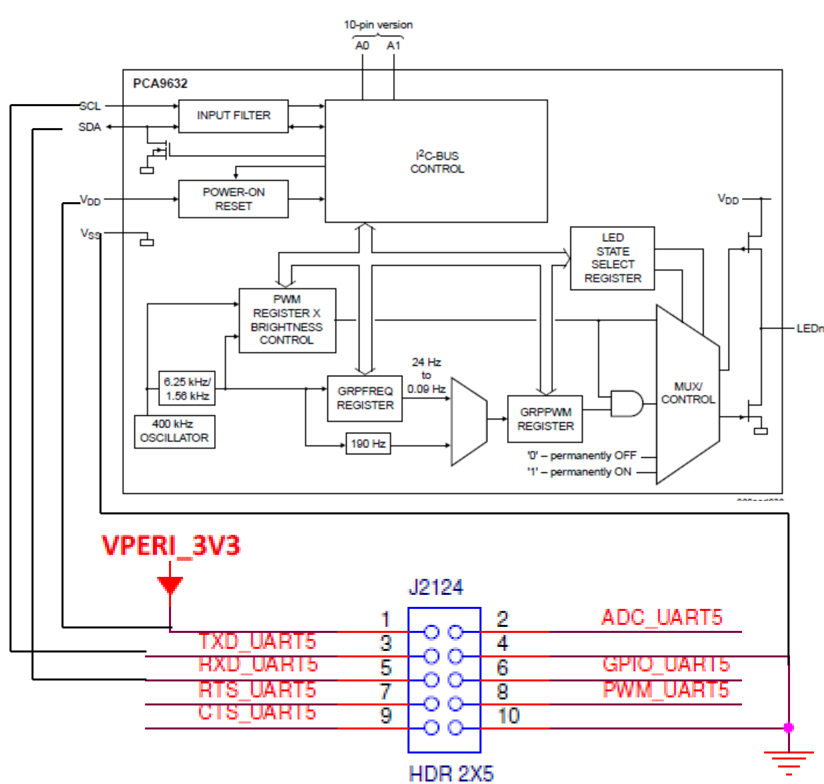
可见 RTC 值仍然保持。

增加 NXP PCA9632 I2C LED 控制器芯片支持(rework 支持)

本章说明如何测试 PCA9632，I2C LED 控制器，PCA9632～9635 只是控 制的 LED 数量不同，是复用同一个驱动的。

PCA9632~9635 的相关资料请联系 NXP。

我们选择 PCA9632DP1，TSSOP8 封装，硬件 rework 如下(I2C 需要电阻上拉)：



实际 rework 照片如下：

注意：硬件 rework 时在 led0~3 管脚上没有加上拉电阻，所以驱动中需要将管脚属性从 open-drain 修改为 push-pull，后面会详述。

Linux L4.1.15_2.0.3 已经有标准驱动：

```
drivers\leds\leds-pca963x.c
```

此驱动 binding 说明文件在\Documentation\devicetree\bindings\leds\pca963x.txt。

PCA9632，I2C LED 控制器驱动的 Makefile 文件：

```
vi drivers\leds\Makefile
obj-$(CONFIG_LEDS_PCA963X)          += leds-pca963x.o
```

PCA9632，I2C LED 控制器的驱动需要编译到内核中了，如下增加：

```
vi arch/arm/configs/imx_v7_defconfig
CONFIG_LEDS_PCA963X =y
```

然后重新编译内核。

如何创建一个新的 DTS 及如何改回到 I2C2 请参考上上上章。

R4.1.15 内核中已经有支持 PCA9632 的 DTS 格式驱动：

```
vi drivers\leds\leds-pca963x.c
static const struct of_device_id of_pca963x_match[] = {
	{ .compatible = "nxp,pca9632", },
	{ .compatible = "nxp,pca9633", },
	{ .compatible = "nxp,pca9634", },
	{ .compatible = "nxp,pca9635", },
	{},
};
static struct i2c_driver pca963x_driver = {
	.driver = {
		.name	= "leds-pca963x",
		.owner	= THIS_MODULE,
		.of_match_table = of_match_ptr(of_pca963x_match),
	},
	.probe	= pca963x_probe,
	.remove	= pca963x_remove,
	.id_table = pca963x_id,
};
module_i2c_driver(pca963x_driver);
```

所在，在 DTS 中增 PCA9632 的支持，请参考 binding 文件 documentation\devicetree\bindings\leds\pca963x.txt：

```
&i2c2 {
    …
```

/* PCA9632 8-pin 版本 i2c slave address is 如下：0x62*/

Fig 6. **Slave address**

```
    pca9632: pca9632@62 { compatible = "nxp,pca9632";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x62>;
        nxp,totem-pole;  /*/* default to open-drain unless totem pole (push-pull) is specified */ johnli change it because have
no external pull up
```

注意驱动默认是使用 open-drain，但是因为我们在 rework 时，led 管脚上没有加上拉电阻，所以需要修改为 push-pull 的，代码请参考：drivers\leds\leds-pca963x.c,函数 pca963x_dt_init

```
        /* default to open-drain unless totem pole (push-pull) is specified */
        if (of_property_read_bool(np, "nxp,totem-pole"))
                pdata->outdrv = PCA963X_TOTEM_POLE;
        else
                pdata->outdrv = PCA963X_OPEN_DRAIN;
*/
        red@0 { label = "red";
                reg = <0>;
                linux,default-trigger = "none"; };
        green@1 { label = "green";
                reg = <1>;
                linux,default-trigger = "none"; };
        blue@2 { label = "blue";
                reg = <2>;
                linux,default-trigger = "none"; };
```

```
        unused@3 { label = "unused";
                reg = <3>;
                linux,default-trigger = "none"; };
};
```

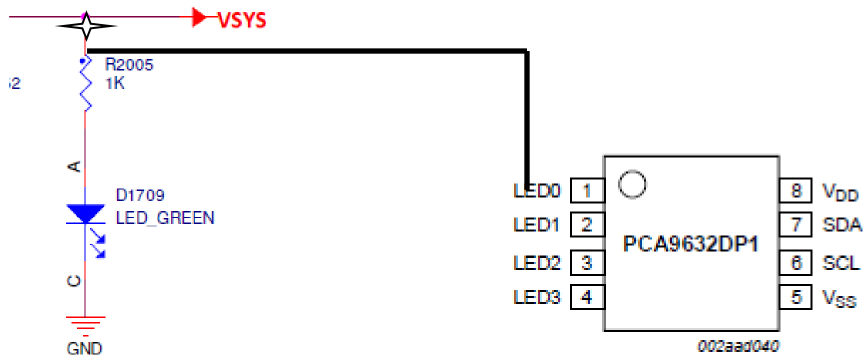
驱动测试：

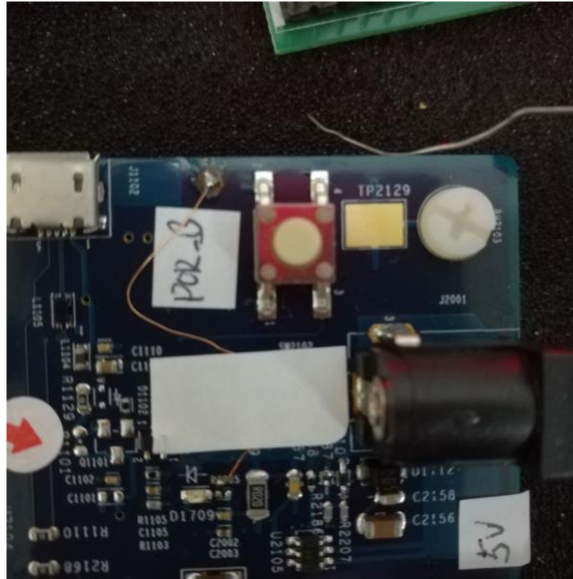
- SYS 文件系统节点

```
root@imx6ulevk:/sys/class/leds# ls
mmc0::  pca963x:blue   pca963x:red     user1  user3
mmc1::  pca963x:green  pca963x:unused  user2  user4
root@imx6ulevk:/sys/class/leds# cd pca963x\:red
root@imx6ulevk:/sys/class/leds/pca963x:red# ls
brightness  device  max_brightness  power  subsystem  trigger  uevent
root@imx6ulevk:/sys/class/leds/pca963x:red#
```

- 硬件连接

i.MX6UL/ULL EVK 扩展串口板上本身设计有一个电源灯，我们将连接到 VSYS 上限流电阻 R2005 断开，再连接到一个定位点做测试点，然后从 PCA9632DP1 LED 管脚(pin1~4)跳线过来就可以测试了，rework 原理图和实际连接如下：





- 测试命令

1. 测试开关

LED 我们默认设置为 linux,default-trigger = "none";

如果要测试开关，我们需要先将 trigger 设置为 gpio:

root@imx6ulevk:/sys/class/leds/pca963x:red# cat trigger

[none] rc-feedback nand-disk mmc0 mmc1 timer oneshot heartbeat backlight gpio

root@imx6ulevk:/sys/class/leds/pca963x:red# echo gpio > trigger

root@imx6ulevk:/sys/class/leds/pca963x:red# cat trigger

none rc-feedback nand-disk mmc0 mmc1 timer oneshot heartbeat backlight [gpio]

然后设置其 brightess

root@imx6ulevk:/sys/class/leds/pca963x:red# cat max_brightness

255

root@imx6ulevk:/sys/class/leds/pca963x:red# cat brightness

0

/*注意与 gpio-led 不同，pca9632 的 brightness 是取反值的，也就是 0 最亮，255 最暗，所以如果要关闭，如下*/

root@imx6ulevk:/sys/class/leds/pca963x:red# echo 255 > brightness

root@imx6ulevk:/sys/class/leds/pca963x:red# cat brightness

255

2. 测试闪烁

root@imx6ulevk:/sys/class/leds/pca963x:red# cat trigger

none rc-feedback nand-disk mmc0 mmc1 timer oneshot heartbeat backlight [gpio]

```
root@imx6ulevk:/sys/class/leds/pca963x:red# echo timer > trigger
root@imx6ulevk:/sys/class/leds/pca963x:red# cat trigger
none rc-feedback nand-disk mmc0 mmc1 [timer] oneshot heartbeat backlight gpio
root@imx6ull14x14evk:/sys/class/leds/pca963x:red# cat delay_off
500
root@imx6ull14x14evk:/sys/class/leds/pca963x:red# cat delay_on
500
```

增加 CH438 EIM 转串口芯片支持(delay)