SMARC Evaluation Kit2

(RM-F6DU1/SO1 with RP-101)

User's Manual

2016 August V1.0

Quick Start Guide

Here is a step by step guide to boot up your SMARC:

a. By **default**, the (Android) or (Linux OS) has been preloaded in the eMMC, (or SD card) of the SMARC demo kit. All you have to do is

- - Connect your SMARC with TV/LCD by using an **HDMI** cable,
- - Or connect your LVDS panel by using LVDS cable if you have them
- - Connect the device with 12V power input directly.

b. To make a recovery SD card, please refer to Chapter 2.

c. To use the root/ serial port debug function, please check Chepter3.2.1 (COM1 debug cable setup) information.

d. To boot up with an installed LVDS panel, please refer to Chapter 3.

Note: different LVDS panel will has different customization; please check with your sales.

e. For advanced users who would be building their own products, please refer to Chapter 3~5.

f. For special requests or assistance, please contact IBASE Sales.

TABLE OF CONTENTS

Quick Start Guide *2*

1. Introduction *6*

1.1. IBASE SMARC Starter Kit2 ..... **6**

1.2. RM-F6DU1-SMC Hardware Specifications **7**

1.3. Boards Dimension **9**

1.4. I/O View *** **10**

1.5. Optional Items **11**

1.6. Installing the SMARC **14**

2. Jumper setting on the Carrier Board *15*

3. Software Setup *37*

3.1. Make a Recovery SD Card **37**

3.2. Boot on the SMARC starter kit **39**

3.3. Parameter Setting on U-boot **40**

3.3.1. Preparation (debug console) 40

3.3.2. Display setting command For Android 41

3.3.3. Display setting for Linux 42

4. Carrier Board Design Guide *43*

4.1. Block Diagram *43*

4.2. Interfaces *44*

4.2.1. PCI Express **44**

4.2.2. SATAII *46*

4.2.3. USB *47*

4.2.4. Parallel RGB LCD Interface *48*

4.2.5. LVDS LCD Interface *49*

4.2.6. HDMI Interface *50*

4.2.7. Serial Camera Interface *51*

4.2.8. Ethernet *52*

4.2.9. eMMC *53*

4.2.10. SD Card Interface 54
4.2.11. I2S Interface 55
4.2.12. Asynchronous Serial Ports 56
4.2.13. CAN Bus 57
4.2.14. SPI Interface 58
4.2.15. I2C Interface 59
4.3. Layout recommendations 59
4.4. SMRC Module (RM-F6xx-SMC) Pin Out Table. 61
5. BSP User Guide (for advanced software engineer only) 64
5.1. Building SMARC BSP Source 64
5.1.1. Preparation 64
5.1.2. Installing Toolchain 64
5.1.3. Building u-boot 66
5.1.4. Building kernel 69
5.1.5. Copying u-boot, kernel to SD card 71
5.1.6. Copying Filesystem to SD card 71
5.1.7. Booting with your SD card 77
6. Appendix A– I2C, GPIO, watchdog reference code Coding. 78
6.1. How to use I2C in Linux 78
6.2. How to use GPIO in Linux 95
6.2.1. GPIO mapping table 95
6.2.2. GPIO sample code 95
6.2.3. How to use Watch dog in Linux 96
7. Appendix B - i.MX6 CPU ball out Table. 97
8. Appendix C : how to Flash the image to eMMC 104
9. iBASE SMARC RM-F6xx1-SMC series difference table 105
10. Appendix D –Useful links 106

Acknowledgments

Freescale™ is a *trademark of Freescale Semiconductor, Inc.*

ARM® Cortex™-A9 is a trademark of ARM Holdings, plc.

SGeT (Standardization Group for Embedded Technologies) is a technical and scientific association with its registered office in Munich.

SMARC ("Smart Mobility ARChitecture") is a versatile small form factor computer Module, defined by SGeT association

Android, name, logo, and other Android trademarks are property of Google Inc.

Linux, trademarks or marks include all trade and service marks and logos owned by the Linux Foundation.

All other product names or trademarks are properties of their respective owners.

1. Introduction

1.1. IBASE SMARC Starter Kit2

SMARC ('Smart Mobility ARChitecture') is a specification published by the Standardization Group for Embedded Technologies e.V. (SGET) for Computer-on-Modules (COMs). SMARC Computer-on-Modules are specifically designed for the development of compact low-power systems. Generally, SMARC modules are based on ARM processors and other low-power SoC architectures.

Measuring 82mm x 50mm, the Ibase RM-F6DU1-SMC SMARC module is integrated with an i.MX6 Dual 800Mhz automotive grade CPU that supports 2D/3D graphic acceleration and 1080p encode/ decode under Linux/Android BSP to allow easy OS upgrade and short time to market.

Measuring 170mm x 170mm (Mini ITX form factor), the Ibase RP-101-SMC (VDDIO=1.8v) carrier board is compatible with 82mm x 50mm - 82mm x 80mm standard SMARC form factors. Engineers can choose the required embedded IOs to verify developed software application under specified Operation System. Besides setting the default HDMI output with O.S preload in RM-F6DU1-SMC eMMC, IBASE can optionally provide driver-ready 3G module, WIFI module, touch panel, cable kit, power adaptor and accessories to speed up the evaluation cycle. ***

1.2. RM-F6DU1-SMC Hardware Specifications

RM-F6DU1-SMC Features

- SMARC Small Form Factor (82mm x 50mm) SoM
- i.MX 6Dual 800MHz Processor
- 1080p hardware encode/decode
- OpenGL ES 2.0 and OpenVG 1.1 hardware accelerators
- 1GB DDR3, 4GB eMMC on board
- 10/100/1000 MBit Ethernet
- Supports 24-bit Parallel LCD, LVDS, & HDMI
- Supports Linux 3.0, Android 4.3

Form Factor	SMARC™ (82mm x 50mm)
CPU	Freescale i.MX6 Dual Cortex-A9 Up to 800MHz with 512KB L2 cache
Memory	1GB DDR3 on board 4GB eMMC on board
Display	Supports 18/24-bit parallel LCD & LVDS Interface Supports HDMI Interface
Video Codec	Multi-format HD1080 video Decode and Encode
Audio Interface	I²S, SPDIF
LAN	10/100/1000 Mbit/sec
USB	2 x USB 2.0 port & 1x USB OTG Interface
Image Capture Interface	CSI Interface for MIPI Camera
Serial	4x UART, 2x SPI Interface
Media Interface	2 x High-Speed MMC/SDIO (MMC 8-bit, SDIO 4-bit)
PCIe	1x PCIe Interface
GPIO	12x GPIO
I²C	3x I²C / 4x I²C (SO1 Only)
CAN Bus	2x CAN2.0B
Operating Temperature	-40°C to +85°C (*Need heat-sink solution)
Board Connector	MXM3.0 314 pins
Operating System	Supports Linux Kernel 3.0, Android 4.3

• This specification is subject to change without prior notice.

RP-101-SMC – ***Carrier*** Board Specifications

RP-101-SMC Features

- For SMARC form factor modules
- 12V DC-in, reset, power, RTC function
- Supports GB LAN, audio, USB OTG, HDMI, COM (232/422/485) **@ Edge IO**
- Micro SD socket, Mini-PCIe with USB, SIM socket on board
- 2x isolated CAN transceiver, TTL, LVDS, HDMI, CSI-MIPI camera

Specifications

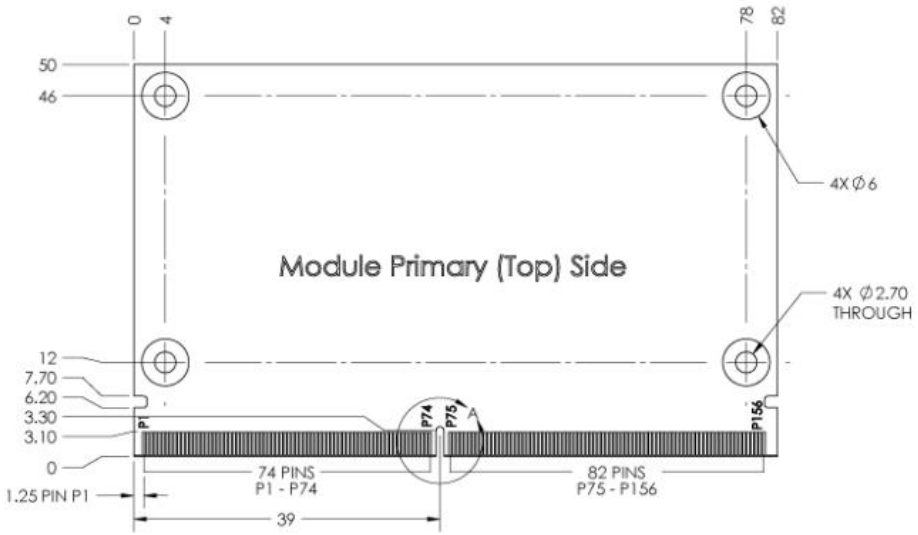
Form Factor	Standard Mini-ITX (170mm x 170mm)
Edge IO	1x GB LAN 1x headphone 1x MIC 2x USB 1x USB OTG 1x HDMI 1x COM (232/422/485)
Internal Headers / Connectors	2x CAN 1x parallel LCD 1x Single CH, 18/24 bit LVDS 1x LCD DDC (I2C) 1x LCD backlight power control header 1x CSI-MIPI 2x USB 2.0 8x GPIO 1x 2COM ports header 1x debug port 1x Mini-PCIe with USB 1x SIM socket 1x speaker Out 1x micro-SD
Jumpers, Switch & Buttons	1x boot select switch (SD/ eMMC) 1x reset button, 1x power button, 1x GPI button 1x (232/422/485 Selection) jumper 1x backlight power (5/12V) jumper

• ***This specification is subject to change without prior notice.***

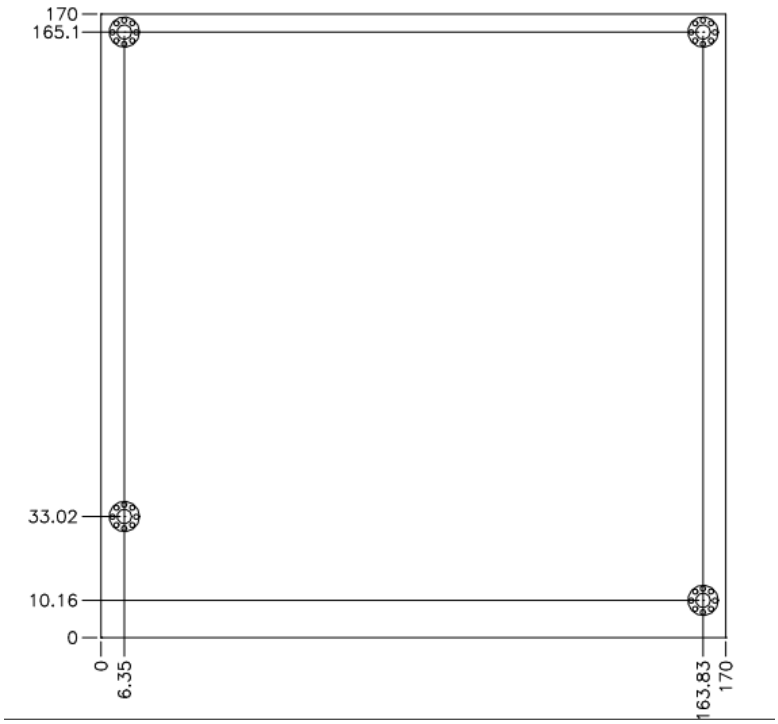
1.3. Boards Dimension

RM-F6DU1-SMC Dimensions

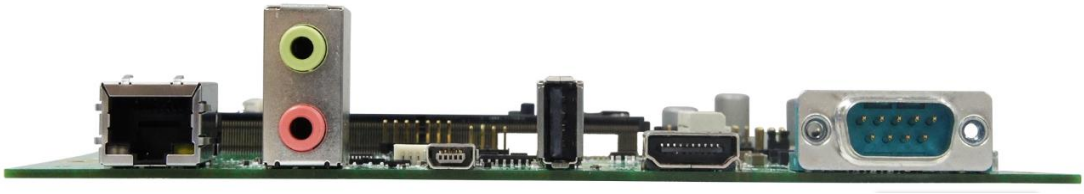
82mm x 50mm Module Outline



RP-101-SMC (MiniITX) 170mm x 170mm



1.4. I/O View ***

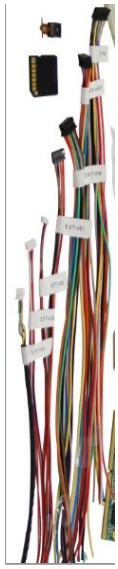


Item	Connector	Item	Connector
1	10/100/1000 LAN	5	HDMI Type A
2	Audio Jack	6	RS232/422/485
3	USB OTG	7	
4	USB Type A	8	

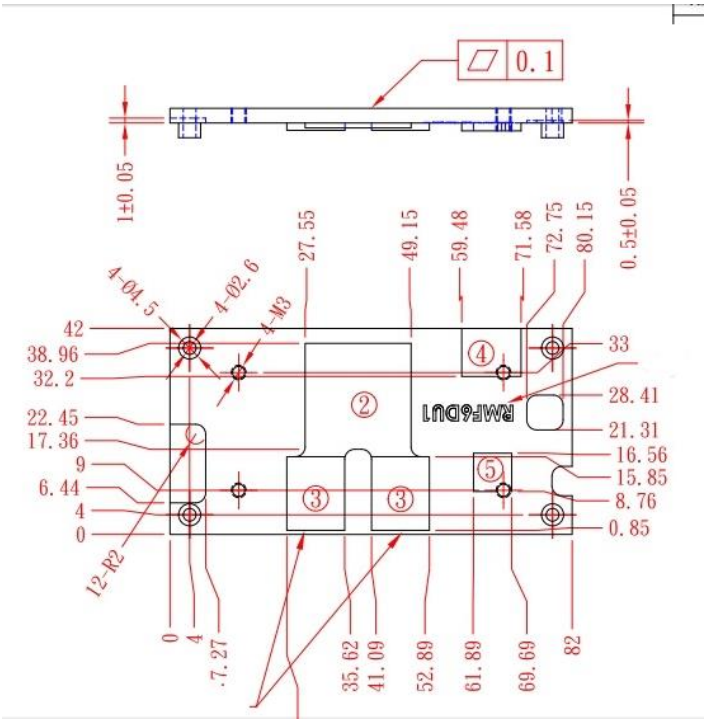
1.5. Optional Items

If you have further request for option items, please contact iBASE sales.

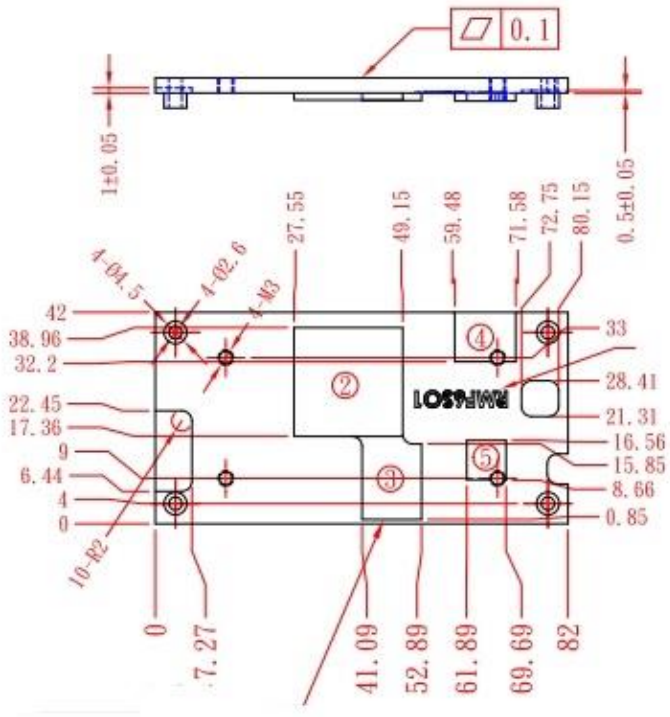
3G Solution	Description	Note
ZU 202	Wireless; 3.75G UMTS/HSPA [ZU202] RoHS (A008WIRELESS00520P)	
ZU 200	Wireless; 3.75G UMTS/HSPA & GPS module [ZU200] RoHS (A008WIRELESS00511P)	
Cable	Cable; Antenna-2 30CM P 2pcs (C501ANT0200300000P)	
Antenna	Antenna; 3G, P, 2pcs (A055ANT0921Q2P000P)	
COM Port Cable	**Description**	
COM & debug cable	Cable; for internal 3x COM ports box header ***COM 1 for debug console:*** ***- User can make your own debug cable too by checking Ch3.2.1 and CN15 connector.***	C501PK19100204000P
Power & LCD	**Description**	
Adaptor	12V, Power Adaptor	A005PS060WFSP0101P
LCD	10 inch. 1280 x 800 LCD	A003LCD0101010300
LCD cable.	LCD312 Cable	C501LCD3120302000P
BT solution	**Description**	
BSLIM2 A10	BLUETOOTH 4.0 BOARD	A008BTBSLIM201000P
Cable kit	**Description**	

For carrier board	USB-29 (C501USB2908303000P): USB 2.0 (USB1) EXT-458 (C501EXT4580301000P): Audio Speaker (CN14) EXT-459 (C501EXT4590301000P): RS232 (CN15) EXT-460 (C501EXT4600301000P): LCD DDC DATA/CLK (CN6) EXT-461 (C501EXT4610301000P): Touch (CN13) EXT-462 (C501EXT4620301000P): SPI (CN7) EXT-463 (C501EXT4630301000P): GPIO (CN5) EXT-464 (C501EXT4640301000P): CAN Bus (J2, J4) Others: ~~500Mega pixel MIPI Camera (A033MODULE0200000P):(J3)~~	

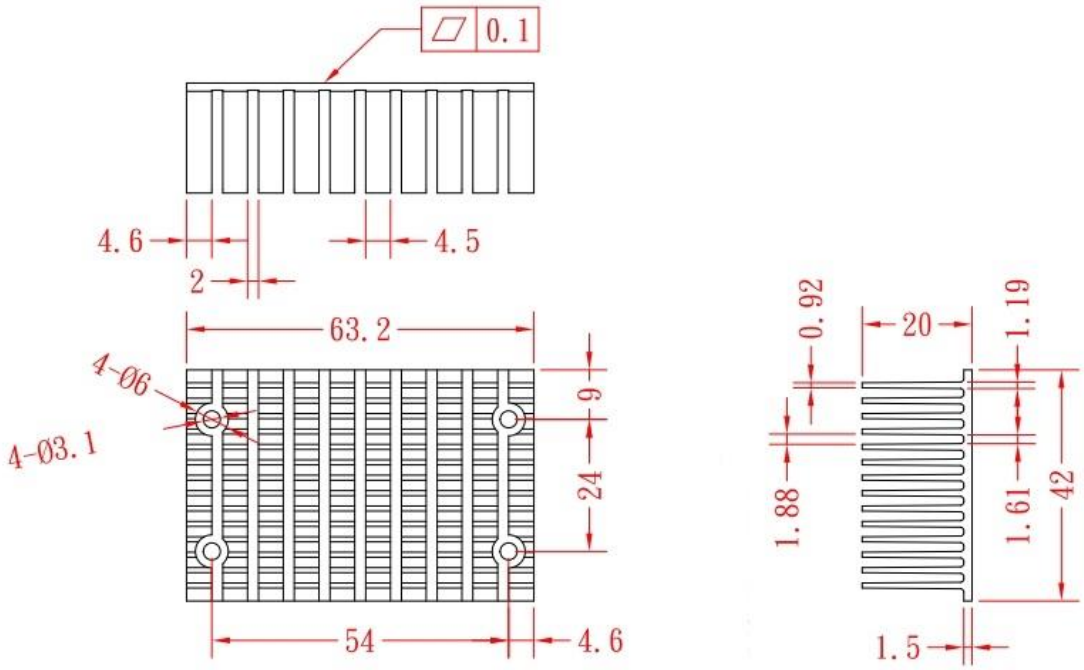
RM-F6DU1-SMC Heat spread reference (PN : H051HSRMF6DU1B10AP)



RM-F6SO1-SMC Heat spread reference (PN : H051HSRMF6SO1B100P)



RM-F6DU1-SMC/RM-F6SO1-SMC Heat Sink reference: PN (H051HSRMF6DU1BA0AP)



1.6. Installing the SMARC

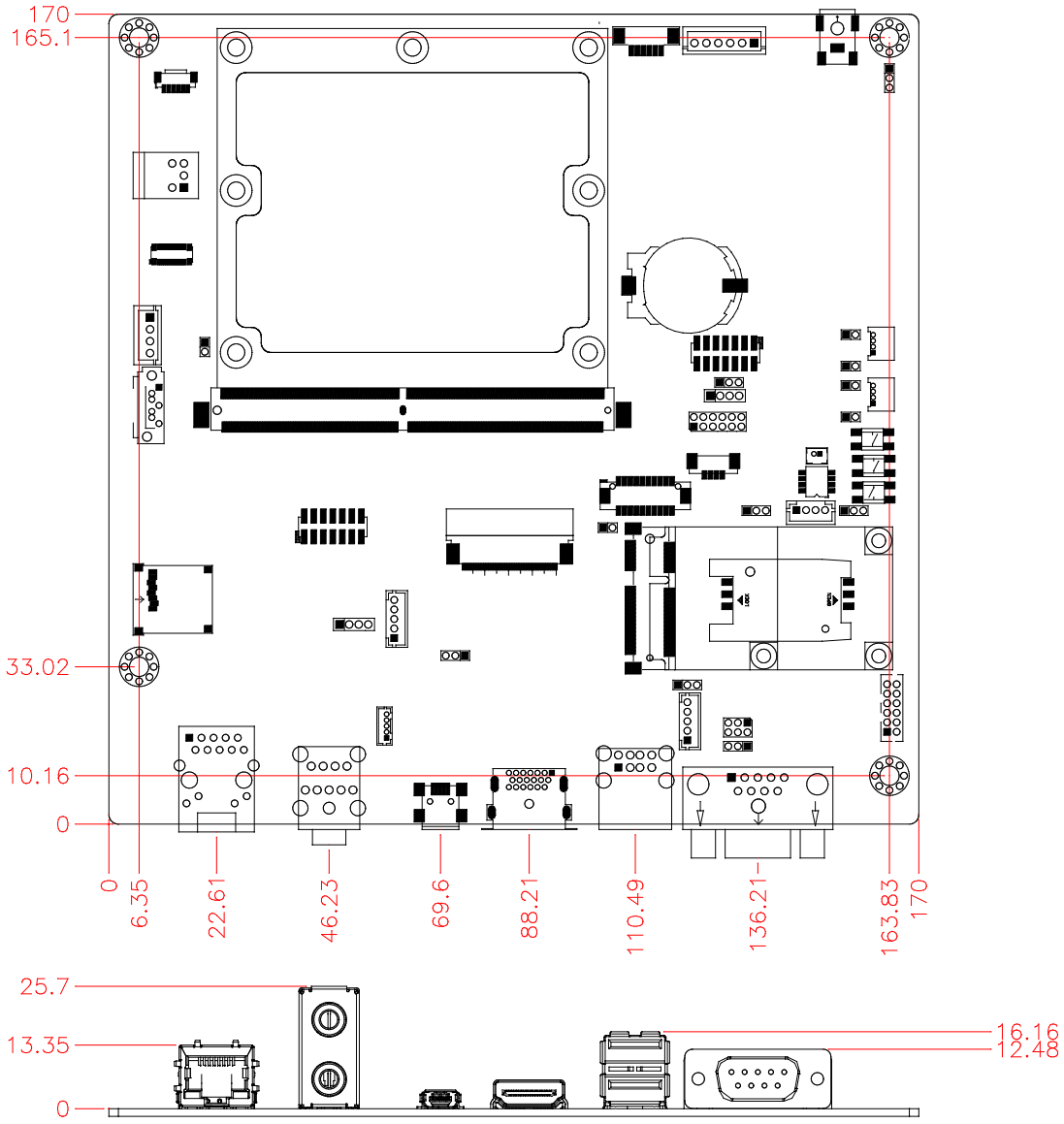
The MXM3.0 connector on RP-101-SMC supports the SMARC form factor (82mm x 50mm and 82mm x 70mm).

To install SMARC modules to the MXM slot on the board, please perform the following steps:

- Hold the SMARC module so that the golden fingers of the SMARC module aligned with the MXM connector.
- Gently push the SMARC module to the MXM connector in 45 degree angle position until the golden finger of SMARC completely touch the bottom of the slot.
- Gently press the SMARC module down and fix it with four screws.

2. Jumper setting on the Carrier Board

[Important] Please check the jumpers, DIP, buttons and switches on RP-101-SMC before doing the panel connection and boot up.



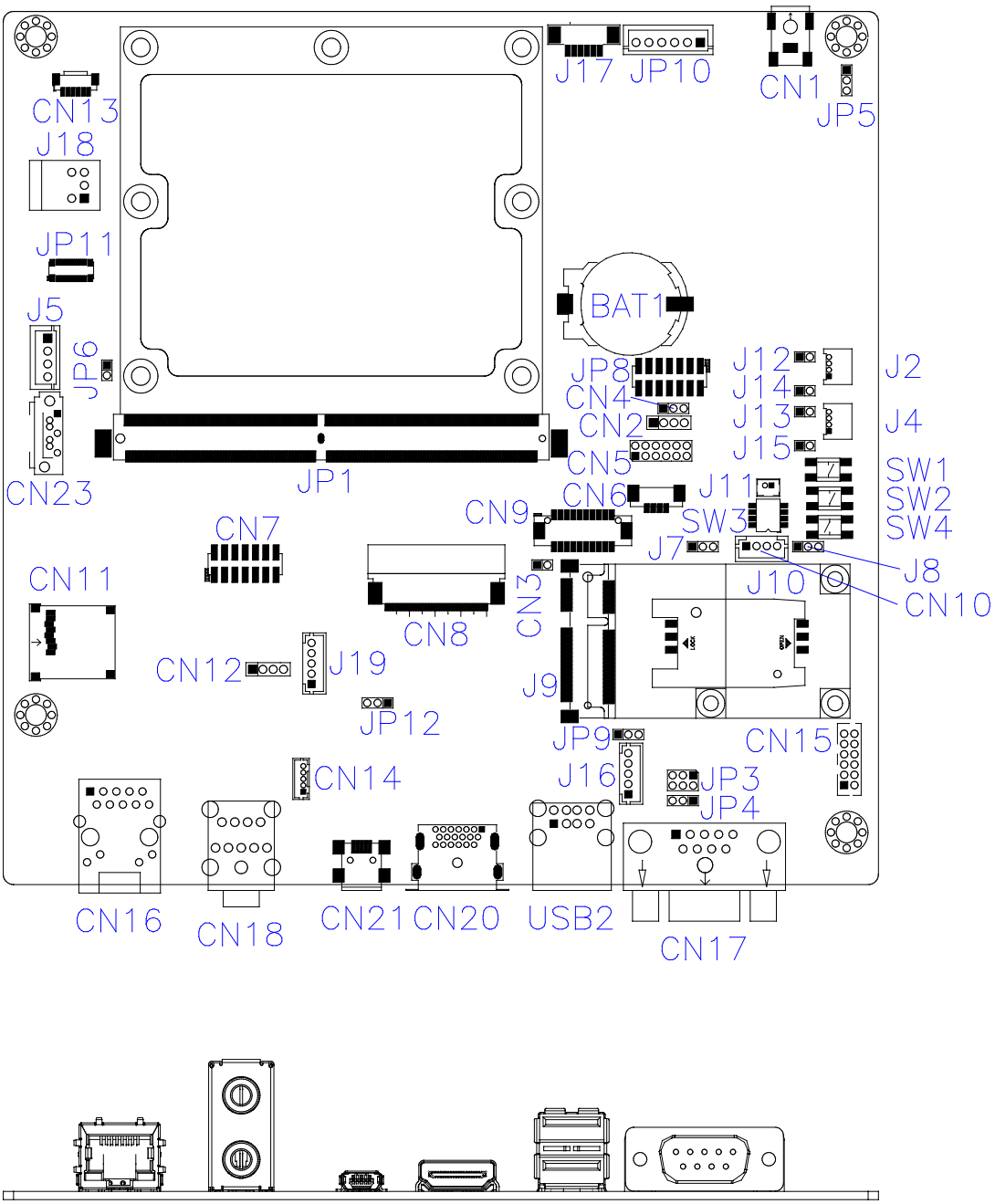
Connectors on RP101

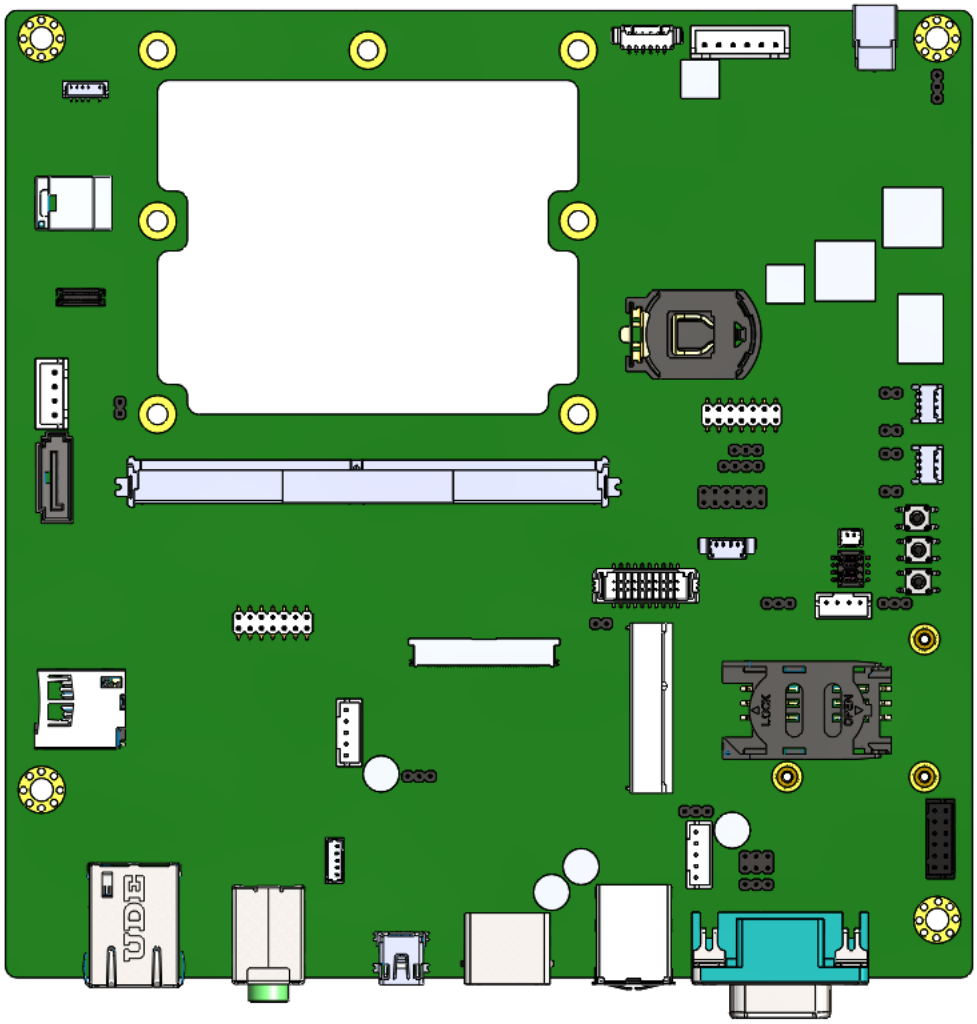
CN1	DC Power Jack
CN2	Power Management pin header
CN4	Charger Power Management GPIO pin header
CN5	GPIO Connector
CN6	LCD DDC CLK/DATA channel Connector
CN7	SPI pin header
CN8	TTL LCD Connector
CN9	LVDS Connector
CN10	LCD Backlight Power/Control Connector
CN11	Micro SD card Socket
CN12	4-wires Resistive Touch Connector
CN13	I2C Connector
CN14	Audio Speaker Out Connector
CN15	RS232 (COM1/COM2/COM5) Connector
CN16	LAN RJ45 Connector
CN17	RS232/422/485 Edge Connector
CN18	Audio Edge Phone Jack
CN20	HDMI Edge Connector
CN21	OTG mini-USB Edge Receptacle
CN23	SATA Connector

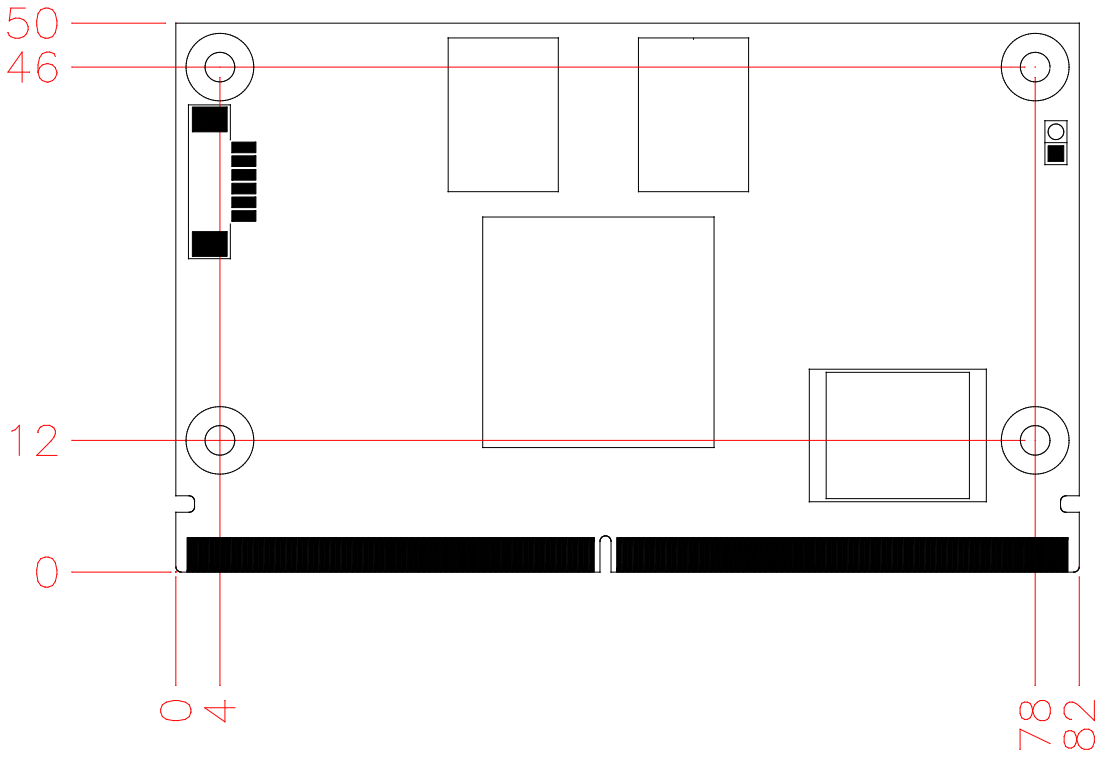
J2	CAN Bus Connector
J4	CAN Bus Connector
J5	SATA power Connector
J7	LCD VDD voltage select jumper
J8	LCD Backlight voltage select jumper
J9	mPCIe socket
J10	SIM card socket
J11	RESET Connector
J12	CAN bus 1 Terminator jumper
J13	CAN bus 2 Terminator jumper
J14	CAN bus 1 Terminator jumper
J15	CAN bus 2 Terminator jumper
J16	USB Connector
J17	MCU Programming Connector (Internal Debug Only)
J18	S/PDIF Connector
J19	USB Connector

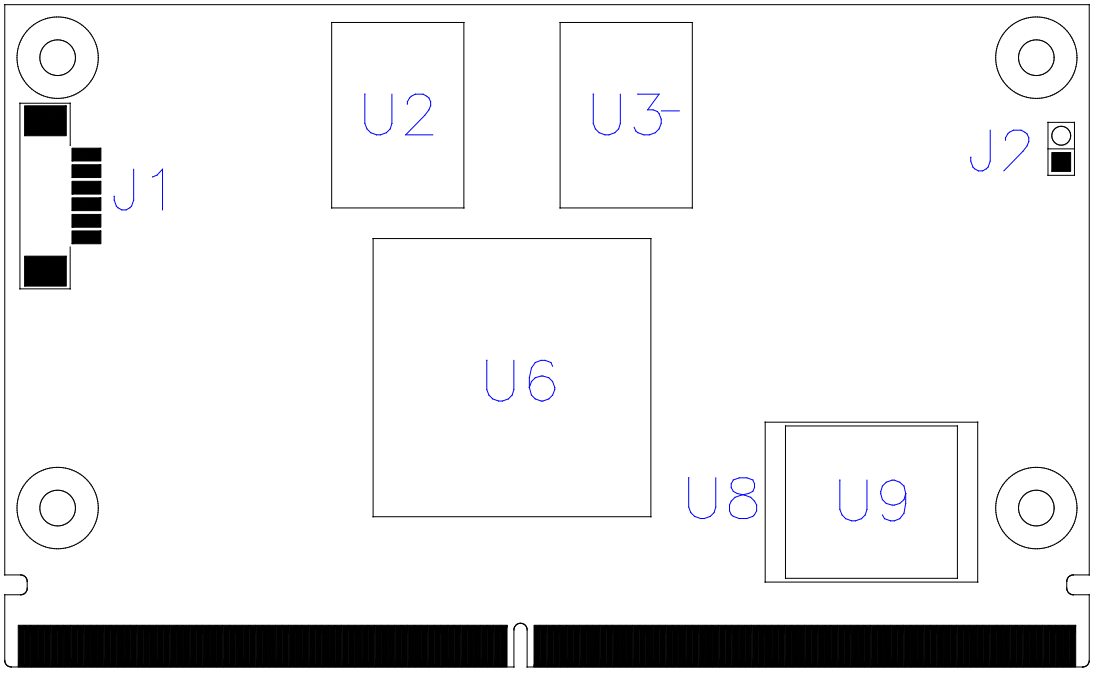
JP1	SMARC Socket
JP3	CN17 RS232/422/485 setting jumper
JP4	Edge 422/485 Terminator jumper
JP5	
JP6	SMARC signal enable jumper (Internal Debug Only)
JP8	Power management control/status connector
JP9	J16 USB connector voltage selector jumper
JP10	SMART Battery Connector (Internal debug only)
JP11	CSI Connector
JP12	J19 USB connector voltage selector jumper

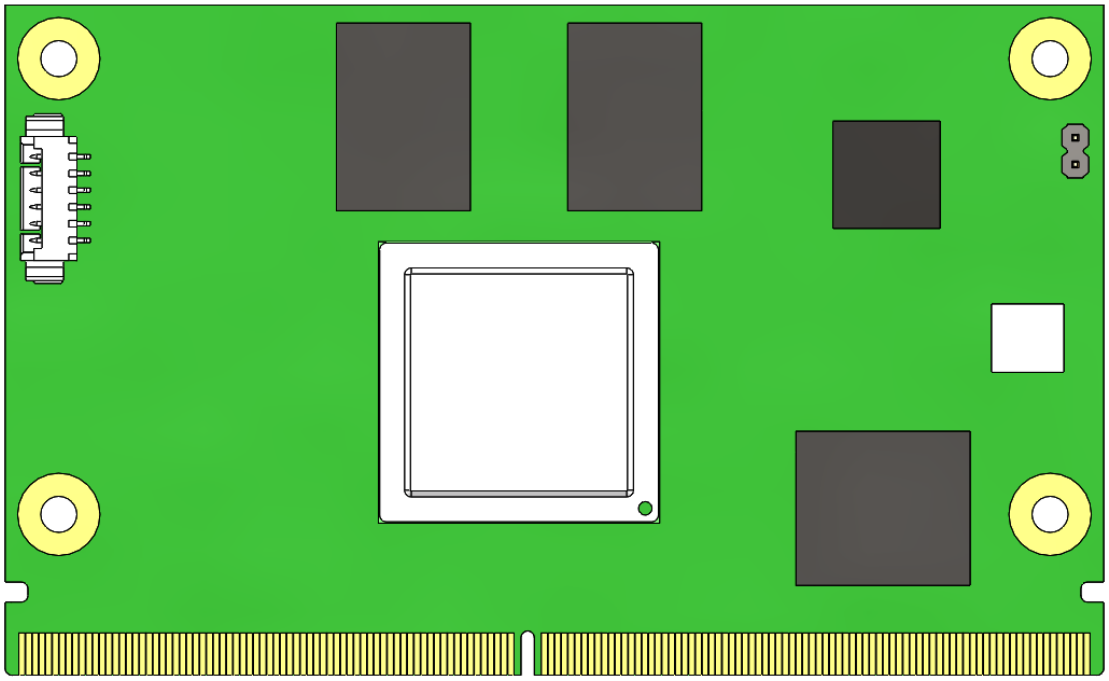
SW1	Power button
SW2	Reset button
SW3	SMARC boot device selector
SW4	GPIO_11 tach switch (CN5)





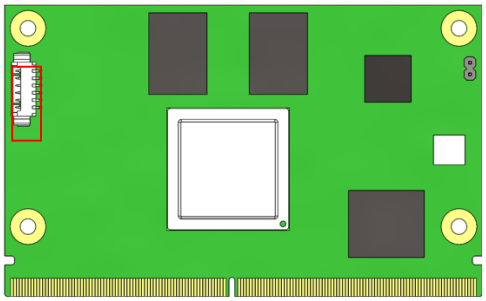






RM-F6DU1-SMC

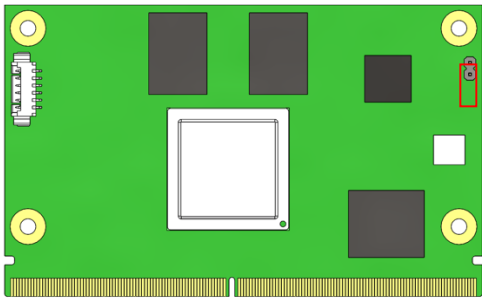
J1 : MCU JTAG, internal debug only



Pin #	Signal Name
1	+3.3V_SB
2	+3.3V_SB
3	+3.3V_SB
4	SBWTCK
5	SBWTDIO
6	GND

J2 : VDDIO PIN1,2 OPEN, for **VDDIO=3.3V** carrier board

PIN1,2 SHORT, for **VDDIO=1.8V** carrier board



Pin #	Signal Name
1	VDD_IO_SEL#
2	GND

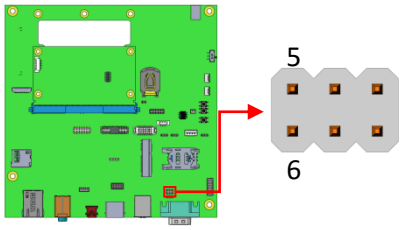
*note:

RM-F6SO1-SMC supports VDDIO=3.3V only;

RM-F6SDU1-SMC supports VDDIO=1.8V / 3.3V by jumper.

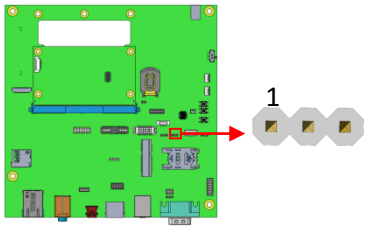
RP-101-SMC

JP3: Setting Jumper for CN17 (RS232/422/485)



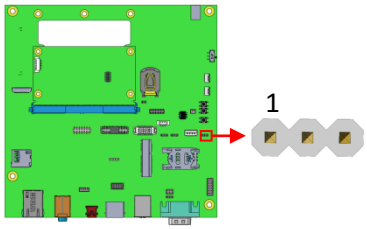
MODE	Jumper Setting
RS232	2-4, 3-5 Shorted (Default)
RS422	3-5, 4-6 Shorted
RS485	1-3, 4-6 Shorted

J7: LCD_VDD Select



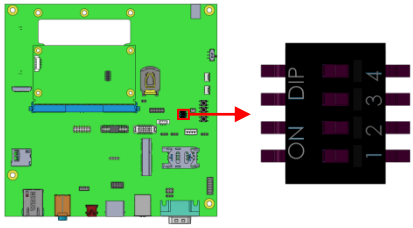
LCD_VDD	Jumper Setting
3.3V	2-3 Shorted (Default)
5V	1-2 Shorted

J8: BKLT_PWM Select



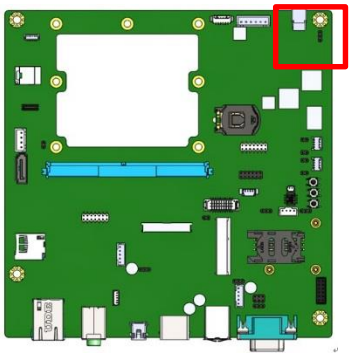
BKLT_PWM	Jumper Setting
3.3V	2-3 Shorted (Default)
5V	1-2 Shorted

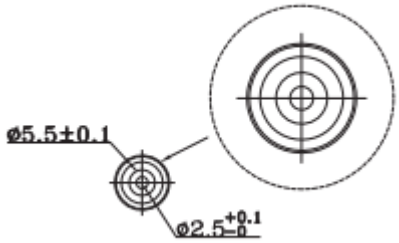
SW3: Boot Select



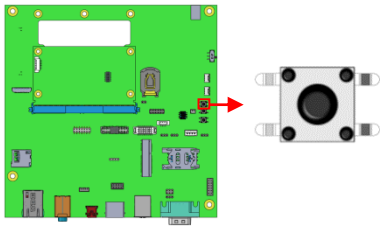
Boot Source	SW3_PIN1	SW3_PIN2	SW3_PIN3	SW3_PIN4
Carrier SATA	1	1	1	X
Carrier SD	0	1	1	X
Carrier eMMC	1	0	1	X
Module eMMC	1	0	0	X

CN1: DC Power Jack

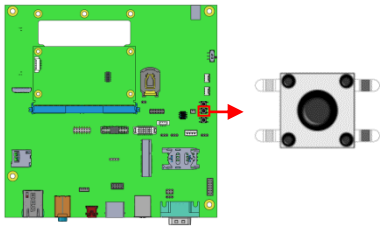




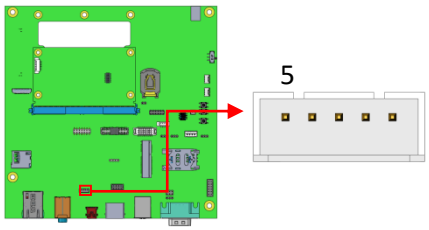
SW1: Power Button



SW2: Reset Button

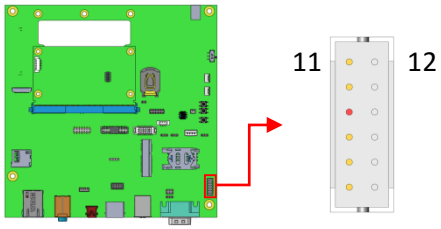


CN14: Audio Speaker out (E-CALL_0110-26110050)



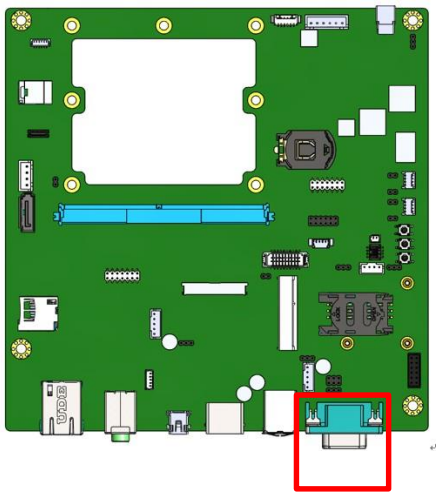
Pin #	Signal Name
1	SPKL+
2	SPKL-
3	GND
4	SPKR-
5	SPKR+

CN15: RS232 x 3 (DF11-12S-PA66H)



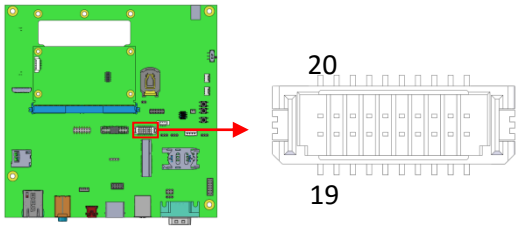
Signal Name	Pin #	Pin #	Signal Name
+5V	1	2	TX1
TX5	3	4	RX1
RTS5#	5	6	GND
RX5	7	8	TX2
CTS5#	9	10	RX2
GND	11	12	GND

CN17: RS232/422/485



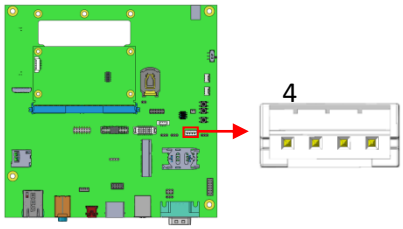
Pin #	RS232	RS422	RS485
1	DCD	TXD-	D-
2	RXD	TXD+	D+
3	TXD	RXD+	
4	DTR	RXD-	
5	GND		
6	DSR		
7	RTS		
8	CTS		
9			

CN9: LVDS (DF13-20DP-1.25V)



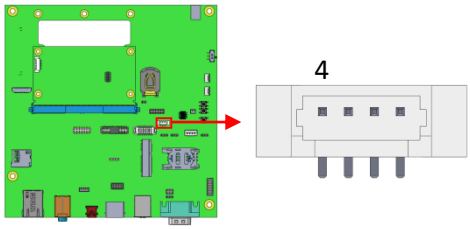
Signal Name	Pin #	Pin #	Signal Name
TX0+	1	2	TX0-
GND	3	4	GND
TX1+	5	6	TX1-
GND	7	8	VDD
TX3+	9	10	TX3-
TX2+	11	12	TX2-
GND	13	14	GND
TXC+	15	16	TXC-
BKLT_PWM	17	18	VDD
BKLT_VCC	19	20	BKLT_VCC

CN10: LCD Backlight Power/Control (E-CALL 0110-161-040)



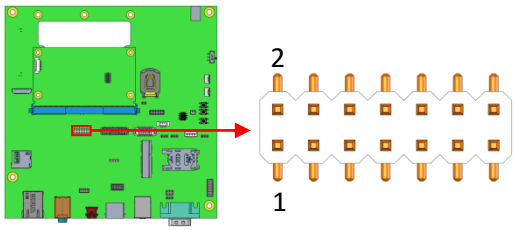
Pin #	Signal Name
1	BKLT_VCC
2	BKLT_EN
3	BKLT_PWM
4	GND

CN6: LCD DDC CLK/DATA (Molex 53398-0471)



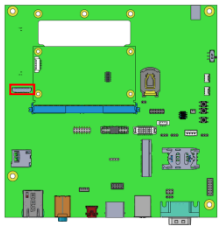
Pin #	Signal Name
1	+5V
2	I2C_SCL
3	I2C_SDA
4	GND

CN7: SPI (E-CALL 0196-01-251-140)



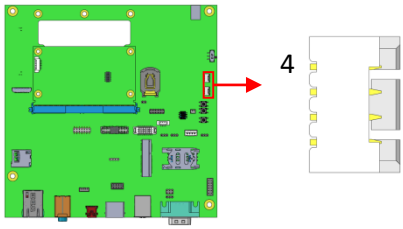
Signal Name	Pin #	Pin #	Signal Name
+3.3V	1	2	+3.3V
NC	3	4	NC
SPI1_CS1#	5	6	SPI2_CS1#
SPI1_MISO	7	8	SPI2_MISO
SPI1_CLK	9	10	SPI2_CLK
SPI1_MOSI	11	12	SPI2_MOSI
GND	13	14	GND

JP11: CSI (PANASONIC_AXT530124)



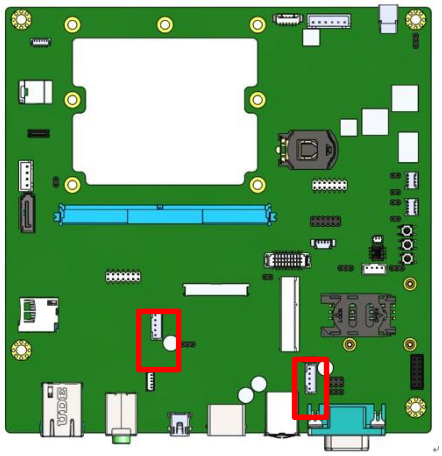
Signal Name	Pin #	Pin #	Signal Name
Reserve	1	2	GND
+2.8V	3	4	GND_CSI
+2.8V	5	6	GND
SCL	7	8	SDA
RESET	9	10	NC
GND	11	12	GND
NC	13	14	GND
GND	15	16	D1+
D1-	17	18	GND
CLK0+	19	20	CLK0-
GND	21	22	D0+
D0-	23	24	GND
MCLK	25	26	GND
NC	27	28	+1.8V
GND	29	30	NC

J2, J4: CAN BUS (E-CALL 0110-2620040)



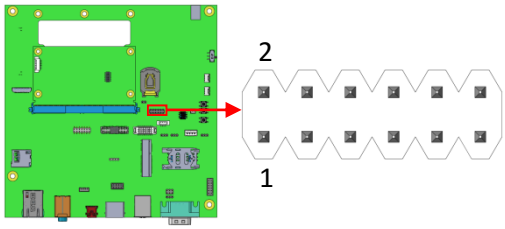
Pin #	Signal Name
1	CANH
2	GND_ISO
3	CANL
4	GND_SHIELD

USB (J16)(J19): USB x 2 (E-CALL_0110-161-050)



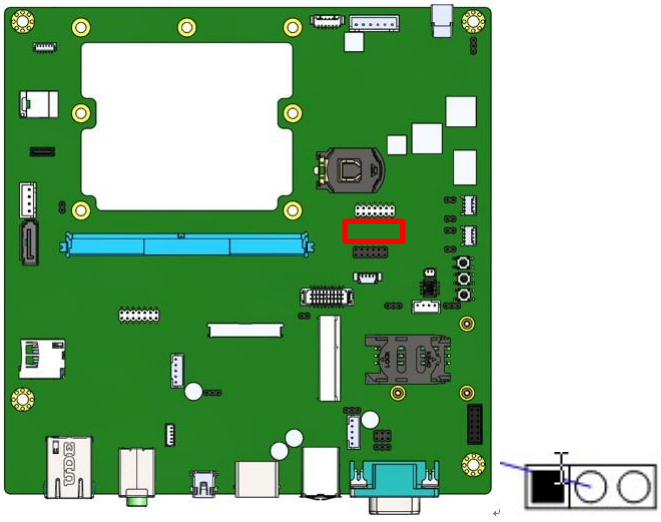
Signal Name	Pin #
+5V	1
USB1_D-	2
USB1_D+	3
GND	4~5

CN5: GPIO (E-CALL 0196-01-200-120)



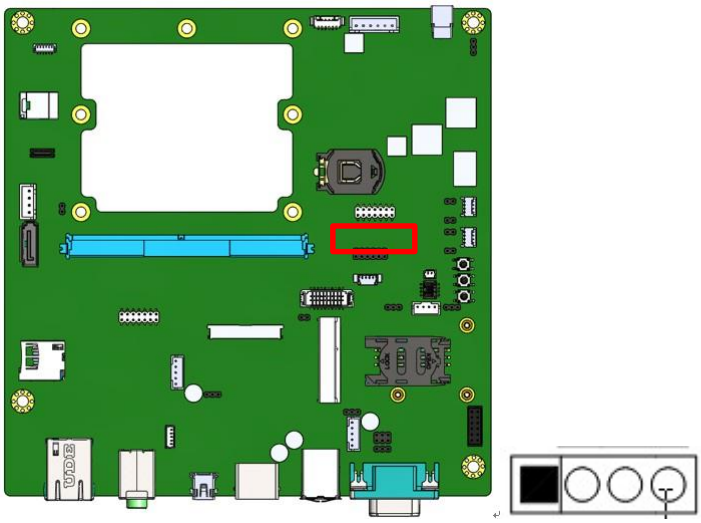
Signal Name	Pin #	Pin #	Signal Name
+3.3V	1	2	GPIO3
GPIO1	3	4	GPIO7
GPIO5	5	6	GPIO9 / MIC Detect
GPIO8	7	8	RESET#
GPIO10 / Headphone Detect	9	10	WDOG#
GPIO11 / SW4	11	12	GND

CN4: Charger Power Management GPIO (E-CALL_0195-01-200-030)(Reserve)



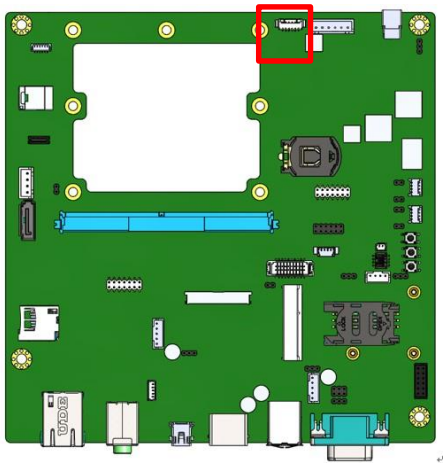
Signal Name	Pin #
CHARGING#	1
CHARGER_PRSNT#	2
BATLOW#	3

CN2: Power Management 2.0mm pin header (E-CALL_0195-01-200-040)(Reserve)



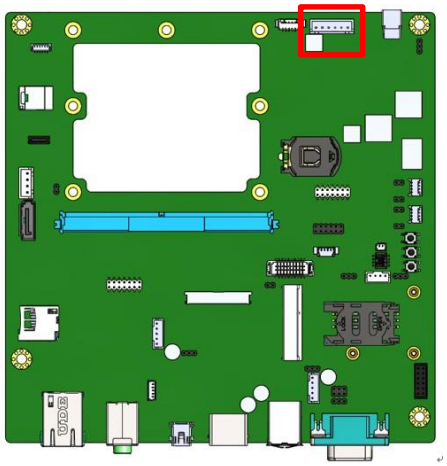
Signal Name	Pin #
Carrier_PWR_ON	1
Carrier_STBY#	2
Carrier_LID#	3
Carrier_SLEEP#	4

J17: MCU Programming box header (TECHBEST_01024062008-L) internal debug only.



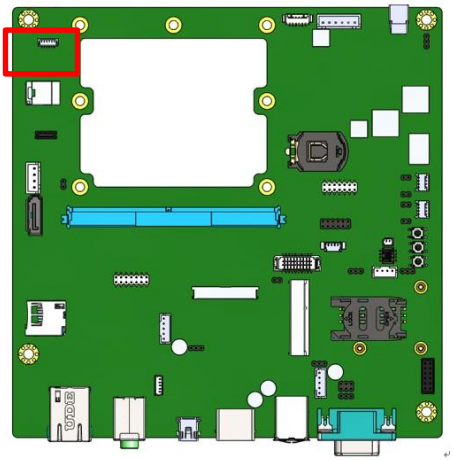
Signal Name	Pin #
+3.3V	1
NC	2
NC	3
TCK	4
TDIO	5
GND	6

JP10: Optional Smart Battery Header (E-CALL_0110-071-060) internal debug only.



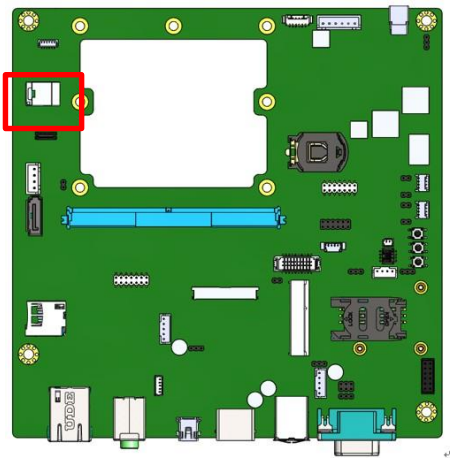
Signal Name	Pin #
Power	1
NC	2
SCL	3
SDA	4
GND	5
NC	6

CN13: I2C box header (TECHBEST_WT02M-30002-06132)



Signal Name	Pin #
Power 1.8V	1
SDA	2
SCL	3
Reserve	4
Reserve	5
GND	6

J18 SPDIF Out (TAKE WING_TWO-584BD-5AT0UK)



3. Software Setup

Users who has Ibase standard image file can refer to this chapter to prepare your own boot-up SD card. Ibase provides HDMI / LVDS output environment by default to let you prepare the software application pre-development easily under Linux / Android platform.

3.1. Make a Recovery SD Card

Preparing your Recovery SD card help to install the Linux/ Android image into eMMC

Please download the **Recovery SD card's image by FTP in advance.**

Host: 219.87.145.180 port: 21

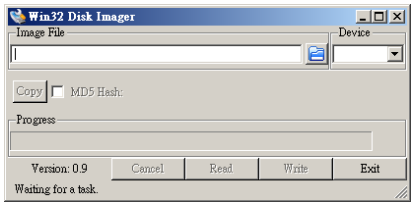
User: bsp

Password: (please check with your sales)

In order to use the evaluation kit, you will need to install an Operating System (OS) onto onboard eMMC by recovery SD card. An Operating System is the set of basic programs and utilities that allow your computer to run.

These instructions will guide you through installing a recovery program on your SD card that will allow you to easily install different OS's and to recover your card when needed.

1. Insert an SD card that is 4GB or greater in size into your computer
2. Format the SD card
    i. Download the SD Association's Formatting Tool (SD Card Formatter 4.0) from **https://www.sdcard.org/downloads/formatter_4/eula_windows/**
    ii. Install and run the Formatting Tool on your machine
    iii. Set "FORMAT SIZE ADJUSTMENT" option to "ON" in the "Options" menu
    iv. Check that the SD card you inserted matches the one selected by the Tool
    v. Click the "Format" button
3. Download the target operating system image from the DVD/ or FTP
    (Descripted in previous page)
4. Download the Win32DiskImager from **http://sourceforge.net/projects/win32diskimager/ and use it to restore the target operating system.**



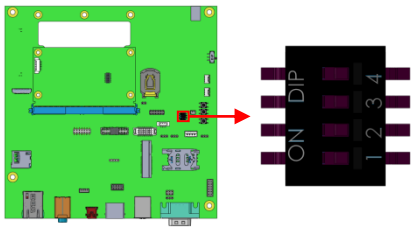
And then, flash the Android/ Linux image into your SD card in your PC (Windows).

6. Please check the DIP switch and make sure it can boot from SD Card.
(See 3.2 Boot on the SMARC starter kit)

3.2. Boot on the SMARC starter kit

Please double check the Boot device selection before power on.

SW3: Boot Select



Boot Source	SW3_PIN1	SW3_PIN2	SW3_PIN3	SW3_PIN4	Linux node name	Android node name
Carrier SD	0	1	1	X		
Module eMMC	1	0	0	X		

Note: 1: Switch On ; 0: Switch Off

1. Insert the SD card/MicroSD into motherboard, make sure the **HDMI panel** is connected, and connect the power supply to boot up the system.
2. Recovery program on your SD card will execute automatically. The eMMC on SMARC will be format, and OS will be installed while the progress bar shows "100%" complete.
3. Remove the power, and the recovery SD. Remember to set the boot source from module eMMC.

Boot Source	SW3_PIN1	SW3_PIN2	SW3_PIN3	SW3_PIN4	Linux node name	Android node name
Carrier SD	0	1	1	X		
Module eMMC	1	0	0	X		

4. **Connect the power** and boot up SMARC, you will see the Linux/ Android boot up pages.

3.3. Parameter Setting on U-boot

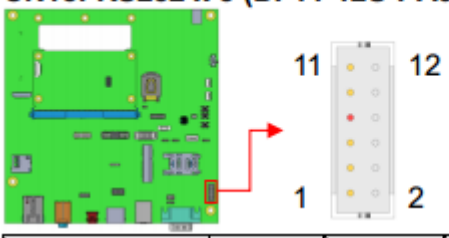
We provide HDMI output and (10.1 " Onation LVDS Panel output command example) for SMARC starter kit. If you have any other LVDS/ TTL panel need to be customized, please contact Ibase sales or FAE staff.

3.3.1. Preparation (debug console)

i. We set the COM1 (Tx1, Rx1) as default debug port, please double check it can be connect to (RX, Tx) of your PC environment.
ii. set 115200 bps (8n1, no flow control) in Windows terminal (for example Putty.exe)
iii. When booting the system, you can ***press "Enter" t***o stop auto boot and modify your environment.

(Note: Users who are not sure the COM connection, please double check your SMARC.COM1.Tx1 connect to PC.COM.Rx ; SMARC.COM1.Rx1 to PC.COM.Tx)

CN15: RS232 x 3 (DF11-12S-PA66H)



Signal Name	Pin #	Pin #	Signal Name
+5V	1	2	TX1
TX5	3	4	RX1
RTS5#	5	6	GND
RX5	7	8	TX2
CTS5#	9	10	RX2
GND	11	12	GND

Note:

The COM1 is map to ttymxc0 device node under Linux& android.

The COM2 is map to ttymxc1 device node under Linux& android.

The COM5 is map to ttymxc4 device node under Linux& android.

3.3.2. Display setting command For Android

Select boot device:

```
MX6SDL SABREDS U-BOOT > setenv bootcmd “booti mmcX”
```

Where mmcX =1, means boot from Carrier’s SD card.

Where mmcX =2, means boot from SMARC module’s eMMC device.

Command to set HDMI+OT101 LVDS panel:

```
setenv bootargs 'console=ttymxc0,115200 androidboot.console=ttymxc0
androidboot.hardware=freescale init=/init ldo_active=on vmalloc=400M
video=mxcfb0:dev=ldb,OT101-XGA,bpp=32 ldb=sep0 video=mxcfb1:dev=hdmi,1280x720M@60
video=mxcfb2:off fbmem=15M,28M'
```

Command to set OT101 10.1” LVDS panel:

```
setenv bootargs 'console=ttymxc0,115200 androidboot.console=ttymxc0
androidboot.hardware=freescale init=/init ldo_active=on vmalloc=400M
video=mxcfb0:dev=ldb,OT101-XGA,bpp=32 ldb=sep0 video=mxcfb1:off video=mxcfb2:off
fbmem=15M'
```

Command to set HDMI output:

```
setenv bootargs 'console=ttymxc0,115200 androidboot.console=ttymxc0
androidboot.hardware=freescale init=/init ldo_active=on vmalloc=400M
video=mxcfb0:dev=hdmi,1280x720M@60 video=mxcfb1:off video=mxcfb2:off fbmem=28M'
```

(please also save the environment and reboot by following command)

```
MX6SDL SABREDS U-BOOT > saveenv
MX6SDL SABREDS U-BOOT > boot
```

3.3.3. Display setting for Linux

Command to set OT101 10.1” Panel:

```
setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/mmcblk0p1 rootwait ldo_active=on
rw video=mxcfb0:dev=ldb,OT101-XGA,if=RGB666 ldb=sep0 rootfstype=ext4'
```

Command to set HDMI output:

```
setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/mmcblk0p1 rootwait ldo_active=on
rw video=mxcfb0:dev=hdmi,1280x720M@60 fbmem=28M rootfstype=ext4'
```

Command to Set the boot device

```
Carrier SD :    root=/dev/mmcblk1p1
```

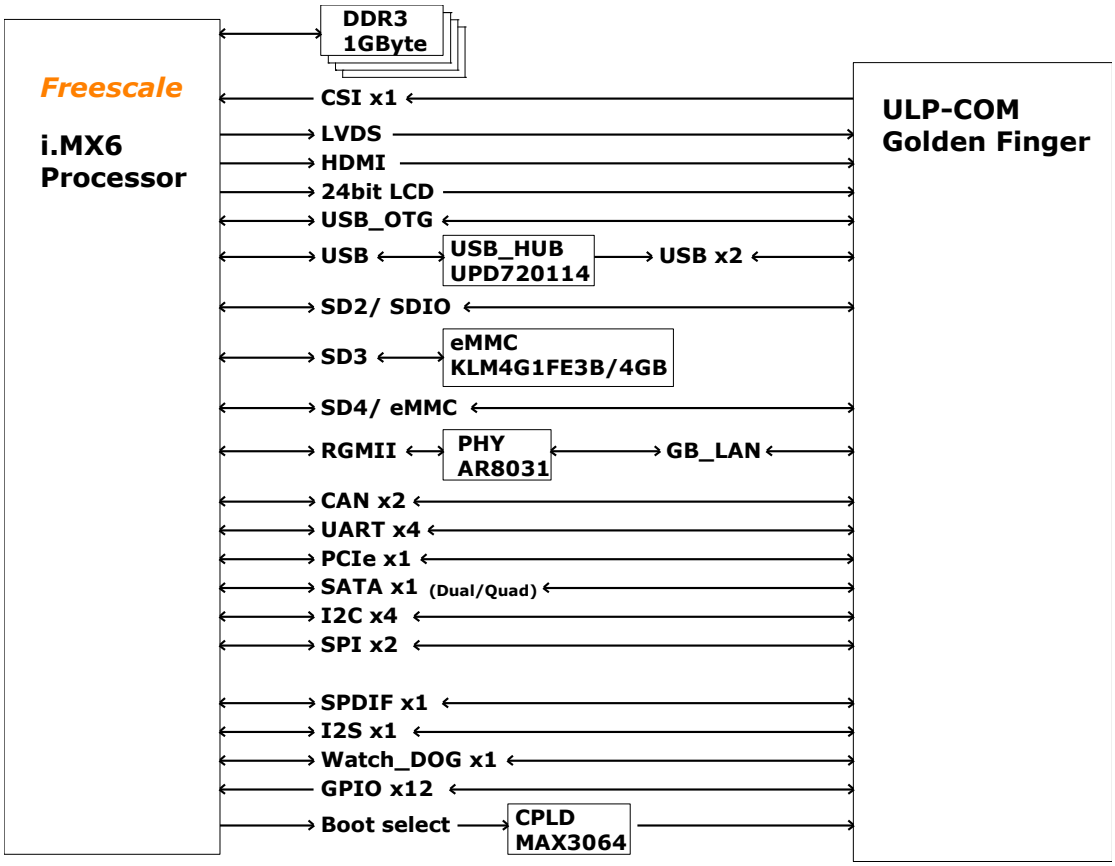
Note: (remember to save the environment and reboot by following command)

```
MX6SDL SABREDS U-BOOT > saveenv
MX6SDL SABREDS U-BOOT > boot
```

4. Carrier Board Design Guide

This Chapter mainly for advanced EE to create your own carrier boards (or products), layout suggestion can be found inside also. for RP-101-SMC carrier board design schematic file. Please contact your sales in advance.

4.1. Block Diagram



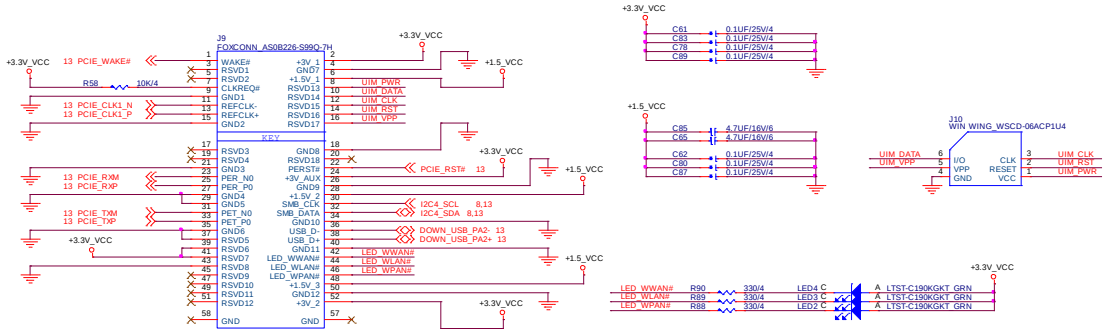
4.2. Interfaces

4.2.1. PCI Express

PCIe Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P83	PCIE_CLK1P	O	PCIe		Differential PCIe reference clock output
P84	PCIE_CLK1N	O	PCIe		
P89	PCIE_TXP	O	PCIe		Differential PCIe transmit data pair
P90	PCIE_TXM	O	PCIe		
P86	PCIE_RXP	I	PCIe		Differential PCIe receive data pair
P87	PCIE_RXM	I	PCIe		
S146	PCIE_WAKE#	I	CMOS	3.3V	PCIe wake up signal
P75	PCIE_RST#	O	CMOS	3.3V	PCIe reset output
S49	I2C4_SDA	I/O	OD	3.3V	I2C interface data, some PICe device need SMB interface for special configuration
S48	I2C4_SCL	O	OD	3.3V	I2C interface clock, some PICe device need SMB interface for special configuration

Reference Schematic



Notice : SMARC PINS48~PINS49 are I2C4, which is activated in RM-F6SO-SMC and RM-F6DU1-SMC module only. Users who need to adopt I2C in RM-F6DU-SMC or RM-F6QD-SMC, please connect it from I2C1~I2C3 in the carrier board.

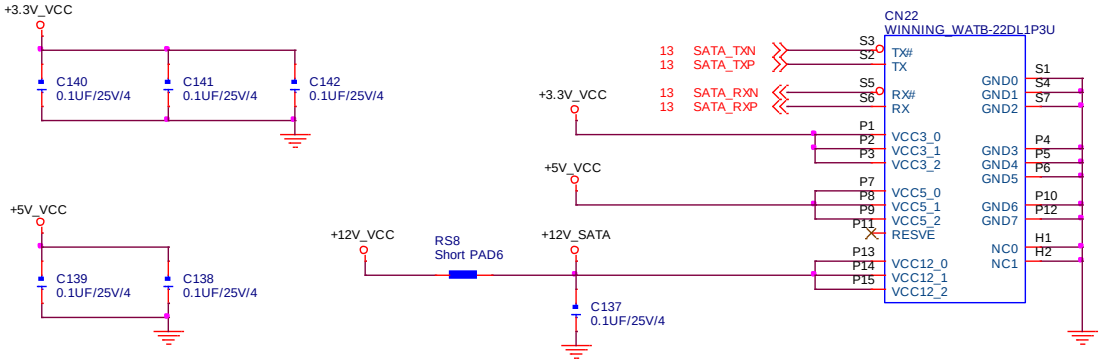
S48	I2C4_SCL (Enable in i.MX6 Solo and i.MX6 Dual-Lite only)
S49	I2C4_SDA(Enable in i.MX6 Solo and i.MX6 Dual-Lite only)

4.2.2. SATAII

SATAII Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P48	SATA_TP	O	SATA		SATA transmit data positive
P49	SATA_TN	O	SATA		SATA transmit data negative
P51	SATA_RP	I	SATA		SATA receive data positive
P52	SATA_RN	I	SATA		SATA receive data negative

Reference Schematic



4.2.3. USB

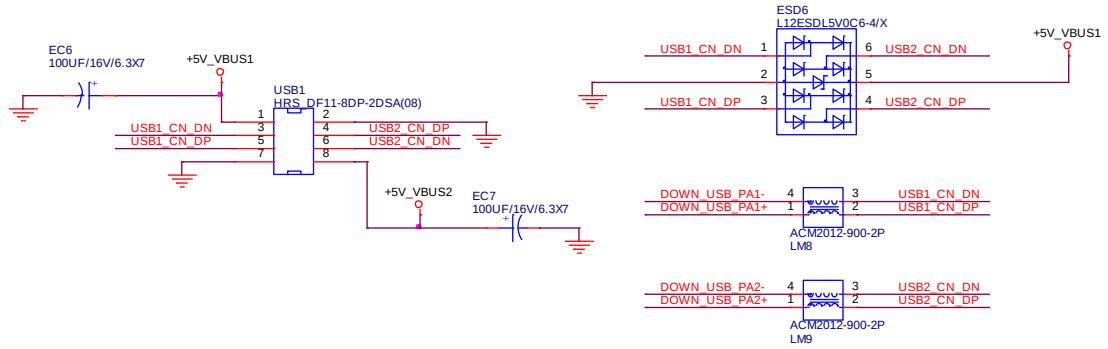
Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P65	DOWN_USB_PA1+	I/O	USB		Positive differential USB host signal
P66	DOWN_USB_PA1-	I/O	USB		Negative differential USB host signal
P67	H2_USB_OC#1	I/O	OD	3.3V	Over current input signal

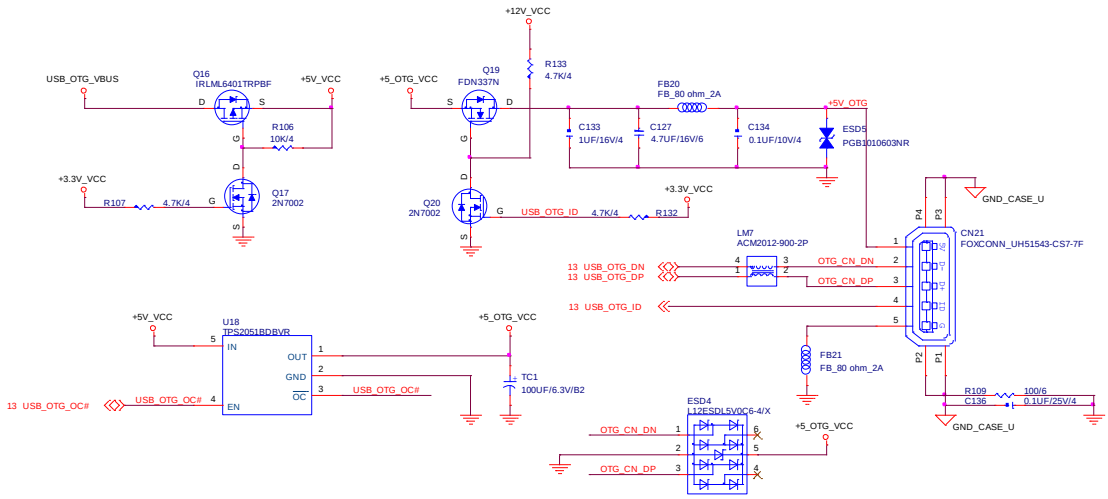
Pin	Signal Name	I/O	Type	Power Rail	Description
P69	DOWN_USB_PA2+	I/O	USB		Positive differential USB host signal
P70	DOWN_USB_PA2-	I/O	USB		Negative differential USB host signal
P71	H2_USB_OC#2	I/O	OD	3.3V	Over current input signal

Pin	Signal Name	I/O	Type	Power Rail	Description
P60	USB_OTG_DP	I/O	USB		Positive differential USB host signal
P61	USB_OTG_DN	I/O	USB		Negative differential USB host signal
P62	USB_OTG_OC#	I/O	OD	3.3V	Over current input signal
P63	USB_OTG_VBUS	O	SATA	5V	SATA transmit data negative
P64	USB_OTG_ID	I	SATA	3.3V	SATA receive data positive

Reference Schematic (USB2.0 schematic)



USB OTG schematic

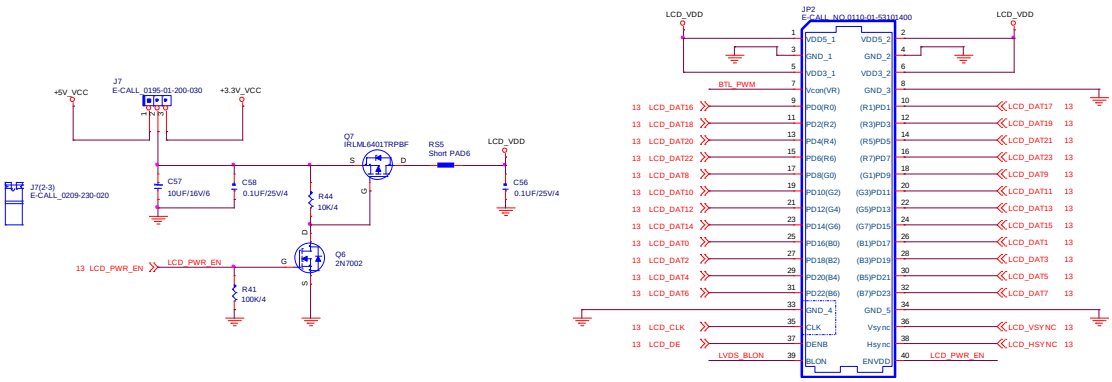


4.2.4. Parallel RGB LCD Interface

Parallel RGB LCD Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
S93	LCD_DAT0	O	CMOS	3.3V	8 bit Blue color data - 18 bit display implementations
S94	LCD_DAT1	O	CMOS	3.3V	leave the two LS bits (D0, D1) not connected
S95	LCD_DAT2	O	CMOS	3.3V	
S96	LCD_DAT3	O	CMOS	3.3V	
S97	LCD_DAT4	O	CMOS	3.3V	
S98	LCD_DAT5	O	CMOS	3.3V	
S99	LCD_DAT6	O	CMOS	3.3V	
S100	LCD_DAT7	O	CMOS	3.3V	
S102	LCD_DAT8	O	CMOS	3.3V	8 bit Green color data - 18 bit display implementations
S103	LCD_DAT9	O	CMOS	3.3V	leave the two LS bits (D8, D9) not connected
S104	LCD_DAT10	O	CMOS	3.3V	
S105	LCD_DAT11	O	CMOS	3.3V	
S106	LCD_DAT12	O	CMOS	3.3V	
S107	LCD_DAT13	O	CMOS	3.3V	
S108	LCD_DAT14	O	CMOS	3.3V	
S109	LCD_DAT15	O	CMOS	3.3V	
S111	LCD_DAT16	O	CMOS	3.3V	8 bit Red color data - 18 bit display implementations
S112	LCD_DAT17	O	CMOS	3.3V	leave the two LS bits (D16, D17) not connected
S113	LCD_DAT18	O	CMOS	3.3V	
S114	LCD_DAT19	O	CMOS	3.3V	
S115	LCD_DAT20	O	CMOS	3.3V	
S116	LCD_DAT21	O	CMOS	3.3V	
S117	LCD_DAT22	O	CMOS	3.3V	
S118	LCD_DAT23	O	CMOS	3.3V	
S120	LCD_DE	O	CMOS	3.3V	Data Enable
S123	LCD_CLK	O	CMOS	3.3V	Pixel Clock
S121	LCD_HSYNC	O	CMOS	3.3V	Horizontal Sync
S122	LCD_VSYNC	O	CMOS	3.3V	Vertical Sync
S127	LCD_BKLT_EN	O	CMOS	3.3V	Backlight Enable
S141	LCD_BKLT_PWM	O	CMOS	3.3V	Display backlight PWM control
S133	LCD_PWR_EN	O	CMOS	3.3V	LCD Power Enable
S140	I2C3_SDA	I/O	OD	3.3V	I2C data, to read LCD display EDID EEPROMs
S139	I2C3_SCL	O	OD	3.3V	I2C Clock, to read LCD display EDID EEPROMs

Reference Schematic

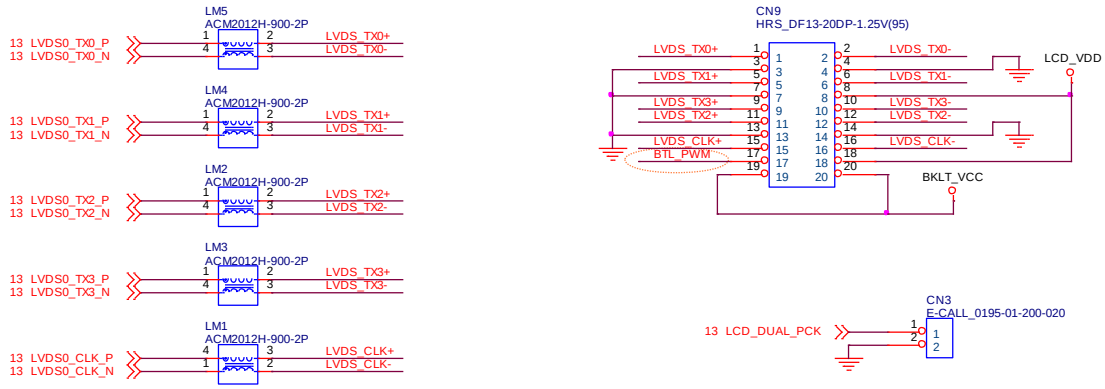


4.2.5. LVDS LCD Interface

LVDS LCD Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
S134	LVDS0_CLK_P	O	LVDS		LVDS clock
S135	LVDS0_CLK_N	O	LVDS		
S125	LVDS0_TX0_P	O	LVDS		LVDS data lane 0
S126	LVDS0_TX0_N	O	LVDS		
S128	LVDS0_TX1_P	O	LVDS		LVDS data lane 1
S129	LVDS0_TX1_N	O	LVDS		
S131	LVDS0_TX2_P	O	LVDS		LVDS data lane 2
S132	LVDS0_TX2_N	O	LVDS		
S137	LVDS0_TX3_P	O	LVDS		LVDS data lane 3
S138	LVDS0_TX3_N	O	LVDS		
S142	LCD_DUAL_PCK	O	LVDS		Pixel clock to support dual channel parallel and LVDS implementations

Reference Schematic

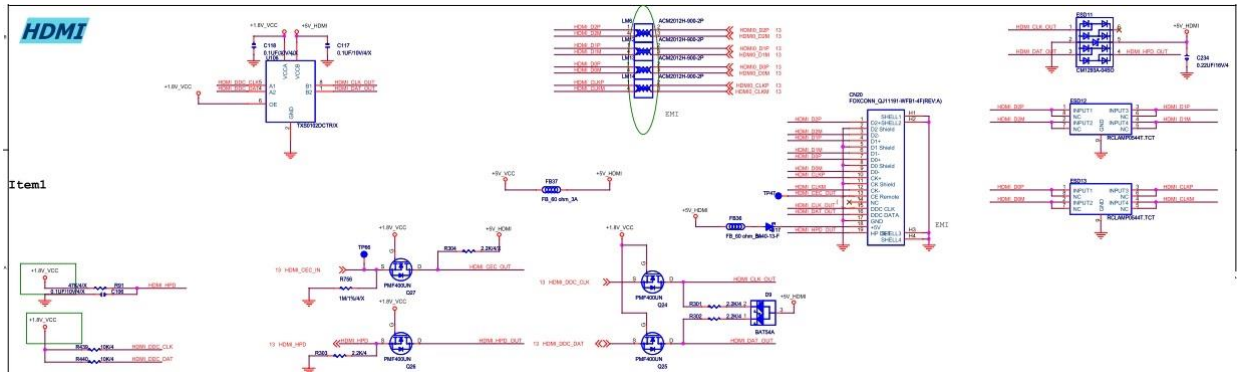


4.2.6. HDMI Interface

HDMI Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P101	HDMI_CLKP	O	TMDS		HDMI clock
P102	HDMI_CLKM	O	TMDS		
P98	HDMI_D0P	O	TMDS		HDMI data lane 0
P99	HDMI_D0M	O	TMDS		
P95	HDMI_D1P	O	TMDS		HDMI data lane 1
P96	HDMI_D1M	O	TMDS		
P92	HDMI_D2P	O	TMDS		HDMI data lane 2
P93	HDMI_D2M	O	TMDS		
P105	HDMI_DDC_CLK	O	OD	3.3V	I2C clock
P106	HDMI_DDC_DATA	I/O	OD	3.3V	I2C data
P104	HDMI_HPD	I	CMOS	3.3V	HDMI Hot Plug Detect input
P107	HDMI_CEC_IN	I/O	OD	3.3V	HDMI Consumer Electronics Control

Reference Schematic

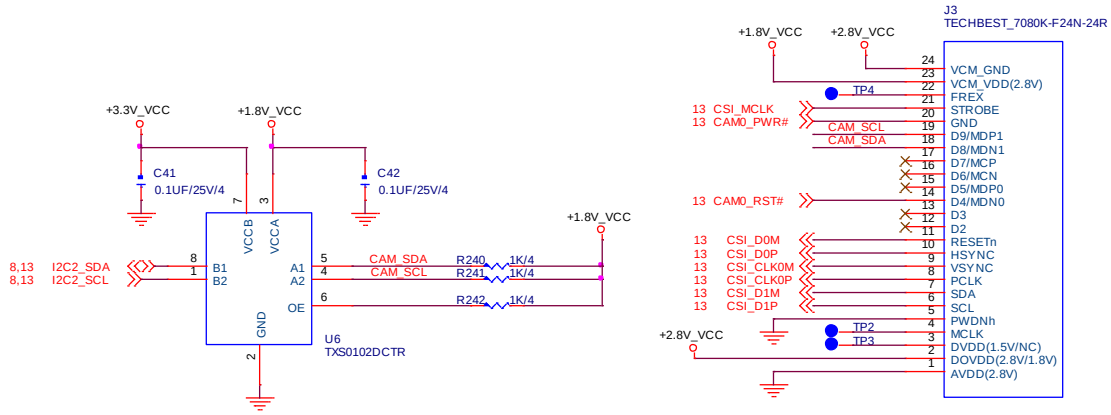


4.2.7. Serial Camera Interface

Serial Camera Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
S8	CSI_CLK0P	I	LVDS D-PHY		CSI clock
S9	CSI_CLK0M	I	LVDS D-PHY		
S11	CSI_D0P	I	LVDS D-PHY		CSI data lane 0
S12	CSI_D0M	I	LVDS D-PHY		
S14	CSI_D1P	I	LVDS D-PHY		CSI data lane 1
S15	CSI_D1M	I	LVDS D-PHY		
S6	CSI_MCLK	O	CMOS	3.3V	Master clock output for CSI camera
S5	I2C2_CLK	O	OD	3.3V	I2C clock
S7	I2C2_SDA	I/O	OD	3.3V	I2C data
P108	CAM0_PWR#	O	CMOS	3.3V	Camera Power Enable
P110	CAM0_RST#	O	CMOS	3.3V	Camera Reset

Reference Schematic

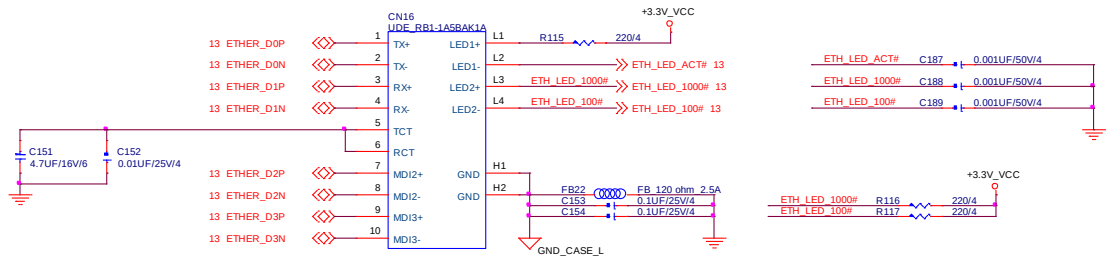


4.2.8. Ethernet

Ethernet Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P30	ETHER_D0P	I/O	MDI		Bi-directional transmit/receive pair 0 to magnetics
P29	ETHER_D0N	I/O	MDI		
P27	ETHER_D1P	I/O	MDI		Bi-directional transmit/receive pair 1 to magnetics
P26	ETHER_D1N	I/O	MDI		
P24	ETHER_D2P	I/O	MDI		Bi-directional transmit/receive pair 2 to magnetics
P23	ETHER_D2N	I/O	MDI		
P20	ETHER_D3P	I/O	MDI		Bi-directional transmit/receive pair 3 to magnetics
P19	ETHER_D3N	I/O	MDI		
P25	ETH_LED_ACT#	O	OD	3.3V	Link / Activity Indication LED
P21	ETH_LED_100#	O	OD	3.3V	Link Speed Indication LED for 100Mbps
P22	ETH_LED_1000#	O	OD	3.3V	Link Speed Indication LED for 1000 Mbps

Reference Schematic

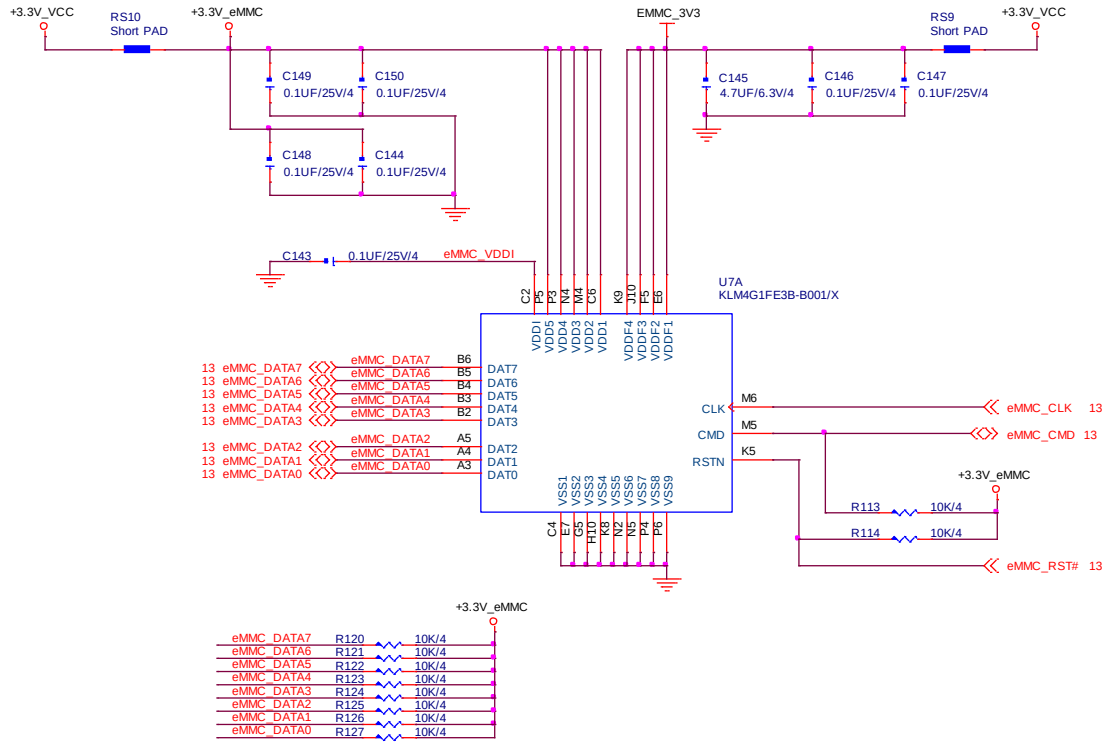


4.2.9. eMMC

eMMC signals

Pin	Signal Name	I/O	Type	Power Rail	Description
S26	eMMC_DATA0	I/O	CMOS	3.3V	8 bit data path (may be used for 4 and 1 bit wide eMMC devices as well)
S27	eMMC_DATA1	I/O	CMOS	3.3V	
S28	eMMC_DATA2	I/O	CMOS	3.3V	
S29	eMMC_DATA3	I/O	CMOS	3.3V	
S30	eMMC_DATA4	I/O	CMOS	3.3V	
S31	eMMC_DATA5	I/O	CMOS	3.3V	
S32	eMMC_DATA6	I/O	CMOS	3.3V	
S33	eMMC_DATA7	I/O	CMOS	3.3V	
S35	eMMC_CLK	O	CMOS	3.3V	Clock
S36	eMMC_CMD	I/O	CMOS	3.3V	Command line
S37	eMMC_RST#	O	CMOS	3.3V	Reset signal to eMMC device

Reference Schematic

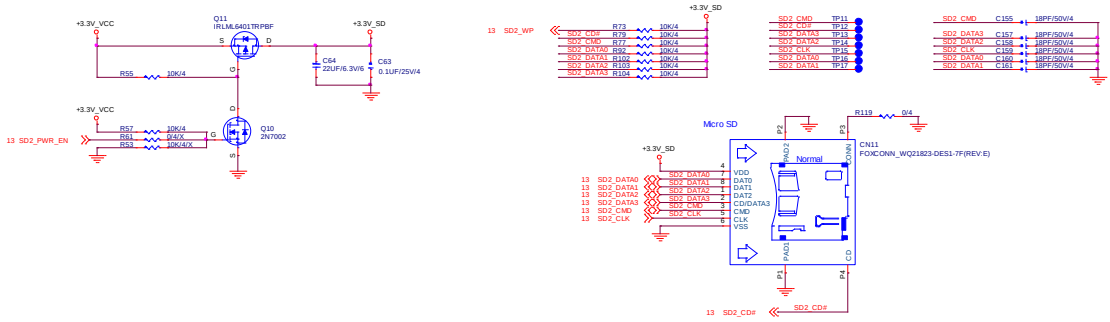


4.2.10. SD Card Interface

SD Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P39	SD2_DATA0	I/O	CMOS	3.3V	4 bit data path
P40	SD2_DATA1	I/O	CMOS	3.3V	
P41	SD2_DATA2	I/O	CMOS	3.3V	
P42	SD2_DATA3	I/O	CMOS	3.3V	
P33	SD2_WP	I	CMOS	3.3V	Write Protect
P35	SD2_CD#	I	CMOS	3.3V	Card Detect
P37	SD2_PWR_EN	O	CMOS	3.3V	SD card power enable
P36	SD2_CLK	O	CMOS	3.3V	Clock
P34	SD2_CMD	I/O	CMOS	3.3V	Command line

Reference Schematic

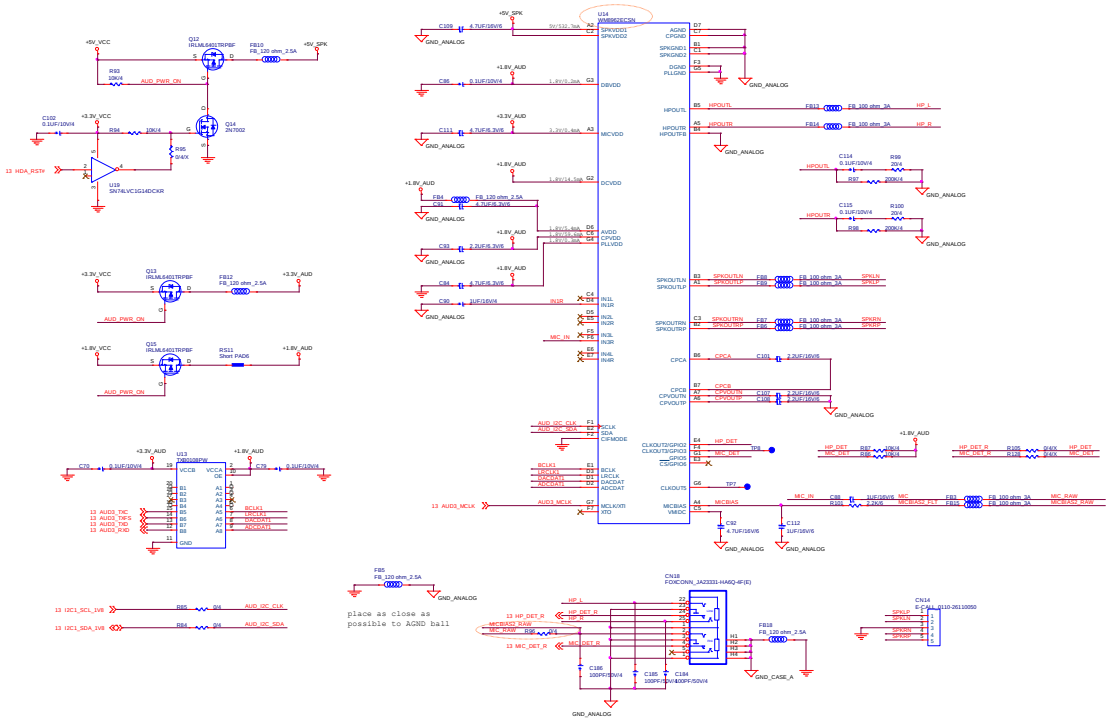


4.2.11. I2S Interface

I2S signals

Pin	Signal Name	I/O	Type	Power Rail	Description
S39	AUD3_TXFS	I/O	CMOS	3.3V	Left& Right audio synchronization clock
S40	AUD3_TXD	O	CMOS	3.3V	Digital audio Output
S41	AUD3_RXD	I	CMOS	3.3V	Digital audio Input
S42	AUD3_TXC	I/O	CMOS	3.3V	Digital audio clock
S38	AUD_MCLK	O	CMOS	3.3V	Master clock output to Audio codecs

Reference Schematic

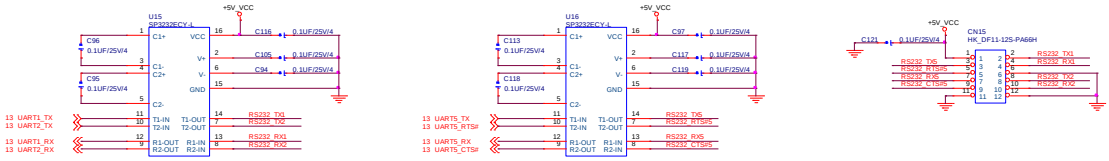


4.2.12. Asynchronous Serial Ports

Serial Port Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P134	UART1_TX	O	CMOS	3.3V	Serial port 1 data out
P135	UART1_RX	I	CMOS	3.3V	Serial port 1 data input
P140	UART2_TX	O	CMOS	3.3V	Serial port 2 data out
P141	UART2_RX	I	CMOS	3.3V	Serial port 2 data input
P129	UART4_TX	O	CMOS	3.3V	Serial port 4 data out
P130	UART4_RX	I	CMOS	3.3V	Serial port 4 data input
P131	U4_RTS#	O	CMOS	3.3V	Request to Send handshake line for serial port 4
P132	U4_CTS#	I	CMOS	3.3V	Clear to Send handshake line for serial port 4
P136	UART5_TX	O	CMOS	3.3V	Serial port 5 data out
P137	UART5_RX	I	CMOS	3.3V	Serial port 5 data input
P138	U5_RTS#	O	CMOS	3.3V	Request to Send handshake line for serial port 5
P139	U5_CTS#	I	CMOS	3.3V	Clear to Send handshake line for serial port 5

Reference Schematic

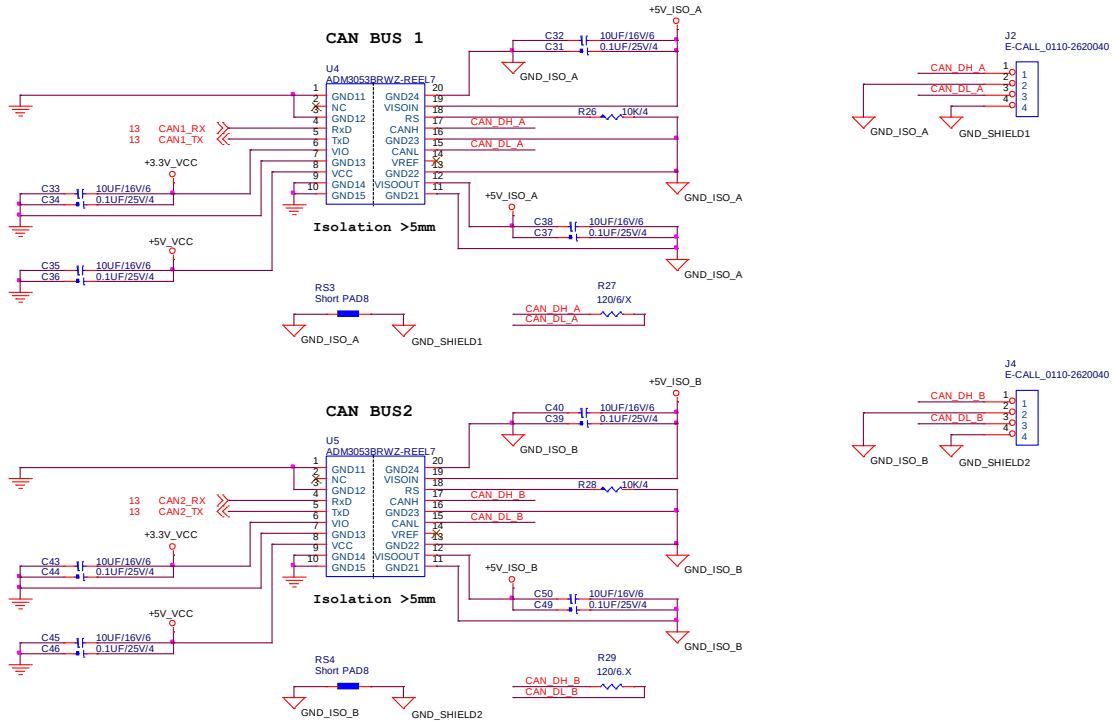


4.2.13. CAN Bus

CAN Bus Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P143	CAN1_TX	O	CMOS	3.3V	CAN1 Transmit output
P144	CAN1_RX	I	CMOS	3.3V	CAN1 Receive input
P145	CAN2_TX	O	CMOS	3.3V	CAN2 Transmit output
P146	CAN2_RX	I	CMOS	3.3V	CAN2 Receive input

Reference Schematic



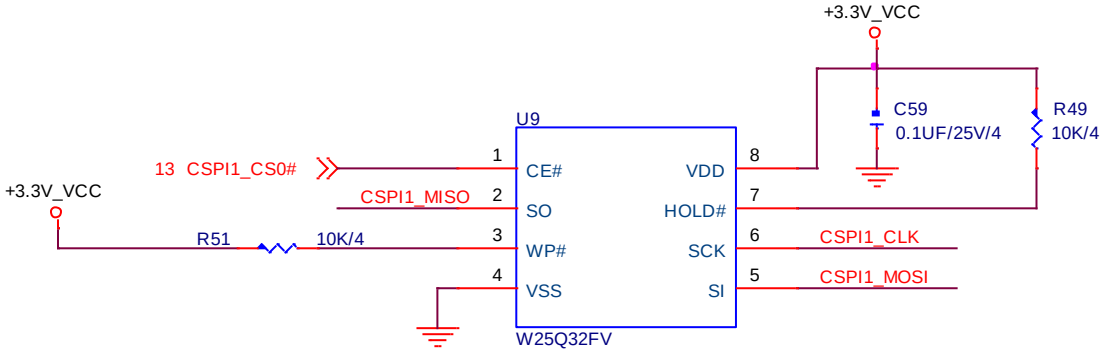
4.2.14. SPI Interface

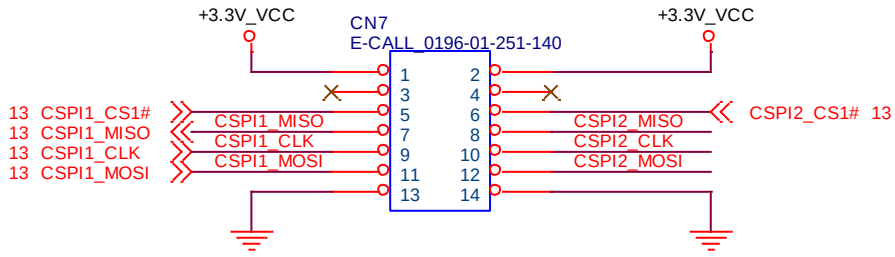
SPI Signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P43	CSPI1_CS0#	O	CMOS	3.3V	SPI1 Master Chip Select 0 output
P31	CSPI1_CS1#	O	CMOS	3.3V	SPI1 Master Chip Select 1 output
P44	CSPI1_CLK	O	CMOS	3.3V	SPI1 Master Clock output
P45	CSPI1_MISO	I	CMOS	3.3V	SPI1 Master Data input
P46	CSPI1_MOSI	O	CMOS	3.3V	SPI1 Master Data output

Pin	Signal Name	I/O	Type	Power Rail	Description
P54	CSPI2_CS0#	O	CMOS	3.3V	SPI2 Master Chip Select 0 output
P55	CSPI2_CS1#	O	CMOS	3.3V	SPI2 Master Chip Select 1 output
P56	CSPI2_CLK	O	CMOS	3.3V	SPI2 Master Clock output
P57	CSPI2_MISO	I	CMOS	3.3V	SPI2 Master Data input
P58	CSPI2_MOSI	O	CMOS	3.3V	SPI2 Master Data output

Reference Schematic





4.2.15. I2C Interface

I2C signals

Pin	Signal Name	I/O	Type	Power Rail	Description
P121	I2C1_SCL_1V8	O	OD	1.8V	I2C Clock signal for Audio CODEC
P122	I2C1_SDA_1V8	I/O	OD	1.8V	I2C Data signal for Audio CODEC
S5	I2C2_SCL	O	OD	3.3V	I2C Clock signal for Camera
S7	I2C2_SCL	I/O	OD	3.3V	I2C Data signal for Camera
S139	I2C3_SCL	O	OD	3.3V	I2C Clock signal for LCD Display
S140	I2C3_SCL	I/O	OD	3.3V	I2C Data signal for LCD Display
S48	I2C4_SCL	O	OD	3.3V	I2C Clock signal for PCI express (SO1 only)
S49	I2C4_SCL	I/O	OD	3.3V	I2C Data signal for PCI express (SO1 only)

4.3. Layout recommendations

- Signal impedance recommendation

Signal Group	Impedance	Layout Tolerance (±)
All signals, unless specified	50 Ω Single End	10%
PCIe, USB Diff signals	90 Ω Differential	10%
Diff signals: LVDS, SATA, HDMI, DDR, MIPI (CSI & DSI), MLB, PHY IC to Ethernet Connector	100 Ω Differential	10%

- PCB stack up and trace width/space recommendation (base on RP-101-SMC reference board)

L1. Top signals

L2. GND

L3. Int1 signals

L4. VCC

L5. GND

L6. Bottom signals

Micro strip			
Layer	Trace width/space	Single-End	Differential
1,6	4	50 +/- 10 %	
1,6	4/6/4		90 +/- 10 %
1,6	4/12/4		100 +/- 10 %

Strip line			
Layer	Trace width/space	Single-End	Differential
3	4	50 +/- 10 %	
3	4/6/4		90 +/- 10 %
3	4/12/4		100 +/- 10 %

Layer	Glass Style & Cu Wt.	Thickness
L1	0.5OZ plating to 1OZ	1.4
P.P	1080	2.6
L2	1 OZ	1.4
CORE	0.08 FR-4	3
L3	1 OZ	1.4
P.P	7628	7.1
CORE	0.71 FR-4	28
P.P	7628	7.1
L4	1 OZ	1.4
CORE	0.08 FR-4	3
L5	1 OZ	1.4
P.P	1080	2.6
L6	0.5OZ plating to 1OZ	1.4

Total thickness 61.8mil 1.57mm

4.4. SMRC Module (RM-F6xx-SMC) Pin Out Table.

In this section, you will find Ibase SMARC module's 314 pins definition by this table.

P-Pin	Primary (Top) Side	S-Pin	Secondary (Bottom) Side
		S1	
P1		S2	
P2	GND	S3	GND
P3		S4	
P4		S5	I2C2_SCL
P5		S6	CSI_MCLK
P6		S7	I2C2_SDA
P7		S8	CSI_CLK0P
P8		S9	CSI_CLK0M
P9	GND	S10	GND
P10		S11	CSI_D0P
P11		S12	CSI_D0M
P12	GND	S13	GND
P13		S14	CSI_D1P
P14		S15	CSI_D1M
P15	GND	S16	GND
P16		S17	
P17		S18	
P18	GND	S19	
P19	ETHER_D3N	S20	
P20	ETHER_D3P	S21	
P21	ETH_LED_100#	S22	
P22	ETH_LED_1000#	S23	
P23	ETHER_D2N	S24	
P24	ETHER_D2P	S25	GND
P25	ETH_LED_ACT#	S26	eMMC_DATA0
P26	ETHER_D1N	S27	eMMC_DATA1
P27	ETHER_D1P	S28	eMMC_DATA2
P28		S29	eMMC_DATA3
P29	ETHER_D0N	S30	eMMC_DATA4
P30	ETHER_D0P	S31	eMMC_DATA5
P31	CSPI1_SS2	S32	eMMC_DATA6
P32	GND	S33	eMMC_DATA7
P33	SD2_WP	S34	GND
P-Pin	**Primary (Top) Side**	**S-Pin**	**Secondary (Bottom) Side**
P34	SD2_CMD	S35	eMMC_CLK
P35	SD2_CD#	S36	eMMC_CMD
P36	SD2_CLK	S37	eMMC_RST#
P37	SD2_PWR_EN	S38	AUD_MCLK
P38	GND	S39	AUD3_TXFS
P39	SD2_DATA0	S40	AUD3_TXD
P40	SD2_DATA1	S41	AUD3_RXD
P41	SD2_DATA2	S42	AUD3_TXC
P42	SD2_DATA3	S43	I2S1_LRCK
P43	CSPI1_SS1	S44	
P44	CSPI1_CLK	S45	
P45	CSPI1_MISO	S46	
P46	CSPI1_MOSI	S47	GND
P47	GND	S48	I2C4_SCL (SO1 Only)
P48	SATA_TP	S49	I2C4_SDA(SO1 Only)
P49	SATA_TN	S50	
P50	GND	S51	

P51	SATA_RP	S52	
P52	SATA_RN	S53	
P53	GND	S54	
P54	CSPI2_SS0	S55	
P55	CSPI2_SS1	S56	
P56	CSPI2_CLK	S57	
P57	CSPI2_MISO	S58	
P58	CSPI2_MOSI	S59	SPDIF_OUT
P59	GND	S60	SPDIF_IN
P60	USB_OTG_DP	S61	GND
P61	USB_OTG_DN	S62	
P62	USB_OTG_OC#	S63	
P63	USB_OTG_VBUS	S64	GND
P64	USB_OTG_ID	S65	
P65	DOWN_USB_PA1+	S66	
P66	DOWN_USB_PA1-	S67	GND
P67	H2_USB_OC#1	S68	
P68	GND	S69	
P69	DOWN_USB_PA2+	S70	GND
P-Pin	**Primary (Top) Side**	**S-Pin**	**Secondary (Bottom) Side**
P70	DOWN_USB_PA2-	S71	
P71	H2_USB_OC#2	S72	
P72		S73	GND
P73		S74	
P74		S75	
P75	PCIE_RST#	S76	
P76		S77	
P77		S78	
P78		S79	
P79	GND	S80	GND
P80		S81	
P81		S82	
P82	GND	S83	GND
P83	PCIe_CKP	S84	
P84	PCIe_CKM	S85	
P85	GND	S86	GND
P86	PCIe_RP	S87	
P87	PCIe_RM	S88	
P88	GND	S89	GND
P89	PCIe_TP	S90	
P90	PCIe_TM	S91	
P91	GND	S92	GND
P92	HDMI_D2P	S93	LCD_DAT0
P93	HDMI_D2M	S94	LCD_DAT1
P94	GND	S95	LCD_DAT2
P95	HDMI_D1P	S96	LCD_DAT3
P96	HDMI_D1M	S97	LCD_DAT4
P97	GND	S98	LCD_DAT5
P98	HDMI_D0P	S99	LCD_DAT6
P99	HDMI_D0M	S100	LCD_DAT7
P100	GND	S101	GND
P101	HDMI_CLKP	S102	LCD_DAT8
P102	HDMI_CLKM	S103	LCD_DAT9
P103	GND	S104	LCD_DAT10
P104	HDMI_HPD	S105	LCD_DAT11
P105	HDMI_DDC_CLK	S106	LCD_DAT12
P-Pin	**Primary (Top) Side**	**S-Pin**	**Secondary (Bottom) Side**

P106	HDMI_DDC_DAT	S107	LCD_DAT13
P107	HDMI_CEC_IN	S108	LCD_DAT14
P108	GPIO_0	S109	LCD_DAT15
P109	GPIO_1	S110	GND
P110	GPIO_2	S111	LCD_DAT16
P111	GPIO_3	S112	LCD_DAT17
P112	GPIO_4	S113	LCD_DAT18
P113	GPIO_5	S114	LCD_DAT19
P114	GPIO_6	S115	LCD_DAT20
P115	GPIO_7	S116	LCD_DAT21
P116	GPIO_8	S117	LCD_DAT22
P117	GPIO_9	S118	LCD_DAT23
P118	GPIO_10	S119	GND
P119	GPIO_11	S120	LCD_DE
P120	GND	S121	LCD_VSYNC
P121	I2C1_SCL_1V8	S122	LCD_HSYNC
P122	I2C1_SDA_1V8	S123	LCD_CLK
P123	BOOT_SEL0#	S124	GND
P124	BOOT_SEL1#	S125	LVDS0_TX0_P
P125	BOOT_SEL2#	S126	LVDS0_TX0_N
P126	RESET_OUT#	S127	LCD_BKLT_EN
P127	RESET_IN#	S128	LVDS0_TX1_P
P128	POWER_BTN#	S129	LVDS0_TX1_N
P129	UART4_TX	S130	GND
P130	UART4_RX	S131	LVDS0_TX2_P
P131	UART4_RTS#	S132	LVDS0_TX2_N
P132	UART4_CTS#	S133	LCD_PWR_EN
P133	GND	S134	LVDS0_CLK_P
P134	UART1_TX	S135	LVDS0_CLK_N
P135	UART1_RX	S136	GND
P136	UART5_TX	S137	LVDS0_TX3_P
P137	UART5_RX	S138	LVDS0_TX3_N
P138	UART5_RTS#	S139	I2C3_SCL
P139	UART5_CTS#	S140	I2C3_SDA
P140	UART2_TX	S141	LCD_BKLT_PWM
P141	UART2_RX	S142	LCD_DUAL_PCK
P-Pin	**Primary (Top) Side**	**S-Pin**	**Secondary (Bottom) Side**
P142	GND	S143	GND
P143	CAN1_TX	S144	
P144	CAN1_RX	S145	WDOG#
P145	CAN2_TX	S146	PCIE_WAKE#
P146	CAN2_RX	S147	P3V3_LICELL
P147	VDD_IN	S148	Carrier_LID#
P148	VDD_IN	S149	Carrier_SLEEP#
P149	VDD_IN	S150	
P150	VDD_IN	S151	CHARGING#
P151	VDD_IN	S152	CHARGER_PRSNT#
P152	VDD_IN	S153	Carrier_STBY#
P153	VDD_IN	S154	Carrier_PWR_ON
P154	VDD_IN	S155	
P155	VDD_IN	S156	BATLOW#
P156	VDD_IN	S157	
		S158	VDD_IO_SEL#

5. BSP User Guide (for advanced software engineer only)

This Chapter mainly for advanced SW engineer to build the image for SMARC starter kit. Any other modification, new device or driver should be handled carefully.

5.1. Building SMARC BSP Source

5.1.1. Preparation

PC: Suggested Host Platform: Ubuntu 10.04 x64 version

Install necessary packages before build:

apt-get install build-essential uboot-mkimage ia32-libs

Note: ** To simplify build process, please run build/installation with root on your x86 host PC. **

5.1.2. Installing Toolchain

Download and extract freescale toolchain (gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tgz)

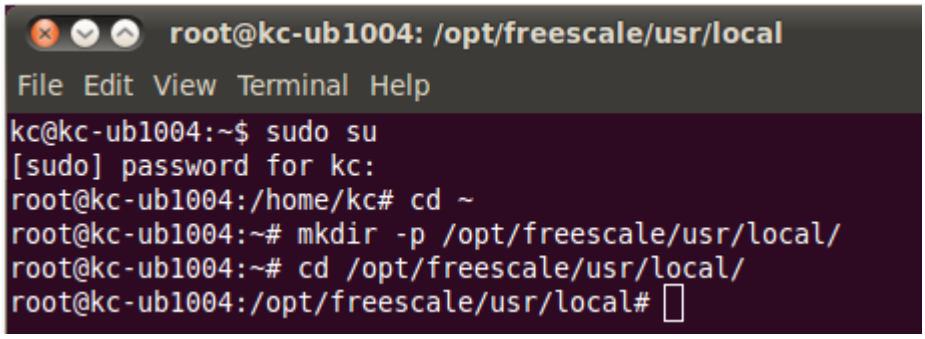
assume your toolchain file is located at root home dir:

sudo su

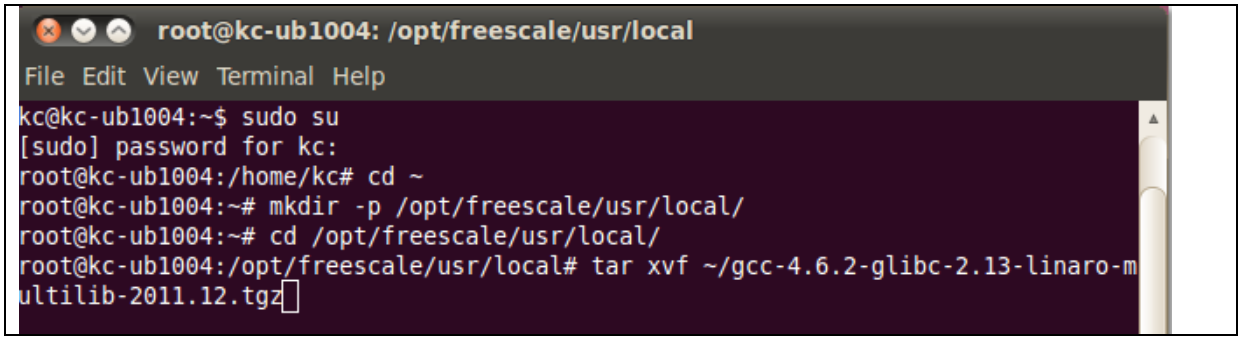
cd ~

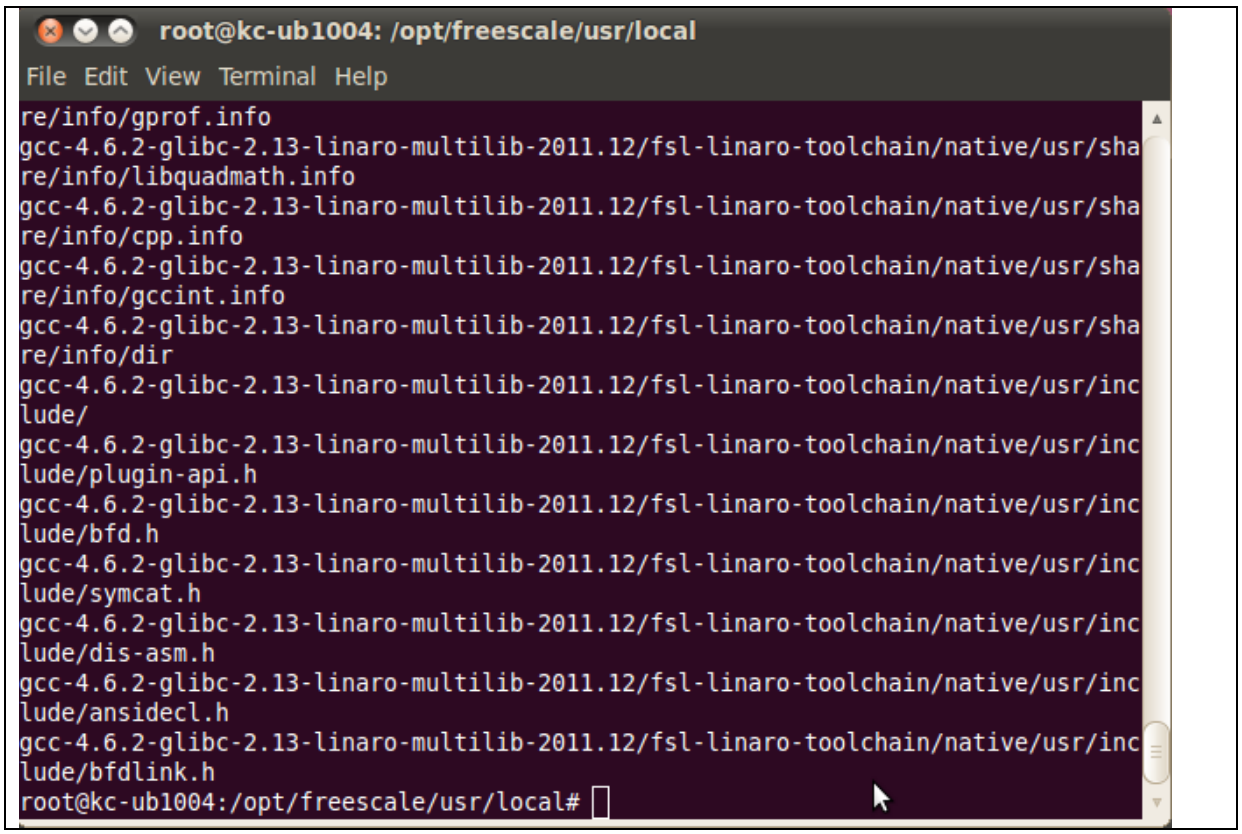
mkdir -p /opt/freescale/usr/local/

cd /opt/freescale/usr/local/



tar xvf ~/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tgz

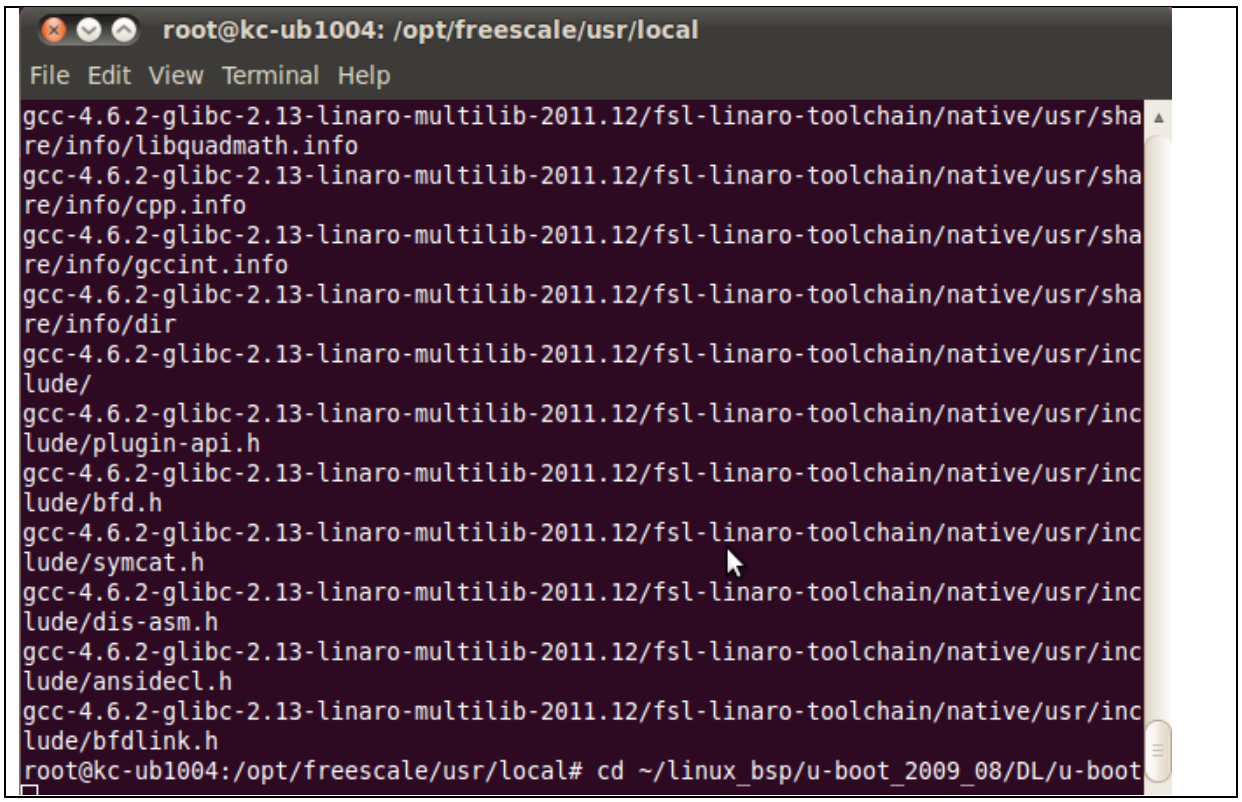




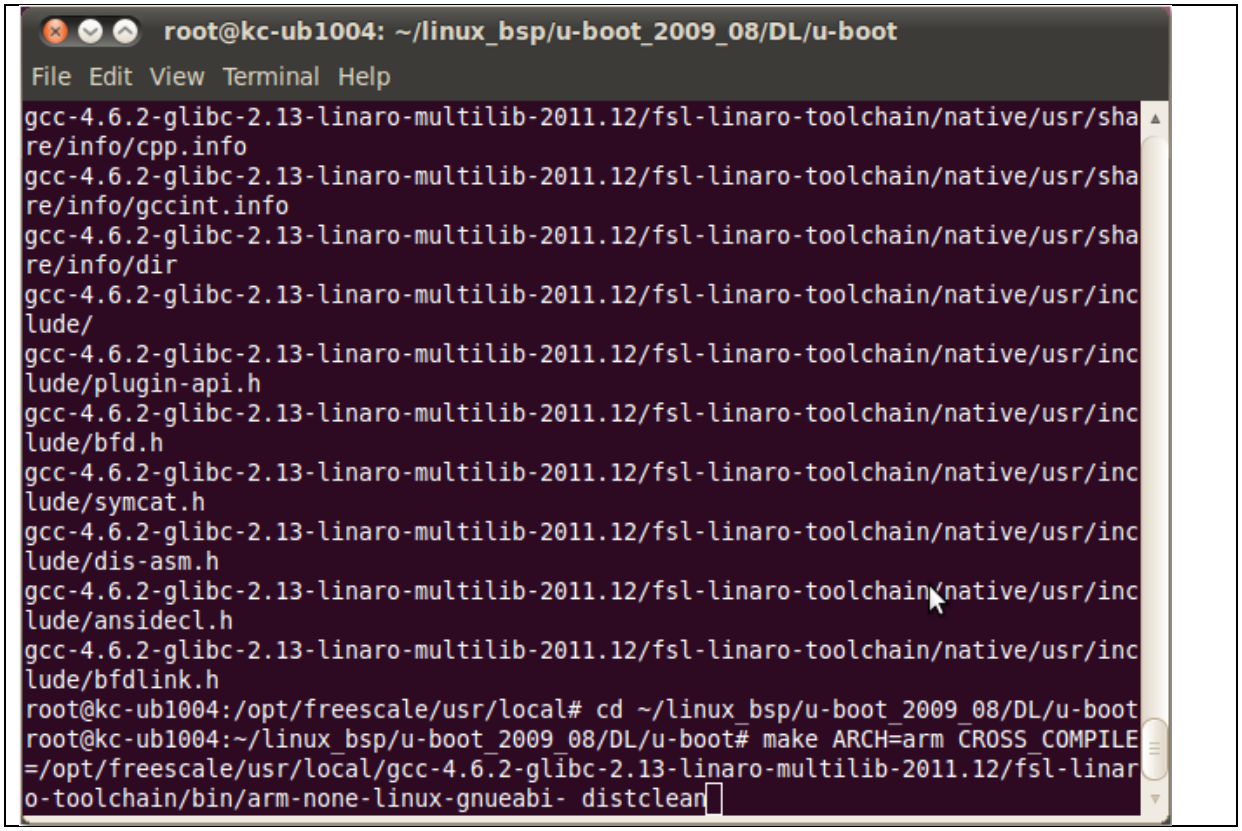
5.1.3. Building u-boot

Assume your linux BSP u-boot source is at ~/linux_bsp/u-boot_2009_08/DL/u-boot

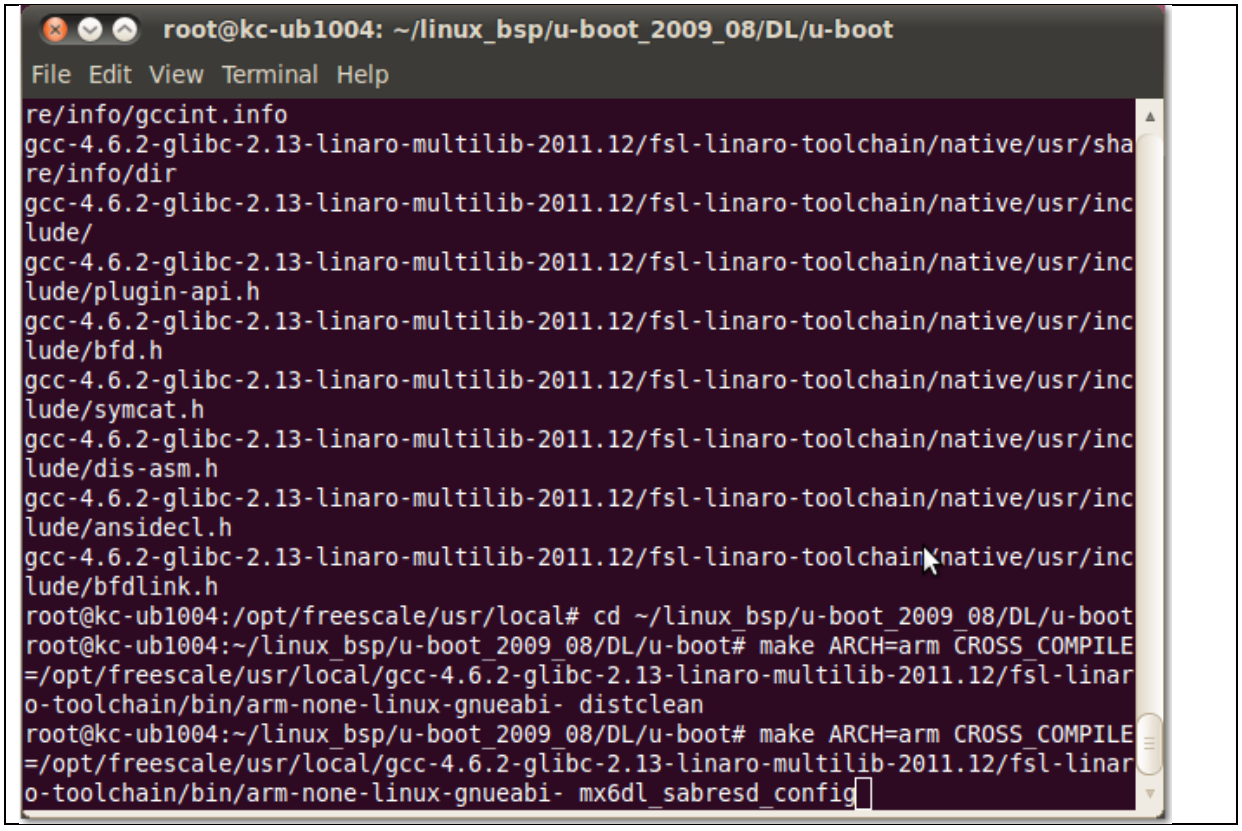
cd ~/linux_bsp/u-boot_2009_08/DL/u-boot



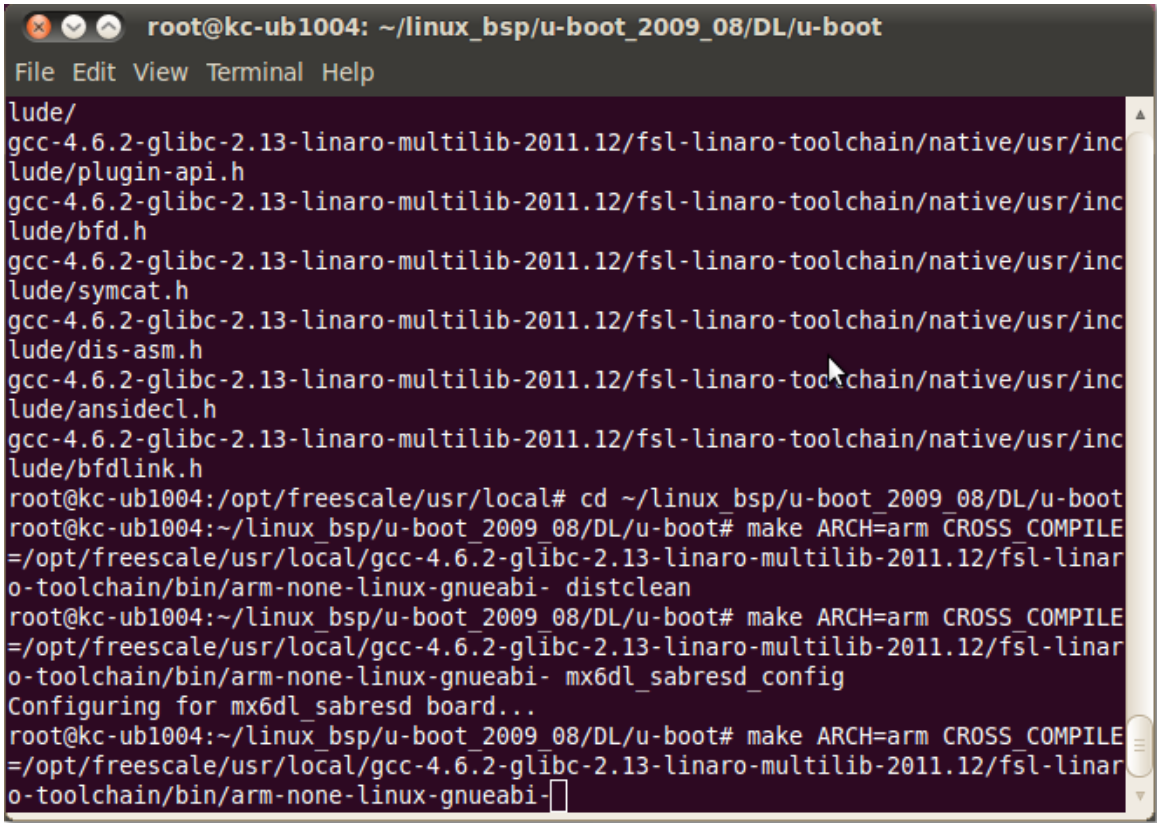
make ARCH=arm CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/arm-none-linux-gnueabi- distclean



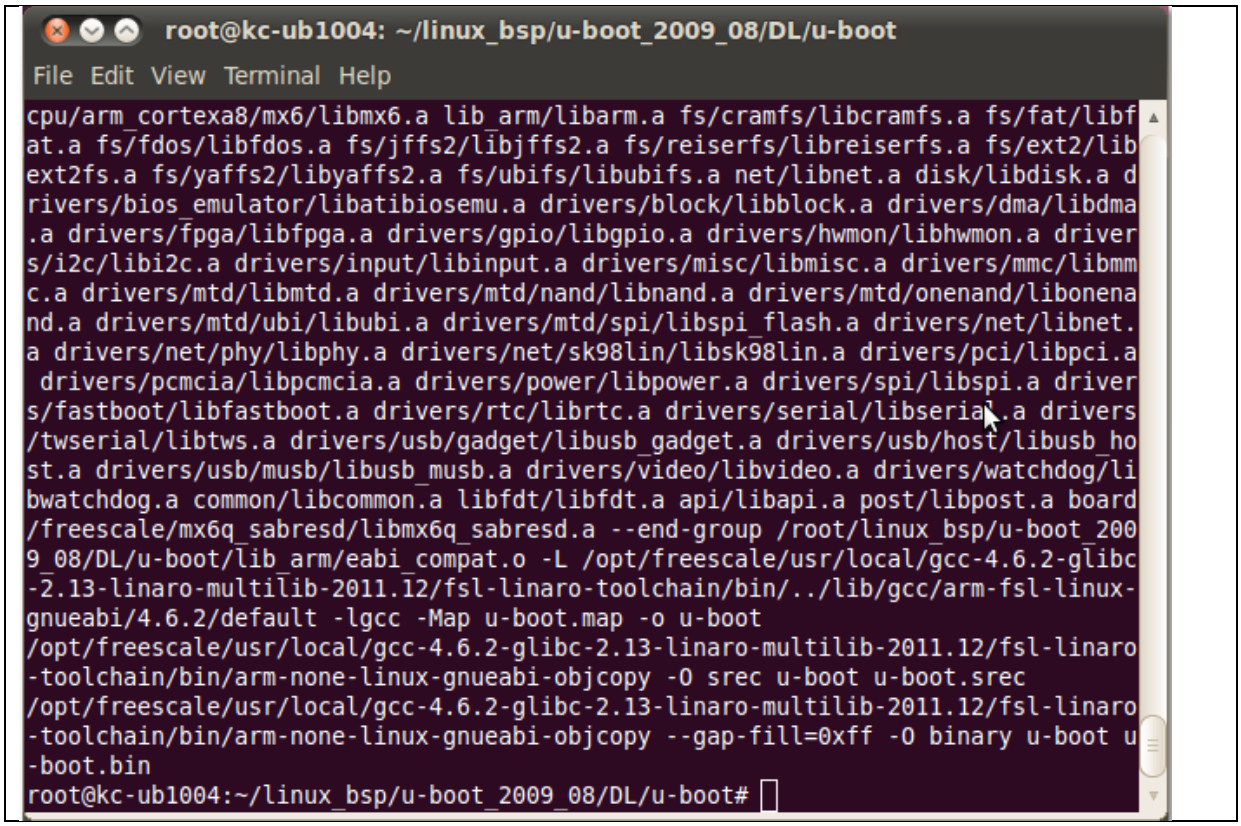
make ARCH=arm CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/arm-none-linux-gnueabi- mx6dl_sabresd_config



make ARCH=arm CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/arm-none-linux-gnueabi-



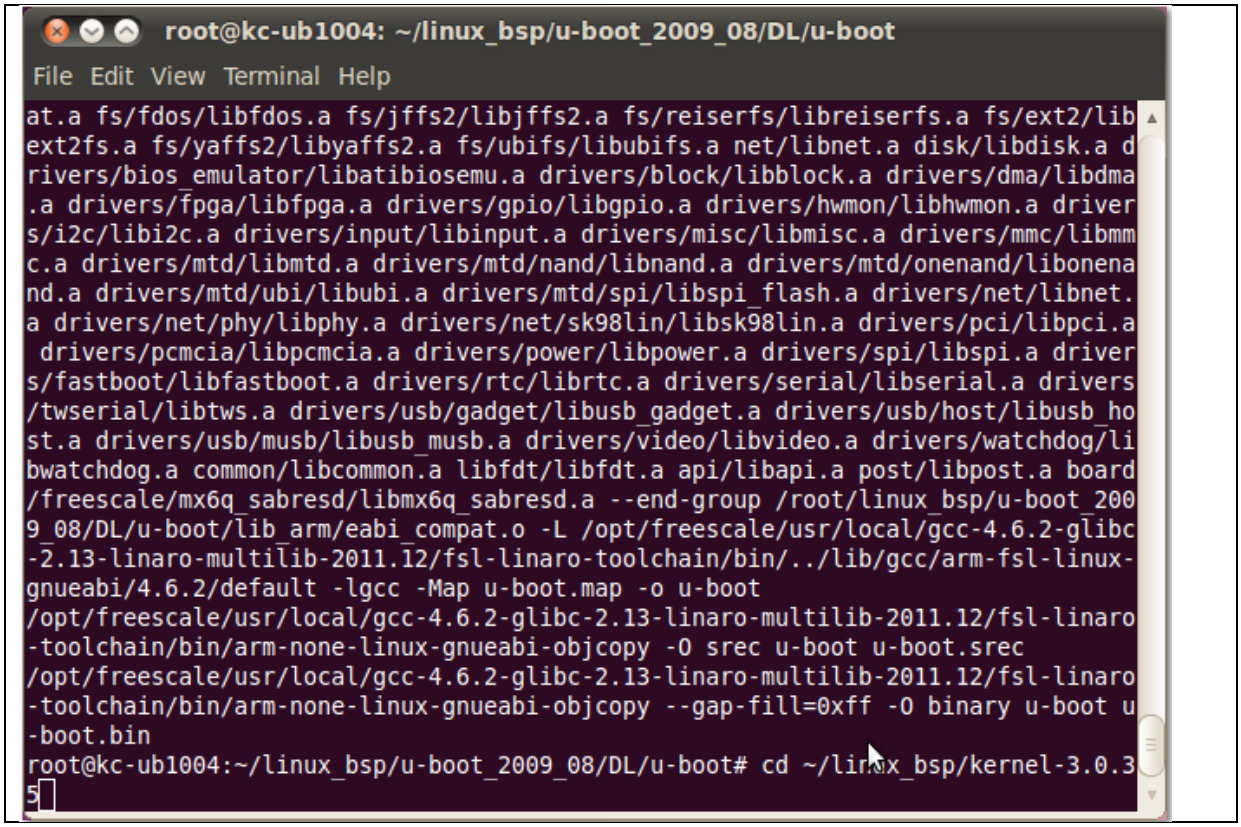
Note: **** If the building process is successful, **u-boot.bin** file will be generated. ****



5.1.4. Building kernel

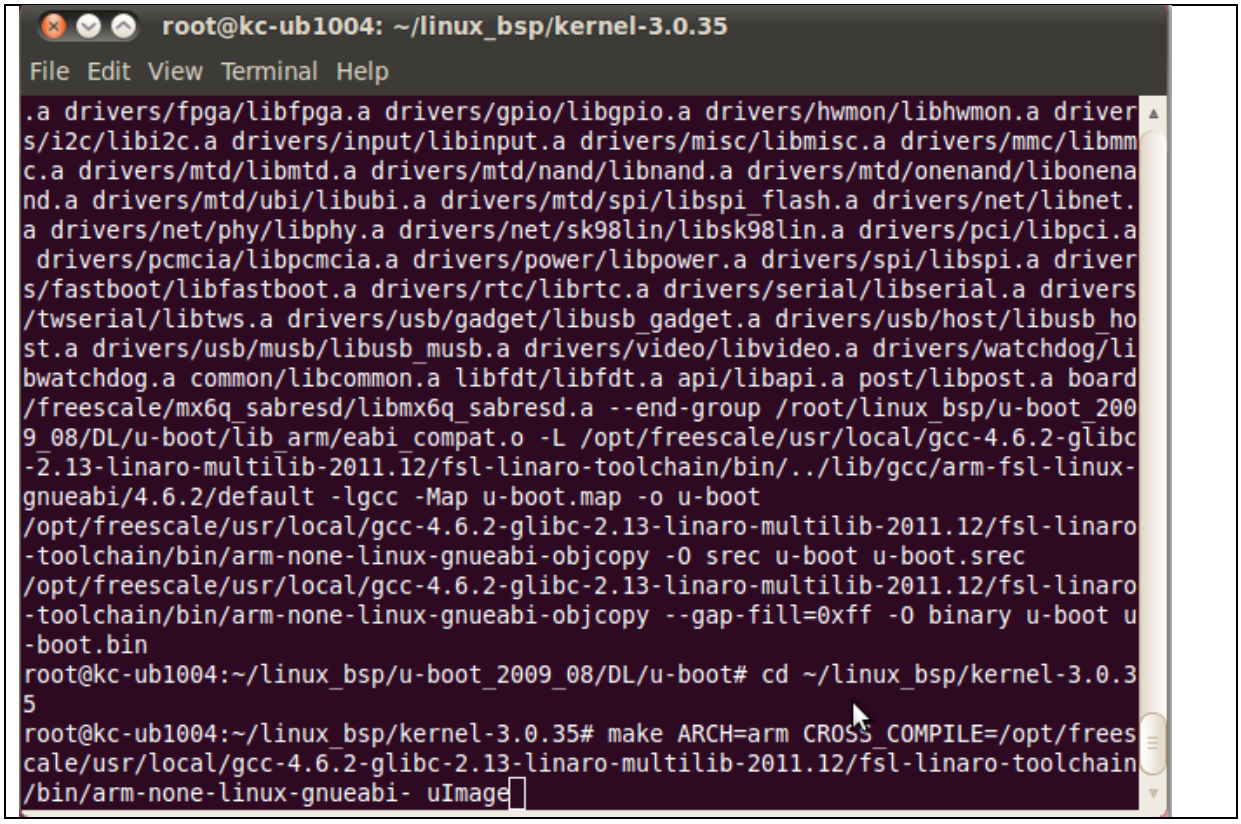
Assume your linux kernel source is at ~/linux_bsp/kernel-3.0.35

cd ~/linux_bsp/kernel-3.0.35

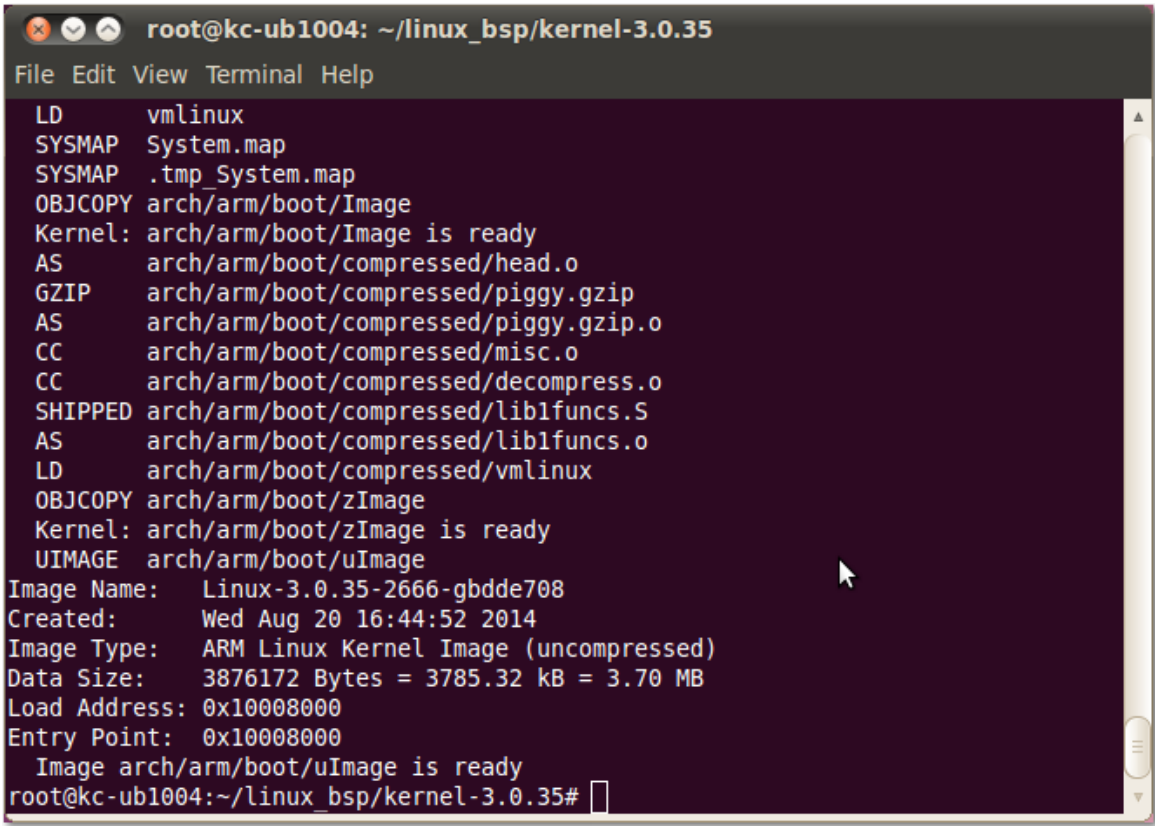


make ARCH=arm clean

make ARCH=arm CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/arm-none-linux-gnueabi- uImage



**** If the building process is successful, **uImage** file will be generated under arch/arm/boot directory. ****



5.1.5. Copying u-boot, kernel to SD card

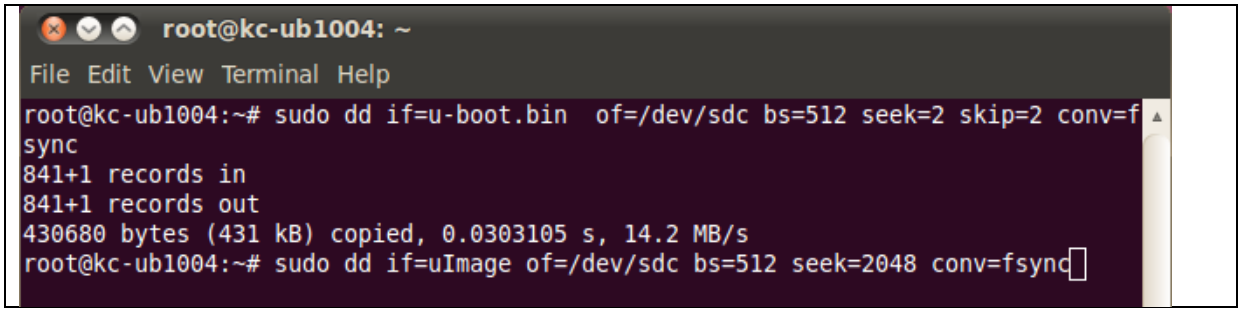
Insert an empty SD card with at least 4GB size and put it in a card reader connecting to your host PC. Assume your SD card is /dev/sdb on your x86 host PC

Copying the u-boot Boot Loader Image

sudo dd if=u-boot.bin of=/dev/sdb bs=512 seek=2 skip=2 conv=fsync

Copying the Kernel Image

sudo dd if=uImage of=/dev/sdb bs=512 seek=2048 conv=fsync



5.1.6. Copying Filesystem to SD card

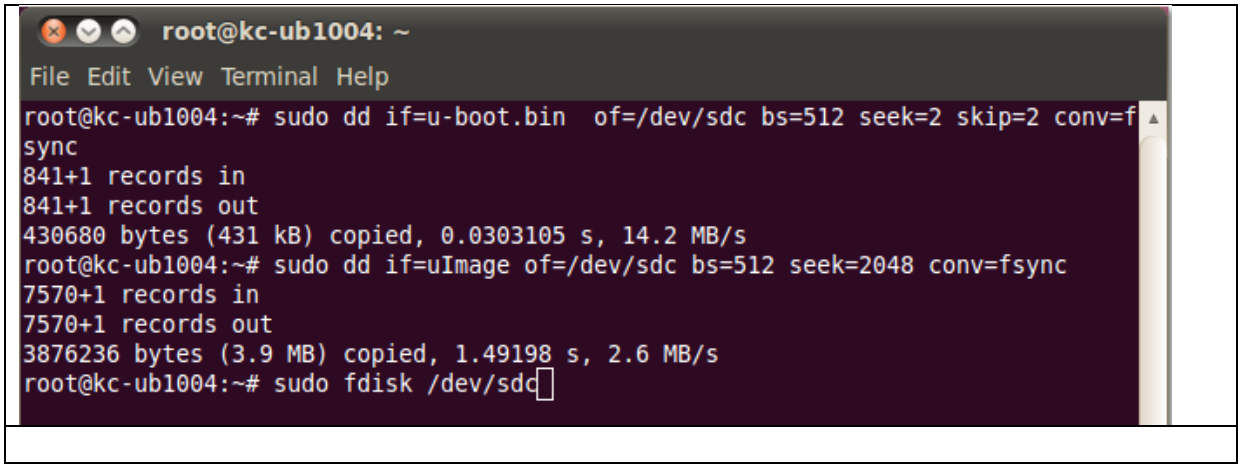
Assume your SD card is /dev/sdb.

Copying the Root File System (rootfs)

First, a partition table must be created. If a partition already exists and it is big enough for the file system you want to deploy, then you can skip this step.

To create a partition, at offset 16384 (in sectors of 512 bytes) enter the following command:

sudo fdisk /dev/sdb



NOTE

On most Linux host operating systems, SD card will be mounted automatically upon insertion. Therefore, before running fdisk, **please make sure that SD card is unmounted** (via 'sudo umount /dev/sdb').

Type the following parameters (each followed by <ENTER>):

u [switch the unit to sectors instead of cylinders]

d [repeat this until no partition is reported by the 'p' command]

n [create a new partition]

p [create a primary partition]

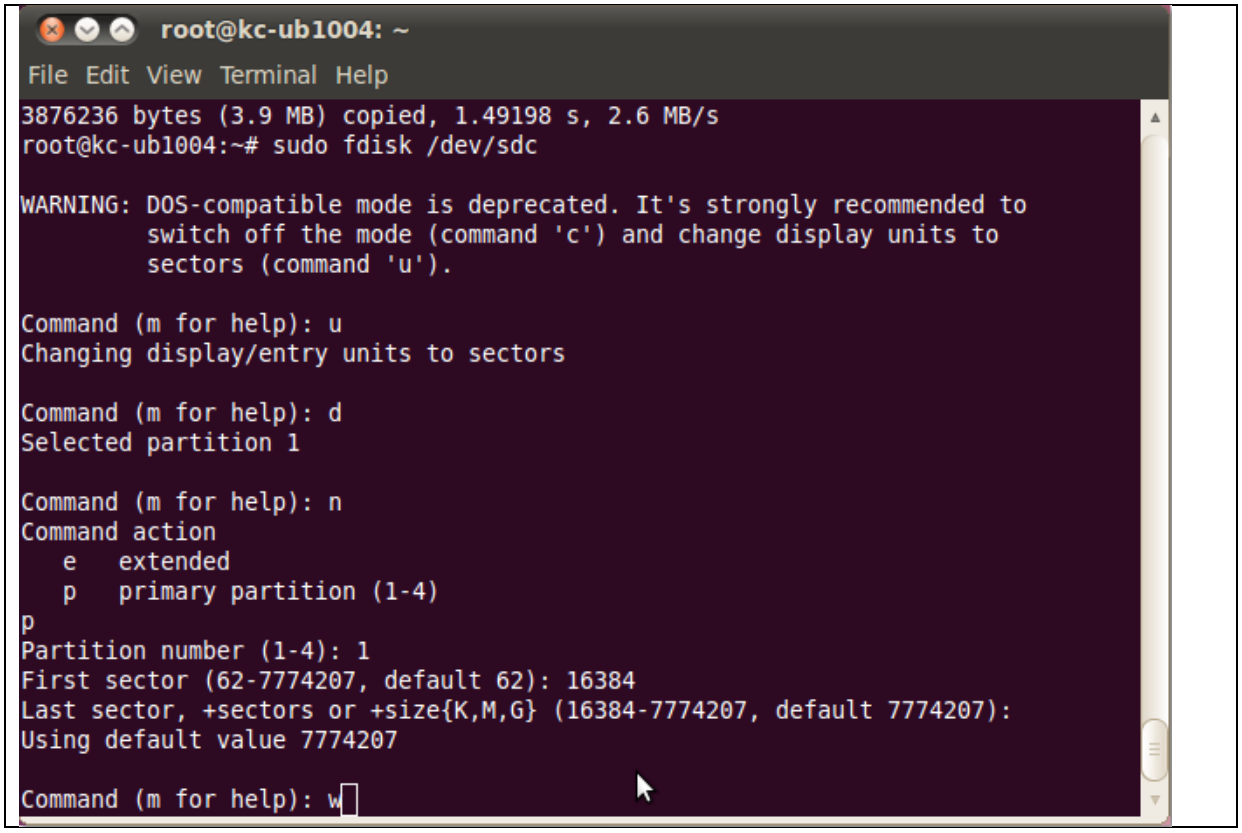
1 [the first partition]

16384 [starting at offset sector #16384, i.e. 8MB, which leaves enough space for the kernel, the boot loader and its configuration data]

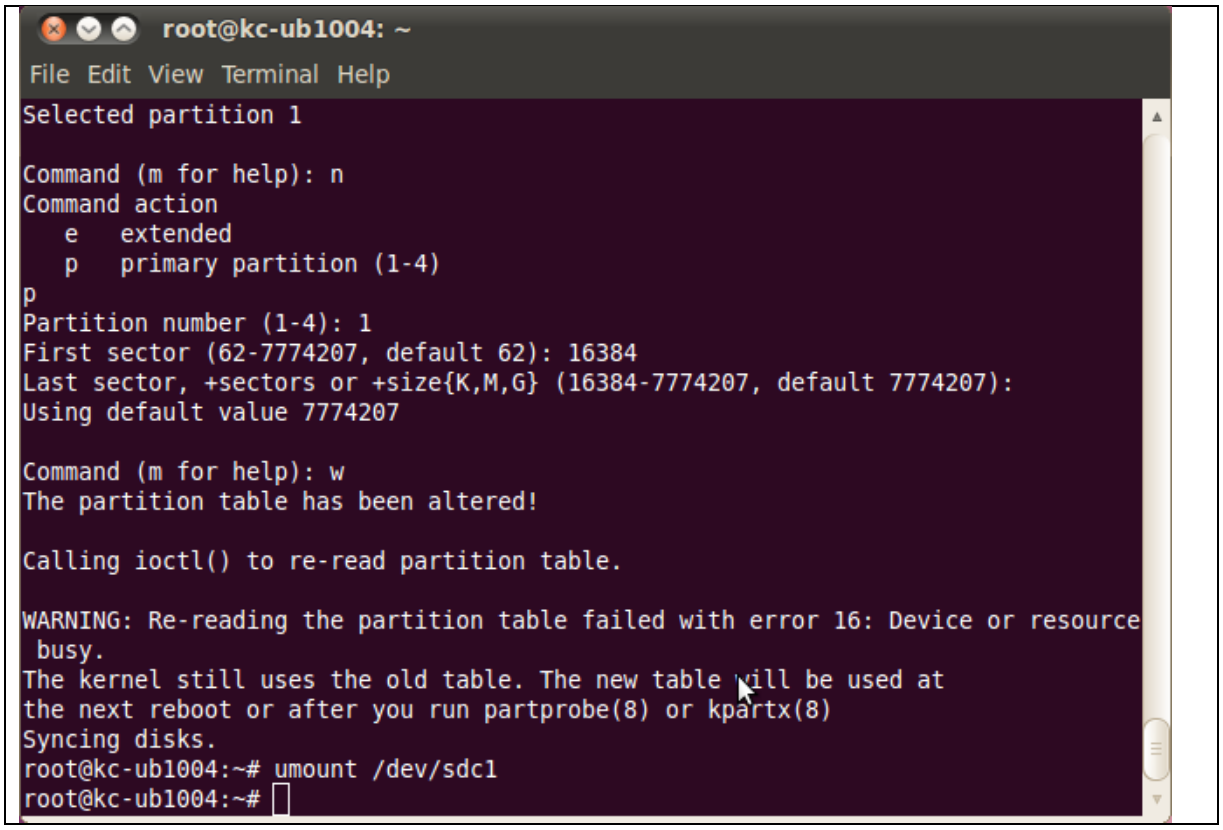
<enter> [using the default value will create a partition that spans to the last sector of the medium]

w [this writes the partition table to the medium and fdisk exits]

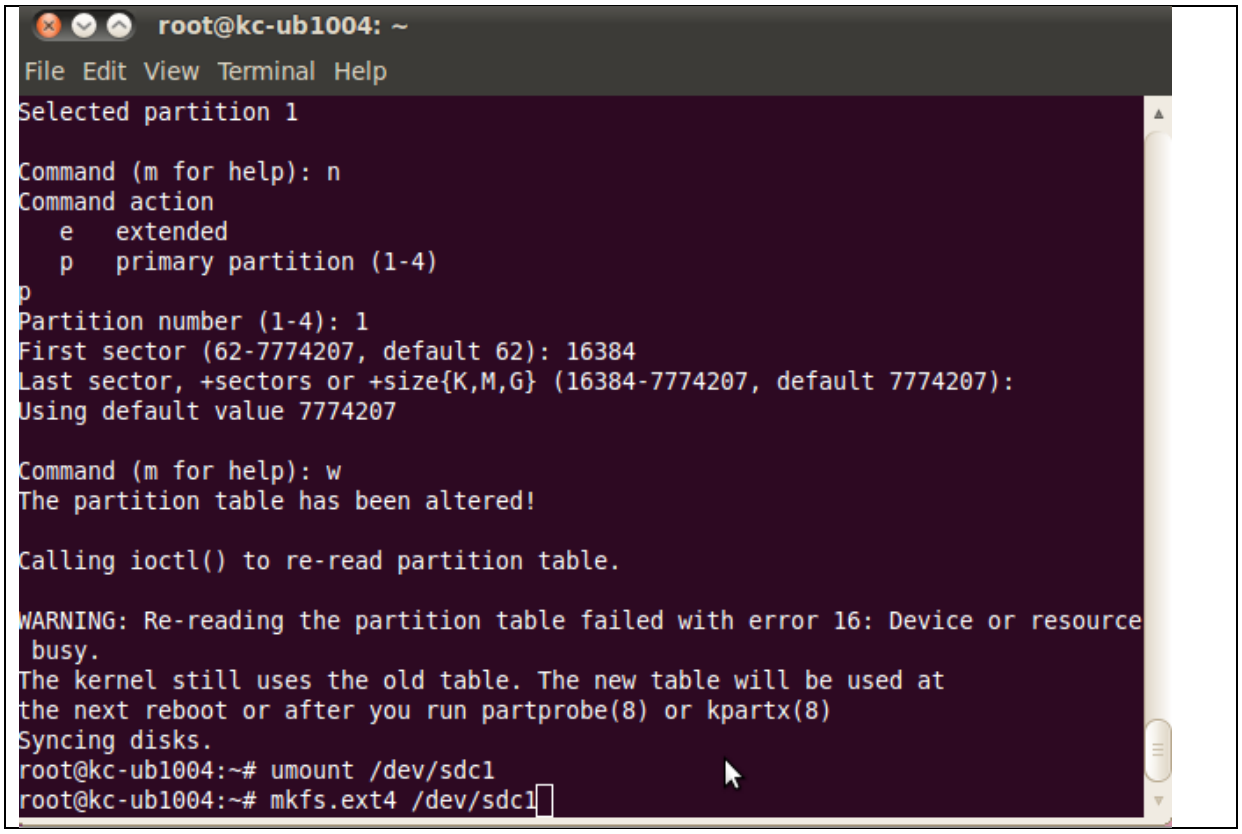
The file system format ext3 or ext4 is a good option for removable media due to the built-in journaling. Run the following command to format the partition:



sudo umount /dev/sdb1



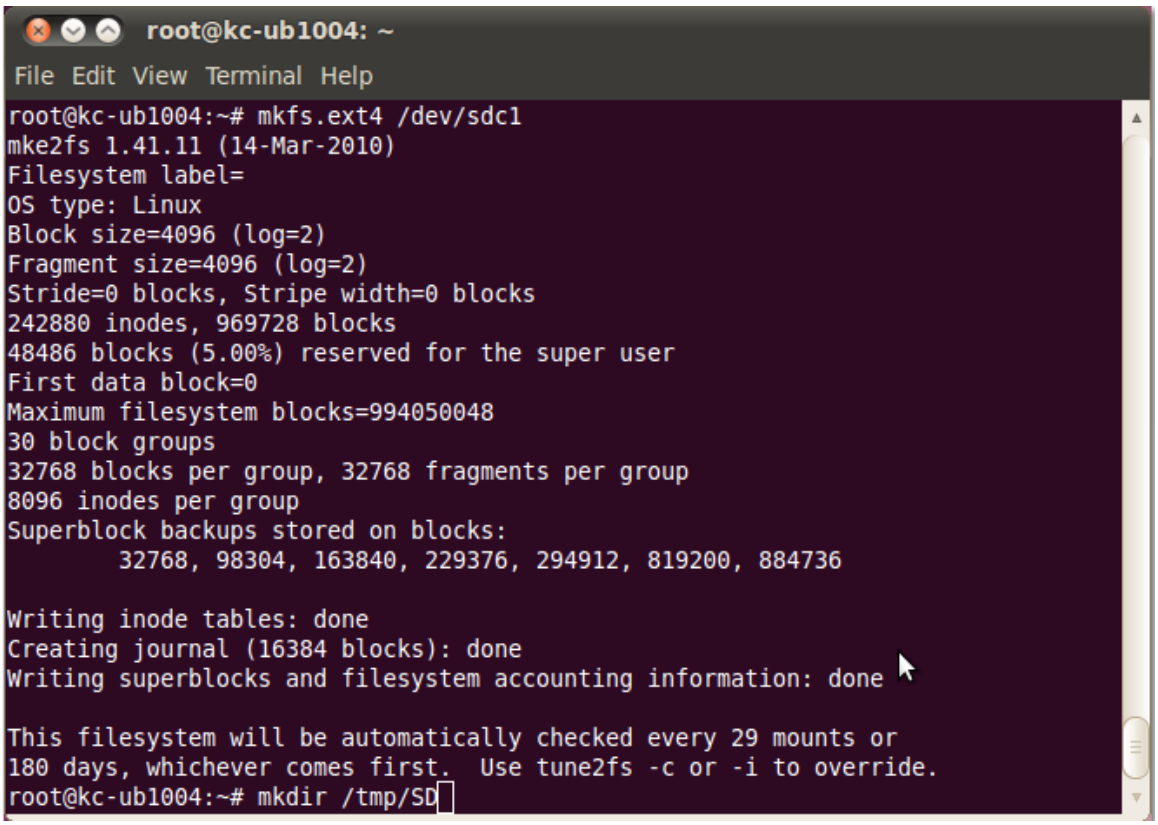
sudo mkfs.ext4 /dev/sdb1



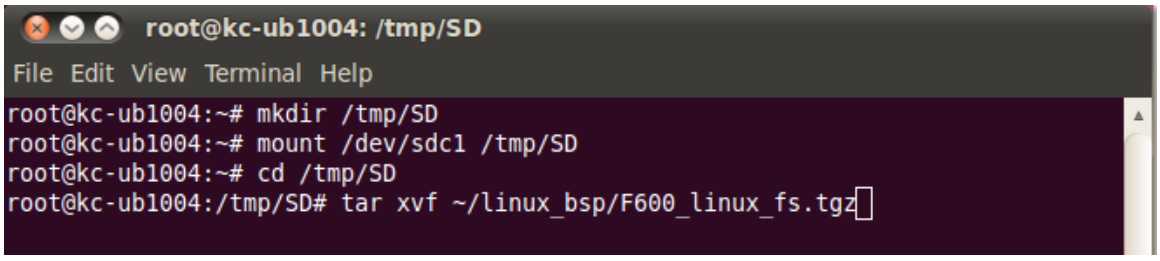
Copy the target file system to SD card partition by extracting rootfs package to mounted directory:

(assume compressed root file system is F6DU1_linux_fs.tgz)

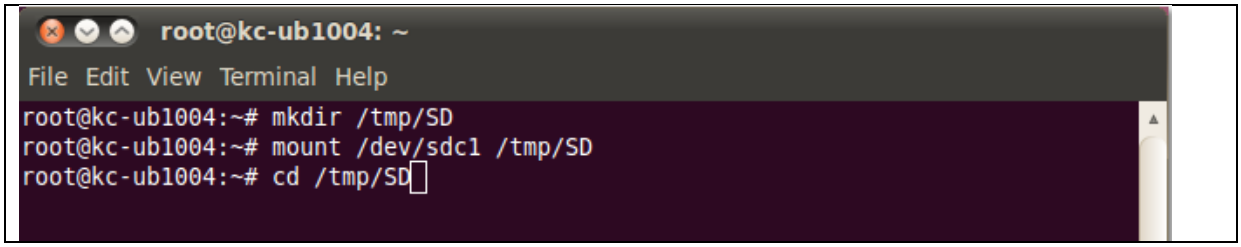
mkdir /tmp/SD



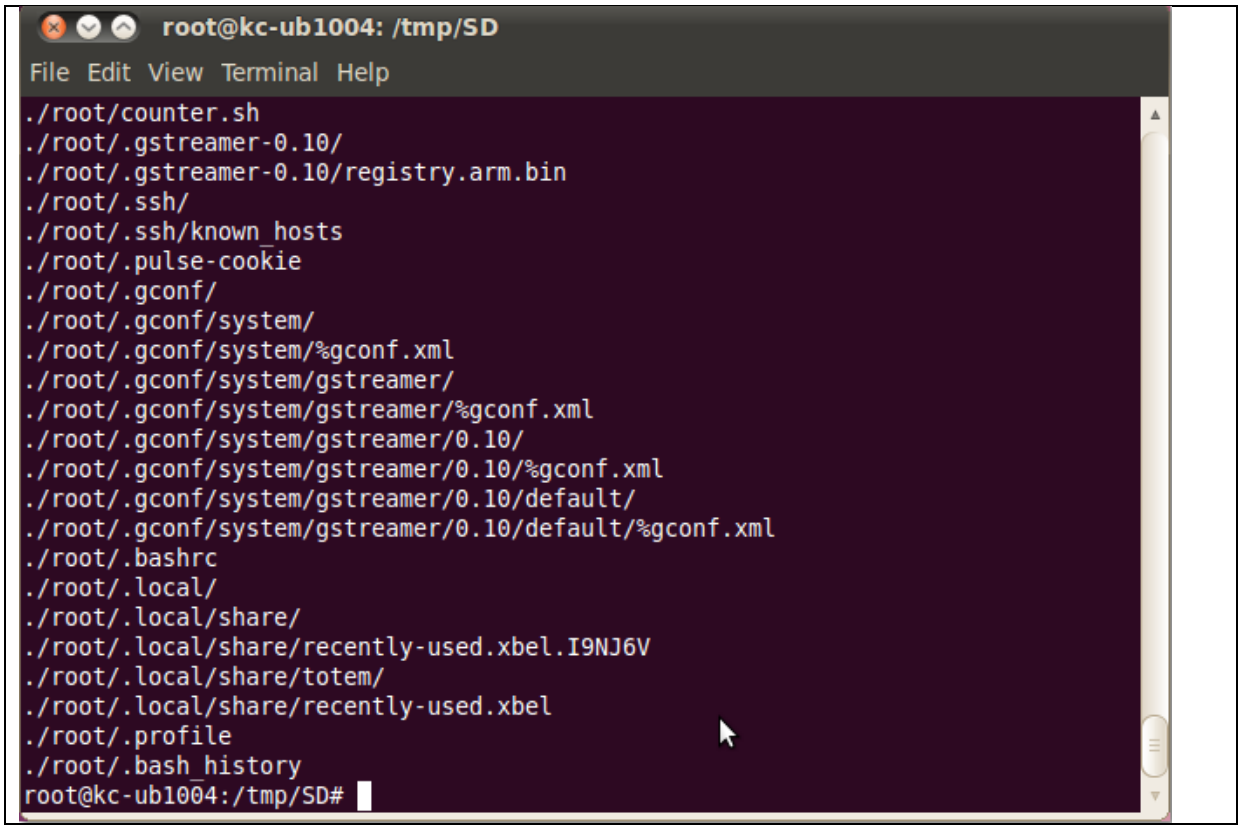
sudo mount /dev/sdb1 /tmp/SD



cd /tmp/SD



tar xvf ~/linux_bsp/F6DU1_linux_fs.tgz



Copying the file system takes several minutes. The file system content is now on the media.

5.1.7. Booting with your SD card

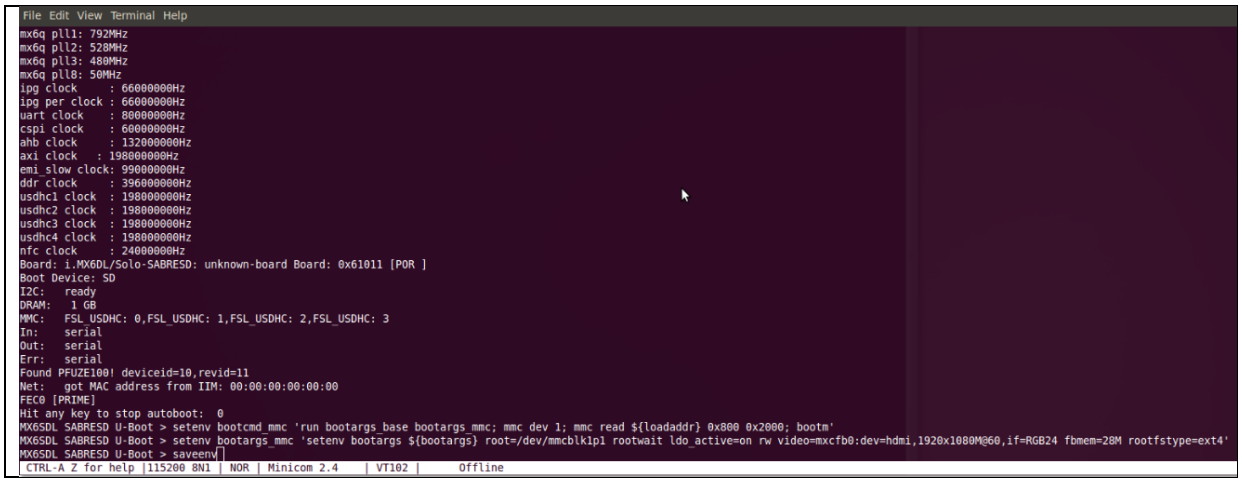
Put SD card in your board and adjust DIP switch to boot from SD. Connect a debug cable to debug port with serial port 115200/N/8/1 setting on your PC's serial port program such hyperterminal/teraterm. Connect a 1080p HDMI monitor to hdmi port. Power on and you will see u-boot prompt.

At u-boot prompt, press Enter before time out. Type the following setting to boot from SD card + HDMI:

setenv bootcmd_mmc 'run bootargs_base bootargs_mmc; mmc dev 1; mmc read ${loadaddr} 0x800 0x2000; bootm'

setenv bootargs_mmc 'setenv bootargs ${bootargs} root=/dev/mmcblk1p1 rootwait ldo_active=on rw video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24 fbmem=28M rootfstype=ext4'

saveenv



After that, prepare your HDMI LCD, power off and ***power on again.***

You can see Ubuntu Linux is running on hdmi monitor.

6. Appendix A– I2C, GPIO, watchdog reference code Coding.

6.1. How to use I2C in Linux

Reading / writing i2c

i2cget.c

```
/*
    i2cget.c - A user-space program to read an I2C register.
    Copyright (C) 2005-2012  Jean Delvare <jdelvare@suse.de>

    Based on i2cset.c:
    Copyright (C) 2001-2003  Frodo Looijaard <frodol@dds.nl>, and
                             Mark D. Studebaker <mdsxyz123@yahoo.com>
    Copyright (C) 2004-2005  Jean Delvare

    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, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
    MA 02110-1301 USA.
*/

#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include "i2cbusses.h"
#include "util.h"
#include "../version.h"

static void help(void) __attribute__ ((noreturn));

static void help(void)
{
    fprintf(stderr,
        "Usage: i2cget [-f] [-y] I2CBUS CHIP-ADDRESS [DATA-ADDRESS [MODE]]\n"
        "    I2CBUS is an integer or an I2C bus name\n"
        "    ADDRESS is an integer (0x03 - 0x77)\n"
        "    MODE is one of:\n"
        "        b (read byte data, default)\n"
        "        w (read word data)\n"
        "        c (write byte/read byte)\n"
        "        Append p for SMBus PEC\n");
    exit(1);
}

static int check_funcs(int file, int size, int daddress, int pec)
{
    unsigned long funcs;

    /* check adapter functionality */
    if (ioctl(file, I2C_FUNCS, &funcs) < 0) {
        fprintf(stderr, "Error: Could not get the adapter "
            "functionality matrix: %s\n", strerror(errno));
```

```
            return -1;
        }

        switch (size) {
        case I2C_SMBUS_BYTE:
            if (!(funcs & I2C_FUNC_SMBUS_READ_BYTE)) {
                fprintf(stderr, MISSING_FUNC_FMT, "SMBus receive byte");
                return -1;
            }
            if (daddress >= 0
             && !(funcs & I2C_FUNC_SMBUS_WRITE_BYTE)) {
                fprintf(stderr, MISSING_FUNC_FMT, "SMBus send byte");
                return -1;
            }
            break;

        case I2C_SMBUS_BYTE_DATA:
            if (!(funcs & I2C_FUNC_SMBUS_READ_BYTE_DATA)) {
                fprintf(stderr, MISSING_FUNC_FMT, "SMBus read byte");
                return -1;
            }
            break;

        case I2C_SMBUS_WORD_DATA:
            if (!(funcs & I2C_FUNC_SMBUS_READ_WORD_DATA)) {
                fprintf(stderr, MISSING_FUNC_FMT, "SMBus read word");
                return -1;
            }
            break;
        }

        if (pec
         && !(funcs & (I2C_FUNC_SMBUS_PEC | I2C_FUNC_I2C))) {
            fprintf(stderr, "Warning: Adapter does "
                    "not seem to support PEC\n");
        }

        return 0;
}

static int confirm(const char *filename, int address, int size, int daddress,
                   int pec)
{
        int dont = 0;

        fprintf(stderr, "WARNING! This program can confuse your I2C "
                "bus, cause data loss and worse!\n");

        /* Don't let the user break his/her EEPROMs */
        if (address >= 0x50 && address <= 0x57 && pec) {
            fprintf(stderr, "STOP! EEPROMs are I2C devices, not "
                    "SMBus devices. Using PEC\non I2C devices may "
                    "result in unexpected results, such as\n"
                    "trashing the contents of EEPROMs. We can't "
                    "let you do that, sorry.\n");
            return 0;
        }

        if (size == I2C_SMBUS_BYTE && daddress >= 0 && pec) {
            fprintf(stderr, "WARNING! All I2C chips and some SMBus chips "
                    "will interpret a write\nbyte command with PEC as a"
                    "write byte data command, effectively writing a\n"
                    "value into a register!\n");
            dont++;
        }

        fprintf(stderr, "I will read from device file %s, chip "
                "address 0x%02x, ", filename, address);
        if (daddress < 0)
            fprintf(stderr, "current data\naddress");
        else
```

```
                fprintf(stderr, "data address\n0x%02x", daddress);
        fprintf(stderr, ", using %s.\n",
                size == I2C_SMBUS_BYTE ? (daddress < 0 ?
                "read byte" : "write byte/read byte") :
                size == I2C_SMBUS_BYTE_DATA ? "read byte data" :
                "read word data");
        if (pec)
                fprintf(stderr, "PEC checking enabled.\n");

        fprintf(stderr, "Continue? [%s] ", dont ? "y/N" : "Y/n");
        fflush(stderr);
        if (!user_ack(!dont)) {
                fprintf(stderr, "Aborting on user request.\n");
                return 0;
        }

        return 1;
}

int main(int argc, char *argv[])
{
        char *end;
        int res, i2cbus, address, size, file;
        int daddress;
        char filename[20];
        int pec = 0;
        int flags = 0;
        int force = 0, yes = 0, version = 0;

        /* handle (optional) flags first */
        while (1+flags < argc && argv[1+flags][0] == '-') {
                switch (argv[1+flags][1]) {
                case 'V': version = 1; break;
                case 'f': force = 1; break;
                case 'y': yes = 1; break;
                default:
                        fprintf(stderr, "Error: Unsupported option "
                                "\"%s\"!\n", argv[1+flags]);
                        help();
                        exit(1);
                }
                flags++;
        }

        if (version) {
                fprintf(stderr, "i2cget version %s\n", VERSION);
                exit(0);
        }

        if (argc < flags + 3)
                help();

        i2cbus = lookup_i2c_bus(argv[flags+1]);
        if (i2cbus < 0)
                help();

        address = parse_i2c_address(argv[flags+2]);
        if (address < 0)
                help();

        if (argc > flags + 3) {
                size = I2C_SMBUS_BYTE_DATA;
                daddress = strtol(argv[flags+3], &end, 0);
                if (*end || daddress < 0 || daddress > 0xff) {
                        fprintf(stderr, "Error: Data address invalid!\n");
                        help();
                }
        } else {
                size = I2C_SMBUS_BYTE;
                daddress = -1;
        }
```

```
	if (argc > flags + 4) {
		switch (argv[flags+4][0]) {
		case 'b': size = I2C_SMBUS_BYTE_DATA; break;
		case 'w': size = I2C_SMBUS_WORD_DATA; break;
		case 'c': size = I2C_SMBUS_BYTE; break;
		default:
			fprintf(stderr, "Error: Invalid mode!\n");
			help();
		}
		pec = argv[flags+4][1] == 'p';
	}

	file = open_i2c_dev(i2cbus, filename, sizeof(filename), 0);
	if (file < 0
	 || check_funcs(file, size, daddress, pec)
	 || set_slave_addr(file, address, force))
		exit(1);

	if (!yes && !confirm(filename, address, size, daddress, pec))
		exit(0);

	if (pec && ioctl(file, I2C_PEC, 1) < 0) {
		fprintf(stderr, "Error: Could not set PEC: %s\n",
			strerror(errno));
		close(file);
		exit(1);
	}

	switch (size) {
	case I2C_SMBUS_BYTE:
		if (daddress >= 0) {
			res = i2c_smbus_write_byte(file, daddress);
			if (res < 0)
				fprintf(stderr, "Warning - write failed\n");
		}
		res = i2c_smbus_read_byte(file);
		break;
	case I2C_SMBUS_WORD_DATA:
		res = i2c_smbus_read_word_data(file, daddress);
		break;
	default: /* I2C_SMBUS_BYTE_DATA */
		res = i2c_smbus_read_byte_data(file, daddress);
	}
	close(file);

	if (res < 0) {
		fprintf(stderr, "Error: Read failed\n");
		exit(2);
	}

	printf("0x%0*x\n", size == I2C_SMBUS_WORD_DATA ? 4 : 2, res);

	exit(0);
}

i2cset.c
/*
	i2cset.c - A user-space program to write an I2C register.
	Copyright (C) 2001-2003  Frodo Looijaard <frodol@dds.nl>, and
	                         Mark D. Studebaker <mdsxyz123@yahoo.com>
	Copyright (C) 2004-2012  Jean Delvare <jdelvare@suse.de>

	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, or
	(at your option) any later version.

	This program is distributed in the hope that it will be useful,
	but WITHOUT ANY WARRANTY; without even the implied warranty of
	MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
	GNU General Public License for more details.

	You should have received a copy of the GNU General Public License
	along with this program; if not, write to the Free Software
	Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
	MA 02110-1301 USA.
*/

#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include "i2cbusses.h"
#include "util.h"
#include "../version.h"

static void help(void) __attribute__ ((noreturn));

static void help(void)
{
	fprintf(stderr,
		"Usage: i2cset [-f] [-y] [-m MASK] [-r] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE] ... [MODE]\n"
		"    I2CBUS is an integer or an I2C bus name\n"
		"    ADDRESS is an integer (0x03 - 0x77)\n"
		"    MODE is one of:\n"
		"        c (byte, no value)\n"
		"        b (byte data, default)\n"
		"        w (word data)\n"
		"        i (I2C block data)\n"
		"        s (SMBus block data)\n"
		"        Append p for SMBus PEC\n");
	exit(1);
}

static int check_funcs(int file, int size, int pec)
{
	unsigned long funcs;

	/* check adapter functionality */
	if (ioctl(file, I2C_FUNCS, &funcs) < 0) {
		fprintf(stderr, "Error: Could not get the adapter "
			"functionality matrix: %s\n", strerror(errno));
		return -1;
	}

	switch (size) {
	case I2C_SMBUS_BYTE:
		if (!(funcs & I2C_FUNC_SMBUS_WRITE_BYTE)) {
			fprintf(stderr, MISSING_FUNC_FMT, "SMBus send byte");
			return -1;
		}
		break;

	case I2C_SMBUS_BYTE_DATA:
		if (!(funcs & I2C_FUNC_SMBUS_WRITE_BYTE_DATA)) {
			fprintf(stderr, MISSING_FUNC_FMT, "SMBus write byte");
			return -1;
		}
		break;

	case I2C_SMBUS_WORD_DATA:
		if (!(funcs & I2C_FUNC_SMBUS_WRITE_WORD_DATA)) {
			fprintf(stderr, MISSING_FUNC_FMT, "SMBus write word");
			return -1;
		}
		break;

	case I2C_SMBUS_BLOCK_DATA:
```

```
                if (!(funcs & I2C_FUNC_SMBUS_WRITE_BLOCK_DATA)) {
                        fprintf(stderr, MISSING_FUNC_FMT, "SMBus block write");
                        return -1;
                }
                break;
        case I2C_SMBUS_I2C_BLOCK_DATA:
                if (!(funcs & I2C_FUNC_SMBUS_WRITE_I2C_BLOCK)) {
                        fprintf(stderr, MISSING_FUNC_FMT, "I2C block write");
                        return -1;
                }
                break;
        }

        if (pec
         && !(funcs & (I2C_FUNC_SMBUS_PEC | I2C_FUNC_I2C))) {
                fprintf(stderr, "Warning: Adapter does "
                        "not seem to support PEC\n");
        }

        return 0;
}

static int confirm(const char *filename, int address, int size, int daddress,
                   int value, int vmask, const unsigned char *block, int len,
                   int pec)
{
        int dont = 0;

        fprintf(stderr, "WARNING! This program can confuse your I2C "
                "bus, cause data loss and worse!\n");

        if (address >= 0x50 && address <= 0x57) {
                fprintf(stderr, "DANGEROUS! Writing to a serial "
                        "EEPROM on a memory DIMM\nmay render your "
                        "memory USELESS and make your system "
                        "UNBOOTABLE!\n");
                dont++;
        }

        fprintf(stderr, "I will write to device file %s, chip address "
                "0x%02x, data address\n0x%02x, ", filename, address, daddress);
        if (size == I2C_SMBUS_BYTE)
                fprintf(stderr, "no data.\n");
        else if (size == I2C_SMBUS_BLOCK_DATA ||
                 size == I2C_SMBUS_I2C_BLOCK_DATA) {
                int i;

                fprintf(stderr, "data");
                for (i = 0; i < len; i++)
                        fprintf(stderr, " 0x%02x", block[i]);
                fprintf(stderr, ", mode %s.\n", size == I2C_SMBUS_BLOCK_DATA
                        ? "smbus block" : "i2c block");
        } else
                fprintf(stderr, "data 0x%02x%s, mode %s.\n", value,
                        vmask ? " (masked)" : "",
                        size == I2C_SMBUS_BYTE_DATA ? "byte" : "word");
        if (pec)
                fprintf(stderr, "PEC checking enabled.\n");

        fprintf(stderr, "Continue? [%s] ", dont ? "y/N" : "Y/n");
        fflush(stderr);
        if (!user_ack(!dont)) {
                fprintf(stderr, "Aborting on user request.\n");
                return 0;
        }

        return 1;
}

int main(int argc, char *argv[])
{
```

```
char *end;
const char *maskp = NULL;
int res, i2cbus, address, size, file;
int value, daddress, vmask = 0;
char filename[20];
int pec = 0;
int flags = 0;
int force = 0, yes = 0, version = 0, readback = 0;
unsigned char block[I2C_SMBUS_BLOCK_MAX];
int len;

/* handle (optional) flags first */
while (1+flags < argc && argv[1+flags][0] == '-') {
	switch (argv[1+flags][1]) {
	case 'V': version = 1; break;
	case 'f': force = 1; break;
	case 'y': yes = 1; break;
	case 'm':
		if (2+flags < argc)
			maskp = argv[2+flags];
		flags++;
		break;
	case 'r': readback = 1; break;
	default:
		fprintf(stderr, "Error: Unsupported option "
			"\"%s\"!\n", argv[1+flags]);
		help();
		exit(1);
	}
	flags++;
}

if (version) {
	fprintf(stderr, "i2cset version %s\n", VERSION);
	exit(0);
}

if (argc < flags + 4)
	help();

i2cbus = lookup_i2c_bus(argv[flags+1]);
if (i2cbus < 0)
	help();

address = parse_i2c_address(argv[flags+2]);
if (address < 0)
	help();

daddress = strtol(argv[flags+3], &end, 0);
if (*end || daddress < 0 || daddress > 0xff) {
	fprintf(stderr, "Error: Data address invalid!\n");
	help();
}

/* check for command/mode */
if (argc == flags + 4) {
	/* Implicit "c" */
	size = I2C_SMBUS_BYTE;
} else if (argc == flags + 5) {
	/* "c", "cp",    or implicit "b" */
	if (!strcmp(argv[flags+4], "c")
	 || !strcmp(argv[flags+4], "cp")) {
		size = I2C_SMBUS_BYTE;
		pec = argv[flags+4][1] == 'p';
	} else {
		size = I2C_SMBUS_BYTE_DATA;
	}
} else {
	/* All other commands */
	if (strlen(argv[argc-1]) > 2
		|| (strlen(argv[argc-1]) == 2 && argv[argc-1][1] != 'p')) {
```

```
                fprintf(stderr, "Error: Invalid mode '%s'!\n", argv[argc-1]);
                help();
        }
        switch (argv[argc-1][0]) {
        case 'b': size = I2C_SMBUS_BYTE_DATA; break;
        case 'w': size = I2C_SMBUS_WORD_DATA; break;
        case 's': size = I2C_SMBUS_BLOCK_DATA; break;
        case 'i': size = I2C_SMBUS_I2C_BLOCK_DATA; break;
        default:
                fprintf(stderr, "Error: Invalid mode '%s'!\n", argv[argc-1]);
                help();
        }
        pec = argv[argc-1][1] == 'p';
        if (size == I2C_SMBUS_BLOCK_DATA || size == I2C_SMBUS_I2C_BLOCK_DATA) {
                if (pec && size == I2C_SMBUS_I2C_BLOCK_DATA) {
                        fprintf(stderr, "Error: PEC not supported for I2C block writes!\n");
                        help();
                }
                if (maskp) {
                        fprintf(stderr, "Error: Mask not supported for block writes!\n");
                        help();
                }
                if (argc > (int)sizeof(block) + flags + 5) {
                        fprintf(stderr, "Error: Too many arguments!\n");
                        help();
                }
        } else if (argc != flags + 6) {
                fprintf(stderr, "Error: Too many arguments!\n");
                help();
        }
}

len = 0; /* Must always initialize len since it is passed to confirm() */

/* read values from command line */
switch (size) {
case I2C_SMBUS_BYTE_DATA:
case I2C_SMBUS_WORD_DATA:
        value = strtol(argv[flags+4], &end, 0);
        if (*end || value < 0) {
                fprintf(stderr, "Error: Data value invalid!\n");
                help();
        }
        if ((size == I2C_SMBUS_BYTE_DATA && value > 0xff)
                || (size == I2C_SMBUS_WORD_DATA && value > 0xffff)) {
                fprintf(stderr, "Error: Data value out of range!\n");
                help();
        }
        break;
case I2C_SMBUS_BLOCK_DATA:
case I2C_SMBUS_I2C_BLOCK_DATA:
        for (len = 0; len + flags + 5 < argc; len++) {
                value = strtol(argv[flags + len + 4], &end, 0);
                if (*end || value < 0) {
                        fprintf(stderr, "Error: Data value invalid!\n");
                        help();
                }
                if (value > 0xff) {
                        fprintf(stderr, "Error: Data value out of range!\n");
                        help();
                }
                block[len] = value;
        }
        value = -1;
        break;
default:
        value = -1;
        break;
}

if (maskp) {
```

```
        vmask = strtol(maskp, &end, 0);
        if (*end || vmask == 0) {
                fprintf(stderr, "Error: Data value mask invalid!\n");
                help();
        }
        if (((size == I2C_SMBUS_BYTE || size == I2C_SMBUS_BYTE_DATA)
             && vmask > 0xff) || vmask > 0xffff) {
                fprintf(stderr, "Error: Data value mask out of range!\n");
                help();
        }
}

file = open_i2c_dev(i2cbus, filename, sizeof(filename), 0);
if (file < 0
 || check_funcs(file, size, pec)
 || set_slave_addr(file, address, force))
        exit(1);

if (!yes && !confirm(filename, address, size, daddress,
                     value, vmask, block, len, pec))
        exit(0);

if (vmask) {
        int oldvalue;

        switch (size) {
        case I2C_SMBUS_BYTE:
                oldvalue = i2c_smbus_read_byte(file);
                break;
        case I2C_SMBUS_WORD_DATA:
                oldvalue = i2c_smbus_read_word_data(file, daddress);
                break;
        default:
                oldvalue = i2c_smbus_read_byte_data(file, daddress);
        }

        if (oldvalue < 0) {
                fprintf(stderr, "Error: Failed to read old value\n");
                exit(1);
        }

        value = (value & vmask) | (oldvalue & ~vmask);

        if (!yes) {
                fprintf(stderr, "Old value 0x%0*x, write mask "
                        "0x%0*x: Will write 0x%0*x to register "
                        "0x%02x\n",
                        size == I2C_SMBUS_WORD_DATA ? 4 : 2, oldvalue,
                        size == I2C_SMBUS_WORD_DATA ? 4 : 2, vmask,
                        size == I2C_SMBUS_WORD_DATA ? 4 : 2, value,
                        daddress);

                fprintf(stderr, "Continue? [Y/n] ");
                fflush(stderr);
                if (!user_ack(1)) {
                        fprintf(stderr, "Aborting on user request.\n");
                        exit(0);
                }
        }
}

if (pec && ioctl(file, I2C_PEC, 1) < 0) {
        fprintf(stderr, "Error: Could not set PEC: %s\n",
                strerror(errno));
        close(file);
        exit(1);
}

switch (size) {
case I2C_SMBUS_BYTE:
        res = i2c_smbus_write_byte(file, daddress);
```

```
            break;
        case I2C_SMBUS_WORD_DATA:
            res = i2c_smbus_write_word_data(file, daddress, value);
            break;
        case I2C_SMBUS_BLOCK_DATA:
            res = i2c_smbus_write_block_data(file, daddress, len, block);
            break;
        case I2C_SMBUS_I2C_BLOCK_DATA:
            res = i2c_smbus_write_i2c_block_data(file, daddress, len, block);
            break;
        default: /* I2C_SMBUS_BYTE_DATA */
            res = i2c_smbus_write_byte_data(file, daddress, value);
            break;
        }
        if (res < 0) {
            fprintf(stderr, "Error: Write failed\n");
            close(file);
            exit(1);
        }

        if (pec) {
            if (ioctl(file, I2C_PEC, 0) < 0) {
                fprintf(stderr, "Error: Could not clear PEC: %s\n",
                        strerror(errno));
                close(file);
                exit(1);
            }
        }

        if (!readback) { /* We're done */
            close(file);
            exit(0);
        }

        switch (size) {
        case I2C_SMBUS_BYTE:
            res = i2c_smbus_read_byte(file);
            value = daddress;
            break;
        case I2C_SMBUS_WORD_DATA:
            res = i2c_smbus_read_word_data(file, daddress);
            break;
        default: /* I2C_SMBUS_BYTE_DATA */
            res = i2c_smbus_read_byte_data(file, daddress);
        }
        close(file);

        if (res < 0) {
            printf("Warning - readback failed\n");
        } else
        if (res != value) {
            printf("Warning - data mismatch - wrote "
                   "0x%0*x, read back 0x%0*x\n",
                   size == I2C_SMBUS_WORD_DATA ? 4 : 2, value,
                   size == I2C_SMBUS_WORD_DATA ? 4 : 2, res);
        } else {
            printf("Value 0x%0*x written, readback matched\n",
                   size == I2C_SMBUS_WORD_DATA ? 4 : 2, value);
        }

        exit(0);
}

Utils/headers
/*
        i2cbusses: Print the installed i2c busses for both 2.4 and 2.6 kernels.
                   Part of user-space programs to access for I2C
                   devices.
*/

/* For strdup and snprintf */
```

```
#define _BSD_SOURCE 1

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>       /* for NAME_MAX */
#include <sys/ioctl.h>
#include <string.h>
#include <strings.h>       /* for strcasecmp() */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <dirent.h>
#include <fcntl.h>
#include <errno.h>
#include "i2cbusses.h"
#include <linux/i2c-dev.h>

enum adt { adt_dummy, adt_isa, adt_i2c, adt_smbus, adt_unknown };

struct adap_type {
	const char *funcs;
	const char* algo;
};

static struct adap_type adap_types[5] = {
	{ .funcs	= "dummy",
	  .algo		= "Dummy bus", },
	{ .funcs	= "isa",
	  .algo		= "ISA bus", },
	{ .funcs	= "i2c",
	  .algo		= "I2C adapter", },
	{ .funcs	= "smbus",
	  .algo		= "SMBus adapter", },
	{ .funcs	= "unknown",
	  .algo		= "N/A", },
};

static enum adt i2c_get_funcs(int i2cbus)
{
	unsigned long funcs;
	int file;
	char filename[20];
	enum adt ret;

	file = open_i2c_dev(i2cbus, filename, sizeof(filename), 1);
	if (file < 0)
		return adt_unknown;

	if (ioctl(file, I2C_FUNCS, &funcs) < 0)
		ret = adt_unknown;
	else if (funcs & I2C_FUNC_I2C)
		ret = adt_i2c;
	else if (funcs & (I2C_FUNC_SMBUS_BYTE |
			  I2C_FUNC_SMBUS_BYTE_DATA |
			  I2C_FUNC_SMBUS_WORD_DATA))
		ret = adt_smbus;
	else
		ret = adt_dummy;

	close(file);
	return ret;
}

/* Remove trailing spaces from a string
   Return the new string length including the trailing NUL */
static int rtrim(char *s)
{
	int i;

	for (i = strlen(s) - 1; i >= 0 && (s[i] == ' ' || s[i] == '\n'); i--)
```

```
                s[i] = '\0';
        return i + 2;
}

void free_adapters(struct i2c_adap *adapters)
{
        int i;

        for (i = 0; adapters[i].name; i++)
                free(adapters[i].name);
        free(adapters);
}

/* We allocate space for the adapters in bunches. The last item is a
   terminator, so here we start with room for 7 adapters, which should
   be enough in most cases. If not, we allocate more later as needed. */
#define BUNCH       8

/* n must match the size of adapters at calling time */
static struct i2c_adap *more_adapters(struct i2c_adap *adapters, int n)
{
        struct i2c_adap *new_adapters;

        new_adapters = realloc(adapters, (n + BUNCH) * sizeof(struct i2c_adap));
        if (!new_adapters) {
                free_adapters(adapters);
                return NULL;
        }
        memset(new_adapters + n, 0, BUNCH * sizeof(struct i2c_adap));

        return new_adapters;
}

struct i2c_adap *gather_i2c_busses(void)
{
        char s[120];
        struct dirent *de, *dde;
        DIR *dir, *ddir;
        FILE *f;
        char fstype[NAME_MAX], sysfs[NAME_MAX], n[NAME_MAX];
        int foundsysfs = 0;
        int count=0;
        struct i2c_adap *adapters;

        adapters = calloc(BUNCH, sizeof(struct i2c_adap));
        if (!adapters)
                return NULL;

        /* look in /proc/bus/i2c */
        if ((f = fopen("/proc/bus/i2c", "r"))) {
                while (fgets(s, 120, f)) {
                        char *algo, *name, *type, *all;
                        int len_algo, len_name, len_type;
                        int i2cbus;

                        algo = strrchr(s, '\t');
                        *(algo++) = '\0';
                        len_algo = rtrim(algo);

                        name = strrchr(s, '\t');
                        *(name++) = '\0';
                        len_name = rtrim(name);

                        type = strrchr(s, '\t');
                        *(type++) = '\0';
                        len_type = rtrim(type);

                        sscanf(s, "i2c-%d", &i2cbus);

                        if ((count + 1) % BUNCH == 0) {
                                /* We need more space */
```

```
                        adapters = more_adapters(adapters, count + 1);
                        if (!adapters)
                                return NULL;
                }

                all = malloc(len_name + len_type + len_algo);
                if (all == NULL) {
                        free_adapters(adapters);
                        return NULL;
                }
                adapters[count].nr = i2cbus;
                adapters[count].name = strcpy(all, name);
                adapters[count].funcs = strcpy(all + len_name, type);
                adapters[count].algo = strcpy(all + len_name + len_type,
                                              algo);
                count++;
        }
        fclose(f);
        goto done;
}

/* look in sysfs */
/* First figure out where sysfs was mounted */
if ((f = fopen("/proc/mounts", "r")) == NULL) {
        goto done;
}
while (fgets(n, NAME_MAX, f)) {
        sscanf(n, "%*[^ ] %[^ ] %[^ ] %*s\n", sysfs, fstype);
        if (strcasecmp(fstype, "sysfs") == 0) {
                foundsysfs++;
                break;
        }
}
fclose(f);
if (! foundsysfs) {
        goto done;
}

/* Bus numbers in i2c-adapter don't necessarily match those in
     i2c-dev and what we really care about are the i2c-dev numbers.
     Unfortunately the names are harder to get in i2c-dev */
strcat(sysfs, "/class/i2c-dev");
if(!(dir = opendir(sysfs)))
        goto done;
/* go through the busses */
while ((de = readdir(dir)) != NULL) {
        if (!strcmp(de->d_name, "."))
                continue;
        if (!strcmp(de->d_name, ".."))
                continue;

        /* this should work for kernels 2.6.5 or higher and */
        /* is preferred because is unambiguous */
        sprintf(n, "%s/%s/name", sysfs, de->d_name);
        f = fopen(n, "r");
        /* this seems to work for ISA */
        if(f == NULL) {
                sprintf(n, "%s/%s/device/name", sysfs, de->d_name);
                f = fopen(n, "r");
        }
        /* non-ISA is much harder */
        /* and this won't find the correct bus name if a driver
             has more than one bus */
        if(f == NULL) {
                sprintf(n, "%s/%s/device", sysfs, de->d_name);
                if(!(ddir = opendir(n)))
                        continue;
                while ((dde = readdir(ddir)) != NULL) {
                        if (!strcmp(dde->d_name, "."))
                                continue;
                        if (!strcmp(dde->d_name, ".."))
```

```
                        continue;
                    if ((!strncmp(dde->d_name, "i2c-", 4))) {
                        sprintf(n, "%s/%s/device/%s/name",
                            sysfs, de->d_name, dde->d_name);
                        if((f = fopen(n, "r")))
                            goto found;
                    }
                }
            }

found:
            if (f != NULL) {
                int i2cbus;
                enum adt type;
                char *px;

                px = fgets(s, 120, f);
                fclose(f);
                if (!px) {
                    fprintf(stderr, "%s: read error\n", n);
                    continue;
                }
                if ((px = strchr(s, '\n')) != NULL)
                    *px = 0;
                if (!sscanf(de->d_name, "i2c-%d", &i2cbus))
                    continue;
                if (!strncmp(s, "ISA ", 4)) {
                    type = adt_isa;
                } else {
                    /* Attempt to probe for adapter capabilities */
                    type = i2c_get_funcs(i2cbus);
                }

                if ((count + 1) % BUNCH == 0) {
                    /* We need more space */
                    adapters = more_adapters(adapters, count + 1);
                    if (!adapters)
                        return NULL;
                }

                adapters[count].nr = i2cbus;
                adapters[count].name = strdup(s);
                if (adapters[count].name == NULL) {
                    free_adapters(adapters);
                    return NULL;
                }
                adapters[count].funcs = adap_types[type].funcs;
                adapters[count].algo = adap_types[type].algo;
                count++;
            }
        }
        closedir(dir);

done:
        return adapters;
}

static int lookup_i2c_bus_by_name(const char *bus_name)
{
        struct i2c_adap *adapters;
        int i, i2cbus = -1;

        adapters = gather_i2c_busses();
        if (adapters == NULL) {
            fprintf(stderr, "Error: Out of memory!\n");
            return -3;
        }

        /* Walk the list of i2c busses, looking for the one with the
           right name */
        for (i = 0; adapters[i].name; i++) {
```

```
            if (strcmp(adapters[i].name, bus_name) == 0) {
                if (i2cbus >= 0) {
                    fprintf(stderr,
                        "Error: I2C bus name is not unique!\n");
                    i2cbus = -4;
                    goto done;
                }
                i2cbus = adapters[i].nr;
            }
        }

        if (i2cbus == -1)
            fprintf(stderr, "Error: I2C bus name doesn't match any "
                "bus present!\n");

done:
        free_adapters(adapters);
        return i2cbus;
}

/*
 * Parse an I2CBUS command line argument and return the corresponding
 * bus number, or a negative value if the bus is invalid.
 */
int lookup_i2c_bus(const char *i2cbus_arg)
{
        unsigned long i2cbus;
        char *end;

        i2cbus = strtoul(i2cbus_arg, &end, 0);
        if (*end || !*i2cbus_arg) {
            /* Not a number, maybe a name? */
            return lookup_i2c_bus_by_name(i2cbus_arg);
        }
        if (i2cbus > 0xFFFFF) {
            fprintf(stderr, "Error: I2C bus out of range!\n");
            return -2;
        }

        return i2cbus;
}

/*
 * Parse a CHIP-ADDRESS command line argument and return the corresponding
 * chip address, or a negative value if the address is invalid.
 */
int parse_i2c_address(const char *address_arg)
{
        long address;
        char *end;

        address = strtol(address_arg, &end, 0);
        if (*end || !*address_arg) {
            fprintf(stderr, "Error: Chip address is not a number!\n");
            return -1;
        }
        if (address < 0x03 || address > 0x77) {
            fprintf(stderr, "Error: Chip address out of range "
                "(0x03-0x77)!\n");
            return -2;
        }

        return address;
}

int open_i2c_dev(int i2cbus, char *filename, size_t size, int quiet)
{
        int file;

        snprintf(filename, size, "/dev/i2c/%d", i2cbus);
        filename[size - 1] = '\0';
```

```
	file = open(filename, O_RDWR);

	if (file < 0 && (errno == ENOENT || errno == ENOTDIR)) {
		sprintf(filename, "/dev/i2c-%d", i2cbus);
		file = open(filename, O_RDWR);
	}

	if (file < 0 && !quiet) {
		if (errno == ENOENT) {
			fprintf(stderr, "Error: Could not open file "
				"`/dev/i2c-%d' or `/dev/i2c/%d': %s\n",
				i2cbus, i2cbus, strerror(ENOENT));
		} else {
			fprintf(stderr, "Error: Could not open file "
				"`%s': %s\n", filename, strerror(errno));
			if (errno == EACCES)
				fprintf(stderr, "Run as root?\n");
		}
	}

	return file;
}

int set_slave_addr(int file, int address, int force)
{
	/* With force, let the user read from/write to the registers
	   even when a driver is also running */
	if (ioctl(file, force ? I2C_SLAVE_FORCE : I2C_SLAVE, address) < 0) {
		fprintf(stderr,
			"Error: Could not set address to 0x%02x: %s\n",
			address, strerror(errno));
		return -errno;
	}

	return 0;
}

/*
	i2cbusses.h
*/

#ifndef _I2CBUSSES_H
#define _I2CBUSSES_H

#include <unistd.h>

struct i2c_adap {
	int nr;
	char *name;
	const char *funcs;
	const char *algo;
};

struct i2c_adap *gather_i2c_busses(void);
void free_adapters(struct i2c_adap *adapters);

int lookup_i2c_bus(const char *i2cbus_arg);
int parse_i2c_address(const char *address_arg);
int open_i2c_dev(int i2cbus, char *filename, size_t size, int quiet);
int set_slave_addr(int file, int address, int force);

#define MISSING_FUNC_FMT	"Error: Adapter does not have %s capability\n"

#endif

/*
	util.c - helper functions
*/

#include <stdio.h>
#include "util.h"
```

```
/* Return 1 if we should continue, 0 if we should abort */
int user_ack(int def)
{
	char s[2];
	int ret;

	if (!fgets(s, 2, stdin))
		return 0; /* Nack by default */

	switch (s[0]) {
	case 'y':
	case 'Y':
		ret = 1;
		break;
	case 'n':
	case 'N':
		ret = 0;
		break;
	default:
		ret = def;
	}

	/* Flush extra characters */
	while (s[0] != '\n') {
		int c = fgetc(stdin);
		if (c == EOF) {
			ret = 0;
			break;
		}
		s[0] = c;
	}
	return ret;
}
/*
	util - helper functions
*/

#ifndef _UTIL_H
#define _UTIL_H

extern int user_ack(int def);

#endif /* _UTIL_H */

Version.h
#define VERSION "3.1.1"
```

6.2. How to use GPIO in Linux

6.2.1. GPIO mapping table

RM-F6DU1-SMC	
Logical Number	**Physical Number**
1	32
3	34
5	36
7	38
8	39

6.2.2. GPIO sample code

```
# GPIO example 1: Output (take GPIO 32 as example)
echo 32 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio32/direction
echo 0 > /sys/class/gpio/gpio32/value
echo 1 > /sys/class/gpio/gpio32/value


# GPIO example 2: Input (take GPIO 32 as example)
echo 32 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio32/direction
cat /sys/class/gpio/gpio32/value
```

6.2.3. How to use Watch dog in Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
	int fd = open("/dev/watchdog", O_WRONLY);
	int ret = 0;
	if (fd == -1) {
		perror("watchdog");
		exit(EXIT_FAILURE);
	}
	while (1) {
		ret = write(fd, "\0", 1);
		if (ret != 1) {
			ret = -1;
			break;
		}
		puts("[WDT] Keep alive");
		sleep(50);
	}
	close(fd);
	return ret;
}
```

7. Appendix B - i.MX6 CPU ball out Table.

This Chapter is referenced for advanced SW Engineer to modify the Kernel driver or make their own BSP. Please check Freescale i.MX6 specification, porting guide and user manuals carefully

Link: http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX6X_SERIES&cof=0&am=0

Group	Circuit Net	Ball	Mixed Functions
CSPI1 [SPI]	CSPI1_SS1	G21	EIM_D19
CSPI1	CSPI1_SS2	F22	EIM_D24
CSPI1	CSPI1_MOSI	V6	KEY_ROW0
CSPI1	CSPI1_MISO	U7	KEY_COL1
CSPI1	CSPI1_CLK	W5	KEY_COL0
CSPI2 [SPI]	CSPI2_SS0	K20	EIM_RW
CSPI2	CSPI2_SS1	K22	EIM_LBA
CSPI2	CSPI2_MOSI	J23	EIM_CS1
CSPI2	CSPI2_MISO	J24	EIM_OE
CSPI2	CSPI2_CLK	H24	EIM_CS0
Battery	CHARGING#	H19	EIM_A25
Battery	BATLOW#	D25	EIM_D25
Battery	CHARGER_PRSNT#	J20	EIM_D30
Carrier Board	RESET_OUT#	P5	GPIO_19 (DU1/SO1)
Carrier Board	Carrier_STBY#	R4	GPIO_5
Carrier Board	Carrier_PWR_ON	R2	PMIC_ON_REQ(DU1/SO1)
Carrier Board	Carrier_LID#	E23	EIM_D21
Carrier Board	Carrier_SLEEP#	G22	EIM_D25
HDMI	HDMI_DDC_DAT_IN	C25	KEY_COL3 (I2C2)
HDMI	HDMI_DDC_CLK_IN	E22	KEY_ROW3 (I2C2)
HDMI	HDMI_HPD	K1	HDMI_HPD
HDMI	HDMI_CEC_IN	K2	HDMI_DDCCEC
HDMI	HDMI_CLKM	J5	HDMI_CLKM
HDMI	HDMI_CLKP	J6	HDMI_CLKP
HDMI	HDMI_D0M	K5	HDMI_D0M

HDMI	HDMI_D0P	K6	HDMI_D0P
HDMI	HDMI_D1M	J3	HDMI_D1M
HDMI	HDMI_D1P	J4	HDMI_D1P
HDMI	HDMI_D2M	K3	HDMI_D2M
HDMI	HDMI_D2P	K4	HDMI_D2P
USB_OTG	USB_OTG_OC#	H20	EIM_D21
USB_OTG	USB_OTG_ID	T4	GPIO_1
USB_OTG	USB_OTG_DN	B6	USB_OTG_DN
USB_OTG	USB_OTG_DP	A6	USB_OTG_DP
USB	USB1_DP	E1	USB1_DP
USB	USB1_DN	F10	USB1_DN
UART1	UART1_TX	M1	CSI0_DAT10
UART1	UART1_RX	M3	CSI0_DAT11
UART2	UART2_TX	E24	EIM_D26
UART2	UART2_RX	E25	EIM_D27
UART4	UART4_TX	M2	CSI0_DAT12
UART4	UART4_RX	L1	CSI0_DAT13
UART4	UART4_RTS#	L4	CSI0_DAT16
UART4	UART4_CTS#	L3	CSI0_DAT17
UART5	UART5_TX	M4	CSI0_DAT14
UART5	UART5_RX	M5	CSI0_DAT15
UART5	UART5_RTS#	M6	CSI0_DAT18
UART5	UART5_CTS#	L6	CSI0_DAT19
PMIC	PWR_BTN_SNS	J19	EIM_D29
PMIC	PMIC_INT#	P6	GPIO_18
PMIC	MX6_ONOFF	D12	ONOFF
PMIC	POR_B	C11	POR#
PMIC	PMIC_STBY_REQ	F11	PMIC_STBY_REQ
PMIC	PMIC_ON_REQ	D11	PMIC_ON_REQ
GPIO	GPIO_0	E19	SD1_DAT2
GPIO	GPIO_1	H25	EIM_A16
GPIO	GPIO_2	G24	EIM_A17
GPIO	GPIO_3	J22	EIM_A18
GPIO	GPIO_4	G25	EIM_A19
GPIO	GPIO_5	H22	EIM_A20
GPIO	GPIO_6	H23	EIM_A21

GPIO	GPIO_7	F24	EIM_A22
GPIO	GPIO_8	J21	EIM_A23
GPIO	GPIO_9	F21	EIM_D17
GPIO	GPIO_10	D24	EIM_D18
GPIO	GPIO_11	G20	EIM_D20
WDT	WDOG#	F18	SD1_DAT3
SD2	SD2_CMD	F19	SD2_CMD
SD2	SD2_CLK	C21	SD2_CLK
SD2	SD2_DATA0	A22	SD2_DATA0
SD2	SD2_DATA1	E20	SD2_DATA1
SD2	SD2_DATA2	A23	SD2_DATA2
SD2	SD2_DATA3	B22	SD2_DATA3
SD2	SD2_CD#	R6	GPIO_4
SD2	SD2_WP	T1	GPIO_2
SD2	SD2_PWR_EN	U6	KEY_ROW1
SD3	SD3_CMD	B13	SD3_CMD
SD3	SD3_CLK	D14	SD3_CLK
SD3	SD3_RST#	D15	SD3_RST
SD3	SD3_DATA0	E14	SD3_DATA0
SD3	SD3_DATA1	F14	SD3_DATA1
SD3	SD3_DATA2	A15	SD3_DATA2
SD3	SD3_DATA3	B15	SD3_DATA3
SD3	SD3_DATA4	D13	SD3_DATA4
SD3	SD3_DATA5	C13	SD3_DATA5
SD3	SD3_DATA6	E13	SD3_DATA6
SD3	SD3_DATA7	F13	SD3_DATA7
eMMC	eMMC_CLK	E16	SD4_CLK
eMMC	eMMC_CMD	B17	SD4_CMD
eMMC	eMMC_RST#	A16	NANDF_ALE
eMMC	eMMC_DATA0	D18	SD4_DAT0
eMMC	eMMC_DATA1	B19	SD4_DAT1
eMMC	eMMC_DATA2	F17	SD4_DAT2
eMMC	eMMC_DATA3	A20	SD4_DAT3
eMMC	eMMC_DATA4	E18	SD4_DAT4
eMMC	eMMC_DATA5	C19	SD4_DAT5
eMMC	eMMC_DATA6	B20	SD4_DAT6

eMMC	eMMC_DATA7	D19	SD4_DAT7
LCD	LCD_PWR_EN	C16	NANDF_CS1
LCD	LCD_BKLT_EN	A17	NANDF_CS2
LCD	LCD_BKLT_PWM	T2	GPIO_9
LCD	LCD_CLK	N19	DI0_DISP_CLK
LCD	LCD_HSYNC	N25	DI0_PIN2
LCD	LCD_VSYNC	N20	DI0_PIN3
LCD	LCD_DE	N21	DI0_PIN15
LCD	LCD_DAT0	P24	DSP0_DAT0
LCD	LCD_DAT1	P22	DSP0_DAT1
LCD	LCD_DAT2	P23	DSP0_DAT2
LCD	LCD_DAT3	P21	DSP0_DAT3
LCD	LCD_DAT4	P20	DSP0_DAT4
LCD	LCD_DAT5	R25	DSP0_DAT5
LCD	LCD_DAT6	R23	DSP0_DAT6
LCD	LCD_DAT7	R24	DSP0_DAT7
LCD	LCD_DAT8	R22	DSP0_DAT8
LCD	LCD_DAT9	T25	DSP0_DAT9
LCD	LCD_DAT10	R21	DSP0_DAT10
LCD	LCD_DAT11	T23	DSP0_DAT11
LCD	LCD_DAT12	T24	DSP0_DAT12
LCD	LCD_DAT13	R20	DSP0_DAT13
LCD	LCD_DAT14	U25	DSP0_DAT14
LCD	LCD_DAT15	T22	DSP0_DAT15
LCD	LCD_DAT16	T21	DSP0_DAT16
LCD	LCD_DAT17	U24	DSP0_DAT17
LCD	LCD_DAT18	V25	DSP0_DAT18
LCD	LCD_DAT19	U23	DSP0_DAT19
LCD	LCD_DAT20	U22	DSP0_DAT20
LCD	LCD_DAT21	T20	DSP0_DAT21
LCD	LCD_DAT22	V24	DSP0_DAT22
LCD	LCD_DAT23	W24	DSP0_DAT23
LVDS	LVDS0_TX0_N	U2	LVDS0_TX0_N
LVDS	LVDS0_TX0_P	U1	LVDS0_TX0_P
LVDS	LVDS0_TX1_N	U4	LVDS0_TX1_N
LVDS	LVDS0_TX1_P	U3	LVDS0_TX1_P
LVDS	LVDS0_TX2_N	V2	LVDS0_TX2_N
LVDS	LVDS0_TX2_P	V1	LVDS0_TX2_P

LVDS	LVDS0_TX3_N	W2	LVDS0_TX3_N
LVDS	LVDS0_TX3_P	W1	LVDS0_TX3_P
LVDS	LVDS0_CLK_N	V4	LVDS0_CLK_N
LVDS	LVDS0_CLK_P	V3	LVDS0_CLK_P
PCIe	PCIE_RST#	R1	GPIO_17
PCIe	PCIE_WAKE#	P3	CSI0_DATA_EN
PCIe	PCIE_CLK1_N	C7	CLK1_N
PCIe	PCIE_CLK1_P	D7	CLK1_P
PCIe	PCIE_RXM	B1	PCIE_RXM
PCIe	PCIE_RXP	B2	PCIE_RXP
PCIe	PCIE_TXM	A3	PCIE_TXM
PCIe	PCIE_TXP	B3	PCIE_TXP
SATA	SATA_RXP	B14	SATA_RXP (DU1)
SATA	SATA_RXN	A14	SATA_RXM (DU1)
SATA	SATA_TXN	B12	SATA_TXM (DU1)
SATA	SATA_TXP	A12	SATA_TXP (DU1)
I2C1	I2C1_SCL (I2C_PM)	N5	CSI0_DAT9
I2C1	I2C1_SDA (I2C_PM)	N6	CSI0_DAT8
I2C2	I2C2_SCL (I2C_CAM0)	U5	KEY_COL3
I2C2	I2C2_SDA (I2C_CAM0)	T7	KEY_ROW3
I2C3	I2C3_SCL (I2C_LCD)	R7	GPIO_3
I2C3	I2C3_SDA (I2C_LCD)	T3	GPIO_6
I2C4	I2C4_SCL (I2C_GP)	R3	GPIO_7 (SO1)
I2C4	I2C4_SDA (I2C_GP)	R5	GPIO_8 (SO1)
Toouch	Touch_INT#(GPIO_6)	B18	NANDF_D5
I2S	AUD3_TXC	N1	CSI0_DAT4
I2S	AUD3_TXD	P2	CSI0_DAT5
I2S	AUD3_TXFS	N4	CSI0_DAT6
I2S	AUD3_RXD	N3	CSI0_DAT7
I2S	HAD_RST# (GPIO_4)	D17	NANDF_D3
CLOCK	GPIO_0_CLKO		GPIO_0
CAN1	CAN1_TX	W6	KEY_COL2
CAN1	CAN1_RX	W4	KEY_ROW2

CAN2	CAN2_TX	T6	KEY_COL4
CAN2	CAN2_RX	V5	KEY_ROW4
SPDIF	SPDIF_IN	W23	GPIO16 (DU1/SO1)
SPDIF	SPDIF_OUT	V21	GPIO19 (DU1/SO1)
LAN	RGMII_INT	W22	ENET_RXD1
LAN	RGMII_nRST	U21	ENET_CRS_DV
LAN	RGMII_MDIO	V23	ENET_MDIO
LAN	RGMII_MDC	V20	ENET_MDC
LAN	RGMII_TXCLK	D21	RGMII_TXC
LAN	RGMII_TXD0	C22	RGMII_TD0
LAN	RGMII_TXD1	F20	RGMII_TD1
LAN	RGMII_TXD2	E21	RGMII_TD2
LAN	RGMII_TXD3	A24	RGMII_TD3
LAN	RGMII_TXEN	C23	RGMII_TX_CTL
LAN	RGMII_RXCLK	B25	RGMII_RXC
LAN	RGMII_RXD0	C24	RGMII_RD0
LAN	RGMII_RXD1	B23	RGMII_RD1
LAN	RGMII_RXD2	B24	RGMII_RD2
LAN	RGMII_RXD3	D23	RGMII_RD3
LAN	RGMII_RXDV	D22	RGMII_RX_CTL
LAN	ENET_REFCLK	V22	ENET_REF_CLK
CSI [MIPI]	CSI_CLK0M	F4	CSI_CLK0M
CSI	CSI_CLK0P	F3	CSI_CLK0P
CSI	CSI_D0M	E4	CSI_D0M
CSI	CSI_D0P	E3	CSI_D0P
CSI	CSI_D1M	D1	CSI_D1M
CSI	CSI_D1P	D2	CSI_D1P
CSI	CAM0_PWR#(GPIO_0)	A18	NANDF_D0
CSI	CAM0_RST#(GPIO_2)	C17	NANDF_D1
BOOT	BOOT_MODE0	C12	BOOT_MODE0
BOOT	BOOT_MODE1	F12	BOOT_MODE1
BOOT	TEST_MODE	E12	TEST_MODE
BOOT	TAMPER	E11	TAMPER
BOOT	BT_CFG1_0	L20	EIM_DA0
BOOT	BT_CFG1_1	J25	EIM_DA1
BOOT	BT_CFG1_2	L21	EIM_DA2

BOOT	BT_CFG1_3	K24	EIM_DA3
BOOT	BT_CFG1_4	L22	EIM_DA4
BOOT	BT_CFG1_5	L23	EIM_DA5
BOOT	BT_CFG1_6	K25	EIM_DA6
BOOT	BT_CFG1_7	L25	EIM_DA7
BOOT	BT_CFG2_0	L24	EIM_DA8
BOOT	BT_CFG2_1	M21	EIM_DA9
BOOT	BT_CFG2_2	M22	EIM_DA10
BOOT	BT_CFG2_3	M20	EIM_DA11
BOOT	BT_CFG2_4	M24	EIM_DA12
BOOT	BT_CFG2_5	M23	EIM_DA13
BOOT	BT_CFG2_6	N23	EIM_DA14
BOOT	BT_CFG2_7	N24	EIM_DA15
JTAG	JTAG_TCK	H5	JTAG_TCK
JTAG	JTAG_TMS	C3	JTAG_TMS
JTAG	JTAG_TDI	G5	JTAG_TDI
JTAG	JTAG_TDO	G6	JTAG_TDO
JTAG	JTAG_nTRST	C2	JTAG_nTRST
JTAG	JTAG_MOD	H6	JTAG_MOD

8. Appendix C : how to Flash the image to eMMC

(for advanced user only) You can flash the current SD image system (standard or customized by user) to eMMC by following method.

Use “fdisk -l” command to check current storage devices, current boot device is represented as /dev/mmcblk1, SMARC module’s eMMC device is /dev/mmcblk0

```
#sudo fdisk –l
```

Flash Module eMMC:

```
# cd flash_emmc/rp100_emmc
# ./fsl-sdcard-partition.sh –f /dev/mmcblk0
```

Remember to check the boot set up of dip switch is “boot from SMARC’s eMMC”, then, you can boot from eMMC with the above concept.

9. SMARC RM-F6xx1-SMC series difference table

	RM-F6**SO**1-SMC	RM-F6**DU**1-SMC
CPU	i.MX6 Solo	i.MX6 Dual
DDR3	1GB	1GB
eMMC(SD3)	4GB	4GB
LAN PHY	x 1	x 1
CSI	x 1	x 1
LVDS	x 1	x 1
HDMI	x 1	x 1
24bit LCD	x 1	x 1
USB-OTG	x 1	x 1
USB	x 2	x 2
SD2	x 1	x 1
SD4	x 1	x 1
CAN	x 2	x 2
UART	x 4	x 4
mPCIe	x 1	x 1
SPI	x 2	x 2
SPDIF	x 1	x 1
I2S	x 1	x 1
WDT	x 1	x 1
SATA	x 0	x 1
I2C 1	for audio codec	for audio codec
I2C 2	for 4 wires touch and MIPI camera	for 4 wires touch and MIPI camera
I2C 3	for pin header.	for pin header.
I2C 4_PM	for mPCIe	NA
GPIO0	for CAN PWR	for CAN PWR
GPIO1		
GPIO2	for CAN RST	for CAN RST
GPIO3		
GPIO4	for HAD_RST	for HAD_RST
GPIO5		
GPIO6	Touch INT	Touch INT

GPIO7		
GPIO8		
GPIO9	HP_DET_R	HP_DET_R
GPIO10	MIC_DET_RST	MIC_DET_RST
GPIO11	SW4	SW4
	(internal)	(internal)

10. Appendix D –Useful links

For more information about Android, please visit:

http://developer.android.com/index.html

For more information Freescale i.MX6 CPU , please visit:

http://www.freescale.com/webapp/sps/site/homepage.jsp?code=IMX_HOME