IB113

Wide Temperature ARM-based SBC with

Freescale i.MX6 Cortex-A9 Dual Core 800MHz SoC

User's Manual

2014 DEC. V1.0b

This page is intentionally left blank.

Quick Start Guide

Here is a step-by-step guide to boot up the IB113:

a. Based on the order, the (Android) or (Linux OS) will be preloaded in the IB113's eMMC. System can be booted up with HDMI (by default) and power input ready.
b. To use the root/ serial port debug function, please check Chapter 4.2.1 (COM1 debug cable setup) information.
c. To boot up with a different LVDS panel, please refer to Chapter 4.2.2.
d. To make a recovery SD card (for IB113 advanced user only), please refer to Chapter 4.

Note: different LVDS panels have different customization; please check with your sales contact.

e. For advanced users who are building their own products, please refer to Chapter 5.
f. For special HW/SW/ panel customization requests or assistance, please check with Ibase sales dept.

TABLE OF CONTENTS

Acknowledgments

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

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

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. IB113 Introduction

IB113 is a 3.5" Disk-Size SBC w/ ARM Base Freescale i.MX6 Cortex-A9 800Mhz CPU. IB113 fulfill industrial ambient operating temperature from -40 ℃ to +85℃. The device offers 3D graphics acceleration, while also supporting numerous peripherals, including RS232/422/485, CAN, USB, USB OTG, 1st/2nd LAN ports, SATA, R/C touch interfaces, that are well suited for industrial applications. All components are selected from industrial grade parts for wide-temperature environment operation.

1.2. IB113 Hardware Specifications

- Freescale ARM Cortex-A9, 800MHz, automotive-grade processor
- Supports -40°C~ 85°C environment, all industrial-grade components
- Supports DC 12V~24V input voltage
- 1GB DDR3, 4GB eMMC on board
- Supports 4/5-wired resistive touch, and I2C/USB header for capacitive touch
- Supports OpenGL ES 2.0 and OpenVG 1.1 hardware accelerators
- Supports COM, CAN,USB-OTG, SD card, HDMI, Dual Channel LVDS
- Supports rich I/O internface for BOM customization

Form Factor	3.5" Disk-Size SBC, 102mm x 147mm
CPU	Freescale Cortex™-A9 i.MX6 Dual 800MHz, automotive grade CPU
System Memory	1GB DDR3 on board
Data Memory	4GB eMMC on board SD socket (up to 32GB)
Display	HDMI 18/24 bit dual channel LVDS (up to 1920 x 1080)
Video Codec	Decode: 1080p, 30fps Encode: 1080p, 30fps
LAN	1st LAN : 10/100/1000 Base-T Ethernet (Gb LAN) 2nd LAN : 10/100 Base-T Ethernet
Touch	PM6000 resistive touch IC (4/5-wires) on board USB/ I2C header on board (for capacitive touch)
Audio	Audio pin header (1x microphone, 1x speaker)
RTC	Seiko RTC IC on board
Watchdog	256 Levels
Edge I/O	1x 1st 10/100/1000 LAN 1x 2nd 10/100 LAN 1x USB 2.0 Host (Type-A) 1x USB OTG (mini-USB Type-B) 1x COM RS-232/422/485 1x SD socket (up to 32GB) 1x DC-in jack 1x HDMI (Type-A) 1x Reset button

Headers & Expansion Slots	Headers: 1x GPIOx8 pin header 1x Debug port pin header 1x USB2.0 Host box header 1x I2C pin header 1x Resistive touch header (4/5-wires) 2x CAN Bus 2.0B (w/ isolation) pin headers 1x Dual channel LVDS box header (supports full HD) 1x Audio pin header (1x microphone, 1x speaker) 1x SATA header Slots: 1x Full size Mini PCIe with USB interface
Power	12V~24V DC-in
Operating Temperature	-40°C~ 85°C (-40°F ~ 185°F)
Devices	WiFi / GPS / 3G module (option)
Software Support	Ubuntu Linux 12.04(kernel 3.0.35)/ Android 4.3
Relative Humidity	10%~90% (non-condensing)

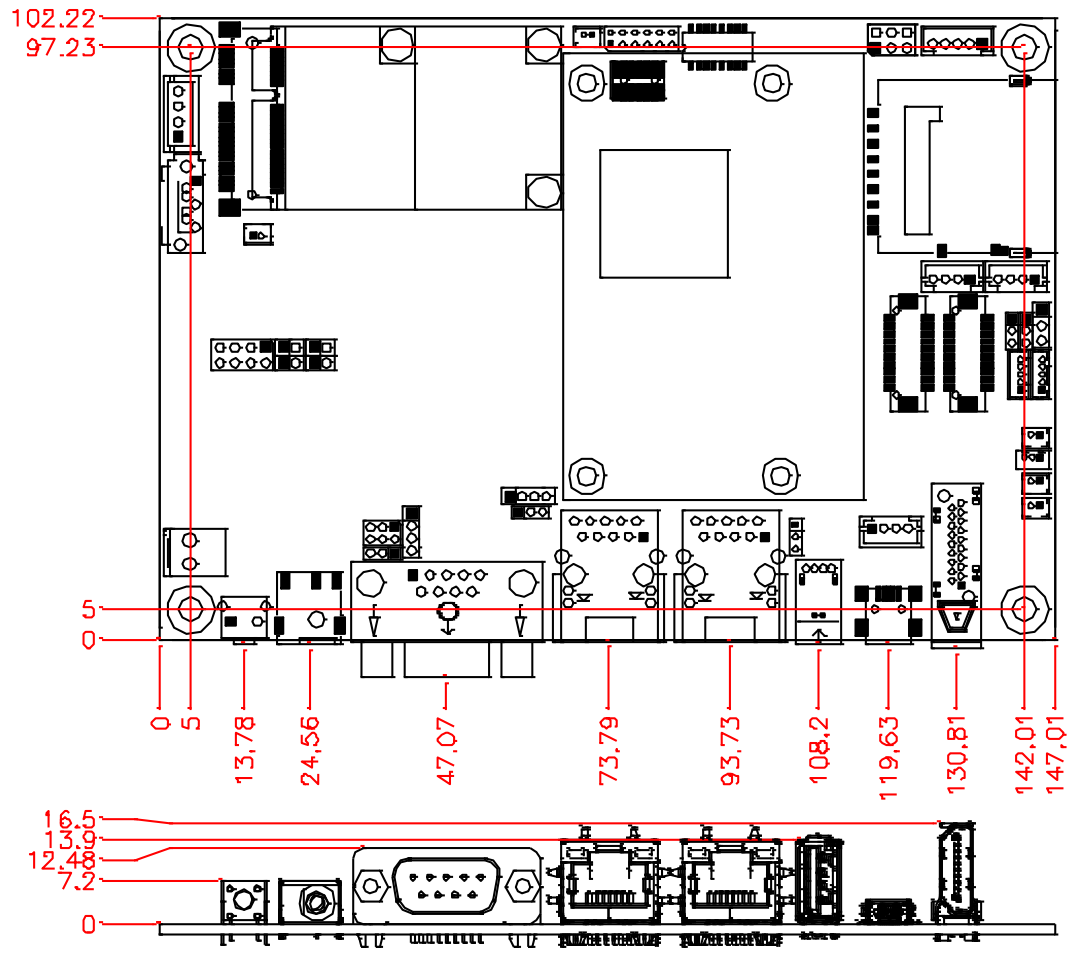
Ordering Information

IB113	3.5-inch SBC w/ Industrial-Grade Freescale i.MX6 dual core (800Mhz), 1GB DDR3, 2x LAN , SD, 1x COM ports, Mini PCI-E(x1) slot, HDMI, 8x GPIO, 2x CAN, 2nd LAN, 4GB eMMC, Resistive touch panel (4/5-wires) support
Accessary (Optional)	RF: - 3G+GPS Combo (Mini PCI-E card) - WIFI+BT Combo (Mini PCI-E card) - WIFI module (USB I/F) - BT module (USB/IF) Display: - 800 x 600, resistive touch panel (5-wires) - LVDS cable: LCD314 Debug cable: - PK1-100A (Please contact with iBASE sales)
HSIB113-BGA-A	Heat-sink for IB113

• This specification is subject to change without prior notice.

I/O View

Board Dimensions



1.3. Optional Items

If you have any optional item request, please contact Ibase sales dept.

Item	Specifications	Part Number	Remarks
Speaker	4 OHM 1.5W 10CM	A057SPP3516K11000P	
Power Supply	60W 12V	A005PS060WFSP0101P	
USB OTG (mini type) Cable	USB- 81 2- HEAD 4C 120CM	C501USB8105A12000P	
DSUB- USB- 4 Cable	TEST- 220	C501TES2200202000P	For internal USB header
Debug Port Cable	PK1-100A	C501PK11003102A00P	For internal debug port.

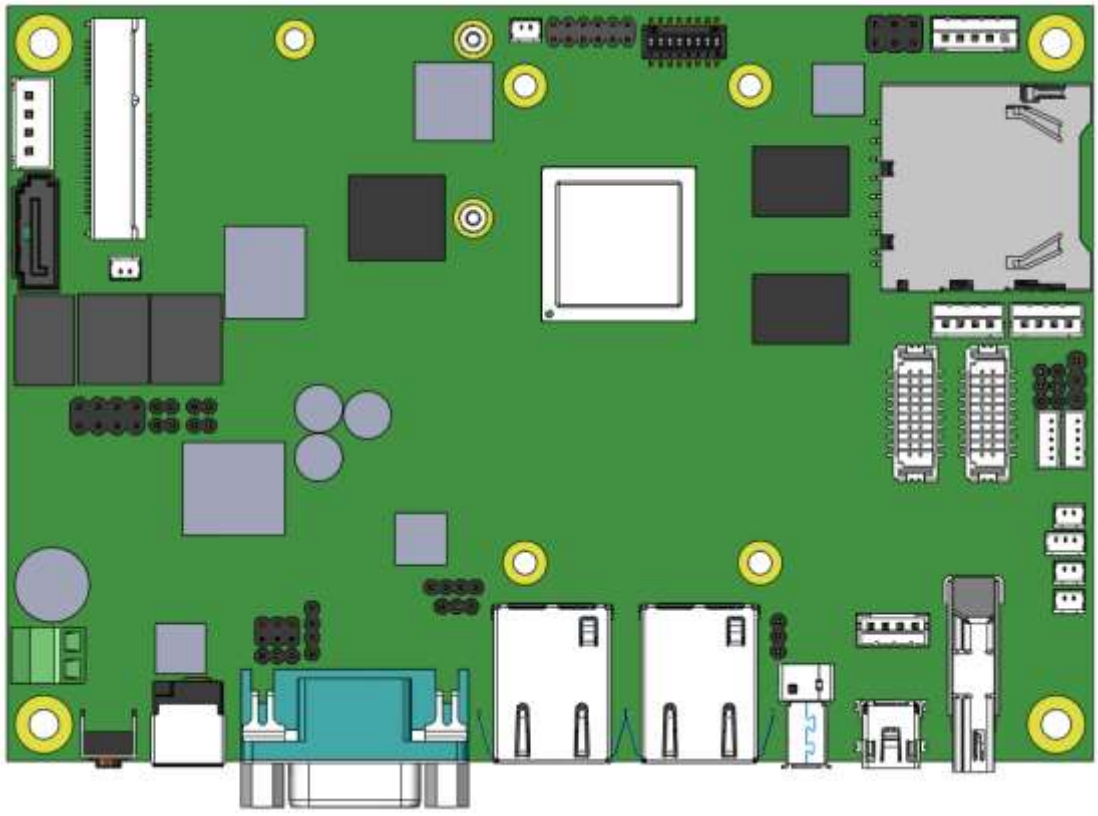
2. Jumper setting on IB113

[Important] Please check the jumpers, DIP, buttons and switches on IB113 before doing the panel connection and boot up.

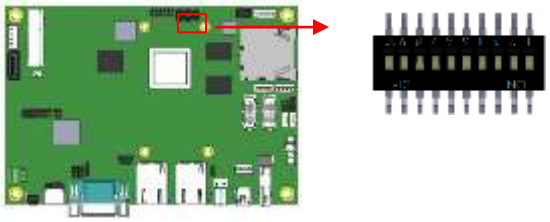
Jumpers are used on IB113 to select various settings and features according to your needs and applications. Contact your supplier if you have doubts about the best configuration for your needs. The following lists the connectors on IB113 and their respective functions.

Jumper Locations on IB113

Top Side

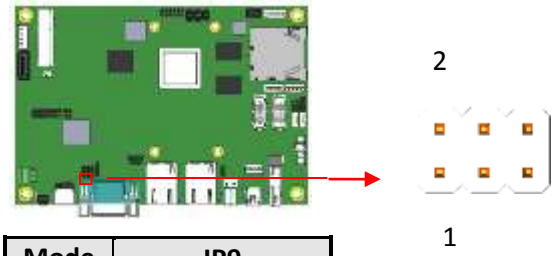


SW1: System Boot Configuration (factory use only)



SW1	Boot From
10101010	SD
01100110	EMMC

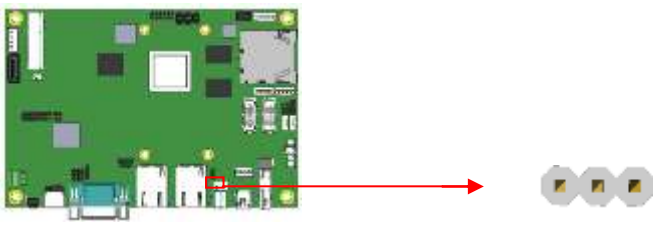
JP9: COM2 RS232, RS422, RS485 Selection



Mode	JP9
RS232	2-4 Short 3-5 Short
RS422	3-5 Short 4-6 Short
RS485	1-3 Short 4-6 Short

Default setting is RS232 mode. JP9 setting for COM2.

JP6: USB +3.3V/+5V Power Setting

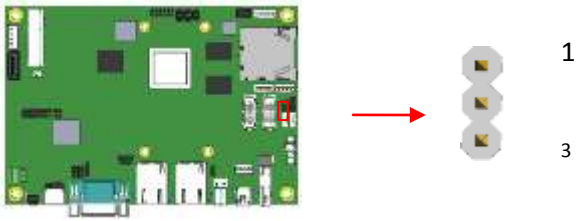


JP6	Setting
	+3.3V
	+5V

Default setting is +5V.

JP6 setting for J9.

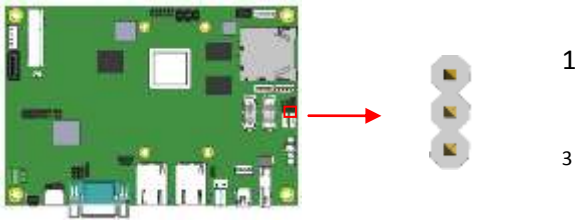
JP4: LVDS +3.3V/+5V Power Setting



JP4	Setting
	+3.3V
	+5V

Default setting is +3.3V. ; JP4 setting for CH1,CH2.

JP3: LED Brightness +5V/+12V Power Setting

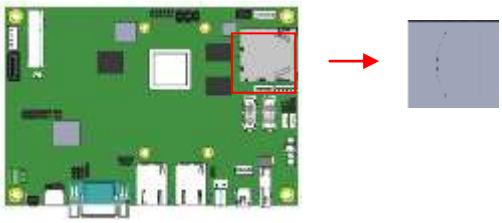


JP3	Setting
	+5V
	+12V

efault setting is +5V.

JP3 setting for CH1,CH2

CN1: SD Card Connector

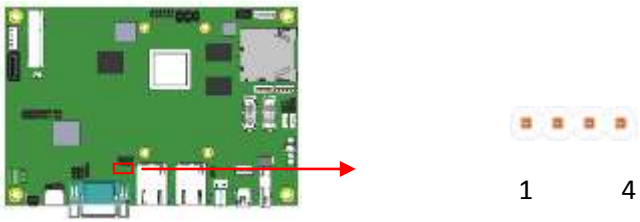


COM1 PORT: COM1 RS232 Connector

(Debug Port, factory use only)

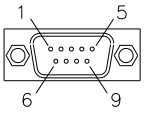
Part Number: 0195-01-200-040

Description: Pin Header 2.0*2.0mm S/T Single Row 4pin



Pin #	Signal Name
1	COM1 RX, Receive data
2	COM1 TX, Transmit data
3	GND, ground
4	NC

COM2 : RS232/RS422/RS485 Serial Port



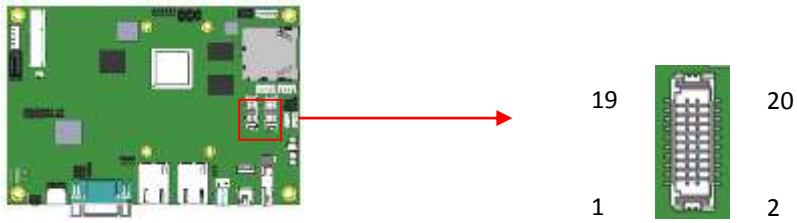
Pin #	Signal Name		
	RS-232	RS-422	RS-485
1	DCD	TX-	DATA-
2	RX	TX+	DATA+
3	TX	RX+	NC
4	DTR	RX-	NC
5	Ground	Ground	Ground
6	DSR	NC	NC
7	RTS	NC	NC
8	CTS	NC	NC
9	NC	NC	NC

Note: Please refer to JP9 setting for RS232, RS422 and RS485 mode selection.

CH1,CH2: LVDS Display Connector

Part Number: DF13-20DP-1.25V(95)

Description: P1.25 SMD 20PIN Male 180D 2R



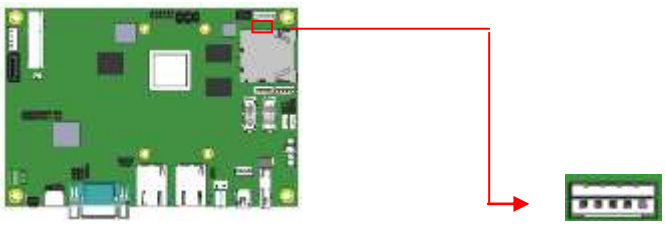
Signal Name	Pin #	Pin #	Signal Name
LCD1_TX0_P	1	2	LCD1_TX0_N
Ground	3	4	Ground
LCD1_TX1_P	5	6	LCD1_TX1_N
Ground	7	8	LCD_VDD
LCD1_TX3_P	9	10	LCD1_TX3_N
LCD1_TX2_P	11	12	LCD1_TX2_N
Ground	13	14	Ground
LCD1_CLK_P	15	16	LCD1_CLK_N

BTL_PWM	17	18	LCD_VDD
BKLT_VCC	19	20	BKLT_VCC

J2: Resistive Touch Panel Connector

Part Number: 0110-161-050

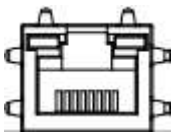
Description: MINI BASE;DIP 180D MINI 5PIN



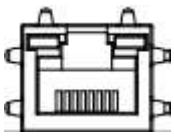
5 1

Pin #	Signal Name
1	JTP_LR
2	JTP_LL
3	JTP_WIP
4	JTP_UR
5	JTP_UL

CN7: 100/1Gb LAN1 (From Freescale i.MX6)



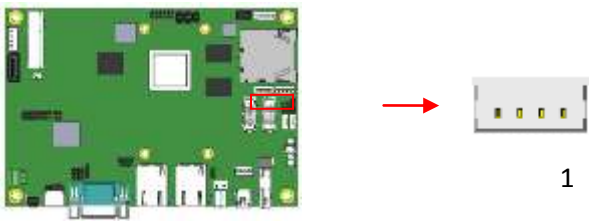
CN8: 10/100Mb LAN2 (USB to Ethernet)



CN3,4: LED Backlight Control Connector

Part Number: 0110-2610040

Description: **JST-PH** Type Wafer 2.0mm 4Pin

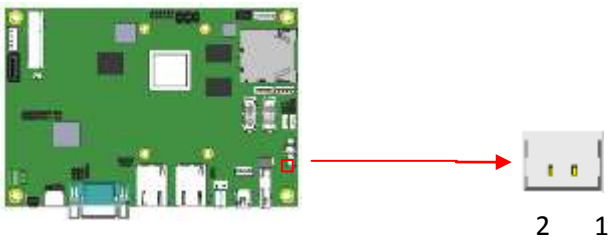


Pin #	Signal Name
1	BKLT_VCC
2	LCD_BKLT_EN
3	LCD_BKLT_PWM
4	GND

J7: Speaker Right Out Connector

Part Number: 0110-2610020

Description: Molex 53047 1.25mm Wafer S/T Type 2pin



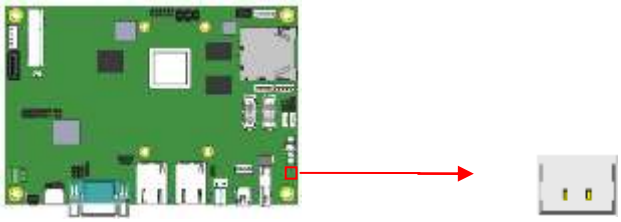
Pin #	Signal Name
1	SPEAKER_RIGHT-
2	SPEAKER_RIGHT+

Note: The maximun output power is 2 W with 4 Ω speaker
or 1.4 W with 8 Ω speaker

J8: Speaker Left Out Connector

Part Number: 0110-2610020

Description: Molex 53047 1.25mm Wafer S/T Type 2pin



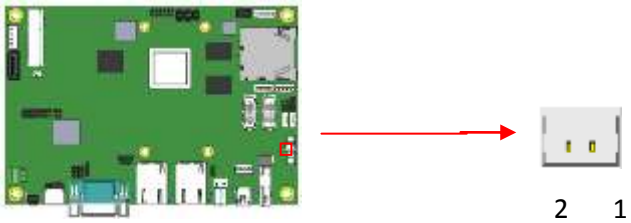
Pin #	Signal Name
1	SPEAKER_LEFT-
2	SPEAKER_LEFT+

Note: The maximun output power is 2 W with 4 Ω speaker or 1.4 W with 8 Ω speaker

JMIC1: Microphone Connector

Part Number: 0110-2610020

Description: Molex 53047 1.25mm Wafer S/T Type 2pin

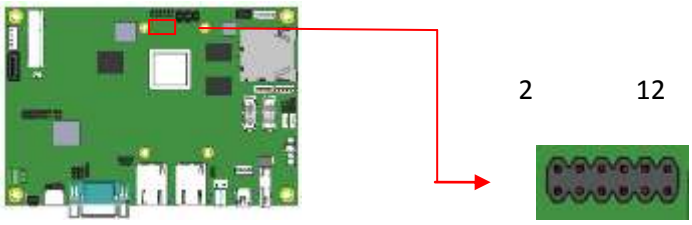


Pin #	Signal Name
1	GND
2	MIC_IN

J1: Digital I/O 4 In/4 Out Connector

Part Number: 0196-01-200-120

Description: MALE HD;DIP MINI 180D 12PIN 2R



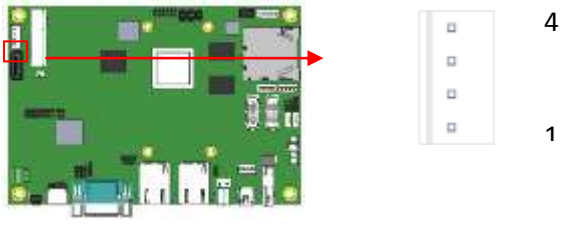
Signal Name	Pin #	Pin #	Signal Name
+3.3V	1	2	GND
GPIO_0	3	4	GPIO_1
GPIO_2	5	6	GPIO_3
GPIO_4	7	8	GPIO_5
GPIO_6	9	10	GPIO_7
GPIO_8	11	12	GPIO_9

Note: All In/Out signals level are 3.3V .

J3: SATA Power

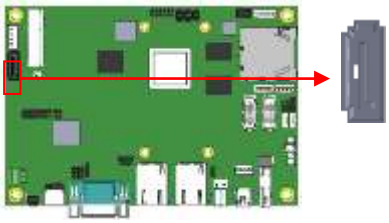
Part Number: WAFER25-104S-2442-ST

Description: 2.5 wafer 4pin 180D



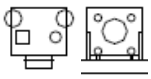
Pin #	Signal Name
1	+5V
2	GND
3	GND
4	12V

CN2: SATA Bus



Pin #	Signal Name
1	GND
2	SATA_TXP
3	SATA_TXN
4	GND
5	SATA_RXN
6	SATA_RXP
7	GND

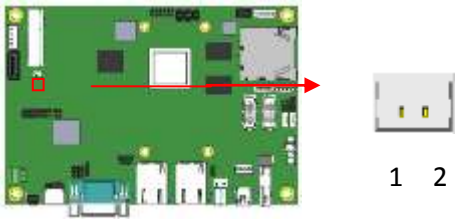
SW2: Push Button for Hardware Reset



BAT: 3.0V Lithium Battery Connector

Part Number: 0110-2610020

Description: Molex 53047 1.25mm Wafer S/T Type 2pin

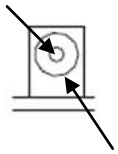


Pin #	Signal Name
1	+VCC
2	GND

CN9: 12V~24V Power Connector

This connector supplies the system board operating voltage.

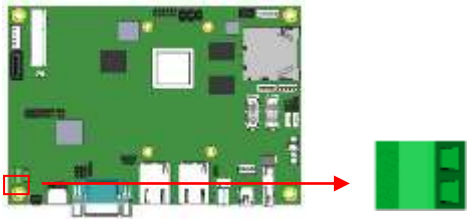
Pin 1



Pin 2

Pin #	Signal Name
1	+12V ~ +24V
2	GND

J10: 12V~24V Power Connector



1

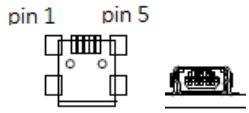
Pin #	Signal Name
1	+12V ~ +24V
2	GND

Note: J10 signals are same as CN9.

JUSB1: USB2.0 Type A Connector

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

CN10: Mini USB OTG Connector



Pin #	Signal Name
1	+5V
2	D-
3	D+
4	ID
5	GND

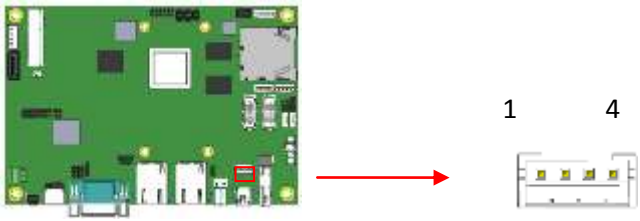
Note: CN10 used as USB device while ID is floating.

(CN16 support USB device only.)

J9: USB2.0 Connector

Part Number: B4B-PH-K-S(LF)(SN)

Description: Mini Base;DIP S 2mm 4P



Pin #	Signal Name
1	+5V / +3.3V
2	D-
3	D+
4	GND

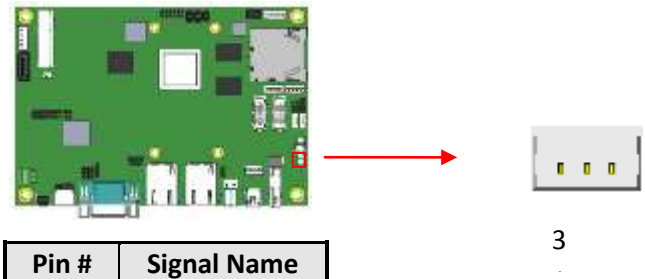
Default setting is +5V.

Please refer to JP6 setting for +5V and +3.3V selection.

J6: Line Out Connector

Part Number: 0110-2610030

Description: Molex 53047 1.25mm Wafer S/T Type 3pin

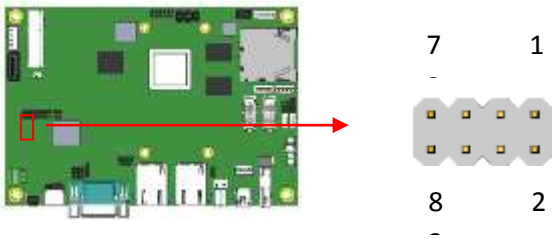


Pin #	Signal Name
1	LINE_OUTL
2	GND
3	LINE_OUTR

J5: CANBUS Connector

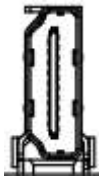
Part Number: 0126-01-203-080

Description: 2.54*2.54mm S/T Dual Rows 2*4pin



Signal Name	Pin #	Pin #	Signal Name
GND_ISO	1	2	GND_ISO
CAN1_H	3	4	CAN2_H
CAN1_L	5	6	CAN2_L
GND_ISO	7	8	GND_ISO

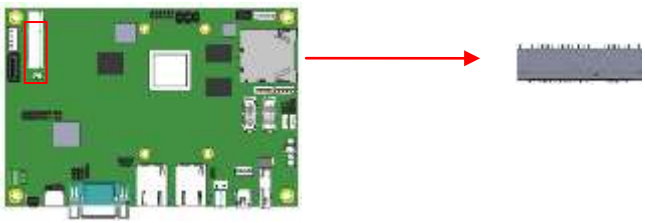
J11: HDMI connector



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

* It is strongly suggested to insert HDMI cable and HDMI panel before system power on.

J4: Mini PCIE Connector



Signal Name	Pin #	Pin #	Signal Name
NC	1	2	+3.3V
NC	3	4	GND
NC	5	6	NC
NC	7	8	NC
GND	9	10	NC
PCIE_CLK1_N	11	12	NC
PCIE_CLK1_P	13	14	NC
GND	15	16	NC
NC	17	18	GND
NC	19	20	+3.3V
GND	21	22	RESET#
PCIE_RXM	23	24	+3.3V
PCIE_RXP	25	26	GND
GND	27	28	NC
GND	29	30	I2C2_SCL
PCIE_TXM	31	32	I2C2_SDA
PCIE_TXP	33	34	GND
GND	35	36	USB2.0 D-
GND	37	38	USB2.0 D+
+3.3V	39	40	GND
+3.3V	41	42	NC
GND	43	44	NC
NC	45	46	NC
NC	47	48	NC
NC	49	50	GND
NC	51	52	+3.3V

3. Software Setup

Basically, the IB113 is preloaded O.S (Android / Linux) into eMMC by default. Connect the **HDMI** with IB113, and 9~36V power directly.

3.1. Make a Recovery SD Card (for advanced user only)

For advanced user who has Ibase standard image file, refer to this chapter to prepare the recovery boot-up SD card. Ibase optionally provides 8" LVDS panel for users to prepare the software application pre-development easily under Linux / Android platform.

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

Note: all data in the eMMC will be erased.

-- for IB113

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)

Image path: (image path may change / update)

/bsp/RISC_IMAGE/IB113/IB113/Linux/IB113-Linux_install_3.0.35-xxxxxx.7z

(xxxxxx is release date, ex: IB113-Linux_install_3.0.35-141104.7z)

/bsp/RISC_IMAGE/IB113/IB113/Android/IB113-Android_install_4.3-xxxxxx.7z

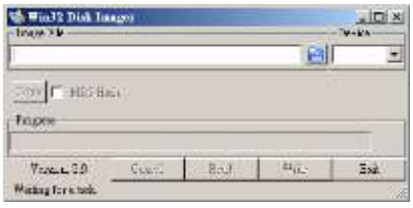
(xxxxxx is release date, ex: IB113-Android_install_4.3-141104.7z)

(based on Freescale BSP: L3.0.35-4.1.0)

For advanced users who want to return to the factory reset status, the instructions below will guide you through installing a recovery program on your SD card to allow you to easily install the default OS's and to recover your card when needed.

1. Insert an SD card that is 8GB 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

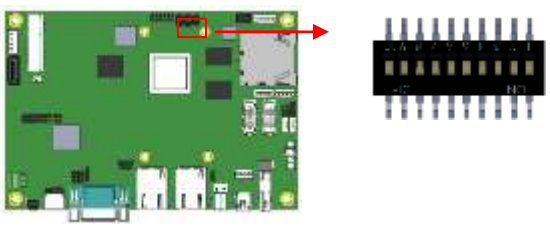
   (Descripted in previous page)
4. Download the Win32DiskImager from **http://sourceforge.net/projects/win32diskimager/ and use it to restore the target operating system.**



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

6. Please check the boot device switch and make sure it can boot from SD Card by

SW1: System Boot Configuration (factory use only)



SW1	Boot From
10101010	SD
01100110	EMMC

--- Boot Up with IB113---

Please double check the Boot device selection before powering on.

(IB113, by default, is set to boot up from eMMC.)

1. Insert the SD card/MicroSD into the motherboard. Make sure the HDMI 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 PCB will be formatted and the OS will be installed while the progress bar shows 100% complete.
3. Remove the power and the recovery SD. Remember to change the boot up device to EMMC by SW1.
4. **Connect the power** and boot up the IB113, you will see the Linux/ Android boot up pages.

3.2. Parameter Setting on U-boot

IB113 supports HDMI output by default. If you have any other LVDS panel to be customized, please contact Ibase sales or FAE staff.

3.2.1. Preparation (debug console)

i. The COM1 (Tx1, Rx1) is the default debug port. Check that it can be connected to (RX, Tx) in your PC environment.
ii. Use 115200 bps (8n1, no flow control) in Windows terminal (for example Putty.exe)
iii. During system boot up, you can ***press "Enter" t***o stop auto boot and modify your environment.

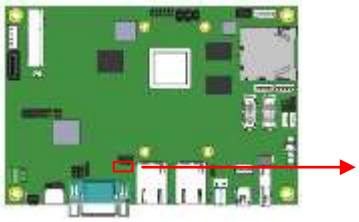
(Note: For users who are not sure about the COM connection, please check if Board.COM1.Tx1 is connected to PC.COM.Rx ; Board.COM1.Rx1 to PC.COM.Tx)

COM1 PORT: COM1 RS232 Connector

(Debug Port, factory use only)

Part Number: 0195-01-200-040

Description: Pin Header 2.0*2.0mm S/T Single Row 4pin



1 4

Pin #	Signal Name
1	COM1 RX, Receive data
2	COM1 TX, Transmit data
3	GND, ground
4	NC

3.2.2. Display setting command For Linux and Android

With the debug port, follow the reference command examples to help you to be familiar with display modification. (for advanced software engineers only)

Command to set HDMI output (default is auto-detect):

```
run sethdmi
```

Command to assign resolution set HDMI output:

```
setenv xres 1400
setenv yres 900
run sethdmi2
```

Command to set LVDS output to 1024x768, 18bit:

```
setenv xres 1024
setenv yres 768
run setlvds18
run setlvds
```

Command to set LVDS output to 1024x768, 24bit:

```
setenv xres 1024
setenv yres 768
run setlvds24
run setlvds
```

Command to set LVDS output to 1920x1080 with dual-link LVDS, 24bit:

```
setenv xres 1920
setenv yres 1080
run setlvds24
run setlvds2
```

Command to set LVDS0, LVDS1 and HDMI output, LVDS is 1024x768, 18bit:

```
setenv xres 1024
setenv yres 768
run setlvds18
run dul1
```

Dual display (same frame) output mode 1. (only work in Android).

Command to set LVDS0 and LVDS1 (same frame), HDMI output with auto-detect, LVDS is 1024x768, 18bit:

```
setenv xres 1024
setenv yres 768
run setlvds18
run dul1
```

Dual display (same frame) output mode 2, (only work in Android).

Command to set dual-link LVDS, HDMI output with auto-detect, LVDS is 1920x1080, 24bit:

```
setenv xres 1920
setenv yres 1080
run setlvds24
run dul2
```

*Note:

- If the resolution of LVDS panels is =< 1366x768, both LVDS0 & LVDS1 will get the same frame from GPU (clone mode) by default.

- If the user wants to try extended desktop mode output in LVDS0+LVDS1 panels under Linux, please change the parameter in u-boot:

```
setenv video ‘setenv vdoset video=mxcfb1:off video=mxcfb2:off video=mxcfb3:off video=mxcfb0:dev=ldb,${xres}x${yres}M@60,if=${lclr},bpp=32 ldb=sin0’
```

- If the resolution of LVDS panel is > 1366 x 768, Dual-Link LVDS mode will be activated automatically by default. (ex.1920x1080),

- For any other configurations, please contact iBASE sales, or FAE support.

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

This Chapter is an example only, and it is mainly for advanced SW engineers to build the image for IBASE ARM PCB. Any other modification, new device or driver should be handled carefully.

4.1. Building BSP Source

4.1.1. Preparation

Suggested Host Platform: Ubuntu 10.04, 12.04, 14.04, x86 and 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. **

4.1.2. Installing Toolchain

** Before download the toolchain and BSP code, please install xz-utils to unpackage. **

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

assume your toolchain file is located at root home dir,
"*gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tar.xz" file in "Downloads" folder*:

** read words is your command **

sudo su

cd ~

mkdir -p ~/IB113_BSP ~/IB113_BSP/linux

cd ~/IB113_BSP/linux

tar Jxf gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tar.xz

```
zetta@u1004x64:~$ sudo su
[sudo] password for zetta:
root@u1004x64:/home/zetta# mkdir –p ~/IB113_BSP/ ~/IB113_BSP/linux
root@u1004x64:/home/zetta# tar Jxf Downloads/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tar.xz -C
~/IB113_BSP/linux/
root@u1004x64:/home/zetta# cd ~
root@u1004x64:~# ls –la IB113_BSP/linux/
lrwxrwxrwx  1 root  root            65 11 月  3 11:19 fsl-linaro-toolchain ->
gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain
drwxr-xr-x  3 root  root          4096 11 月  3 11:19 gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12
```

4.1.3. Building u-boot

u-boot source file is “IB113bsp_linux_uboot_src_XXXXXX.tar.xz”, XXXXXX is release date, assume u-boot source file is in /root/Downloads/ folder:

cd ~/IB113_BSP/linux/

tar Jxf ~/Downloads/IB113bsp_linux_uboot_src_141105.tar.xz

cd uboot-imx/

./mklnx-uboot.sh

```
root@u1004x64:~# cd ~/IB113_BSP/linux
root@u1004x64:~/IB113_BSP/linux# tar Jxf ~/Downloads/IB113bsp_linux_uboot_src_141105.tar.xz
root@u1004x64:~/IB113_BSP/linux# cd uboot-imx
root@u1004x64:~/IB113_BSP/linux/uboot-imx# ./mklnx-uboot.sh
….
/root/IB113_BSP/linux/uboot-imx/../fsl-linaro-toolchain/bin/arm-fsl-linux-gnueabi-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
image size 198KiB
198+0 records in
198+0 records out
202752 bytes (203 kB) copied, 0.000471782 s, 430 MB/s
197+1 records in
197+1 records out
202448 bytes (202 kB) copied, 0.00698078 s, 29.0 MB/s
root@u1004x64:~/IB113_BSP/linux/uboot-imx#
```

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

4.1.4. Building kernel

kernel source file is “IB113bsp_kernel_uboot_src_XXXXXX.tar.xz”, XXXXXX is release date, assume kernel source file is in /root/Downloads/ folder:

cd ~/IB113_BSP/linux/

tar Jxf ~/Downloads/IB113bsp_linux_kernel_src_141105.tar.xz

cd linux-3.0.35/

./mx.sh ib113_defconfig;./mx.sh

./mx.sh modules_install

```
root@u1004x64:~# cd ~/IB113_BSP/linux
root@u1004x64:~/IB113_BSP/linux# tar Jxf ~/Downloads/IB113bsp_linux_kernel_src_141105.tar.xz
root@u1004x64:~/IB113_BSP/linux# cd linux-3.0.35
root@u1004x64:~/IB113_BSP/linux/linux-3.0.35#./mx.sh ib113_defconfig;./mx.sh
….
image size 7MiB
7+0 records in
7+0 records out
7340032 bytes (7.3 MB) copied, 0.00890854 s, 824 MB/s
13028+0 records in
13028+0 records out
6670336 bytes (6.7 MB) copied, 0.160881 s, 41.5 MB/s
root@u1004x64:~/IB113_BSP/linux/linux-3.0.35# ./mx.sh modules_install
…
   DEPMOD    3.0.35
Warning: you may need to install module-init-tools
See http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt
…
root@u1004x64:~/IB113_BSP/linux/linux-3.0.35#
```

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

** kernel module files are installed at ~/IB113_BSP/linux/rootfs/ **

4.1.5. Build RAMdisk image (option.)

** RAMdisk image depend on need, not necessary **

After kernel module files install into ~/IB113_BSP/linux/rootfs/, copy kernel modules define, and kernel module files into ~/IB113_BSP/linux/ramfs/, file struct same as ~/IB113_BSP/linux/rootfs/

Example:

```
root@u1004x64:~/IB113_BSP/linux/ramfs# tree lib/modules/3.0.35/
lib/modules/3.0.35/
├── build -> /root/EL_IB112/ib113-linux/linux-3.0.35
├── kernel
│   ├── drivers
│   │   ├── ata
│   │   │   ├── ahci.ko
│   │   │   └── ata_generic.ko
```

Todo packaged RAMdisk files into RAMdisk cpio image "~/IB113_BSP/linux/IMG/ramfs.bin":

./img.sh

```
root@u1004x64:~/IB113_BSP/linux# ./img.sh
14344 blocks
root@u1004x64:~/IB113_BSP/linux#
```

Todo packaged RAMdisk ans zImage images into kernel image for u-boot using "~/IB113_BSP/linux/IMG/113linux.img":

./fimg.sh

```
root@u1004x64:~/IB113_BSP/linux# ./fimg.sh
image size 7MiB
7+0 records in
7+0 records out
7340032 bytes (7.3 MB) copied, 0.00492305 s, 1.5 GB/s
13028+0 records in
13028+0 records out
6670336 bytes (6.7 MB) copied, 0.128988 s, 51.7 MB/s
root@u1004x64:~/IB113_BSP/linux#
```

4.1.6. Install Linux to SD card

i. Insert an empty SD card with at least 8GB size and put it in a card reader connecting to your host PC. Assume your SD card is /dev/sdf on your x86 host PC

ii. This installer script will destroy every data in your SD card, please make sure file already backup that you needed.

iii. Linux rootfs archive file is “IB113bsp_rootfs_linux_XXXXXX_installer.tar.xz”, XXXXXX is release date. Assume this file is in /root/Downloads/

SDcard map

Sector (512byte) start-end	comment
0-1	Partition table
2-2047	u-boot image
2048-18431	Reversed.
18433-34816	Kernel and RAMdisk image
34818-end	Linux root files system

tar Jxf ~/Downloads/IB113bsp_rootfs_linux_141105.tar.xz –C ~/IB113_BSP/linux/

```
root@u1004x64:~/IB113_BSP/linux/# tar Jxf ~/Downloads/IB113bsp_rootfs_linux_141105.tar.xz –C ~/IB113_BSP/linux/
root@u1004x64:~/IB113_BSP/linux/# cd IMG
root@u1004x64:~/IB113_BSP/linux/IMG/# ./113linux.sh /dev/sdf
…
iBASElinux: 155204/163840 files (0.1% non-contiguous), 536524/655360 blocks
resize2fs 1.42.9 (4-Feb-2014)
Resizing the filesystem on /dev/sdf2 to 1044223 (4k) blocks.
The filesystem on /dev/sdf2 is now 1044223 blocks long.
Completed.
```

** any message from e2fsck are generate by resize, please ignore it**

4.1.7. Booting with your SD card

(For advance software users only)

Put SD card in your board and insert special COM port dongle 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 HDMI monitor. Power on and you will see u-boot prompt.

If you want change to other monitor (or display panel) type, please reference section 3.2.2 to setting video output command for want.

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

You can see Ubuntu Linux is running on monitor.

5. Reference Code

5.1. 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"
```

5.2. How to use GPIO in Linux

5.2.1. GPIO Mapping Table

GPIO	
Logical Number	**Physical Number**
0	**40**
1	**41**
2	**42**
3	**43**
4	**44**
5	**45**
6	**46**
7	**47**
8	**201**
9	**202**

5.2.2. GPIO Sample Code

```
# GPIO example 1: Output (take GPIO 40 as example)
echo 32 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio40/direction
echo 0 > /sys/class/gpio/gpio40/value
echo 1 > /sys/class/gpio/gpio40/value


# GPIO example 2: Input (take GPIO 40 as example)
echo 32 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio40/direction
cat /sys/class/gpio/gpio40/value
```

5.3. How to use Watchdog 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;
}
```

Note: hexdump command “524288” come from “dd bs=1024 seek=512”, 1024*512=524288.

5.4. eMMC test

Note! This operation may damage the data stored in eMMC flash. Please make sure there is no critical data in the eMMC flash being used for this test

5.4.1. Erase and check

#dd if=/dev/zero of=/dev/mmcblk0 bs=1024 count=1 seek=512

1+0 records in

1+0 records out

#hexdump -C /dev/mmcblk0 -s 524288 -n 16

01887800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

5.4.2. Write and check

#echo -n "0123456789ABCDEF" | dd of=/dev/mmcblk0 bs=1024 count=1 seek=512

0+1 records in

0+1 records out

#hexdump -C /dev/mmcblk0 -s 524288 -n 16

01887800 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 |0123456789ABCDEF|

5.5. SATA test

Note! This operation may damage the data stored in SATA disk. Please make sure there is no critical data in the SATA disk being used for this test

5.5.1. Erase and check

#dd if=/dev/zero of=/dev/sda bs=1024 count=1 seek=512

1+0 records in

1+0 records out

#hexdump -C /dev/sda -s 524288 -n 16

01887800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

5.5.2. Write and check

#echo -n "0123456789ABCDEF" | dd of=/dev/sda bs=1024 count=1 seek=512

0+1 records in

0+1 records out

#hexdump -C /dev/sda -s 524288 -n 16

01887800 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 |0123456789ABCDEF|

5.6. USB (flash disk) test

Insert USB flash disk then assure it is in IB113 device list

Note! This operation may damage the data stored in USB flash disk. Please make sure there is no critical data in the USB flash disk being used for this test.

5.6.1. Erase and check:

#dd if=/dev/sdb of=/dev/sdb bs=1024 count=1 seek=512

1+0 records in

1+0 records out

#hexdump -C /dev/sdb -s 524288 -n 16

01887800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

5.6.2. Write and check:

#echo -n "0123456789ABCDEF"|dd of=/dev/sdb bs=1024 count=1 seek=512

0+1 records in

0+1 records out

#hexdump -C /dev/sdb -s 524288 -n 16

01887800 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 |0123456789ABCDEF|

5.7. SDcard test

When booting from eMMC, SDcard is “/dev/mmcblk1” and able to see by “ls /dev/mmcblk1*” command:

/dev/mmcblk1 /dev/mmcblk1p2 /dev/mmcblk1p4 /dev/mmcblk1p5 /dev/mmcblk1p6

When booting from SDcard, replace test pattern “/dev/mmcblk1” to “/dev/mmcblk0”

Note! This operation may damage the data stored in USB flash disk. Please make sure there is no critical data in the USB flash disk being used for this test.

5.7.1. Erase and check

#dd if=/dev/zero of=/dev/mmcblk1 bs=1024 count=1 seek=512

1+0 records in

1+0 records out

#hexdump -C /dev/mmcblk1 -s 524288 -n 16

01887800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

5.7.2. Write and check

#echo -n "0123456789ABCDEF" | dd of=/dev/mmcblk1 bs=1024 count=1 seek=512

0+1 records in

0+1 records out

#hexdump -C /dev/mmcblk1 -s 524288 -n 16

01887800 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 |0123456789ABCDEF|

6. Appendix D – ADB configuration (For Android only)

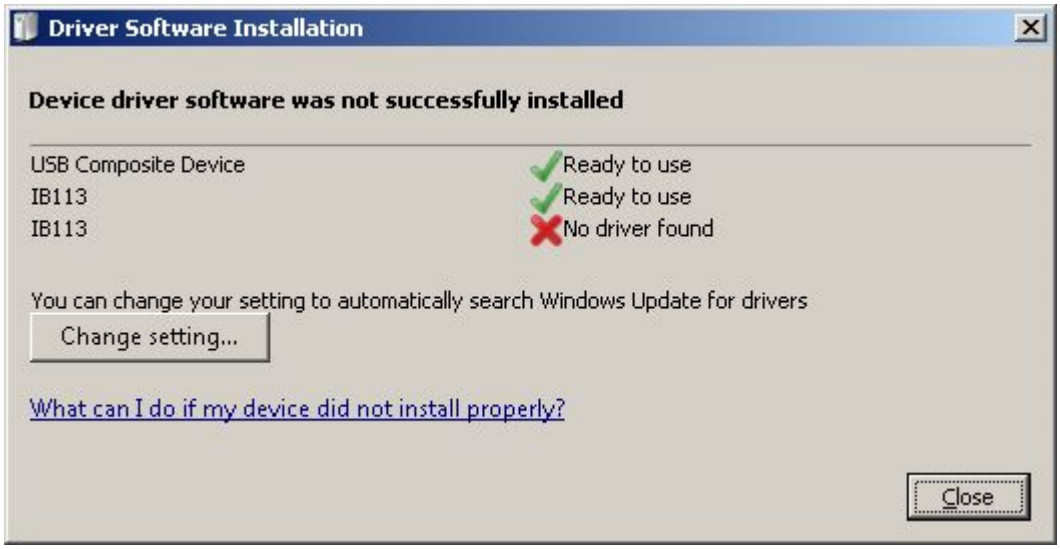
Update the ADB configuration to scan for the new vendor ID. Below are the steps to update the ADB configuration for Windows PC. These steps (and the steps for Linux PC as well) can also be found in the R10,3.x user guide.

1. Download ADB driver from iBASE website, like: IB113_ADB_usb_driver.7z

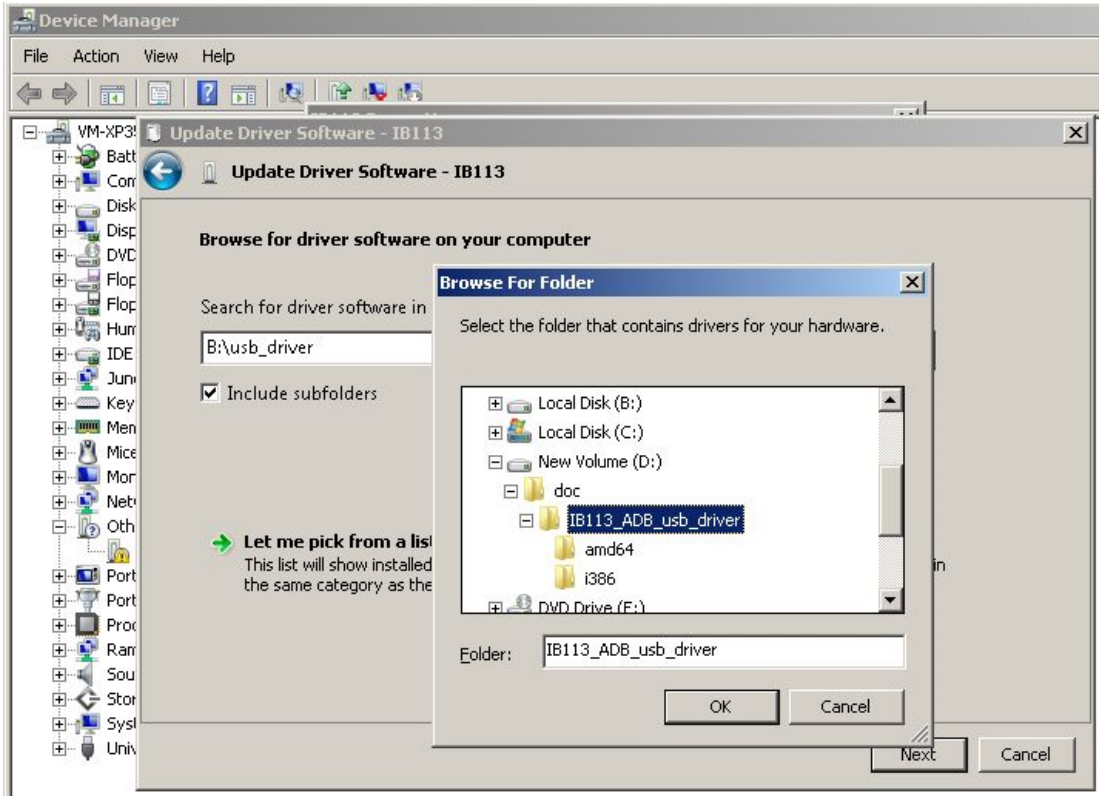
2. Enable the "USB debugging" option on the i.MX6 device

System settings -> Developer options -> USB debugging

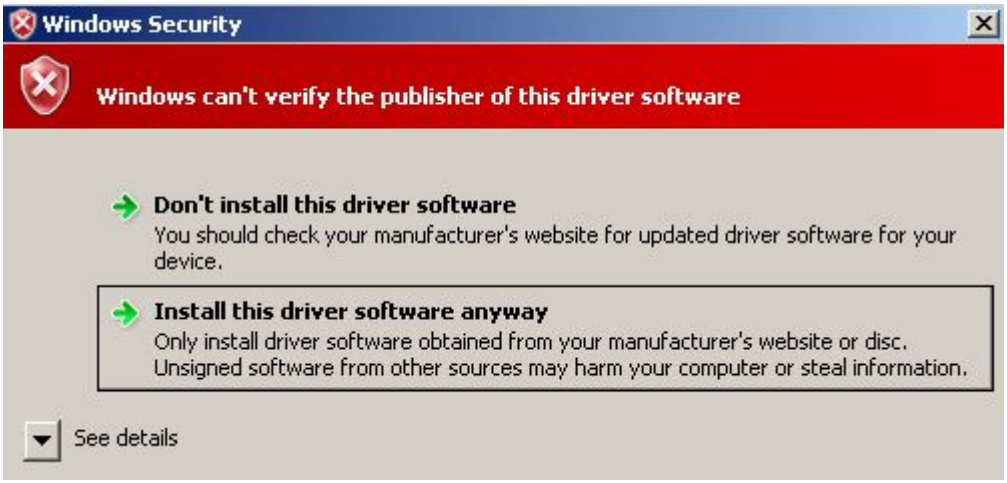
3. Connect the Android Device into PC, uninstall your old driver named "Android Phone" in the device manager, then re-install driver by scanning and locating .inf file under the directory you unpack the IB113_ADB_usd_driver.7z manually:



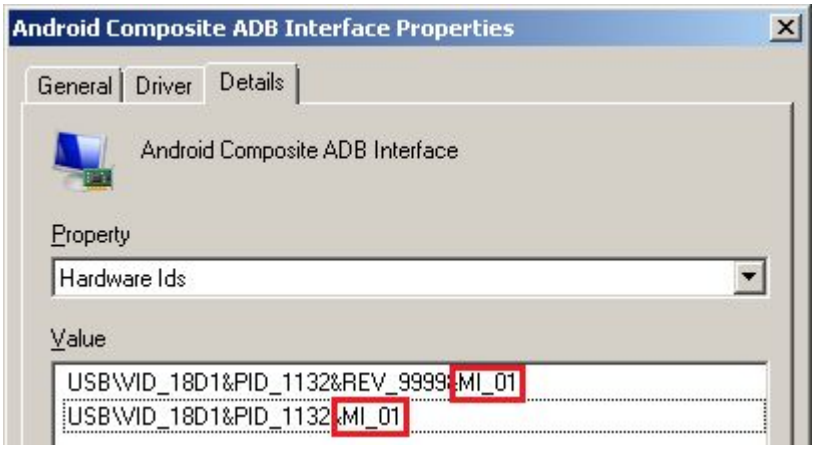
Unpack and install the driver



This is warning message should show If driver match.



If driver can not found, check device properties had “MI_01” string, if “MI_01” does not exist, the ADB and ADB driver not work.



4. Restart the ADB server

D:\adt-bundle-windows-x86_64-20130729\sdk\platform-tools> adb.exe kill-server

D:\adt-bundle-windows-x86_64-20130729\sdk\platform-tools> adb.exe start-server

5. Finally, test your ADB connection

D:\ adt-bundle-windows-x86_64-20130729\sdk\platform-tools> adb.exe devices

List of devices attached

0e46dbc9b9d40f17 device

ps. 0e46dbc9b9d40f17 id device UUID.

7. 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