AIV-QM97V1FL Series

An in-vehicle computer designed for comprehensive mobile applications

User Manual

Acrosser Technology Co., Ltd.
www.acrosser.com

Disclaimer

For the purpose of improving reliability, design and function, the information in this document is subject to change without prior notice and does not represent a commitment on the part of Acrosser Technology Co., Ltd.

In no event will Acrosser Technology Co., Ltd. be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Copyright

Trademarks

The product names appear in this manual are for identification purpose only. The trademarks and product names or brand names appear in this manual are the property of their respective owners.

Purpose

This document is intended to provide the information about the features and use of the product.

Audience

The intended audiences are technical personnel, not for general audiences.

Ver: 120-001
Date: Oct. 26, 2016

To read this User Manual on your smart phone, you will have to install an APP that can read PDF file format first. Please find the APP you prefer from the APP Market.

Table of Contents

1. System Introduction

The AIV-QM97V1FL Series is a fanless In-Vehicle Computer using Intel new 5th generation Core U processor designed to perform multiple in-car applications. These designs include smart power management, high efficient thermal module, and diversity of integrated communication technology such as wireless connectivity powered by 4G LTE.

1.1. Specifications

System

CPU	• **Intel 5th Core i3 -5010U** (3M Cache, 2.10 GHz Dual cores) • **Intel 5th Core i5 -5350U** (3M Cache, up to 2.90 GHz Dual cores) • **Intel 5th Core i7 -5650U** (4M Cache, up to 3.20 GHz Dual cores)
Memory	• DDR3L-1333/1600, Maximum capacity: 16GB • 2x 204-pin SO-DIMM sockets (non-ECC) • 4G/8G x1, x2

Display

Graphic Controller	• Integrated HD Graphic
Video Interface	• 1x HDMI (with locking bracket) • 1x DVI

Storage

SATA	• 1x SATA III Connector • 1x Power Connector (JST 2.54mm, 1x 4-pin)
M.2 (NGFF)	• 1x M.2 Connector
Disk Bay	• 1x Swappable 2.5” HDD bay with Anti-vibration / Anti-shock solution

Communication and I/O

Ethernet	• 2x PCIex1 Intel GbE chip via RJ-45 connectors
USB	• 2x External connectors for USB 3.0 • 2x USB 3.0 ports are set to 2x PCIex1 on M.2 Key-B. (Fixable I/O) • 2x External connectors for USB 2.0 • 1x USB2.0 for mini PCI-e slot • 1x USB2.0 for M.2 Key-B socket
Serial Ports	• COM1, COM2: DB9 (RS-232) • COM3: DB9 (422/485, selected by GPIO)
CANBUS	• Use GPIO DB15 connection 1. Support both CAN 2.0A and 2.0B protocol 2. Totally 9 items are supported for selectable baud rate from 10Kbps to maximum 1Mbps 3. API library for user development 4. CAN bus device status query
GPIO	• GPIO 4-in / 4-out, DB15 male • Input: 1. 4-input isolated channels 2. Max. voltage: 32V 3. Signal type: A. Open/Ground switch input B. Digital Logic Logic High: 3V ~ 32V Logic Low: 0V ~ 0.7V • Maximum input frequency: 10KHz (duty = 50%) • Output: 1. 4 channels 2. Output type: Open drain MOSFET driver 3. Output voltage range: +5V ~ +28V 4. Sink current: maximum 500mA each channel 5. Power on initial state: MOSFET off 6. Use clamped diode protection 7. Output default set: high (from GPO connector)
SIM	• Single SIM card I/O
LED	• 1x3 LED for power & status (onboard)

Expansion

M.2 (NGFF)	• 1x M.2 Key-M Socket 3 for SATA SSD card device • 1x M.2 Key-B Socket 2 for 4G LTE + GPS card device • (Reserve SIM interface)
Mini PCIe Slot	• Mini PCI-e Socket with USB 2.0 supported for WiFi+BT

Other Features

Audio	• 2x 3.5” phone Jack: Pink: MIC-in Green: Line out
Remote Switch	• 1x 3.5” phone Jack (Blue)
CMOS	• RTC (+/- 2 seconds for 24hours) • Lithium Battery (3V) for CMOS data backup
Hardware Monitoring	• RTC battery voltage • CPU and system temperature • CPU voltage • 12V, 5V, 3.3V voltage
Watchdog Timer	• Software programmable 0~255 Seconds, 0= disable timer.

Antenna

Antenna type	• 5x SMA (1x for GPS, 1x for Bluetooth, 1x for 4G LTE, 2x for WiFi)

Power Requirement

Power Supply	• 9V ~ 32V power input

Software

OS Support	• Win 7/8.1 (32/64-bit drive w/o WHQL) • Fedora 21 (32/64 bit) • Ubuntu 15.04 (32/64 bit)

Mechanical & Environment

Thermal Design	• Fanless (Heatsink)
Chassis	• Metal SPGC (Black printing color))
Dimension	• 290mm (W) x 190mm(D) x 45mm(H)
Vibration	• IEC 60068-2-64, 5~500Hz, 3GRMS(CF/SSD)
Shock	• IEC 60068-2-27, 50G 500m/s2 11MS
Operating Temperature	• 0°C ~ 60°C
Storage Temperature	• -40°C ~ 80°C
Certification	• CE / FCC class B / E Mark (ISO7637)

Optional Modules

Bluetooth/WiFi

- Mini PCI-e card device: WLAN (PCI-e) + Bluetooth (USB)

Bluetooth Antenna:

WiFi Antenna:

4G/GPS

- M.2 Key-B: WWAN Type 3042-S3-B: 4G LTE + GPS module

GPS Antenna:

4G LTE Antenna:

SATA SSD

- M.2 TYPE 2280-D2-B-M / 128GB

1.2. Package Contents

Check if the following items are included in the package.

- ☐ 1 x AIV-QM97V1FLCi3, AIV-QM97V1FLCi5, or AIV-QM97V1FLCi7 System
- ☐ 1 x Quick Manual
- ☐ 1 x Driver CD
- ☐ 1 x Screw Pack (For 2.5” HDD bracket: 4pcs)
- ☐ 1 x Terminal Block (Female 3-pin)
- ☐ 1 x Spare Fuse (10A)
- ☐ 1 x Remote Switch Cable
- ☐ 1 x GPIO/CAN/Driver ID Cable
- ☐ 1 x HDMI Locking Bracket

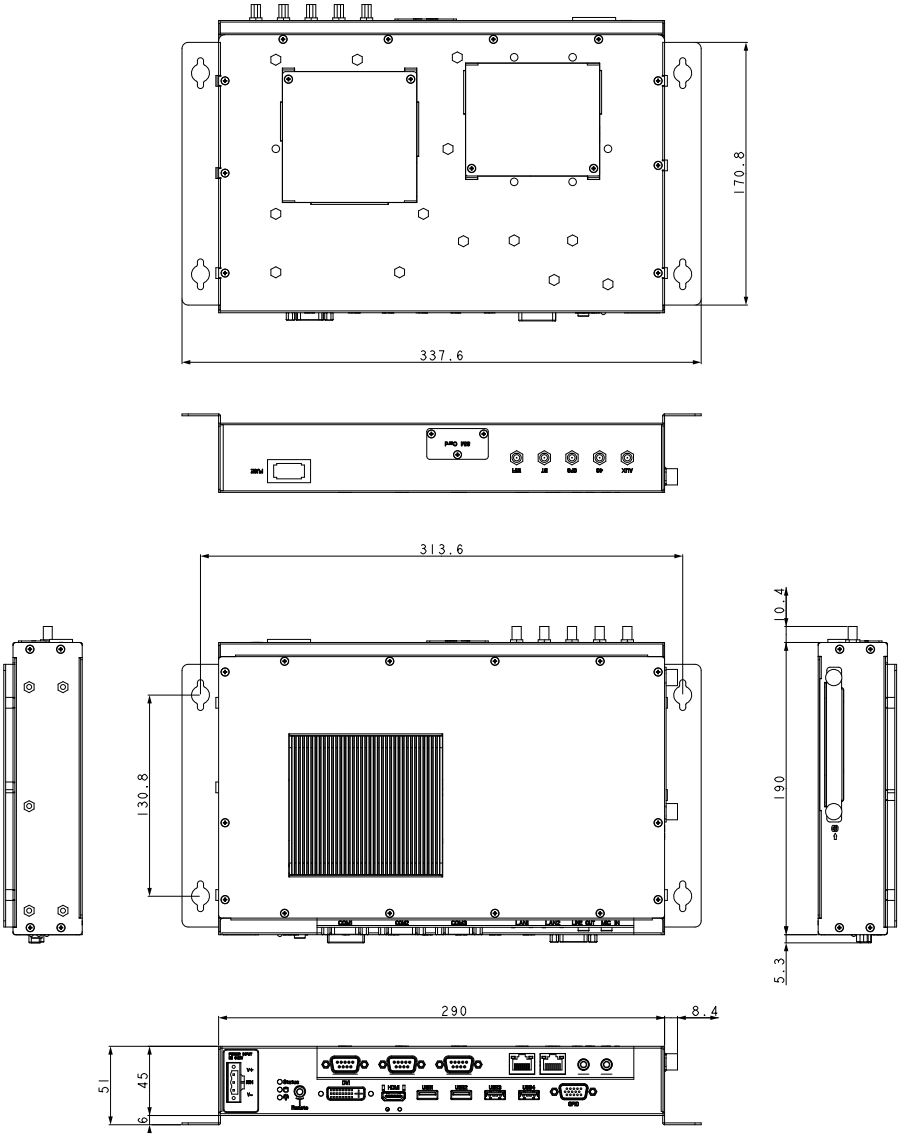
1.2.1. Model Type

Model	Description
AIV-QM97V1FLCi7	• Intel Core i7 -5650U
AIV-QM97V1FLCi5	• Intel Core i5 -5350U
AIV-QM97V1FLCi3	• Intel Core i3 -5010U

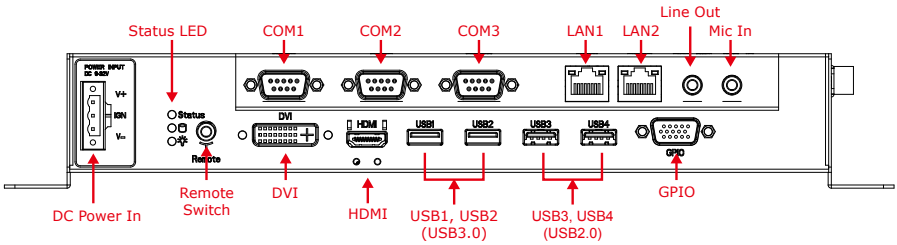
1.3. System Dissection

1.3.1. Dimensions

(Unit: mm)



1.3.2. Front I/O Panel



DC Power In

9~32VDC Power input.

Status/HDD/Power LED Display

LED	Light	Display
G	Green	Status
G	Green	SATA Device Activity
Y	Yellow	Power LED

Status LED Flashing Status:

A Status LED is used to indicate the status of the system. In normal condition, the LED will flash a number of blink to state the status. Each blink remains 200 ms ON followed by a 200 ms OFF. Each Cycle will have a 2-second OFF in between.

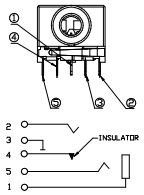
LED Blinking Numbers	Status
0 (Constant On)	Power output runs normally.
1	Standby Mode (System off)
2	Hard Off Delay
3	Power On Delay
4	Shutdown Delay
5	Boot Up Delay
6	Soft Off Delay

If abnormal condition occur, the LED will flash a 1.5-second pulse followed by numbers of 200 ms pulse to indicate the error status.

LED Blinking Numbers	Error Status
1 Long, 1 Short	System cannot be turned on or was turned off because battery voltage is below the Battery Low Voltage.
1 Long, 2 Short	System on/off fail. When motherboard cannot turn on or turn off after retry.

Remote Switch (Blue)

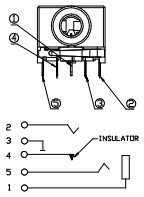
SPST (Single Pole, Single Throw) switch input.



Pin #	Signal
1	GND
2	SPST button-in
3	NC
4	NC
5	SPST button GND

Line Out (Green)

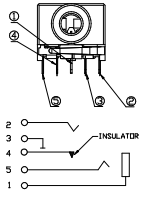
Line out phone jack.



Pin #	Signal
1	GND
2	Line-Out Left channel
3	GND
4	Jack Detect
5	Line-Out Right channel

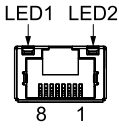
Mic In (Pink)

Microphone input jack.



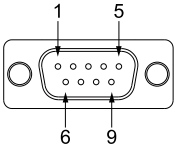
Pin #	Signal
1	GND
2	MIC-in Left channel
3	GND
4	Jack Detect
5	MIC-in Right channel

LAN1 / LAN2



LED	Light	Status
LED1	Off	10Mbps
	Green	100Mbps
	Orange	1000Mbps
LED2	Yellow	Link
	Blink	Link with Activity
	Off	No Link

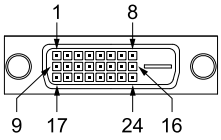
COM1, COM2, COM3



COM1, COM2		COM3		
Pin #	RS-232 Signal	Pin #	RS-422 Signal	RS-485 Signal
1	DCD	1	TX+	DATA+
2	SIN	2	TX-	DATA-
3	SOUT	3		
4	DTR	4		
5	GND	5	GND	GND
6	DSR	6	GND	GND
7	RTS	7		
8	CTS	8	RX-	
9	RI	9	RX+	

DVI

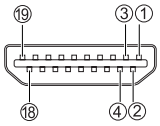
DVI-D single link connector.



Pin#	Signal	Pin#	Signal	Pin#	Signal
1	TMDS Data2 -	9	TMDS Data1 -	17	TMDS Data0 -
2	TMDS Data2 +	10	TMDS Data1 +	18	TMDS Data0 +
3	Shield GND	11	Shield GND	19	Shield GND
4	NC	12	NC	20	NC
5	NC	13	NC	21	NC
6	DDC Clock	14	+5V	22	Shield GND
7	DDC Data	15	GND	23	TMDS Clock +
8	NC	16	Hot Plug Detect	24	TMDS Clock -

HDMI

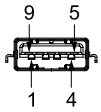
HDMI connector.



Pin #	Signal	Pin #	Signal
1	DATA2+	2	GND
3	DATA2-	4	DATA1+
5	GND	6	DATA1-
7	DATA0+	8	GND
9	DATA0-	10	CLK+
11	GND	12	CLK-
13	NC	14	NC
15	DDCCL	16	DDCDA
17	GND	18	+5V
19	HPD		

USB1, USB2

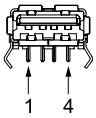
Standard USB 3.0 Type-A connectors.



Pin #	Signal	Pin #	Signal
1	5V	5	SS_RX -
2	Data -	6	SS_RX +
3	Data +	7	GND
4	GND	8	SS_TX -
		9	SS_TX +

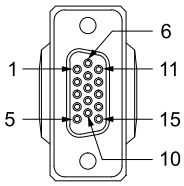
USB3, USB4

Standard USB 2.0 Type-A connectors.



Pin #	Signal
1	VCC (+5V)
2	Data -
3	Data +
4	GND

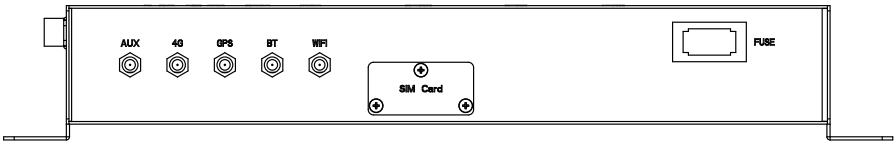
GPIO



GPIO DB15 Cable

Pin #	Definition	Wire Color	Pin #	Definition	Wire Color
1	GPO0	Brown	2	GPO1	Orange
3	GPO2	Green	4	GPO3	Blue
5	GND	Black	6	GND	Glay
7	CAN_H	Red/White	8	CAN_L	White
9	GND	Red	10	i-Button	Purple
11	GPI4	Light Green	12	GPI5	Light Blue
13	GPI6	Pink	14	GPI7	Brown/ White
15	VCC12A	Yellow			

1.3.3. Rear I/O Panel



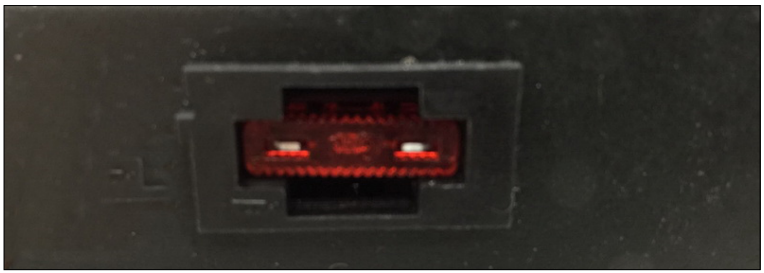
Antenna Socket

Reserved for installation of 5x optional SMA-type antenna (1x for GPS, 1x for Bluetooth, 1x for 4G LTE, 2x for WiFi)

SIM Card Holder

Reserved for installation of your SIM card.

Blade-type Fuse Holder



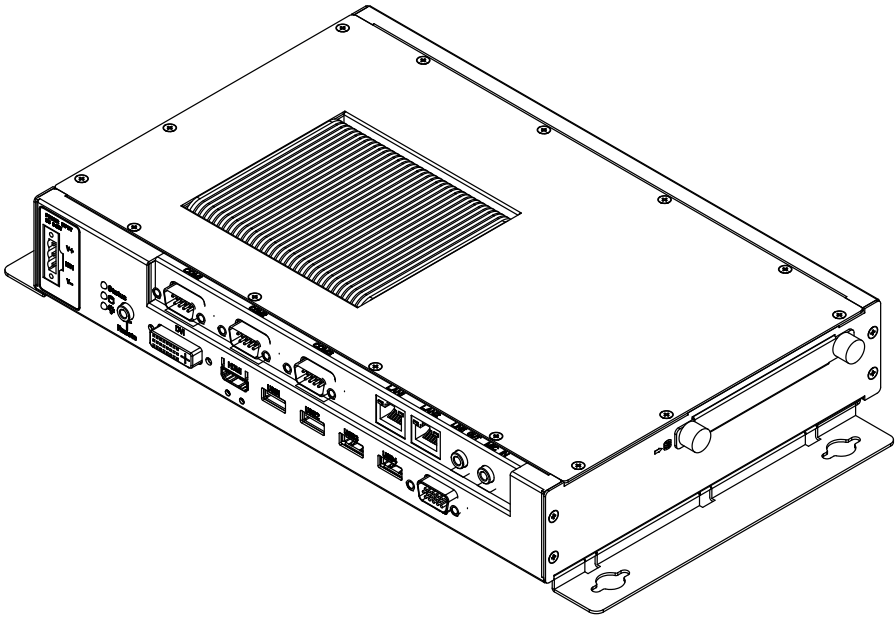
Power-input fuse suggestion:

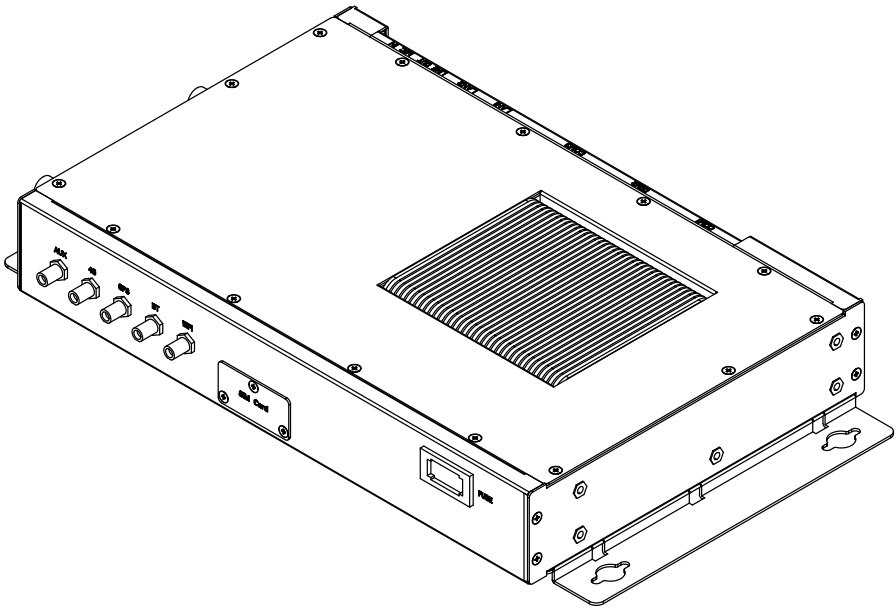
Output: 12V/100W (Input: 9V~32V/111W, Efficiency: 90%)

Car Battery	Blade-type fuse suggestion	Remarks
12V System	CONQUER ATQ-10	Voltage Rating: 32V; Current Rating: 10A
24V System	CONQUER ATQ-5	Voltage Rating: 32V; Current Rating: 5A

Note: *You may have to use a needle-nose pliers to grip on the fuse and pull it out.*

1.3.4. Side I/O Panel

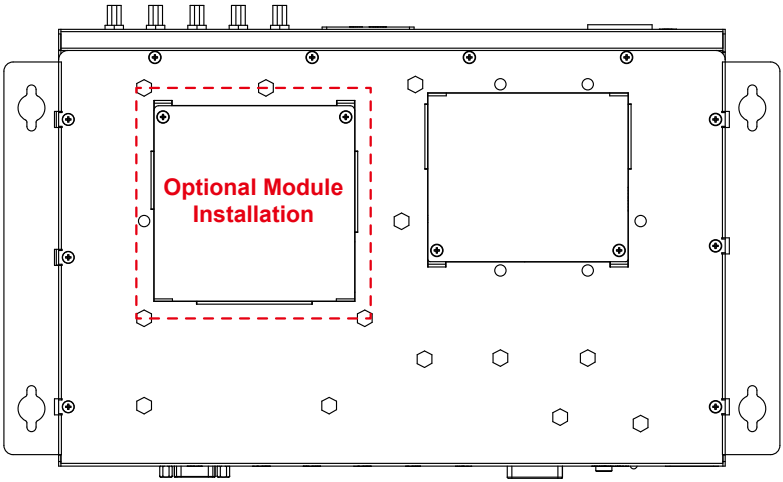




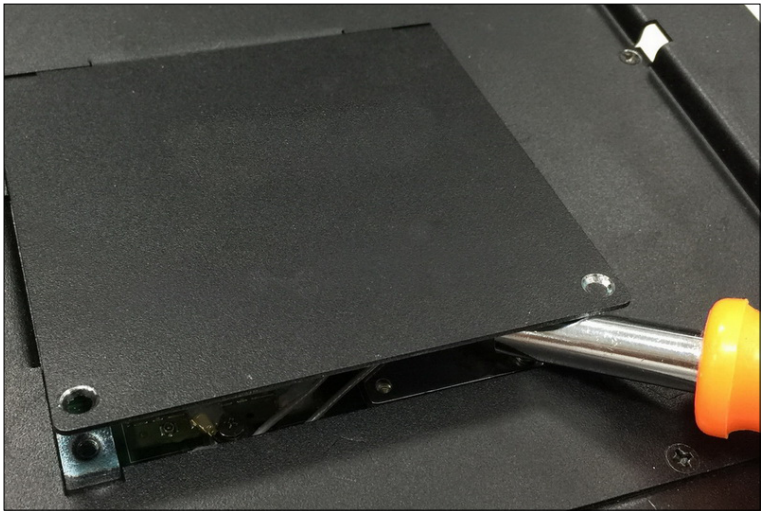
2. Components Assembly

2.1. Optional Module Installation

Step 1: The compartment to install the optional modules is located at the chassis bottom. Loosen the two screws that lock the hatch cover.



Step 2: Use a flat-head screwdriver to lift up the hatch cover.

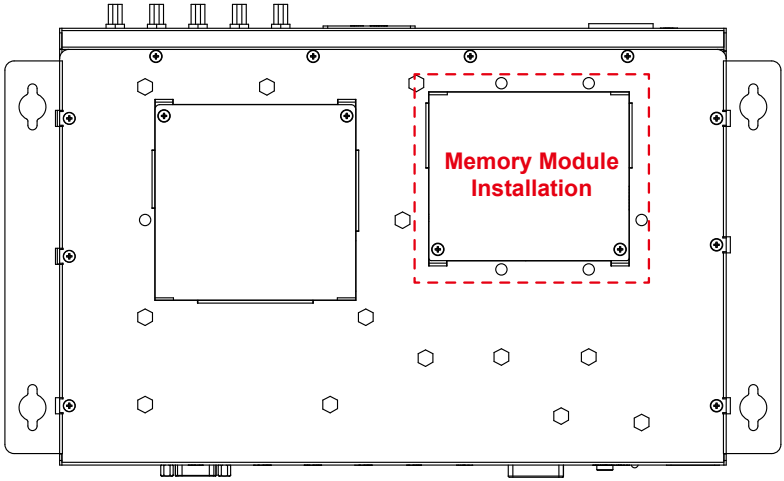


Step 3: Install your optional modules according to your need.

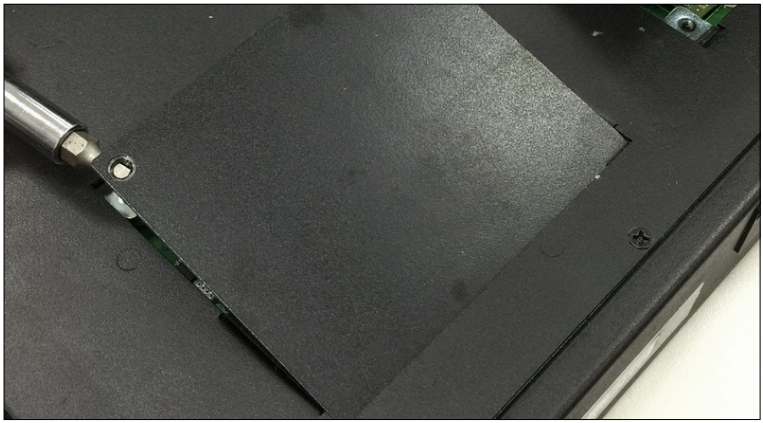
Step 4: After finished installation, close the hatch cover and lock with screws.

2.2. Memory Module Installation

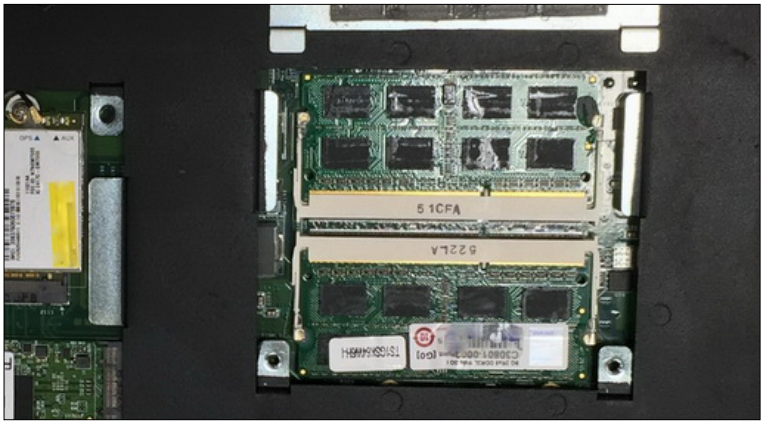
Step 1: The compartment to install the memory modules is located at the chassis bottom. Loosen the two screws that lock the hatch cover.



Step 2: Use a flat-head screwdriver to lift up the hatch cover.



Step 3: Install the DDR3 memory into the socket.
(Align the notch key on the module with the one on the socket when installing the memory module.)



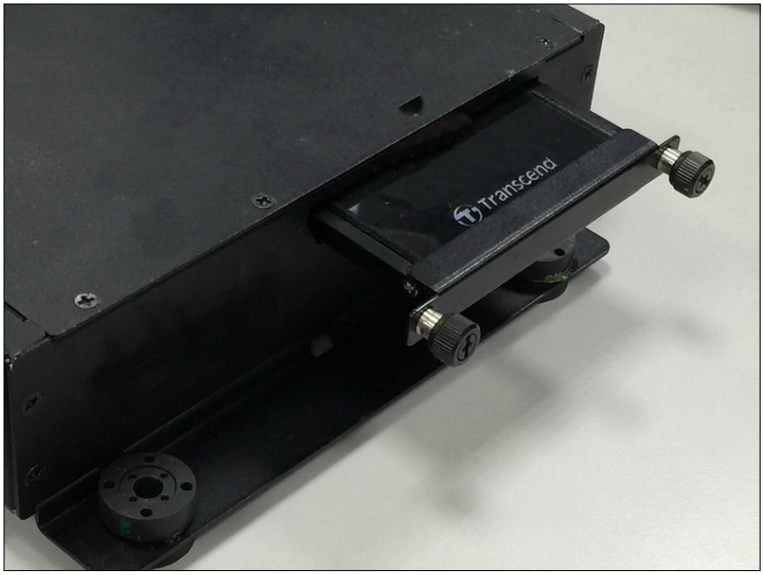
Step 4: After finished installation, close the hatch cover and lock with screws.

2.3. 2.5” SATA SSD Installation

Step 1: Loosen the two disk-tray screws by fingers. Pull out the disk-tray and install your 2.5” SATA disk.

Step 2: Lock the disk with 4 screws provided in the package.

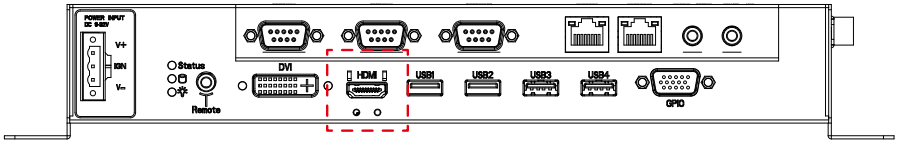
Step 3: Firmly push the disk-tray back into the disk compartment. The disk is now connected with the mainboard. Lock the two disk-tray screws by fingers.



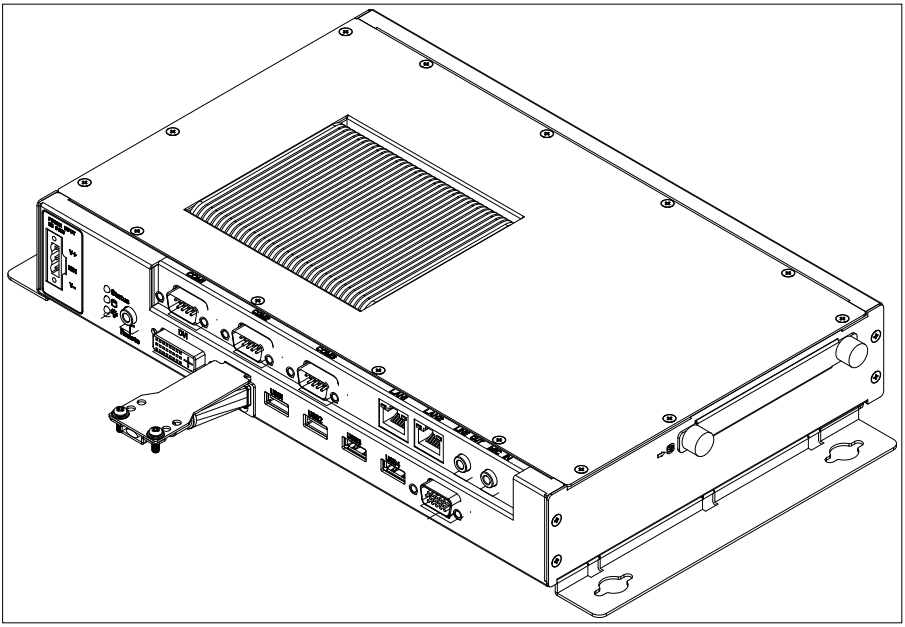
Step 4: Complete.

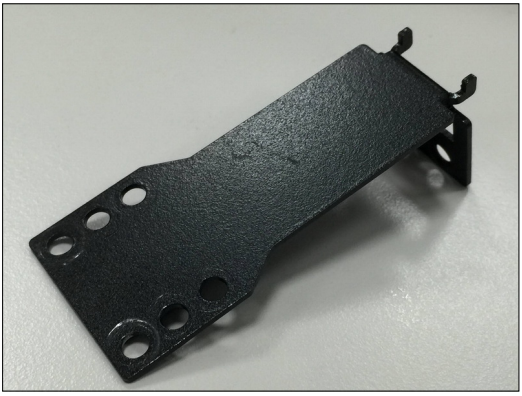
2.4. HDMI Connection

You can find In the package an HDMI locking-bracket set. This gaget is designed to secure your HDMI cable connection.

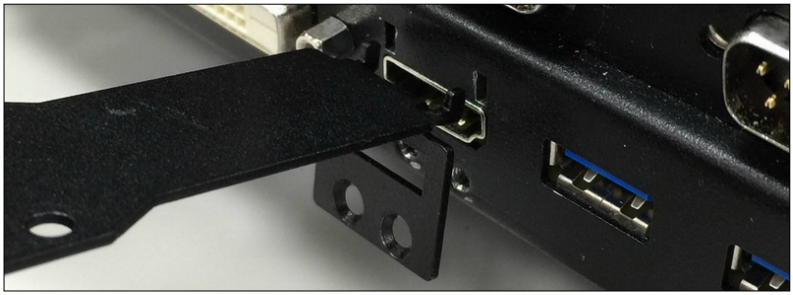


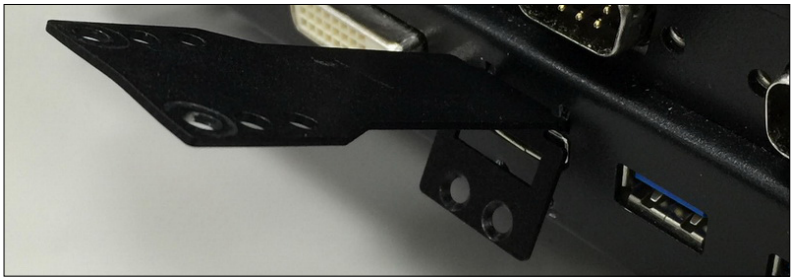
HDMI Connection



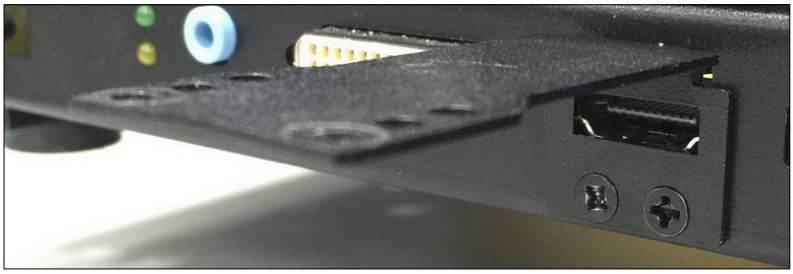


Step 1: As shown below, insert the locking-bracket into the chassis body.





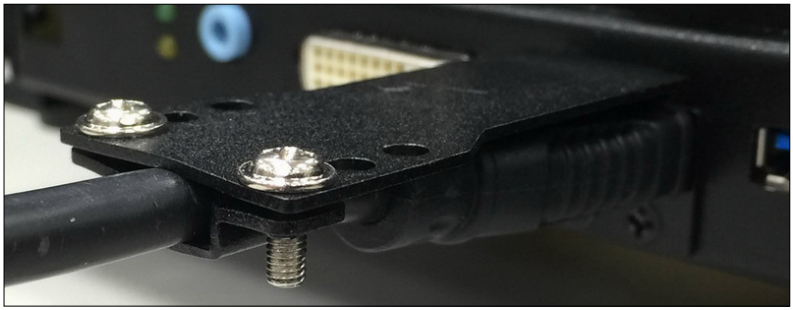
Step 2: Lock the HDMI locking-bracket with the two black screws that came with the package.



Step 3: Plug your HDMI cable head into the HDMI socket.

Step 4: Firmly push the HDMI cable all the way into the socket.

Step 5: Fasten the HDMI cable-end with a cable-holder. Lock the cable-end to the bracket with this cable-holder by two white screws that came with the package. (There are two types of cable-holder provided: 4mm and 7mm. Use the type 4mm for HDMI cable of thinner than 6mm in diameter. Use the type 7mm for HDMI cable of thicker than 6mm in diameter.)

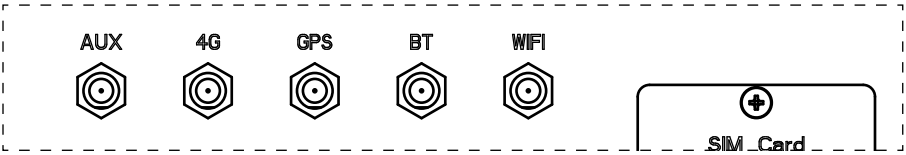


Step 6: The HDMI locking-bracket is now held into position tightly.

Step 7: Complete.

2.5. Antenna Connection

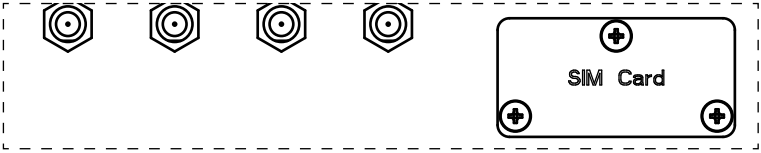
Connect your antennas needed according to your system configuration.



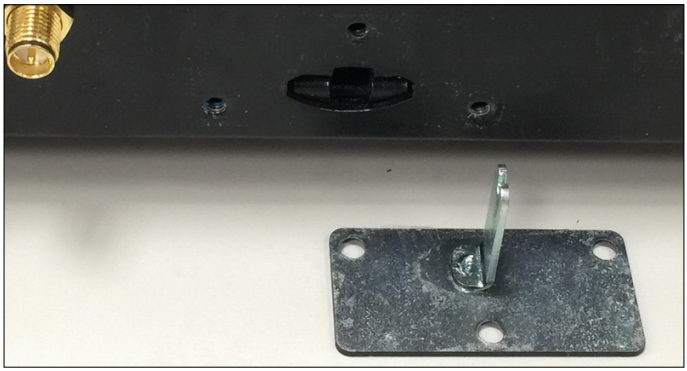
The antennas shown for installation demonstration in this photo from left to right are: AUX (Reserved with a cap covered), 4G LTE, GPS, Bluetooth, WiFi.

2.6. SIM Card Installation

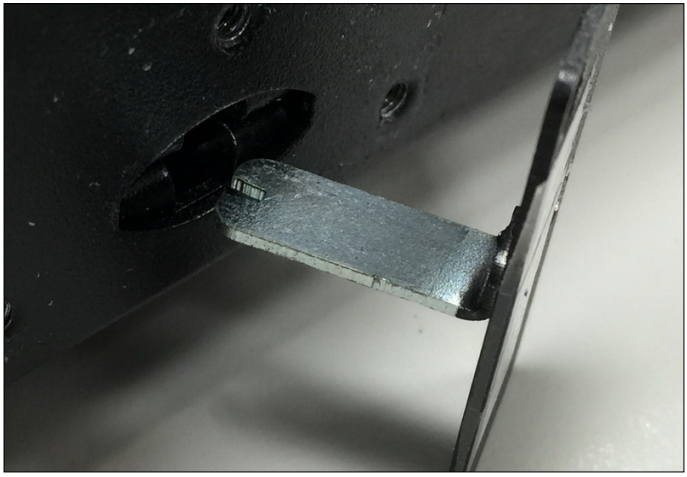
Step 1: Loosen three screws on the SIM card cover-plate.



Step 2: Take out the SIM card cover-plate. Insert your SIM card into the SIM slot. Pay attention to its orientation, and do not scratch the contacts.



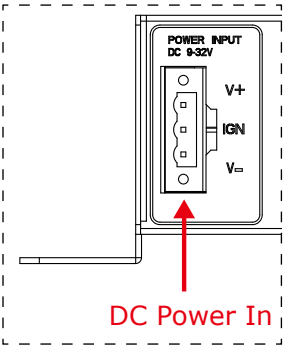
Step 3: The SIM card cover-plate is mechanically designed to fit for pushing in the SIM card into its position. Turn the cover-plate 90 degrees to push the SIM card into its position.

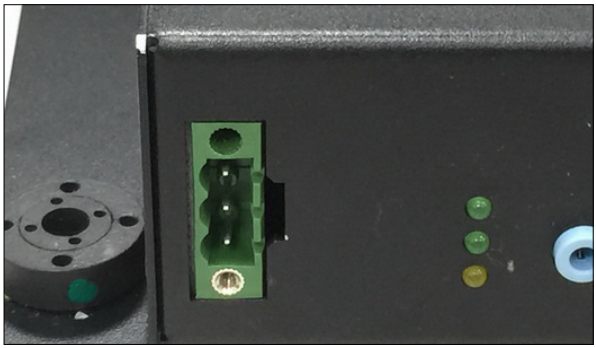


Note: To remove the SIM card, turn the cover-plate 90 degrees and stick it into the slot. Push the SIM card inward and then release. The card bounce outward a little bit. Use a tweezers to pinch the card out.

2.7. Power Connection

Connect your power cable.





	Pin #	Signal
9V ~ 32V DC input connector Terminal Block: 3-pin Pitch: 5.08mm	V+	9V ~ 32V DC Power Input
	IGN	Ignition On (Hi Active)
	V-	GND

3. BIOS Settings

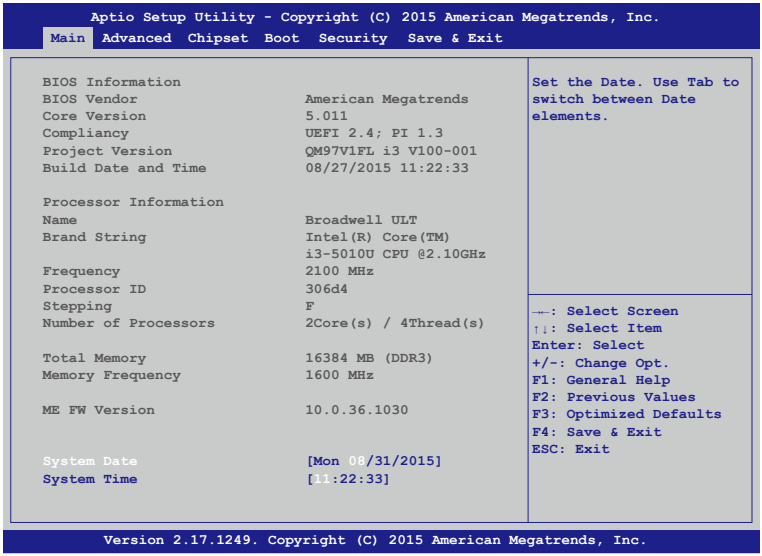
This chapter describes the BIOS menu displays and explains how to perform common tasks needed to get the system up and running. It also gives detailed explanation of the elements found in each of the BIOS menus. The following topics are covered:

- Main Setup
- Advanced Setup
- Chipset Setup
- Boot Setup
- Security Setup
- Save & Exit Setup

Once you enter the Award BIOS™ CMOS Setup Utility, the Main Menu will appear on the screen. Use the arrow keys to highlight the item and then use the <Pg Up> <Pg Dn> keys to select the value you want in each item.

3.1. Main Setup

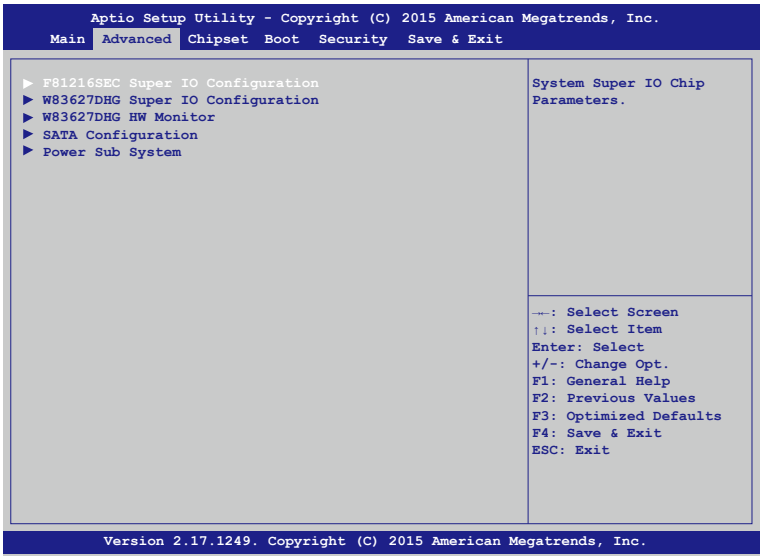
The BIOS setup main menu includes some options. Use the [Up/Down] arrow key to highlight the option, and then press the [**Enter**] key to select the item and configure the functions.



Note: *Listed at the bottom of the menu are the control keys. If you need any help with the item fields, you can press <**F1**> key, and it will display the relevant information.*

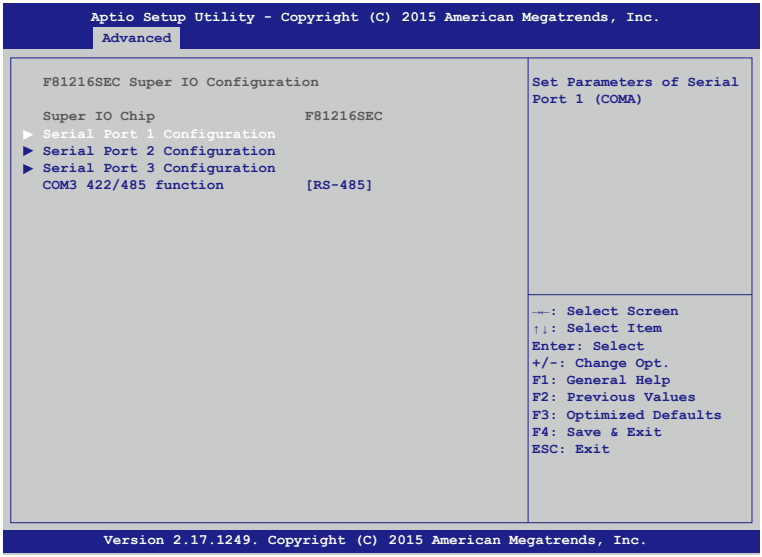
- **System Date**
  Set the system date. Note that the 'Day' automatically changes when you set the date.
- **System Time**
  Set the system time.

3.2. Advanced Setup



3.2.1. F81216SEC Super IO Configuration

System second super IO chip parameters.



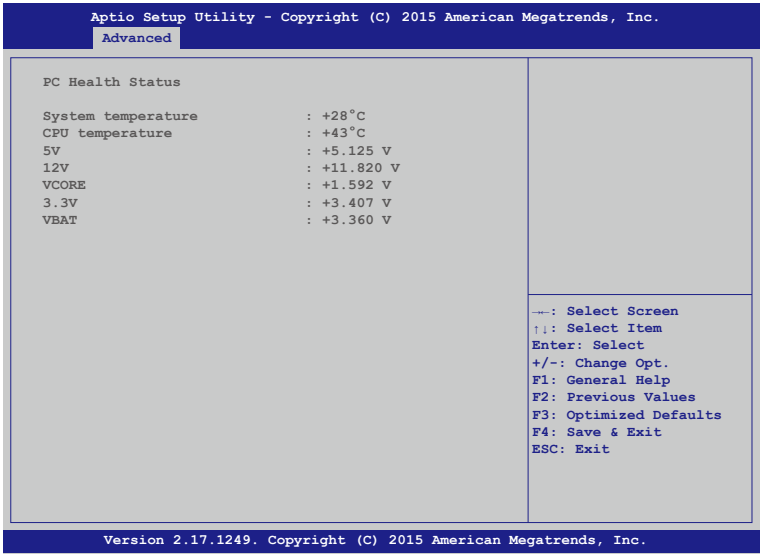
- **Serial Port 1 ~ Serial Port 3 Configuration**
  This option sets the parameters of COM1 ~ COM3.
- **COM3 422/485 function**
  This option sets the COM3 function to RS-422 or RS-485.

3.2.2. W83627DHG Super IO Configuration

- **Serial Port 1 ~ Serial Port 2 Configuration**
  This option sets the parameters of COM1 ~ COM2.

3.2.3. W83627DHG HW Monitor

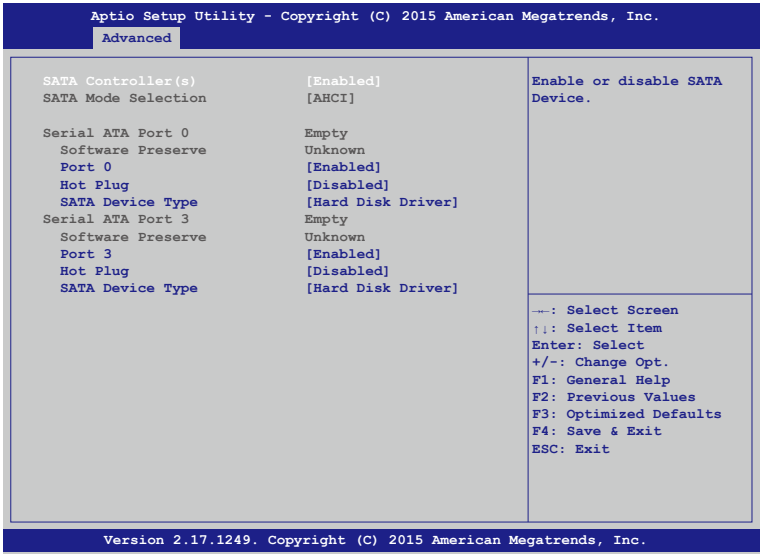
Monitor hardware status.



- **System temperature**
  This item displays the system temperature.
- **CPU temperature**
  This item displays the CPU temperature.
- **5V**
  This item displays the 5V voltage level.
- **12V**
  This item displays the 12V voltage level.
- **VCORE**
  This item displays the VCORE voltage level.
- **3.3V**
  This item displays the 3.3V voltage level.
- **VBAT**
  This item displays the battery voltage level.

3.2.4. SATA Configuration

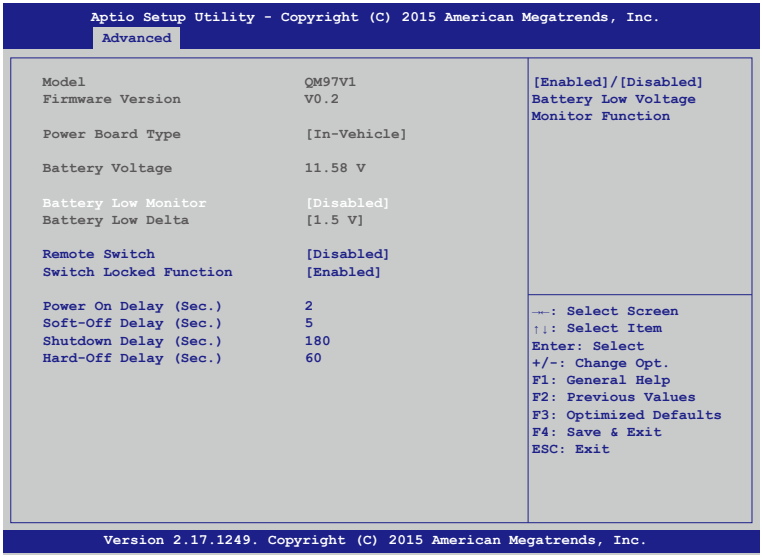
SATA device options settings.



- **SATA Controller(s)**
  Enable or disable SATA device.
- **SATA Mode Selection**
  Determines how SATA controller(s) operate.

3.2.5. Power Sub System

Power Sub System.



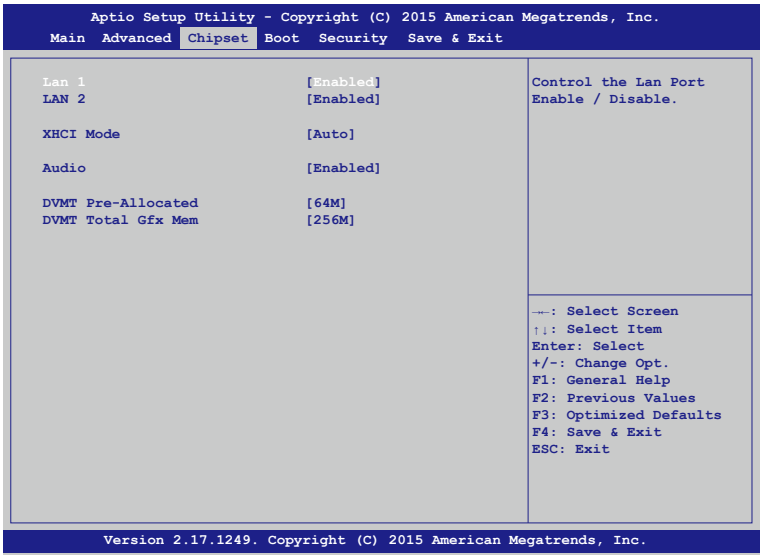
- **Power Board Type**
  Displays the power board type.
- **Battery Voltage**
  Detects and display the battery voltage level.

Note: *The following items apper only if the "Power Board Type" is [In-Vehicle].*

- **Battery Low Monitor**
  Enables or disables the monitor function of low battery voltage.
- **Battery Low Delta**
  Sets the battery delta level. Once the battery voltage drops below this level, the battery will be detected as battery low.
- **Remote Switch**
  Enables or disables the function of remote switch.
- **Switch Locked Function**
  Enables or disables the function of switch lock.
- **Power On Delay (Sec.)**
  The delay between power on and system work.
- **Soft-Off Delay (Sec.)**
  The delay before system shutdown.

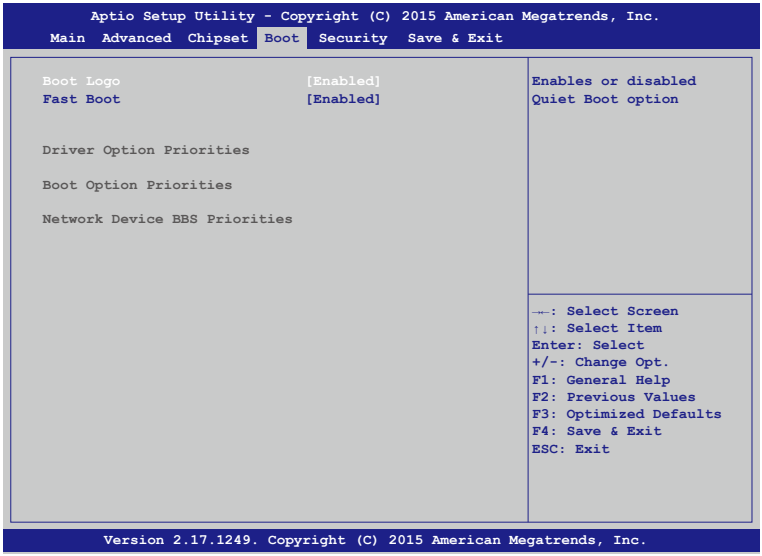
- **Shutdown Delay (Sec.)**
  The delay between system shutdown and system off.
- **Hard-Off Delay (Sec.)**
  The delay before all power off.

3.3. Chipset Setup



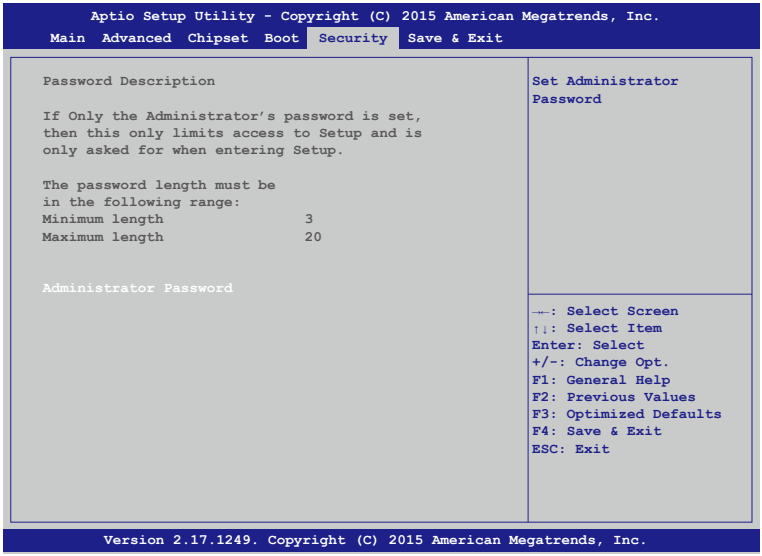
- **Lan 1, Lan 2**
  Control the LAN Port Enable / Disable.
- **xHCI Mode**
  Select the operation mode of xHCI controller.
- **Audio**
  Control the detection of the Azalia device.
- **DVMT Pre-Allocated**
  Select DVMT 5.0 Pre-Allocated (Fixed) Graphics Memory size used by the Internal Graphics Device.
- **DVMT Total Gfx Mem**
  Select DVMT 5.0 Total Graphics Memory size used by the Internal Graphics Device.

3.4. Boot Setup



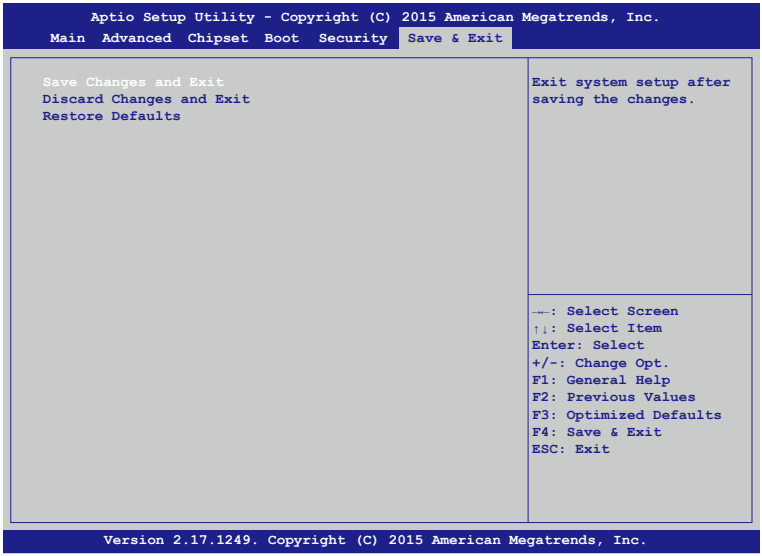
- **Boot Logo**
  Enables or disables Quiet Boot option.
- **Fast Boot**
  Enables or disables boot with initialization of a minimal set of devices required to launch active boot option. Has no effect for BBS boot options.
- **Driver Option Priorities**
  This item enables adding, deleting, or selecting the drive options to be shown in the setup sequence.
- **Boot Option Priorities**
  Set the system boot order.
- **Network Device BBS Priorities**
  Set the system boot order of network device.

3.5. Security Setup



- **Administrator Password**
  Set Administrator Password.

3.6. Save & Exit Setup



- **Save Changes and Exit**
  Exit system setup after saving the changes.
- **Discard Changes and Exit**
  Exit system setup without saving any changes.
- **Restore Defaults**
  Restore/Load Default values for all the setup options.

4. Function Description

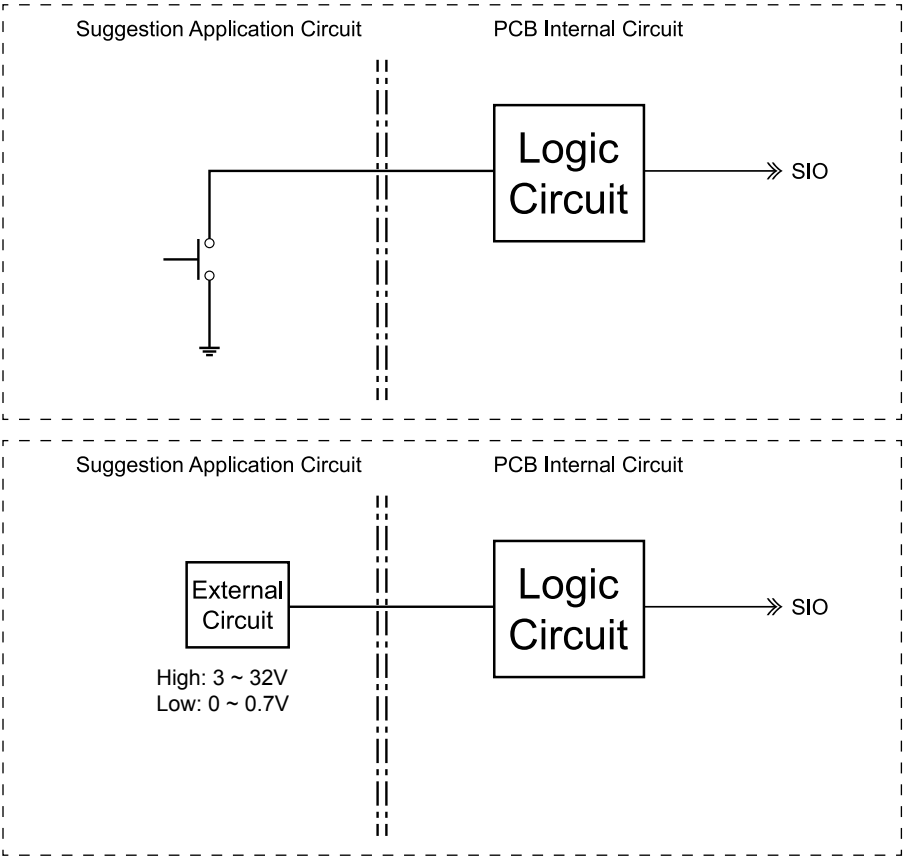
4.1. Power input connection

AIV-QM97V1FL Series needs +9~32V to power the board.

4.2. Digital Inputs

There are 4 clamped diode protection digital inputs on GPIO1 connector. You can read the status of any input through the software API. These digital inputs are general purpose input. You can define their purpose for any digital input function. Please refer to the "**Software Installation and Programming Guide**" chapter for the detailed information on how to use the API.

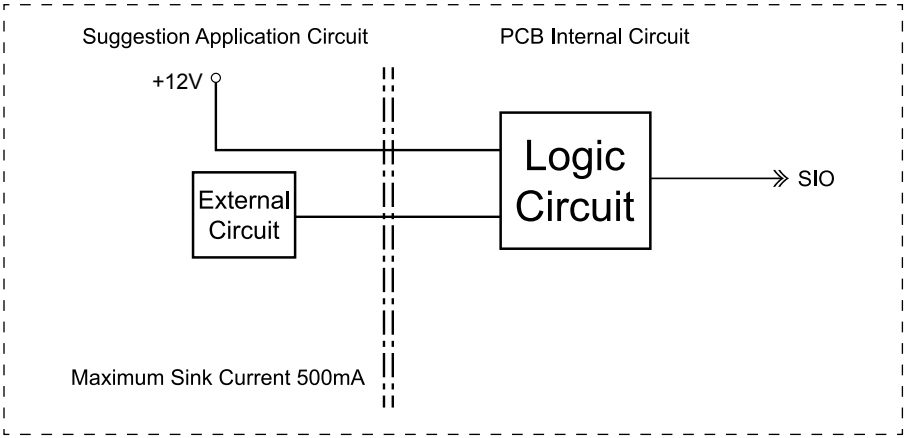
The following diagrams state how to connect the digital inputs to devices on the embedded system.



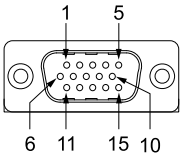
4.3. Digital Outputs

There are 4 clamped diode protection digital outputs on GPIO1 connector. You can control the output status of these digital outputs through the software API. The four digital outputs are capable sink maximum 500 mA current for each channel and maximum output voltage is 12V. The output reference voltage of device, please connect to GPIO #VCC12V(Pin15). These digital outputs are general purpose outputs. Please refer to the "**Software Installation and Programming Guide**" chapter for the detailed information on how to use the API.

The following diagrams state how to connect the digital outputs to the devices on the system.



GPIO pin definition:



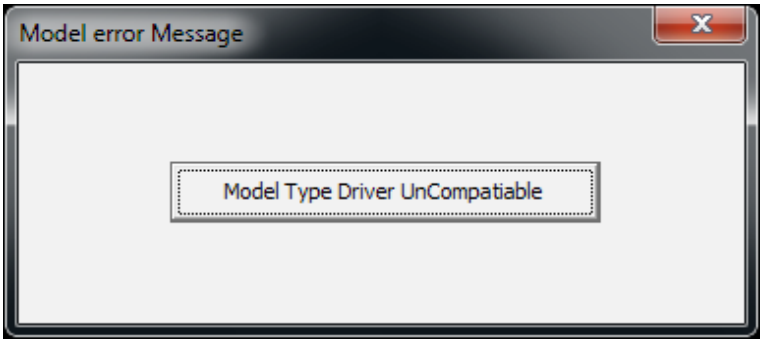
Pin #	Signal	Pin #	Signal
1	GPO0	2	GPO1
3	GPO2	4	GPO3
5	GND	6	GND
7	CAN_H	8	CAN_L
9	GND	10	I-Button
11	GPI4	12	GPI5
13	GPI6	14	GPI7
15	VCC12A		

5. Driver and Utility Installation

5.1. Driver CD Interface Introduction

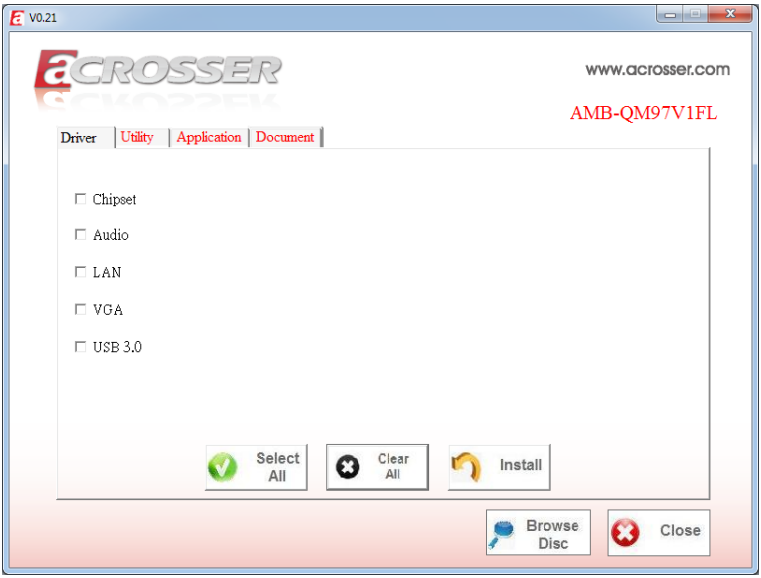
Acrosser provides a Driver CD compiled with all the drivers, utilities, applications and documents this product may need.

Put the Driver CD into your CD-ROM drive. The Driver CD will automatically detect the mainboard information to see if they are matched. The following error messages appear if you use an incorrect Driver CD version with your mainboard. Please find the correct Driver CD to proceed.

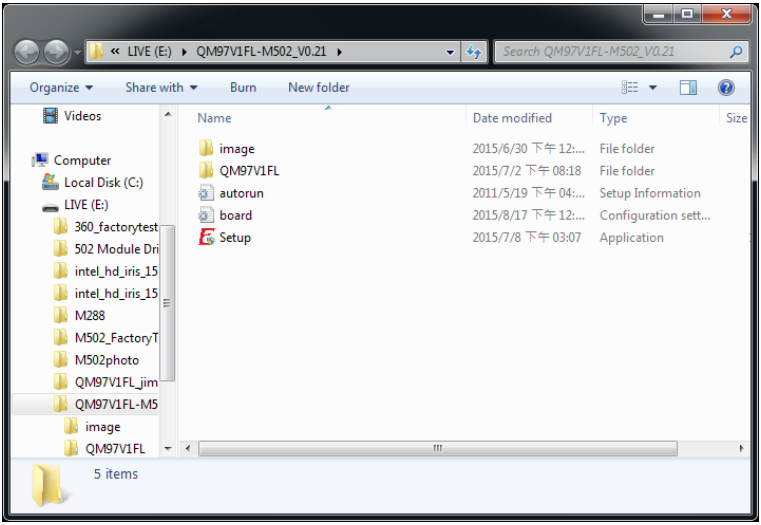


Note: *The following snap shots were taken under Windows 7, slightly different from those under Windows 8.1 installation.*

Put the correct Driver CD of your mainboard into your CD-ROM drive. The following installation screen should appear.

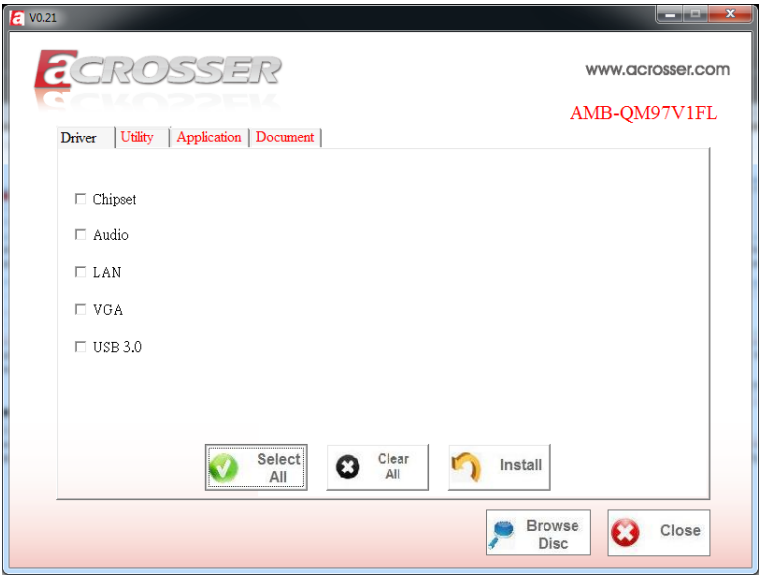


If not, enter the root folder of the Driver CD, run the excution file "**Setup.exe**".

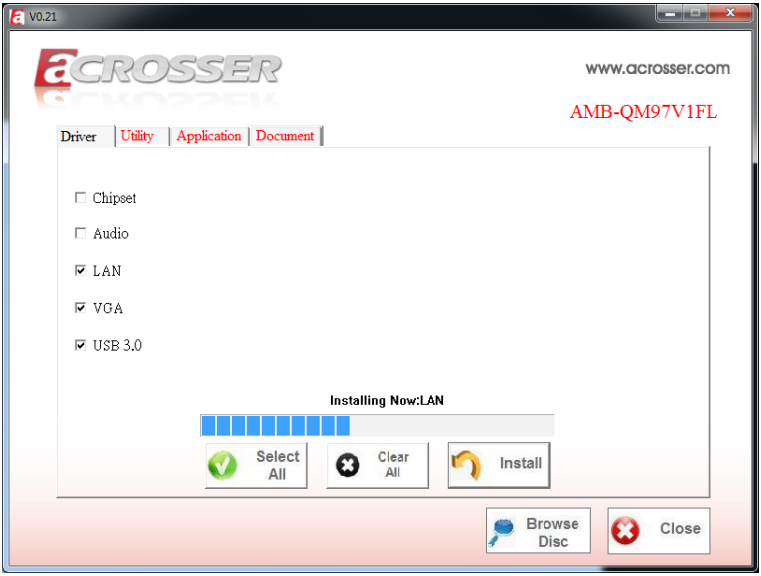


5.2. Driver Installation Page

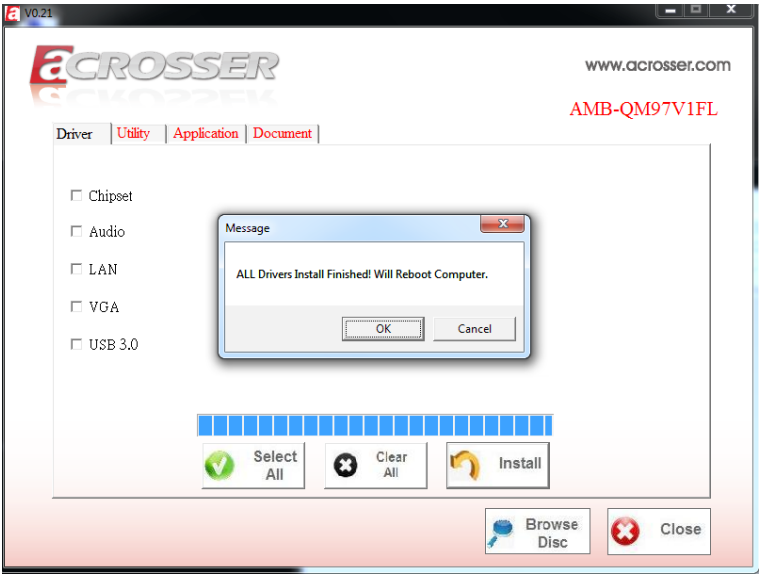
Step 1: Select the "**Driver**" tab.



Step 2: Click the "**Select All**" button to select all the driver checkboxes, and then click "**Install**" button to start installing all the selected drivers.



Step 3: The driver installation completed. The configuration will be valid after reboot.



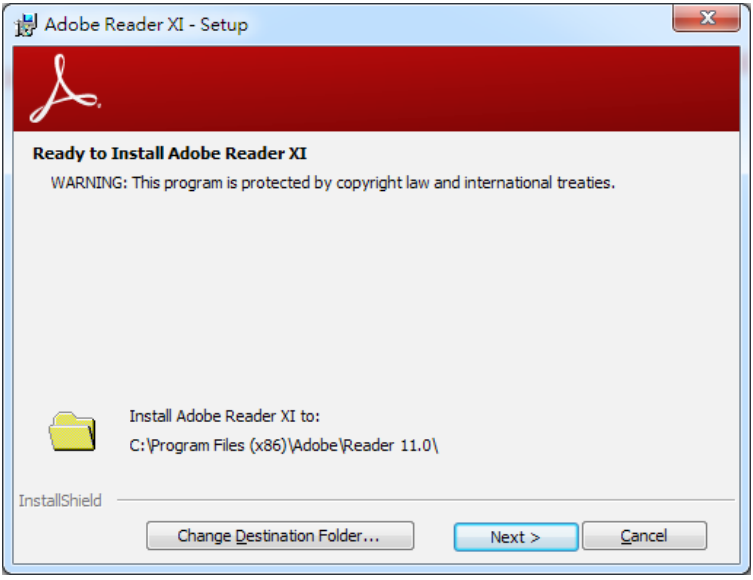
*Note: Select the "**Clear All**" button will clear all the selections, and then you can select the driver you want to install one by one, but the "**Chipset**" driver has to be installed before installing all the others.*

5.3. Application Installation Page

5.3.1. Acrobat Reader

Step 1: Select the "**Application**" tab. Click the "**Acrobat Reader**" box.

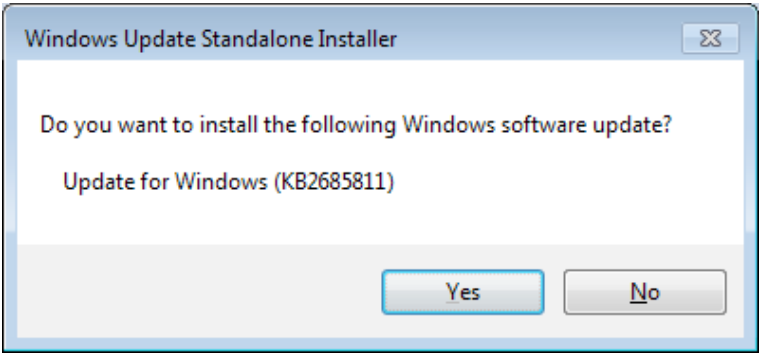
Step 2: Click [**Next**] button to complete the installation. This application is needed for reading the User Manual in PDF format.



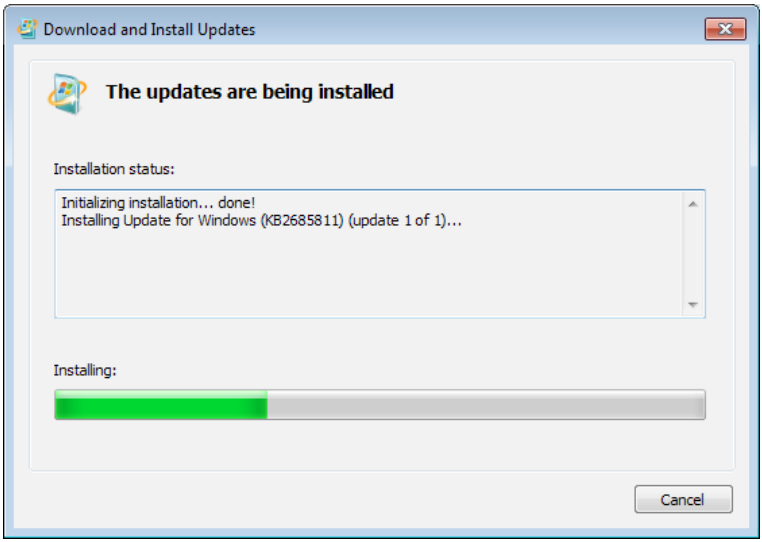
5.3.2. Driver Frameworks

Step 1: Select the "**Application**" tab. Click the "**Driver Frameworks**" box.

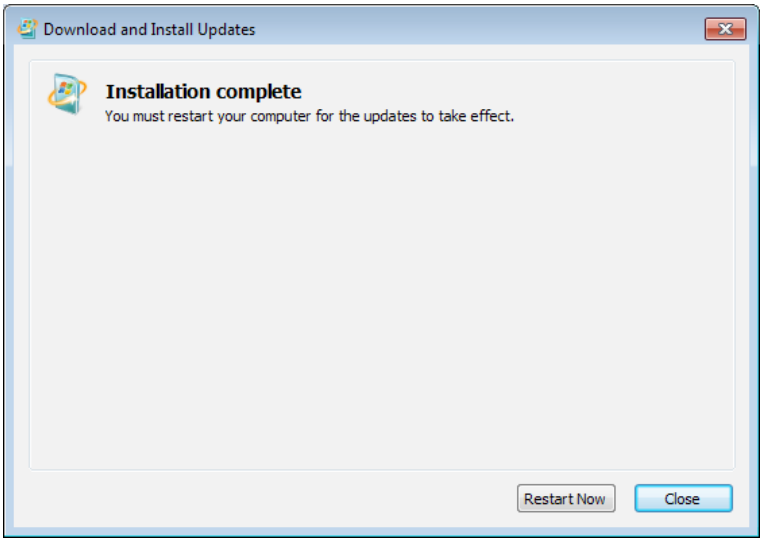
Step 2: Click [**Yes**] button.



Step 3: The Windows is updating now.



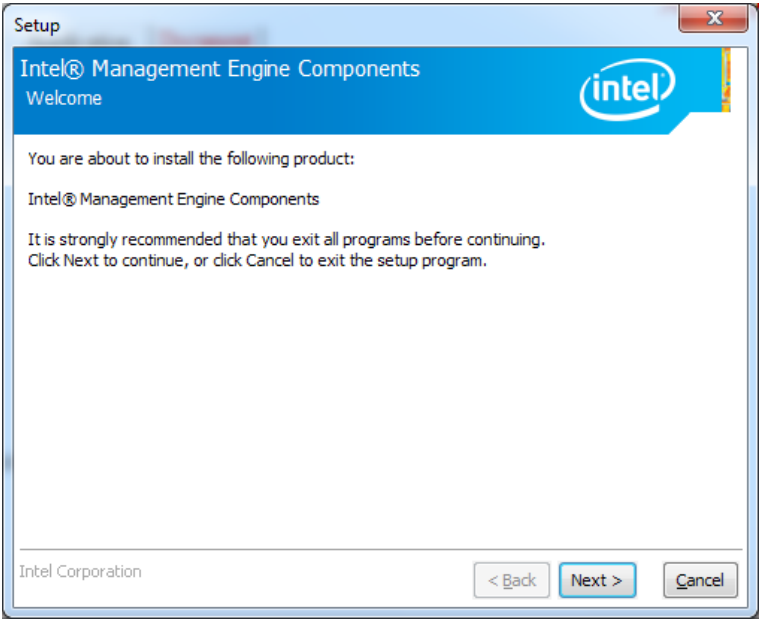
Step 4: Installation complete.



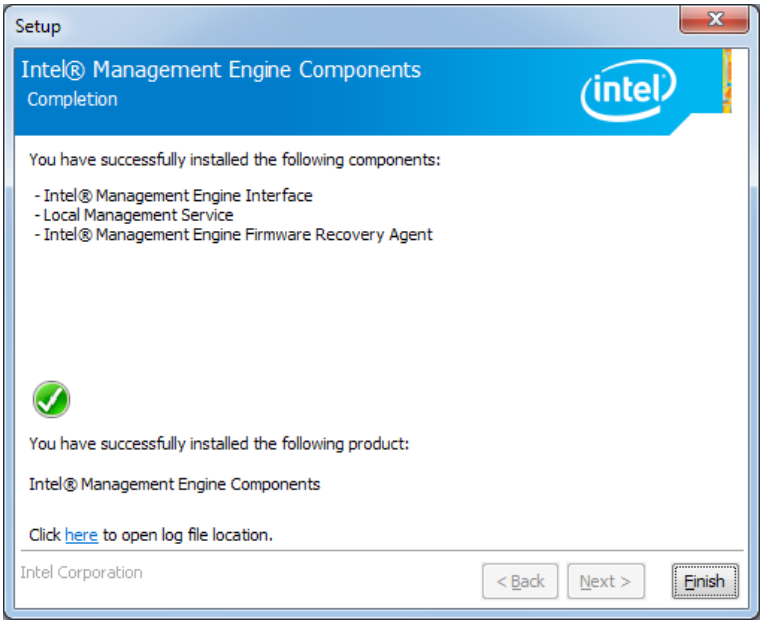
5.3.3. INTEL_MEI

Step 1: Select the "**Application**" tab. Click the "**INTEL_MEI**" box.

Step 2: Click [**Next**] button.

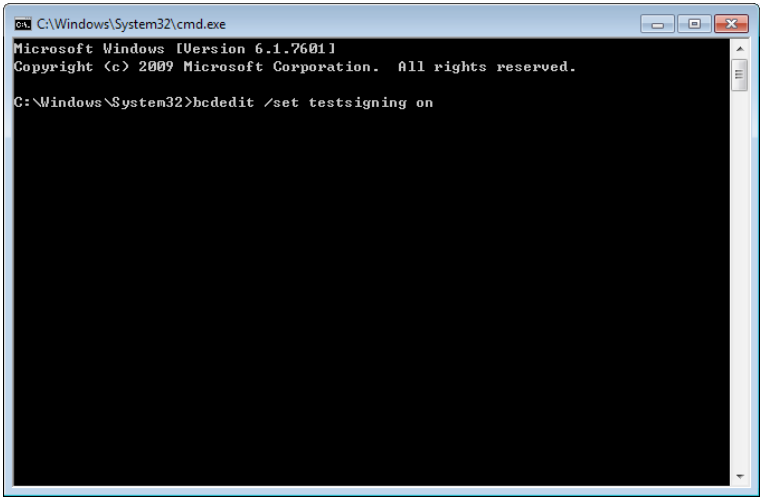


Step 3: Installation complete.



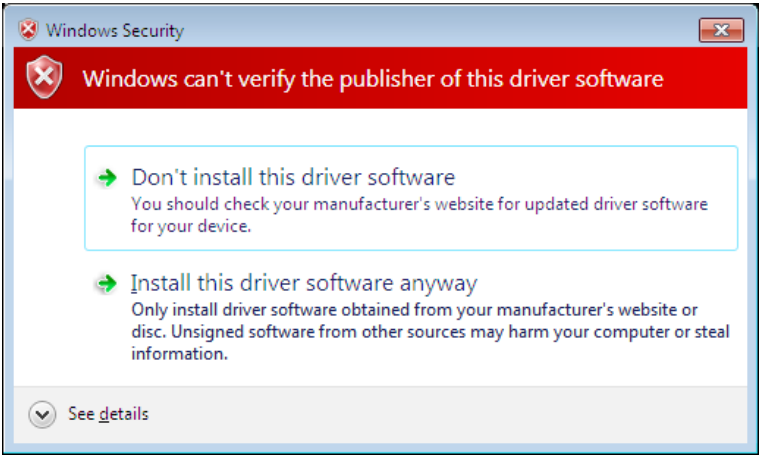
5.3.4. Acrosser Driver

Step 1: To run the Acrosser Driver completely, you should do it at test-signed kernel-mode under Windows 7 x64 by the command “BCDEdit /set testsigning on”.



Step 2: Select the “**Application**” tab. Click the “**Acrosser Driver**” box.

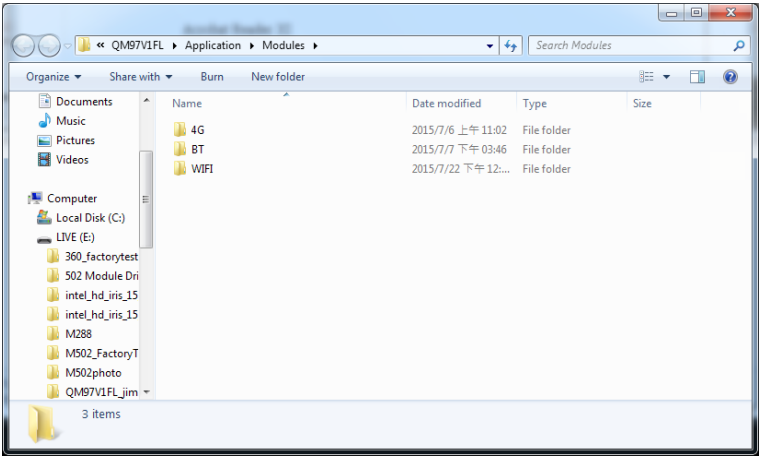
Step 3: If the “**Windows Security**” warning message appears, select “**Install this driver software anyway**” to go on next step.



5.3.5. Drivers for Optional Modules

Step 1: Select the "**Application**" tab. Click the "**Drivers for Optional Modules**" box.

Step 2: Select the driver you want to install.

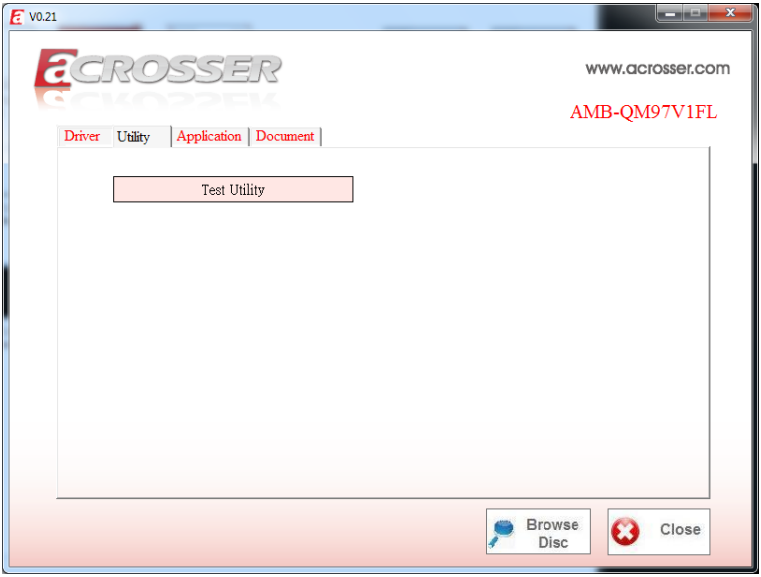


5.4. Utility Installation Page

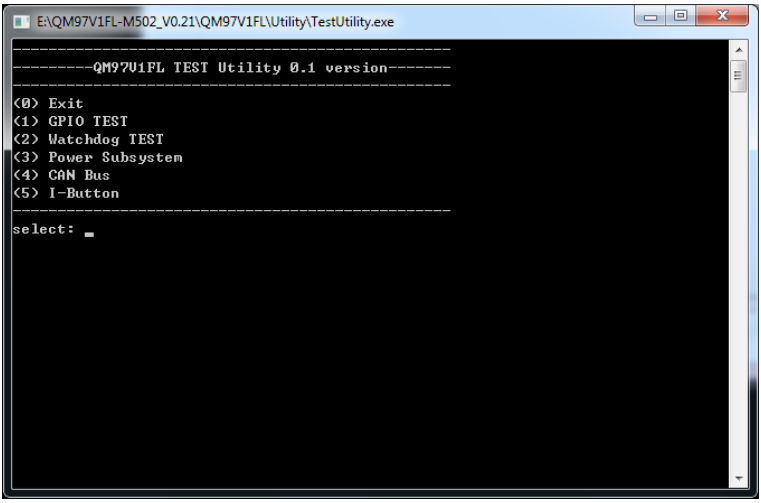
Before launching the utility, you should install "**Driver**" to initiate peripherals, e.g. GPIO and WatchDog.

This "**Test Utility**" can be used to verify both system GPIO and WatchDog features.

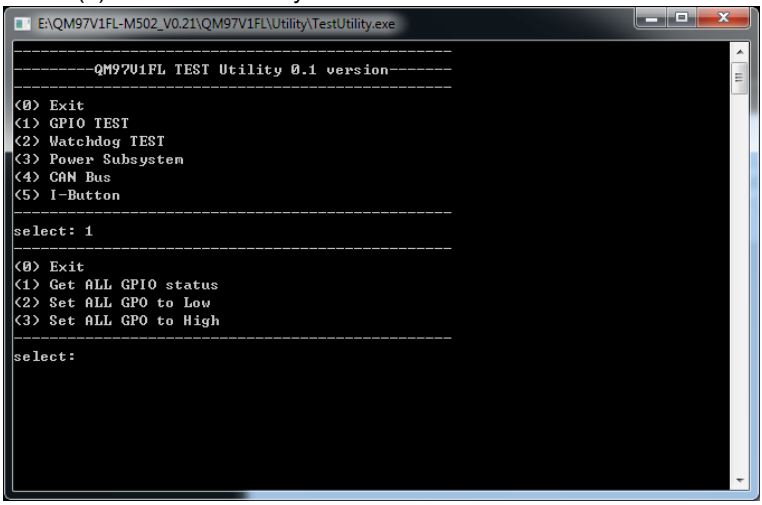
Step 1: Select the "**Utility**" tab. Click the "**Test Utility**" box.



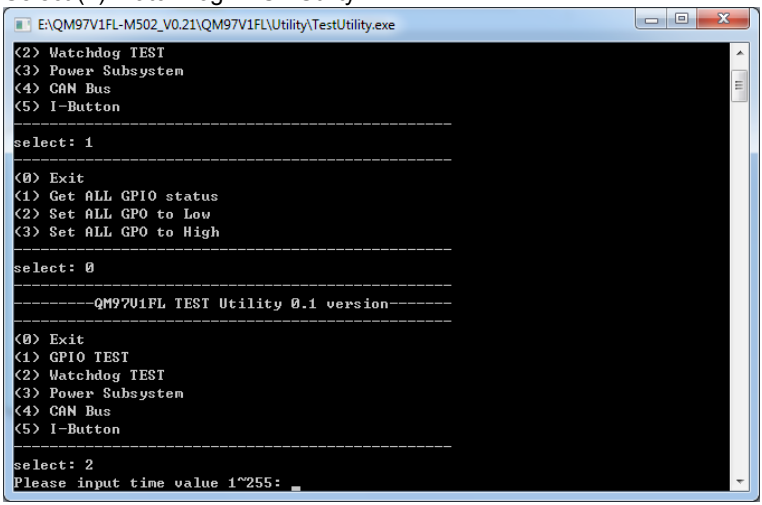
Step 2: The "**Test Utility**" screen appears.



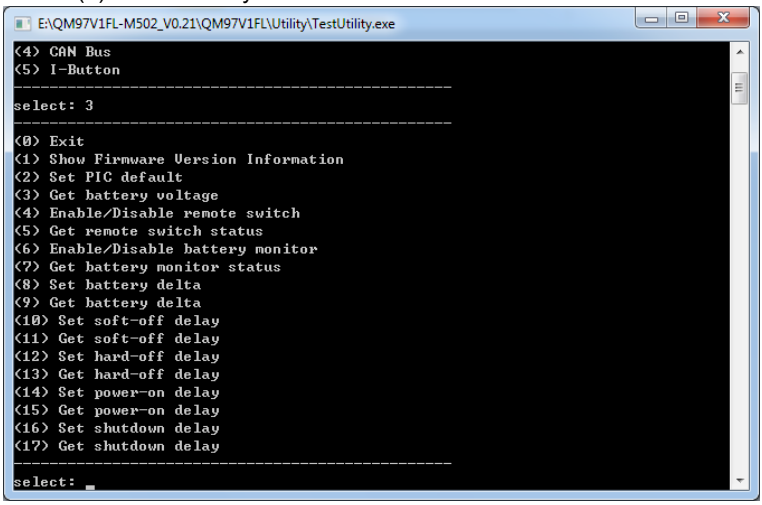
Select (1) GPIO TEST Utility:



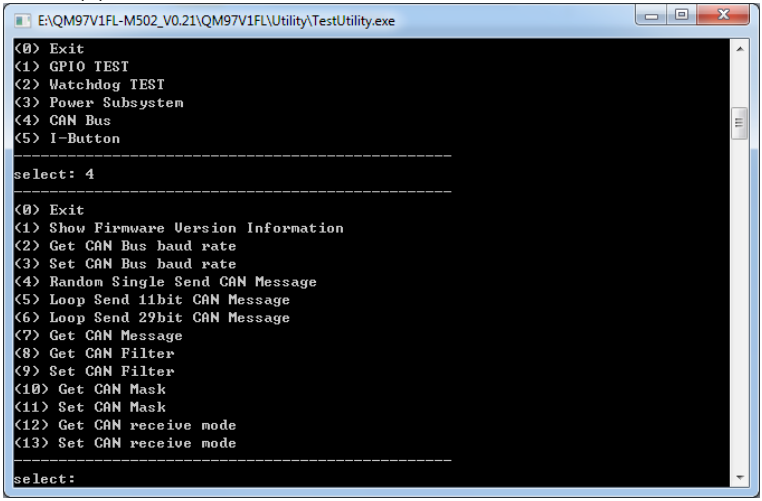
Select (2) WatchDog TEST Utility:



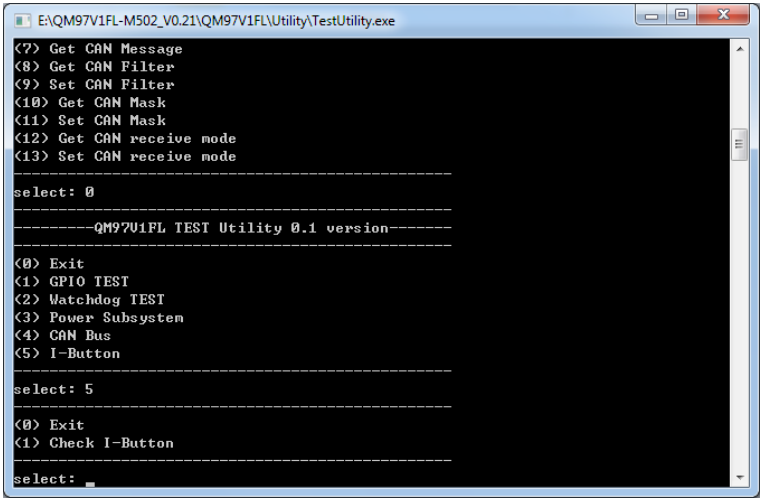
Select (3) Power Subsystem:



Select (4) Can Bus:

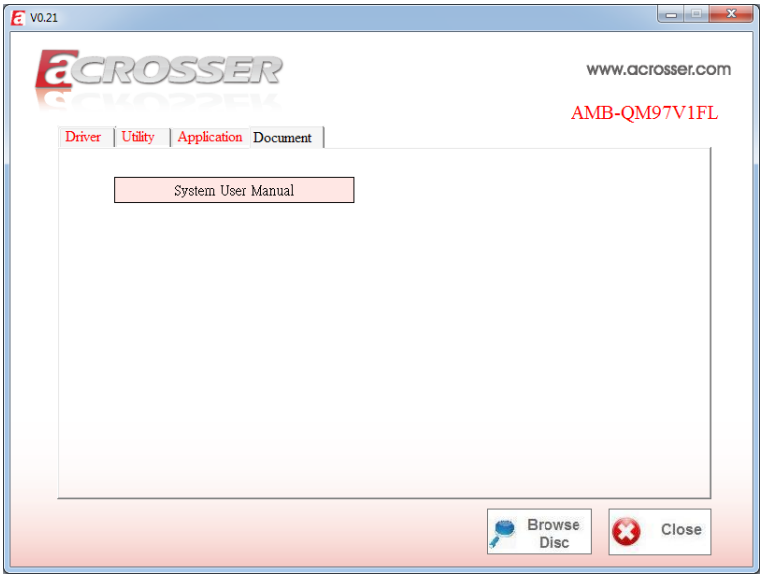


Select (5) I-Button:



5.5. Document Page

The user manual is stored in the "**Document**" folder.



*Note: To read the PDF file, you will have to install "**Acrobat Reader**" first. Please refer to the "**Application Installation Page**".*

6. Software Installation and Programming Guide

6.1. Introduction

6.1.1. CAN Bus

6.1.1.1. Overview

The CAN bus APIs provide interfaces to CAN bus subsystem. By invoking these APIs, programmers can implement the applications which have the functions listed below:

1. Set the BAUD rate.
2. Send the CAN packages over the CAN bus.
3. Receive the CAN packages via the CAN bus hardware interface.
4. Set the CAN package filter to selectively receive CAN packages with specific ID.
5. Set the mask bits to selectively make some filter bits take effect.

In the folder '**QM97V1FL\Utility**' on the CD, we provide:

1. API header file.
2. API library in static library format and shared library format.
3. Test utility.

6.1.1.2. CAN Message Format

```
// TYPE DEFINITION

typedef  char             i8;
typedef  unsigned char    u8;
typedef  short            i16;
typedef  unsigned short   u16;
typedef  unsigned long    u32;
typedef  int              i32;


struct CanMsg {

      u32  id;
      u8   id_type;
      u8   length;
      u8   data[8];
}
```

To transmit a CAN packet, the programmer has to fill in the fields in the variable of type CanMsg and pass this CanMsg variable as an argument to invoke the APIs. The fields in CAN message are described below:

id:

This field holds the ID information of the CAN packet. In a 'Standard Data Frame' CAN packet, the ID field consists of 11 bits of binary digitals. In an 'Extended Data Frame' CAN packet, the ID field consists of 29 bits of binary digitals. That the CAN packet is a 'Standard Data Frame' packet or an 'Extended Data Frame' packet is determined by the 'id_type' field in the CanMsg variable.

The 'id' field in the CanMsg variable is a 32-bit long space. If a CanMsg variable is configured as a 'Standard Data Frame' CAN packet, the bit[0] ~ bit[10] in the 'id' field is the ID of the CAN packet. The bit[11] ~ bit[31] are ignored when the APIs in the library processing the CanMsg variable.

'id' field in the CanMsg variable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1	1	1	0	1	0	1	1

If a CanMsg variable is configured as an 'Extended Data Frame' CAN packet, the bit[0] ~ bit[28] in the 'id' field is the ID of the CAN packet. The bit[29] ~ bit[31] are ignored when the APIs in the library processing the CanMsg variable.

'id' field in the CanMsg variable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	1	0	0	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1	0	1	0	1	1

id_type:

This field identifies that the CAN packet is a 'Standard Data Frame' CAN packet or a 'Extended Data Frame' CAN packet:

```
struct CanMsg canMsg;

canMsg.id_type = EXT_ID;    // A 'Extended Data Frame'
packet

canMsg.id_type = STD_ID;    // A 'Standard Data Frame'
packet
```

length:

This field identifies the number of data bytes in the next field 'data[8]' which are filled with effective data. Because the 'data' field is an 8-byte long array, the range of this field 'length' is 0 ~ 8.

data[8]:

This array of data will be filled with effective data.

For example:

```
struct CanMsg msg;
msg.data[0] = 0xa1;
msg.data[1] = 0xb2;
msg.data[2] = 0xc3;
msg.length = 3;
```

6.1.2. GPIO and Watchdog

6.1.2.1. Overview

This model provides both a GPIO interface and a Watchdog timer. Users can use the GPIO and Watchdog APIs to configure and to access the GPIO interface and the Watchdog timer. The GPIO has four input pins and four output pins. The Watchdog timer can be set to 1~255 seconds. Setting the timer to zero disables the timer. The remaining seconds of the timer to reboot can be read from the timer.

6.1.2.2. Installing Device Driver

Before executing the applications which invoke the GPIO or Watchdog APIs, users should make sure that the Windows device driver has been installed.

On Windows platform, after successfully installing the device driver, there is a device which shows ‘Acrosser Device’ in the ‘Device Manager’. The APIs on Windows platform open this device implicitly.

6.1.3. Power Subsystem

6.1.3.1. Overview

The Power Subsystem APIs can be used to get and set the configuration of power subsystem. By invoking the Power Subsystem APIs, users can:

1. Get the firmware version number of the Power Subsystem.
2. Set all the settings of the Power Subsystem to the default values.
3. Get/Set the status of the remote switch (ENABLE or DISABLE).
4. Get the battery voltage.
5. Get/set the status of the battery monitor (ON or OFF).
6. Get/set the delta value which identifies how much the battery voltage can be lower than the nominal voltage. When the voltage is lower than the tolerable voltage, the power subsystem turns off the system.
7. Get/set the Soft Off deley.
8. Get/set the Hard Off delay.
9. Get/set the Power On delay.
10. Get/set the Shutdown delay.

The power subsystem connects to the main system via the COM port. On the Linux platform, the actual port number to which the Power Subsystem connects is determined by the Linux. The default supported COM interfaces on Linux are COM1 ~ COM4. Users must take extra steps to configure Linux kernel in order to support COM ports which do not fall into the range COM1 ~ COM4. Please refer to Appendix A for more information. Users don't need extraordinary setup on Windows platform to support COM ports.

6.1.4. I-Button Function

In the API library, we provide a set of I-Button functions. Users can use the functions to:

1. Reset the I-Button.
2. Read data from the I-Button.
3. Write data to the I-Button.

6.2. API List and Descriptions

6.2.1. CAN Bus

Syntax:	**i32 getCanFwVer(PicInfo *ver)**
Description:	This function gets the version information of the CAN Bus firmware.

Parameters:	The definition of struct 'PicInfo' is:
	`struct PicInfo {`
	`u8 info[12];`
	`}`
	This API returns the version information and store the information in the memory which is pointed at by the pointer 'ver'.

Return Value:	If this function gets the version information successfully, it returns 0, any other returned value stands for error.

Syntax:	i32 getCanBaudRate(u8 *baud)

Description:

This function gets the current setting of the Baud Rate of the CAN Bus. This function gets an 'unsigned char' to represent the Baud Rate. Here is the table for the Baud Rate:

Unsigned Char	Baud Rate
1	10K
2	20K
3	50K
4	100K
5	125K
6	250K
7	500K
8	800K
9	1000K

Users can use the macros listed below to set the Baud Rate:

```
/* Baud Rate */
#define BAUD_RATE_10K     1
#define BAUD_RATE_20K     2
#define BAUD_RATE_50K     3
#define BAUD_RATE_100K    4
#define BAUD_RATE_125K    5
#define BAUD_RATE_250K    6
#define BAUD_RATE_500K    7
#define BAUD_RATE_800K    8
#define BAUD_RATE_1000K   9
```

Parameters:

This function gets a number which represents the specific Baud Rate and stores it at the memory which is pointed at by the pointer 'baud'.

Return Value:

If this function gets the baud rate successfully, it returns 0, any other returned value stands for error.

Syntax:	i32 setCanBaudRate(u8 baud)
Description:	This function sets the Baud Rate of the CAN Bus.
Parameters:	It takes an 'unsigned char' as the parameter and sets the Baud Rate according to the value stored at the parameter 'baud'. The correspondence between the Baud rate and the value to set to the function is the same as the table listed in the previous API 'getCanBaudRate( )'
Return Value:	If this function sets the baud rate successfully, it returns 0, any other returned value stands for error.

Syntax:	i32 sendCanMessage(struct CanMsg *buffer, u8 count)
Description:	This function sends out CAN packages over the CAN bus.
Parameters:	If there is more than one CAN packet to send, these CAN packages are stored in an array of type 'CanMsg'. This function sends out packets in a sequential fashion. The memory address of the first CAN packet to be sent is pointed at by the parameter 'buffer'. The number of CAN packets to be sent is indicated by the parameter 'count'.

Return Value:	If this function sends the CAN packet successfully, it returns 0, any other returned value stands for error.
	Here is an example:
	If the CAN packets in the array 'canAry[]' have been initialized. The code listed below will send out the CAN packets in the 'canAry[]' over the CAN bus.
	`unsigned int result = 0;`
	`struct CanMsg canAry[30];`
	`/* …`
	`Initialize the CAN packages in the canAry[30]`
	`*/`
	`result = sendCanMessages(canAry, 30);`
	`if(result != 0)`
	`fprintf(stderr, "Send CAN package error!\n");`

Syntax:	i32 getCanMessage(struct CanMsg *buffer, u8 count)
Description:	This function receives CAN packets from the CAN bus subsystem.
Parameters:	This function stores received CAN packages sequentially at an array of type 'CanMsg'. The number of packages to receive is indicated by the parameter 'count'.

Return Value:	If this function receives the CAN packet successfully, it returns 0, any other returned value stands for error.
	Here is an example:
	If the array 'canAry[]' of type 'CanMsg' has been declared and allocated. The code listed below will receive 30 CAN packages from the CAN bus subsystem and stores the packages in the 'canAry[]'.
	`unsigned int result = 0;`
	`struct CanMsg canAry[30];`
	`result = getCanMessage(canAry, 30);`
	`if(result != 0)`
	`fprintf(stderr, "Fail to receive CAN packets!\n");`

Syntax:	i32 getCanMask(struct CanMask *mask)

Description:

This function gets the current setting of the acceptance masks. Masks are used to determine which bits in the ID field of the CAN packet are examined with the filters. There are two acceptance masks (mask0 and mask1) and six acceptance filters (filter0 ~ filter5) in the CAN Bus subsystem. Filter0 ~ filter1 are associated with mask0. Filter2 ~ filter4 are associated with mask1.

Here is the Mask/Filter truth table:

Mask bit n	Filter bit n	Message ID bit n	Accept or reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: x = don't care

Parameters:

This parameter 'mask' is a pointer to a variable of type 'CanMask'. Users use the field 'maskId' to indicate the mask they want and the API put the setting of the mask in the 'mask' field.

```
struct CanMask {
u8 maskId; // 0 or 1
u32 mask;
}
```

Return Value:

If this function receives the mask setting successfully, it returns 0, any other returned value stands for error.

For example:

```
struct CanMask a_mask;
a_mask.maskId = 0;  // indicate the
mask0
i32 result;
result = getCanMask(&a_mask);
// The setting of the mask is put at
// a_mask.mask
if(result != 0)
printf("Fail to get mask!\n");
```

Syntax:	i32 setCanMask(struct CanMask mask)
Description:	This function sets the bit patterns to the indicated mask. The target mask is indicated by the 'maskId' field in a CanMask variable.

Parameters:	This functions takes a variable of type 'CanMask'. User set the bit patterns they want to the 'mask' field in a 'CanMask' variable.
	`struct CanMask {`
	`u8 maskId;  // 0 or 1`
	`u32 mask;`
	`}`
	For example:
	`struct CanMask varMask;`
	`i32 result;`
	`varMask.maskId = 1;`
	`varMask.mask = 0x12345678;`
	`result = setCanMask(varMask);`

Return Value:	If this function sets the mask setting successfully, it returns 0, any other returned value stands for error.

Syntax:	i32 getCanFilter(struct CanFilter *varFilter)
Description:	This function gets the current setting of the acceptance filter. Use the 'filterId' field in a 'CanFilter' variable to indicate the filter you want and the API puts the setting of the indicated filter in the 'filter' field in the CanFilter variable 'varFilter'.

Parameters:	This function takes a pointer to a 'CanFilter' type variable.
	For example:
	`struct CanFilter varFilter;`
	`i32 result;`
	`result = getCanFilter(&varFilter);`
	`if(result != 0)`
	`printf("Fail to get the filter!\n");`

Return Value:	If this function gets the filter successfully, it returns 0, any other returned value stands for error.

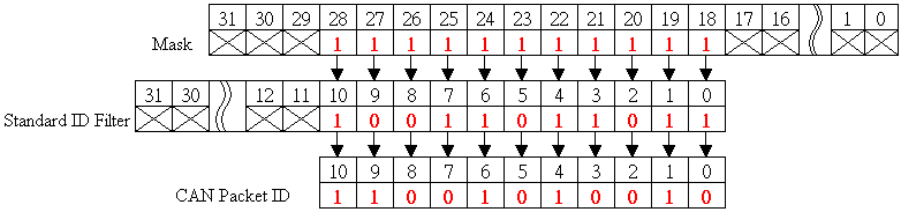
Syntax:	i32 setCanFilter(struct CanFilter *varFilter)

Description:

This function sets the bit pattern to the filter. By indicating the 'filterType' field in the 'varFilter' variable, the bit pattern in the 'filter' field will be taken as an 'Standard ID' filter or 'Extended ID' filter.

```
struct CanFilter {

u8  filterId;  // There are six filters so
the filterId = 0 ~ 5

u8  filterType;  // filterType = STD_ID or
filterType = EXT_ID

u32 filter;

}
```

If a filter is configured as a 'Standard ID' filter, only bit18 ~ bit28 in the mask take effect when filtering the CAN packet.



Parameters:	This function takes a pointer to a variable of type 'CanFilter' as the parameter. Users set up the 'filterId'. There are six filters so the 'filterId' could be 0 ~ 5. Filter0 and filter1 are associated with mask0. Filter2 ~ filter5 are associated with mask1.
	By setting up 'filterType', users indicate the type of the filter. Filter type could be 'STD_ID' or 'EXT_ID'.
	Depending on the filter type, the 'filter' field in the CanFilter variable could be 0x0 ~ 0x7FF (11 bits) when filter type is 'STD_ID'. If the filter type is 'EXT_ID', the 'filter' field in the CanFilter variable could be 0x0 ~ 0x1FFFFFFF (29 bits).
	For example:
	`struct CanFilter varFilter;`
	`i32 result;`
	`varFilter.filterId = 3;`
	`varFilter.filterType = STD_ID;`
	`varFilter.filter = 0x555;`
	`result = setCanFilter(&varFilter);`
	`if(result != 0)`
	`printf("Fail to set up the filter!\n");`

Return Value:	If this function sets the filter successfully, it returns 0, any other returned value stands for error.

6.2.2. GPIO and Watchdog

6.2.2.1. GPIO

Syntax:	**int get_gpo_status(int pin)**
Description:	Get the status of GPIO output pins.
Parameters:	This function fills in an integer variable as the parameter. The pin0 ~ pin3 is the status of the output pins.
Return Value:	0 or 1 (0 is Low, 1 is High)

Syntax:	int get_gpi_status(int pin)
Description:	Set the status of GPIO input pins.
Parameters:	This function fills in an integer variable as the parameter. The pin4 ~ pin7 is the status of the input pins.
Return Value:	0 or 1 (0 is Low, 1 is High)

Syntax:	void set_gpo_status(int pin, int value)
Description:	Set the status of GPIO output pins and value.
Parameters:	Set pin0 ~ pin3 value 0 is Low, 1 is High.
Return Value:	None.

6.2.2.2. Watchdog

Windows Platform:

Syntax:	u8 getWtdTimer(void)
Description:	This function read the value of the watchdog time counter and returns it to the caller.
Parameters:	None.
Return Value:	This function returns the value of the time counter and returns it to the caller as an integer.

Syntax:	void setWtdTimer(u8 val)
Description:	This function sets the watchdog timer register to the value ‘val’ and starts to count down. The value could be 0 ~ 255. The unit is second. Setting the timer register to 0 disables the watchdog function and stops the countdown.
Parameters:	The parameter ‘val’ is the value to set to watchdog timer register. The range is 0 ~ 255.
Return Value:	None.

Linux Platform:

Syntax:	void wdt_start(int _timevalue)
Description:	This function sets the watchdog timer register to the value '_timevalue' and starts to count down. The value could be 0 ~ 255. The unit is second. Setting the timer register to 0 disables the watchdog function and stops the countdown.
Parameters:	The parameter '_timevalue' is the value to set to watchdog timer register. The range is 0 ~ 255.
Return Value:	None.

Syntax:	void wdt_stop(void)
Description:	This function sets the watchdog timer stop.
Parameters:	None.
Return Value:	None.

Syntax:	int get_wdt_count(void)
Description:	This function read the value of the watchdog time counter and returns it to the caller.
Parameters:	None.
Return Value:	This function returns the value of the time counter and returns it to the caller as an integer.

6.2.3. Power Subsystem

Syntax:	i32 getPwrFwVer(struct PicInfo *ver)
Description:	This function gets the version information of the firmware of the Power Subsystem.

Parameters:	The definition of struct 'PicInfo' is:
	`struct PicInfo {`
	`u8 info[12];`
	`}`
	This API returns the version information and store the information in the memory which is pointed at by the pointer 'ver'.

Syntax:	i32 setPicDefault(void)

Description:	The function restores the Power Subsystem to the default values. After calling this API, the items listed below are restored to its default value:
	`Remote Switch → Default: Disabled`
	`Battery Monitor → Default: Disabled`
	`Battery Voltage Delta Value → Default: 1.5V`
	`System Soft Off Delay → Default: 5 seconds`
	`System Hard Off Delay → Default: 1 minute`
	`System Power On Delay → Default: 8 seconds`
	`OS Shutdown Delay → Default: 3 minutes`

Parameters:	None.
Return Value:	If this function works successfully, the function will return 0, any other value standards for error.

Syntax:	i32 getRemoteSwitch(u8 *val)
Description:	The function gets the status of the Remote Switch.

Parameters:	This function takes a pointer to an unsigned char variable as the parameter. After calling this function, the status of the Remote Switch will be put at the memory which is pointed by the parameter 'val'. If the Remote Switch is enabled, '*val' is 0x5A. If the Remote Switch is disabled, the '*val' is 0xA5. Users can use the macros 'ENABLED' (0x5A) and 'DISABLED'(0xA5) to test the status value '*val'.
	For example:
	`u8 val;`
	`i32 result;`
	`result = getRemoteSwitch(&val);`
	`if(result == 0) {`
	`if(val == ENABLED)`
	`printf("Remote Switch is enabled.\n");`
	`else if(val == DISABLED)`
	`printf("Remote Switch is disabled.\n");`
	`}`

Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 setRemoteSwitch(u8 val)
Description:	The function sets the status of the Remote Switch.
Parameters:	This function takes an unsigned char as the parameter. The value of this parameter can be 'ENABLED' (0x5A) or 'DISABLED'(0xA5).
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 getBattValt(float *vol)
Description:	This function gets the battery voltage ant put it in the memory which is pointed at by the pointer 'vol'.
Parameters:	This function takes a pointer to a 'float' variable as the parameter. The reading of the battery voltage is put at the memory which is pointed at by the parameter 'vol'.
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 getBattMonitor(u8 *val)
Description:	The function gets the status of the Battery Monitor.
Parameters:	This function takes a pointer to an unsigned char variable as the parameter. After calling this function, the status of the Battery Monitor will be put at the memory which is pointed by the parameter 'val'. If the Battery Monitor is enabled, '*val' is 0x5A. If the Battery Monitor is disabled, the '*val' is 0xA5. Users can use the macros 'ENABLED' (0x5A) and 'DISABLED'(0xA5) to test the status value '*val'.
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 setBattMonitor(u8 val)
Description:	The function sets the status of the Battery Monitor.
Parameters:	This function takes an unsigned char as the parameter. The value of this parameter can be 'ENABLED' (0x5A) or 'DISABLED'(0xA5).
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 getBattDelta(float *val)
Description:	This function gets the delta value. The delta value is the maximum voltage deviation of the power from its nominal voltage. If the function of Battery Monitor is ON, the Power Subsystem shuts the system down when the voltage deviation of the power is larger than the delta value.
Parameters:	This function takes a pointer to a float variable as the parameter. The delta value will be put at the memory which is pointed by the parameter 'val'.
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 setBattDelta(float val)
Description:	This function sets the voltage delta value. The range is 0.5V ~ 3.0V. The granularity is 0.5V.
Parameters:	This function takes a float variable as the parameter.
Return Value:	If this function works successfully, it returns 0, any other value standards for error.

Syntax:	i32 setSoftOffDelay(u32 setTime)
Description:	The Soft Off Delay is the interval between that the system receives a power off signal and that the system generates a power off signal. This function sets up the interval in seconds.
Parameters:	The parameter is of the type of unsigned long. The value of the parameter ranges from 3~3600. The unit of the value of the parameter is seconds.
Return Value:	If this function works successfully, it returns 0, any other value stands for error.

Syntax:	i32 setHardOffDelay(u32 setTime)
Description:	The Hard Off Delay is the interval between that the system is off and that the power 5VSB is off. This functions set up the interval in seconds.
Parameters:	The parameter is of the type of unsigned long. The value of the parameter ranges from 3~3600. The unit of the value of the parameter is seconds.
Return Value:	If the function works successfully, it returns 0, any other value stands for error.

Syntax:	i32 getSoftOffDelay(u32 *Time)
Description:	The Soft Off Delay is the interval between that the system receives a power off signal and that the system generates a power off signal. This function gets the interval.
Parameters:	The parameter is a pointer which points to an unsigned long variable. The returned value is stored at this variable. The unit of the returned value is in seconds.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 getHardOffDelay(u32 *Time)
Description:	The Hard Off Delay is the interval between that the system is off and that the power 5VSB is off. This function gets the interval.
Parameters:	The parameter is a pointer which points to an unsigned long variable. The returned value is stored at this variable. The unit of the returned value is in seconds.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 getPowerOnDelay(u32 *val)
Description:	This function gets the Power On delay.
Parameters:	This function takes a pointer to an unsigned long variable as the parameter. The delay time will be put at the memory which is pointed by the 'val'.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 setPowerOnDelay(u32 val)
Description:	This function sets the Power On delay.
Parameters:	This function takes an unsigned long variable as the parameter. The range of the Power On delay is 8 ~ 60 seconds.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 getShutdownDelay(u32 *val)
Description:	This function gets the Shutdown delay.
Parameters:	This function takes a pointer to an unsigned long variable as the parameter. The delay time will be put at the memory which is pointed by the parameter 'val'.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 setShutdownDelay(u32 val)
Description:	This function sets the Shutdown delay.
Parameters:	This function takes an unsigned long variable as the parameter. The range of the delay is 120 ~ 3600 seconds.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

6.2.4. I-Button

Syntax:	i32 resetIbutt(void)
Description:	This function resets the I-Button.
Parameters:	None.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 readIbutt(u8 *data)
Description:	This function reads data from the I-Button.
Parameters:	This function takes a pointer to an unsigned char variable. The data to be read from the I-Button is put at the memory which is pointed by the parameter ‘data’.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

Syntax:	i32 writeIbutt(u8 data)
Description:	This function writes command to the I-Button.
Parameters:	This function takes an unsigned char variable as the parameter. The command to be written to the I-Button is the value of the parameter ‘data’.
Return Value:	If this function works successfully, the function returns 0, any other value stands for error.

6.3. Appendix A

Users have to modify the boot loader configuration to support COM6. Take the grub configuration file as an example. Add '8250.nr_uarts=XX noirqdebug' at the setting of kernel. Here, XX represents the number of COM ports the system will support. Because the power subsystem connects to main system via COM6, the XX must be greater or equal to 6.

1. Modify the grub.conf.

```
[root@linux ~]# vi /boot/grub/grub.conf
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Fedora Core (2.6.27.5.117.FC10)
root (hd0,0)
kernel /vmlinuz-2.6.27.5.117.FC10 ro root=/dev/hda2 rhgb
quiet
8250.nr_uarts=6 noirqdebug
initrd /initrd-2.6.27.5.117.FC10.img
```

2. List the status of the COM ports in the system.

```
# setserial -g /dev/ttyS*
# setserial -g /dev/ttyS*
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
/dev/ttyS2, UART: 16550A, Port: 0x03e8, IRQ: 11
/dev/ttyS3, UART: 16550A, Port: 0x02e8, IRQ: 10
/dev/ttyS4, UART: 16550A, Port: 0x04f8, IRQ: 11
/dev/ttyS5, UART: 16550A, Port: 0x04e8, IRQ: 10
```

The node '/dev/ttyS5' corresponds to COM6. The IO port is 0x4e8, IRQ 10.

7. FAQ

Q 1. Does my system support any other OS?

- Please check with Acrosser local sales representative or authorized channels to help you confirm whether a new OS driver is provided.

Q 2. What if the screen blacked out when installing Linux?

- Check the screen cable connection. Reboot the system.
- Check the screen cable connection. Key in **CTRL+ALT+F2** to enter Terminal screen. Log in and then input startx.

Q 3. What should I do if the ubuntu 14.10 cannot be installed correctly?

- Install the ubuntu14.04 first, and then upgrade to ubuntu14.10 to fix this situation.

Q 4. What if the bluetooth device cannot work correctly in Linux?

- Install the "**blueman and bluemon tool**" to fix this situation.

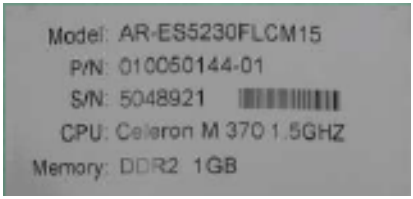
Q 5. How to install Sierra EM7305 EM7355 module driver under Linux?

- Please follow these steps:

  Step 1: Open terminal
  Step 2: su
  Step 3: gedit /etc/modprobe.d/blacklist.qc.conf
  Step 4: add "blacklist qcserial" to /etc/modprobe.d/blacklist.qc.conf
  Step 5: add "blacklist qmi_wwan" to /etc/modprobe.d/blacklist.qc.conf
  Step 6: save /etc/modprobe.d/blacklist.qc.conf
  Step 7: gedit /etc/sysconfig/grup
  Step 8: find GRUB_CMDLINE_LINUX=
  Step 9: GRUB_CMDLINE_LINUX= add rd.driver.blacklist=qmi_wwan
  Step 10: save /etc/sysconfig/grup
  Step 11: grub2-mkconfig -o /boot/grub2/grub.cfg
  Step 12: cd GobiNet
  Step 13: make
  Step 14: make install
  Step 15: cd GobiSerial
  Step 16: make
  Step 17: make install
  Step 18: reboot
  Step 19: use nm-connection-editor set SIM Card Pin Code
  Step 20: Use Default Mobile broadbad AP

Q 6. Where is the serial number located on my system?

- The serial number (S/N) is an alpha-numeric character located on the bottom or side chassis.



(for reference only)

Technical Support Form

We deeply appreciate your purchase of Acrosser products. Please find the "**tech_form.doc**" file in our utility CD. If you have any questions or problems about Acrosser products, please fill in the following information. We will answer your questions in the shortest time possible.

Describe Your Info and Acrosser System Info

- Your Company Name: _____
- Your Contact Info: _____ Phone Number: _____
- Your E-Mail Address: _____
- Your Company Address: _____

  _____
- Acrosser Model Name: _____
- Acrosser Serial Number: _____

Describe System Configuration

- CPU Type: _____
- Memory Size: _____
- Storage Device (e.g. HDD, CF, or SSD): _____
- Additional Peripherals (e.g. Graphic Card): _____
- Operating System & Version (e.g. Windows 7 Embedded): _____
- Special API or Driver: _____

  (If yes, please provide it for debug.)
- Running Applications: _____
- Others: _____

Describe Your Problems or Questions:

Send the above information to one of the following Acrosser contacts:

- Acrosser Local Sales Representative
- Acrosser Authorized Sales Channels
- Acrosser Inquiry --- http://www.acrosser.com/inquiry.html
- Acrosser FAX Number --- 886-2-29992887

To Make Your
Embedded
Idea a Reality

Acrosser Headquarters

241新北市三重區光復路一段61巷26號10樓
10F., No.26, Ln. 61, Sec. 1, Guangfu Rd.,
Sanchong Dist., New Taipei City 241, Taiwan
(R.O.C.)
TEL: +886-2-29999000
FAX: +886-2-29992887 / +886-2-29993960

Acrosser Taichung Office

414台中市烏日區僑仁街8號10樓之1
10F.-1, No.8, Qiaoren St., Wuri Dist.,
Taichung City 414, Taiwan (R.O.C.)
TEL: +886-4-2337-0715
FAX: +886-4-2337-3422

Acrosser China Subsidiary

深圳市欣扬通电子有限公司
深圳市福田区车公庙泰然九路21号
皇冠科技园3栋2楼 (邮编: 518040)
2F., 3rd Building, Crown Science Park, No. 21,
Tai-Ran 9th Rd., Che Gong Miao, Futian Dist.,
Shenzhen, China (Postal: 518040)
TEL: +86-755-83542210
FAX :+86-755-83700087

Acrosser Nanjing Office

欣扬通电子有限公司 南京办事处
江苏省南京市江宁区天元东路228号504室
(邮编: 211100)
Room 504, No. 228, Tian Yuan East Rd., Jiang
Ning Dist., Nanjing City, Jiangsu Province, China
(Postal: 211100)
Mobile: 13611932003
TEL: +86-025-86137002
FAX: +86-025-86137003

Acrosser Beijing Office

欣扬通电子有限公司 北京办事处
北京市昌平区沙河镇沙阳路巩华新村8号楼2单元
1403室 (邮编: 102206)
Room 1403, Unit 2, Building 8, Gonghua Village,
Shahe Town, Changping District, Beijing, China
(Postal: 102206)
Mobile: 13311317329

Acrosser USA Inc.

11235 Knott Ave. Suite A, Cypress, CA 90630, USA
Toll Free: +1-866-401-9463
TEL: +1-714-903-1760
FAX: +1-714-903-5629