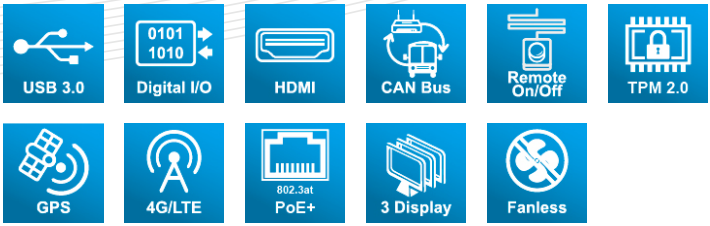
AIV-Q170V1FLS
AIV-Q170V1FLS-OBD
AIV-Q170V1FL
AIV-Q170V1FL-OBD

Fanless In-Vehicle System
Skylake S+TE Ver CPU



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: 100-004
Date: Nov. 18, 2019

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

1.1. Introduction

This is the latest in-vehicle computer of Acrosser Technology. Equipped with generation 6 Intel Skylake-Core i series CPU, fanless design, and rich I/O ports, this brand new AIV-Q170V1FL meets your various requirements, perfect for all kinds of telematics solutions & fleet management, no matter in tracking application or management application.

1.2. Specifications

System

CPU	• Intel® Skylake-S 6th Gen Core™ i7-6700 TE • Intel® Skylake-S 6th Gen Core™ i5-6500 TE • Intel® Skylake-S 6th Gen Core™ i3-6100 TE • Intel® Skylake-S 6th Gen Pentium™ G4400 TE
Chipset	• Intel® Q170
Memory	• 2x DDR4 SO-DIMM- 2133 (Up to 32GB/non-ECC)

Display

Graphic Controller	• Intel® HD Graphics 530 (i3/i5/i7) • Intel® HD Graphics 510 (Celeron/Pentium)
Video Interface	• 1x HDMI (with locking bracket) • 1x VGA • 1x DVI-D

Storage

SATA	• 2x SATA 3 Connectors (Sata 3 signal) • 2x Power Connectors (JST 2.54mm, 1x4 pin)
Disk Bay	• 2x Swappable 2.5” HDD Bay (with Anti-vibration)
CFast	• 1x CFast Socket

Communication and I/O

Ethernet	• 2x GbE Copper (RJ45)+ 4x GbE Copper (RJ45) • Intel i210 IT chip
I2C	• I2C Pin Header

G Sensor
- 1x G Sensor board connect to I2C Pin Header (3-axis accelerometer)

USB
- 8x External Connectors for USB 3.0

Serial Port
- 4x COM --> DB9 (RS232/422/485)

CANBUS
- Use CAN/OBD II DB9 connection
  1. Support CAN Bus 2.0B protocol
  2. J1939 protocol (Colay with STN1110 IC)(Option)
  3. Programmable baud rate:

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

  4. API library for user development
  5. CAN bus device status query

CAN/OBD II
- Colay STN1110 with CAN BUS

GPIO
- GPIO 4-in / 4-out, DB15 male
- Digital Input
  Input Channels: DI x 4
  Input Voltage: 0 to 28 VDC
  Digital Input Levels for Dry Contacts:
  • Logic level 0: Close to GND
  • Logic level 1: Open
  Digital Input Levels for Wet Contacts:
  • Logic level 0: +3 V max.
  • Logic level 1: +10 V to +30 V (COM to DI)
  Isolation: 2 kV optical isolation
- Digital Output
  Output Sink to ground current: maximum 50mA per channel
  Output Channels: DO x 4, sink type
  On-State Voltage: 24 VDC nominal, open collector to 30 V

POE
- 4x PoE (PD mode, 802.3 af compliant) Via PoE LAN1~4

SIM
- 2x SIM card socket

LED
- 1x3 LED for power & status (onboard)

Expansion

Mini PCIe Slot	**3x Mini PCI-e socket** • Mini PCI-e 1 for 4G& GPS(PCI-e+USB signal)(Full size) S Model Mini PCI-e 1:USB + USB 3.0 signal(By BIOS) • Mini-PCI-e 2 for Wi-Fi + BT(PCI-e+USB signal)(Half size) • Mini-PCI-e 3 for reserved(PCI-e+USB signal) (Full size) S Model Mini PCI-e 3:USB + USB 3.0 signal(By BIOS)

Other Features

Audio	• 2x 3.5” Phone Jack: Pink: MIC in Green: Audio out
Remote Switch	• 1x 3.5” Phone Jack (Blue)
CMOS	• RTC (+/- 2 seconds for 24hours) • Lithium Battery (3V) for CMOS data backup
Hardware Monitoring	• CPU Voltage • CPU and System Temperature
Watchdog Timer	• Software Programmable 0~255 Seconds, 0= disable timer.

Antenna

Antenna type	• 2x 2.4G External Antenna for TX/RX, Diversity (For WiFi & BT) • 1x 4G/LTE U.FL Antenna (Diversity, MIMO) • 1x GPS /Glonass Antenna U.FL Antenna

Power Requirement

Power Supply	• 9V ~ 36V Power Input • 12V for System, 54V for POE function

Software

OS Support	• Windows 10 (64 bit) • Linux Kernel 4.4 or above (64 bit)

Mechanical & Environment

Thermal Design	• Fanless (Heatsink)
Chassis	• Aluminum extrusion heat sink & metal chassis (Silver printing color with Acrosser Logo)
Dimension	• 290mm (W) x 190mm(D) x 59.7mm(H)
Vibration	• IEC 60068-2-64, 5~500Hz, 3GRMS (CFast/SSD)
Shock	• IEC 60068-2-27, 50G 500m/s2 11MS
Operating Temperature	• -25°C ~ 60°C • -25°C ~ 55°C (+15°C) follow EN50155 T1
Storage Temperature	• -40°C ~ 80°C
Humidity	• 0 ~ 90%
Certification	• CE / FCC Class B / E Mark

1.3. Package Contents

Check if the following items are included in the package.

	Item	Q'ty
☐	AIV-Q170V1FL System	1
☐	Remote Switch Cable	1
☐	Driver CD	1
☐	Screw Pack (For 2.5" HDD bracket: 8pcs)	1
☐	Terminal Block (Female 3-pin)	1
☐	Spare Fuse	1
☐	HDMI Locking Bracket	1
☐	GPIO Cable	1

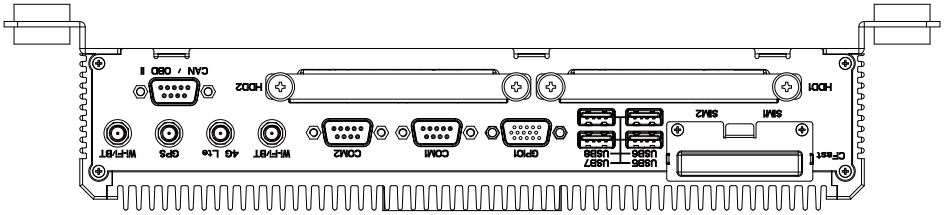
1.4. PCIe Table

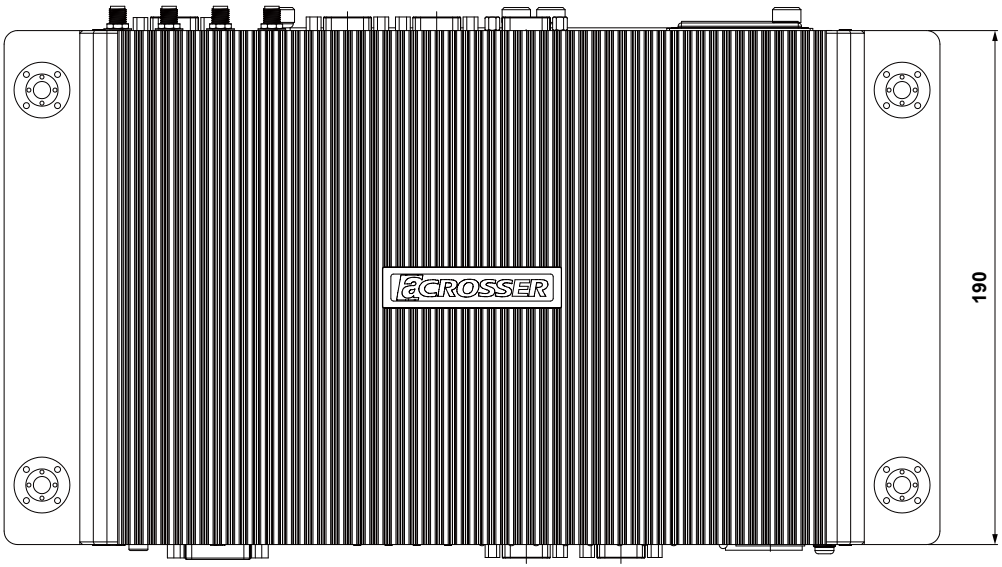
PCIe Option	Signal Source	Supporting Module
Mini-PCIe 1	USB 2.0 + USB 3.0	Support SIERRA 4G Module(SIM1) only
Mini-PCIe 2	USB 2.0 + PCI-e	Support Quectel 4G Module(SIM2), Wi-Fi & BT, UDR/GPS
Mini-PCIe 3	USB 2.0 + PCI-e	Support Wi-Fi & BT, UDR/GPS

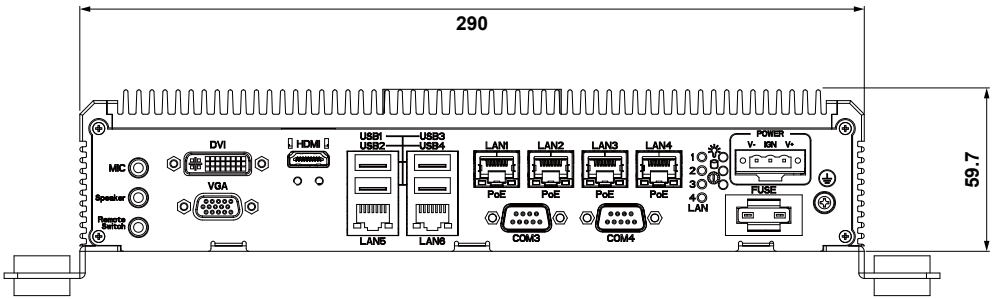
1.5. System Dissection

1.5.1. Dimensions

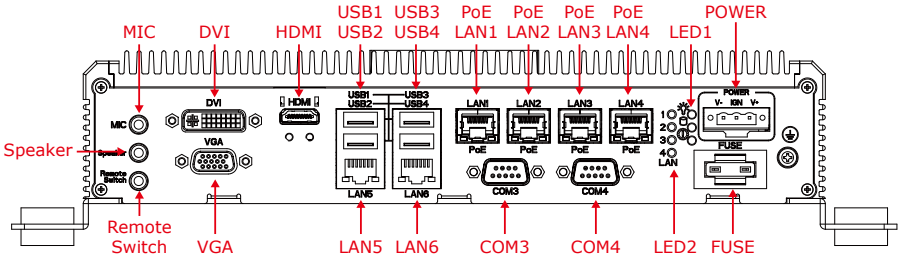
(Unit: mm)



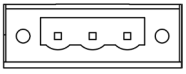




1.5.2. Front I/O Panel

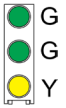


POWER **9V ~ 36V DC Jack**



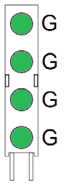
Pin #	Signal
V+	9V ~ 36V
IGN	IGN_ON
V-	GND

LED1 **Status/HDD/Power LED Indicator**



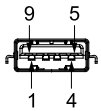
LED	Light	Display
G	Green	Status
G	Green	SATA Device Activity
Y	Yellow	Power LED

LED2 **PoE LAN Power LED Indicator**



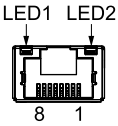
LED	Light	Display
G	Green	PoE LAN4 POWER
G	Green	PoE LAN3 POWER
G	Green	PoE LAN2 POWER
G	Green	PoE LAN1 POWER

USB1 ~ USB4 USB 3.0 Connector



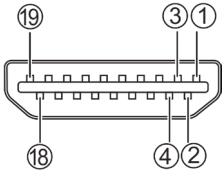
Pin #	Signal	Pin #	Signal
1	VCC5	5	SS_RX -
2	DATA-	6	SS_RX +
3	DATA+	7	GND
4	GND	8	SS_TX -
		9	SS_TX +

LAN1 ~ LAN6 RJ-45 GbE Port



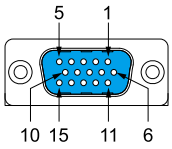
LED	Light	Status
LED1	Off	10Mbps or No Link
	Green	100Mbps
	Orange	1000Mbps
	Blink	NA
LED2	Yellow	Link
	Blink	Link with Activity
	Off	No Link

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	CAN_L
11	GND	12	
13	NC	14	NC
15	DDCCL	16	DDCDA
17	GND	18	+5V
19	HPD		

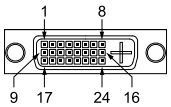
VGA **VGA Connector**



Pin #	Signal	Pin #	Signal
1	VGA_RED	2	VGA_GREEN
3	VGA_BLUE	4	NC
5	GND	6	GND
7	NC	8	GND
9	VCC5	10	GND
11	NC	12	VGA_SDA
13	VGA_HSYNC	14	VGA_VSYNC
15	VGA_SCL		

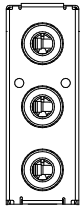
DVI

DVI Connector



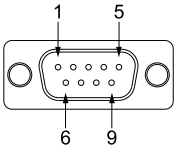
Pin #	Signal	Pin #	Signal
C1	NC	**C2**	NC
C3	NC	**C4**	NC
D1	DATA2-	**D2**	DATA2+
D3	GND	**D4**	NC
D5	NC	**D6**	DDCCLK -
D7	DDCDATA	**D8**	NC
D9	DATA1-	**D10**	DATA1+
D11	GND	**D12**	NC
D13	NC	**D14**	VCC5
D15	GND	**D16**	DVI_HPD
D17	DATA0-	**D18**	DATA0+
D19	GND	**D20**	NC
D21	NC	**D22**	GND
D23	CLK+	**D24**	CLK-

AUDIO1



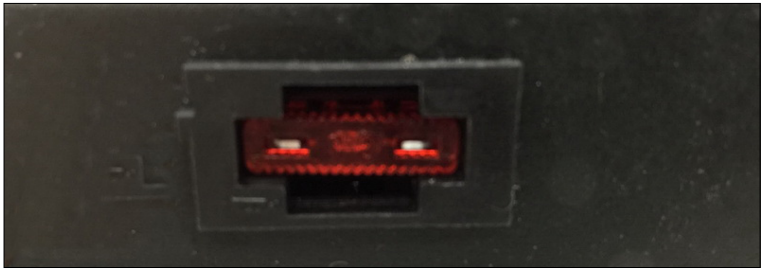
Jack	Function
Remote Switch (Blue)	Remote Switch Input
Speaker (Green)	Line Out
MIC (Pink)	Microphone Input

COM3, COM4



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

Blade-type Fuse Holder



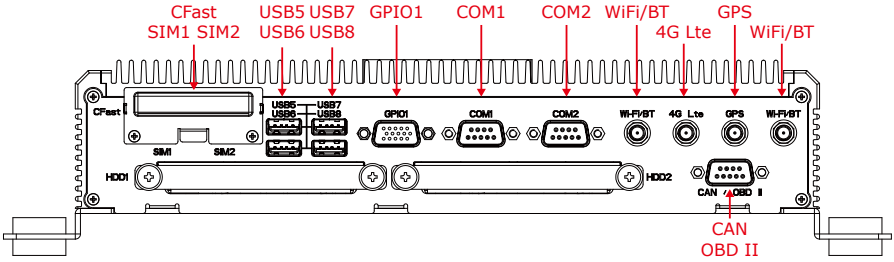
Power-input fuse suggestion:

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

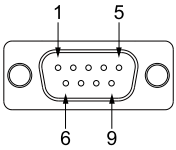
Car Battery	Blade-type fuse suggestion	Remarks
12V System	CONQUER ATQ-10	Voltage Rating: 32V; Current Rating: 10A
24V System	CONQUER ATQ-5	Voltage Rating: 32V; Current Rating: 5A

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

1.5.3. Rear I/O Panel

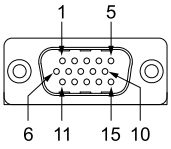


COM1, COM2



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

GPIO1 **GPIO Function**



Pin #	Signal	Pin #	Signal
1	GPO0	2	GPO1
3	GPO2	4	GPO3
5	GND	6	GND
7	N/A	8	N/A
9	GND	10	N/A
11	GPI4	12	GPI5
13	GPI6	14	GPI7
15	EXTPWR		

USB5 ~ USB8 **USB 3.0 Connector**

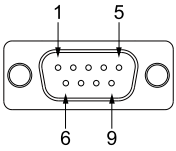
SIM1, SIM2 **SIM Card Holder**

CFast **CFast Card Socket**

WiFi/BT, 4G Lte, GPS

Reserved for installation of 4x optional SMA-type antenna

CAN, OBD II



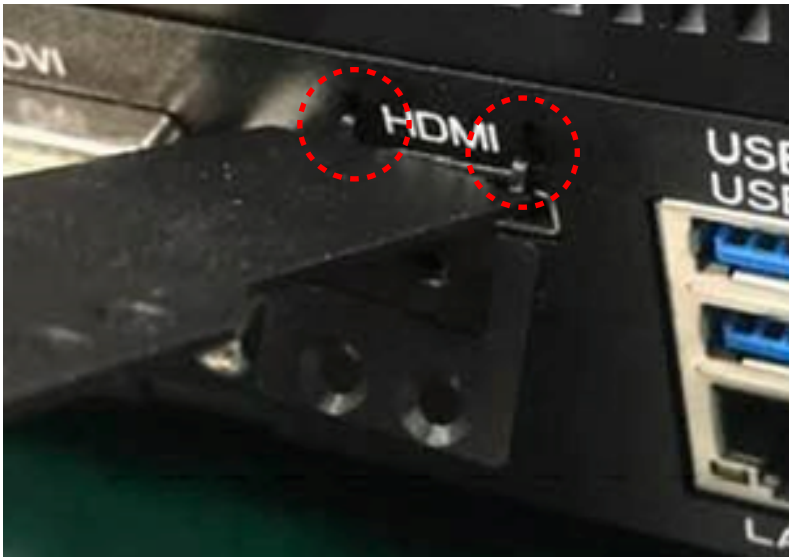
Pin #	AIV-Q170V1FL CAN Bus Pin Out	AIV-Q170V1FL-OBD OBD II Pin Out
1		Signal Ground
2		Chassis Ground
3	Can High	Can High
4		ISO9142-2 K Line
5	Can Low	Can Low
6		J1850 Bus-
7		J1850 Bus+
8		ISO9142-2 L Line
9		Battery Power

2. Components Assembly

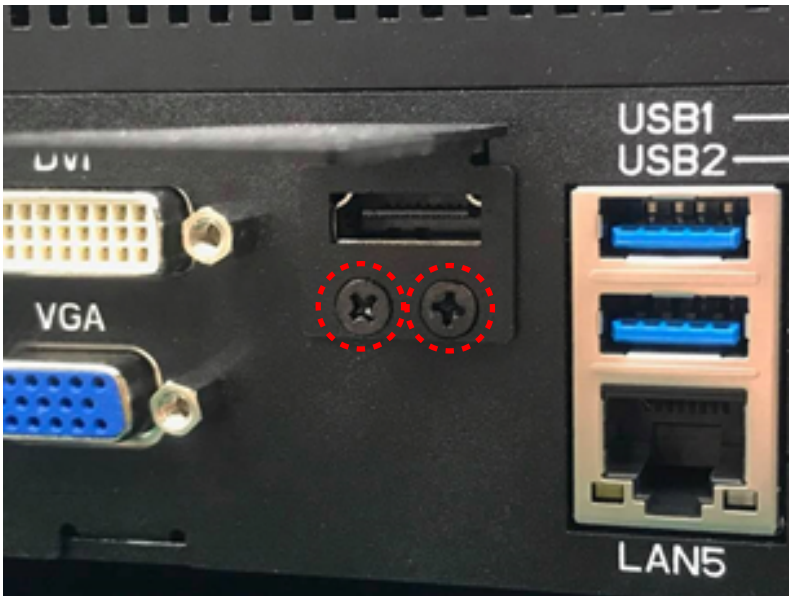
2.1. HDMI Cable Connection

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

Step 1: As shown below, insert the locking-bracket by aligning its hooks with holes on the chassis.



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. Firmly push the HDMI cable all the way into the socket.

Step 4: 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 5: Choose the holes that allows the screw to lock the cable-end with cable-holder.

Step 6: Fasten the cable-end with cable-holder. The HDMI locking-bracket is now held into position tightly.

2.2. 2.5” SATA SSD Installation

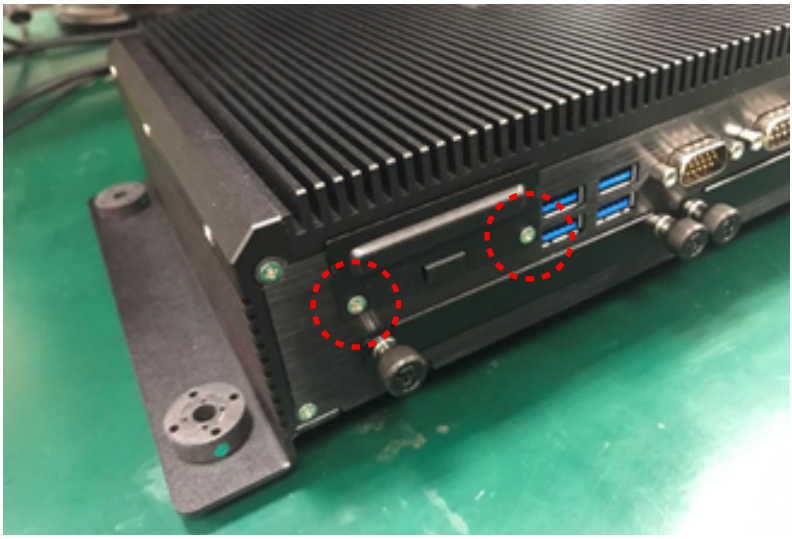
Step 1: Loosen the four disk-tray screws by fingers. Pull out the two disk-trays.

Step 2: Install your 2.5” SATA disk. 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 four disk-tray screws by fingers.

2.3. CFast/SIM Card Installation

Step 1: Loosen two screws on the CFast/SIM card cover-plate.

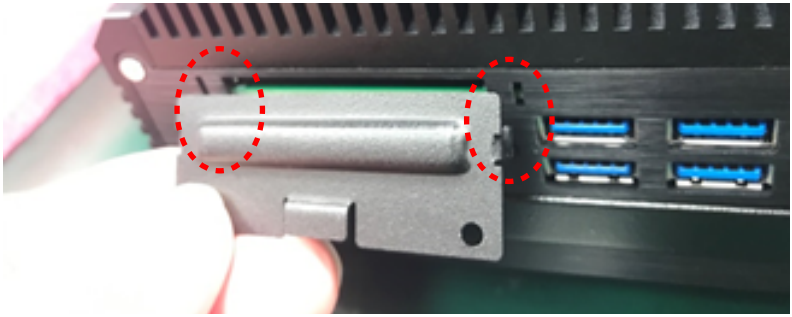


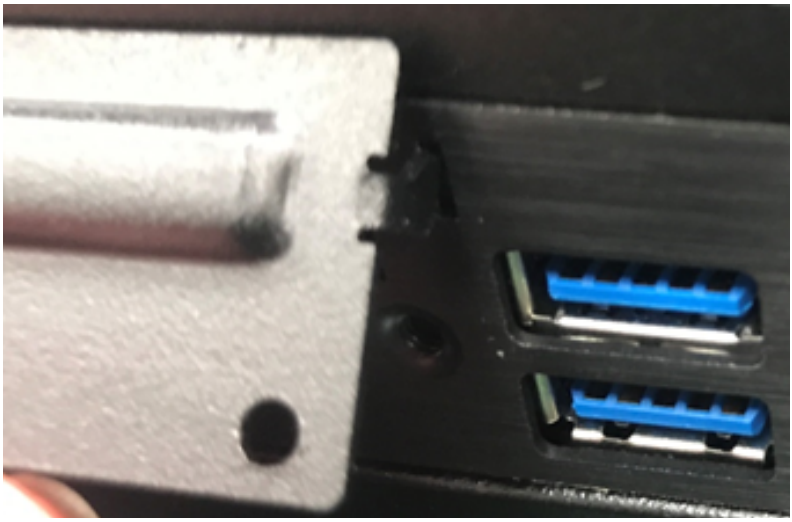
Step 2: Use your finger to pick up the notch on the cover-plate to remove it.

Step 3: Take out the cover-plate.

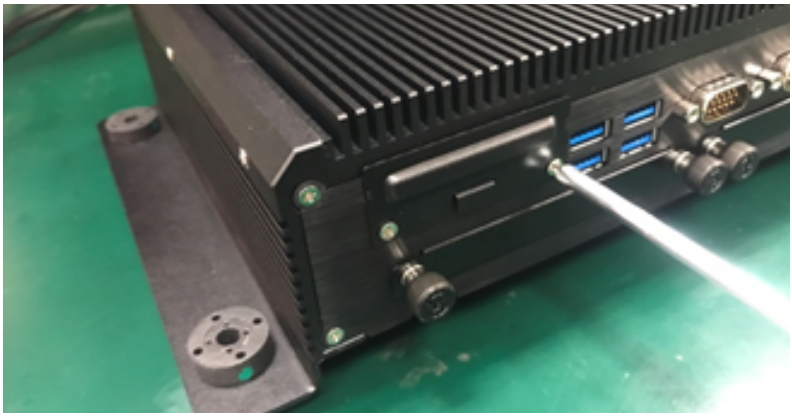
Step 4: Insert your CFast or SIM card into the slot. Pay attention to its orientation, and do not scratch its contacts.

Step 5: Align the two tiny hooks on the cover-plate with two tiny holes on the chassis.





Step 6: Lock up the cover-plate with two screws.



2.4. Antenna Connection

Connect your antennas needed according to your system configuration.

You will have to match the gender in connecting antenna plug with socket.

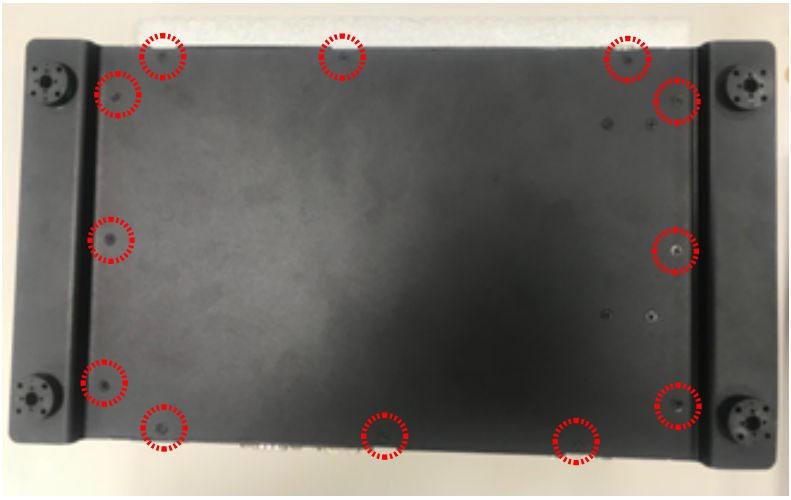
Connect a male type antenna to the female type socket (GPS):

Connect a female type antenna to the male type socket (WiFi/BT):

2.5. Memory Card Installation

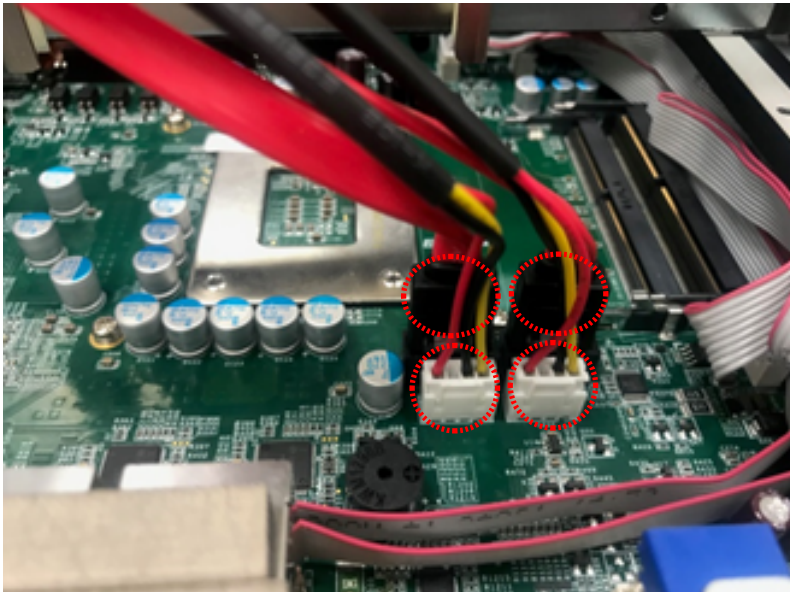
Step 1: Loosen the four disk-tray screws by fingers. Pull out the two disk-trays.

Step 2: Remove the 12 screws that lock up the bottom cover.



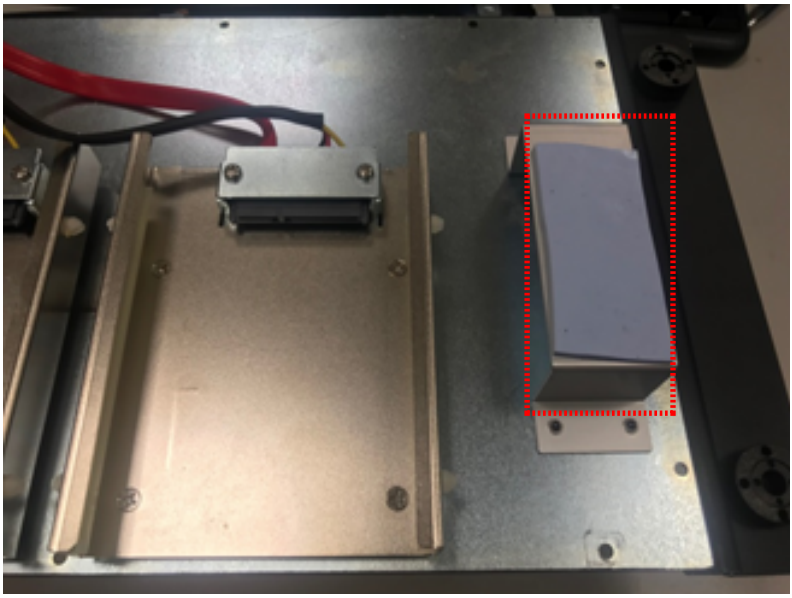
Step 3: Remove the bottom plate.

Step 4: Disconnect the SATA cables and SATA power cables on the mainboard.

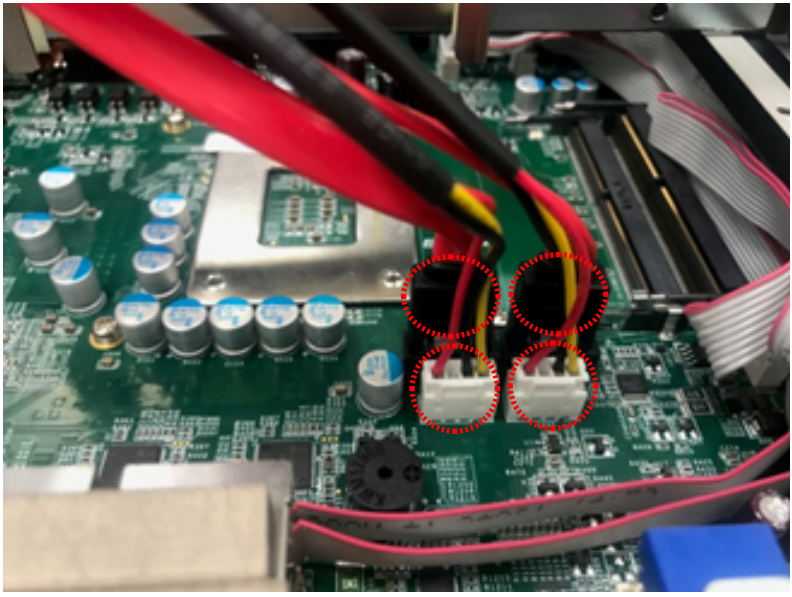


Step 5: Paste a thermal conductive sheet on the mainboard close to the memory slot.

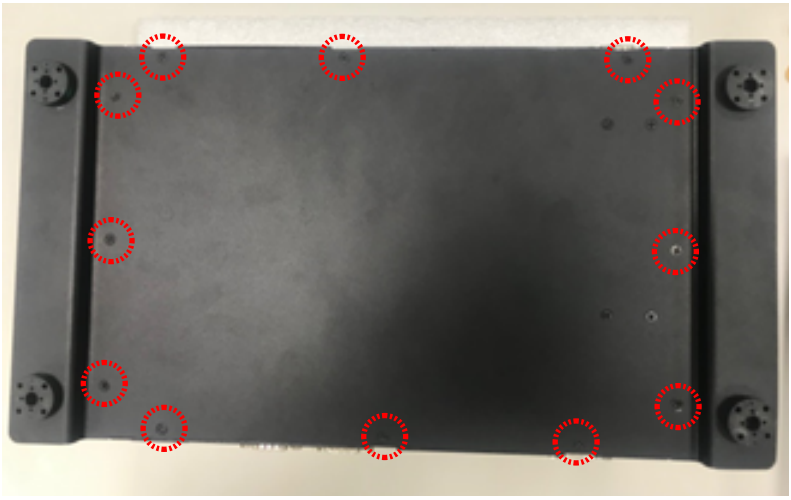
Step 6: For two memory modules installation, paste another thermal conductive sheet on the bracket located at the bottom plate.



Step 7: After finished installation of memory modules, re-connect the SATA cables and SATA power cables on the mainboard.



Step 8: Put back the bottom cover and lock it up with 12 screws.



Step 9: Push in the two disk-trays and lock up the cover-plate screws by fingers.

3. BIOS Settings

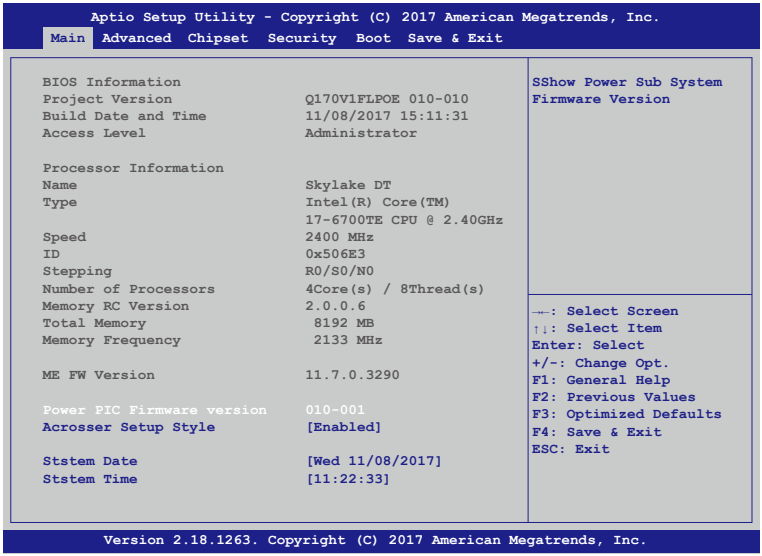
This chapter describes the BIOS menu displays and explains how to perform common tasks needed to get the system up and running. It also gives detailed explanation of the elements found in each of the BIOS menus. The following topics are covered:

- Main Setup
- Advanced Setup
- Chipset Setup
- Security Setup
- Boot 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

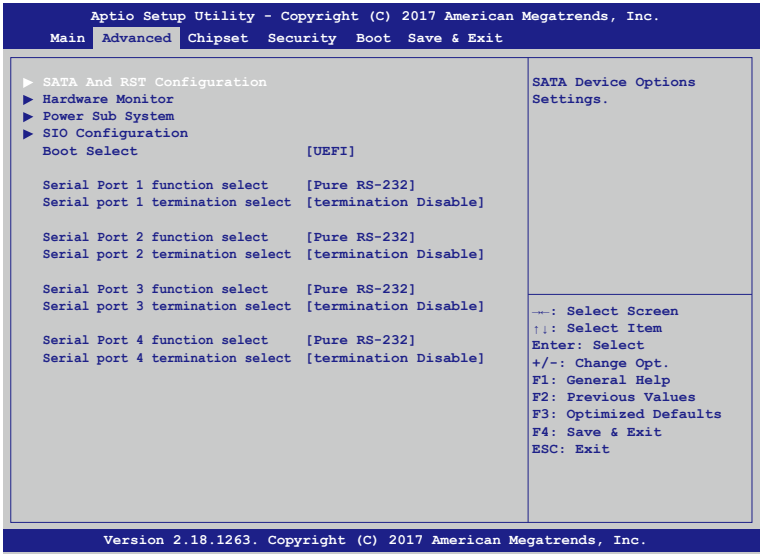
The BIOS setup main menu includes some options. Use the [Up/Down] arrow key to highlight the option, and then press the <**Enter**> key to select the item and configure the functions.



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.*

- **Display All Setup Item**
  Enable to show all setup items.
- **System Language**
  Choose the system default language.
- **System Date**
  Set the system date. Use Tab to switch between Date elements.
- **System Time**
  Set the system time. Use Tab to switch between Time elements.

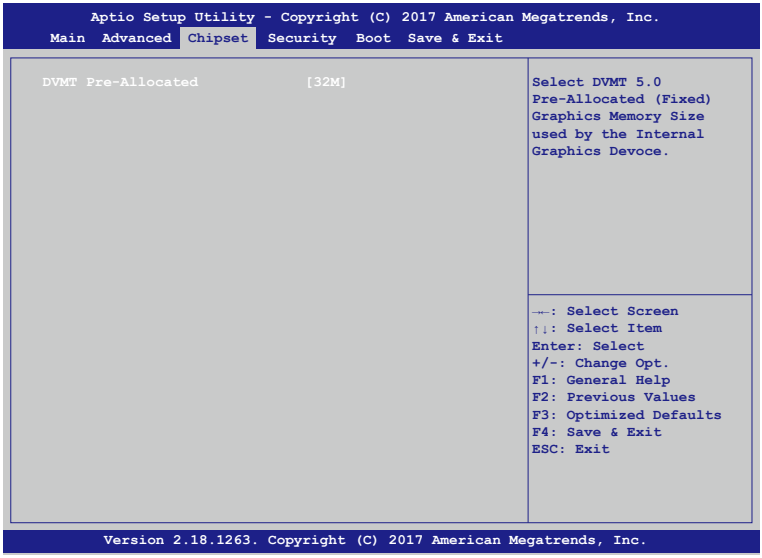
3.2. Advanced Setup



- **SATA And RST Configuration**
  SATA Device Options Settings.
- **Hardware Monitor**
  Monitor hardware status.
- **Power Sub System**
  Power Sub System.
- **SIO Configuration**
  SIO Configuration Parameters.
- **Boot Select**
  This option controls Legacy/UEFI bot.
- **Serial Port 1 function select**
  Select COM PORT function.

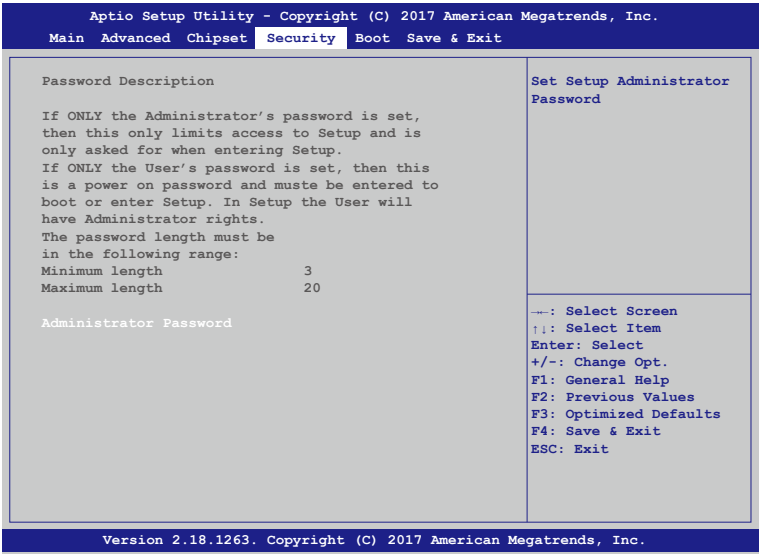
- **Serial port 1 termination select**
  Select termination Enable/Disable.

3.3. Chipset Setup



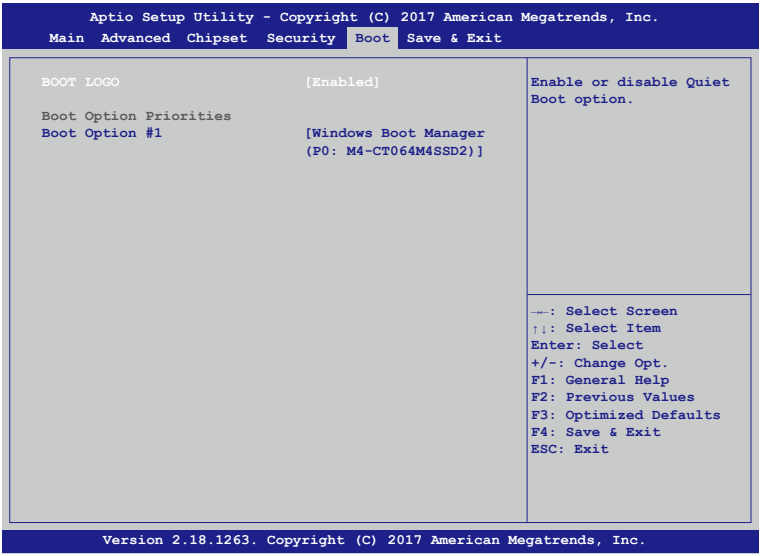
- **DVMT Pre-Allocated**
  Select DVMT 5.0 Pre-Allocated (Fixed) Graphics Memory Size used by the Internal Graphics Devoce.

3.4. Security Setup



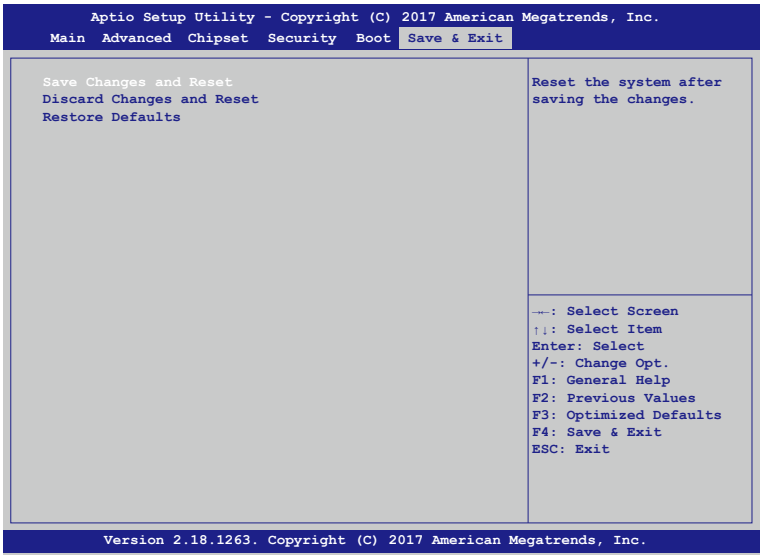
- **Administrator Password**
  Set Administrator Password.

3.5. Boot Setup



- **BOOT LOGO**
  Enable or disable Quiet Boot option.
- **Boot Option #1**
  Sets the system boot order.

3.6. Save & Exit Setup



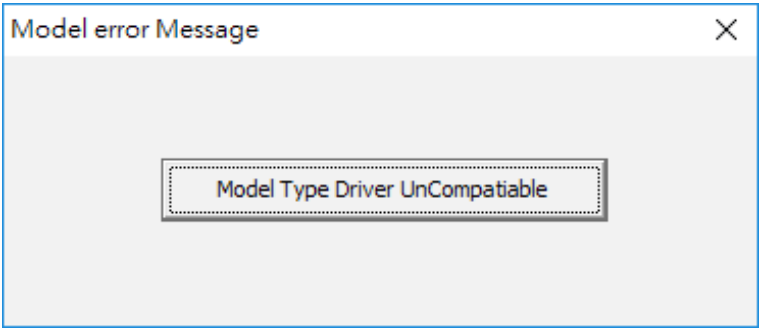
- **Save Changes and Reset**
  Reset the system after saving the changes.
- **Discard Changes and Reset**
  Reset system setup without saving any changes.
- **Restore Defaults**
  Restore/Load Default values for all the setup options.

4. Driver and Utility Installation

4.1. Driver CD Interface Introduction

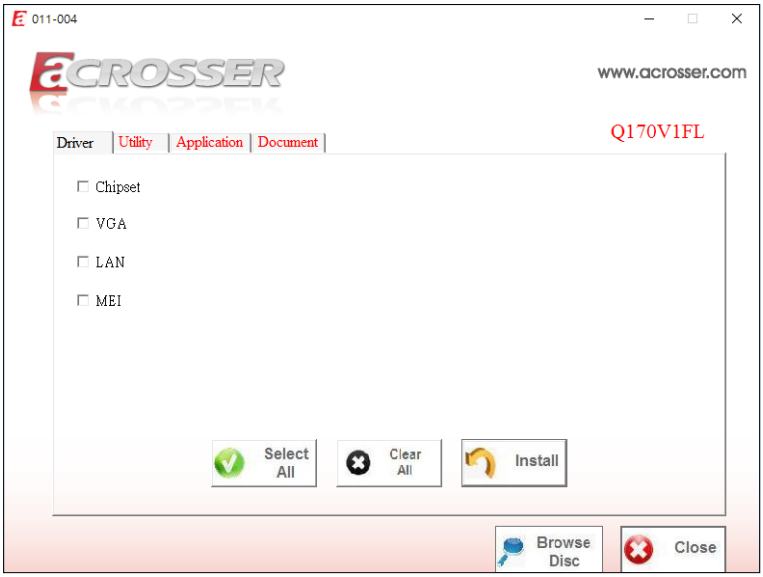
Acrosser provides a Driver CD compiled with all the drivers, utilities, applications and documents this product may need.

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.

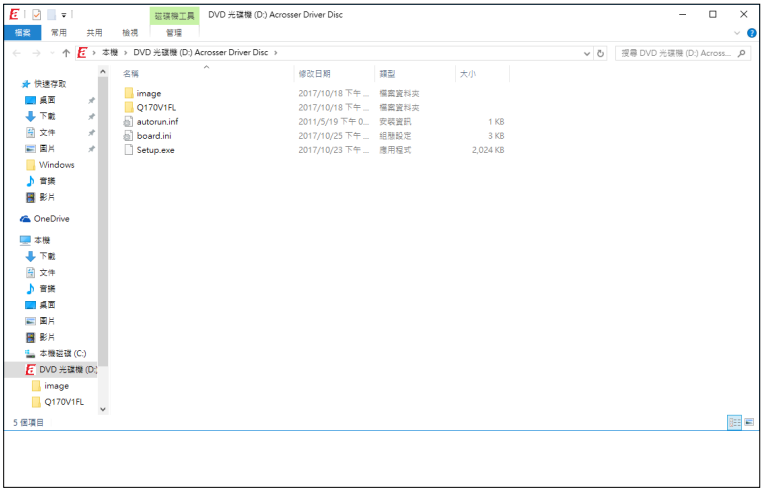


4.2. Windows Installation

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

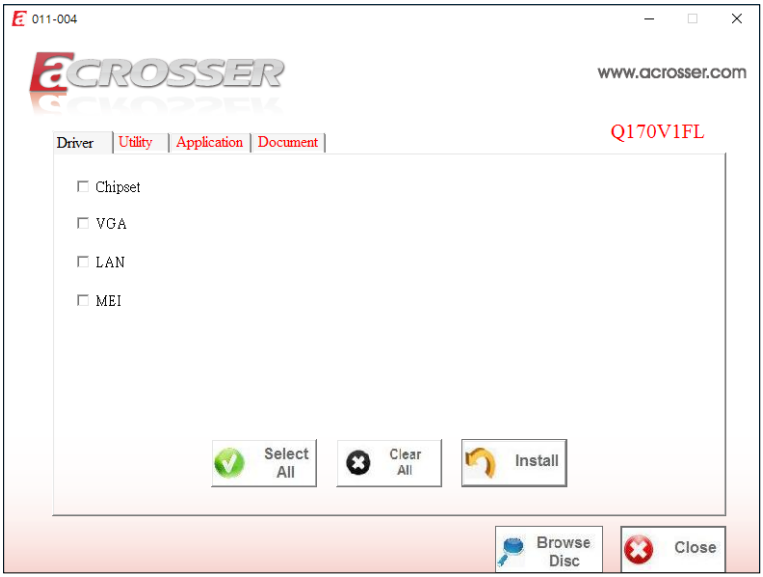


If not, enter the root folder of the Driver CD, run the excution file "**Setup.exe**".

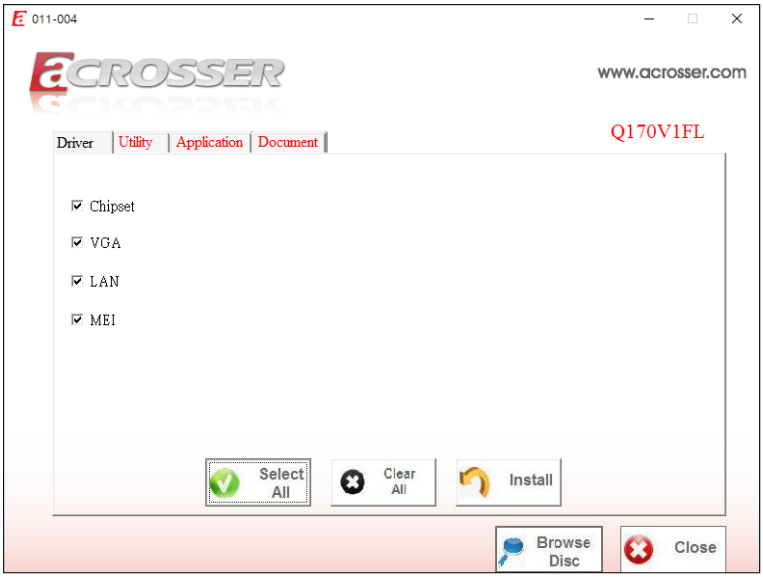


4.2.1. Driver Installation Page

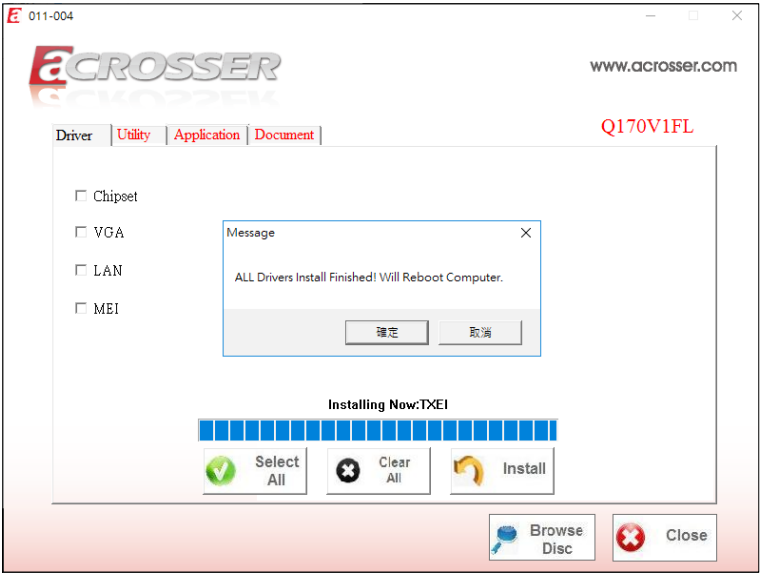
Step 1: Select the “**Driver**” tab.



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.

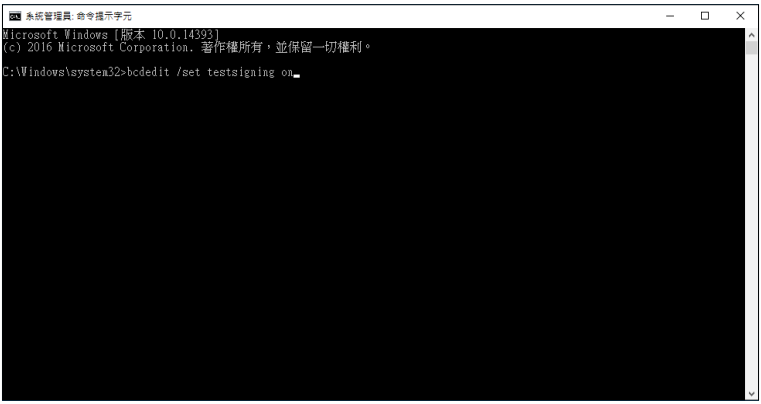


*Note: Select the "**Clear All**" button will clear all the selections, and then you can select the driver you want to install one by one, but the "**Chipset**" driver has to be installed before installing all the others.*

4.2.2. Utility Page

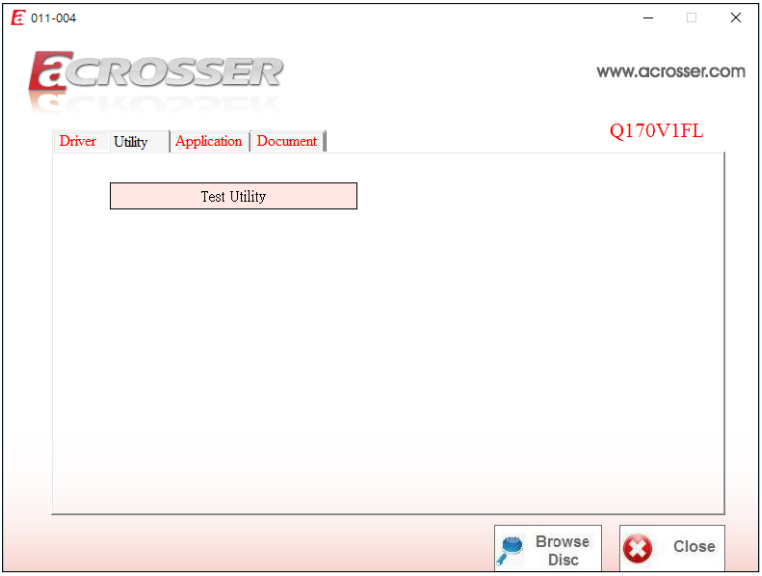
Before launching the utility, you should run the Windows test mode by running the command "**BCDEdit /set testsigning on**", and restart the system.

If you want to call this **AcrosserLib.dll** API file to initiate peripherals function, e.g. GPIO, PIC, or WatchDog, also run this command first, and restart the system.



To shutdown the Windows test mode, run the command "**BCDEdit /set testsigning off**", and restart the system.

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

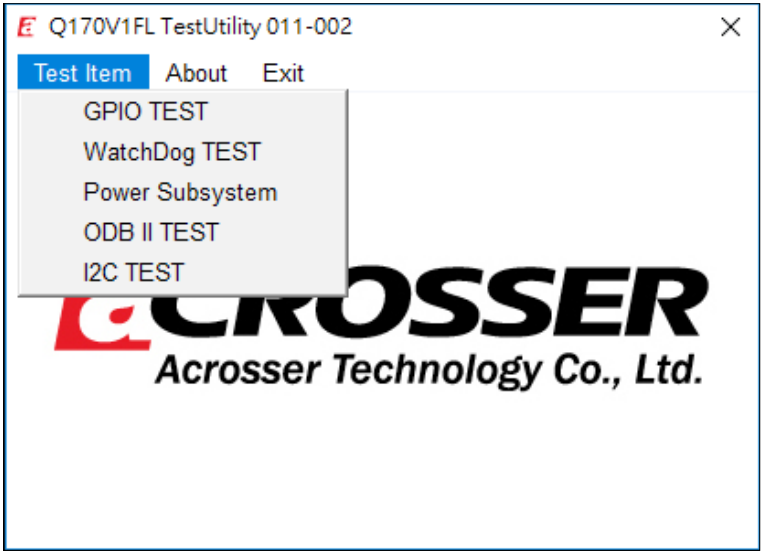


Step 2: The “**Test Utility**” screen appears.

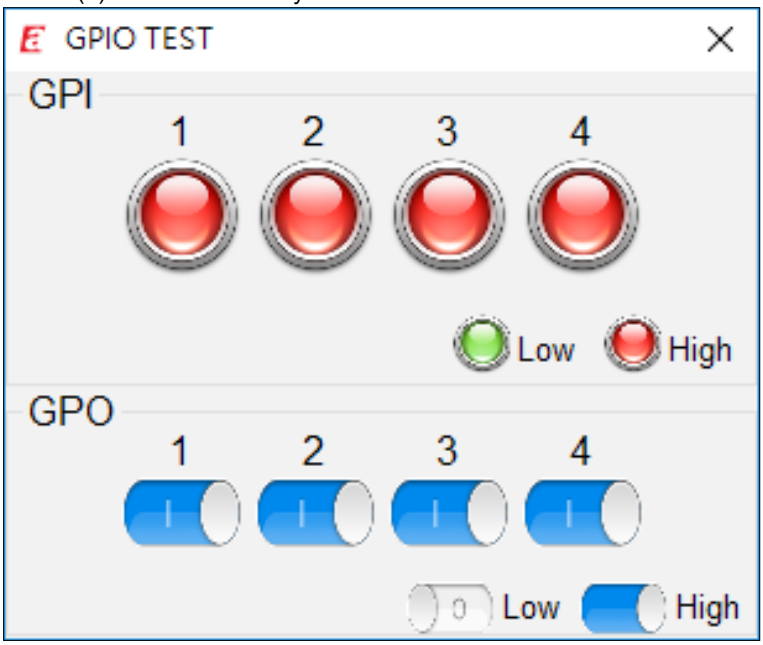
Click Test Item:

For model AIV-Q170V1FL:

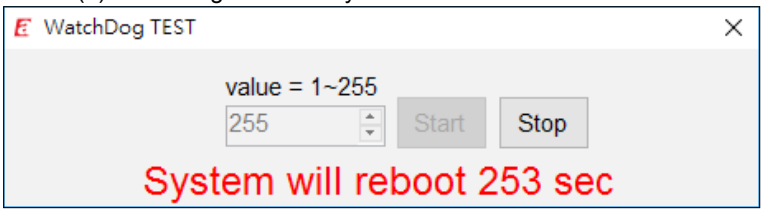
For model AIV-Q170V1FL-OBD:



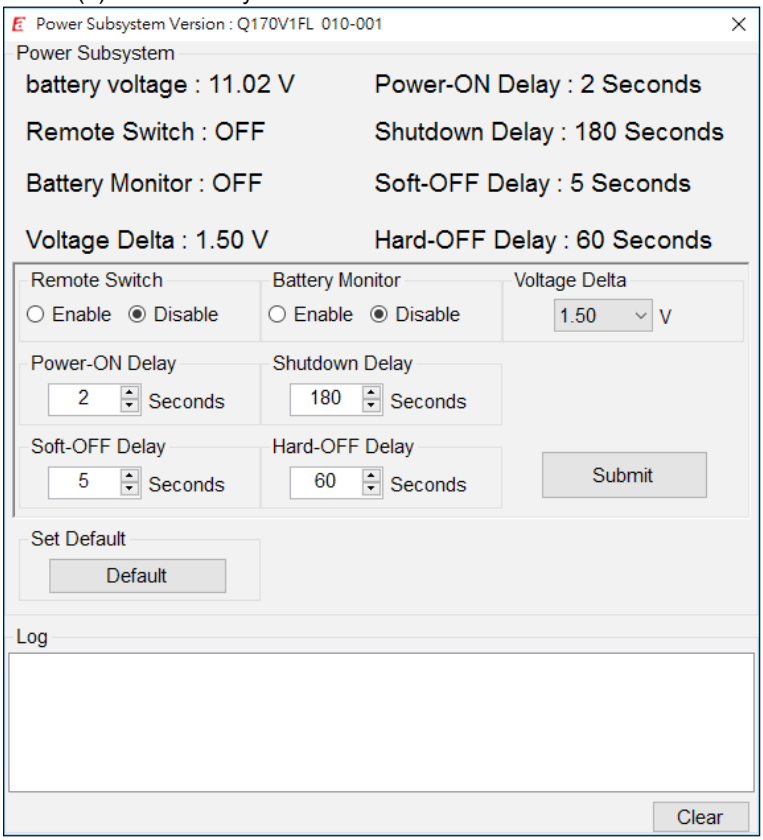
Select (1) GPIO TEST Utility:



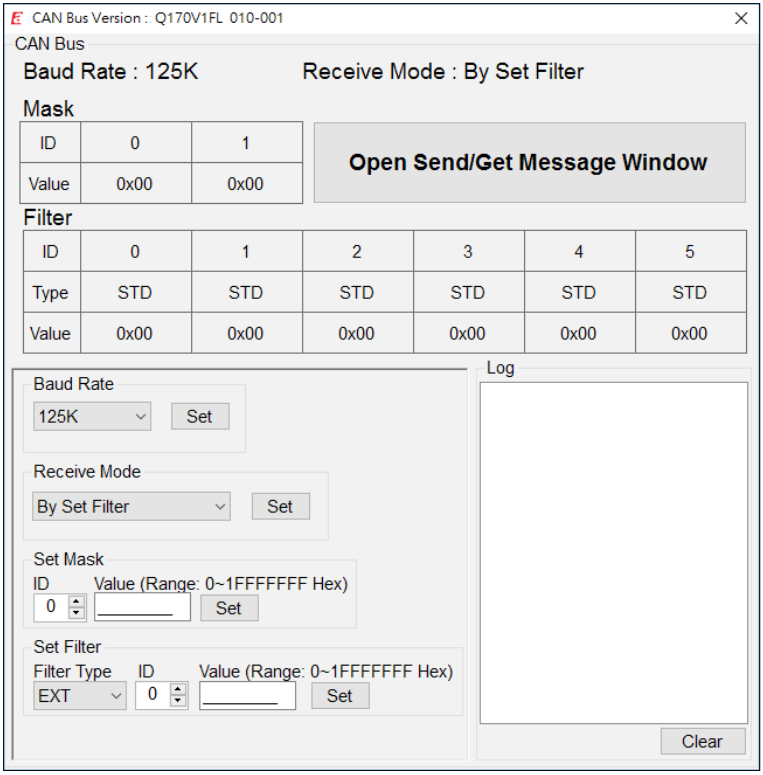
Select (2) WatchDog TEST Utility:



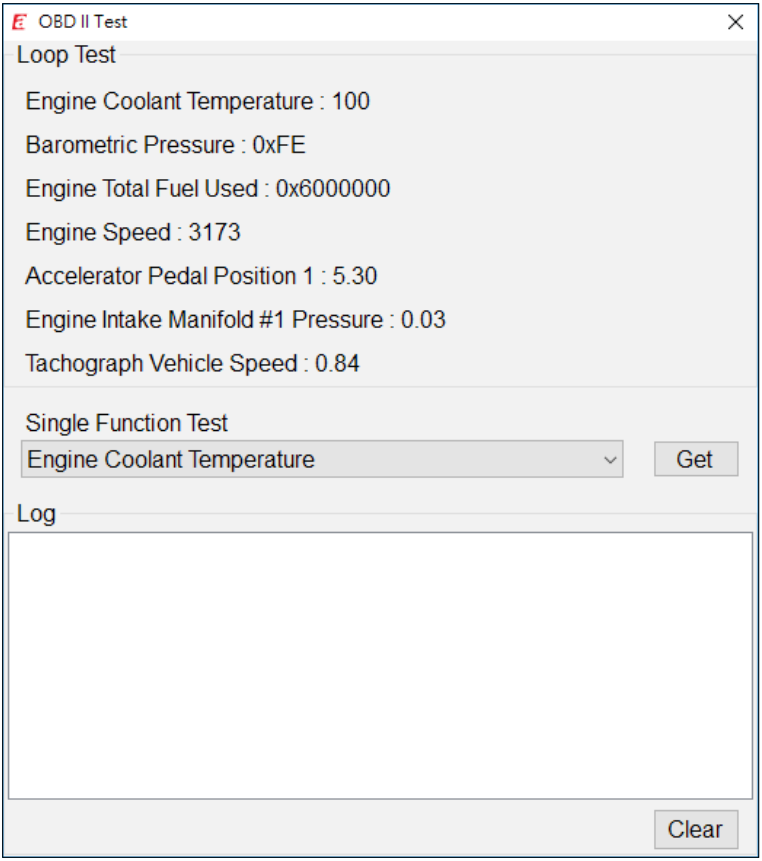
Select (3) Power Subsystem:



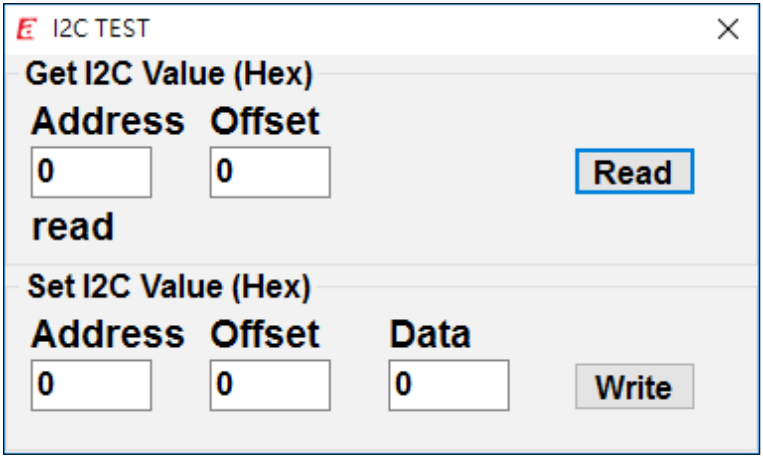
Select (4) CAN Bus: (For model AIV-Q170V1FL)



Or select (4) OBD II Test: (For model AIV-Q170V1FL-OBD)



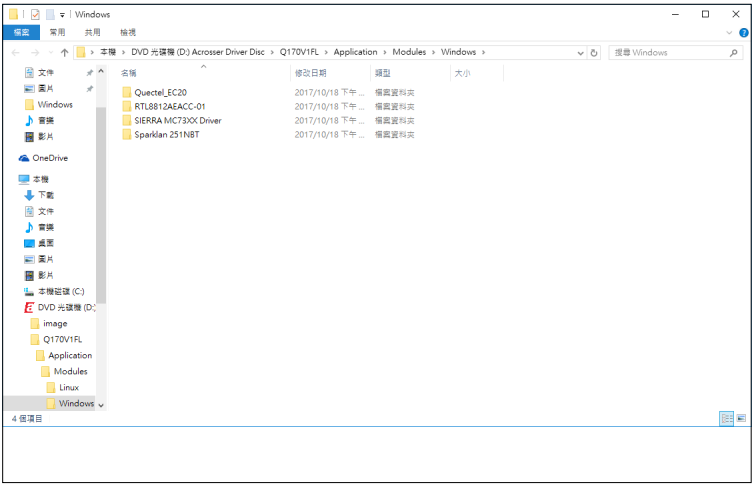
Select (5) I2C TEST:



4.2.3. Application Installation Page

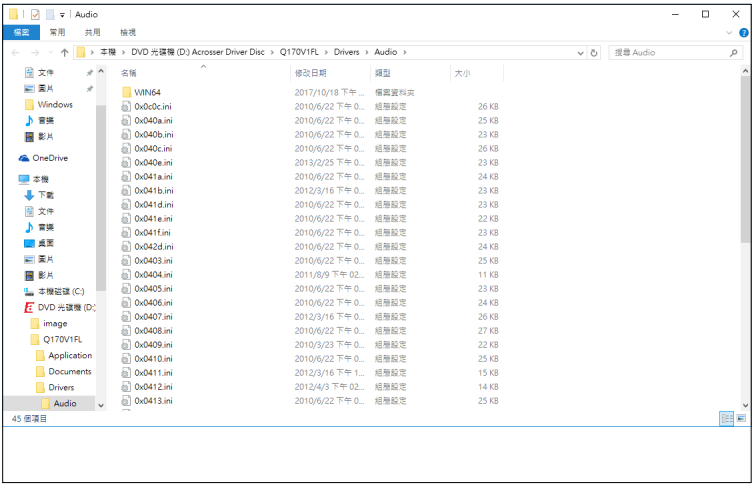
Step 1: Select the "**Application**" tab. Click the "**Drivers for Optional Modules**" box.

Step 2: Select the driver you want to install.



Step 3: Select the “**Application**” tab. Click the “**Audio Driver**” box.

Step 4: Click “**Setup.exe**” to install audio driver.

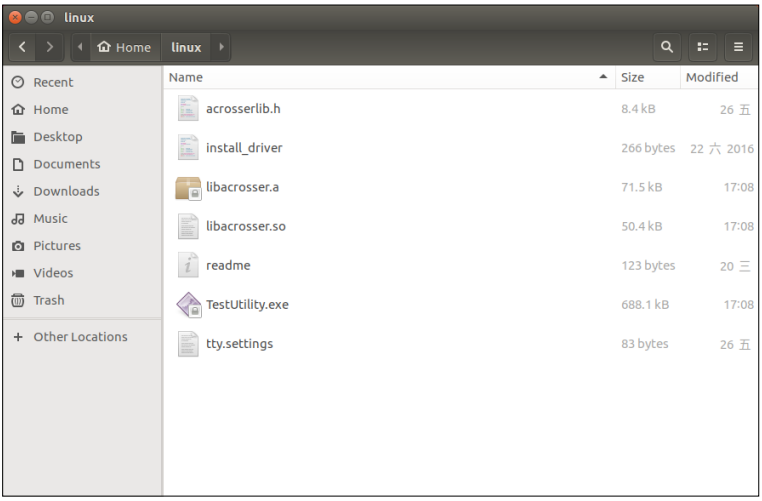


4.2.4. Document Page

The user manual is stored in the “**Document**” folder.

4.3. Linux Configuration

Step 1: Before running the shell script file install_driver to complete the utiltiy, make sure to have Internet access.

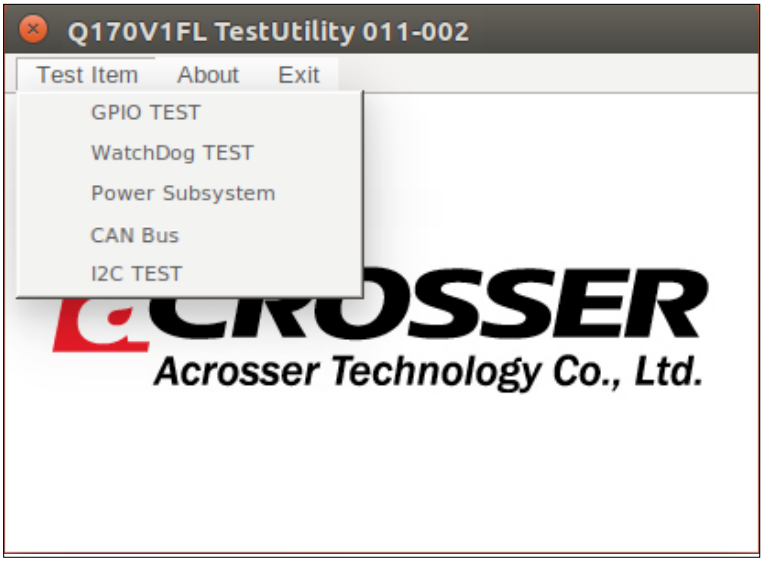


Run the **sudo mono TestUtility.exe**.

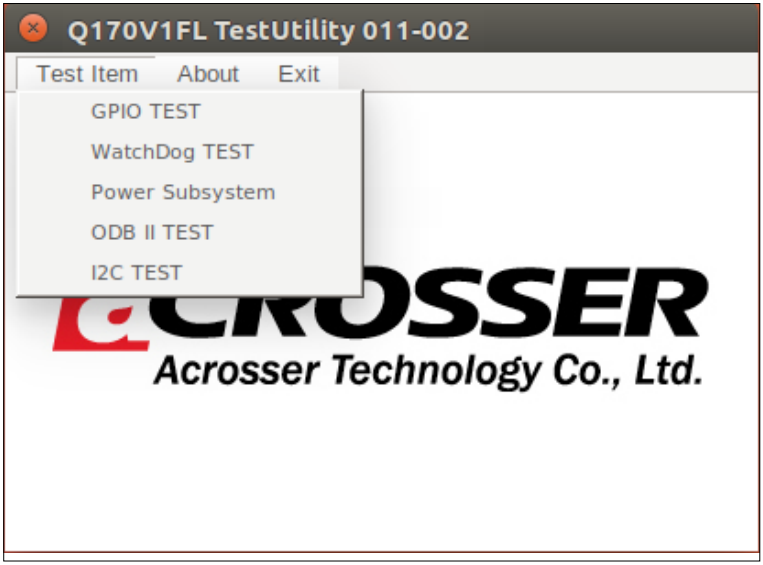
Step 2: The "**Test Utility**" screen appears.

Click Test Item:

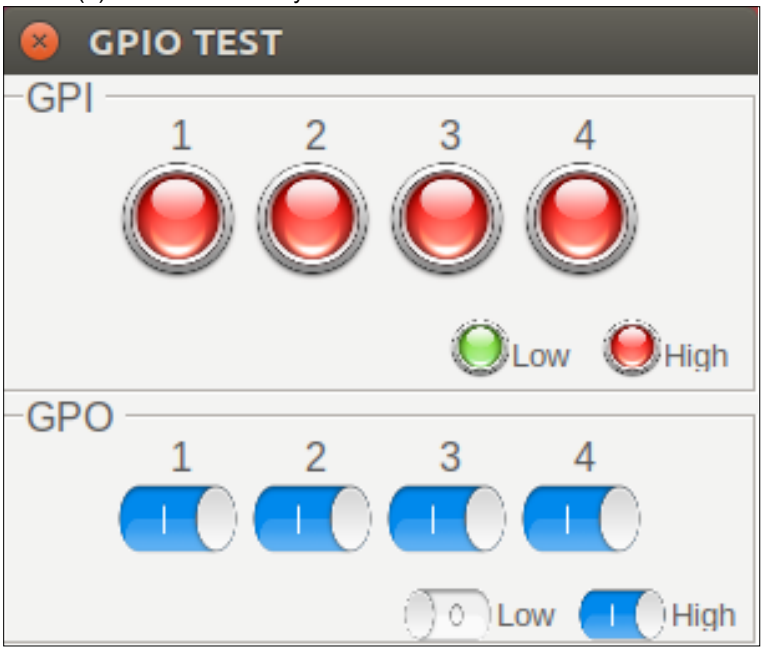
For model AIV-Q170V1FL:



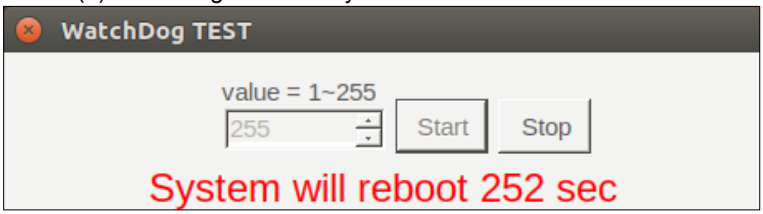
For model AIV-Q170V1FL-OBD:



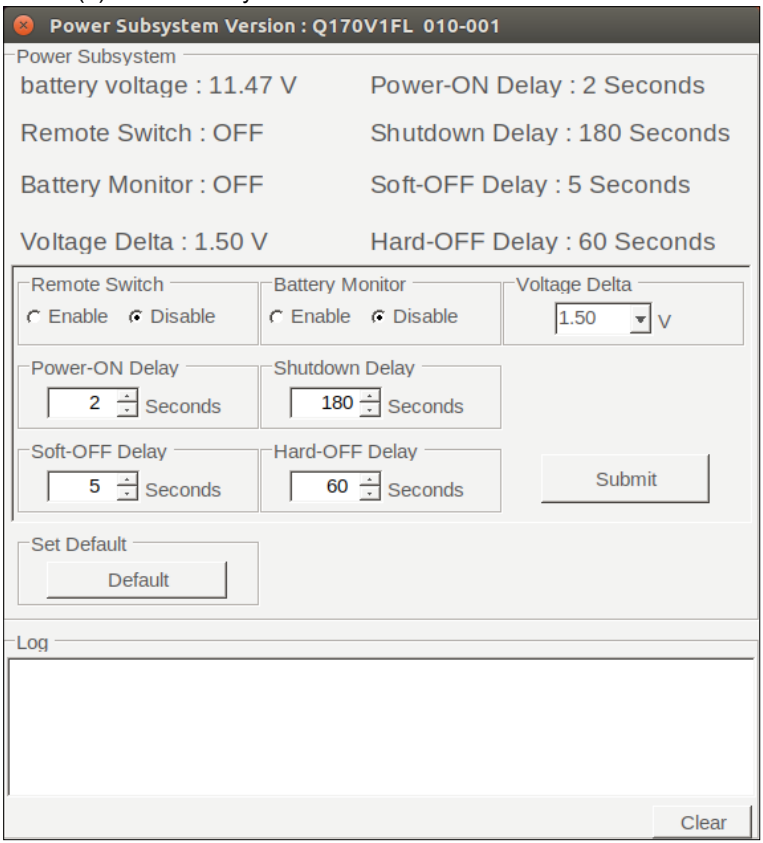
Select (1) GPIO TEST Utility:



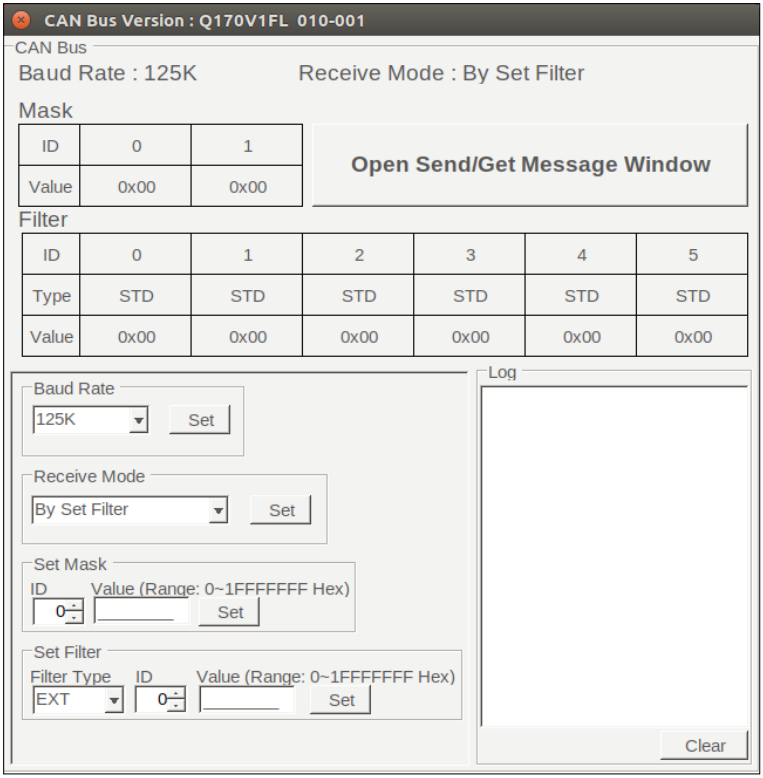
Select (2) WatchDog TEST Utility:



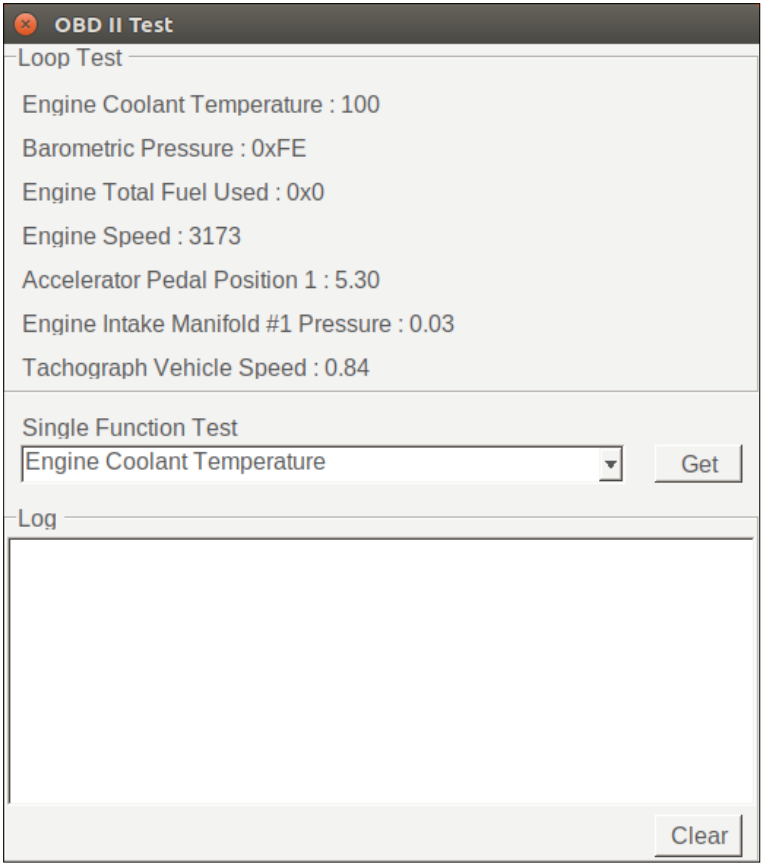
Select (3) Power Subsystem:



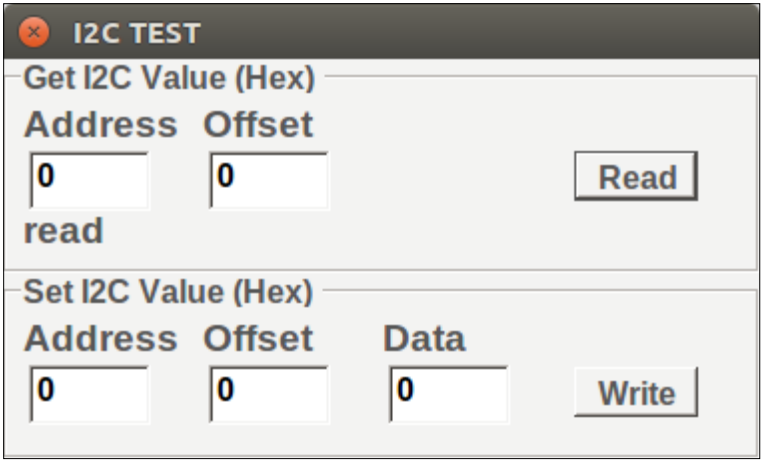
Select (4) CAN Bus: (For model AIV-Q170V1FL)



Or select (4) OBD II Test: (For model AIV-Q170V1FL-OBD)



Select (5) I2C TEST:



5. Software Installation and Programming Guide

5.1. Introduction

5.1.1. Environment

This test utility develop based on kernel 4.4 or above (Ubuntu 16.10 Desktop 64bit), and Windows 10 (64bit).

5.1.2. CAN Bus

5.1.2.1. 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 '**Q170V1FL\Utility\Windows**' on the CD, we provide:

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

5.1.2.2. 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;
```

5.1.3. GPIO and Watchdog

5.1.3.1. 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.

5.1.4. Power Subsystem

5.1.4.1. 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 delay.
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.

5.2. API List and Descriptions

5.2.1. 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[18];`
	`}`
	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 function 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.

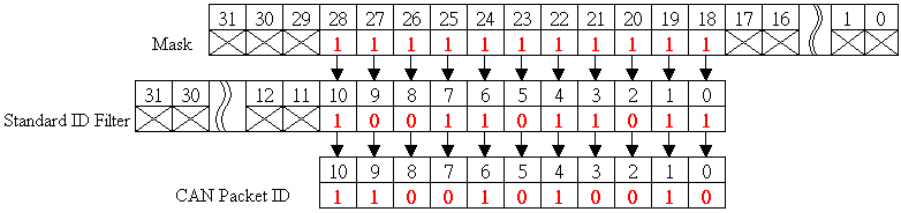
Syntax:	i32 setCanFilter(struct CanFilter *varFilter)

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.

5.2.2. GPIO and Watchdog

5.2.2.1. GPIO

Syntax:	i32 getChLevel(u8 *val)
Description:	Get the status of GPIO input pins and output pins, and put the value at *val.

Parameters:	This function takes a pointer to an unsigned char variable as the parameter.
	The bit0 ~ bit3 in the pointed variable '*val' is the status of the output pins. The bit4 ~ bit7 in the pointed variable '*val' is the status of the input pins.
	For example:
	`u8 val;`
	`i32 result;`
	`result = getChLevel( &val);`
	`if(result != 0)`
	`printf("Fail to get GPIO status!\n");`

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

Syntax:	i32 setChLevel(u8 val)
Description:	Set the status of GPIO Output pins.

Parameters:	This function takes an unsigned char as the parameter. The bit0 ~ bit3 in variable 'val' represent the status of the output pins. The bit3 ~ bit7 in the variable 'val' are of no use and can be neglected.
	For example:
	`u8 val = 0xf;`
	`i32 result;`
	`result = setChLevel(val);`
	`if(result != 0)`
	`printf("Fail to set GPIO!\n");`

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

5.2.2.2. Watchdog

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 unsigned character.

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.

5.2.3. 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: 2 seconds`
	`OS Shutdown Delay → Default: 3 minutes`

Parameters:	None.
Return Value:	If this function works successfully, the function will return 0, any other value stands 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 stands 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 stands 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 stands 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 stands 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 stands 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 stands 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 stands 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 0~3600. 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 function sets up the interval in seconds.
Parameters:	The parameter is of the type of unsigned long. The value of the parameter ranges from 0~3600. 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 2 ~ 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.

5.2.4. J1939(STN1110)

Syntax:	int get_engine_coolant_temperature(void)
Description:	This function can get the Engine Coolant Temperature.
Parameters:	None.
Return Value:	An integer.

Syntax:	int get_engine_fuel_temperature_1(void)
Description:	This function can get the Engine Fuel Temperature 1.
Parameters:	None.
Return Value:	80 fixed

Syntax:	unsigned short get_engine_oil_temperature_1(void)
Description:	This function can get the Engine Oil Temperature 1.
Parameters:	None.
Return Value:	0xFFFF (not yet implemented)

Syntax:	unsigned short get_engine_turbocharger_oil_ temperature(void)
Description:	This function can get the Engine turbocharger oil Temperature.
Parameters:	None.
Return Value:	0xFFFF (not yet implemented)

Syntax:	unsigned char get_engine_intercooler_ temperature(void)
Description:	This function can get the Engine Intercooler Temperature.
Parameters:	None.
Return Value:	0xFF (not yet implemented)

Syntax:	unsigned char get_engine_intercooler_thermostat_opening(void)
Description:	This function can get the Engine Intercooler Thermostat Opening.
Parameters:	None.
Return Value:	0xFF (not yet implemented)

Syntax:	unsigned char get_barometric_pressure(void)
Description:	This function can get the Barometric Pressure.
Parameters:	None.
Return Value:	0xFE (defective)

Syntax:	unsigned short get_cab_interior_temperature(void)
Description:	This function can get the Cab Interior Temperature.
Parameters:	None.
Return Value:	0xFFFF (not available)

Syntax:	int get_ambient_air_temperature(void)
Description:	This function can get the Ambient Air Temperature.
Parameters:	None.
Return Value:	25 (fixed)

Syntax:	int get_engine_air_inlet_temperature(void)
Description:	This function can get the Engine Air Inlet Temperature.
Parameters:	None.
Return Value:	35 (fixed)

Syntax:	unsigned short get_road_surface_temperature(void)
Description:	This function can get the Road Surface Temperature.
Parameters:	None.
Return Value:	0xFFFF (not available)

Syntax:	int get_engine_trip_fuel(void)
Description:	This function can get the Engine Trip Fuel.
Parameters:	None.
Return Value:	0xFFFFFFFF (not used)

Syntax:	int get_engine_total_fuel_used(void)
Description:	This function can get the Engine Total Fuel Used.
Parameters:	None.
Return Value:	incremented every 5 ms by simulator

Syntax:	incremented every 5 ms by simulator
Description:	This function can get the Engine Torque Mode
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_drivers_demand_engine_percent_ torque(void)
Description:	This function can get the Driver's Demand Engine – Percent Torque
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_actual_engine_percent_ torque(void)
Description:	This function can get the Actual engine – Percent Torque
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	int get_engine_speed(void)
Description:	This function can get the Engine Speed
Parameters:	None.
Return Value:	An integer

Syntax:	unsigned char get_source_address_of_controlling_device(void)
Description:	This function can get the Source Address of controlling device.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_engine_starter_mode(void)
Description:	This function can get the Engine Starter Mode.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_engine_demand_percent_torque(void)
Description:	This function can get the Engine Demand – Percent Torque
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_accelerator_pedal_1_low_idle_switch(void)
Description:	This function can get the Accelerator Pedal 1 Low Idle Switch.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_accelerator_pedal_kickdown_switch(void)
Description:	This function can get the Accelerator Pedal kickdown Switch.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_road_speed_limit_status(void)
Description:	unsigned char get_road_speed_limit_status(void).
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_accelerator_pedal_2_low_idle_switch(void)
Description:	This function can get the Accelerator Pedal 2 Low Idle Switch.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	float get_accelerator_pedal_position_1(void)
Description:	This function can get the Accelerator Padal Position 1.
Parameters:	None.
Return Value:	An float

Syntax:	unsigned char get_engine_percent_load_at_current_speed(void)
Description:	This function can get the Engine Percent Load At Current Speed.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_remote_accelerator_pedal_position(void)
Description:	This function can get the Remote Accelerator Pedal Position.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_accelerator_pedal_position_2(void)
Description:	This function can get the Accelerator Padal Position 2.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_vehicle_acceleration_rate_limit_status(void)
Description:	This function can get the Vehicle Acceleration Rate Limit Status.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_actual_maximum_available_percent_torque(void)
Description:	This function can get the Actual Maximum Available Percent Torque.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_engine_particulate_trap_inlet_pressure(void)
Description:	This function can get the Engine Particulate Trap Inlet Pressure.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	float get_engine_intake_manifold_1_pressure(void)
Description:	This function can get the Engine Intake Manifold 1 Pressure.
Parameters:	None.
Return Value:	An float

Syntax:	unsigned char get_engine_intake_manifold_1_temperature(void)
Description:	This function can get the Engine Intake Manifold 1 Temperature.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_engine_air_inlet_pressure(void)
Description:	This function can get the Engine Air Inlet Pressure.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_engine_air_filter_1_differential_pressure(void)
Description:	This function can get the Engine Air Filter 1 Differential Pressure.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned short get_engine_exhaust_gas_temperature(void)
Description:	This function can get the Engine Exhaust Gas Temperature.
Parameters:	None.
Return Value:	0xFFFF(not yet implemented)

Syntax:	unsigned char get_coolant_filter_differential_pressure(void)
Description:	This function can get the Engine Coolant Filter Differential Pressure.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_driver_1_working_state(void)
Description:	This function can get the Driver 1 working state.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_driver_2_working_state(void)
Description:	This function can get the Driver 2 working state.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_vehicle_motion(void)
Description:	This function can get the Vehicle motion.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_driver_1_time_related_states(void)
Description:	This function can get the Driver 1 Time Related States.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_driver_card_drver_1(void)
Description:	This function can get the Driver card, driver 1.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_vehicle_overspeed(void)
Description:	This function can get the Vehicle Overspeed.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	This function can get the Vehicle Overspeed
Description:	This function can get the Driver 2 Time Related States.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_driver_card_drver_2(void)
Description:	This function can get the Driver card, driver 2.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_system_event(void)
Description:	This function can get the System event.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_handling_information(void)
Description:	This function can get Handling information.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_tachograph_performance(void)
Description:	This function can get Tachograph performance.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_direction_indicator(void)
Description:	This function can get Direction indicator.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	unsigned char get_tachograph_output_shaft_speed(void)
Description:	This function can get Tachograph output shaft speed.
Parameters:	None.
Return Value:	0xFF(not yet implemented)

Syntax:	float get_tachograph_vehicle_speed(void)
Description:	This function can get Tachograph vehicle speed.
Parameters:	None.
Return Value:	An float

5.3. 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
```

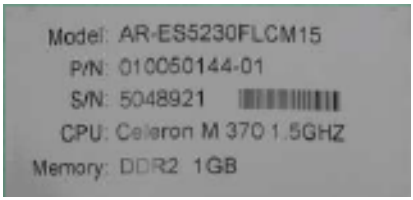
2. List the status of the COM ports in the system.

```
# setserial -g /dev/ttyS*

/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.
```

6. FAQ

Q 1. Where is the serial number located on my system?

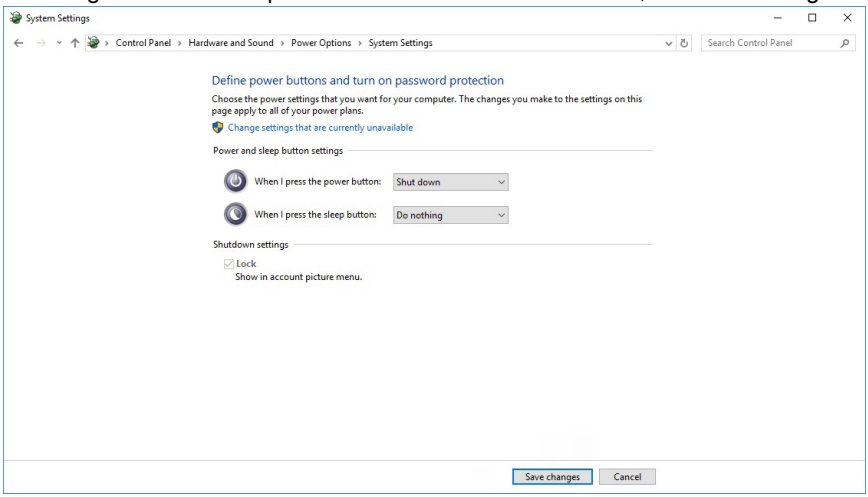
- The serial number (S/N) is an alpha-numeric character located on the bottom or side chassis.



(for reference only)

Q 2. How to setup the Remote Switch and Ignition Off function in Windows 10?

- To change the hardware power button action in Windows 10, do the following.



- Open Control Panel.
- Go to Control Panel > Hardware and Sound > Power Options.
  On the left, click the link "**Choose what the power buttons do**".
- In the drop down list "**When I press the power button**", select "**Shut down**".
- In the drop down list "**When I press the sleep button**", select "**Do nothing**".
- Click "**Save changes**" and exit.

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
深圳市福田区天安车公庙工业区
天经大厦AB座3楼B3-08 (邮编: 518040)
B3-08, 3/F, Block AB, Tianjing Building,
Chegongmiao Industrial Zone, Tian'an, Futian
District, 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.

8351 Elm Ave. Suite 107, Rancho Cucamonga,
CA91730, USA
TEL: +1-909-476-0071
FAX: +1-909-466-9951