

AIV-APL1V1FL-PT1 **AIV-APL1V1FL-PT10B** (OBD II)

Fan-less In-Vehicle System

Apollo Lake SoC with Smart Power System



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

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of Acrosser Technology Co., Ltd.

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: Jul. 22, 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..... | 5 |
| 1.1. Specifications | 5 |
| 1.2. Package Contents | 8 |
| 1.3. System Dissection | 9 |
| 1.3.1. Dimensions | 9 |
| 1.3.2. Front I/O Panel..... | 10 |
| 1.3.3. Rear I/O Panel | 15 |
| 2. Components Assembly..... | 16 |
| 2.1. 2.5" SATA SSD Installation | 16 |
| 2.2. SIM Card Installation | 18 |
| 2.3. Antenna Connection | 19 |
| 2.4. Brackets & Rubber Pads Installation | 21 |
| 3. BIOS Settings..... | 22 |
| 3.1. Main Setup | 22 |
| 3.2. Advanced Setup | 23 |
| 3.2.1. Trusted Computing..... | 24 |
| 3.2.2. F81960 Super IO Configuration | 25 |
| 3.2.3. Hardware Monitor..... | 26 |
| 3.2.4. Power Sub System..... | 27 |
| 3.2.5. CPU Configuration | 28 |
| 3.2.6. AMI Graphic Output Protocol Policy..... | 29 |
| 3.2.7. CSM Configuration | 30 |
| 3.2.8. USB Configuration | 31 |
| 3.3. Chipset Setup | 32 |
| 3.3.1. North Bridge | 33 |
| 3.3.2. South Bridge | 34 |
| 3.3.3. South Cluster Configuration | 35 |
| 3.4. Security Setup | 36 |
| 3.5. Boot Setup..... | 37 |
| 3.6. Save & Exit Setup..... | 38 |
| 4. Function Description..... | 39 |
| 4.1. Power input connection | 39 |
| 4.2. Digital Inputs..... | 39 |
| 4.3. Digital Outputs | 40 |

| | |
|---|-----------|
| 5. Driver and Utility Installation | 41 |
| 5.1. Driver CD Interface Introduction | 41 |
| 5.2. Windows Installation | 42 |
| 5.2.1. Driver Installation Page | 43 |
| 5.2.2. Utility Page | 45 |
| 5.2.3. Application Installation Page | 51 |
| 5.2.4. Document Page | 54 |
| 5.3. Linux Configuration | 55 |
| 6. Software Installation and Programming Guide | 62 |
| 6.1. Introduction | 62 |
| 6.1.1. Environment | 62 |
| 6.1.2. CAN Bus | 62 |
| 6.1.2.1. Overview | 62 |
| 6.1.2.2. CAN Message Format | 62 |
| 6.1.3. GPIO and Watchdog | 64 |
| 6.1.3.1. Overview | 64 |
| 6.1.3.2. Installing Device Driver | 64 |
| 6.1.4. Power Subsystem | 64 |
| 6.1.4.1. Overview | 64 |
| 6.1.5. I ² C | 65 |
| 6.1.5.1. Overview | 65 |
| 6.2. API List and Descriptions | 65 |
| 6.2.1. General | 65 |
| 6.2.2. J1939(STN1110) | 66 |
| 6.2.3. CAN Bus | 75 |
| 6.2.4. GPIO and Watchdog | 82 |
| 6.2.4.1. GPIO | 82 |
| 6.2.4.2. Watchdog | 83 |
| 6.2.5. Power Subsystem | 84 |
| 6.2.6. I ² c | 89 |
| 6.3. Appendix A | 90 |
| 7. FAQ | 91 |
| Q 1. Where is the serial number located on my system? | 91 |

1. System Introduction

The AIV-APL1V1FL Series is a fanless In-Vehicle Computer using Intel Apollo Lake 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 | <ul style="list-style-type: none">• Intel® Pentium® N4200 (1.1GHz, 2M Cache, up to 2.50 GHz) |
| Memory | <ul style="list-style-type: none">• 2x DDR3L SO-DIMM- 1866, (Up to 8GB/non-ECC) |

Display

| | |
|---------------------------|---|
| Graphic Controller | <ul style="list-style-type: none">• Gen9 GPU |
| Video Interface | <ul style="list-style-type: none">• 1x DVI-D• 1x VGA |

Storage

| | |
|----------------------------------|--|
| SATA | <ul style="list-style-type: none">• 2x SATA Connectors (Sata 3 signal)• 2x Power Connectors (JST 2.54mm, 1x4 pin) |
| Mini PCIe Slot | <ul style="list-style-type: none">• 3x Mini PCI-e sockets• Mini PCI-e 1 for 4G & GPS (USB signal) (Full size)• Mini-PCI-e 2 for Wi-Fi + BT (PCI-e + USB signal) (Full size)• Mini-PCI-e 3 for reserved (PCI-e + USB signal) (Full size) |
| I²C Pin Header | <ul style="list-style-type: none">• 1x I²C Pin Header (I²c signal) for G Sensor Board |
| G Sensor | <ul style="list-style-type: none">• 1x G Sensor Board Connect to I²C Pin Header (3-axis Accelerometer) |
| Disk Bay | <ul style="list-style-type: none">• 2x Swappable 2.5" HDD Bay with Anti-vibration |

Communication and I/O

Ethernet • 2x PCIE x 1 Intel i210 IT GbE chip via RJ-45 connector

USB • 4x USB 3.0

Serial Ports • 4x COM → DB9 (RS-232)
• 1x COM → DB9 (RS232/422/485)

VIDEO Input • 1x DB9

CANBUS • Use CAN/OBDII DB9 connection
1. Support CAN bus 2.0B
2. 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 |

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

CAN/OBD II • Use IC STN1110 design a module board for optional CAN BUS function

GPIO • Digital Input

| | |
|---------------------------------------|--|
| Input Channels | 4 |
| Input Voltage | 0 to 36 VDC at 25 Hz |
| Digital Input Levels for Dry Contacts | <ul style="list-style-type: none"> Logic level 0: Close to GND Logic level 1: Open |
| Digital Input Levels for Wet Contacts | <ul style="list-style-type: none"> Logic level 0: +3 V max. Logic level 1: +10 V to +36 V (Source to DI) |
| Isolation | 3 kV optical isolation |

• Digital Output

| | |
|------------------|--|
| Output Channels | 4, sink type |
| On-State Voltage | 24 VDC nominal, open collector to 36 VDC |
| Isolation | 3 kV optical isolation |

SIM • 2x SIM Card Sockets

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

Other Features

| | |
|----------------------------|--|
| Audio | <ul style="list-style-type: none"> 2x 3.5" Phone Jack: Pink: Mic In Green: Audio Out |
| Remote Switch | <ul style="list-style-type: none"> 1x 3.5" Phone Jack (Blue) |
| CMOS | <ul style="list-style-type: none"> RTC (+/- 2 seconds for 24hours) Lithium Battery (3V) for CMOS Data Backup |
| Hardware Monitoring | <ul style="list-style-type: none"> CPU Voltage CPU and System Temperature |
| Watchdog Timer | <ul style="list-style-type: none"> Software Programmable 0~255 Seconds, 0 = Disable Timer. |

Antenna

| | |
|---------------------|--|
| Antenna type | <ul style="list-style-type: none"> SMA-type antenna holes reserved for Wi-Fi, BT, 4G/LTE, or GPS. |
|---------------------|--|

Power Requirement

| | |
|---------------------|--|
| Power Supply | <ul style="list-style-type: none"> Power Sub System: 9~36 V Power Input 12V for System |
|---------------------|--|

Software

| | |
|-------------------|---|
| OS Support | <ul style="list-style-type: none"> Windows 10 (64 bit) Linux kernel 4.4 or above (64 bit) |
|-------------------|---|

Mechanical & Environment

| | |
|---------------------------------------|--|
| Thermal Design | <ul style="list-style-type: none"> Fanless (Heatsink) |
| Chassis | <ul style="list-style-type: none"> Aluminum extrusion heat sink & metal chassis |
| Dimension | <ul style="list-style-type: none"> 260mm (W) x 195mm(D) x 63mm(H) |
| Vibration | <ul style="list-style-type: none"> IEC 60068-2-64, 5~500Hz, 3GRMS(CF/SSD) For SSD only |
| Shock | <ul style="list-style-type: none"> IEC 60068-2-27, 50G 500m/s² 11MS For SSD only |
| Operating Temperature/Humidity | <ul style="list-style-type: none"> -25°C ~ 60°C / 0~90% -25°C ~ 55°C (+15°C) Follow EN50155 T1 |
| Storage Temperature | <ul style="list-style-type: none"> -40°C ~ 80°C |
| Certification | <ul style="list-style-type: none"> CE / FCC class B / E Mark, EN50155 |

1.2. Package Contents

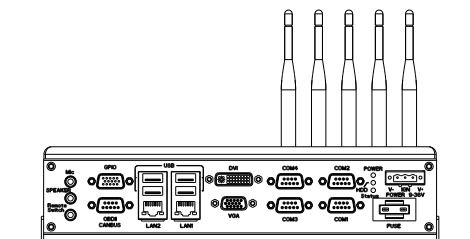
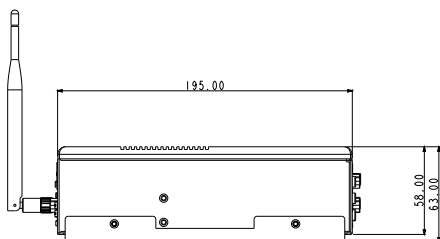
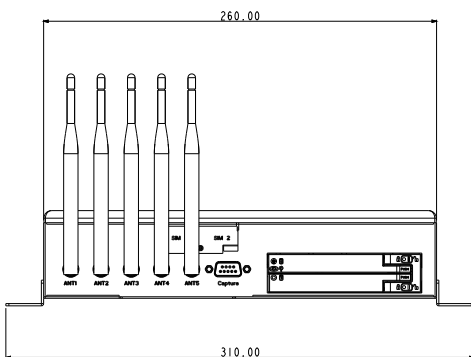
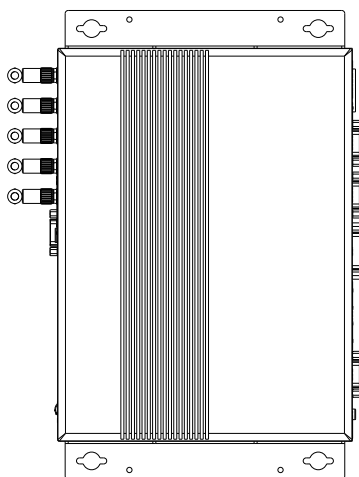
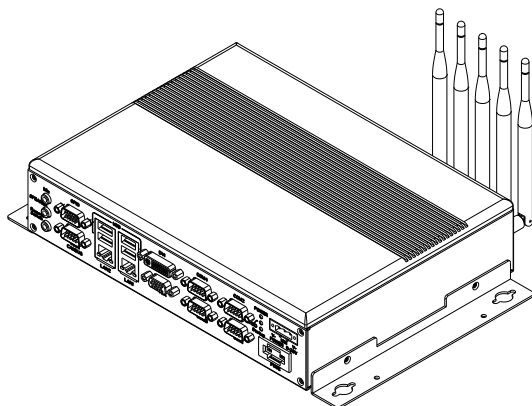
Check if the following items are included in the package.

| | Item | Q'ty |
|--------------------------|---|------|
| <input type="checkbox"/> | AIV-APL1V1FL-PT1 or AIV-APL1V1FL-PT1OB (OBD II) System | 1 |
| <input type="checkbox"/> | Remote Switch Cable | 1 |
| <input type="checkbox"/> | Driver CD | 1 |
| <input type="checkbox"/> | Screw Pack (2.5"HDD bracket: 8 pcs) | 1 |
| <input type="checkbox"/> | Terminal Block (Female 3-pin) | 1 |
| <input type="checkbox"/> | Spare Fuse | 1 |
| <input type="checkbox"/> | GPIO Cable | 1 |
| <input type="checkbox"/> | Bracket | 2 |
| <input type="checkbox"/> | Rubber | 4 |

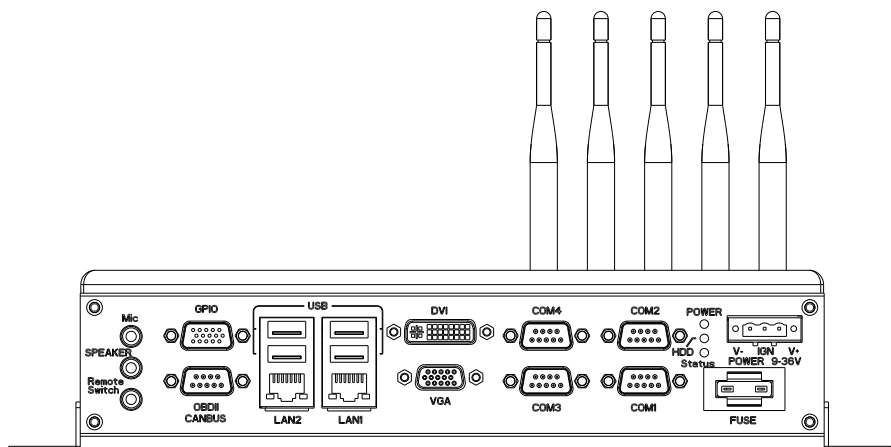
1.3. System Dissection

1.3.1. Dimensions

(Unit: mm)



1.3.2. Front I/O Panel



Mic (Pink)

Microphone input jack.

SPEAKER (Green)

Line out phone jack.

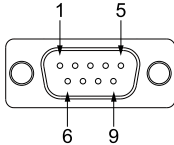
Remote Switch (Blue)

SPST (Single Pole, Single Throw) switch input.

GPIO

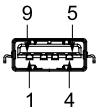
|  GPIO DB15 Cable | Pin # | Definition | Wire Color | Pin # | Definition | Wire Color |
|--|-------|------------|-------------|-------|------------|-------------|
| | 1 | GPO0 | Brown | 2 | GPO1 | Orange |
| | 3 | GPO2 | Green | 4 | GPO3 | Blue |
| | 5 | GND | Black | 6 | GND | Gray |
| | 7 | N/A | Red/White | 8 | N/A | White |
| | 9 | GND | Red | 10 | N/A | Purple |
| | 11 | GPI4 | Light Green | 12 | GPI5 | Light Blue |
| | 13 | GPI6 | Pink | 14 | GPI7 | Brown/White |
| | 15 | EXTPWR | Yellow | | | |

OBDDII CANBUS

|  | ODBDII | | AIV-APL1V1FL-PT1OB | | AIV-APL1V1FL-PT1 | |
|---|--------|------------|--------------------|-----------------|------------------|--------|
| | Pin # | Signal | Pin # | Signal | Pin # | Signal |
| | 1 | GND | 1 | GND | 1 | N/A |
| | 2 | GND | 2 | GND | 2 | N/A |
| | 3 | CAN_H | 3 | CAN_H | 3 | CAN_H |
| | 4 | K_LINE | 4 | K_LINE(RSV) | 4 | N/A |
| | 5 | CAN_L | 5 | CAN_L | 5 | CAN_L |
| | 6 | J1850_BUS- | 6 | J1850_BUS-(RSV) | 6 | N/A |
| | 7 | J1850_BUS+ | 7 | J1850_BUS+(RSV) | 7 | N/A |
| | 8 | L_LINE | 8 | L_LINE(RSV) | 8 | N/A |
| | 9 | DLC_RAW | 9 | DLC_RAW | 9 | N/A |
| | 10 | N/A | 10 | N/A | 10 | N/A |

USB

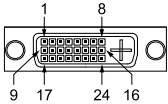
Standard USB 3.0 Type-A connectors.

|  | 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 + |

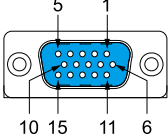
LAN1, LAN2

|  | LED | Light | Status |
|---|------|--------|-------------|
| | LED1 | Off | 10Mbps |
| | | Green | 100Mbps |
| | | Orange | 1000Mbps |
| | LED2 | Yellow | Link/Active |
| | | Off | LAN Off |

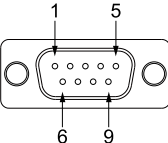
DVI

|  | Pin # | Signal | Pin # | Signal |
|---|-------|----------|-------|-----------|
| | C1 | VGA_RED | C2 | VGA_GREEN |
| | C3 | VGA_BLUE | C4 | VGA_HSYNC |
| | D1 | DATA2- | D2 | DATA2+ |
| | D3 | GND | D4 | VGA_SCL |
| | D5 | VGA_SDA | D6 | DDCCLK |
| | D7 | DDCDATA | D8 | VGA_VSYNC |
| | 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- |

VGA

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

COM1 ~ COM4

|  | | COM1~3, COM4 (RS232) | COM4 (RS422) | COM4 (RS485) |
|---|-------|-------------------------|--------------|--------------|
| | Pin # | Signal | Signal | Signal |
| | 1 | DCD | TX- | DATA- |
| | 2 | SIN | TX+ | DATA+ |
| | 3 | SOUT | RX+ | |
| | 4 | DTR | RX- | |
| | 5 | GND | | |
| | 6 | DSR | | |
| | 7 | RTS | | |
| | 8 | CTS | | |
| | 9 | RI | | |

Status/HDD/Power LED Display

|  | LED | Light | Display |
|---|-----|--------|-----------|
| | G | Green | Status |
| | G | Green | HDD |
| | 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 Flashing Numbers | Status |
|----------------------|-----------------------------|
| 0 (Constant On) | Power output runs normally. |
| 1 | Standby Mode (System off) |
| 3 | Power On Delay |
| 5 | Boot Up Delay |
| 6 | Soft Off Delay |
| 4 | Shutdown Delay |
| 2 | Hard 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 Flashing 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. |

DC Power In

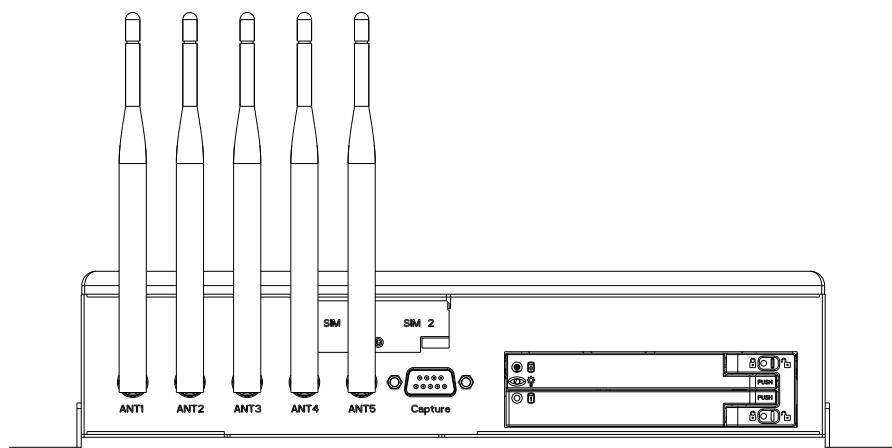
| 9V ~ 36V DC input connector Terminal Block: 3 pin Pitch: 5.08mm | Pin # | Signal |
|--|-------|-------------------------|
| | V+ | 9V ~ 36V DC Power Input |
| | IGN | Ignition On (Hi Active) |
| | V- | GND |

Blade-type Fuse Holder

| Car Battery | Blade-type fuse suggestion | Remarks |
|-------------|----------------------------|---|
| 12V System | CONQUER ATQ-10 | Voltage Rating: 36V; Current Rating: 15A |
| 24V System | CONQUER ATQ-5 | Voltage Rating: 36V; Current Rating: 15A |

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

1.3.3. Rear I/O Panel



Antenna Sockets

Reserved for installation of optional SMA-type antennas.

SIM Card Holders

Reserved for installation of your SIM cards.

Capture

Reserved for installation of optional capture card.

HDD Bays

Reserved for installation of your hard disks.

2. Components Assembly

2.1. 2.5" SATA SSD Installation

Step 1: Push the latch right, a white circle appears. The door is unlocked. Push the door-end marked with the word "PUSH" to let the door opened.



To install an SSD of 7mm thin, you will need to place two 2.5mm-thick spacers atop the SSD so as to fit in the 9.5mm bay.



Step 2: Insert your SSD into the tray. (The contact pins face inward.)



Step 3: Firmly close the door so that the SSD will be slid into its contact position. (Failing to do so could cause a loose contact with the SSD.)



Step 4: Push the latch left, a red circle appears. The door is locked.



2.2. SIM Card Installation

Step 1: Remove the screw that secure the cover plate.



Step 2: Lift the cover plate up a little to open the door. Gently hold the left side of the plate up a little and take it away from the notch.



Step 3: Insert your SIM card. Secure the cover plate.



2.3. Antenna Connection

After havinf installed your wireless module into the mainboard and the antenna socket, you may connect your antennas needed according to your system configuration.

Step 1: Insert the antenna plug into the antenna socket.



Step 2: Turn the antenna body upright.

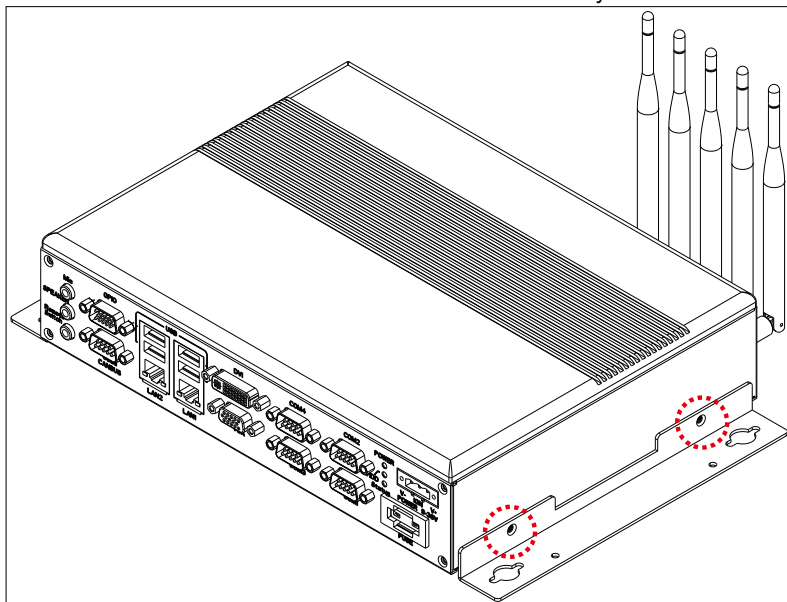


Step 3: Connect your antennas needed according to your system configuration. The photo shows antenna from left to right are: 4G LTE, GPS, WiFi, BT.



2.4. Brackets & Rubber Pads Installation

Step 1: Attach two brackets with screws to each side of the body.



Step 2: Attach four rubber pads to the bracket holes.



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

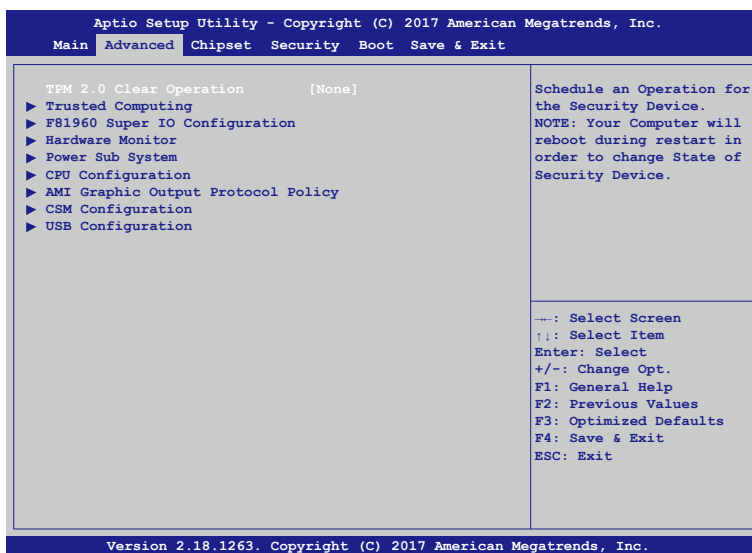
The BIOS setup main menu includes some options. Use the [Up/Down] arrow key to highlight the option, and then press the <Enter> key to select the item and configure the functions.

| Aptio Setup Utility - Copyright (C) 2017 American Megatrends, Inc. | | |
|---|--|--|
| Main Advanced Chipset Security Boot Save & Exit | | |
| BIOS Information BIOS Vendor American Megatrends Core Version 5.12 Compliancey UEFI 2.5; PI 1.4 Project Version APL1V1FL 011-004 Build Date and Time 07/06/2017 11:22:33 | | Set the Date. Use Tab to switch between Date elements. Default Ranges: Year: 2005-2099 Months: 1-12 Days: dependent on month |
| Platform firmware Information BXT SOC B0 MRC Version 0.56 CPU Flavor BXT Notebook/Desktop ... Board ID 0xbow Hill CRB (06) Fab ID FAB A | | |
| Memory Information Total Memory 4096 MB Memory Speed 1866 MHz | | --: Select Screen ++: Select Item Enter: Select +/-: Change Opt. F1: General Help F2: Previous Values F3: Optimized Defaults F4: Save & Exit ESC: Exit |
| System Date [Sun 01/01/2017] System Time [11:22:33] | | |
| Version 2.18.1263. Copyright (C) 2017 American Megatrends, Inc. | | |

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

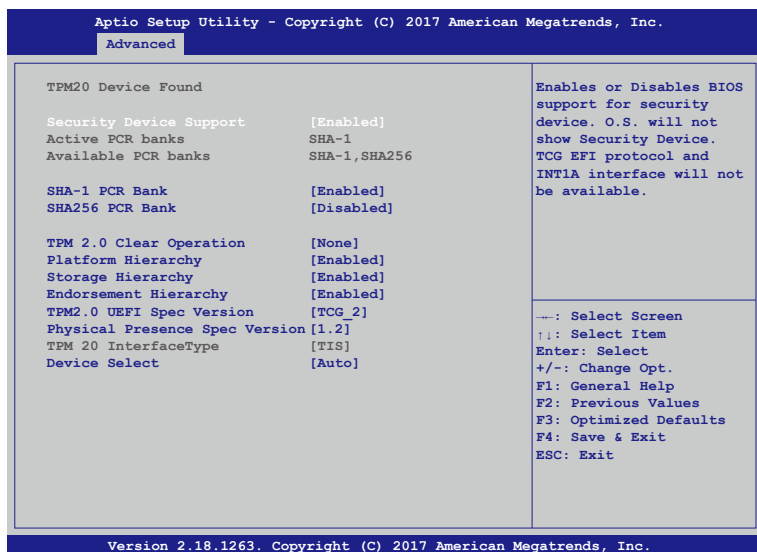


- **TPM 2.0 Clear Operation**
Schedule an Operation for the Security Device.
NOTE: Your Computer will reboot during restart in order to change State of Security Device.
- **Trusted Computing**
Trusted Computing Settings.
- **F81960 Super IO Configuration**
System Super IO Chip Parameters.
- **Hardware Monitor**
Monitor hardware status.
- **Power Sub System**
Power Sub System.

- **CPU Configuration**
CPU Configuration Parameters.
- **AMI Graphic Output Protocol Policy**
User Select Monitor Output by Graphic Output Protocol.
- **CSM Configuration**
Compatibility Support Module Configuration. Enable/Disable Option ROM execution settings, etc.
- **USB Configuration**
USB Configuration Parameters.

3.2.1. Trusted Computing

Set trusted computing settings

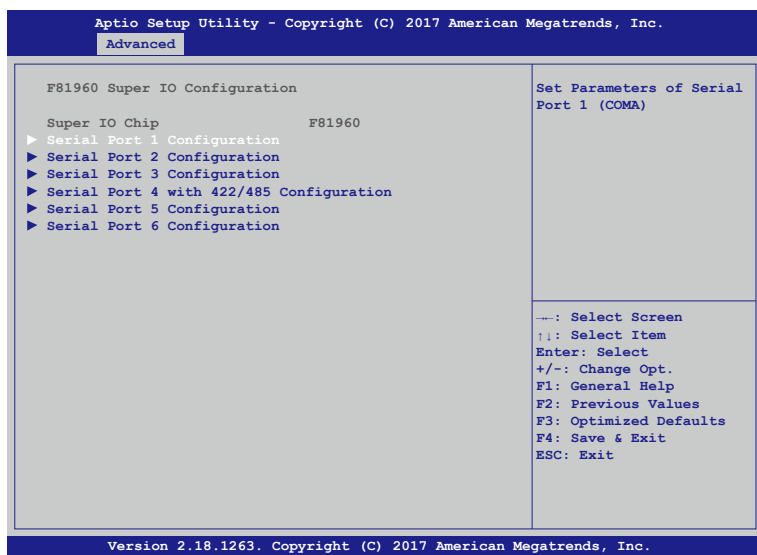


- **Security Device Support**
Enables or Disables BIOS support for security device. O.S. will not show Security Device. TCG EFI protocol and INT1A interface will not be available.
- **SHA-1 PCR Bank**
Enables or Disables SHA-1 PCR Bank.
- **SHA256 PCR Bank**
Enables or Disables SHA256 PCR Bank.
- **TPM 2.0 Clear Operation**
Schedule an Operation for the Security Device. NOTE: Your Computer will reboot during restart in order to change State of Security Device.

- **Platform Hierarchy**
Enables or Disables Pateform Hierarchy.
- **Storage Hierarchy**
Enables or Disables Storage Hierarchy.
- **Endorsement Hierarchy**
Enables or Disables Endorsement Hierarchy.
- **TPM2.0 UEFI Spec Version**
Select the TCG2 Spec Version Support,
TCG_1_2: The Compatible mode for Win8/Win10.
TCG_2: Support new TCG2 protocol and event format for Win10 or later.
- **Physical Presence Spec Version**
Select to Tell O.S. to support PPI Spec Version 1.2 or 1.3. Note some HCK tests might not support 1.3.
- **Device Select**
TPM 1.2 will restrict support to TPM 1.2 devices. TPM2.0 will restrict support to TPM 2.0 devices, Auto will support both with the default set to TPM2.0 devices if not found, TPM1.2 devices will be enumerated

3.2.2. F81960 Super IO Configuration

Set system super IO chip parameters.

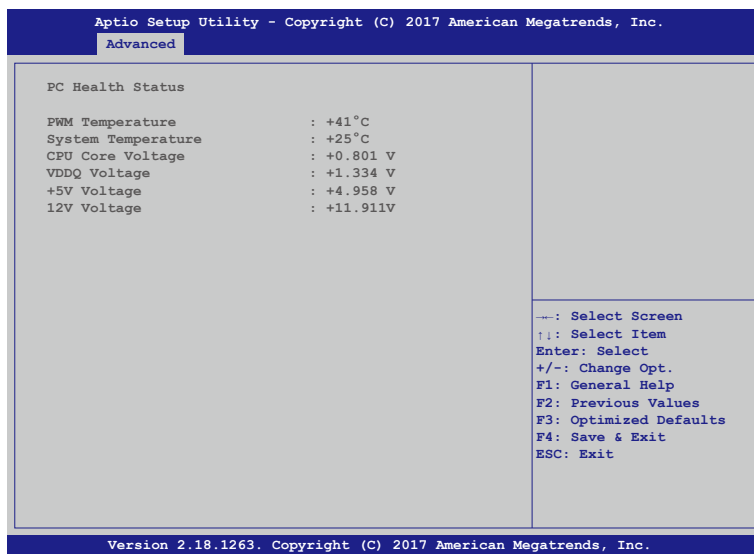


- **Serial Port 1 Configuration**
Set Parameters of Serial Port 1 (COMA).

- **Serial Port 2 Configuration**
Set Parameters of Serial Port 2 (COMB).
- **Serial Port 3 Configuration**
Set Parameters of Serial Port 3 (COMC).
- **Serial Port 4 with 422/485 Configuration**
Set Parameters of Serial Port 4 (COMD).
- **Serial Port 5 Configuration**
Set Parameters of Serial Port 5 (COME).
- **Serial Port 6 Configuration**
Set Parameters of Serial Port 6 (COMF).

3.2.3. Hardware Monitor

Display hardware monitor status.

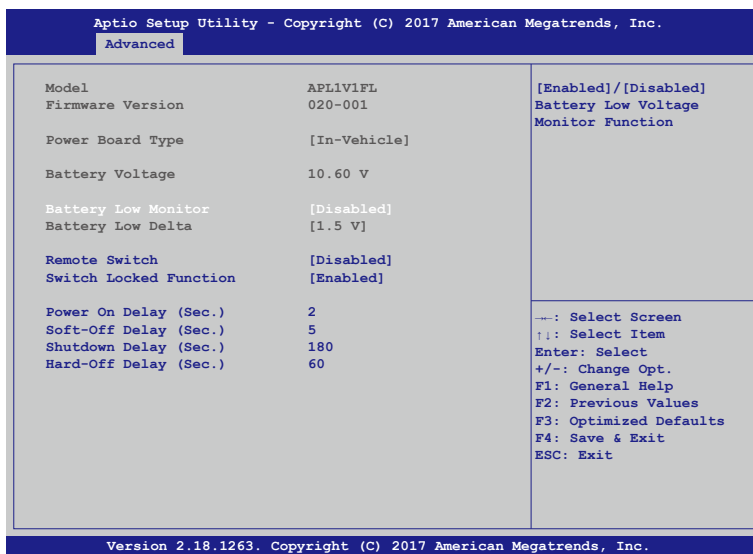


- **PWM Temperature**
This item displays the PWM temperature.
- **System Temperature**
This item displays the system temperature.
- **CPU Core Voltage**
This item displays the VCORE voltage.
- **VDDQ Voltage**
This item displays the VDDQ voltage.

- **+5V Voltage**
This item displays the +5V voltage.
- **12V Voltage**
This item displays the +12V voltage.

3.2.4. Power Sub System

Dsplay hardware monitor status.



- **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.
[Disabled]: Ignition only.
[Enabled]: Ignition+Remote Switch.
- **Switch Locked Function**
Enables or disables the function of switch lock.
[Disabled]: No switch locked.
[Enabled]: All switch will locked 3 min after power on.
- **Power On Delay (Sec.)**
The delay between power on and system work.

2 seconds to 60 seconds with 1 second increment.

- **Soft-Off Delay (Sec.)**

The delay before system shutdown.

0 second to 3600 seconds with 1 second increment.

- **Shutdown Delay (Sec.)**

The delay between system shutdown and system off.

120 seconds to 3600 seconds with 1 second increment.

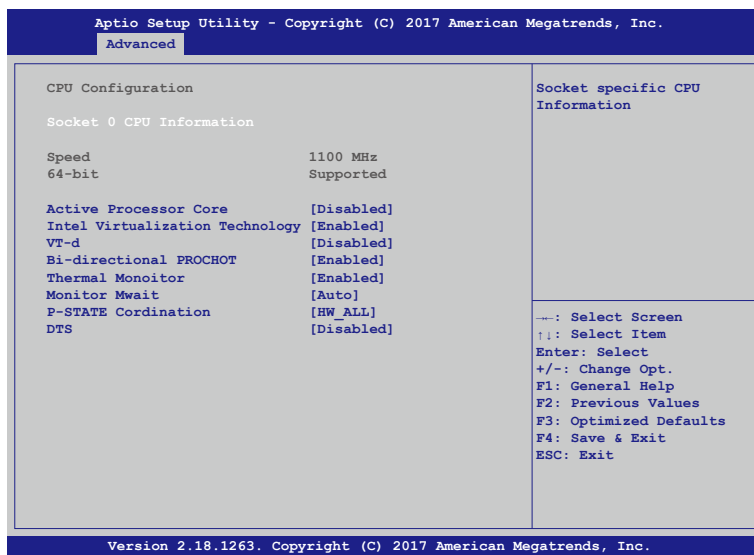
- **Hard-Off Delay (Sec.)**

The delay before all power off.

0 second to 3600 seconds with 1 second increment.

3.2.5. CPU Configuration

Set CPU configuration parameters.



- **Socket 0 CPU Information**

Socket specific CPU Information.

- **Active Processor Core**

Number of cores to enable in each processor package.

- **Intel Virtualization Technology**

When enabled, a VMM can utilize the additional hardware capabilities provided by Vanderpool Technology.

- **VT-d**

Enable/Disable CPU VT-d.

- **Bi-directional PROCHOT**

When a processor thermal sensor trips (either core), the PROCHOT# will be driven. If bi-direction is enabled, external agents can drive PROCHOT# to throttle the processor.

- **Thermal Monitor**

Enable/Disable Thermal Monitor.

- **Monitor Mwait**

Enable/Disable Monitor Mwait.

- **P-STATE Coordination**

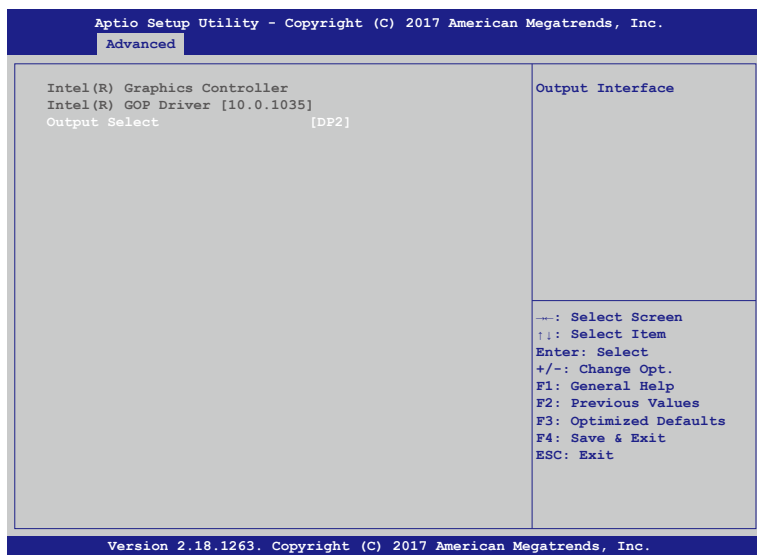
Change P-STATE Coordination type.

- **DTS**

Enabled/Disable Digital Thermal Sensor.

3.2.6. AMI Graphic Output Protocol Policy

User Select Monitor Output by Graphic Output Protocol.

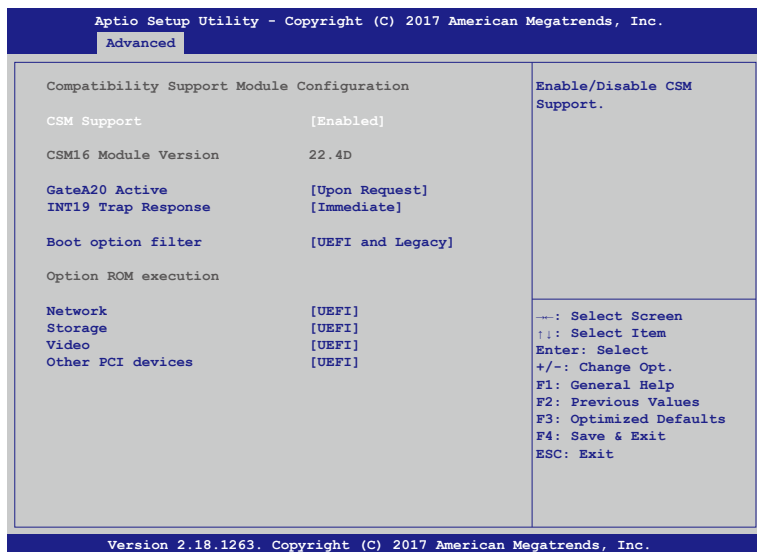


- **Output Select**

Select output interface.

3.2.7. CSM Configuration

Set CSM configuration parameters.



- **CSM Support**
Enable/Disable CSM support.
- **GateA20 Active**
[UPON REQUEST]: GA20 can be disabled using BIOS services.
[ALWAYS]: Do not allow disabling GA20; this option is useful when any RT code is executed above 1MB.
- **INT19 Trap Response**
BIOS reaction on INT19 trapping by Option ROM.
[IMMEDIATE]: Execute the trap right away.
[POSTONED]: Execute the trap during legacy boot.
- **Boot option filter**
This option controls Legacy/UEFI ROMs priority.
- **Network**
Controls the execution of UEFI and Legacy PXE OpROM.
- **Storage**
Controls the execution of UEFI and Legacy Storage OpROM.
- **Video**
Controls the execution of UEFI and Legacy Video OpROM.

- **Other PCI devices**

Determines OpROM execution policy for devices other than Network, Storage, or Video.

3.2.8. USB Configuration

Set USB configuration parameters.

| Aptio Setup Utility - Copyright (C) 2017 American Megatrends, Inc. | |
|---|-----------|
| Advanced | |
| USB Configuration | |
| USB Module Version | 16 |
| USB Controllers: | 22.4D |
| 1 XHCI | |
| USB Devices: | |
| 1 Drive, 1 Keyboard, 1 Mouse | |
| Legacy USB Support | [Enabled] |
| XHCI Hand-off | [Enabled] |
| USB Mass Storage Driver Support | [Enabled] |
| USB hardware delays and time-outs: | |
| USB transfer time-out | [20 sec] |
| Device reset time-out | [20 sec] |
| Device power-up delay | [Auto] |
| Mass Storage Devices: | |
| SSD | [Auto] |
| Enables Legacy USB support. AUTO option disables legacy support if no USB devices are connected. DISABLE option will keep USB devices available only for EFI application. | |
| --: Select Screen ++: Select Item Enter: Select +/-: Change Opt. F1: General Help F2: Previous Values F3: Optimized Defaults F4: Save & Exit ESC: Exit | |

Version 2.18.1263. Copyright (C) 2017 American Megatrends, Inc.

- **Legacy USB Support**

Enables Legacy USB support. AUTO option disables legacy support if no USB devices are connected. DISABLE option will keep USB devices available only for EFI applications.

- **XHCI Hand-off**

This is a workaround for OSes without XHCI hand-off support. The XHCI ownership change should be claimed by XHCI driver.

- **USB Mass Storage Driver Support**

Enable/Disable USB Mass Storage Driver Support.

- **USB transfer time-out**

The time-out value for Control, Bulk, and Interrupt transfers.

- **Device reset time-out**

USB mass storage device Start Unit command time-out.

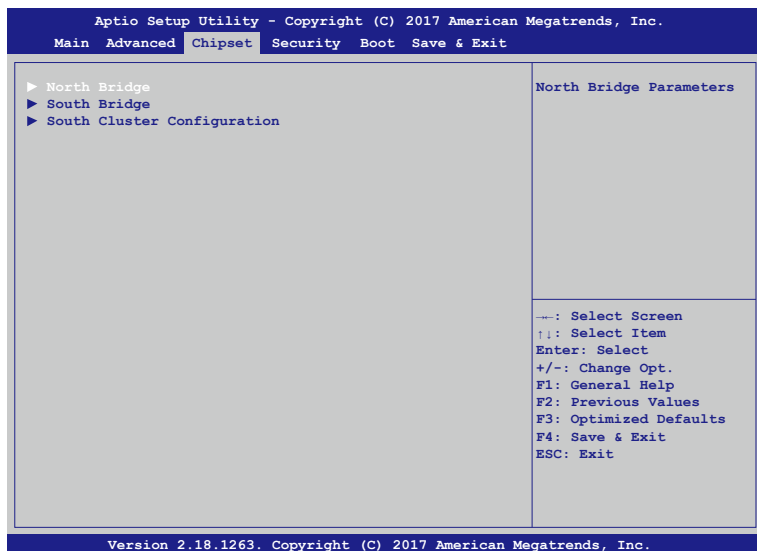
- **Device power-up delay**

Maximum time the device will take before it properly reports itself to the Host Controller. 'AUTO' uses default value: for a Root port it is 100ms, for a Hub port the delay is taken from Hub descriptor.

- **Mass Storage Devices**

Mass storage device emulation type. 'AUTO' enumerates devices according to their media format. Optical drives are emulated as 'CDROM', drives with no media will be emulated according to a drive type.

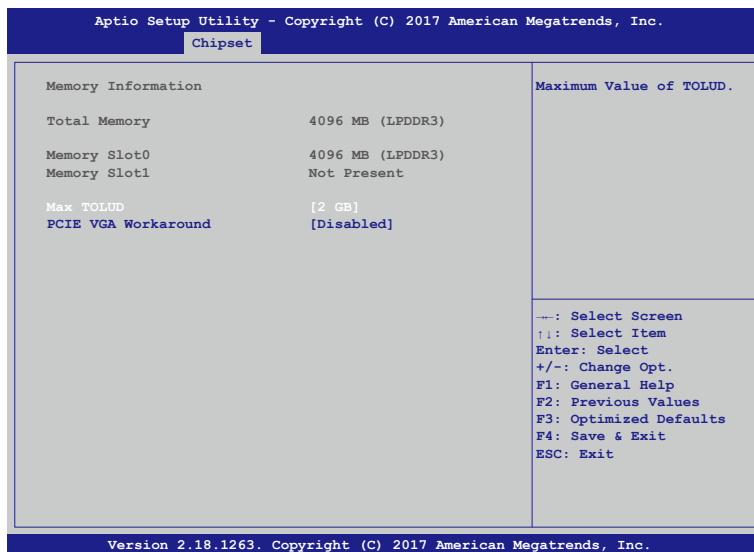
3.3. Chipset Setup



- **North Bridge**
North Bridge Parameters.
- **South Bridge**
South Bridge Parameters.
- **South Cluster Configuration**
South Cluster Configuration.

3.3.1. North Bridge

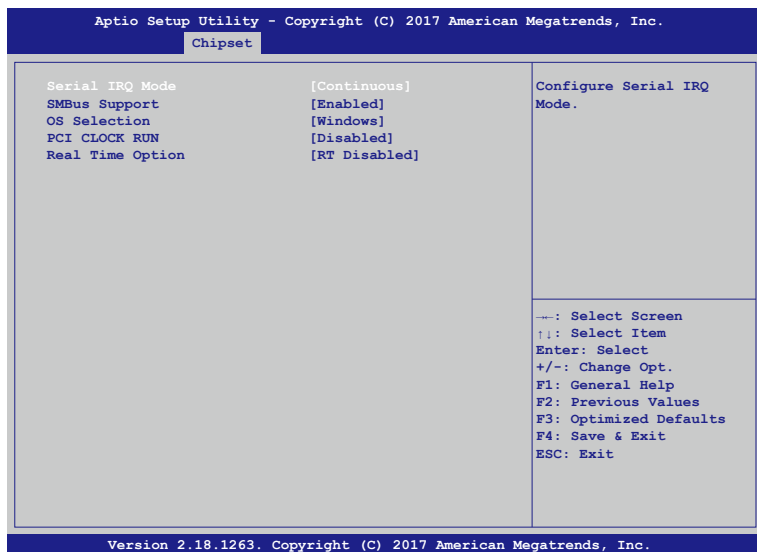
Set North Bridge configuration parameters.



- **Max TOLUD**
Maximum Value of TOLUD.
- **PCIE VGA Workaround**
Enable it if your PCIe card cannot boot to DOS. This is for Test only.

3.3.2. South Bridge

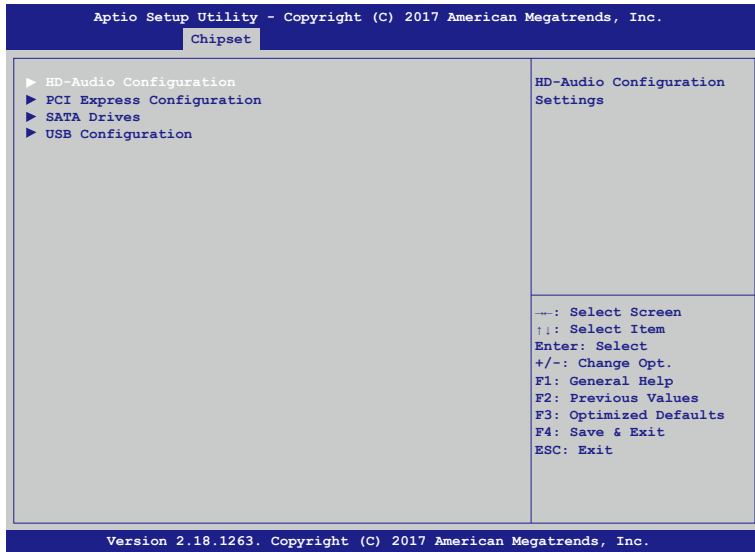
Set South Bridge configuration parameters.



- **Serial IRQ Mode**
Configure Serial IRQ Mode.
- **SMBus Support**
Enable/Disable SMBus Support.
- **OS Selection**
Select the target OS.
- **PCI CLOCK RUN**
Enables CLKRUN# logic to stop PCI clocks.
- **Real Time Option**
Select Read-Time Enable and IDI Agent Real-Time Traffic Mask Bits.

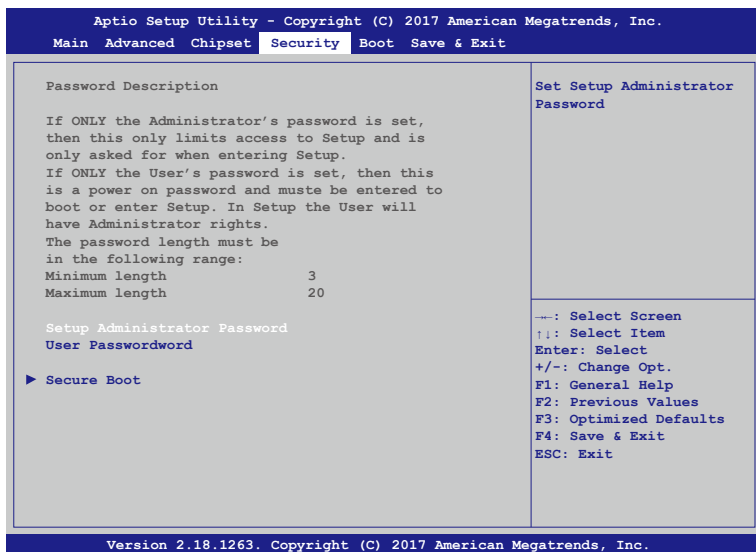
3.3.3. South Cluster Configuration

Set South Cluster configuration parameters.



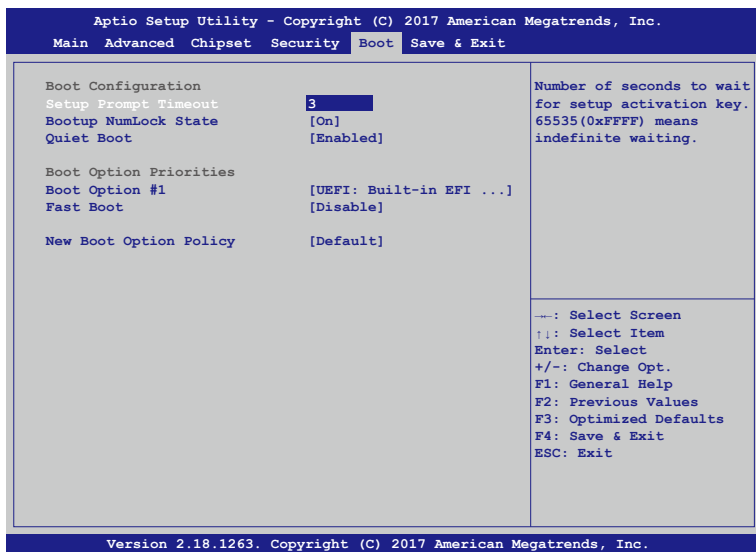
- **HD-Audio Configuration**
HD-Audio Configuration Settings.
- **PCI Express Configuration**
PCI Express Configuration Settings.
- **SATA Drives**
Press <Enter> to select the SATA Device Configuration Setup options.
- **USB Configuration**
USB Configuration Settings.

3.4. Security Setup



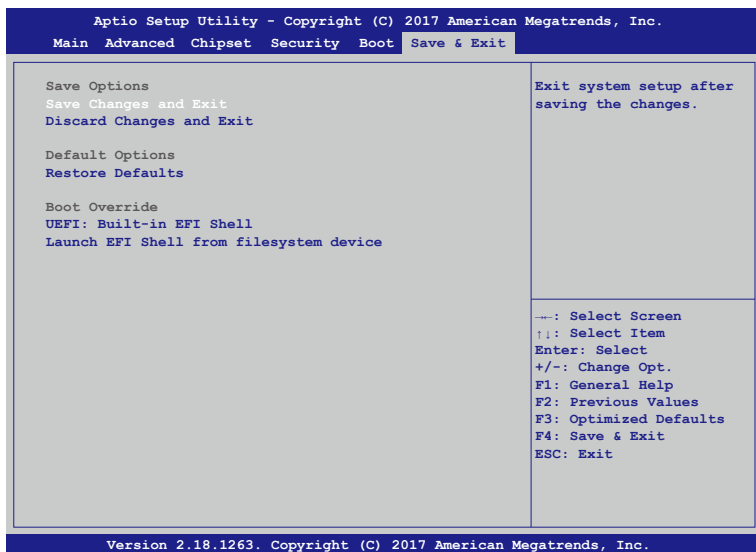
- **Setup Administrator Password**
Set Setup Administrator Password.
- **User Password**
Set User Password.
- **Secure Boot**
Customizable Secure Boot settings.

3.5. Boot Setup



- **Setup Prompt Timeout**
Number of seconds to wait for setup activation key.
65535(0xFFFF) means indefinite waiting.
- **Bootup NumLock State**
Select the keyboard NumLock state.
- **Quiet Boot**
Enables or disables Quiet Boot option.
- **Boot Option #1**
Sets the system boot order.
- **Fast Boot**
Enable or Disable FastBoot features.
Most probes are skipped to reduce time cost during boot.
- **New Boot Option Policy**
Controls the placement of newly detected UEFI boot options.

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.
- **Launch EFI Shell from filesystem device**
Attempts to Launch EFI Shell application (shell.efi) from one of the available filesystem devices.

4. Function Description

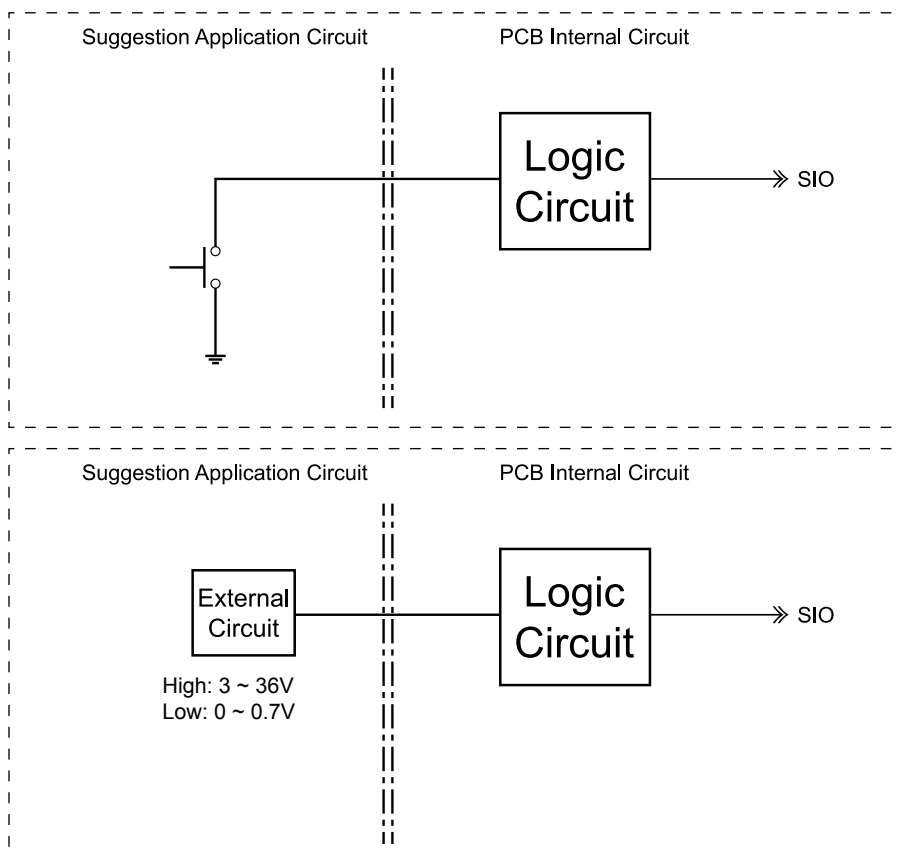
4.1. Power input connection

AIV-APL1V1FL Series needs +9~36V 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.

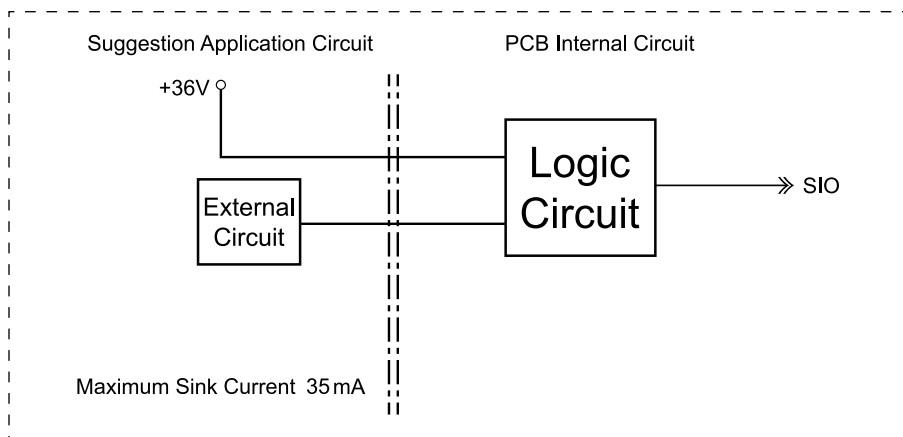
The following diagrams state how to connect the digital inputs to devices on the embedded system.



4.3. Digital Outputs

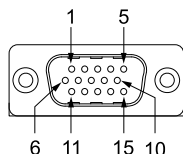
There are 4 clamped diode protection digital outputs on GPIO1 connector. You can control the output status of these digital outputs through the software API. The four digital outputs are capable sink maximum 35 mA current for each channel and maximum output voltage is 36V. The output reference voltage 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 | N/A | 8 | N/A |
| 9 | GND | 10 | N/A |
| 11 | GPI4 | 12 | GPI5 |
| 13 | GPI6 | 14 | GPI7 |
| 15 | EXTPWR | | |

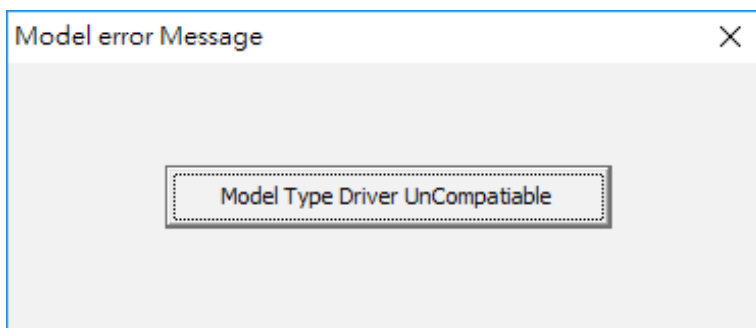


5. Driver and Utility Installation

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.

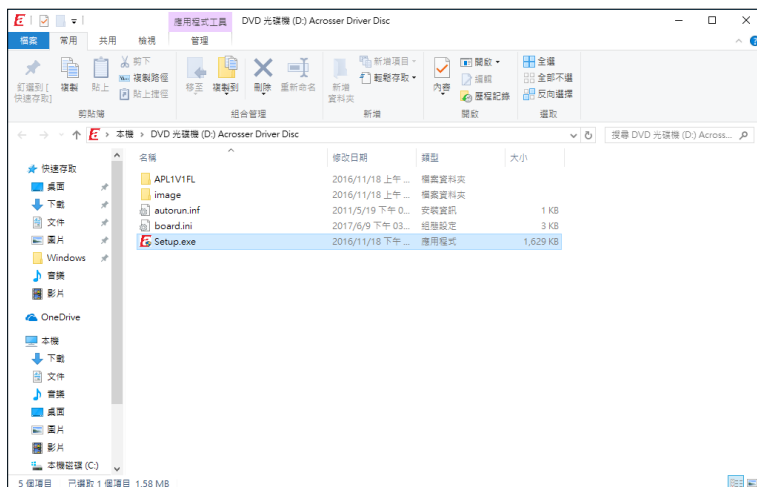


5.2. Windows Installation

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

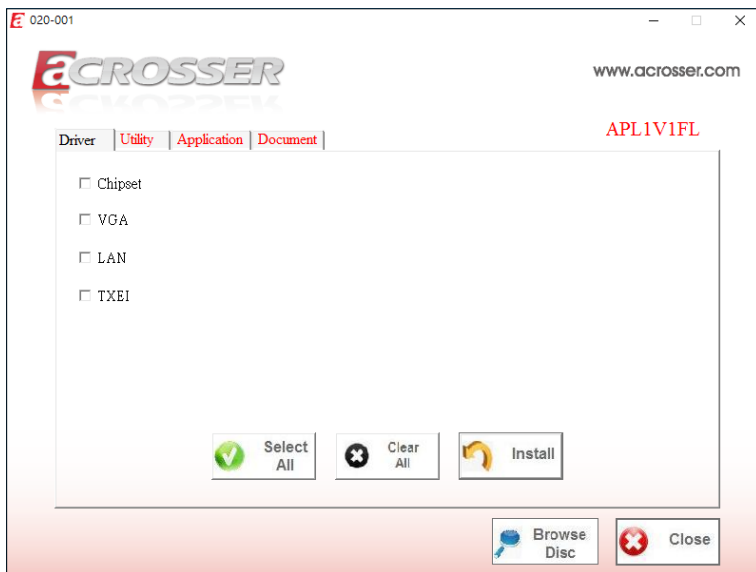


If not, enter the root folder of the Driver CD, run the execution file “Setup.exe”.

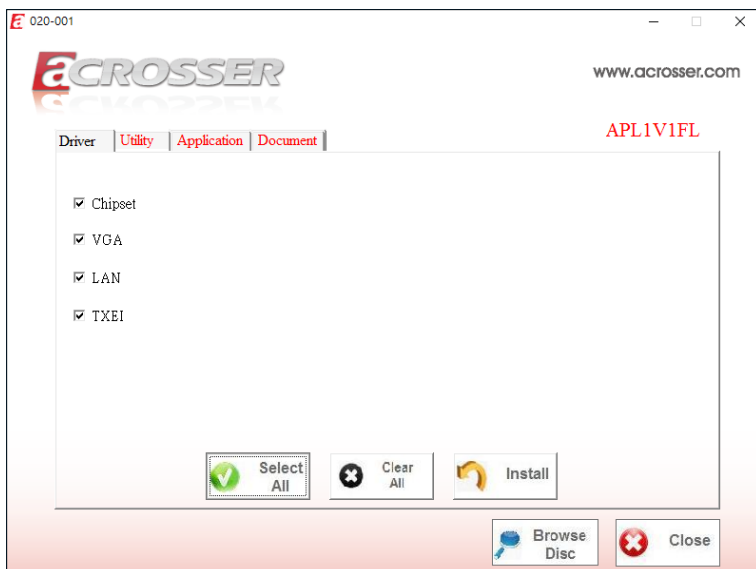


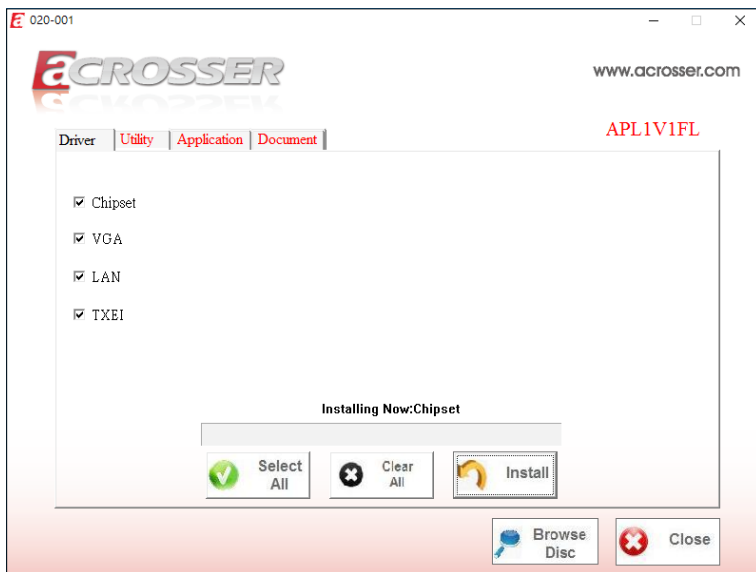
5.2.1. Driver Installation Page

Step 1: Select the “**Driver**” tab.

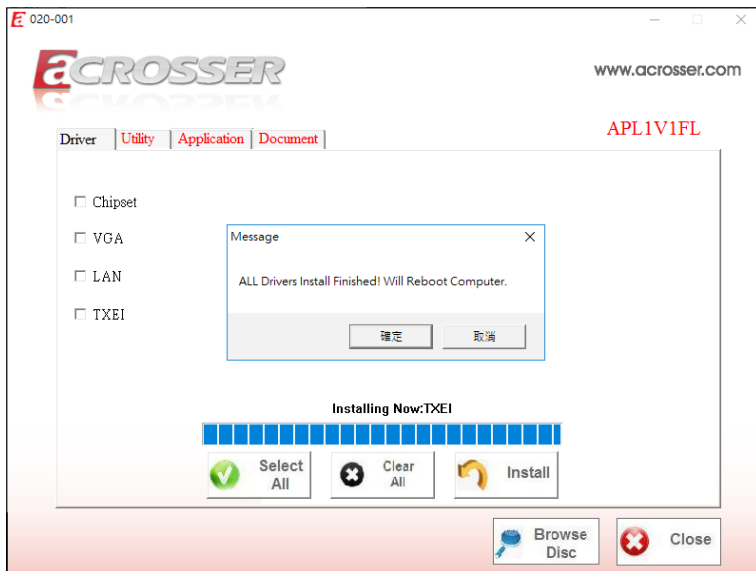


Step 2: Click the “**Select All**” button to select all the driver checkboxes, and then click “**Install**” button to start installing all the selected drivers.





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

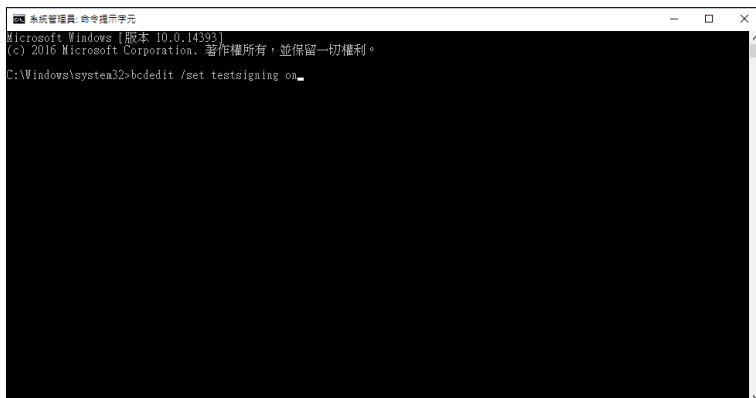


Note: Select the “**Clear All**” button will clear all the selections, and then you can select the driver you want to install one by one, but the “**Chipset**” driver has to be installed before installing all the others.

5.2.2. Utility Page

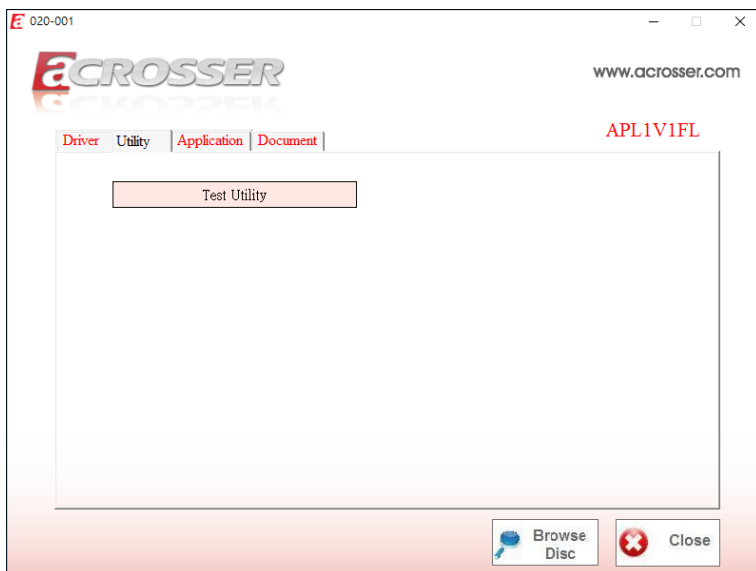
Before launching the utility, you should run the Windows test mode by running the command “**BCDEdit /set testsigning on**”, and restart the system.

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.

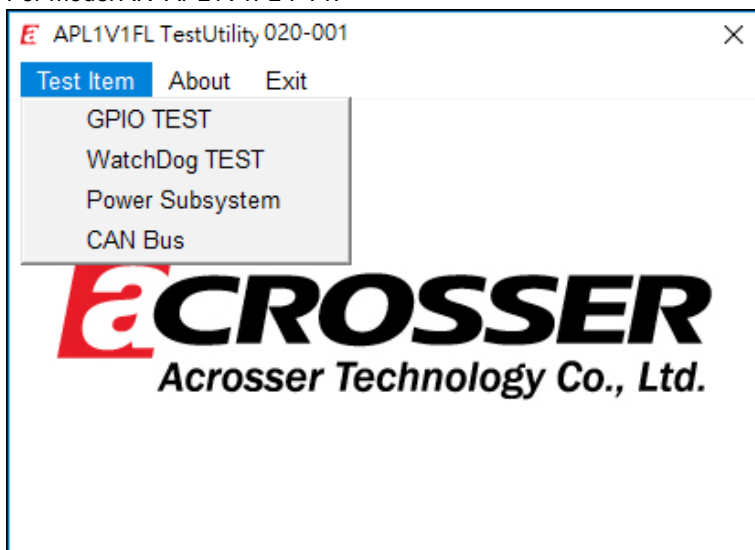


Step 2: The “**Test Utility**” screen appears.

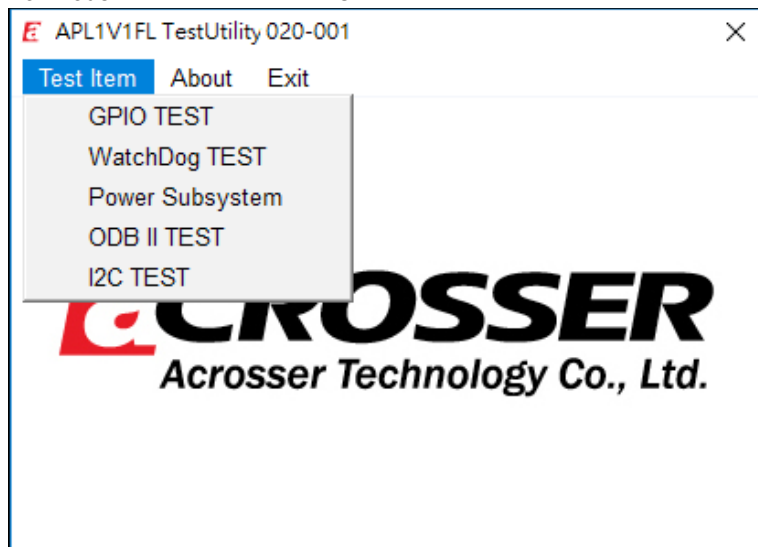


Click Test Item:

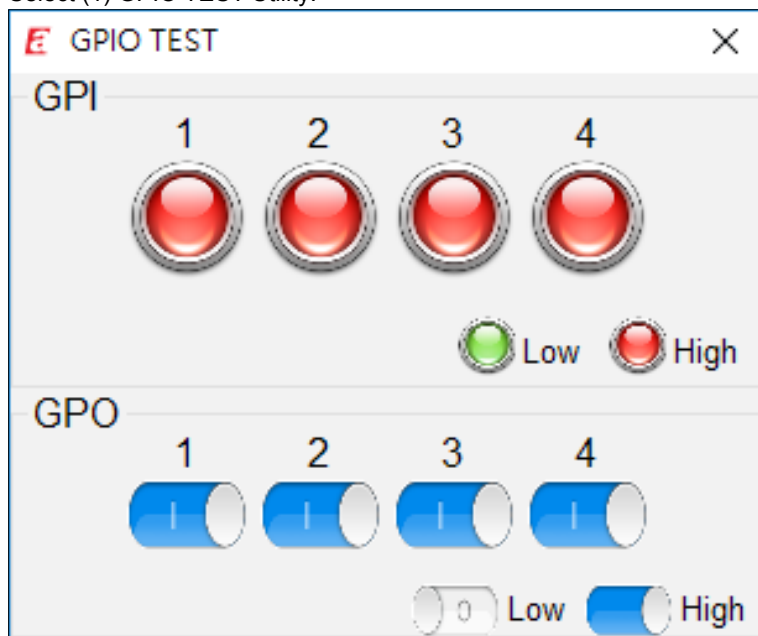
For model AIV-APL1V1FL-PT1:



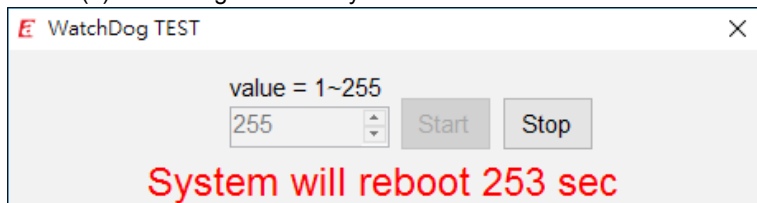
For model AIV-APL1V1FL-PT10B:



Select (1) GPIO TEST Utility:



Select (2) WatchDog TEST Utility:



WatchDog TEST

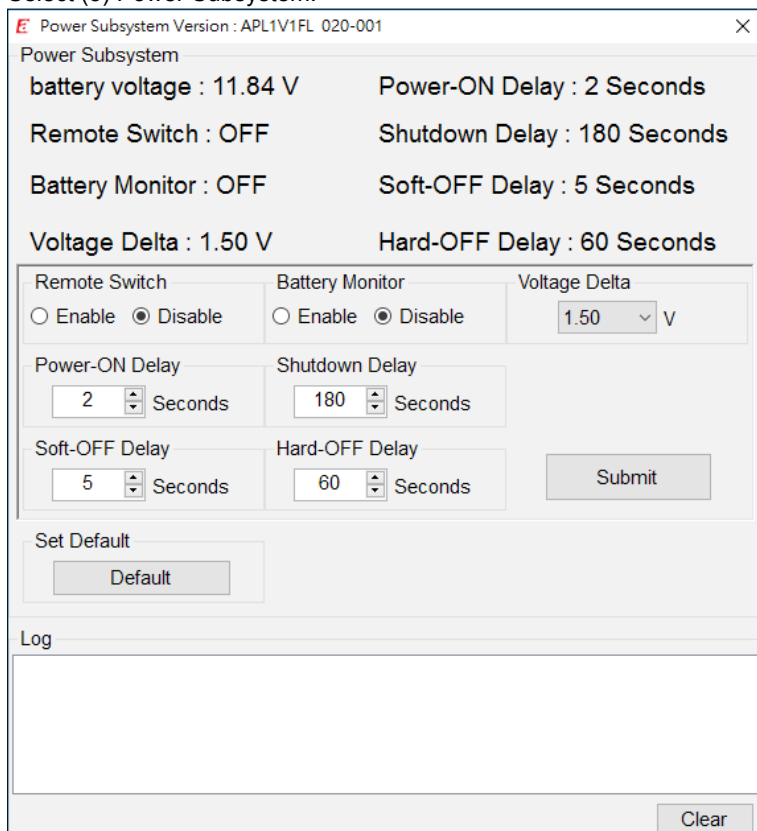
value = 1~255

255

Start Stop

System will reboot 253 sec

Select (3) Power Subsystem:



PowerSubsystem Version : APL1V1FL 020-001

Power Subsystem

battery voltage : 11.84 V Power-ON Delay : 2 Seconds

Remote Switch : OFF Shutdown Delay : 180 Seconds

Battery Monitor : OFF Soft-OFF Delay : 5 Seconds

Voltage Delta : 1.50 V Hard-OFF Delay : 60 Seconds

Remote Switch Battery Monitor Voltage Delta

☐ Enable ☒ Disable ☐ Enable ☒ Disable 1.50 V

Power-ON Delay Shutdown Delay

2 Seconds 180 Seconds

Soft-OFF Delay Hard-OFF Delay

5 Seconds 60 Seconds

Submit

Set Default

Default

Log

Clear

Select (4) CAN Bus: (For model AIV-APL1V1FL-PT1)

CAN Bus Version : APL1V1FL 020-001

CAN Bus

Baud Rate : 125K

Receive Mode : By Set Filter

Mask

| | | |
|-------|------|------|
| ID | 0 | 1 |
| Value | 0x00 | 0x00 |

Open Send/Get Message Window

Filter

| | | | | | | |
|-------|------|------|------|------|------|------|
| ID | 0 | 1 | 2 | 3 | 4 | 5 |
| Type | STD | STD | STD | STD | STD | STD |
| Value | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

Baud Rate

125K Set

Receive Mode

By Set Filter Set

Set Mask

ID

Value (Range: 0~1FFFFFFF Hex)

0 Set

Set Filter

Filter Type

ID


Value (Range: 0~1FFFFFFF Hex)

EXT 0 Set

Log

Clear

Or select (4) OBD II Test: (For model AIV-APL1V1FL-PT10B)


OBD II Test
×

Loop Test

Engine Coolant Temperature : 100
Barometric Pressure : 0xFE
Engine Total Fuel Used : 0x6000000
Engine Speed : 3173
Accelerator Pedal Position 1 : 5.30
Engine Intake Manifold #1 Pressure : 0.03
Tachograph Vehicle Speed : 0.84

Single Function Test

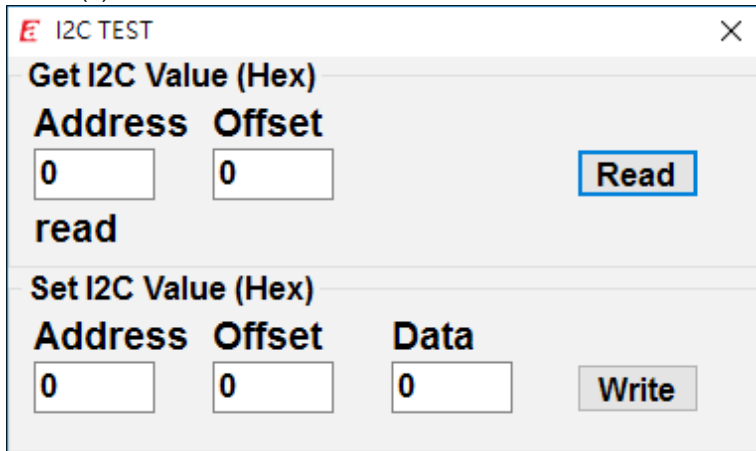
Engine Coolant Temperature
▼

Get

Log

Clear

Select (5) I2C TEST:



I2C TEST

Get I2C Value (Hex)

| Address | Offset | |
|---------|--------|-------------|
| 0 | 0 | Read |

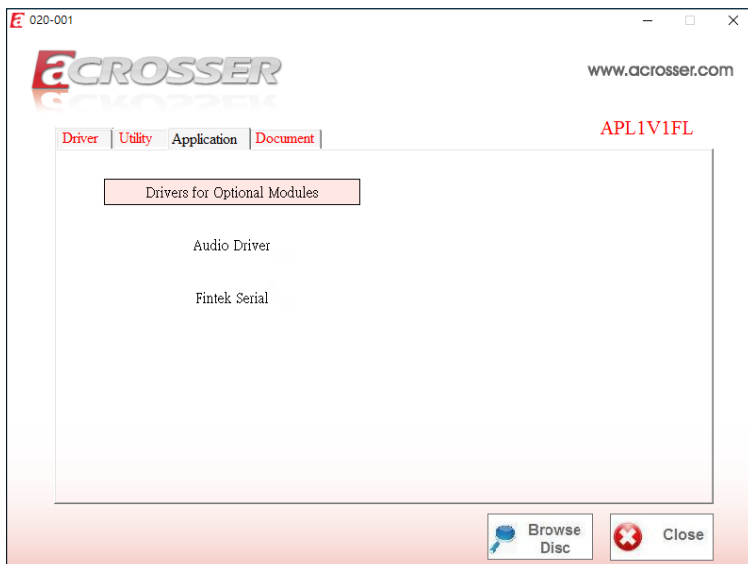
read

Set I2C Value (Hex)

| Address | Offset | Data | |
|---------|--------|------|--------------|
| 0 | 0 | 0 | Write |

5.2.3. Application Installation Page

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



020-001

aCROSSER

www.acrosser.com

Driver | **Utility** | **Application** | **Document**

APL1V1FL

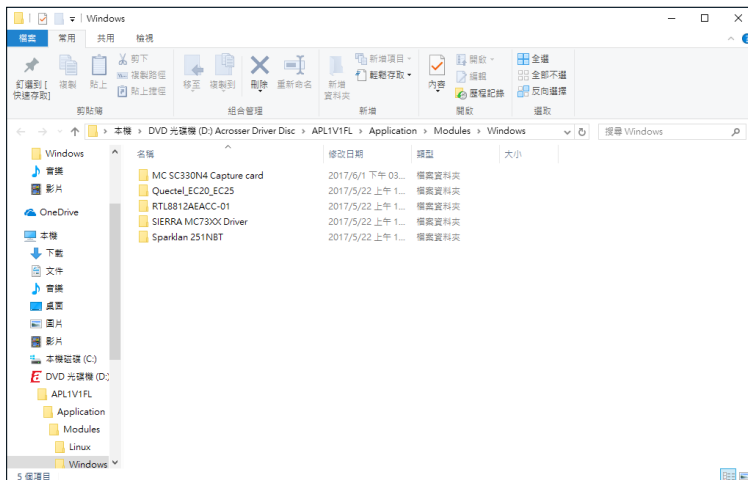
Drivers for Optional Modules

Audio Driver

Fintek Serial

Browse Disc **Close**

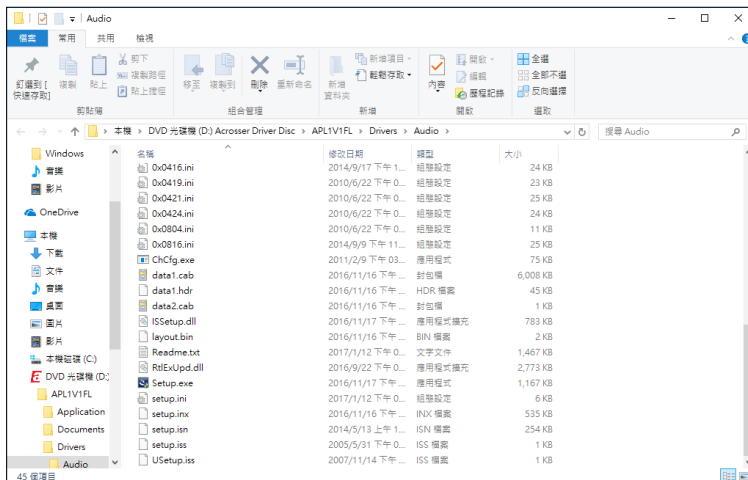
Step 2: Select the driver you want to install.



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



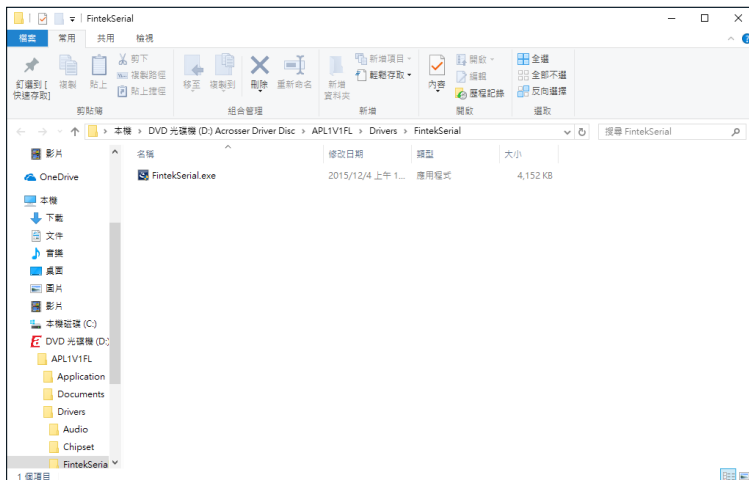
Step 4: Click **"Setup.exe"** to install audio driver.



Step 5: Select the **"Application"** tab. Click the **"Fintek Serial"** box.

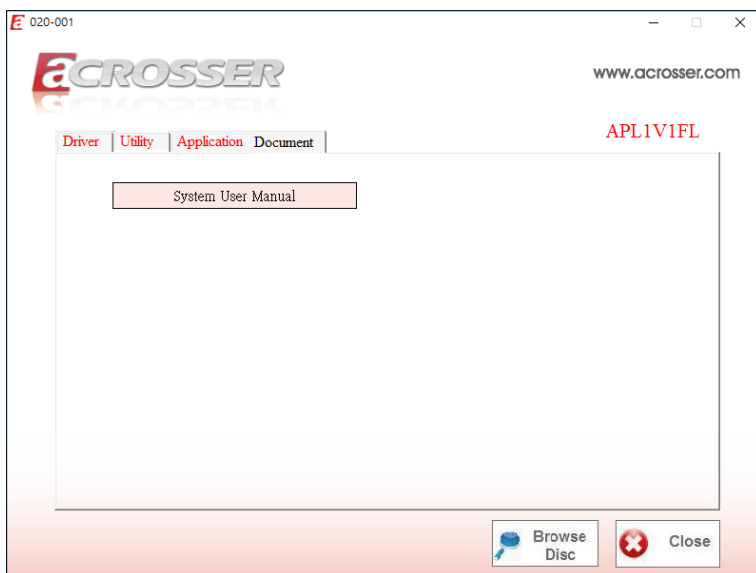


Step 6: Click **"FintekSerial.exe"** to install COM Port driver.



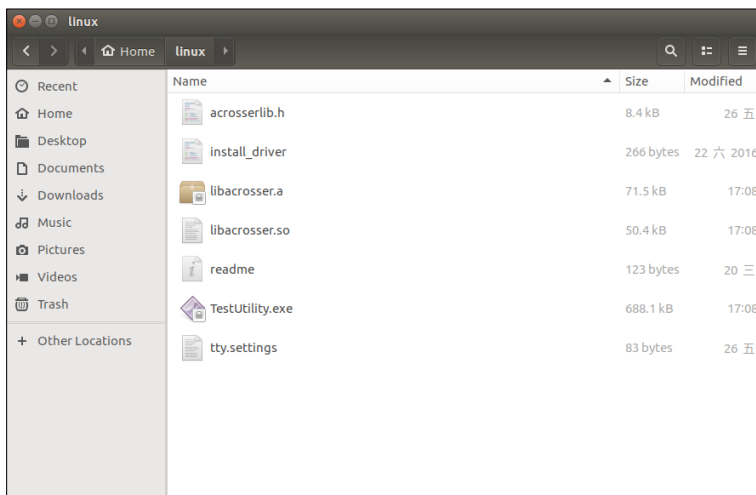
5.2.4. Document Page

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



5.3. Linux Configuration

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



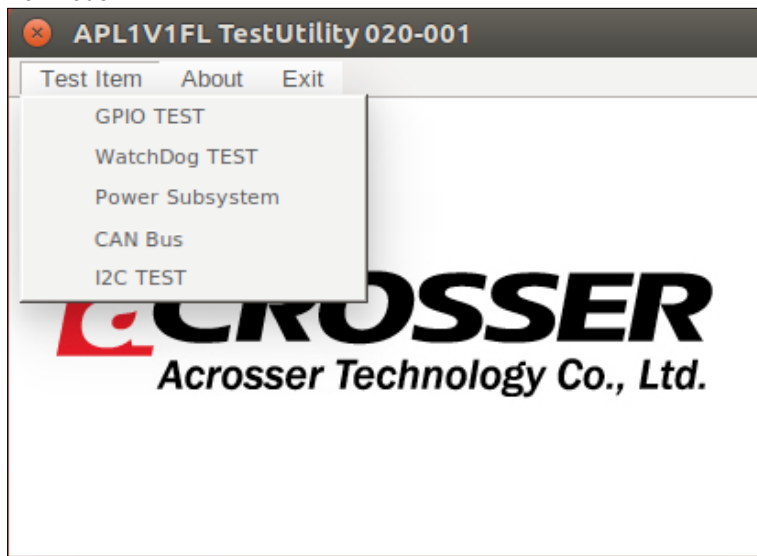
Run the **sudo mono TestUtility.exe**.

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

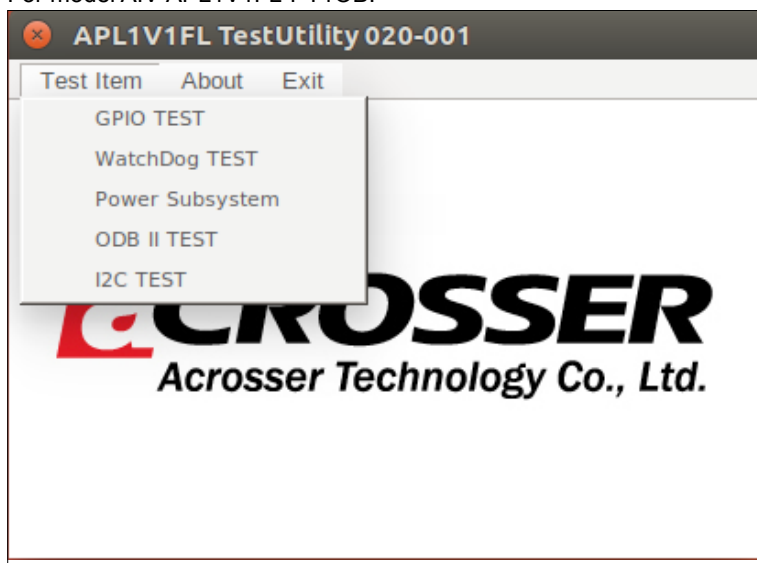


Click Test Item:

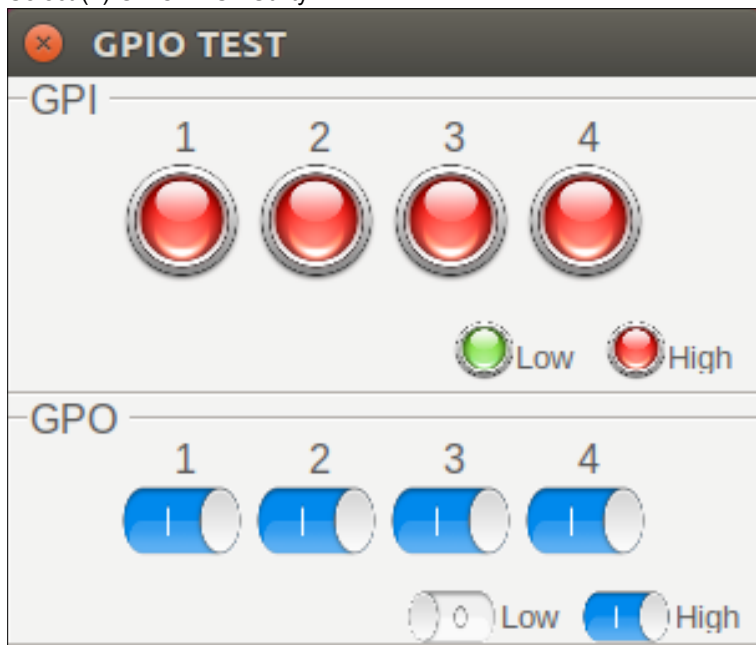
For model AIV-APL1V1FL-PT1:



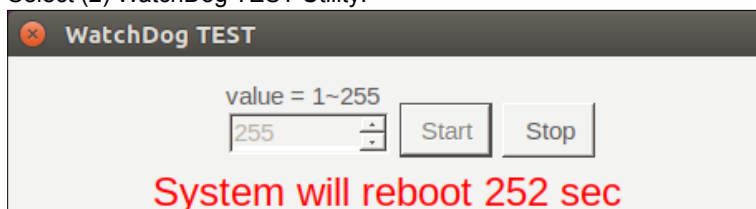
For model AIV-APL1V1FL-PT10B:



Select (1) GPIO TEST Utility:



Select (2) WatchDog TEST Utility:



Select (3) Power Subsystem:

Power Subsystem Version : APL1V1FL 020-001

Power Subsystem

battery voltage : 11.62 V
Power-ON Delay : 2 Seconds

Remote Switch : OFF
Shutdown Delay : 180 Seconds

Battery Monitor : OFF
Soft-OFF Delay : 5 Seconds

Voltage Delta : 1.50 V
Hard-OFF Delay : 60 Seconds

Remote Switch

☐ Enable
☒ Disable

Battery Monitor

☐ Enable
☒ Disable

Voltage Delta

v

Power-ON Delay

Seconds

Shutdown Delay

Seconds

Soft-OFF Delay

Seconds

Hard-OFF Delay

Seconds

Submit

Set Default

Log

Clear

Select (4) CAN Bus: (For model AIV-APL1V1FL-PT1)

✖
CAN Bus Version : APL1V1FL 020-001

CAN Bus

Baud Rate : 125K
Receive Mode : By Set Filter

Mask

| | | |
|-------|------|------|
| ID | 0 | 1 |
| Value | 0x00 | 0x00 |

Open Send/Get Message Window

Filter

| | | | | | | |
|-------|------|------|------|------|------|------|
| ID | 0 | 1 | 2 | 3 | 4 | 5 |
| Type | STD | STD | STD | STD | STD | STD |
| Value | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

Baud Rate

125K ▼

Set

Receive Mode

By Set Filter ▼

Set

Set Mask

ID

0

Value (Range: 0~1FFFFFFF Hex)

Set

Set Filter

Filter Type

EXT ▼

ID

0


Value (Range: 0~1FFFFFFF Hex)

Set

Log

Clear

Or select (4) OBD II Test: (For model AIV-APL1V1FL-PT10B)


OBD II Test

Loop Test

Engine Coolant Temperature : 100
Barometric Pressure : 0xFE
Engine Total Fuel Used : 0x0
Engine Speed : 3173
Accelerator Pedal Position 1 : 5.30
Engine Intake Manifold #1 Pressure : 0.03
Tachograph Vehicle Speed : 0.84

Single Function Test

Engine Coolant Temperature

▼

Get

Log

Clear

Select (5) I2C TEST:

×

I2C TEST

Get I2C Value (Hex)

| Address | Offset | |
|---------|--------|------|
| 0 | 0 | Read |

read

Set I2C Value (Hex)

| Address | Offset | Data | |
|---------|--------|------|-------|
| 0 | 0 | 0 | Write |

6. Software Installation and Programming Guide

6.1. Introduction

6.1.1. Environment

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

6.1.2. CAN Bus

6.1.2.1. 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 '**APL1V1FL\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.

6.1.2.2. 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;
```

6.1.3. GPIO and Watchdog

6.1.3.1. 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.

6.1.3.2. 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.

6.1.4. Power Subsystem

6.1.4.1. 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.

6.1.5. I²C

6.1.5.1. Overview

The I²C APIs can be used to get and set the configuration of I²C, The I²C Device address is Defined 0xA6 By invoking the I²C APIs, the users can:

1. Read i2c index data
2. Write i2c index data

6.2. API List and Descriptions

6.2.1. General

| | |
|----------------------|---|
| Syntax: | lib_init(void) |
| Description: | library initialization, using this library must be call this function first. Note: initialization may be wait 1 mins, because scan pic port. |
| Parameters: | None. |
| Return Value: | 0:Successful, -1:Fail. |

| | |
|----------------------|---|
| Syntax: | lib_close(void) |
| Description: | library close, when you not used this library must be call this function. |
| Parameters: | None. |
| Return Value: | 0:Successful, -1:Fail. |

6.2.2. 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: | 0xFFFF (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: | 0xFF (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: | unsigned char get_engine_torque_mode(void) |
| 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: | This function can get the Road Speed Limit Status |
| 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) |
| <hr/> | |
| Syntax: | float get_accelerator_pedal_position_1(void) |
| Description: | This function can get the Accelerator Pedal Position 1 |
| Parameters: | None. |
| Return Value: | An float |
| <hr/> | |
| 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) |
| <hr/> | |
| 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) |
| <hr/> | |
| Syntax: | unsigned char get_accelerator_pedal_position_2(void) |
| Description: | This function can get the Accelerator Pedal 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_driver_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: | unsigned char get_driver_2_time_related_states(void) |
| 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_driver_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 |

6.2.3. CAN Bus

| | |
|----------------------|---|
| Syntax: | i32 getCanFwVer(PicInfo *ver) |
| Description: | This function gets the version information of the CAN Bus firmware. |
| Parameters: | <p>The definition of struct 'PicInfo' is:</p> <pre>struct PicInfo { u8 info[18]; }</pre> <p>This API returns the version information and store the information in the memory which is pointed at by the pointer 'ver'.</p> |
| 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: | <p>If this function sends the CAN packet successfully, it returns 0, any other returned value stands for error.</p> <p>Here is an example:</p> <p>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.</p> <pre> 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"); </pre> |

| | |
|----------------------|---|
| 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: | <p>If this function receives the CAN packet successfully, it returns 0, any other returned value stands for error.</p> <p>Here is an example:</p> <p>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[]'.</p> <pre> unsigned int result = 0; struct CanMsg canAry[30]; result = getCanMessage(canAry, 30); if(result != 0) fprintf(stderr, "Fail to receive CAN packets!\n"); </pre> |

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: | <p>This function takes a variable of type 'CanMask'. User set the bit patterns they want to the 'mask' field in a 'CanMask' variable.</p> <pre> 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); </pre> |
| 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: | <p>This function takes a pointer to a 'CanFilter' type variable. For example:</p> <pre> struct CanFilter varFilter; i32 result; result = getCanFilter(&varFilter); if(result != 0) printf("Fail to get the filter!\n"); </pre> |
| Return Value: | If this function gets the filter successfully, it returns 0, any other returned value stands for error. |

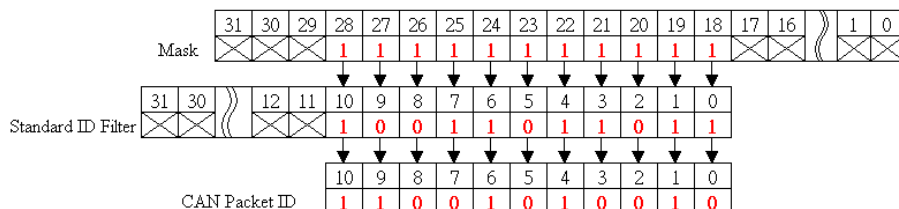
Syntax: i32 setCanFilter(struct CanFilter *varFilter)

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: | <p>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.</p> <p>By setting up 'filterType', users indicate the type of the filter. Filter type could be 'STD_ID' or 'EXT_ID'.</p> <p>Depending on the filter type, the 'filter' field in the CanFilter variable could be 0x0 ~ 0x7FFF (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).</p> <p>For example:</p> <pre> 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"); </pre> |
| Return Value: | <p>If this function sets the filter successfully, it returns 0, any other returned value stands for error.</p> |

6.2.4. GPIO and Watchdog

6.2.4.1. 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 ~ 3 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 pin 0-3 value. 0 is Low, 1 is High |
| Return Value: | None. |

6.2.4.2. Watchdog

| | |
|----------------------|---|
| Syntax: | void wdt_start(int timevalue) |
| 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 'timevalue' is the value to set to watchdog timer register. The range is 0~255. |
| Return Value: | None. |

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

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

6.2.5. Power Subsystem

| | |
|----------------------|---|
| Syntax: | i32 getPicFwVer(struct PicInfo *ver) |
| Description: | This function gets the version information of the firmware of the Power Subsystem. |
| Parameters: | <p>The definition of struct 'PicInfo' is:</p> <pre> struct PicInfo { u8 info[18]; } </pre> <p>This API returns the version information and store the information in the memory which is pointed at by the pointer 'ver'.</p> |
| Syntax: | i32 setPicDefault(void) |
| Description: | <p>The function restores the Power Subsystem to the default values. After calling this API, the items listed below are restored to its default value:</p> <p>Remote Switch → Default: Disabled</p> <p>Battery Monitor → Default: Disabled</p> <p>Battery Voltage Delta Value → Default: 1.5V</p> <p>System Soft Off Delay → Default: 5 seconds</p> <p>System Hard Off Delay → Default: 1 minute</p> <p>System Power On Delay → Default: 2 seconds</p> <p>OS Shutdown Delay → Default: 3 minutes</p> |
| 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: | <p>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'.</p> <p>For example:</p> <pre> 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"); } </pre> |
| 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 getBattVal(float *vol) |
| Description: | This function gets the battery voltage and 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. |

| | |
|----------------------|--|
| Syntax: | i32 getPowerIgnitionStatus(u8* val) |
| Description: | This function get the Power Ignition Status. |
| Parameters: | This function takes a pointer to an unsigned char variable as the parameter. The Status 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 getRemoteSwitchStatus(u8* val) |
| Description: | This function get the Remote Switch Status. |
| Parameters: | This function takes a pointer to an unsigned char variable as the parameter. The Status 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. |

6.2.6. I²c

| | |
|---------------------|---|
| Syntax: | int i2c_read_byte(unsigned char device_address, unsigned char index, unsigned char *data); |
| Description: | This function get the i2c index data. The data value save to pointer data. |
| Parameters: | If this function works successfully, the function returns 0, any other value stands for error. |

| | |
|---------------------|---|
| Syntax: | int i2c_write_byte(unsigned char device_address, unsigned char index, unsigned char data); |
| Description: | This function write the i2c index data. . |
| Parameters: | 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_uares=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_uares=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*

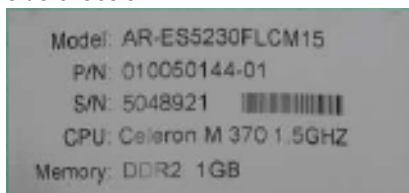
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
/dev/ttyS2, UART: 16550A, Port: 0x03e8, IRQ: 11
/dev/ttyS3, UART: 16550A, Port: 0x02e8, IRQ: 10
/dev/ttyS4, UART: 16550A, Port: 0x04f8, IRQ: 11
/dev/ttyS5, UART: 16550A, Port: 0x04e8, IRQ: 10
```

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

7. 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)

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.

8351 Elm Ave. Suite 107, Rancho Cucamonga,
CA91730, USA

TEL: +1-909-476-0071

FAX: +1-909-466-9951