TEAK 303x Series User's Manual

Feature

- 1U Rack Mount Network Security Platform
- Intel® Atom N270 1.6 GHz Processor , 533 MHz FSB
- Intel® 945GSE North & ICH7-M South Bridge Chipset
- On board DDR2 memory + SODIMM , Up to 1.5 GB w/o ECC
- Build-in 4 ports GbE LAN
- Pre-setting By-pass when power Off
- Fan-less Platform

August, 2009
Version 1.05
ARinfotek Inc., Taipei, Taiwan
http://www.arinfotek.com.tw

Acknowledgments

Intel® Atom are trademark of Intel® Corporation

Award is a trademark of Award Software International, Inc.

CompactFlash™ is a trademark of the Compact Flash Association

Table of Contents

Chapter 1: Introduction

Chapter 2: Software Porting Guide

Chapter 3: The Introduction of BIOS Setup Menu

Chapter 1: Introduction

1-1. Overview

Teak 303x series of network security devices are reliable, high performance and low power consumption devices with DDR2 memory up to 1.5 GB. There are four Intel 82574L Giga LAN controllers and one Intel 82551ER 100 Mbits LAN controller. Two ports of the Ethernets support the bypass function, which is a redirection path to transmit data from one port to another.

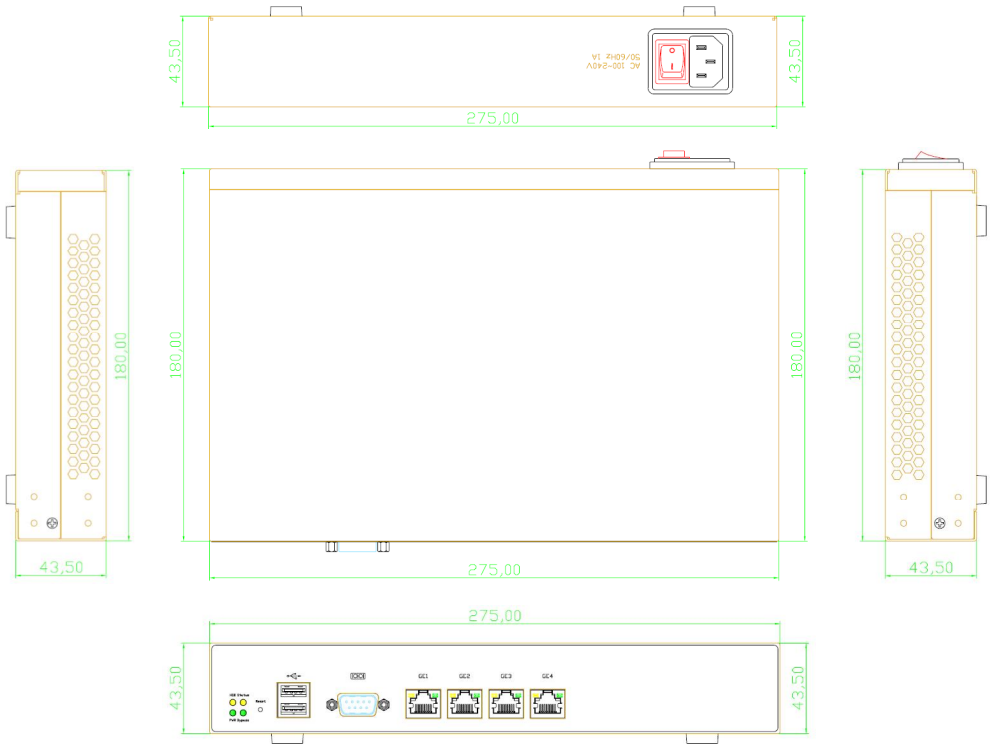
The bypass function on the Teak 303x series is implemented by a CPLD controller. There are two IO ports used to access the settings of the controller. One is an Index port for specifying register to access; the other one is data port for placing data bit to set the function. There are two high speed 16550 compatible UART serial ports, two USB2.0 ports, one mini-PCI Express slot for wireless LAN or something else, and two Serial ATA(SATA) connectors and power socket for hard drive are on board. One CF card slot can be used to install a CF card to boot operation system or to save some data information.

The data configurations for console port are the baudrate 115200, non-parity, 8-bit data byte, and one stop-bit. User can use a terminal software, such as HyperTerm or minicom, to connect to the Teak 303x series through console port to display all messages, including BIOS setup menu.

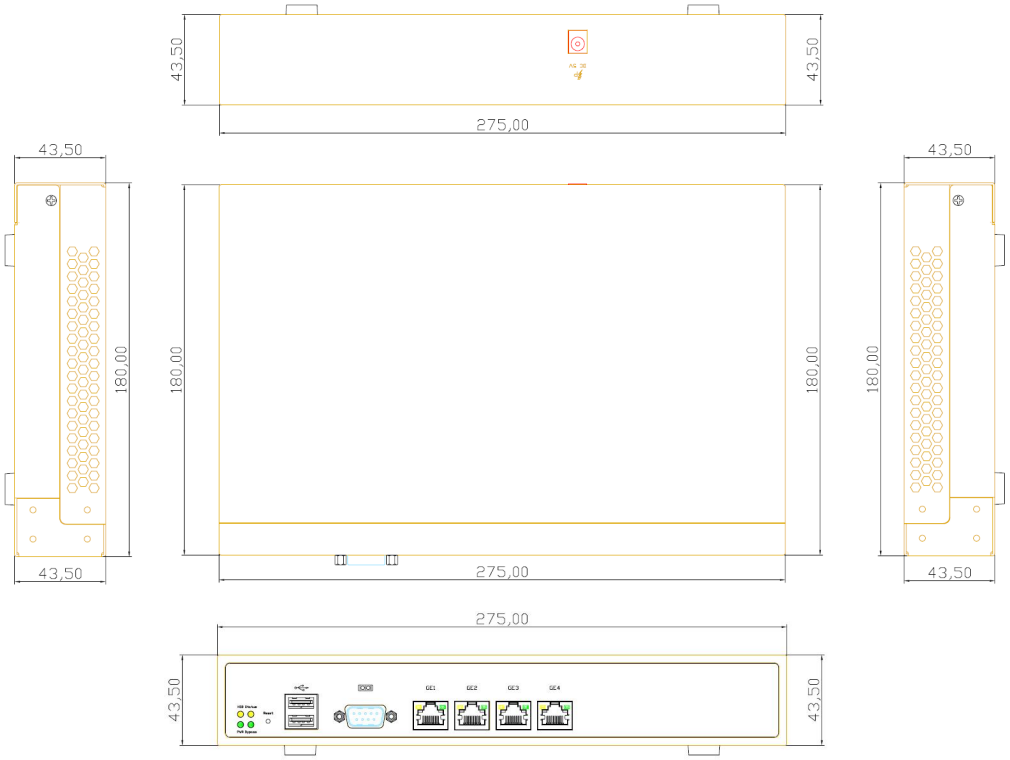
There is one watchdog timer to monitor software application to keep away from deadlock. The board supports Advanced Configuration and Power Interface (ACPI) function implemented in the system BIOS. The built-in hardware sensor can automatically monitor the system temperature, and then starts fan to work when the temperature is up to the specified setting in BIOS. The power of the Teak303x series is up to 150W and input range from ACV 90V to 264V with 4-2A, 47~63Hz.

1-2. Mechanical Diagram

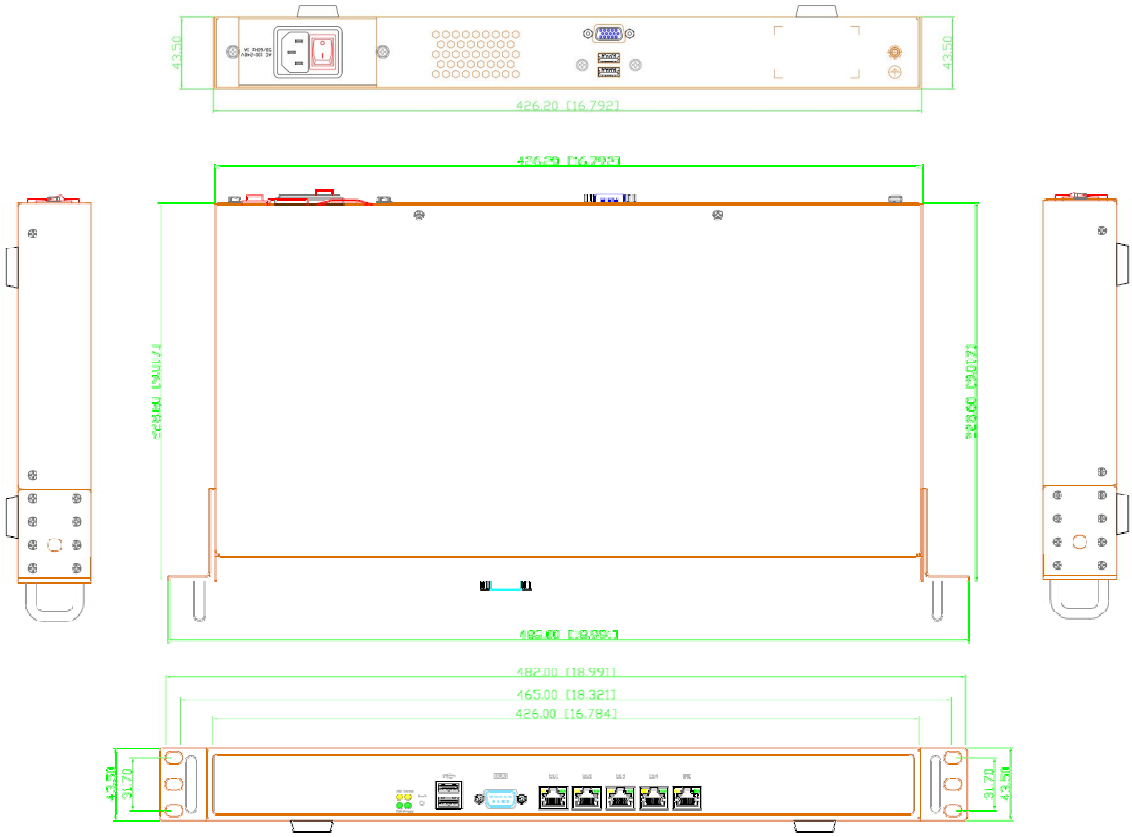
Teak 3030



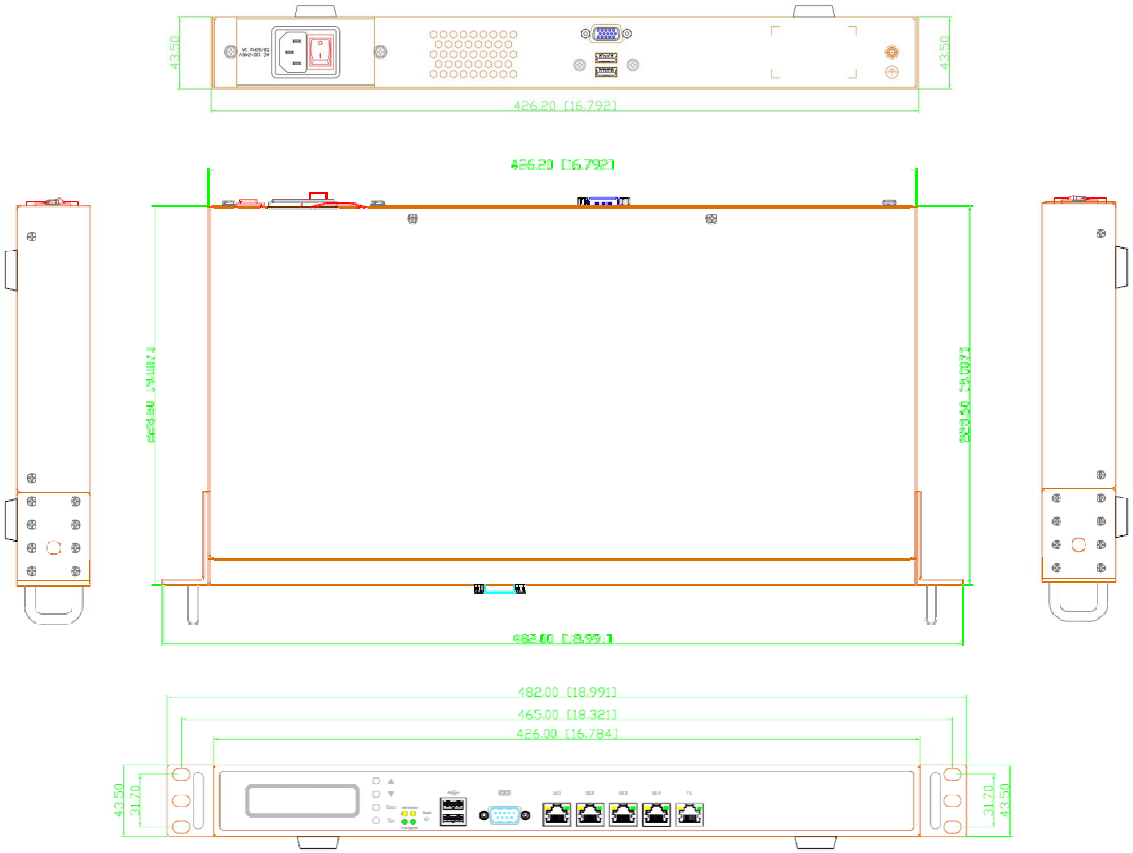
Teak 3031



Teak 3035



Teak 3035L



Teak 303x series Differences:

	Teak 3030	Teak 3031	Teak 3035	Teak 3035L
Disk Storage:	None	Support 2.5”SATA HDD Drive bay	Support 2.5”SATA HDD Drive bay	Support 2.5” SATA HDD Drive bay
Expansion:	Mini-PCI slot (optional)	Mini-PCI slot (optional)	PCI-32 slot (optional)	PCI-32 slot (optional)
Ethernet:	● 4 ports of PCI-E I/F Intel 82574L Giga bit LAN Controller	● 4 ports of PCI-E I/F Intel 82574L Giga bit LAN Controller	● 4 ports of PCI-E I/F Intel 82574L Giga bit LAN Controller ● 1 port of PCI I/F Intel 82551ER FE LAN Controller	● 4 ports of PCI-E I/F Intel 82574L Giga bit LAN Controller ● 1 port of PCI I/F Intel 82551ER FE LAN Controller
Power Supply:	25W industry grade open frame power supply	20W Adapter power supply, DC 5V output	25W industry grade open frame power supply	25W industry grade open frame power supply
Dimension:	275(W) x 180(D) x 43.5(H) mm	275(W) x 180(D) x 43.5(H) mm	482(W) x 228.6(D) x 43.5(H) mm	482(W) x 228.6(D) x 43.5(H) mm
LCM:	None	None	None	Available

1-4. Product Specification

Processor

- Supports Intel® Atom N270 1.6Ghz low power Processor
- 437 pin package, 533 Mhz FSB, 512KB L2 Cache
- Intel 945GSE GMCH integrated Intel® Gen 3.5 Graphics Engine
- PCI V2.3 compliant 3.3V signaling 32 bits interface
- PCI Express root ports. Supports PCI Express 1.0a
- Two SATA Gen1 or Gen2 interfaces
- On Chip 1 channel enhanced IDE and CF, supports Ultra DMA/66/100
- ACPI Function Support
- Watchdog function
- SATA DOM.
- LCM interface
- LPC 1.1 interface

Super I/O

- Intel ICH7-M On Chip USB 1.1 or 2.0 controller that could support up to four USB ports
- On Chip enhanced IDE ATA/100/66/33 channel x 1, support Compact Flash Socket
- ITE8712 Enhanced Super I/O on board.
- Hardware Monitor for system Voltage, Fan and Temperature.
- Two integrated, 16550-compatible UARTs Console port (D-SUB9)
- Watchdog Timer

System Memory

- Supports 64bits width up to 1.5GB (Max.)
- operating frequency: 533MT/Srface
- one SO-DIMM.

Network Controller

- Four PCI-Express Ethernet by Intel 82574L 100/1000 Mbps LAN interface (RJ45 with activity, link, speed LED)
- One Intel 82551ER 10 /100 Base TX support, full duplex
- LAN 3 and LAN 4 have Bypass Function (Power on Bypass enable or not; Power Off Bypass enable or not ; Timer control) . LAN 5 is 100 Mb LAN controller.
- Boot from LAN
- Wake on LAN
- Auto MDIX.

PCIe Slot Extension

- One 1x Mini PCI slot for Wireless LAN module or other expansion

ACPI

- ACPI V.1.0 compliment
- Register sets compatible with “Plug and Play ISA Specification V.1.0a.”

Fan Sensor

- Monitors 2 fan tachometer inputs

Hard Drive Interface

- Two channel SATA interface connectors
  Each channel supports a SATA hard drive and power connector

Low Pin Count Interface

- Comply with Intel Low Pin Count Interface Specification Rev. 1.0
- Supports LDRQ#, SERIRQ protocols
- Supports PCI PME# Interfaces

Enhanced Hardware Monitor

- Built-in-8-bit Analog to Digital Converter
- Three thermal inputs from remote thermal resistor or thermal diode or diode-connected transistor
- Eight voltage monitor inputs (VBAT is measured internally.)
- One chassis open detection input with low power Flip-Flop backed by the battery.
- Watch Dog comparison of all monitored values
- Provides VID0-VID5 support for the CPU
- External Thermal Sensor Host support

Watchdog Timer

- Time resolution 1 minute or 1 second, maximum 65535 minutes or 65535 seconds.
- Output to KRST# when expired

Real Time Clock

- 256 bytes CMOS RAM built-in ICH7-M chipset
- Backup via non-rechargeable Lithium cell batter

Compact Flash Connector

- One CF card slot which is an IDE-CF interface

UART Ports

- Two standard 16C550 UART serial ports.
- Supports IrDA 1.0/ ASKIR protocols and Smart Card Reader protocols

USB Ports

- Two USB v2.0 full, high speed ports at front panel
- Two USB v2.0 full, high speed ports at rear panel(optional).

Keyboard/Mouse Controller

- 8042 compatible for PS/2 keyboard and mouse
- Hardware KBC
- GateA20 and keyboard reset output

LCM Module(Optional)

- 20x2 characters with serial port interface; built-in firmware for user's Programming.Detail descriptions are explained in the hardware section.

System BIOS

- Award System BIOS
- Supports Console Redirection
- Supports Power on after power fail as the BIOS option
- By-pass Enable/Disable selection

Operating System Supports

- Supports DOS 6.22 or above
- Windows 2000/XP
- Linux Family

Power Requirements

- Teak 3030, 3035, 3035L: 25W Open frame
  AC Input: 90~264 full range at 47-63Hz
  DC Output: 5V, 5A
- Teak3031: 20W Power adapter
  AC Input: 90~264 full range at 47-63Hz
  Option: AT Power Supply 150W (max)

Power consumptions

1.6GHz, 2GB DDR2: 5V 3.5A (max)

Environments

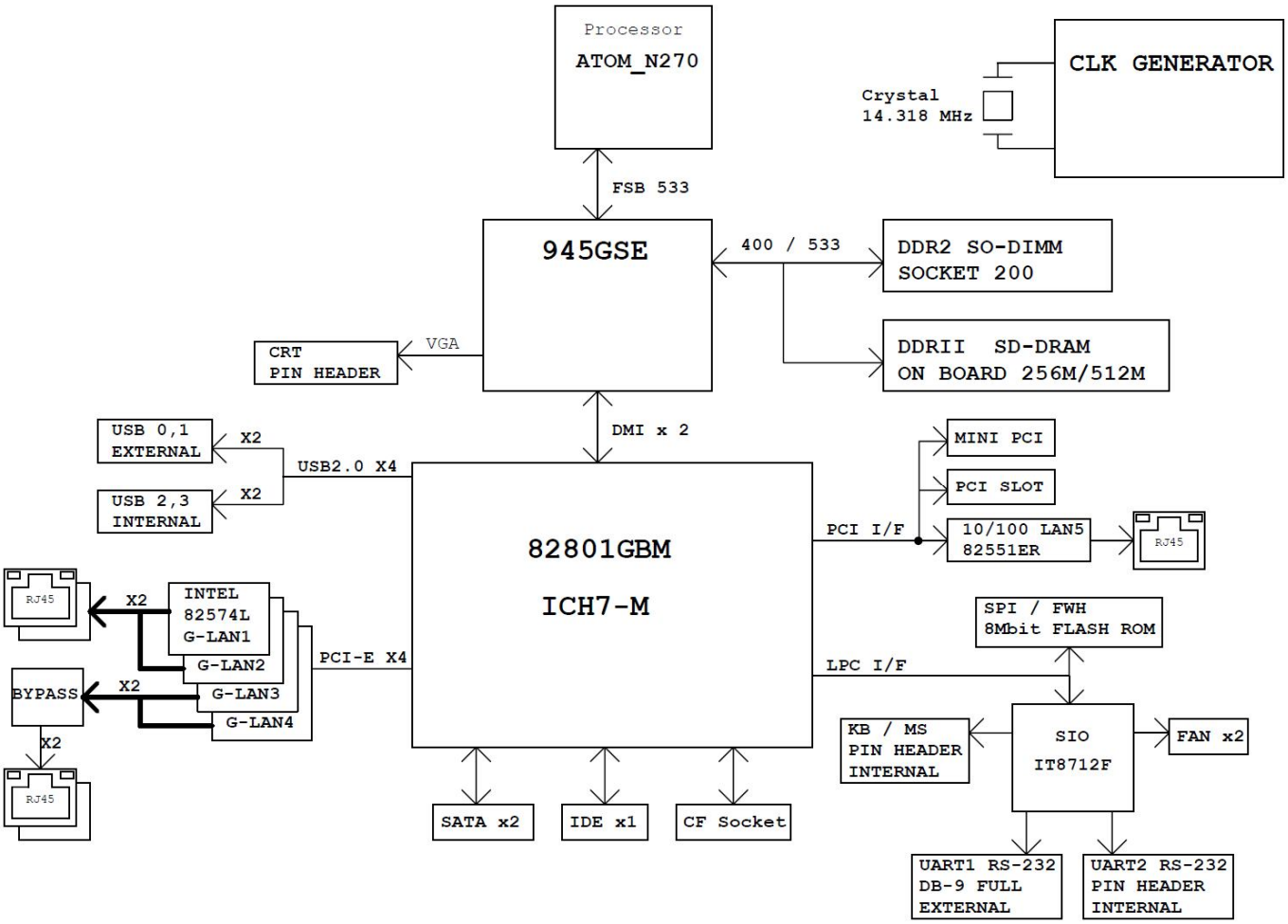
- Operation temperature : 0˚C ~ 40˚C
- Storage temperature : -20˚C ~ 80˚C
- Relative humidity : 10% ~ 90% (Non-condensing)

Dimensions

- Model 3030: 275(W)X180(D)X43.5(H) mm
- Model 3031: 275(W)X180(D)X43.5(H) mm
- Model 3035 3035L: 482(W) X 228.6(D) X 43.5(H) mm

1-5. System Block Diagram

Teak 303x series



1-6. Hardware Description

1-6.1 Ethernet Controller

The 82574 family (82574L and 82574IT) are single, compact, low power components that offer a Physical Layer (PHY) port. The devices use the PCI Express* (PCIe*) architecture (Rev. 2.0).The 82574 family (82574) provide a single-port implementation in a relatively small area so they can be used for server and client configurations as LAN on Motherboard (LOM) designs. The 82574 family can also be used in embedded applications such as switch add-on cards and network appliances.

The 82574 provides one PCIe lane operating at 2.5 GHz with sufficient bandwidth to support 1000 Mb/s transfer rate. 32 KB of on-chip buffering mitigates instantaneous receive bandwidth demands and eliminates transmit under–runs by buffering the entire outgoing packet prior to transmission. Four Intel 82574L Gigabit Ethernet ports support in the Teak 303x series.

The GbE features include:

- Multi-speed operation: 10/100/1000 Mbps
- Half-duplex and full-duplex operation at 10/100 Mbps
- Full-duplex operation at 1000 Mbps
- Power management D0 and D3-hot (D3-Cold on GbE port 0)

PCI Express:

- 64 bit address master support for systems using more than 4GB of physical memory.
- Programmable host memory receive buffers (256 bytes to 16 KB)
- Intelligent interrupt generation features to enhance driver performance
- Descriptor ring management hardware for transmit and receive software controlled reset (resets everything except the configuration space)
- Message Signaled Interrupts (MSI and MSI-X)
- Configurable receive and transmit data FIFO, programmable in 1 KB increments

MAC:

- Flow Control Support compliant with the 802.3X Specification
- VLAN support compliant with the 802.1Q Specification
- MAC Address filters: perfect match unicast Filters; multicast hash filtering, broadcast filter and promiscuous mode
- Statistics for management and RMOM
- MAC loopback

PHY:

- Compliant with the 1 Gb/s IEEE 802.3, 802.3u, 802.3ab Specifications
- IEEE 802.3ab auto negotiation support
- Full duplex operation at 10/100/1000 Mb/s
- Half duplex at 10/100 Mb/s

High Performance:

- TCP segmentation capability compatible with Large Send offloading features
- Support up to 256 KB TCP segmentation (TSO v2)
- Fragmented UDP checksum offload for packet reassemble
- IPv4 and IPv6 checksum offload support (receive, transmit, and large send)
- Split header support
- Receive Side Scaling (RSS) with two hardware receive queues
- 9KB jumbo frame support
- 32 KB packet buffer size

Manageability:

- NC-SI for remote management core
- SMBus advanced pass through interface

Low Power:

- Magic Packet* wake-up enable with unique MAC address
- ACPI register set and power down functionality supporting D0 and D3 states
- Full wake up support (APM and ACPI)
- Smart power down at S0 no link and Sx no link
- LAN disable function

1-6.2 Super I/O -- IT8712F

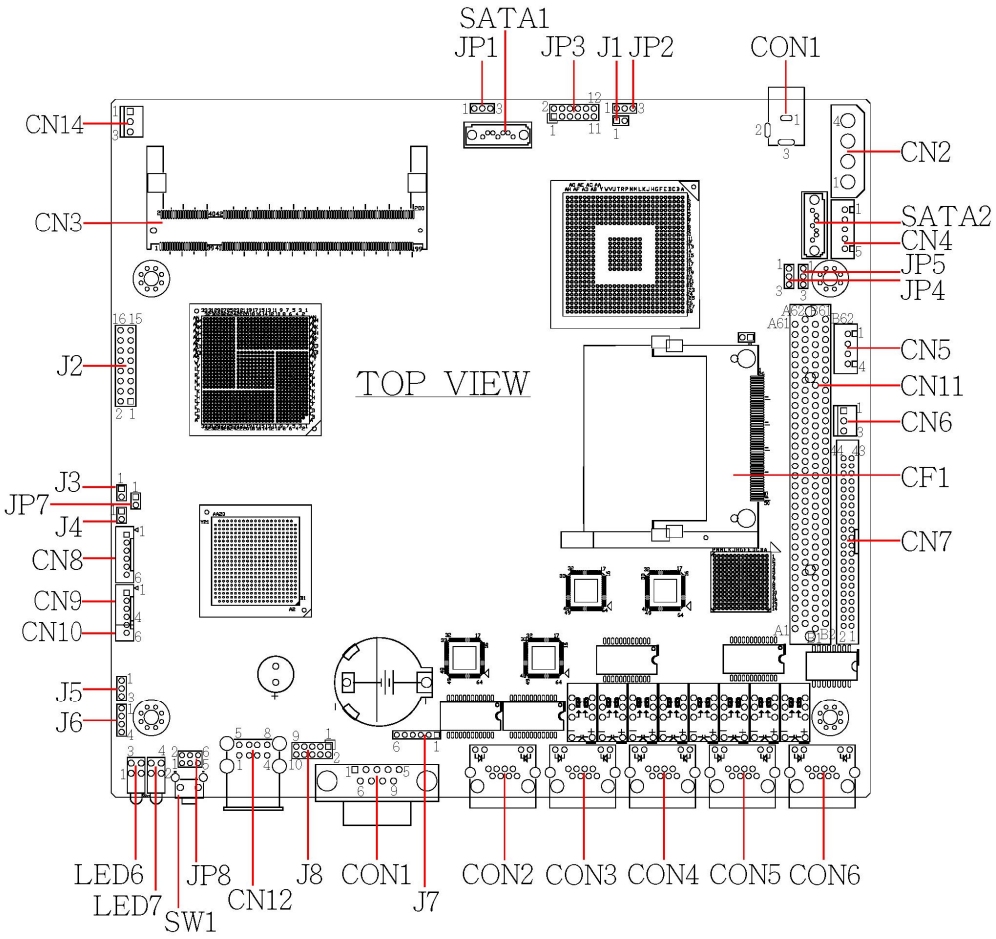
The IT8712F is a Low Pin Count Interface-based highly integrated Super I/O. The IT8712F provides the most commonly used legacy Super I/O functionality plus the latest Environment Control initiatives, such as H/W Monitor, Fan Speed Controller, ITE's "SmartGuardian" function and Smart Card Reader Interface. The device's LPC interface complies with Intel "LPC Interface Specification Rev. 1.0". The IT8712F is ACPI & LANDesk compliant.

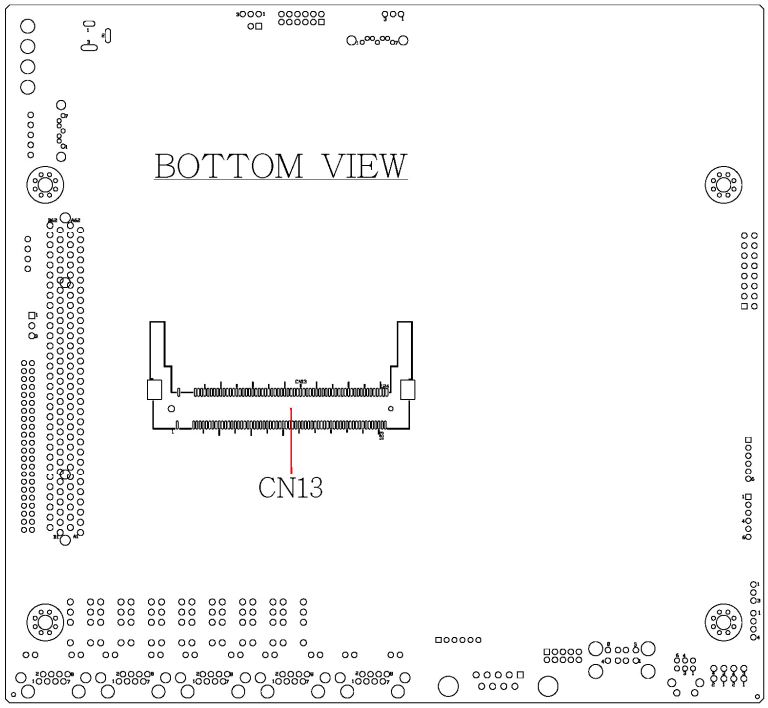
The IT8712F features the enhanced hardware monitor providing 3 thermal inputs from remote thermal resistors, or thermal diode or diode-connected transistor (2N3904). The device also provides the ITE innovative intelligent automatic Fan ON/OFF & speed control functions (SmartGuardian) to protect the system.

1-6.3 LCM module

The LCM module is a 20x2 characters module which communicates with host processor via a serial port. In the Teak 303x series, the firmware is built and running to wait for connection by software handshaking from the host side. The software application demo is distributed in the SDK tool by sample product.

1-7. Teak 303x Board Layout





1-8. Board Pin Definition and Jumper Setting

1-8.1 Pin Definition

CN1: Power Input 1(DC Power Jack_2.5φ) (Option)

Pin	Definition
1	GND
2	GND
3	+5V

CN2: Power Input 3 (WTB_1X4_5.08mm_Male_ATX_DIP)

Pin	Definition
1	+12V
2	GND
3	GND
4	+5V

CN3: DDR2 SO-DIMM_200Pin_0.6mm

CN4: Power Input 2 (WTB_1X5_2.5mm_Male_DIP)

Pin	Definition
1	NC / +12V*
2	+5V
3	+5V
4	GND
5	GND

CN5: 3.5" HDD Power Conn. (WTB_1X4_2.5mm_Male_DIP)

Pin	Definition
1	+12V
2	GND
3	+5V
4	GND

CN6: FAN 2 Connector(WAFER_1X3_2.54mm_Male_DIP)

Pin	Definition
1	GND
2	+12V / +5V*
3	Speed Sensor

CN7: 2.5" IDE Conn. (Box Header_2X22_2.0mm_Male_DIP)

CN8: Internal KB / MS Conn. (WTB_1X6_2.0mm_Male_DIP)

Pin	Definition
1	+5V
2	KB_DT
3	KB_CK
4	MS_DT
5	MS_CK
6	GND

CN9: Internal RS-232 COM2 (WTB_1X4_2.0mm_Male_DIP)

Pin	Definition
1	+5V
2	TX2
3	RX2
4	GND

CN10: Internal RS-232 COM2 (WTB_1X6_2.0mm_Male_DIP) (Option)

Pin	Definiiton
1	+5V
2	TX2
3	RX2
4	GND
5	RTS2
6	CTS2

CN11: PCI Slot (PCI SLOT_120Pin_5V_DIP)

CN12: External USB Conn.(A-Type_2X1_DIP)

Pin	Definition	Pin	Definition
1	+5V	5	+5V
2	DATA0-	6	DATA1-
3	DATA0+	7	DATA1+
4	GND	8	GND

CN13: Mini PCI Socket_(Mini PCI_124Pin_0.8mm_SMD) (Option)

CN14: FAN 1 Connector(WAFER_1X3_2.54mm_Male_DIP)

Pin	Definition
1	GND
2	+12V / +5V*
3	Speed Sensor

CF1：CF Card Socket

CON1：COM1 RS-232 (D-SUB9)

Pin	Definition	Pin	Definition
1	DCD1	2	RX1
3	TX1	4	DTR1
5	GND	6	DSR1
7	RTS1	8	CTS1
9	RI1		

CON2：Gigabyte Ethernet 1(RJ-45 W/LED)

CON3：Gigabyte Ethernet 2(RJ-45 W/LED)

CON4：Gigabyte Ethernet 3(RJ-45 W/LED)

CON5：Gigabyte Ethernet 4(RJ-45 W/LED)

CON6：Fast Ethernet 5(RJ-45 W/LED) (Option)

SATA1：SATAII Conn._ Pin7 Set By JP1

Pin	Definition
1	GND
2	TXP2
3	TXN2
4	GND
5	RXN2
6	RXP2
7	GND* / +5V

SATA2：SATAII Conn._ Pin7 Set By JP5

Pin	Definition
1	GND
2	TXP1
3	TXN1
4	GND
5	RXN1
6	RXP1
7	GND* / +5V

J1：Internal Case Open(Pin Header_1X2_2.0mm_Male_DIP)

pin	Definition
1	+3.3V
2	GND

J2：Internal VGA Conn. (Pin Header_2X8_2.54mm_Male_DIP)

pin	Definition	Pin	Definition
1	RED	2	GREEN
3	BLUE	4	NC
5	GND	6	GND
7	GND	8	GND
9	+5V	10	GND
11	NC	12	SMB_DATA
13	HSYNC	14	VSYNC
15	SMB_CLK	16	NC

J3：Internal Power Button (Pin Header_1X2_2.0mm_Male_DIP)

Pin	Definition
1	+3.3V
2	GND

J4：Internal Reset Button (Pin Header_1X2_2.0mm_Male_DIP)

Pin	Definition
1	+5V
2	GND

J5：Internal Power Status LED (Pin Header_1X3_2.0mm_Male_DIP)

Pin	Definition
1	+5V
2	GND
3	+5VSB

J6：Internal System Status LED (Pin Header_1X4_2.0mm_Male_DIP)

Pin	Definition
1	GND
2	STATUS1
3	STATUS2
4	+5V

- STATUS1：SUPER I/O GPO10，High Active
- STATUS2：SUPER I/O GPO11，Low Active

J7：CPLD JTAG (Pin Header_1X6_2.0mm_Male_DIP)

Pin	Definition
1	+3.3V
2	GND
3	TCK
4	TDO
5	TDI
6	TMS

J8：Internal USB Conn.(Pin Header_2X5_2.0mm_Male_DIP)

Pin	Definition	Pin	Definition
1	+5V	2	+5V
3	DATA0-	4	DATA1-
5	DATA0+	6	DATA1+
7	GND	8	GND
9	+3.3V / NC*	10	NC

LED6：External H.D.D / Power LED Housing

- UP：H.D.D LED，Yellow Color
- DOWN：Power LED，Green Color

LED7：External Status / Bypass LED Housing

UP：Status LED，Yellow Color。SUPER I/O GPO10，High Active

- DOWN：Bypass LED，Green Color.LED ON：Normal，LED OFF：Bypass

SW1：Push Button Function Set By JP8

Pin	Definition
1	Power (ON/OFF) / H/W RESET* / S/W RESET
2	GND

- S/W RESET：SUPER I/O GPI52，Low Active

1-8.2 Jumper Setting

JP1：Set SATA1 Pin7 Power/GND (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2*	Short	GND
2-3	Short	+5V

JP2：CMOS Clear (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2*	Short	Normal
2-3	Short	CMOS Clear

JP3：SPI BIOS Burn (Jumper_2.0mm)

Pin	Jumper Status	Function	Pin	Jumper Status	Function
1-2*	Short	+3.3V	1-2	Open	NC
3-4*	Short	CS	3-4	Open	NC
5-6*	Short	SO	5-6	Open	NC
7-8*	Short	CLK	7-8	Open	NC
9-10*	Short	SI	9-10	Open	NC
11-12*	Short	GND	11-12	Open	NC

JP4：Set CN7 IDE Cable Select (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2*	Short	40pin / Ultra 33MHz
2-3	short	80pin / Ultra 66/100MHz

JP5：Set SATA2 Pin7 Power/GND (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2*	Short	GND
2-3	Short	+5V

JP6：Set CF1 CF Card Master/Slave (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2	Open	Slave
1-2*	Short	Master

JP7：AT/ATX Mode select (Jumper_2.0mm)

Pin	Jumper Status	Function
1-2	Open	ATX Mode
1-2*	Short	AT Mode

JP8：Set SW1 Function (Jumper_2.0mm)

Pin	Jumper Status	Function
1-3*	Short	H/W RESET
3-4	Short	POWER ON/OFF
3-5	Short	S/W RESET

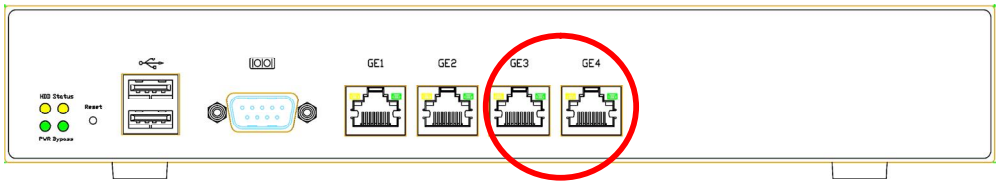
■ S/W RESET：SUPER I/O GPI52，Low Active

Jumper Setting Default:

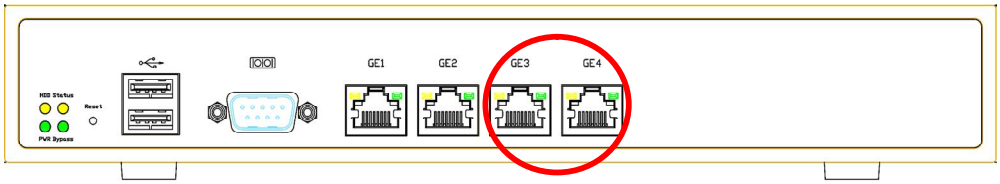
JP1	1-2 Short
JP2	1-2 Short
JP3	1-2 Short, 3-4 Short, 5-6 Short, 7-8 Short, 9-10 Short, 11-12 Short
JP4	1-2 Short
JP5	1-2 Short
JP6	1-2 Short
JP7	1-2 Short
JP8	1-3 Short

1-9. Bypass LAN Ports

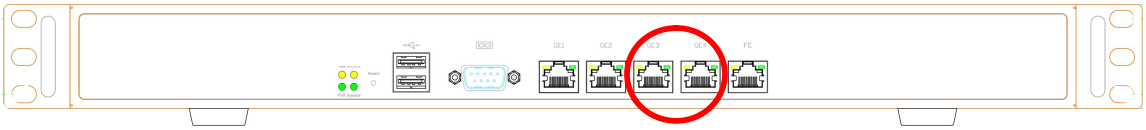
Teak 3030



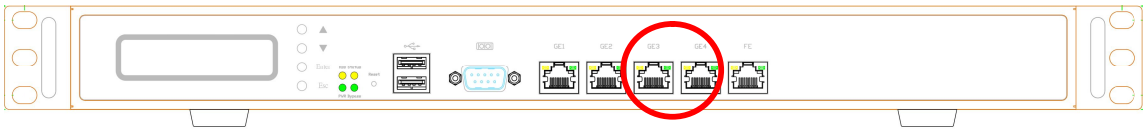
Teak 3031



Teak 3035



Teak 3035L



Chapter 2: Software Porting Guide

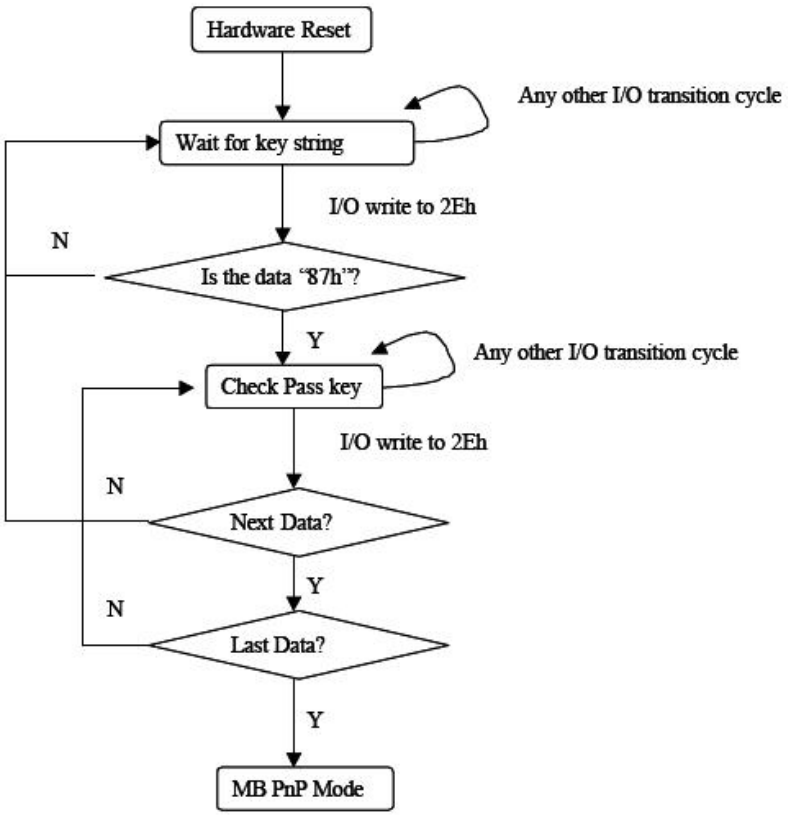
2-1. Data Configuration for Console Port

The console port is a serial UART port with D-SUB 9-pin connector. The BIOS setup menu can be displayed from this port if the Console Redirection function is enabled. The linux kernel or application also can display data via a serial null-modem cable. The data configurations for console port are the baudrate 115200, non-parity, 8-bit data byte, and one stop-bit. User can use a terminal software, such as HyperTerm or minicom, to connect to the device Teak 303x series through console port to display all messages.

2-2. Watchdog Timer Programming

In Teak 303x series, the watchdog function is from the watchdog timer of the Super IO chipset, ITE8712. In the beginning, please check the watchdog device in /dev directory, supposed working in the Linux system, whether the device exists or not. If not, please create a new watchdog device in that directory with major number 10, minor number 130. Then load the device driver distributed by ARinfotek Inc.. The driver is implemented by following the specification of the ITE8712 to reach the watchdog function. The basic flowchart is depicted in the following. Please reference the application code listed in the following to implement your real watchdog timer in your application.

Application can monitor or control the registers of the watchdog function via system call, ioctl(), to set them to your needs, such as timeout counter, enabling or disabling the watchdog function, and periodically reset the watchdog timer.



```
/* This is a watchdog testing code. */
/* 1. Create a watchdog device in /dev directory with major 10 */
/*          #mknod watchdog c 10 130
           2. Operation
                    # E -- Enable the watchdog function
                    # R -- Reset the wathcdog function
                    # D -- Disable the watchdog function
                    # C -- Close function
                    # T -- TimeOut Counter
                    # G -- Get timeOut Counter
                    # Q -- Quit program
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <linux/types.h>
#include <linux/watchdog.h>
typedef  unsigned int                u32;
typedef  unsigned short              u16;

/*********************************************************************
* Prototype
*********************************************************************/

/* The following three statements are defined in key.c */
extern int stdin_init(int);
extern int stdin_read(char *c, int fd);
void set_timer_vector(void);

char Version[]= "V1.10";

/* Start the interrupt-driven serial I/O */
main(int argc, char *argv[]){
           unsigned char c, b;
           int    datafound;
           int ret, i;
           int quit;
//         int readStatus= 0;
//         u32       regVal[4];
//         u32       msrReg;
           unsigned int set;
           static int status= 0;
           unsigned char buf[80];
           //u32                 val[2];
           unsigned char *p=buf;
           //int TxSize;
           int fd;
           u16 TimeOut_Counter=0;

     printf("watchdog-ap: Start Watchdog-AP \n");

           fd= open("/dev/watchdog", O_RDWR);
           // Start the Net
           if(fd < 0) {
                    printf("watchdog-ap:    Can not open device driver\n");
```

```
        exit(0);
}
printf("watchdog-ap: open success fd=%X\n", fd);
//set_timer_vector();
//if(stdin_init(fileno(stdin)) < 0)
//        printf("watchdog-ap: stdin set non-blocking err\n");

printf("watchdog-ap: watchdog AP\n");

quit= 0;
datafound= 0;
status= 0x01;
c= b= 0;

while(!quit) {
 c= 0;

        ret=stdin_read((char *)&c, fileno(stdin));
        //printf("ret=%d C=%X\n", ret, c);
        if( c == 0x0A)
              continue;

        if(ret > 0) {
                switch(c) {
                case 'Q':
                        quit= 1;
                        if(status & 0x01) {
                                close(fd);
                                status= 0x02;
                        }
                        break;
                case 0x0A:          // return code
                        break;
                case 'O':
                        fd= open("/dev/watchdog", O_RDWR);
                        if(fd <= 0)
                                printf("watchdog-ap: Can not open watchdog device, error=%d\n", fd);
                        else {
                                printf("watchdog-ap: Open success code=%d\n", fd);
                                status= 0x01;
                        }
                        break;
                case 'C':
                        if(status & 0x01) {
                                close(fd);
                                status= 0x02;
                        }
                        break;
                case 'R':     // Reset
                        //set= WATCHDOG_RESET;
                        set= WDIOC_KEEPALIVE;
                        ioctl(fd, set, 0);
                        break;
                case 'E':     // enable watchdog
                        set= WDIOS_ENABLECARD;
                        ioctl(fd, set, 0);
                        break;
                case 'D':     // disable watchdog
                        set= WDIOS_DISABLECARD;
                        ioctl(fd, set, 0);
                        printf("disable watchdog\n");
                        break;
                case 'G':     // Get watchdog TimeOut Counter
                        set= WDIOC_GETTIMEOUT;
                        ioctl(fd, set, &TimeOut_Counter);
                        printf("timeOut hex counter= 0x%X\n", TimeOut_Counter);
                        break;
```

```
                        case 'T':
                                p=buf;
                                set= 0;
                                printf("ap:timeOut hex counter=");
                                scanf("%X", &set);
                                *(u16 *)p= (u16)set;
                                //ioctl(fd, WATCHDOG_COUNTER, (unsigned long)buf);
                                set= WDIOC_SETTIMEOUT;
                                ioctl(fd, set, (unsigned long)buf);
                                break;
                        default:
                                break;

                        }
                }
        }       // while loop
        return 0;
}
```

2-3. ByPASS Programming

The bypass function on the Teak 303x series is implemented by a CPLD controller. We only program an application software to monitor or control the function of the bypass, but first installing the bypass device driver to the Linux kernel. The following specification of the bypass is to define the bit settings of the bypass controller registers including the timer and bypass. There are two IO ports used to access the settings of the controller. One is an Index port for specifying register to access; the other one is data port for placing data bit to set the function. They are listed in the following.

First, write register index to the Index Port, 0x200, to indicate which function register you want such as Bypass Timer or Bypass Controller, then read or write data value to the Data Port, 0x201, to retrieve the data bit and setting.

Port Definition:
Index IO Port = 0x200
Data IO Port = 0x201

(A) Bypass Timer Register: Index (0xA0) Default= 0x00
Bit3..0: Based on second timer
0000: Timer not work
0001: 1
0010: 2
0011: 4
0100: 8
0101: 16
0110: 32
0111: 64
1000: 128
1001: 256
1010: 1024
1011...1111:2048
Bit 5,4: Reserved.
Bit 6: Timer control
0: Disable Timer
1: Enable Timer (counting)
Bit 7: Time out bit status (Readonly).
0:Counting or Timer not work
1:TimeOut

(B) Bypass Controller Register : Index(0xA1) Default= 0x00
Bit 1..0: Bit setting for power-off bypass function
00: Set it to Normal (Without Bypass) when major Power was failed.
01: Set it to Bypass action when major Power was failed.
10 and 11: Bypass Mode keeps pre-state when main Power is failed.
Bit 2: Bypass setting when Power Ready
0: Disable Bypass function.
LAN Port is always in normal operation when system power on.
When this bit is set, the other bits are ignored
1: Enable Bypass function.
Bit 4..3: Bypass working type
00: keep pre-state
01: immediately back to Normal
10: immediately Bypass happen
11: Timer control
Bit 6.. 5: Reserved
Bit 7: Bypass Status bit
0: Indicates in Normal Mode
1: Indicates in Bypass Mode

The following code of the bypass is a demo application which statements how to monitor and retrieve the data bit setting via the interface of the linux system. Before using this demo code, you should install the device driver of the bypass, which is distributed by ARinfotek Inc.

```
/* This is a bypass demo AP */
/* This code is used to test the ByPASS function implemented in the board 5010 for ARinfotek */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
typedef  unsigned int                u32;
typedef  unsigned short              u16;
typedef unsigned char                u8;

u16 bypass_timer= 0;
u16 bypass_ctrl= 0;
int fd;


/*********************************************************************
* Prototype
*********************************************************************/

/* The following three statements are defined in key.c */
extern int stdin_init(int);
extern int stdin_read(char *c, int fd);
void set_timer_vector(void);

inline void wr_bypass(u16 set)
{
                write(fd, &set, sizeof(unsigned short));
}

int rd_bypass(u16 *buf)
{
                return read(fd, buf, sizeof(unsigned short));
}
inline void enable_bypass_immediately(void)
{
                bypass_ctrl &= 0x00E7;              // clear
                bypass_ctrl |= 0x14;
                wr_bypass(bypass_ctrl);
}

inline void bypass_back2Normal(void)
{
                bypass_ctrl &= 0x00E7;
                bypass_ctrl |= 0x0C;
                wr_bypass(bypass_ctrl);
}

inline void bypass_timerCtrl(void)
{
                bypass_ctrl &= 0x00E7;
                bypass_ctrl |= 0x1C;                // timer control in controller reg
                bypass_ctrl |= bypass_timer;
                wr_bypass(bypass_ctrl);
}
```

```
inline void bypass_timerSetting(u8 tcode)
{
		bypass_timer &= 0x00F0;
		bypass_timer |= 0x40 | tcode;			// timer setting
		bypass_timer <<= 8;
		//bypass_timer |= bypass_ctrl;			// combine two registers
		printf("bypass timer+ctrl=0x%X\n", bypass_timer);
		//wr_bypass(bypass_timer);
}
inline void bypass_disable_timer(void)
{
		bypass_timer &= 0xBF;
		bypass_timer <<= 8;
		wr_bypass(bypass_timer);
}
inline void check_timer_status(void)
{
		u16 timeout;
		int ret;

		timeout= 0x00;
		ret=rd_bypass(&timeout);
		if(ret) {
			if(timeout & 0x01)
				printf("Timer is TimeOut\n");
			else
			printf("Timer is counting\n");
		}
		else
			printf("read error\n");
}

inline void normal_poweroff(void)
{
		bypass_ctrl &= 0xFC;
		wr_bypass(bypass_ctrl);

}

inline void bypass_poweroff(void)
{
		bypass_ctrl &= 0xFC;
		bypass_ctrl |= 0x01;		// bypass when power off
		wr_bypass(bypass_ctrl);
}

inline void bypass_keep_preState(void)
{
		bypass_ctrl &= 0xFC;
		bypass_ctrl |= 0x02;		// bypass keep pre_state when power off
		wr_bypass(bypass_ctrl);
}
void check_bypass_status(void)
{
		u16 bypass_mode;
		int ret;

		bypass_mode= 0x10;
		ret=rd_bypass(&bypass_mode);
		if(ret) {
		if(bypass_mode & 0x01)
				printf("Current state is BYPASS\n");
			else
				printf("Current state is NORMAL\n");
		}
		else
```

```
                        printf("check bypass status -- read error\n");
}

inline void disable_bypass(void)
{
                bypass_ctrl &= 0xFB;
                wr_bypass(bypass_ctrl);
}


char Version[]= "V1.10";

/* Start the interrupt-driven serial I/O */
main(int argc, char *argv[]){
                unsigned char c, b;
                int    ret;
                int             set;
                int quit;
                int size;
                static int status= 0;
                u8 tcode=0;


          printf("bypass-ap: Start LAN bypass-AP \n");

                fd= open("/dev/bypass", O_RDWR);
                // Start the Net
                if(fd < 0) {
                        printf("bypass-ap:    Can not open device driver\n");
                        exit(0);
                }
                printf("bypass-ap: open success fd=%X\n", fd);

//      if(stdin_init(fileno(stdin)) < 0)
//              printf("bypass-ap: stdin set non-blocking err\n");

                printf("bypass-ap: LAN bypass AP\n");

                quit= 0;

//      datafound= 0;
                status= 0x01;
                c= 0;


                while(!quit) {
                                c= 0;
                        ret=stdin_read((char *)&c, fileno(stdin));

                        if( c == 0x0A)
                          continue;

                        if(ret > 0) {
                                switch(c) {
                                        case '?':
                                                printf("3020-bypass operation command\n");
                                                printf("B - bypass back to normal\n");
                                                printf("C - Check bypass status\n");
                                                printf("D - Disable bypass\n");
                                                printf("E - enable bypass immediately\n");
                                                printf("N - Normal when power off\n");
                                                printf("A - Activate bypass when power off\n");
                                                printf("K - Keep bypass pre-state when power off\n");
                                                printf("t - disable bypass timer\n");
                                                printf("c - check timeout status\n");
                                                printf("T - byass contolled by timer\n");
                                                printf("q - quit\n");
```

```
                                        break;
                                case 'q':
                                        quit= 1;
                                        if(status & 0x01) {
                                                close(fd);
                                                status= 0x02;
                                        }
                                        break;
                        case 0x0A:       // return code
                                        break;
                                case 'B':               // bypass back to normal
                                        bypass_back2Normal();
                                        break;
                                case 'C':               // Check bypass status
                                        check_bypass_status();
                                        break;
                                case 'D': // disable bypass
                                        disable_bypass();
                                        break;
                                case 'E': // enable bypass immediately
                                        enable_bypass_immediately();
                                        break;
                                case 'N': // Normal when power off
                                        normal_poweroff();
                                        break;
                                case 'A':
                                        bypass_poweroff();
                                        break;
                                case 'K':
                                        bypass_keep_preState();      // keep pre_state when power off
                                        break;
                                case 't':  // disable bypass timer
                                        bypass_disable_timer();
                                        break;
                                case 'c':  // check timeout status
                                        check_timer_status();
                                        break;
                                case 'T':
                                        printf("Please timer setting value(hex): ");
                                        scanf("%x", &tcode);
                                        bypass_timerSetting(tcode);                     // 4 sec.
                                        bypass_timerCtrl();
                                        break;
                                default:
                                        printf("3020: bypass command err !\n");
                                 break;
                        } // switch
                }    // if

        }          // while loop
        return 0;
}
```

2-4. LED Indicator Programming

The LED indicator is implemented in the Super IO chipset, ITE8712. This LED can be programmed to be some function which is defined by your application. In the beginning, you should install the LEDIO device driver, provided by ARinfotek Inc, then implement your application software to light on or off your LED to indicate your status. Or modify the LEDIO device driver to meet your need. Remember to create the major number 95 of the /dev/ledio device for the LED indicator in your device directory of the Linux system. The following code is the LED demo application for programming reference. Open the ledio device and write the toggle flag to light on/off the led indicator.

```
/* This is a ledio testing code. */
/* 1. Create a gpio device in /dev directory with major 95 */
/*          #mknod /dev/led c 95 0
            2. Operation
                    # w -- Write GPIO function
                    # t -- Toggle LED auto running
                    # ? -- help
                    # q -- Quit program
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <linux/types.h>
//#include <linux/watchdog.h>
typedef  unsigned int                u32;
typedef  unsigned short              u16;


extern int ledtimer;

/*****************************************************************
* Prototype
*****************************************************************/

/* The following three statements are defined in key.c */
extern int stdin_init(int);
extern int stdin_read(char *c, int fd);
void set_timer_vector(void);

char Version[]= "V1.00";

/* Start the interrupt-driven serial I/O */
main(int argc, char *argv[]){
            unsigned char c, b;
            int    datafound;
            int ret, i;
            int quit;
            unsigned char val;
            unsigned int set;
            static int status= 0;
            unsigned char buf[80];
            unsigned char *p=buf;
```

```
int fd;
int autoled= 0;

printf("ledIO: Start LED IO Application \n");

fd= open("/dev/ledio", O_RDWR);
// Start the Net
if(fd < 0) {
        printf("ledIO:    Can not open device driver\n");
        exit(0);
}
printf("ledIO: open success fd=%X\n", fd);
set_timer_vector();
if(stdin_init(fileno(stdin)) < 0)
        printf("ledIO: stdin set non-blocking err\n");

printf("ledIO: device open done\n");

quit= 0;
datafound= 0;
status= 0x01;
c= b= 0;
val= 0;
autoled= 0;

while(!quit) {
        c= 0;
        ret=stdin_read((char *)&c, fileno(stdin));
        //printf("ret=%d C=%X\n", ret, c);
        if( c == 0x0A)
              continue;

        if(ret > 0) {
                switch(c) {
                case '?':
                        printf("LED IO operation command\n");
                        printf(" w -- write LED IO function\n");
                        printf(" t -- toggle for auto running\n");
                        printf(" ? -- help\n");
                        printf(" q -- Quit program\n");
                        break;
                case 'q':
                        quit= 1;
                        if(status & 0x01) {
                                close(fd);
                                status= 0x02;
                        }
                        break;
                case 0x0A:          // return code
                        break;
                case 'w':     // Get gpio input value
                        b ^= 0x03;
                        ret=write(fd, (char *)&b, 1);
                        if(ret > 0)
                                printf("hex val= 0x%02X\n", b);
                        else
                                printf("read err %d\n", ret);
                        break;
                case 't':
                        autoled ^= 1;
                        if(autoled)
                                printf("ledIO-ap: led ON\n");
                        else
                                printf("ledIO-ap: led OFF\n");
                        break;
                default:
                        printf("ledIO-ap: command err !\n");
```

```
                                break;

                        }
                }
                if(autoled && ledtimer) {
                        ledtimer= 0;
                        b ^= 0x03;
                        ret=write(fd, (char *)&b, 1);
                }
        }       // while loop
        return 0;
}
```

2-5. How to Program to Use a LCM Module

In model Teak 3035, it supports a LCM display to print the relative message from the application. The application should be programmed to communicate with the LCM display. The handshaking protocol is defined in the Distribution CD from ARinfotek Inc. The following code is a complete application and is included in the CD. You can use it and update the code to meet what you want. The module supports the following functions and be controlled or retrieved by application running on the main board through the UART port.

Function Features listing

- Clear display
- Cursor back to home location (HOME)
- Cursor can be shifted to right or left.
- Display backlight is on or off.
- Print message to line1 or line2.
- Display text is shifted right or left.
- Cursor scan be on, off or blink.
- Display text on or off
- Read single key or combined key

All the function routines of the above are implemented in the following code. Application can directly invoke these functions to access the display module through the UART serial port. The data configuration of the serial port is defined to be baudrate 38400, non-parity, 8-bit data byte, and one stop-bit.

The data frame buffer of the handshaking is maximum 32 data length. All the command codes are defined in the header of the program. Please reference the following code to implement your application to save your efforts and time.

```
/* lcm-net.c - A lcm protocol program is to communicate with the lcm controller running on microprocessor through serial port */

/* x86 version -- uncomment the following statement */
#define UART_DEVICE                     "/dev/ttyS1"
/* device version -- uncomment the following statement */
#define UART_IO_FLAGS                   (O_RDWR|O_NONBLOCK)

#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <string.h>
#include <stdlib.h>

#define  CONNECT                        0x05
#define  DISCONNECT                     0x06
```

```
#define  ALARM                          0x07
#define  WRITE                  0x08
#define  PRINT1                 0x09
#define  PRINT2                 0x0A
#define  ACK                    0x0B
#define  NACK                   0x0C
#define  CONFIRM                0x0D
#define  RESET                  0x0E

#define  LCM_CLEAR              0x21
#define  LCM_HOME               0x22
#define    LCM_CURSOR_SHIFT_R   0x23
#define    LCM_CURSOR_SHIFT_L   0x24
#define    BACKLIGHT_ON         0x25
#define    BACKLIGHT_OFF        0x26
#define    LCM_LINE2            0x27
#define    LCM_DISPLAY_SHIFT_R  0x28
#define    LCM_DISPLAY_SHIFT_L  0x29
#define    LCM_CURSOR_ON        x2A
#define    LCM_CURSOR_OFF       0x2B
#define    LCM_CURSOR_BLINK     0x2C
#define    LCM_DISPLAY_ON       0x2D
#define    LCM_DISPLAY_OFF      0x2E


#define  FRAME_MASK             0xFF
#define  TIMEOUT                2
#define  ESC                    0x1B

#define  KEY1                   0x31
#define  KEY2                   0x32
#define  KEY3                   0x33
#define  KEY4                   0x34
#define  KEY12                  0x35
#define  KEY13                  0x36
#define  KEY14                  0x37
#define  KEY23                  0x38
#define  KEY24                  0x39
#define  KEY34                  0x3A

#define  RXFRAME_SIZE       22
#define  TXFRAME_SIZE       32

static char             Mode=0;             // 0:IDLE, 1:command
static unsigned char    dataframe[RXFRAME_SIZE];
static unsigned char    txframe[TXFRAME_SIZE];
static int reEntry= 0;

static int fd= -1;                          // device lcm open file descriptor
#define BAUDRATE B38400

void lcm_fun_key(unsigned char c);

void com_setting(int fd) {
struct termios oldtio;

        tcgetattr(fd,&oldtio);
        oldtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
        oldtio.c_iflag = IGNPAR;
        oldtio.c_oflag = 0;
        oldtio.c_lflag = 0;     //ICANON;
        oldtio.c_cc[VMIN]= 0;
        oldtio.c_cc[VTIME]= 0;

        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&oldtio);
}
```

```
int lcm_open(void)
{
		if(fd < 0) {
			fd= open(UART_DEVICE, UART_IO_FLAGS);
			// Start the Net
			if(fd < 0) {
				printf("lcm-ap:    Can not open device driver\n");
				return -1;
			}
			com_setting(fd);
		  //printf("lcm-ap: open success fd=%X\n", fd);
			return 0;
		}
		else
		  return -1;
}

int lcm_close(void)
{
		if(fd > 0) {
			close(fd);
			return 0;
		}
		return -1;

}

/* This routine is used to transmit a char to the serial port. */
char put_char(unsigned char c) {
		ssize_t ret;

		do {
			ret= write(fd, (void *)&c, 1);			// put char into buffer
			if(ret < 0)
			  usleep(30);					// 100 usec delay
		}while(ret < 0);
		return 1;
}
/* This routine is used to transmit a data string to the serial port. */
int    put_charString(unsigned char *s, int size) {
		ssize_t ret;
	    int t_size= size;

		do {
			ret= write(fd, (void *)s, size);			// put char into buffer
			printf("lcm-net: send data ret=%d left=%d\n", ret, size);
			if(ret < 0)
			  usleep(100);					// 100 usec delay
			else {
			    if(ret <= size) {
			  size -= ret;
				s += ret;
			//printf("lcm-net: left=%d\n", size);
			}
		}
	} while (size > 0);
	//printf("lcm-net: ret\n");
		return t_size;
}

/* Read a char from the serial port and return the size and data */
get_char(unsigned char *c) {
	return read(fd, (void *)c, 1);
}

static int ByteCRC16(int value, int crcin)
```

```
{
		int k = (((crcin >> 8) ^ value) & 255) << 8;
		int crc = 0;
		int bits = 8;
		do {
			if (( crc ^ k ) & 0x8000)
				crc = (crc << 1) ^ 0x1021;
			else
			crc <<= 1;
		k <<= 1;
		} while (--bits);
		return ((crcin << 8) ^ crc);
}

static unsigned short CalcCRC16Bytes(unsigned int count, unsigned char *buffer)
{
		int crc = 0;
	  do {
		int value = *buffer++;
		crc = ByteCRC16(value, crc);
	} while (--count);
	return crc;
}


static void set_txframe(void)
{
		txframe[0]= FRAME_MASK;
		txframe[1]= WRITE;
		txframe[2]= 0x01;				// data length
		txframe[3]= 0;					// data length-2
		txframe[4]= 0;					// key
		txframe[5]= 0;					// checksum-1
		txframe[6]= 0;					// checksum-2
		txframe[7]= 0;
}
static void lcm_sw_init(void) {
	Mode= 0;						// System current mode 0: IDLE

		/* initialize the tx frame data */
		printf("lcm: soft init\n");
		set_txframe();
}

/* Transmit a packetized command frame to remote site */
static void send_cmd_frame(unsigned char cmd) {
	unsigned char cmd_buf[3];
		cmd_buf[0]= FRAME_MASK;
		cmd_buf[1]= cmd;
		cmd_buf[2]= FRAME_MASK;
		put_charString(cmd_buf, 3);
}

/* Transmit a packetized data frame to remote site */
static void send_data_frame(unsigned char cmd, unsigned char *send_data, int length) {
		unsigned short crc;
		int i, j, k;
		unsigned char c;

		i= 4;
		j= 0;
		crc= 0;

		txframe[0]= FRAME_MASK;
		txframe[1]= cmd;
		crc= ByteCRC16((int)cmd, crc);
		txframe[2]= (unsigned char)(length >> 8) & 0xFF;
```

```
                crc= ByteCRC16((int)txframe[2], crc);
                txframe[3]= (unsigned char)(length & 0xFF);
                crc= ByteCRC16((int)txframe[3], crc);

                while(j < length) {
                if((*(send_data+j)==0xFF)||(*(send_data+j)==ESC))
                        txframe[i++]= ESC;
                txframe[i++]= *(send_data+j);
                        crc= ByteCRC16((int)(*(send_data+j)), crc);
                        j++;
                }

                c= (unsigned char)((crc >> 8) & 0xFF);
                if((c==0xFF)||(c==ESC))
                txframe[i++]= ESC;
                txframe[i++]= c;

                c= (unsigned char)(crc & 0xFF);
                if((c==0xFF)||(c==ESC))
                txframe[i++]= ESC;
                txframe[i++]= c;
                txframe[i++]= FRAME_MASK;                       // append frame mask to tail
                j= 0;

                while( i > 0) {
                        k=put_charString((unsigned char *)(txframe+j), i);
                        i -= k;
                        j += k;
                }
}

/* parse data frame and response it */
static void do_frame_analysis(unsigned char *buf, int size) {
        //char i;


          if(Mode != 0x02) {
                send_cmd_frame(NACK); // bad state
                //printf("Bad frame and NACK sent\n");
                  return;
        }
                /* check the CRC value */
                if(CalcCRC16Bytes(size, buf) ==0){
                  //send_cmd_frame(ACK);
                  //printf("Rx data OK\n");
                }
                else {
                send_cmd_frame(NACK);                                           // bad state
                //printf("Rx bad data\n");
                }

                switch(*buf) {
                        case WRITE:     //    Get Key
                                printf("write length=%02X Function Code=%02X\n", *(unsigned short *)(buf+1), *(buf+3));
                                lcm_fun_key(*(buf+3));
                                break;
                        default:
                                printf("bad data frame ignored\n");
                                break;
                }
}

/* do command function except write, print1, print2 */
static void do_cmd_action(unsigned char c) {
                 if(Mode != 0x02) {                     // Check whether it is in IDLE mode
                        switch(c) {
                        case CONNECT:
```

```
                Mode= 0x01;
                send_cmd_frame(ACK);            // tx command ACK
                printf("Connecting and sent ACK\n");
                break;
        case ACK:
                send_cmd_frame(CONFIRM);        // command ack
                Mode= 0x02;
                printf("ACK; connected to confirm Mode=%d\n", Mode);
        break;
        case NACK:
                send_cmd_frame(CONFIRM);        // command ack
                printf("NACK\n");
                Mode= 0;
        break;
        case CONFIRM:                   // confirm signal
                printf("Confirm connection\n");
                Mode= 0x02;
                break;
        case RESET:                     // reset signal
                lcm_sw_init();
                //put_char(FRAME_MASK);                 // put idle code out
                //put_char(FRAME_MASK);
                send_cmd_frame(CONNECT);
Mode= 1;
                printf("Reset\n");
                printf("lcm-net: Send CONNECT\n");
                break;
        default:
                send_cmd_frame(NACK);           // tx bad command
                //printf("NACK sent\n");
                Mode= 0;
                break;
        }
}
else
  switch(c) {
        case ACK:                                               // ACK signal
                send_cmd_frame(CONFIRM);        // command ack
                printf("ACK\n");
                //Mode= 0;
                break;
        case NACK:                                              // NACK signal
                //send_cmd_frame(CONFIRM);      // command ack
                printf("NACK\n");
                break;
        case CONFIRM:                                           // confirm signal
                printf("Confirm, Mode=%d\n", Mode);
                break;
        case CONNECT:                                           // connect command
                send_cmd_frame(NACK);           // tx bad command
                printf("Connect\n");
                break;
        case DISCONNECT:                                        // disconnect command
                //Mode= 0;
                send_cmd_frame(ACK);            // tx command ACK
                printf("Disconnect\n");
                break;
        case RESET:                                             // reset signal
                lcm_sw_init();
                //put_char(FRAME_MASK);                 // put idle code out
                //put_char(FRAME_MASK);
                send_cmd_frame(CONNECT);
                //ModeTimes= 0;
                printf("Reset 2\n");
                printf("lcm-net: Send CONNECT\n");
                break;
        default:
```

```
                                        //printf("Bad Frame skipped\n");
                                        break;
                        }
}

void lcm_fun_key(unsigned char c)
{

                        switch (c ) {
                        case KEY1:
                                        printf("key1=%X\n", c);
                                  break;
                        case KEY2:
                                        printf("key2=%X\n", c);
                                break;
                        case KEY3:
                                        printf("key3=%X\n", c);
                                  break;
                        case KEY4:
                                        printf("key4=%X\n", c);
                                break;
                        case KEY12:
                                        printf("key12=%X\n", c);
                                break;
                        case KEY13:
                                        printf("key13=%X\n", c);
                                break;
                        case KEY14:
                                        printf("key14=%X\n", c);
                                  break;
                        case KEY23:
                                        printf("key23=%X\n", c);
                                break;
                        case KEY24:
                                        printf("key24=%X\n", c);
                                  break;
                        case KEY34:
                                        printf("key34=%X\n", c);
                                break;
                        default:
                                break;
                        }
                                fflush(stdin);
                // The following for debug test
                send_data_frame(WRITE, &c, 1);

}
void lcm_clear(void)
{
                        if(Mode == 2)
                                send_cmd_frame(LCM_CLEAR);
}

void lcm_return_home(void)
{
                        if(Mode == 2)
                                send_cmd_frame(LCM_HOME);
}

void lcm_move_line2(void)
{
                        if(Mode == 2)
                                send_cmd_frame(LCM_LINE2);
}
void lcm_print_msg1(unsigned char *s)
{
                        unsigned short length;
```

```
                if(Mode == 2) {
                        length= strlen((const char *)s);
                        if (length == 0) return;
                        send_data_frame(PRINT1, s, length);
                }
}
void lcm_print_msg2(unsigned char *s)
{
                unsigned short length;
                if(Mode == 2) {
                        length= strlen((const char *)s);
                        if (length == 0) return;
                        send_data_frame(PRINT2, s, length);
                }
}
void lcm_backlight(unsigned char status)
{
                if(Mode == 2) {
                if(status)
                        send_cmd_frame(BACKLIGHT_ON);
                else
                        send_cmd_frame(BACKLIGHT_OFF);
        }
}

void check_connect_status()
{
        if(Mode == 0) {
                send_cmd_frame(CONNECT);
                //printf("timer issues cmd CONNECT\n");
                Mode= 1;
        } else
                if(Mode == 1)
                Mode= 0;
}
void lcm_cursor_shift_right(int i)
{
            int j;
                if(Mode == 2) {
                for(j=0; j < i; j++) {
                        send_cmd_frame(LCM_CURSOR_SHIFT_R);
                        usleep(200);
                }
            }
}

void lcm_cursor_shift_left(int i)
{
             int j;
                if(Mode == 2) {
                  for(j=0; j < i; j++) {
                        send_cmd_frame(LCM_CURSOR_SHIFT_L);
                        usleep(200);
                }
              }
}
void lcm_display_shift_right(int i)
{
        int j;
                if(Mode == 2) {
                  for(j=0; j < i; j++) {
                        send_cmd_frame(LCM_DISPLAY_SHIFT_R);
                        usleep(200);
                }
            }
}
void lcm_display_shift_left(int i)
```

```
{
        int j;
                if(Mode == 2) {
                  for(j=0; j < i; j++) {
                  send_cmd_frame(LCM_DISPLAY_SHIFT_L);
                  usleep(200);
                  }
                }
}

void lcm_cursor_on(void)
{
                if(Mode == 2)
                        send_cmd_frame(LCM_CURSOR_ON);
}
void lcm_cursor_off(void)
{
                if(Mode == 2)
                        send_cmd_frame(LCM_CURSOR_OFF);
}
void lcm_cursor_blink(void)
{
        if(Mode == 2)
                send_cmd_frame(LCM_CURSOR_BLINK);
}

void lcm_display_on(void)
{
        if(Mode == 2)
                send_cmd_frame(LCM_DISPLAY_ON);
}
void lcm_display_off(void)
{
        if(Mode == 2)
                send_cmd_frame(LCM_DISPLAY_OFF);
}

lcm_read_buf(void)
{
                unsigned char c, escFlag= 0;
                int    datafound;
                int    length;
                int quit;
//      int size;

            if(reEntry) return 0;
            reEntry= 1;
                quit= 0;
                length= 0;
                while(!quit) {
                        /* The followings are the Serial port testing code */
                        datafound= get_char(&c);

                        if(datafound == 1) {
                          printf("%02X ", c);
                                if(c == ESC ) {                                 // Encounter successive ESC code
                                if(escFlag) {
                                                escFlag= 0;
                                                dataframe[length++]= c;
                                        }
                                        else
                                        escFlag= 1;
                                        continue;
                                }
                                if(escFlag) {
                                        escFlag= 0;
                                        dataframe[length++]= c;
```

```
                                        continue;
                                }

                        if(c == FRAME_MASK) {
                                 if(length > 0) {
                                        printf("\n");
                                        if(length == 1)
                                        do_cmd_action(dataframe[0]);
                                        else
                                        do_frame_analysis(dataframe, length);
                                        length= 0;
                                }
                        }
                        else
                                        dataframe[length++]= c;
                        } // data found
                else
                quit= 1;
        } // while()
      reEntry= 0;
}
/* Start the interrupt-driven serial I/O */
//lcm_main(void){
main(void){
                unsigned char c;
                int    datafound;
                int    length, ret;
                int quit;
                int size;
                int i;
                unsigned char escFlag= 0;

        printf("lcm-ap: Start Panel-AP \n");
                lcm_sw_init();
                if(lcm_open() < 0) {
                        printf("lcm-ap:    Can not open device driver\n");
                        exit(1);
                }
                set_timer_vector();           // temp removed
                if(stdin_init(fileno(stdin)) < 0)
                        printf("lcm-ap: stdin set non-blocking err\n");

        Mode= 0;
        printf("lcm-ap: lcm-net serial port initialized OK\n");
        send_cmd_frame(RESET);

                quit= 0;
                length= 0;
        if(Mode == 0) {
                send_cmd_frame(CONNECT);
                Mode= 1;
        }

                while(!quit) {
                        /* The followings are the Serial port testing code */
                        datafound= get_char(&c);

                        if(datafound == 1) {
                          //printf("%02X ", c);
                                if(c == ESC ) {                                 // Encounter successive ESC code
                                if(escFlag) {
                                                escFlag= 0;
                                                dataframe[length++]= c;
                                }
                                else
                                  escFlag= 1;
                                continue;
```

```
                        }
                        if(escFlag) {
                                escFlag= 0;
                                dataframe[length++]= c;
                                continue;
                        }

                if(c == FRAME_MASK) {
                        if(length > 0) {
                                printf("\n");
                                if(length == 1)
                                do_cmd_action(dataframe[0]);
                                else
                                do_frame_analysis(dataframe, length);
                                length= 0;
                        }
                }
                  else
                                dataframe[length++]= c;

                } // data found
  ret=stdin_read(&c, fileno(stdin));
                if(ret ==0xFFFFFFFF)
                        continue;
                printf("key= %X ret=%X\n", c, ret);
if(ret > 0) {
        switch(c) {
                case 'q':
                send_cmd_frame(DISCONNECT);
                        Mode= 0;
                        quit= 1;
                        break;
        case 0x0A:          // return code
                        break;
        case 'c':
              send_cmd_frame(CONNECT);
                  Mode= 1;
              break;
        case 'd':
                  if(Mode == 0x02) {
                       send_cmd_frame(DISCONNECT);
                                Mode= 0;
                        }
                        else
                                printf("Can not disconnect\n");
          break;
        case 'h':
              lcm_clear();
             break;
        case 'r':
               lcm_return_home();
              break;
        case 'L':
               lcm_move_line2();
               break;
          case '1':
               lcm_print_msg1((unsigned char *)"1-1234567890ABCDEFG");
               break;
        case '2':
               lcm_print_msg2((unsigned char *)"2-1234567890abcdefg");
               break;
        case 'b':
               lcm_backlight(1);      // backlight on
               break;
        case 'f':
               lcm_backlight(0);      // backlight off
               break;
```

```
                case '3':
                        lcm_cursor_shift_left(5);
                        break;
                case '4':
                        lcm_cursor_shift_right(5);
                        break;
                case '5':
                        lcm_display_shift_left(4);
                        break;
                case '6':
                        lcm_display_shift_right(5);
                        break;
                case '7':
                        lcm_cursor_on();
                        break;
                case '8':
                        lcm_cursor_off();
                        break;
                case '9':
                        lcm_cursor_blink();
                        break;
                case '0':
                        lcm_display_off();
                        break;
                case 'p':
                        lcm_display_on();
                        break;
                default:
                         //put_char(c);
                          break;
                }
                  }    // endif
                    }        // while loop
        lcm_close();

                return 0;
}
```

Chapter 3: The Introduction of BIOS Setup Menu

3-1. BIOS Configuration Overview

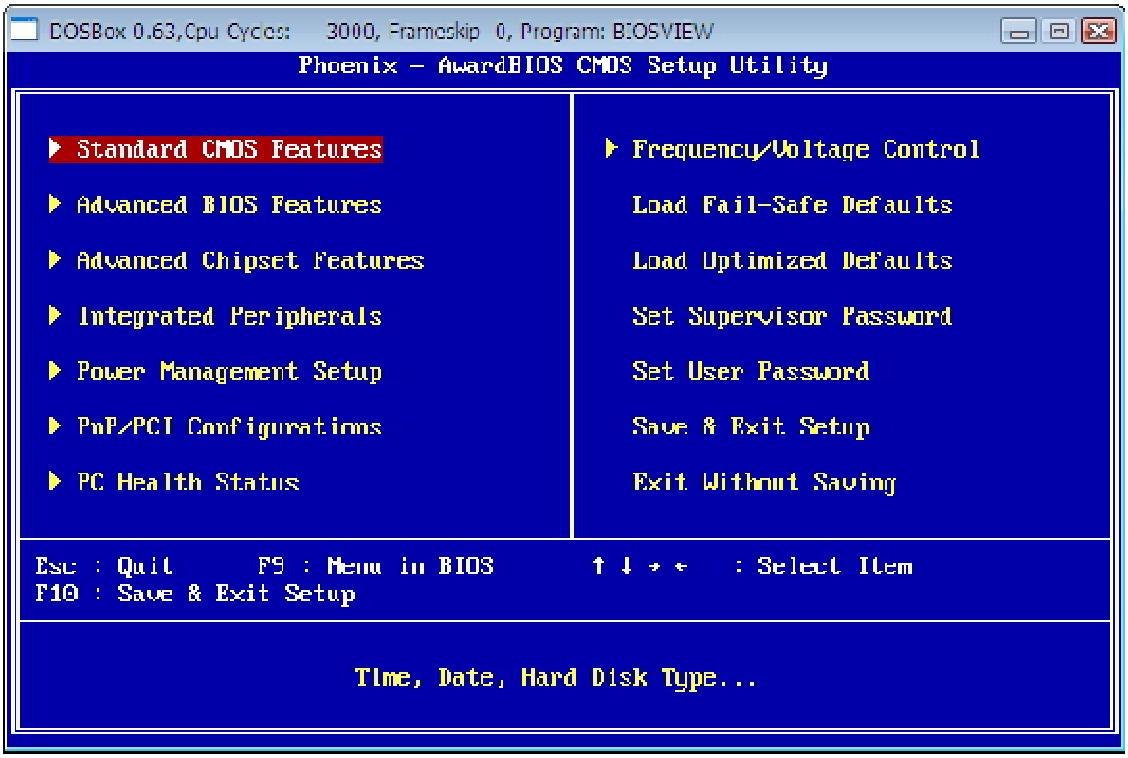
The main board employs the latest Award BIOS. The BIOS (Basic Input and Output System) is a program used to initialize and set up basic I/O peripherals of the computer, which includes the PCI bus and connected devices such as the diskette drive, the keyboard and so on.

In this chapter we will introduce the contents of BIOS used in Teak 303x series . Through understanding BIOS setting will be helpful in application of Teak 303x series.

3-2. Entering BIOS Setup

When the Teak 303x is turned on, the BIOS will perform Power-On Self Test (POST) on the system and display the size of the memory that is being tested. Press the [***Del***] key to enter the BIOS Setup utility, and then the main menu will be showed on the screen.

The BIOS Setup main menu includes some options as the following screen. Use the [***Up/Down***] arrow key to highlight the option that you want to modify, and then press the [***Enter***] key to select the option and configure the functions.

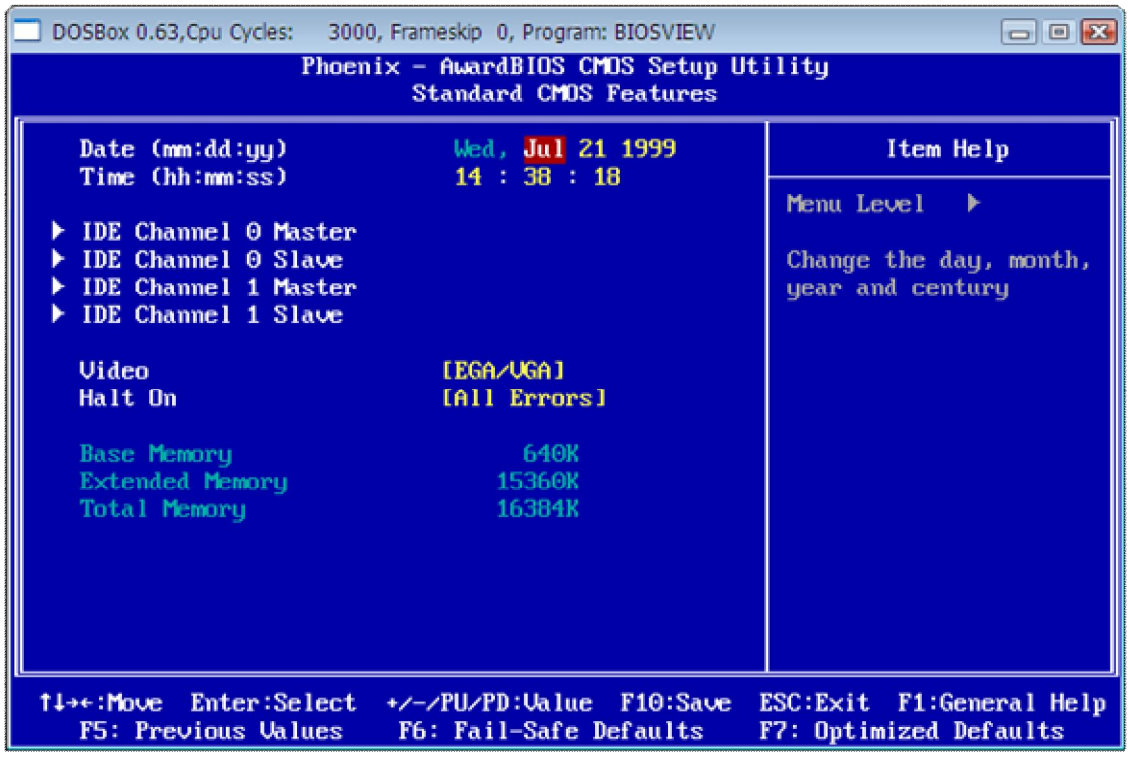


-CMOS Setup Main Menu -

1. In the TEAK 303x series BIOS the factory-default setting is the <***Load Optimized Defaults***>. We recommend using the BIOS default settings, unless you are very familiar with the settings function, or you can contact the technical support engineers (FAE).
2. If the BIOS were lost the settings, the CMOS will detect the <***Load Fail-Safe Defaults***> to boot the operating system. This option will reduce the performance of the system. We recommend choosing the <***Load Optimized Defaults***> in the main menu. This option gives best-case values that should optimize system performance.

3-3. Standard CMOS Features

The <***Standard CMOS Features***> option allows you to set some basic system hardware configurations and set the system clock and error handling. If the CPU board is already installed in a working system, you will not need to select this option anymore.



-Standard CMOS Features -

Date & Time Setup

Highlight the <Date> field and then press the [Page Up] / [Page Down] or [+]/ [-] key to set the current date. Follow the month, day and year format.

Data：Month, Day, Year

Highlight the <Time> field and then press the [Page Up] / [Page Down] or [+]/ [-] key to set the current date. Follow the hour, minute and second format.

Time：Hour, Minute, and Second. Use 24 Hour clock format.
Video [EGA/VGA]

EGA/VGA	Enhanced Graphics Adapter/Video Graphics Array. For EGA, VGA, SVGA and PGA monitor adapters.
CGA 40	CGA 40 Color Graphics Adapter. Power up in 40-column mode
CGA-80	Color Graphics Adapter. Power up in 80-column mode. You should select the setting that matches your video display card. If you have a VGA or any higher resolution card, choose the EGA/VGA setting.
MONO	MONO Monochrome adapter. Includes high-resolution monochrome adapters.

Halt On [All, But Keyboard]

The “Halt On” controls whether the system stops in case of the following error. The options are：

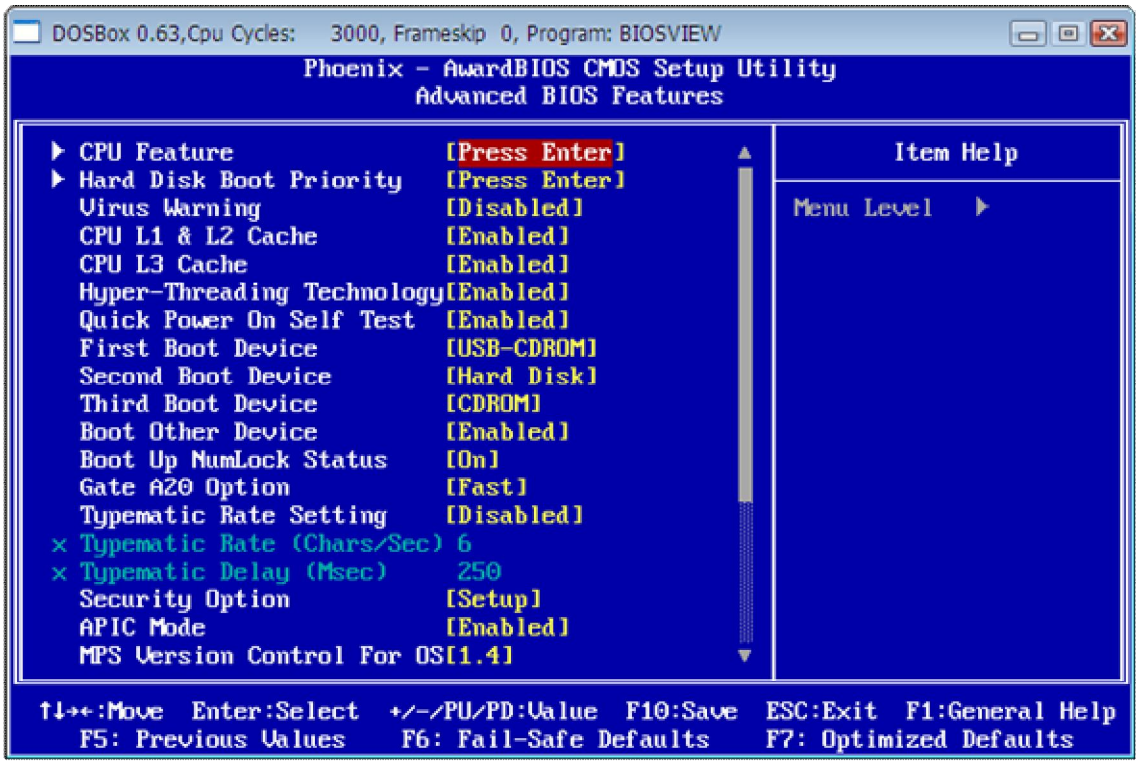
All Errors	The system boot will stop whenever the BIOS detect a non-fatal error.
No Errors	The system boot will not stop for any errors detected.
All, But Keyboard	The system boot will not stop for a keyboard error; it will stop for all other errors.

Base/Extended/Total Memory [Display Only]

These items are automatically detected by the system at start up time. These are display only field. You cannot make changes to these fields.

3-4. Advanced BIOS Features

The <Advanced BIOS Features> option consists of configuration entries that allow you to improve your system performance, or let you set up some system features according to your preference. Note that the page has a scroll-bar to scroll down to more items.



- Advanced BIOS Features -

CPU Feature [Press Enter]

Delay Prior to Thermal [16 Min]

The default setting is 16 minutes.

Hard Disk Boot Priority [Press Enter]

This item is to choose which hard drive boots.

Virus Warning [Disabled]

When enable, any attempt to write to the boot sector or partition table of the hard disk drive will arise the system halted and the warning Message appearing in the mean time. The default value is “Disabled”.

CPU L1&L2 Cache [Enabled]

This setting enables the CPU internal cache (L1&L2 cache). The default setting is “Enabled”. Enable Cache will increase the system performance.

CPU L3 Cache[Enabled]

Hyper-Treading Technology[Enabled]

Quick Power on Self test[Enabled]

1st /2nd /3rd /Boot Other Device

Use these four items to select the priority and order of the devices. The BIOS will boot the operating system according to the sequence of the drive selected. The default setting is “Floppy”, “HDD-0”, “LS120” and “Enabled” respectively.

Boot up NumLock Status [On]

When set to “Disable”, the cursor controls will function on the numeric keypad. When set to “Enable” the numeric keypad function as numerical key. The default value is “Enable”.

Gate A20 Option[fast]

Typematic Rate Setting[Disabled]

Security Option [Setup]

This field controls the password feature. The options are “Setup” and “System”. Selecting “Setup” will protect the BIOS setting from being tampered with. Select “System” if you want to use the password feature every time the system boots up and BIOS setting. The default setting is “Setup”. You can create your password by using the “Supervisor/User Password” utility in the main program screen.

APIC Mode [Enabled]

APIC is advanced programmed interrupt controller. TEAK 303x series support IOAPIC through the system BIOS. Enable “APIC” lets the IO APIC capability to 24 interrupts.

Console Redirection [Enabled])

This function is allowed to display all message to the console port when it starts booting.

MPS Version Control For OS[1.4]

OS Select For DRAM > 64MB [Non-OS2]

Baud Rate [115200]

This selection allows you to select the baud rate if the console redirection is enabled. The default value is 115200 bps.

Agent Connect Via[NULL]

Agent Wait time(min) [1]

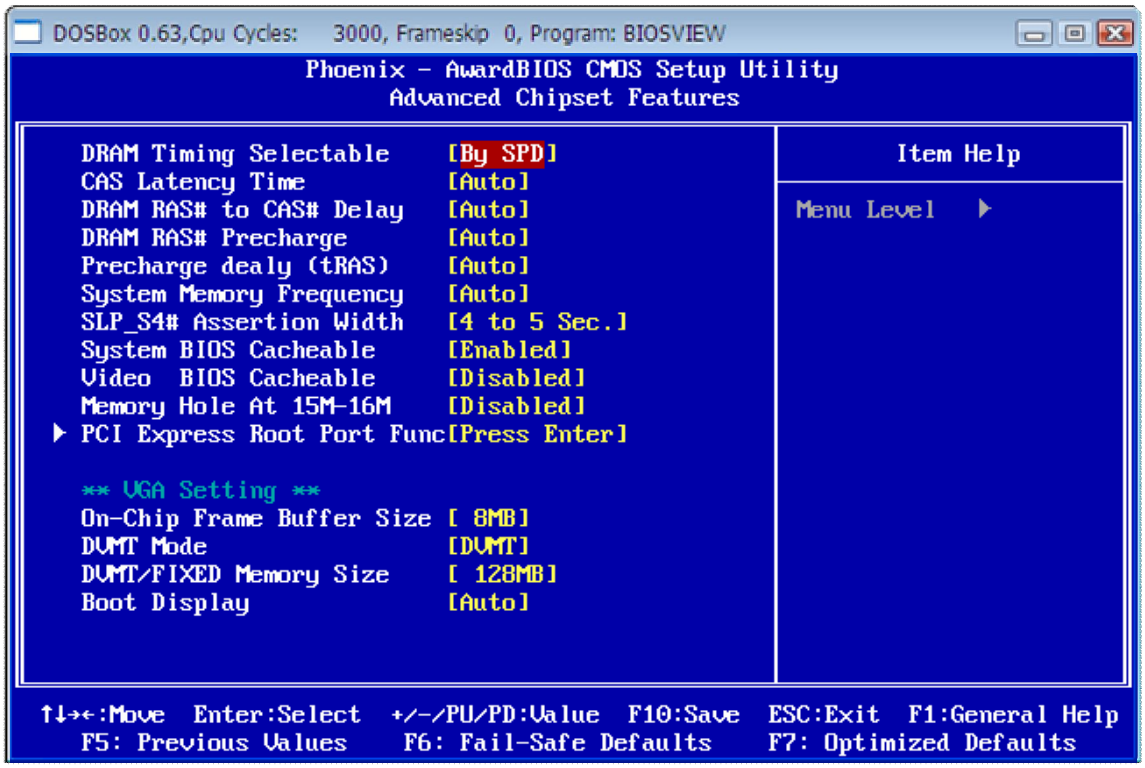
Agent After Boot[Disabled]

Report No FDD For WIN 95

Small Logo(EPA) Show [Disabled]

3-5. Advanced Chipset Features

This section describes the configuration of chipsets on board. Control keys for this screen are the same as for the previous screen.



- Advanced Chipset Features –

DRAM Timing Selectable [By SPD]

SLP_S4# Assertion width[4 to 5 Sec.]

System BIOS Cacheable [Enabled]

When this option is enabled, accesses to the system BIOS ROM addressed at F0000H-FFFFFH are cached to improve the system performance, provided that the cache controller is enabled. The default value is “Enabled”.

Video BIOS Cacheable [Disabled]

Select Enabled allows caching of the video BIOS, resulting in better system performance. However, if any program writes to this memory area, a system error may result. The default setting is “Disabled”.

Memory Hole AT 15M-16M [Disabled]

In order to improve system performance, certain space (15MB-16MB) in memory can be reserved for ISA cards. This memory must be mapped into the memory space below 16MB. When enabled, the CPU assumes the 15-16MB memory range is allocated to the ISA address range instead of the actual system DRAM.

When disabled, the CPU assumes the 15-16MB address range actually contains DRAM memory. The default value is “Disabled.”

PCI Express Root Port [Press Enter]

PCI Express Port 1 [Auto]

PCI Exprees Port 2 [Auto]

PCI Exprees Port 3 [Auto]

PCI Exprees Port 4 [Auto]

PCI Exprees Port 5 [Auto]

PCI Exprees Port 6 [Auto]

PCI- E Compliancy [v1.0a]

VGA Setting

On-Chip Frame Buffer Size[8MB]

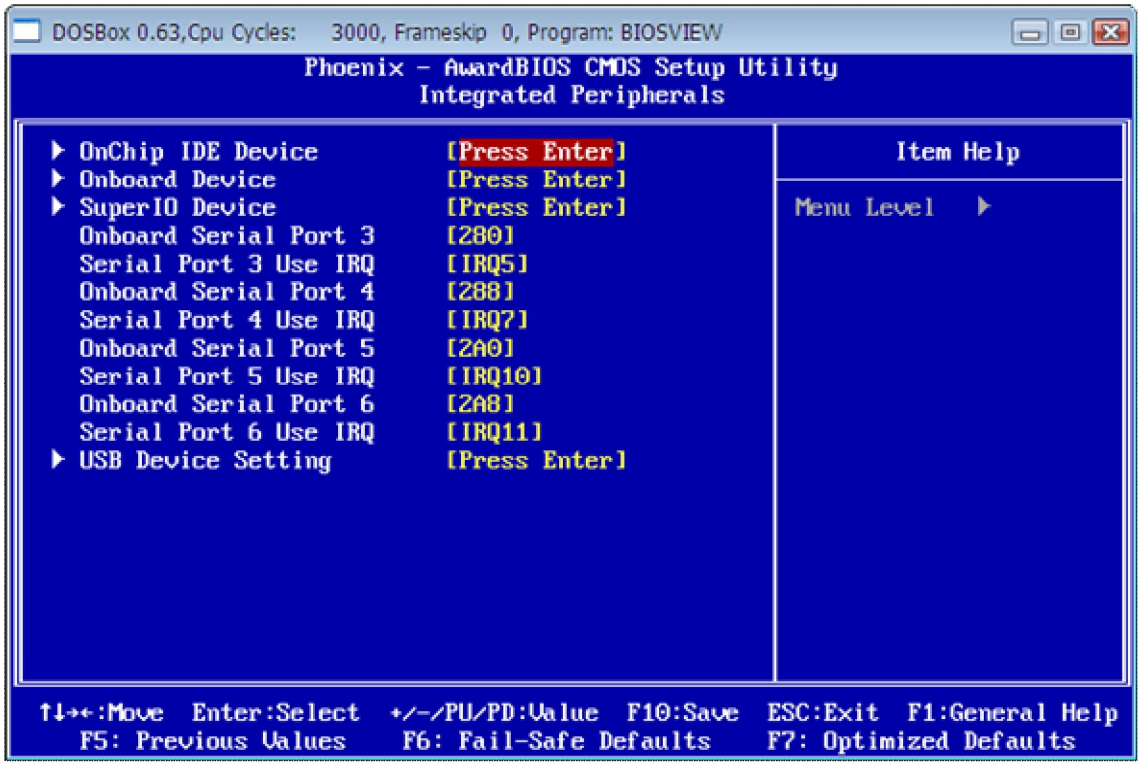
DVMT Mode [DVMT]

DVMT/FIXED Memory Size [8MB]

Boot Display [Auto]

3-6. Integrated Peripherals

This section is used to configure the peripheral chipset features.



– Integrated Peripherals Setup –

OnChip IDE Device [Press Enter]

IDE HDD Block Mode[Enabled]

IDE DMA transfer access[Enabled]

On-Chip Primary PCI IDE [Enabled]

The chipset contains a PCI IDE interface with support for two IDE channels. Select Enabled to activate the Primary IDE interface. Select Disabled to deactivate this interface. The default setting is “Enabled”.

IDE Primary Master PIO [Auto]

IDE Primary Slave PIO [Auto]

In IDE Primary PIO including Master and Slave, there are six IDE PIO Mode (Programmed Input/ Output) fields that lets you set a PIO mode Auto and (0-4) for each of the four IDE devices that the onboard IDE interface supports. Modes 0 through 4 provide successively increased performance. In Auto mode, the system automatically determines the best mode for each device. The Choice: Auto, Mode 0, Mode 1, Mode 2, Mode 3, and Mode 4.

IDE Primary Master UDMA [Auto]

IDE Primary Slave UDMA [Auto]

In IDE Primary, the Ultra DMA-33/66/100 implementation is possible only if your IDE hard drive supports it and the operating environment includes a DMA driver (Windows 95 OSR2 or a third-party IDE bus master driver). If your hard drive and your system software both support Ultra DMA-33/66/100, select Auto to enable BIOS support.

On-Chip Serial ATA [Auto]

Default setting is Auto. That is arranged by BIOS. [Combined Mode]: PATA and SATA are combined. Max. of 2 IDE drives in each channel. [Enhanced Mode]: Enable both SATA and PATA. Max. of 6 IDE drives are supported. [SATA Only]: SATA is operating in legacy mode.

On Board Device [Press Enter]

Super IO Device [Press Enter]

The Super IO device is a Low Pin Count Interface chipset, ITE8712F. The ITE8712F has integrated 12 logical devices, including floppy disk controller, two UART devices, one multi-mode parallel port, keyboard controller and so on. These 12 logical devices can be individually enabled or disabled via software configuration registers, such as BIOS setup menu.

USB Device Setting [Press Enter]

USB 1.0 controller [Enabled]

This item allows you to activate the function of USB on-board. Two choices "Enable" and "Disable" are supported. The default setting is "Enabled".

USB 2.0 Controller [Enabled]

ICH7 Chipset supports USB2.0. It is recommended to set "Enabled" in this item.
USB Operation Mode[High Speed]

USB Keyboard Function [Enabled]

These items allow users to set Keyboard 'Enable" on USB ports. The default setting is "Enabled".

USB Mouse Function [Enabled]

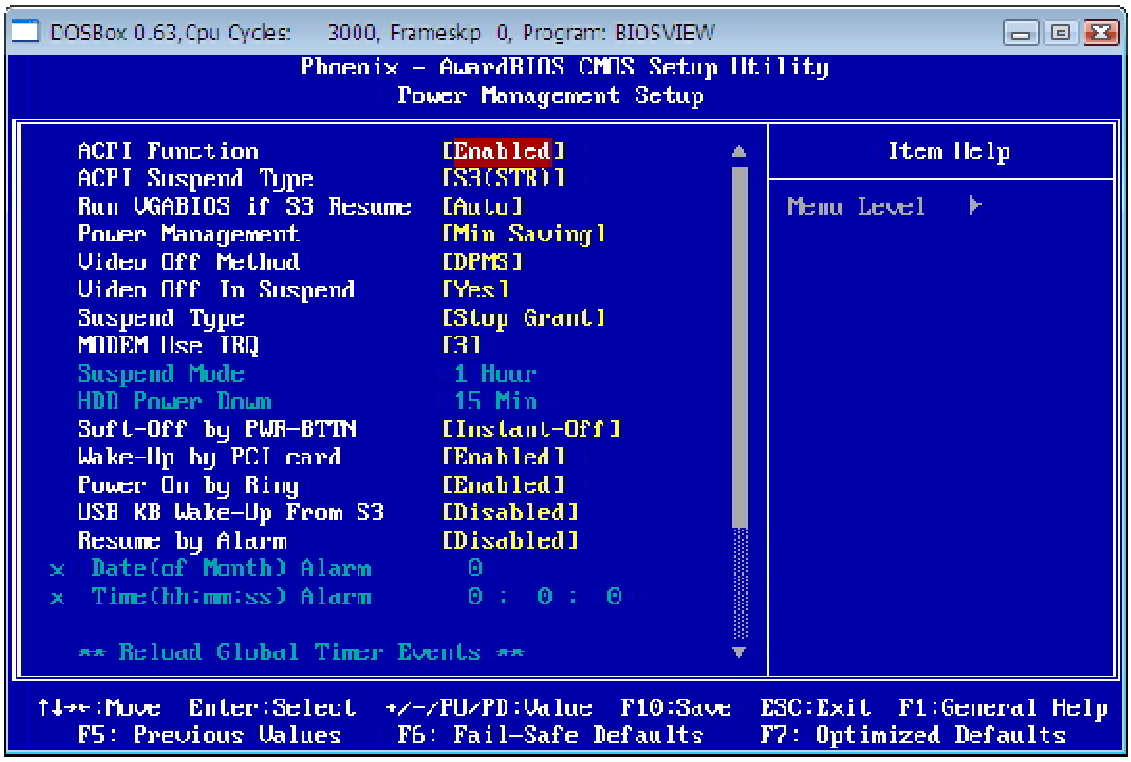
These items allow users to set Mouse on USB ports. The default setting is "Enabled".

USB Storage Function[Enabled]

Define the peripheral device as a storage device. Default setting is "Enabled".

3-7. Power Management Setup

Power Management Setup allows users to configure the system to the most effectively save energy while operation in a manner consistent with your own style of computer use.



- Power Management Setup -

ACPI Function [Enabled]

This item allows you to enable/disable the Advanced Configuration and Power Management (ACPI). The default setting is “Enabled”.

ACPI Suspend Type [S3(STR)]

This field is used to select the type of Suspend mode. The default setting is S3 (STR).

Run VGABIOS if S3 Resume [Auto]

The default setting is “Auto”. Choices: Auto, Yes, and No.

Power Management [User Define]

This option is the master control for the power saving modes, Display Turn off and HDD Power Down that together form the hardware power conservation scheme. There are three settings: User Define, Max Saving, and Min Saving.

User set the HDD/Doze mode/Suspend Mode to one own requirement to achieve power saving.

Video Off Method [DPMS]

This option determines the manner in which the monitor is under power saving mode.

Video Off in Suspend [Yes]

This option determines video whether video off takes effective during suspend mode. The default is “YES” to indicate video off takes off during suspend mode.

Suspend Type [Stop Grant]

This option controls processor power control in two states "PwrOn Suspend" and "Stop Grant". User can select the state depending on your need. It is recommended to select "Stop Grant". The default is "Stop Grant".

MODEM Use IRQ [3]

This item determines the IRQ in which the MODEM can use. The options are: 3 (Default setting), 4, 5, 7, 9, 10, and 11, NA

Suspend Mode [Disabled]

System will enter suspend mode to save power after the duration when system is idle. This option is to set the duration or disable that function system enters into suspend mode.

HDD Power Down [Disabled]

This option is to set the duration when HDD is idle. HDD will power down to save power after the duration when HDD is idle. This option is to set the duration or disable power down function.

Soft-Off by PWR-BTTN [Instant-Off]

This item lets user select the function of PWR-BTTN as "Soft Off" or "Instant Off". "Soft Off "is to press the power button for more than 4 seconds to force the system into off state when the system has "hung." The choice: Delay 4 Sec, Instant-Off.

Wake Up by PCI Card [Enabled]

Wake Up events are I/O events whose occurrence can restart/start the system into normal operation. Users can select the IO device which wakes up the system.

Power On by Ring[Enabled]

USB KB Wake-Up From S3

Resume by Alarm

Reload the Global Timer Events

Primary IDE 0 [Disabled]

Primary IDE 1 [Disabled]

Secondary IDE 0 [Disabled]

Secondary IDE 1 [Disabled]

The item options are used to enable or disable the reload global timer events for the primary or secondary IDE. The default setting is "disabled".

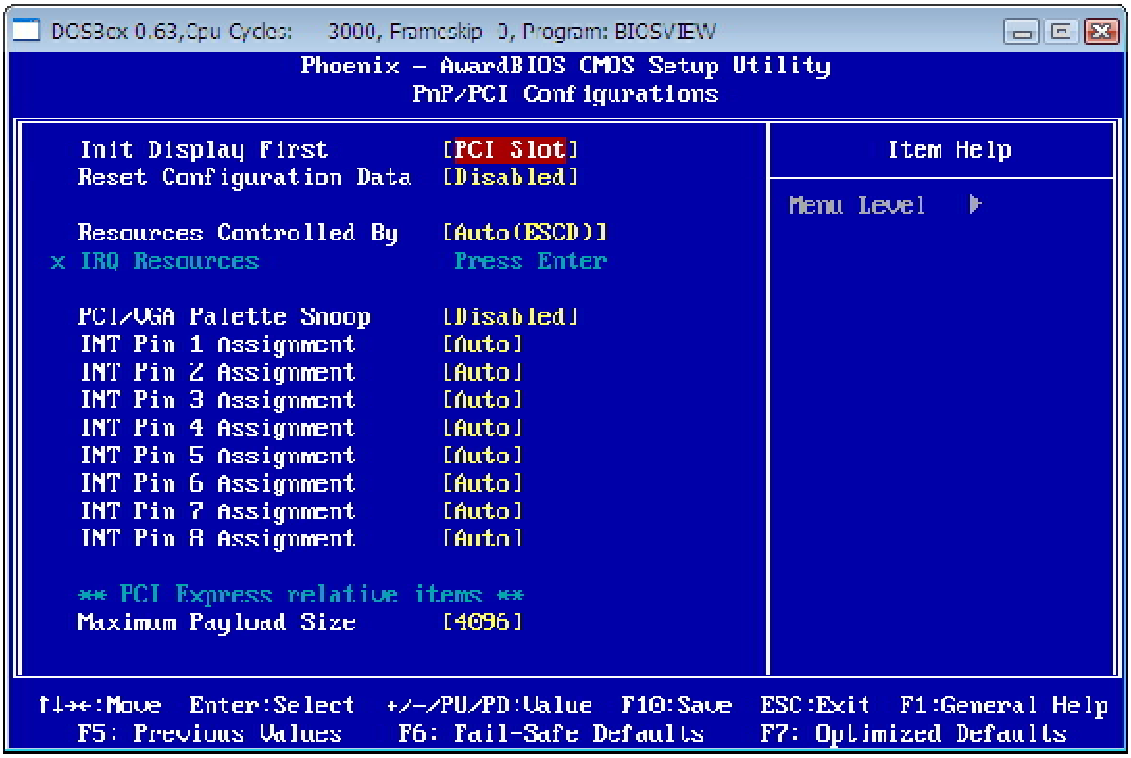
FDD, COM, LPT Port [Disabled]

Set the option enabled or disabled to the FDD, COM, and LPT port. The default setting is "Disabled".

PCI PIRQ[A-D]# [Disabled]

3-8. PnP/PCI Configurations

This section is used to configure PnP & PCI Configurations. The <PnP & PCI Configurations> option configures the PCI bus slots. All PCI bus slots on the system use INTA#, thus all installed PCI cards must be set to this value.



-PnP / PCI Configurations-

Init Display First [PCI Slot]

This item allows you to determine which one Display is used to show the message. The default is “PCI Slot”. The choices: PCIEx and Onboard.

Reset Configuration Data [Disabled]

The default value is “Disabled”. Normally, you leave this field “Disabled”. Select “Enabled” to reset the Extended System Configuration Data(ESCD) when you exit Setup if you have installed a new add-on and the system reconfiguration has caused such a serious configuration conflict that the OS cannot boot.

Resource Controlled by [Auto (ESCD)]

The default value is “Auto“.

The Plug-and-Play BIOS can automatically configure all the boot and Plug-and-Play compatible devices according to ESCD (Extended System Configuration Data) If you select “Auto”. Almost all the add-on cards support PNP.

The resource control is set to “Auto”. If the add-on card is legacy ISA or without PNP function users should select “Manual” in this item.

PCI/VGA Palette Snoop [Disabled]

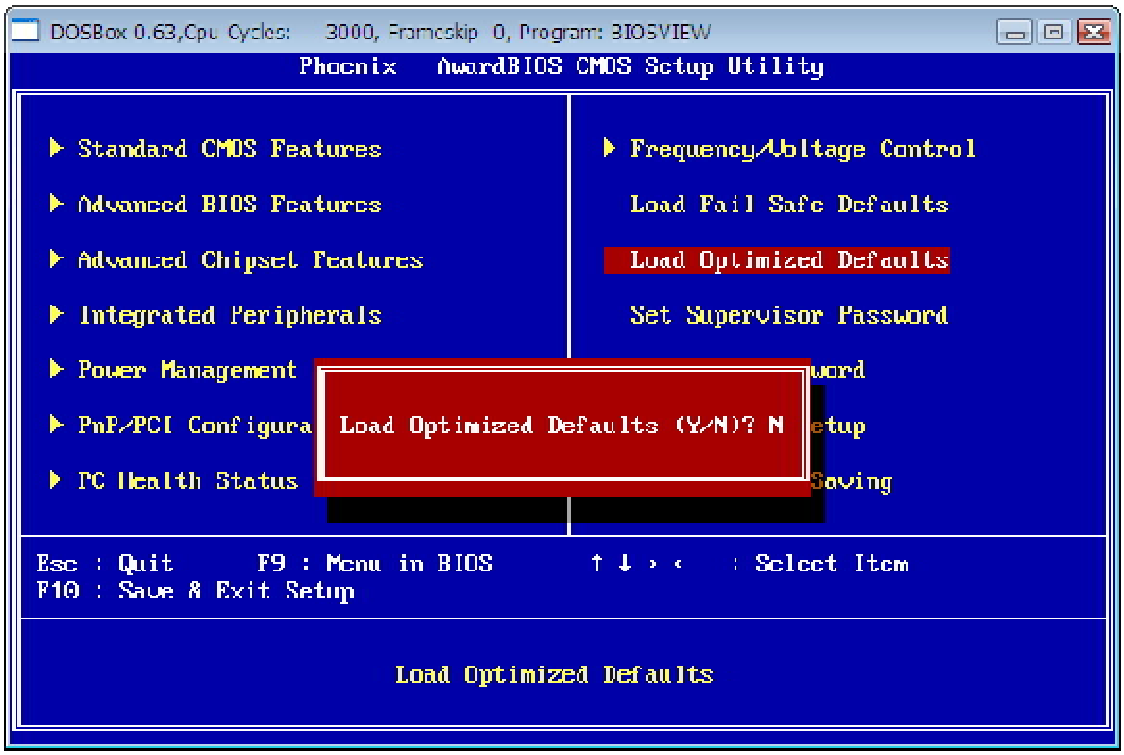
This item set enable or disable for the snoop of display card to Palette. Leave this field at "Disabled" unless the display is abnormal. The default setting is "Disabled".

INT Pin 1 Assignment [Auto]
INT Pin 2 Assignment [Auto]
INT Pin 3 Assignment [Auto]
INT Pin 4 Assignment [Auto]
INT Pin 5 Assignment [Auto]
INT Pin 6 Assignment [Auto]
INT Pin 7 Assignment [Auto]
INT Pin 8 Assignment [Auto]

*** PCI Express relative items ***
Maximum Payload Size [128]

3-9. Load Optimized Default

This section permits users to select a group of settings for all BIOS Setup options. Not only can you use these items to quickly set system configuration parameters, you can choose a group of settings that have a better chance of working when the system is having configuration related problems.



- Load Optimized -

References

1. Mobile Intel Atom Processor N270 Single core Datasheet
2. IT8712F EC-LPC I/O V0.9.1
3. Mobile Intel 945 Express Chipset Family Data sheet
4. Intel I/O Controller Hub7 (ICH7) Family Datasheet
5. IT8712F Environment Control-Low Pin Count Input/ Output Specification
6. Intel 82574 GbE Controller Family Datasheet