MiniOS7 API Functions Reference Manual

(For C Language)

Version 1.0, 28 August 2007

Original Writer：Tim Tsai
Last Editer： Vic Tsai

I-7188 series					
I-7188(D)	I-7188XA(D)	I-7188XB(D)	I-7188XC(D)	I-7188EX(D)	I-7188EA(D)
I-752N	(I-7521(D), I-7522(D), I-7523(D), I-7522A(D), I-7524(D), I-7527(D))				
I-7188EN	(I-7188E1(D), I-7188E2(D), I-7188E3(D), I-7188E4(D), I-7188E5(D), I-7188E8(D)				

I-8000 series					
I-8411	I-8811	I-8431	I-8831	I-8431-80	I-8831-80

Introduction

This manual is to develop user's own application programs with those embedded controllers listed above. This manual introduces what kinds of functions a library has, what kinds of libraries can be used with a embedded controller, and detail descriptions of those libraries and their corresponding functions.

If you have any problem, please contact: service@icpdas.com.

1. Library Selection for All I-7188/I-8000 Series Modules

For I-7188XA/ I-7188XB/ I-7188XC/ I-752N(I-7521,I-7522, I-7523, I-7522A, I-7524, I-7527)/ I-7188EX/ I-7188EA/ I-7188EN (I-7188E1, I-7188E2, I-7188E3, I-7188E4, I-7188E5, I-7188E8)

Supports a single memory model:

(1) 7188EL.LIB, 7188XAL.LIB, 7188XBL.LIB, 7188XCL.LIB is for LARGE MODEL. (TC/BC++/MSC/MSVC++)

All function declarations are included in the 7188x.h file, so the following line should be added to the beginning of the program:

#include "7188E.h"	for I-7188EX/ I-7188EA/ I-7188EN
#include "7188XA.h"	for I-7188XA
#include "7188XB.h"	for I-7188XB/ I-7522A/ I-7524/ I-7527
#include "7188XC.h"	for I-7188XC/ I-7521/ I-7522/ I-7523

(7188EL.LIB, 7188XAL.LIB, 7188XBL.LIB, 7188XCL.LIB and 7188EL.h, 7188XAL.h, 7188XBL.h, 7188XCL.h are located in the following directory:
Companion CD:\Napdos\7188XABC\7188XA\Demo/ or
http://ftp.icpdas.com.tw/pub/cd/8000cd/napdos/7188xabc/7188xa/demo/)

For I-8000 series modules

Supports a single memory model:

(1) 8000E.LIB is for LARGE MODEL. (TC/BC++/MSC/MSVC++)

All function declarations are included in 8000E.h file, so the following line should be added to the beginning of the program:

#include "8000E.h"
(8000E.LIB and 8000E.h are located in the following directory:
Companion CD:\Napdos\8000\841x881x\Demo\Lib or
http://ftp.icpdas.com.tw/pub/cd/8000cd/napdos/8000/841x881x/demo/Lib/)

For I-7188

Supports a single memory model:

(1) 7188L.LIB is for LARGE MODEL. (TC/BC++/MSC/MSVC++)

All function declarations are included in i7188.h file, so the following line should be added to the beginning of the program:

#include "i7188.h"
(7188L.LIB is located in the following directory:
Companion CD:\Napdos\7188\MINIOS7\DEMO\lib\ or
http://ftp.icpdas.com.tw/pub/cd/8000cd/napdos/7188/minios7/demo/lib/)

NOTE: ***InitLib()*** must be called before any other functions are used in the library.
InitLib() described is in Sec.14 (Others (MISC) Functions)

2. COM Ports Functions

The COM Port hardware

[I-7188XB]/[I-7188XC]/[I-752N]

1. By default, I-7188X/I-752N series modules contain two COM Ports, COM1 is both RS-232 and RS-485 (use diffrenet pins to select RS-232 or RS-485), COM2 is RS-485 only.
2. COM1 is the standard I/O port for the MiniOS7 and is used to receive commands and download files. User programs can also use COM1 to connect to other RS-232/RS-485 devices.
3. The two COM PORTS use the internal UART of the CPU (Am188ES), and are not compatible with 16C550 on hardware level.
4. A built-in Self-Tuner is included in the RS-485 of I-7188X/I-752N series modules, so there is no need for the RS-485 data direction to be controlled by the software .
5. I-7522/I-7523 modules are the I-7521 with a daughter board (X501/X502) that adds 1 or 2 additional RS-232 ports (COM3/4).
6. I-7522A/I-7524/I-7527 modules are the I-7188XB with a daughter board (X507/X505/X506) that adds an additional RS-422 ports and an additional 3 or 6 RS-232 ports.

[I-7188EX]/[I-7188EA]/[I-7188EN]

1. By default, I-7188EX series modules contain two COM Ports, except for I-7188E1. COM1 is RS-232 and COM2 is RS-485
2. COM1 is the standard I/O port for the MiniOS7 and is used to receive commands and download files.
3. The two COM PORTS use the internal UART of the CPU (Am188ES) and are not compatible with the 16C550 on the hardware level.
4. A built-in Self-Tuner is included in the RS-485 of I-7188Ex series modules, so there is no need for the RS-485 data direction to be controlled by the software.

[8000]

I-8000 series support up to 5 COM ports:

COM Port	Serial protocol	UART	Descriptions
COM0	RS-232	CPU internal uart	For 87K modules.
COM1	RS-232	CPU internal uart	The default standard I/O port for the MiniOS7 and is used to receive commands and download files. User programs can also use COM1 to connect to other RS-232 devices.
COM2	RS-485	16C550	The maximun Baud Rate is 921000 bps.
COM3	RS-232/RS-485	16C550	Supports RS-232 signals: RX, TX, CTS, RTS, GND

COM4	RS-232	16C550	RS-232 is 9-wire.

[I-7188]/[I-7188XA]

I-7188/I-7188XA support 4 COM ports:

COM Port	Serial protocol	UART	Descriptions
COM1	RS-232/RS-485	16C550	RS-232 is 9-wire and uses jumper to select RS-232 or RS-485. A built-in Self-Tuner is included on the RS-485 of I-7188XA, but not in the I-7188.
COM2	RS-485	16C550	A built-in Self-Tuner is included on the RS-485 of I-7188XA, but not in the I-7188.
COM3	RS-232	CPU internal uart	
COM4	RS-232	CPU internal uart	The default standard I/O port for the MiniOS7 and is used to receive commands and download files. User programs can also use COM4 to connect to other RS-232 devices.

2.1 Common Functions/Variables for all I-7188/I-8000 Series Modules

2.1.1 InstallCom

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	*1
COM 1	*2	*2	*1	*1	*1	*1	*1	*1	*1
COM 2	*2	*2	*1	*1	*1	*1	*1	*1	*2
COM 3	*1	*1	*3	*3	*3	*3	*3	*3	*2
COM 4	*1	*1	*3	*3	*3	*3	*3	*3	*2
COM 5	—	*3	*3	—	*3	*3	*3	*3	—
COM 6	—	*3	*3	—	*3	*3	*3	*3	—
COM 7	—	*3	*3	—	*3	*3	*3	*3	—
COM 8	—	—	*3	—	*3	*3	*3	*3	—
*1: Uses Am188ES internal UART *2: Uses 16C550 *3: Uses 16C550 on the X500~X511/X518/X560									

Description

This function installs the driver for the COM port for COM1 to COM8 in the interrupt vector, which means that all COM ports use the interrupt mechanism to read and write data. (Input buffer size: 1024 bytes, Output buffer size: 1024 bytes)

Prototype

int InstallCom(int port, unsigned long baud, int data, int parity,int stop);

Arguments

	Am188ES internal UART	**16C550**
port	0-8 for COM0~COM8	0-8 for COM0~COM8
baud	300 to 115200 bps	300 to 115200 bps
data	7/8	5/6/7/8
parity	0 (None)/1 (Even)/2 (Odd)	0 (None)/1 (Even)/2 (Odd)/3 (MARK, always 1)/ 4 (SPACE,always 0)
stop	2 (data bit must be 7 bit) or 1	1 or 2

Return Values

0 (NoError)	oO success
-1 (PortError)	port is not 0-8.
-2 (DataError)	data is invalid
-3 (ParityError)	parity is invalid
-4 (StopError)	stop is invalid
-13 (BaudRateError)	baud is invalid
-20 (OutOfMemory)	Th function failed to allocate memory for the output and input buffers.

2.1.2 bCtsChanged_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use bCtsChanged_x if there is a CTS pin for the COM Port.									

Description

This function checks whether the CTS pin has changed. The variable indicates that the CPU has changed the state of the CTS pin since the last time it was ready.

Prototype

```
extern int bCtsChanged_1;
extern int bCtsChanged_3;
extern int bCtsChanged_4;
extern int bCtsChanged_5;
```

Arguments

bCtsChanged_x=1 The CPU has changed the state of the CTS pin since the last time it was ready.
bCtsChanged_x=0 The CPU has not changed the state of the CTS pin since the last time it was ready.

2.1.3 CurCTS_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use CurCTS_x if there is a CTS pin for the COM Port.									

Description

This function reads the current state of the CTS pin.

Prototype

```
extern int CurCTS_1;
extern int CurCTS_3;
extern int CurCTS_4;
extern int CurCTS_5;
```

Arguments

CurCTS_1=1 The CTS pin is active (low).
CurCTS_1=0 The CTS pin is inactive (high).

2.1.4 CurRTS_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use CurRTS_x if there is a RTS pin for the COM Port.									

Description

This function reads the current state of the RTS pin.

Prototype

```
extern int CurRTS_1;
extern int CurRTS_3;
extern int CurRTS_4;
extern int CurRTS_5;
```

Arguments

CurRTS_1=1 The RTS pin is active (low).
CurRTS_1=0 The RTS pin is inactive (high).

2.1.5 fCtsControlMode_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use fCtsControlMode _x if there is a CTS pin for the COM Port.									

Description

This function reads the SetCtsControlMode setting mode value

Prototype

```
extern int fCtsControlMode_1;
extern int fCtsControlMode_3;
extern int fCtsControlMode_4;
extern int fCtsControlMode_5;
```

Arguments

Refer to SetCtsControlMode (Sect.2.1.23) for more details about the CTS setting mode value.

2.1.6 fRtsControlMode_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use fRtsControlMode_x if there is a RTS pin for the COM Port.									

Description

This function reads the SetRtsControlMode setting mode value

Prototype

```
extern int fRtsControlMode_1;
extern int fRtsControlMode_3;
extern int fRtsControlMode_4;
extern int fRtsControlMode_5;
```

Arguments

Refer to SetRtsControlMode (Sect.2.1.24) for more details about the setting mode value.

2.1.7 ClearCom

Description

This function clears all the data in input buffer of the COM port.
It also clears the status of the software flow control (Xon/Xoff) for COM1~COM8 except for COM2 and clears the status of software flow control (RTS/CTS) for COM1~COM8 except for COM2.

Prototype

int ClearCom(int port);

Arguments

port	0-8 for COM0~COM8.

Return Values

0 (NoError)	On success
-1 (PortError)	The port is not valid (0-8).

2.1.8 ClearTxBuffer

Description

This function clears all data from the output buffer of the COM Port. No data are sent.

Prototype

int ClearTxBuffer(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.9 DataSizeInCom

Description

This function reads the size of the data in the input buffer of the COM Port.

Prototype

int DataSizeInCom(int port);

Arguments

port	0-8 for COM0~COM8

Return Value

Non-negative	The amount of in the input buffer
-1 (PortError)	port is not valid (0-8).

2.1.10 GetTxBufferFreeSize

Description

This function reads the value of the available space in the output buffer of the COM Port.

Prototype

int GetTxBufferFreeSize(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

Non-negative	The value of the available space in the output buffer.
-1 (PortError)	port is not valid (0-8).

2.1.11 GetCtsStatus

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use GetCtsStatus if there is a CTS pin for the COM Port									

Description

This function reads the status of the CTS pin of the COM Port.

Prototype

int GetCtsStatus(int port);

Arguments

port	0-8 for COM0~COM8. The CTS pin is effective based on the list above.

Return Values

1	For CTS is active (low)
0	For CTS is not active (high)
-1	Port is not valid (0-8).

2.1.12 InstallComInputData

Description

This function allows user functions to handle data from the COM Port hardware.
After InstallComInputData for specified port, functions related to input data such as IsCom(specified port,….), IsTXBufEmpty(specified port,….), ReadCom(specified port,….), ReadComn(specified port,….) and so on, are invalid and the system calls the user function to handle the input data.

Prototype

int InstallComInputData(int port, int (*DoInputData)(unsigned char data));

Arguments

port	0-8 for COM0~COM8
DoInputData	A function pointer

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.13 IsCom

Description

This function checks whether there is any data in the COM Port input buffer.

Prototype

int IsCom(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

1 (QueueIsNotEmpty)	The queue is not empty.
0 (QueueIsEmpty)	There is no data in the COM Port input buffer
-7 (QueueOverflow)	When the return value is QueueOverflow(-7), ReadCom() or ClearCom() must be called to reset the overflow status.

2.1.14 IsTxBufEmpty

Description

This function checks if the output buffer of the COM port hardware is empty (All data has been sent).

Prototype

int IsTxBufEmpty(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

1	Empty
0	Not Empty
-1 (PortError)	port is not is valid (0-8).

2.1.15 IsComOutBufEmpty

Description

This function checks if the output buffer of COM port software is empty (All data has been sent).

Prototype

int IsComOutBufEmpty(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

1	Empty
0	Not Empty
-1 (PortError)	port is not valid (0-8).

2.1.16 IsDetectBreak

Description

This function checks if the COM port has received a BREAK signal.

Prototype

int IsDetectBreak(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

1	A break signal was detected.
0	No signal detected.
-1 (PortError)	port is not valid (0-8).

2.1.17 printCom

Description

This function allows data to be output from the COM Port in the same way as the printf function in the C library.

Prototype

int printCom(int port,char *fmt,...);

Arguments

port	0-8 for COM0~COM8
fmt	Please refer to library printf in the C.

Return Values

Non-negative	The number of bytes sent to the COM Port.
-1 (PortError)	port is not valid (0-8).
EOF	See the C library.

2.1.18 RestoreCom

Description

If the program calls InstallCom, RestoreCom must be called to restore the COM Port setting before exiting the program.

Prototype

int RestoreCom(int port);

Arguments

port	0 only for I-8000 series modules. 1~8 for I-7188/I-8000 series modules.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.19 ReadCom

Description

This function reads one byte of data from the input buffer of the COM Port.

Prototype

int ReadCom(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

0-255	Data read
-1 (PortError)	port is not valid (0-8).
-6 (QueueEmpty)	The queue is empty

2.1.20 ReadComn

Description

This function reads ‘N’ bytes of data from the input buffer of the COM Port at one time.
For example:
"Num" is the total number of bytes in the input buffer of the COM Port.
If n < "Num", ‘N’=n. So n bytes of data are read from the input buffer of the COM Port at one time.
If n > "Num", ‘N=Num. So "Num" bytes of data are read from the input buffer of the COM Port at one time.

Prototype

int ReadComn(int port, unsigned char *buf, int n);

Arguments

port	0-8 for COM0~COM8
buf	The data buffer stores the data that is read from the input buffer of the COM Port
n	The maximum number of bytes that can be read from the input buffer of the COM port (if n < "Num"). The n value must be less than or equal to the size of buf.

Return Value

Non-negative	The specified amount of data that has been read from the input buffer.
-1 (PortError)	port is not valid (0-8).

2.1.21 SetComTimeout

Description

When the output buffer is full, the ToCom function waits for “t” ms to check whether there is any free space in the output buffer to store data.
The default value is 10 ms

Prototype

int SetComTimeout(int port, unsigned t);

Arguments

port	0-8 for COM0~COM8
t	0-65535 for the timeout value. The unit is ms.

Return Value

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.22 SetComPortBufferSize

Description

This function sets the input and output buffer size for the Com Port.
The default value for both the input and output buffer size is 1024.

Prototype

int SetComPortBufferSize(int port, int in_size, int out_size);

Arguments

port	0-8 for COM0~COM8
in_size	0-32767.
out_size	0-32767.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.23 SetCtsControlMode

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use SetRtsActive if there is a RTS pin for the COM Port.									

Description

This function sets the flow control (RTS/CTS) for the COM Port
The default mode is FLOW_CONTROL_DISABLE

The mode definitions are as follows:
#define FLOW_CONTROL_DISABLE 0
#define FLOW_CONTROL_ENABLE 1
#define FLOW_CONTROL_AUTO_BY_HW 2
#define FLOW_CONTROL_AUTO_BY_SW 3

Prototype

int SetCtsControlMode(int port, int mode);

Arguments

port	0-8 for COM0~COM8
mode	(1) mode= FLOW_CONTROL_DISABLE: Either SetCtsControlMode(port, FLOW_CONTROL_DISABLE) or SetRtsControlMode(port, FLOW_CONTROL_DISABLE) **disables the hardware flow control.** (2) mode= FLOW_CONTROL_ENABLE: Enables the RTS/CTS software flow control, but the active or inactive status of the RTS pin must be controlled by calling the SetRtsActive or SetRtsInactive function to complete the **RTS/CTS software flow control.** (3) mode= FLOW_CONTROL_AUTO_BY_HW: Either SetCtsControlMode(port, FLOW_CONTROL_AUTO_BY_HW) or SetRtsControlMode(port, FLOW_CONTROL_AUTO_BY_HW) **enables the automatic hardware flow control.** (4) mode= FLOW_CONTROL_AUTO_BY_SW: Both SetRtsControlMode(port, FLOW_CONTROL_AUTO_BY_SW) and SetCtsControlMode(port, FLOW_CONTROL_AUTO_BY_SW) **enable the automatic RTS/CTS software flow control.**

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.24 SetRtsControlMode

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use SetRtsActive if there is a RTS pin for the COM Port.									

Description

This function sets the flow control (RTS/CTS) for the COM Port
The default mode is FLOW_CONTROL_DISABLE

The mode definitions are as follows:

```
#define FLOW_CONTROL_DISABLE        0
#define FLOW_CONTROL_ENABLE         1
#define FLOW_CONTROL_AUTO_BY_HW     2
#define FLOW_CONTROL_AUTO_BY_SW     3
```

Prototype

int SetRtsControlMode(int port, int mode);

Arguments

port	0-8 for COM0~COM8
mode	(1) mode= FLOW_CONTROL_DISABLE: Either SetCtsControlMode(port, FLOW_CONTROL_DISABLE) or SetRtsControlMode(port, FLOW_CONTROL_DISABLE) **disables the hardware flow control.** (2) mode= FLOW_CONTROL_ENABLE: This is not used by the SetRtsControlMode function. (3) mode= FLOW_CONTROL_AUTO_BY_HW: Either SetCtsControlMode(port, FLOW_CONTROL_AUTO_BY_HW) or SetRtsControlMode(port, FLOW_CONTROL_AUTO_BY_HW) **enables the automatic hardware flow control.** (4) mode= FLOW_CONTROL_AUTO_BY_SW: Both SetCtsControlMode(port, FLOW_CONTROL_AUTO_BY_SW) and SetCtsControlMode(port, FLOW_CONTROL_AUTO_BY_SW) **enable the automatic RTS/CTS software flow control.**

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.25 SetXonXoffControlMode

Description

This function sets the automatic software flow control (Xon/Xoff) for the COM Port to be enabled or disabled. If the COM Port is RS-485 and is used in half-duplex mode, flow control is not required.
The default mode is 0.

Prototype

int SetXonXoffControlMode(int port, int mode);

Arguments

port	0-8 for COM0~COM8
mode	1: Enabled 0: Disabled

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.26 SetRtsActive

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use SetRtsActive if there is a RTS pin for the COM Port.									

Description

This function sets the output of the RTS pin of the COM Port to LOW (RS-232 is low active).

Prototype

int SetRtsActive(int port);

Arguments

port	0-8 for COM0~COM8. The RTS pin is effective based on the list above.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.27 SetRtsInactive

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	—
COM 1	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	—	—	—	—	—	—	—
COM 3	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
COM 4	—	—	**Y**	—	**Y**	**Y**	**Y**	**Y**	**Y**
COM 5	—	**Y**	**Y**	—	**Y**	**Y**	**Y**	**Y**	—
COM 6	—	—	—	—	—	—	—	—	—
COM 7	—	—	—	—	—	—	—	—	—
COM 8	—	—	—	—	—	—	—	—	—
Y: The COM Port can use SetRtsInactive if there is a RTS pin for the COM Port.									

Description

Set COM port's RTS pin output HIGH (RS-232 is low active).

Prototype

int SetRtsInactive(int port);

Arguments

port	0-8 for COM0~COM8. The RTS pin is effective based on the list above.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.28 SetBaudrate

Description

This function sets the Baud Rate of the COM Port.

Prototype

int SetBaudrate(int port,unsigned long baud);

Arguments

port	0-8 for COM0~COM8
baud	The new Baud Rate. Can be 300 to 115200.

Return Values

0 (NoError)	On success.
-1 (PortError)	The port is not valid (0-8).
-13 (BaudRateError)	The Baud Rate is invalid

2.1.29 SetDataFormat

Description

This function sets the data format for the COM Port.

Prototype

int SetDataFormat(int port, int databit, int parity, int stopbit);

Arguments

	Am188ES internal UART	**16C550**
port	0-8 for COM0~COM8	0-8 for COM0~COM8
data	7/8	5/6/7/8
parity	0 (None)/1 (Even)/2 (Odd)	0 (None)/1 (Even)/2 (Odd)/3 (MARK, always 1)/ 4 (SPACE,always 0)
stop	2 (data bit must be 7 bit) or 1	1 or 2

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).
-2 (DataError)	data is invalid
-3 (ParityError)	parity is invalid
-4 (StopError)	stop is invalid

2.1.30 SendBreak

Description

This function sends a BREAK signal to the COM Port until the timems has elapsed.

Prototype

int SendBreak(int port, unsigned timems);

Arguments

port	0-8 for COM0~COM8.
timems	The time period for sending the break signal. (time unit is ms)

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.31 SetBreakMode

Description

This function sends a BREAK signal to the COM Port or stop a BREAK signal from being sent to COM Port.

Prototype

int SetBreakMode(int port, int mode);

Arguments

port	0-8 for COM0~COM8
mode	1: Start sending a BREAK signal. 0: Stop sending a BREAK signal.

Return Value

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).

2.1.32 ToCom

Description

This function sends one byte of data to the COM Port. (The data are stored in the output buffer first, then when the hardware output buffer is empty, the data are sent by the ISR.)

Prototype

int ToCom(int port, int data);

Arguments

port	0-8 for COM0~COM8
data	0-255, the data to be transmitted.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).
-5 (TimeOut)	Timeout

2.1.33 ToComStr

Description

This function sends a string (ending with '\0') to the COM Port.

Prototype

int ToComStr(int port, char *str);

Arguments

port	0-8 for COM0~COM8
str	The string to be transmitted.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).
-5 (TimeOut)	Timeout

2.1.34 ToComBufn

Description

This function sends n bytes of data to the COM port.

Prototype

int ToComBufn(int port, char *buf, int num);

Arguments

port	0-8 for COM0~COM8
buf	The data buffer.
num	The amount of data (bytes) transmitted.

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).
-5 (TimeOut)	Timeout

2.1.35 WaitTransmitOver

Description

This function waits all data in the output buffer of the software and the hardware of the COM Port to be sent.

Prototype

int WaitTransmitOver(int port);

Arguments

port	0-8 for COM0~COM8

Return Values

0 (NoError)	On success
-1 (PortError)	port is not valid (0-8).
-5 (TimeOut)	Timeout

2.1.36 InstallCom_DMA_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y: The COM Port can use the InstallCom_DMA_x function									

Description

Direct Memory Access (DMA) permits the transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between InstallCom and InstallCom_DMA_x is the behavior of the data transfer.
The InstallCom_DMA_x function has to be called before the other DMA functons are used.

Prototype

int InstallCom_DMA_0(unsigned long baud, int data, int parity, int stop);
int InstallCom_DMA_1(unsigned long baud, int data, int parity, int stop);
int InstallCom_DMA_2(unsigned long baud, int data, int parity, int stop);
int InstallCom_DMA_3(unsigned long baud, int data, int parity, int stop);
int InstallCom_DMA_4(unsigned long baud, int data, int parity, int stop);

Arguments

	Am188ES internal UART
baud	300 to 115200 bps
data	7/8
parity	0 (None)/1 (Even)/2 (Odd)
stop	2 (data bit must be 7 bit) or 1

Return Values

0 (NoError)	On success
-2 (DataError)	data is invalid
-3 (ParityError)	parity is invalid
-4 (StopError)	stop is invalid
-13 (BaudRateError)	baud is invalid
-20 (OutOfMemory)	The function failed to allocate memory for the output and input buffers.

2.1.37 IsCom_DMA_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y: The COM Port can use the IsCom_DMA_x function									

Description

Direct Memory Access (DMA) permits transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between IsCom and IsCom_DMA_x is the behavior of the data transfer.

Prototype

```
int IsCom_DMA_0(void);
int IsCom_DMA_1(void);
int IsCom_DMA_2(void);
int IsCom_DMA_3(void);
int IsCom_DMA_4(void);
```

Arguments

None

Return Values

0 (QueueIsEmpty)	There is no data in the input buffer of the COM Port.
1 (QueueIsNotEmpty)	There is data in the input buffer of the COM Port.

2.1.38 ClearCom_DMA_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y: The COM Port can use the ClearCom_DMA_x function									

Description

Direct Memory Access (DMA) permits the transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between ClearCom and ClearCom_DMA_x is the behavior of the data transfer.

Prototype

```
int ClearCom_DMA_0(void);
int ClearCom_DMA_1(void);
int ClearCom_DMA_2(void);
int ClearCom_DMA_3(void);
int ClearCom_DMA_4(void);
```

Arguments

None

Return Value

0 (NoError)	On success

2.1.39 DataSizeInCom_DMA_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y: The COM Port can use the DataSizeInCom_DMA_x function									

Description

Direct Memory Access (DMA) permits the transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between DataSizeInCom and DataSizeInCom_DMA_x is the behavior of the data transfer.

Prototype

```
int DataSizeInCom_DMA_0(void);
int DataSizeInCom_DMA_1(void);
int DataSizeInCom_DMA_2(void);
int DataSizeInCom_DMA_3(void);
int DataSizeInCom_DMA_4(void);
```

Arguments

None

Return Value

non-negative	The amount of data in the input buffer. (0 means there is no data in the buffer.)

2.1.40 ReadComn_DMA_x

COM port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y:The COM port can use the ReadComn_DMA_x function									

Description

Direct Memory Access (DMA) permits the transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between ReadComn and ReadComn_DMA_x is the behavior of data transfer.

Prototype

```
int ReadComn_DMA_0(unsigned char *buf, int maxsize);
int ReadComn_DMA_1(unsigned char *buf, int maxsize);
int ReadComn_DMA_2(unsigned char *buf, int maxsize);
int ReadComn_DMA_3(unsigned char *buf, int maxsize);
int ReadComn_DMA_4(unsigned char *buf, int maxsize);
```

Arguments

buf	The data buffer stores the data that is read from the input buffer of the COM Port.
maxsize	Refer to ReadComn for details.

Return Values

non-negative	The specified amount of data that has been read from the input buffer.

2.1.41 ReadCom_DMA_x

COM Port	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
COM 0	—	—	—	—	—	—	—	—	**Y**
COM 1	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 2	—	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	—
COM 3	**Y**	**Y**	—	—	—	—	—	—	—
COM 4	**Y**	**Y**	—	—	—	—	—	—	—
Y: The COM Port can use the ReadCom_DMA_x function									

Description

Direct Memory Access (DMA) permits the transfer of data between the memory and the peripherals without involvement of the CPU.
The difference between ReadCom and ReadCom_DMA_x is the behavior of the data transfer.

Prototype

```
int ReadCom_DMA_0(void);
int ReadCom_DMA_1(void);
int ReadCom_DMA_2(void);
int ReadCom_DMA_3(void);
int ReadCom_DMA_4(void);
```

Arguments

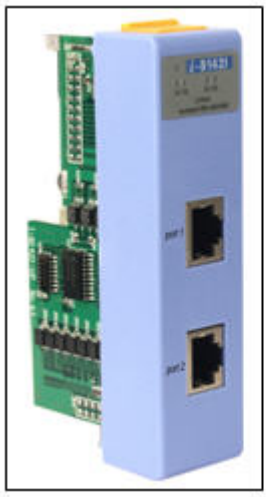
None

Return Values

0-255	The data read
-6 (QueueEmpty)	The queue is empty.

2.2 For I-8142_I-8142i_I-8144_I-8144i_I-8112_I-8114






Remarks
I-8142/I-8142i/I-8144/I-8144i/I-8112/I-8114 can only use the following four slots, slot 0 ~ slot 3.

2.2.1 SetInBufSize

Description

This function sets the size of the software's input buffer for each port of all slots. The input data is stored in the input buffer.

After SetInBufSIze is used, InstallCom8000 must be called to set the software input buffer to the true size.

Prototype

int SetInBufSIze(int size);

Arguments

size	1 to 32767. The Default value is 1024.

Return Value

0 (NoError)	On success

2.2.2 SetOutBufSize

Description

This function sets the size of the software's output buffer for each port of all slots. The output data is stored in the output buffer.

After SetOutBufSIze is used, InstallCom8000 must be called to set the software output buffer to the true size.

Prototype

int SetOutBufSIze(int size);

Arguments

size	1 to 32767. The default value is 1024.

Return Value

0 (NoError)	On success

2.2.3 InstallCom8000

Description

This function sets the configuration for each port of the slot. Automatic hardware and software flow control (CTS/RTS) are supported. Installing the driver for the slot means installing the interrupt service routine. In other words, each port of the slot reads or writes data using the interrupt method.

After running InstallCom8000, the default setting values as follows:

Input buffer=1024 bytes, Output buffer=1024 bytes

Baud Rate=9600, data bit=8, parity bit=none, stop bit=1.

Automatic Hardware or software flow control (CTS/RTS) is disabled.

Prototype

int InstallCom8000(int slot);

Arguments

slot	0/1/2/3.

Return Values

0	On success.
-1	No module is attached to the slot.
-2	The configuration settings for any port of the slot case failed.

2.2.4 RestoreCom8000

Description

If an application calls InstallCom8000(), RestoreCom8000() must then be called to restore the COM Port settings before exiting the application.

Prototype

int RestoreCom8000(int slot);

Arguments

slot	0/1/2/3.

Return Values

0	On success.
-1	There is no module attached to the slot.

2.2.5 _SetBaudrate

Description

This function sets the Baud Rate for the specified COM Port. After calling InstallCom8000, the default Baud Rate value is 9600bps

Prototype

int _SetBaudrate(int slot,int port,unsigned long baud);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
baud	I-8112/I-8114: 15 to 115200 bps. I-8142/I-8142i/I-8144: 15 to 921600 bps.

Return Values

0 (NoError)	On success.
-1	slot or port number is invalid.
-16	baud value is invalid.

2.2.6 _SetDataFormat

Description

This function sets the data format for the specified COM Port. After calling InstallCom8000, the default values as follows: data bit=8, parity bit=none, stop bit=1.

Prototype

int _SetDataFormat(int slot, int port, int data, int parity, int stop);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
data	5/6/7/8.
parity	0 (None)/1 (Even)/2 (Odd)/3 (mark, always 1)/4 (space, always 0)
stop	1/2

Return Values

0	On success.
-1	The slot or port is invalid.
-2	data value is not 5/6/7/8.
-3	parity value is not 0/1/2/3/4.
-4	stop value is not 1/2

2.2.7 ClearCom8000

Description

This function clears all the data from the input buffer of the specified COM Port.

Prototype

int ClearCom8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.8 ClearCom8000TxBuffer

Description

This function clears all the data from the output buffer of the specified COM Port, meaning that the data are not transmitted.

Prototype

void ClearCom8000TxBuffer(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.9 ClrMsrChanged8000

Description

This function clears the parameter that records the change of status of the Modem Status Register (MSR) for the specified COM Port.

Prototype

void ClrMsrChanged8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

None.

2.2.10 GetMsrChanged8000

Description

This function checks whether the status of the Modem Status Register (MSR) for the specified COM Port has changed recently.

Prototype

int GetMsrChanged8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

1	The status of the MSR has changed recently.
0	There has been no change.
-1	The slot or port is invalid.

2.2.11 GetCtsControlMode8000

Description

For the specified COM Port, this function retrieves the latest flow control mode that the SetCtsControlMode8000 was set to. Default mode is FLOW_CONTROL_DISABLE.

The mode definitions are as follows:

#define FLOW_CONTROL_DISABLE 0

#define FLOW_CONTROL_ENABLE 1

#define FLOW_CONTROL_AUTO_BY_HW 2

#define FLOW_CONTROL_AUTO_BY_SW 3

Prototype

int GetCtsControlMode8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

0/1/2/3	On success.
-1	The slot or port is invalid.

2.2.12 GetCom8000TxBufferFreeSize

Description

This function retrieves the amount of available space in the output buffer of the specified COM Port.

Prototype

int GetCom8000TxBufferFreeSize(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

The amount of available space	On success.
-1	The slot or port is invalid.

2.2.13 GetCom8000FifoTriggerLevel

Description

This function retrieves the receiver FIFO trigger level (bytes) value for the specified COM Port. The default trigger level value is 8 (bytes).

Prototype

int GetCom8000FifoTriggerLevel(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

1/ 4/ 8/ 14	On success.
-1	The slot or port is invalid.

2.2.14 GetCurMsr8000

Description

This function retrieves the content of the Modem Status Register (MSR) for the specified COM Port if an MSR interrupt has recently occured.

The definition for each bit is as follows:

#define _MSR_dCTS 0x01

#define _MSR_dDSR 0x02

#define _MSR_TERI 0x04

#define _MSR_dDCD 0x08

#define _MSR_CTS 0x10

#define _MSR_DSR 0x20

#define _MSR_RI 0x40

#define _MSR_DCD 0x80

Prototype

int GetCurMsr8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

The 8-bit status of the MSR	On success.
-1	The slot or port is invalid.

2.2.15 GetCom8000_MSR

Description

This function retrieves the current content of the Modem Status Register (MSR) for the specified COM Port.

The definition for each bit is as follows:

#define _MSR_dCTS 0x01

#define _MSR_dDSR 0x02

#define _MSR_TERI 0x04

#define _MSR_dDCD 0x08

#define _MSR_CTS 0x10

#define _MSR_DSR 0x20

#define _MSR_RI 0x40

#define _MSR_DCD 0x80

Prototype

int GetCom8000_MSR(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Value

The 8-bit status of the MSR	On success.
-1	The slot or port is invalid.

2.2.16 IsCom8000

Description

This function checks if there is any data in the input buffer of the specified COM Port.

Prototype

int IsCom8000(int slot,int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

Non-zero integer	On success (if any data is available).
0	No data is available.
-7	A buffer overflow has occured. Calling ReadCom8000 or ClearCom8000 or ReadCom8000nBytes to reset the overflow status.

2.2.17 IsCom8000OutBufEmpty

Description

This function checks if the software output buffer of specified COM Port is empty (All data was transmitted).

Prototype

int IsCom8000OutBufEmpty(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

1	The output buffer is empty.
0	The output buffer is not empty.
-1	On error.

2.2.18 ReadCom8000

Description

This function reads one byte of data from the input buffer of the specified COM Port.

Prototype

int ReadCom8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

0 to 255	On success.
-1	The slot or port is invalid.
-6	There is no data in the input buffer of the specified COM Port.

2.2.19 ReadCom8000nBytes

Description

This function reads n bytes from the input buffer of the specified COM Port at one time.

For instance:

“Num” is the total number of bytes in the input buffer of the specified COM Port.

“N” is number of bytes actually read.

If maxnum>“Num”, N=”Num”. So “Num” bytes are read from the input buffer at one time.

If maxnum<“Num”, N=maxnum. So maxnum bytes are read from the input buffer at one time.

Prototype

int ReadCom8000nBytes(int slot, int port, char *buf, int maxnum);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
buf	The data buffer stores the data that is read from the input buffer of the specified COM Port.
maxnum	The maximum number of bytes to be read from the input buffer (if maxnum<”Num”). The maxnum value has to be less than or equal to the size of the buf.

Return Values

The specified amount of data to be read from the input buffer.	On success.
-1	The slot or port is invalid.
-6	There is no data in the input buffer of the specified COM Port.

2.2.20 SetRts8000

Description

This function sets the RTS (Request To Send) pin for the specified COM Port as active or inactive.

Prototype

int SetRts8000(int slot, int port, int mode);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mode	1: Active 0: Inactive

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.21 SetCtsControlMode8000

Description

This function sets the flow control (RTS/CTS) for the specified COM Port.

The default mode is FLOW_CONTROL_DISABLE.

The mode definitions are as follows:

#define FLOW_CONTROL_DISABLE 0

#define FLOW_CONTROL_ENABLE 1

#define FLOW_CONTROL_AUTO_BY_HW 2

#define FLOW_CONTROL_AUTO_BY_SW 3

Prototype

int SetCtsControlMode8000(int slot, int port, int mode);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mode	(1) mode=***FLOW_CONTROL_DISABLE***: Either SetCtsControlMode8000 (slot, port, FLOW_CONTROL_DISABLE) or SetRtsControlMode8000 (slot, port, FLOW_CONTROL_DISABLE) **disables the hardware flow control**. (2) mode=***FLOW_CONTROL_ENABLE*** : Enables the RTS/CTS software flow control, but the user needs to control the active or inactive status of the RTS pin by calling the SetRts8000 function to complete the **RTS/CTS software flow control.** (3) mode=***FLOW_CONTROL_AUTO_BY_HW***: Either SetCtsControlMode8000(slot,port, FLOW_CONTROL_AUTO_BY_HW) or SetRtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_HW) **enables automatic hardware flow control**. (4) mode=***FLOW_CONTROL_AUTO_BY_SW***: Both SetCtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_SW) and SetRtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_SW)

	enable automatic RTS/CTS software flow control.

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.22 SetRtsControlMode8000

Description

This function sets the flow control (RTS/CTS) for the specified COM Port.

The default mode is FLOW_CONTROL_DISABLE.

The mode definitions are as follows:

#define FLOW_CONTROL_DISABLE 0

#define FLOW_CONTROL_ENABLE 1

#define FLOW_CONTROL_AUTO_BY_HW 2

#define FLOW_CONTROL_AUTO_BY_SW 3

Prototype

int SetRtsControlMode8000(int slot, int port, int mode);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mode	(1) mode=**FLOW_CONTROL_DISABLE**: Either SetCtsControlMode8000 (slot, port, FLOW_CONTROL_DISABLE) or SetRtsControlMode8000 (slot, port, FLOW_CONTROL_DISABLE) **disables hardware flow control**. (2) mode=**FLOW_CONTROL_ENABLE** : This is not used by the SetRtsControlMode8000 function (3) mode=**FLOW_CONTROL_AUTO_BY_HW**: Either SetCtsControlMode8000(slot, port,FLOW_CONTROL_AUTO_BY_HW) **or** SetRtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_HW) **enables automatic hardware flow control**. (4) mode=**FLOW_CONTROL_AUTO_BY_SW**: Both SetCtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_SW) and SetRtsControlMode8000(slot, port, FLOW_CONTROL_AUTO_BY_SW) **enable automatic software RTS/CTS flow control**.

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.23 SetCom8000FifoTriggerLevel

Description

This function sets the receiver FIFO trigger level (bytes) for the specified COM Port.

The default trigger level value is 8 (bytes).

Prototype

int SetCom8000FifoTriggerLevel(int slot, int port, int level);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
level	1/4/8/14 bytes.

Return Values

-1	The slot or port is invalid.

2.2.24 ToCom8000

Description

This function sends one byte of data to the specified COM Port. The data are initially stored in the software output buffer. When the hardware output buffer is empty, the data are sent using the ISR, Interrupt Service Routine.

Prototype

int ToCom8000(int slot, int port, int data);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
data	0 to 255, the data is ready to be transmitted.

Return Values

0	On success (the data has been stored in the software output buffer).
-1	The slot or port is invalid.
-5	Data transmission has failed.

2.2.25 ToCom8000Str

Description

This function sends a string (terminated with '\0') to the specified COM Port. The string initially is stored in the software output buffer. When the hardware output buffer is empty, the string is sent using the ISR, Interrupt Service Routine.

Prototype

int ToCom8000Str(int slot, int port, char *str);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
str	The string is ready to be transmitted.

Return Values

0	On success (the string has been stored in the software output buffer).
-1	The slot or port is invalid.
-5	Data transmission has failed.

2.2.26 ToCom8000nBytes

Description

This function sends n bytes of data to the specified COM Port. The data are initially stored in the software output buffer. When the hardware output buffer is empty, the data are sent using the ISR, Interrupt Service Routine.

Prototype

int ToCom8000nBytes(int slot, int port, char *buf,int num);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
buf	The data buffer.
num	The number of bytes of data to be sent, where the value “num” is less than or equal to the size of the buf.

Return Values

0	On success (the string has been stored in the software output buffer).
-1	The slot or port is invalid.
-5	Data transmission has failed.

2.2.27 SetCom8000_MCR

Description

This function sets the content of the Modem Control Register (MCR) for the specified COM Port. Only the ability to set bit 0 (DTR) and bit 1 (RTS) of the MCR is supported. To set the RTS as active (also active DTR), call SetCom8000_MCR (slot, port, _MCR_DTR+_MCR_RTS). To set the RTS as inactive (only active DTR), call SetCom8000_MCR (slot, port, _MCR_DTR). Call SetCom8000_MCR (slot, port, 0) to set both RTS and DTR as inactive.

The definition of each bit is as follows:

#define _MCR_DTR 1

#define _MCR_RTS 2

Prototype

int SetCom8000_MCR(int slot, int port, int mcr);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mcr	_MCR_DTR or _MCR_RTS

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.28 SetCom8000_MCR_Bit

Description

This function sets the content of the Modem Control Register (MCR) for the specified COM Port.

Prototype

int SetCom8000_MCR_Bit(int slot, int port, int mcr_bit);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mcr_bit	#define _MCR_DTR 1 #define _MCR_RTS 2 #define _MCR_LOOP 0x10 #define _MCR_AUTO_FLOW_CONTROL 0x20

Return Values

0	On success.
-1	The slot or port is invalid.

2.2.29 ClearCom8000_MCR_Bit

Description

This function clears the Modem Control Register (MCR) for the specified COM Port.

Prototype

int ClearCom8000_MCR_Bit(int slot, int port, int mcr_bit);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mcr_bit	#define _MCR_DTR 1 #define _MCR_RTS 2 #define _MCR_LOOP 0x10 #define _MCR_AUTO_FLOW_CONTROL 0x20

Return Value

0	On success.
-1	The slot or port is invalid.

2.2.30 SetBreakMode8000

Description

This function sets the continuous sending of a break signal as either enabled or disabled for the specified COM Port. The default mode is disabled.

Prototype

void SetBreakMode8000(int slot, int port, int mode);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
mode	Default mode is disabled. 1: Enabled. 0: Disabled.

Return Values

None.

2.2.31 IsDetectBreak8000

Description

This function checks whether the specified COM Port has received a break signal.

Prototype

int IsDetectBreak8000(int slot, int port);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

1	A break signal is detected.
0	No break signal is detected.
-1	The slot or port is invalid.

2.2.32 SendBreak8000

Description

This function sends a continuous break signal to the specified COM Port for a specified period of time (timems).

Prototype

void SendBreak8000(int slot, int port, unsigned timems);

Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3
timems	The period of time to send a continuous break signal (time unit is ms.)

Return Values

None.

2.2.33 WaitCom8000TransmitOver

Description

This function waits for all data in the software and hardware output buffers of of the specified COM Port to be tramsmitted.

Prototype

int WaitCom8000TransmitOver(int slot, int port);

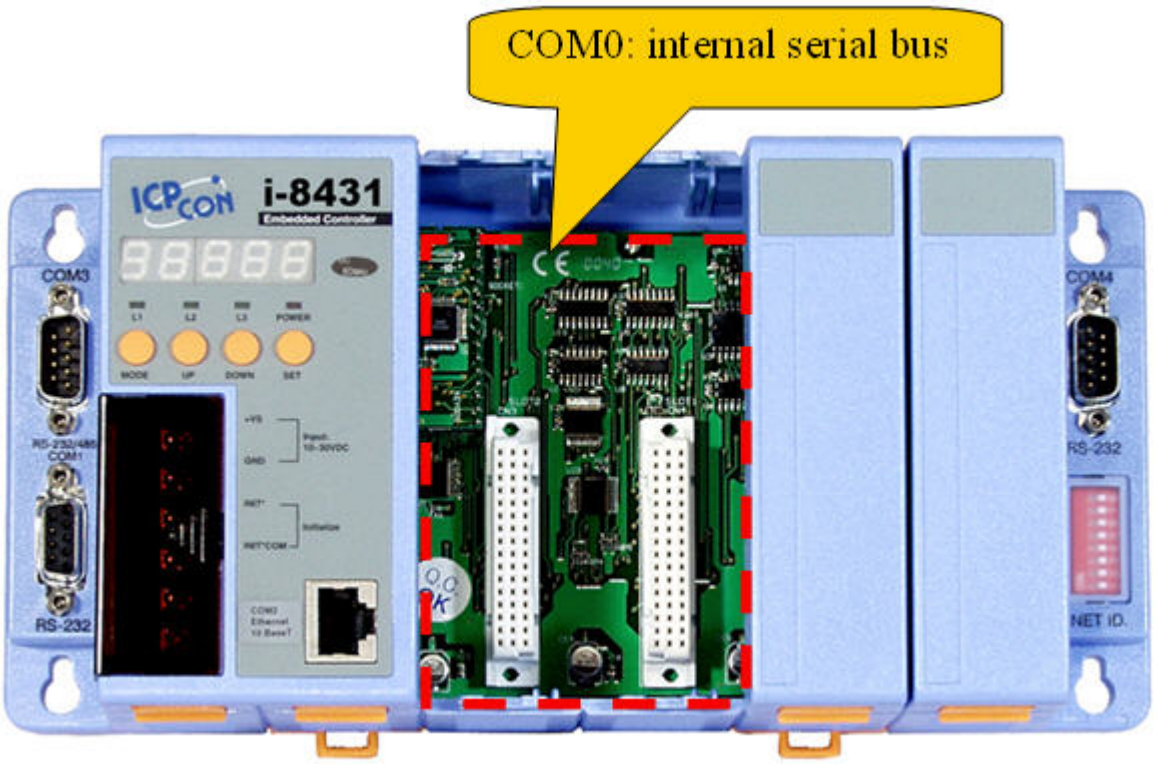
Arguments

slot	0/1/2/3
port	I-8112/I-8142/I-8142i: 0/1 I-8114/I-8144: 0/1/2/3

Return Values

1	The output buffer is empty.
0	On success.
-1	The slot or port is invalid.
-5	Timeout.

2.3 COM0_use_DMA for I-8000 Series



2.3.1 OpenCom0UseDMA

Description

This function uses Direct Memory Access (DMA) method to transmit/receive data to/from COM0.
This function has to be called first and then other DMA functions for COM0 can be used.

Prototype

void OpenCom0UseDMA(void);

Arguments

None

Return Values

None

2.3.2 CloseCom0UseDMA

Description

This function closes the Direct Memory Access (DMA) method for COM0.

Prototype

void CloseCom0UseDMA(void);

Arguments

None

Return Values

None

2.3.3 Com0GetDataSize

Description

This function retrieves the size of the data currently stored in the input buffer of COM0.

Prototype

unsigned Com0GetDataSize(void);

Arguments

None

Return Values

Non-negative	The size of the data stored in the input buffer.

2.3.4 Com0SetInputBuf

Description

This function sets the input buffer where the DMA stores the received data.
User can read Com0GetDataSize to know how much data had been received.

Prototype

void Com0SetInputBuf(char far *ptr, unsigned cnt);

Arguments

ptr	The buffer that stores the data received by the DMA.
cnt	Buffer size.

Return Values

None.

2.4 COM4 without Using Interrupt for I-8431_I-8831

Remark
1. The two functions are suitable only for 40M Hz CPU in this section. 2. When COM0 uses DMA method (Com0GetDataSize, Com0SetInputBuf…) to send/receive data, COM4 has no interrupt to use some functions that send data to COM4 or receive data from COM4, like IsCom(4,..), ReadCom(4,..), ToCom(4,..), ToComStr(4,..), printCom(4,..) and etc. Using EnableMonitorCom4() and DisableMonitorCom4() can solve this problem.

2.4.1 EnableMonitorCom4

Description

This function can let these function that send data to COM4 or receive data from COM4, like IsCom(4,..)/ ReadCom(4,..)/ ToCom(4,..), be used.
For example:

```
…….
EnableMonitorCom4();

If(IsCom(4)){
  …..
  ToCom(4,'\n');
}
……..
DisableMonitorCom4();
```

Prototype

void EnableMonitorCom4(void);

Arguments

None.

Return Values

None.

2.4.2 DisableMonitorCom4

Description

This function can let these function that send data to COM4 or receive data from COM4, like IsCom(4,..)/ ReadCom(4,..)/ ToCom(4,..), be used.
For example:

```
…….
EnableMonitorCom4();

If(IsCom(4)){
  …..
  ToCom(4,'\n');
}
……..
DisableMonitorCom4();
```

Prototype

void DisableMonitorCom4(void);

Arguments

None.

Return Values

None.

3. EEPROM Functions

About the EEPROM hardware
1. The EEPROM of the 7188/I-8000 series modules is 24LC16, contains 8 blocks (block 0 to 7). Each block has 256 bytes (address 0 to 255), so the total size of the EEPROM is 2048 (2K) bytes. 2. The default mode for EEPROM is write-protected mode. 3. Before data can be written to the EEPROM, EE_WriteEnable() must be called to set EEPROM to write-enabled mode. 4. After a write operation to the EEPROM is completed, it is recommended that the EE_WriteProtect() is called to set the EEPROM to write-protect mode.

3.1 Common Variables/Functions for all I-7188/I-8000 Series Modules

3.1.1 EepType

Description

This function reads the type of EEPROM contained in the module.

Prototype

extern int EepType;

Return Values

16	EEPROM is 24LC16. Size is 2k bytes.
128	EEPROM is 24WC128. Size is 16k bytes.

3.1.2 EE_WriteEnable

Description

This function sets the EEPROM to write-enabled mode.

Prototype

void EE_WriteEnable(void);

Arguments

None.

Return Values

None.

3.1.3 EE_WriteProtect

Description

This function sets the EEPROM to write-protected mode.

Prototype

void EE_WriteProtect(void);

Arguments

None.

Return Values

None.

3.1.4 EE_ MultiRead

Description

This function reads multiple bytes of data from the EEPROM (24LC16).

Prototype

int EE_MultiRead(int StartBlock, unsigned StartAddr, int num, char *databuf);

Arguments

StartBlock	0 to 7. (Total is 8 blocks)
StartAddr	0 to 255. (Each block is 256 bytes)
num	1 to 2048. The data length.
databuf	The address where the data is stored.

Return Values

0 (NoError)	On success
-1	The EEPROM is busy or the StartBlock/StartAddr is invalid

3.1.5 EE_ MultiWrite

Description

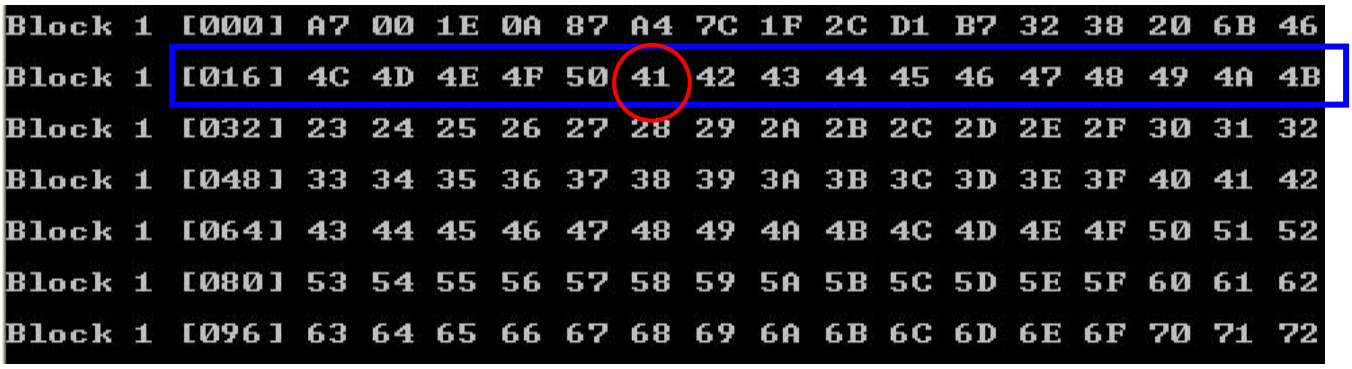
This function writes data to EEPROM (24LC16). The maximum write size is 16 bytes, and must be written to the same block.

1. An additional limitation is that only the lowest 4 bits of the address can be changed.

The value variation: 0->1->2->3->4->5->6->7->8->9->0xA->0xB->0xC->0xD->0xE->0xF->0.

2. The highest 4 bits of the address can't be changed.

For example: EE_MultiWrite(1, 21, 16, "ABCDEFGHIJKLMNOP");



16 bytes "A~P" write to the EEPROM from address 21 (0x15). They should end at 36 (0x24). But according to this limitation, it returns to address 0x10 when data is written to address 0x1F. The data "L~P" are written from address 0x10 to 0x14.

3. It is recommended that "Addr" uses 0x00/0x10/0x20~0xF0.

Prototype

int EE_MultiWrite(int Block,unsigned Addr,int num,char *Data);

Arguments

Block	0 to 7. (Total is 8 blocks)
Addr	0 to 255. (Each block is 256 bytes)
num	1 to 16. The data length.
Data	The starting address of the buffer where the data is stored.

Return Values

0 (NoError)	On success
-1	The EEPROM is busy, or the Block or Addr is invalid.

3.1.6 EE_MultiRead_L

Description

This function reads multiple bytes of data from the EEPROM.

Prototype

int EE_MultiRead_L(unsigned address, unsigned num, char *Data);

Arguments

address	0 to 2047.
num	1 to 2048. The data length.
Data	The address where the data is stored.

Return Values

0 (NoError)	on success
-1	The EEPROM is busy or the address is invalid

3.1.7 EE_MultiWrite_L

Description

The function writes multiple bytes of data to the EEPROM.

Prototype

int EE_MultiWrite_L(unsigned address, unsigned num, char *Data);

Arguments

address	0 to 2047.
num	1 to 2048. Data length.
Data	The starting address of the buffer where the data is stored.

Return Values

0 (NoError)	On success
-1	The EEPROM busy or the address is invalid
-2	The no+Addr is out of range

3.1.8 XEE_Init

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function initializes the parameters for the EEPROM (24LC16) on an X-board.
The functions (XEE_xxxxxxx) can be used after XEE_Init is called.

Prototype

int XEE_Init(int clk_pin, int sda_pin, int wp_pin, int need_pullhigh);

Arguments

clk_pin	The PIO pin number of the CPU that is connected to the CLK pin of the 24LC16
sda_pin	The PIO pin number of the CPU that is connected to the SDA pin of the 24LC16
wp_pin	The PIO pin number of the CPU that is connected to the WP pin of the 24LC16
need_pullhigh	If the sda_pin is set to input mode➔ 0: Pullhigh resistance is not needed. 1: Pullhigh resistance is needed.

Return Values

3.1.9 XEE_InitByName

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function initializes the parameters for the EEPROM (24LC16) on an X-board.
The functions (XEE_xxxxxxx) can be used after XEE_InitByName is called.
XEE_InitByName is the same as XEE_Init except for the input parameters.
The user just chooses either XEE_Init or XEE_InitByName to initialize the EEPROM.

Prototype

int XEE_InitByName(int Xboard);

Arguments

Xboard	#define _X201_ 201 #define _X202_ 202 #define _X203_ 203 #define _X205_ 205 #define _X206_ 206 #define _X300_ 300 #define _X301_ 301 #define _X302_ 302 #define _X303_ 303 #define _X304_ 304 #define _X305_ 305 #define _X306_ 306 #define _X308_ 308 #define _X309_ 309 #define _X310_ 310 #define _X314_ 314 The new value can be added at any time. Refer to the xxxx.h file (for example 7188e.h) to get the latest value.

Return Values

0	On success
-1	The board is unsupported.

3.1.10 XEE_WriteEnable

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function set the EEPROM (24LC16) on the X-board to write-enabled mode.

Prototype

void XEE_WriteEnable(void);

Arguments

None

Return Values

None

3.1.11 XEE_WriteProtect

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function sets EEPROM (24LC16) on the X-board to write-protected mode.

Prototype

void XEE_WriteProtect(void);

Arguments

None

Return Values

None

3.1.12 XEE_MultiRead

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function reads multiple bytes of data from the EEPROM (24LC16) on the X-board.

Prototype

int XEE_MultiRead(int StartBlock, int StartAddr, int num, char *databuf);

Arguments

StartBlock	0 to 7. (Total is 8 blocks)
StartAddr	0 to 255. (Each block is 256 bytes)
num	1 to 2048
databuf	The address where the data is to be stored.

Return Values

0 (NoError)	On success.
-9 (AddrError)	StartAddr is invalid.
-10 (BlockError)	StartBlock is invalid.

3.1.13 XEE_MultiWrite

	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
Support	—	—	Y	Y	—	Y	—	—	—
Y: This module is supported.									

Description

This function writes data to the EEPROM (24LC16) on the X-board. The maximun write size is 16 bytes, and must be written to the same block.

1. An additional limitation is that only the lowest 4 bits of the address can be changed.

The value variation: 0->1->2->3->4->5->6->7->8->9->0xA->0xB->0xC->0xD->0xE->0xF->0.

2. The highest 4 bits of the address can't be changed.

For example: XEE_MultiWrite(1, 21, 16, "ABCDEFGHIJKLMNOP");

```
Block 1 [000] A7 00 1E 0A 87 A4 7C 1F 2C D1 B7 32 38 20 6B 46
Block 1 [016] 4C 4D 4E 4F 50 41 42 43 44 45 46 47 48 49 4A 4B
Block 1 [032] 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32
Block 1 [048] 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42
Block 1 [064] 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52
Block 1 [080] 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62
Block 1 [096] 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72
```

16 bytes "A~P" write to the EEPROM from address 21(0x15). They should end at 36(0x24). But according to this limitation, it returns to address 16(0x10) when data is written to address 1F. The data "L~P" are written from address 0x10 to 0x14.

3. It is recommended that "Addr" uses 0x00/0x10/0x20~0xF0

Prototype

int XEE_MultiWrite(int Block, int Addr, int num, char *Data);

Arguments

Block	0 to 7. (Total is 8 blocks)
Addr	0 to 255. (Each block is 256 bytes)
num	1 to 16
Data	The starting address of the buffer where the data is stored.

Return Values

0 (NoError)	On success
-1	The EEPROM is busy, the Block or Addr is invalid.

4. Flash Memory Functions

Remark
1. There are two size for the Flash Memory used on I-7188/I-8000 series. 256K bytes and 512K bytes. MiniOs7 uses the last 64K bytes, the other parts of the memory are used to store user's program or data. User's program also can use functions to write data to Flash Memory. 2. When user want to write data to Flash Memory, it only can be written from "1" to "0", can not be written from "0" to "1". So in generally, before it is written data to FLASH, it must erase FLASH, the erase process makes all data to 0xFF, that is all data bit is "1". Then any data can be written to it.

4.1 Common Functions for all I-7188/I-8000 Series

4.1.1 FlashReadId

Description

Read Flash memory device code (high byte) and manufacture code (low byte).

Prototype

int FlashReadId(void);

Arguments

None.

Return Values

0xB0C2 (MXIC 29f002NT),0xB001(AMD 29F002T)	256K type
0xA4C2 (MXIC 29f040), 0xA401 (AMD 29f040) 0xB5C2 MXIC 29LV004T	512K type

4.1.2 FlashErase

Description

Erases one sector (64K bytes) of the Flash Memory, all data in that sector are changed to 0xFF.

Prototype

int FlashErase(unsigned seg);

Arguments

seg	(1) For 256K type, seg can be 0xC000, 0xD000, 0xE000. (2) For 512K type, seg can be 0x8000, 0x9000, 0xA000, 0xB000, 0xC000, 0xD000, 0xE000. (3) The segment 0xF000 is used for MiniOs7. If seg=0xF000, FlashErase does nothing.

Return Value

0 (NoError)	On success
-5 (TimeOut)	On failure

4.1.3 FlashWrite

Description

Writes one byte of data to the Flash memory.
When data is written to the the Flash Memory, each bit can only be changed from 1 to 0. So if the data in that position is 0xff, any data can be written. But if the data in that position is 0x01, only 0x00 can be written.
When an attempt is made to change the bit from 0 to 1, TimeoutError occurs. Data can be written again only afters FlashErase is called.

Prototype

int FlashWrite(unsigned int seg, unsigned int offset, char data);

Arguments

seg	(1) For 256K type, seg can be 0xC000, 0xD000, 0xE000. (2) For 512K type, seg can be 0x8000, 0x9000, 0xA000, 0xB000, 0xC000, 0xD000, 0xE000. (3) The segment 0xF000 is used for MiniOs7. If seg=0xF000, FlashWrite does nothing.
offset	0 to 65535(0xffff).
data	0 to 255(8-bit data).

Return Value

0 (NoError)	On success
-12 (SegmentError)	seg is invalid
Otherwise	On failure

4.1.4 FlashReadB

Description

Reads data from the FLASH memory where the size is that of the "unsigned char".
A far pointer can also be used to access the data on the Flash memory directly. The code is:

```
char far *ByteData,databyte;

ByteData=(char far *)MK_FP(segment, offset);
databyte=*ByteData;
```

Prototype

unsigned char FlashReadB(unsigned int seg, unsigned int offset);

Arguments

seg	0 to 65535 (0xffff).
offset	0 to 65535 (0xffff).

Return Value

FlashReadB only returns the value in the seg:offset address.

4.1.5 FlashReadI

Description

Reads data from the FLASH memory where the size is that of the "unsigned int".
A far pointer can be used to access the data on the Flash memory directly. The code is:

```
int far *IntData,dataint;

IntData=(int far *)MK_FP(segment, offset);
dataint=*IntData;
```

Prototype

unsigned FlashReadI(unsigned int seg, unsigned int offset);

Arguments

seg	0 to 65535(0xffff).
offset	0 to 65535(0xffff).

Return Value

FlashReadI only returns the value in the seg:offset address.

4.1.6 FlashReadL

Description

Reads data from the FLASH memory where the size is that of the "unsigned long".
A far pointer can also be used to access the data on the Flash memory directly. The code is:

```
long far *LongData,datalong;

LongData=(long far *)MK_FP(segment, offset);
datalong=*LongData;
```

Prototype

unsigned long FlashReadL(unsigned int seg, unsigned int offset);

Arguments

seg	0 to 65535 (0xffff).
offset	0 to 65535 (0xffff).

Return Value

FlashReadL only returns the value in the seg:offset address.

4.1.7 _MK_FP_

Description

Retrieves a far pointer.
A far pointer can be used to access the data on the Flash memory directly.
_MK_FP_ makes a far pointer from its componet segment and offset parts.
This function is particularly for the MSC compiler because MK_FP function has some problems in MSC.
The code is:

```
char far *ByteData,databyte;
int far *IntData,dataint;
long far *LongData,datalong;

ByteData=(char far *)_MK_FP_(segment, offset);
databyte=*ByteData;
IntData=(int far *)_MK_FP_(segment, offset);
dataint=*IntData;
LongData=(long far *)_MK_FP_(segment, offset);
datalong=*LongData;
```

Prototype

```
int _MK_FP_(unsigned int seg, unsigned int offset);
```

Arguments

seg	0 to 65535 (0xffff).
offset	0 to 65535 (0xffff).

Return Value

A far pointer	On success

5. NVRAM and RTC Functions

About the RTC and NVRAM hardware
1. When the hardware is equipped with a RTC (Real Time Clock), 31 bytes of NVRAM can be used to store data. 2. Only when user's program needs to record/know current time/date, it is needed to use RTC. For example, if it is wanted to turn on the light at 8:00 AM and turn off it at 5:00 PM. 3. NVRAM is SRAM, but it usees battary to keep the data, so the data in NVRAM does not lost its information when the module is power off. And NVRAM has no limit on the number of the re-write times. (FLASH and EEPROM both have the limit on re-write times.) If the leakage current is not happened, the battary can be used 10 years. 4. The following functions work if the module is equipped with RTC. (The standard version of I-7188XC/I-7521/I-7522/I-7523/I-7188EN/ parts of I-8000 series without RTC.)

5.1 Common Functions for all I-7188/I-8000 Series Modules

5.1.1 ReadNVRAM

Description

This function reads data from the NVRAM.

Prototype

int ReadNVRAM(int addr);

Arguments

addr	0 to 30, a total of 31 bytes.

Return Values

0-255	On success
-9 (AddrError)	The addr is invalid

5.1.2 WriteNVRAM

Description

Write data to NVRAM.

Prototype

int WriteNVRAM(int addr, int data);

Arguments

addr	0 to 30
data	One byte data (0-255). If data>255, only the low byte is written to NVRAM.

Return Values

0 (NoError)	On success
-9 (AddrError)	On fail

5.1.3 SetTimeDay

Description

Set the system time and date to the RTC of the module.
When calling SetTimeDate(), it just needs to set the right year, month, day and then the function auto set the weekday.

```
typedef struct {
int year;
char month,day,weekday;
char hour,minute,sec;
}TIME_DATE;
```

Prototype

```
int SetTimeDate(TIME_DATE *timedate);
```

Arguments

timedate	struct variable. timedate->year: (2000-2080). timedate->month:1~12. timedate->day: 1~31. timedate->hour: 0~23. timedate->minute: 0~59. timedate->sec: 0~59.

Return Values

0 (NoError)	On success
-18 (DateError)	Either year, month or day value is invalid
-19 (TimeError)	Either hour, minute or sec value is invalid

5.1.4 GetTimeDate

Description

Get the system time and date to the RTC of the module.
typedef struct {
int year;
char month,day,weekday;
char hour,minute,sec;
}TIME_DATE;

Prototype

void GetTimeDate(TIME_DATE *timedate);

Arguments

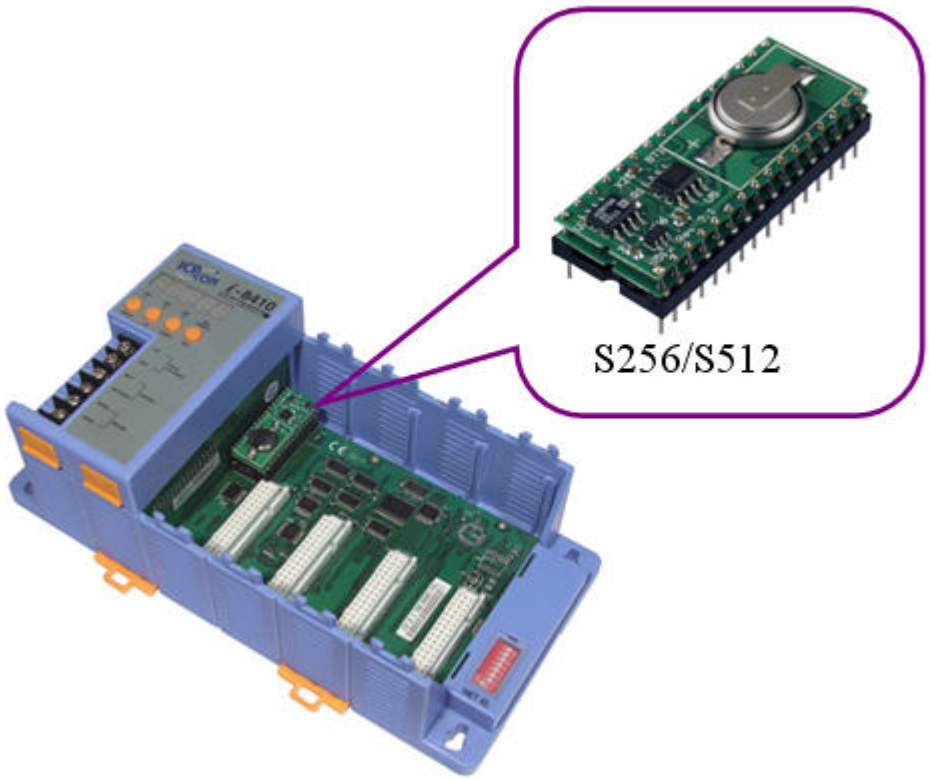
timedate	struct variable. timedate->year: (2000-2080). timedate->month:1~12. timedate->day: 1~31. timedate->weekday: 0~6. 0 is Sunday. 6 is Saturday. timedate->hour: 0~23. timedate->minute: 0~59. timedate->sec: 0~59.

Return Value

None.

6. SRAM Functions

Remark
1. These functions are for S256/S512 modules. 2. For S256: 2048 blocks, every block has 128 bytes. Total size is 256K bytes. 3. For S512: 4096 blocks, every block has 128 bytes. Total size is 512K bytes.



6.1 Common Functions for all I-8000 Series

6.1.1 S256_Init

Description

Do an initializing operation for functions relative to S256/S512.
It must call this function before other S256/S512 functions are used.

Prototype

int S256_Init(void);

Arguments

None.

Return Values

512	Ram size is 512K (S512).
256	Ram size is 256K (S256).
0	No S256 or S512 module is found.

6.1.2 S256_Read

Description

Read data from SRAM.

Prototype

int S256_Read(unsigned block,unsigned offset);

Arguments

block	0-2047 for S256 module, 0-4095 for S512 module.
offset	0-127.

Return Values

0-255	Data sotred in SRAM.
-10 (BlockError)	block is out of range
-100 (OffsetError)	offset is out of range.

6.1.3 S256_Write

Description

Write data to SRAM

Prototype

int S256_Write(unsigned block,unsigned offset,unsigned char data);

Arguments

block	0-2047 for S256 module, 0-4095 for S512 module.
offset	0-127.
data	0-255, the data to send out.

Return Values

0 (NoError)	On success
-10 (BlockError)	block is out of range
-100 (OffsetError)	offset is out of range.

6.1.4 S256_ReadF

Description

Read data from SRAM by linear address.

Prototype

unsigned char S256_ReadF(unsigned long address);

Arguments

address	0-0x3FFFF for S256 0-0x7FFFF for S512

Return Values

0-255	Data sotred in SRAM.

6.1.5 S256_ReadFn

Description

Read n bytes data from SRAM by linear address.

Prototype

int S256_ReadFn(unsigned long address,unsigned num,unsigned char *buf);

Arguments

address	0-0x3FFFF for S256 0-0x7FFFF for S512
num	The byte number to read.
buf	The buffer address to store data.

Return Values

0 (NoError)	Always return 0.

6.1.6 S256_WriteF

Description

Write data to SRAM by linear address.

Prototype

int S256_WriteF(unsigned long address,unsigned char data);

Arguments

address	0-0x3FFFF for S256 0-0x7FFFF for S512
data	0-255, the data to be send out.

Return Values

0 (NoError)	Always return 0.

6.1.7 S256_WriteFn

Description

Write n bytes data to SRAM by linear address.

Prototype

int S256_WriteFn(unsigned long address,unsigned num,unsigned char *data);

Arguments

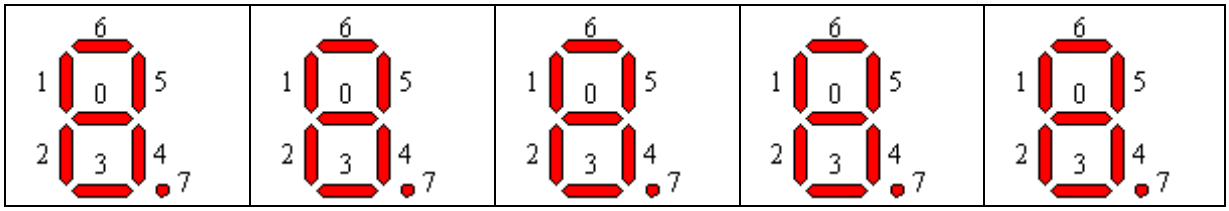
address	0-0x3FFFF for S256 0-0x7FFFF for S512
num	The byte number to read.
data	The buffer address to store data.

Return Values

0 (NoError)	Always return 0.

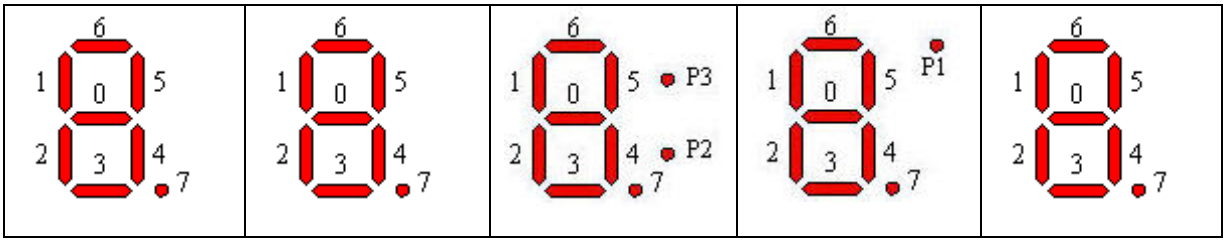
7. 5DigitLed Functions

[I-7188 series]



5DigitLed number is the leftmost is position 1 and the rightmost is position 5

[I-8000 series]



5DigitLed number is the leftmost is position 1 and the rightmost is position 5

1. 5DigitLed for I-7188 series is optional, but for I-8000 series is built-in. It is five 7-seg LED with the point on the left-bottom side. The default message display device for these modules. User can use it to show the message number, address, time, ... and so on.

2. The Red Led lights after the module is power on. User's program also can control it.

7.1 Common Functions for all I-7188/I-8000 Series

7.1.1 Init5DigitLed

Description

Initial 5 digits of the seven-segment LED. All segments are turned off.
The Init5DigitLed() has to be called before the other functions about 5 digits of seven-segment LED are called.

Prototype

void Init5DigitLed(void);

Arguments

None.

Return Value

None.

7.1.2 Disable5DigitLed

Description

Disable the scan circuit for 5 digits of seven-segment LED. All digits are turned off.

Prototype

void Disable5DigitLed(void);

Arguments

None.

Return Value

None.

7.1.3 Enable5DigitLed

Description

Enable the scan circuit for 5 digits of seven-segment LED.
An example of code:
Init5DigitLed();
Show5DigitLed(1,1);
Show5DigitLed(2,2);
Show5DigitLed(3,3);
Show5DigitLed(4,4);
Show5DigitLed(5,5);
Delay(2000); // the 5 digits of seven-segment LED is like "fig-1"
Disable5DigitLed();
Delay(2000); // the 5 digits of seven-segment LED is like "fig-2"
Enable5DigitLed(); // the 5 digits of seven-segment LED is like "fig-3"

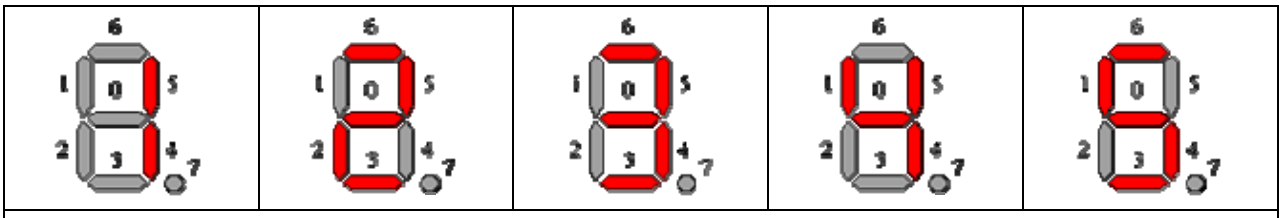


fig-1

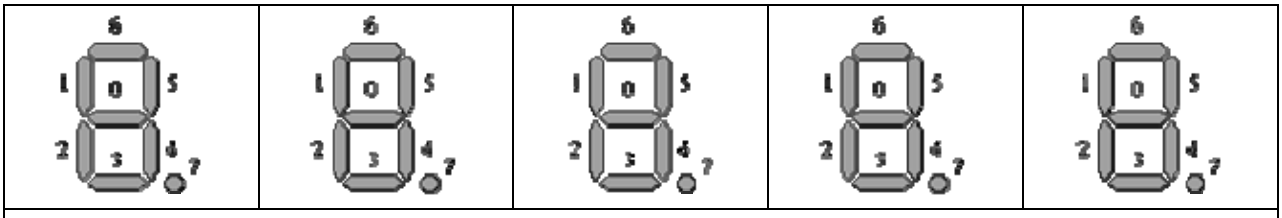


fig-2

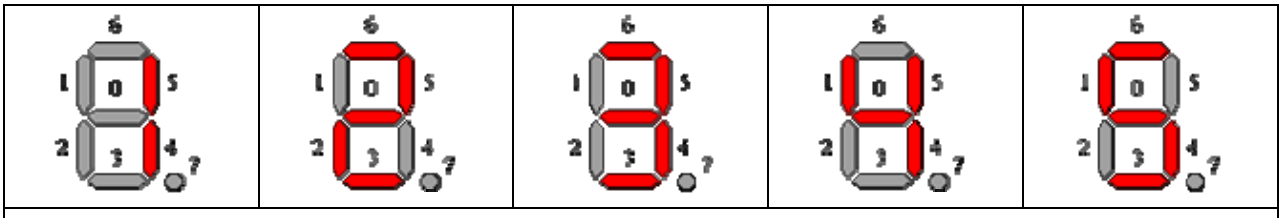


fig-3

Prototype

void Enable5DigitLed(void);

Arguments

None.

Return Value

None.

7.1.4 Show5DigitLed

Description

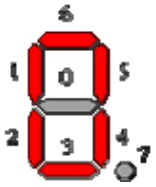
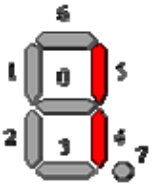
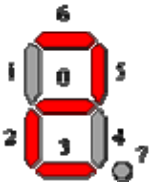
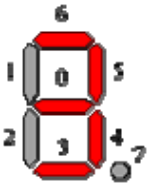
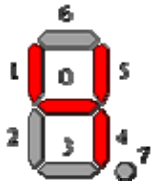
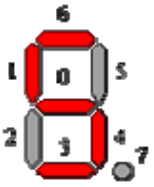
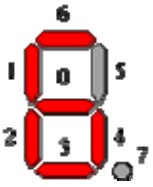
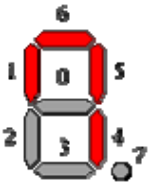
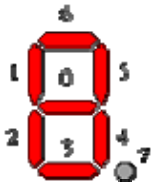
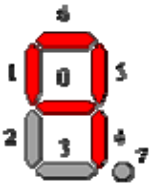
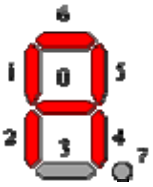
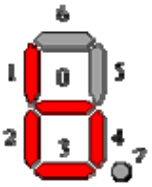
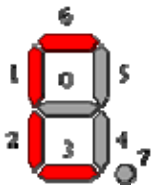
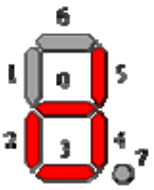
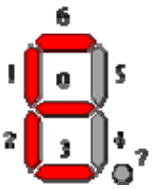
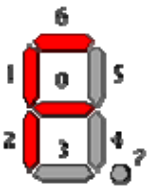
Show 5 digits of the seven-segment LED.

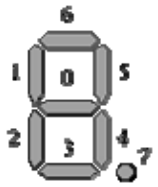
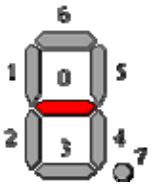
Prototype

void pascal Show5DigitLed(int pos,int data);

Arguments

pos	pos can be 1,2,3,4 or 5. 1 for the leftmost position. 5 for the rightmost position.
data	Range from 0 to 17. 0 to 15 show the hexadecimal Digit('0','1',....,'A','b','C','d','E','F') 16 show blank, 17 show '-'.

0	1	2	3
			
4	5	6	7
			
8	9	10(0x0A)	11(0x0B)
			
12(0x0C)	13(0x0D)	14(0x0E)	15(0x0F)
			
16(space)	17(dash)		

Return Value

None

7.1.5 Show5DigitLedWithDot

Description

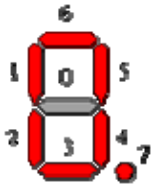
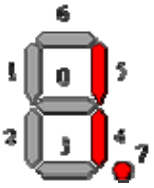
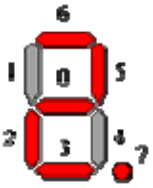
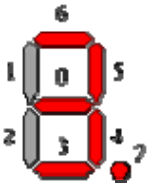
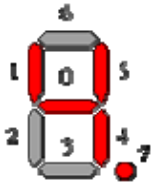
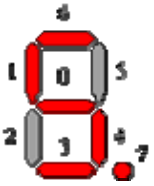
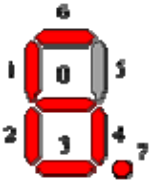
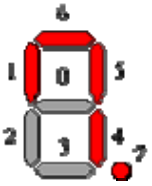
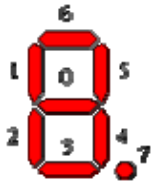
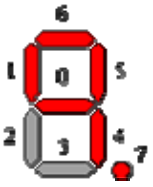
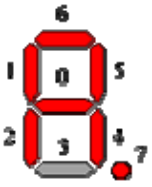
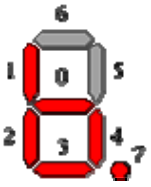
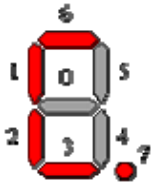
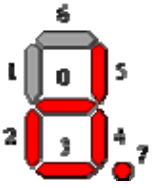
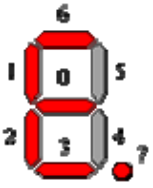
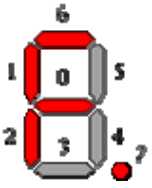
Show 5 digits of the seven-segment LED and also show the DOT '.'.

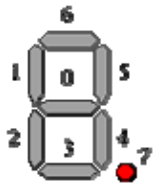
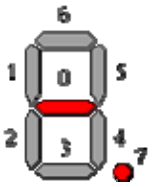
Prototype

void pascal Show5DigitLedWithDot(int pos,int data);

Arguments

pos	pos can be 1,2,3,4 or 5. 1 for the leftmost position. 5 for the rightmost position.
data	range from 0 to 17. 0 to 15 show the hexadecimal Digit with dot ('0.','1.',....,'A.','b.','C.', 'd.','E.','F.') 16 show blank, 17 show '-'.

0	1	2	3
			
4	5	6	7
			
8	9	10(0x0A)	11(0x0B)
			
12(0x0C)	13(0x0D)	14(0x0E)	15(0x0F)
			
16(space)	17(dash)		

Return Value

None

7.1.6 Show5DigitLedSeg

Description

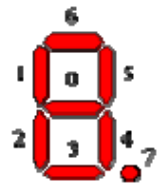
Show any segment of 5 digits of the seven-segment LED and also can show the DOT '.'.

Prototype

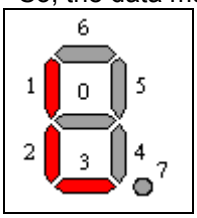
void pascal Show5DigitLedSeg(int pos,unsigned char data);

Arguments

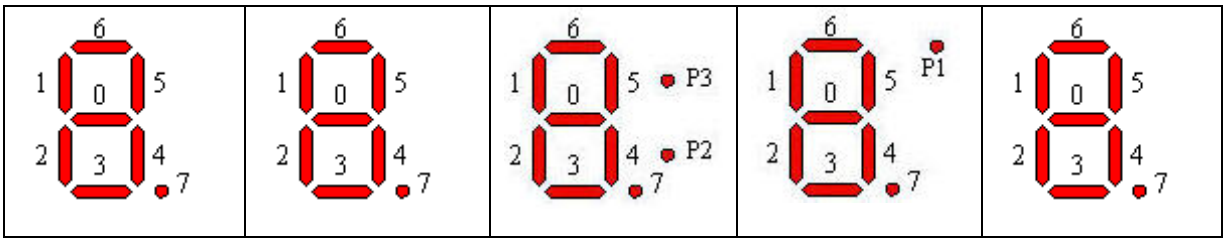
pos	pos can be 1,2,3,4,5 or 6. 1 for the leftmost position. 5 for the rightmost position. 6 for P1, P2 and P3 for I-8000 series.
data	Range from 0 to 255. When pos is 6, Bit 5 controls P1, Bit 6 controls P2 and Bit 7 controls P3.

	Bit 0 to 7 control the 8 segments (7-segment and dot), as the left figure. If the bit is 1, the segment turns on, If the bit is 0, the segment turns off.

For example, if user wants to display an 'L', must light segment 1, 2, 3.
So, the data must be 0x0E (=00001110b).



For I-8000 series:



Return Value

None.

7.1.7 Set5DigitLedTestMode

Description

Set 5 digits of seven-segment LED on Normal MODE or Test MODE.

Prototype

void pascal Set5DigitLedTestMode(int mode);

Arguments

mode	0 for normal mode. 1 for the test mode. On test mode all segment turn on, and use the largest intensity.

Return Value

None.

7.1.8 Set5DigitLedIntensity

Description

Set the scan intensity of LED segments, the default value=7(For I-8000 the default value is 3.).
If user feels that intensity is too dark, it can be set to higher value. (Higher value uses more power.)

Prototype

void pascal Set5DigitLedIntensity(int mode);

Arguments

mode	0 to 15. 0 represents the lowest intensity. 15 represents the highest intensity.

Return Value

None.

8. LED Functions

8.1 Common Functions For all I-7188/I-8000 series

8.1.1 LedOn

Description

Turn on the red LED.

Prototype

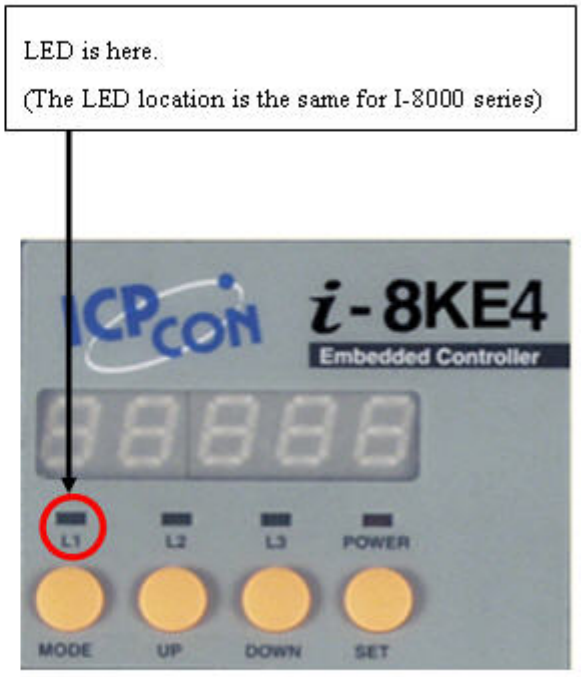
void LedOn(void);

Arguments

None.

Return Value

None.



8.1.2 LedOff

Description

Turn off the red LED.

Prototype

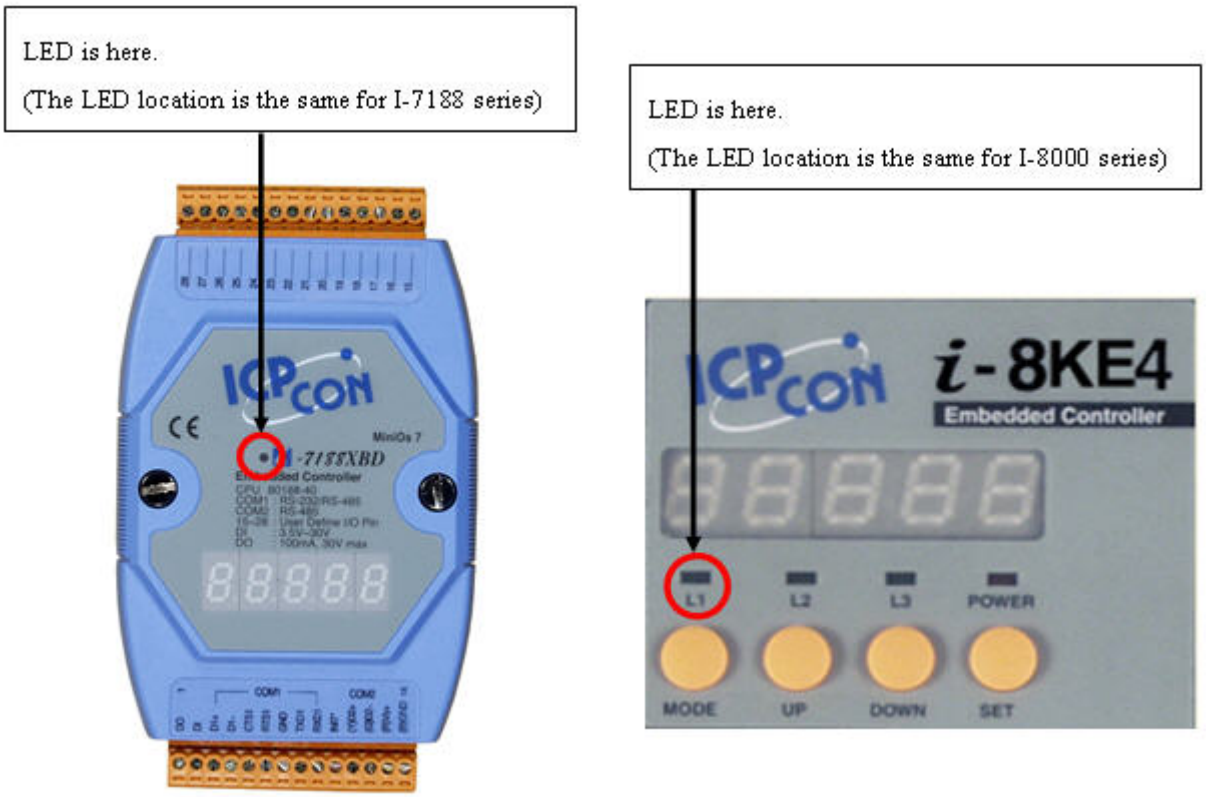
void LedOff(void);

Arguments

None.

Return Value

None.



8.1.3 LedToggle

Description

Toggle the status of the red LED.
If the red LED turns off originally, the red LED turns on after LedToggle() is called.
If the red LED turns on originally, the red LED turns off after LedToggle() is called.

Prototype

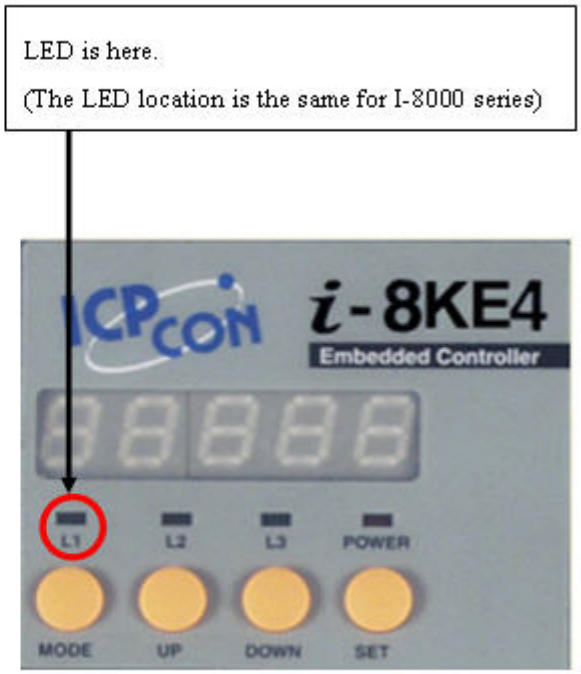
void LedToggle(void);

Arguments

None.

Return Value

None.



8.2 Respective Functions for all I-7188/I-8000 Series

8.2.1 Respective Functions Table

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
ShowErrLedCom8000	—	—	—	—	—	—	—	—	**Y**
SetLedL1 SetLedL2 SetLedL3	—	—	—	—	—	—	—	—	**Y**
Y: The module can use this function.									

8.2.2 ShowErrLedCom8000

Description

This function only for I-8142/ I-8142I/ I-8144/ I-8144I/ I-8112/ I-8114 modules.
This function displays the Error value on the LED (Er1~Er4) for the specified slot.
bit vallue: 0: LED ON

1: LED OFF

bit 0 (0x01) for COM port 1

bit 1 (0x02) for COM port 2

bit 2 (0x04) for COM port 3 (for 8144/8114 only)

bit 3 (0x08) for COM port 4 (for 8144/8114 only)

Prototype

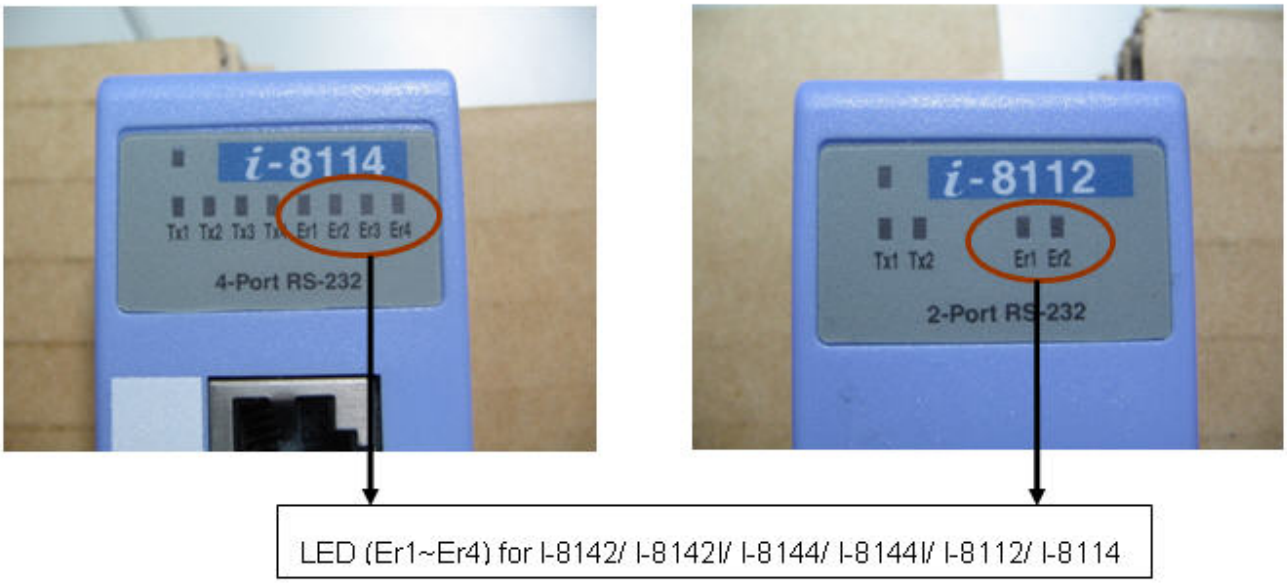
void ShowErrLedCom8000(int slot,int data);

Arguments

slot	0/1/2/3
data	The valid range is from 0 to 15.

Return Value

None.



8.2.3 SetLedLx

Description

Set the display mode of LED (L1/L2/L3) on I-8000 series.

Prototype

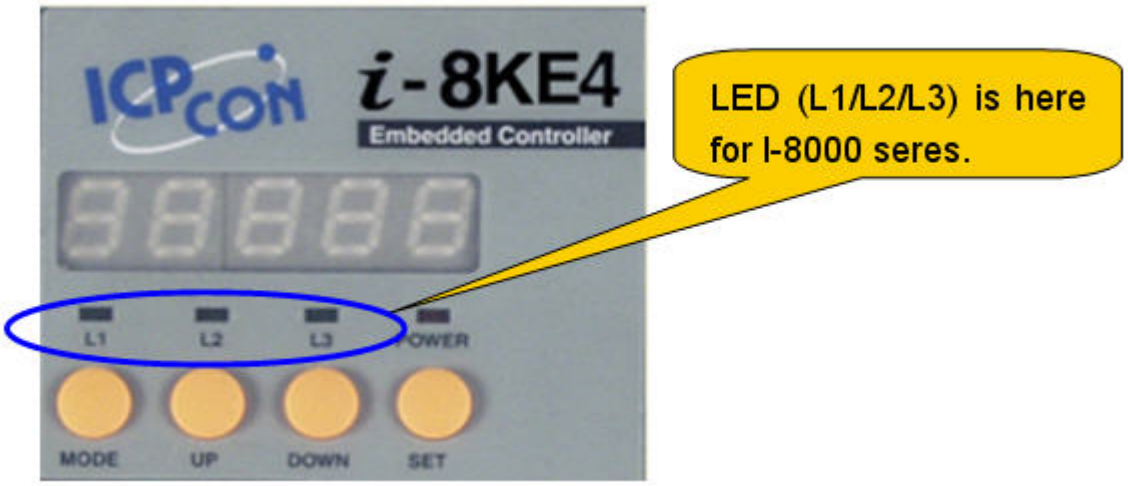
void SetLedL1(int mode);
void SetLedL2(int mode);
void SetLedL3(int mode);

Arguments

mode	0 (LED_OFF): Dark LED. 1 (LED_ON): Light LED. 2 (LED_TOGGLE): Toggle LED status.

Return Value

None.



9. Timer and WatchDogTimer Functions/Variables

Remark
1. MiniOS7 uses the TIMER 2 (one of CPU internal timer) as system Timer. It is 16-bits timer, and be programmed to generate interrupt every 1 ms. So the accuracy of system is 1 ms. P.S. I-8000 system's timer is programmed to generate interrupt every 5 ms. 2. User can install user's timer function, it is called every 1 ms. It is suitable to check system status every fixed time period. 3. The system timer ISR (Interrupt Service Routine) calls INT 9 every time, and every 55 ms calls INT 0x1C. The timer function of the library is installed on INT 9, so it is called every 1ms. P.S. The timer functions of I-8000's library use timer 1,it is programmed to generate interrupt every 1 ms. 4. Timer function supports 8 StopWatchs and 8 CountDownTimers and 1 MainTimer.

9.1 Common Functions/Varlables for all I-7188/I-8000 Series

9.1.1 WatchDogTimer Functions

9.1.1.1 EnableWDT

Description

The WatchDog Timer (WDT) is always enabled, and the system Timer ISR(Interrupt Service Routine) refresh it. When user's program calls EnableWDT(), the system timer ISR stops to refresh WDT, and user's program must do it by call RefreshWDT(). Otherwise, the system is reset by WDT. The timeout period of WDT is 0.8 seconds.

Prototype

void EnableWDT(void);

Arguments

None.

Return Value

None.

9.1.1.2 DisableWDT

Description

After DisableWDT is called, the system timer ISR (Interrupt Service Routine) refreshes WDT, so user's program need not to call RefreshWDT.

Prototype

void DisableWDT(void);

Arguments

None.

Return Value

None.

9.1.1.3 RefreshWDT

Description

After EnableWDT() is called, the system Timer ISR (Interrupt Service Routine) does not refresh WDT, so user's program must call RefreshWDT() to refresh WDT.

Prototype

void RefreshWDT(void);

Arguments

None.

Return Value

None.

9.1.2 CountDownTimer and StopWatch Functions

There are three kinds of countdowntimer and stopwatch functions.

Compare Items	First Kind Functions	Second Kind Functions	Third Kind Functions
Channel number	Max: 8	No channel number limitation	No channel number limitation
Read *TimeTicks value	No	Read it every time when these functions are used. Note: If these functions are used frequently, it takes much time to read *TimeTicks value regularly.	Read it only when T2_UpdateCurrentTimeTicks is called. Note: The error about stopwatch or countdown depends on how long the time is between two consecutive T2_UpdateCurrentTimeTicks calls in a loop.
TimerOpen, TimerClose	Before other functions are used, TimerOpen must be called. Finally, TimerClose also has to be called to stop using timer functions.	TimerOpen and TimerClose need not to be called.	TimerOpen and TimerClose need not to be called.

9.1.2.1 First Kind Functions

9.1.2.1.1 TimerOpen

Description

Start to use timer function. Before using Sec.9.1.2.1 timer function, InstallUserTimer and InstallUserTimer1C, it must call TimerOpen. After TimerOpen is called, user can use 8 StopWatchTimer and 8 CountDownTimer and the system timer.

Prototype

int TimerOpen(void);

Arguments

None.

Return Value

0 (NoError)	On success
1	Timer is already opend

See also

TimerClose

9.1.2.1.2 TimerClose

Description

Stop to use timer function. If the program has called TimerOpen, it must call TimerClose before exiting. Otherwise, the system must malfunction.

Prototype

int TimerClose(void);

Arguments

None.

Return Value:

Always return NoError.

9.1.2.1.3 CountDownTimerStart

Description

Start to use CountDownTimer. (Refer to TimerOpen/TimerClose)

Prototype

int CountDownTimerStart (int channel,unsigned long count);

Arguments

channel	0-7, total 8 channels.
count	The countdown time.

Return Values

0 (NoError)	On success
-15(ChannelError)	channel is out of range

9.1.2.1.4 CountDownTimerReadValue

Description

Read the current value of CountDownTimer(count). (Refer to TimerOpen/TimerClose)
When the return value is 0, time is up.

Prototype

int CountDownTimerReadValue(int channel,unsigned long *value);

Arguments

channel	0-7, total 8 channels.
value	a pointer for the value to be stored

Return Values

Current timer value	On success
-15(ChannelError)	channel is out of range

9.1.2.1.5 StopWatchReset

Description

Reset the StopWatch value to 0. (Refer to TimerOpen/TimerClose)

Prototype

int StopWatchReset(int channel);

Arguments

channel	0-7, total 8 channels.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.6 StopWatchStart

Description

Start to use a StopWatch channel, and reset the StopWatch value to 0. (Refer to TimerOpen/TimerClose)
Then, the system timer ISR (Interrupt Service Routine) increases the StopWatch value by 1 every 1 ms.

Prototype

int StopWatchStart(int channel);

Arguments

channel	0-7, total 8 channels.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.7 StopWatchStop

Description

Disable the StopWatch channel. Then, the system timer ISR (Interrupt Service Routine) stops to increase the StopWatch value.
(Refer to TimerOpen/TimerClose)

Prototype

int StopWatchStop(int channel);

Arguments

channel	0-7, total 8 channels.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.8 StopWatchPause

Description

Pause the StopWatch. After calling StopWatchPause, it can call StopWatchContinue to continue count time. (Refer to TimerOpen/TimerClose)

Prototype

int StopWatchPause(int channel);

Arguments

channel	0-7, total 8 channels.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.9 StopWatchContinue

Description

Continue the StopWatch. (Refer to TimerOpen/TimerClose)

Prototype

int StopWatchContinue(int channel);

Arguments

channel	0-7, total 8 channels.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.10 StopWatchReadValue

Description

Read current StopWatch value, the value stands for the time from StopWatchStart/StopWatchReset is called to now. (Refer to TimerOpen/TimerClose)

Prototype

int StopWatchReadValue(int channel,unsigned long *value);

Arguments

channel	0-7, total 8 channels.
value	A pointer for the value to be stored.

Return Value

0 (NoError)	On success
-15 (ChannelError)	channel is out of range

9.1.2.1.11 TimerResetValue

Description

Reset the main time ticks to 0. (Refer to TimerOpen/TimerClose)

Prototype

void TimerResetValue(void);

Arguments

None.

Return Value:

None.

9.1.2.1.12 TimerReadValue

Description

Read main time ticks. The time unit for ticks is 1 ms. Calling TimerOpen or TimerReset resets the valuse to 0. (Refer to TimerOpen/TimerClose)

Prototype

unsigned long TimerReadValue(void);

Arguments

None.

Return Value

The timer ticks since TimerOpen or TimerReset is called.

9.1.2.2 Second Kind Functions

9.1.2.2.1 T_StopWatchStart

Description

Start to use a stopwatch, record the current *TimeTicks value (sw->ulStart=*TimeTicks) and set sw->uMode=1. The sw->ulStart is the start value of stopwatch.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
void T_StopWatchStart(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable

Return Value

None.

9.1.2.2.2 T_ StopWatchGetTime

Description

Read current stopwatch value.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
ulong T_StopWatchGetTime(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable

Return Value

Non-negative	Current stopwatch value.

9.1.2.2.3 T_ StopWatchPause

Description

Pause the stopwatch. Record current *TimeTicks value to sw->ulPauseTime (sw->ulPauseTime=*TimeTicks) and set sw->uMode=0.
After calling T_StopWatchPause, it can call T_StopWatchContinue to continue count time.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

void T_StopWatchPause(STOPWATCH *sw);

Arguments

sw	Structure pointer variable.

Return Value

None.

9.1.2.2.4 T_ StopWatchContinue

Description

Continue the stopwatch.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
void T_StopWatchContinue(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable.

Return Value

None.

9.1.2.2.5 T_ CountDownTimerStart

Description

Start the countdowntimer. After T_CountDownTimerStart is called, it makes cdt->ulTime=timems cdt->ulStartTime= current *TimeTicks, cdt->uMode=1.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T_CountDownTimerStart(COUNTDOWNTIMER *cdt, ulong timems);
```

Arguments

cdt	Structure pointer variable.
timems	The countdown time. Unit is ms.

Return Value

None.

9.1.2.2.6 T_ CountDownTimerPause

Description

Pause the countdowntimer. It makes cdt->ulPauseTime= current *TimeTicks.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T_CountDownTimerPause(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

None.

9.1.2.2.7 T_ CountDownTimerContinue

Description

Continue the countdowntimer.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T_CountDownTimerContinue(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

None.

9.1.2.2.8 T_ CountDownTimerIsTimeUp

Description

Check the time of countdown is up or not.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
int T_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

0	Time isn't up.
1	Time is up.

9.1.2.2.9 T_ CountDownTimerGetTimeLeft

Description

Get remnant time of countdown.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
ulong T_CountDownTimerGetTimeLeft(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

0	Countdown is over.
Non-negative	Remnant time. Unit is ms.

9.1.2.3 Third Kind Functions

9.1.2.3.1 T2_UpdateCurrentTimeTicks

Description

Store current *TimeTicls value into an internal parameter called ***NormTimeTicks**.
The calculation method for T2_StopWatchxxxx and T2_CountDownTimerxxxx is based on ***NormTimeTicks.**
T2_UpdateCurrentTimeTicks must be called to get new *TimeTicls value every loop so that other functions (T2_StopWatchxxxxx/ T2_CountDownTimerxxxx) can update their staus.

Prototype

void T2_UpdateCurrentTimeTicks(void);

Arguments

None

Return Value

None.

9.1.2.3.2 T2_StopWatchStart

Description

Start to use a stopwatch, record the current ***NormTimeTicks** value (sw->ulStart=***NormTimeTicks**) (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**) and set sw->uMode=1.
The sw->ulStart is the start value of stopwatch.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
void T2_StopWatchStart(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable

Return Value

None.

9.1.2.3.3 T2_StopWatchGetTime

Description

Read current stopwatch value.
***NormTimeTicks** (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**) value affects current stopwatch value. If ***NormTimeTicks** value still not changing, T2_StopWatchGetTime reads the same value continuously.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
ulong T2_StopWatchGetTime(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable

Return Value

Non-negative	Current stopwatch value.

9.1.2.3.4 T2_StopWatchPause

Description

Pause the stopwatch. Record current *TimeTicks value to sw->ulPauseTime (sw->ulPauseTime=***NormTimeTicks** (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**)) and set sw->uMode=0. After calling T2_StopWatchPause, it can call T2_StopWatchContinue to continue count time.

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
void T2_StopWatchPause(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable.

Return Value

None.

9.1.2.3.5 T2_StopWatchContinue

Description

Continue the stopwatch. (Refer to Sec.9.1.2.3.1)

```
typedef struct {
 ulong ulStart,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} STOPWATCH;
```

Prototype

```
void T2_StopWatchContinue(STOPWATCH *sw);
```

Arguments

sw	Structure pointer variable.

Return Value

None.

9.1.2.3.6 T2_CountDownTimerStart

Description

Start the countdowntimer. After T2_CountDownTimerStart is called, it makes cdt->ulTime=timems cdt->ulStartTime= current ***NormTimeTicks** (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**), cdt->uMode=1.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint  uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T2_CountDownTimerStart(COUNTDOWNTIMER *cdt,ulong timems);
```

Arguments

cdt	Structure pointer variable.
timems	The countdown time. Unit is ms.

Return Value

None.

9.1.2.3.7 T2_CountDownTimerPause

Description

Pause the countdowntimer. It makes cdt->ulPauseTime= current ***NormTimeTicks** (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**).

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T2_CountDownTimerPause(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

None.

9.1.2.3.8 T2_CountDownTimerContinue

Description

Continue the countdowntimer. (Refer to Sec.9.1.2.3.1)

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
void T2_CountDownTimerContinue(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

None.

9.1.2.3.9 T2_CountDownTimerIsTimeUp

Description

Check the time of countdown is up or not. (Refer to Sec.9.1.2.3.1)

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
int T2_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

0	Time isn't up.
1	Time is up.

9.1.2.3.10 T2_CountDownTimerGetTimeLeft

Description

Get remnant time of countdown.
***NormTimeTicks** (Refer to Sec.9.1.2.3.1 about ***NormTimeTicks**) value affects remnant time of countdown. If ***NormTimeTicks** value still not changing, T2_CountDownTimerGetTimeLeft reads the same value continuously.

```
typedef struct {
 ulong ulTime,ulStartTime,ulPauseTime;
 uint uMode;  /* 0: pause, 1:run(start) */
} COUNTDOWNTIMER;
```

Prototype

```
ulong T2_CountDownTimerGetTimeLeft(COUNTDOWNTIMER *cdt);
```

Arguments

cdt	Structure pointer variable.

Return Value

0	Countdown is over.
Non-negative	Remnant time. Unit is ms.

9.1.3 User Timer Functions

A*: InstallUserTimerFunction_us

B*: InstallUserTimerFunction_ms

C*: StopUserTimerFun

	InstallUserTimer	InstallUserTimer1C	A*	B*
TimerOpen, TimerClose	Used together	Used together	No	No
C*	No	No	Used together	Used together
Timer Function called	Every 1ms	Every 55 ms	User sets the time	User sets the time

NOTE: InstallUserTimerFunction_us, InstallUserTimerFunction_ms, Delay, Delay_1, Delay_2, i8017H_AD_TimerINT and i8017H_AD_TimerINT_Scan have relations with timer0 or timer1. Some rules have to be obeyed below when these functions are used together.

The first type function: InstallUserTimerFunction_us, InstallUserTimerFunction_ms.
The second type function: Delay, Delay_1, Delay_2.
The third type function: i8017H_AD_TimerINT and i8017H_AD_TimerINT_Scan.

(There are three timers (timer0, timer1 and timer2) in CPU.)

- Rule1: The third type function uses timer0. Therefore, other type functions have to change timer to use timer1 to stop from clashing.
- Rule2: InstallUserTimerFunction_us, InstallUserTimerFunction_ms can't be used at the same time at present, even if they use different timer.
- Rule3: The first type function and the second type function must set different timer when they are used together.

9.1.3.1 InstallUserTimer

Description

Install user's timer function. User's timer function is called every 1 ms. (Refer to Sec.9.1.2.1.1 TimerOpen/ Sec.9.1.2.1.2 TimerClose)

Prototype

void InstallUserTimer(void (*fun)(void));

Arguments

fun	The user's function pointer. The function cannot use input argument, and cannot return value.

Return Value

None.

9.1.3.2 InstallUserTimer1C

Description

Install user's timer function on interrupt 0x1c. (Refer to Sec.9.1.2.1.1 TimerOpen/ Sec.9.1.2.1.2 TimerClose)
System timer calls int 0x1c every 55 ms.

Prototype

void InstallUserTimer1C(void (*fun)(void));

Arguments

fun	The user's function pointer. The function cannot use input argument, and cannot return value.

Return Value

None.

9.1.3.3 InstallUserTimerFunction_us

Description

Install user's timer function. Time unit is 0.1 us.
Default timer is timer 0.
For example:
If user wants timer generates interrupt every 0.5ms (500 us=5000*0.1us) (That is to say system calls your function once every 0.5 ms).
Just use

```
<<============================>>
void fun(void)
{
......
}
.......
InstallUserTimerFunction_us (5000, fun);
<<============================>>
```

Prototype

int InstallUserTimerFunction_us(unsigned time,void (*fun)(void));

Arguments

time	100~65535. Unit is 0.1 us.
*fun	A pointer that pointer to function.

Return Value

0(NoError)	On success
-1	time<100
-2	*fun can't be found

See also

StopUserTimrFun

9.1.3.4 InstallUserTimerFunction_ms

Description

Install user's timer function. Time unit is ms.
Default timer is timer 0 and timer 2. (timer 2's output is timer 0's input)
For example:
If user wants timer generates interrupt every 1 second (1 sec=1000ms) (That is to say system calls your function once every 1 sec).
Just use

```
<<==========================>>
void fun(void)
{
......
}
.......
InstallUserTimerFunction_ms (1000, fun);
<<==========================>>
```

Prototype

int InstallUserTimerFunction_ms(unsigned time,void (*fun)(void));

Arguments

time	0~65535. Unit is ms.
*fun	A pointer that pointer to function.

Return Value

0(NoError)	On success
-2	*fun can't be found

9.1.3.5 StopUserTimerFun

Description

Stop user's timer function.
After InstallUserTimerFunction_us or InstallUserTimerFunction_ms is used, StopUserTimerFun has to be used to stop user's timer function.

Prototype

void StopUserTimerFun(void);

Arguments

None

Return Value

None.

9.1.4 Others Functions

	DelayMs	Delay	Delay_1	Delay_2
Timer	By default, use timer2	By default, use timer0	By default, use timer0	By default, use timer0
Unit	ms	ms	0.1ms	0.01ms
Timer Change	No	Yes.(***note**)	Yes.(***note**)	Yes.(***note**)

***note**: SetDelayTimer in Sec.9.2.2

	TimeTicks	GetTimeTicks	GetTimeTicks_ISR
Timer Interrupt results in error	possible	No	No
Source code comparison	long st; st=*TimeTicks;	long st; _asm cli st=*TimeTicks; _asm sti	long st; _asm pushf _asm cli st=*TimeTicks; _asm popf

9.1.4.1 DelayMs

Description

Delay some time interval, the time unit is ms.
There are three timers in CPU. DelayMs uses the timer2 to do the delay operation.

Prototype

void DelayMs(unsigned t);

Arguments

t	the delay time. Unit is ms.

Return Value

None.

9.1.4.2 Delay

Description

Delay some time interval, the time unit is ms.
There are three timers in CPU. Delay uses the timer0 to do the delay operation.
If user wants to change the timer for Delay, please refer to SetDelayTimer.

When InstallUserTimerFunction_us/ InstallUserTimerFunction_ms and Delay are used together, user has to set different timer for them.
If not, they clash with each other. The SetDelayTimer and SetUserTimer can set timer0 or timer1 to be used.

Prototype

void Delay(unsigned ms);

Arguments

ms	the delay time. Unit is ms.

Return Value

None.

9.1.4.3 Delay_1

Description

Delay some time interval, the time unit is 0.1ms.
There are three timers in CPU. Delay_1 uses the timer0 to do the delay operation.
If user wants to change the timer for Delay_1, please refer to SetDelayTimer.

When InstallUserTimerFunction_us/ InstallUserTimerFunction_ms and Delay_1 are used together, user has to set different timer for them. If not, they clash with each other. The SetDelayTimer and SetUserTimer can set timer0 or timer1 to be used.

Prototype

void Delay_1(unsigned ms);

Arguments

ms	the delay time. Unit is 0.1ms.

Return Value

None.

9.1.4.4 Delay_2

Description

Delay some time interval, the time unit is 0.01ms.
There are three timers in CPU. Delay_2 uses the timer0 to do the delay operation.
If user wants to change the timer for Delay_2, please refer to SetDelayTimer.

When InstallUserTimerFunction_us/ InstallUserTimerFunction_ms and Delay_2 are used together, user has to set different timer for them. If not, they clash with each other. The SetDelayTimer and SetUserTimer can set timer0 or timer1 to be used.

Prototype

void Delay_2(unsigned ms);

Arguments

ms	the delay time. Unit is 0.01ms

Return Value

None.

9.1.4.5 TimeTicks

Description

There are three timers in CPU. The timer2 adds 1 to *TimeTicks per 1 ms. It can be used to count time.
The *TimerTicks' data type is DWORD (unsigned long). So the MOV instruction has to move data twice to complete the read operation in a 16 bit CPU.
If there is timer interrupt to happen during interval of movement twice, the final *TimeTicks value may be wrong.
The safe way is to disable interrupt when *TimeTicks is read.
<<==========>>
For example:
unsigned long TT;
_asm cli
TT=*TimeTicks;
_asm sti
<<==========>>
We also provide two functions, GetTimeTicks and GetTimeTicks_ISR, that both disable interrupt and return *TimeTicks value.

Prototype

const unsigned long far *TimeTicks;

9.1.4.6 GetTimeTicks

Description

The timer2 adds 1 to *TimeTicks per 1 ms. It can be used to count time. GetTimeTicks disables timer interrupt and return *TimeTicks value.
For example:

Original	Correspondence
long st; st=GetTimeTicks();	long st; _asm cli st=*TimeTicks _asm sti

According to above table, interrupt is enabled after GetTimeTicks is called.

Prototype

long GetTimeTicks(void);

Arguments

None.

Return Value

*TimeTicks value	On success

9.1.4.7 GetTimeTicks_ISR

Description

The timer2 adds 1 to *TimeTicks per 1 ms. It can be used to count time. GetTimeTicks_ISR disables timer interrupt and return *TimeTicks value.
For example:

Original	Correspondence
long st; st=GetTimeTicks_ISR();	long st; _asm pushf _asm cli st=*TimeTicks _asm popf

According to above table, interrupt flag does not be changed after GetTimeTicks_ISR is called.

Prototype

long GetTimeTicks_ISR(void)

Arguments

None.

Return Value

*TimeTicks value	On success

9.2 Respective Functions for all I-7188/I-8000 Series

9.2.1 Respective Functions Table

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
SetDelayTimer	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
SetUserTimer	—	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Y: The module can use this function.									

9.2.2 SetDelayTimer

Description

The default timer of delay functions(Dealy()/Delay_1()/Delay_2()) use system's timer 0.
User can call SetDelayTimer(1) to set them to use timer 1.

Prototype

int SetDelayTimer(int num);

Arguments

num	num=0 use timer 0
	num=1 use timer 1

Return Value

0	Use timer 0
1	Use timer 1

9.2.3 SetUserTimer

Description

The default timer of user timer functions (InstallUserTimerFunction_us()/InstallUserTimerFunction_ms()) use system's timer 0. User can call SetUserTimer(1) to set them to use timer 1.

Prototype

int SetUserTimer(int no);

Arguments

num	num=0 use timer 0
	num=1 use timer 1

Return Value

0	Use timer 0
1	Use timer 1

10. Files Functions

Remark
The file system for MiniOS7 support user's program to read files, but not support user's program to write files.

10.1 Disk A and Disk B

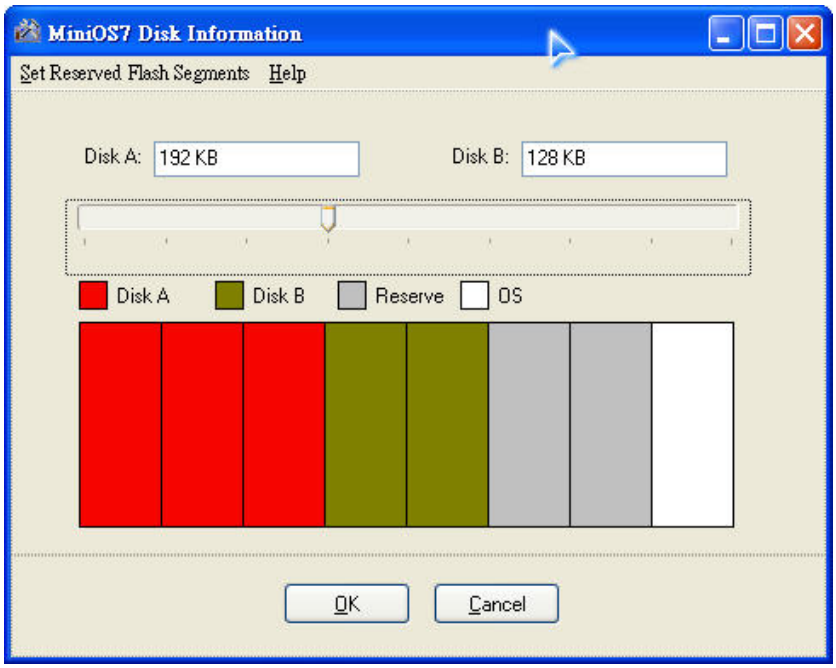
Flash file system space:

1. For 256K type, segment 0xC000, 0xD000, 0xE000 can be used. (***Three*** segments can be used)
2. For 512K type, segment 0x8000, 0x9000, 0xA000, 0xB000, 0xC000, 0xD000, 0xE000 can be used. (**Seven** segments can be used.)
3. The segment 0xF000 is used for MiniOs7.

How to divide file system space (into diskA and diskB)?

MiniOS7 Utility can divide file system space (***Three*** or ***Seven*** segments) into three parts which are diskA, diskB and reserved.

Refer to: http://ftp.icpdas.com.tw/pub/cd/8000cd/napdos/minios7/utility/minios7_utility/



10.2 Common Functions for all I-7188/I-8000 Series

10.2.1 GetFileNo_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Get total number of file stored in Flash Memory.

Prototype

int GetFileNo_AB(int disk);

Arguments

disk	0 is diskA.
	1 is diskB.

Return Value

The file number.

10.2.2 GetFileName_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Use file index to get file name.

Prototype

int GetFileName_AB(int disk,int num,char *fname);

Arguments

disk	0 is diskA.
	1 is diskB.
num	The file index (The first file is index 0).
fname	file name.

Return Value

0 (NoError)	Store the file name to fname
-1	Do not save any data to fname.

10.2.3 GetFileInfoByNo_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Use file index to get file information.

```
typedef struct {
unsigned mark; /* 0x7188 -> is file */
unsigned char fname[12];
unsigned char year;
unsigned char month;
unsigned char day;
unsigned char hour;
unsigned char minute;
unsigned char sec;
unsigned long size;
char far *addr;
unsigned CRC;
unsigned CRC32;
} FILE_DATA;
```

Prototype

FILE_DATA far * GetFileInfoByNo_AB(int disk,int no);

Arguments

disk	0 is diskA.
	1 is diskB.
no	The file index (The first file is index 0).

Return Value

Structure pointer	Point to the start address of the file information.
NULL	On fail.

10.2.4 GetFileInfoByName_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Use file name to get file information.

```
typedef struct {
unsigned mark; /* 0x7188 -> is file */
unsigned char fname[12];
unsigned char year;
unsigned char month;
unsigned char day;
unsigned char hour;
unsigned char minute;
unsigned char sec;
unsigned long size;
char far *addr;
unsigned CRC;
unsigned CRC32;
} FILE_DATA;
```

Prototype

FILE_DATA far *GetFileInfoByName_AB(int disk,char *fname);

Arguments

disk	0 is diskA.
	1 is diskB.
fname	file name.

Return Value

Structure pointer	Point to the start address of the file information.
NULL	On fail.

10.2.5 GetFilePositionByNo_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Use file index to get the start position of file on Flash Memory. User can use the address to get file data.
(If file size > 64K-16, it must use a huge pointer(char huge *) to get file data for offset >64K-16)

Prototype

char far * GetFilePositionByNo_AB(int disk,int num);

Arguments

disk	0 is diskA.
	1 is diskB.
num	The file index (The first file is index 0).

Return Value

Start address of the file.	On success.
NULL	On fail.

10.2.6 GetFilePositionByName_AB

Description

#define DISKA 0
#define DISKB 1
(About the diskA or diskB, refer to Sec.10.1)
Use file name to get file start position on Flash Memory.

Prototype

char far * GetFilePositionByName_AB(int disk,char *fname);

Arguments

disk	0 is diskA.
	1 is diskB.
fname	file name.

Return Value

Start address of the file.	On success.
NULL	On fail.

10.3 Respective Functions/Variables for all I-7188/I-8000 Series

10.3.1 Respective Functions/Variables Table

Variables	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
SizeAB	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
DiskAStartSeg	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
DiskBStartSeg	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_FileDateTimeMode	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
Y: The module can use these variables.									

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
AddFarPtrLong	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
ReadSizeAB	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_DeleteAllFile	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_GetDiskFreeSize	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_OpenWriteFile	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_WriteFile	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
OS7_CloseWriteFile	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_Push	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_MakeTable	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_Pop	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_Set	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_Read	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_AddData	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
CRC16_AddDataN	—	—	—	—	—	**Y**	**Y**	**Y**	**Y**
Modbus_CRC16_Push	—	—	—	—	—	**Y**	**Y**	**Y**	—
Modbus_CRC16_Pop	—	—	—	—	—	**Y**	**Y**	**Y**	—
Modbus_CRC16_Set	—	—	—	—	—	**Y**	**Y**	**Y**	—
Modbus_CRC16_Read	—	—	—	—	—	**Y**	**Y**	**Y**	—
Modbus_GetCRC16	—	—	—	—	—	**Y**	**Y**	**Y**	—
Y: The module can use this function.									

10.3.1.1 ReadSizeAB

Description

Read diskA and diskB information and then write data to SIZE_AB structure, DiskAStartSeg and DiskBStartSeg. (About the diskA or diskB, refer to Sec.10.1)

```
typedef struct {
  unsigned sizeA:3;
  unsigned sizeB:3;
  unsigned sizeC:3;
  unsigned sum:7;
} SIZE_AB;
```

Prototype

void ReadSizeAB(void);

Arguments

None

Return Value

None

10.3.1.2 SizeAB

Description

Read size for diskA, diskB and reserved(sizeC).
It must call ReadSizeAB to write data to SIZE_AB first and then SizeAB can read real value about size of disk.
(About the diskA or diskB, refer to Sec.10.1)

```
typedef struct {
  unsigned sizeA:3;
  unsigned sizeB:3;
  unsigned sizeC:3;
  unsigned sum:7;
} SIZE_AB;
```

Prototype

```
extern SIZE_AB SizeAB;
```

10.3.1.3 DiskAStartSeg_DiskBStartSeg

Description

Read start segment for diskA or diskB.
It must call ReadSizeAB to write data to DiskAStartSeg and DiskBStartSeg first and then DiskAStartSeg and DiskBStartSeg can read real value about start segement of disk
(About the diskA or diskB, refer to Sec.10.1)

Prototype

extern unsigned DiskAStartSeg;

extern unsigned DiskBStartSeg;

10.3.1.4 AddFarPtrLong

Description

Move a far pointer.

Prototype

void far *AddFarPtrLong(void far * ptr1,unsigned long size);

Arguments

ptr1	void far pointer.
size	Moving distance.

Return Value

void far*	A far pointer

10.3.1.5 OS7_DeleteAllFile

Description

Delete all file in the disk.

Prototype

int OS7_DeleteAllFile(int disk);

Arguments

disk	0 is diskA.
	Others are diskB.

Return Value

0	On success.
-1	Delete fail.
-2	No file in the disk.
-3	diskB size is zero.(Only diskA)

10.3.1.6 OS7_GetDiskFreeSize

Description

Get free size of the disk (free size=disk total size-(file head size+file data size)).

Prototype

long OS7_GetDiskFreeSize(int disk);

Arguments

disk	0 is diskA.
	Others are diskB.

Return Value

Non-negative value	Free size which can be written data.

10.3.1.7 OS7_OpenWriteFile

Description

Reserve file head size and write operation is initiated.
OS7_OpenWriteFile, OS7_WriteFile and OS7_CloseWriteFile have to be used together. If a write file operation is executed, the sequence about use of three functions is OS7_OpenWriteFile first, OS7_WriteFile and OS7_CloseWriteFile the last.

Prototype

int OS7_OpenWriteFile(int disk);

Arguments

disk	0 is diskA.
	Others are diskB.

Return Value

0	On success.
-1	No enough free size in the disk. So write operation can't continue.
-2	Pre-write operation has not yet finished.

10.3.1.8 OS7_WriteFile

Description

Write file data.
OS7_OpenWriteFile, OS7_WriteFile and OS7_CloseWriteFile have to be used together. If a write file operation is executed, the sequence about use of three functions is OS7_OpenWriteFile first, OS7_WriteFile and OS7_CloseWriteFile last.

Prototype

int OS7_WriteFile(int disk,void *buf,int size);

Arguments

disk	0 is diskA.
	Others are diskB.
buf	Points to a buffer that the function writes the bytes from.
size	Number of bytes the function attempts to write.

Return Value

Non-negative value	Number of bytes written. It means disk size is exhausted if this number is smaller than the argument “size”.
-2	The argument “disk” is not the same in the OS7_WriteFile and OS7_OpenWriteFile
-3	OS7_OpenWriteFile isn’t called before the function is used.

10.3.1.9 OS7_CloseWriteFile

Description

Write file header and finish write file operation.
OS7_OpenWriteFile, OS7_WriteFile and OS7_CloseWriteFile have to be used together. If a write file operation is executed, the sequence about use of three functions is OS7_OpenWriteFile first, OS7_WriteFile and OS7_CloseWriteFile last.

File Header:

```
typedef struct  {
  unsigned mark;   /* 0x7188 -> is file */
  unsigned char fname[12];
  unsigned char year;
  unsigned char month;
  unsigned char day;
  unsigned char hour;
  unsigned char minute;
  unsigned char sec;
  unsigned long size;
  char far *addr;
  unsigned CRC;
  unsigned CRC32;
} FILE_DATA;
```

File header item	Description
mark, size, addr, CRC, CRC32	Auto setting (user needn't care them)
fname	User defined
Year, month, day, hour, minute, sec	Three modes according to OS7_FileDateTimeMode value: (1) OS7_FileDateTimeMode=0 (default): User defined. (2) OS7_FileDateTimeMode=1: Auto write default value: [2006/01/01 12:00:00] (3) OS7_FileDateTimeMode=2: Auto write time value read from RTC (real time clock).

Prototype

int OS7_CloseWriteFile(int disk,FILE_DATA *f_data);

Arguments

disk	0 is diskA.
	Others are diskB.
f_data	Structure pointer variable. Points to file header address that the function writes file header data from.

Return Value

0	On success.
-2	The argument "disk" is not the same in the OS7_CloseWriteFile and OS7_OpenWriteFile.

-3	OS7_OpenWriteFile isn't called before the function is used.

10.3.1.10 OS7_FileDateTimeMode

Description

The time value setting of file header is affected by this variable.

OS7_FileDateTimeMode value	Description
0	User defines time values of file header.
1	Auto write default time value: [2006/01/01 12:00:00]
2	Auto write time value read from RTC (real time clock).

Prototype

extern int OS7_FileDateTimeMode;

10.3.2 CRC16 functions

CRC16 formula in MiniOS7 is: CRC-CCITT: 0x1021 = x16 + x12 + x5 + 1
Initial value of CRC16 is 0。

1. It takes much time about CRC16 calculation. If CRC16 functions are used in the ISR (Interrupt Service Register), it would be better not to count data too much to avoid affecting normal operation of system interrupt.

2. There are 16 layers in the stack for CRC16_Push/ CRC16_Pop. It means that maximum number of data stored in the stack is 16.

How to Use CRC16 functions

1. It must call CRC16_MakeTable first to build crc16 table that help CRC16_AddData/ CRC16_AddDataN to calculate right crc16 value.

2. Example1: simple usage of CRC16 functions.

```
CRC16_MakeTable();
//...
CRC16_Reset();
for(i;=;i<datano;i++){
  CRC16_AddData(data[i]);
}
CRC16=CRC16_Read();
```

3. Example2: It needs to calculate another CRC16 value under the process of CRC16 calculation.

```
int CheckCrc(char *data,int no,unsigned crc)
{  unsigned CRC16;
   //...
  CRC16_Push();
  CRC16_Reset();
  CRC16_AddDataN(data,datalen);
  CRC16=CRC16_Read();
  CRC16_Pop();
```

```
    return crc==CRC16;
  }

char data[256];
void fun0(void)
{  unsigned crc,CRC16;
   int datanum;
   int blocknum=0;
   int CRC_Error=0;

   CRC16_MakeTable();
   CRC16_Reset();
   while(!quit && blocknum<BLOCK_NO){
         datanum=GetData(data);
         crc=GetCrc();
         blocknum++;
         if(CheckCrc(data,datanum,crc)){
                //crc OK,
                CRC16_AddDataN(data,datanum);
         }
         else {
                // crc error
                quit=1;
                CRC_Error=1;
         }
    }
    if(!CRC_Error){
        CRC16=CRC16_Read();
        //...
     }
}
```

4. The process of CRC16 calculation has to divide many parts.

```
   crc16_n=0;
   //...
   //The following way to handle that adding data to calculate CRC16 value in process,
  CRC16_Push(); //store old crc16 value.(It maybe is the last counting result of CRC16 calculation in another
                //program)
```

```
CRC16_Set(crc16_n); //set initial value of CRC16 in CRC16 calculation of following program segment.
CRC16_AddDataN(data,data_len); //add data。
crc16_n=CRC16_Read(); //read current CRC16 value and store it to use next time.
 CRC16_Pop(); //restore old CRC16 value.
  //...
```

[Annotate]

MiniOS7 load is an example: ICPDAS modules based on MiniOS7 receive a file from PC whose unit is packet. They calculate CRC16 value (***CRC_1***) of each packet and compare CRC16 value recorded in the packet with ***CRC_1*** every time. File header stores CRC16 value of whole file which is calculated when modules receive packets every time. There are two CRC16 operations to calculate at the same time in this situation.

10.3.2.1 CRC16_MakeTable

Description

Get a buffer from system memory and build a table for CRC16.
This function must be called before CRC16_AddData, CRC16_AddDataN_C and CRC16_AddDataN are used,

Prototype

int CRC16_MakeTable(void);

Arguments

None

Return Value

0	On success.
-1	Retrieving memory fail.

10.3.2.2 CRC16_Push

Description

Push CRC16 value into stack. There are 16 layers in the stack.

Prototype

int CRC16_Push(void);

Arguments

None

Return Value

0	On success.
-1	Stack is overflow.

10.3.2.3 CRC16_Pop

Description

Pop CRC16 value off stack. There are 16 layers in the stack.

Prototype

int CRC16_Pop(void);

Arguments

None

Return Value

0	On success.
-1	Stack is underflow.

10.3.2.4 CRC16_Set

Description

Set current CRC16 value.

#define CRC16_Reset() CRC16_Set(0)

Prototype

void CRC16_Set(unsigned val);

Arguments

val	New CRC16 value.

Return Value

None

10.3.2.5 CRC16_Read

Description

Read current CRC16 value.

Prototype

unsigned CRC16_Read(void);

Arguments

None

Return Value

CRC16 value calculated by system.

10.3.2.6 CRC16_AddData

Description

Add one byte data to calculate new CRC16 value that can be read from CRC16_Read.

Prototype

void CRC16_AddData(unsigned char data);

Arguments

data	Add the data to calculate new CRC16 value.

Return Value

None.

10.3.2.7 CRC16_AddDataN

Description

Add n bytes data to calculate new CRC16 value that can be read from CRC16_Read.

Prototype

void CRC16_AddDataN(unsigned char far *data,unsigned length);

Arguments

data	Points to an address that this function want to add data from.
length	Number of data the function attempts to add.

Return Value

None.

10.3.2.8 Modbus_CRC16_Push

Description

Push CRC16 value into stack. There are 16 layers in the stack.

Prototype

int Modbus_CRC16_Push(void);

Arguments

None

Return Value

0	On success.
-1	Stack is overflow.

10.3.2.9 Modbus_CRC16_Pop

Description

Pop CRC16 value off stack. There are 16 layers in the stack.

Prototype

int Modbus_CRC16_Pop(void);

Arguments

None

Return Value

0	On success.
-1	Stack is underflow.

10.3.2.10 Modbus_CRC16_Set

Description

Set current CRC16 value.
It sets usually CRC16 value to 0 before CRC16 value of a set of data is calculated.
(It can also be called directly CRC16_Reset. If it wants to continue CRC16 operation which is not yet finished last time, it needs to set CRC16 value of last calculation last time.)

#define Modbus_CRC16_Reset() Modbus_CRC16_Set(0xFFFF)

Prototype

void Modbus_CRC16_Set(unsigned val);

Arguments

val	New CRC16 value.

Return Value

None

10.3.2.11 Modbus_CRC16_Read

Description

Read current CRC16 value.

Prototype

unsigned Modbus_CRC16_Read(void);

Arguments

None

Return Value

CRC16 value calculated by system.

10.3.2.12 Modbus_GetCRC16

Description

Add n bytes data to calculate new CRC16 value that can be read from Modbus_CRC16_Read.

Prototype

void Modbus_GetCRC16(unsigned char *puchMsg, int DataLen);

Arguments

puchMsg	Points to an address that this function want to add data from.
DataLen	Number of data the function attempts to add.

Return Value

None.

11. Connect to I-7000 Modules Functions

Remark
1. When user wants to use I-7188/I-8000 series to control I-7000 series module, they can use the following three functions. 2. After calling SendCmdTo7000, it must call ReceiveResponseFrom7000_ms or ReceiveResponseFrom7000_loop to receive the response of I-7000 module. (P.S. If the command "SendCmdTo7000" has no response, it need not call ReceiveResponseFrom7000_ms or ReceiveResponseFrom7000_loop.)

11.1 Common Functions for all I-7188/I-8000 Series

11.1.1 SendCmdTo7000

Description

Send command to 7000 series or I-87K. If checksum is enabled, the function adds 2 bytes as checksum on the end of command.

Prototype

int SendCmdTo7000(int iPort, unsigned char *cCmd, int iChksum);

Arguments

iPort	1 to 8 for I-7188x series. 0,1,3,4 for I-8000 series.
cCmd	Command to be send out (Don't need to add CR in back of *cCmd).
iChksum	1 is to enable checksum, 0 is to disable checksum.

Return Value

NoError	On success
Error code(refer to Sec.15,constant defined for I-7188/I-8000 series)	On fail.

11.1.2 ReceiveResponseFrom7000_loop

Description

After calling SendCmdTo7000, user can call ReceiveResponseFrom7000_loop to get the response from 7000 ro I-87K.

Prototype

int ReceiveResponseFrom7000_loop(int iPort, unsigned char *cCmd, long lTimeout, int iChksum);

Arguments

iPort	1 to 8 for I-7188x series. 0,1,3,4 for I-8000 series.
cCmd	Store response received from 7000 or I-87K. If checksum is enabled, the function checks and remove the checksum. The CR is removed.
lTimeout	Set the timeout, unit is times of check COM port. If lTimeout =500, the function checks if any data in input buffer five hundred times.
iChksum	1 is to enable checksum, 0 is to disable checksum.

Return Value

NoError	On success
Error code(refer to Sec.15,constant defined for I-7188/I-8000 series)	On fail.

11.1.3 ReceiveResponseFrom7000_ms

Description

After calling SendCmdTo7000, user can call ReceiveResponseFrom7000_ms to get the response from 7000 or I-87K.

Prototype

int ReceiveResponseFrom7000_ms(int iPort, unsigned char *cCmd, long lTimeout, int iChksum);

Arguments

iPort	1 to 8 for I-7188x series. 0,1,3,4 for I-8000 series.
cCmd	Response received from 7000 or I-87K. If checksum is enabled, the function checks and remove the checksum. The CR is removed.
lTimeout	Set the timeout, unit is ms.
iChksum	1 is to enable checksum, 0 is to disable checksum.

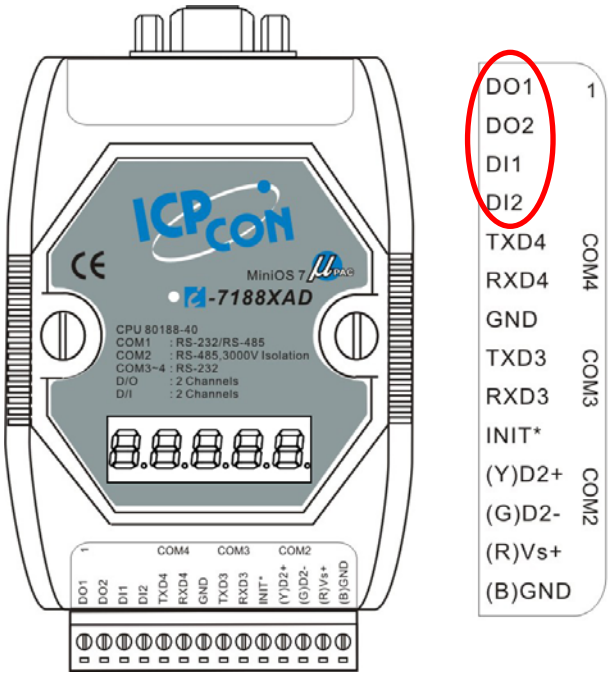
Return Value

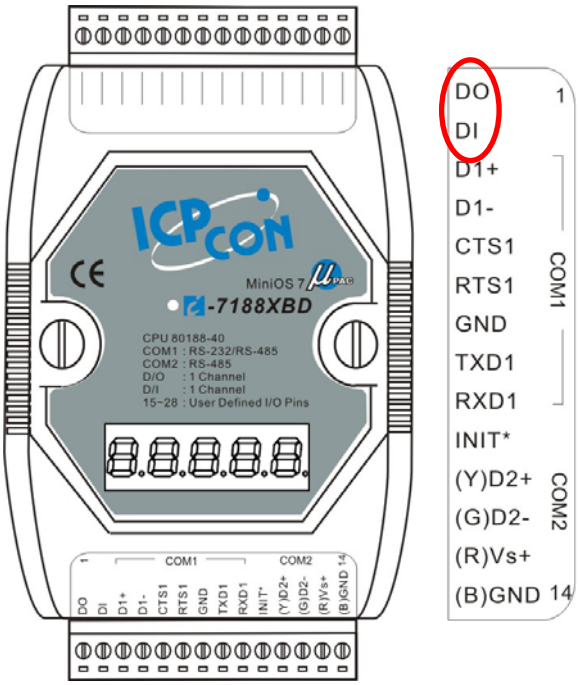
NoError	On success
Error code(refer to Sec.15,constant defined for I-7188/I-8000 series)	On fail.

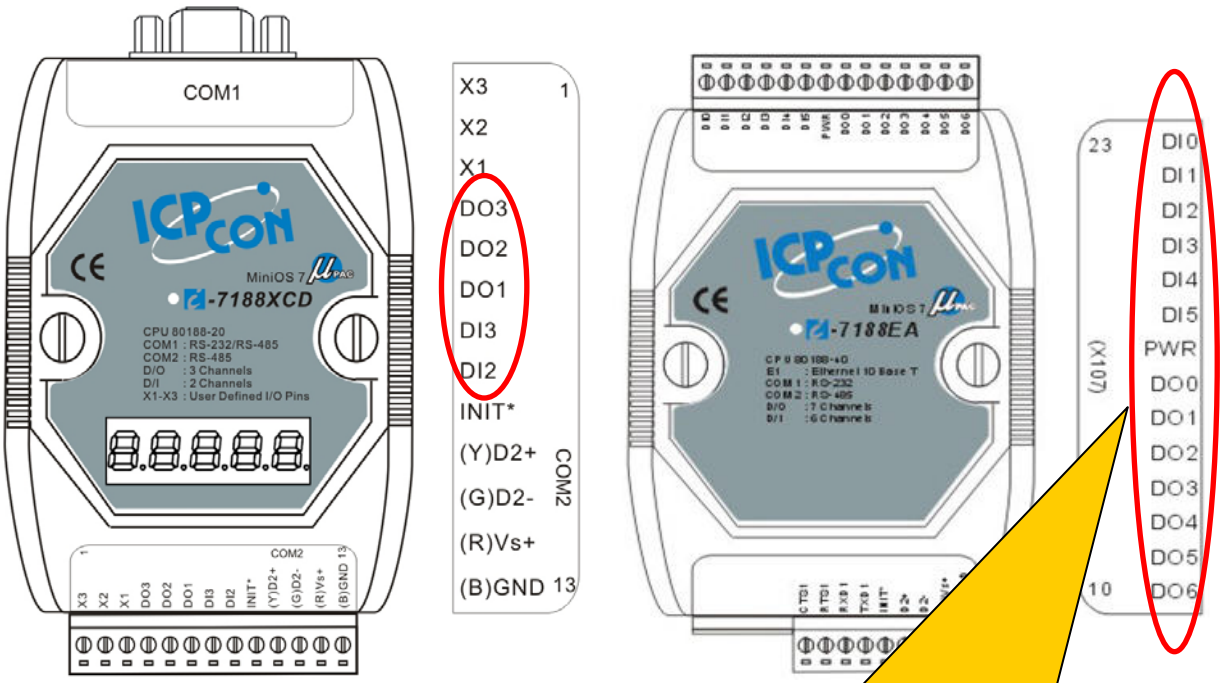
12. Programmable IO Functions

12.1 On Board DIO for I-7188 Series

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
SetDo1	—	Y	Y	Y	Y	—	—	—	—
SetDo2	—	Y	—	Y	Y	—	—	—	—
SetDo3	—	—	—	Y	Y	—	—	—	—
GetDi1	—	Y	Y	—	Y	—	—	—	—
GetDi2	—	Y	—	Y	Y	—	—	—	—
GetDi3	—	—	—	Y	Y	—	—	—	—
GetDo1	—	Y	Y	Y	Y	—	—	—	—
GetDo2	—	Y	—	Y	Y	—	—	—	—
GetDo3	—	—	—	Y	Y	—	—	—	—
Y: The module can use this function.									







DI/DO functions like “int X107_Read_All_DI(void)” for I-7188EA is located at CD:Napdos\7188e\Demo\BC_TC\Lib\Xboard or http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188e/demo/bc_tc/lib/xboard/x107.h

12.1.1 SetDo Functions

Description

Set DO pins to output high or low on the module.

Prototype

void SetDo1(int mode);
void SetDo2(int mode);
void SetDo3(int mode);

Arguments

mode	0 is to output Low. 1 is to output High.

Return Value

None.

12.1.2 GetDI and GetDO Functions

Description

Get status of DO or DI pins on the module.

Prototype

```
int GetDi1(void);
int GetDi2(void);
int GetDi3(void);
int GetDo1(void);
int GetDo2(void);
int GetDo3(void);
```

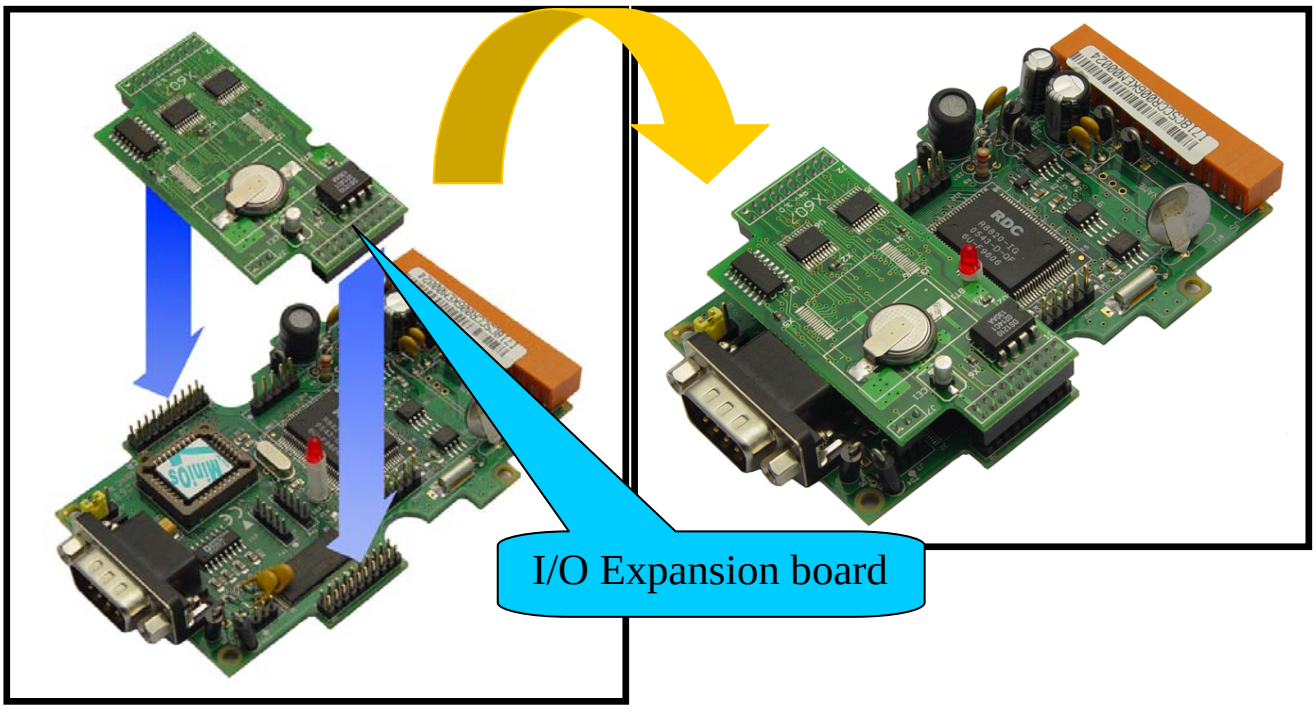
Arguments

None.

Return Value

0	Status is low.
others	Status is high.

12.2 Expansion Board DIO_AIO For I-7188 series



We provide extra functions and library for IO expansion board.

(About DIO/AIO modules of IO expansion board information:

CD: Napdos\7188XABC\Xboard\Document or

http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/xboard/document/)

Functions description is in header file like x607.h.

The header file locations are as follows:

Expansion board for I-7188XA→ CD: Napdos\7188XABC\7188XA\Demo\BC_TC\lib\XBoard or

http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/7188xa/demo/bc_tc/lib/xboard/

Expansion board for I-7188XB→ CD: Napdos\7188XABC\7188XB\Demo\BC_TC\Lib\Xboard or

http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/7188xb/demo/bc_tc/lib/xboard/

Expansion board for I-7188XC→ CD: Napdos\7188XABC\7188XC\Demo\BC_TC\LIB\Xboard or

http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/7188xc/demo/bc_tc/lib/xboard/

12.3 IO For I-8000

12.3.1 I-8K series

12.3.1.1 DO Functions

Description

Set the output value of I-8000 Digital output module.
DO_8() for the D/O modules that have 1-8 output points.
DO_16() for the D/O modules that have 1-16 output points.
DO_32() for the D/O modules that have 1-32 output points.
These functions sets the LED display value automatically.

Prototype

void DO_8(int slot,unsigned char cdata);
void DO_16(int slot,unsigned int cdata);
void DO_32(int slot,unsigned long cdata);

Arguments

slot	For 4 slots: 0-3. For 8 slots: 0-7.
cdata	Each bit for one output point. 1 for Active, 0 for inactive. If the output is Open Collector, active means the transister is turn on. If the output is Relay, active means NO(Normal Open) pin connect to COM pin.

Return Value

None.

12.3.1.2 DI Functions

Description

Read the input value of I-8000 Digital input module.
DI_8() for the D/I modules that have 1-8 points.
DI_16() for the D/I modules that have 1-16 points.
DI_32() for the D/I modules that have 1-32 points.

Prototype

unsigned char DI_8(int slot);
unsigned int DI_16(int slot);
unsigned long int DI_32(int slot);

Arguments

slot	For 4 slots: 0-3. For 8 slots: 0-7.

Return Value

Each bit for one input. 1 for active, 0 for inactive.

12.3.1.3 DIO Functions

Description

Set the output value of I-8000 Digital output module.
DIO_DO_8() for the DI8_DO8 or DI4/DO4 modules.
DIO_DO_16() for the DI16_DO16 modules.
These functions sets the LED display value automatically.

Prototype

void DIO_DO_8(int slot,unsigned char cdata);
void DIO_DO_16(int slot,unsigned cdata);

Arguments

slot	For 4 slots: 0-3. For 8 slots: 0-7.
cdata	Each bit for one output point. 1 for Active, 0 for inactive. If the output is Open Collector, active means the transister is turn on. If the output is Relay, active means NO(Normal Open) pin connect to COM pin.

Return Value

None.

Note:

Read input value on DI8_DO8/ DI4_DO4/ DI16_DO16 modules is to use Sec.12.3.1.2 functions.

12.3.1.4 AI_AO Functions

We provide extra functions and library for AI or AO modules like I-8017H/I-8024.

(About AI/AO modules information:

CD: napdos/dcon/io_module/8k/aio/index.htm or

http://www.icpdas.com/products/PAC/i-8000/8000_IO_modules.htm)

Functions description is in header file like 8017h.h.

The head file locations are as follows:

CD: Napdos\8000\841x881x\demo\Lib or

http://ftp.icpdas.com/pub/cd/8000cd/napdos/8000/841x881x/demo/Lib/

12.3.1.5 Universal DIO Functions

12.3.1.5.1 UDIO_ReadConfig_16

Description

For universal DI/DO, such as 8050.
Get respective setting mode of 16 channels.

Prototype

unsigned UDIO_ReadConfig_16(int slot);

Arguments

slot	For 4 slots: 0-3. For 8 slots: 0-7.

Return Value

0x00 to 0xFF	Each bit represents a channel. Total is 16 bits. 0: for DO mode. 1: for DI mode.

12.3.1.5.2 UDIO_WriteConfig_16

Description

For universal DI/DO, such as 8050.
Set respective setting mode of 16 channels.

Prototype

void UDIO_WriteConfig_16(int slot,unsigned config);

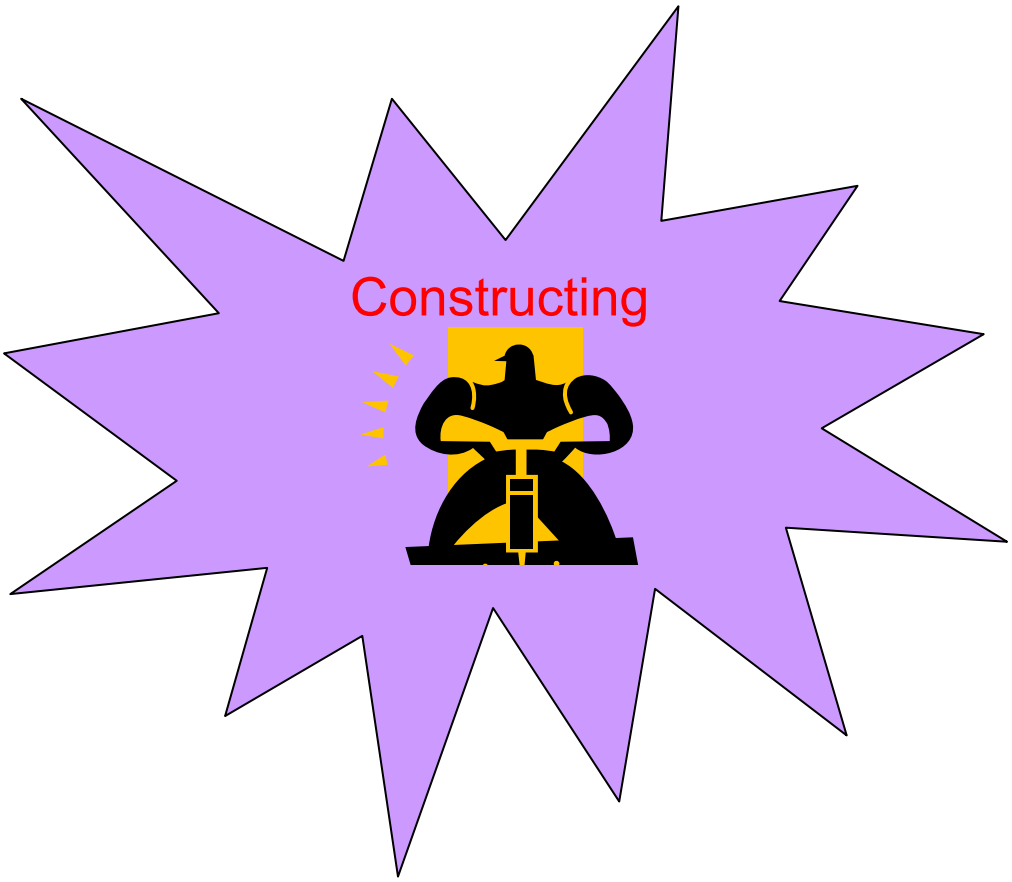
Arguments

slot	For 4 slots: 0-3. For 8 slots: 0-7.
config	Each bit represents a channel. Total is 16 bits. 0: for DO mode. 1: for DI mode.

Return Value

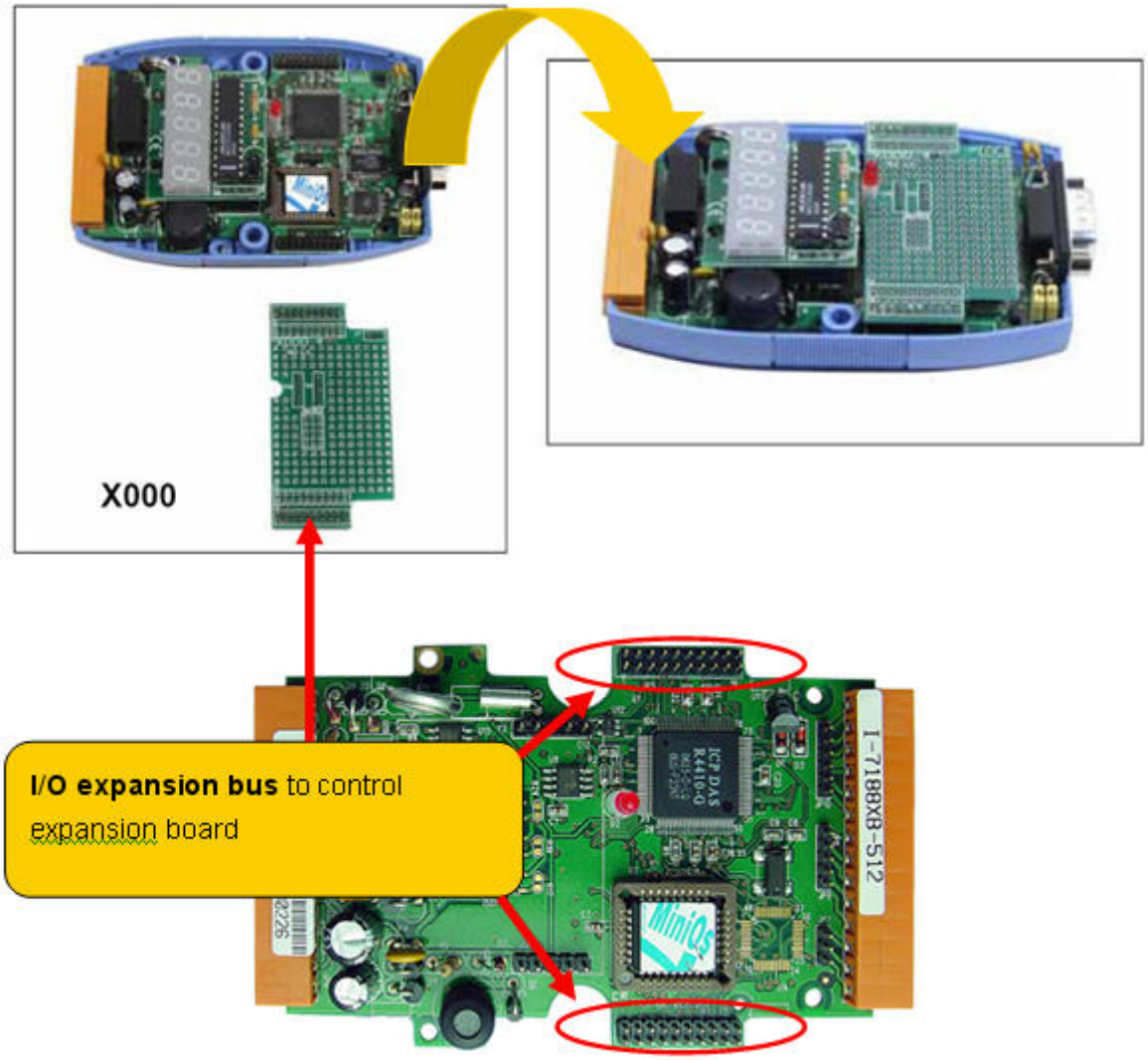
None.

12.3.2 I-87K series



12.4 IO Expansion Bus and CPU Pins

Remark
1. **User can use those function to control 32 PIO pins of CPU(Am188ES) or each pin of I/O expansion bus**. 2. **If the user wants to design new IO expansion board using prototype board like X000, these functions in this section helps you to develop your own expansion board.** 3. I/O expansion bus refers to CD: Napdos\7188XABC\Xboard\Document or http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/xboard/document/



12.4.1 Respective Functions Table

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
SetPioDir	—	Y	Y	Y	Y	Y	Y	Y	Y
SetPio	—	Y	Y	Y	Y	Y	Y	Y	Y
GetPio	—	Y	Y	Y	Y	Y	Y	Y	Y
SetDio4Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio9Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio14Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi0Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi1Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo0Dir	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo1Dir	—	Y	Y	Y	Y	Y	Y	Y	—
GetDio4	—	Y	Y	Y	Y	Y	Y	Y	—
GetDio9	—	Y	Y	Y	Y	Y	Y	Y	—
GetDio14	—	Y	Y	Y	Y	Y	Y	Y	—
GetTi0	—	Y	Y	Y	Y	Y	Y	Y	—
GetTo0	—	Y	Y	Y	Y	Y	Y	Y	—
GetTi1	—	Y	Y	Y	Y	Y	Y	Y	—
GetTo1	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio4High	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio9High	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio14High	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi0High	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo0High	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi1High	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo1High	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio4Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio9Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetDio14Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi0Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo0Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetTi1Low	—	Y	Y	Y	Y	Y	Y	Y	—
SetTo1Low	—	Y	Y	Y	Y	Y	Y	Y	—
ClockHigh	Y	Y	Y	Y	Y	Y	Y	Y	Y

ClockLow	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
ClockHighLow	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Y: The module can use this function.									

12.4.2 SetPioDir

Description

Set the PIO pins of CPU to Input/Output mode.
General purpose function for all PIO pins. Please be careful to use this function.
!!! NOT ALL 32 PIO pins can be used by user. !!!
These pins which can be used are like PIO 6(ARDY), 1(TMROUT1), 10(TMROUT0), 0(TMRIN1), 11(TMRIN1), 4(DT/R), 9(A19), 14(MCS0), 30(INT4).

Prototype

void SetPioDir(unsigned pin,int dir);

Arguments

pin	0~31.
dir	0 : Set the PIO pin to output mode. 1 : Set the PIO pin to input with pull high(for some pin is pull low.) 2 : Set the PIO pin to Input mode without pull high/low. 3 : Set the PIO pin to normal mode.

Return Value

None.

12.4.3 SetPio

Description

Set the PIO pins of CPU to output High/Low.
General purpose function for all PIO pins. Please be careful to use this function.
!!! NOT ALL 32 PIO pins can be used by user. !!!
These pins which can be used are like PIO 6(ARDY), 1(TMROUT1), 10(TMROUT0), 0(TMRIN1), 11(TMRIN1), 4(DT/R), 9(A19), 14(MCS0), 30(INT4).

Prototype

void SetPio(int pin,int mode);

Arguments

pin	0~31.
mode	0: Set the PIO pin to output Low.
	1: Set the PIO pin to output High.

Return Value

None.

12.4.4 GetPio

Description

Get the status of the PIO pins of CPU.

Prototype

int GetPio(int pin);

Arguments

pin	0~31.

Return Value

0	(1) For input mode: the input is low. (2)For output mode: current output is low.
non zero	(1)For input mode: the input is high. (2)For output mode: current output is high.

12.4.5 Set PIO Direction Functions

Description

Set the some PIO pins of CPU to Input/Output mode. These PIO pins connect to I/O expansion bus. About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

```
void SetDio4Dir(int dir);
void SetDio9Dir(int dir);
void SetDio14Dir(int dir);
void SetTi0Dir(int dir);
void SetTi1Dir(int dir);
void SetTo0Dir(int dir);
void SetTo1Dir(int dir);
```

Arguments

dir	0 : Set the PIO pin to output mode. 1 : Set the PIO pin to input with pull high(for some pin is pull low.) 2 : Set the PIO pin to Input mode without pull high/low (TO_0 & TO_1). 3 : Set the PIO pin to normal mode.

Return Value

None.

12.4.6 Get PIO Status Functions

Description

When the PIO pins of CPU are set to output mode, it can get current output status from those functions.
When the PIO pins of CPU are set to intput mode, it can get current intput status from those functions.
These PIO pins connect to I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

int GetDio4(void);
int GetDio9(void);
int GetDio14(void);
int GetTi0(void);
int GetTo0(void);
int GetTi1(void);
int GetTo1(void);

Arguments

None.

Return Value

0	Status is low.
others	Status is high.

12.4.7 Set PIO High Functions

Description

When the PIO pins of CPU are set to output mode, these functions are called and set these pins to output High. These PIO pins connect to I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

```
void SetDio4High(void);
void SetDio9High(void);
void SetDio14High(void);
void SetTi0High(void);
void SetTo0High(void);
void SetTi1High(void);
void SetTo1High(void);
```

Arguments

None.

Return Value

None.

12.4.8 Set PIO Low Functions

Description

When the PIO pins of CPU are set to output mode, these functions are called and set these pins to output Low. These PIO pins connect to I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

```
void SetDio4Low(void);
void SetDio9Low(void);
void SetDio14Low(void);
void SetTi0Low(void);
void SetTo0Low(void);
void SetTi1Low(void);
void SetTo1Low(void);
```

Arguments

None.

Return Value

None.

12.4.9 ClockHigh

Description

Set PIO pin 26th of CPU to output high.
The PIO pin 26th connects to SCLK pin of I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

void ClockHigh(void);

Arguments

None.

Return Value

None.

12.4.10 ClockLow

Description

Set PIO pin 26th of CPU to output low.
The PIO pin 26th connects to SCLK pin of I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual"

Prototype

void ClockLow(void);

Arguments

None.

Return Value

None.

12.4.11 ClockHighLow

Description

Send a clock from the PIO pin 26th of CPU. Set it to high first, then set it to low.
The PIO pin 26th connects to SCLK pin of I/O expansion bus.
About the I/O expansion bus pins, please refer to "I/O Expansion Bus for 7188X/7188E User's Manual".

Prototype

void ClockHighLow(void);

Arguments

None.

Return Value

None.

13. Standard IO Functions

Remark
1. COM1 of I-7188/I-8000 series is the standard I/O port. Before user's program calls InstallCom(1,..) , user can use all functions of this section to send/receive data to/from COM1. [**Note:**]**The standard I/O port of I-7188 & I-7188XA is COM4**. 2. All functions of this section do not work after Installcom(1,..) is called. It must use IsCom(1,..), ToCom(1,..), ReadCom(1,..) ... to send/receive data to/from COM1. These functions work again until RestoreCom(1,..) is called. **[Note:]For I-7188 & I-7188XA are Installcom(4,..), RestoreCom(4,..), IsCom(4,..)….** 3. Scanf() cannot be used on MSC/VC++ . Please use LineInput() + sscanf() instead. 4. When the functions of this section are used, the default format setting for standard I/O port is 115200,N,8,1.

13.1 Common Functions for all I-7188/I-8000 Series

13.1.1 Kbhit

Description

Check if any data is currently available in the input buffer of standard I/O port.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int Kbhit(void);

Arguments

None.

Return Value

0	No any data in input buffer.
others	There is data in input buffer and the return value is the next data in buffer. If the next data is '\0', it returns -1(0xFFFF).

13.1.2 Getch

Description

Read one character from standard I/O port. If there is not any data in the input buffer, the function waits until standard I/O port receives any data.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int Getch(void);

Arguments

None.

Return Value

0 to 255	On success.

13.1.3 Gets

Description

Get a string from standard input port.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int Gets(char *str);

Arguments

str	The buffer which stores the input data.

Return Value

The string arguments	On success.

13.1.4 UngetchI

Description

Put one character back to standard I/O port's another input buffer. Next time when Getch() is called, it first checks another input buffer and return the data which is put in another input buffer by UngetchI.

Prototype

int UngetchI(int key);

Arguments

key	0 to 255. If data > 255, only the low byte are sent out.

Return Value

0 (NoError)	On success.
1	The buffer is full.

13.1.5 LineInput

Description

It continuously receives data from standard I/O and puts them into “**buf**” until receiving 0x0D or data number is bigger than “**maxlen**”. The receiving data in the “**buf**” is a string.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int LineInput(char *buf,int maxlen);

Arguments

buf	The buffer which stores the input data.
maxlen	(size of the buffer) -1

Return Value

The input character number.	On success.

13.1.6 Putch

Description

Send one character to standard I/O port.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

void Putch(int data);

Arguments

data	0 to 255. If data > 255, only the low byte are sent out.

Return Value

None.

13.1.7 Puts

Description

Send a string to standard I/O port.
Note: If Installcom(1,..) is called, this function is not used.

Prototype

void Puts(char *str);

Arguments

str	Point to the string which is to send.

Return Value

None.

13.1.8 Print

Description

This function is used to replace printf and the only difference between Print and printf is Print does not transfer '\n' to '\n'+'\r'.
That is to say the code 0x0A is only sent out about '\n', not 0x0A+0x0D. The printed message is sent out to standard I/O port. (Default format setting: 115200,N,8,1)
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int Print(const char *fmt, ...);

Arguments

Please refer to C's standard function printf.

Return Value

the number of bytes output	On success.
EOF(refer to C's constant define)	On error.

13.1.9 ResetScanBuffer

Description

Set Scanf to use default input buffer (maxlen=80).

Prototype

void ResetScanBuffer(void);

Arguments

None.

Return Value

None.

13.1.10 SetScanBuffer

Description

Set user's buffer for Scanf. The default buffer size is 80 bytes, if more space is need, user's program must support the buffer.

Prototype

void SetScanBuffer(unsigned char *buf,int len);

Arguments

buf	The new buffer for Scanf to use.
len	Length of buffer -1

Return Value

None.

13.1.11 Scanf

Description

Like C's scanf. (C’s scanf cannot used on MSC/VC++)
Note: If Installcom(1,..) is called, this function is not used.

Prototype

int Scanf(char *fmt, ...);

Arguments

Please refer to C's library reference.

Return Value

The number of input fields successfully scanned, converted, and stored. The return value does not include scanned fields that were not stored.	On success
0	No fields were stored
EOF	Attempts to read at end-of-string

14. Others (MISC) Functions

14.1 Common Functions/Variables for all I-7188/I-8000 Series

14.1.1 InitLib

Description

Initialize parameters about hardware in the library.
This is an important function. It must be called before any other functions are used in the library.
It results in unpredictable error if InitLib() doesn't be used.

Prototype

void InitLib(void);

Arguments

None.

Return Value

None

14.1.2 IntVect

Description

Access 80188 CPU's interrupt vertor table. Interrupt vertor table begins at address 00000h that contains up to 256 four-byte address pointers containing the address for the interrupt service routine for each possible interrupt type.
Please refer to AMD 80188 user's manual, "Am186ES and Am188ES User's Manual", for more information.

Prototype

unsigned long far *IntVect;

14.1.3 Is7188x and Is8000x

Description

Check if the program is run under MiniOs7(I-7188x or I-8000x).
It must call InitLib() before calling Is7188x/Is8000x.

Prototype

int Is7188(void);
int Is7188xa(void);
int Is7188xb(void);
int Is7188xc(void);
int Is7188e(void);
int Is8000(void);
int Is8000Tcp(void);

Arguments

None.

Return Value

0 (The program is written for module 'A', but it is downloaded to module 'B' and run. The "Isxxxx()" also return 0.)	Not run on MiniOs7(7188x/8000).
256	Run on MiniOs7 and the flash size is 256K.
512	Run on MiniOs7 and the flash size is 512K.

14.1.4 GetLibVersion

Description

Get the library version of MiniOS7 library.

Prototype

unsigned GetLibVersion(void);

Arguments

None.

Return Value

High byte is major version number, Low byte is the minor version number. For example: Ver. 1.15 returns 0x010F.

14.1.5 GetLibDate

Description

Get library date.

Prototype

void GetLibDate(char *date);

Arguments

date	Store a string about library date. The format is like “Jan 19 2007”.

Return Value

None.

14.1.6 ReadInitPin

Description

Read the status of INIT* pin.

Prototype

int ReadInitPin(void);

Arguments

None.

Return Value

0 (InitPinIsOpen)	Init* pin is floating or connected to VCC
1 (InitPinIsNotopen)	Init* pin is connected to GND

14.1.7 hex_to_ascii

Description

A array that transfers a hexadecimal number to a character. The range is 0x0 to 0xF.

Prototype

extern char hex_to_ascii[16];

Return Value

hex_to_ascii[0x0]= '0' hex_to_ascii[0x1]= '1' hex_to_ascii[0x2]= '2' hex_to_ascii[0x3]= '3' hex_to_ascii[0x4]= '4' hex_to_ascii[0x5]= '5' hex_to_ascii[0x6]= '6' hex_to_ascii[0x7]= '7'	hex_to_ascii[0x8]= '8' hex_to_ascii[0x9]= '9' hex_to_ascii[0xA]= 'A' hex_to_ascii[0xB]= 'B' hex_to_ascii[0xC]= 'C' hex_to_ascii[0xD]= 'D' hex_to_ascii[0xE]= 'E' hex_to_ascii[0xF]= 'F'

14.1.8 ascii_to_hex

Description

Transform a character into a hexadecimal number.

Prototype

int ascii_to_hex(char ascii);

Arguments

ascii	A character that is ready to be transformed. Only ‘0’,’1’,’2’,‘3’,’4’,’5’,‘6’,’7’,’8’,‘9’,’A’,’B’,‘C’,’D’,’E’, F’,’a’,‘b’,’c’,’d’,‘e’ and ’f’ can be transformed.

Return Value

0x0~0xF	On success.

14.1.9 IsResetByPowerOn

Description

Check if the module is reset by power on.

Prototype

int IsResetByPowerOn(void);

Arguments

None.

Return Value

1	Reset by power on.
0	Not reset by power on.

14.1.10 IsResetByWatchDogTimer

Description

Check if the module is reset by watchdog timer.

Prototype

int IsResetByWatchDogTimer(void);

Arguments

None.

Return Value

1	Reset by watchdog timer.
0	Not reset by watchdog timer.

14.2 Respective Functions for all I-7188/I-8000 Series

14.2.1 Respective Functions Table

Functions	I-7188	I-7188XA	I-7188XB	I-7188XC	I-752N	I-7188EX	I-7188EA	I-7188EN	I-8000
SystemSerialNumber	—	**Y**	**Y**	—	Depend on module	**Y**	**Y**	**Y**	**Y**
GetComportNumber	—	**Y**	**Y**	**Y**		**Y**	**Y**	**Y**	**Y**
GetSerialNumber	—	**Y**	**Y**	—		**Y**	**Y**	**Y**	**Y**
Y: The module can use this function.									

14.2.2 SystemSerialNumber

Description

7188X/I-8000/7188E series can add one serial number IC(DS2502). ***After power on, MiniOS7 reads the serial number, and save to ram*** where *SystemSerialNumber read data from.
The serial number length is 8 bytes, SystemSerialNumber[0..7].
If all values are 0x5A means can not find the hardware IC.
If all values are 0xAA means CRC error for the serial number.
Use MiniOS7 command VER shows the serial number. User's program can use following code to show the serial number:

```
Print("Serial Number : ");
for(i=0;i<8;i++){
        Print("%02X ",SystemSerialNumber[i]);
}
```

Prototype

extern const unsigned char far * const SystemSerialNumber;

NOTE

	Data source	Security
*SystemSerialNumber	Read data from RAM	Normal
GetSerialNumber	Read data from IC	High

14.2.3 GetSerialNumber

Description

Read system serial number from hardware IC.
It is unique and 64 bits. The application software can check this number for illegal copy.
Use MiniOS7 command VER shows the serial number.

Prototype

int GetSerialNumber(char *Serial);

Arguments

Serial	Store serial number.

Return Value

0	On success.
-1	Cannot find the hardware IC
-2	Serial number CRC check error. The hardware IC is abnormal.

NOTE

	Data source	Security
*SystemSerialNumber	Read data from RAM	Normal
GetSerialNumber	Read data from IC	High

14.2.4 GetComportNumber

Description

Get COM port number on a module.

Prototype

int GetComportNumber(void);

Arguments

None.

Return Value

Non-negative	COM port number.

14.3 Only For I-8000

Remark
1. To communicate with add-on 87K modules, one must use COM0 to send/receive command, and also use ChangeToSlot( ) to select one slot to connect to COM0. 2. Every I-8000 module can set its system ID, the system ID can be read by function GetNetId( ). 3. There are two kinds of backplane for I-8000, one is 4 slots and the other is 8 slots. Program can call GetNumberOfSlot( ) to get the slot number. 4. There are three system LEDs for I-8000, L1/L2/L3. SetLedL1( )/SetLedL2( )/SetLedL3( ) can be used to control these LEDs (Sec.8.2.3). 5. Terer are four system key for I-8000, SET, MODE, UP, DOWN. IsSytemKey( ) can be used to check if any key is available, use GetSytemKey( ) to read the system key, use ClearSystemKey( ) to clear all system key input on the buffer.

14.3.1 Variables of Name or Type of Module

Description

After system power on or reset, MiniOS7 checks if there are any add-on modules on I-8000 series and store some respective data on bios area.

(1).ModuleType[0..7] saves the type of module. If bit 7 = 1 means that module is parallel module(8nnn), otherwise that module is 87nnn.

(2). ModuleName [0..7] stores the module name. For example, the saved value is 51 if the module is 8051. If the module is 8142 then 142 saved. The saved value is nnn if the module is 8nnn or 87nnn. If nnn=255 means that there's no module on the slot.

(3).ModuleFullName[0] contains the full name of the module on slot 0.
ModuleFullName[1] contains the full name of the module on slot 1.
If there is no module on that slot, the value of ModuleFullName[slot] is "".
For example :
For I-87017, ModuleName[slot]=17, ModuleFullName[slot] is "87017",
For I-87017ML, ModuleName[slot]=17, but ModuleFullName[slot] is "87017ML",
So if the program uses the ModuleName[slot], it can not point out that module is "87017" or "87017ML", but the variable ModuleFullName[slot] can do it.

(4)The code for use of ModuleType and ModuleName to decide the module on the slot of I-8000 are as follows:

```
for(i=0;i & NumberOfSlot;i++){
  if(ModuleType[i]& 0x80){
                    Print("Slot %d = 8%03u\n\r",i, ModuleName [i]);
  }
  else if(ModuleName [i]!=0xFF) Print("Slot %d = 870%02d\n\r",i, ModuleName [i]);
  else Print("Slot %d without any module.\n\r",i);
}
```

Prototype

```
extern unsigned char far *const ModuleType;
extern unsigned char far * const ModuleName;
extern unsigned char far ** const ModuleFullName;
```

14.3.2 ChangeToSlot

Description

Choose which slot can communicate with COM0.

Prototype

void ChangeToSlot(int slot);

Arguments

slot	For 4 slots(84XX): 0-3. For 8 slots(88XX): 0-7.

Return Value

None.

14.3.3 GetNetId

Description

There is a switch which is used to set ID of the module for 8000 series. User's program can use GetNetId() to read the value.

Prototype

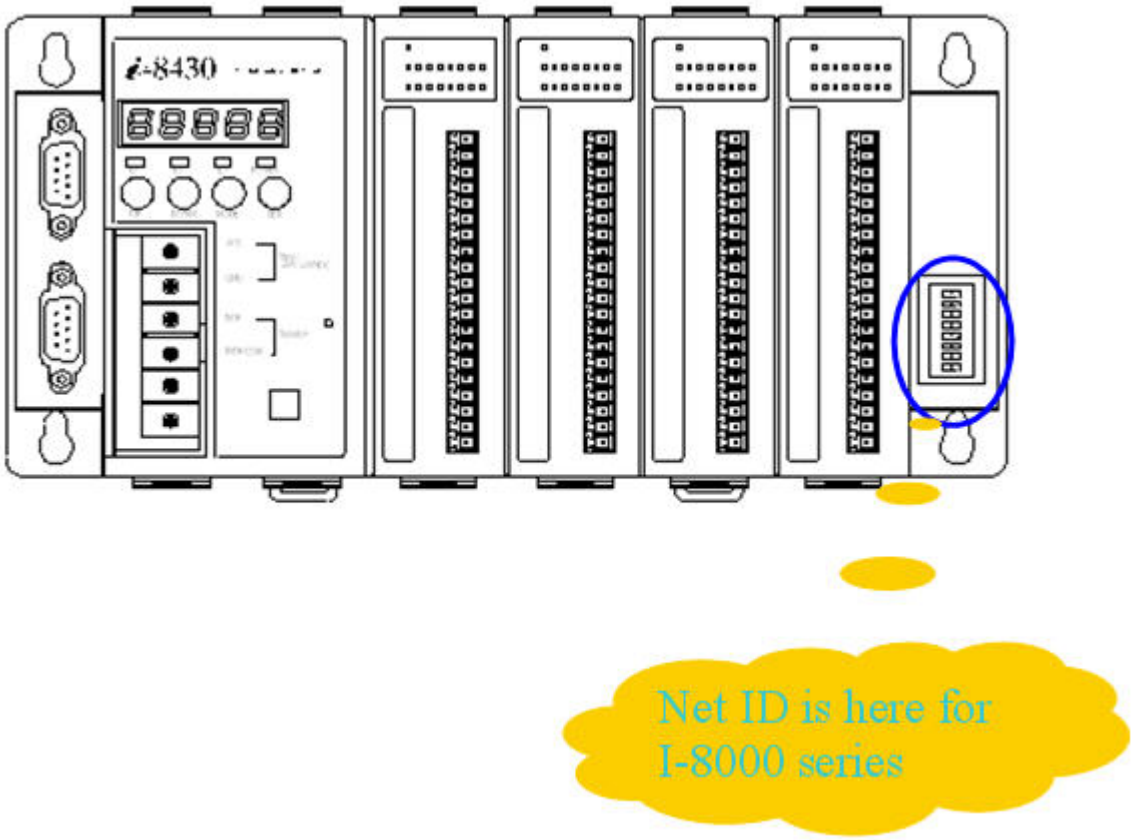
int GetNetId(void);

Arguments

None.

Return Value

0-255。



14.3.4 GetNumberOfSlot

Description

It is used to check the system is 84xx or 88xx. In I-8000 series I-84xx have 4 slots, I-88xx have 8 slots.

Prototype

int GetNumberOfSlot(void);

Arguments

None.

Return Value

4	For I-84xx.
8	For I-88xx.

14.3.5 NumberOfSlot

Description

Get the number of slot to know which type of backplane is, the type of 4 slots or 8 slots.
After GetNumberOfSlot is called, it sets the value into NumberOfSlot.
So it has to call GetNumberOfSlot and then NumberOfSlot is able to be used.

Prototype

extern int NumberOfSlot;

Return Value

4	For I-84xx.
8	For I-88xx.

14.3.6 SlotAddr

Description

Slot0	Slot1	Slot2	Slot3	Slot4	Slot5	Slot6	Slot7
0x80	0xa0	0xc0	0xe0	0x140	0x160	0x180	0x1a0

Prototype

extern int SlotAddr[8];

14.3.7 GetSlotInt

Description

On I-88xx series there are 8 slots. The last 4 slots share the same interrupt signal, it must call GetSlotInt() to check which slot generates the interrupt signal.

Prototype

int GetSlotInt(void);

Arguments

None.

Return Value

4 to 7	The slot that generates the interrupt signal.

14.3.8 ClrSlotInt

Description

I-88xx series has 8 slots for add-on modules. The last 4 slots share the same interrupt signal. Calling ClrSlotInt() can mask the interrupt signal of that slot.

Prototype

void ClrSlotInt(int mask);

Arguments

mask	bit 0~3 represent slot 4~7. One bit mask one slot. 1 for mask. 0 for unmask.

Return Value

None.

14.3.9 ClrAllSlotInt

Description

I-88xx series has 8 slots for add-on modules. The last 4 slots share the same interrupt signal. ClrAllSlotInt() can mask all interrupt singal of all slots.

Prototype

void ClrAllSlotInt(void);

Arguments

None

Return Value

None.

14.3.10 KeyStatus

Description

Read current system key status.
If the bit value is 0, the key status is up.
If the bit value is 1, the key status is down.

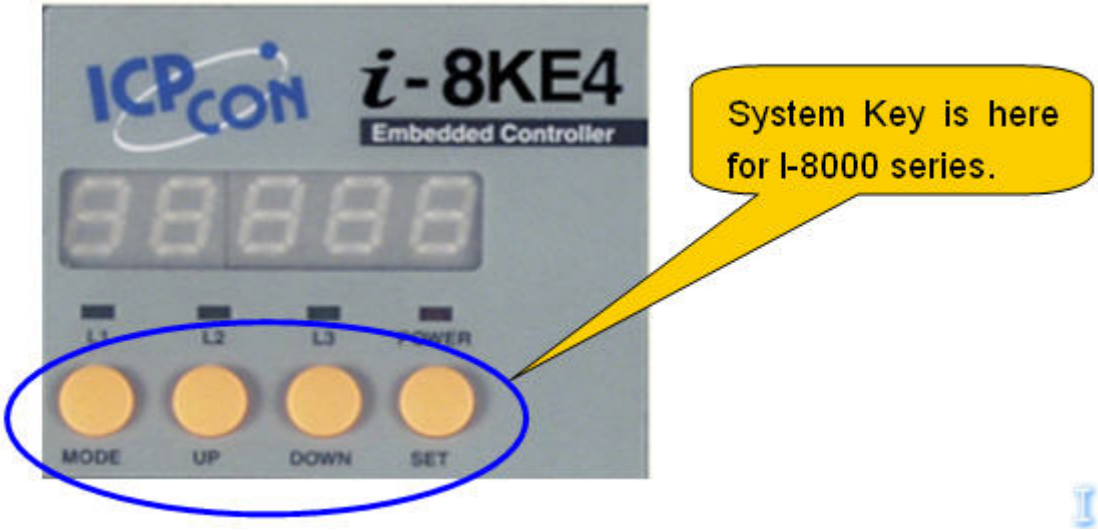
#define SKEY_SET_DOWN 0x08
#define SKEY_DOWN_DOWN 0x04
#define SKEY_UP_DOWN 0x02
#define SKEY_MODE_DOWN 0x01

Prototype

extern unsigned char far * const KeyStatus;

Note

	KeyStatus	GetSystemKey/ _GetSystemKey
Difference	1). Read current system key values 2). Four key data one time. 3). Real time data	1). Read FIFO buffer value. 2). One key data one time. 3). May be real time data.



14.3.11 IsSystemKey

Description

Check if any input of system key is available in the system key input buffer (FIFO buffer, size is 16).
If InstallNewTimer is called, it must use _IsSystemKey instead of IsSystemKey.

Prototype

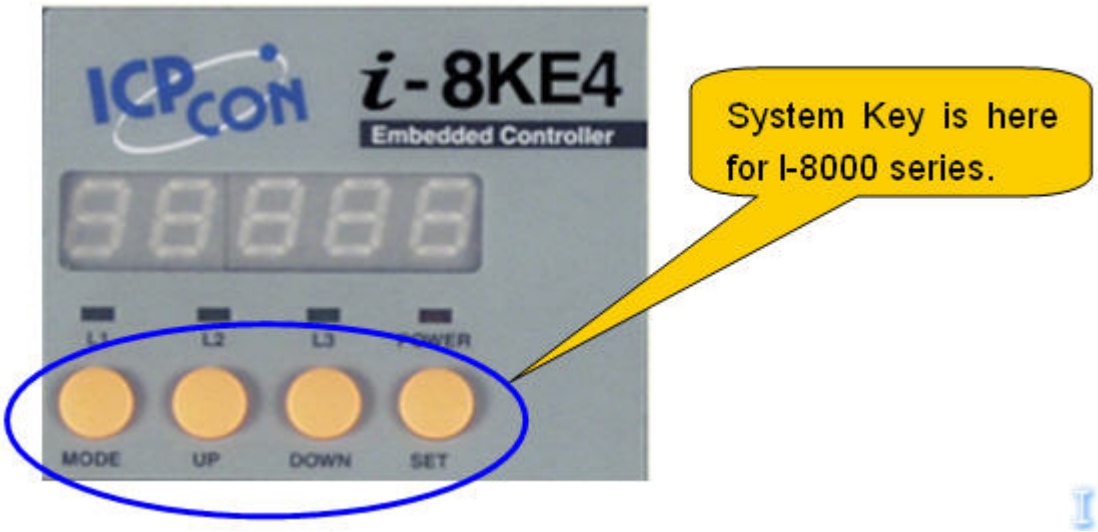
int IsSystemKey(void);
int _IsSystemKey(void);

Arguments

None.

Return Value

0	For no any input of system key.
Others	There is input data of system key in the buffer.



14.3.12 GetSystemKey

Description

On I-8000 series, system support 4 keys, SET, MODE, UP, and DOWN.
GetSystemKey() reads one key input value from system key input buffer (FIFO buffer, size is 16). If there is no system key input value, the function waits until any system key input.
If InstallNewTimer is called, it must use _GetSystemKey instead of GetSystemKey.

Prototype

int GetSystemKey(void);
int _GetSystemKey(void);

Arguments

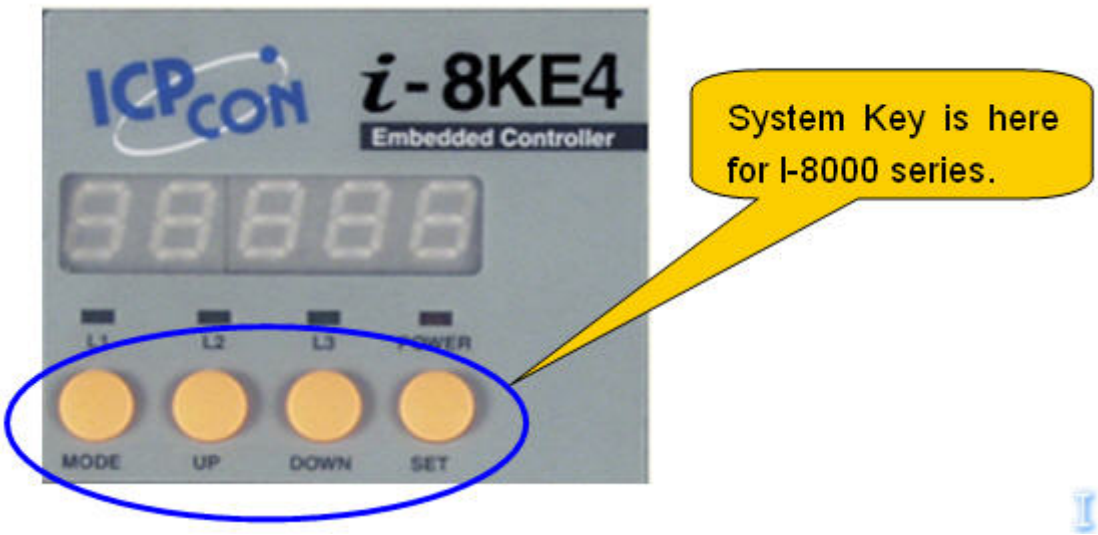
None.

Return Value

1 (SKEY_SET)	For SET key.
2 (SKEY_MODE)	For MODE key.
3 (SKEY_UP)	For UP key.
4 (SKEY_DOWN)	For DOWN key.

Note

	KeyStatus	GetSystemKey/ _GetSystemKey
Difference	1). Read current system key values 2). Four key data one time. 3). Real time data	1). Read FIFO buffer value. 2). One key data one time. 3). May be real time data.



14.3.13 ClearSystemKey

Description

Clear all data in the System key input buffer.
If InstallNewTimer is called, it must use _ClearSystemKey instead of ClearSystemKey.

Prototype

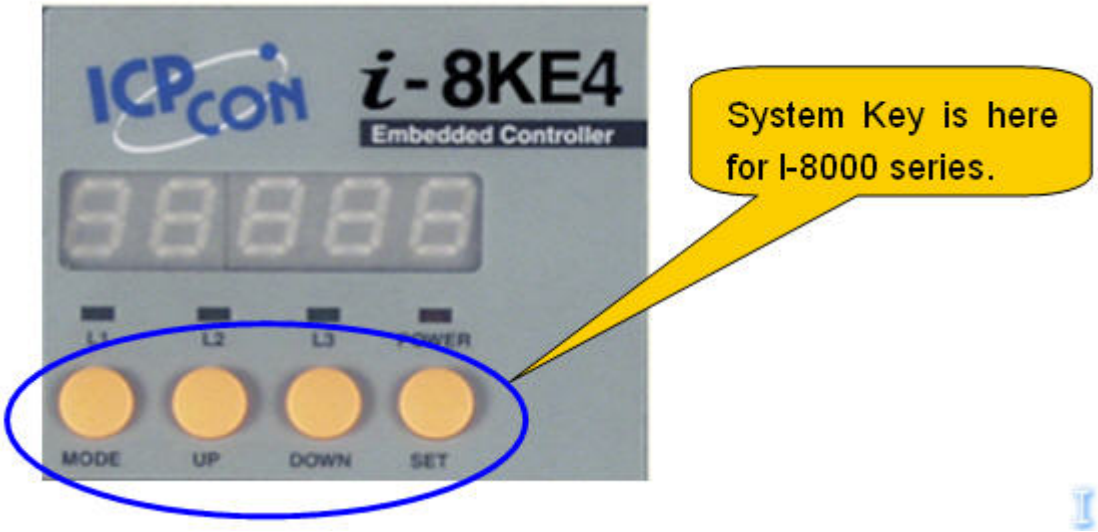
void ClearSystemKey(void);
void _ClearSystemKey(void);

Arguments

None.

Return Value

None.



14.3.14 InstallNewTimer

Description

It changes system timer (default is timer2) to another new timer and shorts the time of timer interrupt executing.
After InstallNewTimer is called, it supports new timer ISR and does not call INT 0x1C(every 55 ms).
After InstallNewTimer is called, it must use _IsSystemKey instead of IsSystemKey, _GetSystemKey instead of GetSystemKey and _ClearSystemKey instead of ClearSystemKey.

Prototype

void InstallNewTimer(void);

Arguments

None

Return Value

None.

14.3.15 AddCom2Fun

Description

After calling AddCom2Fun(), the functions for COM2 are linked into the .exe file.
The default setting is IsCom(2)/ReadCom(2)/... do not work for I-8x1x.
After calling AddCom2Fun(), IsCom(2)/ReadCom(2)/... work for I-8x1x.
Note: IsCom_2()/ReadCom_2()/... always work for I-8x1x.

Prototype

void AddCom2Fun(void);

Arguments

None

Return Value

None.

15. Constant Defined for I-7188/I-8000 Series

Name	Value
NoError	0
InitPinIsOpen	0
InitPinIsNotopen	1
QueueIsEmpty	0
QueueIsNotEmpty	1
PortError	-1
DataError	-2
ParityError	-3
StopError	-4
TimeOut	-5
QueueEmpty	-6
QueueOverflow	-7
PosError	-8
AddrError	-9
BlockError	-10
WriteError	-11
SegmentError	-12
BaudRateError	-13
CheckSumError	-14
ChannelError	-15
TriggerLevelError	-17
DateError	-18
TimeError	-19
OutOfMemory	-20
TimeIsUp	1