IPC@CHIP Documentation index - SC12 @CHIP-RTOS V1.10

End of document

General introduction - SC12 @CHIP-RTOS V1.10

Short general explanation

The name @CHIP-RTOS is used as the term for the operating system of the IPC@CHIP (most of older versions and manuals are still using the term BIOS). For an overview of the provided features and the general architecture of the operating system see **@CHIP-RTOS architecture**.

The operating system is able to execute multiple (up to 12) DOS programs simultaneous. Each program runs as a task of the RTOS kernel. These 16 bit DOS programs can be created with a variety of development tools. However, the instruction opcodes in these programs must be confined to the 80186 instruction set. For C/C++ programming, Beck offers the Borland 5.02 development environment (see **www.beck-ipc.com**).

This document describes the Application Programmers Interface (API) of the @CHIP-RTOS. It should be used as a reference manual for IPC@CHIP application programmers. For a better understanding of this manual the reader should have some experience in programming DOS applications.

The document explains how to call every provided function of the @CHIP-RTOS from user application programs. The API of the @CHIP-RTOS is structured into different sections (e.g. HARDWARE API, RTOS API, TCPIP API, ...). All sections are listed as HTML-links on the **main index page** of this document.

For every API section the @CHIP-RTOS provides a software interrupt with different function numbers for every API call. The software interrupt mechanism is the standard way for DOS programs to call internal functions of the operating system (e.g. file operations, reading/printing characters, ...).

Before calling the specified software interrupt with e.g. the standard C-Library call `int86()` or the Assembler instruction `INT xx`, every necessary function parameter must be loaded into the processor registers. Each API call description details which parameters are required in the registers. The @CHIP-RTOS uses the parameters from the processor registers and executes the corresponding function. After the software interrupt execution some specified processor registers contains the return result(s) from the function. The meaning of the return result(s) are also specified with each API call description.

At our download page under **www.beck-ipc.com**, we provide a collection of program examples, where the usage of API calls is demonstrated.
For a easier and more practical usage of the API, we provide a set of C-Libraries. This collection of C-Files and H-Files contains every API software interrupt call, implemented inside of a C-Function. The application programmer can integrate these files at his program project and then call the appropriate C-function instead of directly calling the software interrupt.

You will find some useful general notes about programming DOS applications for the IPC@CHIP under **Programming notes**.

For some necessary knowledge about hardware details of the IPC@CHIP see the hardware manual

available from the download area of **www.beck-ipc.com**.

End of document

@CHIP-RTOS Software overview

RTOS
- Tasks: 35
- Sum of Semaphores, Timers, Event groups: 60
- Message exchanges: 10

These are the entire RTOS resources. Some of them are used by the RTOS itself. See RTOS API documentation for available user resources.

RTOS Filesystem for
- Internal ramdisk
- Internal flashdiskdrive
- External drive

TCP/IP Stack
- TCP
- UDP
- ARP
- ICMP
- IGMP
- Socket interface
- Sockets: 64
- Internal device interfaces: 4
  - Ethernet
  - PPP server
  - PPP client
  - Loopback

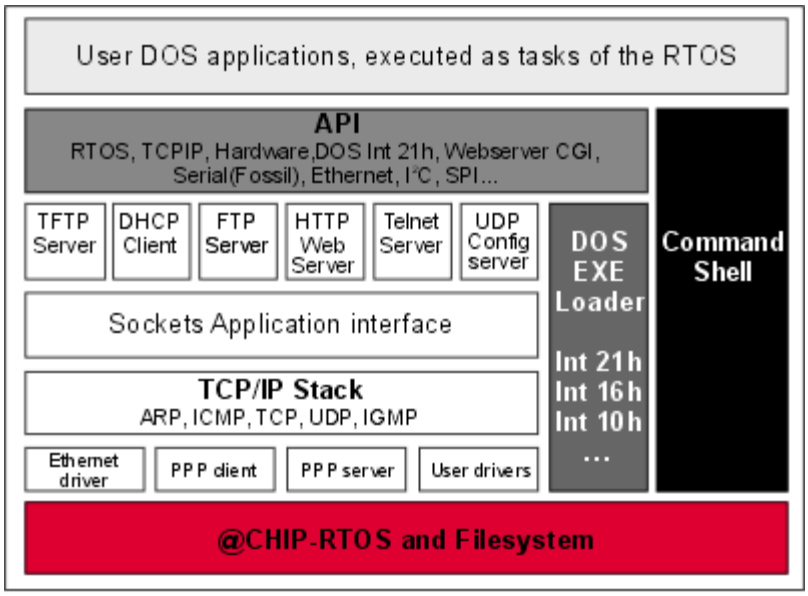
TCP/IP applications
- HTTP Webserver
- FTP server
- Telnet server
- DHCP client
- TFTP server (optional)
- SNMP MIB variable support (optional)
- UDP config server for @CHIP-RTOS upgrade

DOS-EXE Loader
- Up to 12 DOS application programms can run as tasks of the RTOS

DOS-like command shell
- Supports a subset of DOS commands and @CHIP-RTOS specific commands via Telnet, serial devices or user specific stdio

Scalable @CHIP-RTOS
- Support of 6 different versions, including various @CHIP RTOS features

@CHIP-RTOS Upgrade via Ethernet/UDP or serial Bootstraploader



@CHIP-RTOS architecture

Application Programmer Interface
- RTOS
- TCP/IP socketinterface
- DOS interrupt 21h and others
- Webserver CGI
- Hardware
- I2C
- Software SPI (optional)
- Serial devices (Fossil interface)
- Ethernet packet driver (Add. use of ethernet for NON-TCP/IP purposes)
- Special @CHIP-RTOS services

Serial filetransfer via xmodem

DOS application examples
- RTOS API examples
- TCP/IP API examples
  - FTP client
  - HTTP client
  - Other TCP/IP examples
- Webserver CGI examples
- External IDE disk driver
- Hardware API examples
- I2C examples
- Fossil examples
- API examples written in Turbo Pascal
- API C-Libraries
- more

Programming notes - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

Introduction

Here are some useful notes for programming applications for the IPC@CHIP.
These notes contains general rules for programming DOS applications for the IPC@CHIP and important information about internals of the @CHIP-RTOS which can prevent the programmers from making fatal errors.

Common notes for building IPC@CHIP user applications

1. Limited Flash Write Cycles (of IPC@CHIPs internal Flash):
   To open files on the flash disk drive A: for reading or writing, you can use the standard C functions fopen or open. We recommend the use of the fopen/fread/fwrite calls instead of open/read/write to reduce the flash write cycles, because these cycles are limited.
   For further information see the **FlashWriteCycles** document.

2. Compiler option settings:
   We recommend the usage of large memory model for user applications.
   The compiler must produce 186 processor instruction code.
   The data alignment must be set to 8 Bit (Byte alignment), not to 16-Bit (Word alignment)!

3. No "exception handling" at Borland C 4.x or 5.x projects:
   Users of Borland C-Compilers (which provides "exception handling libraries" as it's default setting for 4.5 or 5.02) should choose for their program project "No exceptions" at the "Target Expert" window. This setting saves Flash and RAM memory space.

4. Do not select "Test stack overflow" at Borland C 4.x or 5.x project compiler settings:
   If "Test stack overflow" is selected a stack check is done at every function call. If stack overflow is recognized the program halts.
   This mechanism is not usable in user applications which are using several tasks inside their

application, because every task has its own stack and the Borland runtime library stack functions are not aware of this.

5. Turbo Pascal programmers should include the example file `SC12.INC` (part of each IPC@CHIP Pascal example program). At the start of their program, they should install the procedure `Terminate_Program` as the default exit procedure of the program. Usage of the standard `unit crt` is not allowed.

6. Before starting the first Turbo Pascal program at the IPC@CHIP the **memopt 1** command must be executed (e.g. in the autoexec.bat)

7. If programs that depend on one another are started from a batch file (e.g. if a user program needs a previous driver program) the **Batchmode 1** in chip.ini should be set.

8. The old CLIB library (< V2.00) and some older examples are using the standard `int86` and `int86x` calls from the Borland C standard library. We used these functions for more readable code. If developers need higher performance in their applications, it will be better to use the new CLIB library (>= V2.00) which implements the API calls with inline assembler code. With inline assembler instructions it is possible to load the processor registers directly with the needed parameters and directly invoke the appropriate software interrupt `int xxh`.

9. For certain reasons it is not advisable to design an IPC@CHIP application as a set of several DOS programs:
   1. It is more meaningful to run several tasks (built with the RTOS API) inside of one DOS program because this requires much less RAM and Flash memory space, than running several DOS programs as tasks of the @CHIP-RTOS.
   2. It can lead to high memory fragmentation and insufficient memory, if more than one or two DOS programs are running on the IPC@CHIP, which will be often terminated and restarted again.

10. Most of the used pointer variables and function parameters in the old CLIB library C files (< V2.00) are not declared as FAR pointer, because all of our test programs are using the memory model Large (at model Large all pointers become automatically FAR pointers). If you want to use e.g. memory model Small for your application you should use the new CLIB (>= V2.00) and declare the pointer variables and function parameters inside your program explicit as FAR pointers.

11. It is not advisable to use in your application the Borland C-library `malloc` function with memory model Large. In that case the DOS program tries to increase its own memory block with **int21h 0x4A**. If another program is loaded after that program the `malloc` call fails, because there is no memory space left between the two programs. It is possible to define a memory gap between two loaded programs at **chip.ini** but then you must know what maximum memory is required by your `malloc` calls. It is better to reserve inside of your application the memory by declaring an unsigned char array (or the type you require) with the needed memory size.

12. The @CHIP-RTOS has its own memory strategy.
    The @CHIP-RTOS always allocates memory in the following way:
    DOS programs are always loaded at the first lowest free memory block. For memory blocks allocated inside of the @CHIP-RTOS or e.g. with **DOS service 0x48** the @CHIP-RTOS always start searching for a free memory block from the highest possible RAM address. So the largest free memory block of the system is always located in the middle of the @CHIP-RTOS memory area. The shell command **mem** shows the state of the internal memory map.

Top of list
Index page

Using RTOS API

1. Timer procedures must be implemented as short as possible, because they are executed in the Kernel task. Large timer procedures are blocking the kernel and other tasks of the @CHIP-RTOS. Do not use large Clib functions like `printf` inside of a timer procedure. This can cause a fatal stack overflow in the kernel task (stack size 1024 Bytes)

2. Declaring of timer procedures with Borland C:
```
void huge my_timer(void)
```

3. Microsoft C:
```
void far _saveregs _loadds my_timer(void)
```

4. Turbo Pascal:
```
procedure Timer1_Proc;interrupt;
begin

[... your code ...]

(*********************************************)
(* This is needed at the end of the Timer Proc. *)
     asm
     POP BP
     POP ES
     POP DS
     POP DI
     POP SI
     POP DX
     POP CX
     POP BX
     POP AX
     RETF
     end;
(*********************************************)
end;
```

5. DOS applications are running as a task of the @CHIP-RTOS with a default priority of 25. If the users application doesn't go to **sleep**, lower priority tasks like the FTP server or the Web server will not work. In major loops within user applications, the programmer should insert **sleep calls** if the FTP server or Web server should work during the runtime of the user application.

6. The stack of a task **created** inside of a DOS application should have a minimum stack size of 1024 Bytes. Programmers of task functions who are using Microsoft C-Compilers with C-Library functions, e.g. `sprintf`, which requires a lot of stack space should increase this allocation to 6144 (6 Kbytes). More stack space for the task is also required if your task function uses a large amount of stack for automatic data (local variables) declared inside the task function call.

7. Declaring of task functions with Borland C:
```
void huge my_task(void)
```

8. Declaring of task functions with Microsoft C:
```
void far _saveregs _loadds my_task(void)
```

9. Before exiting an application, every task or timer procedure created inside of this application program must be removed.

10. A **sleep call** with parameter 1 millisecond takes equal or less than one millisecond. If a user needs a minimal sleep time of 1 millisecond then RTX_SLEEP_TIME must be called with value 2 milliseconds.

11. It is possible to **create tasks** with a time slicing value in the **taskdefblock structure.**. Please note that the kernel executes time slicing only between tasks which have same priority. Other priority tasks are not affected.

RTOS API

Top of list
Index page

Programming CGI functions

1. Avoid large loops inside of CGI functions. CGI functions must be executed as fast as possible. Large execution times in CGI functions block the Web server task preventing response to other http requests.
2. Declaring of CGI functions with Borland C:
   ```
   void huge my_cgi_func(rpCgiPtr CgiRequest)
   ```
3. Declaring of CGI functions with Microsoft C:
   ```
   void far _saveregs _loadds my_cgi_func(rpCgiPtr CgiRequest)
   ```
4. Declaring of CGI functions with Borland Turbo Pascal:
   ```
   procedure my_cgi_func;interrupt;
   ```
5. Turbo Pascal users must install their CGI functions with API call **Install CGI Pascal function**
6. C-Programmers must use **Install CGI function**
7. Avoid the declaration of large arrays as local variables inside of the CGI function to prevent stack overflows
8. Users of Microsoft C should set **Web server stacksize** in chip.ini up to 6144 KBytes, to prevent stack overflows, if Microsoft CLib functions like sprintf are used.
9. Dynamic HTML or text pages, which are created inside of a user CGI function should be as small as possible. The Web server inside must allocate a temporary buffer for storing this page before sending it to the browser. If your application builds large dynamic HTML page in RAM, your application should not use all available memory in the IPC@CHIP because the Web server will need some memory for allocating temporary buffers for this page. How much memory should be left: This depends on the application and the sizes of the created dynamic HTML or CGI pages.
10. Before exiting an application, every CGI function installed by the application must be removed with **Remove CGI page**

CGI API

Top of list
Index page

Configure the FTP server

1. The default FTP idle timeout is set to 300 seconds. This is a very long time for waiting, if FTP commands fail. The **idletimeout** can be reduced in chip.ini.

Top of list
Index page

Usage of the TCP/IP API

1. Processing of a socket callback functions (see **Register callback**) should be kept at a minimum to prevent stack overflows.
2. Declaring of socket callback functions with Borland C:
   `void huge my_callback(int socketdescriptor, int eventflagmask)`
3. Declaring of socket callback functions with Microsoft C:
   `void far _saveregs _loadds my_callback(int sd, int eventmask)`
4. The internal TCP/IP stack of the IPC@CHIP allocates memory for buffers smaller than 4096 bytes from a pre-allocated memory block. Larger buffers are allocated direct from the CHIP-RTOS. If user TCP/IP network communication sends/receives packets with a size larger than 4096 bytes, the user application should not use all available memory in the IPC@CHIP because of these additional allocations. There should be in that case always a minimum of 30-40 KBytes of free available memory in the IPC@CHIP. The **mem command** shows the whole memory list of the IPC@CHIP at runtime. The maximum amount of TCP/IP memory can be configured at chip.ini (see **Set TCP/IP memory size**). The application programmer can reduce or increase this size in chip.ini. With the API call **Get TCP/IP memory info** it is possible to control the TCP/IP memory usage at the application runtime.

TCP/IP API

Top of list
Index page

General notes for the usage of the DOS and @CHIP-RTOS int API calls

1. At the start of a user program the Stdio focus should be set to USER. Before ending the application switch the focus back to SHELL or BOTH (see **Set Stdio focus**).
2. If more than one user program runs in the IPC@CHIP, only one of them should read characters from **Stdin**
3. The functionality of most of the shell commands is also available through calls into the **@CHIP-RTOS INT API**. If not, use the @CHIP-RTOS INT call **Execute a shell command**. This call executes a shell command from inside the user application.
4. Install a fatal user error handler, which does a reboot of the IPC@CHIP with **Install user fatal error handler**
5. Used software interrupts (all others are free for use):

   `0x00` - Reserved (@CHIP-RTOS Divide Overflow Handler)
   `0x01` - Reserved (Debugger Trace Interrupt)
   `0x02` - Reserved (Hardware Non-Maskable Interrupt (NMI))
   `0x03` - Reserved (Debugger Breakpoint Interrupt)
   `0x04` - Reserved (@CHIP-RTOS INTO Overflow Handler)
   `0x05` - Reserved (@CHIP-RTOS Array Bounds Exception Handler)

0x06 - Reserved (@CHIP-RTOS Invalid Opcode Exception Handler)
0x07 - Reserved (@CHIP-RTOS ESC Opcode Exception Handler)
0x08 - Reserved (Hardware, Timer #0 Handler)
0x0A - Reserved (Hardware, DMA #0 / INT5 Handler)
0x0B - Reserved (Hardware, DMA #1 / INT6 Handler)
0x0C - Reserved (Hardware, INT0 Handler)
0x0D - Reserved (Hardware, INT1 Ethernet Handler)
0x0E - Reserved (Hardware, INT2 Handler)
0x0F - Reserved (Hardware, INT3 Handler)
0x10 - **Biosint**
0x11 - **Biosint**
0x12 - Reserved (Hardware, Timer #1 Handler)
0x13 - Reserved (Hardware, Timer #2 Handler)
0x14 - **Fossil Interface**
0x16 - **Biosint**
0x1A - **Biosint**
0x1C - Timer Interrupt, see **Set timer 1C interval**
0x20 - Terminate Program (Only for compatibility, instead use **DOS service 0x4C**)
0x21 - **DOSEmu** Interrupt Interface
0xA0 - **Several 'chip' related services**
0xA1 - **Hardware API** (HAL)
0xA2 - **Hardware API** (PFE)
0xAA - **I2C Interface**
0xAB - **CGI Interface**
0xAC - **TCP/IP API**
0xAD - **RTOS API**
0xAE - **Ethernet Packet Driver**
0xAF - Timer Interrupt, see **Set timer AF interval**
0xB0 - **External Disk API**
0xB1 - **External Disk Driver**
0xBF - This vector is reserved to start a DOS executable

If you are using a BIOS variant in which some modules are not included, then the interrupts corresponding to these modules are free for use.

6. External hardware interrupts are enabled (STI opcode) during execution of the following API software interrupts:
   `DOS ints 0x10, 0x14(Fossil), 0x16, 0x20, 0x21, @CHIP-RTOS int 0xA0, CGI int 0xAB, PFE int 0xA2, I2C int 0xAA, CGI int 0xAB, Pkt int 0xAE, Extdisk int 0xB0, Extdisk user int 0xB1`

   The state of the interrupt flag is not changed during the execution on the following API software interrupts (If interrupts are disabled before the API call, they are kept disabled during execution of the call. If interrupts are enabled before the API call, they get reenabled during execution of the call.):
   `TCPIP int 0xAC, RTOS int 0xAD`

   External hardware interrupts are disabled during execution of the following API software interrupts:
   `HAL int 0xA1`

BIOS ints
Int21h

Using Hardware API

1. The HAL functions keep interrupts disabled, so you can call them inside an interrupt routine. The PFE functions are only for choosing and initializing a specific function on the selected pin. They should be called once in your application for initializing your hardware environment and not at runtime or inside interrupt routines.

2. Do not use functions of the RTOS API inside of a normal HW API user ISR. For this purpose install a RTX user ISR, see **Install ISR**.

3. The latency time of the user ISR (from generation of an interrupt until first line of code inside the user ISR) depends on the used IPC@CHIP, see **Performance comparision** .

4. Instead of HAL functions **Read data bus** and **Write data bus** you can call C-functions `inportb` and `outportb` from DOS.H for faster data bus accesses.

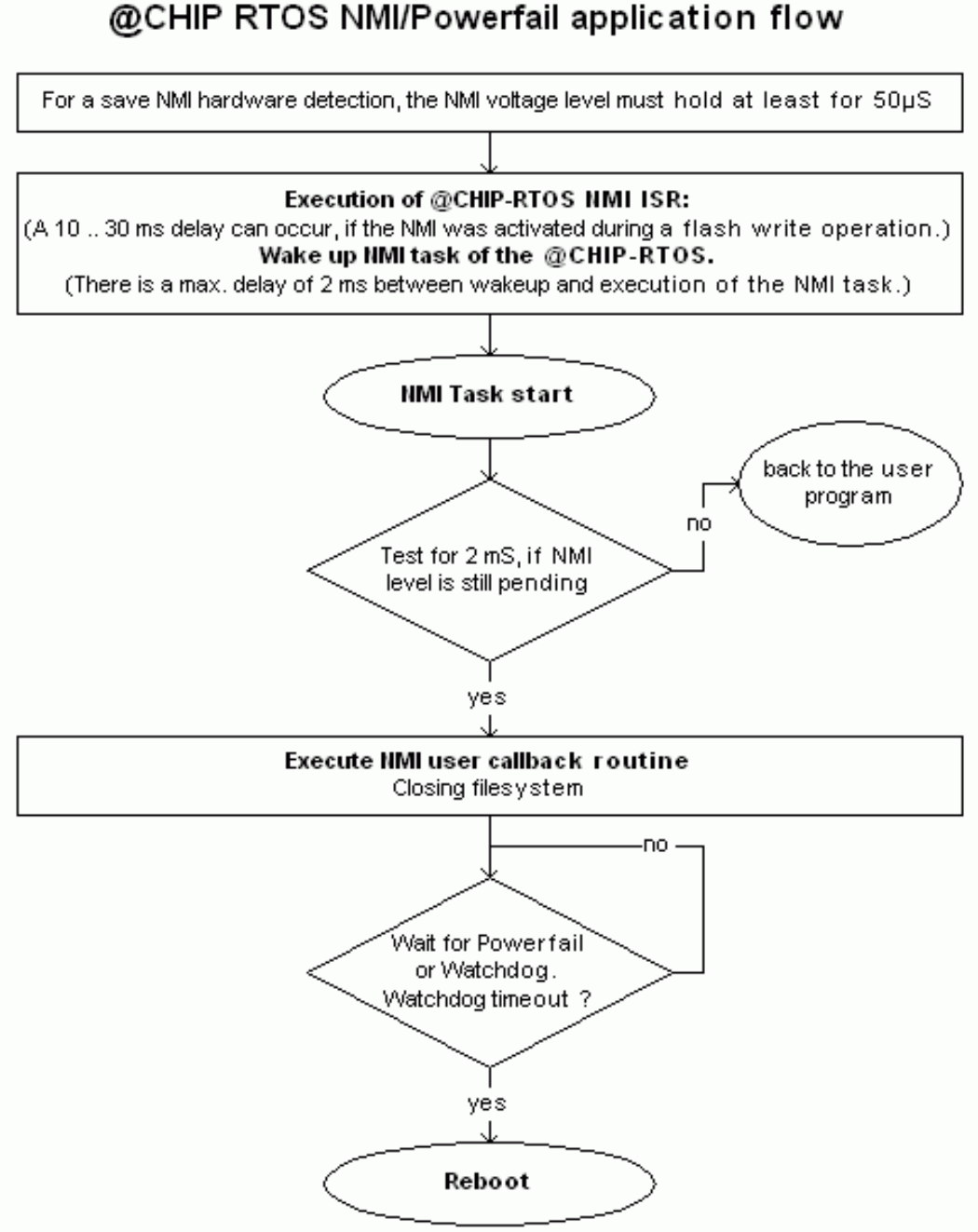
5. **Usage of DMA0/INT5 and DMA1/INT6:**
   DMA0 and INT5 are served inside the @CHIP-RTOS with a single shared interrupt handler. (Likewise for DMA1 and INT6.)
   If an user application has activated **serial EXT DMA mode** and also wants to use external interrupt INT5, the DMA interrupt event is higher priority. (And again, likewise for COM DMA mode and INT6.)
   This means that if a DMA0/1 interrupt occurs simultaneous with an external INT5/6 interrupt, the installed user handler for the external interrupt is not called.

   See **Hardware API**

6. Usage of NMI/Power fail detection:

   The NMI function of the multifunction pin 17 (RESET/NMI/LINK_LED) of the IPC@CHIP SC11/SC12/SC13 is for power fail purposes only. It is not possible to use NMI as a "normal" interrupt pin like e.g. INT0 for generating interrupts. It can only be used as described in the IPC@CHIP hardware documentation.

   The flowchart below describes how the @CHIP-RTOS handles an incoming NMI interrupt.



See also:
Install interrupt service routine
Init non-volatile data
Save non-volatile data

Top of list
Index page

Using Fossil API

1. The **receive and send queue size** can be configured over the CHIP.INI.
2. If you want to use external DMA, you have to disable the **serial DMA receive mode**. Otherwise the DMA receive mode is recommended.

3. Since @CHIP-RTOS 1.02B XON/XOFF mode is available also with the serial DMA receive mode in use.
   Please note: Because of the internal functionality of DMA it is not possible to immediately detect an XON or XOFF character from the connected peer. It is possible that an overrun situation at the peer (e.g. GSM modem) can occur. Nevertheless we enable this mode because some GSM modems (any??) support only XON/XOFF as serial flow control mechanism.

4. The default serial receiver queue size is 1024 bytes. If the default DMA receive mode is used, it is advised to increase the receiver queue size in **Chip.ini** up to a minimum value of 2048 bytes to prevent a possible buffer overrun (even if hard handshake is used). This can only happen with the default queue size of 1024 bytes if the user doesn't call the **Fossil API read block function** frequently enough. If the application programmer does not increase this buffer size up to the recommended value, they should call the **Fossil API read block function** with sufficient buffer size in the CX-Register to flush the internal buffers and prevent a receive buffer overrun.

5. **Serial ports, running with IRQ receive mode:**
   The two serial ports of the IPC@CHIP have no hardware FIFO buffer. Only one incoming character can be stored direct by the ports. As a consequence of this behaviour, it is possible that incoming characters get lost because of missed or delayed receiver interrupt execution (see **Operating mode of serial ports**). E.g.: Writing a flash sector (e.g. happens when writing a file at Drive A:) disables all interrupt execution for about 15 ms. If the serial port is configured with a baud rate of 9600, incoming characters can be lost during this time.
   Loss or delayed execution of serial receiver interrupts depends on the number and the execution frequency of all enabled interrupts in the IPC@CHIP system. It depends also on execution times of users interrupt service functions and the duration of interrupt masking periods (CLI / STI instruction sequences).

6. For a given serial port the fossil functions are not reentrant. Do not call fossil functions for the same serial port from different tasks. However for different serial ports, the fossil functions are reentrant. E.g. task A can operate the COM port using fossil functions concurrently with task B operating the EXT port using the same fossil functions.

   Fossil API

Top of list
Index page

Working with Float Data Types

1. The IPC@CHIP does not provide a floating point co-processor. So if you want to use floating point data types you need to enable the math-emulation in your compiler. In Borland C++ 5.02 see the option "Emulation" under the Target Expert's (right mouse click on your Exe-file in your project) "Math Support".

2. The Borland Math Emulation libraries are not made for usage in a multitasking system like the @CHIP-RTOS. If you want to use float data types in tasks other than your main (DOS) task, please pay attention to the following work around solution (Using Borland 5.02, Memory model large):

   a) For floating point emulator usage, the current stack (the stack of the task) must have offset 0. To achieve this, the task stack must be far array located in a separate segment:

   ```
   unsigned int far task_stack[TASK_STACKSIZE];
   ```

   Note: The CGI callbacks execute on the Web server's stack, which satisifies this "offset 0 present" requirement.

b) The current stack must be pre-initialized for math emulation:

```
void my_emu1st(void)
{
    asm{
        mov word ptr ss:[0x2f],0x0065 // set end of FPU stack
        mov word ptr ss:[0x31],0x0125 // set start of FPU stack
        mov word ptr ss:[0x27],0 // no protected mode
        mov byte ptr ss:[0x26],0 // no 8087
    }
}
```

c) Before executing floating point arithmetic, call the Borland library function `_fpreset`, which initializes the stack for floating point emulation.

Example for a task function:

```
void huge my_task(void)
{
    float a;
    my_emu1st();
    _fpreset();
    // ..... ready for executing floating point arithmetic
}
```

If you are interest in more details, take a look at the Borland 5.02 RTL sources (`fpinit.asm`). We will provide an example for using floating point arithmetic in separate tasks or within CGI callbacks.

3. Using Math emulation with Borland C++ 5.02 (also for older versions of Borland C) produces a conflict between the NMI interrupt of the @CHIP-RTOS (Interrupt Vector 0x02) and the Borland floating exception handling. Borland C math emulation also uses `INT 2` for generating floating point exceptions. Our internal NMI interrupt service function (normally called at power fail case) will no longer execute during the runtime of the user's application. We have provided a solution which moves the Borland exception function from INT 2 to another interrupt vector. We modified the Borland RTL source (`fpinit.asm`) and include the modifications directly into the application project. It is planned to provide an example in our Internet download area.

Top of list
Index page

Configure PPP client or server

1. Connected modems should be configured in chip.ini with the modem command ATE0 **INITCMD**. This prevents the modem from echoing characters to the peer in command mode. The COM and the serial ports of the IPC@CHIP have only 4 lines (TxD/RxD/CTS/RTS) and no line for detecting a hang-up of the modem. Because of this fact we provided the **Idletimeout** and the **MODEMCTRL** configuration features in chip.ini or in the PPP client init structure **type**. But the `idletimeout` and `modemctrl` detection can fail if a modem has switched its echo mode on. If the peer modem hangs up without correctly closing a PPP session, the IPC@chip modem also hangs up and goes into the command mode. Because of the missing line for detecting a modem hang-up, the PPP server doesn't know anything about the broken connection and still sends PPP frames to the modem. It can happen that the modem echoes these characters back to the IPC@CHIP due to being in "Echo on" mode. This will cause the `idletimeout` to not work.

2. We recommend that the PPP server or client and the connected modems should run (if possible) with RTS/CTS flow control (see chip.ini **flow control mode** or the PPP client init structure **type**).

Most modems use RTS/CTS flow control, if they get the AT command AT\Q3.

3. The COM and EXT port of the IPC@CHIP has only CTS, RTS, RxD and TxD lines, so you have to configure your modem with DTR always on. (e.g. AT cmd for a most modem types: AT&D0)

4. If the option PPP_IPCP_PROTOCOL (VJ TCP/IP header compression) is set (see **PPPCLIENT_SET_OPTIONS**), it is possible that the FTP server of the IPC@CHIP is not usable via the PPP interface. This was noticed at PPP sessions connected to a Linux PPP peer.

PPP

Top of list
Index page

End of document

Important information about the IPC@CHIP drives and filesystem (Limited Number of Flash write cycles per sector)

The sectors of the internal Flash of the IPC@CHIP (used for drive A: and memory for @CHIP-RTOS Code) has limited write cycles. So you should avoid cyclic write access like writing into a logfile every hour.
We guarantee 10.000 write cycles per sector. Note that one write access in your program (fwrite or write) could cause more than one write access on the @CHIPs flashdisc.

Optimizations:

To decrease the flash write accesses in your program, you should follow the hints below:
1. Use fopen, fwrite, ...
2. Use setvbuf with a size of 1024 bytes.

Description:

fopen internally works with a 512 byte stream buffer. The filesystem sectors have a size of 512 bytes. Flash physical sector size can be 256 or 1024 bytes, depending on the used flash memory. Setting the stream buffered size to 1024 helps to avoid additional flash writes. Note: The fwrite function in Borland C 5.02 has an unexpected behaviour when the size of data exceeds the buffered size (512 bytes default or size set with setvbuf). Then the data blocks exeeceding the buffered size are always written unbuffered instead of splitting them to smaller packets.

How will a flash defect detected:

The damaging write access respectivly the return code tells you that the write access was not successful. That it is a flash defect can you see if you reboot your @CHIP. There you get an error message ("Flash error at sector: xx") on the boot text. Also the execution of the AUTOEXEC.BAT will be avoid.

Workaround:

If you need to generate logfile anyway, you should use the RAM Disc to store the file. To save the file when the power goes down, you could use the NMI feature (more about NMI is available in the API Docu -> Hardware API).
Another option is to use an IDE Drive (Harddisc, Compactflash, ..) to store the log file. How to connect an IDE Drive to the IPC@CHIP is explained in the API Docu.

End of document

COMMAND

Command processor
Interprets the commands in `autoexec.bat` and those issued at the console.

New in version 1.10B: **Modified IPCFG command**
New in version 1.10B: **Display list of all detected errors.**

- FORMAT A: [/C:n] [/E] [/R:n]
- VER
- MEM
- CGISTAT
- CLOSETELNET
- WEBSTAT
- PING
- TASKS
- UTASKS
- HELP
- ERRORS

In the `autoexec.bat` or any other batch file you can only list the internal commands and the names of any program files on the flash drive. A batch file must have the file extension `.bat`, e.g. `test.bat`.

The commands are executed sequentially, with one difference to the 'normal' DOS: When the IPC@CHIP's **BATCHMODE** is configured for concurrent batch file execution, the next command is executed before a previous command has finished. The only exceptions are the WAIT and REBOOT commands.

Please note that very little syntax checking is done.

DEL filename
Delete a file

Delete a file or all files that match the wildcard.

Example

```
del *.dat
```

Top of list
Index page

DIR filename
List a directory.

List the directory entry or all entries that match the wildcard.

Comments

If no argument is given, *.* is assumed.

Example

```
dir *.exe
```

Top of list

Index page

TYPE file
Type a file.

Show contents of a file on the console.

Top of list
Index page

COPY file1 file2
Copy a file.

Copy a file. The two file specifiers must be complete file names.
Wildcards such as * or ? are not allowed.

Top of list
Index page

REN file1 file2
Rename a file.

Rename a file. The two file specifiers must be complete file names.
Wildcards such as * or ? are not allowed.
Both files must reside in the same directory.

Top of list
Index page

MD dir
Creates a directory.

Creates a directory.

Example

```
md temp
```

Top of list
Index page

XTRANS
File transfer: Send/Receive file with Xmodem.

Send/Receive file with XMODEM/CRC protocol. Possible devices are COM or EXT.

Example

```
XTRANS COM R chip.ini  ;Receive chip.ini file over COM
XTRANS EXT S test.txt  ;Send test.txt file over EXT
```

Top of list
Index page

MEMOPT 0/1
Disable or enable memory optimize when loading exe file.

Use MEMOPT 1 to optimize memory usage when loading an exe file.

By default, the memory optimization is disabled. An exe file will obtain almost all memory available at startup. In the startup code, the program will then resize this memory.

When enabled, the program will obtain only the memory it defines as required in the header of the exe file. This can leave more memory to other programs, but it can result in errors when allocating memory from the heap.

Users of Borland C/C++ will probably not need this command. Only users of Borland Pascal might need it since programs written in Pascal usually do not resize their memory at startup.

Comments

With SC12 @CHIP-RTOS version 0.67, the default for MEMOPT is disabled. Earlier versions had this feature enabled but this resulted in errors with `malloc()`.

Example

```
MEMOPT 0
```

Top of list
Index page

CD dir
Change the current working directory.

Changes the current working directory.

Example

```
cd temp
```

Top of list
Index page

RD dir
Removes a directory.

Removes a directory. This command cannot be executed on directories containing data.

Example

```
rd temp
```

Top of list
Index page

CON
Direct console I/O.

Define what device is used as the console. Possible devices are COM, EXT or TELNET. Multiple devices are possible.

Comments

This setting is only valid until the next reboot.

Example

```
CON COM
CON EXT
CON COM TELNET
```

Related Topics

Default input device **STDIN** initial value
Default output device **STDOUT** initial value

Top of list
Index page

IW
Input word.

Perform a 16 bit input from a given I/O address.
The address and the result are hexadecimal.

Example

```
IW 600
```

Top of list
Index page

IB
Input byte.

Perform an 8 bit input from a given I/O address.
The address and the result are hexadecimal.

Example

```
IB 600
```

Top of list
Index page

OW
Output word.

Perform a 16 bit output on a given I/O address with given data.
The address and the data are hexadecimal. The address is the first parameter followed by data.

Example

```
OW 600 F
```

Top of list
Index page

OB
Output byte.

Perform an 8 bit output on a given I/O address with given data.
The address and the data are hexadecimal. The address is the first parameter followed by data.

Example

```
OB 600 F
```

Top of list
Index page

PCS
Enable chip select.

Enables a chip select line.
The command expects only one parameter: the chip select line. Valid arguments are 0, 1, 2, 3, 5 or 6.

Example

```
PCS 6
```

Top of list
Index page

ALE
Enable/disable ALE pin.

Enables the address latch enable (ALE) pin.
The command expects only one parameter: 1 enable / 0 disable.

Example

```
ALE 1
```

Top of list
Index page

ADR
Enable non-multiplexed address bus pins.

Enables the non-multiplexed address bus pins (A0/A1/A2).
The command expects only one parameter: 0=enable A0 / 1=enable A1 / 2=enable A2

Example

```
ADR 0
```

Top of list
Index page

PIO
Enable and show PIO pins.

Enables the programmable PIO pins (PIO0-13).
The command expects two parameters: PIO MODE

PIO: PIO number (0-13)
MODE: PIO mode
- 1 = Input without pullup/pulldown
- 2 = Input with pullup (not PIO13)
- 3 = Input with pulldown (only for PIO3 and PIO13)
- 4 = Output value = High
- 5 = Output value = Low

When no command line argument is given, the PIO state is shown.

Example

```
PIO 3 5  = PIO3 Output low
PIO      = Shows PIO states
```

Top of list
Index page

IP address
Sets the IP address of the Ethernet interface.

Sets the IP address of this device of the internal Ethernet interface.

Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new address is not used until after a **IPETH** command or a restart of the system.

Use the **IPCFG** command to verify your entry before restarting the system.

Example

```
IP 195.243.140.85
```

Related Topics

IP **address** initial value
Set IP Address **API** function
PPP server initial IP address
Initial **DHCP** setting

Top of list
Index page

NETMASK mask
Set the network mask for IP addressing of the Ethernet interface.

Sets the subnet mask for IP addressing of the internal Ethernet interface.

Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new subnet mask is not used until after a **IPETH** command or a restart of the system.

Use the **IPCFG** command to verify your entry before restarting the system.

Example

```
NETMASK 255.255.255.192
```

Related Topics

IP **subnet mask** initial value
Initial **DHCP** setting
Set IP subnet mask **API** function

Top of list
Index page

GATEWAY address
Define the IP address of the gateway

Sets the IP address of the default gateway to use.

Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new gateway address is not used until after a **IPETH** command or a restart of the system.

Use the **IPCFG** command to verify your entry before restarting the system.

Example

```
GATEWAY 195.243.140.1
```

Related Topics

IP **GATEWAY** initial value
Set gateway IP address **API** function
ADD_DEFAULT_GATEWAY API function
Initial **DHCP** setting

Top of list
Index page

DHCP 0/1
Enable/Disable DHCP.

Enables or disables the use of DHCP for the internal Ethernet interface to obtain an IP configuration.

Comments

DHCP is an abbreviation for "Dynamic Host Configuration Protocol".
Using a DHCP Server, the network administrator can define the IP configuration of the network, without manually configuring each device on the network.
Network servers and some ISDN routers offer a DHCP server.

Example

```
dhcp 1
```

Related Topics

Initial **DHCP** setting

Top of list
Index page

IPETH
Restart the Ethernet interface

Restart the internal Ethernet interface, e.g. after changing the IP configuration, without rebooting the system.

Comments

If the restart command prints an error message, check your IP parameters.
In most cases an invalid gateway IP address is the reason why the restart failed.
The error code 237 signals that a Ethernet configuration was already in progress

Top of list
Index page

TCPIPMEM
Display TCP/IP memory usage

Displays TCP/IP memory usage. This command shows the maximum reserved memory for the TCP/IP stack and the current TCP/IP stack memory used.

Top of list
Index page

BATCHMODE
Set batch file execution mode

Sets the batch file execution mode of DOS programs for either concurrent or sequential execution.
See `BATCHMODE` initialization **documentation** for details.

Example

```
BATCHMODE 1  ;  Selects sequential batch file processing mode

BATCHMODE 0  ;  Selects concurrent batch file processing mode
```

Related Topics

Initial **BATCHMODE** setting
Run-time **batch mode** selection API

Top of list
Index page

FTP 0/1
Enable/Disable FTP.

Enables or disables the start of the FTP server after a reboot.
The file `chip.ini` contains the new value for FTP enable to be applied after rebooting the system.

Example

```
FTP 1
```

Related Topics

Initial **FTP** ENABLE setting

Top of list
Index page

TFTP
Enable/Disable TFTP

Enables or disables file transfers via TFTP server.
0 disables the server, 1 enables TFTP file transfer.

Comments

By default the TFTP server is disabled to avoid security leaks.

Example

```
TFTP 1
```

Top of list
Index page

IPCFG
Display current IP configuration of all installed TCP/IP device interfaces.

Displays for each installed TCP/IP device interface:

Interface name
Type: ETH(Ethernet), LPK(Internal loopback), PPP or Unknown
Internal index number of the device interface
IP address

Network mask
MAC address
Default gateway

Example

```
IPCFG
```

Top of list
Index page

REBOOT
Restart the system

Restarts the system.
First, the file system is closed, then the watchdog is configured to issue a reset.
Please note that the tasks are not informed of this restart !

Example

```
reboot
```

Top of list
Index page

WAIT secs
Suspends the command interpreter.

Suspends execution of the command interpreter for the specified interval.
The time interval is defined in seconds.

Example

```
wait 1
```

Top of list
Index page

FORMAT A: [/C:n] [/E] [/R:n]
Format Flash disk A:.

Format flash disk A:.

All information on drive A: will be lost !

The cluster size parameter /C: is optional, a value of 2 is default on A:, value 4 on B:.

If the /E parameter is specified, the data area will be filled with null-data.

With parameter /R you can select the number of root directory entries. Note: This must be a multiple of 16.

Comments

Make sure that other tasks do not access drive A: when formatting.

Important : If you use retentive operators, only format flash disk with default cluster size!!

Example

```
FORMAT A: /C:2 /E
FORMAT B: /C:4
FORMAT B: /R:256
```

Top of list
Index page

VER
@CHIP-RTOS Version.

Output the IPC@CHIP serial number, @CHIP-RTOS version and build date.

Top of list
Index page

MEM
Display memory map.

Displays a memory map, including the name of the task owning the memory.

Comments

The size indicated is the actual usable size.
One sector (16 bytes) is added for memory management.

Top of list
Index page

CGISTAT
List Installed CGI handlers.

This function will list all installed CGI handlers.

Related Topics

CGI_INSTALL API function

Top of list
Index page

CLOSETELNET
Closing current telnet session.

This function will finish the current Telnet session.

Top of list
Index page

WEBSTAT
Show the current settings of the Web server

This function will show the current settings of the Web server:
e.g. root directory, root drive,...., default start page.

See WEB **config** for the available chip.ini entries for the Web server.

Top of list
Index page

PING
The ICMP echo request (ping)

Test the network connection with the ICMP command ping.

This command sends 4 ICMP echo requests (64 Bytes) to the remote host,
with an interval of 1 second and shows the results.

Example

```
PING 192.168.200.10<nl>
```

Related Topics

PING_OPEN API function

Top of list
Index page

TASKS
Display list of tasks.

Displays a lists of all tasks, including the CPU load caused by the task, the task status and the stack space usage.

Sample output:

```
task 1094 count 3515 MTSK prio= 12 stack=3000 used=35% state=0
task 1606 count 81 ETH0 prio= 5 stack=2048 used=41% state=4
task 256 count 4568 AMXK prio= 0
task 2374 count 1072 WEBS prio= 41 stack=2048 used=24% state=81
```

```
task 2886 count 100 DOS1 prio= 25 stack=128 used=78% state=81
task 3142 count 157 DOS2 prio= 25 stack=128 used=78% state=81
task 1862 count 24 CFGS prio= 7 stack=1400 used=28% state=4
```

At every one millisecond clock tick, the count for the active task is increased by one. After 10 seconds, the counters are copied and reset to zero.

Comments

At the first call of TASKS, the timer interrupt routine of the RTOS is exchanged by a version for the task monitor. Only after 10 seconds will the TASKS command return usable results.

The command shows for DOS applications only a task stack size of 128 Byte, since the DOS program at run time switches to its own internal stack which is is not visible to the Kernel.

A maximum of 35 tasks can by monitored.

Please be aware that using TASKS has a performance penalty. Use **UTASKS** command to shut off the task monitoring.

The listed task state is only a one moment snapshot. The task state bit field is a 16 bit hexadecimal value defined as follows:

Bit0	timer wait (used with other bits)
Bit1	trigger wait (i.e. idle)
Bit2	semaphore wait
Bit3	event group wait
Bit4	message exchange wait
Bit5	message send wait
Bit6	suspended (waiting for resume)
Bit7	waiting for wake
Bit8-15	internal use only

Current running system tasks (if not disabled in `chip.ini`)

```
Very high priority:
   AMXK       prio= 00      Kernel task
   ETH0       prio= 05      Ethernet receiver task
Normal:
   PPPS       prio= 06      PPP server
   TCPT       prio= 06      TCP/IP timer task
   CFGS       prio= 07      UDP config server
   TELN       prio= 11      Telnet server
   MTSK       prio= 12      Console task (command shell)
Low priority:
   WEBS       prio= 41      Web server
   FTPS       prio= 41      FTP server
```

Top of list
Index page

UTASKS
Disables the Task Monitor.

Disables the Task Monitor which was installed using **TASKS** command.

If you do not need the Task Monitor anymore, you should disable it using this command because the Task Monitor has a performance penalty.

Top of list
Index page

HELP
Display list of all console commands.

Displays a list of all available console commands.

Top of list
Index page

ERRORS
Display list of all detected errors.

Displays a list of all detected errors.

Top of list
Index page

End of document

CHIP.INI Documentation - SC12 @CHIP-RTOS V1.10

CONFIG

The IPC@CHIP system configuration is controlled via the CHIP.INI file.
At startup, the system reads the file `A:\chip.ini` and uses the settings found here to initialize the system.

CONFIG News

- PPPSERVER_MODEMCTRL
- PPPSERVER_CTRLTIME
- PPPSERVER_CTRLCMDx
- PPPSERVER_CTRLANSWERx
- PPPSERVER_CTRLTIMEOUTx
- PPPSERVER_CTRLRETRIESx
- PPPSERVER_CMDMODE
- PPPSERVER_HANGUPDELAY
- PPPSERVER_HANGUPCMDx
- PPPSERVER_HANGUPANSWERx
- PPPSERVER_HANGUPTIMEOUTx
- PPPSERVER_HANGUPRETRIESx
- PPPSERVER_CONNECTMSGx
- PPPSERVER_CONNECTANSWERx
- PPPSERVER_CONNECTTIMEOUTx
- RAMDRIVE_SIZE
- TIMER_1C
- TIMER_AF
- FTP_ENABLE
- FTP_CMDPORT
- FTP_LOGINDELAY
- FTP_TIMEOUT
- FTP_USERx
- FTP_PASSWORDx
- FTP_ACCESSRIGHTx
- FTP_DRIVEx
- FTP_ROOTDIRx
- WEB_ENABLE
- WEB_MAINPAGE
- WEB_TEMPPATH
- WEB_DRIVE
- WEB_ROOTDIR
- WEB_MAXCGIENTRIES
- WEB_WEBSERVERSTACK
- WEB_HTTPPORT
- WEB_USER0
- WEB_PASSWORD0
- WEB_SECURE
- WEB_SEC_URLx
- WEB_SEC_USERx
- WEB_SEC_PASSWORDx
- TFTP_TFTPPORT
- TELNET_TELNETPORT
- TELNET_TIMEOUT
- TELNET_LOGINDELAY
- TELNET_LOGINRETRIES
- TELNET_USERx
- TELNET_PASSWORDx
- TELNET_ENABLE
- DEVICE_FILESHARING
- DEVICE_NAME
- TRACE_FLASHWRITE
- TRACE_INTNOTSUPP

- SERIAL_EXT_DMA
- SERIAL_COM_DMA
- SERIAL_SEND_DMA
- SERIAL_EXT_RECVQUEUE
- SERIAL_EXT_SENDQUEUE
- SERIAL_COM_RECVQUEUE
- SERIAL_COM_SENDQUEUE
- SERIAL_COM_BAUD
- SERIAL_EXT_BAUD
- DOSLOADER_MEMGAP
- BATCH_BATCHMODE
- BATCH_EXECTIMEOUT

STDIO

[STDIO]
STDIN=Define standard input device

Define your device for standard input.
Valid devices are COM, EXT and TELNET. You can define several devices simultaneously.

Comments

The following example defines both COM and TELNET for stdin:

```
[STDIO]
STDIN=COM TELNET
```

By default, both COM and TELNET are used.

Top of list
Index page

STDIO

[STDIO]
STDOUT=Define standard output device

Define your device for standard output.
Valid devices are COM, EXT and TELNET. You can define several devices simultaneously.

Comments

The following example defines both COM and TELNET for stdout:

```
[STDIO]
STDOUT=COM TELNET
```

By default, both COM and TELNET are used.

Top of list
Index page

STDIO

[STDIO]
FOCUS=Command shell and/or user executables

Set the stdio focus to the command shell and/or to the user executables.

Valid entries are USER or SHELL
.
If only USER is defined, stdio in the command shell is suppressed.
If only SHELL is defined, stdout and stdin in the user's DOS executables are disabled.

Comments

The following example enables stdio for both USER and SHELL:

```
[STDIO]
FOCUS=SHELL USER
```

By default, stdin and stdout for both SHELL and USER are enabled.

Important : If stdio is enabled for both, there is a rivalry between USER and SHELL.

At runtime, pressing of the focus key (default is Ctrl-F) toggles between these three modes and shows the current mode.

Related Topics

FOCUS KEY configuration

Top of list
Index page

STDIO

[STDIO]
FOCUSKEY=Key

Defines the key that switches the stdio focus.

Comments

The following example sets Ctrl-F (ASCII 6) as the current stdio focus key:

```
[STDIO]
FOCUSKEY=6
```

By default, the focus key is set to CTRL-F (ASCII 6)
At runtime pressing Ctrl-F keys on the console will then cycle the stdio between the three modes:

Stdio: User
Stdio: Shell
Stdio: Both

The new mode is shown on the console.

Key Range: 0..254

If the key is set to zero, defines no focus key and the switching of stdio is disabled.
The focus key is not usable by the command shell or DOS executable.

Note:
The focus key code is filtered out by the system, and will not be visible to either the command shell or a DOS executable.

Related Topics

Initial **FOCUS** configuration

Top of list
Index page

STDIO

[STDIO]
CTRL_C=0/1

Disable/enable termination of the autoexec.bat execution via ctrl-c key.
The following example disables the ctrl-c control.

```
[STDIO]
CTRL_C=0
```

By default, CTRL_C is enabled.

Top of list
Index page

IP

[IP]
ADDRESS=IP Address of the Ethernet interface

Defines the IP address of the internal Ethernet interface, if no DHCP is used.

Comments

Only numerical IP addresses are allowed here.

Example: `ADDRESS=192.168.200.1`

If no address entry was found, the IP address will be set to `1.1.1.1.`

Related Topics

IP command line
Set IP Address **API** function

Top of list
Index page

IP

[IP]
NETMASK=IP Address mask of the Ethernet interface

Defines the subnet mask of the internal Ethernet interface of IPC@CHIP, if no DHCP is used.

Comments

Example: `NETMASK=255.255.255.224`

If no subnet mask entry was found, the subnet mask will be set to `255.255.255.0`.

Related Topics

NETMASK command line
Set IP subnet mask **API** function

Top of list
Index page

IP

[IP]
GATEWAY=Gateway IP Address

Setting the default gateway.

Comments

Example: `GATEWAY=195.243.140.65`

The TCP/IP stack of the IPC@CHIP supports only one valid default gateway for all device interfaces: Ethernet and PPP Interface.
So see also **PPP Server GATEWAY** if you are using PPP.

We provide some additional API functions for modifying the default gateway:

- Interrupt 0xAC, **Service 0x80**: add a default gateway
- Interrupt 0xAC, **Service 0x81**: delete default gateway
- Interrupt 0xAC, **Service 0x82**: get default gateway

Related Topics

CHIP.INI entry **PPPSERVER GATEWAY**
GATEWAY command line
Set IP gateway **API** function

ADD_DEFAULT_GATEWAY API function

Top of list
Index page

IP

[IP]
DHCP=0/1 Ethernet interface

Set to 1 if DHCP client should be used to get the IP configuration for the internal Ethernet interface from a DHCP server.
If defined as 0, a static network configuration is used.

Comments

Any settings for IP Address, subnet mask and gateway are ignored if DHCP is used.

Related Topics

DHCP command line

Top of list
Index page

IP

[IP]
HOSTNAME_OPT=0/1 DHCP hostname option

Set to 1 if DHCP client of the IPC@CHIP should append the device name at the DHCP request option field with DHCP option number 0x0C.
If set to 0 (default), the device name will not be appended to the DHCP request.

Comments

This name feature can be used for the configuration of a DHCP server, e.g. to give a fixed IP to an IPC@CHIP which is configured with a device name.

Top of list
Index page

IP

[IP]
TCPIPMEM=Size

Set the size of the TCP/IP memory block in kBytes. This block is allocated at the start of the TCP/IP stack.

Valid Range: Between 30 kBytes and 160 kBytes (An out of range value for TCPIPMEM will be set to closest of these limit values.)

Default value: 90 kBytes when @CHIP-RTOS configured without PPP capability (server or client)
98 kBytes when @CHIP-RTOS configured with PPP capability

Example: `TCPIPMEM=60`

Comments

The TCP/IP API function 0x78, **GET_MEMORY_INFO**, reports the current used memory of the TCP/IP stack
Since @CHIP-RTOS version 1.02B we allow configuring a maximum value of 160kBytes, because some application programmers may require more then the old limit of 132 kByte.

Top of list
Index page

UDPCFG

[UDPCFG]
LEVEL=mask

Defines the supported functions of the configuration server.

Set LEVEL as a bit mask to define the functions that the configuration server should listen to.
The bit assignments are as follows:
BIT0: Allow detection on the network.
BIT1: Allow change of IP configuration.
BIT4: Allow programming of flash.
If defined as 0, the configuration server task will not start.

Example: `LEVEL=0x03`
This would allow detection on the network and changing the IP configuration, but no @CHIP-RTOS update.

Comments

By default, all options are enabled (==0x13).

Related Topics

Run-time adjustments to **LEVEL**

Top of list
Index page

PPPCLIENT

[PPPCLIENT]
ENABLE=0/1

Disable/enable PPP client task

```
[PPPCLIENT]
ENABLE=1
```

By default, PPP client is enabled.

Top of list
Index page

PPPSERVER

[PPPSERVER]
ENABLE=0/1

Disable/enable PPP server

```
[PPPSERVER]
ENABLE=1
```

By default, PPP server is enabled.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
MODEMTRACE=0/1

Disable/enable the trace of the control communication between IPC@CHIP and a connected modem. The modem strings (AT commands and answers) defined in chip.ini will be printed on STDOUT, if MODEM_TRACE=1.
This can be useful for testing the modem configuration and debugging the PPP dial procedures.

```
[PPPSERVER]
MODEMTRACE=1
```

By default, tracing is disabled.

Comments

Received characters with an ASCII value smaller than 0x20 are printed as numbers.

Top of list
Index page

PPPSERVER

[PPPSERVER]

COMPORT=Define serial device for the PPP server

Define your serial device for the PPP server.
Valid devices are COM or EXT.

Comments

The following example defines EXT as device for the PPP server:

```
[PPPSERVER]
COMPORT=EXT
```

By default, no serial port is enabled.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
ADDRESS=IP Address of the PPP server interface

Defines the IP address for the PPP server.

Comments

Only numerical IP addresses are allowed here.
Example: `ADDRESS=192.168.205.1`
If no address entry was found, the address will be set to `1.1.2.1.`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
REMOTEADDRESS=IP Address for the remote PPP client

Defines the IP address for the remote PPP client.

Comments

Only numerical IP addresses are allowed here.

Example: `REMOTEADDRESS=192.168.205.2`
If no address entry was found, the remote address will be set to `1.1.2.2.`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
NETMASK=IP Address subnet mask of the PPP server

Defines the IP address subnet mask of the PPP server.

Comments

Example: `NETMASK=255.255.255.0`
If no subnet mask entry was found, the subnet mask will be set to `255.255.255.0.`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
GATEWAY=IP Address of gateway

Defines the IP address of the gateway if PPP Connection is established.

Comments

Example: `GATEWAY=195.243.140.65`

If no gateway entry was found, the gateway will be untouched.

The TCP/IP stack of the IPC@CHIP supports only one valid default gateway for all device interfaces: Ethernet, PPP Interface.

If you define a gateway in the PPPSERVER section of the `chip.ini`, it becomes the default gateway for all interfaces when a PPP link to the server is established. The default gateway must be the same IP Address as the remote peer.
It does not make sense to define a different IP, because the remote Peer is the only peer, which is reachable. If a different IP than the remote IP is defined, the remote IP will used for the gateway entry automatically.
During a PPP server connection the command **`ipcfg`** indicates this default gateway. After the PPP session, the old gateway (if any was defined) will be restored.

Also we provide some functions for modifying the default gateway:

- Interrupt 0xAC, **Service 0x80**: add a default gateway
- Interrupt 0xAC, **Service 0x81**: delete default gateway
- Interrupt 0xAC, **Service 0x82**: get default gateway

Related Topics

CHIP.INI entry **IP (Ethernet) GATEWAY**
PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
AUTH=0/1/2

Set PPP authentication mode for the remote PPP client
0: No authentication
1: PAP authentication
2: CHAP authentication

Comments

The following example selects PAP authentication mode:

```
[PPPSERVER]
AUTH=1
```

By default, authentication is disabled.

If AUTH!=0 you must define two **user name** / **password** pairs used to authenticate the PPP client. The client must use one of these pairs to get connected to the IPC@CHIP PPP server.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
IDLETIME=Seconds

Sets the idle time, after which the PPP server closes the connection.

```
[PPPSERVER]
```

```
IDLETIME=500
```

By default, PPP server idle time is 120 seconds. A value of 0 means no idle timeout.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
FLOWCTRL=0/1/2

Set flow control mode of the PPP server's serial device:

0: none
1: XON/XOFF (See caution below!)
2: RTS/CTS

Example: Here XON/XOFF flow control is enabled

```
[PPPSERVER]
FLOWCTRL=1
```

By default, `FLOWCTRL=2` (RTS/CTS)

Comments

Caution:
If you use the default DMA mode for the selected COM port, it is not recommended to choose XON/XOFF flow control mode (see **Fossil API Xon/XOff usage (Chap.3)**).

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
MODEM=0/1

Disable/enable usage of a modem

```
[PPPSERVER]
MODEM=1
```

By default, the usage of a modem is disabled.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
USERx=user

Define the user name for PPP server, using PAP authentication

Comments

You can define a USER0 and a USER1.
Default user is 'ppps' , password is 'ppps' for both USER0 and USER1.
You must specify both the user name and password.
Neither user name nor password are case sensitive.
The entries are only valid if **AUTH**=1 is specified.

Maximum name size: 49 characters

Important notice: To avoid security leaks you must define both user names and passwords.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
PASSWORDx=password

Define the password for a PPP server user, using PAP authentication.

Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1
Default user is 'ppps' , password is 'ppps'.
Neither user name nor password are case sensitive.
The entries are only valid if **AUTH**=1 is specified.

Maximum password size: 49 characters

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
BAUD=BAUD Rate

Sets the BAUD rate of the PPP server serial port.

Comments

The following example sets the PPP server serial port to 19,200 BAUD.

```
[PPPSERVER]
BAUD=19200
```

By default, PPP server BAUD rate is 38400 (with 8 data bits, no parity, 1 stop bit).

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
INITCMDx=modem command

Defines modem commands to initialize your modem connected to the IPC@CHIP
at the start of the IPC@CHIP PPP server and after a modem hang-up following a PPP session.

Comments

You can define a maximum of 3 modem commands e.g. `INITCMD0=ATZ` .
The entries are only valid if **MODEM**=1 is specified.
The maximum length for each command string is 25 characters.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
INITANSWERx=modems answer of init command x

Defines the expected modem answer x for the initialize command x.

Comments

You can define a maximum of 3 modem answers e.g. `INITANSWER0=OK` .
The entries are only valid, if **MODEM**=1 is specified.
The maximum length for each answer string is 80 characters.

Default for all answer strings: OK

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
INITTIMEOUTx=timeout in seconds for wait an the modem's answer

Define the timeout value in seconds for waiting on an answer from the modem.
A value of 0 means wait forever for the modem answer.

Comments

Example: `INITTIMEOUT0=2`
The entries are only valid if **MODEM**=1 is specified.

Default value: 3 seconds

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
INITRETRIESx=Retries, if the modem init answer failed

Define the number of retries used when the modem initial answer fails.

Comments

Example: `INITRETRIES0=2`
This entry is only valid if **MODEM**=1 is specified.

Default value: 1

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER] MODEMCTRL=0/1

Allow modem online control by PPP server.
```
[PPPSERVER]
MODEMCTRL=1
```

By default, the usage of a modem online control is disabled.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER] CTRLTIME=Seconds

Sets the idle interval time, at which the PPP server executes the configured control commands (see **CTRLCMDx**).

```
[PPPSERVER]
CTRLTIME=120
```

By default, PPP server idle control time is 60 seconds.

If the PPP server doesn't receive regular PPP data during this interval, it executes the control commands . If execution of one of the control commands fails, the PPP server then closes the connection.

The CTRLTIME must be a smaller value than the **IDLETIME** of the PPP server.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CTRLCMDx=modem online control command

Defines modem command to control if modem is online or not at the start of the IPC@CHIP PPP server and after a modem hang-up following a PPP session.

Comments

You can define a maximum of 3 modem commands e.g. `CTRLCMD0=+++` .
The maximum length for each control command string is 25 characters.

The entries are only valid, if **MODEMCTRL**=1 is specified.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CTRLANSWERx=modems answer of ctrl command x

Defines the expected modem answer x for the online control command x.

Comments

You can define a maximum of 3 modem answers e.g. `INITANSWER0=OK` .
The entries are only valid, if **MODEMCTRL**=1 is specified.
The maximum length for each answer string is 80 characters.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CTRLTIMEOUTx=timeout in seconds for wait on the modem's answer

Defines the timeout value in seconds for waiting on an answer from the modem.
A value of 0 means wait forever for the modem answer.

Comments

Example: `CTRLTIMEOUT0=2`
The entries are only valid if **MODEMCTRL**=1 is specified.

Default value: 3 seconds

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CTRLRETRIESx=Retries, if the modem online control answer failed

Defines the number of retries used when the modem control answer fails.

Comments

Example: `CTRLRETRIES0=2`
This entry is only valid if **MODEMCTRL**=1 is specified.

Default value: 1

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CMDMODE=switch to modem command mode

Defines the string which switches the modem into the command mode.

Comments

The entries are only valid if **MODEM**=1 is specified.

Default string for CMDMODE:+++

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
HANGUPDELAY=Time in seconds for switching modem into command mode

Defines the time in seconds for switching modem into command mode for hang-up commands.

Comments

The entries are only valid if **MODEM**=1 is specified.

Default time: 2 seconds

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
HANGUPCMDx=modem command

Defines modem commands to hang-up the modem connected to the IPC@CHIP.

Comments

You can define a maximum of 3 modem hang-up commands e.g. `HANGUPCMD0=ATH` ,
which will be executed if the PPP connection is closed.
The maximum length for each command string is 25 characters.

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
HANGUPANSWERx=modems answer for hang-up command x

Defines the expected modem answer x for the hang-up command x

Comments

You can define a maximum of 3 modem answers e.g. `HANGUPANSWER0=OK` .
The maximum length for each answer message is 80 characters.
Default for all answer strings: OK

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
HANGUPTIMEOUTx=timeout in seconds for wait on answer from modem

Defines the timeout value in seconds used when waiting on the modem's answer.
A value of 0 means wait forever.

Comments

Example: `HANGUPTIMEOUT0=2`

Default value: 3 seconds

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
HANGUPRETRIESx=Retries, if the modem hang-up answer failed

Defines the number of retries used if the modem hang-up answer fails.

Comments

Example: `HANGUPRETRIES0=2`

Default value: 1 try

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CONNECTMSGx=modem message

Defines the expected modem message to get connected to a peer modem

Comments

You can define a maximum of three modem messages e.g. `CONNECTMSG0=RING` .
The maximum length for each connect message is 25 characters.

Defaults:
`CONNECTMSG0=RING`
`CONNECTMSG1=CONNECT`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CONNECTANSWERx=modem command for incoming connect message x

Defines the expected modem answer x for the incoming connect message x

Comments

You can define a maximum of three modem answers e.g. `CONNECTANSWER0=ATA` .
The maximum length for each answer string is 80 characters.

Defaults: `CONNECTANSWER0=ATA`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

PPPSERVER

[PPPSERVER]
CONNECTTIMEOUTx=timeout seconds for wait on the modem's connect message x

Defines the timeout value in seconds used when waiting on the modem connect message.
A value of 0 means wait forever.

Comments

Example: `CONNECTTIMEOUT0=0`

Default values:
`CONNECTTIMEOUT0=0`
`CONNECTTIMEOUT1=60`

Related Topics

PPP server **configuration** instructions

Top of list
Index page

RAMDRIVE

[RAMDRIVE]
SIZE=size

Set the size in KByte of the RAM drive C:.
If defined as 0, no RAM drive is configured.
Maximum size is 256 Kbyte

Comments

Default size: 0 (no RAM drive C:)

Top of list
Index page

TIMER

[TIMER]
1C=ms

Sets the interval in milliseconds for timer interrupt 0x1C.
Range: 1 to 32767, Default value=55 ms.

Related Topics

BIOS Ints API **Set timer interrupt 0x1C's interval**

Top of list
Index page

TIMER

[TIMER]
AF=ms

Sets the interval in milliseconds for timer interrupt 0xAF.
Range: 1 to 32767, Default value=4 ms.

Related Topics

BIOS Ints API **Set timer interrupt 0xAF's interval**

Top of list
Index page

FTP

[FTP]
ENABLE=0/1

Define if the FTP server should be activated.

Comments

Use 0 to disable, 1 to enable.
By default, the FTP server is enabled.

Related Topics

FTP enable/disable command
BIOS Ints API Call **Suspend/Resume system servers**

Top of list
Index page

FTP

[FTP]
CMDPORT=port

Set the command port number of the FTP server.
Default FTP command port: 21

Example:

```
[FTP]
CMDPORT=5000
```

Top of list
Index page

FTP

[FTP]
LOGINDELAY=0/1

Define if the delayed login of the FTP server should be (de)activated.

Comments

Use 0 to deactivate, 1 to activate.
By default, the delayed login is enabled.
The delay time starts with 400 milliseconds.
After each following failed login, the delay time will be doubled until it reaches 20 seconds.
After a successful login the delay time will be set back to 400 milliseconds.

Top of list
Index page

FTP

[FTP]
TIMEOUT=sec

Defines the inactivity timeout for the FTP server in seconds.
The minimum value for the timeout is 20 seconds and maximum is 65535 seconds.

Comments

Default FTP timeout is 300 seconds.
RFC 1123 states that the minimum idle timeout should be 5 minutes.

Top of list
Index page

FTP

[FTP]
USERx=user

Defines the user name for FTP.

Comments

You can define a USER0 and a USER1.
Default users are: 'anonymous' (no password) and 'ftp' (password is 'ftp').
You must specify both the user name and their password.
Neither user name nor password are case sensitive.
Maximum name size: 19 characters

Important notice: To avoid security leaks you must define both user names and passwords.

Related Topics

FTP user **write** protection

Top of list
Index page

FTP

[FTP]
PASSWORDx=password

Define the password for a FTP user

Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1
Default users are anonymous (no password) and ftp (password is 'ftp').
Neither user name nor password are case sensitive.
Maximum password size: 19 characters

Top of list
Index page

FTP

[FTP]
ACCESSRIGHTx=Access rights for defined Users

This CHIP.INI entry allows you to deny write access to FTP USER0 or USER1.

0 - write and read access enabled
1 - write access denied, read access enabled

Example which disables FTP write access for USER0:

```
[FTP]
USER0=otto
PASSWORD0=otto53pass
ACCESSRIGHT0=1
```

Comments

You can only forbid write access if you have defined the respective user with the FTP **USERx** and

PASSWORDx entries.

By default write access is enabled for both FTP users.

Top of list
Index page

FTP

[FTP]
DRIVEx=Set user's FTP drive

Set user's FTP drive.

Comments

Entries DRIVE0 or DRIVE1 can be made to specify a particular drive for use by FTP USER0 and USER1 respectively. The drive numbers are coded as follows:

0: Drive A (Default)
1: Drive B
2: Drive C

If the drive does not exist, the default drive A will be set.

The following example defines the root drive for USER0 to be on B: drive.

```
[FTP]
DRIVE0=1
```

Related Topics

FTP user's **root** directory

Top of list
Index page

FTP

[FTP]
ROOTDIRx=Name of the user's FTP server root directory

Defines the name of user's FTP server root directory.

Comments

The following example defines both root directory for USER1

```
[FTP]
ROOTDIR1=USERDIR
```

The default FTP root directory is the drive root directory, "\". If the specified FTP directory doesn't exist, the FTP server closes the connection.

If ROOTDIRx is set you must also specify the FTP **DRIVEx** entry.

Maximum ROOTDIRx path string length: 64 characters

Important notice:

To avoid security leaks you should define one "normal" user with a directory below the "\" root directory. The other user ROOTDIR should not be defined, allowing that "superuser" or "admin" to access all files on the drive.

Related Topics

FTP user's **DRIVE**

Top of list
Index page

WEB

[WEB]
ENABLE=0/1

Define if the Web server should be activated.

Comments

Use 0 to disable, 1 to enable.
By default, the Web server is enabled.

Related Topics

BIOS Ints API Call **Suspend/Resume system servers**

Top of list
Index page

WEB

[WEB]
MAINPAGE=Name of the main page

Defines the name of Web server's main page. The Web server opens this page if a browser request like `http://192.168.200.4/` is received. Typical names are "main.htm" (default) or "index.htm". The console command **`webstat`** shows the current main page.

Related Topics

Set Web Server **Main** Page API Function

Top of list
Index page

WEB

[WEB]
TEMPPATH=Name of a temporary Web server path

Defines a temporary path for finding files.
If the Web server cannot find the file in its default directory, it will try to find it in this temporary path.
Pathname should include the drive specification.

Comments

This function allows the Web server to locate HTML files produced on the IPC@CHIP RAMDISK by application programs.

The maximum string length is 32.

Example:

```
[WEB]
TEMPPATH=C:\web
```

The **webstat** command shows the current temporary path.

Top of list
Index page

WEB

[WEB]
DRIVE=Set Web server's disk drive

Set Web server's disk drive.
- 0: Drive A
- 1: Drive B
- 2: Drive C

If the drive does not exist, the default drive A will be set.
The console command **webstat** shows the current Web server drive.

Top of list
Index page

WEB

[WEB]
ROOTDIR=Name of the root directory

Defines the name of Web server's root directory. If the directory does not exist, the Web server sets "\" as the default root directory.

The console command **webstat** shows the current root directory.

Comments

Important notice: To avoid security leaks you should define a directory below the "\" directory. If you use "\" as web root directory, everybody can read all your files.

Related Topics

Set Web Server **Root** Directory API Function

Top of list
Index page

WEB

[WEB]
MAXCGIENTRIES=Maximum number of available CGI entries

Set the maximum number of entries for the Web server (Default: 10)
Range: 2 to 128

The console command **`cgistat`** shows the current number.

Top of list
Index page

WEB

[WEB]
WEBSERVERSTACK=Stack size

Sets the stack size (bytes) for the Web server task. The default and minimal stack size is 2048 Bytes.

Programmers of CGI functions who are using Microsoft C-Compilers with C-Library functions such as `sprintf`, which require a lot of stack space, should increase the stack size to 6144 (6 KBytes).

The maximum value is 10240 bytes.

Top of list
Index page

WEB

[WEB]
HTTPPORT=port

Sets the port number of the web server.
Default HTTP port: 80

Example:

```
[WEB]
HTTPPORT=81
```

The console command **webstat** shows the current HTTP port number.

Top of list
Index page

WEB

[WEB]
USER0=User name for Web Server PUT method

Defines a user name for transferring files to the Web server's root directory using the HTTP Put method. The standard user name is 'WEB'.

The console command **webstat** shows the user name and password.

Comments

Important notice: To avoid security leaks you should define a user name and password.

Top of list
Index page

WEB

[WEB]
PASSWORD0=Password for Web Server PUT method

Defines the password used to transfer files to the Web server's root directory using the HTTP Put method. The standard password is 'WEB'.

The console command **webstat** shows the Password and Username.

Comments

Important notice: To avoid security leaks you should define a user name and password.

Top of list
Index page

WEB

[WEB]
SECURE=Activated the Web security feature for the Web Server

Defines whether the security feature for the Web Server is active or not.

0 = security feature deactivated (default)
1 = security feature activated

Comments

The Web Server security feature allows up to two **paths** to be protected with **user name** and **password**. When this security feature is activated, users must then authenticate themselves to get Web access to these paths.

Top of list
Index page

WEB

[WEB]
SEC_URLx=Define a path for the security feature

Defines a specified URL for the Web security feature. The user can define SEC_URL0 and SEC_URL1.

All sub URLs of SEC_URLx are then protected by user name and password. The SEC_URL0 path is protected by **SEC_USER0** user name and **SEC_PASSWORD0** password, and SEC_URL1 by **SEC_USER1** user name and **SEC_PASSWORD1** password.

The maximum length for the paths is 63 characters.

Comments

If the **security** feature is activated, the user should define a path, **user name** and **password**.

In the example below all sub URLs of "[IP]/sec" are protected (e.g. "192.168.200.4/sec/page.htm").

Example:

```
[WEB]
SEC_USER0=otto
SEC_PASSWORD0=web
SEC_URL0=/sec
```

Top of list
Index page

WEB

[WEB]
SEC_USERx=Define a user name for the security feature

Defines a user name for the Web security feature.
The user can define SEC_USER0 and SEC_USER1.
The max length of the user name is 19 characters.

Comments

If the **security** feature is activated, the user should define a **path**, user name and **password**.

Top of list
Index page

WEB

[WEB]
SEC_PASSWORDx=Define a password for the security feature

Defines a password for the Web security feature.
The user can define SEC_PASSWORD0 and SEC_PASSWORD1.
The max length of the password is 19 characters.

Comments

If the **security** feature is activated, the user should define a **path**, **user name** and password.

Top of list
Index page

TFTP

[TFTP]
TFTPPORT=port

Sets the port number of the TFTP server.
Default TFTP port: 69

Example:

```
[TFTP]
TFTPPORT=4000
```

Top of list
Index page

TELNET

[TELNET]
TELNETPORT=port

Sets the port number of the Telnet server.
Default Telnet port: 23

Example:

```
[TELNET]
TELNETPORT=5000
```

Top of list
Index page

TELNET

[TELNET]
TIMEOUT=Telnet timeout minutes

Telnet session will automatically close after TIMEOUT minutes without any characters received from the client. A TIMEOUT setting of zero means no timeout.

Default Value: `TIMEOUT=0` (no timeout)

Top of list
Index page

TELNET

[TELNET]
LOGINDELAY=0/1

Define if the delayed login of the Telnet server should be activated.

Comments

Use 0 to deactivate, 1 to activate.
By default, the delayed login is enabled.
The delay time starts with 400 milliseconds.
After each following failed login the delay time will be doubled until it reached 20 seconds.
After successful login the delay time will be set back to 400 milliseconds.

Top of list
Index page

TELNET

[TELNET]
LOGINRETRIES=number of login retries

Defines the number of login retries until Telnet session will be closed.

Example:

```
[TELNET]
LOGINRETRIES=3
```

Comments

The default value is 5.

Top of list
Index page

TELNET

[TELNET]

USERx=user

Defines a user name for Telnet.

Comments

You can define a USER0 and a USER1.
Default user is 'tel', password is 'tel'.
You must specify both the user name and their password.
Neither user name nor password are case sensitive.

Maximum user name size: 19 characters

Important notice: To avoid security leaks you must define both user names and passwords.

Top of list
Index page

TELNET

[TELNET]
PASSWORDx=password

Define the password for a Telnet user

Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1.
Default user is 'tel', password is 'tel'.
Neither user name nor password are case sensitive.

Maximum password size: 19 characters

Top of list
Index page

TELNET

[TELNET]
ENABLE=0/1

Define if the Telnet server should be activated.

Comments

Use 0 to disable, 1 to enable.
By default, the Telnet server is enabled.

Related Topics

BIOS Ints API Call **Suspend/Resume system servers**

Top of list
Index page

DEVICE

[DEVICE]
FILESHARING=0/1

Disable/Enable the file sharing. See also BIOS Interrupt service **0x37**.

Comments

0=disable (default), 1=enable

Top of list
Index page

DEVICE

[DEVICE]
NAME=name

Define the name of this device.

Comments

This name will show up with the 'Chiptool' software when the network is scanned.

Maximum name size: 20 characters

Top of list
Index page

TRACE

[TRACE]
FLASHWRITE=0/1

Trace the activity of flash writes for debug purposes.

Comments

Set to 1 to enable the debug output. On STDOUT a message is printed at every call to the low-level physical flash write routine.

By default the debug output is disabled.

Top of list
Index page

TRACE

[TRACE]
INTNOTSUPP=0/1

Trace the activity of calls to not supported interrupts or functions.

Comments

Set to 0 to disable the debug output. On STDOUT a message is printed at every call to a not supported interrupt vector or not supported function.

By default the debug output is enabled.

Top of list
Index page

SERIAL

[SERIAL]
EXT_DMA=0/1

Disable/enable DMA receive mode (DMA0 / INT5) on the EXT port. If DMA receive mode is disabled, the EXT port works with the standard serial interrupt. The recommended mode is the DMA receive mode. It is only necessary to disable the DMA receive mode for the EXT port if DMA0 is needed by an external device (in the future). In the IRQ receive mode, you may loose characters if the system gets lots of interrupts (e.g. network) or if you are writing to the flash disk (file system calls). See documentation of the **Hardware API**.

Example which disables DMA receive mode on the EXT port.

```
[SERIAL]
EXT_DMA=0
```

By default, DMA receive mode is enabled.

Top of list
Index page

SERIAL

[SERIAL]
COM_DMA=0/1

Disable/enable DMA receive mode (DMA1 / INT6) on the COM port. If DMA transfer is disabled, the COM port works with the standard serial interrupt. The recommended mode is the DMA receive mode. It is only necessary to disable the DMA receive mode for the COM port if DMA1 will be needed by an external device (in the future). In the IRQ receive mode, you may loose characters if the system gets lots of interrupts (e.g. network) or if you are writing to the flash disk (file system calls).

See documentation of the **Hardware API**.

Example which disables DMA receive mode on the COM port.

```
[SERIAL]
COM_DMA=0
```

By default, DMA receive mode is enabled.

Top of list
Index page

SERIAL

[SERIAL]
SEND_DMA=0/1

Selects the DMA send mode for a serial port.

Comments

Use 0 to enable the DMA send mode for the EXT port.
Use 1 to enable the DMA send mode for the COM port.

Example which enables the DMA send mode on the COM port.

```
[SERIAL]
SEND_DMA=1
```

By default, DMA send is disabled.

Important:
The IPC@CHIP has only two DMA channels. By default both are used for receiving characters from the COM and EXT ports. If you want to use the DMA send mode for one serial port (e.g. the EXT port), the second port (e.g. the COM port) automatically switches over to using the serial port's receiver interrupt. Note that **RS485** is not available with serial send DMA.

Top of list
Index page

SERIAL

[SERIAL]
EXT_RECVQUEUE=size

Sets the receive queue size of the EXT port. Minimum size is 1024.
Maximum size is 10240 byte.

Comments

Example:

```
[SERIAL]
EXT_RECVQUEUE=2048
```

By default, the receive queue size is 1024 byte.
If it is planned to use PPP server or PPP client the size should set to 4096.

Top of list
Index page

SERIAL

[SERIAL]
EXT_SENDQUEUE=size

Sets the send queue size of the EXT port. Minimum size is 1024.
Maximum size is 10240 byte.

Comments

Example:
```
[SERIAL]
EXT_SENDQUEUE=2048
```

By default, the send queue size is 1024 byte.
If it is planned to use PPP server or PPP client the size should set to 4096.

Top of list
Index page

SERIAL

[SERIAL]
COM_RECVQUEUE=size

Sets the receive queue size of the COM port. Minimum size is 1024.
Maximum size is 10240 byte.

Comments

Example:
```
[SERIAL]
COM_RECVQUEUE=2048
```

By default, the receive queue size is 1024 byte.
If it is planned to use PPP server or PPP client the size should be set to 4096.

Top of list
Index page

SERIAL

[SERIAL]
COM_SENDQUEUE=size

Sets the send queue size of the COM port. Minimum size is 1024.
Maximum size is 10240 byte.

Comments

Example:
```
[SERIAL]
COM_SENDQUEUE=2048
```

By default, the send queue size is 1024 byte.
If it is planned to use PPP server or PPP client the size should be set to 4096.

Top of list
Index page

SERIAL

[SERIAL]
COM_BAUD=BAUD Rate

Sets the BAUD rate of the COM port.

Comments

Example:

```
[SERIAL]
COM_BAUD=9600
```

By default, the BAUD rate of the COM port is 19200 (with 8 data bits, no parity, 1 stop bit).

Top of list
Index page

SERIAL

[SERIAL]
EXT_BAUD=BAUD Rate

Sets the BAUD rate of the EXT port.

Comments

Example:

```
[SERIAL]
EXT_BAUD=9600
```

By default, the BAUD rate of the EXT port is 19200 (with 8 data bits, no parity, 1 stop bit).

Top of list
Index page

DOSLOADER

[DOSLOADER]

MEMGAP=Paragraphs

Sets a memory gap between the loaded DOS programs as a memory reserve.

Comments

Some programs compiled with Borland C 5.02 (other compilers??) try to increase their program memory block at runtime. This can occur, for example, when opening a file with Borland C-library function `fopen`, where some additional memory is required. The Borland C-library `fopen` function calls int 21h **0x4A**, which is not directly visible to the application programmer. This memory resize call fails if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previously executed program. The program then returns from `fopen` with an error. In this case, the global program variable errno is set to value 8 (not enough memory).

To prevent this error, the @CHIP-RTOS allows a memory gap of a defined size between loaded programs. This memory gap size is specified as a number of paragraphs (where 1 paragraph == 16 Bytes).

This strategy can fail when programs are terminated and restart again.

Example:

```
[DOSLOADER]
MEMGAP=128
```

By default, MEMGAP is set to 0. The maximum value is 2048 paragraphs, where any value larger than this is truncated to 2048

Related Topics

Set memory gap **API** function

Developer Notes

It is not necessary to set this entry if the application doesn't show the described error. Only if a C-library function call sets errno to 8, should this value be defined. We recommend in that case a value of 128 paragraphs (2048 Bytes). The described problem was noticed when the Borland C-library function `fopen` was used. The same can happen with usage of C-library function `malloc` using memory model Large. The `malloc` returns a NULL pointer in this case.

Top of list
Index page

BATCH

[BATCH]
BATCHMODE=0/1

Sets the batch file execution mode of DOS programs.

`BATCHMODE=0` : (Default mode)
The programs listed in the batch file will be **executed concurrently**, starting one after another, without waiting for completion (or going resident) of the predecessor program. The only exceptions are the **WAIT**and **REBOOT**commands.

BATCHMODE=1 :

The listed programs will be **executed sequentially**, one at time (similar to DOS).
The execution of the successor program will be delayed until the current program either finishes, terminates resident by calling DOS Interrupt **21h Service 0x31**or makes the @CHIP-RTOS Interrupt **0xA0 Service 0x15**batch file wakeup call.

The maximum delay time for execution of the next listed program in the batch file is 15 seconds, unless this limit has been deactivated with the EXECTIMEOUT=0**configuration**control.

Important:

If BATCHMODE=1take care that every program in your batch file which has a successor program either exits **(int21h 0x4C)**or terminates resident with **int21h 0x31**.
A program which runs forever should call from the main function @CHIP-RTOS Interrupt **0xA0 Service 0x15**, which immediately enables the further batch file sequencing.

Related Topics

BATCHMODE command
Run-time **batch mode** selection API

Top of list
Index page

BATCH

[BATCH]
EXECTIMEOUT=0/1

Disable (=0) / enable (=1) the batch file DOS program execution delay time limit for BATCHMODE=1.
EXECTIMEOUT=0 :

The successor program in a batch file waits forever if the predecessor program neither finishes nor calls **0xA0 Service 0x15**.

EXECTIMEOUT=1 : (Default mode)

The maximum delay time for execution of the next listed program in a batch file is 15 seconds.

Comments

This Boolean control applies only to BATCHMODE=1.

Example:

```
[BATCH]
EXECTIMEOUT=0
```

Related Topics

BATCHMODE Configuration

Top of list
Index page

CHIP.INI Updates - SC12 @CHIP-RTOS V1.10

CONFIG Updates

The following changes to the system **configuration** initialization are in the indicated @CHIP-RTOS revisions.

New in version 1.10B: **Activate Web security feature**
New in version 1.10B: **Define a path for the security feature**
New in version 1.10B: **Define a user name for the security feature**
New in version 1.10B: **Define a password for the security feature**
New in version 1.10: **Enable/Disable trace of physical flash writes**
New in version 1.10: **Enable/Disable output of INT not supported calls**

End of document

BIOS Interrupts

Here are the interface definition for the BIOS Interrupts

The system BIOS in a regular PC offers many services, only a subset of which is required in embedded systems. This subset is described here.

Unsupported BIOS functions are handled by a default handler which issues a message to the console.

Some additional functions not found in a normal PC BIOS are provided for your convenience.

For some useful comments see **Programming notes**

New in version 1.10B: **Extended: Get serial number**
New in version 1.10B: **Extended: Get @CHIP-RTOS features**
New in version 1.10B: **Extended: Install User Fatal Error Handler with Low Mem Error**
New in version 1.10B: **Extended: Get the IPC@CHIP device names**
New in version 1.10B: **Read/Write persistent user data**
New in version 1.10B: **Get IP address of the PPP Server**
New in version 1.10B: **Get IP address of the PPP Client**
New in version 1.11B: **Get sprintf address**

Interrupt 0x1A service 0x00: Get clock count since midnight

Returns the number of clock ticks since midnight.
The frequency of the clock is 18.2 Hz (e.g. 54.945 ms per tick).

Parameters

AH
Must be 0.

Return Value

Returns the 32 bit tick count in CX (high word) and DX (low word)
If an overflow occurred since the last call, AX is set to 1.

Comments

Please note that the overflow indication returned in register AX can not be relied upon if several tasks are using this service.

Top of list
Index page

Interrupt 0x11 service 0xXX: Get equipment list

Get the BIOS equipment list

Return Value

Returns the equipment list in AX, currently 0x013C. This bit field indicates:

bit 8: 1 : No DMA
bit 6-7: 00: One floppy
bit 4-5: 11: 80x25 mono
bit 2-3: 11: system ram
bit 1: 0 : no 8087
bit 0: 0 : no disk drives

Comments

This function is needed to make sure an application finds no 8087 coprocessor so it can load a floating-point emulator.

Top of list
Index page

Interrupt 0x10 service 0x00: Get char from standard input

Get a character from std in, wait if none available

Parameters

AH
Must be 0.

Return Value

Returns input character in AL
Return at DX the source stdin channel: 1: EXT, 2: COM, 4: Telnet, 8: User channel

Comments

Please note that AH does not contain the scan code, but is always 0.

Top of list
Index page

Interrupt 0x10 service 0x01: Check if a character is available from std in

Check if a character is available from standard input

Parameters

AH
Must be 1.

Return Value

AX=1 if a character is available, AX=0 and zero-flag is cleared if no character is available.

Comments

Please note that AH does not contain the scan code, but is instead always 0.

Top of list
Index page

Interrupt 0x10 service 0x08: Read character at cursor

Read character at cursor position, always returns 0

Parameters

AH
Must be 0x08

Return Value

AX = 0. (This function exists only for PC compatibility.)

Top of list
Index page

Interrupt 0x10 service 0x0E: Teletype output

Write a character to the standard output.

Parameters

AH
Must be 0x0E

AL
Character to write

Return Value

Returns nothing.

Comments

This call returns immediately after space becomes available in the transmit ring buffer.
(The data transfer from the transmit ring buffer to the hardware transmitter is interrupt driven.)

Top of list
Index page

Interrupt 0x10 service 0x0F: Get video state

Get video state

Parameters

AH
must be 0x0F

Return Value

number of screen columns, 80 in AH
mode currently set, 3 in AL
mode currently display page ,0 in BH

Top of list
Index page

Interrupt 0x10 service 0x12: Video subsystem configuration

Video subsystem configuration

Parameters

AH
must be 0x12

Return Value

AX=0x0012
BX=0
CX=0

Top of list
Index page

Interrupt 0x16 service 0x00: Get char from standard input

Get a character from std in, wait if none available

Parameters

AH
Must be 0.

Return Value

Returns character in AL
Returns at DX the source stdin channel of the character: 1: EXT , 2: COM , 4: Telnet

Comments

Please note that AH does not contain the scan code, but is always 0.

Top of list
Index page

Interrupt 0x16 service 0x01: Check if a character is available from std in

Check if a character is available from standard input

Parameters

AH
Must be 1.

Return Value

AX=1 if a character is available, AX=0 and zero-flag is cleared if no character is available.

Comments

Please note that AH does not contain the scan code, but is always 0.

Top of list
Index page

Interrupt 0xA0 service 0x00: Get serial number

Get the serial number of the IPC@CHIP device

Parameters

AH
Must be 0.

Return Value

AX=low word, BX=high word of IPC@CHIP serial number

CX=low word, DX=high word of BECK product serial number (if not available CX=DX=0)
SI contains BECK product hardware revision number HighByte and LowByte (if not available SI=0)

DI contains SCxxx hardware revision number HighByte and LowByte (if not available DI=0)

Comments

The serial number is a 24 bit value.

Top of list
Index page

Interrupt 0xA0 service 0x01: Get IP address of the Ethernet interface

Get the IP address as a string.

Parameters

AH
Must be 1.

ES:DX
Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string.

Related Topics

Ethernet **IP address** initial value
Convert ASCII IP address to **binary**

Top of list
Index page

Interrupt 0xA0 service 0x02: Set IP address of the Ethernet interface

Set the Ethernet interface's IP address based on the supplied string.

Parameters

AH
Must be 2.

ES:DX
Pointer to a 16 byte memory area where the IP address is stored as a null terminated string.

Comments

A new IP configuration must be activated by calling the `ipeth` **command**
or by calling the TCP/IP API interrupt 0xAC **service 0x71** (`RECONFIG_ETHERNET`).

Important:
This API function writes to chip.ini and is not reentrant.
Don't use in different tasks or in combination with @CHIP-RTOS commands,
which are writing to chip.ini, e.g. DHCP. Avoid race conditions with
any other API call which also writes or reads the chip.ini file.
e.g. **service 0x23**

Related Topics

TCP/IP API **RECONFIG_ETHERNET** service
Ethernet **IP address** initial value
IP command line
Convert binary IP address to **ASCII** dotted decimal

Top of list
Index page

Interrupt 0xA0 service 0x03: Get IP subnet mask of the Ethernet interface

Get the IP subnet mask as a string.

Parameters

AH
Must be 3.

ES:DX
Pointer to a 16 byte memory area where the IP subnet mask is to be stored as a null terminated string.

Related Topics

Ethernet **IP subnet mask** initial value

Top of list
Index page

Interrupt 0xA0 service 0x04: Set IP subnet mask

Set the IP subnet mask to the string supplied

Parameters

AH
Must be 4.

ES:DX
Pointer to a 16 byte memory area where the IP subnet mask is stored as a null terminated string.

Comments

A new IP configuration must be activated by calling the `ipeth` **command**
or by calling the TCP/IP API interrupt 0xAC **service 0x71** (`RECONFIG_ETHERNET`).

Important:
This API function writes to chip.ini and is not reentrant.
Don't use in different tasks or in combination with @CHIP-RTOS commands,
which are writing to chip.ini, e.g. DHCP. Avoid race conditions with
any other API call which also writes or reads the chip.ini file.
e.g. **service 0x23**

Related Topics

Ethernet **IP subnet mask** initial value
NETMASK command line
TCP/IP **API** documentation

Top of list
Index page

Interrupt 0xA0 service 0x05: Get IP gateway

Get the IP gateway as a string.

Parameters

AH
Must be 5.

ES:DX
Pointer to a 16 byte memory area where the IP gateway is to be stored as a null terminated string.

Top of list
Index page

Interrupt 0xA0 service 0x06: Set IP default gateway

Set the IP gateway to the string supplied

Parameters

AH
Must be 6.

ES:DX
Pointer to a 16 byte memory area where the IP gateway is stored as a null terminated string.

Comments

A new IP configuration must be activated by calling the `ipeth` **command**
or by calling the TCP/IP API interrupt 0xAC **service 0x71** (`RECONFIG_ETHERNET`).

The TCP/IP stack of the IPC@CHIP supports only one valid default gateway for all device interfaces:
Ethernet, pppserver and pppclient.

The `ipcfg` **command** shows the current default gateway.

Important:
This API function writes to chip.ini and is not reentrant.
Don't use in different tasks or in combination with @CHIP-RTOS commands,
which are writing to chip.ini, e.g. DHCP. Avoid race conditions with
any other API call which also writes or reads the chip.ini file.
e.g. **service 0x23**

Related Topics

Ethernet default **gateway** initialization
GATEWAY command line
ADD_DEFAULT_GATEWAY API function
PPP server default **gateway** initialization

Top of list
Index page

Interrupt 0xA0 service 0x07: Execute a command shell command

Passes a command string to the **command** interpreter.

Parameters

AH
Must be 7.

ES:DX
Pointer to a null terminated command line.

Return Value

AX==0 success
AX==-1 error (loading EXE file failed!)

Comments

Internal commands are processed in the current task, external commands (.exe files) are loaded and executed in a new task.

Top of list
Index page

Interrupt 0xA0 service 0x08: Set timer 0x1C's interval

Defines the interval in milliseconds for timer interrupt 0x1C.

Parameters

AH
Must be 0x08.

BX
Interval in milliseconds

Comments

Use `setvect(0x1c, my_function)` to change the interrupt vector.

Define your routine as:
```
void interrupt my_function(void)
```

You must restore the old timer interrupt vector before ending the program.
Your interrupt routine must be as short as possible without any waiting or endless loops.
Avoid the usage of large C-library functions such as `printf`.

Related Topics

chip.ini **TIMER 0x1C** configuration

Top of list
Index page

Interrupt 0xA0 service 0x09: Set timer interrupt 0xAF's interval

Defines the interval in milliseconds for timer interrupt 0xAF.

Parameters

AH
Must be 0x09.

BX
Interval in milliseconds

Comments

Use `setvect(0xAF, my_function)` to change the interrupt vector.

Define your routine as:
```
void interrupt my_function(void)
```

You must restore the old timer interrupt vector before ending the program.
Your interrupt routine must be as short as possible without any waiting or endless loops.
Avoid the usage of large C-library functions such as `printf`.

Related Topics

chip.ini **TIMER 0xAF** configuration

Top of list

Interrupt 0xA0 service 0x11: Set STDIO focus

Set the focus of STDIO to either console, application or both

Parameters

AH
Must be 0x11

AL
1: command shell (console), 2: Application, 3: both

Comments

If your application requires input from the user, you should set the focus to the application.
You should assure that only one application requests input from STDIO.
The user can change the focus by using the focus hot key (default Ctrl-F).
Changing the focus clears the serial input and output queues immediately.

Important :
All buffered incoming and outgoing characters in the internal serial send and receive queues are lost after this call.

Related Topics

Focus key definition

Top of list
Index page

Interrupt 0xA0 service 0x12: Get bootstrap version number

Get the version number of the bootstrap loader.

Parameters

AH
Must be 0x12

Return Value

AX=version number, AH is major version, AL is minor version.

Comments

Example:
If the function returns 0x0100 in AX, this means that you have version 1.00.

Top of list
Index page

Interrupt 0xA0 service 0x13: Get @CHIP-RTOS version number

Get the version number of the @CHIP-RTOS.

Parameters

AH
Must be 0x13

Return Value

AX=version number, AH is major version, AL is minor version.
If DX is set it is a Beta version.

Comments

Example:
If the function returns 0x0100 in AX, this means that you have version 1.00.

Top of list
Index page

Interrupt 0xA0 service 0x14: Set batch file execution mode.

Sets the batch file execution mode of DOS programs for either concurrent or sequential execution.
See `BATCHMODE` initialization **documentation** for details.

Parameters

AH
Must be 0x14

AL

AL = 0: (Selects default `BATCHMODE=0,` = concurrent)

AL = 1: (Sets `BATCHMODE=1,` = sequential)

BX

BX = 0: Disable the max. delayed execution timeout of DOS programs

BX = 1: Enable the max. delayed execution timeout of DOS programs at a batchfile, if BATCHMODE=1

Return Value

returns nothing

Comments

Important:
If `BATCHMODE=1`take care that every program in your batch file which has a successor
program either exits **(int21h 0x4C)**or terminates resident with **int21h 0x31**.
A program which runs forever should call BIOS Interrupt **0xA0 Service 0x15**, which immediately enables the further batch file sequencing.
By default the maximum delay time for execution of the next listed program in the batch file is 15 seconds.
If BX is set to 0, the successor program in a batch file waits forever for execution, if the predecessor program does not finish or call **0xA0 Service 0x15**

Related Topics

Initial **batch mode** configuration
BATCHMODE command

Top of list
Index page

Interrupt 0xA0 service 0x15: Allow immediate further batch file execution in BATCHMODE 1.

This call allows the next program listed in a batch file to start execution.
This is implemented by waking up a batch file execution task which dispatches any subsequent program listed in the batch file.

Parameters

AH
Must be 0x15

Return Value

returns nothing

Related Topics

Initial **batch mode** configuration
Run-time **batch mode** selection

Top of list
Index page

Interrupt 0xA0 service 0x16: Get information about the @CHIP-RTOS features

Get information about running servers, interfaces and features of the @CHIP-RTOS

Parameters

AH
0x16

Return Value

Bits of AX, BX and DX indicate the services or devices available, coded as:
Bit=0: service or device is not available.
Bit=1: service or device is available.

AX:
Bit 0: Ethernet device for TCP/IP
Bit 1: PPP server
Bit 2: PPP client
Bit 3: Web server
Bit 4: Telnet server
Bit 5: FTP server
Bit 6: TFTP server
Bit 7: DHCP client

BX:
Bit 0: SNMP MIB variables support (see **TCP/IP API service 0x71**)
Bit 1: UDP config server
Bit 2: Ping client (see **TCP/IP API service 0x75**)
Bit 3: TCP/IP device driver API(see **TCP/IP API service 0xA0 - 0xA7**)
Bit 4: Webfile upload (Webserver PUT method)

DX:
Bit 0: I2C-Bus API
Bit 1: Hardware API
Bit 2: RTOS API
Bit 3: Packet driver interface for Ethernet
Bit 4: Serial XMODEM file transfer
Bit 5: External disk interface
Bit 6: Software SPI API

Top of list
Index page

Interrupt 0xA0 service 0x17: Get MAC address of the Ethernet interface

Get the MAC address as a 6 Byte array.

Parameters

AH
Must be 0x17.

ES:DX
Pointer to a 6 byte memory area where the MAC address is to be stored.

Comments

At IPC@CHIP targets without internal Ethernet (e.g. SC11), this number represents a virtual ID.

Top of list
Index page

Interrupt 0xA0 service 0x18: Power save

Slows down the internal timer for the RTOS and puts the CPU in a halt mode until the next interrupt occurs. Please note that the internal time/date will be affected.

Parameters

AH
Must be 0x18.

Comments

Call this function when your program is in idle state.
Power savings are marginal since we use a DRAM. Please note that power consumption may differ slightly when the date code of the IPC@CHIP is changed.

Top of list
Index page

Interrupt 0xA0 service 0x19: Change level for configuration server.

Change the supported level for the configuration server. For a description of the possible levels, please refer to **config.htm** document.

Parameters

AH
Must be 0x19.

BX
The supported level.

Comments

Please note that if the level defined in the **`chip.ini`** is 0 (zero), the configuration server task is not started and changing the supported level does not have any effect. To avoid this, use a unlisted support level such as 0x1000 in the `chip.ini`.
The entry in the `chip.ini` file is not changed by this call.

Top of list
Index page

Interrupt 0xA0 service 0x20: Install a user fatal error handler

Install a user fatal error callback function. This installed function will be called by the system on execution of fatal errors.

Parameters

AH
Must be 0x20.

ES:DI
Address of the user error handler function (or zero to remove a previously installed handler).

Comments

The user is permitted to execute an error handler function if a fatal error occurs in either an application program or the @CHIP-RTOS.

Note that this mechanism will (of course) fail if the user error handler code is itself overwritten (corrupt).

Only one error handler callback per system is supported. An application can remove its handler by calling this install function with a zero (NULL) value in ES:DI, which is an advisable clean-up procedure at program exit.

The callback function must be of type `huge _pascal` with an `errorcode` input parameter (see the example function below). The error codes conveyed by this input parameter are as follows:

1: Invalid processor opcode (usually caused by corrupted memory). The current task will be suspended.
2: Fatal kernel error (usually caused by corrupted memory or a task stack overflow)
3: Fatal internal TCP/IP error, current task will be suspended
4: TCP/IP stack reaches memory limit
5: TCP/IP memory allocation error
6: Ethernet bus error (hardware decfect)
7: Ethernet link error detected (cable not connected?)
8: Flash write error -> Flash defect (**Note:** IPC@CHIP is no longer usable)
9: Low Memory error -> called if a memory allocation (system or user) failed.

In all cases (except errorcode 6,7 and 9) we recommend a reboot with BIOS interrupt 0xA0 **0x21**.

Important: Do not use any message printing inside your error handler if error code is 3 or 4, because when Telnet is part of your stdio, your exit handler will hang inside the print call.

```
// Example for Borland C:  This error handler reboots the system
void huge _pascal user_error_handler(int errorcode)
{
   union  REGS  inregs;
   union  REGS  outregs;

   // e.g. resetting outputs
   outportb(0x600,0x00);

   // rebooting the IPC@CHIP
   inregs.h.ah = 0x21;
   int86(0xA0,&inregs,&outregs);
}

// Installing the error handler function from program's main function
inregs.h.ah=0x20;
sregs.es   =FP_SEG(user_error_handler);
inregs.x.di=FP_OFF(user_error_handler);
int86x(0xA0,&inregs,&outregs,&sregs);
```

Top of list
Index page

Interrupt 0xA0 service 0x21: Rebooting the IPC@CHIP

This function works in the same way as the **reboot** shell command

Parameters

AH
0x21

Return Value

No return from this function occurs due to system reboot.

Top of list
Index page

Interrupt 0xA0 service 0x22: Get version string

Copies the @CHIP-RTOS version information in to a text buffer. The string is null terminated.

Parameters

AH
Must be 0x22

CX
Buffer length, including space for null terminator

ES
Segment of memory buffer for the string

DI
Offset of memory buffer for the string

Related Topics

VER shell command

Top of list
Index page

Interrupt 0xA0 service 0x23: Insert an entry in chip.ini

The functions 0x23 and 0x24 allows the user to modify/place and find/read
your own chip.ini entries.

Parameters

AH
Must be 0x23.

BX:SI
Pointer to section string (max. 40 chars)

ES:DI
Pointer to item name (max. 40 chars)

DS:DX
Pointer to item text (max. 128 chars)

Return Value

AX=0 success , AX=-1 invalid string length

Comments

Important: The API functions 0x23 and 0x24 are not reentrant.
Don't use in different tasks or in combination with @CHIP-RTOS commands,
which are writing to chip.ini e.g. DHCP. Avoid race conditions with
any other API call, which writes or read also to/from chip.ini
e.g. **service 0x02**

Example (tested with Borland C/C++ 5.02):

```
int iniPutString(char *sectionName, char *itemName, char *text)
{
  union  REGS  inregs;
  union  REGS  outregs;
  struct SREGS sregs;

  inregs.h.ah = 0x23;
  inregs.x.bx = FP_SEG(sectionName);
  inregs.x.si = FP_OFF(sectionName);
  sregs.es    = FP_SEG(itemName);
  inregs.x.di = FP_OFF(itemName);
  sregs.ds    = FP_SEG(text);
  inregs.x.dx = FP_OFF(text);
  int86x(0xA0,&inregs,&outregs,&sregs);
  return outregs.x.ax;
}
//Call of this function:
iniPutString("MY_SECTION", "MY_ITEM", "VALUE_TEXT");
//and produces the following chip.ini entry
[MY_SECTION]
MY_ITEM=VALUE_TEXT
<nl>
```

Developer Notes

Keep in mind that this function writes to the chip.ini file. This generates flash write cycles and these cycles are limited.

Top of list
Index page

Interrupt 0xA0 service 0x24: Find an entry in chip.ini

Finds an entry in chip.ini configuration file.

Parameters

AH
Must be 0x24.

CX
Maximum length of target string (without '\0').

BX:SI
Pointer to section string

ES:DI
Pointer to item name

DS:DX
Pointer to target

Return Value

AX= 0 : Entry not found
AX= -1: Could not open chip.ini
else Success: pointer at DS:DX contains the found string AX contains length of the found string

Comments

Important: API functions 0x23 and 0x24 are not reentrant.
Don't use in different tasks or in combination with @CHIP-RTOS commands,
which are writing to chip.ini, e.g. DHCP. Avoid race conditions with
any other API call, which writes or read also to/from chip.ini
e.g. 0xA0 0x02 Set IP address

The maximum length value in CX specifies the number of characters which could be copied into the target string. You have to allocate one more character for the null termination.

Example (tested with Borland C/C++ 5.02):

```
int iniGetString(char *sectionName, char *itemName, char *target, int maxlen)
{
   union  REGS  inregs;
   union  REGS  outregs;
   struct SREGS sregs;

   inregs.h.ah = 0x24;
   inregs.x.bx = FP_SEG(sectionName);
   inregs.x.si = FP_OFF(sectionName);
   sregs.es    = FP_SEG(itemName);
   inregs.x.di = FP_OFF(itemName);
   inregs.x.cx  = maxlen;
   sregs.ds    = FP_SEG(target);
   inregs.x.dx = FP_OFF(target);
   int86x(0xA0,&inregs,&outregs,&sregs);
   return outregs.x.ax;
}
// Declare a target buffer to be filled by iniGetString().
unsigned char target[101];

iniGetString("MY_SECTION", "MY_ITEM", target, 100);
// Now target contains the chip.ini text for this item
```

Top of list
Index page

Interrupt 0xA0 service 0x25: Set the Stdio focus key

Set the Stdio focus key

Parameters

AH
Must be 0x25.

AL
Focus key character (default CTRL-F, ASCII 6)

Return Value

Returns nothing

Comments

By default, the focus key is set to CTRL-F (ASCII 6)
At runtime, the pressed key Ctrl-F toggles between these three modes and shows the current mode.

Key Range: 0..254

If the key is set to zero, the switching of stdio is disabled.
The focus key is not usable by the command shell or dos executable.

Related Topics

Focus key definition

Top of list
Index page

Interrupt 0xA0 service 0x26: Get the IPC@CHIP device names

Get the IPC@CHIP device names

Parameters

AH
Must be 0x26.

Return Value

AX=0
ES:DI contains pointer to the fixed IPC@CHIP device name stored at the IPC@CHIP flash
BX:SI contains pointer to the device name configured at chip.ini.

DX:CX contains pointer to the fixed BECK product name stored in the IPC@CHIP flash

Comments

All returned strings are terminated by 0. These strings should be treated as read only.

Related Topics

Device name definition

Top of list
Index page

Interrupt 0xA0 service 0x27: Suspend/Resume System Servers

Suspend/Resume FTP, Telnet or Web Server

Parameters

AH
Must be 0x27.

AL
0: Resume, 1: Suspend

BX
0: FTP Server, 1: Telnet Server, 2: Web Server

Return Value

AX= 0: Success
AX= 1: Server was already in the postulated state
AX= -1: Invalid Parameter

Comments

If FTP, Telnet or WEB was disabled at startup with a CHIP.INI entry, you can enable it using this call.

Top of list
Index page

Interrupt 0xA0 service 0x28: Fast Findfirst

Provide a faster access to the file system directories

Parameters

AH

Must be 0x28.

CX
File attribute

BX:SI
Null terminated file specification

ES:DI
Pointer to filefind structure

Return Value

AL= 1 DX=0: Success
AL= 0 DX=0: No file found
DX= -1: Findfirst already active

Comments

The three filefind functions (0x28 - 0x30) provide a faster Findfirst/next access than the DOS compatible functions at INT **21h**. They work in a manner similar to the INT 21h Findfirst/next functions. You must first call Findfirst (0x28). After that call, you can call Findnext (0x29) as much as you need. To get the entire directory call repeatedly until Findnext returns an error. The directory being searched will be locked for exclusive access by the calling task. To unlock the directory you have to call Finddone (0x30).

Note: Only a successful call of this function (file found) must be later terminated by a call of the **Fast Finddone** function!
These functions (0x28 - 0x30) are not reentrant, do not call findfirst/findnext sequences from different tasks without semaphore protection.

The filefind structure is defined as follows:

```
typedef struct filefind
{
     char               filename[12];       // Null terminated filename
     char               fileext[4];         // and extension
     unsigned short int fileattr;           // MS-DOS file attributes
     short int          reserved;           // Reserved
     struct tag_filetimestamp
     {
        unsigned short int filedate;        // Date = ((year - 80) shl 9) or (month shl 5)) or day
        unsigned short int filetime;        // Time = (hour shl 11) or (min shl 5)) or (sec / 2)
     } filetimestamp;                       // Time & date last modified
     unsigned long      filesize;           // File size
     char               private_field[180]; // Reserved, used internal
}
```

Related Topics

Fast Findnext
Fast Finddone must be called at end of the search

Top of list
Index page

Interrupt 0xA0 service 0x29: Fast Findnext

Continues a search which was started by Fast Findfirst (0x28)

Parameters

AH
Must be 0x29.

ES:DI
Pointer to filefind structure

Return Value

AL= 1 DX=0: Success
AX= 0 DX=0: no file found
DX= -1: Findfirst not called before

Comments

See **Fast Findfirst function (0x28)** for description.

Note: Successful calls of this function (filefind) must be eventually terminated by a call to the **Fast Finddone** function!

Related Topics

Fast Findfirst must be called to start the search
Fast Finddone must be called at end of the search

Top of list
Index page

Interrupt 0xA0 service 0x30: Fast Finddone

Closes a find access started with Fast Findfirst, thereby allowing a subsequent Fast Findfirst operation.

Parameters

AH
Must be 0x30.

ES:DI
Pointer to filefind structure

Return Value

AX=0 DX=0: Success
DX= -1: Findfirst not active

Comments

See **Fast Findfirst function (0x28)** for description.

Note: This function must be called to close a find operation following a successful call to Fast FindFirst! Otherwise further attempts to use the Fast FindFirst function will fail.

Related Topics

Fast Findfirst
Fast Findnext

Top of list
Index page

Interrupt 0xA0 service 0x31: Detect Ethernet link state

Detect Ethernet link state

Parameters

AH
Must be 0x31.

Return Value

AX=0: Link ok
AX!=0: No Link (no cable connected?)
DX!=0: Initialization or reset procedure of Ethernet device failed

SC13 only: CX holds the current Phy status

```
CX register Phy status word bit definitions (SC13 only):
Bit 14    1 = Link Not detected
Bit 07    1 = 100Base-TX mode, 0=10Base-T mode
Bit 06    1 = Device in Full Duplex mode
```

Top of list
Index page

Interrupt 0xA0 service 0x32: Set a memory gap between the loaded DOS programs

Sets a memory gap between loaded DOS programs as a memory reserve.

Parameters

AH
Must be 0x32.

BX
Number of paragraphs (range between 0 to 2048 paragraphs).

Return Value

AX=0 DX=0: Success
AX=-1: Invalid value found in BX

Comments

Some programs compiled with Borland C 5.02 (other compilers??) try to increase their program memory block at runtime. This can occur, for example, when opening a file with Borland C-library function `fopen`, where some additional memory is required. The Borland C-library `fopen` function calls int 21h **0x4A**, which is not directly visible to the application programmer. This memory resize call fails if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previously executed program. The program then returns from `fopen` with an error. In this case, the global program variable errno is set to value 8 (not enough memory).

To prevent this error, the @CHIP-RTOS allows a memory gap of a defined size between loaded programs. This memory gap size is specified as a number of paragraphs (where 1 paragraph == 16 Bytes).

This value can also be defined in **chip.ini**.

Note: This strategy can fail when programs are terminated and restarted again.

Developer Notes

It is not necessary to set this entry if the application doesn't show the described error. Only if a C-library function call sets errno to 8, should this value be defined. We recommend in that case a value of 128 paragraphs (2048 Bytes). The described problem was noticed when the Borland C-library function `fopen` was used. The same can happen with usage of C-library function `malloc` using memory model Large. The `malloc` returns a NULL pointer in this case.

Top of list
Index page

Interrupt 0xA0 service 0x33: Set stdin/stdout channel

Set the stdin/stdout channel

Parameters

AH
Must be 0x33.

AL
Bit 0 = 1 set stdout, Bit 1 = 1 set stdin

BX
Channel bits, see comment

Return Value

AX=0 DX=0: Success
AX=DX=-1: Invalid parameter

Comments

Channel bits for BX register:
Bit 0: Serial port 0 (PORT EXT)
Bit 1: Serial port 1 (PORT COM)
Bit 2: Telnet server
Bit 3: User channel

Setting a bit to zero deactivates this channel, setting a bit to one activates the specified channel

Top of list
Index page

Interrupt 0xA0 service 0x34: Get stdin/stdout settings

Get the stdin/stdout settings

Parameters

AH
Must be 0x34.

Return Value

AX=0, BX contains stdout settings, CX: stdin settings

Comments

Return values
Bit 0 == 1: Serial port 0 (EXT)
Bit 1 == 1: Serial port 1 (COM)
Bit 2 == 1: Telnet server
Bit 3 == 1: User channel

Top of list
Index page

Interrupt 0xA0 service 0x35: Install user specific stdio handlers

Installs user specific stdio channel handlers.

This API call allows the user to install their own stdio handler functions, e.g. for a user developed input/output device connected to the IPC@CHIP (e.g. a display and/or keyboard device) or a user TCP application similar to Telnet.

The user must implement inside of their application four functions for reading and writing characters from/to their stdin/stdout device.

After installing these functions with this API call and setting stdin and/or stdout to include the user channel with **Set stdio channels**,

the @CHIP-RTOS will call these user functions at each stdin and stdout operation. For further details on programming and installing these handler functions, see the comments below.

Parameters

AH
Must be 0x35.

ES:DI
Pointer to User_Stdio_Funcs structure variable, see comments

Return Value

AX=0 DX=0: Success

Comments

Necessary type definitions:

```
typedef int (huge *User_Kbhit)(void);
typedef void (huge *User_PutCh)(char chr);
typedef void (huge *User_PutStr)(char * pch, int n);
typedef int (huge *User_Getch)(void);


typedef struct tag_user_stdio
{
	User_Kbhit user_kbhit;
	User_Getch user_getch;
	User_PutCh user_putch;
	User_PutStr user_putstr;
}User_Stdio_Funcs;
```

Functions to be implemented by the user:

```
int huge user_kbhit(void);     // returns 1 ,if a character is available, 0 if not
int huge user_getch(void);     // read a char from stdin (wait, if none available)
void huge user_putch(char chr);     // write a single char to stdout
void huge user_putstr(char * pch, int n);    // write a string with n chars to stdout
```

The user must set the four callback function vectors in their User_Stdio_Funcs type variable and then call this API function with the address of the User_Stdio_Funcs structure in ES:DI.

With ES = DI = 0, the user can later uninstall their stdio handlers.

Important:
1. If your applications exits, don't forget to uninstall your stdio handlers, by calling this API call with ES=DI=0.
2. Do not call your implemented stdio functions from inside your application. These installed functions must be called only from inside the @CHIP-RTOS.

Example (Borland C):

```
User_Stdio_Funcs user_stdio_funcs;    // global variable

// Implementation of users stdio functions
int  huge my_kbhit(void)
{
  //........
}

int huge my_getch(void)
{
  //........
}

void huge my_putch(char chr)
{
  //........
}
```

```
void huge my_putstr(char * pch, int n)
{
  //.......
}

void install_mystdio_channel(void)
{
   union  REGS  inregs;
   union  REGS  outregs;
   struct SREGS sregs;

   user_stdio_funcs.user_kbhit  = my_kbhit;
   user_stdio_funcs.user_getch  = my_getch;
   user_stdio_funcs.user_putch  = my_putch;
   user_stdio_funcs.user_putstr = my_putstr;

   inregs.h.ah=0x35;
   sregs.es   =FP_SEG(&user_stdio_funcs);
   inregs.x.di=FP_OFF(&user_stdio_funcs);
   int86x(0xA0,&inregs,&outregs,&sregs);
}

void remove_mystdio_channel(void)
{
   union  REGS  inregs;
   union  REGS  outregs;
   struct SREGS sregs;

   inregs.h.ah=0x35;
   sregs.es   = 0;
   inregs.x.di= 0;
   int86x(0xA0,&inregs,&outregs,&sregs);
}

unsigned int set_stdio_channel(unsigned int channels)
{
        union  REGS  inregs;
        union  REGS  outregs;

        inregs.h.ah=0x33;
        inregs.h.al=3;   //set both: stdin and stdout
        inregs.x.bx = channels;
        int86(0xA0,&inregs,&outregs);
        return outregs.x.ax;
}

int main(void)
{
  //......
  install_mystdio_channel();
  set_stdio_channel(0x0E);  // COM,TELNET,USER
  //....

  // at the end of the program

  set_stdio_channel(0x06);  // COM,TELNET
  remove_mystdio_channel():
}
```

Top of list
Index page

Interrupt 0xA0 service 0x36: Install a System Server Connection Handler function

Installs a user specific System Server Connection Handler function.
The handler function will be called if a client establishes a connection. This functions allows application programmers to implement their own callback functions for controlling access to the default @CHIP-RTOS servers.

Parameters

AH
Must be 0x36.

BX
0: FTP Server, 1: Telnet Server, 2: Web Server

ES:DI
Pointer to handler function

Return Value

AX=0: Success
AX=-1: Invalid Parameter

Comments

A connection handler function must be declared in the following manner:

```
int huge UserConnectionHandler( struct sockaddr_in *sockptr );
```

The connection handler will be called when a client establishes a connection to the server (FTP, WEB, Telnet). The handler can read the IP Address and the Port in the **sockaddr_in struct**. (If desired, the IP address in the `sin_addr` structure member can be converted to ASCII using the TCP/IP **API_INETTOASCII** function.) If the handler returns 0 the connection will be established. If it returns a nonzero value, the connection will be aborted.

To uninstall a connection handler call this function with a null pointer (ES=DI=0).

Example usage:

The implemented handler function can check the source IP address (Clients IP), compare this IP with an application internal list of allowed IP addresses. The connection can then be rejected by returning a non-zero value if the source IP is not a member of the list.

Top of list
Index page

Interrupt 0xA0 service 0x37: Enable/Disable File sharing

Enable disable File sharing of Int21h open/create

Parameters

AH
Must be 0x37.

AL
0: set mode, 1: get mode

BX
0: disable, 1: enable (Sharing mode)

Return Value

AX=0: Success, contains Sharing mode if al=1
AX=-1: Invalid Parameter

Comments

By default file sharing is disabled. This has the effect that a file which is opened for write access can't be opened a second time for read or write access. Also a file which is opened for read access can only be opened for read access a further time.

To avoid this security feature you can enable file sharing. This can also be done with the CHIP.INI entry **FILESHARING**.

NOTE: Be careful when opening a file multiple times with one or more of the openings done with write access!

Top of list

Index page

Interrupt 0xA0 service 0x38: Get file name by handle

Returns the file name string corresponding to a specified file handle.

Parameters

AH
Must be 0x38.

CX
File handle

ES:BX
pointer to string, must be 13 chars long (12 + Null termination)

Return Value

AX=0: Success, contains Sharing mode if al=1
AX=-1: Invalid file handle

Top of list
Index page

Interrupt 0xA0 service 0x40: Install a UDP Cfg Callback

Install a UDP Config Server User Callback function.

Parameters

AH
Must be 0x40.

ES:DI
Pointer to the User Callback function

Comments

This API Function installs a User Callback function for the UDP Config Server. If a UDP Cfg Request with the command number 06 arrives, this User Callback function will be called. This allows you to realize your own UDP Config sub protocol and commands. The UDP Cfg Callback function receives an argument with information about the UDP Cfg Request and its requester. This information structure and the function must be declared as specified in the description below:

```
typedef struct UdpCfgSrv_UserCBInfo
{
    int length;                     // Length of this struct
    struct sockaddr_in *fromAddrPtr;    // Sender address pointer
    int udpCfgSD;               // UDP Config Server's Socket Descriptor
    char *dataPtr;              // Data of Request package
    unsigned dataLength;        // Length of request package
};

void huge MyUdpCfgSrvCB( struct UdpCfgSrv_UserCBInfo *infoPtr );
```

If the callback function returns with the `dataPtr` and `dataLength` fields in the `UdpCfgSrv_UserCBInfo` structure both non-zero, the UDP Config Server will send the `dataLength` bytes from location referenced by `dataPtr` back to the requester. If the pointer is set to null or the `dataLength` field is set to 0, no data will be sent back to the requester.

To remove an installed callback function, call this function with a null pointer.

For more Information on the UDP Config Server and its protocol, refer to the UDP Config Server description available on our website.

Note: The data sent and received is limited to 300 bytes maximum.

Top of list

Index page

Interrupt 0xA0 service 0x45: Write persistent User Data

Writes persistent data into the Flash memory of the @CHIP

Parameters

AH
Must be 0x45.

ES:SI
Pointer to the User data

CL
number of bytes to write (max 192)

Comments

The user can use this function to save product specific persistent data (e.g. own serial number of the product).
The function requires a pointer in [es:si] to memory block, which should be written into the flash memory.
The 192 bytes array in the flash memory will be untouched on a format of the file system and also on an @CHIP-RTOS BIOS Update.

Related Topics

Read persistent User Data

Top of list
Index page

Interrupt 0xA0 service 0x46: Read persistent User Data

Reads persistent data from the Flash memory of the @CHIP

Parameters

AH
Must be 0x46.

ES:DI
Pointer to the User buffer

CL
number of bytes to read (max 192)

Comments

The user can use this function to read product specific persistent data written with function **0x45** (Write Persistent User Data).
The function requires a pointer [es:di] to the memory block into which the read data will be stored.

Related Topics

Write persistent User Data

Top of list
Index page

Interrupt 0xA0 service 0x50: Get IP address of the PPP Server

Get the IP address as a string.

Parameters

AH
Must be 0x50.

ES:DX
Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) this function returns an empty string.

Top of list
Index page

Interrupt 0xA0 service 0x55: Get IP address of the PPP Client

Get the IP address as a string.

Parameters

AH
Must be 0x55.

ES:DX
Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) this function returns an empty string.

Top of list
Index page

Interrupt 0xA0 service 0x56: Get sprintf address

Get the address of the internal RTOS `sprintf` function.

Parameters

AH
Must be 0x56.

Return Value

AX=0
ES:DI points to the internal `sprintf` function of the @CHIP-RTOS

Comments

This function is used by our Clib library to provide a `printf` function for user applications.

Top of list
Index page

End of document

TCP/IP API

This interface provides access to the IPC@CHIP TCP/IP stack's socket interface for programming TCP/IP applications. The TCP/IP API are all reached though software interrupt 0xAC. The desired service is selected with the high order byte of the AX register (AH).

Please note, that we cannot explain the entire functionality of the TCP/IP protocol and the working of the socket interface in this document. Good books for understanding TCP/IP and the socket interface are:

1. Internetworking with TCP/IP, Volume 1-3 from Douglas E.Comer
2. TCP/IP Illustrated, Volume 1 from W. Richard Stevens

For some useful comments see **Programming notes**.

All needed constants (e.g. API_OPENSOCKET) and data structures are defined in the header file tcpipapi.h.

TCP/IP API **News**
TCP/IP API **Error Codes** Listing
TCP/IP API **Developer** Notes
TCP/IP API **Data Structures**
TCP/IP API **Client/Server applications**

Notes :

- "Network byte order" is big endian (like Motorola machines, unlike Intel), where most significant byte appears at the lowest address byte of a multibyte integer.
- At return of most API the DX-Register is used for error checking:
  DX: = 0 =API_ENOERROR ==>success
  DX: = -1 =API_ERROR ==>error, AX contains error code

API Functions :

- **Interrupt_0xAC_function_0x01:_API_OPENSOCKET, open a socket**
- **Interrupt_0xAC_function_0x02:_API_CLOSESOCKET, close a socket**
- **Interrupt_0xAC_function_0x03:_API_BIND, bind TCP or UDP server socket**
- **Interrupt_0xAC_function_0x04:_API_CONNECT, Connect to another socket**
- **Interrupt_0xAC_function_0x05:_API_RECVFROM, Receive UDP message**
- **Interrupt_0xAC_function_0x06:_API_SENDTO, Transmit a UDP datagram**
- **Interrupt_0xAC_function_0x07:_API_HTONS, Convert byte order**
- **Interrupt_0xAC_function_0x08:_API_INETADDR, Convert an IP-String to binary**
- **Interrupt_0xAC_function_0x09:_API_SLEEP, Sleep**
- **Interrupt_0xAC_function_0x0A:_API_MALLOC, Allocate memory**

Interrupt 0xAC service 0x01: API_OPENSOCKET, open a socket

Creates an end-point for communication and returns a socket descriptor (i.e. a handle).

Parameters

AH

0x01 (= API_OPENSOCKET)

AL

type of socket :
AL = 1 (= SOCK_STREAM) ==> TCP
AL = 2 (= SOCK_DGRAM) ==> UDP

Return Value

DX = 0 success AX: socket descriptor
DX != 0 AX: contains **error** code

Comments

This function provides the BSD socket() functionality.

Related Topics

Close socket **API_CLOSESOCKET**

Top of list
Index page

Interrupt 0xAC service 0x02: API_CLOSESOCKET, close a socket

Closes the socket indicated by BX and releases all of its associated resources.

Parameters

AH

0x02 (= API_CLOSESOCKET)

BX

Socket descriptor

Return Value

DX = 0 success AX: 0
DX = -1 = API_ERROR AX: contains **error** code

Related Topics

Open socket **API_OPENSOCKET**

Top of list
Index page

Interrupt 0xAC service 0x03: API_BIND, bind TCP or UDP server socket

Bind a unnamed socket with an IP address and port number.

Parameters

AH
0x03 (= API_BIND)

BX
Socket descriptor

DX:SI
Pointer to a `sockaddr_in` **structure**, the first three elements of which must be filled in by caller prior to calling.

Return Value

DX = 0 success AX: 0
DX != 0 AX: contains **error** code

Comments

The bind call sets a specific port number as an application's source port number. Otherwise a random 16-bit source port number will be used when no bind call is made.

The bind call is necessary for server applications in order that clients can connect to them at the agreed upon ("well known") port number.

The older TCP and UDP echo client examples also used a bind call, but this was not necessary. If you use the bind call in a client application, the client uses the given port number as its own source port number.

Related Topics

Get IP address of the **Ethernet** interface
Convert ASCII IP address to **binary**
For proper port number byte order: **htons** function

Top of list
Index page

Interrupt 0xAC service 0x04: API_CONNECT, Connect to another socket

For TCP only. The connect call attempts to make a connection to another socket (either local or remote). This call is used by a TCP client.

Parameters

AH
0x04 (= API_CONNECT)

BX
Socket descriptor

DX:SI
Pointer to a `sockaddr_in` structure containing host's IP address and port number.

Return Value

DX = 0 success AX: 0
DX != 0 AX: contains **error** code

Comments

The caller must fill in the first three elements of the `sockaddr_in` data **structure** at [DX:SI] prior to calling here.

Top of list
Index page

Interrupt 0xAC service 0x05: API_RECVFROM, Receive UDP message

For UDP only, receives message from another socket.

Parameters

AH
0x05 (= API_RECVFROM)

BX
Socket descriptor

DX:SI
Pointer to a `recv_params` data **structure** which caller must fill in prior to call.

Return Value

DX = 0 success AX: number of received bytes where 0 bytes indicates a timeout
DX != 0 AX: contains **error** code

Comments

This function will output the received data to the buffer referenced by `recv_params.bufferPtr` pointer. Up to `recv_params.bufferLength` bytes will be accepted. The AX return value indicates the number of bytes

put into the buffer.

The sender can be identified by the IP address and port number reported in the `sockaddr_in` **structure**
API_RECVFROM can only return complete UDP datagrams at the user provided buffer. If the next waiting datagram at the internal socket receive queue has a higher size than the specified `recv_param.bufferlength`, API_RECVFROM returns errorcode 240 (Message too long) without writing the datagram into the user buffer. You can use **API_GETRCV_BYTES** to detect the size of the next waiting datagram at the internal receive queue of this socket.

Related Topics

API_SENDTO - Send UDP datagram
API_RECV - Receive TCP data
API_GETRCV_BYTES - Get waiting bytes on a socket

Top of list
Index page

Interrupt 0xAC service 0x06: API_SENDTO, Transmit a UDP datagram

For UDP only, transmits message to another transport end-point.

Parameters

AH
0x06 (= API_SENDTO)

BX
Socket descriptor

DX:SI
Pointer to a `send_params` **structure** which caller must fill in prior to call.

Return Value

DX = 0 success AX: number of bytes sent
DX != 0 AX: contains **error** code

Comments

This function will output up to `send_params.bufferLength` bytes from the buffer at `send_params.bufferPtr` to the IP address specified by the `sockaddr_in` **structure** referenced by the `send_params.toPtr` pointer. The return value indicates the actual number of bytes sent.

Related Topics

API_RECVFROM - Receive UDP datagram
API_SEND - Send TCP data

Top of list
Index page

Interrupt 0xAC service 0x07: API_HTONS, Convert byte order

Converts a short (16 bit) value from host byte order to **network byte order**.

Parameters

AH
0x07 (= API_HTONS)

BX
short value

Return Value

DX = 0, AX contains converted value

Comments

This is used to convert port numbers; e.g. htons(7).

Top of list
Index page

Interrupt 0xAC service 0x08: API_INETADDR, Convert an IP-String to binary

Converts a dotted decimal IP address string to an unsigned long in **network byte order**.

Parameters

AH
0x08 (= API_INETADDR)

BX:SI
Pointer to the IP dotted decimal string set by caller. This ASCII string must be zero terminated.

ES:DI
Pointer to a 32 bit unsigned long variable, where this function outputs the converted IP address value.

Return Value

DX = 0 AX = 0, [ES:DI] contains converted 32 bit binary value
DX != 0 AX = **error** code, syntax error

Related Topics

Get IP address of the **Ethernet** interface
Convert binary IP address to **ASCII** dotted decimal

Top of list
Index page

Interrupt 0xAC service 0x09: API_SLEEP, Sleep

The application sleeps for a specified number of milliseconds.

Parameters

AH
0x09 (= API_SLEEP)

BX
milliseconds

Return Value

DX = 0, AX = 0

Comments

This call has been superceded by the similar RTOS API call **RTX_SLEEP_TIME**. This call here is maintained in the TCP/IP API for compatibility with earlier @CHIP-RTOS version.

For new applications, the RTX function with more meaningful return values is recommended.

Top of list
Index page

Interrupt 0xAC service 0x0A: API_MALLOC, Allocate memory

This function allocates memory direct from the @CHIP-RTOS heap.

Parameters

AH
0x0A (= API_MALLOC)

BX
size in bytes

Return Value

DX = 0 , AX = 0, ES:DI points to the allocated buffer
DX != 0 allocation error, ES:DI is a NULL-Pointer

Comments

This function has been replaced by the DOS **Int21h 0x48** API and is maintained here in the TCP/IP API only for compatibility with earlier @CHIP-RTOS versions (prior to version 1.00).

For new applications, the DOS interrupt 0x21 function is recommended.

Top of list
Index page

Interrupt 0xAC service 0x0B: API_FREE, Free an allocated buffer

Releases a block of allocated memory.

Parameters

AH
0x0B (= API_FREE)

DX:SI
Pointer to the buffer

Return Value

DX = 0 , AX = 0 success
DX != 0 , AX = 0 free failed

Comments

This out dated function releases memory allocated with **API_MALLOC**. It is maintained here only for compatibility with earlier @CHIP-RTOS versions (before version 1.00).

For new applications, the DOS interrupt 0x21 memory allocate **service 0x48** and release **service 0x49** are recommended.

Top of list
Index page

Interrupt 0xAC service 0x0C: API_GETRCV_BYTES, Get waiting bytes count

Get the number of bytes on a socket, waiting for read.

Parameters

AH
0x0C (= API_GETRCV_BYTES)

BX
socket descriptor

Return Value

DX = 0 success, AX contains the number of bytes ready
DX != 0 failed, AX contains **error** code

Comments

If the specified socket is an UDP socket, API_GETRCV_BYTES will return the size (in bytes) of the next waiting datagram at the internal socket receive queue.

Related Topics

API_RECVFROM - UDP Receive
API_RECV - TCP Receive

Top of list
Index page

Interrupt 0xAC service 0x0D: API_ACCEPT, Accept the next incoming connection

This API extracts the first connection on the queue of pending connections (from **API_LISTEN**) and creates a new socket for this connection.

Parameters

AH
0x0D (= API_ACCEPT)

BX
Socket descriptor (must be switched into listen mode using **API_LISTEN**)

DX:SI
Output Parameter: Pointer to a `sockaddr_in` **structure** (see tcpipapi.h)

Return Value

DX = 0 success, AX: contains new socket descriptor for the connection
DX != 0 failure, AX: contains **error** code

Comments

This call is used by a TCP server.

On success, this function fills in the `sockaddr_in` **structure** at [DX:SI] with the IP address and port number of the accepted connection.

The new socket will have the same socket options as the listening socket (BX).

Related Topics

API_SETSOCKOPT - Set socket options

Top of list
Index page

Interrupt 0xAC service 0x0E: API_LISTEN, Listen for incoming connections

Places the socket into passive mode and sets the number of incoming TCP connections that the system will queue.

Parameters

AH
0x0E (= API_LISTEN)

BX
Socket descriptor

CX

The maximum number (limited to 5) of allowed outstanding connections

Return Value

DX = 0 success, AX: 0
DX != 0 failure, AX: contains **error** code

Comments

This call is used by a TCP server.

Related Topics

API_ACCEPT - Accept the next incoming connection

Top of list
Index page

Interrupt 0xAC service 0x0F: API_SEND, Transmit TCP message

For TCP only, this API transmits a message to another transport end-point.

Parameters

AH
0x0F (= API_SEND)

BX
Socket descriptor

DX:SI
Pointer to a `send_params` **structure** which must be filled in by caller.

Return Value

DX = 0 success, AX: number of bytes sent
DX != 0 failure, AX: contains **error** code

Comments

API_SEND may be used only if the socket is in a connected state.

On success, the return value in AX contains the number of bytes which were successfully inserted into the socket's send queue.

Related Topics

API_RECV - Receive TCP data
API_SENDTO - Send UDP datagram

Developer Notes

Since @CHIP-RTOS 1.00 the MSG_DONTWAIT option is available in the `flag` member of `send_params` structure

for the API_SEND function (or `recv_params` structure for API_RECV function). If `flag` is set to MSG_DONTWAIT the send call returns immediately. As much data as fits into the internal TCP buffer is accepted for transmission and the transmit byte count is returned in AX. If none of the data fits then -1 is returned in DX and error code 235 in AX.

If `flag` is set to MSG_BLOCKING, the send call blocks until enough internal buffer space is available or a socket error occurs. By default the blocking mode is set for all sockets at the open call. If a socket has been set to non-blocking with the **API_SETBLOCKINGMODE** function, the MSG_BLOCKING flag is ignored and the call will be non-blocking.

Top of list
Index page

Interrupt 0xAC service 0x10: API_RECV, Receive TCP message

For TCP only, this API receives a message from another socket.

Parameters

AH
0x10 (= API_RECV)

BX
Socket Descriptor

DX:SI
Pointer to `recv_params` **structure** which must be filled in by user.

Return Value

DX = 0 success, AX: number of received bytes; 0 bytes implies timeout
DX != 0 failure, AX: contains **error** code

Comments

API_RECV may only be used when the socket is in a connected state.

On success, the return value in AX contains the number of bytes which were successfully inserted into the caller's receive buffer at `recv_params.bufferPtr`.
The length of the message returned could also be smaller than the parameter bufferlength of the **recv_param structure**. This is not an error.
If `flag` member of `recv_params` structure is set to MSG_DONTWAIT, the API_RECV call returns immediately. If no data is available -1 is returned in DX with error code 235 in AX. If flag is set to MSG_BLOCKING, the API_RECV call waits for a message to arrive.

By default the blocking mode is set for all sockets at the open call. If a socket was set to non-blocking with the **API_SETBLOCKINGMODE** function, the MSG_BLOCKING flag is ignored and the call will be non-blocking.

Related Topics

API_SEND - Send TCP data
API_RECVFROM - Receive UDP datagram

Top of list
Index page

Interrupt 0xAC service 0x11: API_INETTOASCII, Convert IP address to IP string

Converts an unsigned long IP address to an ASCII dotted decimal IP string.

Parameters

AH
0x11 (= API_INETTOASCII)

BX:SI
Input Parameter: Pointer to the 32 bit IP address in **network byte order**

ES:DI
Output Parameter: Pointer to the string buffer, where this function outputs the converted dotted decimal IP string. This buffer must have space for 17 Bytes!

Return Value

DX = 0, AX = 0, [ES:DI] buffer contains converted string value

Comments

The dotted decimal IP address ASCII string output to [ES:DI] buffer is zero terminated.

Related Topics

Convert ASCII dotted decimal IP address to **binary**

Top of list
Index page

Interrupt 0xAC service 0x12: API_RESETCONNECTION, Abort a connection on a socket

Applies only to TCP sockets.

Parameters

AH
0x12 (= API_RESETCONNECTION)

BX
Socket descriptor

Return Value

DX = 0 success AX: 0
DX != 0 failure, AX: contains **error** code

Comments

This API_RESETCONNECTION API does not close the socket, which must be done with API_CLOSESOCKET.

The reset socket can not be used for a new connection. You must close the socket and open a new one to get

a new connection.

Related Topics

Close socket **API_CLOSESOCKET**

Top of list
Index page

Interrupt 0xAC service 0x13: API_SETLINGER, Set linger time on close

Applies only to TCP sockets.

Parameters

AH
0x13 (= API_SETLINGER)

BX
Socket descriptor

CX
Linger time in seconds, default: 60 seconds, 0 means linger turned off.

Return Value

DX = 0 success
DX != 0 failure, AX: contains **error** code

Top of list
Index page

Interrupt 0xAC service 0x14: API_SETREUSE, Set reuse option on a listening socket

Sets **SO_REUSEADDR** socket option to '1'.

Parameters

AH
0x14 (= API_SETREUSE)

BX
Socket descriptor

Return Value

DX =0 success
DX!=0 failure, AX: contains **error** code

Comments

This API applies only to TCP sockets. The reuse option is necessary if a listening socket is closed and then a

new socket is opened and **bound** to the same port as the earlier socket.

Developer Notes

Since @CHIP-RTOS version 071, the **API_OPENSOCKET** call sets the SO_REUSEADDR option to '1' as default for all TCP sockets. So calling this API is no longer necessary.

Top of list
Index page

Interrupt 0xAC service 0x15: API_SETIPTOS, Set IP Type-Of-Service

Sets socket's default *Type-Of-Service* placed in the IP datagram header's TOS field.

Parameters

AH
0x15 (= API_SETIPTOS)

AL
Type-Of-Service for IP datagrams

BX
Socket descriptor

Return Value

DX = 0 success
DX != 0 failure, AX: contains **error** code

Comments

The 8 bits in the IP datagram *Type-Of-Service* field are defined as follows:

Bits 0: unused bit (must be 0)
Bits 1-4: Type of Service, see TCP/IP documentation e.g.
Section 3.2 of "*TCP/IP Illustrated, Volume 1*" by W. Richard Stevens.
At most one of these four bits is set.
Bits 5-7: "Precedence field" (unused today)

Note: Many routers ignore this IP datagram header field.

Top of list
Index page

Interrupt 0xAC service 0x16: API_SETSOCKOPT, Set options on socket

Controls option settings for specfied socket.

Parameters

AH
0x16 (= API_SETSOCKOPT)

BX
Socket descriptor

ES:DI
Pointer to `SetSocketOption` **type** that specifies the socket options (see tcpipapi.h)

Return Value

DX=0 success
DX!=0 AX: contains **error** code

Comments

This API function makes it possible to manipulate options associated with a socket. Prior to calling this function the caller must fill in a `SetSocketOption` type data structure.

The socket options of an incoming connection (using **accept**) will be the same as the socket options from its listening socket.

See `SetSocketOption` **type** definition for example usage of this API function.

Related Topics

`GetSocketOption` **typedef** with option names
API_GETSOCKOPT - Get socket options

Top of list
Index page

Interrupt 0xAC service 0x17: API_GETSOCKOPT, Get options on socket

Reads out an option setting for a specified socket.

Parameters

AH
0x17 (= API_GETSOCKOPT)

BX
Socket descriptor

ES:DI
Pointer to `GetSocketOption` **structure**

Return Value

DX = 0 success, Buffer pointed to by the `optionValue` member of the `GetSocketOption` structure at [ES:DI] contains the requested socket option value.
DX != 0 failure, AX: contains **error** code

Comments

This API function makes it possible to read options associated with a socket.

Prior to calling this function, the caller must fill in a `GetSocketOption` type data structure.
The user must set the `protocol_level` and `optionName` members. Also the pointer `optionValue` must point to a valid buffer in the user application's memory.

The `optionLength` member is both an input and an output parameter. The length of the buffer must be specified at the location referenced by `optionLength` and this length must be sufficient for the size of the requested option data. On success, this API writes to the `optionLength` location the number of bytes output to the `optionValue` buffer.

The socket options of an incoming connection (using **accept**) will be the same as the socket options from its listening socket.

Related Topics

`GetSocketOption` **typedef** with option names
API_SETSOCKOPT - Set socket options

Top of list
Index page

Interrupt 0xAC service 0x18: API_SETBLOCKINGMODE, Set socket mode

Sets a socket in blocking or non-blocking mode

Parameters

AH
0x18 (= API_SETBLOCKINGMODE)

AL
0: switch blocking off, 1: switch blocking on

BX
Socket descriptor

Return Value

DX = 0 success
DX != 0 failure, AX: contains **error** code

Comments

By default all sockets are in blocking mode. If a socket is set to non-blocking mode, socket calls like CONNECT, ACCEPT, ... do not wait until full completion. They will instead return immediately.

Example usage of non-blocking mode:

> The **connect**call returns for a non-blocking socket with -1 in DX and error code 236 if the connection was not completed. The user can call connect within a loop, periodically polling for connection completion condition while performing some other activity.

Top of list
Index page

Interrupt 0xAC service 0x19: API_REGISTERCALLBACK, Register a user callback function

Register a user callback function for events occurring on a TCP socket.

Parameters

AH
0x19 (= API_REGISTERCALLBACK)

BX
Socket descriptor

CX
Event flag mask (see below)

ES:DI
Pointer to callback function (see below)

Return Value

DX = 0 success
DX != 0 AX: contains **error** code

Comments

The TCP/IP stack of the IPC@CHIP is able to execute a registered user callback function, if one or more some specified events happens on a TCP socket connection.

The callback function must be of the following type (Borland C):

```
void huge socketCallBackFunc(int socketdescriptor,
                             int eventFlags)
```

Before closing a socket, you should remove the callback function. This is done by calling this API function with a null pointer in ES:DI and value 0 in register CX.

The input parameter `eventFlags` to the callback is a bit field indicating the event(s) that have occurred.

The set of events for either the CX event mask argument to this API or the `eventFlags` callback parameter are defined in TCPIPAPI.H as follows:

```
#define CB_CONNECT_COMPLT   0x0001   // Connection complete
#define CB_ACCEPT           0x0002   // Remote client has established
                                     //  a connection to our listening server.
#define CB_RECV             0x0004   // Incoming data arrived
#define CB_SEND_COMPLT      0x0010   // Sending of data has been acked by the peer
#define CB_REMOTE_CLOSE     0x0020   // Peer has shutdown the connection
#define CB_SOCKET_ERROR     0x0040   // An error occurred on the connection
#define CB_RESET            0x0080   // Peer has sent a rest on the connection
#define CB_CLOSE_COMPLT     0x0100   // close has been completed
```

Related Topics

API_REGISTERCALLBACK_PASCAL For Pascal callback functions

Top of list
Index page

Interrupt 0xAC service 0x20: API_REGISTERCALLBACK_PASCAL, Pascal user callback

Registers a user callback function written in Pascal for events occurring on a TCP socket.

Parameters

AH
0x20 (= API_REGISTERCALLBACK_PASCAL)

BX
Socket descriptor

CX
Event flag mask (see below)

ES:DI
Pointer to callback function (see below)

Return Value

DX = 0 success
DX != 0 AX: contains **error** code

Comments

The number of Pascal TCP callback functions is limited to 10.

The first action required in your callback function is to read the pointer to the **PacCallBack** record passed in registers ES:DI. This can be accomplished by implementing the callback function as follows (Borland Pascal):

```
procedure socketCallBackFunc; interrupt;
var
    ESReg     : Integer;
    DIReg     : Integer;
    CBParamPtr    : CallBackParamPtr;
begin
    (*********************************)
    (* Required to get the Parameter *)
    asm
        mov ax, es
        mov ESReg, ax
        mov ax, di
        mov DIReg, ax
    end;
    (*********************************)

    [... your code ...]

end;
```

Before closing a socket, you should remove the callback function. This is done by calling this API function with a null pointer in ES:DI and value 0 in register CX.

The event **flags** are defined as follows:

```
const
  CB_CONNECT_COMPLT = $0001;
  CB_ACCEPT = $0002;
  CB_RECV = $0004;
  CB_SEND_COMPLT = $0010;
  CB_REMOTE_CLOSE = $0020;
```

```
CB_SOCKET_ERROR = $0040;
CB_RESET = $0080;
CB_CLOSE_COMPLT = $0100;
```

Related Topics

API_REGISTERCALLBACK For C language callback functions

Top of list
Index page

Interrupt 0xAC service 0x21: API_GET_SOCKET_ERROR, Get last socket error.

Returns the last error which occurred on the specified socket.

Parameters

AH
0x21 (= API_GET_SOCKET_ERROR)

BX
Socket descriptor

Return Value

DX =0: success AX: contains last socket **error** code

Top of list
Index page

Interrupt 0xAC service 0x22: API_GET_TCP_STATE, Find TCP socket and return state.

Search for a TCP socket at a given local port number, returning the socket state, IP address and port number of the remote peer (if any).

Parameters

AH
0x22 (= API_GET_TCP_STATE)

BX
Local port (e.g. htons(23) for telnet)

ES:DI
Pointer to storage 32 Bit remote IP address

Return Value

AL: contains last TCP socket state (see below)
If AL!=20 and AL bigger or equal 2 (a TCP connection is established), the storage at ES:DI holds the 32 bit IP address of the connected remote peer
CX contains remote peer port number

Comments

This function is only available in @CHIP-RTOS version which contain the SNMP MIB feature.

Possible TCP socket states:

0: CLOSED
1: LISTEN
2: SYN_SENT
3: SYN_RECEIVED
4: ESTABLISHED
5: CLOSE_WAIT
6: FIN_WAIT_1
7: CLOSING
8: LAST_ACK
9: FIN_WAIT_2
10: TIME_WAIT
20: INVALID

Top of list
Index page

Interrupt 0xAC service 0x23: API_FINDALL_SOCKETS, Get information about all open sockets.

Find/return information about all open sockets at the internal socket table.

Parameters

AH
0x23 (= API_FINDALL_SOCKETS)

ES:DI
Output Parameter: Pointer to an array of CX `SocketInfo` **structures** in the user's memory space into which this API will report the socket information.

CX
Size of the array (maximum number of table entries reported).

Return Value

AX holds the number of all open sockets. The user provided array at [ES:DI] holds the socket information for up to CX sockets (the lesser of two register counts: AX, CX).

Comments

This function is only available in @CHIP-RTOS versions which contain the SNMP MIB feature.

If CX >= AX then all sockets in the socket table have been reported.

Below is an example of using API_FINDALL_SOCKETS with the `FindAllOpenSockets` C library wrapper function. This example lists up to 64 current entries in the TCP/IP socket table:

```
SocketInfo s[64];
int avail, i;

avail = FindAllOpenSockets(s, 64);
if (avail > 64) avail = 64;

for (i = 0; i < avail; i++)
{
   printf("\r\nSocket index %d Protocol %d, LocalPort %d",
         s[i].socIndex, s[i].protocol, s[i].localport);
}
```

Related Topics

SocketInfo data structure

Top of list
Index page

Interrupt 0xAC service 0x40: PPPCLIENT_INSTALLED, Check if PPP client installed.

Tests if PPP client services are available in this @CHIP-RTOS version and not disabled via CHIP.INI **configuration**.

Parameters

AH
0x40 (= PPPCLIENT_INSTALLED)

Return Value

AX = 0: PPP client is not installed
AX != 0: PPP client is installed

Related Topics

PPPCLIENT_GET_STATUS API - PPP client status

Top of list
Index page

Interrupt 0xAC service 0x41: PPPCLIENT_OPEN, Open a PPP connection

Opens a PPP client connection.

Parameters

AH
0x41 (= PPPCLIENT_OPEN)

ES:DI
Pointer to a `PPPClient_Init` **type** data structure (declared at tcpipapi.h)

Return Value

DX: 0 AX: 0, success [ES:DI] contains the needed IP data for further TCP/IP socket communication
DX: -1 AX: contains **error** code, open failed

Comments

Refer to the PPPCLIE.C example for how to use this API. Also see the `PPPClient_Init` data **structure** documentation.

Note: **Only one PPP client connection can be open at a time!!**

Related Topics

PPPCLIENT_CLOSE - Closes PPP client connection

Top of list
Index page

Interrupt 0xAC service 0x42: PPPCLIENT_CLOSE, Close a PPP client connection

Closes a PPP client connection.

Parameters

AH
0x42 (= PPPCLIENT_CLOSE)

Return Value

DX: 0, AX =0: PPP client connection is closed
DX: -1, AX contains **error** code, Connection close timed out

Comments

At the close call the PPP client also executes after closing the PPP session the modem hang-up commands specified in the `PPPClient_Init` **structure** referenced at the **PPPCLIENT_OPEN** call.
Therefore we recommend waiting approximately 10 seconds before opening a subsequent PPP connection.

Top of list
Index page

Interrupt 0xAC service 0x43: PPPCLIENT_GET_STATUS, Get PPP client status

Gets PPP client's current status.

Parameters

AH
0x43 (= PPPCLIENT_GET_STATUS)

Return Value

AX = -2 (=API_NOT_SUPPORTED), DX =- 2: PPP client **task** is not running
AX = -1 (=API_ERROR), DX = PPP client **status**
AX >= 0, then AX = PPP client status, DX unchanged

Comments

If the PPP client is not supported in this @CHIP-RTOS version, then the value returned in AX is zero and DX is set to -1 (=API_ERROR). In this case an error message is issued to the console.

Related Topics

PPPCLIENT_INSTALLED API - @CHIP-RTOS feature check

Top of list
Index page

Interrupt 0xAC service 0x44: PPPCLIENT_GET_DNSIP, Get DNS IP address

Returns the *Domain Name System* (DNS) addresses as negotiated by the remote PPP server.

Parameters

AH
0x44 (= PPPCLIENT_GET_DNSIP)

BX
1: Get primary DNS address, 2: Get secondary address

ES:DI
Output Parameter: Pointer to an unsigned long variable where the requested DNS IP address will be stored by this API.

Return Value

AX = -2 (=API_NOT_SUPPORTED), DX = -2: PPP client is not **installed**.
AX = -1 (=API_ERROR) DX contains **error** code.
AX = 0, then unsigned long at [ES:DI] holds the DNS IP address.

Comments

The IP address output to [ES:DI] is in **network byte order**.

Top of list
Index page

Interrupt 0xAC service 0x45: PPPCLIENT_SET_OPTIONS, Set options for the PPP client

Sets the PPP options for the PPP client.

Parameters

AH

0x45 (= PPPCLIENT_SET_OPTIONS)

ES:DI

Pointer to an array of `PPP_Option` data **structures**. This array is terminated by first `PPP_Option` structure with a null `optionValuePtr` member (see example below).

Return Value

AX = -3 PPP Connection already established AX = -2 PPP client is not **installed**.
AX = -1 Invalid Option(s) AX = 0 Success

Comments

If you want to use this function, you have to call it before opening a connection (see related links below). Settings options during an established connection will not work!

The settings are only valid for the next established connection. The data structures referenced here with ES:DI must persist until after the subsequent **PPP client open** function completes.

A C-Library function is available in the CLIB file TCPIP.C. Here is an example which sets three PPP options using the C-Library wrapper functions:

```
// Allow remote peer to set primary DNS IP.
unsigned long DNS_Pri_IP  = 0L;
// Allow remote peer to set secondary DNS IP.
unsigned long DNS_Sec_IP  = 0L;
// Allow remote peer to use VJ TCPIP header compression.
unsigned int  ipcp_comp   = 1;

PPP_Option My_Options[] = {
 { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
        (const char *)&ipcp_comp, sizeof(ipcp_comp)},
 { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_PRI,
        (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},
 { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_SEC,
        (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},
   // This last PPP_Option is used to terminate array.
 { 0, 0, 0, NULL, 0}
} ;

//***** Call the C-Library functions as follows ****

// Install options with CLIB function
PPP_Client_SetOptions(&My_Options[0]);  // Point to first member of array

PPP_Client_Open(&pppclient);            // Open the connection
```

Related Topics

PPPCLIENT_OPEN open a PPP connection
PPPCLIENT_CLOSE close a PPP connection

Top of list
Index page

Interrupt 0xAC service 0x50: PPPSERVER_INSTALLED, Check if PPP server installed

Tests if PPP server is available in this @CHIP-RTOS version and not disabled via CHIP.INI **configuration**.

Parameters

AH
0x50 (= PPPSERVER_INSTALLED)

Return Value

AX = 0: PPP server is not installed
AX != 0: PPP server is installed

Top of list
Index page

Interrupt 0xAC service 0x51: PPPSERVER_SUSPEND, Suspend PPP server task

Suspends the PPP server task, "PPPS".

Parameters

AH
0x51 (= PPPSERVER_SUSPEND)

BX
Timeout seconds

Return Value

AX = 0, DX= 0 PPP server is suspended
AX != 0, DX != 0 Suspending PPP server failed, PPP server is not installed or timeout

Comments

Note that the timeout value in BX depends on your timeout entries for the modem **commands** specified in CHIP.INI. If this call returns with -1 in AX and DX, the most likely reason for this is that the modem commands were not finished within the timeout period the timeout specified in BX.

Related Topics

PPPSERVER_ACTIVATE API - Reactivate PPP server
PPPSERVER **HANGUPTIMEOUTx** - CHIP.INI timeout for wait on answer from modem

Top of list
Index page

Interrupt 0xAC service 0x52: PPPSERVER_ACTIVATE, Activate PPP server

Re-activate **suspended** PPP server.

Parameters

AH
0x52 (= PPPSERVER_ACTIVATE)

BX
Timeout seconds

Return Value

AX = 0, DX = 0 PPP server is activated
AX != 0, DX != 0 Activating PPP server failed, PPP server is not installed or timeout

Comments

This API will not activate the PPP server disabled from start up by the CHIP.INI **configuration** entry.

On success, the PPP server is now able to serve a connection.

Note that the timeout value in BX depends on your timeout entries for the modem **commands** specified in CHIP.INI. If this call returns with -1 in AX and DX, the most likely reason for this is that the modem commands were not finished within the timeout period specified in BX.

Related Topics

PPPSERVER_SUSPEND API - Suspend PPP server
PPPSERVER **HANGUPTIMEOUTx** - CHIP.INI timeout for wait on answer from modem

Top of list
Index page

Interrupt 0xAC service 0x53: PPPSERVER_GET_STATUS, Get server state

Gets the current state of the PPP server.

Parameters

AH
0x53 (= PPPSERVER_GET_STATUS)

Return Value

DX != 0: PPP server is not **installed**
DX = 0: AX contains the current PPP server state listed below

Comments

PPP server states:

-1 Error state, should not happen
1 Server disabled
2 Server enabled, waiting for connection
3 PPP connection is established
4 Server tries to hang up modem
5 Server tries to initialize modem

Top of list

Interrupt 0xAC service 0x54: PPPSERVER_GET_CFG, Get PPP server configuration

Gets the current main configuration data of the PPP server.

Parameters

AH
0x54 (= PPPSERVER_GET_CFG)

ES:DI
Output Parameter: Pointer to `PPP_IPCfg_Data` data **structure** where this function will report the configuration data.

Return Value

DX != 0 AX != 0: PPP server is not installed
DX = 0 The user `PPP_IPCfg_Data` structure at [ES:DI] is filled with the PPP server configuration data

Related Topics

PPP_IPCfg_Data **structure** definition

Top of list
Index page

Interrupt 0xAC service 0x55: PPPSERVER_SET_OPTIONS, Set options for PPP server

Sets the PPP options for the PPP server.

Parameters

AH
0x55 (= PPPSERVER_SET_OPTIONS)

ES:DI
Pointer to an array of `PPP_Option` data **structures**. This array is terminated by first `PPP_Option` structure with a null `optionValuePtr` member. (see example below).

Return Value

AX = -3: PPP Connection already established
AX = -2: PPP server is not installed
AX = -1: Invalid Option(s)
AX = 0: Success

Comments

It is only possible to set PPP options when the PPP server is **suspended**. PPP server must be at **status** 1. Setting options while the PPP server is active (state 2,3,4,5) has no effect. The installed options will be applied as the PPP server is **re-activated**. The data structures referenced by ES:DI must persists until that time.

If you want to reset the options, call this function with a null pointer in ES:DI.

A C-Library function for this API is available in the CLIB file TCPIP.C. Here is an example which sets three PPP options using the C-Library wrapper functions:

```
// Allow remote peer to set primary DNS IP.
unsigned long DNS_Pri_IP  = 0L;
// Allow remote peer to set secondary DNS IP.
unsigned long DNS_Sec_IP  = 0L;
// Allow remote peer to use VJ TCPIP header compression.
unsigned int  ipcp_comp   = 1;

PPP_Option My_Options[] = {
 { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
        (const char *)&ipcp_comp, sizeof(ipcp_comp)},
 { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_PRI,
        (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},
 { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_SEC,
        (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},
   // This last PPP_Option is used to terminate array.
 { 0, 0, 0, NULL, 0}
} ;

//***** Call the C-Library functions as follows ****

// Suspend the PPP server
PPP_Server_Suspend(20, &error);
// Install options with CLIB function
PPP_Server_SetOptions(&My_Options[0]);  // Point to first member of array
 // Re-activate the server
PPP_Server_Activate(20, &error);
```

Related Topics

PPPSERVER_SUSPEND suspend PPP server task
PPPSERVER_ACTIVATE activate PPP server
PPPSERVER_GET_STATUS Get status of the PPP server

Top of list
Index page

Interrupt 0xAC service 0x60: API_SNMP_GET, Get internal TCP/IP SNMP variables

Accesses *Management Information Base* (MIB) data structures residing inside the @CHIP-RTOS TCP/IP stack.

Parameters

AH

0x60 (= API_SNMP_GET)

AL

1 Get pointer to **IfMib** data structure (only for the internal Ethernet device interface)
2 Get pointer to **IpMib** data structure
3 Get pointer to **IcmpMib** data structure
4 Get pointer to **TcpMib** data structure
5 Get pointer to **UdpMib** data structure

6 Get pointer to **AtEntry** data structure (only of the internal Ethernet device interface)
7 Get array of far pointers to **IfMib** entries of all current open TCP/IP device interfaces.
(Local Loopback, Ethernet, PPP server, PPP client)
8 Get far pointer to unsigned long sysuptime variable (10 Hz counter).

DS:SI

If parameter AL==7, ES:DI must contain the address of a buffer into which up to CX far pointers to **IpMib** data structures can be reported by this API.

CX

If parameter AL==7, CX specifies the number of far pointers (capacity) which the user buffer at [DS:SI] can hold.

Return Value

DX != 0 AX != 0 : @CHIP-RTOS without internal SNMP MIB variables

DX = AX = 0 ES:DI contains a pointer to the structure unless input parameter AH was 7,
in which case the user provided array (still addressed by DS:SI) contains the far pointers to the `IfMib`entries of the currently installed device interfaces. CX is set to indicate the total number of currently installed devices. If CX is less than it was on entry, then the reported list of pointers is not complete due to lack of user buffer space.

Comments

If this API call is used with AL==1-6, or 8, the pointers are granting direct access to the internal TCP/IP stack counters. It's recommended to execute these calls only one time at the start of your application.

For AL=7, the API call returns the pointers to the interface variables for the currently installed device interfaces. To maintain an up to date array of pointers for all currently opened device interfaces, it is necessary to periodically request (e.g. every 10 seconds) the array of pointers.

Note: These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables. The SNMP MIB variables are not a part of our current official 6 @CHIP-RTOS versions. It is necessary to order directly a @CHIP-RTOS version which includes this feature.

Top of list
Index page

Interrupt 0xAC service 0x65: API_FTP_GET_LOGIN, Get FTP server login counters

Accesses the FTP server login counters.

Parameters

AH

0x65 (= API_FTP_GET_LOGIN)

Return Value

DX != 0 AX != 0 : @CHIP-RTOS doesn't support FTP server
DX = AX = 0
ES:DI contains the address of the 32 Bit (unsigned long) login counter
DS:SI contains the address of the 32 Bit (unsigned long) login fail counter

Top of list
Index page

Interrupt 0xAC service 0x66: API_TELNET_GET_LOGIN, Get Telnet server login counters

Accesses the Telnet server login counters.

Parameters

AH
0x66 (= API_TELNET_GET_LOGIN)

Return Value

DX != 0 AX != 0 : @CHIP-RTOS doesn't support Telnet server
DX = AX = 0
ES:DI contains the address of the 32 Bit (unsigned long) login counter
DS:SI contains the address of the 32 Bit (unsigned long) login fail counter

Top of list
Index page

Interrupt 0xAC service 0x67: API_GET_TELNET_STATE, Test Telnet session active

Check if the Telnet server is currently handling an active Telnet session.

Parameters

AH
0x67 (= API_GET_TELNET_STATE)

Return Value

DX = -1 AX = -1 : @CHIP-RTOS doesn't support Telnet server
DX = 0 AX = 1: Telnet session is active
DX = 0 AX = 0: No Telnet session

Top of list
Index page

Interrupt 0xAC service 0x70: GET_INSTALLED_SERVERS and interfaces

Get information about running servers and interfaces of the IPC@CHIP TCP/IP Stack

Parameters

AH
0x70 (= GET_INSTALLED_SERVERS)

Return Value

Bits of AX and DX contains the requested information

Bit=0 service or device is not available.
Bit=1 service or device is available.

AX:
Bit 0: Ethernet device
Bit 1: PPP server
Bit 2: PPP client
Bit 3: Web server
Bit 4: Telnet server
Bit 5: FTP server
Bit 6: TFTP server
Bit 7: DHCP client

DX:
Bit 0: SNMP MIB variables support
Bit 1: UDP Config server
Bit 2: Ping client

Top of list
Index page

Interrupt 0xAC service 0x71: RECONFIG_ETHERNET, Reconfigure Ethernet interface

Reconfigures Ethernet interface, e.g. after changing the IP configuration

Parameters

AH
0x71 (= RECONFIG_ETHERNET)

Return Value

AX:0 success
else restart failed (should not happen). Error code 237 indicates
that an Ethernet interface configuration was already in progress.

Comments

A new IP configuration set with the prompt **commands** `ip`, `netmask` and `gateway` (or the corresponding @CHIP-RTOS **API** calls) becomes valid after a successful call to this function.

If DHCP is changed from 1 to 0 then a new IP address, subnet mask and gateway should be set with the prompt **commands** `ip`, `netmask` and `gateway` or with the @CHIP-RTOS API interrupt 0xA0 services **0x02**, **0x04**, **0x06** before using this function.

Top of list
Index page

Interrupt 0xAC service 0x72: DHCP_USE, Enable/Disable DHCP usage

Set/Reset DHCP usage by the Ethernet interface.

Parameters

AH
0x72 (= DHCP_USE)

AL
0: DHCP not used, 1:DHCP_USE

Return Value

-- none --

Comments

This entry becomes valid only after rebooting the system or after calling function **0x71**.

If DHCP is changed from 1 to 0 then a new IP address, subnet mask and gateway should be set with the prompt **commands** `ip`, `netmask` and `gateway` or with the @CHIP-RTOS API interrupt 0xA0 services **0x02**, **0x04**, **0x06**.

Top of list
Index page

Interrupt 0xAC service 0x73: DHCP_STAT, Get DHCP status of the Ethernet interface

Gets the DHCP status of the Ethernet interface.

Parameters

AH
0x73 (= DHCP_STAT)

Return Value

AX: 1 System uses DHCP, AX:0 DHCP is not used
DX: 0 System is not configured (is in progress)
DX: 1 System is configured by a DHCP Server
DX: 2 System configure retry failed (or no retry started before)

If DX == 1 ES:DI points to a **UserEthDhcp_Entry** data structure which contains the received DHCP configuration data.

Comments

The returned DHCP configuration data must be treated by the user as read-only information. Do not write to this data structure!

Top of list
Index page

Interrupt 0xAC service 0x74: TCPIP_STATISTICS, Access packet counters

This function returns the address of a structure which contains pointers to network packet counters.

Parameters

AH
0x74 (= TCPIP_STATISTICS)

Return Value

AX: 0, DX:0
ES:DI contains a pointer to the `Packet_Count` **structure**

Comments

The counters `count_all_packets` and `count_all_sended packets` count only Ethernet packets. Other counters also count the packets from and to other devices, e.g. local loopback packets and PPP packets.

The user is free to read and/or reset these counters.

Related Topics

`Packet_Count` structure **type** definition

Top of list
Index page

Interrupt 0xAC service 0x75: PING_OPEN, Open and start ICMP echo requests

This function opens and starts periodic *Internet Control Message Protocol* (ICMP) echo request (ping) to a specified remote host.

Parameters

AH
0x75 (= PING_OPEN)

ES:DI
Pointer to user's `Ping` **structure**, four members of which must be set by caller prior to call.

Return Value

DX: -1, AX contains **error** code, Ping open failed
else
DX: socket descriptor, AX: 0

Comments

Important:
If structure member `count` (set by user) is non-zero, the PING_STATISTICS call closes the ping socket automatically if structure member `transmitted` has reached the `count` value. If structure member `count` is zero, ping process runs until PING_CLOSE is called.

Related Topics

PING_CLOSE API
PING_STATISTICS API

Console **PING** command

Top of list
Index page

Interrupt 0xAC service 0x76: PING_CLOSE, Finish ICMP echo requests

This function stops cyclic ICMP echo request (ping).

Parameters

AH
0x76 (= PING_CLOSE)

BX
Socket descriptor from inside `Ping` **structure** from PING_OPEN call.

Return Value

DX: -1, AX -1, Ping close failed, should not happen, only if socket descriptor is invalid
else
DX: 0, AX: 0: success

Comments

The ping socket is closed here.

Related Topics

PING_OPEN API

Top of list
Index page

Interrupt 0xAC service 0x77: PING_STATISTICS, Retrieve ping information

The user can retrieve ping information by calling PING_STATISTICS.

Parameters

AH
0x77 (= PING_STATISTICS)

ES:DI
Pointer to user's `Ping` **structure** from PING_OPEN call

Return Value

DX = 0, AX = 0: Success
DX = -1, AX = -1: Failure

Comments

Structure at [ES:DI] is filled with the ping statistics on success.

Important:
If structure member `count` (set by user) is non-zero, the PING_STATISTICS call closes the ping socket automatically if structure member `transmitted` has reached the `count` value. If structure member `count` is zero, ping process runs until PING_CLOSE is called.

Related Topics

PING_OPEN API
PING_CLOSE API

Top of list
Index page

Interrupt 0xAC service 0x78: GETMEMORY_INFO, Report TCP/IP memory usage

This function reports the maximum available and the currently used memory of the TCP/IP stack.

Parameters

AH
0x78 (= GETMEMORY_INFO)

ES:DI
Output Parameter: Pointer to user's unsigned long where this API will report the maximum available TCP/IP memory byte count.

DS:SI
Output Parameter: Pointer to user's unsigned long where this API will report the currently used TCP/IP memory byte count.

Return Value

DX: 0, AX 0 (The described memory sizes are stored at [ES:DI] and [DS:SI])

Comments

The maximum available memory for the TCP/IP stack can be configured in `chip.ini` (see **TCPIPMEM**).

The amount of memory required by the TCP/IP stack depends on the number of open sockets and the size and number of transported data packets. For memory blocks equal or smaller than 4096 bytes, the TCP/IP stack allocates memory from this pre-allocated block. Once allocated, the TCP/IP stack does not release this memory back to the system. It will be internally recycled for further usage.

The default size of this memory block is 90 kBytes in the IPC@CHIP Large version and 98 kBytes in the @CHIP-RTOS version including PPP.

Memory blocks larger than 4096 bytes are allocated directly from the @CHIP-RTOS memory. The TCP/IP stack releases these blocks back to the IPC@CHIP memory management. If your application requires a lot of memory you should avoid sending and receiving frames larger than 2048 bytes. Larger packets should be split into some smaller ones prior to sending.

With **BIOS** interrupt 0xA0 it is possible to install a user error handler function, which will be called if the memory limit is reached.

Related Topics

BIOSINT API **Install a user fatal error handler**
IPC@CHIP **TCP/IP memory** `chip.ini` Configuration

Top of list
Index page

Interrupt 0xAC service 0x79: SET_SERVER_IDLE_TIMEOUT, Control FTP/Telnet timeout

Set/Get the Idle timeout for either the FTP or Telnet server.

Parameters

AH
0x79 (= SET_SERVER_IDLE_TIMEOUT)

AL
0: Set Timeout, Non-zero: Get Timeout

BX
0: FTP Server, 1: Telnet Server

DX
Timeout value in seconds (If AL = 0 for set)

Return Value

Set idle timeout -
DX = AX = 0: Success
AX = DX = -1: Server not provided

Get idle timeout -
AX = 0: DX contains timeout value
AX = DX = -1: Server not provided

Comments

If input parameter AL is zero, this API inserts the new timeout value in DX into the CHIP.INI. A reboot is not necessary for the new value to take affect.

If AL is non-zero, this API returns the existing timeout for the specified server.

Related Topics

FTP Timeout in `chip.ini` Configuration
Telnet Timeout in `chip.ini` Configuration

Top of list
Index page

Interrupt 0xAC service 0x7A: IP_USER_CB, Install IP callback function

This function allows the application programmer to install an IP callback function.

Parameters

AH
0x7A (= IP_USER_CB)

ES:DI
Pointer to IP callback function (or set to null to remove a previously installed callback)

Return Value

DX: 0, AX 0 success

Comments

The application programmer can implement a function of the type (Borland C):

```
typedef int (huge *MyIpCallbackFuncPtr)(
                IpUserCallbackInfo_t far *ipInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any incoming IP packet. Inside of this function the user is able to check the given IP parameters (which are available through the **IpUserCallbackInfo_t** parameter) and decide whether the TCP/IP stack should process this packet or ignore it.

If the callback function returns -1 the incoming packet will be ignored by the TCP/IP stack.

To remove the callback function, this API call must be called with a null pointer in ES:DI. ***Do not forget to uninstall the callback, if your application exits!***

Top of list
Index page

Interrupt 0xAC service 0x7B: ARP_USER_CB, Install ARP callback function

This function allows the application programmer to install an ARP callback function.

Parameters

AH
0x7B (= ARP_USER_CB)

ES:DI
Pointer to ARP callback function (or set to null to remove a previously installed callback)

Return Value

DX: 0, AX 0 Success

Comments

The application programmer can implement a function from the type described below:

```
typedef int (huge *MyArpCallbackFuncPtr)(
                ArpUserCallbackInfo_t far *arpInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any incoming ARP

packet. The input parameter to the callback function allows access to the ARP packet. Inside of this function the user is able to check the content of the incoming ARP packet and decide whether the TCP/IP stack should process this packet or ignore it. If the callback function returns -1, the incoming packet will be ignored by the TCP/IP stack.

For structure of an ARP packet see **ArpHeader**.

To remove the callback function, this API call must be called with a null pointer in ES:DI. ***Do not forget to uninstall the callback, if your application exits!***

Top of list
Index page

Interrupt 0xAC service 0x80: ADD_DEFAULT_GATEWAY, Add the default gateway

This function is used to add the default gateway for all interfaces.

Parameters

AH
0x80 (= ADD_DEFAULT_GATEWAY)

BX
Device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client, 3: User device driver

ES:DI
Pointer to an unsigned long containing the gateway IP in **network byte order**.

DX:SI
If BX == 3 DX:SI must contain **User device handle pointer**.

Return Value

DX: 0, AX 0 success
DX: -1, AX contains **error** code

Comments

If this function is used, the gateway entry in the `chip.ini` becomes invalidated, but unchanged.

If the PPP server or PPP client is specified in BX, the gateway is set to the remote peer IP address.

Related Topics

DEL_DEFAULT_GATEWAY API function
DEV_OPEN_IFACE API function
IP **Gateway** `chip.ini` Configuration
PPP server **Gateway** `chip.ini` Configuration

Top of list
Index page

Interrupt 0xAC service 0x81: DEL_DEFAULT_GATEWAY, Delete the default gateway

This function is used to delete the default gateway.

Parameters

AH
0x81 (= DEL_DEFAULT_GATEWAY)

ES:DI
Pointer to an unsigned long containing the gateway IP to be deleted in **network byte order**.

Return Value

DX: 0, AX 0 success
DX:-1, AX contains **error** code

Comments

If this function is used, the gateway entry in the `chip.ini` becomes invalidated, but unchanged.

Related Topics

ADD_DEFAULT_GATEWAY API function
IP **Gateway** `chip.ini` Configuration
PPP server **Gateway** `chip.ini` Configuration

Top of list
Index page

Interrupt 0xAC service 0x82: GET_DEFAULT_GATEWAY, Get the current default gateway

This function is used to get the default gateway for all interfaces.

Parameters

AH
0x82 (= GET_DEFAULT_GATEWAY)

ES:DI
Output Parameter: Pointer to user's unsigned long variable where the gateway IP will be written.

Return Value

DX = 0, AX = 0: Success. Location at [ES:DI] contains the gateway IP in **network byte order**.
DX = -1: AX contains **error** code

Top of list
Index page

Interrupt 0xAC service 0x83: ADD_STATIC_ROUTE, Add a route for an interface

This function is used to add a route for an interface. It allows packets for a different network to be routed by the interface.

Parameters

AH
0x83 (= ADD_STATIC_ROUTE)

BX
Device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client, 3: User device driver

DX:SI
If BX = 3, DX:SI must contain **User device handle pointer**.

ES:DI
Pointer to user `Route_Entry` **structure**

Return Value

DX = 0, AX = 0: Success
DX = -1: AX contains **error** code

Comments

The `Route_Entry` structure is defined in tcpipapi.h:

Related Topics

DEL_STATIC_ROUTE API function

Top of list
Index page

Interrupt 0xAC service 0x84: DEL_STATIC_ROUTE, Delete a route for an interface

This function is used to delete a route from an interface.

Parameters

AH
0x84 (= DEL_STATIC_ROUTE)

ES:DI
Pointer to user `Route_Entry` **structure**

Return Value

DX: 0, AX 0 success
DX:-1, AX contains **error** code

Comments

Only structure members `destIPAddress` and `destNetmask` must be valid at the function's call.

Related Topics

ADD_STATIC_ROUTE API function

Top of list
Index page

Interrupt 0xAC service 0x88: DEL_ARP_ENTRY_BY_PHYS, Delete by physical address

This function deletes the entry from the ARP table by the given physical address.

Parameters

AH
0x88 (= DEL_ARP_ENTRY_BY_PHYS)

ES:DI
Pointer to physical address buffer (6 Byte)

Return Value

DX: 0, AX 0 success
DX: -1, AX contains **error** code

Related Topics

ADD_ARP_ENTRY Add ARP entry
GET_ARPROUTE_CACHE Get ARP cache

Top of list
Index page

Interrupt 0xAC service 0x89: ADD_ARP_ENTRY, Add an entry to the ARP table

This function adds an entry into the *Address Resolution Protocol* (ARP) table.

Parameters

AH
0x89 (= ADD_ARP_ENTRY)

BX:SI
Pointer to IP address (unsigned long)

ES:DI
Pointer to physical address (6 Byte)

Return Value

DX = 0, AX = 0: Success
DX = -1: Failure, AX contains **error** code

Related Topics

DEL_ARP_ENTRY_BY_PHYS Delete ARP entry by physical address
GET_ARPROUTE_CACHE Get ARP/ROUTE cache

Top of list
Index page

Interrupt 0xAC service 0x8A: GET_ARPROUTE_CACHE, Read ARP/Route cache table

This function returns all valid entries in the internal ARP/Route cache table.

Parameters

AH
0x89 (= GET_ARPROUTE_CACHE)

BX:SI
Output Parameter: Pointer to a user provided array of 64 of `ArpRouteCacheEntry` data **structures** for storing ARP/Route cache table.

Return Value

DX = 0, AX = 0, the user provided array at [BX:SI] contains the current ARP/Route cache table.
CX holds the number of valid entries in table.

Comments

The array must provide 64 entries. The API call returns only a copy of the current device interface variables. In order to always have the current table status, it is necessary to cyclically (e.g. every 10 seconds) request the table with this API.

Related Topics

DEL_ARP_ENTRY_BY_PHYS Delete ARP entry by physical address
ADD_ARP_ENTRY Add ARP entry

Top of list
Index page

Interrupt 0xAC service 0x8D: GET_IFACE_ENTRIES, Read table of TCP/IP device interfaces

This function returns information about all installed TCP/IP device interfaces.

Parameters

AH
0x8D (= GET_IFACE_ENTRIES)

BX:SI
Pointer to a user provided array of `Iface_Entry` data **structures** for storing the information.

CX
Number of data structures in array at [BX:SI]. Up to this number of device interfaces will be reported.

Return Value

DX = 0, the user provided array at [BX:SI] contains the current device interface information
AX = Number of entries reported in array at [BX:SI].

Top of list
Index page

Interrupt 0xAC service 0x90: ADD_IGMP_MEMBERSHIP, Install an IP multicast address entry

Install an IP multicast address entry to become a member of an IP multicast group.

Parameters

AH
0x90 (= ADD_IGMP_MEMBERSHIP)

ES:DI
Pointer to multicast IP address of type unsigned long

DS:SI
Pointer to a 6 byte array that contains the corresponded Ethernet address

Return Value

DX = 0, AX = 0: Success
DX = -1: AX contains **error** code, invalid class D address, or no free entry in IGMP table available

Comments

IP Multicasting is the Internet abstraction of hardware multicasting. It allows transmission of IP datagrams to a group of hosts that form a single multicast group. Membership in a multicast group is dynamic. Hosts may join or leave the group at any time. Each multicast group has unique IP multicast address (Class D address). The first four bits of an IP multicast address must match to binary 1110. IP multicast addresses range from 224.0.0.0 through 239.255.255.255.

For the usage of IP multicasting on an Ethernet interface, IP multicast addresses must be mapped to Ethernet hardware addresses. The Ethernet device of the IPC@CHIP will be switched into the Ethernet multicast mode. In this mode it receives any incoming IP packet with the mapped Ethernet multicast address and forwards it to the TCP/IP layer. Each IP multicast packet will be sent with the mapped Ethernet multicast address.

Because of the feature, that a multicast IP packet will be received by any member of a multicast group, sending and receiving of IP multicast packets is only usable with UDP sockets (datagram sockets).

After installing a IP multicast address with this API, the application programmer is able use this address as a destination address when sending datagrams. A UDP socket is able to receive datagrams at the specified multicast address.

The maximum number of supported IP multicast addresses in the IPC@CHIP is limited to 15. Before installing an IP multicast address, you can find out the corresponding Ethernet multicast address with **Map IP multicast address to Ethernet address** API.

Using multicast addresses is only possible on the Ethernet interface.

Related Topics

API function **DROP_IGMP_MEMBERSHIP** - Delete an IP multicast address entry
API function **MCASTIP_TO_MACADDR** - Map IP multicast address to Ethernet address

---

Developer Notes

This implementation does not support multicast routing. Sending and receiving multicast datagrams works only at local network.

Top of list
Index page

Interrupt 0xAC service 0x91: DROP_IGMP_MEMBERSHIP, Delete an IP multicast address entry

Delete an IP multicast address entry, leaving a multicast host group.

Parameters

AH
0x91 (= ADD_IGMP_MEMBERSHIP)

ES:DI
Pointer to multicast IP address of type unsigned long.

Return Value

DX = 0, AX = 0: Success
DX = -1, AX = -1: IP address entry not found

Related Topics

API function **ADD_IGMP_MEMBERSHIP** - Install an IP multicast address entry
API function **MCASTIP_TO_MACADDR** - Map IP multicast address to Ethernet address

Top of list
Index page

Interrupt 0xAC service 0x92: MCASTIP_TO_MACADDR, Map IP multicast address to Ethernet

Maps an IP multicast address to the corresponding Ethernet address.

Parameters

AH
0x92 (= MCASTIP_TO_MACADDR)

ES:DI
Pointer to multicast IP address of type unsigned long

DS:SI

Output Parameter: Pointer to a 6 byte array where the generated Ethernet multicast address will be written.

Return Value

DX = 0, AX = 0: Success, storage at DS:SI contains the generated Ethernet address
DX = -1, AX = -1: Invalid IP address

Comments

This API function computes the MAC address in the following manner: To map an IP multicast address to a corresponding Ethernet multicast address place the low-order 23 bits of the IP multicast address into the low order 23 bits of the special Ethernet multicast address 01 00 5E 00 00 00
e.g. IP multicast address 224.0.0.1 becomes Ethernet address 01 00 5E 00 00 01

Related Topics

API function **ADD_IGMP_MEMBERSHIP** - Install an IP multicast address entry
API function **DROP_IGMP_MEMBERSHIP** - Delete an IP multicast address entry

Top of list
Index page

Interrupt 0xAC service 0xA0: DEV_OPEN_IFACE, Install user device driver

Open and install a user TCP device driver/interface at the internal TCP/IP stack of the @CHIP-RTOS.
Add a new link layer for TCP/IP communication.

Parameters

AH

0xA0 (=DEV_OPEN_IFACE)

ES:DI

Pointer to user filled **DevUserDriver** structure, which contains all needed information (device driver functions, etc.) for the user's driver implementation.

Return Value

DX = 0, AX = 0 (or =236): Success

The `IfaceHandle`member of **DevUserDriver**structure at [ES:DI] contains the valid **Device Handle**, which will be required as a parameter for most of the other device interface API for this device.

If member variables `iface_type`and `use_dhcp`were set to 1, AX should contain code 236. In this case, the user must make the additional call to **DEV_WAIT_DHCP_COMPLETE**API to wait for completion of the IP configuration by DHCP.

DX = -1: AX contains **error** code. Installation failed

Comments

This is the first function to use when adding your own interface to the TCP/IP stack of the @CHIP-RTOS.

The TCP/IP API calls 0xA0 - 0xA7 allow the application developer to implement an additional TCP/IP driver interface for a connected hardware device (e.g. connected UART or an Ethernet controller). This new

interface has its own IP configuration and is used by the TCP/IP stack for IP communication in the same way as the pre-installed internal devices of the IPC@CHIP (e.g. Internal Ethernet, PPP server or PPPclient).

For more detailed description and a generic example see **How to add an TCP/IP device driver/link layer interface**.

On the first DEV_OPEN_IFACE call for installing a TCP/IP device driver, the `IfaceHandle` member of the **DevUserDriver** structure at [ES:DI] must be NULL. If an interface is closed with DEV_CLOSE_IFACE and the user wants to reopen that device interface **(e.g. for changing IP configuration)** , the `IfaceHandle` value set by the system in the first DEV_OPEN_IFACE call must be preserved.

You will find all needed types, constants and prototypes in the CLIB header files `TCPIPAPI.H` and `TCPIP.H`.

Clib `TCPIP.C` implementation of this function:

```
int Dev_Open_Interface( DevUserDriver far * DriverInfo,
                        int * errorcode)
```

Please note: The API calls 0xA0 through 0xA7 for implementing your own TCP/IP device drivers are not part of our six official @CHIP-RTOS versions. For ordering special version including this feature, please contact support@beck-ipc.com.

Related Topics

DEV_CLOSE_IFACE - Close device driver interface
DEV_WAIT_DHCP_COMPLETE - Wait for DHCP IP configuration
Example - Installing device driver

Top of list
Index page

Interrupt 0xAC service 0xA1: DEV_CLOSE_IFACE, Close TCP/IP device driver/interface

Close a TCP device driver/interface, remove an added link layer from the internal TCP/IP stack.

Parameters

AH
0xA1 (=DEV_CLOSE_IFACE)

ES:DI
Device handle from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0: Success.
DX = -1: AX contains **error** code
222: Invalid parameter
236: Closing already in progress
237: Device already closed

Comments

This function closes the device driver/interface. It calls the driver close function and deactivates the layer from the internal TCP/IP stack of the @CHIP-RTOS.

For more detailed description see **How to add a user device driver/link layer interface**

Clib `TCPIP.C` implementation of this function:

```
int Dev_Close_Interface(DevUserIfaceHandle DevHandlePtr,
                        int * errorcode);
```

Related Topics

DEV_OPEN_IFACE - Open device driver interface

Top of list
Index page

Interrupt 0xAC service 0xA2: DEV_RECV_IFACE, Move received data

Receive and process incoming data at the TCP/IP stack.

Parameters

AH
0xA2 (=DEV_RECV_IFACE)

ES:DI
Device handle from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0: Success.
DX = -1: Failure, AX contains **error** code
255: Insufficient memory to complete operation

Comments

This function must be called to process incoming data on the TCP/IP stack. This function in turn calls the driver receive function (specified by the `DevRecv` member of the **DevUserDriver** structure at the **DEV_OPEN_IFACE** call) and inserts the received data into the receive queue of the matching socket (if there is one). Any tasks waiting on the socket (sleeping at **RECV** or **RECVFROM**) are then signaled that there is new data available. Only IGMP and ARP requests are directly processed in the task which does this call.

Do not call this API from inside of an ISR!
This function should be used only in combination with the **Wait for receive event** API call in the user's receiver task for their device driver interface.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

Clib `TCPIP.C` implementation of this function:

```
int Dev_Recv_Interface( DevUserIfaceHandle DevHandlePtr,
                        int * errorcode);
```

Related Topics

Example - Receiver Task which calls this API
Example **Receiver** function which would be called by this API

Top of list
Index page

Interrupt 0xAC service 0xA3: DEV_RECV_WAIT, Wait for received data

Waits for data received signal from Interrupt Service Routine.

Parameters

AH

0xA3 (=DEV_RECV_WAIT)

ES:DI

Device handle from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0: Success
DX = -1: Failure, AX contains **error** code
255: Insufficient memory to complete operation

Comments

This function sleeps until a **DEV_NOTIFY_ISR** signal by the device's ISR indicates that received data is available. This function should be used in combination with the **DEV_RECV_IFACE** API within the user implemented receiver task.

For more detailed description see **How to add a user device driver/link layer interface**

Clib `TCPIP.C` implementation of this function:

```
int Dev_Recv_Wait(DevUserIfaceHandle DevHandlePtr,
                  int * errorcode);
```

Related Topics

Example - Receiver Task

Top of list
Index page

Interrupt 0xAC service 0xA4: DEV_NOTIFY_ISR, Signal from ISR

Notify the TCP/IP stack from inside of an Interrupt Service Routine that there is incoming data available and/or that the device has sent a frame successfully.

Parameters

AH

0xA4 (=DEV_NOTIFY_ISR)

BX
Number of available received packets

CX
Number of sent packets completed (provisional)

ES:DI
Device handle from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0

Comments

This function should only be called from the device driver specific ISR. It can be used to signal a receiver task which is waiting at the **DEV_RECV_WAIT** call. In this case the number of received packets should be set in BX register.

The current implementation does not support a transmit task. The sent packets count in CX is a provision for informing a transmit task about completion of packet transmission.

For more detailed description see **How to add a user device driver/link layer interface**.

CLIB `TCPIP.C` implementation of this function:

```
int Dev_Notify_ISR( DevUserIfaceHandle DevHandlePtr,
                    unsigned int rcvdPackets,
                    unsigned int sentPackets);
```

Related Topics

Example - User device ISR

Top of list
Index page

Interrupt 0xAC service 0xA5: DEV_GET_BUF, Get a buffer from TCP/IP stack

Get a buffer from TCP/IP stack's pre-allocated buffer for storing incoming data from this device

Parameters

AH
0xA5 (=DEV_GET_BUF)

CX
Number of bytes buffer required

ES:DI
Output Parameter: Pointer to a variable of type **DevUserBufferHandle** where the user buffer handle is stored. (This value is for internal use.)

Return Value

DX: 0, AX 0: Success. Returns in DS:SI a pointer to the beginning of the allocated buffer
DX: -1, AX-1, DS=SI=0 (null pointer)

Comments

This call should be only used inside the driver specific receive function (implemented by the user). It allocates a buffer from the internal TCP/IP memory pool which can be used for storing the incoming data.

The usage of this function is optional. It is not required, but is recommended that a user work with these buffers from the TCP/IP memory pool.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

Clib `TCPIP.C` implementation of this function:

```
int Dev_Get_Buffer( DevUserIfaceHandle DevHandlePtr,
                    unsigned char far * far * buf,
                    unsigned int len);
```

Related Topics

Example **Receiver** function
`DevUserBufferHandle` **type** definition
IPC@CHIP **TCP/IP memory** `chip.ini` Configuration

Top of list
Index page

Interrupt 0xAC service 0xA6: DEV_SND_COMPLETE, Signal message send complete

This API signals the TCP/IP stack that the final frame in a message has been sent. (Note that a message can be fragmented into multiple link layer frames.)

Parameters

AH
0xA6 (=DEV_SND_COMPLETE)

ES:DI
Device handle from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0

Comments

This function must be called from the driver send function (implemented by the user) on completion of sending data. After this call, the TCP/IP stack of the @CHIP-RTOS is able to reuse the transmission data buffer.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

Clib `TCPIP.C` implementation of this function:

```
int Dev_Send_Complete(DevUserIfaceHandle DevHandlePtr);
```

Related Topics

Example - User's Device Send Function

Top of list
Index page

Interrupt 0xAC service 0xA7: DEV_WAIT_DHCP_COMPLETE, Wait for DHCP

Waits for completion of IP configuration process by DHCP ("*Dynamic Host Configuration Protocol* "). This call must be used if **DEV_OPEN_IFACE** API was called with DHCP enabled (see **use_dhcp**) in `DevUserDriver` structure.

Parameters

AH
0xA7 (=DEV_WAIT_DHCP_COMPLETE)

CX
Timeout seconds. This value will depend on the DHCP server used. A minimum value of 15 seconds is recommended.

ES:DI
Pointer to same **DevUserDriver** structure used at the **DEV_OPEN_IFACE** call.

Return Value

DX = 0, AX = 0: Success
The variables `IPAddr`and `Netmask`in the **DevUserDriver** structure contain the valid IP configuration. The `Dhcp_Data`structure member points to the full **configuration data**provided by the DHCP server.

DX = -1: AX contains **error** code: 260 Timeout.

Comments

Before calling this function, the device receiver task must be running.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**
You will find all needed types, constants and prototypes in the CLIB header files `TCPIPAPI.H` and `TCPIP.H`.

Clib `TCPIP.C` implementation of this function:

```
int Dev_Wait_DHCP_Complete(
                DevUserDriver far * DriverInfo,
                unsigned int time_s,
                int * errorcode)
```

Related Topics

Example - User's receiver task

Top of list
Index page

TCP/IP API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

TCP/IP API News

The following extensions to the TCP/IP **API** are available in the indicated @CHIP-RTOS revisions.

New in version 1.10B: **ARP_USER_CB, Register an ARP callback function**
New in version 1.10B: **GET_IFACE_ENTRIES, Read internal table of TCP/IP device interfaces**
New in version 1.10B: **Modified: API_SNMP_GET**
New in version 1.10B: **Modified: ADD_STATIC_ROUTE**
New in version 1.10B: **GET_ARPROUTE_CACHE, Read internal ARP cache table**
New in version 1.10B: **API_FINDALL_SOCKETS, Return list of open sockets.**
New in version 1.10B: **DEL_ARP_ENTRY_BY_PHYS, Delete ARP entry by physical address**
New in version 1.10B: **ADD_ARP_ENTRY, Add an entry in the ARP table**
New in version 1.10B: **IP_USER_CB, Register an IP callback function**
New in version 1.10B: **Modified: ADD_DEFAULT_GATEWAY**
New in version 1.10B: **TCP/IP Device driver functions 0xA0-0xA7**

End of document

TCP/IP API Error Codes

Network API error codes returned by **API** calls (stated here in decimal):

- **PPPclient**
- **PPPstatus**

201 Operation not permitted
202 No such file or directory
203 No such process
204 Interrupted system call
205 Input/output error
206 Device not configured
209 Bad file descriptor
210 No child processes
211 Cannot allocate memory
213 Permission denied
214 Bad address
217 File exists
219 Operation not supported by device
220 Not a directory
221 Is a directory
222 Invalid argument
224 No resource available
235 Operation would block
236 Operation now in progress
237 Operation already in progress
238 Socket operation on non-socket
239 Destination address required
240 Message too long
241 Protocol wrong type for socket
242 Protocol not available
243 Protocol not supported
244 Socket type not supported
245 Operation not supported
246 Protocol family not supported
247 Address family not supported by protocol family
248 Address already in use
249 Can't assign requested address
250 Network is down
251 Network is unreachable
252 Network dropped connection on reset
253 Software caused connection abort
254 Connection reset by peer
255 No buffer space available
256 Socket is already connected
257 Socket is not connected
258 Can't send after socket shutdown
259 Too many references: can't splice
260 Operation timed out
261 Connection refused
264 Host is down

265 No route to host

-1 socket call failed
0 no error

PPP client error codes

```
//  Possible client error codes
#define PPP_INV_COMPORT  -1  // Invalid port number or PPP server is active at this port.
                  // This error code also occurs, if
                  // the PPP client is interrupted while dialing
                  // (e.g. user break by setting the flag modem_break
                  //  in the  pppclient_init structure
                  //  or another modem error.
#define PPP_INUSE        -2  // Client is already active.
#define PPP_INV_USER     -3  // Invalid user or password
#define PPP_OPEN_FAIL    -4  // Opening the interface failed.
#define PPP_INV_DEV      -5  // Interface was not found.
#define PPP_IPCFG_FAIL   -6  // Got an invalid IP from the peer.
#define PPP_CONNECT_FAIL -7  // Connection to the peer failed.
#define PPP_CLOSETIMEOUT -8  // Closing connection timed out.
```

Top of list
Index page

PPP client status codes

```
// Possible states of a PPP client connection
#define PPP_NOTAVAIL    -1    // Client is not running
#define PPP_LNKDOWN      0    // Link is down
#define PPP_LNKWILLOPEN  1    // Link opening in progress
#define PPP_LNKUP        2    // Link is established
```

Top of list
Index page
TCP/IP **API** Listing

End of document

TCP/IP Applications

Developer Notes

The given examples may be used and modified by the application programmer. The application programmer should know about how the socket-interface works and these examples can help with gaining that understanding.

Some of the programs are using `int86x` calls with the CPU registers loaded as described. We also provide a C-Library (tcpip.c), which places C wrapper functions around the software interrupt calls.

All program examples built with C-API functions use the files tcpip.c, tcpip.h and tcpipapi.h. The SC12 Beta version contains revised TCP/IP API calls, so you should always use the current TCP/IP API C and H files (tcpipapi.h, tcpip.c, and tcpip.h). These files contain all available API calls.

Available examples:

1. UDPEchoClient, udpclie.c, built with int86x calls
2. UDPEchoServer, udpserv.c, built with int86x calls
3. TCPEchoClient, tcpclie.c, built with int86x calls
4. TCPEchoServer, tcpserv.c, built with int86x calls
5. TCPEchoClient, tcpclie.c, built with C-API functions, using tcpip.c
6. TCPEchoServer, tcpserv.c, built with C-API functions, using tcpip.c
7. Reconfigure Ethernet interface, cfgip.c
8. TCP/IP recv packet counting, pkt_cnt.c
9. PPP server API test, ppps.c
10. PPP client example, pppclie.c

TCP/IP **API**

End of document

Data Structures

Here are the BSD structures and other data types used by the TCP/IP API.
All constants and data structures are defined in the TCPIPAPI.H header file.

Notes:

1. **Byte alignment is required** within all data structures used within the API.
2. The phrase "network byte order" means big endian (like Motorola machines, unlike Intel).

Contents :

ArpHeader

Address Resolution Protocol (ARP) packet header has following form.

```
typedef struct tag_ArpHeader
{
    unsigned int     arpHardware;
    unsigned int     arpProtocol;
    unsigned char    arpHardwareLength;
    unsigned char    arpProtocolLength;
    unsigned int     arpOperation;
    unsigned char    arpSendPhyAddress[6];
    unsigned char    arpSendNetAddress[4];
    unsigned char    arpTargetPhyAddress[6];
    unsigned char    arpTargetNetAddress[4];
} ArpHeader;
```

Related Topics

API function **ARP_USER_CB** - Install ARP callback function

Top of list
Index page

ArpRouteCacheEntry

```
typedef struct tagArpRouteCacheEntry
{
    unsigned int   ifIndex;       // Interface index for this entry
    unsigned long  flags;         // Type/state of entry (see below)
    unsigned long  arpIpAddress;  // Device IP address or network address
    unsigned long  mask;          // Ip network mask for cloned ARP entries
    long           arpTtl;        // TimeToLive in milliseconds
    long           hops;          // Number of routers between this host and route

    union
    {
        struct
        {
            // Device IP address for Local routing entry
            unsigned long  DevIpAddress;
        } LocalNetRouteEntry;
        struct
        {
            // Device Ethernet address for ARP entry
            unsigned char MacAddress[6];
            unsigned char filler[2];
        } ArpEntry;
        struct
        {
            unsigned long  GatewayIpAddress;   // Indirect route
        } GatewayEntry;
    } ArpRouteGwayUnion;
} ArpRouteCacheEntry;
```

Comments

`flags`
This bit field is defined as follows:

Note: Bit 2, 3, and 4 select the view of the `ArpRouteGwayUnion`union.
Bit 2: Indirect route (`GatewayEntry`structure)
Bit 3: Valid ARP entry (`ArpEntry`structure)
Bit 4: Local route (`LocalNetRouteEntry`structure)

Bit 5: Static route
Bit 10, 11: Cloned ARP entry (e.g. ARP entry for default gateway)
Bit 16: Route is up
Bit 17: `arpIpAddress`entry is a host address, 0: network address

Related Topics

API function **GET_ARPROUTE_CACHE** - Read internal ARP/Route cache table

Top of list
Index page

ArpUserCallbackInfo

A structure of this type is passed as an input parameter to an ARP user callback handler.

```
typedef struct ArpUserCallbackInfo
{
   int              size;        // Size of this structure
   unsigned int     dataLength;  // Length of the data in the ARP package
   void    far *    dataPtr;     // Pointer to the ARP data
} ArpCallbackUserInfo_t;
```

Related Topics

ArpHeader data structure
API function **ARP_USER_CB** - Install ARP callback function

Top of list
Index page

---

atEntry

```
typedef struct tagAtEntry {
        long           IfIndex;         // Interface on which this entry maps
        unsigned char  PhysAddress[11]; // Physical address of destination
        unsigned char  PhysAddressLen;  // Length of PhysAddress
        unsigned long  NetAddress;      // IP address of physical address
} atEntry;
```

Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables.

Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

Top of list
Index page

DevUserBuffer

This buffer handle private type is used internally by the TCP/IP stack.

```
typedef void  far *  DevUserBuffer;          // Device buffer handle type
```

Top of list
Index page

DevUserBufferHandle

This type references a buffer handle private type used internally by the TCP/IP stack.

```
typedef DevUserBuffer far *  DevUserBufferHandle; // Pointer to a device buffer handle
```

Related Topics

DevUserBuffer type - Device buffer handle (private)
API function DEV_GET_BUF - Get a buffer from TCP/IP stack

Top of list
Index page

DevUserDriver

```
typedef struct tag_DevUserDriver
{
    int                     DevIndex;        // Internal device index, filled by @CHIP-RTOS
    char far *              DevName;         // Unique device name, max. 13 chars  + 0
    unsigned long           IpAddr;          // IP address
    unsigned long           Netmask;         // Netmask

    DevUserIfaceHandle      IfaceHandle;     // handle to identify the interface internal

    DevUserLinkLayer        LinkLayerHandle; // handle to identify the link layer internal


    unsigned char           iface_type,      // Type of the device driver:
                                             // 0: unknown user specific device,
                                             // 1: device is an Ethernet controller,
                                             // 2: use PPP protocol as a server
                                             //     (currently not supported).
                                             // 3: use PPP protocol as a client
                                             //     (currently not supported).
                                             // 4: use SLIP protocol (currently not supported)

    unsigned char            use_dhcp;       // Boolean, set to 1 if you want to use
                                             //  DHCP server IP configuration.

    UserEthDhcp_Entry far * Dhcp_Data;       // Pointer to DHCP data set by
                                             //  DEV_WAIT_DHCP_COMPLETE API
                                             //  (only applies if use_dhcp=1
                                             //   and iface_type=1 =Ethernet).
                                             //

    void  far *             Reserved1;       // reserved for future extensions
    void  far *             Reserved2;
    void  far *             Reserved3;

    // Set of driver functions which user implements:

    void  far *             DevOpen;         // Driver Open/initialize function,
                                             //  optional (pass a Null pointer if not used).
    void  far *             DevClose;        // Driver Close function,
                                             //  optional (pass a Null pointer if not used).
    void  far *             DevSend;         // Driver send function
    void  far *             DevRecv;         // Driver recv function
```

```
    void  far *            DevGetPhysAddr; // Ethernet driver only, returns MAC address,
                                           //  else pass a Null pointer.
    void  far *            DevFreeRecv;    // Free own allocated buffer,
                                           //  optional (pass a Null pointer if not used).
    void  far *            DevIoctl;       // Currently not supported, pass a Null pointer.

    int   far *            ErrorCode;      // Contains error code, if driver install
                                           //  (API call 0xAC 0xA0) failed.

    void  far *            Reserved4;      // reserved for future extensions
    void  far *            Reserved5;
    void  far *            Reserved6;

} DevUserDriver;
```

Comments

This is the main structure which is needed for installing your own device drivers/interfaces for the TCP/IP stack of the @CHIP-RTOS.

Related Topics

API function **DEV_OPEN_IFACE** - Open/install a user TCP/IP device driver/interface
DevUserIfaceHandle type - `IfaceHandle` structure member
DevUserLinkLayer type - `LinkLayerHandle` structure member
UserEthDhcp_Entry type - `Dhcp_Data` structure member

Top of list
Index page

DevUserIfaceHandle

A handle of this type is returned in the `IfaceHandle` member of the `DevUserDriver` **structure** used in the DEV_OPEN_IFACE API.

```
typedef void far *  DevUserIfaceHandle;   // handle type, to identify a user interface
```

Comments

The data referenced by this handle is private and internal to the @CHIP-RTOS TCP/IP stack.

Related Topics

API function **DEV_OPEN_IFACE** - Open/install a user TCP/IP device driver/interface

Top of list
Index page

DevUserLinkLayer

```
typedef void  far *  DevUserLinkLayer;     // handle type to identify link layer,
                                           //  (needed for internal use)
```

Top of list
Index page

GetSocketOption

This data structure and specified constants are defined in the TCPIPAPI.H library header file.

```
typedef struct tag_getsockopt
{
    int         protocol_level;   // Protocol level: IP level, TCP level or socket level
    int         optionName;       // Option's name
    char  far  *optionValue;      // Pointer to the option value buffer (type varies)
    int   far  *optionLength;     // Length of option value buffer

} GetSocketOption;

 // protocol levels
#define IP_PROTOIP_LEVEL        0
#define IP_PROTOTCP_LEVEL       6
#define SOCKET_LEVEL        0x7fff

 // Remaining constants apply to the .optionName field

 // IP level options
#define IPO_TTL             0x0001
#define IPO_TOS             0x0002

 // TCP level options
#define TCP_NODELAY         0x0001
#define TCP_NOPUSH          0x0004
#define TCP_SLOW_START      0x0200
#define TCP_KEEPALIVE       0x4001
#define TCP_DELAY_ACK       0x4002
#define TCP_KEEPALIVE_INTV  0x4004
#define TCP_KEEPALIVE_CNT   0x4005
#define TCP_FINWT2TIME      0x4006
#define TCP_2MSLTIME        0x4007
#define TCP_MAXRT           0x0010
#define TCP_MAXREXMIT       0x4003

 // Socket level options
#define SO_REUSEADDR       0x0004
#define SO_KEEPALIVE       0x0008
#define SO_SNDBUF          0x1001
#define SO_RCVBUF          0x1002
```

Comments

The type of data found at the location referenced by the `optionValue` member pointer varies depending on the particular socket option that is being dealt with.

When manipulating socket options, the `protocol_level` at which the option resides and the name of the option (`optionName` member) must be specified. The size of the buffer (in bytes) pointed to by the `optionValue` member depends on the option, and this size must be specified in the `optionLength` field. These sizes are stated in the list of options below. The API uses the `optionValue` and `optionLength` members to report option values.

The following list specifies the three different protocol levels and their valid socket options, including a short description and the length of the option value. Protocol level and `optionName` constants referred to here are defined in TCPIPAPI.H header file.

`protocol_level=IP_PROTOIP_LEVEL` (IP level options)

- `optionName=IPO_TOS`: Size: 8 bit, sizeof (char) - Set IP type of service, default 0
- `optionName=IPO_TTL`: Size: 8 bit, sizeof (char) - Set IP time-to-live router hops, default 64

`protocol_level=IP_PROTOTCP_LEVEL` (TCP level options)

- `optionName=TCP_NODELAY`: Size: 16 bit , sizeof (int) - 1: Disable nagle algorithm, default is 0: nagle algorithm is enabled
- `optionName=TCP_NOPUSH`: Size: 16 bit , sizeof (int) - 1: Force TCP to delay sending any TCP data until a full sized segment is buffered in the TCP buffers (useful if continuous large amount of data is sent as with FTP) default is 0
- `optionName=TCP_SLOW_START`: Size: 16 bit, sizeof (int) - 1: Enable the TCP slow start algorithm (default), 0: disabled
- `optionName=TCP_KEEPALIVE`: Size: 16 bit, sizeof (int) - Set idle time seconds before sending keep alive probes. Default 7200 seconds, minimum 10 seconds, maximum 32767 seconds
  Notes:
  1. The socket level option `SO_KEEPALIVE` (see below) must be enabled.
  2. The minimum value was changed in @CHIP-RTOS 070 to 10 seconds (was 7200 seconds in @CHIP-RTOS

069).

- `optionName=TCP_DELAY_ACK`: Size: 16 bit - Set the TCP delay ACK time in milliseconds. Default: 200 milliseconds, minimum 0 ms, maximum 65535 ms
- `optionName=TCP_FINWT2TIME`: Size: 16 bit - Set the maximum amount of time TCP will wait for the remote side to close. Default 600 seconds, minimum 0 seconds, maximum 32767 seconds
- `optionName=TCP_2MSLTIME`: Size: 16 bit - Set the maximum amount of time TCP will wait in the TIME WAIT state, once it has initiated a close. Default 2 seconds, minimum 0 seconds, maximum 32767 seconds
- `optionName=TCP_MAXRT`: Size: 16 bit - Set the TCP/IP timeout in seconds (0=System Default [75 Seconds], -1 = endless). Default 0, minimum 0, maximum 32767 seconds
- `optionName=TCP_MAXREXMIT`: Size: 16 bit - Set the maximum number of TCP/IP send retries. Default 12, minimum 0, maximum 32767
- `optionName=TCP_KEEPALIVE_INTV`: Size: 16 bit - Set keep alive interval probes. Default 75 seconds, minimum 1 second, maximum 600 seconds
  Note:
  This value can not be changed after a connection is established. Also the socket must not be in listen mode.
- `optionName=TCP_KEEPALIVE_CNT`: Size: 16 bit - Set maximum number of keep alive probes before TCP gives up and closes the connection. Default: 8, minimum 0, maximum 32767

`protocol_level=SOCKET_LEVEL`

- `optionName=SO_REUSEADDR`: Size: 16 bit - Enable/disable local address reuse, coding: 1=enable, 0=disable default: enable
- `optionName=SO_KEEPALIVE`: Size: 16 bit - Keep connections alive, coding: 1=enable, 0=disable default: disable
- `optionName=SO_SNDBUF`: Size: 32 bits, sizeof(long) - Socket send buffer size. Default TCP 4096, UDP 2048 bytes. (We recommend a maximum size of 8192 Bytes.)
- `optionName=SO_RCVBUF`: Size: 32 bits, sizeof(long) - Socket input buffer size. Default TCP 4096, UDP 2048 bytes. (We recommend at maximum size of 8192 Bytes.)

Related Topics

API function **API_GETSOCKOPT** - Get socket options
SetSocketOption structure typedef

Top of list
Index page

IcmpMib

```
typedef struct tagIcmpMib
{
   unsigned long       icmpInMsgs;          // Total of ICMP msgs received
   unsigned long       icmpInErrors;        // Total of ICMP msgs received with errors
   unsigned long       icmpInDestUnreachs;
   unsigned long       icmpInTimeExcds;
   unsigned long       icmpInParmProbs;
   unsigned long       icmpInSrcQuenchs;
   unsigned long       icmpInRedirects;
   unsigned long       icmpInEchos;
   unsigned long       icmpInEchoReps;
   unsigned long       icmpInTimestamps;
   unsigned long       icmpInTimestampReps;
   unsigned long       icmpInAddrMasks;
   unsigned long       icmpInAddrMaskReps;
   unsigned long       icmpOutMsgs;
   unsigned long       icmpOutErrors;
   unsigned long       icmpOutDestUnreachs;
   unsigned long       icmpOutTimeExcds;
   unsigned long       icmpOutParmProbs;
   unsigned long       icmpOutSrcQuenchs;
   unsigned long       icmpOutRedirects;
   unsigned long       icmpOutEchos;
   unsigned long       icmpOutEchoReps;
   unsigned long       icmpOutTimestamps;
   unsigned long       icmpOutTimestampReps;
   unsigned long       icmpOutAddrMasks;
   unsigned long       icmpOutAddrMaskReps;
} IcmpMib;
```

Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they will need access to these internal TCP/IP variables.

Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

Top of list
Index page

Iface_Entry

```
typedef struct tag_iface_device
{
     unsigned int  devIndex;       // Internal index number
     char          devName[14];    // Device name, terminated by zero
     unsigned long devIPAddr;      // IP address for this interface
     unsigned long devNetmask;     // Netmask for the route
     unsigned long devDestIpAddr;  // Remote peer address for PPP
     unsigned char PhysAddr[6];    // Physical device address, max. 6 Bytes
     int           devType;        // Type of the device driver:
                                   // 0: unknown,
                                   // 1: Ethernet driver,
                                   // 2: PPP protocol
                                   // 4: SLIP protocol (not supported)
                                   // 5: Internal loopback
     int           devDHCP;        // Interface configured by DHCP? 1:0
     int           devFlag;        // Device status flag
                                   // Bit1 == 1  Device opened
                                   // Bit2 == 1  Device IP config in progress(PPP or DHCP)
                                   // Bit3 == 1  Device open completed
     int           devMTU;         // Max. Transfer Unit
     void far *    reserved;
} Iface_Entry;
```

Related Topics

API function **GET_IFACE_ENTRIES** - Read table of TCP/IP device interfaces

Top of list
Index page

IfMib

```
typedef struct tagIfMib           // Interface table data
{
    long          ifIndex;        // index of this interface
    char          ifDescr[32];    // description of interface
    long          ifType;         // network device type
    long          ifMtu;          // maximum transfer unit
    unsigned long ifSpeed;        // bandwidth in bits/sec
    unsigned char ifPhysAddress[11];  // interface's address
    unsigned char PhysAddrLen;    // length of physAddr: 6
    long          ifAdminStatus;  // desired state of interface, not supported
    long          ifOperStatus;   // current operational status, not supported
    //counters
    unsigned long   devLastChange;      // value of sysUpTime when current state entered
    unsigned long   devInOctets;        // number of octets received on interface
    unsigned long   devInUcastPkts;     // number of unicast packets delivered
    unsigned long   devInMulticastPkts; // number of multicast packets delivered,
```

```
                                        //   not supported.
    unsigned long   devInBroadcastPkts; // broadcasts delivered
    unsigned long   devInDiscards;      // number of broadcasts
    unsigned long   devInErrors;        // number of packets containing errors
    unsigned long   devInUnknownProtos; // number of packets with unknown protocol
    unsigned long   devOutOctets;       // number of octets transmitted
    unsigned long   devOutUcastPkts;    // number of unicast packets sent
    unsigned long   devOutMulticastPkts;// number of multicast packets sent
    unsigned long   devOutBroadcastPkts;// broadcasts sent
    unsigned long   devOutDiscards;     // number of packets discarded with no error
    unsigned long   devOutErrors;       // number of pkts discarded with an error
    unsigned long   devOutQLen;         // number of packets in output queue
    unsigned long   devSpecific;
} IfMib;
```

Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables. The SNMP MIB variables are not a part of our current official 6 @CHIP-RTOS versions. It's necessary to order directly a @CHIP-RTOS version which includes this feature.

Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

Top of list
Index page

---

in_addr

```
struct in_addr
{
    u_long s_addr; // 32 bit netid/hostid address in network byte order
};
```

Related Topics

sockaddr_in data structure

Top of list
Index page

IpMib

```
typedef struct tagIpMib
{
    long                ipForwarding;      // 1
    long                ipDefaultTTL;      // default TTL for pkts originating here
    unsigned long       ipInReceives;      // no. of IP packets received from interfaces
    unsigned long       ipInHdrErrors;     // number of pkts discarded due to header errors
    unsigned long       ipInAddrErrors;    // no. of pkts discarded due to bad address
    unsigned long       ipForwDatagrams;   // number pf pkts forwarded through this entity
    unsigned long       ipInUnknownProtos; // no. of local-addressed pkts w/unknown proto
    unsigned long       ipInDiscards;      // number of error-free packets discarded
    unsigned long       ipInDelivers;      // number of datagrams delivered to upper level
    unsigned long       ipOutRequests;     // number of IP datagrams originating locally
    unsigned long       ipOutDiscards;     // number of error-free output IP pkts discarded
    unsigned long       ipOutNoRoutes;     // number of IP pkts discarded due to no route
    long                ipReasmTimeout;    // seconds fragment is held awaiting reassembly
    unsigned long       ipReasmReqds;      // no. of fragments needing reassembly (here)
```

```
    unsigned long       ipReasmOKs;        // number of fragments reassembled
    unsigned long       ipReasmFails;      // number of failures in IP reassembly
    unsigned long       ipFragOKs;         // number of datagrams fragmented here
    unsigned long       ipFragFails;       // no. pkts unable to be fragmented here
    unsigned long       ipFragCreates;     // number of IP fragments created here
    unsigned long       ipRoutingDiscards;
} IpMib;
```

Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to these internal TCP/IP variables.

Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

Top of list
Index page

IpUserCallbackInfo

A structure of this type is passed as an input parameter to an IP user callback handler.

```
struct IpUserCallbackInfo
{
   int              size;        // Size of this struct
   unsigned long    srcAddr;     // Source IP Address (in network byte order)
   unsigned long    destAddr;    // Destination IP Address (in network byte order))
   unsigned int     srcPort;     // Source Port (in network byte order)
   unsigned int     destPort;    // Destination Port (in network byte order)
   unsigned char    protocol;    // Protocol (see list below)
   int              fragmented;  // 0: it is an unfragmented package, 1: it is a fragment
   unsigned int     dataLength;  // Length of the data in the IP package
                                 //  (only available if package is not fragmented!)
   void far *       dataPtr;     // Pointer to the IP data
                                 //  (only available if package is not fragmented!)
} IpCallbackUserInfo_t;
```

Comments

Protocol types enumerated by the `protocol` member are coded as follows:

1 ICMP (*Internet Control Management Protocol*)
2 IGMP (*Internet Group Management Protocol*)
6 TCP (*Transmission Control Protocol*)
17 UDP (*User Datagram Protocol*)

Note: The IP callback function will be called before defragmenting fragmented IP packages. So if we receive a fragmented package, the IP callback will be called for every received fragment.

Related Topics

API function **IP_USER_CB** - Install IP callback function

Top of list
Index page

Packet_Count

```
typedef struct tag_cnt_packet
{
    unsigned int far * cnt_all_packets;      // count all incoming Ethernet packets
    unsigned int far * cnt_ip_packets;       // count incoming IP packets
    unsigned int far * cnt_arp_packets;      // count incoming ARP packets
    unsigned int far * cnt_tcp_packets;      // count incoming TCP packets
    unsigned int far * cnt_udp_packets;      // count incoming UDP packets
    unsigned int far * cnt_icmp_packets;     // count incoming ICMP packets

    unsigned int far * cnt_all_sended_packets;  // count all sent Ethernet packets
    unsigned int far * cnt_ip_sended_packets;   // count all sent IP packets
    unsigned int far * cnt_arp_sended_packets;  // count all sent ARP packets
    unsigned int far * cnt_tcp_sended_packets;  // count all sent TCP packets
    unsigned int far * cnt_udp_sended_packets;  // count all sent UDP packets
    unsigned int far * cnt_icmp_sended_packets; // count all sent ICMP packets

    unsigned int far * cnt_ip_chksum_errs;   // checksum errors on incoming IP packets
    unsigned int far * cnt_udp_chksum_errs;  // checksum errors on incoming UDP packets
    unsigned int far * cnt_tcp_chksum_errs;  // checksum errors on incoming TCP packets
    unsigned int far * cnt_eth_errs;         // errors on incoming Ethernet packets

} Packet_Count;
```

Comments

The counters `count_all_packets` and `count_all_sended_packets` count only Ethernet packets.
Other counters also count the packets from and to other devices e.g. local loopback packets and PPP packets.

Related Topics

API function **TCPIP_STATISTICS** - Access packet counts.

Top of list
Index page

PasCallBack

```
typedef struct tag_PasCallBack
{
    int         sd;     // socket descriptor
    int         event;  // occurred event
} PasCallBack;
```

Comments

The pointer to this structure will be passed to a Pascal callback function in the registers ES:DI. You can read out the information about the socket and the event which has triggered the callback function.

Related Topics

To register a Pascal callback function see **API_REGISTER_CALLBACK_PASCAL**

Top of list
Index page

Ping

```
typedef struct tag_ping_command
{
   int     sd;                 // Socket descriptor, set by PING_OPEN
   // User must set following four values prior to PING_OPEN API call
   char far *remoteHostNamePtr; // Remote IP
   int     pingInterval;       // seconds
   int     pingDataLength;     // Maximum 1024 bytes
   unsigned long count;        // Limit number of pings sent.  Set to
                               //  zero if ping should run forever (until PING_CLOSE)

   unsigned char pingstate;    // ping socket state, 1: open 0: closed
   // Statistics, filled in by system inside PING_STATISTICS API:
   unsigned long transmitted;  // Sent ping requests count
   unsigned long received;     // Received replies count
   unsigned int lastsenderr;   // Last send error
   unsigned int lastrcverr;    // Last receive error
   unsigned long maxRtt;       // Maximum round trip time (ms), rounded off to 100 ms step
   unsigned long minRtt;       // Minimum round trip time (ms), (100 ms steps)
   unsigned long lastRtt;      // Round trip time (100 ms steps) of
                               //  the last ping request/reply.

} Ping;
```

Comments

Caller to PING_OPEN API must initialize structure members:

- `remoteHostNamePtr`... who to "ping"
- `pingInterval`... block repetition rate in seconds
- `pingDataLength`... size of ping data blocks
- `count`... set to zero, if ping should run forever

The remainder of the data structure is managed within the API functions.

Related Topics

API function **PING_OPEN** - Start ICMP echo requests

Top of list
Index page

PPPClient_Init

```
typedef struct tag_ppp_client
{

   int  port;                   // serial port (0:EXT 1:COM)
   int  auth;                   // 0: no authentication
                                // 1:PAP  Client must send user name and password
                                //        for PAP authentication to the peer
                                // 2:CHAP Client must send user name and password
                                //        for CHAP authentication to the peer
                                // 3:PAP  Client expects PAP user name and password
                                //        from the peer
                                // 4:CHAP Client expects CHAP user name and password
                                //        from the peer

   int  modem;                  // modem usage (0:nullmodem 1:modem)
   int  flow;                   // serial flow control (0: none, 1:XON/XOFF, 2:RTS/CTS)
   long baud;                   // Serial port BAUD rate
   unsigned long idletimeout;   // Closing PPP after idle time seconds
                                //    (0: no closing after idle time)
   char username[50];           // Used if .auth != 0
   char password[50];           // Used if .auth != 0
   void far * dptr;             // dummy ptr

   char PPPClieipAddrStr[16];    // If IP is set to "0.0.0.0" client expect IP from
                                 // the peer, IP is filled in after successful connection
```

```
                                // If IP is set to a string != "0.0.0.0" client
                                // wants to use this IP during the ppp session

   char PPPClieRemipAddrStr[16]; // If RemoteIP is set to "0.0.0.0" client allow
                                // the peer to use its own IP during the PPP session,
                                // the RemoteIP is filled in after successful connection

                                // If RemoteIP is set to a string != "0.0.0.0"
                                // client wants to configure the remote peer with this IP


   char PPPClienetMaskStr[16];   // subnet mask
   char PPPClieipGatewayStr[16]; // gateway

   PPPDial             pppdial[PPP_MAX_DIAL];  // modem/dial entries
   PPP_ModemHangup     modem_hangup;           // modem hang-up commands
   int                 break_modem;            // Flag for breaking SC12 - modem
          // control communication (dialing, waiting for connect)
          // Setting break_modem to 1 breaks current modem control communication
          // between IPC@CHIP and the modem at a PPP client open or close call.
          // The PPP client reads this flag and breaks the dialing, if flag is set.
          // This flag can be set from another task.  It will not break an established
          // PPP link!  Don't forget to clear this flag to zero after breaking.

} PPPClient_Init;
```

Comments

The `PPPClient_Init` structure is used to open a PPP client session.

The flow control mode XON/XOFF applied to PPP has not been tested. It is not advisable to use it. Since @CHIP-RTOS 1.02B XON/XOFF mode is also available if the **DMA** receive mode for the selected serial port is enabled, but due to the internal functionality of DMA it is not possible to immediately detect an XON or XOFF from the peer. Therefore it is possible that an overrun situation can occur at the connected peer (e.g. GSM modem). We now offer this mode because GSM modems (any??) support only XON/XOFF flow control.

The `PPPClient_Init` structure contains an array of the `PPPdial` **structures** used for initializing and dialing a modem. These modem commands will be executed at the start of establishing a connection to a PPP server.

The `PPPClient_Init` structure also contains a `PPP_ModemHangup` **structures** for closing the modem connection.

For how to initialize and use these structures see the PPPCLIE.C example.

Related Topics

API function **PPPCLIENT_OPEN** - Open PPP client session.
PPPdial dial-up command data

Top of list
Index page

PPPDial

```
typedef struct tag_pppdial_init
{
      char far * modemcmd;  // modem command string
      char far * modemans;  // modem answer  string (max. 80 characters + '\0')
      int   timeout;        // seconds, 0 = no time out
      int   retries;        // Maximum number of dial attempts.
      char  expect_send;    // = 0: PPP client sends modem AT command
                            //      and expects modem answer (e.g. OK).
                            // = 1: PPP client expects modem answer
                            //   (e.g. CONNECT) and sends modem command.
} PPPDial;
```

Comments

The PPPDial structure is used during PPP client dial-up.

Related Topics

API function **PPPCLIENT_OPEN** - Open PPP client session
PPPClient_Init PPP client open data structure

Top of list
Index page

PPP_IPCfg_Data

```
typedef struct tag_pppipcfg_data
{
     char          IP[16];          // PPP server IP
     char          RemIP[16];       // Remote IP (given to the client, if connected)
     char          Netmask[16];     // Subnet mask
     char          Gateway[16];     // Gateway
     unsigned int  comport;         // COM port: EXT=0, COM=1
     unsigned int  papauth;         // 0: no authentication 1:PAP 2:CHAP
     unsigned int  modem;           // Analog Modem=1, Null Modem cable=0
     unsigned int  flow;            // Flow control
     long          baud;            // BAUD rate

} PPP_IPCfg_Data;
```

Comments

The PPP_IPCfg_Data structure is used to read out the configuration of the PPP server.

Related Topics

API function **PPPSERVER_GET_CFG** - Get PPP server configuration

Top of list
Index page

PPP_ModemHangup

```
typedef struct tag_pppclie_hangup
{
      char far *modemcmdmode;    // string for switching modem into command mode e.g. +++
      int      delay;            // delay time after switching in seconds
      PPPDial pppdial[PPP_MAX_DIAL];  // modem commands and answer for hang-up procedure

}PPP_ModemHangup;
```

Comments

The PPP_ModemHangup structure is used when closing a PPP modem connection.

Related Topics

API function **PPPCLIENT_OPEN** - Open PPP client session
PPPClient_Init PPP client open data structure

Top of list

PPP_Option

This data structure and associated constants are defined in TCPIPAPI.H header file. It is used to control PPP options for both the PPP server and client.

```
typedef struct tag_ppp_option
{
   int             protocolLevel;     // PPP_LCP_PROTOCOL or PPP_IPCP_PROTOCOL
   int             remoteLocalFlag;   // PPP_OPTION_WANT or PPP_OPTION_ALLOW
   int             optionName;        // From constants defined below
   const char far *optionValuePtr;    // To buffer provided by user
   int             optionLength;      // Number of bytes at *optionValuePtr
} PPP_Option;

 // PPP Protocol levels
#define PPP_LCP_PROTOCOL      0x21c0
#define PPP_IPCP_PROTOCOL     0x2180

 // For .remoteLocalFlag
#define PPP_OPTION_WANT       0
#define PPP_OPTION_ALLOW      1

 // Protocol options for .optionName field
 // LCP protocol
#define PPP_LCP_ACCM           2
#define PPP_LCP_PROTO_COMP     7
#define PPP_LCP_ADDRCTRL_COMP  8

 // IPCP protocol
#define PPP_IPCP_COMP_PROTOCOL  2
#define PPP_IPCP_DNS_PRI       29
#define PPP_IPCP_DNS_SEC       31
```

Comments

The type of data found at the location referenced by the `optionValue` member varies depending on the particular PPP option that is being dealt with.

When manipulating socket options, the `protocol_level` at which the option resides and the name of the option (`optionName` member) must be specified.

The size of the buffer required pointed to by the `optionValue` member depends on the option. These sizes (in bytes) are stated at the descriptions list below. The parameters `optionValue` and `optionLength` are used to access option values.

The structure member `remoteLocalFlag` can have two possible values:
`PPP_OPTION_WANT` or `PPP_OPTION_ALLOW`.

With `remoteLocalFlag` set to `PPP_OPTION_WANT`, the IPC@CHIP sends a PPP configuration request to the connected peer which contains the specified option.

With `remoteLocalFlag` set to `PPP_OPTION_ALLOW`, the IPC@CHIP accepts an incoming PPP configuration request from the connected peer for the specified option.

The following list specifies the PPP options for the two different protocol levels, including a short description and the length of the option needed for setting the `optionLength` member. Protocol level and `optionName` constants referred to here are defined in TCPIPAPI.H

`protocolLevel=PPP_LCP_PROTOCOL` (*Link Control Protocol* level options)

- `optionName=PPP_LCP_ADDRCONTROL_COMP`: Size: 8 bit, sizeof(char) - Activate/Deactivate address control compression, default 0: Deactivated
- `optionName=PPP_LCP_PROTO_COMP`: Size: 8 bit, sizeof(char) - Value 1: Protocol field compression, default 0: No compression
- `optionName=PPP_LCP_ACCM`: Size: 32 bit, sizeof(long) - Set Async control character map. Each bit position of the unsigned long value represents the corresponding character which should be escaped or not during PPP session.

`protocolLevel=PPP_IPCP_PROTOCOL` (IpCp level options)

- `optionName=PPP_IPCP_COMP_PROTOCOL`: Size: 16 bit, sizeof(int) - Value 1: VJ TCP/IP header compression, default 0: No compression

- `optionName=PPP_IPCP_DNS_PRI`: Size: 32 bit, sizeof(long) - Specifies the IP addresses of the primary DNS server we will allow the remote to use or the primary DNS server we want to use. If this option is called with the value 0 and `remotelocalflag==PPP_OPTION_WANT`, then we allow the connected peer to report their primary DNS server IP.
- `optionName=PPP_IPCP_DNS_SEC`: Size: 32 bit, sizeof(long) - Specifies the IP addresses of the secondary DNS server we will allow the remote to use or the secondary DNS server we want to use. If this option is called with the value 0 and `remotelocalflag==PPP_OPTION_WANT`, then we allow the connected peer to report their primary DNS server IP.

Related Topics

API function **PPPCLIENT_SET_OPTIONS** - Set PPP client options
API function **PPPSERVER_SET_OPTIONS** - Set PPP server options
API function **PPPCLIENT_GET_DNSIP** - Get DNS IP

Developer Notes

If the option PPP_IPCP_PROTOCOL (VJ TCP/IP header compression) is set, it can be possible that the FTP server of the IPC@CHIP is not usable via the PPP interface.

Top of list
Index page

---

recv_params

```
struct recv_params
{
   char          far      *bufferPtr;        // Store incoming data here
   int                     bufferLength;     // Maximum bytes to store
   int                     flags;            // Blocking, timeout or no wait
   struct sockaddr  far    *fromPtr;         // Only needed for UDP.
   int              far    *fromlengthPtr;   // Only needed for UDP.
   unsigned long           timeout;          // timeout milliseconds
};
```

Comments

This structure is defined in tcpip.h.

`bufferPtr`
This output parameter specifies the user buffer into which the received data will be placed by the system.

`bufferLength`
Size of user buffer at `bufferPtr`, places upper limit on bytes transferred.

`flags`
Set to one of following constants defined in tcpip.h:
MSG_BLOCKING: Sleep until data comes in
MSG_TIMEOUT : The caller wakes up after timeout or if data comes in. The structure member `timeout`must be set.
MSG_DONTWAIT: Return immediately

`fromPtr` UDP only
This is an output parameter. The referenced `sockaddr`**structure**is cast to a `sockaddr_in`**structure**prior to usage. This field will be set by the system inside the **API_RECVFROM**call to indicate the origin of the received UDP data.

`fromlengthPtr` UDP only
The 16 bit value referenced by this pointer is both an input and output parameter, which should be preset to:

```
sizeof(struct sockaddr_in)
```

prior to calling **API_RECVFROM**.

timeout
If `flags`is set to MSG_TIMEOUT, then the maximum number of milliseconds to wait should be specified here.

Top of list
Index page

Route_Entry

```
typedef struct tag_route_entry{
     unsigned long destIPAddress;  // The IP address to add the route for
     unsigned long destNetmask;    // The netmask for the route
     unsigned long gateway;        // IP address of the gateway of the route
     int hops;                     // Number or routers between this host and route
} Route_Entry;
```

Related Topics

API function **ADD_STATIC_ROUTE** - Add a route for an interface
API function **DEL_STATIC_ROUTE** - Delete a route from an interface

Top of list
Index page

---

send_params

```
struct send_params
{
   char             far   *bufferPtr;        // Pointer to send data
   int                     bufferLength;     // Number of bytes to send
   int                     flags;            // Blocking or no wait
   struct sockaddr  far   *toPtr;           // only needed for UDP
   int              far   *tolengthPtr;      // only needed for UDP
};
```

Comments

This structure is defined in tcpip.h. The elements are used as follows:

bufferPtr
Pointer to data to be sent.

bufferLength
Number of bytes to be sent from `bufferPtr`.

flags
Set to one of following constants defined in tcpip.h:
MSG_BLOCKING: Sleep until data is transferred into the socket transmit queue
MSG_DONTWAIT: Return immediately after loading bytes into available transmit queue space

toPtr UDP only
The referenced structure is cast to a `sockaddr_in`**structure**prior to usage. The datagram destination IP address and port number is specified here by caller.

tolengthPtr UDP only
The 16 bit value referenced by this pointer must be: `sizeof(struct sockaddr_in)`

Top of list
Index page

SetSocketOption

This data structure is defined in the TCPIPAPI.H library header file.

```
typedef struct tag_setsockopt
{
    int     protocol_level;     // Protocol level: IP level, TCP level or socket level
    int         optionName;     // Option's name constant
  const char far *optionValue;  // Pointer to the option value (type varies)
    int        optionLength;    // Length of option value data

} SetSocketOption;

 // Example usage of API_SETSOCKOPT and SetSocketOption type to set
 //   IP Time-to-Live to 69 seconds (or router hops):

    union  REGS  inregs, outregs;
    struct SREGS segregs;
    unsigned char time_to_live = 69;
    int socketdescriptor;  // You must initialize this (not shown here)
    SetSocketOption sockopt = {IP_PROTOIP_LEVEL,            // .protocol_level
                               IPO_TTL,                     // .optionName
                         (const char far *)&time_to_live,   // .optionValue
                               sizeof(unsigned char)};      // .optionLength

    inregs.h.ah = API_SETSOCKOPT;          // Interrupt 0xAC service index
    inregs.x.bx = socketdescriptor;
    segregs.h.es = FP_SEG(&sockopt);       // Fill in ES:DI with a pointer to sockopt
    inregs.x.di  = FP_OFF(&sockopt);
    int86x(TCPIPVECT, &inregs, &outregs, &segregs);  // Call API int 0xAC API_SETSOCKOPT
```

Comments

The type of data found at the location referenced by the `optionValue` member varies depending on the particular socket **option** that is being dealt with.

Related Topics

For list of options and sizes see **GetSocketOption**
API function **API_SETSOCKOPT** - Set socket options

Top of list
Index page

---

sockaddr

```
struct sockaddr
{
    u_char  sa_len;      // Total Length
    u_char  sa_family;   // Address Family AF_xxx
    char    sa_data[14]; // up to 14 bytes of protocol specific address
};
```

Comments

This generic "one size fits all" BSD structure is treated as a `sockaddr_in` **structure** within the TCP/IP API functions.

Top of list
Index page

---

sockaddr_in

```
struct sockaddr_in
{
    short           sin_family;  // AF_INET
    u_short         sin_port;    // 16 bit Port Number in network byte order
    struct  in_addr sin_addr;    // 32 bit netid/hostid in network byte order
    char            sin_zero[8]; // unused
};
```

Comments

The `sin_family` member should be set to `AF_INET` (=2).
The `sin_addr` member's `in_addr` **structure** is simply a long IP **address** in **big endian** byte **order**.

The **htons** function can be used to convert port numbers to network byte order.

Top of list
Index page

SocketInfo

```
typedef struct tag_socket_info{
        unsigned int  socIndex;
        unsigned char protocol;        // 6: TCP   17: UDP
        unsigned int  localPort;
        unsigned long IfIpAddress;
        unsigned int  remotePort;
        unsigned long remoteIP;
        unsigned char tcpState;
} SocketInfo;
```

Comments

Possible TCP socket states:

CLOSED 0
LISTEN 1
SYN_SENT 2
SYN_RECEIVED 3
ESTABLISHED 4
CLOSE_WAIT 5
FIN_WAIT_1 6
CLOSING 7
LAST_ACK 8
FIN_WAIT_2 9
TIME_WAIT 10
INVALID 20

Top of list
Index page

TcpMib

```
typedef struct tagTcpMib
{
   long               tcpRtoAlgorithm;  // retransmission timeout algorithm
   long               tcpRtoMin;        // minimum retransmission timeout (mS)
   long               tcpRtoMax;        // maximum retransmission timeout (mS)
   long               tcpMaxConn;       // maximum TCP connections possible
   unsigned long      tcpActiveOpens;   // number of SYN-SENT -> CLOSED transitions
   unsigned long      tcpPassiveOpens;  // number of SYN-RCVD -> LISTEN transitions
   unsigned long      tcpAttemptFails;  //(SYN-SENT,SYN-RCVD)->CLOSED or SYN-RCVD->LISTEN
   unsigned long      tcpEstabResets;   // (ESTABLISHED,CLOSE-WAIT) -> CLOSED
   unsigned long      tcpCurrEstab;     // number in ESTABLISHED or CLOSE-WAIT state
   unsigned long      tcpInSegs;        // number of segments received
   unsigned long      tcpOutSegs;       // number of segments sent
   unsigned long      tcpRetransSegs;   // number of retransmitted segments
   unsigned long      tcpInErrs;        // number of received errors
   unsigned long      tcpOutRsts;       // number of transmitted resets
} TcpMib;
```

Top of list
Index page

UdpMib

```
typedef struct tagUdpMib
{
        unsigned long      udpInDatagrams; // UDP datagrams delivered to users
        unsigned long      udpNoPorts;     // UDP datagrams to port with no listener
        unsigned long      udpInErrors;    // UDP datagrams unable to be delivered
        unsigned long      udpOutDatagrams;// UDP datagrams sent from this entity
} UdpMib;
```

Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables.

Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

Top of list
Index page

UserEthDhcp_Entry

```
typedef struct tag_UserDhcpEthEntry
{
    unsigned long   BootSIpAddress;      // BOOT Server (TFTP server), not supported
    unsigned long   Dns1ServerIpAddress; // Domain name server
    unsigned long   Dns2ServerIpAddress; // Second domain name server
    unsigned long   Yiaddr;              // Our (leased) IP address
    unsigned long   NetMask;             // Our subnet mask
    unsigned long   DefRouter;           // Default router
    unsigned long   DhcpServerId;        // DHCP selected server IP address
    unsigned long   internal1;           // Internal use only
    unsigned long   internal2;           // Internal use only
    unsigned long   DhcpLeaseTime;       // DHCP Address lease time in milliseconds
    unsigned long   internal3;           // internal use only
    unsigned long   internal4;           // internal use only
    unsigned char   DomainName[64];      // Domain name
    unsigned char   BootSname[64];       // TFTP server name, not used
    unsigned char   BootFileName[128];   // Boot file name (for TFTP download),
```

```
                                        //   not supported.
    unsigned short  BootFileSize;       // Boot file size in 512 Bytes blocks
                                        //    (for TFTP download).
    unsigned short  internal5;          // internal use only
} UserEthDhcp_Entry;
```

Comments

Members of this structure hold the DHCP configuration data if the IPC@CHIP is configured by a DHCP server.

This is read only information!

Related Topics

API function **DHCP_STAT** - Get DHCP status of the internal Ethernet interface of the IPC@CHIP
API function **DEV_WAIT_DHCP_COMPLETE** - Get DHCP status for a user Ethernet interface

Top of list
Index page

End of document

Client-Server

Programming client/server applications

Here is a short description for programming client/server applications with the TCP/IP API.
The most used methods for programming TCP/IP applications are client or server applications.

The term server applies to any process or program that offers a service that can be reached over the network. Servers accept request that arrive over the network, perform their service, and return the result to the requester. An example for the simplest service is the standard echo server application. The server echoes the received data over the network back to the requester. A process becomes a client when its sends a request to a server and waits for an answer. the client-server model is the standard model for interprocess communication. A TCP/IP stack provides two different methods for client-server connections:

1.UDP protocol:

This protocol realizes connectionless communication between a client and server, based on sending and receiving of single datagrams.

TCP/IP API calls for an UDP client:

- **Open a socket**
- **Send an outgoing datagram usually a prerecorded endpoint address.**
- **Receive the next incoming datagram and record its source endpoint address.**
- **Close a socket**

TCP/IP API calls for an UDP server:

- **Open a socket**
- **Bind a socket, assign an address to an unnamed socket**
- **Receive the next incoming datagram and record its source endpoint address.**
- **Send an outgoing datagram usually a prerecorded endpoint address.**

2.TCP protocol:

The TCP protocol is a connection- and byte stream-oriented protocol

TCP/IP API calls for a TCP client:

- **Open a socket**
- **Connect to a remote peer**
- **Send an outgoing stream of characters**

- **Receive an outgoing stream of characters**
- **Close a socket**

TCP/IP API calls for a TCP server:
- **Open a socket**
- **Bind a socket, assign an address to an unnamed socket**
- **Place the socket in a passive mode**
- **Accept the next incoming connection**
- **Receive an outgoing stream of characters**
- **Send an outgoing stream of characters**
- **Close a socket**

We provide several examples for programming client/server applications:
- UDPEchoClient, udpclie.c, built with int86x calls
- UDPEchoServer, udpserv.c, built with int86x calls
- TCPEchoClient, tcpclie.c, built with int86x calls
- TCPEchoServer, tcpserv.c, built with int86x calls
- TCPEchoClient, tcpclie.c, built with C-API-functions, using tcpip.c
- TCPEchoServer, tcpserv.c, built with C-API-functions, using tcpip.c

Note:
All program examples built with C-API-functions use the files TCPIP.C, TCPIP.H and TCPIPAPI.H.

End of document

RTOS API - SC12 @CHIP-RTOS V1.10

RTOS API

Here is the documentation for the RTOS API. This interface provides access to the RTOS of the IPC@CHIP. For a general introduction about multitasking with the @CHIP-RTOS see **Multitasking introduction**. Please note that we cannot explain in detail all principles of a multitasking system in this document.
A good book for understanding the architecture of Real-Time-Kernels is "MicroC/OS" from Jean.J.Labrosse.

Topics

RTOS API **Overview**
RTOS API **News**
RTOS API **Error Codes**
RTOS API **Developer**Notes
RTOS API **Examples**Available
RTOS API **Data Structures**
IPC@CHIP System **Tasks**

API Functions

The RTOS API uses interrupt 0xAD with a service number in the high order byte of the AX register (AH). The implemented RTOS services are listed below.

For some useful comments see also **Programming notes**.

- **Interrupt_0xAD_function_0x00:_RTX_SLEEP_TIME, Sleep for a specified time**
- **Interrupt_0xAD_function_0x01:_RTX_TASK_CREATE, Create and start a task**
- **Interrupt_0xAD_function_0x11:_RTX_TASK_CREATE_WITHOUT_RUN, Create a task**
- **Interrupt_0xAD_function_0x02:_RTX_TASK_KILL, Stop and kill specified task.**
- **Interrupt_0xAD_function_0x03:_RTX_TASK_DELETE, Remove a task from the system**
- **Interrupt_0xAD_function_0x04:_RTX_GET_TASKID, Get ID of the current task**
- **Interrupt_0xAD_function_0x05:_RTX_SLEEP_REQ, Sleep until wake request**
- **Interrupt_0xAD_function_0x06:_RTX_WAKEUP_TASK, Wake up a task**
- **Interrupt_0xAD_function_0x07:_RTX_END_EXEC, End execution of task**
- **Interrupt_0xAD_function_0x08:_RTX_CHANGE_PRIO, Change priority of a task**
- **Interrupt_0xAD_function_0x14:_RTX_CREATE_SEM, Create a semaphore**
- **Interrupt_0xAD_function_0x15:_RTX_DELETE_SEM, Delete a semaphore**
- **Interrupt_0xAD_function_0x16:_RTX_FREE_RES, Free a resource semaphore**
- **Interrupt_0xAD_function_0x17:_RTX_GET_SEM, Get counting semaphore (no wait)**
- **Interrupt_0xAD_function_0x18:_RTX_RELEASE_SEM, Release a resource semaphore**
- **Interrupt_0xAD_function_0x19:_RTX_RESERVE_RES, Get use of a resource semaphore**

- Interrupt_0xAD_function_0x1A:_RTX_SIGNAL_SEM, Signal a counting semaphore
- Interrupt_0xAD_function_0x1B:_RTX_WAIT_SEM, Wait on a counting semaphore
- Interrupt_0xAD_function_0x1C:_RTX_FIND_SEM, Find semaphore by name
- Interrupt_0xAD_function_0x28:_RTX_GET_TIMEDATE, Get system time and date
- Interrupt_0xAD_function_0x29:_RTX_SET_TIMEDATE, Set system time and date
- Interrupt_0xAD_function_0x2A:_RTX_GET_TICKS, Get tick count of system clock
- Interrupt_0xAD_function_0x09:_RTX_ACCESS_FILESYSTEM, Enable file access in task
- Interrupt_0xAD_function_0x0A:_RTX_GET_TASK_STATE, Get state of a task
- Interrupt_0xAD_function_0x0B:_RTX_GET_TASK_LIST, Get list of tasks
- Interrupt_0xAD_function_0x0C:_RTX_START_TASK_MONITOR, Enable task monitoring
- Interrupt_0xAD_function_0x0D:_RTX_STOP_TASK_MONITOR, Disable task monitoring
- Interrupt_0xAD_function_0x0E:_RTX_SUSPEND_TASK, Suspend a task
- Interrupt_0xAD_function_0x0F:_RTX_RESUME_TASK, Resume a task
- Interrupt_0xAD_function_0x10:_RTX_RESTART_TASK, Start task
- Interrupt_0xAD_function_0x12:_RTX_GET_TASK_STATE_EXT, Get task state
- Interrupt_0xAD_function_0x20:_RTX_DISABLE_TASK_SCHEDULING, Task Lock
- Interrupt_0xAD_function_0x21:_RTX_ENABLE_TASK_SCHEDULING, Release Task Lock
- Interrupt_0xAD_function_0x30:_RTX_INSTALL_TIMER, Install a timer procedure
- Interrupt_0xAD_function_0x31:_RTX_REMOVE_TIMER, Remove a timer procedure
- Interrupt_0xAD_function_0x32:_RTX_START_TIMER, Start periodic timer procedure
- Interrupt_0xAD_function_0x33:_RTX_STOP_TIMER, Stop execution of a timer procedure
- Interrupt_0xAD_function_0x40:_RTX_CREATE_EVENTGROUP, Create an event group
- Interrupt_0xAD_function_0x41:_RTX_DELETE_EVENTGROUP, Delete an event group
- Interrupt_0xAD_function_0x42:_RTX_SIGNAL_EVENTS, Signal event(s) in a group
- Interrupt_0xAD_function_0x43:_RTX_WAIT_EVENTS, Wait for events in a group
- Interrupt_0xAD_function_0x44:_RTX_GET_EVENTGROUP_STATE, Read the event states
- Interrupt_0xAD_function_0x45:_RTX_GET_EVENT_FLAGS, Get the saved event flags
- Interrupt_0xAD_function_0x46:_RTX_FIND_EVENTGROUP, Find an event group
- Interrupt_0xAD_function_0x50:_RTX_CREATE_MSG, Create a Message Exchange
- Interrupt_0xAD_function_0x51:_RTX_DELETE_MSG, Delete a Message Exchange
- Interrupt_0xAD_function_0x52:_RTX_SEND_MSG, Send message
- Interrupt_0xAD_function_0x53:_RTX_GET_MSG, Poll Message Exchange
- Interrupt_0xAD_function_0x54:_RTX_WAIT_MSG, Wait for a message
- Interrupt_0xAD_function_0x55:_RTX_FIND_MSG, Find a Message Exchange

At return from most of the API calls, the DX-Register is used for error checking as follows:

```
DX:   0  RTX_ENOERROR... success
DX:  -1  RTX_ERROR ... error, AX contains error code
DX:  -2  RTX_NOT_SUPPORTED ... service is not supported by the API
```

All needed constants and data **structures** for use with the RTOS API are defined in header file rtxapi.h. For a better understanding of the RTOS API, some **example** programs written in C are provided. The user should read these example and modify them for your own applications.

Interrupt 0xAD service 0x00: RTX_SLEEP_TIME, Sleep for a specified time

Parameters

AH

0 (=RTX_SLEEP_TIME)

BX

Sleep time in milliseconds in range 1 to 32767, inclusive. Note this 16 bit value is treated as signed. Any non-positive values are translated within this API to 1 millisecond.

Return Value

DX = 0 success AX: 0
DX != 0 failure AX: contains **error** code

Comments

The **RTX_WAKEUP_TASK** API (service 0x06) can wake up a sleeping task before its sleep timer has expired. In this case this API returns with error code -5 in AX.

The system maintains a "*Task Wakeup Pending* " flag for each task. This flag is set when **RTX_WAKEUP_TASK** is called for a task which is not currently awaiting wakeup (i.e. when bit 7 in **Task State** is zero). In the case where "*Task Wakeup Pending* " flag had been set, this RTX_SLEEP_TIME API clears this flag and returns immediately with error code -5 in AX.

Related Topics

RTOS **Task** Control Services
RTX_SLEEP_REQ Sleep until wake up call
RTX_WAKEUP_TASK Wake up a task

Developer Notes

A sleep call with BX parameter 1 millisecond sleeps for less than or equal to one millisecond. If a user needs a minimum sleep time of 1 millisecond they must call RTX_SLEEP_TIME with value 2 in BX. In general with BX sleep time specified to be N milliseconds, the resulting sleep time will range from N-1 milliseconds up to N milliseconds (inclusive).

Top of list
Index page

Interrupt 0xAD service 0x01: RTX_TASK_CREATE, Create and start a task

Parameters

AH

0x01 (= RTX_TASK_CREATE)

BX:SI

Pointer to 16 bit storage for the taskID, allocated by the caller

ES:DI

Pointer to a **TaskDefBlock** type data structure

Return Value

DX =0 success AX: 0, task is running, location [BX:SI] contains the 16 bit taskID
DX!=0 failure AX: contains **error** code

Comments

The caller must fill in portions of the **TaskDefBlock** structure prior to making this call.

The new task is immediately placed in the system's task ready queue. Execution begins if the task is higher priority than any other task currently ready (including task which called RTX_TASK_CREATE). The alternate API **RTX_TASK_CREATE_WITHOUT_RUN** can be used if it is not desired that the task be free to run immediately on creation.

Related Topics

IPC@CHIP System **Tasks**
RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x11: RTX_TASK_CREATE_WITHOUT_RUN, Create a task

Parameters

AH
0x11 (= RTX_TASK_CREATE_WITHOUT_RUN)

BX:SI
Pointer to 16 bit storage for the taskID, allocated by the caller

ES:DI
Pointer to a **TaskDefBlock** type data structure

Return Value

DX =0 success AX: 0, task is running, location [BX:SI] contains the 16 bit taskID
DX!=0 failure AX: contains **error** code

Comments

The caller must fill in portions of the **TaskDefBlock** structure prior to making this call.

Unlike the alternative **RTX_TASK_CREATE** API, this API call does not start the new task. The new task can be started with a **RTX_RESTART_TASK** call.

Related Topics

IPC@CHIP System **Tasks**
RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x02: RTX_TASK_KILL, Stop and kill specified task.

Terminate a specified task, but do not remove it from system.

Parameters

AH
0x02 (= RTX_TASK_KILL)

BX
taskID

Return Value

DX =0 success AX: 0, task is terminated
DX!=0 failure AX: contains **error** code

Comments

You should not kill a task which is waiting for a semaphore, an event in an event group, or a message from a message exchange. Failure to observe this restriction can lead to unpredictable results.

This function does not remove the task from the system. The task can be restarted by calling **RTX_RESTART_TASK** API function.

Related Topics

RTOS **Task** Control Services
RTX_TASK_DELETE Remove task from system
RTX_END_EXEC Task terminates itself

Top of list
Index page

Interrupt 0xAD service 0x03: RTX_TASK_DELETE, Remove a task from the system

Remove specified task from system.

Parameters

AH
0x03 (= RTX_TASK_DELETE)

BX
taskID

Return Value

DX =0 success AX: 0, task is removed
DX!=0 failure AX: contains **error** code

Comments

You should not delete a task which is waiting for a semaphore, an event in an event group, or a message from a message exchange. Failure to observe this restriction can lead to unpredictable results.

After making this call, the taskID is no longer valid. A task can delete itself.

Related Topics

RTOS **Task** Control Services
RTX_TASK_KILL Terminate task execution

Top of list
Index page

Interrupt 0xAD service 0x04: RTX_GET_TASKID, Get ID of the current task

Returns ID of the calling task.

Parameters

AH
0x04 (= RTX_GET_TASKID)

Return Value

DX=0 (success always) AX: contains the TaskID

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x05: RTX_SLEEP_REQ, Sleep until wake request

The calling task will be suspended until some other task issues a **RTX_WAKEUP_TASK** call to awake this calling task.

Parameters

AH
0x05 (= RTX_SLEEP_REQ)

Return Value

DX=0 (always success) AX: 0

Comments

The system maintains a "*Task Wakeup Pending* " flag for each task. This flag is set when **RTX_WAKEUP_TASK** is called for a task which is not currently awaiting wakeup (i.e. when bit 7 in **Task State** is zero). In the case where "*Task Wakeup Pending* " flag had been set, this RTX_SLEEP_REQ API clears this flag and returns immediately.

Related Topics

RTOS **Task** Control Services
RTX_WAKEUP_TASK Wake up a task
RTX_SLEEP_TIME Timed sleep

Top of list
Index page

Interrupt 0xAD service 0x06: RTX_WAKEUP_TASK, Wake up a task

To wake up a task known to be waiting because of a RTX_SLEEP_REQ or RTX_SLEEP_TIME call.

Parameters

AH
0x06 (= RTX_WAKEUP_TASK)

BX
taskID

Return Value

DX =0 success
DX!=0 failure AX: contains **error** code

Comments

An immediate task switch will occur if the task being wakened is of higher priority than the current task.

If this API is called for a task which is not currently awaiting wakeup, error code -6 is returned. In this case a wakeup is left pending such that when the specified task eventually makes a call to RTX_SLEEP_REQ or RTX_SLEEP_TIME API, it will react (once) to this pending wakeup and return immediately without a sleep period.

Related Topics

RTOS **Task** Control Services
RTX_SLEEP_REQ Sleep until wake up call
RTX_SLEEP_TIME Timed sleep

Top of list
Index page

Interrupt 0xAD service 0x07: RTX_END_EXEC, End execution of task

This call terminates the calling task.

Parameters

AH
0x07 (= RTX_END_EXEC)

Return Value

There is no return from this function

Comments

This call is equivalent to returning to the system from a task's main procedure.

The task can later be restarted with the **RTX_RESTART_TASK** API.

Related Topics

RTOS **Task** Control Services
RTX_TASK_KILL Kill specified task

Top of list
Index page

Interrupt 0xAD service 0x08: RTX_CHANGE_PRIO, Change priority of a task

Parameters

AH
0x08 (= RTX_CHANGE_PRIO)

BX
taskID

CX
priority, range 3 to 127 inclusive (3 is highest priority)

Return Value

DX =0 success AX: 0 DX!=0 failure AX: contains **error** code

Comments

An out of range priority value (CX) will be limited to range 3..127 inside this function.

Note:

Internally all tasks have a unique priority. When a task is **created**or its priority is changed, that task is given a lower internal task priority than any other task in the system with the same user task priority.

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x14: RTX_CREATE_SEM, Create a semaphore

Create a resource or counting semaphore.

Parameters

AH

0x14 (= RTX_CREATE_SEM)

BX:SI

Pointer to 16 bit storage allocated by caller where this API will output a semaphoreID

CX

Initial value:
Set to -1 for resource semaphore
Set in range [0 .. 32767] (inclusive) for counting semaphore

ES:DI

Pointer to 4 character unique name tag for the new semaphore, which need not be null terminated but must contain four bytes.

Return Value

DX =0 success AX: 0, Location referenced by [BX:SI] contains the unique semaphoreID
DX!=0 failure AX: contains **error** code

Comments

A resource semaphore is created by setting CX = -1. A non-negative value in CX creates a counting semaphore.

A resource semaphore is created in the free state, ready for use.

A counting semaphore is initially available the number of times specified in CX. The **RTX_SIGNAL_SEM** API increments this count and semaphore access via **RTX_GET_SEM** or **RTX_WAIT_SEM** decrements the count. A counting semaphore is not available when its count reaches zero.

Related Topics

RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x15: RTX_DELETE_SEM, Delete a semaphore

Removes specified semaphore from system.

Parameters

AH
0x15 (= RTX_DELETE_SEM)

BX
ID of the semaphore acquired by RTX_CREATE_SEM

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code

Comments

You must be certain that no other task, Interrupt Service Routine or **Timer** procedure is in any way using or about to use this semaphore. Failure to observe this restriction can lead to unpredictable faults.

Related Topics

RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x16: RTX_FREE_RES, Free a resource semaphore

The resource semaphore's use count is set to zero which unconditionally frees the resource.

Parameters

AH
0x16 (= RTX_FREE_RES)

BX
ID of the resource semaphore acquired from **RTX_CREATE_SEM**

Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains **error** code

Comments

Only the task owning the resource semaphore can make this call successfully. Error code -12 is returned if the calling task does not own the semaphore (or if the semaphore is a counting semaphore type which are not owned by tasks).

This API is useful to unwind in one stroke N calls made to **RTX_RESERVE_RES**. Alternatively, the **RTX_RELEASE_SEM** API could be called N times to release the resource semaphore.

After being freed, the resource will immediately be given to the task (if any) which is waiting at the head of this resource semaphore's wait queue. An immediate task switch occurs if this waiting task is higher **priority** than the calling task that just gave up ownership of this semaphore.

Related Topics

RTX_RESERVE_RES Reserve resource semaphore
RTX_RELEASE_SEM Release resource semaphore
RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x17: RTX_GET_SEM, Get counting semaphore (no wait)

Attempt acquisition of a counting semaphore without waiting.

Parameters

AH
0x17 (= RTX_GET_SEM)

BX
ID of the semaphore acquired from **RTX_CREATE_SEM**

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, semaphore is in use

Comments

This call returns an error code -51, "semaphore busy", if the semaphore is not available.

This function must not be called for resource type semaphores, as this will foul up the operation of the resource semaphore.

Related Topics

RTX_WAIT_SEM Wait for counting semaphore access
RTX_SIGNAL_SEM Signal counting semaphore

RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x18: RTX_RELEASE_SEM, Release a resource semaphore

Down count (unwind) a resource semaphore's "use count".

Parameters

AH
0x18 (= RTX_RELEASE_SEM)

BX
ID of the resource semaphore acquired from **RTX_CREATE_SEM**

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code

Comments

The resource's use count is decremented by one if the calling task presently owns this resource semaphore (otherwise error code -12, "not owner"). The resource is not freed until the use count reaches zero, at which point the caller no longer "owns" this semaphore.

Once freed, the resource will immediately be given to the task (if any) which is waiting at the head of this resource semaphore's wait queue. An immediate task switch occurs if this waiting task is higher **priority** than the calling task that just gave up ownership of this semaphore.

Note: Attempting to use this function on a counting semaphore will fail with error code -12.

Related Topics

RTX_FREE_RES Free resource semaphore
RTX_RESERVE_RES Reserve resource semaphore
RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x19: RTX_RESERVE_RES, Get use of a resource semaphore

Reserves a resource semaphore.

Parameters

AH

0x19 (= RTX_RESERVE_RES)

BX
ID of the semaphore acquired from **RTX_CREATE_SEM**

ES:DI
Pointer to signed long buffer containing the timeout in milliseconds
if timeout == 0, the caller waits forever for the resource
if timeout < 0, value is illegal resulting in error code -48

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, semaphore is in use

Comments

This call waits for a defined time to reserve a semaphore and returns with an error code -27 if the semaphore is still in use by another task after the timeout period expires.

The callers wait in FIFO order for the semaphore.

On success, the calling task then owns the resource semaphore. A task which owns the semaphore is free to call here repeated times, reserving the same semaphore more than once. Each such call increments a "use count" internal to the semaphore. This use count must be restored to zero before any other task can be granted ownership of this resource semaphore.

Related Topics

RTX_FREE_RES Free resource semaphore
RTX_RELEASE_SEM Release resource semaphore
RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x1A: RTX_SIGNAL_SEM, Signal a counting semaphore

Make semaphore access available to one additional task.

Parameters

AH
0x1A (= RTX_SIGNAL_SEM)

BX
ID of the semaphore acquired from **RTX_CREATE_SEM**

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code

Comments

Use this function to either surrender access to a counting semaphore, or to indicate that some resource guarded by this semaphore has become available.

Upon this signal, the semaphore will be given to the task (if any) which is waiting at the head of this semaphore's wait queue. This can result in an immediate task switch if this waiting task is higher **priority** than the calling task.

This function must not be called for resource semaphores, as this will cause the resource semaphore to malfunction.

Related Topics

RTX_WAIT_SEM Wait for counting semaphore access
RTX_GET_SEM Get counting semaphore without waiting
RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x1B: RTX_WAIT_SEM, Wait on a counting semaphore

Wait up to a specified time for access to a counting semaphore.

Parameters

AH
0x1B (= RTX_WAIT_SEM)

BX
ID of the semaphore acquired from **RTX_CREATE_SEM**

ES:DI
Pointer to signed long buffer containing the timeout in milliseconds
if timeout == 0, the caller waits forever for the resource
if timeout < 0, value is illegal resulting in error code -48

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code

Comments

This call waits for a defined time to acquire a counting semaphore and returns with an error code -27 if the semaphore is still not available after the timeout period expires.

The callers wait in FIFO order for the semaphore.

Related Topics

RTX_SIGNAL_SEM Signal counting semaphore

RTX_GET_SEM Get counting semaphore without waiting
RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x1C: RTX_FIND_SEM, Find semaphore by name

Get Semaphore ID using 4-char name.

Parameters

AH
0x1C (= RTX_FIND_SEM)

ES:DI
Pointer to 4 character name **tag** (no zero terminator needed)

Return Value

DX =0 success AX: contains the semaphore ID
DX!=0 failure AX: contains **error** code

Comments

The semaphore ID obtained here is the handle required for the semaphore API functions.
A semaphore created in some other program can be accessed in this manner.

If more than one semaphore was created with the same tag, you will get back the semaphore ID of one with this tag. But which one is not certain.

Related Topics

RTOS **Semaphore** Services

Top of list
Index page

Interrupt 0xAD service 0x28: RTX_GET_TIMEDATE, Get system time and date

Parameters

AH
0x28 (= RTX_GET_TIMEDATE)

BX:SI
Output parameter: Pointer to `TimeDate_Structure` **type** allocated by user.

Return Value

DX=0 success AX: 0, Location at [BX:SI] contains system date and time

Related Topics

Set Time/Date
RTOS **Time/Date** Services
TimeDate_Structure **type** definition

Top of list
Index page

Interrupt 0xAD service 0x29: RTX_SET_TIMEDATE, Set system time and date

Parameters

AH
0x29 (= RTX_SET_TIMEDATE)

BX:SI
Pointer to `TimeDate_Structure` **type** filled in by user.

Return Value

DX=0 success AX: 0

Comments

The *Day Of Week* field (`.dow`) in `TimeDate_Structure` need not be set by caller. This API function computes this field based on the other member data.

Caution: Values for time/date supplied by the caller are not checked for validity.

Related Topics

Get Time/Date
RTOS **Time/Date** Services
TimeDate_Structure **type** definition

Top of list
Index page

Interrupt 0xAD service 0x2A: RTX_GET_TICKS, Get tick count of system clock

Reads out the system millisecond clock tick count.

Parameters

AH
0x2A (= RTX_GET_TICKS)

BX:SI
Output Parameter: Pointer to an unsigned long where the tick count will be stored.

Return Value

Caller's unsigned long at [BX:SI] contains system tick count.

Comments

The system clock runs at 1000 Hz. So each tick represents 1 millisecond.

Related Topics

RTOS **Time/Date** Services

Top of list
Index page

Interrupt 0xAD service 0x09: RTX_ACCESS_FILESYSTEM, Enable file access in task

Enable file access for the calling task.

Parameters

AH
0x09 (= RTX_ACCESS_FILESYSTEM)

Return Value

DX =0 success AX: 0
DX!=0 failure (Mostly to many processes with file access)

Comments

This API call is only necessary for tasks created within applications with the **RTX_TASK_CREATE** or **RTX_TASK_CREATE_WITHOUT_RUN** API. DOS applications' main tasks already have access to the file system, so this call is not necessary for them.

If DX is -3 then file access is enabled, but reserving a data entry for **findfirst/findnext** failed.

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x0A: RTX_GET_TASK_STATE, Get state of a task

Get state information for a specified task.

Parameters

AH

0x0A (= RTX_GET_TASK_STATE)

ES:DI

Pointer to 4 character unique name tag of the task whose state information is desired. This need not be a null terminated, but must be four bytes.

DS:SI

Output parameter: Pointer to `Task_StateData` type **structure** allocated by the user to be filled by this API

Return Value

On Success:

DX = 0
AX = taskID
`Task_StateData`**structure**at [DS:SI] contains the current task state data

Failure:

DX = -1
IF AX is zero THEN
"Task **Monitoring**is not enabled".
ELSE
"Specified task not found".
ENDIF

Comments

Task monitoring mode must first be enabled in order for this API to work.

The alternative function, **RTX_GET_TASK_STATE_EXT**, can be used without starting the Task Monitor, but offers less information about the task.

Related Topics

`Task_StateData` **structure** definition
RTOS **Task** Control Services
Start Task Monitor API

Top of list
Index page

Interrupt 0xAD service 0x0B: RTX_GET_TASK_LIST, Get list of tasks

Get list of current tasks in the system.

Parameters

AH

0x0B (= RTX_GET_TASK_LIST)

ES:DI

Output Parameter: Pointer to array of `TaskList` **type** structures allocated by user.

CX

Length of the list, number of data structures in user's array at [ES:DI].

Return Value

DX=0
BX = number of tasks listed in [ES:DI] output array.

Comments

For a full report, the caller must allocate sufficient buffer space at [ES:DI] to allow all tasks to be reported including those **created** by the system. At most, CX tasks will be reported.

Related Topics

`TaskList` **structure** definition
RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x0C: RTX_START_TASK_MONITOR, Enable task monitoring

Installs an alternate 1000 Hz timer interrupt which monitors task execution.

Parameters

AH

0x0C (= RTX_START_TASK_MONITOR)

Return Value

DX=0, AX=0

Comments

This function installs a task timing function in the 0x13 timer interrupt which will poll at 1000 Hz to check which task is currently executing (or most recently if system is idle). **Data** collected by this timing function provides a coarse indication of which tasks are occupying the CPU.

Note that the Task Monitor places a considerable load on the system.

Related Topics

Get **Task** state
Stop Task Monitor API call
RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x0D: RTX_STOP_TASK_MONITOR, Disable task monitoring

Remove special 1000 Hz timer interrupt service used by Task Monitor function and restore original handler.

Parameters

AH
0x0D (= RTX_STOP_TASK_MONITOR)

Return Value

DX=0, AX=0

Comments

This function installs the system's normal timer 0x13 interrupt handler. This action is performed irrespective of whether or not the alternate 1000 Hz Task Monitor handler had been installed (**RTX_START_TASK_MONITOR** API).

This API performs a reset of the 1000 Hz Timer #2 count which extends the period of the current one millisecond real-time interrupt period by up to one millisecond.

Related Topics

RTOS **Task** Control Services
Start Task Monitor API call

Top of list
Index page

Interrupt 0xAD service 0x0E: RTX_SUSPEND_TASK, Suspend a task

Suspend the execution of a specified task until **RTX_RESUME_TASK** is called to resume the task.

Parameters

AH
0x0E (= RTX_SUSPEND_TASK)

BX
taskID (value from **RTX_TASK_CREATE** call)

Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains **error** code, invalid taskID

Comments

Note that the @CHIP-RTOS implementation maintains a separate Boolean representing "Suspended" for each task. Consequently, to suspend a task which is already waiting for some other **reason** (Trigger, Semaphore, Event Group, Message, or Sleep) means that the task will remain inactive after the other wait condition is released (e.g. after the task is granted a semaphore).

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x0F: RTX_RESUME_TASK, Resume a task

Re-enables the execution of a suspended task.

Parameters

AH
0x0F (= RTX_RESUME_TASK)

BX
taskID (value from **RTX_TASK_CREATE** call)

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, invalid taskID

Comments

This can result in an immediate task switch if the suspended task is higher **priority** than the calling task.

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x10: RTX_RESTART_TASK, Start task

Start execution of a specified task which was in the "Trigger Wait" **state**.

Parameters

AH
0x10 (= RTX_RESTART_TASK)

BX
taskID

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, invalid taskID

Comments

A task is in the "Trigger Wait" state either before it has been started for the first time after being created by the **RTX_TASK_CREATE_WITHOUT_RUN** API, or after termination.

A task termination results from either returning to the system from a task's entry procedure or due to API calls **RTX_TASK_KILL** or **RTX_END_EXEC**.

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x12: RTX_GET_TASK_STATE_EXT, Get task state

Get state of specified task without task monitoring mode active.

Parameters

AH
0x12 (= RTX_GET_TASK_STATE_EXT)

ES:DI
Pointer to 4 character unique name tag of the task whose state information is desired

Return Value

Success:

DX = 0
AX = taskID
BX = task state bit **field**

Failure (task not found)

DX != 0

Comments

The task name is not a null terminated string. It must contain four bytes (not necessarily ASCII).

The task state returned in BX is a bit **field** coded same as `taskState` member of the `Task_StateData` data structure.

Related Topics

API function **RTX_GET_TASK_STATE** - Get state of a task

Top of list
Index page

Interrupt 0xAD service 0x20: RTX_DISABLE_TASK_SCHEDULING, Task Lock

Task switching is inhibited until follow up call(s) to **RTX_ENABLE_TASK_SCHEDULER** is made. Interrupt service routines continue to execute, however **Timer** procedures will be delayed until task switching is re-enabled.

Parameters

AH
0x20 (= RTX_DISABLE_TASK_SCHEDULING)

Comments

After making this call, the task must remain compute bound until it follows up with a call to **RTX_ENABLE_TASK_SCHEDULER**. The task must not call any API which waits or ends the task.

Note that this is implemented as a spin lock, such that if for some reason the task calls here N times then N calls to **RTX_ENABLE_TASK_SCHEDULER** are required to unwind the spin lock and re-enable the task switching.

Caution: This call must be followed by a call to **RTX_ENABLE_TASK_SCHEDULER** as soon as possible to re-enable the task switching. Should the task lock period be excessive, the system **watchdog** must be triggered by the user until the task switching is re-enabled.

Related Topics

RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x21: RTX_ENABLE_TASK_SCHEDULING, Release Task Lock

Unwinds the task switching spin lock to re-enable task switching.

Parameters

AH
0x21 (= RTX_ENABLE_TASK_SCHEDULING)

Comments

This API reverses the affect of the **RTX_DISABLE_TASK_SCHEDULER** API.

When task switching is re-enabled, an immediate task switch may result if there is an active task with higher **priority** than the task making this API call.

Related Topics

API function **RTX_DISABLE_TASK_SCHEDULER**
RTOS **Task** Control Services

Top of list
Index page

Interrupt 0xAD service 0x30: RTX_INSTALL_TIMER, Install a timer procedure

Install a timer procedure that will be periodically executed by the kernel.

Parameters

AH
0x30 (= RTX_INSTALL_TIMER)

ES:DI
Pointer to a `TimerProc_Structure` **type**

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, no free timer available

Comments

See the `TimerProc_Structure` **type** description for instructions on how to call this function.

A timer ID is output to the 16 bit location referenced by `timerID` **member** of your `TimerProc_Structure`.

You must call the **RTX_START_TIMER** API function to get the kernel to start calling your new timer procedure.

Important:
Timer procedures are executed on the stack of the kernel task at a high priority, so they should be as short as possible. Avoid calling large functions like `printf()`.

Related Topics

`TimerProc_Structure` **definition**
RTOS **Timer** Procedures

Top of list
Index page

Interrupt 0xAD service 0x31: RTX_REMOVE_TIMER, Remove a timer procedure

Stop execution and remove a timer procedure.

Parameters

AH
0x31 (= RTX_REMOVE_TIMER)

BX
timerID produced by the **RTX_INSTALL_TIMER** call

Return Value

DX =0 success AX: 0
DX!=0, failure AX contains **error** code, invalid timerID.

Comments

It is safe to call this API from within the timer procedure being removed.

It is possible to reinstall a timer procedure after removing it from the system.

Related Topics

RTOS **Timer** Procedures

Top of list
Index page

Interrupt 0xAD service 0x32: RTX_START_TIMER, Start periodic timer procedure

Starts the periodic execution of a timer procedure.

Parameters

AH
0x32 (= RTX_START_TIMER)

BX
timerID produced by the **RTX_INSTALL_TIMER** call

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, invalid timerID.

Comments

The first execution of this timer procedure will occur within one millisecond of this call, after the subsequent @CHIP-RTOS 1000 Hz real-time interrupt.

The user is free to make this call on a timer which has already been started, the affect being that the phase of the periodic timer callback is shifted to within one millisecond of the call to this API.

Related Topics

RTOS **Timer** Procedures

Top of list
Index page

Interrupt 0xAD service 0x33: RTX_STOP_TIMER, Stop execution of a timer procedure

Stops execution of a timer procedure.

Parameters

AH
0x33 (= RTX_STOP_TIMER)

BX
timerID produced by the **RTX_INSTALL_TIMER** call

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, invalid timerID.

Comments

The timer procedure can later be **restarted**.

Related Topics

RTOS **Timer** Procedures

Top of list
Index page

Interrupt 0xAD service 0x40: RTX_CREATE_EVENTGROUP, Create an event group

Creates a new event group of 16 user definable event flags.

Parameters

AH

0x40 (= RTX_CREATE_EVENTGROUP)

BX

Initial value of the 16 event flags of the group.

ES:DI

Output Parameter: Pointer to 16 bit storage where Event Group ID is output by this API.

DS:SI

Pointer to unique four character tag, which need not be a zero terminated string but must consists of four bytes.

Return Value

DX = 0 success AX: 0 , location at [ES:DI] contains the unique group ID.
DX != 0 failure AX: contains **error** code, no free event group entry available

Comments

Each event group provides 16 Boolean event flags, encoded in a 16 bit word. Users are free to assign any meaning to these Booleans that suites their application.

Event groups can provide control and communicate between programs. One program creates the event group and the other programs must know the event group's unique 4 character tag. The other programs can then obtain the handle to this event group following its creation using the **RTX_FIND_EVENTGROUP** API.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x41: RTX_DELETE_EVENTGROUP, Delete an event group

Deletes specified event group.

Parameters

AH

0x41 (= RTX_DELETE_EVENTGROUP)

BX

Event group ID acquired by **RTX_CREATE_EVENTGROUP** call.

Return Value

DX = 0 success AX: 0
DX != 0 failure AX: contains **error** code, event group still in use or invalid group ID

Comments

You must not delete an event group which is in use by another task or timer procedure, as this can lead to unpredictable results.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x42: RTX_SIGNAL_EVENTS, Signal event(s) in a group

Set or clear up to 16 events under a mask in a group.

Parameters

AH
0x42 (= RTX_SIGNAL_EVENTS)

BX
Event group ID acquired by **RTX_CREATE_EVENTGROUP** call.

CX
16-Bit event group editing mask. The '1' bits here mark event flags to be set or cleared based on the corresponding bit value supplied in DX. Other flags in the event group are unaffected.

DX
New event values for the 16 event flags. Only the bits marked '1' in the CX mask are relevant here.

Return Value

DX = 0 success AX: 0
DX != 0 failure AX: contains **error** code

Comments

The Event Manager wakes up any tasks that are waiting on these events and satisfied with the resulting collection of event states (in case of AND waiting condition). This can lead to an immediate task switch if any of these released tasks are higher **priority** than the signaling task.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x43: RTX_WAIT_EVENTS, Wait for events in a group

The calling task waits for up to a specified number of milliseconds for event(s) in an event group to assume specified value(s).

Parameters

AH
0x43 (= RTX_WAIT_EVENTS)

BX
Event group ID acquired by **RTX_CREATE_EVENTGROUP** call.

ES:DI
Pointer to user `RTX_Wait_Event` **type** structure filled in by caller.

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code

Comments

The user must fill in the `RTX_Wait_Event` **structure** before making this call.

The caller can select whether to await all specified events or a single event.

Related Topics

`RTX_Wait_Event` **type** definition
RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x44: RTX_GET_EVENTGROUP_STATE, Read the event states

Returns the current state of the 16 event flags (bits) of a specified event group.

Parameters

AH
0x44 (= RTX_GET_EVENTGROUP_STATE)

BX
Event group ID acquired by **RTX_CREATE_EVENTGROUP** call.

ES:DI
Output Parameter: Pointer to 16 bit location to receive the current state of the event flags

Return Value

DX = 0 success AX: 0, Location at [ES:DI] contains the event states of the specified group
DX != 0 failure AX: contains **error** code, invalid group ID

Comments

Note that this API accesses the *current* event group states.

If instead the states recorded at return from the most recent **RTX_WAIT_EVENTS** call are desired, the alternate **RTX_GET_EVENT_FLAGS** API can be used.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x45: RTX_GET_EVENT_FLAGS, Get the saved event flags

Return the state of the 16 event flags as they were at the time the calling task most recently completed a **RTX_WAIT_EVENTS** call.

Parameters

AH
0x45 (= RTX_GET_EVENT_FLAGS)

Return Value

DX =0 success AX: contains the saved event states

Comments

The returned event flags apply to this task's most recent event wait wake up or timeout.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x46: RTX_FIND_EVENTGROUP, Find an event group

Find an event group by specified name tag and return the unique event group ID.

Parameters

AH

0x46 (= RTX_FIND_EVENTGROUP)

ES:DI
Pointer to 4 character name tag. This string need not be zero terminated but most be four bytes length.

Return Value

DX = 0 success AX: contains the event group ID
DX != 0 failure AX: contains **error** code, not found

Comments

This function allows event groups to be accessed by another program which knows the agreed upon name for the group, thereby providing a communication and control mechanism between programs.

Related Topics

RTOS **Event** Manager

Top of list
Index page

Interrupt 0xAD service 0x50: RTX_CREATE_MSG, Create a Message Exchange

Creates a new Message Exchange.

Parameters

AH
0x50 (= RTX_CREATE_MSG)

ES:DI
Pointer to user `RTX_Msg` **type** structure

Return Value

DX = 0 success AX: 0, RTX_Msg structure contains the new msgID
DX != 0 failure AX: contains **error** code

Comments

The user must fill in portions of the `RTX_Msg` **structure** prior to calling here.

The Message Exchange Manager returns a 16-Bit unique ID to the caller.

You can provide a unique 4 character tag to identify the Message Exchange, thereby allowing other programs access to it by **name**.

The maximum number of Message Exchanges supported by the system is ten.

Related Topics

RTX_Msg **type** definition
RTOS **Message** Exchange Manager

Top of list
Index page

Interrupt 0xAD service 0x51: RTX_DELETE_MSG, Delete a Message Exchange

Deletes specified Message Exchange.

Parameters

AH
0x51 (= RTX_DELETE_MSG)

BX
Message Exchange ID acquired by **RTX_CREATE_MSG** call.

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, Message Exchange still in use or invalid ID

Comments

You must not delete a Message Exchange which is in use by another task or timer procedure, as this may result in unpredictable faults.

Related Topics

RTOS **Message** Exchange Manager

Top of list
Index page

Interrupt 0xAD service 0x52: RTX_SEND_MSG, Send message

Send provided message to a specified Message Exchange.

Parameters

AH
0x52 (= RTX_SEND_MSG)

BX
Message Exchange ID acquired by **RTX_CREATE_MSG** call.

CX
Message priority (mailbox) 0 - 3 where 0 is highest priority

ES:DI
Pointer to a 12 byte message to be sent

Return Value

DX = 0 success AX: 0
DX != 0 failure AX: contains **error** code

Comments

If one or more tasks are waiting at the exchange for a message, the message will be immediately given to the task waiting at the head of the exchange's wait queue. This will result in an immediate task switch if the task receiving the message is higher **priority** than the task that is sending the message.

The format of the 12 byte message being sent is defined by the application program. These 12 bytes can contain a pointer to further data, if required. This API copies these 12 bytes into an internal message envelope, so the message at [ES:DI] need not persist beyond the call to this API. (However, any data referenced by pointers contained in the message would, of course, need to be maintained until the message is received by some task.)

The message mailbox priority in CX is applicable for cases where messages are accumulating on the exchange for which there are no immediate waiting message consumer tasks. In this case, a message from the highest priority non-empty mailbox FIFO queue will be given to the next caller of either **RTX_GET_MSG** or **RTX_WAIT_MSG**.

Related Topics

RTOS **Message** Exchange Manager

Top of list
Index page

Interrupt 0xAD service 0x53: RTX_GET_MSG, Poll Message Exchange

Get an available message from a specified Message Exchange without waiting.

Parameters

AH
0x53 (= RTX_GET_MSG)

BX
Message Exchange ID acquired by **RTX_CREATE_MSG** call.

ES:DI
Output Parameter: Pointer to a 12 byte user buffer for storing the message (if any).

Return Value

DX = 0 success AX: 0, Location at [ES:DI] holds the message
DX != 0 failure AX: contains **error** code, invalid ID or -28: no message available

Comments

This function always returns immediately. If no message is currently available, DX is non-zero and AX = -28.

On success, a message from the highest priority non-empty FIFO mailbox is copied into the 12 byte store at [ES:DI] and then this delivered message is removed from the mailbox.

Related Topics

RTOS **Message** Exchange Manager

Top of list
Index page

Interrupt 0xAD service 0x54: RTX_WAIT_MSG, Wait for a message

Wait up to a specified number of milliseconds for a message from a Message Exchange.

Parameters

AH
0x54

ES:DI
Pointer to user `RTX_Wait_Msg` **type** structure

Return Value

DX = 0 success AX: 0
DX != 0 failure AX: contains **error** code

Comments

The user must fill in the `RTX_Wait_Msg` **structure** prior to calling here.

On success, the highest priority message available on the exchange is copied into your message container at location specified by the `msg` member of the `RTX_Wait_Msg` structure. Then the message is removed from the Message Exchange.

Each caller to this API can specify their priority for access to messages. To wait in FIFO order, all callers have to wait with the same priority.

Related Topics

`RTX_Wait_Msg` **type** definition
RTOS **Message** Exchange Manager

Top of list
Index page

Interrupt 0xAD service 0x55: RTX_FIND_MSG, Find a Message Exchange

Find a Message Exchange by specified name. **tag** and return the unique exchange ID.

Parameters

AH
0x55 (= RTX_FIND_MSG)

ES:DI
Pointer to 4 character name **tag** (no zero terminator needed)

Return Value

DX = 0 success AX: contains the Message Exchange ID
DX != 0 failure AX: contains **error** code, not found

Comments

This API can be used to access a Message Exchange created with an agreed upon name by another program, thereby providing a communication path between programs.

If more than one message exchange was created with the same tag, you will get back the message exchange ID of one with this tag, but which one is not certain.

Related Topics

RTOS **Message** Exchange Manager

Top of list
Index page

End of document

RTOS API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

RTOS API News

The following extensions to the RTOS **API** are available in the indicated BIOS revisions.

No changes since last version.

End of document

Multitasking with @Chip-RTOS

Some common explanations about multitasking with the @CHIP-RTOS

Multitasking Introduction

The @Chip-RTOS provides for a multitasking operation. A task provides a thread of execution. Each task has its own context, including an instruction pointer and program stack. Each task in the @Chip-RTOS has a unique priority of execution, providing a preemptive form of multitasking. By executing multiple tasks at the same time, various activities can be performed concurrently.

Abbreviations Used

Acronym	Stands For
API	Application Programmers Interface
ISR	Interrupt Service Routine
RTI	Real-Time Interrupt
RTOS	Real-Time Operating System
TCP/IP	Transport Control Protocol / Internet Protocol

Table 1) Abbreviations

Reasons for using Multitasking

Here are some situations where multitasking can be helpful.

Different Priority Work

Probably the most compelling reason for using multitasking is when required activities have different priorities. For example, an application with a user interface may need to be responsive to keyboard entry from a user console while at the same time the program is conducting a time consuming calculation or data base search. A low priority task could perform the calculation/data base search while a high priority task periodically polls (or sleeps waiting) for keyboard input. Typically there would be some communication between these two tasks, such as having the keyboard task cancel the

background data base search in the event that the user presses the escape key.

In more involved applications, there might be a complete spectrum of concurrent activity at different priorities. Just for an example, the set of tasks within a real-time application might be as follows. Listed in priority order:

- 50 Hz Data Acquisition Loop (top priority)
- 10 Hz Data Pre-processing Loop
- 1 Hz Data Processing Loop
- 0.1 Hz Kalman Filter
- Background Loop for Integrity Self-test

Event Triggered Actions

Sometimes an activity is required only upon an event such as some data becoming available. For these situations a task could be made to block, awaiting the event or data arrival. For example, a task could make a blocking call on a TCP/IP port with the recv() library function. The task sleeps until some data arrives at the respective socket, or until a prescribed time-out period expires.

Note that polling can usually be an alternative software design approach, as opposed to applying a dedicated task to each event or data source.

Reasons not to use Multitasking

While multitasking can be an ideal solution for some applications, it also has its disadvantages. Some of these disadvantages are noted here. The point is that you should not complicate an application with multiple tasks unless to do so truly makes your application simpler.

Resources Expended per Task

Each task requires its own stack space. This stack must be made large enough to support the system's interrupt handlers, some of which operate without a switch to a system stack. A minimum stack space of 1024 bytes is recommended.

Critical Sections

Objects (memory or devices) shared between tasks can lead to critical sections. A critical section is a section of code in a task which is sensitive to order of execution relative to code in some other task, such that there is a possible (prior to adding the necessary protection) firing order of the two instruction streams which leads to incorrect results.

When you design your application with only a single task, it is safe to say you have no critical sections. When more than one task is used, **you must beware**. Due to the importance of understanding and recognizing critical sections in multitasking systems, this topic is covered in more detail below on chapter Critical Sections in Programs.

Primary Task Attributes

Two important attributes of each task are task priority and state.

Task Priority

Each task executing under the @Chip-RTOS has a unique internal task priority. This internal task priority is represented with a 16 bit value, the most significant byte of which is the user task priority which is set at task creation time or using the RTX_Change_Task_Prio() API (and is visible through the RTX_Get_Task_State() API). The hidden least significant byte of the task priority is used internally by the @Chip-RTOS to assign each task at a given user priority a unique priority by appending a sequence number to the upper byte.

A task is assigned the lowest internal priority of all tasks with the same user priority whenever that task is appended to the list of tasks at that given user priority. This occurs when:

a) the task is created

b) the task's priority is changed

c) the task's time-slice period times out (see chapter Time-Slicing).

Application program tasks can range in user priority from 3 to 127 (inclusive), where 3 is higher priority. Generally, user task priorities between 20 and 30 are recommended. This recommendation is based on the priority assignments of the built-in system tasks. Too high a priority for an application task may block urgent system tasks: e.g. the Ethernet receiver task.

Task State

In Table 2 below, the possible states and sub-states for @Chip-RTOS tasks are summarized. There are three primary states: Active, Blocked and Suspended.

State	Sub-State	Notes and State Transitions
Active	Executing	Highest priority non-waiting task
	Pending	In queue ordered by task priority
Blocked	Trigger Wait[1]	RTX_Restart_Task() → **Active**
	Semaphore Wait[2]	RTX_Signal_Sem(), RTX_Release_Sem() → **Active**
	Event Group Wait[2]	RTX_Signal_Events() → **Active**
	Message Exchange Wait[2]	RTX_Send_Msg() → **Active**
	Asleep[2]	RTX_Wakeup() → **Active**

Suspended	Free to run	RTX_Resume_Task() → **Active**
	Trigger Wait[1]	RTX_Resume_Task() → **Blocked** RTX_Restart_Task() → , Free to run[3]
	Semaphore Wait[2]	RTX_Resume_Task() → **Blocked** Granted semaphore → , Free to run[3]
	Event Group Wait[2]	RTX_Resume_Task() → **Blocked** RTX_Signal_Events() → , Free to run[3]
	Message Exchange Wait[2]	RTX_Resume_Task() → **Blocked** RTX_Send_Msg() → , Free to run[3]
	Asleep2	RTX_Resume_Task() → **Blocked** RTX_Wakeup() → , Free to run[3]

1) - Trigger Wait sub-state is entered after RTX_Create_Task_Without_Start() or after a task has terminated.
2) - A specified time-out period in milliseconds can be applied to these states.
3) - Only the sub-state has changed here.

Table 2) @Chip-RTOS Task States

The set of active tasks we speak of as executing concurrently. However, only a single task (at most[1]) is executing at any given time since the IPC@Chip contains only a single CPU. The task selected for execution by the @Chip-RTOS will always be the highest priority of the tasks that are in the active state.

The C-library routines which force a task to exit the Blocked and Suspended states when called by some other task or Interrupt Service Routine (ISR) are stated in the table. The two inactive states, Blocked and Suspended, differ in their exit state transitions. The RTX_Suspend_Task() API transitions a task into the Suspended state.

1 Hardware interrupt service routines can momentarily suspend the executing task.

Top of this document
IPC@CHIP Documentation Index

DOS Program Tasks

Each DOS program is launched as a task under @Chip-RTOS. These tasks are created with initial priority 25 and time-slicing disabled. Within these DOS programs, users can create additional tasks with the RTX_Task_Create() or RTX_Task_Create_Without_Run() API

System Timing

The @Chip-RTOS uses a 1000 Hz Real-Time Interrupt (RTI) for its time base. Therefore one millisecond is the lower resolution available for task timing.

Users can install Timer Callback procedures with the RTX_Install_Timer() API. Your callback procedure is invoked within the top priority kernel task at a specified interval

Time-Slicing

For the tasks created within DOS programs by the user, a time-slicing feature is available. This feature is enabled for a specific task by specifying a non-zero number of milliseconds in the time_slice member of the TaskDefBlock structure passed to the RTX_Task_Create() or RTX_Task_Create_Without_Run() functions.

A time-sliced task will be permitted to execute for time_slice milliseconds (RTI ticks) after which time it will be cycled to the end of the list of tasks at this task's user priority. (The task is not charged for ticks during which it was blocked, suspended or active-pending preempted by some higher priority task.) In the special case where it is the only active task at that user priority, it would then on time-out immediately be given another time_slice milliseconds execution time budget and allowed to continue execution. Otherwise one of the other active tasks pending execution at this same user priority will begin execution and the previously executing task whose time-slice expired will be cycled to the end of the priority queue in a round-robin fashion. Note that the next task to execute may or may not be configured for time_slice operation.

The time-slice operation would apply primarily to fully independent tasks which do not pass any data between each other. Time-slicing can introduce chaos into a program which could execute more orderly using explicit yields (e.g. RTX_Sleep_Time API). The extra task switching due to time-slice operation can cause critical sections to appear where they otherwise would not if the task was permitted to execute up to where the program yields voluntarily.

There may be times where time-slicing is the graceful design solution, but reliance on this technique raises the suspicion that the software design was not thought out thoroughly. Also keep in mind that any task with lower user priority than the time-sliced tasks will never be executed so long as any of the time-sliced tasks are active. During execution of a time-sliced task, there is a very slight additional load placed on the system's 1000 Hz RTI.

Periodic Tasks

Periodic tasks can be created in either of two ways, depending on how accurate the execution period is required to be for the respective application.

Roughly Periodic

The simplest form for a periodic task uses a RTX_Sleep_Time call within a loop as shown below in Figure 1. The period of this loop will not be exact, but would be close enough for many applications.

```
#define SLEEP_10HZ  (90)  // Assuming 10 ms CPU load per 100 ms

void huge Task_Roughly_10Hz(void)
{
    while (1)
    {
         Activity_10Hz() ;   // Get here about each 100 ms
         RTX_Sleep_Time(SLEEP_10HZ) ;
    }
}
```

Figure 1) Sleep Based Periodic Loop

The SLEEP_10HZ constant used in this example is adjusted based on the expected system loading, including CPU dwell within the Activity_10Hz procedure. This would require some timing measurements to be made during the program's development.

Precisely Periodic

A precisely periodic loop can be controlled with an RTOS timer. This will result in a periodic loop which on average tracks the CPU quartz clock. A RTOS timer periodically wakes up the periodic task loop as illustrated below in Figure 2.

```
static int TaskID_10Hz ;

void huge Task_10Hz(void)
{
    while (1)           // 10 Hz loop
    {
         RTX_Sleep_Request () ;
         Activity_10Hz() ;   // Get here each 100 ms.
    }
}

static void huge Timer_Callback(void) // Clean 10 Hz
{
    RTX_Wakeup (TaskID_10Hz) ;
}

static TimerID ;
static TimerProc_Structure Timer_Spec = {
    &TimerID,
    Timer_Callback,
    0,
    { '1', '0', 'H', 'z'},
    100              // .interval = 100 milliseconds
} ;

void Initialize(void)
{
    extern TaskDefBlock Task_10Hz_Def ;
    RTX_Create_Task (&TaskID_10Hz, &Task_10Hz_Def) ;
    RTX_Install_Timer (&Timer_Spec) ;
    RTX_Start_Timer (TimerID) ;
}
```

Figure 2) Timer Based Periodic Loop

An alternative method here would be to use RTX_Suspend_Task in Task_10Hz and RTX_Resume_Task in Timer_Callback. However, the RTX_Wakeup has the advantage of a wakeup pending flag used in the implementation which covers for the case where, due to CPU loading, the Task_10Hz may not yet have reached the RTX_Sleep_Request call before the RTX_Wakeup is executed in the Timer_Callback. In this case when the RTX_Sleep_Request is later called in Task_10Hz, the API will return immediately after clearing the task's internal wakeup pending flag, which had been set when Timer_Callback called RTX_Wakeup before Task_10Hz reached its sleep.

Critical Sections in Programs

When multitasking is used, the programmer must beware of critical sections which may occur between threads.

Critical sections can be protected with proper design. The important step at program design time is to identify these code sections, which are not always obvious. Most programmers are well aware of what critical sections are. However, due to their importance when multitasking, a simple example is provided here for emphasis.

Example Critical Section

Data sharing between tasks leads to critical sections when the shared data object can not be read or written atomically, within a single non-interruptible machine instruction.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
        //  Borland compiler's machine code:
        //     ADD    My_Ticker,1H   ; Increment LS 16 bits
        // → Sensitive to task switch here!
        //     ADC    My_Ticker+2,0H ; Carry into MS 16 bits
    }
}


void huge Task_B(void)
{
    if (My_Ticker == 0)
    {
        Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

Figure 3) Critical Section Example

After a brief review of the C code in the above example, the C programmer might suspect a hardware problem if the Surprised_to_be_Here() function was to ever execute. However, with a closer examination of the resulting machine assembly code and multitasking consideration, we will see that execution of the Surprised_to_be_Here() function is possible[2].

2 And "if something bad can happen, it will happen".

All tasks in the @Chip-RTOS system have a unique task priority. So in the above example either Task_A

can potentially interrupt execution of Task_B, or visa-versa, depending on the assigned priorities. Consider the case where priority of Task_A is lower (higher numerically) than priority of Task_B, such that Task_B can preempt Task_A. This case can lead to execution of the Surprised_to_be_Here() function under the following circumstances.

Let us say that Task_A has already executed 0xFFFE times and on its 0xFFFF'th execution it is preempted by Task_B at the indicated "Sensitive" point immediately after executing the ADD opcode which increments the lower half of the 32 bit up counter. At this exact point, the My_Ticker value will read zero due to the carry from the lower 16 bits not yet being applied to the upper half word. And thus Task_B lands in the Surprised_to_be_Here() function when it encounters the half updated My_Ticker reading.

Protecting Critical Sections

Three methods for protecting critical sections are presented here.

1) Semaphore

2) Interrupt Masking

3) RTOS Task Switch Lock

Each method has its advantages and limitations, which are summarized at the end of this discussion. The choice of which method to use will depend on the design situation.

Semaphore Protection

A common way to protect critical sections is with the use of semaphores. The @Chip-RTOS provides resource semaphores which provide a mutually exclusive access to a resource or data object.

The example defective code from Figure 3 above can be corrected with the use of a resource semaphore as shown below.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    RTX_Reserve_Sem(semID, 0) ;
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    RTX_Release_Sem(semID) ;
}


void huge Task_B(void)
{
    unsigned long ticker ;
    RTX_Reserve_Sem(semID, 0) ;
    ticker = My_Ticker ;
    RTX_Release_Sem(semID) ;
    if (ticker == 0)
```

```
    {
          Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

Figure 4) Protected Critical Section with Semaphore

Now the Surprised_to_be_Here() function will never be executed.

A potential disadvantage to using semaphores is a possible task priority inversion, where a high priority task is blocked by a lower priority task as it awaits the semaphore. To illustrate this point, consider an example where task priorities are designed as follows:

Task_A - Priority 60 (low priority)

Task_B - Priority 4 (very high priority)

If Task_A is suspended while it has possession of the semaphore, Task_B will have to wait if it then tries to access the same semaphore at that moment. This wait is effectively at the very low priority 60, which would mean that Task_B (priority 4) must sit waiting behind time consuming system activities such as FTP transfer (priority 41). In applications where this potential priority inversion is not acceptable, either the interrupt masking or task lock methods of protecting critical sections discussed below can be considered as an alternative to using semaphores.

Interrupt Masking

Interrupt masking can in some cases be a safe alternative to using semaphores to protect critical sections. This fast method places a minimum load on the system, so is most suitable where performance is a concern. The interrupt masking method is used in the example below.

```
define MASK_INTERRUPTS   asm{CLI}
#define ENABLE_INTERRUPTS asm{STI}

static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    MASK_INTERRUPTS ;   // Needed if Task_A is lower priority
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    ENABLE_INTERRUPTS ;
}


void huge Task_B(void)
{
    unsigned long ticker ;
    MASK_INTERRUPTS ;  // Needed if Task_B is lower priority
    ticker = My_Ticker ;
    ENABLE_INTERRUPTS ;
    if (ticker == 0)
    {
          Surprised_to_be_Here() ;  // How did this happen?
```

```
    }
}
```

Figure 5) Protected Critical Section with Interrupt Masking

This method of protection is safe to use when the section being protected executes in very few machine cycles, as is the case in this example. The concern is the hardware interrupt latency created by this interrupt mask period. Masking interrupts for as long as 50 microseconds should be tolerable on most systems. Caution must be used to assure that interrupts are always quickly re-enabled when ever they are disabled!

Note that when the nature of the two tasks competing for access to the resource (Task_A and Task_B in this example) dictates that one is higher priority than the other, only the lower priority task requires the interrupt masking. It is not possible that the lower priority task could preempt the higher priority task (unless the program design was to change task priorities dynamically somewhere).

RTOS Task Switch Lock

A further alternative to using semaphores to protect critical sections is to prevent task switching within the critical section. This method is shown in the example below.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
      // Disable needed if Task_A is lower priority
    RTX_Disable_Task_Scheduling() ;  // Task switch lock
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    RTX_Enable_Task_Scheduling() ;   // Resume task switching
}


void huge Task_B(void)
{
    unsigned long ticker ;
      // Disable needed if Task_B is lower priority
    RTX_Disable_Task_Scheduling() ;  // Task switch lock
    ticker = My_Ticker ;
    RTX_Enable_Task_Scheduling() ;   // Resume task switching
    if (ticker == 0)
    {
          Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

Figure 6) Protected Critical Section with Task Lock

Hardware interrupts continue to be serviced during the task lock, so you can include more work now within the critical section than was possible with the interrupt masking method. However, the task lock

period should still be keep to some reasonable small amount of time. Note that task locks also inhibit all system timer activity.

Critical Section Protection Methods Summary

The design trade-offs for the three methods presented above for protecting critical sections are summarized in Table 3.

Method	Advantage	Limitations
Semaphore	A long duration of critical section does not adversely affect portions of system not accessing the semaphore.	Can result in a priority inversion.
Interrupt Mask	Most efficient of all three methods. (Executes quickly!) No priority inversion	Impact on system operation becomes a concern if interrupt mask time can exceed around 50 us.
Task Lock	No priority inversion.	Impact on system operation becomes questionable if task lock duration exceeds around 400 us.

Table 3) Critical Section Protection Methods

Top of this document
IPC@CHIP Documentation Index

Control and Communication between Tasks

The @Chip-RTOS provides the following mechanisms for tasks to control one another and to communicate. These interactions can either be between tasks within the same DOS application, or across applications.

- Semaphores
- Event Groups
- Message Exchanges

The usage of these @Chip-RTOS resources is covered in depth within the on-line HTML help.

Top of this document
IPC@CHIP Documentation Index

RTOS Overview - SC12 @CHIP-RTOS V1.10

RTOS Overview

The RTOS **API** services split into the following groups:

- **Task_Control Services**
- **Semaphore_Services**
- **Time_/ Date Services**
- **Timer_Procedures**
- **Event_Manager**
- **Message_Exchange Manager**

Task Control Services

RTX_TASK_CREATE	Create and start a task
RTX_TASK_CREATE_WITHOUT_RUN	Create a task
RTX_END_EXEC	End execution of a task by itself
RTX_TASK_KILL	Kill a task
RTX_RESTART_TASK	Trigger start of task execution
RTX_TASK_DELETE	Remove a task from the system
RTX_GET_TASKID	Get ID of the current task
RTX_CHANGE_PRIO	Change the priority of a task
RTX_SLEEP_TIME	Sleep for a specified time
RTX_SLEEP_REQ	Sleep until a wake request
RTX_WAKEUP_TASK	Wake a task which is sleeping
RTX_SUSPEND_TASK	Suspend a task
RTX_RESUME_TASK	Resume a task
RTX_ACCESS_FILESYSTEM	Enable file access for the calling task
RTX_GET_TASK_STATE	Get state of a task (task monitoring)
RTX_GET_TASK_STATE_EXT	Get state of a task (without task monitoring)
RTX_GET_TASK_LIST	Get list of tasks in system
RTX_START_TASK_MONITOR	Start task monitoring
RTX_STOP_TASK_MONITOR	Stop task monitoring

Top of list
Index page

Semaphore Services

Semaphores are used to guarantee a task mutually exclusive access to a critical resource. Semaphores synchronize asynchronous occurring activities. They are an essential part of a multitasking system. A good description of multitasking systems and semaphores is available in the book "Operating systems" from Andrew Tanenbaum.

The @CHIP-RTOS API provides two types of semaphores:

- A **counting semaphore** is a semaphore with an associated counter, which can be incremented (signal) and decremented (wait). The resource controlled by the semaphore is free (available) when the counter is greater than 0.
- A **resource semaphore** is a counting semaphore with a maximum of count of one. It can be used to provide mutually exclusive access to a single resource. A resource semaphore is also called a binary semaphore. It differs from a counting semaphore in one significant feature: The resource ownership is tied to a specific task. No other task except the task owning the resource is allowed to signal the associated semaphore to release the resource.

The counting and resource semaphores provide automatic timeout. Tasks can specify the maximum time for waiting on a semaphore. The tasks wait in FIFO order for a resource. A semaphore is created with RTX_CREATE_SEMservice call. The @CHIP-RTOS needs a unique four byte semaphore name and on success returns a new semaphore ID (or handle) to the caller. This handle is needed for the other semaphore services.

Using a counting semaphore:

A counting semaphore is created by specifying an initial count greater or equal to zero in the call to **RTX_CREATE_SEM**. If a semaphore is initialized with a value n, it can be used to control access to n resources, e.g. a counting semaphore with the initial value three assures that no more than three tasks can own a resource at any one time. Access to a resource controlled by a counting semaphore is acquired with a call to **RTX_WAIT_SEM**or **RTX_GET_SEM**. If the resource is available the @CHIP-RTOS gives it to the task immediately. When the task is finished using the resource, it signals its release by calling **RTX_SIGNAL_SEM**.

Using a resource semaphore:

A resource semaphore is created by specifying an initial count of -1 in the call of **RTX_CREATE_SEM**. The @CHIP-RTOS creates a resource semaphore and automatically gives it an initial value of one indicating that the resource is free. A resource is reserved by calling **RTX_RESERVE_RES**with the semaphore ID returned by RTX_CREATE_SEM. The resource is released with a call of **RTX_RELEASE_SEM**.

Semaphore Services:

RTX_CREATE_SEM	Create a semaphore
RTX_DELETE_SEM	Delete a semaphore
RTX_FREE_RES	Free a resource semaphore
RTX_GET_SEM	Get use of a counting semaphore(no wait)
RTX_RELEASE_SEM	Release a resource semaphore
RTX_RESERVE_RES	Reserve a resource semaphore
RTX_SIGNAL_SEM	Signal a counting semaphore
RTX_WAIT_SEM	Wait on a counting semaphore (optional timeout)

Top of list
Index page

Time / Date Services

The following Time/Date services are available.

RTX_GET_TIMEDATE Get system time and date
RTX_SET_TIMEDATE Set system time and date
RTX_GET_TICKS Get tick count of system clock

Related Topics

TimeDate_Structure **type** definition

Top of list
Index page

Timer Procedures

The @CHIP-RTOS API provides four calls for the usage of timer procedures:

RTX_INSTALL_TIMER Install a timer procedure
RTX_REMOVE_TIMER Remove a timer procedure from the system
RTX_START_TIMER Start periodic execution of a installed timer procedure
RTX_STOP_TIMER Stop periodic execution of a timer procedure

The @CHIP-RTOS implements timer procedures using kernel objects, of which there are 60 total. The sum of semaphores + event groups + timer procedures is limited to 60, since all three of these require use of a kernel object.

The kernel can execute periodic user timer procedures at a specified time interval. Your timer procedure must be as short as practical without any waiting or endless loops. Avoid the usage of large C-library functions such as `printf()`. However, any of the following kernel services are reasonable to call from within a timer procedure:

- **RTX_REMOVE_TIMER** Service 0x31
- **RTX_START_TIMER** Service 0x32
- **RTX_STOP_TIMER** Service 0x33
- **RTX_WAKEUP_TASK** Service 0x06
- **RTX_SIGNAL_SEM** Service 0x1A
- **RTX_SIGNAL_EVENTS** Service 0x42
- **RTX_SEND_MESSAGE** Service 0x52
- **RTX_SUSPEND_TASK** Service 0x0E
- **RTX_RESUME_TASK** Service 0x0F

Top of list
Index page

Event Manager

The internal @CHIP-RTOS Event Manager provides a convenient mechanism for coordinating tasks waiting for events with tasks and/or timer procedures which can signal the events. The Event Manager

allows more than one task to simultaneously wait for a particular event. Tasks can also wait for a particular combination of events or for any one in a set of events.

The Event Manager provides a set of event flags which can be associated with specific events in your system. These event flags are provided in groups with 16 event flags per group. The number of event groups which can be created is limited by the 60 kernel objects available. One kernel object is expended to create a semaphore, event group or timer procedure.

The Event Manager is useful when two or more tasks will wait for the same event, e.g. waiting for the start of a motor. An event flag is defined to represent the state of the motor (off or on). When tasks must wait for the motor, they do so by calling the Event Manager requesting a wait until the motor control event flag indicates that the motor is on. When the motor control task or timer procedure detects that the motor is on, it signals the event with a call to the Event Manager. The Event Manger wakes up all tasks which are waiting for the motor to be on. For further explanations read the function description in the API call specifications.

Event Services:

RTX_CREATE_EVENTGROUP Create an event group
RTX_DELETE_EVENTGROUP Delete an event group
RTX_SIGNAL_EVENTS Signal one or more events in a group
RTX_WAIT_EVENTS Wait for all/any of a set of events in a group
RTX_GET_EVENTGROUP_STATE Read current state of events in a group
RTX_GET_EVENT_FLAGS Get saved event flags
RTX_FIND_EVENTGROUP Find the group ID of an group with a specific name

Top of list
Index page

Message Exchange Manager

The internal @CHIP-RTOS Message Exchange Manager provides a mechanism for interprocess communication and synchronization. In particular, it offers an instant solution to a common producer/consumer problem:

> One or more processes (producers) having to asynchronously deliver requests for service to one or more servers (consumers) whose identity is unknown to the producer.

An often cited example of using message exchange is a print request queue. Assume that there are two different server tasks (consumers), each of which is connected to a different printer. There are some other tasks (producers) which want to asynchronously use one of the two servers for printing and they don't care which of the two printers is used. The solution is to synchronize those requests: The producer tasks send their requests (messages) to the Message Exchange Manager. The two server tasks waiting for messages take a message (if any) from the message queue and execute the requested print job.

The internal Message Exchange Manager uses a message exchange to deliver messages. A message exchange consists of four mailboxes into which messages can be deposited. The mailboxes are ordered according to priority (0-3), where mailbox 0 has the highest priority. Maximum depth of a mailbox is four.

Messages are delivered to the mailboxes of the message exchange in message envelopes. The system's maximum number of available messages envelopes is 64. The maximum number of message exchanges is ten. The maximum message length is 12 bytes. (Note: Larger messages can be implemented with a pointer and a length parameter in the message.) Any task or timer procedure can send a message to a Message Exchange. The sender indicates the priority of its message (0-3), thereby identifying the mailbox into which it will be delivered.

Any task or timer procedure can request a message from a Message Exchange, but only tasks are allowed to wait for the arrival of a message, if none is available. A task can specify the priority at which it

is willing to wait and the maximum time.

Message Exchange Services:

RTX_CREATE_MSG	Create a message exchange
RTX_DELETE_MSG	Delete a message exchange
RTX_SEND_MSG	Send a message to a message exchange
RTX_GET_MSG	Get a message, if any (no wait)
RTX_WAIT_MSG	Wait for message to arrive (optional timeout)
RTX_FIND_MSG	Find a message exchange, specified by name

Top of list
Index page

End of document

Data Structures

Here are the data structures used by the RTOS API. All constants and data structures are defined in the header file rtxapi.h

Notes:

- **Byte alignment is required** for all data structures used within the API.

Content :

RTX_Msg

```
typedef struct tag_rtx_msg{
  unsigned int  msgID;    // Unique Message Exchange ID
  char name[4];           // 4 characters, not null terminated
  int  mb0;               // Numbers of message envelopes which can reside
  int  mb1;               //  in each of the four exchange mailboxes,
  int  mb2;               //  maximum 4 envelopes.
  int  mb3;

} RTX_Msg;
```

Comments

This structure is defined in the header file rtxapi.h.

Prior to making the RTX_CREATE_MSG call, the caller must set the following members of the `RTX_Msg` data structure.

`name`
Here you can give the Message Exchange a unique four character name. This field is optional. If you plan to use the RTX_FIND_MSGAPI then you should provide a unique name here for all Message Exchanges.

`mb0` through **`mb3`**
State here the number of message envelopes for each of this Message Exchange's four priority mailboxes. These counts are truncated to a maximum of 4 envelopes per mailbox inside this API. The

system's maximum number of available messages envelopes is 64 (since @CHIP-RTOS version 1.02B, else 32).

The `msgID` member is an output parameter from the `RTX_CREATE_MSG` API call.

Related Topics

API function **RTX_CREATE_MSG** - Create a Message Exchange
RTOS **Message** Exchange Manager

Top of list
Index page

RTX_Wait_Event

```
typedef struct tag_rtx_event_wait
{
  unsigned int mask;   // 16-Bit mask identifying the flags of interest of the group.
  unsigned int value;  // 16 Bit value, which specifies the states of interest
                       //  for each flag selected by the mask.
  int match;           // event match requirements, 0: only one flag must match
                       // with value, !=0: all by mask specified flags must match
  long timeout;        // Maximum time (milliseconds) for waiting for an event match

} RTX_Wait_Event;
```

Comments

This structure is defined in the header file rtxapi.h.

Prior to making the **RTX_WAIT_EVENTS** call, the caller must set the members of the `RTX_Wait_Event` data structure.

`mask`
Event bits set to '1' in this mask indicate the events in `value`which will be matched. All other event bits in the group are ignored.

`value`
The desired event state for the event bits set to '1' in `mask`are stated here. User may wait on an event bit transition to either '1' or '0', as specified here.

`match`
This is a Boolean value which indicates whether you require all indicated events to match (AND, `match`!= 0), or whether it is sufficient that only a single event of those specified match (OR, `match`=0).

`timeout`
Maximum time to wait for events, specified in milliseconds. Zero here indicates wait forever. This signed value must be non-negative.

Related Topics

API function **RTX_WAIT_EVENTS** - Wait for events in a group
RTOS **Event** Manager

Top of list
Index page

RTX_Wait_Msg

```
typedef struct tag_rtx_wait_msg{
  unsigned int msgID; // ID of the message exchange
  unsigned int prio;  // Priority for wait [0 .. 0xFFFF], 0 = highest
  char far *msg;      // Pointer to user buffer to store the arrived message
  long  timeout;      // Maximum time (milliseconds) for waiting for a message

} RTX_Wait_Msg;
```

Comments

This structure is defined in the header file rtxapi.h.

Prior to calling the **RTX_WAIT_MSG** ALI, the caller must set the following members of the `RTX_Wait_Msg` structure.

`msgID`
Here you put the message exchange ID acquired by the **RTX_CREATE_MSG**call.

`prio`
Specify here the priority of the calling task's access to the messages. To wait in FIFO order, have all `RTX_WAIT_MSG`API callers use the same value here. A task can cut in line ahead of other waiting tasks by setting this field to a higher priority (lower number) than used by the other tasks. (Note that this priority has no connection to either **task**priority or Message send **mailbox**priority.)

`msg`
Put here a far pointer to a 12 byte buffer allocated in your application program memory. The Message Exchange Manager will copy the message to this buffer if a message is available.

`timeout`
Here you can specify the maximum number of milliseconds you are willing to wait for a message. A value of zero indicates you will wait forever. Negative values are not permitted.

Related Topics

API function **RTX_WAIT_MSG** - Wait for a message from a message exchange
RTOS **Message** Exchange Manager

Top of list
Index page

TaskDefBlock

```
typedef struct tag_taskdefblock
{
        void     (far *proc)();    // Task entry vector (far)
        char           name[4];    // Task name, 4 characters not null terminated
   unsigned int far *stackptr;     // Task stack pointer (far)
        unsigned int   stacksize;    // size of stack (bytes)
        unsigned short attrib;       // task attributes (not supported by the RTOS API)
        short int      priority;     // task priority,  range: 3..127 inclusive
        unsigned short time_slice;   // 0: none, !=0: number of milliseconds before task
                                     // is forced to relinquish processor
        short       mailboxlevel1;   // not used
        short       mailboxlevel2;   // not used
        short       mailboxlevel3;   // not used
        short       mailboxlevel4;   // not used

} TaskDefBlock;
```

Comments

This structure is defined in the header file rtxapi.h.

Prior to making the RTX_TASK_CREATE API call, the caller must set the following members of the TaskDefBlock structure.

proc

Here you must set the far vector to your task's entry point. The task's main procedure should be declared (depending on compiler used) as:

Borland C: `void huge taskfunc(void)`
Microsoft C: `void far _saveregs _loadds taskfunc(void)`

These assure that the data segment register (DS) is loaded on entry into your task.

name

Make up a four letter name for your task by which it can be uniquely identified. Avoid names already occupied by the **system**tasks.

stackptr

This far pointer should point to the top of you task's stack space (highest address). Your tasks stack pointer will be initialized to this value, which points to the first byte of memory following your actual stack space. (Note that x86 CPU decrements the stack pointer prior to pushing data onto the stack.) The stack memory space resides within your application program.

stacksize

Here you specify the size of your task's stack space in bytes. The amount of stack space required for your task depends on the nature of your task. If large automatic objects are declared in your task's procedures, a large amount of stack space will be needed. **Caution:**

Some of the system's interrupts use your task's stack. Consequently we recommend a minimum stack space of 1024 bytes per task.

Since the problems resulting from stack overflow are often difficult to diagnose and analyze, the following design steps are recommended:

1. Initially allocate way more stack space then you believe you will need.
2. When you have your task performing what it was designed to do, measure the amount of stack space being used by your task. The **RTX_GET_TASK_STATE** API can be used to obtain this measurement.
3. Refine your stack allocation based on this measurement, arriving at a compromise between the conflicting requirements: efficiency on the one hand (small stack desired) and program maintainability and reliability on the other hand (big stack desired).

Software maintainability becomes an issue here if you have the stack space wired so tight that the slightest code change will lead to stack overflow. Reliability is an issue when paths in your task (or interrupts) are executed that did not execute during your stack space measurement trials.

priority

Application program tasks can range in priority from 3 to 127 (inclusive), where 3 is higher priority. Generally, task priorities between 20 and 30 are recommended. This recommendation is based on the priority assignments of the built-in **system**tasks. Too high a priority for an application task may block important system tasks: e.g. the Ethernet receiver task. However, in some cases higher priority application tasks are appropriate design, but in these cases you must keep the execution dwell of these high priority task very short. User DOS applications are started at priority 25.

Note:

Internally all tasks have a unique priority. When a task is **created**or its priority is **changed**, that task is given a lower internal task priority than any other task in the system with the same user task priority.

time_slice

Set this value to zero if you do not want round-robin time slicing between this task and others at the same priority level. If you do want round-robin switching, then specify here the number of milliseconds of CPU time that this task should receive before the system switches to the next task at this same priority. System timing granularity is one millisecond. Time slicing is disabled on all DOS application main tasks.

mailboxlevelN

These provisional four values are not used in the current implementation.

Related Topics

API function **RTX_TASK_CREATE** - Create and start a task

Top of list
Index page

TaskList

```
typedef struct tagtasklist
{
     unsigned int taskID;         // Task handle
     char         taskname[5];    // Four character string terminated with zero.

} TaskList;
```

Comments

This structure is defined in the header file rtxapi.h

Related Topics

API function **RTX_GET_TASK_LIST** - Get list of current tasks in the system

Top of list
Index page

Task_StateData

```
typedef struct tag_task_statedata
{
    unsigned int taskID;      // Task handle
    unsigned int taskPrio;    // Task priority
    unsigned int taskState;   // Bit field
    unsigned int taskCount;   // Duty cycle indicator
    unsigned int stackused;   // Percentage of stack space used
    unsigned int stacksize;   // Task's total stack size, in bytes

} Task_StateData;
```

Comments

This structure is defined in the header file rtxapi.h. Task **Monitoring** mode must be enabled to obtain all the data listed here.

taskID

Task handle used for the RTX API functions.

taskPrio

Current task **priority**.

taskState

This bit field is zero for active tasks. The reasons for an inactive task waiting are coded as follows:

B0: Timer wait (used in combination with other bits: B2, B3, B4, B7)
B1: Trigger wait (i.e. idle, see note)
B2: Semaphore wait
B3: Event Group wait

B4: Message Exchange wait
B5: -- not implemented --
B6: Suspended (waiting for resume)
B7: Asleep waiting for wakeup
B8 - B15 internal use only

Notes:

- A task is in "trigger wait" state prior to starting (e.g. **RTX_TASK_CREATE_WITHOUT_RUN**) or after termination. A **RTX_RESTART_TASK** call exits this state.
- Where as B1, B2, B3, B4 and B7 are mutually exclusive wait conditions, B6 "**Suspended**" condition can be added (OR'ed in) to these other wait conditions.

taskCount

This value, ranging from 0 to 10000, provides a rough indication of the amount load placed on the system by this task. At each execution of the @CHIP-RTOS 1000 Hz real-time interrupt handler, this count is incremented if this particular task is either currently executing or if the CPU is idle and this task was the most recent task to be executed. The running count is recorded at 10 second intervals, so a task which is active 100% of the time would score a `taskCount`of 10000. (Same true if the task had been sleeping for the last 10 seconds, but it was the most recently active task ... oh well.)

stackused

This value is the percentage of the task's stack space used. The system presets stack space to all zeroes prior to starting tasks. The deepest (lowest address) non-zero value on the stack indicates the "high water mark". This value is only meaningful for tasks created within application programs. DOS programs normally switch stacks on entry, so this the stack space usage measurement is defeated.

stacksize

This is the total number of bytes in the task's stack space. This value is only meaningful for tasks created within application programs, due to that DOS programs normally switch stacks on entry.

Related Topics

API function **RTX_GET_TASK_STATE** - Get state of a task

Top of list
Index page

TimeDate_Structure

```
typedef struct tag_time
{
    unsigned char sec;      // Seconds  (0-59)
    unsigned char min;      // Minutes  (0-59)
    unsigned char hr;       // Hours    (0-23)
    unsigned char dy;       // Day      (1-31)
    unsigned char mn;       // Month    (1-12)
    unsigned char yr;       // Year     (0-99)
    unsigned char dow;      // Day of week (Mon=1 to Sun=7)
    unsigned char dcen;     // Century if time/date is correct

} TimeDate_Structure;
```

Comments

This structure is defined in the header file rtxapi.h.

Related Topics

API function **RTX_GET_TIMEDATE** - Get system time and date
API function **RTX_SET_TIMEDATE** - Set system time and date

TimerProc_Structure

```
typedef struct tag_timer_proc
{
   unsigned int  far *timerID;  // pointer to storage the unique timerID
   void (far *proc)();          // pointer to the procedure to be executed
   void  far *dummyptr;         // Optional parameter to timer procedure
   char  name[4];               // -- Field not used --
   long  interval;              // timer execution interval in milliseconds

} TimerProc_Structure;
```

Comments

This structure is defined in the header file rtxapi.h.

Before calling the **RTX_INSTALL_TIMER** API function the caller must allocate a `TimerProc_Structure` with the following members set as specified here:

`timerID`
Put here a far pointer to a 16 bit location in your program's memory space. The `RTX_INSTALL_TIMER`API function will output a Timer ID to this referenced location. This Timer ID value is used as handle for this new timer procedure within the other Timer Procedure API **functions**.

`proc`
This is a far vector to your timer procedure. This routine will be called periodically from the kernel. Your timer procedure should be declared (depending on compiler used) as:

Borland C: `void huge MyTimerProc(void)`
Microsoft C: `void far _saveregs _loadds MyTimerProc(void)`
Turbo Pascal:

```
procedure Timer1_Proc;interrupt;
begin

[... your code ...]

(*********************************************)
(* This is needed at the end of the Timer Proc. *)
     asm
     POP BP
     POP ES
     POP DS
     POP DI
     POP SI
     POP DX
     POP CX
     POP BX
     POP AX
     RETF
     end;
(*********************************************)
end;
```

so that the compiler will generate code to set the CPU's DS register, enabling access to your program's data. Have also a look to **some important notes**concerning Timer Procedures.

`dummyptr`
This four byte value is pushed onto the stack by the system as an input parameter to your timer procedure. You are free to ignore this parameter as suggested by the above `MyTimerProc(void)`declarations. Alternatively you could also use, for example, either of the following timer procedure declarations (Borland):

```
void huge MyTimerProc(unsigned int timer_id,
                      unsigned long lParam) ;
```

or

```
void huge MyTimerProc(unsigned int timer_id, void far *ptr) ;
```

The `timer_id`parameter will be the same as output to `timerID`location in this data structure. In either case, the second parameter is a copy of the four byte `dummyptr`.

`name`
This provisional field is not used by the system.

`interval`
Specify here the interval, in milliseconds, at which you want your timer procedure to be called.

Related Topics

API function **RTX_INSTALL_TIMER** - Install a timer procedure
RTOS **Timer** Procedures

Top of list
Index page

End of document

IPC@CHIP System Tasks

The RTOS itself creates a set of tasks used to support the IPC@CHIP services. Your application programs will be executing concurrently with these *built-in* tasks. The fact that these RTOS tasks exists is particularly relevant when you choose priorities for your own application program's tasks.

Each task in the IPC@CHIP system must have a unique four letter task name. Consequently the user must take care to avoid the names used by these system tasks when **naming** tasks.

System Task List

The priorities stated here are in decimal, where 0 is highest priority. Some tasks may not be present in your system due to your system configuration specified in the `chip.ini` configuration file. For these tasks the relevant configuration file parameter is stated here with a hyper-link.

```
AMXK     priority= 0     Kernel task
TCPT     priority= 4     TCP/IP timer task
ETH0     priority= 5     Ethernet receiver task
PPPS     priority= 6     PPP server (PPPSERVER ENABLE)
PPPC     priority= 6     PPP client (PPPCLIENT ENABLE)
CFGS     priority= 7     UDP config server
TELN     priority= 11    Telnet server (TELNET ENABLE)
MTSK     priority= 12    Console task (command shell)
WEBS     priority= 41    Web server (WEB ENABLE)
FTPS     priority= 41    FTP server (FTP ENABLE)
```

Related Topics

API function **RTX_TASK_CREATE** - Create and start a task

@CHIP-RTOS API Error Codes

All error codes here are stated in decimal.

RTOS error codes returned by RTOS **API** calls in the **DX-Register**

```
  0      call successful
 -1      RTOS call failed
 -2      RTOS API function number (AH input) is not supported
```

RTOS specific error codes returned by RTOS API calls in the **AX register**

```
  3      task is still waiting
  2      task not waiting; wake is pending
  1      no buffer available
  0      no error
 -1      invalid taskid
 -2      If DX=-1, no free message available;
         If DX=-2, Bad API function number
 -3      no mailbox defined
 -4      mailbox full
 -5      wakened before timeout
 -6      task not waiting (after 2nd wake)
 -7      calling task not waiting
 -8      invalid message call

-12      resource not owned by you (caller)
-13      no such buffer pool (invalid ID)
-14      not enough memory
-15      memory error
-16      memory error

-17      invalid task priority
-18      no free Task Control Block
-19      no free interval timer
-20      task abort (stop, kill, delete) not allowed
-21      access error
-22      invalid semaphore ID
-23      semaphore already in use
-24      invalid semaphore value

-27      timed out
-28      no message available
```

```
-29      calling task still waiting

-30      no buffers defined
-31      memory error

-32      no free event group
-33      event group in use

-37      memory not available
-38      invalid memory block
-39      memory block not in use
-40      memory block use count overflow

-41      no such message exchange (invalid ID)
-42      no free message exchange
-43      message exchange in use
-44      invalid message mailbox size

-45      no free semaphore
-46      no such event group (invalid ID)
-47      no such timer (invalid ID)
-48      invalid timing interval
-49      invalid result status
-50      memory fill exceeds segment limit
-51      semaphore is busy
-52      invalid task trap type
```

End of document

RTOS Application Developers Note - SC12 @CHIP-RTOS V1.10

@CHIP-RTOS Application Notes

RTOS **API** Listing

Developer Notes

The provided services are a subset of the RTOS calls. If it should be necessary, we will add needed functions in the future. The given examples should be used and modified by the API programmer. The API programmer should know something about the basics of programming multitasking applications!!

It's very important to declare large enough task **stack**. The example taskexp1 shows this problem. When the program is compiled with Microsoft C V1.52, we must set the stack size of the tasks to 3072 words. The same program compiled with Borland 4.52 requires only a stack size of 512 words. It's not advised to use the `printf` functions in a task procedure because it requires a lot of stack space. In the example program taskexp1.exe, we use `printf` calls and it works, but there is no guarantee that it will work in other applications. Timer procedures are executed on the stack of the kernel task, so they should be as short as possible. Avoid the calling of large C-Library functions like `printf`.

Task priorities:

We recommend the usage of a task priority between 20 and 30, because a task with a higher priority is able to block other important tasks of our system. e.g. the serial and Ethernet receiver tasks.

End of document

RTOS Examples Available - SC12 @CHIP-RTOS V1.10

RTOS Examples

Available RTOS **API** examples:

1. `taskexp1.c` Creating and starting tasks and usage of a resource **semaphore**
2. `tcpservm.c` tcpechoserver, which is able to serve a maximum of three clients at the same time
3. `timer.c` **Timer** procedure example for DK40
4. `Pastimer.c` **Timer** procedure example written in Turbo Pascal
5. `event.c` Example of using **event** groups
6. `msg.c` Example of using **message** exchange

The examples are compiled with Borland C 4.5 or 5.02.

Also we build a C-Library (rtos.c), which contains the described software interrupt calls.
All example programs built with C API-functions use the files rtos.c, rtos.h and rtxapi.h.

End of document

DOS Interface Documentation - SC12 @CHIP-RTOS V1.10

DOS

Here are the DOS interface definitions. DOS uses interrupt 0x21 with a service number in the high order byte of the AX register (AH).

For some useful comments see also under **Programming notes**

DOS API News

All implemented DOS services are listed here:

- Interrupt_0x21_function_0x41:_Delete File
- Interrupt_0x21_function_0x42:_Set Current File Position
- Interrupt_0x21_function_0x43:_Get/Set File Attributes
- Interrupt_0x21_function_0x44:_IOCTL, Set/Get Device Information
- Interrupt_0x21_function_0x47:_Get Current Working Directory
- Interrupt_0x21_function_0x48:_Allocate Memory
- Interrupt_0x21_function_0x49:_Free Allocated Memory
- Interrupt_0x21_function_0x4A:_Resize Memory
- Interrupt_0x21_function_0x4B:_EXEC
- Interrupt_0x21_function_0x4C:_End Process
- Interrupt_0x21_function_0x4E:_Find First File
- Interrupt_0x21_function_0x4F:_Find Next File
- Interrupt_0x21_function_0x50:_Debugger Support
- Interrupt_0x21_function_0x51:_Get PSP Segment Address
- Interrupt_0x21_function_0x56:_Change Directory Entry, Rename File
- Interrupt_0x21_function_0x57:_Get/Set File Date and Time
- Interrupt_0x21_function_0x58:_Get/Set memory strategy (dummy function)
- Interrupt_0x21_function_0x62:_Get PSP Segment Address
- Interrupt_0x21_function_0x63:_Get Leading Byte (stub)
- Interrupt_0x21_function_0x68:_Flush DOS Buffers to Disk

Any service not listed is not supported. A warning will be issued on the console when an unimplemented DOS interrupt 0x21 service is requested. If you need a function that is not supported, please let us know at mailto:atchip@beck-ipc.com

A maximum of 12 DOS programs can be run simultaneously.

All DOS tasks together can open a maximum of 10 files.

Interrupt 0x21 service 0x00: Terminate Program

Refer to interrupt 0x21, **service 0x4C**.

Parameters

AH
0x00

Comments

This service has been replaced by service 0x4C. The system treats both as the same function.

Top of list
Index page

Interrupt 0x21 service 0x02: Output Character to standard output

Sends the character in DL to the standard output.

Parameters

AH
0x02

DL
Character to be output to **stdout**

Return Value

Returns nothing

Comments

Each potential output device has its own output buffer. This function queues the provided output character into each device's output buffer for which **stdout** is configured.

The transmitters are interrupt driven buffered I/O. If space is available in the transmit buffer(s) when this call is made, the character is stored and control returned immediately to the caller. Otherwise a wait loop is entered, awaiting space in each configured transmit buffer.

This function does not check for Ctrl-C.

Related Topics

stdout configuration

Top of list
Index page

Interrupt 0x21 service 0x06: Direct Console Output

If DL!=0xFF: Send the character in DL to the standard output.
If DL==0xFF: read character from `stdin` if one is available.

Parameters

AH
0x06

DL
Character to be output to `stdout`

Return Value

If call with DL!=0xFF then no return value (only output to `stdout`)
If call with DL==0xFF then
    If input character is available at `stdin`then
        Set **BX**register to indicate the `stdin`channel source of character,
            where: 1: EXT , 2: COM , 4: Telnet.
        Return input character in **AL**and reset CPU's zero flag
    Else
        Set CPU's zero flag to indicate no character available

```
        Endif
Endif
```

Comments

Output is buffered and interrupt driven. This function will return after placing output character into the transmit buffer when DL != 0xFF.

This function does not check for Ctrl-C.

Related Topics

`stdin` configuration
`stdout` configuration

Top of list
Index page

Interrupt 0x21 service 0x07: Direct Console Input

Wait for a character to be read from standard input.

Parameters

AH
0x07

Return Value

Returns the character read in AL.
Returns in BX the source `stdin` channel of the character, where: 1: EXT , 2: COM , 4: Telnet

Comments

This function is identical to interrupt 0x21, **service 0x08**.

This function does not echo the received character and it does not check for Ctrl-C.

Related Topics

`stdin` configuration

Top of list
Index page

Interrupt 0x21 service 0x08: Read Keyboard

Wait for a character to be read from standard input.

Parameters

AH
0x08

Return Value

Returns the character read in AL.
Returns in BX the source `stdin` channel of the character, where: 1: EXT , 2: COM , 4: Telnet

Comments

This function is identical to interrupt 0x21, **service 0x07**.

This function does not echo the character and it does not check for Ctrl-C.

Related Topics

`stdin` configuration

Top of list
Index page

Interrupt 0x21 service 0x09: Send string to standard output

Sends a string to `stdout` ending with '$' or null terminated.

Parameters

AH
0x09

DS:DX
Specifies a pointer to the first character of the string.

Return Value

Returns nothing.

Comments

This function does not check for Ctrl-C.

Related Topics

`stdout` configuration

Top of list
Index page

Interrupt 0x21 service 0x0B: Character Available Test

Check if a character from standard input is available.

Parameters

AH
0x0B

Return Value

AL=0x00: No character is available.
AL=0xFF: Character is available.

Comments

This function does not check for Ctrl-C.

Related Topics

`stdin` configuration

Top of list
Index page

Interrupt 0x21 service 0x0E: Set Default Drive

Changes the default drive for the current task.

Parameters

AH
0x0E

DL
New default drive (00h = A:, 01h = B:, etc)

Related Topics

Get current drive

Top of list
Index page

Interrupt 0x21 service 0x19: Get Current Drive

Returns the current drive for this process.

Parameters

AH
0x19

Return Value

AL = drive where 0 is A:, 1 is B:, ..., 4 is E:

Related Topics

Set default drive

Top of list
Index page

Interrupt 0x21 service 0x1A: Set Disk Transfer Area Address

Sets address of the Disk Transfer Area (DTA) needed for `findfirst/findnext` functions.

Parameters

AH
0x2A

DS:DX
Pointer to DTA

Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task, which gets access to the file system (via **`RTX_ACCESS_FILESYSTEM`** service) reserves an entry in this list.

Only ten DTA entries for user tasks are available.

Related Topics

DOS **Get** Disk Transfer Area address service
RTOS's **`RTX_ACCESS_FILESYSTEM`** Service

Top of list
Index page

Interrupt 0x21 service 0x25: Set IRQ Vector

This function allows you to set an interrupt vector to your interrupt function.

Parameters

AH

0x25

AL
Specifies vector number.

DS:DX
Vector to your interrupt procedure.

Return Value

No return value.

Comments

You can use the following IRQ's

0x0A DMA0 / INT5
0x0B DMA1 / INT6
0x0C INT0
0x0E INT2
0x0F INT3
0x10 INT4

IRQ 0x0D (Ethernet) and IRQ 0x13 (Timer) cannot be changed !

Also this DOS service interrupt 0x21 vector cannot be changed using this service.

Related Topics

Get IRQ vector

Top of list
Index page

Interrupt 0x21 service 0x2A: Get System Date

Returns the system date.

Parameters

AH
0x2A

Return Value

CX=Year (full 4 digits), DH=Month, DL=Day, AL=day of week (0=Sunday)

Related Topics

Set system date
Get system time

Top of list
Index page

Interrupt 0x21 service 0x2B: Set System Date

Sets the system date.

Parameters

AH
0x2B

CX
Year (including century, e.g. 2001)

DH
Month (1..12)

DL
Day (1..31)

Comments

This function performs no error checking on entered date.

Related Topics

Get system date
Set system time

Top of list
Index page

Interrupt 0x21 service 0x2C: Get System Time

Returns the system time.

Parameters

AH
0x2C

Return Value

CH=Hour, CL=Minute, DH=Second, DL=0

Related Topics

Get system date

Set system time

Top of list
Index page

Interrupt 0x21 service 0x2D: Set System Time

Sets the system time.

Parameters

AH
0x2D

CH
Hour

CL
Minute

DH
Second

Related Topics

Set system date
Get system time

Top of list
Index page

Interrupt 0x21 service 0x2F: Get Disk Transfer Area Address

Gets address of the Disk Transfer Area (DTA) needed for `findfirst/findnext`.

Parameters

AH
0x2F

Return Value

Returns the address of the DTA in ES:BX

Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task, which gets access to the file system (via **`RTX_ACCESS_FILESYSTEM`** service) reserves an entry in this list.

Only ten DTA entries for user tasks are available.

Related Topics

DOS **Set** Disk Transfer Area address service
RTOS's **RTX_ACCESS_FILESYSTEM** Service

Top of list
Index page

Interrupt 0x21 service 0x30: Get DOS Version

Get the version number of DOS.

Parameters

AH
0x30

Return Value

Returns the DOS version in AX.

Comments

This function always returns 0x0003 (meaning DOS version 3.00).

However, this does not mean that we have a full implementation of DOS 3.0 !

Top of list
Index page

Interrupt 0x21 service 0x31: Keep Process

Makes a program remain resident after it terminates.

Parameters

AH
0x31

DX
Memory size, in paragraphs, required by the program

Return Value

None

Top of list

Interrupt 0x21 service 0x35: Get IRQ Vector

Gets the address of an interrupt service routine.

Parameters

AH
0x35

AL
Specifies vector number.

Return Value

Returns the vector in ES:BX

Related Topics

Set IRQ vector

Top of list
Index page

Interrupt 0x21 service 0x36: Get Disk Free Space

DOS function for detecting disk free space.

Parameters

AH
0x36

DL
Drive (0 current drive, 1=A, ...)

Return Value

AX: -1 Invalid drive number
else
AX: number of sectors per cluster
BX: number of free clusters
CX: number of bytes per sector
DX: number of clusters per drive

Comments

When call is successful (AX=1), free disk space can be computed from the return values as:

free disk space (bytes) = AX * BX * CX

Note:
In @CHIP-RTOS version 1.00, this function had a bug: Parameter 0 in DL was taken as drive A:, which should have meant the current drive.

Top of list
Index page

Interrupt 0x21 service 0x39: Create Directory

Creates a new subdirectory.

Parameters

AH
0x39

DS:DX
Pointer to null terminated path name.

Return Value

Carry flag is cleared on success, set on error.
On error AX contains error code:
2: File not found
3: Path not found
5: Path exists or access denied

Related Topics

Remove Directory

Top of list
Index page

Interrupt 0x21 service 0x3A: Remove Directory

Deletes a subdirectory.

Parameters

AH
0x3A

DS:DX
Pointer to null terminated path name.

Return Value

Carry flag is cleared on success, set on error.
On error AX contains error code:

2: File not found
3: Path not found
5: access denied, not a directory, not empty, or in use

Comments

The subdirectory must not contain any files.

Related Topics

Create Directory

Top of list
Index page

Interrupt 0x21 service 0x3B: Set Current Working Directory

Sets the current working directory.

Parameters

AH
0x3B

DS:DX
Null terminated path of new current working directory

Return Value

Carry flag is cleared on success, set on error.

Comments

If the path contains a drive name, the current working directory of that drive is changed without changing the default drive. Otherwise, the current working directory is changed for the default drive.

Each task has it's own current working directory.

Related Topics

Get current working directory

Top of list
Index page

Interrupt 0x21 service 0x3C: Create New File Handle

Creates a file of specified name. If a file by this name already exists, it is deleted. The returned file handle is for a new empty file.

Parameters

AH
0x3C

CX
File attributes (bit field):
B0 - Read Only
B1 - Hidden File
B2 - System File
B5 - Archive Flag

DS:DX
Pointer to a null terminated file name and path

Return Value

Success: Carry flag cleared, AX = file handle
Failure: Carry flag set, AX = error code:
AX=2: Path not found
AX=3: File name length exceeded 147 character limit
AX=4: Too many files open
AX=5: Invalid file name or access denied

Comments

Files are always opened in a non-sharing mode.

Related Topics

Open Existing File
Close File Handle
Get/Set File **Date/Time**

Top of list
Index page

Interrupt 0x21 service 0x3D: Open an Existing File

Opens an existing file.

Parameters

AH
0x3D

AL
Open mode:
AL=0: Open for read
AL=1: Open for write
AL=2: Open for read and write

DS:DX

Pointer to a null terminated file path.

Return Value

Success: Carry flag cleared, AX = file handle
Failure: Carry flag set, AX = error code:
AX=2: Path or file not found
AX=4: Too many files open
AX=5: Access denied

Comments

In write mode, files are always opened in a non-sharing mode.

The file system does not distinguish between file not found or invalid path. The error return value in both cases is 2.

Related Topics

Create New File Handle
Close File Handle
Get/Set File **Attributes**
Get/Set File **Date/Time**

Top of list
Index page

Interrupt 0x21 service 0x3E: Close File Handle

Closes an open file.

Parameters

AH
0x3E

BX
File handle

Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX = error code:
AX=6: file not open

Related Topics

Create New File Handle
Open Existing File
Get/Set File **Date/Time**

Top of list
Index page

Interrupt 0x21 service 0x3F: Read from File

Reads a number of bytes from a file, the handle of which is specified in BX.

Parameters

AH
0x3F

BX
File handle

CX
Number of bytes to read

DS:DX
Pointer to the destination data buffer.

Return Value

Success: Carry flag cleared, AX = number of bytes read into buffer from file
Failure: Carry flag set, AX = error code:
AX=5: Access denied
AX=6: Invalid file handle

Related Topics

Create New File Handle
Open Existing File
Write to File
Set Current File Position

Top of list
Index page

Interrupt 0x21 service 0x40: Write to File

Writes a number of bytes to a file, the handle of which is specified in BX.

Parameters

AH
0x40

BX
File handle

CX

Number of bytes to write

DS:DX
Pointer to the source data buffer.

Return Value

Success: Carry flag cleared, AX = number of bytes written into the file
Failure: Carry flag set, AX = error code:
AX=5: Access denied
AX=6: Invalid file handle

Comments

Requesting zero bytes written to file (CX=0) truncates the file at the current position.

Related Topics

Create New File Handle
Open Existing File
Read from File
Set Current File Position

Top of list
Index page

Interrupt 0x21 service 0x41: Delete File

Deletes a file. Wildcards are not allowed.

Parameters

AH
0x41

DS:DX
Pointer to null terminated file name and path.

Return Value

Success: Carry flag is cleared
Failure: Carry flag is set, AX holds error code:
AX=2: File not found
AX=5: Access denied

Comments

Files with read only attribute cannot be deleted.

Related Topics

Create New File Handle

Get/Set File **Attributes**

Top of list
Index page

Interrupt 0x21 service 0x42: Set Current File Position

The operating system maintains a 32 bit file pointer that it uses for read or write requests to the respective file. This service can be used to either read or set this file pointer. The file pointer associated with handle is set to a new byte position offset relative to the origin of the move.

Parameters

AH
0x42

AL
Origin of move
0x00: Relative to start of file
0x01: Relative to current position
0x02: Relative to end of file

BX
File handle

CX,DX
Offset of the displacement with higher order word in CX.

Return Value

Success: Carry flag is cleared, DX,AX holds the new position relative to the start of the file with high word in DX.
Failure: Carry flag is set, AX holds error code:
AX = 0x06: Invalid handle
AX = 0x19: Invalid displacement

Comments

If you attempt to seek beyond the end of file, the file pointer will be positioned at the end of the file.

To read current file position without changing it, call with AL=1, CX:DX = 0:0.

Related Topics

Read from File
Write to File

Top of list
Index page

Interrupt 0x21 service 0x43: Get/Set File Attributes

Use this function to inspect or change the attributes of a file.

Parameters

AH
0x43

AL
0: get, 1: set

CX
File attributes (bit field):
B0 - Read Only
B1 - Hidden File
B2 - System File
B5 - Archive Flag

DS:DX
Pointer to a null terminated string holding the file path.

Return Value

Success: Carry flag cleared, file attributes in CX (bit field):
... same flag bits as CX input parameter with additional bits ...
B3 - Volume
B4 - Directory Entry
Failure: Carry flag set, AX holds error code:
AX=1: Invalid function (wrong value in AL)
AX=2: File not found
AX=5: Access denied

Comments

Input parameter CX is not used when input parameter AL = 0.

Related Topics

Delete File

Top of list
Index page

Interrupt 0x21 service 0x44: IOCTL, Set/Get Device Information

Changes the data that DOS uses to control a device.

Parameters

AH
0x44

AL
0: Get device data, 1: Set device data

BX
Handle

DX
Device data

Return Value

Success: Carry flag cleared, Device data in DX.
Failure: Carry flag set, AX holds error code
AX=1: Invalid function (wrong value in AL)
AX=6: Invalid handle

Comments

If bit 7 of the data is 1, the handle refers to a device and data bit assignments are as follows:

```
Bit   Meaning when bit is set to '1'
B15    Reserved
B14    Device can handle function 0x44, codes 2 and 3
B13    Device supports output until busy
B12    Reserved
B11    Device understands open/close
B10-8  Reserved
B7     '1' indicates handle refers to a device
B6     Not "end of file" on input
B5     Don't check for control characters
B4     Reserved
B3     Clock device
B2     Null device
B1     Console Output device
B0     Console Input device
```

If bit 7 of the data is 0, the handle refers to a file and data bit assignments are as follows:

```
Bit   Meaning
B15-8 Reserved
B7    Set to '0' to indicate handle refers to a file.
B6    Set to '0' when the file has been written
B0-5  Drive number (0=A, 1=B, etc)
```

The first three file handles are used for the **stdio** devices:
0: Input
1: Output
2: Error

This service was implemented to be compatible with the older DOS compilers. The data is saved when you issue a write, but the data is not used by the @CHIP-RTOS. Control characters are not recognized as such. Function 0x44 codes 2 and 3 are not supported.

Top of list
Index page

Interrupt 0x21 service 0x47: Get Current Working Directory

Gets the current working directory for a drive.

Parameters

AH
0x47

DS:SI
Pointer to a 64 byte memory area to receive null terminated path of current working directory (CWD).

DL
Drive number, 0 for current, 1 for A, ..

Return Value

Success: Carry flag cleared, Buffer at [DS:SI] contains CWD path.
Failure: Carry flag set, error code in AX=15

Comments

Each task has it's own current working directory. When a program starts, its current drive and working directory will be set to the drive and directory that was current before the program started.

Related Topics

Set current working directory

Top of list
Index page

Interrupt 0x21 service 0x48: Allocate Memory

Allocates memory for the calling process.

Parameters

AH
0x48

BX
Size counted in paragraphs

Return Value

Success:
Carry flag cleared, AX holds the segment of the memory area
Failure:
Carry flag set due to not enough memory available.
AX = 0.
BX holds the size of the largest free block available expressed by paragraph count.

Comments

A paragraph is 16 bytes in length.

Related Topics

Free allocated memory
Resize memory

Top of list
Index page

Interrupt 0x21 service 0x49: Free Allocated Memory

Releases the specified allocated memory.

Parameters

AH
0x49

ES
Segment of the memory area to be released, allocated with function **0x48**

Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX holds the error code = 9.

Related Topics

Allocate memory
Resize memory

Top of list
Index page

Interrupt 0x21 service 0x4A: Resize Memory

Increases or decreases the size of a memory allocation block.

Parameters

AH
0x4A

BX
Desired new size expressed in paragraphs

ES
Segment address of memory block

Return Value

Success: Carry flag cleared
Failure: Carry flag set due to not enough memory available and the size of the largest free block is returned in BX (paragraph count).

Comments

A paragraph is 16 bytes in length.

This call may fail if a user tries in their application to increase a memory block, allocated with int21h 0x48. In that case it is only possible to decrease the size of the allocated memory block.

Related Topics

Allocate memory
Free memory

Top of list
Index page

Interrupt 0x21 service 0x4B: EXEC

Load and/or execute a program. This function loads the program and builds its PSP (Program Segment Prefix) based on a parameter block that you provide.

Parameters

AH
0x4B

AL
Type of load:
00h - Load and execute
01h - Load but do not execute

DS:DX
Null terminated program name (must include extension)

ES:BX
parameter block

```
Offset  Size     Description 00h    WORD     Segment of environment to copy
for child process
                  (copy caller's environment if 0000h)
02h    DWORD    Pointer to command tail to be copied into child's PSP
06h    DWORD    Pointer to first FCB to be copied into child's PSP
0Ah    DWORD    Pointer to second FCB to be copied into child's PSP
0Eh    DWORD    (AL=01h) will hold subprogram's initial SS:SP on return
12h    DWORD    (AL=01h) will hold entry point (CS:IP) on return
```

Return Value

Success:
Carry flag cleared

AX = New program's **task**ID
BX = segment of PSP (add sizeof PSP/16 for relocation offset)

Failure:
Carry flag set
AX = 1

Comments

Use 'Type of Load' AL = 0x00 to load and execute another program.

'Type of Load' AL = 0x01 is used to load a program without executing it. This option is available for debuggers. The new task is waiting for it's **trigger** when started in this manner.

For both functions, the calling process must ensure that there is enough unallocated memory available; if necessary, by releasing memory with services **AH=0x49** or **AH=0x4A**.

Top of list
Index page

Interrupt 0x21 service 0x4C: End Process

Terminates a DOS program.

Parameters

AH
0x4C

Comments

The memory used by the process is released, with following exception. This function will not free system memory **allocated** by a task that was created within a program. Only memory allocated by the program's main task will be freed here.

Related Topics

Delete @CHIP-RTOS task

Top of list
Index page

Interrupt 0x21 service 0x4E: Find First File

Find first file matching file name specification and attribute. The results are stored in the Disk Transfer Area (DTA).

Parameters

AH
0x4E

CX
File attribute

DS:DX
Null terminated file specification (may include path and wildcards)

Return Value

Success: Carry flag cleared, search results are in DTA
Failure: Carry flag set

Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. This DTA address should be set with interrupt 0x21 function **0x1A** before calling this `findfirst` function. The `findfirst`/**`findnext`** sequence is handled by your compiler library so the Disk Transfer Area is therefore not described here.

Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task which gets **access** to the file system reserves an entry in this list.

Only ten DTA entries for user tasks are available.

Related Topics

Find **next** file
RTOS **RTX_ACCESS_FILESYSTEM** Service
Fast **Findfirst** file

Top of list
Index page

Interrupt 0x21 service 0x4F: Find Next File

Finds the next file in the `findfirst/findnext` sequence. The results are stored in the Disk Transfer Area (DTA).

Parameters

AH
0x4F

Return Value

Success: Carry flag cleared, search results are in DTA
Failure: Carry flag set

Comments

The task that calls `findnext` must be the same task that called `findfirst`.

Since the entire state of a `findfirst/findnext` sequence is held in the DTA data block, other disk

operations such as renaming, moving, deleting, or creating files can cause inaccurate directory searches such as finding the same file twice. Please look at the `findfirst` function (Interrupt 0x21, function **0x4E**) for further restrictions.

The `findnext` function is designed to be called in a loop until it fails, which indicates the last file has been found.

Related Topics

Find **first** file

Top of list
Index page

Interrupt 0x21 service 0x50: Debugger Support

Gets address of internal program task list and sets callback vector used for program startup notification.

Parameters

AH
0x50

DX:BX
Callback vector

CX
Sanity Check = 0x8765

Return Value

DX:BX contains pointer to internal task list.
AX = Task list length = 12
CX = size of task list elements
SI = RTOS Private Data Segment

Comments

This function is intended only for debugger usage.

Top of list
Index page

Interrupt 0x21 service 0x51: Get PSP Segment Address

Gets PSP (Program Segment Prefix) segment address

Parameters

AH
0x51

Return Value

BX contains the PSP segment address, if BX=0, PSP was not found

Comments

This function will not work if called from tasks **created** within programs.

This function is identical to interrupt 0x21 service **0x62**.

Top of list
Index page

Interrupt 0x21 service 0x56: Change Directory Entry, Rename File

Rename a file by changing its directory entry

Parameters

AH
0x56

DS:DX
Pointer to null terminated old file name

ES:DI
Pointer to null terminated new file name (without path name!)

Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX holds error code
AX: 1 - File not found
AX: 4 - New file name already exists
AX: 5 - Internal error
AX: 7 - Directory update failed

Top of list
Index page

Interrupt 0x21 service 0x57: Get/Set File Date and Time

Get or set file date and time. The file is specified by file handle.

Parameters

AH
0x57

AL
AL=0 to get the date and time of the last modification.
AL=1 to set the file date and time.

BX
File handle

CX
if AL==1: Time in the format described below

DX
if AL==1: Date in the format described below

Return Value

Success: Carry flag cleared, if input parameter AL=0 then:
CX = time of last modification
DX = date of last modification
Failure: Carry flag set

Comments

The time/date registers are coded as follows:
CX time of last modification

```
bits 15-11:  hours (0..23)
bits 10-5:   minutes
bits 4-0:    seconds/2
```

DX date of last modification

```
bits 15-9:   year-1980
bits 8-5:    month
bits 4-0:    day
```

Related Topics

Create New File Handle
Open Existing File

Top of list
Index page

Interrupt 0x21 service 0x58: Get/Set memory strategy (dummy function)

Get/Set memory strategy, only a dummy function for compatibility

Parameters

AH
0x58

AL
AL=0 Get strategy
AL=1 Set strategy.

BX
(Al=1) Strategy 0: First fit, 1: Best fit, 2: Last fit

Return Value

If parameter AL == 0 (get strategy)

AX contains the memory strategy dummy value
Carry flag cleared

If parameter AL == 1 (set strategy)

```
IF input parameter BX >2 THEN
   AX = 1
ELSE
   AX = input parameter BX
```

Comments

This is only a dummy function, added for compatibility.

The @CHIP-RTOS has its own fixed memory strategy. Memory is always allocated in the following manner:

DOS programs are always loaded into the lowest free memory block in the @CHIP-RTOS heap memory area. For memory blocks allocated internally in the @CHIP-RTOS or with **Int21h 0x48**, the @CHIP-RTOS searches for a free memory block starting at the highest heap RAM address. So the largest free memory block of the system is always located in the middle of the @CHIP-RTOS heap memory area.

The shell command **mem** shows the state of the internal memory map.

Top of list
Index page

Interrupt 0x21 service 0x62: Get PSP Segment Address

Gets PSP (Program Segment Prefix) segment address

Parameters

AH
0x62

Return Value

BX contains the PSP segment address, if BX=0, PSP was not found

Comments

This function will not work if called from tasks **created** within programs.

This function is identical to interrupt 0x21 service **0x51**.

Top of list
Index page

Interrupt 0x21 service 0x63: Get Leading Byte (stub)

Dummy function, not supported by the IPC@CHIP @CHIP-RTOS

Parameters

AH
0x63

Return Value

Always returns with Carry flag set
AL = 0xFF
DS = 0
SI = 0

Top of list
Index page

Interrupt 0x21 service 0x68: Flush DOS Buffers to Disk

Flush DOS file buffers to disk for specified file.

Parameters

AH
0x68

BX
file handle

Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX holds error code
AX : 2 Invalid handle
AX : 7 I/O error occurred

Related Topics

Create New File Handle
Open Existing File
Close File Handle

Top of list
Index page

DOS API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

DOS API News

The following extensions to the DOS **API** are available in the indicated @CHIP-RTOS revisions.

No changes since last version.

End of document

Hardware API - SC12 @CHIP-RTOS V1.10

Hardware API

Here are the interface definitions for access to the IPC@CHIP's hardware.

Topics

Hardware API **Layer**Model
Hardware API **News**

API Functions

The hardware API uses interrupts 0xA2 (**PFE** functions) and 0xA1 (**HAL** functions) with a service number in the high order byte of the AX register (AH). The implemented hardware services are listed below.

For some useful comments see also under **Programming notes**

- **Interrupt_0xA2_function_0x80:_PFE: Enable Data Bus**
- **Interrupt_0xA2_function_0x81:_PFE: Enable Non-Multiplexed Address Bus**
- **Interrupt_0xA2_function_0x82:_PFE: Enable Programmable I/O Pins**
- **Interrupt_0xA2_function_0x83:_PFE: Enable Programmable Chip Selects**
- **Interrupt_0xA2_function_0x84:_PFE: Enable External Interrupt Requests**
- **Interrupt_0xA2_function_0x85:_PFE: Enable External Timer Inputs/Outputs**
- **Interrupt_0xA2_function_0x86:_PFE: Set Edge/Level Interrupt Mode**
- **Interrupt_0xA2_function_0x87:_PFE: Enable PWD Mode**
- **Interrupt_0xA2_function_0x88:_PFE: Enable External DMA**
- **Interrupt_0xA2_function_0x89:_PFE: Enable INT0 / INTA cascade mode**
- **Interrupt_0xA2_function_0x8A:_PFE: Set wait states for PCS0-3**
- **Interrupt_0xA2_function_0x90:_PFE: Get Hardware API Function Pointers**
- **Interrupt_0xA1_function_0x10:_HAL: Set int0 Vector**
- **Interrupt_0xA1_function_0x80:_HAL: Read Data Bus**
- **Interrupt_0xA1_function_0x81:_HAL: Write Data Bus**
- **Interrupt_0xA1_function_0x82:_HAL: Read Programmable I/O Pins**
- **Interrupt_0xA1_function_0x83:_HAL: Write Programmable I/O Pins**
- **Interrupt_0xA1_function_0x84:_HAL: Install Interrupt Service Routine**
- **Interrupt_0xA1_function_0x85:_HAL: Initialize Timer Settings**
- **Interrupt_0xA1_function_0x86:_HAL: Start Timer**
- **Interrupt_0xA1_function_0x87:_HAL: Stop timer**

Interrupt 0xA2 service 0x80: PFE: Enable Data Bus

Initialize data bus I/O mask and ALE usage.

Parameters

AH

Must be 0x80.

AL

0: Disable ALE, 1: Enable ALE

DX

Mask

Bit 0 = 0: Data bus bit 0 is input, 1: is output
Bit 1 = 0: Data bus bit 1 is input, 1: is output
:
:
Bit 7 = 0: Data bus bit 7 is input, 1: is output
Bit 8..15 not used (for future extensions)

Return Value

none

Comments

The I/O mask defines which data bits on the bus are inputs and which are outputs. The DX mask bit for bi-directional data bus lines (read/write) should be set to '1'.

used pins:
  ALE, AD[0..7], RD#, WR#
excluded pins:

if ALE is used, then PCS0# is not available.

Top of list
Index page

Interrupt 0xA2 service 0x81: PFE: Enable Non-Multiplexed Address Bus

The IPC@CHIP has three non-multiplexed address bit outputs, A0 through A2. The enabling of these pins is done here.

Parameters

AH
Must be 0x81.

DX
Mask
Bit 0 = 1 Enable A0
Bit 1 = 1 Enable A1
Bit 2 = 1 Enable A2
Bit 3..15 not used

Return Value

none

Comments

used pins:
A[0..2], AD[0..7], RD#, WR#
excluded pins:
If A0 is enabled then PCS1#, TMRIN0, PIO4 are not available
If A1 is enabled then PCS[5..6]#, TMRIN1, TMROUT1, PIO3 are not available
If A2 is enabled then PCS[5..6]#, PIO2 are not available.

Top of list
Index page

Interrupt 0xA2 service 0x82: PFE: Enable Programmable I/O Pins

Enable used programmable I/O pins. Define which pins are inputs and which are outputs.

Parameters

AH
Must be 0x82.

AL
Mode
0 = Only read PIO state
1 = Input without pullup/pulldown
2 = Input with pullup (not PIO13)

3 = Input with pulldown (only for PIO3 and PIO13)
4 = Output init value = High
5 = Output init value = Low

DX

PIO pin
Bit 0 = 1 Enable PIO0
Bit 1 = 1 Enable PIO1
:
:
Bit 13 = 1 Enable PIO13
Bit 14..15 not used (for future extensions)

Return Value

AX = wPIO
Bit 0 = 1: PIO0 is PIO
Bit 1 = 1: PIO1 is PIO
:
:
Bit 13 = 1: PIO13 is PIO
Bit 14..15 not used (for future extensions)
DX = wINPUTS (all pins, including non-PIO pins)
Bit 0 = 1: PIO0 is input
Bit 1 = 1: PIO1 is input
:
:
Bit 13 = 1: PIO13 is input
Bit 14..15 not used (for future extensions)
CX = wOUTPUTS (all pins, including non-PIO pins)
Bit 0 = 1: PIO0 is output
Bit 1 = 1: PIO1 is output
:
:
Bit 13 = 0: PIO13 is output
Bit 14..15 not used (for future extensions)
AX = DX = CX = 0, Error: Wrong arguments

Comments

This function can be called several times for definition of different PIO pins. With repeated selection of the same pin, the definition made last is valid. The selection of a PIO pin can be cancelled by calling the appropriate PFE function that causes the respective PIO pin to be used for another purpose (e.g. function **0x83** for PIO[2..6] pins).

used pins:
PIO[0..13]
excluded pins:
All other functionality on the selected PIO pin.

Related Topics

Read Specific I/O Pin
Write to Specific I/O Pin
Read Programmable I/O Pins
Write Programmable I/O Pins
Initialize the **I2C** Bus

Top of list
Index page

Interrupt 0xA2 service 0x83: PFE: Enable Programmable Chip Selects

Enable chip selects PCS[0..3]#, PCS[5..6]#.

Parameters

AH
Must be 0x83.

DX
Mask
Bit 0 = 1 Enable PCS0#, active when I/O address between 000h..0FFh
Bit 1 = 1 Enable PCS1#, active when I/O address between 100h..1FFh
Bit 2 = 1 Enable PCS2#, active when I/O address between 200h..2FFh
Bit 3 = 1 Enable PCS3#, active when I/O address between 300h..3FFh
Bit 4 = don't care
Bit 5 = 1 Enable PCS5#, active when I/O address between 500h..5FFh
Bit 6 = 1 Enable PCS6#, active when I/O address between 600h..6FFh
Bit 7..15 = don't care

Return Value

none

Comments

used pins:
PCS[0..3]#, PCS[5..6]#
excluded pins:
if PCS0#: ALE (multiplexed address / data bus)
if PCS1#: A0, PIO4, TMRIN0
if PCS2#: PIO6, INT2, INTA#, PWD, hw flow control serial port 1,
cascaded interrupt controller
if PCS3#: PIO5, INT4, hw flow control serial port 1
if PCS5#: A[1..2], PIO3, TMROUT1, TMRIN1
if PCS6#: A[1..2], PIO2

Top of list
Index page

Interrupt 0xA2 service 0x84: PFE: Enable External Interrupt Requests

Enable external interrupt requests INT[0], INT[2..6].

Parameters

AH
Must be 0x84.

DX

Mask
- Bit 0 = 1 Enable INT0
- Bit 1 = don't care
- Bit 2 = 1 Enable INT2
- Bit 3 = 1 Enable INT3
- Bit 4 = 1 Enable INT4
- Bit 5 = 1 Enable INT5
- Bit 6 = 1 Enable INT6
- Bit 7..15 = don't care

Return Value

none

Comments

used pins:
- INT0, INT[2..6]

excluded pins:
- if INT0: PIO13, TMROUT0, cascaded interrupt controller
- if INT2: PIO6, PCS2#, INTA#, PWD, hw flow control serial port 1
- if INT3: PIO12, serial port 1
- if INT4: PIO5, PCS3#, hw flow control serial port 1
- if INT5: PIO1, DRQ0, default I²C-Bus pins
- if INT6: PIO0, DRQ1, default I²C-Bus pins

Related Topics

Set **Edge/Level** Interrupt Mode

Top of list
Index page

Interrupt 0xA2 service 0x85: PFE: Enable External Timer Inputs/Outputs

Enable external timer inputs (TMRIN0, TMRIN1) or timer outputs (TMROUT0, TMROUT1).

Parameters

AH

Must be 0x85.

DX

Mode

```
Bit 0..1 = 10 Enable TMRIN0
         = 11 Enable TMROUT0
Bit 2..3 = 10 Enable TMRIN1
         = 11 Enable TMROUT1
Bit 4..15 = don't care
```

Return Value

none

Comments

If on a given timer the external input is selected, then that timer's external output is not available and vice-versa.

used pins:
TMRIN[0..1], TMROUT[0..1]
excluded pins:
if TMRIN0: A0, PCS1#, PIO4, TMROUT0
if TMRIN1: A[1..2], PCS5#, TMROUT1
if TMROUT0: PIO13, INT0, cascaded interrupt controller, TMRIN0
if TMROUT1: A[1..2], PCS5#, TMRIN1, PIO3

Related Topics

HAL **Initialize** Timer Settings

Top of list
Index page

Interrupt 0xA2 service 0x86: PFE: Set Edge/Level Interrupt Mode

Set edge/level interrupt mode for INT0, INT2, INT3, INT4.

Parameters

AH
Must be 0x86.

AL
1 = active high, level-sensitive interrupt
0 = low-to-high, edge-triggered interrupt

DX
Mask, bits set to designate interrupts affected:
Bit 0 = INT0
Bit 1 = don't care
Bit 2 = INT2
Bit 3 = INT3
Bit 4 = INT4
Bit 5..15 = don't care

Return Value

none

Comments

Default for all interrupts is edge-triggered mode. In each case (edge or level) the interrupt pins must remain high until they are acknowledged.

Level-sensitive mode for INT5 / INT6 is not supported. The INT5 / INT6 interrupts operate only in edge-triggered mode.

Related Topics

Enable External Interrupt Requests

Top of list
Index page

Interrupt 0xA2 service 0x87: PFE: Enable PWD Mode

Enable Pulse Width Demodulation (PWD)

Parameters

AH
Must be 0x87.

Return Value

none

Comments

In PWD mode, TMRIN0, TMRIN1, INT2 and INT4 are configured internal to the chip to support the detection of rising (INT2) and falling (INT4) edges on the PWD input pin and to enable either timer0 when the signal is high or timer1 when the signal is low. The INT4, TMRIN0 and TMRIN1 pins are not used in PWD mode and so are available for use as PIO's.

The ISR for the INT2 and the INT4 interrupts should examine the current count of the associated timer, timer1 for INT2 and timer0 for INT4, in order to determine the pulse width. The ISR should then reset the timer count in preparation for the next pulse.

Overflow conditions, where the pulse width is greater than the maximum count of the timer, can be detected by monitoring the MaxCount bit in the associated timer or by setting the timer to generated interrupt requests.

used pins:
PWD
excluded pins:
TMRIN0, TMRIN1, TMROUT0, TMROUT1, INT4, INT2
PCS2#, INTA#, PIO6, hw flow control serial port 1

Top of list
Index page

Interrupt 0xA2 service 0x88: PFE: Enable External DMA

Enables DRQ pin to start DMA transfer

Parameters

AH
Must be 0x88.

AL
DRQ channel:
0 = DRQ0
1 = DRQ1

Return Value

AX = 0 no error
AX = -1 invalid DRQ channel
AX = -2 DMA channel is used for serial interface

Comments

You must disable the serial port DMA mode to use external DMA. This is done with a **CHIP.INI entry**.
The COM port uses DRQ1 and the EXT port uses DRQ0.

used pins:
DRQ[0..1]
excluded pins:
if DRQ0: PIO1, INT5, default I²C-Bus pins
if DRQ1: PIO0, INT6, default I²C-Bus pins

Top of list
Index page

Interrupt 0xA2 service 0x89: PFE: Enable INT0 / INTA cascade mode

Enable INT0 / INTA cascade mode

Parameters

AH
Must be 0x89.

Return Value

none

Comments

To install a service interrupt routine in cascade mode, the
HAL function **0x84** "Install Interrupt Service Routine" can not be used, because
the cascaded interrupt controller supply the interrupt type over the bus.
Install a normal interrupt service routine (ISR) using the `setvect` function.
At the end of your ISR, issue an EOI to both the cascaded controller and to
the **internal interrupt controller** (INT0).

used pins:
INT0, INTA#
excluded pins:
PIO13, TMROUT0, INT0 in normal mode, PIO6, PCS2#, PWD, hw flow control serial port 1

Related Topics

HAL 0x8E Give EOI

Top of list
Index page

Interrupt 0xA2 service 0x8A: PFE: Set wait states for PCS0-3

Set wait states for programmable chip selects PCS0#-PCS3#

Parameters

AH
Must be 0x8A.

AL
Bit 0-3
0000b = 0 wait states
0001b = 1 wait states
0010b = 2 wait states
0011b = 3 wait states
1000b = 5 wait states
1001b = 7 wait states
1010b = 9 wait states
1011b = 15 wait states
Bit 4-7
don't care

Return Value

none

Comments

Default for PCS0#-PCS3# are 15 wait states.

Related Topics

Enable programmable chip selects

Top of list
Index page

Interrupt 0xA2 service 0x90: PFE: Get Hardware API Function Pointers

Get the Function Pointers to Hardware API Functions, so that the functions can be called directly (without Software Interrupt). This is much faster.

Parameters

AH
: Must be 0x90.

ES:DI
: Pointer to a `HwApiFunctionStruct` data structure which will be filled with vectors to @CHIP-RTOS Hardware API Functions.

Return Value

AX= -1 if `size` parameter was too large. The `size` field of the structure will be set in this case to the number of supported vectors.

AX= 0 on success

Comments

Note that the `size` member of the `HwApiFunctionStruct` serves as both an input and output parameter. The caller must set this value to the number of vectors to be filled into the data structure by this API call. In the event that this value exceeds the number of vectors available, the AX value returned will be -1 and this API will set the `size` to 2, which is the number of vectors available by this API's current implementation.

In any case, the resulting count in `size` indicates the number of vectors output to the caller's `HwApiFunctionStruct` data structure at [ES:DI] by this API call.

```
typedef struct
{
   int size; // number of function entries
   unsigned  (huge *readPios)(void);
   void      (huge *writePios)(unsigned);
} HwApiFunctionStruct;

// Example for usage:

int main()
{
  HwApiFunctionStruct hwApi;
  unsigned            pios;

  [...]
  hwApi.size = 2;          // space for two functions in the struct
  pfe_get_hwapi_func_ptr(&hwApi);

  hwApi.writePios(0x55);   // write something to the pios
  pios = hwApi.readPios(); // read something from the pios
  [...]
}
```

Top of list
Index page

Interrupt 0xA1 service 0x10: HAL: Set int0 Vector

Define the interrupt handler for the hardware interrupt 0

Parameters

AH
Must be 0x10

ES:DX
Pointer to your interrupt handler

Return Value

none

Comments

The interrupt handler should be defined as
```
void interrupt my_handler(void)
```

Interrupts are enabled upon entry into your routine.

There is no need to signal any end of interrupt. This is handled by the system after your handler performs it's return.

The stack size must be at least 400 bytes.

Do not use any floating point arithmetic in your interrupt service routine.

Developer Notes

This function is only for compatibility to older version of the hardware API.
Please use interrupt **0xA2** function 0x84 and interrupt **0xA1** function 0x84 instead.

Top of list
Index page

Interrupt 0xA1 service 0x80: HAL: Read Data Bus

Read from specified address.

Parameters

AH
Must be 0x80.

DI
Address

BX
wAND

CX
wXOR

Return Value

8 Bit data in ax, ax = (databus & wAND) ^ wXOR

Comments

& = bit wise AND
^ = bit wise XOR

The result is combined with wAND and wXOR parameters. To read the data bus without change, set wAND=0xFFFF and wXOR=0x0000.

Related Topics

Block **Read** Data Bus
Write Data Bus

Top of list
Index page

Interrupt 0xA1 service 0x81: HAL: Write Data Bus

Write to specified address.

Parameters

AH
Must be 0x81.

DI
Address

DL
8 bit data

BX
wAND

CX
wXOR

Return Value

none

Comments

& = bit wise AND
^ = bit wise XOR

The provided parameters are combined as follows to form the output byte value:

```
output value = (data & wAND) ^ wXOR
```

To write the value in DL to the address without modification, set wAND=0xFFFF and wXOR=0x0000.

Related Topics

Block **Write** Data Bus
Read Data Bus

Top of list
Index page

Interrupt 0xA1 service 0x82: HAL: Read Programmable I/O Pins

Read the programmable I/O pins.

Parameters

AH
Must be 0x82.

BX
wAND

CX
wXOR

Return Value

ax = (PIO[0..13] & wAND) ^ wXOR.

Comments

& = bit wise AND
^ = bit wise XOR

The result is combined with the wAND and wXOR parameters. To read the PIO pins without modification, set wAND=0xFFFF and wXOR=0x0000. To read only the input pins, set wAND = wPIO & wINPUTS. The **wPIO** and **wINPUTS** values are return values from PFE **function** 0x82.

Related Topics

Read **Specific** I/O Pin
Write Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

Top of list
Index page

Interrupt 0xA1 service 0x83: HAL: Write Programmable I/O Pins

Write to the programmable I/O pins.

Parameters

AH
Must be 0x83.

DX
data applied to PIO[0..13]
where DX bit 0 maps to PIO[0]
and DX bit 13 maps to PIO[13]

BX
wAND

CX
wXOR

Return Value

none

Comments

& = bit wise AND
^ = bit wise XOR

Before the value is written, the value is combined with the wAND and wXOR parameters as:

```
PIO[0..13] = (data & wAND) ^ wXOR
```

To write value in DX to the programmable I/O pins without change, set wAND=0xFFFF and wXOR=0x0000.

Only PIO pins that are defined as outputs can be written.

Related Topics

Write **Specific** I/O Pin
Read Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

Top of list
Index page

Interrupt 0xA1 service 0x84: HAL: Install Interrupt Service Routine

Install user interrupt service routine to be invoked by system interrupt handler.

Parameters

AH
Must be 0x84.

DH
Bit mask for the ISR type:
BIT0..6 must be 0

BIT7: 0=normal ISR, 1=RTX ISR (RTX ISR allow RTX Calls, e.g. wake a task), please notice the comment below!

DL

HAL interrupt number from following list:

0 = INT0 (external)
1 = Network controller (internal) (*)
2 = INT2 (external)
3 = INT3 (external)
4 = INT4 (external)
5 = INT5 (external) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt Channel 1 (if DMA is used)
7 = reserved
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = Timer 1ms (internal) (*)
11 = Serial port 0 (internal)(*)
12 = Serial port 1 (internal)(*)
13 = reserved
14 = reserved
15 = NMI (internal/external)
(* = internal used by @CHIP-RTOS, not available for user interrupt service functions)

CX

Number of interrupts generated before new user interrupt service routine is called.
CX = 0 disables the user ISR (same as a NULL in ES:BX).

ES:BX

far pointer to user interrupt service routine
if pointer is NULL user ISR is disabled

Return Value

Far pointer to old handler in ES:BX

Comments

The user-defined ISR is called from a system ISR with the interrupt identifier number in the BX CPU register, thus allowing for a single user ISR to handle multiple interrupt sources. The user ISR can be declared in either of the following forms:

```
void huge My_ISR(void) ;     // More efficient form

void interrupt My_ISR(void) ;     // Tolerated form
```

The more efficient `huge` procedures are recommended. For assembly language ISR implementations, a far RET is recommended and a IRET on exit is tolerated. The user ISR function must preserve only the DS and BP registers, so there is no requirement for the full register save/restore provided by the `interrupt` declarations.

Any required EOI signal is issued by the system ISR which calls your user ISR function. This EOI is issued after your function returns.

If you install a RTX ISR you can use the following RTX calls in your ISR:

RTX_RESTART_TASK
RTX_WAKEUP_TASK
RTX_SIGNAL_SEM

RTX_SIGNAL_EVENTS
RTX_SEND_MSG
RTX_START_TIMER
RTX_STOP_TIMER
RTX_REMOVE_TIMER

Important Notes:
A RTX ISR is slower than a normal ISR. RTX ISR are not recommended for INT5 or INT6 if DMA is used by the **Fossil** serial ports interface, because the slower RTX ISR could result in UART receiver character loss.

If you are using a RTX ISR for timer0, timer1, INT5 or INT6 there must not exist a normal ISR on the system which enables the interrupts during its execution! Do not install a RTX ISR for NMI.!

Also important : The NMI function of the multifunction pin 17 (RESET/NMI/LINK_LED) of the IPC@CHIP SC11/SC12/SC13 is for power fail purposes only. It is not possible to use NMI as a "normal" interrupt pin like INT0 for generating interrupts. It can only be used as described in the IPC@CHIP hardware documentation.

Top of list
Index page

Interrupt 0xA1 service 0x85: HAL: Initialize Timer Settings

Initialize the timer settings of timer0 or timer1.

Parameters

AH
Must be 0x85.

AL
Timer
0=Timer0 / 1=Timer1

DX
Mode
Bit 0: 0=run single time / 1=run continuous
Bit 1: 0=disable timer interrupt / 1=enable timer interrupt
Bit 2: 0=use internal clock / 1=use TMRIN pin as external clock
Bit 3..15: not used

CX
Clock divider (maximum count value)

Return Value

none

Comments

The clock divider value serves as a comparator for the associated timer count. The timer count is a 16 bit value that is incremented by the processor internal clock (see HAL **function** 0x8A) or can also be configured to increment based on the TMRIN0 or TMRIN1 external signals (see PFE **function** 0x85). The TMROUT0 und TMROUT1 signals can be used to generate waveforms of various duty cycles. The

default is a 50% duty cycle waveform (Change waveform with HAL **function** 0x8B).

Note that TMRIN pin and TMROUT pin can not be used at the same time.

If the clock divider value is set to 0000h, the timer will count from 0000h to FFFFh (maximum divider). When the timer reaches the clock divider value, it resets to 0 during the same clock cycle. The timer count never dwells equals to the clock divider value (except for special case when divider value is set to 0000h).

When the timer is configured to run in single time mode, the timer clears the count and then halts on reaching the maximum count (clock divider value).

If the timer interrupt is enabled, the interrupt request is generated when the count equals the clock divider value. Use HAL **function** 0x84 to install your interrupt service routine.

If "use internal clock" is selected the associated TMRIN pin serves as a gate. A "high" on the TMRIN pin keeps the timer counting. A "low" holds the timer value.

Related Topics

Initialize Timer Settings Ext
HAL **Start** Timer function
Read Timer Count
Write Timer Count

Developer Notes

The timer output frequency is dependent on the internal processor clock.
For compatibility with future versions of @Chip, please use the HAL **function** 0x8A, "Get frequencies", to compute the correct clock divider value.

Available examples
1. TimerIn example, timerin.c
2. TimerOut example, timerout.c

Top of list
Index page

Interrupt 0xA1 service 0x86: HAL: Start Timer

Enable the specified timer to count.

Parameters

AH
Must be 0x86.

AL
Timer
0=Timer0 / 1=Timer1

Return Value

none

Related Topics

HAL **Initialize** Timer Settings
Stop Timer function

Top of list
Index page

Interrupt 0xA1 service 0x87: HAL: Stop timer

Stop the specified timer's counting

Parameters

AH
Must be 0x87.

AL
Timer
0=Timer0 / 1=Timer1

Return Value

none

Comments

The specified timer is disabled.

Related Topics

Start Timer function

Top of list
Index page

Interrupt 0xA1 service 0x88: HAL: Read Timer Count

Read the timer count.

Parameters

AH
Must be 0x88.

AL
Timer
0=Timer0 / 1=Timer1

Return Value

AX = Counter reading
DX = 1=MaxCount reached / 0=MaxCount not reached

Comments

AX contains the current count of the associated timer. The count is incremented by the processor internal clock (see HAL **function** 0x8A), unless the timer is configured for external clocking (then it is clocked by the TMRIN0 and TMRIN1 **signals**).

Related Topics

HAL **Initialize** Timer Settings
Stop Timer function
Write Timer Count

Top of list
Index page

Interrupt 0xA1 service 0x89: HAL: Write Timer Count

Preset the specified timer's count register to provided value.

Parameters

AH
Must be 0x89.

AL
Timer
0=Timer0 / 1=Timer1

DX
Value to write to 16 bit counter

Return Value

none

Comments

The timer count can be written at any time, regardless of whether the corresponding timer is running.

Related Topics

HAL **Initialize** Timer Settings
Stop Timer function
Read Timer Count

Top of list
Index page

Interrupt 0xA1 service 0x8A: HAL: Get Frequencies

Get the system frequencies.

Parameters

AH

Must be 0x8A.

AL

Which frequency to get:

0 = Return processor frequency
1 = Return timer base frequency
2 = Return maximum baud rate
3 = Return PWD timer frequency

Return Value

DX:AX frequency [Hz]

Comments

Use the timer base frequency to compute the correct timer clock divider value, where:

```
Output frequency = timer base frequency / clock divider
```

Use the maximum baud rate compute the correct baud rate for the processor specific baud rate initialize function (See Fossil *Extended line control initialization* **function**).

```
Baud rate = maximum baud rate / baud rate divider
```

Related Topics

HAL **Initialize** Timer Settings

Top of list
Index page

Interrupt 0xA1 service 0x8B: HAL: Set Timer Duty Cycle Waveform

Set the duty cycle waveform of specified timer.

Parameters

AH

Must be 0x8B.

AL

Which Timer
0=Timer0 / 1=Timer 1

DX

Mode
0=disable duty cycle / 1=enable duty cycle

CX

Alternate clock divider (if DX = 1)

Return Value

none

Comments

Use this function to modify the timer waveform behavior. For example a 50% duty cycle waveform can be generated by specifying here an *alternate clock divider* value in CX that is the same value as was used for the main *clock divider* value set in the Timer Initialization **function** call.

Please note that the timer frequency will change if you use this function. If you disable the duty cycle, the timer output will no longer generate a rectangle signal. When duty cycle mode is disabled, the TMROUT pin switches low for only one clock cycle after the maximum count is reached. If you want an alternate duty cycle waveform, compute it with the following formula:

```
Output frequency = Timer base frequency * 2
            / (divider high level + divider low level)
```

The `divider high level` is the divider set by the Timer Initialization **function**. The `divider low level` is the alternate clock divider passed to this function in the CX register.

Related Topics

HAL **Initialize** Timer Settings

Top of list
Index page

Interrupt 0xA1 service 0x8C: HAL: Read Specific I/O Pin

Read specified programmable I/O pin.

Parameters

AH

Must be 0x8C.

AL

IPC@CHIP PIO No. [0..13]

Return Value

AX=0 PIO pin is low, AX!=0 PIO pin is high

Related Topics

Write to Specific I/O Pin
Read Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

Top of list
Index page

Interrupt 0xA1 service 0x8D: HAL: Write to Specific I/O Pin

Write to a specified programmable I/O pin. Only PIO pins that are defined as outputs can be written.

Parameters

AH
Must be 0x8D.

AL
IPC@CHIP PIO No. [0..13]

DX
DX = 0 ==> set PIO to low
DX non-zero ==> set PIO to high

Return Value

none

Related Topics

Read Specific I/O Pin
Write Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

Top of list
Index page

Interrupt 0xA1 service 0x8E: HAL: Give EOI

Give End-Of-Interrupt for INT0-INT4

Parameters

AH
Must be 0x8E.

AL
Type (0=INT0 ... 4=INT4)

Return Value

none

Comments

When installing a interrupt service routine through HW API, it's not necessary to call this function, because the HW API does it for you.
This function is provided for writing your own interrupt service routines without the HAL function **Install Interrupt Service Routine**.

This function is especially needed for generating an EOI for INT0 when using cascaded mode of the interrupt controller with INT0/INTA .

Related Topics

Enable INT0/INTA cascade mode

Top of list
Index page

Interrupt 0xA1 service 0x8F: HAL: Initialize Timer Settings Ext

Some more useful timer settings (extends the Initialize Timer Settings **function**)

Parameters

AH
Must be 0x8F.

AL
Timer
0=Timer0 / 1=Timer1

DX
Mode
Bit 0..2: must be 0
Bit 3: 0=disable prescale t2 / 1=enable prescale t2
Bit 4: 0=disable retrigger / 1=enable retrigger
Bit 5..15: must be 0

Return Value

none

Comments

The Initialize Timer Settings **function** must be called prior to this function!

Prescale T2:
If the Prescale feature is enabled, the timer (specified in AL) will use Timer2 output for its timer base (clock input), providing a 1000 Hz timer clock rate. (Timer2 is used internally by the @CHIP-RTOS as a one millisecond interval timer.) If this Prescale bit is disabled, the timer base will

instead be the frequency reported by the **Get Frequencies**API function.

Retrigger:
If the retrigger setting is enabled, a 0 to 1 edge transition on TMRIN0 (or TMRIN1) resets the timer count. When retrigger is disabled (DX bit 4 set to 0), a High input on TMRIN0 (or TMRIN1) enables counting and a Low input holds the timer value.

Related Topics

Initialize Timer Settings
HAL **Start** Timer function
Read Timer Count
Write Timer Count

Top of list
Index page

Interrupt 0xA1 service 0x90: HAL: Get/Set Watchdog Mode

Get or set the watchdog mode.

Parameters

AH
Must be 0x90.

AL
Mode
0 = only get mode
2 = Watchdog will be triggered by user program
3 = Watchdog will be triggered by @CHIP-RTOS (default)

Return Value

AX=mode

Comments

The watchdog timeout period is 800 ms.

If you select the *User Program* mode, you must call the HAL **Refresh** Watchdog function 0x91 at least every 800 ms to prevent the watchdog from resetting the system. In @CHIP-RTOS mode, the @CHIP-RTOS performs the watchdog strobing provided that the system's timer interrupt is allowed to execute. Beware that excessive interrupt masking periods can lead to system resets.

Related Topics

Refresh Watchdog Function

Top of list

Interrupt 0xA1 service 0x91: HAL: Refresh Watchdog

Strobe the hardware watchdog to reset its timeout period.

Parameters

AH
Must be 0x91.

Return Value

none

Comments

If the watchdog is in *User Program* mode, this function must be called at least every 800 ms to prevent a CPU reset due to watchdog timeout.

Related Topics

Get/Set **Watchdog** Mode

Top of list
Index page

Interrupt 0xA1 service 0x92: HAL: Mask/Unmask Int

Mask or unmask an external Interrupt Request

Parameters

AH
Must be 0x92.

AL
1=Mask, 0=Unmask

DX
HAL interrupt number from following list:
0 = INT0 (external)
1 = Network controller (internal)
2 = INT2 (external)
3 = INT3 (external)
4 = INT4 (external)
5 = INT5 (external) / Terminal Count DMA Channel 0 (if DMA is used)
6 = INT6 (external) / Terminal Count DMA Channel 1 (if DMA is used)
7 = reserved
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = Timer 1ms (internal)

11 = Serial port 0 (internal)
12 = Serial port 1 (internal)
13 = reserved
14 = reserved
15 = NMI (Not maskable!)

Return Value

none

Comments

Some interrupts share the same mask bit. If you mask one of them, the other interrupts which are assigned to the same bit are also masked. Here are the groups which are masked together:

Timer0, Timer1, Timer 1ms
DMA0, INT5
DMA1, INT6

CAUTION:
Masking any of the three timer interrupts will suspend the @CHIP-RTOS 1000 Hz real-time interrupt, essential for system operation. Consequently this mask period should be very brief, if used at all.

Top of list
Index page

Interrupt 0xA1 service 0xA0: HAL: Block Read Data Bus

Read block of bytes from data bus into provided buffer.

Parameters

AH
Must be 0xA0.

DI
First address

SI
Second address

ES:BX
Pointer to buffer

CX
Number of bytes to read into buffer

Return Value

none

Comments

IF SI != DI, this function will alternate reads between the two addresses until CX bytes are read, starting at first address. Set SI == DI if you want to read from only a single address.

Related Topics

Block **Write** Data Bus
Read Data Bus

Top of list
Index page

Interrupt 0xA1 service 0xA1: HAL: Block Write Data Bus

Output byte stream from buffer to specified address or addresses.

Parameters

AH
Must be 0xA1.

DI
First address

SI
Second address

ES:BX
Pointer to buffer

CX
Number of bytes in buffer to output

Return Value

none

Comments

IF SI != DI, this function will alternate between writes to first and second address. Set SI == DI if you want all data written to a single address.

Related Topics

Block **Read** Data Bus
Write Data Bus

Top of list
Index page

Interrupt 0xA1 service 0xB0: HAL: Start DMA Mode

Starts the DMA mode. After calling this function, the DMA transfer will be started when the external DRQ pins is activated.

Parameters

AH

Must be 0xB0

AL

DMA channel:
0 = DRQ0
1 = DRQ1

CX

Number of bytes which has to be transferred

DX

Control Register:
Bit 0: 1=Priority for the channel / 0=Priority for the other channel
Bit 1: 1=Source synchronized / 0=No source synchronization
Bit 2: 1=Destination synchronized / 0=No destination synchronization
Bit 3: 1=Use interrupt at end of transfer / 0=do not use an interrupt
Bit 4: must be set to '1'
Bit 5: 1=Source address increment / 0=No increment of source address
Bit 6: 1=Source address decrement / 0=No decrement of source Address
Bit 7: 1=Source is in memory space / 0=Source is in IO space
Bit 8: 1=Destination address increment / 0=No increment of destination address
Bit 9: 1=Destination address decrement / 0=No decrement of destination address
Bit 10: 1=Destination is in memory space / 0=Destination is in IO space
Bit 11: 1=Word Transfer / 0=Byte Transfer

Bit1 and Bit2 can't be set at the same time.

BX:SI

Pointer to unsigned long (20-Bit physical) source address

ES:DI

Pointer to unsigned long (20-Bit physical) destination address

Return Value

Success: AX = 0
AX = -1 Invalid DMA channel
AX = -2 DMA channel used for serial interface

Comments

This function starts a DMA transfer. After calling this function the DMA controller is ready for transfer. Once the DRQ pin is activated the DMA transfer will be started. The number of bytes which will be transferred has to be specified in the CX register. Before calling this function you have to enable the DRQ pin for this channel (also the serial DMA mode must be disabled in the **CHIP.INI**).

Related Topics

Enable external DRQ Pin

HAL **Stop** DMA Transfer

Top of list
Index page

Interrupt 0xA1 service 0xB1: HAL: Stop DMA Transfer

Disables the DMA controller. A running DMA transfer will be halted.

Parameters

AH
Must be 0xB1

AL
DMA channel:
0 = DRQ0
1 = DRQ1

Return Value

Success: AX = 0
AX = -1 Invalid DMA channel
AX = -2 DMA channel used for serial interface

Comments

Stops a DMA transfer (disables the DMA controller). The transfer could be continued by reading the current DMA values (using function **Get DMA Info**) and starting the DMA Transfer again with the read values (using function **Start DMA Mode**)

Related Topics

Start DMA Transfer

Top of list
Index page

Interrupt 0xA1 service 0xB2: HAL: Get DMA Info

Get the state of the DMA channel.

Parameters

AH
Must be 0xB2

AL
DMA channel:
0 = DRQ0

1 = DRQ1

BX:SI

Output Parameter: Pointer to unsigned long where this function will write the (20-Bit physical) source address

ES:DI

Output Parameter: Pointer to unsigned long where this function will write the (20-Bit physical) destination address

Return Value

AX = 0 DMA channel disabled
AX = 1 DMA channel (user mode) enabled for transfer
AX = -1 Invalid DMA channel
AX = -2 DMA channel used for serial interface
CX = DMA counter (bytes which have yet to be transferred)
DX = Control Register: Bit 0: 1=Priority for the channel / 0=Priority for the other channel
Bit 1: 1=Source synchronized / 0=No source synchronization
Bit 2: 1=Destination synchronized / 0=No destination synchronization
Bit 3: 1=Use interrupt at end of transfer / 0=do not use an interrupt
Bit 4: must be set to '1'
Bit 5: 1=Source address increment / 0=No increment of source address
Bit 6: 1=Source address decrement / 0=No decrement of source Address
Bit 7: 1=Source is in memory space / 0=Source is in IO space
Bit 8: 1=Destination address increment / 0=No increment of destination address
Bit 9: 1=Destination address decrement / 0=No decrement of destination address
Bit 10: 1=Destination is in memory space / 0=Destination is in IO space

Bit 11: 1=Word Transfer / 0=Byte Transfer
[BX:SI] = contains unsigned long (20-Bit physical) DMA source address
[ES:DI] = contains unsigned long (20-Bit physical) DMA destination address

Comments

This function returns the status of the DMA channel in AX.

Related Topics

Start DMA Transfer

Top of list
Index page

Interrupt 0xA1 service 0xC0: HAL: Initialize/Restore Non-Volatile Data

Initialize/Restore non-volatile data. Tell the @CHIP-RTOS where your variables are located, which should be saved and reload their saved values, if available.
The non-volatile (remanent) data is stored in `A:\rema.bin` file.

Parameters

AH

Must be 0xC0

ES:BX

Pointer to a _REMOP structure:

```
struct _REMOP
{
    unsigned entries; // Number of entries in struct
                      // REMOP_ENTRY x[].
    unsigned segment; // Common segment address

    struct REMOP_ENTRY
    {
        unsigned offs;      // Address offset
        unsigned size;      // Number of bytes
        unsigned maxsize;   // Obsolete, set to 0
        unsigned elemsize;  // Number of bytes per data
                            // element
        unsigned distance;  // Distance to next data element
                            // (must be >= elemsize).
    }x[MAX_RETENTIVE_AREAS];
};
```

Return Value

Success: AX = 0
Failure: AX < 0, Could not create file

Comments

Call this function at the beginning of your program.

The _REMOP structure reference by ES:BX must be static. (This function does not make a copy of the structure's content.)

The number of entries in the x array of REMOP_ENTRY data structures can be defined by the user. This number must be specified in the `entries` field of the _REMOP data structure.

All data saved / restored must reside in the same segment, specified by the `segment` field in _REMOP. The individual data item locations in this segment are then listed in the `offs` fields of the REMOP_ENTRY structure array.

Related Topics

Save Non-Volatile Data

Top of list
Index page

Interrupt 0xA1 service 0xC1: HAL: Save Non-Volatile Data

This function saves your non-volatile data listed in the _REMOP structure registered with HAL **function** 0xC0. The data is stored in `A:\rema.bin` file.

Parameters

AH

Must be 0xC1

Return Value

none

Comments

Call this function on exit from your program and in your NMI (Non-Maskable Interrupt) handler. Your hardware around the IPC@CHIP must support the Pfail signal, so that the IPC@CHIP can generate an NMI sufficiently in advance of power loss to the CPU.

Reminder : The DK40 does not support the Pfail signal.

Related Topics

Initialize/Restore Non-Volatile Data
Install **Interrupt** Service Routine

Top of list
Index page

Interrupt 0xA1 service 0xC2: HAL: Get Reboot Reason

Check cause of most recent reboot.

Parameters

AH
Must be 0xC2

Return Value

AX = reason:
0 = UNKNOWN
3 = WATCHDOG
4 = POWER FAIL

Comments

This function only returns valid results if the init/restore **function** (0xC0) was called following the reboot.

Related Topics

Initialize/Restore Non-Volatile Data

Top of list
Index page

End of document

Hardware API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

Hardware API Updates

The following extensions to the hardware **API** are available in the indicated @CHIP-RTOS revisions.

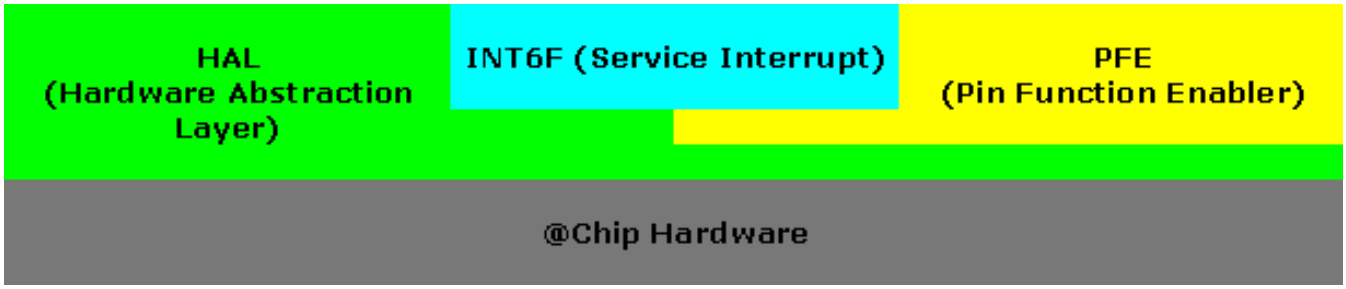
New in version 1.10B: **New HAL function for some more Timer features**
New in version 1.10B: **Modified HAL 84h: Allow to install an RTX ISR**
New in version 1.10B: **New HAL for mask/unmask external interrupts**

End of document

Hardware API Layers - SC12 @CHIP-RTOS V1.10

Hardware API Layers

Layer model:



1. **INT 6Fh - Service Interrupt Handler**

   The service interrupt provides software compatibility with the Beck FEC product line. This interrupt is used only in Beck products based on the IPC@CHIP and is not part of the standard @CHIP-RTOS. These services use the PFE and HAL interfaces to contact the actual hardware.

2. **PFE - Pin Function Enabler**

   This part of the hardware **API** provides functions to control the IPC@CHIP's multi-function I/O pins.

3. **HAL - Hardware Abstraction Layer**

   This part of the hardware API provide an isolation layer between your application software and the IPC@CHIP hardware (PIO pins, timers, etc.) which minimizes hardware dependencies.

End of document

I2C Bus / SPI API

Here are the interface definitions for access to the IPC@CHIP's I2C bus and Software SPI.

Philips was the inventor of the Inter-IC or I²C-bus, and it is now firmly established as the worldwide de-facto solution for embedded applications. It is used extensively in a variety of microcontroller-based professional, consumer and telecommunications applications as a control, diagnostic and power management bus. As a two-wire serial bus, its inherently simple operation was crucial to its emergence as the worldwide de-facto standard.

SPI is a serial bus standard established by Motorola. The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be Serial EEPROMs, shift registers, display drivers, A/D converters, etc.

- **I2C Bus / SPI API News**

API Functions

The I2C Bus / SPI API provides interrupt 0xAA with a service number in the high order byte of the AX register (AH). This interface provides access to the I2C Bus and Software SPI of the IPC@CHIP for application programs.

Interrupt 0xAA service 0x80: Initialize the I2C Bus

This function initializes the I2C bus. It enables the two defined PIO pins for I2C usage (see also function 8eh and 8fh).

Parameters

AH
0x80

Comments

The user can specify which two PIO are used for I2C **clock** and **data**. After calling this initialization function, these two pins will no longer be available as PIO pins unless the **PFE Enable** function is called for these pins following this function call.

Related Topics

Select I2C **Clock** Pin
Select I2C **Data** Pin
PFE: **Enable** Programmable I/O Pins

Top of list
Index page

Interrupt 0xAA service 0x81: Scan I2C Devices

Report addresses of slave devices, one at a time.

Parameters

AH
0x81

AL
First slave address (even address, LSB=0)

CL
Last slave address (even address, LSB=0)

Return Value

AL: 0 no slave found
AL: -1 Timeout
AL: address of the first found slave

Comments

This is an iterator function which is called repetitively to determine all connected slaves. Specify on each successive call a new restricted slave address range until no further address is returned by this function.

Top of list
Index page

Interrupt 0xAA service 0x82: Transmit / Receive Character

Send or receive a single character.

Parameters

AH
0x82

AL
Slave address, LSB:0 => Transmit, LSB:1 => Receive

CL
If Transmit: CL = Byte to transmit
If Receive: CL = 0 for last char to be received

Return Value

Success: Carry flag cleared and CH contains received byte (if receiving)
Failure: Carry flag set and AL contains I2C error **code**

Comments

The IPC@CHIP is the I2C bus master.

The least significant bit of the slave address determines the direction of the communication.
Even address: Master sending to slave
Odd address: Master receiving from slave

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

I2C error codes:

8: Timeout
9: Slave faulty or not available

Related Topics

I2C Transmit/Receive **Block**

Top of list
Index page

Interrupt 0xAA service 0x83: Transmit/ Receive Block

Parameters

AH
0x83

AL
Slave address, LSB:0 => Transmit, LSB:1 => Receive

CX
Number of bytes to transmit or receive

ES:BX
Buffer address

Return Value

Success: Carry flag cleared
Failure: Carry flag set and AL contains I2C error **code**

Comments

If an odd slave address is specified in AL then this function will dwell here until either CX bytes are received and stored in user buffer at [ES:BX], or until an error occurs. For an even slave address this function will dwell here until CX bytes from user buffer at [ES:BX] are transmit or until an error occurs.

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

Related Topics

I2C Transmit/Receive **Character**

Top of list
Index page

Interrupt 0xAA service 0x84: Release I2C Bus

Parameters

AH
0x84

Comments

This function generates an I2C stop condition. (The Bus will be released). At next call of transmit or receive functions, the address will be send again.

Top of list
Index page

Interrupt 0xAA service 0x8b: Restart the I2C Bus

Parameters

AH
0x8b

Return Value

CF: 0

Comments

This function generates an I2C start condition. (The Bus will be reserved)

Top of list
Index page

Interrupt 0xAA service 0x8E: Select I2C Clock Pin

Select IPC@CHIP I/O pin to be used for I2CCLK signal.

Parameters

AH
0x8E

AL
PIO pin number, [0..13]

Return Value

none

Comments

The default I2C clock pin is PIO 0
To change the I2CCLK pin this function must be called before the I2C initialize **function** (0x80) is called.

Related Topics

Initialize I2C Bus Function
Select I2C **Data** Pin

Top of list
Index page

Interrupt 0xAA service 0x8F: Select I2C Data Pin

Select IPC@CHIP I/O pin to be used for I2CDAT signal.

Parameters

AH
0x8F

AL
PIO pin number, [0..13]

Return Value

none

Comments

The default I2C data pin is PIO 1
To change the I2CDAT pin this function must be called before the I2C initialize **function** (0x80) is called.

Related Topics

Initialize I2C Bus Function
Select I2C **Clock** Pin

Top of list
Index page

Interrupt 0xAA service 0x90: SPI Init

Init the SPI Interface.

SPI specifies four signals: clock (SCLK); master data output, slave data input (MOSI); master data input, slave data output (MISO); and slave select (CSS). The slave select signal pin is not specified with the SPI init function and not activated with the SPI read and write functions. The user has to enable any other PIO pin for that purpose and should serve this pin manually.

Parameters

AH
Must be 0x90

BX
Mode, must be 0

CL
Pio number for SPI Clock Pin (SCLK)

CH
Pio number for SPI Input Pin (MISO)

DL
Pio number for SPI Output Pin (MOSI)

Return Value

none

Comments

none

Related Topics

Read SPI Block
Write SPI Block

Top of list
Index page

Interrupt 0xAA service 0x95: SPI Write Block

Write n bytes out to the SPI Output pin

Parameters

AH
Must be 0x95.

ES:SI
Pointer to write buffer

CX
Number of bytes in buffer to write

Return Value

none

Comments

None

Related Topics

Init SPI
Read Block SPI

Top of list
Index page

Interrupt 0xAA service 0x96: SPI Read Block

Reads n bytes from the SPI Input pin

Parameters

AH
Must be 0x96

ES:DI
Pointer to buffer for storing read data

CX
Number of bytes to read

Return Value

none

Comments

None

Related Topics

Init SPI
Write SPI Block

Top of list
Index page

Interrupt 0xAA service 0x97: SPI Read and Write Block

Reads n bytes from the SPI Input pin and write n bytes out to the SPI Output pin simultaneously.

Parameters

AH
Must be 0x97

DS:DI
Pointer to buffer for storing read data

ES:SI
Pointer to buffer which stores write data

CX
Number of bytes to read and write

Return Value

none

Comments

None

Related Topics

Init SPI
Write SPI Block
Read SPI Block

Top of list
Index page

End of document

I2C Bus and SPI API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

I2C Bus / SPI API News

The following extensions to the I2C/SPI **API** are available in the indicated @CHIP-RTOS revisions.

New in version 1.10B: **Implemented SPI Functions**
New in version 1.10: **Implemented a combined SPI read / write function**

End of document

FOSSIL

Here is the API definition for access to the COM and EXT serial ports.

The @Chip-RTOS offers the Fossil API for serial port communication. The Fossil standard uses software interrupt 0x14. These functions provide access to the @Chip-RTOS internal serial port driver for receiving and sending data via the serial ports.

Here is a short description how the @Chip-RTOS internal serial port driver operates:

Each serial port has two software buffers (queues), one for data pending transmission and one for storing received data bytes. The default size of each queue is 1024 bytes. The size of these transmit and receive queues are configurable via **chip.ini** entries.

Transfers between these software queues and the serial port hardware are carried out by either Direct Memory Access (DMA) hardware or by hardware interrupt (IRQ) driven software. By default, the two available DMA machines are applied to the two serial port receivers. This configuration reduces the likelihood of character loss at the receiver. This leaves the two serial port transmitters operating with hardware interrupt (IRQ) driven software. Alternately, receivers can be configured for interrupt driven mode (IRQ receive mode) with the **chip.ini** options, which frees up the DMA device for other usage. The DMA can be applied to transmitters with the SERIAL **SEND_DMA** option.

The serial port hardware issues a signal to either software (IRQ hardware interrupt) or to the DMA when ever the serial port transmit register is empty or a receiver byte is ready. This signal initiates the next byte transfer between the appropriate software queue and the hardware register, in or out. (This discussion is slightly over simplified. The driver actually uses an additional intermediate RAM buffer for DMA transfers.)

Note that the serial port hardware send/receive buffers are only one byte deep. So interrupt driven receivers (as opposed to DMA driven) can easily lose characters, particularly at higher baud rates.

The COM / EXT serial ports may also be referred to as a UART (abbreviation for "Universal Asynchronous Receiver/Transmitter").

For some useful comments see **Programming notes**

New in version 1.10B: **Install a user callback function**
New in version 1.10B: **Enable/Disable UART receiver**
New in version 1.10B: **Enable/Disable UART transmitter**

- **Interrupt_0x14_function_0x00:_Set baud rate**
- **Interrupt_0x14_function_0x01:_Put byte in output buffer, wait if needed.**
- **Interrupt_0x14_function_0x02:_Get a byte from the serial port, wait if none available.**
- **Interrupt_0x14_function_0x03:_Status request**

Interrupt 0x14 service 0x00: Set baud rate

Set the baud rate of the serial port

Parameters

AH

0x00

AL

Configuration parameter.
Bits 7--5: Baud rate, 4--3: Parity, 2: Stop bits, 1--0 Word length.

Bits 7--5: Baud rate
- 000 19200
- 001 38400
- 010 300
- 011 600
- 100 1200
- 101 2400
- 110 4800
- 111 9600

Bits 4--3: Parity
- 00 None
- 01 Odd
- 11 Even

Bit 2: Stop bits
- 0: 1 Stop bit
- 1: 2 Stop bits (available only when no parity is set)

Bits 1--0: Word length
- 1 0: 7 bits

1 1: 8 bits

DX
Port number: 0 for EXT, 1 for COM

Return Value

AH: Status code (see service **0x03**)

Comments

For higher baud rates use service **0x81** "Extended line control initialization"
Two stop bits are only available if no parity is set.

Top of list
Index page

Interrupt 0x14 service 0x01: Put byte in output buffer, wait if needed.

Character is queued for transmission. If there is space in the transmitter buffer when this call is made, the character will be stored and control returned to caller. If the buffer is full, the driver will wait for space. (This can be dangerous when used in combination with flow control.)

Parameters

AH
0x01

AL
Byte to be written

DX
Port number: 0 for EXT, 1 for COM

Return Value

AH: Status code (see service **0x03**)

Top of list
Index page

Interrupt 0x14 service 0x02: Get a byte from the serial port, wait if none available.

Reads a byte from the receiver buffer. Wait for a byte to arrive if none is available.

Parameters

AH
0x02

DX
Port number: 0 for EXT, 1 for COM

Return Value

AL: The byte received

Top of list
Index page

Interrupt 0x14 service 0x03: Status request

Return the status of the serial port.

Parameters

AH
0x03

DX
Port number: 0 for EXT, 1 for COM

Return Value

AH: Status code (bit field):
bit 6: Set if output buffer is empty.
bit 5: Set if output buffer is not full.
bit 4: Line break detected
bit 3: Framing error detected
bit 2: Parity error detected
bit 1: Set if overrun occurred on receiver.
bit 0: Set if data is available in receiver buffer.

Comments

Any reported UART error flags are cleared by hardware after the read is made for this call.

Top of list
Index page

Interrupt 0x14 service 0x04: Initialize fossil driver

Initialize the fossil driver for specified port.

Parameters

AH
0x04

DX
Port specifier: 0 for EXT, 1 for COM

Return Value

AX: 0x1954 if success

Comments

Use this function to detect if the fossil driver is available for this port. The user must make sure that only one process opens a port. If this port is used for standard **input** or **output** (console), then stdin/stdout will be disabled for this port.

Developer Notes

If the DMA mode (send or receive mode) is enabled in **chip.ini**, the following port settings are not allowed:
1. 8N2
2. 8S2
3. 8M2

Top of list
Index page

Interrupt 0x14 service 0x05: Deinitialize fossil driver

Deinitialize the fossil driver for specified port.

Parameters

AH
0x05

DX
Port specifier: 0 for EXT, 1 for COM

Return Value

none

Top of list
Index page

Interrupt 0x14 service 0x08: Flush output buffer waiting until done.

Wait for all output in the output buffer to be transmitted.

Parameters

AH

0x08

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Top of list
Index page

Interrupt 0x14 service 0x09: Purge output buffer.

Remove all data from the output buffer.

Parameters

AH
0x09

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Top of list
Index page

Interrupt 0x14 service 0x0A: Purge receive buffer.

Remove all data from the receive buffer.

Parameters

AH
0x0A

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Top of list
Index page

Interrupt 0x14 service 0x0B: Transmit byte, do not wait.

Place a byte into the transmit buffer if there is space available. Otherwise simply return with AX=0, without handling the transmit byte.

Parameters

AH
0x0B

AL
Byte to transmit

DX
Port number: 0 for EXT, 1 for COM

Return Value

AX=0 if byte was not accepted (no space in buffer)
AX=1 if byte was placed in buffer

Top of list
Index page

Interrupt 0x14 service 0x0C: Peek if next byte is available.

Returns the next byte available in the receive buffer, without removing it from the buffer.

Parameters

AH
0x0C

DX
Port number: 0 for EXT, 1 for COM

Return Value

AX=0xFFFF if no byte was available
AH=0x00 and AL=next byte, if a byte was available.

Top of list
Index page

Interrupt 0x14 service 0x0F: Enable/disable flow control.

Configure the flow control for a port.

Parameters

AH

0x0F

AL

Bit mask describing requested flow control.

DX

Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

Bit fields for FOSSIL data flow control:

B0: XON/XOFF on transmit (watch for XOFF while sending)
B1: CTS/RTS (CTS on transmit/RTS on receive)
B2: reserved
B3: XON/XOFF on receive (send XOFF when buffer near full)
B4-B7: Ignored

Notes:

- XON/XOFF and CTS/RTS are not allowed at the same time.

Developer Notes

XON/XOFF mode is also available if the **DMA** mode for the serial port is enabled but because of the internal functionality of DMA it is not possible to detect an XON or XOFF of the peer immediately. It is possible that an overrun situation at the connected peer (e.g. GSM modem) could occur. We now provide this mode because GSM modems (any??) support only XON/XOFF flow ctrl.

Top of list
Index page

Interrupt 0x14 service 0x18: Read block of data

Read up to a specified number of bytes from a serial port.

Parameters

AH

0x18

CX

Maximum number of bytes to transfer.

DX

Port number: 0 for EXT, 1 for COM

ES:DI
Pointer to user buffer.

Return Value

AX= Number of bytes transferred.

Top of list
Index page

Interrupt 0x14 service 0x19: Write a block of data

Write a block of data to the serial port output buffer.

Parameters

AH
0x19

CX
Maximum number of bytes to transfer.

DX
Port number: 0 for EXT, 1 for COM

ES:DI
Pointer to user buffer.

Return Value

AX= Number of bytes actually transferred.

Top of list
Index page

Interrupt 0x14 service 0x1B: Get driver info

Get information about a serial port and driver

Parameters

AH
0x1B

CX
Size of buffer

DX
Port number: 0 for EXT, 1 for COM

ES:DI
Pointer to user buffer.

Return Value

AX=Number of bytes actually transferred.

Comments

Offset 0 (word) = Structure size
Offset 2 (byte) = FOSSIL spec version (not used)
Offset 3 (byte) = Driver rev level (not used)
Offset 4 (dword) = Pointer to ASCII ID (not used)
Offset 8 (word) = Input buffer size
Offset 0A (word) = Bytes available (input)
Offset 0C (word) = Output buffer size
Offset 0E (word) = Bytes available (output)
Offset 10 (byte) = Screen width, chars (not used)
Offset 11 (byte) = Screen height, chars (not used)
Offset 12 (byte) = Baud rate mask (not used)

This function was provided for compatibility with older Fossil applications.

Top of list
Index page

Interrupt 0x14 service 0x1E: Extended set baud rate

Set the baud rate of the serial port

Parameters

AH
0x1E

BH
Parity
00h None
01h Odd
02h Even
03h Mark
04h Space

BL
Stop bits
00h: 1 Stop bit
01h: 2 Stop bits (available only when no parity is set)

CH
Word length
02h: 7 bits
03h: 8 bits

CL
Baud rate

00h: 110
01h: 150
02h: 300
03h: 600
04h: 1200
05h: 2400
06h: 4800
07h: 9600
08h: 19200
80h: 28800
81h: 38400
82h: 57600
83h: 76800
84h: 115200

DX
Port number: 0 for EXT, 1 for COM

Return Value

AH: Status code (see service **0x03**)

Comments

Two stop bits are only available if no parity is set.

Top of list
Index page

Interrupt 0x14 service 0x80: Enable/Disable RS485 mode

Enable the RS485 mode.

Parameters

AH
0x80

AL
0=TxEnable low active
1=TxEnable high active
2=Disable RS485 mode

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

By default the RTS0 and RTS1 signals (pins) are used to enable/disable the respective (EXT or COM)

transmitter. (TxEnable)
Note that RS485 is not available with **serial send DMA**!

Top of list
Index page

Interrupt 0x14 service 0x81: Extended line control initialization

Extended line control initialization.

Parameters

AH
0x81

AL
UART character data bits
2: 7 bits
3: 8 bits

BH
Parity
0: no parity
1: odd parity
2: even parity
3: mark parity (always 1)
4: space parity (always 0)

BL
Stop bits
0: 1 Stop bit
1: 2 Stop bits (only available when no parity is selected)

CX
Baud rate divider
(for maximum baud rate see HAL function **0x8A**)

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

Two stop bits are only available if no parity is set.

Developer Notes

Parity settings "Mark" and "Space", and two stop bits are not checked on received data by the @Chip (UART) or the API. This is due to these modes are not available in hardware. These modes are provided to communicate

with hardware that can operate only in these modes.

If the DMA mode (send or receive mode) is enabled in **chip.ini**, the following port settings are not allowed:
1. 8N2
2. 8S2
3. 8M2

Top of list
Index page

Interrupt 0x14 service 0x82: Select RS485 pin

Select the RS485 TxEnable pin

Parameters

AH
0x82

AL
No of PIO pin [0..13]

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

By default the RTS0 and RTS1 signals (pins) are used to enable/disable the respective transmitter. (TxEnable) This function lets you select any PIO from 0-13 as TxEnable, but not all make sense. To change the default, call this function before you call the RS485 **Enable** function.

Top of list
Index page

Interrupt 0x14 service 0x83: Send break

Send long or short break

Parameters

AH
0x83

AL
1: long break (2,5 frames)

2: short break (1 frame)
3: extra long break (3 frames)

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

A short break is a continuous Low on the TXD output for a duration of more than one frame transmission time `M`, where:

```
M = startbit + data bits (+ parity bit) + stop bit
```

A long break is a continuous Low on the TXD output for a duration of more than two frame transmission times plus the transmission time for three additional bits (2M+3).

A extra long break is a continuous Low on the TXD output for a duration of more than three frame transmission times.

Top of list
Index page

Interrupt 0x14 service 0x84: Enable/disable UART receiver

Enable/Disable UART receiver

Parameters

AH
0x84

AL
0: disable receiver 1: enable receiver

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

This function is useful when using the SM35 as RS485 adapter. Use this function to disable the receiver before transmitting data. Wait until all data is sent and then reenable the receiver. This prevents receiving your own transmitted data.

By default the receiver is enabled.

Top of list
Index page

Interrupt 0x14 service 0x85: Enable/disable UART transmitter

Enable/Disable UART transmitter

Parameters

AH
0x85

AL
0: disable transmitter 1: enable transmitter

DX
Port number: 0 for EXT, 1 for COM

Return Value

none

Comments

By default the transmitter is enabled.

Top of list
Index page

Interrupt 0x14 service 0xA0: Get number of Tx bytes in UART

Returns the number of bytes which are currently in the UART transmitter hardware.

Parameters

AH
0xA0

DX
Port number: 0 for EXT, 1 for COM

Return Value

AX= Number of bytes in the UART transmitter hardware

Comments

With this function you can check how many bytes are currently in the UART. This could be necessary to know if your communication pauses because of a handshake problem.

The GetDriverInfo API (**0x1B**) reports the number of bytes remaining in the serial port driver's software

send queue.

This function reports the number of bytes currently in the UART transmit hardware. By adding this function's return value to the software transmit buffer byte count reported by the GetDriverInfo API, you can determine the total number of transmit data bytes still pending output.

The maximum count returned here will be 2 bytes, accounting for the UART's transmit shift register (1 byte) and transmit holding register (1 byte).

Top of list
Index page

Interrupt 0x14 service 0xA1: Install a Fossil User Callback Function

The user callback function will be called when a Fossil serial port event occurs.

Parameters

AH
0xA1

DX
Port number: 0 for EXT, 1 for COM

ES:DI
Pointer to the User Callback function

Comments

This call installs an user callback function for the specified serial port. The callback function must be very short! Long Fossil callback functions can lead to character loss.

To use the Fossil callback functions, you have to switch the serial port into the IRQ Mode (see **CHIP.INI**). In DMA mode the callback functions will not work!

The callback function must conform to this prototype:

```
fossil_event_t far *(huge my_fossil_callback)(
                        fossil_event_t far *e ) ;
```

The `fossil_event_t` structure e passed in contains the `event` which has occurred. Following events are possible:

```
#define FE_DATA_AVAIL 0x01     // New Data Received
#define FE_READY_FOR_SEND 0x02 // (currently not supported)
#define FE_ERROR_DETECTED 0x10 // (currently not supported)
```

Newly received `data_length` bytes can be found at location referenced by `data` pointer in structure e.

The callback function can also return a `fossil_event_t` structure. The returned structure contains an action `event` which the Fossil serial port driver will respond to as follows:

```
#define FE_IGNORE_DATA 0x01 // Do not copy the received data
                            // into the receive queue.
#define FE_DATA_TO_SEND 0x02 // (currently not supported)
```

If the callback does not return any event, the return value must be a null pointer (=0L).

The definition of the `fossil_event_t` structure is as follows:

```
typedef struct
{
    int size;      // size of the structure
    int port;      // serial port (0=EXT, 1=COM)
    int event;     // event (see above)
    void far *data;      // data pointer
    unsigned data_length; // data length
} fossil_event_t;
```

The `size` element should be set to `sizeof(fossil_event_t)`.

Top of list
Index page

End of document

Introduction

Adding a user specific device driver/linklayer interface to the TCP/IP stack

The @CHIP-RTOS of the IPC@CHIP provides four internal TCP/IP device interfaces:

1. Ethernet controller
2. PPP server
3. PPP client (PPP uses one of the serial ports of the IPC@CHIP)
4. Internal loopback (Virtual loopback device with IP address 127.0.0.1)

For each of these internal devices the necessary specific driver functions are implemented inside of the @CHIP-RTOS.

The provided **TCP/IP API calls 0xA0 - 0xA7** allow the application developer to install an additional TCP/IP driver interface for a connected hardware device (e.g. an additional UART or an Ethernet controller). This new interface then has its own IP configuration and is used by the TCP/IP stack for IP communication in the same way as with the pre-installed internal devices.

The following sections explain how to implement and install a user specific device driver for TCP/IP. The generic example code shown here uses C-library functions provided for the TCP/IP API, which can be found in source file `TCPIP.C`. We are also using several functions from the C-library files `HWAPI.C` and `RTOS.C`. The C-library files are available at **www.beck-ipc.com** in the Internet download area of the IPC@CHIP. All needed TCP/IP related types and constants are declared in the C-library header files `TCPIPAPI.H` and `TCPIP.H`.

Important notes:

1. IP configuration of the user device is not adjustable with the settings in the **IP section** of chip.ini configuration file. You can create your own section in chip.ini for storing IP configuration of the new device interface with **BIOSINT 0xA0 functions 0x23/0x24**. For possible IP configuration via the @CHIP-RTOS UDP config server see **Install UDP Config Server Callback**.

2. Setting the default gateway (reachable via the installed interface) is now possible with the expanded **ADD_DEFAULT_GATEWAY** API.

In the following sections we describe the set of driver specific functions you may have to provide. Possible implementations of an interrupt service function and receiver task for the device interface are provided. These device driver functions must be installed with the API call **DEVOPENIFACE**. These a callback functions which are invoked by the internal TCP/IP stack of the @CHIP-RTOS. ***Do not call these functions directly from within your application!***

The driver functions are internally locked by semaphores and block every other device driver function. Because of this behavior, it's not advisable to wait (sleep) for long periods of time as this can lead to deadlock situations (primarily between the send and recv calls).

- **Device Open function**
- **Device Close function**
- **Device Send function**

- Device Receive function
- Device FreeReceive function
- Device Get PhysicalAddress function
- Implementation of an interrupt service routine (ISR) and receiver task
- Install the device driver

Device Open function

The TCP/IP stack calls this (optional) function to initialize the hardware and (optional) to install an Interrupt Service handler. If Borland C compilers are used the driver functions must be declared as `huge` (see below). Microsoft C users must declare driver functions as `far _saveregs _loadds` .

The function should return 0 if initialization was successful. If initialization failed, return -1.

Generic Example:

```
int huge myDevOpen(DevUserIfaceHandle ifaceHandle)
{
    // Install (if necessary) a RTOS Interrupt Service function with
    //  HWAPI handler function 0xA1 service 0x84
    return 0;
}
```

Top of list
Index page

Device Close function

The TCP/IP stack will execute this (optional) function when the device driver interface is closed with the DEV_CLOSE_IFACE API call.

This callback function should return 0 if the closing of the device was successful, otherwise -1 on failure.

Generic example:

```
int huge myDevClose(DevUserIfaceHandle ifaceHandle)
{
   // DeInitialize the hardware
   // Remove the ISR handler
   return 0;
}
```

Top of list
Index page

Device Send function

This callback function is used by the TCP/IP stack to send the data out the device. The TCP/IP stack does not call this function from within a separate transmit task. This callback executes in the thread which made the send call, e.g. API_SEND API.

This callback function should return 0 if the sending of data was successful, otherwise -1 on error.

Important:

If the input parameter `flag`has value 1 (this indicates this is last frame in block) you must call DEV_SND_COMPLETEto tell the TCP/IP stack that the send buffer is no longer in use.

Generic example:

```
int huge myDevSend(DevUserIfaceHandle ifaceHandle,
                   unsigned char far * dataPtr,
                   int dataLength,
                   int flag )
 {
     int errorcode;

     // Hardware specific: Send the data (dataPtr) out the device

     // Do not wait(sleep) here for indefinite times,
     //  to avoid blocking of other function calls.

     // Is this now the last frame in message block?
     if (flag & 0x1)    // Bit0 flag set?
     {
         // Inform TCP/IP stack that transmit buffer is now free.
         Dev_Send_Complete(ifaceHandle, &errorcode);  // C-Lib
     }
     return 0;
 }
```

Related Topics

DEV_SND_COMPLETE API - Dev_Send_Complete's implementation
DevUserIfaceHandle type definition

Top of list
Index page

Device Receive function

In this function, a received packet is passed back up into the protocol stack. The TCP/IP stack calls this function to receive a data frame from the device. TCP/IP calls this function from within your separate receiver task, which you are required to create (see final **example**).

The receive callback should return 0 if receiving of data was successful, otherwise -1 on failure.

Important :

It's optional but recommended to store incoming data in a buffer from the TCP/IP pre-allocated memory pool (see **TCPIPMEM**). API call **DEV_GET_BUF**returns you a buffer pointer for storing the incoming data (see example below).

If you are using your own buffer allocation for storing the incoming data, you must null out the location referenced by the input parameter `bufferHandle` (see example). In this case, you should also implement and install the device driver **callback** function:

```
int (far * DevFreeRecvFunc)(DevUserIfaceHandle ifaceHandle,
                    unsigned char far * dataPtr);
```

The TCP/IP stack calls this function to indicate that the receive buffer is no longer used by TCP/IP. The vector to this callback is placed in the `DevFreeRecv` member of the **DevUserDriver** structure at the **DEV_OPEN_IFACE** call.

Two generic examples for receiver functions follow:

myDevReceive1 : Using buffer from the TCP/IP memory pool
myDevReceive2 : Using your own receive buffer

```
// Generic example using TCP/IP memory pool receive buffer:

int huge myDevReceive1(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle)
{
    int                 errorCode;
    unsigned int        rcvdLength;
    unsigned char far  *tcp_buffer ;

    // Hardware specific: Check how many incoming data bytes are available.
    rcvdLength = .....;   // =byte count

    // Get a buffer from TCP/IP by calling API service 0xA5 and save at dataPtr
    Dev_Get_Buffer(bufferHandle, dataPtr, rcvdLength);  // C-Lib call
    tcp_buffer = *dataPtr ;       // Check if memory allocation successful
    if (tcp_buffer != (unsigned char far *)0)
    {
       // Hardware specific: Move received data from device to tcp_buffer
       // Do not wait (sleep) here for indefinite times,
       //   to avoid blocking of other function calls.

       *dataLength = recvdLength;   // Report number of bytes now in tcp_buffer
       return 0;  // success
    }
    else
    {
       return -1; // out of memory
    }
}


 // Generic example for using your own receive buffer:

int huge myDevReceive2(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle)
{
    // Save the pointer to the beginning of the data
    *dataPtr = myBuffer;  // myBuffer somehow allocated by the user

    // Hardware specific: Read data from your device and store in myBuffer

    // Save the length (in bytes) of received data
    *dataLength = deviceDataLength;

    // IMPORTANT: Null out the bufferhandle pointer
    *bufferHandle = (DevUserBuffer)0;
    return 0;
}
```

Related Topics

DEV_GET_BUF API - Dev_Get_Buffer's implementation
DevUserIfaceHandle type definition

Top of list
Index page

Device FreeReceive function

Implementation of this function is necessary if you decide to use your own buffers for receiving incoming data.

The TCP/IP stack will call this function to inform you that the receive buffer (input parameter `dataPtr`) is no longer used by TCP/IP.

This callback should return 0 if ok, else -1 on failure.

Generic example:

```
int huge myDevFreeRecv(DevUserIfaceHandle ifaceHandle, unsigned char far *dataPtr )
{
     // Somehow free your allocated buffer at dataPtr
     my_free(dataPtr);

     return 0;
}
```

Related Topics

DevUserIfaceHandle type definition

Top of list
Index page

Device Get PhysicalAddress function

This function applies only to Ethernet controllers.

The 6 byte array referenced by the `PhysicalAddress` input parameter should be filled with the MAC address of your connected Ethernet controller.

Generic example:

```
int huge myDevGetPhysAddr(DevUserIfaceHandle ifaceHandle,
                          unsigned char far * physicalAddress)
{
    // Hardware specific: copy MAC address into physicalAddress
    _fmemcpy(physicalAddress, myEthernet_MAC, 6) ;
    return 0;
}
```

Related Topics

DevUserIfaceHandle type definition

Top of list
Index page

Implementation of an interrupt service routine (ISR) and receiver task

The implementation of a device specific ISR is optional. If your hardware device is able to generate interrupts on device events (e.g. incoming data available), you can implement an ISR like the example below. The CPU time spent within an ISR must be keep to a minimum, as the length of this interrupts masked period impacts the interrupt latency of the other critical system ISR's. Consequently, your ISR should only notify events (incoming data, error,..) at the device and not directly handle device events itself (e.g. retrieve incoming data) immediately

within the ISR.

With API call **DEV_NOTIFY_ISR** the ISR should wakeup a user provided task, which receives the incoming data from the device and moves the data into the TCP/IP stack. This task should use API call **DEV_RECV_WAIT** and **DEV_RECV_IFACE** (see example below).

Instead of a creating a new task, it is also possible to use your program's main thread for receiving by having it perform the `MyReceiveTask()` actions shown below.

If your device doesn't support interrupts, you could create a polling task (or again, simply use your program's main thread for this purpose) which periodically checks your device for incoming data as illustrated in the `MyReceiveTask_Polling` example below.

Important : An ISR must be installed as a RTOS ISR with the **Install Interrupt Service Routine** of the Hardware API.

Generic examples for an ISR and a two forms of receiver task functions follow.

```
// Interrupt Service Routine

void interrupt MyDeviceISRHandler(void)
{
   int receivedFrames;
   int errorCode;

   // Hardware specific: Check if there are incoming data packets available

   // Wakeup receiver task
   Dev_Notify_ISR(MyDevHandle, receivedFrames, 0, &errorCode);  // C-Lib call

   // Note: Issue no EOI here.
   //   (EOI for the ISR is issued inside of the CHIP-RTOS.)
}

 // Generic example for a receiver task function, which waits for an event from ISR:

void huge MyReceiveTask(void)
{
    int errorCode;
    int statRecv;
    // Optional: do some initialization

    while(1)
    {
       // Wait for a wakeup from ISR
       Dev_Recv_Wait(mydevdriver.IfaceHandle, &errorCode);   // C-Lib call

       // After wakeup received and move incoming data into the stack
       do
       {              // C-Lib call
          statRecv = Dev_Recv_Interface(mydevdriver.IfaceHandle, &errorCode);
       } while (statRecv != -1);
    }
}


 // Generic example for receiver task, polling for incoming data:

void huge MyReceiveTask_Polling(void)
{
    int errorCode;

    // Optional: do some initialization

    // Wait for completion of interface installation (Intr 0xAC 0xA0)
    while (install_done == 0)
    {
       RTX_Sleep_Time(10); // Go to sleep for a defined time.
```

```
    }

    while(1)
    {
       // Check if there is data available inside of your device.
       if (myDeviceDataAvail())
       {
           // Receive and move incoming data into the stack
           Dev_Recv_Interface(mydevdriver.IfaceHandle, &errorCode);
       }
       RTX_Sleep_Time(10); // Go to sleep for a defined time.
    }
}
```

Related Topics

DEV_NOTIFY_ISR API - Dev_Notify_ISR's implementation
DEV_RECV_WAIT API - Dev_Recv_Wait's implementation
DEV_RECV_IFACE API - Dev_Recv_Interface's implementation
RTX_SLEEP_TIME API - RTX_Sleep_Time's implementation

Top of list
Index page

Install the device driver

Based on the previous sections of this document, the following generic example should make clear the main steps required to install a user implemented device driver:

```
#include "tcpip.h"
#include "rtos.h"

int huge myDevOpen(DevUserIfaceHandle ifaceHandle);

int huge myDevClose(DevUserIfaceHandle ifaceHandle);

int huge myDevSend(DevUserIfaceHandle ifaceHandle,
                   unsigned char far * dataPtr,
                   int dataLength, int flag);

int huge myDevReceive1(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle);

int huge myDevGetPhysAddr(DevUserIfaceHandle ifaceHandle,
                          unsigned char far * physicalAddress);

void interrupt MyDeviceISRHandler(void);

void huge MyReceiveTask(void);

unsigned int       recvID;              // task ID
unsigned int       myrecv_stack[1024];  // stack for receiver task
unsigned char      install_done = 0;    // waiting flag  for receiver task
unsigned int       errorCode;
DevUserIfaceHandle MyDevHandle;

TaskDefBlock  myrecv_defblock =
{
    MyReceiveTask,                 // task function
    {'D','E','V',' '},             // a name: 4 chars
```

```
    &myrecv_stack[1024],          // top of stack
    1024*sizeof(int),             // size of stack
    0,                            // attributes, not supported
    20,                           // priority 20(high) ... 127(low)
    0,                            // no time slicing
    0,0,0,0                       // mailbox depth,
};

char far * mydevicename  = "MyDev";
char far * IPString      = "192.168.200.020";
char far * NetmaskString = "255.255.255.000";

DevUserDriver mydevdriver;

int main(void)
{
    //*********************************************************************
    // Initialize struct mydevdriver;
    //*********************************************************************
    mydevdriver.DevName = mydevicename;            // Unique device name,
    //  max. 13 chars + 0.

    inet_addr(IPString     , &mydevdriver.IpAddr);  // IP address
    inet_addr(NetmaskString, &mydevdriver.Netmask); // Netmask

    mydevdriver.iface_type = 1;                    // Ethernet device
    mydevdriver.use_dhcp  =  0;                    // no DHCP

    //Important:
    // At the first DEV_OPEN_IFACE call for a device, IfaceHandle must be NULL.
    mydevdriver.IfaceHandle = 0 ;

    // Note: If the interface should be restarted by calling DEV_CLOSE_IFACE
    //  and DEV_OPEN_IFACE (e.g. for changing IP configuration) the
    //  IfaceHandle must contain at DEV_OPEN_IFACE the valid IfaceHandle handle
    //  from the first DEV_OPEN_IFACE call.

    // Install your driver functions
    mydevdriver.DevOpen        = (void far *)mydevOpen;
    mydevdriver.DevOpen        = (void far *)mydevClose;
    mydevdriver.DevSend        = (void far *)mydevSend;
    mydevdriver.DevRecv        = (void far *)myDevReceive1;
    mydevdriver.DevFreeRecv    = (void far *)0;  // Since using TCP/IP buffers
    mydevdriver.DevGetPhysAddr = (void far *)myDevGetPhysAddr;
    mydevdriver.DevIoctl       = (void far *)0;  // Currently not supported,
                                                 //   pass a Null pointer.

    //*********************************************************************
    // Install the device driver interface
    //*********************************************************************
    result = Dev_Open_Interface(&mydevdriver, &errorCode);  // C-Lib

    // if(result).....

    //*********************************************************************
    // Create the receiver task
    //*********************************************************************
    result = RTX_Create_Task(&recvID, &myrecv_defblock);   // C-Lib

    // if(result).....

    // Optional, but recommended: Change priority of receiver task to high prio 4
    RTX_Change_TaskPrio(recvID, 4, &errorCode);

    //*********************************************************************
    //If device interface should be configured by DHCP, wait for completion
    //of the DHCP configuration process
    //*********************************************************************
    if (mydevdriver.use_dhcp == 1)
```

```
    {
        result= Dev_Wait_DHCP_Complete(&mydevdriver, 20, &errorCode); // C-Lib
        // if(result)....
    }
    //  Wait forever, or stay resident with int21h call 31h.
    //  It is possible to close and restart the interface inside of an application.
    //  But due to the internal architecture of the TCP/IP stack, it is not
    //  possible to exit the driver program and restart the interface with the
    //  same unique name.
    //  Your driver program should run forever.  Avoid killing the receiver task.

    while(1) RTX_Sleep_Time(100);

}// End of main(void)
```

Related Topics

API_INETADDR API - inet_addr's implementation
DEV_OPEN_IFACE API - Dev_Open_Interface's implementation
DEV_WAIT_DHCP_COMPLETE API - Dev_Wait_DHCP_Complete's implementation
RTX_TASK_CREATE API - RTX_Create_Task's implementation
RTX_SLEEP_TIME API - RTX_Sleep_Time's implementation
DevUserDriver data structure type definition

Top of list
Index page

End of document

PPP Interface - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

Introduction

Here is a short description of how to configure the IPC@CHIP PPP server.

The PPP server is available starting with @CHIP-RTOS version SC12V0067PPP. (There also exists a @CHIP-RTOS version SC12V0067 without the PPP server.) The PPP client API is available starting with @CHIP-RTOS 070.

The PPP client and server API calls are part of the TCP/IP API. These API calls are described in the **TCP/IP API** documentation. Only configuration of the PPP server is described here.

Topics:

About PPP

Since SC12 @CHIP-RTOS version 067 Beta a PPP server is available in the SC12.
PPP (the Point to Point Protocol) is a mechanism for creating and running TCP/IP over a serial link - be that a direct serial connection (using a null-modem cable), or a link made using an analogue modem.

Other computers can dial into the IPC@CHIP PPP server and communicate via the TCP/IP link using FTP, Telnet, Web, etc. in the same manner as with an Ethernet TCP/IP link.
One major difference between a PPP and an Ethernet connection is of course the speed.
A standard Ethernet connection operates at 10 Mbs maximum theoretical throughput, whereas an analogue modem operates at speeds up to 56 kbps.

PPP is strictly a peer to peer protocol; there is no difference between the machine that dials in and the machine that is dialed into. However, it is still useful to think in terms of servers and clients. When you dial into a site to establish a PPP connection, you are a considered the client. The machine to which you connect (e.g. the IPC@CHIP) is considered the server.

PPP on the IPC@CHIP includes the subprotocols LCP and NCP(IPCP).
Supported authentication protocols are PAP and CHAP.

Top of list

Configuring the PPP Server

The PPP server must be configured using the `chip.ini` file sections [PPPSERVER] and [SERIAL]. All entries for configuring the PPP server are listed in the **config.htm** file.

Here is an outline of the steps required to configure the PPP server.

1. Disable/**enable PPP server**
   ENABLE=0 or ENABLE=1
   By default, PPP server is enabled.

2. Select a serial port for the PPP server to use with the **COMPORT**directive.

   Important :
   The COM and EXT port of the IPC@CHIP has only CTS,RTS,RxD and TxD lines, so you have to configure your modem with DTR always on.
   (e.g. AT cmd for a ZyXEL modem: AT&D0)

3. Increase the **send** and **receive** queue sizes of the chosen serial port (EXT or **COM**).
   Recommended size is 4096 Bytes.

   Example `chip.ini` settings for the EXT port:

   ```
   [SERIAL]
   EXT_RECVQUEUE=4096
   EXT_SENDQUEUE=4096
   ```

4. Select the **flow control mode** for the PPP server serial port.

5. Set the baud rate of PPP server's serial port e.g. **BAUD** =19200 (default is 38400)

6. Select usage of an analogue modem: **MODEM**=1.
   The default is 0 (null modem cable).

7. Set the **IP Address** of the PPP server interface e.g. `ADDRESS=192.168.206.4`

8. Set the **REMOTE IP Address** for the connected host.

   There are three different possibilities for configuring PPP server IP addresses on the IPC@CHIP:

   a) If valid addresses for **IP Address**and **REMOTE IP Address**are declared in chip.ini, the PPP server wants to use this configured IP for its own and wants to configure the remote peer with the defined remote address.

   b) If only **IP Address**is declared in chip.ini and **REMOTE IP Address**is set to 0.0.0.0, the PPP server wants to use this configured IP address and expects the client to use their own address.

   c) If both entries **IP Address**and **REMOTE IP Address**are set to 0.0.0.0, the PPP server expects an IP address from the peer.

9. Define **net mask** and router **default gateway**, e.g.
   `NETMASK=255.255.255.0`
   `GATEWAY=192.168.206.4`

   Note about IP FORWARDING:
   Since @CHIP-RTOS version 067 the SC12 has two network interfaces, Ethernet and PPP, so the IPC@CHIP can forward IP packets between these interfaces. If you define a

gateway in the PPPSERVER section of the `chip.ini`for the PPP server interface, it becomes the default gateway for all interfaces when a PPP link to the server is established. During a PPP server connection the command **`ipcfg`** indicates this default gateway. After the PPP session, the old gateway (if any) for the Ethernet interface will be restored. As of @CHIP-RTOS version 070, the TCP/IP API supports adding and deleting a default gateway:

10. Choose **authentication mode** with the AUTH directive.

11. Initialize the analogue modem.

    You can define three sets of modem initialization parameters. These parameters are used to initialize the modem at the start of the IPC@CHIP @CHIP-RTOS and after a modem hang-up following a PPP session.

    Each of the three parameter sets consists of the following four parameters:
    - **INITCMD** string - Command sent to the modem to initialize it.
    - **INITANSWER** string - Expected modem response to initialization command.
    - **INITTIMEOUT** integer - Number of seconds to wait on answer from modem.
    - **INITRETRIES** integer - Number of times to repeat modem initialization sequence if a previous attempt fails.

    Example:

    ```
    INITCMD0=ATZ
    INITANSWER0=OK
    INITTIMEOUT0=2
    INITRETRIES0=3

    INITCMD1=AT&D0
    INITANSWER1=OK
    INITTIMEOUT1=2
    INITRETRIES1=3

    INITCMD2=AT
    INITANSWER2=OK
    INITTIMEOUT2=2
    INITRETRIES2=2
    ```

    A timeout value 0 means wait forever for the modem's answer.

    If you enter the string NULL at an INITANSWER (e.g. INITANSWER0=NULL), the IPC@CHIP PPP server will not wait for an answer from the modem.

12. Define a maximum of three modem commands for getting **connected** to the remote peer.
    Example:

    ```
    CONNECTMSG0=RING
    CONNECTANSWER0=ATA
    CONNECTTIMEOUT0=0
    CONNECTMSG1=CONNECT
    CONNECTTIMEOUT1=60
    ```

    These are the default values for modem connect commands.
    In this example the PPP server waits forever for the RING message and sends an ATA to the modem if it responds to the RING. After that the server waits a maximum of 60 seconds for a response to the CONNECT message. The modem link is established. The server now establishes the PPP connection to the remote client.

    Note:
    Do not use the AT command ATS0=1. This will cause the modem to automatically answer

the call without waiting for the PPP server. This is too fast for the PPP server.

13. Hang-up the connection.

    The PPP server will attempt to hang-up the modem when either a connection is closed by a remote peer, or if the modem initialization failed during the IPC@CHIP boot process.

    `. Switch the modem into the command mode (**CMDMODE** and **HANGUPDELAY**)

    Example:

    ```
    CMDMODE=+++
    HANGUPDELAY=2
    ```

    These are the default values.

    a. Define modem commands and expected answers for **hang-up**.
    Again, up to three sets of these parameters can be given here.

    Example:

    ```
    HANGUPCMD0=ATH0
    HANGUPANSWER0=OK
    HANGUPTIMEOUT0=2
    HANGUPRETRIES0=1
    ```

    If you enter the string NULL at an HANGUPANSWERx (e.g. HANGUPANSWER0=NULL), the IPC@CHIP PPP
    server will not wait for an answer from the modem.

14. Control the online state while PPP session is open.

    You can define three sets of modem control parameters. These parameters are used to check the online state of the modem at an open PPP connection.

    `. Enable online control sequence **MODEMCTRL**=1.
    The default is 0 (disabled).

    Example:

    ```
    MODEMCTRL=1
    ```

    Also you have to configure a control time interval (in seconds). After each time interval during which the PPP server receive no data, the server executes the configured modem commands. The server closes the connection, if one of the expected answers timed out.

    a. Define a **CTRLTIME**.
    E.g. `CTRLTIME=120` default is 60 seconds

    Each of the three parameter sets consists of the following four parameters:
    - **CTRLCMD** string - Command sent to the modem to initialize it.
    - **CTRLANSWER** string - Expected modem response to control command.
    - **CTRLTIMEOUT** integer - Number of seconds to wait on answer from modem.
    - **CTRLRETRIES** integer - Number of times to repeat modem control sequence if a previous attempt fails.

    Example and default settings:

    ```
    CTRLCMD0=+++
    CTRLANSWER0=OK
    ```

```
CTRLTIMEOUT0=3
CTRLRETRIES0=1

CTRLCMD1=AT0
CTRLANSWER1=NULL
CTRLTIMEOUT1=1
CTRLRETRIES1=0
```

If you enter the string NULL at an CTRLANSWERx (e.g. CTRLANSWER0=NULL), the IPC@CHIP PPP server will not wait for an answer from the modem.

15. Define a **time out** value in seconds after which the PPP server hangs up the connection if no data comes in from client during this timeout period.

    E.g. `IDLETIME=160` default is 120

Top of list
Index page

PPP Server API

The TCP/IP API provides five calls that apply to the PPP server.

1. Interrupt 0xAC, **Service 0x50**: Check if the PPP server is installed.
2. Interrupt 0xAC, **Service 0x51**: Suspend PPP server task
3. Interrupt 0xAC, **Service 0x52**: Activate PPP server
4. Interrupt 0xAC, **Service 0x53**: Get current state of the PPP server
5. Interrupt 0xAC, **Service 0x54**: Get the PPP server configuration
6. Interrupt 0xAC, **Service 0x55**: Set the PPP negotiate options

Top of list
Index page

PPP Client API

The TCP/IP API provides four calls that apply to the PPP client.

1. Interrupt 0xAC, **Service 0x40**: Check if the PPP client is installed.
2. Interrupt 0xAC, **Service 0x41**: Open a connection to PPP server
3. Interrupt 0xAC, **Service 0x42**: Close connection
4. Interrupt 0xAC, **Service 0x43**: Get current state of the PPP client
5. Interrupt 0xAC, **Service 0x44**: Get PPP primary and secondary DNS IP addresses
6. Interrupt 0xAC, **Service 0x45**: Set the PPP negotiate options

Top of list
Index page

Available Examples

The following example code is available:

- PPP server API test, PPPS.C
- PPP client example, PPPCLIE.C

Top of list
Index page

End of document

Web server CGI interface - SC12 @CHIP-RTOS V1.10

CGI API

The CGI ("Common Gateway Interface") API uses interrupt 0xAB with a service number in the high order byte of the AX register (AH). This interface provides access to the CGI implementation of the IPC@CHIP Web server. CGI at the IPC@CHIP provides the possibility to install at the IPC@CHIP Web server own programmed CGI functions. These functions are bound with a fixed name and are executed by the Web server task, if a http request with such a fixed name comes in. This mechanism allows dynamic usage of the IPC@CHIP Web server. For better understanding of CGI and possibilities for using, see description of available program examples at the example link below.

File Upload:
Also there is a possibility to upload files to the @CHIP's disk. This does not use the CGI interface but the Web server. Therefore you only need a defined HTML formular which sends the required data to the @CHIP Web server. An example can be found in the CGI Example Zip file.

Notes:

1. For test and demonstration purpose two pages are preinstalled at the IPC@CHIP:
   a)**main.htm:** Static html introduction page.
   b)**chipcfg** **:** Dynamic page with system time/date and configuration data of the requested IPC@CHIP.
2. For configuring the webserver and CGI, see the provided **chip.ini entries**.
3. For some useful comments see also under **Programming notes**

Topics

Web Server **Overview**
Web Server **File**Types

CGI API **News**
CGI API **Error Codes**
CGI API **Developer**Notes
CGI API **Examples**Available
CGI API **Data Structures**

API Functions Available

- **Interrupt_0xAB_function_0x01:_CGI_INSTALL, Install a CGI function**
- **Interrupt_0xAB_function_0x02:_CGI_REMOVE, Remove a CGI function**
- **Interrupt_0xAB_function_0x03:_CGI_SETMAIN, Set a new main page**
- **Interrupt_0xAB_function_0x04:_CGI_SETROOTDIR, Set Web server's root directory**
- **Interrupt_0xAB_function_0x05:_CGI_GETROOTDIR, Get Web server's root directory**
- **Interrupt_0xAB_function_0x06:_CGI_GETMAIN, Get main page name**

- Interrupt_0xAB_function_0x07:_CGI_GETFORMITEM, Split a formular into name and value
- Interrupt_0xAB_function_0x08:_CGI_FINDNEXTITEM, Return the address of the next formular tag
- Interrupt_0xAB_function_0x09:_CGI_INSTALL_PAS, Install a Turbo Pascal CGI procedure

---

Interrupt 0xAB service 0x01: CGI_INSTALL, Install a CGI function

Parameters

AH
0x01 (= CGI_INSTALL)

DX:SI
Pointer to a temporary `CGI_Entry` **type** structure.

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains error code CGI_NO_FREE_ENTRY or CGI_INVALID_METHOD

Comments

This API function makes no copy of the information in the provided `CGI_Entry` structure, so your structure at [DX:SI] must be persistent. The maximum number of available CGI pages is configurable at the chip.ini file, see **CGI entries**.

Related Topics

CGI API **Error** Codes
`cgistat` **command** line
CGI_INSTALL_PAS API Function, for Pascal CGI Procedures

Top of list
Index page

Interrupt 0xAB service 0x02: CGI_REMOVE, Remove a CGI function

Parameters

AH
0x02 (= CGI_REMOVE)

DX:SI
Pointer to the null terminated URL path name

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains error code CGI_NOT_FOUND

Comments

The CGI function to be deleted is identified by the provided URL string. It is also possible to remove the two predefined cgi functions **main.htm** and **chipcfg** from the table.

Related Topics

CGI API **Error** Codes
`cgistat` **command** line

Top of list
Index page

Interrupt 0xAB service 0x03: CGI_SETMAIN, Set a new main page

Parameters

AH
0x03 (= CGI_SETMAIN)

DX:SI
Pointer to name of new main page

Return Value

DX =0 success AX: 0
DX!=0 failure AX: error code CGI_INVALID_NAME

Comments

The string at [DX:SI] is null terminated with a maximum length of 64 characters (not counting the terminating zero).

Related Topics

CGI_GETMAIN API Function
CGI API **Error** Codes

Top of list
Index page

Interrupt 0xAB service 0x04: CGI_SETROOTDIR, Set Web server's root directory

Parameters

AH
0x04 (= CGI_SETROOTDIR)

DX:SI
Pointer to the name of new root directory

Return Value

DX =0 success AX: 0
DX!=0 failure AX: error code CGI_INVALID_DIR

Comments

The string at [DX:SI] is null terminated with a maximum length of 64 characters (not counting the terminating zero). The root directory is also configurable at the chip.ini file, see **ROOTDIR** and **DRIVE**.

Related Topics

CGI_GETROOTDIR API Function
CGI API **Error** Codes

Top of list
Index page

Interrupt 0xAB service 0x05: CGI_GETROOTDIR, Get Web server's root directory

Parameters

AH
0x05 (= CGI_GETROOTDIR)

Return Value

DX=0 AX=0 , ES:DI contains pointer to root directory name

Comments

The string referenced by [ES:DI] is null terminated and is in the RTOS's data space.

Related Topics

CGI_SETROOTDIR API Function

Top of list
Index page

Interrupt 0xAB service 0x06: CGI_GETMAIN, Get main page name

Parameters

AH
0x06 (= CGI_GETMAIN)

Return Value

DX=0 AX=0 , ES:DI contains pointer to current main page name

Comments

The string referenced by [ES:DI] is null terminated and is in the RTOS's data space.

Related Topics

CGI_SETMAIN API Function

Top of list
Index page

Interrupt 0xAB service 0x07: CGI_GETFORMITEM, Split a formular into name and value

Parse the argument buffer to obtain name and value.

Parameters

AH
0x07 (= CGI_GETFORMITEM)

BX:SI
Pointer to argument buffer to be parsed.

ES:DI
Pointer to a `FormItem` **type** structure

Return Value

DX=0 AX=0, User buffers referenced by pointers in `FormItem` structure at [ES:DI] are filled in with name and value

Comments

On initial call, the argument buffer pointer provided by the caller in BX:SI is a copy of the `fArgumentBufferPtr` member of the `rbCgi` **structure** passed by the Web server to the CGI callback function. On subsequent calls here to pick up additional formular, the pointer returned from the `CGI_FINDNEXTITEM` API call can be used here.

The caller must set the two members of the `FormItem` **structure** prior to calling here. Both pointers reference buffers allocated by the user, which will receive strings produced by this API call.

For preventing internal buffer overruns, the user should provide 64 bytes of buffer addressed by `FormItem.NamePtr` and 256 bytes pointed by `FormItem.ValuePtr`. The max. size for a full pathname (pagename+complete argumentbuffer)
e.g. `"example?Name1=Value1&Name2=Value2"`
is 256 bytes. Because of this behaviour, the storage addressed by `FormItem.ValuePtr` must have the max. size of a full pathname (256 bytes).
See **example** submit.c.

Related Topics

CGI_FINDNEXTITEM API Function

Top of list
Index page

Interrupt 0xAB service 0x08: CGI_FINDNEXTITEM, Return the address of the next formular tag

Most formulars have more than one item, this function searches for the next form item in a CGI request argument string. This function can only be used after a CGIFORMITEM API call. (See example submit2.c)

Parameters

AH
0x08 (= CGI_FINDNEXTITEM)

BX:SI
CGI request argument pointer

Return Value

DX=0 AX=0, ES:DI: pointer to the found item
DX=-1 AX=0, no next item was found

Comments

The CGI request argument buffer pointer provided by the caller in BX:SI is initially taken from the `rbCgi` structure passed by the Web server to the CGI callback function.

This function scans the buffer at [BX:SI] for an ampersand character, '&', and if found returns a pointer to the character in the string following the ampersand.

The strings must be null terminated.

Related Topics

CGI_GETFORMITEM API Function

Top of list
Index page

Interrupt 0xAB service 0x09: CGI_INSTALL_PAS, Install a Turbo Pascal CGI procedure

Special install function for Turbo Pascal CGI procedures

Parameters

AH

0x09 (= CGI_INSTALL_PAS)

DX:SI
Pointer to a temporary `CGI_Entry` **type** structure.

Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains error code CGI_NO_FREE_ENTRY or CGI_INVALID_METHOD

Comments

This API function makes a copy of the information in the provided `CGI_Entry` structure, so your structure at [DX:SI] need not be persistent.

Related Topics

CGI_INSTALL API Function, for C CGI Procedures

Top of list
Index page

End of document

CGI API Updates - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

CGI API News

The following extensions to the CGI **API** are available in the indicated @CHIP-RTOS revisions.

No changes since last version.

End of document

Web Server Overview - SC12 @CHIP-RTOS V1.10

Web Server Overview

The explanations here assume the reader is somewhat familiar with the HTTP protocol.

Web Server / CGI topics:

- **CGI_API Functions**
- **CGI_in the IPC@CHIP**
- **Built-In_CGI Functions**
- **Building_a CGI function using a C compiler**
- **File_Upload**

CGI API Functions

The CGI **API**uses interrupt 0xAB, enabling the application programmer to install their own DOS program CGI functions in the IPC@CHIP Web server's CGI table. This makes it possible to visualize and control an application running in the IPC@CHIP via an Internet browser, using all features of modern Internet technology.

Top of list
Index page

CGI in the IPC@CHIP

The Web server of the IPC@CHIP uses an internal CGI table. The CGI table is an array of `CGI_Entry`**type**structures. Defined for each entry is the URL name, the expected HTTP **method**(Get, Head or Post) and a pointer to a **function**which will be executed if a matching browser request arrives at the IPC@CHIP.

Top of list
Index page

Built-In CGI Functions

There are two pre-installed CGI functions in the IPC@CHIP, one of them named "ChipCfg". The preset entry in the internal CGI table has the name "ChipCfg" **(Note: Since SC12 @CHIP-RTOS V1.10 Beta, SC13**

@CHIP V0.92 Beta, URL names of CGI are no longer case sensitive!). A CGI callback function for this entry supports the method "Get" by producing a Web "file" in RAM. This function is executed if a browser makes the following request to an IPC@CHIP device with an IP address of `192.168.205.4`:

```
http://192.168.205.4/ChipCfg
```

The CGI function then produces a HTML page in memory which contains specific configuration data, like IP address, subnet mask, serial number, etc..

Top of list
Index page

Building a CGI function using a C compiler

A CGI function built with the Borland C compilers must be declared as:

```
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

Using the Microsoft Visual C compilers the CGI functions are declared as:

```
void far _saveregs _loadds _pascal
            CGI_Func(rpCgi far *CgiRequest);
```

The Web server calls this function with the address of a `rpCgi`**type**data structure, which contains all needed information about the browser request. Another part of this structure is the response fields. These fields must be set by the CGI function. They hold information needed by the Web server.
Important: The max. length of a web page in RAM produced by a users CGI function is 65519 characters.

Top of list
Index page

File Upload

It is possible to upload a file to the Web server. This feature is not available with the standard @CHIP-RTOS variants (Tiny, Small, Medium, Large, ..). If you want a special @CHIP-RTOS variant with this feature, you have to request it from us via email.

The Upload will be handled using a HTML Form. The form has to include the following field:

```
DESTINATION-PATH (e.g. "A:\HELLO.EXE")
REDIRECT-PATH (e.g. "A:\WEB\SUCCESS.HTM")
FILE-CONTENT
```

The DESTINATION-PATH defines the name and the location where the file will be stored on the IPC@CHIP's drive. REDIRECT-PATH links to a file, which will be returned on a successful upload to the Browser. The file must be located on an IPC@CHIP's drive. The value of the field FILE-CONTENT will be handled from your Web Browser automatically. It allows the user to choose a local file from the PC's drive using the browse button (which appears with the type "file").

Password and user name for the File Upload are defined in the **CHIP.INI**(standard: web, web). For example your can use a form like the following one to upload a file to the IPC@CHIP's Web server:

```
<html>
<head>
<title>FileUpLoad to the IPC@CHIP Web Server</title>
</head>
<body>
<form action="http://192.168.201.4" enctype="multipart/form-data" method="POST">
<p>
  Choose file from your PC for Upload:<br>
  Destination Filename:
         <input type="text" name="DESTINATION-PATH" value=""><br>
  Linkto:
         <input type="text" name="REDIRECT-PATH" value=""><br>
  Local File Path:
         <input name="FILE-CONTENT" type="file" size="50" ><br>

         <input type="submit" value=" Absenden ">
</p>
</form>
</body>
</html>

Comment:
You can also use the Type "hidden" if you want to hard code some value.
e.g: <input type="hidden" name="REDIRECT-PATH" value="success.htm">
```

Top of list
Index page

End of document

Data Structures

Here are the data structures used by the CGI **API**.
All constants and data structures are defined in the header file cgi.h

Notes:

- **Byte alignment is required** for all data structures used within the API.

Contents :

---

typedef CGI_Entry

```
typedef struct tag_cgi_table
{
    char         *PathPtr;     // Name of the page, URL
    int           method;      // http method: Get, Head  or Post
    RpCgiFuncPtr  CgiFuncPtr;  // ptr to callback function for this page

} CGI_Entry;
```

Comments

The members of the `CGI_Entry` structure should be filled as follows.

`PathPtr`
This is a far pointer to a zero terminated URL address of the page. (URL's are case sensitive.)

`method`
This `int`is an **enumerator**used to specify what HTTP method this CGI function supports.

`CgiFuncPtr`
This is a far vector to the CGI function for this Web page. For Borland C compilers the `RpCgiFuncPtr`type is:

```
typedef void (huge _pascal *RpCgiFuncPtr)
            (rpCgi far *CgiRequestPtr);
```

... and therefore the CGI function itself is declared as ...

```
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

For Microsoft Visual C compilers the `RpCgiFuncPtr`type is:

```
typedef void far _saveregs _loadds _pascal
    (*RpCgiFuncPtr)(rpCgi far *CgiRequestPtr);
```

... and the CGI function declared as ...

```
void far _saveregs _loadds _pascal
        CGI_Func(rpCgi far *CgiRequest);
```

Related Topics

API function **CGI_INSTALL** - Install a CGI function
`rpCgi` **structure** type

Top of list
Index page

---

define CGI HTTP Request

```
#define CgiHttpGet  1       //  Cgi request is HTTP GET
#define CgiHttpHead 2       //  Cgi request is HTTP HEAD
#define CgiHttpPost 3       //  Cgi request is HTTP POST
```

Comments

These defines are used as enumeration names for request methods.

Related Topics

method member of CGI_Entry type

Top of list
Index page

---

typedef FormItem

```
typedef struct tag_form_item
{
    char   *NamePtr;  //Buffer, pointed by NamePtr should have 64 bytes length
    char   *ValuePtr; //Buffer, pointed by ValuePtr should have 256 bytes length
} FormItem;
```

Comments

Both strings referenced here are null terminated.

Related Topics

API function **CGI_GETFORMITEM** - Split a formular into name and value

Top of list
Index page

---

typedef rpCgi

```
typedef struct {
    //*****************************************************
    //    Request fields  (Read Only!!!!)
    //*****************************************************
    unsigned char  fConnectionId;          // -- internal use only --
    int            fHttpRequest;           // get, post, head
    char          *fPathPtr;               // URL
    char          *fHostPtr;               // Host:
    char          *fRefererPtr;            // (at time not supported)
    char          *fAgentPtr;              // (at time not supported)
    char          *fLanguagePtr;           // (at time not supported)
    unsigned long  fBrowserDate;           // Date:   (internal)
    char          *fArgumentBufferPtr;     // Pointer to argument buf
    long           fArgumentBufferLength;  // Length of argument buf
    char          *fUserNamePtr;           // Username from Authorization
    char          *fPasswordPtr;           // Password from Authorization
    long          *fRemoteIPPtr;           // new at V1.00 Beta, points to the remoteIP,
                                           // you must split the octets
                                           // For using the IP to etsablish TCP/IP
                                           // connections, you have to exchange
                                           // lowbyte and highbyte!

    //*****************************************************
    //    Response fields  (Set by CGI function)
    //*****************************************************
    int            fResponseState;         // -- internal, do not modify --
    int            fHttpResponse;          // Response httpmsg e.g. CgiHttpOk
    int            fDataType;              // Content type, e.g. text/HTML, text/plain
    char          *fResponseBufferPtr;     // Pointer to the created page
    long           fResponseBufferLength;  // Length of the page (max. length is 65519 chars)
    unsigned long  fObjectDate;            // -- internal, do not modify --
    unsigned int   fHostIndex;             // -- internal, do not modify --

} rpCgi, *rpCgiPtr;
```

Comments

fDataType
This is an enumeration type that specifies the **file**type.

Related Topics

CGI Callback **Function**

Top of list
Index page

End of document

CGI File Types - SC12 @CHIP-RTOS V1.10

CGI File Extensions

Here is a mapping between values for the `fDatatype` member of the `rpCGI` **type** and file extensions supported by the IPC@CHIP Web server.

fDatatype Value	File Extension	Web Server File Type
0	*.htm	text/HTML
1	*.gif	image/gif
3	*.txt	text/plain
4	*.jpg	image/jpeg
5	*.pct	image/pict
6	*.tif	image/tiff
10	*.css	text/css
11	*.xml	text/xml
11	*.xsl	text/xml
12	*.wav	audio/wav
13	*.pdf	application/pdf
14	*.jar	application/java-archive
16	*.wml	text/vnd.wap.wml
17	*.wmp	image/vnd.wap.wbmp
18	*.wmc	application/vnd.wap.wmlc
19	*.wms	text/vnd.wap.wmlscript
20	*.wmx	text/vnd.wap.wmlscriptc
21	*.svg	image/svg+xml
7		image/png
8		application/x-www-form-urlencoded
9		application/ipp
15		application/octet-stream

The default type is application/octet-stream (`fDatatype` = 15)

Top of list
Index page

End of document

IPC@CHIP Documentation **Index**

CGI API Error Codes

All error codes listed here are defined in the header file, cgi.h.

CGI error codes returned by CGI **API** calls in the **DX-Register**

```
 0 = CGI_ENOERROR          -> success
-1 = CGI_ERROR             -> error, AX contains error code
-2 = CGI_NOT_SUPPORTED     -> invalid function number was in the AH-reg.
```

CGI specific error codes returned by CGI API calls in the **AX register** when DX register was non-zero (error indication):

```
-1 = CGI_INVALID_METHOD      -> invalid method
-2 = CGI_INVALID_NAME        -> invalid URL name
-3 = CGI_INVALID_DIR         -> invalid directory
-4 = CGI_NO_FREE_ENTRY       -> no space in CGI table
-5 = CGI_NOT_FOUND           -> entry not found
```

End of document

CGI Application Developers Note - SC12 @CHIP-RTOS V1.10

Developer Notes

Since @CHIP-RTOS version 0.65, we added five new content **types** for CGI.

Since @CHIP-RTOS version 0.65, it is possible to write CGI procedures with Turbo Pascal. Consequently the parameter passing mechanism for CGI functions has been changed to that used by Pascal. C programmers will now need to use the following forms of CGI functions.

A CGI function using the Borland C compilers must be declared as follows:

```
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

... and using the Microsoft Visual C compilers:

```
void far _saveregs _loadds _pascal
        CGI_Func(rpCgi far *CgiRequest);
```

CGI **API** Listing

End of document

CGI Examples Available - SC12 @CHIP-RTOS V1.10

CGI Examples

Available CGI **API** **examples** in C:

1. `minicgi.c` - **Builds** an HTML page
2. `countcgi.c` - Builds a **dynamic** HTML page
3. `dk40cgi.c` - Reads DK40 I/O **pins**
4. `dk40_set.c` - Reads and **writes** DK40 I/O pins
5. `secure.c` - Example of **password** protected page
6. `submit.c` - Building of **formular**
7. `submit2.c` - Formulars with **multiple** items

Also there are source files (cgiapi.c, cgiapi.h) containing wrapper functions which provide a C-Library interface to the assembly language/software interrupt based CGI **API**.

Available Turbo **Pascal** examples:

1. `cgimini.pas` - **Builds** an HTML page
2. `countcgi.pas` - Builds a **dynamic** HTML page

Examples in C

For understanding CGI in the IPC@CHIP we provide some example programs written in C. The examples are compiled with various versions of Borland and Microsoft compilers. Which compiler version was used for a particular example is stated within the example's source files. The include file `cgi.h` contains all the required type definitions and constants.

1. **minicgi.exe**
   The CGI function installed by this program produces a HTML page which contains some of the browser's request parameters. This program was tested and compiled with Borland C 3.0 and Microsoft Visual C 1.52 The compiler differences are described in the source files.

   Examples browser inputs:
   ```
   http://192.168.205.4/minicgi
   http://192.168.205.4/minicgi?Argument
   ```

2. **countcgi.exe**
   We build a dynamic HTML page which contains the current value of a counter incremented in the main loop of the program. This program is tested and compiled with Borland C 3.0 and Microsoft Visual C 1.52. The compilers differences are described in the source files.

   Example browser input:
   ```
   http://192.168.205.4/countcgi
   ```

3. **dk40cgi.exe**
   The CGI function of this program produces a HTML page which contains the current values of the DK40 I/O pins.

   Example browser input:
   ```
   http://192.168.205.4/dk40
   ```

4. **dk40_set.exe**
   Demonstrates set and reset control over the DK40 output pins via browser.

   Example browser input:
   ```
   http://192.168.205.4/dk40
   ```

5. **secure.exe**
   This is an example of a protected page.

   The first browser input ...
   ```
   http://192.168.205.4/dk40_secure
   ```
   requires the input of a valid user name and password, e.g.:
   ```
   User name: user
   Password: password
   ```

6. **submit.exe**
   The building of formulars is demonstrated here.
7. **submit2.exe**
   The building of formulars with more than one item is demonstrated here.

Important:

1. The recommended memory model for DOS programs is "Large".
2. CGI functions should be programmed as short as possible, without long or endless waits.
3. CGI functions compiled with Borland C must be declared as "huge".
4. CGI functions compiled with Microsoft C must be declared as " far _saveregs _loadds".
5. Users of Microsoft Visual C must set the compiler option "struct member byte alignment" to "1 byte" or must use "#pragma pack(1)" in their source.
6. The **command** `cgistat` at the IPC@CHIP command prompt lists all CGI functions installed.
7. URL names for CGI functions are case-sensitive.
8. If you use Microsoft C-Compilers then increase the Web server's **WEBSERVERSTACK** stack size value in the `chip.ini` file. The default stack size of the Web server task is 2048 Bytes. Programmers of CGI functions who are using Microsoft C-Compilers with C-Library functions, e.g. `sprintf`, which requires a lot of stack space should increase this allocation to 6144 (6 Kbytes). More stack space for the Web server task is also required if your CGI function uses a large amount of stack for automatic data (local variables) declared inside the CGI function call.

Building Turbo Pascal CGI procedures

Since @CHIP-RTOS version 0.65, it is possible to write CGI procedures with Turbo Pascal
This is a little bit different from writing a CGI function with the C compilers.
A Turbo Pascal program which contain a CGI procedure uses the same data structures (records)
as a C language CGI function.

Declaration of Turbo Pascal CGI procedures

A CGI procedure written with Turbo Pascal must be declared without any parameters and with the interrupt declaration, e.g.:

```
procedure CgiMini_Proc;interrupt;
```

The interrupt declaration will motivate the compiler to emit code that sets the CPU's DS data segment register on entry to the procedure, thus allowing access to the Pascal program's data.

The IPC@CHIP Web server handles the call to a Pascal CGI function differently than it does the call to a C language CGI function. One input parameter is needed by all CGI functions, C or Pascal. This one parameter is a far pointer to a `rpCgi` **type** structure (or in Pascal, a record). When calling a C language CGI function this pointer is pushed onto the stack using a normal parameter passing mechanism. For Pascal CGI functions this pointer is instead passed in the CPU's ES:DI registers. Consequently this pointer must be recovered by the Pascal program. This can be done as follows:

```
procedure CgiMini_Proc;interrupt;
    var
    ESreg    :    Integer;
    DIreg    :    Integer;
    CGIRequest    :    rpCGIptr;

    begin
        asm
        mov ax,es
        mov Esreg,ax
        mov ax,di
        mov DIreg,ax
    end;
    CGIRequest    :=    ptr(ESreg,DIreg);
    ............
end;
```

Installing a Pascal CGI procedure:

Pascal CGI procedure must be installed at the start of the DOS program with the **CGI_INSTALL_PAS** API call. (CGI functions written in C must still be installed with the standard **CGI_INSTALL** API.)

For better understanding of programming CGI with Turbo Pascal, we provide some example programs compiled with Borland Pascal 7.0.

1. **cgiMini.exe**
   The CGI function installed by this program produces a HTML page which contains some of the request parameters.

   Examples for browser inputs:
   ```
   http://192.168.205.4/cgimini
   http://192.168.205.4/cgimini?Argument
   ```

2. **countcgi.exe**
   We build a dynamic HTML page which contains the current value of a counter incremented in the main loop of the program.

   Browser input: `http://192.168.205.4/countcgi`

End of document

Ethernet Packet Driver Interface - SC12 @CHIP-RTOS V1.10

Ethernet: Packet Driver Interface

Packet driver interface definition for direct access to the IPC@CHIP Ethernet device.

Packet drivers provide a simple, common programming interface that allows multiple applications to share a network interface at the data link level. The packet driver demultiplex incoming packets among the applications by using the network media's standard packet type. The packet driver provides calls to initiate access to a specific packet type, to end access to it, to send a packet,and to get information about the interface. The implemented services are only a small subset of the **common packet driver interface**.

The IPC@CHIP RTOS uses interrupt 0xAE with a service number in the high order byte of the AX register (AH) to access the IPC@CHIP Ethernet packet driver. All supported services are listed here.

- **Interrupt_0xAE_function_0x01:_DRIVER_INFO, Get Driver Information**
- **Interrupt_0xAE_function_0x02:_ACCESS_TYPE, Install Access Handler**
- **Interrupt_0xAE_function_0x03:_RELEASE_TYPE, Unload an Access Handler**
- **Interrupt_0xAE_function_0x04:_SEND_PKT, Send an Ethernet packet**
- **Interrupt_0xAE_function_0x06:_GET_ADDRESS, Get the IPC@CHIP Ethernet address**
- **Interrupt_0xAE_function_0x14:_SET_RCV_MODE, Set receive mode**
- **Interrupt_0xAE_function_0x15:_GET_RCV_MODE, Get current receive mode**
- **Interrupt_0xAE_function_0x16:_SET_MULTICAST, Set Ethernet multicast address**
- **Interrupt_0xAE_function_0x27:_DEL_MULTICAST, Remove Ethernet multicast address**
- **Interrupt_0xAE_function_0x28:_INSTALL_WILDCARD, Install wildcard access handler**

On return the CPU carry flag is set if an error has occurred. The DH register holds an error code:

```
0x00    NO_ERROR
0x01    BAD_HANDLE        // Invalid handle number
0x05    BAD_TYPE          // Bad packet type specified
0x09    NO_SPACE          // Insufficient space
0x0A    TYPE_INUSE        // Type already in use
0x0B    BAD_COMMAND       // Command not supported
0x0C    CANT_SEND         // Packet couldn't be sent
                          // (hardware error?).
```

Important:

One important difference to other standard packet drivers is, that the software interrupt 0xAE does not have the typical PKTDRVR signature at offset 3 in the interrupt source code. So instead of looking for this signature, use BIOS interrupt 0xA0 **function**0x16 to check if the packet driver interface is available in the IPC@CHIP @CHIP-RTOS.

The maximum number of installed packet handlers is five.

Examples for the usage of the packet interface:

1. PKTDRV.C: C-Source for the API calls
2. PKTDRV.H: Defines, typedefs, prototypes, ... for the API calls
3. REC.C : Example for a receiver handler function, installed with function ACCESS_TYPE
4. SEND.EXE and RCV.EXE (with source): Simple program pair for sending and receiving of Ethernet packets

Interrupt 0xAE service 0x01: DRIVER_INFO, Get Driver Information

Added only for compatibility

Parameters

AH
0x01

Return Value

Carry flag: 0, Success:
AL: 1, Basic functions present
BX: 0x0B, Version
CH: 0x01, Class
CL: 0x00, Number
DL: 0x36, Type
DH: 0x00, Error code = NO_ERROR
DS:SI : Pointer to the null terminated driver name string, "SC1xPKT"

Related Topics

ACCESS_TYPE Install Access Handler

Top of list
Index page

Interrupt 0xAE service 0x02: ACCESS_TYPE, Install Access Handler

Parameters

AH
0x02

AL
class, from **DRIVER_INFO call**

CX
type length, must be 2!

DL
number, from **DRIVER_INFO call**

DS:SI
Pointer to the desired packet type, e.g. 0x800 for IP
(The object pointed to here must be persistent, not a momentary value.)

ES:DI
Vector to user's receiver handler function for this packet type

Return Value

Carry flag: 0 Success, AX contains the handle number (needed for **RELEASE_TYPE call**)
Carry flag :1 Failure, DH contains error code

Comments

We support the following Ethernet frame format (802.3):

48 Bits (6 Bytes) Destination address

48 Bits (6 Bytes) Source address
16 Bits (2 Bytes) Type field e.g 0x0608 for ARP, 0x0008 for IP or user defined types
46 to 1500 Bytes of user data.

The maximum number of installed handlers is limited to five. In all BIOS versions except the TINY version, setting of ARP or IP handlers is not allowed here.

When an Ethernet packet of the type specified here at [DS:SI] is received by the IPC@CHIP's network device driver, this driver will perform callbacks to the receiver handler function specified here in [ES:DI]. This callback will be done twice per accepted packet:

- First call:
  Input parameters to your handler: AX = 0, CX = received packet length
  Return Value from your handler: ES:DI - Pointer to buffer where driver can load the CX received bytes.
- Second call:
  Input parameters to your handler: AX = 1, data is now ready in your buffer
  Return Value from your handler: -- none --

On the first call, your handler produces a buffer into which the driver can transfer the received packet. This byte transfer occurs between the two calls to your handler function.

Important:

Because of our Little Endian processor you must exchange the bytes for the packet types e.g. use 0x0008 for the IP type instead of 0x0800.

Simple example of an receiver callback function:

```
void interrupt receiver (unsigned bp, unsigned di, unsigned si,
                                                unsigned ds, unsigned es, unsigned dx,
                                                unsigned cx, unsigned bx, unsigned ax,
                                                unsigned ip, unsigned cs, unsigned flags )
{
  if (ax == 0) // ax==0 -> Packet driver asks for a buffer, give it at es:di
  {
        es = FP_SEG(&receivepacket);
        di = FP_OFF(&receivepacket);
  }

  if (ax == 1) //Packet driver has filled our provided buffer
  {
        //Call our external c-function packet_receive to process the received data
        packet_received(cx);
  }
}
```

Related Topics

RELEASE_TYPE Unload an Access Handler
INSTALL_WILDCARD Install wildcard access handler

Top of list
Index page

Interrupt 0xAE service 0x03: RELEASE_TYPE, Unload an Access Handler

The user installed Ethernet packet type access handler is removed.

Parameters

AH

0x03

BX

Handle from **ACCESS_TYPE call** 0x02

Return Value

Carry: 0, AX:0, DX:0: success
Carry: 1, DH contains error code

Related Topics

ACCESS_TYPE Install Access Handler

Top of list
Index page

Interrupt 0xAE service 0x04: SEND_PKT, Send an Ethernet packet

Send bytes in provided packet buffer over Ethernet.

Parameters

AH
0x04

CX
Length of packet

DS:SI
Pointer to packet buffer

Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, DH contains error code

Comments

The data packet buffer should have the Ethernet packet form defined by IEEE 802.3:

```
typedef struct MY_PACKET_
{
unsigned char dest[6];     // Destination MAC address
unsigned char src[6];      // Source MAC address
unsigned int type;         // Packet type (big endian)
unsigned char data[DATA_SIZE];
} MY_PACKET ;
```

where `DATA_SIZE` can range up to 1500 bytes.

The `src` field can be set using the **GET_ADDRESS** API.

Related Topics

GET_ADDRESS Get the Ethernet address

Top of list
Index page

Interrupt 0xAE service 0x06: GET_ADDRESS, Get the IPC@CHIP Ethernet address

Get the IPC@CHIP Ethernet address.

Parameters

AH
0x06

ES:DI
Pointer to user buffer (6 bytes), for storing the Ethernet address

Return Value

Carry flag:0, AX:0, DX:0: success
Location at [ES:DI] contains the six bytes of the Ethernet address.

Top of list
Index page

Interrupt 0xAE service 0x14: SET_RCV_MODE, Set receive mode

Set receive mode of IPC@CHIP Ethernet device.

Parameters

AH
0x14

CX
Receive mode
3: Receiving packets with own Ethernet address or broadcast address
6: Promiscuous mode (receive all)

Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, AX:-1;DX:-1 Bad parameter in CX

Comments

Default receive mode is 3.
Receive mode 6 should only be used in the IPC@CHIP Tiny version (@Chip-RTOS without the TCP/IP stack).
Calling **SET_MULTICAST** overrides the current mode with mode 5.
API call **Get receive mode** returns the current mode.

Related Topics

GET_RCV_MODE Get receive mode

Top of list
Index page

Interrupt 0xAE service 0x15: GET_RCV_MODE, Get current receive mode

Get current receive mode of the Ethernet device.

Parameters

AH
0x15

Return Value

Carry flag:0, AX:3 accept any incoming packet with own address or broadcast address
AX:5 accept any incoming packet with own address, broadcast address or installed multicast addresses
AX:6 accept all incoming packets Carry flag:1, DH contains error code

Related Topics

SET_RCV_MODE Set receive mode
SET_MULTICAST Set Ethernet multicast address

Top of list
Index page

Interrupt 0xAE service 0x16: SET_MULTICAST, Set Ethernet multicast address

Set Ethernet multicast address.

Parameters

AH
0x16

ES:DI
Pointer to multicast MAC address buffer 6 bytes

CX
length of ES:DI buffer, must be 6

Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, DH contains error code

Related Topics

DEL_MULTICAST Remove Ethernet multicast address

Developer Notes

The address will be installed inside the Ethernet device of the IPC@CHIP. A maximum of 64 addresses can be installed.

Top of list
Index page

Interrupt 0xAE service 0x27: DEL_MULTICAST, Remove Ethernet multicast address

Remove the Ethernet multicast address.

Parameters

AH
0x27

ES:DI
Pointer to multicast MAC address buffer 6 bytes

CX
length of ES:DI buffer, must be 6

Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, DH contains error code

Related Topics

SET_MULTICAST Set Ethernet multicast address

Top of list
Index page

Interrupt 0xAE service 0x28: INSTALL_WILDCARD, Install wildcard access handler

Installs a wildcard packet type access handler.

Parameters

AH
0x28

AL
class, from DRIVER_INFO **call**

CX
type length, must be 2!!

DL
number, from DRIVER_INFO **call**

DS:SI
Pointer to the desired packet type, e.g. 0xFFFF for accepting any incoming Ethernet packet
(The object pointed to here must be persistent, not a momentary value.)

ES:DI
Vector to user's receiver handler function for this packet type

Return Value

Carry flag: 0 Success, AX contains the handle number
Carry flag :1 Failure, DH contains error code

Comments

Any incoming Ethernet packet will be accepted for which one or more bits of the packet type matches to the bits of the installed wildcard packet type. If a packet type of 0xFFFF is installed, any incoming packet will be accepted.

Only one wildcard can be installed at a time. Installing a new wildcard type overwrites the previous.

Delete a wild card handle by installing a wildcard with the packet type 0x0000. "Normal" handlers installed with **ACCESS_TYPE** API are not overwritten by installing a wildcard

Related Topics

ACCESS_TYPE Install Access Handler

Top of list
Index page

End of document

External Disk Drive

Here is the interface definition for an external disk B: drive . This interface allows you to add an external B: drive. This drive must be block (sector) oriented. Each sector should be 512 bytes long. The application must provide a software interrupt 0xB1 function to read and write these sectors on this drive.

Maximum disk size is about 2 Gigabytes.

- **Interrupt_0xB0_function_0x01:_Install External Disk**
- **Interrupt_0xB0_function_0x02:_Deinstall External Disk**

Interrupt 0xB0 service 0x01: Install External Disk

This function will logically install a B: drive.
Prior to this call the application must provide a disk read/write function **installed** at interrupt 0xB1. This installed handler function should expect:

```
AX       1 for write, 0 for read.
BX,DX    Sector number (BX is MSH of unsigned long)
CX       Number of sectors to read/write
ES:DI    Segment:Offset of memory area to read/write
```

The handler should return 0 in AX if OK.

Parameters

AH
0x01

BX,DX
An unsigned long with the total number of sectors. BX holds high word.

Return Value

AX is 0 if OK.

Top of list
Index page

Interrupt 0xB0 service 0x02: Deinstall External Disk

Used to deinstall drive B: which was installed before with function 0x01 (above).

Parameters

AH
0x02

Return Value

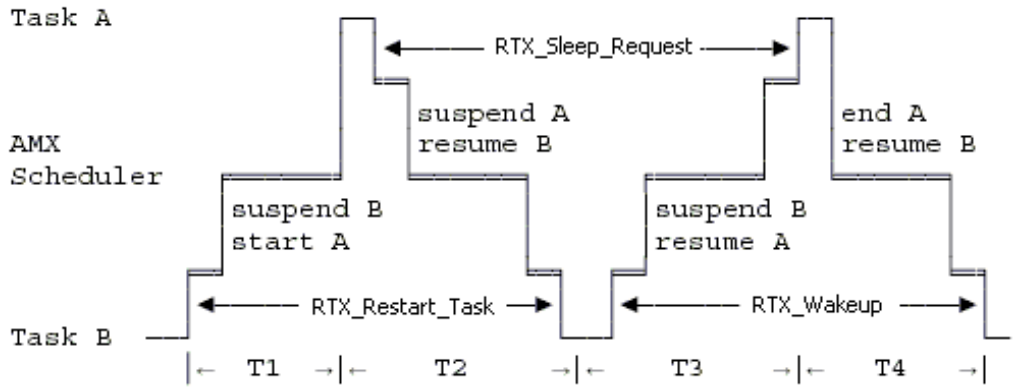
AX =0 success
AX !=0 failed (may be drive was not installed?)

Top of list
Index page

End of document

RTOS task switch time

The following diagramm and C code shows how the measured values were calculated.




```
void huge taskA(void)
{
   RTX_Sleep_Request();
}

void huge taskB(void)
{
   RTX_Restart_Task (taskAID);
   RTX_Wakeup (taskAID);
}

void main (void)
{
   //...
   result = RTX_Create_Task_Without_Run(&taskAID , &taskAdefblock);
   RTX_Sleep_Time(10);
   result = RTX_Create_Task(&taskBID , &taskBdefblock);
   //...
}
```

	T1	T2	T3	T4
SC12	104 µs	80 µs	100 µs	60 µs
SC13/SC11	35 µs	35 µs	36 µs	26 µs

Interrupt latency

The interrupt handler latency is the time from the processor's first response to an interrupt request signal through to the first useful instruction inside of the user interrupt service procedure.

The user interrupt service procedure can be of type HW API or RTX.

	HW API handler latency	RTX handler latency
SC12	84 µs	90 µs
SC13/SC11	21 µs	25 µs

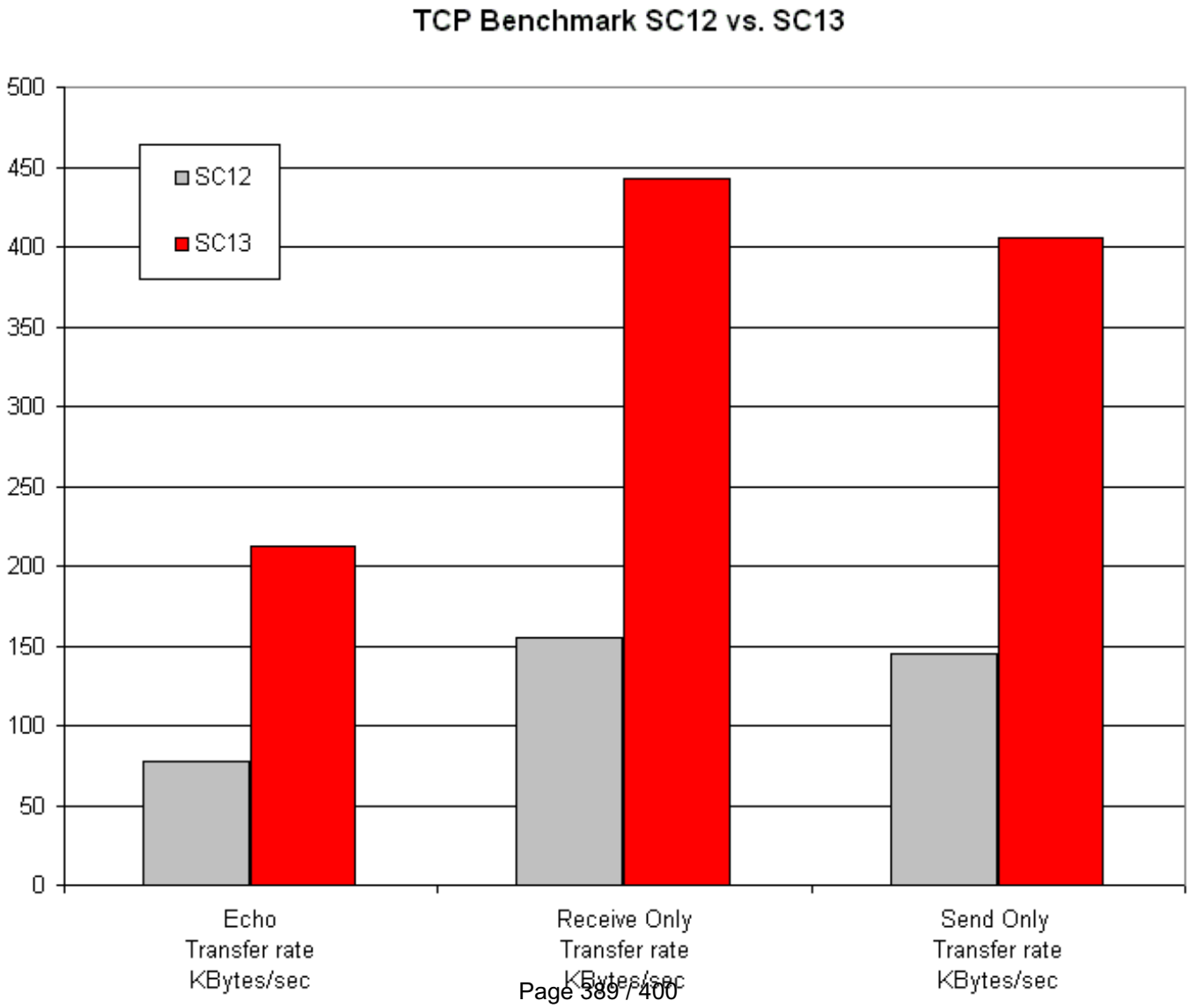
TCP echo

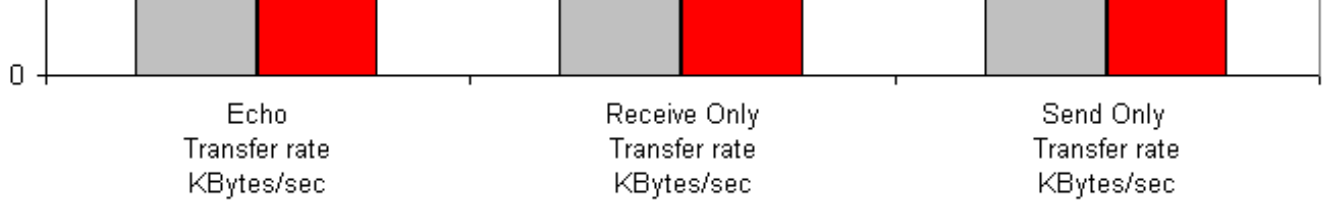
A TCP Echo client application, running on Win2000 (AMD Athlon PC, 1GHz) establishes a TCP connection to an IPC@CHIP (SC12 or SC13) TCP Echo server application. The TCP server echoes the incoming data back to the client. The client application measures the response time and calculates the amount of databytes in KBytes/sec, which the IPC@Chip is able to echo back to the client.
The tests were made with a datasize of 10 MBytes.

Echotest : Measure the throughput of sending and receiving echo data.
Receive only: Measure the throughput only of receiving data from IPC@CHIP.
Send only : Measures the throughput, if the IPC@CHIP only sends data.

This test was made under special test conditions. We cannot guarantee that the measured performance results are achievable on other test environment and conditions.

	Echo transfer rate KByte/s	**Receive only transfer rate KByte/s**	**Send only transfer rate KByte/s**
SC12 10 Mbit/s	78	155	145
SC13 100 Mbit/s	212	443	406

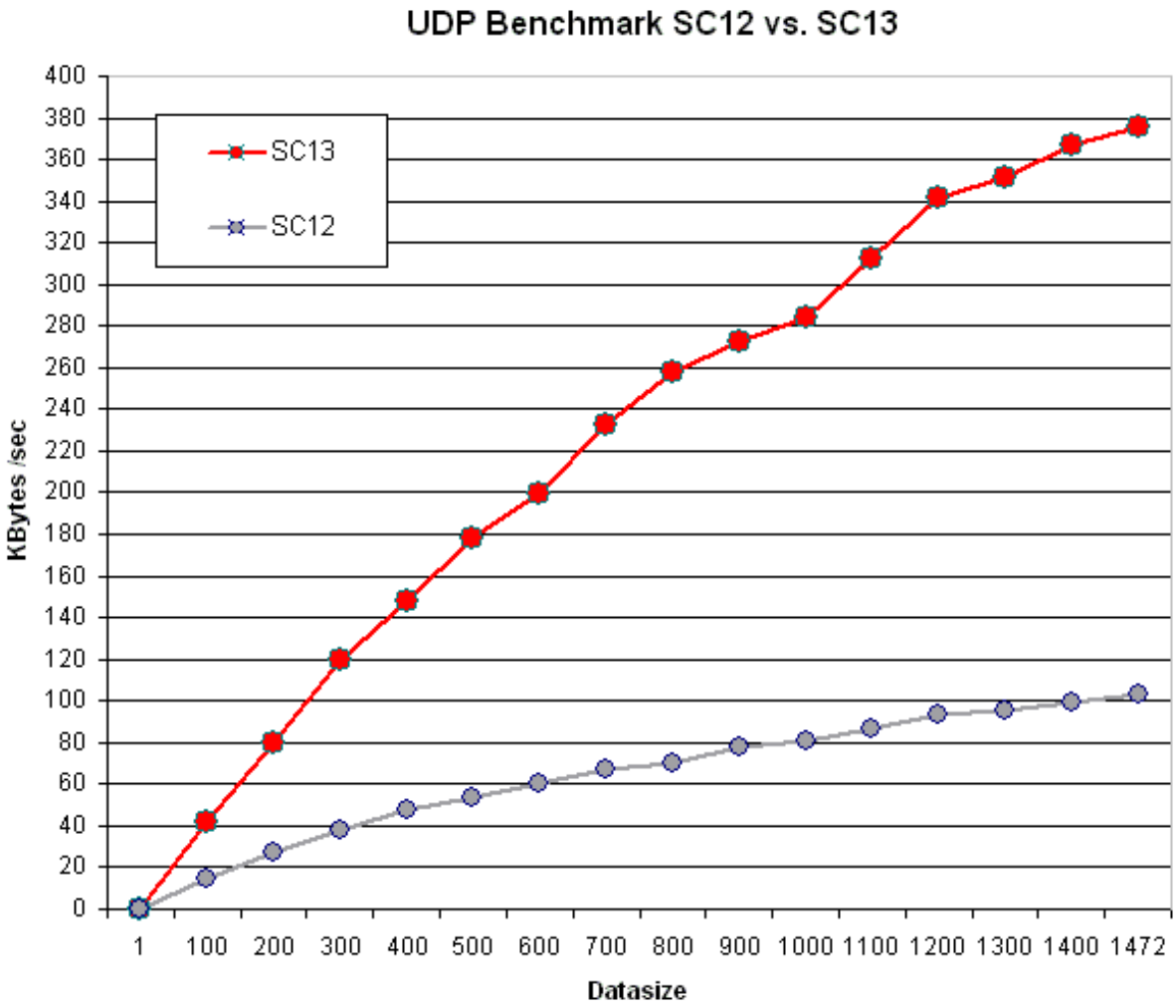




UDP echo

An UDP Echo client application, running on Win2000 (AMD Athlon PC, 1GHz) sends UDP datagrams of different data sizes to an IPC@CHIP (SC12 or SC13) UDP Echo server application. The UDP server echoes the incoming datagramms back to the client. The client application measures the response time and calculates the amount of databytes in KBytes/sec, which the IPC@Chip is able to echo back to the client in dependency of the datagram size. The communication peers (Win2000 PC and IPC@CHIP) are connected with a "Twisted pair crossover cable" to avoid disturbtion by other network traffic.

This test was made under special test conditions. We cannot guarantee that the measured performance results are achievable on other test environment and conditions.



End of document

Scalable @CHIP-RTOS variants for the SC12 IPC@CHIP

@CHIP-RTOS for the IPC@CHIP is provided by 6 official variants with a different set of included features. The variants with their provided features and the available RAM and Flash memory sizes are listed at the tables below. It is also possible to ask for special variants with other combinations of required @CHIP-RTOS features. Please contact **support@beck-ipc.com.**

1. SC12Vxxxx_TINY.hex
2 .SC12Vxxxx_SMALL.hex
3 .SC12Vxxxx_MEDIUM.hex
4. SC12Vxxxx_LARGE.hex
5 .SC12Vxxxx_MEDIUM_PPP.hex
6. SC12Vxxxx_LARGE_PPP.hex

	Tiny	Small	Medium	Medium PPP	Large	Large PPP
RTOS kernel	x	x	x	x	x	x
Serial port fossil driver	x	x	x	x	x	x
RTOS filesystem	x	x	x	x	x	x
Ext Disk			x	x	x	x
XMODEM filetransfer	x	x	x	x	x	x
TCP/IP Ethernet driver		x	x	x	x	x
TCP/IP PPP Client/server				x		x
Ethernet packet interface	x	x	x	x	x	x
TCP/IP protocol stack		x	x	x	x	x
I2C	x	x	x	x	x	x
Software SPI	x	x	x	x	x	x
Hardware API	x	x	x	x	x	x
CFG server		x	x	x	x	x
Webserver					x	x
FTP server			x	x	x	x
Telnet server			x	x	x	x

Please note:
The additional features of the @CHIP-RTOS (TFTP server, SNMP MIB support, Webfile upload,...) are not a part of the 6 official @CHIP-RTOS variants. Customers can ask for a special variant which includes the required @CHIP-RTOS features.
Each @CHIP-RTOS variant with TCPIP protocol stack includes a DHCP client. It is possible to order a @CHIP-RTOS variant without DHCP.

The IPC@Chip offers 512kB RAM and 512kB Flash disk.

Here are the sizes of available RAM and flash disk for the different variants of the current @CHIP-RTOS version

	Available RAM(kBytes)	Available Flash memory(kBytes)
Tiny	463	396
Small	364	287
Medium	358	256
Medium_PPP	354	209
Large	333	232
Large_PPP	329	186

End of document

Programmable I/O Pins

At turn-on the IPC@Chip I/O pins are configured as follows:

Pin1: RXD0/PIO7 = RXD0
Pin2: TXD0/PIO8 = TXD0
Pin3: CTS0/PIO9 = Input pullup
Pin4: RTS0/PIO10= Input pullup
Pin5: TXD1/PIO11= TXD1
Pin6: RXD1/PIO12= RXD1
Pin7: TMROUT0/INT0/PIO13 = Input pulldown
Pin17: RESET/PFAIL/LILED = Input
Pin24: ALE/PCS0 = Output, value 1
Pin25: CTS1/PCS2/PIO6/INT2 = Input pullup
Pin26: RTS1/PCS3/PIO5/INT4 = Input pullup
Pin27: PCS1/PIO4/TMRIN0/A0 = Input pullup
Pin28: PCS5/PIO3/TMROUT1/TMRIN1/A1= Input pullup
Pin29: PCS6/PIO2/A2= Input pullup
Pin30: I2CDAT/INT5/PIO1 = Input
Pin31: I2CCLK/INT6/PIO0 = Input

Back to main index page

TFTP server - SC12 @CHIP-RTOS V1.10

Introduction

Here is a short description of the IPC@CHIP TFTP server.

The TFTP server is available starting with @CHIP-RTOS version SC12V0100. It allows only sending and receiving of files to a remote TFTP client. (Thus the term *Trivial File Transfer Protocol* , TFPT.)

The TFTP server is not a part of our current six official @CHIP-RTOS versions. You must directly order a @CHIP-RTOS version with this feature.

For security reasons, TFTP file transfers are disabled by default. You can enable/disable TFTP with the TFTP 0/1 shell **command**. This shell command can be executed from within an application using the **Execute a shell command** API.

TFTP can be used as a simple alternative to FTP. TFTP does not provide extended file system features like listing directories or deleting files. If the user does not need these extended features of the FTP server, an IPC@CHIP @CHIP-RTOS version with TFTP instead of the FTP may be useful. This saves 13 KByte of flash memory.

By default the server listens at the standard TFTP port 69 for incoming TFTP client requests. The TFTP port can be **configured** in chip.ini.

The server is able to serve only one client at one time. Only the binary (octet) transfer mode is supported. The size of TFTP data packets must be 512 bytes.

End of document

Security notes - SC12 @CHIP-RTOS V1.10

IPC@CHIP Documentation **Index**

Introduction

Here some notes how to protect the IPC@CHIP against unauthorized access.
By default most of the further described security features are disabled.
This to enable every starting user to have access to them from the start
for his IPC@CHIP development.

- **WEB server**
- **TELNET server**
- **FTP server**
- **PPP server**
- **Chiptool UDP config server**
- **TFTP server**
- **General TCPIP network security**

WEB server

Steps to protect the IPC@CHIP against unauthorized access via HTTP:

1. **Webserver default drive**: Set a webserver root drive at chip.ini
2. **Webserver root directory**: Set a webserver root directory at chip.ini
3. **Remove CGI page ChipCfg**: Delete this preconfigured page from the CGI table with the CGI API
4. **PUT Method User and Password**: Define User and Password if you have a @CHIP-RTOS Variant which provides the HTTP PUT method. Otherwise everyone can transfer files to your server with the standard password and user 'WEB'. The HTTP PUT method wont be provided by the standard CHIP-RTOS Variants LARGE, MEDIUM, SMALL, TINY.
5. **Additional protection of a specified path**: Access is only, if the user authenticate himself by a defined **username** and **password**.

Top of list
Index page

TELNET server

Steps to protect the IPC@CHIP against unauthorized access via Telnet:

1. **Telnet timeout minutes**: Define telnet idle timeout minutes at chip.ini
2. **Telnet login delay**: Enable telnet login delay at chip.ini

3. **Telnet login retries**: Set telnet login retries at chip.ini
4. **Telnet user and passwords**: Define both user and password names at chip.ini
5. **Set the Stdio Focuskey to zero**: at chip.ini or inside of the application with **Set Stdio focus key** This disables the switching of stdio.

Comments

Since CHIP-RTOS version 1.01B telnet doesn't tell if the input of the username or the password input was wrong.

Top of list
Index page

FTP server

Steps to protect the IPC@CHIP against unauthorized access via FTP:

1. **FTP timeout**: Define FTP idle timeout seconds at chip.ini
2. **FTP login delay**: Enable FTP login delay at chip.ini
3. **FTP user and passwords**: You should define both user and password names at chip.ini
4. **FTP user root directory**: For a "normal" user you should define a root directory above "\".
5. **FTP user drive**: If you specify a rootdirectory you also must set a drive.
6. **Hide files with int21h 0x43**: Hidden files are not visible at FTP sessions or by the DIR command

Comments

Since @CHIP-RTOS version 1.01B FTP doesn't tell if the username input or the password input was wrong.

Top of list
Index page

PPP server

Steps to protect the IPC@CHIP against unauthorized access via PPP:

1. **PPP server idle timeout**: Define PPP server idle timeout seconds at chip.ini
2. **PPP users and passwords**: Define both user and password names for the the PPP server at chip.ini

Top of list
Index page

Chiptool UDP config server

Protect the IPC@CHIP against unauthorized access by using the Chiptool program:

1. **UDP config server**: Set the UDP config server security level at chip.ini

Top of list
Index page

TFTP server

Protect the IPC@CHIP against unauthorized access via TFTP:

1. **Disable/enable TFTP**: Disable/enable TFTP with shell command

Top of list
Index page

General TCPIP network security

1. **Install System Server Connection Handlers** provides the possibility to generate IP- and/or Port - filters and forbid connections to FTP, WEB or Telnet
2. The BIOSINT API function **Suspend System Servers** allows you to Supend/Resume the FTP, Web and Telnet Server while runtime.
3. TCPIP API function **Register an IP callback filter function** allows the application programmer to install a filter callback function on every incoming IP packet.
4. TCPIP API function **Register an ARP callback filter function** allows the application programmer to install a filter callback function on every incoming ARP packet.

Top of list
Index page

End of document

Boot Flow chart SC12 @CHIP-RTOS V1.10

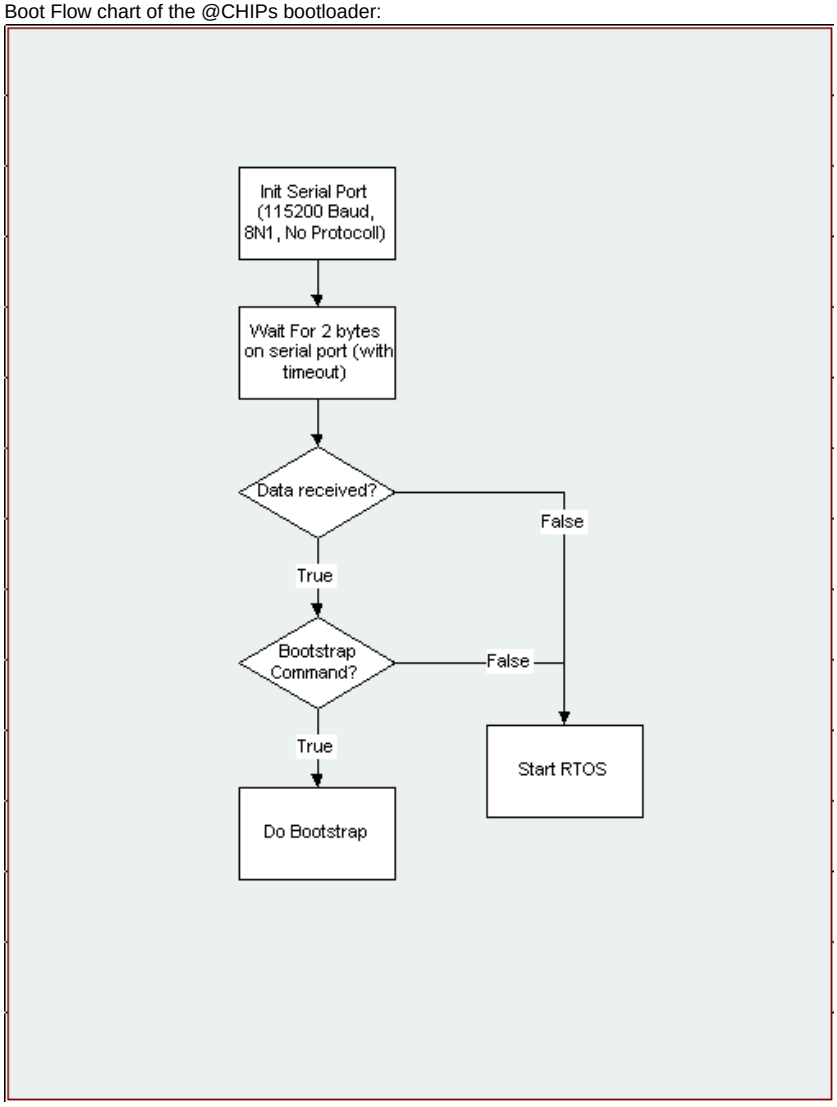
Introduction

This page describes the boot process and its error handling of the IPC@CHIP. Also a list of all possible errors during boot up is attached.

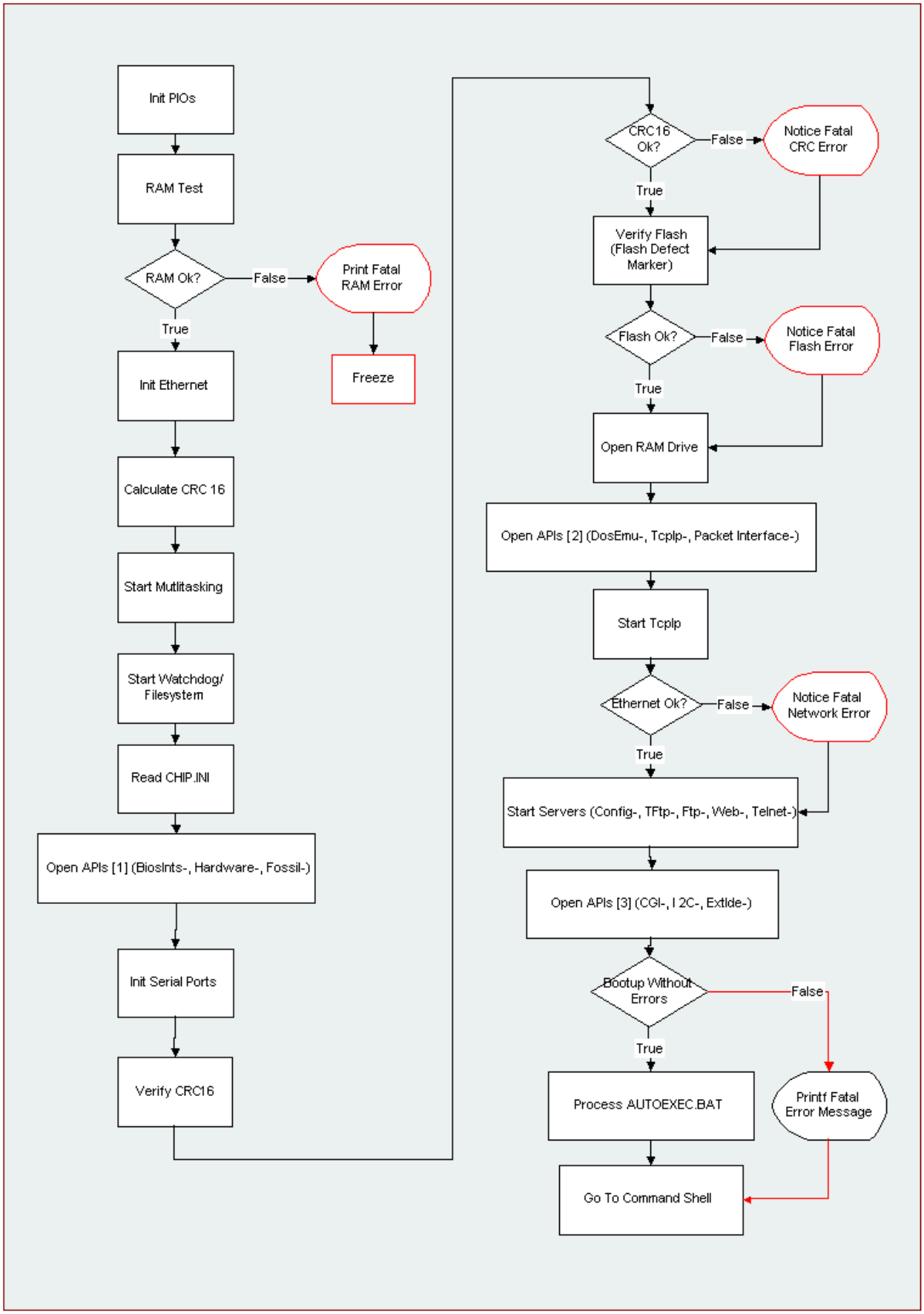
Boot Flow Chart

The boot process consists of two steps. In the first step the bootloader will be executed. It checks on the serial EXT port if an @CHIP-RTOS update request comes in (from CHIPTOOL). If not, the bootloader goes to the second step by jumping into the @CHIP-RTOS startup code. Below there are the flow charts of step one (bootloader) and step two (@CHIP-RTOS)
The errors in the diagrams will be described in the next point (**Possible Error while the boot process**)

1. Boot Flow chart of the @CHIPs bootloader:



2. Boot Flow chart of the @CHIP-RTOS:



Top of list
Index page

Possible errors during the boot process

During the boot process there could occur some errors. All possible errors are mentioned in the list below. They do prevent the start of the AUTOEXEC.BAT, so your application will not start automaticly (to prevent fatal errors while runtime). The error which occurs will be printed on the

Command Shell via the COM port.

1. RAM Error:
   On startup of the @CHIP-RTOS a RAM test will be performed. If it fails, the @CHIP prints the error message "RAM ERROR" on the COM port (using 19200 Baud and 8N1, regardless the entry in the CHIP.INI). The @CHIP-RTOS freezes after the error message output (does not boot!).
2. Checksum Error:
   During the boot process a checksum over the complete @CHIP-RTOS image in the flash will be generated. This checksum must be equal to the checksum which was written with the @CHIP-RTOS update (in a special sector on the flash). If the checksum is not equal, the error message "Fatal BIOS checksum error" will be shown on the serial console. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.
3. Flash Error:
   Every flash write access of the @CHIP-RTOS will be verified. If the flash access fails, an internal flag will be set, to mark the sector which should be written as defect. On the next reboot the error message "Flash error at sector: xx" will be shown on the serial console. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.
4. Network Error:
   During the init process of the TCP/IP it could be possible that a network error occurs (mostly it is a hardware defect). In this case the message "Fatal Network Error" will be printed on the serial console of the IPC@CHIP. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.

Top of list
Index page

End of document
