Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

eZ80® CPU

Zilog TCP/IP Stack API

Reference Manual

RM004012-0707

Warning: DO NOT USE IN LIFE SUPPORT

Revision History

Each instance in the Revision History reflects a change to this document from its previous revision. For more details, refer to the corresponding pages or appropriate links given in the table below.

Date	Revision Level	Description	Page No
July 2007	12	Globally updated ZiLOG as Zilog.	All
July 2007	11	Globally updated for ZTP v2.1.0 release.	All
June 2007	10	Updated document as per Zilog Style Guide. Updated ioctlsocket, ftp_connect, do_programatic_login, do_a_ftp_command, Http_Request Structure, accept, listen, hgleave, name2ip, xc_ascdate, Table 15, Table 16. Removed Kernel API's, Process Manipulation Functions, Semaphore Functions, Mailbox Messaging Functions, Message Port Functions, Miscellaneous Operating System Functions, Kernel Macros, Sample usage in `.c` and `.asm` Files sections. Removed Appendix B, Appendix C, Appendix D.	All
July 2006	09	Globally updated for ZTP v2.0.0 release.	All
April 2006	08	Globally updated for ZTP v1.7.0 release. Added the registered trademark symbol (®) for eZ80Acclaim! and eZ80.	All

Table of Contents

Introduction

This Reference Manual describes the APIs associated with Zilog's TCP/IP (ZTP) Stack v2.1.0 for Zilog's eZ80® CPU-based microprocessors and microcontrollers. This ZTP release supports the eZ80 family of devices, which includes eZ80L92 microprocessor, and eZ80Acclaim!® family of devices (that is, eZ80F91, eZ80F92, and eZ80F93 microcontrollers).

About This Manual

Zilog® recommends that you read and understand everything in this manual before using the product. We have designed this manual to be used as a reference guide for ZTP APIs.

Intended Audience

This document is written for Zilog customers who are familiar with real-time operating systems and are experienced at working with microprocessors, in writing assembly code, or in writing higher level languages such as C.

Manual Organization

This Reference Manual is divided into fifteen sections and an appendix. A brief description of each section and appendix is provided below.

ZTP API Reference

This chapter describes the ZTP APIs in detail. It also comprises of the following sub-sections.

- ZTP Networking APIs

- HTTP Function
- SNMP Functions
- SMTP Function
- Telnet Functions
- TimeP Protocol Function
- DNS Functions
- RARP Function
- IGMP Functions
- TFTP Functions
- FTP Functions
- Ping Functions
- SNTP Functions

Appendix A—Definitions and Codes

This appendix lists the enumerations and different data type definitions used in ZTP.

Related Documents

Table 1 lists the related documents that you must be familiar with to use ZTP efficiently.

Table 1. Related RZK Documents

Document Title	Document Number
eZ80L92 Product Specification	PS0130
eZ80F91 Product Specification	PS0192
eZ80F92/eZ80F93 Flash MCU Product Specification	PS0153

Table 1. Related RZK Documents (Continued)

Document Title	Document Number
eZ80F92/eZ80F93 Ethernet Module Product Specification	PS0186
eZ80F92/eZ80F93 Flash Module Product Specification	PS0189
eZ80 CPU User Manual	UM0077
Zilog Real-Time Kernel Reference Manual	RM0006

Manual Conventions

The following convention is adopted to provide clarity and ease of use:

Courier Typeface

Code lines and fragments, functions, and various executable items are distinguished from general text by appearing in the Courier typeface. For example, `#include <socket.h>`.

Safeguards

When you use ZTP along with one of Zilog's development platforms, always use a grounding strap to prevent damage resulting from electrostatic discharge (ESD) to avoid permanent damage to the development platform.

ZTP API Reference

Zilog TCP/IP Stack consists of a rich-set of APIs for accessing the TCP/IP protocol stack. This section provides a description of each ZTP API including inputs and outputs. Each API is classified according to the protocol or command that it is associated with.

Table 2 provides a quick reference to ZTP APIs based on its protocol.

Table 2. ZTP API Quick Reference

ZTP Networking APIs
HTTP Function
HTTPS Function
SNMP Functions
SMTP Function
Telnet Functions
TimeP Protocol Function
DNS Functions
RARP Function
IGMP Functions
TFTP Functions
FTP Functions
Ping Function
SNTP Functions

ZTP Networking APIs

This section describes the user interfaces to the ZTP stack. All the APIs listed in this section return a negative value if an error occurs. Positive values are considered to be the expected output.

Table 3 provides a quick reference to ZTP Networking APIs.

Table 3. ZTP Networking APIs Quick Reference

socket	recvfrom
bind	sendto
accept	ioctlsocket
listen	getsockname
connect	getpeername
recv	inet_addr
send	inet_ntoa
close_s	

socket

Include

```
#include <socket.h>
```

Prototype

```
INT16 socket (
 INT16 af,
 INT16 type,
 INT16 protocol
);
```

Description

The `socket` function creates a socket that is bound to a specific service provider.

Argument(s)

`af`	An address family specification. ZTP supports only the `AF_INET` internet address family.
`type`	A `type` specification for the new socket. ZTP supports the following two types of sockets: SOCK_STREAM—Provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism. Uses TCP for the Internet address family. SOCK_DGRAM—Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP for the Internet address family. Socket type definitions appear in the `socket.h` header file.
`protocol`	The `protocol` function is a particular protocol to be used with sockets that are specific to an indicated address family. As this parameter is not used, the value passed must be zero across all versions of ZTP.

The `socket` function causes a socket descriptor and any related resources to be allocated and bound to a specific transport service provider.

Return Value(s)

If successful, the `socket` function returns the socket descriptor, the value of which must be greater than or equal to 0.

If the returned value is less than 0, one of the following errors is returned.

EPROTONOSUPPORT	Protocol not supported
ENOBUFS	Buffer not available

bind

Include

```
#include <socket.h>
```

Prototype

```
INT16                 bind (
 INT16                s,
 struct sockaddr * name,
 INT16                namelen
);
```

Description

The sockets' `bind` function associates a local address with a socket.

Argument(s)

`s`	A descriptor identifying an unbound socket.
`name`	The address to assigned to the socket from the `sockaddr` structure.
`namelen`	The length of the name parameter.

Note: *The* `bind` *function is used on an unconnected socket before subsequent calls to the* `connect` *and* `listen` *functions. It is used to bind either connection-oriented (stream) or connectionless (datagram) sockets. Use* `bind` *function to establish a local association of the socket by assigning a local name to an unnamed socket.*

ReturnValue(s)

If successful, the `bind` function returns `ZTP_SOCK_OK`.

If less than 0, one of the following errors is returned.

EFAULT	Address family not supported.

EINVAL	Invalid socket descriptor (descriptor already in use).
EBADF	Invalid socket descriptor (not allocated).

See Also

sockaddr Structure

accept

Include

```
#include <socket.h>
```

Prototype

```
INT16                 accept
(
 INT16                s,
 struct sockaddr      *peername,
 INT16                *peernamelen
);
```

Description

The sockets' `accept` function accepts an incoming connection attempt on a socket.

Argument(s)

`s`	A descriptor identifying a socket that has been placed in a listening state with the `listen` function. The connection is made with the socket that is returned by `accept`.
`peername`	An optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer. The exact format of the `peername` parameter is determined by the address family established when the socket connection was created.
`peernamelen`	An optional pointer to an integer that contains the length of the `peernamelen`.

Notes: 1. *The* `accept` *function extracts the first connection on the queue of pending connections on socket* `s`*. It then creates a new socket and returns a handle to the new socket. The newly-created socket is the socket that handles the actual connection. The* `accept` *function can block the caller until a connection is present if no pending connec-*

tions are present in the queue, and the socket is marked as blocking. If the socket is marked nonblocking and no pending connections are present in the queue, `accept` *returns an error, see* Return Value(s) *below. After successful completion,* `accept` *returns a new socket handle. The original socket remains open and listens for new connection requests.*

2. The `addr` parameter is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. `addrlen` is a value-result parameter that should initially contain the amount of space pointed to by `addr`; upon return, it contains the actual length
   (in bytes) of the returned address.
3. *The* `accept` *function is used with connection-oriented socket types such as* `SOCK_STREAM`*.*

Return Value(s)

Success	If no error occurs, `accept` returns a value of type `INT16` that is a descriptor for the new socket. The integer referred to by `addrlen` initially contains the amount of space pointed to by `addr`. Upon return, it contains the actual length in bytes of the address returned.
Failure	One of the following error codes is returned.
	EOPNOTSUPP—Socket type not supported.
	EBADF—Invalid socket descriptor.
	EINVL—Invalid socket descriptor.
	ENOCON—Connection not arrived.
	EFAULT—Error accepting new socket.

See Also

sockaddr Structure

listen

Include

```
#include <socket.h>
```

Prototype

```
INT16 listen (
INT16 s,
INT16 backlog
);
```

Description

The sockets' `listen` function places a socket into a state within which it listens for an incoming connection.

Argument(s)

`s`	A descriptor identifying a bound, unconnected socket.
`backlog`	The maximum length of the queue of pending connections. If this value is MAXSOCKS, then the underlying service provider responsible for socket `s` sets the backlog to a maximum *reasonable* value.

Notes:

1. *The socket* `s` *is placed into passive mode in which incoming connection requests are acknowledged and queued pending acceptance by the process.*
2. *Servers that can facilitate more than one connection request at a time use the* `listen` *function.*

Return Value(s)

Success	If no error occurs, `listen` returns a 0.

Failure	One of the following values is returned.
	EINVAL—Invalid socket descriptor.
	EBADF—Invalid socket descriptor (not allocated).
	EOPNOTSUPP—Socket type not supported.
	EFAULT—backlog exceeding MAXSOCKS.

connect

Include

```
#include <socket.h>
```

Prototype

```
INT16                connect
(
 INT16               s,
 struct sockaddr     *peername,
 INT16               peernamelen
);
```

Description

The sockets' `connect` function establishes a connection to a specified socket.

Argument(s)

`s`	A descriptor identifying an unconnected socket.
`peername`	A pointer to the socket structure specifying the host to connect to.
`peernamelen`	The size of the `peername` parameter structure.

Notes: 1. *The* `connect` *function is used to create a connection to a specified destination. If the socket* `s` *is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound.*

2. *By default,* `connect` *is a blocking call and is not returned unless connection is established or is refused.*

ReturnValue(s)

Success	If no error occurs, `connect` returns `ZTP_SOCK_OK`.
Failure	One of the following errors is returned.
	EAFNOSUPPORT—Address family not supported.
	EINVAL—Invalid descriptor.
	ECONNREFUSED—Connection refused by peer.

See Also

sockaddr Structure

recv

Include

```
#include <socket.h>
```

Prototype

```
INT16                  recv
(
 INT16                 s,
 INT8                  * buf,
 INT16                 nbyte,
 INT16                 flags
);
```

Description

The sockets' `recv` function receives data from a connected socket.

Argument(s)

`s`	A descriptor identifying a connected socket.
`buf`	A pointer to a buffer for the incoming data.
`nbyte`	The length of `buf`.
`flags`	Reserved for future use.

Notes:

1. *The `recv` function reads incoming data on connection-oriented sockets. The sockets must be connected before calling `recv`. For a connected socket, the `recv` function restricts the addresses from which received messages are accepted. The function only returns messages from the remote address specified in the connection. Messages from other addresses are silently discarded.*
2. *For connection-oriented sockets (type `SOCK_STREAM` for example), calling recv returns as much information as is currently available (up to the size of the buffer supplied).*

3. *Zilog recommends not using* `recv()` *with datagram sockets.*

ReturnValue(s)

Success	If no error occurs, `recv()` returns the number of bytes received. If the connection has been gracefully closed, the return value is EFAULT.
Failure	One of the following error codes is returned:
	EDEADSOCK—Socket is closed.
	EBADF—Invalid descriptor.
	EPIPE—Invalid socket type.
	ZTP_ALREADY_BLOCKED (-18)—One thread is already blocked.

send

Include

```
#include <socket.h>
```

Prototype

```
INT16 send
(
 INT16 s,
 INT8 *buf,
 INT16 nbyte,
 INT16 flags
);
```

Description

This sockets' `send` function sends data on a connected socket.

Argument(s)

`s`	A descriptor identifying a connected socket.
`buf`	A buffer containing the data to be transmitted.
`nbyte`	The length of the data in `buf`.
`flags`	An indicator specifying the method in which a call is made. If used, `tcp_FlagPUSH`, the appropriate outbound TCP segment, contains a PSH flagset in code bits.

Notes: 1. *The* `send` *function is used to write outgoing data on a connected socket. The successful completion of a* `send` *does not indicate that the data was successfully delivered.*

2. *If no buffer space is available within the transport system to contain the data to be transmitted,* `send` *blocks unless the socket is placed in a nonblocking mode.*

3. *On non-blocking stream-oriented sockets, the number of bytes written is between one and the requested length, depending on buffer availability on both client and server.*

Return Value(s)

Success	If no error occurs, `send` returns the total number of bytes sent, which can be less than the number indicated by `len` for nonblocking sockets.
Failure	One of the following errors is returned:
	EDEADSOCK—The socket is closed.
	EBADF—Invalid descriptor.
	EPIPE—Invalid socket type.
	ZTP_ALREADY_BLOCKED (-18)—One thread is already blocked.

See Also

ZTP Core Macros

close_s

Include

```
#include <socket.h>
```

Prototype

```
INT16 close_s (INT16 s);
```

Description

The sockets' `close_s` function closes an existing socket.

Argument(s)

`s`	A descriptor identifying a socket to close.

Notes:

1. *The* `close_s` *function closes an active socket. This function is used to release the socket descriptor* `s` *so that further references to* `s` *fail. Any pending asynchronous or blocking calls issued by any thread in this process are cancelled without any notification messages displayed. To return any socket resources to the system, an application must contain a matching call to* `close_s` *for each successful call to the socket.*

2. *If* `close_s` *is issued on a master socket (a socket used in TCP server application and passed to the accept call as a parameter), all listening sockets on the same port are closed to accept those sockets that are already in the established state.*

Return Value(s)

Success	ZTP_SOCK_OK
Failure	EBADF—Invalid socket descriptor (not allocated).