User's Manual

TCP/IP for Embedded system M3S-T4-Tiny

User's Manual

RENESAS MCU

Renesas Electronics
www.renesas.com

Rev.1.11 2019.06

How to Use This Manual

This document describes a software library M3S-T4-Tiny for Renesas microcomputer.

1. Organization of This Manual

This manual can be broadly divided into the following units.

Table of Contents

1. Introduction

1.1 Overview

The Tiny TCP/IP library M3S-T4-Tiny (hereafter referred to as "the T4") is the network software library for the following series and groups of microcomputers. This manual provides the common information necessary to create application programs using the T4. The MCU-dependent information (e.g., development environment) is supplied in the respective Introduction Guide.

Caution This is scheduled to expand for corresponding CPU and for corresponding protocol, Ethernet and PPP. The latest information of T4 (support MCU and support network (Ethernet/PPP)) is shown on Renesas Web page.

Figure 1.1 shows the position of the T4 in a software group associated with network middleware.

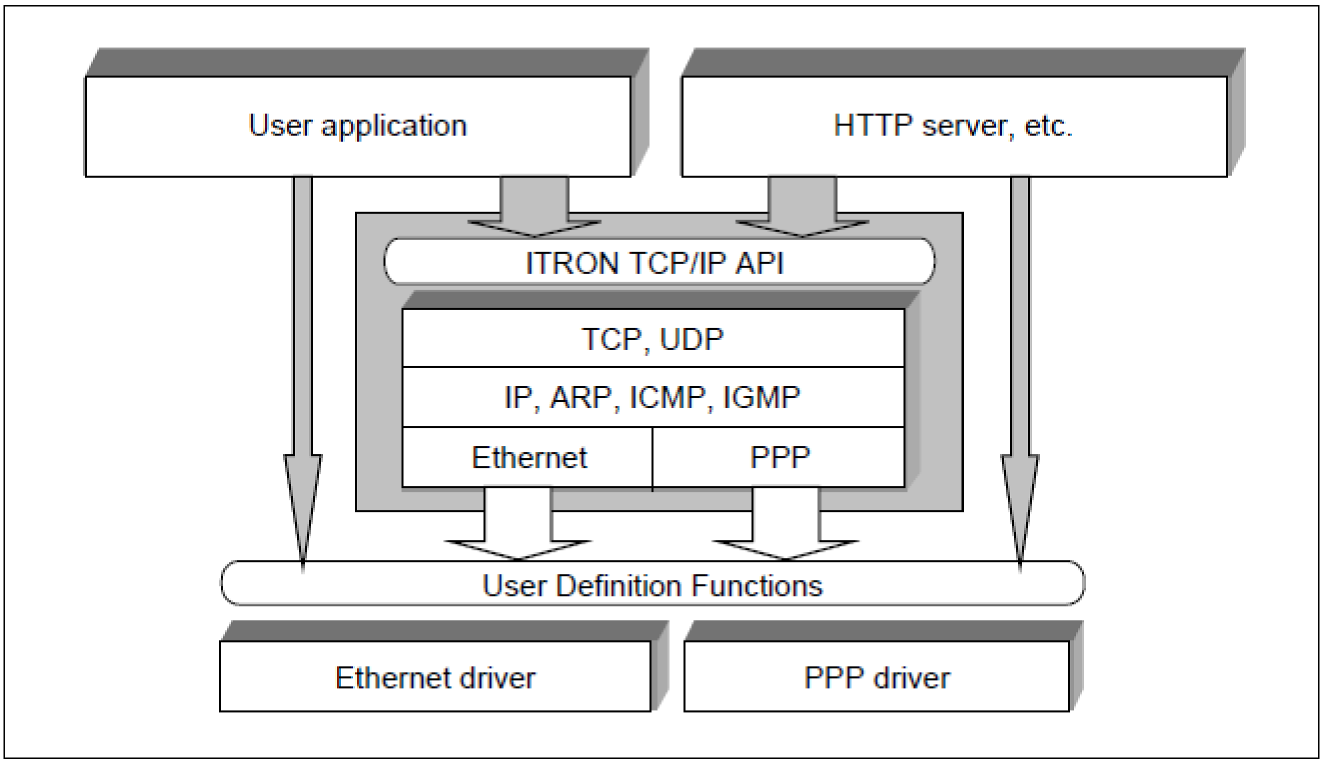


Figure 1.1 Position of the T4 in Network Software Structure

Figure 1.1 shows the position of the T4 in a general structure of the software that comprises network middleware.

Shown in the gray background at center is the T4.

The arrows represent function calls.

The T4 performs protocol processing between the Ethernet or PPP driver and an application program as it sends or receives data to and from the network.

Included as protocol processing in the T4 are TCP, UDP, IP, ICMP, IGMP, ARP and PPP.

To perform TCP or UDP communication from a user application or a higher-level protocol program, functions must be called from within the library by using the API (compliant with ITRON TCP/IP API) that is stipulated in the T4. IP, ICMP, IGMP, ARP and PPP are the lower-level protocols below TCP and UDP, so that in no case will the functions to process these protocols be called directly from a user application. (Regarding IGMP, the user calls the multicast group join and leave functions as required.)

The T4 supports Ethernet and PPP as the data link and physical layers. Of these, part of the library that depends on the user hardware such as LAN controller or serial I/O is separated as a driver from the library body.

To create drivers matched to the hardware used, refer to specifications for the driver API that are written in the user's manual for the driver concerned.

Caution The user's manuals for drivers are included with the product package separately from this manual.

When using the T4 to create a program, make sure that the driver's initialization and termination functions are called directly from the program. (In **Figure 1.1**, these function calls are shown by the gray arrows.) Specifications of these API functions are described in the T4 manual, not just in the user's manual for the driver.

T4 supports several LAN port. It is enable to set the several IP address to the several LAN ports.

This product package comes with sample programs for drivers. Refer to these sample programs when creating your original driver.

The T4 library does not use the real-time OS.

1.2 Basic Knowledge of TCP/IP

This section describes the basic knowledge of TCP/IP which may be needed to use the T4.

The T4 has adopted APIs suitable for embedded systems, and is therefore partly different from general TCP/IP such as those in BSD or Linux. These differences are described later, as compared with the general TCP/IP described in this section.

The contents described here are limited to the absolute minimum that is needed to be able to understand what is written in this manual. For more information about TCP/IP, please read technical books generally available on the market.

1.2.1 About the Connection

TCP creates a virtual communication path to communicate. This virtual communication path is referred to as a "connection". To have a connection with the other node of communication is referred to as "establish a connection". One program (a program running on the computer used for communication) can have two or more connections, and can therefore communicate with multiple computer programs. However, only one node can be communicated in one connection.

To establish one connection, a negotiation between the local and remote node (i.e., between the two sides of communication) must be performed. Once a connection is established, communication can be performed in two directions. (There is no need to have two different connections, one for transmission and one for reception.) The procedure for establishing a connection this way is referred to as "request a connection".

On the other hand, UDP does not establish a connection to perform communication. Because communication is attempted unilaterally without confirming the communication status of the other node, no steps are taken to establish a connection or negotiate with the other node to confirm whether communication has terminated normally.

1.2.2 About the Client-Server Model

TCP uses the concept of a client-server model in the negotiation to establish a connection described above. One node of communication plays the role of a "client", and the other plays the role of a "server". When a connection is established, the server only waits for a request for connection to be issued by some unidentified node of communication, while the client specifies the address of the server to communicate with to identify the other node of communication before requesting a connection.

Once a connection is established through steps taken on both sides, whichever side - client or server - can perform transmit or receive operation as desired.

Generally speaking, however, the TCP/IP communication program itself is implemented in the form of a client-server model. In this case, the server is implemented so as to wait for a request from the client and return information to the client.

1.2.3 IP Address and Port Number

The IP address and port number are the address information necessary to indicate the local or remote node in TCP/IP or UDP/IP communication.

- The IP address is a unique address that indicates the communication port through which to communicate.
- The port number is a number that indicates the application program with which to communicate.

The IP address must be a unique number throughout the world except for local networks not connected to Internet. (There is a special organization by which IP addresses are assigned).

The port number must be unique among all communication programs operating in the same computer.

1.2.4 About the Socket

General TCP/IP uses an API called the "socket" in communication. Although complicated processing is performed in TCP/IP and UDP/IP packet communication, various APIs such as those for initialization, request connection, transmit, receive and termination processing are available to use when creating application programs, allowing for communication processing to be programmed easily - as if programming C language file input/outputs - without concern for complicated packet processing. An abstract data structure known as the socket is used in APIs.

1.3 Basic Knowledge of ITRON TCP/IP API Specification

This section describes the basic knowledge of ITRON TCP/IP API Specification (hereafter called simply ITRON Specification) that is adopted in the T4. However, the contents described here are limited to the absolute minimum that is needed to be able to understand what is written in this manual. For more information about ITRON TCP/IP API Specification, please read the specifications issued by T-Engine Forum, which are published at the home page shown below.

URL: https://www.tron.org/specifications/

Document No.: TEF024-S003-02.00.00

1.3.1 Handling of Connection

In ITRON TCP, the procedure for requesting a connection is referred to as "request connection" or "wait for connection request". The difference between the two will be described later.

1.3.2 Handling of a Client/Server Model

In ITRON TCP too, the client/server model is used in connection procedures as in general TCP/IP. However, the term "client" or "server" is not used in ITRON TCP specifications. Instead, two separate connection-related APIs are provided, one for server use which is called "Wait for Connection Request (Passive Open)", and one for client use which is called "Request Connection (Active Open)". Namely, one that "requests a connection" is the client, and one that "waits for a connection request" is the server.

1.3.3 Reception Points and Communication End Points

In ITRON TCP, the "socket" structure is not used in communication. Instead, the structures called the "reception point" and "communication end point" are used. These types of structures have been conceived as a result of considerations given to the following requirements for embedded systems:

(1) Minimization of the memory area for buffers and the number of times data are copied.

(2) No dynamic memory management done within the protocol stack is desirable.

(3) Support for an asynchronous interface or non-blocking calls.

(4) Comprehensibility of detailed information on errors for the respective API functions is desirable.

(Excerpt from specifications issued by T-Engine Forum)

1.4 Definition of Terms

1.4.1 Reception Point

The reception point is a structure used in an API during TCP communication. More specifically, it is used in an API (function tcp_acp_cep()) that a node (server) uses in order to wait for a connection request from the other node of communication (client). There can be two or more reception points in the entire program, which are defined as one array of structures in the entire program.

The API waiting for a connection request uses a parameter to specify the reception point used. This specification is made by specifying a unique ID number beginning with 1, and not by specifying a pointer to the structure. The ID number, rather than being defined by the programmer, is automatically assigned by the T4 sequentially beginning with the first ID number when defining an array of reception point structures.

Therefore, the programmer can keep track of which reception point in a defined structure array is assigned which ID number.

In specifications stipulated by the TRON Association, the reception point has as its members an attribute and the local IP address and port number.

1.4.2 Communication End Points

The communication end point is a structure used in APIs during TCP and UDP communication. The communication end point (as for the reception point) is defined as an array of structures, with each structure identified by an ID number that is assigned sequentially beginning with the first structure. These IDs for communication end points are used when establishing a connection with the other node, sending or receiving data, and closing the connection.

In specifications stipulated by the TRON Association, the TCP communication end point has as its members an attribute, top address and size of transmit window, top address and size of receive window, and an address of callback routine.

In TCP, the communication end point whose connection is completely closed is referred to as in an "unused" state.

In specifications stipulated by the TRON Association, the UDP communication end point has as its members an attribute, the local IP address and port number, and an address of callback routine.

1.4.3 Timeout

When an API is called, its processing may take time, to be kept waiting in the function. In this case, if the processing cannot be completed within a specified time (timeout period), it can be canceled, allowing for return from the API. This condition is referred to as "timeout".

1.4.4 Non-Blocking Calls and Callback

A non-blocking call refers to one in which when a task is kept waiting in an API, it returns from the API while letting processing to continue. Then, after returning from the API, the task may notify the application that the continued processing has finished, which is referred to as "callback".

1.4.5 Polling

Polling is the same as a timeout processing in which the timeout period is set to 0.

1.4.6 Pending

The state of a task kept waiting in the API is referred to as "pending".

1.4.7 MSS (Maximum Segment Size)

MSS refers to the maximum size of a segment that can be transmitted at a time in TCP protocol processing. It is determined uniformly in a system. (*1 Even in the T4, MSS must be defined.) If the transmit data passed in an API from an application program is larger than the MSS, the TCP protocol processing unit divides it into MSS for transmission as several segments. The transmit data that is divided this way is a data-only part that does not include the TCP header.

In TCP, the MSS information on the local and remote node are exchanged at the initial stage of communication, and data is transmitted in a smaller MSS size of the two. The MSS thus selected is placed in the option part of the TCP header. Therefore, this feature is expressed as "MSS in header option".

Caution 1 The T4 supports this feature.

1.4.8 IP Fragments

One of the optional features of IP protocol processing, this is a mechanism used in cases where the router uses a communication path with small MTU (Maximum Transmission Unit) and the data to transmit is larger than the MTU in size. In such a case, the data is divided into fragments smaller than the MTU before being transmitted, and the data ultimately is reconstructed on the receiver side. In Ethernet, for example, the MTU is 1,500 bytes, so that if the data to transmit is larger than that, it must be divided into smaller sizes.

Caution The T4 does not support this feature. Therefore, if the need arises to use a communication path with small MTU to communicate, the MSS must be set to a size smaller than the MTU.

1.4.9 Receive Window

This refers to a buffer area in which to store the received data. In TCP, the sender waits for an acknowledge from the other node each time it transmits a segment. However, because this results in an increased overhead, the receiver notifies the sender of the remaining size of the receive window by using the TCP header during communication, allowing the sender to transmit segments successively without waiting for acknowledges each time until the receive window is filled.

When the receive window is filled during communication, the sender stops transmitting segments upon receipt of notification, and waits until it is notified by the receiver that the receive window is empty.

Caution T4 waits for transmitting the ACK packet and transmits it twice.

1.4.10 PPP (Point to Point Protocol)

This protocol is used to place IP protocol on a serial communication line. It includes as its ancillary protocol a procedure for authenticating the user after connecting the line.

1.4.11 PAP (Password Authentication Protocol)

One of authentication protocols in PPP connection, this is used to authenticate the correctness of the password transmitted by the originator. Although passwords are encrypted on the server side, they are not encrypted on communication lines. One of the similar protocols is CHAP, which because passwords are encrypted, provides higher reliability than PAP.

1.4.12 ICMP (Internet Control Message Protocol)

Always used along with IP, this protocol is provided for notifying errors that occurred when delivering IP packets, as well as for inspecting or notifying the IP status of each host.

1.4.13 ARP (Address Resolution Protocol)

This protocol is used to convert the IP address into the MAC address (e.g., Ethernet address).

1.4.14 IGMP (Internet Group Management Protocol)

IGMP is the protocol for multicast. In order to implement multicasting, it is necessary to join to the multicast group that managed by IGMP-router. Multicasting is a technology that boosts communication efficiency when transmitting the same data simultaneously to multiple hosts, and it is principally used to stream data such as video or audio. IGMP is a multicast protocol designed for IPv4 networks, and it sits on top of the IP layer. A similar protocol exists in IPv6, but it is called Multicast Listener Discovery (MLD) rather than IGMP.

1.4.15 DHCP (Dynamic Host Configuration Protocol)

DHCP is the protocol that automatically allocates the information for TCP/IP communication. DHCP server exists on the network manages the state of allocated IP address for DHCP client for own network. DHCP client gets the information for TCP/IP communication from DHCP server via using DHCP initiation sequence (Send DISCOVER / Receive OFFER / Send REQUEST / Receive ACK). This information has the limit of utilization time called "lease time". When half of this time is passed, DHCP client will execute extending this time. This execution is called "RENEW". This behavior will be repeatedly executed until success. When succeeded, TCP/IP communication is still available. When failed and lease time expired, the IP address is also expired, and execute DHCP initiation sequence. This behavior will not be occurred if DHCP server specifies lease time as "no limit".

1.5 Program Development Produce

The flowchart in **Figure 1.2** shows the development of application program for the T4.

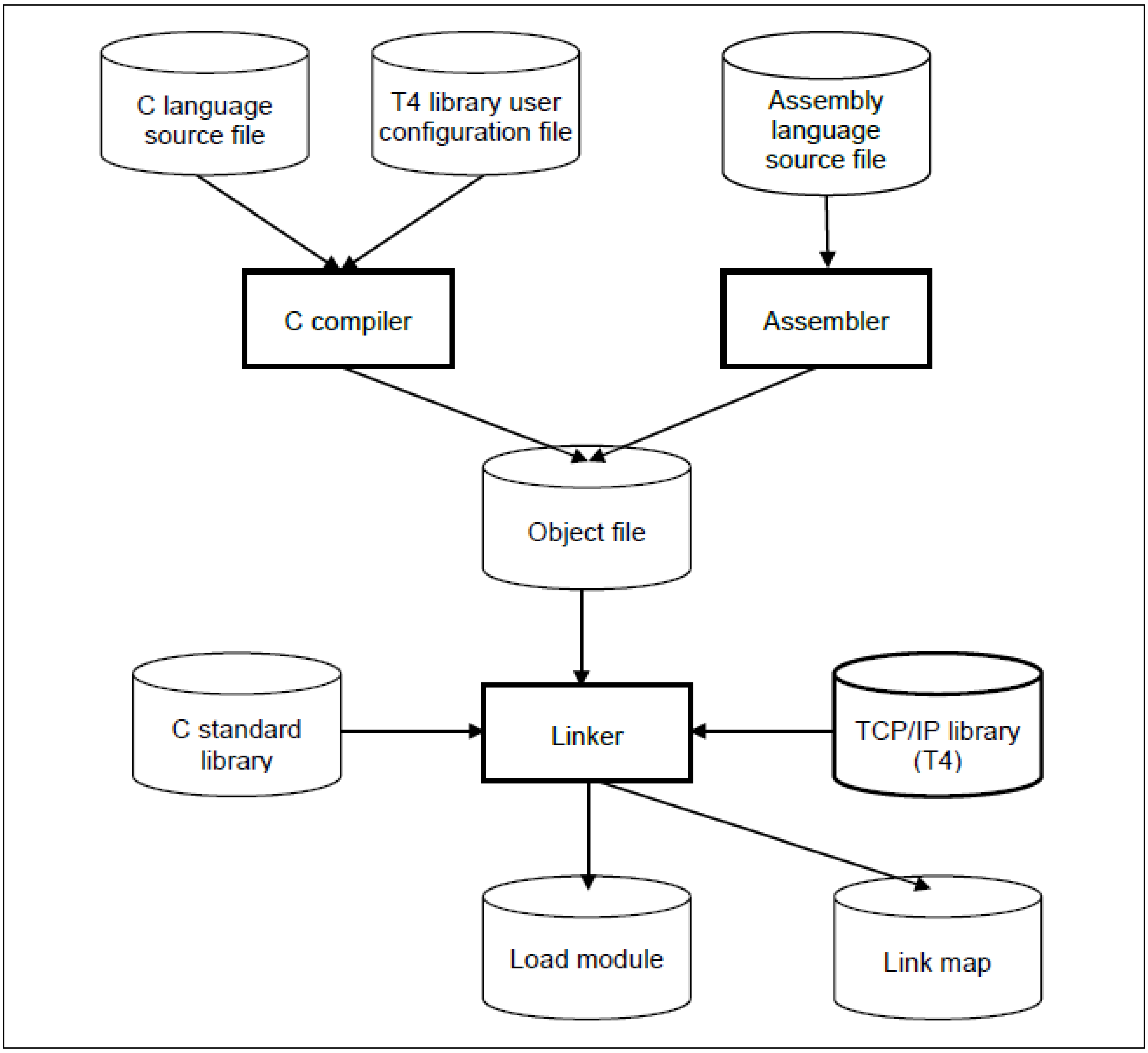


Figure 1.2 The Development Flowchart of Application Program

2. Outline of the T4

2.1 Product Information

The T4 consists of library files, sample drivers and sample application.

Table 2.1 Directory Structure for the T4

Item		Description
Document (doc)		
	rzzz1-t4tiny.pdf	User's manual (this note)
	rzzz2-t4tiny.pdf	Ethernet driver interface specification
	rzzz3-t4tiny.pdf	PPP driver interface specification
	rzzz4-xxx-t4.pdf	Introduction Guide
T4 Library (lib)		
	T4_Library_ether_xxx_yyy.lib	T4 Library file (for Ethernet)
	T4_Library_ether_xxx_yyy_debug.lib	Debugging information with T4 Library file (for Ethernet)
	T4_Library_ppp_xxx_yyy.lib	T4 Library file (for PPP)
	r_t4_itcpip.h	Header file for T4
Make Library environment (make_lib)		
	make_lib.zip	Make Library environment (include source code)
T4 Configuration File (src)		
	config_tcpudp.c	T4 Configuration File

Caution CPU kind is into the xxx (ex. rx600, h8s2600, sh2a), CPU settings is into the yyy (ex. big, little, advanced), and document name is into the zzz1-zzz4.

2.1.1 Document

These are complete package of information where information necessary to use T4 was described. The Introduction Guide has described information, version information, and the update history, etc. that depend on the microcomputer.

2.1.2 T4 Library

(1) T4 Library files (binary file)

These files contain API functions and various protocol processing programs. Use these files along with the application program after linking.

Use T4_Library_ether_xxx_yyy.lib when using Ethernet or T4_Library_ppp_xxx_yyy.lib when using PPP.

(2) Header files (r_t4_itcpip.h)

The T4 API prototype declarations, macro definitions, etc. are written in this file. For application programs that call T4 APIs, include this header file.

2.1.3 T4 Library make environment

These files contain source code and a set of projects for generating T4 library files. Users can debug the source code or customize the code for the system.

2.1.4 T4 Configuration File

This file contains configuration information of various protocols.The user must change according to the system. For details of setting information, see chapters 6 and 7.

2.2 Outline Library Specifications

Outline specifications of the T4 are shown in Table 2.2. This library supports the TCP, UDP, IP, ICMP, ARP and PPP protocols. The Application Program Interface (API) is compliant with ITRON TCP/IP API Specification.

Table 2.2 Outlines Specifications by Protocol

Protocol	Item	Specification
TCP	API	Compliant with ITRON TCP/IP API Specification
	Non-blocking call	Supported
	Callback feature	Supported
	Queuing	Not supported
	Maximum size *1	TCP data segment length (data length): 1,480 (1,460) bytes
	TCP header option	Only MSS is supported Reception: Header options except MSS are ignored Transmission: No header options except MSS are accepted
	TCP reception point	30 points
	Communication endpoint	30 points
UDP	API	Compliant with ITRON TCP/IP API Specification
	Non-blocking call	Supported
	Callback feature	Supported
	Queuing	Not supported
	Maximum size *1	UDP datagram length (data length): 1,480 (1,472) bytes
	Communication endpoint	30 points
	Multicast	Transmission and reception supported (224.0.0.0 to 239.255.255.255)
IP	Version	IPv4 (version 4) only
	Fragment	Not supported Reception: Fragmented datagrams are discarded Transmission: datagrams cannot be fragmented
	Header option	Not supported Reception: datagrams that include header options are discarded Transmission: No header options are accepted
	Maximum size *1	IP datagram (data length): 1,500 (1,480) bytes
ICMP	Message type	Only echo responses are supported Reception: Only echo request, Other messages discarded Transmission: Only echo response
IGMP	Message type	Only IGMPv1 and IGMPv2 supported
ARP	Cache entry	depend on User definition
	Cache retention time	Approx. 10 minutes
PPP	Server / Client	Server/Client are supported.
	Maximum size *1	PPP frame (data length): 1,504 (1,500) bytes
	Authentication method	PAP, CHAP with MD5
	Compression option	Compression-related options are not supported Setup requests for compressing the protocol field, address and control field and TCP/IP header are rejected
DHCP	Server / Client	Client

Caution 1 The maximum sizes of TCP, UDP, IP, and PPP depend on the transmission/reception buffers sizes of the driver. Ether can support several port, PPP supports one channel.

3. T4 Library Type Definitions

This section gives the details about the type definitions used in the library.

Table 3.1 Definitions Used in the Library

Datatype	Typedef
signed char	int8_t
unsigned char	uint8_t
signed short	int16_t
unsigned short	uint16_t
signed long	int32_t
unsigned long	uint32_t
signed char	B
signed short	H
signed long	W
unsigned char	UB
unsigned short	UH
unsigned long	UW
signed char	VB
signed short	VH
signed long	VW
void far	*VP
void	(*FP)();
signed long	INT
unsigned long	UINT
signed short	ID
signed short	PRI
signed long	TMO
signed short	HNO
signed long	ER
unsigned short	ATR
signed long	FN
ER(*)(ID cepid, FN fncd , VP p_parblk)	callback_from_tcpudp_t
ER(*)(UB channel, UW eventid, VP param))	callback_from_system_t

4. T4 Library Structures

This section gives the details of the structures used in the library.

User does not need using gray out members.

4.1 TCP Object Structure

Table 4.1 The TCPUDP_ENV structure is Internet Protocol related definition

Datatype	Structure Element	Explanation
UB	ipaddr[4]	Local IP address
UB	maskaddr[4]	Subnet mask
UB	gwaddr[4]	Gateway address

Table 4.2 The T_IPV4EP structure is IP address/Port No. information

Datatype	Structure Element	Explanation
UW	ipaddr	IP address
UH	portno	Port number

Table 4.3 The T_TCP_CREP structure is TCP reception point

Datatype	Structure Element	Explanation
ATR	repatr	TCP reception point attribute
T_IPV4EP	myaddr	Local IP address and port number

Table 4.4 The T_TCP_CCEP structure is TCP communication end point

Datatype	Structure Element	Explanation
ATR	cepatr	TCP communication end point attribute
VP	sbuf	Top address of transmit window buffer
INT	sbufsz	Size of transmit window buffer
VP	rbuf	Top address of receive window buffer
INT	rbufsz	Size of receive window buffer
ER	(*callback_from_tcpudp_t) (ID cepid, FN fncd, VP p_parblk)	Callback routine
UW	lan_port_number	Number of LAN port: 0 - LAN port number -1 Please refer to the section 6.1. to know LAN port settings
UW	keepalive_enable	KeepAlive function setting (0: Disable, 1: Enable)

4.2 UDP Object Structure

Table 4.5 The T_UDP_CCEP structure is UDP communication end point

Datatype	Structure Element	Explanation
ATR	cepatr	UDP communication end point attribute
T_IPV4EP	myaddr	Local IP address and port number
ER	(*callback_from_tcpudp_t) (ID cepid, FN fncd, VP p_parblk)	Callback routine
UW	lan_port_number	Number of LAN port: 0 - LAN port number -1 Please refer to the section 6.1. to know LAN port settings

5. T4 Library Enums

This section gives the details of the enums used in the library.

5.1 TCP Function Return Code

Table 5.1 TCP Function Return Code

Enum Value	Value	Significance
TFN_TCP_ACP_CEP	-0x205	Wait for TCP connection request
TFN_TCP_CON_CEP	-0x206	TCP connection request(active open)
TFN_TCP_SHT_CEP	-0x207	TCP data transmission end
TFN_TCP_CLS_CEP	-0x208	TCP communication end point close
TFN_TCP_SND_DAT	-0x209	Transmission of TCP data
TFN_TCP_RCV_DAT	-0x20A	Reception of TCP data
TFN_TCP_ALL	0	Select all TCP API function code

5.2 UDP Function Return Code

Table 5.2 UDP Function Return Code

Enum Value	Value	Significance
TFN_UDP_SND_DAT	-0x223	Transmission of UDP data
TFN_UDP_RCV_DAT	-0x224	Reception of UDP data (After call udp_rcv_dat())
TEV_UDP_RCV_DAT	0x221	Reception of UDP data (Before call udp_rcv_dat())
TFN_UDP_ALL	0	Select all UDP API function code

5.3 Error Codes Used in APIs

Table 5.3 Error Codes Used in APIs

Enum Value	Value	Significance
E_OK	0	Terminated normally
E_SYS	-5	System error
E_NOSPT	-9	Function is not supported
E_PAR	-17	Parameter error
E_OBJ	-41	Object status error
E_QOVR	-43	Queuing overflow
E_WBLK	-57	Non-blocking call accept
E_TMOUT	-50	Timeout
E_RLWAI	-49	Forced release from waiting
E_CLS	-52	Failed to connect
E_BOVR	-58	Buffer overflow

5.4 UDP Function Return Code

Table 5.4 Timeout

Enum Value	Value	Significance
TMO_POL	0	Polling
TMO_FEVR	-1	Waiting forever
TMO_NBLK	-2	Non-blocking call

5.5 Special IP Address and Port Numbers

Table 5.5 Special IP Address and Port Numbers

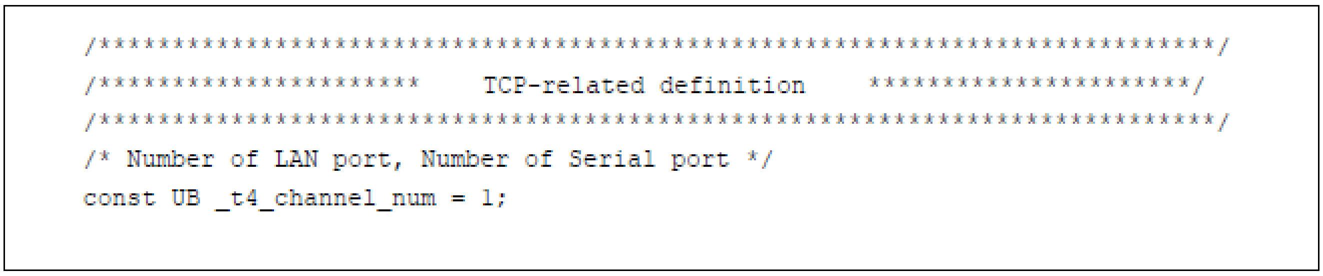
Enum Value	Value	Significance
TCP_PORTANY	0	TCP port number specification omitted
NADR	0	Invalid address

6. T4 Configuration File

Define TCP reception points, TCP communication end points, UDP communication end points, TCP options, IGMP, DHCP, and local node settings (Ethernet or PPP). These definitions can be made in an application program providing that they are set before initializing the T4. However, all items must be set regardless of how they will be used.

Each of the following definitions is explained by using the configuration file config_tcpudp.c as an example.

6.1 Definition of LAN Port Number



```
/****************************************************************************/
/**********************    TCP-related definition    **********************/
/****************************************************************************/
/* Number of LAN port, Number of Serial port */
const UB _t4_channel_num = 1;
```

Figure 6.1 Definition of LAN Port Number (from config_tcpudp.c)

Please set the LAN port number (=1) to the variable _t4_channel_num.

Do not set other value to the _t4_channel_num.

6.2 Definition of TCP Reception Points

```
/***************************************************************************/
/**********************    TCP-related definition    **********************/
/***************************************************************************/
/** Definition of TCP reception point (only port number needs to be set) ***/
const T_TCP_CREP tcp_crep[] =
{
/* {attribute of reception point, {local IP address, local port number}} */
   { 0x0000, { 0, 1024 }}, /* TCP reception point ID: 1, port number: 1024 */
   { 0x0000, { 0, 1025 }}, /* TCP reception point ID: 2, port number: 1025 */
};
/* Total number of TCP reception points */
const H __tcprepn = sizeof(tcp_crep)/sizeof(T_TCP_CREP);
```

Figure 6.2 Definition of TCP Reception Points (Excerpt from tcpudp_config.c)

The TCP reception points are statically generated by using T_TCP_CREP structures (see Chapter 4). An example definition of TCP reception points is shown in **Figure 6.2**.

The current version of the software requires that of these three fields, only the local port number should be set.

The attribute is not supported, so that the value set for it has no effect.

The local IP address is obtained from the local IP address that is set in the variable tcpudp_env when initializing the T4, so that the value set for it in the configuration file has no effect.

T4 has some TCP reception points already specified in the configuration. But the required RAM size increases while using these reception points. These TCP reception points have the same reception port number with different reception points.

The variable __tcprepn does not need to be changed, because the total number of TCP reception points is automatically set.

When this definition is made, TCP-based connection from connection request wait with a specified port number is possible. Even when not using TCP, be sure to set at least one reception point.

6.3 Definition of TCP Communication End Points

```
/******************************************************************************/
/***********************     TCP-related definition     ***********************/
/******************************************************************************/
                  :
                  :|
/***  Definition of TCP communication end point
      (only receive window size needs to be set) ***/
const T_TCP_CCEP tcp_ccep[] =
{
    /* { attribute of TCP communication end point,
         top address of transmit window buffer, size of transmit window buffer,
         top address of receive window buffer, size of receive window buffer,
         address of callback routine, lan port number, keep-alive enable }
    */
    { 0, 0, 0, 0, 64, callback, 0, 0 }, /* TCP communication end point ID: 1 */
    { 0, 0, 0, 0, 64, callback, 0, 0 }, /* TCP communication end point ID: 2 */
};

/* Total number of TCP communication end points */
const H __tcpcepn = sizeof(tcp_ccep) / sizeof(T_TCP_CCEP);
```

Figure 6.3 Definition of the TCP Communication End Points (Excerpt from config_tcpudp.c)

The TCP communication end points are statically generated by using T_TCP_CCEP structures (see Chapter 4). An example definition of TCP communication end points is shown in **Figure 6.3**.

A definition of one TCP communication end point consists of eight fields, as shown below.

{ TCP communication endpoint attribute, top address of transmit window, size of transmit window, top address of receive window, size of receive window, address of callback routine (or callback function name), LAN port number, Keepalive function }

The current version of the software requires that of these eight fields, only the size of receive window, address of callback routine, and Keepalive function should be set. Please set the LAN port number to 0. The others are not supported, so that the values set for them have no effect, if non-blocking call is to be used, then the "callback" field needs to be specified.

Transmit window and that size is unsupported, so setting value is not read.

The top address of receive window is automatically assigned when initializing the T4, so that the value set for it in the configuration file has no effect.

If callback is to be used, then set the callback routine address in the field "callback". If callback is not to be used, set the "callback" field to "0".

If the TCP communication endpoint is to be used KeepAlive function, set the "keepalive_enable" field to 1. If the TCP communication endpoint is not to be used KeepAlive function, set the "keepalive_enable" field to 0.

T4 has some TCP communication end points already specified in the configuration. But the required RAM size increases while using these communication end points.

The variable __tcpcepn does not need to be changed, because the total number of TCP communication end points is automatically set.

When this definition is made, TCP-based establishment of a connection, data transmission/reception and closing of a connection are possible. Even when not using TCP, be sure to set at least one communication end point.

6.4 Definition of UDP Communication End Points

```
/******************************************************************************/
/***********************     UDP-related definition     **********************/
/******************************************************************************/
/***  Definition of UDP communication end point  ***/
const T_UDP_CCEP udp_ccep[] =
{
    { 0, { 0, 1365 }, callback, 0 }, /* UDP communication end point ID: 1, Port No: 1365, lan port number: 0 */
    { 0, { 0, 1366 }, callback, 0 }, /* UDP communication end point ID: 2, Port No: 1366, lan port number: 0 */
};

/* Total number of UDP communication end points */
const H __udpcepn = (sizeof(udp_ccep) / sizeof(T_UDP_CCEP));
```

Figure 6.4 Definition of UDP Communication End Points (Excerpt from config_tcpudp.c)

The UDP communication end points are statically generated by using T_UDP_CCEP structures (see Chapter 4). An example definition of UDP communication end point is shown in **Figure 6.4**.

A definition of one UDP communication end point consists of five fields, as shown below.

{ UDP communication endpoint attribute, {local IP address, local port number}, address of callback routine (or callback function name), LAN port number}

The current version of the software requires that of these five fields, the local port number and address of callback routine should be set. Please set the LAN port number to 0.

The local IP address is obtained from the local IP address that is set in the variable tcpudp_env when initializing the T4, so that the value set for it in the configuration file has no effect.

When using callback feature be sure to set the address of the callback function in the address of callback routine. When not using, be sure to set 0 in there.

The variable __udpcepn does not need to be changed, because the total number of UDP communication end points is automatically set.

When this definition is made, UDP-based data transmission or reception with a specified port number is possible. Even when not using UDP, be sure to set at least one communication end point.

6.5 Definition of IP Header-Related

```
/***  TTL for multicast transmission  ***/
const UB __multi_TTL = 1;
```

Figure 6.5 Definition of IP Header-Related (Excerpt from config_tcpudp.c)

For the variable __multi_TTL, set the TTL of the IP datagram to be sent to the multicast destination. The TTL is an element of the IP header and represent the number of routers the IP datagram can pass through.

If the IP datagram is sent to other than the multicast destination, the default value (= 80) is set in the TTL of the IP header.

Since the protocol for the IP datagram addressed to the multicast destination normally is UDP, this set value is used when sending the UDP data addressed to the multicast destination. However, because the parameters specified in APIs basically are not checked for correctness in the T4, if the remote IP address is set in the multicast address in a connection requesting API of TCP, the improper TCP packet addressed to the multicast destination will be sent.

Do not set the multicast address as the remote IP address of TCP.

6.6 Definition of TCP Options

```
/*****************************************************************************/
/*********************    TCP-related definition    *********************/
/*****************************************************************************/
                :
                :
/***  TCP MSS  ***/
const UH _tcp_mss = 64;     /* MAX:1460 bytes */

/***  2MSL wait time (unit:10ms)  ***/
const UH    _tcp_2msl = (1 * 60 * (1000 / 10));  /* 1 minute */

/***  Maximum value of retransmission timeout period (unit:10ms)  ***/
const UH    _tcp_rt_tmo_rst = (10 * 60 * (1000 / 10)); /* 10 minutes */

/***  Transmit for delay ack (ON=1/OFF=0) ***/
const UB    _tcp_dack = 1;

/***  Time to first transmit Keepalive packet (unit:10ms) ***/
const UW _tcp_keepalive_time = (2 * 3600 * (1000 / 10));

/***  Second Keepalive packets transmission interval (unit:10ms) ***/
const UW _tcp_keepalive_interval = (10 * (1000 / 10));

/***  Keepalive packet transmission count ***/
const UW _tcp_keepalive_count = 10;
```

Figure 6.6 Definition of TCP Options (Excerpt from config_tcpudp.c)

An example definition of TCP options is shown in **Figure 6.6**.

In the definition of TCP options, there are several items that need to be set. These include the default value of MSS (Maximum Segment Size), whether to send delayed ACK, a 2 MSL wait time and the maximum value of retransmission timeout period. Each of these options is detailed below.

6.6.1 MSS for TCP

For the variable _tcp_mss, set the maximum number of data (bytes) that can be transmitted in one TCP segment during TCP communication. This does not include the header size. It can be set in the range of 1 to 1,460. If any value other than that is set, the RFC default value of 536 is assumed.

This value is transmitted to the other node of communication as a TCP header option when a TCP connection is established. If this option is sent from the other node too, a smaller value of the two is used thereafter as the MSS in this connection. If this option is not sent from the other node, the default value 536 is assumed to have been sent from the other node.

However, if the receive window size at the communication end point used is smaller than _tcp_mss, the receive window size is transmitted as the MSS to the other node.

Furthermore, because the T4 does not support IP fragments, if the application is likely to use a communication path with small MTU to communicate, _tcp_mss must be set to a size smaller than the MTU.

6.6.2 2-MSL Wait Time

Set the length of time for which the program stays in a TIME_WAIT state during TCP state transition. The time must be set in 10-ms units. In the above example setting,

(1 * 60 * 1000 / 10) * 10 ms = 60,000 ms = 60 sec = 1 minute

6.6.3 Maximum Value of Retransmission Timeout Period

Set the length of time for TCP retransmission. When this set time is reached as a retransmission has been performed repeatedly after transmitting the first segment, a reset segment is transmitted to the other node of communication to close the connection.

The time must be set in 10-ms units. In the above example setting,

(10 * 60 * 1000 / 10) * 10 ms = 600,000 ms = 600 sec = 10 minutes

The interval time from the first transfer to the successive retransmissions attempted thereafter is exponentially extended by an exponential backoff algorithm. The maximum value for this interval time is 60 seconds. If the interval time in a given retransmission reaches 60 seconds, the subsequent retransmissions are performed at intervals of 60 seconds repeatedly.

6.6.4 Presence of TCP Multiple Transmit for Delay ACK

Specify "1" for variable _tcp_dack when T4 transmit the TCP multiple. Moreover, specify "0" for variable _tcp_dack when you invalidate this function. When communicating with an effective other party of the communication to be late ACK when the TCP multiple is transmitted, useless waiting time is not generated.

RFC provides for the rule (Nagle algorithm) to bring the data transmission together. There is a possibility of waiting for the receiving side for 0.5 seconds or less when the sending end transmits only one TCP packet to this. (0.2 seconds in case of Windows)

Wait for ACK after dividing data as follows by the transmission data length and MSS when making this function effective. Moreover, the transmission data length is adjusted according to the size of the window on the receiving side.

- The MSS byte is transmitted twice and the ACK waiting:
  (Remainder transmission data length ≧ MSS * 2)
- The MSS byte is transmitted once and (remainder transmission data length- MSS) byte is transmitted once and ACK waiting.
  (MSS * 2 > remainder transmission data length > MSS)
- (Remainder transmission data length- MSS) byte is transmitted once and 1byte is transmitted once and ACK waiting.
  MSS ≧ remainder transmission data length > 1
- 1 byte is transmitted once and the ACK waiting
  Remainder transmission data length = 1

Caution Data can transmit efficiently when the data length specified by transmission API specifies the twice MSS.
T4 comes to wait for ACK at each data transmission when the tcp_dack value is made "0".
Do not change the tcp_dack value with the TCP connection has established it.

6.6.5 Keepalive transmission start time

The variable _tcp_keepalive_time sets the time until the transmission of the KeepAlive packet is started with the TCP connection established.

The time must be set in 10-ms units. In the above example setting,

```
(2 * 3600 * (1000 / 10)) * 10 ms = 7,200,000 ms = 7,200 seconds = 2 hours
```

6.6.6 Keepalive retransmission interval time

Set the keepalive retransmission interval to the variable _tcp_keepalive_interval.

The time must be set in 10-ms units. In the above example setting,

```
(10 * (1000 / 10)) * 10 ms = 10,000 ms = 10 seconds
```

6.6.7 Keepalive retransmission send count

The variable _tcp_keepalive_count sets the number of keep-alive transmissions.

Default value is 10. If there is no response from the communication partner after sending keepalive 10 times, send a reset segment and discard the connection.

6.7 UDP Option Settings

```
    /*** Behavior of UDP zero checksum ***/
    const UB _udp_enable_zerochecksum = 0;   /* 0 = disable, other = enable */
```

Figure 6.7 UDP Option Settings (Excerpt from config_tcpudp.c)

6.7.1 UDP Zero Checksum Definition

This variable decides operation, when T4 receives UDP packets that have zero checksum.

In UDP specification, receiver does not confirm the UDP checksum value in UDP header. And, sender stored 0xffff value to the checksum field in UDP header when result of calculation of check sum is Zero. This case, receiver interprets checksum value as zero.

When sets the zero(default) to the UDP zero checksum variable, T4 detects zero checksum at the UDP header, T4 would not check the checksum value and treat as normal packet reception.

When sets the none-zero to the UDP zero checksum variable, T4 detects zero checksum at the UDP header, T4 would treat as abnormal packet reception, and annul the packet.

6.8 Definition of DHCP

```
/*****************************************************************************/
/**********************     TCP-related definition    **********************/
/*****************************************************************************/
/* Number of LAN port, Number of Serial port */
const UB _t4_channel_num = 2;
const UB _t4_dhcp_enable = 1;
const UB _t4_dhcp_ip_reply_arp_delay = 300;    /* unit:10ms */
```

Figure 6.8 Definition of DHCP (Excerpt from config_tcpudp.c)

When user selects _t4_dhcp_enable = 1, T4 will execute auto configuration using DHCP. When user selects _t4_dhcp_enable = 0, DHCP function will not be executed.

The Automatic Private IP Addressing (APIPA) address (RFC3927: 169.254.1.0-169.254.254/24) will be allocated in randomly when failed getting the IP address.

Caution Please do not use UDP port 68 for your communication endpoint settings. If you use this port, DHCP function will not operate in normally. DHCP client complies the RFC2131, RELEASE / INFORM / REBINDE functions are not implemented.

DHCP client will send the GratutousARP(Detect Collision Packet) after ACK received via DISCOVER-OFFER-REQUEST. DHCP client will send IP ReplyARP(Update ARP table packet), _t4_dhcp_ip_reply_arp_delay settings is for this delay time. Setting unit is 10ms. 100 means 1 second. The default value is 3.

6.9 Settings of System Callback

Notify the TCP/IP system status if user registers the callback function to the g_fp_user in config_tcpudp.c. User cannot use TCP/IP communication until received DHCP_EV_LEASE_IP / DHCP_EV_APIPA. And, DHCP_EV_PLEASE_RESET mean "Unsustainable" all communication endpoint because IP address is changed. Please call tcpudp_reset() when user detects these event code.

```
extern ER user_cb(UB channel, UW eventid, VP param);
const callback_from_system_t g_fp_user = user_cb;
```

Figure 6.9 Settings of system callback (Excerpt from config_tcpudp.c)

When user registers the user function to g_fp_user, DHCP system notifies the following event with callback routine.

```
/********************************
 *     callback eventID       *
 ********************************/
/* L2:ether layer */
#define ETHER_EV_LINK_OFF          (0u)
#define ETHER_EV_LINK_ON           (1u)
#define ETHER_EV_COLLISION_IP      (2u)
/* L7:dhcp */
#define DHCP_EV_LEASE_IP           (20u)
#define DHCP_EV_LEASE_OVER         (21u)
#define DHCP_EV_INIT               (22u)
#define DHCP_EV_INIT_REBOOT        (23u)
#define DHCP_EV_APIPA              (24u)
#define DHCP_EV_NAK                (25u)
#define DHCP_EV_FATAL_ERROR        (26u)
#define DHCP_EV_PLEASE_RESET       (27u)
```

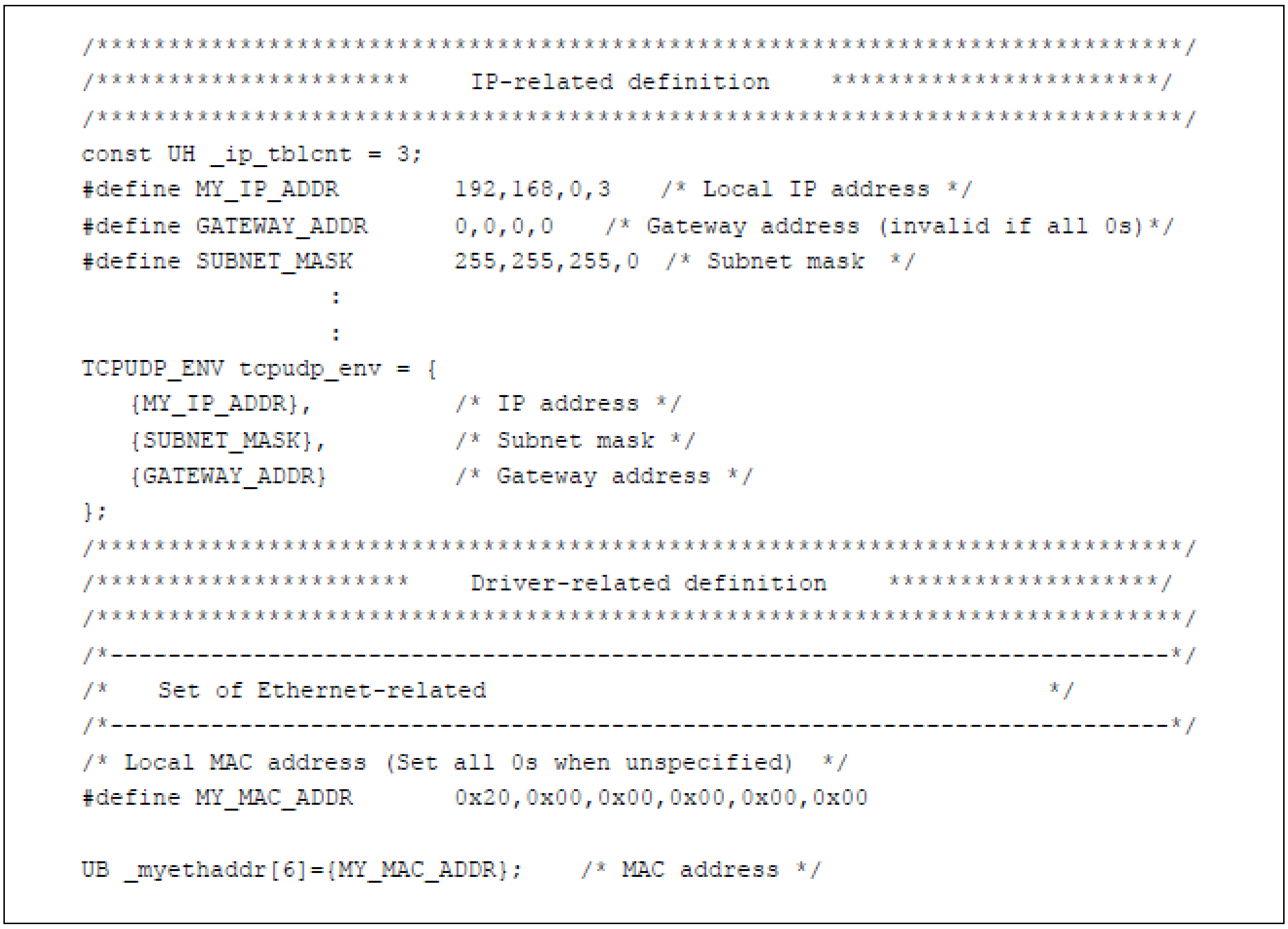
Figure 6.10 Event definition of system callback (Excerpt from r_t4_itcpip.h)

The 1st argument "channel" is LAN port number.

The 2nd argument "eventid" is event code defined in above.

The 3rd argument "param" is void pointer with DHCP information when event id is DHCP_EV_LEASE_IP / DHCP_EV_APIPA. This argument is implemented as void pointer for future. Please cast to the DHCP structure pointer.

6.10 Local Node Settings (Ethernet)



```
/******************************************************************************/
/**********************    IP-related definition    **********************/
/******************************************************************************/
const UH _ip_tblcnt = 3;
#define MY_IP_ADDR        192,168,0,3    /* Local IP address */
#define GATEWAY_ADDR      0,0,0,0    /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK       255,255,255,0  /* Subnet mask  */
                  :
                  :
TCPUDP_ENV tcpudp_env = {
    {MY_IP_ADDR},          /* IP address */
    {SUBNET_MASK},         /* Subnet mask */
    {GATEWAY_ADDR}         /* Gateway address */
};
/******************************************************************************/
/**********************    Driver-related definition    *******************/
/******************************************************************************/
/*--------------------------------------------------------------------------*/
/*    Set of Ethernet-related                                               */
/*--------------------------------------------------------------------------*/
/* Local MAC address (Set all 0s when unspecified)  */
#define MY_MAC_ADDR       0x20,0x00,0x00,0x00,0x00,0x00

UB _myethaddr[6]={MY_MAC_ADDR};    /* MAC address */
```

Figure 6.11 Local Node Settings when using Ethernet (Excerpt from config_tcpudp.c)

Set the IP address, subnet mask, gateway address and Ethernet address (MAC address) as the information associated with the local node when using Ethernet. An example definition is shown in **Figure 6.11**.

Local node settings consist of a TCPUDP_ENV structure including the three fields shown below (see Chapter 4) and an Ethernet address (MAC address).

tcpudp_env = { IP address, subnet mask, gateway address }

_myethaddr = { MAC address }

Definitions by define may also be used for each information, as shown in **Figure 6.11**. Separate the value for each defined information with a comma.

When not using the DHCP function, the IP address and subnet mask must always be set.

When using any gateway address, set the address that is used. When not using, set all 0s like {0, 0, 0, 0}. Only one gateway address can be set.

When specifying MAC address for the Ethernet controller in a program, set the MAC address that is to be specified. If the MAC address of the Ethernet controller is automatically set by EEPROM, etc., set all 0s like {0, 0, 0, 0, 0, 0}.

Furthermore, tcpudp_env and _myethaddr must be located in the RAM area.

The number of cash entries of ARP used by T4 is set to variable_ip_tblcnt. In T4, one cash entry per host who communicates is used.

The value more than "the number of hosts which may communicate simultaneously + 1" is recommended. When a value smaller than "the number of hosts which may communicate simultaneously" is set, the program may behave erratically. For example, when using TCP and UDP simultaneously and communicating with different host, seT4 or more values.

Moreover, the maintenance period of a cash entry is 10 minutes. So, when communicating with many hosts frequently in the meantime, if the value more than the number of hosts is set, communication efficiency will go up. For example, when communicating by UDP with four hosts and order, if the four or less number of cash entries is set, solution of ARP is needed each time.

6.11 Local Node Settings (PPP)

```
/*****************************************************************************/
/***************************     PPP definition    ***************************/
/*****************************************************************************/
UB ppp_mode = PPP_MODE_CLIENT;
//UB ppp_mode = PPP_MODE_SERVER;

const UH ppp_auth  = AUTH_PAP;
/* PAP = AUTH_PAP, CHAP with MD5 = AUTH_CHAP, NONE = AUTH_NON */

/*****************************************************************************/
/************    Driver-related definition ( for PPP server)    ************/
/*****************************************************************************/
UB user1_name[] = "abcde";
UB user1_passwd[] = "abc00";
UB user2_name[] = "xxx";
UB user2_passwd[] = "yyy";
UB *user_table[] = {                /* user authentication data table */
    user1_name, user1_passwd,
    user2_name, user2_passwd,
};
H __usern = sizeof(user_table)/sizeof(UB*)/2;       /* Number of user */
                              /* IP address that PPP server allocates */
UB client_ip_address[] = {192,168,0,100};
UB serverName[] = "T4 PPP SERVER";
UB serverName_len = sizeof(serverName)-1;

/*****************************************************************************/
/************    Driver-related definition ( for PPP client)    ************/
/*****************************************************************************/
UB user_name[6] =  "abcde";        /* user name */
UB user_passwd[6] = "abc00";       /* password  */

/* Dial up-related setting */
const UB peer_dial[] = "0,123";    /* Destination telephone number */
                                   /* Modem initialization command */
const UB at_commands[] = "ATW2S0=2&C0&D0&S0M0X3";
```

Figure 6.12 Local Node Settings when Using PPP (Excerpt from config_tcpudp.c)

6.11.1 Initial Value of Sequence Number

Set the IP address, PAP-related items and dial-up related items as the information associated with the local node when using PPP client. An example definition is shown in **Figure 6.12**.

To set the IP address, use the TCPUDP_ENV structure shown below (see Chapter 4) as in the case of Ethernet. For the subnet mask and gateway address, set all 0s.

tcpudp_env = { IP address, subnet mask, gateway address }

Definitions by define may also be used, as shown in **Figure 6.12**. Separate each specified value with a comma. The variable tcpudp_env must be located in the RAM area. Using T4 as a PPP client, IP addresses are requested to be assigned by the PPP server, set all 0s for the local IP address like {0, 0, 0, 0}, as shown in Fig. Set "PPP_MODE_CLIENT" to ppp_mode variable.

To request a specific IP address from the PPP server, set the IP address to request. However, if the PPP server does not approve the IP address requested by the PPP client, the local IP address is one that is assigned by the PPP server.

When IP addresses are assigned by the PPP server, the IP address in the variable tcpudp_env is updated to one that has been assigned by the PPP server.

Although the subnet mask originally has no effect in PPP, the T4 uses a subnet mask to check to see if the local and remote node belong to the same network. Therefore, if the subnet mask is all 0s, communication can be performed normally; otherwise, an error will be returned to the API, making it incapable of communication.

When PAP or CHAP are used as a means of authentication, the user name and password must be set. Therefore, set AUTH_PAP or AUTH_CHAP_MD5 in the variable ppp_auth, a user name in the variable user_name and a password in the variable user_passwd.

If not used for authentication, set AUTH_NON in the variable ppp_auth. In this case, the variables user_name and user_passwd do not need to be set values. (The set values, if any, are ignored.)

6.11.2 Local Node Settings (PPP Server)

This section explains difference of PPP client.

Set "PPP_MODE_SERVER" to ppp_mode variable. T4 allocates IP address that specified client_ip_address variable. (PPP client that connects in same time is 1).

When PAP or CHAP are used as a means of authentication, the user name and password must be set. Therefore, set AUTH_PAP or AUTH_CHAP in the variable ppp_auth, a user name and password in the variable user_table.

If not used for authentication, set AUTH_NON in the variable ppp_auth. In this case, the variable user_table does not need to be set values. (The set values, if any, are ignored.)

6.11.3 Common Settings for Modem (PPP Server/PPP Client)

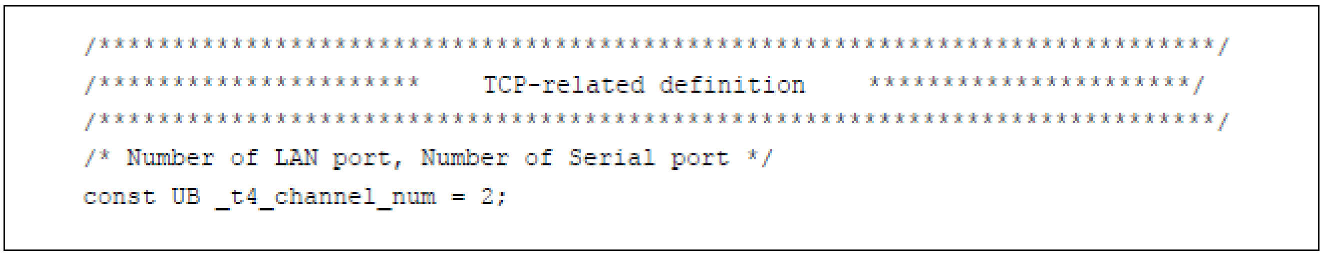
The dial-up related definition requires setting a destination telephone number and modem initialization commands. Therefore, set a destination telephone number in the variable peer_dial and modem initialization commands in the variable at_commands. Refer to the user's manual of your modem for details about the modem initialization command.

7. T4 Configuration File (Several LAN Ports)

Define TCP reception points, TCP communication end points, UDP communication end points, TCP options,IGMP,DHCP and local node settings (Ethernet or PPP). These definitions can be made in an application program providing that they are set before initializing the T4. However, all items must be set regardless of how they will be used.

Each of the following definitions is explained by using the configuration file config_tcpudp.c as an example.

7.1 LAN Port Number Definition



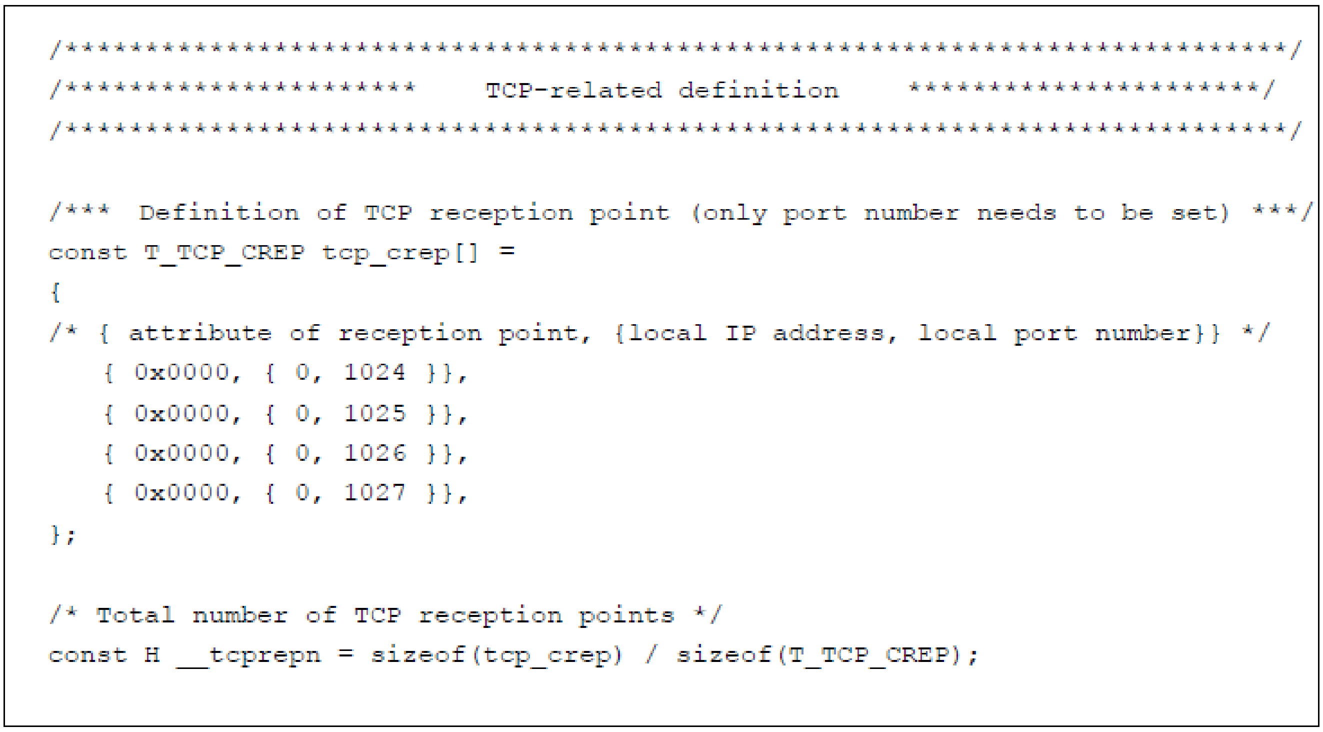
```
/****************************************************************************/
/**********************    TCP-related definition    ***********************/
/****************************************************************************/
/* Number of LAN port, Number of Serial port */
const UB _t4_channel_num = 2;
```

Figure 7.1 LAN Port Number Definition (From config_tcpudp.c)

Please set the LAN port numbers to the variable _t4_channel_num.

Do not set 0 value to the _t4_channel_num.

7.2 Definition of TCP Reception Points



```
/*****************************************************************************/
/**********************     TCP-related definition     **********************/
/*****************************************************************************/

/***  Definition of TCP reception point (only port number needs to be set) ***/
const T_TCP_CREP tcp_crep[] =
{
/* { attribute of reception point, {local IP address, local port number}} */
    { 0x0000, { 0, 1024 }},
    { 0x0000, { 0, 1025 }},
    { 0x0000, { 0, 1026 }},
    { 0x0000, { 0, 1027 }},
};

/* Total number of TCP reception points */
const H __tcprepn = sizeof(tcp_crep) / sizeof(T_TCP_CREP);
```

Figure 7.2 Definition of TCP Reception Points (Excerpt from tcpudp_config.c)

The TCP reception points are statically generated by using T_TCP_CREP structures (see Chapter 4). An example definition of TCP reception points is shown in **Figure 7.2**.

Please refer to the section 6.2 to know option variables.

7.3 Definition of TCP Communication End Points

```
/*****************************************************************************/
/**********************     TCP-related definition     **********************/
/*****************************************************************************/
                  :
                  :
/***  Definition of TCP communication end point
      (only receive window size needs to be set) ***/
const T_TCP_CCEP tcp_ccep[] =
{
    /* { attribute of TCP communication end point,
         top address of transmit window buffer, size of transmit window buffer,
         top address of receive window buffer, size of receive window buffer,
         address of callback routine, lan port number, keep-alive enable }
    */
    { 0, 0, 0, 0, 64, callback, 0, 0 }, /* TCP communication end point ID: 1 */
    { 0, 0, 0, 0, 64, callback, 0, 0 }, /* TCP communication end point ID: 2 */
    { 0, 0, 0, 0, 64, callback, 1, 0 }, /* TCP communication end point ID: 3 */
    { 0, 0, 0, 0, 64, callback, 1, 0 }, /* TCP communication end point ID: 4 */
};

/* Total number of TCP communication end points */
const H __tcpcepn = sizeof(tcp_ccep) / sizeof(T_TCP_CCEP);
```

Figure 7.3 Definition of the TCP Communication End Points (Excerpt from config_tcpudp.c)

The TCP communication end points are statically generated by using T_TCP_CCEP structures (see Chapter 4). An example definition of TCP communication end points is shown in **Figure 7.3**.

A definition of one TCP communication end point consists of eight fields, as shown below.

{ TCP communication endpoint attribute, top address of transmit window, size of transmit window, top address of receive window, size of receive window, address of callback routine (or callback function name), LAN port number, Keepalive function }

The current version of the software requires that of these eight fields, only the size of receive window, address of callback routine, and Keepalive funcion should be set. The others are not supported, so that the values set for them have no effect, If non-blocking call is to be used, then the "callback" field needs to be specified.

The top address of receive window is automatically assigned when initializing the T4, so that the value set for it in the configuration file has no effect.

If callback is to be used, then set the callback routine address in the field "callback". If callback is not to be used, set the "callback" field to 0.

Please set LAN port number to the field of the LAN port number. This value will through to the Ethernet Driver Interface. Please use this value in Ethernet driver.

This value’s maximum limit is "LAN port number -1"

If the TCP communication endpoint is to be used KeepAlive function, set the "keepalive_enable" field to 1. If the TCP communication endpoint is not to be used KeepAlive function, set the "keepalive_enable" field to 0.

T4 has some TCP communication end points already specified in the configuration. But the required RAM size increases while using these communication end points.

The variable __tcpcepn does not need to be changed, because the total number of TCP communication end points is automatically set.

When this definition is made, TCP-based establishment of a connection, data transmission/reception and closing of a connection are possible. Even when not using TCP, be sure to set at least one communication end point.

7.4 Definition of UDP Communication End Points

```
/*****************************************************************************/
/***********************     UDP-related definition     *********************/
/*****************************************************************************/
/***  Definition of UDP communication end point  ***/
const T_UDP_CCEP udp_ccep[] =
{
    { 0, { 0, 1365 }, callback, 0 }, /* UDP communication end point ID: 1, Port No: 1365, lan port number: 0 */
    { 0, { 0, 1366 }, callback, 0 }, /* UDP communication end point ID: 2, Port No: 1366, lan port number: 0 */
    { 0, { 0, 1365 }, callback, 1 }, /* UDP communication end point ID: 3, Port No: 1365, lan port number: 1 */
    { 0, { 0, 1366 }, callback, 1 }, /* UDP communication end point ID: 4, Port No: 1366, lan port number: 1 */
};

/* Total number of UDP communication end points */
const H __udpcepn = (sizeof(udp_ccep) / sizeof(T_UDP_CCEP));
```

Figure 7.4 Definition of UDP Communication End Points (Excerpt from config_tcpudp.c)

The UDP communication end points are statically generated by using T_UDP_CCEP structures (see Chapter 4). An example definition of UDP communication end point is shown in **Figure 7.4**.

A definition of one UDP communication end point consists of five fields, as shown below.

{UDP communication endpoint attribute, {local IP address, local port number}, address of callback routine (or callback function name), LAN port number}

The current version of the software requires that of these five fields, the local port number and address of callback routine and LAN port number should be set.

The local IP address is obtained from the local IP address that is set in the variable tcpudp_env when initializing the T4, so that the value set for it in the configuration file has no effect.

When using callback feature be sure to set the address of the callback function in the address of callback routine. When not using, be sure to set 0 in there.

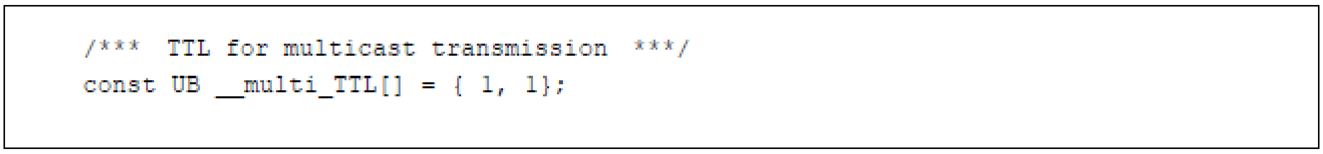
Please set LAN port number to the field of the UDP communication endpoint (LAN port number). This value will through to the Ethernet Driver Interface. Please use this value in Ethernet driver.

This value’s maximum limit is "LAN port number -1"

The variable __udpcepn does not need to be changed, because the total number of UDP communication end points is automatically set.

When this definition is made, UDP-based data transmission or reception with a specified port number is possible. Even when not using UDP, be sure to set at least one communication end point.

7.5 Definition of IP Header-Related



```
/***  TTL for multicast transmission  ***/
const UB __multi_TTL[] = { 1, 1};
```

Figure 7.5 Definition of IP Header-Related (Excerpt from config_tcpudp.c)

Please refer to the section 6.5 to know IP header definition.

7.6 Definition of TCP Options

```
/************************************************************************/
/***********************     TCP-related definition     ***********************/
/************************************************************************/
                  :
                  :
/***  TCP MSS  ***/
const UH _tcp_mss[] = {64, 64};     /* MAX:1460 bytes */

/***  2MSL wait time (unit:10ms)  ***/
const UH    _tcp_2msl[] =
{
    (1 * 60 * (1000 / 10)),  /* 1 minute */
    (1 * 60 * (1000 / 10)),  /* 1 minute */
};

/***  Maximum value of retransmission timeout period (unit:10ms)  ***/
const UH    _tcp_rt_tmo_rst[] =
{
    (10 * 60 * (1000 / 10)), /* 10 minutes */
    (10 * 60 * (1000 / 10)), /* 10 minutes */
};

/***  Transmit for delay ack (ON=1/OFF=0) ***/
const UB    _tcp_dack[] = {1, 1};

/***  Time to first transmit Keepalive packet (unit:10ms) ***/
const UW _tcp_keepalive_time[] =
{
    (2 * 3600 * (1000 / 10)),
    (2 * 3600 * (1000 / 10)),
};

/***  Second Keepalive packets transmission interval (unit:10ms) ***/
const UW _tcp_keepalive_interval[] =
{
    (10 * (1000 / 10)),
    (10 * (1000 / 10)),
};
/***  Keepalive packet transmission count ***/
const UW _tcp_keepalive_count[] = {10, 10};
```

Figure 7.6 Definition of TCP Options (Excerpt from config_tcpudp.c)

An example definition of TCP options is shown in **Figure 7.6**.

Please refer to the section 6.6 to know option settings.

7.7 Definition of UDP Options

```
/*** Behavior of UDP zero checksum ***/
const UB _udp_enable_zerochecksum[] = {0, 0}; /* 0 = disable, other = enable */
```

Figure 7.7 UDP Option Settings (Excerpt from config_tcpudp.c)

7.7.1 UDP Zero Checksum Definition

Please refer to the section 6.7.1 to know this settings.

7.8 Definition of DHCP

Please refer to the section 6.8 to know this settings.

7.9 Setting of system callback

Please refer to the section 6.9 to know this settings.

7.10 Local Node Settings (Ethernet)

```
/******************************************************************************/
/**********************    IP-related definition    ***********************/
/******************************************************************************/
const UH _ip_tblcnt[] = {3,3};
#define MY_IP_ADDR0       192,168,0,3  /* Local IP address */
#define GATEWAY_ADDR0     0,0,0,0      /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK0      255,255,255,0/* Subnet mask  */

#define MY_IP_ADDR1       192,168,0,10 /* Local IP address */
#define GATEWAY_ADDR1     0,0,0,0      /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK1      255,255,255,0/* Subnet mask  */
                 :
                 :
TCPUDP_ENV tcpudp_env[] =
{
   {{MY_IP_ADDR0},{SUBNET_MASK0},{GATEWAY_ADDR0}},
   {{MY_IP_ADDR1},{SUBNET_MASK1},{GATEWAY_ADDR1}}
};
/******************************************************************************/
/**********************    Driver-related definition    *******************/
/******************************************************************************/
/*----------------------------------------------------------------------------*/
/*    Set of Ethernet-related                                         */
/*----------------------------------------------------------------------------*/
/* Local MAC address (Set all 0s when unspecified)  */
#define MY_MAC_ADDR0      0x20,0x00,0x00,0x00,0x00,0x00
#define MY_MAC_ADDR1      0x40,0x00,0x00,0x00,0x00,0x00

UB _myethaddr[][6]={ {MY_MAC_ADDR0}, {MY_MAC_ADDR1}}; /* MAC address */
```

Figure 7.8 Local Node Settings when using Ethernet (Excerpt from config_tcpudp.c)

Set the IP address, subnet mask, gateway address and Ethernet address (MAC address) as the information associated with the local node when using Ethernet. An example definition is shown in **Figure 7.8**.

Local node settings consist of a TCPUDP_ENV structure including the three fields shown below (see Chapter 4) and an Ethernet address (MAC address).

tcpudp_env[] ={ { IP address, subnet mask, gateway address }, {}, … }

_myethaddr[][6] ={ { MAC address }, {}, … }

Definitions by define may also be used for each information, as shown in Fig. Separate the value for each defined information with a comma.

The IP address and subnet mask must always be set.

When using any gateway address, set the address that is used. When not using, set all 0s like {0, 0, 0, 0}. Only one gateway address can be set.

When specifying MAC address for the Ethernet controller in a program, set the MAC address that is to be specified. If the MAC address of the Ethernet controller is automatically set by EEPROM, etc., set all 0s like {0, 0, 0, 0, 0, 0}.

Furthermore, tcpudp_env and _myethaddr must be located in the RAM area.

The number of cash entries of ARP used by T4 is set to variable_ip_tblcnt. In T4, one cash entry per host who communicates is used.

The value more than "the number of hosts which may communicate simultaneously + 1" is recommended. When a value smaller than "the number of hosts which may communicate simultaneously" is set, the program may behave erratically. For example, when using TCP and UDP simultaneously and communicating with different host, seT4 or more values.

Moreover, the maintenance period of a cash entry is 10 minutes. So, when communicating with many hosts frequently in the meantime, if the value more than the number of hosts is set, communication efficiency will go up. For example, when communicating by UDP with four hosts and order, if the four or less number of cash entries is set, solution of ARP is needed each time.

7.11 Local Node Settings (PPP)

Please refer to the section 6.11 to know this settings.

8. T4 Library Functions

Table 8.1 lists the APIs that are supported by the T4. The TCP and UDP APIs each are compliant with ITRON TCP/IP Specification. The number of APIs that can executed at the same time in the T4 is one for TCP and UDP each. (excluding cancel API)

Table 8.1 List of T4 APIs

T4 APIs		C Language API
TCP	Wait for Connection Request (Passive Open)	tcp_acp_cep()
	Request Connection (Active Open)	tcp_con_cep()
	Terminate data transmission	tcp_sht_cep()
	Close communication end point	tcp_cls_cep ()
	Transmit data	tcp_snd_dat()
	Receive data	tcp_rcv_dat()
	TCP API cancel	tcp_can_cep()
UDP	Transmit packet	udp_snd_dat()
	Receive packet	udp_rcv_dat()
	UDP API cancel	udp_can_cep()
PPP	Open PPP	ppp_open()
	Close PPP	ppp_close()
	Refer PPP status	ppp_status()
	Request PPP process	ppp_api_req()
Initialization	Get work memory size used	tcpudp_get_ramsize()
	Open T4 library	tcpudp_open()
	T4 channel reset	tcpudp_reset()
Cyclic process	TCP/IP protocol process	_process_tcpip()
Termination processing	Close T4 library	tcpudp_close()
IGMP	Join multicast group	igmp_join_group()
	Leave multicast group	igmp_leave_group()

Caution Refer to Introduction Guide for the stack size of each APIs.

The following describes the data structures and macro definitions used in APIs. This is followed by a detailed description of each API, which is given in the format shown below.

< Format >

Shows a format in which the function is called.

< Explanation >

Shows the functionality and behavior of each API and the precautions to be observed when using API.

< Argument >

Shows the meaning of parameters to the API and limitations on acceptable values.

< Return Value >

Shows the type of value or error code returned by the API and the conditions under which an error occurs.

8.1 tcp_acp_cep

< Format >

ER tcp_acp_cep(ID cepid, ID repid, T_IPV4EP *p_dstaddr, TMO tmout)

< Explanation >

This API waits for a connection request for the TCP reception point repid. When a connection request is received, the API establishes a connection by using the TCP communication end point cepid and stores the IP address and port number of the remote node which requested a connection in the area indicated by the parameter p_dstaddr before returning that information to the task.

The API is in a wait state until a connection is established. A timeout period may be specified for this wait state. An error code E_TMOUT is returned if a connection cannot be established within the specified time.

When a connection is established, the return value E_OK is returned. If a connection is not established, one of the above error codes other than E_OK is returned depending on the cause of error. The cause of error is indicated in ( ) for each error code.

If the value "TMO_NBLK" is specified for "tmout, the API is a non-blocking call. This API does not have the status "waiting for API end". Upon normal termination of non-blocking call, this API returns value "E_WBLK". When connection is established, callback routine is called. Callback routine has three parameters viz.,"TCP communication end point number", "function code = TFN_TCP_ACP_CEP", "pointer of error code".

If anything else apart from "TMO_NBLK is specified in the callback routine, this API returns E_PAR.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
repid	Input	ID	Specify the ID of a TCP reception point ("1" to "30")
p_dstaddr	Output	T_IPV4EP	Get the remote IP address and port number Get the IP address and port number of the remote node that requested a connection
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until a connection is completed. The unit of time is 10 ms TMO_FEVR: Keeps waiting until a connection is completed (waiting forever) TMO_NBLK: non-blocking call

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (connection established)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (communication end point ID in use is specified)
E_TMOUT	Time out (time set in tmout expired)
E_WBLK	Non-blocking call is accepted
E_SYS	System error

8.2 tcp_con_cep

< Format >

ER tcp_con_cep(ID cepid, T_IPV4EP *p_myaddr, T_IPV4EP *p_dstaddr, TMO tmout)

< Explanation >

This API requests a connection to the IP address and port number of the other node to which to connect by using the TCP communication end point cepid, and keeps waiting until a connection is completed. Although the API remains waiting until a connection is established, a timeout period may be specified for this wait state.

An error code E_TMOUT is returned if a connection cannot be established within the specified time.

If the connection request is canceled due to time out or if the connection request is rejected for reasons that for example, a port number unsupported by the remote node is specified, the communication end point cepid returns to an "unused" state.

When this API is called, whether specified communication end point is using is checked first.

The local IP address that is set in the variable tcpudp_env is assumed for the local IP address.

If any value other than 0 is specified for the local port number, this value is set. On the other hand, if TCP_PORTANY is specified for the local port number, the API assigns a port number from within the range of numbers 49,152 to 65,535 in the T4.

If NADR is specified for the local IP address and port number p_myaddr, the API assigns the IP address that is set in the variable tcpudp_env for the local IP address, and for the local port number, it assigns a port number from within the range of numbers 49,152 to 65,535 in the T4.

When a connection is established, the return value E_OK is returned. If a connection is not established, one of the above error codes other than E_OK is returned depending on the cause of error. The cause of error is indicated in ( ) for each error code.

If the value "TMO_NBLK" is specified for tmout, the API is a non-blocking call. This API does not have the status "waiting for API end". Upon normal termination of non-blocking call, this API returns value "E_WBLK". When connection is established, callback routine is called. Callback routine has three parameters viz.,"TCP communication end point number", "function code= TFN_TCP_CON_CEP", "pointer of error code".

If anything else apart from "TMO_NBLK" is specified in the callback routine, this API returns E_PAR.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
p_myaddr	Input	T_IPV4EP	Specify the local IP address and port number
p_dstaddr	Input	T_IPV4EP	Specify the remote IP address and port number
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until a connection is completed. The unit of time is 10 ms TMO_FEVR: Keeps waiting until a connection is completed (waiting forever) TMO_NBLK: non-blocking call

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (connection established)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (communication end point ID in use is specified)
E_TMOUT	Time out (time set in tmout expired)
E_CLS	Failure of connection (reject the connection)
E_WBLK	Non-blocking call is accepted
E_SYS	System error

< Note >

Please specify IP address and port number to structure "T_IP_V4EP" to this function.

Example: connect to IP address 192.168.123.250, port 80.

< Using Example >

```
T_IPV4EP dst, src;

dst.ipaddr = 0xc0a87bfa;     // 192 = 0xc0, 168 = 0xa8, 123= 0x7b, 250 = 0xfa

dst.portno = 80;

tcp_con_cep(1, &src, &dst, TMO_NBLK);
```

8.3 tcp_sht_cep

< Format >

ER tcp_sht_cep(ID cepid)

< Explanation >

This API terminates data transmission as a preparatory procedure before closing a connection to the TCP communication end point cepid. More specifically, it sends FIN upon receiving ACK for the transmitted data.

Because this API only prepares for disconnection processing, it does not enter a wait state.

After this API is called, no data can be transmitted to the TCP communication end point cepid. If transmission is attempted, an error code E_OBJ is returned. Data can be received though.

When a data transmission terminating procedure is completed, a value E_OK is returned. If failed to terminate data transmission, one of the above error codes other than E_OK is returned depending on the cause of error. The cause of error is indicated in ( ) for each error code.

In the case, Specified communication end point number (cepid) does not have a callback routine specified in configuration, this API call is blocking call.

In the case, Specified communication end point number (cepid) has a callback routine specified in configuration, this API call is non-blocking call.

This API returns on transmission of a shutdown packet.

If anything else apart from TMO_NBLK is specified in the callback routine, this API returns E_NOSPT.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (data transmission terminated)
E_PAR	Parameter error (illegal cepid)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (communication end point ID in use is specified)
E_NOSPT	called in callback routine
E_WBLK	Non-blocking call is accepted
E_SYS	System error

8.4 tcp_cls_cep

< Format >

```
ER    tcp_cls_cep( ID cepid, TMO tmout )
```

< Explanation >

This API disconnect for the TCP communication end point cepid. After this API is called, the data transmitted from the other node is discarded.

If the connection closing processing is canceled due to time out, RST is sent from the TCP communication end point specified in this API to forcibly close the connection. In this case, because the connection is not closed normally, an error code E_TMOUT is returned.

The API is a non-blocking call. This API does not have the status "waiting for API end". Upon normal termination of non-blocking call, this API returns value "E_WBLK". When connection is closed, callback routine is called. Callback routine has three parameters viz.,"TCP communication end point number", "function code=TFN_TCP_CLS_CEP", "pointer of error code"

If anything else apart from "TMO_NBLK is specified in the callback routine, this API returns E_PAR.

This API waits until the communication end point enters an "unused" state regardless of whether disconnected normally or forcibly before it returns, the TCP communication end point cepid can be used immediately after returning from the API. The communication end point does not enter an "unused" state unless a connection is completely closed. According to TCP/IP standards, the communication end point remains in a TIME_WAIT state for some time which normally is several minutes until a connection is completely closed. The TIME_WAIT time, 2MSL, can be set in the T4 configuration file (see Chapter 6).

When a connection is closed normally, a value E_OK is returned. If forcibly closed, an error code E_TMOUT is returned. When one of these codes is returned, the specified TCP communication end point is in an "unused" state because a connection has been completely closed. If the TCP communication end point specified in this API does not connect, E_OBJ is returned.

There is the case continuing API process, when connection line has abnormally status after tcp_sht_cep().

If the connection line has not terminated normally after tcp_sht_cep(), it is recommended that the API tcp_cls_cep() be called with timeout parameter, or as a non-blocking call.

[Supplements]

In ITRON specifications, if data transmission is not completed yet, the API waits until the transmission finishes, and disconnects by sending FIN. In the T4, because multiple APIs cannot be executed at the same time, data transmission is already completed by the time this API is issued. Therefore, the API will not wait until FIN segment is sent.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until a connection is completed. The unit of time is 10 ms TMO_FEVR: Keeps waiting until a connection is completed (waiting forever) TMO_NBLK: non-blocking call

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (connection closed as normally)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (specified communication end point is unconnected)
E_TMOUT	Time out (time set in tmout expired / connection is forcibly closed)
E_WBLK	Non-blocking call is accepted
E_SYS	System error

8.5 tcp_snd_dat

< Format >

ER tcp_snd_dat(ID cepid, VP data, INT len, TMO tmout)

< Explanation >

This API transmits data from the TCP communication end point cepid. When transmitted normally, it returns the transmitted data size.

The T4 differs from ITRON TCP Specification in several points for reasons of increased RAM usage efficiency and transmission speed. In this library, the area in which transmit data is stored (hereafter called the user transmit buffer) serves as the transmit window defined in ITRON TCP Specification. Similarly, the transmit window size equals the transmit data length, and the size varies with the usage condition of this API.

In ITRON Specification, the task returns from the API when it finished copying data from the user transmit buffer to the transmit window. In this API, however, the task returns from the API when it received ACK for the data it transmitted. More specifically, if a TCP-level segment division occurs for reasons of the MSS or the receive window size of the other node, the task returns from the API only when ACK is received for all of the transmitted data, and not for the first divided data transmitted.

The API is a non-blocking call. This API does not have the status "waiting for API end". Upon normal termination of non-blocking call, this API returns value E_WBLK". When transmission is completed, callback routine is called. Callback routine has three parameters. "TCP communication end point number", "function code= TFN_TCP_SND_DAT", "pointer of error code".

If anything else apart from "TMO_NBLK is specified in the callback routine, this API returns E_PAR.

In ITRON specifications, the size of transmit data is determined by the size of free space in the transmit window. Therefore, the value returned by the API when it normally terminated does not always match the third parameter len. This requires caution when the API is ported to the protocol stack compliant with ITRON TCP/IP API specifications other than the T4.

The user should issue tcp_cls_cep() to disconnect the endpoint after tcp_can_cep() has been issued while tcp_snd_dat() is running.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
data	Input	VP	Specify the top address of transmit data The top address of the data to be sent by the user
len	Input	INT	Specify the length of transmit data Positive value
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until it finishes ending. The unit of time is 10 ms TMO_FEVR: Keeps waiting until it finishes sending (waiting forever) TMO_NBLK: non-blocking call

< Return Value >

Return Value	Explanation
Positive value	Terminated normally (transmitted data size in bytes)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (specified communication end point is unconnected)
E_TMOUT	Time out (time set in tmout expired / connection is forcibly closed)
E_CLS	Disconnect from peer (in case RST packet transmit/receive)
E_WBLK	Non-blocking call is accepted
E_SYS	System error

8.6 tcp_rcv_dat

< Format >

ER tcp_rcv_dat(ID cepid, VP data, INT len, TMO tmout)

< Explanation >

This API receives data from the TCP communication end point cepid.

The data transmitted from the other node of communication is stored in the receive window. This API copies data from the receive window to the user area indicated by the parameter data (hereafter referred to as "retrieving data") and then returns. If the receive window is empty, the API is kept waiting until data is received.

If the data size stored in the receive window is smaller than the data size len to be received, data is retrieved from the receive window until it is emptied and the retrieved data size is returned.

When the connection is closed normally by the remote node and all of data is retrieved from the receive window so that no data exists in it, value 0 is returned from the API.

When data has been received normally, the received data size is returned. If failed to receive, one of the above error codes is returned depending on the cause of error. The cause of error is indicated in ( ) for each error code.

When specified TMO_NBLK to the tmout argument, this API will be called as none-blocking call, and this API will not wait until processing end. This API returns value "E_WBLK". When reception is completed, callback routine is called. Callback routine has three parameters. "TCP communication end point number", "function code=TFN_TCP_RCV_DAT", "pointer of error code" Transmitted data size is stored into the error code.

If anything else apart from "TMO_NBLK" or "TMO_POL" is specified in the callback routine, this API returns E_PAR.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
data	Output	VP	Specify the top address of area in which received data will be stored The top address of the buffer reserved by the user for storing the received data
len	Input	INT	Maximum size in which to store the received data Positive value
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until it finishes receiving. The unit of time is 10 ms TMO_FEVR: Keeps waiting until it finishes receiving (waiting forever) TMO_NBLK: non-blocking call TMO_POL: polling

< Return Value >

Return Value	Explanation
Positive value	Terminated normally (received data size in bytes)
0	Data terminated (connection closed normally. All data is received until disconnected)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_OBJ	Object status error (specified communication end point is unconnected)
E_TMOUT	Time out (time set in tmout expired)
E_CLS	Connection closed and the receive window is empty (forcibly closed by transmission/reception RST)
E_WBLK	Non-blocking call is accepted
E_SYS	System error

8.7 tcp_can_cep

< Format >

ER tcp_can_cep(ID cepid, FN fncd)

< Explanation >

Before calling any other TCP API, this API is called for cancelling any TCP API called by a non-blocking call. When a non-blocking call is canceled, then the callback routine will be called. At that time an event code specified by fncd will be canceled, and error code address is pointer to error code "E_RLWAI".

The other API will be called after this API under the following conditions:

(1) The return value is E_OK when callback routine returns E_RLWAI

(2) The return value is E_OBJ when cancellation is failure. (nothing to cancel)

The user should issue tcp_cls_cep() to disconnect the endpoint after tcp_can_cep() has been issued while tcp_snd_dat() is running.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a TCP communication end point ("1" to "30")
fncd	Input	FN	Event code (the following are corresponding event codes) TFN_TCP_ACP_CEP, TFN_TCP_CON_CEP, TFN_TCP_CLS_CEP,TFN_TCP_SND_DAT, TFN_TCP_RCV_DAT,TFN_TCP_ALL

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (While cancelling the processing pending at the specified TCP communication end point.)
E_OBJ	Object status error (when API specified by fncd is not pending)
E_PAR	Incorrect parameter error (invalid “cepid”, " fncd" value)
E_NOSPT	None support (when this API is called inside the callback routine)
E_SYS	System error

8.8 udp_snd_dat

< Format >

ER udp_snd_dat(ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout)

< Explanation >

This API sends UDP datagram from the UDP communication end point cepid after specifying the remote IP address and the port number. The API returns when the UDP datagram is stored in the transmit buffer. The transmit buffer referred to here is not the one reserved internally in the T4, but is the internal transmit buffer of the Ethernet controller (for Ethernet) or the transmit buffer reserved by the serial driver (for PPP). When data is stored in the transmit buffer, the size of the transmitted data (= value of the fourth parameter len) is returned.

If the API failed to send, one of the above error codes is returned, with the cause of the error indicated in ( ).

In this API, a unicast address or multicast address or broadcast address or directed broadcast address can be specified for the remote IP address before sending data.

In the T4, the maximum value of the UDP datagram size that can be sent at a time (It’s referred to as N.) depends on the size of the transmit buffer. If M bytes of Ethernet / PPP frame data can be transmitted with the driver interface function lan_write() / ppp_write(), N is as follows.

N = M - Size of Frame header (F) - Minimum size of IP header (I) - Size of UDP header (U)

M <= 1514 (for Ethernet), M <= 1504 (for PPP)

F = 14 (for Ethernet), F = 4 (for PPP)

I = 20

U = 8

Because IP fragments are not supported in the T4, if the data larger than N bytes needs to be sent, the data must be divided to smaller than or equal to N bytes. If a data size larger than N bytes is specified, the behavior and the returned value of the API are indeterminate.

If TMO_FEVR is specified for the timeout specification tmout, the API is kept waiting until data is stored in the transmit buffer. If a positive value is specified for the timeout specification tmout, the API is kept waiting until the specified time is reached. If no data is stored in the transmit buffer by the specified time, the error code E_TMOUT is returned. If data is stored in the transmit buffer by the specified time, the size of the stored data is returned.

If TMO_NBLK is specified for the timeout specification tmout, a non-blocking call is assumed and this API is not kept waiting. When a transmit request is accepted, non-blocking call accepted E_WBLK is returned. In this case, a callback function is called at the time data is stored in the transmit buffer. The callback function has the UDP communication end point ID, function code TFN_UDP_SND_DAT, and the pointer to error code passed to it as arguments. The size of the stored data is indicated in the error code.

The following period, it is assumed that UDP transmit processing is underway. Therefore, do not rewrite the transmit data during this processing period.

- Within a period from when this API is blocking-called to when the API returns.

- For a while until the callback function (with function code TFN_UDP_SND_DAT) is called after the non-blocking call.

In the T4, except when udp_rcv_dat() is called by polling specification (TMO_POL) in the callback function called by the event code TEV_UDP_RCV_DAT, multiple UDP functions (udp_rcv_dat() and udp_snd_dat()) cannot be issued at the same time. If this API is issued while any UDP function is being processed, the error code E_QOVR is returned.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a UDP communication end point ("1" to "30")
*p_dstaddr	Input	T_IPV4EP	Specify the remote IP address and port number
data	Input	VP	Specify the top address of transmit data The top address of the data to be sent by the user
len	Input	INT	Specify the length of transmit data No fewer than 0, nor more than 1,472
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until it finishes sending The unit of time is 10 ms TMO_FEVR: Keeps waiting until it finishes sending (waiting forever) TMO_NBLK: non-blocking call

< Return Value >

Return Value	Explanation
0, positive value	Terminated normally (transmitted data size in bytes = value of len)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_TMOUT	Time out (time set in tmout expired)
E_WBLK	Non-blocking call is accepted
E_CLS	Connection Failed (Cannot exchange ARP protocol)
E_SYS	System error

8.9 udp_rcv_dat

< Format >

ER udp_rcv_dat(ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout)

< Explanation >

This API receives UDP datagram from the UDP communication end point cepid and gets the remote IP address and port number. When data has been received, the size of the received data is returned. If the API failed to receive, one of the above error codes is returned, with the cause of the error indicated in ( ).

In the T4, the maximum value of the UDP datagram size that can be received at a time (It's referred to as N.) depends on the size of the receive buffer of the driver. If M bytes of Ethernet / PPP frame data can be transmitted with the driver interface function lan_read() / ppp_read(), N is as follows.

N = M - Size of Frame header (F) - Minimum size of IP header (I) - Size of UDP header (U)

M <= 1514 (for Ethernet), M <= 1504 (for PPP)

F = 14 (for Ethernet), F = 4 (for PPP)

I = 20

U = 8

Because IP fragments are not supported in the T4, if the data larger than N bytes is received, the data is discarded.

If TMO_FEVR is specified for the timeout specification tmout, the API is kept waiting until UDP datagram is received. If a positive value is specified for the timeout specification tmout, the API is kept waiting until the specified time is reached. If no UDP datagram is received by the specified time, the error code E_TMOUT is returned.

If TMO_NBLK is specified for the timeout specification tmout, a non-blocking call is assumed and this API is not kept waiting. When a receive request is accepted, the non-blocking call-accepted E_WBLK is returned. In this case, a callback function is called at the time the received data is stored in the user's receive buffer. The callback function has the UDP communication end point ID, function code TFN_UDP_RCV_DAT, and the pointer to error code passed to it as arguments. The size of the received data is indicated in the error code, and the received data is stored in the user area data that is specified in the API in which the non-blocking call was invoked.

The following period, it is assumed that the API is waiting for the UDP data to receive and that UDP receive processing is underway.

- Within a period from when this API is blocking-called to when the API is exited.

- For a while until the callback function (with function code TEV_UDP_RCV_DAT) is invoked after the non-blocking call.

If UDP datagram is received while the API is waiting for the UDP data to receive, the received UDP datagram is copied to the received data area data beginning with the first part byte of data. If the size of the received data is equal to or less than the specified data size len, all amounts of the received data is copied. If the size of the received data is larger than the specified data size len, the received data is copied for up to an amount equal to the specified data size len and the rest is discarded. The error code returned in the former case is the size of the received data, and that in the latter case is the buffer overflow E_BOVR.

If UDP datagram is received while the API is not waiting for the UDP data to receive, a callback function for the event code TEV_UDP_RCV_DAT is called. The callback function has the UDP communication end point ID, event code TEV_UDP_RCV_DAT, and the pointer to error code (size of the received data) passed to it as arguments. The size of received data is indicated in the error code. Data can be read out only when this API that specifies polling TMO_POL for the timeout specification tmout is called in the callback function. If data is not read out by polling specification, the receive buffer of the driver is freed after exiting the callback function.

In the T4, except when udp_rcv_dat() is called by polling specification (TMO_POL) in the callback function called by the event code TEV_UDP_RCV_DAT, multiple UDP functions (udp_rcv_dat() and udp_snd_dat()) cannot be issued at the same time. If this API is issued while any UDP function is being processed, the error code E_QOVR is returned.

In this API, it is possible to receive UDP datagrams whose remote IP address is a unicast address or multicast addresses (224.0.0.0-239.255.255.255) or Broadcast address (255.255.255.255). And Directed broadcast address (Example: When 192.168.0.0/24, 192.168.0.255).

[Supplements]

When using a non-blocking call, note that depending on the call timing, a callback function for the event code TEV_UDP_RCV_DAT may be called before that. In this case, a callback function for the function code TFN_UDP_RCV_DAT is called upon receiving the next UDP data.

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a UDP communication end point ("1" to "30")
*p_dstaddr	Output	T_IPV4EP	Get the remote IP address and port number Get the IP address and port number of the remote node that transmitted data
data	Output	VP	Specify the top address of area in which to store the received data The top address of the buffer reserved by the user for storing the received data
len	Input	INT	Specify the maximum size in which to store the received data No fewer than 0, nor more than 1,472
tmout	Input	TMO	Specify a timeout. Positive value: Timeout period for waiting until it finishes receiving The unit of time is 10 ms TMO_FEVR: Keeps waiting until it finishes receiving (waiting forever) TMO_NBLK: Non-blocking call TMO_POL: Polling using in callback function for the event code TEV_UDP_RCV_DAT

< Return Value >

Return Value	Explanation
0, positive value	Terminated normally (received data size in bytes)
E_PAR	Incorrect parameter error (invalid “cepid”, "tmout" value)
E_QOVR	Queuing overflow (total transmit/receive count that can be queued is exceeded)
E_TMOUT	Time out (time set in tmout expired)
E_WBLK	Non-blocking call is accepted
E_BOVER	Buffer overflow (large data received that exceeds the received data storage area)
E_SYS	System error

8.10 udp_can_cep

< Format >

ER udp_can_cep(ID cepid, FN fncd)

< Explanation >

Before calling any other UDP API, this API is called for cancelling any UDP API called by a non-blocking call. When a non-blocking call is canceled, then the callback routine will be called. At that time an event code specified by fncd will be canceled, and error code address is pointer to error code "E_RLWAI".

The other API will be called after this API under the following conditions:

(1) The return value is E_OK when callback routine returns E_RLWAI

(2) The return value is E_OBJ when cancellation is failure. (nothing to cancel)

< Argument >

Argument	I/O	Type	Explanation
cepid	Input	ID	Specify the ID of a UDP communication end point ("1" to "30")
fncd	Input	FN	Event code (the following are corresponding event codes) TFN_UDP_SND_DAT, TEV_UDP_RCV_DAT, TFN_UDP_ALL

< Return Value >

Return Value	Explanation
E_OK	Terminated normally (While cancelling the processing pending at the specified UDP communication end point.)
E_OBJ	Object status error (when API specified by fncd is not pending)
E_PAR	Incorrect parameter error (invalid “cepid”, " fncd" value)
E_NOSPT	None support (when this API is called inside the callback routine)
E_SYS	System error

8.11 ppp_open

< Format >

ER ppp_open(void)

< Explanation >

To make a connection to the PPP server, establish a linkage with the PPP server, perform authentication using PAP and enter IPCP in an open state.

In PAP authentication, use the user name and password stored in the global variables user_name and user_passwd that are set in the T4 configuration file.

In IPCP, if the IP address in the variable tcpudp_env that is set in the T4 configuration file is not all 0s, assign a specific IP address to the PPP server. Or, if the IP address in the variable tcpudp_env is all 0s, request the automatic assignment of IP addresses by the PPP server. Even when a specific IP address is requested, there is no guarantee that the PPP server will approve it. The IP address finally approved by the PPP server is stored in the IP address of the variable tcpudp_env.

Call this API after the initialization of the T4 library function tcpudp_open().

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to initialize

8.12 ppp_close

< Format >

ER ppp_close(void)

< Explanation >

This API close connection with the PPP server.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to close

8.13 ppp_status

< Format >

UH ppp_status(void)

< Explanation >

The connection status of PPP is returned.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
0x0001(PS_DEAD)	Link disconnected
0x0002(PS_ESTABLISH)	LCP phase
0x0004(PS_AUTHENTICATE)	Authentication phase (PAP)
0x0008(PS_NETWORK)	NCP phase (IPCP)
0x0010(PS_NETOPEN)	Network established
0x0020(PS_TERMINATE)	Link being disconnected
Negative value	Failed to initialize

8.14 ppp_api_req

< Format >

ER ppp_api_req (UH type, void far *parblk, H tmout)

< Explanation >

To the PPP processing of the T2 library, this function issues a request to send the AT command, receive response code or wait for a designated time. Each request is specified by the request number type. The parameters required for the respective requests are specified by the pointer parblk. If the issued request is not completed by a designated time, a timeout error E_TMOUT (-85) is returned.

When a request is issued by this function, the following driver functions are executed in the PPP processing functions.

(1) If type is PPP_SNDCOMMAND

modem_write()

(2) If type is PPP_RCVRZLT

modem_read()

(3) If type is PPP_WAIT

None

The following shows the value returned by this function.

(1) If type is PPP_SNDCOMMAND

The returned value of modem_write()

(2) If type is PPP_RCVRZLT

The returned value of modem_read()

(3) If type is PPP_WAIT

E_TMOUT (-85)

Caution This function is one of the functions included in the T4 library. It does not need to be created by the user. Use it when creating API functions for the driver.

< Argument >

Argument	I/O	Type	Explanation
type	Input	UH	The request number issued to the PPP processing function. Set one of the following. Do not use any other values because they are reserved. 3:(PPP_API_SNDCOMMAND) Request to send AT command 4:(PPP_API_RCVRZLT) Request to receive response code 5:(PPP_API_WAIT) Request to wait for a designated time
parblk	Input	void *	The pointer to the parameter corresponding to the request. If type is PPP_API_WAIT, this pointer has no effect. (The format of the data indicated by the pointer parblk can be customized.)
tmout	Input	H	Specify a timeout. Positive value: Timeout period (in 10 ms units) -1: Waiting forever Any other values: Reserved (cannot be set)

< Return Value >

Return Value	Explanation
0 or more	Terminated normally.
E_TMOUT (-85)	Terminated due to timeout.
Negative value	Terminated due to other errors. (Depends on driver implementation)

8.15 tcpudp_get_ramsize

< Format >

W tcpudp_get_ramsize(void)

< Explanation >

This API gets the size of the work area used by the T4 (RAM size) which is returned as its return value. The work area is a memory area used for the TCP receive window and other purposes, which needs to be reserved in a program and initialized with parameters to the initialization function of API, tcpudp_open(). The top address of the work area is aligned to 4 byte boundaries.

EX) When this API's return value is 100, the work area used by the T4 is aligned as follows:

UW work[100/4];

This API can be used for the following purposes:

To reserve a work area using a static array

Although this API calculates the size of the work area used by the T4 according to the contents set in the TCP/IP configuration file, the user will have difficulty calculating it in advance. Therefore, when the contents of the T4 configuration file have been determined, rebuild the program and execute this API in a debugger to examine its returned value. Then reserve as much memory array for the work area as the size indicated by the returned value.

Note, however, that if the T4 configuration file is altered, the necessary size of the work area changes. In that case, recalculate the size of the work area by using this API. Furthermore, as a processing to check for errors, always be sure to call this API in the initialization processing and compare the returned value with the final size of the work area determined by the user. If they do not match, branch to error handling. That way, errors can be found at the debugging or test stages.

To reserve a work area from dynamic memory

Call this API in the initialization step of the application program to calculate the size of the work area. Reserve the calculated size of the work area from dynamic memory and pass the reserved area to the initialization function tcpudp_open() to have the work area initialized with it.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
positive value	The size of the work area used by the T4 (in bytes)

8.16 tcpudp_open

< Format >

ER tcpudp_open(UW *work)

< Explanation >

This API initializes the T4 library. In the library initialization processing, the API performs memory allocation and initialization of the internal managed area, as well as invokes the TCP/IP cyclic processing each task used by the library.

Specify the top address of the work area used by the T4 for work. The necessary size of the work area may be obtained from the returned value of tcpudp_get_ramsize().

Do not call tcpudp_open() from an interrupt handler or from a callback routine.

< Argument >

Argument	I/O	Type	Explanation
*work	Input	UW	Address of the work area used by the T4

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to initialize

8.17 tcpudp_reset

< Format >

ER tcpudp_reset(UB channel)

< Explanation >

This function resets the specified channel that used into T4 library.
This reset process initializes the memory for internal management.
User specifies the reset channel to the argument.
Please do not call this function in the interrupt function or callback function.

< Argument >

Argument	I/O	Type	Explanation
channel	Input	UB	Channel specification to be reset

< Return Value >

Return Value	Explanation
E_OK	Terminated normally

8.18 _process_tcpip

< Format >

```
void    _process_tcpip( void )
```

< Explanation >

This API executes the TCP/IP process.

This API calles some kind of driver interface functions, for example read/write interface function. etc.

This API's processing time would be changed by the implement of driver layer and communication status.

This API's stack size would be changed by the implement of driver layer.

Please call this API in repeatedly. 10ms cyclic calling is recommended. (please use timer interrupt)

And please call this API in send/receive complete interrupt.

T4 does the time management by using tcpudp_get_time().

As for the return value of tcpudp_get_time(), it is necessary to do the increment with 10msec. The following problems occur when the interval of the increment is not 10msec.

- The timeout specified in an API does not occur within the designated time.
- Retransmission operations are not performed at designated intervals.
- The zero window probe is not performed at designated intervals.
- The timeout period of 2MSL for disconnecting does not conform to the designated time.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
None	-

8.19 tcpudp_close

< Format >

ER tcpudp_close(void)

< Explanation >

This API closes the T4 library. In the library closing processing, the API terminates the TCP/IP cyclic processing used by the library.

Before calling this API, be sure to disconnect all of the TCP and UDP communication end points and enter an "unused" state.

The work area specified by the library open function tcpudp_open() is released after executing this API, so that the work area is free to use in the application program.

Do not call tcpudp_open() from an interrupt handler or from a callback routine.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to close

8.20 igmp_join_group

< Format >

UW igmp_join_group(UW *mCastAdr, UW RJ45port)

< Explanation >

This API sends an IGMP join report to register a group in the multicast table of the router. The router that corresponds to IGMP functions sends IGMP Query packet in repeatedly, and receive and process the IGMP Join Report packet.

This API sends IGMP Join Report packet that includes IGMPv2 field in first. When the router that processes the IGMP Join Report Packet can only process the IGMPv1 packet, this API send IGMP Join Report packet that includes IGMPv1 field in following sequence.

[host: send IGMPv2 join report] -> [router: receive IGMPv2 join report. not understood] ->

[router: send IGMPv1 query] -> [host: receive IGMPv1 query] -> [host: send IGMPv1 report] -> [router: detect host]

In the case of either an IGMPv2 or IGMPv1 join report, the internal version transitions automatically, according to the IGMP version of the IGMP Query received from the router.

Ensure that TCP communication endpoint initialization (tcpudp_open()) has finished and the status is ready to communicate before calling this API.

Within the class D address range (224.0.0.0 to 239.255.255.255), multicast addresses in the local network control block defined by IANA (224.0.0.x address range) cannot be registered as multicast group addresses.

The same multicast addresses may not be registered for the same RJ45 interface. (It is, however, possible to register the same multicast address for a different RJ45 interface because each RJ45 interface is treated as independent.)

5 multicast groups can be registered. The number of multicast groups that can be registered is specified by the following definition in igmp.h. If user needs changing this value, user needs to change T4 Library source code igmp.h and rebuild.

#define MULTI_CAST_MAX_GROUP_COUNT 5

< Argument >

Argument	I/O	Type	Explanation
*mCastAdr	Input	UW	Specifies the multicast address to be registered, within the range 224.0.1.0 to 239.255.255.255. Specifying an address in the 224.0.0.x range results in a registration error.
RJ45port	Input	UW	Specifies the RJ45 interface.

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
E_IGMP_MULTICAST_OUT_OF_RANGE	Outside multicast group address range
E_IGMP_MULTICAST_DOUBLE_ENTRY	Same multicast address already registered
E_IGMP_MULTICAST_MAX_ENTRY	Maximum registration count exceeded
E_IGMP_SYSTEM_ERROR	lan_write() returned an error

8.21 igmp_leave_group

< Format >

UW igmp_leave_group(UW *mCastAdr, UW RJ45port)

< Explanation >

This API performs the sending IGMP Leave Packet to leave a multicast group.

- If the router is IGMPv1.

Simply initializing the group registration work does not cause an IGMP leave report to be sent. The behavior of the router in cases where no leave report has been send by the T4 Library will be that the router will continue to distribute multicast packets until the time stipulated by RFC elapses.

- If the router is IGMPv2.

An IGMP leave report is sent. However, no IGMP leave report is sent if the other host belonging to the multicast group.

< Argument >

Argument	I/O	Type	Explanation
*mCastAdr	Input	UW	Specifies the multicast address to be left.
RJ45port	Input	UW	Specifies the RJ45 interface.

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
E_IGMP_MULTICAST_NOT_ENTRY	The multicast address has not been registered
E_IGMP_SYSTEM_ERROR	lan_write() returned an error

9. T4 Library Callback

9.1 callback_from_tcpudp

< Format >

ER callback_from_tcpudp(ID cepid, FN fncd, VP p_parblk)

< Explanation >

The TCP or UDP callback functions are called when the process of a non-blocking call has completed.

The processing of the TCP or UDP callback functions consist in each notification, one that is created by the user. The argument fncd indicates the type of notification. Kinds of events and the timing with which the callback function is called are shown in Appendix 3.

The return value that is for completed API is stored into the area p_parblk pointed. For example, user calls tcp_snd_dat() with TMO_NBLK (None-blocking call), and complete sending, the callback_from_tcpudp() function will be called with p_parblk includes sending data length.

Callback functions can be set to each communication endpoint defined in "config_tcpudp.c". Please refer to the 6.3 Definition of TCP communication endpoint, or 6.4 Definition of UDP communication endpoint.

< Argument >

Argument	I/O	Type	Explanation
cepid	Output	ID	This argument has the communication endpoint ID that was used in completed API
fncd	Output	FN	This argument has the function code that indicates completed API
p_parblk	Output	VP	This argument has the pointer that is stored return value for completed API

< Return Value >

Return Value	Explanation
0	Set always "0"

9.2 callback_from_system

< Format >

ER callback_from_system(UB channel, UW eventid, VP param)

< Explanation >

System callback function notifies the event code (about changing states, etc) detected from T4. User makes this callback function for these event code.

The argument eventid represents the kind of notification. The following list is callback timing of each event. The area of param notifies using type of DHCP structure pointer when DHCP_EV_LEASE_IP / DHCP_EV_APIPA event occurred. This structure includes IP address, Subnetmask, Gateway address, DNS1 address, DNS2 address, domain name, MAC address. The param will be notified as ZERO when excluding these event code.

Callback function can be set into the config_tcpudp.c. Please refer to the 6.9 Settings of System Callback for detail about settings.

Table 9.1 The kind of eventid

eventid	Value	Explanation
ETHER_EV_LINK_OFF	0	Disconnected ETHER Link Layer
ETHER_EV_LINK_ON	1	Connected ETHER Link Layer
ETHER_EV_COLLISION_IP	2	Detect IP address collision
DHCP_EV_LEASE_IP	20	Completed IP address configuration using DHCP
DHCP_EV_LEASE_OVER	21	Expired IP address lease time using DHCP
DHCP_EV_INIT	22	Starting DHCP sequence
DHCP_EV_INIT_REBOOT	23	Re-Starting DHCP sequence
DHCP_EV_APIPA	24	Setting the APIPA address
DHCP_EV_NAK	25	Received NAK from DHCP server
DHCP_EV_FATAL_ERROR	26	Fatal error occurred
DHCP_EV_PLEASE_RESET	27	Please T4 reset using tcpudp_reset()

< Argument >

Argument	I/O	Type	Explanation
channel	Output	UB	LAN port number
eventid	Output	UW	Kind of notification
param	Output	VP	Pointer for user to provide the parameter

< Return Value >

Return Value	Explanation
0	Set always "0"

10. Ethernet and PPP Driver API Specifications

The T4 driver APIs are listed in Table 10.1, Table 10.2, and Table 10.3. And Table 10.4 shows the relationship between the T4 library API and the T4 driver related API.

These APIs are RENESAS-original APIs developed independently of ITRON TCP/IP API Specification.

Table 10.1 Ethernet Driver APIs

Ethernet Driver API	C Language API	Remarks
Open Ethernet driver	ER lan_open(void)	APIs called from the user application
Close Ethernet driver	ER lan_close(void)	
Receive data	H lan_read(UB, B **)	APIs called internally from the library
Transmit data	H lan_write(UB, B *, H, B *, H)	
Reset Ethernet controller	void lan_reset(UB)	

Caution Prototype declarations for the APIs in Table 10.1 are included in r_t4_itcpip.h file.

Table 10.2 PPP Driver APIs

PPP Driver API	C Language API	Remarks
Open serial I/O	void sio_open(UB)	APIs called from the user application
Close serial I/O	void sio_close(void)	
Connect modem (T4 is PPP client)	ER modem_active_open(void)	
Connect modem (T4 is PPP server)	ER modem_passive_open(void)	
Disconnect modem	ER modem_close(void)	
Get PPP status	UH ppp_status(void)	
Receive PPP frame	H ppp_read(UB **)	APIs called internally from the library
Transmit PPP frame	H ppp_write(B *, H, B **, H *, H)	
Receive response code of modem	H modem_read(UB **)	
Transmit modem command	H modem_write(void far *)	
Get PPP driver status	UH ppp_drv_status(void)	
Wait for completion of PPP API	void ppp_api_slp(void)	
Cancel the wait state of the PPP API completion	void ppp_api_wup(void)	

Caution Prototype declarations for the APIs in Table 10.2 are included in r_t4_itcpip.h file.

Table 10.3 Common APIs for Ethernet/PPP Driver

Ethernet/PPP Driver API	C Language API	Remarks
Release receive buffer of driver	H rcv_buff_release(UB)	APIs called internally from the library
Wait for completion of TCP API	void tcp_api_slp(ID)	
Cancel the wait state of the TCP API completion	void tcp_api_wup(ID)	
Wait for completion of UDP API	void udp_api_slp(ID)	
Cancel the wait state of the UDP API completion	void udp_api_wup(ID)	
Control cyclic activation of TCP/IP processing function	void tcpudp_act_cyc (UB)	
Get time information	UH tcpudp_get_time(void)	
Enable Interrupt temporary	void ena_int(void)	
Disable Interrupt temporary	void dis_int(void)	
Report error	void report_error(UB, H, UB*)	
Get random number	void get_random_number(UB *, UW)	
Register the callback function pointer from link layer	void register_callback_linklayer (callback_from_system_t)	

Caution Prototype declarations for the APIs in Table 10.3 are included in r_t4_itcpip.h file.

Table 10.4 List of Ethernet / PPP driver API called by T4 library Functions

T4 Library Functions	Called Ethernet / PPP driver APIs by T4 Library Functions
tcp_acp_cep()	dis_int(), ena_int(), tcp_api_slp()
tcp_con_cep()	dis_int(), ena_int(), tcp_api_slp()
tcp_rcv_dat()	dis_int(), ena_int(), tcp_api_slp()
tcp_snd_dat()	dis_int(), ena_int(), tcp_api_slp()
tcp_sht_cep()	dis_int(), ena_int(), tcp_api_slp()
tcp_cls_cep()	dis_int(), ena_int(), tcp_api_slp()
tcp_can_cep()	dis_int(), ena_int()
udp_rcv_dat()	dis_int(), ena_int(), udp_api_slp()
udp_snd_dat()	dis_int(), ena_int(), udp_api_slp()
udp_can_cep()	dis_int(), ena_int()
tcpudp_get_ramsize()	-
tcpudp_open()	register_callback_linklayer(), tcpudp_act_cyc(), lan_check_link()
_process_tcpip()	tcp_api_wup(), udp_api_wup(), rcv_buff_release(), lan_write(), lan_read(), lan_reset(), tcpudp_get_time(), report_error(), get_random_number() callback_from_tcpudp() callback_from_system()
tcpudp_close()	tcpudp_act_cyc(), register_callback_linklayer()
tcpudp_reset()	dis_int(), ena_int()
igmp_join_group()	dis_int(), ena_int(), lan_write(), get_random_number()
igmp_leave_group()	dis_int(), ena_int(), lan_write()

The APIs listed in Table 10.1, Table 10.2, and Table 10.3 can be classified into the following three:

1. APIs that need to be called from an application program
2. APIs that are called from the T4 library and do not need to be called from an application (those in the shaded sections of the above table)
3. APIs that assist users in creating a driver API and do not need to be called from an application

This manual describes how to use the APIs classified into 1. above, i.e. those that need to be called from the application program.

Refer to another documents "Ethernet Driver Interface Specification" and "PPP Driver Interface Specification" for details about each API specification. Follow the API specifications as you create driver functions.

[Supplement]

Of the APIs listed in Table 10.2, the functions ppp_open(), ppp_close() and ppp_api_req(), which are used to control the PPP driver, have their entities incorporated in the T4 library. (For the stack size, refer to the Introduction Guide.)

In contrast, all of the other APIs are separated as Ethernet or PPP drivers from the T4 library, and the sample sources for these APIs each are supplied as a driver sample program.

10.1 lan_open

< Format >

ER lan_open(void)

< Explanation >

This API initializes the Ethernet controller and Ethernet driver. If the global variable _myethaddr set in the T4 configuration file is all 0s, the Ethernet address stored in ROM is set in the Ethernet controller and also copied to _myethaddr. If _myethaddr is not all 0s, the defined value is set in the Ethernet controller. Refer to Chapter 6 for details on how to set the global variable _myethaddr.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to initialize

10.2 lan_close

< Format >

ER lan_close(void)

< Explanation >

This API stops the Ethernet controller and terminates the Ethernet driver.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Terminated normally
Negative value	Failed to close

10.3 sio_open

< Format >

void sio_open(UB rate)

< Explanation >

This function initializes the serial I/O to be used, to make it ready to transmit and receive. The baud rate is passed to the parameter rate by a value (0 - 5) representing each baud rate. These values are defined by BR96 - BR1152 in the header file r_t4_itcpip.h.

< Argument >

Argument	I/O	Type	Explanation
rate	Input	UB	Sets the baud rate of serial I/O 0: (BR96) 9600bps 1: (BR192) 19200bps 2: (BR288) 28800bps 3: (BR384) 38400bps 4: (BR576) 57600bps 5: (BR1152) 115200bps

< Return Value >

Return Value	Explanation
None	-

10.4 sio_close

< Format >

void sio_close(void)

< Explanation >

This function disables the transmission/reception interrupts of the serial I/O used, to make the serial I/O incapable of transmission/reception operations.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
None	-

10.5 modem_active_open

< Format >

ER modem_active_open(void)

< Explanation >

This function initializes a modem and connects it to a phone line to establish a modem connection. In initializing a modem, it uses the user-specified initialization AT command at_commands to set up operation of the modem. Furthermore, in connecting to a phone line, it uses the modem to place a call to the destination telephone number peer_dial.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Successfully connected a modem
-1	Failed to connect a modem

10.6 modem_passive_open

< Format >

ER modem_passive_open(void)

< Explanation >

This function initializes a modem and connects it to a phone line to establish a modem connection. In initializing a modem, it uses the user-specified initialization AT command at_commands to set up operation of the modem. Furthermore, in connecting to a phone line, and waiting "incoming call" This function waits till incoming call.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Successfully connected a modem
-1	Failed to connect a modem

10.7 modem_close

< Format >

ER modem_close(void)

< Explanation >

This function enters the modem from communication mode into command mode and then disconnects the phone line.

< Argument >

Argument	I/O	Type	Explanation
None	None	None	-

< Return Value >

Return Value	Explanation
E_OK	Successfully disconnected a modem
-1	Failed to disconnect a modem

11. T4 Limitations and Usage Precautions

The T4 limitations and usage precautions are described in below.

1. The APIs tcp_get_buf(), tcp_snd_buf(), tcp_rcv_buf() and tcp_rel_buf(), are not supported.
2. The emergency data transmit and receive functions tcp_snd_oob() and tcp_rcv_oob() are not supported.
3. The option setting and acquisition functions tcp_set_opt(), tcp_get_opt(), udp_set_opt() and udp_get_opt() are not supported.
4. In TCP APIs, multicast address, broadcast address and loopback address cannot be specified for the remote IP address.
5. In UDP APIs, loopback address cannot be specified for the remote IP address.
6. Polling (TMO_POL) that is specified to be the UDP receive function udp_rcv_dat() is accepted in only the callback function called by the event code TEV_UDP_RCV_DAT. If polling (TMO_POL) is specified in other than the callback function, a parameter error (E_PAR) is returned.
7. The number of APIs that can be executed simultaneously in the T4 is one each for TCP and UDP. If multiple APIs are executed at the same time, the APIs return the parameter E_QOVR.
8. For TCP, the T4 only supports the MSS in header option. All options but MSS are ignored when receiving TCP segments.
9. For IP, the T4 does not support IP options and fragments. If the received IP datagram contains IP options or has been fragmented, the datagram is discarded.
10. For ICMP, the T4 only supports receiving an echo request and transmitting an echo response in return for that. If the received packet contains any other ICMP messages, the packet is discarded.
11. For PPP, the T4 does not support compression-related options (i.e., protocol field compression, address and control field compression and TCP/IP header compression). If a request is received that requires setting compression-related options, Configure-Rejects of setting is transmitted in response.
12. Before calling the function tcpudp_close(), be sure to close all of the communication end points and enter an "unused" state. If tcpudp_close () is called while communication is on, the program may behave erratically.
13. Please do not call tcpudp_close() and tcpudp_open() and tcpudp_reset() in the callback function.
14. Please allocate the T_IPV4EP structure variable that is specified by argument as the global variable.
15. Please start TCP/IP communication after DHCP_EV_LEASE_IP / DHCP_EV_APIPA when you use the DHCP function. When these event code will be occurred, TCP/IP communication is not available.
16. When API (with blocking call) returns E_SYS, this means "Unsustainable" communication endpoint because IP address has been changed.
17. When callback with DHCP_EV_PLEASE_RESET, this means "Unsustainable" communication endpoint because IP address has been changed. To recover the communication, please execute tcpudp_reset(). Please refer to the sample program.

Appendix A. Return Values of TCP APIs

A.1 Behavior of the API when the Line is Disconnected Unintentionally

If the line is disconnected as when cable has gotten out of place, the API is not notified of it. The behavior of the API (return value) is the same as when the other side of communication hung up or is rebooted becoming unable to communicate normally. For APIs where timeout period is set by an argument, failure can be detected by timeout.

The point at which a return value is returned to the API when the line has been disconnected is basically

(time1) when the timeout period that is positive value specified by an argument of the API is reached

(time2) when the maximum value of retransmission timeout time is reached while sending a TCP segment

If TMO_FEVR is specified for timeout in the API, (time2) applies for the timing that returns from the API. Otherwise, (time1) or (time2) whichever shorter applies. Therefore, if the time (time2) is shorter than the other, the API times out when the specified time in (time2) is reached and the return value is that of (time2).

A.2 Behavior of the API when the Line is Disconnected Unintentionally

For the cases (time1) and (time2) in A1 above, and for the case (R) where RST is received from the remote, the following describes the return value of each API and the status of communication end point.

Table A.1 Return value of Each TCP API and the Status of Communication End Point

API	Occurred event	Return value	Status of communication end point
tcp_acp_cep()	time1	E_TMOUT	Unused
	time2	Not return	Kept in LISTEN state
	R	Not return	Kept in LISTEN state
tcp_con_cep()	time1	E_TMOUT	Unused
	time2	E_CLS	Unused
	R	E_CLS	Unused
tcp_cls_cep()	time1	E_TMOUT	Unused
	time2	E_OK	Unused
	R	E_OK	Unused
tcp_snd_dat() *1	time1	E_TMOUT	ESTABLISHED
	time2	E_CLS	CLOSED
	R	E_CLS	CLOSED
tcp_rcv_dat()	time1	E_TMOUT	ESTABLISHED
	time2	Does not apply because no data is sent	-
	R	E_CLS	CLOSED *2

Caution1 Return value of tcp_snd_dat()
Even when FIN is received from the other side of communication, it is possible to send data until FIN is sent out from the local node. Therefore, tcp_snd_dat() does not return the return value E_CLS even when FIN is received from the remote.
If RST is received from the remote, the return value E_CLS is returned.

Caution2 Processing in tcp_rcv_dat() when RST is received.
If RST is received from the other side of communication, the return value E_CLS is returned after reading out all of the data in the receive window. While reading out data from the receive window, the return value is the size of the read data.

Supplement Return value of tcp_sht_cep().
Because this API does not enter to a pending, even when the line is disconnected, the same return value as usual is returned.

A.3 Return Value and Operation of API for TCP in Each State

The return value and the operation when API for TCP was called in the CLOSED state, are indicated in table A.2. The return value and the operation when API for TCP was called in the ESTABLISHED state, are indicated in table A.3.

The calling conditions of table A.2 and table A.3 are as follows.

1. Specifying TMO_FEVR in main process
2. Specifying TMO_NBLK in main process
3. Specifying TMO_FEVR in callback routine
4. Specifying TMO_NBLK in callback routine
5. Using blocking call in main process (no registration of callback routine)
6. Using non-blocking call in main process (registration of callback routine)
7. Using blocking call in callback routine (no registration of callback routine)
8. Using non-blocking call in callback routine (registration of callback routine)

1-4 is a condition of API with the time-out argument. 5-8 is a condition of API without the time-out argument.

Table A.2 Return Value and Operation of API for TCP in CLOSED

API called in CLOSED	Calling Conditions	Return Value	Operation
tcp_acp_cep()	1	E_OK	T4 returns from API after connection establishment
	2	E_WBLK	T4 calls callback routine with parameter "E_OK" after establishment
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OK" after establishment
tcp_con_cep()	1	E_OK	T4 returns from API after establishment
	2	E_WBLK	T4 calls callback routine with parameter "E_OK" after establishment
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OK" after establishment
tcp_snd_dat()	1	E_OBJ	T4 returns API in first T4 cycle of time after calling API.
	2	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
tcp_rcv_dat()	1	E_OBJ	T4 returns API in first T4 cycle of time after calling API.
	2	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
tcp_sht_cep()	5	E_OBJ	T4 returns API in first T4 cycle of time after calling API.
	6	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	7	E_NOSPT	T4 returns immediately after calling API.
	8	E_NOSPT	T4 returns immediately after calling API.
tcp_cls_cep()	1	E_OBJ	T4 returns API in first T4 cycle of time after calling API.
	2	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	3	E_OBJ	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
tcp_can_cep()	5	E_OBJ	T4 returns immediately after calling API.
	6	E_OK	T4 calls callback routine with parameter "E_RLWAI" after completing cancellation. (If specified API is not pending, T4 calls callback routine with parameter "E_OBJ".)
	7	E_NOSPT	T4 returns immediately after calling API.
	8	E_NOSPT	T4 returns immediately after calling API.

Table A.3 Return value and Operation of API for TCP in ESTABLISHED

API called in ESTABLISHED	Calling Conditions	Return Value	Operation
tcp_acp_cep()	1	E_OK	T4 returns API in first T4 cycle of time after calling API.
	2	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
tcp_con_cep()	1	E_OK	T4 returns API in first T4 cycle of time after calling API.
	2	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
tcp_snd_dat()	1	Positive value	T4 returns after completing transmission
	2	E_WBLK	T4 calls callback routine with positive number after completed transmitting.
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with positive number after completing transmission
tcp_rcv_dat()	1	Positive value	T4 returns after completing reception
	2	E_WBLK	T4 calls callback routine with positive number after completing reception
	3	E_PAR	T4 returns immediately after calling API.
	4	E_WBLK	T4 calls callback routine with positive number after completing reception
tcp_sht_cep()	5	E_OK	T4 returns API in first T4 cycle of time after calling API.
	6	E_WBLK	T4 calls callback routine with parameter "E_OBJ" in first T4 cycle of time after calling API.
	7	E_NOSPT	T4 returns immediately after calling API.
	8	E_NOSPT	T4 returns immediately after calling API.
tcp_cls_cep()	1	E_OK	T4 returns after normal closing.
	2	E_WBLK	T4 calls callback routine with parameter "E_OK" after normal closing.
	3	E_OK	T4 returns after normal closing.
	4	E_WBLK	T4 calls callback routine with parameter "E_OK" after normal closing.
tcp_can_cep()	5	E_OBJ	T4 returns immediately after calling API.
	6	E_OK	T4 calls callback routine with parameter "E_RLWAI" after completing cancellation. (If specified API is not pending, T4 calls callback routine with parameter "E_OBJ".)
	7	E_NOSPT	T4 returns immediately after calling API.
	8	E_NOSPT	T4 returns immediately after calling API.

Appendix B. T4 Software Structure

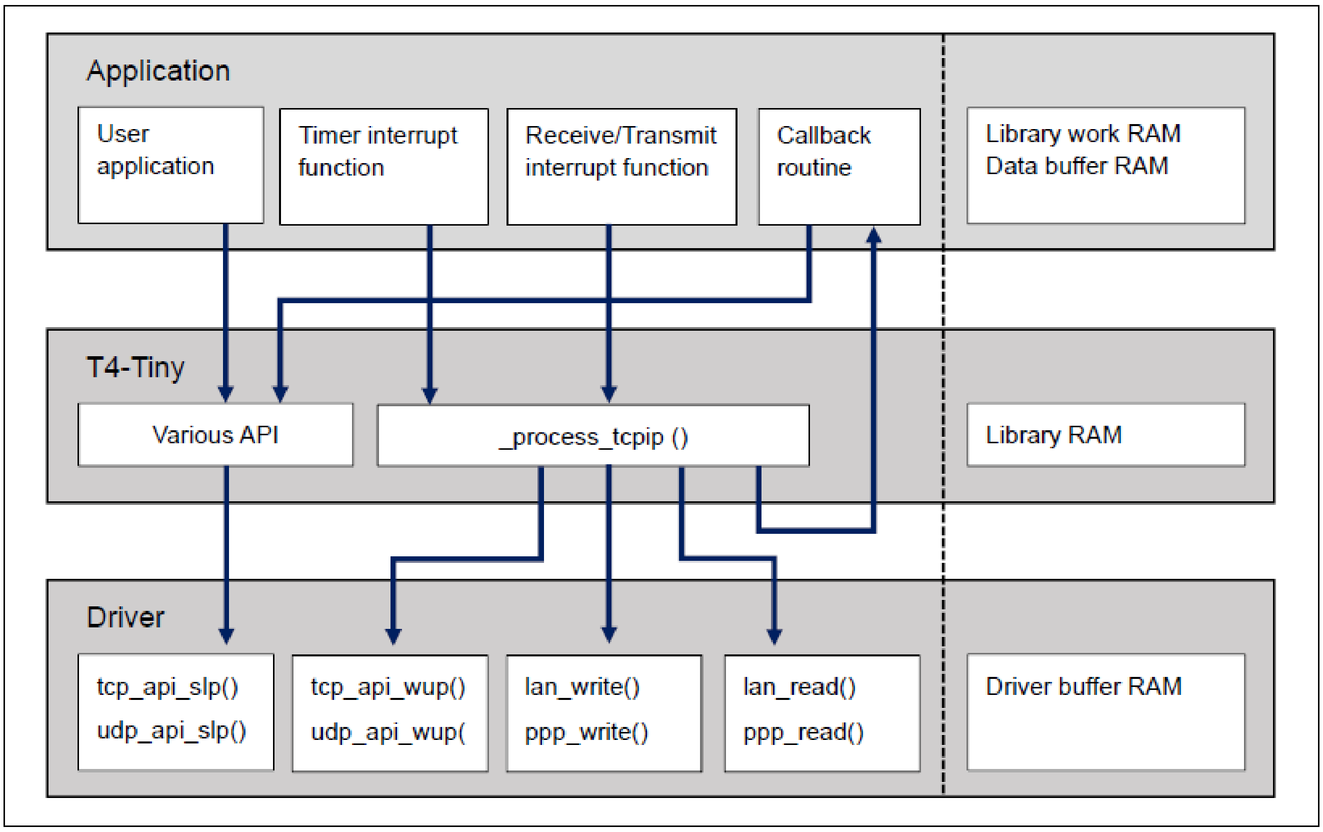


Figure B.1 T4 Software Structure

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct. 07, 2010	—	First Edition issued
1.01	Jan. 06, 2011	—	Change api_wup() prototype
1.02	Jan. 24, 2011	10	Change library Datatype list
1.03	Aug. 23, 2011	20	Add UDP option settings.
		44	Add user definition function "report_error()"
		54	Revised flow processing
1.04	Apr. 01, 2012	—	Change spec, api_wup() and api_slp()
		—	Change spec, modem_open()
		—	Change spec, _process_tcpip()
		—	Change spec, tcp_rcv_dat() can use TMO_POL
		—	Add PPP server / PPP client switch function
		—	Add PPP authentication, CHAP(MD5)
1.05	Jun. 21, 2013	10	Changed library Datatype list
		33	Changed spec, udp_snd_dat()
		36	Changed spec, udp_rcv_dat()
		43	Added information about callback routine.
		48	Changed T4 Limitations and Usage Precautions
1.06	Apr. 01, 2014	—	Supported Several LAN port
1.07	Aug. 07, 2015	—	Applied latest document template
1.08	Dec. 01, 2015	—	Added I/O information for each function’s arguments.
		—	Added IGMP functionalit.
		68	Added igmp_join_group().
		69	Added igmp_leave_group().
		79	Added get_random_number().
		—	Precautions updated for each API.
		48	Precaution added for when tcp_can_cep() is used with tcp_snd_dat().
		52	Precaution added for when tcp_can_cep() is used.
		65	Precaution added for when tcpudp_open() is used.
		67	Precaution added for when tcpudp_close() is used.
1.09	Nov. 30, 2016	—	Applied latest document template
		13	Added DHCP to the definition of Terms.
		15	Added DHCP to the outline of protocol.
		16	Added type of DHCP definition used by library.
		17	Changed the callback function name.
		21	Added IGMP, DHCP name.
		31	Added DHCP settings.
		32	Added system callback settings.
		33	Corrected explanation with “~settings is need for excluding DHCP”
		37	Added IGMP, DHCP name.

Rev.	Date	Description	
		Page	Summary
1.09	Nov. 30, 2016	44	Added DHCP definition.
		44	Added system callback definition.
		46	Added Local Nodes Settings(PPP)
		79	Added Section 9, T4 library callback
		103	Added Limitation about DHCP to (14)~(16)
		19	Added error code "E_SYS" for APIs.
		—	Added error code "E_SYS" and API return value for section 8.1-8.6, 8,8, 8.9.
		28	Deleted the explanation of Initial Value of Sequence Number.
		77,79	IGMP API prototype mistyping correction.
		91-102	Replaced sample programs with DHCP compatible echo_server.The UDP flowchart was deleted.
1.10	Dec. 10, 2018	—	Removed 11.Sample Program and moved it to Introduction Guide(r20an0051).
1.11	Jun. 20, 2019	14	Change spec, Table 2.1 Directory Structure for the T4
		18	Change spec, Table 4.4 The T_TCP_CCEP structure is TCP communication end point
		19	Change spec, Table 4.5 The T_UDP_CCEP structure is UDP communication end point
		24,40	Changed Definition of TCP Communication End Points
		25,41	Changed Definition of UDP Communication End Points
		29,31, 45	Added Keep-alive configuration settings
		80	Changed Table 10.4 List of Ethernet / PPP driver API called by T4 library Functions

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 5.2

TCP/IP for Embedded system M3S-T4-Tiny

Renesas Electronics Corporation

R20UW0031EJ0111