HITACHI

S10mini

S10mini
HARDWARE MANUAL

OPTION

ET.NET

SME-1-103 (E)

S10mini
HARDWARE MANUAL

OPTION
ET.NET

First Edition, November 1998, SME-1-103(A) (out of print)
Second Edition, May 2000, SME-1-103(B) (out of print)
Third Edition, November 2001, SME-1-103(C) (out of print)
Fourth Edition, October 2008, SME-1-103(D) (out of print)
Fifth Edition, February 2009, SME-1-103(E)

Printed in Japan.

BI-KB-SK<IC-IC> (FL-MW20, AI8.0, PS)

SAFETY PRECAUTIONS

Be sure to read this manual and all other attached documents carefully before installing, operating inspecting or conducting maintenance on this unit. Always use this unit properly. Be sure to carefully read the information about the device, the safety information and precautions before using this unit. Be sure that the person(s) responsible for maintenance receives and understands this manual completely.

This manual divides the safety precautions into DANGERs and CAUTIONs.

DANGER : Failure to observe these warnings may result in death or serious injury.

CAUTION : Failure to observe these cautions may result in injury or property damage.

Failure to observe any **CAUTION** may lead to serious consequences.

All of these DANGERs and CAUTIONs provide very important precautions and should always be observed.
Additional safety symbols representing a prohibition or a requirement are as follows:

: Prohibition. For example, “Do not disassemble” is represented by:

: Requirement. For example, if a ground is required, the following will be shown:

DANGER

- Devise an emergency stop circuit, interlock circuit, and other similar circuits outside the programmable controller. Disregarding this rule may result in damage to the equipment or cause an accident if the programmable controller fails.
- Keep it in mind that this hardware unit operates on a high voltage. If the user touches a high-voltage terminal inadvertently during connection or disconnection of this hardware unit or its cable, he or she may suffer from an electric shock. Also, this hardware unit may be damaged due to a short circuit or noise. Be sure to switch off the hardware unit before connecting or disconnecting it or its cable.

CAUTION

- This hardware unit may fail if the ambient temperature is too high. The hardware unit may also malfunction due to interference by electromagnetic waves from adjacent hardware. To dissipate heat and reduce electromagnetic interference, provide the specified mount of space between the cubicle and this hardware unit and between the hardware unit and other ones.
- After installing this hardware unit, measure temperatures near the in-cubicle controller and the mount base during operation, and check whether the measurements are within the limits. If the specified amount of space cannot be provided or the measured temperature is too high, use a cooling fan.
- At an extremely high temperature, this hardware unit may fail. Secure the mount base to a vertical surface. If the mount base is secured horizontally, heat does not dissipate efficiently, resulting in an extremely high temperature. This may further cause the hardware unit to fail or its parts to deteriorate.
- This hardware unit may be damaged due to static electricity. Ground yourself before setting switches or connecting or disconnecting cables or connectors with the hardware unit.
- This hardware unit may be damaged during its installation or removal unless the following rules are observed:
  - Check that the connector pins are not damaged (bent or broken), are aligned straight and are free from dust.
  - Move the hardware unit along an imaginary vertical surface to the face of the mount base. If the product is inserted or removed slantwise from the connector on the mount base, connector pins may be bent.

REQUIREMENT

An electric shock may lead to a death or burn. Noise may cause the system to malfunction. Ground the line ground (LG), frame ground (FG), and shield (SHD) terminals, as described below.

- Electrically insulate the mount base from the cubicle. To assure this, do not remove the insulating sheet from the mount base.
- Ground the LG and FG terminals separately to prevent mutual interference. The LG terminal is grounded to prevent intrusion of power line noise, while FG and SHD terminals are grounded to suppress intrusion of line noise into external interfaces for remote I/O modules, interface modules, and other modules.
- Connect the FG terminal on each module to the FG terminal on the mount base. Note, however, that the FG terminal for each remote I/O line or JPCN-1 line must be connected separately to a single place on the terminating side.

REQUIREMENT

Excessive accumulation of heat in the cubicle may cause a fire or hardware failure. When the temperature in the cubicle reaches 48℃ or higher, the maximum output current of the power supply module is limited. At 55℃, for instance, it is limited to 5.85 A. Where this is very likely, install a cooling fan in the cubicle or reduce the number of modules installed therein.

PROHIBITION

If a part in a module is damaged, do not replace the part, but replace the faulty module in its entirety, except when the part is the battery for the CPU.

WARRANTY AND SERVICING

Unless a special warranty contract has been arranged, the following warranty is applicable to this product.

1. Warranty period and scope
   Warranty period
   The warranty period for this product is for one year after the product has been delivered to the specified delivery site.

   Scope
   If a malfunction should occur during the above warranty period while using this product under normal product specification conditions as described in this manual, please deliver the malfunctioning part of the product to the dealer or Hitachi Engineering & Services Co., Ltd. The malfunctioning part will be replaced or repaired free of charge. If the malfunctioning is shipped, however, the shipment charge and packaging expenses must be paid for by the customer.

   This warranty is not applicable if any of the following are true.

   - The malfunction was caused by handling or use of the product in a manner not specified in the product specifications.
   - The malfunction was caused by a unit other than that which was delivered.
   - The malfunction was caused by modifications or repairs made by a vendor other than the vendor that delivered the unit.
   - The malfunction was caused by a relay or other consumable which has passed the end of its service life.
   - The malfunction was caused by a disaster, natural or otherwise, for which the vendor is not responsible.

   The warranty mentioned here means the warranty for the individual product that is delivered. Therefore, we cannot be held responsible for any losses or lost profits that result from the operation of this product or from malfunctions of this product. This warranty is valid only in Japan and is not transferable.

2. Range of services
   The price of the delivered product does not include on-site servicing fees by engineers. Extra fees will be charged for the following:

   - Instruction for installation and adjustments, and witnessing trial operations.
   - Inspections, maintenance and adjustments.
   - Technical instruction, technical training and training schools.
   - Examinations and repairs after the warranty period is concluded.
   - Even if the warranty is valid, examination of malfunctions that are caused by reasons outside the above warranty scope.

This manual provides information for the following hardware product:

<Hardware product>
ET.NET (LQE020)

<Changes added to this manual>

Description of added changes	Page
The following information is newly added: setting the module no. setting switch (or, simply, MODU No. switch) in 4- or 5-position may result in a route information setting error.	3-2, 7-15

In addition to the above changes, all the unclear descriptions and typographical errors found are also corrected without prior notice.

Revision record

Revision No.	Revision Record (revision details and reason for revision)	Month, Year	Remarks
A	First Edition	November 1998	
D	Supplementary, “Replacing or adding on the module” is newly added.	October 2008	
E	The following information is newly added: setting the module no. setting switch (or, simply, MODU No. switch) in 4- or 5-position may result in a route information setting error.	February 2009	

PREFACE

We greatly appreciate your making use of the CPU option ET.NET module.
This hardware manual on the option ET.NET describes how to handle the ET.NET module. Read this manual carefully to use the module properly.

Two specifications are available for S10mini series products: standard specifications and environmental resistance specifications.
The products with the environmental resistance specifications have thicker plating and more strengthened coating than those with standard specifications.
The model name of the products with the environmental resistance specifications have "-Z" after those with standard specifications.
Example: Standard specifications: LQE020
Environmental resistance specifications: LQE020-Z
The manuals for standard specifications and environmental resistance specifications are commonly used. The module types indicated in the manuals are those with standard specifications.
When you use a product with environmental resistance specifications, follow these manuals for proper use.

When S10mini and Windows® programming tools are connected through an ET.NET module, up to four programming tools (ladder drawing or HI-FLOW system) can be connected to S10mini at a time under the following conditions.
Note that, under the conditions other than these, only one programming tool can be connected.

- The revision No. of the ET.NET module is E or later. (Check that the seal attached on the upper end of the case is E or later or the CPU indicator display is ETM 3.1 or later.)
- The version or revision No. of a ladder drawing system or HI-FLOW system is 07-00 or later. (However, in case of HI-FLOW, a HI-FLOW system is only one unit and others are HI-FLOW for monitors.)
- The ET.NET module (LQE020) of revision "F" (having the "H" or later label on the lower left of the casing or whose CPU indicator indicates "ETM 3.2" or "ETS 3.2") updates the content (program data) of flash memory at intervals of about 1 month to increase the reliability of the program data as the mask of the flash memory is changed. For this updating, the socket handler is made to wait for about 3 seconds.
  If the reception wait time of tcp_receive() or udp_receive() is set smaller than 3 seconds, a timeout error may occur. In this case, retry.

<Note for storage capacity calculations>

- Memory capacities and requirements, file sizes and storage requirements, etc. must be calculated according to the formula 2^n. The following examples show the results of such calculations by 2^n (to the right of the equals signs).
  1 KB (kilobyte) = 1,024 bytes
  1 MB (megabyte) = 1,048,576 bytes
  1 GB (gigabyte) = 1,073,741,824 bytes
- As for disk capacities, they must be calculated using the formula 10^n. Listed below are the results of calculating the above example capacities using 10^n in place of 2^n.
  1 KB (kilobyte) = 1,000 bytes
  1 MB (megabyte) = $1{,}000^2$ bytes
  1 GB (gigabyte) = $1{,}000^3$ bytes

CONTENTS

1 BEFORE USE 1-1
1.1 CPU Mount Base 1-2
1.2 Mounting Optional Modules 1-2
1.3 Ground Wiring 1-4

2 SPECIFICATIONS 2-1
2.1 Usage 2-2
2.2 Specifications 2-2
2.2.1 System specifications 2-2
2.2.2 Line specifications 2-2

3 NAMES AND FUNCTIONS OF EACH PART AND CABLING 3-1
3.1 Names and Functions of Each Part 3-2
3.2 Cabling 3-4

4 USER GUIDE 4-1
4.1 System Configuration of 10BASE-5 4-2
4.2 10BASE-T System Configuration 4-8
4.3 Example of System Configuration with S10mini 4-10
4.4 System Definition Information 4-11
4.4.1 Physical address 4-11
4.4.2 IP address 4-11
4.4.3 Subnetwork mask 4-13
4.4.4 Route information 4-13
4.5 Software Configuration of ET.NET 4-16
4.6 ET.NET System Programs 4-17
4.6.1 Socket handler 4-17
4.6.2 Socket driver 4-17
4.6.3 TCP program 4-17
4.6.4 UDP program 4-18
4.6.5 IP program 4-18
4.6.6 Driver 4-18
4.7 User-created Program 4-19
4.7.1 User program 4-19
4.8 Socket Handler 4-20

4.8.1 Socket handler function list 4-21
4.9 Examples of Socket Handler Issuance Procedure 4-55
4.9.1 Example of using TCP/IP program 4-55
4.9.2 Example of using UDP/IP program 4-56

5 PROGRAM EXAMPLES 5-1
5.1 Example of Programs for Communication between CPUs by Socket Handler 5-2
5.1.1 System configuration 5-2
5.1.2 Program structure 5-3
5.1.3 Flowchart of program at CPU01 5-5
5.1.4 Example of C language program at CPU01 5-7
5.1.5 Flowchart of program at CPU02 5-9
5.1.6 Example of C language program at CPU02 5-10
5.2 Example of Programs for Continuous Communication between CPUs by Socket Handler 5-12
5.2.1 System configuration 5-12
5.2.2 Program structure 5-13
5.2.3 Flowchart of program at CPU01 5-15
5.2.4 Example of C language program at CPU01 5-17
5.2.5 Flowchart of program at CPU02 5-19
5.2.6 Example of C language program at CPU02 5-21

6 OPERATION 6-1
6.1 Start-up Procedure 6-2

7 MAINTENANCE 7-1
7.1 Maintenance Inspection 7-2
7.2 Troubleshooting 7-3
7.2.1 Procedure 7-3
7.2.2 Before suspecting a failure 7-4
7.3 Errors and Actions To Be Taken 7-6
7.3.1 CPU LED display messages 7-6
7.3.2 Hardware errors 7-7
7.3.3 Error codes from the socket handler 7-11
7.3.4 Route information setting error table 7-14

8 APPENDIX 8-1
8.1 Network Components 8-2
8.1.1 Problem of connection between LQE020 and Ethernet 8-2
8.1.2 Component list 8-2
8.2 Cabling of Coaxial Cable 8-5
8.2.1 Laying cable segment 8-5
8.3 Installation of Transceiver (Connector Type) 8-6
8.4 Installation of Transceiver (TapType) 8-10
8.5 Attaching Coaxial Connector 8-10
8.6 Attaching Tap Connector 8-13
8.7 Attaching Transceiver Cable 8-15
8.8 Attaching Terminators 8-15
8.9 Attaching Repeater 8-16
8.10 Grounding the System 8-17
8.11 Attaching Ground Terminal 8-17
8.12 Setting Single-port Transceiver 8-18
8.13 Setting and Display of Multi-port Transceiver 8-19
8.14 CPU Memory Map 8-21
8.15 Memory Map of ET.NET Module 8-22
8.16 Trouble Investigation Sheet 8-23

SUPPLEMENTARY Z-1
Supplementary: Replacing or adding on the module Z-2

FIGURES

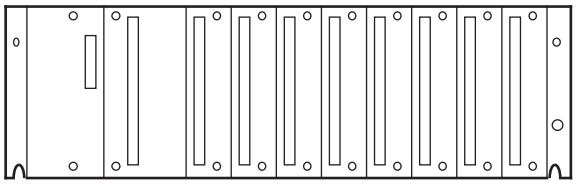
Figure 4-1 Minimum Configuration (No Repeater Used and Segment Length of Up to 500 m) 4-3
Figure 4-2 Medium-scale Configuration (Repeaters Used and Distance between Transceivers of Up to 1,500 m) 4-3
Figure 4-3 Large-scale Configuration (Repeaters and Link Segments Used and Distance between Transceivers of Up to 2,500 m) 4-4
Figure 8-1 Installation on Wall (1) 8-7
Figure 8-2 Installation on Wall (2) 8-8
Figure 8-3 Installation on Wall (3) 8-8
Figure 8-4 Installation on Wall (4) 8-8
Figure 8-5 Installation in Box (1) 8-9
Figure 8-6 Installation in Box (2) 8-9
Figure 8-7 Tap Connector Assembly Drawing 8-13
Figure 8-8 Connection of Connector and Transceiver 8-14

TABLE

Table 8-1 Switch Setting 8-20

1 BEFORE USE

1.1 CPU Mount Base



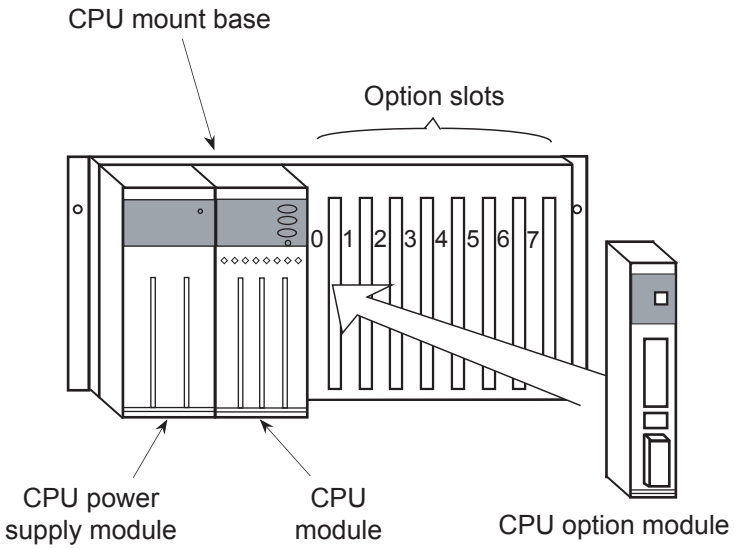
8-slot mount base

There are three types of CPU mount bases:

- 2-slot mount base (model: HSC-1020)
- 4-slot mount base (model: HSC-1040)
- 8-slot mount base (model: HSC-1080)

On the 8-slot mount base, for example, up to eight modules, except the power supply module and CPU module, can be mounted.

1.2 Mounting Optional Modules



CPU mount base: HSC-1080

PS slot: A slot into which the CPU power supply (LQV000, LQV020 or LQV100) module is inserted.

CPU slot: A slot into which the CPU module (LQP000, LQP010, LQP011 or LQP120) is inserted.

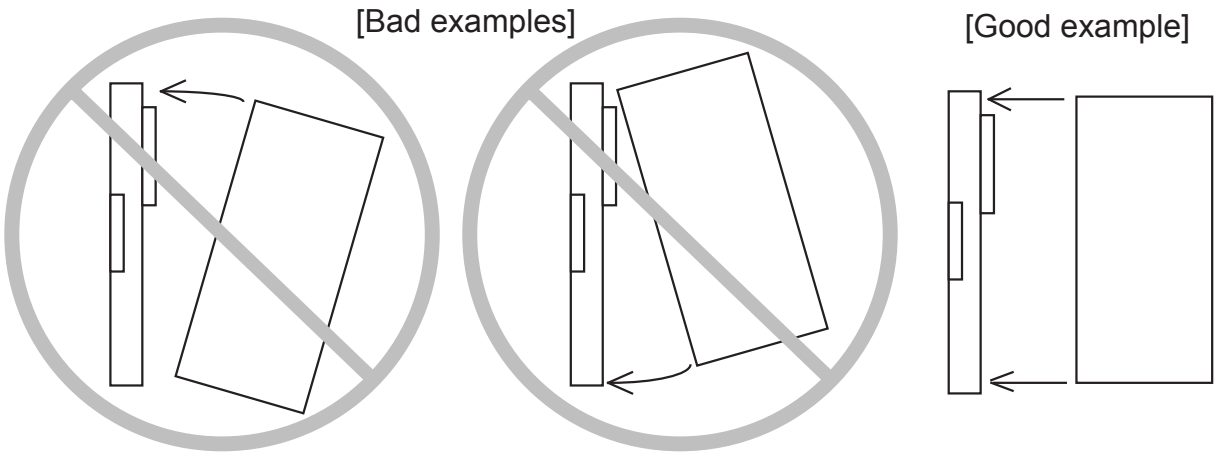
Slots 0 to 7: Slots into which optional modules or I/O modules.

CAUTION

- Insert ET.NET modules sequentially into the slots, starting from the leftmost slot, without creating any empty slots in between. Do not insert I/O modules between ET.NET modules.
- When only one ET.NET module is inserted, set it as the main module.

When mounting an option module, observe following rules.

- Mount the module straight to the front of the mount base. If it is mounted at a slant as shown in the bad examples, the connectors may be damaged and the option module may malfunction.



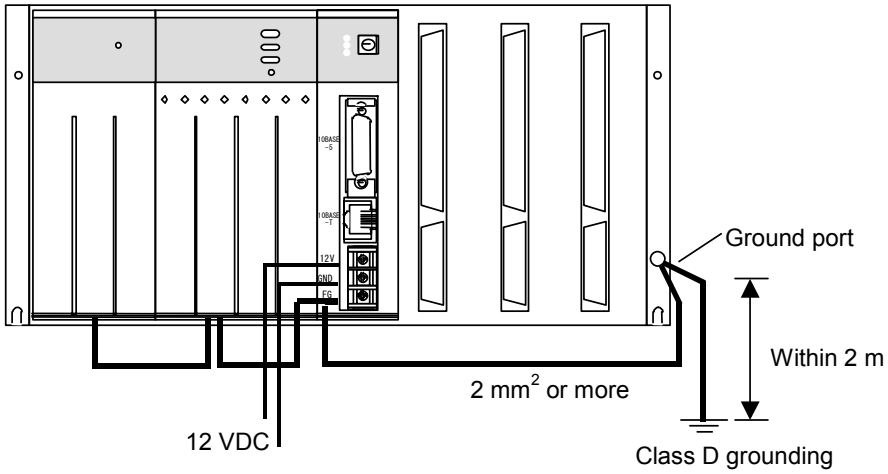
CAUTION

When the CPU mount base is located over the head because of the cabinet structure used, take care not to mount the optional modules aslant by using a stepladder or the like.

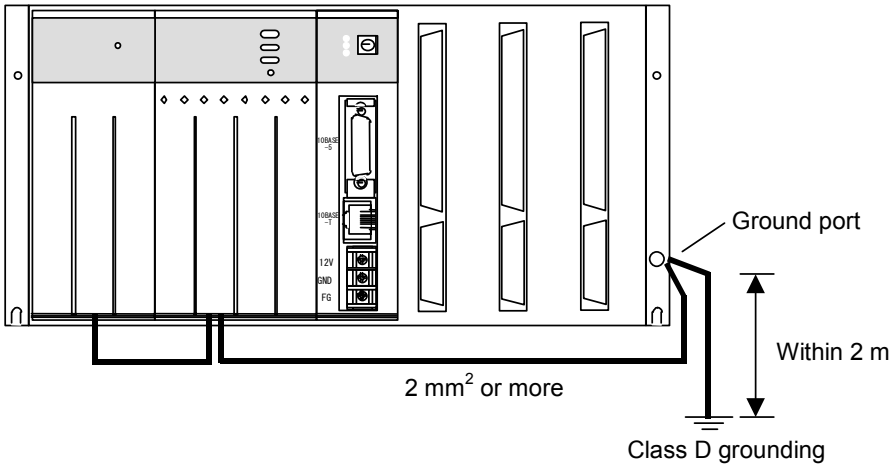
1.3 Ground Wiring

Ground the unit according to the following figure:

• Grounding for 10BASE-5



• Grounding for 10BASE-T (Do not connect the FG of the ET.NET module.)



* Class D grounding is defined in the Technical Standard for Electrical Facilities of Japan. This standard states that the grounding resistance must be 100 ohms or less for equipment operating on 300 VAC or less, and 500 ohms or less for devices that shut down automatically within 0.5 seconds when shorting occurs in low tension lines.

REQUIREMENT

- Ground the FG (frame ground) terminal as follows: Connect the FG terminal on each module provided with external terminals to the grounding terminal on the mount base. Make sure that the line used for grounding is at most 2 m long. Perform Class D grounding for the grounding terminal on the mount base.
- Use ground lines whose size is $2mm^2$ or more.
- Do not touch the 10BASE-5 connector during power-on. Otherwise, the system may malfunction due to static electricity, etc.

2 SPECIFICATIONS

2.1 Usage

The ET.NET module (model: LQE020) is connected to a local area network conforming to the IEEE802.3 specifications, and performs data communication by the TCP/IP or UDP/IP protocol.

2.2 Specifications

2.2.1 System specifications

Item	Specification
Model	LQE020
Maximum number of installable ET.NET modules	2 per CPU. (Insert ET.NET modules sequentially into the slots, starting from the leftmost slot.)
Module width	One slot wide
Mass	240 g

NOTE

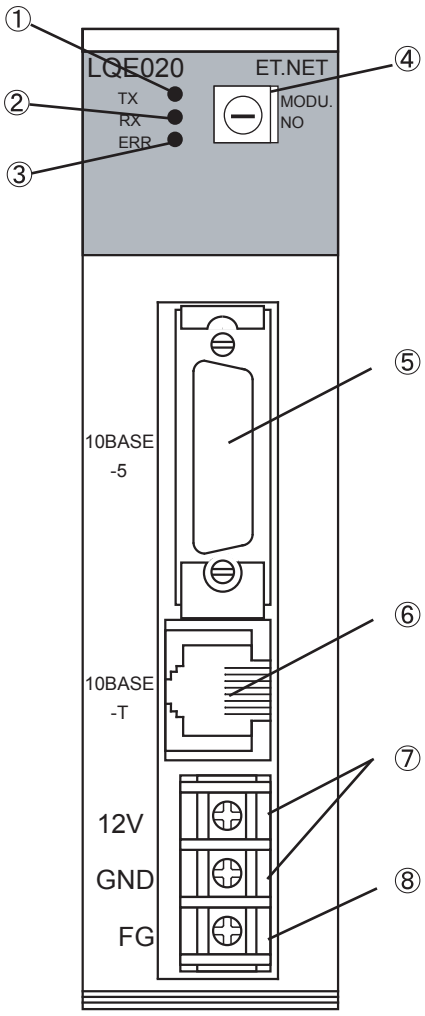
When using 10BASE-5 connections, a 12-VDC external power supply is required.

2.2.2 Line specifications

Item	Specification
Transmission method	Serial (bit serial) transmission
Electrical interface	Conforming to IEEE 802.3 (conforming to CSMA/CD) standard
Coding system	Manchester
Protocol	TCP/IP or UDP/IP
Maximum number of connectable units	10BASE-5: 100 per segment 10BASE-T: n per hub. (The value of n depends on the hub.)
Maximum number of stations	1024 per network
Connection cable	10BASE-5 coaxial cable: Up to 500 m per segment 10BASE-5 transceiver cable: Up to 50 m 10BASE-T twisted-pair cable: Up to 100 m per segment
Data transmission rate	10 Mbps

3 NAMES AND FUNCTIONS OF EACH PART AND CABLING

3.1 Names and Functions of Each Part



No.	Name	Function
①	TX LED	Lights during data transfer.
②	RX LED	Lights when data flows on the transmission line (when a carrier is detected).
③	ERR LED	Lights when a hardware error is detected.
④	Module number switch	Specifies the main module or submodule and also sets a communication port type. The setting of this switch becomes effective when resetting of the computer system is completed.
⑤	10BASE-5 connector	Connects with a PC or another controller.
⑥	10BASE-T connector	Connects to a personal computer or another controller.
⑦	Power input terminal	Connects with the power supply (12 VDC) for a transceiver which is connected with 10BASE-5.
⑧	Frame ground	Connected to the shield line of the transceiver cable.

Module No.		Description
Main	Sub	
0	1	Communication using 10BASE-5 connections
2	3	Communication using 10BASE-T connections
4	5	Communication with tools (Windows PC) via 10BASE-T connections (*)
6	7	Error
8	9	Error
A	B	Error
C	D	Error
E	F	Error

If the module number is set to 4 or 5, the IP address must be as shown below. Up to four windows can be opened on the tool (Windows PC) at the same time.
IP address: 192.192.192.001

(*) Setting the MODU No. switch in one of these two positions may result in a route information setting error. For more information, see "7.3.4 Route information setting error table."

CAUTION

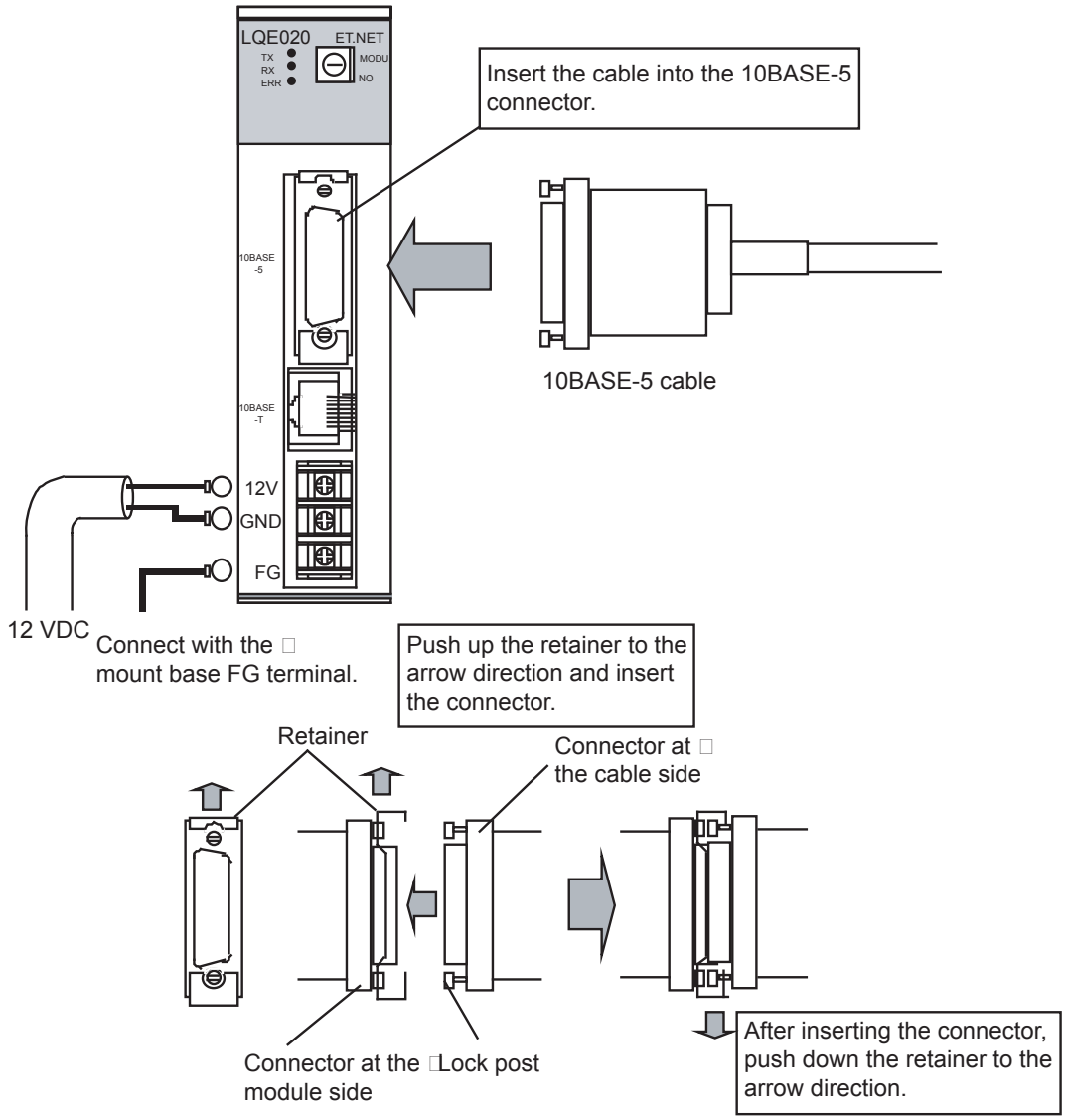
When setting the module No. switch, turn off the power switch. Otherwise, the system may malfunction.

NOTE

The following 12 VDC external power supply is recommended. Use the recommended power supply.
Power supply model name: HK-25A-12 (manufacturer: Densei-Lambda K.K.)

3.2 Cabling

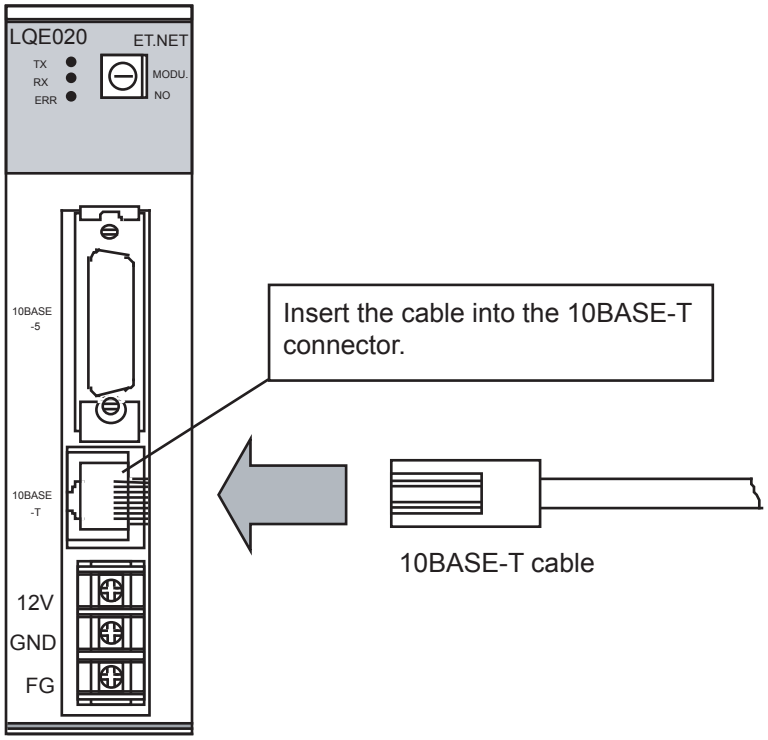
(1) Wiring for 10BASE-5



CAUTION

- This hardware unit may malfunction if it is connected poorly or has a broken line. After connecting the 10BASE-5 connector, check whether the locking post is locked by the retainer.
- Do not touch the 10BASE-5 connector during power-on. Otherwise, the system may malfunction due to static electricity, etc.

(2) Wiring for 10BASE-T



NOTE

- When using 10BASE-T, do not wire the FG terminal.
- There are two types of 10BASE-T twist-pair cable available: straight cable and cross cable. The user should choose one of these two types according to the requirements of the hardware unit to which this product is to be connected.

Hardware unit	Cable type
Hub	Straight
PC	Cross

4 USER GUIDE

4.1 System Configuration of 10BASE-5

As shown in Figure 4-1, a basic configuration consists of a single coaxial cable of up to 500 m and stations connected to this cable. Each station is connected to the coaxial cable via a transceiver cable and a transceiver. (The station means Ethernet equipment including LQE020.)
This basic configuration is also called a segment; up to 100 stations can be connected in one segment.
When the distance between stations exceeds 500 m, the number of segments can be increased by branching by using repeaters. (See Figure 4-2.) This figure shows an example of a system in which the maximum distance between stations does not exceed 1,500 m. Construct the system so that the number of repeaters between any two stations is two or less.
Figure 4-3 is an example in which the maximum distance between stations is 2,500 m. A repeater to which a link cable (up to 500 m) is attached is counted as one repeater, which is called a link segment.
The parameters related to system configuration are listed below.

Item	Specification
Maximum segment length	500 m
Maximum number of transceivers in segment	100
Maximum distance between stations	2,500 m or less (excluding transceiver cable)
Maximum number of stations in system	1,024
Maximum length of transceiver cable	50 m
Maximum number of repeaters in route between stations	2

NOTE

- Connect a repeater to a coaxial cable via a transceiver cable and a transceiver.
- A repeater can be attached to a transceiver at any position in the coaxial segment.
- Do not attach a station to a link cable.
- The distance between attached transceivers shall be a multiple of 2.5 (m).
- If a tool PC is connected in this way to call up MCS screens on that PC, only up to four such screens can be viewed on the PC.

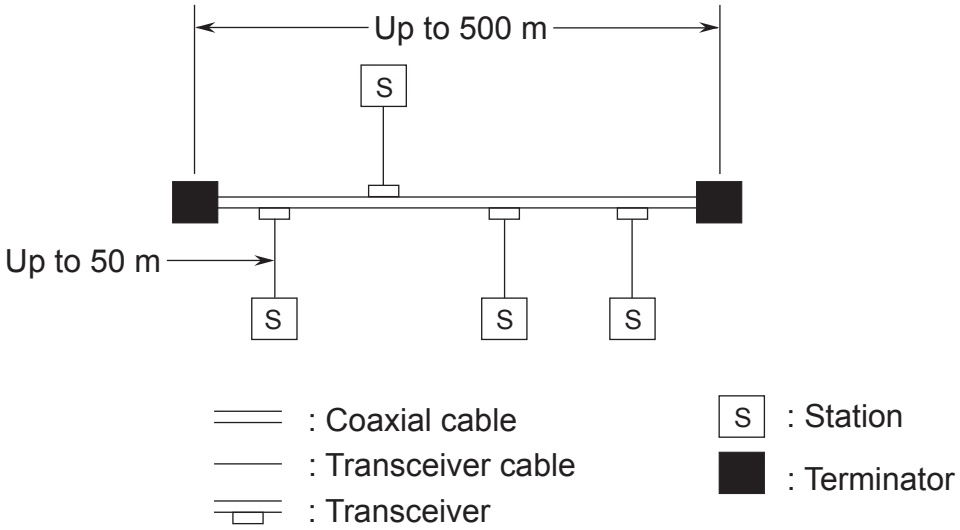


Figure 4-1 Minimum Configuration (No Repeater Used and Segment Length of Up to 500 m)

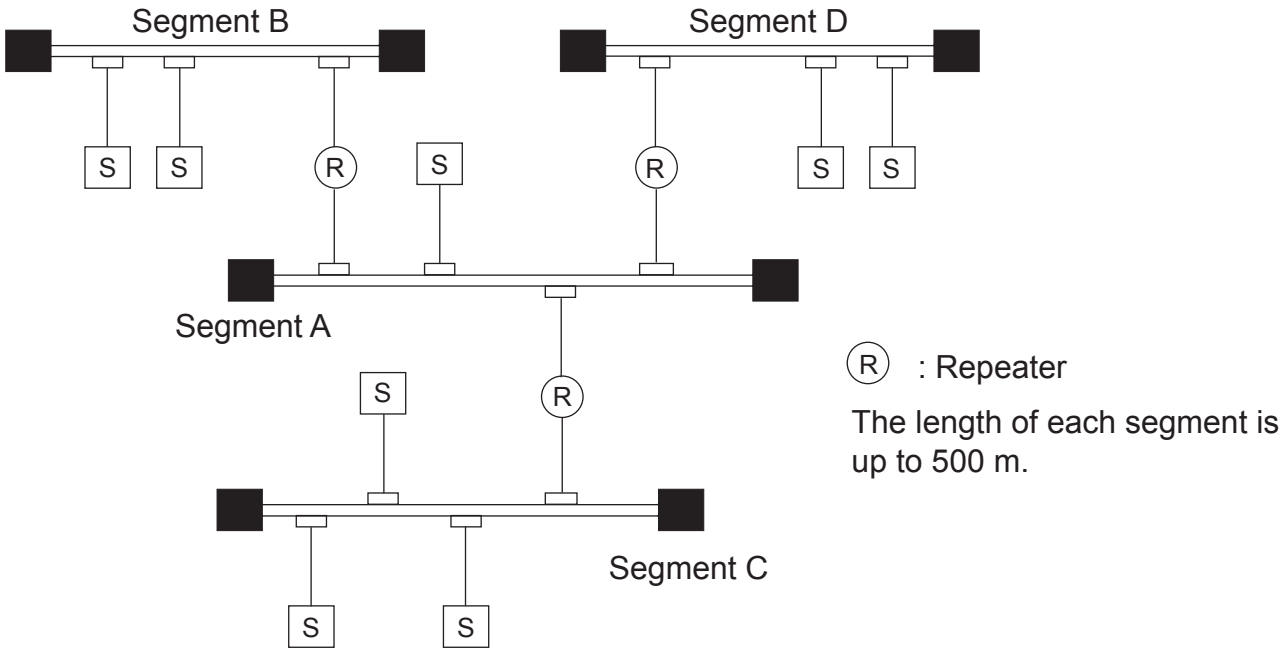


Figure 4-2 Medium-scale Configuration (Repeaters Used and Distance between Transceivers of Up to 1,500 m)

NOTE

- The number of repeaters between any two stations shall be two(2) or less.
- The number of segments to which two or more repeaters can be connected shall be one(1).

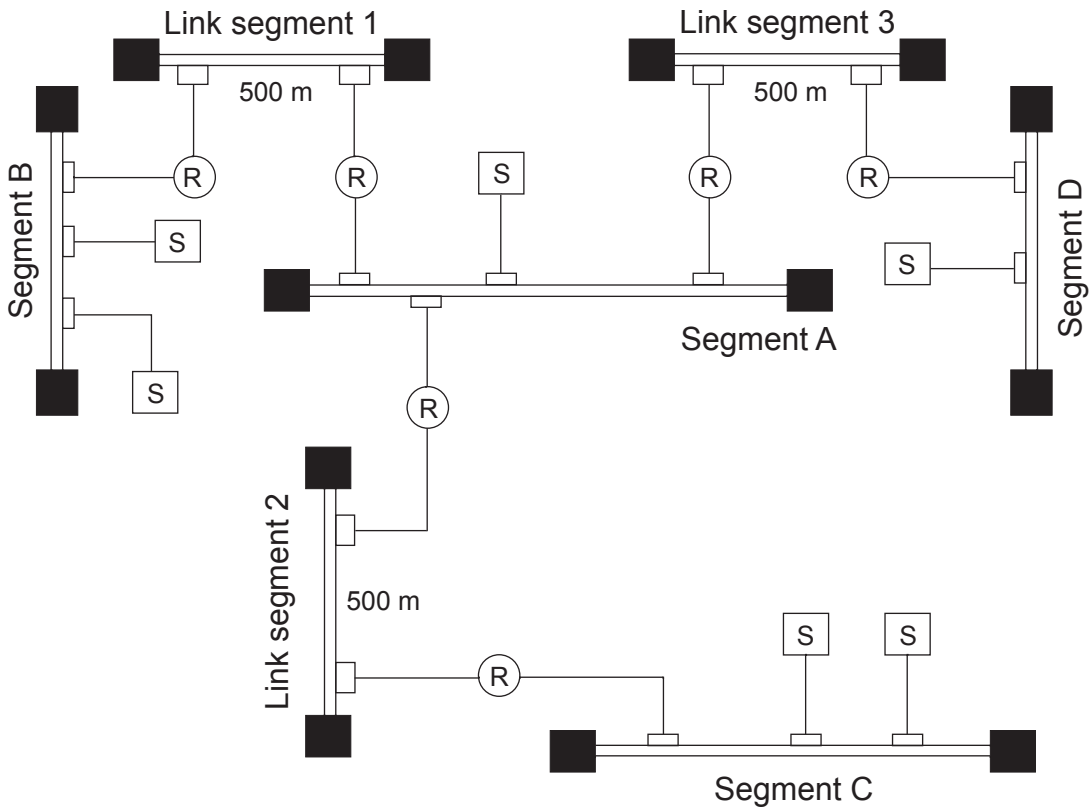


Figure 4-3 Large-scale Configuration (Repeaters and Link Segments Used and Distance between Transceivers of Up to 2,500 m)

NOTE

- The maximum length of a link segment is 500 m.
- Do not attach a station to a link segment.
- The number of repeaters between any two stations shall be two(2) or less.
- The number of segments to which two or more repeaters can be connected shall be one(1).
- A link segment including the repeaters at both ends is regarded as one repeater.

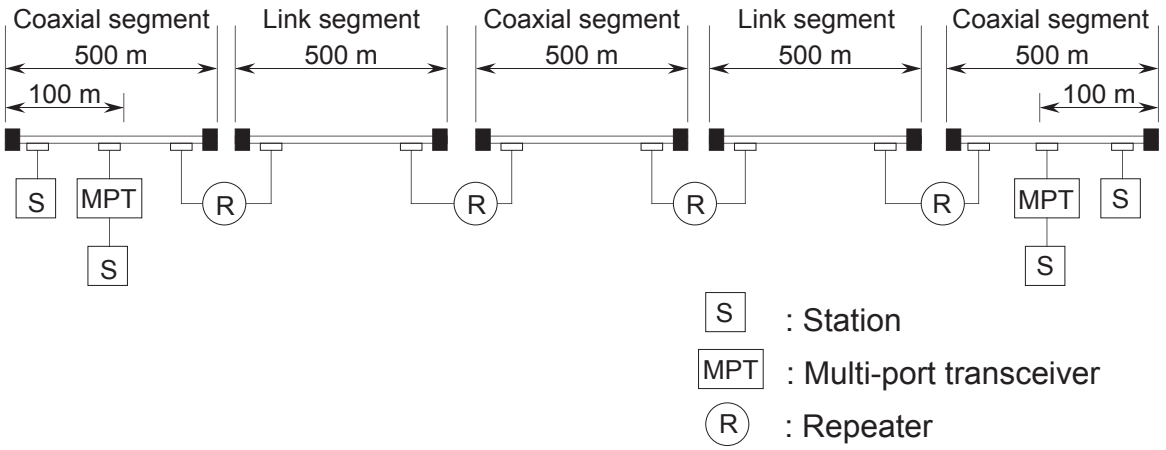
NOTE

Restrictions on multi-port transceiver installation positions

- When multi-port transceivers are installed on the most distant coaxial cable segment in a system in which the maximum length of coaxial cables is 2,500 m (five segments), data delay time increases due to the installation. To avoid this, restrictions are placed on the multi-port transceiver installation positions. The maximum distance between stations via multi-port transceivers decreases by 100 m (in terms of coaxial cable length) if it passes one single multi-port transceiver. For this reason, there is the following restriction on the coaxial cable length (L [m]) of the route from a station to another station:

  $L\ [m] \leq 2{,}500\ [m] - 100 \times N\ [m]$

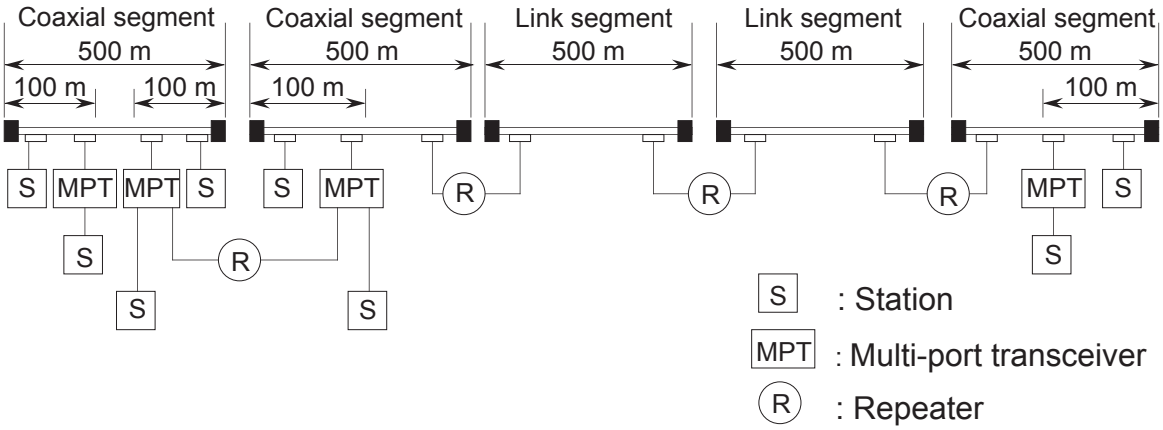
  N: Total number of passing multi-port transceivers
- In a system consisting of coaxial cables of 2,500 m in total, set a multi- port transceiver 100 m or more inside from the most distant coaxial cable terminator (such terminator position decreasing the distance between stations).



(cont.)

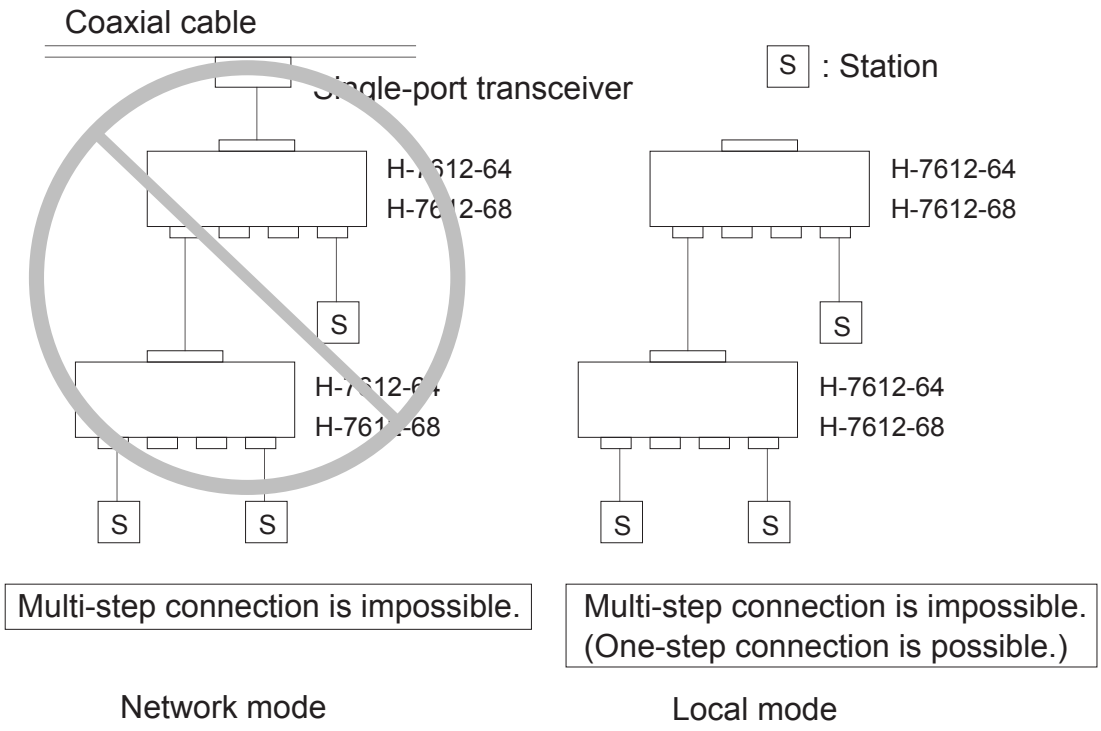
NOTE

When connecting a repeater between segments by using multi-port transceivers, it is also necessary to set the multi-port transceivers at the positions decreasing the distance between the most distant stations by 100 m each time in passes one of the multi-port transceiver.



NOTE

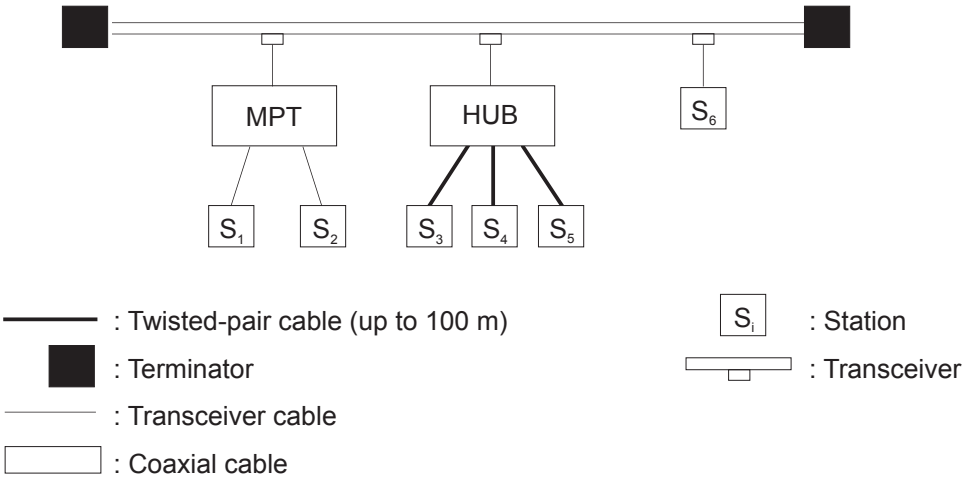
- When multi-port transceivers (H-7612-64/68) are used in network mode, multi-step connection is impossible due to the restrictions on transmission characteristics.



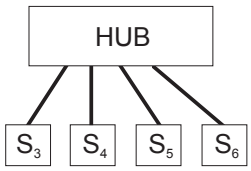
- In network mode, use the following specified device types as the single-port transceivers connected at higher level than multi-port transceivers, so that the condition to operate on the 12 VDC power supplied from the multi-port transceivers is assured:
  - HLT-200TB (manufactured by Hitachi Cable, Ltd.)
  - HLT-200 (manufactured by Hitachi Cable, Ltd.)
  - HBN-200TZ (manufactured by Hitachi Cable, Ltd.)
  - HLT-200TD (manufactured by Hitachi Cable, Ltd.)

4.2 10BASE-T System Configuration

Connecting the HUB (multi-port repeater) to a transceiver through a transceiver cable (AUI cable) enables connecting multiple stations to the HUB. For connecting stations to the HUB, use twisted-pair cables (10BASE-T).



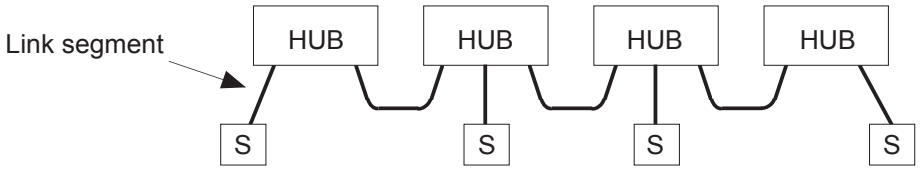
When the distance between stations is relatively short (within 200 meters), each station can be connected directly to the HUB through twisted-pair cables without using any coaxial cable or transceiver, as shown in the figure below.



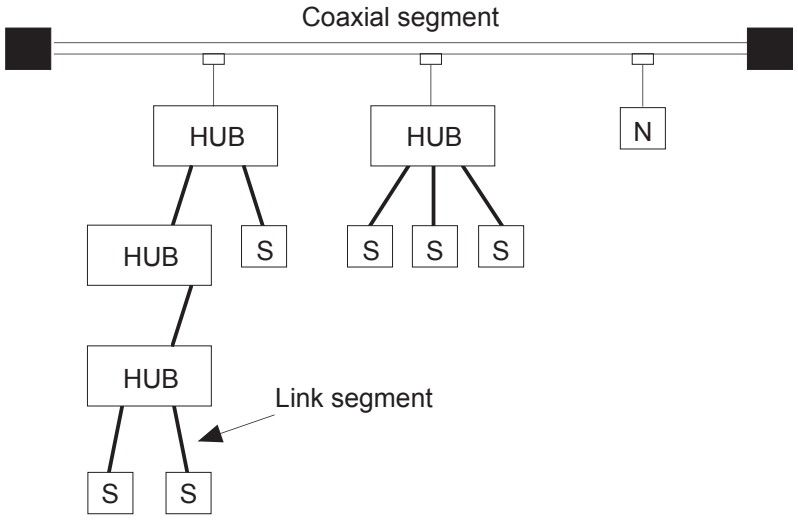
NOTE

Constraints on multi-HUB connection

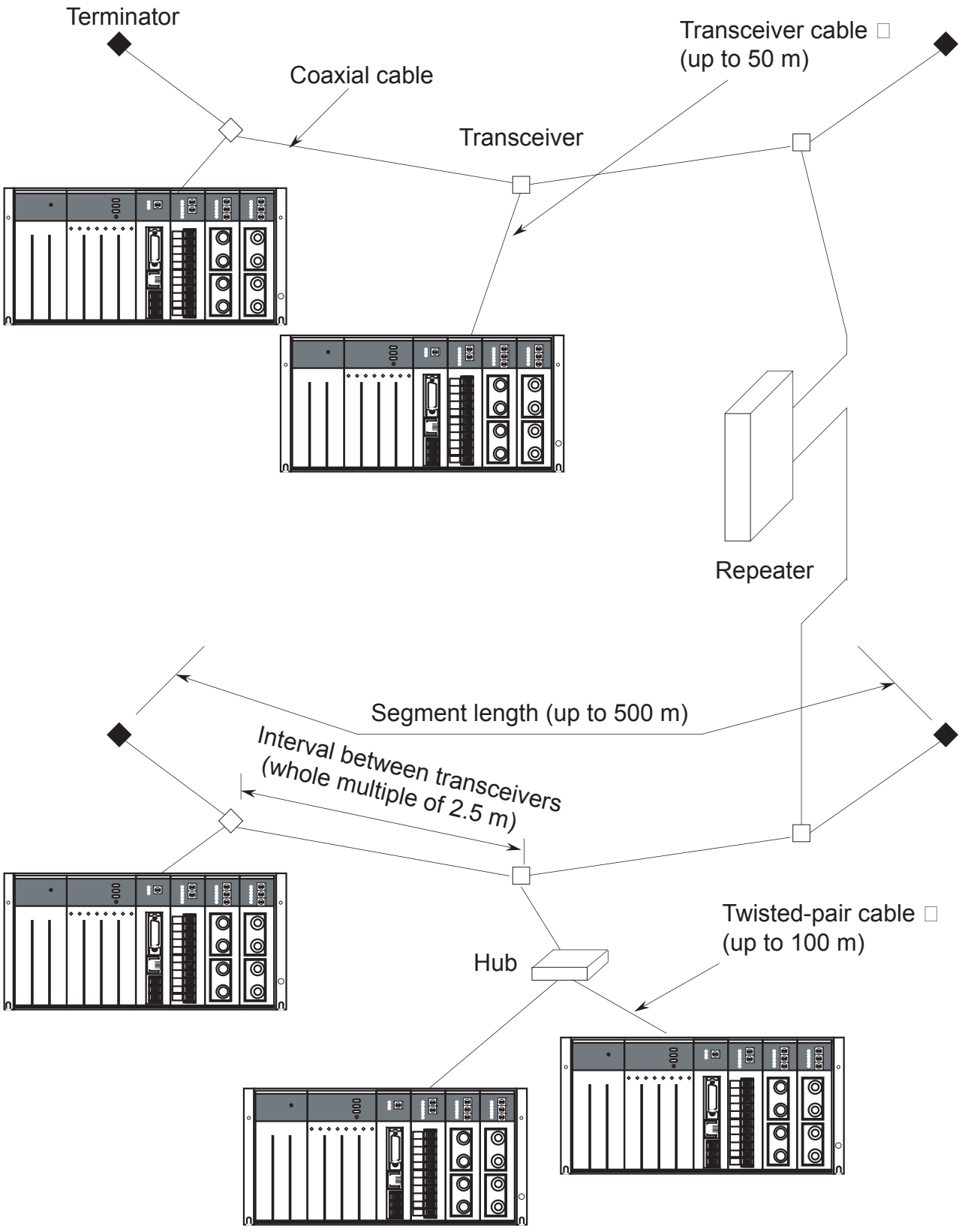
- When using multiple HUBs, configure the system so that the number of HUBs are up to four and the number of link segments up to five for any routing between stations.



- When connecting HUBs with a coaxial cable, also configure the system so that the number of HUBs are up to four and the number of ring segments up to five (three for coaxial segments) for any routing between stations.



4.3 Example of System Configuration with S10mini



4.4 System Definition Information

Set the following ② and ③ information for ET.NET (LQE020). To connect a station to another network through a router, define item④, too. Do not use a same address as that of another station. Item③needs to have a consistent value throughout one single subnetwork.

① Physical address —— An original number is set for each ET.NET ROM.
② IP address
③ Subnetwork mask ┐├ Define these items for each ET.NET by using the ET.NET system tool
④ Route information —— Define this item when connecting a station to another network through a router. The item can be set by the ET.NET system tool or by a user program.

4.4.1 Physical address

A 48-bit physical address is assigned to each ET.NET.
This is a unique address which is set on the ROM; the user cannot change it. An example of a physical address (in hexadecimal) is shown below.
Example:
00008700B001

4.4.2 IP address

The IP address used for TCP/IP and UDP/IP is a 32-bit logical address. An IP address consists of a network number and a host number. There are three types of address assignment depending on the number of hosts.

(i) Class A. (The high-order one bit of the network number is set to 0.)

Network number (8 bits)	Host number (24 bits)

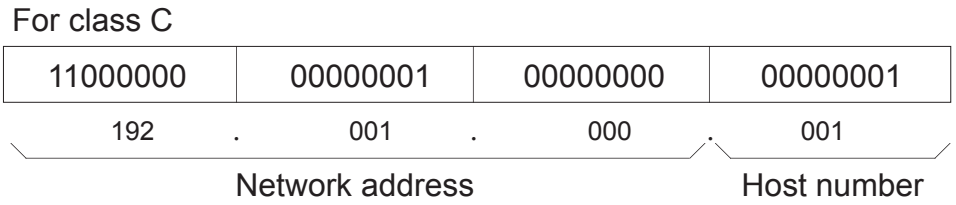
(ii) Class B. (The high-order two bits of the network number are set to 10 in binary.)

Network number (16 bits)	Host number (16 bits)

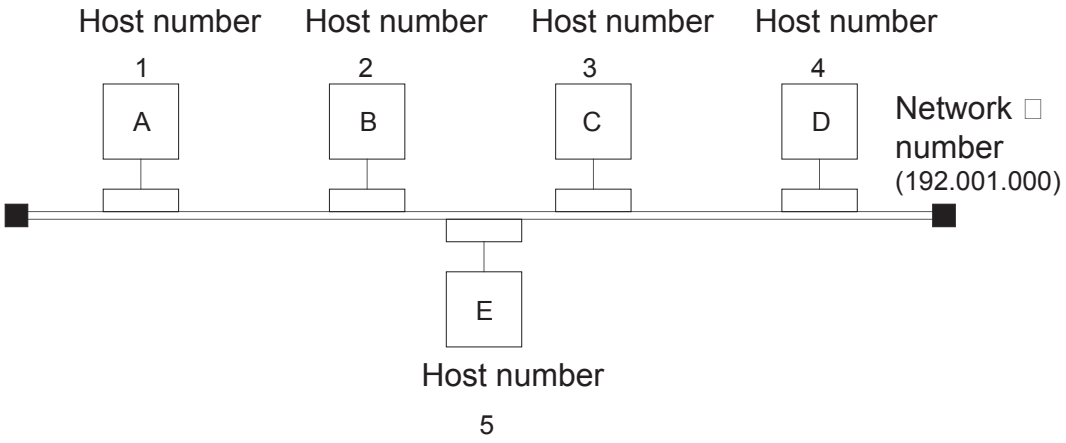
(iii) Class C. (The high-order three bits of the network number are set to 110 in binary.)

Network number (24 bits)	Host number (8 bits)

An IP address is represented in decimal; the eight-bit values are delimited from each other by a period (“.”). For example, an IP address of class C is represented as shown below.



A network is determined by a network number. Define a unique host number for each host in the network. If the number of hosts in a network is 200 or less, select class C. For instance, assume that the number 192.001.000 is selected as the network number.



As the stations A, B, C, D, and E belong to the same network, assign the numbers 1 to 5 as the unique host numbers. In this case, the IP addresses of the stations A to E are as follows:

Station A: 192.001.000.001
Station B: 192.001.000.002
Station C: 192.001.000.003
Station D: 192.001.000.004
Station E: 192.001.000.005

There are two special IP addresses: one indicates the entire network by setting all bits of host number to 0, and the other is the broadcast address in which all bits of host number are set to 1. The broadcast address is used when data is sent to all stations belonging to the network. (In this case, send data by UDP/IP communication.)

4.4.3 Subnetwork mask

When splitting an IP address into subnetworks, define the boundary between subnetwork number and local host number by a subnetwork mask. If a subnetwork mask is used with other than the default value, the address is a the broadcast address as shown in the example below.

Example: For class B:

IP address	Subnetwork mask	Broadcast address
128.123.000.001	255.255.000.000	128.123.255.255
128.123.001.001	255.255.255.000	128.123.001.255

4.4.4 Route information

Route information must be defined if you want to connect a station to another network through a router. As the route information, the IP addresses of both the communication destination and router are registered in a pair.

(1) IP address of communication destination
For each communication destination, an IP address is registered. When multiple communication destinations exist in the same network, a network address may be set as a generic address. (The host number of the IP address that has been set to “0” is used as the network address.)

(2) IP address of router
The IP address of the router in the same network as the ET.NET module is registered. When multiple routers is involved in the communication route to the destination, register only the router in the same network as the ET.NET module.

The following two methods are available for setting route information.

- Setting in the socket handler route_add() in a C program
  - Refer to the item pertaining the socket hander route_add().

- Setting by using the Windows® version of ET.NET system tool (V6 or later)
  - Refer to the OPTION ET.NET For Windows software manual (manual number SAE-3-148).

NOTE

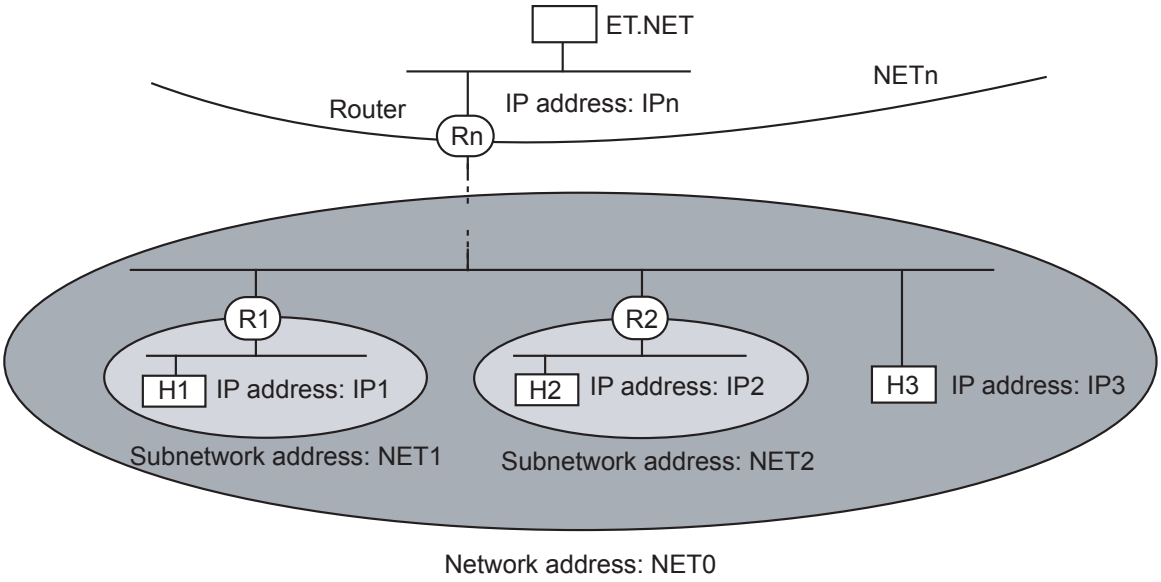
- Routing information setting function by a Windows® ET.NET system can be used only when the LQE020 module revision No. is C (the CPU indicator display is ETM 2.0 or ETS 2.0) or later and the ET.NET system tool version is V06 or later.
- Up to 15 items of route information including both route_add() and tool settings can be registered.
- If the same setting is made by route_add() and the tool, the setting made by the latter has priority and that made by route_add() is invalidated. In this case, an error return code will be given back.
- The route information setting is supported only by the Windows® version tool (it is not supported by the PSEα version tool).
- The addresses that can be registered are IP and network addresses. No subnetwork address can be registered.
  This is because the ET.NET module recognizes route information as an IP address or network address but not as a subnetwork address. Even if a subnetwork address is registered, it is not recognized as an IP address, so no communication can be performed.

NOTE

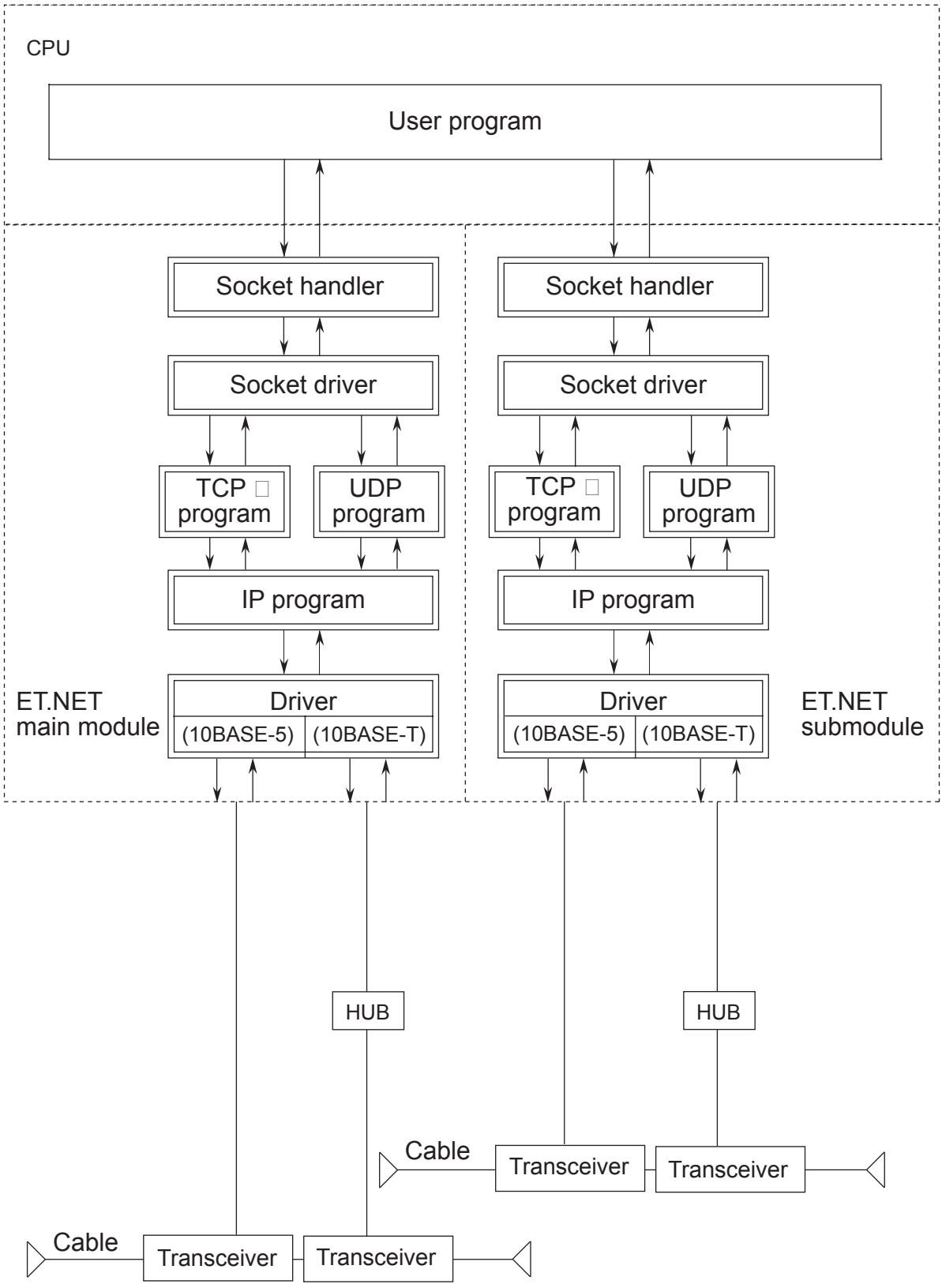
The following are examples of route information registered for the network configuration shown in the figure below.

- Examples of registering route information -

- Route information registered for communication with host H1
  - IP address of router Rn: IPn
  - IP address of host H1: IP1
- Route information registered for communication with host H3
  - IP address of router Rn: IPn
  - IP address of host H3: IP3 or network address NET0



4.5 Software Configuration of ET.NET



4.6 ET.NET System Programs

This section explains the system programs shown in Section 4.5, “Software Configuration of ET.NET”.
The system programs are classified into the six types listed below. Each program runs on a CPU or ET.NET module.

- Socket handler
- Socket driver
- TCP program
- UDP program
- IP program
- Driver

4.6.1 Socket handler

The socket handler, invoked as a function in C, controls the ET.NET module for user program. By using the socket handler, the user can create programs without considering the hardware specifications and communication protocol.

4.6.2 Socket driver

The socket driver passes commands from the socket handler to the TCP or UDP program via the memory interface for subsequent processing.

4.6.3 TCP program

The TCP program as a higher-level protocol conducts high-reliability data transmission/reception management.

The functions of the TCP program are listed below.

- Reliability check
  - Confirmation of reception response signal (ACK)
  - Sequence check by sequence numbers
  - Data checksum check
- Data retransmission (when an error is detected by reliability check)
- Flow control for receivable data amount
- Simultaneous communication with multiple processes (multiplexing)
- Logical connection by connection establishment
- Data security and priority management

4.6.4 UDP program

The UDP program as a higher-level protocol manages high-speed transmission/reception of a large amount of data.
The UDP program has the following functions:

- Connectionless communication
- Simultaneous communication
- Packet-based data transmission

4.6.5 IP program

The IP program as a low-level protocol conducts logical connection of communication paths.
The IP program has the following functions:

- Disassembling or reassembling data according to the maximum packet length
- Exchanging IP address and physical address

4.6.6 Driver

The driver controls the communication circuit, and sends data to and receives data from lines (transceivers).
The driver has the following functions:

- CRC (Cyclic Redundancy Check) for transmission/reception of data
- Data collision detection during transmission/reception and retransmission

4.7 User-created Program

This section describes programs that needs to be created by the user.

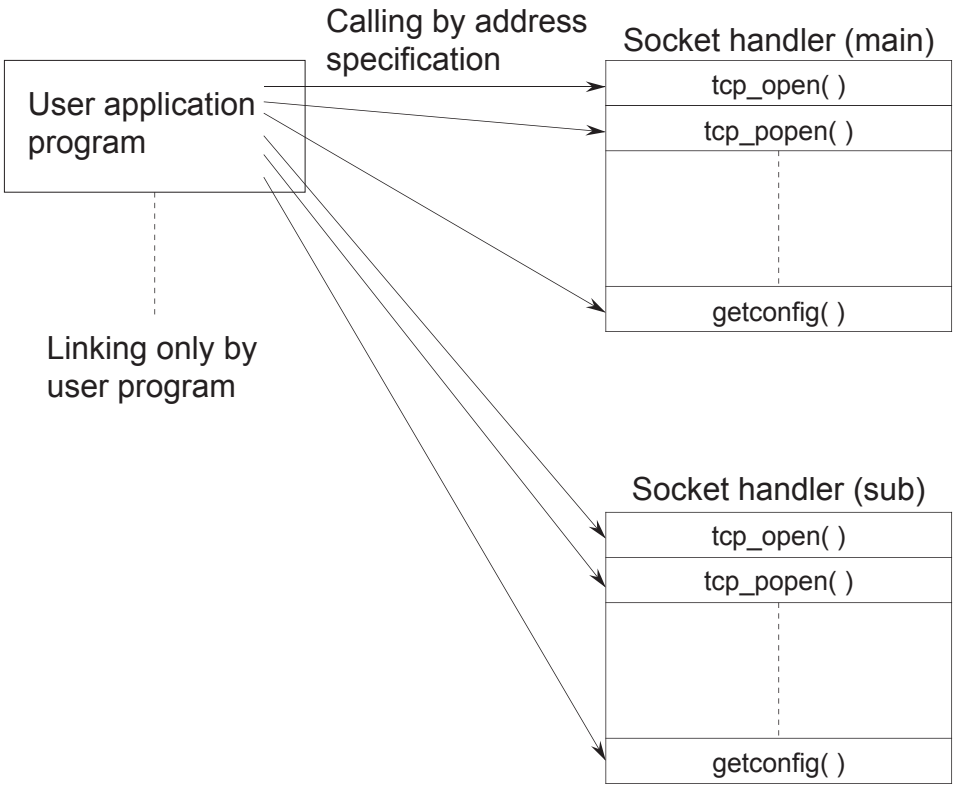
4.7.1 User program

The user program starts the socket handler, and sends or receives data.
Create the user program as a C mode program, and load it into the S10mini series.
A C mode program is written in programming languages such as C, assembler language, etc., and can be executed in the form of tasks or P coil. Use the CPMS (Compact Process Monitor System) as the OS. Extended memory is required for a C mode program.

The socket handler is explained in Section 4.8, “Socket Handler.”
For programming using the socket handler, see CHAPTER 5, “PROGRAM EXAMPLES.”

4.8 Socket Handler

The socket handler, invoked as a function in C, controls the ET.NET module for user program, and carries out data transmission and reception. The socket handler consists of 20 functions.
Call the socket handler by specifying its entry addresses. A user program cannot be created (linked) in a form including the socket handler.



4.8.1 Socket handler function list

The table below lists the functions of the socket handler.

Name	Subroutine call address		Function	Corresponding program
	Main	Sub		
tcp_open()	/874100	/8F4100	Actively opens TCP.	TCP/IP
tcp_popen()	/874106	/8F4106	Passively opens TCP.	TCP/IP
tcp_accept()	/87410C	/8F410C	Accepts a TCP connection request.	TCP/IP
tcp_close()	/874112	/8F4112	Terminates a TCP connection.	TCP/IP
tcp_abort()	/87411E	/8F411E	Kills a TCP connection.	TCP/IP
tcp_getaddr()	/874124	/8F4124	Reads TCP socket information.	TCP/IP
tcp_stat()	/87412A	/8F412A	Reads TCP connection status.	TCP/IP
tcp_send()	/874130	/8F4130	Sends TCP data.	TCP/IP
tcp_receive()	/874136	/8F4136	Receives TCP data.	TCP/IP
udp_open()	/874160	/8F4160	Opens UDP.	UDP/IP
udp_close()	/874166	/8F4166	Closes UDP.	UDP/IP
udp_send()	/87416C	/8F416C	Sends UDP data.	UDP/IP
udp_receive()	/874172	/8F4172	Receives UDP data.	UDP/IP
route_list()	/874178	/8F4178	Reads routing information.	TCP/IP and UDP/IP
route_del()	/87417E	/8F417E	Deletes routing information.	TCP/IP and UDP/IP
route_add()	/874184	/8F4184	Registers routing information.	TCP/IP and UDP/IP
arp_list()	/87418A	/8F418A	Reads ARP information.	TCP/IP and UDP/IP
arp_del()	/874190	/8F4190	Deletes ARP information.	TCP/IP and UDP/IP
arp_add()	/874196	/8F4196	Registers ARP information.	TCP/IP and UDP/IP
getconfig()	/87419C	/8F419C	Reads configuration information.	TCP/IP and UDP/IP

NOTE

- The maximum number of sockets that can be used simultaneously by one single module is 12 for TCP and 8 for UDP.
- The port numbers 0 to 9999 are reserved by the system; the user can use port numbers 10000 to 65535.
- The length of data to be transmitted or received in each invocation of a function is 1 to 4096 bytes for TCP and 1 to 1472 bytes for UDP.
- The IP addresses and subnet masks are set in the operating system table in the CPU. When the CPU is replaced or the operating system is reloaded, these items need be set again.

- Kill of task -

If a task using the socket handler is killed, the socket remains in registered state (except when the task has executed tcp_close() or udp_close() for the socket used by that task). That is, the socket status at the time of task kill remains undeleted although the task terminated. The socket in such a state is called a *floating socket*.

As a floating socket cannot be used by other tasks, take any one of the following actions for the floating socket or module:

1. Execute tcp_close() or udp_close() for the floating socket by another task or built-in subroutine.
2. Reset the CPU.
3. Cut off the power supply, then recover it.

tcp_open()

Function

This function registers a socket of the TCP/IP program, reserves a port, and issues a connection request for a remote station. The registered socket ID or an error code is returned as the return value. This function transmits SYN and waits for connection establishment (SYN reception from remote station). If there is no response from the remote station within 75 seconds, this function ends up with a port release error (error code: 0xF0FF). In this case, reissue tcp_open().

Linking procedure

C language

Main

```
struct   open_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     notuse;
   char     ttl;
};
                〉
   short    ( *tcp_open )( );
   short    rtn;
   struct    open_p    *padr;
                〉
            tcp_open =( short (*) ( ) )0x874100;
                〉
            rtn = ( *tcp_open )( padr );
                〉
```

Sub

```
struct   open_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     notuse;
   char     ttl;
};
                〉
   short    ( *tcp_open )( );
   short    rtn;
   struct    open_p    *padr;
                〉
            tcp_open =( short (*) ( ) )0x8F4100;
                〉
            rtn = ( *tcp_open )( padr );
                〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> dst_ip: IP address of remote station

padr -> dst_port: Port number of remote station

padr -> src_port: Port number of local station

padr -> notuse: Fixed at 0 (unused)

padr -> ttl: Time to live

If ttl is set to 0, the default value (30) is assumed.

Output parameters:

Return value: The registered socket ID or an error code is returned.

(0 to 0x000F) Registered socket ID

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_popen()

Function This function registers a socket for the TCP/IP program, and puts the socket into passive state. The registered socket ID or an error code is returned as the return value. This function is equivalent to socket+bind+listen in UNIX. If dst_ip and dst_port are set to 0, a connection request from any remote station can be accepted. If src_port is set to 0, optional port from 1024 to 2047 is reserved.

Linking procedure

C language

Main

```
struct   popen_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     listennum;
   char     ttl;
};
                 ≀
   short    ( *tcp_popen )( );
   short    rtn;
   struct    popen_p    *padr;
                 ≀
            tcp_popen = (short (*)( ) )0x874106;
                 ≀
            rtn = ( *tcp_popen )( padr );
                 ≀
```

Sub

```
struct   popen_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     listennum;
   char     ttl;
};
                 ≀
   short    ( *tcp_popen )( );
   short    rtn;
   struct    popen_p    *padr;
                 ≀
            tcp_popen = (short (*)( ) )0x8F4106;
                 ≀
            rtn = ( *tcp_popen )( padr );
                 ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> dst_ip: IP address of remote station

padr -> dst_port: Port number of remote station

padr -> src_port: Port number of local station

padr -> listennum: Maximum number of connections not accepted (fixed at 0: reserved for future extension)

padr -> ttl: Time to live

If no remote station is specified, set dst_ip and dst_port to 0.

If ttl is set to 0, the default value (30) is assumed.

Output parameters:

Return value: The registered socket ID or an error code is returned.

(0 to 0x000F) Registered socket ID

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_accept()

Function This function waits for a connection request (SYN reception) for the socket ID that was placed in passive state by the tcp_popen() function in the TCP/IP program, and accepts connection establishment. The socket ID registered after connection establishment or an error code is returned as the return value. The socket ID in an input parameter and that registered after connection establishment have the same value. This function continues waiting until the remote station is connected.

Linking procedure

C language

Main	Sub

struct accept_p {
short s_id;
};
{
short (*tcp_accept)( );
short rtn;
struct accept_p *padr;
{
tcp_accept =(short (*) ( ))0x87410C;
}
rtn = (*tcp_accept)(padr);
}
	struct accept_p {
	short s_id;
	};
	{
	short (*tcp_accept)( );
	short rtn;
	struct accept_p *padr;
	{
	tcp_accept =(short (*) ( ))0x8F410C;
	}
	rtn = (*tcp_accept)(padr);
	}

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

Return value: The registered socket ID or an error code is returned.

(0 to 0x000F) Registered socket ID

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_close()

Function This function terminates the connection corresponding to a socket ID, and deletes the socket. The processing result is returned as the return value. This function transmits FIN characters and waits for connection termination (FIN reception from remote station). If there is no response from the remote station within 30 seconds, this function ends up with a socket driver timeout error (error code: 0xF012). In this case, issue tcp_abort().

Linking procedure

C language

Main

```
struct   close_p {
   short   s_id;
};
                〉
   short    ( *tcp_close )( );
   short    rtn;
   struct    close_p   *padr;
               〉
             tcp_close = (short (*) ( ) )0x874112;
               〉
             rtn = ( *tcp_close )( padr );
               〉
```

Sub

```
struct   close_p {
   short   s_id;
};
                〉
   short    ( *tcp_close )( );
   short    rtn;
   struct    close_p   *padr;
               〉
             tcp_close = (short (*) ( ) )0x8F4112;
               〉
             rtn = ( *tcp_close )( padr );
               〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_abort()

Function This function kills (by sending RST characters) the connection corresponding to a socket ID, and deletes the socket. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   sid_p {
   short   s_id;
};
            ≀
   short    ( *tcp_abort )( );
   short    rtn;
   struct    sid_p   *padr;
            ≀
            tcp_abort = (short (*) ( ) )0x87411E;
            ≀
            rtn = ( *tcp_abort )( padr );
            ≀
```

Sub

```
struct   sid_p {
   short   s_id;
};
            ≀
   short    ( *tcp_abort )( );
   short    rtn;
   struct    sid_p   *padr;
            ≀
            tcp_abort = (short (*) ( ) )0x8F411E;
            ≀
            rtn = ( *tcp_abort )( padr );
            ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_getaddr()

Function This function obtains the IP address of the remote station to be connected corresponding to a socket ID and the port numbers of the local and remote stations. The processing result is returned as the return value. When the result is normal termination, the obtained information at outinf is validated.

Linking procedure

C language

Main

```
struct   sid_p {
   short    s_id;
};
struct   getaddr_p {
   long     ipaddr;
   short    src_port;
   short    dst_port;
};
                〉
   short    ( *tcp_getaddr )( );
   short    rtn;
   struct    sid_p    *padr;
   struct    getaddr_p    *outinf;
                〉
             tcp_getaddr = (short(*)( ) )0x874124;
                〉
             rtn = ( *tcp_getaddr )( padr, outinf );
                〉
```

Sub

```
struct   sid_p {
   short    s_id;
};
struct   getaddr_p {
   long     ipaddr;
   short    src_port;
   short    dst_port;
};
                〉
   short    ( *tcp_getaddr )( );
   short    rtn;
   struct    sid_p    *padr;
   struct    getaddr_p    *outinf;
                〉
             tcp_getaddr = (short(*)( ) )0x8F4124;
                〉
             rtn = ( *tcp_getaddr )( padr, outinf );
                〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

outinf: Starting address of output parameters

outinf -> ipaddr: IP address of remote station

outinf -> src_port: Port number of local station

outinf -> dst_port: Port number of remote station

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_stat()

Function This function obtains the status of the connection corresponding to a socket ID. The processing result is returned as the return value. When the result is normal termination, the obtained information at outinf is validated.

Linking procedure

C language

Main

```
struct   sid_p {
   short    s_id;
};
struct   stat_p {
   unsigned   short   stat;
   unsigned   short   urg;
   unsigned   short   sendwin;
   unsigned   short   recvwin;
};
                〉
   short    ( *tcp_stat )( );
   short    rtn;
   struct    sid_p   *padr;
   struct    stat_p   *outinf;
                〉
             tcp_stat =(short(*)( ) )0x87412A;
                〉
             rtn = ( *tcp_stat )( padr, outinf );
                〉
```

Sub

```
struct   sid_p {
   short    s_id;
};
struct   stat_p {
   unsigned   short   stat;
   unsigned   short   urg;
   unsigned   short   sendwin;
   unsigned   short   recvwin;
};
                〉
   short    ( *tcp_stat )( );
   short    rtn;
   struct    sid_p   *padr;
   struct    stat_p   *outinf;
                〉
             tcp_stat =(short(*)( ) )0x8F412A;
                〉
             rtn = ( *tcp_stat )( padr, outinf );
                〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

outinf: Starting address of output parameters

outinf -> stat: Connection status

0: CLOSED
1: LISTEN
2: SYN_SENT
3: SYN_RECEIVED
4: ESTABLISHED
5: CLOSE_WAIT
6: FIN_WAIT_1
7: CLOSING
8: LAST_ACK
9: FIN_WAIT_2
10: TIME_WAIT

outinf -> urg: Whether there is urgent data

0: There is no urgent data.
Other than 0: Number of urgent data items

outinf -> sendwin: Remaining quantity of send data of send window

outinf -> recvwin: Amount of receive data that has arrived

Return value: The processing result is returned.

(0) Normal termination
(0xF000 to 0xFFFF) Error code
For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_send()

Function

This function sends data to the connection corresponding to a socket ID. The starting address and the length of the sent data are indicated by parameters buf and len, respectively. The processing result is returned as the return value. If the value 0xF012 is returned as the processing result, confirm that transmission is being retried, by checking the connection status and the residual quantity of the send window obtained by the tcp_stat() function. The tcp_send() function makes a return when the data is stored in the send window. Confirm the data transmission status by the remaining quantity of send data of the send window obtained by tcp_stat().

Linking procedure

C language

Main

```
struct   send_p {
   short    s_id;
   short    len;
   char     *buf;
};
                〉
   short    ( *tcp_send )( );
   short    rtn;
   struct    send_p    *padr;
                〉
            tcp_send = (short(*) ( ) )0x874130;
                〉
            rtn = ( *tcp_send )( padr );
                〉
```

Sub

```
struct   send_p {
   short    s_id;
   short    len;
   char     *buf;
};
                〉
   short    ( *tcp_send )( );
   short    rtn;
   struct    send_p    *padr;
                〉
            tcp_send = (short(*) ( ) )0x8F4130;
                〉
            rtn = ( *tcp_send )( padr );
                〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

padr -> len: Length of sent data (1 to 4096 bytes)

padr -> buf: Starting address of sent data

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

tcp_receive()

Function This function receives data from the connection corresponding to a socket ID. The received data is stored in the receive buffer whose the starting address is indicated by parameter buf. The data length is specified by parameter len. The processing result is returned as the return value. In this function, receive wait time can be specified for parameter tim. However, this function makes a return when the data is received, even if the wait time has not elapsed.

Linking procedure

C language

Main

```
struct   receive_p {
   short    s_id;
   short    len;
   char     *buf;
   long     tim;
};
                 }
   short    ( *tcp_receive )( );
   short    rtn;
   struct    receive_p    *padr;
                 {
              tcp_receive =(short(*) ( ) )0x874136;
                 }
              rtn = ( *tcp_receive )( padr );
                 }
```

Sub

```
struct   receive_p {
   short    s_id;
   short    len;
   char     *buf;
   long     tim;
};
                 }
   short    ( *tcp_receive )( );
   short    rtn;
   struct    receive_p    *padr;
                 {
              tcp_receive =(short(*) ( ) )0x8F4136;
                 }
              rtn = ( *tcp_receive )( padr );
                 }
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

padr -> len: Receive buffer length (1 to 4096 bytes)

padr -> buf: Starting address of receive buffer

padr -> tim: Receive wait time (0 to 86400000 ms [24 hours])

Output parameters:

Return value: The processing result is returned.

(0) Normal termination (no receive data)

(0x0001 to 0x1000) Normal termination (number of received bytes)

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

udp_open()

Function This function registers a socket for the UDP/IP program, and reserves a port. The registered socket ID or an error code is returned as the return value.
If a 0 is specified for parameter dst_ip, packets can be received from an arbitrary host.
If a 0 is specified in the parameter dst_port, data can be received from an arbitrary port.
If a 0 is specified in the parameter src_port, unused ports from 1024 to 2048 are reserved.

Linking procedure

C language

Main

```
struct   uopen_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     pktmode;
   char     ttl;
};
                ≀
   short    ( *udp_open )( );
   short    rtn;
   struct    uopen_p    *padr;
                ≀
            udp_open =(short(*) ( ) )0x874160;
                ≀
            rtn = ( *udp_open )( padr );
                ≀
```

Sub

```
struct   uopen_p {
   long     dst_ip;
   short    dst_port;
   short    src_port;
   char     pktmode;
   char     ttl;
};
                ≀
   short    ( *udp_open )( );
   short    rtn;
   struct    uopen_p    *padr;
                ≀
            udp_open =(short(*) ( ) )0x8F4160;
                ≀
            rtn = ( *udp_open )( padr );
                ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> dst_ip: IP address of remote station

padr -> dst_port: Port number of remote station

padr -> src_port: Port number of local station

padr -> pktmode: Packet mode (fixed to 0)

padr -> ttl: Time to live

If ttl is set to 0, the default value (30) is assumed.

Output parameters:

Return value: The registered socket ID or an error code is returned.

(0x0020 to 0x0027) Registered socket ID

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

udp_close()

Function This function deletes the socket identified by a given socket ID. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   uclose_p {
   short    s_id;
};
                ≀
   short    ( *udp_close )( );
   short    rtn;
   struct    uclose_p    *padr;
                ≀
            udp_close =(short(*) ( ) )0x874166;
                ≀
            rtn = ( *udp_close )( padr );
                ≀
```

Sub

```
struct   uclose_p {
   short    s_id;
};
                ≀
   short    ( *udp_close )( );
   short    rtn;
   struct    uclose_p    *padr;
                ≀
            udp_close =(short(*) ( ) )0x8F4166;
                ≀
            rtn = ( *udp_close )( padr );
                ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

udp_send()

Function This function sends data to the socket identified by a given socket ID. The starting address and the length of the sent data are indicated by the parameters buf and len, respectively. The processing result is returned as the return value. As for specifications of dst_ip and dst_port, those specified in udp_open() have priority.

Linking procedure

C language

Main

```
struct   usend_p {
   short    s_id;
   short    notuse;
   long     dst_ip;
   short    dst_port;
   short    len;
   char     *buf;
};
            }
   short    ( *udp_send )( );
   short    rtn;
   struct    usend_p    *padr;
            {
            udp_send =(short(*) ( ) )0x87416C;
            }
            rtn = ( *udp_send )( padr );
            }
```

Sub

```
struct   usend_p {
   short    s_id;
   short    notuse;
   long     dst_ip;
   short    dst_port;
   short    len;
   char     *buf;
};
            }
   short    ( *udp_send )( );
   short    rtn;
   struct    usend_p    *padr;
            {
            udp_send =(short(*) ( ) )0x8F416C;
            }
            rtn = ( *udp_send )( padr );
            }
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

padr -> notuse: Fixed at 0 (unused)

padr -> dst_ip: IP address of remote station

padr -> dst_port: Port number of remote station

padr -> len: Length of sent data (1 to 1472 bytes)

padr -> buf: Starting address of send data

If a value other than 0 is specified in udp_open(), dst_ip and dst_port specifications in udp_open() are used.

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

NOTE

Specifications of dst_ip and dst_port

- If a value other than 0 is specified in udp_open(), the parameters specified in udp_open() are used.
- If a 0 is specified in udp_open(), the parameters specified in udp_send() are used.
- If a 0 is specified in both udp_open() and udp_send(), the function returns with an invalid address error (error code: 0xFFF0). In this case, correct the user program.

udp_receive()

Function This function receives data from the socket identified by a given socket ID. The received data is stored in the receive buffer whose starting address is indicated by the parameter buf.
The processing result is returned as the return value. In this function, receive wait time can be specified in the parameter tim. However, this function makes a return when the data is received, even if the wait time has not elapsed.

Linking procedure

C language

Main

```
struct   ureceive_p {
   short    s_id;
   short    notuse;
   char     *buf;
   long     tim;
};
                 ≀
   short    ( *udp_receive )( );
   short    rtn;
   struct    ureceive_p    *padr;
                 ≀
            udp_receive =(short(*) ( ) )0x874172;
                 ≀
            rtn = ( *udp_receive )( padr );
                 ≀
```

Sub

```
struct   ureceive_p {
   short    s_id;
   short    notuse;
   char     *buf;
   long     tim;
};
                 ≀
   short    ( *udp_receive )( );
   short    rtn;
   struct    ureceive_p    *padr;
                 ≀
            udp_receive =(short(*) ( ) )0x8F4172;
                 ≀
            rtn = ( *udp_receive )( padr );
                 ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> s_id: Socket ID

padr -> notuse: Fixed at 0 (unused)

padr -> buf: Starting address of receive buffer

padr -> tim: Receive wait time (0 to 86400000 ms [24 hours])

Output parameters:

Return value: The processing result or an error code is returned.

(0) Normal termination (no receive data)

(0x0001 to 0x05C0) Normal termination (number of received bytes)

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

NOTE
Because the udp_receive() function receives data in units of packets, reserve a buffer area of 1472 bytes.

route_list()

Function This function obtains routing information. (The maximum size in of the routing information table is 16 [routes].) The number of obtained entries is returned as the return value. If a 0 is specified for parameter len, only the number of obtained entries is returned. For len, specify a multiple of 16 (bytes).

Linking procedure

C language

Main

```
struct   lstrt_p {
   short    len;
   short    notues;
   void     *buf;
};
                 }
   short    ( *route_list)( );
   short    rtn;
   struct    lstrt_p    *padr;
                 {
              route_list = (short(*) ( ) )0x874178;
                 }
              rtn = ( *route_list )( padr );
                 }
```

Sub

```
struct   lstrt_p {
   short    len;
   short    notues;
   void     *buf;
};
                 }
   short    ( *route_list)( );
   short    rtn;
   struct    lstrt_p    *padr;
                 {
              route_list = (short(*) ( ) )0x8F4178;
                 }
              rtn = ( *route_list )( padr );
                 }
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> len: Data length (number of bytes; multiple of 16)

padr -> notes: Fixed at 0 (unused)

padr -> buf: Starting address of data

Output parameters:

Return value: The number of obtained entries is returned.

(0) No entry

(0x0001 to 0x0010) Number of obtained entries

Structure of obtained data (contents of buf):

```
typedef struct{
  unsingined long dstaddr: P address of remote station
  unsigined long getwayadder: IP address of gateway
  unsigined short metric: Metric (number of gateways passed)
  unsigined short rt_types: Type
  unsigineed short refcnt: Reference counter
  unsigined short notuse: (Unused)
}routeentry
```

route_del()

Function This function deletes routing information from the routing information table. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   delrt_p {
   long     dstaddr;
   long     gtwayaddr;
};
                 ≀
   short    ( *route_del )( );
   short    rtn;
   struct    delrt_p    *padr;
                 ≀
            route_del = (short(*) ( ) )0x87417E;
                 ≀
            rtn = ( *route_del )( padr );
                 ≀
```

Sub

```
struct   delrt_p {
   long     dstaddr;
   long     gtwayaddr;
};
                 ≀
   short    ( *route_del )( );
   short    rtn;
   struct    delrt_p    *padr;
                 ≀
            route_del = (short(*) ( ) )0x8F417E;
                 ≀
            rtn = ( *route_del )( padr );
                 ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> dstaddr: IP address of remote station

padr -> gtwayaddr: IP address of gateway

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

route_add()

Function This function adds routing information to the routing information table. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   addrt_p {
   long     dstaddr;
   long     gtwayaddr;
   short    metric;
};
                ﴿
   short    ( *route_add )( );
   short    rtn;
   struct    addrt_p    *padr;
                ﴿
            route_add = (short(*) ( ) )0x874184;
                ﴿
            rtn = ( *route_add )( padr );
                ﴿
```

Sub

```
struct   addrt_p {
   long     dstaddr;
   long     gtwayaddr;
   short    metric;
};
                ﴿
   short    ( *route_add )( );
   short    rtn;
   struct    addrt_p    *padr;
                ﴿
            route_add = (short(*) ( ) )0x8F4184;
                ﴿
            rtn = ( *route_add )( padr );
                ﴿
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> dstaddr: IP address of remote station

padr -> gtwayaddr: IP address of gateway

padr -> metric: Metric (number of gateways passed)

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

arp_list()

Function This function obtains ARP information. (The maximum size in of the ARP information table is 32 [ARPs].) The number of obtained entries is returned as the return value. If a 0 is specified for parameter len, only the number of obtained entries is returned. For len, specify a multiple of 12 (bytes).

Linking procedure

C language

Main

```
struct   lstarp_p {
   short    len;
   short    notuse;
   void     *buf;
};
                ≀
   short    ( *arp_list )( );
   short    rtn;
   struct    lstarp_p    *padr;
                ≀
            arp_list =(short (*)( ) )0x87418A;
                ≀
            rtn = ( *arp_list )( padr );
                ≀
```

Sub

```
struct   lstarp_p {
   short    len;
   short    notuse;
   void     *buf;
};
                ≀
   short    ( *arp_list )( );
   short    rtn;
   struct    lstarp_p    *padr;
                ≀
            arp_list =(short (*)( ) )0x8F418A;
                ≀
            rtn = ( *arp_list )( padr );
                ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> len: Data length (number of bytes; multiple of 12)

padr -> notuse: Fixed at 0 (unused)

padr -> buf: Starting address of data

Output parameters:

Return value: The number of obtained entries is returned.

(0) No entry

(0x0001 to 0x0020) Number of obtained entries

Structure of obtained data (contents of buf):

```
typedef struct{
  unsigined long ip_addr: IP address of remote station
  unsigined char et_addr(6): Physical address of remote station
  unsigined char ar_timer: Timer
  unsigined char ar_flags: Flag
}arpt-t
```

arp_del()

Function This function deletes ARP information from the ARP information table. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   delarp_p {
   unsigned   long   ipaddr;
   unsigned   char   etaddr[6];
};
                〉
   short    ( *arp_del )( );
   short    rtn;
   struct    delarp_p   *padr;
                〉
            arp_del =(short(*) ( ) ) 0x874190;
                〉
            rtn = ( *arp_del )( padr );
                〉
```

Sub

```
struct   delarp_p {
   unsigned   long   ipaddr;
   unsigned   char   etaddr[6];
};
                〉
   short    ( *arp_del )( );
   short    rtn;
   struct    delarp_p   *padr;
                〉
            arp_del =(short(*) ( ) ) 0x8F4190;
                〉
            rtn = ( *arp_del )( padr );
                〉
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> ipaddr: IP address of remote station

padr -> etaddr[6]: Physical address of remote station

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

arp_add()

Function This function adds ARP information to the ARP information table. The processing result is returned as the return value.

Linking procedure

C language

Main

```
struct   addarp_p {
   long      ipaddr;
   char      etaddr[6];
   short     flag;
};
                 ≀
   short     ( *arp_add )( );
   short     rtn;
   struct     addarp_p    *padr;
                 ≀
             arp_add =(short(*) ( ) )0x874196;
                 ≀
             rtn = ( *arp_add )( padr );
                 ≀
```

Sub

```
struct   addarp_p {
   long      ipaddr;
   char      etaddr[6];
   short     flag;
};
                 ≀
   short     ( *arp_add )( );
   short     rtn;
   struct     addarp_p    *padr;
                 ≀
             arp_add =(short(*) ( ) )0x8F4196;
                 ≀
             rtn = ( *arp_add )( padr );
                 ≀
```

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> ipaddr: IP address of remote station

padr -> etaddr[6]: Physical address of remote station

padr -> flag: Flag (fixed at 0)

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

(0xF000 to 0xFFFF) Error code

For error codes, see Section 7.3, “Errors and Actions To Be Taken.”

getconfig()

Function This function obtains the configuration blocks. The processing result is returned as the return value.

Linking procedure

C language

Main	Sub

struct config_p {
void *config_ptr;
};
〜
short (*getconfig)( );
short rtn;
struct config_p *padr;
〜
getconfig = (short(*) ( ))0x87419C;
〜
rtn = (*getconfig)(padr);
〜
	struct config_p {
	void *config_ptr;
	};
	〜
	short (*getconfig)( );
	short rtn;
	struct config_p *padr;
	〜
	getconfig = (short(*) ( ))0x8F419C;
	〜
	rtn = (*getconfig)(padr);
	〜

Parameters

Input parameters:

padr: Starting address of input parameters

padr -> config_ptr: Starting address of configuration block

Output parameters:

Return value: The processing result is returned.

(0) Normal termination

Configuration block:

Data structure of configuration block

struct config_ptr{

long ip_addr: IP address (network order) of local station (optional)

long netmask: Subnetwork mask (optional)

long broadcast: Broadcast address (optional)

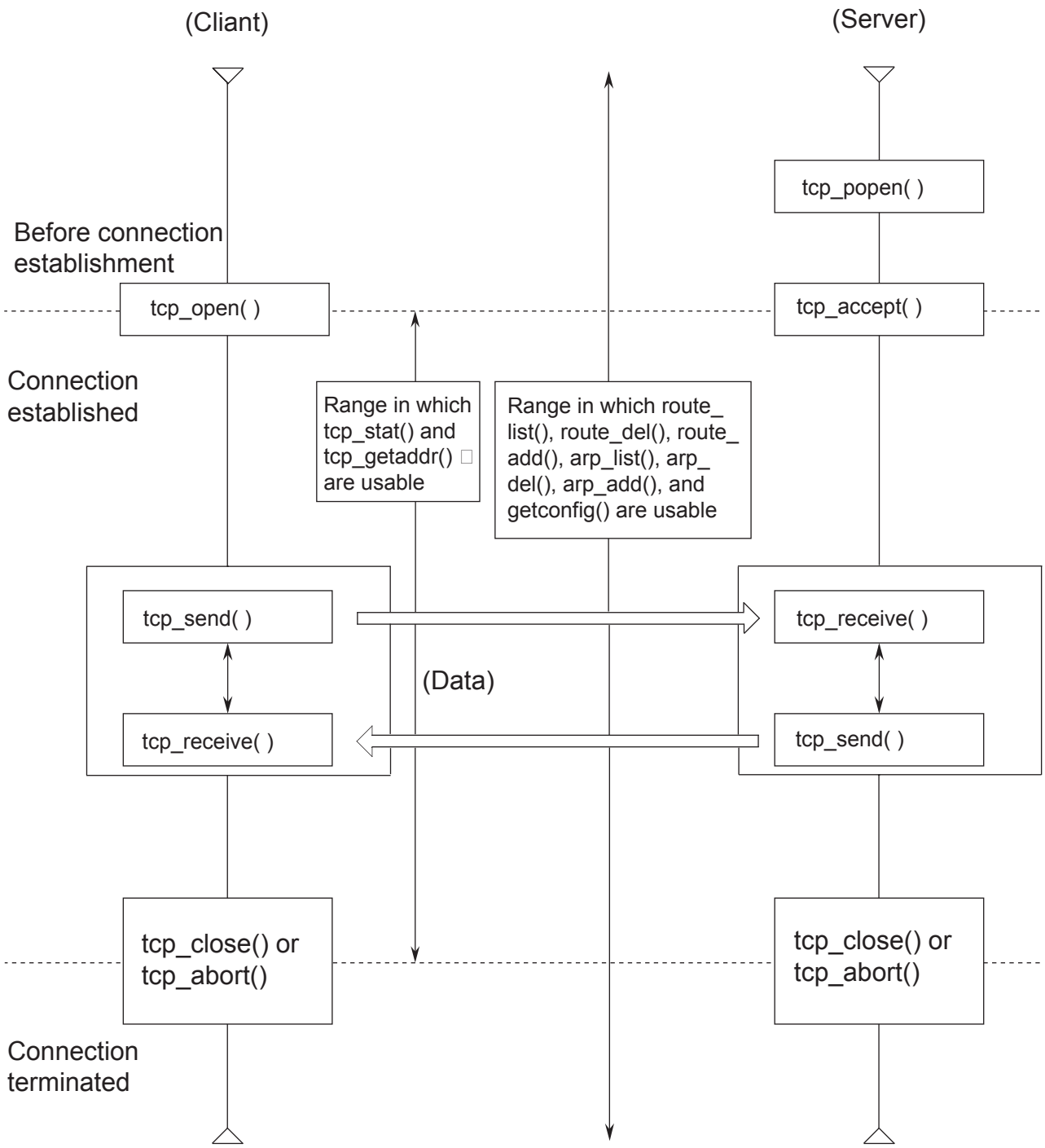
char tcp_num: Maximum number of TCP sockets (16)

char udp_num: Maximum number of UDP sockets (8)

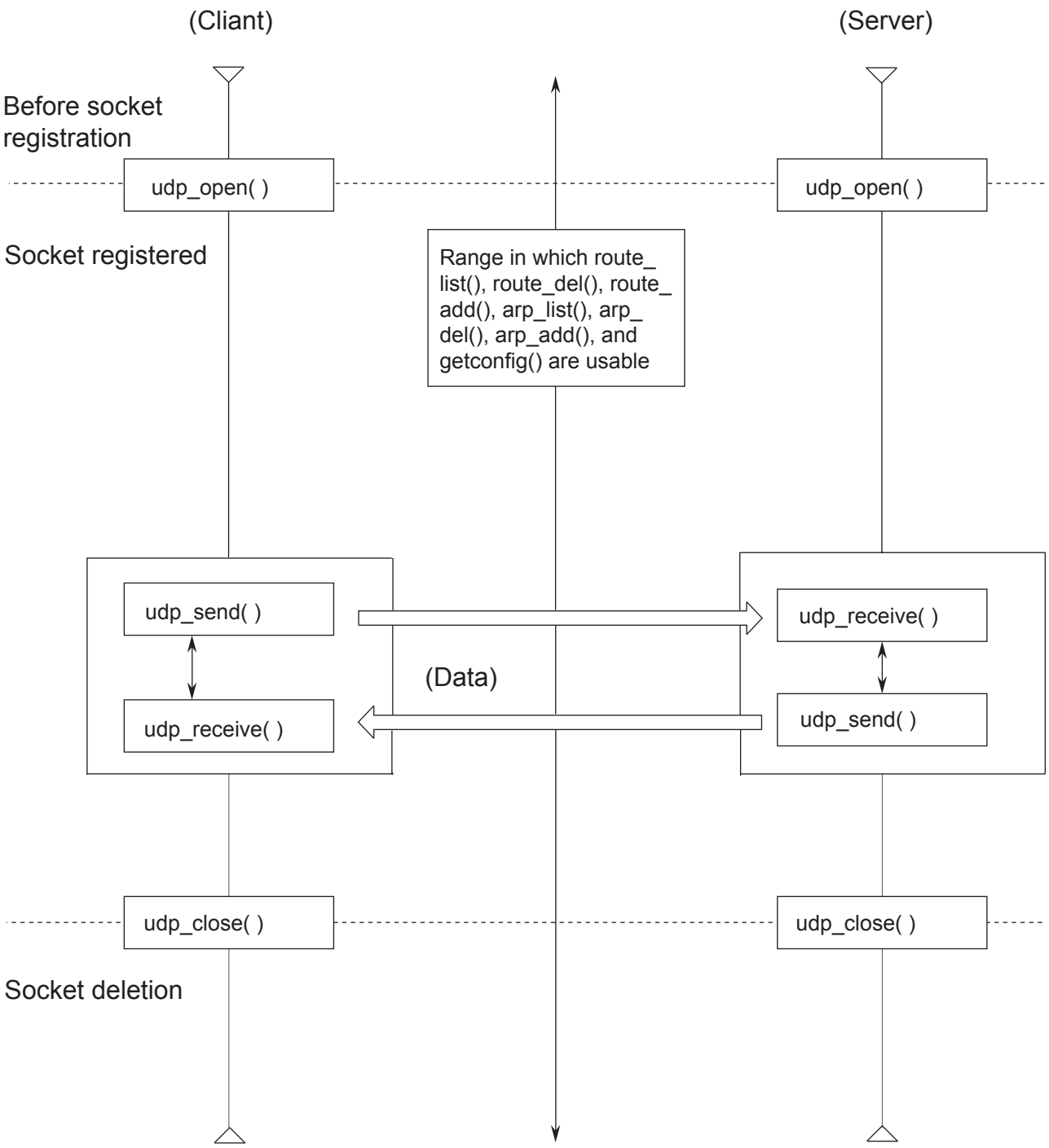
```
  char rt_num: Size of routing information table (16)
  char arp_num: Size of ARP information table (32)
  short tcp_win: Size of TCP send/receive window (1024)
};
```

4.9 Examples of Socket Handler Issuance Procedure

4.9.1 Example of using TCP/IP program



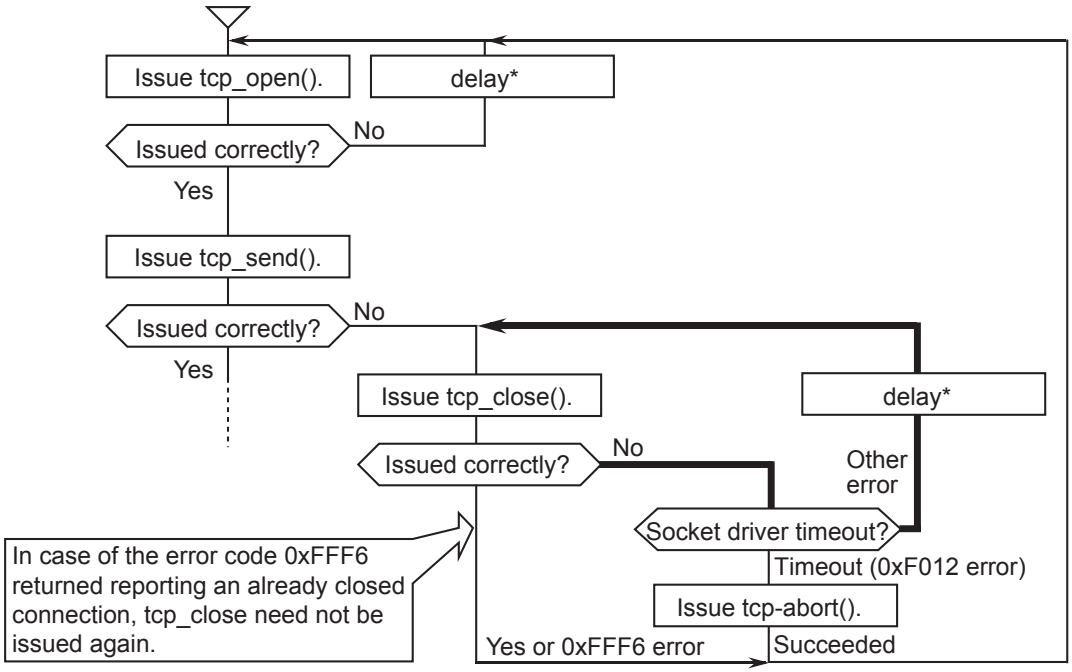
4.9.2 Example of using UDP/IP program



NOTE

Note the following points before using the S10mini Ethernet module LQE020:

1. Error handling for tcp_close
   You may issue tcp_close when the return code from a socket handler function indicates an error. If you have issued it, also check the return code from tcp_close. If the code indicates an error, issue tcp_close again as indicated in the table, which lists codes associated with errors detected by the socket handler, in order to eliminate the cause of the error. Otherwise, a connection may not be established again or a floating socket may be generated. An example of programming (flowchart) showing how the socket handler issues socket library functions is given below.



Note: This flowchart also applies to error handling for udp_close.

2. Inhibited asynchronous access to the same socket
   Multiple socket library functions asynchronously issued to a single socket may result in incorrect execution results of functions. This problem is likely to occur when multiple tasks issue socket library functions to the same socket. Make sure that one task handles one socket.

* For the delay macro instruction, refer to the SOFTWARE MANUAL GENERAL DESCRIPTION & MACROS COMPACT PMS V5 (manual number SAE-3-201).

NOTE

3. Time to detect a transmission timeout
   When the LQE020 issues a socket library function, an ACK packet may cause a timeout due to a communication error or a failure in the remote device. It takes time to detect a timeout as indicated in the table below. Therefore, at least the time in the table is required after a timeout of the socket handler is detected before the socket library function is issued again or a connection is established again. Assuming that communication errors are inevitable, confirm at the design stage that the timeout values in the table do not cause a problem.

Time	Value	Description
Time to detect a tcp_open timeout (SYN retry interval)	75s	When receiving no response from the remote device, the socket handler retries SYN at the following intervals: 6s, 12s, 24s, and 33s
Time to detect a tcp_send timeout (SEND retry interval)	30s	When receiving no response from the remote device, the socket handler retransmits at the following intervals: 1s, 2s, 4s, 8s, and 16s If 30 seconds pass after the socket handler has issued tcp_send, the socket handler detects a socket driver timeout (return code: 0xF012).
Time to detect a tcp_close timeout (FIN retry interval)	30s	When receiving FIN from the remote device and detecting the normal line disconnection, the socket handler ends immediately. When the module (LQE020) sends FIN to disconnect the line, the socket handler also ends immediately. When receiving no response from the remote device, the socket handler retries FIN at the following intervals: 1s, 2s, 4s, 8s, and 16s If 30 seconds pass after the socket handler has issued tcp_close, the socket handler detects a socket driver timeout (return code: 0xF012). Issue tcp_abort to disconnect the line.

(cont.)

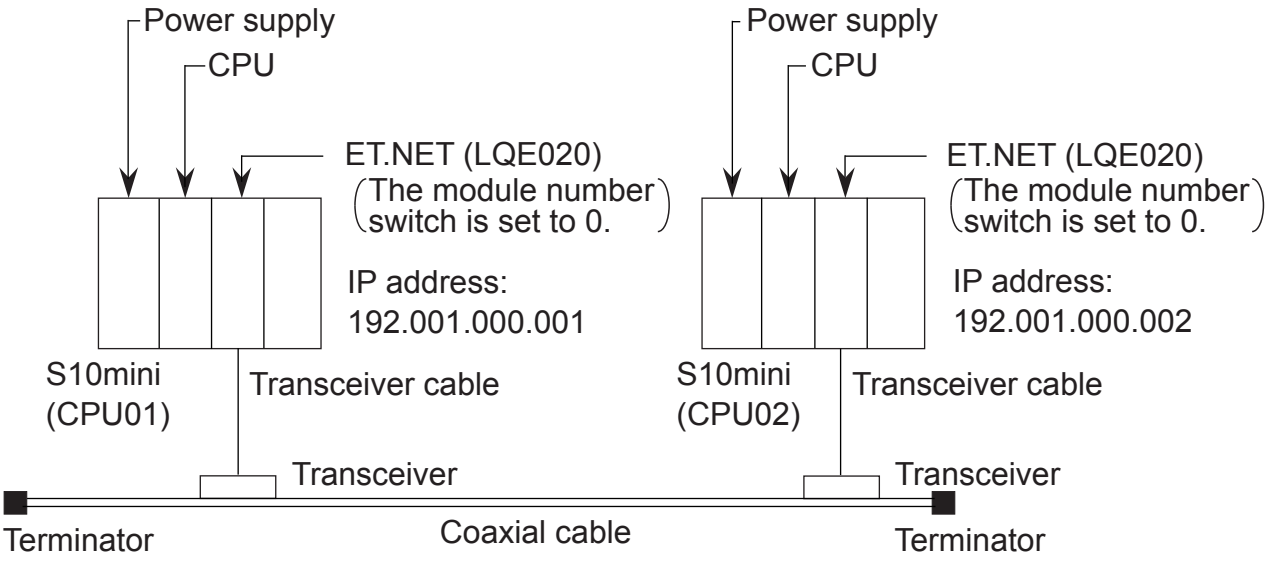
NOTE

Time		Value	Description
Time to detect a response timeout	tcp_close,tcp_send, udp_close	30s	Time from when the socket handler issues a command to a microprogram until it is judged that there is no response.
	tcp_abort,route_list, route_del,route_add, arp_list,arp_del, arp_add,getconfig, udp_send,tcp_getaddr, tcp_stat	10s	

5 PROGRAM EXAMPLES

5.1 Example of Programs for Communication between CPUs by Socket Handler

5.1.1 System configuration

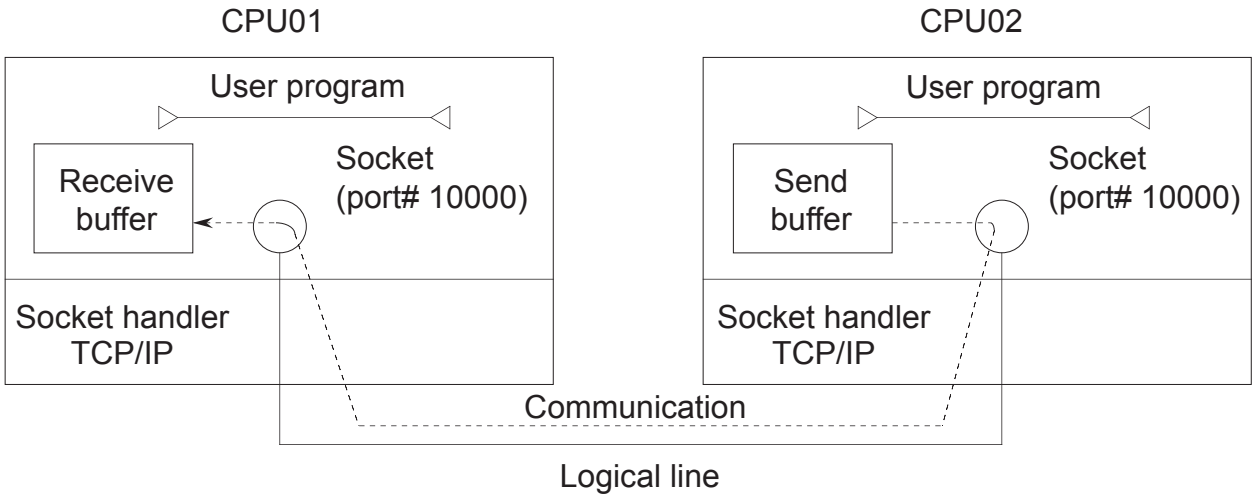


System components list

Product name	Model	Quantity	Remarks
Power supply	LQV000	2	
CPU	LQP010	2	
ET.NET	LQE020	2	
Transceiver cable	HBN-TC-100	2	Manufactured by Hitachi Cable, Ltd.
Transceiver	HLT-200TB	2	Manufactured by Hitachi Cable, Ltd.
Coaxial cable	HBN-CX-100	1	Manufactured by Hitachi Cable, Ltd.
Terminator	HBN-T-NJ	2	Manufactured by Hitachi Cable, Ltd.

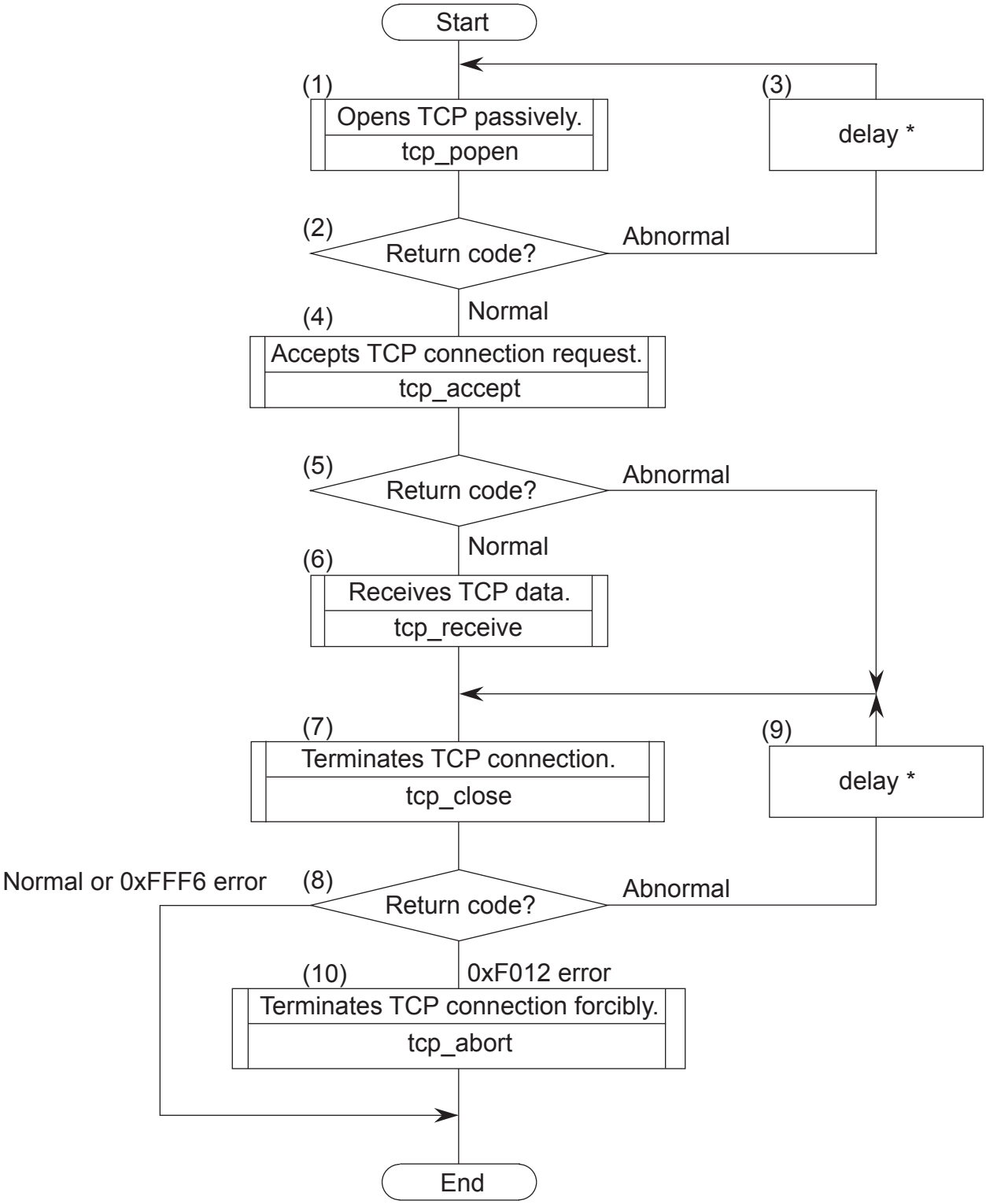
5.1.2 Program structure

The program structure is shown below. The ET.NET module of CPU01 and that of CPU02 are connected by logical line. The ET.NET module of CPU02 sends 1024 bytes of data, and the ET.NET module of CPU01 receives that much data.
When operating this program, start the user program from CPU01.



Item \ CPU		CPU01	CPU02
Function		Reception	Transmission
Send buffer		———	Address: 0x1E6000 Number of bytes: 1024
Receive buffer		Address: 0x1E6000 Number of bytes: 1024	———
Port number		10000	10000
Starting address of socket handler	tcp_open()	0x874100	0x874100
	tcp_popen()	0x874106	0x874106
	tcp_accept()	0x87410C	0x87410C
	tcp_close()	0x874112	0x874112
	tcp_abort()	0x87411E	0x87411E
	tcp_getaddr()	0x874124	0x874124
	tcp_stat()	0x87412A	0x87412A
	tcp_send()	0x874130	0x874130
	tcp_receive()	0x874136	0x874136
	udp_open()	0x874160	0x874160
	udp_close()	0x874166	0x874166
	udp_send()	0x87416C	0x87416C
	udp_receive()	0x874172	0x874172
	route_list()	0x874178	0x874178
	route_del()	0x87417E	0x87417E
	route_add()	0x874184	0x874184
	arp_list()	0x87418A	0x87418A
	arp_del()	0x874190	0x874190
	arp_add()	0x874196	0x874196
	getconfig()	0x87419C	0x87419C

5.1.3 Flowchart of program at CPU01



(1) Registers a socket with port number 10000, and puts the socket into passive state.
(2) The registered socket ID is returned as the return code. When the return code is normal, it is regarded that the socket has been registered normally.
(3) Issues the delay macro, then repeats processes (1) and (2).
(4) Accepts the connection request from CPU02.
(5) Judges whether normal or abnormal by the return code.
(6) Reads the data sent from CPU02 into the receive buffer.
(7) Terminates the established connection.
(8) Judges whether normal or abnormal by the return code. When the return code is 0xFFF6 (error), terminates the program as if no error had occurred normally. When the return code is 0xF012(error), then executes process (10).
(9) Issues the delay macro, then repeats processes (7) and (8).
(10) Terminates the connection as no response is returned from the remote station.

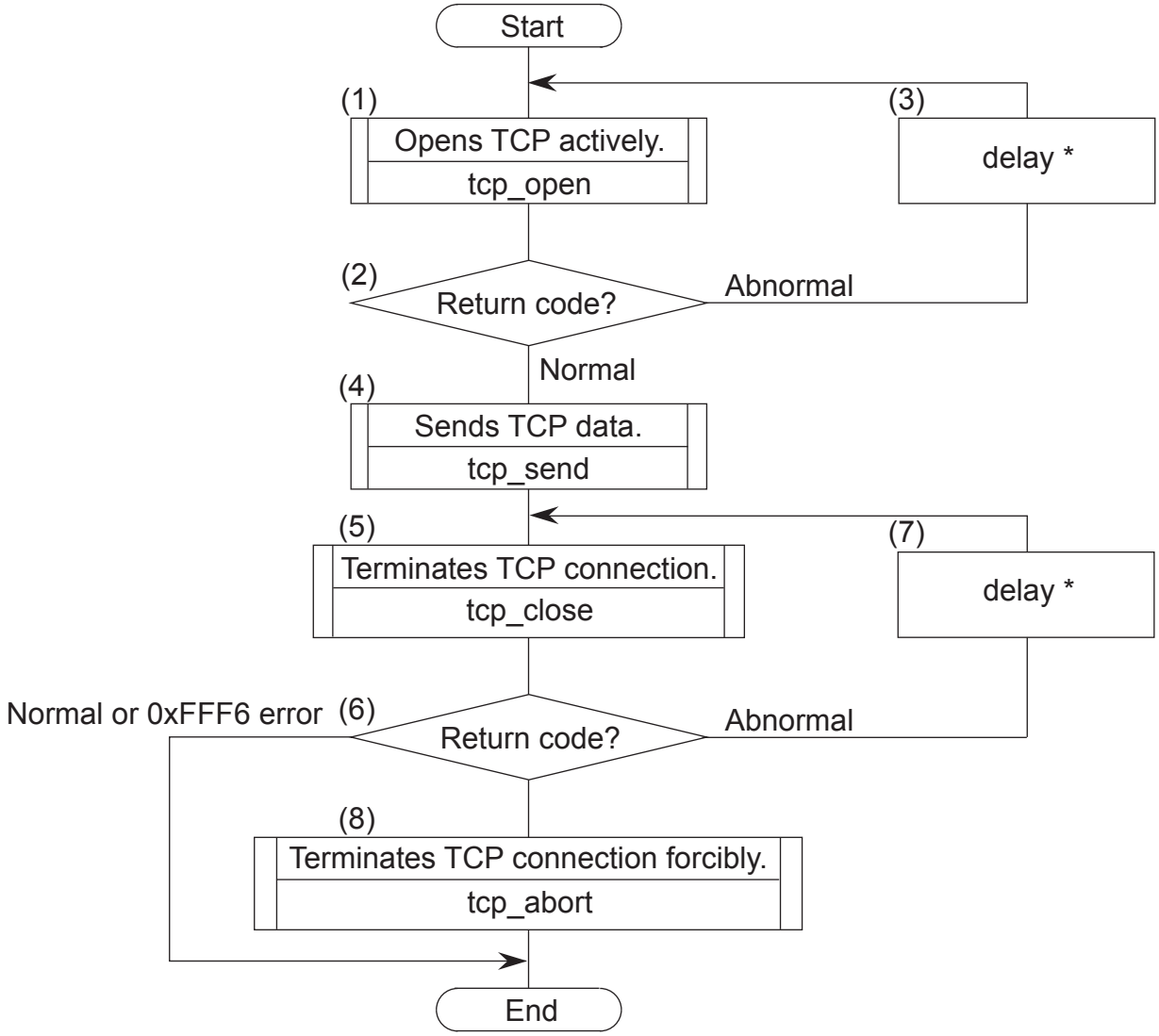
* For the delay macro instruction, refer to the SOFTWARE MANUAL GENERAL DESCRIPTION & MACROS COMPACT PMS V5 (manual number SAE-3-201).

5.1.4 Example of C language program at CPU01

```
#define TCP_POPEN   0x874106L   /* tcp_popen( )   starting address(main)     */
#define TCP_ACCEPT  0x87410CL   /* tcp_accept( )  starting address(main)     */
#define TCP_CLOSE   0x874112L   /* tcp_close( )   starting address(main)     */
#define TCP_RECEIVE 0x874136L   /* tcp_receive( ) starting address(main)     */
#define TCP_ABORT   0x87411EL   /* tcp_abort( )   starting address           */
#define IPADDR      0xC0010002L /* IP address of remote station              */
#define RBUFADDR    0x1E6000L   /* Starting address of receive buffer        */
#define PARADDR     0x1E5000L   /* Starting address of parameter strage area */

struct  popen_p{
    long    dst_ip;         /* IP address of remote station             */
    short   dst_port;       /* Port number of remote station            */
    short   src_port;       /* Port number of local station             */
    char    listennum;      /* Maximum number of unaccepted connections */
    char    ttl;            /* Time to live                             */
};

struct  accept_p{
    short   s_id;           /* Socket ID                                */
};

struct  receive_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Buffer length                            */
    char    *buf;           /* Starting address of buffer               */
    long    tim;            /* Receive wait time(ms)                    */
};

struct  close_p{
    short   s_id;           /* Socket ID                                */
};
struct  abort_p{
    short   s_id;           /* Socket ID                                */
};
/************************/
/* task2: Server(CPU01) */
/************************/
main()
{
    register    short ( *tcp_popen )( );
    register    short ( *tcp_accept )( );
    register    short ( *tcp_receive )( );
    register    short ( *tcp_close )( );
    register    short ( *tcp_abort )( );
    long    time;
    short   rtn;
    char    *rbuf;
    struct  popen_p     *popen;
    struct  accept_p    *accpt;
    struct  receive_p   *recv;
    struct  close_p     *close;
    struct  abort_p     *abort;

    popen = (struct popen_p   *)PARADDR;       /* Starting address of input parameter storage area */
    accpt = (struct accept_p  *)(popen + 1);
    recv  = (struct receive_p *)(accpt + 1);
    close = (struct close_p   *)(recv  + 1);
    abort = (struct abort_p   *)(close + 1);

    while( 1 ){
        popen->dst_ip    = IPADDR;             /* IP address of remote station     */
        popen->dst_port  = 10000;              /* Port number of remote station    */
        popen->src_port  = 10000;              /* Port number of local station     */
        popen->listennum = 0;                  /* Maximum number of                */
                                               /* unaccepted connections           */
        popen->ttl       = 0;                  /* Time to live                     */
        tcp_popen  = ( short (*)())TCP_POPEN;
        rtn        = (tcp_popen)(popen);       /* Opens TCP passively              */
        if( rtn > 0 ){                         /* Return code normal?              */
```

```
            break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro      */
        delay( &time);
    }
    accpt->s_id  = rtn;                     /* Socket ID                        */
    tcp_accept    = ( short (*)())TCP_ACCEPT;
    rtn           = (tcp_accept)(accpt);    /* Accepts TPC connection request.  */
    recv->s_id    = rtn;                    /* Socket ID                          */
    if( rtn > 0 ){                          /* Return code normal?                */
        recv->len = 1024;                   /* Receive buffer length(bytes)       */
        recv->buf   = ( char *)RBUFADDR;    /* Starting address of receive buffer */
        recv->tim = 60000;                  /* Receive wait time(ms)              */
        tcp_receive = ( short (*)())TCP_RECEIVE;
        rtn         = (tcp_receive)(recv);  /* Receives TCP                       */
        close->s_id = recv->s_id;           /* Socket ID                          */
    } else {
        close->s_id = accpt->s_id;          /* Socket ID                          */
    }
    while( 1 ){
        tcp_close = ( short (*)())TCP_CLOSE;
        rtn = (tcp_close)(close);           /* Terminates TCP connection.         */
        if( rtn == 0 || rtn == ( short )0xFFF6 ){
            break;
    } else if ( rtn == (short )0xF012 ) {
         tcp_abort = ( short (*)())TCP_ABORT;
         rtn = (tcp_abort)(abort);          /* Terminates TCP connection forcibly */
         break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro        */
        delay( &time);
    }
    return;
}
```

5.1.5 Flowchart of program at CPU02



(1) Registers a socket with port number 10000, and puts the socket into active state.
(2) The registered socket ID is returned as the return code. When the return code is normal, it is regarded that the socket has been registered normally.
(3) Issues the delay macro, then repeats processes (1) and (2).
(4) Transmits the data in the send buffer to CPU01.
(5) Terminates the established connection.
(6) Judges whether normal or abnormal by the return code. When the return code is 0xFFF6 (error), terminates the program as if no error had occurred normally. When the return code is 0xF012(error), then executes process (8).
(7) Issues the delay macro, then repeats processes (5) and (6).
(8) Terminates the connection as no response is returned from the remote station.

* For the delay macro instruction, refer to the SOFTWARE MANUAL GENERAL DESCRIPTION & MACROS COMPACT PMS V5 (manual number SAE-3-201).

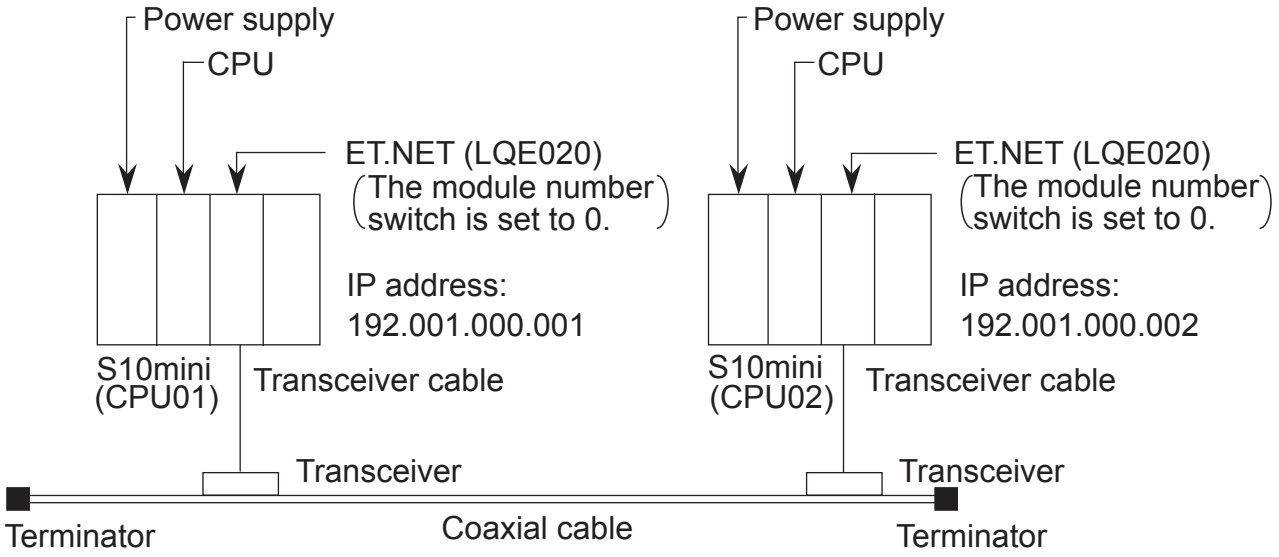
5.1.6 Example of C language program at CPU02

```
#define TCP_OPEN    0x874100L   /* tcp_open( )    starting address          */
#define TCP_CLOSE   0x874112L   /* tcp_close( )   starting address          */
#define TCP_SEND    0x874130L   /* tcp_send( )    starting address          */
#define TCP_ABORT   0x87411EL   /* tcp_abort( )   starting address          */
#define IPADDR      0xC0010001L /* IP address of remote station             */
#define SBUFADDR    0x1E6000L   /* Starting address of send buffer          */
#define PARADDR     0x1E5000L   /* Starting address of parameter strage area*/

struct  open_p{
    long    dst_ip;         /* IP address of remote station             */
    short   dst_port;       /* Port number of remote station            */
    short   src_port;       /* Port number of local station             */
    char    notuse;         /* Unused(0)                                */
    char    ttl;            /* Time to live                             */
};

struct  send_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Send data length(bytes)                  */
    char    *buf;           /* Starting address of send data            */
};

struct  close_p{
    short   s_id;           /* Socket ID                                */
};
struct  abort_p{
    short   s_id;           /* Socket ID                                */
};
/************************/
/*  task3: Client(CPU02) */
/************************/
main()
{
    register    short ( *tcp_open )( );
    register    short ( *tcp_send )( );
    register    short ( *tcp_close )( );
    register    short ( *tcp_abort )( );
    long    time;
    short   rtn;
    struct  open_p      *open;
    struct  send_p      *send;
    struct  close_p     *close;
    struct  abort_p     *abort;

    open  = (struct open_p    *)PARADDR;    /* Starting address of input parameter storage area */
    send  = (struct send_p    *)(open  + 1);
    close = (struct close_p   *)(send  + 1);
    abort = (struct abort_p   *)(close + 1);

    while( 1 ){
        open->dst_ip    = IPADDR;           /* IP address of remote station  */
        open->dst_port  = 10000;            /* Port number of remote station */
        open->src_port  = 10000;            /* Port number of local station  */
        open->notuse    = 0;                /* Unused                        */
        open->ttl       = 0;                /* Time to live                  */
        tcp_open = (short (*)())TCP_OPEN;
        rtn      = (tcp_open)(open);        /* Opens TCP actively.           */
        if( rtn > 0 ){                      /* Return code normal?           */
            break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro   */
        delay( &time);
```

```
    }
    send->s_id  = rtn;                      /* Socket ID                      */
    send->len   = 1024;                     /* Send data length(bytes)        */
    send->buf   = ( char *)SBUFADDR;        /* Starting address of send data  */
    tcp_send    = ( short (*)())TCP_SEND;
    rtn         = (tcp_send)(send);         /* Sends TCP data.                */
    close->s_id = send->s_id;               /* Socket ID                      */
    while( 1 ){
        tcp_close   = ( short (*)())TCP_CLOSE;
        rtn         = (tcp_close)(close);   /* Terminates TCP connection.     */
        if( rtn == 0 || rtn == ( short)0xFFF6 ){
            break;
    } else if ( rtn == (short )0xF012 ) {
         tcp_abort = ( short (*)())TCP_ABORT;
         rtn = (tcp_abort)(abort);          /* Terminates TCP connection forcibly */
         break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro    */
        delay( &time);
    }
    return;
}
```

5.2 Example of Programs for Continuous Communication between CPUs by Socket Handler

5.2.1 System configuration

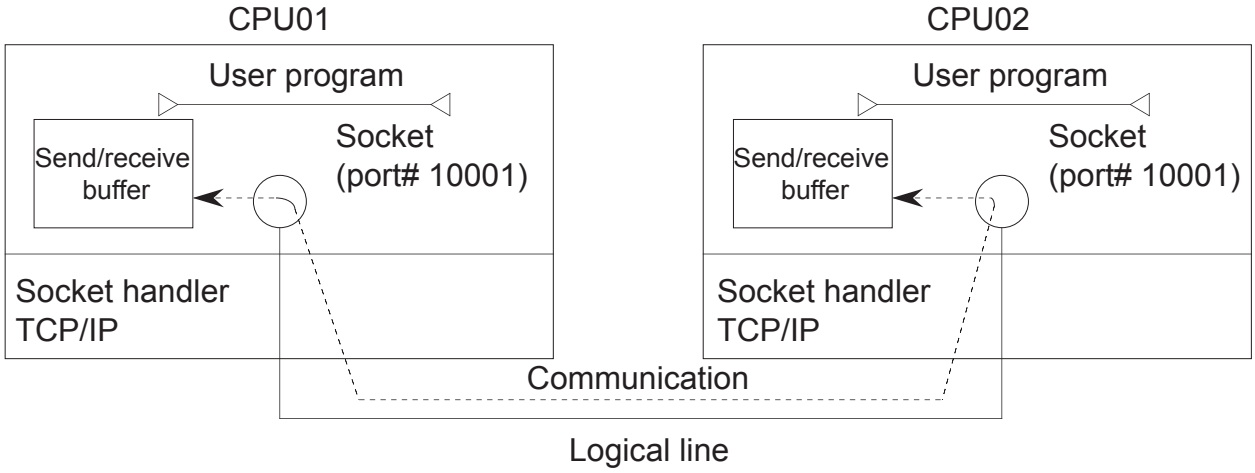


System Components List

Product name	Model	Quantity	Remarks
Power supply	LQV000	2	
CPU	LQP010	2	
ET.NET	LQE020	2	
Transceiver cable	HBN-TC-100	2	Manufactured by Hitachi Cable, Ltd.
Transceiver	HLT-200TB	2	Manufactured by Hitachi Cable, Ltd.
Coaxial cable	HBN-CX-100	1	Manufactured by Hitachi Cable, Ltd.
Terminator	HBN-T-NJ	2	Manufactured by Hitachi Cable, Ltd.

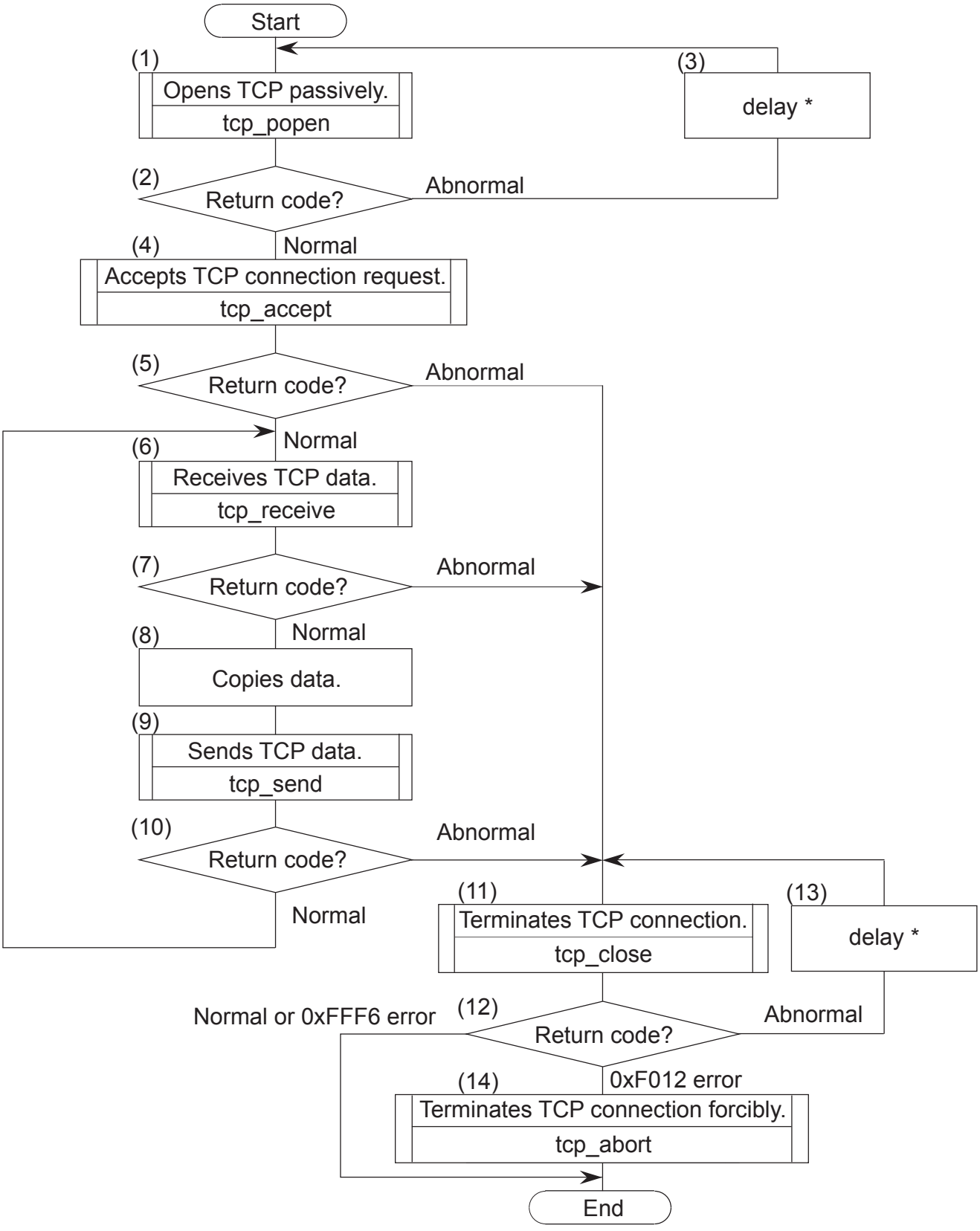
5.2.2 Program structure

The program structure is shown below. The ET.NET module of CPU01 and that of CPU02 are connected by logical line. A total of 1024 bytes of data is transmitted between the ET.NET module of CPU02 and that of CPU01.
When running this program, start the user program from CPU01.



Item \ CPU		CPU01	CPU02
Function		Transmission/reception	Transmission/reception/ comparison
Send buffer		Address: 0x1E1000 Number of bytes: 1024	Address: 0x1E1000 Number of bytes: 1024
Receive buffer		Address: 0x1E2000 Number of bytes: 1024	Address: 0x1E2000 Number of bytes: 1024
Port number		10001	10001
Starting address of socket handler	tcp_open()	0x874100	0x874100
	tcp_popen()	0x874106	0x874106
	tcp_accept()	0x87410C	0x87410C
	tcp_close()	0x874112	0x874112
	tcp_abort()	0x87411E	0x87411E
	tcp_getaddr()	0x874124	0x874124
	tcp_stat()	0x87412A	0x87412A
	tcp_send()	0x874130	0x874130
	tcp_receive()	0x874136	0x874136
	udp_open()	0x874160	0x874160
	udp_close()	0x874166	0x874166
	udp_send()	0x87416C	0x87416C
	udp_receive()	0x874172	0x874172
	route_list()	0x874178	0x874178
	route_del()	0x87417E	0x87417E
	route_add()	0x874184	0x874184
	arp_list()	0x87418A	0x87418A
	arp_del()	0x874190	0x874190
	arp_add()	0x874196	0x874196
	getconfig()	0x87419C	0x87419C

5.2.3 Flowchart of program at CPU01



(1) Register a socket with port number 10001, and put the socket into passive state.
(2) The registered socket ID is returned as the return code. When the return code is normal, it is regarded that the socket has been registered normally.
(3) Issues the delay macro, then repeats processes (1) and (2).
(4) Accept the connection request from CPU02.
(5) Judge whether normal or abnormal by the return code.
(6) Read the data sent from CPU02 into the receive buffer.
(7) When the return code is an error code or it indicates that there is no read data, jump to step (11).
(8) Copy the data in the receive buffer into the transmission buffer.
(9) Send the data in the send buffer to CPU02.
(10) Judge whether normal or abnormal by the return code. When normal, repeats (6) to (10).
(11) Terminate the established connection.
(12) Judges whether normal or abnormal by the return code. When the return code is 0xFFF6 (error), terminates the program as if no error had occurred normally. When the return code is 0xF012(error), then executes process (14).
(13) Issues the delay macro, then repeats processes (11) and (12).
(14) Terminates the connection as no response is returned from the remote station.

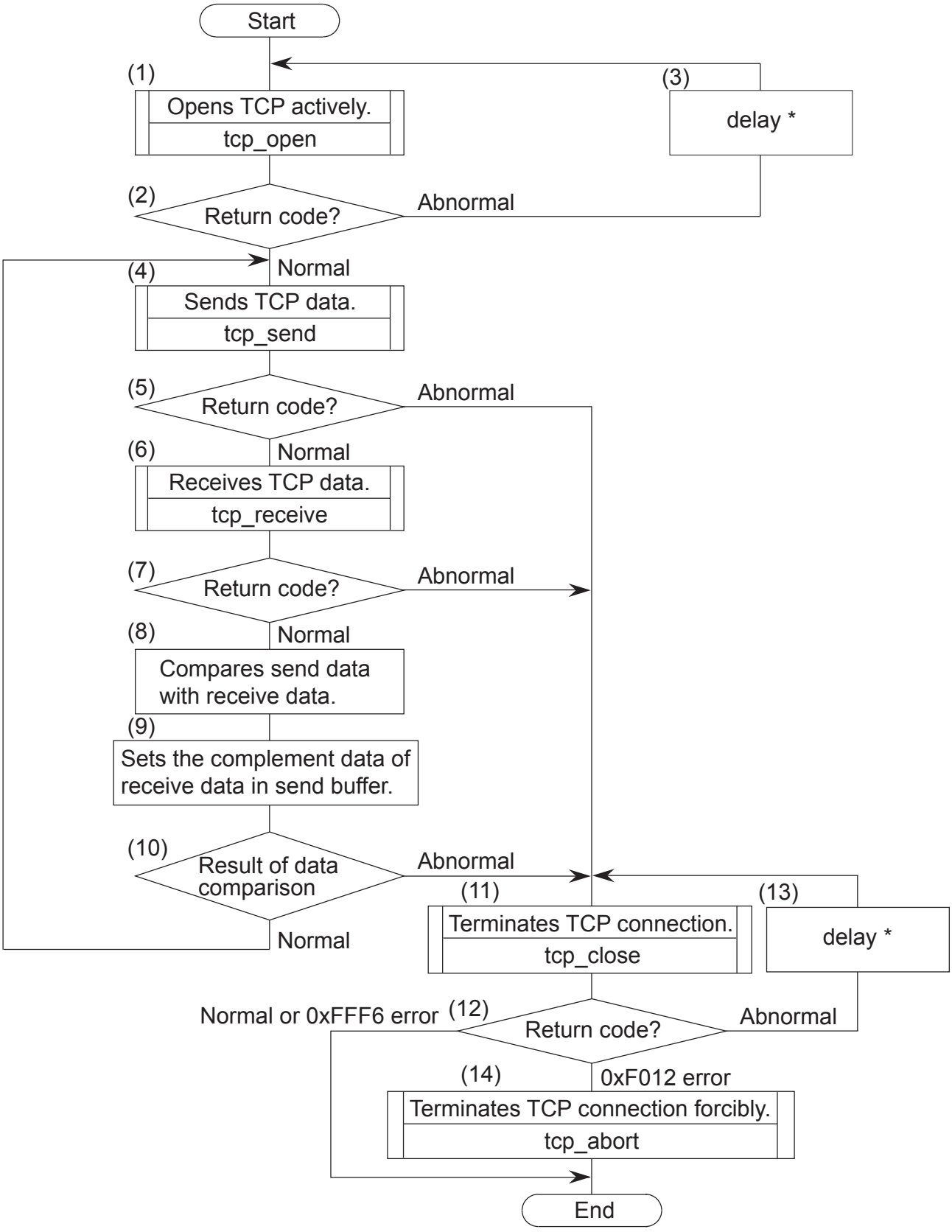
* For the delay macro instruction, refer to the SOFTWARE MANUAL GENERAL DESCRIPTION & MACROS COMPACT PMS V5 (manual number SAE-3-201).

5.2.4 Example of C language program at CPU01

```
#define TCP_POPEN   0x874106L   /* tcp_popen( )   starting address(main)      */
#define TCP_ACCEPT  0x87410CL   /* tcp_accept( )  starting address(main)      */
#define TCP_RECEIVE 0x874136L   /* tcp_receive( ) starting address(main)      */
#define TCP_SEND    0x874130L   /* tcp_send( )    starting address(main)      */
#define TCP_CLOSE   0x874112L   /* tcp_close( )   starting address(main)      */
#define TCP_ABORT   0x87411EL   /* tcp_abort( )   starting address            */
#define IPADDR      0xC0010002L /* IP address of remote station               */
#define SBUFADDR    0x1E1000L   /* Starting address of send buffer            */
#define RBUFADDR    0x1E2000L   /* Starting address of receive buffer         */
#define PARADDR     0x1E5000L   /* Starting address of parameter storage area */

struct  popen_p{
    long    dst_ip;         /* IP address of remote station             */
    short   dst_port;       /* Port number of remote station            */
    short   src_port;       /* Port number of local station             */
    char    listennum;      /* Maximum number of unaccepted connections */
    char    ttl;            /* Time to live                             */
};

struct  accept_p{
    short   s_id;           /* Socket ID                                */
};

struct  receive_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Buffer length                            */
    char    *buf;           /* Starting address of buffer               */
    long    tim;            /* Receive wait time(ms)                    */
};

struct send_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Send data length(bytes)                  */
    char    *buf;           /* Starting address of send data            */
};

struct  close_p{
    short   s_id;           /* Socket ID                                */
};
struct  abort_p{
    short   s_id;           /* Socket ID                                */
};
/*************************/
/*  task2: Server(CPU01)  */
/*************************/
main()
{
    register    short ( *tcp_popen )( );
    register    short ( *tcp_accept )( );
    register    short ( *tcp_receive )( );
    register    short ( *tcp_send )( );
    register    short ( *tcp_close )( );
    register    short ( *tcp_abort )( );
    long    time;
    short   rtn, i;
    char    *sbuf, *rbuf;
    struct  popen_p     *popen;
    struct  accept_p    *accpt;
    struct  receive_p   *recv;
    struct  send_p      *send;
    struct  close_p     *close;
    struct  abort_p     *abort;

    popen = (struct popen_p   *)PARADDR;        /* Starting address of input parameter storage area */
    accpt = (struct accept_p  *)(popen + 1);
    recv  = (struct receive_p *)(accpt + 1);
    send  = (struct send_p    *)(recv  + 1);
    close = (struct close_p   *)(send  + 1);
    abort = (struct abort_p   *)(close + 1);
```

```
    while( 1 ){
        popen->dst_ip    = IPADDR;              /* IP address of remote station   */
        popen->dst_port  = 10001;               /* Port number of remote station  */
        popen->src_port  = 10001;               /* Port number of local station   */
        popen->listennum = 0;                   /* Maximum number of              */
                                                /* unaccepted connections         */
        popen->ttl       = 0;                   /* Time to live                   */
        tcp_popen       = ( short (*)())TCP_POPEN;
        rtn             = (tcp_popen)(popen);   /* Opens TCP passively.                */
        if( rtn > 0 ){                          /* Return code normal?                 */
            break;
        }
        time = 100;                             /* Issue of 100-ms Delay macro         */
        delay( &time);
    }
    accpt->s_id    = rtn;                       /* Socket ID                           */
    tcp_accept    = ( short (*)())TCP_ACCEPT;
    rtn           = (tcp_accept)(accpt);        /* Accepts TCP connection request.     */
    if( rtn > 0 ){                              /* Return code normal?                 */
        recv->s_id    = rtn;                    /* SocketID                            */
        while( 1 ){
            recv->len = 1024;                   /* Receive buffer length(bytes)        */
            recv->buf = ( char*)RBUFADDR;       /* Starting address of receive buffer  */
            recv->tim = 60000;                  /* Receive wait time(ms)               */
            tcp_receive = ( short (*)())TCP_RECEIVE;
            rtn         = (tcp_receive)(recv);  /* Receives TCP data.                  */
            if( rtn < 0){                       /* Return code abnormal?               */
                break;
            }
            sbuf = ( char *)SBUFADDR;           /* Starting address of send buffer     */
            rbuf = ( char *)RBUFADDR;           /* Starting address of receive buffer  */
            for( i = 0 ; i < 1024 ; i++ ){
                sbuf[i] = rbuf[i];
            }
            send->s_id = recv->s_id;            /* Socket ID                           */
            send->len  = 1024;                  /* Send data length(bytes)             */
            send->buf  = ( char *)SBUFADDR;     /* Starting address of send data       */
            tcp_send  = ( short (*)())TCP_SEND;
            rtn       = (*tcp_send)(send);      /* Sends TCP data.                     */
            if( rtn < 0 ){                      /* Return code abnormal?               */
                break;
            }
        }
        close->s_id = recv->s_id;               /* Socket ID                           */
    } else {
        close->s_id = accpt->s_id;              /* Socket ID                           */
    }
    while( 1 ){
        tcp_close = ( short (*)())TCP_CLOSE;
        rtn = (tcp_close)(close);               /* Terminates TCP connection.          */
        if( rtn == 0 || rtn == ( short )0xFFF6 ){
            break;
    } else if ( rtn == (short )0xF012 ) {
         tcp_abort = ( short (*)())TCP_ABORT;
         rtn = (tcp_abort)(abort);              /* Terminates TCP connection forcibly  */
         break;
        }
        time = 100;                             /* Issue of 100-ms Delay macro         */
        delay( &time);
    }
    return;
}
```

5.2.5 Flowchart of program at CPU02



(1) Register a socket with port number 10001, and put the socket into active state.
(2) The registered socket ID is returned as the return code. When the return code is normal, it is regarded that the socket has been registered normally.
(3) Issues the delay macro, then repeats processes (1) and (2).
(4) Send the data in the send buffer to CPU01.
(5) Judge whether normal or abnormal by the return code.
(6) Read the data sent from CPU01 into the receive buffer.
(7) Judge whether normal or abnormal by the return code.
(8) Compare the data in the send buffer with that in the receive buffer of the local station.
(9) Copy the complement of the receive data into the send buffer.
(10) Judge whether the comparison result is normal or abnormal. When it is normal, repeats steps (4) to (10).
(11) Terminate the established connection.
(12) Judges whether normal or abnormal by the return code. When the return code is 0xFFF6 (error), terminates the program as if no error had occurred normally. When the return code is 0xF012(error), then executes process (14).
(13) Issues the delay macro, then repeats processes (11) and (12).
(14) Terminates the connection as no response is returned from the remote station.

* For the delay macro instruction, refer to the SOFTWARE MANUAL GENERAL DESCRIPTION & MACROS COMPACT PMS V5 (manual number SAE-3-201).

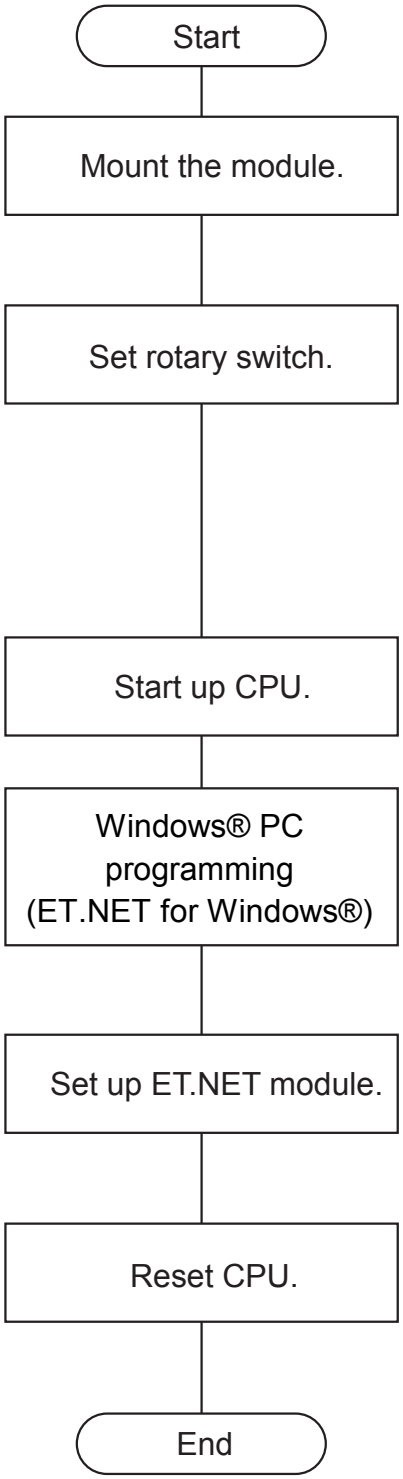
5.2.6 Example of C language program at CPU02

```
#define TCP_OPEN    0x874100L   /* tcp_open( )    starting address(main)      */
#define TCP_CLOSE   0x874112L   /* tcp_close( )   starting address(main)      */
#define TCP_SEND    0x874130L   /* tcp_send( )    starting address(main)      */
#define TCP_RECEIVE 0x874136L   /* tcp_receive( ) starting address(main)      */
#define TCP_ABORT   0x87411EL   /* tcp_abort( )   starting address           */
#define IPADDR      0xC0010001L /* IP address of remote station              */
#define SBUFADDR    0x1E1000L   /* Starting address of send buffer           */
#define RBUFADDR    0x1E2000L   /* Starting address of receive buffer        */
#define PARADDR     0x1E5000L   /* Starting address of parameter storage area */

struct  open_p{
    long    dst_ip;         /* IP address of remote station             */
    short   dst_port;       /* Port number of remote station            */
    short   src_port;       /* Port number of local station             */
    char    notuse;         /* Unused(0)                                */
    char    ttl;            /* Time to live                             */
};

struct  send_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Send data length(bytes)                  */
    char    *buf;           /* Starting address of send data            */
};

struct  receive_p{
    short   s_id;           /* Socket ID                                */
    short   len;            /* Buffer length                            */
    char    *buf;           /* Starting address of buffer               */
    long    tim;            /* Receive wait time(ms)                    */
};

struct  close_p{
    short   s_id;           /* Socket ID                                */
};
struct  abort_p{
    short   s_id;           /* Socket ID                                */
};
/************************/
/*  task3: Client(CPU02) */
/************************/
main()
{
    register    short ( *tcp_open )( );
    register    short ( *tcp_send )( );
    register    short ( *tcp_receive )( );
    register    short ( *tcp_close )( );
    register    short ( *tcp_abort )( );
    long    time;
    short   rtn, i, cerr_flg;
    char    *sbuf, *rbuf;
    struct  open_p      *open;
    struct  send_p      *send;
    struct  receive_p   *recv;
    struct  close_p     *close;
    struct  abort_p     *abort;

    open  = (struct open_p    *)PARADDR;    /* Starting address of input parameter storage area*/
    send  = (struct send_p    *)(open  + 1);
    recv  = (struct receive_p *)(send  + 1);
    close = (struct close_p   *)(recv  + 1);
    abort = (struct abort_p   *)(close + 1);

    sbuf = ( char *)SBUFADDR;               /* Starting address of send buffer */
    for( i = 0 ; i < 1024 ; i++ ){
        sbuf[i] = 0x55;
    }
    while( 1 ){
        open->dst_ip    = IPADDR;           /* IP address of remote station    */
        open->dst_port  = 10001;            /* Port number of remote station   */
```

```
        open->src_port  = 10001;            /* Port number of local station    */
        open->notuse    = 0;                /* Unused                          */
        open->ttl       = 0;                /* Time to live                    */
        tcp_open = (short (*)())TCP_OPEN;
        rtn      = (tcp_open)(open);        /* Opens TCP actively.             */
        if( rtn > 0 ){                      /* Return code normal?             */
            break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro        */
        delay( &time);
    }
    send->s_id = rtn;                       /* Socket ID                       */
    recv->s_id = rtn;                       /* Socket ID                       */
    while( 1 ){
        send->len   = 1024;                 /* Send data length(bytes)         */
        send->buf   = ( char *)SBUFADDR;    /* Starting address of send data   */
        tcp_send    = ( short (*)())TCP_SEND;
        rtn         = (tcp_send)(send);     /* Sends TCP data.                 */
        if( rtn < 0 ){                      /* Return cond abnormal?           */
            break;
        }
        recv->len = 1024;                   /* Receive buffer length(bytes)       */
        recv->buf = ( char*)RBUFADDR;       /* Starting address of receive buffer */
        recv->tim = 60000;                  /* Receive wait time(ms)           */
        tcp_receive = ( short (*)())TCP_RECEIVE;
        rtn         = (tcp_receive)(recv);  /* Receive TCP data.               */
        if( rtn < 0){                       /* Return cond abnormal?           */
            break;
        }
        cerr_flg = 0;                       /* Clears compare error flag.         */
        sbuf     = ( char *)SBUFADDR;       /* Starting address of srnd buffer    */
        rbuf     = ( char *)RBUFADDR;       /* Starting address of receive buffer */
        for( i = 0 ; i < 1024 ; i++ ){
            if( sbuf[i] != rbuf[i]){
                cerr_flg = 1;               /* Sets compare error flag.        */
                break;
            }
            sbuf[i] = ~rbuf[i];             /* Sets complement                    */
        }
        if( cerr_flg == 1 ){                /* Compare error?                  */
            break;
        }
    }
    close->s_id = send->s_id;               /* Socket ID                       */
    while( 1 ){
        tcp_close   = ( short (*)())TCP_CLOSE;
        rtn         = (tcp_close)(close);   /* Terminates TCP connection.      */
        if( rtn == 0 || rtn == ( short )0xFFF6 ){
            break;
    } else if ( rtn == (short )0xF012 ) {
         tcp_abort = ( short (*)())TCP_ABORT;
         rtn = (tcp_abort)(abort);          /* Terminates TCP connection forcibly */
         break;
        }
        time = 100;                         /* Issue of 100-ms Delay macro        */
        delay( &time);
    }
    return;
}
```

6 OPERATION

6.1 Start-up Procedure



[1] Turn off the power to the CPU, and mount the ET.NET module.

[2] Set the MODU. NO. switch of the ET.NET module as shown below.

Module No.		Description
Main	Sub	
0	1	Communication using 10BASE-5 connections
2	3	Communication using 10BASE-T connections
4	5	Communication with a tool

[3] Turn on the power to the CPU.

[4] Connect the CPU to the Windowsfi PC via an RS-232C interface cable. Then, set up ET.NET for Windows. (Refer to the SOFTWARE MANUAL, OPTION ET.NET For Windows (manual number SAE-3-148)).

[5] Set up the ET.NET module. (Set IP address and subnetwork mask.)

[6] Press the CPU reset switch for one second or more to reset the CPU.

NOTE

- If the IP address of the host is set to all /0s or all /Fs, then an input error will occur.
  If the setup menu for the Windows® Tool is called up with no ET.NET module installed, the physical address /FFFFFFFFFFFF is displayed. When you want to reference physical addresses, install an ET.NET module beforehand. However, IP addresses and subnet masks can be set and referenced even when no ET.NET module is installed.
- When it is found that the IP address of an ET.NET module is not set, or when it is lost due to memory initialization during loading of the operating system, the ERR LED for the ET.NET module lights and one of the following messages appears in the CPU indicator. At the same time, the communication stops.

  If the IP address of the main module is not set: “ETM IPNG”
  If the IP address of the submodule is not set: “ETS IPNG”

7 MAINTENANCE

7.1 Maintenance Inspection

To use the S10mini in an optimum condition, check the items listed below. Make this check at routine inspection or periodic inspection (twice or more per year).

(1) Module appearance
Check that no fissure or crack exists in the module case. If the case has such a damage, there is a possibility that the internal circuit may also be damaged, resulting in a system malfunction.

(2) Indicator's ON status and indication
From the indicator status, check that no special fault exists.

(3) Looseness of mounting screws and terminal base screws
Check that the mounting screws and terminal base screws of the module are not loose. If any of these screws is found to be loose, tighten it. Such a loose screw may result in a system malfunction or a burn-out due to overheating.

(4) Module replacement
Hot swapping of modules will lead to hardware or software damage. Be sure to replace a module in a power OFF state.

(5) Cable sheath condition
Check that the cable sheath is not abnormal. A peeled sheath may cause a system malfunction or electric shock, or may result in a burn-out due to short circuit.

(6) Dust sticking condition
Check if dust and dirt collects on the module. If dust collects on the module, remove it with a vacuum cleaner. Dust on the module may short the internal circuit, resulting in a burn-out.

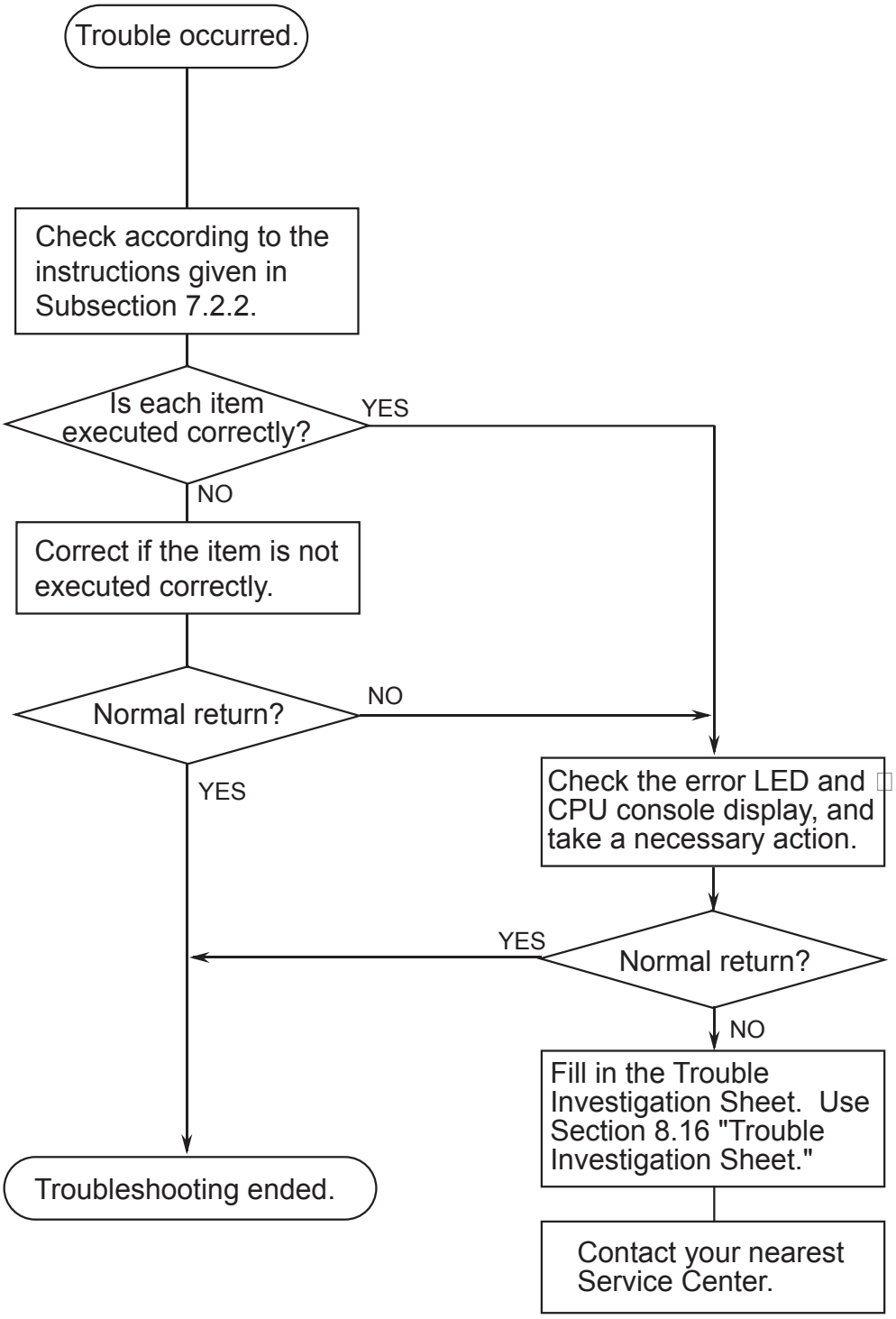
(7) Power supply voltage
Check that neither the internal power supply of the module nor the external power supply to it is out of the specified range. If the power supply voltage deviates from the rating, a system malfunction may result.

CAUTION

Static electricity may damage the module. Before starting the work, discharge all electrostatic charge from your body.

7.2 Troubleshooting

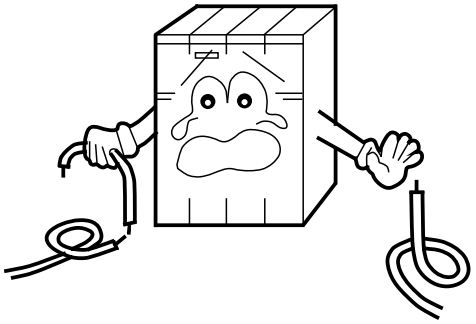
7.2.1 Procedure



7.2.2 Before suspecting a failure

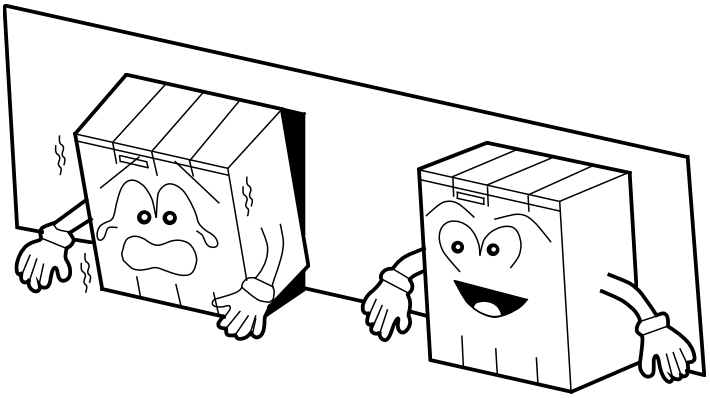
Is cabling normal?

- Check that there is disconnection or erroneous connection of cables.
- Check that a cable with shielded ground wire is used as the transceiver cable.



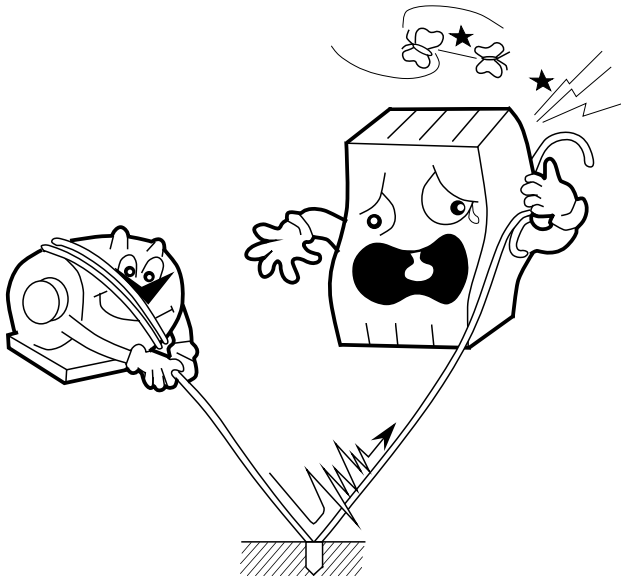
Are the modules mounted correctly?

- Check that the ET.NET modules are inserted from the left.
- Check that no set screws loosen.



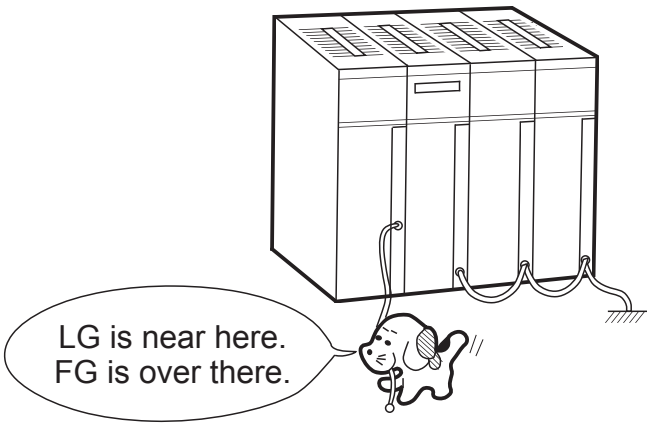
Is grounding made correctly?

- Separate the grounding from that of high-voltage equipment.
- Perform grounding work conforming to Class D grounding or higher.



Are LG and FG separated?

- If electrical noise from the power supply enters the FG (frame ground) via the LG (line ground), a malfunction may result. To prevent this, LG and FG must be separated.
- Ground LG at the power supply side.



7.3 Errors and Actions To Be Taken

7.3.1 CPU LED display messages

In the CPU LED display, a distinction is made between the main module and submodule, as shown in the table below.

Module	Display message	Explanation	User action
Main	ETM @. @	The ET.NET module (main) was started up normally.	This is not an error.
	ETM □□□□	A hardware error was detected on the board of the ET.NET module (main).	See Subsection 7.3.2, "Hardware errors."
	EXD2 PTY	A parity error occurred when the CPU read memory of the ET.NET module (main).	Set the CPU switch OFF, then set it ON again. If this message still appears, replace the ET.NET module.
Sub	ETS @. @	The ET.NET module (sub) was started up normally.	This is not an error.
	ETS □□□□	A hardware error was detected on the board of the ET.NET module (sub).	See Subsection 7.3.2, "Hardware errors."
	EXD3 PTY	A parity error occurred when the CPU read memory of the ET.NET module (sub).	Set the CPU switch OFF, then set it ON again. If this message still appears, replace the ET.NET module.

- The "@. @" above indicates the version and revision of the ET.NET module.
- The "□□□□" indicates the error display data in Subsection 7.3.2, "Hardware errors."

7.3.2 Hardware errors

If the ET.NET module detects a hardware error, an error message as in the table below is displayed with the CPU LED; The error LED lights and error freeze information is collected; and the operation of the ET.NET module stops.

Display message	Error	User action
BUS	Bus error	The ET.NET module may have failed. Replace the module.
ADDR	Address error	
ILLG	Invalid instruction	
ZERO	Division by zero	
PRIV	Privilege violation	
FMAT	Format error	
SINT	Spurious interrupt	
EXCP	Unused exception	
PTY	Parity error	
MDSW	Module switch setting error	Check the module switch setting.
ROM1	ROM1 sum error	The ET.NET module may have failed. Replace the module.
RAM1	RAM1 compare error	
RAM2	RAM2 compare error	
ROM3	ROM3 sum error	
IPNG	IP address not registered	Register an IP address.
MAC	MAC address not registered	The ET.NET module may have failed. Replace the module.
PRG	Microprogram error	
R_NG	Route information setting error	The set route information is erroneous. See Subsection 7.3.4, “Route information setting error table,” and correct the route information.

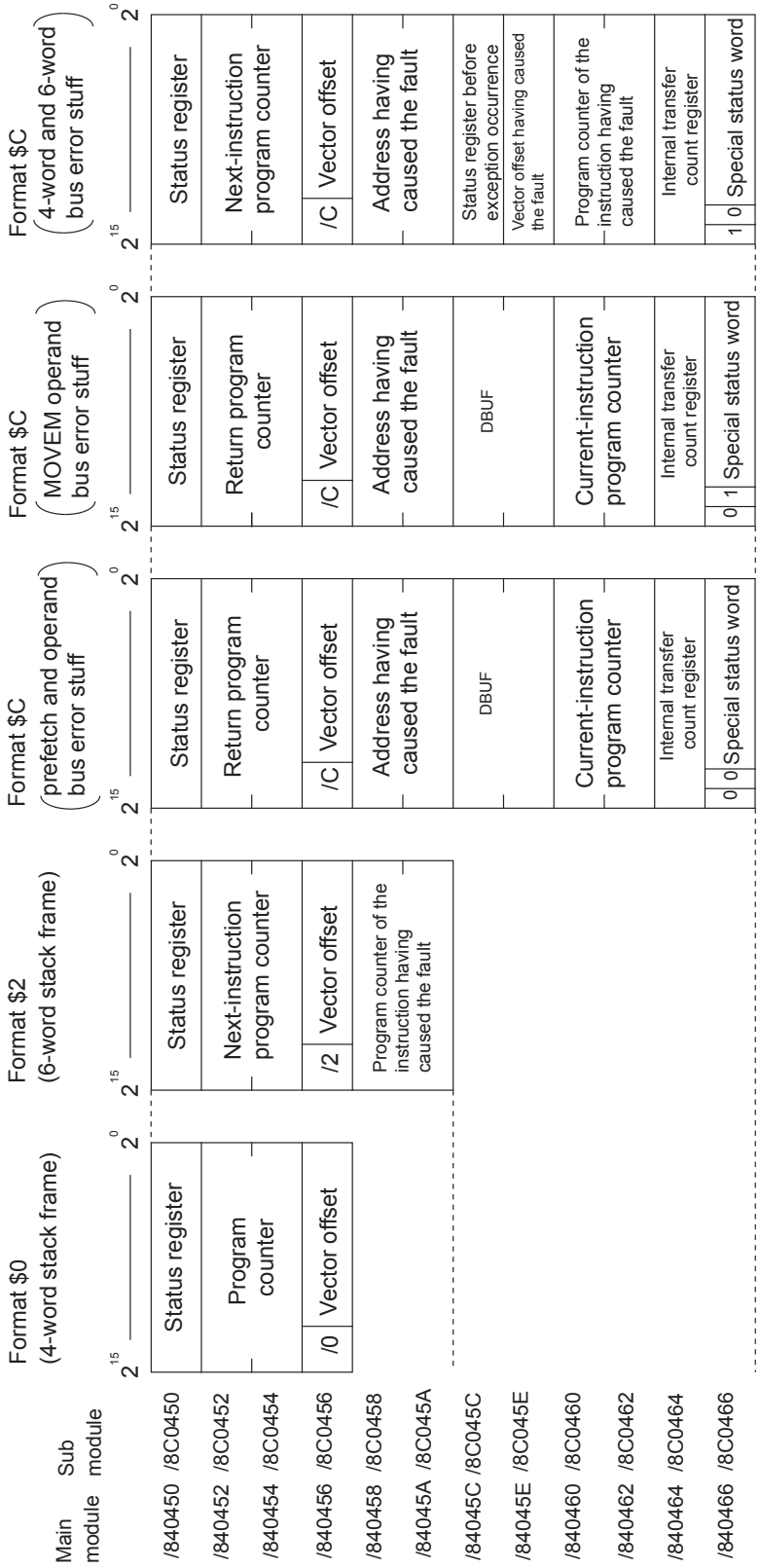
If the ET.NET module detects a hardware error, the error LED lights and error freeze information is registered, and the operation of the ET.NET module stops.

Main module	Submodule	2^{31} —— 2^{16} 2^{15} —— 2^{0}
/840400	/8C0400	Error code | ——
/840404	/8C0404	——
/840410	/8C0410	D0 register
/840414	/8C0414	D1 register
/840418	/8C0418	D2 register
/84041C	/8C041C	D3 register
/840420	/8C0420	D4 register
/840424	/8C0424	D5 register
/840428	/8C0428	D6 register
/84042C	/8C042C	D7 register
/840430	/8C0430	A0 register
/840434	/8C0434	A1 register
/840438	/8C0438	A2 register
/84043C	/8C043C	A3 register
/840440	/8C0440	A4 register
/840444	/8C0444	A5 register
/840448	/8C0448	A6 register
/84044C	/8C044C	A7 register
/840450	/8C0450	Stack frames (4 words, 6 words, bus error)
/8404FC	/8C04FC	

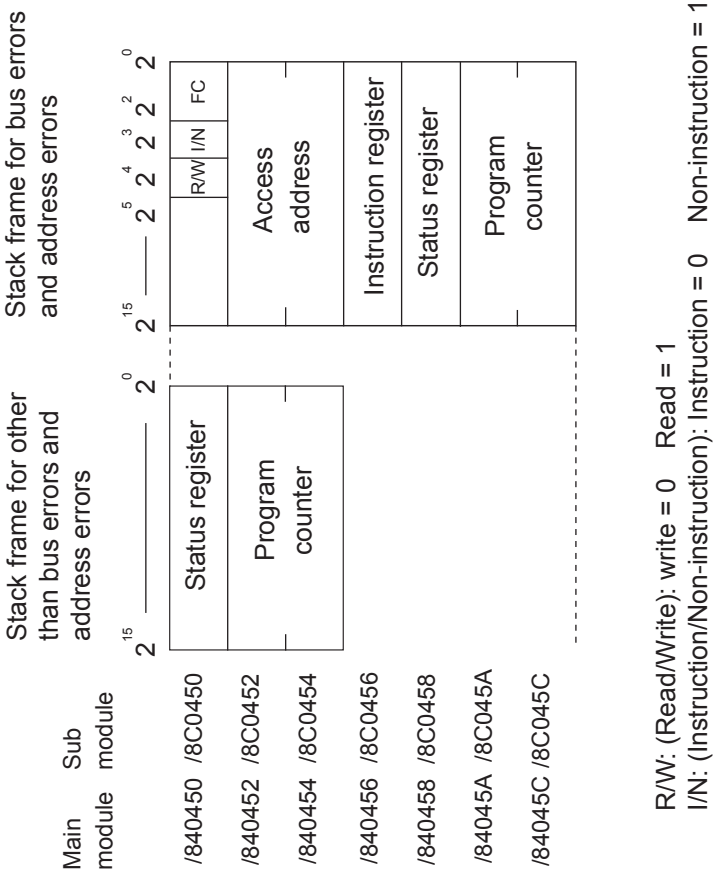
No.	Code	Error
1	0010H	Bus error
2	0011H	Address error
3	0012H	Invalid instruction
4	0013H	Division by zero
5	0014H	Privilege violation
6	0016H	Format error
7	0017H	Spurious interrupt
8	0018H	Unsupported exception (CHK, TRAPV, L1010, etc.)
9	0019H	Parity error
10	001AH	Power failure notice
11	0100H	Module switch setting error
12	0102H	ROM1 sum error
13	0103H	RAM1 compare error
14	0105H	RAM2 compare error
15	010BH	ROM3 sum error
16	0112H	Microprogram error
17	0113H	IP address not registered
18	0114H	MAC address error
19	0200H	Route information setting error

Note: The details of the stack frames are shown on the next page.

The following shows the details of the stack frames in the error freeze information table.
Below are shown the details of the stack frames in the error freeze information table of the ET.NET module (LQE020) of revision “L” or later (having the label “L” or later on the top of the casing or whose CPU indicator indicates “ETM 4.0” or “ETS 4.0” or later)



Below are shown the details of the stack frames in the error freeze information table of the ET.NET module (LQE020) of revision “L” or later.



7.3.3 Error codes from the socket handler

The table below lists the error codes returned from the socket handler and the user actions to be taken against those errors.

Error code	Error	Cause	User action
0xF000	Connection not established	When the handler was started, a connection was not yet established or a port was already released.	Issue tcp_open or tcp_popen to establish a connection. Then re-call the handler function.
0xF002	FIN received	An FIN was received when the handler was started.	Issue tcp_close to terminate the connection. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xF010	Invalid socket ID	• The socket ID was out of range. (For TCP, 1 ≤ ID ≤ 15; for UDP, 0 × 20 ≤ ID ≤ 0 × 27) • The ID of an unused socket or already-released socket was specified. • A connection was not yet established.(Applicable to tcp_accept only)	Check the user program, for example, to see whether a value returned by tcp_open or tcp_popen is specified as the socket ID.
0xF011	Too many sockets	An attempt was made to register more sockets than the limit. (For TCP, 12; for UDP, 8)	Close unused sockets using tcp_close or udp_close. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xF012	Socket driver time-out	The socket driver did not respond within the specified time.	Issue tcp_close to terminate the connection. Then, issue tcp_open or tcp_popen to re-establish a connection. If communication still is not resumed, check the connectors, cables, and the remote station for any abnormality. When this error occurs due to tcp_close, issue tcp_abort, disconnect the line, and issue tcp_open or tcp_popen to re-establish a connection.

Error code	Error	Cause	User action
0xF013	Module stopped	When the handler was started, initialization of the socket driver was not terminated within 100 seconds.	Issue tcp_close within the range allowed for the application. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xF020	Invalid send data length	The send data length was out of range. (For TCP, 1 ≤ data length ≤ 4096; for UDP, 1 ≤ data length ≤ 1472)	Check the user program to see whether the send data length is specified correctly.
0xF021	Invalid receive data length	The receive data length was out of range. (1 ≤ data length ≤ 4096)	Check the user program to see whether the receive data length is specified correctly.
0xF0FF	Port released	• After the handler was started, a port was released (RST was received). (tcp_open) • When an attempt was made to start the handler, the port was already released. (tcp_send or tcp_receive)	• Issue tcp_open or tcp_popen to re-establish a connection. • Issue tcp_close to terminate the connection. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xFFF0	Invalid address	• Both udp_open and udp_send set the IP address and port number of the remote station to 0. • udp_send caused an Ethernet-level error such as a collision.	• Check the user program. • Retry udp_send when the current amount of traffic is reduced.
0xFFF3	Invalid argument	An invalid argument was specified.	Check the user program.
0xFFF5	Connection time-out	The remote station did not respond.	Issue tcp_close to terminate the connection. Then, issue tcp_open or tcp_popen to re-establish a connection. If communication still is not resumed, check the connectors, cables, and the remote station for any abnormality.
0xFFF6	Already closed	A command was issued for a socket ID for which a connection had been terminated (closed or aborted).	Issue tcp_open or tcp_popen to re-establish a connection.

Error code	Error	Cause	User action
0xFFF8	FIN received	An FIN was received from the remote station.	Issue tcp_close to close the socket.
0xFFFA	Forcibly terminated connection	A connection was forcibly terminated by the remote station (RST was received). (tcp_receive was issued after RST was received.)	Issue tcp_close to terminate the connection. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xFFFC	Invalid network handle	An attempt was made to perform transmission or reception using the number of a handle not yet opened by TCP or UDP. This error is likely to occur when an RST is received. (An RST was received during waiting for reception by tcp_receive.)	Issue tcp_close to close the socket. Then, issue tcp_open or tcp_popen to re-establish a connection.
0xFFFD	Duplicate socket number	The same socket already existed. (The IP address of the remote station, the port number of the remote station, and the port number of the local station were duplicated.)	Check the user program.
0xFFFE	Invalid control block	An attempt was made to use more sockets than the limit.	Close unused sockets using tcp_close or udp_close. Then, issue tcp_open or tcp_popen to re-establish a connection.

7.3.4 Route information setting error table

When a route information setting error is detected, its error code is set in the following table:

Main module	Submodule	2^{31} ——— 2^{0}
/873880	/8F3880	Default
/873884	/8F3884	User (1)
/873888	/8F3888	User (2)
/87388C	/8F388C	User (3)
/873890	/8F3890	User (4)
/873894	/8F3894	User (5)
/873898	/8F3898	User (6)
/87389C	/8F389C	User (7)
/8738A0	/8F38A0	User (8)
/8738A4	/8F38A4	User (9)
/8738A8	/8F38A8	User (10)
/8738AC	/8F38AC	User (11)
/8738B0	/8F38B0	User (12)
/8738B4	/8F38B4	User (13)
/8738B8	/8F38B8	User (14)

+0	Error code
+2	Duplicate user No.

Error code : See the table below.

Duplicate user No.: A stored user number is duplicated.
(Default = 0,
Other users = 1 to 14)

No.	Code	Contents	Duplicate user No. stored or not
1	0010H	The remote station IP address is duplicated with the local station IP address.	Not stored
2	0011H	The remote station IP address is duplicated with another gateway IP address.	Stored
3	0012H	The remote station IP address is duplicated with another remote station IP address.	Stored
4	0013H	The same network address as the local station's is set as the network address of a remote station IP address.	Not stored
5	0014H	The network address of a remote IP address is duplicated with the network address of a remote station IP address.	Stored
6	0016H	The remote station IP address is 255.255.255.255.	Not stored
7	0020H	The gateway IP address is duplicated with the local station IP address.	Not stored
8	0022H	The gateway IP address is duplicated with another local station IP address.	Stored
9	0023H	The same network address as the local station's is set as the network address of a gateway IP address.	Not stored
10	0024H	The network address of a gateway IP address is duplicated with the network address of another local station IP address.	Stored
11	0026H	The gateway IP address is 255.255.255.255.	Not stored
12	0030H	The subnetwork identified by a gateway IP address does not match the subnetwork of the local station. (*)	Not stored

(*) The user should be careful when, after setting necessary route information, he/she is connecting the ET.NET module with the tool by cable. Setting the ET.NET module's MODU No. switch in 4- or 5-position at that time may result in the display of the "ET*R_NG" error message on the CPU's LED indicator, except when the local station's IP address uses the network address "192.192.192.0". Recovery from this type of error can be made by simply setting the MODU No. switch back in its previous position. Even if this type of error is reported by error message, normal data communication is possible with the tool (personal computer) as long as the ET.NET module is connected directly with that tool by cable.

8 APPENDIX

8.1 Network Components

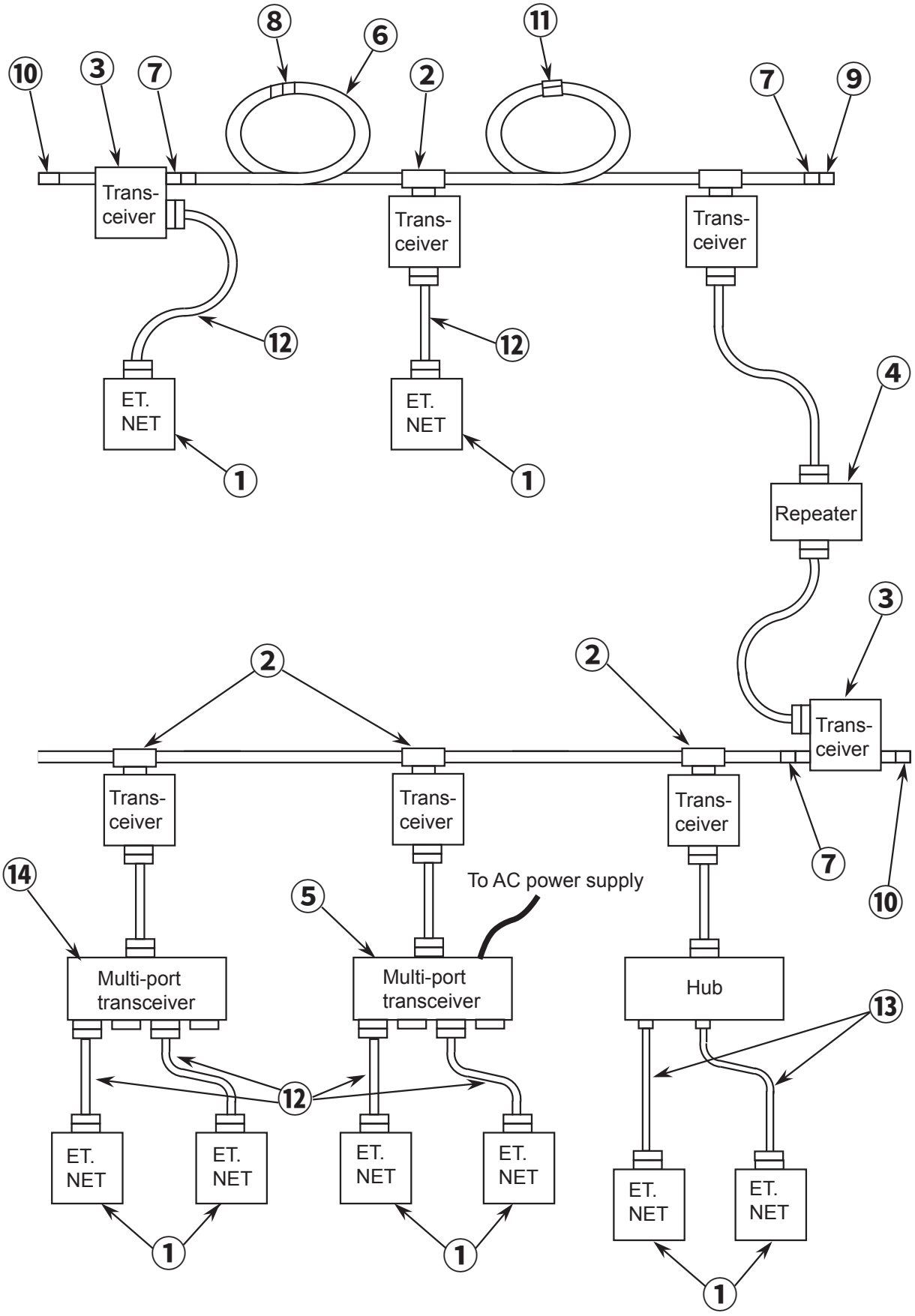
8.1.1 Problem of connection between LQE020 and Ethernet*

LQE020 is a standard product conforming to the international standard IEEE802.3.
However, when combining different manufacturers' transceivers, repeaters, and so forth conforming to the same standard, the system may not operate normally due to incompatibility.
For LQE020, therefore, use the transceivers, repeaters, coaxial cables, connectors, and terminators that are recommended by Hitachi.
In Ethernet, there are two types of specifications: IEEE802.3-conforming specifications and original Ethernet specifications. Note that devices of the original Ethernet specifications cannot be connected to LQE020.
* Ethernet is a registered trademark of Xerox Corp.

8.1.2 Component list

No.	Product name	Manufacturer	Model	Remarks
①	ET.NET	Hitachi, Ltd.	LQE020	IEEE802.3-conforming LAN controller mounted in S10mini
②	Transceiver	Hitachi Cable, Ltd.	HLT-200TB HBN200TZ HBN200TD	Tap-type transceiver
③	Transceiver	Hitachi Cable, Ltd.	HLT-200	Connector-type transceiver
④	Repeater	Hitachi Cable, Ltd.	HLR-200H	Repeater for extending transmission distance of coaxial cable
⑤	Multi-port transceiver	Hitachi, Ltd.	H-7612-64 H-7612-68	4-port/8-port transceiver AC power supply built in
⑥	Coaxial cable	Hitachi Cable, Ltd.	HBN-CX-100	Indoor coaxial cable Cable length specified (up to 500 m)
⑦	Coaxial connector	Hitachi Cable, Ltd.	HBN-N-PC	Connector for coaxial cable
⑧	Relay connector	Hitachi Cable, Ltd.	HBN-N-AJJ	Relay connector for coaxial cable
⑨	Terminator	Hitachi Cable, Ltd.	HBN-T-NJ	J type
⑩	Terminator	Hitachi Cable, Ltd.	HBN-T-NP	P type
⑪	Ground terminal	Hitachi Cable, Ltd.	HBN-G-TM	Ground terminal for coaxial cable

No.	Product name	Manufacturer	Model	Remarks
⑫	Transceiver cable	Hitachi Cable, Ltd.	HBN-TC-100	With male and female D-sub 15 pin connectors Up to 50 m
⑬	Twisted pair cable	Hitachi Cable, Ltd.	HUTP-CAT5 4P	Twisted pair cable
⑭	Multi-port transceiver	Hitachi Cable, Ltd.	HBM-400TZ	4-port transceiver


Trans-
ceiver
Trans-
ceiver
Trans-
ceiver
ET.
NET
ET.
NET
Repeater
Trans-
ceiver
Trans-
ceiver
Trans-
ceiver
Trans-
ceiver
To AC power supply
Multi-port
transceiver
Multi-port
transceiver
Hub
ET.
NET
ET.
NET
ET.
NET
ET.
NET
ET.
NET
ET.
NET

8.2 Cabling of Coaxial Cable

The coaxial cable shall be laid in an indoor cabling duct and must be separated from 100 V or higher wiring.
Before laying the cable, never fail to check that there is no short circuit nor break.

8.2.1 Laying cable segment

(1) The methods of laying the cable depending on the cabling location. The major methods are listed below.
- Rolling cabling in ceiling
- Cabling in cable rack
- Open cabling on wall surface
- Free-access cabling in floor pit
- Cabling in conduit

(2) The notes on cabling work are described below.
- In principle, lay this cable indoors.
- The weight of the cable is about 1.9 kg per 10 m.
- Do not add the tension of 245N or more to the cable body during cable laying.
- The bend radius of the cable should be 250 mm (150 mm when unavoidable) or more both when the cable is being laid and when it is finally fixed.
- Use a saddle when fixing the cable to a wall surface or ceiling. Except for special cases, the standard fixing interval is 1 m. When fixing the cable, take care not to deform the cable by tightening the saddle.
- When fixing the cable to a cable rack, the standard fixing interval is 2 m.
- For cabling in conduit, use a conduit whose inside diameter is 22 mm or more except for special cases (e.g., when it is used in the penetrated part of a fire wall).
- The bend radius of the conduit used shall be 300 mm or more.
- When the cable is laid on a floor or floor edge, it is apt to be deformed or damaged by walking or heavy objects. Protect the cable by tying or the like.
- For safety, ground the external conductor of the cable. Ground it at one point on a segment. Class D grounding or higher shall be applied. Insulate the connectors and terminators by covering them with the attached boots or by winding insulating tapes onto them, so that the exposed metallic parts of the cable except those at the grounding point do not touch the earth or other metallic parts.

8.3 Installation of Transceiver (Connector Type)

(1) For the transceiver, the installation location and method differ depending on the conditions of the site. The major installation locations may be as follows:
- On a wall
- Beside a station

Figures 8-1 to 8-6 show installation examples.

(2) Notes on transceiver installation are given below.
- Fix the transceiver by wood screws or the like via metal fittings.
- The installation interval of the transceiver shall be 2.5 mm or more.

(3) Installing the transceiver
Use the HBN-N-PC connector as the connector for the coaxial cable. Fix the transceiver at the four tapped holes so that excessive force is not added to the cable. The external conductor of the coaxial cable floats from the ground potential. Therefore, insulate the coaxial connector by rubber boots or vinyl tape so that it does not touch other metal products. (The case of the transceiver body is kept at the ground potential by connection of the transceiver cable. However, insulate the case at installation time to prevent multi-point grounding.)
For the method of attaching the connector to the coaxial cable, see Section 8.6, “Attaching Coaxial Connector.”

(4) When selecting installation location, strictly observe the following rules:
- The looseness of the connectors and terminators can be checked.
- The looseness of the transceiver cable connectors can be checked.
- The attached LED can be checked.

Installation examples of transceivers and transceiver cables

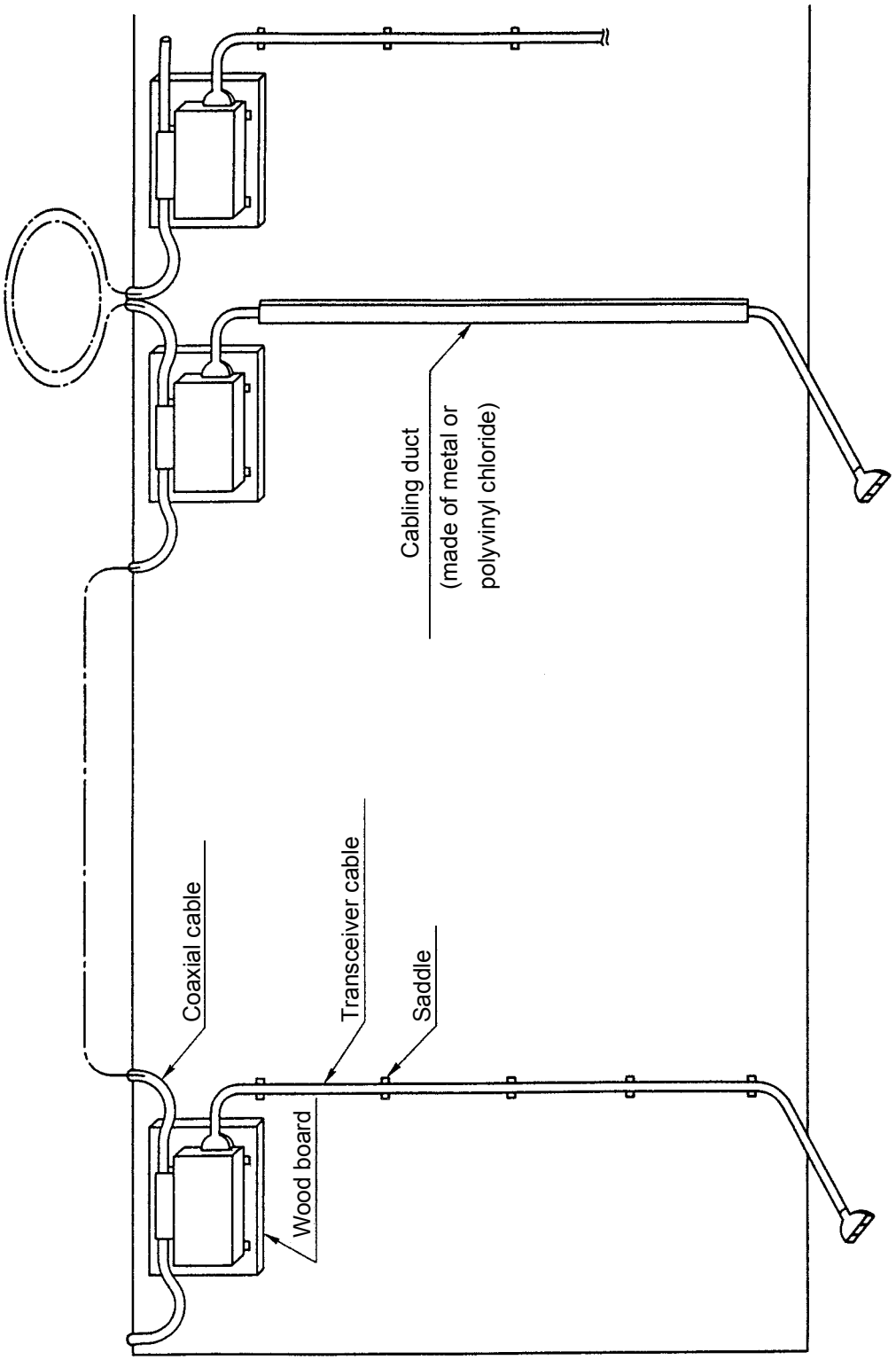


Figure 8-1 Installation on Wall (1)

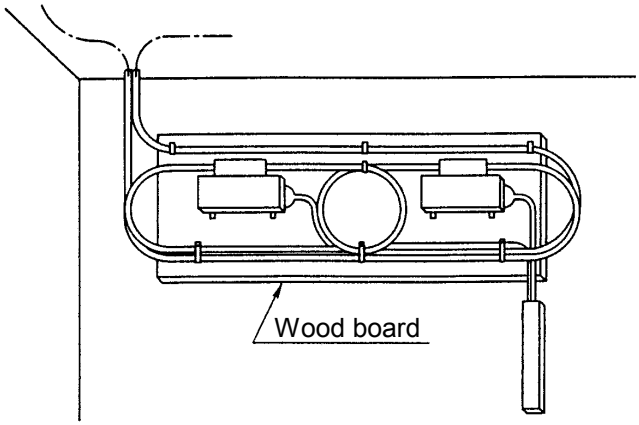


Figure 8-2 Installation on Wall (2)

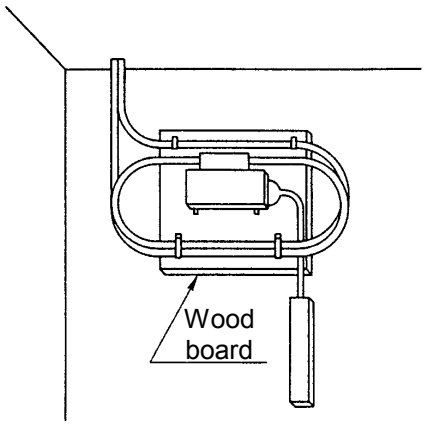


Figure 8-3 Installation on Wall (3)

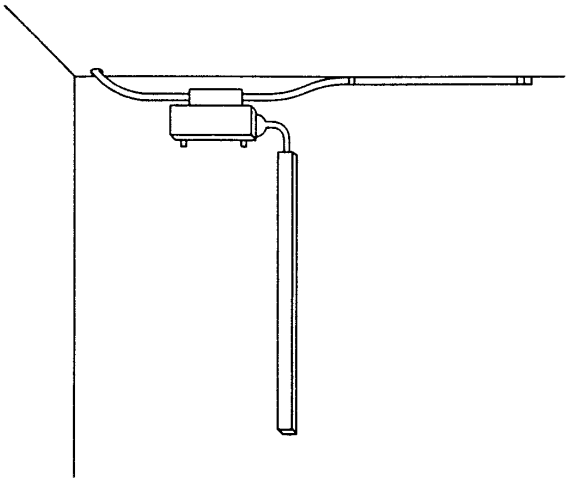


Figure 8-4 Installation on Wall (4)

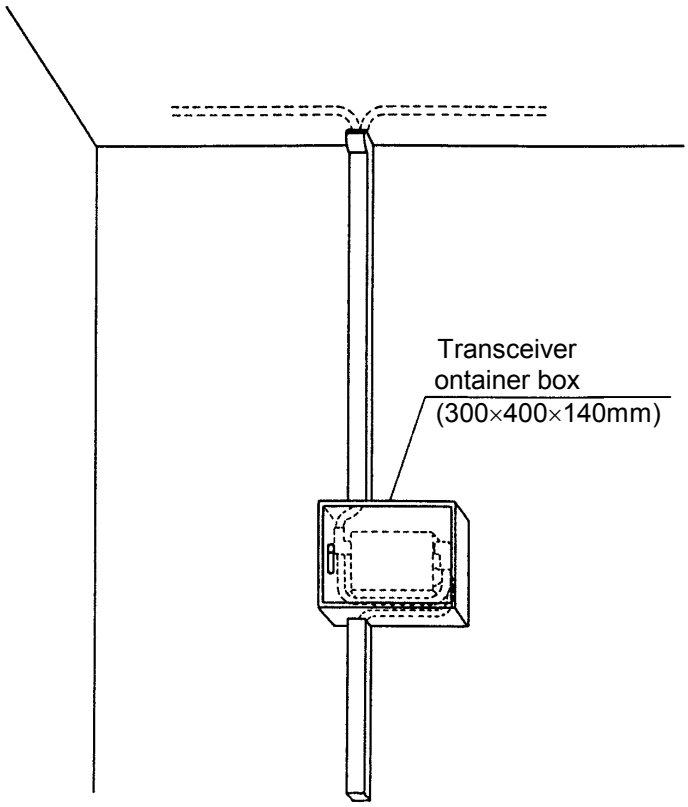


Figure 8-5 Installation in Box (1)

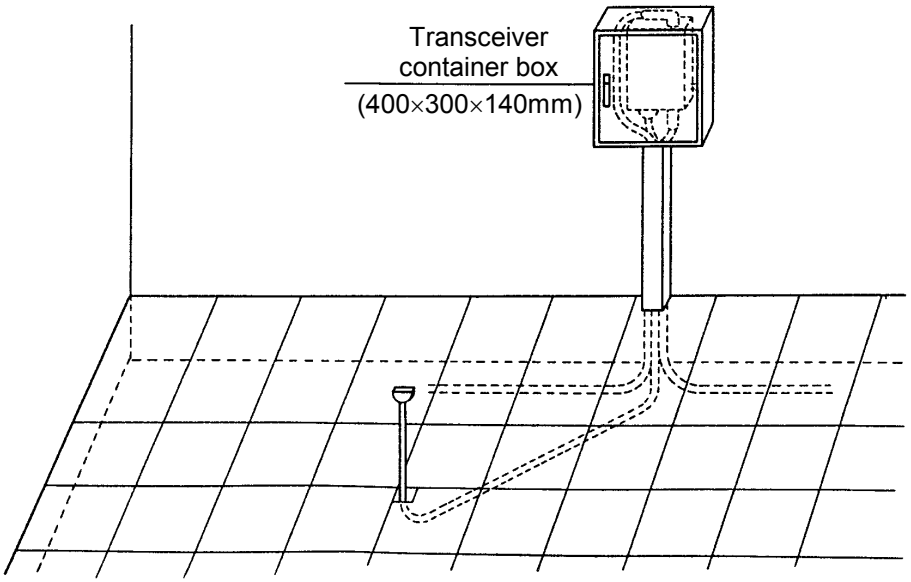


Figure 8-6 Installation in Box (2)

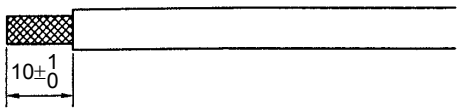
8.4 Installation of Transceiver (TapType)

The installation location and method for the transceiver and the notes on installation are the same as those for the connector-type transceiver described in Section 8.4.
For the method of attaching the tap connector to the coaxial cable, see Section 8.7, "Attaching Tap Connector."

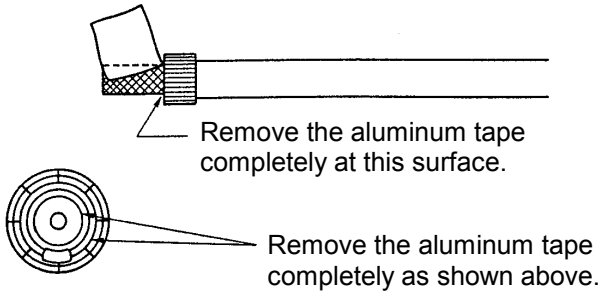
8.5 Attaching Coaxial Connector

(1) Connector attachment procedure
The procedure for attaching the coaxial connector is shown below.

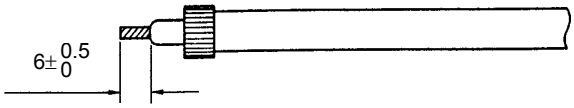
① Peeling off the PVC sheath



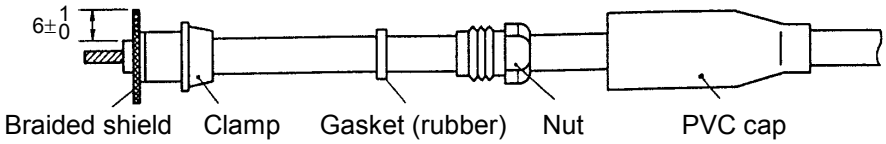
② Removing the aluminum tape



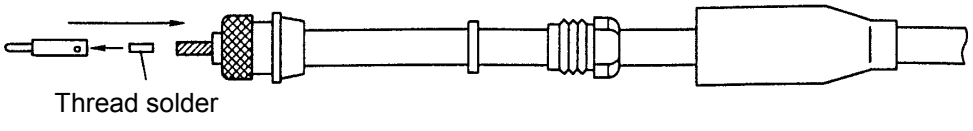
③ Peeling off the insulator



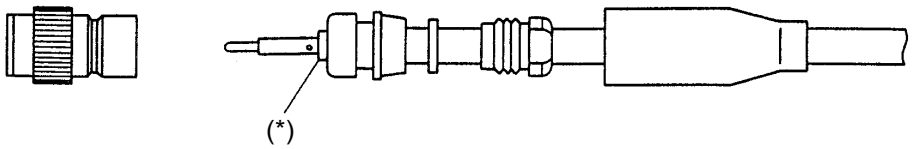
④ Parts setting and shield treatment



⑤ Shield processing and soldering of pin contact



⑥ Assembly

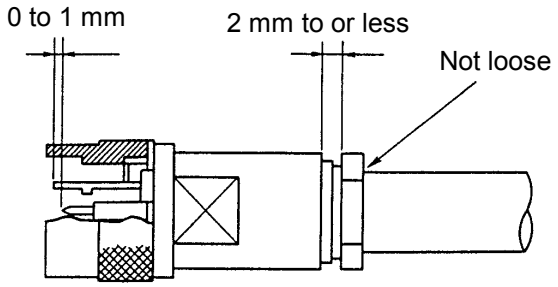


(*) There shall be no gap of 1 mm or more in the pin contact insulator and no bite in the insulator.

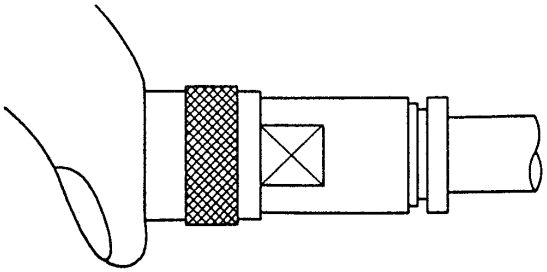
(2) Check after attaching the connector

(a) Dimensions of connector opening

- The difference between the external conductor at the top end of the connector and the inside contact shall be 0 to 1 mm. There shall be no abnormal bump nor dent on the inside contact.



- When putting a thumb to the connector opening, the top end of the inside contact slightly touches the surface of the digital pulp.



- There shall be no abnormal eccentricity of the central conductor found by visual inspection.

(b) Checking looseness

After attaching the connector, grasp and twist the connector body and coaxial cable to confirm that there is no looseness. After tightening, the gap between the tightening nut and the body shall be about 2 mm or less.

(c) Insulation resistance

(Remove the terminator.)

- When no transceiver is set

  Between internal and external conductors: 1000 M Ω/km or more (500 VDC)

- When a transceiver is set

  Measure the external conductor by an ordinary line tester with the internal battery set to the positive pole. At this time, ∞ shall be displayed.

CAUTION

Never fail to discharge electricity after the test, otherwise you will get an electric shock.

8.6 Attaching Tap Connector

Connect the tap connector of the tap-type transceiver and the coaxial cable according to the procedure below.

(1) To fix the coaxial cable ①, insert the cable into the groove of the tap connector body ③, and attach the cover ② from the upper part.
(2) Tighten the hexagon bolt ⑥ by a nutdriver according to the predetermined torque, and connect it to the external conductor of the coaxial cable ①.
Clamping torque for hexagon bolt ⑥: 3 to 4 [N・m]
(3) Tighten the backup probe ⑤ and signal probe ④ slowly, in this order, by using nutdrivers simultaneously from both sides according to the predetermined torque, and connect them to the central conductor of the coaxial cable ①.
Clamping torque for signal probe ④ and backup probe ⑤: 2 to 3 [N・m]
(4) Set the attached cap ⑦ on the backup probe ⑤.
Connection of the tap connector and coaxial cable is completed with the above steps.
As the top ends and thread ridges of the signal probe ④ and backup probe ⑤ are easily deformed, they should be handled carefully.

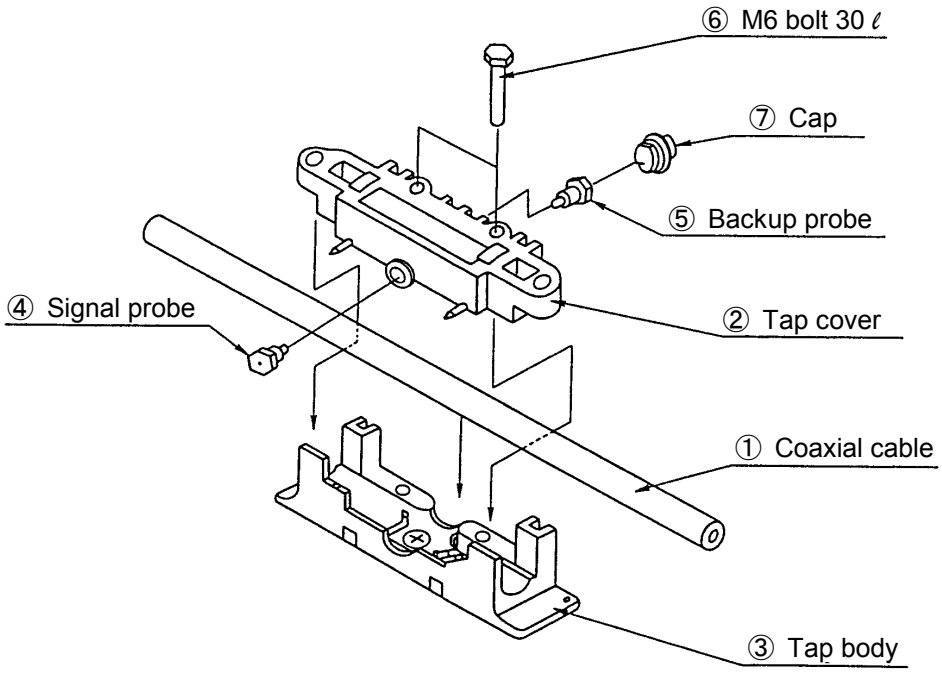


Figure 8-7 Tap Connector Assembly Drawing

CAUTION

- When attaching the tap connector to the coaxial cable, do it according to the order described above.
  If the coaxial cable is attached after the probes ④ and ⑤ are attached, the probes will be destroyed. To prevent this, attach the coaxial cable when probes ④ and ⑤ are completely removed.
- After tightening the probes ④ and ⑤, do not tighten the bolt ⑥ further. Otherwise, the probes may be destroyed because excessive force is added to them.

Connect the tap connector and the transceiver according to the procedure below.

(1) Attach the tap connector ⑧ to the side of the transceiver ⑨, and the probe and ground terminal of the tap connector ⑧ are inserted into the mounting holes of the transceiver ⑨ and are connected.

(2) Tighten the hexagon bolt ⑩ with a nutdriver according to the predetermined torque, and the transceiver ⑨ and the tap connector ⑩ are fixed completely.
Clamping torque for hexagon bolt ⑩: 3 to 4 [N・m]

Connection of the tap connector and transceiver is completed with the above steps.

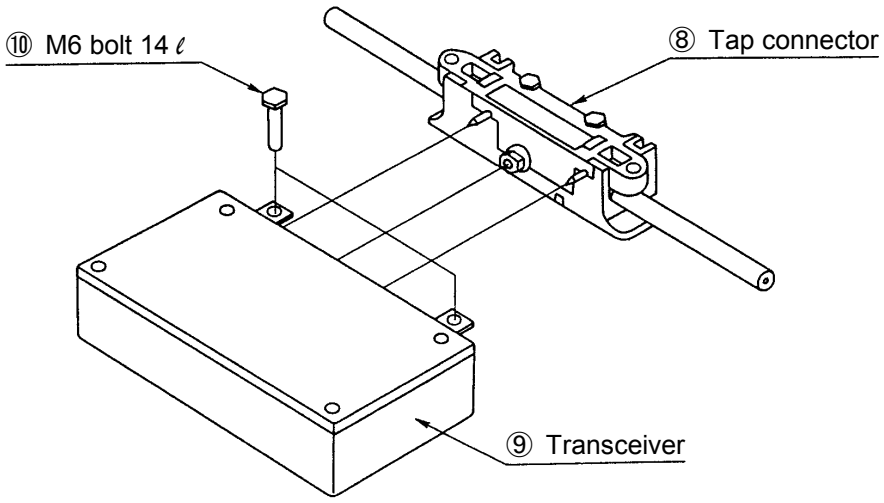


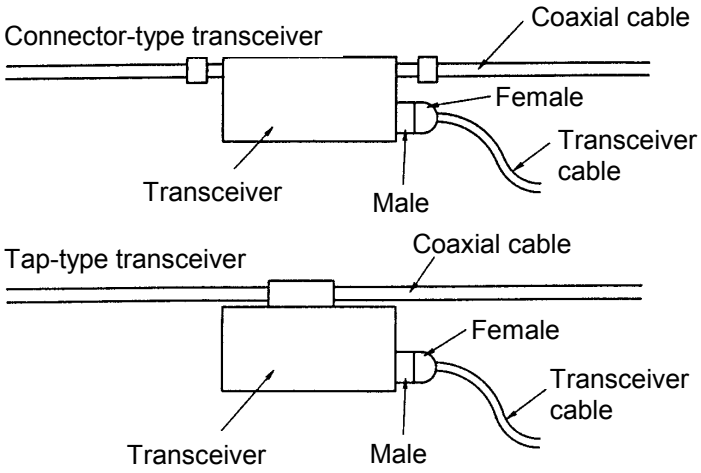
Figure 8-8 Connection of Connector and Transceiver

8.7 Attaching Transceiver Cable

The maximum length of the transceiver cable is 50 m.

Attaching transceiver cable

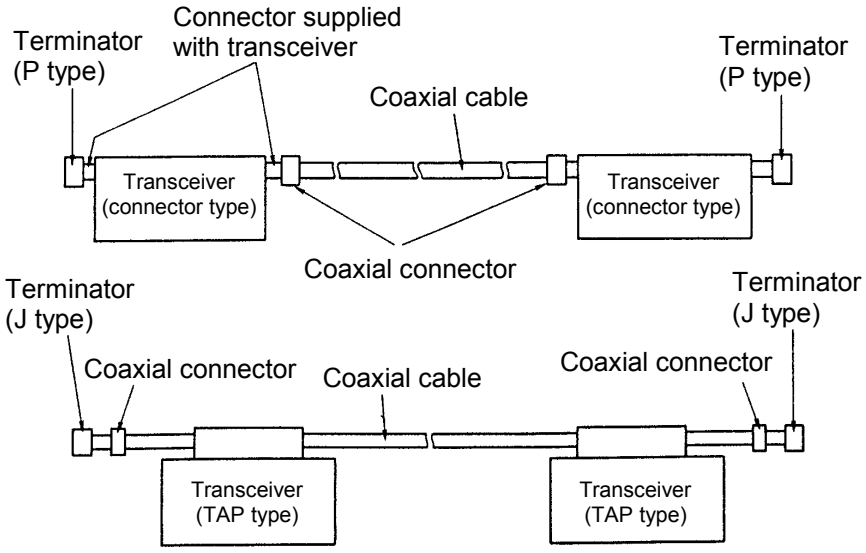
To connect the transceiver cable to the transceiver body, slide the locking retainer of the cable and attach the cable so that is completely locked at the locking post of the transceiver body. The transceiver body is male, and the transceiver cable is female.



8.8 Attaching Terminators

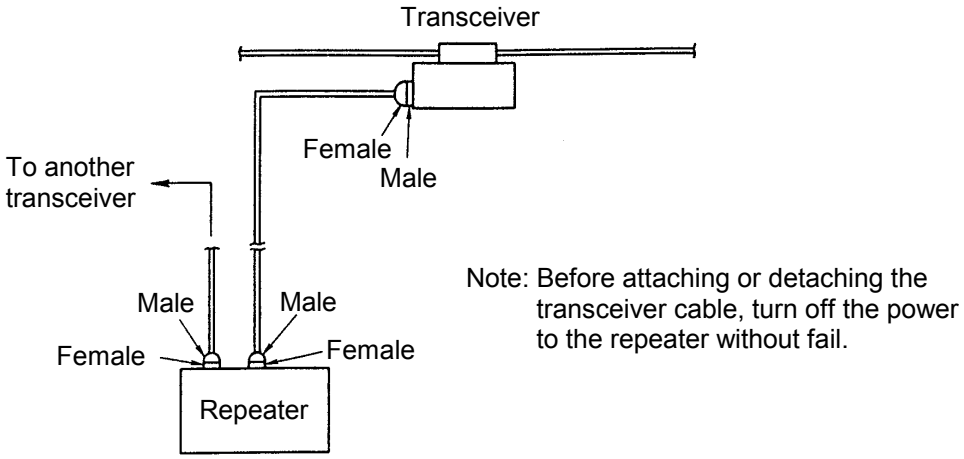
Attaching terminators

Connect the terminators to both ends of the coaxial segment without fail.



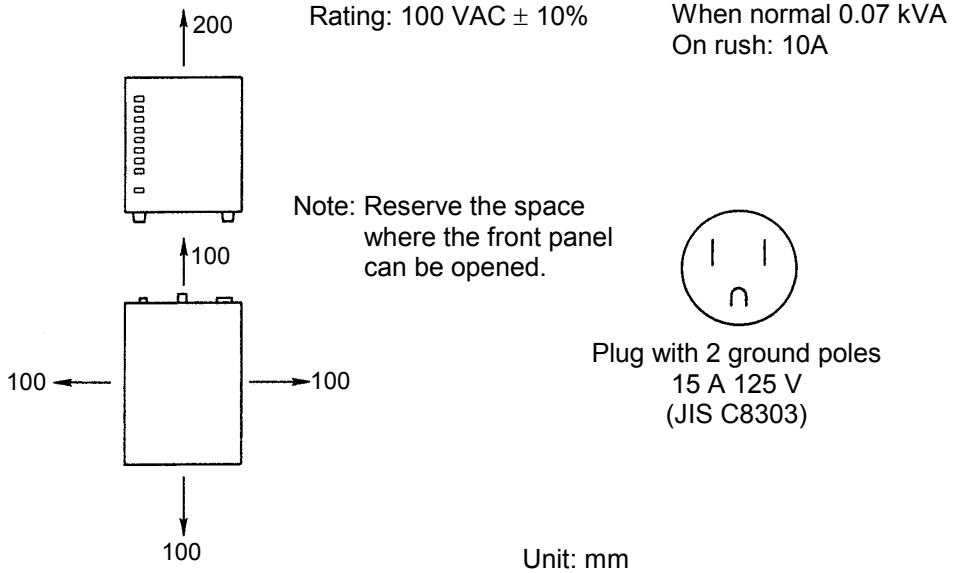
8.9 Attaching Repeater

(1) Connection method



(2) Reserving installation location and space

- The repeater should be installed near a workstation (server). It is necessary that maintenance work can be easily done at this place. (Roof space or underground space of an office is inappropriate.) Reserve sufficient space around and over the repeater. The minimum space to be reserved is shown in the figure below. As AC power supply is required for the repeater, prepare a grounded outlet.



- Do not use the repeater in a place where there is much dust.
- There is an air inlet at the base and an air outlet on the top. Do not cover these parts.
- Considering maintenance, a telephone should be installed near the repeater installation place.

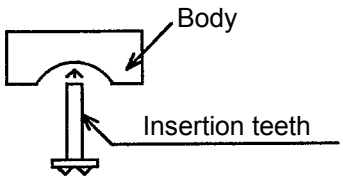
- Use independent power supply to prevent the power from being cut off erroneously. If the power to the repeater is cut off, the transmission function stops.

8.10 Grounding the System

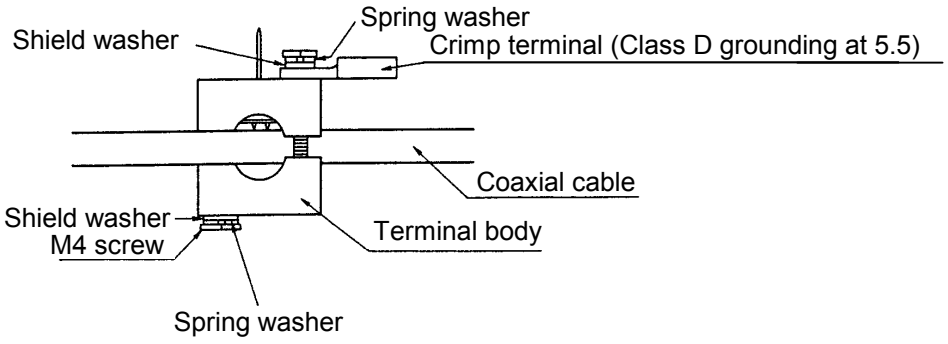
- Grounding the repeater
  Use three-pole power supply for the repeater, or ground the repeater by a ground terminal.
- Grounding each station
  Execute Class D grounding or higher for all devices connected to the LAN control processor. If there is an ungrounded device in the system, an electric shock may occur between this device and a grounded device. A data error (CRC error) may also be caused.
- Grounding the coaxial cable
  For each segment, perform single-point grounding for the coaxial cable.
  The above grounding is for the purpose of safety. Further, it prevents the occurrence of electric noise due to incomplete contact with the earth.
  For grounding, use ground terminals.

8.11 Attaching Ground Terminal

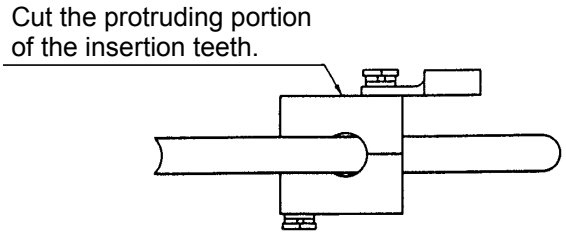
(1) Insert the insertion teeth into the body.



(2) Attach the terminal to the coaxial cable, and tighten the M4 screws alternately. At this time, attach the crimp terminal to one of the screws.
Attach the terminal at any one position on the coaxial segment where it can be attached easily.



(3) After tightening the screws, cut the protruding portions of the insertion teeth.

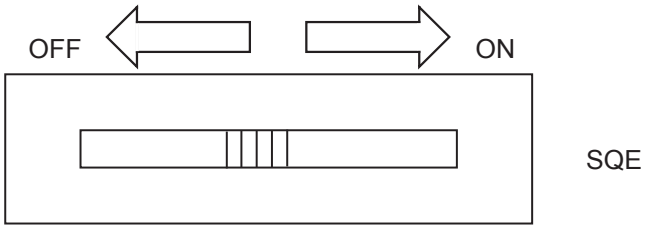


8.12 Setting Single-port Transceiver

(1) Setting the SQE switch of single-port transceiver
For the SQE switch of a single-port transceiver, the setting change shown below is required depending on the connected device.

Connected device / SQE switch	ET.NET controller	Multi-port transceiver	Repeater
Setting	ON	OFF	OFF

For the single transceivers HLT-200 and HLT-200TB, the SQE switch is contained in the case. When changing the setting, open the case to do the work. (The switch is set to ON by turning it to the “SQE” side of silk printing on the board.)



8.13 Setting and Display of Multi-port Transceiver

(1) Setting operation mode

The multi-port transceiver can be used in two operation modes: network and local. Operation mode can be set by operating the switch on the rear panel.

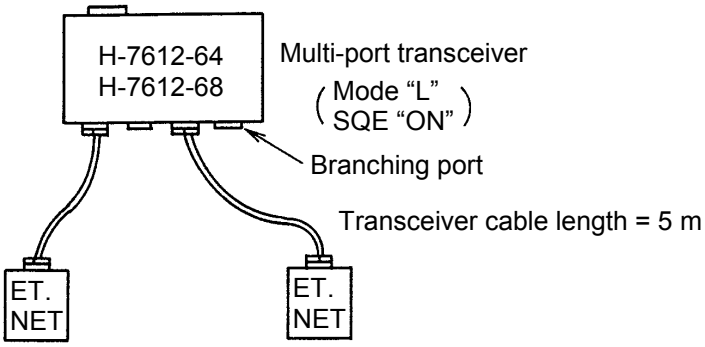
- Local mode

  In local mode, the transceiver is disconnected from the coaxial cable and is used independently.

  Do not connect the transceiver cable to the relay port.

  Set the mode switch to 'L' (local mode).

  At this moment, set the SQE switch of the support port to 'ON'.

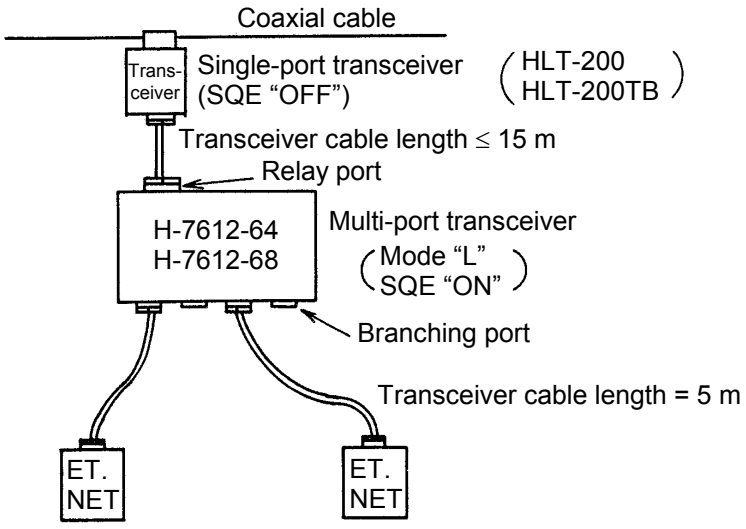


- Network mode

  In network mode, the transceiver is connected to the coaxial cable.

  Set the mode switch to 'N' (network mode).

  At this moment, set the SQE switch of the single-port transceiver connected to the relay port to 'OFF'.



(2) Switch setting

The multi-port transceiver is equipped with two switches; the functions of each switch are mentioned in Table 8-1.

Table 8-1 Switch Setting

Switch type	Switch position	Function	Setting at shipment time
SQE switch	Rear panel	Setting SQE function to ON/OFF	"ON"
Operation mode switch	Rear panel	Switching operation mode	'N' (network mode)

(3) Setting SQE switch on repeater connection

When connecting a repeater to a multi-port transceiver, set the SQE switch of the corresponding branching port of the multi-port transceiver to 'OFF'.

(4) Power switch

Set the switch on the rear panel to 'I', and the power of the multi-port transceiver is turned 'ON'.

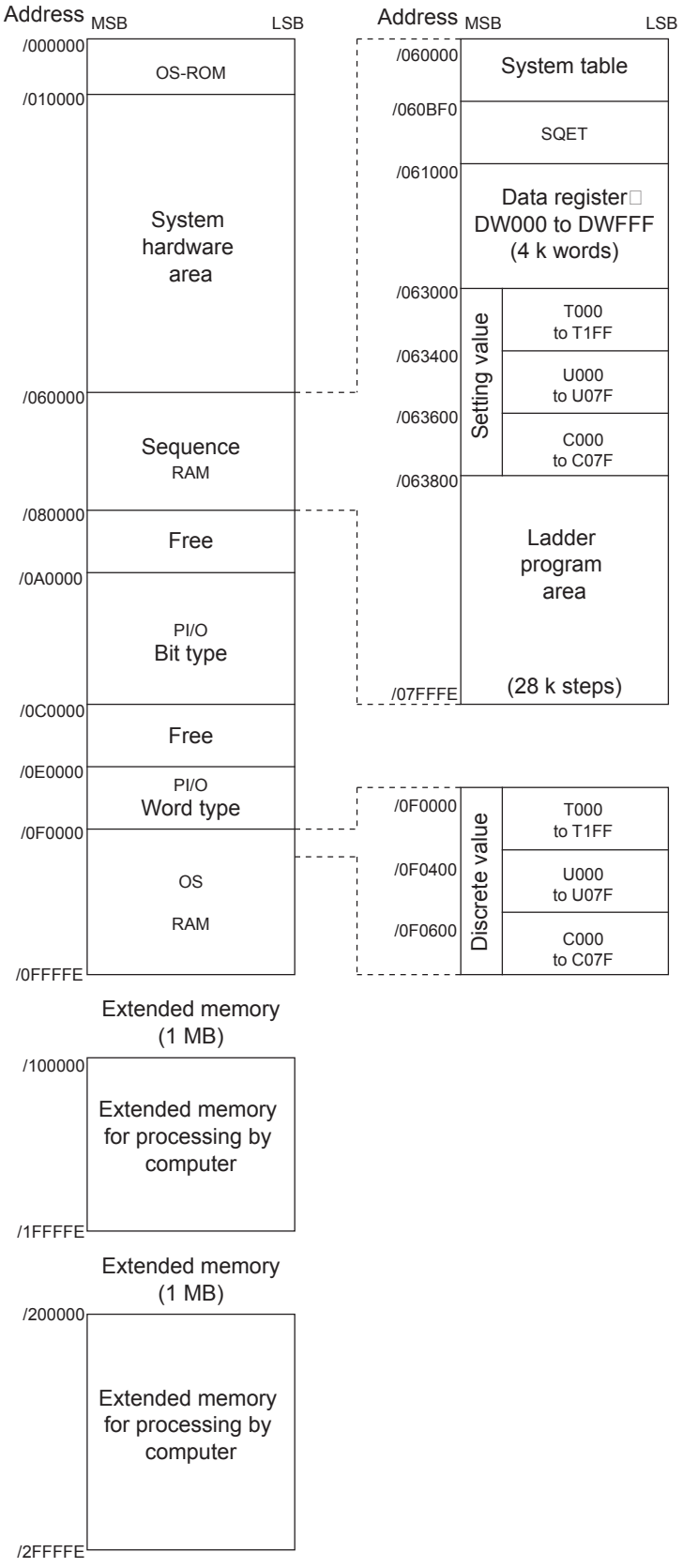
(5) LED display

The "POWER" LED and the "LINK" LEDs for each branching port are placed on the front panel of the cabinet.

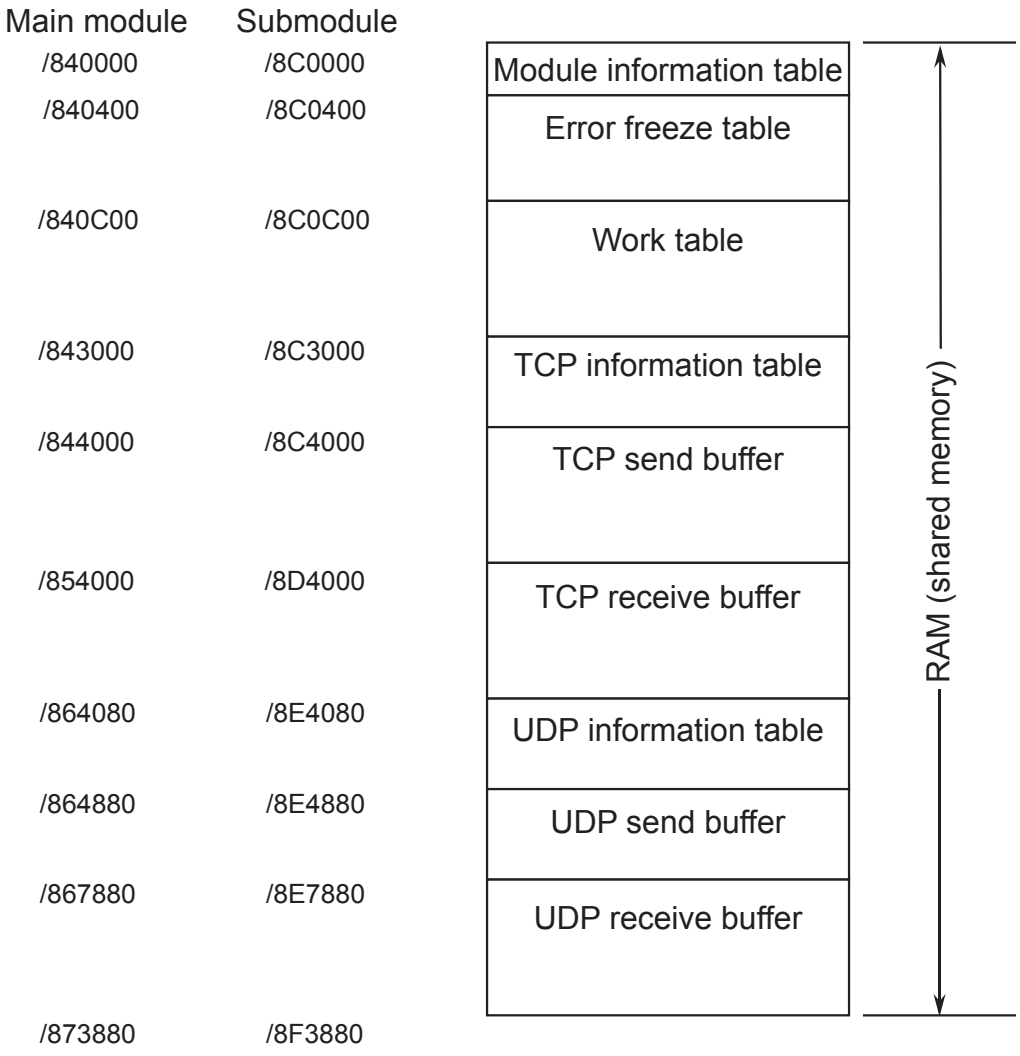
"POWER" LED: Lights when the power switch is 'ON'.

"LINK" LED: Lights when the information station is connected to the branching port of the multi-port transceiver (when 12 VDC is supplied from the information station).

8.14 CPU Memory Map



8.15 Memory Map of ET.NET Module



8.16 Trouble Investigation Sheet

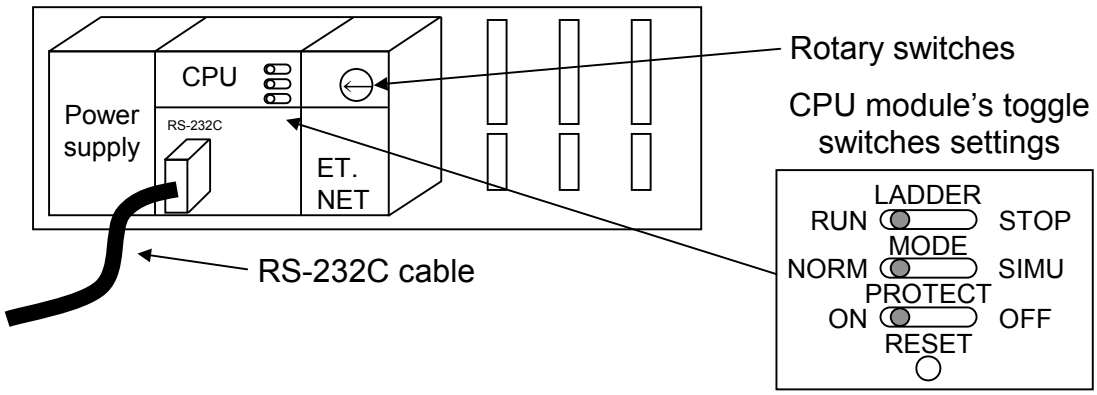
■ Trouble Investigation Sheet

Your company name				
Person in charge		Date and time of occurence		
Contact address and numbers	Address			
	Phone			
	Fax			
Model of defective module		CPU model		
OS Ver. Rev.	Program name:		Ver.	Rev.
Support program	Program name:		Ver.	Rev.
Symptom of defect				
Connection load	Type			
	Model			
	Cabling status			
System configuration and switch setting				
Space for correspondence				

SUPPLEMENTARY

Supplementary: Replacing or adding on the module

- What you should get in preparation
  ① Personal computer (with Hitachi's S10 ET.NET System installed in it)
  ② RS-232C cable
  ③ New or add-on ET.NET module (LQE020)
  ④ Copies of the parameter values for the module to be replaced. (These copies are prepared for use in cases where the parameters are not accessible for some reason.)
- Replacement procedure
  ① Write down, on a piece of paper, the current settings of the rotary switches that are, as shown below, accessible at the front side of the ET.NET module to be replaced.
  ② Write down also the current settings of three switches, labeled LADDER (toggle switch), MODE (toggle switch), and PROTECT (toggle switch), respectively, that are, as shown below, accessible at the front side of the CPU module.



  ③ Connect the personal computer and the CPU module together with the RS-232C cable.
  ④ Start Hitachi's S10 ET.NET System and make a hand-written record of the currently used IP address. (If the existing parameters are not accessible for some reason, use the copies of their set values [item ④] that were obtained in preparation.)
  ⑤ Set the CPU module's LADDER switch in STOP position and turn off the power supply of the controller unit.
  ⑥ Remove the connecting cables from the ET.NET module to be replaced.
  ⑦ Replace the existing ET.NET module with the new one and set the new ET.NET module's rotary switches in the same way as you wrote down in Step ①.
  ⑧ Turn on the power supply of the controller unit. Then, set the same IP address as you recorded in Step ④, by using the S10 ET.NET System.

⑨ Check that the set IP address is identical to the one that was recorded in Step ④.
⑩ Turn off the power supply of the controller unit.
⑪ Remove the RS-232C cable from both the personal computer and CPU module, which were connected together in Step ③.
⑫ Connect to the new ET.NET module the connecting cables that you removed in Step ⑥.
⑬ Set the CPU module's LADDER, MODE, and PROTECT switches in the same way as you wrote down in Step ②.
⑭ Turn on the power supply of the controller unit and check that the new ET.NET module is running normally.

- Add-on procedure

① Write down, on a piece of paper, the current settings of three switches, labeled LADDER (toggle switch) , MODE (toggle switch), and PROTECT (toggle switch), respectively, that are accessible at the front side of the CPU module, the one that is installed in the controller unit in which you are adding on a ET.NET module.
② Ensure that your application system has been shut down. Then, set the CPU module's LADDER switch in STOP position and turn off the power supply of the controller unit.
③ Mount the add-on ET.NET module in place according to the instructions given under "1.2 Mounting Optional Modules."
④ Set the add-on ET.NET module's rotary switches in such a way that a new module No. setting, which must be a sub-module No. setting, will not duplicate with the current rotary switch settings of the existing main ET.NET module.
⑤ Connect the personal computer and the CPU module together with the RS-232C cable. Then, turn on the power supply of the controller unit and set parameters for the add-on ET.NET module by using the S10 ET.NET System.
⑥ Turn off the power supply of the controller unit and connect the connecting cables to the add-on ET.NET module.
⑦ Set the CPU module's LADDER, MODE, and PROTECT switches in the same way as you wrote down in Step ①.
⑧ Remove the RS-232C cable from both the personal computer and CPU module, which were connected together in Step ⑤.
⑨ Turn on the power supply of the controller unit and check that the add-on ET.NET module is running normally.