# RL78/G1D Beacon Stack

## Multi-Hop Feature (without Security)

R01AN4375EJ0100
Rev.1.00
Dec. 14, 2018

## Introduction

This application note explains the sample program, which runs on Bluetooth® Low Energy microcontroller RL78/G1D device and executes uses Multi-Hop Feature by using RL78/G1D Beacon Stack.

The sample program consists of Application Layer and Multi-Hop Layer.

The Multi-Hop layer controls Advertising and Scan of Bluetooth Low Energy to transmit, receive and relay Multi-Hop frames with flooding method. In addition, this layer provides Security feature to authenticate and encrypt frames.

The application layer is provided to execute transmitting and receiving Multi-Hop frames by RL78/G1D Evaluation Board. When the switch on the evaluation board is pushed, the sample program transmits the Multi-Hop frames periodically. And upon receiving the frame addressed to own node, the sample program outputs the frame data log via UART.

Note that the following application notes are released for the Multi-Hop Feature. The difference is whether the security feature is implemented or not.

| | |
|---|---|
| Multi-Hop Feature (with Security) | R01AN4466 |
| Multi-Hop Feature (without Security) | R01AN4375 |

## Target Device

RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

## Related Documents

| Document Name | Document No. |
|---|---|
| RL78/G1D | |
| User's Manual: Hardware | R01UH0515 |
| RL78/G1D Evaluation Board | |
| User's Manual | R30UZ0048 |
| E1 Emulator | |
| User's Manual | R20UT0398 |
| Additional Document for User's Manual (Notes on Connection of RL78) | R20UT1994 |
| Renesas Flash Programmer V3.05 Flash memory programming software | |
| User's Manual | R20UT4307 |
| CC-RL Compiler | |
| User's Manual | R20UT3123 |
| RL78/G1D Beacon Stack | |
| User's Manual | R01UW0171 |

## Contents

# 1. Overview

## 1.1 Use-Case

This section explains use-cases using Multi-Hop.

### 1.1.1 Broadcasting a command (e.g. Lighting Control)

Figure 1-1 shows a use-case controlling lightings by using Multi-Hop.

Controller transmits a command to change a lighting level of either all or a part of lightings. When a lighting near the controller receives this command, it retransmits the command to relay to other lightings around. They also relay the command. In this way, the command is relayed by each lighting successively and reaches all lightings.

Even if some lightings are too far to receive a command from a controller directly, they can receive a command by using Multi-Hop.

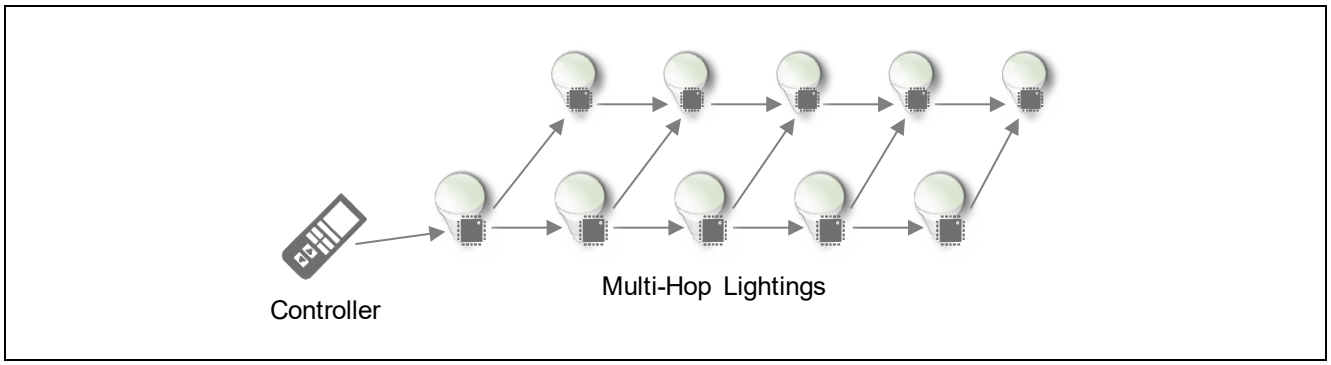



**Figure 1-1 Broadcasting a command**

### 1.1.2 Gathering a data (e.g. Sensor Data)

Figure 1-2 shows a use-case gathering sensor data by using Multi-Hop.

Each sensor transmits sensing data periodically. When other sensors receive these data, they retransmit the data to relay. In this way, these data are relayed successively and reaches a collector.

Even if some sensors are too far to transmit data to a collector directly, they can transmit data by using Multi-Hop.

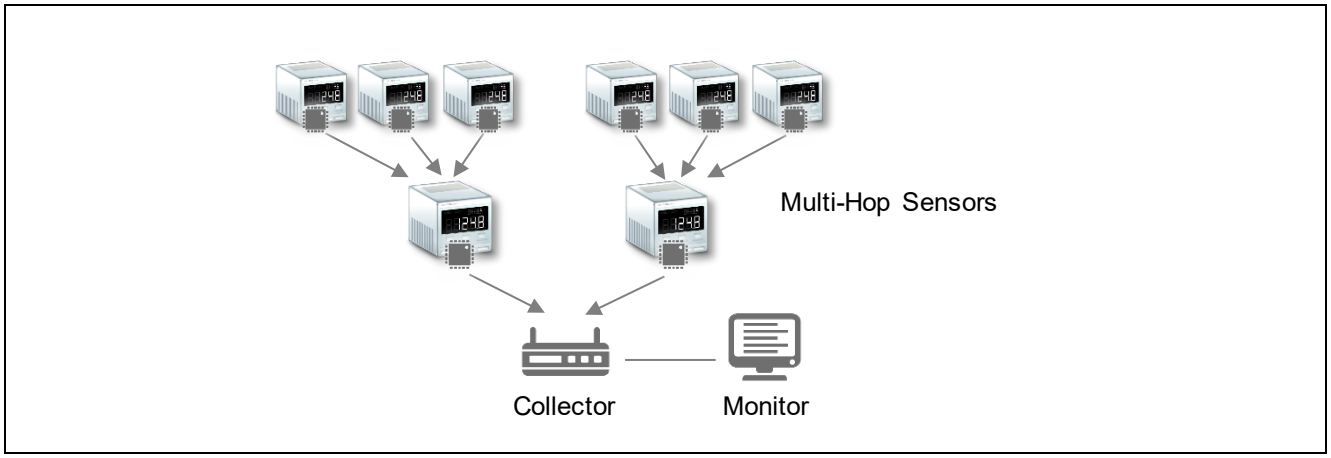



**Figure 1-2 Gathering a data**

## 2. System architecture

Figure 2-1 shows system architecture of the sample program.

- Application Layer

  Executes Multi-Hop frames transmission and reception.

- Multi-Hop Layer

  Controls Advertising and Scan to manages Multi-Hop frames transmission and reception.

- Beacon Stack

  Controls RF unit of RL78/G1D to execute Advertising and Scan.

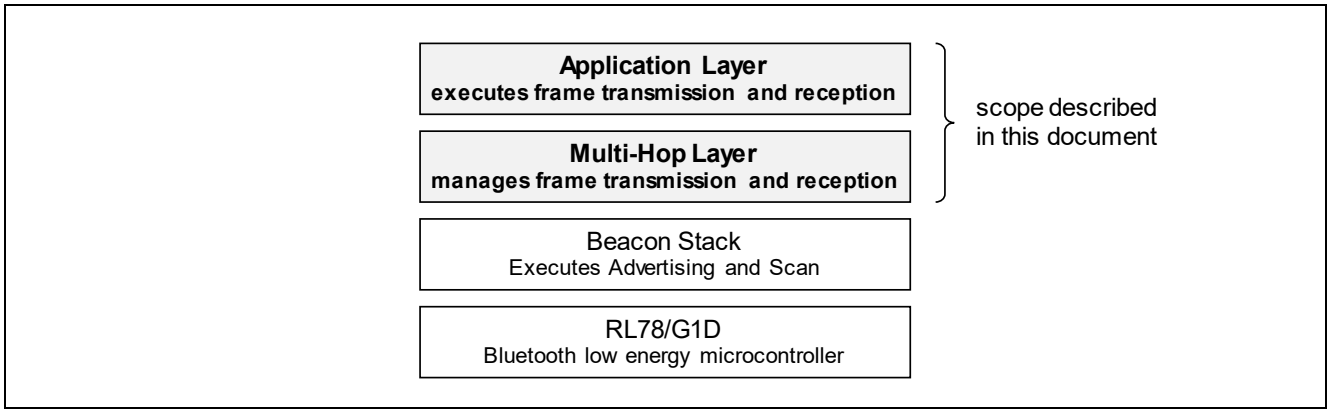



**Figure 2-1 System Architecture**

Application Layer and the Multi-Hop Layer are provided as the following source code files.

Application Layer: Project_Source\application\src\r_node.c

Multi-Hop Layer: Project_Source\application\src\r_multihop.c

Application Layer transmits and receives frames periodically by calling Multi-Hop API. You can develop various software using Multi-Hop feature by customizing this layer.

Multi-Hop Layer provides Multi-Hop Feature. Basically, it is not necessary to change this layer.

Regarding the specification of Beacon Stack, refer to RL78/G1D Beacon Stack User's Manual (R01UW0171).

Regarding the details of RL78/G1D, refer to RL78/G1D User's Manual: Hardware (R01UH0515).

Note that Beacon Stack Library included in this project is different from the one included in the application note of "RL78/G1D Beacon Stack Basic Operation" (R01AN3045) and is optimized for Multi-Hop Feature. When you develop an application, use the library included in this project.

# 3. Multi-Hop Layer Specification

This section explains Multi-Hop Layer specification. Regarding Multi-Hop API, refer to chapter 8 "Multi-Hop API" in this document.

## 3.1 Basic Operation

Figure 3-1 shows basic operation of Multi-Hop.

Each device participating a network is called a Multi-Hop Node. And each data unit to transmit an information is called a Multi-Hop Frame.

The frames are transmitted from originator node to destination node(s) by relaying. Each node executes the following steps. Each circle in this figure indicates a range that each node can transmit a frame directly.

1. Originator node transmits frame addressed to destination node.
2. When relay nodes receive the frame, the nodes retransmit the frame to relay.
3. Each relay nodes relay the frame successively, and then the frame reaches destination node. As a supplement, each relay node retransmits the frame after randomized time to avoid frame collision.
4. If already retransmitted frame is received, each relay node discards this frame.
5. If already retransmitted frame is received, destination node discards this frame.

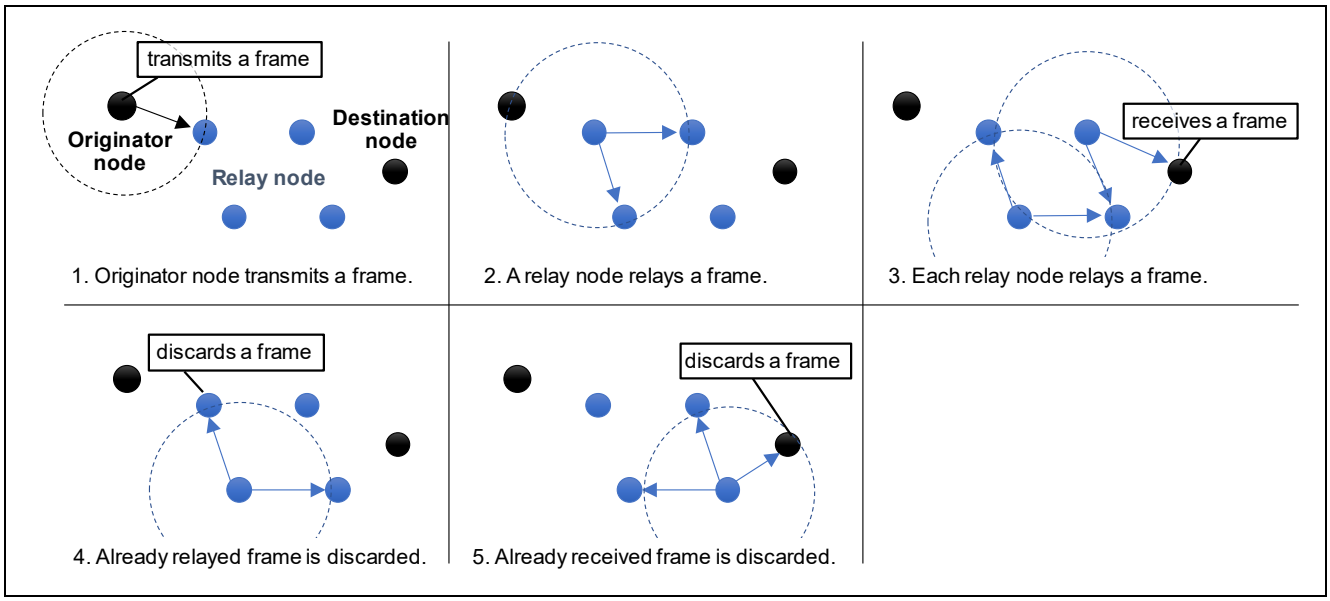



**Figure 3-1 Frame Transmission of Multi-Hop**

## 3.2 Networks and Nodes

Figure 3-2 shows networks and nodes of Multi-Hop.

Network is a cluster of nodes exchange frames. Each network is identified by a network ID. Each node in a same network is identified by a node ID. It is not affected that same node ID is assigned to nodes in different network.

Frames are transmitted and received among nodes in same network; frames cannot be transmitted to nodes in different network.

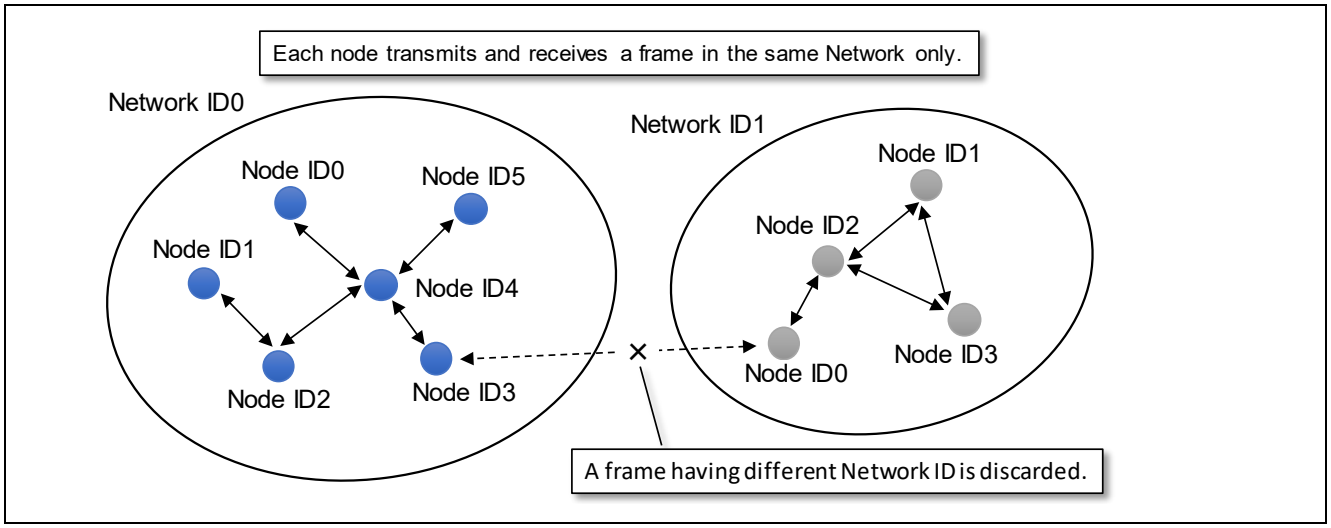



**Figure 3-2 Networks and Nodes**

To set individual a network ID and a node ID to each device, a mechanism called System Configuration is implemented in the sample program. Regarding the detail, refer to section 4.3 "System Configuration" in this document.

## 3.3 Security

Multi-Hop Layer provides Security feature to authenticate and encrypt frame. (R01AN4466 only)

Authentication and encryption are executed with AES-CCM*. To decrypt a frame encrypted by another node, all nodes should share same 128-bit encryption key in advance.

Security feature executes the following processing.

- When originator node transmits a frame, this feature encrypts data and adds MIC to frame.
- When relay node receives a frame, this feature calculates MIC of frame to authenticate.
- When destination node receives a frame, this feature authenticates frame and decrypts data.

By using Security feature, it can be expected the following.

- Data Encryption

  Security feature encrypts data, so data is not understood by a device which does not have encryption key.

- Frame Authentication

  Security feature authenticates, and discards frame falsified by a device which does not have encryption key. Moreover, because frame including invalid counter is discarded, replay attack can be prevented.

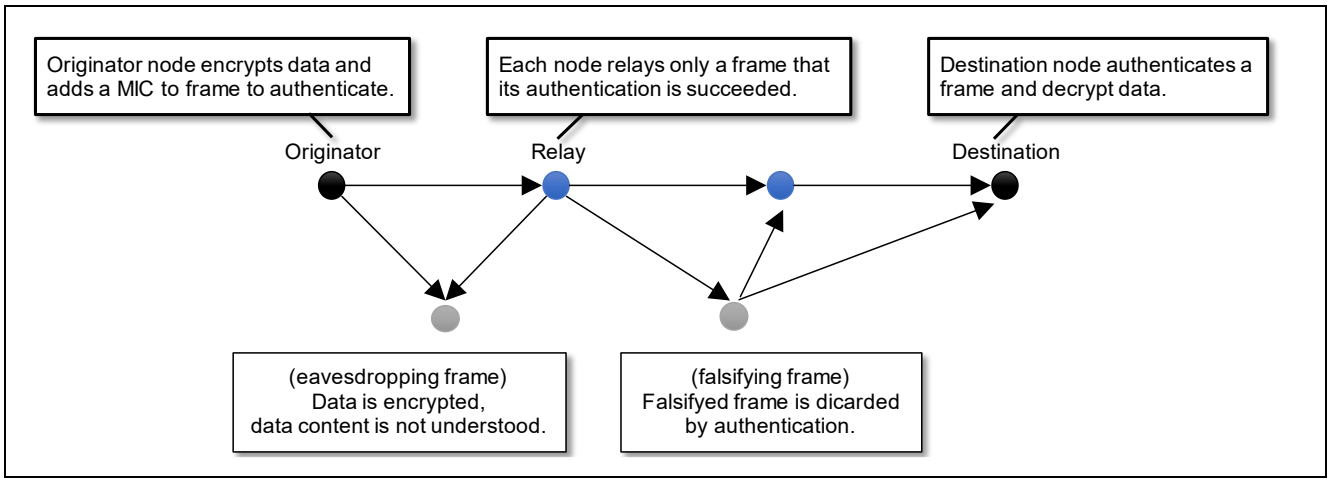



**Figure 3-3 Security with Encryption and Authentication**

When Security feature is enabled, Multi-Hop Layer discards all un-encrypted frame. Similarly, when Security feature is disabled, Multi-Hop Layer discards all encrypted frame.

Security feature uses AES Library. Regarding the detail, refer to the following URL.

AES Library for the RL78 Family

https://www.renesas.com/software-tool/crypto-library

Note that Security feature implemented in the sample program is not guaranteed that complete security of system.

Implement additional security mechanism, if necessary.

## 3.4 Multi-Hop Frame

Figure 3-4 and Table 3-1 show the field composition of Multi-Hop frame.

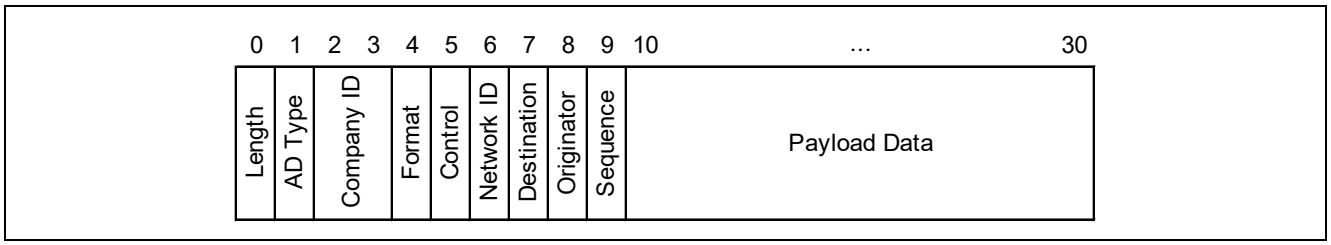



**Figure 3-4 Field Composition of Multi-Hop Frame**

Multi-Hop frame is transmitted as a <<Manufacturer Specific Data>> by Advertising packet of Bluetooth Low Energy. Regarding the specification of Manufacturer Specific Data, refer to Part A, Section 1.4 "MANUFACTURER SPECIFIC DATA" in Supplement to the Bluetooth Core Specification | CSS Version 7.

Network ID is set in a *Network ID* field. Node IDs of originator and destination are set to *Originator* and *Destination* field respectively. Data transferred from originator to destination is stored *Payload Data* field.

**Table 3-1 Field Composition of Multi-Hop Frame**

| Offset | Size (byte) | Field | Description |
|---|---|---|---|
| 0 | 1 | Length | AdvData Length (byte)<br>length from AD Type field to Payload Data field |
| 1 | 1 | AD Type | AdvData<br>0xFF: <<Manufacturer Specific Data>> |
| 2 | 2 | Company ID | Bluetooth Company ID (LSO: Least Significant Octet First)<br>https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers |
| 4 | 1 | Format | Multi-Hop Frame Format Version<br>0x01: Frame Format ver.1.00 |
| 5 | 1 | Control | Multi-Hop Control |

| Bit | Field | Description |
|---|---|---|
| 0:3 | hop limit | Remaining Hopping Limit Count<br>0 to 15<br>When a frame is relayed, hop limit is decreased by 1.<br>If hop limit is 0, frame is not relayed. |
| 4:5 | counter | Frame Transmission Count<br>This count shows the number of transmitting same frame.<br>0: first transmission<br>1: second transmission<br>2: third transmission<br>To improve transport factor, same frame is transmitted three times. |
| 6 | encrypted | Encrypted Flag<br>This flag shows whether payload data is encrypted or not.<br>0: not encrypted<br>1: encrypted (R01AN4466 only) |
| 7 | optional | Option Data Flag<br>This flag shows whether payload data is Option Data or not.<br>0: not Option Data<br>1: Option Data |

| | | | |
|---|---|---|---|
| 6 | 1 | Network ID | Network ID<br>0x00 to 0xFF |
| 7 | 1 | Destination | Destination Node ID<br>0x00 to 0xFE is transmitted to individual one node.<br>0xFF is transmitted to all nodes. |
| 8 | 1 | Originator | Originator Node ID<br>0x00 to 0xFE is ID of node transmitting frame. |
| 9 | 1 | Sequence | Sequence Number<br>This number is to identify each frame and used to check if frame is duplicated.<br>0x00 to 0xFF<br>This number is increased by 1 in each frame transmission. |
| 10 | Max.21 | Payload Data | Payload Data |

### 3.4.1 Encrypted Frame

Figure 1-1 and Table 3-2 show the field composition of Encrypted Multi-Hop frame.

When frame is encrypted, *Payload Data* field of the encrypted frame consists of three sub-field: *Counter*, *Encrypted Data*, and *MIC*. And the *encrypted* bit of *Control* field is set.

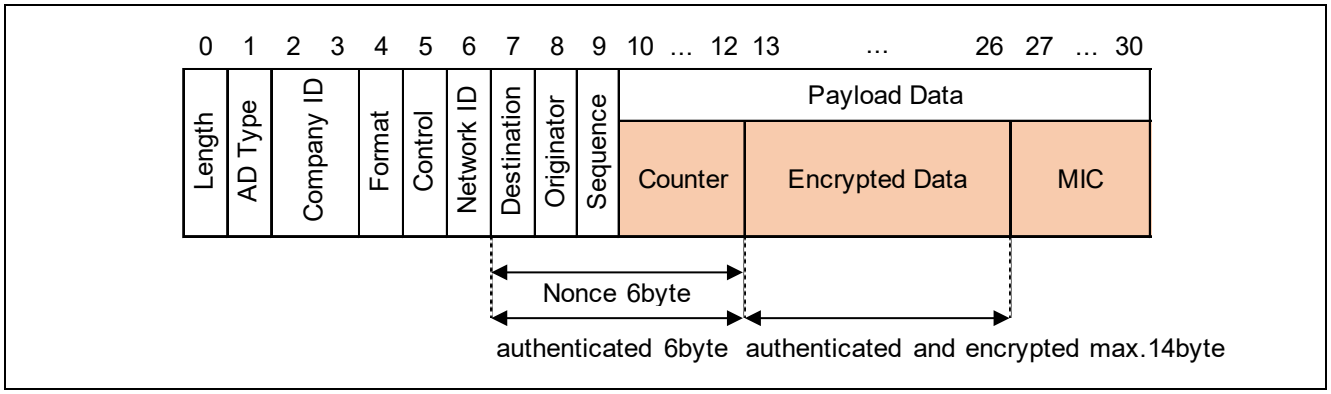



**Figure 3-5 Sub-field Composition of Payload Data in Encrypted Frame**

**Table 3-2 Field Composition of Encrypted Multi-Hop Frame**

| Offset | Size (byte) | Field | Description |
|---|---|---|---|
| 10 | 3 | Counter | Upper 3byte Counter Value<br>Sequence field and Counter field are used as a 4byte counter which is LSO (Least Significant Octet)-First.<br>This 4byte counter is used as Nonce for AES-CCM.<br>Each node has an individual Nonce Counter.<br>This Nonce is increased by 1 in each frame transmission.<br>(When Sequence is lapped around, Counter field is increased by 1.) |
| 13 | Max.14 | Encrypted Data | Encrypted Data |
| Max.27 | 4 | MIC | Message Integrity Code |

To randomize encrypted data, fields from *Destination* field to *Counter* field are used as a Nonce by Multi-Hop layer.

Fields to be authenticated are from *Destination* field to *Encrypted Data* field. Field to be encrypted is *Encrypted Data* field.

*MIC: Message Integrity Code* is added to the last of frame.

### 3.4.2 Option Data Frame

A feature to check relayed path (see Section 9.1) is implemented in Multi-Hop Layer. This feature uses Option Data Frame described in the following.

Figure 3-6, Figure 3-7, and Table 3-3 show the field composition of Multi-Hop Option Data Frame.

*Payload Data* field of the option data frame consists of three sub-field: *ID*, *Length*, and *Option Data*. And the *optional* bit of *Control* field is set.

Upon receiving the option data frame, Multi-Hop Layer executes option feature in accordance with Option Data ID.

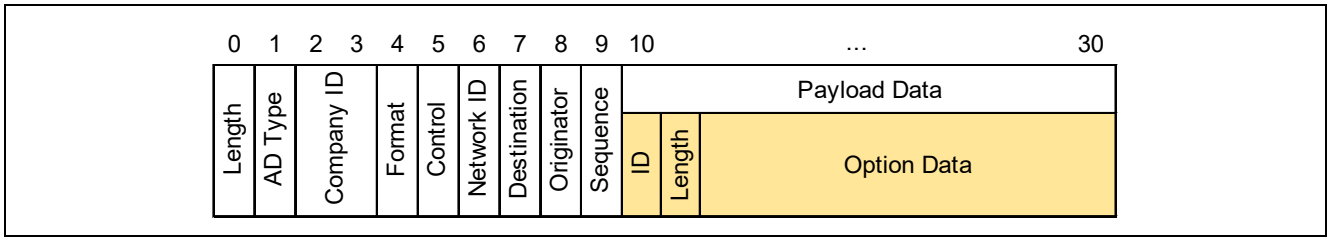



**Figure 3-6 Sub-field Composition of Payload Data in Option Data Frame**

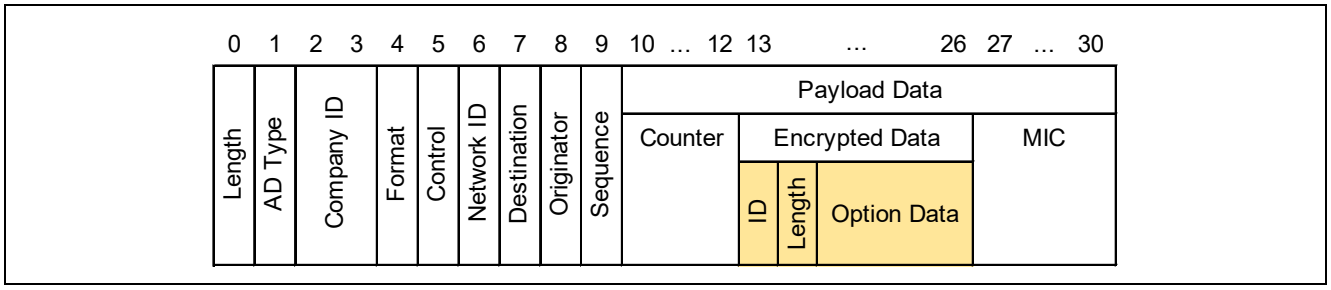



**Figure 3-7 Sub-field Composition of Payload Data in Encrypted Option Data Frame**

**Table 3-3 Field Composition of Multi-Hop Option Data Frame**

| Offset | Size (byte) | Field | Description |
|---|---|---|---|
| 10 | 1 | ID | Option Data ID |
| 11 | 1 | Length | Option Data Length |
| 12 | Max.19<br>(If encrypted, Max.12) | Option Data | Option Data |

Regarding the definition of Option Data Structure, refer to subsection 8.3.5 "Option Data Structure" in this document.

## 3.5 Transmitting and Receiving

Each node transmits a frame by Advertising and receives a frame by Scan. Figure 3-8 shows an example of how frame is transmitted and received. In this figure, there are nodes assigned as ID0 to ID3 respectively, and they relay a frame on the path of ID3→ ID2→ ID1→ ID0.

By executing the following transmission and reception processing, frame is transferred from originator node to destination node.

1. All nodes start receiving operation.
2. Node of ID3 transmits a frame addressed to ID0.

   This node transmits a frame three times to reach to other nodes as many as possible.
3. Nodes of ID2 and ID1 receives the frame and then retransmits the frame received.

   Each node retransmits the frame three times to reach to other nodes as many as possible.

   Each node retransmits the frame at randomized timing to avoid frame collision.

   After that, if same frame is received again, each node discards the frame and retransmit no frame.
4. Node of ID0 receives the frame addressed to ID0, frame data is notified to application.

   After that, if same frame is received again, the frame is discarded.

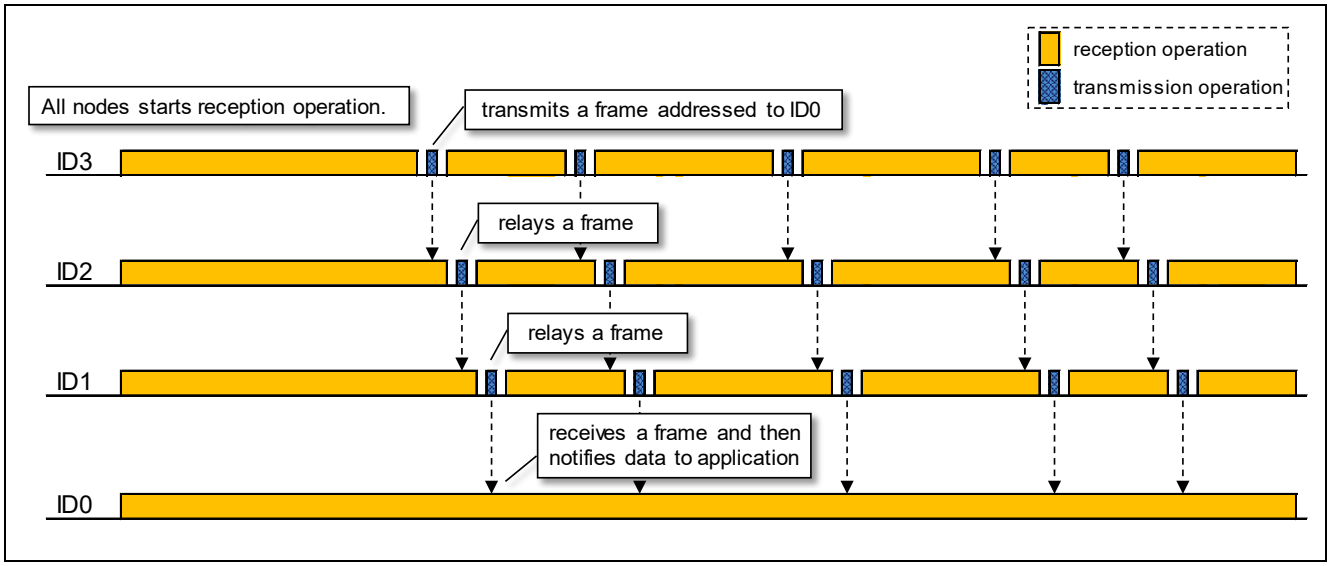



**Figure 3-8 Example of Frame Transmission and Reception**

Frame is transmitted three times to all advertising channels. Regarding the details, refer to section 8.8 "Frame Transmission".

Receiving operation switches advertising channels periodically. Regarding the details, refer to section 8.7 "Frame Reception".

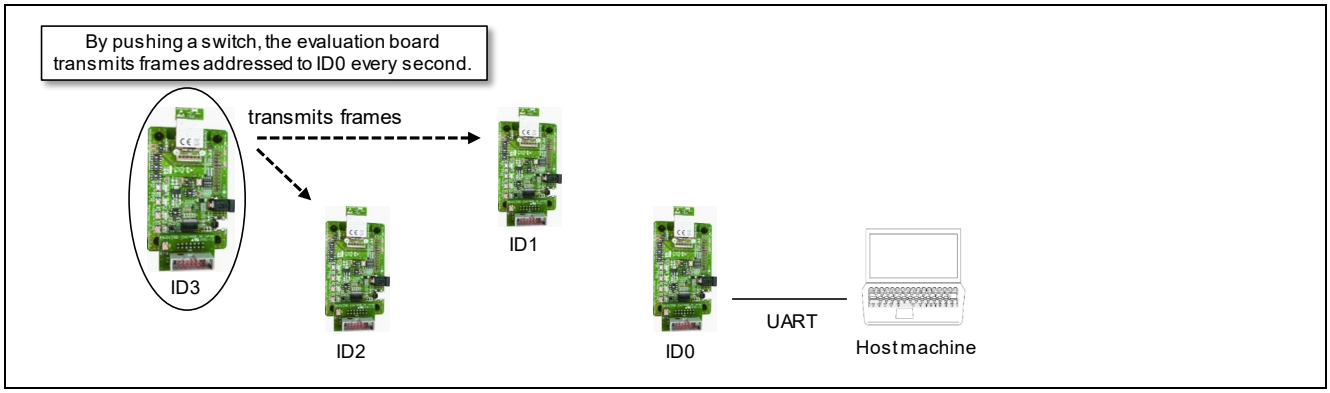
# 4. Application Layer Specification

This chapter explains Application Layer Specification. Regarding the operation procedure of the sample program, refer to chapter 5 "Operating Procedure".

## 4.1 Operation

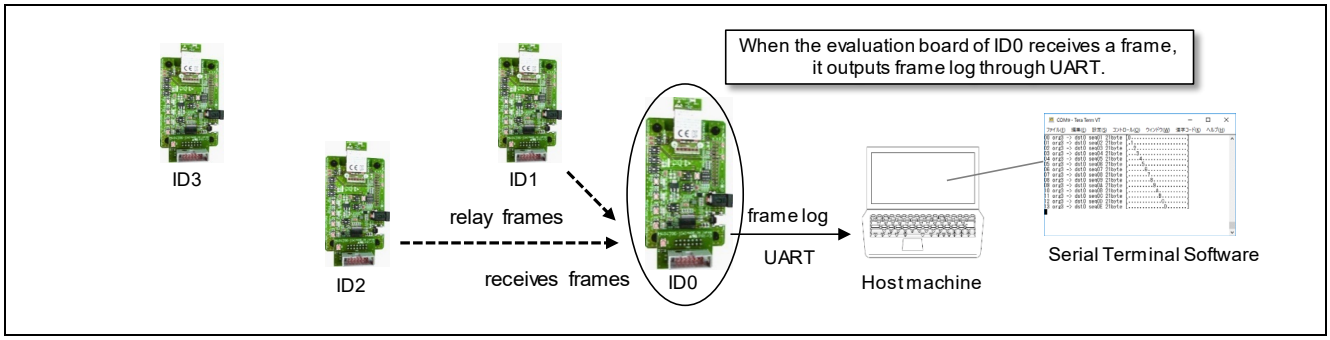
The sample program runs on the evaluation board. It starts receiving operation after starting-up. If switch is pushed, it transmits frames addressed to ID0 periodically.

For example, there are four evaluation board assigned ID as from ID0 to ID3. By pushing a switch of evaluation board ID3, it transmits a frame addressed to ID0 every second. Each frame has a different sequence number.




Upon receiving a frame, the evaluation board of ID1 and ID2 relay the frame respectively. Relaying is executed by Multi-Hop layer automatically and Application layer is not notified that relay operation is executed.

Upon receiving a frame, the evaluation board of ID0 outputs a frame data log via UART. After that, if it receives a frame which has same sequence number again, log is not output.



## 4.2 Frame Data

When switch SW2 is pushed, the application transmits frames to ID0 periodically.

Data to be transmitted is implemented as an array in the following source code. The sample program transmits data in sequence repeatedly.

- Project_Source\application\src\r_node.c

```
101:    static RMH_DATA demo_payload_data[] =
102:    {
103:        {{"0....................."}, 21, 0},
104:        {{".1...................."}, 21, 0},
105:        {{"..2..................."}, 21, 0},
106:        {{"...3.................."}, 21, 0},
107:        {{"....4................."}, 21, 0},
108:        {{".....5................"}, 21, 0},
109:        {{"......6..............."}, 21, 0},
110:        {{".......7.............."}, 21, 0},
111:        {{"........8............."}, 21, 0},
112:        {{".........9............"}, 21, 0},
113:        {{"..........A..........."}, 21, 0},
114:        {{"...........B.........."}, 21, 0},
115:        {{"............C........."}, 21, 0},
116:        {{".............D......."}, 21, 0},
117:        {{"..............E......"}, 21, 0},
118:        {{"...............F....."}, 21, 0},
119:    };
```

data: Frame Data

len: Frame Data Size (Max.21byte)

node: Destination Node ID (0x00 to 0xFF)

Node of ID0 outputs the received frame data log via UART with the following format.

*SerialNumber ORG* -> *DST SEQ FrameDataSize* [*FrameData (ASCII)*]

For example, node of ID3 transmits frames and then node of ID0 receives the frames. Node of ID0 outputs the data log as shown in Figure 4-1. By checking log, you can confirm if node of ID0 receives frames transmitted by node of ID3 sequentially.

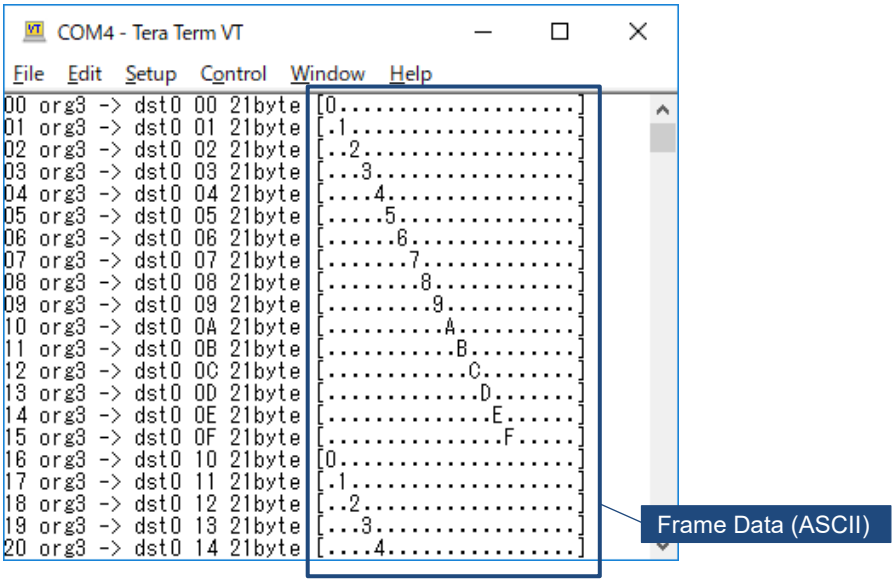



**Figure 4-1 Example Log of Received Data**

When Security feature is enabled, frame data is encrypted by Multi-Hop layer. Data to be encrypted and transmitted is also implemented as an array in the following source code. (R01AN4466 only) The sample program encrypts and transmits data in sequence repeatedly.

- Project_Source\application\src\r_node.c

```
125:    static RMH_DATA demo_sec_data[] =
126:    {
127:        {{"0............"}, 14, 0},
128:        {{".1............"}, 14, 0},
129:        {{"..2............"}, 14, 0},
130:        {{"...3............"}, 14, 0},
131:        {{"....4............"}, 14, 0},
132:        {{".....5............"}, 14, 0},
133:        {{"......6............"}, 14, 0},
134:        {{".......7............"}, 14, 0},
135:        {{"........8............"}, 14, 0},
136:        {{".........9............"}, 14, 0},
137:        {{"..........A..."}, 14, 0},
138:        {{"...........B.."}, 14, 0},
139:        {{"............C."}, 14, 0},
140:        {{".............D"}, 14, 0},
141:    };
```

node: Destination Node ID (0x00 to 0xFF)

data: Frame Data

len: Frame Data Size (Max.14byte)

For example, node of ID3 transmits encrypted frames. By the way, Scan program (see section 5.4) is included in this application note. It scans all Multi-Hop frames and outputs frame log as shown in **Figure 4-2**. By using Scan program, you can confirm that data content is encrypted.

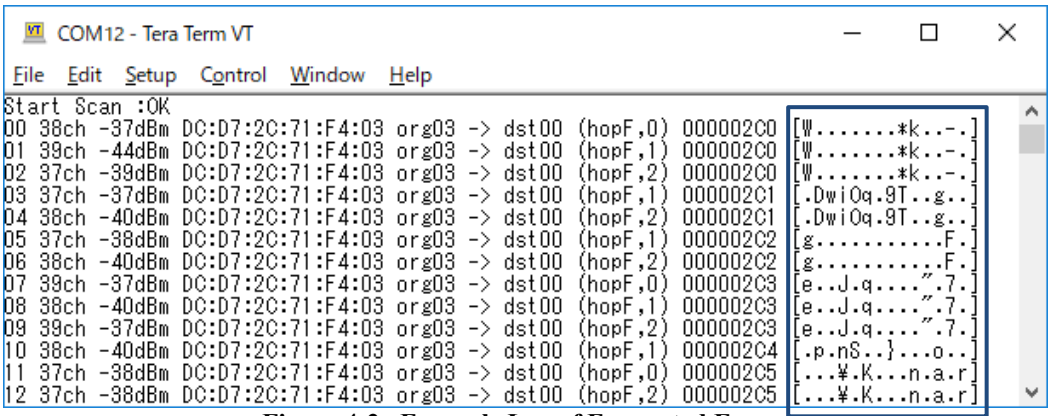



**Figure 4-2 Example Log of Encrypted Frame**

When node of ID0 receives the encrypted frame, the node outputs frame log as shown in Figure 4-3 By checking log, you can confirm that encrypted data is decrypted correctly.

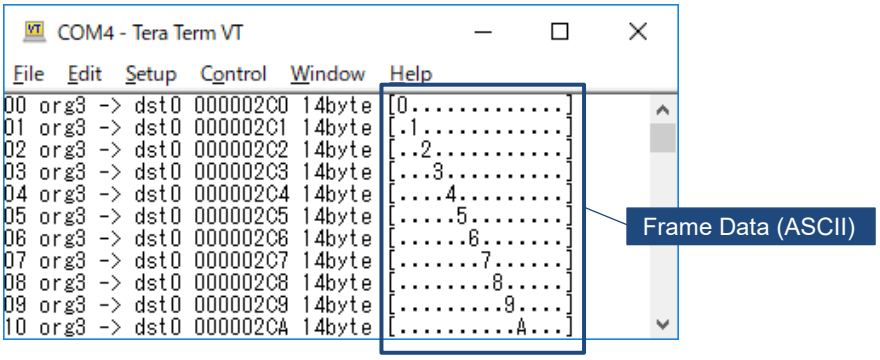



**Figure 4-3 Example Log of Received Data**

## 4.3 System Configuration

Each node needs to be assigned ID which is different from each other. So, the sample program provides a mechanism to set individual parameters which is called System Configuration.

Figure 4-4 shows the overview of System Configuration.

To write the system configuration, you can use the Unique Code Embedding Function of Renesas Flash Programmer. And a list of the system configuration is described to Unique Code data file.

By executing Renesas Flash Programmer, you can write common firmware and individual system configuration to each node.

When the firmware starts to run, the sample program reads parameters such as node ID from the system configuration and uses the parameters as a Multi-Hop setting.

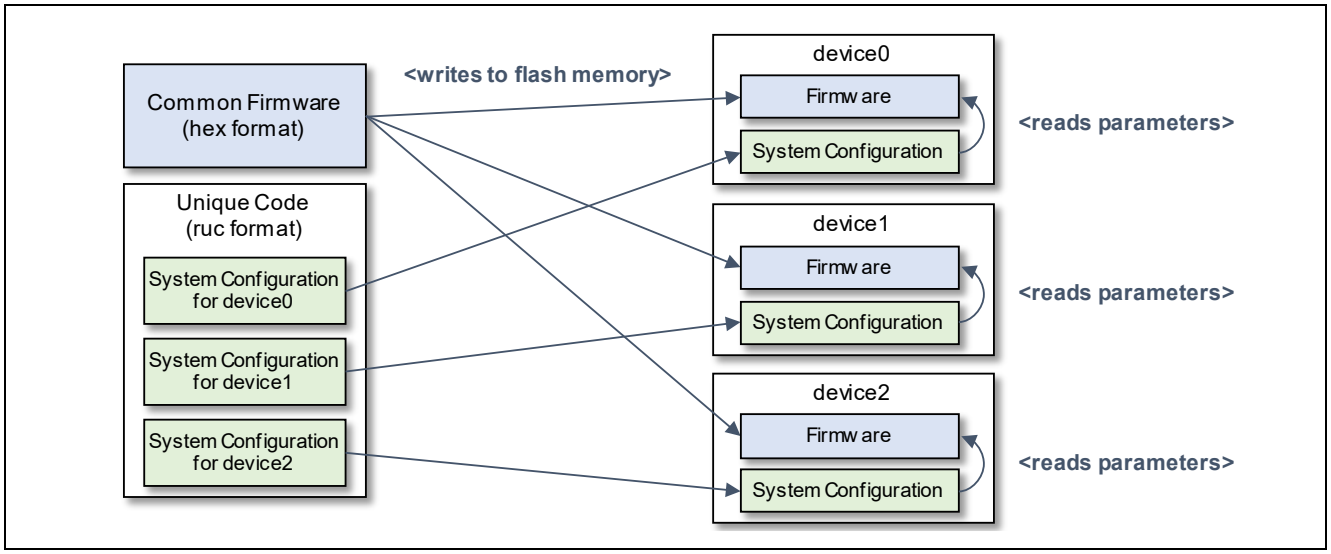



**Figure 4-4 System Configuration**

Table 4-1 shows the start address of the system configuration.

The system configuration is allocated out of the program area in Code Flash memory. You can change the start address of the system configuration by modifying the source code.

**Table 4-1 Start Address of System Configuration**

| RL78/G1D | Start Address |
|---|---|
| R5F11AGG | 0x1FC00 |
| R5F11AGH | 0x2FC00 |
| R5F11AGJ | 0x3F400 |

Table 4-2 shows the system configuration specification. You can change the specification by modifying the source code.

Note1: When you use RL78/G1D module (RY7011), the sample program uses a device address not in the system configuration but in Block255 of Code Flash memory preferentially, which is written when manufacturing.

**Table 4-2 Structure of System Configuration**

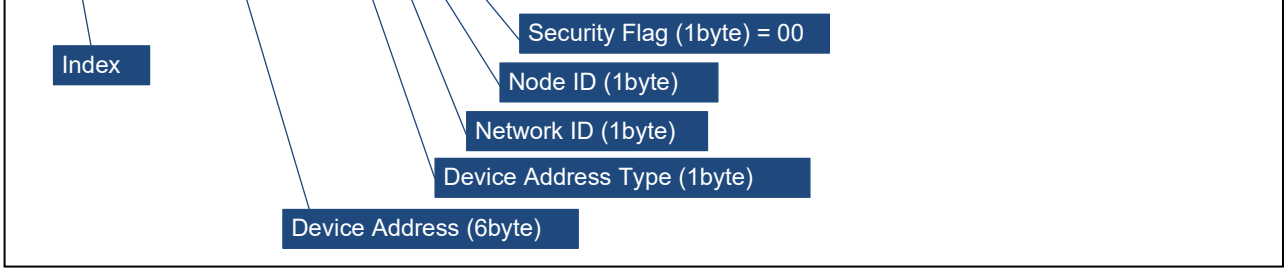
| Offset | Size (byte) | Data |
|---|---|---|
| 0 | 6<br>(RBLE_BD_ADDR Structure) | Device Address, See Note1 |
| 6 | 1<br>(uint8_t type) | Device Address Type<br>0x00: public, 0x01: random |
| 7 | 1<br>(uint8_t type) | Network ID<br>0x00 to 0xFF |
| 8 | 1<br>(uint8_t type) | Node ID<br>0x00 to 0xFE |
| 9 | 1<br>(bool type) | Security Flag<br>0x01: enable Security (R01AN4466 only)<br>0x00: disable Security |
| 10 | 16<br>(uint8_t[16] type) | 128bit Encryption Key<br>If Security is disabled, there is no need to set this key. |

The system configuration is described in the following unique code file.

- RUC_File\r5f11agj_syscfg.ruc

The unique code file is shown below. Each system configuration is indexed and written to each RL78/G1D in order. Off course, you can edit this file.

```
// ---------------------------------------------------------------
// -- System Configuration for RL78/G1D Multi Hop Sample Program --
// --   Device Part Number : R5F11AGJ                            --
// ---------------------------------------------------------------
format hex
area user flash
address 0x3f400
size 10
index data
//     |----------|        uint8_t[6]: Device Address (LSB-first)
//                ||             uint8_t: Device Address Type (00:Public, 01:Random)
//                 ||            uint8_t: Network ID (0x00 to 0xFF)
//                  ||    uint8_t: Node ID (0x00 to 0xFE)
//                   ||   bool: Security Flag (00:disable, 01:enable)
000000 00F4712CD7DC010000
000001 01F4712CD7DC010001
000002 02F4712CD7DC010002
000003 03F4712CD7DC010003
000004 04F4712CD7DC010004
000005 05F4712CD7DC010005
000006 06F4712CD7DC010006
000007 07F4712CD7DC010007
```



The system configuration to enable Security feature is described in the following unique code file. (R01AN4466 only)

- RUC_File\r5f11agj_syscfg_sec.ruc

List of the unique code file is shown below. To enable the security feature, Security Flag is set and 128bit Encryption Key is added at last to each system configuration.

```
// ------------------------------------------------------------------
// -- System Configuration for RL78/G1D Multi Hop Sample Program --
// --   Device Part Number : R5F11AGJ                              --
// ------------------------------------------------------------------
format hex
area user flash
address 0x3f400
size 10
index data
//     |----------|        uint8_t[6]: Device Address (LSB-first)
//                ||             uint8_t: Device Address Type (00:Public, 01:Random)
//                 ||            uint8_t: Network ID (0x00 to 0xFF)
//                  ||    uint8_t: Node ID (0x00 to 0xFE)
//                    ||  bool: Security Flag (00:disable, 01:enable)
//                |-----------------------------| uint8_t[16]: Encryption Key
(128bit)
000000 00F4712CD7DC0100000101020304050607080090A0B0C0D0E0F10
000001 01F4712CD7DC0100010101020304050607080090A0B0C0D0E0F10
000002 02F4712CD7DC0100020101020304050607080090A0B0C0D0E0F10
000003 03F4712CD7DC0100030101020304050607080090A0B0C0D0E0F10
000004 04F4712CD7DC0100040101020304050607080090A0B0C0D0E0F10
000005 05F4712CD7DC0100050101020304050607080090A0B0C0D0E0F10
000006 06F4712CD7DC0100060101020304050607080090A0B0C0D0E0F10
000007 07F4712CD7DC0100070101020304050607080090A0B0C0D0E0F10
```

Encryption Key (16byte)

Security Flag (1byte) = 01

## 4.4 Sequence

Figure 4-4 shows a sequence of the application layer.

When the application layer starts, initialization function: node_init() executes R_MH_Init() to initialize Multi-Hop layer and then calls R_MH_Receive() to start receiving operation.

When a switch is pushed, an external input interrupt: INTP5 occurs and then interrupt handler: input_callback() executes R_MH_Send() to transmit frame. And this handler starts 12-bit interval timer operation.

Furthermore, interval interrupt: INTIT of 12-bit interval timer occurs every second and then interval handler: it_callback() executes R_MH_Send() to transmit a frame.

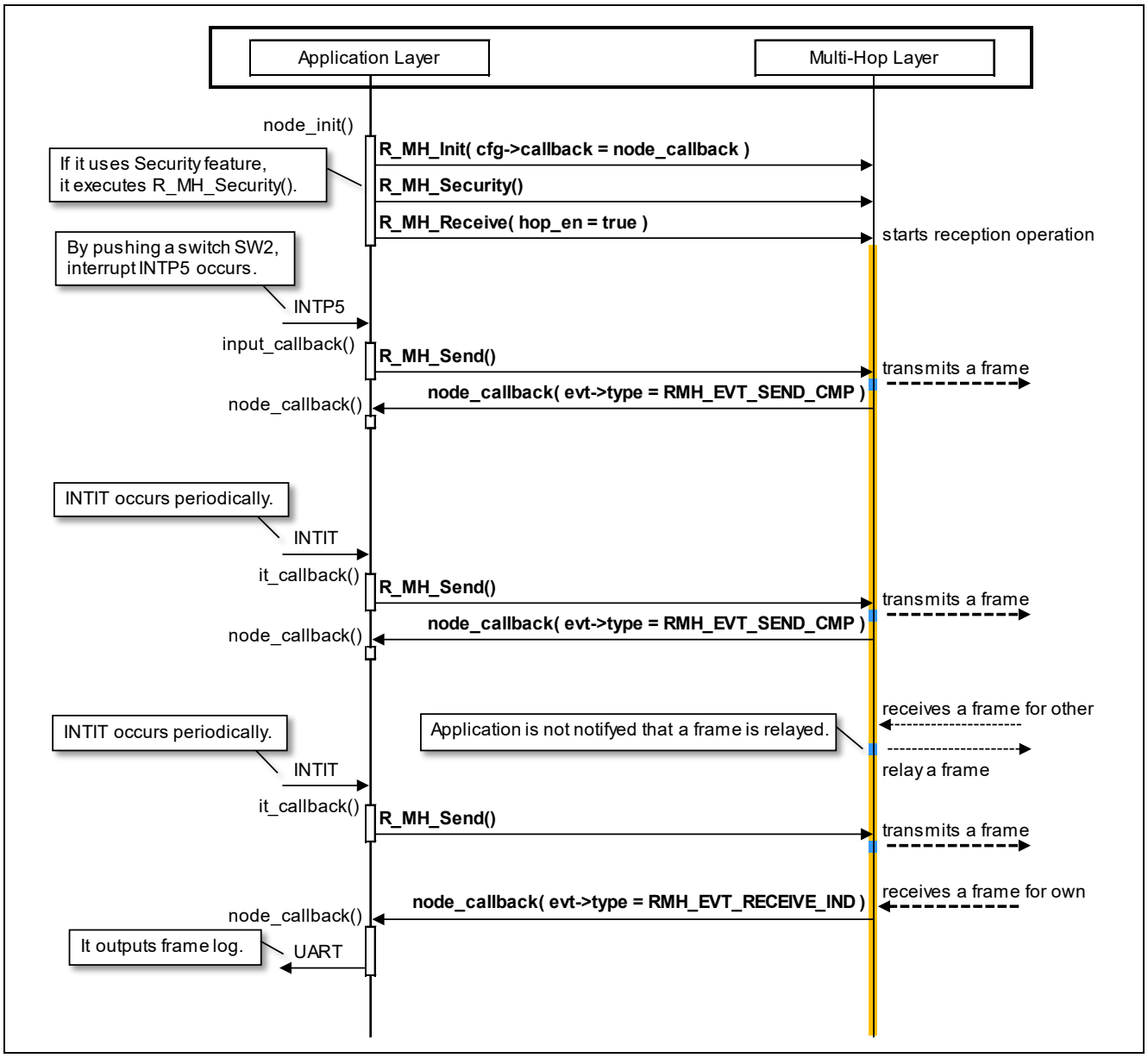



**Figure 4-5 Sequence of Sample Program**

Regarding the Multi-Hop API specification, refer to chapter 8 "Multi-Hop API".

Note that the application layer should execute R_MH_Proc() periodically. By executing R_MH_Proc(), Multi-Hop layer relays frames to other nodes or notifies events by calling a callback function: node_callback(). Regarding how to implement, refer to subsection 8.4.2 "R_MH_Proc".

# 5. Operating Procedure

This chapter explains the operating procedure of the sample program.

## 5.1 Operation Environment

To write a firmware of the sample program to the evaluation board, you can use Flash memory programming software: Renesas Flash Programmer.

Renesas Flash Programmer (Programming GUI)
https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui

The necessary hardware and software environment for evaluating the sample program is as follow:

- Hardware Environment
  - Host
    - PC/AT™ compatible computer
  - Device
    - RL78/G1D Evaluation Board (RTK0EN0001D01001BZ): at least 2 boards
    - USB cable (A type make/mini-B type make): 2 cables
  - Tool
    - Renesas On-chip Debugging Emulator E1 (R0E000010KCE00)

- Software Environment
  - Windows® 10
  - Renesas Flash Programmer v3.05.00
  - Tera Term Pro (or Terminal software which can connect to serial port)
  - UART-USB conversion device driver

Note: It may be that device driver for UART-USB conversion IC *FT232RL* is requested when you connect RL78/G1D Evaluation Board to Host first time. In this case, you can get the device driver from below website.

- FTDI (Future Technology Devices International) - Drivers
  http://www.ftdichip.com/Drivers/D2XX.htm

## 5.2 Slide-Switch Setting

Figure 5-1 shows slide-switch of RL78/G1D Evaluation Board.

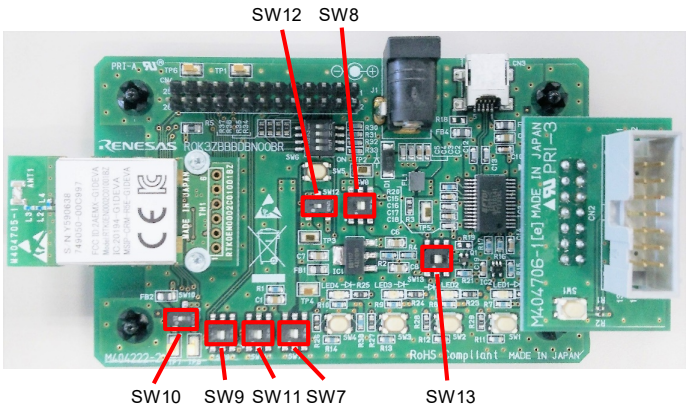



**Figure 5-1 Slide-Switch on RL78/G1D Evaluation Board**

Table 5-1 shows the slide-switch setting to evaluate the sample program.

**Table 5-1 Slide-Switch Setting**

| Switch | Setting | Description |
|---|---|---|
| SW7 | 2-3 connected (right) | Power is supplied from DC/USB VBUS via a regulator.<br>If 1-2 is connected (left), power is directly supplied from a battery. |
| SW8 | 2-3 connected (right) | Power is supplied from USB VBUS to a regulator.<br>If 1-2 is connected, power is supplied from DC to a regulator. |
| SW9 | 2-3 connected (right) | Connected to the USB device. |
| SW10 | 1-2 connected (left) | Power is supplied to the module. |
| SW11 | 2-3 connected (right) | Power is supplied from a source other than the E1 debugger (3.3V). |
| SW12 | 2-3 connected (right) | Unused |
| SW13 | 1-2 connected (left) | USB interface is connected |

Regarding the slide-switch of the evaluation board, refer to the section 6.1 "Power Line System" in RL78/G1D Evaluation Board User's Manual (R30UZ0048).

## 5.3 Writing Firmware

Figure 5-2 shows the overview of writing firmware.

Host machine and E1 Emulator are needed to write the firmware. By using Renesas Flash Programmer on the host machine, you can write firmware. In addition, by using the Unique Code Embedded Function of Renesas Flash Programmer, you can set an individual node ID to each RL78/G1D evaluation board respectively.

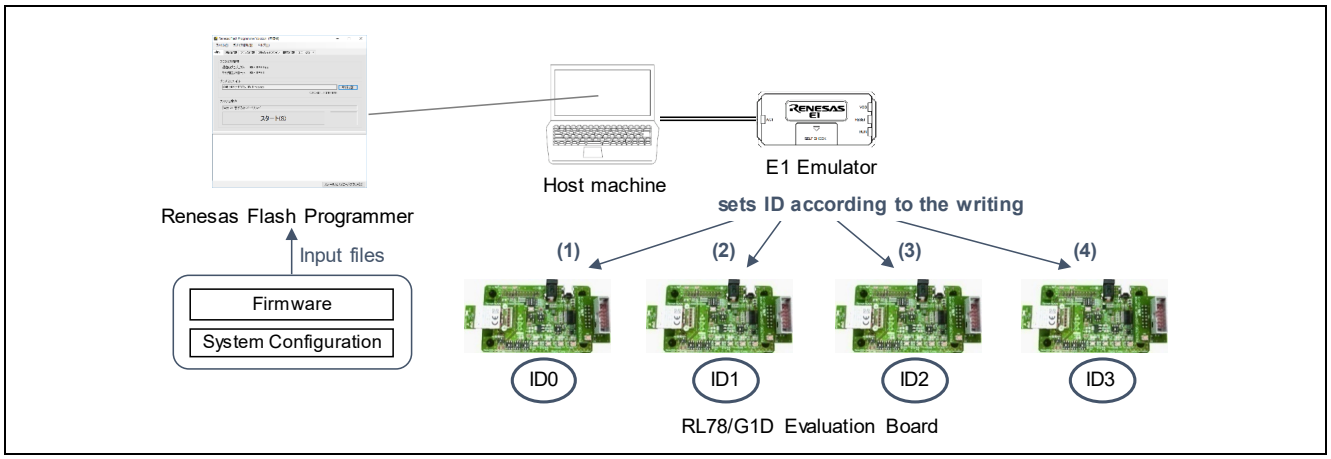



**Figure 5-2 Writing Firmware**

Regarding the details of E1 Emulator, refer to E1 Emulator User's Manual (R20UT0398) and E1 Emulator Additional Document for User's Manual (Notes on Connection of RL78) (R20UT1994).

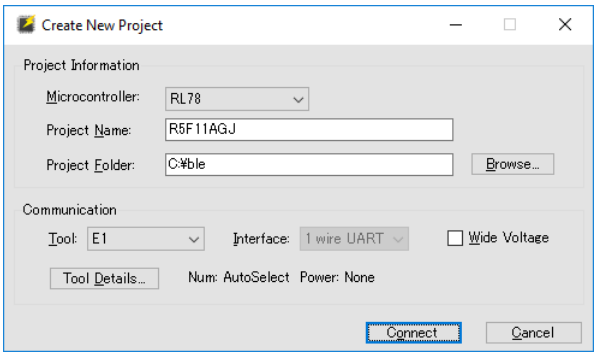
How to write a firmware to RL78/G1D Evaluation Board is below:

1. Connect E1 Emulator to the evaluation board as well as E1 Emulator to Host machine.
2. Connect the evaluation board to Host machine or AC-USB power supply adapter to supply power.
3. Start Renesas Flash Programmer and create a project according to the following steps.

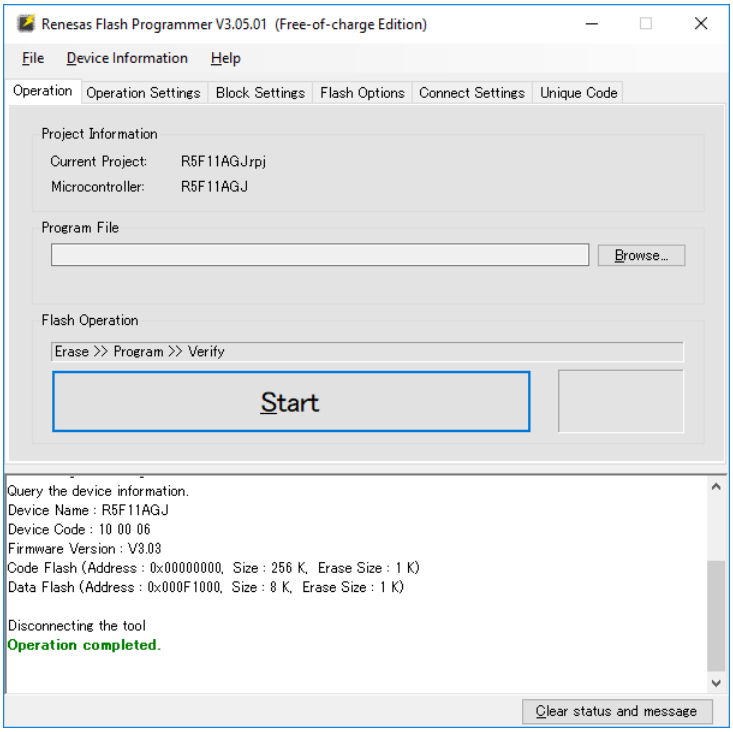
   Once you create a project, you can skip these steps.

   3-1. Select [File]→[Create a new project].

   3-2. Select [RL78] as a Microcontroller, input a project name and click [Connect] in [Create New Project] dialog.



3-3. Confirm [Operation completed] message in Log output panel.



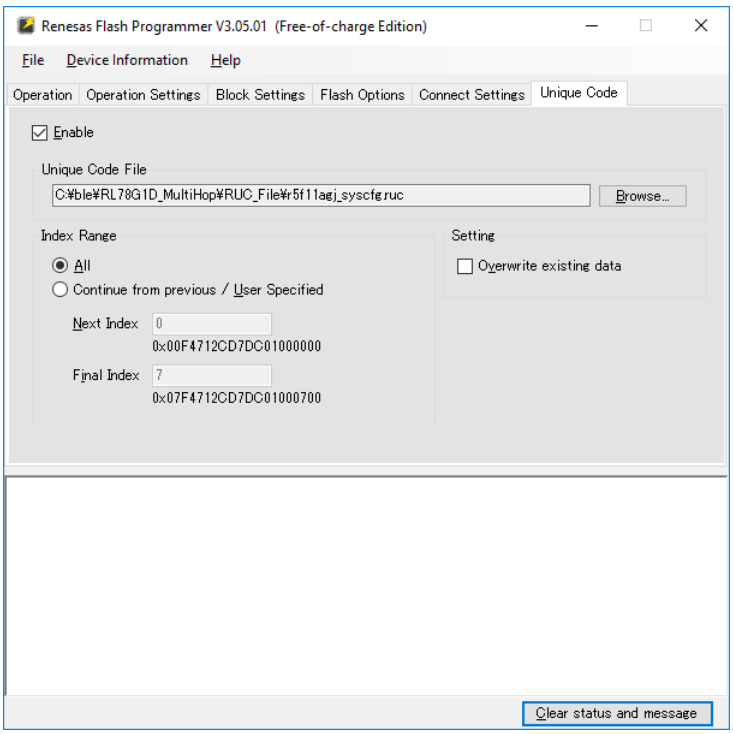

4. Specify a unique code file describing the system configuration according to the following steps.

4-1. Select [Unique Code] tab.

4-2. Check [Enable].

4-3. Specify the below unique code file at [Unique Code File].

- (R01AN4375) RUC_File\r5f11agj_syscfg.ruc
- (R01AN4466) RUC_File\r5f11agj_syscfg_sec.ruc



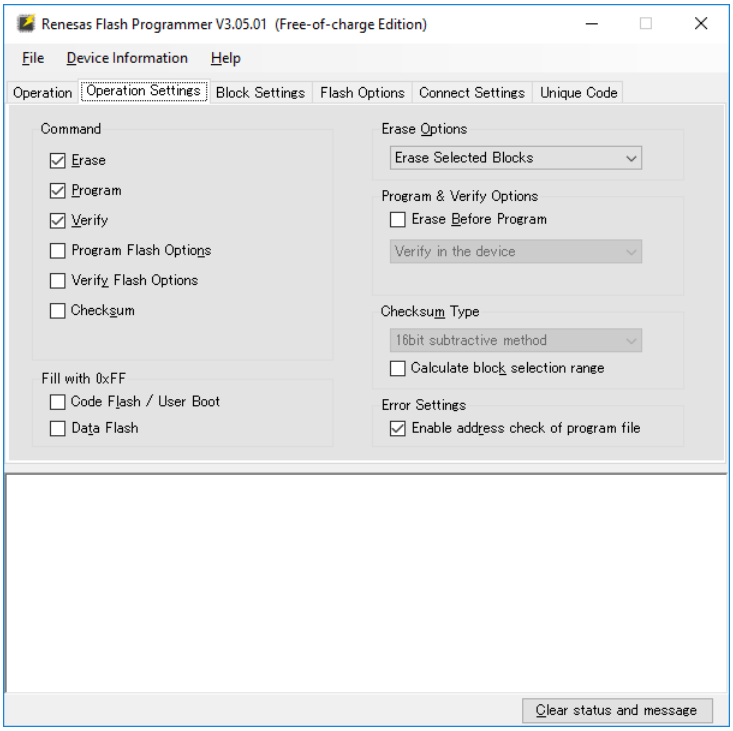
Regarding the system configuration, refer to section 4.3 "System Configuration" in this document.

Regarding to the Unique Code Embedding Function, refer to subsection 2.3.6 "[Unique Code] Tabbed Page" in Renesas Flash Programmer V3.05 Flash memory programming software User's Manual (R20UT4307).

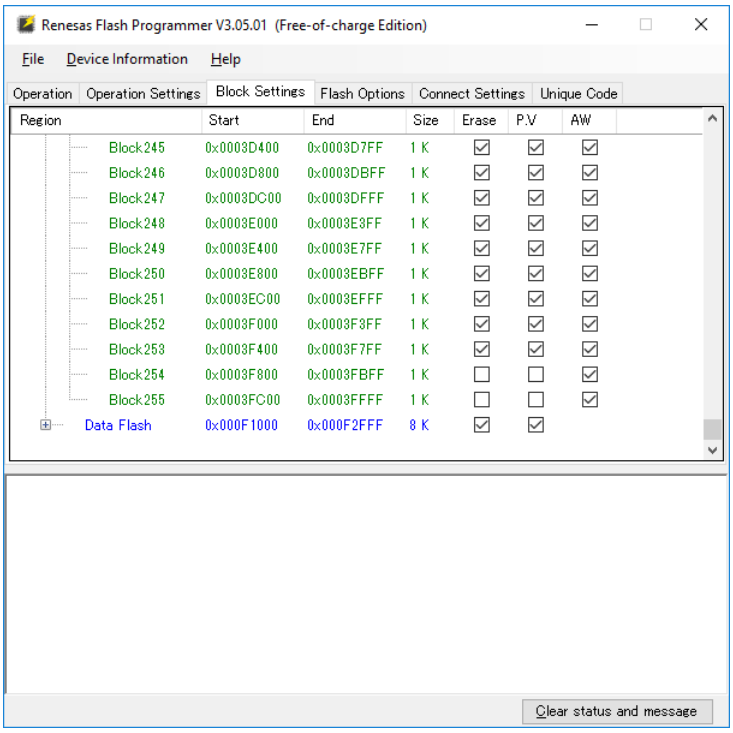
5. Prevent erasing Block 254, 255 in Code Flash memory according to the following steps.

   In RL78/G1D Module, Shipping Check Flag is written in Block 254 and Device Address is written in Block 255 respectively.

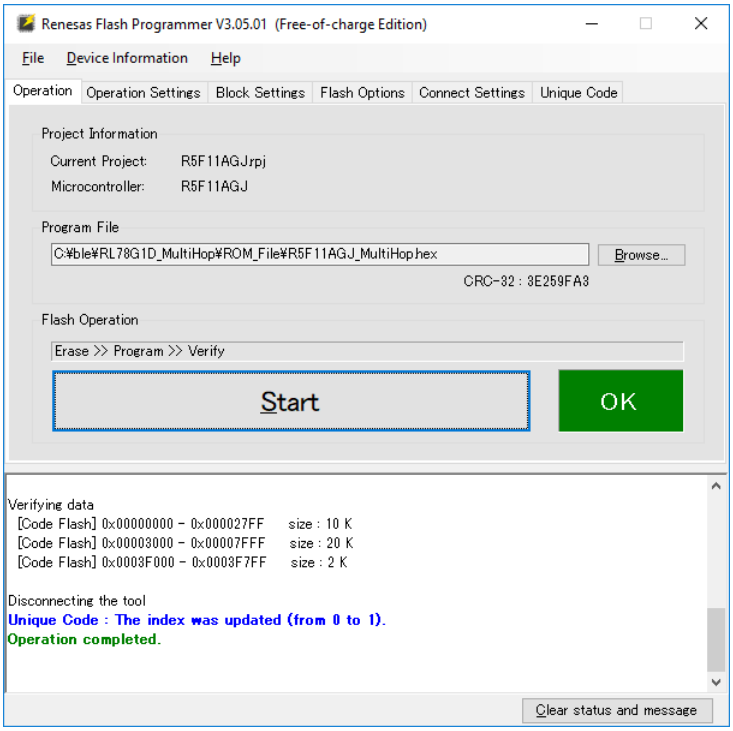
   5-1. Select [Operation Setting] tab and select [Erase Selected Blocks] at [Erase Option].




   5-2. Select [Block Setting] tab and uncheck each [Erase], [P.V] of Block254, 255.



6. Select [Operation] table and specify the following firmware at [Program File].
   - (R01AN4375) ROM_File\R5F11AGJ_MultiHop.hex
   - (R01AN4466) ROM_File\R5F11AGJ_MultiHopSEC.hex
7. Click [Start] button to start writing the firmware.
8. Confirm [Operation completed] message.




9. Disconnect E1 Emulator and Power Supply from the evaluation board.
10. Connect E1 Emulator and Power Supply to next evaluation board and execute step 7 again.
    Repeat above steps until you write firmware to all evaluation boards.

## 5.4 Operation

Figure 5-3 shows the operation of the sample program. The evaluation boards of other than ID0 transmit and relay frames successively. And the evaluation board of ID0 receives frames as a destination.

By the way, a device filter is implemented in the sample program. This filter allows the sample program to receive only a frame transmitted from specified device. By using a device filter, you can confirm relaying frame within a restricted path; e.g. ID3→ID2→ID1→ID0. Regarding the details, refer to section 9.2 "Device Filter".

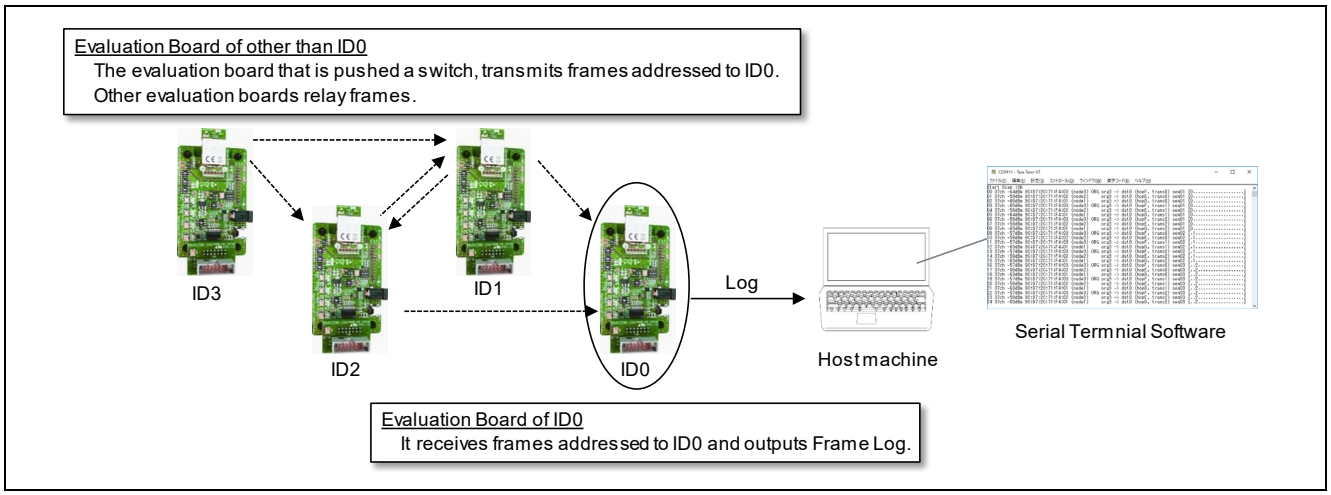



**Figure 5-3 Operation of Sample Program**

How to execute frame transmission and reception is shown below:

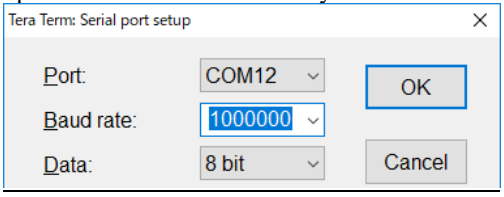
Evaluation Board of ID0:

1. Connect the evaluation board to Host machine with USB cable.
2. Start terminal software (e.g. Tera Term) on Host machine. Then set serial communication setting in accordance with Table 5-2.

**Table 5-2 Serial Communication Setting**

| Item | | Setting |
|---|---|---|
| Serial Port | Port | USB Serial Port<br>Note that COM number is different from each evaluation board. |
| | Baud Rate | 1,000,000bps |
| | Data Bit Length | 8bit |
| | Parity | None |
| | Stop Bit Length | 1bit |
| | Flow Control | None |
| New Line | Receive | LF |
| | Transmit | LF |
| Terminal Size | Horizontal | over than 128 characters |

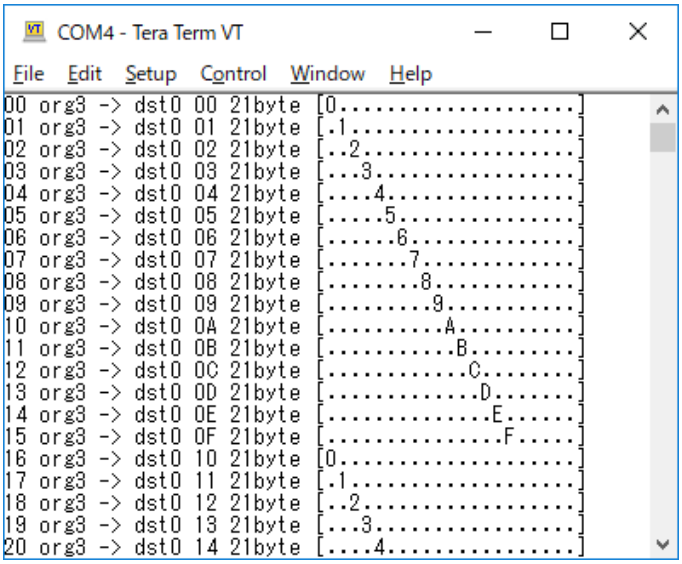
When Tera Term is used as terminal software, there is no "1,000,000bps" in the drop-down list of Baud Rate. Thus, it is necessary to enter "1000000" to the input box of Baud Rate directly.



Evaluation Boards of other than ID0:

1. Start to supply power the evaluation board via either DC jack or USB interface selected by slide-switch.
2. Push a switch SW2 on an evaluation board of ID3.

   The evaluation board pushed a switch transmits a frame addressed to ID0 every second.
3. When the evaluation board of ID0 receives frames, you can confirm that the following frame data log is displayed on a terminal software.




If you use the board that switch SW is not implemented, write the following firmware which starts frame transmission automatically to the boards of other than ID0.

- ROM_File\R5F11AGJ_MultiHop(NO_SW).hex

To start frame transmission automatically, change the macro *TEST_NO_SW* to (1) defined in the following file.

- Project_Source\application\src\r_node.c

```
54:     /* 1: start to sends frame immediately after application initialization */
55:     /* 0: start to sends frame when switch is pushed */
56:     #define TEST_NO_SW              (0)
                                        change to (1)
```

Scan program which scans Multi-Hop frame is included in the package of the sample program.

The firmware of the scan program is the following file.

- ROM_File\R5F11AGJ_Scan.hex

Figure 5-4 shows the overview of the scan program.

Write the scan program firmware to the evaluation board, and then connect it to the Host machine.

To check the log output by the scan program, use the terminal software as well as how to evaluate the node of ID0.

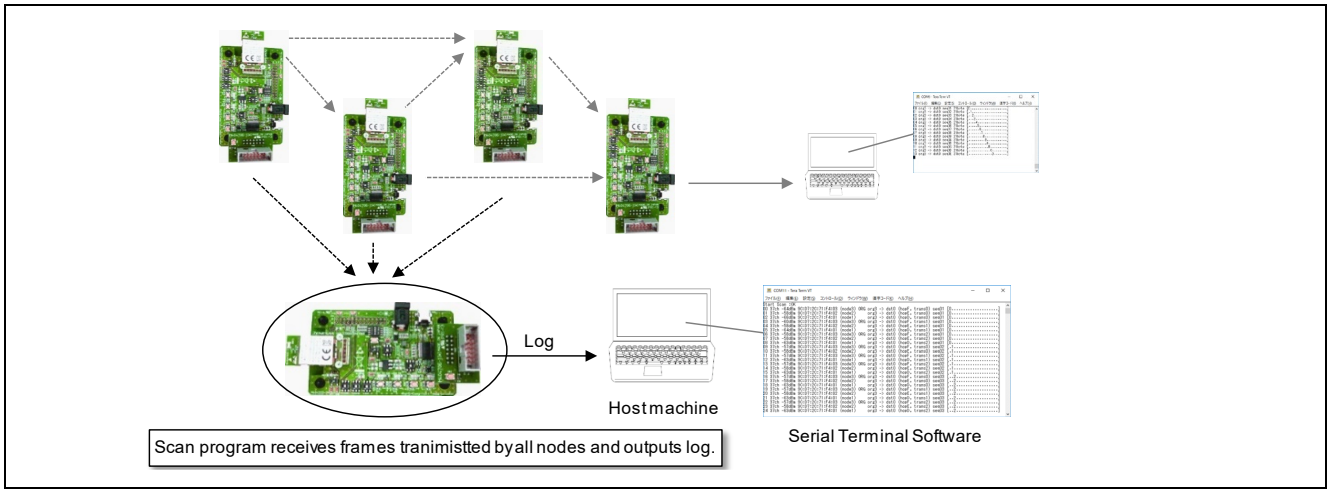



**Figure 5-4 Operation of Scan Program**

The scan program outputs the log of received frame via UART with the following format.

*SerialNumber Channel RSSI DeviceAddress DeviceAddressType* (*NodeID*) *ORG* -> *DST* (*HopLimit*, *TransmissionCount*) *SEQ FrameDataSize* [*FrameData (ASCII)*]

Figure 5-5 shows example log output by the scan program.

COM12 - Tera Term VT

File Edit Setup Control Window Help

```
Start Scan :OK
00 39ch -43dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,0) 00 [0.....................]
01 37ch -41dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,1) 00 [0.....................]
02 37ch -33dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,1) 00 [0.....................]
03 37ch -43dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,1) 00 [0.....................]
04 38ch -42dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,2) 00 [0.....................]
05 38ch -43dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,2) 00 [0.....................]
06 38ch -34dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,2) 00 [0.....................]
07 39ch -39dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,0) 01 [.1....................]
08 39ch -35dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,0) 01 [.1....................]
09 37ch -41dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,1) 01 [.1....................]
10 37ch -45dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,1) 01 [.1....................]
11 38ch -33dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,1) 01 [.1....................]
12 38ch -40dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,2) 01 [.1....................]
13 39ch -35dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,2) 01 [.1....................]
14 39ch -45dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,2) 01 [.1....................]
15 39ch -39dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,0) 02 [..2...................]
16 37ch -33dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,0) 02 [..2...................]
17 38ch -42dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,1) 02 [..2...................]
18 39ch -39dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,2) 02 [..2...................]
19 39ch -48dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,2) 02 [..2...................]
20 39ch -34dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,2) 02 [..2...................]
21 37ch -40dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,0) 03 [...3..................]
22 37ch -43dBm DC:D7:2C:71:F4:02 org03 -> dst00 (hopE,0) 03 [...3..................]
23 38ch -41dBm DC:D7:2C:71:F4:03 org03 -> dst00 (hopF,1) 03 [...3..................]
24 38ch -34dBm DC:D7:2C:71:F4:01 org03 -> dst00 (hopE,1) 03 [...3..................]
```

**Figure 5-5 Example Log of Received Frame**

# 6. Building Procedure

This chapter explains the building procedure of the sample program.

## 6.1 Developing Environment

To develop and build a firmware of the sample program, you can use either CS+ for CC or e$^2$ studio.

CS+
https://www.renesas.com/software-tool/cs
e$^2$ studio
https://www.renesas.com/software-tool/e-studio

The necessary hardware and software environment for compiling is as follow:

- Hardware Environment
  - Host
    - PC/AT™ compatible computer
  - Device
    - RL78/G1D Evaluation Board (RTK0EN0001D01001BZ): at least 2 boards
    - USB cable (A type make/mini-B type make): 2 cables
  - Tool
    - Renesas On-chip Debugging Emulator E1 (R0E000010KCE00)

- Software Environment
  - Windows® 10
  - **Renesas CS+ for CC V7.00.00 / Renesas CC-RL V1.07.00**

    or

    **Renesas e$^2$ studio Version 7.0.0 / Renesas CC-RL V1.07.00**
  - Tera Term Pro (or Terminal software which can connect to serial port)
  - UART-USB conversion device driver

Note: It may be that device driver for UART-USB conversion IC *FT232RL* is requested when you connect RL78/G1D Evaluation Board to Host first time. In this case, you can get the device driver from below website.

- FTDI (Future Technology Devices International) - Drivers

  http://www.ftdichip.com/Drivers/D2XX.htm

The following libraries are included in the package. (R01AN4466 only). These libraries are required to build the sample program corresponding to Security feature.

AES Library: AES Library for the RL78 Family V1.05 Release 00

Data Flash Library: EEPROM Emulation Library Pack02 for the CC-RL Compiler for the RL78 Family

## 6.2 File Composition

File and folder composition of the sample program is shown below.

```
RL78G1D_MultiHop
  ├ROM_File
  │    R5F11AGJ_MultiHop.hex                          Sample Program Firmware (R01AN4375 only)
  │    R5F11AGJ_MultiHop(DEV_FILTER).hex                (DEV_ADDR_FILTER=1)
  │    R5F11AGJ_MultiHop(NO_SW).hex                     (TEST_NO_SW=1)
  │    R5F11AGJ_MultiHopSEC.hex                        Sample Program Firmware (R01AN4466 only)
  │    R5F11AGJ_MultiHop(DEV_FILTER).hex                (DEV_ADDR_FILTER=1)
  │    R5F11AGJ_MultiHop(NO_SW).hex                     (TEST_NO_SW=1)
  │    R5F11AGJ_Scan.hex                               Scan Program Firmware
  ├RUC_File
  │    r5f11agj_syscfg.ruc                            System Configuration
  │    r5f11agj_syscfg_sec.ruc                        System Configuration (R01AN4466 only)
  └Project_Source
    ├library
    │    r_arch.h
    │    r_compiler.h
    │    r_iodefine.h
    │    r_ll.h                                       Beacon Stack Library
    │    r_port.h
    │    r_bcn_api.h
    │    BLE_BEACON_CC.lib
    └application
      ├src
      │  │  cstart.asm                                Start-up Routine
      │  │  r_config.h                                Beacon Stack Configuration
      │  │  r_interrupt.c                             Beacon Stack Interrupt Handler
      │  │  r_main.c                                  Application Entry Point
      │  │  r_multihop.c                              Multi-Hop Layer
      │  │  r_multihop.h
      │  │  r_node.c                                  Node Application
      │  │  r_node.h
      │  │  r_utility.c                               Utility
      │  │  r_utility.h
      │  ├aes
      │  │  │  P_AesProto.h                           AES-CCM Conversion
      │  │  │  P_Ccmz.c                               (R01AN4466 only)
      │  │  └library
      │  │       r_aes.h
      │  │       r_mw_version.h                       AES Library
      │  │       r_stdint.h                           (R01AN4466 only)
      │  │       aes_rl78_s2_m_ccrl.lib
      │  └driver
      │    ├dataflash
      │    │  │  r_dataflash.c                        Data Flash Driver
      │    │  │  r_dataflash.h                        (R01AN4466 only)
      │    │  │  r_eel_descriptor_t02.c               EEPROM Emulation Library Descriptor
      │    │  │  r_eel_descriptor_t02.h               (R01AN4466 only)
      │    │  │  r_fdl_descriptor_t02.c               Data Flash Library Descriptor
      │    │  │  r_fdl_descriptor_t02.h               (R01AN4466 only)
      │    │  └library
      │    │       eel.h
      │    │       eel.lib                            EEPROM Emulation Library
      │    │       eel_types.h                        (R01AN4466 only)
      │    │       fdl.h
      │    │       fdl.lib                            Data Flash Library
      │    │       fdl_types.h                        (R01AN4466 only)
      │    ├input
      │    │  r_input.c
```

```
│       │  r_input.h                    External Input Interrupt Driver
│       ├─it
│       │  r_it.c                       12-bit Interval Timer Driver
│       │  r_it.h
│       ├─led
│       │  r_led.c                      LED Driver
│       │  r_led.h
│       ├─plf
│       │  r_plf.c                      Platform Driver
│       │  r_plf.h
│       ├─timer
│       │  r_timer.c                    Timer Array Unit Driver
│       │  r_timer.h
│       │  r_timerx.c                   Index Timer Feature
│       └─uart
│         r_log.c                       Log Output Feature
│         r_uart.c                      UART Driver
│         r_uart.h
└─project
  └─cs_cc
    └─BLE_Software
      │  BLE_Software.mtpj               CS+ for CC Project File
      ├─R5F11AGJ_MultiHop
      │     R5F11AGJ_MultiHop.mtsp       CS+ for CC Sub-Project File (R01AN4375 only)
      └─R5F11AGJ_MultiHopSEC
          R5F11AGJ_MultiHopSEC.mtsp      CS+ for CC Sub-Project File (R01AN4466 only)
```

Note that Beacon Stack Library included in this project is different from the one included in the application note of "RL78/G1D Beacon Stack Basic Operation" (R01AN3045) and is optimized for Multi-Hop Feature. When you develop an application, use the library included in this project.

## 6.3 Building Firmware

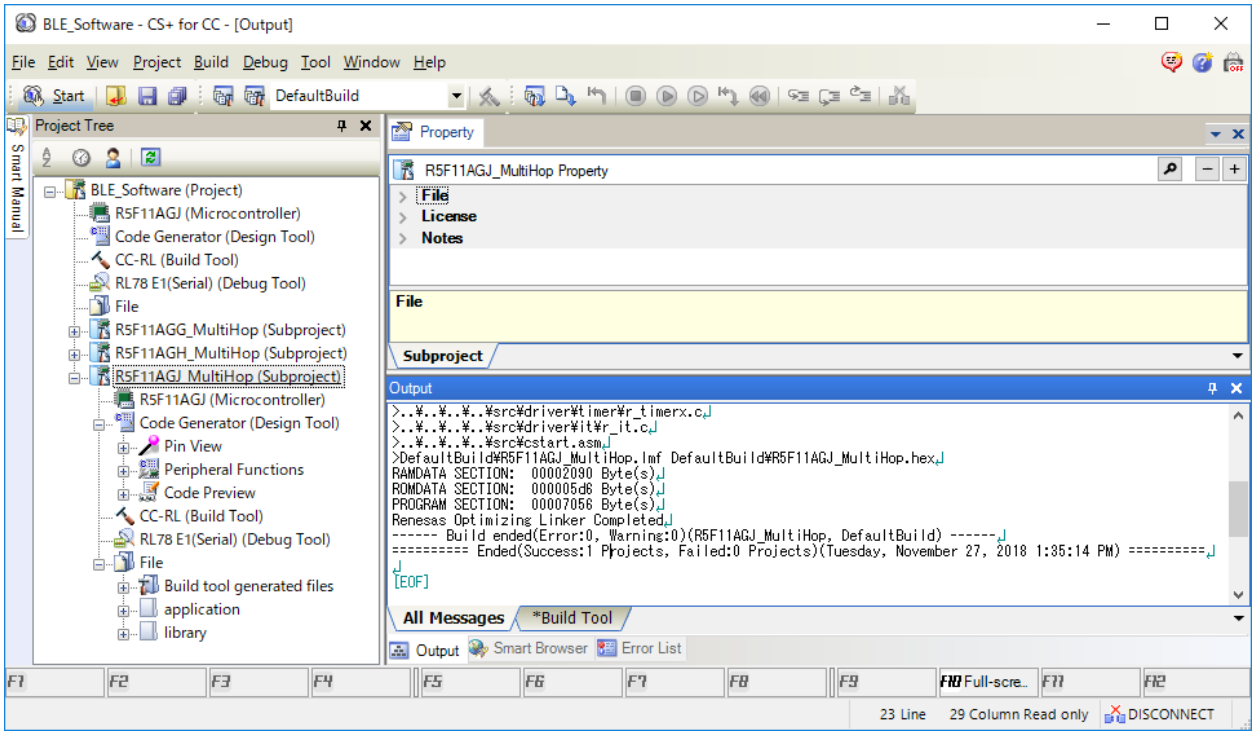
You can use either CS+ for CC or e2 studio as an Integrated Development Environment for building a firmware of the sample program.

Using CS+ for CC:

1. Start CS+ for CC and open the project named *BLE_Software.mtpj* from below folder by selecting *Open* from *File* menu bar: [File]→[Open…].
   - Project_Source\application\project\cs_cc\BLE_Software\
2. (R01AN4375) Right-click *R5F11AGJ_Multihop(Subproject)* in Project Tree window, then select *Rebuild R5F11AGJ_Multihop* in the right-click menu to build the firmware.

   (R01AN4466) Right-click *R5F11AGJ_MultihopSEC(Subproject)* in Project Tree window, then select *Rebuild R5F11AGJ_MultihopSEC* in the right-click menu to build the firmware.
3. Confirm that no error occurs, and it succeeds in building a firmware.
4. (R01AN4375) Confirm that the firmware *R5F11AGJ_MultiHop.hex* is generated in the following folder.
   - Project_Source\application\project\cs_cc\BLE_Software\R5G11AGJ_MultiHop\DefaultBuild\

   (R01AN4466) Confirm that the firmware *R5F11AGJ_MultiHopSEC.hex* is generated in the following folder.
   - Project_Source\application\project\cs_cc\BLE_Software\R5G11AGJ_MultiHopSEC\DefaultBuild\



Using e$^2$ studio:

1. Start Renesas e$^2$ studio and select below path as a workspace.
   - Project_Source\
2. Select Import from File menu bar: [File]→[Import] to open Import dialog.
3. Select Existing Project into Workspace from General: [General]→[Existing Project into Workspace] and click [Next] button.
4. (R01AN4375) Select below path as a root folder and select R5F11AGJ_MultiHop in [Projects].

   (R01AN4466) Select below path as a root folder and select R5F11AGJ_MultiHopSEC in [Projects].
   - Project_Source\
5. Click [Finish] button to close Import dialog.
6. Close [Welcome].
7. (R01AN4375) Select R5F11AGJ_MultiHop in the Project Explorer.

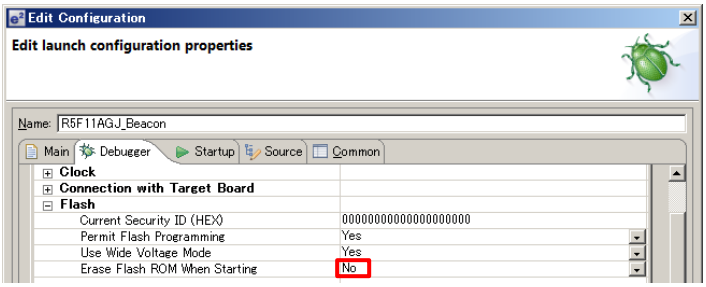
   (R01AN4466) Select R5F11AGJ_MultiHopSEC in the Project Explorer.
8. Select Build Project from Project menu: [Project]→[Build Project] and confirm that successful compilation.
9. (R01AN4375) Confirm that the firmware R5F11AGJ_MultiHop.hex is generated in the below path.
   - Project_Source\application\project\e2_cc\BLE_Software\R5F11AGJ_MultiHop\DefaultBuild\

   (R01AN4466) Confirm that the firmware R5F11AGJ_MultiHopSEC.hex is generated in the below path.
   - Project_Source\application\project\e2_cc\BLE_Software\R5F11AGJ_MultiHopSEC\DefaultBuild\

Note: Default debugger setting of e$^2$ studio erases flash memory before writing firmware.

In the case of developing by using e$^2$ studio, change the debugger setting before starting debugging, to avoid erasing Shipping Checking Flag and Device Address written in RL78/G1D Module. When changing the debugger setting, disconnect the E1 Emulator from RL78/G1D Module at first.

- Select [Debugger] tab in [Edit launch configuration properties] dialog, and set [No] in [Erase Flash ROM When Starting].



## 6.4 Company ID

Figure 6-1 shows Company ID field in Multi-Hop frame.

Multi-Hop frame is transferred as a Manufacturer Specific Data in Advertising packet of Bluetooth Low Energy. Manufacturer Specific Data need to be set Bluetooth Company ID.

When you develop a product, set your company ID to the sample program.

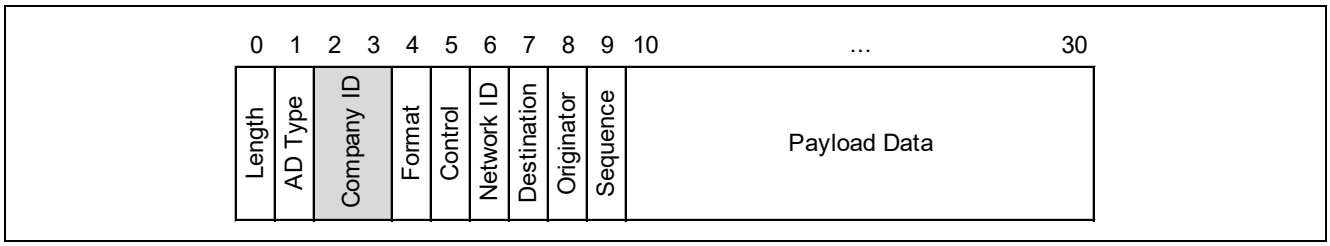



**Figure 6-1 Company ID Field of Multi-Hop Frame**

To set a Bluetooth Company ID, change the macro COMPANY_ID defined in the following file.

- Project_Source\application\src\r_node.c

```
50:     /* Bluetooth Company ID */
51:     /* https://www.bluetooth.com/specifications/assigned-numbers/company-identifie
52:     #define COMPANY_ID              (0xFFFF)
                                        change to Bluetooth Company ID
```

You can confirm Bluetooth Company ID by the following web site. To assign a company ID, an application to Bluetooth SIG is required.

https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers

## 7. Hardware Resource Used

Hardware resources, which used by the sample program with default settings, are shown below.

| | | |
|---|---|---|
| MCU Unit | | |
| | Clock Generator | Only the following clock frequency of High-speed on-chip oscillator: $f_{IH}$ can be set as a MCU Main System Clock: $f_{MAIN}$.<br>• 4MHz<br>• 8MHz: default setting<br>• 16MHz<br>• 32MHz<br>Only High-speed on-chip oscillator: $f_{IH}$ can be used as a MCU Main System Clock: $f_{MAIN}$.<br>External Main System Clock: $f_{EX}$ cannot be used. |
| | | Whether to use XT1 oscillation clock: $f_{XT}$ is used or not is selectable.<br>• use XT1oscillation: default setting<br>• not use XT1 oscillation (use RF on-chip oscillator)<br>If XT1 oscillation is used, clock should be input from MCU to RF; clock is output from PCLBUZ0 pin and is input to EXSLK_RF pin.<br>If XT1 oscillation is used, connecting to a 32.768kHz resonator is required. |
| | Clock output/buzzer output | Output clock from PCLBUZ0 pin is selectable as a RF slow clock.<br>• use 16.384kHz clock from PCLBUZ0<br>• use 32.768kHz clock from PCLBUZ0<br>• not use clock from PCLBUZ0: default setting<br>If not use clock from PCLBUZ0, RF on-chip oscillation clock is required. |
| | Timer Array Unit | Operation Clock CK00 is set to 1MHz<br>Operation Clock CK01 is set to 250kHz<br>Beacon Stack uses TM00 and its operation clock is CK00<br>Multi-Hop Layer uses TM01 and TM02 and its operation clock is CK01 |
| | Serial array unit | use CSI21 |
| | DMA controller | use DMA2 and DMA3 |
| | Interrupt | use INTRF<br>use INTDMA2<br>use INTDMA3<br>use INTTM00 |
| RL78/G1D RF Unit | | |
| | DC-DC Converter | Whether to use RF DC-DC converter or not is selectable.<br>• use RF on-chip DC-DC converter: default setting<br>• not use RF on-chip DC-DC converter<br>If DC-DC converter is used, external inductor/capacitor is required. |
| | Oscillator for RF slow clock | Whether to use RF on-chip oscillator or not is selectable.<br>• use RF on-chip oscillator: default setting<br>• not use RF on-chip oscillator<br>If not use RF on-chip oscillator, input clock to EXSLK_RF pin is required. |
| | Clock Output | Divided RF Base Clock can be output to CLKOUT_RF pin.<br>• not output clock: default setting<br>• output 16MHz clock<br>• output 8MHz clock<br>• output 4MHz clock<br>If not output clock is selected, CLKOUT_RF pin is input mode. |

# 8. Multi-Hop API

This chapter explains the specification of Multi-Hop API provided by Multi-Hop layer.

## 8.1 Type

| Type Name | Standard Type | Description |
|---|---|---|
| uint8_t | unsigned char | un-signed 8bit integer |
| uint16_t | unsigned short | un-signed 16bit integer |
| uint32_t | unsigned long | un-signed 32bit integer |
| int8_t | signed char | signed 8bit integer |
| int16_t | signed short | signed 16bit integer |
| int32_t | signed long | signed 32bit integer |
| bool | unsigned char | boolean; either true or false |
| int_t | signed int | signed integer |
| uint_t | unsigned int | un-signed integer |
| char_t | char | character |
| RBLE_STATUS | unsigned char | return value of Multi-Hop function |

## 8.2 Macro

### 8.2.1 Status Macro

| Macro Name | Value | Description |
|---|---|---|
| RBLE_OK | 0x00 | Success |
| RBLE_ERR_PARAM | 0x01 | Error: invalid parameter |
| RBLE_ERR_WL | 0x02 | Error: White List is empty |
| RBLE_ERR_PWRDOWN | 0x03 | Error: supplying power to RF unit is stopped |
| RBLE_ERR_PWRUP | 0x04 | Error: supplying power to RF unit is already started |
| RBLE_ERR_RFRX | 0x05 | Error: RF unit Rx is disabled |
| RBLE_ERR_START | 0x06 | Error: RF operation is already started |
| RBLE_ERR_STOP | 0x07 | Error: RF operation is stopped |
| RBLE_ERR_HW_STANDBY | 0x08 | Error: RF unit abnormality in STANDBY_RF |
| RBLE_ERR_HW_STANDBYRX | 0x09 | Error: RF unit abnormality in STANDBY_RF and enabling Rx |
| RBLE_ERR_HW_IDLE | 0x0A | Error: RF unit abnormality in IDLE_RF |

### 8.2.2 Device Address Type Macro

| Macro Name | Value | Description |
|---|---|---|
| RBLE_ADDR_PUBLIC | 0x00 | Public Device Address |
| RBLE_ADDR_RANDOM | 0x01 | Random Device Address |

### 8.2.3 Event Macro

| Macro Name | Value | Description |
|---|---|---|
| RMH_EVT_RECEIVE_IND | 0x01 | Frame Reception Indication |
| RMH_EVT_OPTION_IND | 0x02 | Frame Reception Indication (Option Data) |
| RMH_EVT_STOP_CMP | 0x03 | Reception Stop Completion |
| RMH_EVT_SEND_CMP | 0x04 | Transmission Completion |
| RMH_EVT_ENCCNT_WRN | 0x05 | Near an End of Encryption Counter Warning |
| RMH_EVT_HOP_WRN | 0x06 | Frame Discard by Relaying Buffer Full Warning |
| RMH_EVT_DUP_WRN | 0x07 | Frame Discard by Duplicate Check Buffer Full Warning |

## 8.3 Structure

### 8.3.1 Device Address Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RBLE_BD_ADDR | | | |
| addr | uint8_t[6] | 0 | Device Address |

### 8.3.2 Multi-Hop Configuration Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_CFG | | | |
| own_addr | RBLE_BD_ADDR | 0 | Device Address |
| own_addr_type | uint8_t | 6 | Device Address Type |
| reserved | uint8_t | 7 | (reserved) |
| company | uint16_t | 8 | Company ID |
| network | uint8_t | 10 | Network ID |
| node | uint8_t | 11 | Node ID |
| callback | void (*)(RMH_EVT*) | 12 | Multi-Hop Callback Function |

### 8.3.3 Security Configuration Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_SEC | | | |
| enable | bool | 0 | enable or disable Security |
| reserved | uint8_t | 1 | (reserved) |
| key | uint8_t[16] | 2 | Encryption Key |
| counter | uint32_t | 18 | Initial Nonce Counter |

### 8.3.4 Frame Data Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_DATA | | | |
| data | uint8_t[21] | 0 | Frame Data |
| len | uint8_t | 21 | Frame Data Length |
| node | uint8_t | 22 | Node ID |
| reserved | uint8_t | 23 | (reserved) |

### 8.3.5 Option Data Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_OPTION | | | |
| id | uint8_t | 0 | Option Data ID |
| len | uint8_t | 1 | Option Data Length |
| union data | | | |
| buf | uint8_t[19] | 2 | Option Data |
| rootcheck | ROOTCHECK | 2 | Relayed Path Log |

### 8.3.6 Relayed Path Log Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct ROOTCHECK | | | |
| counter | uint16_t | 0 | Transmission Counter |
| rootlog | uint8_t[15] | 2 | Relayed Path Log |
| tail | uint8_t | 17 | Tail of Relayed Path Log |

### 8.3.7 Multi-Hop Event Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_EVT | | | |
| type | uint8_t | 0 | Event Type |
| reserved | uint8_t | 1 | (reserved) |
| union param | | | |
| send | RMH_SEND | 2 | Transmission Completion Parameter |
| receive | RMH_RECEIVE | 2 | Reception Indication Parameter |
| discard | RMH_DISCARD | 2 | Frame Discard Warning |

### 8.3.8 Frame Transmission Indication Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_SEND | | | |
| union num | | | |
| seq | uint8_t | 0 | Sequence Number when enc is false |
| counter | uint32_t | 0 | Nonce Counter when enc is true |
| enc | bool | 4 | Encrypted Frame Flag |

### 8.3.9 Frame Reception Indication Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_RECEIVE | | | |
| data | uint8_t[21] | 0 | Frame Data |
| len | uint8_t | 21 | Frame Data Length |
| dst | uint8_t | 22 | Destination Node ID |
| org | uint8_t | 23 | Originator Node ID |
| union num | | | |
| seq | uint8_t | 24 | Sequence Number when enc is false |
| counter | uint32_t | 24 | Nonce Counter when enc is true |
| enc | bool | 28 | Encrypted Frame Flag |

### 8.3.10 Frame Discard Warning Structure

| Member Name | Type | Offset | Description |
|---|---|---|---|
| struct RMH_DISCARD | | | |
| dst | uint8_t | 0 | Destination Node ID |
| org | uint8_t | 1 | Originator Node ID |
| union num | | | |
| seq | uint8_t | 2 | Sequence Number when enc is false |
| counter | uint32_t | 2 | Nonce Counter when enc is true |
| enc | bool | 6 | Encrypted Frame Flag |

## 8.4 Function

Table 8-1 shows the functions provided by Multi-Hop layer.

**Table 8-1 Multi-Hop API Functions**

| Function | Operation |
|---|---|
| R_MH_Init() | Initializes Multi-Hop Feature. |
| R_MH_Proc() | Executes Multi-Hop Processing. |
| R_MH_Security() | Enables Security feature. (R01AN4466 only) |
| R_MH_Receive() | Starts Receiving Multi-Hop Frame. |
| R_MH_Stop() | Stops Receiving Multi-Hop Frame. |
| R_MH_Send() | Sends Multi-Hop Frame. |
| R_MH_CheckRoot() | Sends Multi-Hop Relayed Path Frame to Check Relayed Path. |

### 8.4.1 R_MH_Init

void R_MH_Init( RMH_CFG* cfg );

This function initializes Multi-Hop Feature.
To transmit and receive Multi-Hop Frames, it is necessary to execute this function once on boot.

Regarding the implementation of callback function specified by the argument *cfg->callback*, refer to Usage.

Parameters:

| | | |
|---|---|---|
| *cfg | own_addr | Device Address of own device |
| | own_addr_type | Device Address Type of own device<br>Regarding the setting, refer to subsection 8.2.2 "Device Address Type Macro". |
| | company | Company ID of own node<br>https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers |
| | network | Network ID of own node<br>0x00 to 0xFF |
| | node | Node ID of own node<br>0x00 to 0xFE |
| | callback | Callback Function to notify Multi-Hop Events<br>typedef void (*RMH_CALLBACK)(RMH_EVT* evt); |

Return:

None

Usage:

An example of callback function implementation is shown below.

```
static void node_callback(RMH_EVT* evt)
{
    switch (evt->type)
    {
        case RMH_EVT_SEND_CMP:
            /* reach here after transmitting multi-hop frame */
            break;

        case RMH_EVT_RECEIVE_IND:
            /* reach here when receiving multi-hop frame */
            break;

        case RMH_EVT_STOP_CMP:
            /* reach here after stopping frame reception */
            break;

        default:
            break;
    }
}
```

### 8.4.2 R_MH_Proc

void R_MH_Proc( void );

This function executes Multi-Hop processing.

This function takes events out from Beacon Stack and executes below processing.

- After frame transmission, this function notifies that transmission is completed to application.
- After stopping reception processing, this function notifies that reception is stopped to application.
- Upon receiving a frame addressed to own node, this function notifies that frame is received to application.
- Upon receiving a frame addressed to other node, this function transmits the frame to relay.

When there is no event of Beacon Stack to be handled, this function returns.

This function needs to be executed repeatedly.
Regarding the implementation, refer to Usage.

Parameters:

None

Return:

None

Usage:

This function needs to be executed when Beacon Stack event occurs.
An example of implementation in main loop is shown below.

```
while (1)
{
    /* execute Hulti-Hop processing */
    R_MH_Proc();
}
```

MCU current consumption can be reduced by executing HALT or STOP instruction in main loop.
Note that some MCU peripherals are stopped by entering STOP mode. So, it is necessary to use either HALT or STOP instruction properly depends on whether MCU peripheral is operated or not.
An example of HALT and STOP instruction implementation is shown below.

```
while (1)
{
    /* execute Multi-Hop processing */
    R_MH_Proc();

    __disable_interrupt();
    if(R_TIMER_IsActive())
    {
        /* HALT mode ends by un-masked interruption of either rf or other peripherals */
        __halt();
    }
    else
    {
        /* STOP mode ends by un-masked interruption of either rf or other peripherals */
        __stop();
    }
    __enable_interrupt();
}
```

### 8.4.3 R_MH_Security

| void R_MH_Security( RMH_SEC* sec ); | | | |
|---|---|---|---|
| This function enables Security feature and sets both an encryption key and a nonce counter.<br>If application does not use Security feature, it is not necessary to use this function.<br>Note that this function is implemented in R01AN4466 only.<br><br>Encryption key is used for transmitting, relaying and receiving encrypted frame.<br>So, it is necessary to set the same encryption key to all node in the network.<br><br>After enabling Security feature, Multi-Hop layer transmits encryption frame when application executes R_MH_Send().<br>And Multi-Hop layer receives and relays encryption frames when application executes R_MH_Receive(). | | | |
| Parameters: | | | |
| | sec | enable | Boolean whether to enable Security feature or not<br>true：enable<br>false：disable |
| | | key | 128bit Common Encryption Key<br>Encryption Key should be shared by all nodes having the same Network ID. |
| | | counter | Initial Value of 32bit Nonce Counter<br>This value is used as a Nonce to randomize encrypted data. |
| Return: | | | |
| | RBLE_OK | | Success |
| | RBLE_ERR_START | | Error: Multi-Hop processing is working. |
| | RBLE_ERR_PARAM | | Error: invalid parameter |

### 8.4.4 R_MH_Receive

RBLE_STATUS R_MH_Receive( bool hop_en );

This function starts Multi-Hop frame reception processing.
When application sets the argument hop_en to true, Multi-Hop layer relays frames addressed to other nodes.

When Multi-Hop layer receives the frame addressed to either own node or all nodes, RMH_EVT_RECEIVE_IND event is notified by the callback function.
When Multi-Hop layer relays the frame addressed to other nodes, no event is notified.

To stop reception processing, execute R_MH_Stop().
Even if reception processing works, application can send frame by executing R_MH_Send().

Parameters:

| | |
|---|---|
| hop_en | Boolean whether to enable relaying frames<br>true: enable<br>false: disable |

Return:

| | |
|---|---|
| RBLE_OK | Success |
| RBLE_ERR_START | Error: Multi-Hop reception processing is working. |
| RBLE_ERR_PARAM | Error: invalid parameter |

### 8.4.5 R_MH_Stop

RBLE_STATUS R_MH_Stop( void );

This function stops reception processing started by R_MH_Receive().

After stopping reception processing, RMH_EVT_STOP_CMP event is notified by the callback function.

Parameters:

None

Return:

| | |
|---|---|
| RBLE_OK | Success |
| RBLE_ERR_STOP | Error: Multi-Hop reception processing is not working. |

### 8.4.6 R_MH_Send

RBLE_STATUS R_MH_Send( RMH_DATA* data );

This function sends Multi-Hop frame.

After the frame transmission, RMH_EVT_SEND_CMP event is notified by the callback function.
If application sends some frames, execute this function after the RMH_EVT_SEND_CMP event for each frame.

Parameters:

| | | |
|---|---|---|
| *data | data | Multi-Hop Frame Data |
| | len | Multi-Hop Frame Data Length (byte)<br>un-encrypted frame: Max.21byte<br>encrypted frame: Max.14byte (R01AN4466 only) |
| | node | Destination Node ID of Multi-Hop Frame<br>0x00 to 0xFE individual node<br>0xFF all nodes |

Return:

| | |
|---|---|
| RBLE_OK | Success |
| RBLE_ERR_START | Error: Multi-Hop transmission processing is working. |
| RBLE_ERR_PARAM | Error: invalid parameter |

### 8.4.7 R_MH_CheckRoot

RBLE_STATUS R_MH_CheckRoot(uint8_t dst, uint16_t counter);

This function sends Relayed Path Frame to check relayed path.
Note that this function can be used when Path Check feature is enabled. Regarding the Path Check feature, refer to section 9.1 "Path Check Feature".

After the Relayed Path Frame transmission, RMH_EVT_SEND_CMP event is notified by the callback function.
When Multi-Hop layer receives the Relayed Path Frame, RMH_EVT_OPTION_IND event is notified by the callback function and Relayed Path Log Structure (refer to subsection 8.3.6) is set as a Frame Data in the event parameter.

Parameters:

| | | |
|---|---|---|
| dst | Destination Node ID of Relayed Path Frame | |
| | 0x00 to 0xFE | individual node |
| | 0xFF | all nodes |
| counter | counter value to identify each frame | |

Return:

| | |
|---|---|
| RBLE_OK | Success |
| RBLE_ERR_START | Error: Multi-Hop transmission processing is working. |
| RBLE_ERR_PARAM | Error: invalid parameter |

## 8.5 Event

Table 8-2 shows the events provided by Multi-Hop layer.

Multi-Hop layer notifies the following events by calling callback function registered by R_MH_Init().

Callback function to notify events is executed by R_MH_Proc().

**Table 8-2 Multi-Hop API Events**

| Event | Detail |
|---|---|
| RMH_EVT_RECEIVE_IND | Notifies that frame addressed to own node was received. |
| RMH_EVT_OPTION_IND | Notifies that frame having Option Data and addressed to own node was received. |
| RMH_EVT_STOP_CMP | Notifies that reception processing was stopped completely. |
| RMH_EVT_SEND_CMP | Notifies that frame transmission was finished completely. |
| RMH_EVT_ENCCNT_WRN | Warns that near end of Encryption Nonce Counter (R01AN4466 only) |
| RMH_EVT_HOP_WRN | Warns that frame was discarded by Relaying Buffer Full. |
| RMH_EVT_DUP_WRN | Warns that frame was discarded by Duplicate Check Buffer Full. |

### 8.5.1 RMH_EVT_RECEIVE_IND

RMH_EVT_RECEIVE_IND

This event notifies that frame addressed to own node was received.
When the frame that addressed to either own node ID or all nodes is received by executing R_MH_Receive(), this event is notified by the callback function.
If an encrypted frame is received, data decrypted is stored in the parameter *data*.

Parameters:

| | |
|---|---|
| data[21] | Frame Data |
| len | Frame Data Length (byte) |
| dst | Destination Node ID |
| org | Originator Node ID |
| seq | un-encrypted frame: Sequence Number |
| counter | encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>true：Encrypted frame<br>false：Un-encrypted frame |

### 8.5.2 RMH_EVT_OPTION_IND

RMH_EVT_RECEIVE_IND

This event notifies that the option data frame addressed to own node was received.
When the optional data frame that addressed to either own node ID or all nodes is received by executing R_MH_Receive(), this event is notified by the callback function.
If an encrypted option data frame is received, data encrypted is stored in the parameter *data*.

Parameters:

| | |
|---|---|
| data[21] | Option Data |
| len | Option Data Length (byte) |
| dst | Destination Node ID |
| org | Originator Node ID |
| seq | un-encrypted frame: Sequence Number |
| counter | encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>true：Encrypted frame<br>false：Un-encrypted frame |

### 8.5.3 RMH_EVT_STOP_CMP

| RMH_EVT_STOP_CMP |
|---|
| This event notifies that reception processing is stopped completely.<br>When reception processing is stopped by executing R_MH_Stop(), this event is notified by the callback function. |
| Parameters:<br>None |

### 8.5.4 RMH_EVT_SEND_CMP

| RMH_EVT_SEND_CMP | |
|---|---|
| This event notifies that frame transmission is finished completely.<br>After the frame is transmitted by executing R_MH_Send(), this event is notified by the callback function. | |
| Parameters: | |
| seq | un-encrypted frame: Sequence Number |
| counter | encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>true：Encrypted frame<br>false：Un-encrypted frame |

### 8.5.5 RMH_EVT_ENCCNT_WRN

| RMH_EVT_ENCCNT_WRN | |
|---|---|
| This event notifies that Encryption Nonce Counter that is set to an encrypted frame is near end and is reset to 0 soon. (R01AN4466 only).<br>This event is notified only when Security feature is enabled by R_MH_Security() and Nonce counter value exceeds the threshold of warning Nonce counter.<br><br>If the event is notified, it is necessary to notify that Nonce counter is reset to all other nodes.<br>(Note that this handling is not implemented in this version.) | |
| Parameters: | |
| counter | Encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>always true：Encrypted frame |
| Supplementation:<br>The threshold of Nonce counter is defined by the macro *ENCCNT_THRESHOLD* in the following file.<br>– Project_Source\application\src\r_multihop.c<br><br>106: /* encryption counter warning threshold to notify application */<br>107: #define ENCCNT_THRESHOLD (0xFFFFFFF1UL) | |

### 8.5.6 RMH_EVT_HOP_WRN

RMH_EVT_HOP_WRN

This event notifies that received frame should be relayed to other nodes but discarded.
If a frame is discarded because the relay frame buffer is full and a frame cannot be stored, this event is notified by the callback function.

If this event is notified, the following factors can be considered.

- The number of nodes transmitting frames is too many.
- Frame transmission frequency sent by application is too high.

If this event is notified, it is necessary to reduce a relay frequency by reconfiguring a frame transmission frequency of application. In addition, refer to Supplementation and expand the size of a buffer to store frame.

Parameters:

| | |
|---|---|
| dst | Destination Node ID |
| org | Originator Node ID |
| seq | un-encrypted frame: Sequence Number |
| counter | encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>true：Encrypted frame<br>false：Un-encrypted frame |

Supplementation:

The number of the relaying frame buffer is defined by the macro *HOP_BUFFER_NUM* implemented in the following file.

- Project_Source\application\src\r_multihop.c

Relay feature uses the timer, so the number of time management buffer is equal to the number of relaying frame buffer.

```
95:    /* hop frame buffer size */
96:    #define HOP_BUFFER_NUM          (INDEXTIMER_NUM)
```

The number pf time management buffer is defined by the macro *INDEXTIMER_NUM* implemented in the following file.
If you change this macro, it is necessary to change it to the value which is power of 2.

- Project_Source\application\src\driver\timer\r_timer.h

```
45:    /* the number of timer index (power of 2 (2^N)) */
46:    #define INDEXTIMER_NUM                  (0x10)
```

### 8.5.7 RMH_EVT_DUP_WRN

RMH_EVT_DUP_WRN

This event notifies that the new frame is discarded.
If a new frame is discarded because the frame check buffer is full and a frame cannot be stored, this event is notified by the callback function.

If this event is notified, it is necessary to expand the size of a buffer to store frame. Regarding how to expand the buffer size, refer to Supplementation.

Parameters:

| | |
|---|---|
| dst | Destination Node ID |
| org | Originator Node ID |
| seq | un-encrypted frame: Sequence Number |
| counter | encrypted frame: Nonce Counter |
| enc | Encrypted Frame Flag<br>true：Encrypted frame<br>false：Un-encrypted frame |

Supplementation:

The number of duplicate frame check buffer is defined by the macro *MH_NODE_NUM* implemented in the following file.

- Project_Source\application\src\r_multihop.c

Change the macro value to the number of nodes existing in the same network. This value can be set up to 0xFE.

```
48:    /* the number of nodes existing in the network */
49:    #define MH_NODE_NUM             (0x40)
```

## 8.6 Sequence

Subsection 8.6.1 and 8.6.2 show the sequence of Multi-Hop API.

Application layer initializes Multi-Hop layer by executing R_MH_Init() and then starts frame reception operation by executing R_MH_Receive(). When it uses the security feature, it enables the security feature, and sets and encryption key to Multi-Hop layer by executing R_MH_Security(). (R01AN4466 only)

Multi-Hop layer notifies each event by calling callback function registered by R_MH_Init().

Note that application layer should execute R_MH_Proc(), to execute processing of Multi-Hop layer for relaying frames or notifying events by calling callback function. Regarding how to implement, refer to subsection 8.4.2 "R_MH_Proc".

### 8.6.1 Frame Reception

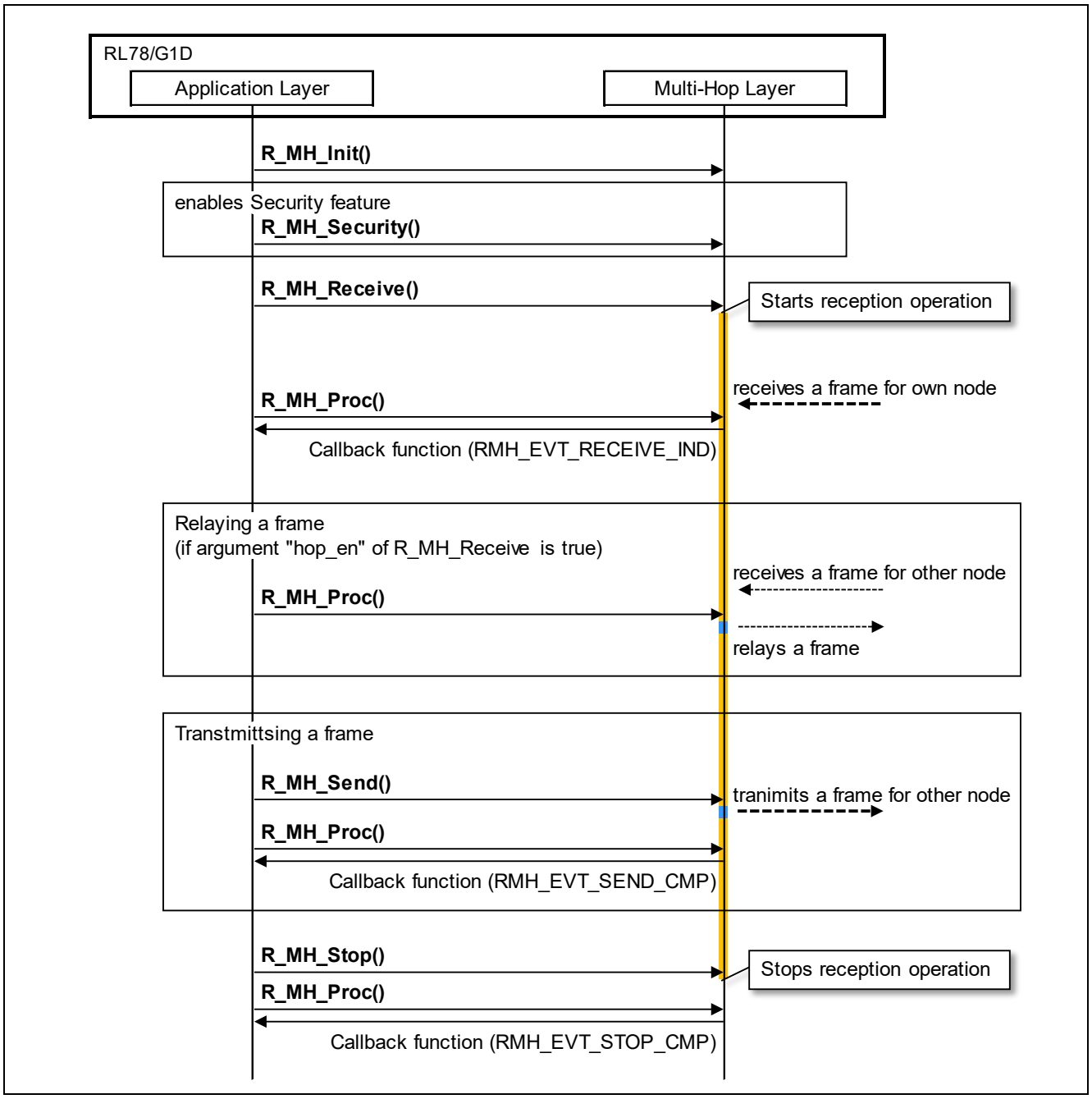



**Figure 8-1 Frame Reception**

### 8.6.2 Frame Transmission

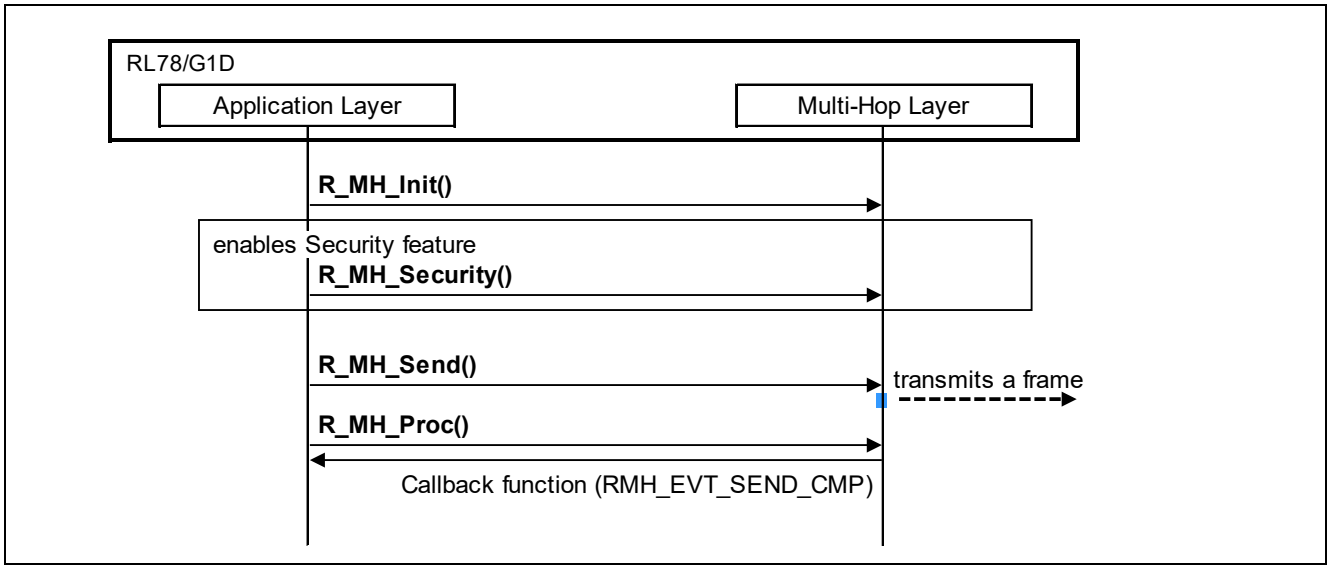



**Figure 8-2 Frame Transmission**

## 8.7 Frame Reception

By executing R_MH_Receive(), frame reception operation starts.

Reception operation switches Advertising channel; 37, 38, and 39ch periodically and receive frames.

By executing R_MH_Stop(), frame reception operation stops. After stopping, callback function is called and RMH_EVT_STOP_CMP event is notified.

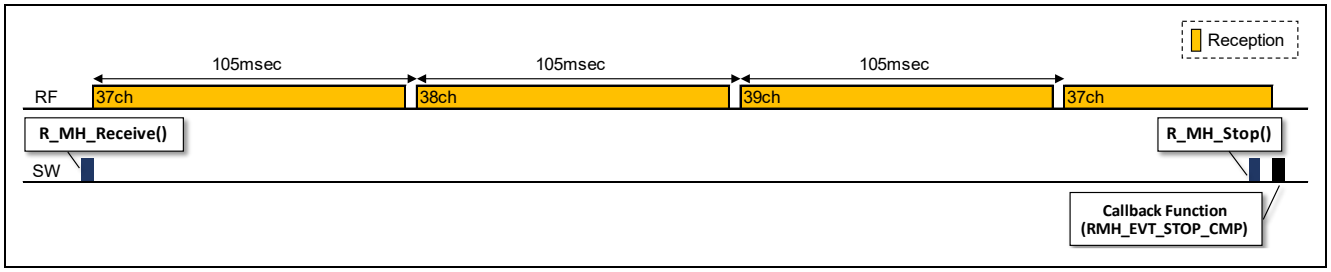



**Figure 8-3 Frame Reception**

### 8.7.1 Receiving Frame addressed to own node

When a frame addressed to own node is received, callback function is called and RMH_EVT_RECEIVE_IND event is notified.

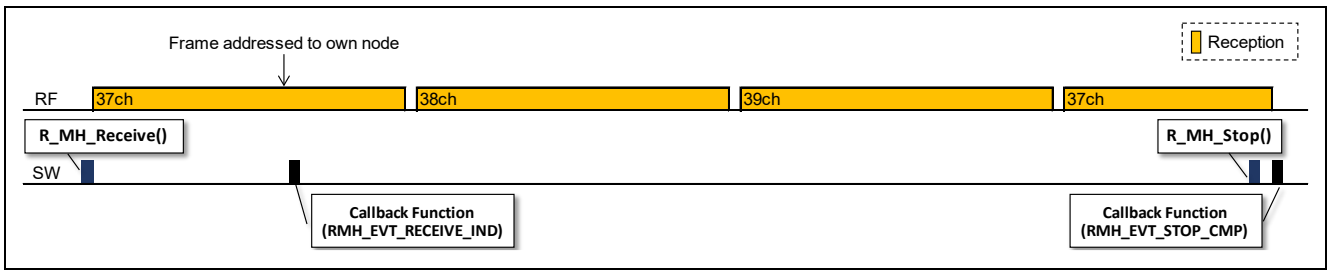



**Figure 8-4 Frame Reception**

### 8.7.2 Relaying Frame addressed to another node

By executing R_MH_Receive() that its argument hop_en is true, frame relaying operation is enabled.

When a frame addressed to another node is received, this frame is relayed.

A frame is transmitted after randomized time to avoid collision of frame transmitted by around nodes. And a frame is transmitted to all Advertising channel three times to reach to other nodes as many as possible.

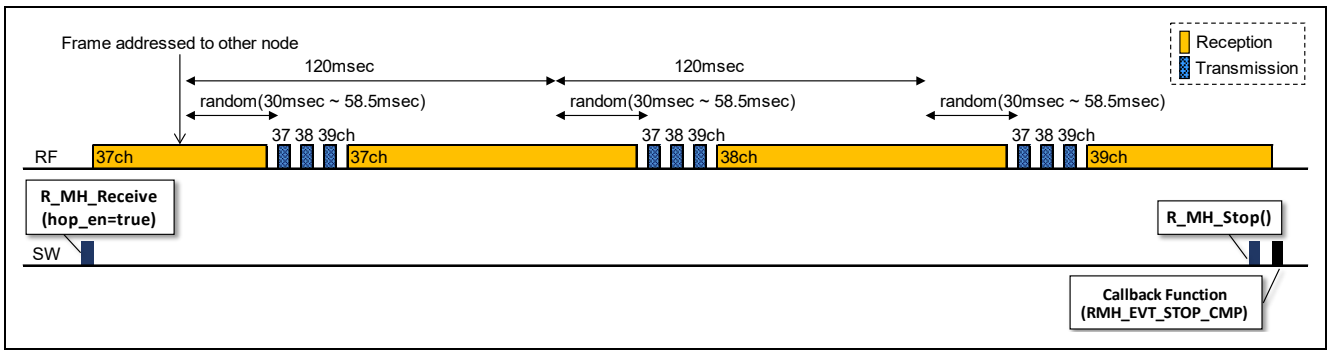



**Figure 8-5 Frame Relay**

## 8.8 Frame Transmission

By executing R_MH_Send(), frame is transmitted.

A frame is transmitted after randomized time to avoid collision of frame transmitted by around nodes. And a frame is transmitted to all Advertising channel three times to reach to other nodes as many as possible.

After frame transmission, RMH_EVT_SEND_CMP event is notified by calling callback function.

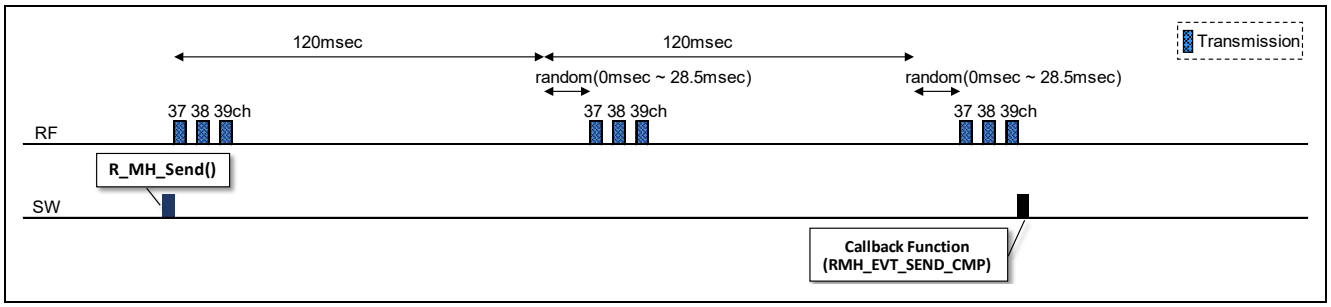



**Figure 8-6 Frame Transmission**

### 8.8.1 Transmitting frame in reception operation

It is possible to transmit a frame even if frame reception operation has executed.

By executing R_MH_Send() in reception operation, a frame is transmitted. After transmitting, callback function is called and RMH_EVT_RECEIVE_IND event is notified.

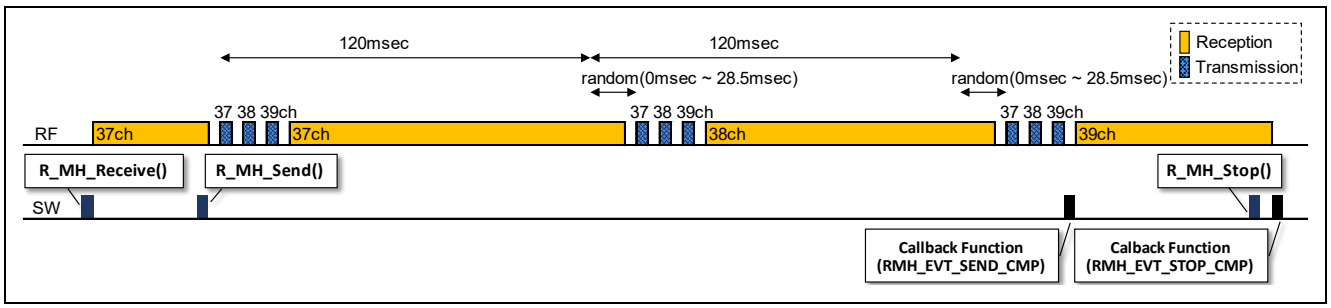



**Figure 8-7 Frame Transmission**

## 8.9 Transmission and Reception Channels

As shown in section 8.7 and 8.8, Multi-Hop layer transmits and receives frames by all Advertising channels.

In addtion, transmittion and reception channel can be limited only one channel. To limit the channel, change the macro MH_ALL_CH_EN to (0) and MH_SINGLE_CH to channel used. These macros are defined in the following file.

- Project_Source\application\src\r_multihop.c

```
51:     /* 1: uses ALL    advertising channels for multi-hop transmission/reception. *
52:     /* 0: uses SINGLE advertising channel  for multi-hop transmission/reception. *
53:     #define MH_ALL_CH_EN            (1)
54:     #if !MH_ALL_CH_EN
55:         #define MH_SINGLE_CH        (RBLE_ADV_CHANNEL_37)
56:     #endif
```

Change to (0)

Change to either 37, 38 or 39

Figure 8-8 shows the frame reception operation on only one channel.

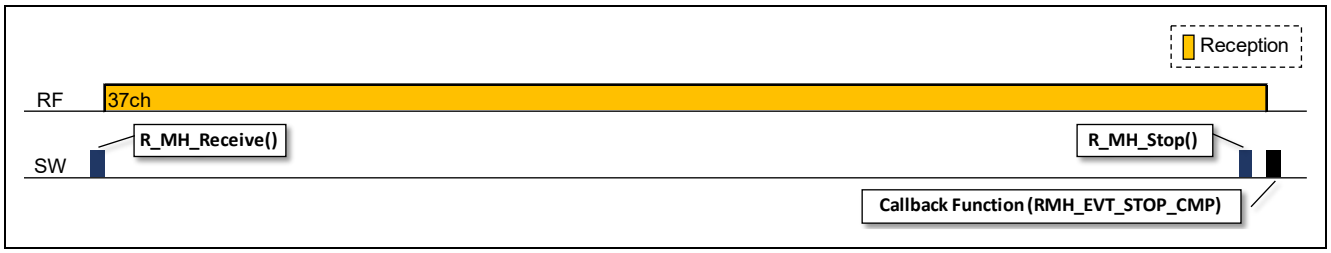



**Figure 8-8 Frame Reception on Only One Channel**

Figure 8-9 shows the frame transmission operation on only one channel. A frame is transmitted three times.

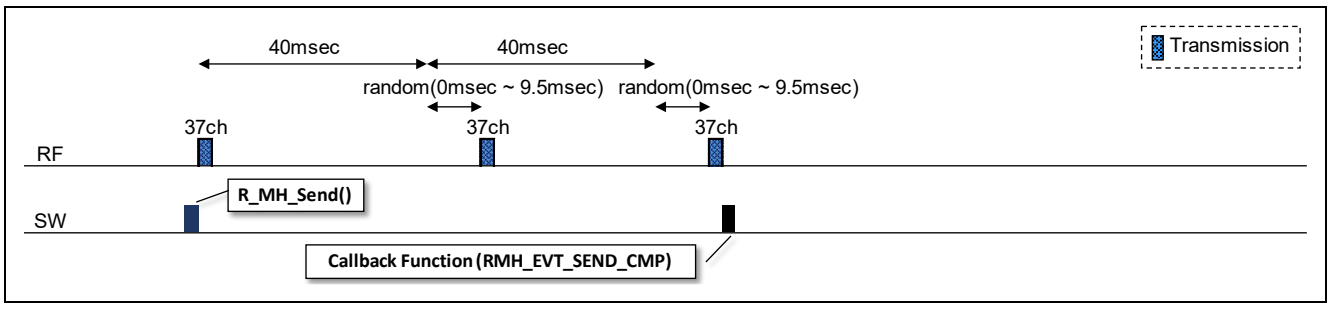



**Figure 8-9 Frame Transmission on Only One Channel**

# 9. Appendix

## 9.1 Path Check Feature

A feature to check frame relay path is implemented in the sample program. By using this feature, you can confirm that which node relayed a frame.

To enable path check feature, change the macro HOP_ROOT_CHECK to (1) defined in the following file.

Note that by enabling this feature, Security feature is disabled.

- Project_Source\application\src\r_multihop.h

```
44:     /* 1: enables the function to check hopping root */
45:     /* 0: normal operation; disable the function to check */
46:     #define HOP_ROOT_CHECK          (1)
```

Change to (1)

Next, write the firmware which is enabled this feature to all evaluation board. By pushing the switch SW2 on the evaluation board, it starts to transmit frames to ID0 periodically.

Node of ID0 outputs the relay path log of received frame via UART with the following format.

*SerialNumber* *ORG* -> *DST* *SEQ* *FrameDataSize* [*FrameNumber*] *RelayPath*

Figure 9-1 shows example log of relay path that node of ID 3 transmits to node of ID0.

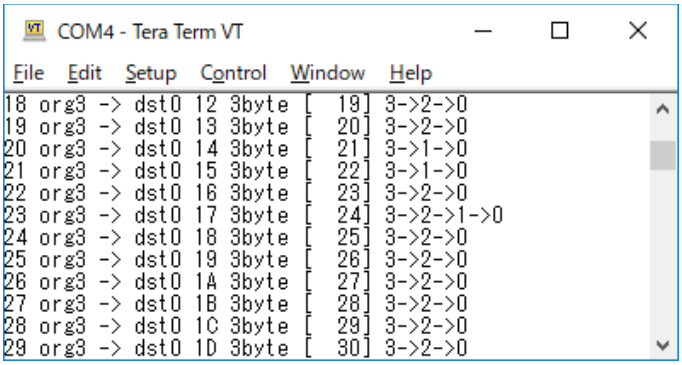



**Figure 9-1 Example Log of Relay Path**

By checking this path log, you can confirm that each frame was relayed the following path.

| | |
|---|---|
| ID=3→ID=2→ID=1→ID=0 | : frame 24 |
| ID=3→ID=1→ID=0 | : frame 21 and 22 |
| ID=3→ID=2→ID=0 | : the other frames |

## 9.2 Device Filter

For demonstration use, device filter is implemented in the sample program. By using it, only advertising packets that is transmitted by the specified device is received, but other advertising packets are discarded. You can evaluate Multi-Hop feature in the case that each frame is relayed in the restricted path.

To enable the device filter, change the macro *DEV_ADDR_FILTER* to (1) defined in the following file.

- Project_Source\application\src\r_multihop.h

```
48:     /* 1: receives the frame from only the device addresses registered in addr_fil
49:     /* 0: normal operation; receives the frame without filtering. */
50:     #define DEV_ADDR_FILTER          (0)
                                          Change to (1)
```

Device Address which is written by using the system configuration included in the package is DC:D7:2C:71:F4:xx of Random Device Address; XX is Node ID.

e.g.) Node ID→0 : DC:D7:2C:71:F4:00
Node ID→1 : DC:D7:2C:71:F4:01

If you use the device filter with no change from default setting, each device transmits and receives frames only between the device having a neighborhood node ID: that means own node ID +1 or own node -1.

Figure 9-2 shows the frame relay path restricted by the device filter of the default setting.

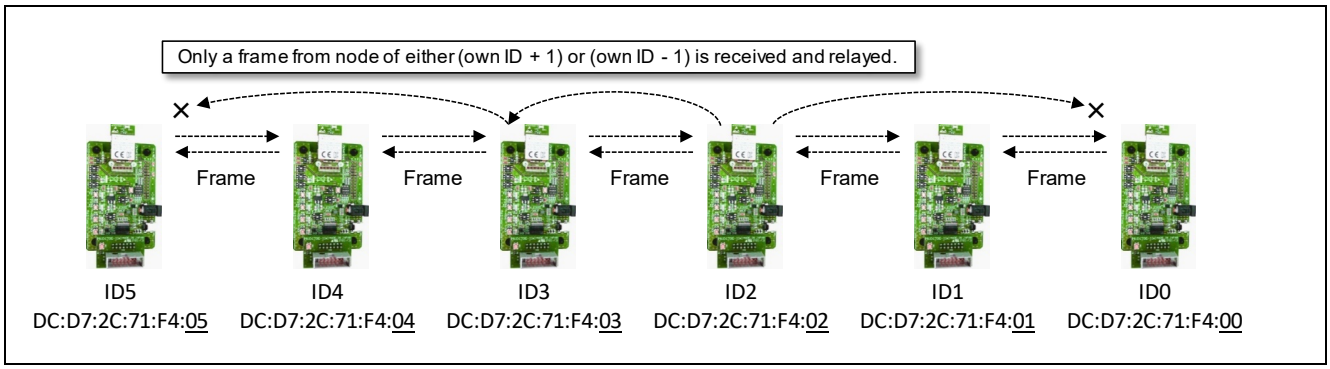



**Figure 9-2 Path Restricted by Device Filter**

The following firmware is built by changing the macro *DEV_ADDR_FILTER* to (1).

- ROM_File\R5F11AGJ_MultiHop(DEV_FILTER).hex

## 9.3 Device Driver

This section explains the specification of device driver functions implemented in the sample program.

Device drivers can be customized, because source code files of device driver are included in the package.

### 9.3.1 Platform (Clock and Port)

The sample program initializes platform: clock and port.

The source codes of the platform driver are included in the following folder.

- Project_Source\application\src\driver\plf

The specification of the platform driver is described as follows.

| bool R_PLF_Init( void ); | |
|---|---|
| This function initializes the platform, clock and ports.<br>- Hi-speed on-chip oscillator<br>- XT1 oscillation and Low-speed on-chip oscillator<br>- Sub Clock output<br>- I/O ports<br><br>This function is assumed that the sample program works on RL78/G1D Evaluation Board (RTK0EN0001D01001BZ). If you use other RL78/G1D board, it is necessary to modify the ports setting implemented in this function. | |
| Parameters: | |
| None | |
| Return: | |
| true | Success |
| false | Failed, User Option Byte is wrong |

Regarding the user option bytes, refer to refer to the chapter25 "Option Byte" in RL78/G1D User's Manual: Hardware (R01UH0515).

### 9.3.2 12-bit Interval Timer

Application layer used 12-bit Interval Timer to transmit a frame periodically.

The source codes of the 12-bit interval timer driver are included in the following folder.

- Project_Source\application\src\driver\it

The specification of the 12-bit interval timer driver is described as follows.

| void R_IT_Init( void ); | |
|---|---|
| This function initializes 12-bit Interval Timer. | |
| Parameters: | |
| None | |
| Return: | |
| None | |

| void R_IT_Start( uint16_t time, IT_CALLBACK callback ); | |
|---|---|
| This function starts 12-bit Interval Timer Operation.<br>The 12-bit Interval timer can still operate even when MCU is STOP mode.<br>When the timer expires, callback function specified by argument *callback* is executed. | |
| Parameters: | |
| time | Compare Value<br>0x0001 to 0x0FFF<br>subsystem clock (32.768kHz) : (time+1) × 30.5usec<br>low-speed on-chip oscillator clock (15kHz) : (time+1) × 66.6usec |
| callback | Callback Function to notify Timer Expires<br>typedef void (*IT_CALLBACK)( void ); |
| Return: | |
| None | |

| void R_IT_Stop( void ); |
|---|
| This function stops 12-bit Interval Timer Operation. |
| Parameters: |
| None |
| Return: |
| None |

### 9.3.3 Timer Array Unit

Multi-Hop layer uses Timer Array Unit to control frame transmission cycle. In addition, application can use the timer array unit for various use.

channel 0: used by Beacon Stack

channel 1 and 2: used by Multi-Hop layer

channel 3: used by UART driver

channel 4 to 7: not used, application can use them

The source codes of the timer array unit driver are included in the following folder.

- Project_Source\application\src\driver\timer

The specification of the timer array unit driver is described as follows.

| void R_TIMER_Init( void ); | |
|---|---|
| This function initializes Timer Array Unit.<br>After initialization, Channel 1 to 7 of the Timer Array Unit can be used. | |
| Parameters: | |
| None | |
| Return: | |
| None | |

| bool R_TIMER_IsActive( void ); | |
|---|---|
| This function checks operation state of Timer Array Unit. | |
| Parameters: | |
| None | |
| Return: | |
| true | any channel of Timer Array Unit is operating. |
| false | all channel of Timer Array Unit is not operating. |

| void R_TIMER1_Start( uint16_t time, TIMER_CALLBACK callback );<br>:<br>void R_TIMER7_Start( uint16_t time, TIMER_CALLBACK callback ); | |
|---|---|
| Each function starts timer operation of each channel 1 to 7of Timer Array Unit.<br>The Timer Array Unit stops when MCU is STOP mode.<br>When the timer expires, callback function specified by argument *callback* is executed. | |
| Parameters: | |
| time | Compare Value<br>1 to 500<br>The timer per 1 clock is 0.5msec |
| callback | Callback Function to notify Timer Expires<br>typedef void (*TIMER_CALLBACK)( void ); |
| Return: | |
| None | |

| void R_TIMER1_Stop( void );<br>:<br>void R_TIMER7_Stop( void ); |
|---|
| Each function stops timer operation of each channel 1 to 7of Timer Array Unit. |
| Parameters: |
| None |
| Return: |
| None |

| void R_TIMER1_Elapsed( void );<br>:<br>void R_TIMER7_Elapsed( void ); | |
|---|---|
| Each function checks elapsed time of each channel 1 to 7of Timer Array Unit. | |
| Parameters: | |
| None | |
| Return: | |
| Elapsed Time<br>1 to 500<br>The time per 1 clock is 0.5msec | |

| void R_TIMERX_Init( TIMERX_CALLBACK callback ); | |
|---|---|
| This function initializes Index Timer.<br>The Index Timer uses only Timer Array Unit Channel 0 and can measure multiple time simultaneously.<br>Each Index Timer is identified by index. Index can be specified in the range of 0 to 15.<br>The index of expired timer is notified by argument of the callback function. | |
| Parameters: | |
| callback | Callback Function to notify Each Index Timer Expires.<br>typedef void (*TIMERX_CALLBACK)( uint8_t );<br>The index of timer expiring is notified by argument of the callback function. |
| Return: | |
| None | |

| void R_TIMERX_Start( uint8_t idx, uint16_t time ); | |
|---|---|
| This function starts each Index Timer operation.<br>Note that the Index Timer uses Timer Array Unit. So, this timer also stops if MCU is STOP mode.<br>When the Index Timer expires, the callback function specified by argument *callback* of R_TIMERX_Init(). | |
| Parameters: | |
| idx | Index of Index Timer<br>0 to 15 |
| time | Compare Value<br>1 to 500<br>The timer per 1 clock is 0.5msec |
| Return: | |
| None | |

### 9.3.4 Data Flash

Application layer uses data flash memory to store Nonce counter value for the security feature.

The source codes of the data flash driver are included in the following folder. (R01AN4466 only)

- Project_Source\application\src\driver\dataflash

Note that this driver uses Data Flash Library.

EEPROM Emulation Library Pack02 Package Ver.2.00(for CA78K0R/CC-RL Compiler) for RL78 Family
https://www.renesas.com/software-tool/data-flash-libraries

The specification of the data flash driver is described as follows.

| uint8_t R_DFL_Init( void ); | |
|---|---|
| This function initializes Data Flash Driver. | |
| Parameters: | |
| None | |
| Return: | |
| DFL_OK | Success |
| others | Error |

| uint8_t R_DFL_Start( void ); | |
|---|---|
| This function starts Data Flash operation.<br>After executing this function, Data Flash becomes accessible and below functions can be executed.<br>- R_DFL_Read()<br>- R_DFL_Write()<br>- R_DFL_Refresh()<br>- R_DFL_Format()<br><br>After executing this function, MCU cannot be changed into HALT mode or STOP mode.<br>Before changing into either HALT mode or STOP mode, it is necessary to execute R_DFL_STOP(). | |
| Parameters: | |
| None | |
| Return: | |
| DFL_OK | Success |
| others | Error |

| uint8_t R_DFL_Stop( void ); | |
|---|---|
| This function stops Data Flash operation.<br>After executing this function, Data Flash becomes inaccessible. | |
| Parameters: | |
| None | |
| Return: | |
| DFL_OK | Success |
| others | Error |

| uint8_t R_DFL_Read( const uint8_t id, void* addr ); | | |
|---|---|---|
| This function reads data specified by argument *id* from Data Flash.<br>Below Data ID are defined in advance.<br>Data ID for 4byte: EEL_ID_DATA01 and EEL_ID_DATA02<br>Data ID for 8byte: EEL_ID_DATA03 and EEL_ID_DATA04<br>Data ID for 16byte: EEL_ID_DATA05 and EEL_ID_DATA06<br>Data ID for 32byte: EEL_ID_DATA07 and EEL_ID_DATA08 | | |
| Parameters: | | |
| | id | Data ID |
| | addr | Address of Data Buffer |
| Return: | | |
| | DFL_OK | Success |
| | DFL_ERR_PARAMETER | Error: invalid parameter |
| | others | Error |

| uint8_t R_DFL_Write( const uint8_t id, void* addr ); | | |
|---|---|---|
| This function writes data specified by argument *id* to Data Flash.<br>Below Data ID are defined in advance.<br>Data ID for 4byte: EEL_ID_DATA01 and EEL_ID_DATA02<br>Data ID for 8byte: EEL_ID_DATA03 and EEL_ID_DATA04<br>Data ID for 16byte: EEL_ID_DATA05 and EEL_ID_DATA06<br>Data ID for 32byte: EEL_ID_DATA07 and EEL_ID_DATA08 | | |
| Parameters: | | |
| | id | Data ID |
| | addr | Address of Data Buffer |
| Return: | | |
| | DFL_OK | Success |
| | DFL_ERR_PARAMETER | Error: invalid parameter |
| | DFL_ERR_NO_INSTANCE | Error: data not existed |
| | DFL_ERR_POOL_FULL | Error: block used is full |
| | others | Error |

| uint8_t R_DFL_Refresh( void ); | | |
|---|---|---|
| This function refreshes block state of Data Flash.<br>Data in Data Flash are maintained. | | |
| Parameters: | | |
| | None | |
| Return: | | |
| | DFL_OK | Success |
| | others | Error |

| uint8_t R_DFL_Format( void ); | | |
|---|---|---|
| This function formats Data Flash.<br>Data in Data Flash are erased. | | |
| Parameters: | | |
| | None | |
| Return: | | |
| | DFL_OK | Success |
| | others | Error |

### 9.3.5 UART

Application layer uses UART to output frame data log.

The source codes of the UART driver are included in the following folder.

- Project_Source\application\src\driver\uart

The specification of the UART driver is described as follows.

| bool R_UART_Init( UART_INFO* info ); | | | |
|---|---|---|---|
| This function initializes UART0 of Serial Array Unit. | | | |
| Parameters: | | | |
| | info | baudrate | UART Baud Rate<br>UART_BAUDRATE_9600_BPS 9,600bps<br>UART_BAUDRATE_115200_BPS 115,200bps<br>UART_BAUDRATE_1M_BPS 1,000,000bps |
| | | rx_tout | UART Reception Timeout<br>1 to 100msec<br>If data of the specified size is not received within this timeout time, received data is discarded and UART reception restarts.<br>If 0 is set, Rx Timeout is disabled. |
| | | rx_callback | callback function to notify data is received after executing R_UART_Rx() |
| | | tx_callback | callback function to notify data is transmitted by executing R_UART_Tx() |
| | | err_callback | callback function to notify reception error occurs after executing R_UART_Rx() |
| Return: | | | |
| | true | | Success |
| | false | | Failed |

| bool R_UART_IsActive( void ); | | |
|---|---|---|
| This function checks UART operation status. | | |
| Parameters: | | |
| | None | |
| Return: | | |
| | true | UART operation works. |
| | false | UART operation does not work. |

| void R_UART_Rx( __near uint8_t *rxbuf, const uint16_t size ); | | |
|---|---|---|
| This function starts UART reception operation.<br>Received data is stored in the data buffer specified by the argument *rxbuf*.<br>When data of the size specified by the argument *size* is received completely, UART driver executes callback function registered by the argument *info->rx_callback* of R_UART_Init(). | | |
| Parameters: | | |
| | rxbuf | Received Data Buffer |
| | size | Data Size<br>1 to 1024byte |
| Return: | | |
| | None | |

| void R_UART_Tx( __near const uint8_t *txbuf, const uint16_t size ); | | |
|---|---|---|
| This function starts UART data transmission.<br>Data in the data buffer specified by the argument *txbuf* is transmitted.<br>When data of the size specified by the argument *size* is transmitted completely, UART driver executes callback function registered by the argument *info->tx_callback* of R_UART_Init(). | | |
| Parameters: | | |
| | txbuf | Data to be transmitted |
| | size | Data Size<br>1 to 1024byte |
| Return: | | |
| | None | |

| void R_LOG_Init( uint8_t baudrate ); | | |
|---|---|---|
| This function initializes the log output feature.<br>The log output feature has multiple log buffer and transmits data stored in the buffer by using UART.<br>This function executes R_UART_Init() to use UART. | | |
| Parameters: | | |
| | baudrate | UART Baudrate<br>UART_BAUDRATE_9600_BPS 9,600bps<br>UART_BAUDRATE_115200_BPS 115,200bps<br>UART_BAUDRATE_1M_BPS 1,000,000bps |
| Return: | | |
| | true | Success |
| | false | Failed |

| bool R_LOG_Send( char_t* string ); | | |
|---|---|---|
| This function transmits log string.<br>Log string specified by the argument *string* is stored to log buffer and is transmitted by UART.<br>This function executes R_UART_Tx() to transmits log string. | | |
| Parameters: | | |
| | string | Log String<br>1 to 200byte |
| Return: | | |
| | true | Success to store log to buffer.<br>Stored log is transmitted in order. |
| | false | Failed to store log to buffer, because of buffer full. |

### 9.3.6 External Input Interrupt

Application layer uses External Input Interrupt to detect that switch of the evaluation board is pushed.

The source codes of the external input interrupt driver are included in the following folder.

- Project_Source\application\src\driver\input

The specification of the external input interrupt driver is described as follows.

| void R_INPUT_Init( input_callback_t callback ); | | |
|---|---|---|
| This function initializes external input interrupt.<br>This function enables rising edge detection of INTP5/P16 connected to switch SW2 of the evaluation board.<br>When the external input interrupt occurs, the external input interrupt driver executes callback function specified by the argument *callback*. | | |
| Parameters: | | |
| | callback | Callback Function to notify External Input Interrupt<br>typedef void (*input_callback_t)(void); |
| Return: | | |
| | None | |

### 9.3.7 LED

Upon receiving a frame, Application layer changes LED status of the evaluation board. Moreover, upon relaying a frame, Multi-Hop layer changes LED status.

The source codes of the LED driver are included in the following folder.

- Project_Source\application\src\driver\led

The specification of the LED driver is described as follows.

| void R_LED_Init( void ); | |
|---|---|
| This function initializes P120, P147, P03, and P60 connected to LED1 to LED4 on RL78/G1D Evaluation Board. | |
| Parameters: | |
| None | |
| Return: | |
| None | |

| void R_LED_Set( uint8_t led ); | | |
|---|---|---|
| This function changes LED states on RL78/G1D Evaluation Board. | | |
| Parameters: | | |
| | led | LED status<br>bit0 : LED1 (0: off, 1: on)<br>bit1 : LED2 (0: off, 1: on)<br>bit2 : LED3 (0: off, 1: on)<br>bit3 : LED4 (0: off, 1: on) |
| Return: | | |
| | None | |

## 9.4 Example of Measuring Frame Transport Ratio

This section shows the evaluation result of the frame transport ratio of Multi-Hop Feature in our environment.

### 9.4.1 Notice

The result described in this section is NOT guaranteed performance but for reference only. It is recommended to measure with the condition of actual use case.

### 9.4.2 Operation

This evaluation was conducted in a building of ours company. Many wireless LAN devices and other Bluetooth devices had run in this environment.

Figure 9-3 shows the overview of the evaluation. Each RL78/G1D Evaluation Board was placed at intervals of 6.5meters. A firmware to evaluate the frame transport ratio was written to these boards. And, ID number was assigned to each board in order. The board of ID0 was connected to PC via USB, and other boards were connected to a battery.

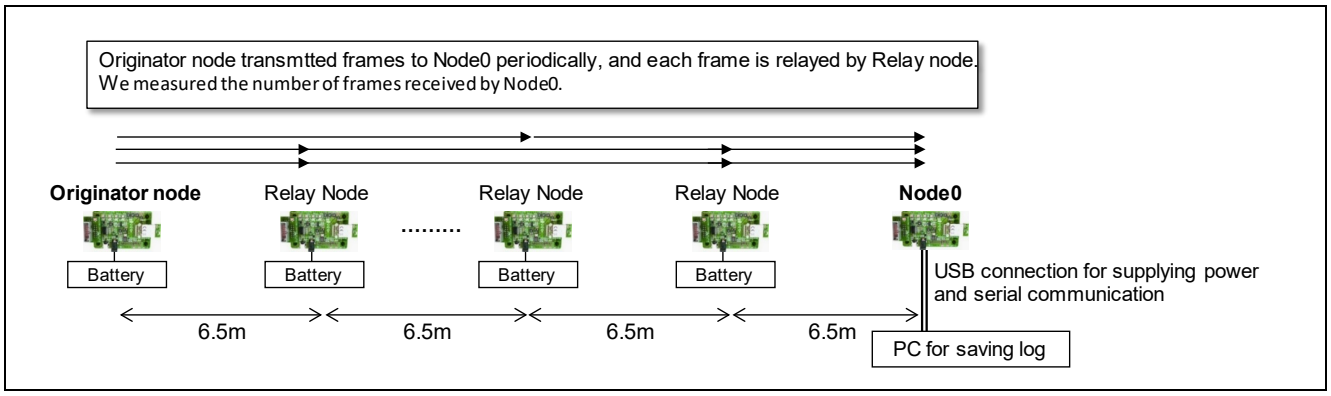



**Figure 9-3 Overview of Measuring Frame Transport Ratio**

Originator node transmitted Multi-Hop frames to node 0. In the evaluation, we measured a ratio that node0 received frames as a frame transport ratio.

Table 9-1 shows the configuration of the evaluation program. For this evaluation, transmit power was changed from the default configuration of Multi-Hop layer.

**Table 9-1 Configuration of the evaluation program**

| Item | Condition |
|---|---|
| MCU Main System Clock | 8MHz |
| DC-DC converter | Uses RF on-chip DC-DC converter. |
| Oscillator for RF slow clock | Uses RF on-chip oscillator and executes calibration. |
| Transmission Power | -15dBm (changed from default)<br>To shorten a reach distance of frame transmitted by each node, transmission power is changed from 0dBm to -15dBm. |
| Transmission Channel | Uses 3channels (37,38,39ch) |
| Multi-Hop Operation | Enables Path Check feature and disables Security feature.<br>Regarding the Path Check feature, refer to section 9.1 "Path Check Feature". |

Figure 9-4 shows the sequence of the evaluation program measuring frame transport ratio.

Originator node transmits frames to node0. Nodes other than originator node and node0 relay frames. Frames reach node0 finally. Note that frame relay path may be different from each other.

Frame transport ratio is calculated with the following expression.

*Frame transport ratio = the number of frames received by node0 / the number of frames transmitted by originator*

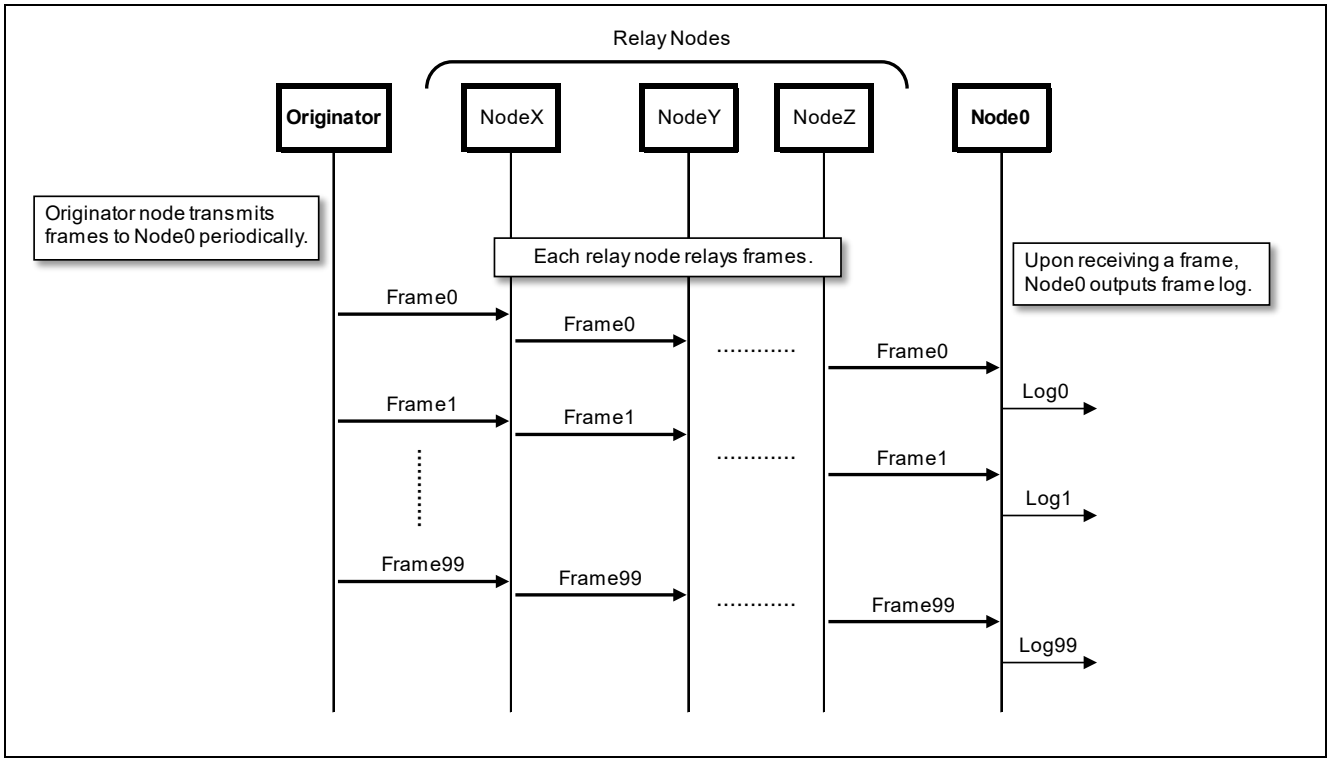



**Figure 9-4 Sequence of Measuring Frame Transport Ratio**

### 9.4.3 Frame transport ratio in the case of with or without relaying

(1) **Procedure**

We measured a transport ratio in the case of both with and without relay nodes. By this measurement, we checked the effect of frame relaying by Multi-Hop Feature.

Figure 9-5 shows the node composition with relay nodes. Similarly, Figure 9-6 shows the composition without relay node.

In the both composition, originator node transmitted a frame to node0 every 500 milliseconds. Further, each transport ratio was measured respectively in condition that originator node is node1 to node 6.

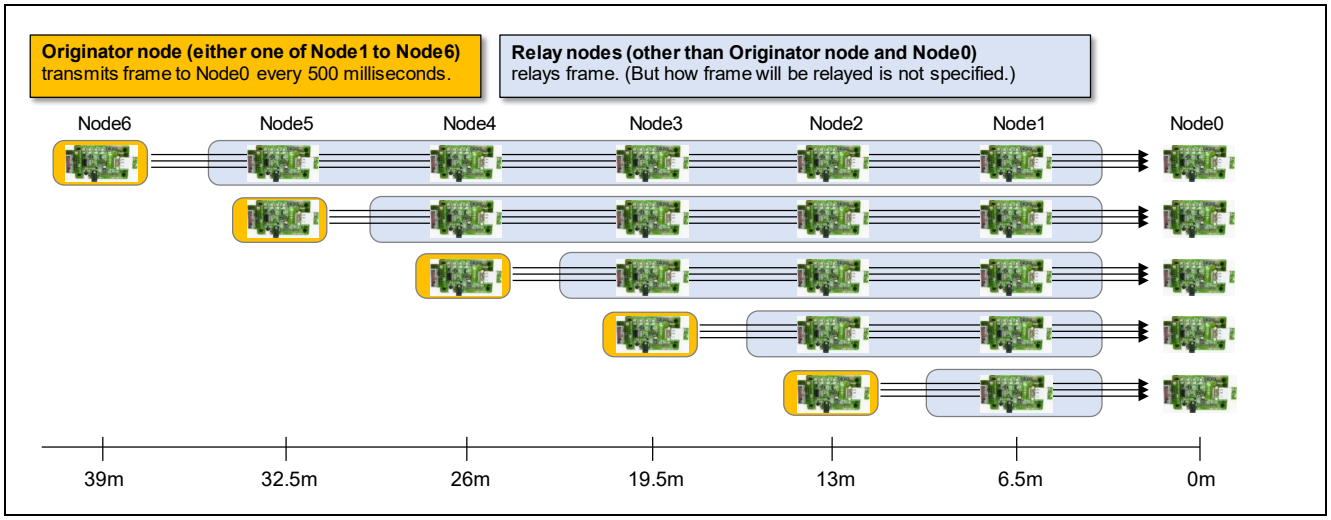



**Figure 9-5 Measurement Composition without Relay Node**

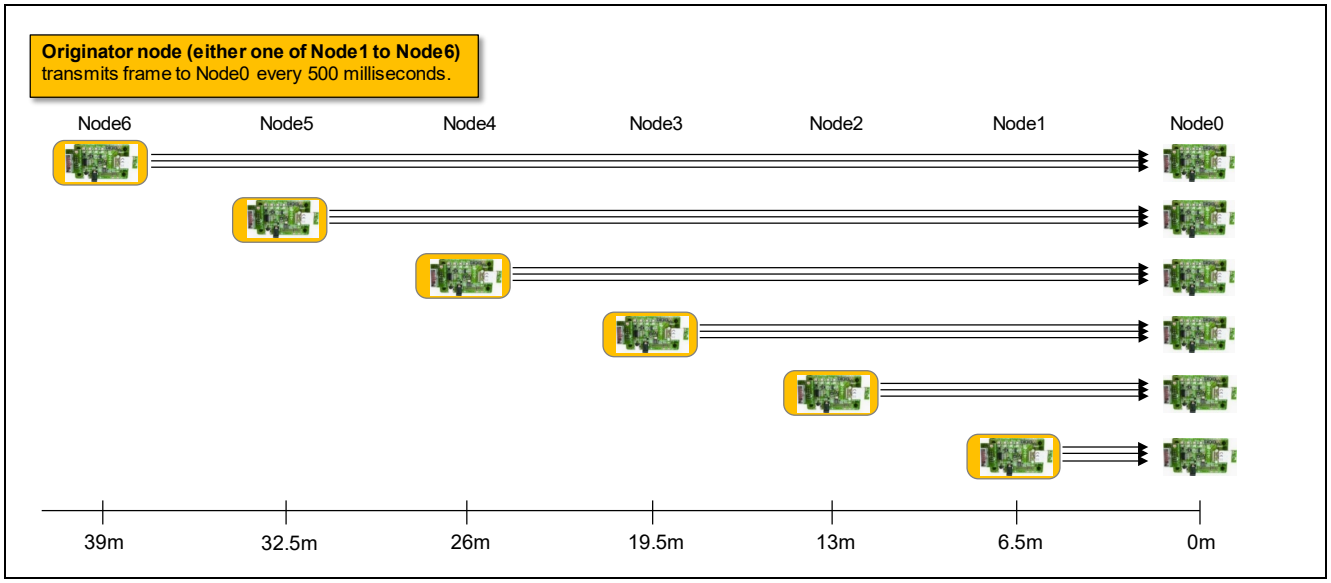



**Figure 9-6 Measurement Composition with Relay Node**

(2) **Result**

Table 9-2 and Figure 9-7 shows the measurement result of frame transport ratio in the case of both with and without relay node. Each originator node transmitted frames every 500 milliseconds.

**Table 9-2 Result of frame transport ratio in the case of both with and without relay node**

| Originator | Without Relay | With Relays |
|---|---|---|
| **Node1** | 99.5% | |
| **Node2** | 100.0% | 100.0% |
| **Node3** | 70.5% | 100.0% |
| **Node4** | 0.0% | 100.0% |
| **Node5** | 0.0% | 100.0% |
| **Node6** | 1.0% | 100.0% |

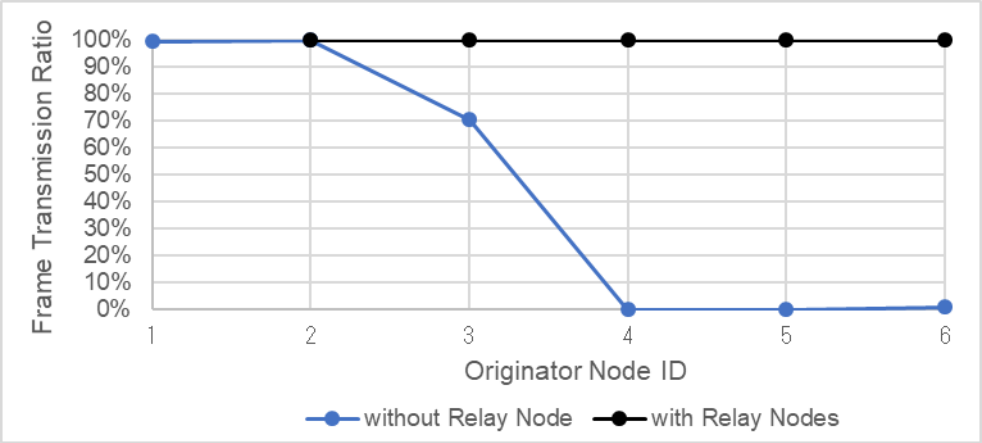



**Figure 9-7 The Result of Frame Transport Ratio**

In the measurement result, when there was not relay node, the transport ratio declined gradually as it gets longer. When originator was either node4 to node6, the transport ratio was almost 0%.

On the other hand, when there were relay nodes, the transport ratio was 100% even if originator node was either node3 to node6.

### 9.4.4 Frame transport ratio in the case of shortened frame transmission cycle

In the previous measurement, originator node transmitted frames every 500 milliseconds. This subsection shows a frame transport ratios of shortened transport cycle.

(1) **Procedure**

We also measured a transport ratio in the case of shorted frame transmission cycle. The cycle was 100 milliseconds, 250 milliseconds, and 500 milliseconds.

By the following configuration change, frame transmission cycle and frame relay cycle were shortened.

- Project_Source\application\src\r_multihop.c

Before changing r_multihop.c:

```
71:     /* unit:0.5msec value:120msec */
72:     #define MH_TX_INTERVAL      (240)
73:
74:     /* unit:0.5msec resolution:1.5msec range:30msec to 58.5msec */
75:     #define MH_RAND_RANGE       (57)
76:     #define MH_RAND_OFFSET      (60)
77:     #define MH_RAND_TIME()      (((((uint16_t)rand()) & 0xFF) * MH_RAND_RANGE * 3)
78:     #define MH_RAND_MAX         (MH_RAND_RANGE + MH_RAND_OFFSET)
 ------------------------------------------------------------------------------------
724:     R_HopTimer_Start(idx, gs_hop_buf[idx].delay);
```

After changing r_multihop.c:

```
71:     /* unit:0.5msec value:10msec */
72:     #define MH_TX_INTERVAL      (20)
73:
74:     /* disable random delay */
75:     #define MH_RAND_RANGE       (0)     /* 0msec */
76:     #define MH_RAND_OFFSET      (0)     /* 0msec */
77:     #define MH_RAND_TIME()      (0)     /* 0msec */
78:     #define MH_RAND_MAX         (0)
 ------------------------------------------------------------------------------------
724:     R_HopTimer_Start(idx, MH_TX_INTERVAL);
```

(2) **Result**

Table 9-3 shows the result of frame transport ratio in the case of shortened frame transmission cycle: 100 milliseconds, 250 milliseconds. and 500 milliseconds.

**Table 9-3 Result of frame transport ratio in the case of shortened frame transmission cycle**

| Originator | Cycle 100msec | Cycle 250msec | Cycle 500msec |
|---|---|---|---|
| **Node2** | 99.7% | 100.0% | 100.0% |
| **Node3** | 99.7% | 100.0% | 100.0% |
| **Node4** | 99.3% | 99.7% | 100.0% |
| **Node5** | 100.0% | 100.0% | 100.0% |
| **Node6** | 100.0% | 100.0% | 100.0% |

In the measurement result, by shortening a transmission cycle, the transport ratio declined slightly. At least in this environment, it is necessary to set a transmission cycle over than 250 milliseconds.

Shortening a transmission cycle might increase a probability of frame collision; for example, a frame may collide with frames from originator node or other relay node, or packets from another wireless communication device. And then a transport ratio might decline.

In this measurement, each node was placed in a line. However, note that when nodes are placed at high density, a probability of frame collision may increase.

**Website and Support**

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

## General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   ¾ The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   ¾ The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   ¾ The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   ¾ When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   ¾ The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

# Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

## SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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


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

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

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

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

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

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

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

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

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

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

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

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




Colophon 7.1