R-IN32M3 Module (RY9012A0)

User's Implementation Guide (uGOAL Edition)

R30AN0402EJ0103
Rev.1.03
Dec.15.2023

Introduction

This document describes how to apply the sample software of host CPU that controls R-IN32M3 Module included in "R-IN32M3 Module (RY9012A0) Sample Package" to development of Industrial Ethernet products. Implementation of user application and hardware layer, and examples of profile setting including device definition file of each industrial Ethernet protocol are described.

Target Device

R-IN32M3 Module (RY9012A0)

Contents

List of Abbreviations and Acronyms

In this document, the terms below are defined as follows:

Terms	Description
This sample	The sample program for the host microcomputer that controls the R-IN32M3 Module in the industrial network sample program for the R-IN32M3 Module.
API	Application Programming Interface
GOAL / uGOAL	Generic Open Abstraction Layer See "R-IN32M3 Module (RY9012A0) User's Manual: Software (R17US0002ED****)"
FSP	Flexible Software Package An enhanced software package designed to provide easy-to-use, scalable, high-quality software for embedded system designs using the Renesas RA Family. See Renesas Web FSP site in detail.

Related documents

Document Type	Document Title	Document No.
Data Sheet	R-IN32M3 Module (RY9012A0) Datasheet	R19DS0109ED****
User's Manual	R-IN32M3 Module (RY9012A0) User's Manual: Hardware	R19UH0122ED****
User's Manual	R-IN32M3 Module (RY9012A0) User's Manual: Software	R17US0002ED****
Application Note	R-IN32M3 Module (RY9012A0) Management Tool Instruction Guide	R30AN0390EJ****
Application Note	RA Sample Application (uGOAL Edition)	R30AN0398EJ****

1. Introduction

This document describes an implementation guide for host CPU sample software that controls R-IN32M3 Module.

In Chapter 2, an overview of the sample software and the implementation guideline of user application are described.

In Chapter 3, the implementation guidelines of hardware driver layer are described including hardware design guides for using R-IN32M3 modules.

In Chapter 4 to 6, the vendor ID and device ID (call as Device Identity) that are common set for all protocols, and the data and its size (call as Data Model) that handled according to the protocol are described separately for each protocol. In addition, a description example of the following device definition file used when communicating with the master device are described.

Device definition file paired with each protocol:

- PROFINET: GSDML file
- EtherNet/IP: EDS file
- EtherCAT: ESI file

The following description has examples of program files and device definition files. Those are explained using a sample RA microcomputer, which is one of the host CPUs that control the R-IN32M3 module, as an example.

2. Sample Software Overview

2.1 Overall

This sample software is equipped with uGOAL middleware and is structured based on its design philosophy. Figure 2-1 shows the configuration diagram of the uGOAL system.

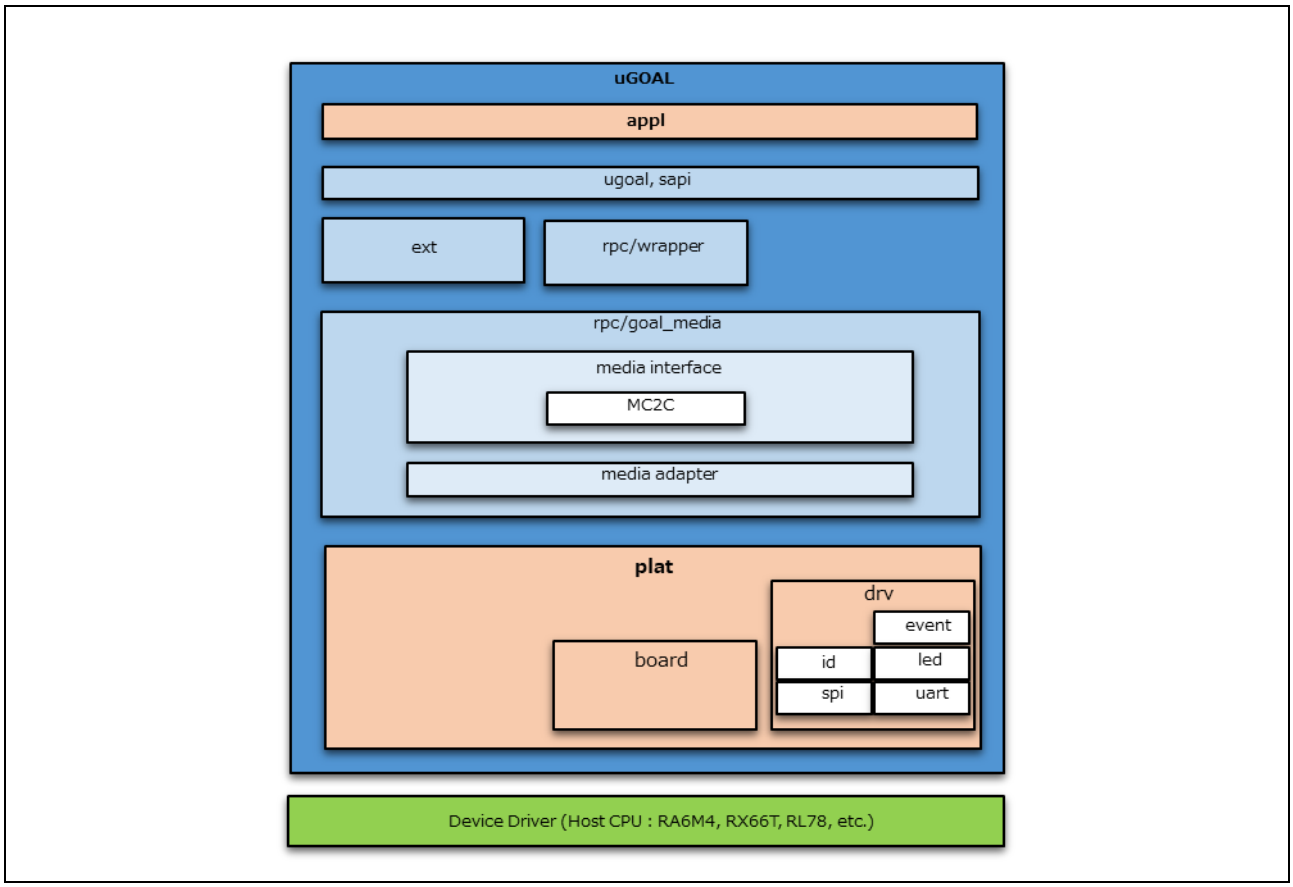

Figure 2-1 SW Configuration of uGOAL

For more information about uGOAL, see “R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED****)”.

2.1.1 Folder structure

The folder structure of this sample software is shown below.

RA6_uCCM_V***	
├──appl	User application
│ ├──01_pnio	PROFINET sample application
│ ├──02_eip	EtherNet/IP sample application
│ ├──03_ecat	EtherCAT sample application
│ ├──04_pnio_largesize	PROFINET Large data size sample application
│ ├──05_eip_largesize	EtherNet/IP Large data size sample application
│ ├──06_ecat_largesize	EtherCAT Large data size sample application
│ ├──07_modbus_tcp_slave	Modbus TCP sample application
│ ├──10_multi_protocol	multi-protocol [01_pnio, 02_eip, 03_ecat, 07_modbus] sample application
│ ├──11_pnio_http	c
│ ├──12_eip_http	02_eip sample Enhanced [web saver and host MCU update function]
│ └──13_ecat_http	03_ecat sample Enhanced [web saver and host MCU update function]
│	
├──plat	HW-dependent components (OS-dependent part, board spec, drivers)
├──projects	Project files corresponding to each user application
│	
└──ugoal	Main part of uGOAL (Generic Open Abstraction Layer *)
├──rpc	Functional parts related to RPC (Remote Procedure Call) including NW protocols and MCTC
├──sapi	Simple API
└──ext	external software component

* For more information about uGOAL, see “R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED****)”.

2.1.2 User Cording part

The following parts need to be modified by customer when they develop their products by use of the sample software.

Table 2-1 User cording part

appl	User application
plat	Board dependent parts and drivers Board dependent: Init and setting Driver: Init and setting for I2C, SPI, UART

Table 2-2 Independent part (No need cording)

uGOAL middleware	
ugoal	Main part of uGOAL
rpc	Functional parts related to RPC including NW protocols and MCTC
sapi	Simple API
Ext	external software component

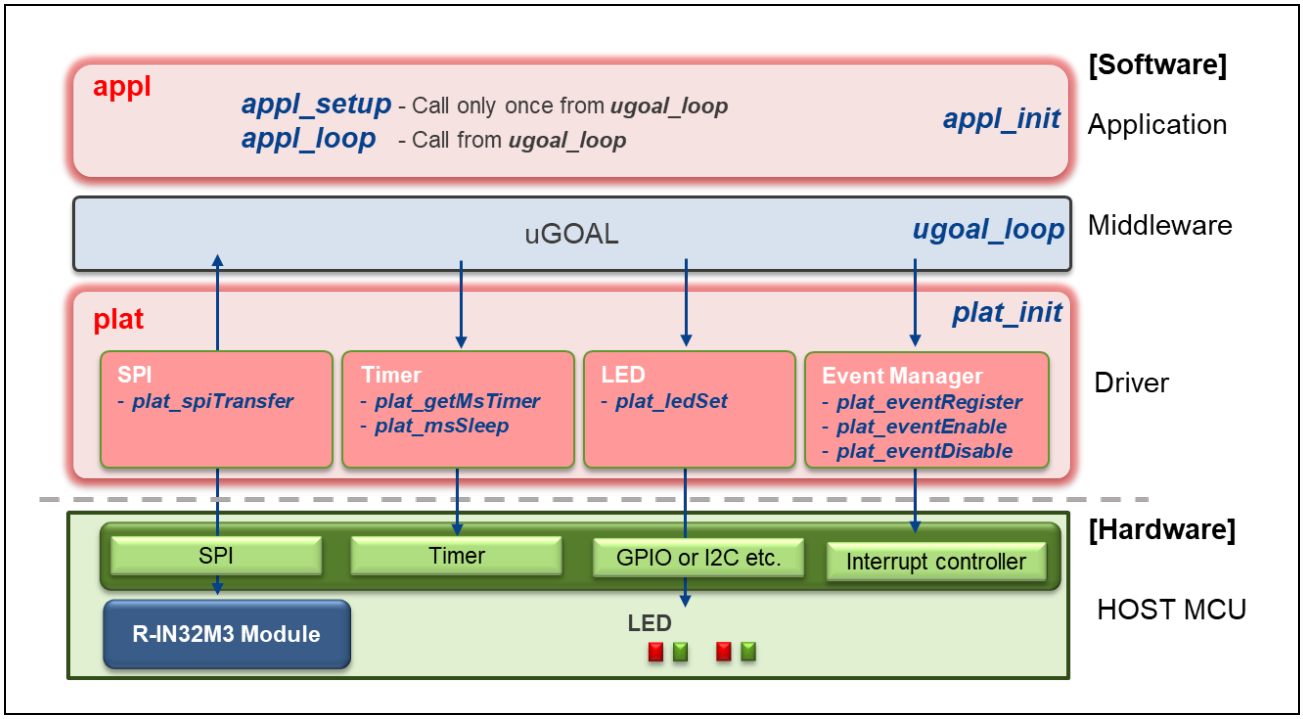

Figure 2-2 software structure

2.1.3 General Flow

The uGOAL general flow is below. In the main routine, plat_init(), som_init() and appl_init() is called as the initialization process and ugoal_loop() is called as the loop process.
When embedding the uGOAL system in the user's program, implement these three functions in the initialization process and the ugoal_loop() function in the loop process.

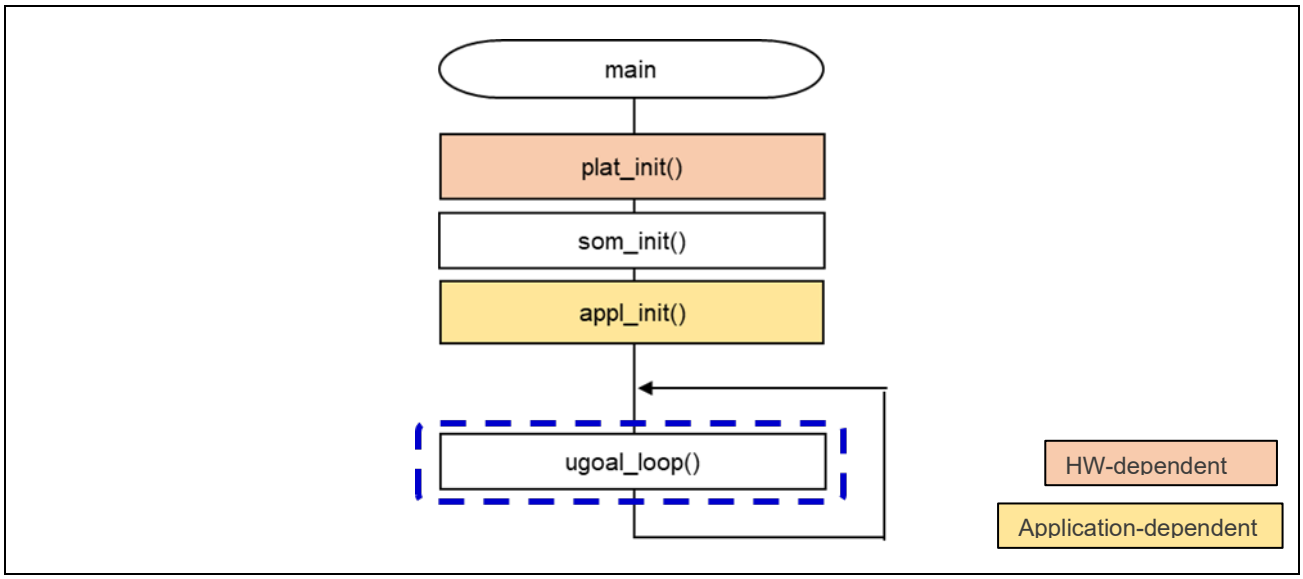

Figure 2-3 uGOAL flow

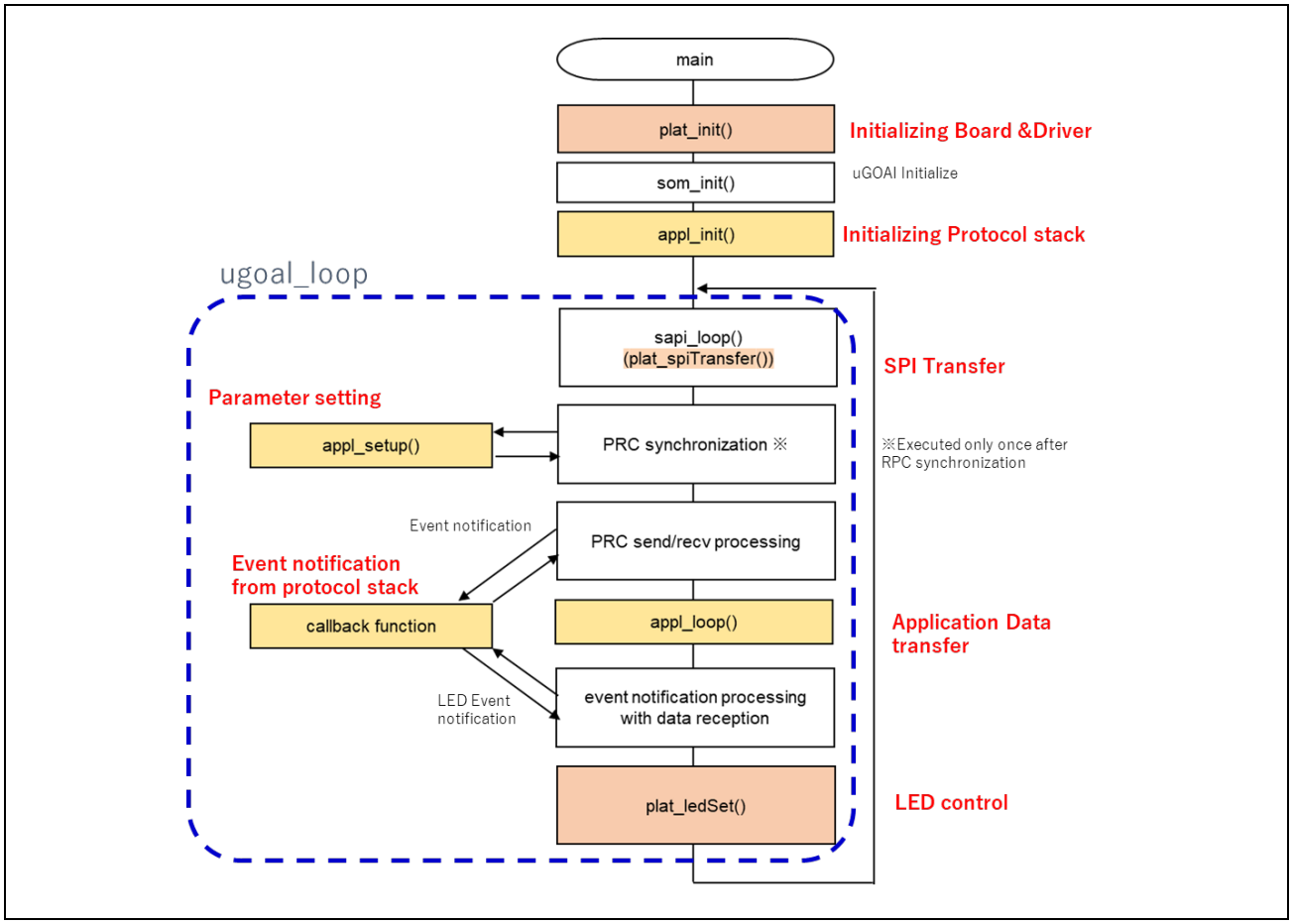

Figure 2-4 uGOAL flow (detailed)

As shown in Figure 2-4, the required user implementation parts called from the initialization process and the loop process are below.

Table 2-3 Required user implementation part

User implementation parts	File	uGOAL functions to call
appl_init() (refer to 2.2.1, 2.3.1 and 2.4.1) - Initialization the protocol stack and each module of uGOAL.	goal_appl.c	Initialization processing
appl_setup() (refer to 2.2.2, 2.3.2 and 2.4.2) - Setting the protocol profile (vendor ID etc.) and the Input / Output.		ugoal_loop()
appl_loop() (refer to 2.2.3, 2.3.3 and 2.4.3) - Loop processing called in the uGOAL loop period (about 1 ms).		
callback function (refer to 2.2.3, 2.3.3 and 2.4.3) - Processing according to the event for each protocol.		
plat_init() (refer to 3.1(a)) - Initialization of hard-dependent part. In this function, the open processing (make it available) of each driver is required to call.	ra6m4ek \ plat.c	Initialization processing
plat_spiTransfer() (refer to 3.2.3.2(1)(a)) - SPI communication called in the uGOAL loop period (about 1 ms).	ra6m4ek \ plat.c	ugoal_loop()
plat_ledSet() (refer to 3.3.1.1(2)(a)) - LED update via I2C called in the uGOAL loop period (about 1 ms).		

uGOAL provides appl_init(), appl_setup(), appl_loop(), and protocol-specific callback functions for user applications functions.
As shown in Figure 2-4, in the initialization process, appl_init() is called. In ugoal_loop(), appl_setup () is called once, appl_loop () is called periodically, and the callback function is called depending on the event for each protocol.

The user applications for each Industrial Ethernet Protocol are described below.

2.2 PROFINET User Application

This section describes the PROFINET sample application.

Target samples are shown in Table 2 2.

Table 2-4 PROFINET Sample software

Sample software	Overview
01_pnio	Cyclic communication sample
04_pnio_largesize	Cyclic and RPC communication sample
10_multi_protocol	Multi-protocol sample: 01_pnio, 02_eip, 03_ecat, 07_modbus
11_pnio_http	01_pnio sample Enhanced [web saver and host MCU update function]

2.2.1 appl_init() - Initialization

Initialize PROFINET protocol stack and each uGOAL module to be used. There is no need for the user to add processing.

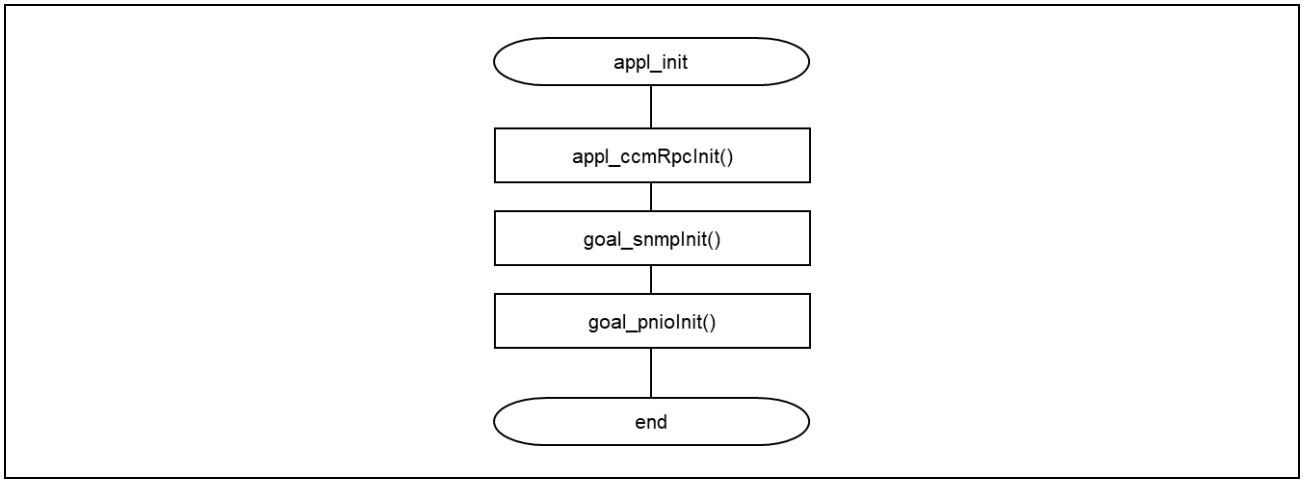

Figure 2-5 PROFINET appl_init() flow

2.2.2 appl_setup() – Parameter setting

Initialize LED setting, configure profile settings for each protocol stack, such as vendor ID settings. Also configure Input / Output data settings. Therefore, implement these settings according to the user specification by referring to Chapter 4.1 and 4.2.

Also, prepare for the event notified from the protocol stack by registering the callback function (appl_pnioCb()) in the goal_pnioNew().

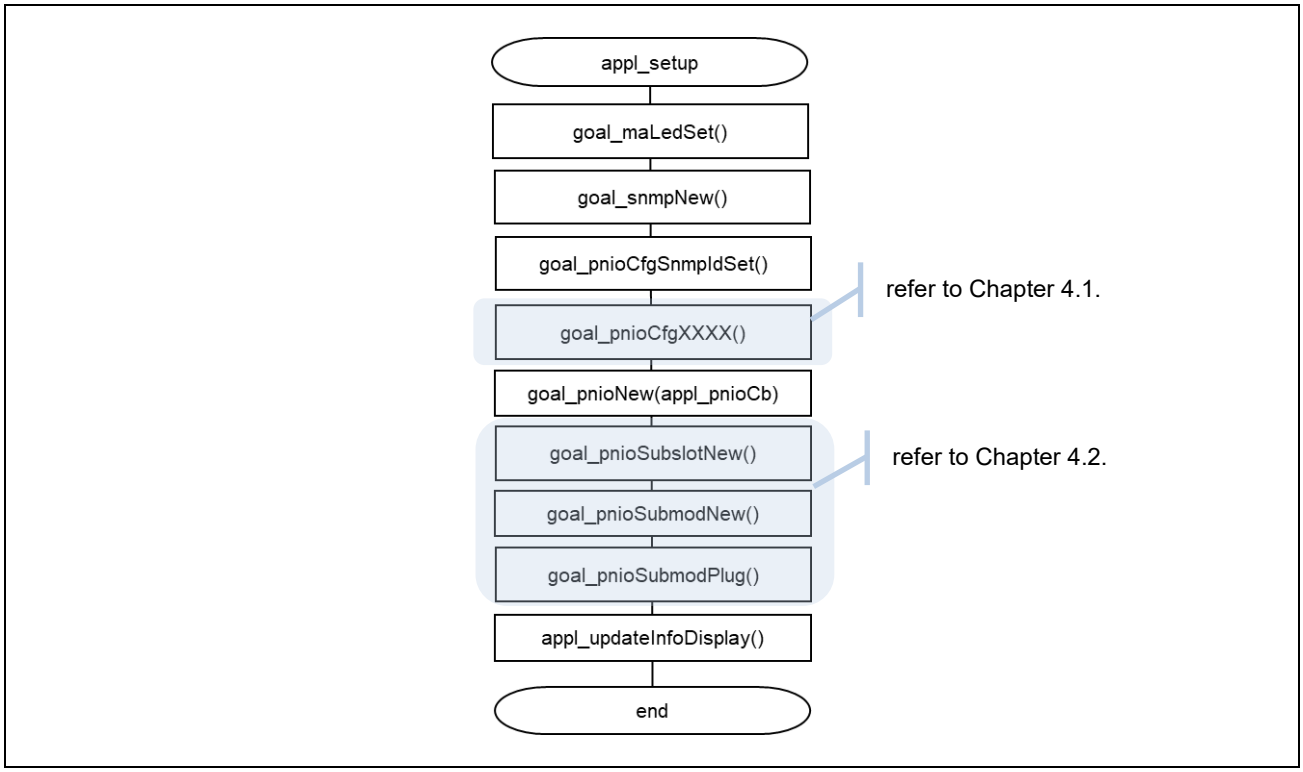

Figure 2-6 PROFINET appl_setup() flow

2.2.3 appl_loop()– Data transmission / reception

In appl_loop(), as the mirror response by cyclic communication, the acquired Output data is set to Input data as it is. And the update cycle of each data is set 1ms*. Therefore, implement these settings according to the user specification by referring to Chapter 4.3.

[* Note] 1 ms cycle is based on the 01_pnio sample. The update cycle should be considered depending on the data size to be handled. See also: C-Data Size Limitation

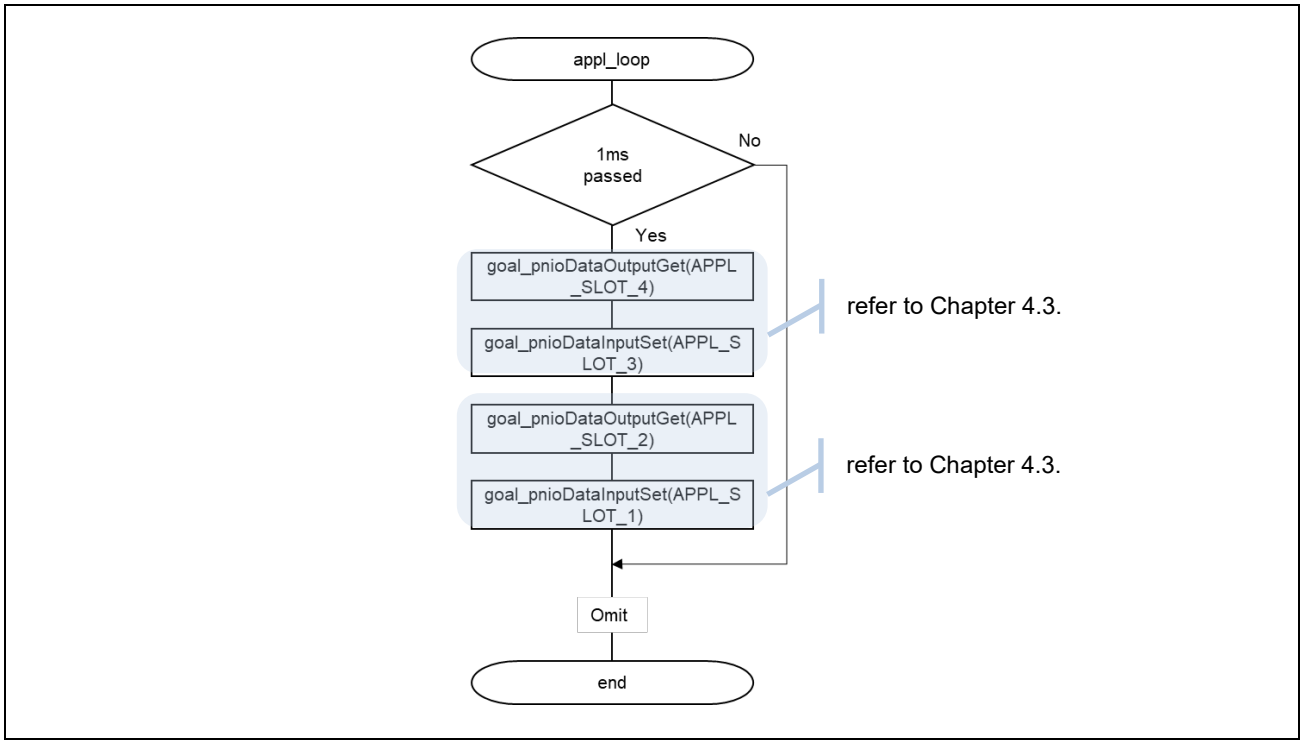

Figure 2-7 PROFINET appl_loop() flow

2.2.4 callback

In the callback function (appl_pnioCb()), execute the processing according to several states acquired from the protocol stack. Implement any process according to the user specification.

For more information about the status reported from the protocol stack, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

As one of the processing according to several states, shape LED status(see Chapter 3.3). This shaped status is used as an argument of plat_ledSet() and updates the state of each LED when this function is called.

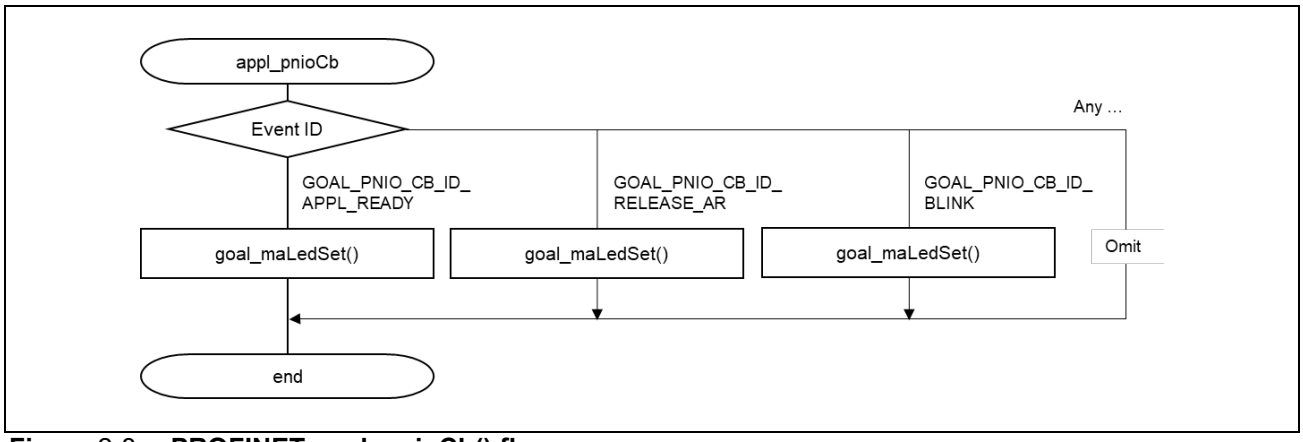

Figure 2-8 PROFINET appl_pnioCb() flow

2.3 EtherNet/IP User Application

This section describes the EtherNet/IP sample application.

Target samples are shown in Table 2-5.

Table 2-5 EtherNet/IP Sample software

Sample software	Overview
02_eip	Cyclic communication sample
05_eip_largesize	Cyclic and RPC communication sample
10_multi_protocol	Multi-protocol sample: 01_pnio, 02_eip, 03_ecat, 07_modbus
12_eip_http	02_eip sample Enhanced [web saver and host MCU update function]

2.3.1 appl_init() - Initialization

Initialize EtherNet/IP protocol stack and each uGOAL module to be used. There is no need for the user to add processing.

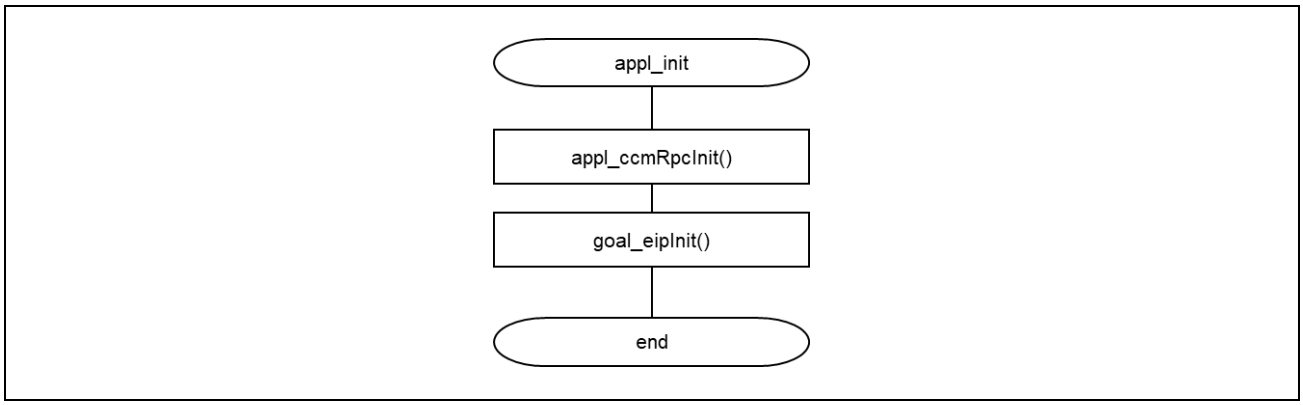

Figure 2-9 EtherNet/IP appl_init() flow

2.3.2 appl_setup() – Parameter setting

Initialize LED setting, configure profile settings for each protocol stack, such as vendor ID settings. Also configure Input / Output data settings. Therefore, implement these settings according to the user specification by referring to Chapter 5.1 and 5.2.

Also, prepare for the event notified from the protocol stack by registering the callback function (main_eipCallback ()) in the goal_eipNew().

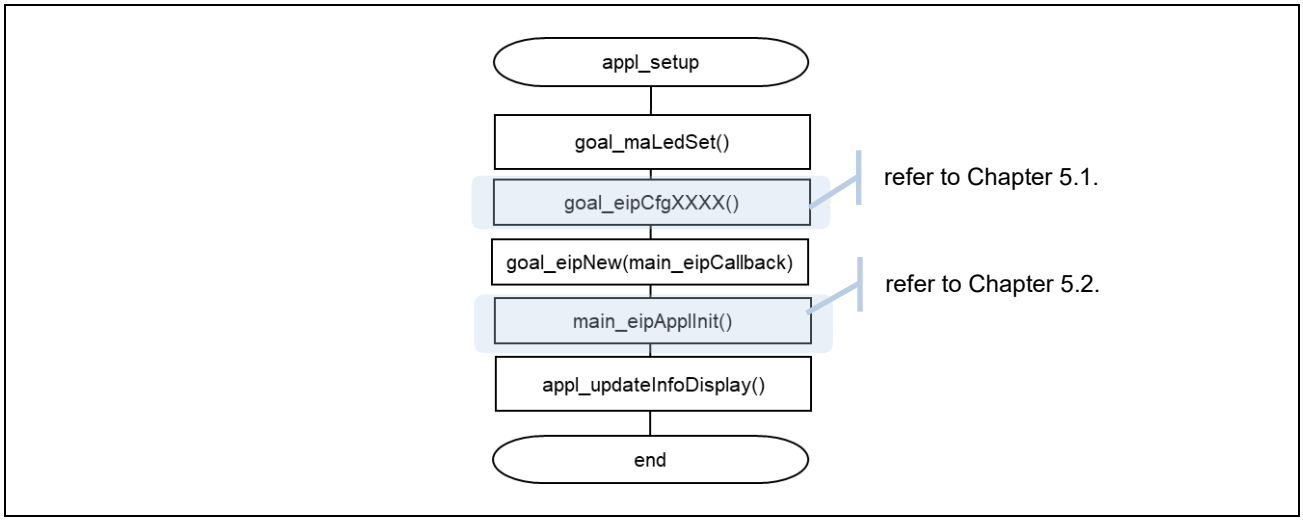

Figure 2-10 EtherNet/IP appl_setup() flow

2.3.3 appl_loop()

In appl_loop(), as the mirror response by cyclic communication, the acquired Output data is set to Input data as it is. And the update cycle of each data is set 1ms*. Therefore, implement these settings according to the user specification by referring to Chapter 5.3.

[* Note] 1 ms cycle is based on the 02_eip sample. The update cycle should be considered depending on the data size to be handled. See also: C-Data Size Limitation

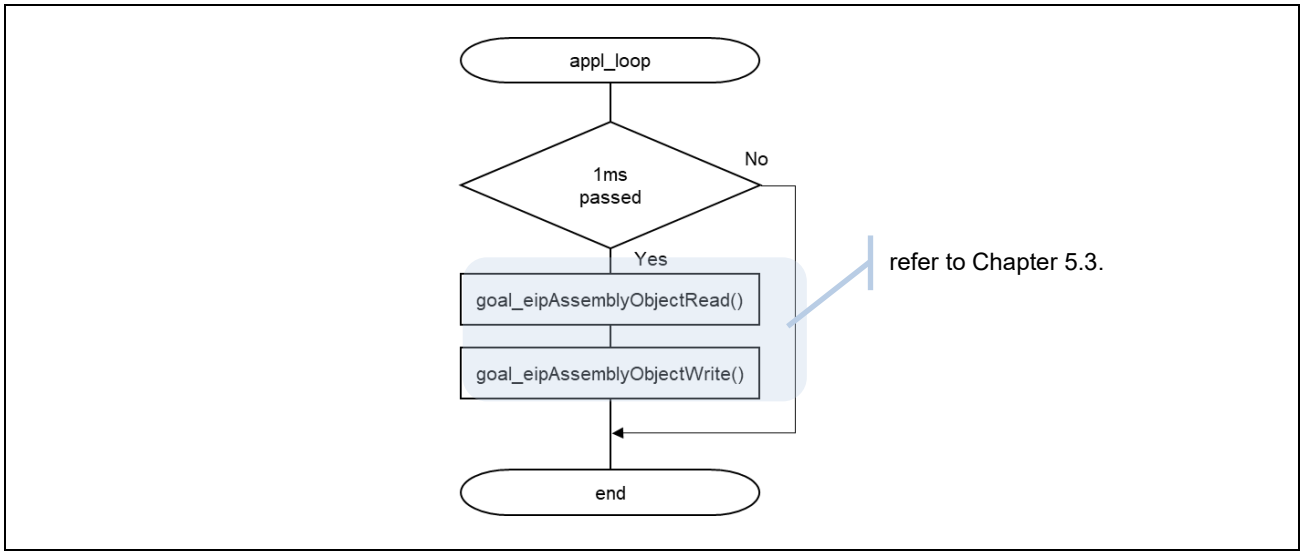

Figure 2-11 EtherNet/IP appl_loop() flow

2.3.4 callback

In the callback function (main_eipCallback()), execute the processing according to several states acquired from the protocol stack. Implement any process according to the user specification.

For more information about the status reported from the protocol stack, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

As one of the processing according to several states, shape LED status(see Chapter 3.3). This shaped status is used as an argument of plat_ledSet() and updates the state of each LED when this function is called.

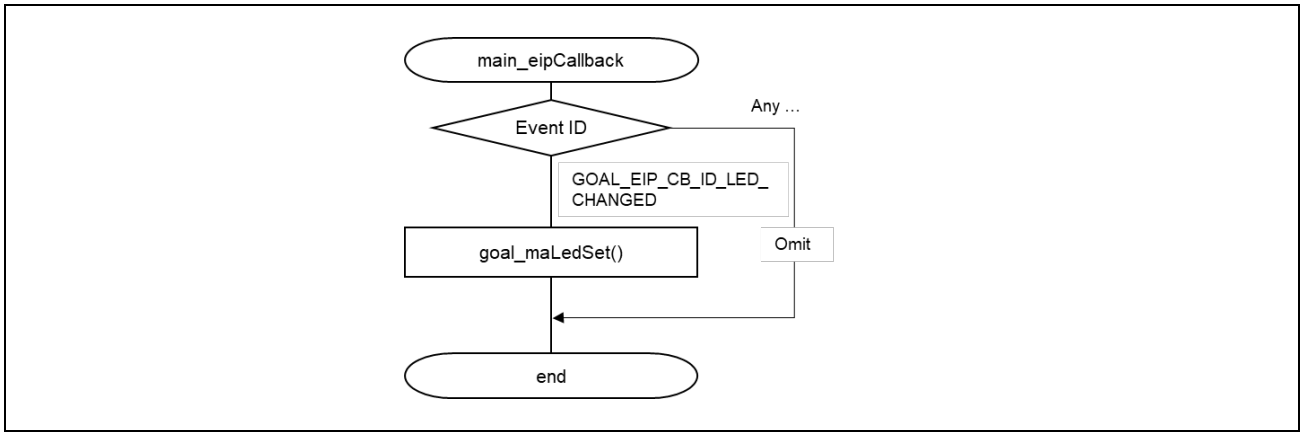

Figure 2-12 EtherNet/IP main_eipCallback() flow

2.4 EtherCAT User Application

This section describes the EtherCAT sample application.

Target samples are shown in Table 2-6.

Table 2-6 EtherNet/IP Sample software

Sample software	Overview
03_ecat	Cyclic communication sample
06_ecat_largesize	Cyclic and RPC communication sample
10_multi_protocol	Multi-protocol sample: 01_pnio, 02_eip, 03_ecat, 07_modbus
13_ecat_http	03_ecat sample Enhanced [web saver and host MCU update function]

2.4.1 appl_init() - Initialization

Initialize EtherCAT protocol stack and each uGOAL module to be used. There is no need for the user to add processing.

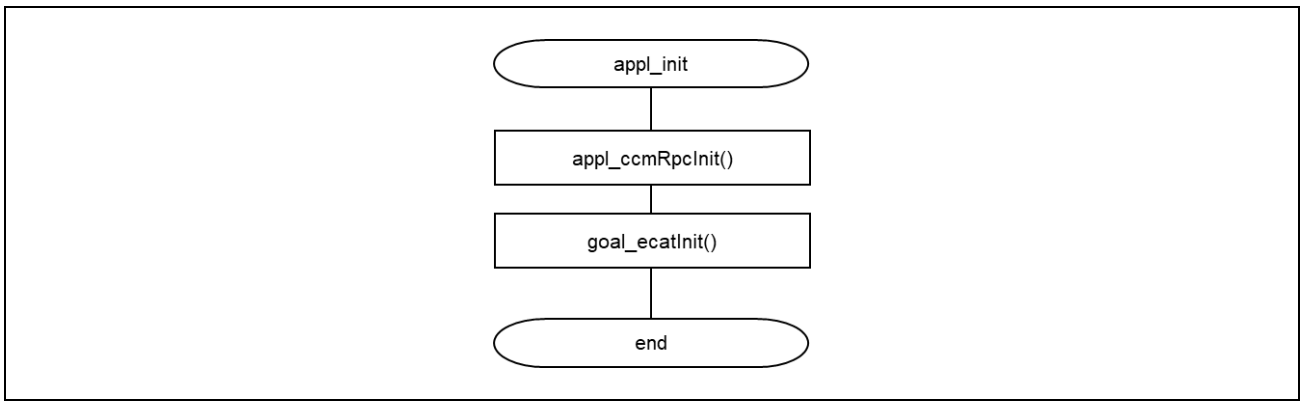

Figure 2-13 EtherCAT appl_init() flow

2.4.2 appl_setup() – Parameter setting

Initialize LED setting, configure profile settings for each protocol stack, such as vendor ID settings. Also configure Input / Output data settings. Therefore, implement these settings according to the user specification by referring to Chapter 6.1 and 6.2.

Also, prepare for the event notified from the protocol stack by registering the callback function (appl_ecatCallback ()) in the goal_ecatNew().

For more, by calling goal_maEventOpen() to enable IRQ interrupts for DC (Distributed Clock) functionality.

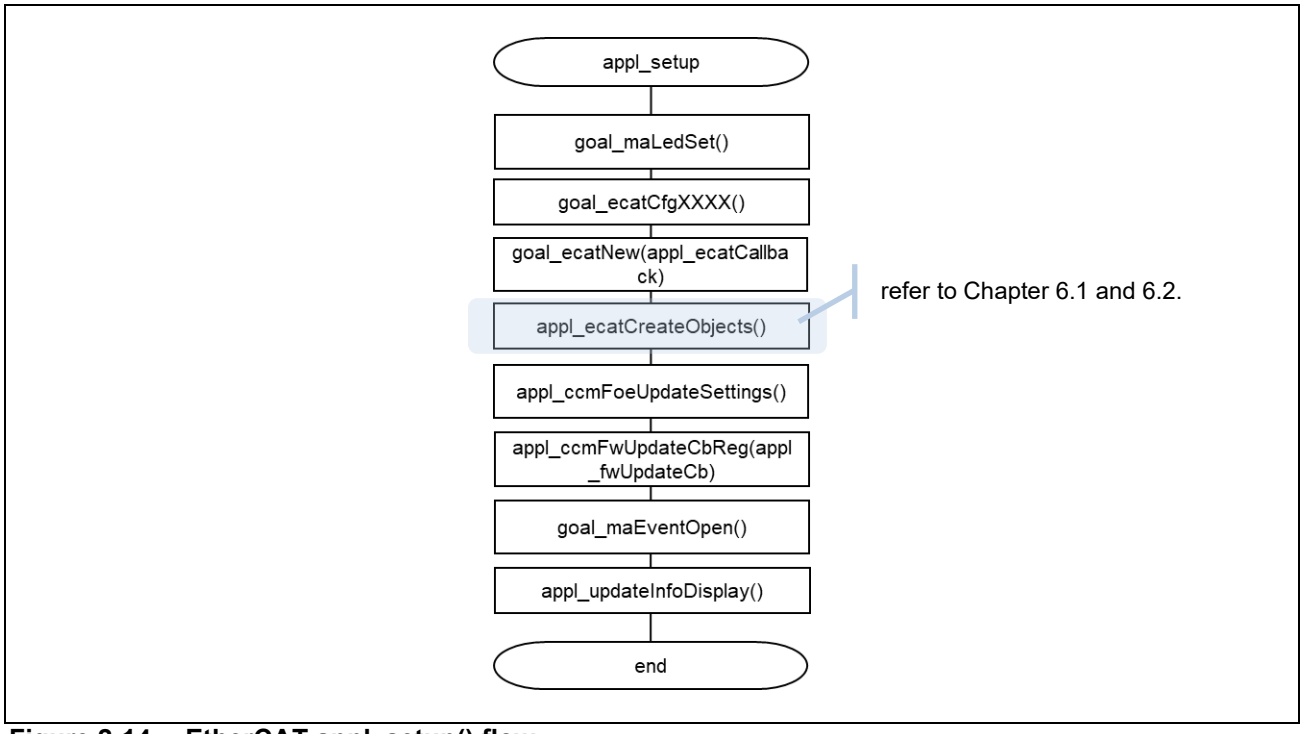

Figure 2-14 EtherCAT appl_setup() flow

2.4.3 appl_loop()– Data transmission / reception

In appl_loop(), as the mirror response by cyclic communication, the acquired Output data is set to Input data as it is. And the update cycle of each data is set 1ms*. Therefore, implement these settings according to the user specification by referring to Chapter 6.3.

[* Note] 1 ms cycle is based on the 03_ecat sample. The update cycle should be considered depending on the data size to be handled. See also: C-Data Size Limitation

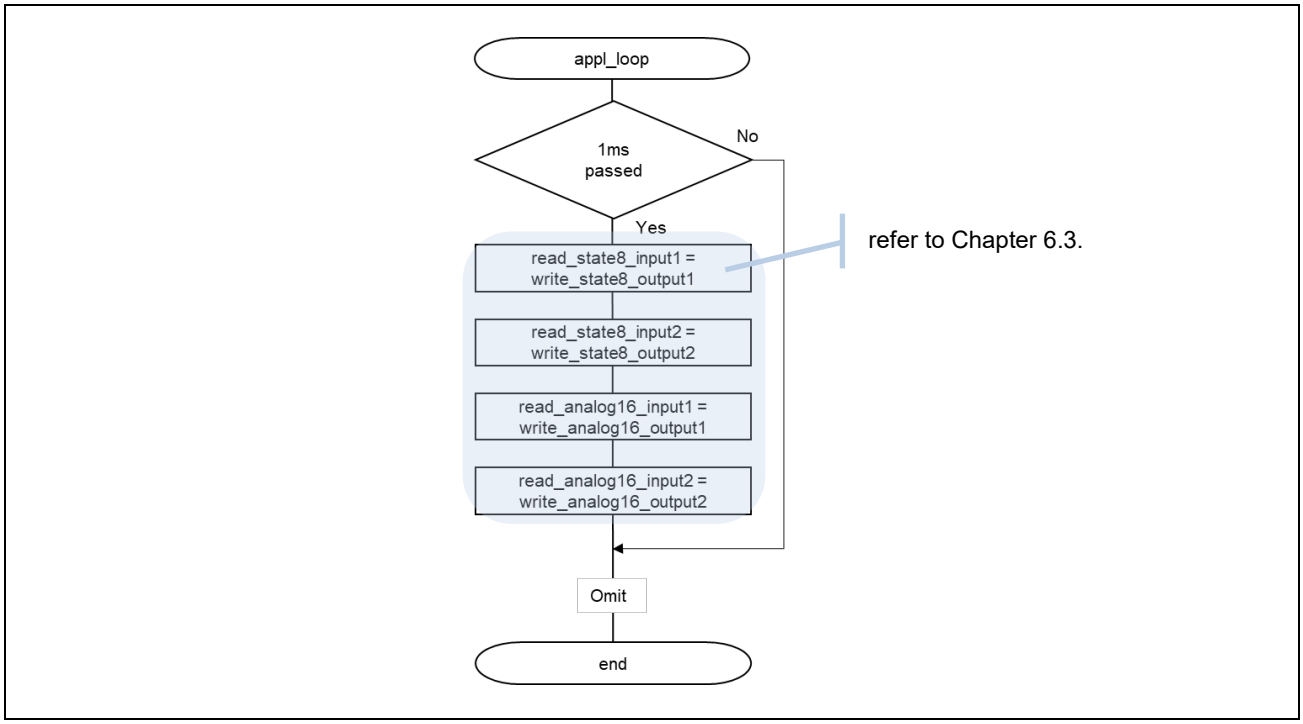

Figure 2-15 EtherCAT appl_loop() flow

2.4.4 callback

In the callback function (appl_ecatCallback()), execute the processing according to several states acquired from the protocol stack. Implement any process according to the user specification.

For more information about the status reported from the protocol stack, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

As one of the processing according to several states, shape LED status(see Chapter 3.3). This shaped status is used as an argument of plat_ledSet() and updates the state of each LED when this function is called.

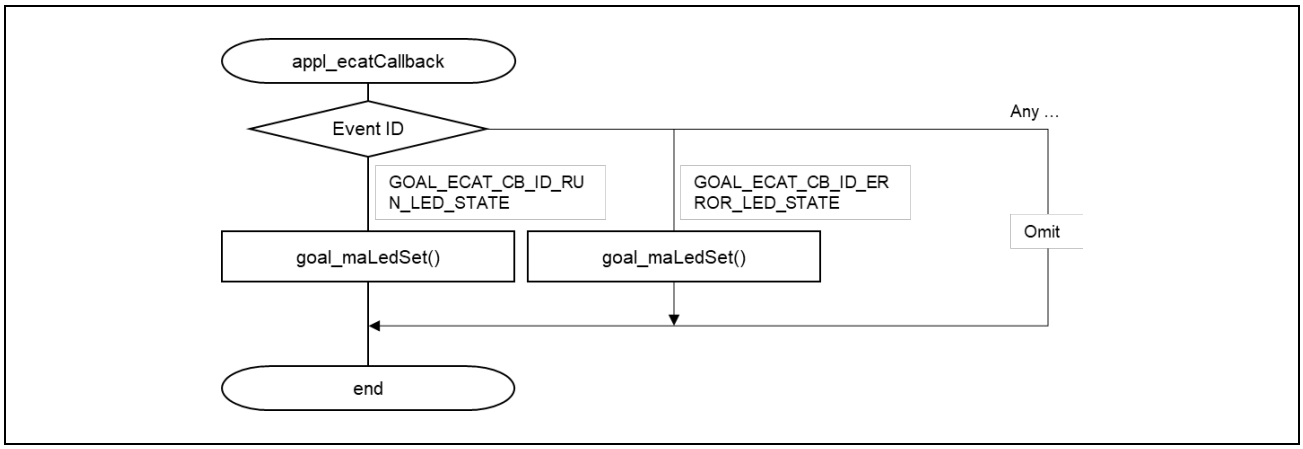

Figure 2-16 EtherCAT appl_ecatCallback() flow

3. Hardware Layer

This chapter describes implementation guidelines of hardware driver layer, including hardware design guides for using R-IN32M3 Module.
For more information about recommended circuits and layout notes of hardware, see “R-IN32M3 Module (RY9012A0) User's Manual Hardware (R19UH0122ED**)”.

Example of each implementation is described below.

3.1 Board Initialization and Driver Initialization

Initialize the hardware settings according to the specifications of the board equipped with the host CPU.

uGOAL provides API for initialization of hard-dependent parts (*plat_init*). Add the initialization process required for the host CPU, such as terminal settings of the host CPU. Also, release the reset of R-IN32M3 Module with the API.

Also, initialize the I2C, SPI, and UART drivers that are shown below.

The following is an implementation example of the sample software for RA6M4.

Target File：

plat\ra6m4ek\plat.c

Target API：

(a) plat_init

- **Initialization of peripheral modules**

  In the sample software for RA6M4 sample, since the peripheral modules are managed by the Smart Configurator which is added-on e2studio, there is no need for the user to initialize them.

- **I/O port settings**

  In the sample software for RA6M4 sample, since the I/O ports are managed by the Smart Configurator which is added-on e2studio, there is no need for the user to initialize them.

- **Release Reset signal of R-IN32M3 Module**
  In the following code, the reset is released by controlling I/O port allocated for reset signal.

```
19.     /* reset AC */
20.     platResetPeer(NULL);
```

```
19.  static GOAL_STATUS_T platResetPeer(
20.      struct GOAL_MI_MCTC_INST_T *pMiMctc        /**< MI MCTC instance */
21.  )
22.  {
23.      UNUSEDARG(pMiMctc);
24.
25.      g_ioport.p_api->pinWrite(g_ioport.p_ctrl, GOAL_IEM_RST_PIN, GOAL_IEM_RST_SELECT);
26.      g_ioport.p_api->pinWrite(g_ioport.p_ctrl, GOAL_IEM_RST_PIN, GOAL_IEM_RST_DESELECT);
27.  #if GOAL_CONFIG_RESET_DELAY == 1
28.      {
29.          bsp_io_level_t value;
30.          do
31.          {
32.              g_ioport.p_api->pinRead(g_ioport.p_ctrl, GOAL_IEM_RST_PIN, &value);
33.          }
34.          while (value != GOAL_IEM_RST_DESELECT);
35.      }
36.  #endif
37.
38.      return GOAL_OK;
39.  }
```

- **Set up system timer**
  Channel 0 of the general-purpose PWM timer (GPT) module built into Smart Configurator has started to operate as a system timer. In this sample software, ***goal_tgtCommonSystickHandler*** is set to be called every 1ms as a callback function.

```
1.      /* start timer */
2.      g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
3.      g_timer0.p_api->start(g_timer0.p_ctrl);
```

- **Set up I2C, SPI and UART drivers**
  Open I2C, SPI and UART channels registered on Smart Configurator and make them available.

```
1.      /* I2C */
2.      g_i2c_master0.p_api->open(g_i2c_master0.p_ctrl, g_i2c_master0.p_cfg);
3.      g_i2c_master0.p_api->abort(g_i2c_master0.p_ctrl);
4.
5.      /* SPI */
6.      g_spi0.p_api->open(g_spi0.p_ctrl, g_spi0.p_cfg);
7.
8.      /* UART */
9.      g_uart0.p_api->open(g_uart0.p_ctrl, g_uart0.p_cfg);
```

3.2 Connection with R-IN32M3 Module

The connection between the R-IN32M3 Module and the host CPU is described in this chapter.

3.2.1 Host CPU Specifications

The recommended specifications of the host CPU are as follows.

ROM capacity: More than 64 KB

RAM capacity: More than 32 KB

SPI packet transfer size: 128 bytes (8 bits x 128 times) bulk data transfer

---

<Note>

- Memory capacity requirements vary by development environment, microcomputer platform, and compiler.
- When using a Renesas CPU, select the Simple SPI (SCI), as the maximum batch data transfer amount of RSPI is 32 bytes.

3.2.2 Examples of hardware connections

Figure 3-1 shows an example of connecting the R-IN32M3 Module to the host CPU.

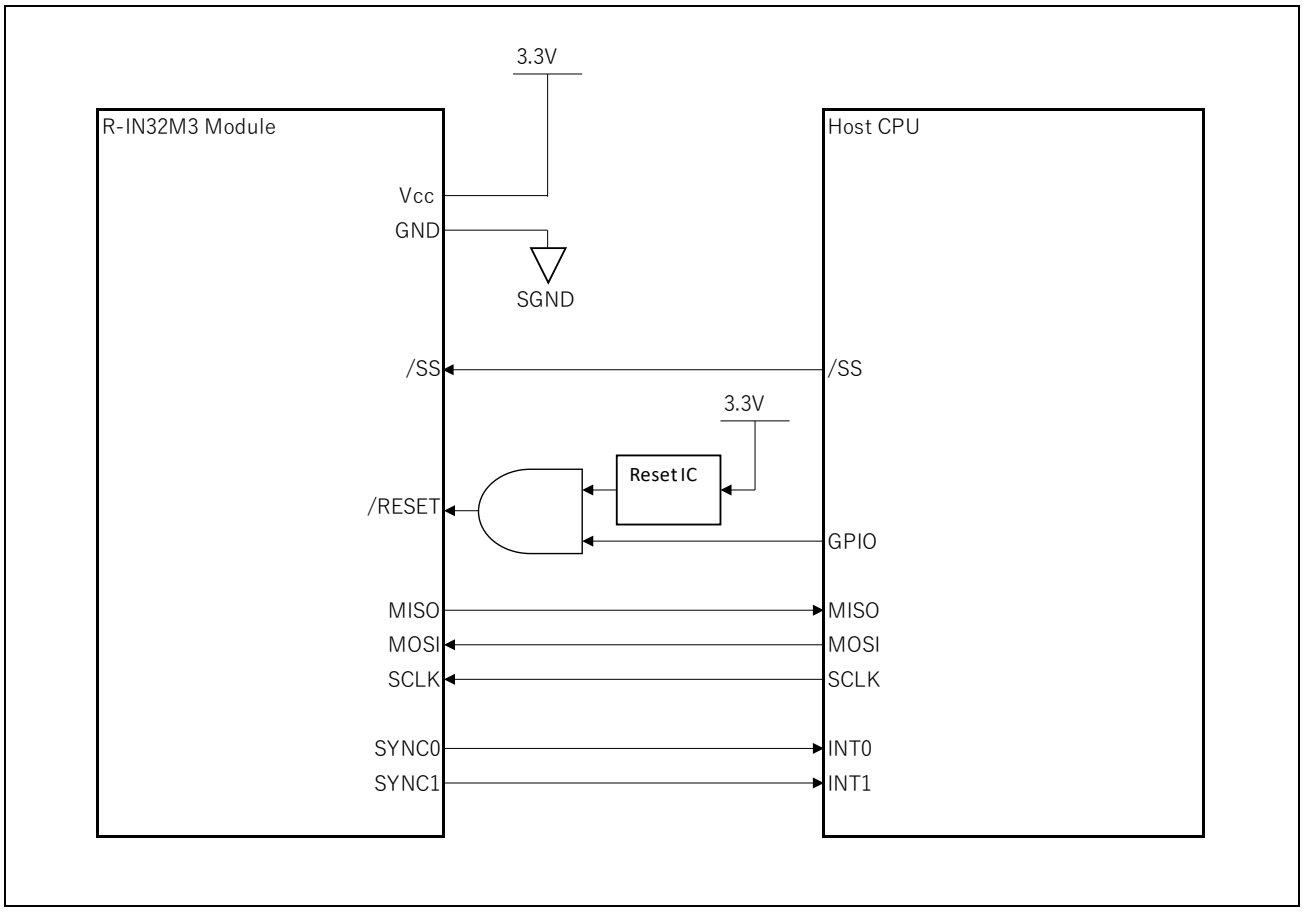

Figure 3-1 Connection to a Host CPU

3.2.2.1 Module Pins

The R-IN32M3 Module pins are for the interface with the power supply, the SPI, which is a slave interface, applying a reset, and clock signals.

Table 3-1 Pin Description

Pin	Signal	I/O	Description
1	V_{CC}	—	3.3V ±0.15V DC power supply
2	GND	—	Ground
3	/SS	I	Slave select: Active low to enable the slave device
4	/RESET	I	Reset of the whole R-IN32M3 Module: Active low
5	MISO	O	Master in slave out. Data from slave to master
6	MOSI	I	Master out slave in. Data from master to slave
7	SCLK	I	Serial clock: The master provides the clock to shift the data.
8	SYNC0	O	EtherCAT sync signal for distributed clocks
9	SYNC1	O	EtherCAT sync signal for distributed clocks

Note. Pin 8 and pin 9 EtherCAT sync signals for distributed clocks are only used for EtherCAT protocol.

Internal circuit of each pin is shown in Figure 3-2.

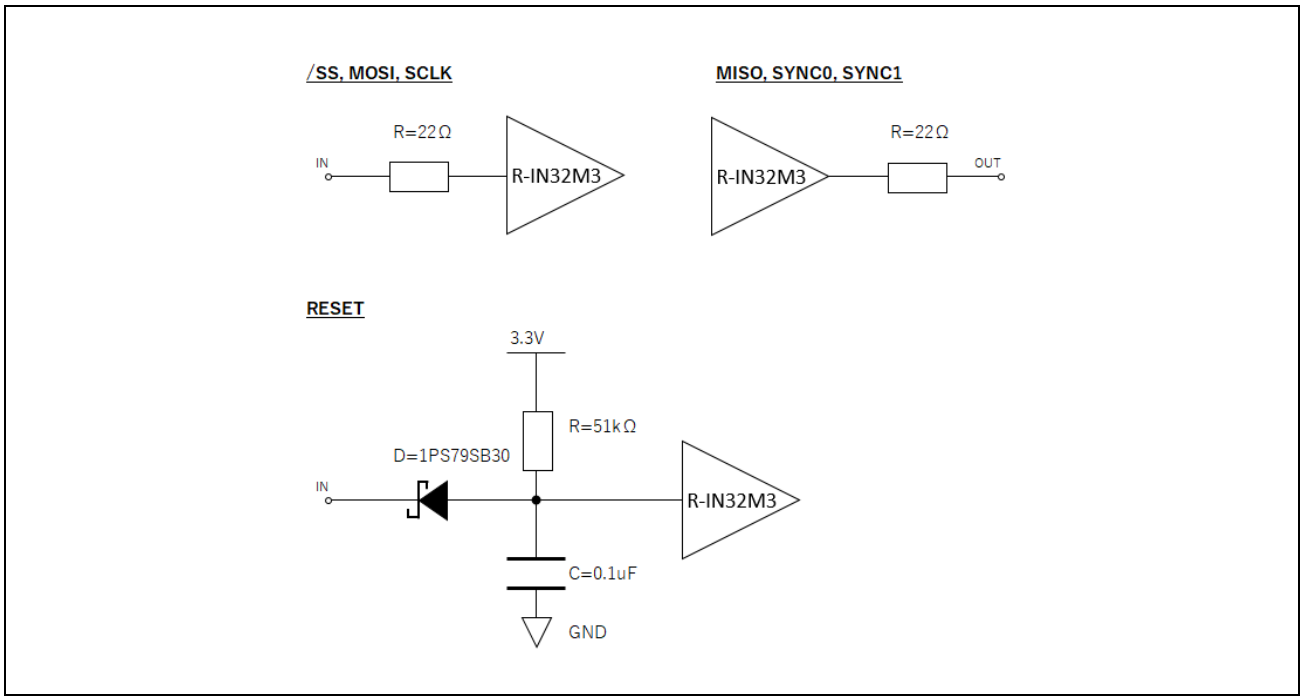

Figure 3-2 R-IN32M3 Module internal circuits

3.2.2.2 Power Supply and Reset

The recommended sequence of supplying and cutting off power to the R-IN32M3 Module and Reset timing are shown in Figure 3-3.

The supply voltage for the R-IN32M3 Module is specified with 3.3V DC ± 0.15V DC (3.15 V to 3.45 V). Since the R-IN32M3 Module has a maximum power consumption of about 2.0 W, it is recommended that the external power be capable of supplying 1.0 A (or more).

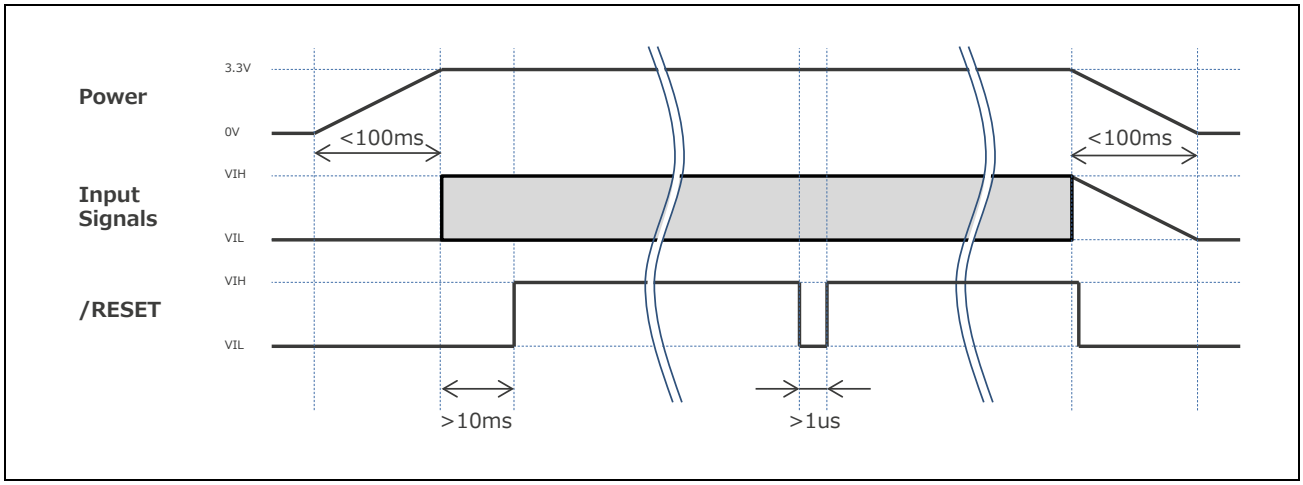

Figure 3-3 Power and Reset sequence

Since the reset pin has a built-in pull-up resistor inside the module (Figure 3-2), there is no need to place an external pull-up resistor. Figure 3-4 is a reference circuit to satisfy the reset sequence in Figure 3-3. Here, /RESET signal from host CPU is connected to a 3.3V supervisor (ISL88011H529Z-TK) and set as open drain output port (NCODR=1). For ISL88011H529Z-TK, enable the compile macro of *GOAL_CONFIG_RESET_DELAY* to wait for /RESET of the R-IN32M3 Module to be released.

The host CPU release reset signal in the board initialization process *(plat_init*). See section 3.1(a) in detail.

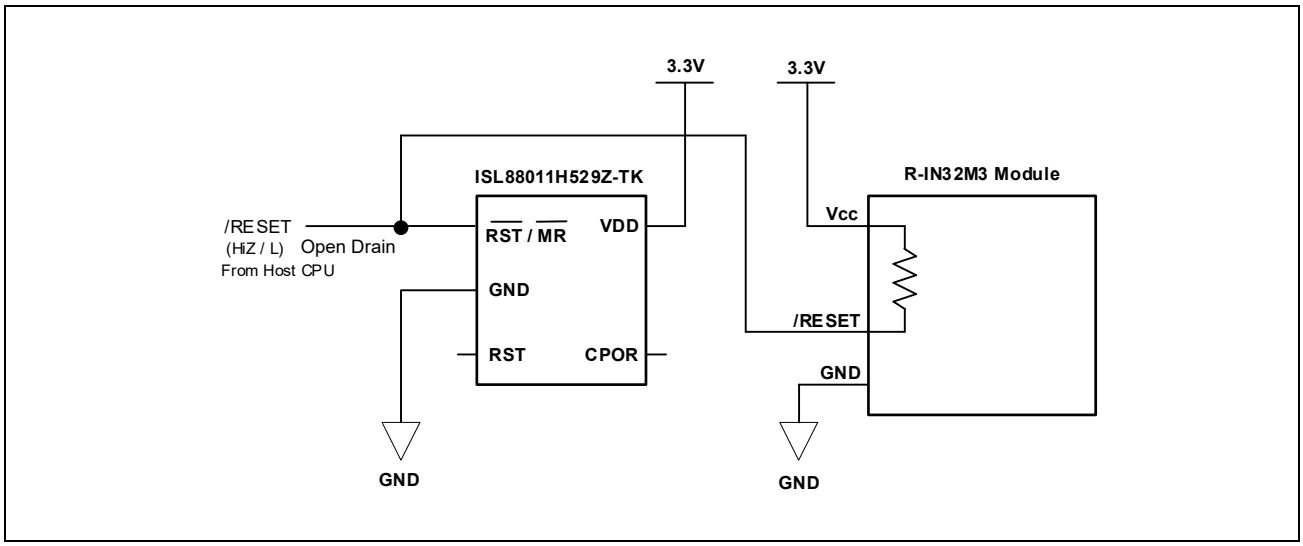

Figure 3-4 Reset Circuit Example (ISL88011H529Z-TK)

3.2.2.3 Layout Design Guideline

Figure 3-5 shows the footprints needed to implement the R-IN32M3 Module.

- ✓ Gray areas indicate through holes and red areas indicate land patterns.
- ✓ The R-IN32M3 Module mounting surface should be solid ground except in the wiring lead-out parts.
- ✓ No restrictions apply to the inner layers of the P.C.B.
- ✓ The thickness of the P.C.B should be 1.6 mm. Any mounting of components on the back side of the
- ✓ R-IN32M3 Module is prohibited.

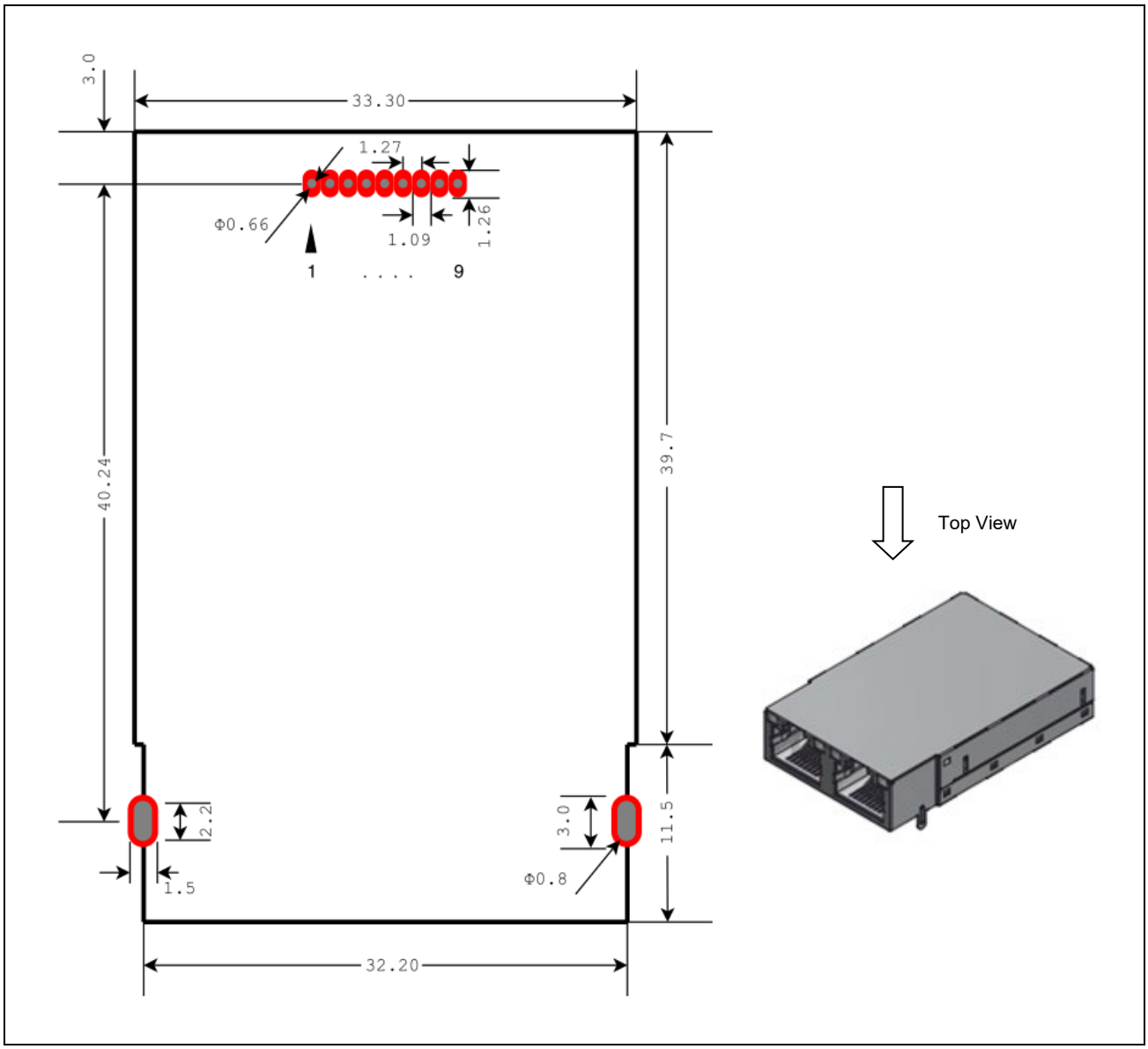

Figure 3-5 Footprint Drawing (Top View)

3.2.3 SPI

3.2.3.1 SPI Specification

The R-IN32M3 Module offers a serial peripheral interface which is supported on the R-IN32M3-EC controller. This communication interface uses 4 signal lines in Table 3-2.

The R-IN32M3 Module works always in slave mode and allow to transmit configuration and process data to the host CPU.

Table 3-2 SPI Signal Description

Signals	Description
SCLK	Serial clock input (output from master)
MOSI	Master output slave Input, or master out slave in (data output from master)
MISO	Master input slave output, or master in slave out (data output from slave)
/SS	Slave select (active low, output from master)

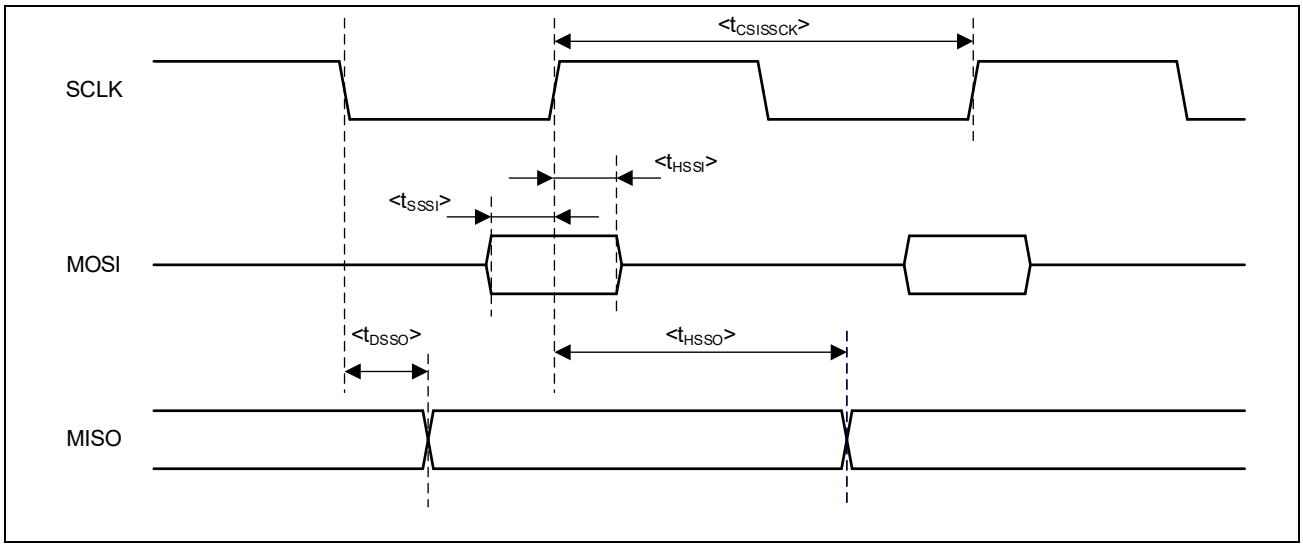

Figure 3-6 SPI Timing Chart

Table 3-3 SPI Specifications (V_{CC}＝3.3±0.15V、T_a＝−40～+70°C)

Parameter	Symbol	Conditions	Min.	Max.	Unit
SCLK input cycle	$t_{CSISSCK}$	—	60	—	ns
SCLK output high level width	t_{WSKH}	—	$t_{CSIMSCK}$×0.5−5.0	—	ns
SCLK output low level width	t_{WSKL}	—	$t_{CSIMSCK}$×0.5−5.0	—	ns
MOSI input setup time (to CSISCKn ↑)	t_{SSSI}	—	10	—	ns
MOSI input setup time (to CSISCKn ↓)	t_{SSSI}	—	10	—	ns
MOSI input hold time (from CSISCKn ↑)	t_{HSSI}	—	15	—	ns
MOSI input hold time (from CSISCKn ↓)	t_{HSSI}	—	15	—	ns
MISO output delay time (from CSISCKn ↑)	t_{DSSO}	CL=15pF	—	10	ns
MISO output delay time (from CSISCKn ↓)	t_{DSSO}		—	10	ns
MISO output hold time (from CSISCKn ↑)	t_{HSSO}		$t_{CSISSCK}$×0.5−5.0	—	ns
MISO output hold time (from CSISCKn ↓)	t_{HSSO}		$t_{CSISSCK}$×0.5−5.0	—	ns

3.2.3.2 SPI Communication

(1) Software Implement

SPI (serial peripheral interface) is used to communication with R-IN32M3 Module in the frame structure shown in Table 3-4. uGOAL provides a driver (plat/ra6m4ek) for controlling SPI communication and realizes the function by implementing a hardware control part in the driver.

When implementing the SPI communication feature, the driver needs to be designed to store data in a specified buffer in 128 bytes. (There are 128 bytes for sending buffer (sendData[]) and 128 bytes for receiving buffer (receiveData[]) for communication, these are used as arguments of plat_spiTransfer.)

Set /SS signal to low during 128 Byte SPI frame transfer. In this sample software, assert (/SS = Low) in plat_spiTransfer and negate (High) in goal_drvSpiSynCb .

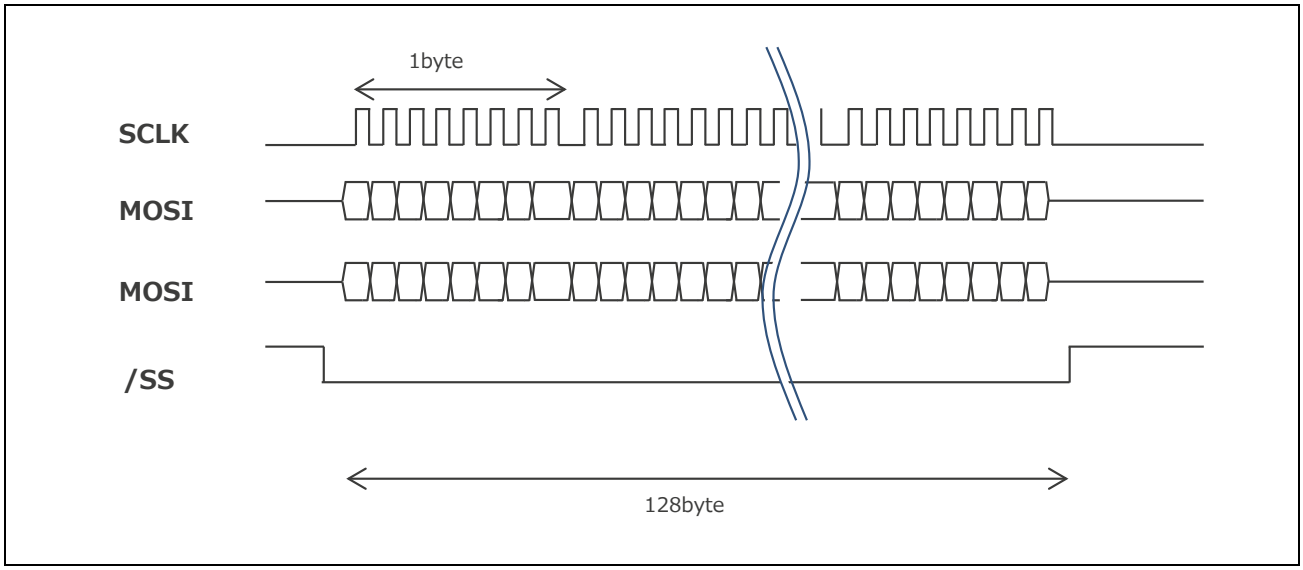

Figure 3-7 SPI Frame Timing

In Table 3-4, the Cyclic Data frame transfers periodic communication data updated every SPI frame. On the other hand, RPC Data frames are not updated for each SPI frame, but can transfer large amounts of data by dividing them. Therefore, RPC is mainly used for asynchronous communication, but can also be used for large-volume synchronous communication. (Appendix C)

Table 3-4 SPI Frame Structure

Bytes 0..1	Byte 2	Byte 3	Bytes 4..76	Bytes 77..127
Fletchar-16 Checksum with offset 0x0007 (little endian)	Sequence	Data length	Cyclic Data	RPC Data

SPI communication in this sample software is implemented in the form of incorporating the API of the SCI module into uGOAL's SPI communication control driver. The following is an implementation example for RA6M4.

For more information on SPI cycle, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED****)".

Target File：

plat\ra6m4ek\plat.c

Target API：

In this sample software, SCI module (r_spi) for SPI communication is incorporated from the Smart Configurator and controlled via an FSP driver.

(a) plat_spiTransfer

This API handles managing SPI communication. This process is executed by the loop cycle of uGOAL and SPI communication is started. In this sample software, **assert** operation of /SS pin and read/write operation of SPI communication is implemented by driver API of SCI module.

```
1.      /* Execute write */
2.      g_ioport.p_api->pinWrite(g_ioport.p_ctrl, GOAL_DRV_SPI_CS_PIN, GOAL_DRV_SPI_CS_SELECT);
3.
    … omit …
4.
5.      /* transfer */
6.      fsp_res = g_spi0.p_api->writeRead(g_spi0.p_ctrl,
7.                                        (uint8_t *)txBuf,
8.                                        (uint8_t *)rxBuf,
9.                                        size,
10.                                       SPI_BIT_WIDTH_8_BITS);
11.     if (fsp_res != FSP_SUCCESS)
12.     {
13.        /* disable CS */
14.        g_ioport.p_api->pinWrite(g_ioport.p_ctrl, GOAL_DRV_SPI_CS_PIN, GOAL_DRV_SPI_CS_DESELECT);
15.        return GOAL_ERR_SPECIFIC;
16.     }
```

(b) goal_drvSpiSynCb

This API is a callback function that is called when SPI communication is completed. In this sample software, **negate** operation of /SS pin is implemented by driver API of SCI module.

```
1.      /* Deselect CS signal */
2.      g_ioport.p_api->pinWrite(g_ioport.p_ctrl, GOAL_DRV_SPI_CS_PIN, GOAL_DRV_SPI_CS_DESELECT);
```

3.3 Hardware Control by Each Protocol (LED, ID Selector)

This chapter describes the circuit design requirements and software implementation for each industrial Ethernet protocol.

R-IN32M3 Module has an RJ45 female connector with two indicator LEDs (RJ45-LED 0G,0Y,1G,1Y) on each Ethernet port (**Figure 3.8**). The green LED indicates the link status and the yellow LED lights up in response to network activity. (See Table 3-9 in EtherCAT)

These two indicator LEDs are controlled by the R-IN32M3 Module side, so there is no need to design them on the user side.

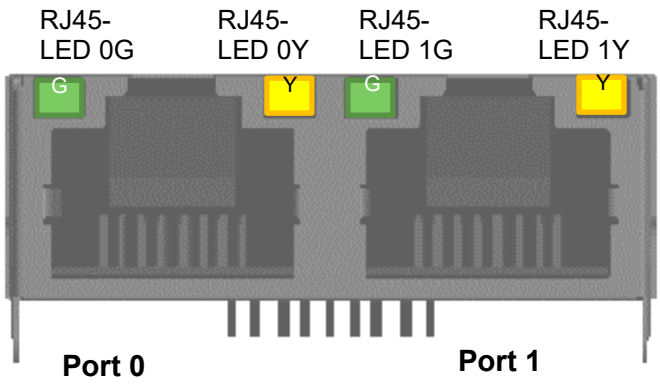

Figure 3.8 RJ45-LED of R-IN32M3 Module

To comply with Industrial Ethernet, the LEDs shown in Table 3-5 must be added for each supported protocol.

The host CPU must control the LEDs according to the LED status received from the R-IN32M3 Module.

Table 3-5 State Indication

Industrial Ethernet Standard	Status LED2		Status LED3	
PROFINET Note1	SF	*Red*	BF	*Red*
	Connection	*Green*	DCP Indicator	*Green*
EtherNet/IP Note2	MS	*Green/ Red*	NS	*Green/ Red*
EtherCAT Note3	RUN	*Green*	ERR	*Red*

Note1 PROFINET Diagnosis Guideline V1.4 Chapter 6.7

Note2 The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP

Note3 EtherCAT Indicator and Labeling ETG.1300S(R) Vx.x.x

The following describes the implementation of hardware and software for each protocol.

3.3.1 PROFINET

3.3.1.1 LED Control

(1) Hardware Design

PROFINET requires one DCP (Discovery and Configuration Protocol) signal indicator, which indicate which device is assigned a symbolic name and IP addresses. Also, it is not mandatory, but normaly recommended to add several LEDs. R-IN32M3 Module supports the following LED controls.

BF (Bus Failure)

SF (System Fialure)

Connection

DCP blink signaling

Figure 3-9 shows a circuit example of PROFINET LED connection.

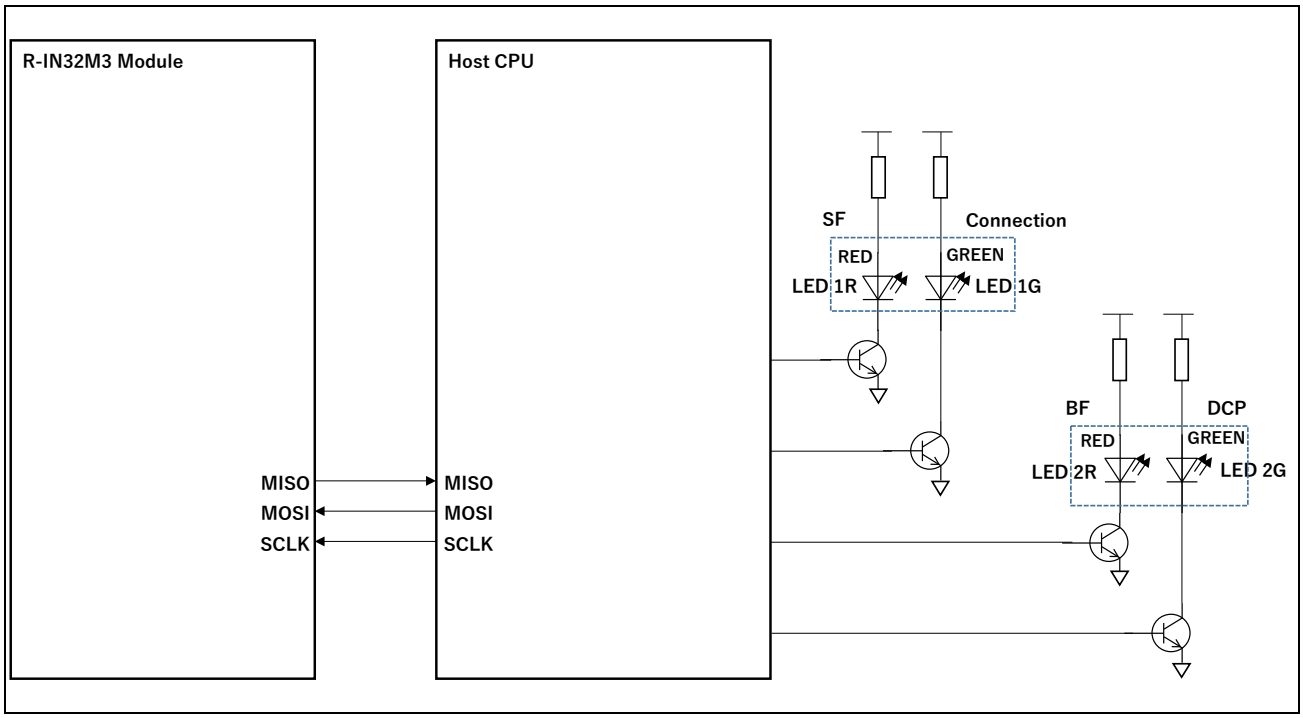

Figure 3-9 Example of PROFINET LED Connections

(2) Software Implement

The LEDs in Figure 3-9 are mounted separately from R-IN32M3 Module and controlled by the host controller. It controls the LEDs according to the LED status received from R-IN32M3 Module. See Table 3-6 below.

Table 3-6 PROFINET State Indication by the host controller

LED	LED Name	Colour	State	Description
1R	SF	Red	ON	Maintenance required. At least one diagnosis exists.
			OFF	No bus error is present
1G	Connection	Green	ON	Connection established
			OFF	Not connected
2R	BF	Red	ON	ERROR. Bus error occurred; the connection was deleted. An alarm was issued.
			OFF	No error is present
2G	DCP	Green	Blink	DCP blink
			OFF	No DCP service

uGOAL provides LED drivers (plat/ra6m4ek) for LED control by the host CPU and realizes the function by implementing the hardware control part in the driver.

The sample software for RA6M4 controls the LEDs, "LED 1RG and LED 2RG" which is implemented on R-IN32M3 Module solution board by using I2C communication via Arduino™ interface. The following is an implementation example for RA6M4.

Target File:

plat\ra6m4ek\plat.c

Target API :
In this sample software, the LEDs are controlled via I2C driver, so I2C master driver (I2C Master on IIC) is incorporated on the Smart Configurator and controlled via the FSP driver.

(a) plat_ledSet

Update the status of each LED held on uGOAL via I2C communication.

```
1.      /* set new value for LED states */
2.      g_i2c_master0.p_api->slaveAddressSet(g_i2c_master0.p_ctrl, GOAL_DRV_IIC_LED_ADDR,
        I2C_MASTER_ADDR_MODE_7BIT);
3.
4.      ledState[0] = 0x16;                  /* register offset */
5.      ledState[1] = (state >> 0) & 0xFF;
6.      ledState[2] = (state >> 8) & 0xFF;
7.      ledState[3] = (state >> 16) & 0xFF;
8.      goal_drvIicClearXferStatus();
9.      g_i2c_master0.p_api->write(g_i2c_master0.p_ctrl, (uint8_t *) &ledState[0], 4, false);
10.     goal_drvIicCheckXferStatus(I2C_MASTER_EVENT_TX_COMPLETE);
```

The LED status received from R-IN32M3 Module is shaped in the application layer and set to the argument of this API. Each bit of the shaped argument LED information (4 bytes) is assigned LED ON/OFF information. The LED is ON when the corresponding bit is 1.

Ex)

PROFINET connection established: 0x0404.

PROFINET disconnection: 0x0410.

The I2C communication data for control the LED is 4 bytes and allocated 3 bytes of the LED information of the argument as shown below.

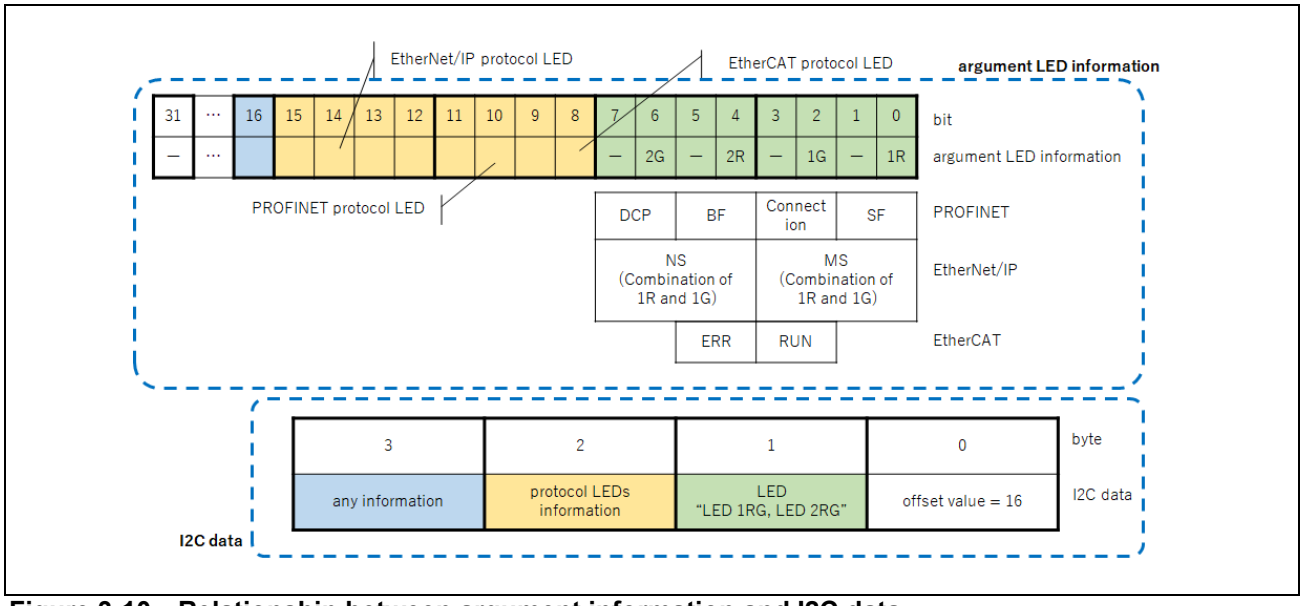

Figure 3-10 Relationship between argument information and I2C data

If controlling the LEDs, "LED 1RG and LED 2RG" by using a general purpose I/O port, refer to the following example.

(b) plat_ledSet (for GPIO)

Update the status of each LED held on uGOAL via the general purpose I/O port.

```
1.  g_ioport.p_api->pinWrite(LedPinSetting(state, LED1_RED));
2.  g_ioport.p_api->pinWrite(LedPinSetting(state, LED1_GREEN));
3.  g_ioport.p_api->pinWrite(LedPinSetting(state, LED2_RED));
4.  g_ioport.p_api->pinWrite(LedPinSetting(state, LED2_GREEN));
5.  g_ioport.p_api->pinWrite(LedPinSetting(state, ETHERCAT));
6.  g_ioport.p_api->pinWrite(LedPinSetting(state, ETHERNETIP));
7.  g_ioport.p_api->pinWrite(LedPinSetting(state, PROFINET));
```

3.3.2 EtherNet/IP

3.3.2.1 LED Control

(1) Hardware Design

In addition to Table 3-5, EtherNet/IP has more detailed LED control guidelines. EtherNet/IP communication requires two types of bi-color LED displays:

MS (Module Status Indicator)

NS (Network Status Indicator)

Figure 3-11 shows a circuit example of EtherNet/IP LED connection.

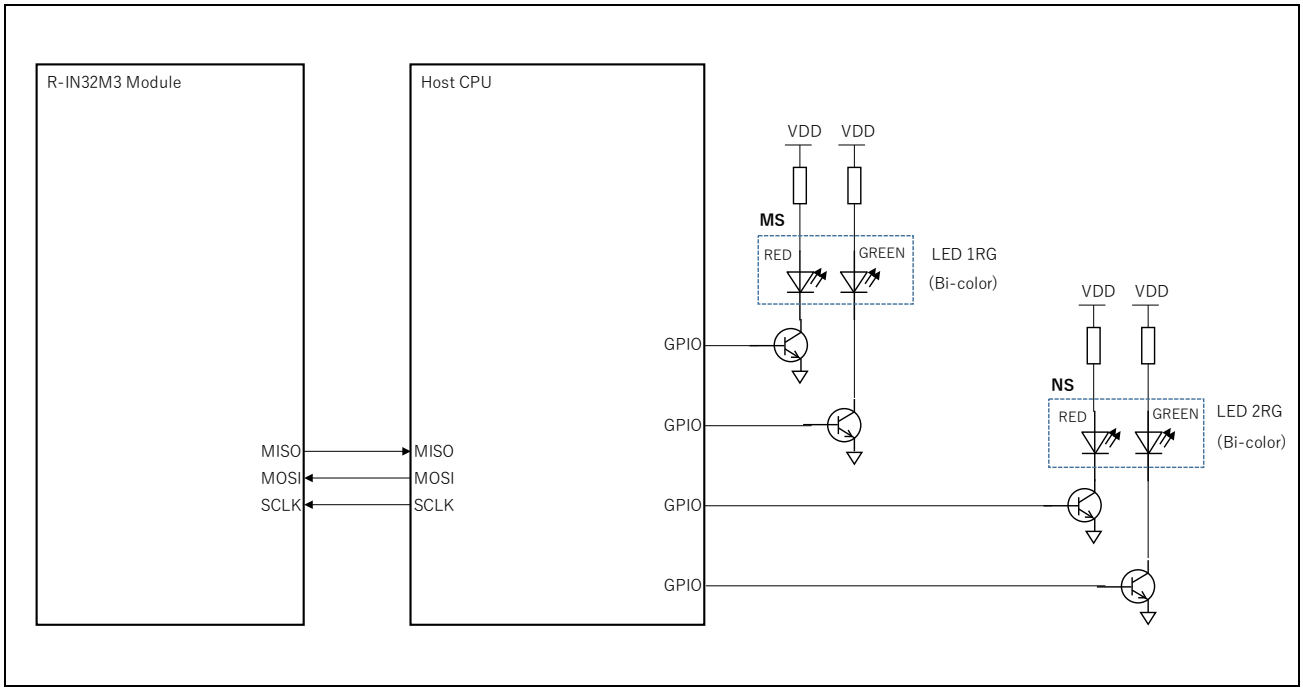

Figure 3-11 Example of EtherNet/IP LED Connections

(2) Software Implement

The MS and NS indicators must be implemented separately from the R-IN32M3 Module and controlled by the application controller. The host CPU must control the LEDs, "LED 1RG and LED 2RG", according to the LED status received from the R-IN32M3 Module. For LED control methods, use LED drivers in uGOAL as well as PROFINET. For more information, see Chapter 3.3.1.1(2).

Refer to Table 3-7 and Table 3-8 for detail specification.

Table 3-7 MS (Module Status Indicator LED 1RG)

Indicator State	Summary	Requirements
Steady off	No power	If power is not being supplied to the device, the module state indicator is to be steadily off.
Steady green	Device operational	If the device is operating correctly, the module status indicator shall be steady green.
Flashing green	Standby	If the device has not been configured, the module state indicator is to be flashing green.
Flashing red	Major recoverable fault	If the device has detected a major recoverable fault, the module state indicator is to be flashing red.
Steady red	Major unrecoverable fault	If the device has detected a major non-recoverable fault, the module state indicator is to be steadily red.
Flashing green / red	Self-test	While the device is performing its power up testing, the module status indicators shall apply the test sequence described below. - The module state indicator shall turn green for approximately 0.25 seconds, turn red for approximately 0.25 seconds, and then turn green and retain that state until the power-up test has been completed. - If both module state and network state indicators are present, the module state indicator test sequence shall occur before or simultaneously with the network state indicator test sequence(s). If more than one network state indicator is present, then each network state indicator test sequence may proceed in succession or simultaneously. - After completion of this power-up test, the indicator(s) are to represent a normal operational state.

Table 3-8 NS (Network Status Indicator LED 2RG)

Indicator State	Summary	Requirement
Steady off	Not powered, no IP address	Power to the device is off or is on but no IP address has been configured (Interface Configuration attribute of the TCP/IP Interface Object).
Flashing green	No connections	An IP address has been configured but no CIP (common industrial protocol) connections Note 1 have been established, and an exclusive owner connection Note 2 has not reached time-out.
Steady green	Connected	An IP address has been configured, at least one CIP connection (in any transport class) has been established, and an exclusive owner connection has not reached time-out.
Flashing red	Connection timeout	An IP address has been configured, and the exclusive owner connection for which this device is the target has reached time-out. The network state indicator shall only return to being steadily green when all timed-out exclusive owner connections are re-established. In devices that support a single exclusive owner connection, a transition to being steadily green is to proceed when any subsequent exclusive owner connection is established. Devices that support multiple exclusive owner connections shall retain the O->T (Originator to Target) connection path information when an exclusive owner connection reaches time-out. The network status indicator shall make the transition from flashing red to steadily green only when all connections to previously timed-out O->T connection points have been re-established. Time-out of connections other than exclusive owner connections are not to cause the indicator to flash red. The flashing red state applies to target connections only. Originators and CIP routers are not to cause the LED to enter this state.
Flashing green / red	Self-test	While the device is performing its power up testing, the network status indicator shall perform a test sequence as described in Table 3-7.

Note 1 The common industrial protocol (CIP) is an open application layer protocol, and EtherNet/IP uses this protocol for the application layer. Refer to the Ethernet/IP Specifications for details.

Note 2 The exclusive owner connection is used for controlling the outputs of the module and shall not be dependent on any other condition. Only one exclusive owner connection can be opened against the module. Refer to the Ethernet/IP Specifications for details.

3.3.3 EtherCAT

3.3.3.1 LED Control

(1) Hardware Design

In addition to Table 3-5, EtherCAT has more detailed LED control guidelines. EtherCAT communication requires four types of LED displays:

L/A IN (Link/Activity In)

L/A OUT (Link Activity Out)

RUN (device status indicator)

ERR (error status indicator)

And EtherCAT requires an explicit device ID selector. The ID selector can be any type, such as a rotary switch or a display with controls.

When connecting multiple units such as daisy chain topology or ring topology as EtherCAT standard, it is necessary to connect communication port 0 as IN and communication port 1 as OUT, according to the EtherCAT standard.

The relationship between each communication port and IN and OUT is fixed in terms of hardware according to the EtherCAT standard, and it is necessary to indicate IN and OUT on the device.

Figure 3-12 shows an example of an EtherCAT LED connection, ID selector. In the example, the RUN LED (LED 2G) and ERR LED (LED 2R) are independent LEDs, but they can be replaced with a single bi-color LED (STATUS LED).

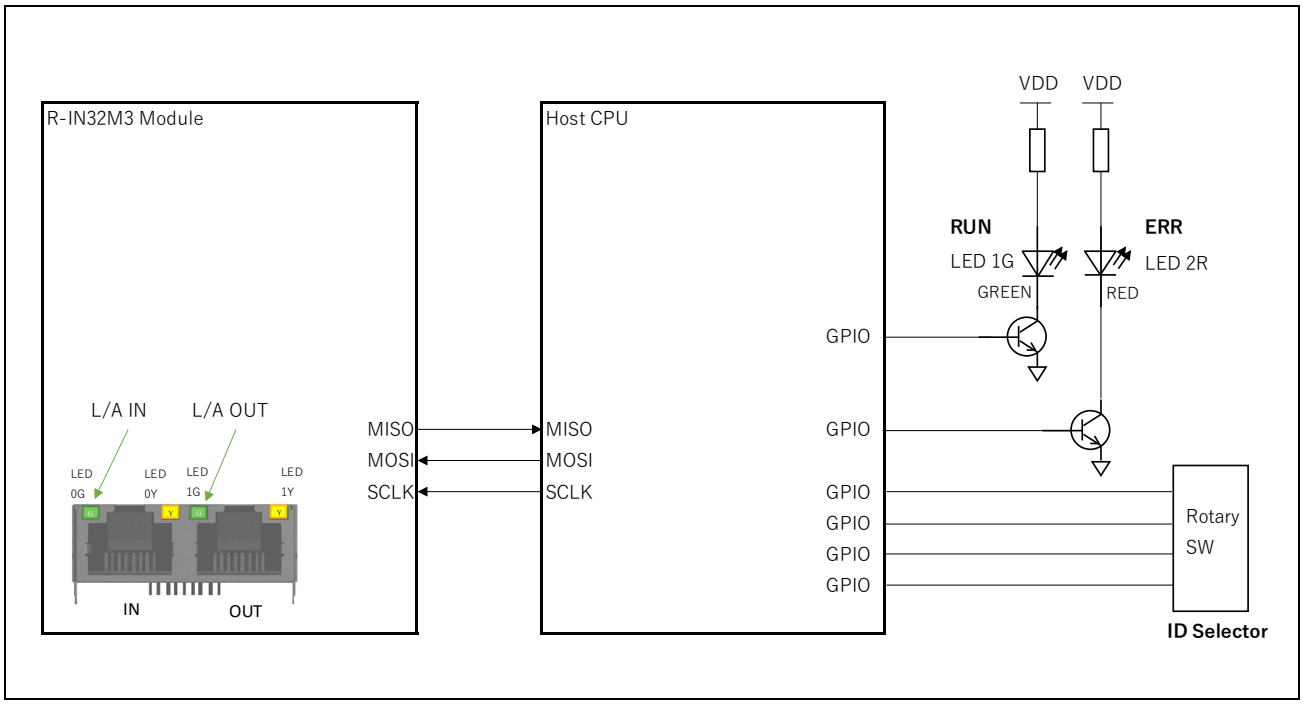

Figure 3-12 Example of EtherCAT LED and ID-selector connection

The LEDs for L / A IN and L / A OUT are included in the R_IN32M3 Module and are controlled by the R-IN32M3 Module. Therefore, it does not need to be designed by the user. Details are shown in Table 3-9.

Table 3-9 LED status display for L /A IN and L/ A OUT by R-IN32M3 Module

RJ45- LED	Name	Color	State	Description
0G, 1G	L/A IN and L/A OUT Indicator	Green	OFF	The Link is not established.
			Flickering	The Link is established with transmitted and received data
			ON	The Link is not established without transmitted and received data

(2) Software Implement

RUN LED and ERR LED must be implemented separately from the R-IN32M3 Module and controlled by the host controller. The host controller can obtain information to control the LEDs from the R-IN32M3 Module via SPI protocol. For LED control methods, use LED drivers in uGOAL as well as PROFINET. For more information, see Chapter 3.3.1.1(2).

The example in Figure 3-12 shows details in Table 3-10.

Table 3-10 EtherCAT status indicator by host controller

LED	LED Name	Color	State	Description
1G	RUN	Green	OFF	The device is in state INIT
			Blinking	The device is in state PRE-OPERATIONAL
			Single flash	The device is in state SAFE-OPERATIONAL
			ON	The device is in state OPERATIONAL
2R	ERR	Red	OFF	No error. The EtherCAT communication of the device is in working condition
			Blinking	General Configuration Error
			Single flash	Synchronization Error.
			Double flash	Sync Manager Watchdog timeout
			Flickering	Booting Error.
			ON	PDI Error.

3.3.3.2 ID Selector

(1) Hardware Design

It is mandatory for EtherCAT standard to have explicit function to select EtherCAT device ID. The shape of the ID selector can be any type, such as a rotary switch or a controlled display.

Figure 3-12 contains an example of connecting an ID selector.

The sample software for RA6M4 is implemented and verified with the hardware connecting "Pmod SWT" board (by Digilent, Inc.) to the 1-6pin of the Pmod2 connector of the EK-RA6M4, which is connected with R-IN32M3 Module solution board (YCONNECT-IT-I-RJ4501) via Arduino™ interface.

(2) Software Implement

uGOAL provides ID driver (plat/ra6m4ek) to take device ID set by the ID selector. The function is realized by implementing the hardware control part in the driver.

The following is an implementation example for RA6M4.

Target File：

plat\ra6m4ek\plat.c

Target API：

(a) goal_drvIdSynGet

Returns the value of the generic I/O port (P413, P411, P410 and P412) connected to the upper stage (1-6pin) of the Pmod2 connector as the device ID.

```
1.  if (NULL != pId) {
2.      g_ioport.p_api->pinRead(g_ioport.p_ctrl, GOAL_PMOD_ECAT_ID1, &value);
3.      *pId = value;
4.      g_ioport.p_api->pinRead(g_ioport.p_ctrl, GOAL_PMOD_ECAT_ID2, &value);
5.      *pId |= (value << 1);
6.      g_ioport.p_api->pinRead(g_ioport.p_ctrl, GOAL_PMOD_ECAT_ID3, &value);
7.      *pId |= (value << 2);
8.      g_ioport.p_api->pinRead(g_ioport.p_ctrl, GOAL_PMOD_ECAT_ID4, &value);
9.      *pId |= (value << 3);
10. }
```

3.3.3.3 DC Mode

(1) Hardware Design

When using Distributed Clocks (DC) synchronization mode on EtherCAT, the SYNC0 and SYNC1 signals of the R-IN32M3 Module must be connected to the external interrupt terminal of the host CPU, as shown in the Figure 3-1 connection example.

<Note>

When using an adapter board (YCONNECT-IT-I-RJ4501) equipped with an R-IN32M3 Module, the terminals of the applicable Arduino connector must be shorted accordingly, depending on the host CPU board to be connected.

Short the 3pin - 6 pin and 4pin -7pin of the Arduino connector J10, respectively, when using sample software for EK-RA6M4 board and RL78/G14 Fast Prototyping Board. For more information, see chapter 2.5 of the “YCONNECT-IT-I-RJ4501 User's Manual (R12UZ0094EJ0****)”.

---

(2) Software Implement

To enable Distributed clocks (DC) synchronization mode, uGOAL receives SYNC0/SYNC1 signal as external interrupts to the host CPU generated by the R-IN32M3 Module and converts it to events in uGOAL for processing. The function is realized by implementing the hardware control portion in the driver (plat/ra6m4ek) for event processing.

The following is an implementation example for RA6M4.

Target File：

plat\ra6m4ek\plat.c

Target API：
In this sample software, external interrupts are applied to IRQ interrupts. An interrupt control unit driver (r_icu) is incorporated on Smart Configurator and controlled via an FSP driver.

(a) plat_eventRegister

This API is called when IRQ interrupts are ready for use. In this sample software, the open processing [goal_maEventOpen()] of the ICU module is called, and the IRQ interrupt registered on Smart Configurator is ready to be used.

```
1.      result = R_ICU_ExternalIrqOpen(mEventCfg[cnt].pInst->p_ctrl, mEventCfg[cnt].pInst->p_cfg);
2.      if (FSP_SUCCESS != result) {
3.          return GOAL_ERR_NULL_POINTER;
4.      }
5.      if (GOAL_TRUE == flgAutoStart){
6.          R_ICU_ExternalIrqEnable(mEventCfg[cnt].pInst->p_ctrl);
7.      }
```

(b) plat_eventEnable

Enable IRQ interrupt.

```
1.  R_ICU_ExternalIrqEnable(mEventCfg[cnt].pInst->p_ctrl);
```

(c) plat_eventDisable

Disable IRQ interrupts.

```
1.  R_ICU_ExternalIrqDisable(mEventCfg[cnt].pInst->p_ctrl);
```

3.4 OS

uGOAL provides wrapper functions for OS-independent functions. Also, the sample software used in the description of this document is an OS-less sample, so there is no need to for user to implement it.

3.5 Timer

uGOAL uses one channel of hardware timer as a system timer. Implement a tick counter, g_ulTickCounter, to count up every time one tick elapses with the system timer. By default, it is set to 1ms per tick (GOAL_TGT_TIMER_TICKS_PER_SECOND setting).

In this sample software, after implementing the timer module with Smart Configurator, execute the timer in the uGOAL's API for initialization (plat_init). See Chapter 3.1 for specific implementations.

3.6 UART

(1) Software Implement

There is a function of UART communication between PC and host CPU. UART is used for debugging of host CPU feature. This debugging feature is enabled by changing the following compile macros. But this feature is disabled in all sample by default.

Table 3-11 Compile macro for enabling debugging feature

Compile macro	Default value
GOAL_UGOAL_CONFIG	0

For more information of enabling debugging feature, see "RA Sample Application (uGOAL Edition) (R30AN0398EJ****)".

<Note>

In sample software for RL78, the function to output log message for debug is prohibited due to the memory allocation limitation of RL/78G14 device.

---

uGOAL uses plat_logInfo() to call vprintf, which is a standard I/O library, for outputting strings as a debug. Therefore, by preparing a separate “_write” function and replacing it with the override when compiling this vprintf, PC terminal software output via the UART driver can be realized.

uGOAL provides a UART driver (plat/ra6m4ek) for controlling UART communication and realizes the function by implementing a hardware control part in the driver.

Target File：

plat\ra6m4ek\plat.c

Target API：
This sample is implemented using the UART function of the SCI module. The UART driver (r_sci_uart) of the SCI module is incorporated on Smart Configurator to make calls to each API.

(a) _write

This function prepared separately is called when writing strings in UART communication. This sample is implemented by SCI write processing of SCI module.

```
for (index = 0; index < len; index++)
{
   if (*ptr == '\n')
   {
      cr = '\r';
      flg_rawtrans_done = GOAL_FALSE;
      g_uart0.p_api->write(g_uart0.p_ctrl, &cr, 1);
      while (flg_rawtrans_done != GOAL_TRUE) {
      }
   }
   flg_rawtrans_done = GOAL_FALSE;
   g_uart0.p_api->write(g_uart0.p_ctrl, ptr++, 1);
   while (flg_rawtrans_done != GOAL_TRUE) {
   }
}
```

(b) goal_drvSciUartSynCallback

This API is a callback function that is called when UART communication write is completed. In this sample software, flgTransferDone is set.

```
flg_rawtrans_done = GOAL_TRUE;
```

4. PROFINET

The profile settings that you should set in this sample are shown below.

Slave profile information is provided to the master device in a GSDML (General Station Description Markup Language) file, which provides device information about the device. Therefore, the device program parameter in the slave and the parameter in the GSDML file must match.

Slave : appl\ 01_pnio\goal_appl,h

GSDML : appl \01_pnio\gsdml\GSDML-V2.4-Renesas-irj45-20211014.xml

The XML editor can be used to create and modify GSDML files, and the PROFINET GSD Checker provided by PROFIBUS & PROFINET International (PI) can be used to create and modify the XML files and check the validity of the registration information.

profinet-gsd-checker - www.profibus.com

For details on the API shown below, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

4.1 Device Identity

This section describes Vendor ID, Vendor Name, etc.

These setting values are defined by macros in the program files.

Table 4-1 PROFINET Device Identify

Item	Macro	Default Value [01_pnio]
Vendor ID	APPL_PNIO_VENDOR_ID	0x02c7
Vendor Name	APPL_PNIO_VENDOR	Renesas Electronics
Device ID	APPL_PNIO_DEVICE_ID	0x0300
IP address	MAIN_APPL_IP	192.168.0.100

4.1.1 Vendor ID

The default value is 0x02C7 (Renesas Electronics).

(1) Slave program parameter

```
1.  #ifndef APPL_PNIO_VENDOR_ID
2.  # define  APPL_PNIO_VENDOR_ID   (0x02c7)          /**< vendor id */
3.  #endif
```

(2) GSDML file

```
4.      <DeviceIdentity VendorID="0x02C7" DeviceID="0x0300">
5.        <InfoText TextId="TOK_DevIdent_InfoText"/>
6.        <VendorName Value="Renesas Electronics"/>
7.      </DeviceIdentity>
```

4.1.2 Vendor Name

The default value is “Renesas Electronics” (Renesas Electronics).

(1) Slave program parameter

```
#ifndef APPL_PNIO_VENDOR
# define  APPL_PNIO_VENDOR    Renesas Electronics   /**< vendor name */
#endif
```

(2) GSDML file

```
    <DeviceIdentity VendorID="0x02C7" DeviceID="0x0300">
      <InfoText TextId="TOK_DevIdent_InfoText"/>
      <VendorName Value="Renesas Electronics"/>
    </DeviceIdentity>
```

4.1.3 Device ID

The default value is 0x0300 (Renesas Electronics R-IN32M3 Module).

(1) Slave program parameter

```
#ifndef APPL_PNIO_DEVICE_ID
# define  APPL_PNIO_DEVICE_ID    (0x0300)        /**< device id */
#endif
```

(2) GSDML file

```
    <DeviceIdentity VendorID="0x02C7" DeviceID="0x0300">
      <InfoText TextId="TOK_DevIdent_InfoText"/>
      <VendorName Value="Renesas Electronics"/>
    </DeviceIdentity>
```

4.1.4 IP Address

Any IP address can be set for evaluation purposes, the IP address value is defined by a macro in the following file. For details on the setting method, refer to Chapter A of Appendix.

(1) Slave program parameter

```
1. #define MAIN_APPL_IP      GOAL_NET_IPV4(192, 168, 0, 100)
2. #define MAIN_APPL_NM      GOAL_NET_IPV4(255, 255, 255, 0)
3. #define MAIN_APPL_GW      GOAL_NET_IPV4(0, 0, 0, 0)
```

(2) GSDML file

No configuration for GSDML file.

4.2 Data Model

The configuration of Module and Slot used in the PROFINET communication process is briefly described.

In PROFINET I/O devices, the communication process is executed with a configuration in which Ethernet access point information called DAP (Device Access Point) and one or more I/O Modules are mapped to each slot.

- Module: A lump of I/O data, this is with one or more Submodule
- Submodule: I/O data information defined in Module (input/output settings, data length etc.)
- Slot: location to store I / O modules, it is with one or more Subslot
- Subslot: I/O interface defined in Slot

As an example, the Module and Slot configurations used in 01_pnio sample in this sample are shown below.

See 4.2.3 Output Data and 4.2.4 Input Data for the respective settings.

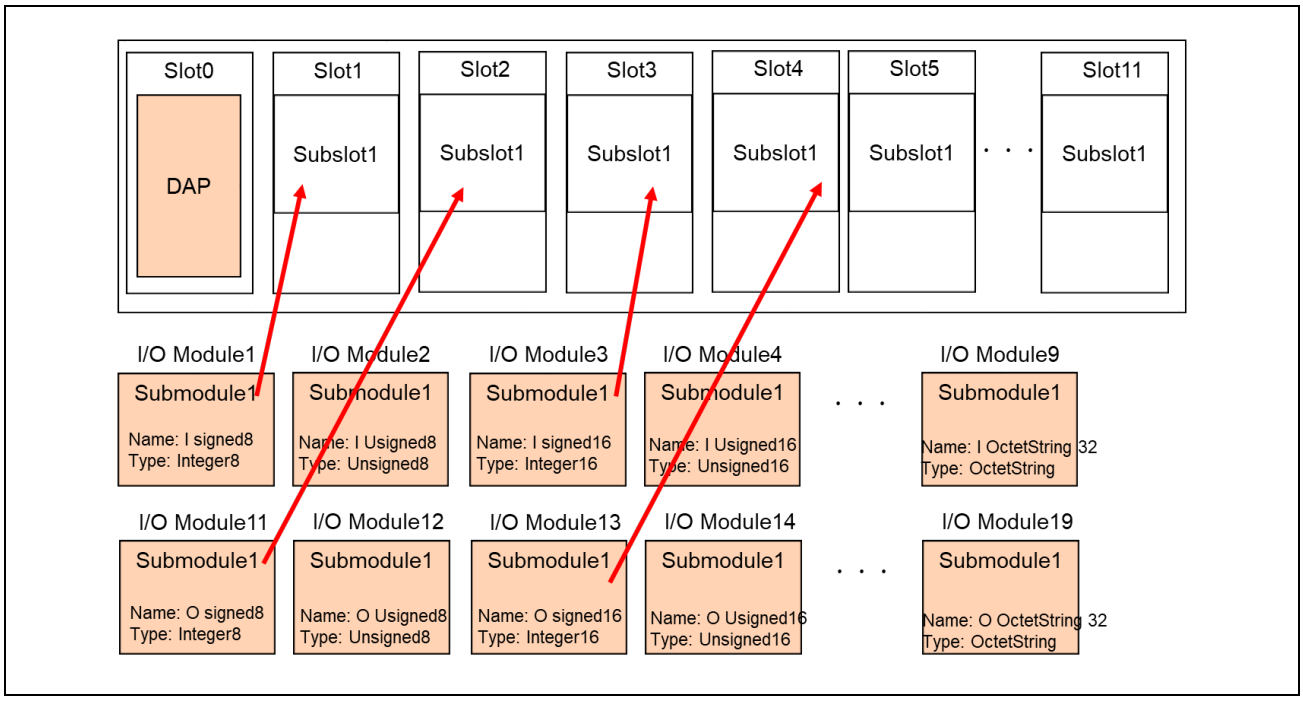

Figure 4-1 Module and Slot configuration [01_pnio]

4.2.1 Slot maximum

The set values are defined by macros in the following file.

```
1.  #define APPL_SLOT_COUNT    (12)          /**< maximum number of slots */
```

4.2.2 Subslot maximum

The set values are defined by macros in the following file.

```
1.  #define APPL_SUBSLOT_COUNT   (4)          /**< maximum number of subslots */
```

4.2.3 Output Data

The following is a sample software configuration for data received from the master device.

Table 4-2 Module and Slot Output configuration

sample		Output variable	Module	Submodule	Slot	Subslot	Size
04_pnio_large	01_pnio	dataDm_1	APPL_MOD_11 (201)	APPL_MOD_11_SUB_1 (1)	APPL_SLOT_02 (2)	APPL_SLOT_02_SUB_1 (1)	APPL_SIZE_11_SUB_1_OUT (1)
		dataDm_2	APPL_MOD_13 (203)	APPL_MOD_13_SUB_1 (1)	APPL_SLOT_04 (4)	APPL_SLOT_04_SUB_1 (1)	APPL_SIZE_13_SUB_1_OUT (16)
	-	dataRpc_1	APPL_MOD_15 (205)	APPL_MOD_15_SUB_1 (1)	APPL_SLOT_06 (6)	APPL_SLOT_06_SUB_1 (1)	APPL_SIZE_15_SUB_1_OUT (32)
		dataRpc_2	APPL_MOD_16 (206)	APPL_MOD_16_SUB_1 (1)	APPL_SLOT_08 (8)	APPL_SLOT_08_SUB_1 (1)	APPL_SIZE_16_SUB_1_OUT (32)
		dataRpc_3	APPL_MOD_17 (207)	APPL_MOD_17_SUB_1 (1)	APPL_SLOT_10 (10)	APPL_SLOT_10_SUB_1 (1)	APPL_SIZE_17_SUB_1_OUT (32)

To change Output data, it is necessary to change the slave program file and GSDML file.

(1) Slave program parameter

1. Change the size macro according to type (bool, Integer8, Integer16, Integer32, Unsigned8, Unsigned16, Unsigned32, etc.) of I/O Module and Submodule used for Output data.

```
1.  #define APPL_SIZE_11_SUB_1_OUT    (1)     /**< size for module 11 sub 1 output */
    … omit …
2.  #define APPL_SIZE_13_SUB_1_OUT    (16)    /**< size for module 13 sub 1 output */
    … omit …
```

```
1.  #define APPL_MOD_11               (0x201)   /**< module 11 */
2.  #define APPL_MOD_11_SUB_1         (0x01)    /**< submodule for module 11 */
    … omit …
3.  #define APPL_MOD_13               (0x203)   /**< module 13 */
4.  #define APPL_MOD_13_SUB_1         (0x01)    /**< submodule for module 13 */
    … omit …
```

2. Change the macros for the identifier (ID) to match the Slot and Subslot used for the Output data.

```
1.  #define APPL_SLOT_02              (2)       /**< slot 2 */
2.  #define APPL_SLOT_02_SUB_1        (1)       /**< subslot for slot 2 */
    … omit …
3.  #define APPL_SLOT_04              (4)       /**< slot 4 */
4.  #define APPL_SLOT_04_SUB_1        (1)       /**< subslot for slot 4 */
    … omit …
```

3. Change the function call of goal_pnioSubmodNew() that creates Submodule according to the number of Submodule used for Output data. For the argument, use Module and Submodule macros set above.

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …

5.      /* create submodules */
    … omit …                                                    Module11

6.      res = goal_pnioSubmodNew(pPnio, APPL_MOD_11, APPL_MOD_11_SUB_1,
    GOAL_PNIO_MOD_TYPE_OUTPUT, 0, APPL_SIZE_11_SUB_1_OUT, GOAL_PNIO_FLG_AUTO_GEN);
7.      if (GOAL_RES_ERR(res)) {
8.          goal_logErr("failed to add submodule");
9.          return res;
10.     }
11.
    … omit …                                                    Module13
12.
13.     res = goal_pnioSubmodNew(pPnio, APPL_MOD_13, APPL_MOD_13_SUB_1,
    GOAL_PNIO_MOD_TYPE_OUTPUT, 0, APPL_SIZE_13_SUB_1_OUT, GOAL_PNIO_FLG_AUTO_GEN);
14.     if (GOAL_RES_ERR(res)) {
15.         goal_logErr("failed to add submodule");
16.         return res;
17.     }
18.
    … omit …
```

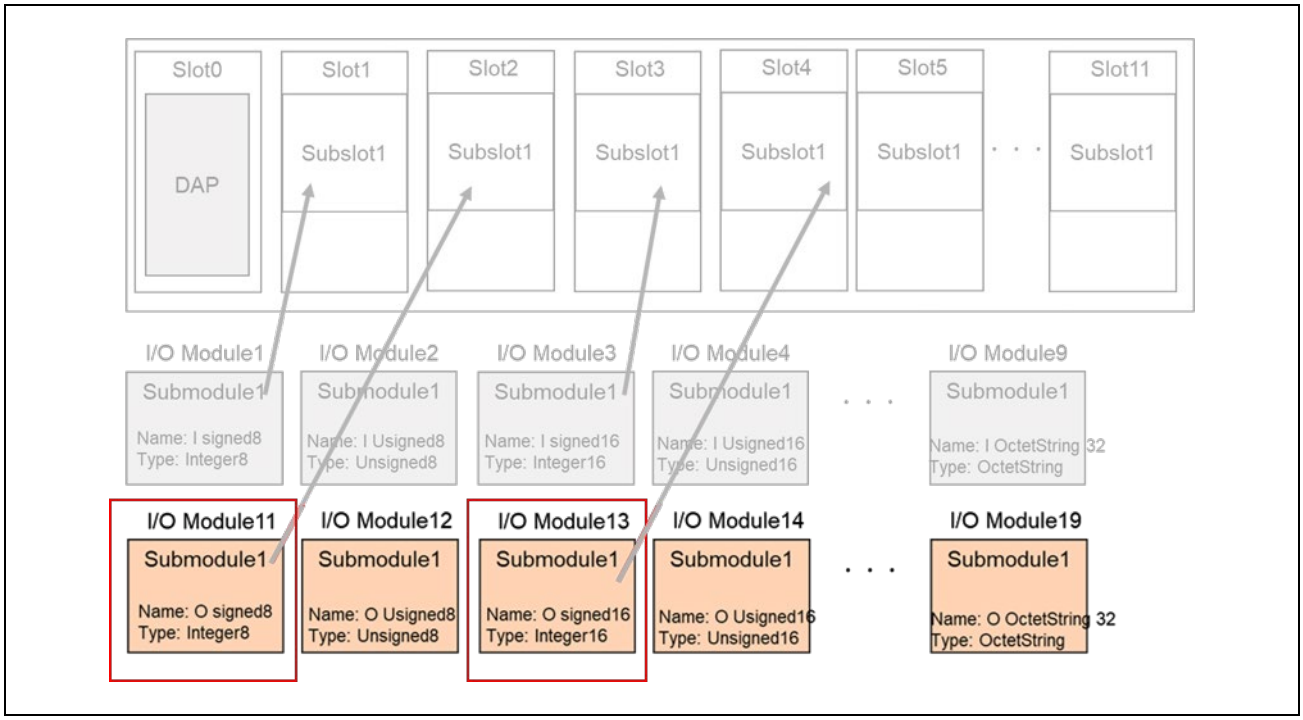

Figure 4-2 Create Submodule - Output

4. Change the function call of goal_pnioSubslotNew() that creates Subslot according to the number of Subslot used for Output data. For the argument, use Slot and Subslot macros set above.

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …
5.      /* create subslots */
    … omit …
6.
7.      res = goal_pnioSubslotNew(pPnio, APPL_API, APPL_SLOT_02, APPL_SLOT_02_SUB_1,
    GOAL_PNIO_FLG_AUTO_GEN);
8.      if (GOAL_RES_ERR(res)) {
9.          goal_logErr("failed to add subslot");
10.         return res;
11.     }
12.
    … omit …
13.
14.     res = goal_pnioSubslotNew(pPnio, APPL_API, APPL_SLOT_04, APPL_SLOT_04_SUB_1,
    GOAL_PNIO_FLG_AUTO_GEN);
15.     if (GOAL_RES_ERR(res)) {
16.         goal_logErr("failed to add subslot");
17.         return res;
18.     }
19.
    … omit …
```

Slot2(Subslot1)

Slot4(Subslot1)

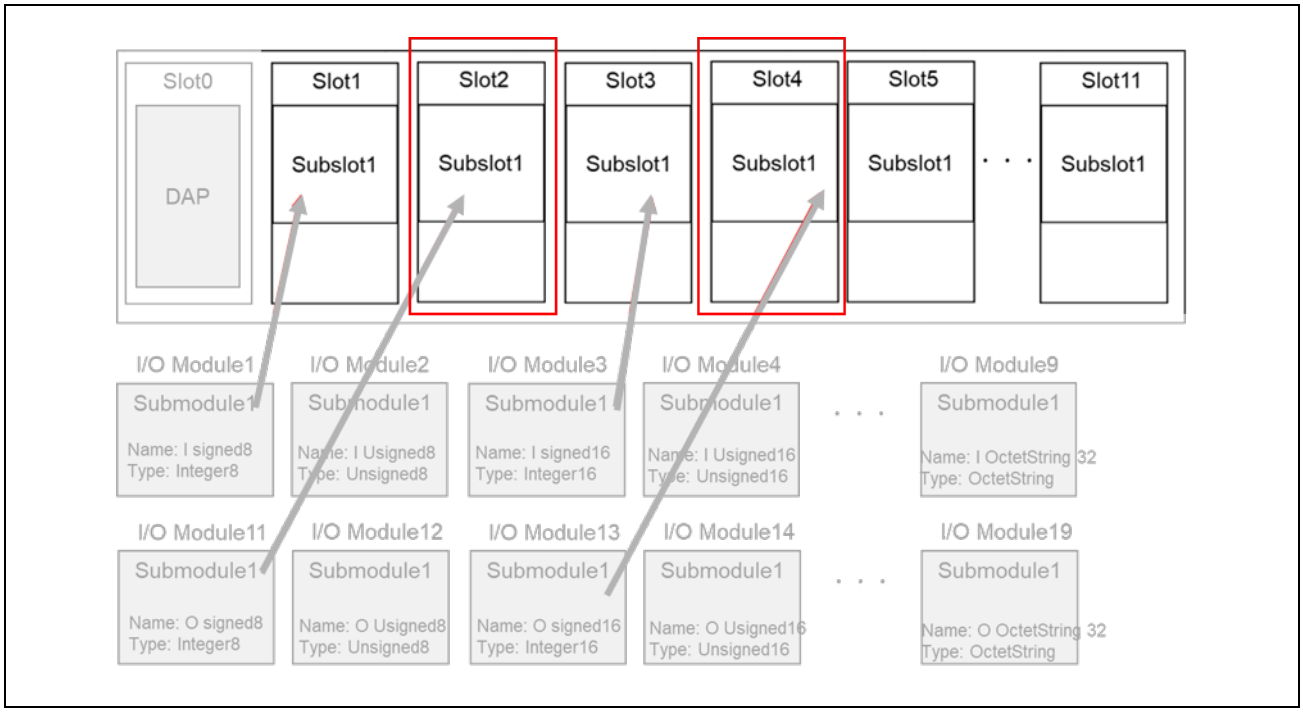

Figure 4-3 Create Subslot - Output

5. Change the function call of goal_pnioSubmodPlug() * that mapping Module and Slot according to the configuration of Module and Slot used for Output data. For the argument, use Slot, Subslot, Module and Submodule macros set above.

   * In case of RPC transfer, call goal_pnioRpcSubmodPlug()

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …
5.      /* plug modules into slots */

    … omit …
6.      res = goal_pnioSubmodPlug(pPnio, APPL_API, APPL_SLOT_02, APPL_SLOT_02_SUB_1,
    APPL_MOD_11, APPL_MOD_11_SUB_1);
7.      if (GOAL_RES_ERR(res)) {
8.          goal_logErr("failed to plug submodule");
9.          return res;
10.     }
11.
    … omit …

12.     res = goal_pnioSubmodPlug(pPnio, APPL_API, APPL_SLOT_04, APPL_SLOT_04_SUB_1,
    APPL_MOD_13, APPL_MOD_13_SUB_1);
13.     if (GOAL_RES_ERR(res)) {
14.         goal_logErr("failed to plug submodule");
15.         return res;
16.     }
17.
    … omit …
```

Module11 - Slot2(Subslot1)

Module13 – Slot4(Subslot1)

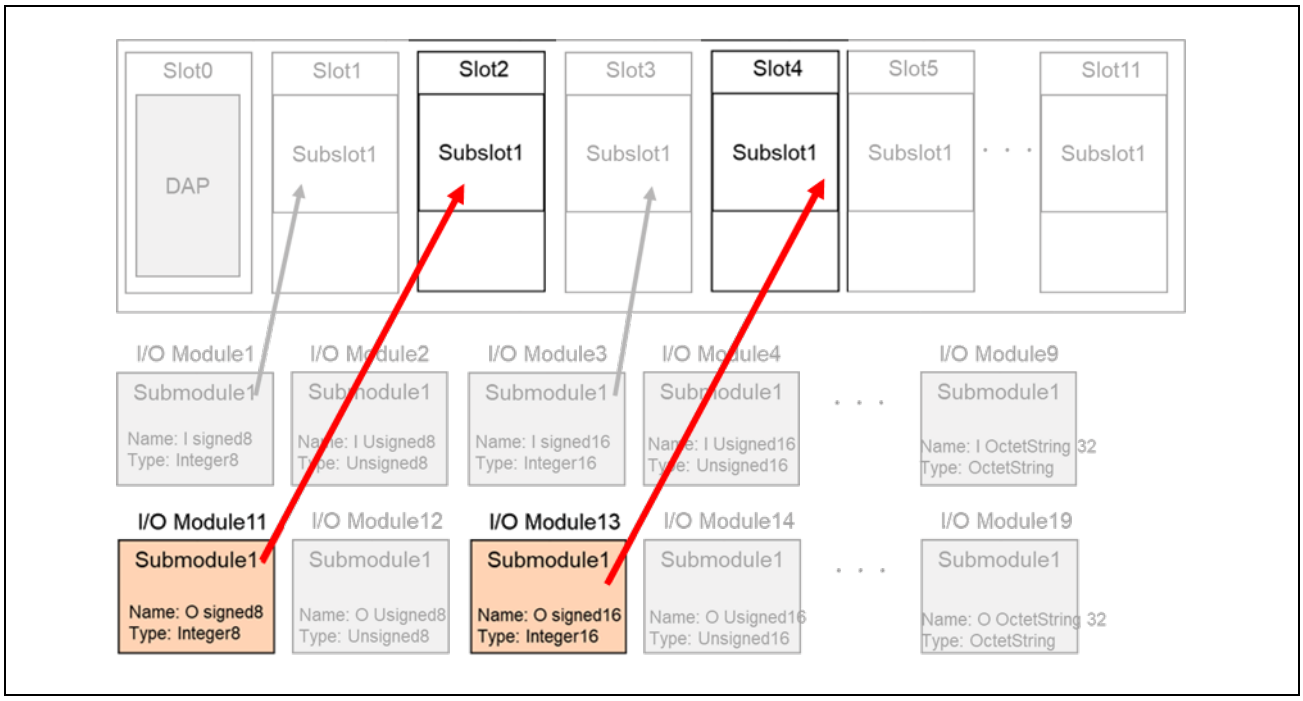

Figure 4-4 Plugin - Output

(2) GSDML file

1. Change the contents of the following "UseableModules" according to the mapping of Module and Slot used for Output data.

```
<UseableModules>
  … omit …

  <ModuleItemRef ModuleItemTarget="ID_Mod_201" FixedInSlots="2"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_202"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_203" FixedInSlots="4"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_204"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_205"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_206"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_207"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_208"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_209"/>
  <ModuleItemRef ModuleItemTarget="ID_Mod_210"/>
</UseableModules>
```

2. Change the contents of the following "ModuleList" according to the count, data type, character string information of Module/Submodule used for Output data.

```
<ModuleList>
  … omit …

  <ModuleItem ID="ID_Mod_201" ModuleIdentNumber="0x00000201">
    <ModuleInfo>
      <Name TextId="TOK_TextId_Module_ID201"/>
      <InfoText TextId="TOK_InfoTextId_Module_ID201"/>
      <HardwareRelease Value="1.0"/>
      <SoftwareRelease Value="1.0"/>
    </ModuleInfo>
    <VirtualSubmoduleList>
      <VirtualSubmoduleItem ID="ID_Mod_201_Sub_1" SubmoduleIdentNumber="0x0001" API="0"
MayIssueProcessAlarm="false">
        <IOData>
          <Output Consistency="All items consistency">
            <DataItem DataType="Unsigned8" TextId="TOK_Output_DataItem_Unsigned8"/>
          </Output>
        </IOData>
        <ModuleInfo>
          <Name TextId="TOK_TextId_Module_ID201"/>
          <InfoText TextId="TOK_InfoTextId_Module_ID201"/>
        </ModuleInfo>
      </VirtualSubmoduleItem>
    </VirtualSubmoduleList>
  </ModuleItem>
  <ModuleItem ID="ID_Mod_202" ModuleIdentNumber="0x00000202">
    <ModuleInfo>
… omit …
```

The character string information is linked by "TestId" and managed as a list by "ExternalTextList" in the end of the GSDML file.

```
<ExternalTextList>
  <PrimaryLanguage>
    <!--english-->
… omit …
```

```
    <!--module name-->
    … omit …
    <Text TextId="TOK_TextId_Module_ID201" Value="O LED Request"/>
    <Text TextId="TOK_TextId_Module_ID202" Value="O Unsigned8"/>
    <Text TextId="TOK_TextId_Module_ID203" Value="O OctedString 16 bytes"/>
    <Text TextId="TOK_TextId_Module_ID204" Value="O Unsigned16"/>
    <Text TextId="TOK_TextId_Module_ID205" Value="O OctedString 32 bytes"/>
  … omit …
```

```
    <!--module info name-->
    … omit …
    <Text TextId="TOK_InfoTextId_Module_ID201" Value="Output module (LED request)"/>
    <Text TextId="TOK_InfoTextId_Module_ID202" Value="Output module (Unsigned8)"/>
    <Text TextId="TOK_InfoTextId_Module_ID203" Value="Output module (Octed String 16) "/>
    <Text TextId="TOK_InfoTextId_Module_ID204" Value="Output module (Unsigned16) "/>
    <Text TextId="TOK_InfoTextId_Module_ID205" Value="Output module (Octed String 32) "/>
  … omit …
```

4.2.4 Input Data

The following is a sample software configuration for data send from the slave device.

Table 4-3 Module and Slot Input configuration

sample		Input variable	Module	Submodule	Slot	Subslot	Size
04_pnio_large	01_pnio	dataDm_1	APPL_MOD_01 (101)	APPL_MOD_01_SUB_1 (1)	APPL_SLOT_01 (1)	APPL_SLOT_01_SUB_1 (1)	APPL_SIZE_01_SUB_1_IN (1)
		dataDm_2	APPL_MOD_03 (103)	APPL_MOD_03_SUB_1 (1)	APPL_SLOT_03 (3)	APPL_SLOT_03_SUB_1 (1)	APPL_SIZE_03_SUB_1_IN (16)
	-	dataRpc_1	APPL_MOD_05 (105)	APPL_MOD_05_SUB_1 (1)	APPL_SLOT_05 (5)	APPL_SLOT_05_SUB_1 (1)	APPL_SIZE_05_SUB_1_IN (32)
		dataRpc_2	APPL_MOD_06 (106)	APPL_MOD_06_SUB_1 (1)	APPL_SLOT_07 (7)	APPL_SLOT_07_SUB_1 (1)	APPL_SIZE_06_SUB_1_IN (32)
		dataRpc_3	APPL_MOD_07 (107)	APPL_MOD_07_SUB_1 (1)	APPL_SLOT_09 (9)	APPL_SLOT_09_SUB_1 (1)	APPL_SIZE_07_SUB_1_IN (32)

To change Input data, it is necessary to change the slave program file and GSDML file.

(1) Slave program parameter

1. Change the size macro according to type (bool, Integer8, Integer16, Integer32, Unsigned8, Unsigned16, Unsigned32, etc.) of I/O Module and Submodule used for Input data. Also set a macro for each identifier (ID).

```
1.  #define APPL_SIZE_01_SUB_1_IN      (1)      /**< size for module 1 sub 1 input */
    … omit …
2.  #define APPL_SIZE_03_SUB_1_IN      (16)     **< size for module 3 sub 1 input */
3.
    … omit …
```

```
2.  #define APPL_MOD_01                (0x101)  /**< module 1 */
4.  #define APPL_MOD_01_SUB_1          (0x01)   /**< submodule for module 1 */
    … omit …
5.  #define APPL_MOD_03                (0x102)  /**< module 3 */
6.  #define APPL_MOD_03_SUB_1          (0x01)   /**< submodule for module 3 */
    … omit …
```

2. Change each identifier (ID) macro according to Slot and Subslot used for Input data.

```
1.  #define APPL_SLOT_01               (1)      **< slot 1 */
2.  #define APPL_SLOT_01_SUB_1         (1)      /**< subslot for slot 1 */
    … omit …
3.  #define APPL_SLOT_03               (3)      /**< slot 3 */
4.  #define APPL_SLOT_03_SUB_1         (1)      /**< subslot for slot 3 */
    … omit …
```

3. Change the function call of goal_pnioSubmodNew() that creates Submodule according to the number of Submodule used for Input data. For the argument, use Module and Submodule macros set above.

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …

5.      /* create submodules */
    … omit …                                                    Module1

6.      res = goal_pnioSubmodNew(pPnio, APPL_MOD_01, APPL_MOD_01_SUB_1,
    GOAL_PNIO_MOD_TYPE_INPUT, APPL_SIZE_01_SUB_1_IN, 0, GOAL_PNIO_FLG_AUTO_GEN);
7.      if (GOAL_RES_ERR(res)) {
8.          goal_logErr("failed to add submodule");
9.          return res;
10.     }
11.
    … omit …                                                    Module3
12.
13.     res = goal_pnioSubmodNew(pPnio, APPL_MOD_03, APPL_MOD_03_SUB_1,
    GOAL_PNIO_MOD_TYPE_INPUT, APPL_SIZE_03_SUB_1_IN, 0, GOAL_PNIO_FLG_AUTO_GEN);
14.     if (GOAL_RES_ERR(res)) {
15.         goal_logErr("failed to add submodule");
16.         return res;
17.     }
18.
    … omit …
```

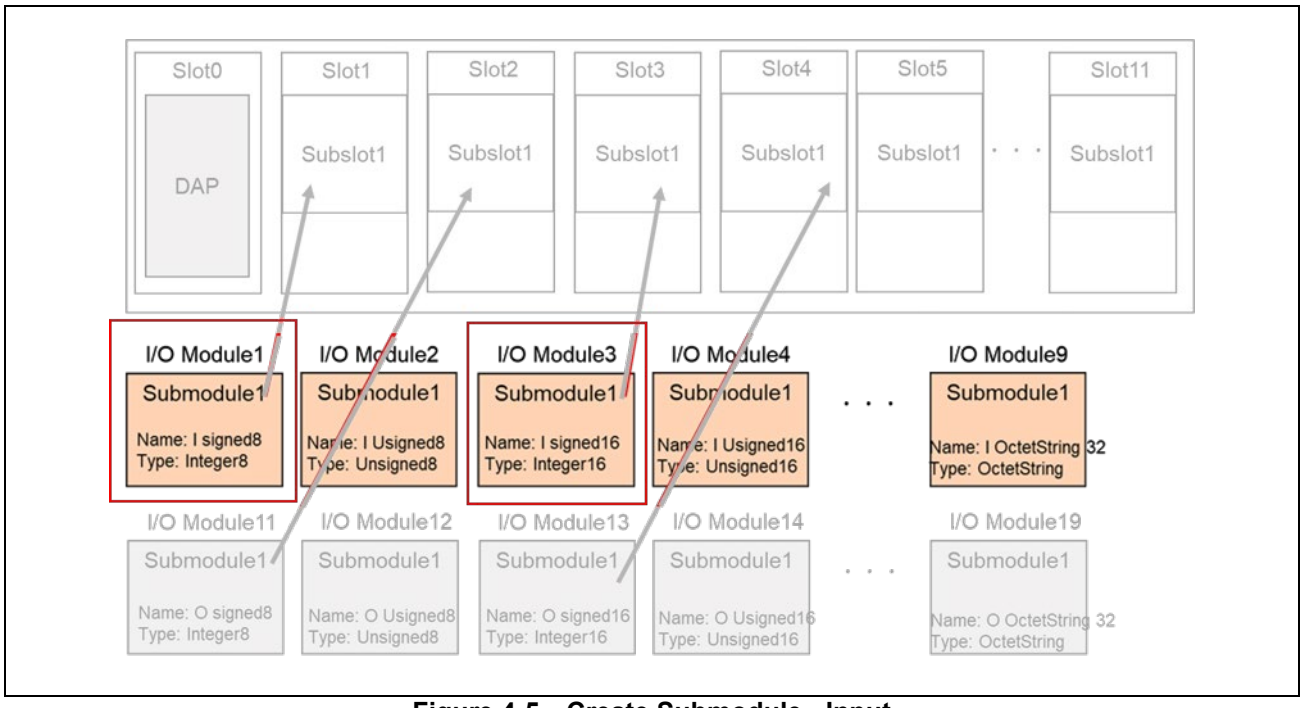

Figure 4-5 Create Submodule - Input

4. Change the function call of goal_pnioSubslotNew() that creates Subslot according to the number of Subslot used for Input data. For the argument, use Slot and Subslot macros set above.

```
2.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …
5.      /* create subslots */
    … omit …                                                    Slot1(Subslot1)
6.
7.      res = goal_pnioSubslotNew(pPnio, APPL_API, APPL_SLOT_01, APPL_SLOT_01_SUB_1,
    GOAL_PNIO_FLG_AUTO_GEN);
8.      if (GOAL_RES_ERR(res)) {
9.          goal_logErr("failed to add subslot");
10.         return res;
11.     }
12.
    … omit …                                                    Slot3(Subslot1)
13.
14.     res = goal_pnioSubslotNew(pPnio, APPL_API, APPL_SLOT_03, APPL_SLOT_03_SUB_1,
    GOAL_PNIO_FLG_AUTO_GEN);
15.     if (GOAL_RES_ERR(res)) {
16.         goal_logErr("failed to add subslot");
17.         return res;
18.     }
19.
    … omit …
```

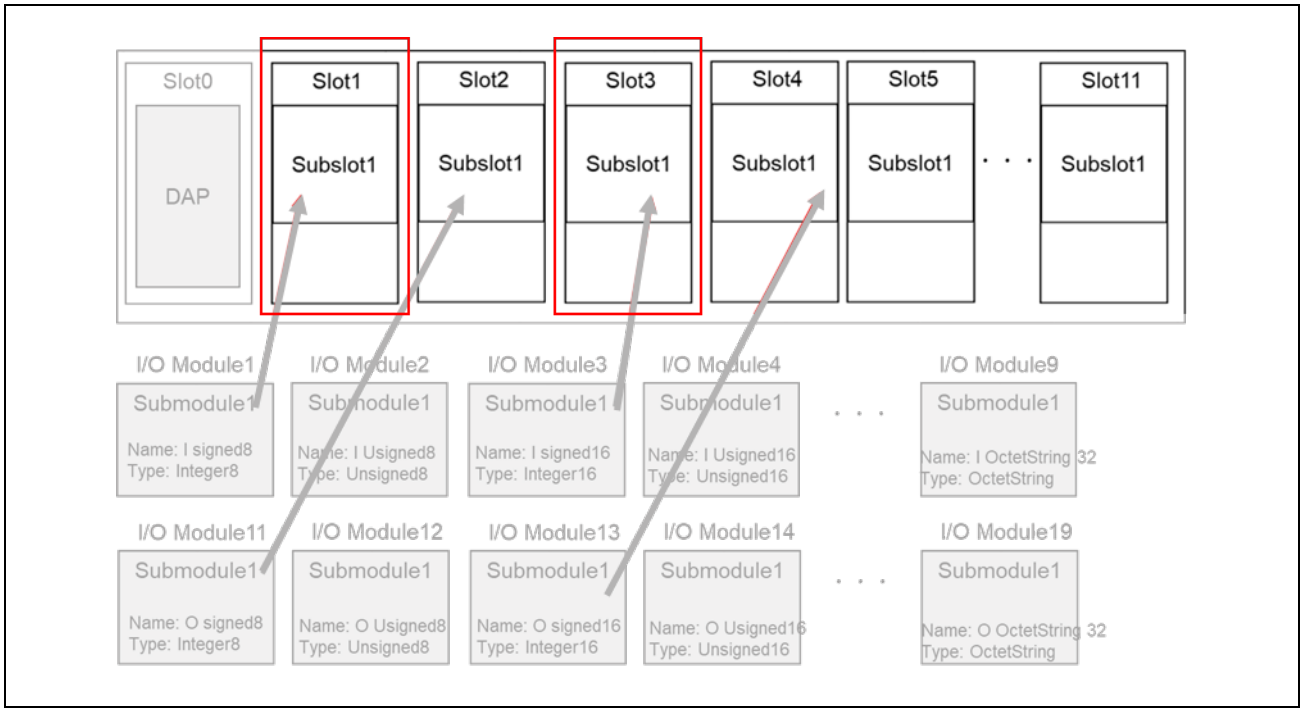

Figure 4-6 Create Subslot - Input

5. Change the function call of goal_pnioSubmodPlug() * that mapping Module and Slot according to the configuration of Module and Slot used for Input data. For the argument, use Slot, Subslot, Module and Submodule macros set above.

   * In case of RPC transfer, call goal_pnioRpcSubmodPlug()

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …
5.
6.      /* plug modules into slots */
7.
    … omit …                                          Module1 – Slot1(Subslot1)
8.
9.      res = goal_pnioSubmodPlug(pPnio, APPL_API, APPL_SLOT_01, APPL_SLOT_01_SUB_1,
    APPL_MOD_01, APPL_MOD_01_SUB_1);
10.     if (GOAL_RES_ERR(res)) {
11.         goal_logErr("failed to plug submodule");
12.         return res;
13.     }
14.
    … omit …                                          Module3 – Slot3(Subslot1)
15.
16.     res = goal_pnioSubmodPlug(pPnio, APPL_API, APPL_SLOT_03, APPL_SLOT_03_SUB_1,
    APPL_MOD_03, APPL_MOD_03_SUB_1);
17.     if (GOAL_RES_ERR(res)) {
18.         goal_logErr("failed to plug submodule");
19.         return res;
20.     }
21.
    … omit …
```

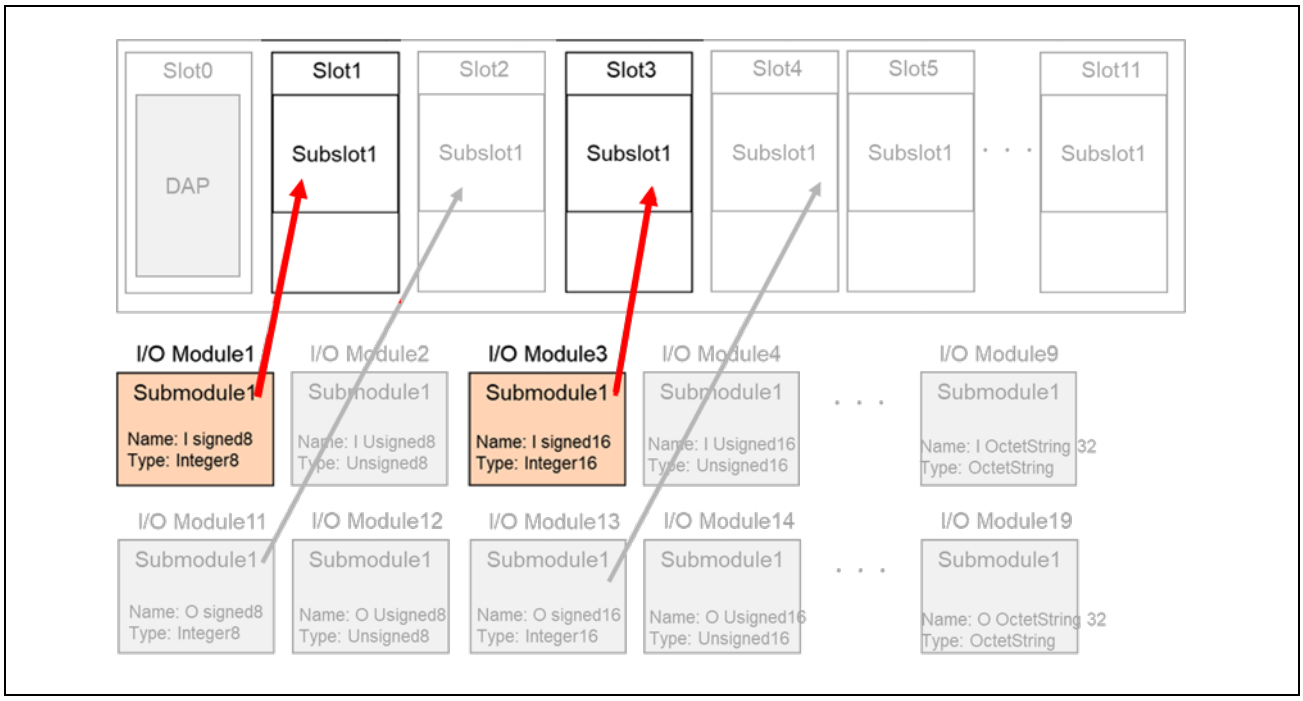

Figure 4-7 Plugin - Input

(2) GSDML file

1. Change the contents of the following "UseableModules" according to the mapping of Module and Slot used for Input data.

```
<UseableModules>
   <ModuleItemRef ModuleItemTarget="ID_Mod_101" FixedInSlots="1"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_102"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_103" FixedInSlots="3"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_104"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_105"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_106"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_107"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_108"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_109"/>
   <ModuleItemRef ModuleItemTarget="ID_Mod_110"/>

   … omit …
</UseableModules>
```

2. Change the contents of the following "ModuleList" according to the count, data type, character string information of Module/Submodule used for Input data.

```
<ModuleList>
  <ModuleItem ID="ID_Mod_101" ModuleIdentNumber="0x00000101">
    <ModuleInfo>
      <Name TextId="TOK_TextId_Module_ID101"/>
      <InfoText TextId="TOK_InfoTextId_Module_ID101"/>
      <HardwareRelease Value="1.0"/>
      <SoftwareRelease Value="1.0"/>
    </ModuleInfo>
  <VirtualSubmoduleList>
    <VirtualSubmoduleItem ID="ID_Mod_101_Sub_1" SubmoduleIdentNumber="0x0001" API="0"
MayIssueProcessAlarm="false">
      <IOData>
        <Output Consistency="All items consistency">
          <DataItem DataType="Unsigned8" TextId="TOK_Input_DataItem_Unsigned8"/>
        </Output>
      </IOData>
      <ModuleInfo>
        <Name TextId="TOK_TextId_Module_ID101”/>
        <InfoText TextId=”TOK_InfoTextId_Module_ID101"/>
      </ModuleInfo>
    </VirtualSubmoduleItem>
  </VirtualSubmoduleList>
  </ModuleItem>
  <ModuleItem ID="ID_Mod_102" ModuleIdentNumber="0x00000102">
    <ModuleInfo>
    … omit …
```

The character string information is linked by "TestId" and managed as a list by "ExternalTextList" in the end of the GSDML file.

```
<ExternalTextList>
 <PrimaryLanguage>
   <!--english-->
… omit …

   <!--module name-->
   <Text TextId="TOK_TextId_Module_ID101" Value="I Switch Status"/>
   <Text TextId="TOK_TextId_Module_ID102" Value="I Unsigned8"/>
   <Text TextId="TOK_TextId_Module_ID103" Value="I OctedString 16 bytes"/>
   <Text TextId="TOK_TextId_Module_ID104" Value="I Unsigned16"/>
   <Text TextId="TOK_TextId_Module_ID105" Value="I OctedString 32 bytes"/>
… omit …

   <!--module info name-->
   <Text TextId="TOK_InfoTextId_Module_ID101" Value="Input module (Switch status)"/>
   <Text TextId="TOK_InfoTextId_Module_ID102" Value="Input module (Unsigned8)"/>
   <Text TextId="TOK_InfoTextId_Module_ID103" Value="Input module (Octed String 16) "/>
   <Text TextId="TOK_InfoTextId_Module_ID104" Value="Input module (Unsigned16) "/>
   <Text TextId="TOK_InfoTextId_Module_ID105" Value="Input module (Octed String 32) "/>
  … omit …
```

4.3 User application Output/Input Data Handling

The sample software implements two types of data transmission/reception applications as example applications.

- **Remote-IO (LED/Switch):** LED lighting control and Switch status from the evaluation board
- **Mirror:** Sends data received from the master and mirrored back

Table 4-4 API and Sample application data variable

sample		Sample app.	Data variable	Purpose	API	reference
04_pnio_large	01_pnio	get LED Data [#3]	dataDm_1 (Slot 2)	Reception	goal_pnioDataOutputGet()	4.2.3 Output Data
		get Mirror Data [#1]	dataDm_2 (Slot 4)			
		set Switch Data [#4]	dataDm_1 (Slot 1)	Transmission	goal_pnioDataInputSet()	4.2.4 Input Data
		set Mirror data [#2]	dataDm_2 (Slot 3)			
	-	get Mirror Data_1	dataRpc_1 (Slot 6)	Reception	goal_pnioDataOutputGet()	4.2.3 Output Data
		get Mirror Data_2	dataRpc_2 (Slot 8)			
		get Mirror Data_3	dataRpc_3 (Slot 10)			
		set Mirror Data_1	dataRpc_1 (Slot 5)	Transmission	goal_pnioDataInputSet()	4.2.4 Input Data
		set Mirror Data_2	dataRpc_2 (Slot 7)			
		set Mirror Data_3	dataRpc_3 (Slot 9)			

The following is a code example of the 01_pnio sample.

```
1.    void appl_loop(
2.        void
3.    )
4.    {
  … omit …

5.        if ((GOAL_TRUE == flgAppReady) && (plat_getElapseTime(tsTout) >= APPL_DM_TIMEOUT_TRIG_VAL)) {
6.
7.    /* read data from output module */
8.      res = goal_pnioDataOutputGet(pPnio, APPL_API, APPL_SLOT_04, APPL_SLOT_04_SUB_1, dataDm_2,
      APPL_SIZE_13_SUB_1_OUT, &iops);
9.      if (GOAL_RES_ERR(res)) {
10.         return;
11.     }
12.     /* copy data to input module */
13.     res = goal_pnioDataInputSet(pPnio, APPL_API, APPL_SLOT_03, APPL_SLOT_03_SUB_1, dataDm_2,
      APPL_SIZE_3_SUB_1_IN , GOAL_PNIO_IOXS_GOOD);
14.     if (GOAL_RES_ERR(res)) {
15.         return;
16.     }
17.
18.     /* read data from output module */
19.     res = goal_pnioDataOutputGet(pPnio, APPL_API, APPL_SLOT_02, APPL_SLOT_02_SUB_1, dataDm_1,
      APPL_SIZE_11_SUB_1_OUT, &iops);
20.     if (GOAL_RES_ERR(res)) {
21.         return;
22.     }
23.
24.     /* sample application: set LED */
25.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE1, (dataDm_1[0] & 0x01) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
26.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE2, (dataDm_1[0] & 0x02) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
27.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE3, (dataDm_1[0] & 0x04) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
28.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE4, (dataDm_1[0] & 0x08) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
29.
30.   /* sample application: get switch state */
31.   dataDm_1[0] = 0;
32.   for (uint8_t i = 0; i < 4; i++) {
33.       dataDm_1[0] |= (plat_remoteIoSwGet(i) << i);
34.   }
35.
36.   /* copy data to input module */
37.     res = goal_pnioDataInputSet(pPnio, APPL_API, APPL_SLOT_01, APPL_SLOT_01_SUB_1, dataDm_1,
      APPL_SIZE_1_SUB_1_IN, GOAL_PNIO_IOXS_GOOD);
38.     if (GOAL_RES_ERR(res)) {
39.         return;
40.     }
41.
  … omit …
42.
43.   }
```

[#1] get Mirror Data

[#2] set Mirror Data

[#3] get LED Data

[#4] set Switch Data

Application data update is controlled in the appl_loop() function, which is called in the uGOAL loop cycle. The data update cycle depends on the data transfer method.

In the sample software, the update cycle is as shown in Table 4-5, depending on the transfer method.

Table 4-5 sample soft update cycle

Data Transfer	Update cycle Define	Sample soft default [ms]	Sample	
			01	04
Cyclic	APPL_DM_TIMEOUT_TRIG_VAL	1	✓	✓
RPC	APPL_RPC_TIMEOUT_TRIG_VAL	310	-	✓

The data transfer using RPC process data to be sent and received at or above the PROFINET maximum send/receive data size of 69 bytes, but the update cycle of the application will increase. Refer to Table 4-6 to specify the update cycle according to the data size to be handled.

Table 4-6 Data size and update cycle comparison

Data Transfer	Data size [byte]	Average cycle [ms]
Cyclic	69	1
RPC	128	40
	256	70
	512	125
	1024	230
	1434	310

4.4 MRP

MRP (Media Redundancy Protocol) feature is enabled by default. When disable this feature, change the below.

(1) Slave program parameter

In the program file, change the argument of goal_pnioCfgFlgMrpCfgSet () from “GOAL_TRUE” to “GOAL_FALSE”.

```
1.  GOAL_STATUS_T appl_setup(
2.      void
3.  )
4.  {
    … omit …
5.
6.      /* enable MRP */
7.      res = goal_pnioCfgFlgMrpCfgSet(GOAL_TRUE);
8.      if (GOAL_RES_ERR(res)) {
9.          goal_logErr("failed to enable MRP");
10.         return res;
11.     }
12.
    … omit …
13. }
```

(2) GSDML file

Delete or comment out the line starting with “MediaRedundancy…”.

```
1.  <SystemDefinedSubmoduleList>
     … omit …

2.      </ApplicationRelations>
3.        <MediaRedundancy AdditionalProtocolsSupported="true" SupportedRole="Client"/>
4.    </InterfaceSubmoduleItem>
```

5. EtherNet/IP

The profile settings that you should set in this sample are shown below.

Slave profile information is provided to the master device in EDS (Electric Data Sheet) file, which provides device information about the device. Therefore, the device program parameter in the slave and the parameter in the EDS file must match.

Slave : appl\02_eip\goal_appl,h

EDS file : appl\02_eip\eds\eip_goal_renesas.eds

EDS files can be created and modified using a text editor, but can be easily created and modified using EZ-EDS provided by Open DeviceNet Vendor Association, Inc. (ODVA).

Order EZ-EDS | ODVA Network Specifications

For details on the API shown below, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

5.1 Device Identity

This section describes vendor ID, vendor name, product code, etc.

These settings are defined in the program file.

Table 5-1 EtherNet/IP Device Identify

Item	Macro	Default Value [02_eip]
Vendor ID	APPL_EIP_VENDOR_ID	1105
Vendor Name	APPL_EIP_VENDOR	Renesas Electronics
Product Code	APPL_EIP_PRODUCT_CODE	768
Product Name	APPL_EIP_PRODUCT	R-IN32M3 Module
IP address	MAIN_APPL_IP	192.168.0.100

5.1.1 Vendor ID

The vendor ID is assigned by ODVA, and the default value is 1105 (Renesas Electronics).

(1) Slave program parameter

```
#ifndef APPL_EIP_VENDOR_ID
# define  APPL_EIP_VENDOR_ID   (1105)          /**< vendor id */
#endif
```

(2) EDS file

Change the value of "VendorCode" below.

```
[Device]
        VendCode = 1105;
        VendName = "Renesas Electronics";
        ProdType = 43;
        ProdTypeStr = "Generic Device";
        ProdCode = 768;
        MajRev = 1;
        MinRev = 1;
        ProdName = "R-IN32M3 Module";
```

< EZ-EDS 表示>

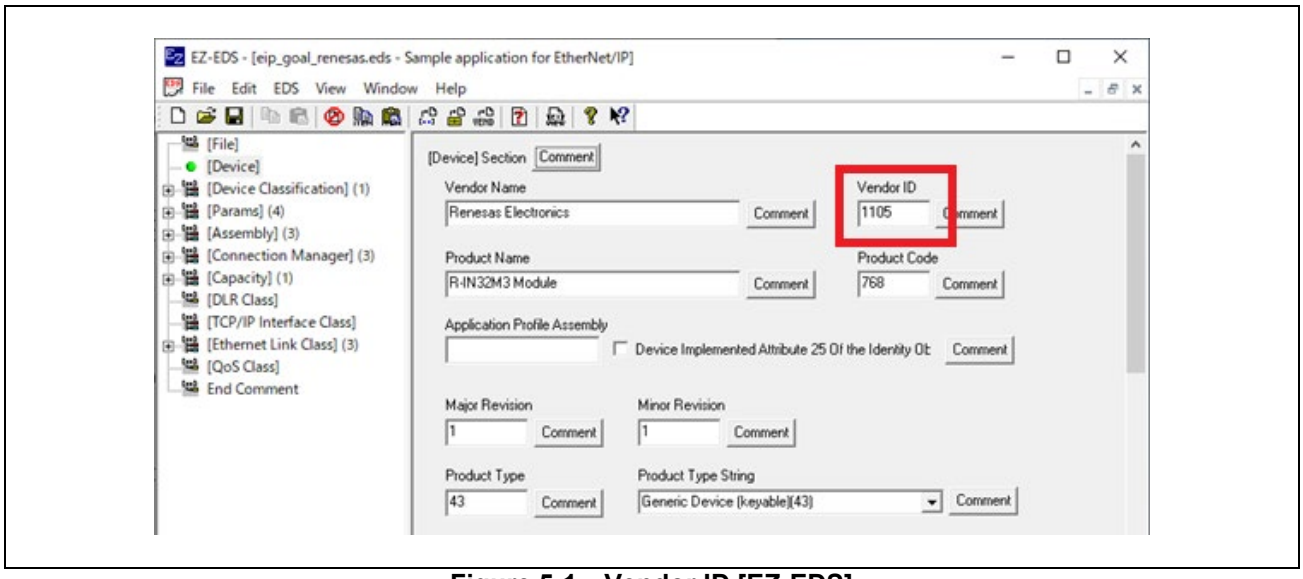

Figure 5-1 Vendor ID [EZ-EDS]

5.1.2 Vendor Name

The default value is “Renesas Electronics” (Renesas Electronics).

(1) Slave program parameter

```
#ifndef APPL_EIP_VENDOR
# define APPL_EIP_VENDOR   Renesas Electronics    /**< vendor name */
#endif
```

(2) EDS file

Change the value of "VendorName" below.

```
[Device]
        VendCode = 1105;
        VendName = "Renesas Electronics";
        ProdType = 43;
        ProdTypeStr = "Generic Device";
        ProdCode = 768;
        MajRev = 1;
        MinRev = 1;
        ProdName = "R-IN32M3 Module";
```

< EZ-EDS 表示 >

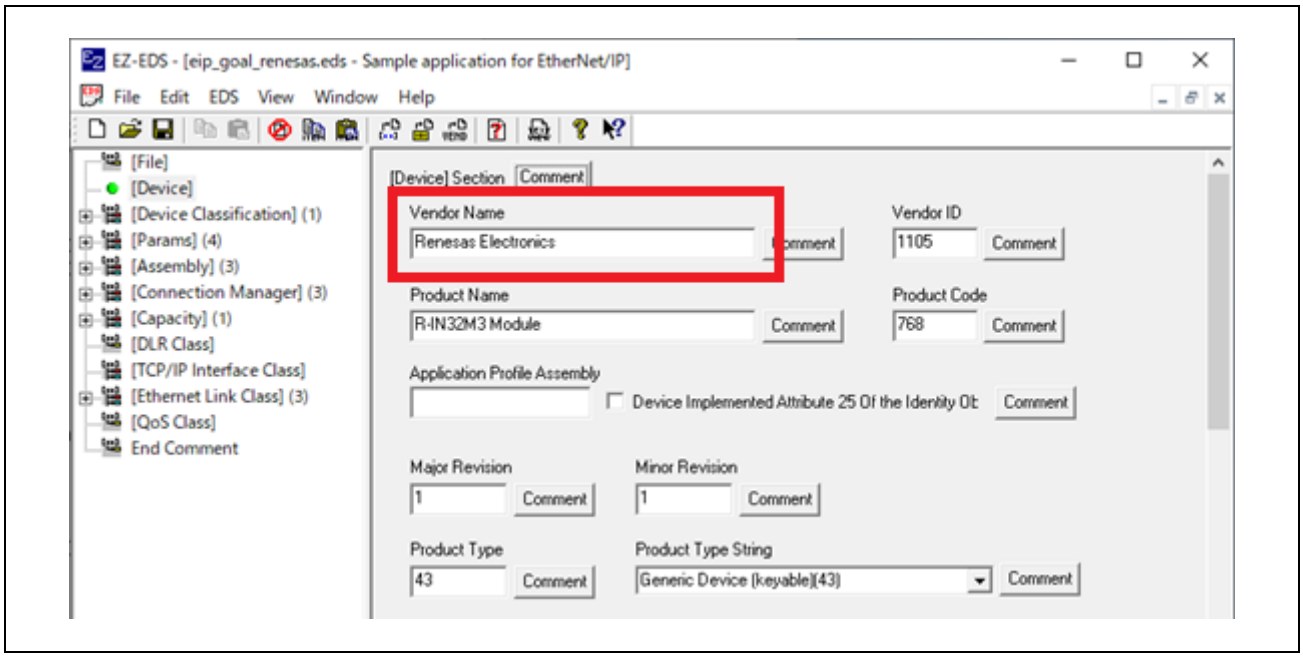

Figure 5-2 Vendor Name [EZ-EDS]

5.1.3 Product Code

The default value is 768 (0x0300).

(1) Slave program parameter

```
#ifndef APPL_EIP_PRODUCT_CODE
# define APPL_EIP_PRODUCT_CODE (768)          /**< product code */
#endif
```

(2) EDS file

Change the value of "ProdCode" below.

```
[Device]
        VendCode = 1105;
        VendName = "Renesas Electronics";
        ProdType = 43;
        ProdTypeStr = "Generic Device";
        ProdCode = 768;
        MajRev = 1;
        MinRev = 1;
        ProdName = "R-IN32M3 Module";
```

< EZ-EDS 表示 >

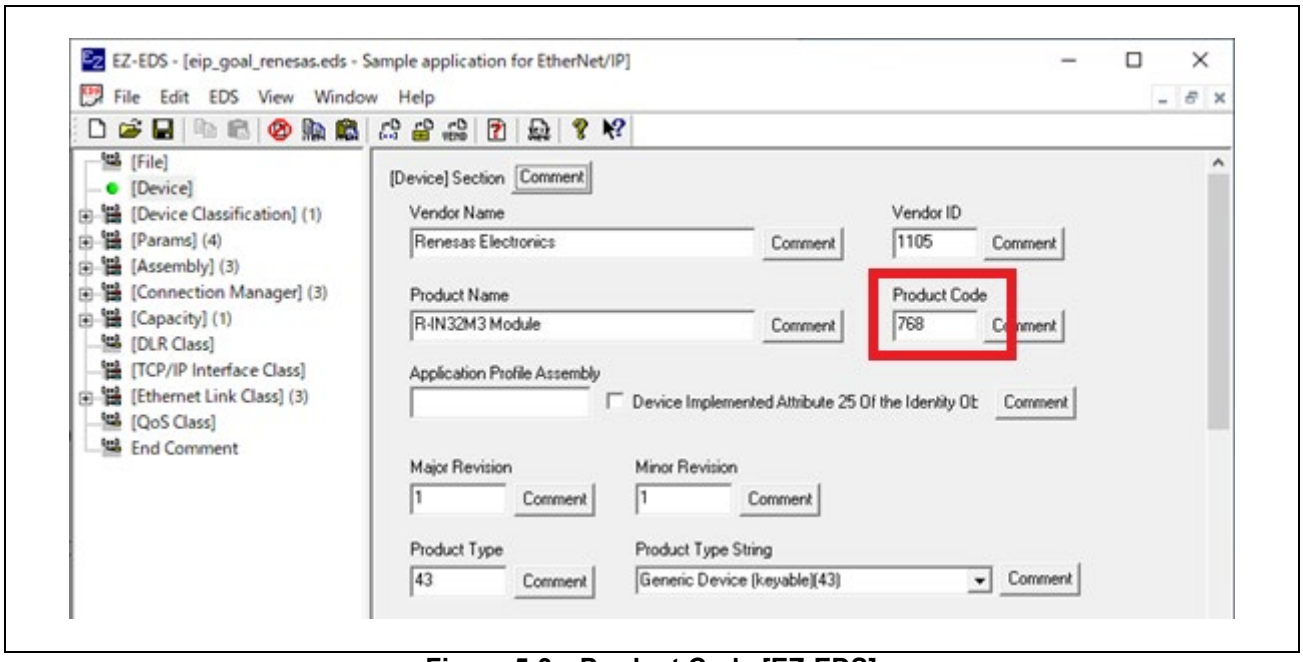

Figure 5-3 Product Code [EZ-EDS]

5.1.4 Product Name

The default value is “R-IN32M3_Module”.

(1) Slave program parameter

```
#ifndef APPL_EIP_PRODUCT
# define  APPL_EIP_PRODUCT  R-IN32M3 Module      /**< product name */
#endif
```

(2) EDS file

Change the value of "ProdName" below.

```
[Device]
        VendCode = 1105;
        VendName = "Renesas Electronics";
        ProdType = 43;
        ProdTypeStr = "Generic Device";
        ProdCode = 768;
        MajRev = 1;
        MinRev = 1;
        ProdName = "R-IN32M3 Module";
```

< EZ-EDS 表示 >

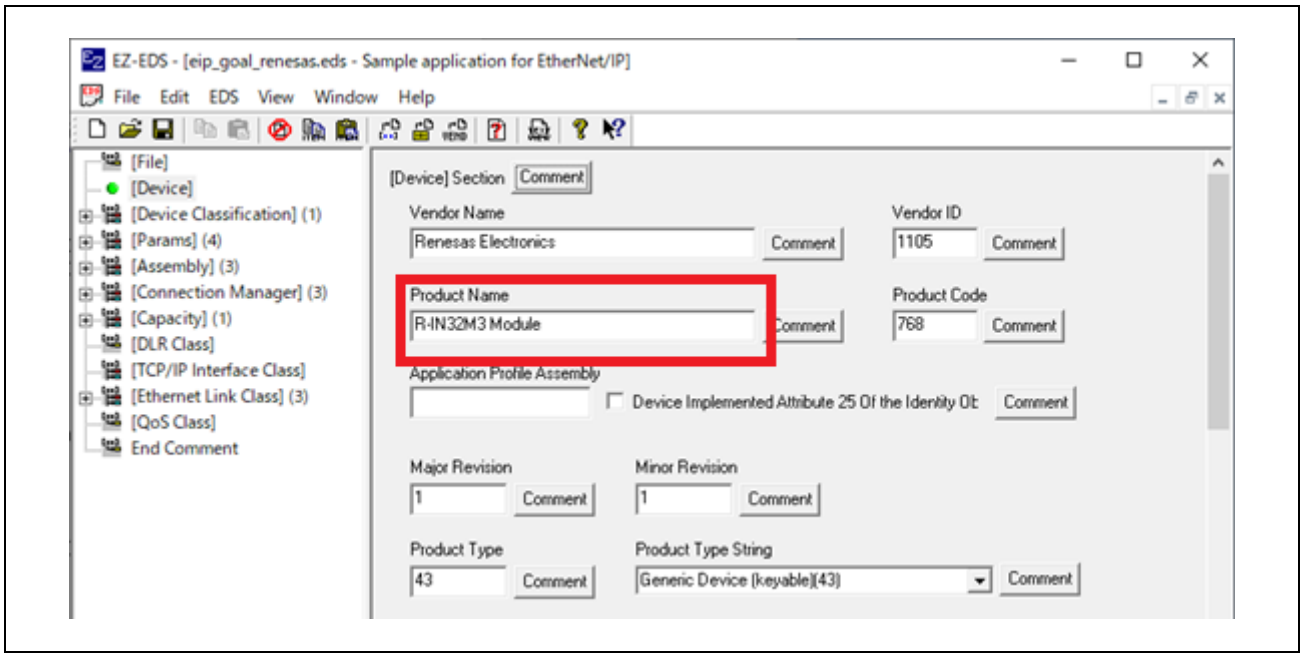

Figure 5-4 Product Name [EZ-EDS]

5.1.5 IP Address

Any IP address can be set for evaluation purposes, the IP address value is defined by a macro in the following file. For details on the setting method, refer to Chapter A of Appendix.

(1) Slave program parameter

```
1. #define MAIN_APPL_IP      GOAL_NET_IPV4(192, 168, 0, 100)
2. #define MAIN_APPL_NM      GOAL_NET_IPV4(255, 255, 255, 0)
3. #define MAIN_APPL_GW      GOAL_NET_IPV4(0, 0, 0, 0)
```

(2) EDS file

No configuration for EDS file.

5.2 Data Model

The Input/Output data and size to be set according to the application are described.

5.2.1 Output Data

The following is a sample software configuration for data received from the scanner device.

Table 5-2 Output configuration

sample		variable	Assembly ID	size
05_eip_large	02_eip	ledRequest	GOAL_APP_ASM_ID_OUTPUT_DM1 (150)	GOAL_APP_ASM_SIZE_OUTPUT_DM1 (1)
		outData[]	GOAL_APP_ASM_ID_OUTPUT_DM2 (151)	GOAL_APP_ASM_SIZE_OUTPUT_DM2 (16)
	-	rpcData[]	GOAL_APP_ASM_ID_OUTPUT_RPC1 (152)	GOAL_APP_ASM_SIZE_OUTPUT_RPC (32)
		rpcData[]	GOAL_APP_ASM_ID_OUTPUT_RPC2 (153)	GOAL_APP_ASM_SIZE_OUTPUT_RPC (32)
		rpcData[]	GOAL_APP_ASM_ID_OUTPUT_RPC3 (154)	GOAL_APP_ASM_SIZE_OUTPUT_RPC (32)

To change Output data size, it is necessary to change the program file and EDS file.

(1) Slave program parameter

The setting value is defined by a macro in the following file, so change that value.

```
1.  #define  GOAL_APP_ASM_SIZE_OUTPUT_DM1    (1)      /**< Output Assembly Data */
```

(2) EDS file

Change the contents of the following "Assem150".

```
1.  [Assembly]
    … omit …

2.      Assem150 =
5.          "Output Assembly",
6.          "",
7.          1,
8.          0x0000,
9.          ,,
10.         8,Param150,
    … omit …
```

< EZ-EDS 表示 >

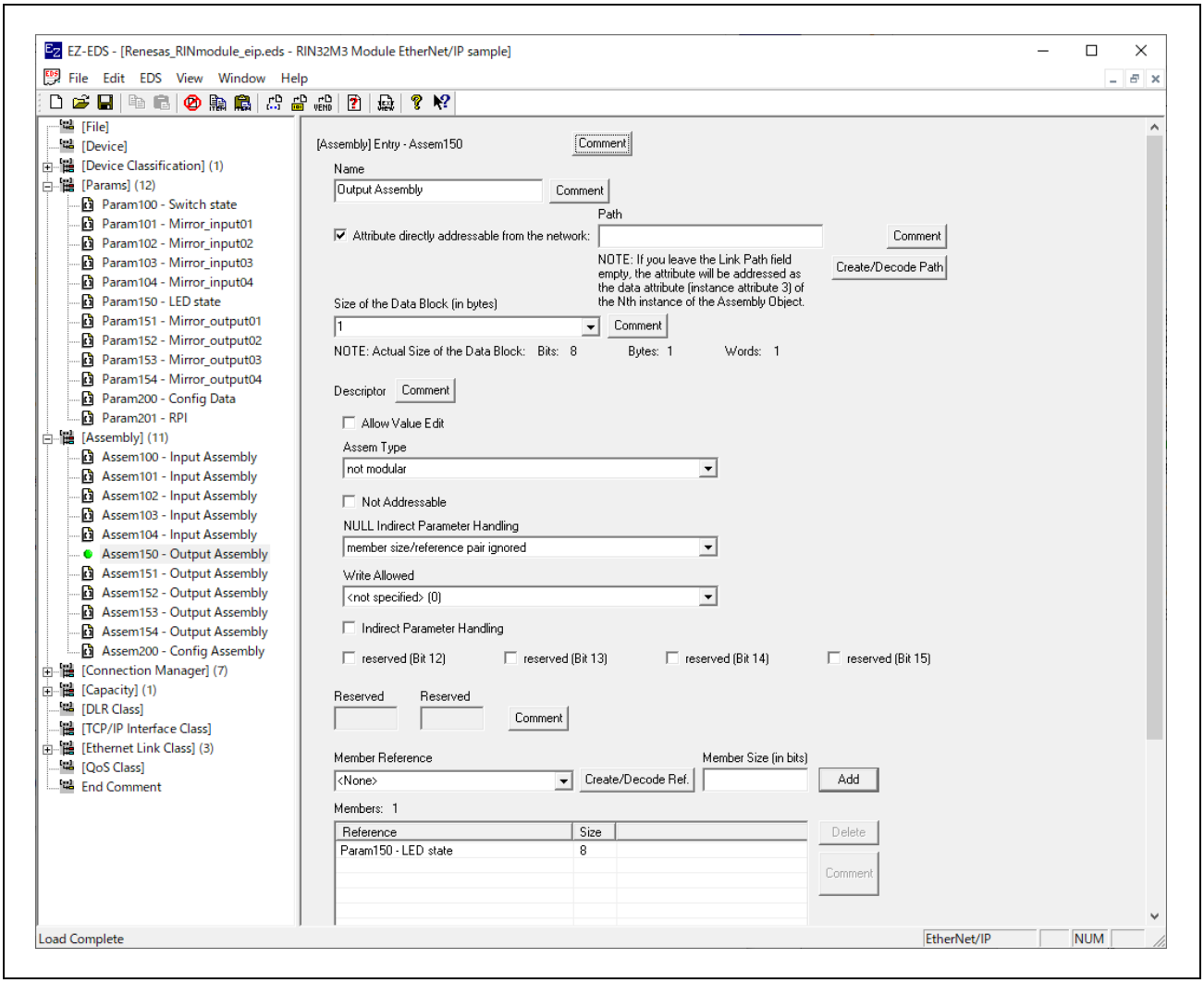

Figure 5-5 Output Data – Assem150 [EZ-EDS]

5.2.2 Input Data

The following is a sample software configuration for data send from the adapter (R-IN32M3 module) device.

ュール)がスキャナへ送信するデータを定義します。

Table 5-3 Input configuration

sample		variable	Assembly ID	size
05_eip_large	02_eip	ledRequest	GOAL_APP_ASM_ID_INPUT_DM1 (100)	GOAL_APP_ASM_SIZE_INPUT_DM1 (1)
		outData[]	GOAL_APP_ASM_ID_ INPUT_DM2 (101)	GOAL_APP_ASM_SIZE_ INPUT_DM2 (16)
	'	rpcData[]	GOAL_APP_ASM_ID_ INPUT_RPC1 (102)	GOAL_APP_ASM_SIZE_ INPUT_RPC (32)
		rpcData[]	GOAL_APP_ASM_ID_ INPUT_RPC2 (103)	GOAL_APP_ASM_SIZE_ INPUT_RPC (32)
		rpcData[]	GOAL_APP_ASM_ID_ INPUT_RPC3 (104)	GOAL_APP_ASM_SIZE_ INPUT_RPC (32)

To change Input data size, it is necessary to change the program file and EDS file.

(1) Slave program parameter

The setting value is defined by a macro in the following file, so change that value.

```
1.  #define GOAL_APP_ASM_SIZE_INPUT_DM1    (1)     /**< Input Assembly Data */
```

(2) EDS file

Change the contents of the following "Assem100".

```
1.  [Assembly]
2.      Object_Name = "Assembly Object";
3.      Object_Class_Code = 0x04;
4.      Number_Of_Static_Instances = 6;
5.      Assem100 =
11.         "Input Assembly",
12.         "",
13.         1,
14.         0x0000,
15.         ,,
16.         8,Param100,
    … omit …
```

< EZ-EDS 表示 >

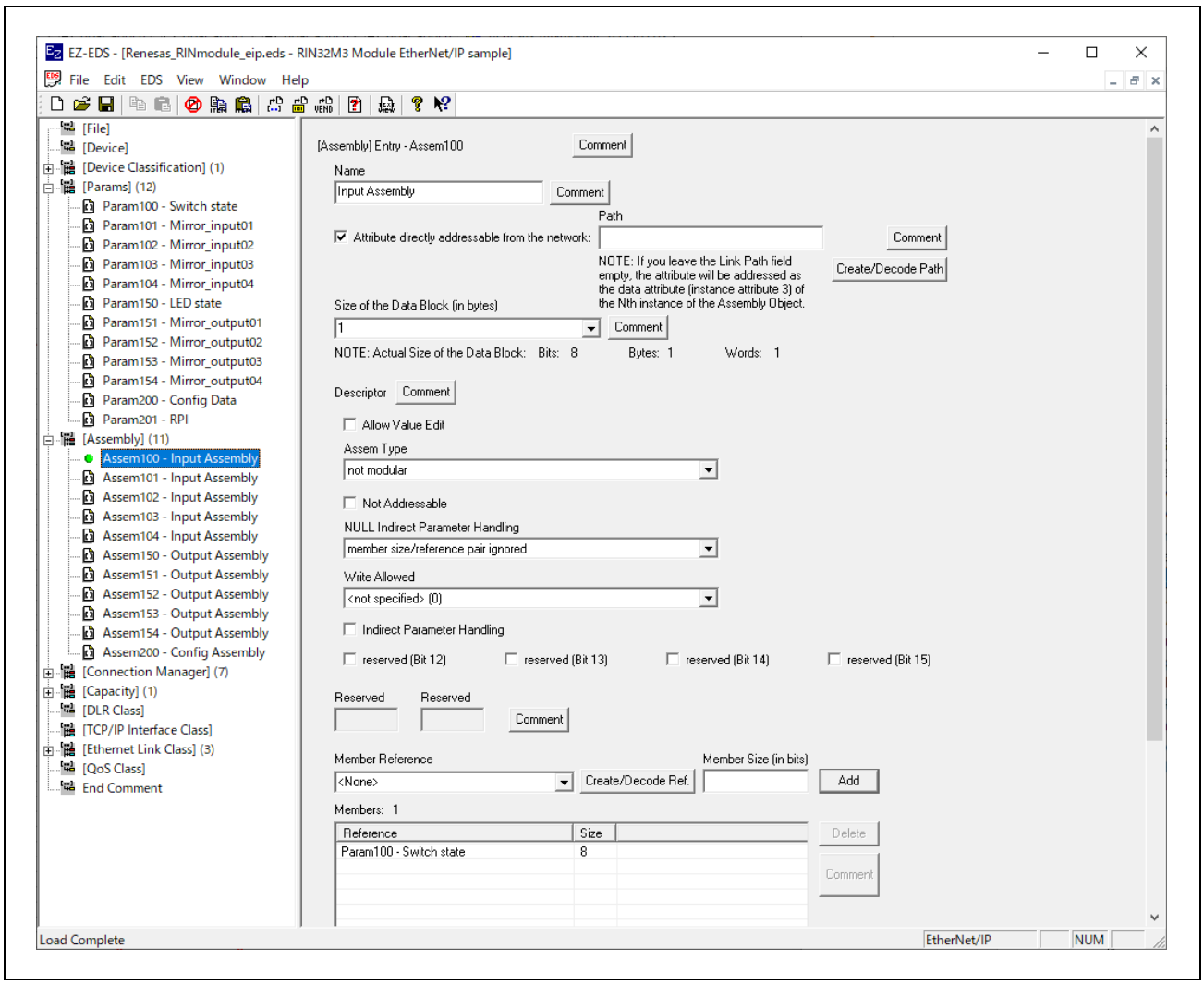

Figure 5-6 Input Data – Assem100 [EZ-EDS]

5.3 User application Output/Input Data Handling

The sample software implements two types of data transmission/reception applications as example applications.

- **Remote-IO (LED/Switch):** LED lighting control and Switch status from the evaluation board
- **Mirror:** Sends data received from the master and mirrored back

Table 5-4 API and Sample application data variable

sample		Sample app.	Data variable	Purpose	API	reference
05_eip_large	02_eip	get LED Data [#1]	ledRequest	Reception	goal_eipAssemblyObjectRead()	5.2.1 Output Data
		get Mirror Data [#3]	outData[]			
		set Switch Data [#2]	swStatus	Transmission	goal_eipAssemblyObjectWrite()	5.2.2 Input Data
		set Mirror data [#4]	inData[]			
	-	get Mirror Data_1	rpcData[]	Reception	goal_eipAssemblyObjectRead()	5.2.1 Output Data
		get Mirror Data_2	rpcData[]			
		get Mirror Data_3	rpcData[]			
		set Mirror Data_1	rpcData[]	Transmission	goal_eipAssemblyObjectWrite()	5.2.2 Input Data
		set Mirror Data_2	rpcData[]			
		set Mirror Data_3	rpcData[]			

The following is a code example of the 02_eip sample.

```
1.  void appl_loop(
2.      void
3.  )
4.  {
5.      GOAL_STATUS_T res;                  /* result */
6.
7.      if ((GOAL_TRUE == flgAppReady) && (plat_getElapseTime(tsTout) >=
    APPL_DM_TIMEOUT_TRIGGER_VAL))
8.      {
9.  /** USER APPLICATION --> **/
10.     uint8_t ledRequest;
11.     uint8_t swStatus;
12.
13.     /* GET Output Assembly Data (O->T) */
14.     res = goal_eipAssemblyObjectRead(pHdlEip, GOAL_APP_ASM_ID_OUTPUT_DM1, &ledRequest,
    GOAL_APP_ASM_SIZE_OUTPUT_DM1);
15.
16.     /* sample application(RemoteIO) set LED */
17.   goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE1, (ledRequest & 0x01) ?
    GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
18.   goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE2, (ledRequest & 0x02) ?
    GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
19.   goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE3, (ledRequest & 0x04) ?
    GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
20.   goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE4, (ledRequest & 0x08) ?
    GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
21.
22.     /* sample application(RemoteIO) get Switch */
23.     if (GOAL_RES_OK(res)) {
24.         swStatus = 0;
25.         for (uint8_t i = 0; i < 4; i++) {
26.             swStatus |= (plat_remoteIoSwGet(i) << i);
27.     }
28.     /* SET Input Assembly Data (T->O) */
29.     res = goal_eipAssemblyObjectWrite(pHdlEip, GOAL_APP_ASM_ID_INPUT_DM1, &swStatus,
    GOAL_APP_ASM_SIZE_INPUT_DM1);
30.   }
31.
32.
33.     /* GET Output Assembly Data (O->T) */
34.     if (GOAL_RES_OK(res)) {
35.     res = goal_eipAssemblyObjectRead(pHdlEip, GOAL_APP_ASM_ID_OUTPUT_DM2, &outData[0],
    GOAL_APP_ASM_SIZE_OUTPUT_DM2);
36.   }
37.
38.   /* mirror output data to input data */
39.   if (GOAL_RES_OK(res)) {
40.     GOAL_MEMCPY(&inData[0], &outData[0], GOAL_APP_ASM_SIZE_OUTPUT_DM2);
41.
42.     /* SET Input Assembly Data (T->O) */
43.     res = goal_eipAssemblyObjectWrite(pHdlEip, GOAL_APP_ASM_ID_INPUT_DM2, &inData[0],
    GOAL_APP_ASM_SIZE_INPUT_DM2);
44.   }
45.   /** <-- USER APPLICATION **/
46.
47.     /* update base timestamp */
48.     tsDmTout = goal_timerTsGet();
49.     }
    … omit …
50.   }
```

[#1] get LED Data

[#2] set Switch Data

[#3] get Mirror Data

[#4] set Mirror Data

Application data update is controlled in the appl_loop() function, which is called in the uGOAL loop cycle. The data update cycle depends on the data transfer method.

In the sample software, the update cycle is as shown in Table 5-5, depending on the transfer method.

Table 5-5 sample soft update cycle

Data Transfer	Update cycle Define	Sample soft default [ms]	Sample	
			02	05
Cyclic	APPL_DM_TIMEOUT_TRIG_VAL	1	✓	✓
RPC	APPL_RPC_TIMEOUT_TRIG_VAL	125	-	✓

The data transfer using RPC process data to be sent and received at or above the EtherNet/IP maximum send/receive data size of 69 bytes, but the update cycle of the application will increase. Refer to Table 5-6 to specify the update cycle according to the data size to be handled.

Table 5-6 Data size and update cycle comparison

Data Transfer	Data size [byte]	Average cycle [ms]
Cyclic	69	1
RPC	128	40
	256	70
	495	125

6. EtherCAT

The profile settings that you should set in this sample are shown below.

Slave profile information is provided to the master device in ESI (EtherCAT Slave Information) file, which provides device information about the device. Therefore, the device program parameter in the slave and the parameter in the ESI file must match.

Slave :	appl\03_ecat\goal_appl.h
Object dictionary :	appl\03_ecat\goal_appl_ecat_objects.c
ESI file :	appl\03_ecat\esi\Renesas_RINmodule_03ecat.xml

The XML editor can be used to create and modify ESI files, and the Conformance Test Tool (CTT) provided by EtherCAT Technology Group (ETG) can be used to check the validity of the registration information.

Conformance Test Tool | EtherCAT Technology Group

[memo]
XML Notepad [Microsoft Open Source] or other editors that can display XML in a tree view make it easy to edit nodes and definitions.
XmlNotepad (microsoft.github.io)

In EtherCAT, profile setting information must be defined in an information table called an object dictionary in addition to the program file and ESI file. This object dictionary is an information table that bridges the communication part and the application part. This table covers the specifications handled by the device and uses a set of parameters to set functions and monitor setting information.

Here is an example of the object dictionary for the sample software, where the Default value represents the value of the 03_ecat sample.

Table 6-1 Object dictionary (1/3)

Index	Name	Subindex	Data type	Default value	Sample	
					03	06
0x1000	**Device Type**	0x00	UINT32	0x00001389	✓	✓
0x1001	**Error Register**	0x00	UINT8	0x00	✓	✓
0x1008	**Manufacturer Device Name**	0x00	STRING	RIN32M3 Module	✓	✓
0x1009	**Manufacturer Hardware Version**	0x00	STRING	1.0.00	✓	✓
0x100A	**Manufacturer Software Version**	0x00	STRING	1.0.30	✓	✓
0x1018	**Identity Object**	0x00	UINT8	0x04	✓	✓
	Vendor Id	0x01	UINT32	0x00000766	✓	✓
	Product Code	0x02	UINT32	0x00000800 *	✓	✓
	Revision Number	0x03	UINT32	0x000000002	✓	✓
	Serial Number	0x04	UINT32	0x000001234	✓	✓
0x1600	**Receive PDO Mapping Parameter 1**	0x00	UINT8	0x01	✓	✓
	Mapping Entry 1	0x01	UINT32	0x62000108	✓	✓
0x1601	**Receive PDO Mapping Parameter 2**	0x00	UINT8	0x01	✓	✓
	Mapping Entry 1	0x01	UINT32	0x62010180	✓	✓
0x1700	**Receive PDO Mapping Parameter 3**	0x00	UINT8	0x03	-	✓
	Mapping Entry 1	0x01	UINT32	0x621001F8	-	✓
	Mapping Entry 2	0x02	UINT32	0x621002F8	-	✓
	Mapping Entry 3	0x03	UINT32	0x621003F8	-	✓
0x1A00	**Transmit PDO Mapping Parameter 1**	0x00	UINT8	0x01	✓	✓
	Mapping Entry 1	0x01	UINT32	0x60000108	✓	✓
0x1A01	**Transmit PDO Mapping Parameter 2**	0x00	UINT8	0x00	✓	✓
	Mapping Entry 1	0x01	UINT32	0x60010180	✓	✓
0x1B00	**Transmit PDO Mapping Parameter 3**	0x00	UINT8	0x03	-	✓
	Mapping Entry 1	0x01	UINT32	0x601001F8	-	✓
	Mapping Entry 2	0x02	UINT32	0x601002F8	-	✓
	Mapping Entry 3	0x03	UINT32	0x601003F8	-	✓

* 06_ecat_largesize: Product Code = 0x00000804

Table 6-2 Object dictionary (2/3)

Index	Name	Subindex	Data type	Default value	Sample	
					03	06
0x1C00	**Sync Manager Communication Type**	0x00	UINT8	0x04	✓	✓
	Communication Type Sync Manager 0	0x01	UINT8	0x01	✓	✓
	Communication Type Sync Manager 1	0x02	UINT8	0x02	✓	✓
	Communication Type Sync Manager 2	0x03	UINT8	0x03	✓	✓
	Communication Type Sync Manager 3	0x04	UINT8	0x04	✓	✓
0x1C12	**Sync Manager 2 PDO Assignment**	0x00	UINT8	0x02 *	✓	✓
	PDO Mapping 1	0x01	UINT16	0x1600	✓	✓
	PDO Mapping 2	0x02	UINT16	0x1601	✓	✓
	PDO Mapping 3	0x03	UINT16	0x1700	-	✓
0x1C13	**Sync Manager 3 PDO Assignment**	0x00	UINT8	0x02 *	✓	✓
	PDO Mapping 1	0x01	UINT16	0x1A00	✓	✓
	PDO Mapping 2	0x02	UINT16	0x1A01	✓	✓
	PDO Mapping 3	0x03	UINT16	0x1B00	-	✓
0x1C32	**Output SyncManager parameter**	0x00	UINT8	0x20	✓	✓
	Synchronization Type	0x01	UINT16	0x0000	✓	✓
	Cycle Time	0x02	UINT32	0x00000000	✓	✓
	Synchronization Types Supported	0x04	UINT16	0x0000	✓	✓
	Minimum Cycle Time	0x05	UINT32	0x00000000	✓	✓
	Calc and Copy Time	0x06	UINT32	0x00000000	✓	✓
	SM-Event Missed Counter	0x0B	UINT16	0x0000	✓	✓
	Cycle Time Too Small	0x0C	UINT16	0x0000	✓	✓
	Sync Error	0x20	UINT8	0x00	✓	✓
0x1C33	**Input SyncManager parameter**	0x00	UINT8	0x20	✓	✓
	Synchronization Type	0x01	UINT16	0x0000	✓	✓
	Cycle Time	0x02	UINT32	0x00000000	✓	✓
	Synchronization Types Supported	0x04	UINT16	0x0000	✓	✓
	Minimum Cycle Time	0x05	UINT32	0x00000000	✓	✓
	Calc and Copy Time	0x06	UINT32	0x00000000	✓	✓
	SM-Event Missed Counter	0x0B	UINT16	0x0000	✓	✓
	Cycle Time Too Small	0x0C	UINT16	0x0000	✓	✓
	Sync Error	0x20	UINT8	0x00	✓	✓

* 06_ecat_largesize: Mapping Entry = 0x03

Table 6-3 Object dictionary (3/3)

Index	Name	Subindex	Data type	Default value	Sample	
					03	06
0x6000	**Read Input data**	0x00	UINT8	0x01	✓	✓
	Switch Input 1-8	0x01	UINT8	0x00	✓	✓
0x6001	**data in (dm)**	0x00	UINT8	0x01	✓	✓
	data in (dm) 1-16	0x01	ARRAY[0:15]	0x00	✓	✓
0x6010	**data in (rpc)**	0x00	UINT8	0x03	-	✓
	data in (rpc) 1-31	0x01	ARRAY[0:30]	0x00	-	✓
	data in (rpc) 32-62	0x02	ARRAY[0:30]	0x00	-	✓
	data in (rpc) 63-93	0x03	ARRAY[0:30]	0x00	-	✓
0x6200	**Write Output data**	0x00	UINT8	0x01	✓	✓
	LED Output 1-8	0x01	UINT8	0x00	✓	✓
0x6201	**data out (dm)**	0x00	UINT8	0x01	✓	✓
	data out (dm) 1-16	0x01	UINT16	0x00	✓	✓
0x6210	**data in (rpc)**	0x00	UINT8	0x03	-	✓
	data out (rpc) 1-31	0x01	ARRAY[0:30]	0x00	-	✓
	data out (rpc) 32-62	0x02	ARRAY[0:30]	0x00	-	✓
	data out (rpc) 63-93	0x03	ARRAY[0:30]	0x00	-	✓

For details on the API shown below, see "R-IN32M3 Module (RY9012A0) User's Manual Software (R17US0002ED ****)".

6.1 Device Identity

This section describes vendor ID, vendor name, product code, etc.

These settings are defined in the program file.

Table 6-4 EtherCAT parameter macro

Parameter	Macro	Default value [03_ecat]
Vendor ID	APPL_ECAT_VENDOR_ID	0x00000766
Vendor Name	APPL_ECAT_VENDOR	Renesas Electronics Corp.
Product Code	APPL_ECAT_PRODUCT_CODE	0x00000800 *
Revision	APPL_ECAT_REVISION_NUMBER	0x00000002
Serial	APPL_ECAT_SERIAL_NUMBER	0x00001234

* 06_ecat_largesize: Product Code = 0x00000804

6.1.1 Vendor ID

The default value is 0x0766 (Renesas Electronics).

(1) Slave program parameter

```
#ifndef APPL_ECAT_VENDOR_ID
# define  APPL_ECAT_VENDOR_ID   (0x00000766)       /**< Vendor ID */
#endif
```

(2) Object Dictionary

For object dictionary file, “Index = 0x1018 (Identity Object) and Subindex = 0x01” is the information about Vendor ID. No setting is required in the object dictionary file because the value of “APPL_ECAT_VENDOR_ID” is reflected in the value of "uint32ValueDef" below.

```
/***********************************************/
/* 1018 - Identity Object                      */
/***********************************************/
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1018, GOAL_ECAT_OBJCODE_RECORD, 0x0023);
}
 … omit …

if (GOAL_RES_OK(res)) {

    uint32ValueMin = 0x0;
    uint32ValueDef = APPL_ECAT_VENDOR_ID;
    uint32ValueMax = 0xFFFFFFFF;

    res = goal_ecatdynOdSubIndexAdd(
        pHdlEcat,
        0x1018,
        0x01,
        GOAL_ECAT_DATATYPE_UNSIGNED32,
        EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC,
        (uint8_t *) &uint32ValueDef,
        (uint8_t *) &uint32ValueMin,
        (uint8_t *) &uint32ValueMax,
        4,
        NULL);
}
```

(3) ESI file

Change the value of "Id" and “DefaultValue” below.

```
<Vendor>
  <Id>#x00000766</Id>
  <Name>Renesas Electronics Corp.</Name>
</Vendor>
```

```
<Object>
  <Index>#x1018</Index>
  <Name>Identity Object</Name>
… omit …
    <SubItem>
      <Name>Vendor Id</Name>
      <Info>
        <DefaultValue>#x00000766</DefaultValue>
      </Info>
    </SubItem>
```

< XML tree view >

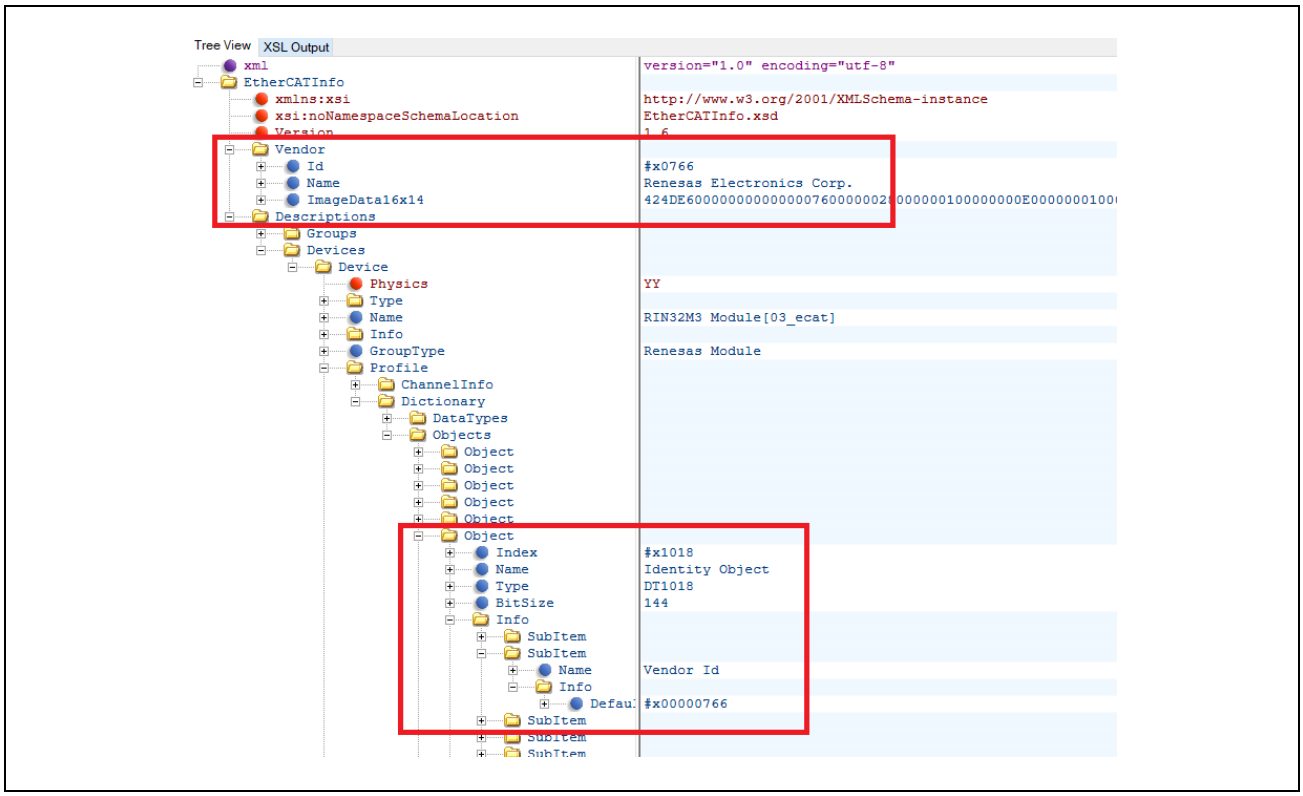

Figure 6-1 Vendor ID

6.1.2 Vendor Name

The default value is “Renesas Electronics” (Renesas Electronics).

(1) Slave program parameter

```
1.  #ifndef APPL_ECAT_VENDOR
2.  # define  APPL_ECAT_VENDOR   Renesas Electronics Corp.    /**< Vendor Name */
3.  #endif
```

(2) Object Dictionary

No setting is required in the object dictionary file.

(3) ESI file

Change the value of "Name" below.

```
1.  #ifndef APPL_ECAT_VENDOR
2.  # define  APPL_ECAT_VENDOR   Renesas Electronics Corp.    /**< Vendor Name */
3.  #endif
```

< XML tree view >

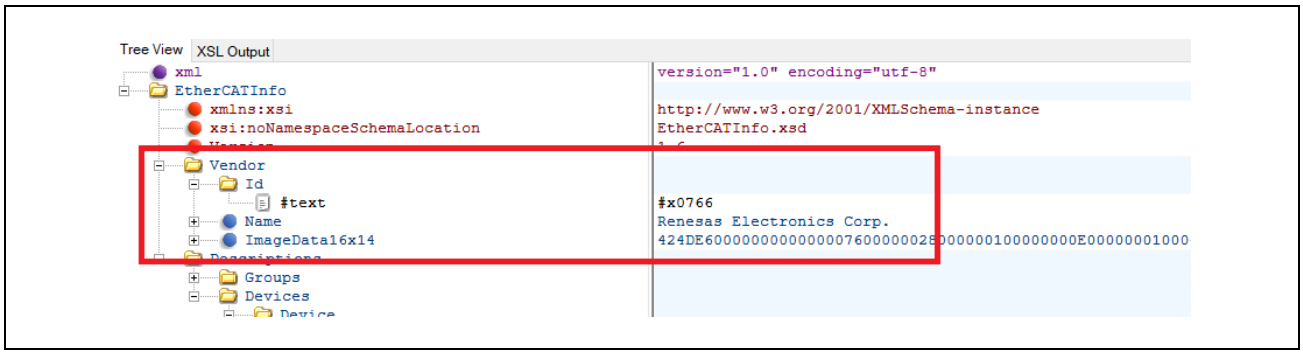

Figure 6-2 Vendor Name

6.1.3 Product Code

The default value is 0x0800.

(1) Slave program parameter

```
#ifndef APPL_ECAT_PRODUCT_CODE
# define  APPL_ECAT_PRODUCT_CODE    (0x00000800)     /**< Product code */
#endif
```

(2) Object Dictionary

For object dictionary file, “Index = 0x1018 (Identity Object) and Subindex = 0x02” is the information about Product Code. No setting is required in the object dictionary file because the value of “APPL_ECAT_PRODUCT_CODE” is reflected in the value of "uint32ValueDef" below.

```
    /*********************************************/
    /* 1018 - Identity Object                    */
    /*********************************************/
    if (GOAL_RES_OK(res)) {
        res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1018, GOAL_ECAT_OBJCODE_RECORD, 0x0023);
    }
  … omit …

    if (GOAL_RES_OK(res)) {

        uint32ValueMin = 0x0;
        uint32ValueDef = APPL_ECAT_PRODUCT_CODE;
        uint32ValueMax = 0xFFFFFFFF;

        res = goal_ecatdynOdSubIndexAdd(
            pHdlEcat,
            0x1018,
            0x02,
            GOAL_ECAT_DATATYPE_UNSIGNED32,
            EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC,
            (uint8_t *) &uint32ValueDef,
            (uint8_t *) &uint32ValueMin,
            (uint8_t *) &uint32ValueMax,
            4,
            NULL);
    }
```

(3) ESI file

Change the value of "ProductCode" and “DefaultValue” below.

```
1.  <Devices>
2.    <Device Physics="YY">
3.      <Type ProductCode="#x00000800" RevisionNo="#x000000001">Renesas Module</Type>
4.      <Name>RIN32M3 Module[Remote-03_ecat]</Name>
5.      <info>
```

```
1.  <Object>
2.    <Index>#x1018</Index>
3.    <Name>Identity Object</Name>
      … omit …
4.      <SubItem>
5.        <Name>Product Code</Name>
6.        <Info>
7.          <DefaultValue>#x00000800</DefaultValue>
8.        </Info>
9.      </SubItem>
```

< XML tree view >

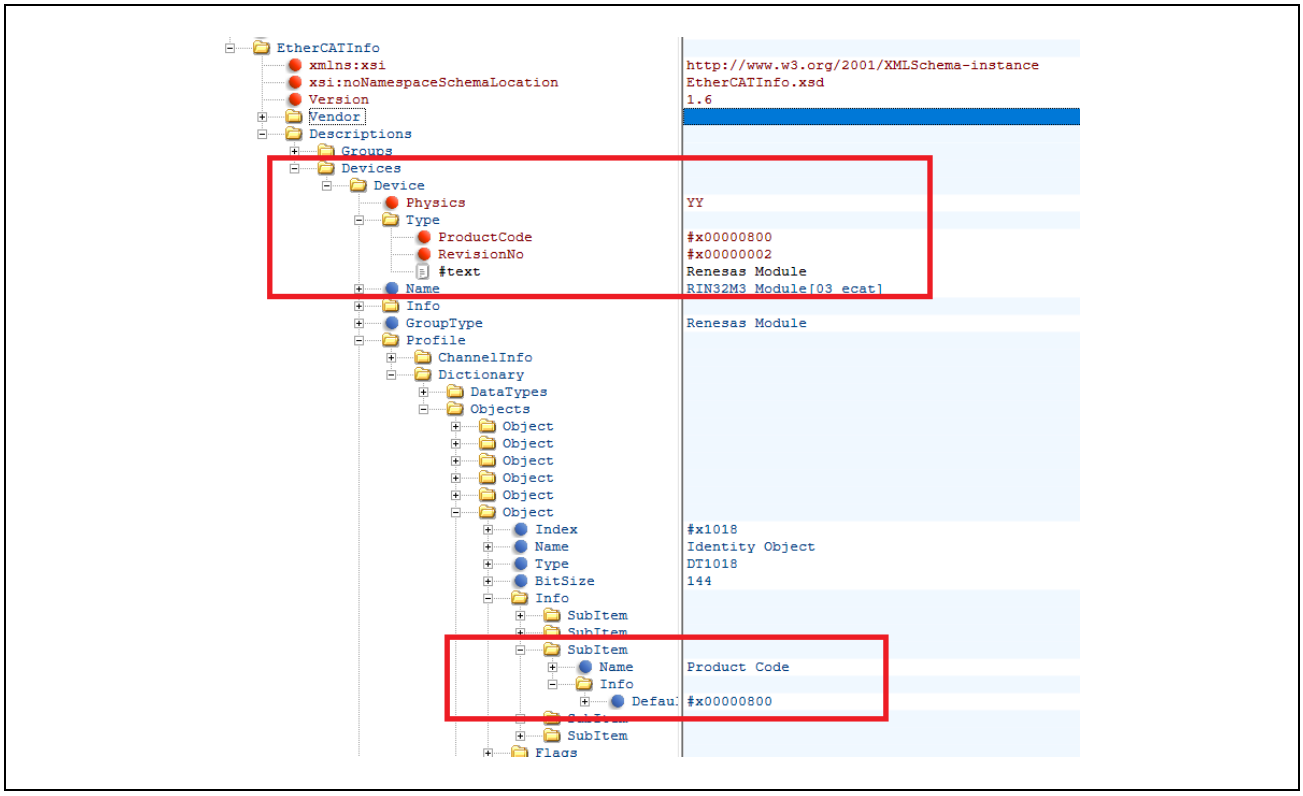

Figure 6-3 Product Code

6.1.4 Revision Number

The default value is 0x00000002.

(1) Slave program parameter

```
#ifndef APPL_ECAT_REVISION_NUMBER
# define  APPL_ECAT_REVISION_NUMBER   (0x00000002)    /**< Revision Number */
#endif
```

(2) Object Dictionary

For object dictionary file, “Index = 0x1018 (Identity Object) and Subindex = 0x03” is the information about Revision Number. No setting is required in the object dictionary file because the value of “APPL_ECAT_REVISION_NUMBER” is reflected in the value of "uint32ValueDef" below.

```
/***********************************************/
/* 1018 - Identity Object                      */
/***********************************************/
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1018, GOAL_ECAT_OBJCODE_RECORD, 0x0023);
}
… omit …

if (GOAL_RES_OK(res)) {

    uint32ValueMin = 0x0;
    uint32ValueDef = APPL_ECAT_REVISION_NUMBER;
    uint32ValueMax = 0xFFFFFFFF;

    res = goal_ecatdynOdSubIndexAdd(
        pHdlEcat,
        0x1018,
        0x03,
        GOAL_ECAT_DATATYPE_UNSIGNED32,
        EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC | EC_OBJATTR_NO_DFLT,
        (uint8_t *) &uint32ValueDef,
        (uint8_t *) &uint32ValueMin,
        (uint8_t *) &uint32ValueMax,
        4,
        NULL);
}
```

(3) ESI file

Change the value of "RevisionNo" and “DefaultValue” below.

```
1.  <Devices>
2.    <Device Physics="YY">
3.      <Type ProductCode="#x00000800" RevisionNo="#x000000002">Renesas Module</Type>
4.      <Name>RIN32M3 Module[03_ecat]</Name>
5.      <info>
```

```
1.  <Object>
2.    <Index>#x1018</Index>
3.    <Name>Identity Object</Name>
    … omit …

4.      <SubItem>
5.        <Name>Revision number</Name>
6.        <Info>
7.          <DefaultValue>#x00000002</DefaultValue>
8.        </Info>
9.      </SubItem>
```

< XML tree view>

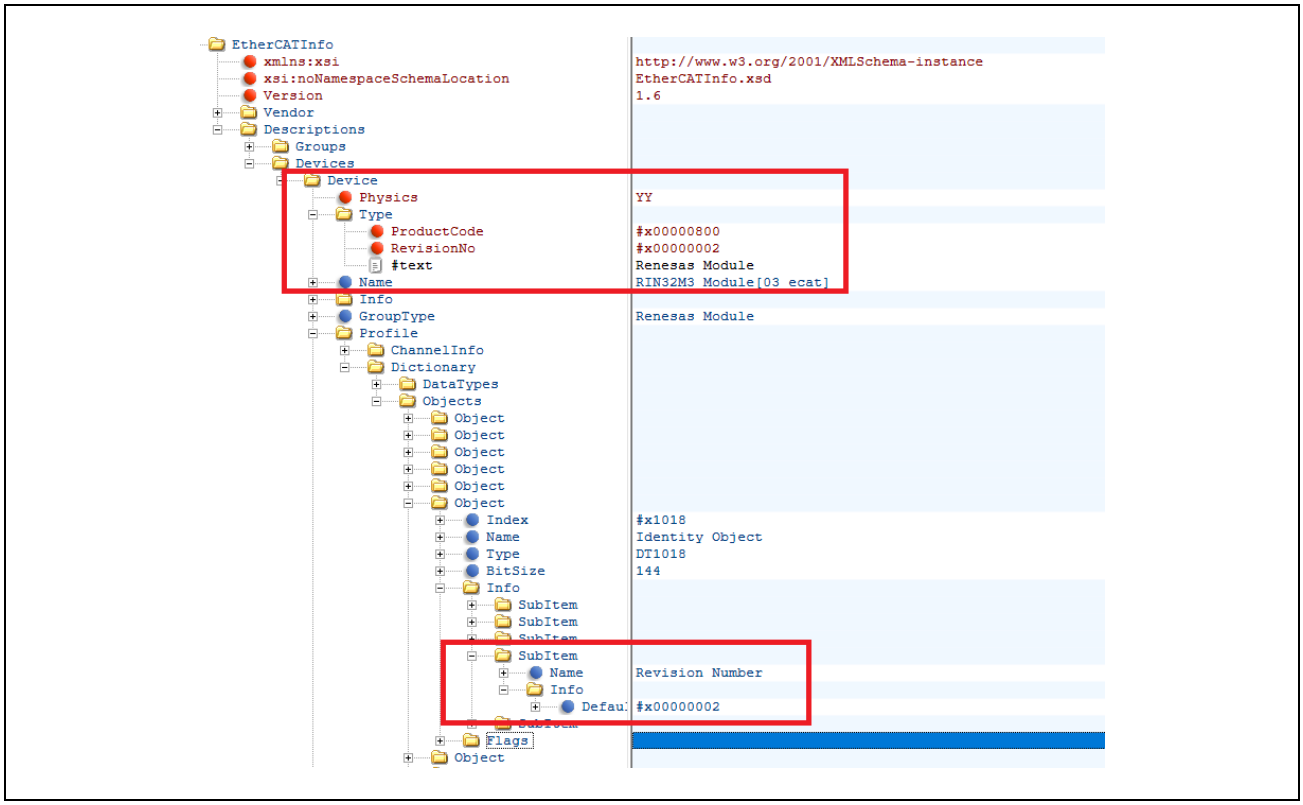

Figure 6-4 Revision Number

6.1.5 Serial Number

The default value is 0x00001234.

(1) Slave program parameter

```
#ifndef APPL_ECAT_SERIAL_NUMBER
# define  APPL_ECAT_SERIAL_NUMBER    (0x00001234)     /**< Serial Number */
#endif
```

(2) Object Dictionary

For object dictionary file, “Index = 0x1018 (Identity Object) and Subindex = 0x04” is the information about Serial Number. No setting is required in the object dictionary file because the value of “APPL_ECAT_SERIAL_NUMBER” is reflected in the value of "uint32ValueDef" below.

```
    /***********************************************/
    /* 1018 - Identity Object                      */
    /***********************************************/
    if (GOAL_RES_OK(res)) {
        res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1018, GOAL_ECAT_OBJCODE_RECORD, 0x0023);
    }
  … omit …

    if (GOAL_RES_OK(res)) {

        uint32ValueMin = 0x0;
        uint32ValueDef = APPL_ECAT_SERIAL_NUMBER;
        uint32ValueMax = 0xFFFFFFFF;

        res = goal_ecatdynOdSubIndexAdd(
            pHdlEcat,
            0x1018,
            0x04,
            GOAL_ECAT_DATATYPE_UNSIGNED32,
            EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC | EC_OBJATTR_NO_DFLT,
            (uint8_t *) &uint32ValueDef,
            (uint8_t *) &uint32ValueMin,
            (uint8_t *) &uint32ValueMax,
            4,
            NULL);
    }
```

(3) ESI file

Change the value of “DefaultValue” below.

```
<Object>
  <Index>#x1018</Index>
  <Name>Identity Object</Name>
… omit …

    <SubItem>
      <Name>Serial number</Name>
      <Info>
        <DefaultValue>#x00001234</DefaultValue>
      </Info>
    </SubItem>
```

< XML tree view >

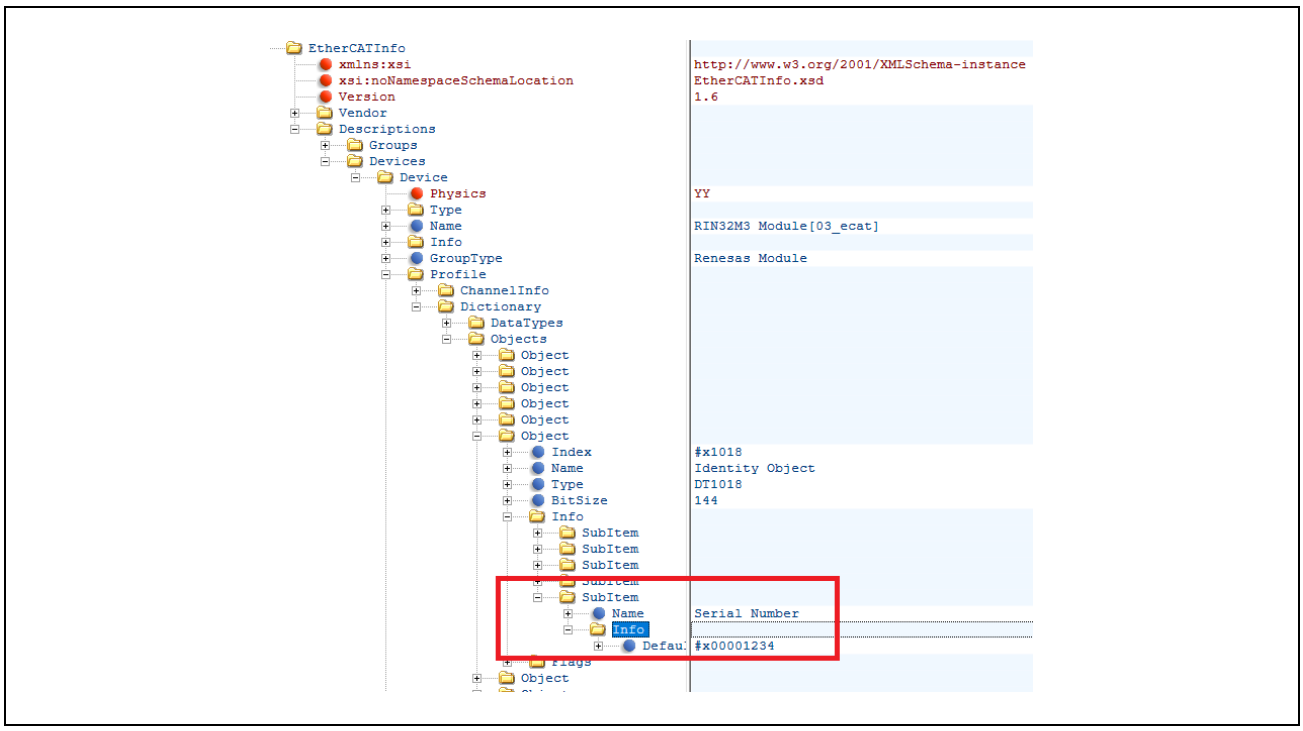

Figure 6-5 Serial Number

6.2 Data Model

The Input/Output data and size to be set according to the application are described.

6.2.1 Output Data

The following is a sample software configuration for data received from the master device.

Table 6-5 RxPDO contents

Direction	Index	subindex	size	Name	Sample	
					03	06
RX	0x6200	0x00	1	Number of Entries	✓	✓
		0x01	1	LED Output 1-8	✓	✓
	0x6201	0x00	OD_RPC_SUBIDX_DEF (3)	Number of Entries	✓	✓
		0x01	OD_DM_SUBIDX_SIZE (16)	data out (dm) 1-16	✓	✓
	0x6210	0x00	OD_RPC_SUBIDX_DEF (3)	Number of Entries	-	✓
		0x01	OD_RPC_SUBIDX_SIZE (31)	data out (rpc) 1-31	-	✓
		0x02	OD_RPC_SUBIDX_SIZE (31)	data out (rpc) 32-62	-	✓
		0x03	OD_RPC_SUBIDX_SIZE (31)	data out (rpc) 63-93	-	✓

Table 6-6 RxPDO Assignment

Direction	SyncManager	Index	Name	Size	Mapping	Sample	
						03	06
RX	SyncManager2 (0x1C12)	0x1600	RxPDO1	1	0x6200 [1]	✓	✓
		0x1601	RxPDO2	16	0x6201 [1]	✓	✓
		0x1700	RxPDO3	93	0x6210 [1], 0x6210 [2], 0x6210 [3]	-	✓

To change Output data, it is necessary to change the program file the object dictionary file and ESI file.

(1) Slave program parameter

```
1.  extern uint8_t  g_dmobj_led;
```

(2) Object Dictionary

1. Define the RxPDO contents to be received from the main device.

1-1. Create object and define name

1-2. Define number of entries at Subindex 0

1-3. Generate data parameter at Subindex 1~ and define name

1-1. Create object and define name

```
1.  /***************************************************/
2.  /* 6200 - Write Digital Output data            */
3.  /***************************************************/
4.  if (GOAL_RES_OK(res)) {
5.      res = goal_ecatdynOdObjAdd(pHdlEcat, 0x6200, GOAL_ECAT_OBJCODE_ARRAY,
    GOAL_ECAT_DATATYPE_UNSIGNED8);
6.  }
7.
8.  /* set name */
9.  if (GOAL_RES_OK(res)) {
10.     res = goal_ecatdynOdObjNameAdd(pHdlEcat, 0x6200, "Write Output data");
11. }
12.
```

Create Object

Set Object Name

1-2. Define number of entries at Subindex 0

```
1.  /* 0x6200[0] Number of Entries */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint8ValueMin = 0x00;
5.      uint8ValueMax = 0x01;
6.      uint8ValueDef = 0x01;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x6200,
11.         0x00,
12.         GOAL_ECAT_DATATYPE_UNSIGNED8,
13.         EC_OBJATTR_RD | EC_OBJATTR_OPT | EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint8ValueDef,
15.         (uint8_t *) &uint8ValueMin,
16.         (uint8_t *) &uint8ValueMax,
17.         1,
18.         NULL);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x6200, 0x00, "Number of Entries");
23. }
24.
```

Create - Subindex 0

Set Name - Subindex 0

1-3. Generate data parameter at Subindex 1~ and define name

```
1.  /* 0x6200[1] digital output 1-8 */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint8ValueMin = 0x00;
5.      uint8ValueMax = 0xFF;
6.      uint8ValueDef = 0x00;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x6200,
11.         0x01,
12.         GOAL_ECAT_DATATYPE_UNSIGNED8,
13.         EC_OBJATTR_RD | EC_OBJATTR_WR | EC_OBJATTR_RXPDOMAPPING | EC_OBJATTR_OPT |
    EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint8ValueDef,
15.         (uint8_t *) &uint8ValueMin,
16.         (uint8_t *) &uint8ValueMax,
17.         1,
18.         &g_dmobj_led);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x6200, 0x01, "digital output 1-8");
23. }
24.
```

Create - Subindex 1

Mapp to RxPDO

Set Name - Subindex 1

2. Assign the Output data generated in 1 above to RxPDO.

2-1. Create RxPDO object and define name

2-2. Define number of entries at Subindex 0

2-3. Generate data parameter at Subindex 1~, Mapping on PDO and define name

2-1. Create RxPDO object and define name

```
/*************************************************************/
/* 0x1600 - Receive PDO Mapping Parameter 1     */
/*************************************************************/
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1600, GOAL_ECAT_OBJCODE_RECORD,
EC_DEFTYPE_PDO_MAP);
}

/* set name */
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjNameAdd(pHdlEcat, 0x1600, "Receive PDO Mapping Parameter 1");
}
```

Create Object

Set Name - Object

2-2. Define number of entries at Subindex 0

```
/* 0x1600[0] Number of Entries */
if (GOAL_RES_OK(res)) {

    uint8ValueMin = 0x00;
    uint8ValueMax = 0x01;
    uint8ValueDef = 0x01;

    res = goal_ecatdynOdSubIndexAdd(
        pHdlEcat,
        0x1600,
        0x00,
        GOAL_ECAT_DATATYPE_UNSIGNED8,
        EC_OBJATTR_RD_PREOP | EC_OBJATTR_WR_PREOP | EC_OBJATTR_RD_SAFEOP |
EC_OBJATTR_RD_OP | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC,
        (uint8_t *) &uint8ValueDef,
        (uint8_t *) &uint8ValueMin,
        (uint8_t *) &uint8ValueMax,
        1,
        NULL);
}

if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x1600, 0x00, "Number of Entries");
}
```

Create - Subindex 0

Set Name - Subindex 0

2-3. Generate data parameter at Subindex 1~, Mapping on PDO and define name

```
1.  /* 0x1600[1] Mapping Entry 1 */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint32ValueMin = 0x00000000;
5.      uint32ValueMax = 0xFFFFFFFF;
6.      uint32ValueDef = 0x62000108;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x1600,
11.         0x01,
12.         GOAL_ECAT_DATATYPE_UNSIGNED32,
13.         EC_OBJATTR_RD_PREOP | EC_OBJATTR_WR_PREOP | EC_OBJATTR_RD_SAFEOP |
    EC_OBJATTR_RD_OP | EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint32ValueDef,
15.         (uint8_t *) &uint32ValueMin,
16.         (uint8_t *) &uint32ValueMax,
17.         4,
18.         NULL);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x1600, 0x01, "Mapping Entry 1");
23. }
24.
```

MAPPING: Object 0x6200, Subindex [1], 1Byte

Create - Subindex 1

Set Name - Subindex 1

(3) ESI file

1. Defines the object of Output data to be received from the main device.

The following example shows the setting of object 0x6200.

```
1.   <Object>
2.     <Index>#x6200</Index>
3.     <Name>Write Output data</Name>
4.     <Type>DT6200</Type>
5.     <BitSize>24</BitSize>
6.     <Info>
7.       <SubItem>
8.         <Name>Number of Entries</Name>
9.         <Info>
10.          <MinValue>#x00</MinValue>
11.          <MaxValue>#x01</MaxValue>
12.          <DefaultValue>#x01</DefaultValue>
13.        </Info>
14.      </SubItem>
15.      <SubItem>
16.        <Name>LED Outputs 1-8</Name>
17.        <Info>
18.          <MinValue>#x00</MinValue>
19.          <MaxValue>#xFF</MaxValue>
20.          <DefaultValue>#x00</DefaultValue>
21.        </Info>
22.      </SubItem>
23.    </Info>
24.  </Object>
25.
```

Object / Name

Subindex 0

Subindex 1

< XML tree view >

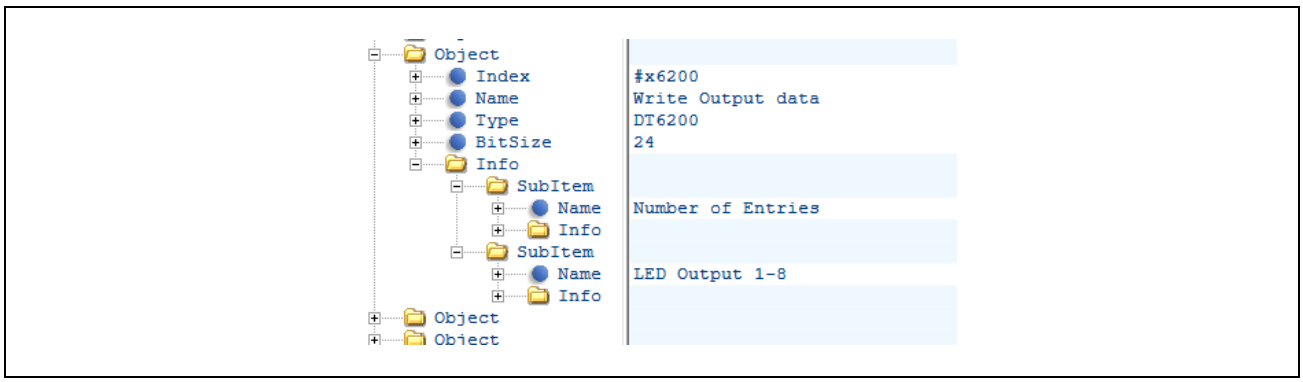

Figure 6-6 Set Object [0x6200]

2. Mapping the Output data defined in 1 above to RxPDO.

```
<Object>
  <Index>#x1600</Index>
  <Name>Receive PDO Mapping Parameter 1</Name>
  <Type>DT1600</Type>
  <BitSize>48</BitSize>
  <Info>
    <SubItem>
      <Name>Number of Entries</Name>
      <Info>
        <MinValue>#x00</MinValue>
        <MaxValue>#x01</MaxValue>
        <DefaultValue>#x01</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>Mapping Entry 1</Name>
      <Info>
        <MinValue>#x00000000</MinValue>
        <MaxValue>#xFFFFFFFF</MaxValue>
        <DefaultValue>#x62000108</DefaultValue>
      </Info>
    </SubItem>
  </Info>
  <Flags>
    <Access>ro</Access>
  </Flags>
</Object>
```

Object / Name

Subindex 0

Subindex 1

MAPPING: Object 0x6200, Subindex [1], 1Byte

< XML tree view >

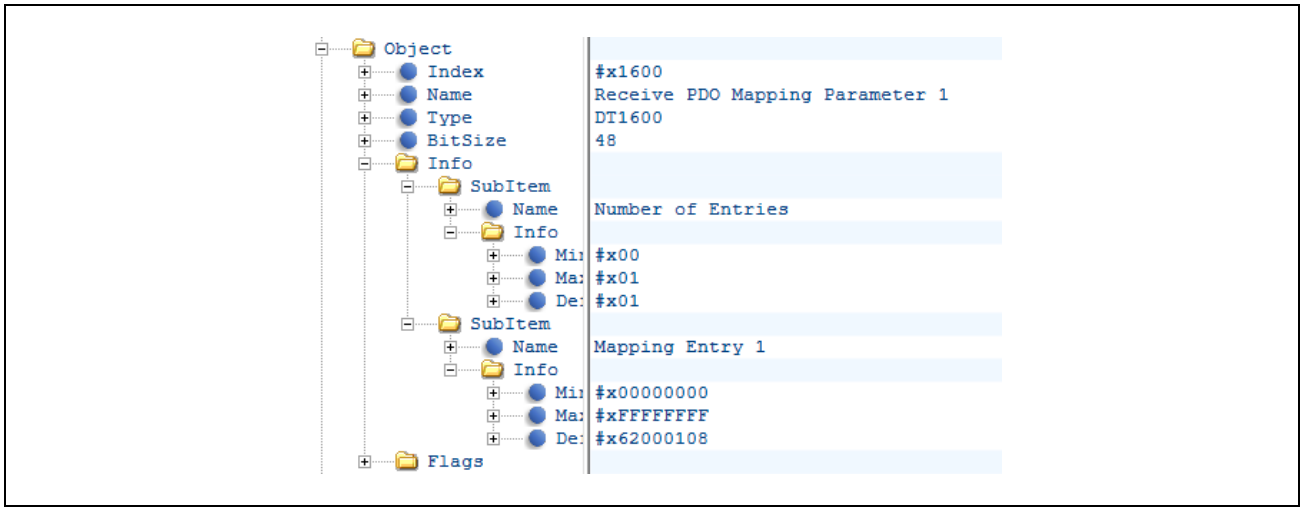

Figure 6-7 Set Object [0x1600]

3. Assigning the RxPDO defined in 1 above to Sync Manager.

```
<Object>
  <Index>#x1C12</Index>
  <Name>RxPDO assign</Name>
  <Type>DT1C12</Type>
  <BitSize>64</BitSize>
  <Info>
    <SubItem>
      <Name>SubIndex 000</Name>
      <Info>
        <MinValue>#x00</MinValue>
        <MaxValue>#x03</MaxValue>
        <DefaultValue>#x03</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 001</Name>
      <Info>
        <MinValue>#x1600</MinValue>
        <MaxValue>#x17FF</MaxValue>
        <DefaultValue>#x1600</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 002</Name>
      <Info>
        <MinValue>#x1600</MinValue>
        <MaxValue>#x17FF</MaxValue>
        <DefaultValue>#x1601</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 003</Name>
      <Info>
        <MinValue>#x1600</MinValue>
        <MaxValue>#x17FF</MaxValue>
        <DefaultValue>#x1700</DefaultValue>
      </Info>
    </SubItem>
  </Info>
  <Flags>
    <Access WriteRestrictions="PreOP">rw</Access>
  </Flags>
</Object>
```

Object / Name - RxPDO

Subindex 0

Subindex 1

ASSIGNMENT: Object 0x1600

Subindex 2

ASSIGNMENT: Object 0x1601

Subindex 3 *

ASSIGNMENT: Object 0x1700

* subindex 3 is available 06_ecat_largesize sample only

< XML tree view >

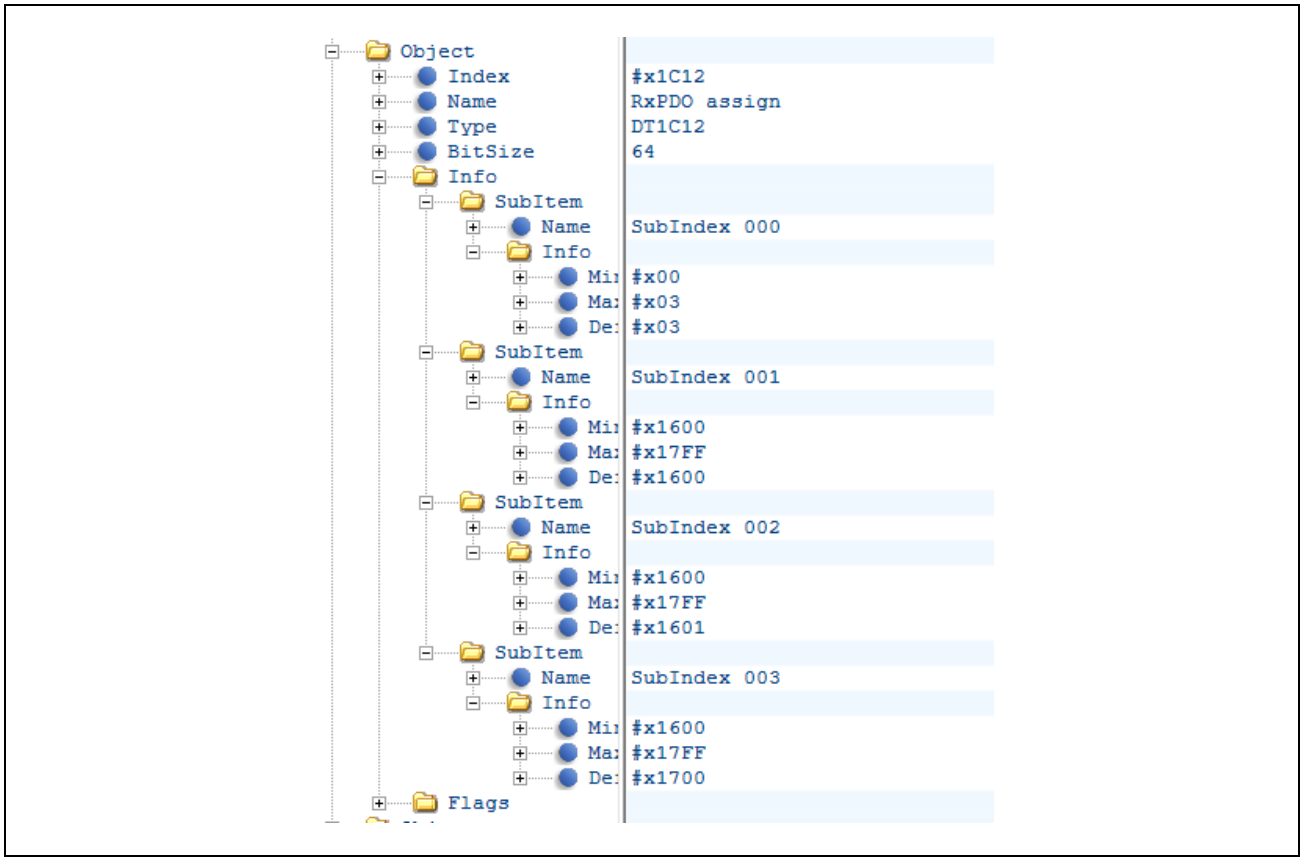

Figure 6-8 Set Object [0x1C12]

4. Define the RxPDO entry defined above. The RxPDO is entered into the Sync Manager 2 PDO Assignment.

```
<RxPdo Fixed="0" Sm="2">
  <Index>#x1600</Index>
  <Name>RxPDO 1</Name>
  <Entry>
    <Index>#x6200</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>8</BitLen>
    <Name>LED</Name>
    <DataType>USINT</DataType>
  </Entry>
</RxPdo>
<RxPdo Fixed="0" Sm="2">
  <Index>#x1601</Index>
  <Name>RxPDO 2</Name>
  <Entry>
    <Index>#x6201</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>128</BitLen>
    <Name>dout_dm_1</Name>
    <DataType>ARRAY [0..15] OF BYTE</DataType>
  </Entry>
</RxPdo>
<RxPdo Fixed="0" Sm="2">
  <Index>#x1700</Index>
  <Name>RxPDO 3</Name>
  <Entry>
    <Index>#x6210</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>248</BitLen>
    <Name>dout_rpc_1</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
  <Entry>
    <Index>#x6210</Index>
    <SubIndex>#x02</SubIndex>
    <BitLen>248</BitLen>
    <Name>dout_rpc_2</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
  <Entry>
    <Index>#x6210</Index>
    <SubIndex>#x03</SubIndex>
    <BitLen>248</BitLen>
    <Name>dout_rpc_3</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
</RxPdo>
```

RxPDO1:
ENTRY: Sync Manager 2 Assignment
Index 0x1600,
Object 0x6200 [1]

RxPDO2:
ENTRY: Sync Manager 2 Assignment
Index 0x1601,
Object 0x6201 [1]

RxPDO3: *
ENTRY: Sync Manager 2 Assignment
Index 0x1700,
Object 0x6210 [1]
Object 0x6210 [2]
Object 0x6210 [3]

* RxPDO3 is available 06_ecat_largesize sample only

< XML tree view >

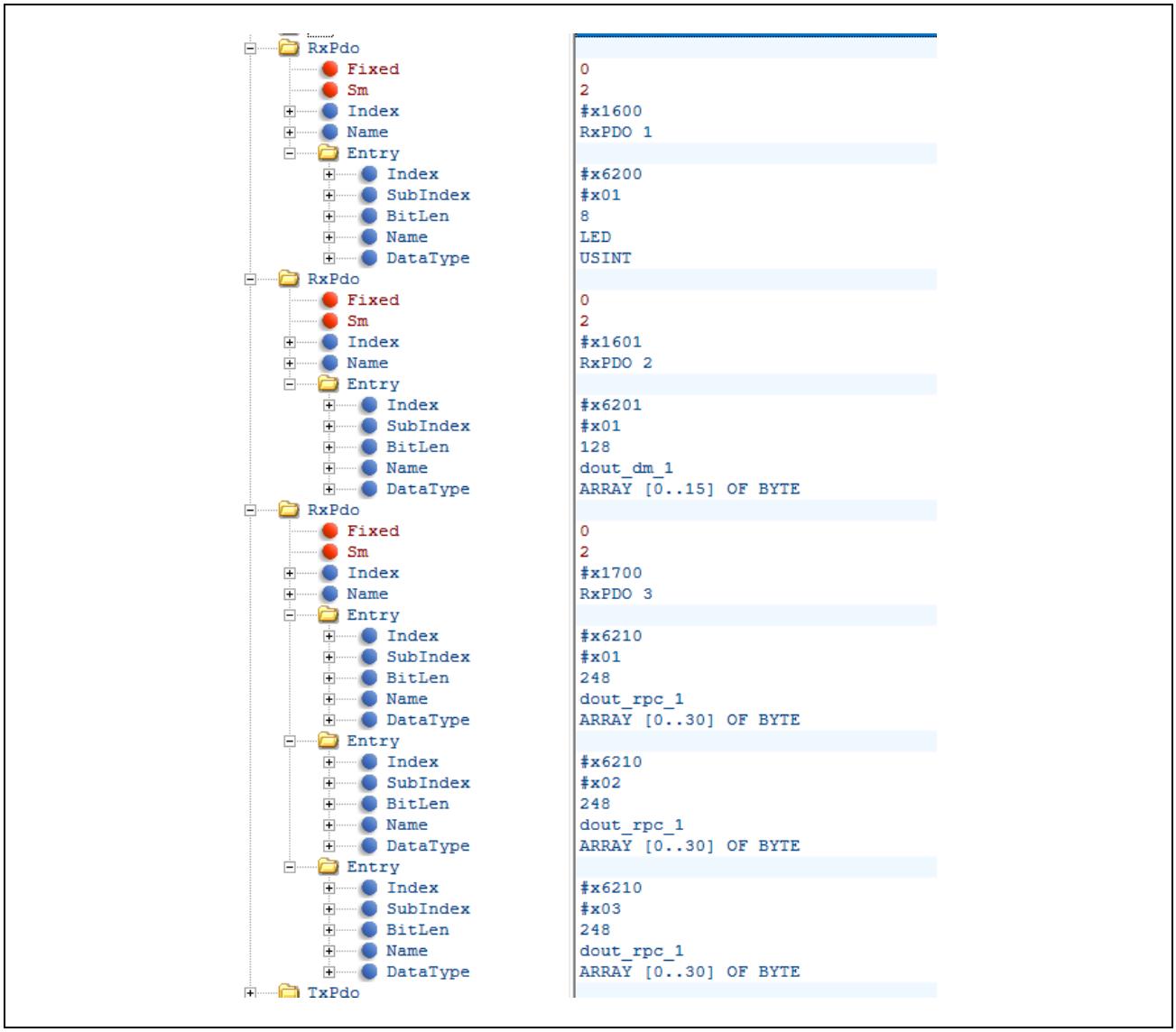

Figure 6-9 RxPDO Entry

6.2.2 Input Data

Defines the data to be sent by the sub-device (this module) to the main device.

Table 6-7 TxPDO content

Direction	Index	subindex	size	Name	Sample	
					03	06
TX	0x6000	0x00	1	Number of Entries	✓	✓
		0x01	1	Switch Input 1-8	✓	✓
	0x6001	0x00	OD_RPC_SUBIDX_DEF (3)	Number of Entries	✓	✓
		0x01	OD_DM_SUBIDX_SIZE (16)	data in (dm) 1-16	✓	✓
	0x6200	0x00	OD_RPC_SUBIDX_DEF (3)	Number of Entries	-	✓
		0x01	OD_RPC_SUBIDX_SIZE (31)	data in (rpc) 1-31	-	✓
		0x02	OD_RPC_SUBIDX_SIZE (31)	data in (rpc) 32-62	-	✓
		0x03	OD_RPC_SUBIDX_SIZE (31)	data in (rpc) 63-93	-	✓

Table 6-8 TxPDO Assignment

Direction	SyncManager	Index	Name	Size	Mapping	Sample	
						03	06
TX	SyncManager3 (0x1C13)	0x1A00	TxPDO1	1	0x6000 [1]	✓	✓
		0x1A01	TxPDO2	16	0x6001 [1]	✓	✓
		0x1B00	TxPDO3	93	0x6010 [1], 0x6010 [2], 0x6010 [3]	-	✓

To change Input data, it is necessary to change the program file the object dictionary file and ESI file.

(1) Slave program parameter

```
1.  uint8_t  g_dmobj_sw;
```

(2) Object Dictionary

1. Define the TxPDO contents to be received from the main device.

1-1. Create object and define name

1-2. Define number of entries at Subindex 0

1-3. Generate data parameter at Subindex 1~ and define name

1-1. Create object and define name

```
1.  /*********************************************/
2.  /* 6000 - Read Digital Input data       */
3.  /*********************************************/
4.  if (GOAL_RES_OK(res)) {
5.      res = goal_ecatdynOdObjAdd(pHdlEcat, 0x6000, GOAL_ECAT_OBJCODE_ARRAY,
    GOAL_ECAT_DATATYPE_UNSIGNED8);
6.  }
7.
8.  /* set name */
9.  if (GOAL_RES_OK(res)) {
10.     res = goal_ecatdynOdObjNameAdd(pHdlEcat, 0x6000, "Read Input data");
11. }
12.
```

Create Object

Set Name - Object

1-2. Define number of entries at Subindex 0

```
1.  /* 0x6000[0] Number of Entries */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint8ValueMin = 0x00;
5.      uint8ValueMax = 0x01;
6.      uint8ValueDef = 0x01;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x6000,
11.         0x00,
12.         GOAL_ECAT_DATATYPE_UNSIGNED8,
13.         EC_OBJATTR_RD | EC_OBJATTR_OPT | EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint8ValueDef,
15.         (uint8_t *) &uint8ValueMin,
16.         (uint8_t *) &uint8ValueMax,
17.         1,
18.         NULL);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x6000, 0x00, "Number of Entries");
23. }
24.
```

Create - Subindex 0

Set Name - Subindex 0

1-3. Generate data parameter at Subindex 1~ and define name

```
1.  /* 0x6000[1] digital input 1-8 */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint8ValueMin = 0x00;
5.      uint8ValueMax = 0x02;
6.      uint8ValueDef = 0x00;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x6000,
11.         0x01,
12.         GOAL_ECAT_DATATYPE_UNSIGNED8,
13.         EC_OBJATTR_RD | EC_OBJATTR_TXPDOMAPPING | EC_OBJATTR_OPT |
    EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint8ValueDef,
15.         (uint8_t *) &uint8ValueMin,
16.         (uint8_t *) &uint8ValueMax,
17.         1,
18.         &g_dmobj_sw);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x6000, 0x01, "Switch Input 1-8");
23. }
24.
```

Create - Subindex 1

Mapp to TxPDO

Set Name - Subindex 1

2. Assign the Output data generated in 1 above to TxPDO.

2-1. Create TxPDO object and define name

2-2. Define number of entries at Subindex 0

2-3. Generate data parameter at Subindex 1~, Mapping on PDO and define name

2-1. Create RxPDO object and define name

```
/*************************************************************/
/* 1A00 - Transmit PDO Mapping Parameter 1      */
/*************************************************************/
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjAdd(pHdlEcat, 0x1A00, GOAL_ECAT_OBJCODE_RECORD,
EC_DEFTYPE_PDO_MAP);
}

/* set name */
if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdObjNameAdd(pHdlEcat, 0x1A00, "Transmit PDO Mapping Parameter 1");
}
```

Create Object

Set Name - Object

2-2. Define number of entries at Subindex 0

```
/* 0x1A00[0] Number of Entries */
if (GOAL_RES_OK(res)) {

    uint8ValueMin = 0x00;
    uint8ValueMax = 0x01;
    uint8ValueDef = 0x01;

    res = goal_ecatdynOdSubIndexAdd(
        pHdlEcat,
        0x1A00,
        0x00,
        GOAL_ECAT_DATATYPE_UNSIGNED8,
        EC_OBJATTR_RD_PREOP | EC_OBJATTR_WR_PREOP | EC_OBJATTR_RD_SAFEOP |
EC_OBJATTR_RD_OP | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC,
        (uint8_t *) &uint8ValueDef,
        (uint8_t *) &uint8ValueMin,
        (uint8_t *) &uint8ValueMax,
        1,
        NULL);
}

if (GOAL_RES_OK(res)) {
    res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x1A00, 0x00, "Number of Entries");
}
```

Create - Subindex 0

Set Name - Subindex 0

2-3. Generate data parameter at Subindex 1~, Mapping on PDO and define name

```
1.  /* 0x1A00[1] Mapping Entry 1 */
2.  if (GOAL_RES_OK(res)) {
3.
4.      uint32ValueMin = 0x00000000;
5.      uint32ValueMax = 0xFFFFFFFF;
6.      uint32ValueDef = 0x60000108;
7.
8.      res = goal_ecatdynOdSubIndexAdd(
9.          pHdlEcat,
10.         0x1A00,
11.         0x01,
12.         GOAL_ECAT_DATATYPE_UNSIGNED32,
13.         EC_OBJATTR_RD_PREOP | EC_OBJATTR_WR_PREOP | EC_OBJATTR_RD_SAFEOP |
    EC_OBJATTR_RD_OP | EC_OBJATTR_NUMERIC,
14.         (uint8_t *) &uint32ValueDef,
15.         (uint8_t *) &uint32ValueMin,
16.         (uint8_t *) &uint32ValueMax,
17.         4,
18.         NULL);
19. }
20.
21. if (GOAL_RES_OK(res)) {
22.     res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x1A00, 0x01, "Mapping Entry 1");
23. }
24.
```

MAPPING: Object 0x6000, Subindex [1], 1Byte

Create - Subindex 1

Set Name - Subindex 1

(3) ESI file

1. Defines the object of Input data to be received from the main device.

The following example shows the setting of object 0x6000.

```
54.  <Object>
55.    <Index>#x6000</Index>
56.    <Name>Read INtput data</Name>
57.    <Type>DT6000</Type>
58.    <BitSize>32</BitSize>
59.    <Info>
60.      <SubItem>
61.        <Name>Number of Entries</Name>
62.        <Info>
63.          <MinValue>#x00</MinValue>
64.          <MaxValue>#x01</MaxValue>
65.          <DefaultValue>#x01</DefaultValue>
66.        </Info>
67.      </SubItem>
68.      <SubItem>
69.        <Name>Switch Input 1-8</Name>
70.        <Info>
71.          <MinValue>#x00</MinValue>
72.          <MaxValue>#xFF</MaxValue>
73.          <DefaultValue>#x00</DefaultValue>
74.        </Info>
75.      </SubItem>
76.    </Info>
77.  </Object>
```

Object / Name

Subindex 0

Subindex 1

< XML tree view>

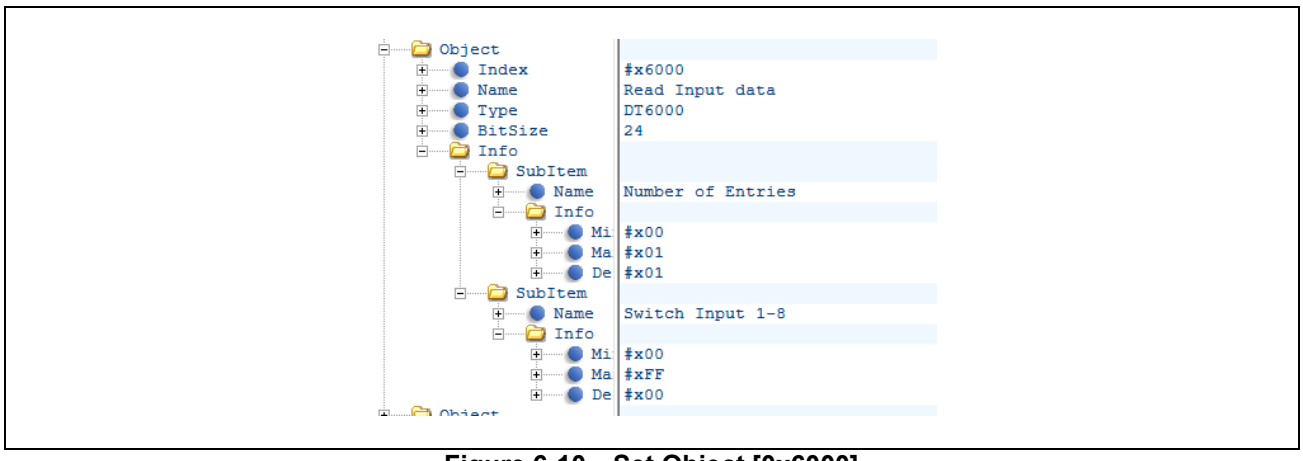

Figure 6-10 Set Object [0x6000]

2. Mapping the Output data defined in 1 above to TxPDO.

```
1.  <Object>
2.    <Index>#x1A00</Index>
3.    <Name>Receive PDO Mapping Parameter 1</Name>
4.    <Type>DT1A00</Type>
5.    <BitSize>48</BitSize>
6.    <Info>
7.      <SubItem>
8.        <Name>Number of Entries</Name>
9.        <Info>
10.         <MinValue>#x00</MinValue>
11.         <MaxValue>#x01</MaxValue>
12.         <DefaultValue>#x01</DefaultValue>
13.       </Info>
14.     </SubItem>
15.     <SubItem>
16.       <Name>Mapping Entry 1</Name>
17.       <Info>
18.         <MinValue>#x00000000</MinValue>
19.         <MaxValue>#xFFFFFFFF</MaxValue>
20.         <DefaultValue>#x60000108</DefaultValue>
21.       </Info>
22.     </SubItem>
23.   </Info>
24.   <Flags>
25.     <Access>ro</Access>
26.   </Flags>
27. </Object>
28.
```

Object / Name

Subindex 0

Subindex 1

MAPPING: Object 0x6000, Subindex [1], 1Byte

< XML ツリービュー >

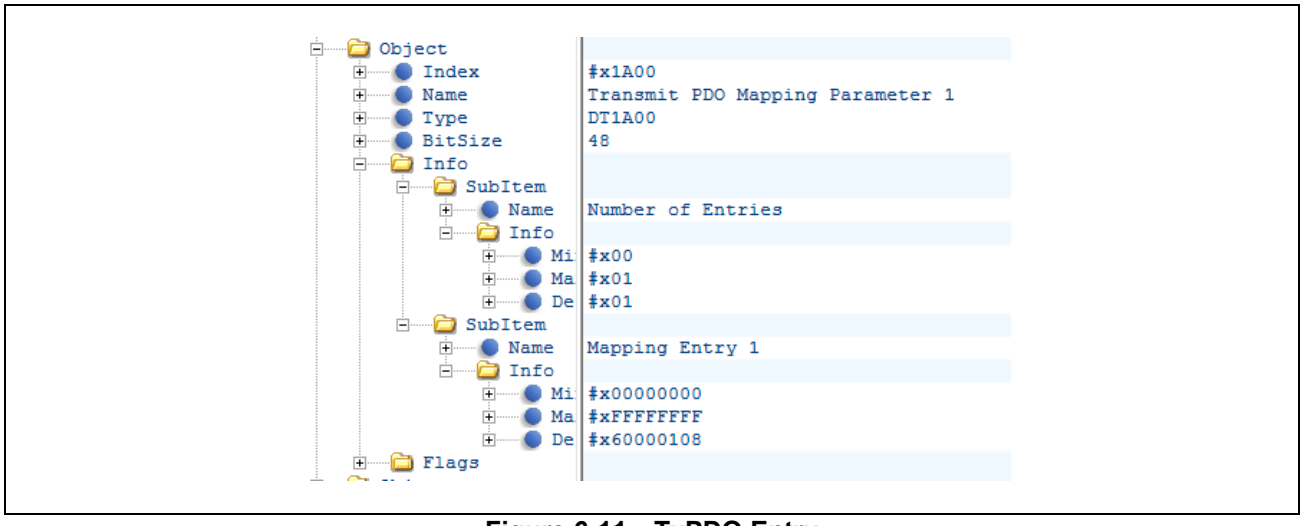

Figure 6-11 TxPDO Entry

3. Assigning the TxPDO defined in 1 above to Sync Manager.

```
<Object>
  <Index>#x1C13</Index>
  <Name>TxPDO assign</Name>
  <Type>DT1C13</Type>
  <BitSize>64</BitSize>
  <Info>
    <SubItem>
      <Name>SubIndex 000</Name>
      <Info>
        <MinValue>#x00</MinValue>
        <MaxValue>#x03</MaxValue>
        <DefaultValue>#x03</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 001</Name>
      <Info>
        <MinValue>#x1A00</MinValue>
        <MaxValue>#x1BFF</MaxValue>
        <DefaultValue>#x1A00</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 002</Name>
      <Info>
        <MinValue>#x1A00</MinValue>
        <MaxValue>#x1BFF</MaxValue>
        <DefaultValue>#x1A01</DefaultValue>
      </Info>
    </SubItem>
    <SubItem>
      <Name>SubIndex 003</Name>
      <Info>
        <MinValue>#x1A00</MinValue>
        <MaxValue>#x1BFF</MaxValue>
        <DefaultValue>#x1B00</DefaultValue>
      </Info>
    </SubItem>
  </Info>
  <Flags>
    <Access WriteRestrictions="PreOP">rw</Access>
  </Flags>
</Object>
```

Object / Name - TxPDO

Subindex 0

Subindex 1

ASSIGNMENT: Object 0x1A00

Subindex 2

ASSIGNMENT: Object 0x1A01

Subindex 3 *

ASSIGNMENT: Object 0x1B00

* subindex 3 is available 06_ecat_largesize sample only

< XML tree view >

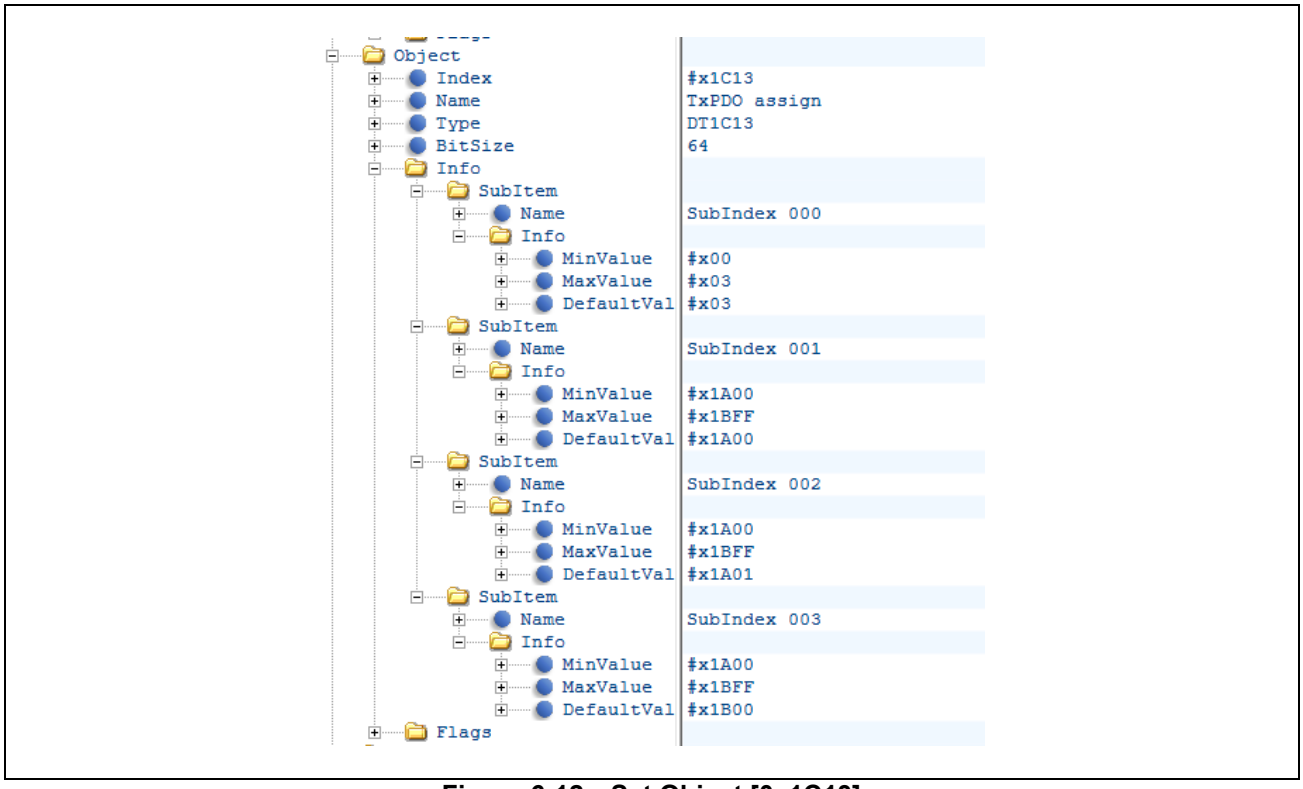

Figure 6-12 Set Object [0x1C13]

4. Define the TxPDO entry defined above. The TxPDO is entered into the Sync Manager 3 PDO Assignment.

```
<TxPdo Fixed="0" Sm="3">
  <Index>#x1A00</Index>
  <Name>TxPDO 1</Name>
  <Entry>
    <Index>#x6000</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>8</BitLen>
    <Name>SW</Name>
    <DataType>USINT</DataType>
  </Entry>
</TxPdo>
<TxPdo Fixed="0" Sm="3">
  <Index>#x1A01</Index>
  <Name>TxPDO 2</Name>
  <Entry>
    <Index>#x6001</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>128</BitLen>
    <Name>din_dm_1</Name>
    <DataType>ARRAY [0..15] OF BYTE</DataType>
  </Entry>
</TxPdo>
<TxPdo Fixed="0" Sm="3">
  <Index>#x1B00</Index>
  <Name>TxPDO 3</Name>
  <Entry>
    <Index>#x6010</Index>
    <SubIndex>#x01</SubIndex>
    <BitLen>248</BitLen>
    <Name>din_rpc_1</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
  <Entry>
    <Index>#x6010</Index>
    <SubIndex>#x02</SubIndex>
    <BitLen>248</BitLen>
    <Name>din_rpc_2</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
  <Entry>
    <Index>#x6010</Index>
    <SubIndex>#x03</SubIndex>
    <BitLen>248</BitLen>
    <Name>din_rpc_3</Name>
    <DataType>ARRAY [0..30] OF BYTE</DataType>
  </Entry>
</TxPdo>
```

TxPDO1:
ENTRY: Sync Manager 3 Assignment
Index 0x1A00,
Object 0x6000 [1]

TxPDO2:
ENTRY: Sync Manager 3 Assignment
Index 0x1A01,
Object 0x6001 [1]

TxPDO3: *
ENTRY: Sync Manager 3 Assignment *
Index 0x1B00,
Object 0x6010 [1]
Object 0x6010 [2]
Object 0x6010 [3]

* TxPDO3 is available 06_ecat_largesize sample only

< XML tree view >

Figure 6-13 TxPDO Entry

6.3 User application Output/Input Data Handling

The sample software implements two types of data transmission/reception applications as example applications.

- **Remote-IO (LED/Switch):** LED lighting control and Switch status from the evaluation board
- **Mirror:** Sends data received from the master and mirrored back

Table 6-9 API and Sample application data variable

sample		Sample app.	Data variable	Purpose	API	reference
06_ecat_large	03_ecat	get LED Data [#1]	g_dmobj_led	Reception	goal_ecatdynOdSubIndexAdd()	6.2.1. Output Data
		get Mirror Data [#3]	g_dmobj_outdata			
		set Switch Data [#2]	g_dmobj_sw	Transmission		6.2.2. Input Data
		set Mirror data [#4]	g_dmobj_indata			
	-	get Mirror Data_1	rpcData[]	Reception	goal_ecatdynOdSubIndexRpcAdd ()	6.2.1. Output Data
		get Mirror Data_2	rpcData[]			
		get Mirror Data_3	rpcData[]			
		set Mirror Data_1	rpcData[]	Transmission		6.2.2. Input Data
		set Mirror Data_2	rpcData[]			
		set Mirror Data_3	rpcData[]			

The following is a code example of the 03_ecat sample.

```
1.    void appl_loop(
2.        void
3.    )
4.    {
      …. omit ….
5.
6.        if ((GOAL_TRUE == flgAppReady) && (plat_getElapseTime(tsTout) >=
      APPL_DM_TIMEOUT_TRIGGER_VAL)) {
7.    /* output data reflected to LED */
8.      goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE1, (g_dmobj_led & 0x01) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
9.      goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE2, (g_dmobj_led & 0x02) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
10.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE3, (g_dmobj_led & 0x04) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
11.     goal_maLedSet(pMaLed, GOAL_MA_LED_REMOTE4, (g_dmobj_led & 0x08) ?
      GOAL_MA_LED_STATE_ON : GOAL_MA_LED_STATE_OFF);
12.
13.   /* switch state be set to input data */
14.   g_dmobj_sw = 0;
15.   for (uint8_t i = 0; i < 4; i++) {
16.       g_dmobj_sw |= (plat_remoteIoSwGet(i) << i);
17.   }
18.
19.   /* Received data(RxPDO) copy to Transport data(TxPDO) */
20.   GOAL_MEMCPY(g_dmobj_indata, g_dmobj_outdata, OD_DM_SUBIDX_SIZE);
21.
79.   /* update base timestamp */
80.   tsTout = goal_timerTsGet();
81.   }

      … omit …

82.   }
```

[#1] LED Data

[#2] Switch Data

[#3] [#4] Mirror Data

Application data update is controlled in the appl_loop() function, which is called in the uGOAL loop cycle. The data update cycle depends on the data transfer method.

In the sample software, the update cycle is as shown in Table 6-10, depending on the transfer method.

Table 6-10 sample soft update cycle

Data Transfer	Update cycle Define	Sample soft default [ms]	Sample	
			01	04
Cyclic	APPL_DM_TIMEOUT_TRIG_VAL	1	✓	✓
RPC	APPL_RPC_TIMEOUT_TRIG_VAL	310	-	✓

The data transfer using RPC process data to be sent and received at or above the EtherCAT maximum send/receive data size of 64 bytes, but the update cycle of the application will increase. Refer to Table 6-11 to specify the update cycle according to the data size to be handled.

Table 6-11 Data size and update cycle comparison

Data Transfer	Data size [byte]	Average cycle [ms]
Cyclic	64	1
RPC	128	40
	256	70
	512	125
	1024	230
	1434	310

Appendix

A. IP Address Setting

This chapter describes how to set the IP address of R-IN32M3 Module.

The IP address of the R-IN32M3 Module is set according to the GOAL_ID_NET (12) configuration stored in the internal nonvolatile memory at startup. It is also possible to set the IP address from the host CPU by calling *goal_maNetIpSet()*.
In the default setting in the sample applications of "01_pnio", "02_eip", "04_pnio_large" and "05_eip_large", the IP address is set by the configurations stored inside (Configured IP). Defining the macro of "GOAL_CONFIG_STATIC_IP" in the program enables to set arbitrary IP address (Static IP).

Table A-1 IP Configuration (GOAL_ID_NET)

Variable Name	Variable ID	Type	Max. Size	Description
IP	0	GOAL_CM_IPV4	4	IP address of first interface
NETMASK	1	GOAL_CM_IPV4	4	NETMASK of first interface
GW	2	GOAL_CM_IPV4	4	GATEWAY of first interface
VALID	3	GOAL_CM_UINT8	1	Validity of IP address: 0, Stored IP address is not valid, interface settings originate from network stack of system 1, Stored IP address is valid, will be applied to interface at start of device
DHCP_ENABLED	4	GOAL_CM_UINT8	1	DHCP enable: 0, DHCP disabled 1, DHCP enabled

Please note that VALID needs to be set "1" to activate IP address configurations stored in nonvolatile memory. By executing the "*goal_maNetIpSet ()"* API, configurations of IP, NETMASK, and GW are stored in the nonvolatile memory, and whether to save the VALID setting can be specified by the last argument, flgTemp. (GOAL_FALSE: Update VALID settings, GOAL_TRUE: not updated)

```
1.  GOAL_STATUS_T goal_maNetIpSet(
2.      GOAL_MA_NET_T *pNetHdl,          /**< pointer to store NET handler */
3.      uint32_t addrIp,                 /**< IP address */
4.      uint32_t addrMask,               /**< subnet mask */
5.      uint32_t addrGw,                  /**< gateway */
6.      GOAL_BOOL_T flgTemp              /**< temporary IP config flag */
7.  );
```

Also, DHCP mode is enabled by setting the "DHCP_ENABLED" in GOAL_ID_NET (12) to 1 or call the API of *goal_eipCfgDhcpOn()* for EtherNet/IP. In the sample software of 02_eip, DHCP is enabled by defining a "GOAL_CONFIG_ENABLE_DHCP" macro as "1" in the program.

Table A-2 provides a list of how to set up an IP address.

Table A-2 IP address setting list

Methods	Descriptions
Configured IP	- Use the value held in the non-volatile memory of R-IN32M3 module - The value can be changed using the Management Tool. For more information, see "R-IN32M3 Module (RY9012A0) Management Tool Instruction Guide (R30AN0390EJ****)". - This method is used as the default setting for “01_pnio”, “02_eip”, “04_pnio_large” and “05_eip_large” sample application of this sample.
Static IP	- Mainly used for evaluation. - The changed value is hold in the non-volatile memory of R-IN32M3 Module. - The value can be changed with “01_pnio”, “02_eip”, “04_pnio_large” and “05_eip_large” sample application of this sample. By defining “GOAL_CONFIG_STATIC_IP” macro in the program with 1, any IP address can be set.
DHCP	- It is possible to change enable / disable by using Management Tool. - It is also possible to change using "02_eip" and “05_eip_large” sample application of this sample software, the default value is disable. By defining “GOAL_CONFIG_ENABLE_DHCP” macro in the program with 1, DHCP become enable. - If DHCP is enabled and there is no DHCP server on the network, the value held in the non-volatile memory of R-IN32M3 Module will be used.

B. Module Name and Customer ID Setting

This chapter describes how to set Module Name and Customer ID, which is one of the internal information of R-IN32M3 Module.

The Module Name and Customer ID of the R-IN32M3 Module is set according to the GOAL_ID_DD (34) configuration stored in the internal nonvolatile memory at startup. If there is a difference from these values defined in the host CPU, they are set by *appl_ccmCfgVarSet()* and stored in the nonvolatile memory by *appl_ccmCfgSave()*.

Table B-3 IP Configuration (GOAL_ID_DD)

Variable Name	Variable ID	Type	Max. Size	Description
MODULENAME	0	GOAL_CM_STRING	20	Customer specific name of the module
CUSTOMERID	1	GOAL_CM_UNIT32	4	Customer Id
RESERVED	2	GOAL_CM_UINT8	1	-
FEATURE_DISABLE	3	GOAL_CM_UINT32	4	Each bit disables a function: bit 0, disable “HELLO DETECTION” bit 1, disable WINK bit 2, disable GETLIST bit 3, disable GET VALUE bit 4, disable SET VALUE

The Module Name and Customer ID are defined in goal_appl.h file in each sample.

Regarding the Module Name, change the value of "APPL_DD_MODULE_NAME" defined in the macro. The default value is "R-IN32M3_Module".

Ex)

appl\ugoal\01_pnio\goal_appl.h

```
1.  #ifndef APPL_DD_MODULE
25. # define APPL_DD_MODULE        R-IN32M3_Module     /**< module name */
26. #endif
27.
28. …. omit ….
29.
30. #define APPL_DD_MODULE_NAME     STRING(APPL_DD_MODULE)
```

Regarding the Customer ID, change the value of "APPL_DD_CUSTOMER_ID" defined in the macro. The default value is 0x00000001.

Ex)

appl\ugoal\01_pnio\goal_appl.h

```
1.  #ifndef APPL_DD_CUSTOMER_ID
31. # define APPL_DD_CUSTOMER_ID    (0x00000001)       /**< customer id */
32. #endif
```

C. Data Size Limitation

According to the specifications of the uGOAL middleware used in this sample, there is an upper limit to the size of data that can be handled, which varies depending on the protocol and whether it is "long packet" data or not.

RPC Transfer is a method of periodic communication using RPC data frames in SPI frames (128 bytes) between R-IN32M3 Module and the host microcontroller (see Table 3-4) Since RPC frames, which are originally intended for asynchronous data communication, are used, it is possible to send larger data than the method using ordinary Cyclic data frames, but the update cycle of the application is restricted.

Table C-1 Maximum data size for each protocol

Protocol	RPC transfer	Sample application	Maximum data size [byte]	Refer
PROFINET	-	01_pnio	69	Table 4-6
	✔	04_pnio_largesize	1434	
EtherNet/IP	-	02_eip	69	Table 5-6
	✔	05_eip_largesize	495	
EtherCAT	-	03_ecat	64	Table 6-11
	✔	06_ecat_largesize	1408	

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct/15/2021	—	First Edition
1.01	Aug/5/2022	8	Update the folder structure
		23	Update the memory capacity
1.02	May/31/2023	-	Minor Correction
		8	2.1.1: Changed to the latest sample software configuration
		44	4: Changes due to sample software updates
		66	4.3: Added update cycle and data size for Cyclic and RPC transfer
		66	5: Changes due to sample software updates
		77	5.3: Added update cycle and data size for Cyclic and RPC transfer
		80	6: Changes due to sample software updates
		116	6.3: Added update cycle and data size for Cyclic and RPC transfer
1.03	Dec/15/2023	75	Modified Table 5-3
		28, 122	Added explanation about RPC transfer

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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.