

## RL78/G14

### Analog Input/Output (Arduino API)

---

#### Introduction

This application note describes how to control LED light brightness by PWM output according to the analog input from an ambient light sensor in the program description (such as Arduino language) using the RL78/G14 Fast Prototyping Board (FPB).

#### Target Device

RL78/G14

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

**Contents**

1. Specifications .....	3
1.1 Program Execution Environment.....	3
1.2 Program (Sketch) Configuration.....	6
1.3 Preparations for Project Startup.....	6
1.4 Definitions in the Program (sketch) .....	7
1.5 Initial Setting Processing .....	8
1.6 Main Processing Part .....	8
1.7 Ambient Light Sensor Control .....	9
2. Operating Conditions .....	9
3. Related Application Notes.....	9
4. Hardware .....	10
4.1 Example of Hardware Configuration .....	10
4.2 List of Pins Used.....	10
5. Software .....	11
5.1 Summary of Operation .....	11
5.2 List of Constants.....	12
5.3 List of Variables .....	12
5.4 List of Functions .....	13
5.5 Specification of Functions .....	15
5.6 Flowcharts .....	30
5.6.1 Initial setting function.....	30
5.6.3 Processing function to obtain the PWM signal duty ratio from the current value of the ambient light sensor.....	33
6. Sample Code.....	34
7. Reference Documents .....	34
Revision History.....	35

## 1. Specifications

This application note provides the program description (such as Arduino language) using the FPB, where the ambient light sensor value is read by an analog input and the analog PWM signal that controls LED light brightness is output.

When the switch is pressed or at intervals of 10 seconds, the ambient light sensor output value converted to a voltage value by a resistor is read by the A/D converter. LED light brightness is adjusted based on control of the PWM duty ratio according to the read result.

Due to a wide dynamic range (0.2  $\mu$ A to 2 mA with illuminance 0.1 lux to 1000 lux) of the ambient light sensor (NJL7302L-F3) output current, load resistors 51 k $\Omega$  and 1 k $\Omega$  must be switched. This sample code uses a load resistor of 1 k $\Omega$  to comply with illuminance 10 lux to 1000 lux.

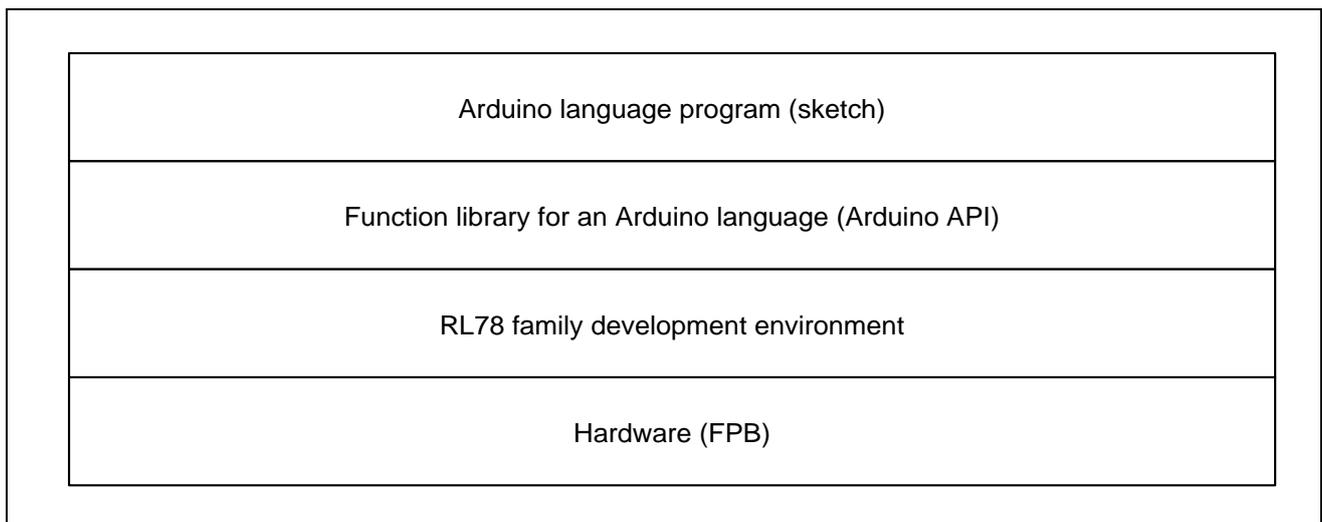
Table 1-1 lists peripheral functions to be used.

**Table 1-1 Peripheral Function and Use**

Peripheral Function	Use
Digital input	Reads switch (SW_USR) status.
Digital output	Selects ambient light sensor sensitivity.
A/D Converter	Measures ambient light sensor output.
Timer array unit	Generates a PWM signal by software. (D5 pin)
Timer RD	Generates a PWM signal by hardware. (D6 pin)
Timer array unit	Measures elapsed time.

### 1.1 Program Execution Environment

In this application note, a program in an Arduino language is executed in a development environment specific to the RL78 family. A conceptual diagram of the program execution environment is shown in Figure 1.1.



**Figure 1.1 Program execution environment**

Library functions that can be used in this application note are shown in Table1.2 to Table1.4.

**Table1.2 Library functions (1/3)**

Item	Library Function	Function
Digital I/O	pinMode(pin, mode)	Specifies the operation mode (input mode/output mode/input mode with internal pull-up resistor enabled) for the pin specified by "pin".
	digitalWrite (pin, value)	Sets the pin specified by "pin" to the state specified by "value" (high level/low level).
	digitalRead(pin)	Reads out the state of the pin specified by "pin".
Time control	millis()	Returns, in millisecond units, the time from the start of program execution to the present time.
	micros()	Returns, in microsecond units, the time from the start of program execution to the present time.
	delay (ms)	Stops the program for the specified time in millisecond units.
	delayMicroseconds (us)	Stops the program for the specified time in microsecond units.
Analog I/O	analogRead(pin)	Converts the signal from the analog input pin to a digital signal and reads the digital signal.
	analogWrite(pin, value)	Outputs the PWM signal of the specified duty ratio (0 to 255) from the analog output pin.
	analogReference(type)	Specifies the analog input reference voltage.

**Table1.3 Library functions (2/3)**

Item	Library Function	Function
I2C control	Wire.begin()	Initializes IICA0 and connects to the I <sup>2</sup> C bus as the master.
(Wire)	Wire.requestFrom (saddr7, bytes, stop)	Receives data with the size specified by "bytes" from the specified slave.
	Wire.requestFrom (saddr7, bytes)	The Wire.available() function is used to obtain the number of bytes and the Wire.read() function is used to read data.
	Wire.beginTransmission (saddr7)	Prepares for sending data to the specified slave. Then, the Wire.write() function is used to enqueue data and the Wire.endTransmission() function is used to send the data.
	Wire.endTransmission (stop)	Sends data from the queue to the slave, and then ends processing.
	Wire.write(data)	Enqueues data that is to be sent to the slave.
	Wire.available()	Uses the Wire.read() function to check the number of bytes that can be read.
	Wire.read()	Reads receive data from the slave.

Note: The slave function of the I2C bus is not supported. For some functions, a limit is placed on the arguments that can be specified or the number of arguments that can be specified.

Table1.4 Library functions (3/3)

Item	Library Function	Function
Simplified IIC control (Wire1)	Wire1.begin()	Initializes IIC00 (Wire1) connected to Pmod connector 1 and connects to the I <sup>2</sup> C bus as the master.
	Wire1.requestFrom(saddr7, bytes, stop) Wire1.requestFrom(saddr7, bytes)	Receives data with the size specified by "bytes" from the specified slave.  The Wire1.available() function is used to obtain the number of bytes and the Wire1.read() function is used to read data.
	Wire1.beginTransmission(saddr7)	Prepares for sending data to the specified slave.  Then, the Wire1.write() function is used to enqueue data and the Wire1.endTransmission() function is used to send the data.
	Wire1.endTransmission(stop)	As Wire1, sends data from the queue to the slave, and then ends processing.
	Wire1.write(data)	Enqueues data that is to be sent to the slave of Wire1.
	Wire1.available()	Uses the Wire1.read() function to check the number of bytes that can be read.
	Wire1.read()	Dequeues data that was received from the slave of Wire1.
	Simplified IIC control (Wire2)	Wire2.begin()
Wire2.requestFrom(saddr7, bytes, stop) Wire2.requestFrom(saddr7, bytes)		Receives data with the size specified by "bytes" from the specified slave.  The Wire2.available() function is used to obtain the number of bytes and the Wire2.read() function is used to read data.
Wire2.beginTransmission(saddr7)		Prepares for sending data to the specified slave.  Then, the Wire2.write() function is used to enqueue data and the Wire2.endTransmission() function is used to send the data.
Wire2.endTransmission(stop)		As Wire2, sends data from the queue to the slave, and then ends processing.
Wire2.write(data)		Enqueues data that is to be sent to the slave of Wire2.
Wire2.available()		Uses the Wire2.read() function to check the number of bytes that can be read.
Wire2.read()		Dequeues data that was received from the slave of Wire2.

Note: The slave function of the I<sup>2</sup>C bus is not supported. For some functions, a limit is placed on the arguments that can be specified or the number of arguments that can be specified.

## 1.2 Program (Sketch) Configuration

Subfolders are prepared for each integrated development environment below the folder (workspace) in which the project is stored. In the folders for each of the integrated development environments the files are stored that are used in the RL78 family development environment.

In each sketch subfolder, AR\_SKETCH.c is stored which is the Arduino language program (sketch). When viewing or modifying sketch, the "AR\_SKETCH.c" file in the sketch subfolder is used.

## 1.3 Preparations for Project Startup

A folder or zip file (in the case of e2studio) compliant with three integrated development environments can be generated by decompressing the stored archive (zipped sample code). Use the folder or zip file best suited for the integrated development environment to be used.

For details, refer to the following application note.

RL78 Family Arduino API Introduction Guide (R01AN5413)

## 1.4 Definitions in the Program (sketch)

Definitions in the program (sketch) are indicated in Figure 1.2.

```

1)
int swPin      = 18;           // assign D18 pin to swPin for SW_USER. ↓
↓
int sensorin   = ANALOGPin0;  // assign A0 pin for sensor data ↓
int sensorrefh = ANALOGPin2;  // assign A2 pin for reference high ↓
int sensorrefl = ANALOGPin3;  // assign A3 pin for reference low ↓
↓
int high_range = 1;           // assign D1 pin to select high illumination ↓
int low_range  = 0;           // assign D0 pin to select low illumination ↓
↓
int pwm_s      = SOFTPWM2;    // assign SOFTPWM (D5) pin to PWM signal ↓
int pwm_h      = HARDPWM1;    // assign HARDPWM (D6) pin to PWM signal ↓

2)
#define SECOND10 ( 10000/16 ) // 10 seconds divided by 16milli sec ↓
↓
unsigned int old_time = 0x0000; // previous time (milli sec.) ↓
int count16ms = 0x0000;        // for count 10 seconds ↓
char sw_work = 0xFF;           // work for switch check ↓
↓
unsigned int ad_value[2][4];    // 4 times A/D conversion value ↓
char ad_time;                  // A/D conversion time count ↓

```

Figure 1.2 Program definition details

- 1) The “swPin” pin to control the onboard switch (SW\_USR) is set to 18 and assigned to D18. The A0 pin (ANALOGPin0) is specified for measuring the analog output of the ambient light sensor. The A2 pin (ANALOGPin2) and A3 pin (ANALOGPin3) are specified for measuring the selection circuit reference voltage. Furthermore, the control pins to specify the measurement range of the ambient light sensor are set to 0 and 1 and assigned to D0 and D1. In addition, D5 (software PWM) and D6 (hardware PWM) are assigned as analog PWM signal output pins that control LED brightness.
- 2) Next, 16-bit variable “old\_time” is defined to check elapsed time (milliseconds), 16-millisecond counter “count16ms” is defined to count 10 seconds, and checking variable “sw\_work” is defined to check switch status. Furthermore, two-dimensional array “ad\_value” that stores measured values sent from the sensor is defined. The number of measurements is counted by variable “ad\_time”.

## 1.5 Initial Setting Processing

The initial settings section of the program (sketch) is shown in Figure 1.3.

In this figure, the switch input pin is designated for input and the ambient light sensor control pins are designated for output as a setup function.

```
void setup(void) {
  // put your setup code here, to run once:
  pinMode(swPin, INPUT); // set D18pin to input mode
  pinMode(high_range, OUTPUT); // set D1 pin to output mode
  pinMode(low_range, OUTPUT); // set D0 pin to output mode
  analogReference(DEFAULT); // use VDD as reference for analog in
}
```

Figure 1.3 Initial setting processing section

## 1.6 Main Processing Part

The leading section of the main processing, which is executed repeatedly, is shown in Figure 1.4. When preparations for project startup have been set correctly, the sketch will be downloaded, then executed until the leading section of the main processing.

```
void loop(void){
  // put your main code here, to run repeatedly:
  static char m_time = 1;
  char work;
  char sw_data;
  unsigned int time_work;
  unsigned long long_work;
  /*-----
   * wait for 16milli seconds interval.
   *-----*/
  time_work = ( int )( millis() & 0xFFFF0 ); // read milli sec data
  if ( old_time != time_work ) // check 16 milli seconds passed
  {
```

Figure 1.4 The leading section of the main process

## 1.7 Ambient Light Sensor Control

The 10-bit A/D converter does not have sufficient accuracy due to the wide dynamic range (0.2  $\mu$ A to 2 mA with illuminance 0.1 lux to 1000 lux) of the ambient light sensor (NJL7302L-F3) output current. For this reason, it is necessary to use a 1-k $\Omega$  load resistor within an illuminance range of 10 lux to 1000 lux and a 51-k $\Omega$  load resistor within an illuminance range of 0.1 lux to 20 lux.

In this sample code, LED light brightness is controlled by PWM output according to the analog input from the ambient light sensor. The PWM duty ratio is linearly controlled (256 divisions) from 100% (50 lux) to 0% (500 lux).

## 2. Operating Conditions

The operation of the sample code provided with this application note has been tested under the following conditions.

Table 2.1 Operating conditions

Item	Description
Microcontroller used	RL78/G14 (R5F104MLAFB : RL78G14_FPFB)
Operating frequency	<ul style="list-style-type: none"> <li>● High-speed on-chip oscillator clock (<math>f_{IH}</math>): 32 MHz</li> <li>● CPU/peripheral hardware clock: 32 MHz</li> </ul>
Operating voltage	3.3 V (can be operated at 2.75 V to 5.5 V) LVD operation: Reset mode LVD detection voltage ( $V_{LVD}$ ) At rising edge: 2.81 V typ. (2.76 V to 2.87 V) At falling edge: 2.75 V typ. (2.70 V to 2.81 V)
Integrated development environment	Renesas Electronics CS+ for CC V8.05.00 Renesas Electronics e <sup>2</sup> studio V7.7.0 IAR Systems IAR Embedded Workbench for RL78
C compiler	Renesas Electronics CC-RL V1.10.00 IAR Systems IAR C/C++ Compiler v4.20.1 for RL78

## 3. Related Application Notes

The application notes related to this application note are shown below.

Refer to these together with this application note.

RL78 Family Arduino API Introduction Guide (R01AN5413)

RL78/G14 Onboard LED Flashing Control (Arduino API) (R01AN5384)

## 4. Hardware

### 4.1 Example of Hardware Configuration

Figure 4.1 shows the hardware that is used in this application note.

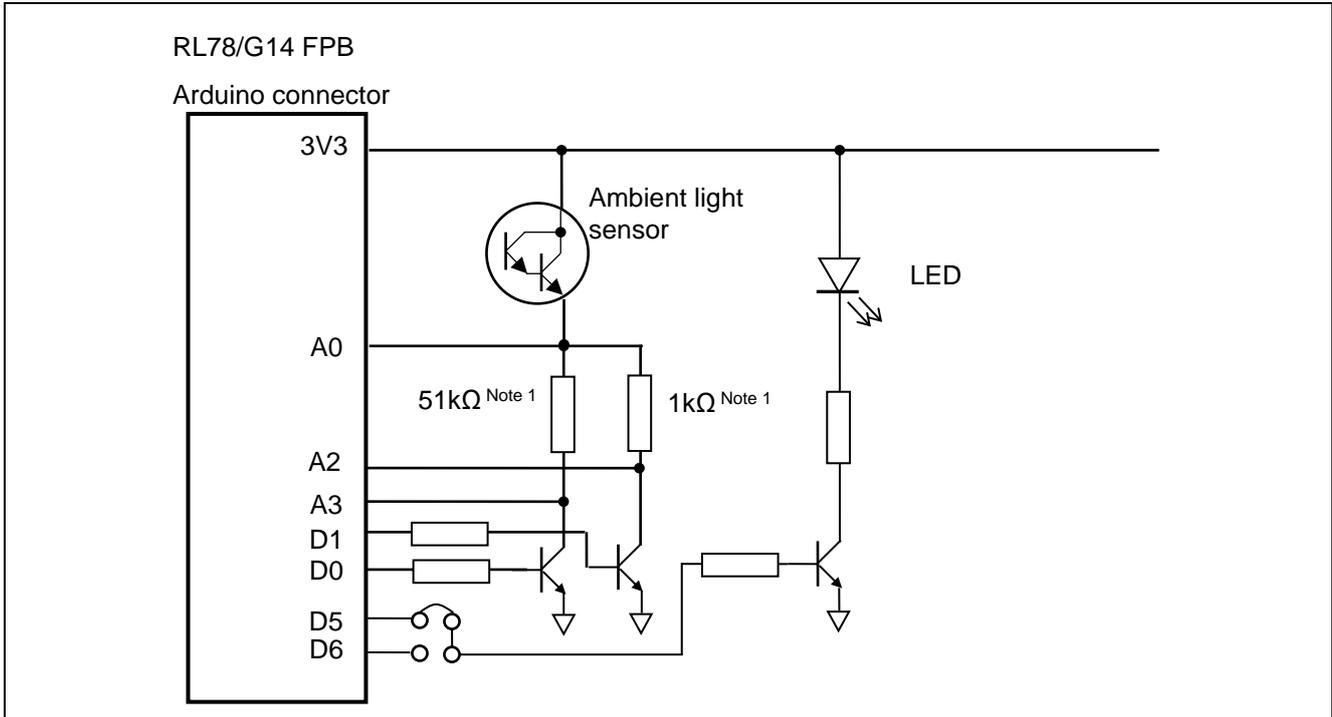


Figure 4.1 Hardware configuration example

Note 1: Metal oxide film resistor

Note 2: This conceptual diagram is simplified in order to summarize the connections.

As the power supply voltage, 3.3 V is supplied via USB.

### 4.2 List of Pins Used

Table 4.1 shows the pins used and their functions.

Table 4.1 Pins used and their functions

Pin	Port Name	I/O	Function
D0	P143	Output	Low-illuminance selection signal
D1	P144	Output	High-illuminance selection signal
D5	P77	Output	PWM (software) signal
D6	P10	Output	PWM signal
A0	P26	Analog Input	Illuminance signal input
A2	P24	Analog Input	Comparison voltage input for high illuminance
A3	P23	Analog Input	Comparison voltage input for low illuminance
D18	P137	Input	Switch (SW_USR) input

## 5. Software

### 5.1 Summary of Operation

In this application note, when the main processing (loop) starts upon completion of the initial pin settings, ambient light sensor measurement starts. Measurement is performed four times. The duty ratio of the LED drive PWM signal is calculated from the average value of four measured values, and then the duty ratio is output from the analog output pin as a PWM signal.

Subsequently, measurement is performed at intervals of 10 seconds or at each press of the switch.

Details are explained in (1) to (2) below.

#### (1) Use the "setup" function to specify the settings of the pins to be used.

- Specify the onboard SW\_USR switch read pin (swPin) for digital input.
- Specify the pins to set the load resistor of the ambient light sensor (D1 and D0 pins) for digital output.

#### (2) Use the "loop" function to perform the main processing.

- Obtain 12 bits (bit15 to bit4) (in units of 16 milliseconds) of elapsed time (milliseconds) from startup.
- Check whether the obtained data has changed from the old data (old\_time).
- If the data remains unchanged, terminate the processing and return to the beginning of the loop function.
- If the data has changed (16 milliseconds passed), update the old data (old\_time) to the obtained data.
- Count up the 10-second (625) counter for activating the sensor.
- Check the state of the switch connected to D18.
- If the switch has not been pressed and 10 seconds have not passed, the beginning of the loop function is resumed.
- When the switch has been pressed or 10 seconds have passed, measure the illuminance.
  - ◆ Apply the load resistance of the ambient light sensor to start measurement.
  - ◆ Wait until the data becomes stable (approximately 16 milliseconds).
  - ◆ Measure the sensor output voltage using the analog input (A0 pin) and the load resistor drive voltage using the analog input (A2 pin) four times respectively, and then calculate the average values.
  - ◆ Disconnect the load resistor.
  - ◆ Calculate the current value from the obtained voltage.
  - ◆ Calculate the illuminance from the current value.
  - ◆ Calculate the duty ratio of the LED drive PWM signal from the obtained illuminance value, and then set the duty ratio in the analog output function.
- The beginning of the loop function is resumed.

## 5.2 List of Constants

Table 5.1 shows constants that are used in the sample code.

**Table 5.1 Constants used in sample code**

Constant Name	Setting Value	Description
swPin	18	SW_USR read pin number
high_range	1	High-illuminance selection pin number
low_range	0	Low-illuminance selection pin number
sensorin	0	Ambient light sensor voltage input pin number
sensorrefh	2	Ambient light sensor high-illuminance load drive voltage input pin number
sensorrefl	3	Ambient light sensor low-illuminance load drive voltage input pin number
pwm_s	5	PWM using pin 5
pwm_h	6	PWM using pin 6
DEFAULT	0	A/D reference voltage = VDD
INTERNAL	1	A/D reference voltage = internal reference voltage
EXTERNAL	2	A/D reference voltage = external power voltage
SECOND10	625	Number of 16-millisecond pulses for measuring 10 seconds

## 5.3 List of Variables

Table 5.2 lists global variables.

**Table 5.2 Global variables**

Type	Variable Name	Description	Function used
unsigned int	old_time	Elapsed time (milliseconds) from the previous startup	loop()
int	count16ms	16-millisecond counter for counting 10 seconds	loop()
char	sw_work	For checking switch status	loop()
unsigned int	ad_value[2][4]	Buffer for four analog conversion results	loop()
char	ad_time	For counting the number of analog conversions	loop()
unsigned char	g_dachannel [6]	PWM control data (for input)	analogWrite(), r_tau0_channel0_interrupt()
unsigned char	g_da_time[3]	Software PWM control data	r_tau0_channel0_interrupt()

## 5.4 List of Functions

Table 5.3 to Table 5.4 shows a list of functions.

**Table 5.3 List of functions (1/2)**

Function Name	Overview
loop	Main processing (sketch)
setup	Initialization function (sketch)
get_PWM_duty	Calculates the PWM output duty ratio from the current value of the ambient light sensor.
pinMode	Specifies the operation mode (input mode, output mode, or input mode with an internal pull-up resistor enabled) for the pin.
digitalWrite	Outputs data to the pin.
digitalRead	Reads the pin state.
micros	Returns the time in microseconds from the start of program execution to the present time.
millis	Returns the time in milliseconds from the start of program execution to the present time.
delay	Stops the program for the time specified in milliseconds.
delayMicroseconds	Stops the program for the time specified in microseconds.
analogRead	Reads the value of the specified analog pin.
analogWrite	Outputs the analog value (PWM signal) from the specified pin.
analogReference	Specifies the reference voltage for A/D conversion of analog input.
Wire.begin	Initializes the I <sup>2</sup> C library and connects it as the master.
Wire.requestFrom	Starts reading data from the specified slave. The read operation is processed as an interrupt.
Wire_requestFromS	Starts reading data from the specified slave. The read operation is processed as an interrupt. It is possible to specify that stop conditions are generated when reception is completed. This is an internal processing function for Wire.requestFrom.
Wire_requestFromsub	This is an internal processing function for Wire.requestFrom.
Wire.available	Returns the number of bytes that can be read by using Wire.read from the reception buffer.
Wire.read	Reads data from the reception buffer.
Wire.beginTransmission	Prepares for sending data to the specified slave.
Wire.write	Writes the send data to the transmission buffer.
Wire_writec	Adds one-character data to the transmission buffer. This is an internal processing function for Wire.write.
Wire_writetb	Adds a data block to the transmission buffer. This is an internal processing function for Wire.write.
Wire.endTransmission	Actually sends the send data that is set in the buffer via the I <sup>2</sup> C bus. It is possible to specify that stop conditions are generated when transmission is completed.

Table 5.4 List of functions (2/2)

Function Name	Overview
Wire1.begin	Initializes the I <sup>2</sup> C library of Pmod connector 1 and connects it as the master.
Wire1.requestFrom	Starts reading data from the specified slave. The read operation is processed as an interrupt.
Wire1_requestFromS	Starts reading data from the specified slave. The read operation is processed as an interrupt. It is possible to specify that stop conditions are generated when reception is completed. This is an internal processing function for Wire1.requestFrom.
Wire1_requestFromsub	This is an internal processing function for Wire1.requestFrom.
Wire1.available	Returns the number of bytes that can be read by using Wire1.read from the reception buffer.
Wire1.read	Reads data from the reception buffer.
Wire1.beginTransmission	Prepares for sending data to the specified slave.
Wire1.write	Writes the send data to the transmission buffer.
Wire1_writec	Adds one-character data to the transmission buffer. This is an internal processing function for Wire1.write.
Wire1_writEb	Adds a data block to the transmission buffer. This is an internal processing function for Wire1.write.
Wire1.endTransmission	Actually sends the send data that is set in the buffer via the I <sup>2</sup> C bus. It is possible to specify that stop conditions are generated when transmission is completed.
Wire2.begin	Initializes the I <sup>2</sup> C library of Pmod connector 2 and connects it as the master.
Wire2.requestFrom	Starts reading data from the specified slave. The read operation is processed as an interrupt.
Wire2_requestFromS	Starts reading data from the specified slave. The read operation is processed as an interrupt. It is possible to specify that stop conditions are generated when reception is completed. This is an internal processing function for Wire2.requestFrom.
Wire2_requestFromsub	This is an internal processing function for Wire2.requestFrom.
Wire2.available	Returns the number of bytes that can be read by using Wire2.read from the reception buffer.
Wire2.read	Reads data from the reception buffer.
Wire2.beginTransmission	Prepares for sending data to the specified slave.
Wire2.write	Writes the send data to the transmission buffer.
Wire2_writec	Adds one-character data to the transmission buffer. This is an internal processing function for Wire2.write.
Wire2_writEb	Adds a data block to the transmission buffer. This is an internal processing function for Wire2.write.
Wire2.endTransmission	Actually sends the send data that is set in the buffer via the I <sup>2</sup> C bus. It is possible to specify that stop conditions are generated when transmission is completed.

## 5.5 Specification of Functions

The function specifications of the sample code are shown below.

[Function name]	loop
<b>Overview</b>	Main function
<b>Header</b>	AR_LIB_PORT.h, AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, AR_SKETCH.h, r_cg_userdefine.h, LCD_LIB.h
<b>Declaration</b>	void loop(void);
<b>Description</b>	Upon activation, this function checks the elapsed time from startup and checks switch status at intervals of 16 milliseconds. When the switch is depressed or one minute has passed, the sensor (HS3001) is activated. After approximately 48 milliseconds have passed after startup, this function reads the measurement result from the sensor, calculates temperature and humidity, and then displays the calculation result on the LCD.
<b>Argument</b>	None
<b>Return value</b>	None
[Function name]	setup
<b>Overview</b>	Initialization function
<b>Header</b>	AR_LIB_PORT.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h
<b>Declaration</b>	void setup(void);
<b>Description</b>	This function sets up the pins and A/D converter used by the program (sketch).
<b>Argument</b>	None
<b>Return value</b>	None
[Function name]	get_PWM_duty
<b>Overview</b>	This function calculates the PWM signal duty ratio from the current value of the ambient light sensor.
<b>Header</b>	r_cg_macrodriver.h, r_cg_userdefine.h
<b>Declaration</b>	uint8_t get_PWM_duty (uint16_t value);
<b>Description</b>	The duty ratio (0 to 255) of the 255-cycle PWM signal is calculated from the 10-bit data specified by the argument.
<b>Argument</b>	uint16_t value : Current value of the ambient light sensor
<b>Return value</b>	uint8_t : Duty ratio (0 to 255)
[Function name]	pinMode
<b>Overview</b>	Function to set the pin function
<b>Header</b>	AR_LIB_PORT.h, r_cg_macrodriver.h, r_cg_userdefine.h
<b>Declaration</b>	void pinMode(uint8_t pin, uint8_t mode);
<b>Description</b>	The pin indicated by the first argument is set to the mode indicated by the second argument.
<b>Argument</b>	uint8_t pin : Number of the pin to be specified uint8_t mode : Specifies the pin mode with OUTPUT/INPUT/INPUT_PULLUP
<b>Return value</b>	None

---

[Function Name]	digitalWrite	
<b>Overview</b>	Function to output digital data to a pin	
<b>Header</b>	AR_LIB_PORT.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	void digitalWrite(uint8_t pin, uint8_t value);	
<b>Description</b>	The data indicated by the second argument is output to the pin indicated by the first argument.	
<b>Argument</b>	uint8_t pin :	Number of the pin for data output
	uint8_t value :	Data to output (HIGH/LOW)
<b>Return value</b>	None	

---

[Function Name]	digitalRead	
<b>Overview</b>	Function to read out digital data from a pin	
<b>Header</b>	AR_LIB_PORT.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t digitalRead(uint8_t pin);	
<b>Description</b>	The state of the pin specified by the argument is read out	
<b>Argument</b>	uint8_t pin :	Number of the pin to be read out
<b>Return value</b>	uint8_t :	Data that was red out (HIGH/LOW)

---

[Function Name]	micros	
<b>Overview</b>	Function to obtain the elapsed time in microsecond units	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	uint32_t micros(void);	
<b>Description</b>	Returns the time elapsed from startup, in microsecond units.	
<b>Argument</b>	None	
<b>Return value</b>	uint32_t	Elapsed time in microsecond units

---

[Function name]	millis	
<b>Overview</b>	Function to obtain the elapsed time in millisecond units	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	uint32_t millis (void);	
<b>Description</b>	Returns the time elapsed from startup, in millisecond units.	
<b>Argument</b>	None	
<b>Return value</b>	uint32_t :	Elapsed time in millisecond units

---

[Function Name]	delay	
<b>Overview</b>	A function that waits for a certain length of time in milliseconds	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	uint32_t delay(uint32_t time);	
<b>Description</b>	This function waits for the length of time specified for an argument in milliseconds.	
<b>Argument</b>	uint32_t time	Wait time (in milliseconds)
<b>Return value</b>	None	

---

[Function Name]	delayMicroseconds	
<b>Overview</b>	A function that waits for a certain length of time in microseconds	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	void delayMicroseconds(uint32_t time);	
<b>Description</b>	This function waits for the length of time specified for an argument in microseconds.	
<b>Argument</b>	uint32_t time	Wait time (in microseconds)
<b>Return value</b>	None	
[Function Name]	analogRead	
<b>Overview</b>	Analog input function	
<b>Header</b>	AR_LIB_ANALOG.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	uint16_t analogRead (uint8_t pin);	
<b>Description</b>	This function converts the analog voltage of the pin specified by the argument to a digital voltage, and then returns the A/D conversion result.	
<b>Argument</b>	uint8_t	Pin to be converted
<b>Return value</b>	uint16_t	A/D conversion result
[Function Name]	analogWrite	
<b>Overview</b>	Analog output function	
<b>Header</b>	AR_LIB_ANALOG.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	void analogWrite (uint8_t pin, value);	
<b>Description</b>	This function outputs the PWM signal of the specified duty ratio to the pin specified by the argument.	
<b>Argument</b>	uint8_t	Output pin
	uint8_t	Value to be output
<b>Return value</b>	None	A/D conversion result
[Function Name]	analogReference	
<b>Overview</b>	Analog reference voltage setting function	
<b>Header</b>	AR_LIB_ANALOG.h, r_cg_macrodriver.h, r_cg_userdefine.h	
<b>Declaration</b>	void analogReference (uint8_t type);	
<b>Description</b>	This function specifies the A/D conversion reference voltage.	
<b>Argument</b>	uint8_t	Reference voltage setting DEFAULT: VDD = reference voltage INTERNAL: Internal reference voltage EXTERNAL: ADREF pin voltage
<b>Return value</b>	None	

[Function Name]	Wire.begin						
<b>Overview</b>	Function that prepares for using the I <sup>2</sup> C bus						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire.begin(void);						
<b>Description</b>	This function initializes IICA0 as a preparation for using the I <sup>2</sup> C bus.						
<b>Argument</b>	None						
<b>Return value</b>	None						
[Function Name]	Wire.requestFrom						
<b>Overview</b>	Function that prepares for receiving data from the slave						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);						
<b>Description</b>	This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.						
<b>Argument</b>	<table border="0"> <tr> <td>uint8_t saddr7</td> <td>7-bit slave address</td> </tr> <tr> <td>uint16_t bytes</td> <td>Number of bytes to be received</td> </tr> <tr> <td>uint8_t stop</td> <td>Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)</td> </tr> </table>	uint8_t saddr7	7-bit slave address	uint16_t bytes	Number of bytes to be received	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
uint8_t saddr7	7-bit slave address						
uint16_t bytes	Number of bytes to be received						
uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)						
<b>Return value</b>	uint8_t 0x00: Normal 0x01: Buffer overflow 0x04: Other errors						
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>						
[Function Name]	Wire_requestFromS						
<b>Overview</b>	Function that prepares for receiving data from the slave						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire_requestFromS(uint8_t saddr7, uint16_t bytes);						
<b>Description</b>	This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it issues the stop condition and releases the bus. (This function is used for the internal processing of Wire.requestFrom.)						
<b>Argument</b>	<table border="0"> <tr> <td>uint8_t saddr7</td> <td>7-bit slave address</td> </tr> <tr> <td>uint16_t bytes</td> <td>Number of bytes to be received</td> </tr> </table>	uint8_t saddr7	7-bit slave address	uint16_t bytes	Number of bytes to be received		
uint8_t saddr7	7-bit slave address						
uint16_t bytes	Number of bytes to be received						
<b>Return value</b>	uint8_t 0x00: Normal 0x01: Buffer overflow 0x04: Other errors						
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>						

[Function Name]	Wire_requestFromSub	
<b>Overview</b>	Internal function that prepares for receiving data from the slave	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire_requestFromSub(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
<b>Description</b>	<p>This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.</p> <p>(This function is an internal function for Wire.requestFrom.)</p>	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
<b>Return value</b>	None	
<b>Remarks</b>	<p>g_status: Communication status            If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).            The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>g_erflag: Error flag            0x00: normal, 0x01: buffer overflow, 0x04: other errors</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>	

[Function Name]	Wire.available	
<b>Overview</b>	Function that returns the number of bytes that can be read	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire.available(void);	
<b>Description</b>	This function uses the Wire.requestFrom function to receive data and then returns the number of bytes of the data stored in a buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Number of bytes that can be read from the buffer

[Function Name]	Wire.read	
<b>Overview</b>	Function that reads data from the receive buffer	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire.read(void);	
<b>Description</b>	This function reads data from the buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Data read from the buffer (or 0x00)
[Function Name]	Wire.beginTransmission	
<b>Overview</b>	Function that prepares for sending data to the slave	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire.beginTransmission(uint8_t saddr7);	
<b>Description</b>	This function converts the slave address to an 8-bit address, stores it in the "sladdr8" variable, and then issues the start condition to secure the bus.	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
<b>Return value</b>	uint8_t	0x00: Normal 0x04: Other errors
<b>Remarks</b>	g_erflag: Communication status If the value that is set is 0x00, startup is successful. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	
[Function Name]	Wire.write	
<b>Overview</b>	Function that sets the send data	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire.write( uint8_t data); uint8_t Wire.write(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
<b>Argument 1</b>	uint8_t data	Data to be sent
<b>Argument 2</b>	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	
[Function Name]	Wire_writec	
<b>Overview</b>	Function that sets the send data	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire_writec(uint8_t data);	
<b>Description</b>	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire.write function.)	
<b>Argument</b>	uint8_t data	Data to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

[Function Name]	Wire_writeb	
<b>Overview</b>	Function that sets the send data	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire_writeb(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores the data of the block specified for an argument in the send buffer. (This function is an internal function that processes a block in the Wire.write function.)	
<b>Argument</b>	uint8_t *buff uint8_t bytes	Address of the data block to be sent Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

[Function Name]	Wire.endTransmission	
<b>Overview</b>	Function that sends data to the slave	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
<b>Declaration</b>	void Wire_endTransmission(uint8_t STOP);	
<b>Description</b>	This function sends data from the send buffer to the slave.	
<b>Argument</b>	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
<b>Return value</b>	uint8_t	Result of sending: 0: Success 1: The number of bytes exceeded the buffer size. 2: NACK was replied to the slave address. 3: NACK was replied to the send data. 4: Other errors

[Function Name]	Wire1.begin						
<b>Overview</b>	Function that prepares for using the I <sup>2</sup> C bus of PMOD1 connector						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire1.begin(void);						
<b>Description</b>	This function initializes IICA0 as a preparation for using the I <sup>2</sup> C bus.						
<b>Argument</b>	None						
<b>Return value</b>	None						
[Function Name]	Wire1.requestFrom						
<b>Overview</b>	Function that prepares for receiving data from the slave using the I <sup>2</sup> C bus of PMOD1 connector						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire1.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);						
<b>Description</b>	This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.						
<b>Argument</b>	<table border="0"> <tr> <td>uint8_t saddr7</td> <td>7-bit slave address</td> </tr> <tr> <td>uint16_t bytes</td> <td>Number of bytes to be received</td> </tr> <tr> <td>uint8_t stop</td> <td>Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)</td> </tr> </table>	uint8_t saddr7	7-bit slave address	uint16_t bytes	Number of bytes to be received	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
uint8_t saddr7	7-bit slave address						
uint16_t bytes	Number of bytes to be received						
uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)						
<b>Return value</b>	<table border="0"> <tr> <td>uint8_t</td> <td>0x00: Normal 0x01: Buffer overflow 0x04: Other errors</td> </tr> </table>	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors				
uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors						
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>						

[Function Name]	Wire1_requestFromS	
<b>Overview</b>	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire1_requestFromS(uint8_t saddr7, uint16_t bytes);	
<b>Description</b>	<p>This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it issues the stop condition and releases the bus.</p> <p>(This function is used for the internal processing of Wire1.requestFrom.)</p>	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
<b>Return value</b>	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>	

[Function Name]	Wire1_requestFromSub	
<b>Overview</b>	Internal function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire1_requestFromSub(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
<b>Description</b>	<p>This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.</p> <p>(This function is an internal function for Wire1.requestFrom.)</p>	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
<b>Return value</b>	None	
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>g_erflag: Error flag</p> <p>0x00: normal, 0x01: buffer overflow, 0x04: other errors</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>	

---

[Function Name]	Wire1.available	
<b>Overview</b>	Function that returns the number of bytes that can be read from the receive buffer of Wire1.	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire1.available(void);	
<b>Description</b>	This function uses the Wire1.requestFrom function to receive data and then returns the number of bytes of the data stored in a buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Number of bytes that can be read from the buffer

---

[Function Name]	Wire1.read	
<b>Overview</b>	Function that reads data from the receive buffer	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire1.read(void);	
<b>Description</b>	This function reads data from the buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Data read from the buffer (or 0x00)

---

[Function Name]	Wire1.beginTransmission	
<b>Overview</b>	Function that prepares for sending data to the slave on the I2C bus of the Pmod1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire1.beginTransmission(uint8_t saddr7);	
<b>Description</b>	This function converts the slave address to an 8-bit address, stores it in the "sladdr8" variable, and then issues the start condition to secure the bus.	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
<b>Return value</b>	uint8_t	0x00: Normal 0x04: Other errors
<b>Remarks</b>	g_erflag: Communication status If the value that is set is 0x00, startup is successful. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

[Function Name]	Wire1.write	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire1.write( uint8_t data); uint8_t Wire1.write(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
<b>Argument 1</b>	uint8_t data	Data to be sent
<b>Argument 2</b>	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

---

[Function Name]	Wire1_writec	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire1_writec(uint8_t data);	
<b>Description</b>	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire1.write function.)	
<b>Argument</b>	uint8_t data	Data to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

[Function Name]	Wire1_writeb	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire1_writeb(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores the data of the block specified for an argument in the send buffer. (This function is an internal function that processes a block in the Wire1.write function.)	
<b>Argument</b>	uint8_t *buff	Address of the data block to be sent
	uint8_t bytes	Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

[Function Name]	Wire1.endTransmission	
<b>Overview</b>	Function that sends data to the slave on the I2C bus of the Pmod1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire1_endTransmission(uint8_t STOP);	
<b>Description</b>	This function sends data from the send buffer to the slave.	
<b>Argument</b>	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
<b>Return value</b>	uint8_t	Result of sending: 0: Success 1: The number of bytes exceeded the buffer size. 2: NACK was replied to the slave address. 3: NACK was replied to the send data. 4: Other errors

---

[Function Name]	Wire2.begin
<b>Overview</b>	Function that prepares for using the I <sup>2</sup> C bus of PMOD1 connector
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h
<b>Declaration</b>	void Wire2.begin(void);
<b>Description</b>	This function initializes IICA0 as a preparation for using the I <sup>2</sup> C bus.
<b>Argument</b>	None
<b>Return value</b>	None

[Function Name]	Wire2.requestFrom						
<b>Overview</b>	Function that prepares for receiving data from the slave using the I <sup>2</sup> C bus of PMOD1 connector						
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h						
<b>Declaration</b>	void Wire2.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);						
<b>Description</b>	This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.						
<b>Argument</b>	<table> <tr> <td>uint8_t saddr7</td> <td>7-bit slave address</td> </tr> <tr> <td>uint16_t bytes</td> <td>Number of bytes to be received</td> </tr> <tr> <td>uint8_t stop</td> <td>Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)</td> </tr> </table>	uint8_t saddr7	7-bit slave address	uint16_t bytes	Number of bytes to be received	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
uint8_t saddr7	7-bit slave address						
uint16_t bytes	Number of bytes to be received						
uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)						
<b>Return value</b>	<table> <tr> <td>uint8_t</td> <td>0x00: Normal 0x01: Buffer overflow 0x04: Other errors</td> </tr> </table>	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors				
uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors						
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>						

[Function Name]	Wire2_requestFromS	
<b>Overview</b>	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire2_requestFromS(uint8_t saddr7, uint16_t bytes);	
<b>Description</b>	<p>This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it issues the stop condition and releases the bus.</p> <p>(This function is used for the internal processing of Wire2.requestFrom.)</p>	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
<b>Return value</b>	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>	

[Function Name]	Wire2_requestFromSub	
<b>Overview</b>	Internal function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire2_requestFromSub(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
<b>Description</b>	<p>This function issues the start condition and sends the slave address so that data can be received under the conditions specified by using arguments. The subsequent processing is performed as forms of interrupts. When this function ends, it performs the processing specified by the third argument.</p> <p>(This function is an internal function for Wire2.requestFrom.)</p>	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
	uint8_t stop	Processing to be performed when the function ends (If this argument is omitted, the function releases the bus.) 0: Issues the restart condition. (The bus is held.) 1: Issues the stop condition. (The bus is released.)
<b>Return value</b>	None	
<b>Remarks</b>	<p>g_status: Communication status</p> <p>If the value that is set is 0x50, startup is successful. Afterward, the value changes to 0x60 (now receiving), and then to 0x70 (reception completed).</p> <p>The other values are as follows: 0x81: buffer error, 0x84: no data received, 0x8F: startup failed</p> <p>g_erflag: Error flag</p> <p>0x00: normal, 0x01: buffer overflow, 0x04: other errors</p> <p>Processing that starts communication with the I<sup>2</sup>C bus must not be performed during execution of this function.</p>	

---

[Function Name]	Wire2.available	
<b>Overview</b>	Function that returns the number of bytes that can be read from the receive buffer of Wire2.	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire2.available(void);	
<b>Description</b>	This function uses the Wire2.requestFrom function to receive data and then returns the number of bytes of the data stored in a buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Number of bytes that can be read from the buffer

---

[Function Name]	Wire2.read	
<b>Overview</b>	Function that reads data from the receive buffer	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire2.read(void);	
<b>Description</b>	This function reads data from the buffer.	
<b>Argument</b>	None	
<b>Return value</b>	uint8_t	Data read from the buffer (or 0x00)

---

[Function Name]	Wire2.beginTransmission	
<b>Overview</b>	Function that prepares for sending data to the slave on the I2C bus of the Pmod2 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire2.beginTransmission(uint8_t saddr7);	
<b>Description</b>	This function converts the slave address to an 8-bit address, stores it in the "sladdr8" variable, and then issues the start condition to secure the bus.	
<b>Argument</b>	uint8_t saddr7	7-bit slave address
<b>Return value</b>	uint8_t	0x00: Normal 0x04: Other errors
<b>Remarks</b>	g_erflag: Communication status If the value that is set is 0x00, startup is successful. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

[Function Name]	Wire2.write	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire2.write( uint8_t data); uint8_t Wire2.write(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
<b>Argument 1</b>	uint8_t data	Data to be sent
<b>Argument 2</b>	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

---

[Function Name]	Wire2_writec	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire2_writec(uint8_t data);	
<b>Description</b>	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire2.write function.)	
<b>Argument</b>	uint8_t data	Data to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

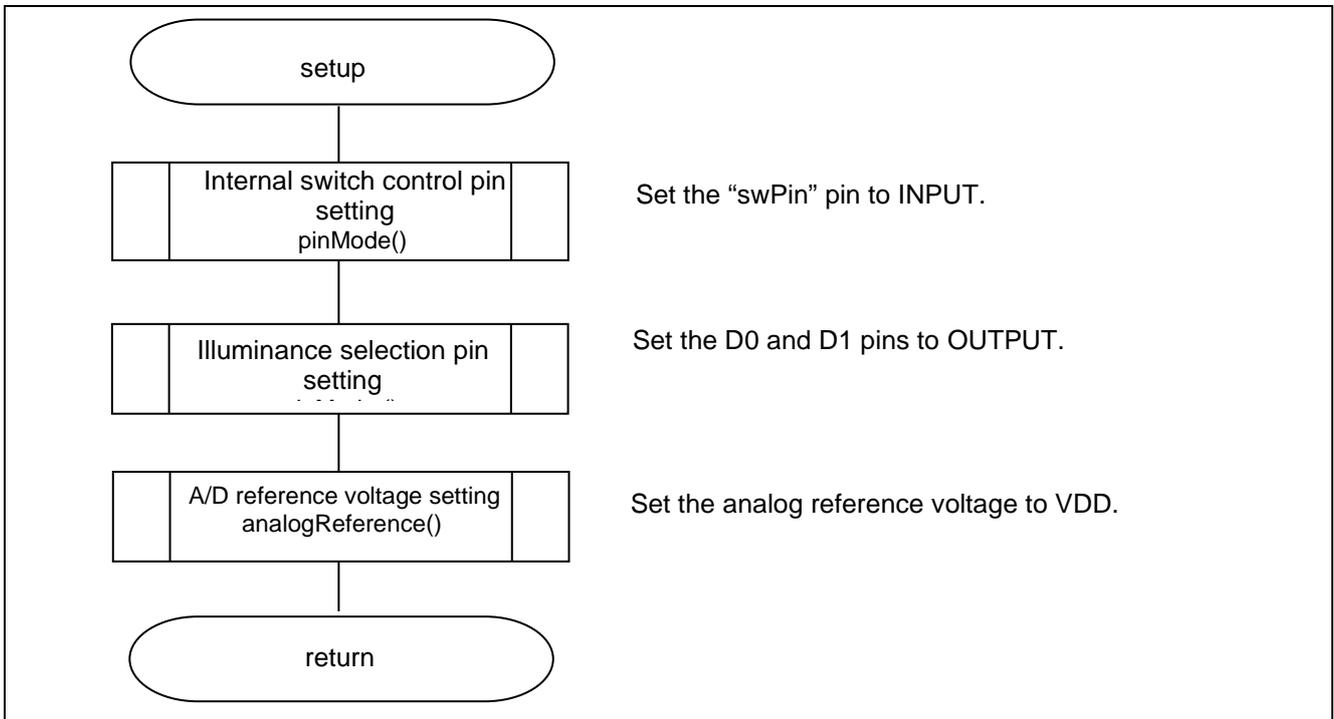
[Function Name]	Wire2_writeb	
<b>Overview</b>	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	uint8_t Wire2_writeb(uint8_t *buff, uint8_t bytes);	
<b>Description</b>	This function stores the data of the block specified for an argument in the send buffer. (This function is an internal function that processes a block in the Wire2.write function.)	
<b>Argument</b>	uint8_t *buff uint8_t bytes	Address of the data block to be sent Number of bytes to be sent
<b>Return value</b>	uint8_t	Number of bytes stored in the buffer
<b>Remarks</b>	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I <sup>2</sup> C bus.	

[Function Name]	Wire2.endTransmission	
<b>Overview</b>	Function that sends data to the slave on the I2C bus of the Pmod2 connector	
<b>Header</b>	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
<b>Declaration</b>	void Wire2_endTransmission(uint8_t STOP);	
<b>Description</b>	This function sends data from the send buffer to the slave.	
<b>Argument</b>	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
<b>Return value</b>	uint8_t	Result of sending: 0: Success 1: The number of bytes exceeded the buffer size. 2: NACK was replied to the slave address. 3: NACK was replied to the send data. 4: Other errors

## 5.6 Flowcharts

### 5.6.1 Initial setting function

Figure 5.1 shows the flowchart of the initial setting.



**Figure 5.1 Initial setting function**

5.6.2 Main Processing Function

Figure 5.2 to Figure 5.3 show a flowchart of the main processing function.

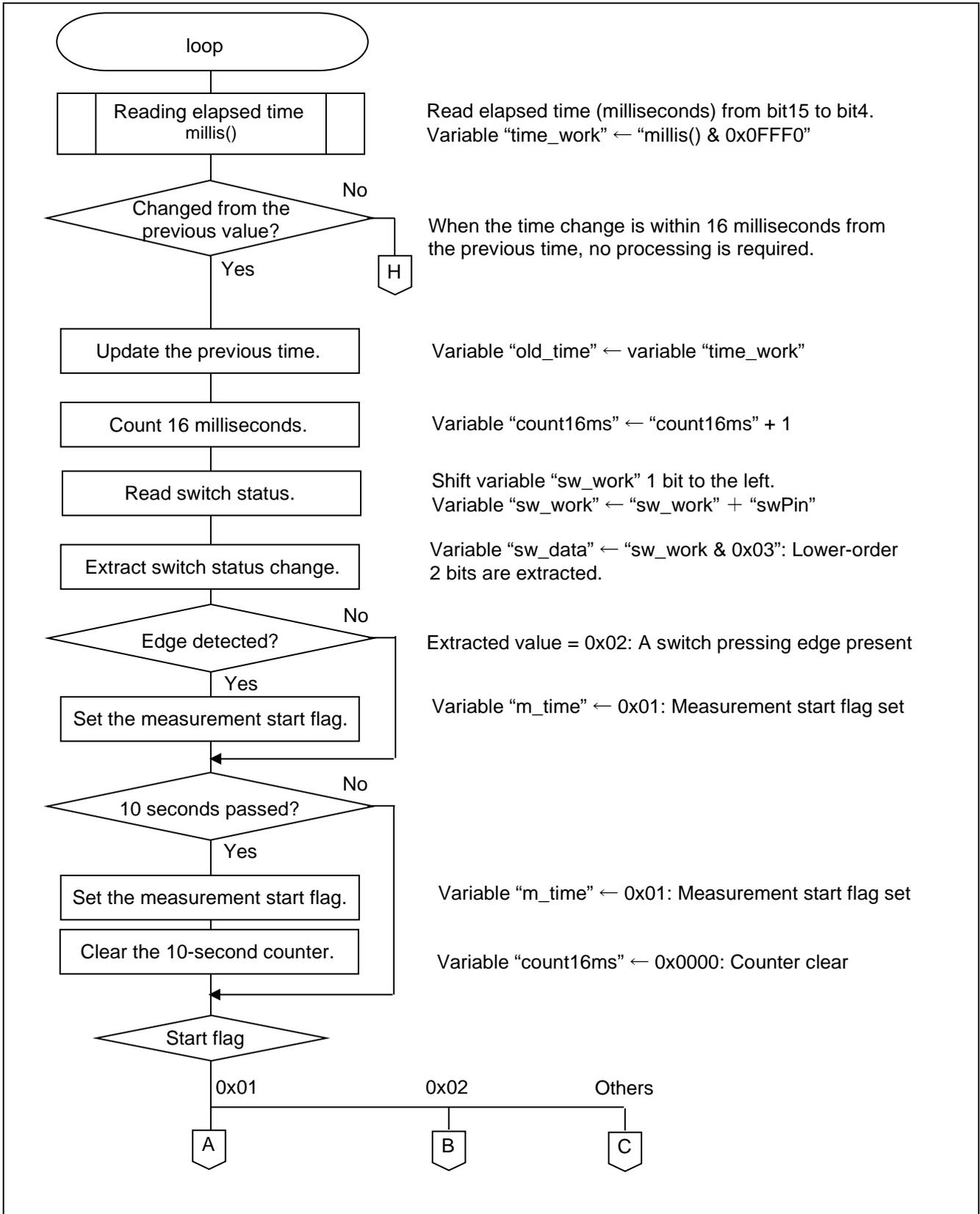


Figure 5.2 Main Function (1/2)

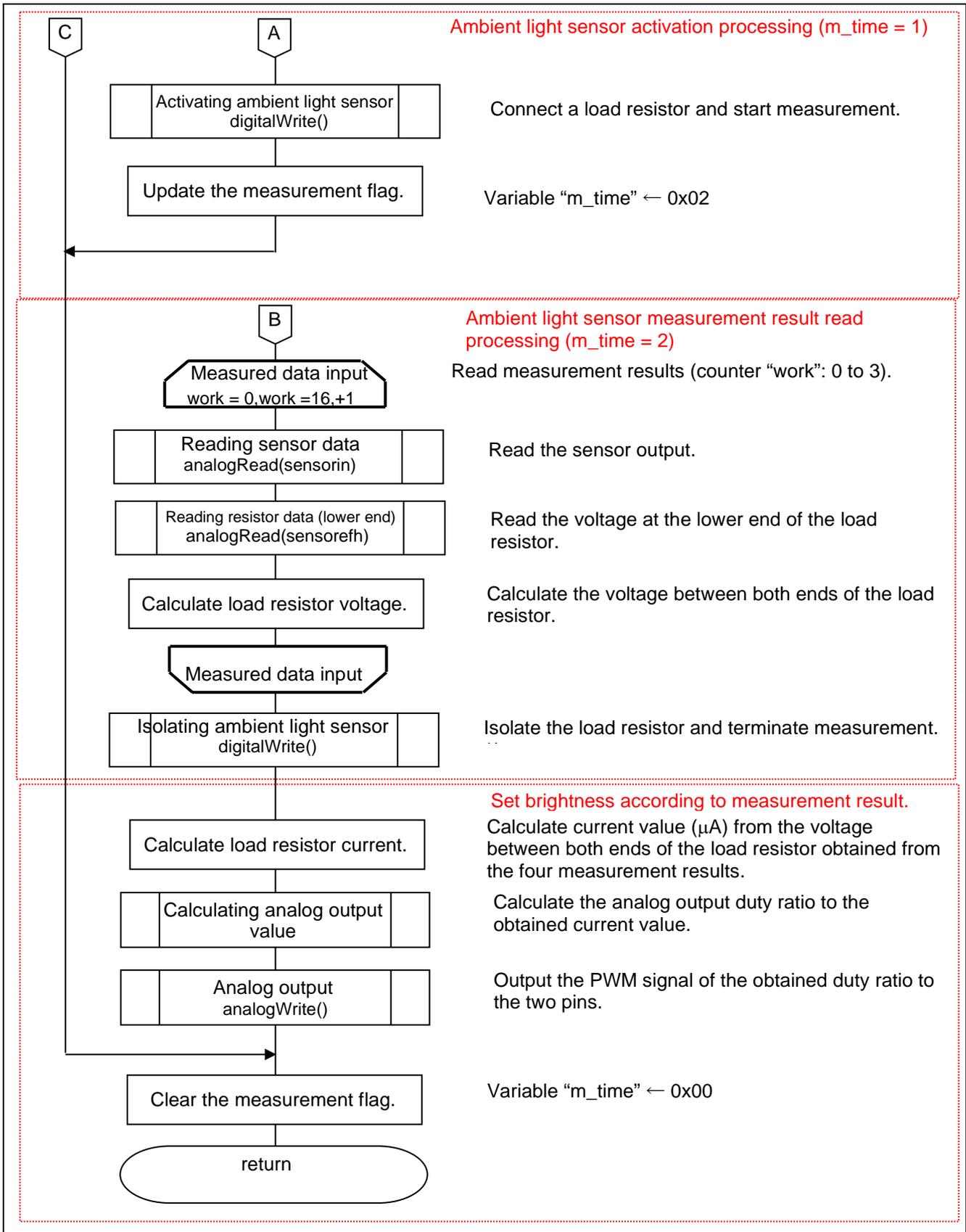
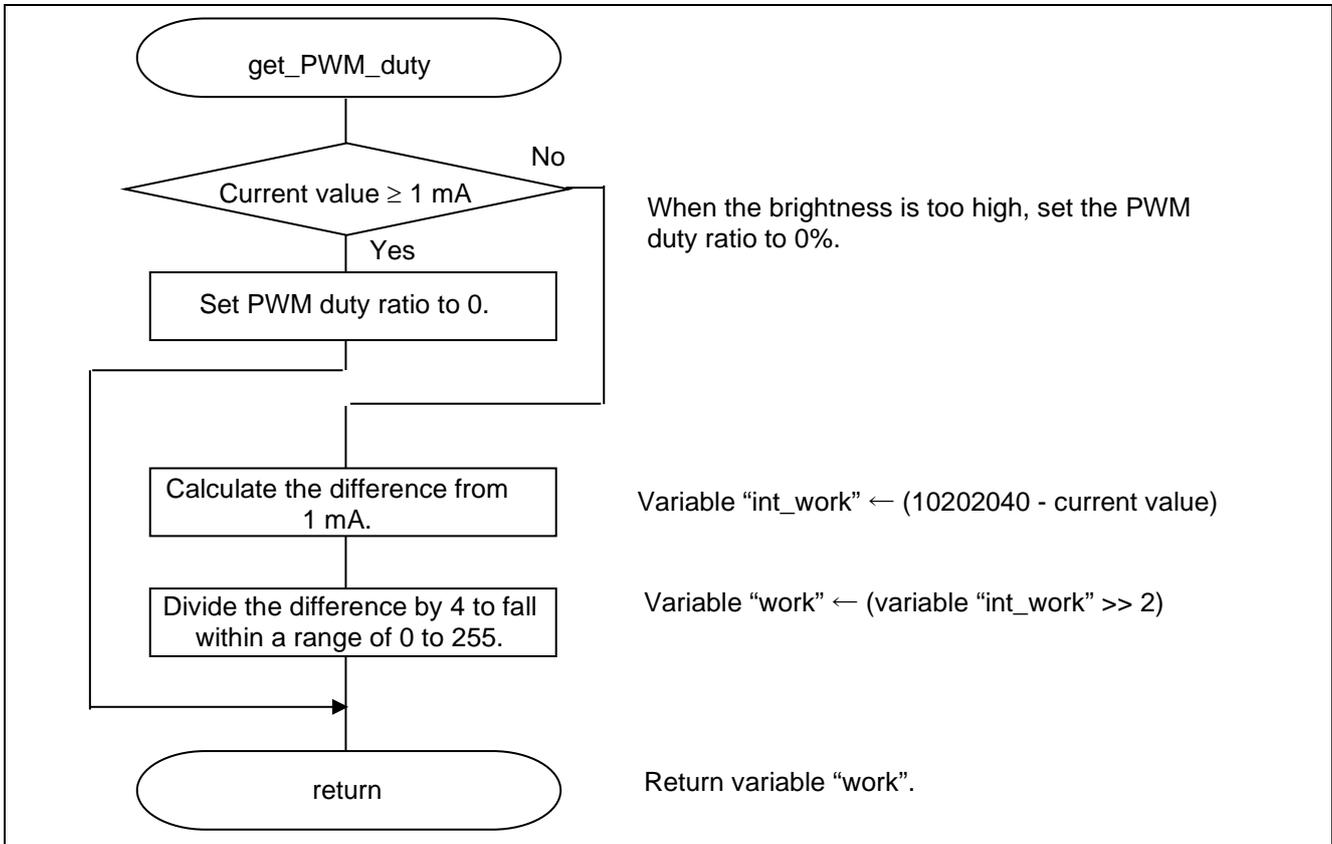


Figure 5.3 Main Function (2/2)

**5.6.3 Processing function to obtain the PWM signal duty ratio from the current value of the ambient light sensor**

Figure 5.4 shows a flowchart of the processing function to obtain the PWM signal duty ratio from the current value of the ambient light sensor.



**Figure 5.4 Processing function to obtain the PWM signal duty ratio from the current value of the ambient light sensor**

Remark: To simplify the calculation, the upper-limit value is set to 1020 (μA) and the difference from this value is calculated. Right-shift the calculation result by two bits so that a value from 0 to 1020 falls within a range of 0 to 255.

## 6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

## 7. Reference Documents

RL78/G14 User's Manual: Hardware (R01UH0186)

RL78 family User's Manual: Software (R01US0015)

RL78/G14 Fast Prototyping Board User's Manual (R20UT4573)

(The latest versions can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News

(The latest versions can be downloaded from the Renesas Electronics website.)

## Website and Support

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/contact/>

All trademarks and registered trademarks are the property of their respective owners.

**Revision History**

Rev.	Date	Description	
		Page	Summary
1.00	2022.3.14	—	First Edition

## 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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://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/](http://www.renesas.com/contact/).