RL78/G14

I2C Bus Control Using Simplified IIC (Arduino API)

Introduction

In this application note, a program written in a language such as Arduino is used to control the temperature/humidity sensor HDC1080 connected to the I2C bus of the Pmod connector on 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

This application note describes how the temperature/humidity sensor HDC1080 is controlled in Fast mode (at 380 kbps) by using a program coded in a language such as Arduino via the I^{2}C bus of Pmod connector 1 on the FPB. This application note also describes how the data obtained from the temperature/humidity sensor is displayed on the LCD indicator (16 chars x 2 lines) in Standard mode (at 85 kbps) via the I^{2}C bus of the Arduino connector.

Data is obtained from the temperature/humidity sensor and displayed on the LCD indicator on a regular basis at one-minute intervals or by pressing a switch.

Pmod connectors 1 and 2 on the RL78/G14 FPB use different pin positions for SCL and SDA signals on the I^{2}C bus from those pin positions on the standard Pmod connector. To connect to an I^{2}C module that supports Pmod, as shown in Figure 1.1, prepare a separate conversion board that exchanges SCL and SDA signals.

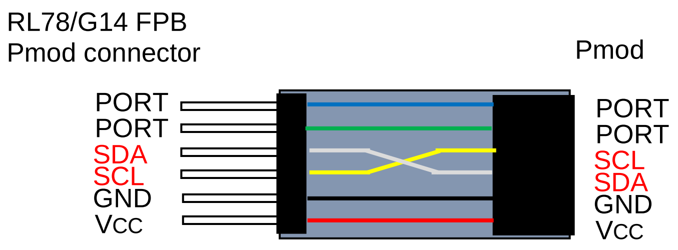
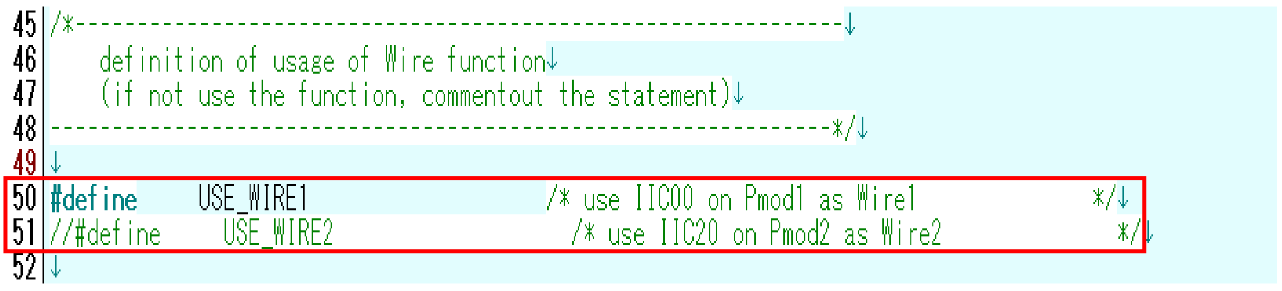


Figure 1.1 Signal conversion

The RL78/G14 FPB is equipped with two Pmod connectors. The temperature/humidity sensor HDC1080 is controlled by using Wire1 allocated to Pmod connector 1.

To use Wire2 allocated to Pmod connector 2, processing to enable Wire2 must be performed. In r_cg_userdefine.h, comment out line 50, and then uncomment line 51 and change the name of the API function used in AR_SKETCH.c from Wire1 to Wire2 (see the lines surrounded in a red frame in Figure 1.2).



```
45 /*---------------------------------------------------------------↓
46     definition of usage of Wire function↓
47     (if not use the function, commentout the statement)↓
48 ---------------------------------------------------------------*/↓
49 ↓
50 #define     USE_WIRE1                    /* use IIC00 on Pmod1 as Wire1              */↓
51 //#define     USE_WIRE2                    /* use IIC20 on Pmod2 as Wire2              */↓
52 ↓
```

Figure 1.2 Lines where the WireAPI function to be used is defined

Table 1.1 lists peripheral functions to be used.

Table 1.1 Peripheral functions used and their uses

Peripheral Function	Use
Digital input	Reads the status of the switch (SW_USR).
IICA0	Controls the LCD indicator via the I^{2}C bus.
IIC00	Controls the sensor via the I^{2}C bus of the Pmod1 connector.
IIC20	Controls the sensor via the I^{2}C bus of the Pmod2 connector.
Timer array unit	Measures the 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 1.3.

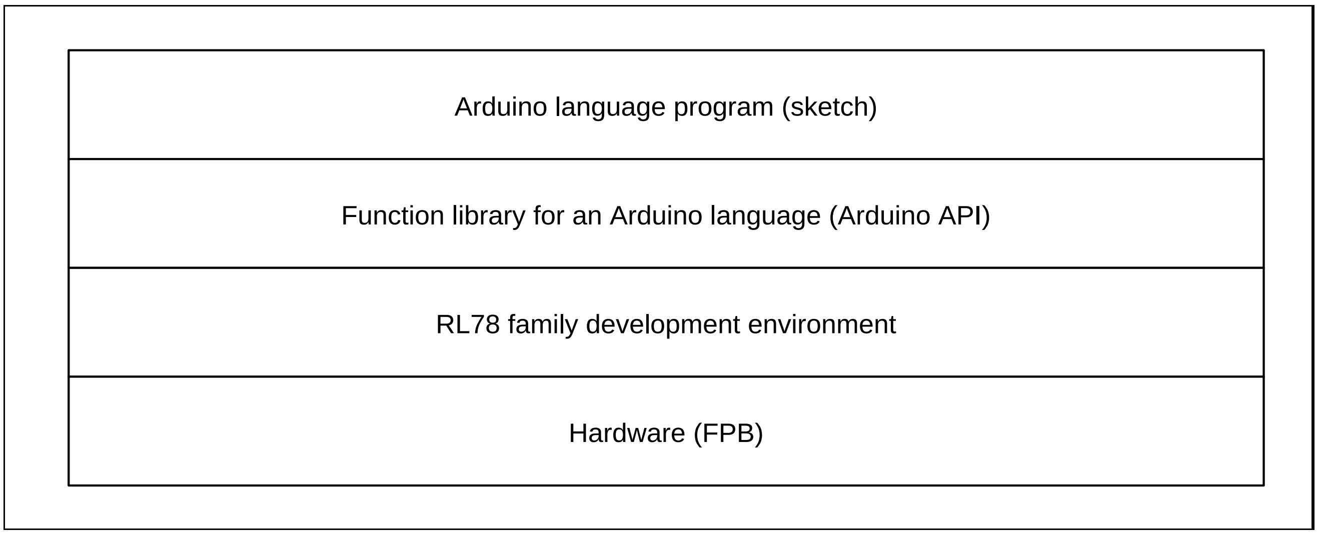

Arduino language program (sketch)
Function library for an Arduino language (Arduino API)
RL78 family development environment
Hardware (FPB)

Figure 1.3 Program execution environment

Library functions that can be used in this application note are shown in Table1.2 to Table 1.4.

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.

Table1.3 Library functions (2/3)

Item	Library Function	Function
I2C control (Wire)	Wire.begin()	Initializes IICA0 and connects to the I^{2}C bus as the master.

	Wire.requestFrom(saddr7, bytes, stop)
	Wire.requestFrom(saddr7, bytes)
		Receives data with the size specified by "bytes" from the specified slave.
		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.
Simplified IIC control (Wire1)	Wire1.begin()	Initializes IIC00 (Wire1) connected to Pmod connector 1 and connects to the I^{2}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.

Note: The slave function of the I^{2}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.

Table 1.4 Library functions (3/3)

Simplified IIC control (Wire2)	Wire2.begin()	Initializes IIC00 (Wire2) connected to Pmod connector 1 and connects to the I^{2}C bus as the master.

	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^{2}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.

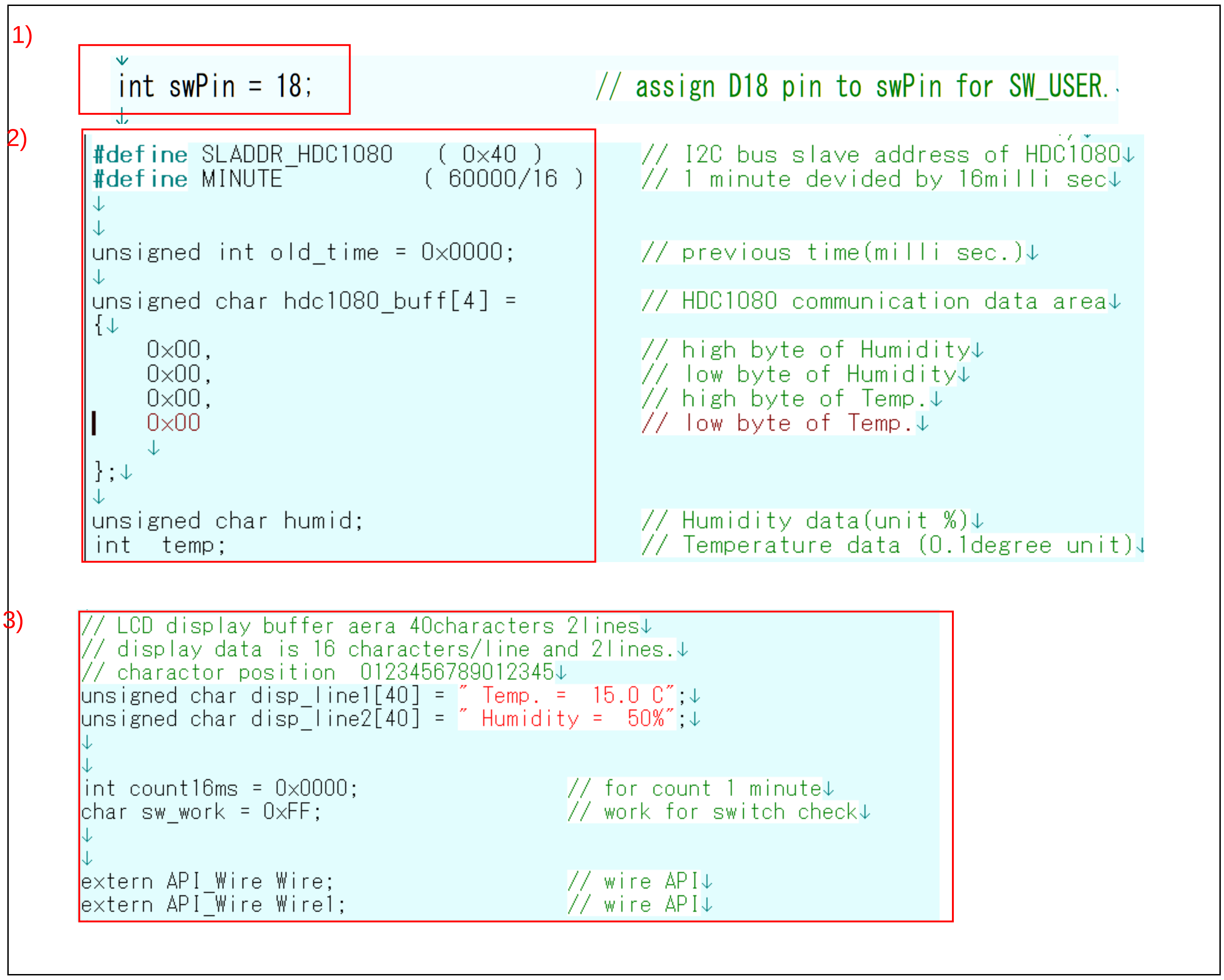
1.3 Preparations for Project Startup

Preparations for project startup are different depending on the integrated development environment 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 1.4.



1)

```
int swPin = 18;                          // assign D18 pin to swPin for SW_USER.
```

2)

```
#define SLADDR_HDC1080    ( 0x40 )       // I2C bus slave address of HDC1080
#define MINUTE           ( 60000/16 )    // 1 minute devided by 16milli sec


unsigned int old_time = 0x0000;          // previous time(milli sec.)

unsigned char hdc1080_buff[4] =          // HDC1080 communication data area
{
        0x00,                            // high byte of Humidity
        0x00,                            // low byte of Humidity
        0x00,                            // high byte of Temp.
        0x00                             // low byte of Temp.

};

unsigned char humid;                     // Humidity data(unit %)
int   temp;                              // Temperature data (0.1degree unit)
```

3)

```
// LCD display buffer aera 40characters 2lines
// display data is 16 characters/line and 2lines.
// charactor position  0123456789012345
unsigned char disp_line1[40] = " Temp.  =  15.0 C";
unsigned char disp_line2[40] = " Humidity =  50%";


int count16ms = 0x0000;                  // for count 1 minute
char sw_work = 0xFF;                     // work for switch check


extern API_Wire Wire;                    // wire API
extern API_Wire Wire1;                   // wire API
```

Figure 1.4 Program definition details

1) "18" is set for the swPin pin that controls the on-board SW_USR switch so that the pin is assigned to D18.

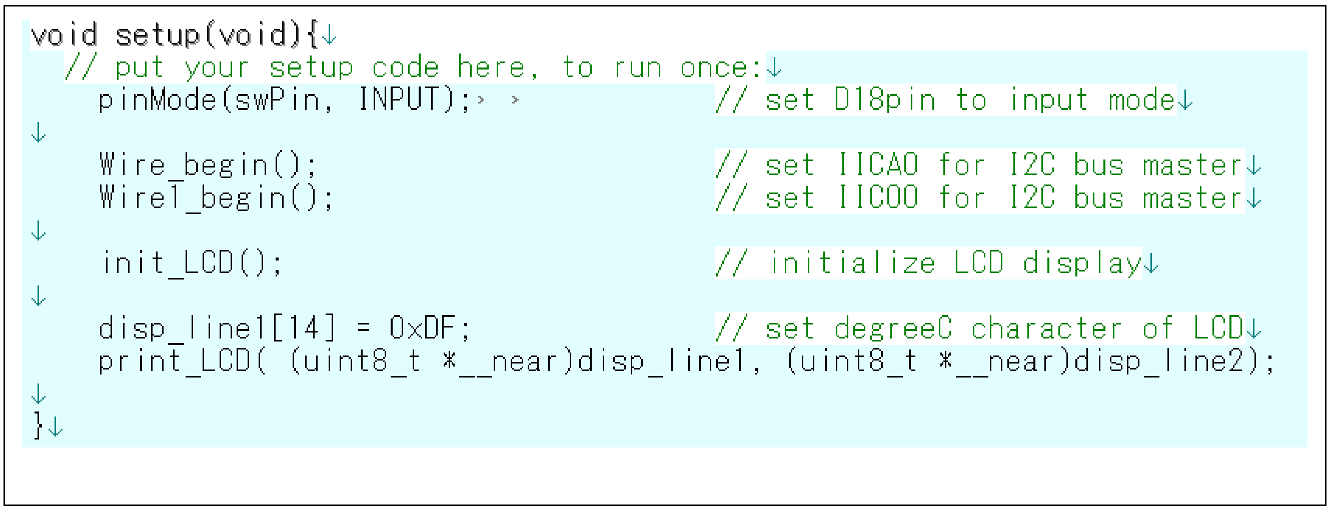
2) Then, the following items are defined: the "old_time" 16-bit variable to check the elapsed time (in milliseconds), the "hdc1080_buff" 4-byte array for communication use to control the HDC1080 sensor, the "humid" variable to store the humidity data obtained, and the "temp" variable to store temperature data in units of 0.1 degrees.

3) In the display data area for the LCD indicator, the following 40-byte arrays are defined: the "disp_line1" variable to store the data for line 1, the "disp_line2" variable to store the data for line 2.
In addition, the "count16ms" counter to obtain 1 minute by counting 16-ms intervals and the "sw_work" variable to check the switch are defined.
The "Wire1" API_Wire-type structure is used to reference any objects that are defined by AR_LIB_WIRE1.c that provides Wire-related API functions.

1.5 Initial Setting Processing

The initial settings section of the program (sketch) is shown in Figure 1.5.

The "setup" function specifies that the switch input pin be used for input. Also, IICA0 and IIC00 are set as the I²C bus master. Then, the initial display data is set for the LCD indicator.

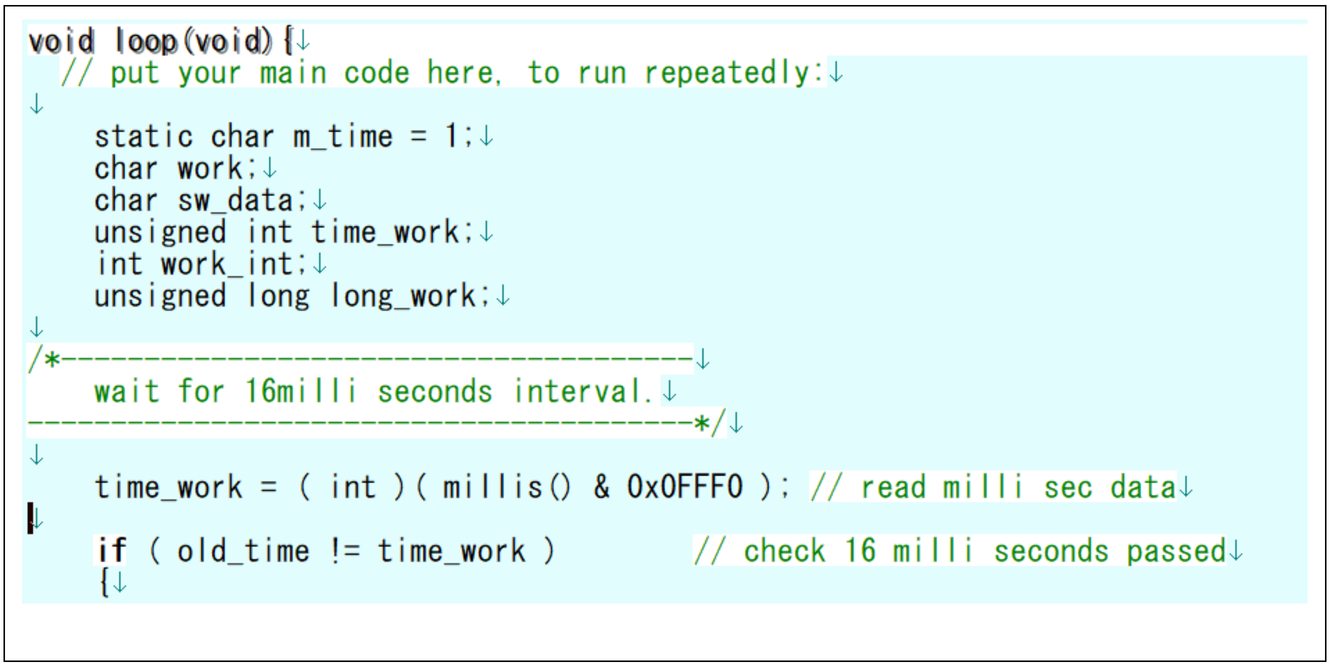


```
void setup(void){↓
  // put your setup code here, to run once:↓
    pinMode(swPin, INPUT);› ›           // set D18pin to input mode↓
↓
    Wire_begin();                       // set IICA0 for I2C bus master↓
    Wire1_begin();                      // set IIC00 for I2C bus master↓
↓
    init_LCD();                         // initialize LCD display↓
↓
    disp_line1[14] = 0xDF;              // set degreeC character of LCD↓
    print_LCD( (uint8_t *__near)disp_line1, (uint8_t *__near)disp_line2);
↓
}↓
```

Figure 1.5 Initial setting processing section

1.6 Main Processing Part

The leading section of the main processing, which is executed repeatedly, is shown in Figure 1.6. 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;↓
    int work_int;↓
    unsigned long long_work;↓
↓
/*----------------------------------------↓
    wait for 16milli seconds interval.↓
----------------------------------------*/↓
↓
    time_work = ( int )( millis() & 0x0FFF0 ); // read milli sec data↓
↓
    if ( old_time != time_work )          // check 16 milli seconds passed↓
    {↓
```

Figure 1.6 The leading section of the main process

1.7 Data Processing Performed by HDC1080

Normally, HDC1080 is placed in sleep mode. To obtain the humidity data and temperature data, a measurement request must be issued.

The measurement request is issued by writing the slave address of HDC1080 followed by 0x00.

To obtain 14-bit humidity/temperature data from HDC1080, measurement takes 6.5 ms (TYP.) to complete.

In this application note, after issuing an measurement request, the software waits for 32 ms and then reads the measurement results.

エラー! 参照元が見つかりません。 shows an example of 4-byte data read from HDC1080. In the example, temperature data is indicated in red and humidity data is indicated in blue.

hdc1080_buff[0]								hdc1080_buff[1]								hdc1080_buff[2]								hdc1080_buff[3]							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 1.7 HDC1080 Data Format

Temperature is obtained by using the following expressions. Expression ① is used to get 16-bit length data. ② is used to convert the resulting data to a value in 0.1°C degrees. ③ is used to subtract 40°C as the offset. The temperature in units of 0.1°C degrees is obtained in this way.

```
① long_work = ( hdc1080_buff[0] * 0x100UL + ( hdc1080_buff[1] );
② long_work *= 1650;     // multiply 10 times of Maximum temperature
③ temp = (int)((long_work >> 16) - 400); // adjust offset (40degreeC)
```

Humidity is obtained by using the following expressions. Expression ① is used to get 16-bit length data. Expression ② is used to multiply the value by 100. ③ is used to obtain the humidity data. The humidity data in percentage is obtained in this way.

```
① long_work = ( hdc1080_buff[2] * 0x100UL + hdc1080_buff[3] );
② long_work *= 100UL;     // get percentage
③ humid = (unsigned char)(long_work >>16); // get humidity
```

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_FPB)
Operating frequency	● High-speed on-chip oscillator clock (f_{IH}): 32 MHz ● CPU/peripheral hardware clock: 32 MHz
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^2 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

エラー! 参照元が見つかりません。 shows the hardware (FPB) that is used in this application note.

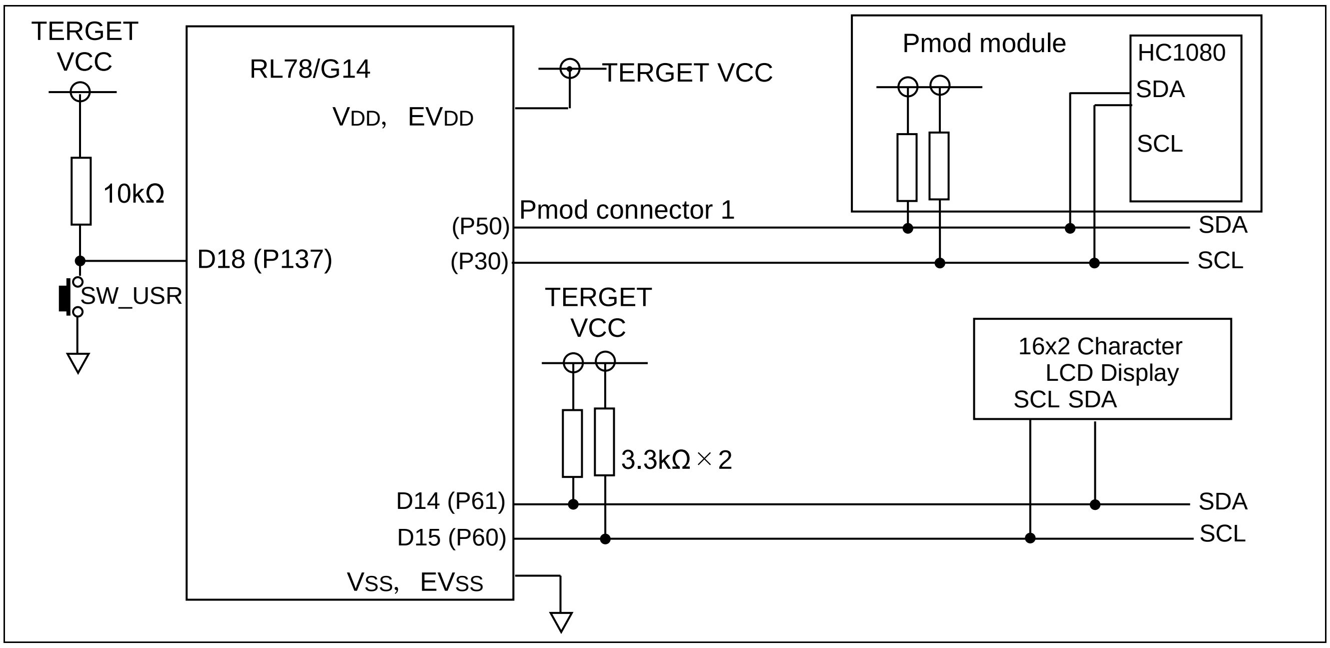


Figure 4.1 Hardware configuration example

Note: 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
D14	**P61**	**Input/Output**	SDA (Data line of I2C-bus)
D15	**P60**	**Input/Output**	SCL (Clock line of I2C-bus)
-	**P50**	**Input/Output**	SDA of PMOD1 connector (Data line of I2C-bus)
-	**P30**	**Input/Output**	SCL of PMOD1 connector (Clock line of I2C-bus)
-	**P14**	**Input/Output**	SDA of PMOD2 connector (Data line of I2C-bus)
-	**P15**	**Input/Output**	SCL of PMOD2 connector (Clock line of I2C-bus)
D18	**P137**	**Input**	Switch (SW_USR) input

5. Software

5.1 Summary of Operation

In this application note, when the software completes initial setup (pin setup) and the main processing (loop) starts, the LCD indicator is placed in the initial display status.

Data is obtained from the temperature/humidity sensor HDC1080 on a regular basis at one-minute intervals or by pressing a switch. The temperature and humidity are calculated from the obtained data, and then the calculation results are displayed on the LCD indicator.

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

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

- The software sets the read pin of the on-board SW_USR (swPin) to digital input.
- To control the I^2C bus by using the D14 and D15 pins, the software sets IICA0 to Master.
- To control the I^2C bus by using Pmod connector 1, the software sets IIC00.
- The software initializes the LCD indicator connected to the I^2C bus to reset the indication.

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

- The software obtains the data of 12 bits (in units of 16 milliseconds) from bits 15 to 4 as the time elapsed since startup in milliseconds.
- The software checks whether the obtained data has changed from the old data (old_time).
- If the data has not changed, the software terminates processing and returns control to the beginning of the "loop" function.
- If the data has changed (16 milliseconds elapsed), the software replaces the data in old_time by the obtained data.
- The software starts the counter that is reset at one-minute intervals (0xEA6).
- The software checks the status of the switch connected to D18.
- If the switch is not pressed and one minute has not elapsed, the software returns control to the beginning of the "loop" function. Note
- If the switch is pressed or one minute has elapsed, the software measures the temperature and humidity. Note
- The temperature/humidity sensor HDC1080 exits standby mode and the software starts measurement.
    - ◆ The software waits for about 16 milliseconds until the data that can be obtained becomes stable.
    - ◆ The software reads data from the sensor.
    - ◆ The software calculates the temperature and humidity from the data that was read.
- The software transfers the calculation results to the LCD indicator.
- The software returns control to the beginning of the "loop" function.

Note: Immediately after startup, the software uses the sensor to measure the temperature and humidity.

5.2 List of Constants

Table 5.1 エラー! 参照元が見つかりません。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**	Number of the pin from which SW_USR is read
DUMMY_DATA	**0xFF**	Data to be written for starting reception during master reception
RELEASE	**1**	Specifies that stop conditions are generated when communication is completed.
RESTART	**0**	Specifies that restart conditions are generated when communication is completed.
SLADDR_HC1080	**0x40**	Slave address of the sensor (7 bits)
SLADDR_LCD	**0x50**	Slave address of the LCD indicator (7 bits)
COMBYTE	**0x00**	Data specifying that a command is sent to the LCD indicator
DATABYTE	**0x80**	Data specifying that data is transferred to the LCD indicator
CLRDISP	**0x01**	Command that clears the indication of the LCD indicator
HOMEPOSI	**0x02**	Moves the cursor of the LCD indicator to the home position.
LCD_Mode	**0x38**	Specifies that each character is displayed with 5x8 dots on two lines.
DISPON	**0x0F**	Turns on and blinks the cursor.
ENTRY_Mode	**0x06**	Moves the display position each time one character is transferred.
LOOPLIMIT	**1000**	Sets the maximum number of times starts and stops can be detected to 1,000.
SUCCESS	**0x00**	Indicates that processing of the I^2C bus terminated normally.
BUS_FREE	**0x00**	The I^2C bus is idle.
BUS_ERROR	**0x8F**	The I^2C bus could not be secured.
GET_BUS	**0x10**	The I^2C bus was secured.
GET_BUS4TX	**0x20**	The I^2C bus was secured for transmission.
TX_MODE	**0x30**	Transmission mode
TX_END	**0x40**	Transmission was completed.
GET_BUS4RX	**0x50**	The I^2C bus was secured for reception.
RX_MODE	**0x60**	Reception mode
RX_END	**0x70**	Reception was completed.
BUFF_OVER	**0x81**	The number of bytes that were sent exceeded the buffer capacity.
NO_SLAVE	**0x82**	The relevant slave does not exist.
NO_ACK	**0x83**	NACK was replied to the data that was sent.
NO_DATA	**0x84**	The number of bytes that were received is 0.
MINUTE	**60000/16**	The counter is incremented by16 milliseconds to obtain 1 minute.
TX_DELAY1	**1**	Transmission interval of Wire1 of Pmod connector 1 (1 μs)
RX_DELAY1	**10**	Reception interval of Wire1 of Pmod connector 1 (10 μs)
TX_DELAY2	**1**	Transmission interval of Wire2 of Pmod connector 2 (1 μs)
RX_DELAY2	**10**	Reception interval of Wire2 of Pmod connector 2 (10 μs)

5.3 List of Variables

Table 5.2 to Note: This is shown by the name of the internal processing function, not the Arduino API.

Table 5.4 lists global variables.

Table 5.2 Global variables (1/3)

Type	Variable Name	Description	Function used Note
unsigned int	old_time	Time elapsed since the previous startup (in milliseconds)	loop()
unsigned char	hdc1080_buff[4]	Buffer for the data read from the sensor	loop()
unsigned char	humid	Humidity data	loop()
unsigned int	temp	Temperature data in units of 0.1 °C	loop()
char	disp_line1[40]	Data displayed on the LCD indicator (line 1)	loop()
char	disp_line2[40]	Data displayed on the LCD indicator (line 2)	loop()
int	count16ms	The counter is incremented by 16 milliseconds to obtain 1 minute.	loop()
char	sw_work	Variable for checking the switch status every 16 milliseconds	loop()
unsigned char	g_lcd_command[2]	Variable for setting a command on the LCD indicator	set_command()
unsigned char	g_lcd_data[2]	Variable for setting data on the LCD indicator	set_dat()
uint8_t	gp_tx_set	Pointer for writing data to the transmission buffer (maximum: 255)	Wire_begin(), Wire_beginTransmission(), Wire_write()
uint8_t	gp_tx_get	Pointer for reading data from the transmission buffer	Wire_begin(), Wire_beginTransmission(), r_IICA0_interrupt()
uint8_t	g_tx_buff[256]	Transmission buffer	Wire_write(), r_IICA0_interrupt()
uint8_t	gp_rx_set	Pointer for writing data to the transmission buffer (maximum: 255)	Wire_begin(), Wire_requestFrom(), r_IICA0_interrupt()
uint8_t	gp_rx_get	Pointer for reading data from the reception buffer	Wire_begin(), Wire_requestFrom(), Wire_read()
uint8_t	g_rx_buff[256]	Reception buffer	r_IICA0_interrupt(), Wire_read()
uint16_t	g_rx_num	Number of bytes received	Wire_requestFrom(), r_IICA0_interrupt()
uint8_t	sladdr8	8-bit slave address	Wire_beginTransmission(), Wire_requestFromSub(), Wire_requestFromb()
uint8_t	g_stop_flag	Flag indicating whether to generate stop conditions at termination 0: Performs nothing. 1: Generates stop conditions at termination.	Wire_endTransmission(), Wire_requestFrom(), r_IICA0_interrupt(), r_operation_end()
uint8_t	g_status	IICA0 status flag 0x00: BUS FREE 0x8F: BUS Error 0x10: Get bus 0x20: Get bus to transmit 0x30: Transmit operation 0x40: Transmit end 0x50: Get bus to receive 0x60: Receive operation 0x70: Receive end 0x81: Data size over buffer size 0x82: NACK for slave address 0x83 : No ACK for data	r_IICA0_interrupt(), Wire_beginTransmission(), Wire_endTransmission(), Wire_requestFromb(), Wire_requestFromSub(), r_IICA0_interrupt(), r_operation_end()
uint8_t	g_erflag	0x00: Success 0x01: Buffer overflow 0x02: No slave exists. 0x03: NACK was replied to the data that was sent. 0x04: Other errors	Wire_endTransmission()

Note: This is shown by the name of the internal processing function, not the Arduino API.

Table 5.3 Global variables (2/3)

Type	Variable Name	Description	Function used Note
uint8_t	gp_tx1_set	Pointer for writing data to the transmission buffer (max: 255)	Wire1_begin(), Wire1_beginTransmission(), Wire1_write()
uint8_t	gp_tx1_get	Pointer for reading data from the transmission buffer	Wire1_begin(), Wire1_beginTransmission(), r_IIC00_interrupt()
uint8_t	g_tx1_buff[256]	Transmission buffer	Wire1_write(), r_IIC00_interrupt()
uint8_t	gp_rx1_set	Pointer for writing data to the reception buffer (max: 255)	Wire1_begin(), Wire1_requestFrom(), r_IIC00_interrupt()
uint8_t	gp_rx1_get	Pointer for reading data from the reception buffer	Wire1_begin(), Wire1_requestFrom(), Wire1_read()
uint8_t	g_rx1_buff[256]	Reception buffer	r_IIC00_interrupt(), Wire1_read()
uint16_t	g_rx1_num	Number of bytes received	Wire1_requestFrom(), r_IIC00_interrupt()
uint8_t	sladdr8_1	8-bit slave address	Wire1_beginTransmission(), Wire1_requestFromSub(), Wire1_requestFromb()
uint8_t	g_stop_flag_1	Flag indicating whether to generate stop conditions at termination 0: Generates restart conditions at termination. 1: Generates stop conditions at termination.	Wire1_endTransmission(), Wire1_requestFrom(), r_IIC00_interrupt(), r_operation_end_1()
uint8_t	g_status_1	IIC00 status flag 0x00: BUS FREE 0x8F: BUS Error 0x10: Get bus 0x20: Get bus to transmit 0x30: Transmit operation 0x40: Transmit end 0x50: Get bus to receive 0x60: Receive operation 0x70: Receive end 0x81: Data size over buffer size 0x82: NACK for slave address 0x83: No ACK for data	Wire1_beginTransmission(), Wire1_endTransmission(), Wire1_requestFromb(), Wire1_requestFromSub(), r_IIC00_interrupt(), r_operation_end_1()
uint8_t	g_erflag_1	0x00: Success 0x01: Buffer overflow 0x02: No slave exists. 0x03: NACK was replied to the data that was sent. 0x04: Other errors	Wire1_endTransmission()

Note: This is shown by the name of the internal processing function, not the Arduino API.

Table 5.4 Global variables (3/3)

Type	Variable Name	Description	Function used Note
uint8_t	gp_tx2_set	Pointer for writing data to the transmission buffer (max: 255)	Wire2_begin(), Wire2_beginTransmission(), Wire2_write()
uint8_t	gp_tx2_get	Pointer for reading data from the transmission buffer	Wire2_begin(), Wire2_beginTransmission(), r_IIC20_interrupt()
uint8_t	g_tx2_buff[256]	Transmission buffer	Wire2_write(), r_IIC20_interrupt()
uint8_t	gp_rx2_set	Pointer for writing data to the reception buffer (max: 255)	Wire2_begin(), Wire2_requestFrom(), r_IIC20_interrupt()
uint8_t	gp_rx2_get	Pointer for reading data from the reception buffer	Wire2_begin(), Wire2_requestFrom(), Wire2_read()
uint8_t	g_rx2_buff[256]	Reception buffer	r_IIC20_interrupt(), Wire2_read()
uint16_t	g_rx2_num	Number of bytes received	Wire2_requestFrom(), r_IIC20_interrupt()
uint8_t	sladdr8_2	8-bit slave address	Wire2_beginTransmission(), Wire2_requestFromSub(), Wire2_requestFromb()
uint8_t	g_stop_flag_2	Flag indicating whether to generate stop conditions at termination 0: Generates restart conditions at termination. 1: Generates stop conditions at termination.	Wire2_endTransmission(), Wire2_requestFrom(), r_IIC20_interrupt(), r_operation_end_2()
uint8_t	g_status_2	IIC00 status flag 0x00: BUS FREE 0x8F: BUS Error 0x10: Get bus 0x20: Get bus to transmit 0x30: Transmit operation 0x40: Transmit end 0x50: Get bus to receive 0x60: Receive operation 0x70: Receive end 0x81: Data size over buffer size 0x82: NACK for slave address 0x83: No ACK for data	Wire2_beginTransmission(), Wire2_endTransmission(), Wire2_requestFromb(), Wire2_requestFromSub(), r_IIC20_interrupt(), r_operation_end_2()
uint8_t	g_erflag_2	0x00: Success 0x01: Buffer overflow 0x02: No slave exists. 0x03: NACK was replied to the data that was sent. 0x04: Other errors	Wire2_endTransmission()

Note: This is shown by the name of the internal processing function, not the Arduino API.

5.4 List of Functions

Table 5.5 to Table 5.6 shows a list of functions.

Table 5.5 List of functions (1/2)

Function Name	Overview
loop	Main processing (sketch)
setup	Initialization function (sketch)
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.
Wire.begin	Initializes the I^{2}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_writeb	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^{2}C bus. It is possible to specify that stop conditions are generated when transmission is completed.
init_LCD	Initializes the LCD indicator.
print_LCD	Displays 1-screen data (16 characters x 2 lines) for the LCD indicator.
move_cursor	Specifies the cursor position at which to set the data to be displayed on the LCD indicator.
set_2digit	Displays 1-byte data with 2 digits.
set_1digit	Displays the lower-bit data with one digit.
set_command	Sends a command to the LCD indicator.
set_data	Sends the data to be displayed to the LCD indicator.

Table 5.6 List of functions (2/2)

Function Name	Overview
Wire1.begin	Initializes the I^{2}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^{2}C bus. It is possible to specify that stop conditions are generated when transmission is completed.
Wire2.begin	Initializes the I^{2}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^{2}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
Overview	Main function
Header	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
Declaration	void loop(void);
Description	After startup, this function checks the status of the switch at 16-ms intervals. If the switch is pressed or 1 minute elapses, the function starts the sensor (HDC1080). After the sensor starts, when about 16 milliseconds elapse, the function reads the measurement results of the sensor. The function then calculates the temperature and humidity from the measurement results and displays the calculation results on the LCD indicator.
Argument	None
Return value	None

[Function name]	setup
Overview	Initialization function
Header	AR_LIB_PORT.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h
Declaration	void setup(void);
Description	This function sets up the pins, IICA0, and LCD indicator used by the program (sketch).
Argument	None
Return value	None

[Function name]	pinMode	
Overview	Function to set the pin function	
Header	AR_LIB_PORT.h, r_cg_macrodriver.h, r_cg_userdefine.h	
Declaration	void pinMode(uint8_t pin,uint8_t mode);	
Description	The pin indicated by the first argument is set to the mode indicated by the second argument.	
Argument	uint8_t pin	Number of the pin to be specified
	uint8_t mode	Specifies the pin mode with OUTPUT/INPUT/INPUT_PULLUP
Return value	None	

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

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

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

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

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

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

[Function Name]	Wire.begin	
Overview	Function that prepares for using the I^{2}C bus	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	void Wire.begin(void);	
Description	This function initializes IICA0 as a preparation for using the I^{2}C bus.	
Argument	None	
Return value	None	

[Function Name]	Wire.requestFrom	
Overview	Function that prepares for receiving data from the slave	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	void Wire.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
Description	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.	
Argument	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.)
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire_requestFromS	
Overview	Function that prepares for receiving data from the slave	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	void Wire_requestFromS(uint8_t saddr7, uint16_t bytes);	
Description	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.)	
Argument	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire_requestFromSub	
Overview	Internal function that prepares for receiving data from the slave	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	void Wire_requestFromSub(uint8_t saddr7, uint16_t bytes , uint8_t stop);	
Description	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. (This function is an internal function for Wire.requestFrom.)	
Argument	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.)
Return value	None	
Remarks	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 g_erflag: Error flag 0x00: normal, 0x01: buffer overflow, 0x04: other errors Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

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

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

[Function Name]	Wire.beginTransmission	
Overview	Function that prepares for sending data to the slave	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	void Wire.beginTransmission(uint8_t saddr7);	
Description	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.	
Argument	uint8_t saddr7	7-bit slave address
Return value	uint8_t	0x00: Normal 0x04: Other errors
Remarks	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^{2}C bus.	

[Function Name]	Wire.write	
Overview	Function that sets the send data	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	uint8_t Wire.write(uint8_t data); uint8_t Wire.write(uint8_t *buff, uint8_t bytes);	
Description	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
Argument 1	uint8_t data	Data to be sent
Argument 2	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^{2}C bus.	

[Function Name]	Wire_writec	
Overview	Function that sets the send data	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	uint8_t Wire_writec(uint8_t data);	
Description	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire.write function.)	
Argument	uint8_t data	Data to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I²C bus.	

[Function Name]	Wire_writeb	
Overview	Function that sets the send data	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	uint8_t Wire_writeb(uint8_t *buff, uint8_t bytes);	
Description	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.)	
Argument	uint8_t *buff	Address of the data block to be sent
	uint8_t bytes	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	If the value of "g_erflag" is 0x01, the send buffer has overflowed. If the value is 0x04, the function failed to secure the I²C bus.	

[Function Name]	Wire.endTransmission	
Overview	Function that sends data to the slave	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	void Wire_ endTransmission(uint8_t STOP);	
Description	This function sends data from the send buffer to the slave.	
Argument	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
Return value	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]	init_LCD	
Overview	Function that initializes the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	uint8_t init_LCD(void);	
Description	This function sets the LCD indicator in 16 (characters) x 2 (lines) mode and clears the display.	
Argument	None	
Return value	uint8_t	Communication result: 0: Success 2: The LCD indicator does not respond.

[Function Name]	print_LCD	
Overview	Function that sets 1-screen data (16 characters x 2 lines) for the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	void print_LCD(uint8_t *point, uint8_t *point2);	
Description	This function displays 32 characters from the address passed by an argument on two lines of the LCD indicator.	
Argument	uint8_t *point	Specifies the start address of disp_line1.
	uint8_t *point2	Specifies the start address of disp_line2.
Return value	None	

[Function Name]	move_cursor	
Overview	Function that sets the cursor position on the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	void move_cursor(uint8_t col, uint8_t row);	
Description	This function moves the cursor to the position passed by an argument.	
Argument	uint8_t col	Specifies the column position on the line.
	uint8_t row	Specifies the line position.
Return value	None	
Remarks	After this function is run, the next write operation must not be performed before 60 microseconds elapse.	

[Function Name]	set_2digit	
Overview	Function that displays a numeric value with 2-digit ASCII codes on the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	void set_2digit(uint8_t datacode);	
Description	This function receives hexadecimal or BCD data via arguments, converts the data to 2-digit ASCII codes, and then sends the ASCII codes as display data to the LCD indicator.	
Argument	uint8_t datacode	8-bit data code (hexadecimal or BCD data) to be sent to the LCD indicator
Return value	None	

[Function Name]	set_1digit	
Overview	Function that displays a numeric value with a 1-digit ASCII code on the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h	
Declaration	void set_1digit(uint8_t datacode);	
Description	This function receives data via an argument, converts the last 4 bits of the data to an ASCII code, and then sends the conversion results as display data to the LCD indicator.	
Argument	uint8_t datacode	Data code to be sent to the LCD indicator (hexadecimal or BCD data)
Return value	None	

[Function Name]	set_command	
Overview	Function that sends a command to the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	uint8_t set_command(uint8_t lcd_command);	
Description	This function receives a command code via an argument and sends it as a command to the LCD indicator.	
Argument	uint8_t lcd_command	Command code to be sent to the LCD indicator
Return value	uint8_t	Communication result: 0: Success 2: The LCD indicator does not respond.
Remarks	After this function is run, the next write operation must not be performed before 60 microseconds elapse.	

[Function Name]	set_data	
Overview	Function that sends display data to the LCD indicator	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, r_cg_userdefine.h	
Declaration	uint8_t set_data(uint8_t datacode);	
Description	This function receives a data code via an argument and sends it as data to the LCD indicator.	
Argument	uint8_t datacode	Data code to be sent to the LCD indicator
Return value	uint8_t	Communication result: 0: Success 2: The LCD indicator does not respond.
Remarks	After this function is run, the next write operation must not be performed before 60 microseconds elapse.	

[Function Name]	Wire1.begin	
Overview	Function that prepares for using the I^{2}C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1.begin(void);	
Description	This function initializes IICA0 as a preparation for using the I^{2}C bus.	
Argument	None	
Return value	None	

[Function Name]	Wire1.requestFrom	
Overview	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
Description	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.	
Argument	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.)
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire1_requestFromS	
Overview	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1_requestFromS(uint8_t saddr7, uint16_t bytes);	
Description	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 Wire1.requestFrom.)	
Argument	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire1_requestFromSub	
Overview	Internal function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1_requestFromSub(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
Description	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. (This function is an internal function for Wire1.requestFrom.)	
Argument	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.)
Return value	None	
Remarks	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 g_erflag: Error flag 0x00: normal, 0x01: buffer overflow, 0x04: other errors Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

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

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

[Function Name]	Wire1.beginTransmission	
Overview	Function that prepares for sending data to the slave on the I2C bus of the Pmod1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1.beginTransmission(uint8_t saddr7);	
Description	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.	
Argument	uint8_t saddr7	7-bit slave address
Return value	uint8_t	0x00: Normal 0x04: Other errors
Remarks	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^{2}C bus.	

[Function Name]	Wire1.write	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	uint8_t Wire1.write(uint8_t data); uint8_t Wire1.write(uint8_t *buff, uint8_t bytes);	
Description	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
Argument 1	uint8_t data	Data to be sent
Argument 2	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^{2}C bus.	

[Function Name]	Wire1_writec	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	uint8_t Wire1_writec(uint8_t data);	
Description	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire1.write function.)	
Argument	uint8_t data	Data to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^{2}C bus.	

[Function Name]	Wire1_writeb	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	uint8_t Wire1_writeb(uint8_t *buff, uint8_t bytes);	
Description	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.)	
Argument	uint8_t *buff	Address of the data block to be sent
	uint8_t bytes	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^{2}C bus.	

[Function Name]	Wire1.endTransmission	
Overview	Function that sends data to the slave on the I2C bus of the Pmod1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE1.h, r_cg_userdefine.h	
Declaration	void Wire1_ endTransmission(uint8_t STOP);	
Description	This function sends data from the send buffer to the slave.	
Argument	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
Return value	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	
Overview	Function that prepares for using the I^{2}C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2.begin(void);	
Description	This function initializes IICA0 as a preparation for using the I^{2}C bus.	
Argument	None	
Return value	None	

[Function Name]	Wire2.requestFrom	
Overview	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
Description	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.	
Argument	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.)
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire2_requestFromS	
Overview	Function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2_requestFromS(uint8_t saddr7, uint16_t bytes);	
Description	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 Wire2.requestFrom.)	
Argument	uint8_t saddr7	7-bit slave address
	uint16_t bytes	Number of bytes to be received
Return value	uint8_t	0x00: Normal 0x01: Buffer overflow 0x04: Other errors
Remarks	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 Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

[Function Name]	Wire2_requestFromSub	
Overview	Internal function that prepares for receiving data from the slave using the I2C bus of PMOD1 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2_requestFromSub(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
Description	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. (This function is an internal function for Wire2.requestFrom.)	
Argument	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.)
Return value	None	
Remarks	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 g_erflag: Error flag 0x00: normal, 0x01: buffer overflow, 0x04: other errors Processing that starts communication with the I^{2}C bus must not be performed during execution of this function.	

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

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

[Function Name]	Wire2.beginTransmission	
Overview	Function that prepares for sending data to the slave on the I2C bus of the Pmod2 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2.beginTransmission(uint8_t saddr7);	
Description	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.	
Argument	uint8_t saddr7	7-bit slave address
Return value	uint8_t	0x00: Normal 0x04: Other errors
Remarks	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^2C bus.	

[Function Name]	Wire2.write	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	uint8_t Wire2.write(uint8_t data); uint8_t Wire2.write(uint8_t *buff, uint8_t bytes);	
Description	This function stores one character specified for argument 1 or the data block specified for argument 2 in the send buffer.	
Argument 1	uint8_t data	Data to be sent
Argument 2	uint8_t *buff	Data block to be sent
	uint8_t byte	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^2C bus.	

[Function Name]	Wire2_writec	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	uint8_t Wire2_writec(uint8_t data);	
Description	This function stores one character specified for argument 1. (This function is an internal function that processes 1 character in the Wire2.write function.)	
Argument	uint8_t data	Data to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^2C bus.	

[Function Name]	Wire2_writeb	
Overview	Function that sets the send data to the slave on the I2C bus of the Pmod2 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	uint8_t Wire2_writeb(uint8_t *buff, uint8_t bytes);	
Description	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.)	
Argument	uint8_t *buff	Address of the data block to be sent
	uint8_t bytes	Number of bytes to be sent
Return value	uint8_t	Number of bytes stored in the buffer
Remarks	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^2C bus.	

[Function Name]	Wire2.endTransmission	
Overview	Function that sends data to the slave on the I2C bus of the Pmod2 connector	
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE2.h, r_cg_userdefine.h	
Declaration	void Wire2_ endTransmission(uint8_t STOP);	
Description	This function sends data from the send buffer to the slave.	
Argument	uint8_t STOP	Processing performed when sending is completed: 0: Issues the restart condition to secure the bus. 1: Releases the bus.
Return value	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.

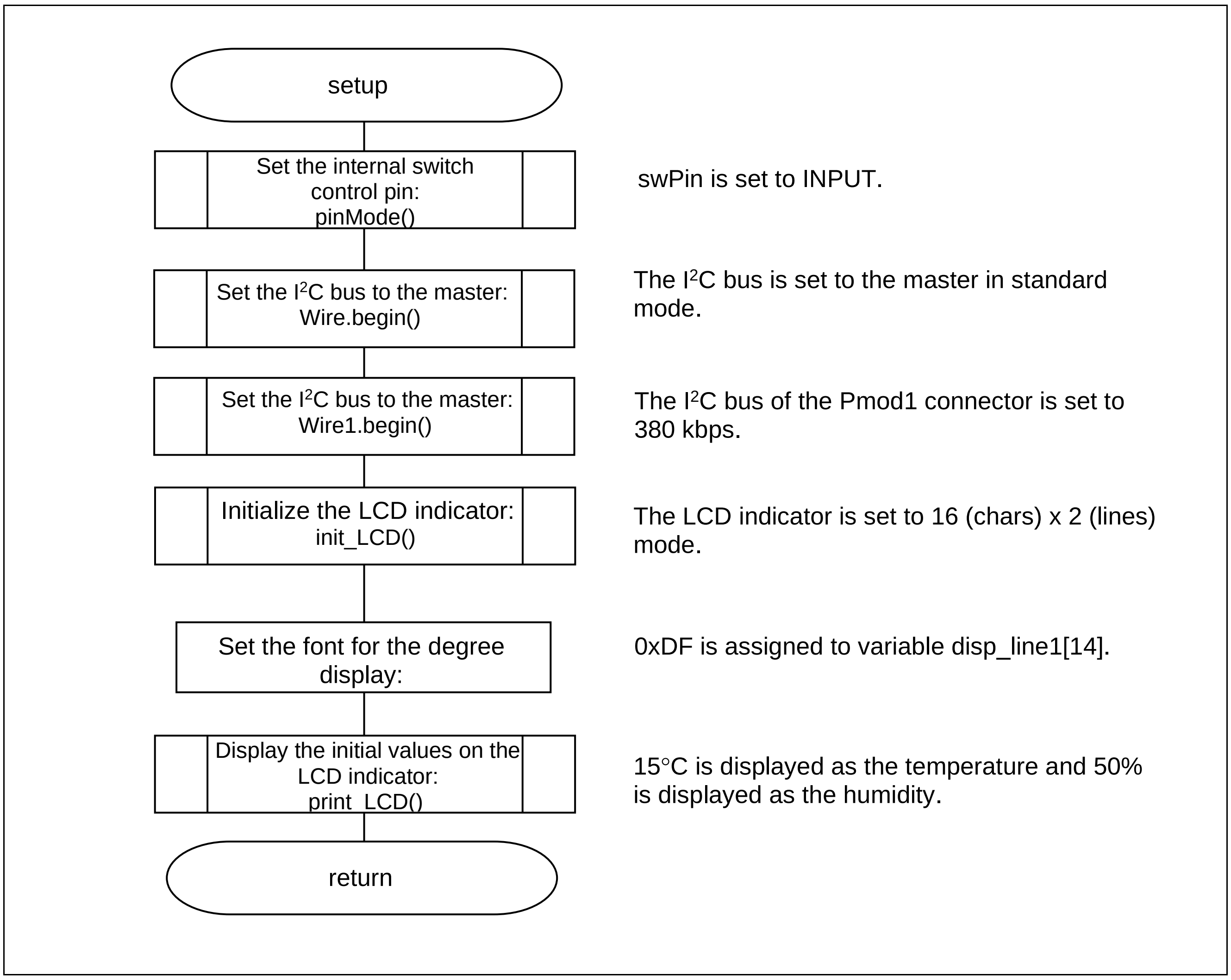
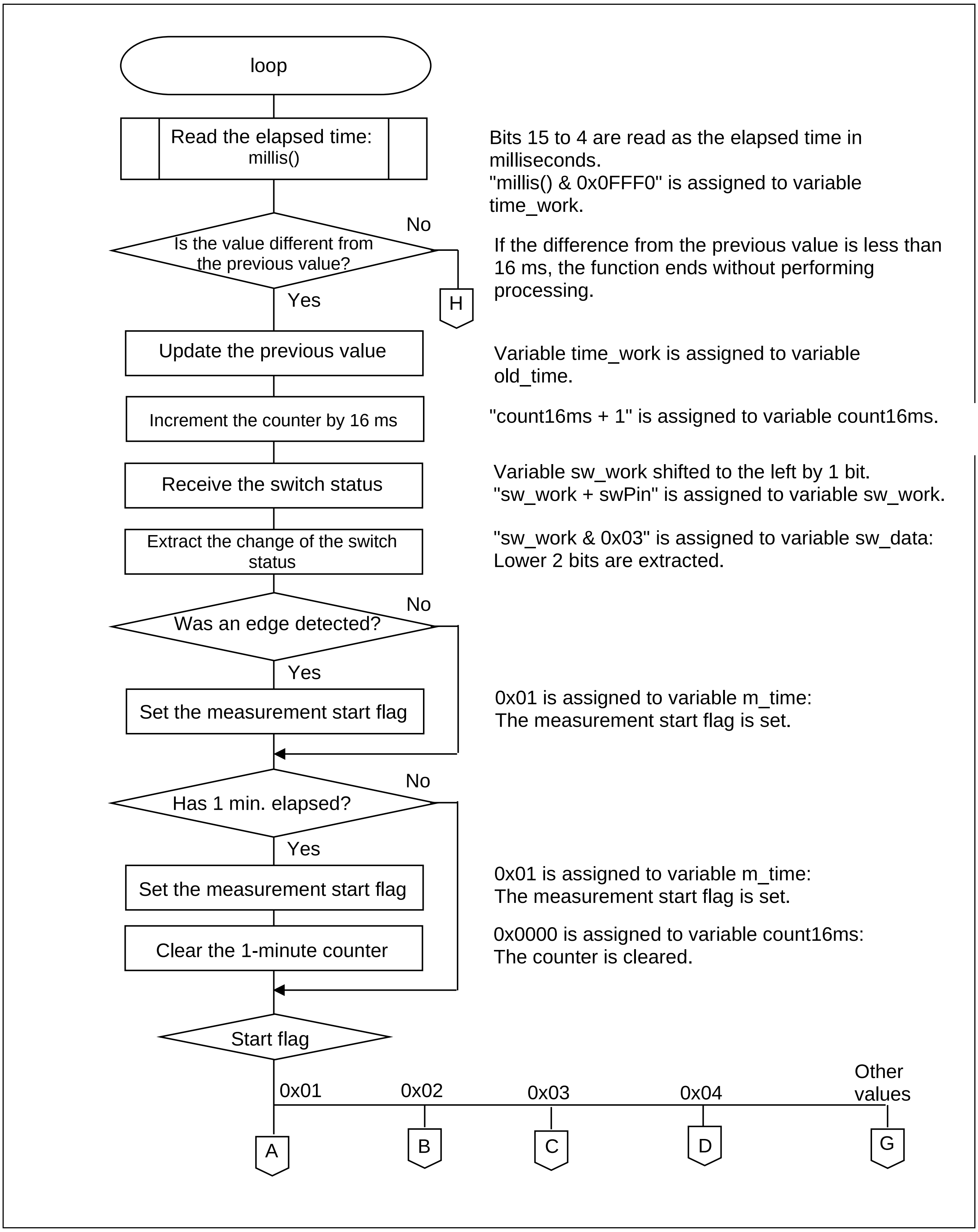

setup
Set the internal switch control pin: pinMode()
swPin is set to INPUT.
Set the I2C bus to the master: Wire.begin()
The I2C bus is set to the master in standard mode.
Set the I2C bus to the master: Wire1.begin()
The I2C bus of the Pmod1 connector is set to 380 kbps.
Initialize the LCD indicator: init_LCD()
The LCD indicator is set to 16 (chars) x 2 (lines) mode.
Set the font for the degree display:
0xDF is assigned to variable disp_line1[14].
Display the initial values on the LCD indicator: print_LCD()
15°C is displayed as the temperature and 50% is displayed as the humidity.
return

Figure 5.1 Initial setting function

5.6.2 Main Processing Function

Figure 5.2 to Figure 5.5 show a flowchart of the main processing function.



loop
Read the elapsed time: millis()
Bits 15 to 4 are read as the elapsed time in milliseconds. "millis() & 0x0FFF0" is assigned to variable time_work.
Is the value different from the previous value?
No
Yes
H
If the difference from the previous value is less than 16 ms, the function ends without performing processing.
Update the previous value
Variable time_work is assigned to variable old_time.
Increment the counter by 16 ms
"count16ms + 1" is assigned to variable count16ms.
Receive the switch status
Variable sw_work shifted to the left by 1 bit. "sw_work + swPin" is assigned to variable sw_work.
Extract the change of the switch status
"sw_work & 0x03" is assigned to variable sw_data: Lower 2 bits are extracted.
Was an edge detected?
No
Yes
Set the measurement start flag
0x01 is assigned to variable m_time: The measurement start flag is set.
Has 1 min. elapsed?
No
Yes
Set the measurement start flag
0x01 is assigned to variable m_time: The measurement start flag is set.
Clear the 1-minute counter
0x0000 is assigned to variable count16ms: The counter is cleared.
Start flag
0x01
0x02
0x03
0x04
Other values
A
B
C
D
G

Figure 5.2 Main Function (1/4)

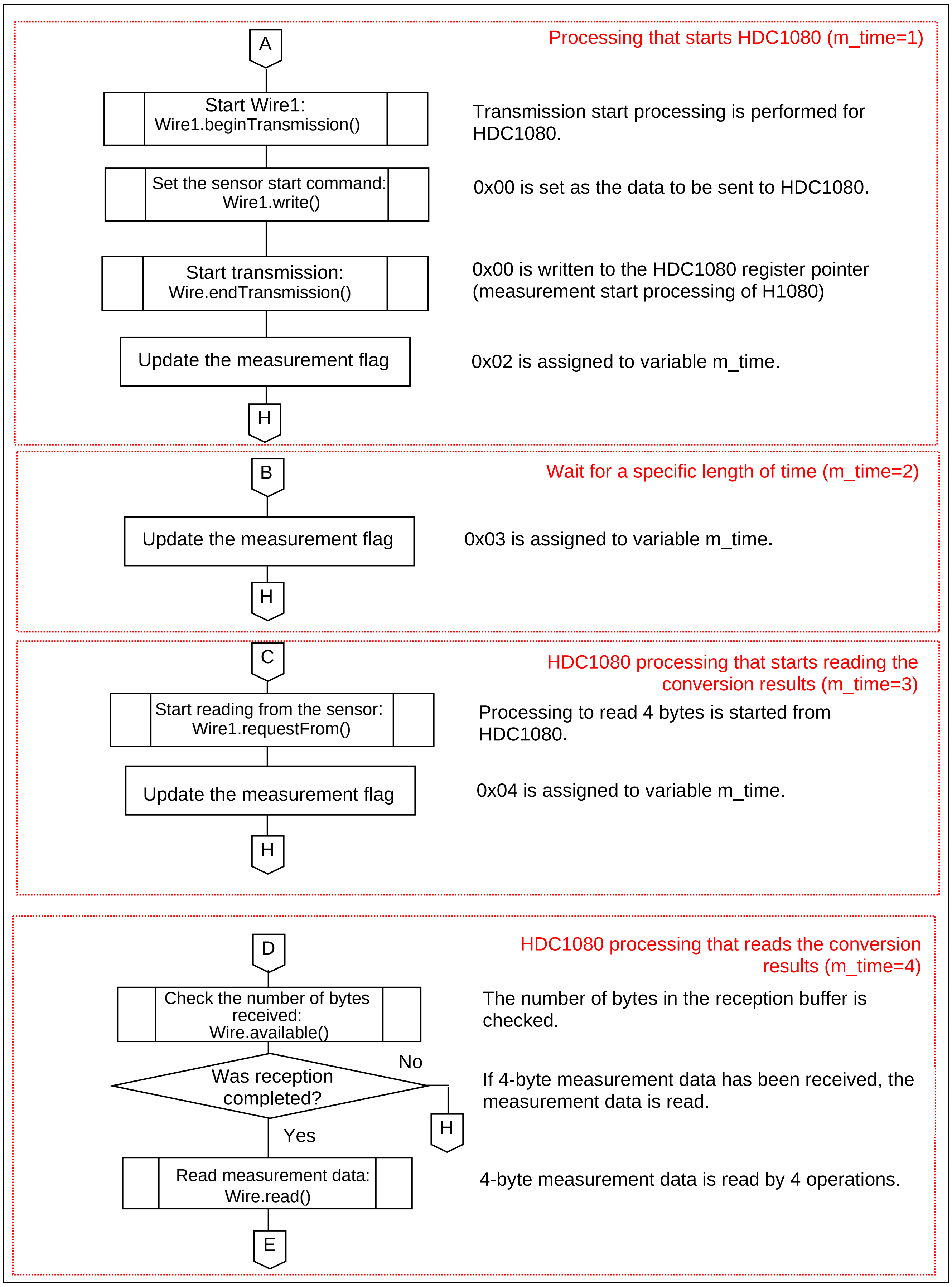

Processing that starts HDC1080 (m_time=1)
A
Start Wire1: Wire1.beginTransmission()
Transmission start processing is performed for HDC1080.
Set the sensor start command: Wire1.write()
0x00 is set as the data to be sent to HDC1080.
Start transmission: Wire.endTransmission()
0x00 is written to the HDC1080 register pointer (measurement start processing of H1080)
Update the measurement flag
0x02 is assigned to variable m_time.
H
Wait for a specific length of time (m_time=2)
B
Update the measurement flag
0x03 is assigned to variable m_time.
H
HDC1080 processing that starts reading the conversion results (m_time=3)
C
Start reading from the sensor: Wire1.requestFrom()
Processing to read 4 bytes is started from HDC1080.
Update the measurement flag
0x04 is assigned to variable m_time.
H
HDC1080 processing that reads the conversion results (m_time=4)
D
Check the number of bytes received: Wire.available()
The number of bytes in the reception buffer is checked.
Was reception completed?
No
H
If 4-byte measurement data has been received, the measurement data is read.
Yes
Read measurement data: Wire.read()
4-byte measurement data is read by 4 operations.
E

Figure 5.3 Main Function (2/4)

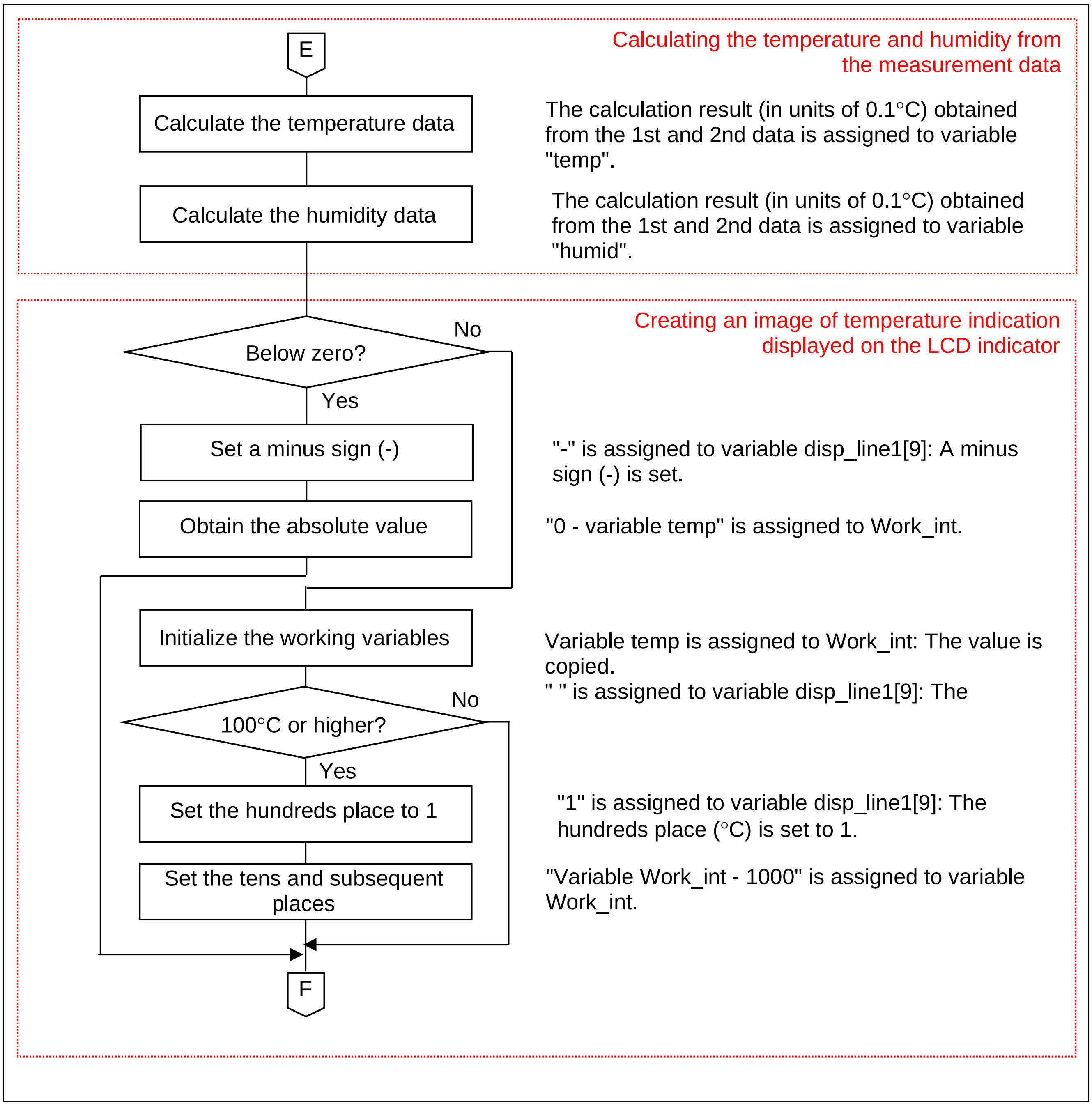

E
Calculate the temperature data
Calculate the humidity data
Calculating the temperature and humidity from the measurement data
The calculation result (in units of 0.1°C) obtained from the 1st and 2nd data is assigned to variable "temp".
The calculation result (in units of 0.1°C) obtained from the 1st and 2nd data is assigned to variable "humid".
Creating an image of temperature indication displayed on the LCD indicator
Below zero?
No
Yes
Set a minus sign (-)
"-" is assigned to variable disp_line1[9]: A minus sign (-) is set.
Obtain the absolute value
"0 - variable temp" is assigned to Work_int.
Initialize the working variables
Variable temp is assigned to Work_int: The value is copied.
" " is assigned to variable disp_line1[9]: The
100°C or higher?
No
Yes
Set the hundreds place to 1
"1" is assigned to variable disp_line1[9]: The hundreds place (°C) is set to 1.
Set the tens and subsequent places
"Variable Work_int - 1000" is assigned to variable Work_int.
F

Figure 5.4 Main Function (3/4)

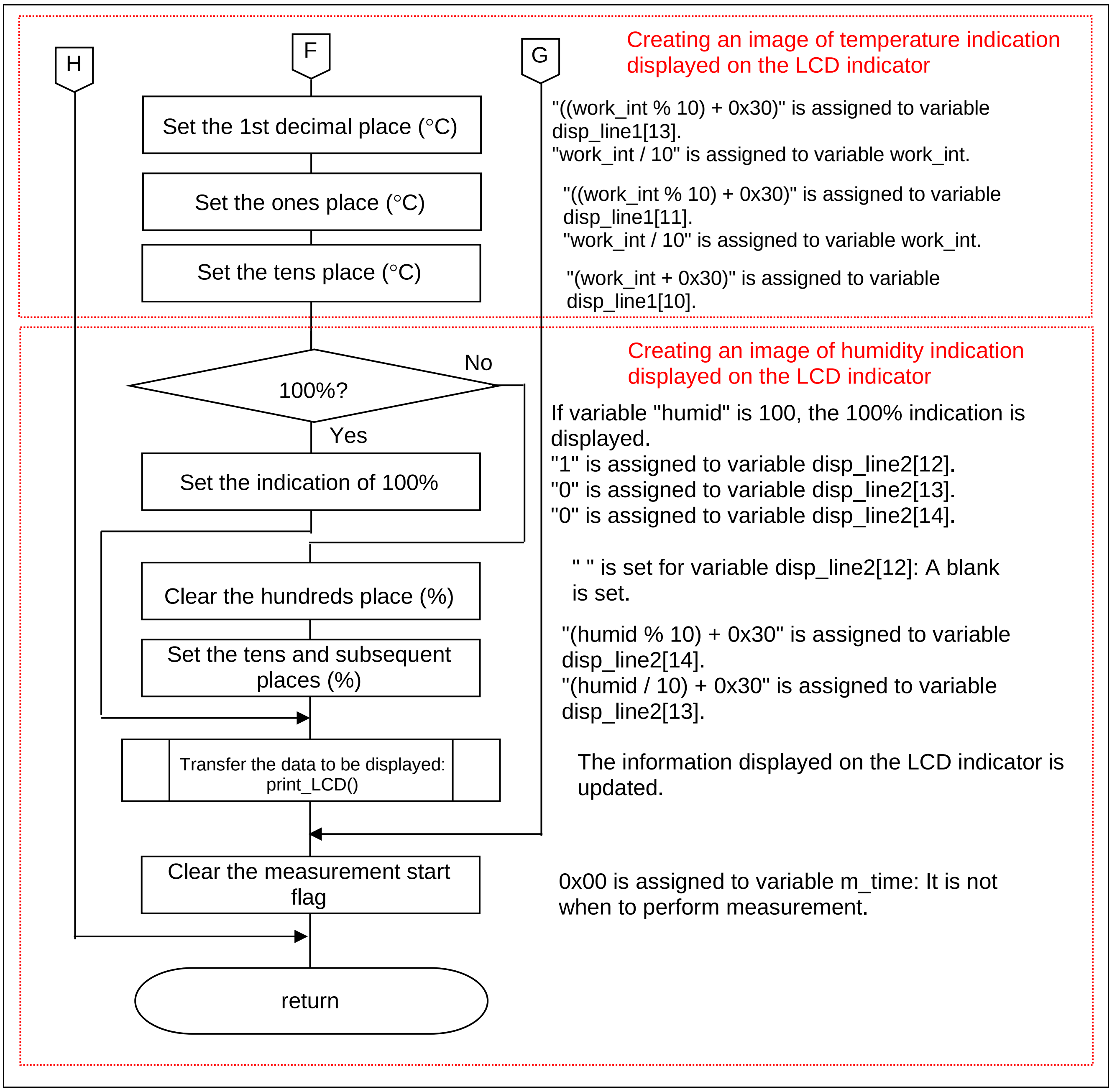

H
F
G
Set the 1st decimal place (°C)
Set the ones place (°C)
Set the tens place (°C)
Creating an image of temperature indication displayed on the LCD indicator
"((work_int % 10) + 0x30)" is assigned to variable disp_line1[13].
"work_int / 10" is assigned to variable work_int.
"((work_int % 10) + 0x30)" is assigned to variable disp_line1[11].
"work_int / 10" is assigned to variable work_int.
"(work_int + 0x30)" is assigned to variable disp_line1[10].
100%?
No
Yes
Set the indication of 100%
Clear the hundreds place (%)
Set the tens and subsequent places (%)
Transfer the data to be displayed: print_LCD()
Clear the measurement start flag
return
Creating an image of humidity indication displayed on the LCD indicator
If variable "humid" is 100, the 100% indication is displayed.
"1" is assigned to variable disp_line2[12].
"0" is assigned to variable disp_line2[13].
"0" is assigned to variable disp_line2[14].
" " is set for variable disp_line2[12]: A blank is set.
"(humid % 10) + 0x30" is assigned to variable disp_line2[14].
"(humid / 10) + 0x30" is assigned to variable disp_line2[13].
The information displayed on the LCD indicator is updated.
0x00 is assigned to variable m_time: It is not when to perform measurement.

Figure 5.5 Main Function (4/4)

5.6.3 LCD Indicator Initialization Function

Figure 5.6 shows a flowchart of the LCD indicator initialization function.

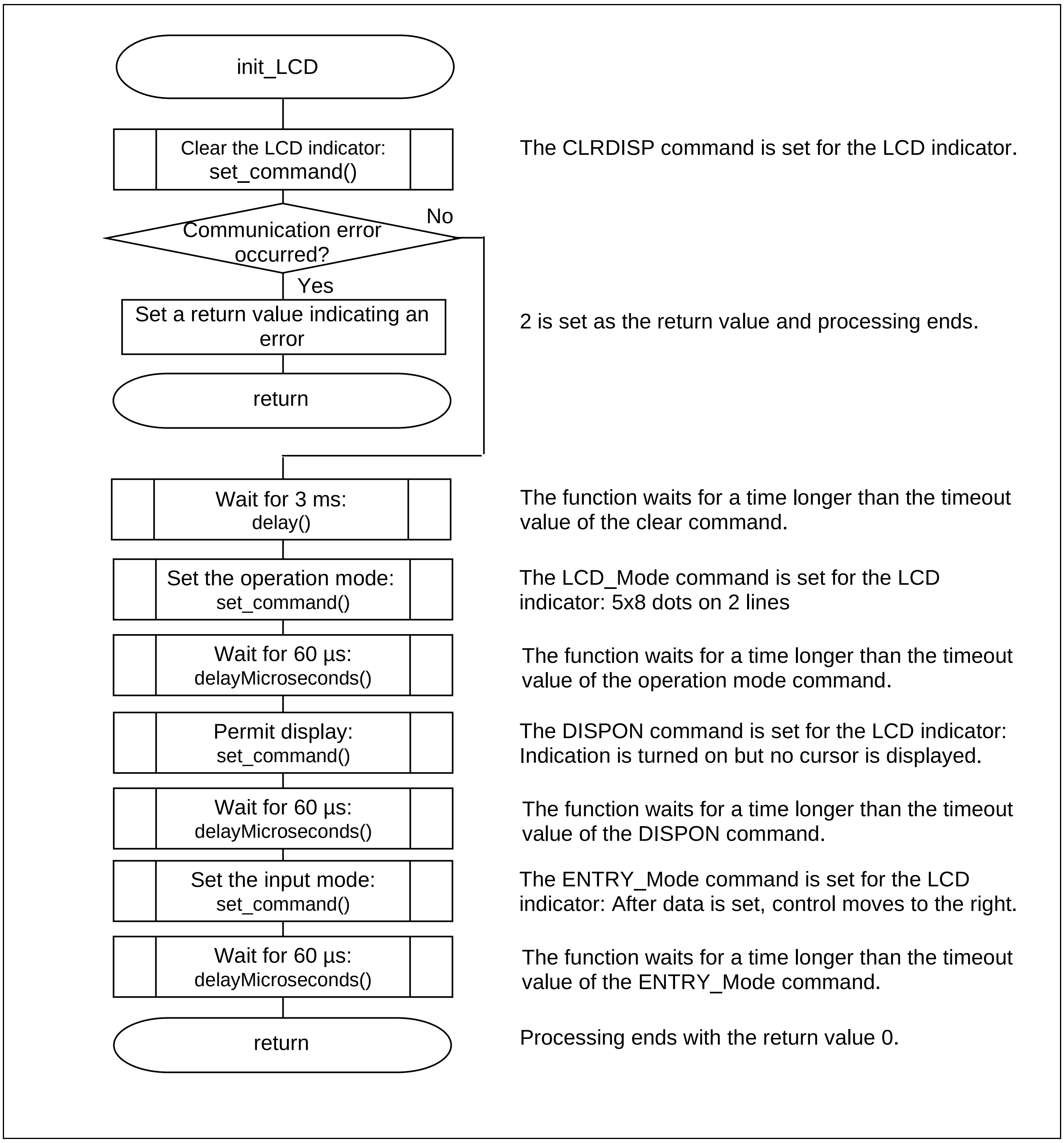

init_LCD
Clear the LCD indicator: set_command()
The CLRDISP command is set for the LCD indicator.
Communication error occurred?
No
Yes
Set a return value indicating an error
2 is set as the return value and processing ends.
return
Wait for 3 ms: delay()
The function waits for a time longer than the timeout value of the clear command.
Set the operation mode: set_command()
The LCD_Mode command is set for the LCD indicator: 5x8 dots on 2 lines
Wait for 60 µs: delayMicroseconds()
The function waits for a time longer than the timeout value of the operation mode command.
Permit display: set_command()
The DISPON command is set for the LCD indicator: Indication is turned on but no cursor is displayed.
Wait for 60 µs: delayMicroseconds()
The function waits for a time longer than the timeout value of the DISPON command.
Set the input mode: set_command()
The ENTRY_Mode command is set for the LCD indicator: After data is set, control moves to the right.
Wait for 60 µs: delayMicroseconds()
The function waits for a time longer than the timeout value of the ENTRY_Mode command.
return
Processing ends with the return value 0.

Figure 5.6 LCD Indicator Initialization Function

5.6.4 Function that Sets Full-Screen Display for the LCD Indicator

Figure 5.7 and Figure 5.8 show a flowchart of the function that sets full-screen display for the LCD indicator.

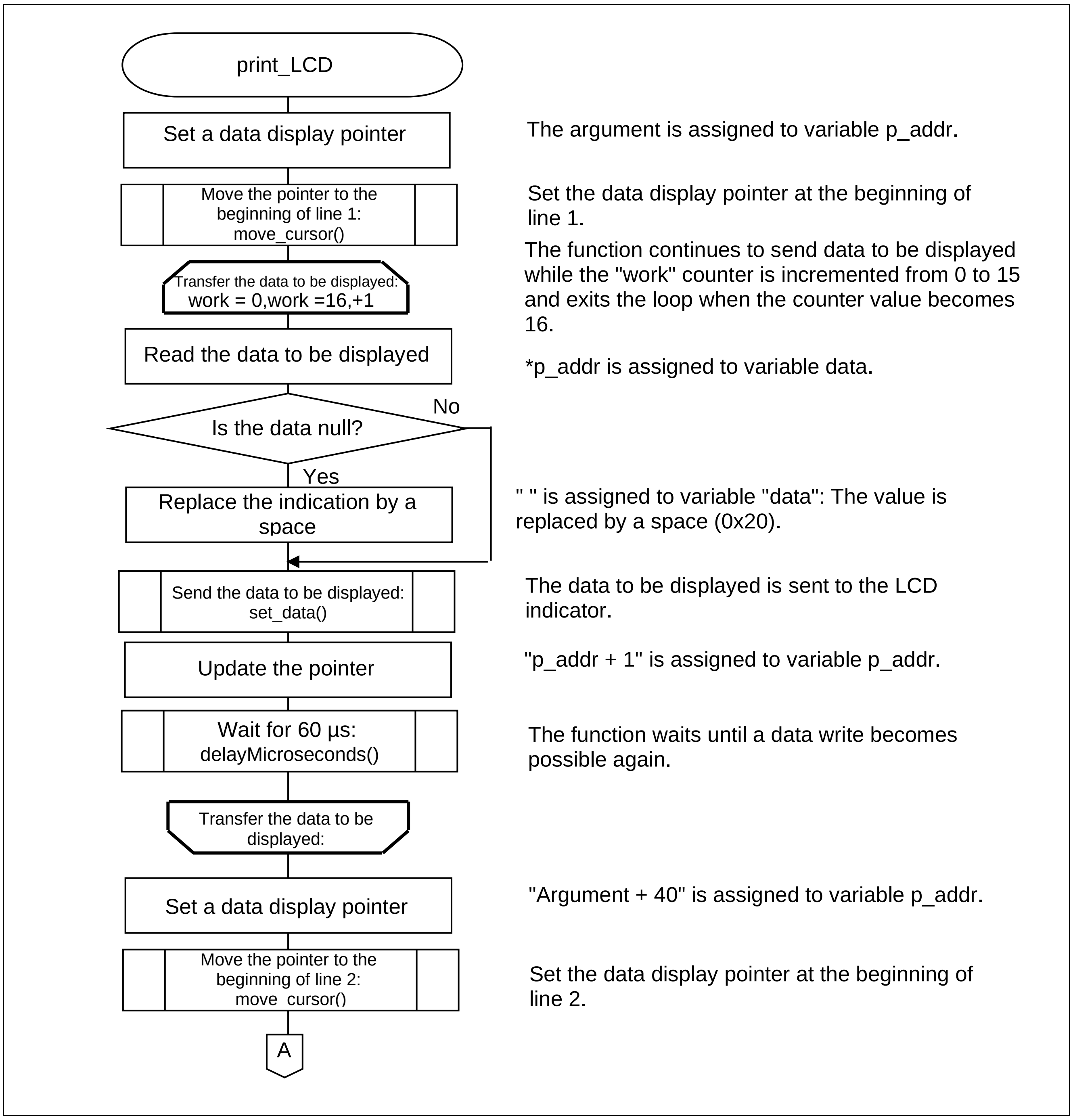

print_LCD
Set a data display pointer
The argument is assigned to variable p_addr.
Move the pointer to the beginning of line 1: move_cursor()
Set the data display pointer at the beginning of line 1.
Transfer the data to be displayed: work = 0,work =16,+1
The function continues to send data to be displayed while the "work" counter is incremented from 0 to 15 and exits the loop when the counter value becomes 16.
Read the data to be displayed
*p_addr is assigned to variable data.
Is the data null?
No
Yes
Replace the indication by a space
" " is assigned to variable "data": The value is replaced by a space (0x20).
Send the data to be displayed: set_data()
The data to be displayed is sent to the LCD indicator.
Update the pointer
"p_addr + 1" is assigned to variable p_addr.
Wait for 60 μs: delayMicroseconds()
The function waits until a data write becomes possible again.
Transfer the data to be displayed:
Set a data display pointer
"Argument + 40" is assigned to variable p_addr.
Move the pointer to the beginning of line 2: move_cursor()
Set the data display pointer at the beginning of line 2.
A

Figure 5.7 Function that Sets Full-Screen Display for the LCD Indicator (1/2)

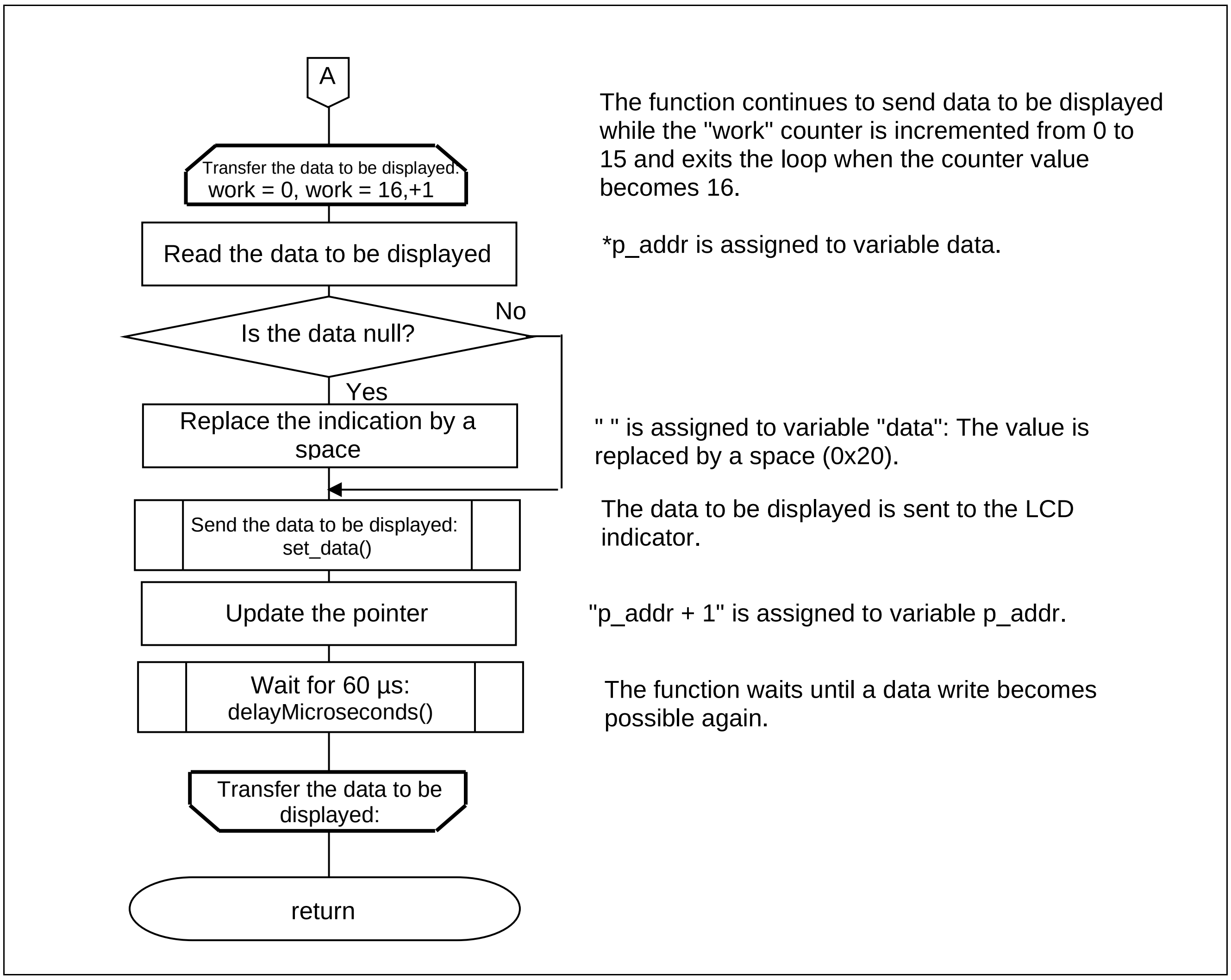

A
Transfer the data to be displayed:
work = 0, work = 16,+1
Read the data to be displayed
Is the data null?
No
Yes
Replace the indication by a space
Send the data to be displayed: set_data()
Update the pointer
Wait for 60 µs: delayMicroseconds()
Transfer the data to be displayed:
return
The function continues to send data to be displayed while the "work" counter is incremented from 0 to 15 and exits the loop when the counter value becomes 16.
*p_addr is assigned to variable data.
" " is assigned to variable "data": The value is replaced by a space (0x20).
The data to be displayed is sent to the LCD indicator.
"p_addr + 1" is assigned to variable p_addr.
The function waits until a data write becomes possible again.

Figure 5.8 Function that Sets Full-Screen Display for the LCD Indicator (2/2)

5.6.5 Function that Sets the Data Display Position for the LCD Indicator

Figure 5.9 shows a flowchart of the function that sets the data display position for the LCD indicator.

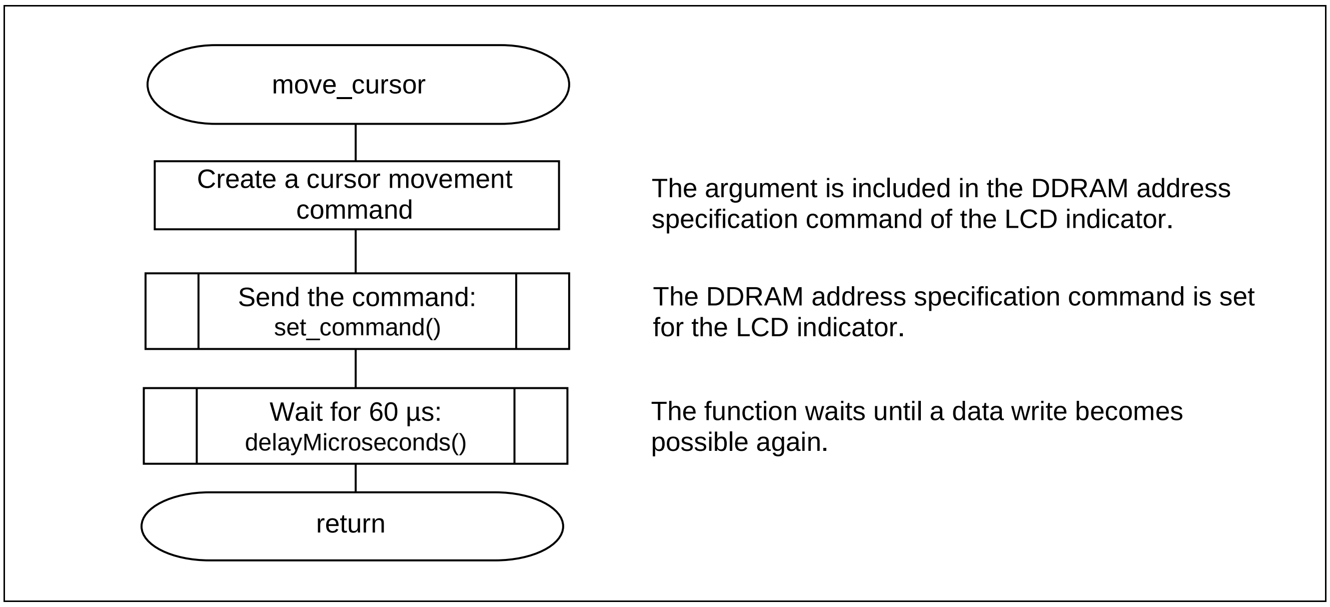

move_cursor
Create a cursor movement command
The argument is included in the DDRAM address specification command of the LCD indicator.
Send the command: set_command()
The DDRAM address specification command is set for the LCD indicator.
Wait for 60 µs: delayMicroseconds()
The function waits until a data write becomes possible again.
return

Figure 5.9 Function that Sets the Data Display Position for the LCD Indicator

5.6.6 Function that Sets a Command for the LCD Indicator

Figure 5.10 shows a flowchart of the function that sets a command for the LCD indicator.

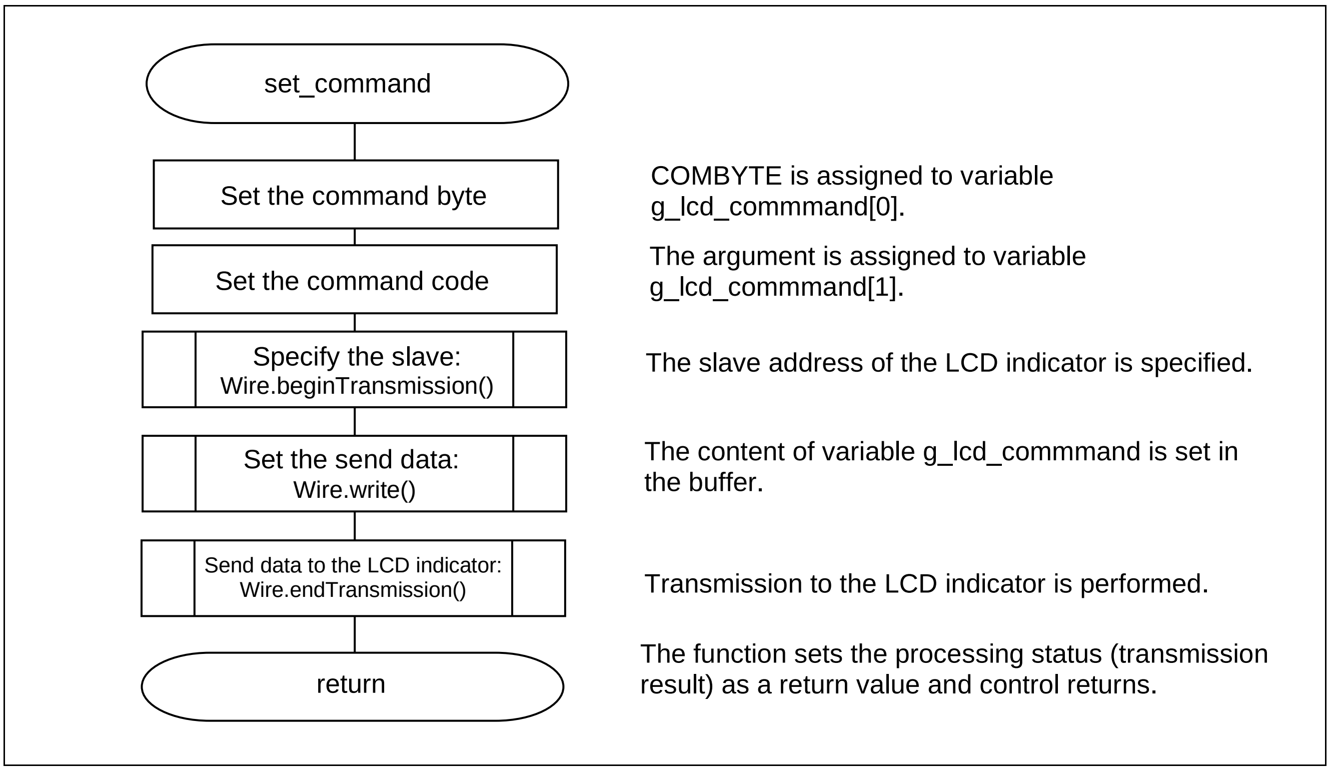

set_command
Set the command byte
COMBYTE is assigned to variable g_lcd_commmand[0].
Set the command code
The argument is assigned to variable g_lcd_commmand[1].
Specify the slave: Wire.beginTransmission()
The slave address of the LCD indicator is specified.
Set the send data: Wire.write()
The content of variable g_lcd_commmand is set in the buffer.
Send data to the LCD indicator: Wire.endTransmission()
Transmission to the LCD indicator is performed.
return
The function sets the processing status (transmission result) as a return value and control returns.

Figure 5.10 Function that Sets a Command for the LCD Indicator

5.6.7 Function that Sets Data for the LCD Indicator

Figure 5.11 shows a flowchart of the function that sets data for the LCD indicator.

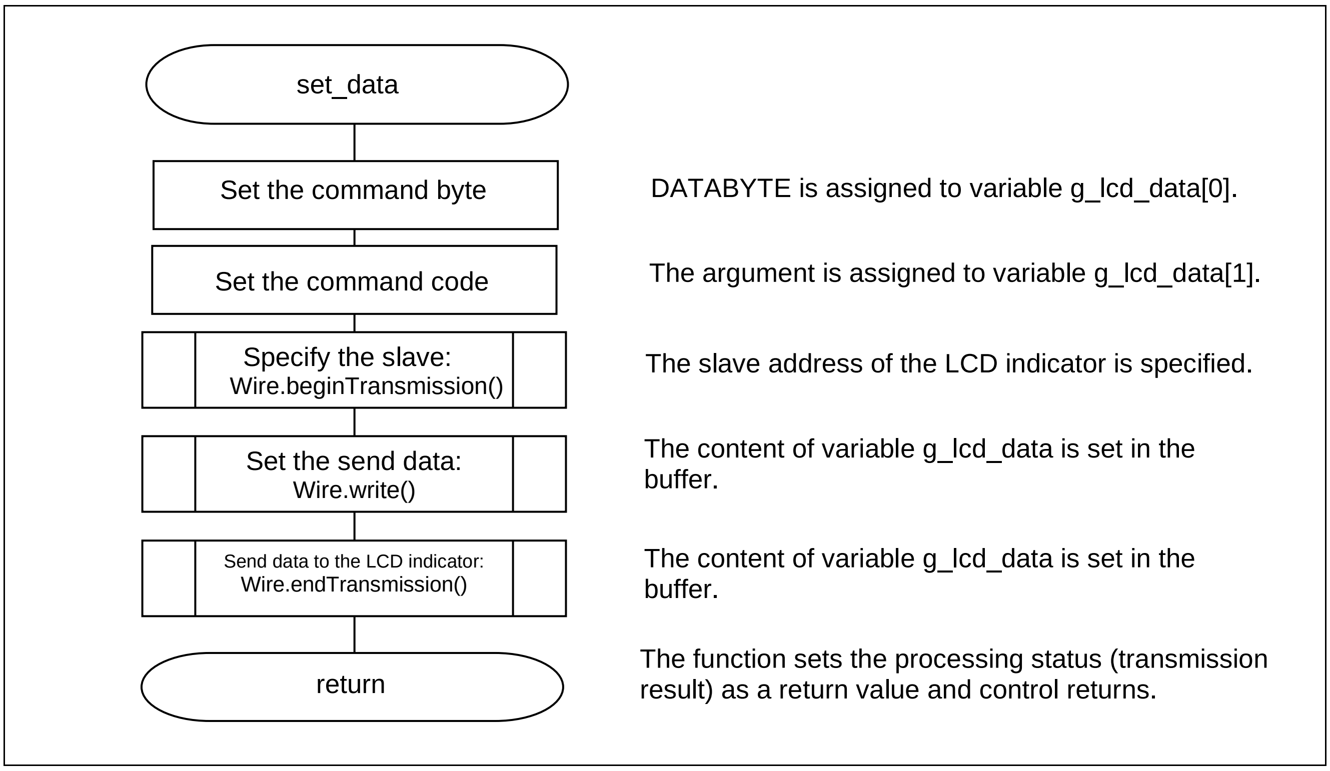

set_data
Set the command byte
DATABYTE is assigned to variable g_lcd_data[0].
Set the command code
The argument is assigned to variable g_lcd_data[1].
Specify the slave: Wire.beginTransmission()
The slave address of the LCD indicator is specified.
Set the send data: Wire.write()
The content of variable g_lcd_data is set in the buffer.
Send data to the LCD indicator: Wire.endTransmission()
The content of variable g_lcd_data is set in the buffer.
return
The function sets the processing status (transmission result) as a return value and control returns.

Figure 5.11 Function that Sets Data for the LCD Indicator

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/

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	2021.11.01	—	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.

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/.