RL78/G14

Controlling an I2C-bus (Arduino API)

Introduction

This application note explains how you can code a program in a language such as Arduino to control the HS3001 sensor and LCD indicator via the I2C bus by 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

In this application note, a program written in a language such as Arduino is used to control the temperature and humidity sensor connected to the Fast Prototyping Board (FPB) via the I²C bus in standard mode. The bus is set to operate at 80 kbps although a maximum of 100 kbps is available in standard mode. The measurement results are displayed on the LCD indicator that is also connected to the same I²C bus. The LCD indicator used is HD44780 or an equivalent device that supports I²C communication and provides two lines, each of which can display 16 characters.

When a switch is pressed or at one-minute intervals, the program controls the sensor via I²C communication to measure the temperature and humidity. The measurement results obtained are sent via I²C communication to the LCD indicator, which then displays the temperature and humidity.

Table 1.1 lists peripheral functions to be used.

Table 1.1 Peripheral functions used and their uses

Peripheral Function	**Use**
Digital input	Reading the state of the switch (SW_USR)
IICA0	Controlling the sensor and LCD indicator via the I²C bus
Timer array unit	Measuring 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 1.1.

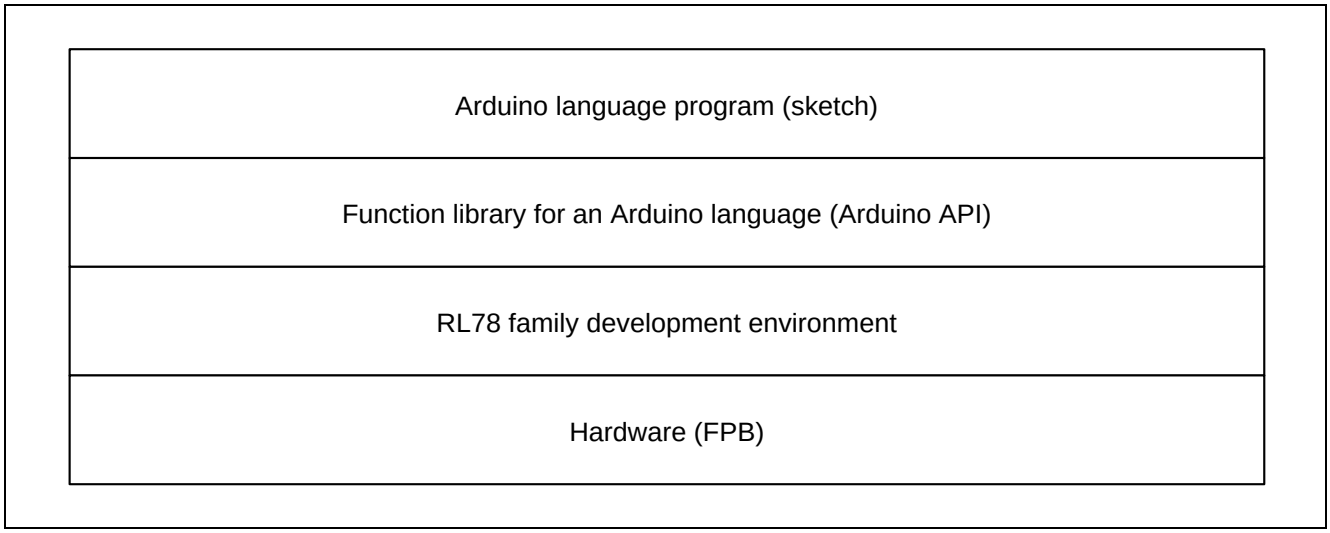


Figure 1.1 Program execution environment

Library functions that can be used in this application note are shown in Table 1.2 and Table 1.3.

Table 1.2 Library functions (1/2)

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.

Table 1.3 Library functions (2/2)

Item	Library Function	Function
I2C control	Wire.begin()	Initializes IICA0 and connects to the I²C 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.

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.

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 1.2.

1)

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

2)

```
#define SLADDR_HS3001    ( 0x44 )      // I2C bus slave address of HS3001
#define MINUTE           ( 60000/16 )  // 1 minute devided by 16milli sec
unsigned int old_time = 0x0000;        // previous time(milli sec.)
unsigned char hs3001_buff[4] =         // HS3001 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 coun
char sw_work = 0xFF;                   // work for

extern API_Wire Wire;          // wire API
```

Figure 1.2 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 "hs3001_buff" 4-byte array for communication use to control the HS3001 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 "Wire" API_Wire-type structure is used to reference any objects that are defined by AR_LIB_WIRE.c that provides Wire-related API functions.

1.5 Initial Setting Processing

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

The "setup" function specifies that the switch input pin be used for input. Also, IICA0 is 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

    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 1.3 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 startup screen is as in Figure 1.4 .

```
void loop(void){
  // put your main code here, to run repeatedly:

    int time_work;                      // present time buffer
    char com_sel;                       // common select signal
    char seg_data;                      // segment data
    char sw_work;                       // work for switch check

/*-------------------------------------
    wait for 4milli seconds interval.
-------------------------------------*/

    time_work = ( int )( millis() & 0xFFFC ); // read milli sec data

    if ( old_time != time_work )        // check timing of change LED
    {
```

Figure 1.4 Leading section of main processing

1.7 Data Processing Performed by HS3001

Normally, HS3001 is placed in sleep mode. To obtain the humidity data and temperature data, a measurement request (MR) must be issued. The MR is issued by writing information (slave address = 0x88 and the number of bytes = 0) to HS3001.

To obtain 14-bit humidity/temperature data from HS3001, measurement takes 33.9 ms to complete.

In this application note, after issuing an MR, the software waits for 16 ms three consecutive times and then reads the measurement results when measurement is completed.

Figure 1.5 shows an example of 4-byte data read from HS3001. In the example, humidity data is indicated in red and temperature data is indicated in blue.

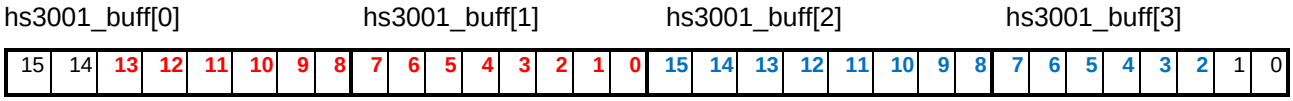


Figure 1.5 HS3001 Data Format

Humidity is obtained by using the following expressions. Expression ① is used to convert data to 14-bit length data. Expression ② is used to multiply the value by 100. Then, expression ③ is used to obtain the humidity data. The value is divided by (2^14 – 1). The humidity data in percentage is obtained in this way.

① long_work = (hs3001_buff[0] * 0x100UL + hs3001_buff[1]);

② long_work *= 100UL; // get percentage

③ humid = (unsigned char)(long_work / 0x3FFF); // get humidity

Similarly, temperature is obtained by using the following expressions. Expression ① is used to convert data to 14-bit length data. Expression ② is used to convert the resulting data to a value in 0.1°C degrees. Then, expression 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 = (hs3001_buff[2] * 0x40UL + (hs3001_buff[3] >>2));

② long_work *= 1650;

③ temp = (int)(long_work / 0x3FFF -400); // adjust offset(40 degree C)

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 e² 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 (FPB) that is used in this application note.

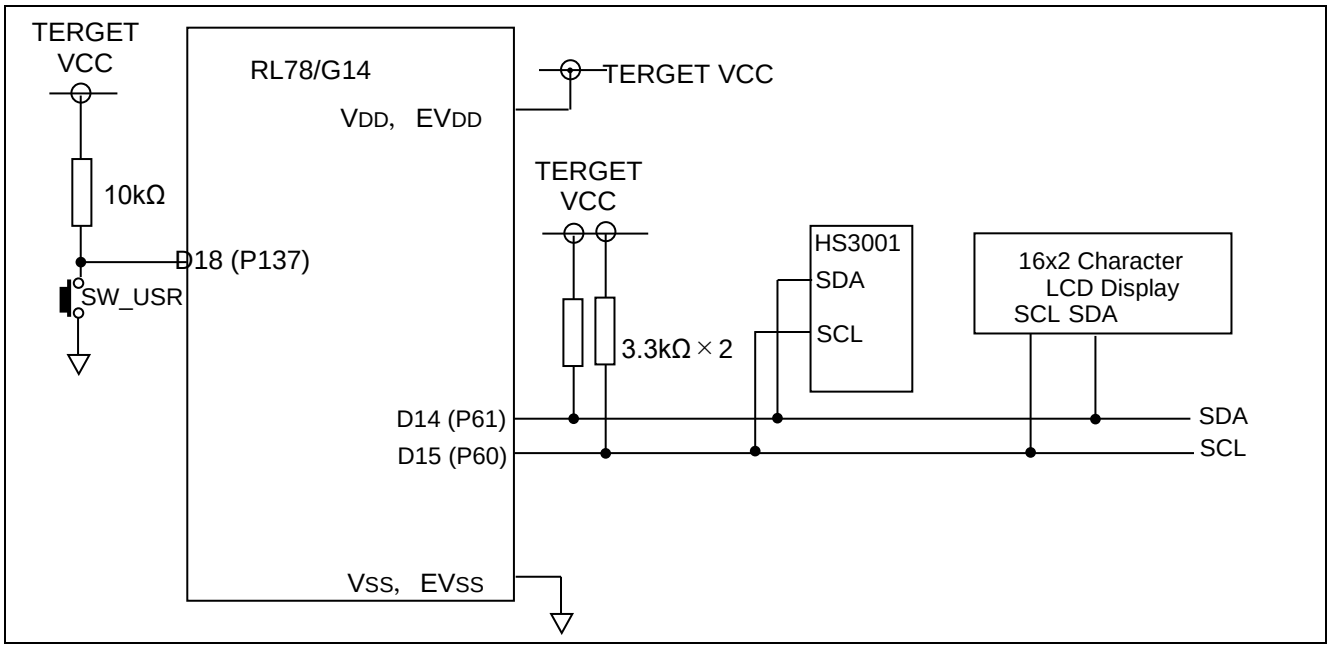


Figure 4.1 Hardware configuration example

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)
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. The software starts H3001 via the I^2C bus to measure the humidity and temperature. The software calculates the humidity and temperature from the obtained data, and then displays the results on the LCD indicator via the I^2C bus.

The software refreshes the displayed humidity and temperature when the switch is pressed or at 1-minute intervals.

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

(1) The "setup" function is used to set up the pins to be used.

- Sets the swPin pin that is used to read the on-board SW_USR switch for digital input.
- Sets IICA0 as the master so that the I^2C bus can be controlled by using the D14 and D15 pins.
- Initializes the LCD indicator that is connected to the I^2C bus to place the indicator in the initial display status.

(2) The "loop" function is used to perform the main processing.

- Obtains the settings of bits 15 to 4. These 12 bits indicate the number of milliseconds from the startup (in units of 16 milliseconds).
- Checks whether the obtained data has been changed from the old data (old_time).
- If the obtained data has not been changed, the software ends processing and control returns to the beginning of the "loop" function.
- If the obtained data has been changed (16 ms has elapsed), the software replaces old_time by the obtained data.
- Increments the 1-minute (0xEA6) counter for sensor startup.
- Checks the status of the switch connected to the D18 pin.
- If neither the switch is pressed nor 1 minute has elapsed, control returns to the beginning of the "loop" function.[Note]
- If the switch is pressed or 0 minute has elapsed, the software measures the humidity and temperature.[Note]
  - The software releases the sensor from the standby status to measure the humidity and temperature.
  - The software waits for about 48 ms so that it can obtain stable data.
  - The software reads data from the sensor.
  - The software calculates the humidity and temperature from the obtained data.
- The software transfers the obtained calculation results to the LCD indicator.
- Control returns to the beginning of the "loop" function.

Note: When the software has just started, it uses the sensor to measure the humidity and temperature.

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**	Number of the pin that is used to read SW_USR
DUMMY_DATA	**0xFF**	Data to be written to start reception during master reception
RELEASE	**1**	Specifies that the stop condition is issued when communication is completed.
RESTART	**0**	Specifies that the restart condition is issued when communication is completed.
SLADDR_HS3001	**0x44**	I^2C bus address of the sensor (7 bits)
SLADDR_LCD	**0x50**	I^2C bus address of the LCD indicator (7 bits)
COMBYTE	**0x00**	Data specifying that a command is transferred to the LCD indicator
DATABYTE	**0x80**	Data specifying that data is transferred to the LCD indicator
CLRDISP	**0x01**	Command that clears the display of the LCD indicator
HOMEPOSI	**0x02**	Moves the cursor of the LCD indicator to the home position.
LCD_Mode	**0b00111000**	Specifies that the following display format is used: 2 lines on which one character is represented using 5×8 dots.
DISPON	**0b00001111**	Blinks the cursor and turns display on.
ENTRY_Mode	**0b00000110**	Moves the display position each time 1 character is transferred.
LOOPLIMIT	**1000**	Sets the maximum number of start and stop operations that can be detected to 1,000.
SUCCESS	**0x00**	The processing with the I^2C bus has terminated normally.
BUS_FREE	**0x00**	The I^2C bus is not in use.
BUS_ERROR	**0x8F**	An attempt to secure the I^2C bus failed.
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 send bytes exceeded the buffer capacity.
NO_SLAVE	**0x82**	No applicable slave exists.
NO_ACK	**0x83**	NACK is replied to the transmitted data.
NO_DATA	**0x84**	The number of received bytes is 0.
MINUTE	**60000/16**	Obtains 1 minute by counting 16-millisecond intervals.

5.3 List of Variables

Table 5.2 lists global variables.

Table 5.2 Global variables

Type	Variable Name	Description	Function used Note
unsigned int	old_time	Time elapsed from the previous startup (in milliseconds)	loop()
unsigned char	hs3001_buff[4]	Buffer for 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 to be displayed on line 1 of the LCD indicator	loop()
char	disp_line2[40]	Data to be displayed on line 2 of the LCD indicator	loop()
Int	count16ms	Obtains 1 minute by counting 16-millisecond intervals.	loop()
char	sw_work	Variable used to check the switch status at 16-millisecond intervals	loop()
unsigned char	g_lcd_command[2]	Variable used to set a command for the LCD indicator	set_command()
unsigned char	g_lcd_data[2]	Variable used to set data for the LCD indicator	set_dat()
uint8_t	gp_tx_set	Pointer for writing data to the transmission buffer (255 or less)	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 receive buffer (255 or less)	Wire_begin()、Wire_requestFrom()、r_IICA0_interrupt()
uint8_t	gp_rx_get	Pointer for reading data from the receive buffer	Wire_begin()、Wire_requestFrom()、Wire_read()
uint8_t	g_rx_buff[256]	Receive buffer	r_IICA0_interrupt()、Wire_read()
uint16_t	g_rx_num	Number of received bytes	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 issue the stop condition at termination: 0: Issues the restart condition at termination. 1: Issues the stop condition 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 transmitted data. 0x04: Other errors	Wire_endTransmission(),

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

5.4 List of Functions

Table 5.3 shows a list of functions.

Table 5.3 List of functions

Function Name	Overview
loop	Main processing (sketch)
setup	Initialization function (sketch)
pinMode	Specifies the operation mode of a pin (input mode, output mode, or input mode with the internal pull-up resistance enabled).
digitalWrite	Reads the status of a pin.
digitalRead	Outputs data to a pin.
micros	Returns the number of microseconds from when the program execution started to the current time.
millis	Returns the number of milliseconds from when the program execution started to the current 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^2C library and connects it as the master.
Wire.requestFrom	Starts reading data from the specified slave. The reading is processed as an interrupt.
Wire_requestFromS	Starts reading data from the specified slave. The reading is processed as an interrupt. Issuance of the stop condition at the time when reception is completed can be specified. This function is used for internal processing of Wire.requestFrom.
Wire_requestFromsub	This function is used for internal processing of Wire.requestFrom.
Wire.available	Returns the number of bytes that can be read by Wire.read from the receive buffer.
Wire.read	Reads data from the receive buffer.
Wire.beginTransmission	Prepares for sending data to the specified slave.
Wire.write	Writes (to the send buffer) the data to be sent.
Wire_writec	Adds 1-character data to the send buffer. This function is used for the internal processing of Wire.write.
Wire_writeb	Adds a data block to the send buffer. This function is used for the internal processing of Wire.write.
Wire.endTransmission	Sends the send data from the buffer actually via the I^2C bus. Issuance of the stop condition at the time when sending is completed can be specified.
init_LCD	Initializes the LCD indicator.
print_LCD	Displays text (16 characters x 2 lines) on the LCD indicator.
move_cursor	Specifies the cursor position at which the data to be displayed on the ICD indicator is set.
set_2digit	Displays 1-byte data with 2 digits.
set_1digit	Displays the low-order bit data with 1 digit.
set_command	Sends a command to the LCD indicator.
set_data	Sends the display data to the LCD indicator.

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, and 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 (HS3001). After the sensor starts, when about 48 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, and 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] 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]	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]	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, and 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, and 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 I²C bus
Header	AR_LIB_TIME.h, r_cg_macrodriver.h, AR_LIB_WIRE.h, and r_cg_userdefine.h
Declaration	void Wire.begin(void);
Description	This function initializes IICA0 as a preparation for using the I²C 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, and 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 I²C 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, and 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 I²C 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, and 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 I²C 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, and 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, and 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, and 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 I²C 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, and 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 I²C 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, and 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 I^2C bus.	

[Function Name]	Wire.endTransmission	
Overview	Function that sends data to be 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, and 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, and 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.	

5.6 Flowcharts

5.6.1 Initial setting function

Figure 5.1 shows the flowchart of the initial setting.

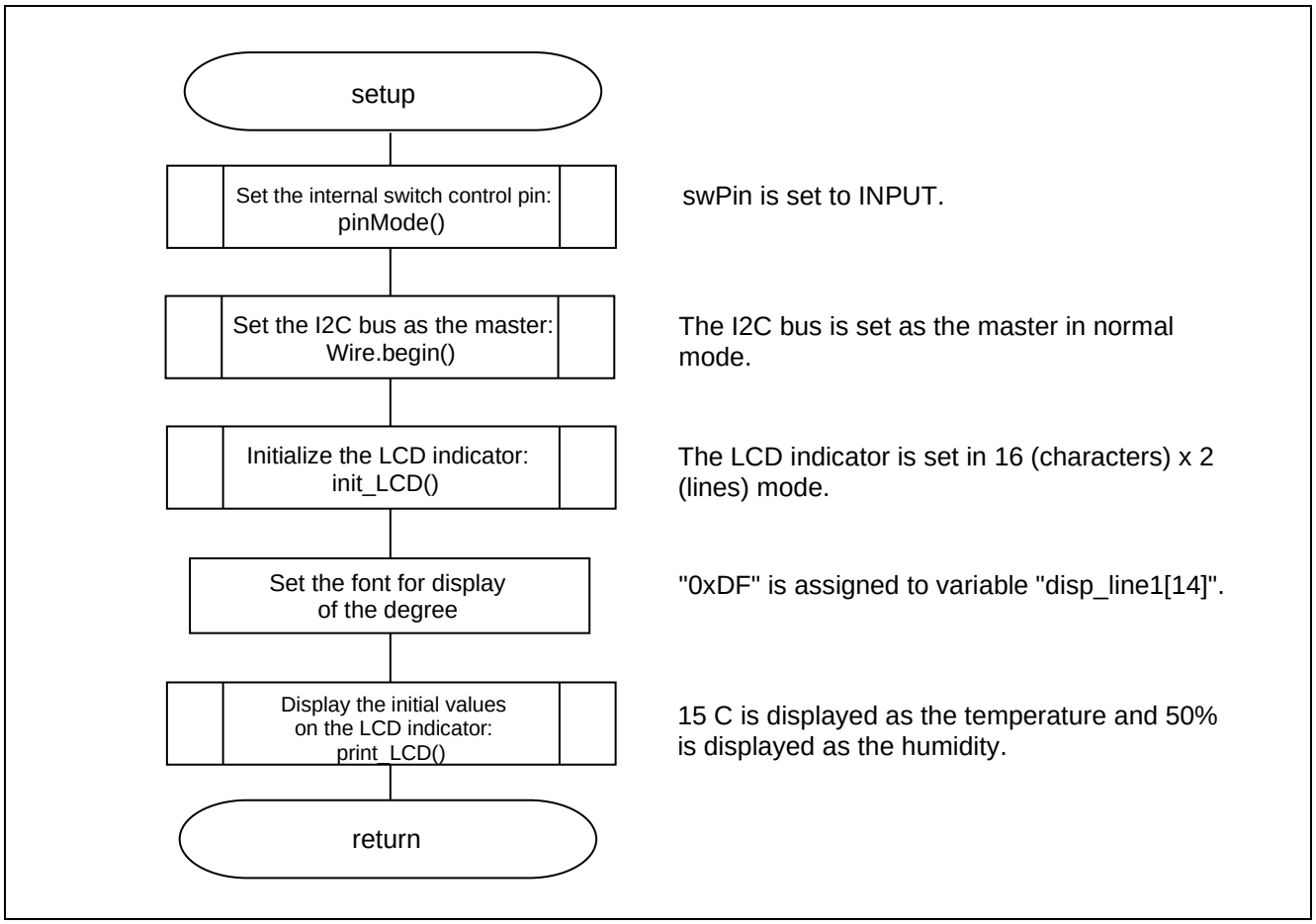


Figure 5.1 Initial setting function

5.6.2 Main Processing Function

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

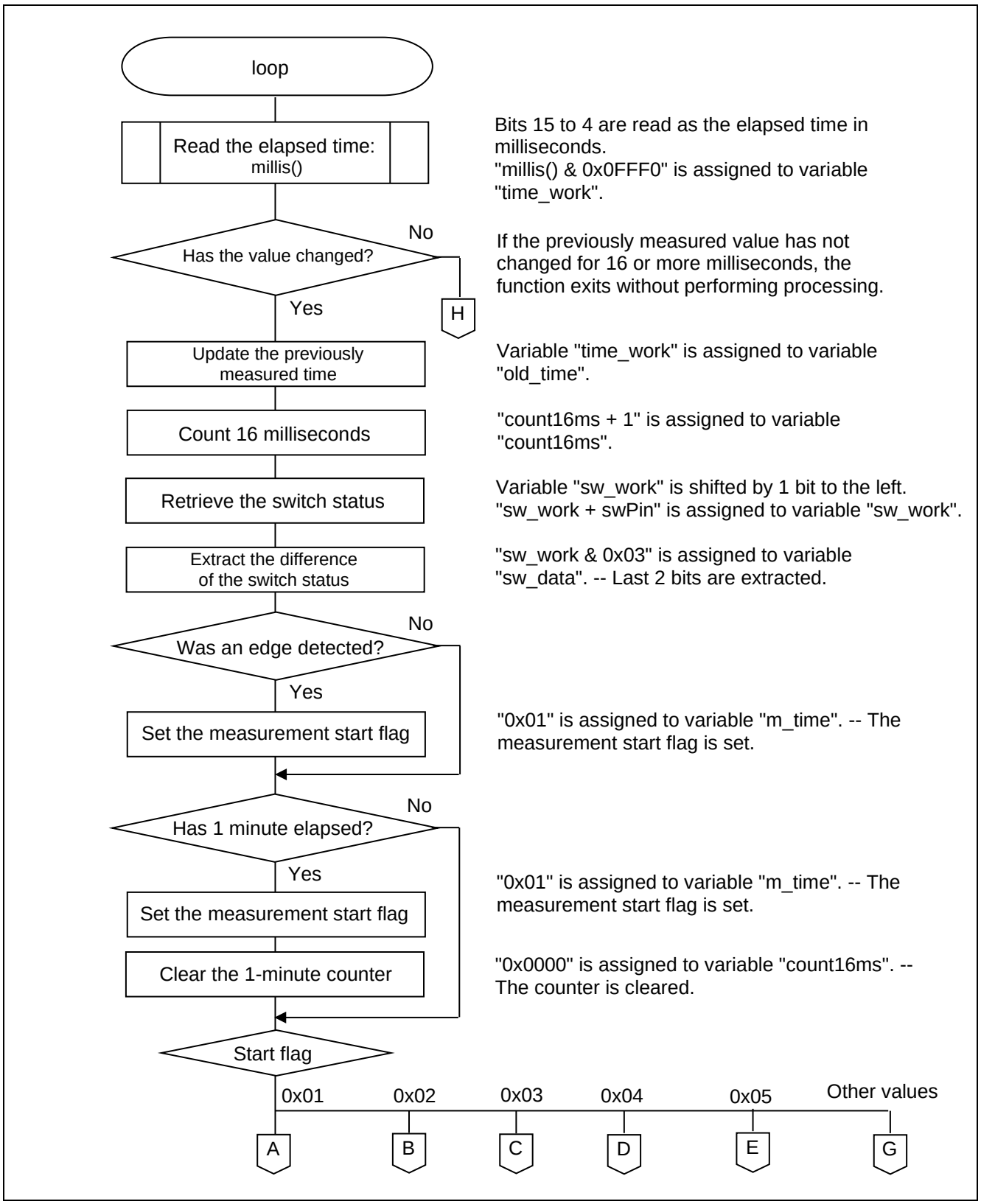


Figure 5.2 Main Function (1/4)

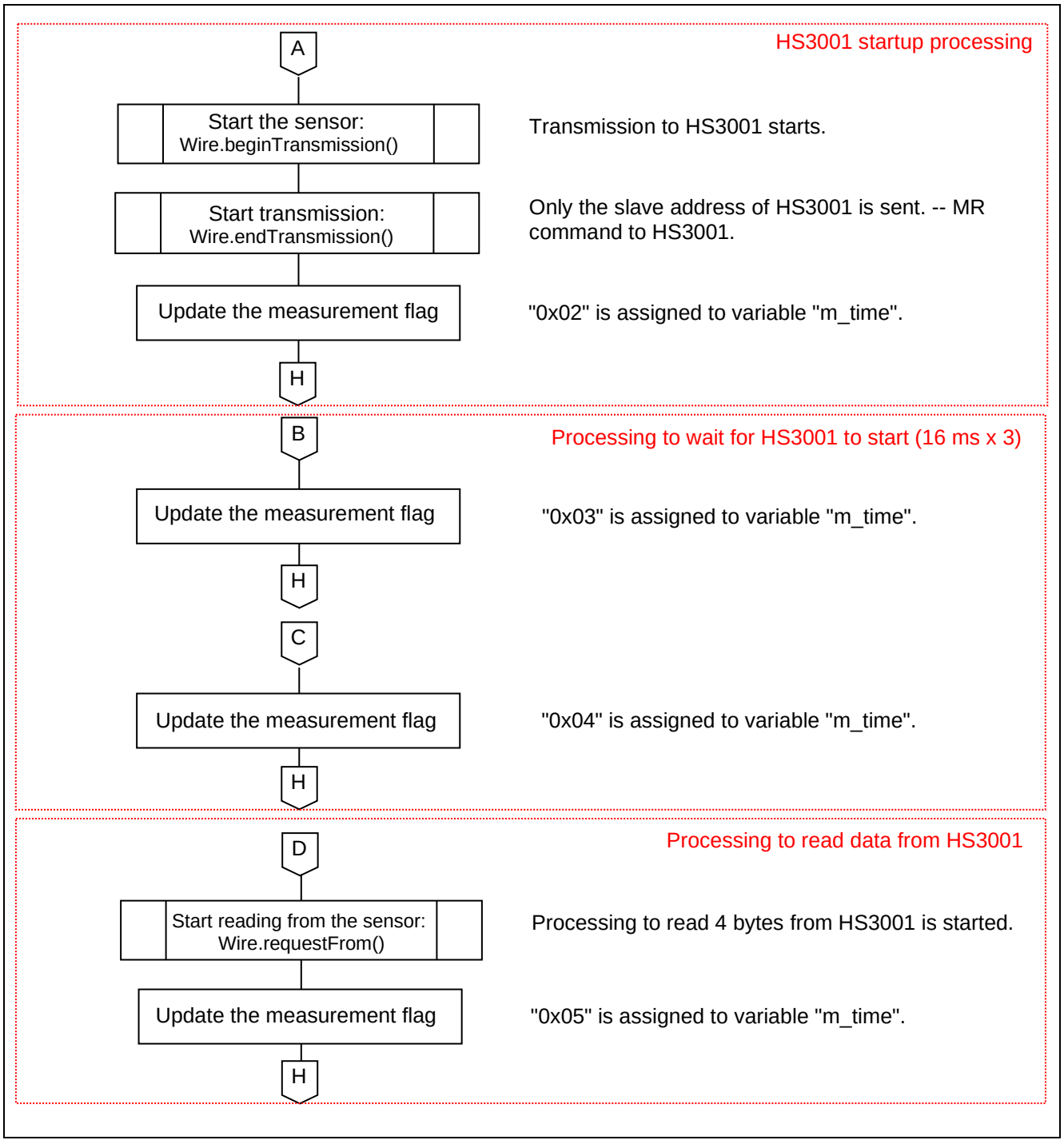


Figure 5.3 Main Function (2/4)

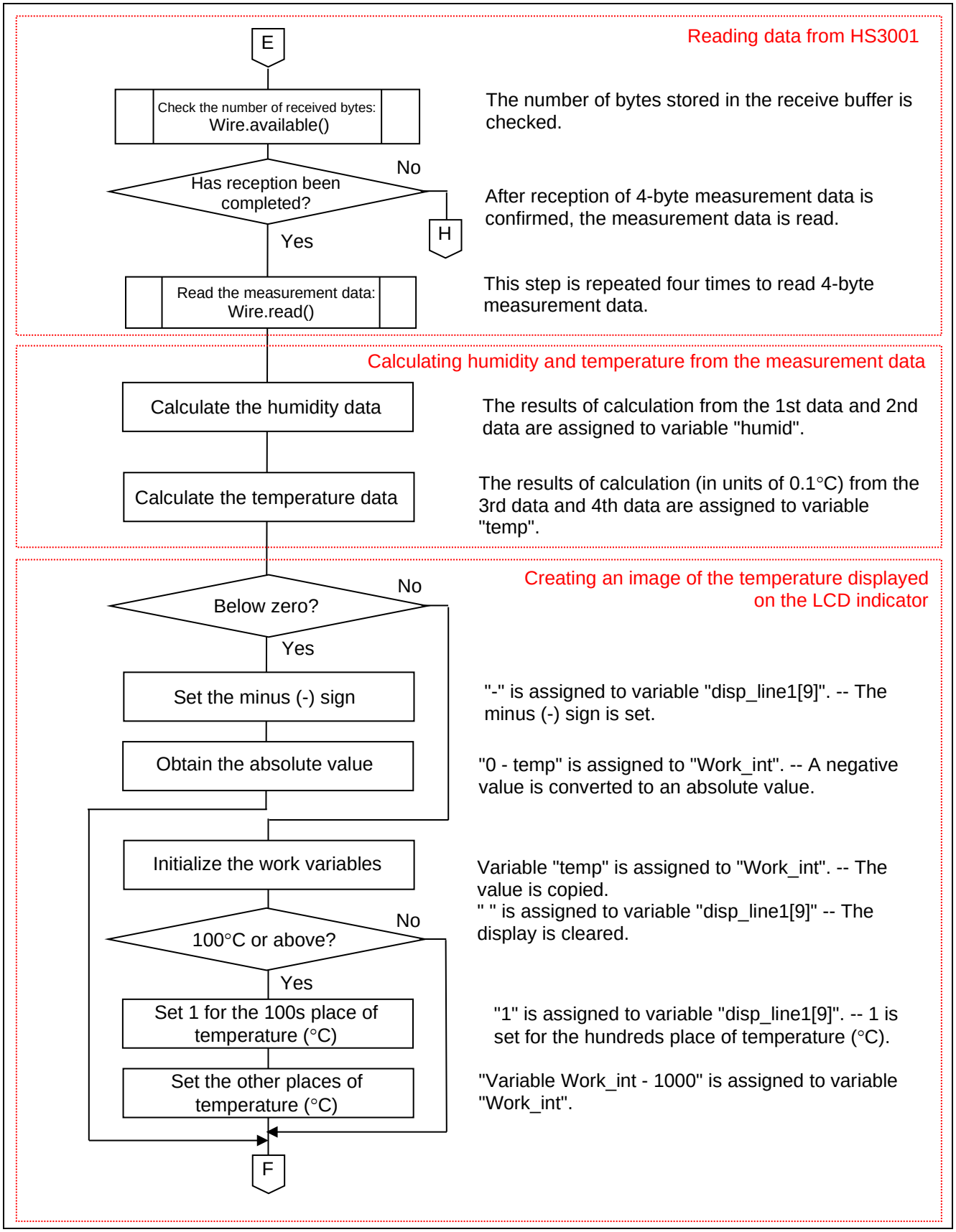


Figure 5.4 Main Function (3/4)

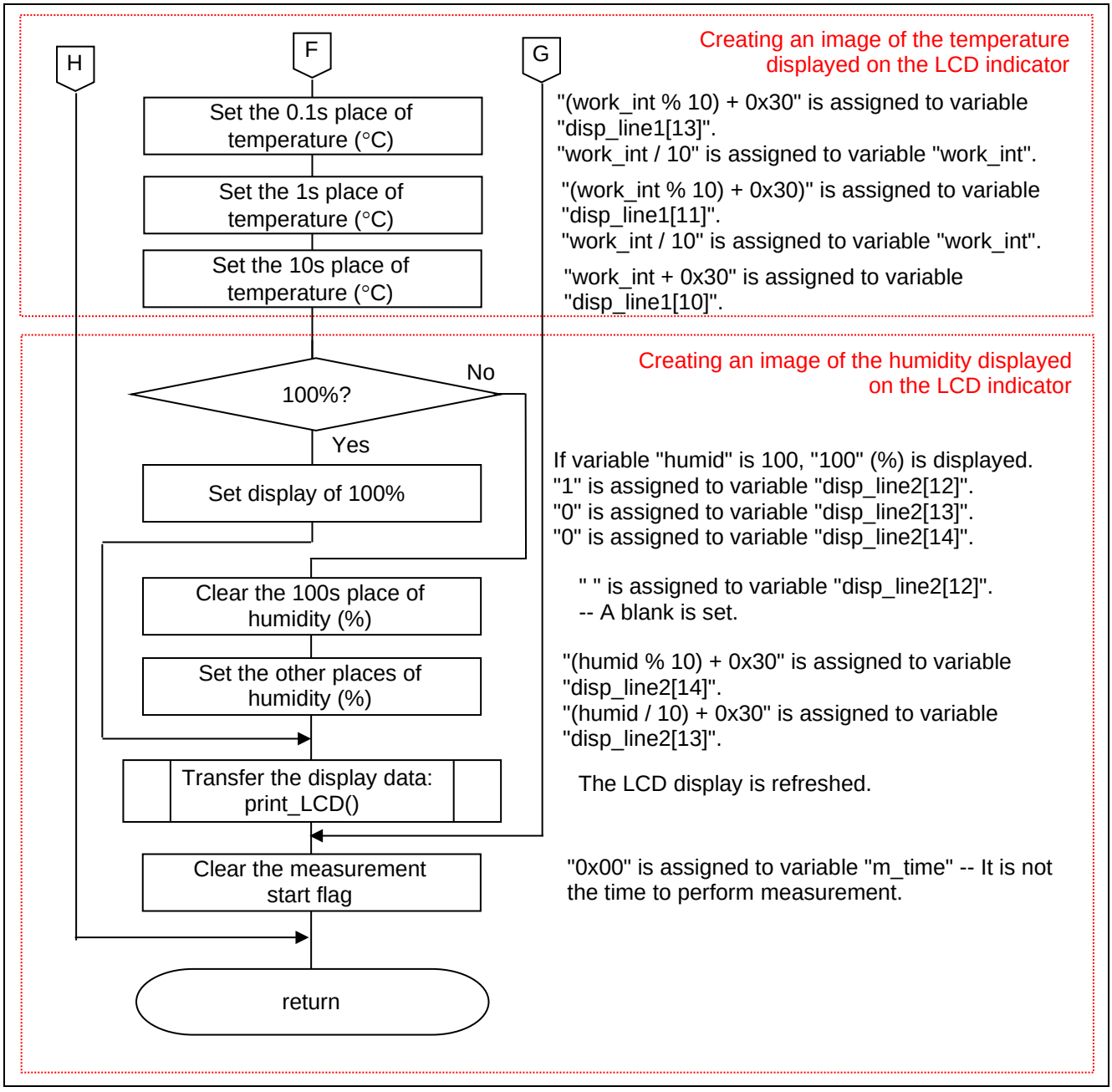


Figure 5.5 Main Function (4/4)

5.6.3 LCD Indicator Initialization Function

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

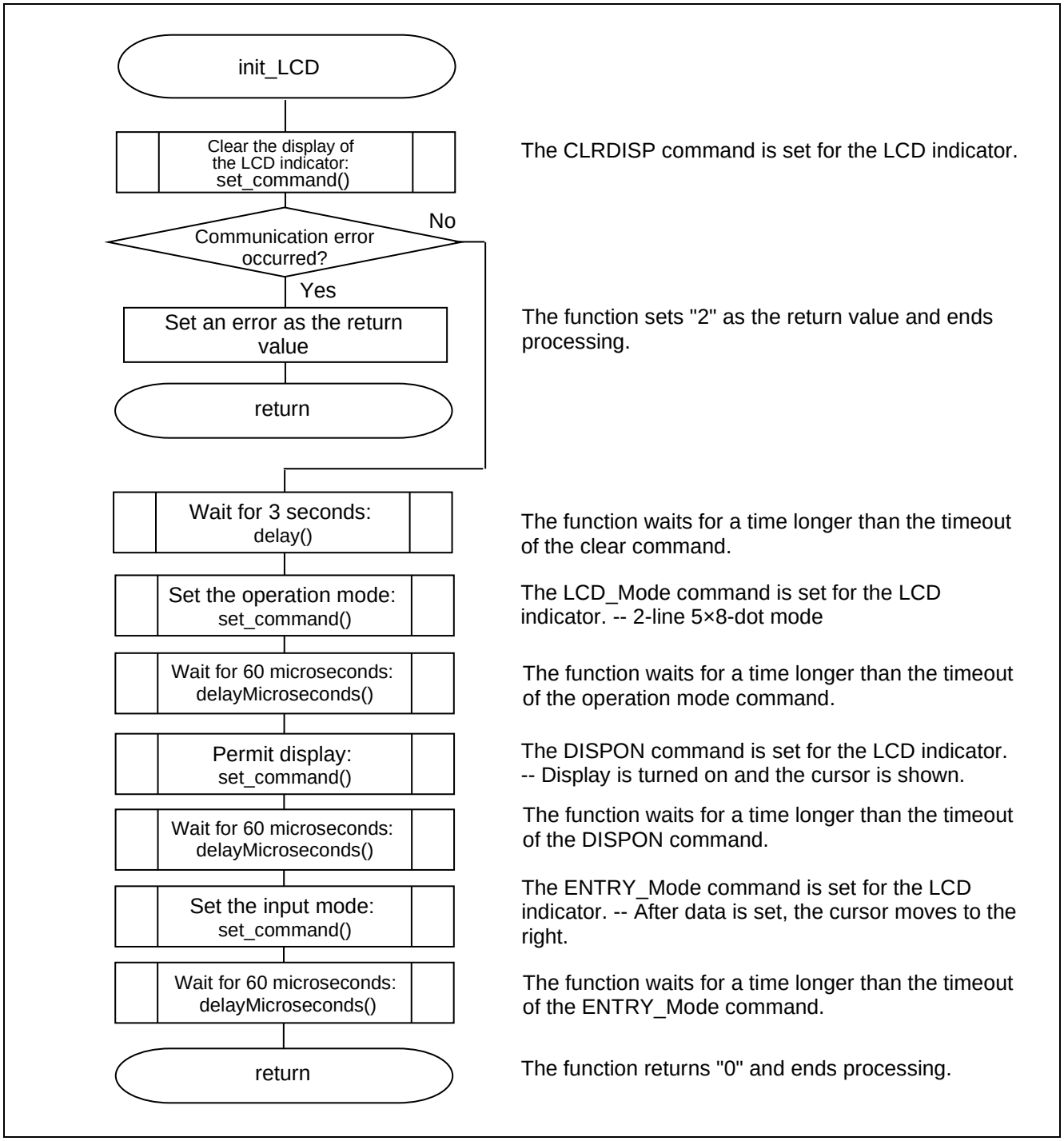


Figure 5.6 LCD Indicator Initialization Function

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.

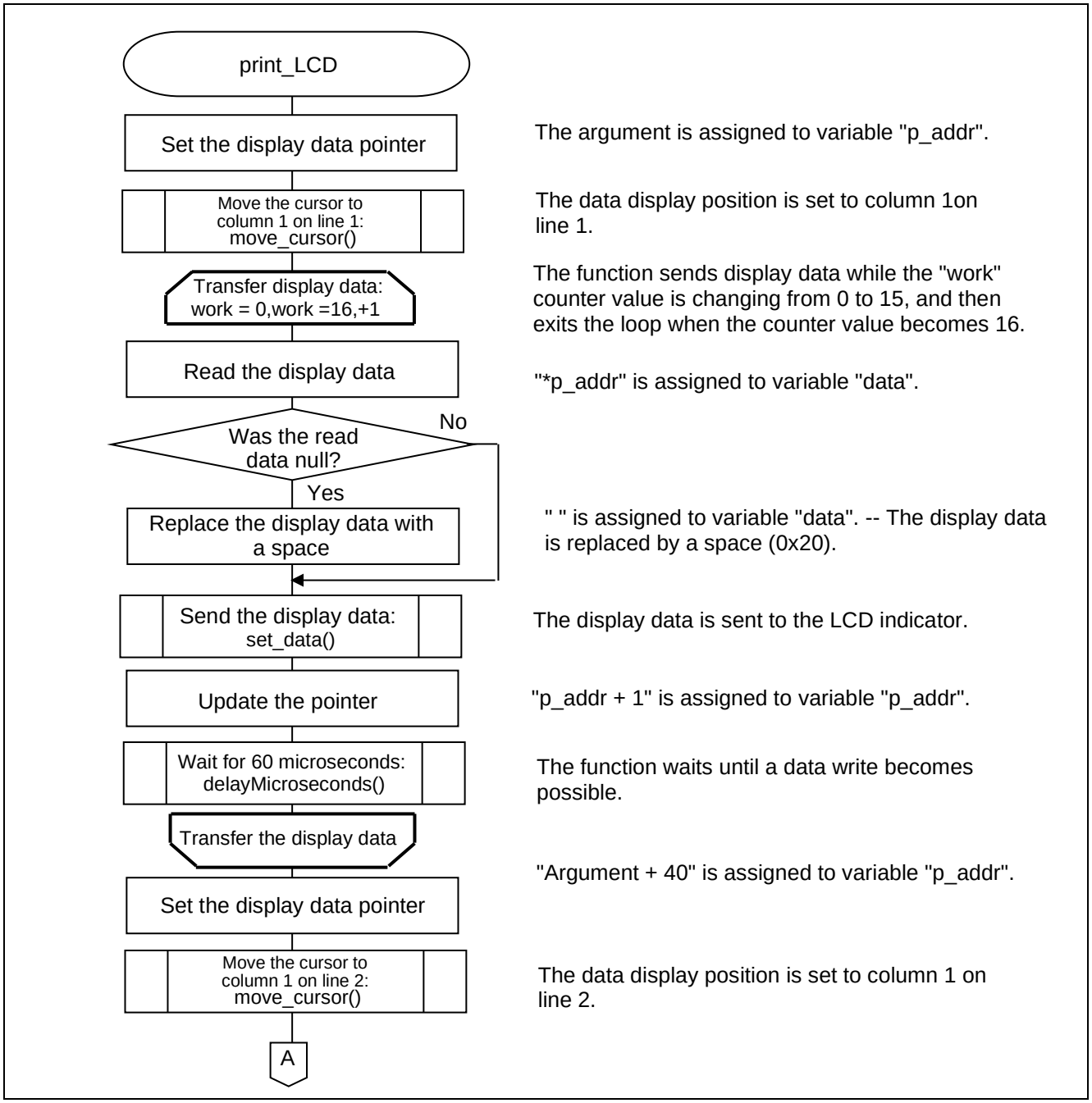
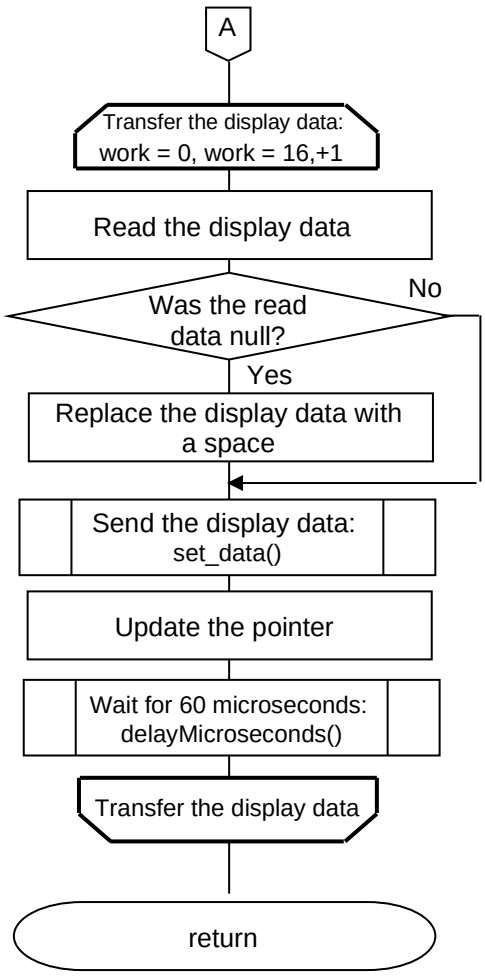


Figure 5.7 Function that Sets Full-Screen Display for the LCD Indicator (1/2)



The function sends display data while the "work" counter value is changing from 0 to 15, and then exits the loop when the counter value becomes 16.

"*p_addr" is assigned to variable "data".

" " is assigned to variable "data". -- The display data is replaced by a space (0x20).

The display data is sent to the LCD indicator.

"p_addr + 1" is assigned to variable "p_addr".

The function waits until a data write becomes possible.

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.

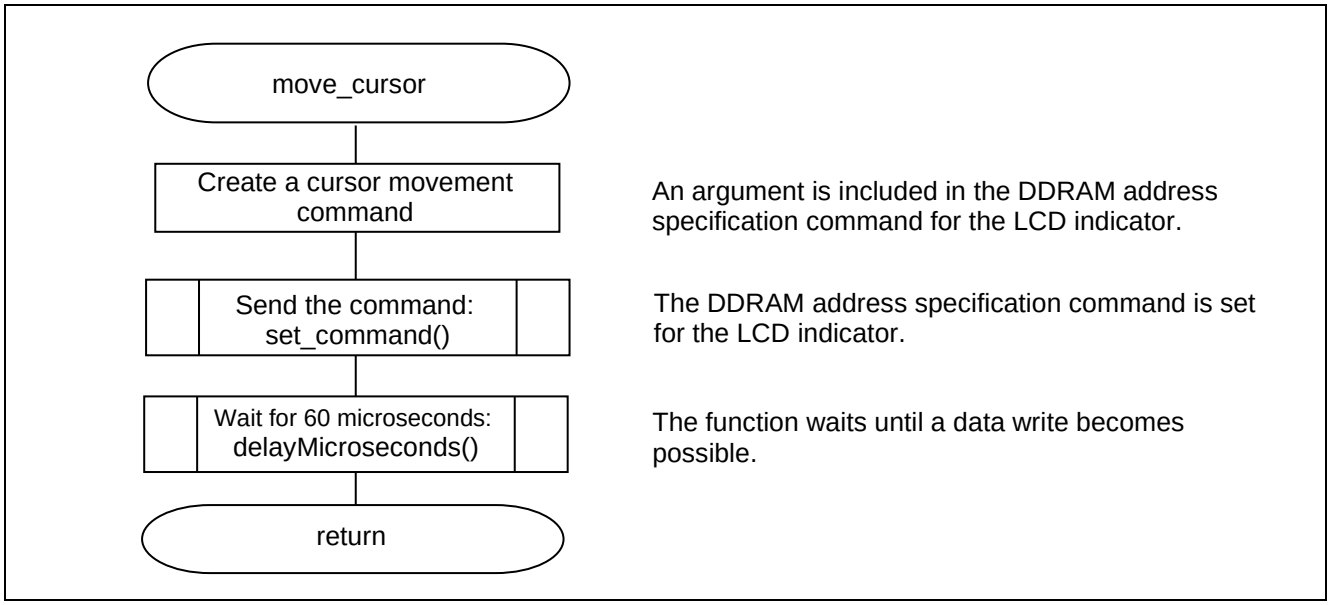


Figure 5.9 Function that Sets the Data Display Position for the LCD Indicator

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.

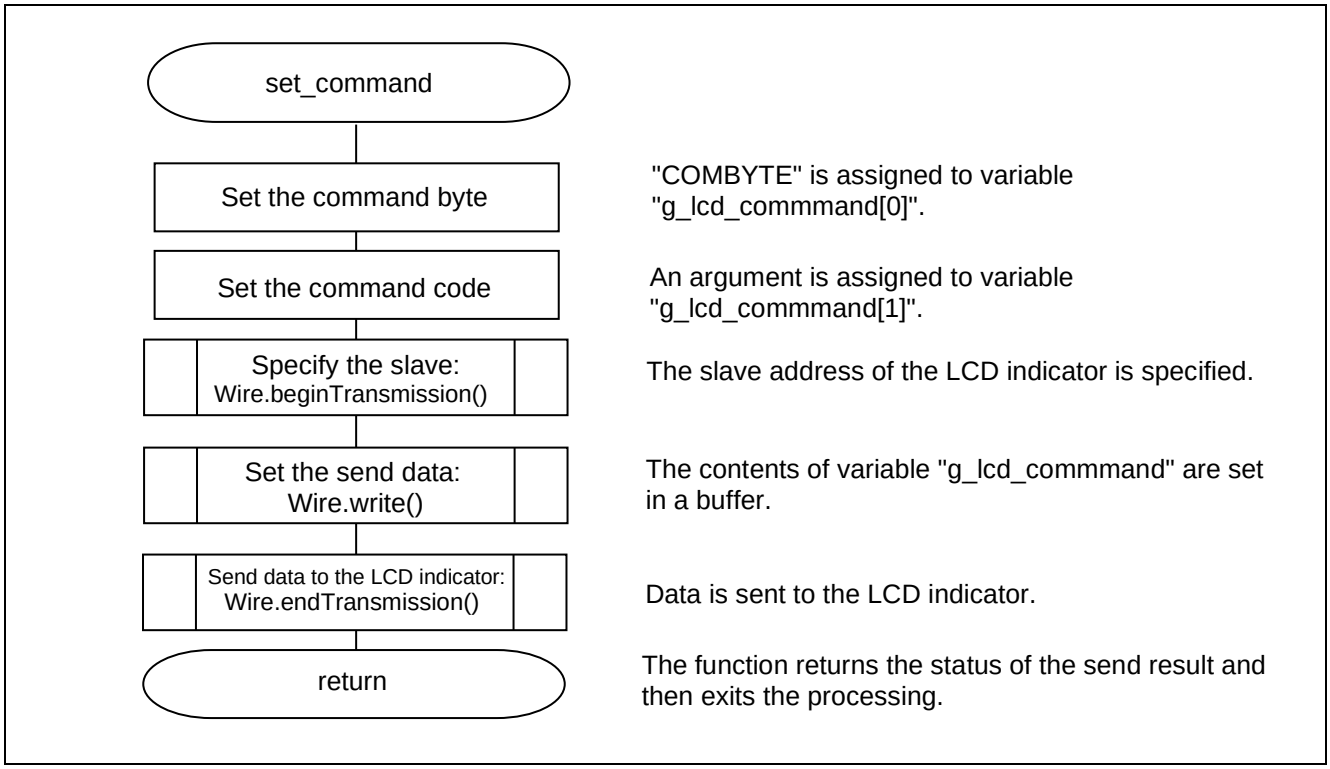


Figure 5.10 Function that Sets a Command for the LCD Indicator

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.

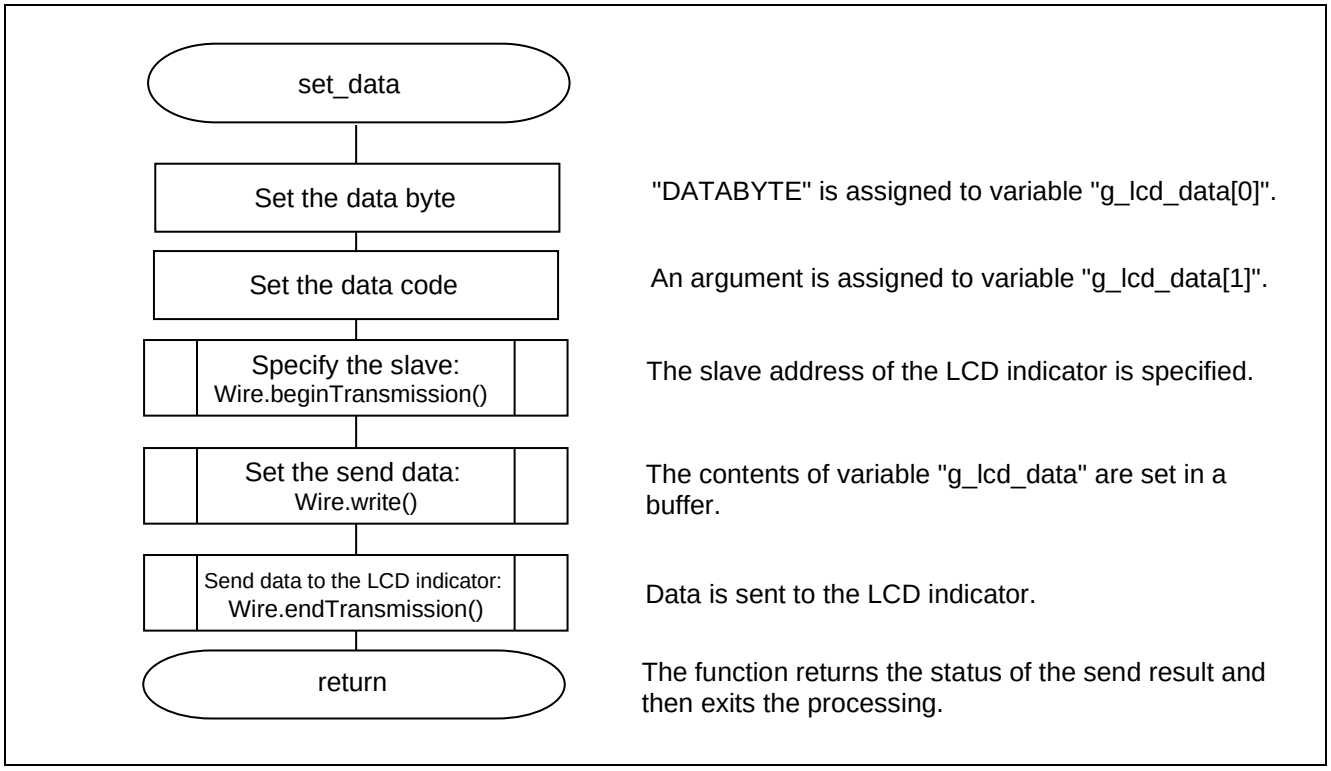


Figure 5.11 Function that Sets Data for the LCD Indicator

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	Jun.16.21	—	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

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/.