Introduction

This application note describes how to implement the multiple slave addresses by using the I2C bus slave function using software.

Operation Checked Device

RL78/I1D

When applied to other microcontrollers, this application note should be modified according to the specifications of the microcontroller used and a thorough evaluation should be made.
Contents

1. Basic Specifications of I2C Bus as Slave ................................................................. 4
   1.1 I2C Bus Specifications .......................................................................................... 4
   1.2 Slave Function Specifications ............................................................................. 4

2. Conditions for Confirming Operations .................................................................. 4

3. Related Application Notes ..................................................................................... 5

4. Hardware Descriptions ......................................................................................... 6
   4.1 Hardware Configuration Example ........................................................................ 6
   4.2 List of Pins Used .................................................................................................. 7

5. Software Descriptions .......................................................................................... 7
   5.1 Operation Summary ............................................................................................ 7
   5.2 List of Settings Reflected to Option Bytes .......................................................... 8
   5.3 List of Constants ................................................................................................ 9
   5.4 List of Variables .................................................................................................. 9
   5.5 List of Functions .................................................................................................. 12
   5.6 Function Specifications ....................................................................................... 13
   5.7 Flowcharts .......................................................................................................... 18

6. Functions as Slaves .............................................................................................. 72

   6.1 Communication Implementation through Software .......................................... 71
   6.2 Functions as Slaves ........................................................................................... 72
6.2.1 LED Display Function ................................................................. 72
6.2.2 A/D Conversion Function .......................................................... 72
6.2.3 RAM Function ........................................................................... 72
6.3 Library Interface Specifications .................................................... 73
6.3.1 I2C Communication Flags .......................................................... 73
6.3.2 Next Communication Starting Functions ..................................... 73
6.3.3 Stop Condition Detection Flag ................................................... 73
6.4 Slave Address Specification ......................................................... 74
6.4.1 Slave Address Table ................................................................. 74
6.4.2 ACK Response Flag ................................................................. 74
6.5 Protocol for Accessing Slaves ....................................................... 75
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.3 Library Interface Specifications .................................................... 73
6.3.1 I2C Communication Flags .......................................................... 73
6.3.2 Next Communication Starting Functions ..................................... 73
6.3.3 Stop Condition Detection Flag ................................................... 73
6.4 Slave Address Specification ......................................................... 74
6.4.1 Slave Address Table ................................................................. 74
6.4.2 ACK Response Flag ................................................................. 74
6.5 Protocol for Accessing Slaves ....................................................... 75
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
6.5.1 Display on LED ........................................................................... 75
6.5.2 Reading A/D Conversion Results ............................................. 75
6.5.3 Reading Data from RAM ............................................................ 76
6.5.4 Writing Data to RAM ............................................................... 76
1. Basic Specifications of I2C Bus as Slave

1.1 I2C Bus Specifications

The following shows the basic specifications of the I2C bus.

- I2C bus connected: Fast mode (200 kbps max.) or standard mode\textsuperscript{Note}
- Slave address 1: 0x60 (A/D conversion and LED display functions)
- Slave address 2: 0x70 (RAM function)
- Slave address 3: 0x80 (not used)
- Slave address 4: 0x90 (not used)
- Extension code: Not supported (ignores code and withdraws from communication)
- Addressing: 8 bits following the slave address used to specify the RAM address

Note: The communication speed when 24 MHz is selected for the CPU/peripheral hardware clock.

1.2 Slave Function Specifications

The following three slave functions are provided. One of the three functions is selected depending on the slave address and the state of transmission/reception.

- LED display function: 8-bit data is displayed on LED. Two display data units are switched over using SW.
- A/D conversion function: 4-channel analog input is converted to digital data. The moving average of the 16 samples is sent to the master.
- RAM function: 128-byte RAM function is provided. The master can read from and write to the arbitrary address by specifying it.

2. Conditions for Confirming Operations

The sample code operations described in this application note are confirmed under the following conditions.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller used</td>
<td>RL78/I1D(R5F117GC)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>• High-speed on-chip oscillator (HOCO) clock: 24 MHz</td>
</tr>
<tr>
<td></td>
<td>• CPU/peripheral hardware clock: 24 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3V (operation possible within 2.9 V to 3.6 V)</td>
</tr>
<tr>
<td></td>
<td>LVD operating mode: reset mode; voltage: 2.75 V</td>
</tr>
<tr>
<td>Integrated development</td>
<td>Renesas Electronics</td>
</tr>
<tr>
<td>environment</td>
<td>CS+ V3.03.00\textsuperscript{Note}</td>
</tr>
<tr>
<td>Assembler</td>
<td>Renesas Electronics</td>
</tr>
<tr>
<td></td>
<td>CC-RL V1.02.00\textsuperscript{Note}</td>
</tr>
<tr>
<td>Board used</td>
<td>RL78/I1D target board (equipped with R5F117GC, LED (8 out of 10 of a module used), SW, and the like.)</td>
</tr>
</tbody>
</table>

[Note] Used in CA78K0R-compatible mode.
3. Related Application Notes

Refer to the following application notes, which are related to this application note.

RL78/G13 Initial Setting Application Note (R01AN2575J)
RL78/I1D I2C Master Communication Control using Serial Array Unit (Simple I2C) Application Note (R01AN3288J)
4. Hardware Descriptions

4.1 Hardware Configuration Example

Figure 4.1 shows an example of the hardware configuration described in this application note.

Notes:
1. The above figure is a simplified circuit image for showing the outline of the connections. The actual circuit should be designed so that the pins are connected appropriately and that electrical characteristics are satisfied (input-only ports should be each connected to \( \text{VDD} \) or \( \text{VSS} \) via a resistor).

2. Set \( \text{VDD} \) to the reset-release voltage (\( \text{V}_{\text{LVD}} \)) specified by LVD or greater.
4.2 List of Pins Used

Table 4.1 lists the pins used and their functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTP6/P32</td>
<td>I/O</td>
<td>I2C communication data signal</td>
</tr>
<tr>
<td>INTP5/P33</td>
<td>I/O</td>
<td>I2C communication clock signal</td>
</tr>
<tr>
<td>P03 to P00</td>
<td>Output</td>
<td>Data output to LED</td>
</tr>
<tr>
<td>P61, P60</td>
<td>Output</td>
<td>LED turning-on timing output</td>
</tr>
<tr>
<td>ANI3 to ANI0</td>
<td>Input</td>
<td>Analog signal input</td>
</tr>
<tr>
<td>P137</td>
<td>Input</td>
<td>SW input</td>
</tr>
</tbody>
</table>

5. Software Descriptions

5.1 Operation Summary

a) Initial Settings

In this application note, the CS+ code creation function is used only for the initial settings of the on-chip peripheral functions. After making the initial settings of the on-chip peripheral functions, data is initialized and the timers for A/D conversion and turning on LED are started.
- A/D conversion and turning-on of LED are processed on the background using the timer interrupts.
- I2C bus communication is processed on the background using the INTP5 and INTP6 interrupts.

b) Main Process

The main process waits for completion of 4-channel A/D conversion. When conversion is completed, the moving average is transferred to the I2C bus transmission buffer. The data transferred to the transmission buffer is transmitted to the I2C bus in response to the instruction from the master. During 4-channel A/D conversion, if the stop condition is detected on the I2C bus, data is transferred from the data reception buffer for turning on LED to the buffer for controlling turning on of LED.

c) A/D conversion end interrupt process

In the A/D conversion end interrupt process, the conversion result of each channel is added. When the count of data to be added reaches 16, the oldest data is replaced with the latest data. When A/D conversion of channels 0 to 3 is completed in scan mode, the main process is informed of completion of A/D conversion.

d) 5-ms timer interrupt process

The 5-ms timer interrupt is used to turn on LED and check SW. The upper 4-bit and lower 4-bit data for turning on LED are used in this order to turn on LED in the time division manner. The state of SW is checked every 50 ms to determine which data to be used.

e) I2C communication interrupt process

The changes of the SDA and SCL signals cause INTP5 and INTP6 interrupts. These interrupts are used as I2C communication interrupts. When an I2C communication interrupt is generated, the communication contents are analyzed and sent to the upper software. After completion of 1-byte communication, if the communication is intended for the slave itself, the communication status and received data are set in the variables, and the transmission/reception end flag (variable _g_IIC_IF) is set. For details, refer to 6.3 Specifications of Library Interface. If the stop condition is detected on the I2C bus, the variable for interfacing (_g_stop_det) is used to inform the main process that the stop condition has been detected, which indicates completion of the I2C bus communication.

As described above, almost all processes are performed based on the interrupts and flags. The main process sets data in the appropriate buffer so that the data prepared by an interrupt process can be used by another interrupt process.
5.2 List of Settings Reflected to Option Bytes

Table 5.1 shows the sample settings reflected to the option bytes.

<table>
<thead>
<tr>
<th>Address</th>
<th>Setting</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00C0</td>
<td>0b11101110</td>
<td>Watchdog timer is stopped. (Counting stopped after a reset release)</td>
</tr>
<tr>
<td>0x00C1</td>
<td>0b01111111</td>
<td>LVD reset mode; 2.75V (2.70 V to 2.87 V)</td>
</tr>
<tr>
<td>0x00C2</td>
<td>0b11100000</td>
<td>HS mode; HOCO: 24 MHz</td>
</tr>
<tr>
<td>0x00C3</td>
<td>0b10000100</td>
<td>On-chip debugging is enabled.</td>
</tr>
</tbody>
</table>
### 5.3 List of Constants

Table 5.2 lists the constants used in sample codes.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Setting</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUTH</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>POWER</td>
<td>4</td>
<td>Specification of A/D conversion sampling count (specify a factorial of 2)</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>2 &lt;&lt; (POWER-1)</td>
<td>A/D conversion sampling count</td>
</tr>
<tr>
<td>DATA_NUMBER</td>
<td>2</td>
<td>Number of data to be displayed on LED</td>
</tr>
<tr>
<td>INT_MASK</td>
<td>1</td>
<td>Interrupt disabled (masked)</td>
</tr>
<tr>
<td>INT_ENABLE</td>
<td>0</td>
<td>Interrupt enabled (mask canceled)</td>
</tr>
<tr>
<td>DETECT_START</td>
<td>0b11110010</td>
<td>Mask bit for detecting to be selected as slave</td>
</tr>
<tr>
<td>DETECT_TRC</td>
<td>0b00000100</td>
<td>TRC (transmission enable) bit</td>
</tr>
<tr>
<td>DETECT_ACK</td>
<td>0b00000100</td>
<td>ACK detection bit</td>
</tr>
<tr>
<td>DETECT_STD</td>
<td>0b00000010</td>
<td>Start condition detection bit</td>
</tr>
<tr>
<td>DETECT_STOP</td>
<td>0b00000001</td>
<td>Stop condition detection bit</td>
</tr>
<tr>
<td>DISP_OFF_DATA</td>
<td>0b00000011</td>
<td>P6 data for turning LED off</td>
</tr>
<tr>
<td>TIMING1</td>
<td>0b00000010</td>
<td>P6 data for lighting upper 4 bits</td>
</tr>
<tr>
<td>TIMING2</td>
<td>0b00000001</td>
<td>P6 data for lighting lower 4 bits</td>
</tr>
<tr>
<td>KEY_TIMING</td>
<td>10</td>
<td>Value for SW state check timing</td>
</tr>
<tr>
<td>DATA_MAX</td>
<td>4</td>
<td>Number of I2C transmission data</td>
</tr>
<tr>
<td>LED_MAX</td>
<td>2</td>
<td>Number of reception data for lighting LED</td>
</tr>
<tr>
<td>TX_LIMIT</td>
<td>DATA_MAX -1</td>
<td>Mask data for transmission pointer</td>
</tr>
<tr>
<td>RX_LIMIT</td>
<td>LED_MAX - 1</td>
<td>Mask data for LED lighting data reception pointer</td>
</tr>
<tr>
<td>SADR_TBL</td>
<td></td>
<td>Table of slave addresses used</td>
</tr>
<tr>
<td>ACK_TBL</td>
<td></td>
<td>ACK response table for each slave address ID</td>
</tr>
<tr>
<td>P_IIC</td>
<td>P3</td>
<td>Port used by the I2C bus</td>
</tr>
<tr>
<td>P_SCL</td>
<td>P3.3</td>
<td>Port used by the SCL signal</td>
</tr>
<tr>
<td>P_DATA</td>
<td>0b00001100</td>
<td>Data for extracting SCL and SDA</td>
</tr>
<tr>
<td>P_DATA_SCL</td>
<td>0b00001000</td>
<td>SCL is high; SDA is low.</td>
</tr>
<tr>
<td>P_SDA</td>
<td>P3.2</td>
<td>Port used by SDA signal</td>
</tr>
<tr>
<td>PM_SCL</td>
<td>PM3.3</td>
<td>Register for controlling SCL signal</td>
</tr>
<tr>
<td>PM_SDA</td>
<td>PM3.2</td>
<td>Register for controlling SDA signal</td>
</tr>
<tr>
<td>ENG_SCL</td>
<td>EGN0.5</td>
<td>SCL falling edge detection enabled</td>
</tr>
<tr>
<td>EPG_SCL</td>
<td>EPG0.5</td>
<td>SCL rising edge detection enabled</td>
</tr>
<tr>
<td>DIS_INTSCL</td>
<td>PMK5</td>
<td>SCL edge detection interrupt mask</td>
</tr>
<tr>
<td>DIS_INTSDA</td>
<td>PMK6</td>
<td>DA edge detection interrupt mask</td>
</tr>
<tr>
<td>RQ_INTSCL</td>
<td>PIF5</td>
<td>SCL edge detection interrupt request</td>
</tr>
<tr>
<td>RQ_INTSDA</td>
<td>PIF6</td>
<td>SDA edge detection interrupt request</td>
</tr>
<tr>
<td>D_SDA</td>
<td>0xFFED.E2</td>
<td>SDA bit in P3 image</td>
</tr>
<tr>
<td>D_SCL</td>
<td>0xFFED.E3</td>
<td>SCL bit in P3 image</td>
</tr>
<tr>
<td>F_TRC</td>
<td>0xFFED.F3</td>
<td>Transmission mode bit in status (g_IICS)</td>
</tr>
<tr>
<td>F_ACKD</td>
<td>0xFFED.F2</td>
<td>ACK detection bit in status (g_IICS)</td>
</tr>
<tr>
<td>F_STD</td>
<td>0xFFED.F1</td>
<td>Start condition detection bit in status (g_IICS)</td>
</tr>
<tr>
<td>F_SPD</td>
<td>0xFFED.0</td>
<td>Stop condition detection bit in status (g_IICS)</td>
</tr>
</tbody>
</table>
### 5.4 List of Variables

Table 5.3 and table 5.4 list the variables used in sample codes.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16_t</code></td>
<td><code>g_conv_data</code></td>
<td>A/D conversion data buffer</td>
<td>R_ADC_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_adc_interrupt()</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td><code>g_sum_data</code></td>
<td>A/D conversion sum buffer</td>
<td>R_ADC_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_adc_interrupt()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_adc_end</code></td>
<td>A/D conversion end flag</td>
<td>R_ADC_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_adc_interrupt()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td><code>gp_set_pt</code></td>
<td>A/D conversion result storage</td>
<td>R_ADC_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointer</td>
<td>r_adc_interrupt()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td><code>gp_sum_pt</code></td>
<td>A/D conversion result sum</td>
<td>R_ADC_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointer</td>
<td>r_adc_interrupt()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_disp_data_bf</code></td>
<td>LED lighting data</td>
<td>R_LED_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R_LED_DispData()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_tau0_channel3_interrupt()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_sel_data</code></td>
<td>Lighting data specification</td>
<td>R_LED_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_tau0_channel3_interrupt()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_disp_timing</code></td>
<td>Lighting timing</td>
<td>R_LED_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_tau0_channel3_interrupt()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_ram_area</code></td>
<td>Buffer for RAM function</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_rx_data</code></td>
<td>Reception data buffer</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R_IICA0_Get()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td><code>g_tx_data</code></td>
<td>Transmission data buffer</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R_IICA0_Put()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_low_data_temp</code></td>
<td>For storing lower byte of</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transmission data</td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_low_data_index</code></td>
<td>Lower byte transmission flag</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_rx_data</code></td>
<td>Flag for address register</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_ptrx_data2</code></td>
<td>I2C transmission/reception</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointer</td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_ptrx_data</code></td>
<td>I2C transmission/reception</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointer</td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
<tr>
<td><code>uint8_t</code></td>
<td><code>g_status</code></td>
<td>I2C communication state flag</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IICA0_Status()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Rx_data()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_R_IIC_Tx_data()</td>
</tr>
</tbody>
</table>
### Table 5.4  List of Global Variables (for Assembly Language Definitions)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>__g_stop_det</td>
<td>Stop condition detection flag</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td>(_g_stop_det)</td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_iic_SDA_interrupt</td>
</tr>
<tr>
<td></td>
<td>__g_IIC_IF</td>
<td>Transmission/reception end flag</td>
<td>R_IICA0_Init()</td>
</tr>
<tr>
<td></td>
<td>(_g_IIC_IF)</td>
<td></td>
<td>r_iic_int_chk()</td>
</tr>
<tr>
<td></td>
<td>__g_IICS</td>
<td>Communication status</td>
<td>r_iic_request()</td>
</tr>
<tr>
<td></td>
<td>(_g_IICS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>__g_IICA</td>
<td>Reception data</td>
<td>r_iic_request()</td>
</tr>
<tr>
<td></td>
<td>(_g_IICA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_ACKE_tbl</td>
<td>Internal buffer for storing initial values to control ACK response to the address</td>
<td>R_IICSS_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_set_ACKE_table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_get_ACKE_table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td>g_ACKE</td>
<td>For controlling ACK response</td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>next_proc</td>
<td>Next INTP5 processing function address</td>
<td>R_IICSS_Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r_iic_SDA_interrupt</td>
</tr>
<tr>
<td></td>
<td>bit_count</td>
<td>Transmission/reception bit count</td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_IICA</td>
<td>Data being shifted during transmission/reception</td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_P_image</td>
<td>For storing P3 data</td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_IICS</td>
<td>I2C status</td>
<td>r_iic_SCL_interrupt</td>
</tr>
<tr>
<td></td>
<td>^1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_IICS</td>
<td>I2C status</td>
<td></td>
</tr>
<tr>
<td></td>
<td>^2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: 1.  bit_count and g_IICA may be accessed simultaneously by a 16-bit access.

2.  The variable g_IICS for indicating I2C communication state has a structure conforming to that of the IICA0.

<table>
<thead>
<tr>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3</td>
<td>ID2</td>
<td>ID1</td>
<td>ID0</td>
<td>F_TRC</td>
<td>F_ACKD</td>
<td>F_STD</td>
<td>F_SPD</td>
</tr>
</tbody>
</table>

[Remarks] The variables __g_stop_det (_g_stop_det), __g_IIC_IF (_g_IIC_IF), __g_IICS (_g_IICS), and __g_IICA (_g_IICA) can be accessed from C language description. The other variables can only be used in library functions described in the assembly language.
5.5 List of Functions

Table 5.5 lists the functions used.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_IICA0_Init()</td>
<td>Initializes variables relating to I2C communication.</td>
</tr>
<tr>
<td>R_IICA0_Status()</td>
<td>Reads out I2C communication state.</td>
</tr>
<tr>
<td>R_IICA0_Get()</td>
<td>Reads out I2C reception data.</td>
</tr>
<tr>
<td>R_IICA0_Put()</td>
<td>Sets data in I2C transmission buffer.</td>
</tr>
<tr>
<td>r_iic_int_chk()</td>
<td>Checks I2C communication completion.</td>
</tr>
<tr>
<td>r_iic_request()</td>
<td>I2C communication end interrupt processing</td>
</tr>
<tr>
<td>R_ADC_Init()</td>
<td>Initializes variables relating to A/D conversion.</td>
</tr>
<tr>
<td>R_ADC_Start()</td>
<td>Starts A/D conversion.</td>
</tr>
<tr>
<td>r_adc_interrupt()</td>
<td>Processes A/D conversion end interrupt.</td>
</tr>
<tr>
<td>R_LED_Init()</td>
<td>Initializes variables relating to LED display.</td>
</tr>
<tr>
<td>R_TM03_Start()</td>
<td>Starts 5-ms interval timer.</td>
</tr>
<tr>
<td>R_LED_DispData()</td>
<td>Sets LED light-emitting data.</td>
</tr>
<tr>
<td>r_tau0_channel3_interrupt()</td>
<td>Processes 5-ms interval timer interrupt.</td>
</tr>
<tr>
<td>__R_IICSS_Init</td>
<td>Initializes I2C.</td>
</tr>
<tr>
<td>__R_IICSS_Status</td>
<td>Returns I2C state (g_IICS value).</td>
</tr>
<tr>
<td>__set_ACKE_table</td>
<td>Sets ACK response for the slave address ID</td>
</tr>
<tr>
<td>__get_ACKE_table</td>
<td>Reads out ACK response setting for the slave address ID</td>
</tr>
<tr>
<td>r_iic_SCL_interrupt</td>
<td>Processes SCL signal edge detection interrupt.</td>
</tr>
<tr>
<td>r_iic_SDA_interrupt</td>
<td>Processes SDA signal edge detection interrupt.</td>
</tr>
<tr>
<td>__Tx_data_sub</td>
<td>Cancels I2C bus wait and starts next data transmission.</td>
</tr>
<tr>
<td>__Tx_end_sub</td>
<td>Exits communication and cancels I2C bus wait.</td>
</tr>
<tr>
<td>__Rx_data_sub</td>
<td>Cancels I2C bus wait and starts next data reception.</td>
</tr>
</tbody>
</table>

A triple line indicates a border between different modules.
5.6 Function Specifications

The following gives the specifications of the functions used in the sample code.

### [Function name] R_IICA0_Init

| Summary | Initializes I2C communication. |
| Header | r_cg_macrodriver.h, r_cg_userdefine.h |
| Declaration | void R_IICA0_Init(void); |
| Description | Initializes variables used for the I2C communication. |
| Arguments | None |
| Return values | None |
| Remarks | None |

### [Function name] R_IICA0_Status

| Summary | Checks I2C communication state. |
| Header | r_cg_macrodriver.h, r_cg_userdefine.h |
| Declaration | uint8_t R_IICA0_Status(void); |
| Description | Reads the variable g_IICS indicating the I2C communication state. If the I2C communication has been completed, performs the corresponding processing. |
| Arguments | None |
| Return values | Value of variable g_IICS (g_status) |
| Remarks | None |

### [Function name] R_IICA0_Get

| Summary | Reads reception data from I2C reception data buffer. |
| Header | r_cg_macrodriver.h, r_cg_userdefine.h |
| Declaration | uint8_t R_IICA0_Get(uint8_t ptr); |
| Description | Reads data specified with the argument (lighting data) from the I2C reception buffer. If the I2C communication has been completed, the corresponding processing is performed. |
| Arguments | Specifies reception data buffer. |
| Return values | Received data |
| Remarks | None |

### [Function name] R_IICA0_Put

| Summary | Sets data in I2C transmission buffer. |
| Header | r_cg_macrodriver.h, r_cg_userdefine.h |
| Declaration | void R_IICA0_Put(uint8_t ptr,uint16_t data); |
| Description | Stores data indicated by the second argument (A/D conversion result) into the address specified by the first argument in the I2C transmission buffer. If the I2C communication has been completed, the corresponding processing is performed. |
| Arguments | First argument Data storage address Second argument Data to be transmitted |
| Return values | None |
| Remarks | None |
### Function Summary

#### r_iic_int_chk
- **Summary**: Checks I2C communication completion.
- **Header**: `r_cg_macrodriver.h, r_cg_userdefine.h`
- **Declaration**: `uint8_t r_iic_int_chk (void);`
- **Description**: Checks the I2C communication end interrupt flag and calls `r_iic_request` if the communication has been completed.
- **Arguments**: None
- **Return values**: I2C status
- **Remarks**: None

#### r_iic_request
- **Summary**: Performs I2C communication completion processing.
- **Header**: `r_cg_macrodriver.h, r_cg_userdefine.h`
- **Declaration**: `void r_iic_request (void);`
- **Description**: This processing corresponds to INTIICA0 of IICA0. Performs processing according to the I2C status `_g_IICS` value.
- **Arguments**: None
- **Return values**: None
- **Remarks**: `_g_IICS` has I2C communication status. `_g_IICA` has receive data.

#### R_ADC_Init
- **Summary**: Makes A/D conversion initial settings.
- **Header**: `r_cg_macrodriver.h, r_cg_userdefine.h`
- **Declaration**: `void R_ADC_Init(void);`
- **Description**: Initializes variables relating to A/D conversion.
- **Arguments**: None
- **Return values**: None
- **Remarks**: None

#### R_ADC_Start
- **Summary**: Starts A/D conversion.
- **Header**: `r_cg_userdefine.h`
- **Declaration**: `void R_ADC_Start(void);`
- **Description**: Starts the A/D converter.
- **Arguments**: None
- **Return values**: None
- **Remarks**: None

#### r_adc_interrupt
- **Summary**: Processes an A/D conversion end interrupt.
- **Header**: `r_cg_macrodriver.h, r_cg_userdefine.h`
- **Declaration**: `#pragma interrupt r_adc_interrupt(vect=INTAD,bank=RB2,enable=true)
__interrupt static void r_adc_interrupt(void);`
- **Description**: Started by an A/D conversion end interrupt; stores the obtained conversion results in the buffer, and simultaneously adds the results 16 times for each channel.
- **Arguments**: None
- **Return values**: None
- **Remarks**: None
### [Function name] R_LED_Init

<table>
<thead>
<tr>
<th>Summary</th>
<th>Performs initialization for LED lighting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h, r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_LED_Init(void);</td>
</tr>
<tr>
<td>Description</td>
<td>Initializes variables for controlling LED lighting.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>None</td>
</tr>
</tbody>
</table>

### [Function name] R_TM03_Start

<table>
<thead>
<tr>
<th>Summary</th>
<th>Starts TM03 (interval timer).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_TM03_Start(void);</td>
</tr>
<tr>
<td>Description</td>
<td>Starts TM03 (5-ms interval timer).</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>None</td>
</tr>
</tbody>
</table>

### [Function name] R_LED_DispData

<table>
<thead>
<tr>
<th>Summary</th>
<th>Sets LED light-emitting data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h, r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_LED_DispData(uint8_t CH_No,uint8_t in_data);</td>
</tr>
<tr>
<td>Description</td>
<td>Stores data specified by the second argument in the buffer specified by the first argument.</td>
</tr>
<tr>
<td>Arguments</td>
<td>First argument, Second argument</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>None</td>
</tr>
</tbody>
</table>

### [Function name] r_tau0_channel3_interrupt

<table>
<thead>
<tr>
<th>Summary</th>
<th>Processes a 5-ms interval timer interrupt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h, r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>#pragma interrupt r_tau0_channel3_interrupt(vect=INTTM03, enable=true) static void r_tau0_channel3_interrupt(void);</td>
</tr>
<tr>
<td>Description</td>
<td>Started by a 5-ms interrupt; controls dynamic LED lighting in 4-bit units. Checks the state of SW connected to P137 every 50 ms and changes data to be lighted.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>None</td>
</tr>
</tbody>
</table>

### [Function name] __R_IICSS_Init

<table>
<thead>
<tr>
<th>Summary</th>
<th>Initializes I2C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void __R_IICSS_Init(void);</td>
</tr>
<tr>
<td>Description</td>
<td>Initializes I2C control variables and hardware.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return values</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>None</td>
</tr>
</tbody>
</table>
[Function name] __R_IICSS_Status
Summary Checks I2C status.
Declaration uint8_t _R_IICSS_Status (void);
Description Passes the value of variable g_IICS indicating I2C bus state.
Arguments None
Return values I2C bus status
Remarks None

[Function name] __set_ACKE_table
Summary Sets ACK response for the slave address ID.
Declaration void _set_ACKE_table(uint8_t ACKE);
Description Sets the ACK response for the slave address ID indicated by the argument bits 4 to 1, to the state indicated by bit 0.
Arguments ACK response
Return values None
Remarks None

[Function name] __get_ACKE_table
Summary Reads out ACK response setting for the slave address ID.
Declaration uint8_t _get_ACKE_table(uint8_t ID);
Description Returns ACK response setting for the slave address passed by the argument.
Arguments Slave address ID
Return values ACK response setting value
Remarks None

[Function name] r_iic_SCL_interrupt
Summary Processes an SCL edge detection interrupt.
Declaration r_iic_SCL_interrupt .VECTOR 0x00012
Description Detects the SCL signal edge, reads SCL and SDA, and performs the corresponding processing. The processed contents are indicated by the address stored in the variable next_proc.
Arguments None
Return values None
Remarks None

[Function name] r_iic_SDA_interrupt
Summary Processes an SDA edge detection interrupt.
Declaration r_iic_SDA_interrupt .VECTOR 0x00014
Description Started on detection of the SDA signal edge; detects the start and stop conditions based on the SDA and SCL signal state.
Arguments None
Return values None
Remarks None
### __Tx_data_sub__

**Summary**
Starts next data transmission.

**Declaration**
```c
void _Tx_data_sub(uint8_t data);
```

**Description**
Outputs MSB of the data passed by the argument to the SDA signal, cancels the I2C bus wait, and starts data transmission.

**Arguments**
Next transmission data

**Return values**
None

**Remarks**
None

### __Rx_data_sub__

**Summary**
Starts next data reception.

**Declaration**
```c
void _Rx_data_sub(void);
```

**Description**
Cancels the I2C bus wait, and starts data reception.

**Arguments**
None

**Return values**
None

**Remarks**
None

### __Tx_end_sub__

**Summary**
Performs transmission completion processing.

**Declaration**
```c
void _Tx_end_sub(void);
```

**Description**
Exits the communication and cancels the I2C bus wait.

**Arguments**
None

**Return values**
None

**Remarks**
None
5.7 Flowcharts

Figure 5.1 shows the overall flow of the process described in this application note.

![Overall Flow](image)

Before initial function execution, option bytes are referred to.

- **Start**
  - Initial setting function `hdwinit()`
  - Main process `main()`
  - End

This is processed by the start-up routine (r_cg_cstart.asm etc.). The memory-related settings are made between calling the initial setting function and main process function.

5.7.1 Initial Setting Function

Figure 5.2 shows the flowchart of the initial setting function.

![Initial Setting Function](image)

- `hdwinit()`
  - Disable interrupts.
  - System function `R_Systeminit()`
  - Return

IE ← 0

Initial setting of internal peripheral functions
5.7.2 System Function

Figure 5.3 shows the flowchart of the system function.

![System Function Flowchart]

- **R_Systeminit()**
  - Set peripheral I/O redirection.
    - PIOR0 register ← 0x00
  - Set I/O ports.
    - R_PORT_Create()
  - Set CPU clocks.
    - R_CGC_Create()
  - Set TAU.
    - R_TAU0_Create()
  - Set A/D converter.
    - R_ADC_Create()
  - Set external interrupts.
    - R_INTC_Create()
  - Disable detection of illegal memory access.
    - IAWCTL register ← 0x00
  - Return
5.7.3 Setting CPU Clocks

Figure 5.4 shows the flowchart for setting the CPU clocks.

R_CGC_Create()

Set clock oscillators.

Select subsystem clock.

Select CPU/peripheral hardware clock (fCLK).

Enable high-speed OCO oscillation.

return

Figure 5.4 Setting CPU Clocks

CMC register ← 0b00000000 : High-speed system clock not used; subsystem clock not used
MSTOP bit ← 1
MIOEN bit ← 0 : Disable middle-speed OCO.

SELLOSC bit ← 1 : Select LOCO for subsystem clock.
XTSTOP bit ← 1 : Stop XT1 clock oscillation.
WUTMMCK0 bit ← 1 : Supply fIL to interval timer.

CSS bit ← 0 : Select high-speed OCO for CPU/peripheral hardware clock (fCLK).
MCM0 bit ← 0 : Select high-speed OCO (fH).
MCM1 bit ← 0 :
HIOSTOP bit ← 0 : Enable high-speed OCO oscillation.
5.7.4 Setting I/O Ports

Figure 5.5 shows the flowchart for setting the I/O ports.

Note: Design unused ports so that the electrical characteristics are satisfied by appropriately treating the pertinent pins. Separately connect unused input-only pins to VDD or VSS via a resistor.

```
R_PORT_Create()

Set P0 and P6 output latch.

Set P40 pull-up resistor.

Set P0 operating mode.

Set P0, P4, and P6 modes.

return
```

- P0 register ← 0x00 : Set P03 to P00 to 0.
- P6 register ← 0x03 : Set P61 and P60 to 1.

- PU4 register ← 0x01 : Set P40 pull-up resistor.

- PMC0 register ← 0xF3 : Set P03 and P02 as digital I/O.

- PM0 register ← 0xF0 : Set P03 to P00 as output port.
- PM4 register ← 0xFF : Set P40 as input port.
- PM6 register ← 0xFC : Set P61 and P60 as output.

Figure 5.5 Setting I/O Ports
5.7.5 Setting Timer Array Unit

Figure 5.6 shows the flowchart for setting the timer array unit.

```
R_TAU0_Create()

Reset TAU0.
Release TAU0 from reset.
Supply clocks to timer array unit.
Set TAU operating clock.
Disable TAU operation.
Set TAU interrupts.
Set TM03 operating mode.
Disable TM03 output.
```

`TAU0RES bit ← 1 : Reset TAU0.
TAU0RES bit ← 0 : Release TAU0 from reset.
TAU0EN bit ← 1`

`TPS0 register ← 0x0001 : Operating clock 0 (CK00): 12 MHz`

`TT0 register ← 0x0A0F : Stop operation of all channels.
TMMK03 to TMMK00 bits ← 1111 : Mask interrupt requests.
TMIF03 to TMIF00 bits ← 0000 : Clear interrupt requests.
TMPR103 and TMPR003 bits ← 10 : Interrupt priority level 2
TMR03 register ← 0x0000 : Interval timer
TDR03 register ← 0xEA5F : 5-ms intervals
TOM03 bit ← 0 : Master channel mode output
TO03 bit ← 0 : Positive logic output
TOE03 bit ← 0 : Disable TM03 output.
return

Figure 5.6 Setting Timer Array Unit

Resetting TAU0
- Peripheral reset control register 0 (PRR0)
  Reset TAU0.
Symbol: PRR0

```
7 6 5 4 3 2 1 0
0 0 0 0 0 0 TAU0RES
0 0 0 0 0 x 0 1
```

Bit 0

<table>
<thead>
<tr>
<th>SAU0RES</th>
<th>Reset control of timer array unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Releases the timer array unit from the reset state.</td>
</tr>
<tr>
<td>1</td>
<td>Resets the timer array unit.</td>
</tr>
</tbody>
</table>

- Peripheral reset control register 0 (PRR0)
  Reset TAU0.
Symbol: PRR0
Starting clock supply to timer array unit 0
- Peripheral enable register 0 (PER0)
  Start clock supply to timer array unit 0.

Symbol: PER0

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCWEN 0</td>
</tr>
<tr>
<td>ADCEN 0</td>
</tr>
<tr>
<td>ADEN 0</td>
</tr>
<tr>
<td>SAU0EN 0</td>
</tr>
<tr>
<td>TAU0EN 0</td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>TAU0EN</th>
<th>Control of input clock to timer array unit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>入力クロック供給停止</td>
</tr>
<tr>
<td>1</td>
<td>Supplies input clock.</td>
</tr>
</tbody>
</table>

Setting timer clock frequency
- Timer clock select register 0 (TPS0)
  Select the operating clock for timer array unit 0.

Symbol: TPS0

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 PRS 031 0 PRS 030 0 0 PRS 021 0 PRS 020 0 PRS 013 0 PRS 012 0 PRS 011 0 PRS 010 0 PRS 003 0 PRS 002 0 PRS 001 0 PRS 000</td>
</tr>
<tr>
<td>0 0 x x 0 0 x x x x 0 0 0 1</td>
</tr>
</tbody>
</table>

Bits 3 to 0

<table>
<thead>
<tr>
<th>PRS 003</th>
<th>PRS 002</th>
<th>PRS 001</th>
<th>PRS 000</th>
<th>Operating clock (CK00) selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>fCLK</td>
<td>1 MHz</td>
<td>2 MHz</td>
<td>4 MHz</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>fCLK/2</td>
<td>500 kHz</td>
<td>1 MHz</td>
<td>2 MHz</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>fCLK/2^n</td>
<td>250 kHz</td>
<td>500 kHz</td>
<td>1 MHz</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>fCLK/2^n</td>
<td>125 kHz</td>
<td>250 kHz</td>
<td>2 MHz</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>fCLK/2^n</td>
<td>62.5 kHz</td>
<td>125 kHz</td>
<td>1 MHz</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>fCLK/2^n</td>
<td>31.3 kHz</td>
<td>62.5 kHz</td>
<td>125 kHz</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>fCLK/2^n</td>
<td>15.6 kHz</td>
<td>31.3 kHz</td>
<td>62.5 kHz</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>fCLK/2^n</td>
<td>7.81 kHz</td>
<td>15.6 kHz</td>
<td>31.3 kHz</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>fCLK/2^n</td>
<td>3.91 kHz</td>
<td>7.81 kHz</td>
<td>15.6 kHz</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>fCLK/2^n</td>
<td>1.95 kHz</td>
<td>3.91 kHz</td>
<td>7.81 kHz</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>fCLK/2^n</td>
<td>0.977 kHz</td>
<td>1.95 kHz</td>
<td>3.91 kHz</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>fCLK/2^n</td>
<td>0.977 kHz</td>
<td>1.95 kHz</td>
<td>3.91 kHz</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>fCLK/2^n</td>
<td>244 kHz</td>
<td>488 kHz</td>
<td>977 kHz</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>fCLK/2^n</td>
<td>122 Hz</td>
<td>244 Hz</td>
<td>488 Hz</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>fCLK/2^n</td>
<td>61 Hz</td>
<td>122 Hz</td>
<td>244 Hz</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>fCLK/2^n</td>
<td>30.5 Hz</td>
<td>61 Hz</td>
<td>122 Hz</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Stopping timer operation
- Timer channel stop register 0 (TT0)
  Select to stop timer channel operation.

Symbol: TT0

<table>
<thead>
<tr>
<th>Bit</th>
<th>Trigger for stopping channel n operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>トリガ動作しない</td>
</tr>
<tr>
<td>1</td>
<td>Clears TE0n bit to 0 to stop counting operation (stop-trigger generated).</td>
</tr>
</tbody>
</table>

Setting timer count end interrupt
- TMMK03 bit in interrupt mask flag register (MK1L)
  Set interrupt masks.
- TMIF03 bit in interrupt request flag register (IF1L)
  Clear interrupt request flags.
- MPR003 and TMPR103 bits in priority setting flag registers (PR01L, PR11L)
  Set TM03 interrupts to priority level 2.

Symbol: MK1L

Bit 5

<table>
<thead>
<tr>
<th>TMMK03</th>
<th>Control of interrupt processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>割り込み処理許可</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>

Symbol: IF1L

Bit 5

<table>
<thead>
<tr>
<th>TMIF03</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupt request signals have been generated.</td>
</tr>
<tr>
<td>1</td>
<td>割り込み要求信号が発生し、割り込み要求状態</td>
</tr>
</tbody>
</table>

Symbol: PR01L, PR11L

Bit 5

<table>
<thead>
<tr>
<th>TMPR003</th>
<th>INTTM03 priority level selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>レベル 0 を指定(高優先順位)</td>
</tr>
<tr>
<td>0 1</td>
<td>レベル 1 を指定</td>
</tr>
<tr>
<td>1 0</td>
<td>Level 2</td>
</tr>
<tr>
<td>1 1</td>
<td>レベル 3 を指定(低優先順位)</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Setting channel 3 operating mode

- Timer mode register 03 (TMR03)
  Select operating clock (fMCK).
  Select the counting clock.
  Set start and capture triggers.
  Select valid edge of timer input.
  Set the operating mode.

Symbol: TMR03

<table>
<thead>
<tr>
<th>CKS031</th>
<th>CKS030</th>
<th>Channel 3 operating clock (fMCK) selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Operating clock CK00 specified with timer clock selection register 0 (TPS0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>タイマクロック選択レジスタ0 (TPS0) で設定した動作クロックCK02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>タイマクロック選択レジスタ0 (TPS0) で設定した動作クロックCK01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>タイマクロック選択レジスタ0 (TPS0) で設定した動作クロックCK03</td>
</tr>
</tbody>
</table>

Bit 12

<table>
<thead>
<tr>
<th>CCS03</th>
<th>Channel 3 counting clock (fTCL) selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operating clock (fMCK) specified with CKS031 and CKS030 bits</td>
</tr>
<tr>
<td>1</td>
<td>TI03 端子からの入力信号の有効エッジ</td>
</tr>
</tbody>
</table>

Bit 11

<table>
<thead>
<tr>
<th>SPLIT03</th>
<th>8-bit timer/16-bit timer operation selection of channel 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16-bit timer operation</td>
</tr>
<tr>
<td>1</td>
<td>8 ビット・タイマとして動作</td>
</tr>
</tbody>
</table>

Bits 10 to 8

<table>
<thead>
<tr>
<th>STS032</th>
<th>STS031</th>
<th>STS030</th>
<th>Setting for channel 3 start and capture triggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Only software trigger start is valid (deselect the other trigger sources.)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TI00 端子入力の有効エッジを、スタート・トリガ、キャプチャ・トリガの両方に使用</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>TI00 端子入力の有効エッジを、スタート・トリガとキャプチャ・トリガに分けて使用</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>マスタ・チャネルの割り込み信号を使用（複数チャネル連動動作機能のスレーブ・チャネル時）</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Bits 7 and 6

<table>
<thead>
<tr>
<th>CIS031</th>
<th>CIS030</th>
<th>Valid edge selection of TI03 pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Falling edge</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>立ち上がりエッジ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>落ちエッジ（ロウ・レベル幅測定時）</td>
</tr>
<tr>
<td></td>
<td></td>
<td>スタート・トリガ：立ち下がりエッジ、キャプチャ・トリガ：立ち上がりエッジ</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>落ちエッジ（ハイ・レベル幅測定時）</td>
</tr>
<tr>
<td></td>
<td></td>
<td>スタート・トリガ：立ち上がりエッジ、キャプチャ・トリガ：立ち下がりエッジ</td>
</tr>
</tbody>
</table>

Bits 3 to 0

<table>
<thead>
<tr>
<th>MD033</th>
<th>MD032</th>
<th>MD031</th>
<th>MD030</th>
<th>Channel 3 operating mode</th>
<th>Corresponding functions</th>
<th>TCR counting operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/0</td>
<td>Interval timer mode</td>
<td>Interval timer/square wave output/divider function/PWM output (master)</td>
<td>Decrementing</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1/0</td>
<td>キャプチャ・モード</td>
<td>入力パルス間隔測定</td>
<td>アップ・カウント</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>イベント・カウンタ・モード</td>
<td>外部イベント・カウンター</td>
<td>ダウン・カウント</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/0</td>
<td>ワンカウンタ・モード</td>
<td>ディレイ・カウンタ／ワンショット・パルス出力／PWM出力（スレーブ）</td>
<td>ダウン・カウント</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>キャプチャ＆ワンカウンタ・モード</td>
<td>入力信号のハイ/ロウ・レベル幅測定</td>
<td>アップ・カウント</td>
</tr>
</tbody>
</table>

Setting delay time

- Timer data register 03 (TDR03)
  Set the delay time.

Symbol: TDR03

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.

Disabling timer output

- Timer output mode register 0 (TOM0L)
  Set master mode output.
- Timer output level register 0 (TOL0L)
  Set positive logic output.
- Timer output register 0 (TO0L)
  Set output to 0.
- Timer output enable register 0 (TOE0L)
  Enable/disable timer output of each channel.
Bit 3

<table>
<thead>
<tr>
<th>Symbol: TOM0L</th>
<th>Control of channel 3 timer output mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Master channel output mode</td>
</tr>
<tr>
<td>1</td>
<td>スレーブ・チャネル出力モード</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>Symbol: TOL0L</th>
<th>Control of channel 3 timer output level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Positive logic output (active high)</td>
</tr>
<tr>
<td>1</td>
<td>反転出力（アクティブ・ロー）</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>Symbol: TO0L</th>
<th>Control of channel 3 timer output level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low</td>
</tr>
<tr>
<td>1</td>
<td>ハイ・レベル</td>
</tr>
</tbody>
</table>

Bit 3

<table>
<thead>
<tr>
<th>Symbol: TOE0L</th>
<th>Control of channel 3 timer output enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disables TO03 (timer channel output bit) operation triggered by counting operation.</td>
</tr>
<tr>
<td>1</td>
<td>カウント動作によるTO03（タイマ・チャネル出力ビット）の動作許可。</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
5.7.6 Setting A/D Converter

Figure 5.7 shows the flowchart for setting the A/D converter.

![Flowchart for Setting A/D Converter](image)

**R_ADC_Create()**

- **Reset ADC.**
- **Cancel ADC reset.**
- **Supply clock to A/D converter.**
- **Stop A/D converter operation.**
- **Set A/D converter interrupts.**
- **Set multiplexed ports.**
- **Set A/D converter operating mode.**
- **Set conversion result comparison value.**
- **Set scan range.**
- **Wait for a specified time period.**
- **Enable voltage comparator operation.**
- **return**

**ADRES bit ← 1 : Reset the ADC.**

**ADRES bit ← 0 : Cancel the ADC reset.**

**ADCE bit ← 1**

**ADMK bit ← 1 : Mask interrupt requests.**

**ADIF bit ← 0 : Clear interrupt requests.**

**ADPR1 and ADPR0 bits ← 01 : Interrupt priority level 1**

**PM13 to PM10 bits ← 1111 : Set input mode.**

**ADM0 register ← 0x00 : Stop the operation.**

**Stop A/D converter operation.**

**ADM0 register ← 0x40 : Scan mode, 9 μs as standard**

**ADM1 register ← 0x00 : Software trigger, continuous conversion**

**ADM2 register ← 0x00 : AVREF = VDD and VSS**

**ADUL register ← 0xFF : Set the upper limit.**

**ADLL register ← 0x00 : Set the lower limit.**

**Set multiplexed ports.**

**PM13 to PM10 bits ← 1111 : Set input mode.**

**Set A/D converter operating mode.**

**PM13 to PM10 bits ← 1111 : Set input mode.**

**Set conversion result comparison value.**

**ADU0L register ← 0xFF : Set the upper limit.**

**ADLL register ← 0x00 : Set the lower limit.**

**Wait for a specified time period.**

**ADCE bit ← 1 : Enable operation of the A/D voltage comparator.**

---

**Resetting ADC**

- **Peripheral reset control register 0 (PRR0)**

  Resets the ADC.

**Symbol: PRR0**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ADCRES</td>
<td>0</td>
<td>0</td>
<td>SAU0RES</td>
<td>0</td>
<td>TAU0RES</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1/0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

**Bit 5**

<table>
<thead>
<tr>
<th>ADCRES</th>
<th>Control of A/D converter reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cancels the reset of A/D converter.</td>
</tr>
<tr>
<td>1</td>
<td>Resets the A/D converter.</td>
</tr>
</tbody>
</table>
Starting clock supply to A/D converter
- Peripheral enable register 0 (PER0)
  Starts supplying clock to the A/D converter.
Symbol: PER0

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTCEN</td>
<td>0</td>
<td>ADCEN</td>
<td>I2CA0EN</td>
<td>SAU1EN</td>
<td>SAU0EN</td>
<td>0</td>
<td>TAU0EN</td>
</tr>
</tbody>
</table>

Bit 5
ADCEN  Control of A/D converter input clock
0  入力クロック供給停止
1  Supplies input clock.

Stopping A/D converter operation
- A/D converter mode register 0 (ADM0)
  Stops A/D converter operation.
Symbol: ADM0

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCS</td>
<td>ADMD</td>
<td>FR2</td>
<td>FR1</td>
<td>FR0</td>
<td>LV1</td>
<td>LV0</td>
<td>ADCE</td>
</tr>
</tbody>
</table>

Bit 7
ADCS  Control of A/D conversion operation
0  停止変換動作許可
1  変換動作許可

Bit 0
ADCE  Control of A/D voltage comparator operation
0  停止A/D電圧コンパレータ動作許可
1  A/D電圧コンパレータ動作許可

Setting A/D conversion end interrupt
- ADMK bit in the interrupt mask flag register (MK1H)
  Set interrupt masks.
- ADIF bit in the interrupt request flag register (IF1H)
  Clear interrupt request flags.
- ADPR0 and ADPR1 bits in the priority order specification flag register (PR01H, PR11H)
  Set the A/D conversion end interrupt priority to level 1.

Bit 0
ADMK  Control of interrupt processing
0  割り込み処理許可
1  Disables interrupt processing.

Bit 0
ADIF  Interrupt request flag
0  割り込み要求信号が発生したが割り込み要求状態
1  割り込み要求信号は発生していない。
Setting multiplexed ports

- Port mode control register 1 (PMC1)
  Set the pins to analog input.
- Port mode register 1 (PM1)
  Turn off the output buffer of the port.

Symbol: PMC1

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>ADPR1</th>
<th>ADPR0</th>
<th>Selection of priority level.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>レベル 0 を指定（高優先順位）</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td><strong>Specifies level 1.</strong></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>レベル 2 を指定</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>レベル 3 を指定（低優先順位）</td>
</tr>
</tbody>
</table>

Bits 3 to 0

<table>
<thead>
<tr>
<th>PM1n</th>
<th>Digital IO/analog input selection for P1n pin (n = 0-7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>デジタル入出力(アナログ入力以外の兼用機能)</td>
</tr>
<tr>
<td>1</td>
<td>Analog input</td>
</tr>
</tbody>
</table>

Symbol: PM1

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>PM1n</th>
<th>P1n pin I/O mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>出力モード（出力バッファ・オン）</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Input mode (output buffer turned off)</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
### Symbol: ADM0

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5 to 1</th>
<th>ADSC</th>
<th>ADMD</th>
<th>FR2</th>
<th>FR1</th>
<th>LV0</th>
<th>LV1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Bit 6

<table>
<thead>
<tr>
<th>ADMD</th>
<th>A/D conversion channel selection mode setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>セレクト・モード</td>
</tr>
<tr>
<td>1 0</td>
<td>Scan mode</td>
</tr>
</tbody>
</table>

#### Bits 5 to 1

<table>
<thead>
<tr>
<th>FR2</th>
<th>FR1</th>
<th>FR0</th>
<th>LV1</th>
<th>LV0</th>
<th>Conversion time for 12-bit resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( f_{\text{CLK}}=1\text{MHz} )</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>設定禁止</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>設定禁止</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>54μs</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40.5μs</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.75μs</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1 1</td>
<td>0</td>
<td>0</td>
<td>54μs</td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>27μs</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>13.5μs</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66μs</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66μs</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49.5μs</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41.25μs</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66μs</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66μs</td>
</tr>
</tbody>
</table>

### Symbol: ADM1

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>ADTMD1</th>
<th>ADTMD0</th>
<th>ADSCM</th>
<th>ADTR1</th>
<th>ADTR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

#### Bits 7 and 6

<table>
<thead>
<tr>
<th>ADTMD1</th>
<th>ADTMD0</th>
<th>Selection of A/D conversion trigger mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Software trigger mode</td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td>ハードウェア・トリガ・ノーウェイト・モード</td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>ハードウェア・トリガ・ウェイト・モード</td>
</tr>
</tbody>
</table>

#### Bit 5

<table>
<thead>
<tr>
<th>ADSCM</th>
<th>Selection of A/D conversion operating mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Continuous conversion mode</td>
</tr>
<tr>
<td>1</td>
<td>ワンショット変換モード</td>
</tr>
</tbody>
</table>
### Symbol: ADM1

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADTMD1</td>
<td>ADTMD0</td>
<td>ADSCM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ADTRS1</td>
<td>ADTRS0</td>
</tr>
</tbody>
</table>

#### Bits 1 and 0

<table>
<thead>
<tr>
<th>ADTRS1</th>
<th>ADTRS0</th>
<th>Selection of hardware trigger signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Timer channel 1 count end or capture end interrupt signal (INTTM01)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ELC で選択されたイベント信号</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>リアルタイム・クロック 2 割り込み信号 (INTRTC)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ビット・インターバル・タイマ 割り込み信号 (INTIT)</td>
</tr>
</tbody>
</table>

### Symbol: ADM2

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADREFP1</td>
<td>ADREFP0</td>
<td>ADREFM</td>
<td>0</td>
<td>ADRCK</td>
<td>AWC</td>
<td>0</td>
<td>ADTYPE</td>
</tr>
</tbody>
</table>

#### Bits 7 and 6

<table>
<thead>
<tr>
<th>ADREFP1</th>
<th>ADREFP0</th>
<th>Selection of A/D converter plus-side reference voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Supplied from AVDD</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>AVREFP/ANI0 から供給</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>内部基準電圧 (1.45 V) から供給</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>設定禁止</td>
</tr>
</tbody>
</table>

#### Bit 5

<table>
<thead>
<tr>
<th>ADREFM</th>
<th>Selection of A/D converter minus-side reference voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Supplied from AVSS.</td>
</tr>
<tr>
<td>1</td>
<td>AVREFM/ANI1 から供給</td>
</tr>
</tbody>
</table>

#### Bit 3

<table>
<thead>
<tr>
<th>ADRCK</th>
<th>Check of conversion result upper-limit/lower-limit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt signal (INTAD) is generated when ADLL register (\leq) ADCR register (\leq) ADUL register (AREA1)</td>
</tr>
<tr>
<td>1</td>
<td>ADCR レジスタ (\leq) ADLL レジスタ (AREA2), ADCR レジスタ (\leq) ADUL レジスタ (AREA3) のとき割り込み信号 (INTAD) が発生</td>
</tr>
</tbody>
</table>

#### Bit 2

<table>
<thead>
<tr>
<th>AWC</th>
<th>SNOOZE mode setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SNOOZE mode not used</td>
</tr>
<tr>
<td>1</td>
<td>SNOOZE モード機能を使用する</td>
</tr>
</tbody>
</table>

#### Bit 0

<table>
<thead>
<tr>
<th>ADTYPE</th>
<th>Selection of A/D conversion resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12-bit resolution</td>
</tr>
<tr>
<td>1</td>
<td>8 ビット分解能</td>
</tr>
</tbody>
</table>

**Note:** For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Setting conversion result upper limit and lower limit
- Conversion result comparison upper limit setting register (ADUL)
  Set the upper limit value.
- Conversion result comparison lower limit setting register (ADLL)
  Sets the lower limit value.

Symbol: ADUL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Symbol: ADLL

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Setting A/D conversion channels
- Analog input channel specification register (ADS)
  Set ANI0 to ANI3.

Symbol: ADS

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 4 to 0

<table>
<thead>
<tr>
<th>ADS4</th>
<th>ADS3</th>
<th>ADS2</th>
<th>ADS1</th>
<th>ADS0</th>
<th>Analog input channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ANI0—ANI3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>ANI1—ANI4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>ANI2—ANI5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>ANI3—ANI6</td>
</tr>
</tbody>
</table>

The rest of the combinations are omitted.
5.7.7 Setting External Interrupts

Figure 5.8 shows the flowchart for setting the external interrupts.

```plaintext
R_INTC_Create()

Disable all the external interrupts.

Set interrupt priority order.

Set detected edge.

Set multiplexed ports.

return
```

PMK6 to PMK0 bits ← 1 : Mask all the interrupt requests.
PIF6 to PIF0 bits ← 0 : Clear the interrupt requests.

PPR15, PPR05 bits ← 0, 0 : Set INTP5 to the highest priority.
PPR16, PPR06 bits ← 0, 0 : Set INTP6 to the highest priority.

EGN0 register ← 0x40 : Detect INTP6 falling edge.
EGP0 register ← 0x60 : Detect INTP5 and INTP6 rising edges.

PM3.3 and PM3.2 bits ← 1, 1 : Turns the output buffer off.

Disabling all the external interrupts
- Interrupt mask flag register (MK0L, MK0H)
  Mask interrupt requests.
- Interrupt request flag register (IF0L, IF0H)
  Clear interrupt requests.

Symbol: PMK0L

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMK5</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Symbol: PMK0H

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTITMK</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

Bit n

<table>
<thead>
<tr>
<th>PMKn</th>
<th>Control of interrupt processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>許可割り込み処理許可</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>
Setting interrupt priority level

- PPR15, PPR15, PPR16, and PPR06 bits in the priority order specification flag register
  Set the highest priority level.

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>PPR15</th>
<th>PPR05</th>
<th>Selection of INTP5 priority level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Sets level 0 (highest priority)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>レベル1を指定</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>レベル2を指定</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>レベル3を指定（低優先順位）</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>PPR16</th>
<th>PPR06</th>
<th>Selection of INTP6 priority level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>Sets level 0 (highest priority)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>レベル1を指定</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>レベル2を指定</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>レベル3を指定（低優先順位）</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Setting edge detection

- External interrupt rising edge enable register (EGP0)
  Enable INTP5 and INTP6 edge detection.
- External interrupt falling edge enable register (EGN0)
  Enable INTP6 edge detection.

Symbol: EGP0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EGP6</td>
<td>EGP5</td>
<td>EGP4</td>
<td>EGP3</td>
<td>EGP2</td>
<td>EGP1</td>
<td>EGP0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Symbol: EGN0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EGN6</td>
<td>EGN5</td>
<td>EGN4</td>
<td>EGN3</td>
<td>EGN2</td>
<td>EGN1</td>
<td>EGN0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits 6 and 5

<table>
<thead>
<tr>
<th>EGPn</th>
<th>EGNn</th>
<th>Selection of valid edge of INTPn pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>エッジ検出禁止</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>立ち下がりエッジ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Rising edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Both of rising and falling edges</td>
</tr>
</tbody>
</table>

Setting multiplexed ports

- PM3.3 and PM3.2 bits in port mode register3 (PM3)
  Turn the output buffer off.

Symbol: PM3

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PM33</td>
<td>PM32</td>
<td>PM31</td>
<td>PM30</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Bits 3 and 2

<table>
<thead>
<tr>
<th>PM3n</th>
<th>P3n pin I/O mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>出力モード(出力ポートとして機能(出力バッファ・オン))</td>
</tr>
<tr>
<td>1</td>
<td>Input mode (functions as an input port (output buffer turned off).)</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
5.7.8 Main Process

Figure 5.9 shows the flowchart of the main process.

![Figure 5.9 Main Process](image)

Make initial settings necessary for the main processing.

Polls the I2C communication end interrupt request, and if communication is complete, perform necessary processing.

Check the stop condition with the variable `g_stop_det`.

Variable `g_stop_det ← 0x00`: Clear the detection flag.

Update the LED lighting data on detection of the stop condition.

Read reception data from the master. Polls the I2C communication completion interrupt request, and if communication is complete, perform necessary processing. Perform LED lighting processing for the read data.

Set the A/D conversion result in the I2C transmission buffer.

Variable `ad_data ← sum_data[i] >> 4`: Average the 16 conversion data.

Transfer averaged data to the I2C transmission buffer. Polls the I2C communication completion interrupt request, and if communication is complete, perform necessary processing.
5.7.9 R_MAIN_UserInit Process

Figure 5.10 shows the flowchart of the R_MAIN_UserInit process.

![Flowchart of R_MAIN_UserInit Process](image)

- **R_MAIN_UserInit()**
  - Initial setting for A/D conversion
    - R_ADC_Init()
  - Initialization for LED lighting control
    - R_LED_Init()
  - Initialization for I2C control
    - R_I2C_A0_Init()
  - Start A/D conversion
    - R_ADC_Start()
  - Start TM03
    - R_TM03_Start()
  - EI()

- **return**

Initiate parameters for controlling A/D conversion.
Initiate parameters for controlling LED lighting.
Initiate parameters for controlling I2C communication.
Enable A/D converter operation.
Start timer 03 for controlling LED lighting.
Enable vector interrupts.
5.7.10 Initial Setting of A/D Conversion

Figure 5.11 shows the flowchart for making A/D conversion initial settings.

```
R_ADC_Init ()

// Initialize conversion data storage pointer.
Pointer gp_set_pt ← &g_conv_data[0][0]

// Initialize sum storage pointer.
Pointer gp_sum_pt ← &g_sum_data[0]

// Clear A/D conversion end flag.
Variable g_adc_end ← FALSE (0)

// Start A/D conversion.
R_ADC_Start ()
```

Figure 5.11 A/D Conversion Initial Settings

5.7.11 Starting A/D Conversion

Figure 5.12 shows the flowchart for starting A/D conversion.

```
R_ADC_Start ()

// Clear A/D conversion interrupt request flag.
ADIF bit ← FALSE (0) : Clear the interrupt request.

// Cancel A/D conversion interrupt mask.
ADMK bit ← INT_ENABLE (0) : Clear the interrupt request mask.

// Start A/D conversion.
ADCS bit ← TRUTH (1) : Enable A/D conversion operation.
```

Figure 5.12 Starting A/D Conversion
Setting A/D conversion interrupts

- ADIF bit in the interrupt request flag register 1H (IF1H)
  Clear the interrupt request.
- ADMK bit in the interrupt request mask flag register 1H (MK1H)
  Cancel the interrupt request mask.

Bit 0

<table>
<thead>
<tr>
<th>ADIF</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt request signal has not been generated.</td>
</tr>
<tr>
<td>1</td>
<td>割り込み要求信号が発生し、割り込み要求状態</td>
</tr>
</tbody>
</table>

Bit 0

<table>
<thead>
<tr>
<th>ADMK</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
</tr>
<tr>
<td>1</td>
<td>割り込み処理禁止</td>
</tr>
</tbody>
</table>

Starting A/D conversion

- A/D converter mode register 0 (ADM0)
  Enable A/D converter operation.

Symbol: ADM0

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCS  ADMD  FR2  FR1  FR0  LV1  LV0  ADCE</td>
</tr>
<tr>
<td>1   1   1   0   1   0   0   1</td>
</tr>
</tbody>
</table>

Bit 7

<table>
<thead>
<tr>
<th>ADCS</th>
<th>Control of A/D conversion operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>変換動作停止</td>
</tr>
<tr>
<td>1</td>
<td>Enables conversion operation.</td>
</tr>
</tbody>
</table>

Note: For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
5.7.12 A/D Conversion End interrupt Process

Figure 5.13 shows the flowchart of the A/D conversion end interrupt process.

```plaintext
r_adc_interrupt()
  Acquire A/D conversion result.
  Variable data_work ← ADCR: Read conversion results.
  Correct A/D conversion result sum.
  Variable *gp_sum_pt ← Variable *gp_sum_pt - *gp_set_pt:
  Subtract the oldest data from the sum.
  Store A/D conversion result sum.
  Variable *gp_set_pt ← data_work: Store the conversion result.
  Add A/D conversion result to sum.
  Variable *gp_sum_pt ← Variable *gp_sum_pt + data_work:
  Add the new conversion result.
  Pointer gp_sum_pt ← Pointer gp_sum_pt + 1
  Pointer gp_set_pt ← Pointer gp_set_pt + 1
  Update pointer.
  On completion of processing for 4 channels, rewind the pointer to the top.
  Pointer gp_sum_pt ← &g_sum_data[0]
  No
  Storage buffer end?
  If the storage pointer points to the buffer end, move the pointer to the top.
  Pointer gp_set_pt ← &g_conv_data[0][0]
  Yes
  Initialize the sum pointer.
  No
  Yes
  Set A/D conversion end flag.
  Variable g_adc_end ← TRUTH (1)
  return
```

**Figure 5.13 A/D Conversion End interrupt Processing**

**Acquiring A/D conversion results**
- 12-bit A/D conversion result register (ADCR)
  Read the A/D conversion results.

**Symbol: ADCR**

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>bit11</td>
<td>bit10</td>
<td>bit9</td>
<td>bit8</td>
<td>bit7</td>
<td>bit6</td>
<td>bit5</td>
<td>bit4</td>
<td>bit3</td>
<td>bit2</td>
</tr>
</tbody>
</table>
```
5.7.13 Initializing LED Lighting
Figure 5.14 shows the flowchart for initializing the LED lighting.

```plaintext
R_LED_Init()
  Clear lighting data.
  i=0, DATA_NUMBER, +1
  Clear LED lighting data.
  Variable g_sel_data ← 0x00 : Set data 0 to be displayed.
  Clear lighting data.
  Select data 0.
  Variable g_disp_timing ← 0x00 : Set the upper nibble to be displayed.
  Initialize lighting timing.
  return
```

Figure 5.14 Initialization for LED Lighting

5.7.14 Starting TM03
Figure 5.15 shows the flowchart for starting the TM03.

```plaintext
R_TM03_Start()
  Clear TM03 interrupt request flag.
  TMIF03 bit ← FALSE (0) : Clear the interrupt request.
  Cancel TM03 interrupt mask.
  TMMK03 bit ← INT_ENABLE (0) : Clear the interrupt request mask.
  Start TM03.
  TS0 register ← Set bit 3 to 1 : Enable TM03 operation.
  return
```

Figure 5.15 Starting TM03

Setting TM03 interrupt
- TMIF03 bit in interrupt request flag register 1L (IF1L)
  Clear the interrupt request.
- TMMK03 bit in interrupt request mask flag register 1L (MK1L)
  Cancel the interrupt request mask.
5.7.15 Setting LED Light-Emitting Data

Figure 5.16 shows the flowchart for setting the LED light-emitting data.

```
R_LED_DispData()

Set lighting data.
```

Variable `g_disp_data_bf ← Lighting data`

```
return
```

Figure 5.16 Setting LED Light-Emitting Data

---

**Note:** For details of register settings, refer to the RL78/I1D User's Manual: Hardware.
5.7.16 5-ms Interval Timer Interrupt Process
Figure 5.17 shows the flowchart of the 5-ms interval timer interrupt process.

![Flowchart of 5-ms Interval Timer Interrupt Process](Image)

**r_tau0_channel3_interrupt ()**

- Enable multiple interrupts.
- Turn LED off.
- Acquire lighting data.

Timing 1?
- Yes
  - Set lighting data.
  - Light upper nibble.

Timing 2?
- No
  - Set lighting data.
  - Light lower nibble.

Update the timing.

SW timing?
- Yes
  - Initialize the timing.

SW On?
- Yes
  - Select data 2.

- No
  - Select data 1.

IE bit ← 1 : Enable vector interrupts.
P6 register ← Set bits 1 and 0.
Variable disp_work ← Lighting data

Check the lower/upper lighting timing.
P0 register ← disp_work >> 4 : Set the upper nibble.
P6.1 bit ← 0 : Light the upper nibble.
P0 register ← disp_work & 0x0F : Set the lower nibble.
P6.0 bit ← 0 : Light the lower nibble.

Variable g_disp_timing ← g_disp_timing + 1

Check the SW state with P13.7 and select the lighting data.
Set data pointer to data 2.
Set data pointer to data 1.

Figure 5.17 5-ms Interval Timer Interrupt Process
5.7.17 Initializing I2C Communication

Figure 5.18 shows the flowchart for initializing the I2C communication.

Variable \( g\_ram\_area[i] \leftarrow i \)  
: Write 0x00 to 0x7F patterns in RAM area as the I2C slave.

Variable \( g\_low\_data\_index \leftarrow 0 \)  
: Process upper byte next.

Variable \( g\_stop\_det \leftarrow 0 \)  
: Not detected

Variable \( g\_IIC\_IF \leftarrow 0 \)  
: No interrupt request
5.7.18 Checking I2C Communication State
Figure 5.19 shows the flowchart for checking the I2C communication state.

![Flowchart for checking I2C communication state](image)

```plaintext
R_IICA0_Status ()
Check I2C interrupt requests.
r_iic_int_chk ()
Copy the status.
return
```

Check the I2C interrupt requests and perform transfer processing if necessary.

Variable “status” ← Variable g_status : Copy the status.

Return value ← Variable “status”

Figure 5.19 Checking I2C Communication State

5.7.19 Reading I2C Reception Data
Figure 5.20 shows the flowchart for reading I2C reception data.

![Flowchart for reading I2C receive data](image)

```plaintext
R_IICA0_Get ()
Check I2C interrupt requests.
r_iic_int_chk ()
Read reception data.
return
```

Check the I2C interrupt requests and perform transfer processing if necessary.

Variable rx_data ← g_rx_data[(ptr & RX_LIMIT)]

Return value ← Variable rx_data

Figure 5.20 Reading I2C Receive Data
5.7.20 Setting Data in I2C Transmission Buffer

Figure 5.21 shows the flowchart for setting data in the I2C transmission buffer.

```
R_IICA0_Put()

Check I2C interrupt requests.
r_iic_int_chk()

Set write pointer.

Set transmission data.

return
```

Figure 5.21 Setting Data in I2C Transmit Buffer

5.7.21 Checking I2C Communication End interrupt Request

Figure 5.22 shows the flowchart for checking the I2C communication end interrupt request.

```
r_iic_int_chk()

Read I2C status.
_R_IICSS_Status()

Interrupt requested?

No

Yes

I2C interrupt processing
r_iic_request()

Clear transmission/reception end flag.

return
```

Variable “status” ← I2C communication status (g_IICS)

If _g_IIC_IF is 0x01, perform I2C interrupt processing.

Variable _g_IIC_IF ← 0 : Clear the transmission/reception end flag.

Set the variable “status” as the return value.

Figure 5.22 I2CA0 Communication End interrupt Reception Process
5.7.22 I2C Communication End interrupt Process

Figures 5.23 to 5.26 show the flowcharts of the I2C communication end interrupt process.

![Flowchart of I2C Communication End interrupt Process](image-url)

**Figure 5.23** I2C Communication End interrupt Transmission Process (1/4)

1. **r_iic_request ()**
   - Set I2C communication status.

2. **ACK response?**
   - No
     - Set I2C communication status.
   - Yes
     - Branch according to the selected slave address.

3. **ID**
   - 1
     - Branch according to the communication mode.
       - Continue transmission
         - Start transmission
           - Read A/D conversion result.
           - Set upper data in transmission buffer.
           - Start data transmission.
             - _Tx_data_sub ()
           - Set lower byte in buffer.
           - Set data pointer to 1.
           - Set lower byte flag to 1.
           - return
         - Continue reception
           - Processing to start A/D conversion result transmission
             - Variable datawork ← Variable g_low_data[0] : G_channel [0]
             - Variable _g_IICA ← datawork >> 8
             - Start transmitting data of variable _g_IICA.
             - Variable g_low_data_temp ← datawork & 0xFF
             - Variable g_ptrx_data ← 0x01 : Process channel 1 next.
             - Variable g_low_data_index ← TRUTH : Lower data exist.
6. **= 2**
   - Branch according to the selected slave address.

7. **= 3**
   - Branch according to the communication mode.
     - Continue reception
       - Processing for slave address ID=1 (A/D conversion result and LED lighting information)
         - Variable slave ← Slave address ID
         - Variable mode ← TRC and STD bits
         - Start transmission
           - _Tx_data_sub ()
           - Set lower byte in buffer.
           - Set data pointer to 1.
           - Set lower byte flag to 1.
           - return
         - Continue reception
           - Variable “slave” ← Slave address ID
           - Variable “mode” ← TRC and STD bits
           - Processing to start A/D conversion result transmission
             - Variable datawork ← Variable g_travel_data[0] : G_channel [0]
             - Variable _g_IICA ← datawork >> 8
             - Start transmitting data of variable _g_IICA.
             - Variable g_low_data_temp ← datawork & 0xFF
             - Variable g_ptrx_data ← 0x01 : Process channel 1 next.
             - Variable g_low_data_index ← TRUTH : Lower data exist.

8. **= 4**
   - Branch according to the communication mode.

9. **Others**
   - Branch according to the communication mode.

Variable g_status ← Variable _g_IICS : Read I2C status.

If ACK is detected, perform the processing.

Variable “slave” ← Slave address ID
Variable “mode” ← TRC and STD bits
Figure 5.24  I2C Communication End interrupt Process (2/4)
Set data in transmission buffer.

Start data transmission.

Variable \( \_g\_IICA \) ← \( g\_ram\_area\[g\_ptrx\_data2\] \\
\( g\_ptrx\_data2 \) ← \((g\_ptrx\_data2 +1) \) & RAM_MASK

Update address register.

return

Set address register flag.

Start data reception.

Variable \( g\_regadr \) ← TRUTH

Start data reception (address register).

return

Is address register to be set?

No

Yes

Set address register.

Clear address register flag.

Set reception data in RAM data area.

Update address register.

Start data reception.

return

Determine address register or data according to the variable \( g\_regadr \) value.

\( g\_ptrx\_data2 \) ← Variable \( g\_IICA \) & 0x7F

Set reception data in the address register.

Variable \( g\_regadr \) ← FALSE

\( g\_ram\_area \[g\_ptrx\_data2\] \) ← Variable \( g\_IICA \)

\( g\_ptrx\_data2 \) ← \((g\_ptrx\_data2 +1) \) | 0x7F

Start next data reception.

Figure 5.25  I2C Communication End interrupt Process (3/4)
Figure 5.26  I2C Communication End interrupt Process (4/4)

Describe processing according to the communication mode.

- **E** Mode
  - Start transmission
  - Branch processing according to the communication mode.
  - Continue transmission
  - Start reception
  - Continue reception

  Processing for slave address ID=3 (Prepare the frame only.)

- **F** Mode
  - Start transmission
  - Branch processing according to the communication mode.
  - Continue transmission
  - Start reception
  - Continue reception

  Processing for slave address ID=4 (Prepare the frame only.)

- **G**
  - Clear communication status.
  - Cancel the wait.
  - _Tx_end_sub()

  Variable g_status ← 0x00

  Cancel the I2C bus wait and exit the communication.

  return

- **H**
  - Clear communication status.
  - Cancel the wait.
  - _Tx_end_sub()

  Variable g_status ← 0x00

  Cancel the I2C bus wait and exit the communication.

  return

[Remarks] Only frames are provided for slave addresses ID=3 and ID=4 processes in this sample code. Whether _SADDR3_ and _SADDR4_ are previously defined or not determines whether to include the corresponding processes as targets of build between #ifdef and #endif. To use these parts, remove the comment-out for SADDR3 and SADDR4 definitions.
5.7.23 Initializing I2C (Assembler Section)

Figure 5.27 shows the flowchart for initializing the I2C for the assembler section.

Figure 5.27  Initializing I2C (Assembler Section)

Clear output latch.

Clear communication status.

Disable SCL edge detection interrupt.

Enable SCL rising edge detection.

Enable SDA edge detection.

Enable SDA edge detection interrupt request.

Clear SCL edge detection interrupt requests.

Transfer ACK response data.

j=0, 16, +1

Copy ROM data to RAM.

Transfer ACK response data.

RET

Set 0 to the output latch of P3.2 and P3.3 used by the SDA and SCL signals.

Variable g_I2CS ← 0

PMK5 bit (DIS_INTSCL) ← 1 : Mask the interrupt.

EGP0.5 bit (EPG_SCL) ← 1 : Enable rising edge detection.

EGN0.5 bit (ENG_SCL) ← 0 : Disable falling edge detection.

EGN0.6 bit ← 1 : Enable falling edge detection.

EGP0.6 bit ← 1 : Enable rising edge detection.

PIF6 bit (RQ_INTSDA) ← 0 : Clear the interrupt request.

PMK6 bit (DIS_INTSDA) ← 0 : Enable the interrupt.

PIF5 bit (RQ_INTSCL) ← 0 : Clear the interrupt request.

Copy the initial value of ACK response data for each slave address ID to the working RAM.

Clearing output latch

- Port register 3 (P3)

Clear P3.3 and P3.2.

Symbol: P3

\[
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & x & x & x \\
\end{array}
\]

Bits 3 and 2

<table>
<thead>
<tr>
<th>P3n</th>
<th>Data written to output latch</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sets 0.</td>
</tr>
<tr>
<td>1</td>
<td>1 を設定</td>
</tr>
</tbody>
</table>

For details of register settings, refer to the RL78/I1D User’s Manual: Hardware.
Disabling SCL edge detection interrupt
- PMK5 bit in the interrupt mask flag register 0L (MK0L)
  Mask the INTP5 interrupt request.

Bit 7

<table>
<thead>
<tr>
<th>PMK5</th>
<th>Control of interrupt processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>割り込み処理許可</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>

Setting SCL and SDA edge detection
- External interrupt rising edge enable register (EGP0)
  Set the EGP6 and EGP5 bits.
- External interrupt falling edge enable register (EGN0)
  Set the EGN6 bit and clear the EGN5 bit.

Symbol: EGP0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EGP6</td>
<td>EGP5</td>
<td>EGP4</td>
<td>EGP3</td>
<td>EGP2</td>
<td>EGP1</td>
<td>EGP0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Symbol: EGN0

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EGN6</td>
<td>EGN5</td>
<td>EGN4</td>
<td>EGN3</td>
<td>EGN2</td>
<td>EGN1</td>
<td>EGN0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bits 6 and 5

<table>
<thead>
<tr>
<th>EGPn</th>
<th>EGNn</th>
<th>Selection of valid edge of INTPn pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>エッジ検出禁止</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>立ち下がりエッジ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Rising edge (INTP5: SCL edge)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Rising and falling edges (INTP6: SDA edge)</td>
</tr>
</tbody>
</table>

Clearing interrupt requests
- Interrupt request flag register (IF0L, IF0H)
  Clear the INTP5 and INTP6 interrupt requests.

Symbol: IF0L

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIF5</td>
<td>PIF4</td>
<td>PIF3</td>
<td>PIF2</td>
<td>PIF1</td>
<td>PIF0</td>
<td>LVIF</td>
<td>WDTIIF</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Symbol: IF0H

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTITIF</td>
<td>TMIF00</td>
<td>SREIF0</td>
<td>0</td>
<td>0</td>
<td>SRIF0</td>
<td>STIF0</td>
<td>PIF6</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit n

<table>
<thead>
<tr>
<th>PIFn</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>割り込み要求信号が発生し、割り込み要求状態</td>
</tr>
<tr>
<td>1</td>
<td>割り込み要求信号が発生し、割り込み要求状態</td>
</tr>
</tbody>
</table>

PMK5 bit in the interrupt mask flag register 0L (MK0L)
Mask the INTP5 interrupt request.
Enabling SDA edge detection interrupts
- PMK6 bit in the interrupt mask flag register 0H (MK0H)
  Cancel the INTP6 interrupt request mask.
Symbol: MK0H

<table>
<thead>
<tr>
<th>Bit</th>
<th>Control of interrupt processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
</tr>
<tr>
<td>1</td>
<td>割り込み処理禁止</td>
</tr>
</tbody>
</table>

### 5.7.24 Reading I2C Communication Status

Figure 5.28 shows the flowchart for reading the I2C communication status.

```plaintext
__R_IICSS_Status

Set I2C communication status.

Register A ← Variable g_IICS : Set the communication status.

RET
```

**Figure 5.28** Reading I2C Communication Status

### 5.7.25 Setting ACK Response

Figure 5.29 shows the flowchart for setting ACK responses.

```plaintext
__set_ACKE_table

Mask unused bits.

Register A ← A & 0x1F

Extract ID and disable/enable.

Shift data in register A to right by 1 bit.

Set ID.

Register B ← Register A

Create ACK response data.

Register A ← 0x00 : For disabling
Register A ← 0xF : For enabling

Set ACK response data.

Variable g_ACKE_tbl[B] ← Register A

return
```

**Figure 5.29** Setting ACK Response
5.7.26 Reading ACK Responses

Figure 5.30 shows the flowchart for reading the ACK responses.

```
__get_ACKE_table
Extract slave address ID.
Set ID.
Read out ACK response.
```

Figure 5.30 Reading ACK Responses

5.7.27 SCL Edge Detection Interrupt Entry Process

Figure 5.31 shows the flowchart of the SCL edge detection interrupt entry process.

```
r_iic_SCL_interrupt
Change register banks.
Read out SCL and SDA signal state.
Read out processing address.
```

Figure 5.31 SCL Edge Detection Interrupt Entry Process

This section only shows the entry processing of the SCL edge detection interrupt. For the description of actual processing of the interrupt, see 5.8.29, SCL Edge Detection Interrupt Processing.

Reading Out SCL and SDA Signals
- Port register 3 (P3)
  Read out SCL and SDA signals.

Symbol: P3
```
<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCL</td>
<td>SDA</td>
<td>P31</td>
<td>P30</td>
</tr>
</tbody>
</table>
```

5.7.28 SDA Edge Detection Interrupt Process

Figures 5.32 to 5.33 show the flowcharts of the SDA edge detection interrupt process.

---

**Figure 5.32 SDA Edge Detection Interrupt Process (1/2)**

- Set RB3 as the register bank to be used.
- Change register banks.
- Read SCL and SDA signal state.
- Clear unused bits.
- If bit 3 is 1 and bit 2 is 0 (SDA falling), start condition is detected.
  - PIF5 bit (RQ_INTSCL) ← 0: Clear the interrupt request.
  - F_STD bit of variable g_IICS ← 1: Set to perform detection.
  - Variable next_proc ← #LOWW wait_SA
  - Enable SCL interrupt request.
  - PMK5 bit (DIS_INTSCL) ← 0: Cancel the mask.

- Start condition?
  - Yes
    - Enable SCL edge detection.
    - Clear SCL edge detection interrupt request.
    - Set to detect start condition.
    - Set SCL interrupt processing address.
    - Enable SCL interrupt request.
    - RETI

- No
  - Stop condition?
    - Yes
      - Set stop condition detection.
      - Disable SCL edge detection interrupt.
      - Initialize SCL edge detection setting.
      - RETI
    - No
      - Set to detect start condition.
      - Set SCL interrupt processing address.
      - Enable SCL interrupt request.
      - RETI
The following control registers are used for both the SCL edge detection interrupt processing and SDA edge detection interrupt processing. This section describes these control registers collectively.

**Reading SCL and SDA signals**
- Port register 3 (P3)

  Read SCL and SDA signals.

  Symbol: P3

  Symbol: EGP0

  Symbol: EGN0

  Symbol: EGP5, EGN5

  Table: Selection of valid edge of INTP5 pin

**Figure 5.33 SDA Edge Detection Interrupt Process (2/2)**
Controlling SCL and SDA edge detection interrupts

- Interrupt mask flag register (MK0L, MK0H)
  
  Control interrupt requests.

- Interrupt request flag register (IF0L, IF0H)
  
  Control interrupt requests.

Symbol: MK0L

<table>
<thead>
<tr>
<th>PMK5</th>
<th>PMK4</th>
<th>PMK3</th>
<th>PMK2</th>
<th>PMK1</th>
<th>PMK0</th>
<th>LVIMK</th>
<th>WDTIMK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Symbol: MK0H

<table>
<thead>
<tr>
<th>RTITMK</th>
<th>TMMK00</th>
<th>SREM0</th>
<th>STM0</th>
<th>SRM0</th>
<th>PMK6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>0/1</td>
</tr>
</tbody>
</table>

Bit n

<table>
<thead>
<tr>
<th>PMKn</th>
<th>Control of interrupt processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enables interrupt processing.</td>
</tr>
<tr>
<td>1</td>
<td>Disables interrupt processing.</td>
</tr>
</tbody>
</table>

Symbol: IF0L

<table>
<thead>
<tr>
<th>PIF5</th>
<th>PIF4</th>
<th>PIF3</th>
<th>PIF2</th>
<th>PIF1</th>
<th>PIF0</th>
<th>LVIF</th>
<th>WDTIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Symbol: IF0H

<table>
<thead>
<tr>
<th>RTITIF</th>
<th>TMIF00</th>
<th>SREIF0</th>
<th>SRIF0</th>
<th>STIF0</th>
<th>PIF6</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Bit n

<table>
<thead>
<tr>
<th>PIFn</th>
<th>Interrupt request flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Interrupt request signal has not been generated.</td>
</tr>
<tr>
<td>1</td>
<td>割り込み要求信号が発生し，割り込み要求状態</td>
</tr>
</tbody>
</table>
5.7.29 SCL Edge Detection Interrupt Process

Figures 5.34 to 5.52 show the flowcharts of SCL edge detection interrupt process.

(0) Waiting for stop/start condition

- **SCL_High**
  - Clear SDA edge detection interrupt request.
  - PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
  - Enable SDA edge detection interrupt.
  - PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
  - Disable SCL edge detection interrupt.
  - PMK5 (DIS_INTSCL) bit ← 1 : Set the interrupt mask.

  **RETI**

**Figure 5.34 Waiting for Stop/Start Condition**

(1) Waiting for slave address reception start (sequence (4))

- **wait_SA**
  - Disable SDA edge detection interrupt.
  - PMK6 (DIS_INTSDA) bit ← 1 : Set interrupt mask.
  - Disable SCL falling edge detection.
  - EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.
  - Set initial value for reception data.
  - Variable g_IICA ← 0x01 : Initial value for reception
  - Set SCL interrupt processing address.
  - Variable next_proc ← #LOWW capt_data : Address reception processing

  **RETI**

**Figure 5.35 Waiting for Slave Address Reception Start**
(2) Receiving slave address (sequence (5))

- **sequence(5)**
  - **capt_data**
    - Clear stop condition detection.
    - Acquire received data bits.
    - Store data being received.
    - **Slave address continued?**
      - **No**
        - **RETI**
      - **Yes**
        - **Set communication direction flag.**
        - **Check address.**
          - **Addresses match?**
            - **No**
            - **Set address match flag.**
            - **Set ACK response flag.**
            - **A**
              - **send_ack**
                - Disable SCL rising edge detection.
                - **Check ACK response flag.**
                - **Wait for SCL falling.**
                - **sequence(6)**
                  - **A**
                    - **send_ack**
                      - Disable SCL rising edge detection.
                      - **Check ACK response flag.**
                      - **Wait for SCL falling.**
                      - **sequence(7)**
                        - **Start ACK response.**
                          - Enable SCL falling edge detection.
                          - **Set SCL interrupt processing address.**
                          - **RETI**

- **F_SPD bit ← 0**: Clear the detection flag.
- **CY flag ← D_SD bit**: Acquire SDA data bits.
- **Register A ← Variable g_IICA**: Read data being shifted.
- **Register A ← CY flag**: Shift in received data rightmost bit first.
- **Variable g_IICA ← Register A**: Store data being shifted.
- If the slave address reception continues, end the interrupt processing.

- **F_TRC bit ← CY flag**: Set the communication direction flag.
- **Shift the register A to right by 1 bit and compare the register value**
  **and the upper 7 bits of its own slave address, referring**
  **to the slave address table. If they do not match (if the table-**
  **reference result is 0), the processing proceeds to the non-**
  **selection (not_selected) processing.**

- **Set slave address ID in the upper 4 bits of g_IICS.**
- Read out the ACK response of the slave address ID from the**
  **table and set the value to variable g_ACKE.**
- **ACK response start processing**
- **EGP5 (EPG_SCL) bit ← 0**: Disable rising edge detection.
- **F_ACKD bit ← ACK response flag**
  **CY flag ← Invert the ACK response flag.**
- **Poll P_SCL bit and wait for SCL falling.**
- **PM_SDA bit ← CY**: ACK/NACK response
- **EGN5 (ENG_SCL) bit ← 1**: Enable falling edge detection.
- Variable next_proc ← #LOWW ack_end : ACK response completion

Figure 5.36 Receiving Slave Address
(3) Non-Selection Processing (sequence(6'))

- Clear communication status.
  - Variable `g_IICS ← 0x00`: Clear the I2C status.
- Set SCL interrupt processing address.
  - Variable `next_proc ← #LOWW addr_end`: Address end processing (sequence(21))

Figure 5.37 Non-Selection Processing

(4) ACK Response End Processing (sequence(8))

- Request the master to wait.
  - PM_SCL bit ← 0: Set SCL signal to 0.
- End ACK response.
  - PM_SDA bit ← 1: Re-set SDA signal to input.
- Enable SCL rising edge detection.
  - EGP5 (EPG_SCL) bit ← 1: Enable rising edge detection.

Transmission?

- Yes: Perform transmission or reception according to F_TRC bit value.
- No: C

Figure 5.38 ACK Response End Processing
(5) Reception Completion (Operation Start Request) Processing (sequence(8)‘)

After completion of one-byte data reception, set the communication end flag to the upper software and end processing.

![Diagram](image)

- **rx_start**
  - Set reception data.
  
- **TRG_INTIIC**
  - Set communication status.
  
- **Set transmission/reception end flag.**
  
- **RETI**

Notify the upper process that the slave address reception has been completed and that slave has been selected in reception mode, and then continue reception processing.

Variable `__g_IICA ← Variable g_IICA` : Set reception data.

Variable `__g_IICS ← Variable g_IICS` : Set communication status.

Variable `__g_IIC_IF ← 0x01`  
Transfer reception data and communication status to C-description reception processing part (interface processing to C-description part).

Figure 5.39  Reception Completion Processing
(6) Receiving Data (sequence(9))

After SCL signal rising at the first bit of the next data, detect SDA state change (restart or stop condition) and prepare for the communication.

- F_STD bit ← 0 : Clear the STD flag.
- F_ACKD bit ← 0 : Clear the ACKD flag.

Variable next_proc ← #LOWW rx_data2 : Data reception processing (sequence(10))

- EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.
- PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
- PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.

CY flag ← D_SDA bit : Acquire SDA data bits.
Register A ← Variable g_IICA : Read data being shifted
Register A ← CY flag : Shift in received data rightmost bit first.
Variable g_IICA ← Register A : Store data being shifted

If the 8-bit reception is completed, the processing proceeds to ACK response processing (send_ack). If reception continues, the processing exits the interrupt processing and enters the next bit reception wait state.

(7) Starting Reception Operation (sequence(10))

After SCL signal falling at the first bit, start data communication operation (restart or stop condition is not detected).

- PMK6 (DIS_INTSDA) bit ← 1 : Set the interrupt mask.
- EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.

Variable next_proc ← #LOWW rx_data3 : Data reception processing (sequence(11))
(8) Requesting Transmission Operation Start (sequence(14))

```
sequence(14)

<table>
<thead>
<tr>
<th>D</th>
<th>Data transmission processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set SCL interrupt processing address.</td>
<td>Variable next_proc ← #LOWW tx_start : Data transmission processing (sequence (15))</td>
</tr>
<tr>
<td>Initialize transmission bit counter.</td>
<td>Variable bit_count ← 0x07: Set bit count – 1.</td>
</tr>
<tr>
<td>TRG_INTIIC</td>
<td>Set the communication status, set the transmission/reception end flag, and end the processing.</td>
</tr>
</tbody>
</table>
```

Figure 5.42  Requesting Transmission Operation Start

(9) Starting Transmission Operation (sequence(15))

```
sequence(15)

<table>
<thead>
<tr>
<th>tx_start</th>
<th>EGP5(EPG_SCL) bit ← 0 : Disable rising edge detection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable SCL rising edge detection.</td>
<td>Variable g_IICS: Clear variable g_IICS bits other than COI and TRC bits.</td>
</tr>
<tr>
<td>Set I2C communication status.</td>
<td>Variable next_proc ← #LOWW tx_data : Data transmission processing (sequence(16))</td>
</tr>
<tr>
<td>Set SCL interrupt processing address.</td>
<td></td>
</tr>
<tr>
<td>RETI</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 5.43  Starting Transmission Operation
(10) Transmission Processing (sequence(16))

- tx_data
- sequence(16)
- Prepare transmission data.
- Transmit data.
- Store the remaining data.
- Count the remaining bits.
- Transmission completed?
  - No
  - Store the remaining data.
  - CY flag ← A.7 bit: Shift data from A.7 to CY.
  - PM_SDA bit ← CY flag: Transmit the bits.
  - Variable g_IICA ← Register A: Store the remaining bits.
  - Variable bit_count ← bit_count − 1: Count the number of remaining bits.
  - Variable next_proc ← #LOWW tx_data_end: Data transmission processing (sequence(17))
  - Yes
  - Set SCL interrupt processing address.
  - RETI

Figure 5.44 Transmission Processing

(11) Transmission End Processing (sequence(17))

- tx_data_end
- sequence(17)
- Disable SDA output.
- Enable SCL rising edge detection.
- Set SCL interrupt processing address.
- PM_SDA bit ← 1: Re-set the port to input.
- EGP5 (EPG_SCL) bit ← 1: Enable rising edge detection.
- Variable next_proc ← #LOWW ack_chk: Data transmission processing (sequence(18))
- RETI

Figure 5.45 Transmission End Processing
(12) Checking ACK (sequence(18))

ack_chk

sequence(18)

NACK response? Yes

Set SCL interrupt processing address.

Clear SDA edge detection interrupt request.

Enable SDA edge detection interrupt.

RETI

NACK response from the master indicate the communication completion.

Variable next_proc ← #LOWW nack_end : To the communication exit processing (sequence(20))

PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.

PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.

The processing proceeds to the start condition/stop condition detection wait state.

ACK response from the master instructs to continue communication.

Variable next_proc ← #LOWW nack_end : Next data transmission processing (sequence(14))

(RETI)

ack_det

sequence(19)

Set SCL interrupt processing address.

RETI

Figure 5.46 Checking ACK

(13) NACK Completion Processing (sequence(20))

nack_end

sequence(20)

Request the master to wait.

Disable SCL falling edge detection.

Disable SCL interrupt.

TRG_INTIIC

PM_SCL bit ← 0 : Set SCL signal to 0.

EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.

PMK5 (DIS_INTSCL) ← 1 : Disable INTP5.

Set the communication status, set transmission/reception end flag, and end the processing.

Figure 5.47 NACK Completion Processing
(14) Non-Selection Processing (1/2) (sequence(21))

Even when the slave itself is not selected and it does not perform data processing, the slave needs to monitor the communication state by counting the number of SCLs.

In the non-selection processing below, only the SCL counting is performed to reduce the CPU processing during the not-selected state (this operation is referred to as skip-read).

![Diagram of Non-Selection Processing (1/2) with sequence(21) and sequence(22) steps]

- **Sequence (21)**
  - **addr_end**
  - **ACK response?**
    - No
    - Set SCL interrupt processing address.
    - Set skip-read SCL count.
    - RETI
  - **ACK response from another slave.**
    - Variable `next_proc ← #LOWW next_data : Next data transmission processing (sequence (28))`
    - Variable `g_work_count ← 0x08 : Set the skip-read count.`

- **Sequence (22)**
  - **no_ACK**
    - Set skip-read SCL count.
    - **no_ACK1**
      - Set SCL interrupt processing address.
      - Clear SDA edge detection interrupt request.
      - Enable SDA edge detection interrupt.
      - Enable SCL falling edge detection.
      - RETI
    - Variable `g_work_count ← 0x02 : Set the skip-read count.`

- **Sequence (24)**
  - **no_ACK2**
    - Set SCL interrupt processing address.
    - Clear SDA edge detection interrupt request.
    - Enable SDA edge detection interrupt.
    - Enable SCL falling edge detection.
    - RETI
  - Variable `g_work_count ← 0x02 : Set the skip-read count.`

- **Sequence (23)**
  - Variable `next_proc ← #LOWW no_ACK2 : Skip-read processing (sequences (23)(25))`

- **Sequence (25)**
  - PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.
  - PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.
  - EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.
(15) Non-Selection Processing (2/2) (sequences (23), (25))

![Sequence Diagram]

Figure 5.49 Non-Selection Processing (2/2)

(16) Skip-Read Processing 2 (sequence(26))

![Sequence Diagram]

Figure 5.50 Skip-Read Processing 2
(17) Skip-Read Processing 3 (sequence(28))

sequence(28)

next_data

Clear SDA edge detection interrupt request.

Enable SDA edge detection interrupt.

Enable SCL falling edge detection.

Set SCL interrupt processing address.

RET

PIF6 (RQ_INTSDA) bit ← 0 : Clear the interrupt request.

PMK6 (DIS_INTSDA) bit ← 0 : Cancel the interrupt mask.

EGN5 (ENG_SCL) bit ← 1 : Enable falling edge detection.

Variable next_proc ← #LOWW next_data2 : Skip-read processing (sequence(29))

Figure 5.51 Skip-Read Processing 3

(18) Skip-Read Processing 4 (sequence(29))

sequence(29)

next_data2

Disable SDA edge detection interrupt.

Disable SCL falling edge detection.

Set SCL interrupt processing address.

RET

PMK6 (DIS_INTSDA) bit ← 1 : Set the interrupt mask.

EGN5 (ENG_SCL) bit ← 0 : Disable falling edge detection.

Variable next_proc ← #LOWW next_data3 : Skip-read processing (sequence(26))

Figure 5.52 Skip-Read Processing 4
5.7.30 Starting Next Data Transmission
Figure 5.53 shows the flowchart for starting the next data transmission.

```
 Figure 5.53  Next Data Transmission Start Processing
```

```
5.7.31 Starting Next Data Reception
Figure 5.54 shows the flowchart for starting the next data reception.

```
 Figure 5.54  Next Data Reception Start Processing
```

```
5.7.32 Aborting Data Transmission
Figure 5.55 shows the flowchart for aborting data transmission.

```
 Figure 5.55  Data Transmission Abort Processing
```
6. I2C Bus Basics

With the I2C bus, the I2C bus master controls communication. Slaves transmit or receive data according to the instructions from the master. Slaves can only send the ACK or NACK response to the data transmitted from the master, and pull the SCL signal low to keep the master waiting for synchronization with the master. However, some masters do not support the wait function and special approach is necessary in this case.

Slaves do not necessarily always follow the master. Slaves follow the master’s instructions with respect to the communication protocol details; however, in the upper layer, the master is required to meet the slave’s specifications. This is because the functions that slaves provide to the master via the I2C bus are specified by slaves.

Therefore, it is first defined what functions to provide as the slave. The master performs communication according to the definition.

As described above, with slaves, the functions provided determine the processes for access from the I2C bus, and thus hierarchical I2C bus control with slaves is difficult, unlike with the master. The slave performs the appropriate process according to the instructions from the master.

6.1 Communication Implementation through Software

The RL78/I1D does not have the I2C communication functions corresponding to the slaves. Therefore, to connect it as the slave to the I2C bus, it is necessary to prepare the program processes using ports and external interrupts. In this case, there are some limitations on the communication speed, conditions, and signals.

Figure 6.1 shows the SCL signal standard in this application note. The values shown here must also be applied to the setup time and hold time of the start condition and stop condition.

![Example of Corresponding SCL Waveform](image)

The ports and interrupts shown in figure 6.2 are used. Note that P32 and P33 are not provided with the function to set N-ch O.D. output. The same function is implemented by setting 0 to the output latch and controlling it through PM.

![Pins Used](image)

Implementation through software requires considerable CPU power. Particularly, special consideration is required since the I2C bus state needs to be constantly monitored even when the CPU itself is not selected.

To perform communication while monitoring the I2C bus state, the appropriate response time to the signal change is important. Since the delay caused by other interrupts has a significant influence, it is necessary to give top priority to the INTP5 and INTP6 interrupts and to enable the other interrupt processes at the beginning of the process. Therefore, for the other interrupts, `enable=true` is additionally declared on `#pragma` interrupt declaration.

In addition, the I2C hardware control processing part is independent as a library, and is written in the assembly language. For convenient use, the interface part written in the C language is provided, which enables easy use of the library also from the program written in the C language.
6.2 Functions as Slaves

6.2.1 LED Display Function

As the LED display device, eight LEDs are used to display 8-bit data. Two-byte data can be displayed, and either of the 2 bytes of data can be specified according to the SW input. While SW is not being pressed, data at register address 0x00 is displayed and while SW is being pressed, data at register address 0x01 is displayed.

Data is divided into the upper 4 bits and lower 4 bits and displayed in a time-division manner. The display frequency is 10 ms. Display data from the master is fixed as display data when the master issues the stop condition. The fixed data can be displayed after 50 ms at the latest.

6.2.2 A/D Conversion Function

It is possible to convert 4-channel analog input and obtain the moving average of the 16 latest conversions. The specifications of A/D conversion are given below.

- Analog input: 4 channels (channels 0 to 3)
- Conversion method: Continuous conversion in scan mode
- Conversion resolution: 12 bits
- Conversion time: 18 μs/channel
- Buffer: 16 data/channel (128 bytes in total)

Conversion result of channel 0 is first read out, and then that of 1, 2, and 3 are read out. The upper 4 bits and lower 8 bits of the 12-bit conversion result of each channel are read out in this order. After the lower 8 bits of channel 3 are read out, channel 0 is read out as shown in figure 6.3.

![Figure 6.3  Reading A/D Conversion Results](image)

6.2.3 RAM Function

The 128-byte area is provided to temporarily store 128 bytes of data. In the initial state, data is stored at 0x00 to 0x7F. When specified with slave address 0x70, RAM can be accessed. The data is written to RAM immediately after it is received. The address is automatically updated each time RAM is accessed. Access to address 0x7F is followed by access to address 0x00.
6.3 Library Interface Specifications

The library written in the assembly language provides the following three types of interfaces.
- Transmission/reception end flag
- Stop condition detection flag
- Communication restart processing function

![Figure 6.4 I2C Bus Control Structure](image)

6.3.1 I2C Communication Flags

After completion of 1-byte data transmission/reception for the slave itself, the master is kept waiting to stop
the communication, and then the following variables/flags are set.

- Variable _g_IICA: Stores data received in reception mode.
- Variable _g_IICS: Stores communication status, like the IICS0 register.
- Variable _g_IIC_IF: Flag indicating that 1-byte communication is completed. (transmission/reception end flag)

If the upper software checks the transmission/reception end flag to find that the flag is set, it refers to the
communication status (variable _g_IICS) and executes the appropriate process.

After completing the process, the upper software calls the library functions (shown in section 6.3.2 Next
Communication Starting Functions), prepares for the next communication, and cancels the I2C bus wait state
to start the next communication.

6.3.2 Next Communication Starting Functions

The following three functions are provided to restart the I2C bus communication.
- _Rx_data_sub function: In the reception process, starts the next data reception.
- _Tx_data_sub function: In the transmission process, starts transmission of the data passed to the
  argument.
- _Tx_end_sub function: In responding to the NACK response from the master, cancels the wait state and
  withdraws from communication.

According to the communication status, one of the above functions is called.

6.3.3 Stop Condition Detection Flag

When the library written in the assembly language detects the stop condition, 0x01 is set to the variable
_g_stop_det. Unlike the I2C status (variable _g_IICS), it remains set until cleared by the upper program. This
is used in such applications that the process is started upon detection of the stop condition.

In the main process in this application note, the received data for turning on LED is sent to the program
that processes turning on of LED, and is used as a trigger to actually process turning on of LED.
6.4 Slave Address Specification

6.4.1 Slave Address Table

This library holds information of the slave addresses to use as a table. The table is referred to with the upper 7 bits of the slave address transmitted from the master.

The upper 4 bits of the obtained value is the address ID (the lower 4 bits are 0). If the obtained value is 0x00, it means that the received address is not the address of that slave, and thus that slave does not participate in communication.

If the address ID type is one of 1 to F, it means that slave has been selected. In other words, 15 independent slave addresses can be used.

Although a single address ID can be assigned to multiple slave addresses, if the same address ID is used, the same process is basically applied.

The obtained address ID is set to the upper 4 bits of the communication status (variable _g_IICS). Figure 6.5 shows the communication status structure.

<table>
<thead>
<tr>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3</td>
<td>ID2</td>
<td>ID1</td>
<td>ID0</td>
<td>F_TRC</td>
<td>F_ACKD</td>
<td>F_STD</td>
<td>F_SPD</td>
</tr>
</tbody>
</table>

Figure 6.5 Communication Status Structure

The initial ACK response values and address table are stored in r_iicss_adr.asm as constant files. Figure 6.6 shows an example of the slave address table. In this example, 0x10 (address ID is 1), 0x20 (address ID is 2), 0x30 (address ID is 3), and 0x40 (address ID is 4) are set to 0x30 (address is 0x60), 0x38 (address is 0x70), 0x40 (address is 0x80), and 0x48 (address is 0x90), respectively.

<table>
<thead>
<tr>
<th>SADR_TBL:</th>
</tr>
</thead>
<tbody>
<tr>
<td>;0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x00-0x07</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x08-0x0F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x10-0x17</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x18-0x1F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x20-0x27</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x28-0x2F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x30-0x37</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x38-0x3F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x40-0x47</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x48-0x4F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x50-0x57</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x58-0x5F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x60-0x67</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x68-0x6F</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x70-0x77</td>
</tr>
<tr>
<td>.DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x78-0x7F</td>
</tr>
</tbody>
</table>

Figure 6.6 Slave Address Table

6.4.2 ACK Response Flag

Each address ID has a flag to control the ACK response. 0x0F is set to send the ACK response, and 0x00 is set not to send the ACK response.

Figure 6.7 shows the ACK response table structure. In this example, address IDs 1 to 4 are set to send the ACK response.

| ACK_TBL: |
| ;0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F |
| .DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x00-0x07 |
| .DB 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 ; 0x08-0x0F |
6.5 Protocol for Accessing Slaves

6.5.1 Display on LED

Figure 6.8 shows how to access the slave for displaying data on LED.

In this example, 0x3C is written to 0xA5 and 0x01 following the start condition (ST) and slave address (0x60). At the end, the stop condition (SP) is issued to inform the slave of transmission completion.

In responding to that, the slave sends ACK.

6.5.2 Reading A/D Conversion Results

Figure 6.9 shows how to access the slave for reading the A/D conversion results.

In this example, by selecting them with the start condition (ST) and slave address (0x61) first, the upper and lower A/D conversion results of channel 0 are read out, in this order. In this figure, the upper A/D conversion result of channel 0 is 0x02.

In the timing shown at the bottom, the master returns the NACK response after reading the conversion result (0x013C), the slave determines it as completion of communication, and withdraws from communication. Finally, the master issues the stop condition (SP) and releases the I2C bus to complete communication.

When 4-channel A/D conversion is completed, the obtained moving average is set to the IIC buffer for conversion result transmission. Meanwhile, the latest conversion result can be obtained by repeatedly reading the A/D conversion results since the upper conversion result of channel 0 is read after the lower conversion result of channel 3 is read.
6.5.3 Reading Data from RAM

Figure 6.10 shows how to access the slave for reading data from RAM.

![Figure 6.10 Timing of Reading Data from RAM](image)

In this example, following the start condition (ST) and slave address (0x70), register address 0x00 (= RAM address 0x00) is specified. Then, by restarting and selecting the slave (0x71) for reading, the value is read from the specified RAM address 0x00. In this example, the value at address 0x00 is 0x00.

In the timing shown at the bottom, the master returns the NACK response after reading the values from RAM addresses 0x7E and then 0x7F, the slave determines it as completion of communication, and withdraws from communication. Finally, the master issues the stop condition (SP) and releases the I2C bus to complete communication.

6.5.4 Writing Data to RAM

Figure 6.11 shows how to access the slave for writing data to RAM.

![Figure 6.11 Timing of Writing Data to RAM](image)

In this example, following the start condition (ST) and slave address (0x70), register address 0x00 is specified.

Then, 0xAA and 0x55 are transmitted as data to be written to addresses 0x00 and the next address, respectively. After transmitting 2-byte data and completing communication, the master issues the stop condition and releases the I2C bus.
7. Basic Control of I2C Bus through Software

To implement the I2C bus slave function through software, it is necessary to detect the rising and falling edges of the SCL and SDA signals.

For detection, INTP5 (SCL signal edge detection) and INTP6 (SDA signal edge detection) are used.

7.1 Edge Detection Interrupts

To process the detected edges in the limited time, the edge detection interrupts are given top priority over the other interrupts in the nested interrupt system. Even so, it takes 9 to 16 clock cycles for hardware to start the interrupt process. If the interrupt request is generated immediately after the lower priority interrupt is accepted, it takes another 9 clock cycles. Taking this into consideration, the process is performed as quickly as possible.

7.1.1 SCL Edge Detection

Basically, the SCL edge is used for data transmission/reception. Therefore, the edge to be detected is frequently switched between rising and falling to minimize the total processing time. Figure 7.1 shows the valid edges of the SCL and SDA signals when the address is received.

![Figure 7.1 SCL and SDA Edges Used According to Timing](image)

Until the start condition is detected, detection of the both edges of SDA is valid. Even after the stop condition is detected at the falling edge of the SDA signal, detection of all the edges is valid until the falling edge of the SCL signal is detected and slave address reception is started. This enables support of the case in which the stop condition is issued immediately after the start condition.

When slave address reception is started, the SDA signal edge detection interrupt is disabled. When slave address reception is completed at the rising edge of the eight SCL clock cycle, the addresses are compared. If the addresses agree, the program is used to wait for the SCL signal falling edge, and ACK response is started, and the edge of the SCL signal to be detected is changed to the falling edge. (The reason why the program is used to wait for the SCL signal falling edge is accepting the interrupt twice a cycle is too wasteful while there is much processing to do.)

After that, ACK response is ended at the falling edge of the SCL signal and slave address reception is completed. Detection of the rising edge of the SCL signal is enabled for the next.

In the period in which the next SCL signal is high (first clock cycle), detection of all the edges is valid because the start or stop condition may be issued.

7.1.2 SDA Edge Detection

Normal communication operation is sequential operation during SCL edge detection. Meanwhile, edge detection of the SDA signal is used to terminate the sequential operation. Therefore, when to enable acceptance is very important. As shown in figure 7.1, acceptance is enabled only while the SCL signal of the specific timing is high.
7.2 Control Processes

To reduce the processing time, the interrupt processing part is written in the assembly language. Besides, in the INTP5 interrupt process, the processing address according to the contents of the next process is set to the variable next_proc in advance.

This is because the process based on the SCL signal edge is sequential and thus the next process is limited. On the other hand, the process based on the SDA signal edge, which involves SP (stop condition) and ST (start condition), suspends the sequential process, like the interrupt process does.

Furthermore, at the beginning of the INTP5 interrupt process, the state of the input ports for the SCL and SDA signals are taken in the variable g_P_image.

To use such a process without paying any attention, the interface to the upper software is restricted.

7.2.1 Sequences based on SCL Edge Detection Interrupt (1)

Sequences are defined assuming sequence 1 is the state in which both the SCL and SDA signals are high before communication starts. Table 7.1 shows the sequences for address reception and non-selection.

<table>
<thead>
<tr>
<th>Sequence No.</th>
<th>State/Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence 1</td>
<td>Initial state (SCL and SDA signals are high.)</td>
</tr>
<tr>
<td>Sequence 2</td>
<td>Detection of stop condition (waits for start condition next.)</td>
</tr>
<tr>
<td>Sequence 3</td>
<td>Detection of start condition (waits for SCL falling.)</td>
</tr>
<tr>
<td>Sequence 4</td>
<td>SCL falling after start condition detection (waits for address reception.)</td>
</tr>
<tr>
<td>Sequence 5</td>
<td>SCL rising edge (takes in slave address.)</td>
</tr>
<tr>
<td>Sequence 6</td>
<td>8th SCL rising (slave addresses agree.)</td>
</tr>
<tr>
<td>Sequence 6'</td>
<td>8th SCL rising (slave addresses disagree.)</td>
</tr>
<tr>
<td>Sequence 7</td>
<td>8th SCL falling (starts ACK response.)</td>
</tr>
<tr>
<td>Sequence 8</td>
<td>9th SCL falling (completes ACK response.)</td>
</tr>
<tr>
<td>Sequence 21</td>
<td>9th SCL rising (slave addresses disagree.)</td>
</tr>
<tr>
<td>Sequence 22</td>
<td>9th SCL rising without ACK response (waits for ST and SP.)</td>
</tr>
<tr>
<td>Sequences 23 and 25</td>
<td>9th or 1st SCL falling (discontinues waiting for ST and SP; skips reading SCL.)</td>
</tr>
<tr>
<td>Sequence 24</td>
<td>9th SCL rising (waits for ST and SP.)</td>
</tr>
<tr>
<td>Sequence 26</td>
<td>Counting SCL rising (waits for 9th SCL rising.)</td>
</tr>
</tbody>
</table>

7.2.2 Sequences based on SCL Edge Detection Interrupt (2)

Table 7.2 shows the reception process sequences. In the reception process, when 8-bit data is complete, the SCL bus is placed in the wait state to inform the upper process (received data is set in the variable _g_IICA, the communication status is set in the variable _g_IICS, and _g_IIC_IF is set to 0x01).

At the first SCL cycle, restart/stop or start of next data reception may occur; therefore, all the sources are enabled beforehand. If restart/stop is detected, the reception process sequence is cancelled. If SCL falls, it means continuation of reception, and thus SDA edge detection is disabled and process is continued.

<table>
<thead>
<tr>
<th>Sequence No.</th>
<th>State/Process</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence 8'</td>
<td>9th SCL falling (informs the upper software of information at reception completion.)</td>
<td>_Rx_data_sub is performed next.</td>
</tr>
<tr>
<td>Rx_data_sub</td>
<td>Releases I2C bus from wait state.</td>
<td>Sequence 9 is performed next.</td>
</tr>
<tr>
<td>Sequence 9</td>
<td>1st SCL rising (takes in received data.)</td>
<td>Sequence 11 is performed next.</td>
</tr>
<tr>
<td>Sequence 6</td>
<td>8th SCL rising (prepares for ACK response.)</td>
<td></td>
</tr>
<tr>
<td>Sequence 7</td>
<td>8th SCL falling (starts ACK response.)</td>
<td>Sequence 8→8' is performed next.</td>
</tr>
<tr>
<td>Sequence 11</td>
<td>Takes in data at SCL rising (waits for ST, SP, and SCL falling edge.)</td>
<td>Sequence 6 is performed next.</td>
</tr>
<tr>
<td>Sequence 10</td>
<td>1st SCL falling (discontinues waiting for ST and SP.)</td>
<td>Sequence 11 is performed next.</td>
</tr>
</tbody>
</table>
Having processed the received data in the variable _g_IICA, the upper process calls the function _Rx_data_sub to release the I2C bus from the wait state and restart the next reception.

7.2.3 Sequences based on SCL Edge Detection Interrupt (3)

Table 7.3 shows the transmission process sequences. In the transmission process, after ACK response upon agreement of slave addresses, the SCL bus is placed in the wait state to inform the upper process (the communication status is set in the variable _g_IICS and _g_IIC_IF is set to 0x01).

<table>
<thead>
<tr>
<th>Sequence No.</th>
<th>State/Process</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence 14</td>
<td>Selection for transmission (informs the upper software of information upon address reception completion.)</td>
<td>_Tx_data_sub is performed next.</td>
</tr>
<tr>
<td>_Tx_data_sub</td>
<td>Outputs 1st bit; releases I2C bus from wait state.</td>
<td>Sequence 15 is performed next.</td>
</tr>
<tr>
<td>Sequence 15</td>
<td>1st SCL rising; clears ACKD, STD, and SPD.</td>
<td>Sequence 16 is performed next.</td>
</tr>
<tr>
<td>Sequence 16</td>
<td>SCL falling (data transmission timing; repeats 7 times.)</td>
<td></td>
</tr>
<tr>
<td>Sequence 17</td>
<td>8th SCL falling (starts ACK reception.)</td>
<td>Sequence 18 is performed next.</td>
</tr>
<tr>
<td>Sequence 18</td>
<td>Receives ACK at 9th SCL rising.</td>
<td></td>
</tr>
<tr>
<td>Sequence 19</td>
<td>Detects ACK at 9th SCL rising.</td>
<td>Sequence 14 is performed next.</td>
</tr>
<tr>
<td>Sequence 20</td>
<td>9th SCL falling (informs the upper software of NACK.)</td>
<td>_Tx_end_sub is performed next.</td>
</tr>
<tr>
<td>_Tx_end_sub</td>
<td>Clears communication status and releases I2C bus.</td>
<td></td>
</tr>
</tbody>
</table>

Having set the transmission data in the argument (A register), the upper process calls the function _Tx_data_sub and outputs the next data to SDA to release the I2C bus from the wait state and restart the next data transmission.

For slave transmission, the slave is driving the SCA signal except for the 9th clock cycle; therefore, when the master is to perform any operation, it is necessary to return NACK response to the slave to stop transmission. When the slave receives NACK, it informs the upper software of NACK response, and releases the I2C bus from the wait state by using the function _Tx_end_sub called by the upper software to withdraw from communication (SCL edge detection is disabled).

7.3 I2C Slave File Configuration

The library of the I2C slave functions provided by this software consists of the following three files.

- _r_iicss_lib.asm: Program body that controls I2C (recommended not to be modified)
- _r_iicss_adr.asm: Definitions including slave addresses on the I2C bus. Modify as necessary.
- _r_intlic.c: Equivalent part of INTIICA0 processing part in IICA0. Write the processes using the I2C bus. In this sample program, the functions described in 6.2 Functions as Slave are implemented. To use the other functions, modify the processes here.
8. Settings through Code Creation

Set the following items with "API function output control" of "File creation mode" of Property set to "Output only initialization functions".

(1) Clock generation circuit settings
   (a) Pin assignment settings: Fix as they are.
   (b) Clock settings
      - Operating mode settings: High-speed main mode $2.7 (V) \leq VDD \leq 3.6 (V)$
      - Main system clock (fMAIN) settings: High-speed on-chip oscillator clock (fIH)
      - High-speed on-chip oscillator clock settings: 24 (MHz)
      - Middle-speed on-chip oscillator clock settings: Do not check "operate".
      - High-speed system clock settings: Do not check "operate".
      - Subsystem clock (fSUB) settings: Do not check "operate".
      - Low-speed on-chip oscillator clock (fIL) settings: Frequency 15 (kHz)
      - RTC, FMC, interval timer, PCLBUZ operating clock settings: fIL
      - CPU and peripheral clock settings: 24000 (fIH) (kHz)
   (c) On-chip debugging settings
      - On-chip debugging operation settings: Use.
      - RRM function settings: Do not use.
      - Security ID settings: Set security ID.
      - Settings upon security ID authentication failure: Delete flash memory data.
   (d) Reset source checking
      - Reset source checking function output: Remove the check mark.
   (e) Safety functions: Select "Do not use" for all.
   (f) Data flash: Prohibit access to data flash.

(2) Port settings
   - Set P0.0 to P0.3 to output. (data: 0)
   - Set P6.0 and P6.1 to output 1.
   - Leave the other ports as default (Do not use).

(3) Timer settings
   (a) General settings Channel 3: Interval timer
   (b) Channel 3
      - Operating mode settings: 16 bits
      - Interval time (16 bits) settings: 50 ms
      - Leave the other settings as default.

(4) Frequency measurement circuit settings
   - Leave as default (Do not use).

(5) 12-bit interval timer settings
   - Leave as default (Do not use).

(6) 8-bit interval timer settings
   - Leave as default (Do not use).

(7) Clock output/buzzer output settings
   - Leave as default (Do not use) for all.

(8) Watchdog timer settings
   - Operation settings in HALT/STOP/SNOOZE mode: Stop.
- Watchdog timer operation settings: Do not use.

(9) A/D converter settings
- A/D converter operation settings: Use
- Comparator operation settings: Enable
- Resolution settings: 10 bits
- VREF(+) settings: AVDD
- VREF(-) settings: AVSS
- Trigger mode settings: Software trigger mode
- Operating mode settings: Continuous scan mode
- ANI0 – ANI3 analog input pin settings: ANI0 – ANI3
- ANI16 – ANI18 analog input pin settings: Remove all check marks.
- Conversion start channel settings: ANI0 – ANI3
- Reference voltage: 2.7V \leq AVDD \leq 3.6V
- Conversion time mode: Standard 1
- Conversion time: 9 (216/fCLK) (μs)
- Conversion result upper/lower limit settings: Interrupt request signal (INTAD) generated when ADLL \leq ADCRH \leq ADUL
- Interrupt settings: Check the A/D interrupt enable (priority level = 1)

(10) Comparator settings
Leave as default (Do not use).

(11) Op amp. settings
Leave as default (Do not use).

(12) Serial array unit settings
Leave as default (Do not use).

(13) Data operation circuit settings
Leave as default (Do not use).

(14) Data transfer controller settings
Leave as default (no check marks).

(15) Event link controller settings
Leave as default (no check marks).

(16) Interrupt settings
- INTP5 setting: Rising edge, priority level: high
- INTP6 setting: Both edges, priority level: high
Leave the other settings as default (no check marks).

(17) Key interrupt function settings
Leave as default (no check marks).

(18) Voltage detection circuit settings
- Voltage detection operation settings: Use.
- Operating mode settings: Reset mode
- Detected voltage settings: 2.75 (V)
9. Sample Code
   The user can get the sample code from the Renesas Electronics website.

10. Reference Documents
    RL78/I1D User’s Manual: Hardware Rev.2.10 (R01UH0474J)
    RL78 Family User’s Manual: Software Rev.2.00 (R01US0015J)
    (Get the latest version from the Renesas Electronics website.)

    Technical Updates/Technical News
    (Get the latest information from the Renesas Electronics website.)
Website and Support

Renesas Electronics Website

http://japan.renesas.com/

Inquiries

http://japan.renesas.com/inquiry

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Revision Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>2016.11.15</td>
<td>—</td>
<td>Newly created.</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

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

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

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

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of 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. No licenses, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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 tool, personal electronic equipment, and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-criminal systems, and safety equipment etc.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of a failure of a Renesas Electronics product, 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. Please make an evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacturer, use, or sale is prohibited under applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technology described in this document for any purposes relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document; Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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