RL78/G14 Group

I2C Repeated Start Signal

R01AN1947EC0100
Rev.1.00
Jan 25, 2014

Introduction

This application note is to explain how to modify the I2C sample code generated by code generator to support I2C repeat start.

Target Device

RL78/G14

The sample code was evaluated by RL78/G14.

This sample code can also be used on another group of RL78 by changing startup program.

Table of content

1. Introduction

During an I2C transfer, there is often need to first send a command and then read back an answer. This has to be done without the risk of another (multi-master) device interrupting this atomic operation. The I2C protocol defines a so-called repeated start condition.

RL78 IICA module can support I2C repeat start condition as below.

ST	AD6 to AD0	R/$\overline{\text{W}}$	$\overline{\text{ACK}}$	D7 to D0	$\overline{\text{ACK}}$	ST	AD6 to AD0	R/$\overline{\text{W}}$	$\overline{\text{ACK}}$	D7 to D0	$\overline{\text{ACK}}$	SP

ST:	Start condition
AD6 to AD0:	Address
R/$\overline{\text{W}}$:	Transfer direction specification
$\overline{\text{ACK}}$:	Acknowledge
D7 to D0:	Data
SP:	Stop condition

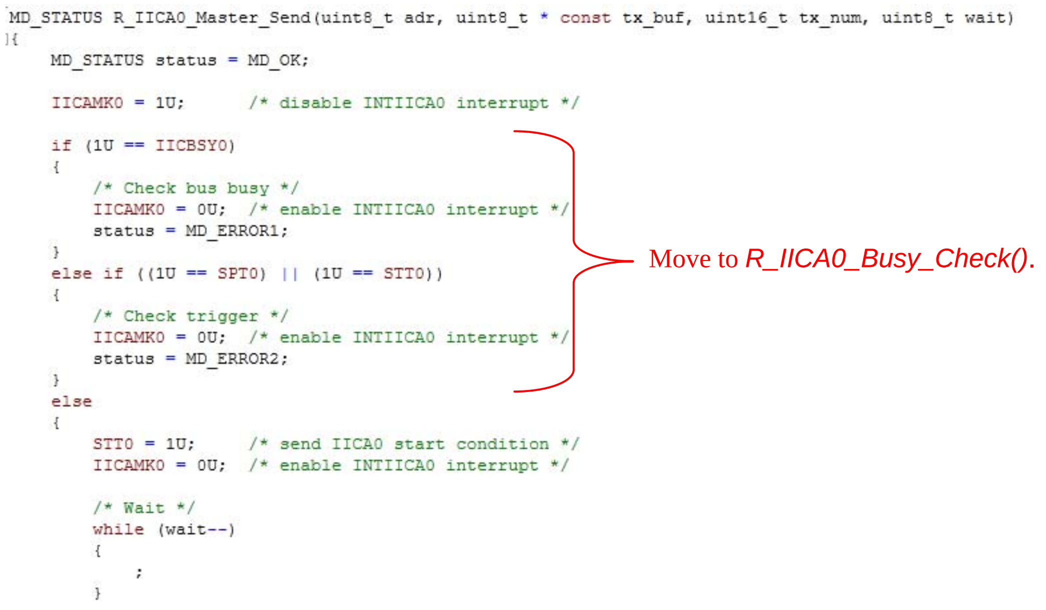
However, the I2C sample code generated by code generator don't support I2C repeat start condition yet. After modified the sample program, the repeat start condition can be supported. This application note is to describe how to modify the sample code generated by code generator to support I2C repeat start (master and slave).

2. I2C Master sample code modification

Ignore bus busy check

In the I2C Master sample program generated by code generator, IICBSY register will be checked before sending the IIC slave address. If IICBSY is 1, error will be returned.

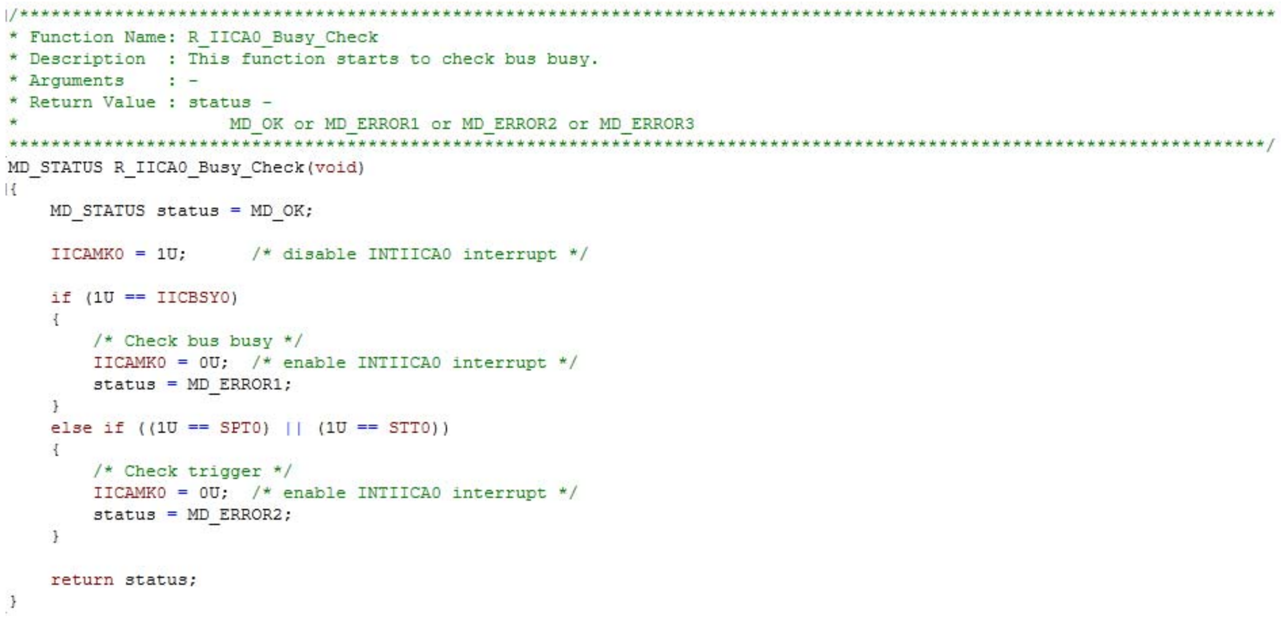
However, the bus status is busy in repeat start condition. The program should ignore checking IICBSY in repeat start condition. We should move the code below in *R_IICA0_Master_Send()* and *R_IICA0_Master_Receive()* to a new subroutine *R_IICA0_Busy_Check()*.



```
MD_STATUS R_IICA0_Master_Send(uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num, uint8_t wait)
{
    MD_STATUS status = MD_OK;

    IICAMK0 = 1U;      /* disable INTIICA0 interrupt */

    if (1U == IICBSY0)
    {
        /* Check bus busy */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR1;
    }
    else if ((1U == SPT0) || (1U == STT0))
    {
        /* Check trigger */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR2;
    }
    else
    {
        STT0 = 1U;     /* send IICA0 start condition */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */

        /* Wait */
        while (wait--)
        {
            ;
        }
```

Move to *R_IICA0_Busy_Check()*.

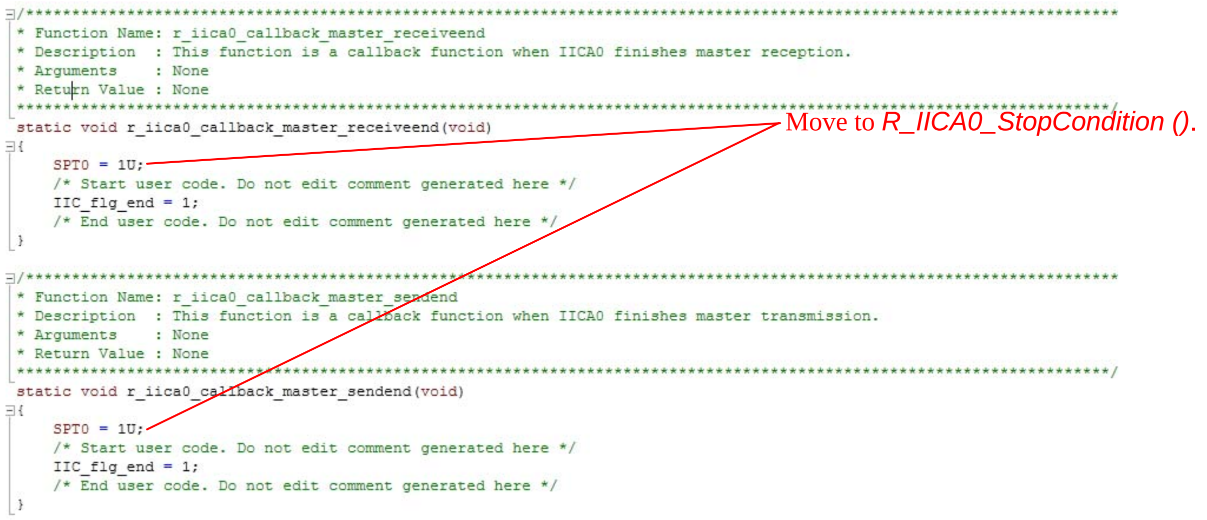
The code below is the new created subroutine *R_IICA0_Busy_Check().*



```
/**************************************************************************************************************
* Function Name: R_IICA0_Busy_Check
* Description  : This function starts to check bus busy.
* Arguments    : -
* Return Value : status -
*                    MD_OK or MD_ERROR1 or MD_ERROR2 or MD_ERROR3
**************************************************************************************************************/
MD_STATUS R_IICA0_Busy_Check(void)
{
    MD_STATUS status = MD_OK;

    IICAMK0 = 1U;      /* disable INTIICA0 interrupt */

    if (1U == IICBSY0)
    {
        /* Check bus busy */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR1;
    }
    else if ((1U == SPT0) || (1U == STT0))
    {
        /* Check trigger */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR2;
    }

    return status;
}
```

Skip stop condition

For repeat start condition, stop condition should be sent out at the end of the I2C command frame. The code for stop condition in *r_iica0_callback_master_sendend()* and *r_iica0_callback_master_receiveend()* should be move to a new subroutine *R_IICA0_StopCondition().*



```
/***********************************************************************************************************************
* Function Name: r_iica0_callback_master_receiveend
* Description  : This function is a callback function when IICA0 finishes master reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
static void r_iica0_callback_master_receiveend(void)
{
    SPT0 = 1U;
    /* Start user code. Do not edit comment generated here */
    IIC_flg_end = 1;
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************************
* Function Name: r_iica0_callback_master_sendend
* Description  : This function is a callback function when IICA0 finishes master transmission.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
static void r_iica0_callback_master_sendend(void)
{
    SPT0 = 1U;
    /* Start user code. Do not edit comment generated here */
    IIC_flg_end = 1;
    /* End user code. Do not edit comment generated here */
}
```

The code below is the new created subroutine *R_IICA0_StopCondition().*

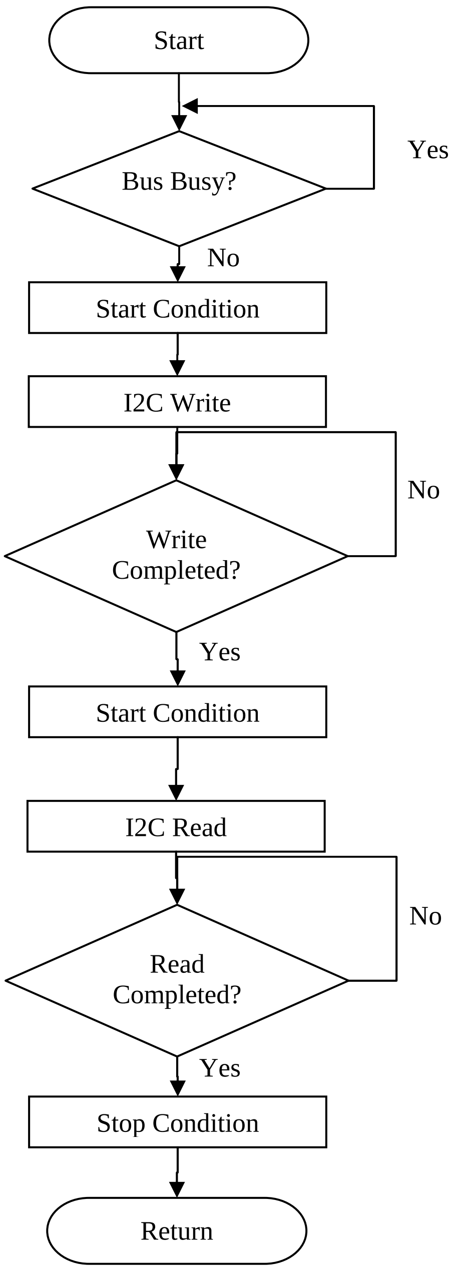
```
/***********************************************************************************************************************
* Function Name: R_IICA0_StopCondition
* Description  : This function sets IICA0 stop condition flag.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
void R_IICA0_StopCondition(void)
{
    SPT0 = 1U;    /* set stop condition flag */
}
```

Main program for sending I2C command with repeat start condition

For sending the I2C using the modified code, pls refer to the below code example.

```
if (R_IICA0_Busy_Check() == MD_OK)  //Check bus busy
{
        IIC_flg_end = 0;
        R_IICA0_Master_Send(0xA0, &IIC_data[0], 1, 0);    //Send data to slave

        while(IIC_flg_end == 0)         //Wait for transmit end
                NOP();
        IIC_flg_end = 0;

        IIC_data[0]= 0;
        R_IICA0_Master_Receive(0xA0, &IIC_data[0], 1, 0);   //Read data from slave
        timer = 0;
        while(IIC_flg_end == 0)         //Wait for recieved end
                NOP();
        IIC_flg_end = 0;

        R_IICA0_StopCondition();        //Send stop condition
}
```

3. Program flow of I2C master operation with repeat start condition

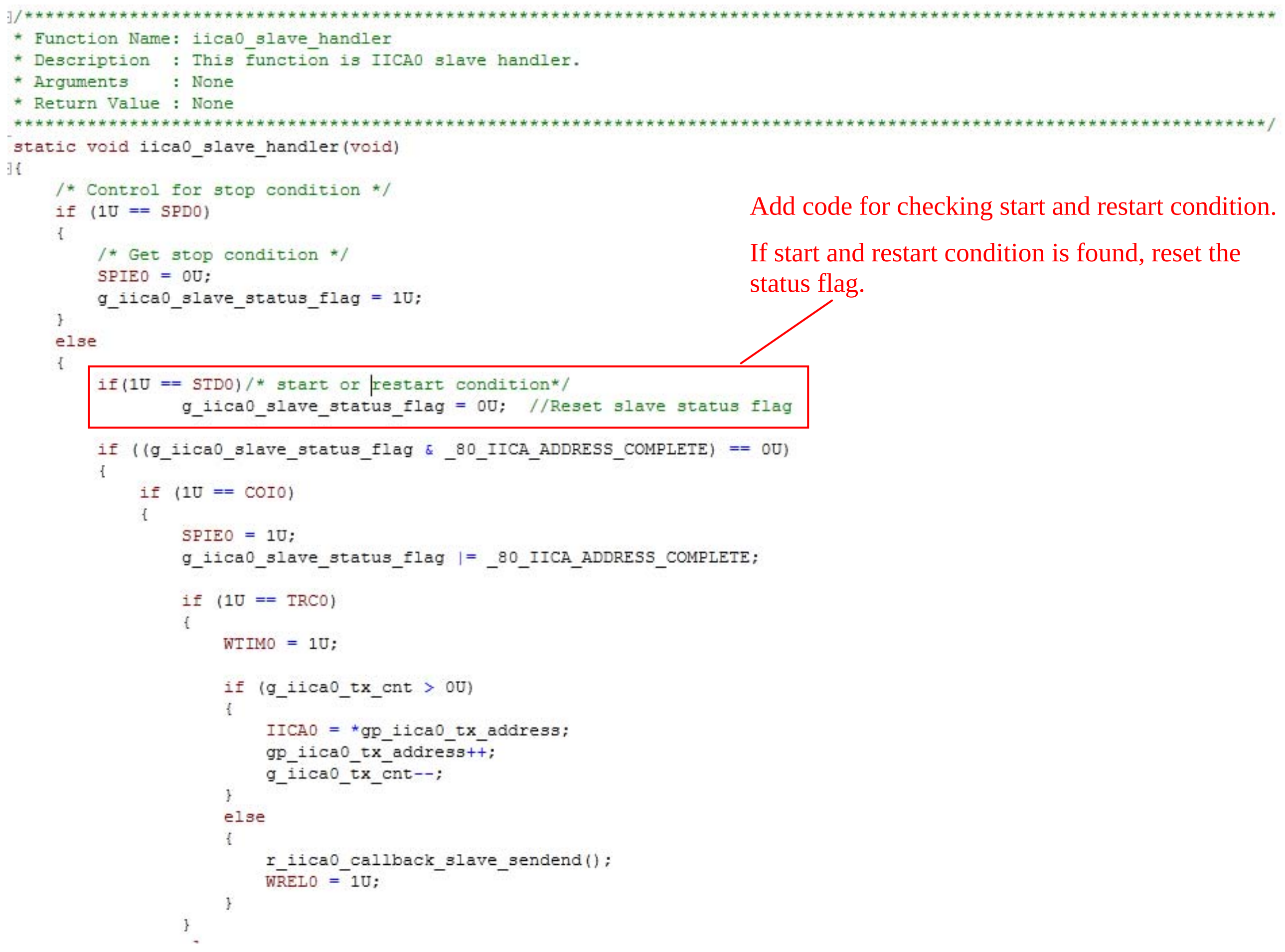


4. I2C Slave sample code modification

Add checking for repeat start condition

In the I2C Slave sample program generated by code generator, the start condition hasn't been checked. The state machine will not reset when received the start/restart condition from I2C master device, and it will cause problem in restart condition.

In order to support repeat start condition, we can add procedure to check the start condition and reset the state machine. The modification code was shown as below.

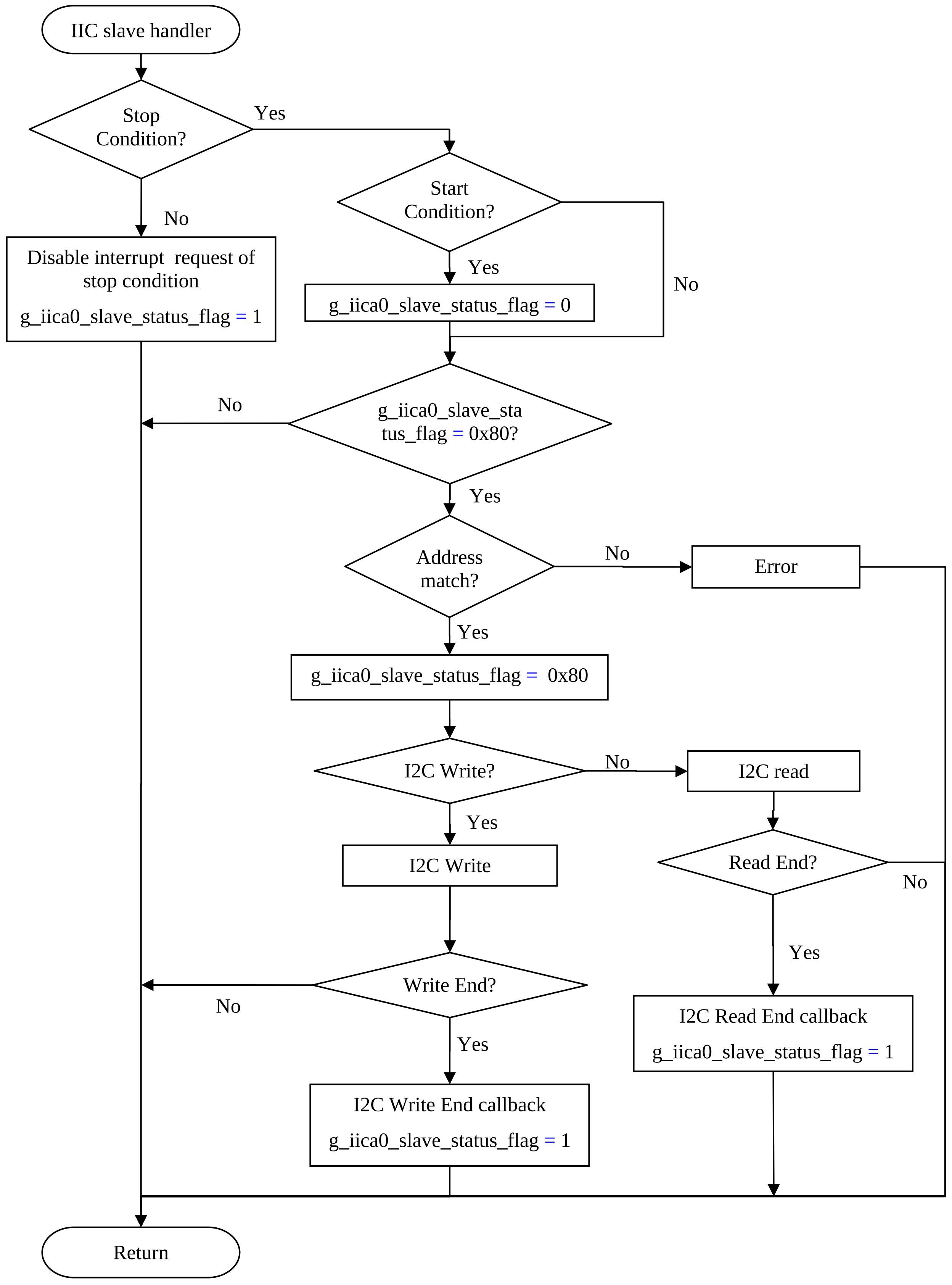


```
/***********************************************************************************************************************
* Function Name: iica0_slave_handler
* Description  : This function is IICA0 slave handler.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
static void iica0_slave_handler(void)
{
    /* Control for stop condition */
    if (1U == SPD0)
    {
        /* Get stop condition */
        SPIE0 = 0U;
        g_iica0_slave_status_flag = 1U;
    }
    else
    {
        if(1U == STD0)/* start or restart condition*/
                g_iica0_slave_status_flag = 0U;  //Reset slave status flag

        if ((g_iica0_slave_status_flag & _80_IICA_ADDRESS_COMPLETE) == 0U)
        {
            if (1U == COI0)
            {
                SPIE0 = 1U;
                g_iica0_slave_status_flag |= _80_IICA_ADDRESS_COMPLETE;

                if (1U == TRC0)
                {
                    WTIM0 = 1U;

                    if (g_iica0_tx_cnt > 0U)
                    {
                        IICA0 = *gp_iica0_tx_address;
                        gp_iica0_tx_address++;
                        g_iica0_tx_cnt--;
                    }
                    else
                    {
                        r_iica0_callback_slave_sendend();
                        WREL0 = 1U;
                    }
                }
```

Add code for checking start and restart condition.

If start and restart condition is found, reset the status flag.

5. Program flow of I2C slave operation with repeat start condition


IIC slave handler
Stop Condition?
Yes
No
Disable interrupt request of stop condition
g_iica0_slave_status_flag = 1
Start Condition?
Yes
No
g_iica0_slave_status_flag = 0
g_iica0_slave_status_flag = 0x80?
No
Yes
Address match?
No
Error
Yes
g_iica0_slave_status_flag = 0x80
I2C Write?
No
I2C read
Yes
I2C Write
Read End?
No
Yes
Write End?
No
Yes
I2C Read End callback
g_iica0_slave_status_flag = 1
I2C Write End callback
g_iica0_slave_status_flag = 1
Return

6. I2C Master Sample Program

```
/*****************************************************************************
* File Name    : r_main.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements main function.
* Creation Date: 13/9/2013
*****************************************************************************/

/*****************************************************************************
Pragma directive
*****************************************************************************/

/*****************************************************************************
Includes
*****************************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_serial.h"
#include "r_cg_userdefine.h"

/*****************************************************************************
Global variables and functions
*****************************************************************************/
uint8_t IIC_data[250], timer,i;
uint16_t l;
uint8_t IIC_flg_end = 0;


/*****************************************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
*****************************************************************************/
void main(void)
{
    for(i=0; i<250; i++)
       IIC_data[i]=0;

    timer = 0;

    while (1U)
    {
          for(l=0; l<55000; l++)
             NOP();

          if (R_IICA0_Busy_Check() == MD_OK)  //Check bus busy
          {
             timer = 0;
             IIC_flg_end = 0;
             //Send data to slave
             R_IICA0_Master_Send(0xA0, &IIC_data[0], 1, 0);
             //Wait for transmit end
             while(IIC_flg_end == 0)
```

```
                    NOP();
                IIC_flg_end = 0;

                IIC_data[0]= 0;
                //Read data from slave
                R_IICA0_Master_Receive(0xA0, &IIC_data[0], 1, 0);
                timer = 0;
                //Wait for recieved end
                while(IIC_flg_end == 0)
                    NOP();
                IIC_flg_end = 0;

                R_IICA0_StopCondition();   //Send stop condition
        }
  }
}
```

```
/***********************************************************************
* File Name    : r_cg_serial.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements device driver for Serial module.
* Creation Date: 13/9/2013
***********************************************************************/

/***********************************************************************
Pragma directive
***********************************************************************/

/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_serial.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************
Global variables and functions
***********************************************************************/
volatile uint8_t   g_iica0_master_status_flag; /* iica0 master flag */
volatile uint8_t   g_iica0_slave_status_flag;  /* iica0 slave flag */
volatile uint8_t * gp_iica0_rx_address;        /* iica0 receive buffer address */
volatile uint16_t  g_iica0_rx_len;             /* iica0 receive data length */
volatile uint16_t  g_iica0_rx_cnt;             /* iica0 receive data count */
volatile uint8_t * gp_iica0_tx_address;        /* iica0 send buffer address */
volatile uint16_t  g_iica0_tx_cnt;             /* iica0 send data count */
/* Start user code for global. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

/***********************************************************************
* Function Name: R_IICA0_Create
* Description  : This function initializes the IICA0 module.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_IICA0_Create(void)
{
    IICA0EN = 1U;  /* supply IICA0 clock */
    IICE0 = 0U;    /* disable IICA0 operation */
    IICAMK0 = 1U;  /* disable INTIICA0 interrupt */
    IICAIF0 = 0U;  /* clear INTIICA0 interrupt flag */
    /* Set INTIICA0 high priority */
    IICAPR10 = 0U;
    IICAPR00 = 0U;
    /* Set SCLA0, SDAA0 pin */
    P1 &= 0xFCU;
    PM1 &= 0xFCU;
    SMC0 = 1U;
    IICWL0 = _15_IICA0_IICWL_VALUE;
    IICWH0 = _14_IICA0_IICWH_VALUE;
    DFC0 = 0U;     /* digital filter off */
    IICCTL01 |= _01_IICA_fCLK_HALF;
    SVA0 = _10_IICA0_MASTERADDRESS;
    STCEN0 = 1U;
    IICRSV0 = 1U;
    SPIE0 = 0U;
    WTIM0 = 1U;
    ACKE0 = 1U;
    IICAMK0 = 0U;
    IICE0 = 1U;
    LREL0 = 1U;
    /* Set SCLA0, SDAA0 pin */
    PM1 &= 0xFCU;
}

/******************************************************************************
* Function Name: R_IICA0_Stop
* Description  : This function stops IICA0 module operation.
* Arguments    : None
* Return Value : None
******************************************************************************/
void R_IICA0_Stop(void)
{
    IICE0 = 0U;    /* disable IICA0 operation */
}

/******************************************************************************
* Function Name: R_IICA0_StopCondition
* Description  : This function sets IICA0 stop condition flag.
* Arguments    : None
* Return Value : None
******************************************************************************/
void R_IICA0_StopCondition(void)
{
    SPT0 = 1U;    /* set stop condition flag */
}

/******************************************************************************
* Function Name: R_IICA0_Busy_Check
* Description  : This function starts to check bus busy.
* Arguments    : -
* Return Value : status -
*                    MD_OK or MD_ERROR1 or MD_ERROR2 or MD_ERROR3
******************************************************************************/
MD_STATUS R_IICA0_Busy_Check(void)
{
    MD_STATUS status = MD_OK;

    IICAMK0 = 1U;      /* disable INTIICA0 interrupt */

    if (1U == IICBSY0)
    {
        /* Check bus busy */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR1;
    }
    else if ((1U == SPT0) || (1U == STT0))
    {
        /* Check trigger */
        IICAMK0 = 0U;  /* enable INTIICA0 interrupt */
        status = MD_ERROR2;
    }

    return status;
}


/***********************************************************************************
* Function Name: R_IICA0_Master_Send
* Description  : This function starts to send data as master mode.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
*                wait -
*                    wait for start condition
* Return Value : status -
*                    MD_OK or MD_ERROR1 or MD_ERROR2 or MD_ERROR3
***********************************************************************************/
MD_STATUS R_IICA0_Master_Send(uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num,
uint8_t wait)
{
    MD_STATUS status = MD_OK;

    IICAMK0 = 1U;      /* disable INTIICA0 interrupt */
    STT0 = 1U;     /* send IICA0 start condition */
    IICAMK0 = 0U;  /* enable INTIICA0 interrupt */

    /* Wait */
    while (wait--)
    {
       ;
    }

    if(0U == STD0)
    {
        status = MD_ERROR3;
    }

    /* Set parameter */
    g_iica0_tx_cnt = tx_num;
    gp_iica0_tx_address = tx_buf;
    g_iica0_master_status_flag = _00_IICA_MASTER_FLAG_CLEAR;
    adr &= (uint8_t)~0x01U; /* set send mode */
    IICA0 = adr;           /* send address */

    return (status);
}
```

```
/******************************************************************************
* Function Name: R_IICA0_Master_Receive
* Description  : This function starts to receive IICA0 data as master mode.
* Arguments    : rx_buf -
*                    receive buffer pointer
*                rx_num -
*                    buffer size
*                wait -
*                    wait for start condition
* Return Value : status -
*                    MD_OK or MD_ERROR1 or MD_ERROR2 or MD_ERROR3
******************************************************************************/
MD_STATUS R_IICA0_Master_Receive(uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num,
uint8_t wait)
{
    MD_STATUS status = MD_OK;

    IICAMK0 = 1U;  /* disable INTIIA0 interrupt */

    {
        STT0 = 1U;     /* set IICA0 start condition */
        IICAMK0 = 0U;  /* enable INTIIA0 interrupt */

        /* Wait */
        while (wait--)
        {
            ;
        }

        if(0U == STD0)
        {
            status = MD_ERROR3;
        }

        /* Set parameter */
        g_iica0_rx_len = rx_num;
        g_iica0_rx_cnt = 0U;
        gp_iica0_rx_address = rx_buf;
        g_iica0_master_status_flag = _00_IICA_MASTER_FLAG_CLEAR;
        adr |= 0x01U;    /* set receive mode */
        IICA0 = adr;    /* receive address */
    }

    return (status);
}
```

```
/***********************************************************************
* File Name    : r_cg_serial_user.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements device driver for Serial module.
* Creation Date: 13/9/2013
***********************************************************************/

/***********************************************************************
Pragma directive
***********************************************************************/
#pragma interrupt INTIICA0 r_iica0_interrupt

/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_serial.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************
Global variables and functions
***********************************************************************/
extern volatile uint8_t   g_iica0_master_status_flag;  /* iica0 master flag */
extern volatile uint8_t   g_iica0_slave_status_flag;   /* iica0 slave flag */
extern volatile uint8_t * gp_iica0_rx_address;         /* iica0 receive buffer
address */
extern volatile uint16_t  g_iica0_rx_cnt;              /* iica0 receive data
length */
extern volatile uint16_t  g_iica0_rx_len;              /* iica0 receive data
count */
extern volatile uint8_t * gp_iica0_tx_address;         /* iica0 send buffer
address */
extern volatile uint16_t  g_iica0_tx_cnt;              /* iica0 send data
count */
extern uint8_t IIC_flg_end;

/***********************************************************************
* Function Name: r_iica0_interrupt
* Description  : This function is INTIICA0 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************/
__interrupt static void r_iica0_interrupt(void)
{
      P20 = 0xff;
    if ((IICS0 & _80_IICA_STATUS_MASTER) == 0x80U)
    {
        iica0_master_handler();
    }
        P20 = 0x00;
}
```

```
/******************************************************************************
* Function Name: iica0_master_handler
* Description  : This function is IICA0 master handler.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void iica0_master_handler(void)
{
    /* Control for communication */
    if ((0U == IICBSY0) && (g_iica0_tx_cnt != 0U))
    {
        r_iica0_callback_master_error(MD_SPT);
    }
    /* Control for sended address */
    else
    {
        if ((g_iica0_master_status_flag & _80_IICA_ADDRESS_COMPLETE) == 0U)
        {
            if (1U == ACKD0)
            {
                g_iica0_master_status_flag |= _80_IICA_ADDRESS_COMPLETE;
                if (1U == TRC0)
                {
                    WTIM0 = 1U;
                    if (g_iica0_tx_cnt > 0U)
                    {
                        IICA0 = *gp_iica0_tx_address;
                        gp_iica0_tx_address++;
                        g_iica0_tx_cnt--;
                    }
                    else
                    {
                        r_iica0_callback_master_sendend();
                    }
                }
                else
                {
                    ACKE0 = 1U;
                    WTIM0 = 0U;
                    WREL0 = 1U;
                }
            }
            else
            {
                r_iica0_callback_master_error(MD_NACK);
            }
        }
        else
        {
            /* Master send control */
            if (1U == TRC0)
            {
                if ((0U == ACKD0) && (g_iica0_tx_cnt != 0U))
                {
                    r_iica0_callback_master_error(MD_NACK);
                }
                else
                {
                    if (g_iica0_tx_cnt > 0U)
                    {
                        IICA0 = *gp_iica0_tx_address;
                        gp_iica0_tx_address++;
                        g_iica0_tx_cnt--;
                    }
                    else
                    {
                        r_iica0_callback_master_sendend();
                    }
                }
            }
            /* Master receive control */
            else
            {
                if (g_iica0_rx_cnt < g_iica0_rx_len)
                {
                    *gp_iica0_rx_address = IICA0;
                    gp_iica0_rx_address++;
                    g_iica0_rx_cnt++;

                    if (g_iica0_rx_cnt == g_iica0_rx_len)
                    {
                        ACKE0 = 0U;
                        WREL0 = 1U;
                        WTIM0 = 1U;
                    }
                    else
                    {
                        WREL0 = 1U;
                    }
                }
                else
                {
                    r_iica0_callback_master_receiveend();
                }
            }
        }
    }
}

/******************************************************************************
* Function Name: r_iica0_callback_master_error
* Description  : This function is a callback function when IICA0 master error
occurs.
* Arguments    : flag -
*                    status flag
* Return Value : None
******************************************************************************/
static void r_iica0_callback_master_error(MD_STATUS flag)
{
    /* Start user code. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}
```

```
/******************************************************************************
* Function Name: r_iica0_callback_master_receiveend
* Description  : This function is a callback function when IICA0 finishes
master reception.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void r_iica0_callback_master_receiveend(void)
{
    IIC_flg_end = 1;
}

/******************************************************************************
* Function Name: r_iica0_callback_master_sendend
* Description  : This function is a callback function when IICA0 finishes
master transmission.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void r_iica0_callback_master_sendend(void)
{
    IIC_flg_end = 1;
}
```

7. I2C Slave Sample Program

```
/******************************************************************************
* File Name    : r_main.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements main function.
* Creation Date: 13/9/2013
******************************************************************************/

/******************************************************************************
Pragma directive
******************************************************************************/

/******************************************************************************
Includes
******************************************************************************/#
include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_serial.h"
#include "r_cg_userdefine.h"

/******************************************************************************
Global variables and functions
******************************************************************************/
extern unsigned char rx_end;
extern unsigned char tx_end;
uint8_t  g_read_value[250]; //buffer I2C read
uint8_t  g_write_value[250];//buffer I2C write

/******************************************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
******************************************************************************/
void main(void)
{
     R_IICA0_Slave_Receive(&g_read_value[0], 1);

     g_write_value[0] = 0x55;
    R_IICA0_Slave_Send(&g_write_value[0], 1);
    while (1U)
    {
       if (rx_end == 1)    //1 byte IIC data recieved?
       {
          rx_end = 0;     //clr recieve flag
          //set recieve another IIC data
          R_IICA0_Slave_Receive(&g_read_value[0], 1);    }

          if (tx_end == 1)    //1 byte IIC data transmit?
          {
             tx_end = 0;
             g_write_value[0] = 0x55;
             //set recieve another 250 IIC data
```

```
            R_IICA0_Slave_Send(&g_write_value[0], 1);  }
      }
  }
```

```
/***********************************************************************
* File Name    : r_cg_serial.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements device driver for Serial module.
* Creation Date: 13/9/2013
***********************************************************************/

/***********************************************************************
Pragma directive
***********************************************************************/

/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_serial.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************
Global variables and functions
***********************************************************************/
volatile uint8_t   g_iica0_master_status_flag; /* iica0 master flag */
volatile uint8_t   g_iica0_slave_status_flag;  /* iica0 slave flag */
volatile uint8_t * gp_iica0_rx_address;        /* iica0 receive buffer address */
volatile uint16_t  g_iica0_rx_len;             /* iica0 receive data length */
volatile uint16_t  g_iica0_rx_cnt;             /* iica0 receive data count */
volatile uint8_t * gp_iica0_tx_address;        /* iica0 send buffer address */
volatile uint16_t  g_iica0_tx_cnt;             /* iica0 send data count */
/* Start user code for global. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

/***********************************************************************
* Function Name: R_IICA0_Create
* Description  : This function initializes the IICA0 module.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_IICA0_Create(void)
{
    IICA0EN = 1U; /* supply IICA0 clock */
    IICE0 = 0U; /* disable IICA0 operation */
    IICAMK0 = 1U; /* disable INTIICA0 interrupt */
    IICAIF0 = 0U; /* clear INTIICA0 interrupt flag */
    /* Set INTIICA0 high priority */
    IICAPR10 = 0U;
    IICAPR00 = 0U;
    /* Set SCLA0, SDAA0 pin */
    P6 &= 0xFCU;
    P6 |= 0x03;
    PM6 |= 0x03U;
    SMC0 = 1U;
    IICWL0 = _15_IICA0_IICWL_VALUE;
    IICWH0 = _14_IICA0_IICWH_VALUE;
    DFC0 = 0U; /* digital filter off */
    IICCTL01 |= _01_IICA_fCLK_HALF;
    SVA0 = 10_IICA0_SLAVEADDRESS;
    STCEN0 = 1U;
    IICRSV0 = 1U;
    SPIE0 = 0U;
    WTIM0 = 1U;
    ACKE0 = 1U;
    IICAMK0 = 0U;
    IICE0 = 1U;
     LREL0 = 1U;
     /* Set SCLA0, SDAA0 pin */
    PM1 &= 0xFCU;
}

/******************************************************************************
* Function Name: R_IICA0_Stop
* Description  : This function stops IICA0 module operation.
* Arguments    : None
* Return Value : None
******************************************************************************/
void R_IICA0_Stop(void)
{
    IICE0 = 0U;    /* disable IICA0 operation */
}

/******************************************************************************
* Function Name: R_IICA0_Slave_Send
* Description  : This function sends data as slave mode.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : None
******************************************************************************/
void R_IICA0_Slave_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    g_iica0_tx_cnt = tx_num;
    gp_iica0_tx_address = tx_buf;
    g_iica0_slave_status_flag = 0U;
}

/******************************************************************************
* Function Name: R_IICA0_Slave_Receive
* Description  : This function receives data as slave mode.
* Arguments    : rx_buf -
*                    receive buffer pointer
*                rx_num -
*                    buffer size
* Return Value : None
******************************************************************************/
void R_IICA0_Slave_Receive(uint8_t * const rx_buf, uint16_t rx_num)
{
    g_iica0_rx_len = rx_num;
    g_iica0_rx_cnt = 0U;
    gp_iica0_rx_address = rx_buf;
    g_iica0_slave_status_flag  = 0U;
}
```

```
/***********************************************************************
* File Name    : r_cg_serial_user.c
* Version      : CodeGenerator for RL78/I1A V2.00.00.04 [21 Feb 2013]
* Device(s)    : R5F107AE
* Tool-Chain   : CA78K0R
* Description  : This file implements device driver for Serial module.
* Creation Date: 13/9/2013
***********************************************************************/

/***********************************************************************
Pragma directive
***********************************************************************/
#pragma interrupt INTIICA0 r_iica0_interrupt

/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_serial.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************
Global variables and functions
***********************************************************************/
extern volatile uint8_t   g_iica0_master_status_flag;  /* iica0 master flag */
extern volatile uint8_t   g_iica0_slave_status_flag;   /* iica0 slave flag */
extern volatile uint8_t * gp_iica0_rx_address;         /* iica0 receive buffer address
*/
extern volatile uint16_t  g_iica0_rx_cnt;              /* iica0 receive data length */
extern volatile uint16_t  g_iica0_rx_len;              /* iica0 receive data count */
extern volatile uint8_t * gp_iica0_tx_address;         /* iica0 send buffer address */
extern volatile uint16_t  g_iica0_tx_cnt;              /* iica0 send data count */
uint8_t rx_end = 0;                                      //rx end flag
uint8_t char tx_end = 0;                                    //tx end flag
extern uint8_t  g_read_value[250];                       //buffer for data read
extern uint8_t  g_write_value[250];                      //buffer for data write

/***********************************************************************
* Function Name: r_iica0_interrupt
* Description  : This function is INTIICA0 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************/
__interrupt static void r_iica0_interrupt(void)
{
    P3 = 0x08;
    if ((IICS0 & _80_IICA_STATUS_MASTER) == 0U)
    {
        iica0_slave_handler();
    }
     P3 = 0x00;
}
```

```
/******************************************************************************
* Function Name: iica0_slave_handler
* Description  : This function is IICA0 slave handler.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void iica0_slave_handler(void)
{
    /* Control for stop condition */
    if (1U == SPD0)
    {
        /* Get stop condition */
        SPIE0 = 0U;
        g_iica0_slave_status_flag = 1U;
    }
    else
    {
  if(1U == STD0)/* start or restart condition*/
   g_iica0_slave_status_flag = 0U;  //Reset slave status flag

        if ((g_iica0_slave_status_flag & _80_IICA_ADDRESS_COMPLETE) == 0U)
        {
            if (1U == COI0)
            {
                SPIE0 = 1U;
                g_iica0_slave_status_flag |= _80_IICA_ADDRESS_COMPLETE;

                if (1U == TRC0)
                {
                    WTIM0 = 1U;

                    if (g_iica0_tx_cnt > 0U)
                    {
                        IICA0 = *gp_iica0_tx_address;
                        gp_iica0_tx_address++;
                        g_iica0_tx_cnt--;
                    }
                    else
                    {
                        r_iica0_callback_slave_sendend();
                        WREL0 = 1U;
                    }
                }
                else
                {
                    ACKE0 = 1U;
                    WTIM0 = 0U;
                    WREL0 = 1U;
                }
            }
            else
            {
                r_iica0_callback_slave_error(MD_ERROR);
            }
        }
        else
        {
            if (1U == TRC0)
            {
                if ((0U == ACKD0) && (g_iica0_tx_cnt != 0U))
                {
                    r_iica0_callback_slave_error(MD_NACK);
                }
                else
                {
                    if (g_iica0_tx_cnt > 0U)
                    {
                        IICA0 = *gp_iica0_tx_address;
                        gp_iica0_tx_address++;
                        g_iica0_tx_cnt--;
                    }
                    else
                    {
                        r_iica0_callback_slave_sendend();
                        WREL0 = 1U;
                    }
                }
            }
            else
            {
                if (g_iica0_rx_cnt < g_iica0_rx_len)
                {
                    *gp_iica0_rx_address = IICA0;
                    gp_iica0_rx_address++;
                    g_iica0_rx_cnt++;

                    if (g_iica0_rx_cnt == g_iica0_rx_len)
                    {
                        ACKE0 = 0U;
                        WTIM0 = 1U;
                        WREL0 = 1U;
                        r_iica0_callback_slave_receiveend();
                    }
                    else
                    {
                        WREL0 = 1U;
                    }
                }
                else
                {
                    WREL0 = 1U;
                }
            }
        }
    }
}
```

```
/******************************************************************************
Function Name: r_iica0_callback_slave_error
* Description  : This function is a callback function when IICA0 slave error occurs.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void r_iica0_callback_slave_error(MD_STATUS flag)
{
    /* Start user code. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}

/******************************************************************************
Function Name: r_iica0_callback_slave_receiveend
* Description  : This function is a callback function when IICA0 finishes slave
reception.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void r_iica0_callback_slave_receiveend(void)
{
    rx_end = 1;
}

/******************************************************************************
Function Name: r_iica0_callback_slave_sendend
* Description  : This function is a callback function when IICA0 finishes slave
transmission.
* Arguments    : None
* Return Value : None
******************************************************************************/
static void r_iica0_callback_slave_sendend(void)
{
    tx_end = 1;
}
```

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

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

s

Revision History

		Description	
Revision	**Date**	**Page**	**Summary**
1.00	Jan 25, 2014	—	First release version

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.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 LanGao Rd., Putuo District, Shanghai, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141