RX Family

How to Start Transmission Immediately After Writing Transmit Data in SCI Asynchronous Mode

Introduction

This application note describes methods of starting transmission immediately after writing transmit data in asynchronous mode of the serial communication interface (SCI).

Target Devices

RX Family

Target Tools

- Smart Configurator V2.19.0 or earlier
- e2 studio Version 2023-10 or earlier

Please use the Smart Configurator implemented in e2 studio 2024-01 and Smart Configurator V2.20.0 or later version because equivalent functions have been implemented in Smart Configurator.

Confirmed Devices

RX660 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Contents

1. Wait Time in the SCI

1.1 Overview of the Wait Time

In asynchronous mode, data transmission starts when transmit data is written to the TDR register after the SCR.TE bit is set to 1.

However, after the SCR.TE bit is set to 1, an internal wait time that is equal to the time required to transmit one frame of data is generated before data transmission starts, and this internal wait time occurs each time the SCR.TE bit is set to 1.

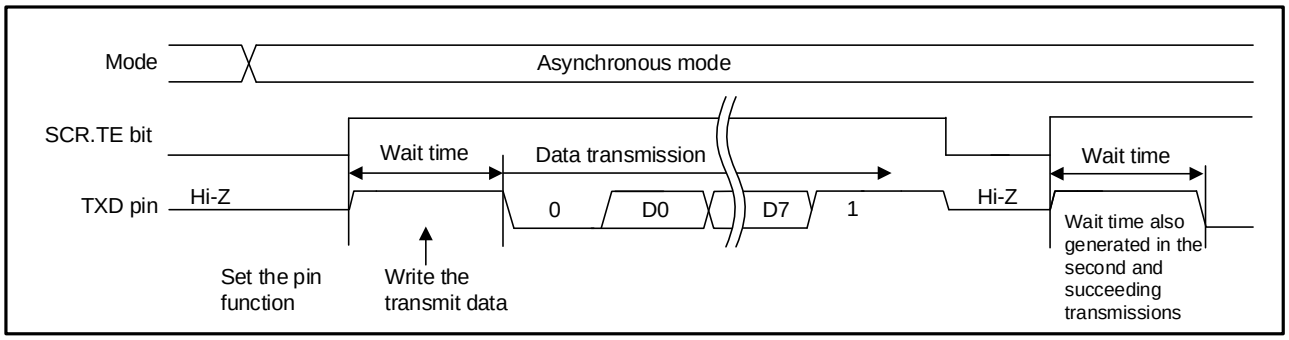


Figure 1.1 Internal Wait Time in Data Transmission

1.2 Instant Transmission Function

In some products that support the SEMR.ITE bit, the SCI can be used without generating internal wait times by controlling this bit.

1.3 Overview of the Code Generation Function of Smart Configurator

The SCI components generated by the code generation function support processing equivalent to the flowchart example for serial transmission in asynchronous mode described in the user's manual.

In the serial transmission flowchart example, the transmission process is controlled by the TE bit, so an internal wait time occurs when the SCI components are used.

In this application note, the SCI components generated by the code generation function are used for a sample program, so the processing has been modified not to generate internal wait times. Sections 3 and 4 describe modification procedures.

2. Operation Confirmation Conditions

Table 2.1 Operation Confirmation Conditions

Item	Description
MCU used	R5F56609HDFB (RX660N Group)
Operating frequency	• Main clock: 24 MHz • PLL: 240 MHz (Main clock, divided by 1, multiplied by 10) • System clock (ICLK): 120 MHz (PLL divided by 2) • Peripheral module clock B (PCLKB): 60 MHz (PLL divided by 4)
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio Version 2023-01 (23.1.0)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00
	Compile options -lang = c99
iodefine.h version	Version 1.00
Endian order	Little endian
Operating mode	Single-chip mode
Processor mode	Supervisor mode
Sample program version	Version 1.00
Board used	Renesas Starter Kit for RX660 (Product No.: RTK556609xxxxxxxxx)

3. Description of Software

This section describes how to start transmission immediately after writing transmit data. This sample program uses SCI components (SCIh module (SCI12)) that do not have an instant transmission function.

3.1 Control Example by Using the Transmit Data Empty Interrupt

3.1.1 Overview

An internal wait time is generated when the TE bit changes from 0 to 1 for the reason described in section 1.1. The modified method described in this section controls interrupts by using a 0-to-1 state change of the IEN bit as the trigger while maintaining the state of TE = 1, and writes transmit data within the interrupt process so that no internal wait time is generated in the second and subsequent transmissions.

Figure 3.1 is a timing chart before modification, and Figure 3.2 is a timing chart after modification.

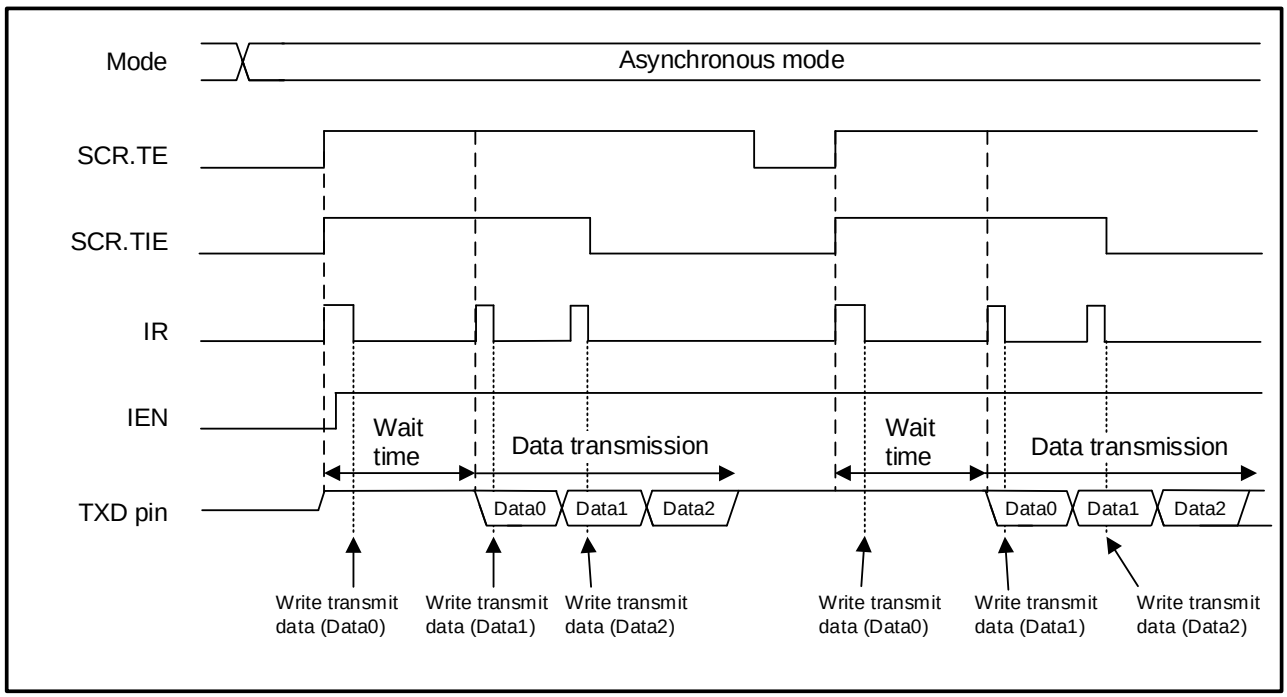


Figure 3.1 Timing Chart Before Modification

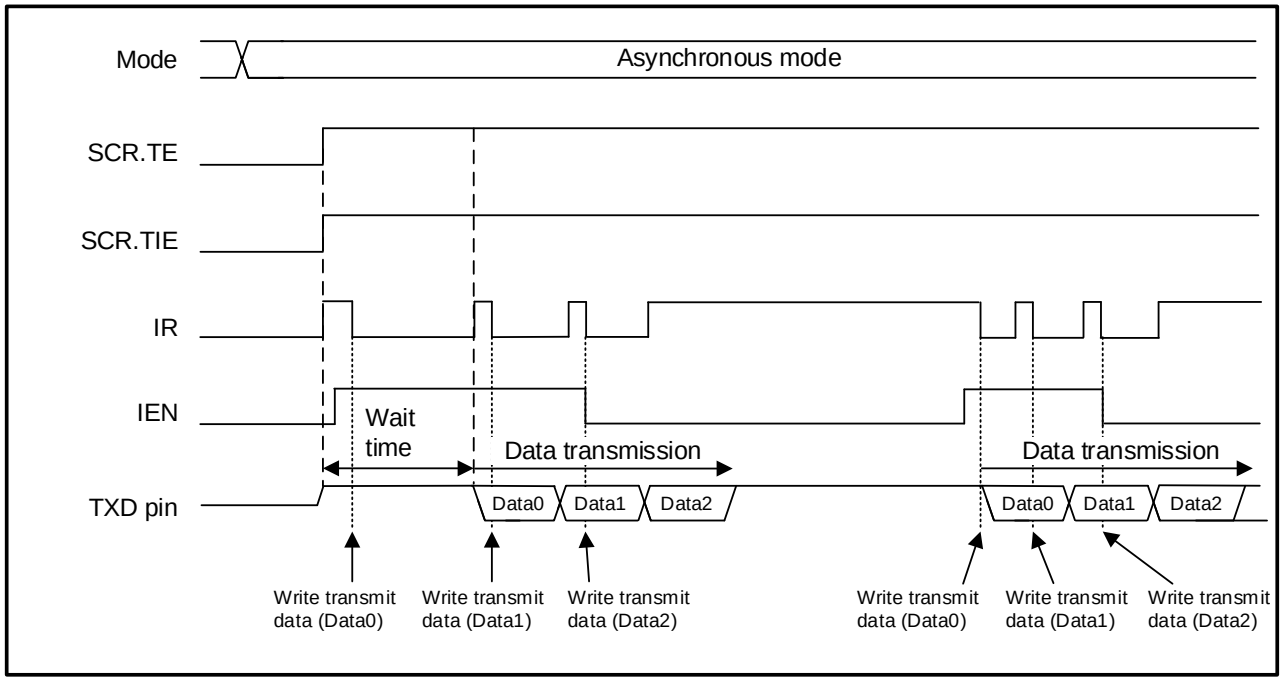
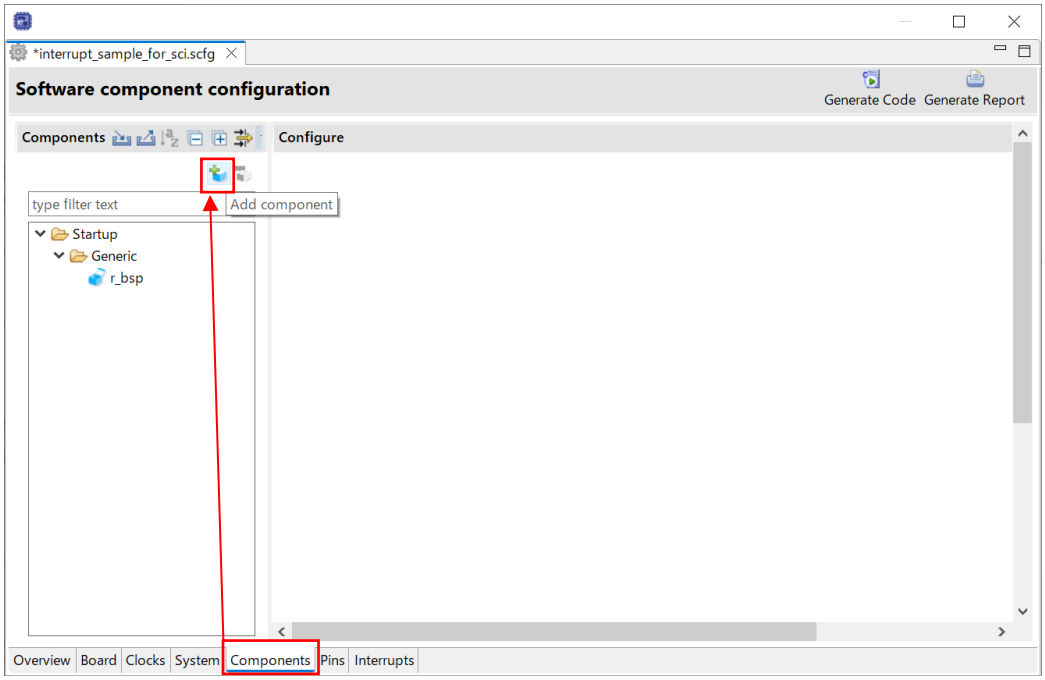


Figure 3.2 Timing Chart After Modification

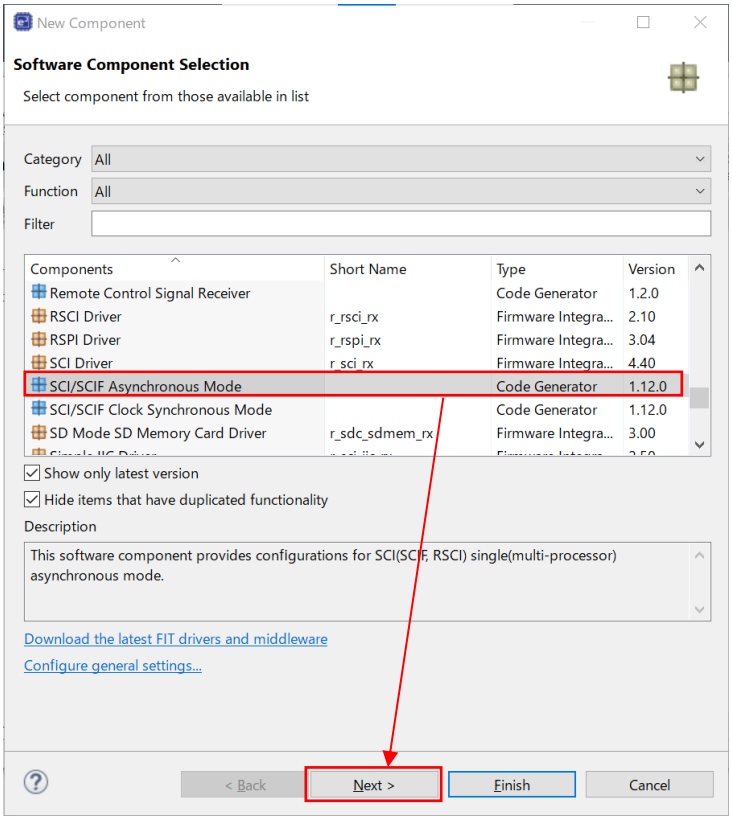
3.1.2 Code Generation Procedure Example

This section describes a code generation procedure example in Smart Configurator.

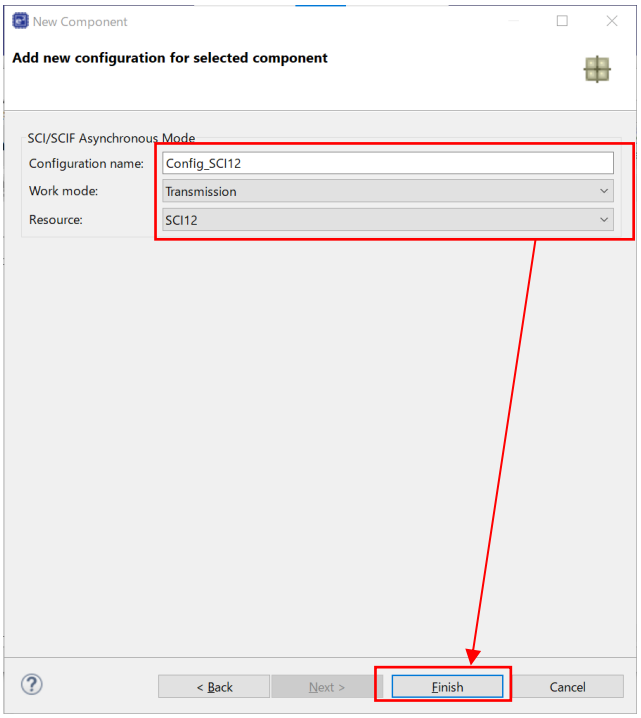
1. Open the [Components] tab and click on the [Add component] icon.



2. In the [Software Component Selection] window, select [SCI (SCIF) Asynchronous Mode], and then click on [Next].

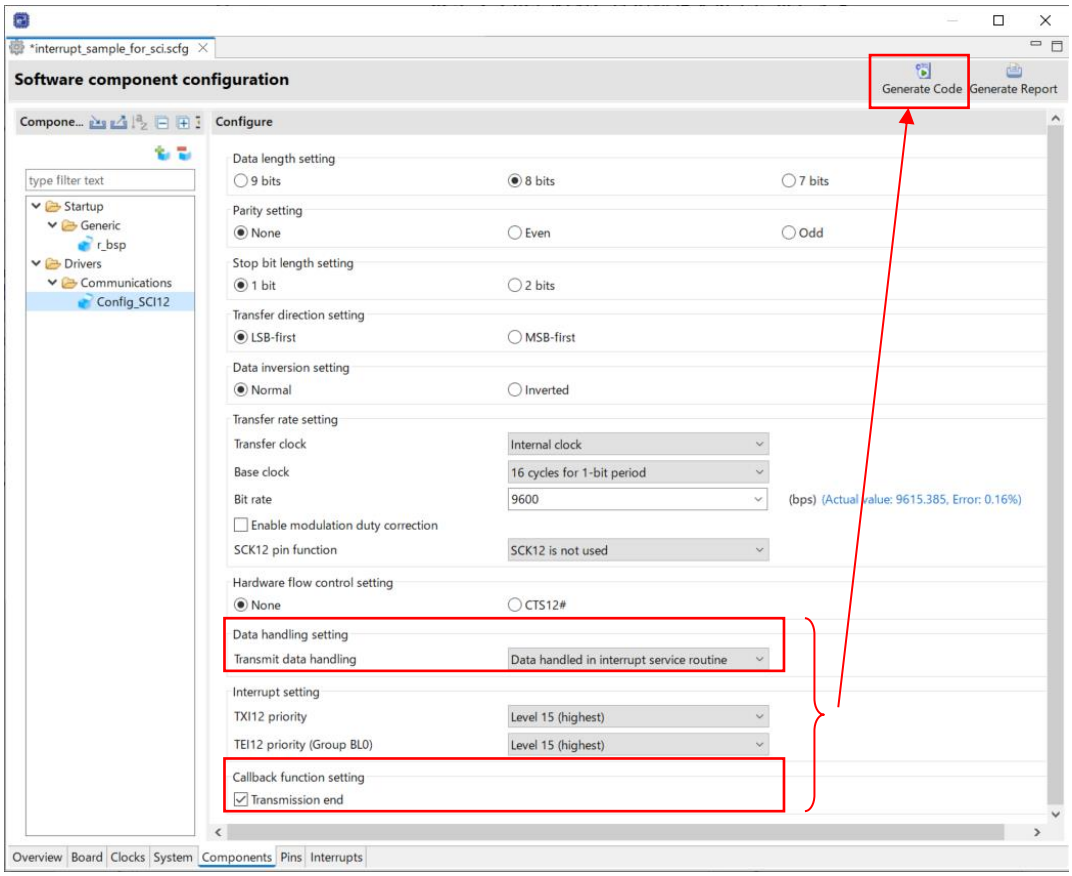


3. Specify an appropriate configuration name, work mode, and resource, and then click on [Finish].
   In this sample, the configuration name is "Config_SCI12", the work mode is "Transmission", and the resource is "SCI12".

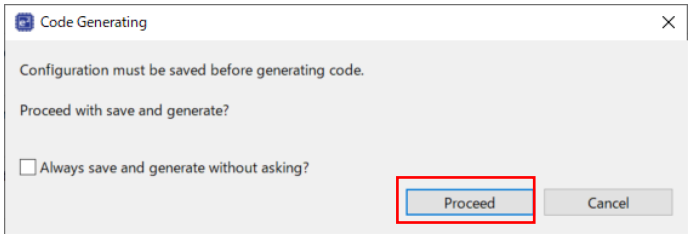


4. Configure software component settings for SCI12, and then click on the [Generate Code] icon.
   The required settings for this sample are listed below.

   Transmit data handling: Data handled by the interrupt service routine
   Callback function setting: "Transmission end" selected



5. A message box for code generation appears. Click on [Proceed].



3.1.3 Modifying the Program

Table 3.1 lists the functions to be modified.

Table 3.1 Functions to Be Modified

Function Name
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
r_Config_SCI12_transmit_interrupt
r_Config_SCI12_transmitend_interrupt

R_Config_SCI12_Start ()

Before modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    IEN(SCI12, TXI12) = 1U;
    ICU.GENBL0.BIT.EN16 = 1U;
}
```

To be deleted because IEN is set to 1 at the start of transmission by the R_Config_SCI12_Serial_Send function

After modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    ICU.GENBL0.BIT.EN16 = 1U;
    SCI12.SCR.BYTE |= 0xA0U;

   /* Set TXD12 pin */
    PORTA.PMR.BYTE |= 0x10U;

}
```

To prevent a 0-to-1 TE state change from working as a trigger, set TE = 1 by this function to inhibit control by the R_Config_SCI12_Serial_Send function.

Although TE = 1 sets IR of TXI12 to 1, no interrupt is generated to the CPU because IEN = 0. The interrupt is held pending.

Also, set the TXD12 pin when setting TE = 1.

R_Config_SCI12_Serial_Send ()

Before modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    if (1U > tx_num)
    {
        status = MD_ARGERROR;
    }
    else if (0U == IEN(SCI12, TXI12))
    {
        status = MD_ERROR;
    }
    else
    {
        gp_sci12_tx_address = tx_buf;
        g_sci12_tx_count = tx_num;
        IEN(SCI12, TXI12) = 0U;
        SCI12.SCR.BYTE |= 0xA0U;

        /* Set TXD12 pin */
        PORTA.PMR.BYTE |= 0x10U;
        IEN(SCI12, TXI12) = 1U;
    }

    return (status);
}
```

To be deleted because the R_Config_SCI12_Start function does not set IEN

To be deleted because this function is executed when IEN is 0

To be deleted because SCI12.SCR.BYTE and the TXD12 pin are set by the R_Config_SCI12_Start function

After modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    if (1U > tx_num)
    {
        status = MD_ARGERROR;
    }
    else
    {
        gp_sci12_tx_address = tx_buf;
        g_sci12_tx_count = tx_num;

        IEN(SCI12, TXI12) = 1U;
    }

    return (status);
}
```

Setting IEN = 1 enables the TXI12 interrupt that has been held pending.

r_Config_SCI12_transmit_interrupt ()

Before modification:

```
static void r_Config_SCI12_transmit_interrupt(void)
{
    if (0U < g_sci12_tx_count)
    {
        SCI12.TDR = *gp_sci12_tx_address;
        gp_sci12_tx_address++;
        g_sci12_tx_count--;
    }
    else
    {
        SCI12.SCR.BIT.TIE = 0U;
        SCI12.SCR.BIT.TEIE = 1U;
    }
}
```

Interrupts are to be controlled by changing this to "if (1U < g_sci12_tx_count)" to change IEN from 0 to 1 after transmitting the last data because IEN must change from 1 to 0 after finishing transmission.

IEN is to change from 1 to 0 with "else if(1U == g_sci12_tx_count)" after writing the last data.

The conditional expressions must be changed because interrupts must be disabled after transmitting the last data. "else" is unnecessary because the processing after transmitting the last data changes.

After modification:

```
static void r_Config_SCI12_transmit_interrupt(void)
{
    if (1U < g_sci12_tx_count)
    {
        SCI12.TDR = *gp_sci12_tx_address;
        gp_sci12_tx_address++;
        g_sci12_tx_count--;
    }
    else if(1U == g_sci12_tx_count)
    {
        SCI12.TDR = *gp_sci12_tx_address;
        gp_sci12_tx_address--;
        IEN(SCI12, TXI12) = 0U;
        SCI12.SCR.BIT.TEIE = 1U;
    }
    else
    {
        nop(); /* Actions not normally performed */
    }
}
```

Except for the last data, the same processing as before the change is performed.

Interrupts are disabled after transmitting the last data.

Although this processing is usually not performed, it is added as exception handling.

r_Config_SCI12_transmitend_interrupt ()

Before modification:

```
void r_Config_SCI12_transmitend_interrupt(void)
{
    /* Set TXD12 pin */
    PORTA.PMR.BYTE &= 0xEFU;

    SCI12.SCR.BIT.TIE = 0U;
    SCI12.SCR.BIT.TE = 0U;
    SCI12.SCR.BIT.TEIE = 0U;

    r_Config_SCI12_callback_transmitend();
}
```

To be deleted to maintain the transmission setting (TE = 1)

After modification:

```
void r_Config_SCI12_transmitend_interrupt(void)
{
    SCI12.SCR.BIT.TEIE = 0U;

    r_Config_SCI12_callback_transmitend();
}
```

3.1.4 Sample Program

The following shows how the sample program works.

[Main function]
(1) Call the R_Config_SCI12_Start function to configure transmission.
(2) Set g_flag = 0.
(3) Call the R_Config_SCI12_Serial_Send function to transmit 3-byte data {0x11, 0x22, 0x33}.
(4) Wait until transmission is completed. (Wait until g_flag = 1 is set by the transmission end interrupt.)
(5) Insert 300-µs wait time.
(6) Repeat steps (2) to (5) twice.
(7) After transmission is completed, call the R_Config_SCI12_Stop function to disable the transmission setting.

Note: Since the R_Config_SCI12_Create function is called in the R_Systeminit function before reaching the main function, it does not need to be called in the main function.

```
/***********************************************
Global variables
***********************************************/
volatile uint8_t g_flag;

/***********************************************
Private (static) variables and functions
***********************************************/
uint8_t          g_data[] = {0x11, 0x22, 0x33};

void main(void);

/***********************************************
* Function Name: main
* Description  : This function uses the modified program to send data without a wait time.
* Arguments : None
* Return Value : None
***********************************************/
void main(void)
{
    uint8_t i           = 0;
    uint8_t send_count = 2;

    R_Config_SCI12_Start();

    for (i = 0; i < send_count; i++)
    {
        g_flag = 0;
        R_Config_SCI12_Serial_Send(g_data, 3);

        while (0 == g_flag)
        {
            nop();
        }
        R_BSP_SoftwareDelay((uint32_t)300, BSP_DELAY_MICROSECS);
    }

    R_Config_SCI12_Stop();

    while (1)
    {
        nop();
    }
}
```

Figure 3.3 "main" Function of the Sample Program

[SCI12 transmission end callback function]

(1) Set the transmission end flag (g_flag = 1).

```
static void r_Config_SCI12_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI12_callback_transmitend. Do not edit comment
generated here */
    g_flag = 1;
    /* End user code. Do not edit comment generated here */
}
```

Figure 3.4 "r_Config_SCI12_callback_transmitend" Function in the Sample Program

3.1.5 Operation After Modification

Figure 3.5 and Figure 3.6 show the operation waveforms before and after modification when 3-byte data {0x11, 0x22, 0x33} is transmitted.

Operation before modification

Figure 3.5 Waveform Before Modification

Before the modification, it can be confirmed that an internal wait time occurs when TE changes from 0 to 1.

Operation after modification:

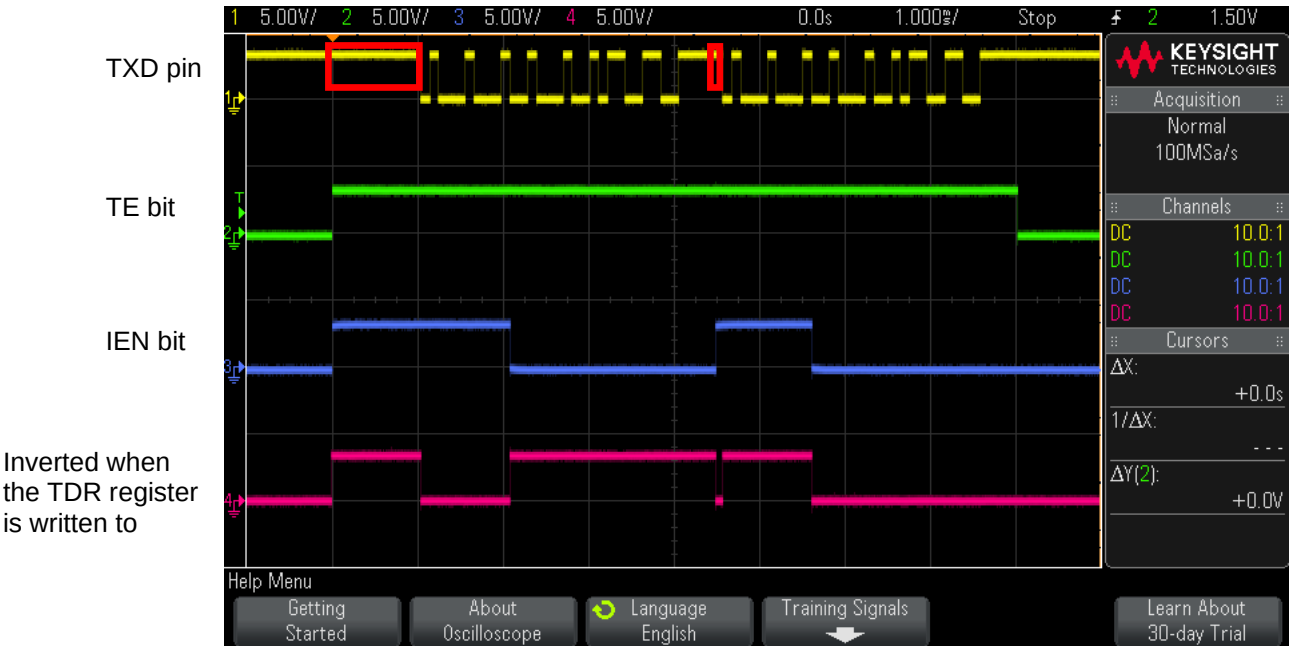


Figure 3.6 Waveform After Modification

It can be confirmed that during the first transmission, an internal wait time occurs when TE changes from 0 to 1, but during the second transmission, the TE bit is not controlled, so the transmission starts without generating an internal wait time. For the first transmission, no internal wait time occurs either if the R_Config_SCI12_Serial_Send function is executed at least one frame after the R_Config_SCI12_Start function is called.

3.2 Control Example Using the DMAC

3.2.1 Overview

As in section 3.1, Control Example by Using the Transmit Data Empty Interrupt, this section describes the case where the TE = 1 state is maintained and transmit data is written by DMAC transfer.

Figure 3.7 is a timing chart after modification (writing transmit data by DMAC transfer).

(Timings before modification are equivalent to those in Figure 3.1 except for DMAC operation.)

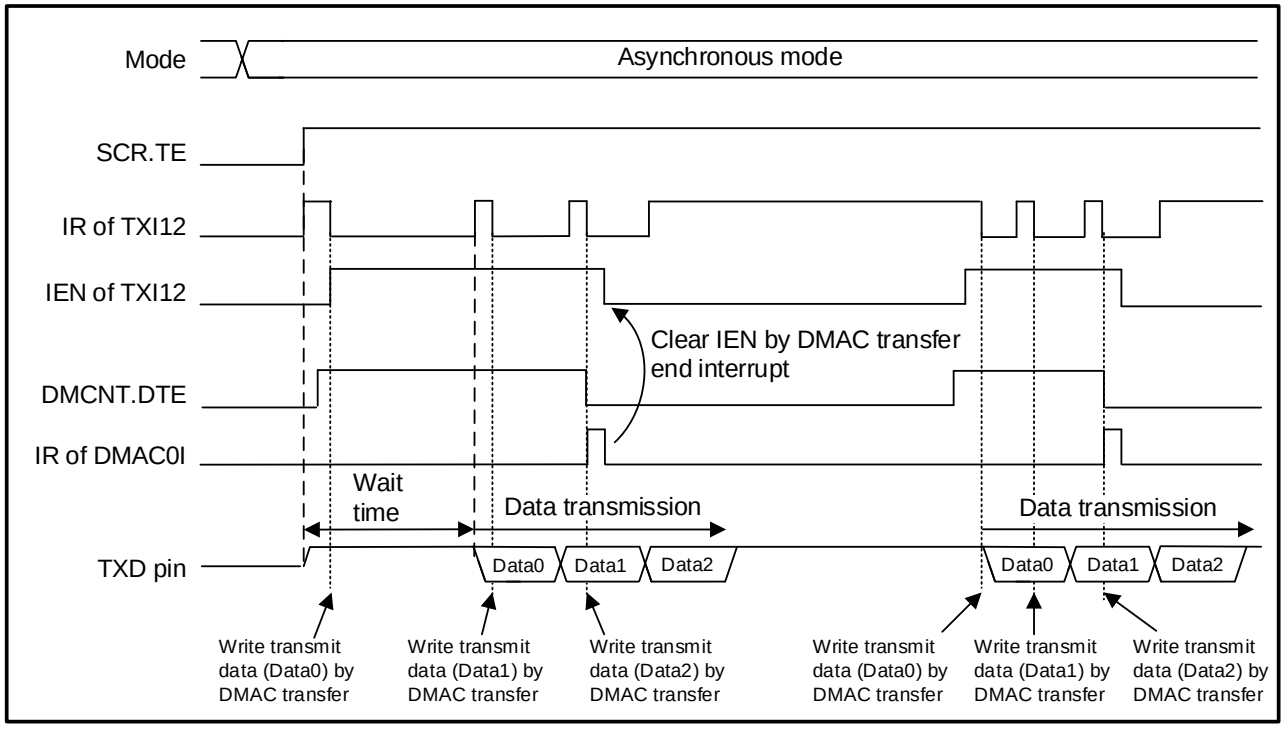


Figure 3.7 Timing Chart After Modification (Writing Transmit Data by DMAC Transfer)

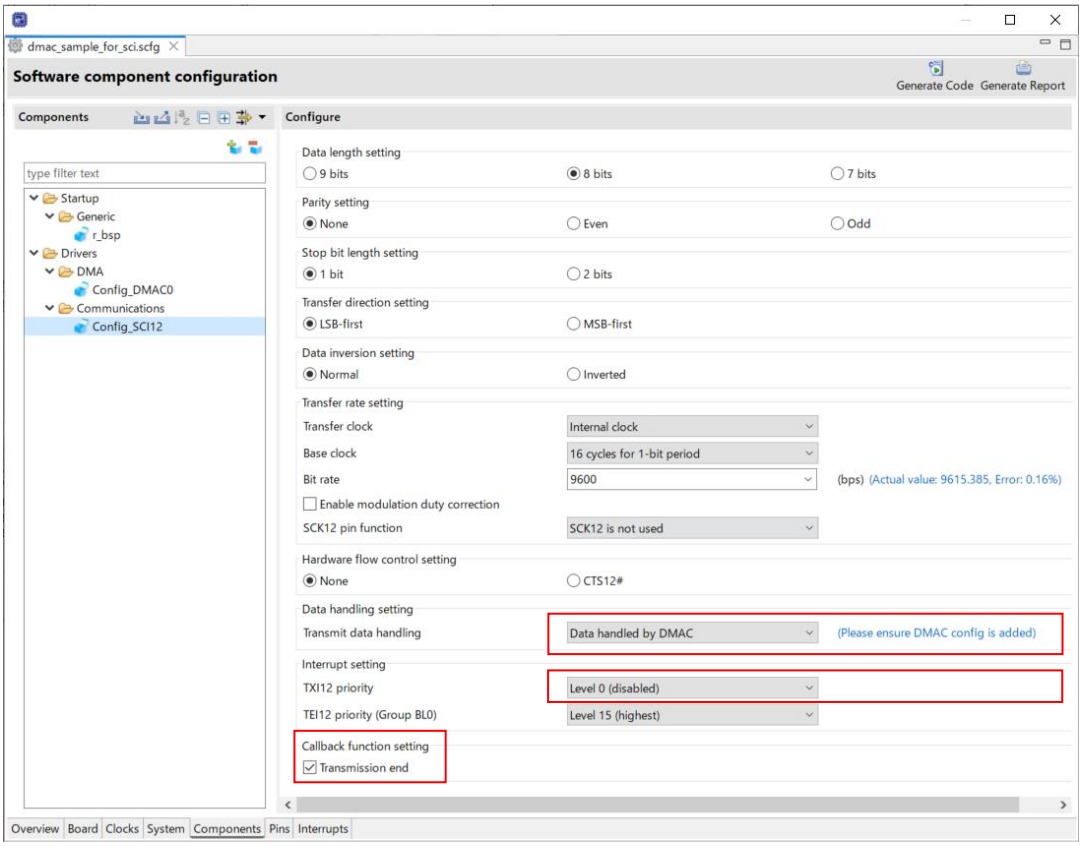
3.2.2 Code Generation Procedure Example

Code generation is performed using Smart Configurator in the same way as in the code generation procedure example in section 3.1, Control Example by Using the Transmit Data Empty Interrupt.

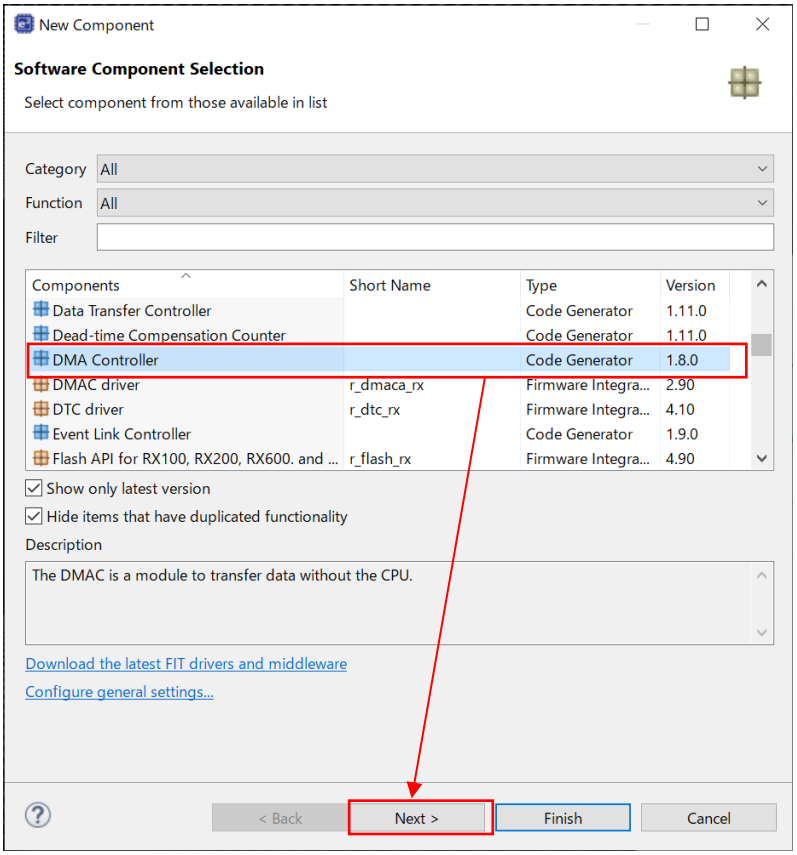
1. Perform the same procedure as steps 1 through 3 in section 3.1.2, Code Generation Procedure Example.

2. Configure software component settings for SCI12, and then click on the [Generate Code] icon.
   The required settings for this sample are listed below.

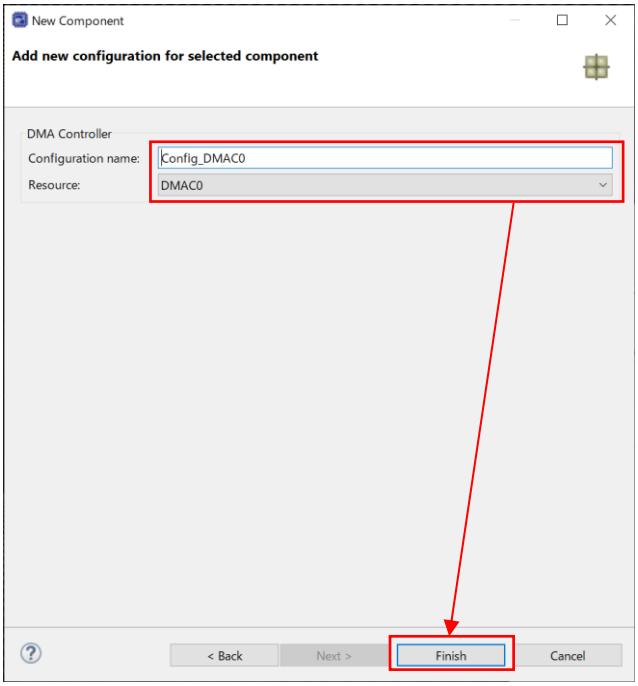
   Transmit data handling: Data handled by DMAC
   TXI12 priority: Level 0 (disabled)
   Callback function setting: "Transmission end" selected



3. Go to the [Software Component Selection] window again. Select [DMA Controller], and then click on [Next].

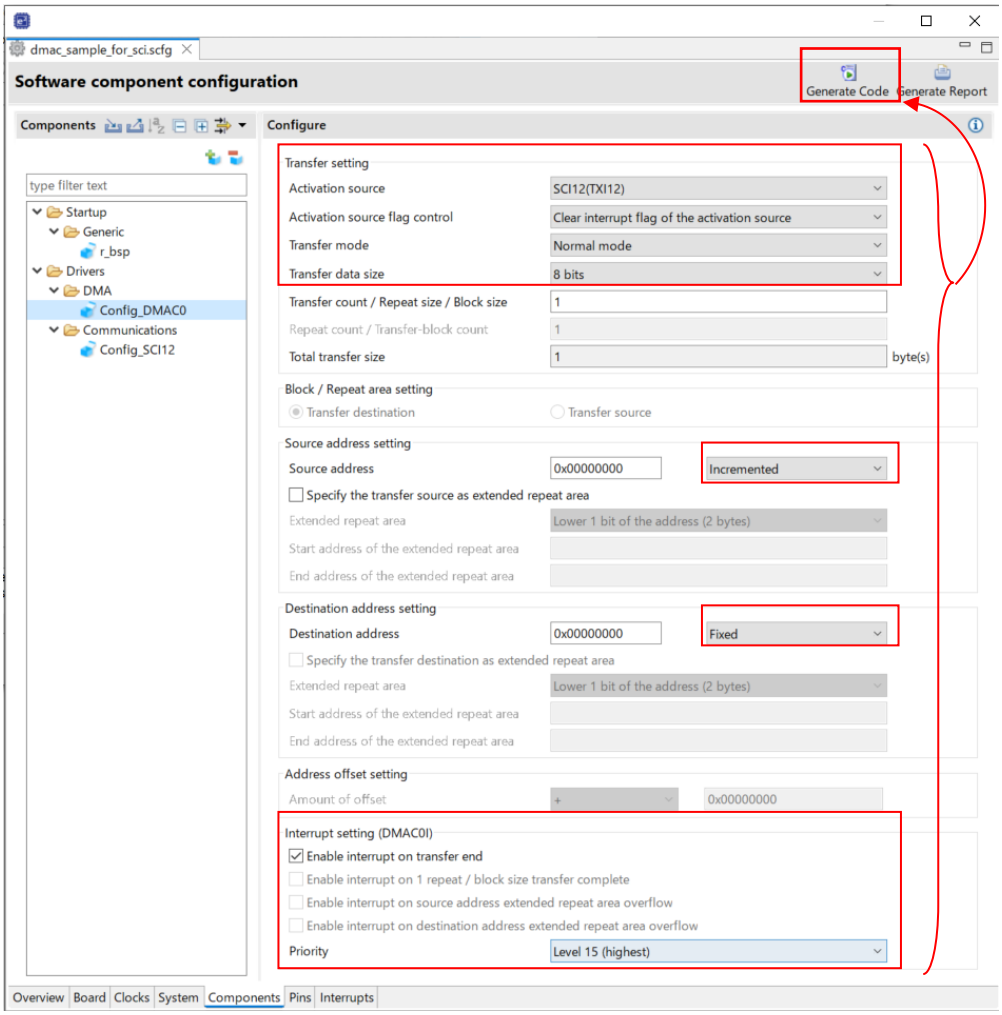


4. Specify an appropriate configuration name and resource, and then click on [Finish].
In this sample, the configuration name is "Config_DMAC0" and the resource is "DMAC0".



5. Configure software component settings for DMAC0, and then click on the [Generate Code] icon.
   The required settings for this sample are listed below.

   Activation source: SCI12 (TXI12)
   Activation source flag control: Clear interrupt flag of the activation source
   Transfer mode: Normal mode
   Transfer data size: 8 bits
   Source address: Incremented
   Destination address: Fixed
   Enable interrupt on transfer end: Selected
   Priority: Level 15 (highest)



Note: The transfer count, source address, and destination address are to be changed by software, so here they are left at their default values.

3.2.3 Modifying the Program

Table 3.2 lists the functions to be modified.

Table 3.2 Functions to Be Modified

Function Name
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
R_Config_DMAC0_Create
R_Systeminit

R_Config_SCI12_Start ()

Before modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    IEN(SCI12, TXI12) = 1U;
    ICU.GENBL0.BIT.EN16 = 1U;
}
```

To be deleted because IEN is set to 1 at the start of transmission by the R_Config_SCI12_Serial_Send function

After modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    ICU.GENBL0.BIT.EN16 = 1U;
    SCI12.SCR.BYTE |= 0xA0U;

    /* Set TXD12 pin */
    PORTA.PMR.BYTE |= 0x10U;

}
```

To prevent a 0-to-1 TE state change from working as a trigger, set TE = 1 by this function to inhibit control by the R_Config_SCI12_Serial_Send function.

Although TE = 1 sets IR of TXI12 to 1, no DMAC transfer request is generated because IEN = 0. The request is held pending.

Also, set the TXD12 pin when setting TE = 1.

R_Config_SCI12_Serial_Send ()

Before modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    SCI12.SCR.BYTE |= 0xA0U;
    /* Set TXD12 pin */
    PORTA.PMR.BYTE |= 0x10U;

    return MD_OK;
}
```

To be deleted because SCI12.SCR.BYTE and the TXD12 pin are set by the R_Config_SCI12_Start function

After modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    IEN(SCI12, TXI12) = 1U;

    return MD_OK;
}
```

Set IEN to 1 to enable TXI interrupt that is the activation source for the DMAC.

R_Config_DMAC0_Create ()

Before modification:

```
void R_Config_DMAC0_Create(void)
{
    /* Cancel DMAC/DTC module stop state in LPC */
    MSTP(DMAC) = 0U;

    /* Disable DMAC interrupts */
    IEN(DMAC,DMAC0I) = 0U;

    /* Disable DMAC0 transfer */
    DMAC0.DMCNT.BIT.DTE = 0U;

    /* Set DMAC0 activation source */
    ICU.DMRSR0 = _75_DMAC0_ACTIVATION_SOURCE;

    /* Set DMAC0 transfer address update and extended repeat setting */
    DMAC0.DMAMD.WORD = _8000_DMAC_SRC_ADDR_UPDATE_INCREMENT |
                       _0000_DMAC_DST_ADDR_UPDATE_FIXED |
                           _0000_DMAC0_SRC_EXT_RPT_AREA |
_0000_DMAC0_DST_EXT_RPT_AREA;

    /* Set DMAC0 transfer mode, data size and repeat area */
    DMAC0.DMTMD.WORD = _0000_DMAC_TRANS_MODE_NORMAL |
_2000_DMAC_REPEAT_AREA_NONE |
                       _0000_DMAC_TRANS_DATA_SIZE_8 |
                           _0001_DMAC_TRANS_REQ_SOURCE_INT;

    /* Set DMAC0 interrupt flag control */
    DMAC0.DMCSL.BYTE = _00_DMAC_INT_TRIGGER_FLAG_CLEAR;

    /* Set DMAC0 source address */
    DMAC0.DMSAR = (void *)_00000000_DMAC0_SRC_ADDR;

    /* Set DMAC0 destination address */
    DMAC0.DMDAR = (void *)_00000000_DMAC0_DST_ADDR;

    /* Set DMAC0 transfer count */
    DMAC0.DMCRA = _00000001_DMAC0_DMCRA_COUNT;

    /* Set DMAC0 interrupt settings */
    DMAC0.DMINT.BIT.DTIE = 1U;

    /* Set DMAC0 priority level */
    IPR(DMAC,DMAC0I) = _0F_DMAC_PRIORITY_LEVEL15;

    /* Enable DMAC activation */
    DMAC.DMAST.BIT.DMST = 1U;

    R_Config_DMAC0_Create_UserInit();
}
```

To be changed so that the transfer source address and the transfer count can be set as arguments.

The transfer source address and the transfer count are to be set from arguments.

The transfer destination address is to be changed to SCI12.TDR.

After modification:

```
void R_Config_DMAC0_Create(void *sar, uint16_t count)
{
    /* Cancel DMAC/DTC module stop state in LPC */
    MSTP(DMAC) = 0U;

    /* Disable DMAC interrupts */
    IEN(DMAC,DMAC0I) = 0U;

    /* Disable DMAC0 transfer */
    DMAC0.DMCNT.BIT.DTE = 0U;

    /* Set DMAC0 activation source */
    ICU.DMRSR0 = _75_DMAC0_ACTIVATION_SOURCE;

    /* Set DMAC0 transfer address update and extended repeat setting */
    DMAC0.DMAMD.WORD = _8000_DMAC_SRC_ADDR_UPDATE_INCREMENT |
                       _0000_DMAC_DST_ADDR_UPDATE_FIXED |
                           _0000_DMAC0_SRC_EXT_RPT_AREA |
_0000_DMAC0_DST_EXT_RPT_AREA;

    /* Set DMAC0 transfer mode, data size and repeat area */
    DMAC0.DMTMD.WORD = _0000_DMAC_TRANS_MODE_NORMAL |
_2000_DMAC_REPEAT_AREA_NONE |
                       _0000_DMAC_TRANS_DATA_SIZE_8 |
                           _0001_DMAC_TRANS_REQ_SOURCE_INT;

    /* Set DMAC0 interrupt flag control */
    DMAC0.DMCSL.BYTE = _00_DMAC_INT_TRIGGER_FLAG_CLEAR;

    /* Set DMAC0 source address */
    DMAC0.DMSAR = sar;

    /* Set DMAC0 destination address */
    DMAC0.DMDAR = (void *)(&(SCI12.TDR));

    /* Set DMAC0 transfer count */
    DMAC0.DMCRA = count;

    /* Set DMAC0 interrupt settings */
    DMAC0.DMINT.BIT.DTIE = 1U;

    /* Set DMAC0 priority level */
    IPR(DMAC,DMAC0I) = _0F_DMAC_PRIORITY_LEVEL15;

    /* Enable DMAC activation */
    DMAC.DMAST.BIT.DMST = 1U;

    R_Config_DMAC0_Create_UserInit();
}
```

Changed so that the transfer source address and the transfer count can be set as arguments.

The transfer source address and the transfer count are set from arguments.

The transfer destination address is changed to SCI12.TDR.

Modifying the prototype declaration in the Config_DMAC0.h file:

```
/***********************************************************************
Global functions
***********************************************************************/
void R_Config_DMAC0_Create(void *sar, uint16_t count);
```

R_Systeminit()

Before modification:

```
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and
software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE3.POECR2.WORD = 0x0000U;

    /* Initialize clocks settings */
    R_CGC_Create();

    /* Set peripheral settings */
    R_Config_SCI12_Create();
    R_Config_DMAC0_Create();

    /* Set interrupt settings */
    R_Interrupt_Create();

    /* Register undefined interrupt */

    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT,
     (bsp_int_cb_t)r_undefined_exception);

    /* Register group BL0 interrupt TEI12 (SCI12) */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_TEI12,
     (bsp_int_cb_t)r_Config_SCI12_transmitend_interrupt);

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

The initial settings are to be deleted because DMAC settings will be changed to such a specification that the transmit buffer and the transfer count are set before transmission from the SCI.

After modification:

```
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and
software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE3.POECR2.WORD = 0x0000U;

    /* Initialize clocks settings */
    R_CGC_Create();

    /* Set peripheral settings */
    R_Config_SCI12_Create();

    /* Set interrupt settings */
    R_Interrupt_Create();

    /* Register undefined interrupt */

    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT,
     (bsp_int_cb_t)r_undefined_exception);

    /* Register group BL0 interrupt TEI12 (SCI12) */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_TEI12,
     (bsp_int_cb_t)r_Config_SCI12_transmitend_interrupt);

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

3.2.4 Sample Program

The following shows how the sample program works.

[Main function]

(1) Call the R_Config_SCI12_Start function to configure transmission.
(2) Call the R_Config_DMAC_Create function to set transmit data for the DMAC transfer source, the SCI transmit buffer for the transfer destination, and the transfer count.
(3) Set g_flag = 0, and set g_flag = 1 only after transmission is complete to prevent further processing until transmission is complete.
(4) Execute the R_Config_DMAC_Start function to put the DMAC in a state waiting for transfer.
(5) Call the R_Config_SCI12_Serial_Send function to enable the TXI interrupt that works as an activation source for the DMAC. DMAC transfer occurs at the timing of the TXI interrupt and transmit data is written to the transmit buffer.
(6) Wait until transmission is completed.
(7) Insert 300-µs wait time.
(8) Repeat steps (2) to (7) twice.
(9) After transmission is completed, call the R_Config_DMAC_Stop function and the R_Config_SCI12_Stop function to disable the transmission setting.

Note: Since the R_Config_SCI12_Create function is called in the R_Systeminit function before reaching the main function, it does not need to be called in the main function.

```
/***********************************************
Global variables
***********************************************/
volatile uint8_t g_flag;

/***********************************************
Private (static) variables and functions
***********************************************/
uint8_t          g_data[] = {0x11, 0x22, 0x33};

void main(void);

/***********************************************
* Function Name: main
* Description  : This function uses the modified program to send data without a wait time.
* Arguments : None
* Return Value : None
***********************************************/
void main(void)
{
    uint8_t i           = 0;
    uint8_t send_count = 2;

    R_Config_SCI12_Start();

    for (i = 0; i < send_count; i++)
    {
        g_flag = 0;
       R_Config_DMAC0_Create((void *)g_data, 3 );
        R_Config_DMAC0_Start();
        R_Config_SCI12_Serial_Send(NULL, 0);

        while (0 == g_flag)
        {
            nop();
        }
        R_BSP_SoftwareDelay((uint32_t)300, BSP_DELAY_MICROSECS);
    }

    R_Config_DMAC0_Stop();
    R_Config_SCI12_Stop();

    while (1)
    {
        nop();
    }
}
```

Figure 3.8 "main" Function of the Sample Program

[DMAC0 transfer end callback function]

(1) The DMAC0 transfer end callback function (r_dmac0_callback_transfer_end) is called when all transmit data has been transferred to the SCI's transmit buffer.
(2) The function disables the TXI interrupt request and enables the transmission end interrupt (TEIE = 1).

```
/*****************************************
* Function Name: r_dmac0_callback_transfer_end
* Description  : This function is dmac0 transfer end callback function
* Arguments    : None
* Return Value : None
*****************************************/
static void r_dmac0_callback_transfer_end(void)
{
    /* Start user code for r_dmac0_callback_transfer_end. Do not edit comment
generated here */

  /* Interrupt processing when DMAC transfer is completed */
  IEN(SCI12, TXI12) = 0U;
  SCI12.SCR.BIT.TEIE = 1U;

  /* End user code. Do not edit comment generated here */
}
```

Figure 3.9 "r_dmac0_callback_transfer_end" Function in the Sample Program

[SCI12 transmission end callback function]

(1) The function disables the transmission end interrupt and sets the transmission end flag (g_flag = 1).
Note: Since the r_Config_SCI12_callback_transmitend function is also called during the TXI12 interrupt, it should only be processed when the transmission end interrupt is enabled.

```
/*****************************************
* Function Name: r_Config_SCI12_callback_transmitend
* Description  : This function is a callback function when SCI12 finishes
transmission
* Arguments    : None
* Return Value : None
*****************************************/
static void r_Config_SCI12_callback_transmitend(void)
{
  /* Start user code for r_Config_SCI12_callback_transmitend. Do not edit
comment generated here */
  if(1 == SCI12.SCR.BIT.TEIE)
  {
   /* Interrupt processing when SCI12 transmit end */
   SCI12.SCR.BIT.TEIE = 0U;
   g_flag = 1;
  }
  /* End user code. Do not edit comment generated here */
}
```

Figure 3.10 "r_dmac0_callback_transfer_end" Function in the Sample Program

3.2.5 Operation of the Sample Program

Figure 3.11 shows the waveforms when the control example using the DMAC is operated.

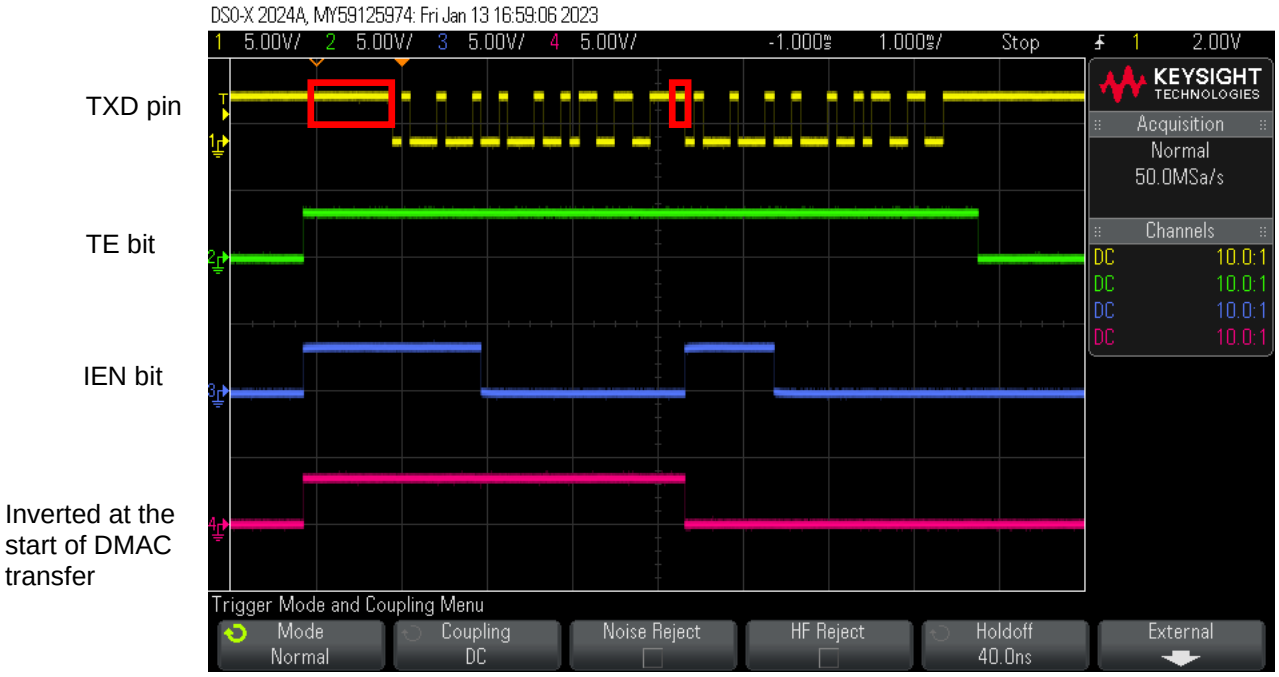


Figure 3.11 Waveforms of the Sample Program

It can be confirmed that during the first transmission, an internal wait time occurs when TE changes from 0 to 1, but during the second transmission, the TE bit is not controlled, so the transmission starts without generating an internal wait time. For the first transmission, no internal wait time occurs either if the R_Config_SCI12_Serial_Send function is executed at least one frame after the R_Config_SCI12_Start function is called.

3.3 Control Example Using the DTC

3.3.1 Overview

As in section 3.1, Control Example by Using the Transmit Data Empty Interrupt, this section describes the case where the TE = 1 state is maintained and transmit data is written by DTC transfer.

Figure 3.12 is a timing chart after modification (writing transmit data by DTC transfer).

(Timings before modification are equivalent to those in Figure 3.1 except for DTC operation.)

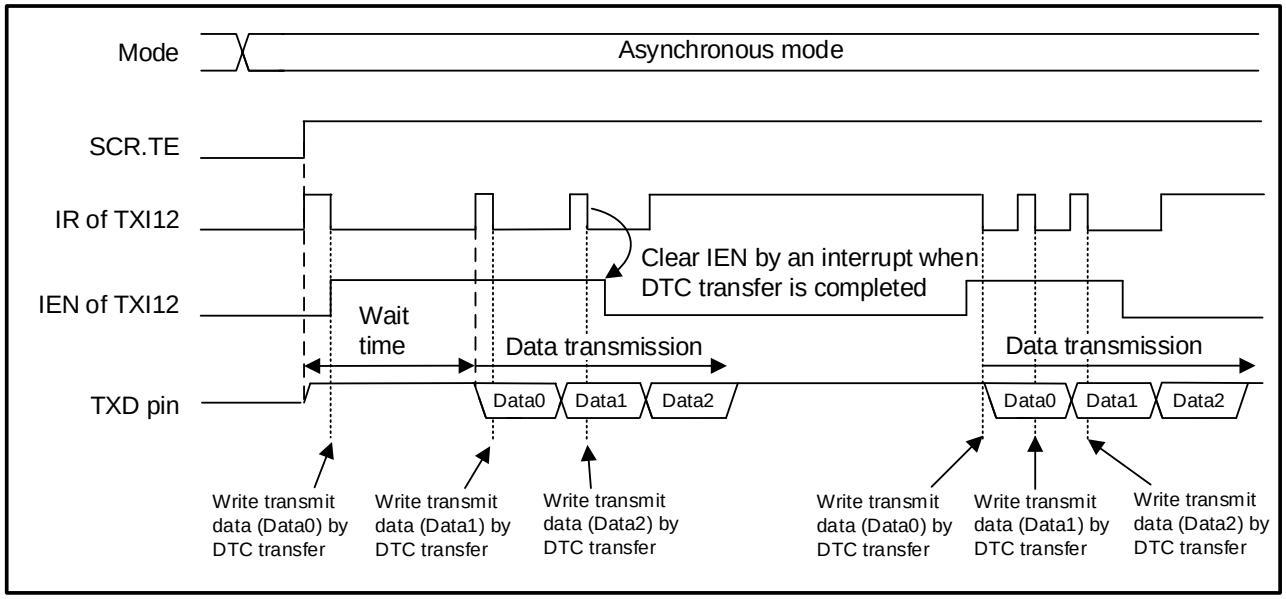


Figure 3.12 Timing Chart After Modification (Writing Transmit Data by DTC Transfer)

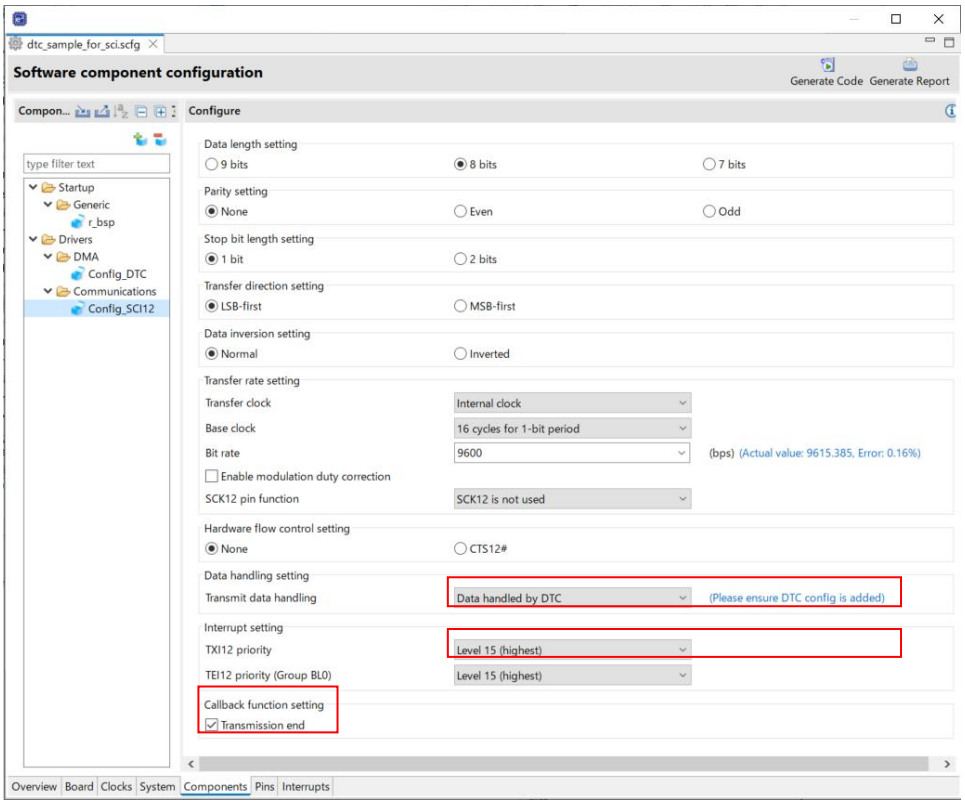
3.3.2 Code Generation Procedure Example

Code generation is performed using Smart Configurator in the same way as in the code generation procedure example in section 3.1, Control Example by Using the Transmit Data Empty Interrupt.

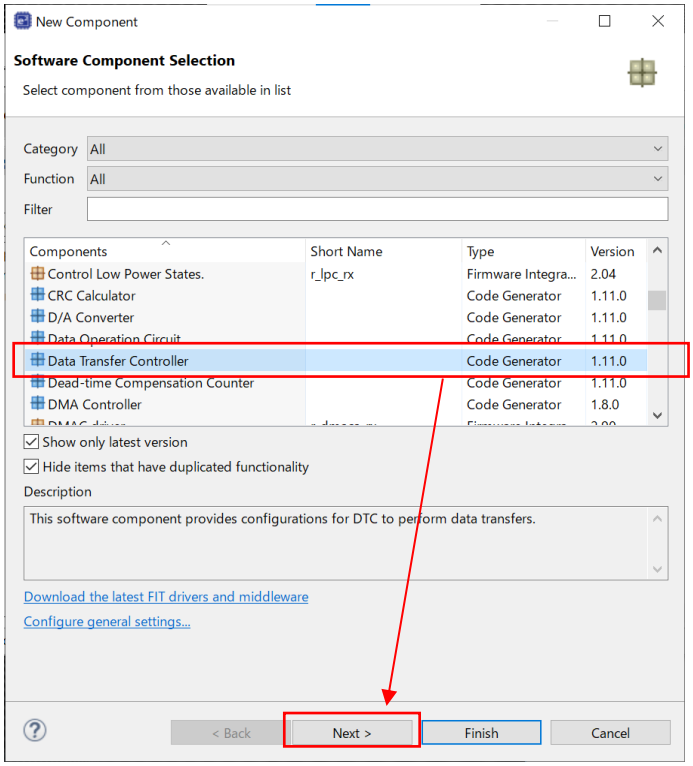
1. Perform the same procedure as steps 1 through 3 in section 3.1.2, Code Generation Procedure Example.

2. Configure software component settings for SCI12, and then click on the [Generate Code] icon.
   The required settings for this sample are listed below.

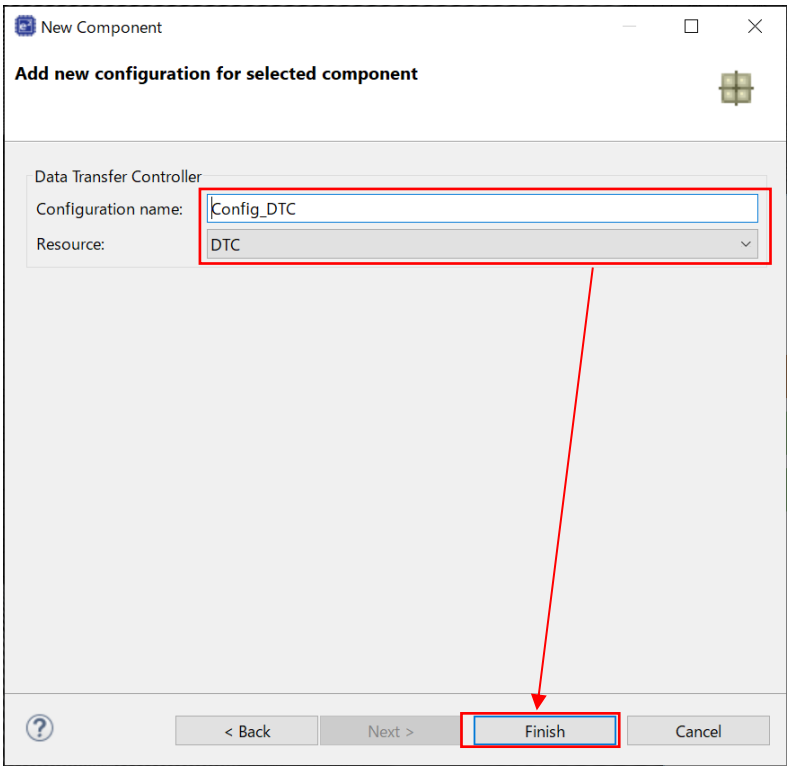
   Transmit data handling: Data handled by DTC
   TXI12 priority: Level 15 (highest)
   Callback function setting: "Transmission end" selected



3. Go to the [Software Component Selection] window again. Select [Data Transfer Controller], and then click on [Next].

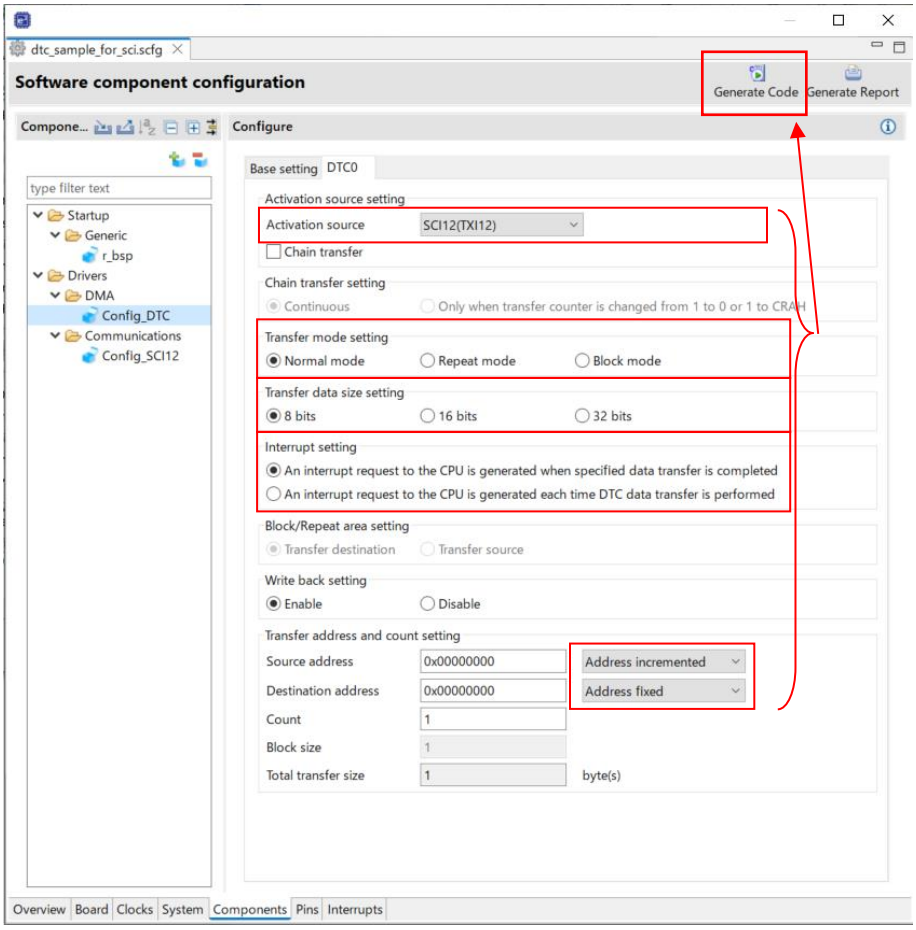


4. Specify an appropriate configuration name and resource, and then click on [Finish].
In this sample, the configuration name is "Config_DTC" and the resource is "DTC".



5. Configure software component settings for the DTC, and then click on the [Generate Code] icon.
   The required settings for this sample are listed below.

   Activation source: SCI12 (TXI12)
   Transfer mode: Normal mode
   Transfer data size: 8 bits
   Interrupt setting: An interrupt request to the CPU is generated when specified data transfer is completed
   Source address: Address incremented
   Destination address: Address fixed



Note: The transfer count, source address, and destination address are to be changed by software, so here they are left at their default values.

3.3.3 Modifying the Program

Table 3.2 lists the functions to be modified.

Table 3.3 Functions to Be Modified

Function Name
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
R_Config_DTC_Create
R_Systeminit

R_Config_SCI12_Start ()

Before modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    IEN(SCI12, TXI12) = 1U;
    ICU.GENBL0.BIT.EN16 = 1U;
}
```

To be deleted because IEN is set to 1 at the start of transmission by the R_Config_SCI12_Serial_Send function

After modification:

```
void R_Config_SCI12_Start(void)
{
    /* Clear interrupt flag */
    IR(SCI12, TXI12) = 0U;

    /* Enable SCI interrupt */
    ICU.GENBL0.BIT.EN16 = 1U;
    SCI12.SCR.BYTE |= 0xA0U;

   /* Set TXD12 pin */
    PORTA.PMR.BYTE |= 0x10U;
}
```

To prevent a 0-to-1 TE state change from working as a trigger, set TE = 1 by this function to inhibit control by the R_Config_SCI12_Serial_Send function.

Although TE = 1 sets IR of TXI12 to 1, no DTC transfer request is generated because IEN = 0. The request is held pending.

Also, set the TXD12 pin when setting TE = 1.

R_Config_SCI12_Serial_Send ()

Before modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    SCI12.SCR.BYTE |= 0xA0U;
    /* Set TXD12 pin */
    PORTA.PMR.BYTE |= 0x10U;

    return MD_OK;
}
```

To be deleted because SCI12.SCR.BYTE and the TXD12 pin are set by the R_Config_SCI12_Start function

After modification:

```
MD_STATUS R_Config_SCI12_Serial_Send(uint8_t * const tx_buf, uint16_t tx_num)
{
    IEN(SCI12, TXI12) = 1U;

    return MD_OK;
}
```

Set IEN to 1 to enable TXI interrupt that is the activation source for the DTC.

R_Config_DTC_Create ()

Before modification:

```
void R_Config_DTC_Create(void)
{
    /* Cancel DTC module stop state */
    MSTP(DTC) = 0U;

    /* Disable transfer data read skip to clear the flag */
    DTC.DTCCR.BYTE = _08_DTC_TRANSFER_READSKIP_DISABLE;

    /* Set DTC0 transfer data */
    dtc_transferdata_vector117.mra_mrb = ((uint32_t)(_00_DTC_WRITE_BACK_ENABLE
|
                                          _08_DTC_SRC_ADDRESS_INCREMENTED |
                                          _00_DTC_TRANSFER_SIZE_8BIT |
                                          _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |

((uint32_t)(_00_DTC_DST_ADDRESS_FIXED |
                                          _00_DTC_INTERRUPT_COMPLETED)<<16U);
    dtc_transferdata_vector117.sar      = _00000000_DTC0_SRC_ADDRESS;
    dtc_transferdata_vector117.dar      = _00000000_DTC0_DST_ADDRESS;
    dtc_transferdata_vector117.cra_crb = (uint32_t)
                                         (_0001_DTC0_TRANSFER_COUNT_CRA) << 16U;

    /* Set transfer data start address in DTC vector table */
    dtc_vector117 = (uint32_t) &dtc_transferdata_vector117;

    /* Set address mode */
    DTC.DTCADMOD.BYTE = _00_DTC_ADDRESS_MODE_FULL;

    /* Set base address */
    DTC.DTCVBR = (void *)0x0001FC00UL;

    /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;

    R_Config_DTC_Create_UserInit();
}
```

To be changed so that the transfer source address and the transfer count can be set as arguments.

The transfer source address and the transfer count are to be set from arguments.

The transfer destination address is to be changed to SCI12.TDR.

After modification:

```
void R_Config_DTC_Create(uint32_t sar, uint16_t count)
{
    /* Cancel DTC module stop state */
    MSTP(DTC) = 0U;

    /* Disable transfer data read skip to clear the flag */
    DTC.DTCCR.BYTE = _08_DTC_TRANSFER_READSKIP_DISABLE;

    /* Set DTC0 transfer data */
    dtc_transferdata_vector117.mra_mrb = ((uint32_t)(_00_DTC_WRITE_BACK_ENABLE
|
                                          _08_DTC_SRC_ADDRESS_INCREMENTED |
                                          _00_DTC_TRANSFER_SIZE_8BIT |
                                          _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |

((uint32_t)(_00_DTC_DST_ADDRESS_FIXED |
                                         _00_DTC_INTERRUPT_COMPLETED)<<16U);
    dtc_transferdata_vector117.sar      = sar;
    dtc_transferdata_vector117.dar      = (uint32_t)(&(SCI12.TDR));
    dtc_transferdata_vector117.cra_crb = (uint32_t)count << 16U;

    /* Set transfer data start address in DTC vector table */
    dtc_vector117 = (uint32_t) &dtc_transferdata_vector117;

    /* Set address mode */
    DTC.DTCADMOD.BYTE = _00_DTC_ADDRESS_MODE_FULL;

    /* Set base address */
    DTC.DTCVBR = (void *)0x0001FC00UL;

    /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;

    R_Config_DTC_Create_UserInit();
}
```

Changed so that the transfer source address and the transfer count can be set as arguments.

The transfer source address and the transfer count are set from arguments.

The transfer destination address is changed to SCI12.TDR.

Modifying the prototype declaration in the Config_DTC.h file:

```
/***********************************************************************************
Global functions
***********************************************************************************/
void R_Config_DTC_Create(uint32_t sar, uint16_t count);
```

R_Systeminit()

Before modification:

```
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and
software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE3.POECR2.WORD = 0x0000U;

    /* Initialize clocks settings */
    R_CGC_Create();

    /* Set peripheral settings */
    R_Config_SCI12_Create();
    R_Config_DTC_Create();

    /* Set interrupt settings */
    R_Interrupt_Create();

    /* Register undefined interrupt */

    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT,
     (bsp_int_cb_t)r_undefined_exception);

    /* Register group BL0 interrupt TEI12 (SCI12) */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_TEI12,
     (bsp_int_cb_t)r_Config_SCI12_transmitend_interrupt);

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

The initial settings are to be deleted because DTC settings will be changed to such a specification that the transmit buffer and the transfer count are set before transmission from the SCI.

After modification:

```
void R_Systeminit(void)
{
    /* Enable writing to registers related to operating modes, LPC, CGC and
software reset */
    SYSTEM.PRCR.WORD = 0xA50BU;

    /* Enable writing to MPC pin function control registers */
    MPC.PWPR.BIT.B0WI = 0U;
    MPC.PWPR.BIT.PFSWE = 1U;

    /* Write 0 to the target bits in the POECR2 registers */
    POE3.POECR2.WORD = 0x0000U;

    /* Initialize clocks settings */
    R_CGC_Create();

    /* Set peripheral settings */
    R_Config_SCI12_Create();

    /* Set interrupt settings */
    R_Interrupt_Create();

    /* Register undefined interrupt */

    R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT,
     (bsp_int_cb_t)r_undefined_exception);

    /* Register group BL0 interrupt TEI12 (SCI12) */
    R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI12_TEI12,
     (bsp_int_cb_t)r_Config_SCI12_transmitend_interrupt);

    /* Disable writing to MPC pin function control registers */
    MPC.PWPR.BIT.PFSWE = 0U;
    MPC.PWPR.BIT.B0WI = 1U;

    /* Enable protection */
    SYSTEM.PRCR.WORD = 0xA500U;
}
```

3.3.4 Sample Program

The following shows how the sample program works.

[Main function]

(1) Call the R_Config_SCI12_Start function to configure transmission.
(2) Call the R_Config_DTC_Create function to set transmit data for the DTC transfer source, the SCI transmit buffer for the transfer destination, and the transfer count.
(3) Set g_flag = 0, and set g_flag = 1 only after transmission is complete to prevent further processing until transmission is complete.
(4) Execute the R_Config_DTC_Start function to put the DTC in a state waiting for transfer.
(5) Call the R_Config_SCI12_Serial_Send function to enable the TXI interrupt that works as an activation source for the DTC. DTC transfer occurs at the timing of the TXI interrupt, and transmit data is written to the transmit buffer.
(6) Wait until transmission is completed.
(7) Insert 300-µs wait time.
(8) Repeat steps (2) to (7) twice.
(9) After transmission is completed, call the R_Config_DTC_Stop function and the R_Config_SCI12_Stop function to disable the transmission setting.

Note: Since the R_Config_SCI12_Create function is called in the R_Systeminit function before reaching the main function, it does not need to be called in the main function.

```
/***********************************************
Global variables
***********************************************/
volatile uint8_t g_flag;

/***********************************************
Private (static) variables and functions
***********************************************/
uint8_t          g_data[] = {0x11, 0x22, 0x33};

void main(void);

/***********************************************
* Function Name: main
* Description  : This function uses the modified program to send data without a wait time.
* Arguments : None
* Return Value : None
***********************************************/
void main(void)
{
    uint8_t i           = 0;
    uint8_t send_count = 2;

    R_Config_SCI12_Start();

    for (i = 0; i < send_count; i++)
    {
        g_flag = 0;
        R_Config_DTC_Create((uint32_t)g_data, 3 );
        R_Config_DTC_Start();
        R_Config_SCI12_Serial_Send(NULL, 0);

        while (0 == g_flag)
        {
            nop();
        }
        R_BSP_SoftwareDelay((uint32_t)300, BSP_DELAY_MICROSECS);
    }

    R_Config_DTC_Stop();
   R_Config_SCI12_Stop();

    while (1)
    {
        nop();
    }
}
```

Figure 3.13 "main" Function of the Sample Program

[SCI12 transmission end callback function]

(1) The SCI12 transmit end callback function (r_Config_SCI12_callback_transmitend) is called upon completion of DTC transfer and upon completion of SCI transfer.
(2) When the transmission end interrupt is disabled (TEIE = 0), the function judges that it is called because DTC transfer is completed, so it disables the TXI interrupt request and enables the transmission end interrupt (TEIE = 1).
(3) When the transmission end interrupt is enabled (TEIE = 1), the function judges that it is called because transmission is completed, so it disables the transmission end interrupt and sets the transmission end flag (g_flag = 1).

```
/******************************************
* Function Name: r_Config_SCI12_callback_transmitend
* Description  : This function is a callback function when SCI12 finishes
transmission
* Arguments    : None
* Return Value : None
******************************************/
static void r_Config_SCI12_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI12_callback_transmitend. Do not edit
comment generated here */
  if(0 == SCI12.SCR.BIT.TEIE)
  {
   /* Interrupt processing when DTC transfer is completed */
   IEN(SCI12, TXI12) = 0U;
    SCI12.SCR.BIT.TEIE = 1U;
  }
  else
  {
   /* Interrupt processing when SCI12 transmit end */
   SCI12.SCR.BIT.TEIE = 0U;
   g_flag = 1;
  }
  /* End user code. Do not edit comment generated here */
}
```

Figure 3.14 "r_Config_SCI12_callback_transmitend" Function in the Sample Program

3.3.5 Operation of the Sample Program

Figure 3.15 shows the waveforms when the control example using the DTC is operated.

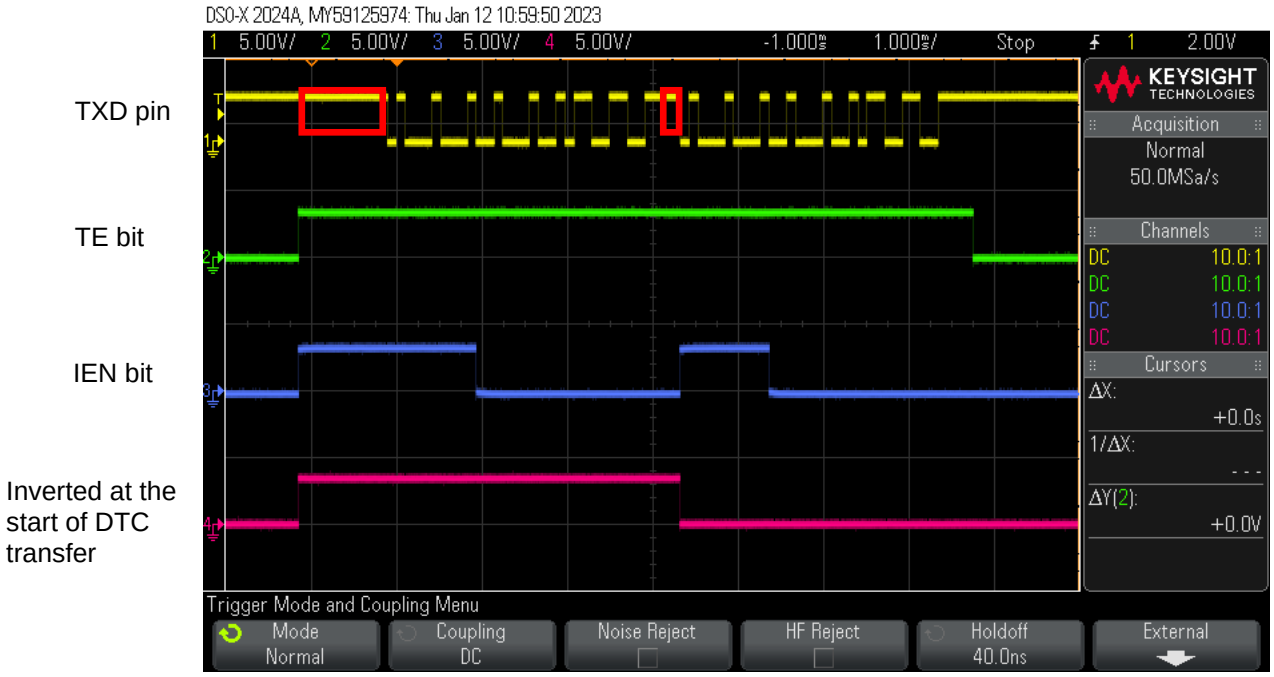


Figure 3.15 Waveforms of the Sample Program

It can be confirmed that during the first transmission, an internal wait time occurs when TE changes from 0 to 1, but during the second transmission, the TE bit is not controlled, so the transmission starts without generating an internal wait time. For the first transmission, no internal wait time occurs either if the R_Config_SCI12_Serial_Send function is executed at least one frame after the R_Config_SCI12_Start function is called.

4. Disabling Automatic Code Regeneration at Build Time

Smart Configurator is provided with a function that automatically regenerates code at build time.

In this sample program, this function is disabled because code-generated components are modified for use.

Note that if you regenerate the code manually, the components will be overwritten.

If you generate the code manually, salvage the components from the trash folder or make a backup beforehand.

Figure 4.1 shows how to disable code regeneration when building with e² studio.

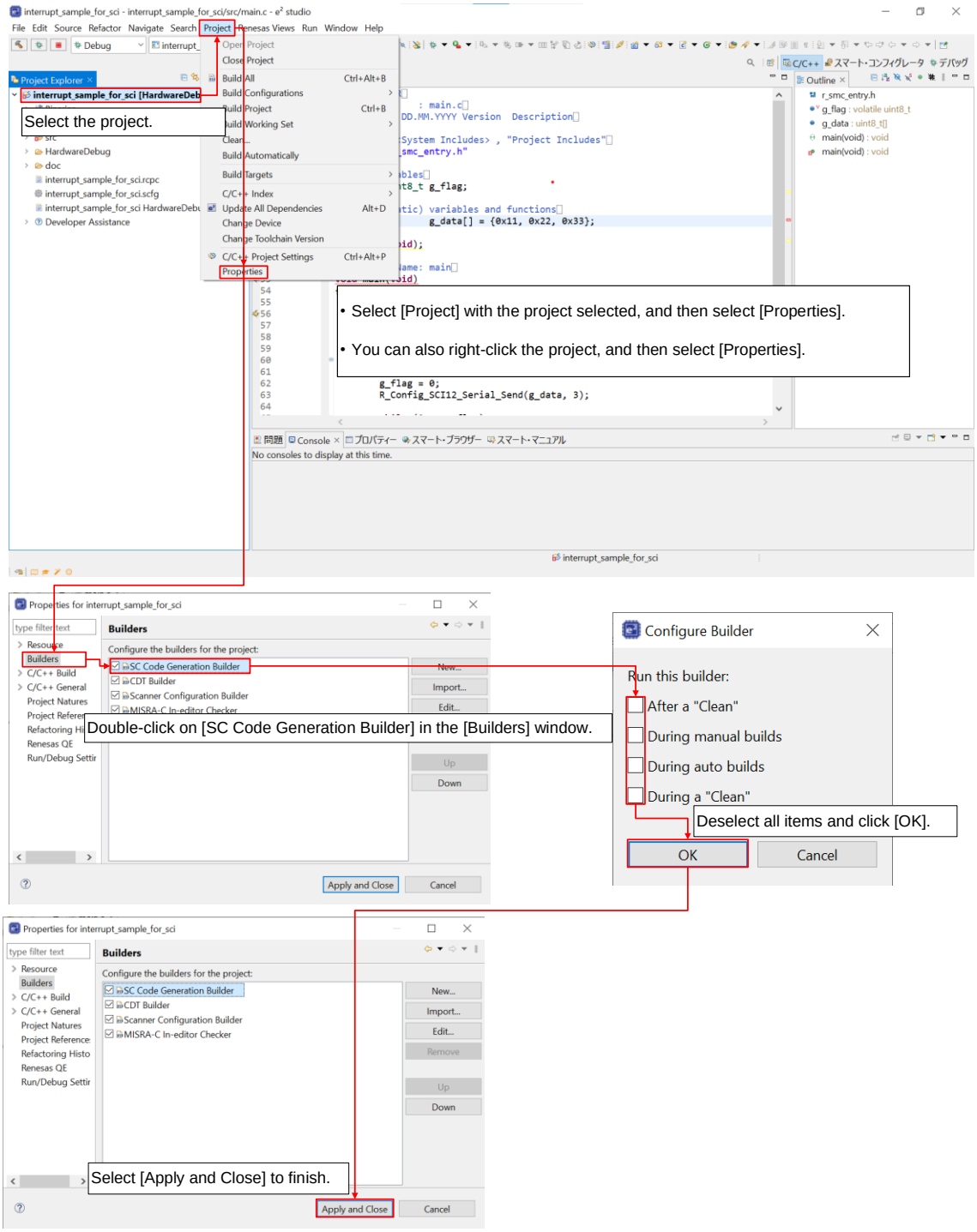


Figure 4.1 Disabling Automatic Code Regeneration at Build Time

5. Importing a Project

The sample programs are distributed in e² studio project format. This section shows how to import a project into e² studio or CS+. After importing the sample project, make sure to confirm build and debugger setting.

5.1 Importing a Project into e² studio

To use sample programs in e² studio, follow the steps below to import them into e² studio.

In projects managed by e² studio, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

(Note that depending on the version of e² studio you are using, the interface may appear somewhat different from the screenshots below.)

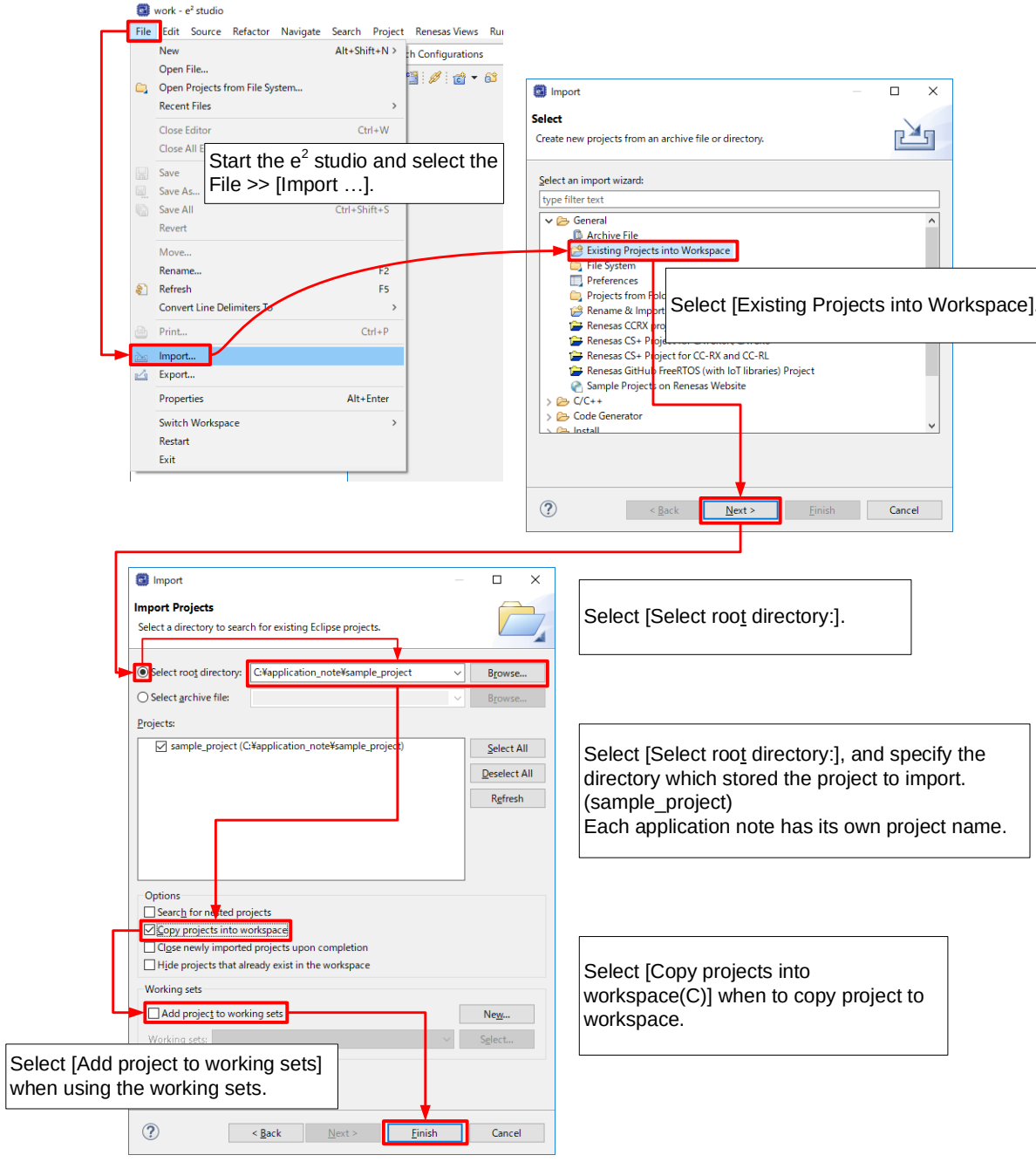


Figure 5.1 Importing a Project into e² studio

5.2 Importing a Project into CS+

To use sample programs in CS+, follow the steps below to import them into CS+.

In projects managed by CS+, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

(Note that depending on the version of CS+ you are using, the interface may appear somewhat different from the screenshots below.)

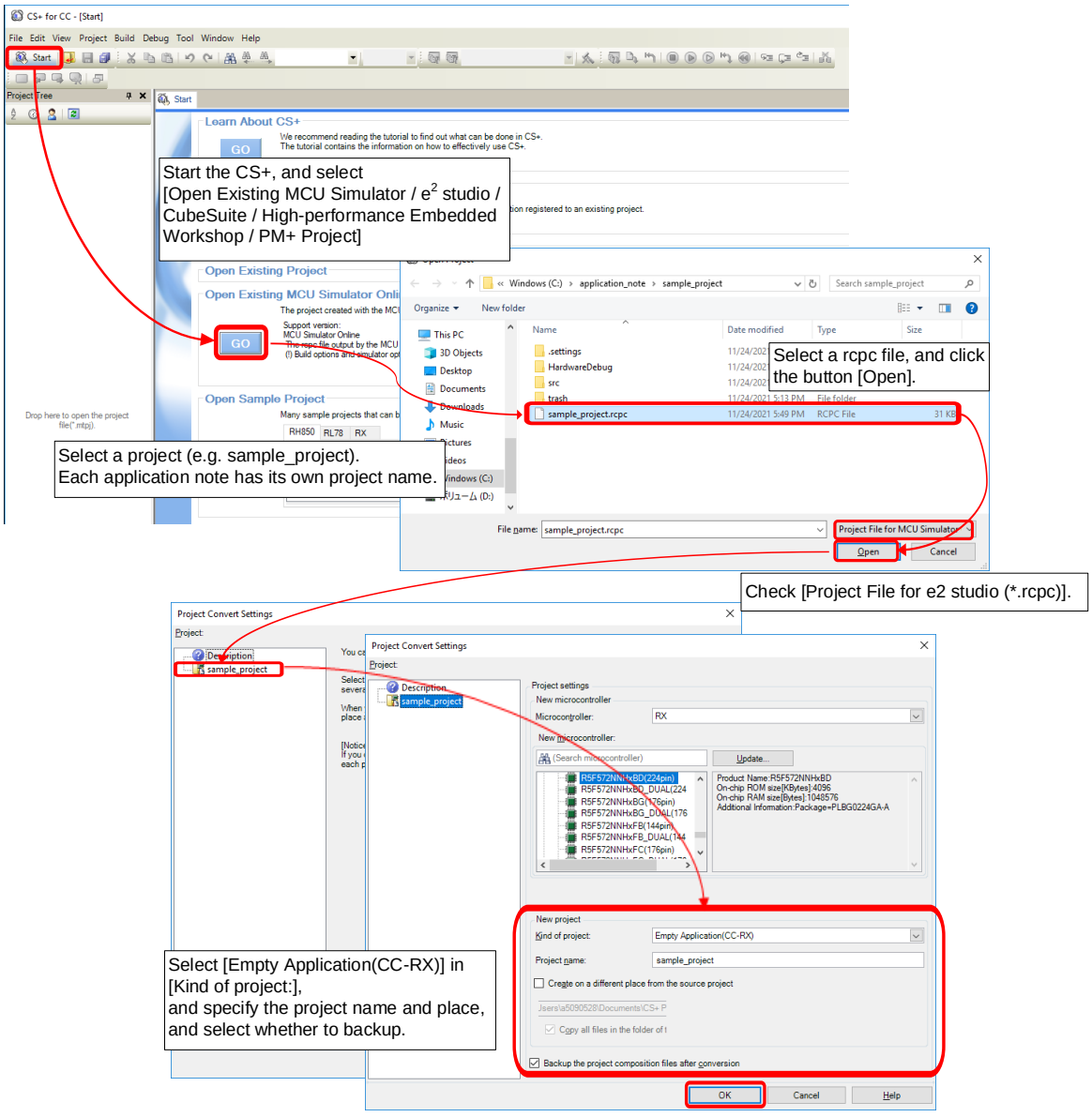


Figure 5.2 Importing a Project into CS+

6. Reference Documents

- RX660 Group User's Manual: Hardware (R01UH0937)
- Renesas e^2 studio Smart Configurator User's Guide (R20AN0451)
- Renesas Starter Kit for RX660 User's Manual (R20UT5017)
- Renesas Starter Kit for RX660 CPU Board Circuit Diagram (R20UT5016)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.20.23	-	First edition issued
1.01	Mar.21.24	1	Added Target Tools

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

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

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

© 2024 Renesas Electronics Corporation. All rights reserved.