RXファミリ

SCIの調歩同期式モードで送信データ書き込み後すぐに送信開始する方法

要旨

本アプリケーションノートは、シリアルコミュニケーションインターフェース(以下、SCI)の調歩同期式モードで送信データ書き込み後すぐに送信開始する方法について説明します。

対象デバイス

RX ファミリ

対象ツール

- スマート・コンフィグレータ V2.19.0 以前
- e2 studio 2023-10 以前

e2 studio 2024-01 以後に実装されているスマート・コンフィグレータおよびスマート・コンフィグレータ V2.20.0 以降のバージョンではスマート・コンフィグレータに同等の機能が実装されていますので、そちらをご使用ください。

動作確認デバイス

RX660 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. SCI の待機期間について

1.1 待機期間の概要

調歩同期式モードでは SCR.TE ビットを“1”にした後、送信データを TDR レジスタにライトするとデータ送信が開始されます。

しかし SCR.TE ビットを“1”にしてからデータ送信開始までに 1 フレーム分の内部待機期間が発生し、内部待機期間は SCR.TE ビットを“1”にするたびに発生します。

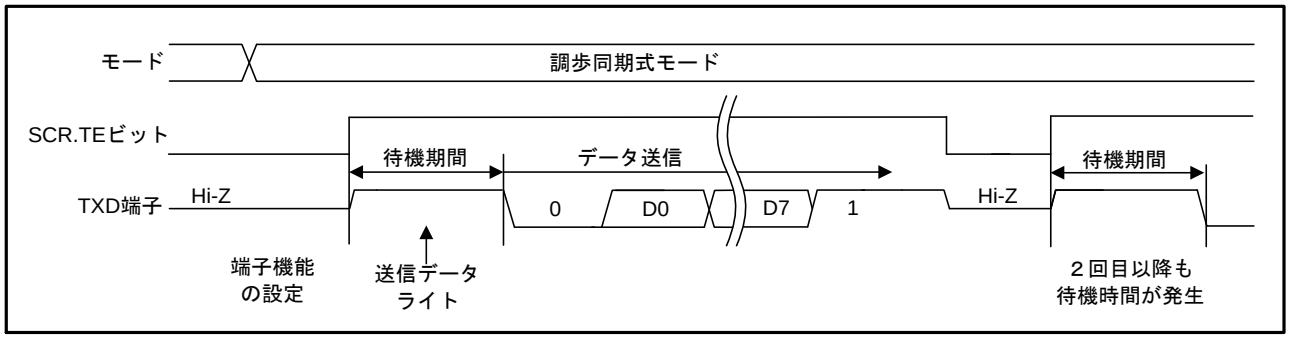


図 1.1　データ送信タイミングでの内部待機期間

1.2 即時送信機能

SEMR.ITE ビットをサポートする一部製品では、本ビットを制御することで内部待機期間を発生させずに SCI を使用できます。

1.3 スマート・コンフィグレータにおけるコード生成機能の概要

コード生成機能で生成した SCI のコンポーネントは、ユーザーズマニュアルに記載している調歩同期式モードのシリアル送信のフローチャート例と同等の処理内容で生成されます。

シリアル送信のフローチャート例では TE ビットで送信処理を制御しているため、SCI のコンポーネントを使用すると内部待機期間が発生します。

本アプリケーションノートでは、コード生成機能で生成された SCI コンポーネントをサンプルプログラムとして使用するため、内部待機期間を発生させないような処理に修正しています。修正方法は 3 章および 4 章で説明しています。

2. 動作確認条件

表 2.1　動作確認条件

項目	内容
使用マイコン	R5F56609HDFB (RX660N グループ)
動作周波数	• メインクロック: 24MHz • PLL: 240MHz (メインクロック 1 分周 10 逓倍) • システムクロック(ICLK): 120MHz (PLL 2 分周) • 周辺モジュールクロック B (PCLKB): 60MHz (PLL 4 分周)
動作電圧	3.3V
統合開発環境	ルネサスエレクトロニクス製 e² studio Version 2023-01 (23.1.0)
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V.3.05.00
	コンパイルオプション -lang = c99
iodefine.h のバージョン	Version 1.00
エンディアン	リトルエンディアン
動作モード	シングルチップモード
プロセッサモード	スーパバイザモード
サンプルプログラムのバージョン	Version 1.00
使用ボード	Renesas Starter Kit for RX660 (型名：RTK556609xxxxxxxxx)

3.　ソフトウェア説明

本章では、送信データ書き込み後すぐに送信開始する方法を記載しています。本サンプルプログラムには即時送信機能の無い SCI のコンポーネント(SCIh モジュール(SCI12))を使用しています。

3.1　送信データエンプティ割り込みを使用した制御例

3.1.1　概要

内部待機期間は1.1項の理由から TE=0→1 をトリガに発生しています。本章では TE=1 の状態を維持しながら、IEN=0→1 をトリガに割り込みを制御し、割り込み処理内で送信データの書き込みを行い、2 回目以降の送信タイミングで内部待機期間が発生しないように修正しています。

図 3.1に修正前のタイミング図を、図 3.2に修正後のタイミング図を示します。

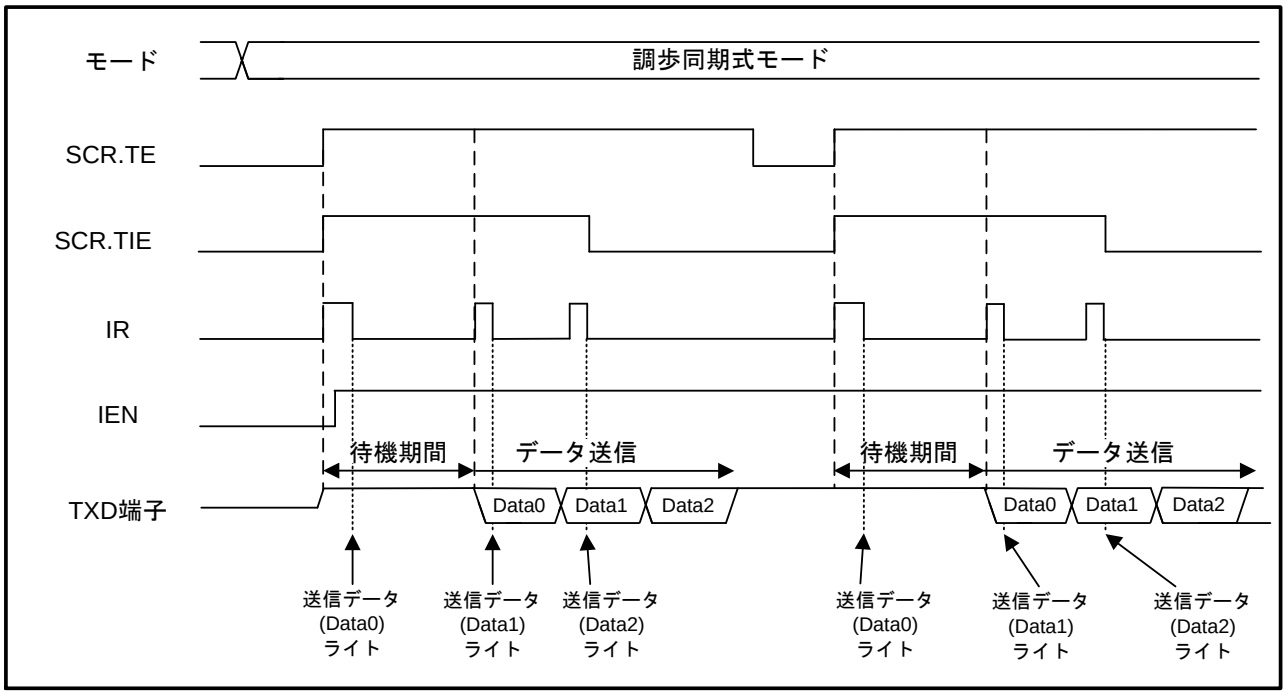


図 3.1　修正前のタイミング図

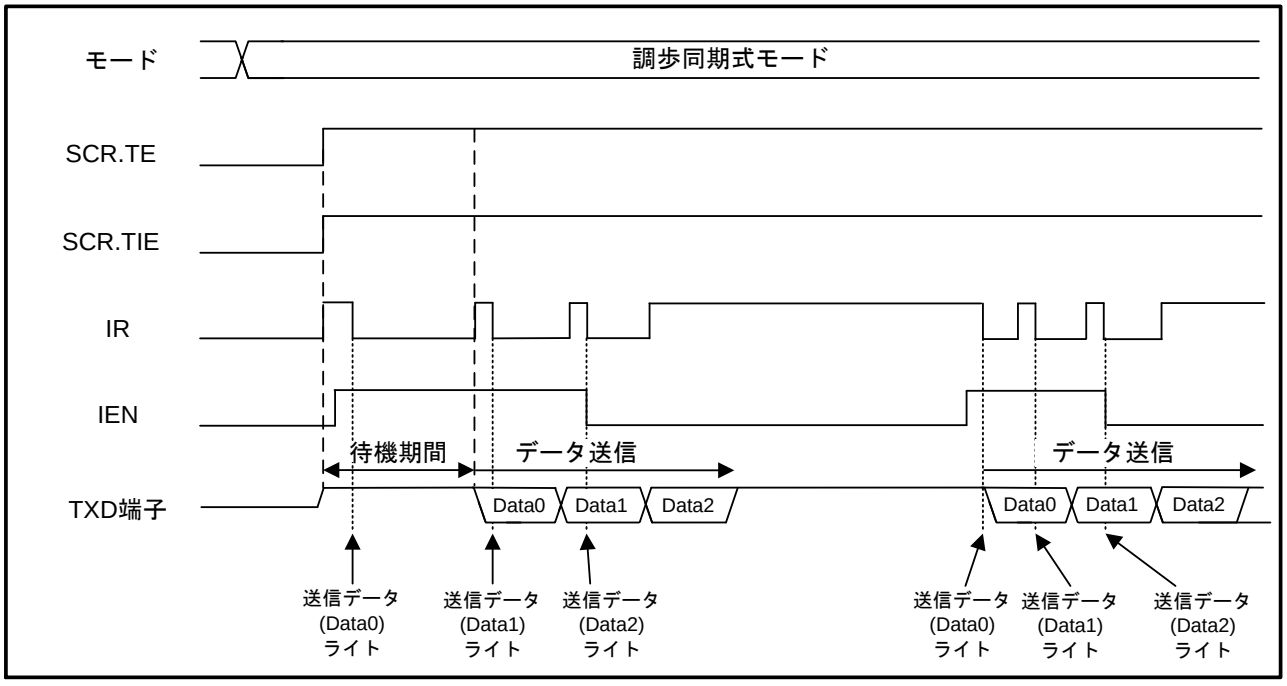
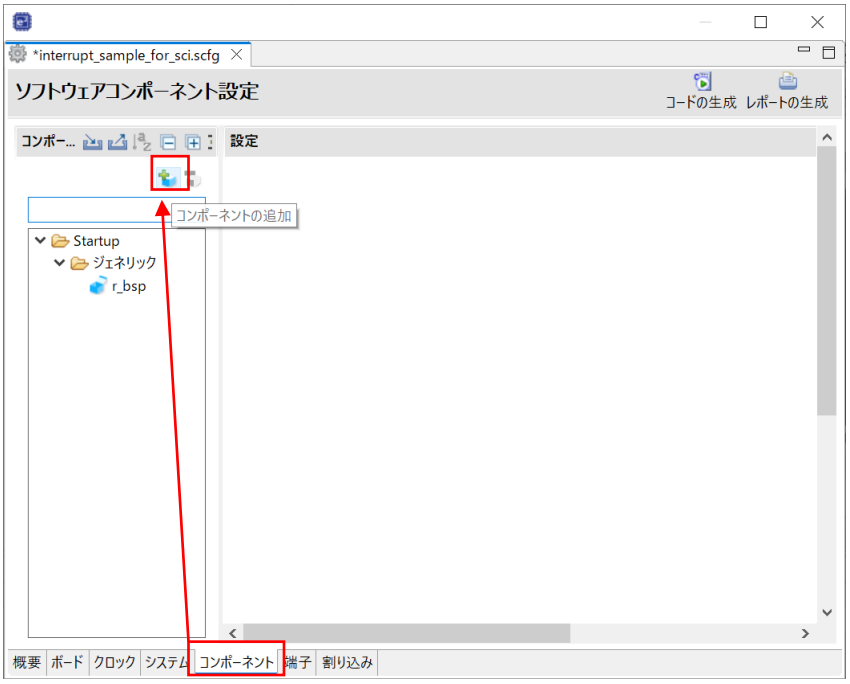


図 3.2　修正後のタイミング図

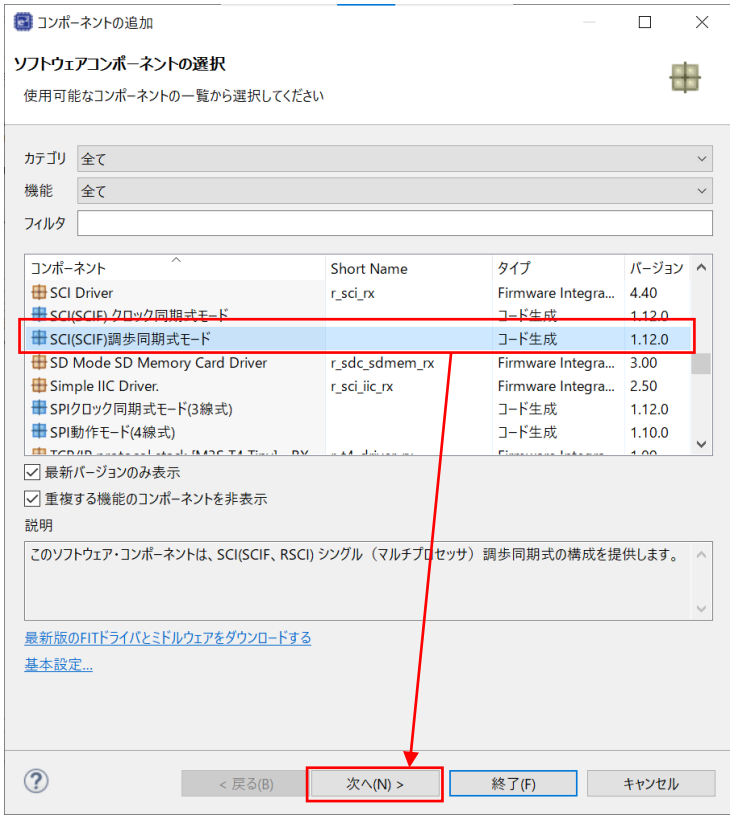
3.1.2　コード生成手順例

スマート・コンフィグレータでのコード生成手順例を示します。

1.　［コンポーネント］タブを開き、コンポーネントの追加を選択する。



2.　ソフトウェアコンポーネントの選択画面で「SCI(SCIF)調歩同期式モード」を選択し、次へを実行する。

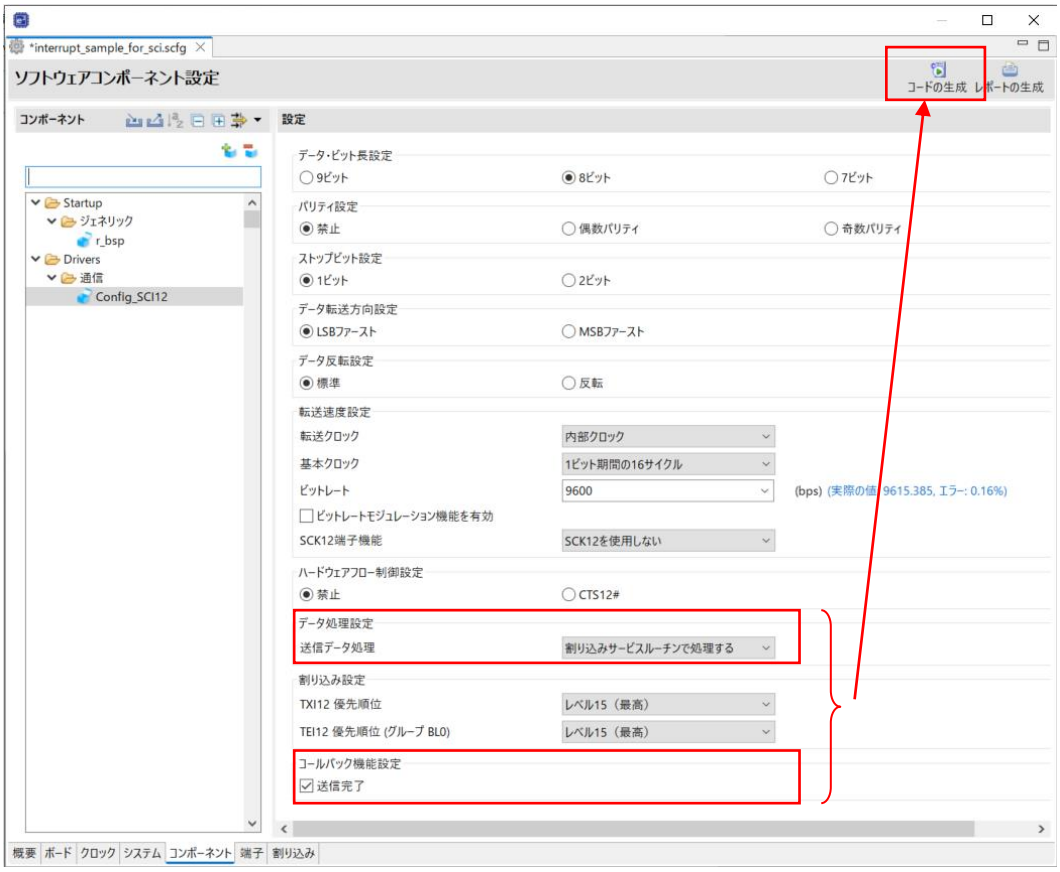


3. コンフィグレーション名、作業モード、リソースを選択して終了を実行する。
本サンプルでは、コンフィグレーション名を「Config_SCI12」、作業モードを「送信」、リソースを「SCI12」としています。

4. SCI12 のソフトウェアコンポーネント設定を行い、コード生成を実行する。
本サンプルで必須となる設定を以下に示します。

送信データ処理：割り込みサービスルーチンで処理する

コールバック機能設定：送信完了あり



5. コード生成のメッセージボックスが表示されるので、続行を実行する。

3.1.3　プログラムの修正

修正する関数を表 3.1に示します。

表 3.1　修正する関数

関数名
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
r_Config_SCI12_transmit_interrupt
r_Config_SCI12_transmitend_interrupt

R_Config_SCI12_Start ()

修正前：

```
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;
}
```

IEN=1 は R_Config_SCI12_Serial_Send 関数で送信開始時に行うため削除。

修正後：

```
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;

}
```

TE=0→1 をトリガにしないため本関数で TE=1 にして R_Config_SCI12_Serial_Send 関数で制御しないようにする。

TE=1 にすることで TXI12 の IR=1 になるが、IEN=0 のため CPU への割り込みが発生せず保留される。

端子設定も TE=1 にするタイミングで設定を行う。

R_Config_SCI12_Serial_Send ()

修正前：

```
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);
}
```

R_Config_SCI12_Start 関数で IEN の設定をしていないため削除。

IEN が”0”の状態で本関数が実行されるため削除。
SCI12.SCR.BYTE 及び端子設定は
R_Config_SCI12_Start 関数で設定しているため削除。

修正後：

```
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);
}
```

IEN=1 にすることで、保留されていた
TXI12 割り込みが発生する

r_Config_SCI12_transmit_interrupt ()

修正前：

```
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;
    }
}
```

送信完了後に IEN=1→0 に設定する必要があるため if (1U < g_sci12_tx_count)に変更して最後のデータ送信完了後に IEN=0→1 を設定することにより割り込みを制御。

最後のデータを書き込み後 else if(1U == g_sci12_tx_count)で IEN=1→0 に設定。

最終データ送信後、割り込み禁止にする必要があるため条件式の変更が必要。最終データ送信後の処理が変わるので else は不要。

修正後：

```
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 */
    }
}
```

最終データ以外は、修正前同様送信。

最終データ送信後、割り込みを禁止。

基本的に通ることがない処理になるが例外処理として追加。

r_Config_SCI12_transmitend_interrupt ()

修正前：

```
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();
}
```

送信設定（TE=1）を保持するため、削除。

修正後：

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

    r_Config_SCI12_callback_transmitend();
}
```

3.1.4　サンプルプログラム

サンプルプログラムの動作を示します。

[メイン関数]
(1) 送信設定するために R_Config_SCI12_Start 関数を呼び出す。
(2) g_flag=0 に設定。
(3) R_Config_SCI12_Serial_Send 関数を呼び出し{0x11, 0x22, 0x33}の 3 バイトデータを送信。
(4) 送信完了まで待機。（送信完了割り込みで g_flag=1 に設定されるまで待つ。）
(5) 300us の待ち時間を挿入。
(6) (2)～(5)を 2 回繰り返す。
(7) 送信終了後、送信設定を解除のため R_Config_SCI12_Stop 関数を呼び出す。

【注】R_Config_SCI12_Create 関数は main 関数に到達する前の R_Systeminit 関数で呼び出されているため、main 関数での呼び出しは不要です。

```
/***********************************************
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();
    }
}
```

図 3.3　サンプルプログラム main 関数

[SCI12 送信完了コールバック関数]
(1) 送信完了フラグの設定(g_flag=1)を行う

```
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 */
}
```

図 3.4　サンプルプログラム r_Config_SCI12_callback_transmitend 関数

3.1.5　修正後の動作

図 3.5、図 3.6は{0x11, 0x22, 0x33}の 3 バイトデータを送信した時の修正前、修正後の動作波形です。

修正前の動作：

図 3.5　修正前の波形

修正前では TE=0→1 のタイミングで内部待機期間が発生していることが確認できます。

修正後の動作：

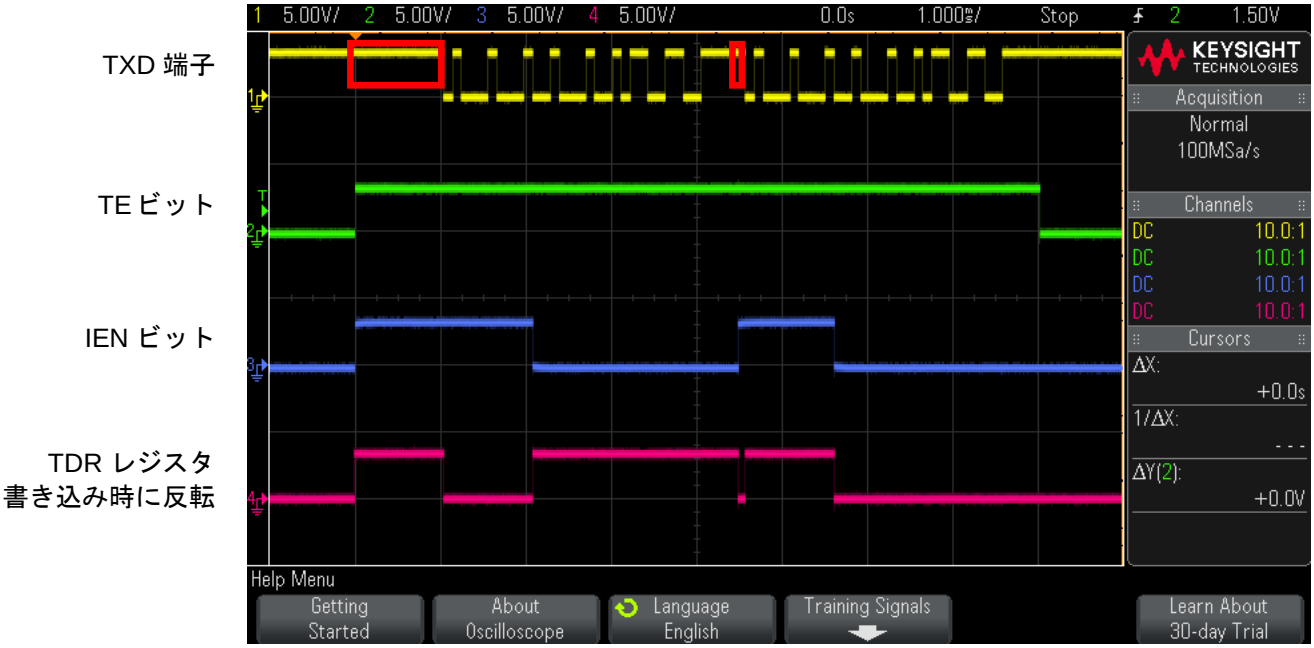


図 3.6　修正後の波形

1 回目の送信時、TE=0→1 のタイミングで内部待機期間が発生するが、2 回目の送信時は TE ビットを制御しないため、内部待機期間を発生させずに送信開始していることが確認できます。1 回目の送信に関しても、R_Config_SCI12_Start 関数呼び出し後、1 フレーム以上経過後に R_Config_SCI12_Serial_Send 関数を実行すると内部待機期間は発生しません。

3.2　DMAC を使用した制御例

3.2.1　概要

本章では、「3.1 送信データエンプティ割り込みを使用した制御例」と同様に、TE=1 の状態を維持し、DMAC 転送で送信データを書き込む場合について説明します。

図 3.7に修正後のタイミング図（DMAC転送にて送信データを書き込む場合）を示します。

（修正前のタイミングは、DMAC 動作を除き図 3.1と同等です。）

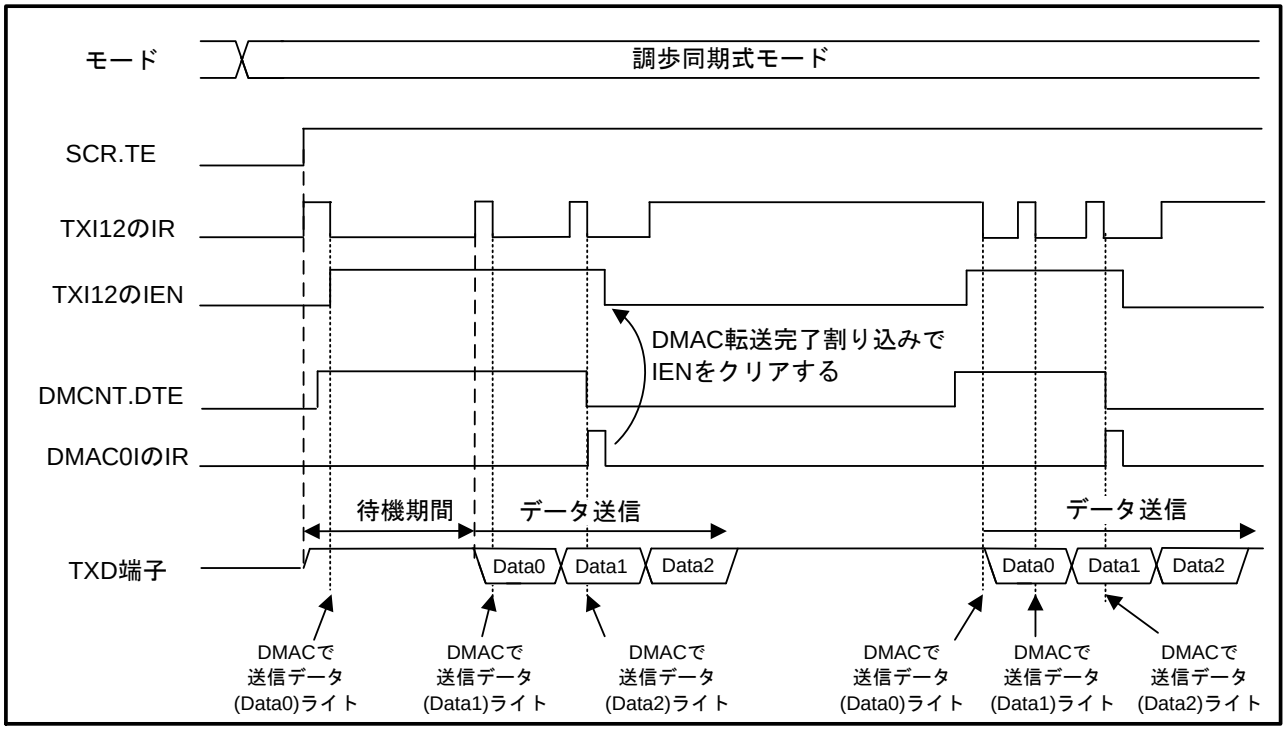


図 3.7　修正後のタイミング図（DMAC 転送にて送信データを書き込む場合）

3.2.2　コード生成手順例

「3.1 送信データエンプティ割り込みを使用した制御例」のコード生成手順例と同様にスマート・コンフィグレータを使用してコード生成を行います。

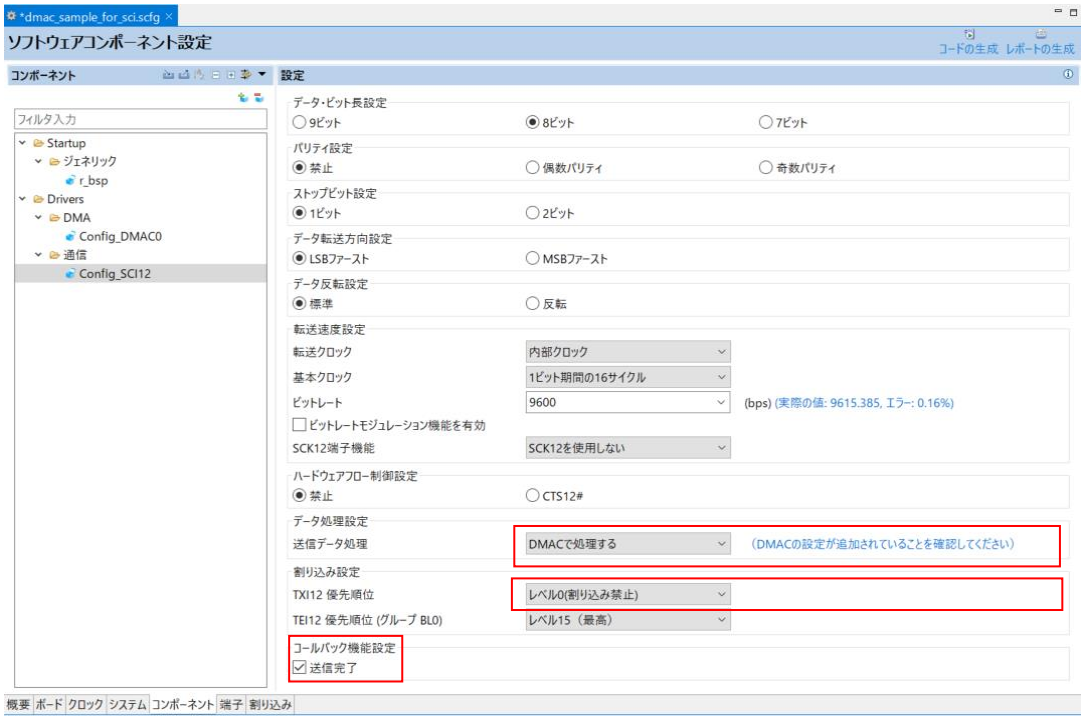
1. 「**3.1.2** コード生成手順例」の手順 1～3 と同様の手順を実行する。

2. SCI12 のソフトウェアコンポーネント設定を行い、コード生成を実行する。
   本サンプルで必須となる設定を以下に示します。

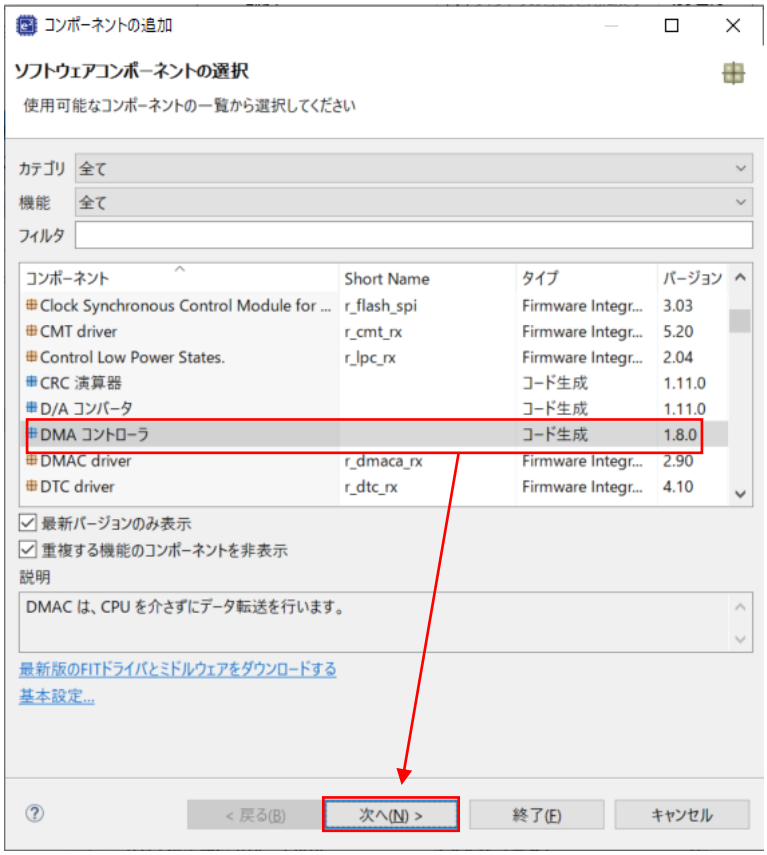
   送信データ処理：DMAC で処理する

   TXI12 優先順位：レベル 0 (割り込み禁止)

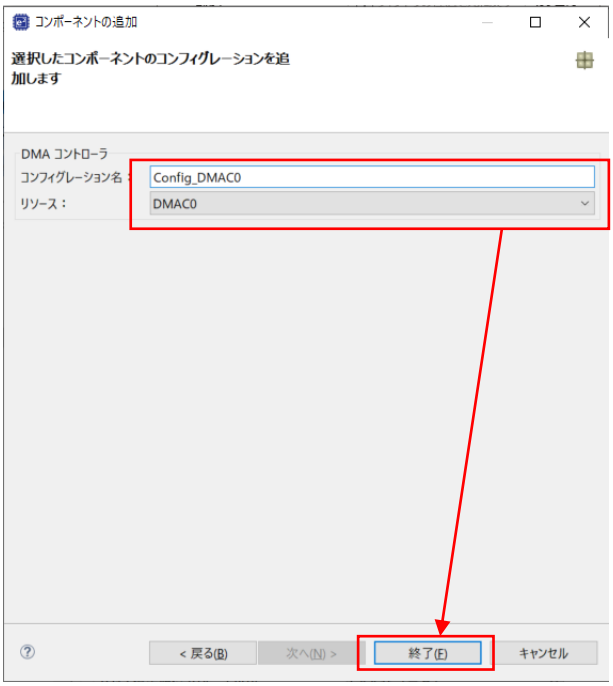
   コールバック機能設定：送信完了あり



3. 再度、コンポーネントの追加を行う。「DMA コントローラ」を選択し、次へを実行する。



4. コンフィグレーション名、リソースを選択して終了を実行する。
本サンプルでは、コンフィグレーション名を「Config_DMAC0」、リソースを「DMAC0」としています。



5. DMAC0 のソフトウェアコンポーネント設定を行い、コード生成を実行する。
本サンプルで必須となる設定を以下に示します。

起動要因：SCI12 (TXI12)

転送要因フラグ制御：転送要因フラグをクリアする

転送モード：ノーマルモード

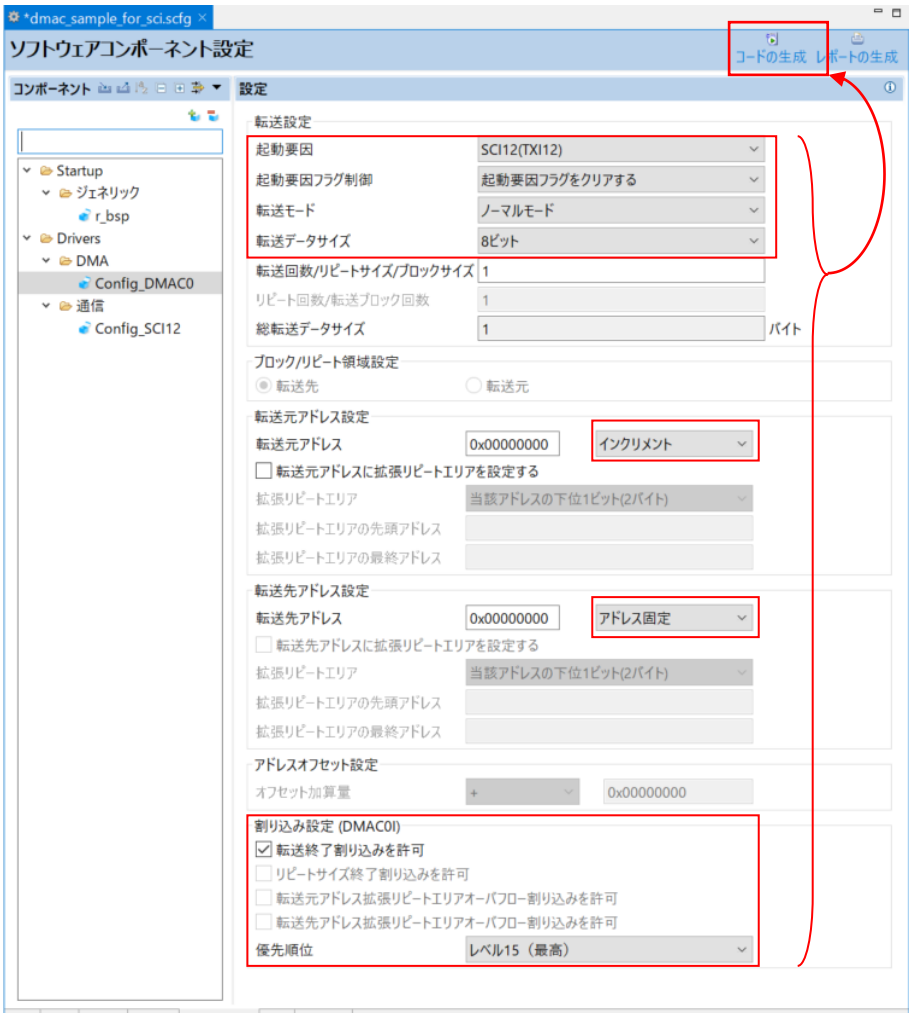
転送データサイズ：8 ビット

転送元アドレス：インクリメント

転送先アドレス：アドレス固定

転送完了割り込みを許可：チェック

優先順位：レベル 15 (最高)



【注】　転送回数、転送元アドレスの番地、転送先アドレスの番地はソフトウェアで設定するよう変更するため、ここではデフォルト値のままとします。

3.2.3 プログラムの修正

修正する関数を表 3.2に示します。

表 3.2　修正する関数

関数名
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
R_Config_DMAC0_Create
R_Systeminit

R_Config_SCI12_Start ()

修正前：

```
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;
}
```

IEN=1 は R_Config_SCI12_Serial_Send 関数で送信開始時に行うため削除。

修正後：

```
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;

}
```

TE=0→1 をトリガにしないため本関数で TE=1 にして R_Config_SCI12_Serial_Send 関数で制御しないようにする。

TE=1 にすることで TXI12 の IR=1 になるが、IEN=0 のため DMAC 転送要求が発生せず保留される。

端子設定も TE=1 にするタイミングで設定を行う。

R_Config_SCI12_Serial_Send ()

修正前：

```
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;
}
```

SCI12.SCR.BYTE 及び端子設定は
R_Config_SCI12_Start 関数で設定しているため削除。

修正後：

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

    return MD_OK;
}
```

IEN=1 に設定し DMAC に起動要因となる
TXI 割り込みを許可にする。

R_Config_DMAC0_Create ()

修正前：

```
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();
}
```

引数に転送元アドレス、
転送回数を設定できるように変更。

転送元アドレス、
転送回数を引数から設定。
転送先アドレスは
SCI12.TDR に変更。

修正後：

```
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();
}
```

引数に転送元アドレス、転送回数を設定できるように変更。

転送元アドレス、
転送回数を引数から設定。
転送先アドレスは
SCI12.TDR に変更。

Config_DMAC0.h ファイルのプロトタイプ宣言修正：

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

R_Systeminit()

修正前：

```
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;
}
```

DMAC の設定は、SCI 送信前に送信バッファ及び送信回数を設定する仕様に変更するため、初期設定は削除

修正後：

```
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　サンプルプログラム

サンプルプログラムの動作を示します。

[メイン関数]

(1) 送信設定するために R_Config_SCI12_Start 関数を呼び出す。
(2) R_Config_DMAC_Create 関数を呼び出し、DMAC の転送元に送信データ、転送先に SCI 送信バッファ、転送回数を設定。
(3) g_flag=0 に設定、送信完了後に g_flag=1 にすることで送信完了まで次の処理を行わない。
(4) R_Config_DMAC_Start 関数を実行し、DMAC の転送待ち状態にする。
(5) R_Config_SCI12_Serial_Send 関数を呼び出し、DMAC の転送要因となる TXI 割り込みを許可。TXI 割り込み発生タイミングで DMAC 転送が発生し、送信バッファに送信データが書き込まれる。
(6) 送信完了まで待機。
(7) 300us の待ち時間を挿入。
(8) (2)～(7)を 2 回繰り返す。
(9) 送信終了後、送信設定を解除のため R_Config_DMAC_Stop 関数および R_Config_SCI12_Stop 関数を呼び出す。

【注】R_Config_SCI12_Create 関数は main 関数に到達する前の R_Systeminit 関数で呼び出されているため、main 関数での呼び出しは不要です。

```
/***********************************************
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();
    }
}
```

図 3.8　サンプルプログラム main 関数

[DMAC0 転送完了コールバック関数]

(1) DMAC0 転送完了コールバック関数(r_dmac0_callback_transfer_end)は、すべての送信データが SCI の送信バッファへ転送が完了した時に呼び出される。
(2) TXI 割り込み要求の禁止、および送信完了割り込みの許可(TEIE=1)の設定を行う

```
/******************************************
* 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 */
}
```

図 3.9　サンプルプログラム r_dmac0_callback_transfer_end関数

[SCI12 送信完了コールバック関数]

(1) 送信完了割り込みの禁止、および送信完了フラグの設定(g_flag=1)を行う
※ r_Config_SCI12_callback_transmitend 関数は TXI12 割り込み時にも呼び出されるため、送信完了割り込みが許可状態の場合のみ処理する

```
/******************************************
* 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 */
}
```

図 3.10　サンプルプログラム r_dmac0_callback_transfer_end関数

3.2.5　サンプルプログラムの動作

DMAC を使用した制御例を動作させた場合の波形を図 3.11に示します。

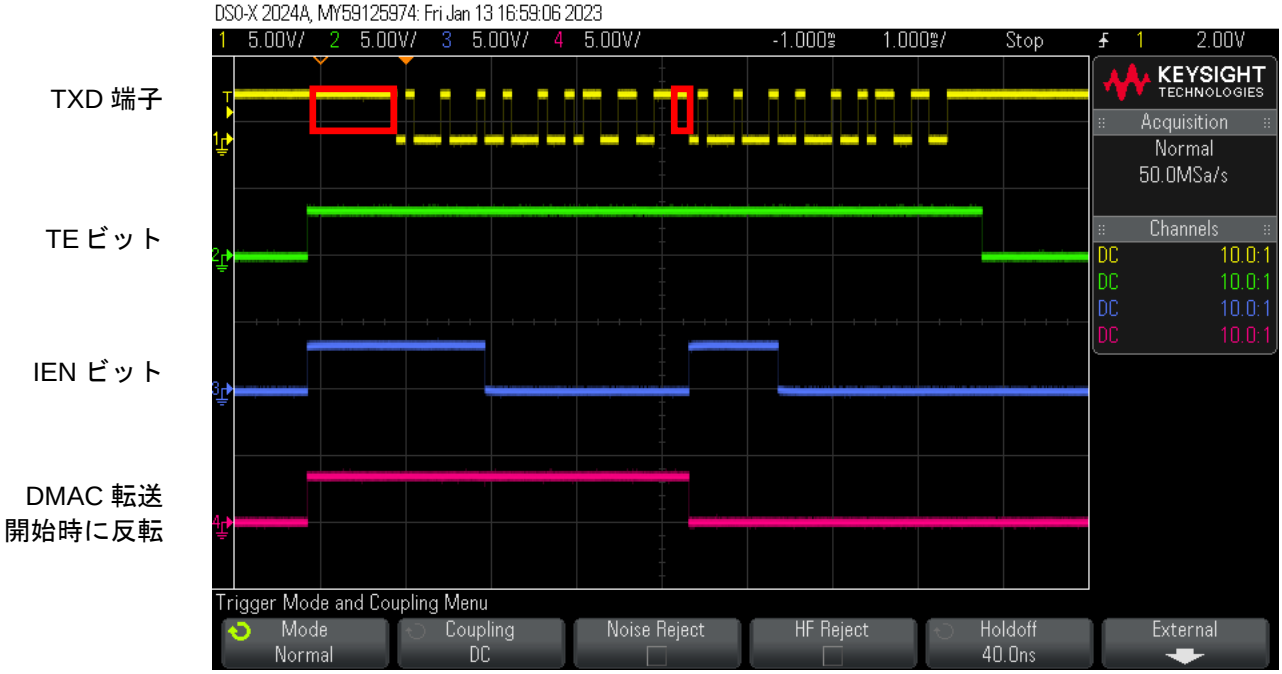


図 3.11　サンプルプログラムの波形

1 回目の送信時、TE=0→1 のタイミングで内部待機期間が発生するが、2 回目の送信時は TE ビットを制御しないため、内部待機期間を発生させずに送信開始していることが確認できます。1 回目の送信に関しても、R_Config_SCI12_Start 関数呼び出し後、1 フレーム以上経過後に R_Config_SCI12_Serial_Send 関数を実行すると内部待機期間は発生しません。

3.3　DTC を使用した制御例

3.3.1　概要

本章では、「3.1 送信データエンプティ割り込みを使用した制御例」と同様に、TE=1 の状態を維持し、DTC 転送で送信データを書き込む場合について説明します。

図 3.12に修正後のタイミング図（DTC転送にて送信データを書き込む場合）を示します。

（修正前のタイミングは、DTC 動作を除き図 3.1と同等です。）

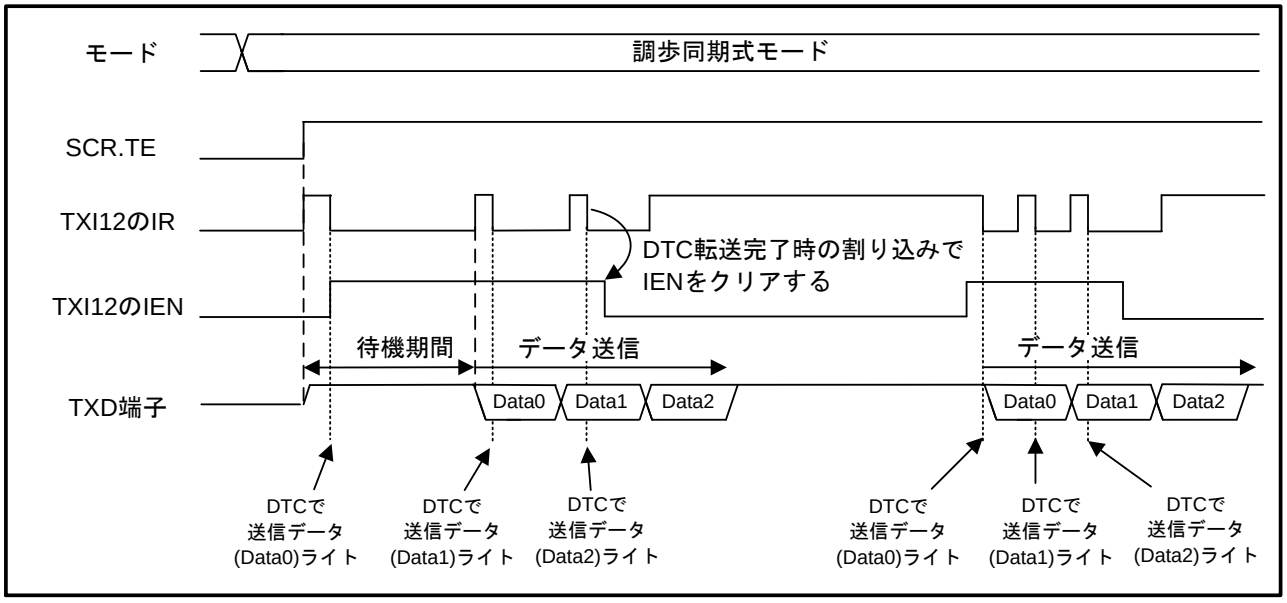


図 3.12　修正後のタイミング図（DTC 転送にて送信データを書き込む場合）

3.3.2　コード生成手順例

「3.1 送信データエンプティ割り込みを使用した制御例」のコード生成手順例と同様にスマート・コンフィグレータを使用してコード生成を行います。

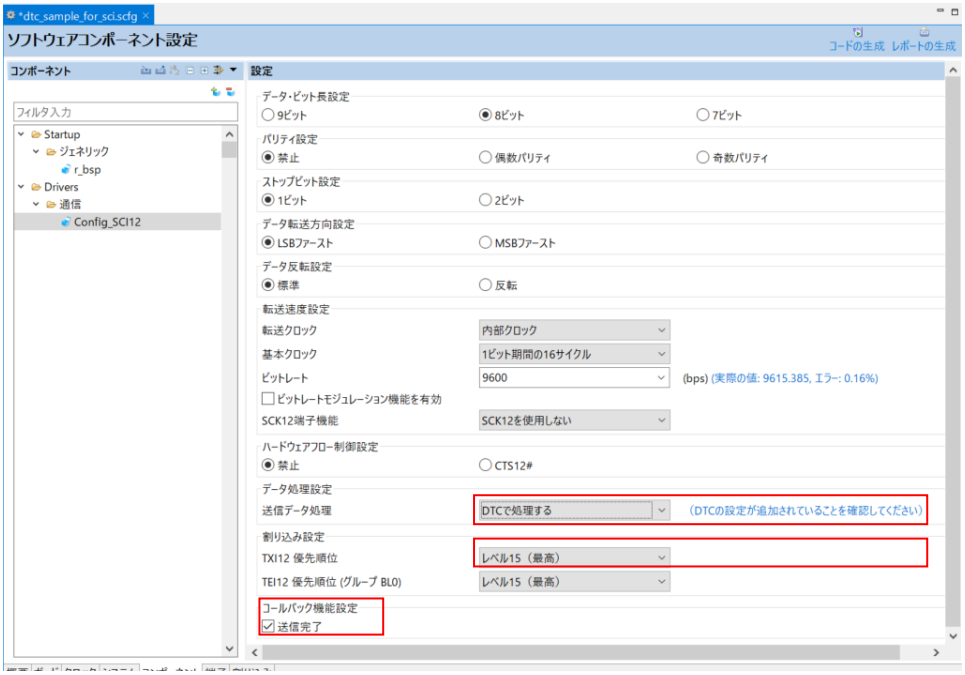
1.　「**3.1.2** コード生成手順例」の手順 1～3 と同様の手順を実行する。

2.　SCI12 のソフトウェアコンポーネント設定を行い、コード生成を実行する。
本サンプルで必須となる設定を以下に示します。

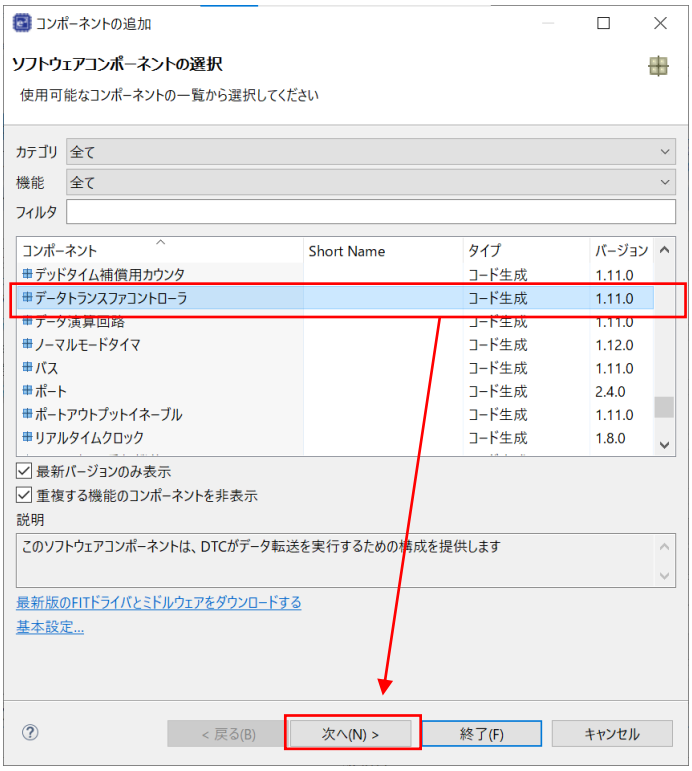
送信データ処理：DTC で処理する

TXI12 優先順位：レベル 15 (最高)

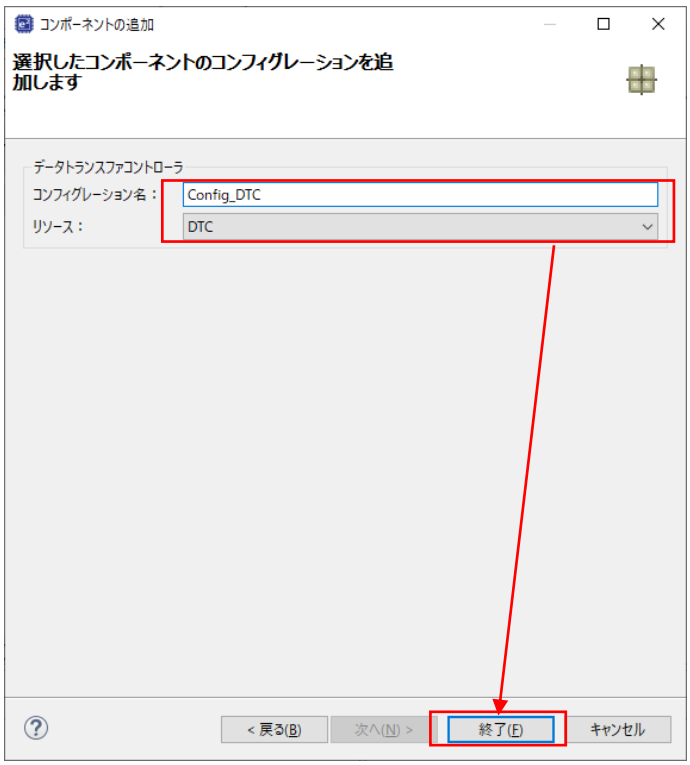
コールバック機能設定：送信完了あり



3. 再度、コンポーネントの追加を行う。「データトランスファコントローラ」を選択し、次へを実行する。



4. コンフィグレーション名、リソースを選択して終了を実行する。
本サンプルでは、コンフィグレーション名を「Config_DTC」、リソースを「DTC」としています。



5. DTC のソフトウェアコンポーネント設定を行い、コード生成を実行する。
本サンプルで必須となる設定を以下に示します。

起動要因：SCI12 (TXI12)

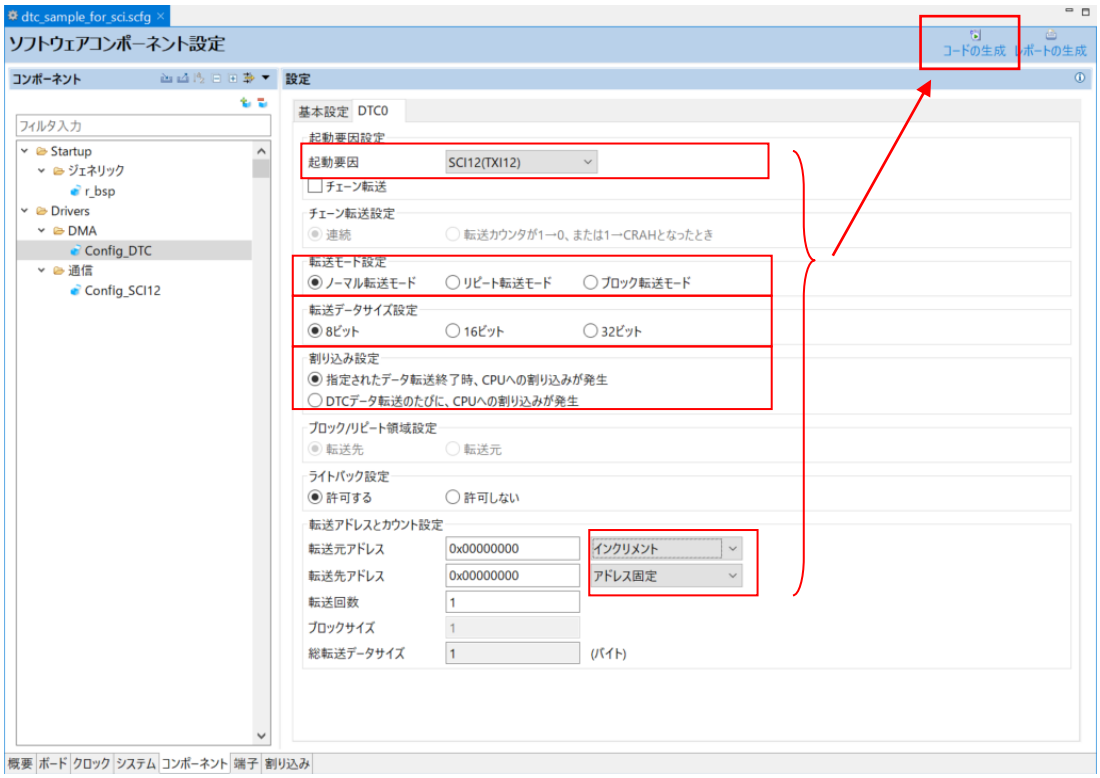
転送モード：ノーマルモード

転送データサイズ：8 ビット

割り込み設定：指定されたデータ転送終了時、CPU への割り込みが発生

転送元アドレス：インクリメント

転送先アドレス：アドレス固定



【注】　転送回数、転送元アドレスの番地、転送先アドレスの番地はソフトウェアで設定するよう変更するため、ここではデフォルト値のままとします。

3.3.3 プログラムの修正

修正する関数を表 3.2に示します。

表 3.3　修正する関数

関数名
R_Config_SCI12_Start
R_Config_SCI12_Serial_Send
R_Config_DTC_Create
R_Systeminit

R_Config_SCI12_Start ()

修正前：

```
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;
}
```

IEN=1 は R_Config_SCI12_Serial_Send 関数で送信開始時に行うため削除。

修正後：

```
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;
}
```

TE=0→1 をトリガにしないため本関数で TE=1 にして R_Config_SCI12_Serial_Send 関数で制御しないようにする。

TE=1 にすることで TXI12 の IR=1 になるが、IEN=0 のため DTC 転送要求が発生せず保留される。

端子設定も TE=1 にするタイミングで設定を行う。

R_Config_SCI12_Serial_Send ()

修正前：

```
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;
}
```

SCI12.SCR.BYTE 及び端子設定は
R_Config_SCI12_Start 関数で設定しているため削除。

修正後：

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

    return MD_OK;
}
```

IEN=1 に設定し DTC に起動要因となる
TXI 割り込みを許可にする。

R_Config_DTC_Create ()

修正前：

```
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();
}
```

引数に転送元アドレス、
転送回数を設定できるように変更。

転送元アドレス、
転送回数を引数から設定。
転送先アドレスは
SCI12.TDR に変更。

修正後：

```
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();
}
```

引数に転送元アドレス、転送回数を設定できるように変更。

転送元アドレス、
転送回数を引数から設定。
転送先アドレスは
SCI12.TDR に変更。

Config_DTC.h ファイルのプロトタイプ宣言修正：

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

R_Systeminit()

修正前：

```
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;
}
```

DTC の設定は、SCI 送信前に送信バッファ及び送信回数を設定する仕様に変更するため、初期設定は削除

修正後：

```
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　サンプルプログラム

サンプルプログラムの動作を示します。

[メイン関数]

(1) 送信設定するために R_Config_SCI12_Start 関数を呼び出す。
(2) R_Config_DTC_Create 関数を呼び出し、DTC の転送元に送信データ、転送先に SCI 送信バッファ、転送回数を設定。
(3) g_flag=0 に設定、送信完了後に g_flag=1 にすることで送信完了まで次の処理を行わない。
(4) R_Config_DTC_Start 関数を実行し、DTC の転送待ち状態にする。
(5) R_Config_SCI12_Serial_Send 関数を呼び出し、DTC の転送要因となる TXI 割り込みを許可。TXI 割り込み発生タイミングで DTC 転送が発生し、送信バッファに送信データが書き込まれる。
(6) 送信完了まで待機。
(7) 300us の待ち時間を挿入。
(8) (2)～(7)を 2 回繰り返す。
(9) 送信終了後、送信設定を解除のため R_Config_DTC_Stop 関数および R_Config_SCI12_Stop 関数を呼び出す。

【注】R_Config_SCI12_Create 関数は main 関数に到達する前の R_Systeminit 関数で呼び出されているため、main 関数での呼び出しは不要です。

```
/***********************************************
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();
    }
}
```

図 3.13　サンプルプログラム main 関数

[SCI12 送信完了コールバック関数]

(1) SCI12 送信完了コールバック関数(r_Config_SCI12_callback_transmitend)は、DTC の転送完了時、および SCI 転送完了時に呼び出される。
(2) 送信完了割り込みが禁止状態(TEIE=0)の場合は、DTC 転送完了時の呼び出しと判断し、TXI 割り込み要求の禁止、および送信完了割り込みの許可(TEIE=1)の設定を行う
(3)送信完了割り込みが許可状態(TEIE=1)の場合は、送信完了時の呼び出しと判断し、送信完了割り込みの禁止、および送信完了フラグの設定(g_flag=1)を行う

```
/***********************************************
* 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 */
}
```

図 3.14　サンプルプログラム r_Config_SCI12_callback_transmitend 関数

3.3.5　サンプルプログラムの動作

DTC を使用した制御例を動作させた場合の波形を図 3.15に示します。

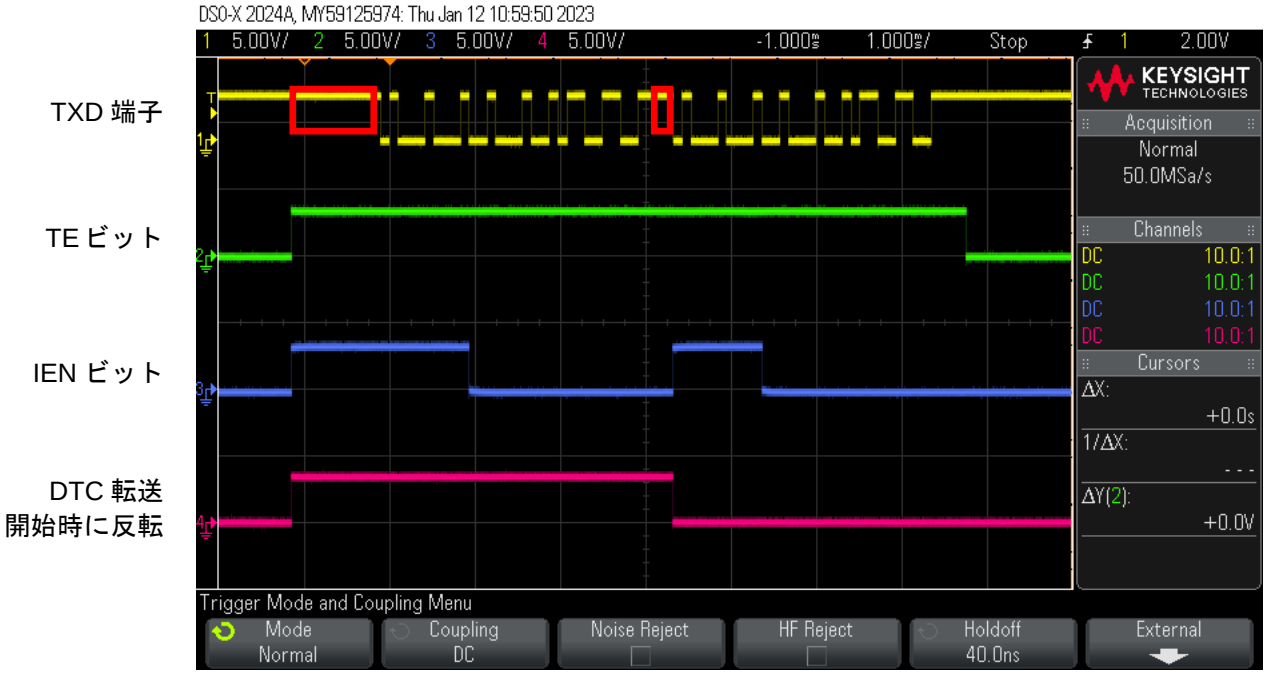


図 3.15　サンプルプログラムの波形

1 回目の送信時、TE=0→1 のタイミングで内部待機期間が発生するが、2 回目の送信時は TE ビットを制御しないため、内部待機期間を発生させずに送信開始していることが確認できます。1 回目の送信に関しても、R_Config_SCI12_Start 関数呼び出し後、1 フレーム以上経過後に R_Config_SCI12_Serial_Send 関数を実行すると内部待機期間は発生しません。

4.　ビルド時に自動的にコードが再生成される機能を無効にする方法

スマート・コンフィグレータにはビルド時に自動的にコードを再生成する機能があります。

本サンプルプログラムでは、コード生成したコンポーネントを修正して使用していますのでこの機能を無効にしています。

手動でコードを再生成する場合はコンポーネントが上書きされるのでご注意ください。

手動でコード生成する場合は、trash フォルダからサルベージするか、あらかじめバックアップを取るなどの対応をしてください。

図 4.1に e² studio でのビルド時にコード再生成無効にする方法を示します。

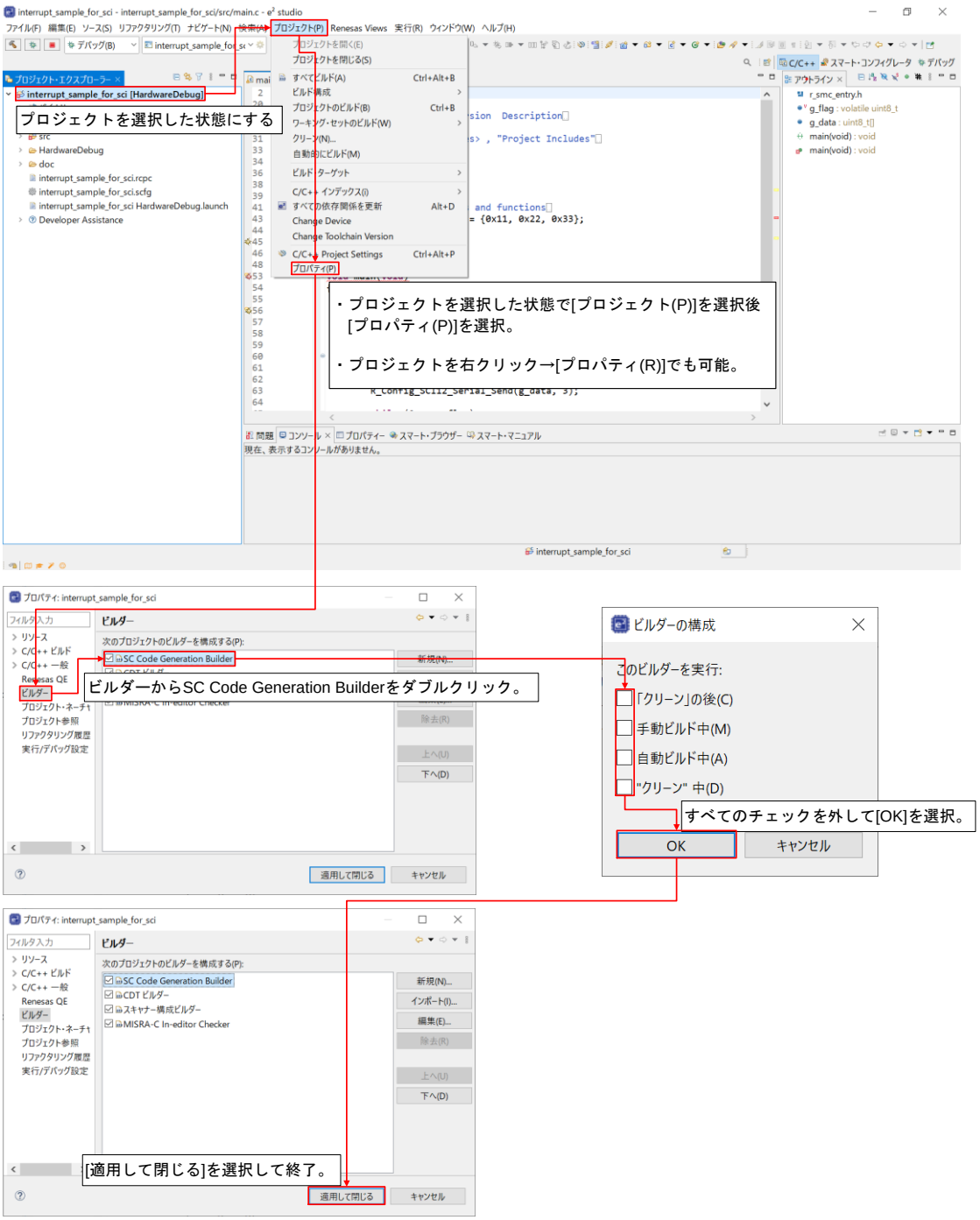


図 4.1　ビルド時に自動的にコードが再生成される機能を無効にする方法

5.　プロジェクトをインポートする方法

サンプルプログラムは e² studio のプロジェクト形式で提供しています。本章では、 e² studio および CS+ へプロジェクトをインポートする方法を示します。インポート完了後、ビルドおよびデバッグの設定を確認してください。

5.1　e² studio での手順

e² studio でご使用になる際は、下記の手順で e² studio にインポートしてください。

なお、e² studio で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、 空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する e² studio のバージョンによっては画面が異なる場合があります。)

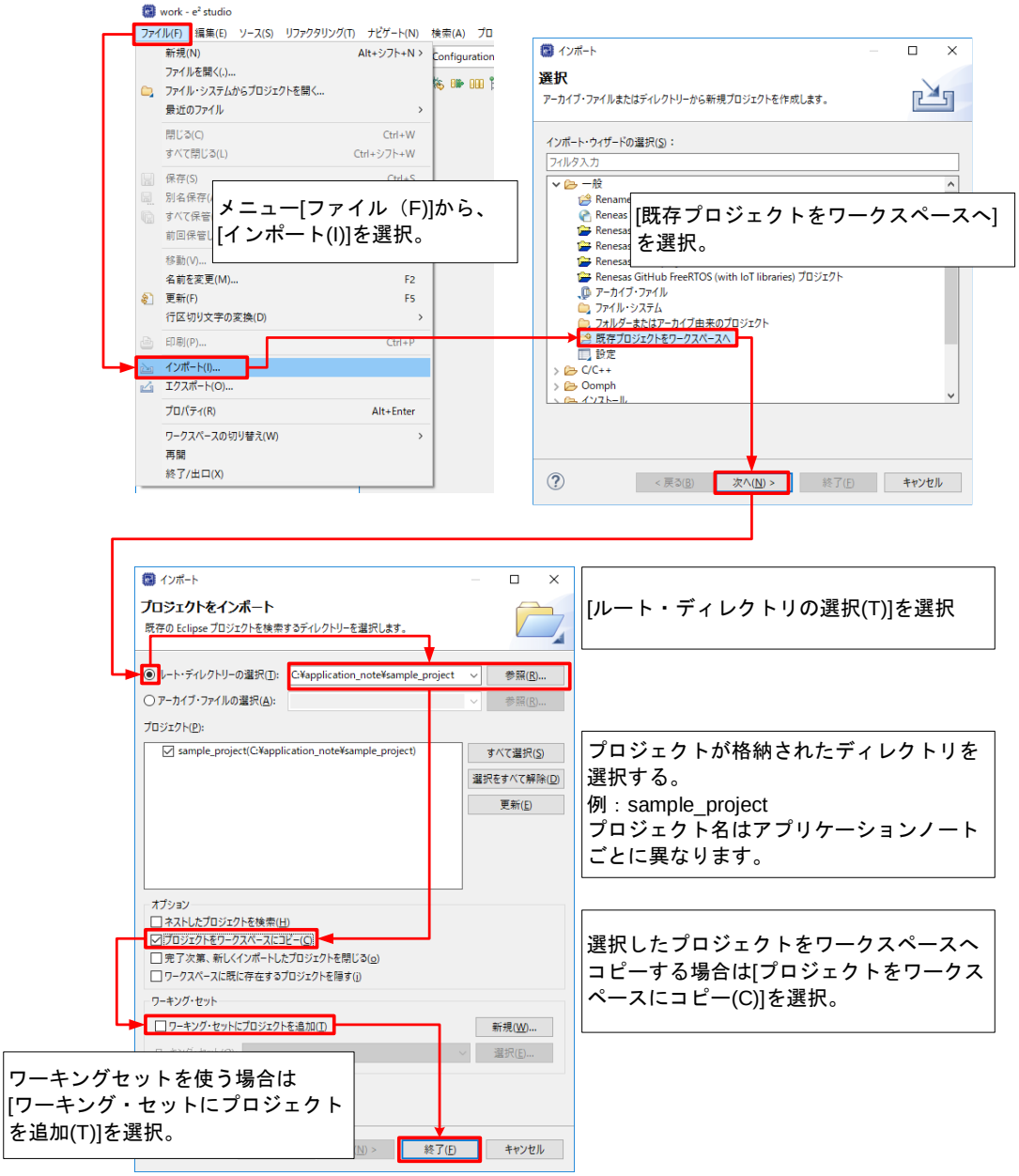


図 5.1　プロジェクトを e² studio にインポートする方法

5.2　CS+での手順

CS+でご使用になる際は、下記の手順で CS+にインポートしてください。

なお、CS+で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、 空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する CS+のバージョンによっては画面が異なる場合があります。)

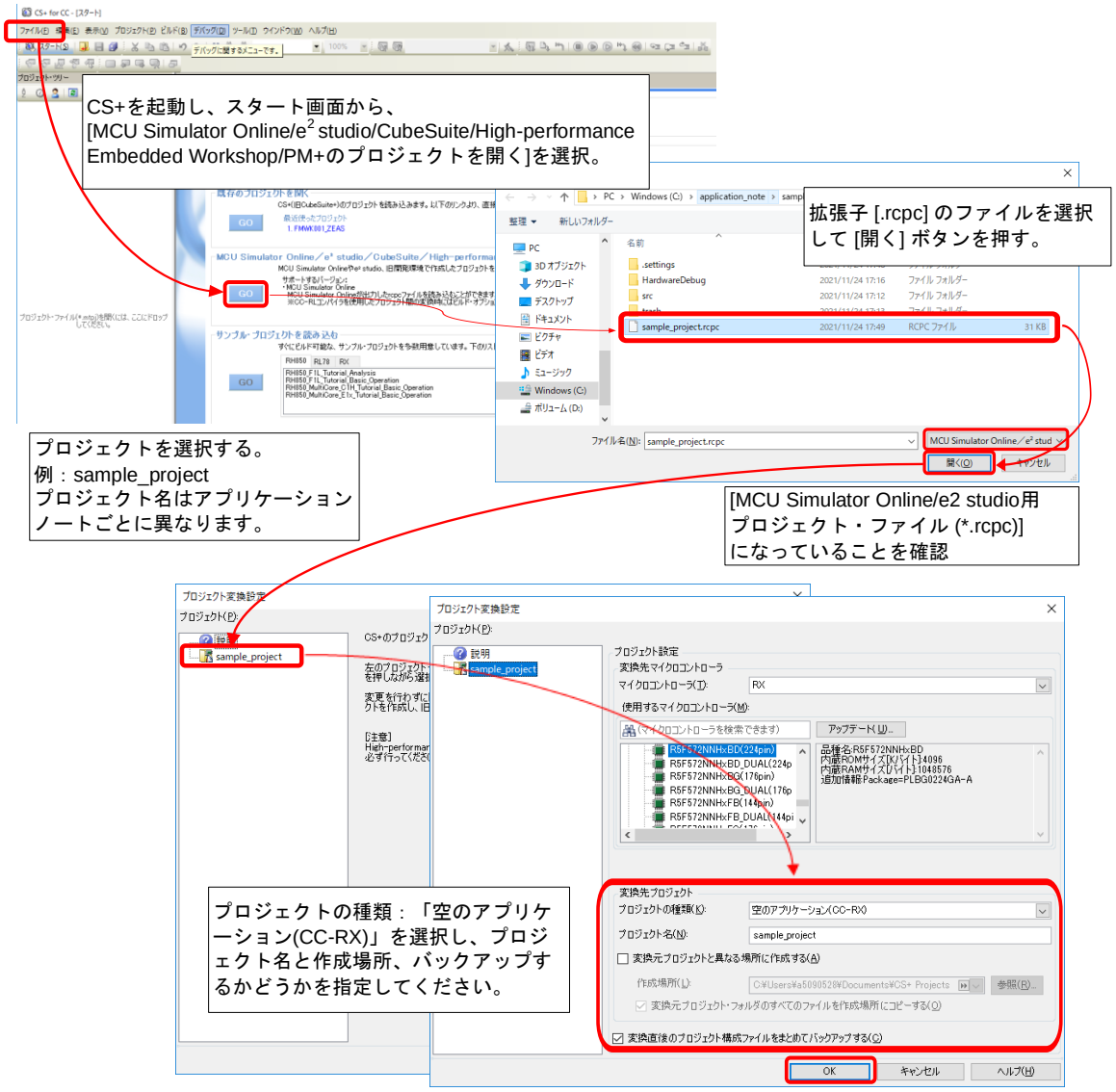


図5.2　プロジェクトを CS+にインポートする方法

6. 参考資料

- RX660 グループ ユーザーズマニュアル ハードウェア編 (R01UH0937)
- Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
- Renesas Starter Kit for RX660 ユーザーズマニュアル (R20UT5017)
- Renesas Starter Kit for RX660 CPU ボード回路図 (R20UT5016)

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Mar.20.23	-	初版発行
1.01	Mar.21.24	1	対象ツール追加

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。