ルネサス RX ファミリ

Tracealyzer®を用いた FreeRTOS のデバッグ

要旨

FreeRTOS は Amazon Web Services の RTOS で、高性能の組み込み型カーネルをベースとしています。

Percepio Tracealyzer® (以降 Tracealyzer®)は、RTOS または Linux ベースの組み込みソフトウェアシステム開発者向けの便利なビジュアルトレース診断ソリューションです。

Tracealyzer®は、RTOS の以下のような内部状態の詳細をリアルタイムにモニタすることができます。

①RTOS システムコールの利用状況
②タスク毎の CPU 負荷率とその積算
③タスク毎のヒープ利用状況とその積算

本アプリケーションノートでは、e² studio でのアプリケーション開発における FreeRTOS のスレッドとオブジェクトの状態（リソース）をチェックする手順を説明します。Tracealyzer®の起動手順についても説明します。

動作確認デバイス

RX65N グループ(R5F565NEHDFB)

動作環境

ターゲットボード	CK-RX65N
統合開発環境(IDE)	e² studio バージョン 2023-01
トレースツール	Percepio Tracealyzer® v4.6.6
OS	FreeRTOS 10.4.3
ツールチェーン	CC-RX V3.05
USB-シリアル変換	Pmod USBUART モジュール(Digilent 社製)

【注】 あらかじめ、以下の URL のドキュメントを参照し、e2 studio、CC-RX、Tracealyzer®をダウンロードしてください。

- 統合開発環境 e² studio 2020-04、e² studio v7.8 ユーザーズマニュアル 入門ガイドのダウンロードサイト：
  e² studio 2020-04、e² studio v7.8 ユーザーズマニュアル入門ガイド
- RX スマート・コンフィグレータ ユーザーガイド: e² studio 編のダウンロードサイト：
  RX スマート・コンフィグレータ ユーザーガイド: e² studio 編
- Tracealyzer® for FreeRTOS ユーザーマニュアルのサイト：
  Tracealyzer® for FreeRTOS - User Manual
- Percepio Tracealyzer®のダウンロードサイト：
  Download Tracealyzer® - Percepio AB

目次

1. Tracealyzer®のインストール

Tracealyzer® for FreeRTOS User Manual を参照してインストールしてください。

2. e² studio プロジェクトの作成

e² studio にはプロジェクト生成ウィザードが用意されており、RX 用プロジェクトを簡単に生成することができます。

“統合開発環境 e2 studio 2020-04、e2 studio v7.8 ユーザーズマニュアル 入門ガイド”を参照し、e² studio と CC-RX をインストールしてください。

“RX スマート・コンフィグレータ ユーザーガイド: e² studio 編”の「2.2 RTOS プロジェクトの作成」の章を参照して CC-RX の RTOS プロジェクトを作成してください。

このとき、RTOS 及びターゲット・デバイスの情報を次のように入力します。

— RTOS：FreeRTOS (kernel only)【注】
— RTOS Version：**10.4.3-rx-1.0.5**
— Target Board：**CK-RX65N**

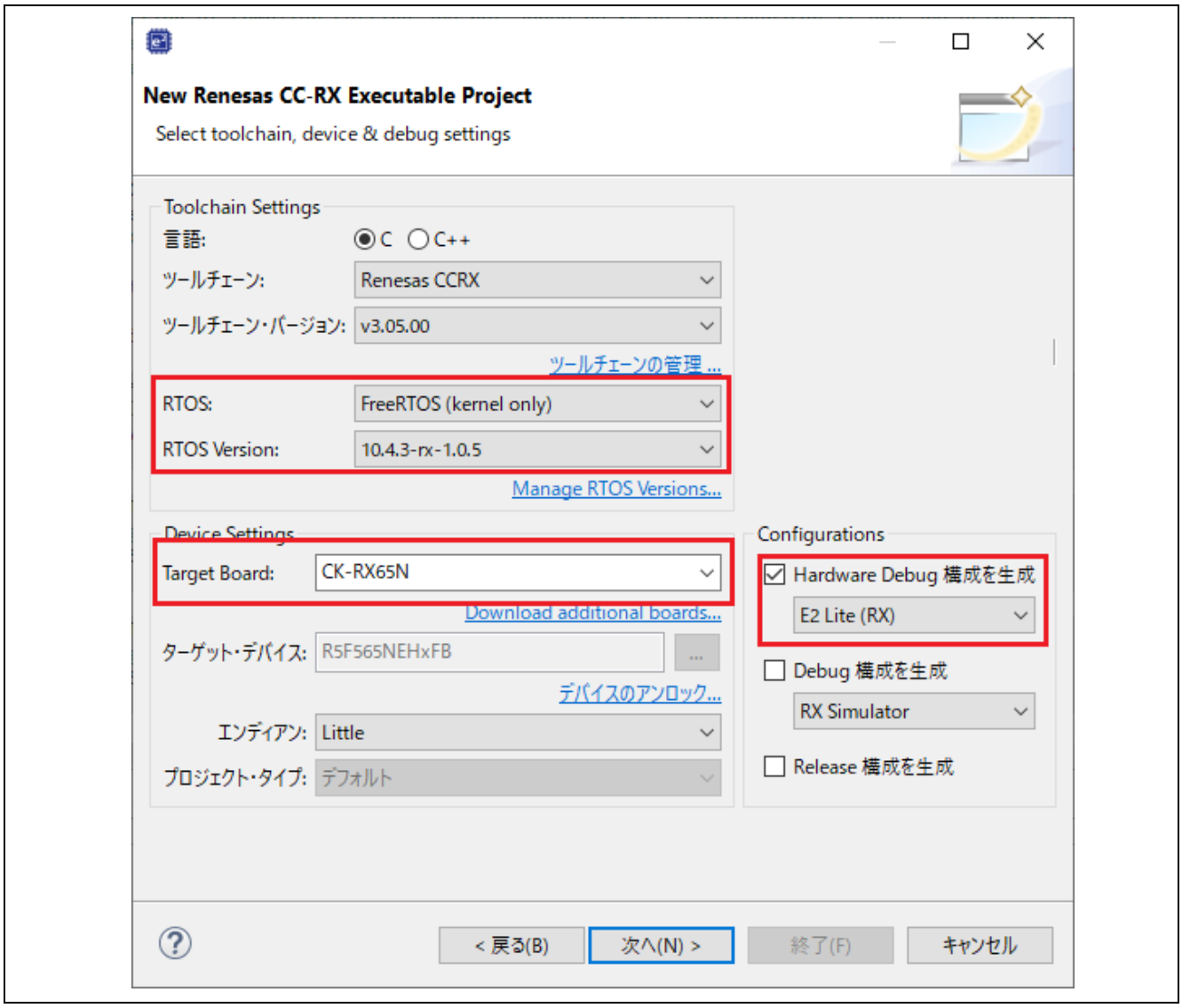

図 2-1 RTOS とターゲット・デバイスの選択

【注】 RTOS : FreeRTOS (with IoT Libraries) の場合、モニタデータをネットワーク通信路(Ethernet)等に重畳することができます。今後この方法も追記する予定です。FreeRTOS (kernel only)の場合、モニタデータの送出の為 SCI(UART モード)を 1ch 占有します。

3. Tracealyzer®を用いたデバッグ（UART 使用）

この章では、UART を用いて Tracealyzer®を使用する方法を説明します。

3.1 プロジェクトへの Tracealyzer® for FreeRTOS の組み込み

3.1.1 Tracealyzer®インストールフォルダに Tracealyzer® for FreeRTOS ソースファイルのコピー

Windows のファイルエクスプローラーで、"Program Files¥Percepio¥Tracealyzer 4¥FreeRTOS¥TraceRecorder"フォルダをワークスペースフォルダ"src"にコピーします。

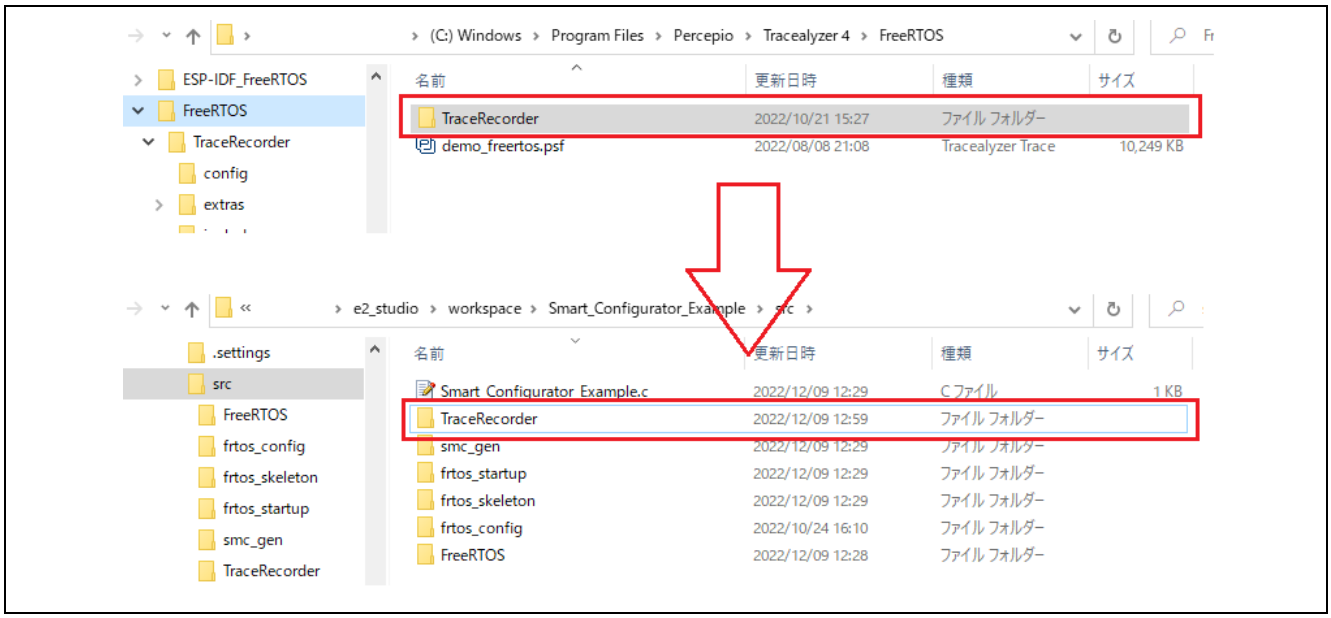

図 3-1　フォルダのコピー

3.1.2 不要フォルダの削除

ワークスペースフォルダ"src/TraceRecorder/streamports"にあるサブフォルダをすべて削除します。

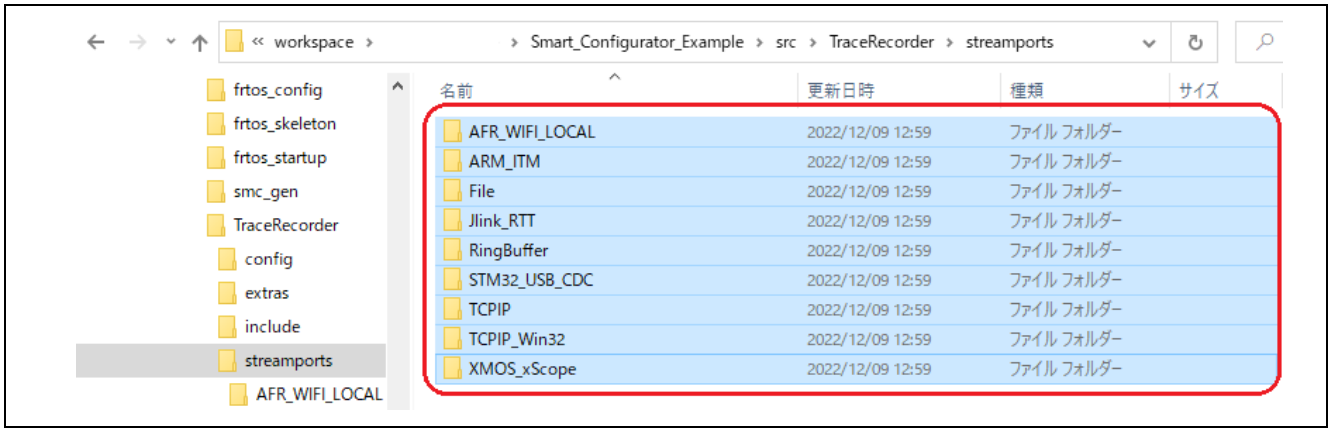

図 3-2　フォルダの削除

3.1.3　UART 通信用のファイルの作成

e² studio のプロジェクト・エクスプローラ上で、streamports フォルダ下に「Renesas_RX_UART」フォルダとその内部に config フォルダ、include フォルダを作り、「trcStreamPort.c」「trcStreamPort.h」「trcStreamPortConfig.h」「Readme-Streamport.txt」の名前でそれぞれ空のファイルを作成します。

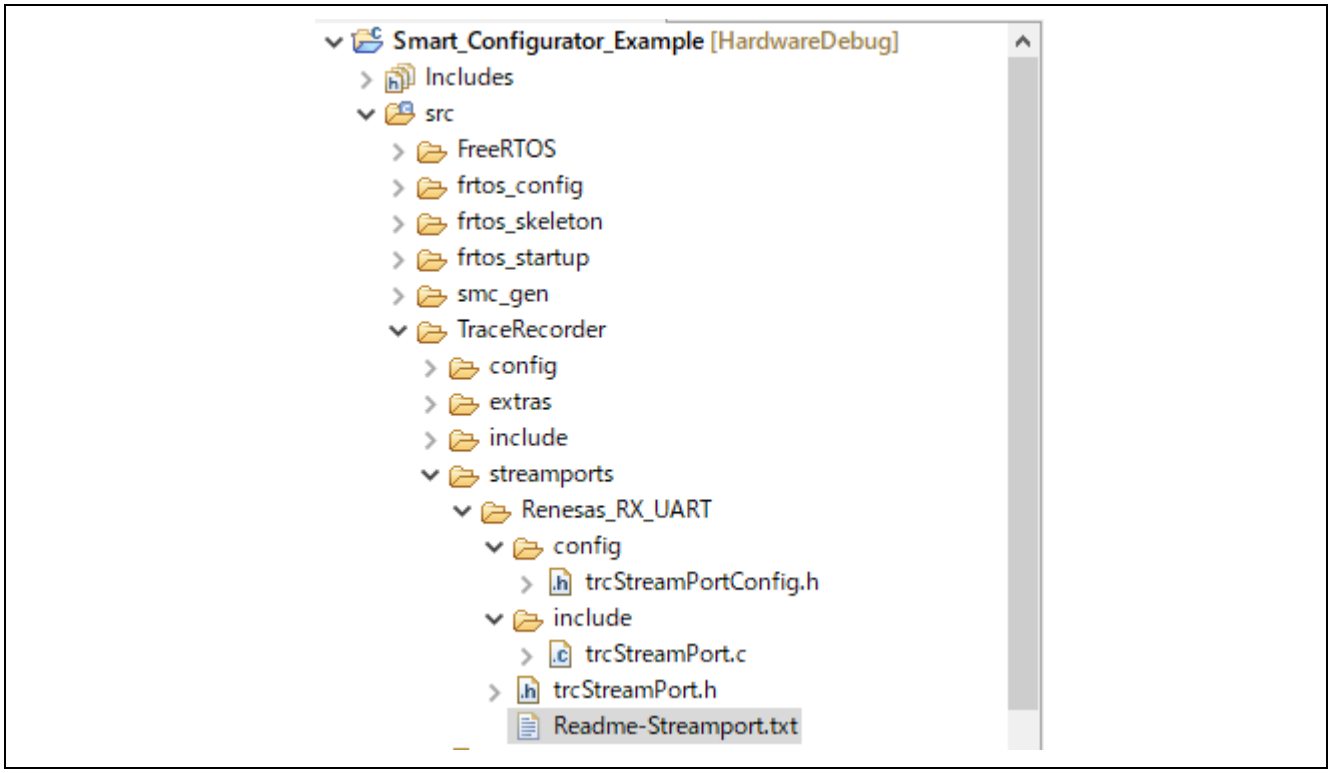

図 3-3　フォルダとファイルの作成

trcStreamPort.c に以下をコピーペーストします。

```
#include <string.h>

#include "trcRecorder.h"
#include "r_sci_rx_if.h"
#include "r_sci_rx_pinset.h"

#if (TRC_CFG_RECORDER_MODE == TRC_RECORDER_MODE_STREAMING)
#if (TRC_USE_TRACEALYZER_RECORDER == 1)

static uint8_t string[1024];
static uint8_t sci_buffer[1024];
static uint32_t sci_current_received_size = 0;
static volatile uint32_t wait_sending = 0;

extern sci_hdl_t sci_handle_tracealyzer;

void sci_callback_tracealyzer(void *arg);

traceResult xTraceStreamPortInitialize(TraceStreamPortBuffer_t* pxBuffer)
{
    TRC_ASSERT_EQUAL_SIZE(TraceStreamPortBuffer_t,
TraceStreamPortUSBBuffers_t);
```

```
      if (pxBuffer == 0)
      {
            return TRC_FAIL;
      }

      return xTraceInternalEventBufferInitialize(pxBuffer->buffer,
sizeof(pxBuffer->buffer));
  }

  traceResult prvTraceUARTTransmit(void* pvData, uint32_t uiSize, int32_t*
piBytesSent)
  {
      int32_t error_code = -1;

      while(1)
      {
            if(wait_sending)
            {
                  xTraceKernelPortDelay(1);
            }
            else
            {
                  break;
            }
      }

      if(uiSize < sizeof(string))
      {
            memcpy(string, pvData, uiSize);
            if(SCI_SUCCESS == R_SCI_Send(sci_handle_tracealyzer, string,
uiSize))
            {
                  wait_sending = 1;
                  *piBytesSent = uiSize;
                  error_code = 0;
            }
      }
      return error_code;
  }

  traceResult prvTraceUARTReceive(void* data, uint32_t uiSize, int32_t*
piBytesReceived)
  {
      if(sci_current_received_size == uiSize)
      {
            memcpy(data, sci_buffer, sci_current_received_size);
            *piBytesReceived = sci_current_received_size;
            sci_current_received_size = 0;
      }
      return 0;
  }

  void sci_callback_tracealyzer(void *arg)
  {
      sci_cb_args_t   *p_args;
      p_args = (sci_cb_args_t *)arg;

      if (SCI_EVT_RX_CHAR == p_args->event)
      {
            R_SCI_Receive(p_args->hdl,
&sci_buffer[sci_current_received_size], 1);
```

```
            if(sci_current_received_size == (sizeof(sci_buffer) - 1)) /* -1
means string terminator after "¥n" */
            {
                  sci_current_received_size = 0;
            }
            else
            {
                  sci_current_received_size++;
            }
      }
      else if(SCI_EVT_TEI == p_args->event)
      {
            wait_sending = 0;
      }
  }

  #endif
  #endif
```

trcStreamPortConfig.h に以下をコピーペーストします。

```
  #ifndef TRC_STREAM_PORT_CONFIG_H
  #define TRC_STREAM_PORT_CONFIG_H

  #ifdef __cplusplus
  extern "C" {
  #endif

  /*******************************************************************************
******
  * Configuration Macro: TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE
  *
  * Specifies the size of the internal buffer.
  *******************************************************************************
****/
  #define TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE 1024

  #ifdef __cplusplus
  }
  #endif

  #endif /* TRC_STREAM_PORT_CONFIG_H */
```

trcStreamPort.h に以下をコピーペーストします。

```
  #ifndef TRC_STREAM_PORT_H
  #define TRC_STREAM_PORT_H

  #include <trcTypes.h>
  #include <trcStreamPortConfig.h>

  #ifdef __cplusplus
  extern "C" {
  #endif

  typedef struct TraceStreamPortBuffer
  {
      uint8_t buffer[(TRC_CFG_STREAM_PORT_INTERNAL_BUFFER_SIZE) +
sizeof(TraceUnsignedBaseType_t)];
  } TraceStreamPortBuffer_t;

  traceResult prvTraceUARTReceive(void* data, uint32_t uiSize, int32_t*
piBytesReceived);

  traceResult prvTraceUARTTransmit(void* pvData, uint32_t uiSize, int32_t*
piBytesSent);

  /**
   * @internal Stream port initialize callback.
   *
   * This function is called by the recorder as part of its initialization
phase.
   *
   * @param[in] pxBuffer Buffer
   *
   * @retval TRC_FAIL Initialization failed
   * @retval TRC_SUCCESS Success
   */
  traceResult xTraceStreamPortInitialize(TraceStreamPortBuffer_t* pxBuffer);

  /**
   * @brief Allocates data from the stream port.
   *
   * @param[in] uiSize Allocation size
   * @param[out] ppvData Allocation data pointer
   *
   * @retval TRC_FAIL Allocate failed
   * @retval TRC_SUCCESS Success
   */
  #define xTraceStreamPortAllocate(uiSize, ppvData) ((void)uiSize,
xTraceStaticBufferGet(ppvData))

  /**
   * @brief Commits data to the stream port, depending on the
implementation/configuration of the
   * stream port this data might be directly written to the stream port
interface, buffered, or
   * something else.
   *
   * @param[in] pvData Data to commit
   * @param[in] uiSize Data to commit size
   * @param[out] piBytesCommitted Bytes committed
   *
```

```
   * @retval TRC_FAIL Commit failed
   * @retval TRC_SUCCESS Success
   */
  #define xTraceStreamPortCommit xTraceInternalEventBufferPush

  /**
   * @brief Writes data through the stream port interface.
   *
   * @param[in] pvData Data to write
   * @param[in] uiSize Data to write size
   * @param[out] piBytesWritten Bytes written
   *
   * @retval TRC_FAIL Write failed
   * @retval TRC_SUCCESS Success
   */
  #define xTraceStreamPortWriteData prvTraceUARTTransmit

  /**
   * @brief Reads data through the stream port interface.
   *
   * @param[in] pvData Destination data buffer
   * @param[in] uiSize Destination data buffer size
   * @param[out] piBytesRead Bytes read
   *
   * @retval TRC_FAIL Read failed
   * @retval TRC_SUCCESS Success
   */
  #define xTraceStreamPortReadData prvTraceUARTReceive

  #define xTraceStreamPortOnEnable(uiStartOption) ((void)(uiStartOption),
TRC_SUCCESS)

  #define xTraceStreamPortOnDisable() (TRC_SUCCESS)

  #define xTraceStreamPortOnTraceBegin() (TRC_SUCCESS)

  #define xTraceStreamPortOnTraceEnd() (TRC_SUCCESS)

  #ifdef __cplusplus
  }
  #endif

  #endif /* TRC_STREAM_PORT_H */
```

Readme-Streamport.txt には何も記入する必要はありません。

FreeRTOSConfig.h の一番下に#include "trcRecorder.h" を追加します。

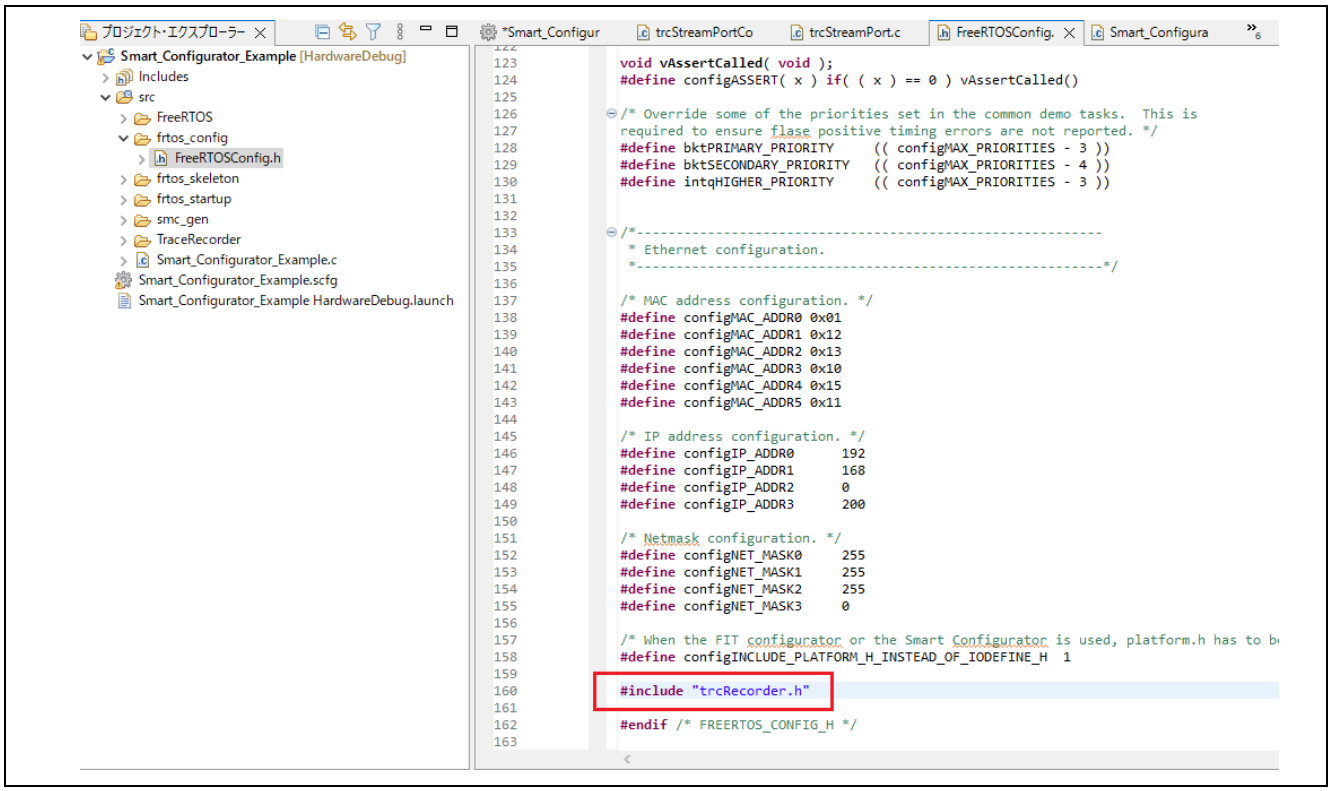

図 3-4　FreeRTOSConfig.h にコードの追加

trcConfig.h を以下のように変更します。

#error ... の行をコメントアウトします。
TRC_CFG_HARDWARE_PORT に TRC_HARDWARE_PORT_Renesas_RX600 を指定します。

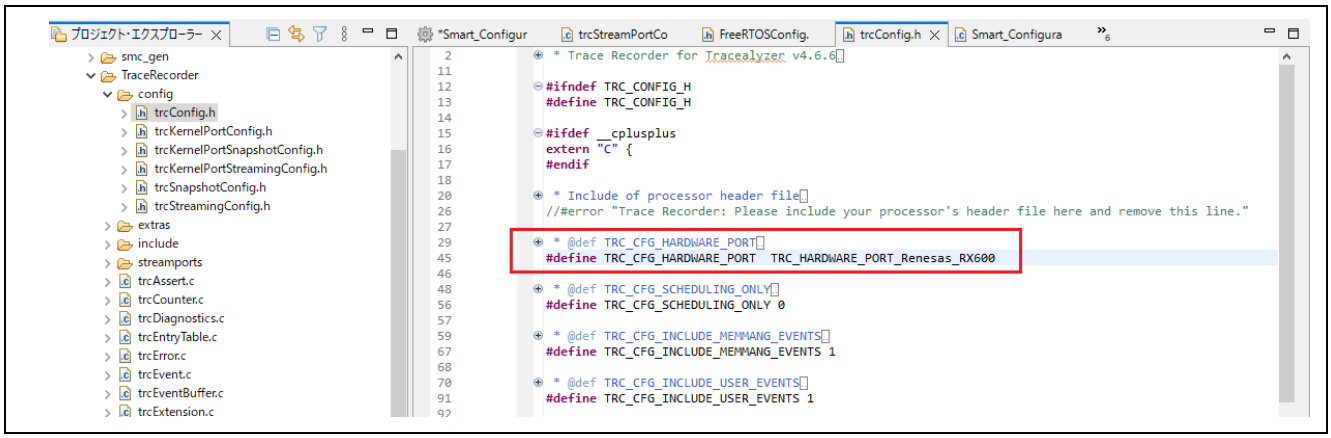

図 3-5　　trcConfig.h の修正

trcKernelPortConfig.h を以下のように変更します。

TRC_CFG_RECORDER_MODE に TRC_RECORDER_MODE_STREAMING を指定
TRC_CFG_FREERTOS_VERSION に TRC_FREERTOS_VERSION_10_4_1 を指定

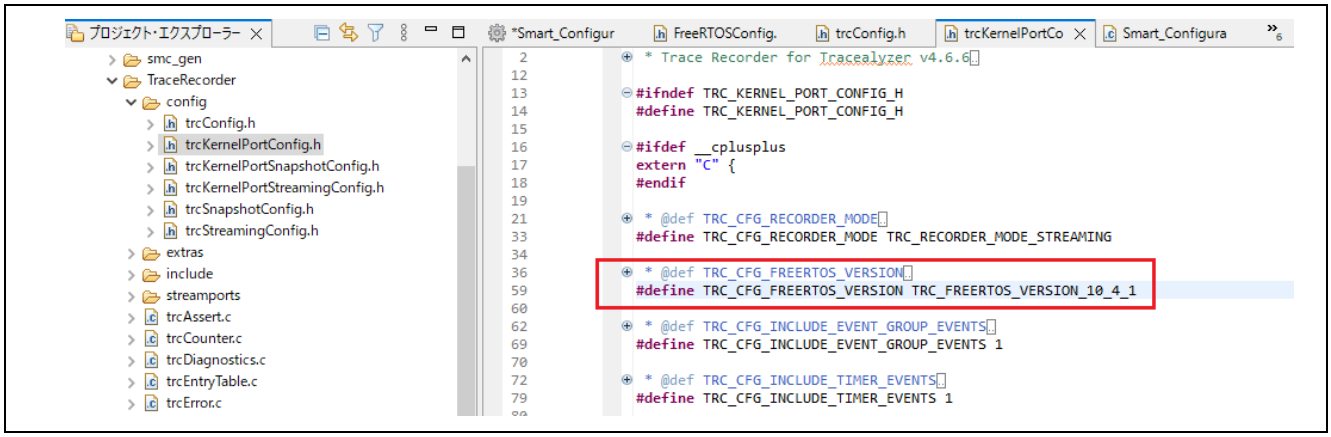

図 3-6　trcKernelPortConfig.h の修正

3.2　プロジェクト側の設定方法

3.2.1　Tracealyzer®のモニタデータ出力用の UART の設定

スマート・コンフィグレータで SCI の FIT モジュールを追加/設定します。

図 3-7 のように、ソフトウェアコンポーネントの選択のダイアログを開き、フィルタに「SCI」と入力し、「SCI Driver」を選択して[終了]します。

フィルタに「SCI」を入力しても SCI の FIT モジュールが表示されない場合は、「最新版の FIT ドライバとミドルウェアをダウンロードする」を選択して、ダウンロード後に SCI モジュールを追加設定してください

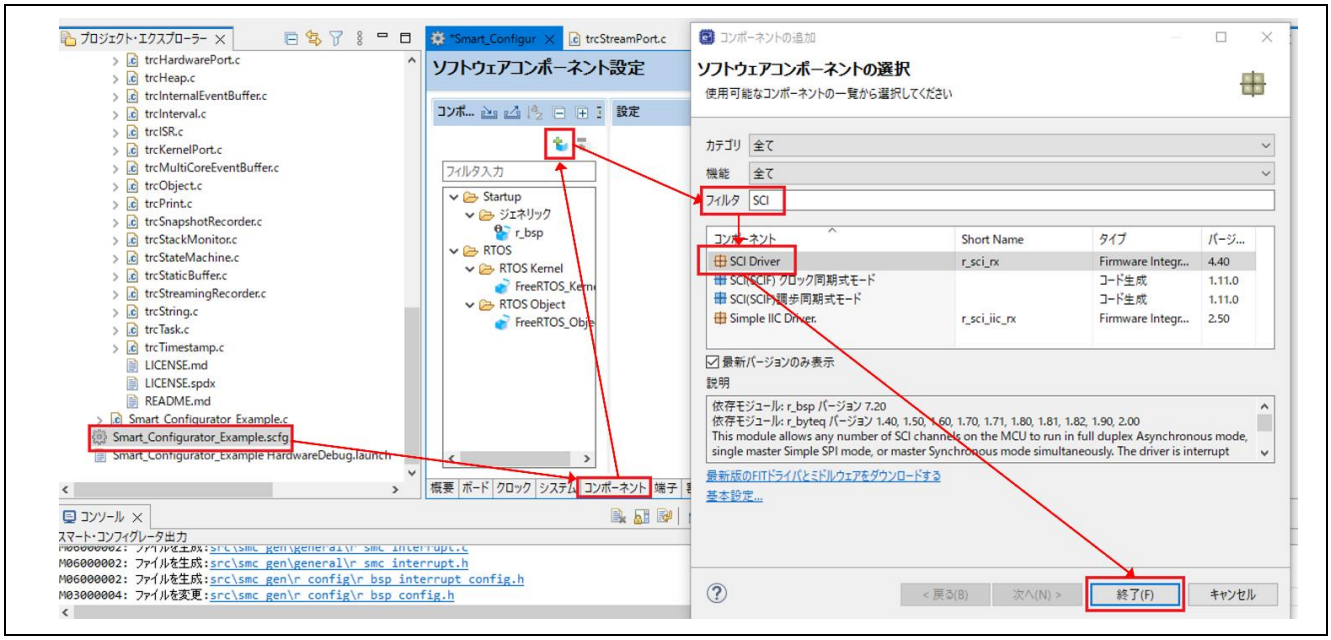

図 3-7　　SCI の FIT モジュールの追加

CK-RX65N の PMOD1 を使用しますので、SCI チャネル 6 を使用します。

コンポーネントタブで[r_sci_rx]を選択し、SCI チャネル 6 を[Include]に設定します。

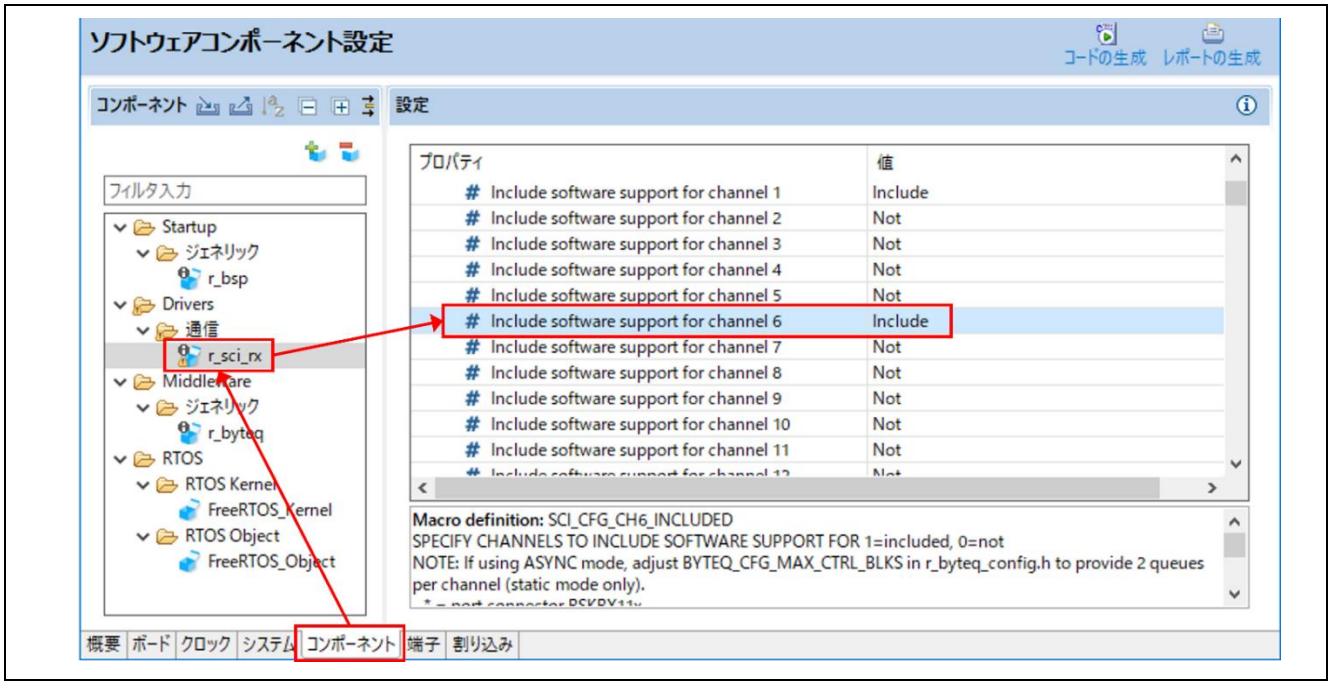

図 3-8　　SCI チャネル 6 の設定

チャネル 6 の送信バッファの容量を 80 バイトから 1024 バイトに変更します。

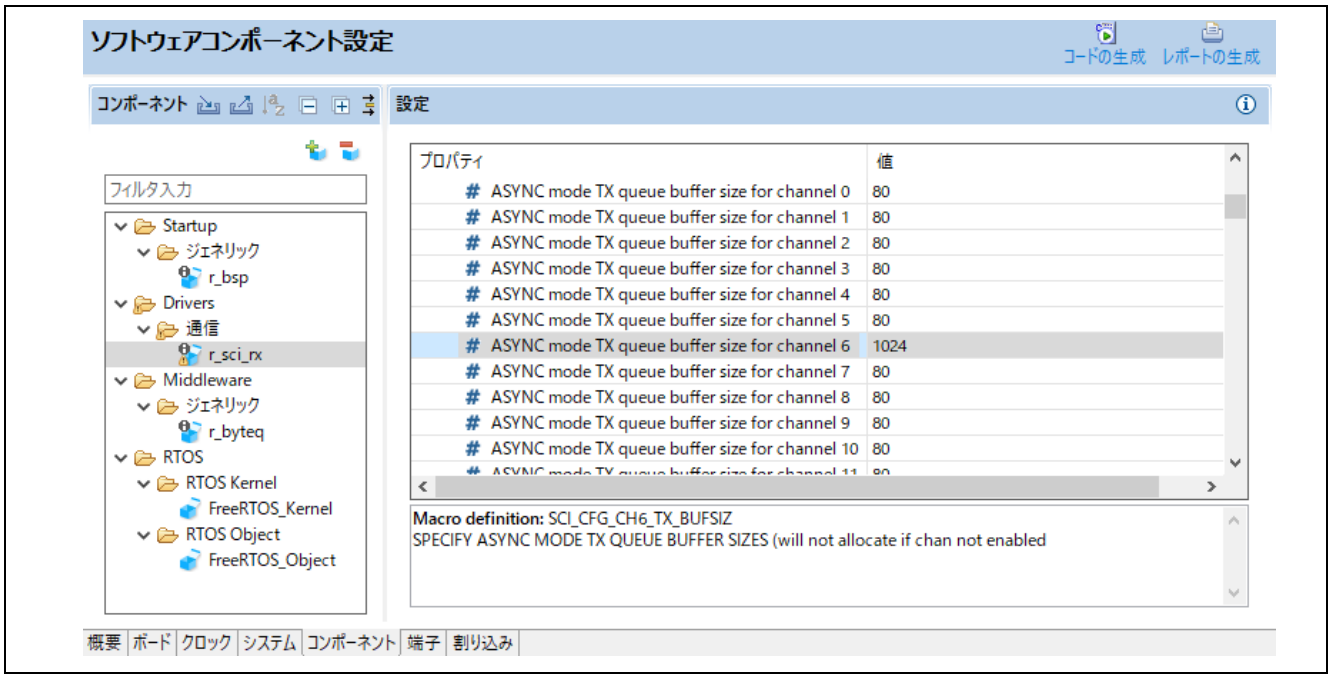

図 3-9　SCI チャネル 6 の送信バッファの変更

チャネル 6 の送信バッファ空割り込みを使用する設定に変更します。

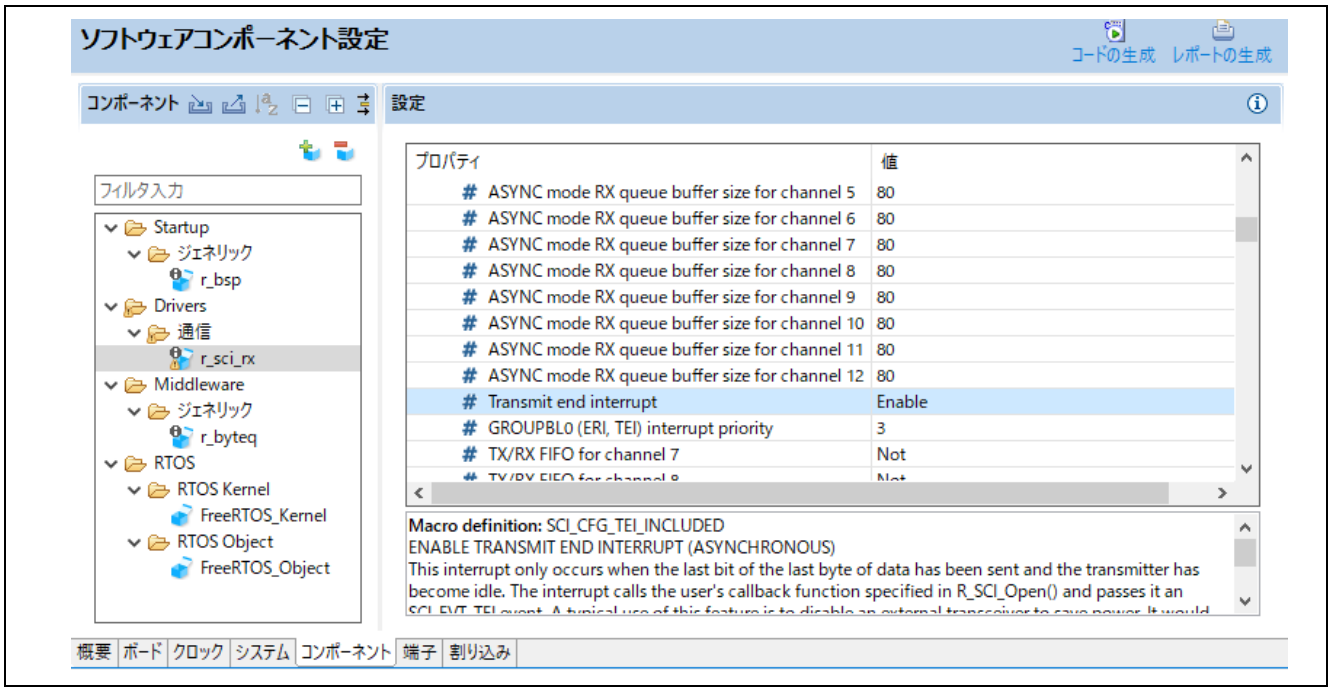

図 3-10　SCI チャネル 6 の割込みの設定

チャネル 6 をフロー制御無し UART で利用するので、フロー制御用端子(RTS/CTS)を無効化し、送受信用端子(TxD/RxD)のみ有効にします。

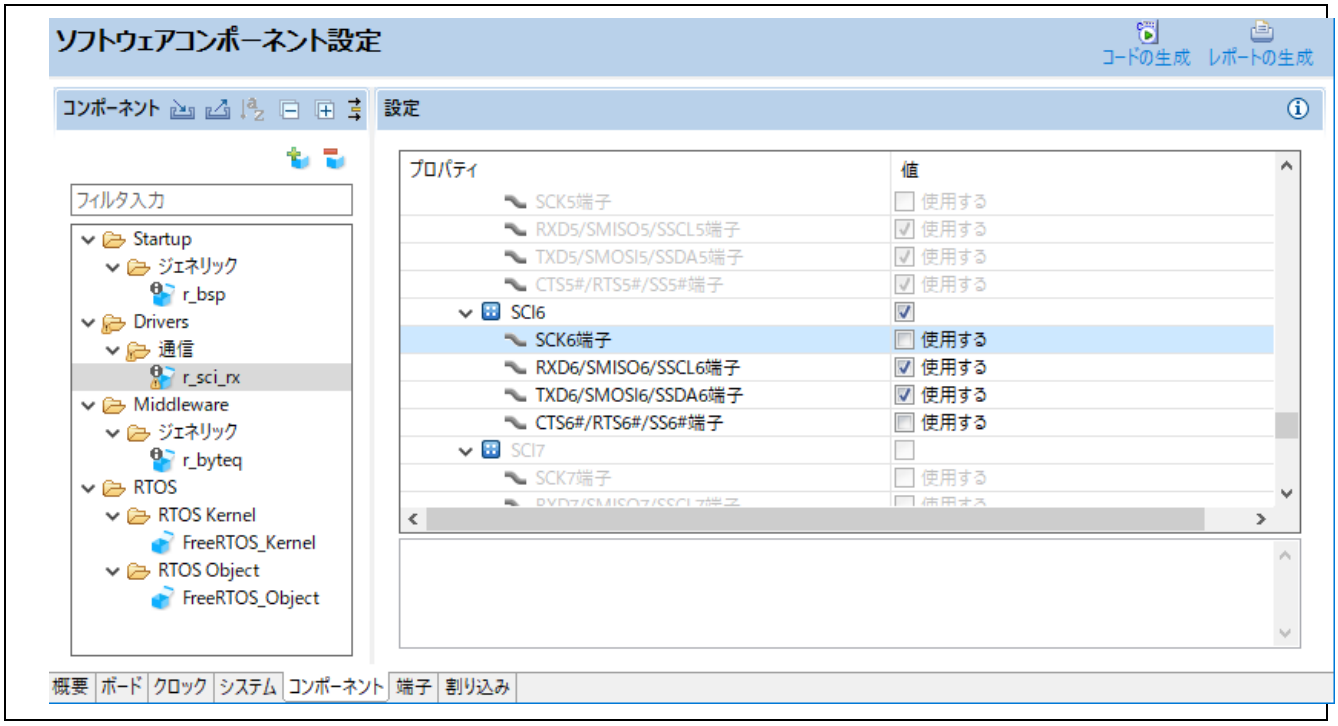

図 3-11　SCI チャネル 6 の端子設定

端子タブで SCI チャネル 6 を設定します。

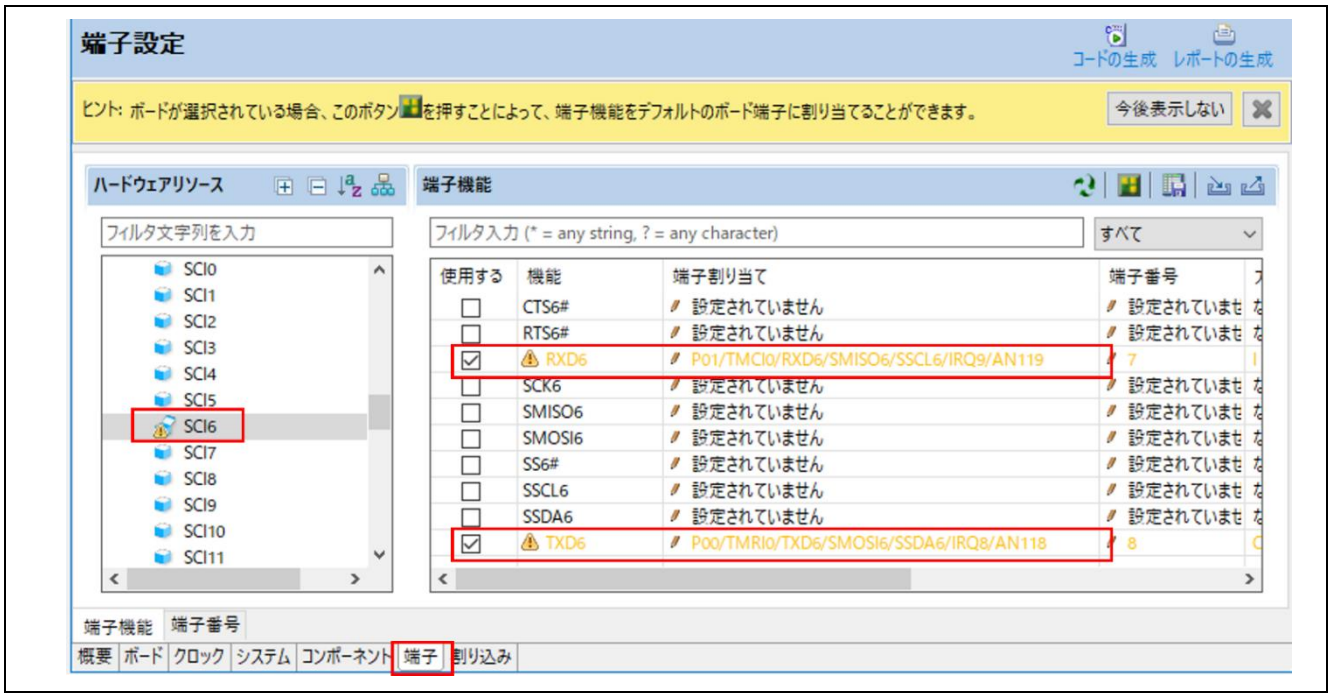

図 3-12　SCI チャネル 6 の端子機能設定

3.3　コンパイラの設定

3.3.1　コンパイラ設定で Tracealyzer®に必要なインクルードパスを追加

プロジェクト・エクスプローラでプロジェクト名を右クリックし、プロパティを選択します。

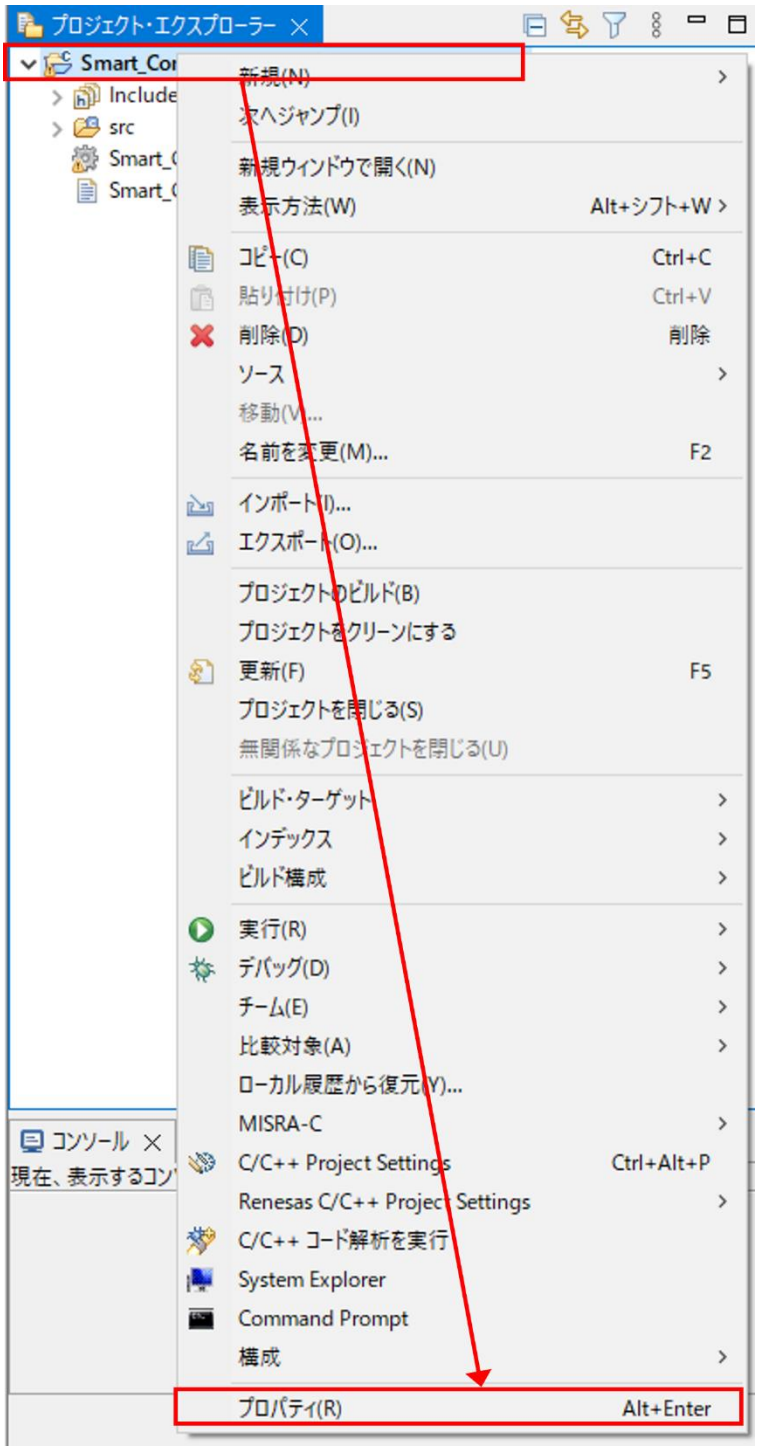

図 3-13　プロジェクトのプロパティ

C/C++ビルド -> 設定 -> ツール設定 -> Compiler -> ソース -> 追加 ボタンを押します。

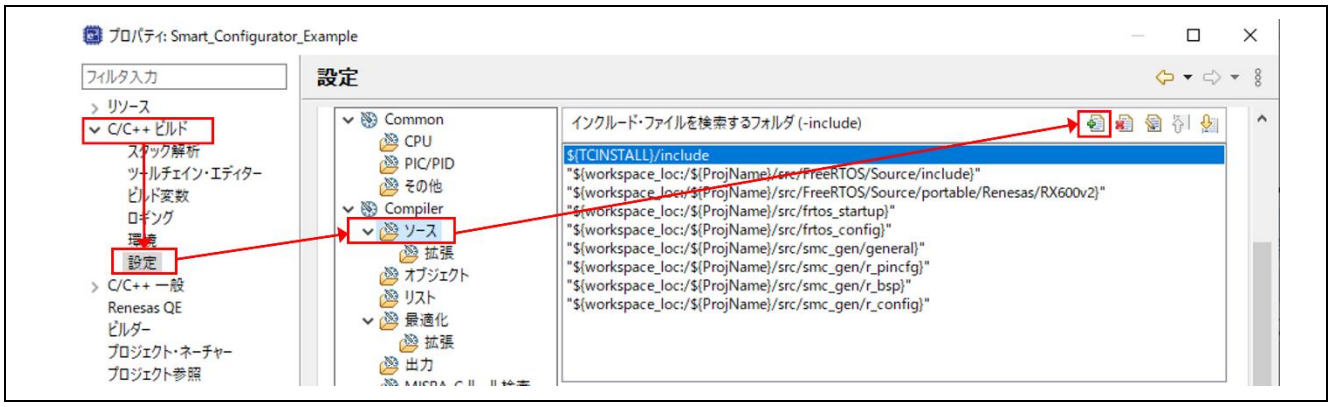

図 3-14　パスの追加

以下 5 種類のパスを追加します。

"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rx65n/register_access/ccrx}"
"${workspace_loc:/${ProjName}/src/TraceRecorder/config}"
"${workspace_loc:/${ProjName}/src/TraceRecorder/include}"
"${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/Renesas_RX_UART/config}"
"${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/Renesas_RX_UART/include}"

注意："${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rx65n/register_access/ccrx}" は、スマート・コンフィグレータによりコード生成が実行される度に削除されるため、その都度再度同じパスを設定する必要があります。

3.4　FreeRTOS の設定

3.4.1　FreeRTOS カーネルの portmacro.h の修正

FreeRTOS カーネル の portmacro.h を Tracealyzer®の呼び出しに対応させるために修正します。

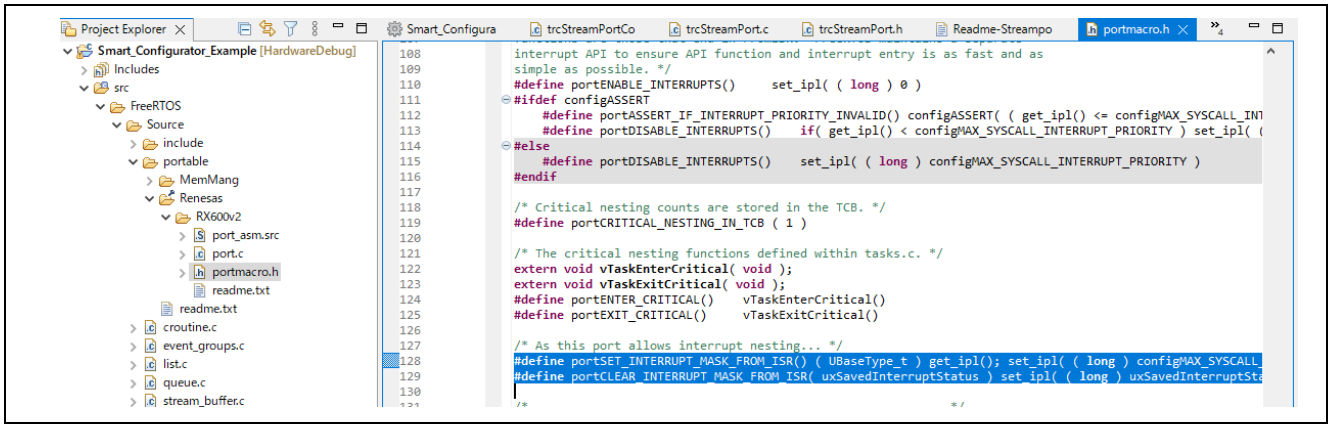

図 3-15　portmacro.h

/* As this port allows interrupt nesting... */以下を以下のように修正します。

```
  /* As this port allows interrupt nesting... */
        static int32_t set_interrupt_mask_from_isr( void );
        static int32_t set_interrupt_mask_from_isr( void )
        {
         int32_t tmp = __get_ipl();
         __set_ipl( ( long ) configMAX_SYSCALL_INTERRUPT_PRIORITY );
         return tmp;
        }
        #define portSET_INTERRUPT_MASK_FROM_ISR()
set_interrupt_mask_from_isr()
        #define portCLEAR_INTERRUPT_MASK_FROM_ISR( uxSavedInterruptStatus )
set_ipl( ( long ) uxSavedInterruptStatus )
```

3.4.2　FreeRTOS カーネル起動前のフック関数の修正

FreeRTOS カーネル起動前のフック関数(freertos_start.c の Processing_Before_Start_Kernel())に Tracealyzer®および SCI の初期化コードを挿入します。

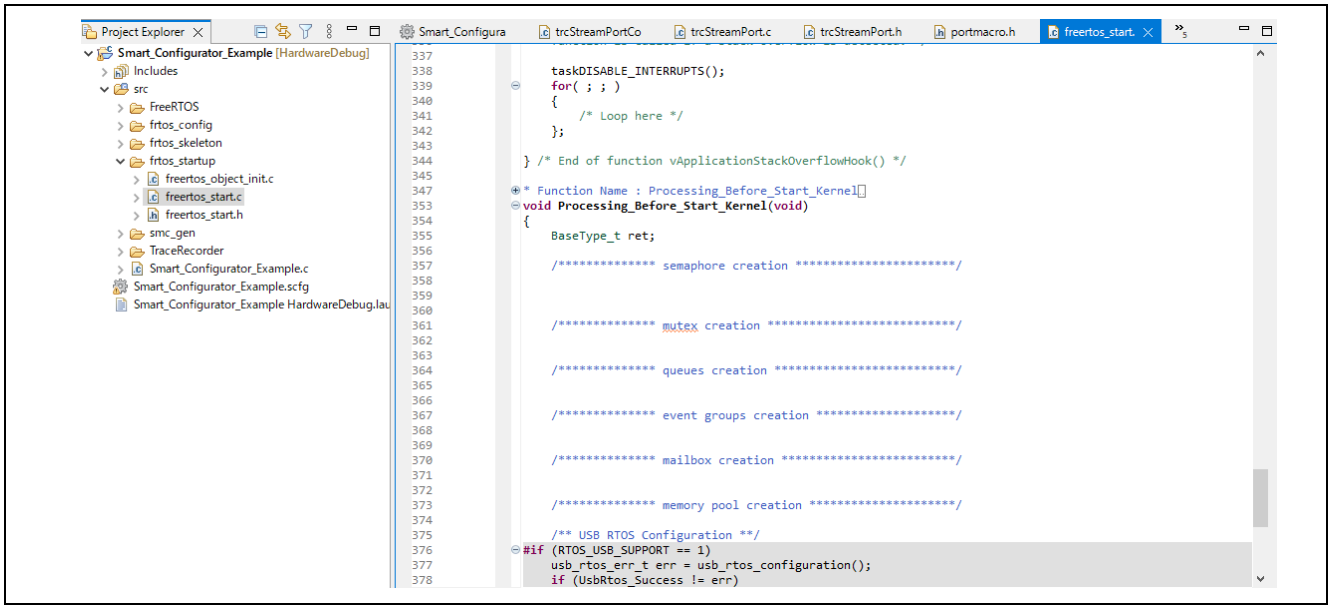

図 3-16　freertos_start.c

```
  #include "r_sci_rx_if.h"
  #include "r_sci_rx_pinset.h"

  static sci_cfg_t my_sci_config;
  sci_hdl_t sci_handle_tracealyzer;

  extern void sci_callback_tracealyzer(void *arg);

  void Processing_Before_Start_Kernel(void)
  {
      BaseType_t ret;

      /* Create all other application tasks here */
      /* Set up the configuration data structure for asynchronous (UART)
operation. */
      my_sci_config.async.baud_rate    = 921600;
      my_sci_config.async.clk_src     = SCI_CLK_INT;
      my_sci_config.async.data_size    = SCI_DATA_8BIT;
      my_sci_config.async.parity_en    = SCI_PARITY_OFF;
      my_sci_config.async.parity_type  = SCI_EVEN_PARITY;
      my_sci_config.async.stop_bits    = SCI_STOPBITS_1;
      my_sci_config.async.int_priority = 15; /* disable 0 - low 1 - 15 high
*/

      R_SCI_Open(SCI_CH6, SCI_MODE_ASYNC, &my_sci_config,
sci_callback_tracealyzer, &sci_handle_tracealyzer);
      R_SCI_PinSet_SCI6();

      xTraceInitialize();
```

3.4.3　main タスクに Tracealyzer®の動作開始のコードの挿入

main タスク(Smart_Configurator_Example.c)で Tracealyzer®の動作開始のコード xTraceEnable(TRC_START); を挿入します。

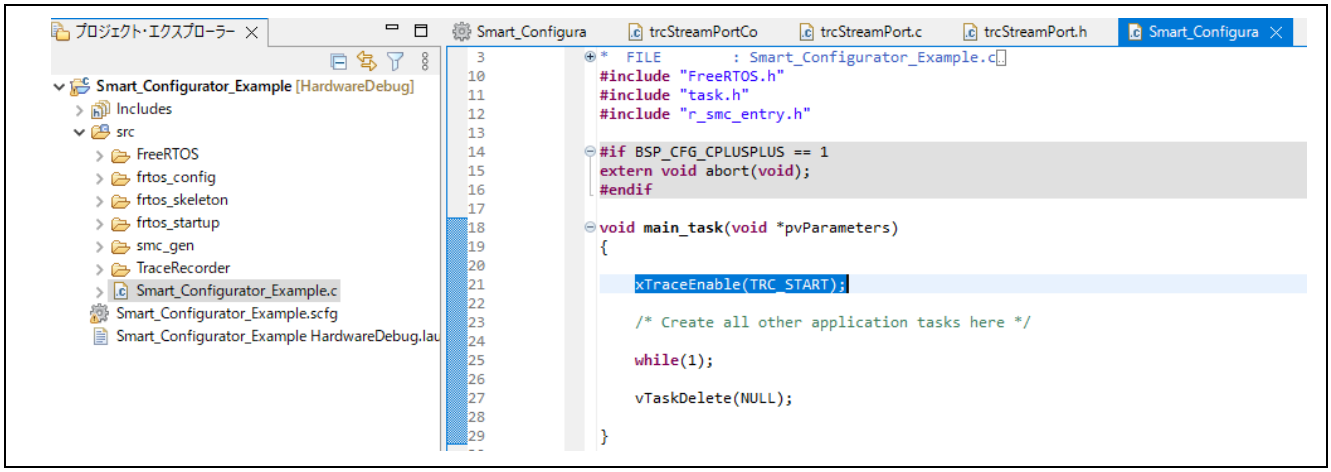

図 3-17　main タスク

なお、プロジェクトのヒープサイズは、Tracealyzer®の利用時は 128KB 以上を推奨します。

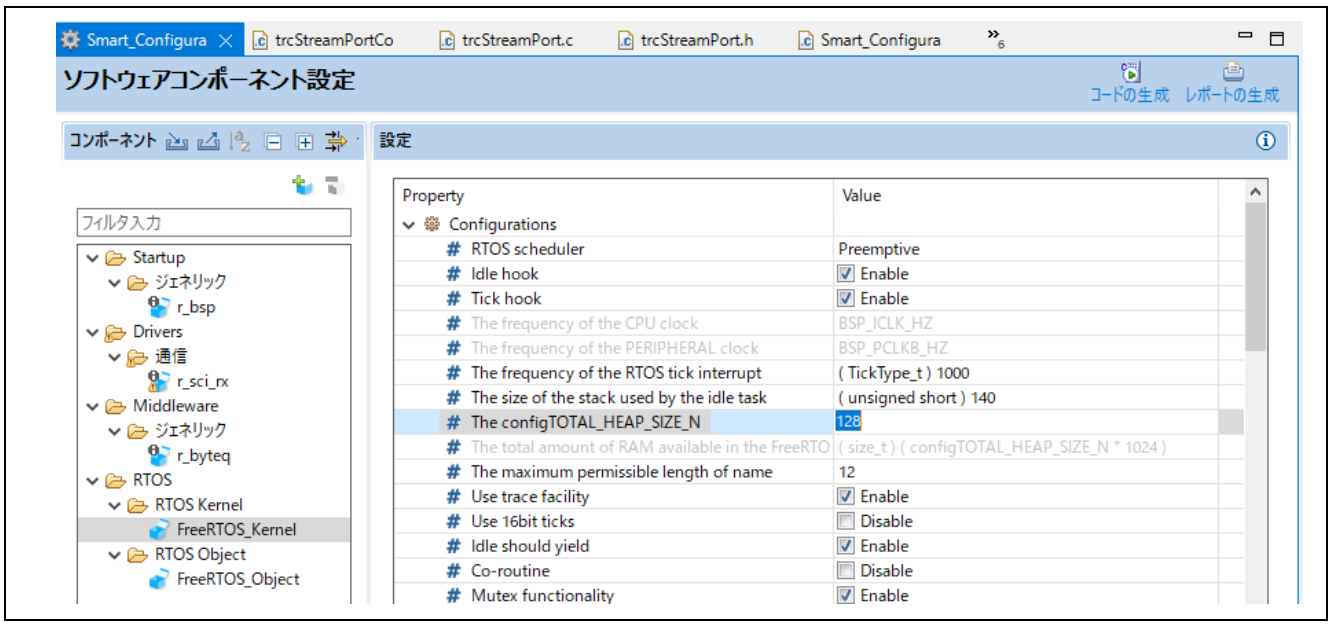

図 3-18　ヒープサイズの変更

3.4.4　プロジェクトのビルド

プロジェクトを右クリックして、[プロジェクトのビルド]を選択します。エラーがないことを確認してください。

3.5　ホスト PC と CK-RX65N ボードの接続

ホスト PC と CK-RX65N との接続には Pmod USBUART モジュール(Digilent 社製)を使用します。

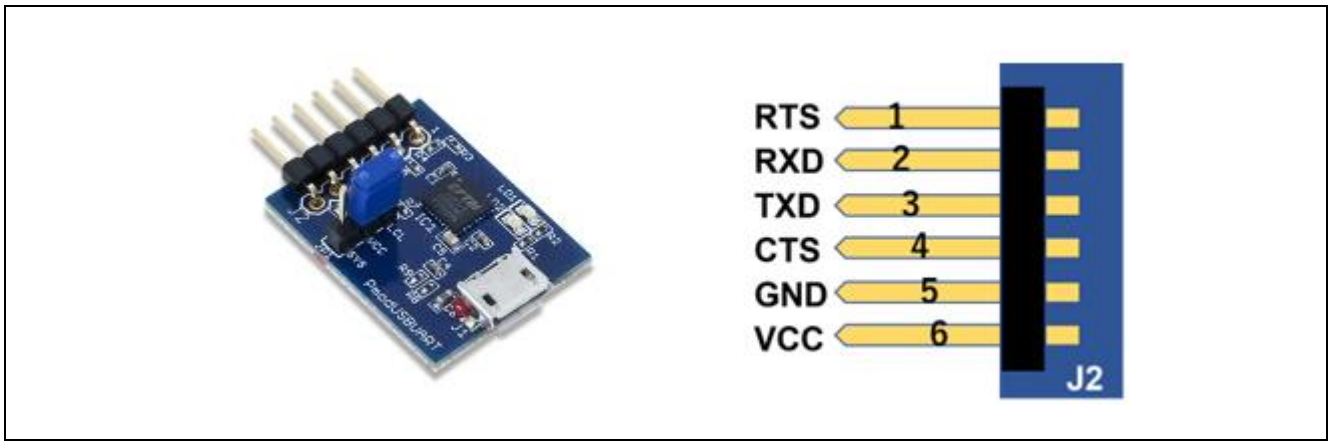

図 3-19　Pmod USBUART モジュール(Digilent 社製)の外観とピン配置

Pmod USBUART モジュールの 1～6pin を CK-RX65N の Pmod1 コネクタの上段 Pin1～6 に接続します。

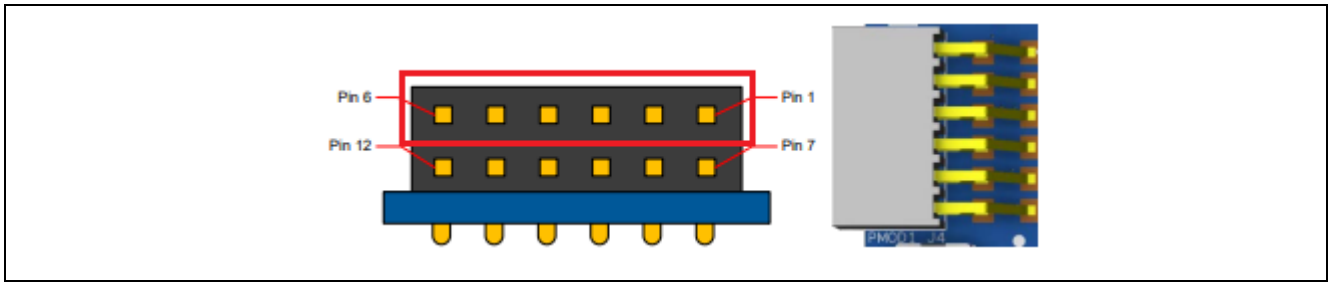

図 3-20　CK-RX65N Pmod1 コネクタ

デバッグ用のハードウェア設定を以下に示します。

表 3-1 ジャンパーの設定

ジャンパー	位置	機能
J15	オープン	E2OB ノーマルデバッグモード
J16	Pins 1-2	デバッガ有効

Pmod の USB コネクタと CK-RX65N のデバッグ用 UBS コネクタをホスト PC に接続します。

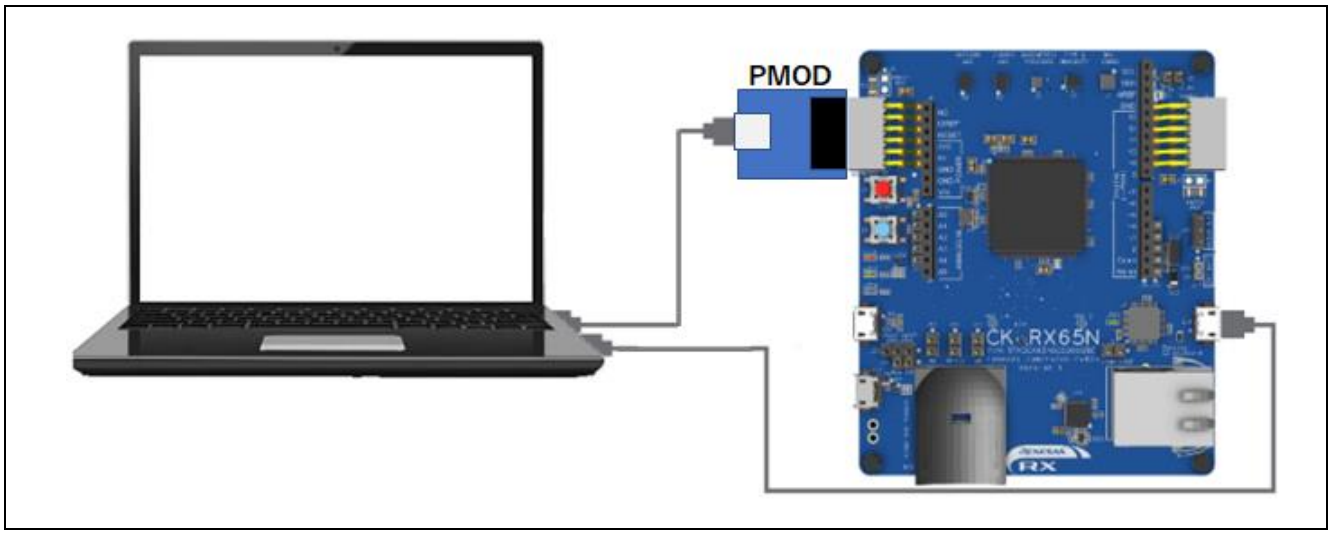

図 3-21　PC と CK-RX65N の接続図

補足：
本システムでは PMOD に接続された RX65N マイコンの SCI の ch6 を占有します。
また、本システムは Tracealyzer のマニュアルの以下「Custom Streaming」を参考に構築されています。
Tracealyzer がマイコン内部で生成し外部に出力するモニタデータの生成レートはこの資料によると 20-200 KB/s となっています。
今回紹介している Pmod USBUART モジュールが対応可能なボーレートは 921600bps(=112.5KB/s)であるため、複雑なタスク構成の場合はモニタデータが不完全になる可能性があります。
この場合、より高速なインタフェース（Ethernet 等）でのモニタデータ出力をご検討ください。
以下 Tracealyzer マニュアルを合わせて参考にしてください。
Percepio Tracealyzer Documentation

3.6　RTOS リソースビューの使用

e² studio には[RTOS リソース]ビュー機能があり、FreeRTOS のリソースの状態を表示できます。ここでは、[RTOS リソース]ビューの使用手順を説明します。

3.6.1　RTOS リソースビューの表示

[RTOS リソース]ビュー機能が使用できるのはデバッガ実行中のみです。デバッガを起動してから[Renesas Views] → [パートナーOS] → [RTOS リソース]の順に選択します。[OS 選択]ダイアログボックスが表示されたら図 3-22 のように"FreeRTOS"を選択してください。図 3-23 のように[RTOS リソース]ビューが開きます。

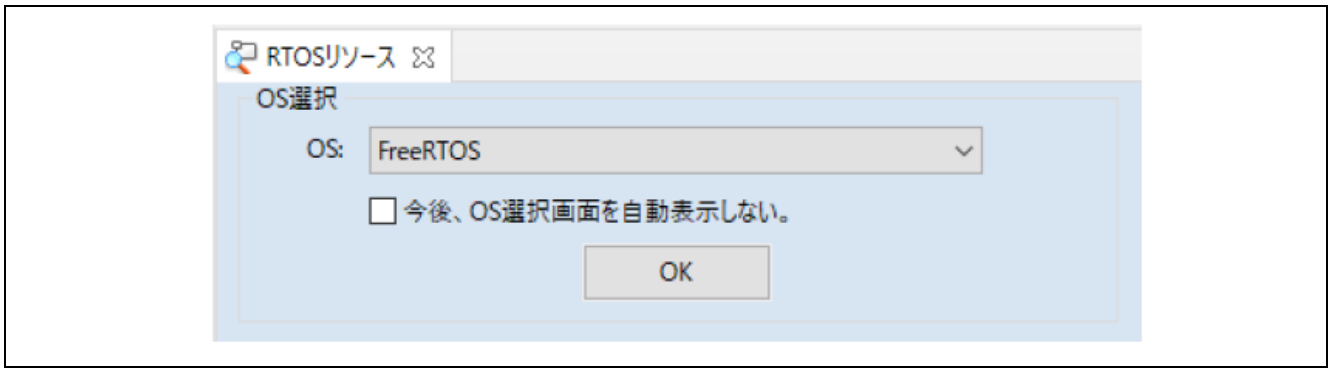

図 3-22　OS の選択

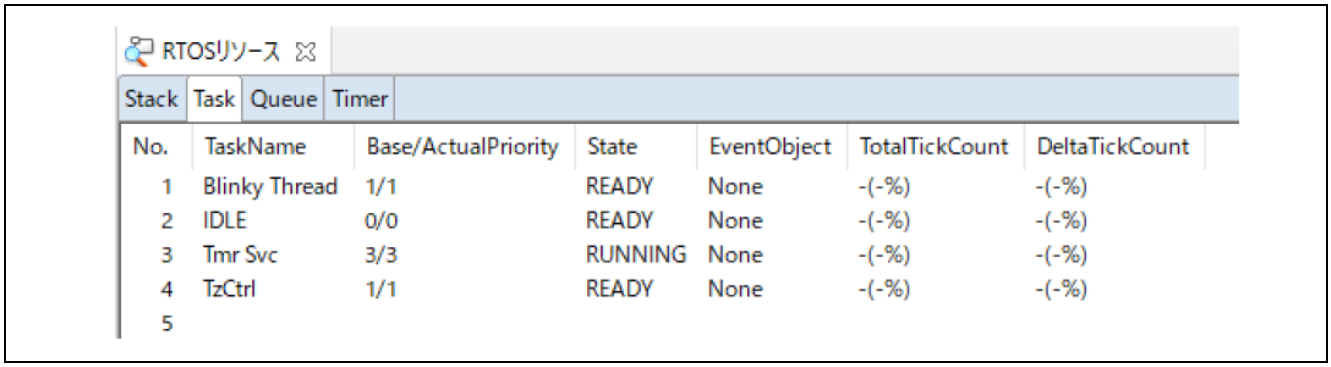

図 3-23　[RTOS リソースビュー]

3.6.2　コンテキストメニュー

[RTOS リソース]ビュー上でマウスの右ボタンをクリックすると、コンテキストメニューが開きます。

図 3-24　コンテキストメニュー

メニューの説明：

- **リアルタイム・リフレッシュカラム**
  表示されている項目の内容をリアルタイムに更新できます。
  このメニューは、プログラム実行中はグレー表示で選択できません。
- **リアルタイム・リフレッシュ間隔**
  リアルタイム表示を更新する間隔を指定します。設定できる間隔は 500 ms から 10000 ms の範囲です。
  このメニューは、プログラム実行中はグレー表示で選択できません。
- **スタック設定**
  スタックデータロードを有効または無効にし、スタック警告のしきい値を設定します。
  このメニューは、プログラム実行中はグレー表示で選択できません。
- **最新の情報へ更新**
  表示を最新の情報に更新します。
- **ソースへジャンプ**
  [エディタ]ビューを開き、タスク/スレッドまたはハンドラのソースコードを表示します。[エディタ]ビューは、タスク/スレッドまたはハンドラをダブルクリックしても開くことができます。
  このメニューは、プログラム実行中はグレー表示で選択できません。
- **ファイルへ保存**
  現在選択されているタブのデータをテキストファイル（*.txt）に保存します。
  このメニューは、プログラム実行中はグレー表示で選択できません。
- **OS の選択**
  [OS 選択]ダイアログボックスを開きます。
  このメニューは、プログラム実行中はグレー表示で選択できません。

3.6.3　スタック設定

スタックデータロードを有効にし、スタックしきい値を設定します。

1. コンテキストメニューを開き、[スタック設定]を選択します。
2. スタックデータを[RTOS リソース]ビューにロードするには、[スタック設定]ダイアログボックスの"スタックデータロードを有効にする"のチェックボックスを選択します。これを有効にしていないと、次回のデバッグセッションでスタックデータがロードされません。

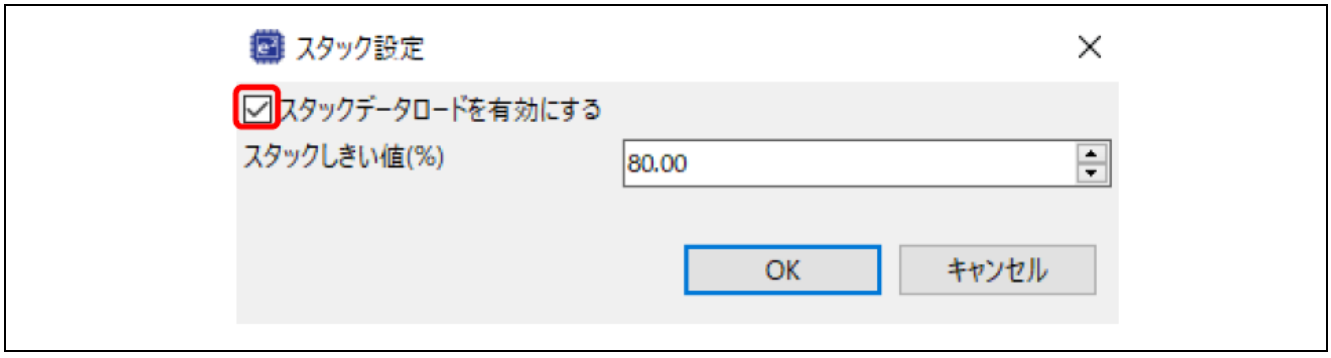

図 3-25　スタックデータロードの有効化

3. 任意のスタックしきい値を[スタックしきい値(%)]テキストボックスに設定できます。[OK]で設定を保存します。

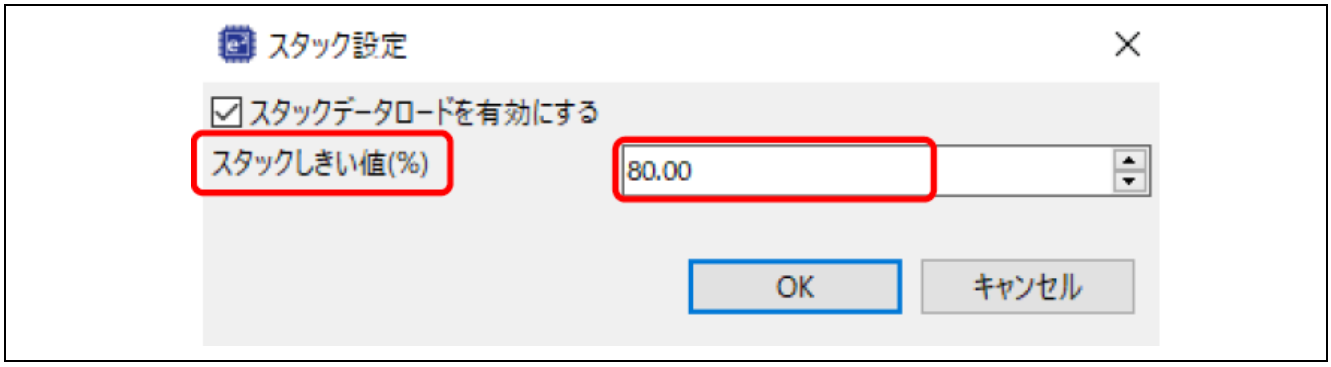

図 3-26　スタックしきい値の設定

4. ターゲットプロジェクトを実行してから中断し、スタックデータをロードします。設定したスタックしきい値に達すると、しきい値の警告が表示されます。

5. 警告表示は 2 種類あります。1 つは[Stack Threshold Warning]で、設定したスタックしきい値に達したスレッドの一覧を表示します。もう 1 つは[Stack Overflow Warning]で、スタック使用サイズが 100%に達したスレッドの一覧を表示します。

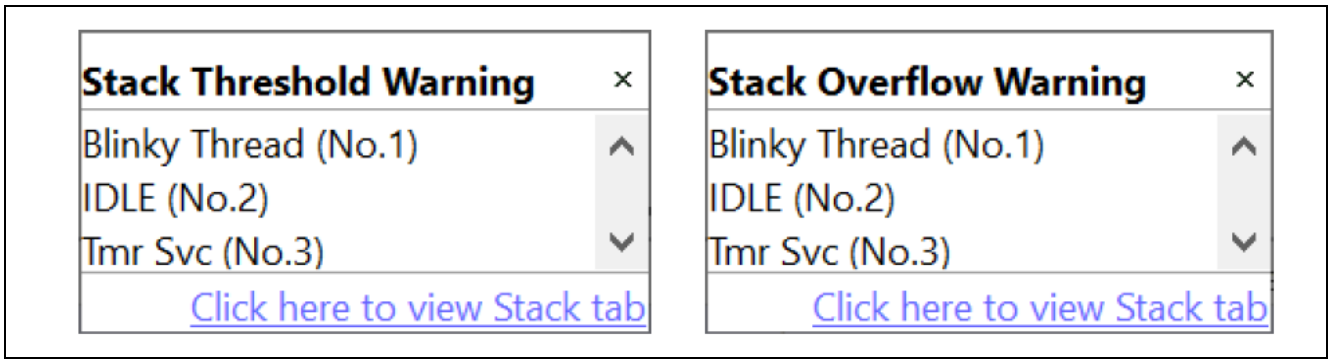

図 3-27　[Stack Threshold Warning]表示（左）と[Stack Overflow Warning]表示（右）

3.6.4　タブメニュー

各タブに表示する項目を表 3-2 に示します。

表 3-2 各タブに表示する内容

[RTOS リソース] ビューのタブの名称	表示する情報と選択項目の名称	表示内容
Stack	No.	行番号
	TaskName	タスク作成時に割り当てられた名称
	StartOfStack	スタックの開始アドレス
	EndOfStack	スタックの終了アドレス
	TopOfStack	スタック内容保存時に最後に書き込まれたスタック先頭アドレス
	StackSize(bytes)	総スタックサイズ
	StackUsageSize	スタックの最大使用サイズ
	StackUsageRatio	総スタックサイズに対する最大使用サイズの比率（%）
Task	No.	行番号
	TaskName	タスク作成時に割り当てられた名称
	Base/ActualPriority	優先度継承メカニズムで使用するベース優先度/タスクが使用する実際の優先度
	State	タスク状態：RUNNING、READY、BLOCKED、SUSPENDED のいずれか
	EventObject	タスクのブロック要因となったキューの名称
	TotalTickCount	タスクがアクティブになるまでの総ティック数
	DeltaTickCount	前回の中断イベントからタスクがアクティブになるまでのティック数
Queue	No.	行番号
	Name (Type)	登録時にキューに割り当てられた名称とそのタイプ（Queue、Semaphore、Mutex）
	Address	キューハンドルのアドレス
	MaxLength	キューに保存できるアイテムの最大数
	ItemSize	キュー内のアイテムごとのサイズ (byte)
	CurrentLength	キューに現在保存されているアイテムの数
	#WaitingTx	キューへの送信を待機中にブロックされたタスクの数
	#WaitingRx	キューからの受信を待機中にブロックされたタスクの数
Timer	No.	行番号
	Name	現在のタイマ時間（システムティック数）
	Period	オートリロードの有効/無効。 On：オートリロード有効。タイマ時間が満了するたびにタイマがリセットされます。 Off：オートリロード無効。タイマ時間が満了しても何も行ないません。
	CallbackFn	タイマが終了するたびに実行されるコールバック関数のアドレスと関数名
	TimerID	タイマ作成時に割り当てられた ID 番号（16 進数）

3.7　Tracealyzer®を使用したプロジェクトデバッグの開始

3.7.1　e² studio でのデバッガの起動

メニューから[実行] → [デバッグの構成] → [Debugger]タブ → [Connection Setting]タブで、[エミュレータから電源を供給する]の選択を「いいえ」に変更します。

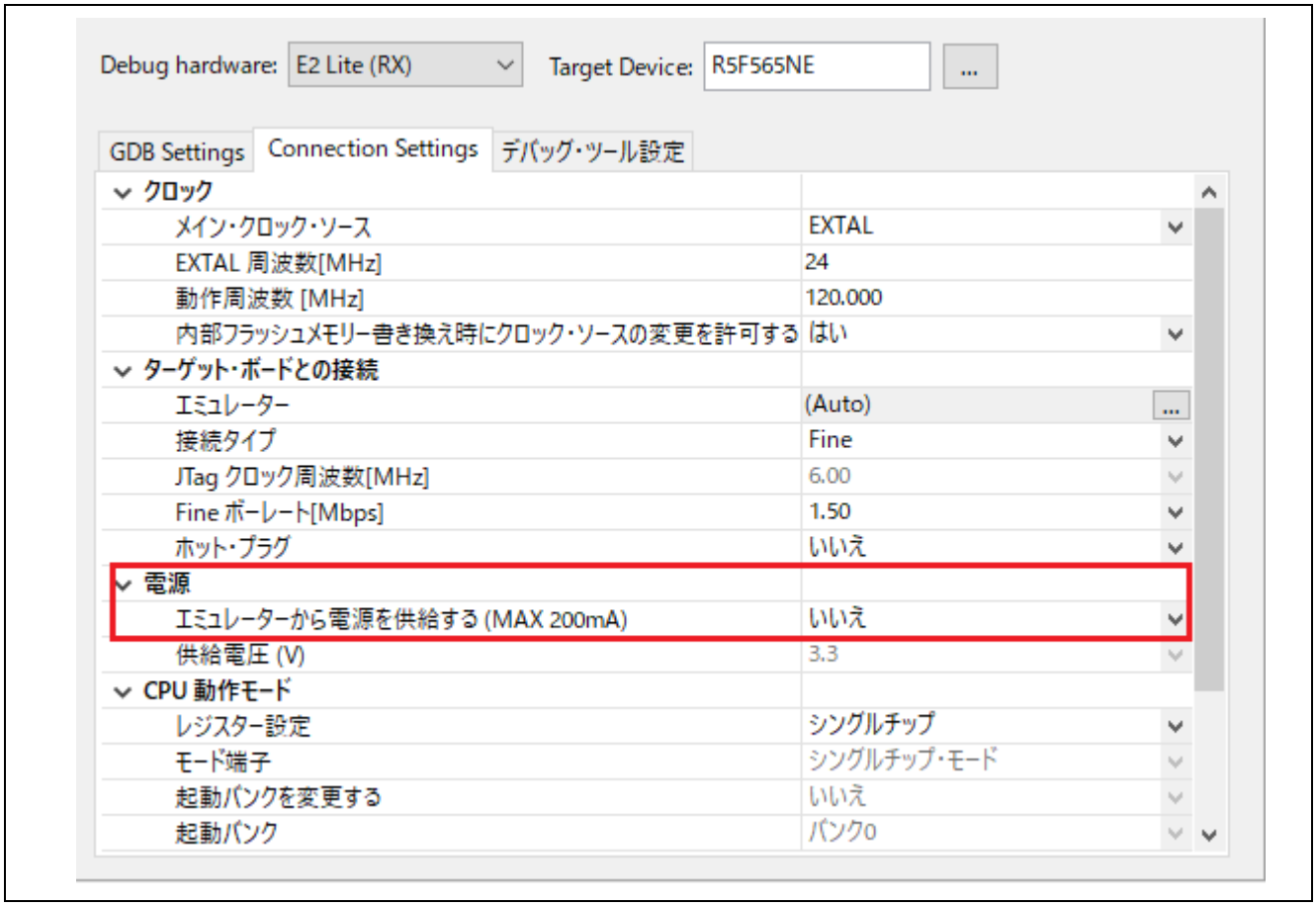

図 3-28　接続設定の変更

メニューから[実行] → [デバッグ]の順に選択するとデバッガが起動します。

3.7.2　Tracealyzer®の起動

Tracealyzer®を起動します。
[Recording Settings]をクリックし、[PSF Streaming Settings]を選択して、以下のように設定します。

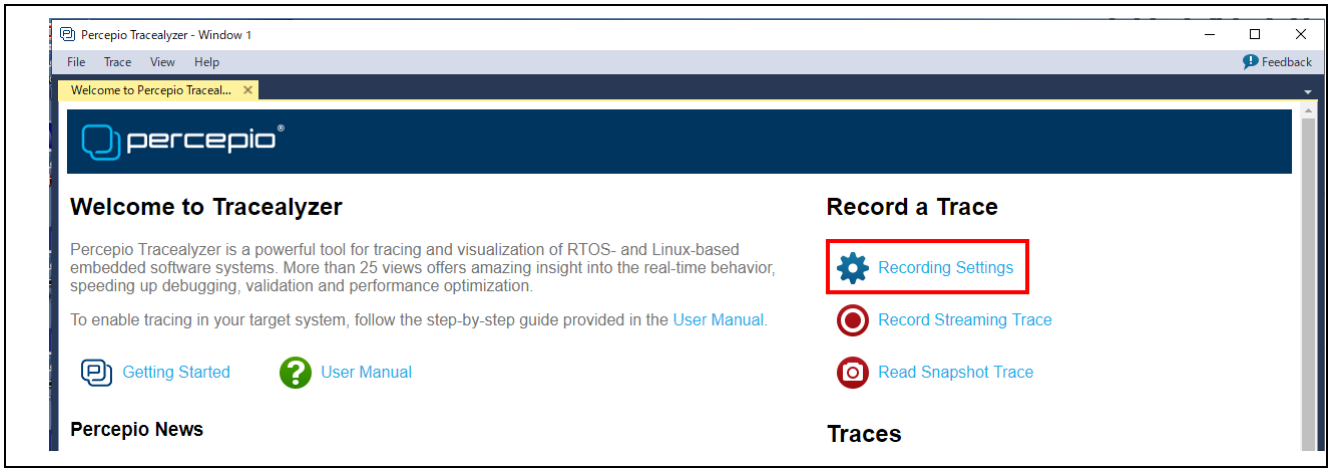

図 3-29　Recording Settings

- Device：（ユーザ PC システムポート）
- Data bits：**8**
- Data rate：**921600**
- Handshake：**None**
- Parity：**None**
- Stop bits：**One**

COM ポートの番号は、CK-RX65N の Pmod に接続された USB シリアル変換チップと対応するものを設定します。

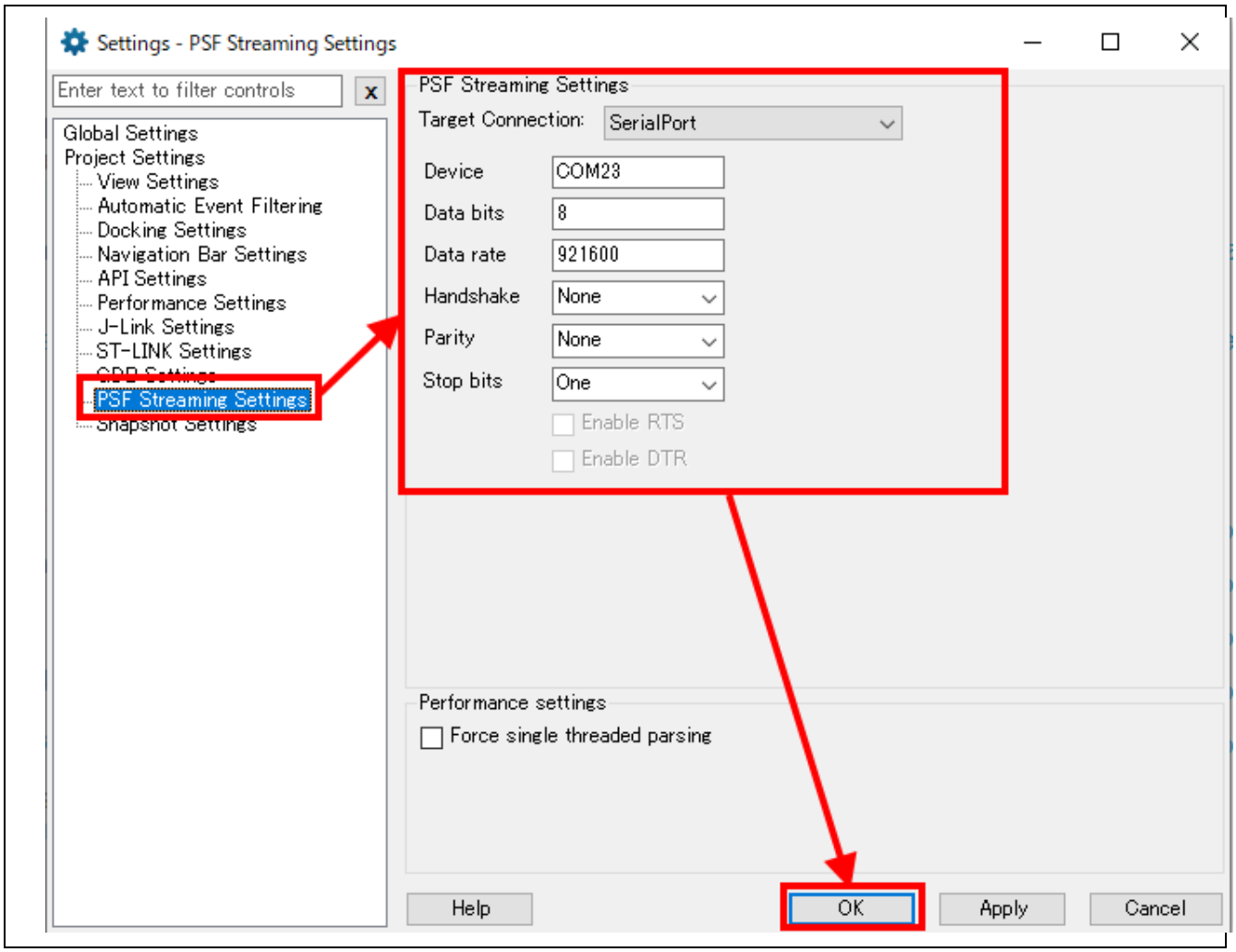

図 3-30　PSF Streaming Settings の設定内容

次に[Record Streaming Trace]を選択します。

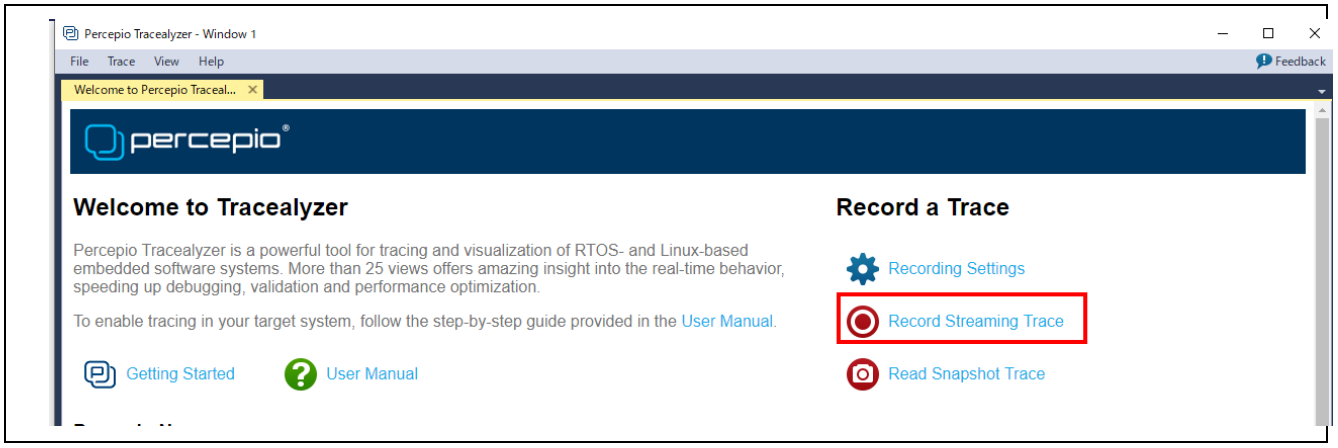

図 3-31 Record Streaming Trace

[Reconnect]から[Start Session] を選択すると Tracealyzer®側が待ち状態になります。

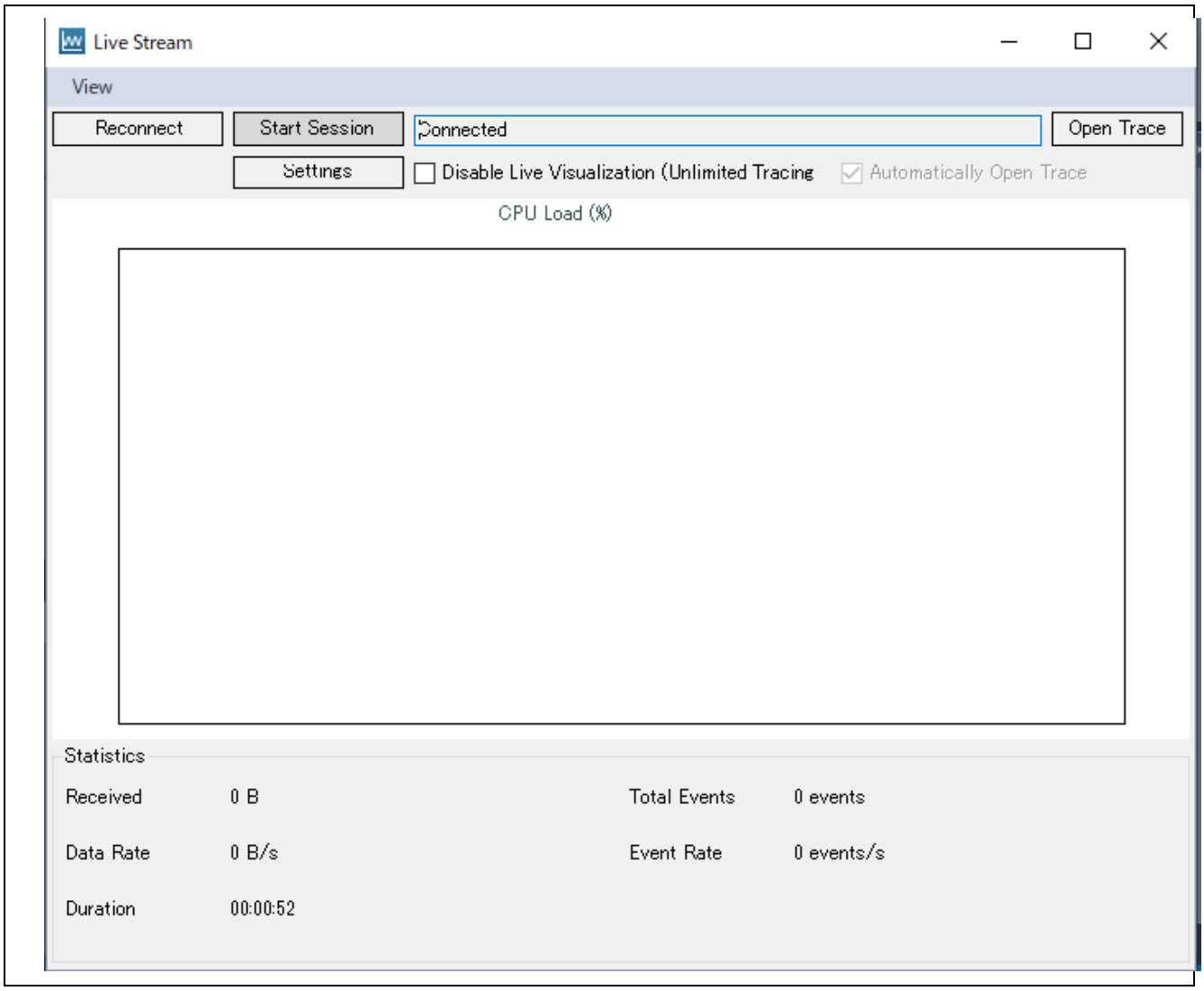

図 3-32　Tracealyzer®を待ち状態にする

3.7.3　ソフトウェアの実行

e² studio のメニューから[実行] → [再開]でソフトウェアを動作状態にすると、CK-RX65N と PC(Tracealyzer)の通信が始まり、FreeRTOS の内部状態が Tracealyzer®上に表示されます。

3.7.4　トレース情報の表示

各種の分析モードが使用できます。詳細は、[Help]を参照してください。

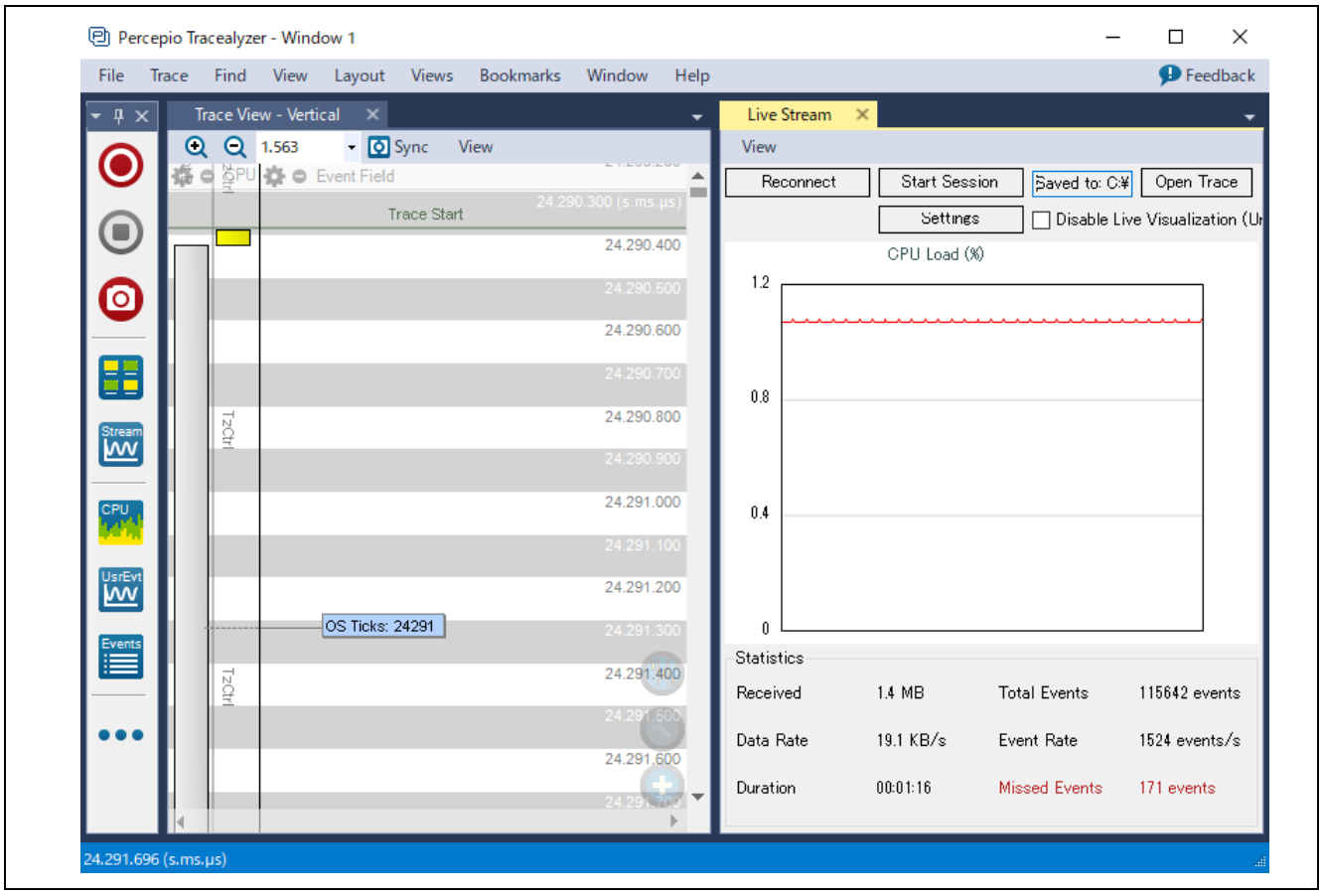

図 3-33　トレース情報の表示

ホームページとサポート窓口

RX ファミリの主な機能、ダウンロードコンポーネント、関連ドキュメント、サポートについては、下記のサイトにアクセスしてください。

RX ファミリ製品情報	www.renesas.com/rx
RX ファミリサポートフォーラム	www.renesas.com/rx/forum
ルネサスサポートサイト	www.renesas.com/support

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Mar.25.23	-	初版発行
1.01	May.25.23	5,6,7	trcStreamPort.c のコードの修正

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。