ルネサス RA 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®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。

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

RA6M3 MCU グループ (R7FA6M3AH)

動作環境（UART 使用時）

ターゲットボード	EK-RA6M3
統合開発環境（IDE）	e² studio バージョン 2021-04 および FSP v3.0.0
トレースツール	Percepio Tracealyzer® v4.4.2
OS	FreeRTOS 10.4.3
ツールチェーン	GNU Arm Embedded Toolchain: 9-2020-q2-major (GNU ARM Embedded 9.3.1.20200408)
ケーブル	FTDI TTL-232R-3V3（USB-TTL シリアルケーブル）

動作環境（J-Link RTT 使用時）

ターゲットボード	EK-RA6M3
統合開発環境（IDE）	e² studio バージョン 2021-04 および FSP v3.0.0
トレースツール	Percepio Tracealyzer® v4.4.2
OS	FreeRTOS 10.4.3
ツールチェーン	GNU Arm Embedded Toolchain: 9-2020-q2-major (GNU ARM Embedded 9.3.1.20200408)

【注】 あらかじめ、以下の URL からツールをダウンロードしてインストールしてください。

- RA 向け e² studio クイックスタートガイドのダウンロードサイト：
  Quick Start Guide for e² studio for RA
- プラットフォームインストーラ（FSP with e² studio installer）のダウンロードサイト：
  https://github.com/renesas/fsp/releases
- EK-RA6M3 Example Project Bundle - Sample Code のダウンロードサイト：
  EK-RA6M3 Example Project Bundle - Sample Code
- 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.　プラットフォームインストーラ（FSP with e² studio）によるインストール

Renesas e² studio 2021-04 or higher User's Manual: Quick Start Guide の「2.1　Installing the FSP with e² studio Installer」を参照して FSP をインストールしてください。

2.　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。

3.　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、プロジェクト名、関連づけるデバイスとボード、ドライバを指定して RA プロジェクトを生成することができます。

e² studio を起動して、起動画面でワークスペースフォルダを選択してください。新規 RA プロジェクトは、以下の手順で作成します。

1. [ファイル]メニュー → [新規] → [Renesas C/C++ Project] → [Renesas RA]の順に選択します。
2. "Renesas RA C/C++ Project"テンプレートを選択します。[次へ]で次に進みます。

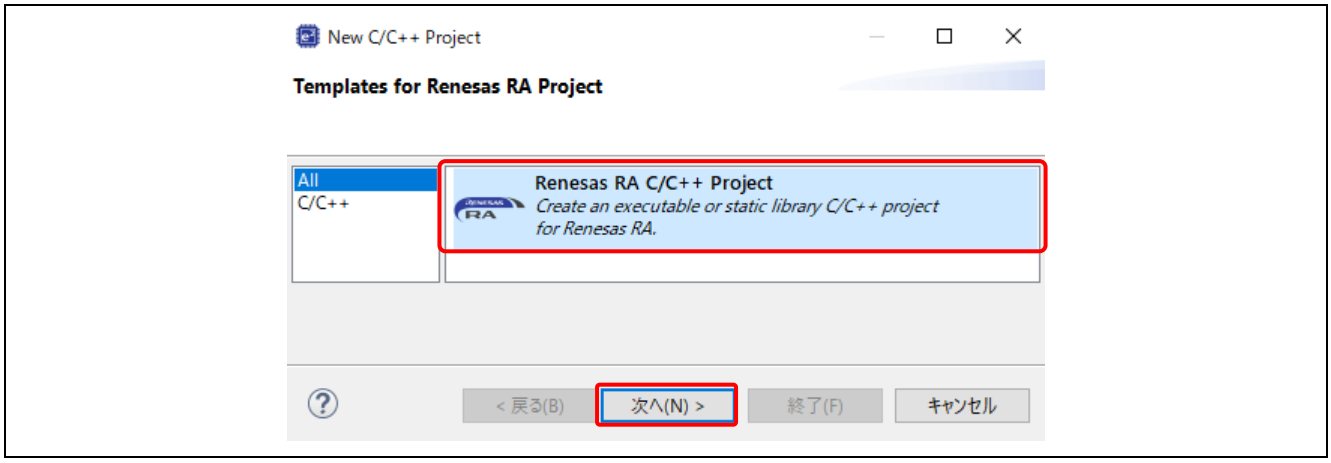


図 1　テンプレートの選択

3. 表示されたダイアログボックスでプロジェクト名を入力し、[次へ]をクリックします。

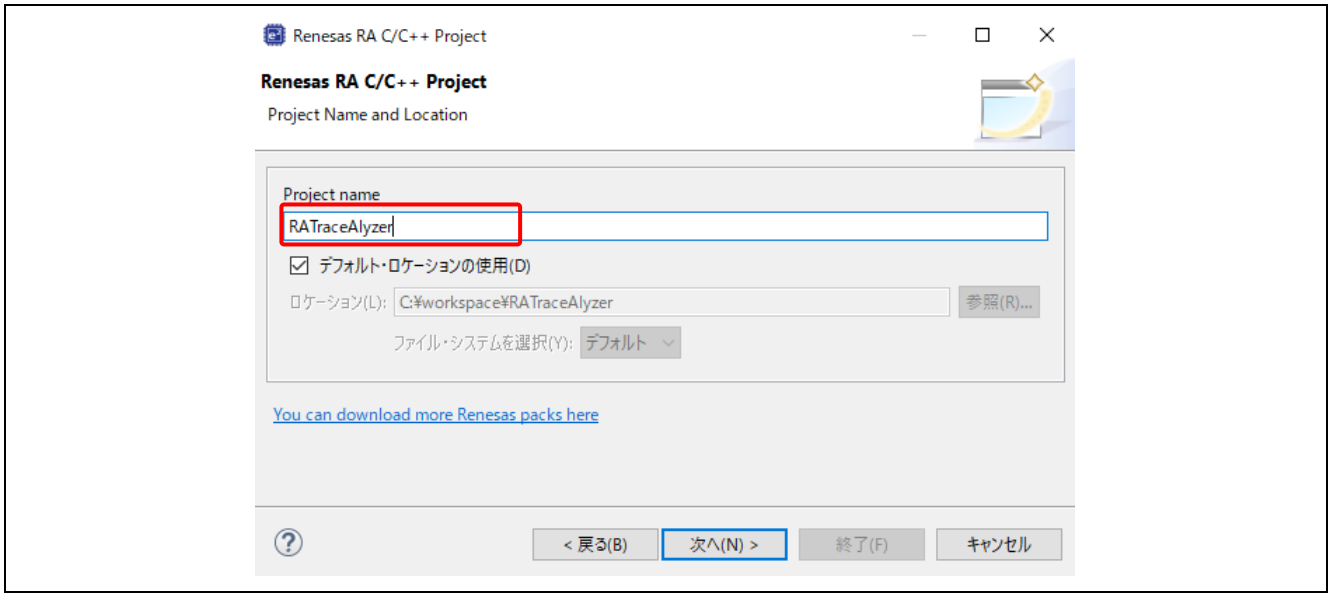


図 2　プロジェクト名と作成場所

4. デバイス選択の画面で、デバイスとツールの情報を次のように入力します。
   - FSP Version：**3.0.0**
   - Board：**EK-RA6M3**
   - Device：自動的に選択されます。
   - Language：**C**
   - Toolchains：ルネサス RA ファミリ向けに認定された最新の GNU Arm Embedded Toolchain を選択します。（例：GCC ARM Embedded 9.3.1.20200408）
   - Debugger：**J-Link ARM**
   - [次へ]で次に進みます。

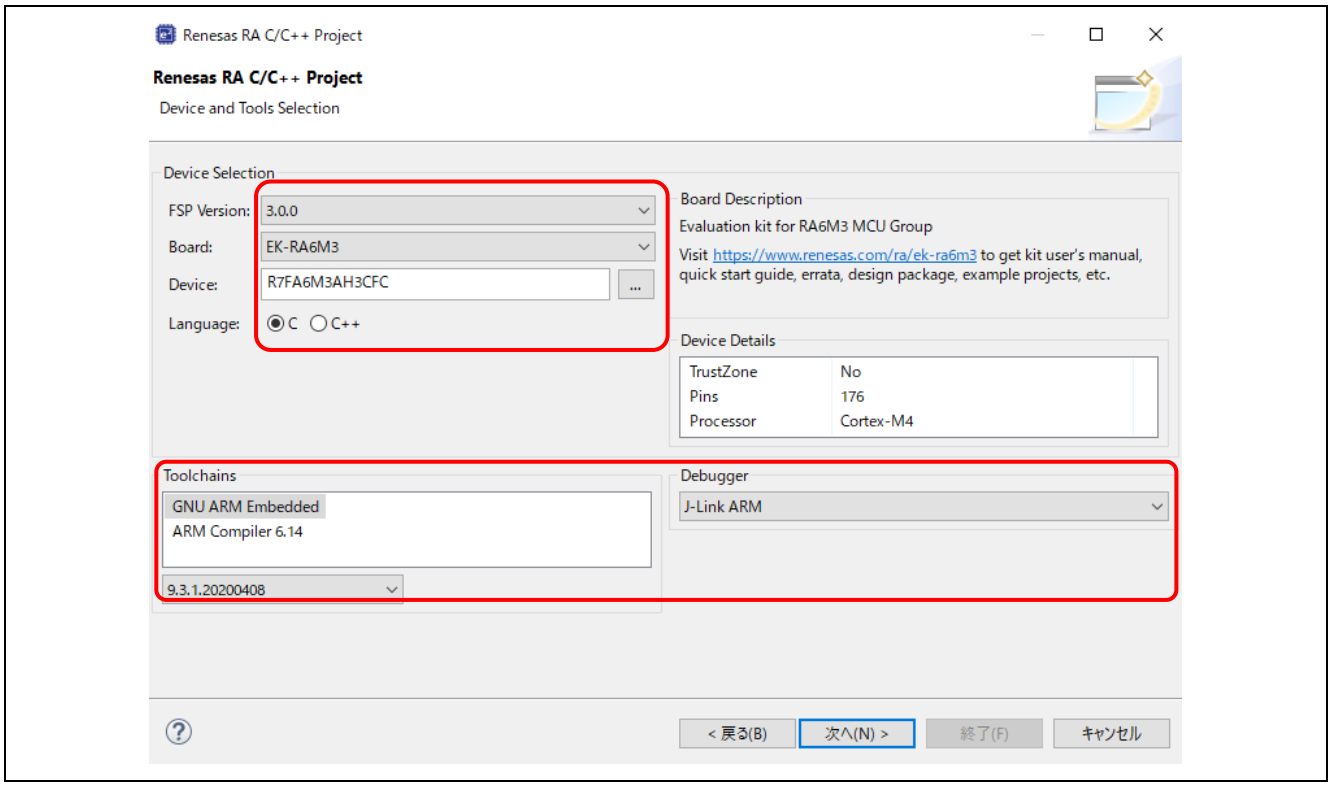


図 3　EK-RA6M3 用の新規プロジェクトの作成

5. Build Artifact Selection：**Executable** を選択します。
   RTOS Selection：**FreeRTOS** を選択します。

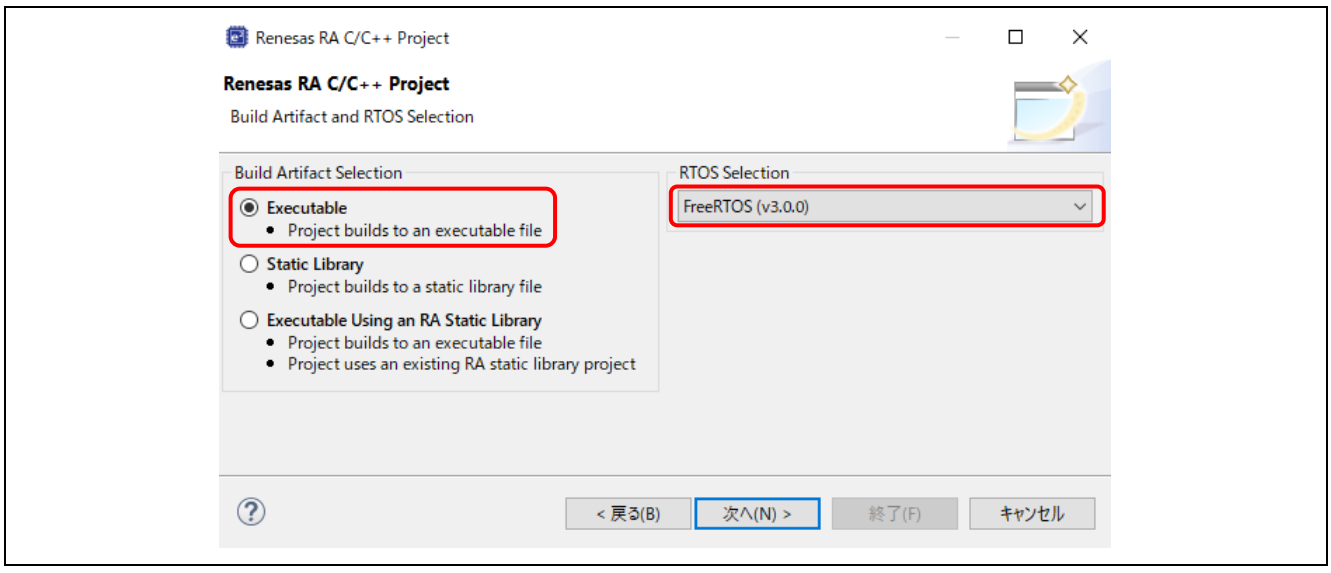


図 4　Build Artifact と RTOS の選択

6. プロジェクトテンプレート選択の画面で、"FreeRTOS - Blinky - Static Allocation"を選択し、[終了]をクリックします。

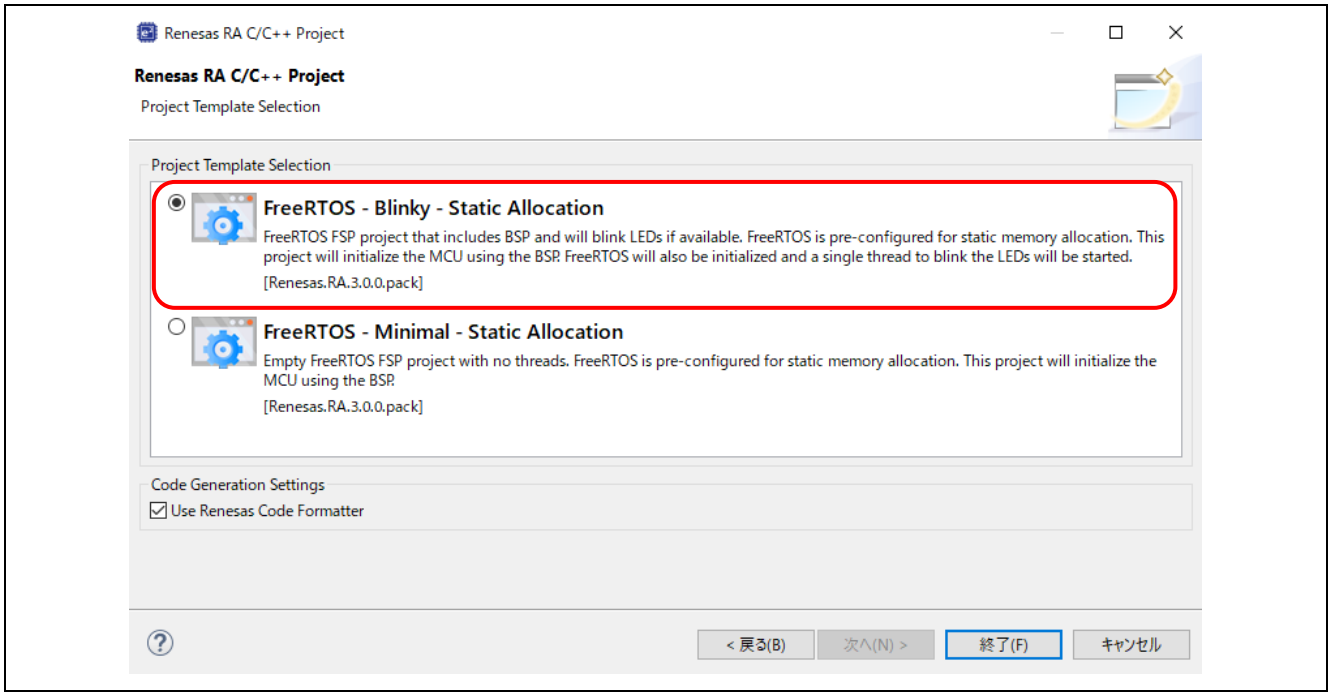


図 5　プロジェクトテンプレートの選択

7. ここまでの手順が完了すると、e² studio は新規プロジェクトを作成します。[FSP Configuration]パースペクティブが開き、プロジェクトの設定が可能となります。

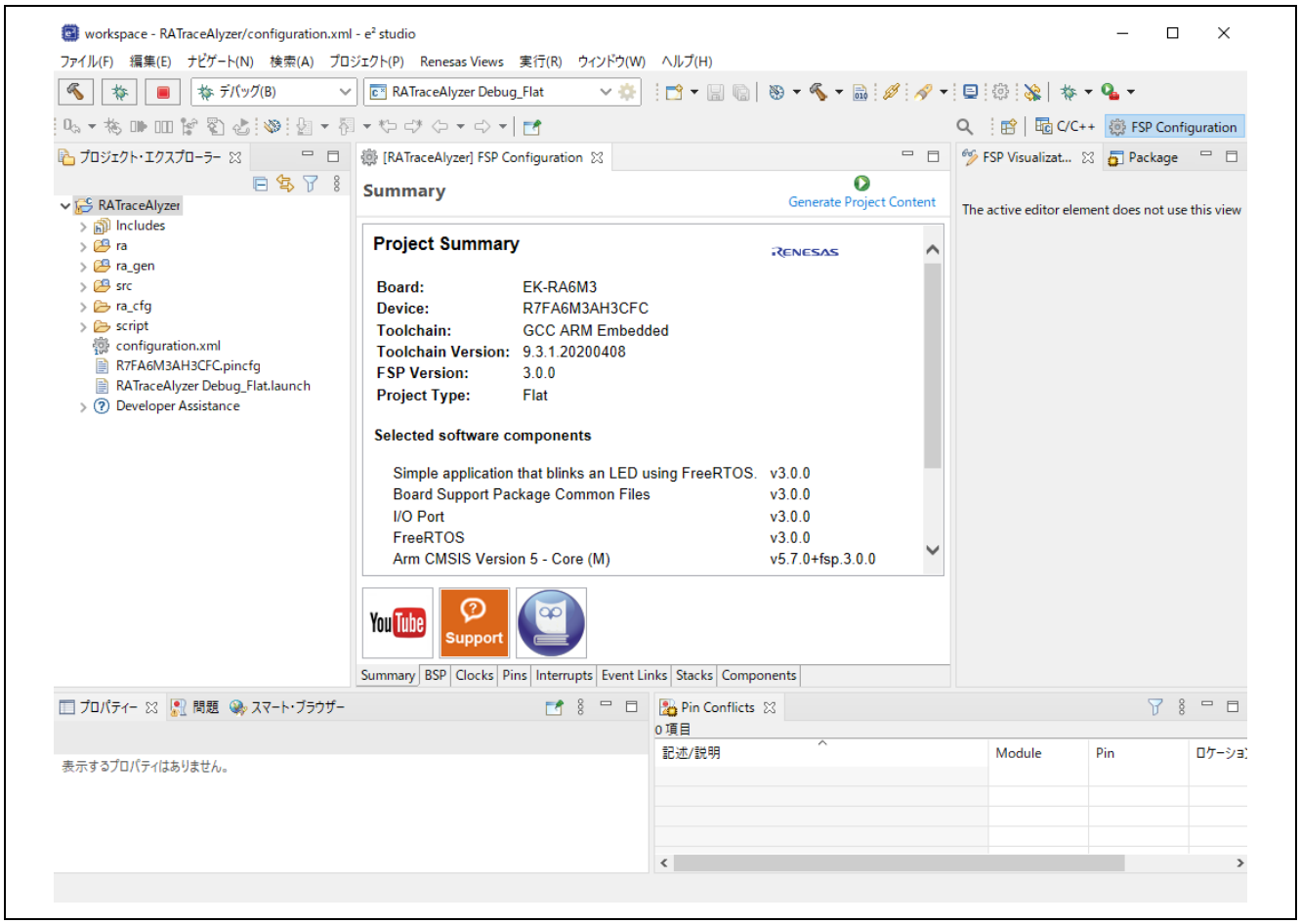


図 6　EK-RA6M3 用の新規プロジェクト

4.　Tracealyzer®を用いたデバッグ（UART 使用時）

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

4.1　プロジェクトへの Tracealyzer® for FreeRTOS のコピーと不要フォルダの削除

4.1.1　Tracealyzer®インストールフォルダへの 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。

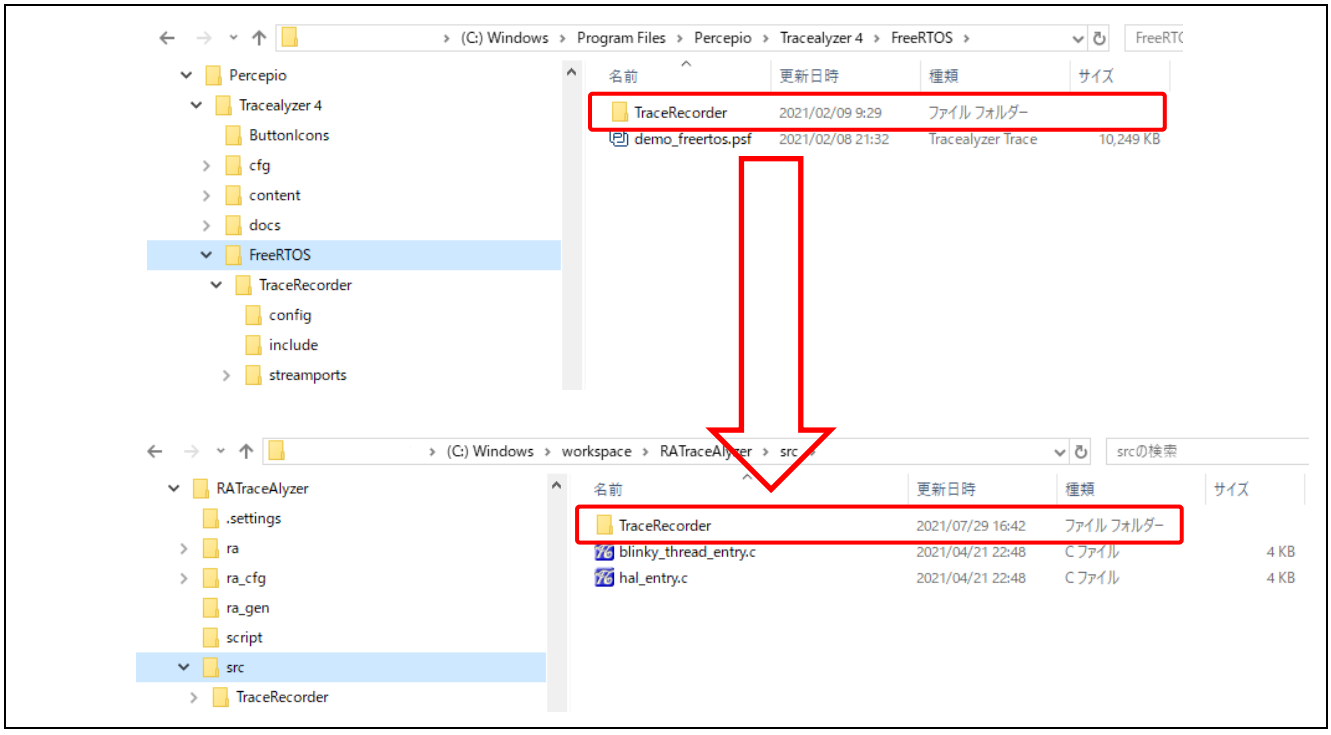


図 7　フォルダのコピー

4.1.2　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。

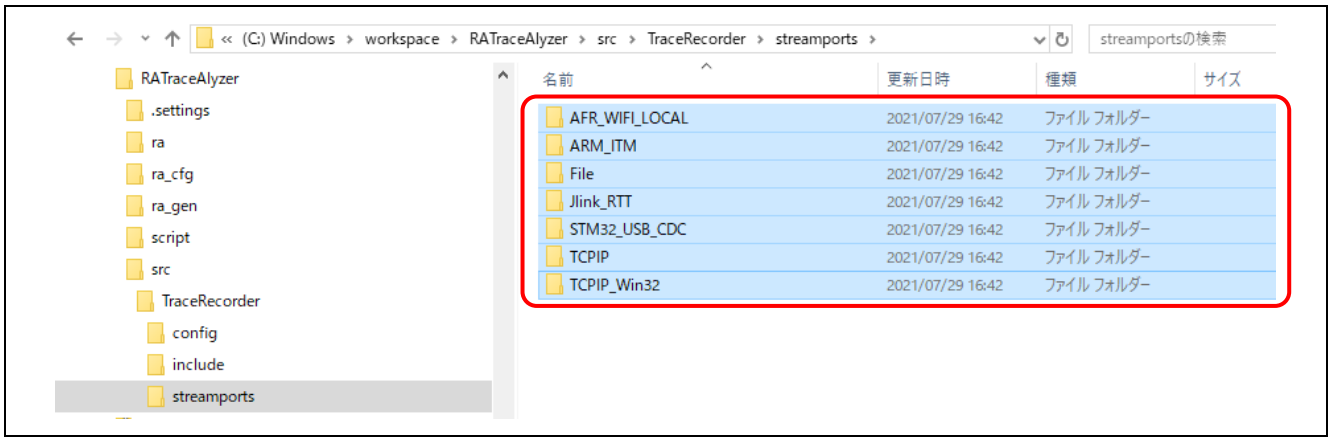
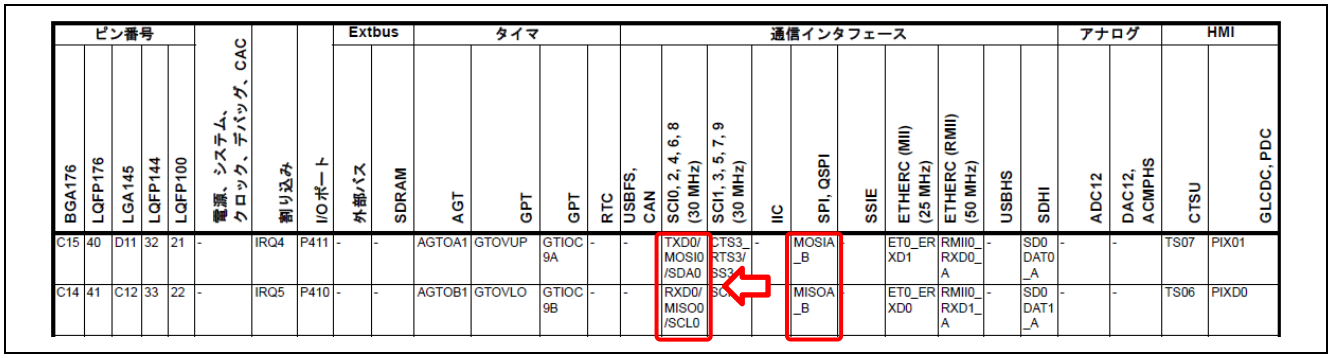


図 8　フォルダの削除（UART）

4.2　FSP の構成

4.2.1　UART ドライバ設定

UART 通信に P410 と P411 端子（SCI0）を使用します。EK-RA6M3 のデフォルト設定では SPI0 が選択されています。



ピン番号								Extbus		タイマ				通信インタフェース									アナログ		HMI	
BGA176	LQFP176	LGA145	LQFP144	LQFP100	電源、システム、クロック、デバッグ、CAC	割り込み	I/O ポート	外部バス	SDRAM	AGT	GPT	GPT	RTC	USBFS, CAN	SCI0, 2, 4, 6, 8 (30 MHz)	SCI1, 3, 5, 7, 9 (30 MHz)	IIC	SPI, QSPI	SSIE	ETHERC (MII) (25 MHz)	ETHERC (RMII) (50 MHz)	USBHS	SDHI	ADC12	DAC12, ACMPHS	CTSU	GLCDC, PDC
C15	40	D11	32	21	-	IRQ4	P411	-	-	AGTOA1	GTOVUP	GTIOC 9A	-	-	TXD0/ MOSI0 /SDA0	CTS3_ RTS3/ SS3	-	MOSIA _B		ET0_ER XD1	RMII0_ RXD0_ A	-	SD0 DAT0 _A	-	-	TS07	PIX01
C14	41	C12	33	22	-	IRQ5	P410	-	-	AGTOB1	GTOVLO	GTIOC 9B	-	-	RXD0/ MISO0 /SCL0	SC		MISOA _B		ET0_ER XD0	RMII0_ RXD1_ A	-	SD0 DAT1 _A	-	-	TS06	PIXD0

図 9　EK-RA6M3 の端子設定

SCI0（UART ドライバ）を使用するために、P410 と P411 端子に割り当てられている SPI0 を無効にします。

- [Pins]タブを選択します。
- [Peripherals] → [Connectivity:SPI] → [SPI0]の順に選択します。
- Operation Mode: "Disabled"に設定します。

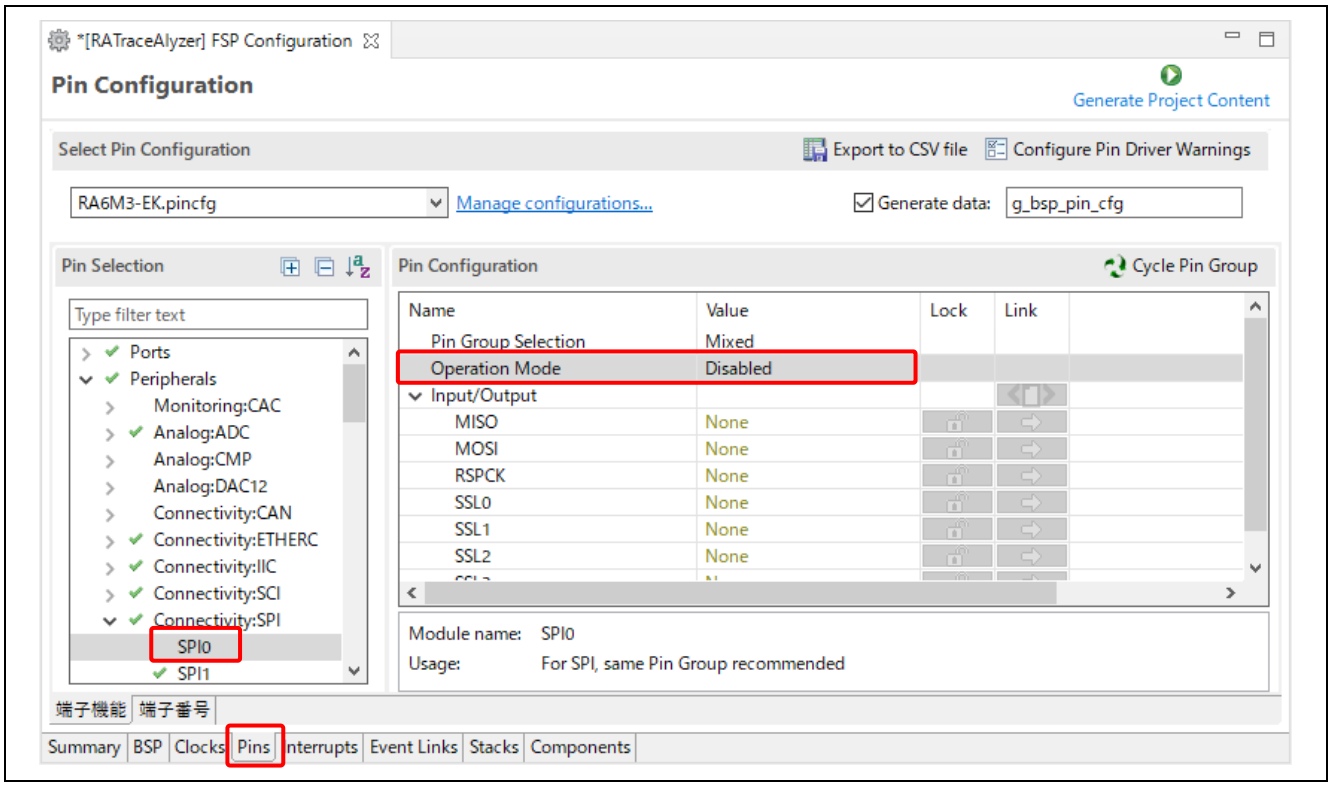


図 10　[Pin Configuration]で SPI0 を無効化

[Stacks]タブを選択します。[New Stack] → [Driver] → [Connectivity] → [UART Driver on r_sci_uart]の順に選択して UART ドライバを追加します。

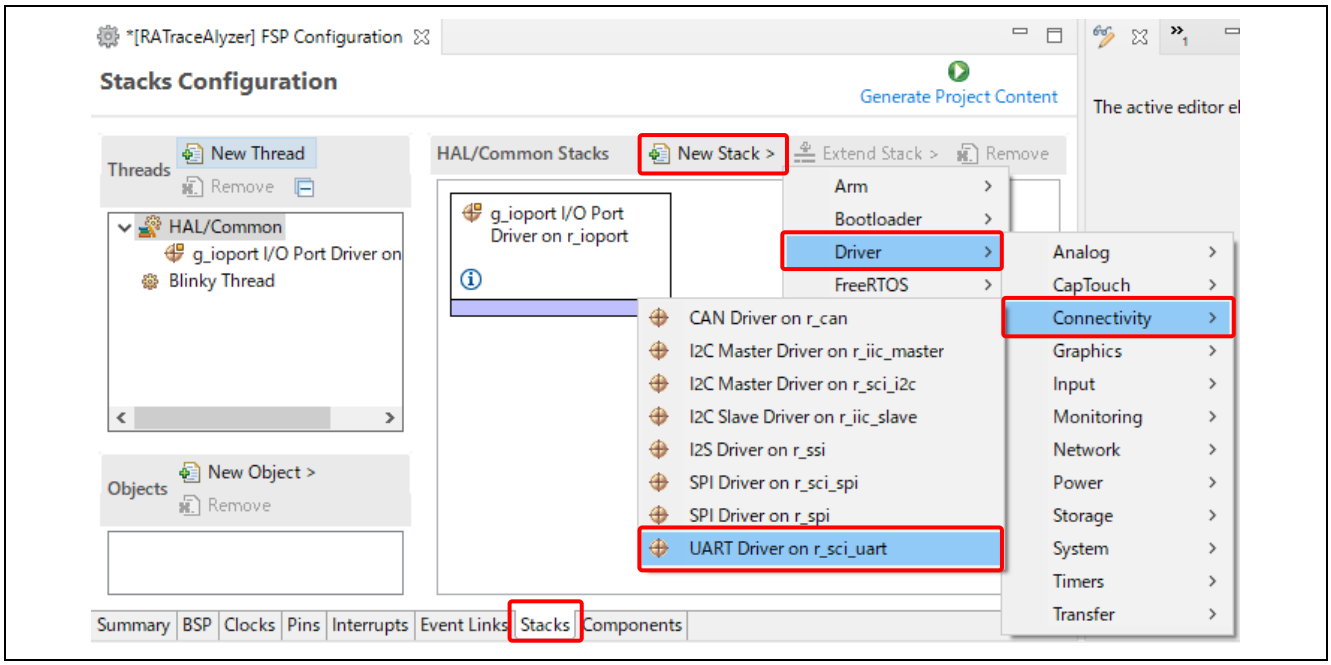


図 11　[Stacks Configuration]で UART ドライバを追加

UART ドライバのプロパティを次のように変更します。

- Baud Rate：**921600**
- Callback：**sci_callback_tracealyzer**
- TXD_MOSI：**P411**
- RXD_MISO：**P410**

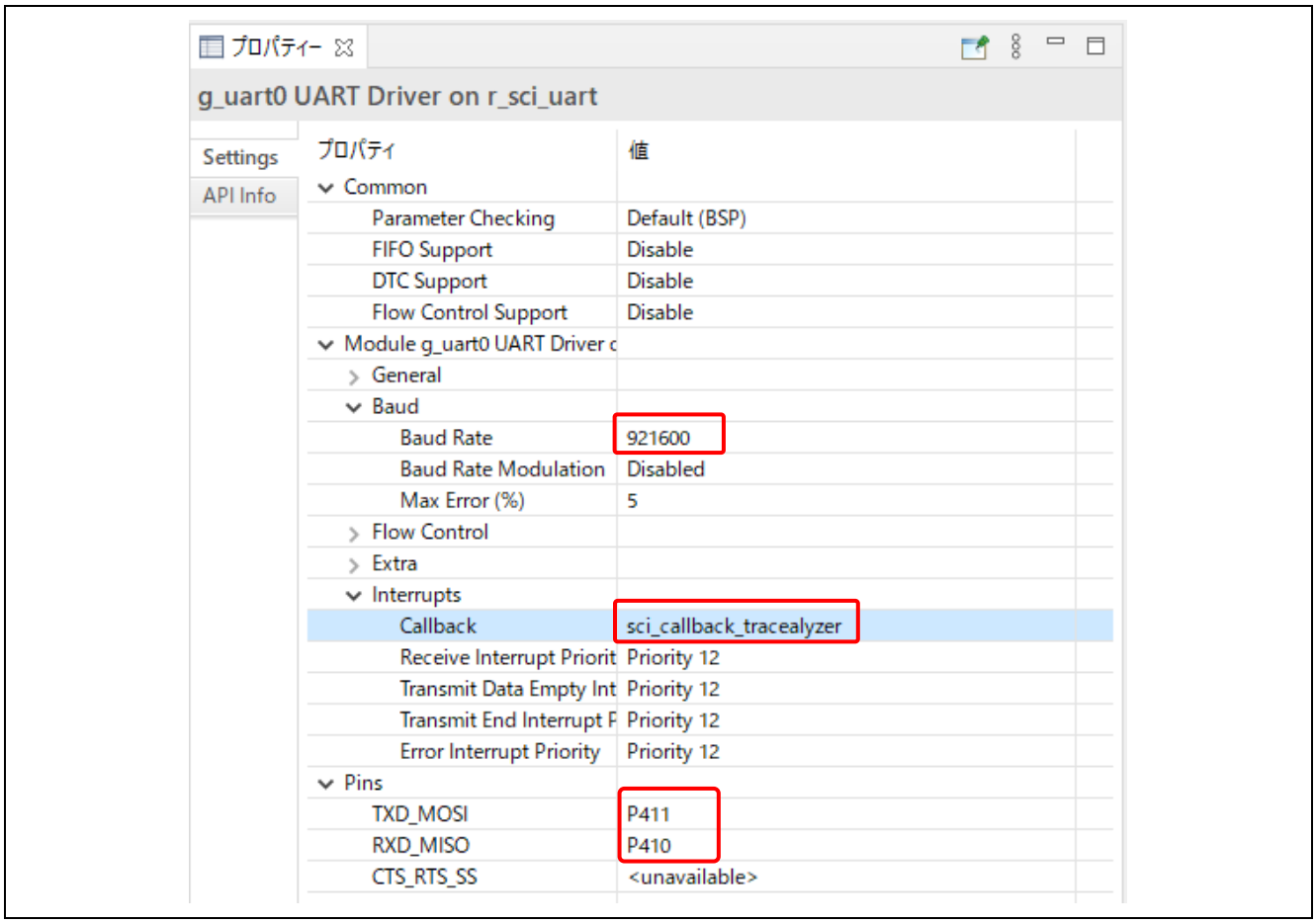


図 12　UART プロパティ

4.2.2　Blinky スレッドの設定

Blinky スレッドに次のようにヒープを追加します。

- [Stacks]タブを開きます。
- [Blinky Thread]を選択します。
- [New Stack] → [FreeRTOS] → [Memory Management] → [Heap 1]の順に選択します。

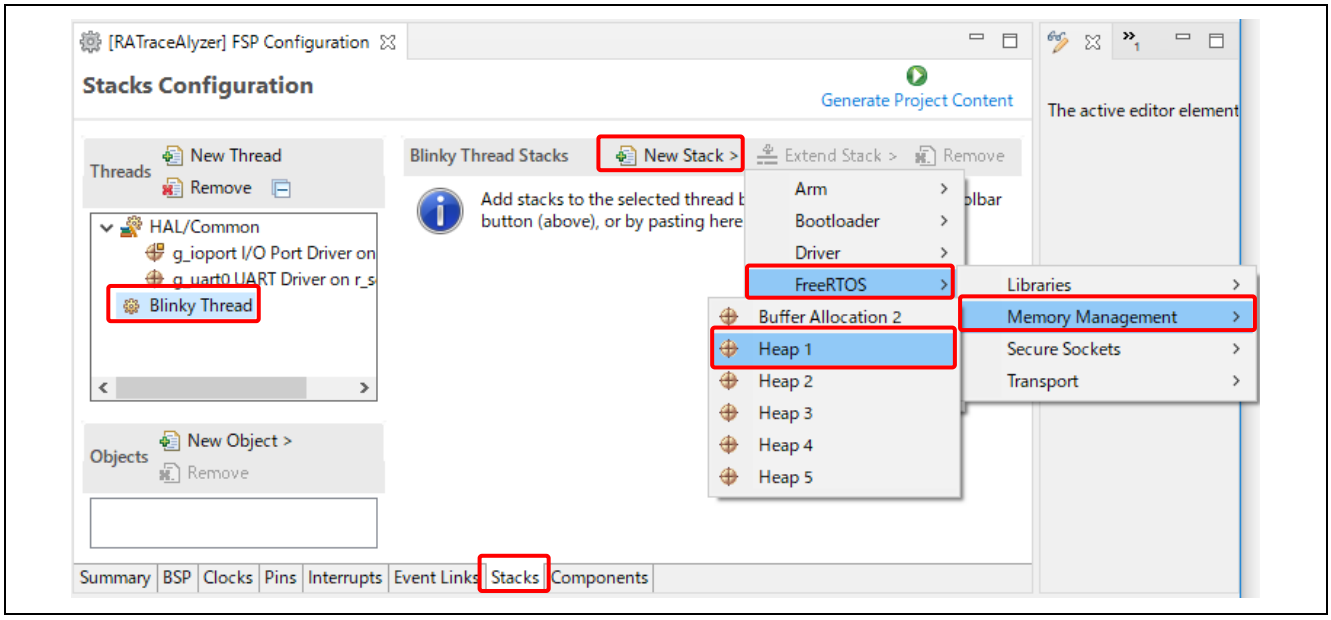


図 13　[Stacks Configuration]でヒープを追加

[Blinky Thread]で、[プロパティ] → [General]の設定を次のように変更します。

- Minimal Stack Size：**512**
- Use Mutexes：**Enabled**
- Use Recursive Mutexes：**Enabled**
- Use Queue Sets：**Enabled**
- Enable Backward Compatibility：**Enabled**

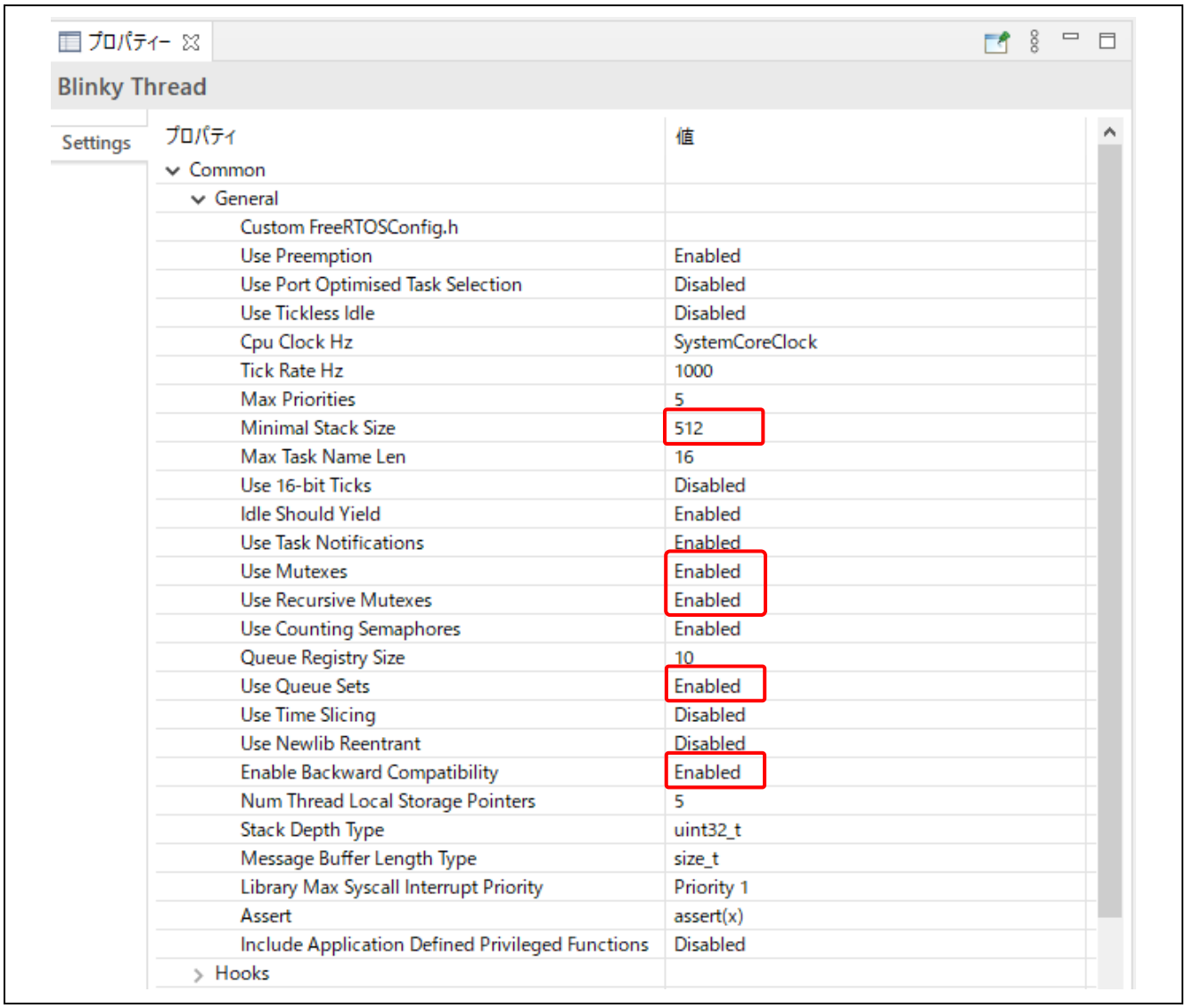


図 14　Blinky スレッドのプロパティ 1

[Blinky Thread]で、[プロパティ] → [Hooks]、[Stats]、[Memory Allocation]、[Timers]の設定を次のように変更します。

- Use Idle Hook：**Disabled**
- Use Malloc Failed Hook：**Enabled**
- Use Trace Facility：**Enabled**
- Use Stats Formatting Functions：**Enabled**
- Support Dynamic Allocation：**Enabled**
- Total Heap Size：**262,144** (256 * 1,024)
- Timer Task Static Depth：**3,072** (1024 * 3)

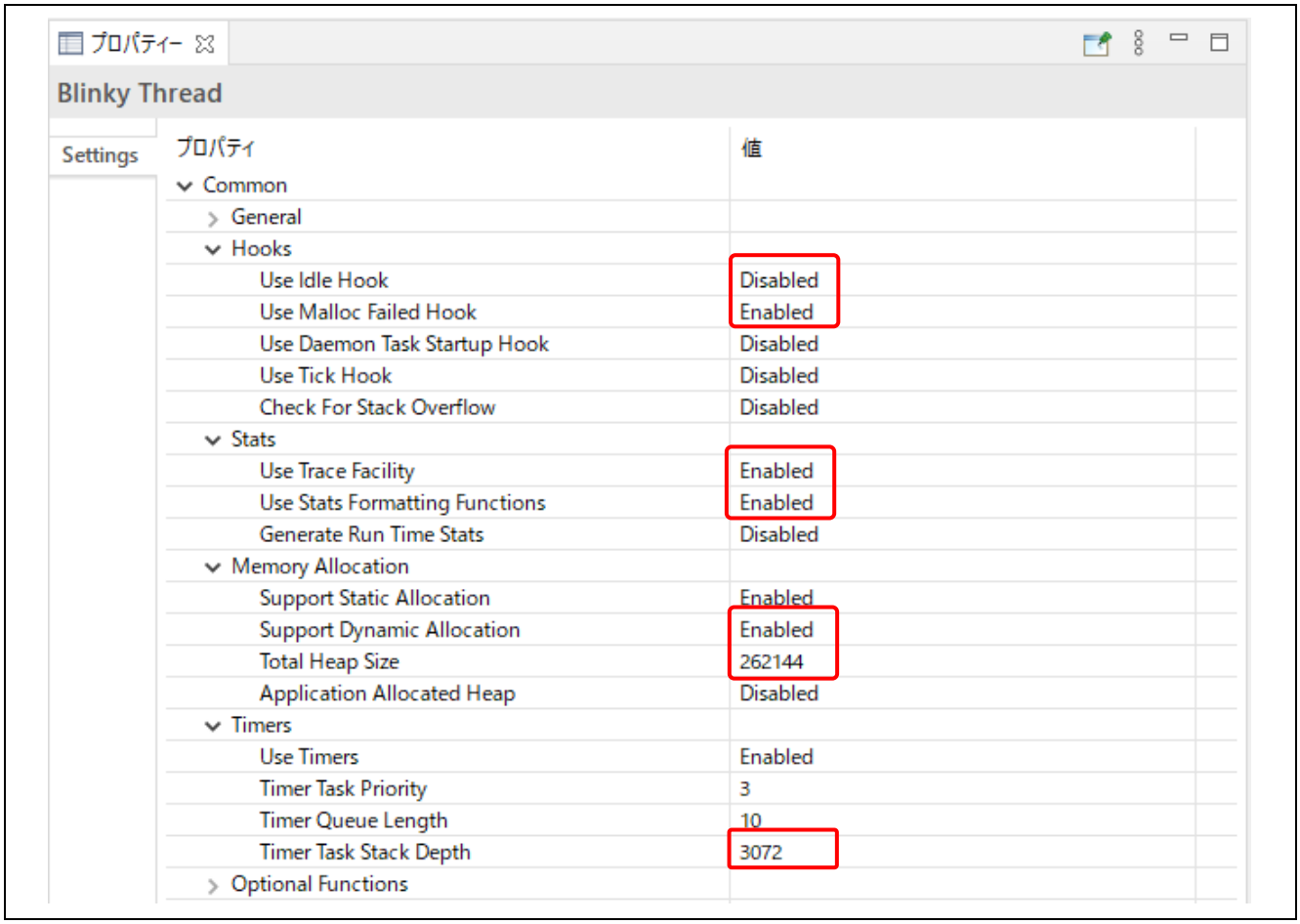


図 15　Blinky スレッドのプロパティ 2

[Blinky Thread]で、[プロパティ] → [Optional Functions]、[RA]、[Logging]の設定を次のように変更します。

- uxTaskGetStackHighWaterMark() Function：**Enabled**
- eTaskGetState() Function：**Enabled**
- xTimerPendFunctionCall() Function：**Enabled**
- xTaskAbortDelay() Function：**Enabled**
- Hardware Stack Monitor：**Enabled**
- Logging Include Time and Task Name：**Enabled**

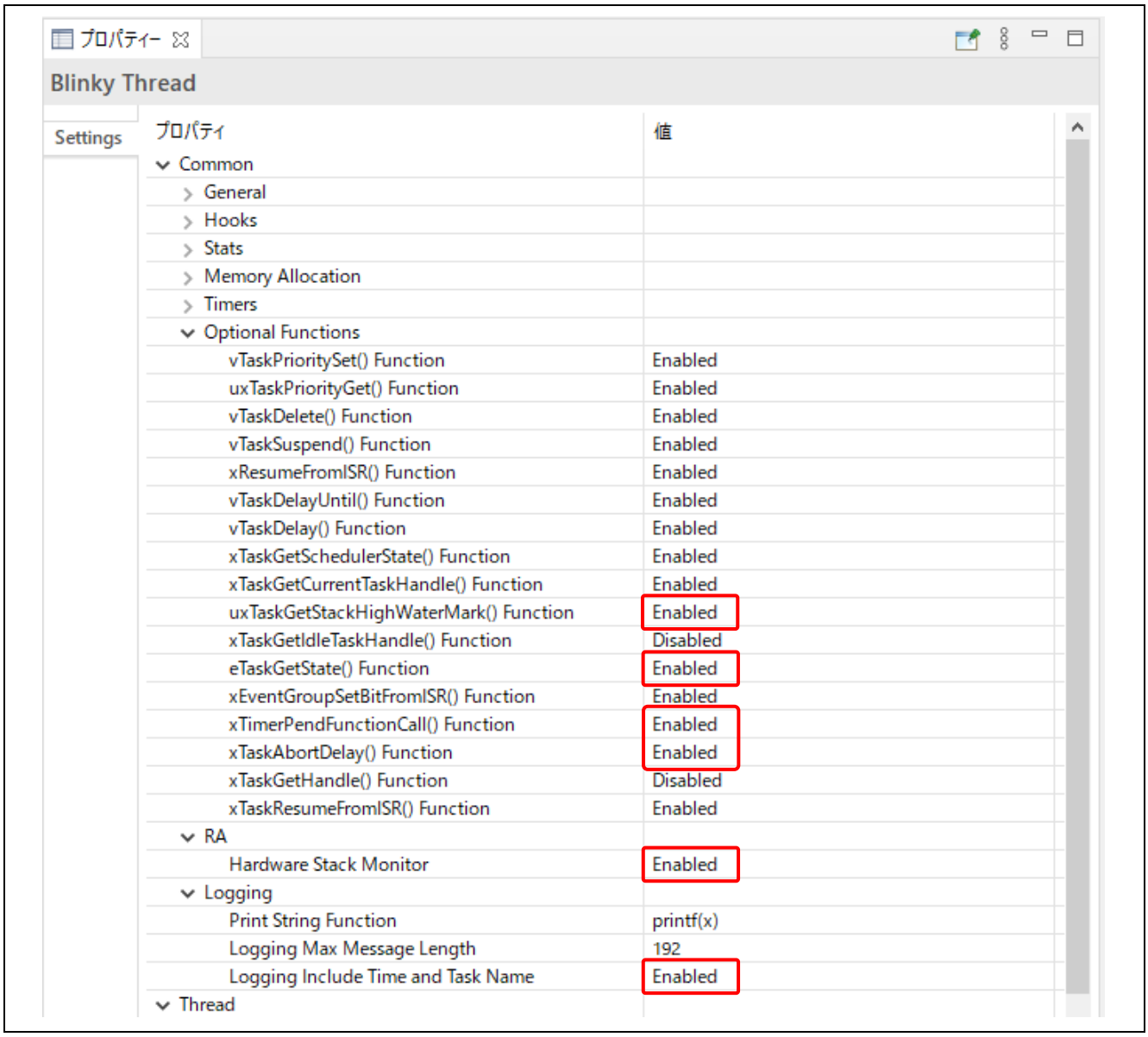
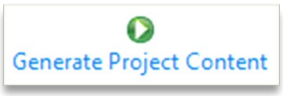


図 16　Blinky スレッドのプロパティ 3

4.2.3　プロジェクト内容の生成



ボタンをクリックしてソースファイルを生成します。

4.3　Tracealyzer®接続のためのコード編集

4.3.1　UART 用のフォルダとファイルの作成

- ここでは例として、"uart"フォルダと"trcStreamingPort.c"および"trcStreamingPort.h"ファイルを作成します。
- 図 18 および図 19 に示す内容で"trcStreamingPort.c"および"trcStreamingPort.h"ファイルを作成します。

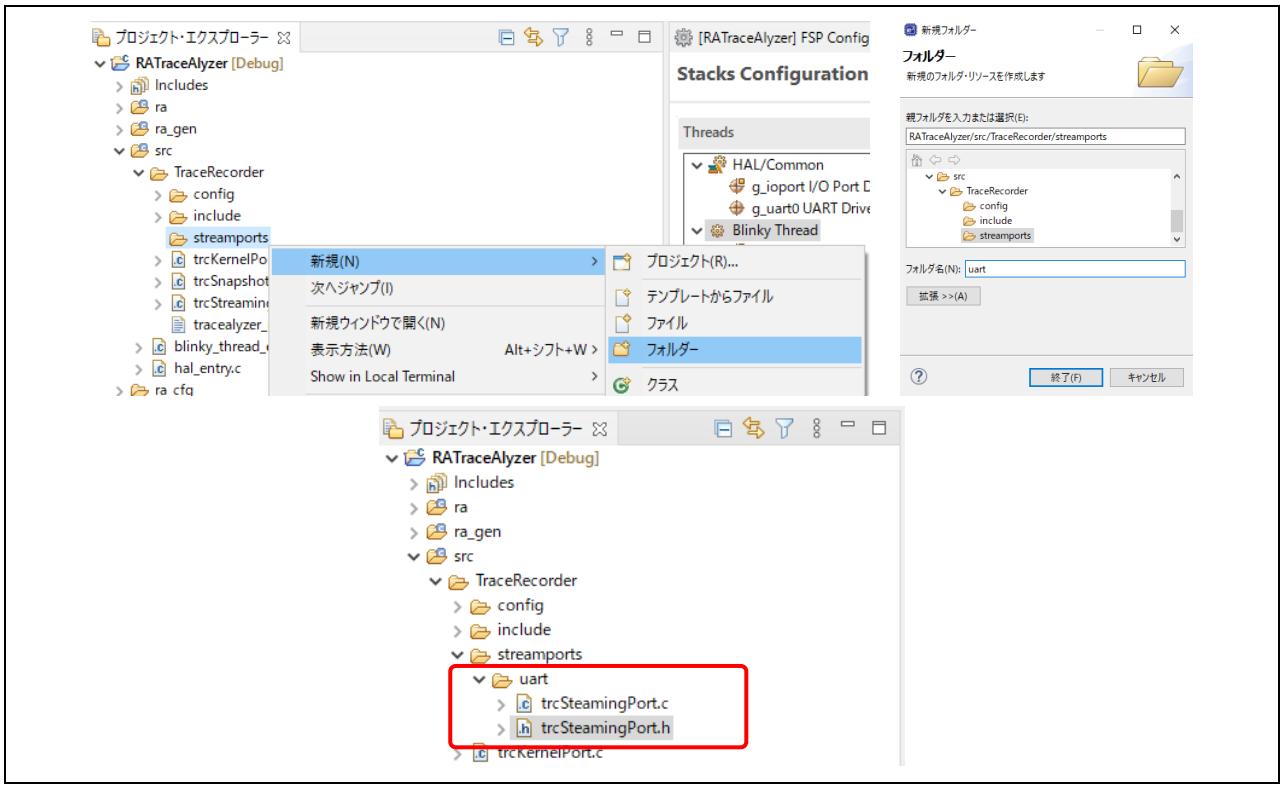


図 17　フォルダとファイルの作成

— "trcStreamingPort.c"ファイルを作成します。

```
#include "bsp_api.h"
#include "trcRecorder.h"
#include "r_sci_uart.h"
#include "r_uart_api.h"
#include <string.h>
#include "semphr.h"
#if (TRC_CFG_RECORDER_MODE == TRC_RECORDER_MODE_STREAMING)
#if (TRC_USE_TRACEALYZER_RECORDER == 1)
static uint8_t s_u8_string[1024];
static signed portBASE_TYPE xHigherPriorityTaskWoken;
static uint8_t sci_buffer[1024];
static uint32_t sci_current_received_size = 0;
extern sci_uart_instance_ctrl_t g_uart0_ctrl;
extern SemaphoreHandle_t semaphore_handle_1;

int32_t trcUartWrite(void* data, uint32_t size, int32_t *ptrBytesWritten)
{
    fsp_err_t err   = FSP_SUCCESS;
    int32_t error_code = -1;
    if(size < sizeof(s_u8_string))
    {
            memcpy(s_u8_string, data, size);
            /* Writing to terminal */
            err = R_SCI_UART_Write (&g_uart0_ctrl, s_u8_string, size);
            if(err == FSP_SUCCESS)
            {
                    xSemaphoreTake( semaphore_handle_1, portMAX_DELAY );
                    *ptrBytesWritten = size;
                    error_code = 0;
            }
    }
    return error_code;
}

int32_t trcUartRead(void* data, uint32_t size, int32_t *ptrBytesRead)
{
    if(sci_current_received_size == size)
    {
            memcpy(data, sci_buffer, sci_current_received_size);
            *ptrBytesRead = sci_current_received_size;
            sci_current_received_size = 0;
    }
    return 0;
}

void sci_callback_tracealyzer(uart_callback_args_t *p_args)
{
    if(UART_EVENT_RX_CHAR == p_args->event)
    {
            sci_buffer[sci_current_received_size] = (uint8_t ) p_args->data;
            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(UART_EVENT_TX_COMPLETE == p_args->event)
    {
            xHigherPriorityTaskWoken = pdFALSE;
            xSemaphoreGiveFromISR(semaphore_handle_1, &xHigherPriorityTaskWoken);
            portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
    }
    else
    {
    }
}
#endif /*(TRC_USE_TRACEALYZER_RECORDER == 1)*/
#endif /*(TRC_CFG_RECORDER_MODE == TRC_RECORDER_MODE_STREAMING)*/
```

図 18　"trcStreamingPort.c"ファイルの作成

— "trcStreamingPort.h"ファイルを作成します。

```
#ifndef TRC_STREAMING_PORT_H
#define TRC_STREAMING_PORT_H

#define TRC_STREAM_PORT_READ_DATA(_ptrData, _size, _ptrBytesRead) trcUartRead(_ptrData, _size,
_ptrBytesRead)
#define TRC_STREAM_PORT_WRITE_DATA(_ptrData, _size, _ptrBytesSent) trcUartWrite(_ptrData, _size,
_ptrBytesSent)

#endif
```

図 19　"trcStreamingPort.h"ファイルの作成

4.3.2　task.c および timers.c へのインクルードファイルの追加

- `#include "trcRecorder.h"`

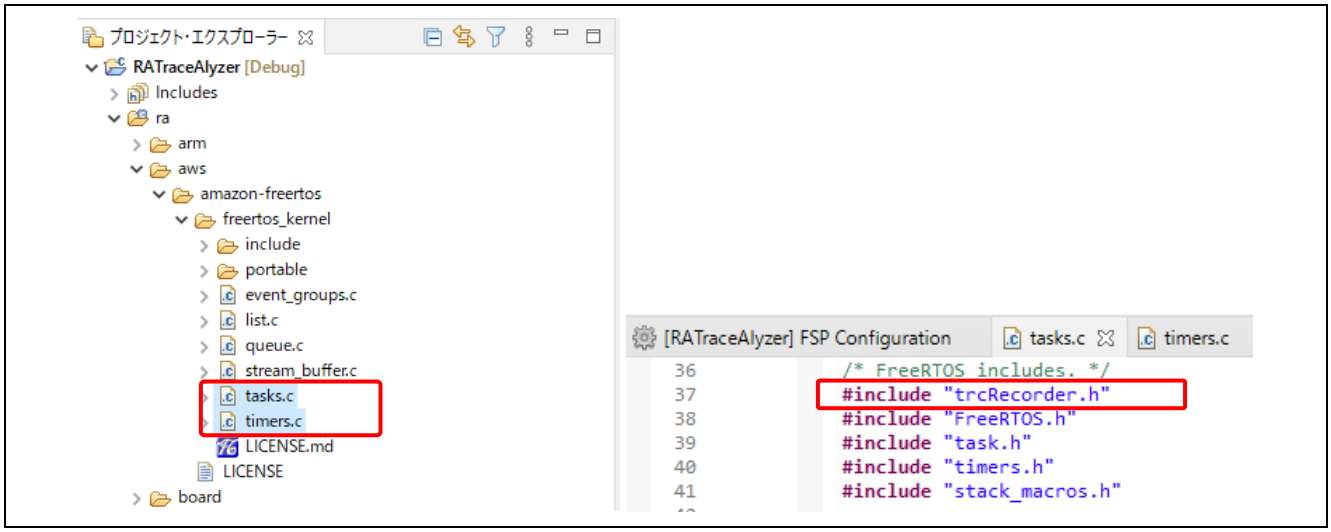


図 20　"freertos_kernel"にインクルードファイルを追加

4.3.3　trcKernelPort.c および trcStreamingRecorder.c へのインクルードファイルの追加

- `#include "bsp_api.h"`
- `#include "trcRecorder.h"`

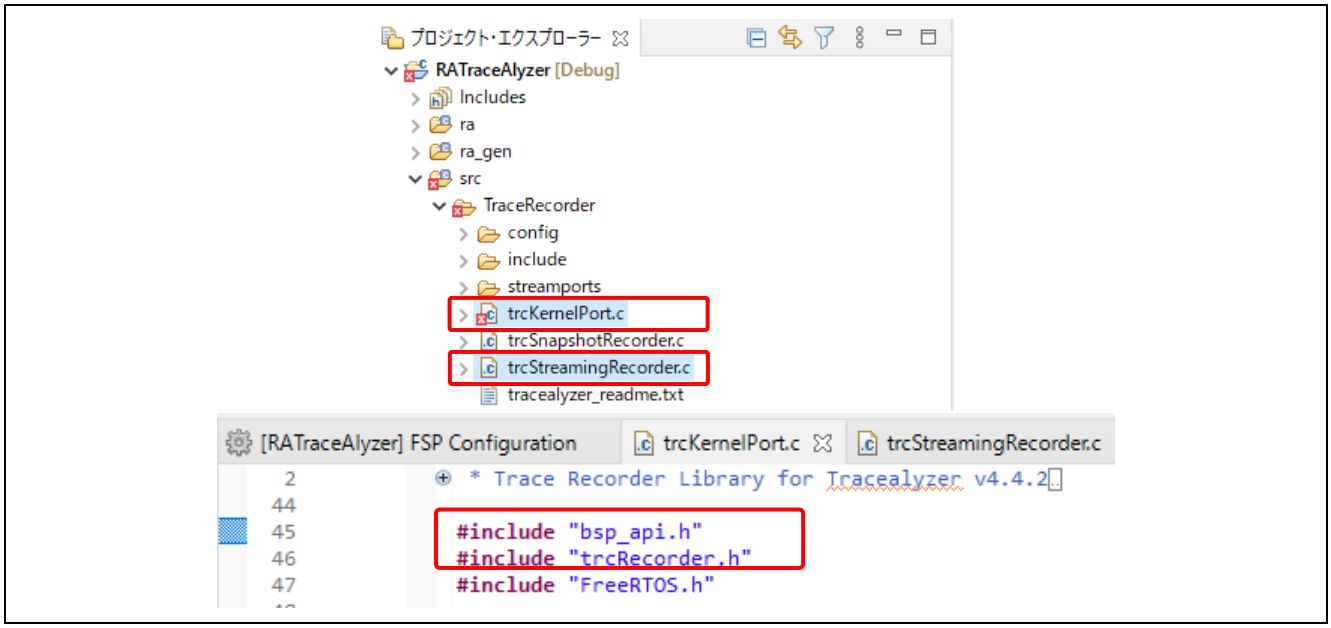


図 21　"TraceRecorder"にインクルードファイルを追加

4.3.4　trcConfig.h ファイルでのマクロ定義の変更

"trcConfig.h"ファイルでのマクロ定義のうち、以下に示すように赤字部分のみを変更します。

- #include “bsp_api.h”
- //#error "Trace Recorder: Please include your processor's header file here and remove this line."
- #define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_ARM_Cortex_M
- #define TRC_CFG_RECORDER_MODE TRC_RECORDER_MODE_STREAMING
- #define TRC_CFG_FREERTOS_VERSION TRC_FREERTOS_VERSION_10_4_1

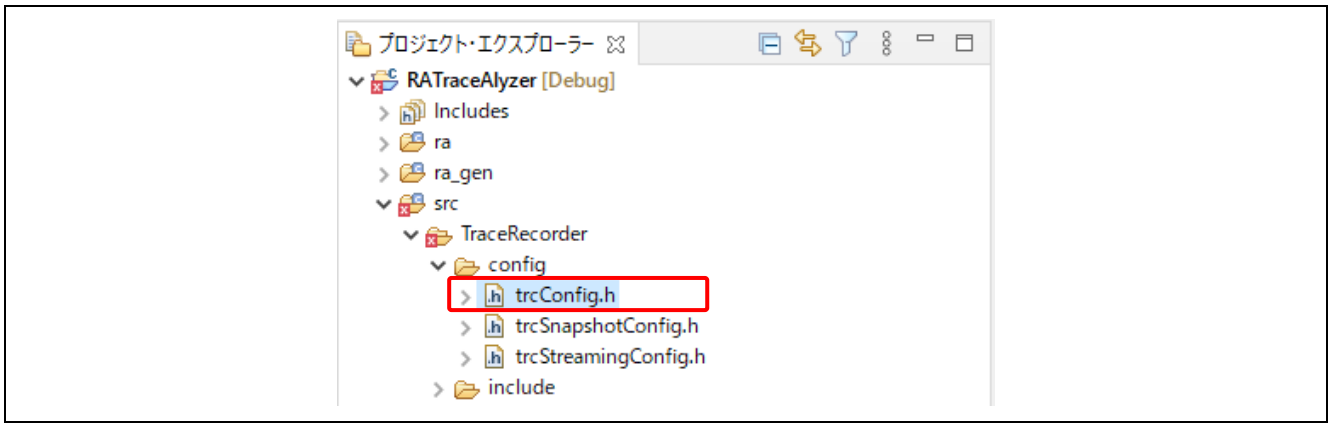


図 22　"trcConfig.h"のマクロ定義の変更

4.3.5　e² studio プロパティへのインクルードパスの追加

メニューから[Project] → [Properties]の順に選択した後、[設定] → [Includes]をクリックしてインクルードパスを追加します。

- "${workspace_loc:/${ProjName}/src/TraceRecorder}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/config}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/include}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/uart}"

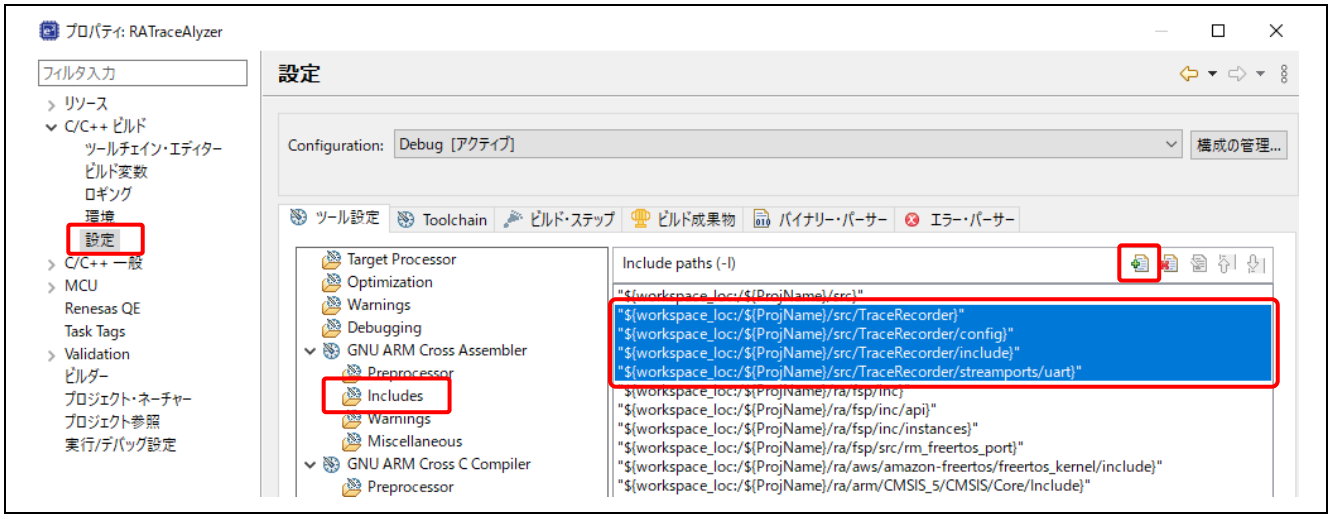


図 23　プロジェクトプロパティへのインクルードパスの設定

4.3.6　hal_entry.c へのコードの追加

以下の赤字部分を"hal_entry.c"のソースコードに追加します。

```
#include "bsp_api.h"
#include "trcRecorder.h"
#include "FreeRTOS.h"
#include "semphr.h"
#include "hal_data.h"

SemaphoreHandle_t semaphore_handle_1;
StaticSemaphore_t semaphore_handle_1_memory;
void R_BSP_WarmStart(bsp_warm_start_event_t event);

/*******************************************************************************************************************//**
 * This function is called at various points during the startup process.  This implementation uses the event that is
 * called right before main() to set up the pins.
 *
 * @param[in]  event    Where at in the start up process the code is currently at
 **********************************************************************************************************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable here, before clock and
         * C runtime initialization, should negate the need for a delay since the initialization will typically take more than 6us. */
#endif
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }

    fsp_err_t err = FSP_SUCCESS;

    /* Initialize UART channel with baud rate 115200 */
    err = R_SCI_UART_Open (&g_uart0_ctrl, &g_uart0_cfg);
    if (FSP_SUCCESS != err)
    {
    }
    semaphore_handle_1 = xSemaphoreCreateBinaryStatic (&semaphore_handle_1_memory);
    vTraceEnable(TRC_INIT);
}
```

図 24　UART ストリーミングのための hal_entry.c の変更

4.3.7　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。

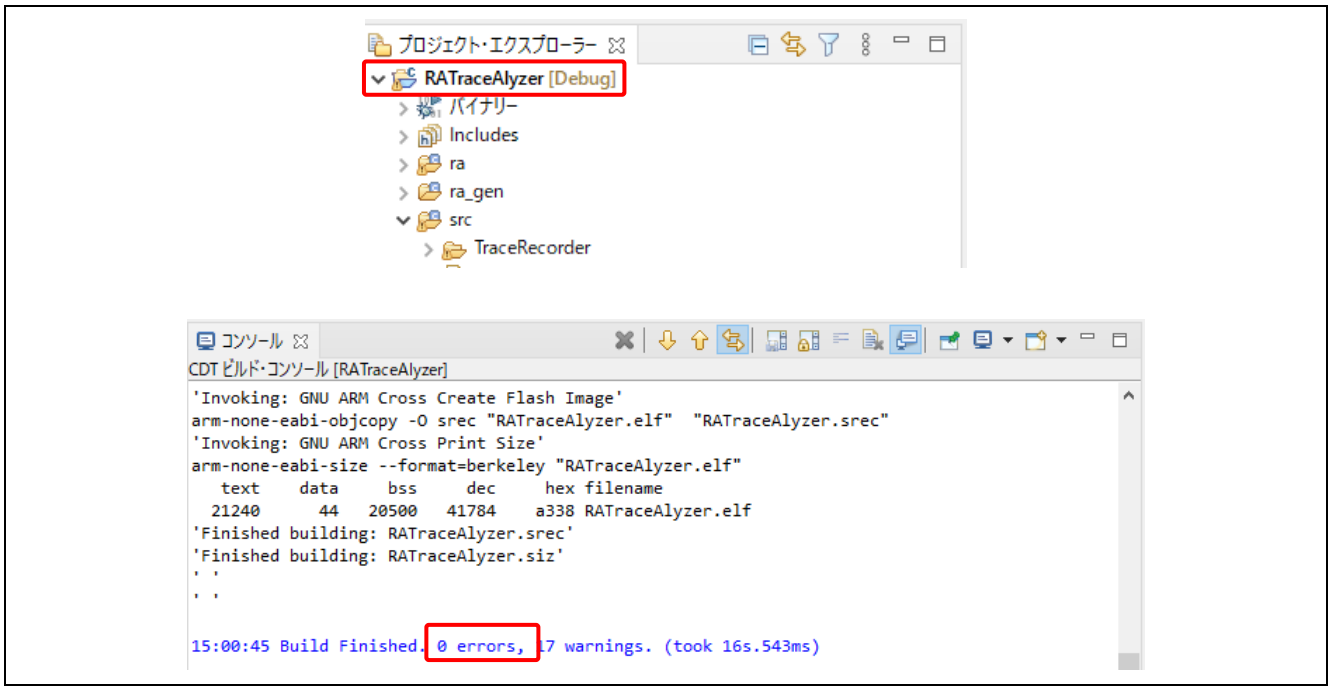


図 25　プロジェクトのビルド

4.4　ホスト PC と EK-RA6M3 ボードの接続

図 26 にホスト PC と EK-RA6M3 ボードの接続を示します。

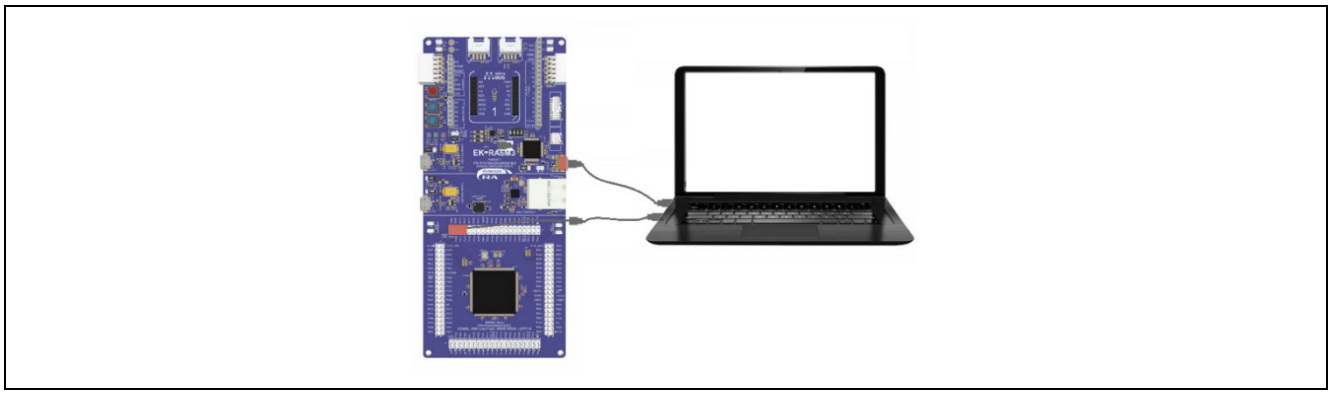


図 26　EK-RA6M3 ボードの接続

ハードウェア設定を以下に示します。

表 1　デバッグモード別のジャンパ接続一覧

デバッグモード	J8	J9	J29
オンボードデバッグ	ピン 1-2 にジャンパを取り付け	オープン	ピン 1-2、3-4、5-6、7-8 にジャンパを取り付け

端子接続 (UART) :

- ホスト PC の TXD ポート（オレンジ色）→ EK-RA6M3 ボードの P410 (RXD_MISO)
- ホスト PC の RXD ポート（黄色）　　→ EK-RA6M3 ボードの P411 (TXD_MOSI)
- ホスト PC の GND ポート（黒色）　　→ EK-RA6M3 ボードの GND

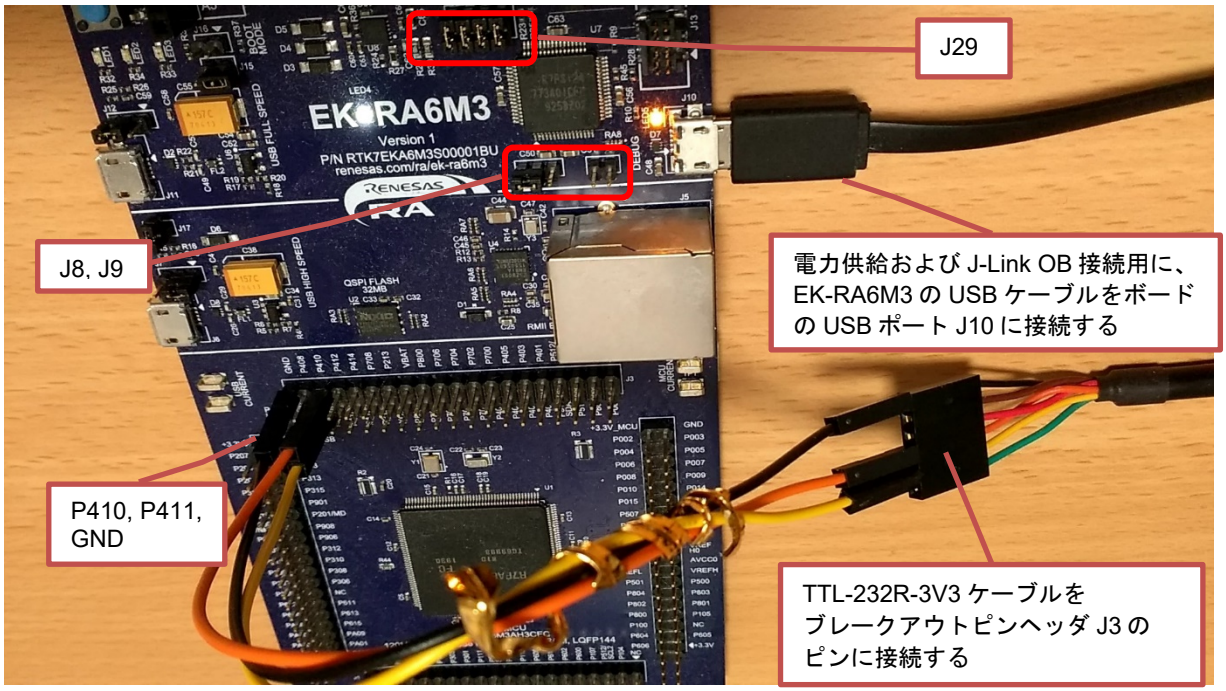


図 27　ホスト PC と EK-RA6M3 ボードの接続

4.5　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。

4.5.1　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]ダイアロクボックスが表示されたら図 28 のように"FreeRTOS"を選択してください。図 29 のように[RTOS リソース]ビューが開きます。

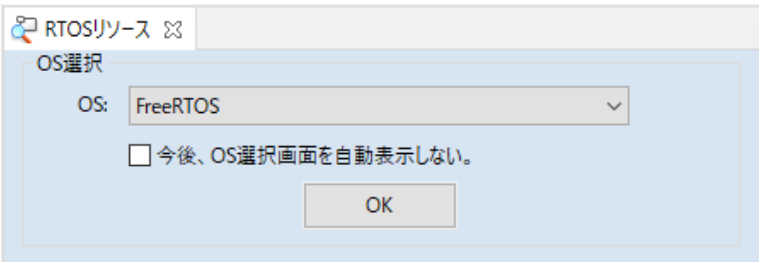


図 28　OS の選択

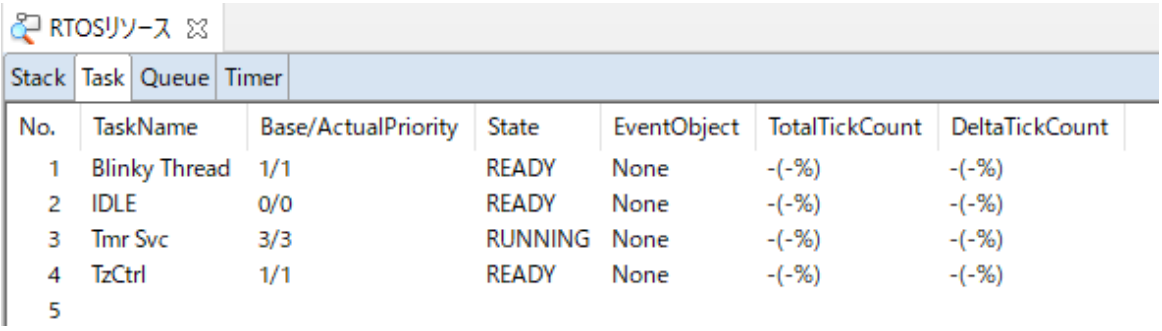


図 29　[RTOS リソース]ビュー

4.5.2　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。

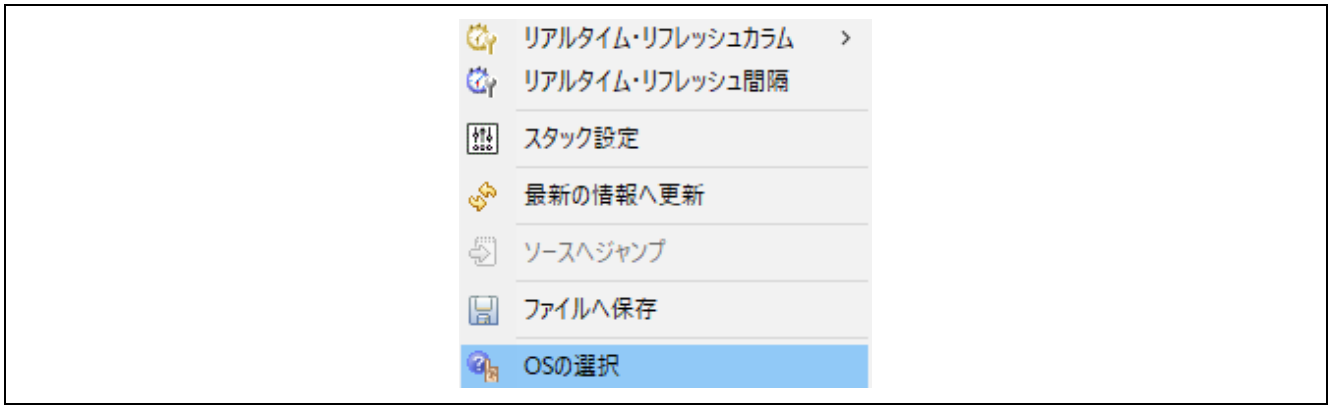


図 30　コンテキストメニュー

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。

4.5.3　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。

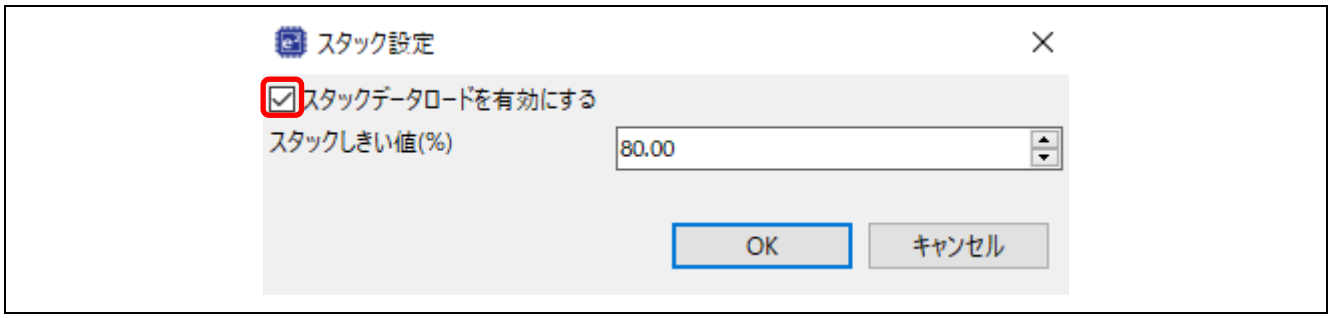


図 31　スタックデータロードの有効化

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。

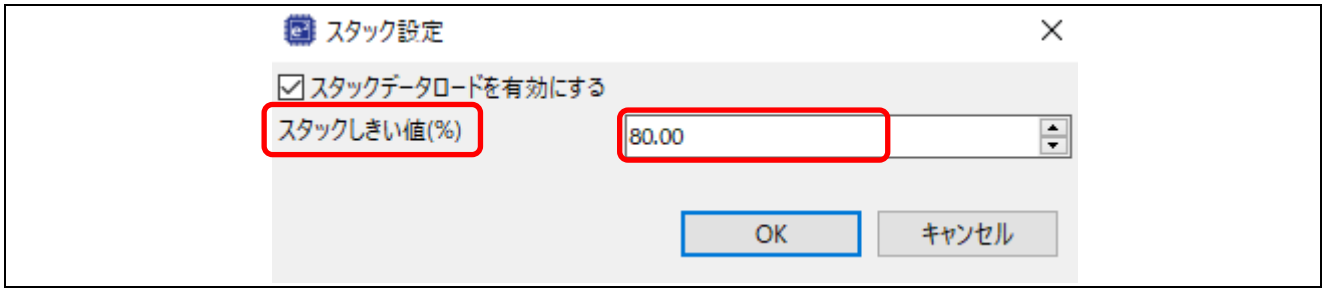


図 32　スタックしきい値の設定

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。

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。

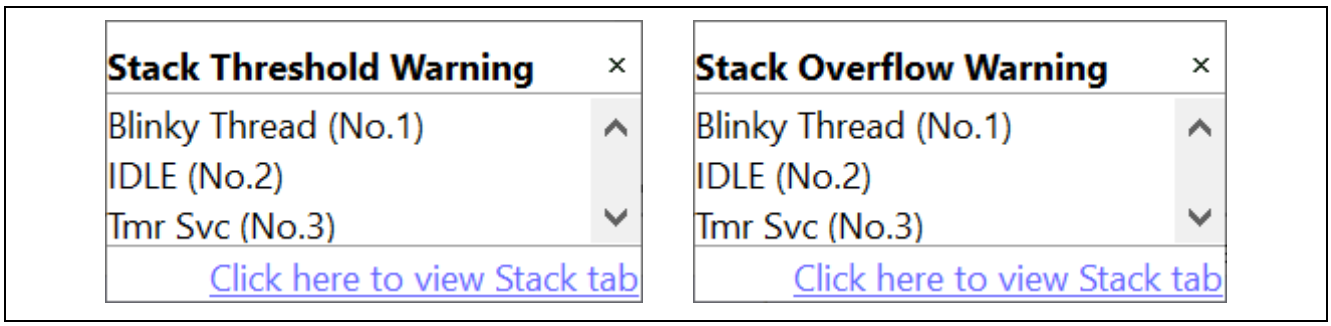


図 33　[Stack Threshold Warning]表示（左）と[Stack Overflow Warning]表示（右）

4.5.4　タブメニュー

各タブに表示する項目を表 2 に示します。

表 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)	キュー登録時に割り当てられた名称とタイプ（キュー、セマフォ、ミューテックスのいずれか）
	Address	キューハンドルのアドレス
	MaxLength	キュー内の要素のサイズ（バイト）
	ItemSize	メッセージサイズ
	CurrentLength	現在キューに格納されている要素の数
	#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）

4.6　Tracealyzer®を使用したプロジェクトデバッグの開始

4.6.1　e² studio でのデバッガの起動

メニューから[実行] → [デバッグ]の順に選択してデバッガを起動します。

4.6.2　Tracealyzer®の起動

ホスト PC にインストールした Tracealyzer 4 を起動します。

4.6.3　Recording Settings の設定

Tracealyzer で[Recording Settings]をクリックし、[PSF Streaming Settings]を選択して、次のように設定します。

- Device：**COM8**（ユーザ PC システムポート）
- Data bits：**8**
- Data rate：**921600**
- Handshake：**None**
- Parity：**None**
- Stop bits：**One**

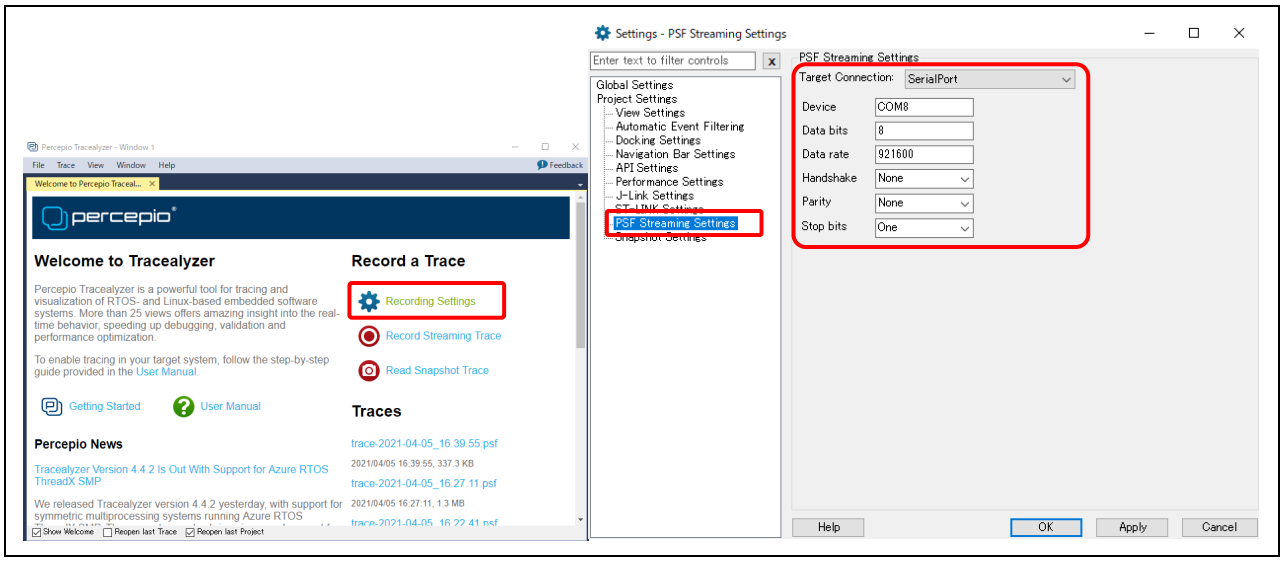


図 34　トレース記録に用いる UART の設定

4.6.4　トレース記録の開始

[Record Streaming Trace]をクリックしてトレース記録を開始します。

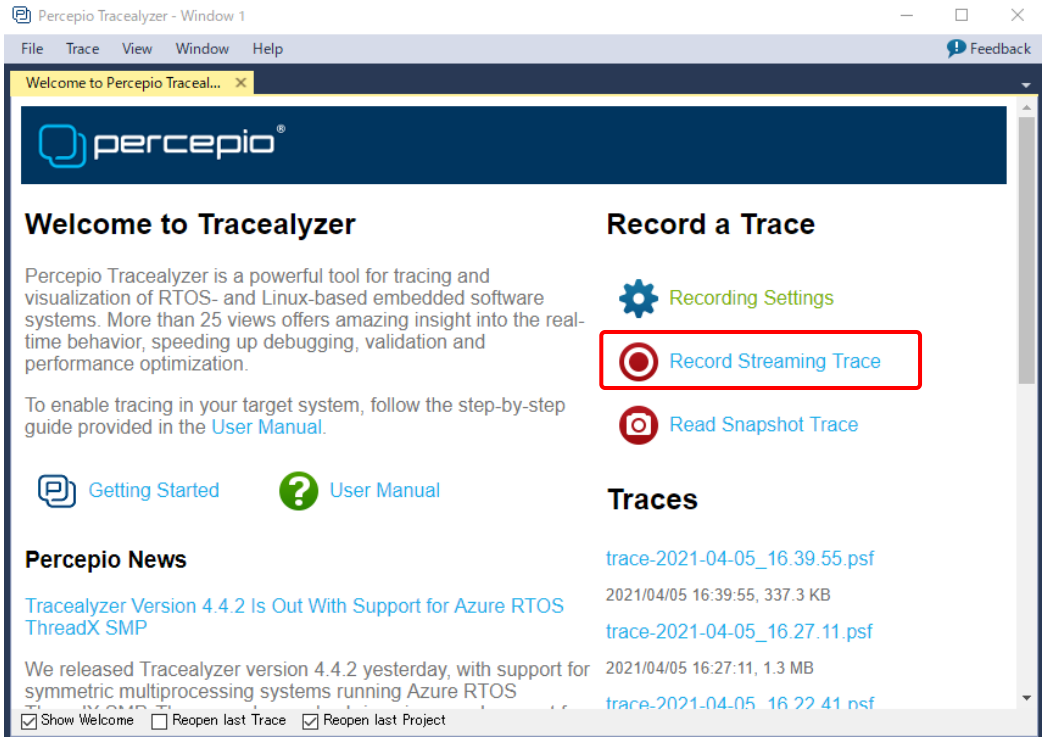


図 35　ストリーミングトレース記録の開始

4.6.5　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。

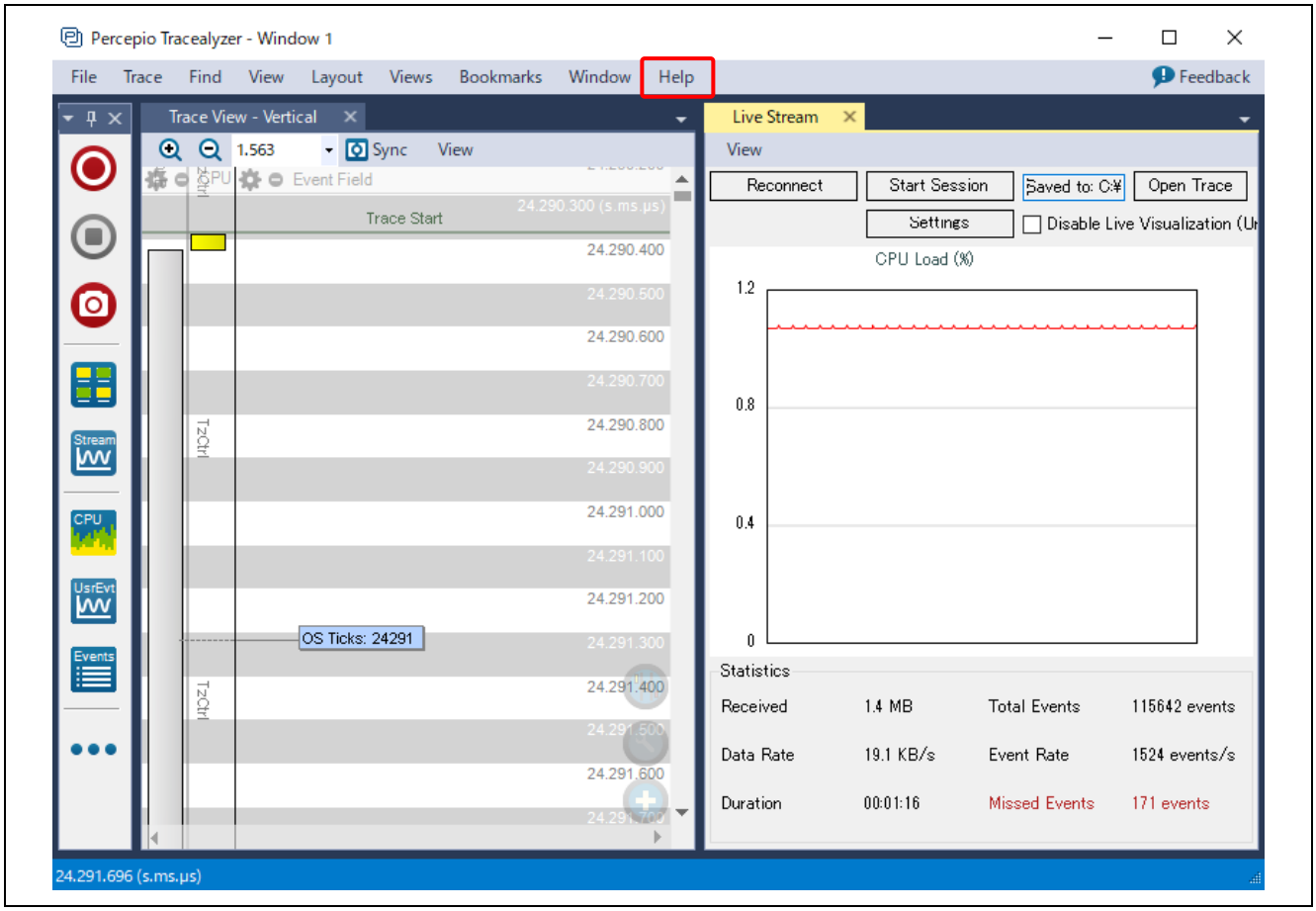


図 36　トレース情報の表示

5.　Tracealyzer®を用いたデバッグ（J-Link RTT 使用時）

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

5.1　プロジェクトへの Tracealyzer® for FreeRTOS のコピーと不要フォルダの削除

5.1.1　Tracealyzer®インストールフォルダへの 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。

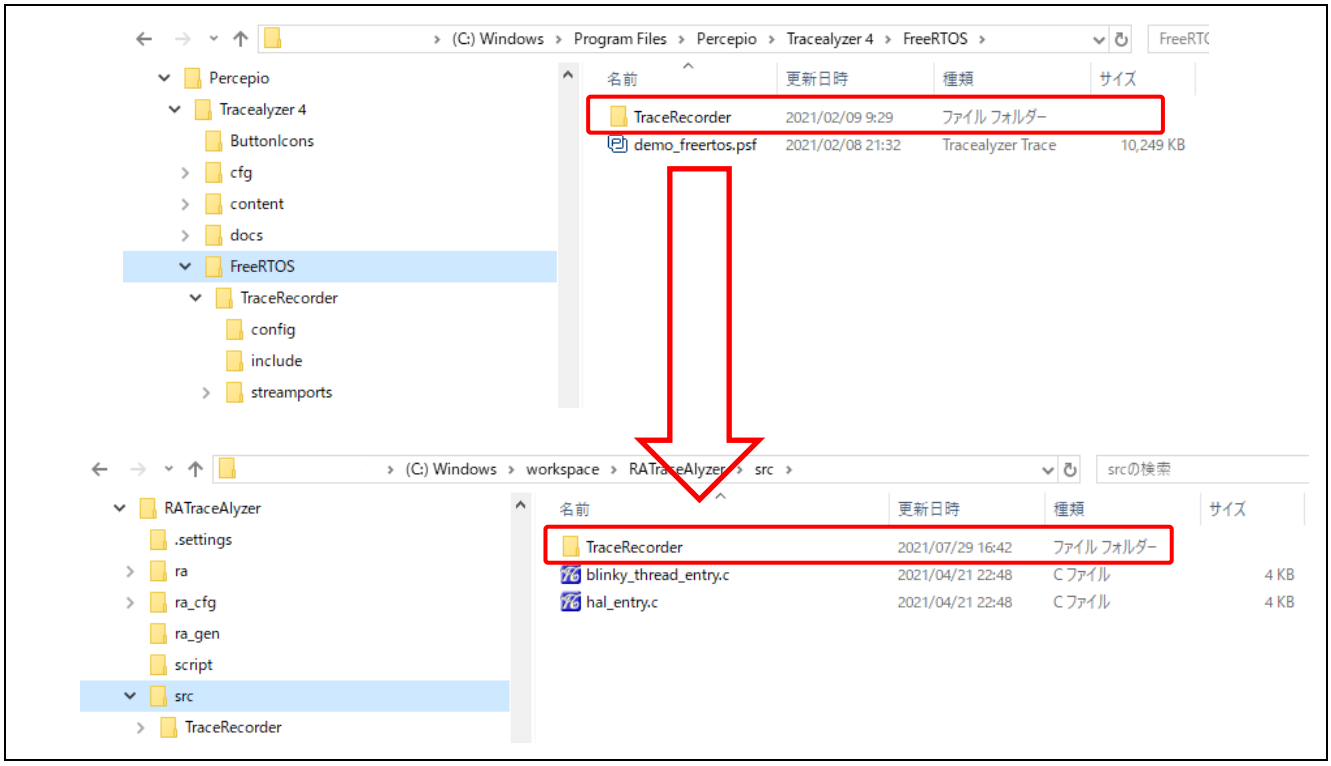


図 37　フォルダのコピー

5.1.2　不要フォルダの削除

1. ワークスペースフォルダ"`src/TraceRecorder/streamports`"から、図 38 の赤枠で囲んだサブフォルダを削除します。

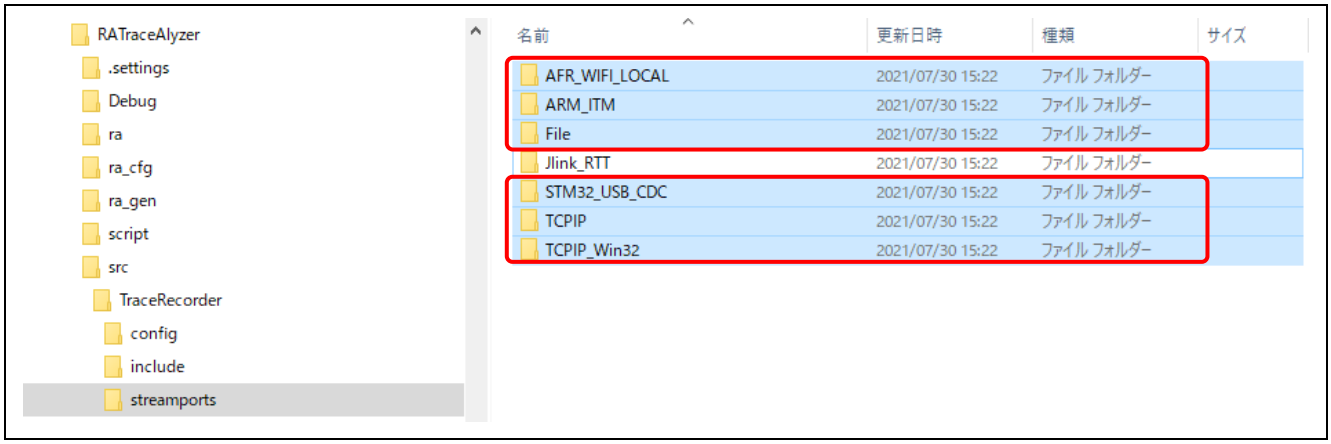


図 38　フォルダの削除（J-Link RTT）

2. J-Link RTT のファイルを EK-RA6M3 Example Project Bundle - Sample Code にコピーします。
   ファイルエクスプローラーで、
   "\ek_ra6m3\sci_uart\sci_uart_ek_ra6m3_ep\e2studio\src\SEGGER_RTT"フォルダのファイルをワークスペースフォルダ"src/TraceRecorder/streamports"に上書きします。
   — SEGGER_RTT.c ファイル
   — SEGGER_RTT_printf.c ファイル
   — SEGGER_RTT.h ファイル
   — SEGGER_RTT_Conf.h ファイル

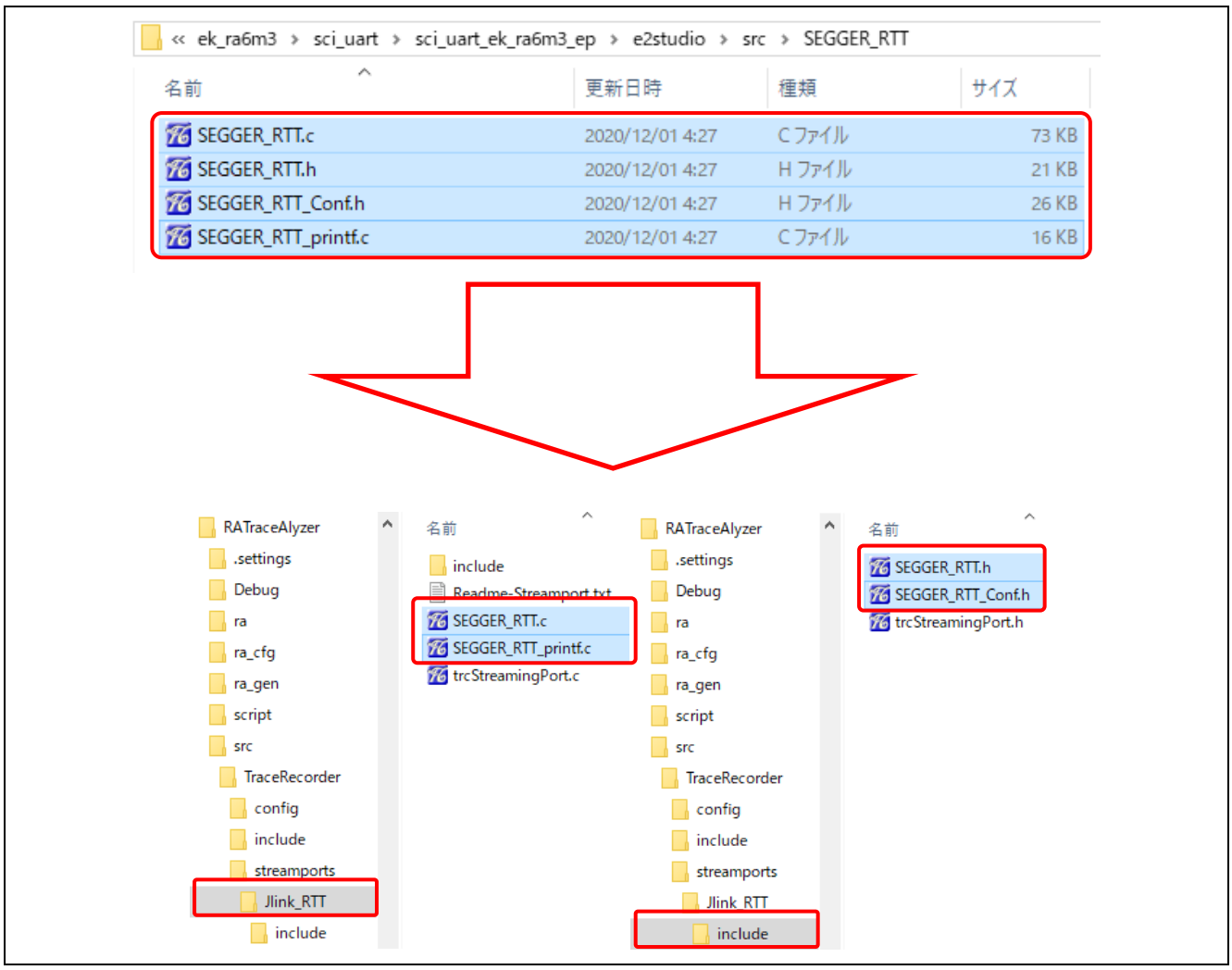


図 39　SEGGER_RTT ファイルのコピー

5.2　FSP の構成

5.2.1　Blinky スレッドの設定

Blinky スレッドに次のようにヒープを追加します。

- [Stacks]タブを開きます。
- [Blinky Thread]を選択します。
- [New Stack] → [FreeRTOS] → [Memory Management] → [Heap 1]の順に選択します。

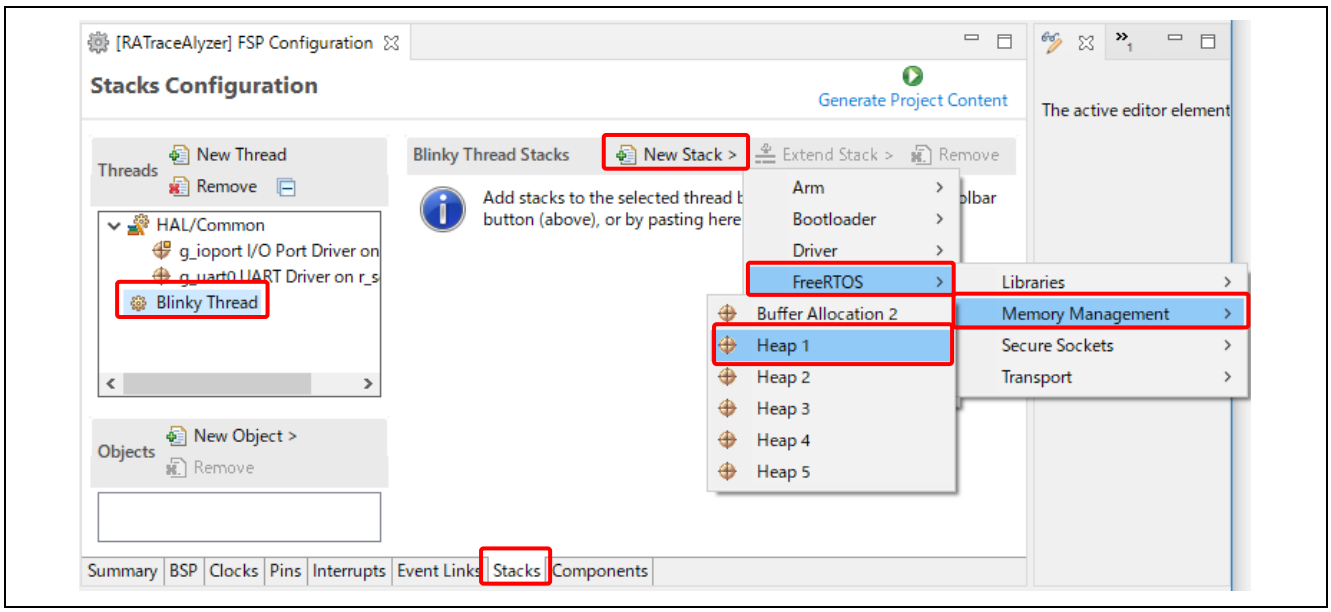


図 40　[Stacks Configuration]でヒープを追加

[Blinky Thread]で、[プロパティ] → [General]の設定を次のように変更します。

- Minimal Stack Size：**512**
- Use Mutexes：**Enabled**
- Use Recursive Mutexes：**Enabled**
- Use Queue Sets：**Enabled**
- Enable Backward Compatibility：**Enabled**

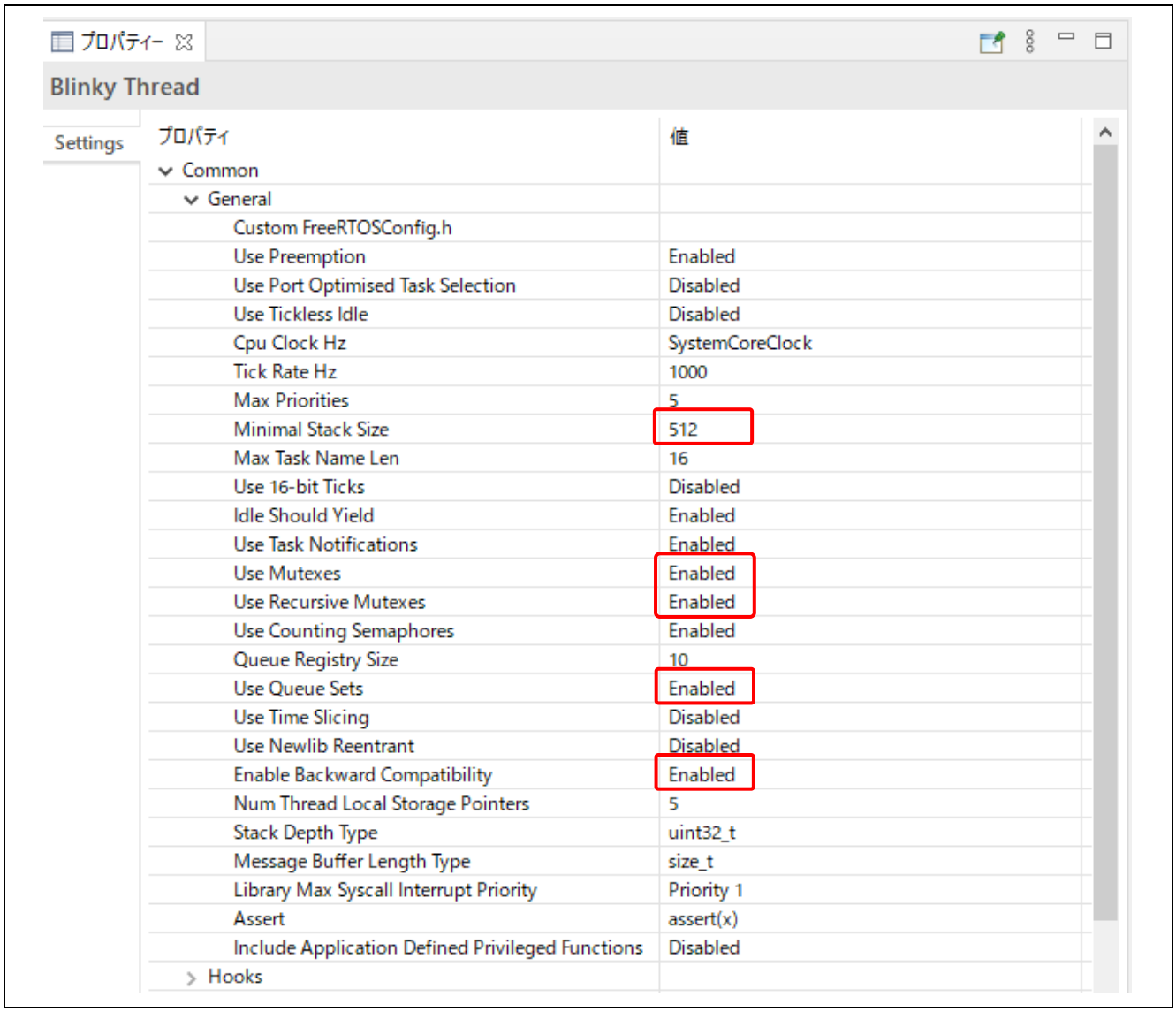


図 41　Blinky スレッドのプロパティ 1

[Blinky Thread]で、[プロパティ] → [Hooks]、[Stats]、[Memory Allocation]、[Timers]の設定を次のように変更します。

- Use Idle Hook：**Disabled**
- Use Malloc Failed Hook：**Enabled**
- Use Trace Facility：**Enabled**
- Use Stats Formatting Functions：**Enabled**
- Support Dynamic Allocation：**Enabled**
- Total Heap Size：**262,144** (256 * 1,024)
- Timer Task Static Depth：**3,072** (1024 * 3)

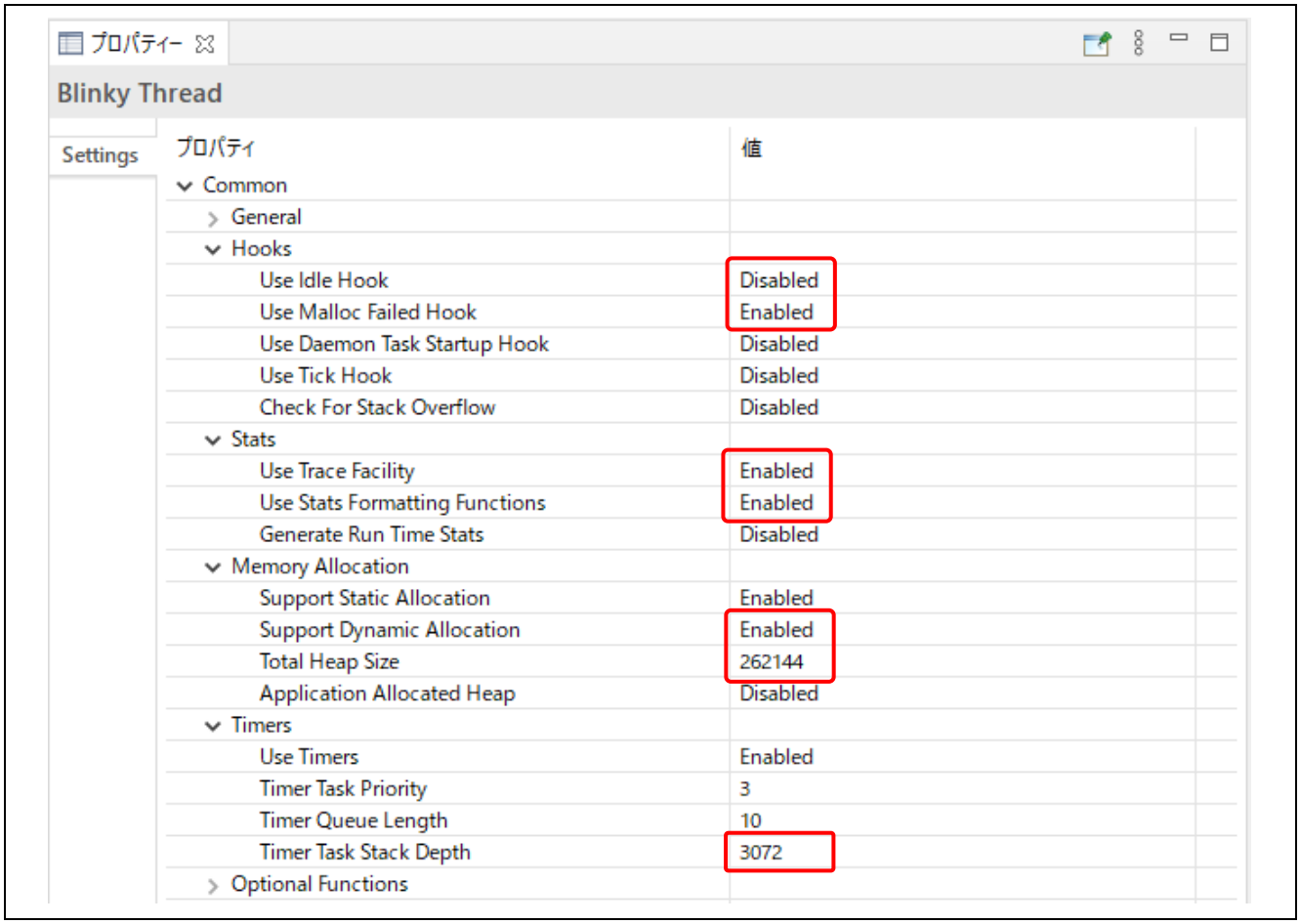


図 42　Blinky スレッドのプロパティ 2

[Blinky Thread]で、[プロパティ] → [Optional Functions]、[RA]、[Logging]の設定を次のように変更します。

- uxTaskGetStackHighWaterMark() Function：**Enabled**
- eTaskGetState() Function：**Enabled**
- xTimerPendFunctionCall() Function：**Enabled**
- xTaskAbortDelay() Function：**Enabled**
- Hardware Stack Monitor：**Enabled**
- Logging Include Time and Task Name：**Enabled**

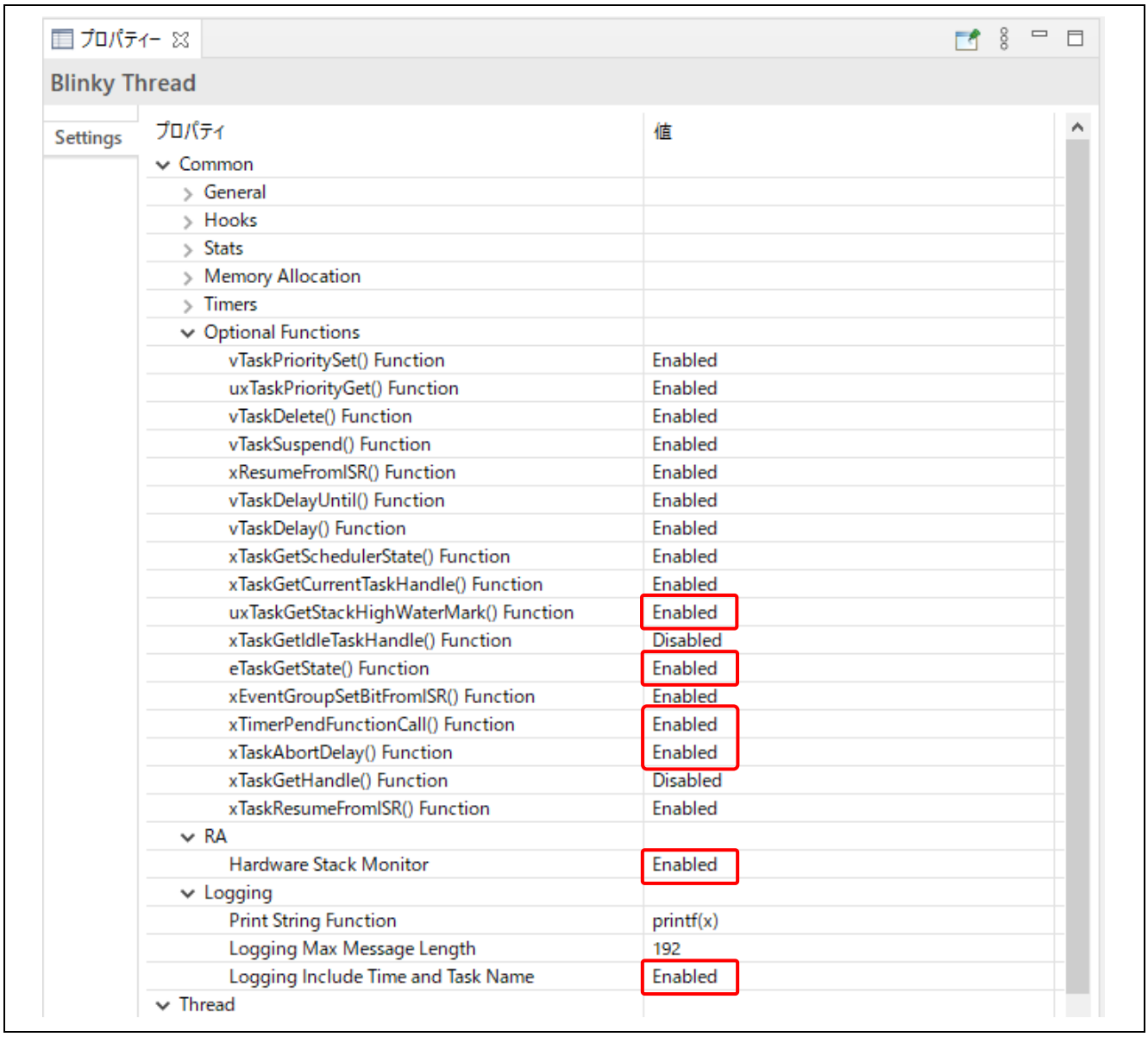
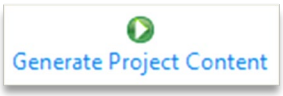


図 43　Blinky スレッドのプロパティ 3

5.2.2　プロジェクト内容の生成



ボタンをクリックしてソースファイルを生成します。

5.3　Tracealyzer®接続のためのコード編集

5.3.1　task.c および timers.c へのインクルードファイルの追加

- `#include "trcRecorder.h"`

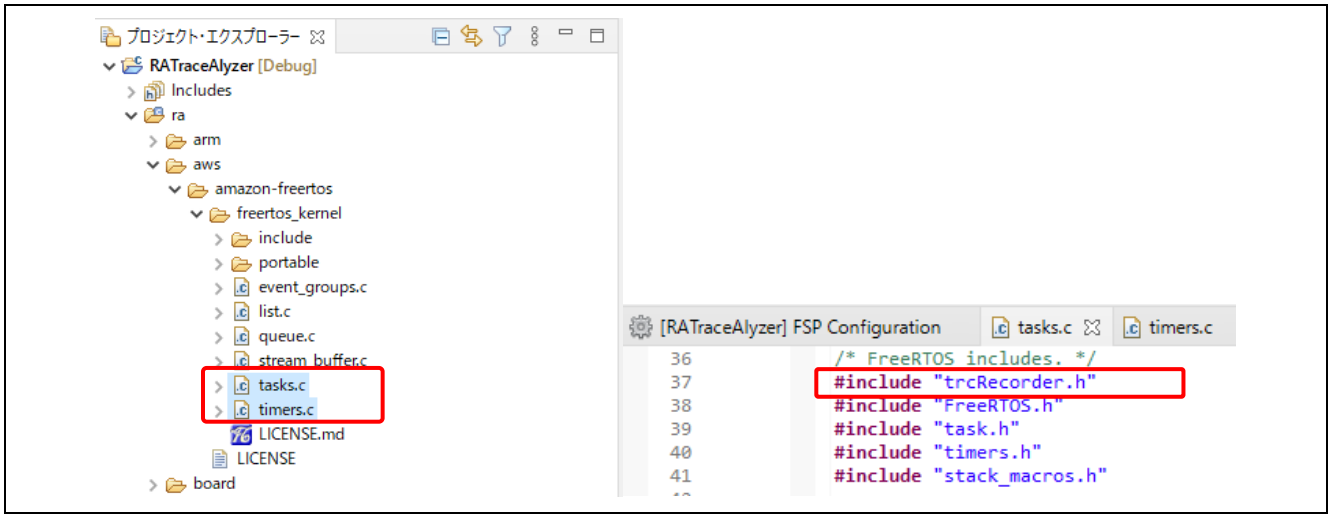


図 44　"freertos_kernel"にインクルードファイルを追加

5.3.2　trcKernelPort.c および trcStreamingRecorder.c へのインクルードファイルの追加

- `#include "bsp_api.h"`
- `#include "trcRecorder.h"`

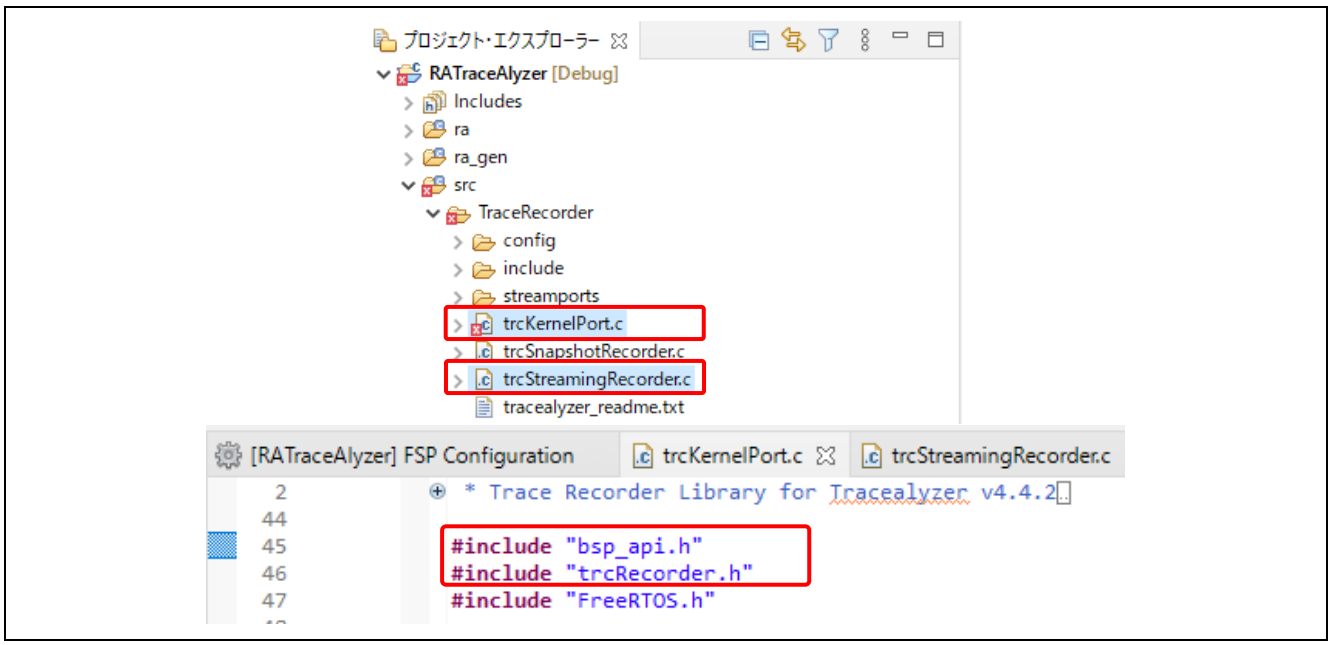


図 45　"TraceRecorder"にインクルードファイルを追加

5.3.3　trcConfig.h ファイルでのマクロ定義の変更

"trcConfig.h"ファイルでのマクロ定義のうち、以下に示すように赤字部分のみを変更します。

- #include “bsp_api.h”.
- //#error "Trace Recorder: Please include your processor's header file here and remove this line.".
- #define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_ARM_Cortex_M.
- #define TRC_CFG_RECORDER_MODE TRC_RECORDER_MODE_STREAMING.
- #define TRC_CFG_FREERTOS_VERSION TRC_FREERTOS_VERSION_10_4_1.

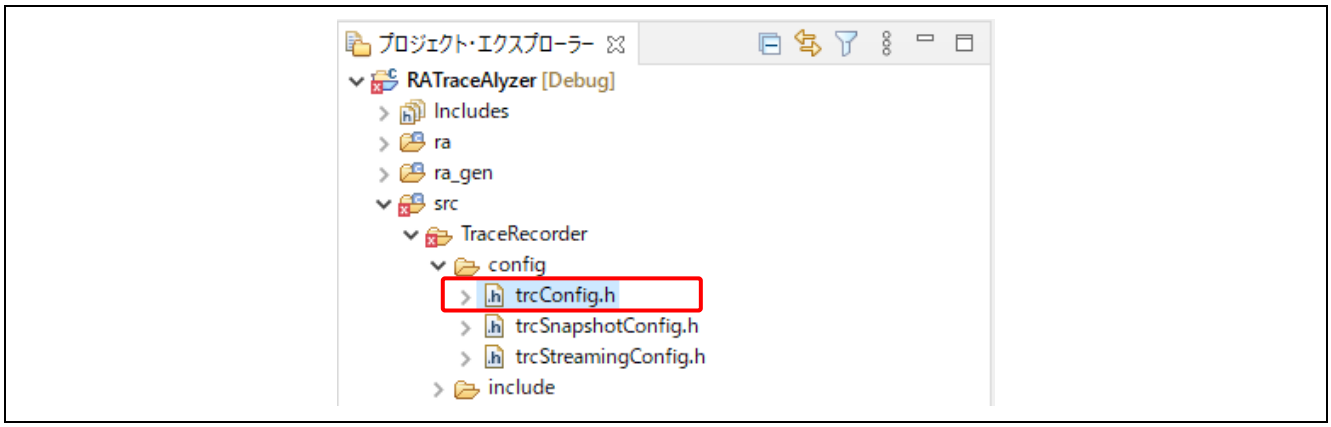


図 46　"trcConfig.h"のマクロ定義の変更

5.3.4　e² studio プロパティへのインクルードパスの追加

メニューから[Project] → [Properties]の順に選択した後、[設定] → [Includes]をクリックしてインクルードパスを追加します。

- "${workspace_loc:/${ProjName}/src/TraceRecorder}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/config}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/include}"
- "${workspace_loc:/${ProjName}/src/TraceRecorder/streamports/Jlink_RTT/include}"

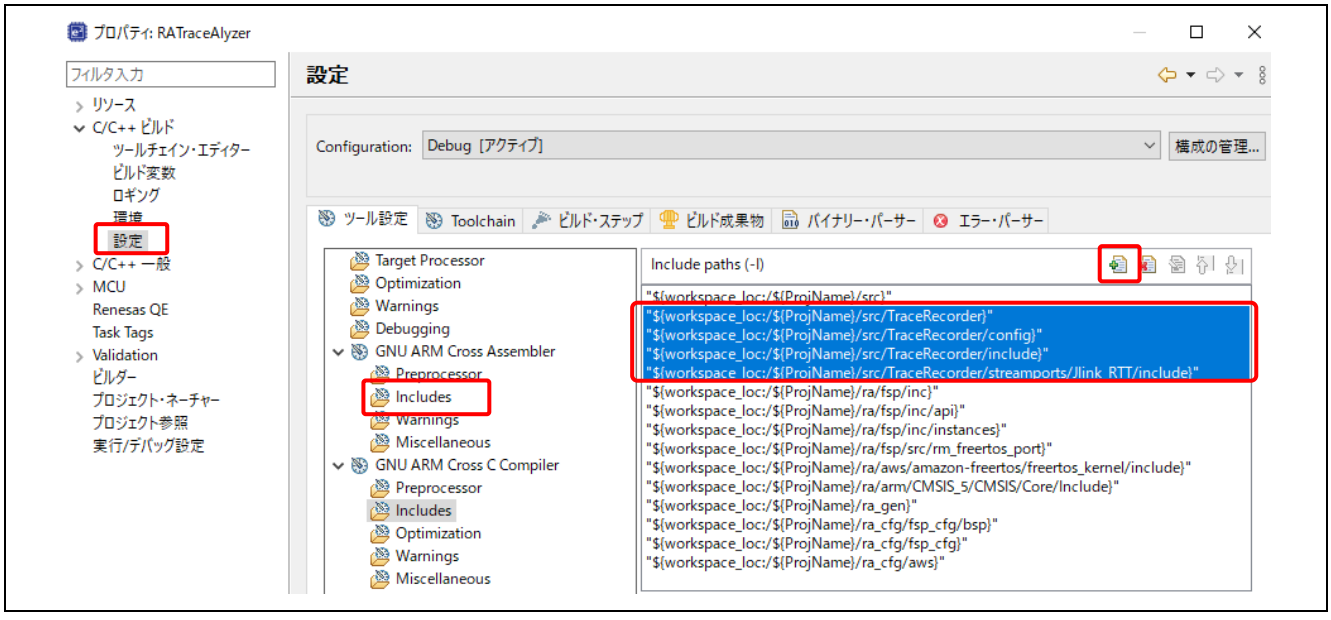


図 47　プロジェクトプロパティへのインクルードパスの設定

5.3.5　hal_entry.c へのコードの追加

以下の赤字部分を"hal_entry.c"のソースコードに追加します。

```
#include "bsp_api.h"
#include "trcRecorder.h"
#include "FreeRTOS.h"
#include "semphr.h"
#include "hal_data.h"

void R_BSP_WarmStart(bsp_warm_start_event_t event);

/*******************************************************************************************************************//**
 * This function is called at various points during the startup process.  This implementation uses the event that is
 * called right before main() to set up the pins.
 *
 * @param[in]  event    Where at in the start up process the code is currently at

**********************************************************************************************************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait tDSTOP(6us) for data flash recovery. Placing the enable here, before clock and
         * C runtime initialization, should negate the need for a delay since the initialization will typically take more than 6us. */
#endif
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }

    vTraceEnable(TRC_INIT);
}
```

図 48　J-Link RTT ストリーミングのための hal_entry.c 変更

5.3.6　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。

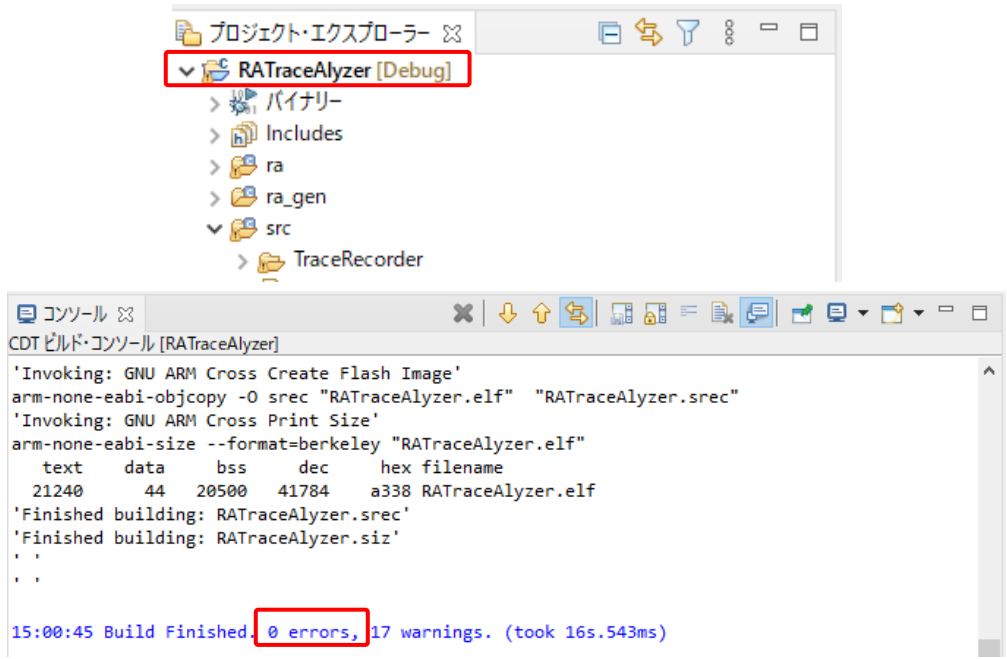


図 49　プロジェクトのビルド

5.4　ホスト PC と EK-RA6M3 ボードの接続

図 50 にホスト PC と EK-RA6M3 ボードの接続を示します。

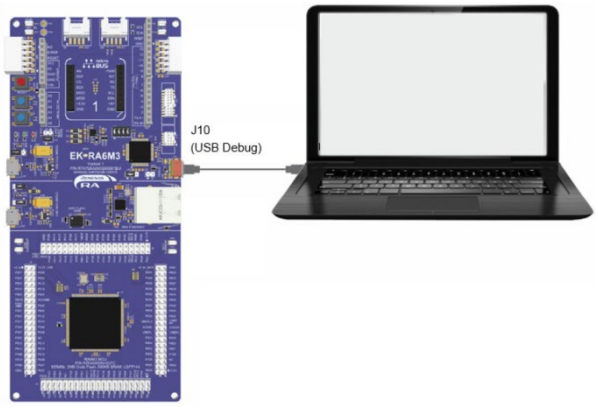


図 50　EK-RA6M3 ボードの接続

ハードウェア設定を以下に示します。

表 3　デバッグモード別のジャンパ接続一覧

デバッグモード	J8	J9	J29
オンボードデバッグ	ピン 1-2 にジャンパを取り付け	オープン	ピン 1-2、3-4、5-6、7-8 にジャンパを取り付け

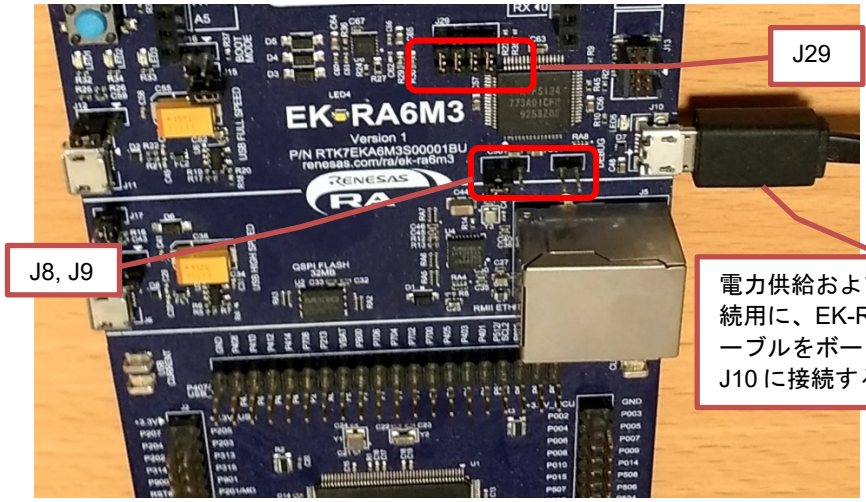


図 51　ホスト PC と EK-RA6M3 ボードの接続

5.5　RTOS リソースビューの使用

「4.5　RTOS リソースビューの使用」を参照してください。

5.6　Tracealyzer®を使用したプロジェクトデバッグの開始

5.6.1　e² studio でのデバッガの起動

メニューから[実行] → [デバッグ]の順に選択してデバッガを起動します。

5.6.2　Tracealyzer®の起動

ホスト PC にインストールした Tracealyzer 4 を起動します。

5.6.3　Recording Settings の設定

図 52 に示すように、map ファイルで RTT コントロールブロックのアドレス（赤字部分）を確認します。

```
*(COMMON)
 COMMON          0x1ffe2fa4
0xa8  ./src/TraceRecorder/streamports/Jlink_RTT/SEGGER_RTT.o
                               0x1ffe2fa4                _SEGGER_RTT
```

図 52　RATraceAlyzer.map（map ファイル）

Tracealyzer で[Recording Settings]をクリックし、[J-Link Settings]と[PSF Streaming Settings]を選択して、次のように設定します。

- Target Device：**R7FA6M3AH**（EK-RA6M3）を選択。
- RTT Control Block Address：**0x1FFE2FA4** を設定。図 52 を参照してください。

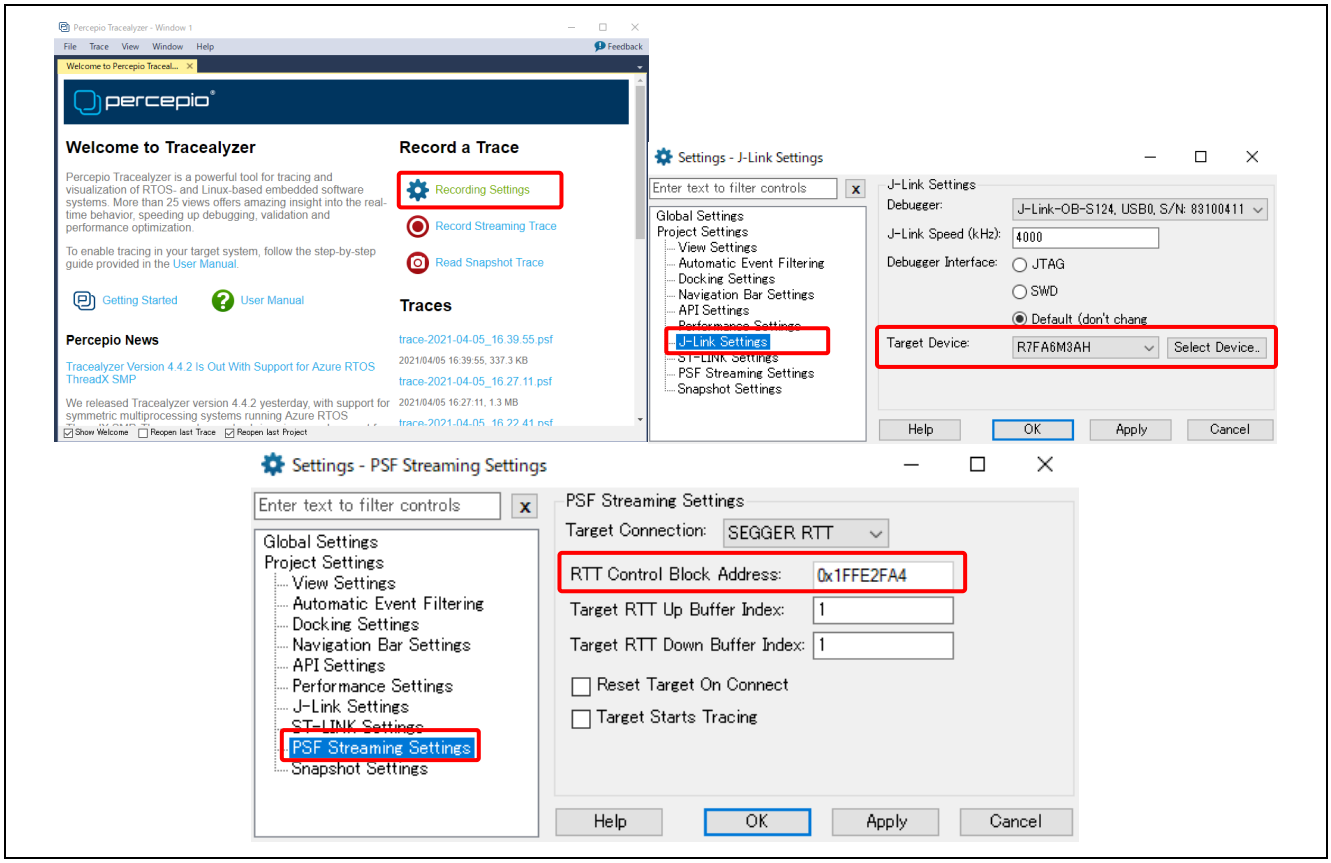


図 53　トレース記録に用いる J-Link RTT の設定

5.6.4　トレース記録の開始

[Record Streaming Trace]をクリックしてトレース記録を開始します。

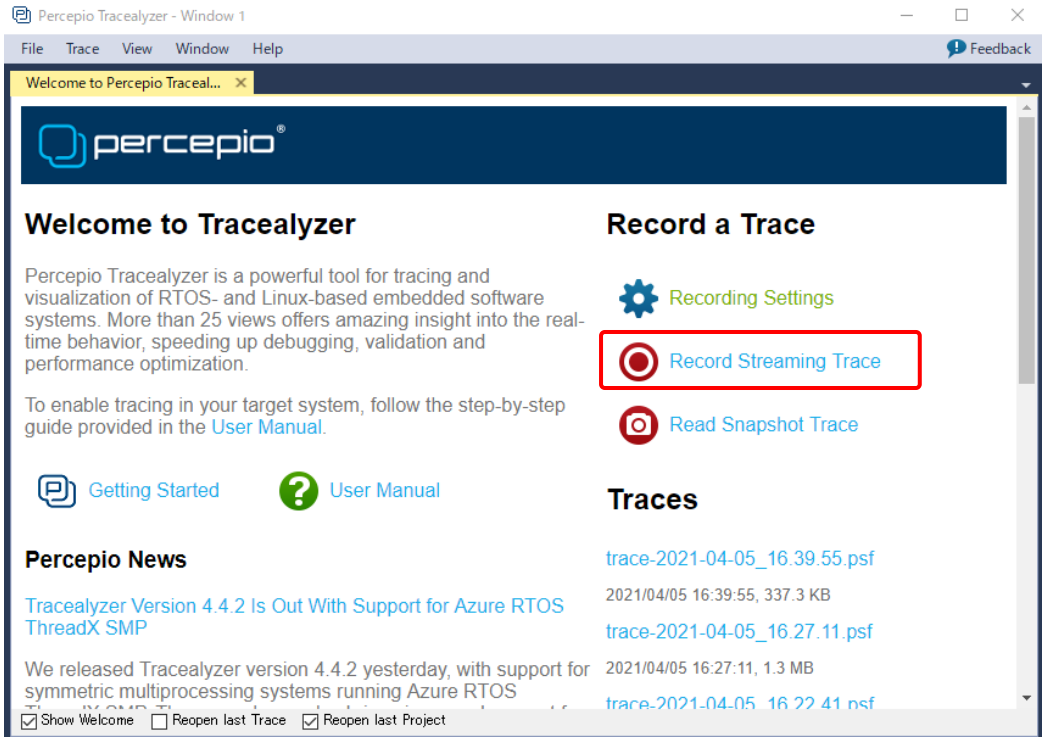


図 54　ストリーミングトレース記録の開始

5.6.5　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。

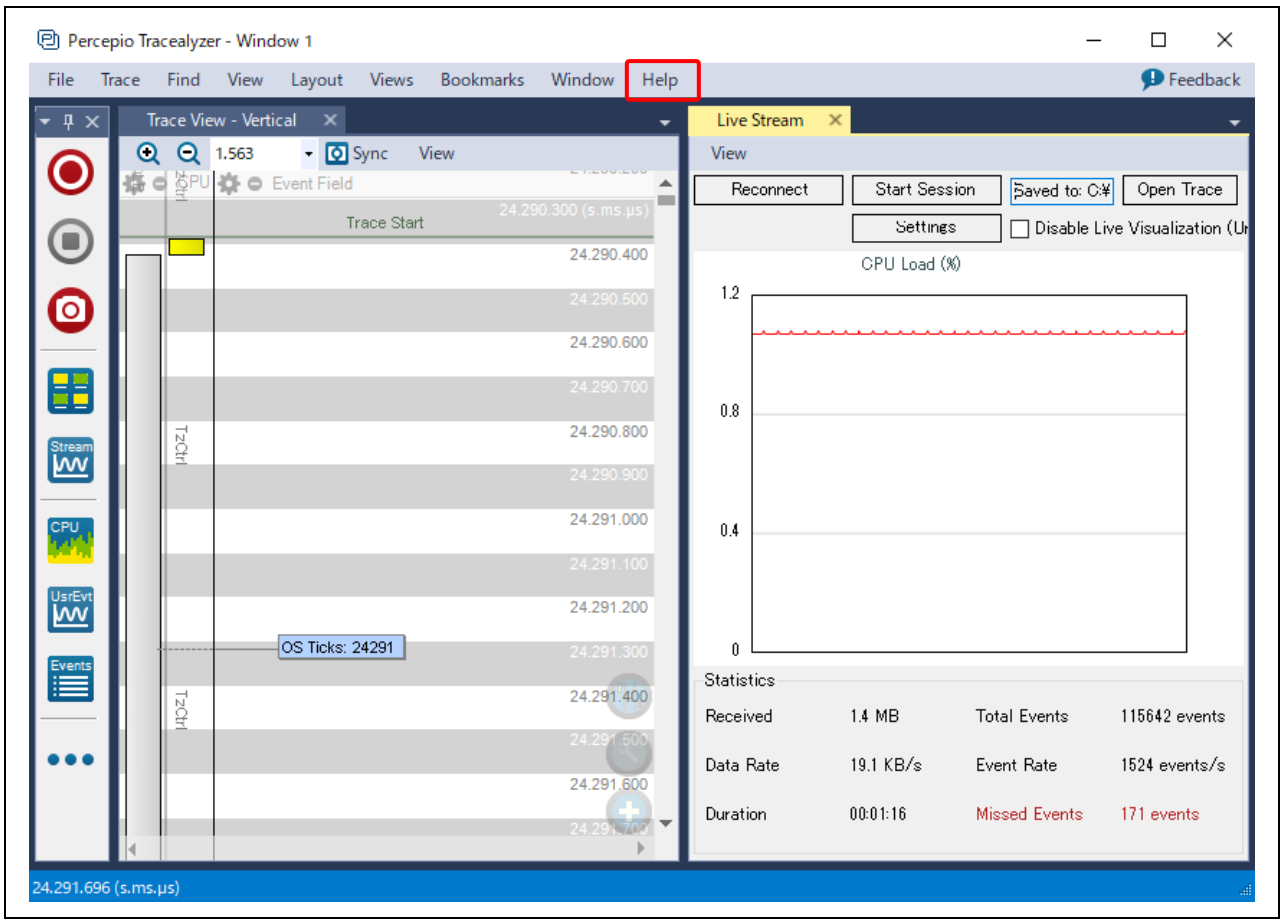


図 55　トレース情報の表示

ホームページとサポート窓口

RA 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。

RA ファミリ製品情報	www.renesas.com/ra
RA ファミリのサポートフォーラム	www.renesas.com/ra/forum
RA ファミリ Flexible Software Package	www.renesas.com/FSP
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	Sep.13.21	–	初版発行

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。