

## RZ/A2Mグループ

### RZ/A2M Software Package クイックスタートガイド

---

#### 1. 要旨

ルネサス製の RZ/A2M CPU ボード及び RZ/A2M SUB ボードと e2 studio 環境で動作する RZ/A2M Software Package のクイックスタートガイドになります。本書では、このパッケージに含まれる各サンプルプロジェクトを実行する手順について記載しています。

## 2. 準備

### 2.1 ツール

RZ/A2M Software Package は、以下の環境で動作致します。ご確認ください。

ツール:

- IDE: e2 studio 2021-04 Windows 64-bit product version or later
- Tool Chain: GNU ARM Embedded Toolchain 6-2017-q2-update

IDE は以下のサイトから入手可能です。

<https://www.renesas.com/software-tool/e-studio>

Tool Chain は IDE に同梱されています。個別では以下のサイトから入手可能です。

<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads/6-2017-q2-update>

e2 studio のインストール方法は以下文書を参照ください。

— [e2 studio Integrated Development Environment User's Manual: Getting Started](#)

ターゲットボード:

RZ/A2M CPU board (RTK7921053C00000BE)

RZ/A2M SUB board (RTK79210XXB00000BE)

ブートローダー:

本パッケージには、ブートローダーがバイナリで提供されています。ソースコードを入手したい場合、以下サイトからダウンロードしてください。

<https://www.renesas.com/rza2m-evaluation-board-kit>

### 2.2 USB シリアルポート接続

RZ/A2M SUB ボードの CN5 は、USB シリアル変換 IC を介してシリアルポート接続を提供します。最初に接続した時、適切なドライバを PC が自動的に見つけてインストールします。シリアルポートに割り当てられた COM ポート番号は、Windows™ デバイスマネージャーで確認することが可能です。

### 2.3 シリアルターミナル

— シリアルターミナルソフト(PuTTY、HyperTerminal、Tera Term など)の設定を以下に記載します。

- ボーレート: 115200
- データビット: 8
- パリティ: 無し
- ストップビット: 1
- フロー制御: 無し
- COM ポート: Windows™ デバイスマネージャー参照

### 3. サンプルプロジェクトの立ち上げ

#### 3.1 開発環境へのインポート

本パッケージは、アーカイブファイルで提供しておりビルドする環境を e2 studio にインポートすることが可能です。本章の手順に従うことで、ユーザが各サンプルプロジェクトのビルド環境をインポートすることが可能です。

- パッケージを入手します。
- パッケージを展開します。
- 使用するプロジェクトの zip ファイルを短いパスのフォルダに展開します。
- スタートメニューから e2 studio を起動します。
- 各サンプルプロジェクトのサブディレクトリの上にあるトップディレクトリをワークスペースディレクトリに設定します(図 3-1)。

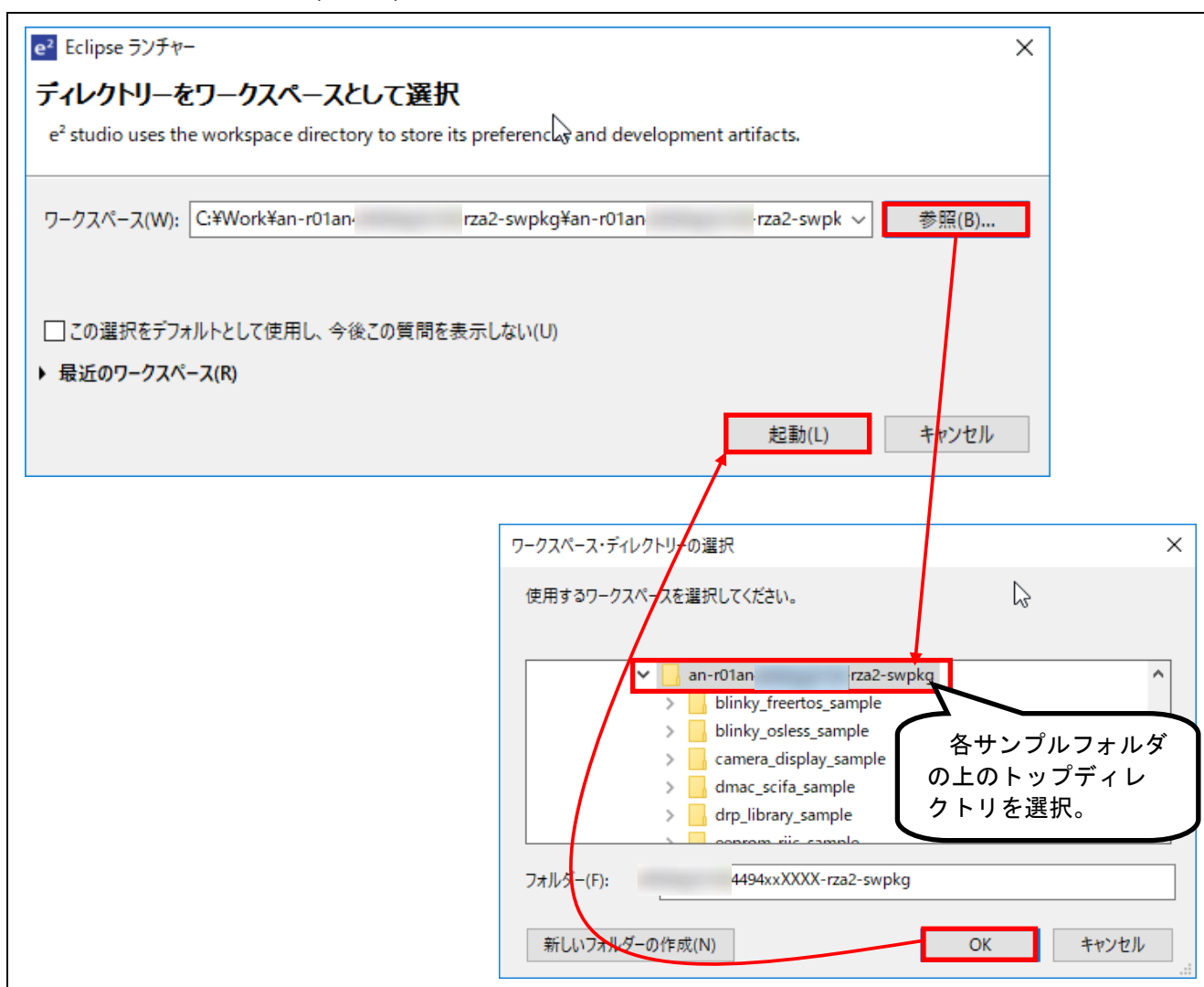


図 3-1 e2 studio の起動

— e2 studio の Welcome to e2 studio 画面で "ワークベンチ" をクリックします(図 3-2)。

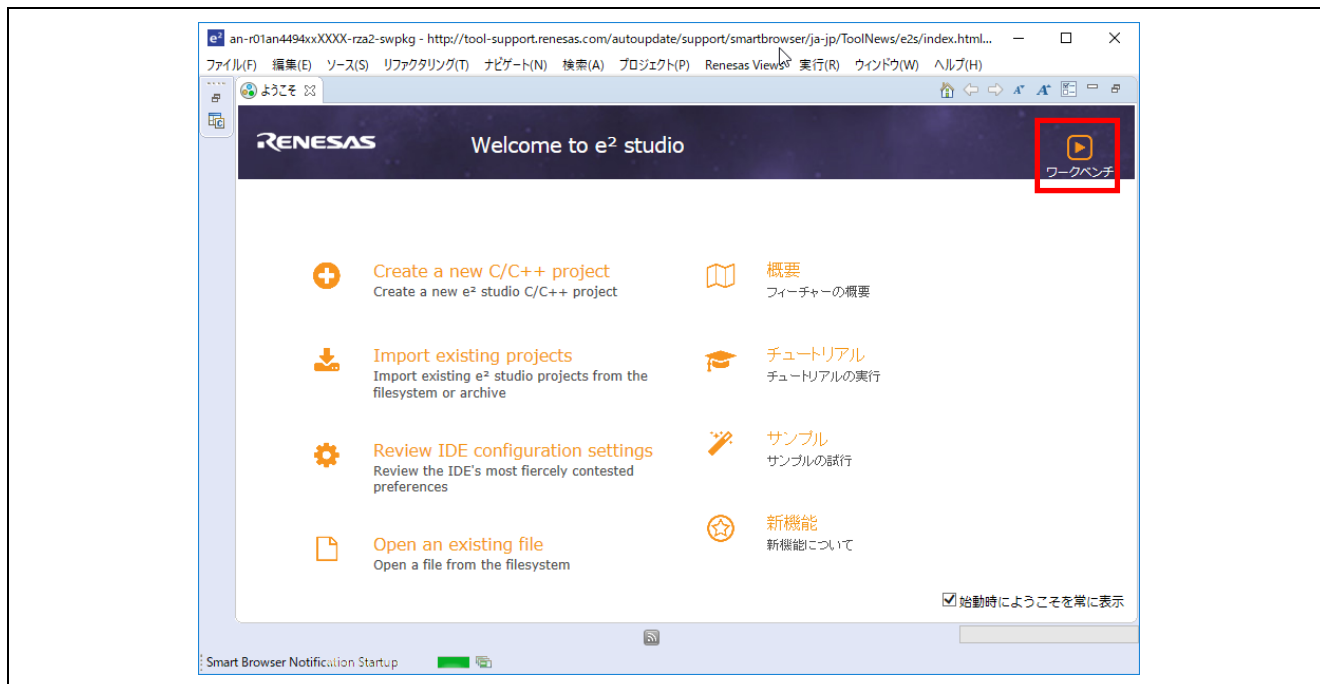


図 3-2 ワークベンチの切り替え

— プロジェクトエクスプローラーウィンドウで右クリックし "インポート" を選択します(図 3-3)。

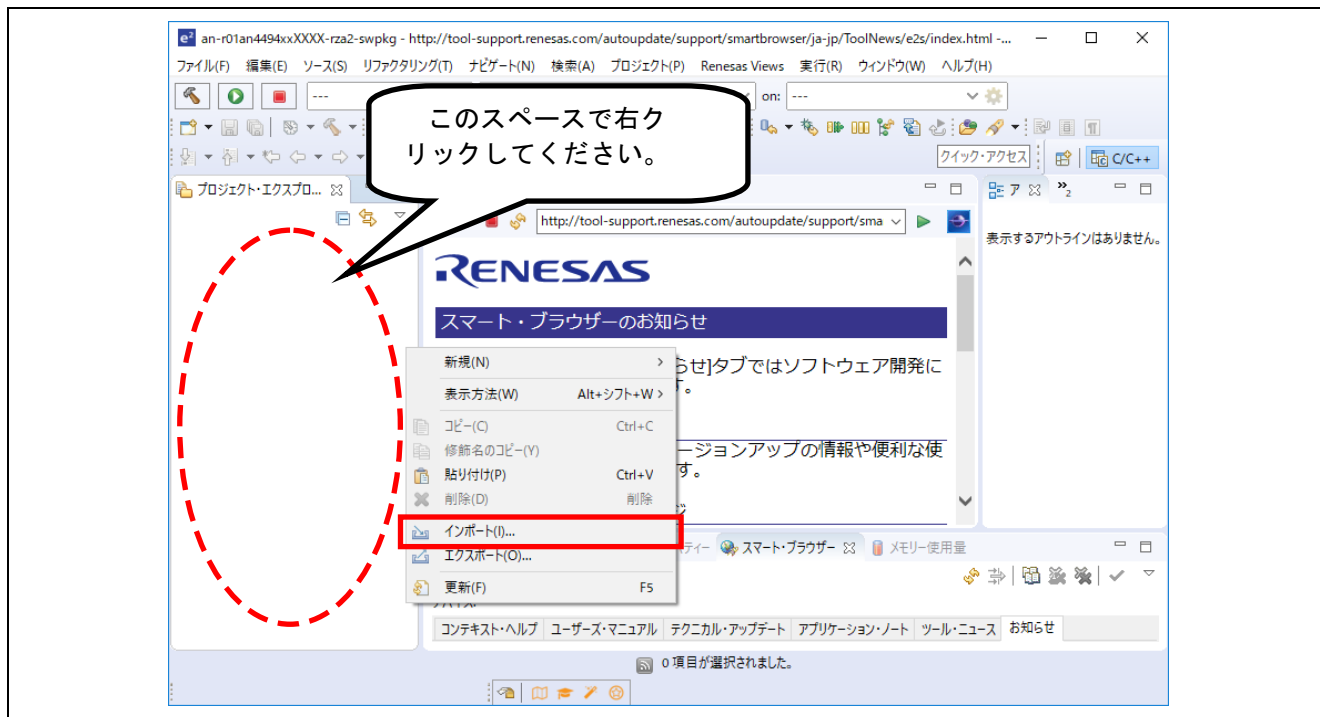


図 3-3 インポートの選択

- インポートダイアログで "一般" → "既存のプロジェクトワークスペースへ" を選択し、"次へ" のボタンをクリックします(図 3-4)。

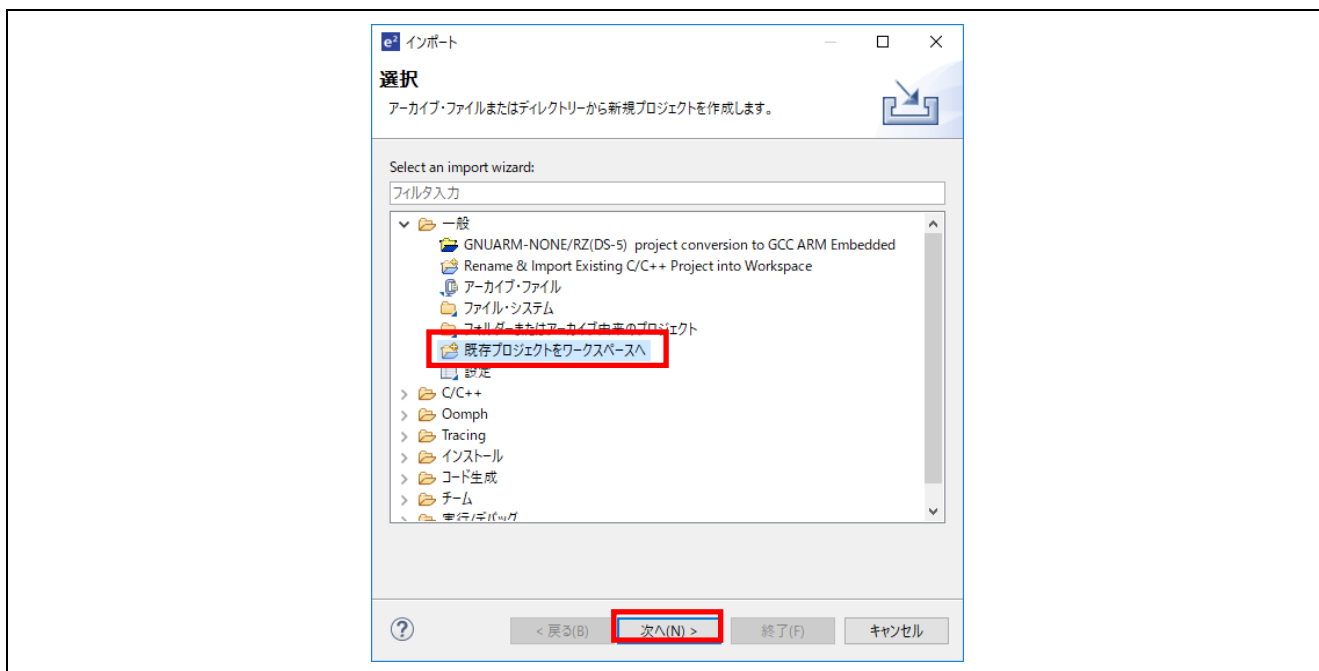


図 3-4 インポートダイアログ

- "ルートディレクトリの選択" の右にある "参照" ボタンを選択すると、"フォルダ参照" ダイアログが表示されます。そのまま "OK" ボタンを押してください(図 3-5)。

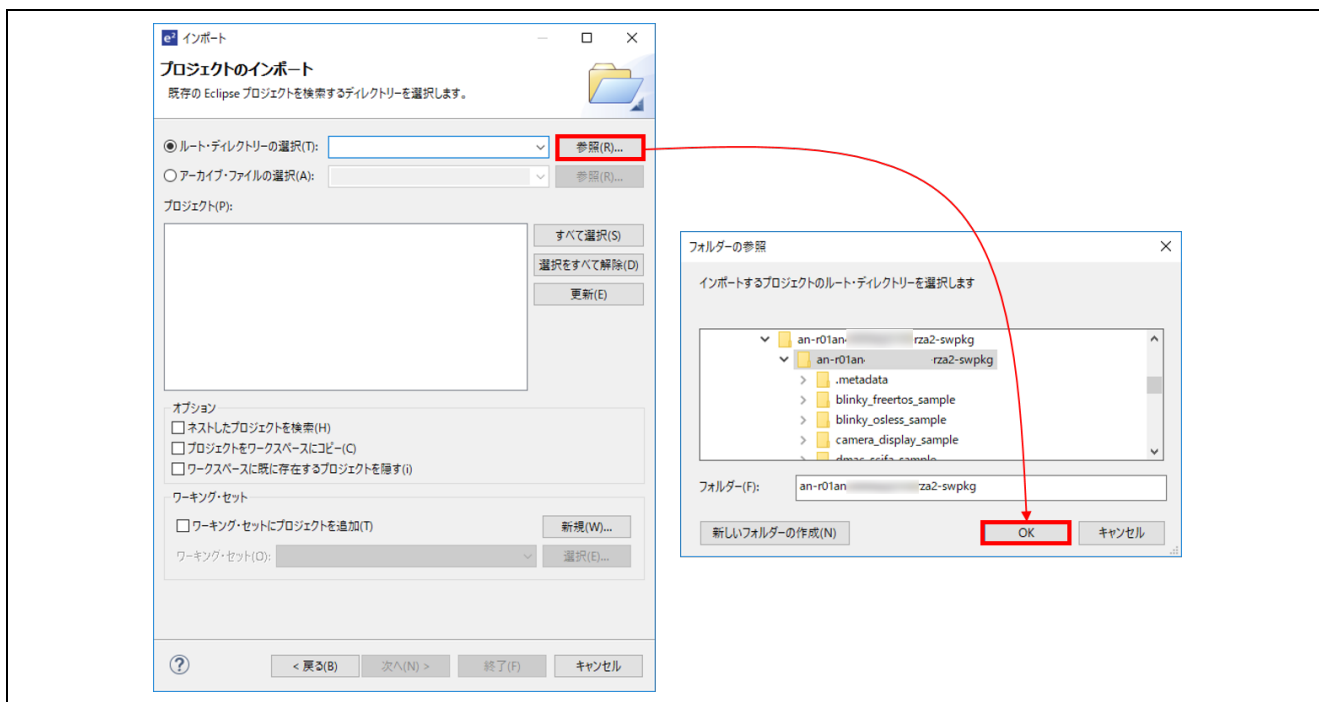


図 3-5 ルートディレクトリに選択

- ターゲットプロジェクトがチェックされていることを確認し "終了" ボタンをクリックします(図 3-6)。  
(注:次の図のプロジェクトは参考になります。実際にご使用になりたいターゲットプロジェクト名を選択してください。)

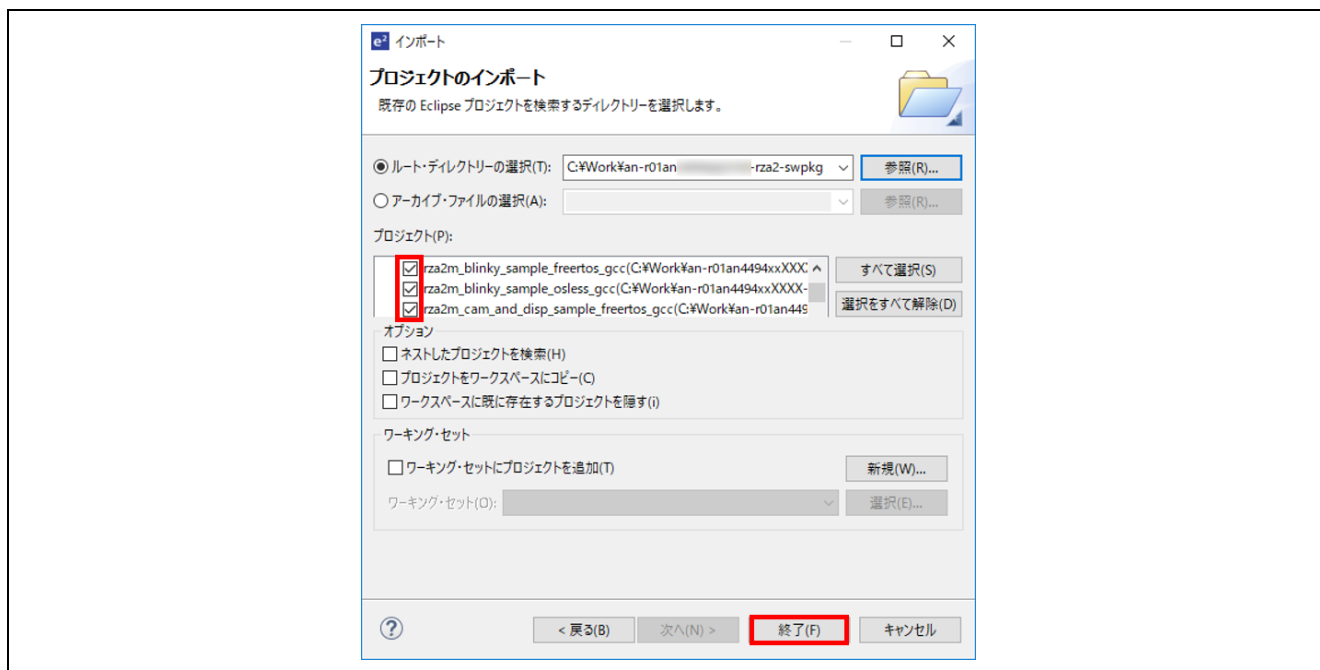


図 3-6 ターゲットプロジェクトのインポート

- ターゲットプロジェクトのインポートが完了すると、プロジェクトエクスプローラーウィンドウで確認することができます(図 3-7)。

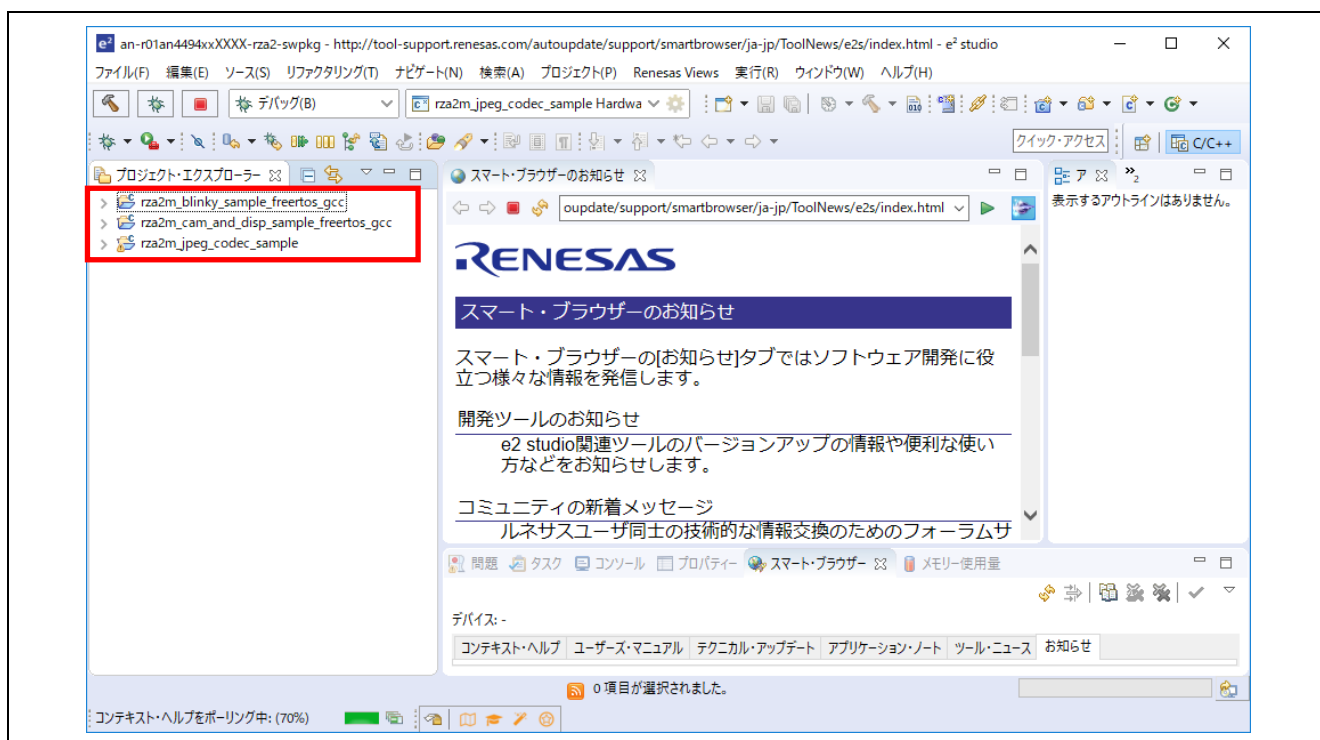


図 3-7 プロジェクトエクスプローラーウィンドウでの確認

### 3.2 プロジェクトのビルドと評価ボードへのダウンロード

- ターゲットプロジェクトを選択しビルドボタン(トンカチアイコン)の横にある矢印をクリックし、ドロップダウンメニューから"HardwareDebug"を選択するとビルドが開始されます(図 3-8)。次回からは、ビルドボタン(トンカチアイコン)でビルドが可能になります。

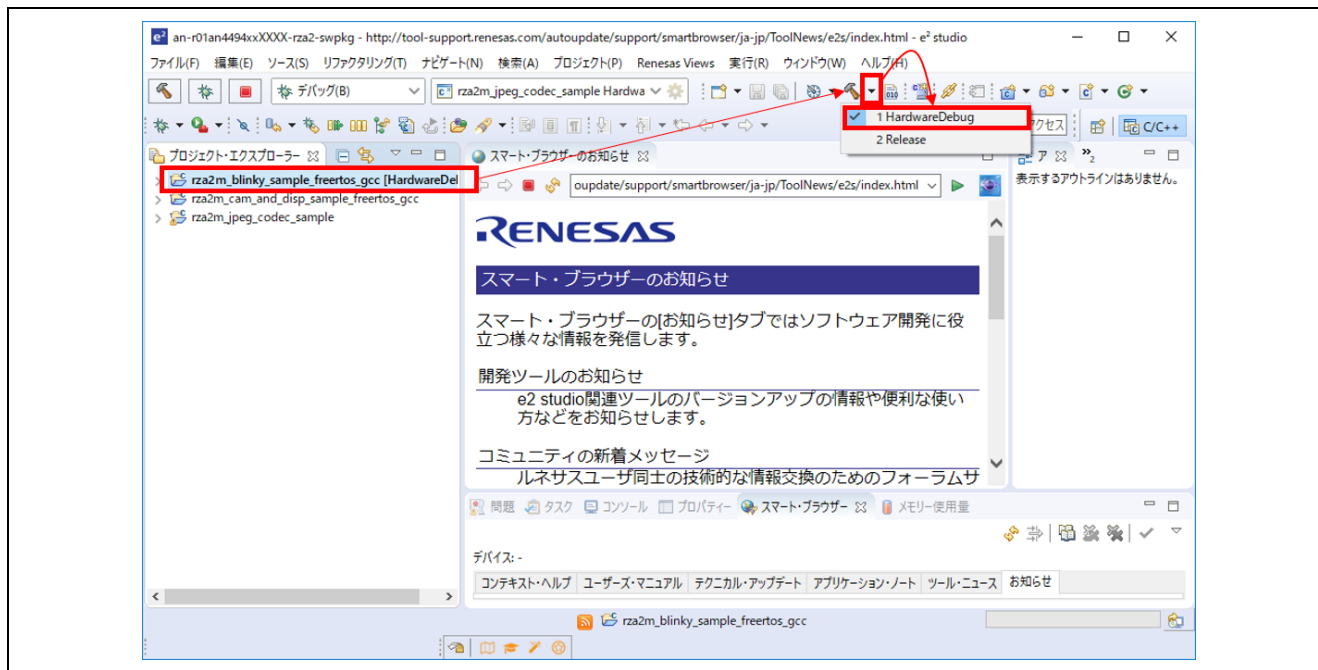


図 3-8 ターゲットプロジェクトのビルド

- ビルドが開始されコンソールウィンドウでそのステータスを確認することができます(図 3-9)。(注:ワークスペースフォルダまでのパスの長さに注意してください。パスが長すぎるとビルドエラーになる場合があります。)

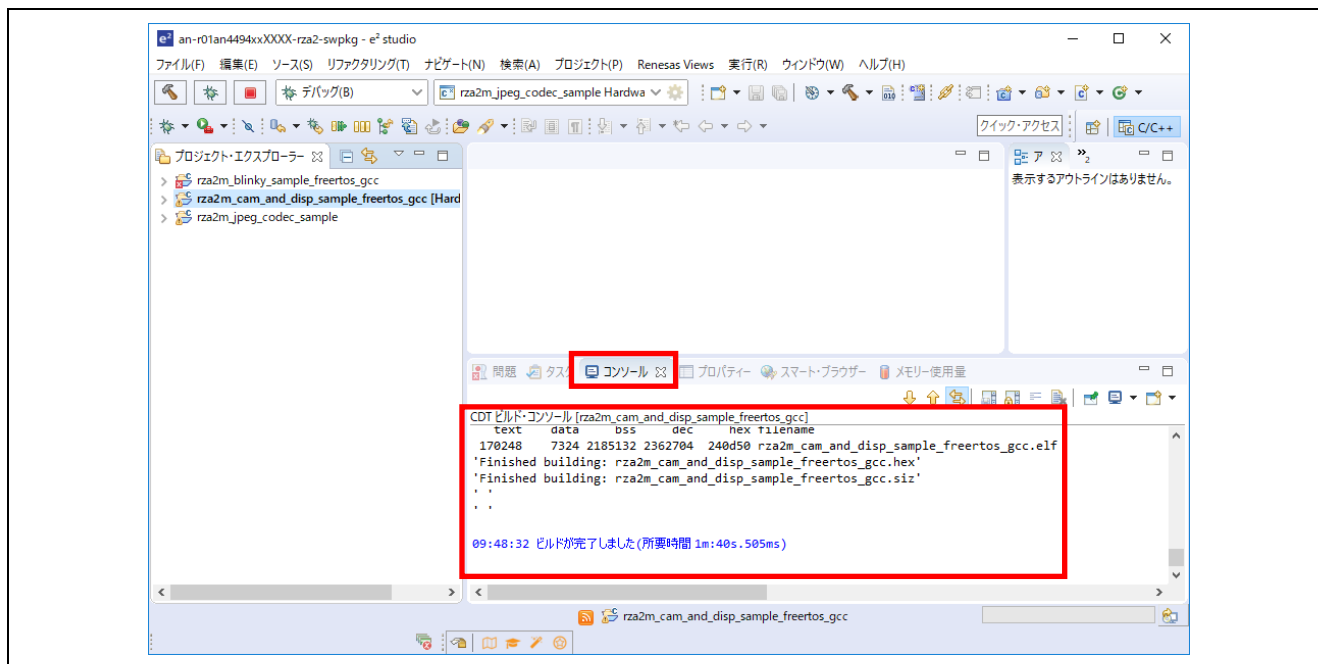


図 3-9 ビルドステータスの確認

- ビルドが完了したら、デバッグボタン(虫アイコン)の横にある "矢印" をクリックしドロップダウンリストの "デバッグの構成" を選択することで、デバッグの構成ダイアログを開きます。"Renesas GDB Hardware Debugging"にある各ターゲットプロジェクトを選択し "デバッグ" ボタンをクリックするとデバッグが開始されます(図 3-10)。次回からは、デバッグボタン(虫アイコン)のクリックのみでデバッグが開始可能です。

(注: ターゲットプロジェクトがプロジェクトエクスプローラーウィンドウで選択されていることを確認してください。また、ビルドが正常に終了していない場合、"デバッグ" ボタンが有効になりません。)

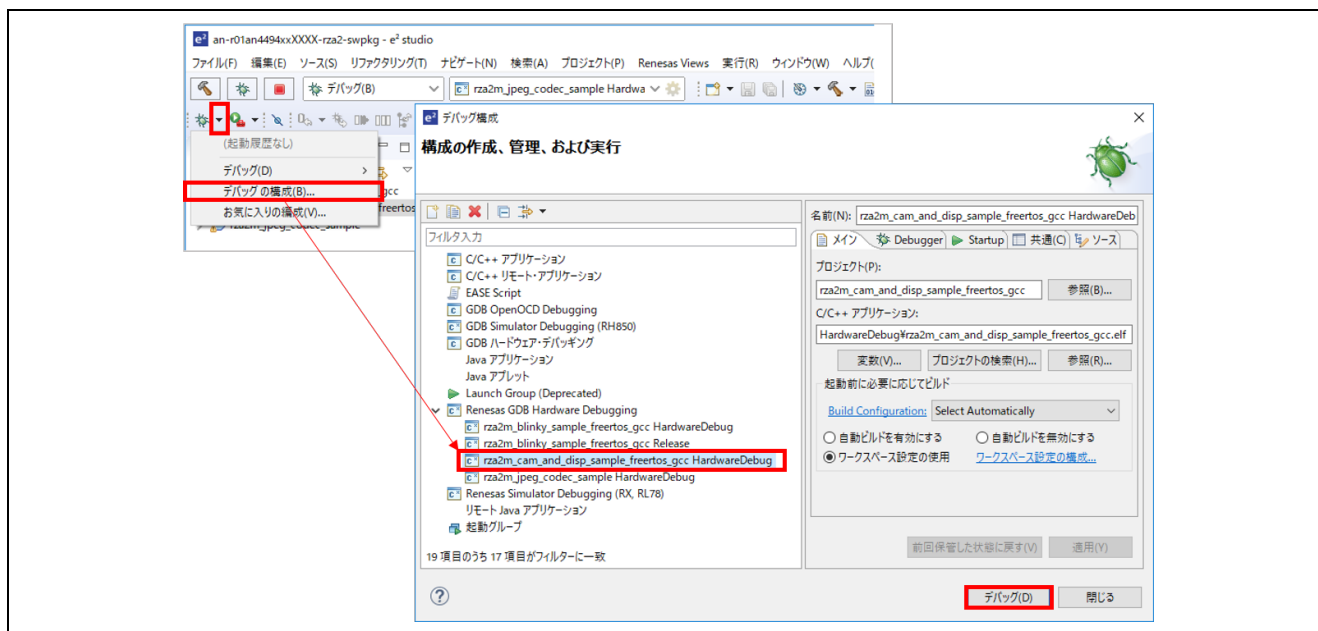


図 3-10 デバッグの開始

- ターゲットのプログラムがダウンロードされ、e2 studio から "パースペクティブの切り替え確認"ダイアログが開きます。"はい" ボタンをクリックします(図 3-11)。

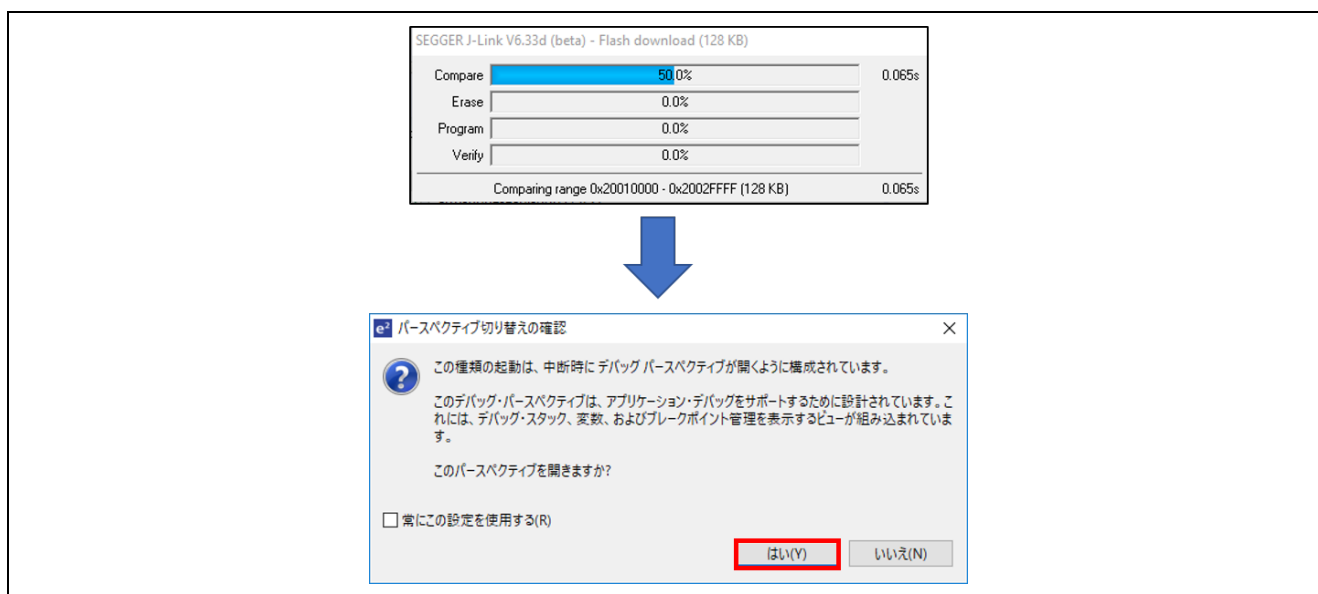


図 3-11 ターゲットプログラムのダウンロード



- プログラムのダウンロードが完了したら "再開" ボタンをクリックしてターゲットプログラムを実行してください(図 3-12)。

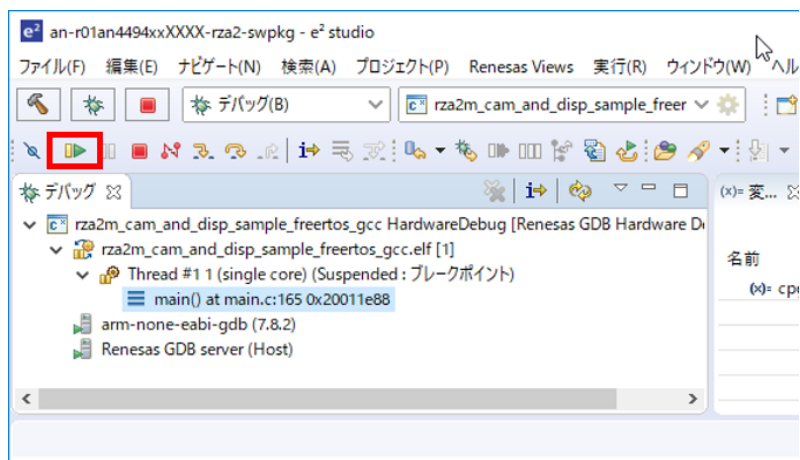


図 3-12 ターゲットプログラムの実行

- 次の図のようにターゲットプロジェクトの"デバッグ構成"ダイアログのブレークポイント設定先に main が設定されている場合、main 関数の先頭でブレークします。このようなターゲットプロジェクトでは "再開" ボタンをクリックして実行を再開してください(図 3-13)。

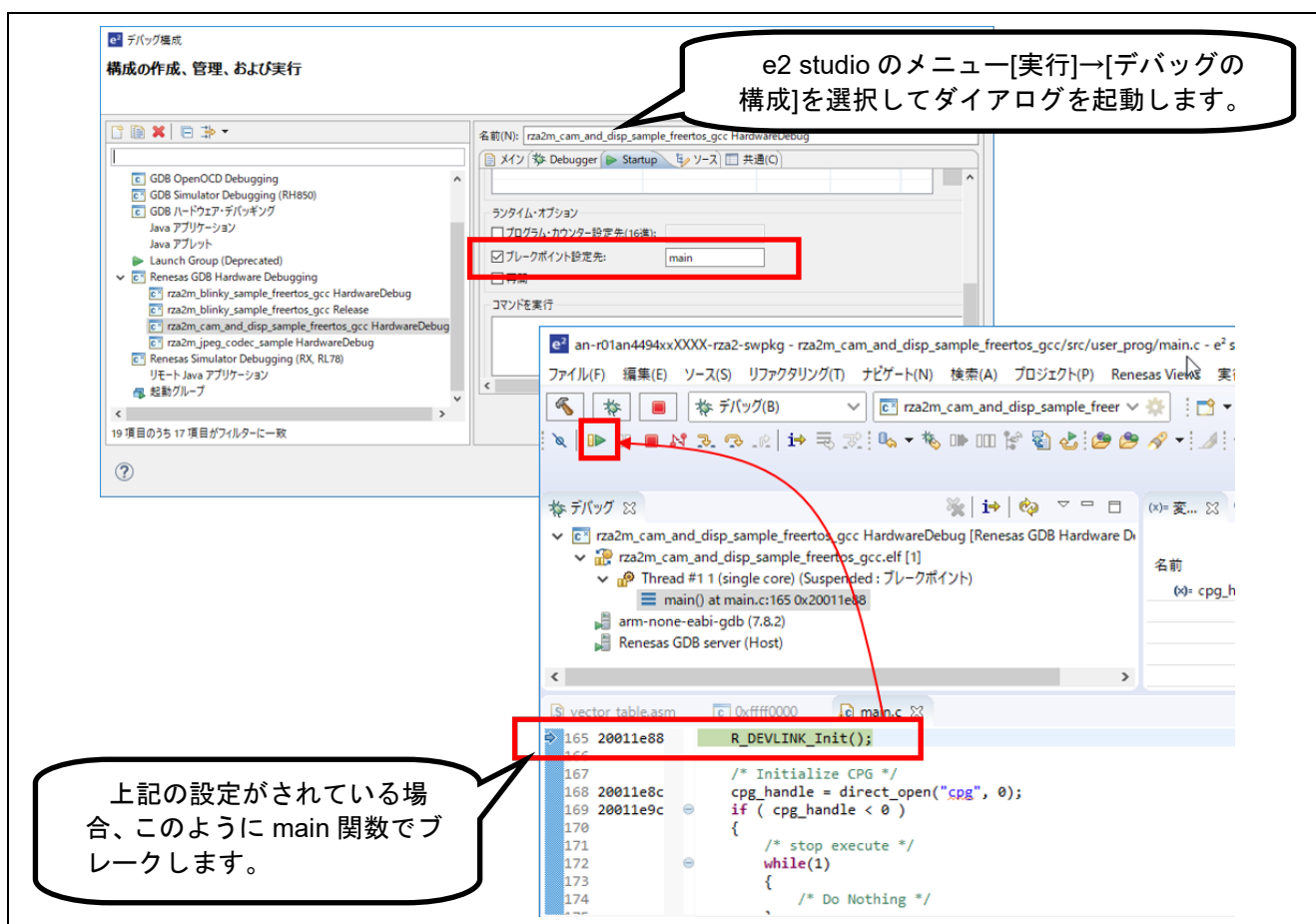


図 3-13 実行後のブレーク

#### 4. ドライバ、ミドルウェアの追加

本章では、本パッケージ同梱のプロジェクトに対して、ドライバ、ミドルウェアを追加する方法を記載します。

RZ/A2M Software Package では、ドライバ、ミドルウェアをコンポーネントとして管理しており、e2 studio 上からコンポーネントを追加することができます。

各ドライバ、ミドルウェアの使用例は、RZ/A2M Simple Applications Package(R01AN4494)の各サンプルプロジェクトをご参照ください。

各ドライバ、ミドルウェアを e2 studio にインストールする方法など、スマート・コンフィギュレータの使用法は、[RZ/A2M スマート・コンフィギュレータ ユーザーガイド: e2 studio編\(R20AN0583\)](#)を参照ください。

4.1 コンポーネントとサンプルプロジェクトの関連

各サンプルプロジェクトで使用されているコンポーネントを以下の表に示します。

Package	Component (説明)	Sample Program (名称)																															
			sillex (Sillex Wi-Fi ライブラリ)	fatts (FatFS フォイルシステム)	r_vdc (VDC6 フォイ)	r_usb_cdc (USB CDC フォイ)	r_usb_basic (USB 基本 フォイ)	r_usb1_msc (USB 基本 フォイ)	r_usb1_basic (USB 基本 フォイ)	r_usb1_hidc (USB 基本 HID フォイ)	r_usb0_msc (USB 基本 MSC フォイ)	r_usb0_basic (USB 基本 フォイ)	r_usb0_hidc (USB 基本 HID フォイ)	r_ssif (SSIF フォイ)	r_sdhi_simplified (SDHI フォイ)	r_scifa (SCiFa フォイ)	r_nvapi (Video ユーティリティ)	r_rtc (RTC フォイ)	r_rspi (RSPi フォイ)	r_ric (RIIC フォイ)	r_rga (グラフィックライブラリ"RGA")	r_ostn (OS タイマ フォイ)	r_mipi (MIPI フォイ)	r_lpm (LPM フォイ)	r_jcu (JCU フォイ)	r_hyperbus (Hyperbus フォイ)	r_gpt (GPT フォイ)	r_ether (Ethernet フォイ)	r_drp (DRP フォイ)	r_dmac (DMAC フォイ)	r_ceu (CEU フォイ)	r_cbuffer (バッファ)	r_adc (ADC フォイ)
simple applications	adc <sup>*3</sup>	✓	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	blinky_freertos <sup>*1</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	blinky_osless <sup>*2</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	cam_and_disp <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	dmac_scifa <sup>*3</sup>	-	✓	-	✓	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_basic <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_dynamic1 <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_dynamic2 <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_dynamic3 <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_parallel <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	drp_simple_isp_sample1 <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	eeeprom_riic <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	
	ethernet <sup>*3</sup>	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	fw_update_boot <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	fw_update_sample <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	gpt-pwm <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	lpm <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	jpeg_codec <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	rtc <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	sdhi_fat <sup>*3</sup> <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	sprite_engine <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	ssif <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	touch_panel <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	usbh_msc_fat <sup>*3</sup> <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	usbh_hid <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	usbf_cdc <sup>*3</sup> <sup>*4</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
	wifi_pmod_esp32 <sup>*3</sup>	✓	✓	-	-	-	✓	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
2d_barcode	2d_barcode <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
iris	iris <sup>*3</sup>	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
Graphics RGA	graphics <sup>*3</sup>	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-
Wi-Fi	wifi_sx_sdmac_eval <sup>*3</sup>	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	✓	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	✓
	wifi_sx_sdmac <sup>*3</sup>	✓	✓	-	-	-	✓	-	-	✓	✓	-	-	-	✓	✓	✓	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	✓

- 【注】
- ✓

: コンポーネントが使用されている
- : コンポーネントは使用されていない
- \*1

: e<sup>2</sup> studio 上に表示されるプロジェクト名は「rza2m\_blinky\_sample\_freertos\_gcc」
- \*2

: e<sup>2</sup> studio 上に表示されるプロジェクト名は「rza2m\_blinky\_sample\_osless\_gcc」
- \*3

: e<sup>2</sup> studio 上に表示されるプロジェクト名は「rza2m\_[表記載の名称]\_sample\_freertos\_gcc」
- \*4

: e<sup>2</sup> studio 上に表示されるプロジェクト名は「rza2m\_[表記載の名称]\_sample\_osless\_gcc」

## 4.2 コンポーネントの追加方法

- e2 studio のプロジェクトエクスプローラーウィンドウで、対象プロジェクトのプロジェクトツリーを開き、その中にある.scfg ファイルをダブルクリックします。
- コンポーネントタブに移動し、コンポーネントの追加ボタンから、対象のコンポーネントを追加します。
- 対象のコンポーネントを追加後、コードの生成ボタンをクリックします。

以上で、対象プロジェクトの generate¥sc\_drivers と .settings¥smartconfigurator にコンポーネントが追加されます(図 4-1)。コンポーネント追加後は、3.2章に示す手順に従い、ビルドを行ってください。

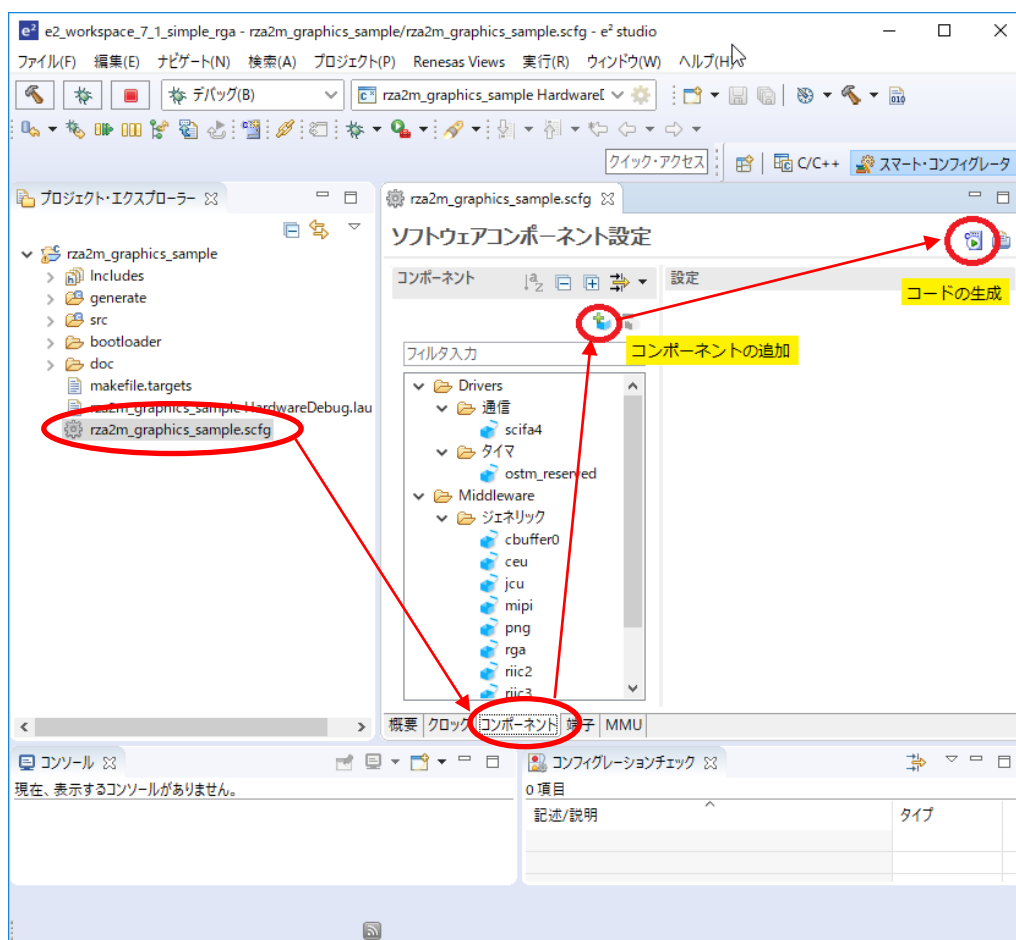


図 4-1 コンポーネントの追加方法

### 4.3 各コンポーネントの実装方法

各コンポーネントのプロジェクトへの実装は以下の手順で行います。

- 4.2章の方法でコンポーネントの追加
- Smart Configurator によるコンポーネントの設定
- コンポーネントのコード生成
- コンポーネントのドキュメントで API 関数名や API 記載ファイル名を確認
- コンポーネント使用箇所を API 関数名や API 記載ファイル名で検索
- 検索した使用箇所を参考にプロジェクトに実装する

一例として `rza2m_blinky_sample_freertos_gcc` に `RIIC3` を用いた `EEPROM` アクセスを追加する例を示します。

- rza2m\_blinky\_sample\_freertos\_gcc プロジェクト内の rza2m\_blinky\_sample\_freertos\_gcc.scfg ファイルを開き、4.2章の方法で RIIC3 を追加します。
- riic3 コンポーネントの設定を行います。RZ/A2M Evaluation Board Kit に実装されている EEPROM を使用する場合は、以下の設定となります(図 4-2)。
  - Clock Frequency を 400KHz に変更
  - RIIC3 を Used に変更
  - RIIC3SCL Pin を Used に変更
  - RIIC3SDA Pin を Used に変更

Property	Value
▼  Configurations	
# RIIC Transfer Mode	Master Mode
# Own Slave Address0 Enable	<input checked="" type="checkbox"/> Used
# Own Slave Address1 Enable	<input type="checkbox"/> Unused
# Own Slave Address2 Enable	<input type="checkbox"/> Unused
# Own Slave Address0	1
# Own Slave Address1	1
# Own Slave Address2	1
# Own Slave Address0 Length	7bit Address Mode
# Own Slave Address1 Length	7bit Address Mode
# Own Slave Address2 Length	7bit Address Mode
# SCL Clock Frequency	400KHz
# SCL Duty Cycle	50%
# SCL Rise Time (nsec)	0
# SCL Fall Time (nsec)	0
# Noise Filter	Unused
# SCL Timeout Mode	Unused
# RIIC Bus Format	I2C Format
# SMBus Host address detection	<input type="checkbox"/> Unused
# TEND Interrupt priority level (0 - 31)	9
# RDRF Interrupt priority level (0 - 31)	9
# TDRE Interrupt priority level (0 - 31)	9
# STOP Interrupt priority level (0 - 31)	9
# START Interrupt priority level (0 - 31)	9
# NACKF Interrupt priority level (0 - 31)	9
# AL Interrupt priority level (0 - 31)	9
# TMOF Interrupt priority level (0 - 31)	9
▼  Resources	
▼  RIIC	
▼  RIIC3	<input checked="" type="checkbox"/>
RIIC3SCL Pin	<input checked="" type="checkbox"/> Used
RIIC3SDA Pin	<input checked="" type="checkbox"/> Used

図 4-2 コンポーネントの設定

— Generate ボタン(図 4-3)を押して、コードを生成します。

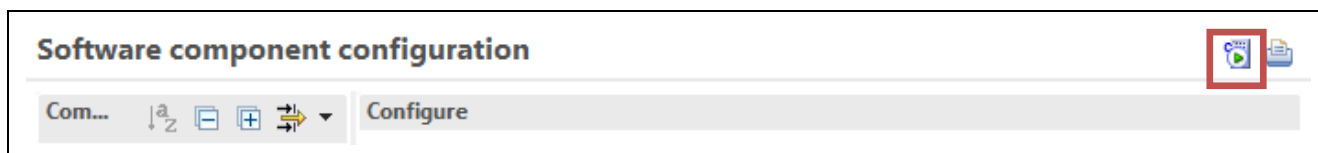


図 4-3 コードの生成

— Console ビューに生成されたコードが表示されます。riic ドライバのドキュメントが generate¥sc\_drivers¥r\_riic¥doc 以下に生成されたことが分かります(図 4-4)。

```
M00000001: Code generation is started
M03000002: File generated:generate\drivers\r_cpg\inc\r_cpg_drv_sc_cfg.h
M04020001: File generated:generate\sc_drivers\r_riic\doc\r01an4645eg0100-rza2-riic-driver.pdf
M04020001: File generated:generate\sc_drivers\r_riic\inc\r_riic_drv_api.h
M04020001: File generated:generate\sc_drivers\r_riic\inc\r_riic_drv_link.h
M04020001: File generated:generate\sc_drivers\r_riic\inc\r_riic_drv_sc_cfg.h
M04020001: File generated:generate\sc_drivers\r_riic\inc\r_riic_hld_api.h
M04020001: File generated:generate\sc_drivers\r_riic\inc\r_riic_hld_prv.h
```

図 4-4 生成されたコードのリスト

— riic ドライバのドキュメントを読むと、ドライバの API が r\_riic\_drv\_api.h に記載されていることが分かります。

— 4.1章の表から、RIIC ドライバは rza2m\_eeprom\_riic\_sample\_freertos\_gcc で使用されていることが分かります。rza2m\_eeprom\_riic\_sample\_freertos\_gcc を e² studio に import します。

— rza2m\_eeprom\_riic\_sample\_freertos\_gcc 内で r\_riic\_drv\_api.h を#include している箇所を探します。または、ドライバ API の使用箇所を探します。  
「Search」メニューの「File...」を選択します(図 4-5)。

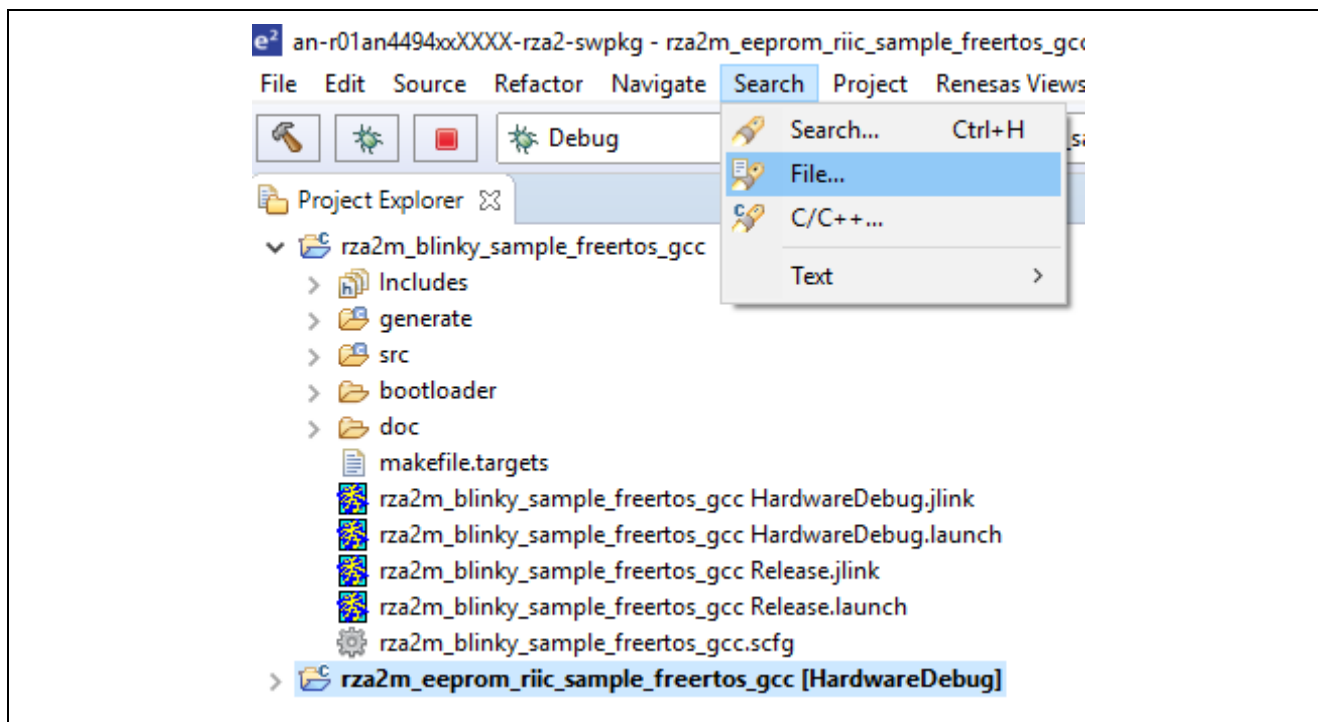


図 4-5 r\_riic\_drv\_api.h を#include されている箇所の検索 (1)

— Containing text:に「r\_riic\_drv\_api.h」と入力し、「Search」を押下します(図 4-6)。

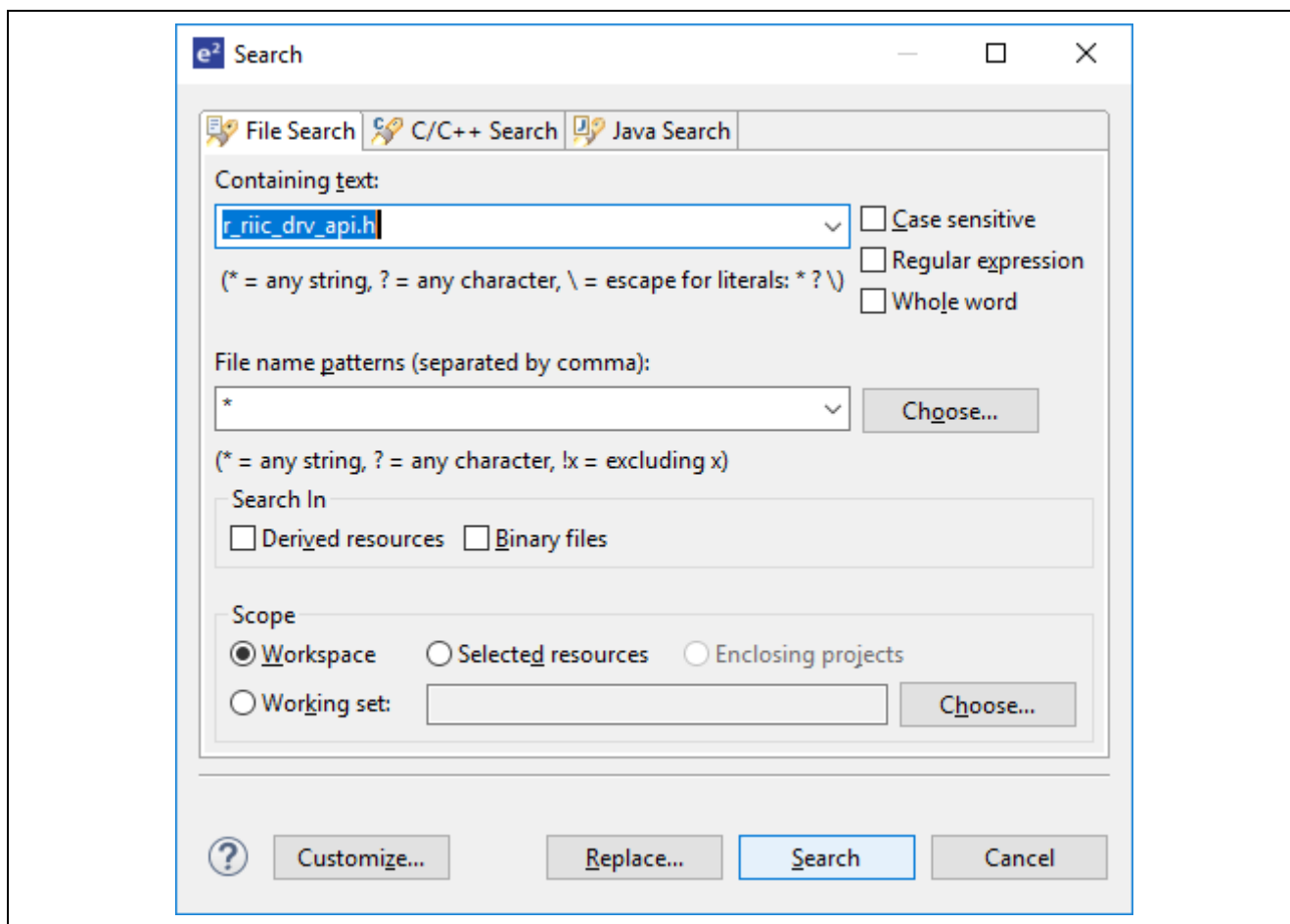


図 4-6 r\_riic\_drv\_api.h を#include されている箇所の検索 (2)

— rza2m\_eeprom\_riic\_sample\_freertos\_gcc プロジェクトの  
src¥renesas¥application¥r\_eeprom\_sample.cでr\_riic\_drv\_api.hが#include されていることが分かります(図 4-7)。  
r\_eeprom\_sample.c をダブルクリックして開きます。

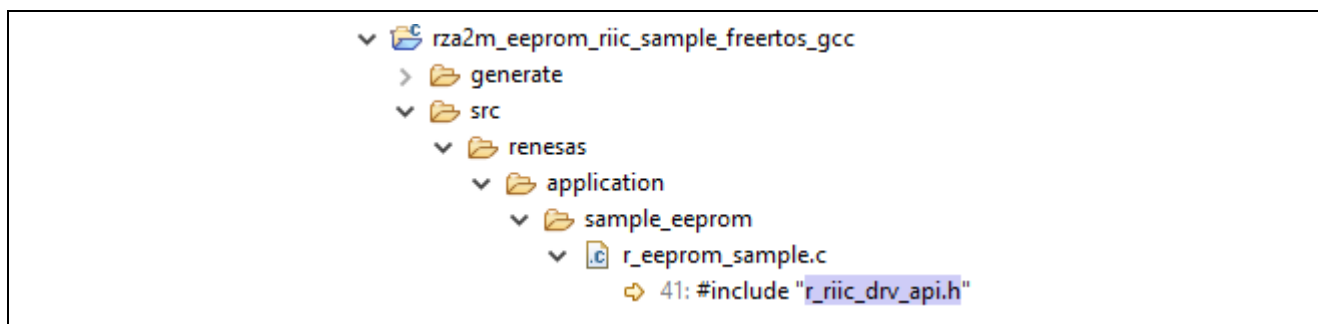


図 4-7 r\_riic\_drv\_api.h を#include されている箇所の検索 (3)



— r\_eeeprom\_sample.c に、open によるドライバの初期化、close によるドライバの終了、control の引数によるデータの読み書きの例が記載されています(図 4-8)。

```
/* RIIC driver handle */
static int_t gs_handle;

⊕ Exported global variables and functions (to be accessed)
⊕ * Function Name: sample_riic_eeeprom_init
⊖ int_t sample_riic_eeeprom_init(void)
{
    int_t ret = NO_ERROR;

    gs_handle = open(RIIC_CH_EEPROM_RZA2M, O_RDWR);

    ⊖ if (0 > gs_handle)
    {
        ret = ERROR_FAILURE;
    }

    /* Successful initialisation */
    return ret;
}
```

```
⊖ int_t sample_riic_eeeprom_uninit(void)
{
    close(gs_handle);

    return NO_ERROR;
}
```

```
/* EEPROM page write */
st_r_drv_riic_transfer_t i2c_write;
i2c_write.device_address = EEPROM_DEV_ADDRESS;
i2c_write.sub_address_type = RIIC_SUB_ADDR_WIDTH_16_BITS;
i2c_write.sub_address = write_addr;
i2c_write.number_of_bytes = write_size;
i2c_write.p_data_buffer = p_data;

drv_ret = control(gs_handle, CTL_RIIC_WRITE, &i2c_write);
```

```
st_r_drv_riic_transfer_t i2c_dummy_read;
i2c_dummy_read.device_address = EEPROM_DEV_ADDRESS;
i2c_dummy_read.sub_address_type = RIIC_SUB_ADDR_WIDTH_16_BITS;
i2c_dummy_read.sub_address = dummy_addr;
i2c_dummy_read.number_of_bytes = dummy_size;
i2c_dummy_read.p_data_buffer = dummy_buf;

error_cnt = 0;
while (1)
{
    /* wait for ACK from EEPROM */
    /* Since NACK is returned until writing is completed, an error occurs */
    drv_ret = control(gs_handle, CTL_RIIC_READ, &i2c_dummy_read);
    if (drv_ret != NO_ERROR)
    {
        error_cnt++;
    }
}
```

図 4-8 r\_eeeprom\_sample.c 内の riic ドライバ使用例

— r\_eeeprom\_sample.c を参考に、rza2m\_blinky\_sample\_freertos\_gcc の main.c を変更します。

図 4-9に rza2m\_blinky\_sample\_freertos\_gcc の main.c に追加した処理を示します。

図 4-10に変更後のプロジェクトを実行した時のデバッグ出力を示します。

```
int_t gs_handle;

/* Open RIIC driver */
gs_handle = open(RIIC_CH_EEPROM_RZA2M, O_RDWR);

if (0 <= gs_handle)
{
    int_t drv_ret = DRV_SUCCESS;
    /* EEPROM page0 write 1byte, 0x5a */
    st_r_drv_riic_transfer_t i2c_write;
    const uint8_t data = 0x5a;
    i2c_write.device_address = EEPROM_DEV_ADDRESS;
    i2c_write.sub_address_type = RIIC_SUB_ADDR_WIDTH_16_BITS;
    i2c_write.sub_address = 0;
    i2c_write.number_of_bytes = 1;
    i2c_write.p_data_buffer = &data;

    drv_ret = control(gs_handle, CTL_RIIC_WRITE, &i2c_write);
    if (DRV_ERROR != drv_ret)
    {
        st_r_drv_riic_transfer_t i2c_read;
        uint8_t result=0;
        /* wait 100ms */
        R_OS_TaskSleep(100);
        /* EEPROM page0 read 1byte */
        i2c_read.device_address = EEPROM_DEV_ADDRESS;
        i2c_read.sub_address_type = RIIC_SUB_ADDR_WIDTH_16_BITS;
        i2c_read.sub_address = 0;
        i2c_read.number_of_bytes = 1;
        i2c_read.p_data_buffer = &result;

        drv_ret = control(gs_handle, CTL_RIIC_READ, &i2c_read);
        if (DRV_ERROR != drv_ret)
        {
            printf ("EEPROM success data = %02X\n",result);
        }
    }
    /* Close RIIC driver */
    close(gs_handle);
}
```

図 4-9 rza2m\_blinky\_sample\_freertos\_gcc の main.c に追加した処理

```
RZ/A2M blinky_sample for GCC Ver. 3.1
Copyright (C) 2018 Renesas Electronics Corporation. All rights reserved.
Build Info Date Mar 19 2019 at 21:49:31
FreeRTOS OS Abstraction Version 3.0

SAMPLE> EEPROM success data = 5A
```

図 4-10 rza2m\_blinky\_sample\_freertos\_gcc の出力

## 5. FreeRTOS プロジェクトの作成方法

FreeRTOS を使用する新しいプロジェクトを作成する場合は、目的に応じて以下の作成方法を選択してください。

Amazon AWS を使用する場合：5.1を参照してください。

Amazon AWS を使用しない場合：5.2を参照してください。

### 5.1 Amazon AWS を使用する FreeRTOS プロジェクトを作成する方法

e<sup>2</sup>studio の C/C++プロジェクト新規作成機能を使用してプロジェクトを作成してください。

手順の詳細については、以下のドキュメントを参照してください。

<https://www.renesas.com/document/qsg/getting-started-renesas-rza2m-evaluation-board-kit-0?language=en&r=1305306>

### 5.2 Amazon AWS を使用しない FreeRTOS プロジェクトを作成する方法

e<sup>2</sup>studio によって新規作成された FreeRTOS プロジェクトには、常に Amazon AWS が含まれています。

FreeRTOS カーネルのみを使用する場合は、既存のサンプルプロジェクトをインポートして開発を開始することをお勧めします。

以下のプロジェクトをベースとして使用してください。

RZ/A2M Simple Applications Package(R01AN4494)の rza2m\_blinky\_sample\_freertos\_gcc.zip

## 6. FreeRTOS デバッグ機能

この章では e<sup>2</sup> studio の FreeRTOS デバッグ機能について説明します。

この機能により、生成されたタスク、キュー、タイマの一覧と状態をプログラム停止中に確認することができます。

### 6.1 FreeRTOS デバッグ機能の追加

#### 6.1.1 e<sup>2</sup> studio を新規インストールまたは更新する場合

- e<sup>2</sup> studio のインストーラを起動します。e<sup>2</sup> studio がインストール済みの場合、Upgrade または Install を選択してください(図 6-1)。

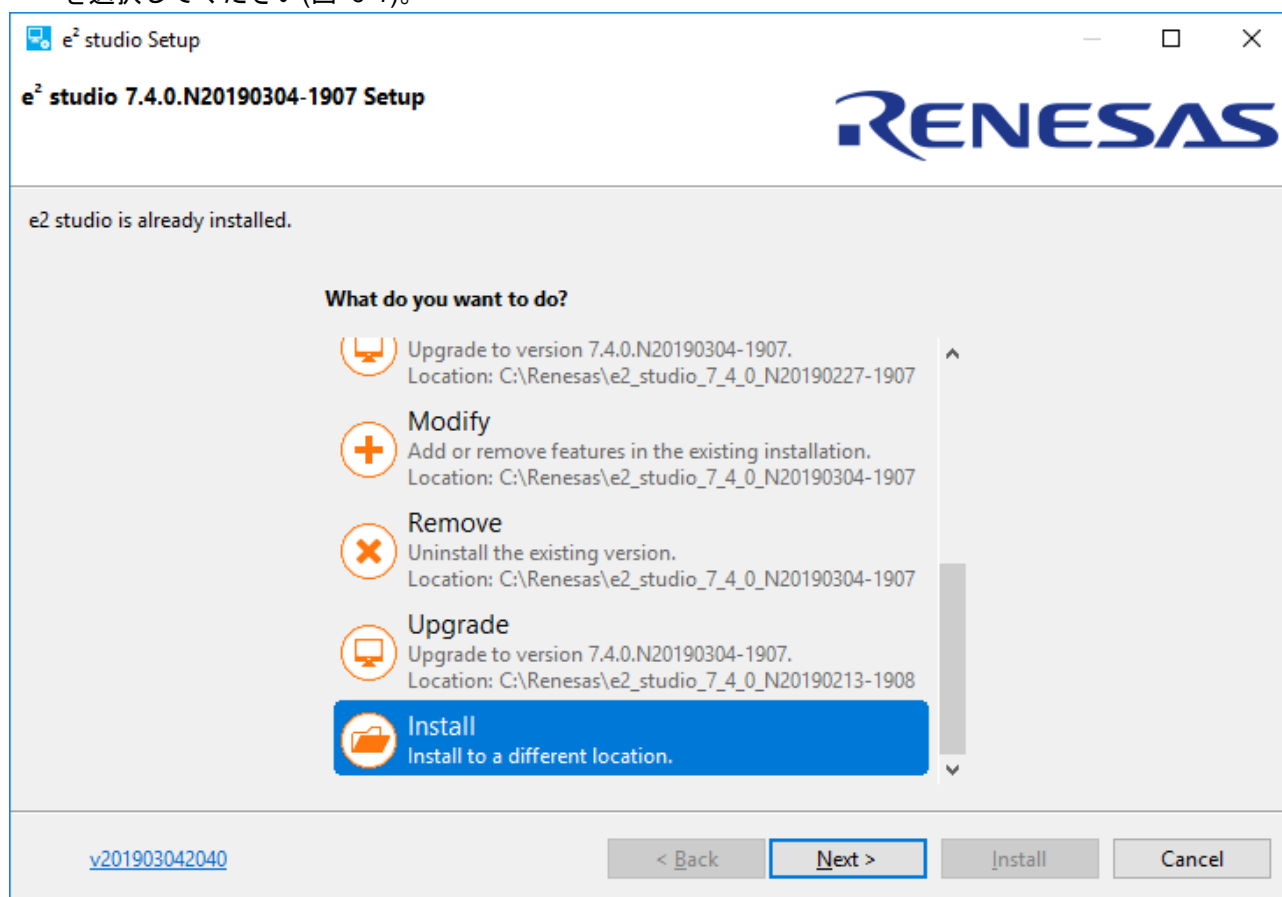


図 6-1 e<sup>2</sup> studio の新規インストールにおける、FreeRTOS デバッグの追加方法 1

— Device Families を選択するときに「RZ」にチェックを入れます(図 6-2)。

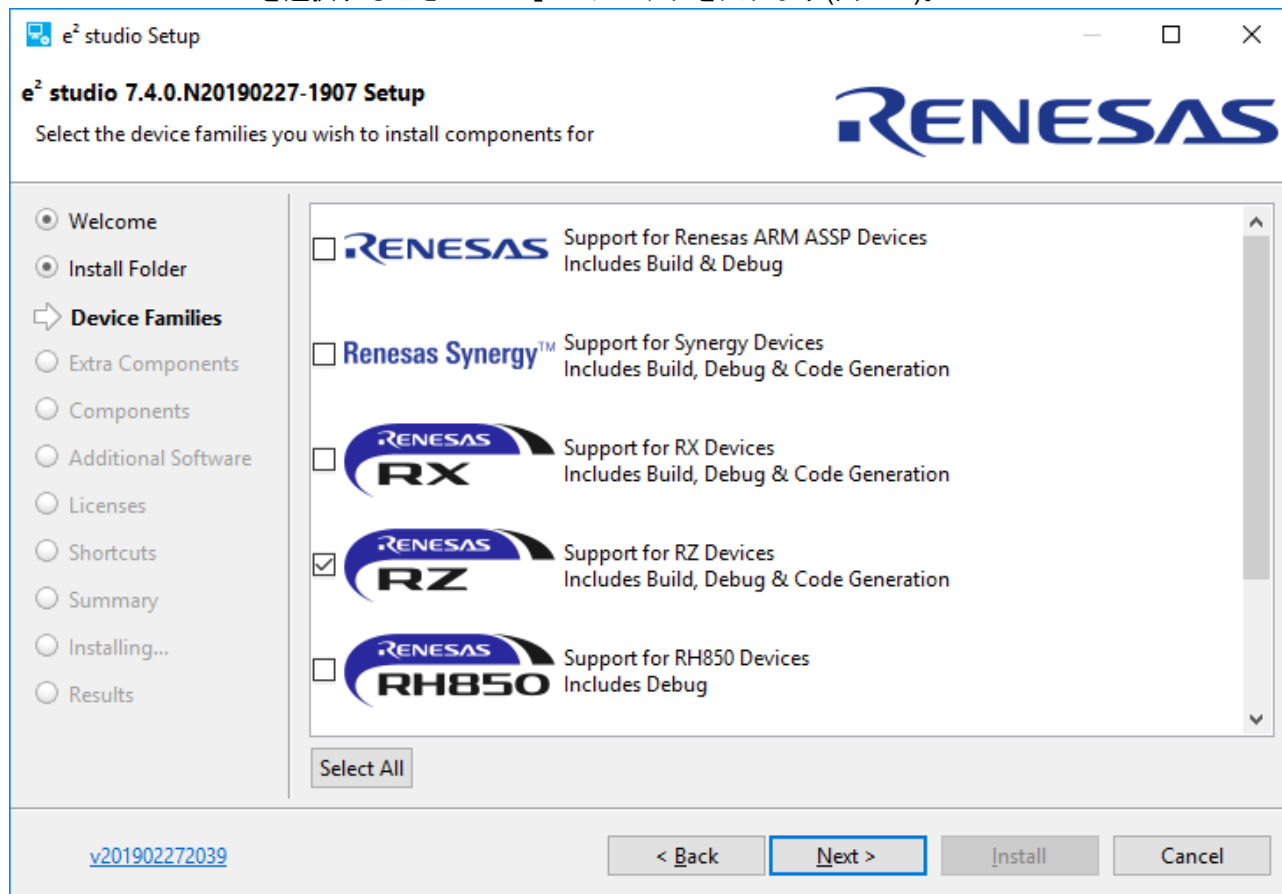


図 6-2 e² studio の新規インストールにおける、FreeRTOS デバッグの追加方法 2

— Extra Components を選択するときに「RTOS」にチェックを入れます(図 6-3)。

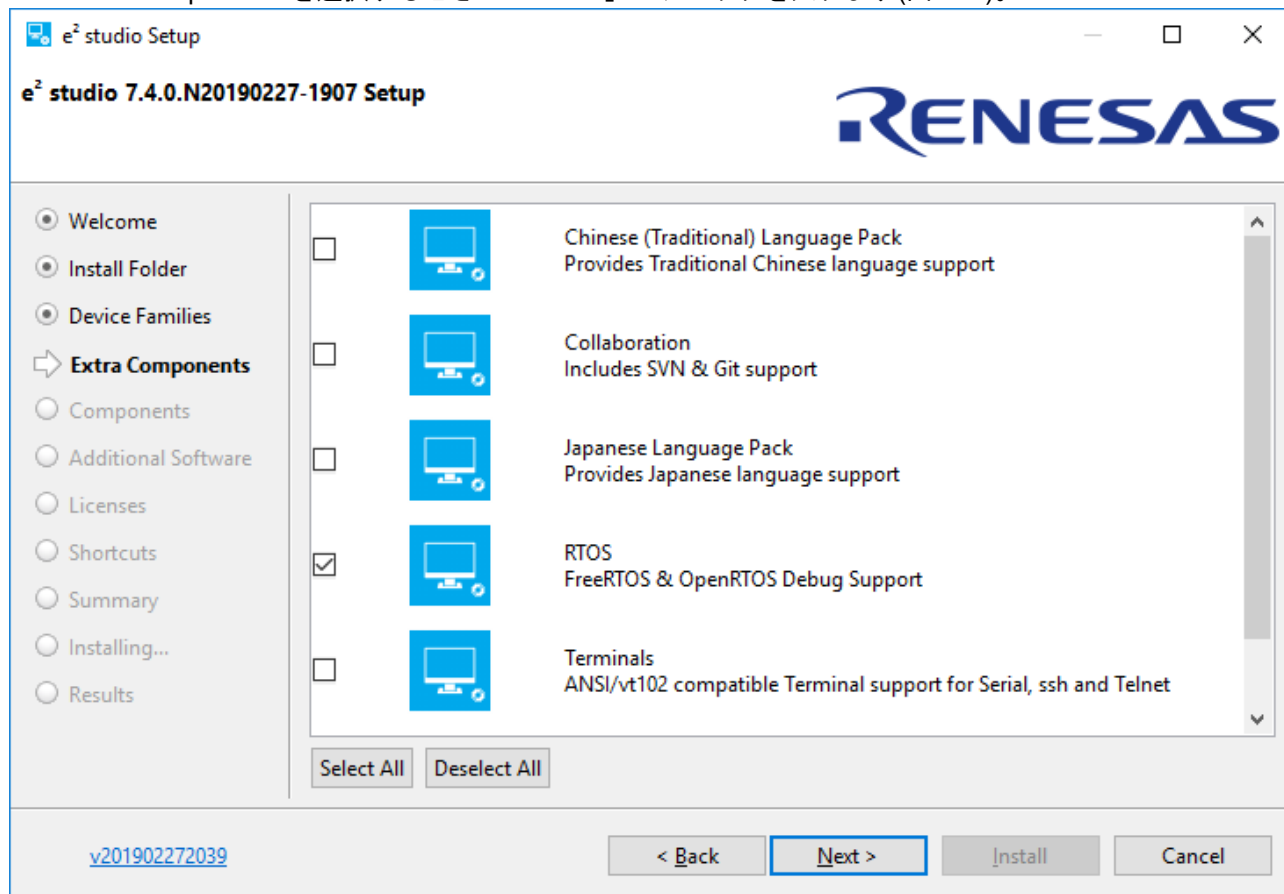


図 6-3 e² studio の新規インストールにおける FreeRTOS デバッグの追加方法 3

— Additional Software を選択するときに「GCC ARM Embedded 6 2017q2」と「LibGen for GCC ARM Embedded」にチェックを入れます(図 6-4)。

以上の設定により、FreeRTOS デバッグ機能が使用可能となります。

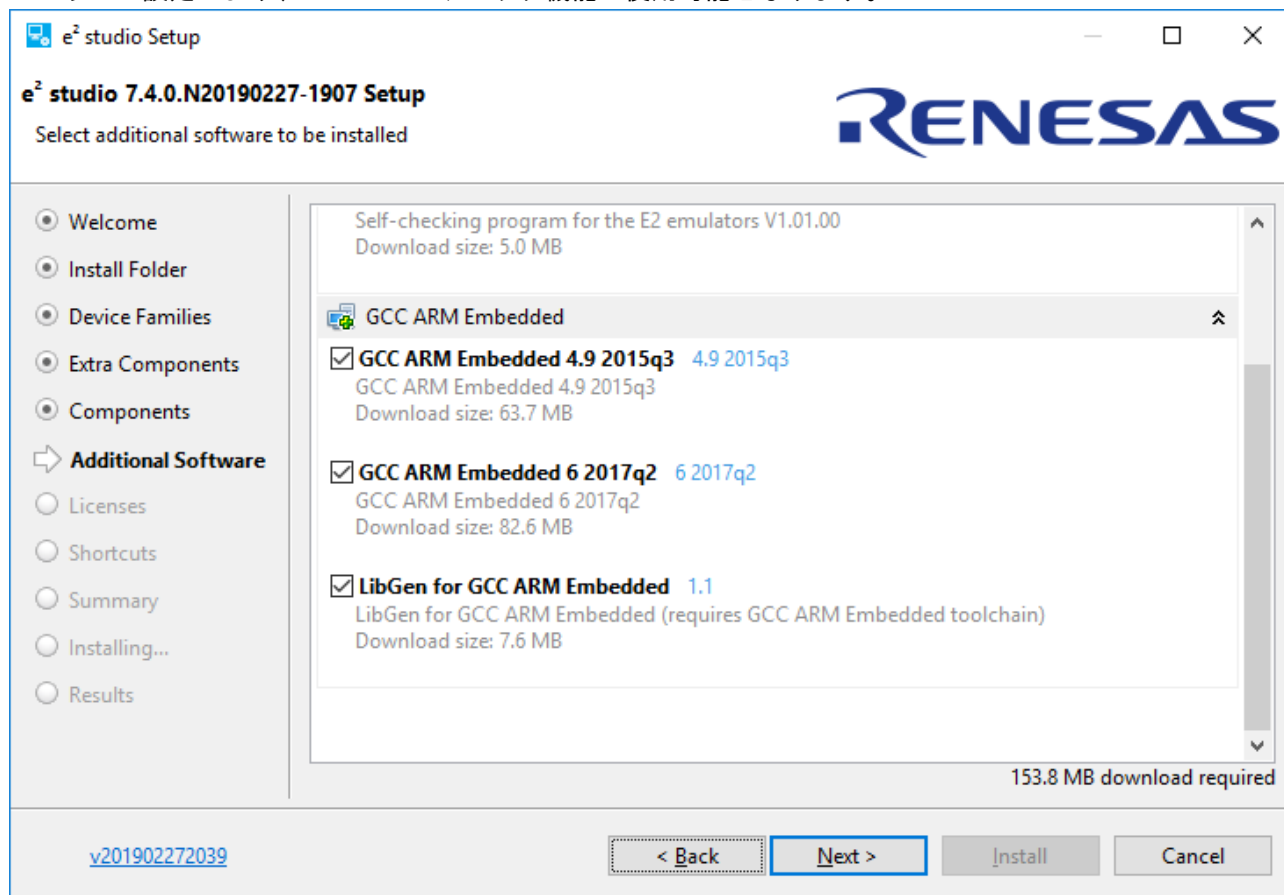


図 6-4 e² studio の新規インストールにおける、FreeRTOS デバッグの追加方法 4

## 6.1.2 インストール済みの e2 studio へ追加する場合

— e<sup>2</sup> studio のインストーラを起動します。Modify を選択してください(図 6-5)。

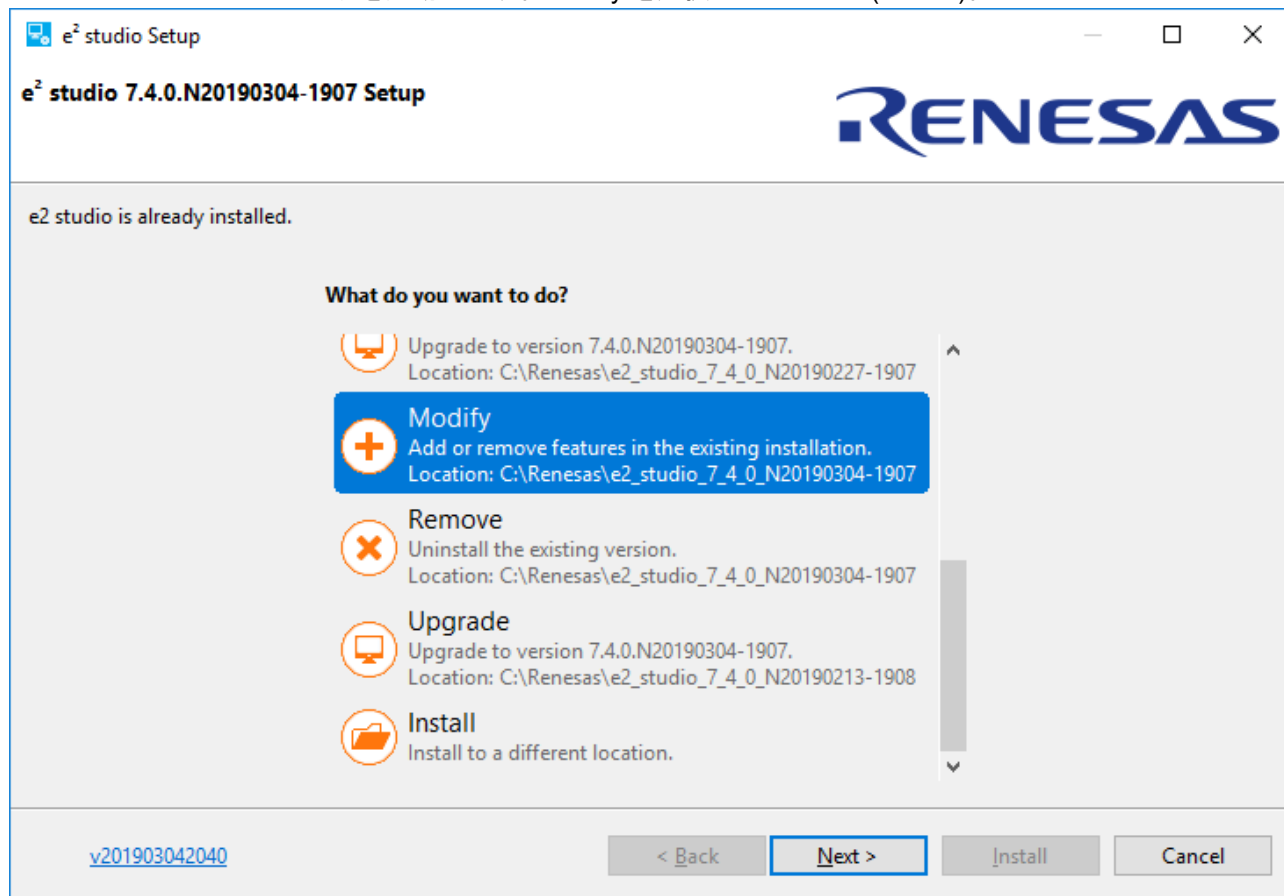


図 6-5 e<sup>2</sup> studio の更新インストールにおける、FreeRTOS デバッグの追加方法 1



— Components を選択するときに選択画面で「Renesas RTOS Debug Views」にチェックを入れます(図 6-6)。

以上の設定により、FreeRTOS デバッグ機能が使用可能となります。

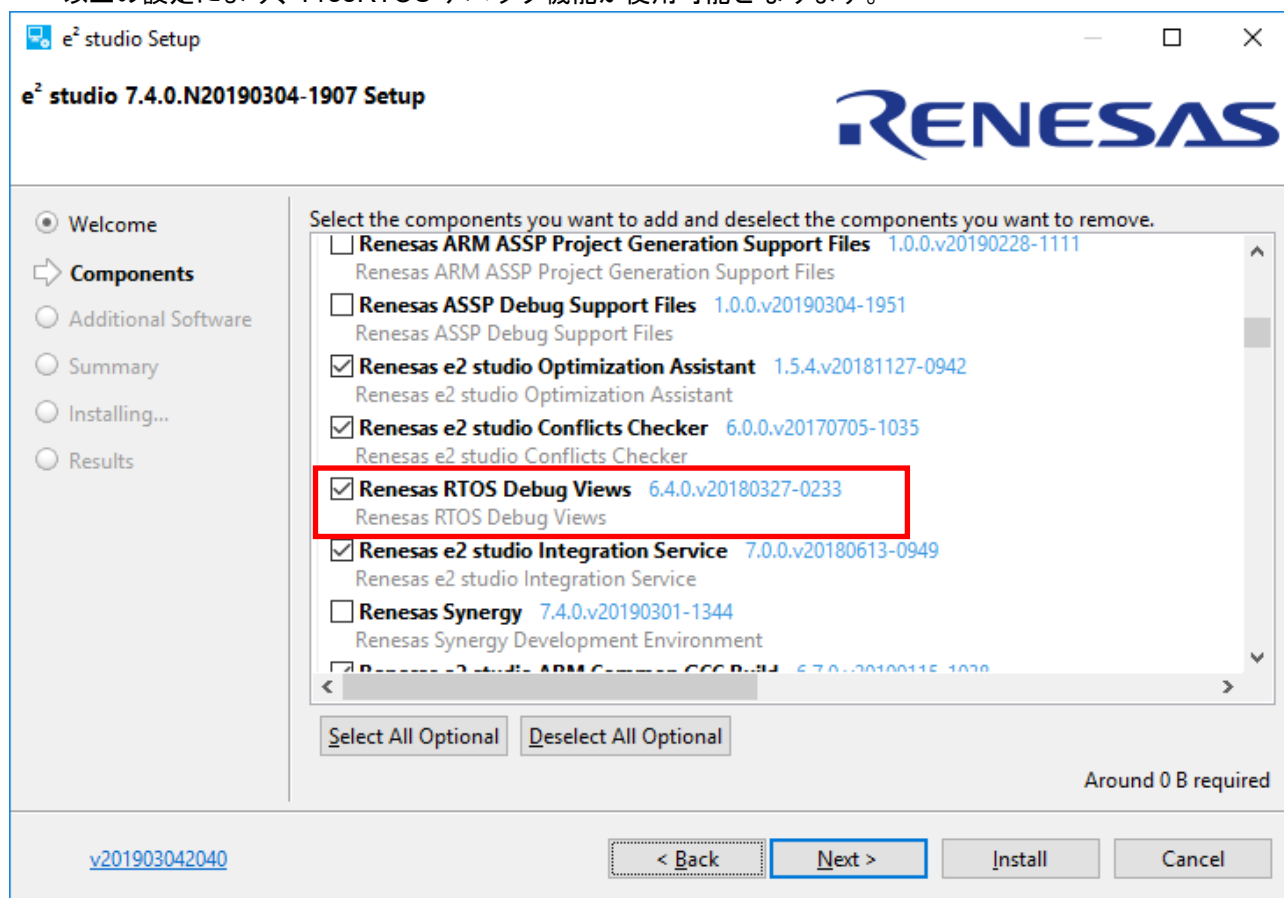


図 6-6 e2 studio の更新インストールにおける、FreeRTOS デバッグの追加方法 2

## 6.2 使用方法

- 1 FreeRTOS を使用するプログラムをボードにダウンロードします。
- 2 ダウンロードしたプログラムを実行します。
- 3 プログラムを一時停止します。
- 4 「Window」メニューの「Show view」を選択し、「Other」を選択します(図 6-7)。

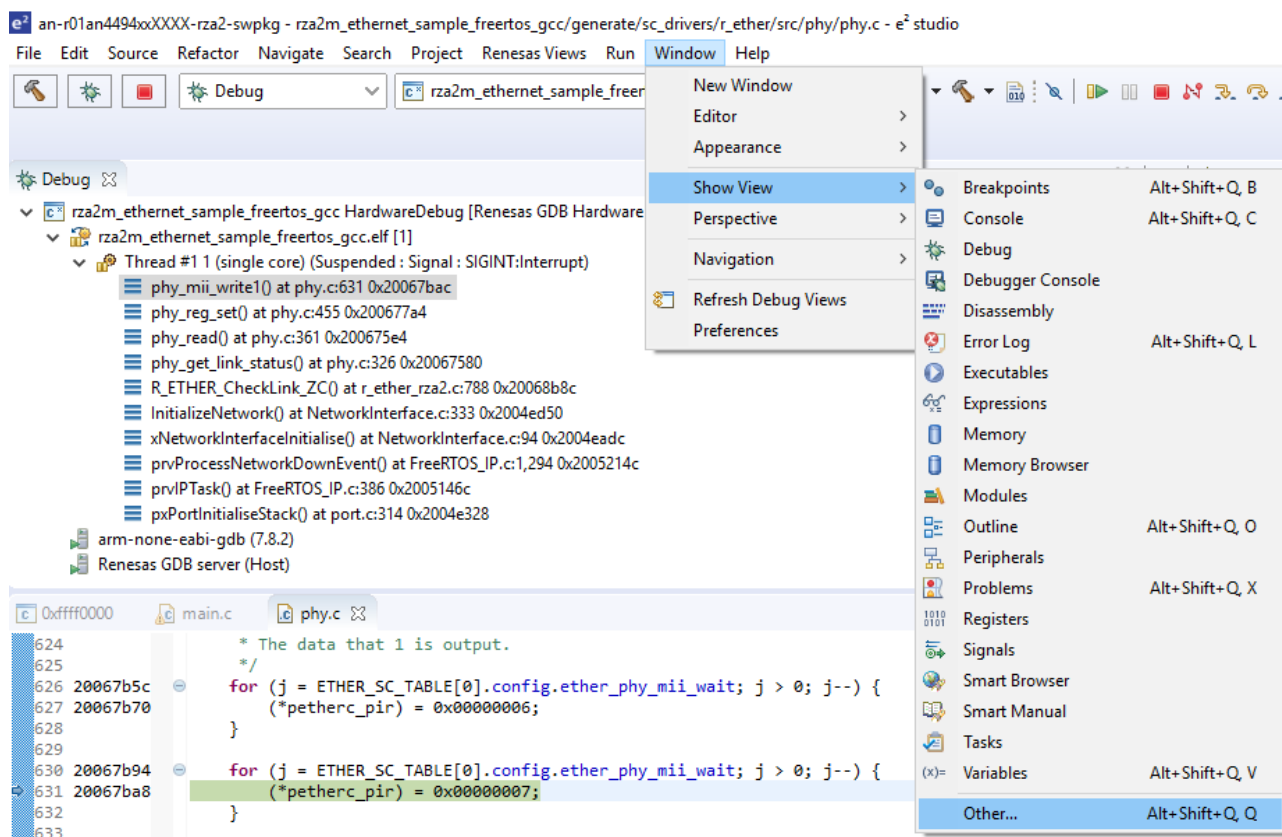


図 6-7 FreeRTOS デバッグの使用方法 4

- 5 「OpenRTOS Viewer」以下の「Queue Table」を選択し、「Open」を押します(図 6-8)。
- 6 同様の手順で「Task Table」を選択し、「Open」を押します。
- 7 同様の手順で「Timer Table」を選択し、「Open」を押します。

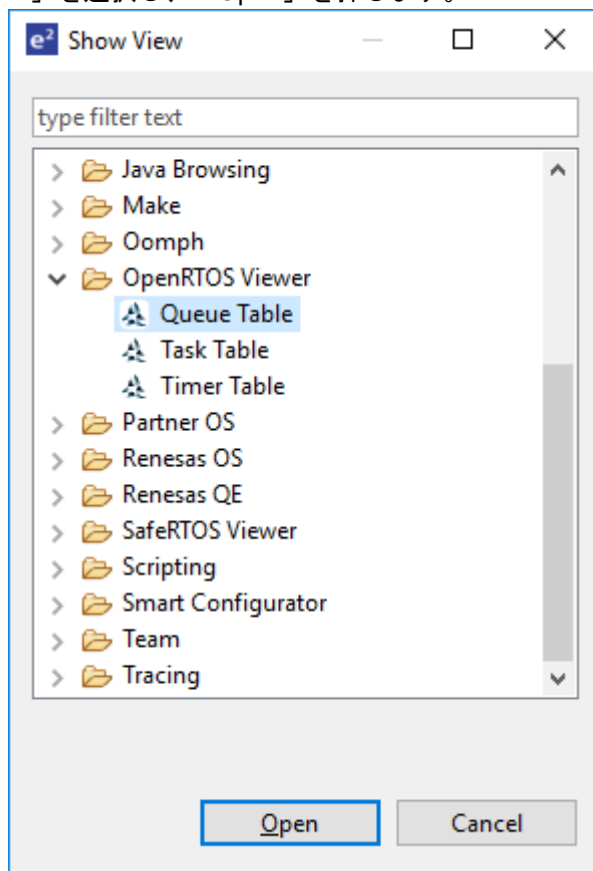
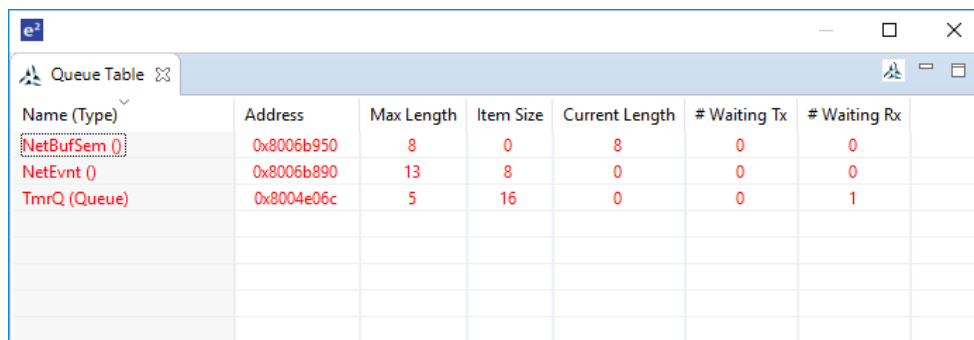


図 6-8 FreeRTOS デバッグの使用方法 5,6,7

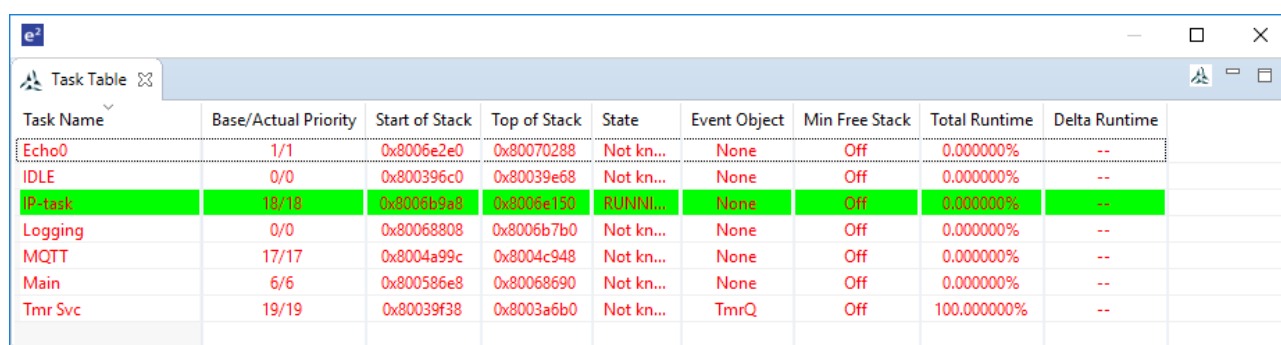
## 8. FreeRTOS のステータスが表示されます。

図 6-9に Queue Table 画面、図 6-10に Task Table 画面、図 6-11に Timer Table 画面の例をそれぞれ示します。



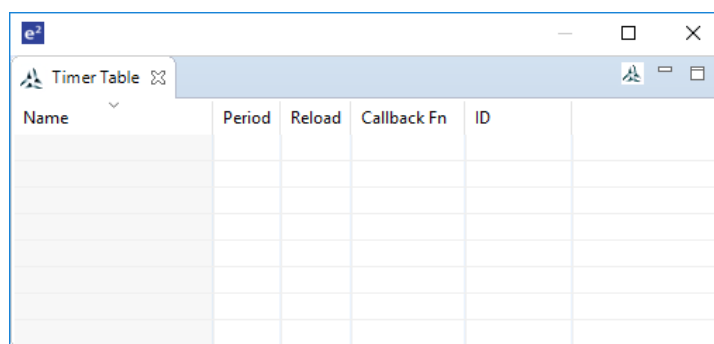
Name (Type)	Address	Max Length	Item Size	Current Length	# Waiting Tx	# Waiting Rx
NetBufSem ()	0x8006b950	8	0	8	0	0
NetEvt ()	0x8006b890	13	8	0	0	0
TmrQ (Queue)	0x8004e06c	5	16	0	0	1

図 6-9 FreeRTOS デバッグの使用 (Queue Table)



Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime	Delta Runtime
Echo0	1/1	0x8006e2e0	0x80070288	Not kn...	None	Off	0.000000%	--
IDLE	0/0	0x800396c0	0x80039e68	Not kn...	None	Off	0.000000%	--
IP-task	18/18	0x8006b9a8	0x8006e150	RUNN...	None	Off	0.000000%	--
Logging	0/0	0x80068808	0x8006b7b0	Not kn...	None	Off	0.000000%	--
MQTT	17/17	0x8004a99c	0x8004c948	Not kn...	None	Off	0.000000%	--
Main	6/6	0x800586e8	0x80068690	Not kn...	None	Off	0.000000%	--
Tmr Svc	19/19	0x80039f38	0x8003a6b0	Not kn...	TmrQ	Off	100.000000%	--

図 6-10 FreeRTOS デバッグの使用 (Task Table)



Name	Period	Reload	Callback Fn	ID
------	--------	--------	-------------	----

図 6-11 FreeRTOS デバッグの使用 (Timer Table)

## 7. e<sup>2</sup>studio の仮想コンソールの使用方法

e<sup>2</sup>studio には仮想コンソール機能があります。

プロジェクトで特定の設定を行うことにより、標準 I/O を仮想コンソールにリダイレクトすることができます。

ここでは、仮想コンソールの使用条件、設定手順、注意事項について説明します。

### 7.1 仮想コンソールを使用するための要件

仮想コンソールは、以下の条件を満たす場合に使用できます。

- e<sup>2</sup>studio version 7.7 以降
- 対象プロジェクトが OSless

### 7.2 仮想コンソールの制限事項

仮想コンソールには以下の制限があります。既存の標準 I/O を完全に置き換えるわけではありません。

制限事項をご理解の上、仮想コンソールをご利用ください。

- 通信速度が非常に遅いため、大量の入出力には適していません。
- getc または scanf の入力を待っている間、プログラムは完全に停止します。（割り込みは発生しません）
- 仮想コンソールに対するバッファリング操作(fflush, setvbuf など)は禁止です。
- 仮想コンソールが有効になっている場合、プログラムはスタンドアロンで実行できません。仮想コンソールを無効にして、最終的なプロジェクトを作成してください。

### 7.3 仮想コンソールを有効にする方法

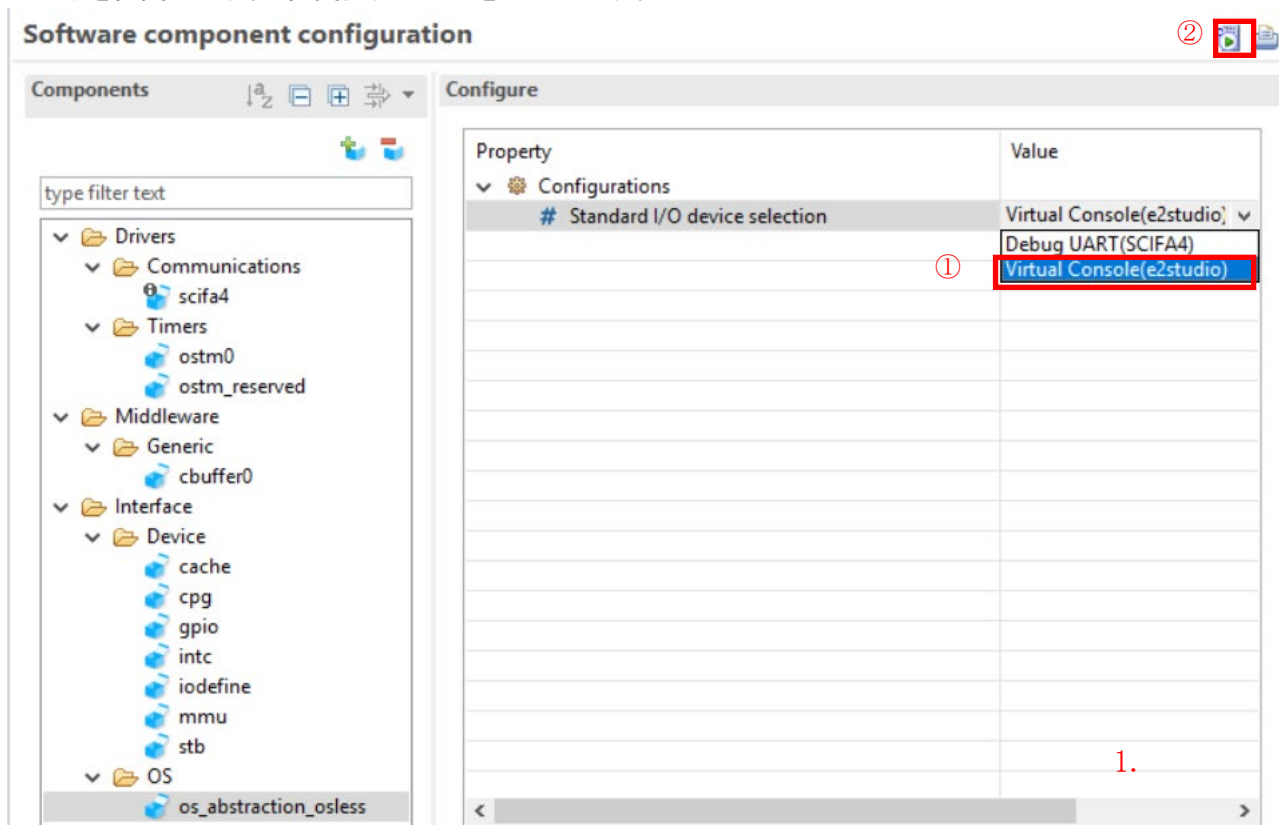
このセクションでは、RZ/A2M Simple Applications Package (R01AN4494) にバンドルされている `rza2m_blinky_sample_osless_gcc` プロジェクトの仮想コンソールを有効にする方法について説明します

#### 手順の要約

1. Smart Configurator で仮想コンソールを有効化
2. `r_devlink_wrapper_cfg.h` を編集して次の 2 点を変更
  - `#include "r_os_abstraction_api.h"` を追加
  - 仮想コンソールの標準入出力ファイル名を登録
3. アプリケーション中の標準 I/O のバッファ操作の削除。
  - 全ての `setvbuf` コールを削除
  - 全ての `fflush` コールを削除
4. プロジェクトをビルド
5. マップファイルの `svc_handler` アドレスを確認
6. `svc_handler` アドレスをデバッグ接続設定ファイルに登録
7. プログラムのダウンロード
8. 仮想コンソールのビューをオープン
9. プログラムの実行

詳細な手順を下記に示します。

1. Smart Configurator で仮想コンソールを有効にする  
SmartConfigurator の[コンポーネント]タブで、`os_abstraction_osless` を選択し、次の図に示すようにプロパティを変更します。 変更後、コードを生成します。



2. 仮想コンソールの標準入出力ファイル名を登録します。  
 generate¥configuration¥r\_devlink\_wrapper\_cfg.h を編集します。  
 まず、40 行目と 41 行目を次のように変更します。

```
/* Modified by user, drivers that are not under the control of sc added here */
#include "r_os_abstraction_api.h"
/* End of user modification */
```

次に、62～71 行目を次のように変更します。

```
#if (R_OS_ENABLE_VIRTUAL_CONSOLE == 0)
    /** SCIFA Channel 4 Driver added by USER */
    {"stdin", (const st_r_driver_t*)&g_scifa_driver, R_SC0},

    /** SCIFA Channel 4 Driver added by USER */
    {"stdout", (const st_r_driver_t*)&g_scifa_driver, R_SC0},

    /** SCIFA Channel 4 Driver added by USER */
    {"stderr", (const st_r_driver_t*)&g_scifa_driver, R_SC0},
#else
    {"stdin", (const st_r_driver_t*)&g_stdio_driver, R_SC0},
    {"stdout", (const st_r_driver_t*)&g_stdio_driver, R_SC0},
    {"stderr", (const st_r_driver_t*)&g_stdio_driver, R_SC0},
#endif
```

3. 標準 I/O のバッファ操作の削除。

まず、src¥user\_prog¥main.c を編集して、setvbuf の呼び出しをコメント化します。  
 次に、src¥renesas¥application¥console¥console.c を編集して、fflush および setvbuf へのすべての呼び出しをコメント化します。

4. プロジェクトをビルドします。

以降では、HardwareDebug 構成が構築されていると想定しています。

5. マップファイルの svc\_handler アドレスを確認します。

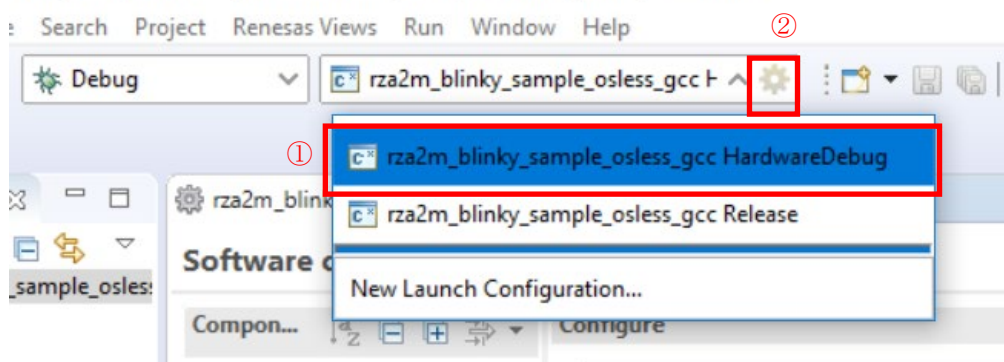
エディターで Hardwaredebug¥rza2m\_blinky\_sample\_osless\_gcc.map を開き、svc\_handler のアドレスを確認します。

ここでは、以下のようにアドレスを 0x20011460 として説明を続けます。

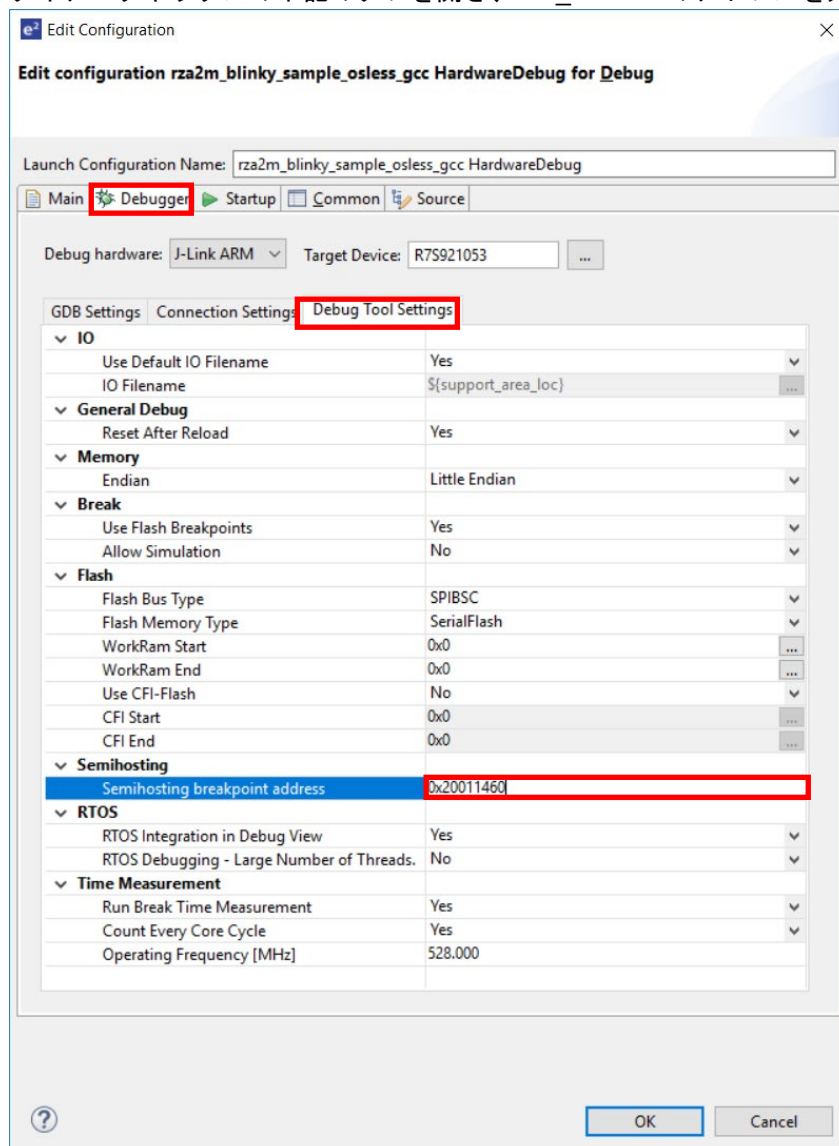
RESET_HANDLER	0x20011280	0x1f0 ./generate/compiler/asm/reset_handler.o
	0x20011280	reset_handler
	0x2001145c	undefined_handler
	0x20011460	svc_handler
	0x20011464	prefetch_handler
	0x20011468	abort_handler
	0x2001146c	reserved_handler

6. svc\_handler アドレスをデバッグ接続設定ファイルに登録します。  
e<sup>2</sup>studio 上部のバーから、HardwareDebug のデバッガー接続設定の編集を選択します。

blinky\_sample\_osless\_gcc/rza2m\_blinky\_sample\_osless\_gcc.scfg - e<sup>2</sup> studio



ダイアログボックスの下記のタブを開き、svc\_handler のアドレスを入力します。





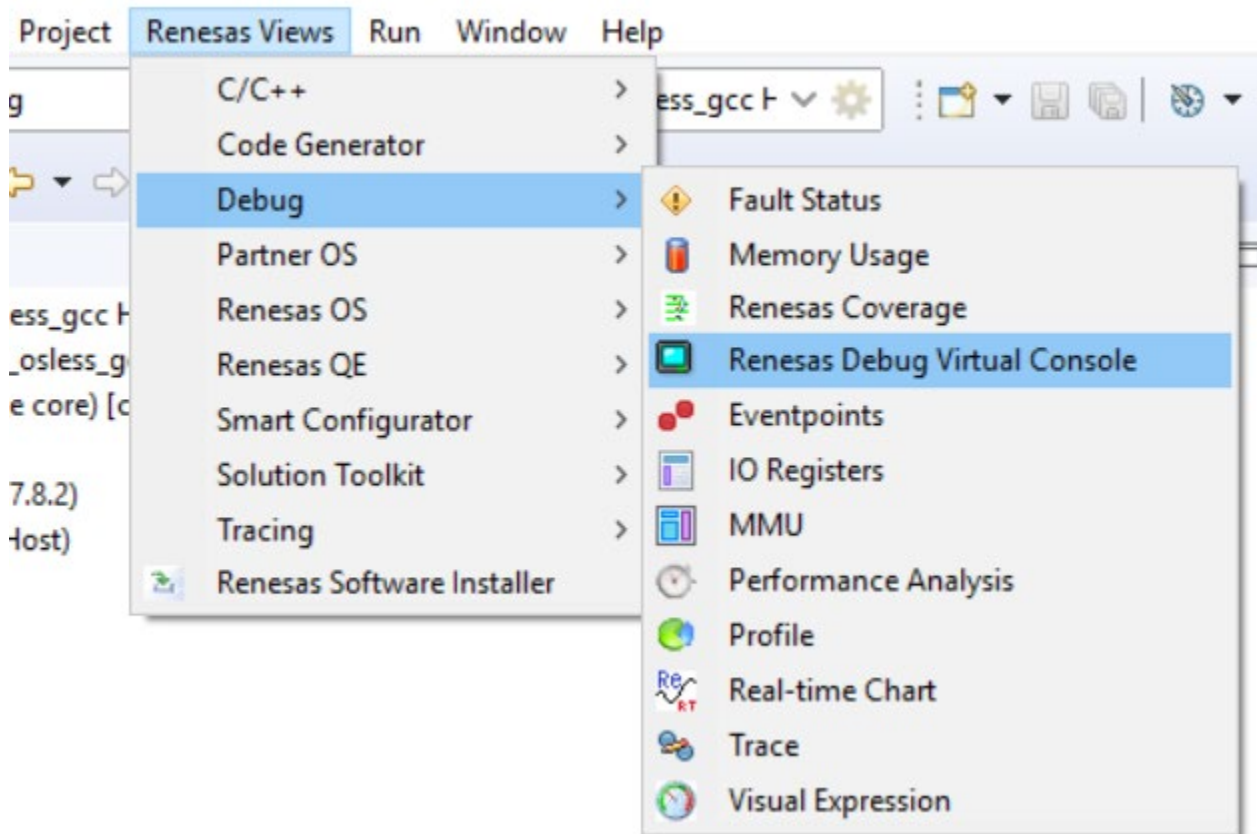
## 7. プログラムのダウンロード

前の項で設定を行った環境で、デバッグを開始してください。

## 8. 仮想コンソールのビューをオープン

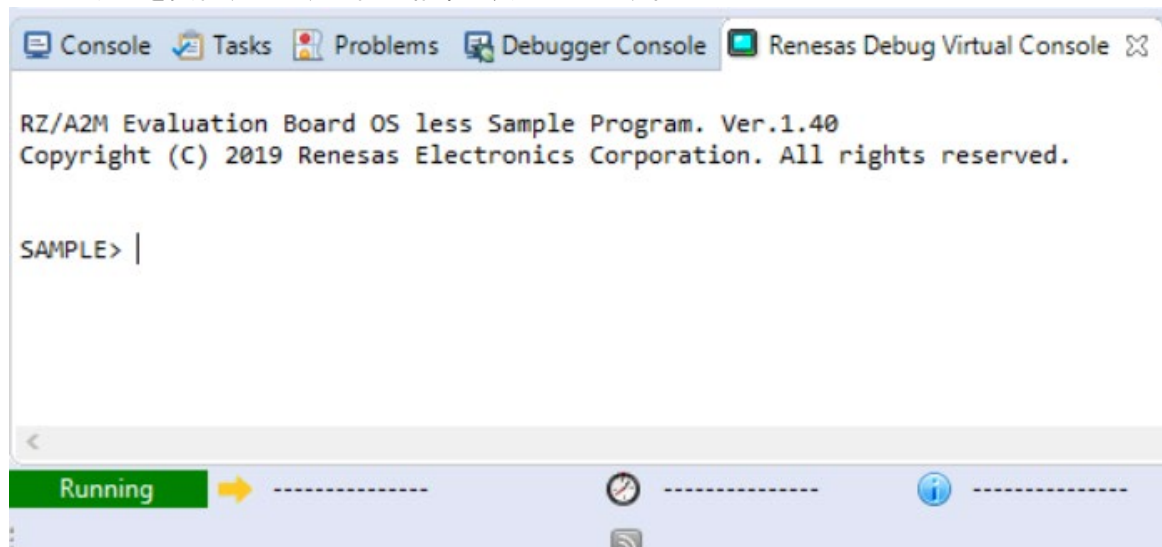
e2studio の[Renesas Views]メニューから、下記のビューを選択してください。

- e<sup>2</sup> studio



## 9. プログラムの実行

プログラムを実行すると、下記の結果が表示されます。



注：コンソール入力待ちでプログラムが停止しているため、LED の点滅は停止しています。

## 8. ドライバアップデート時の注意事項

最新のドライバを既存のプロジェクトに適用する場合、プロジェクトを手動で変更する必要があるいくつかの作業があります。

以下の手順に従ってプロジェクトを更新してください。

### 8.1 r\_iodefne

古いプロジェクトには、r\_iodefne コンポーネントが含まれていません。r\_iodefne を含まないプロジェクトでは、ヘッダーファイルが直接配置されます。

このようなプロジェクトに r\_iodefne を追加する場合は、以下の古いヘッダーファイルを手動で削除してください。

- generate¥system¥inc¥iobitmask.h
- generate¥system¥inc¥iodefne.h
- generate¥system¥inc¥iobitmasks folder
- generate¥system¥inc¥iodefnes folder

### 8.2 r\_os\_abstraction

#### 8.2.1 3.04 以前からの更新

一部のコンパイラ依存ファイルは、仮想コンソールをサポートするために r\_os\_abstraction に移動されました。

以下のファイルを削除してください。

- generate¥compiler¥inc¥swi.h
- generate¥compiler¥init¥\_exit.c
- generate¥compiler¥init¥\_kill.c
- generate¥compiler¥init¥syscalls.c

現在のバージョンが 3.03 以前の場合は、次のセクションも参照してください。

#### 8.2.2 3.03 以前からの更新

generate¥compiler¥inc¥r\_compiler\_abstraction\_api.h に下記の定義を追加してください。

- #define R\_COMPILER\_WEAK \_\_attribute\_\_((weak))

また、r\_os\_abstraction\_freertos を使用している場合は、下記の変更も必要です。

ヒープ実装は、FreeRTOS フォルダから r\_os\_abstraction\_freertos に移動しました。

以下のファイルを削除してください。

- <freertos folder>¥portable¥memmang¥heap5\_renesas.c

## 改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	28.Dec.18	—	初版
1.01	15.Apr.19	11	4.1章「コンポーネントとサンプルプロジェクトの関連」を追加
		13	4.3章「各コンポーネントの実装方法」を追加
		20	6章「FreeRTOSデバッグ機能」を追加
1.02	17.May.19	10	4章に RZ/A2M スマート・コンフィグレータ ユーザーガイドについての記述を追加
1.03	30.Sep.19	2	ツールのインストールガイドの案内を追加
		2	e2 studio のバージョンを 7.5 に変更
		11	sdhi_fat サンプルに osless 版を追加
		11	adc サンプル及び r_adc ドライバを追加
		11	usbf_cdc サンプル及び r_usbf_basic, r_usbf_cdc ドライバを追加
		11	wifi_pmod_esp32 サンプルを追加
1.04	Dec.17.19	2	e2 studio のバージョンを 7.6 に変更
		7	ダウンロード手順変更
		11	rtc サンプル及び r_rtc ドライバを追加
1.05	Mar.31.20	2	e2 studio のバージョンを 7.7 に変更
		11	fw_update_boot サンプル及び fw_update_sample を追加 gpt-pwm サンプル及び r_gpt ドライバを追加 r_hyperbus ドライバを追加 touch_panel サンプルを追加
1.06	Jun.30.20	2	e2 studio のバージョンを 7.8 に変更
		19	「5. FreeRTOSプロジェクトの作成方法」を追加
		29	「7. e2studioの仮想コンソールの使用方法」を追加
		34	「8. ドライバアップデート時の注意事項」を追加
1.07	Sep.30.20	2	e2 studio のバージョンを 2020-07 に変更
		11	ssif サンプル及び r_ssif ドライバを追加 graphics サンプル及び r_rga ドライバを追加 wifi_sxdmac サンプル及び silex ドライバを追加
1.08	Apr.20.21	2	e2 studio のバージョンを 2021-04 に変更

## 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イやマガジンケース、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、入力信号や入出力ブルアップ電源を入れないでください。入力信号や入出力ブ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。お客様の機器・システムの設計において、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には、お客様の責任において行ってください。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まれます。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、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、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。
  5. 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。
  6. 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。
  7. 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。
  8. 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、当社は、一切その責任を負いません。
  9. 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。
  10. 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。
  11. 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。
  12. 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.4.0-1 2017.11)

## 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](http://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/](http://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。