CAN スタータキット RX/RA
CAN スタータキット SmartRX
ソフトウェア編　マニュアル

ルネサス エレクトロニクス社　RX/RA マイコン搭載
HSB シリーズマイコンボード　評価キット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.15.0.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

警告 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

注意 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプの点灯中に電源を切ったり、パソコンをリセットをしないでください。

製品の故障や、データ消失の原因となります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

概要

本書は、「CAN スタータキット RX/RA」「CAN スタータキット SmartRX」付属 CD に含まれる、サンプルプログラムの解説を行う資料となります。

本書で説明するサンプルプログラムは、CAN の通信を行うプログラムで以下の内容となっています。

・SAMPLE1
データフレームの送受信（割り込み未使用）

・SAMPLE2
データフレームの送受信（割り込み使用）

・SAMPLE3
データフレームとリモートフレームの送受信（割り込み使用）

・SAMPLE4
データフレームとリモートフレームの送受信（割り込み使用）, CANFD フレーム　※CANFD 対応マイコンのみ

プログラムは、通信を行うコードがシンプルで判り易くなるよう構成しています。エラー等の処理は含まれていませんので、実際のアプリケーションを構築する際は、必要な処理を追加してください。

SAMPLE1 から徐々に機能を追加していく流れになっており、CAN の通信を始めて学びたいという方をターゲットにしています。

サンプルプログラム CD

製品に付属しているサンプルプログラム CD の内容を下記に示します。

－RX 向け－

マイコン種毎のプロジェクト

フォルダ		内容
SOURCE¥RX_CANST¥	RX23E-B_100¥	RX23E-B 100pin 向け プロジェクトフォルダ
	RX24U_144_100¥	RX24U/RX24T 144/100pin 向け プロジェクトフォルダ
	RX26T_100	RX26T/100pin 向け プロジェクトフォルダ
	RX64M_176	RX64M 176pin 向け プロジェクトフォルダ
	RX65_1M_144_100¥	RX65(ROM:1M) 144/100pin 向け プロジェクトフォルダ
	RX65_144_100¥	RX65 144/100pin 向け プロジェクトフォルダ
	RX65_176¥	RX65 176pin 向け プロジェクトフォルダ
	RX66N_144_100¥	RX66N 144/100pin 向け プロジェクトフォルダ
	RX66N_176¥	RX66N 176pin 向け プロジェクトフォルダ
	RX66T_100A¥	RX66T 100pin（チップバージョン A) 向けプロジェクトフォルダ
	RX66T_100B¥	RX66T 100pin（チップバージョン B) 向けプロジェクトフォルダ
	RX66T_144¥	RX66T 144pin 向け プロジェクトフォルダ
	RX71M_100¥	RX71M 100pin 向け プロジェクトフォルダ
	RX71M_176¥	RX71M 176pin 向け プロジェクトフォルダ
	RX72M_176¥	RX72M 176pin 向け プロジェクトフォルダ
	RX72N_144_100¥	RX72N 144/100pin 向け プロジェクトフォルダ
	RX72N_176¥	RX72N 176pin 向け プロジェクトフォルダ
	RX72T_144¥	RX72T 144pin 向け プロジェクトフォルダ
	RX140_80	RX140/80pin 向け プロジェクトフォルダ
	RX231_100¥	RX231 100pin 向け プロジェクトフォルダ
	RX261_100¥	RX261 100pin 向け プロジェクトフォルダ

フォルダ		内容
SOURCE¥RX_CANST¥	RX660_144_100¥	RX660 144/100pin 向け プロジェクトフォルダ
	RX671_144_100¥	RX671 144/100pin 向け プロジェクトフォルダ
	SmartRX¥	SmartRX!!!向け プロジェクトフォルダ

ソース（共通）

フォルダ		内容
SOURCE¥RX_CANST¥source¥ALL¥	common	速度設定など
	sci	SCI(UART)ドライバ

ソース（CAN モジュール向け）

フォルダ		内容
SOURCE¥RX_CANST¥source¥CAN_module¥	can	CAN モジュール共通定義フォルダ
	can_ch0	CAN モジュール ch0
	can_ch1	CAN モジュール ch1
	can_ch2	CAN モジュール ch2
	main	メイン関数

ソース（RSCAN モジュール向け）

フォルダ		内容
SOURCE¥RX_CANST¥source¥RSCAN_module¥	rscan	RSCAN モジュール
	rscan_ch0	RSCAN モジュール ch0
	main	メイン関数

ソース（CANFD_Lite モジュール向け）

フォルダ		内容
SOURCE¥RX_CANST¥source¥CANFDLite_module¥	canfd_lite	CANFD_Lite モジュール共通定義フォルダ
	canfd_lite_ch0	CANFD_Lite モジュール ch0
	main	メイン関数

サンプルプログラムは、ルネサスエレクトロニクス CS+(CS+forCC)向けのプロジェクトで作成されています。
CS+ for CC Ver8.12 以降を予めインストール願います。

e2studio から CS+プロジェクトをインポートして使用する事も可能です。

－RA 向け－

マイコン種毎のプロジェクト

フォルダ		内容
SOURCE¥RA_CANST¥mcu¥	RA2A1¥settings¥	RA2A1 向け 設定ファイルフォルダ
	RA2L1¥settings¥	RA2L1 向け 設定ファイルフォルダ
	RA4E1¥settings¥	RA4E 向け 設定ファイルフォルダ
	RA4E2¥settings¥	RA4E2 向け 設定ファイルフォルダ
	RA4M1¥settings¥	RA4M1 向け 設定ファイルフォルダ
	RA4M2¥settings¥	RA4M2 向け 設定ファイルフォルダ
	RA4M3¥settings¥	RA4M3 向け 設定ファイルフォルダ
	RA4T1¥settings¥	RA41 向け 設定ファイルフォルダ
	RA6E1¥settings¥	RA6E1 向け 設定ファイルフォルダ
	RA6E2¥settings¥	RA6E2 向け 設定ファイルフォルダ
	RA6M1¥settings¥	RA6M1 向け 設定ファイルフォルダ
	RA6M2¥settings¥	RA6M2 向け 設定ファイルフォルダ
	RA6M3¥settings¥	RA6M3 向け 設定ファイルフォルダ
	RA6M4¥settings¥	RA6M4 向け 設定ファイルフォルダ
	RA6M5¥settings¥	RA6M5 向け 設定ファイルフォルダ
	RA6T1¥settings¥	RA6T1 向け 設定ファイルフォルダ
	RA6T2¥settings¥	RA6T2 向け 設定ファイルフォルダ
	RA6T3¥settings¥	RA6T3 向け 設定ファイルフォルダ
	RA8E1¥settings¥	RA8E1 向け 設定ファイルフォルダ
	RA8M1¥settings¥	RA8M1 向け 設定ファイルフォルダ
	RA8T1¥settings¥	RA8T1 向け 設定ファイルフォルダ

ソース（共通）

フォルダ		内容
SOURCE¥RA_CANST¥source¥	hal_entry.c	エントリ関数サンプル

ソース（共通）

フォルダ		内容
SOURCE¥RA_CANST¥source¥ALL¥	clock	クロック設定
	common	ボード設定
	intr	割り込み設定
	sci	SCI(UART)ドライバ

ソース（CAN モジュール向け）

フォルダ		内容
SOURCE¥RA_CANST¥source¥CAN_module¥	can	CAN 共通定義フォルダ
	can_ch0	CAN モジュール ch0
	can_ch1	CAN モジュール ch1
	main	メイン関数

ソース（CANFD モジュール向け）

フォルダ		内容
SOURCE¥RA_CANST¥source¥CANFD_module¥	canfd	CANFD 共通定義フォルダ
	canfd_ch0	CANFD モジュール ch0
	canfd_ch1	CANFD モジュール ch1
	main	メイン関数

ソース（CANFD_B モジュール向け）

フォルダ		内容
SOURCE¥RA_CANST¥source¥CANFD_B_module¥	canfd_b	CANFD_B 共通定義フォルダ
	canfd_b_ch0	CANFD_B モジュール ch0
	main	メイン関数

ソース（CANFD_2 モジュール向け）

フォルダ		内容
SOURCE¥RA_CANST¥source¥CANFD_2_module¥	canfd_2	CANFD_2 共通定義フォルダ
	canfd_2_ch0	CANFD_2 モジュール ch0
	canfd_2_ch1	CANFD_2 モジュール ch1
	main	メイン関数

サンプルプログラムは、GCC ARM Embedded (13.2.1.arm-13-7)で動作を確認しています。

1. プロジェクトフォルダ

1.1. RX 向け CS+プロジェクトフォルダ

RX 向けのサンプルプログラムは、ルネサスエレクトロニクス CS+向けのプロジェクトとして作成されています。

CS+のプロジェクトフォルダは
SOURCE¥RX_CANST¥RX*xxx*
（*xxx* はマイコンタイプ名で分かれます）となっています。プログラムを動かすマイコンボードに応じて適切なプロジェクトフォルダを選択してください。

対象マイコンボード	プロジェクトフォルダ
SmartRX!!!	SmartRX
HSBRX71M176	RX71M_176
HSBRX71M100	RX71M_100
HSBRX72M176	RX72M_176
HSBRX72N176	RX72N_176
HSBRX72N144 HSBRX72N100	RX72N_144_100
HSBRX72T144	RX72T_144
HSBRX64MC	RX64M_176
HSBRX65N176 HSBRX651F176	RX65_176
HSBRX65N144A HSBRX65N100A HSBRX651F144A HSBRX651F100A	RX65_144_100
HSBRX65N144 HSBRX65N100 HSBRX651F144 HSBRX651F100	RX65_1M_144_100
HSBRX660-144H HSBRX660-100B	RX660_144_100
HSBRX66N176	RX66N_176
HSBRX66N144 HSBRX66N100	RX66N_144_100
HSBRX66T144	RX66T_144
HSBRX66T100A	RX66T_100A
HSBRX66T100B	RX66T_100B
HSBRX671F144 HSBRX671F100	RX671_144_100
HSBRX231F100	RX231_100
HSBRX23E-B100	RX23E-B_100
HSBRX24U144 HSBRX24U100 HSBRX24T100B	RX24U_144_100

対象マイコンボード	プロジェクトフォルダ
HSBRX261-100	RX261_100
HSBRX26T100	RX26T_100
HSBRX140F80	RX140_80

プロジェクトは、マイコン種毎に用意しています。ソースファイルは、

SOURCE¥RX_CANST¥source

以下に格納されており、各プロジェクトから共通で参照される様になっています。

CD 内の RX_CANST フォルダを、PC のストレージ上の任意の場所にコピーして、プロジェクトフォルダ内の、

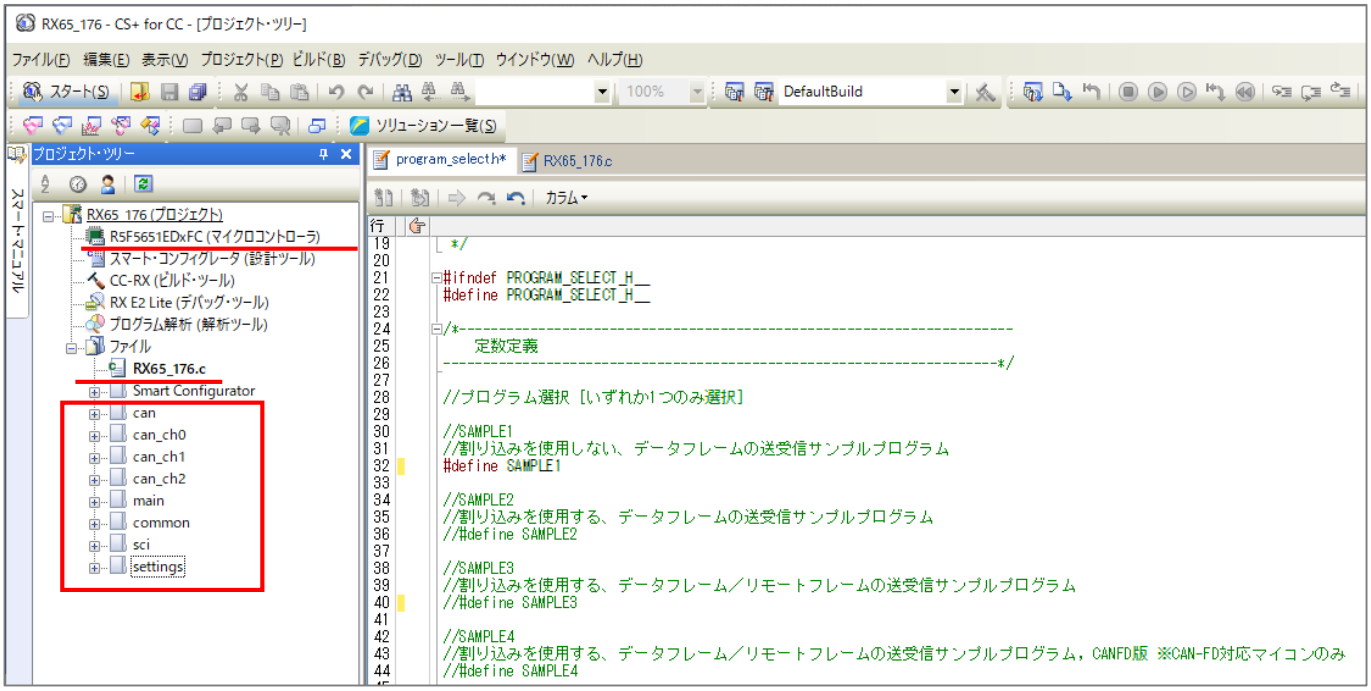
プロジェクト名.mtpj

から、CS+を起動してください。

1.2. RX 向け CS+プロジェクト

CS+プロジェクトフォルダ内の、プロジェクトファイル(.mtpj)をダブルクリックで CS+を起動するとプロジェクトが開かれます。

※以下の画面では RX65_176 のプロジェクトの例を示しています



プロジェクトで使用している設定、ファイルに関して説明します。

・マイコン名　R5F565NEDxFC（マイクロコントローラ）
使用するマイコン名を選択します。HSBRX65N176 に搭載されているのが、R5F565NEDDFC となりますので、RX65_176 のプロジェクトでは、このマイコンを選択しています。
※基本的には、接続するマイコンボードに搭載しているマイコンに合わせて変更する項目です（変更しなくても、プログラムのビルド結果は変わりませんので、変更は必須ではありません）

・プロジェクト名.c（上記では RX65_176.c）
[メイン関数呼び出し用テンプレート]
選択した(SAMPLEn)に対応するメイン関数を呼び出します。

can
can_ch0
can_ch1
can_ch2
main
main
sci

は、source 以下のフォルダを参照しています。（複数のプロジェクトで共通のソースファイルを参照しています。

settings

は、プロジェクトフォルダ¥src¥usr_src¥ 以下のフォルダを参照しています（プロジェクト毎に独立です）。

2. サンプルプログラムの構成

2.1. CAN モジュール種別

CAN モジュール	マイコン種	特徴
CAN	RX64M RX65 RX66N RX66T RX671 RX71M RX72M RX72N RX72T RA6E1 RA6M4 RA6M3 RA6M2 RA6M1 RA6T1 RA4E1 RA4M3 RA4M2 RA4M1 RA2A1 RA2L1	・「メールボックス」という単位でメッセージのやり取りを行う（FIFO もサポートされている） ・最大 3ch の CAN ポートをサポート[RX] ・最大 2ch の CAN ポートをサポート[RA] ・最大 1Mbps
RSCAN	RX24U RX24T RX231 RX23E-B RX140	・現行の RX200/100 系の CAN ポートは 1ch のみ ・送受信バッファ、または FIFO でメッセージのやり取りを行う ・最大 1Mbps
CANFD	RA6M5	・最大 2ch の CAN をサポート ・フレキシブルデータレート（データ通信速度を最大 8Mbps まで上げられる） ・送受信メッセージバッファ、または FIFO でメッセージのやり取りを行う
CANFD_2 (*1)	RA8E1 RA8M1 RA8T1	・CANFD のサブセット ・2ch の CAN をサポート ・最大 8Mbps
CANFD_B	RA6T2 RA6T3 RA6E2 RA4E2 RA4T1	・CANFD のサブセット ・現状 CAN-1ch ・最大 5Mbps
CANFD_Lite	RX660 RX261 RX26T	・CANFD のサブセット （CANFD_B と CANFD_Lite は類似性あり） ・現状 CAN-1ch ・最大 8Mbps[RX660,RX26T]，最大 5Mbps[RX261]

(*1)CANFD_2 という名称は、CANFD と区別するために付けた名称で公式のものではありません

CAN スタータキット RX/RA 取扱説明書 (CAN_STARTER_KIT_RXRA_REV_x_x_x_x_s.pdf)に CAN 種別に関しての記載がありますので、詳しくはそちらを参照してください。

RX600/700/RA（CAN モジュール）系のマイコンでは、「CAN モジュール」が搭載されています。また、RX24x/RX23x/RX100 系のマイコンでは「RSCAN モジュール」、RA6M5 では「CANFD モジュール」が搭載されています。その他 CANFD_2, CANFD_B と CANFD_Lite も存在します。これらのモジュールは、プログラム上の互換性はなく、モジュールに合わせたプログラムを組まなければなりません。本キットのサンプルプログラムでも、6 系統のモジュールでプログラムが別になっていますので、ターゲットとするマイコンが搭載しているモジュール種別に関しては意識するようにしてください。

RSCAN と CANFD(CANFD_2, CANFD_B, CANFD_Lite)は、バッファや FIFO の数は違いますが、同系統のモジュールです。（プログラムの構成は似ています）

なお、どのモジュール間でも通信は行えます。1 つの CAN バスに複数のタイプのモジュールが接続される事に関しては、なんら問題がありません。

CANFD（フレキシブルデータレート）をサポートしているモジュール(CANFD, CANFD_2, CANFD_B, CANFD_Lite)で CANFD パケットを送出した場合は、CANFD に対応していないモジュール(CAN, RSCAN)では受信できません。

2.2. ソースファイルフォルダ

※モジュール種別が増えたため、REV1.8 からフォルダ構成を変更しています, REV1.13 でフォルダ構成を変更

－RX－

SOURCE¥RX_CANST¥

以下に、実際のソースファイルが格納されています。

(1)共通

フォルダ	ファイル名	説明
source¥ALL¥common	board_settings.h	ボード名、CAN マクロ種別名、マイコンタイプ名の定義
	common.h	CAN で使用する各種定義
	can_operation.h	速度等の設定
	can_operation2.h	SAMPLE1~SAMPLE4 に依存する送受信方法等の設定
source¥ALL¥sci	sci.c	SCI(UART)ドライバ
	sci.h	

(2)マイコン毎の設定ファイル

フォルダ	ファイル名	説明
RXxxx¥usr_src¥settings	board.h	ボード固有の設定ファイル
	program_select.h	プログラム選択ヘッダ

RXxxx はプロジェクト名（RX**660_144_100** 等が入ります）。

※RX では settings フォルダは、CS+のプロジェクトフォルダ以下に存在しています

(a)CAN モジュール向け

フォルダ	ファイル名	説明
source¥CAN_module¥can	can.c	CAN モジュール CAN-ch 非依存の処理
	can.h	グローバル変数の定義
	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数
	can_general.h	受信リングバッファのアクセス関数等
	can_intr.c	CAN モジュール CAN-ch 非依存の割り込み関数
	can_intr.h	
source¥CAN_module¥can_ch0	can_ch0.c	CAN モジュール ch0 向けの関数
	can_ch0.h	
	can_ch0_intr.c	CAN モジュール ch0 向けの割り込み関数
	can_ch0_intr.h	
source¥CAN_module¥can_ch1	can_ch1.c	CAN モジュール ch1 向けの関数
	can_ch1.h	
	can_ch1_intr.c	CAN モジュール ch1 向けの割り込み関数
	can_ch1_intr.h	
source¥CAN_module¥can_ch2	can_ch2.c	CAN モジュール ch2 向けの関数
	can_ch2.h	
	can_ch2_intr.c	CAN モジュール ch2 向けの割り込み関数
	can_ch2_intr.h	

(a)CAN モジュール向け（続き）

フォルダ	ファイル名	説明
source¥CAN_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数

(b)RSCAN モジュール向け

フォルダ	ファイル名	説明
source¥RSCAN_module¥rscan	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数 受信リングバッファのアクセス関数等
	can_general.h	
	rscan.c	RSCAN モジュール CAN-ch 非依存の処理 グローバル変数の定義
	rscan.h	
	rscan_intr.c	RSCAN モジュール CAN-ch 非依存の割り込み関数
	rscan_intr.h	
source¥RSCAN_module¥rscan_ch0	rscan_ch0.c	RSCAN モジュール ch0 向けの関数
	rscan_ch0.h	
	rscan_ch0_intr.c	RSCAN モジュール ch0 向けの割り込み関数
	rscan_ch0_intr.h	
source¥RSCAN_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数

(c)CANFD_Lite モジュール向け

フォルダ	ファイル名	説明
source¥CANFDLite_module¥canfd_lite	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数 受信リングバッファのアクセス関数等
	can_general.h	
	canfd_lite.c	CANFD_Lite モジュール CAN-ch 非依存の処理 グローバル変数の定義
	canfd_lite.h	
	canfd_lite_intr.c	CANFD_Lite モジュール CAN-ch 非依存の割り込み関数
	canfd_lite_intr.h	
source¥CANFDLite_module¥canfd_lite_ch0	canfd_lite_ch0.c	CANFD_Lite モジュール ch0 向けの関数
	canfd_lite_ch0.h	
	canfd_lite_ch0_intr.c	CANFD_Lite モジュール ch0 向けの割り込み関数
	canfd_lite_ch0_intr.h	
source¥CANFDLite_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数
	main_s4.c	SAMPLE4 メイン関数

CAN モジュールのマイコンでは、(1)(2)と(a)に含まれるソースを使用します。RSCAN モジュールのマイコンでは(1)(2)(b)、CANFD_Lite モジュールのマイコンでは(1)(2)(c)の組み合わせでの使用となります。

RSCAN 及び CANFD_Lite で現状 2ch 以上の CAN をサポートするマイコンはありませんが、チャンネル非依存の処理と、チャネル依存の処理でソースを分けています。

－RA－

SOURCE¥RA_CANST¥

以下に、実際のソースファイルが格納されています。

(1)共通

フォルダ	ファイル名	説明
source¥ALL¥clock	clock.c	クロック設定
	clock.h	
source¥ALL¥common	board_settings.h	ボード名、CAN マクロ種別名、マイコンタイプ名の定義
	common.h	CAN で使用する各種定義
	can_operation.h	速度等の設定
	can_operation2.h	SAMPLE1~SAMPLE4 に依存する送受信方法等の設定
source¥ALL¥intr	intr.c	割り込み設定
	intr.h	
source¥ALL¥sci	sci.c	SCI(UART)ドライバ
	sci.h	

(2)マイコン毎の設定ファイル

フォルダ	ファイル名	設定ファイル
mcu¥RAxxx¥settings	board.h	ボード固有の設定ファイル
	program_select.h	プログラム選択ヘッダ

xxx には対応するマイコン名（RA**6M5** 等が入ります）。

(a)CAN モジュール向け

フォルダ	ファイル名	説明
source¥CAN_module¥can	can.c	CAN モジュール CAN-ch 非依存の処理
	can.h	グローバル変数の定義
	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数
	can_general.h	受信リングバッファのアクセス関数等
	can_intr.c	CAN モジュール CAN-ch 非依存の割り込み関数
	can_intr.h	
source¥CAN_module¥can_ch0	can_ch0.c	CAN モジュール ch0 向けの関数
	can_ch0.h	
	can_ch0_intr.c	CAN モジュール ch0 向けの割り込み関数
	can_ch0_intr.h	
source¥CAN_module¥can_ch1	can_ch1.c	CAN モジュール ch1 向けの関数
	can_ch1.h	
	can_ch1_intr.c	CAN モジュール ch1 向けの割り込み関数
	can_ch1_intr.h	

(a)CAN モジュール向け（続き）

フォルダ	ファイル名	説明
source¥CAN_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数

(b)CANFD モジュール向け

フォルダ		説明
source¥CANFD_module¥canfd	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数 受信リングバッファのアクセス関数等
	can_general.h	
	canfd.c	CANFD モジュール CAN-ch 非依存の処理 グローバル変数の定義
	canfd.h	
	canfd_intr.c	CANFD モジュール CAN-ch 非依存の割り込み関数
	canfd_intr.h	
source¥CANFD_module¥canfd_ch0	canfd_ch0.c	CANFD モジュール ch0 向けの関数
	canfd_ch0.h	
	canfd_ch0_intr.c	CANFD モジュール ch0 向けの割り込み関数
	canfd_ch0_intr.h	
source¥CANFD_module¥canfd_ch1	canfd_ch1.c	CANFD モジュール ch1 向けの関数
	canfd_ch1.h	
	canfd_ch1_intr.c	CANFD モジュール ch1 向けの割り込み関数
	canfd_ch1_intr.h	
source¥CANFD_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数
	main_s4.c	SAMPLE4 メイン関数

(c)CANFD_B モジュール向け

フォルダ		説明
source¥CANFD_B_module¥canfd	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数 受信リングバッファのアクセス関数等
	can_general.h	
	canfd_b.c	CANFD_B モジュール CAN-ch 非依存の処理，グローバル変数の定義
	canfd_b.h	
	canfd_b_intr.c	CANFD_B モジュール CAN-ch 非依存の割り込み関数
	canfd_b_intr.h	
source¥CANFD_B_module¥canfd_b_ch0	canfd_b_ch0.c	CANFD_B モジュール ch0 向けの関数
	canfd_b_ch0.h	
	canfd_b_ch0_intr.c	CANFD_B モジュール ch0 向けの割り込み関数
	canfd_b_ch0_intr.h	
source¥CANFD_B_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数
	main_s4.c	SAMPLE4 メイン関数

(d)CANFD_2 モジュール向け

フォルダ		説明
source¥CANFD_2_module¥canfd	can_error_handle.c	エラー処理
	can_error_handle.h	
	can_general.c	速度パラメータの設定関数 受信リングバッファのアクセス関数等
	can_general.h	
	canfd_2.c	CANFD_2 モジュール CAN-ch 非依存の処理，グローバル変数の定義
	canfd_2.h	
	canfd_2_intr.c	CANFD_2 モジュール CAN-ch 非依存の割り込み関数
	canfd_2_intr.h	
source¥CANFD_2_module¥canfd_2_ch0	canfd_2_ch0.c	CANFD_2 モジュール ch0 向けの関数
	canfd_2_ch0.h	
	canfd_2_ch0_intr.c	CANFD_2 モジュール ch0 向けの割り込み関数
	canfd_2_ch0_intr.h	
source¥CANFD_2_module¥canfd_2_ch1	canfd_2_ch1.c	CANFD_2 モジュール ch1 向けの関数
	canfd_2_ch1.h	
	canfd_2_ch1_intr.c	CANFD_2 モジュール ch1 向けの割り込み関数
	canfd_2_ch1_intr.h	
source¥CANFD_2_module¥main	main_monitor.c	モニタプログラムメイン関数
	main_s1.c	SAMPLE1 メイン関数
	main_s2.c	SAMPLE2 メイン関数
	main_s3.c	SAMPLE3 メイン関数
	main_s4.c	SAMPLE4 メイン関数

CAN モジュールのマイコンでは、(1)(2)と(a)に含まれるソースを使用します。CANFD モジュールのマイコンでは(1)(2)(b)、CANFD_B モジュールのマイコンでは(1)(2)(c)、CANFD_2 モジュールのマイコンでは(1)(2)(d)の組み合わせでの使用となります。

main 関数は、サンプルプログラム毎(SAMPLE***n***)にソースが分かれています。
main_s***n***.c

というファイル名になっています。

3. サンプルプログラムのビルド方法

サンプルプログラムは以下の手順でビルドを行ってください。

3.1. RX(CS+)

(1)マイコンボード（搭載マイコン種）に応じた CS+プロジェクトを開く

付属 CD の SOURCE¥RX_CANST 以下を、PC のドライブ上にコピーする。（以下コピー先を、[COPY 先]で記載します）

[COPY 先]¥RX_CANST¥プロジェクトフォルダ¥RXxxx.mtpj
（RX65_176 では、[COPY 先]¥RX_CANST¥RX65_176¥RX65_176.mtpj）
をダブルクリックする

(2)サンプルプログラム(SAMPLEn)を選択する

settings カテゴリに含まれる、program_select.h を開き、編集してください。

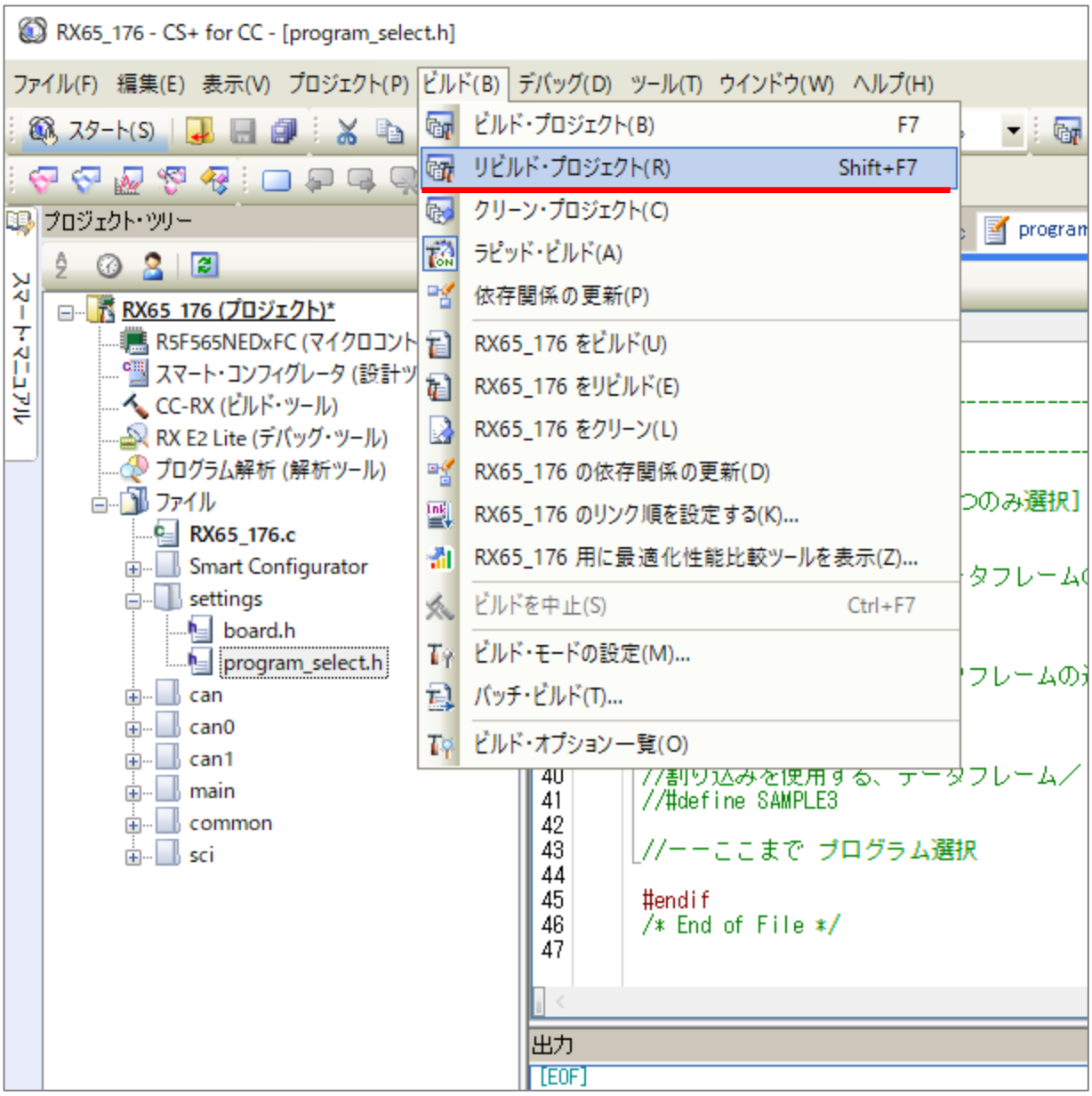
program_select.h

```
/*----------------------------------------------------------------------
定数定義
----------------------------------------------------------------------*/

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
//#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム，CANFD版 ※CAN-FD対応マイコンのみ
//#define SAMPLE4

//MONITOR
//CANパケット送受信のモニタ用プログラム
//#define SAMPLE_MONITOR

//ーーここまで プログラム選択
```

#define の行の内のいずれか 1 つのみ有効（行頭に//が無い状態）としてください。
（2 行以上を有効化することはできません）
無効にする場合は、行頭に//を入れるか削除してください。

※SAMPLE4 は CANFD 対応マイコンのみ

(3)リビルド・プロジェクトを実行



ビルドーリビルド・プロジェクト（もしくは、Shift+F7）で、プロジェクトをリビルドしてください。
（サンプル種の変更を行わない場合は、リビルドではなく通常のビルドで構いません）

ビルドの結果、コンパイル、リンクに問題がなければ、

[COPY 先]¥RX_CANST¥プロジェクト名¥DefaultBuild¥プロジェクト名.mot
（RX65_176 では、[COPY 先]¥RX_CANST¥RX65_176¥DefaultBuild¥RX65_176.mot)

が、ビルドで作成された MOT ファイルとなりますので、このファイルをマイコンボードに書き込んでください。
（書き込みの手順は、CAN スタータキット RX/RA のマニュアルの 6 章を参照してください）

(4)デバッガ接続（オプション）

ルネサスデバッガ（E1, E2, E2Lite, E20）をお持ちの場合は、デバッガをマイコンボード（もしくは USB-ADAPTER-RX14）の 14P コネクタに接続し、ビルド後、

デバッグーデバッグツールへダウンロード

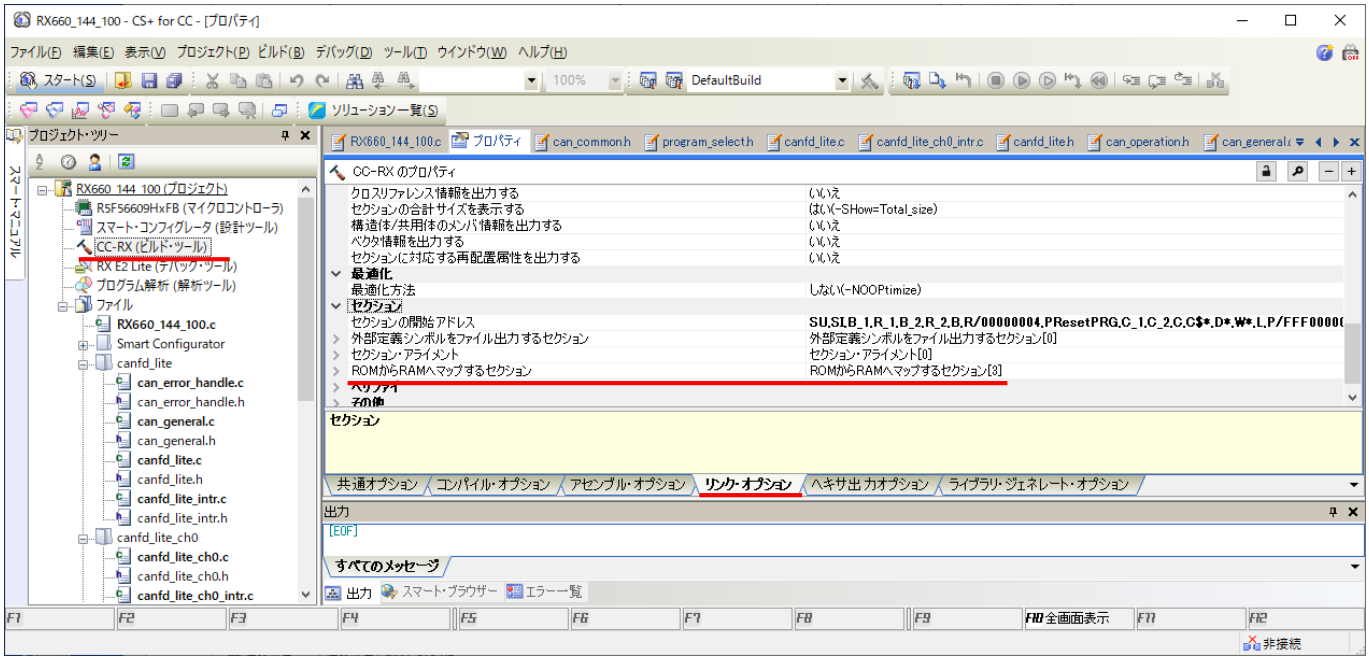
で、マイコンボードに対して、プログラムの書き込み（及びデバッグ）を行う事ができます。

デバッガをお持ちであれば、RenesasFlashProgrammer を使用した書き込みより、こちらの方が手早く行えるかと考えます。

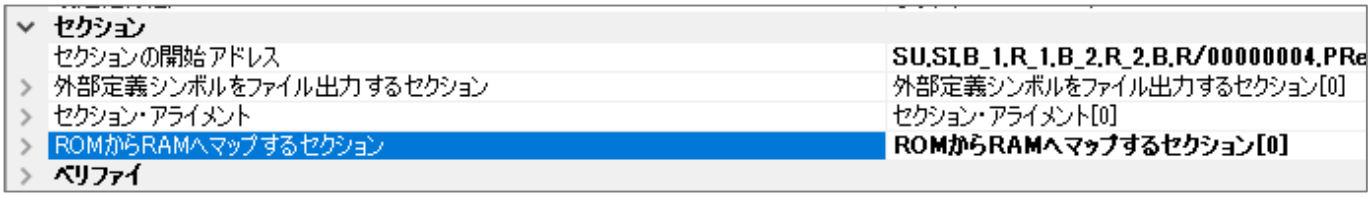
※プロジェクトのデバッガ設定値は「E2Lite」「FINE」に設定しています。
デバッガ　E2Lite　→接続するデバッガに応じて変更
接続方法　FINE　→RX600/700 系, RX26T のマイコンでは JTAG 接続も選択できますが、FINE を選択してください

－ビルドしたプログラムが全く動かないように見える場合－

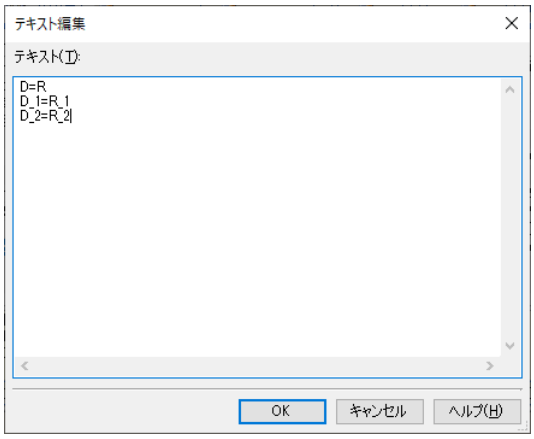
プログラム起動後、端末に文字表示がされない場合は、
CC-RX（ビルド・ツール） リンクオプションタブ　セクション－ROM から RAM へマップされるセクション



の欄を確認してください。



この部分が**[0]**になっていると、プログラムはまともに動作しません。

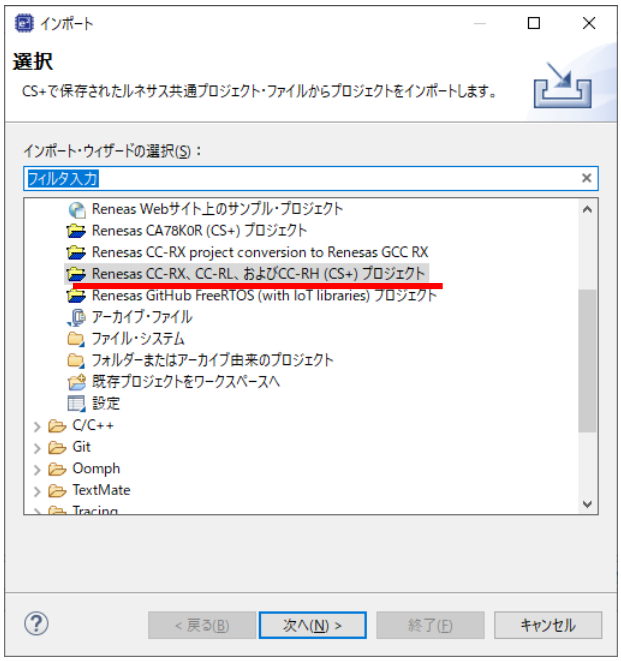


マイコンによっても異なりますが、上記の様な値が設定されているのが正しい状態です。（RX651/65N, RX72N 等ですと、もっと多数のマッピングが定義されています。）

3.2. RX(e2studio)

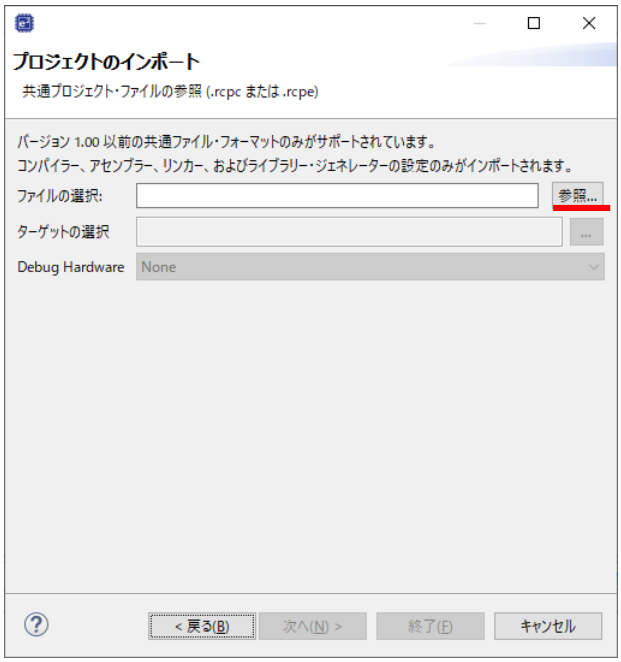
(1)CS+プロジェクトをインポートする

ファイルーインポート

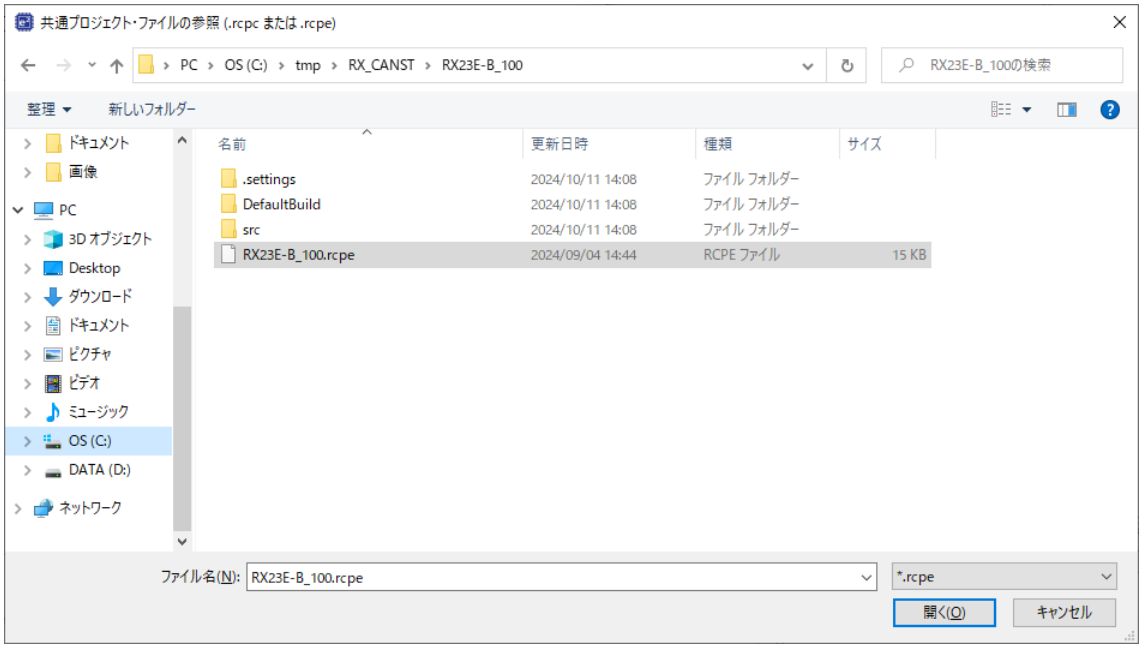


Renesas CC-RX...プロジェクトを選択して

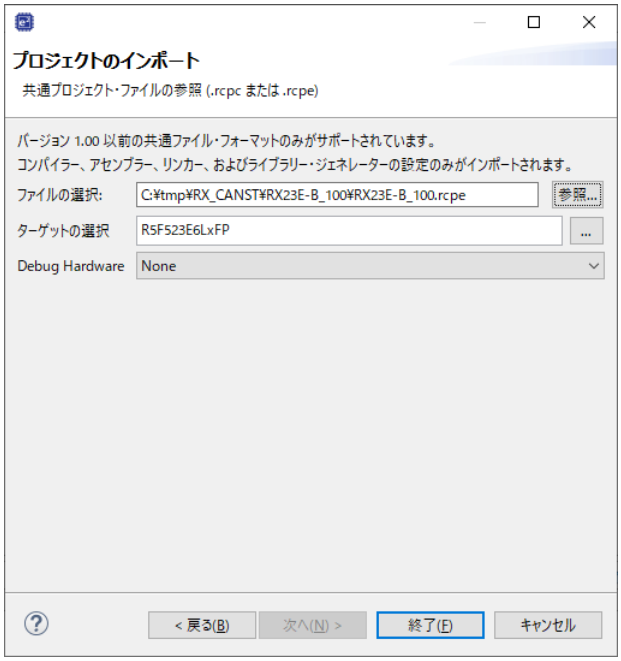
「次へ」



「参照」を押して、CS+プロジェクトに含まれる.rcpe ファイルを選択。

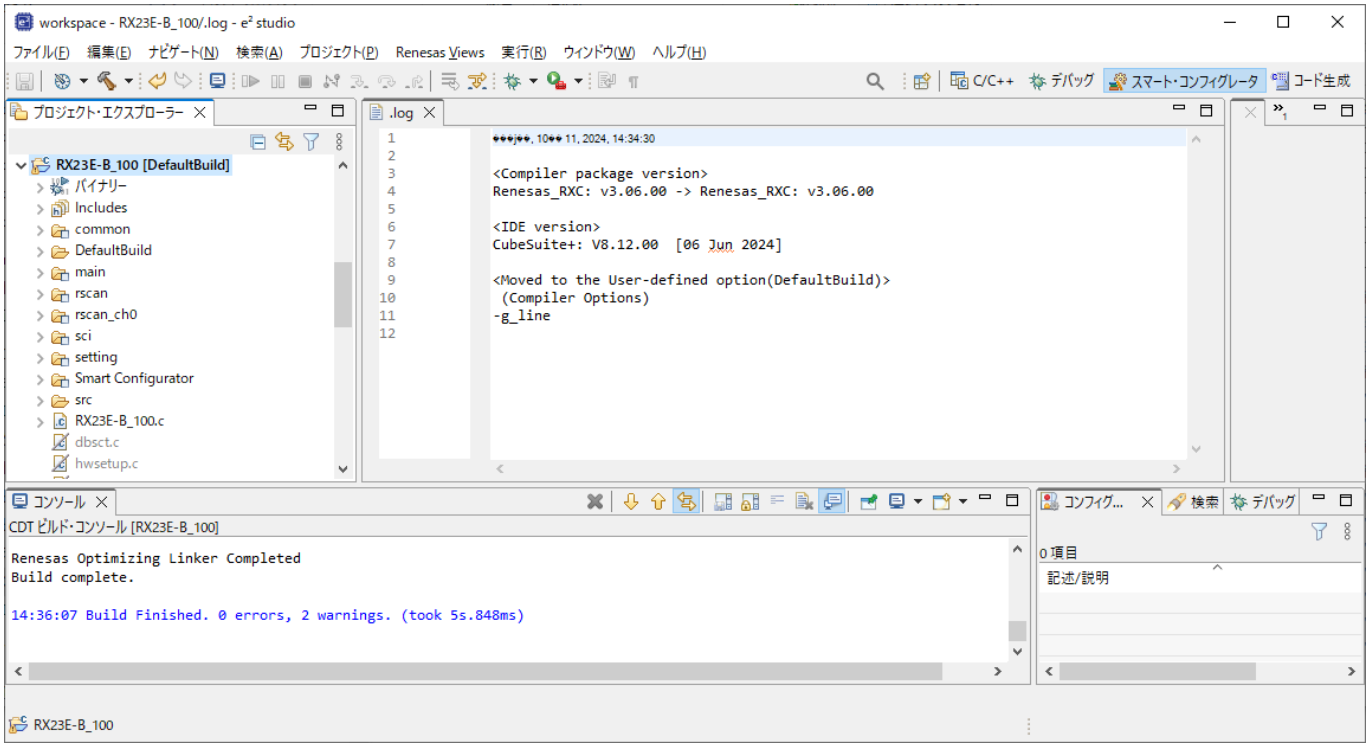


CS+プロジェクトに含まれる.rcpe ファイルを選択して「開く」。



「終了」を押す。

プロジェクトをビルドすると



インポートに問題が無ければ、ビルドが成功するはずです。

3.3. RA(e2studio)

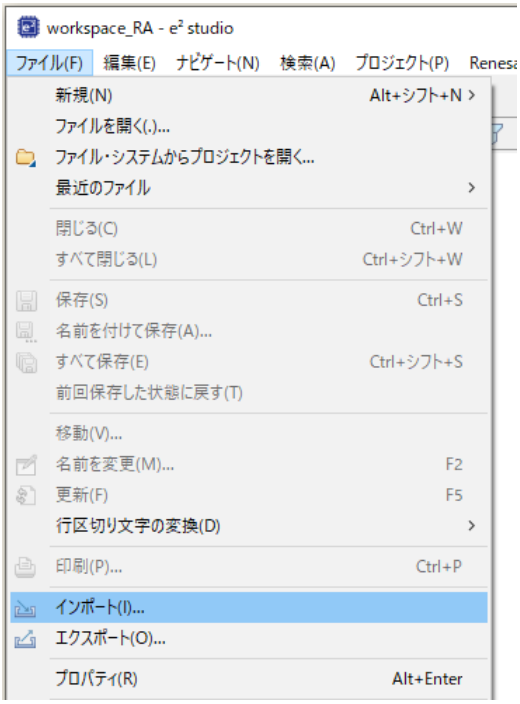
※REV1.11 から、e2studio のプロジェクトのアーカイブを CD 内に含める事と致しました

付属 CD の PROJECT¥RA_CANST¥RA***xxx***_CANST.zip

（***xxx*** には、マイコン名が入ります。RA6M5 等。）

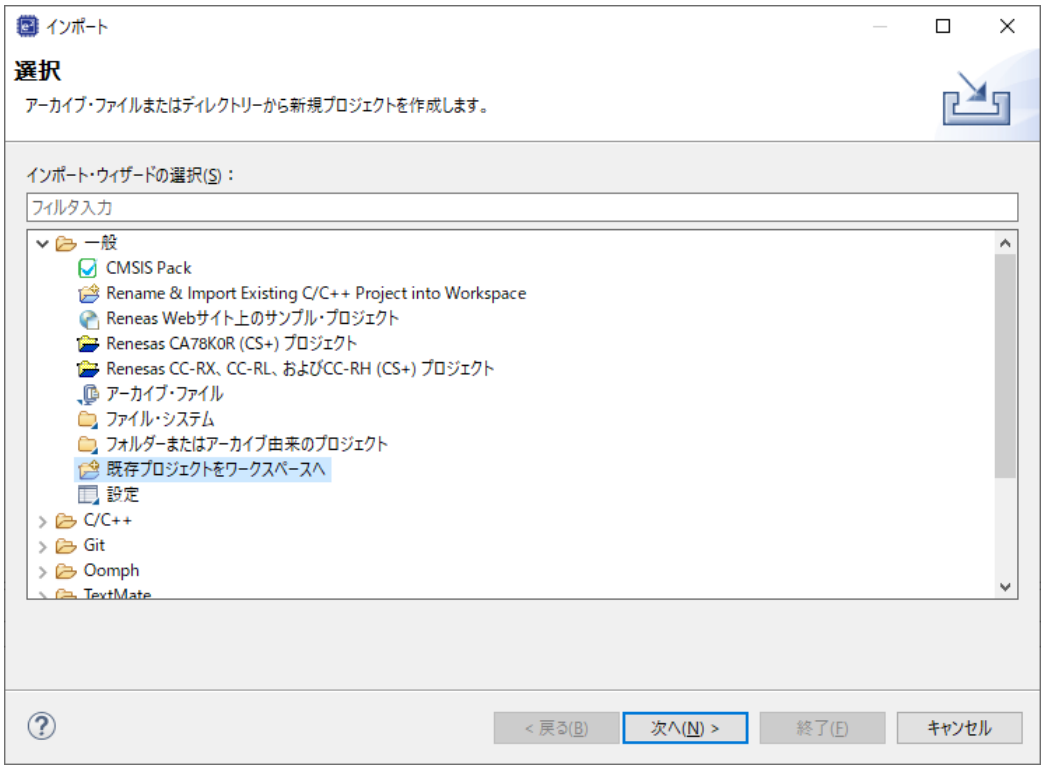
(1)マイコンボード（搭載マイコン種）に応じたプロジェクトをインポートする

e2studio を起動し、

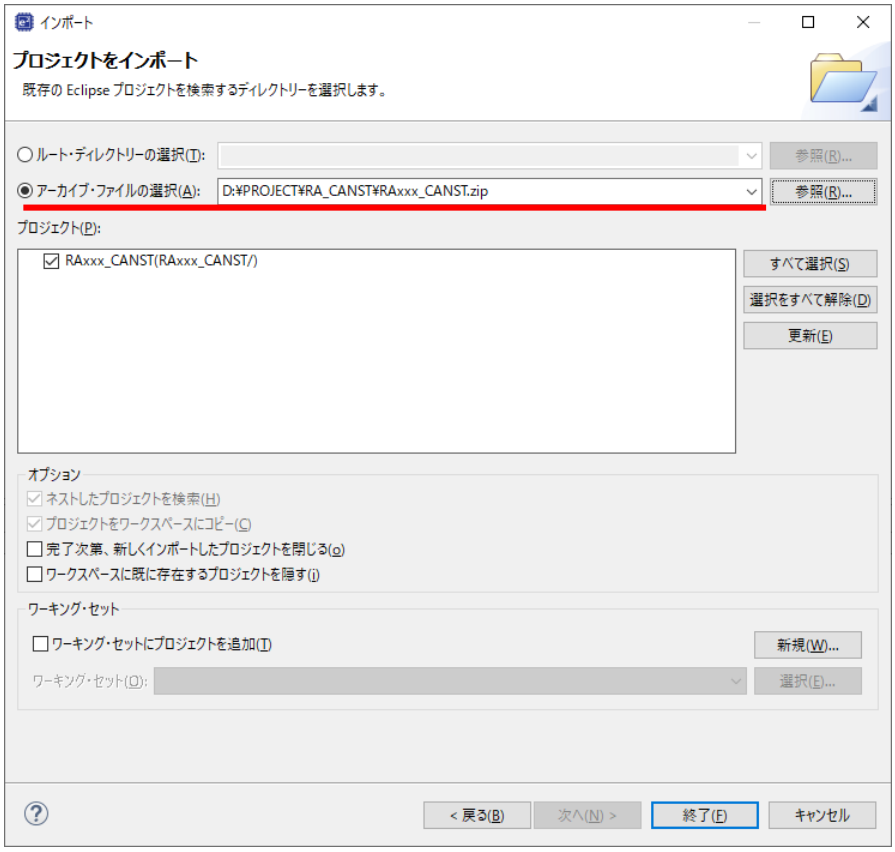


ファイルーインポート

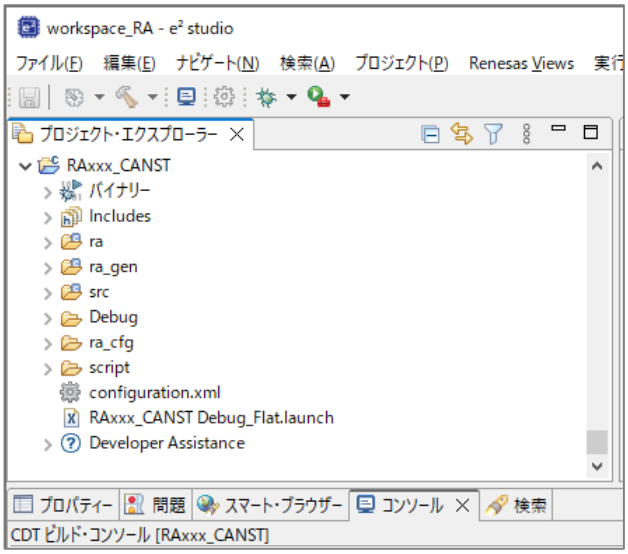
※e2studio は、e2studio 上に RA 向けの FSP をインストール
または、RA 向けに FSP と e2studio がセットになったものをインストールして使用してください



「既存プロジェクトをワークスペースへ」を選択「次へ」



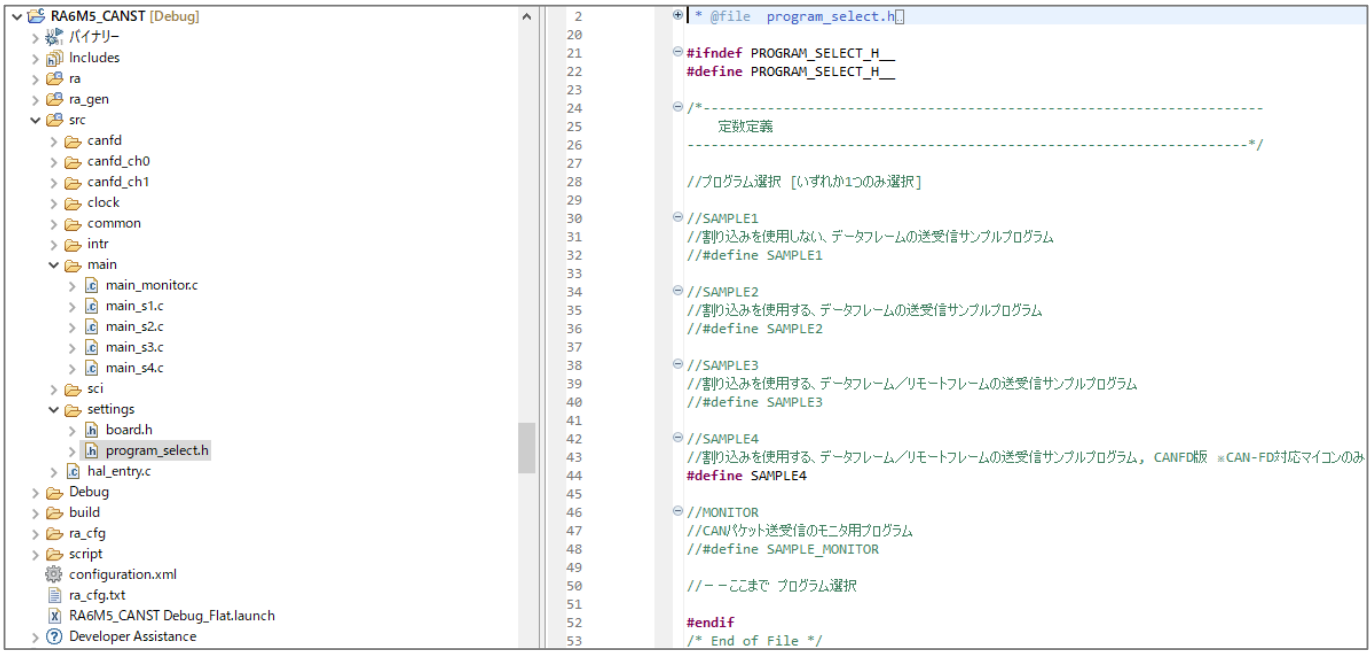
アーカイブ・ファイルの選択に、インポートを行う、RA***xxx***_CANST.zip を入力してください。「終了」を押す。



プロジェクト・エクスプローラ上に、RAxxx_CANST プロジェクトがインポートされます。(RX***xxx*** は RA**6M5** 等です)

(2)サンプルプログラム(SAMPLEn)を選択する

settings フォルダに含まれる、program_select.h を開き、編集してください。



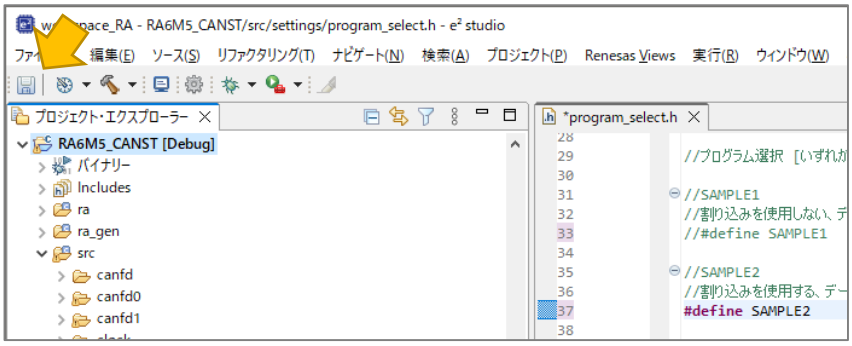
#define SAMPLE***n*** の行の内のいずれか 1 つのみ有効(行頭に//が無い状態)としてください。
(2 行以上を有効化することはできません)
無効にする場合は、行頭に//を入れるか削除してください。

program_select.h

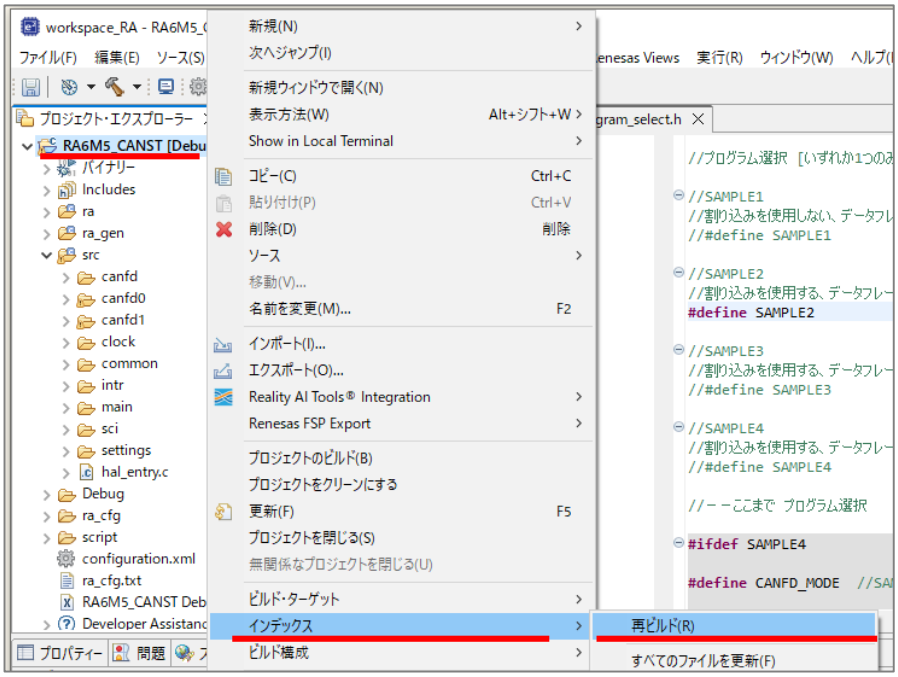
```
/*---------------------------------------------------------------------
    定数定義
---------------------------------------------------------------------*/

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
//#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
//#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム，CANFD版 ※CAN-FD対応マイコンのみ
#define SAMPLE4

//MONITOR
//CANパケット送受信のモニタ用プログラム
//#define SAMPLE_MONITOR

//ーーここまで プログラム選択
```

※SAMPLE4 は CANFD 対応プロジェクトのみ

(3)プログラムのビルド



編集後、ファイル(program_select.h)を保存（フロッピーのアイコン、もしくは Ctrl-S）してください。



SAMPLEn(n=1~4)を変更した際や #define の定数を変更した際は、

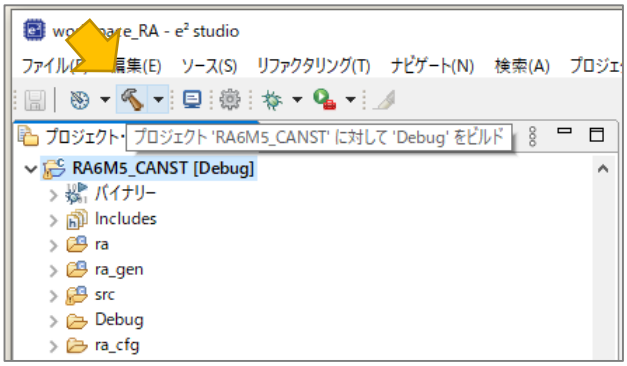
インデックス　－　再ビルド

を実行してください。

プロジェクト名(RA***xxx***_CANST)を右クリック、メニューを開き

インデックス－再ビルド

を実行してください。（#define 文を変更した際は、本操作を行うと、#define の変更が即時反映されます。本操作は省略可能ですが、#define を変えてもソースの有効範囲（グレーアウトしていない部分）が変わらない場合は、本操作を行ってください。）



トンカチのアイコンを押す（もしくは、プロジェクト名を右クリックープロジェクトのビルド）。

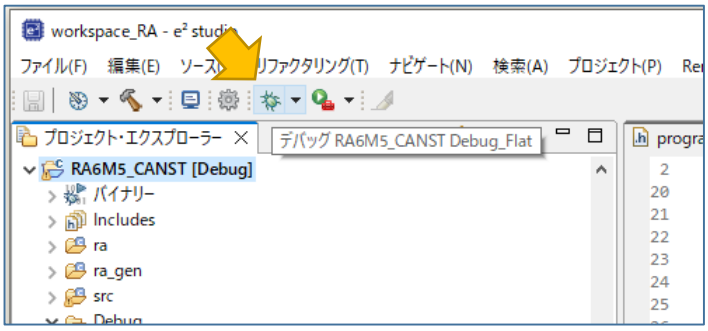
ビルドの結果、コンパイル、リンクに問題がなければ、

[Workspace]¥RA***xxx***_CANST¥Debug¥RA***xxx***_CANST.srec
（RA***xxx***_CANST の部分は[プロジェクトフォルダ]で、プロジェクト名と同じとなります）

が、ビルドで作成された MOT ファイル（Motrola-S レコードファイル）となりますので、このファイルをマイコンボードに書き込んでください。
（書き込みの手順は、CAN スタータキット RX/RA のマニュアルの 6 章を参照してください）

(4)デバッガ接続（オプション）

ルネサスデバッガ（E2, E2Lite）をお持ちの場合は、デバッガをマイコンボードの 14P コネクタ（もしくは、オプションの 20P ハーフピッチコネクタ）に接続し、ビルド後、



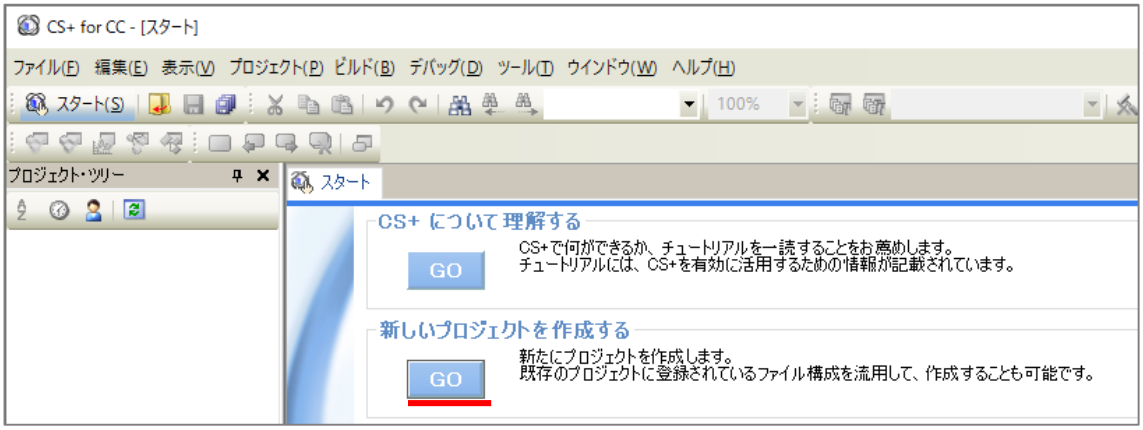
虫のアイコンをクリックして、デバッガ接続を行ってください。

※デバッガ設定は、E2Lite を選択しています。E2Lite 以外のデバッガ使用時は変更してください。
E2 など JTAG に対応したデバッガ使用時、接続方式は SWD を選択してください。

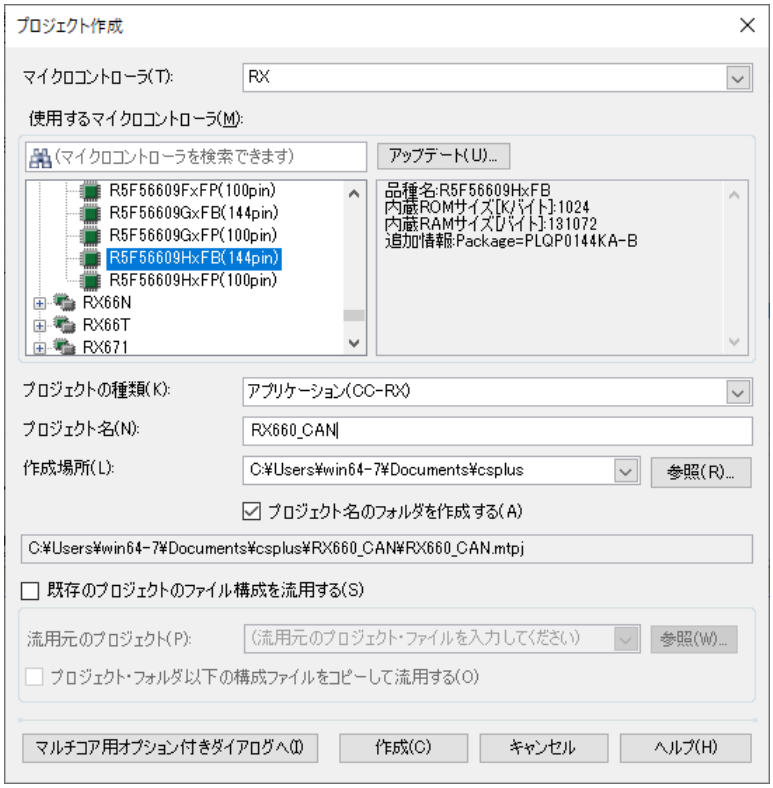
4. サンプルプログラムのプロジェクトを作成する手順

4.1. RX(CS+)

(1)マイコンボード（搭載マイコン種）に応じたプロジェクトを作成

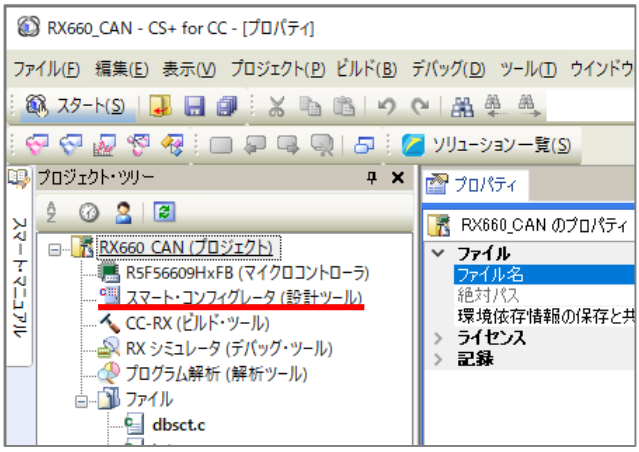


新しいプロジェクトを作成する。

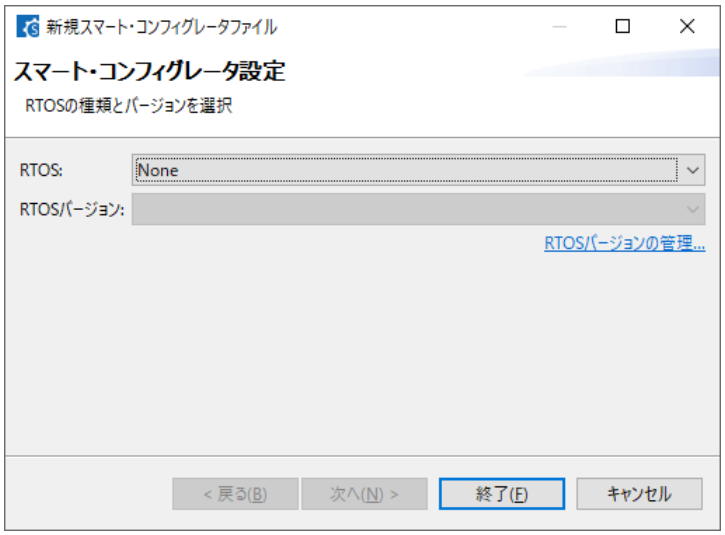


ターゲットマイコンの選択、プロジェクト名の設定等を行い、「作成」を押す。

(2)スマート・コンフィグレータでクロックの設定を行う



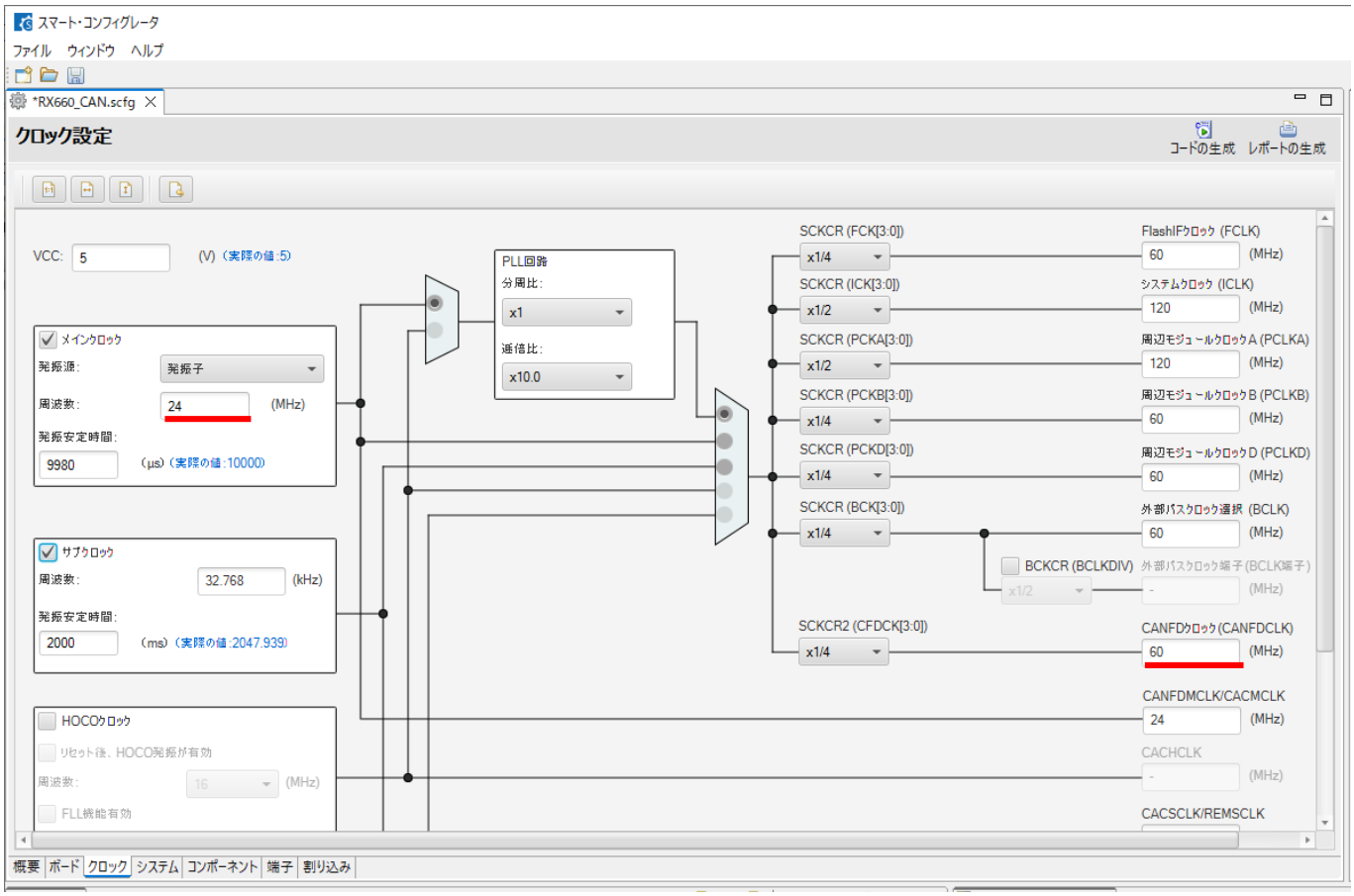
スマート・コンフィグレータをダブルクリックで起動。
（RX スマート・コンフィグレータを使用します。RX スマート・コンフィグレータは CS+とは別に、インストールが必要です。）



RTOS の選択は、基本的には None を選択して「終了」。


スマート・コンフィグレータ
ファイル　ウィンドウ　ヘルプ
RX660_CAN.scfg
概説
機能概要
概要
概要をクリックすると、スマート・コンフィグレータの機能を確認することができます。
動画
スマート・コンフィグレータの紹介
関連動画
最新情報
最新情報をクリックすると、最新リリースの情報を確認することができます。
現在の設定状態
使用しているボード/デバイス: R5F56609HxFB (ROM size: 1 MB, RAM size: 128 KB, Pin count: 144)
生成先ロケーション (PROJECT_LOC¥):
src¥smc_gen
編集...
使用しているコンポーネント:
コンポーネント
バージョン
設定
Board Support Packages. (r_bsp)
7.40
r_bsp(使用中)
概要
ボード
クロック
システム
コンポーネント
端子
割り込み
コンソール

「クロック」タブを選択。



メインクロックの周波数は、ボード搭載の水晶振動子の周波数を入力。

※RX660 の場合は、CANFD クロックが 60MHz になる様に設定

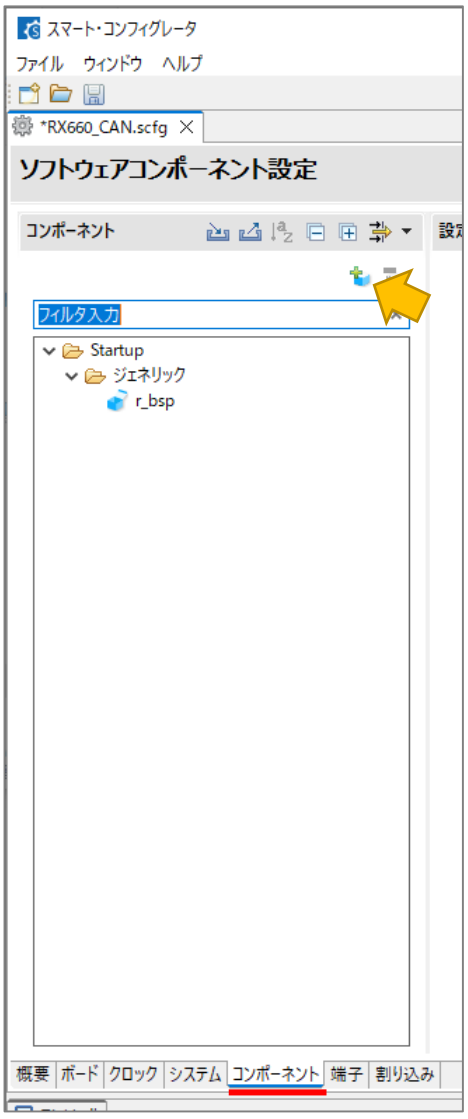
（CAN のクロックは、マイコン種毎に異なります。PCLKB を使用するマイコンや、メインクロックを使用するマイコン等。）

サンプルプログラムの規定値で使用する場合、下記値を参考に設定してください。

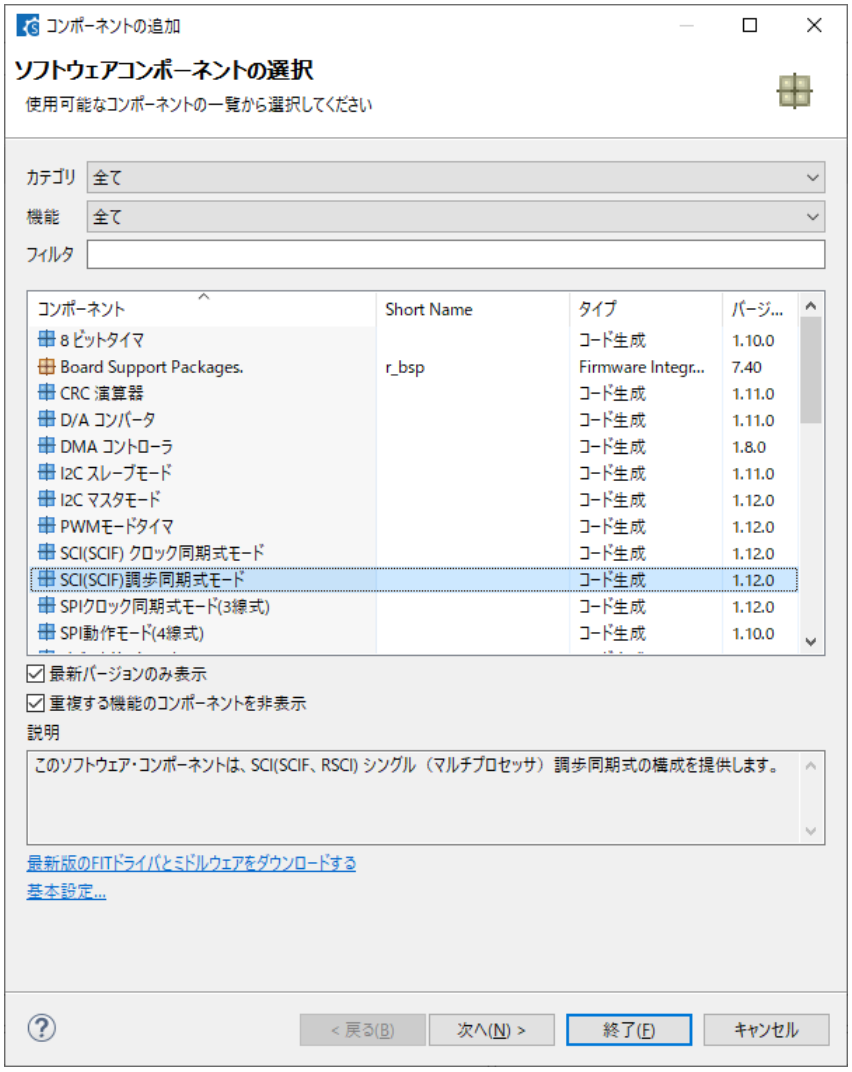
マイコン種	搭載水晶振動子 MainOSC	CAN クロックソース（周波数）
RX71M	24MHz	PCLKB(60MHz)
RX72M	24MHz	PCLKB(60MHz)
RX72N	24MHz	PCLKB(60MHz)
RX72T	24MHz	PCLKB(50MHz)
RX64M	24MHz	PCLKB(60MHz)
RX65/RX671	24MHz	PCLKB(60MHz)
RX66N	24MHz	PCLKB(60MHz)
RX66T(100A/100B)	20MHz	PCLKB(40MHz)
RX66T(144)	24MHz	PCLKB(40MHz)
RX660	24MHz	CANFDCLK(60MHz)
RX231	8MHz	MainOSC(8MHz)
RX23E-B	8MHz	PCLKB/2(16MHz)
RX24T	10MHz	PCLKB/2(20MHz)
RX24U	10MHz	PCLKB/2(20MHz)
RX261	8MHz	CANFDCLK(32MHz]
RX26T	24MHz	CANFDCLK(60MHz)
RX140	8MHz	PCLKB/2(12MHz)

※画面のハードコピーでは、サブクロックを有効化していますが、CAN のプログラムではサブクロックは未使用です。

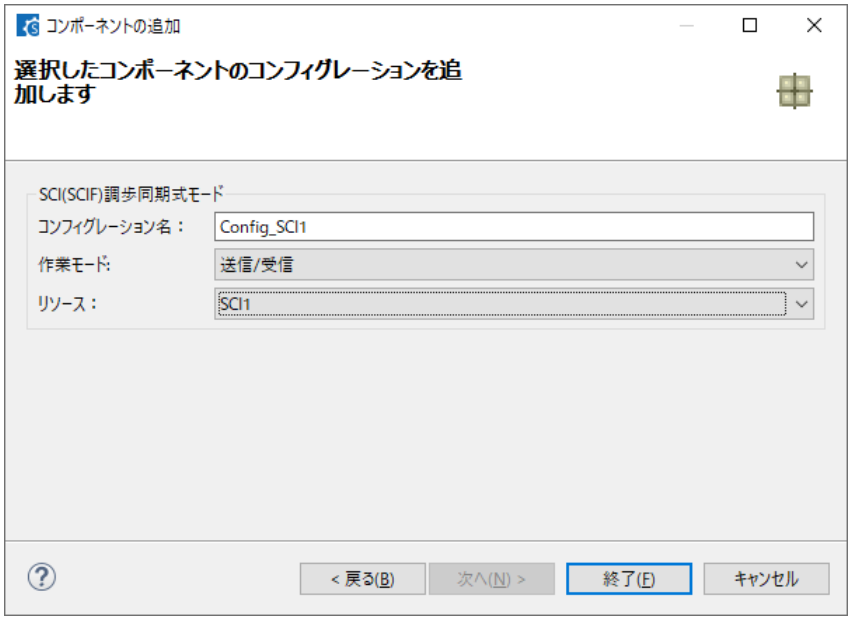
(3)スマート・コンフィグレータでコンポーネントの追加を行う



「コンポーネント」タブ、上方のコンポーネント追加アイコンをクリック。



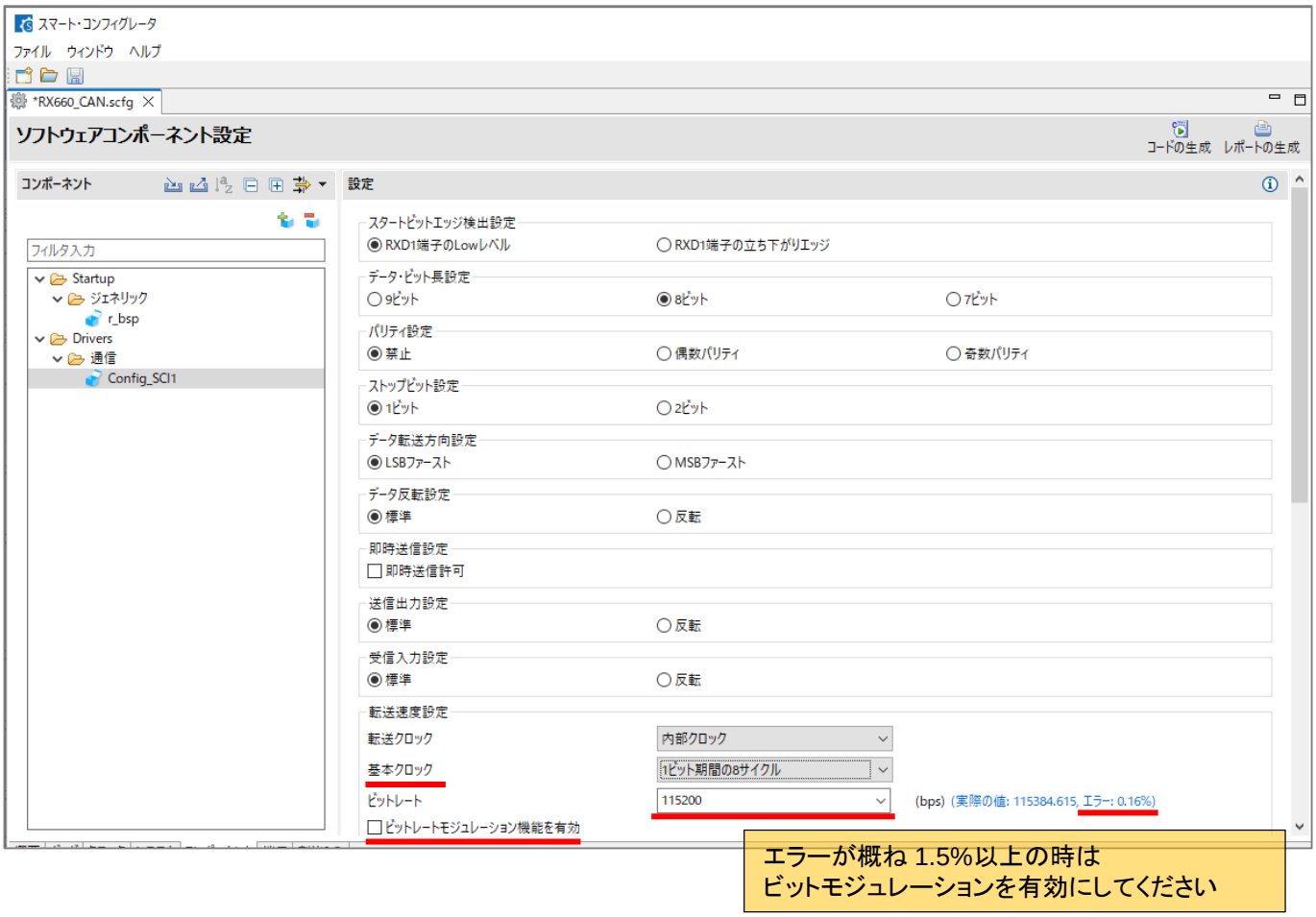
SCI(SCIF)調歩同期式モードを選択し「次へ」。



「送信/受信」

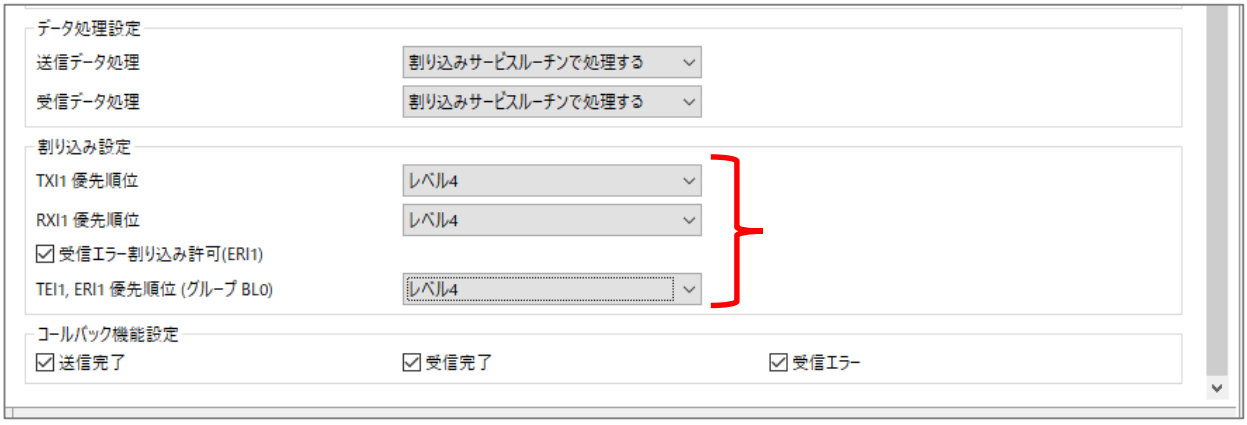
「SCI1」

を選択し、「終了」。

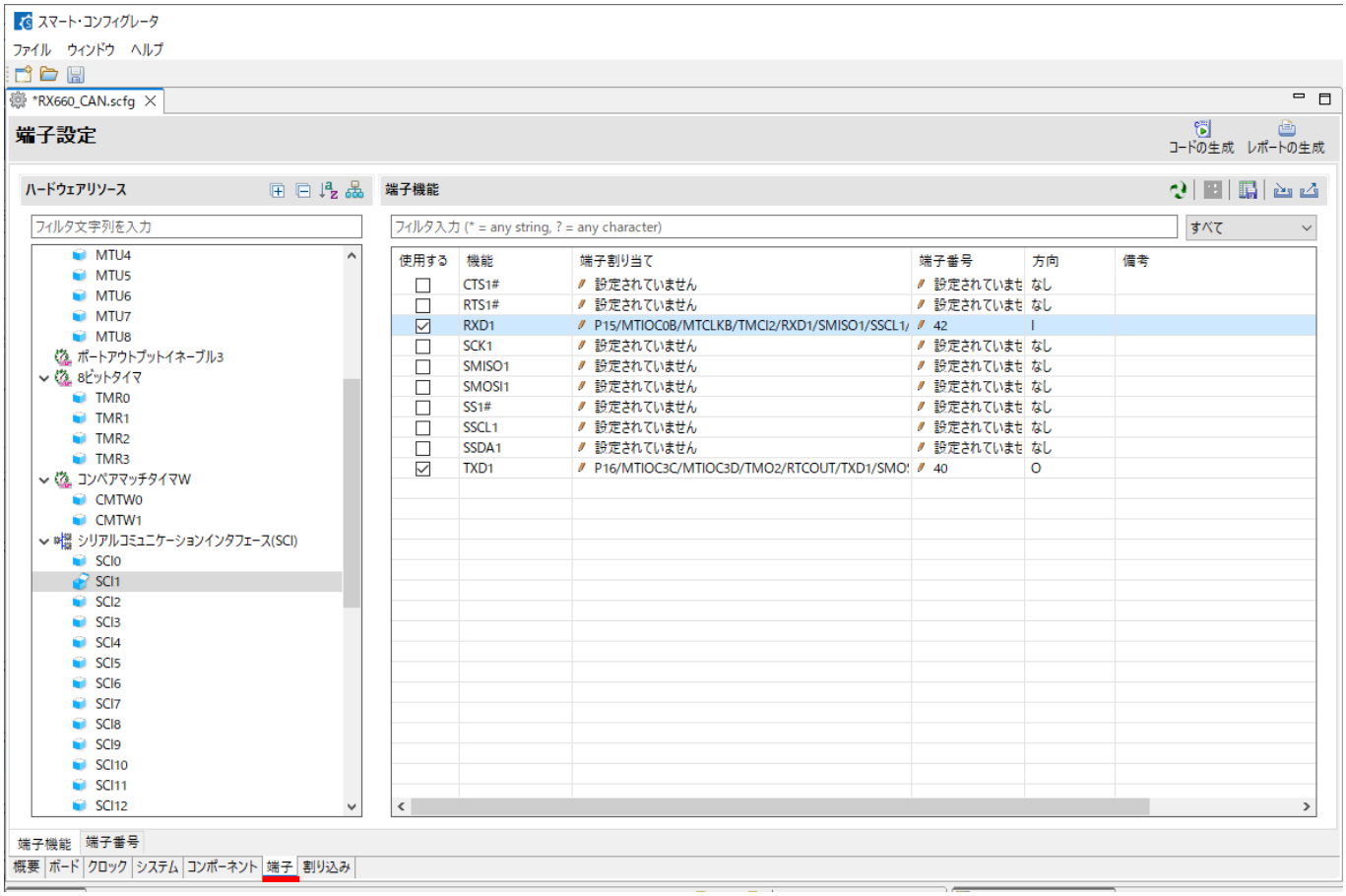


ビットレートを「115200」と入力してください。（速度設定値は任意の値に設定頂いても問題ありません。）

（「基本クロック」、「ビットモジュレーション機能を有効」の部分は、表示されるエラー:?%の値を見て、ある程度誤差が小さな値となる様に調整してください。）（※1.5%未満の誤差であれば概ね問題ないと考えます）



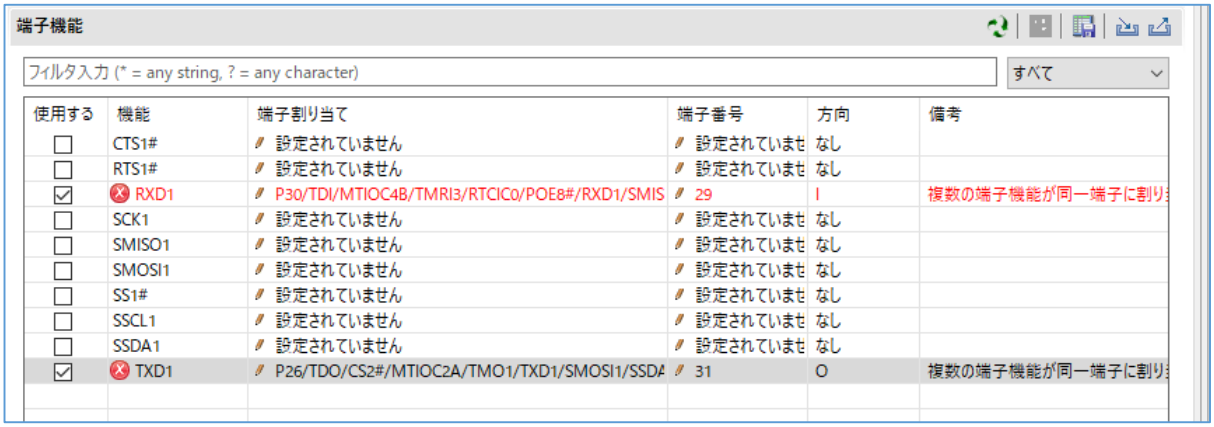
設定値は任意ですが、サンプルプロジェクトでは、割り込みの優先度をデフォルト（最高優先度）から下げています。



「端子」タブ。

RXD1 と TXD1 の端子を、14P コネクタに結線されている信号に合わせる様に変更します。
（変更が不要な場合もあります）

本ハードコピーの RX660 では、デフォルトでは P15, P16 が選択されていますが、これを、P30, P26 に変更します。



[参考　HSBRX660-144H の取扱説明書]

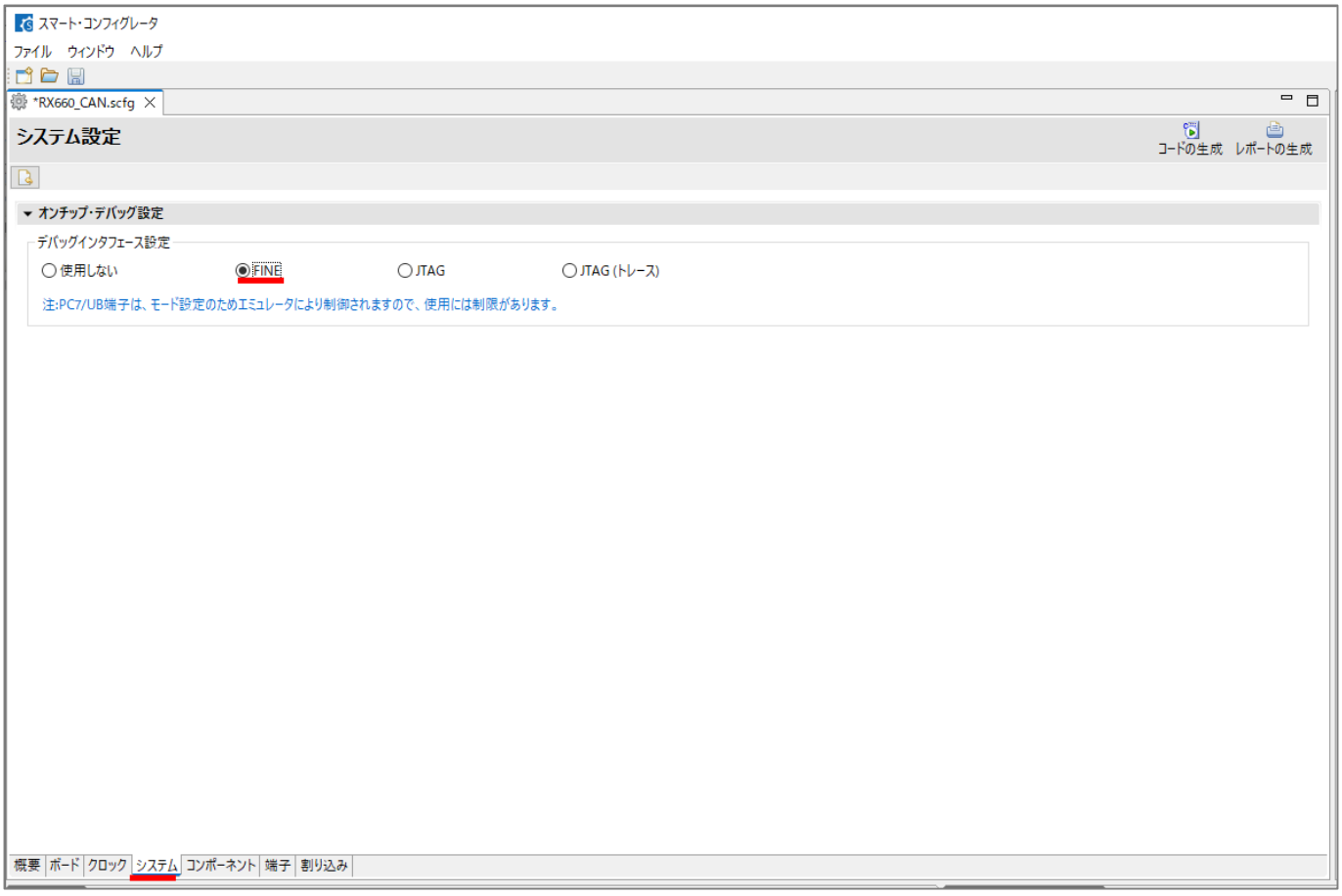
表 2-1a エミュレータインタフェース信号表 (J5) [HSBRX660-144H]

No	マイコンピン番号	信号名	No	マイコンピン番号	信号名
1	30	P27/TCK	2	-	VSS
3	25	P34/*TRST	4	10	EMLE
5	**31**	**P26/TDO/TXD1**	6	-	(NC)
7	16	MD/FINED	8	-	VCC
9	28	P31/TMS	10	60	PC7/UB
11	**29**	**P30/TDI/RXD1**	12	-	VSS
13	19	*RESET	14	-	VSS

マイコンボードの 14P コネクタの 5 番ピン(TXD1), 11 番ピン(RXD1)が選択される様に変更します。

RX700/600 シリーズですと、TXD1/TDO, RXD1/TDI となっている端子を選択します。

但し、この場合は、スマート・コンフィグレータ側で、×（端子割り当て機能の重複）が指摘されています。

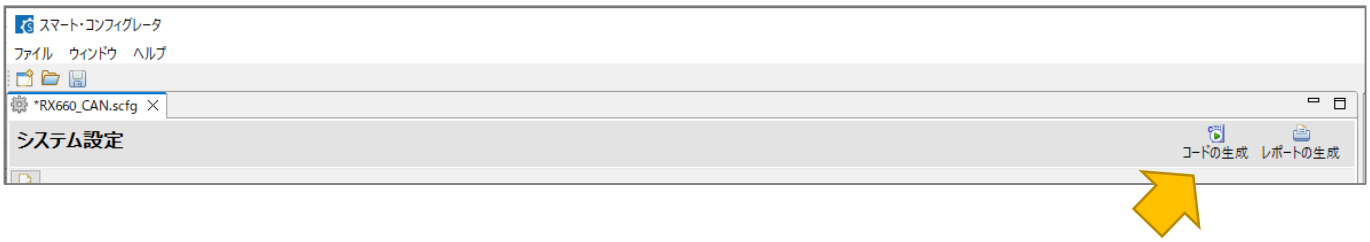


「システム」タブのオンチップ・デバッグ設定で、「FINE」を選んでください。

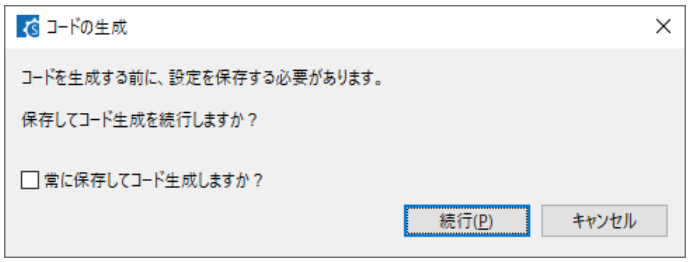
→先程の端子の選択エラーは解消されます

※RX700/600/26T の JTAG 対応のマイコンのみ本設定が必要です

(4)スマート・コンフィグレータでコード生成を行う

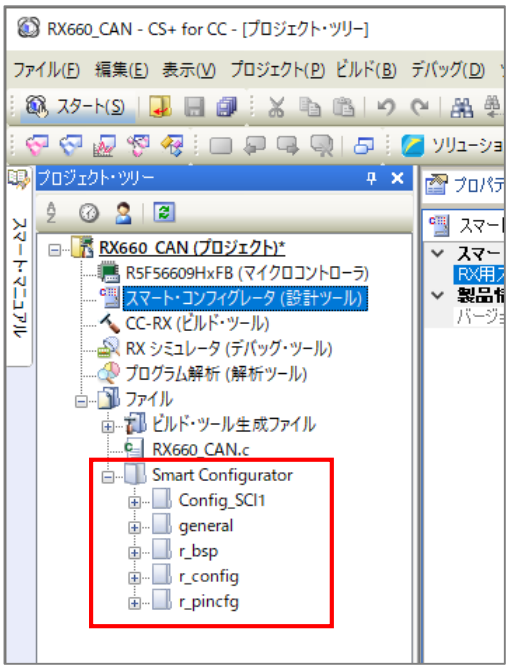


「コード生成」ボタンを押してください。



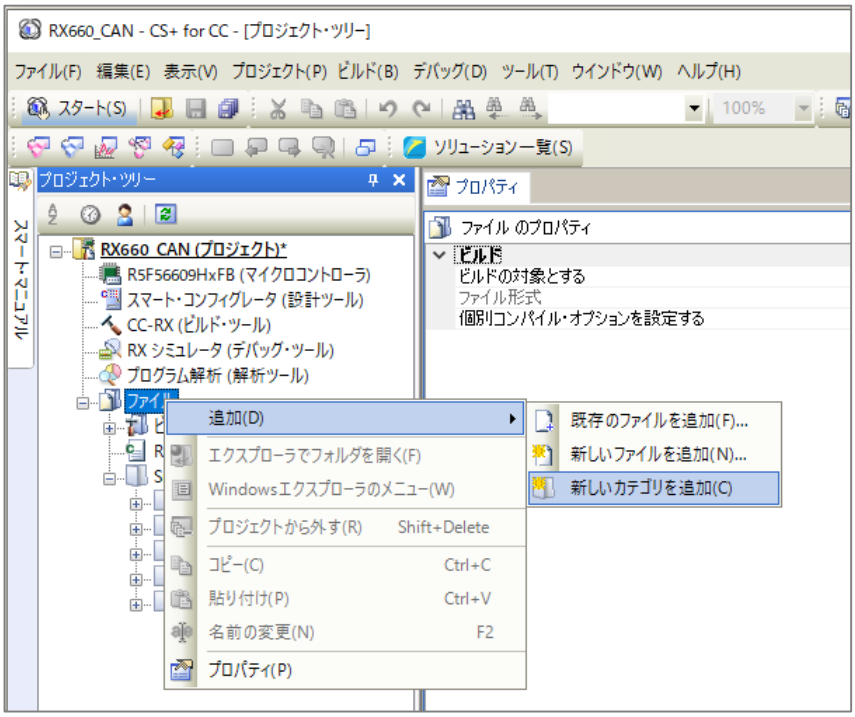
上記画面が出た場合は「続行」。

上記画面が出た場合は「はい」。

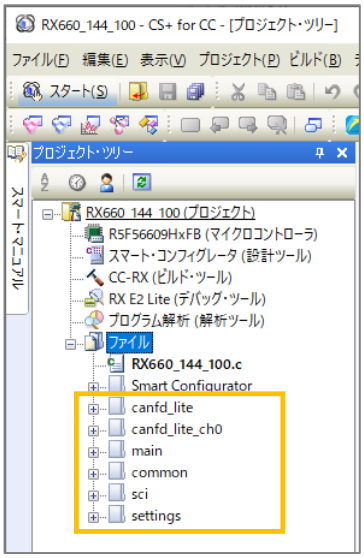


コード生成により、CS+のプロジェクト・ツリーに、スマート・コンフィグレータ生成コードが追加されます。

(5)ソースコードをプロジェクトに追加する



「ファイル」を右クリック追加ー新しいカテゴリを追加



－追加するカテゴリー－

マイコンタイプ	作成するカテゴリ	備考
CAN モジュール対応マイコン	can	
	can_ch0	
	can_ch1	
	can_ch2	CAN-ch2 非搭載のマイコンでも作成してください
	main	
RSCAN モジュール対応マイコン	rscan	
	rscan_ch0	
	main	
CANFD_Lite モジュール対応マイコン	canfd_lite	
	canfd_lite_ch0	
	main	

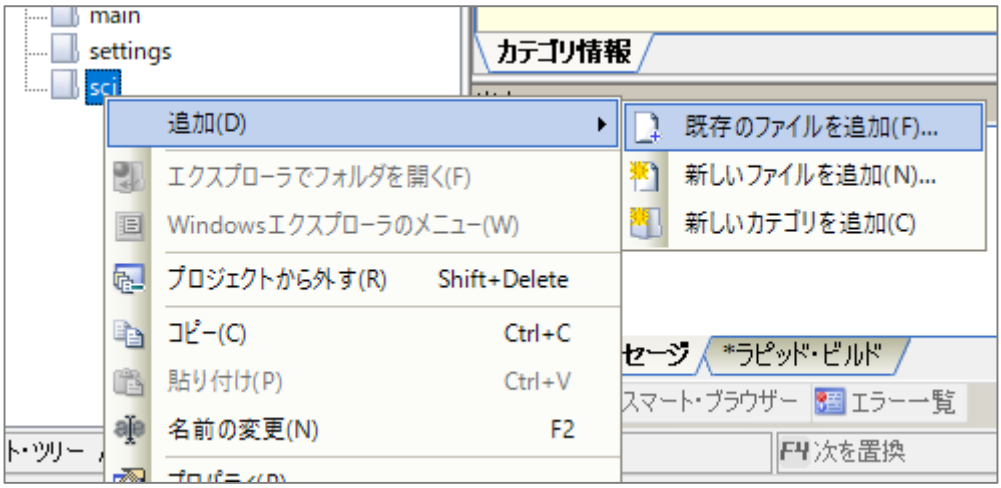
－マイコン種に拠らず追加するカテゴリー－

マイコンタイプ	作成するカテゴリ	備考
全てのマイコン	common	
	sci	

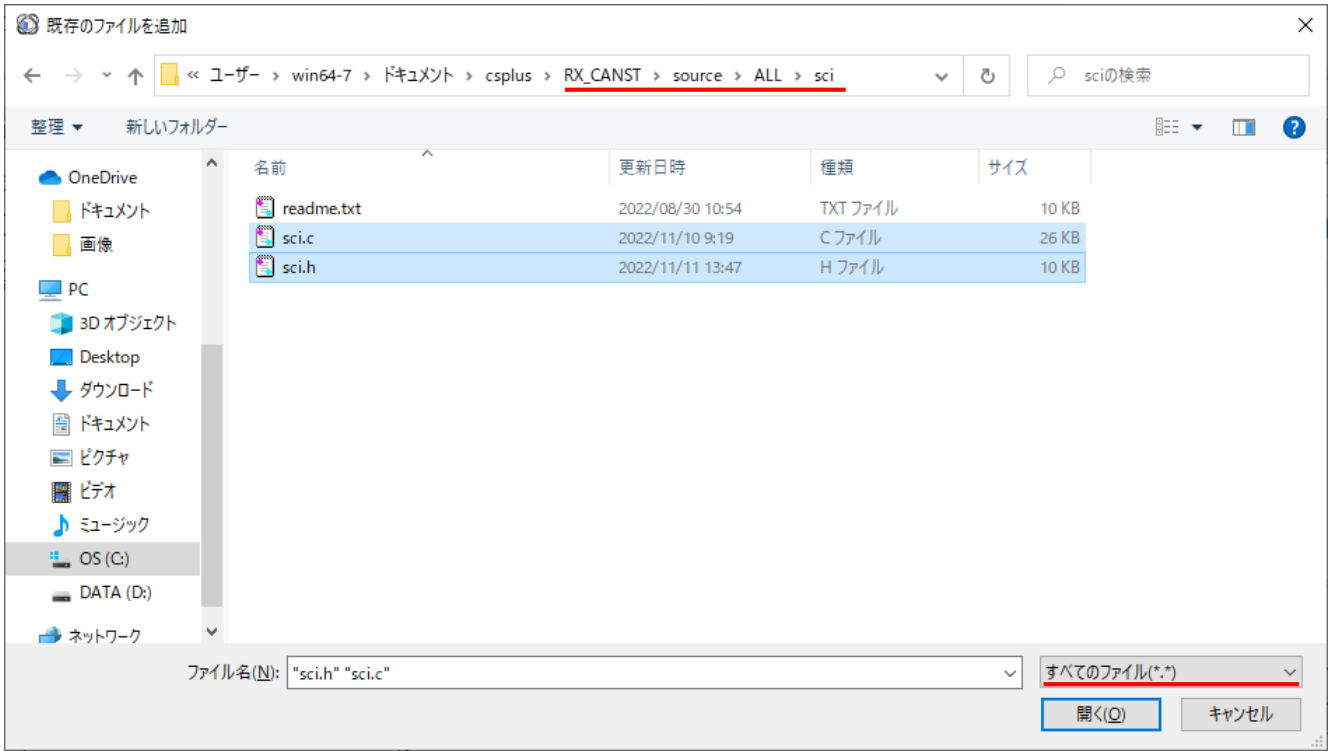
マイコンタイプ	作成するカテゴリ	備考
	settings	

上記のカテゴリ追加後、ソースをカテゴリ内に追加します。

※本資料では、追加するカテゴリ名は、ソースコードが含まれるフォルダ名と同一としています
（任意のカテゴリ名に変更頂いても問題はありません）



カテゴリ（上記では sci を選択）右クリック－追加－既存のファイルを追加



「すべてのファイル」を選択し、

CD 内では、
¥SOURCE¥RX_CANST¥source¥ALL¥sci

※CD を取り外すとソースファイルが見えなくなるので、PC のストレージにコピーして、コピー先のファイルを指定してください

内の、（この例では sci フォルダ内の）.c, .h を全て選択して「開く」。

ーカテゴリ内に追加するソースコードー

・CAN モジュール対応マイコン

カテゴリ名	追加するファイルの格納先 CD 内の¥SOURCE¥RX_CANST¥
can	source¥CAN_module¥can
can_ch0	source¥CAN_module¥can_ch0
can_ch1	source¥CAN_module¥can_ch1
can_ch2	source¥CAN_module¥can_ch2 (*1)
main	source¥CAN_module¥main

(*1)CAN-ch2 をサポートしないマイコンでもソースコードの追加を行ってください
（メイン関数で ch2 の関数を参照しているため、can_ch2 のソースコードが見えないと、リンク時に「未定義の関数」のエラーが出ます。ご自身で CAN-ch2 をサポートしないマイコン向けのプログラムを作成する場合は、追加する必要はありません。）

・RSCAN モジュール対応マイコン

カテゴリ名	追加するファイルの格納先 CD 内の¥SOURCE¥RX_CANST¥
rscan	source¥RSCAN_module¥can
rscan_ch0	source¥RSCAN_module¥can_ch0
main	source¥RSCAN_module¥main

・CANFD_Lite モジュール対応マイコン

カテゴリ名	追加するファイルの格納先 CD 内の¥SOURCE¥RX_CANST¥
canfd_lite	source¥CANFDLite_module¥ anfd_lite
canfd_lite_ch0	source¥CANFDLite_module¥ canfd_lite_ch0
main	source¥CANFDLite_module¥main

※main は、モジュール毎に異なります（モジュール毎のフォルダ以下の main を追加）。

・全てのマイコン

カテゴリ名	追加するファイルの格納先 CD 内の¥SOURCE¥RX_CANST¥
common	source¥ALL¥common
sci	source¥ALL¥sci

common, sci は全モジュールで共通（ALL 以下のファイルを追加）。

・全てのマイコン

カテゴリ名	追加するファイルの格納先 CD 内の¥SOURCE¥RX_CANST¥
settings	RX***xxx***¥src¥users¥settings (*2)

RX***xxx*** の部分はマイコン種毎。
(*2)SmartRX マイコンボードの場合、RXxxx の部分が SmartRX になります

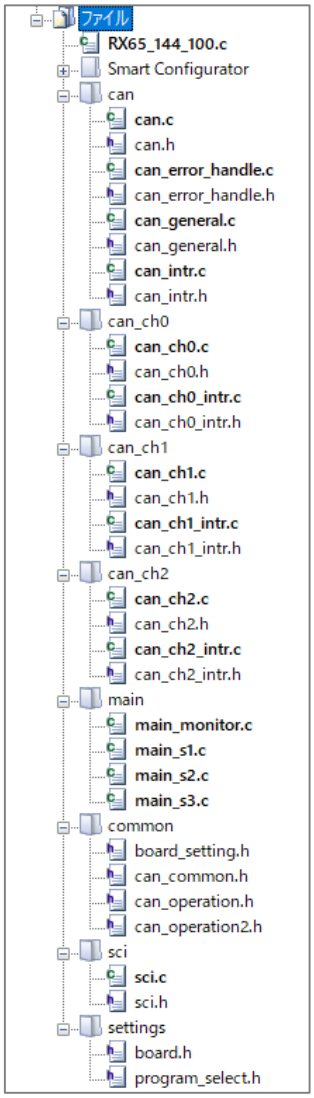
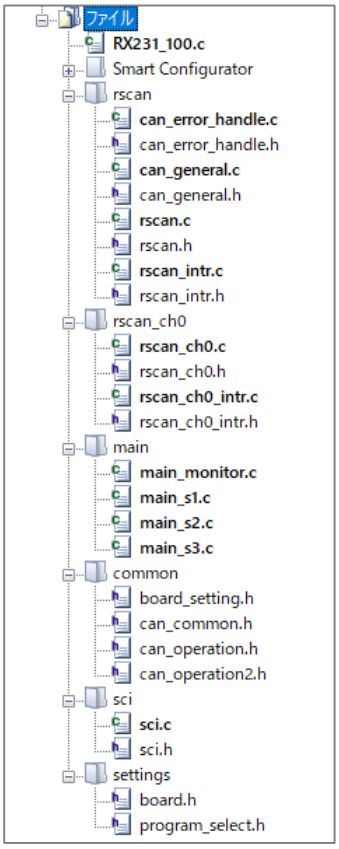
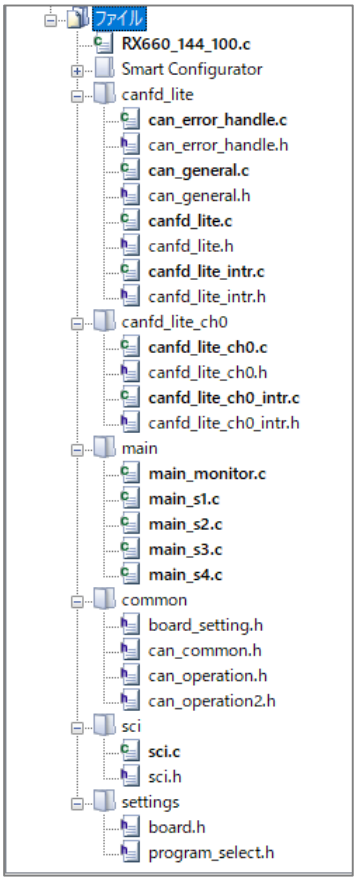
can, can_ch0, can_ch1, can_ch2, main のカテゴリには、CAN_module
rscan, rscan_ch0, main のカテゴリには RSCAN_module
canfd_lite, canfd_lite_ch0, main のカテゴリには CANFDLite_module
以下のファイルを追加。

common, sci のカテゴリには、ALL 以下のファイルを追加。

settings は、マイコン毎に異なる（マイコン種毎のフォルダの下にある settings を使用）。

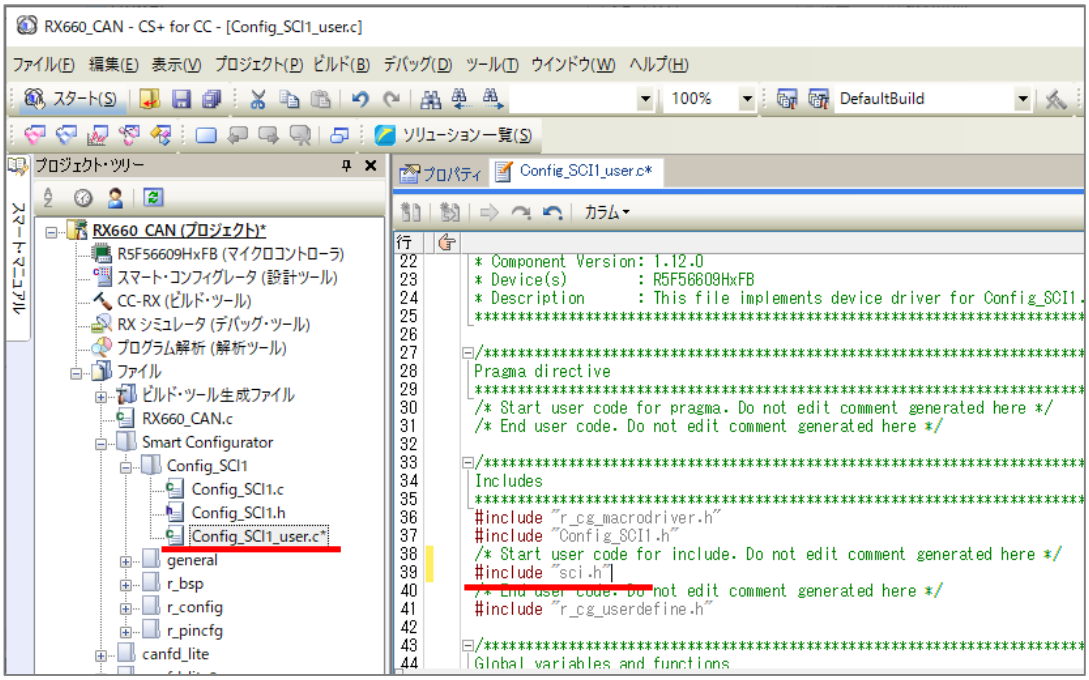
となります。フォルダ内の、.c, .h ファイルを全て、カテゴリ内に追加してください。

ソースコード追加後のプロジェクト・ツリーは、以下の様になります。

CAN モジュールの場合	RSCAN モジュールの場合	CANFD_LIte モジュールの場合
		

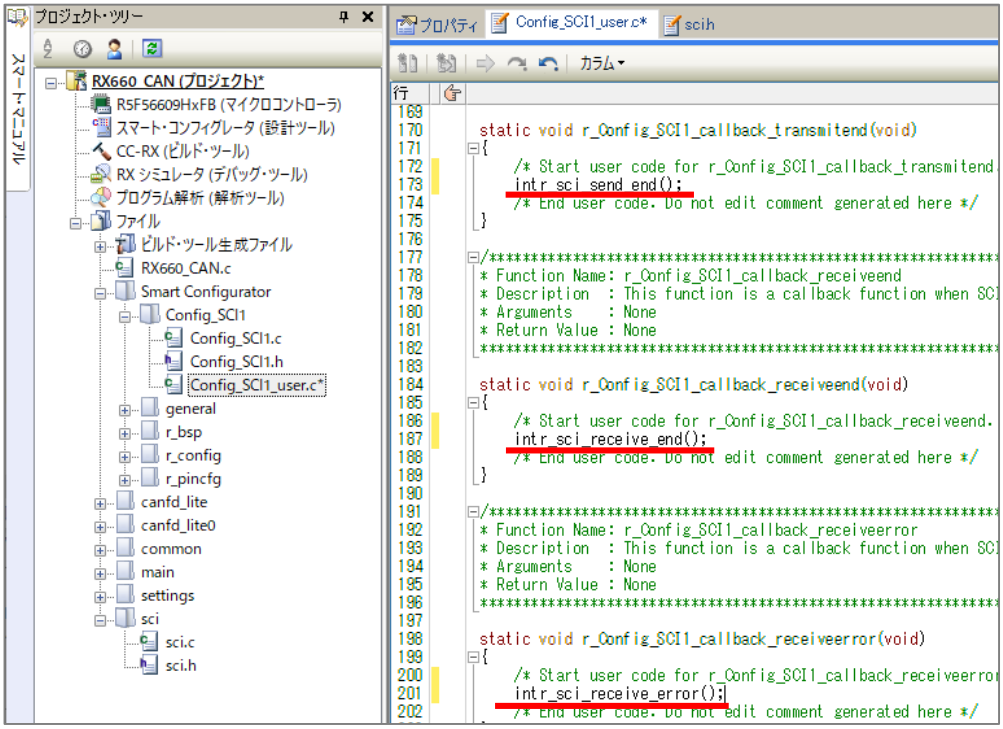
(6)スマート・コンフィグレータ生成コードにユーザ関数を追加する

次に、スマート・コンフィグレータ生成ファイルに記述を追加します。



変更するファイルは、Smart Configurator – Config_SCI1 – Config_SCI1_user.c で、

#include "sci.h" を追加します。



また、
static void r_Config_SCI1_callback_transmitend(void) 内に intr_sci_send_end();
static void r_Config_SCI1_callback_receiveend(void) 内に intr_sci_receive_end();
static void r_Config_SCI1_callback_receiveerror(void) 内に intr_sci_receive_error();
を追加してください。

sci.h 内に下記コメントがありますので、

```
/*
 RX, RL78, RH850の場合、ツールで生成した コード( Config_????_user.c /
r_cg_????_user.c )
 内に以下の記載を追加 してください（詳しくは readme.txt を参照）

・送信完了時に呼び出される( ~transmitend(), ~sendend() )内に記載

intr_sci_send_end();

・受信完了時に呼び出される( ~receiveend() )内に記載

intr_sci_receive_end();

・受信エラー時に呼び出される( ~receiveerror(), ~error() )内に記載

intr_sci_receive_error();

*/
```

コピー&ペーストで貼り付けてください。

なお、スマート・コンフィグレータ生成コードにユーザ側で記載を追加する場合は、

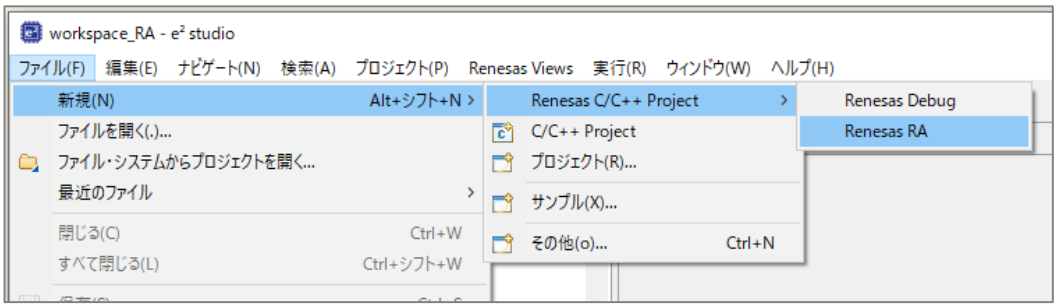
/* Start user code for r_Config_SCI1_callback_transmitend. Do not edit comment generated here */
//この部分に記載
/* End user code. Do not edit comment generated here */

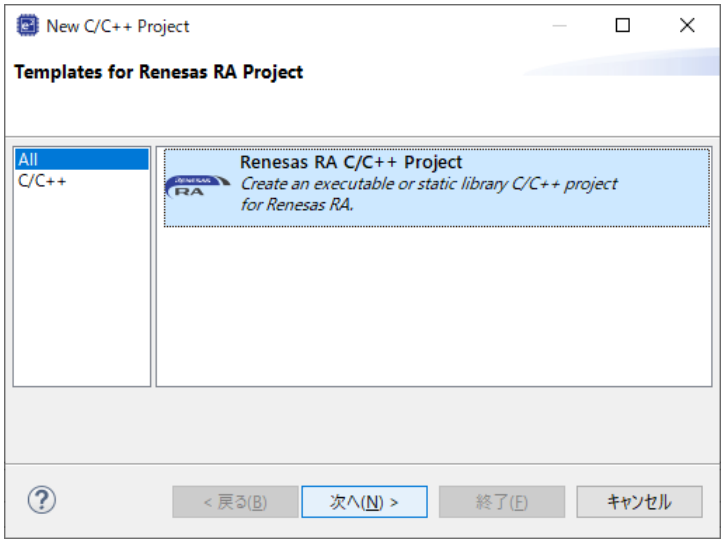
Start... と End.... に挟まれる位置に記載を追加してください。その他の位置に記載した場合、コード生成ボタンを押した際に、消されてしまいます。

以上で、プロジェクトの新規作成は完了ですので、3.1 章の手順でプログラムのビルドを行ってください。

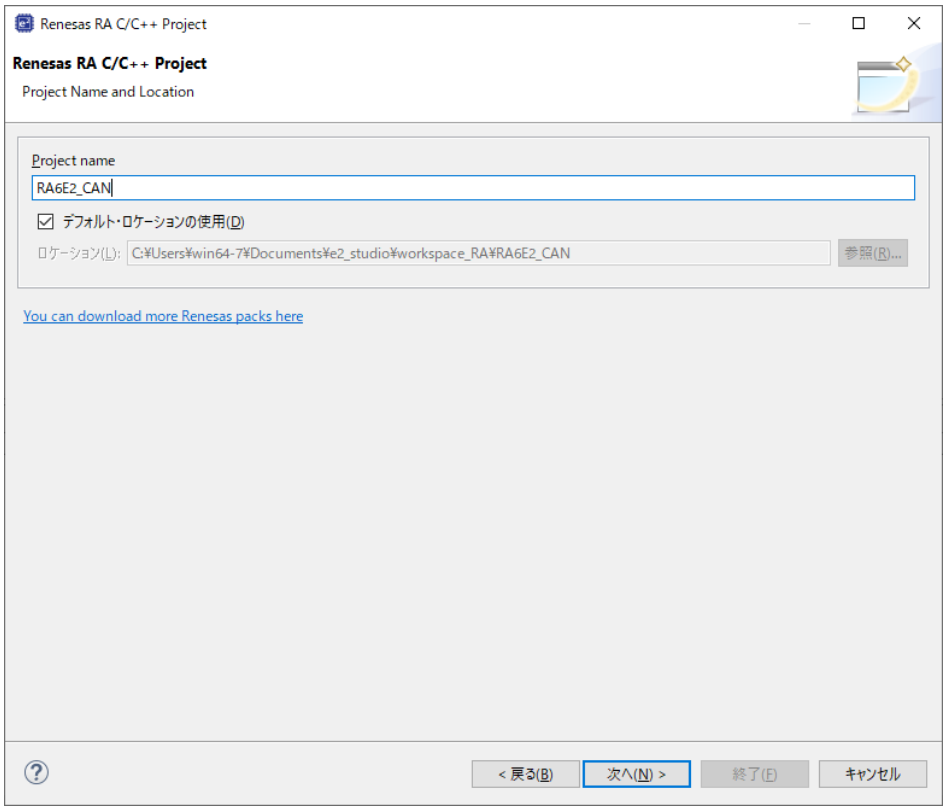
4.2. RA(e2studio)

(1)マイコンボード（搭載マイコン種）に応じたプロジェクトを作成

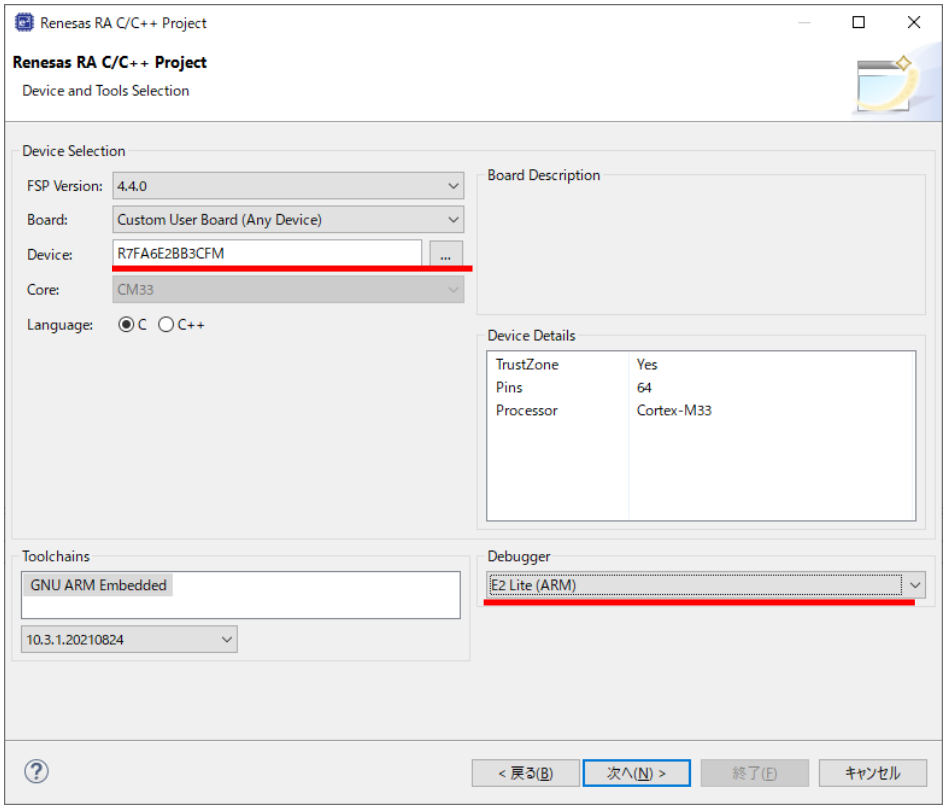




「次へ」

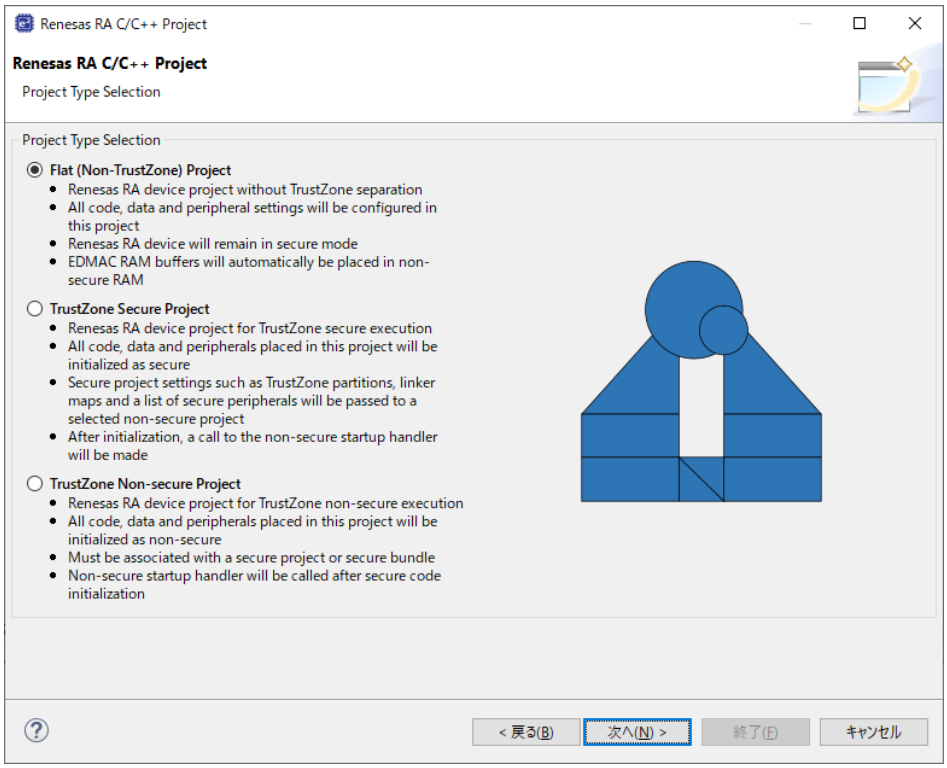


任意の名称を指定して、「次へ」。

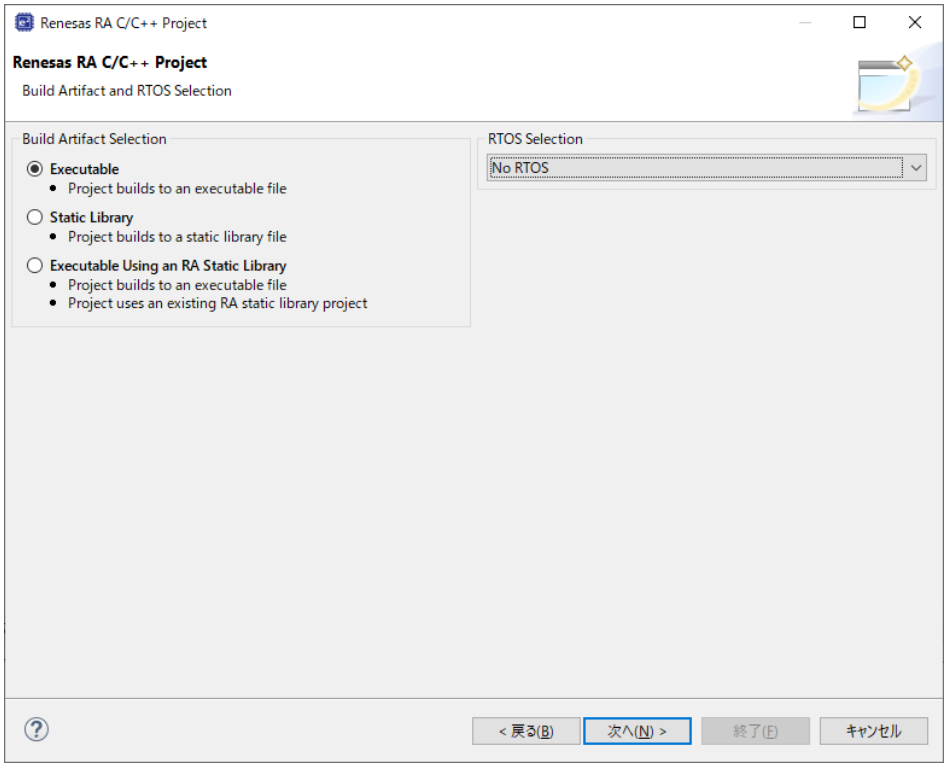


Device 欄は、プロジェクトを作成するターゲットマイコン型名を選択、Debugger 欄は任意ですが、使用するデバッガを選択。

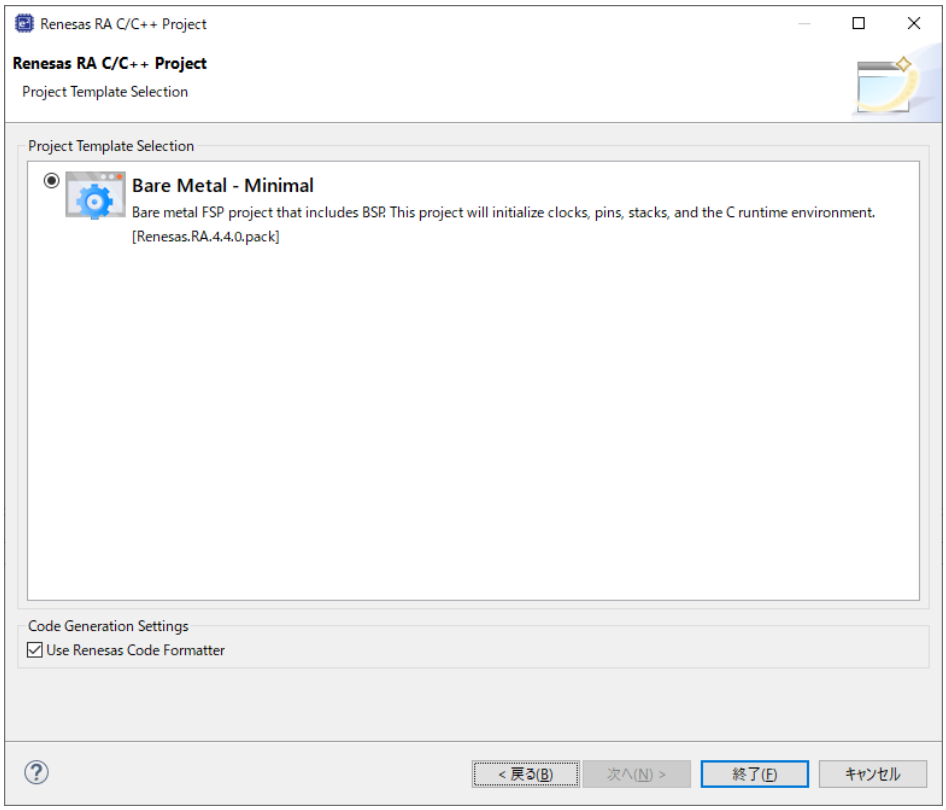
－TrustZone 対応マイコンのみ－



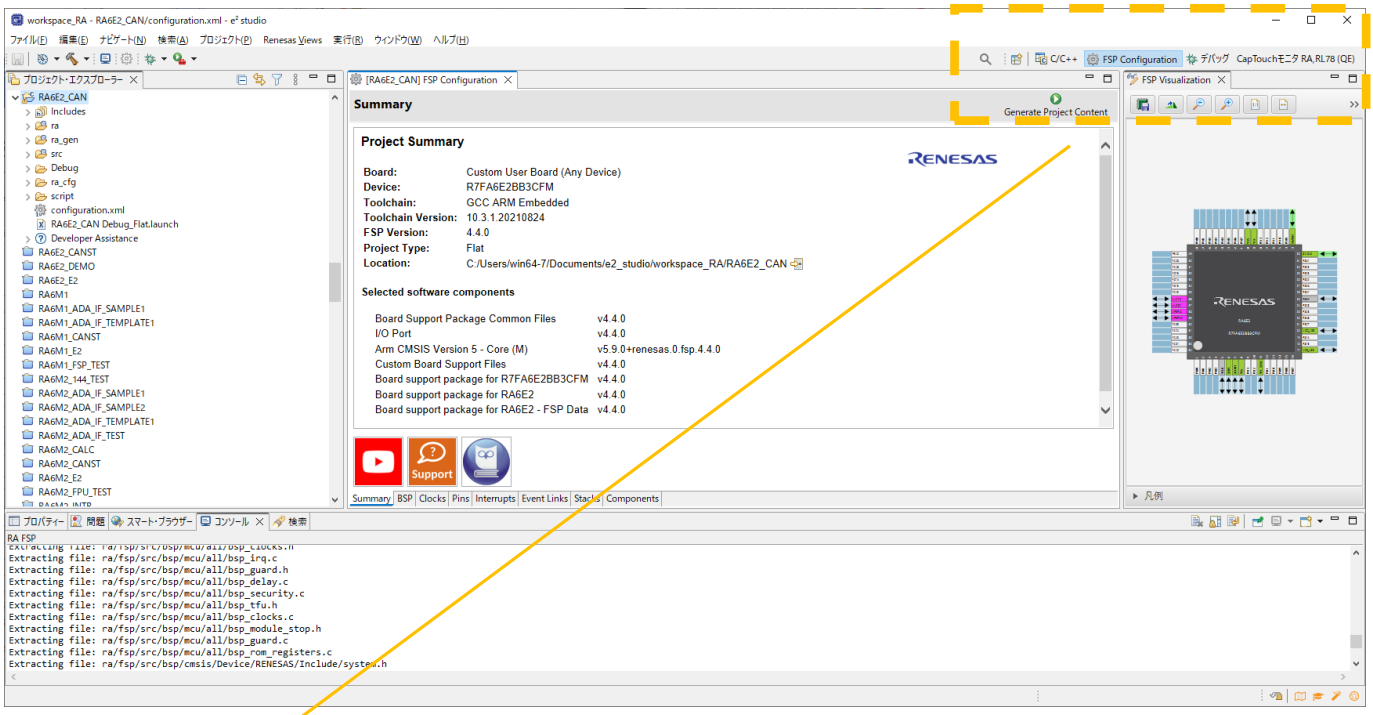
TrustZone の設定があります。本サンプルでは、TrustZone の機能は使っておらず、Flat Project で動作確認をしています。(Flat を選択して)「次へ」。

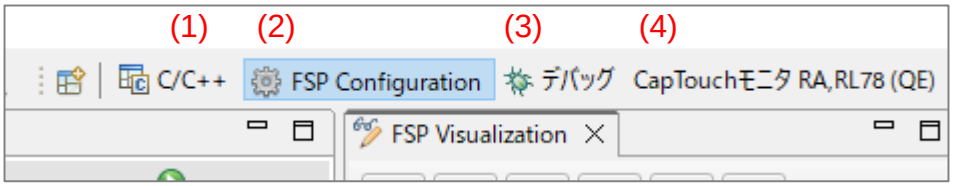


「Executable」を選択、RTOS はサンプルプロジェクトでは使用していません(No RTOS)、「次へ」



Bare Metal - Minimal を選択。「終了」。

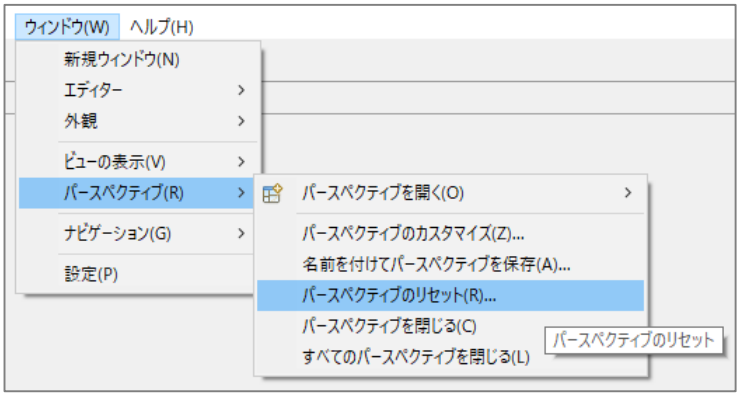




e2studio では、右上に、どのモード（パースペクティブ）を開くかの設定ボタンがあります。

(1)C/C++　ソースコードを編集するモード
(2)FSP Configuration　FSP の設定をするモード
(3)デバッグ　デバッグを行うモード
(4)CapTouch モニタ（ミドルウェア）[オプション]

例えば、FSP の設定を行う場合は、必ず(2)が選択されている必要があります。このモードが、合っていないと必要な設定画面が表示されませんので、ご注意ください。



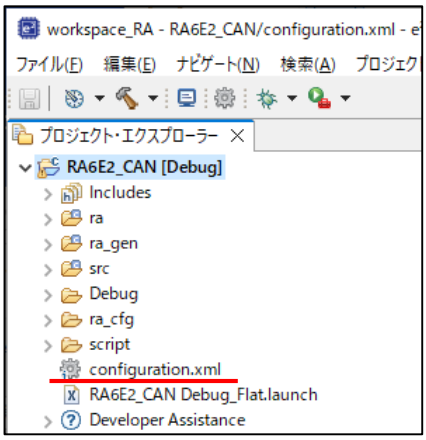
表示画面に必要な設定項目等が表示されない場合は、

パースペクティブのリセット

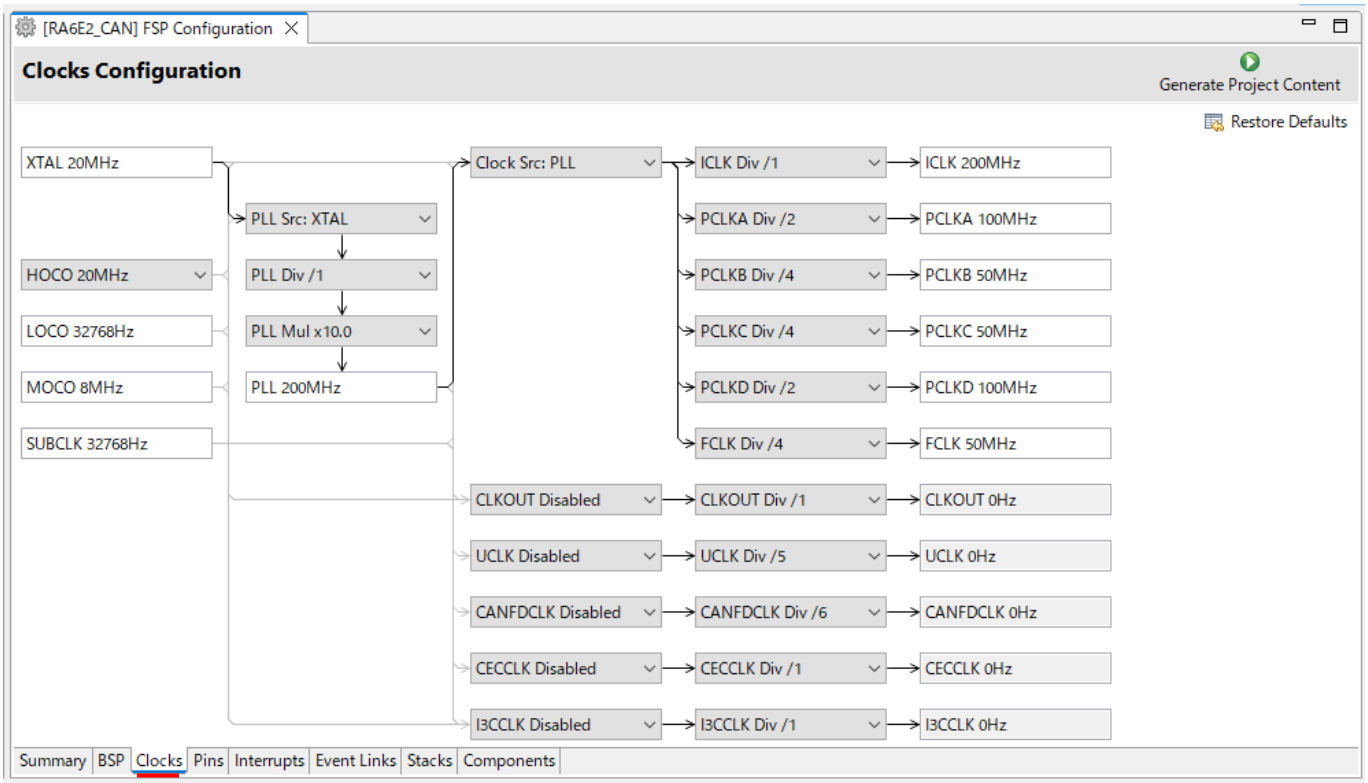
を試してみてください。表示画面が初期化されます

(2)クロックの設定

FSP のクロックの設定を行います。前頁のモードは「FSP Configuration」が選択されている状態としてください。

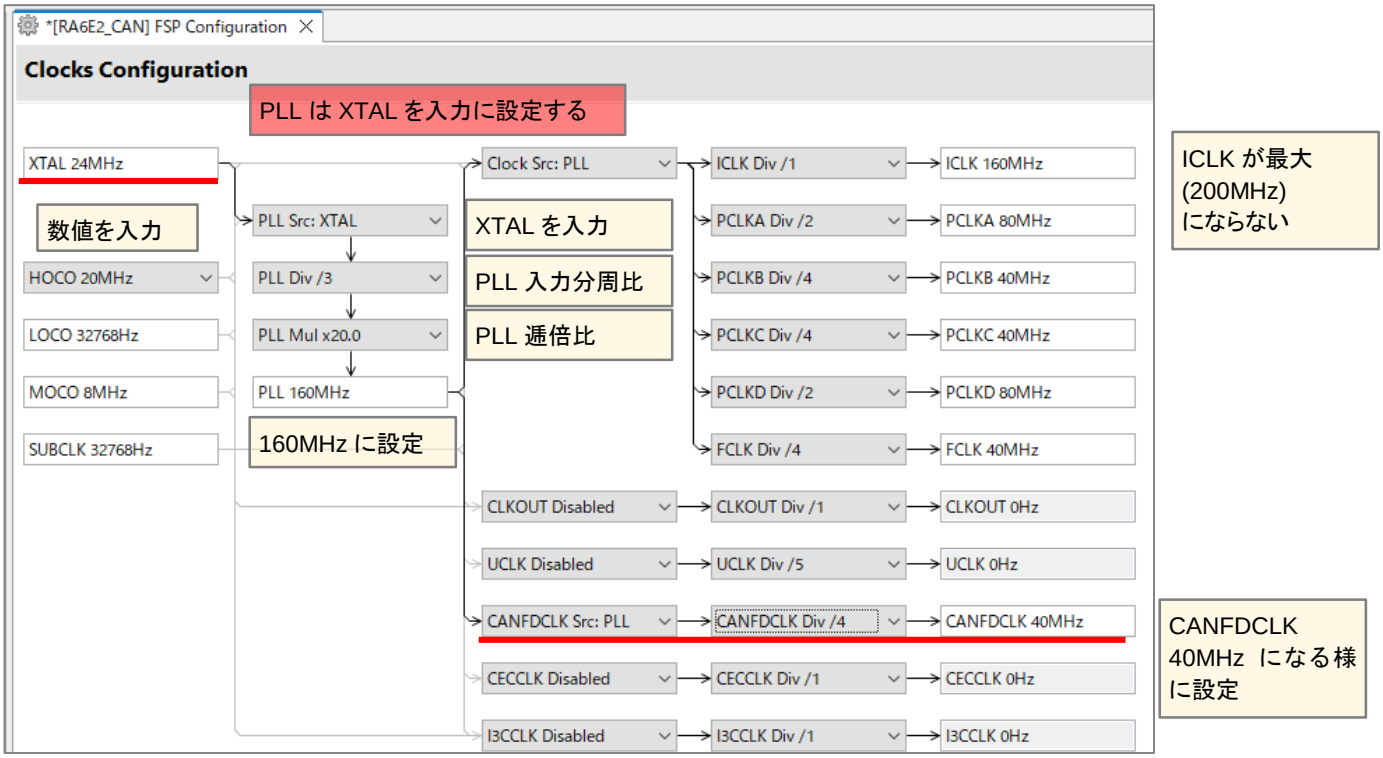


FSP の設定は、configuration.xml の部分を変更します。（このファイルが開いていない場合は、ダブルクリックで開いてください。プロジェクト作成直後は、ファイルが開いた状態になっています。）



Configuration.xml の Clocks タブを開く

・RA6T3/RA6E2 の設定例（サンプルプロジェクトでの設定）



上記は設定例です。（サンプルプロジェクトの、RA6E2 では上記の設定としています。）

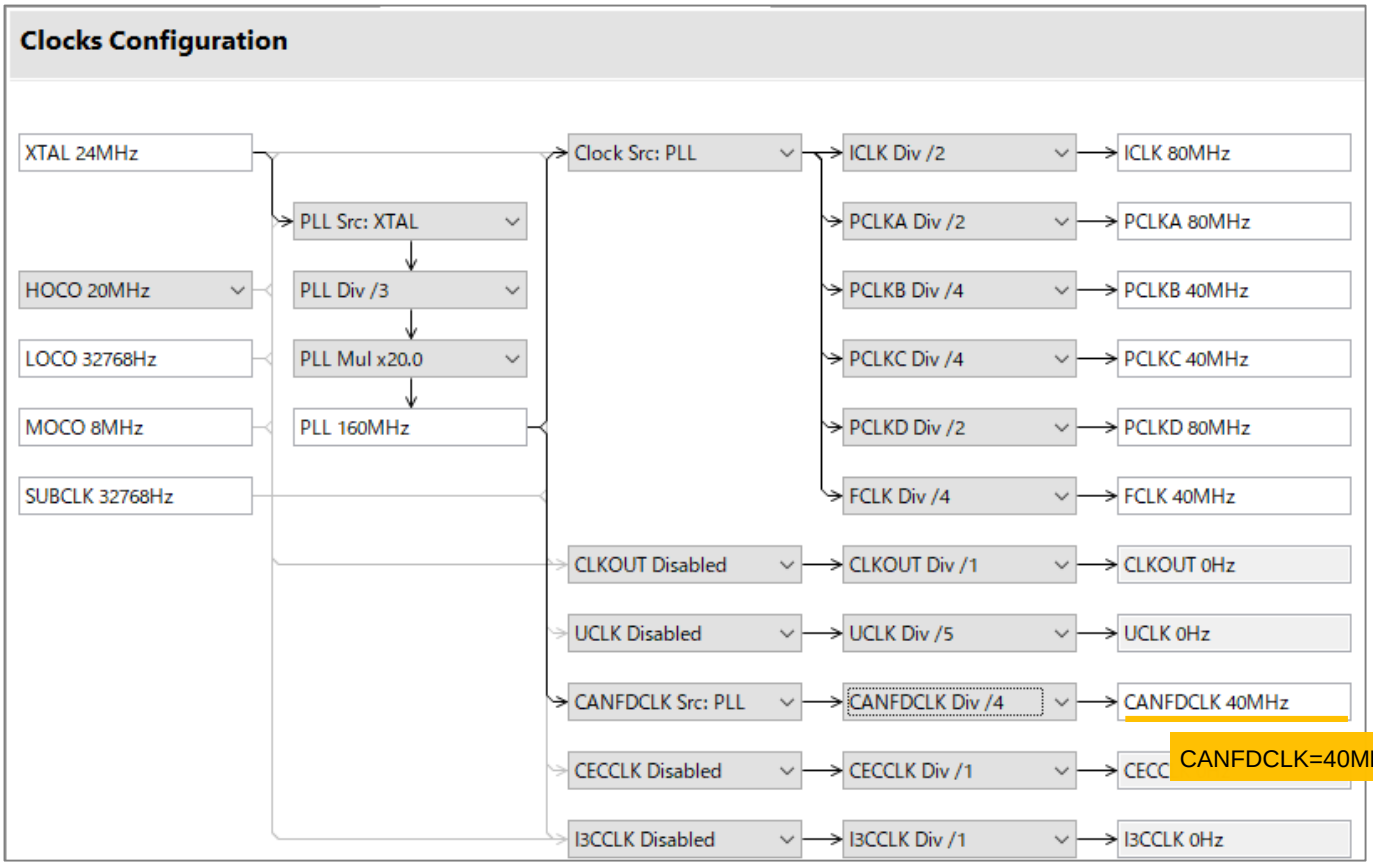
マイコン種	搭載水晶振動子 XTAL	CAN クロックソース（周波数）
RA8E1 RA8M1 RA8T1	24MHz	CANFDCLK(80MHz)
RA6T1	24MHz	PCLKB(60MHz)
RA6T2	24MHz	CANFDCLK(40MHz)
RA6M5	24MHz	CANFDCLK(40MHz)
RA6M3 RA6M2 RA6M1	24MHz	PCLKB(60MHz)
RA6E1 RA6M4 RA4E1 RA4M3 RA4M2	24MHz	PCLKB(50MHz)
RA6E2 RA6T3 RA4E2 RA4T1	24MHz	CANFDCLK(40MHz)
RA4M1	8MHz	PCLKB(24MHz)
RA2A1	20MHz	MainOSC(20MHz)
RA2L1	16MHz	MainOSC(16MHz)

XTAL の値は、ボード毎に上記値。本サンプルプロジェクトをそのまま使用する場合は、CAN のクロックソースの周波数を上記に設定してください。

※RA6E2 等は、CANFDCLK を 40MHz に設定すると、マイコン ICLK を最大周波数で動かす事が出来ないので、上記設定が最適解という訳ではありません。(クロックソースの周波数に応じてプログラムを一部変更すれば)どのような周波数に設定するかは、自由です。RA6E2 等 CANFDCLK を持つマイコンの場合、CAN を使用するためには CANFDCLK を有効化する必要があります。

CAN のクロックは精度が要求されますので、XTAL(マイコン内蔵のオシレータに対し精度が良い)をそのまま使用するか(RA2 系)、XTAL を PLL で逓倍したクロックを使用する事を推奨します。(HOCO ベースでの使用は非推奨です。)

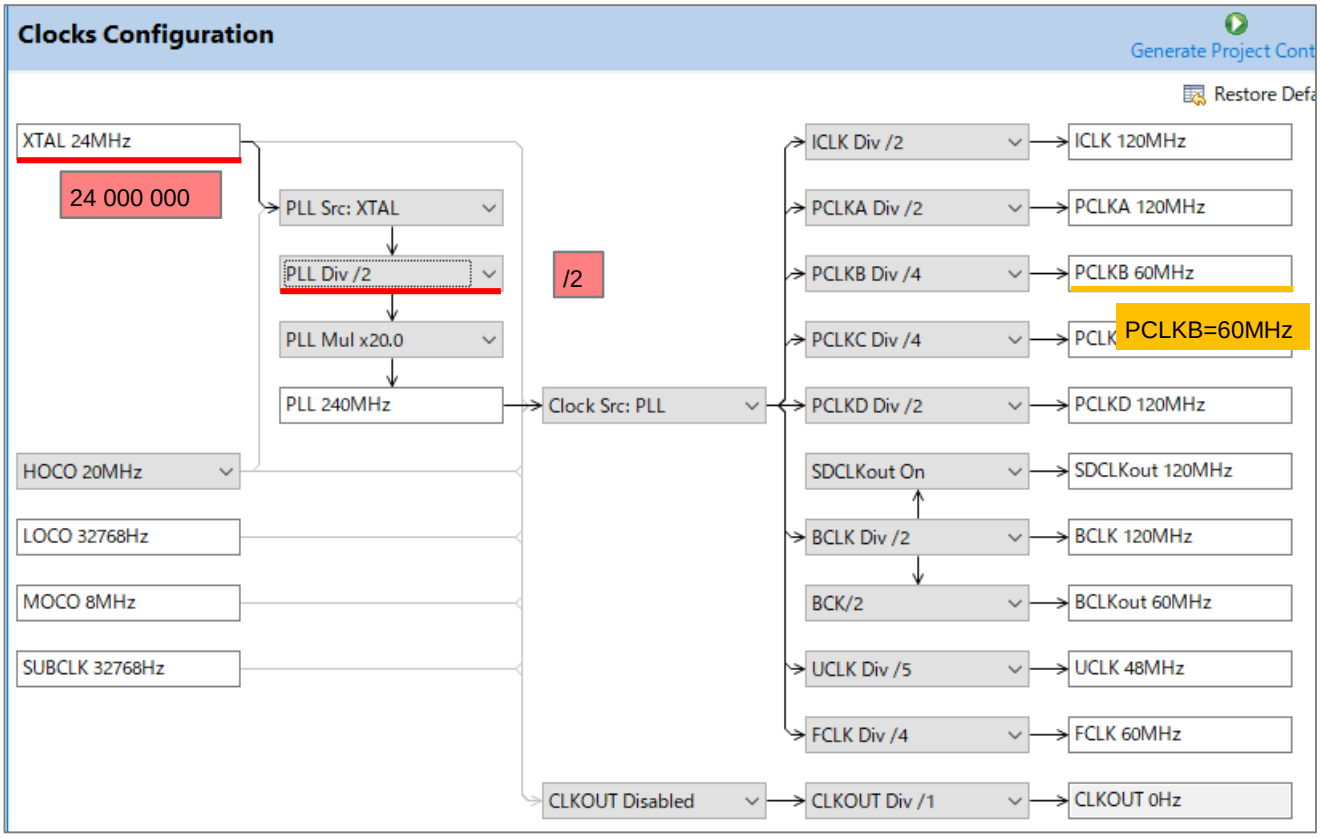
・RA4E2/RA4T1 の設定例(サンプルプロジェクトでの設定)



RA6E2 同様、CANFDCLK を最大(40MHz)設定とすると、ICLK が最大値とならない(ICLK=80MHz, 最大 100MHz)事となります。

※CAN より、CPU コアの動作周波数を優先する場合は、PLL の出力を 200MHz として、ICLK=100MHz, CANFDCLK=25MHz と設定する方法もあります。CAN の速度設定に関しては、CAN スタータキット RX/RA 取扱説明書内に、CAN クロックと 1Tq 時間の関係の記載がありますので、参照してください。
(CANFDCLK=25MHz に設定した場合、CANFD の通信速度として 4Mbps や 2Mbps には設定不可となります。)

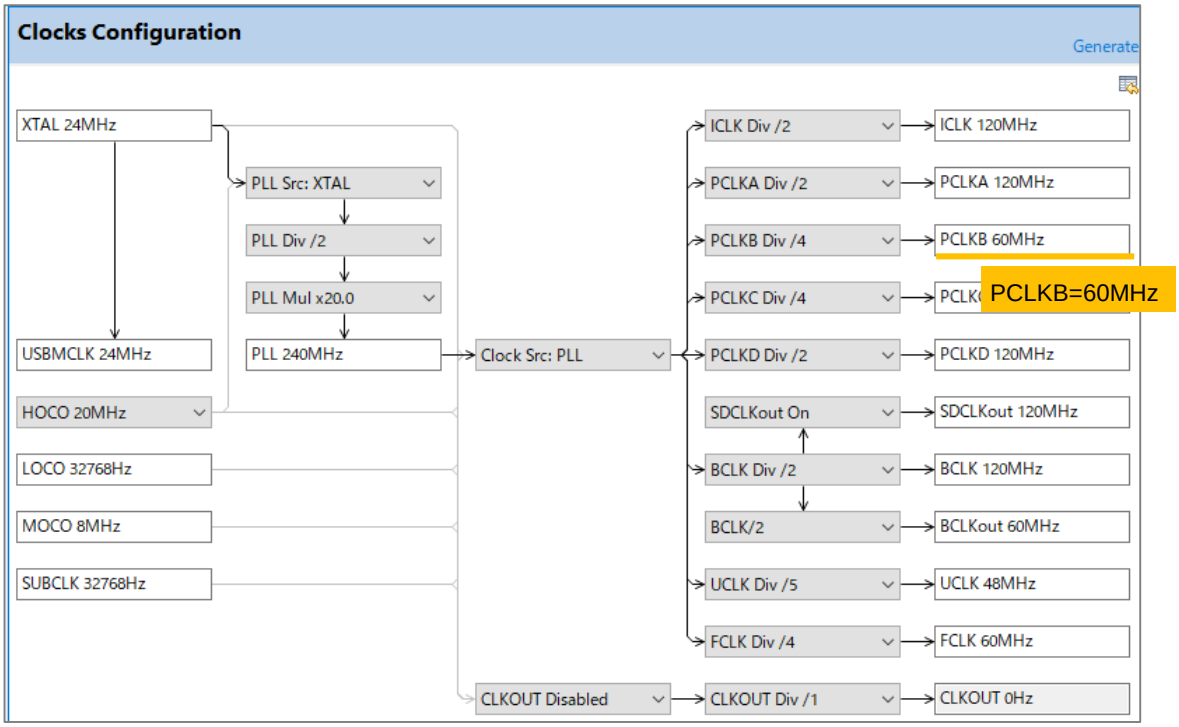
・RA6M1/RA6M2 の場合



※「PLL DIV /1」「PLL Mul x10.0」の組み合わせでも問題ありません

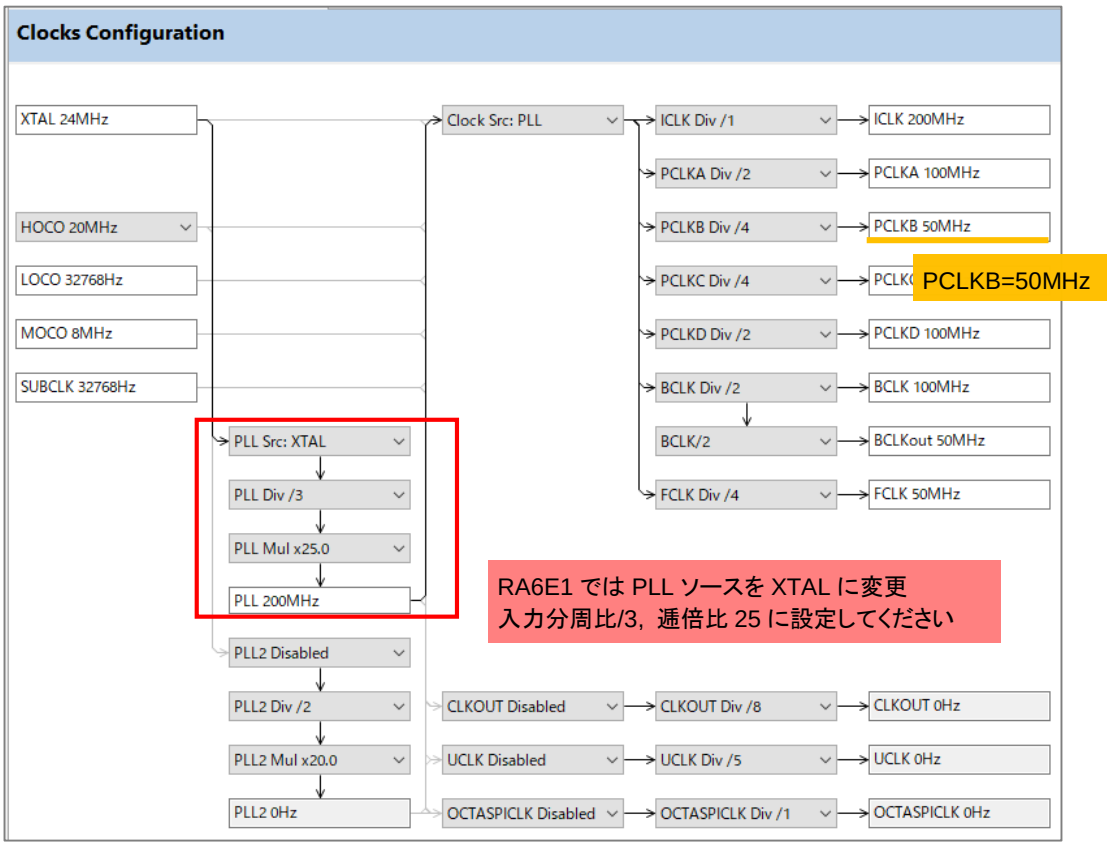
※画面は RA6M2 のもので他のマイコンでは一部表示が異なりますが、設定項目は変わりません

・RA6M3 の場合



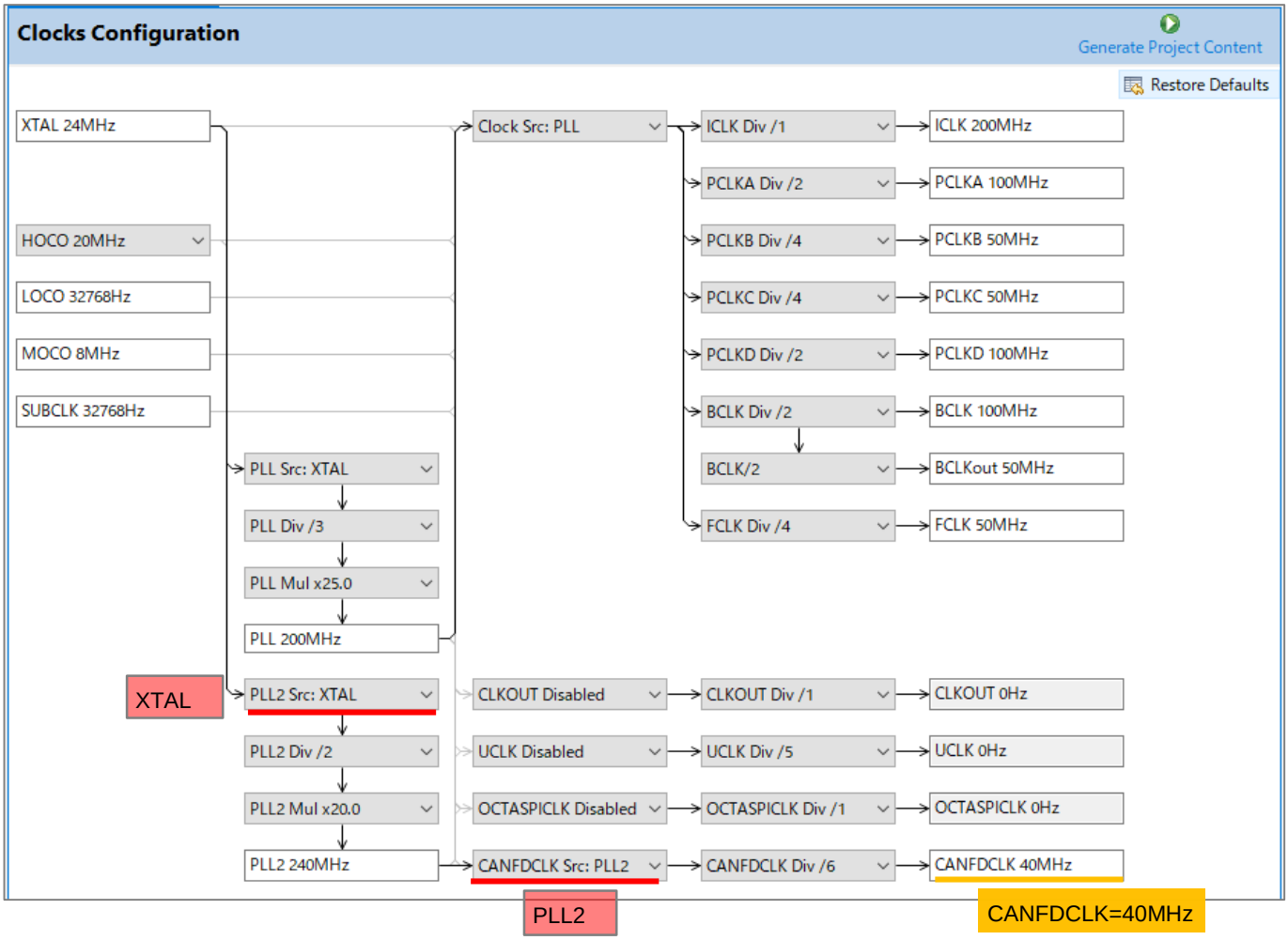
設定変更が必要な部分はありません。（デフォルト値のままで問題ありません）

・RA6M4/RA6E1 の場合



RA6M4 では設定変更が必要な部分はありません。（デフォルト値のままで問題ありません）

・RA6M5 の場合

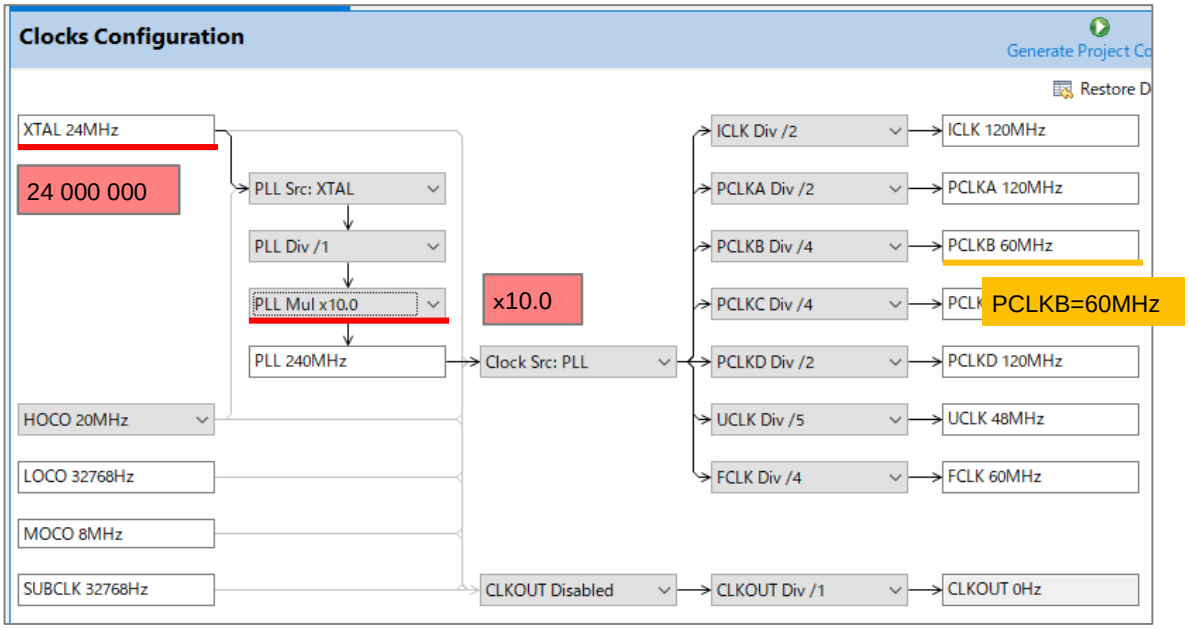


RA6M5 の場合、CANFDCLK が 40MHz になる様に設定してください。

（デフォルトでは、Disabled になっています）

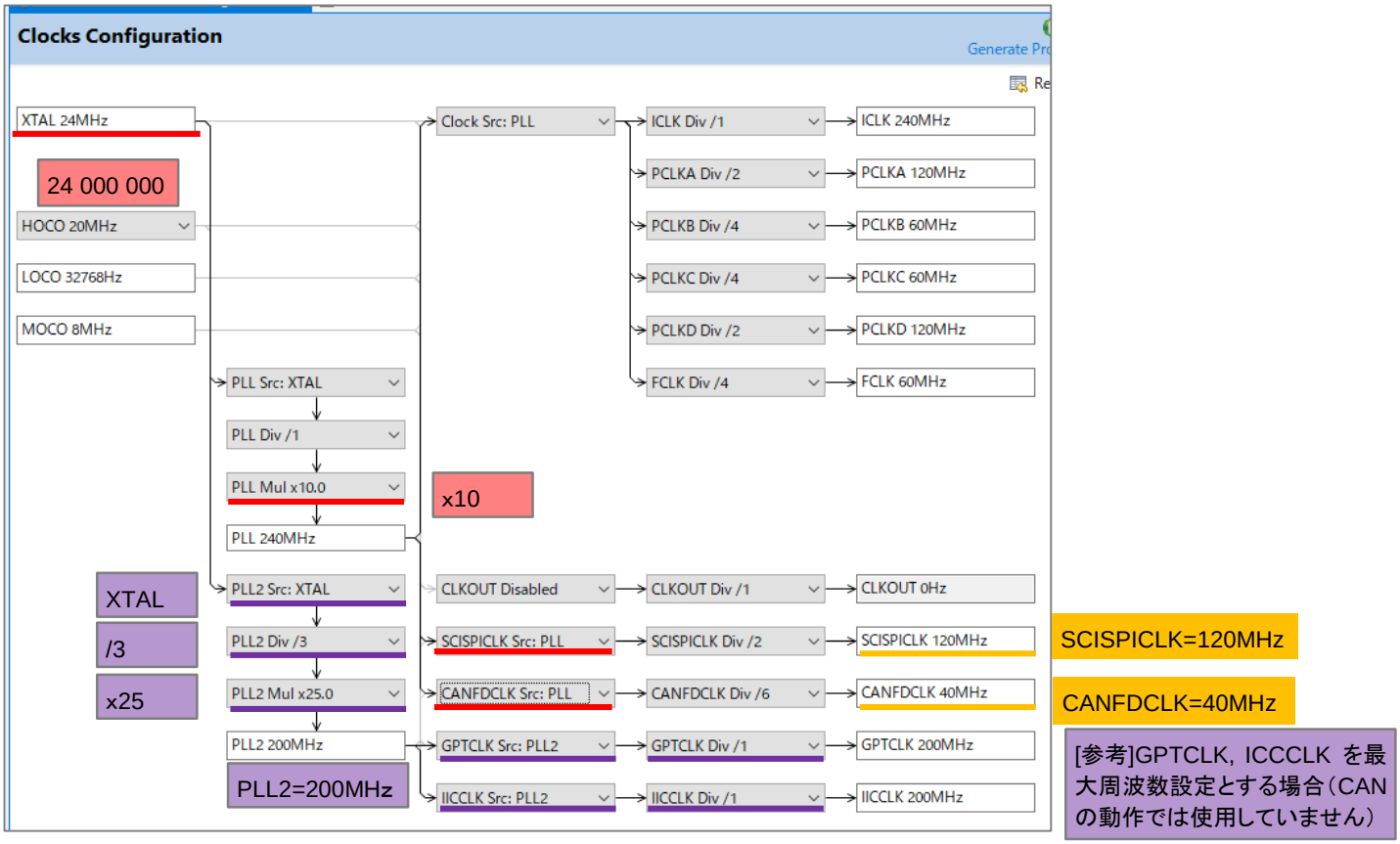
（PLL2 を 240MHz の設定とし、240MHz/6 で 40MHz 設定となります）

・RA6T1 の場合



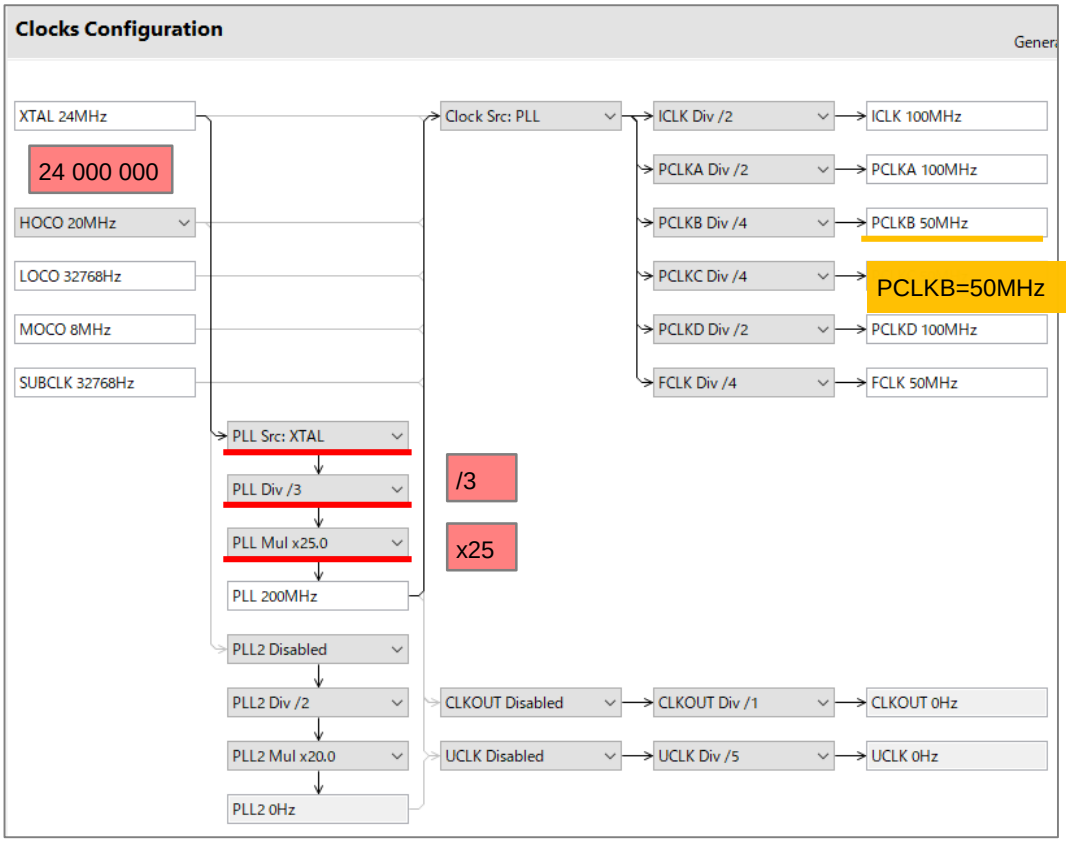
XTAL を 24MHz に、PLL の逓倍率を 10 に設定してください。

・RA6T2 の場合



XTAL を 24MHz に、PLL の逓倍率を 10 に設定してください。CANFDCLK を Disabled から PLL/6 設定で 40MHz に設定してください。SCISPICLK も同様に Disabled から PLL/2 設定で 120MHz に設定してください（SCISPICLK は UART(SCI)を使った画面出力に使用しています）。

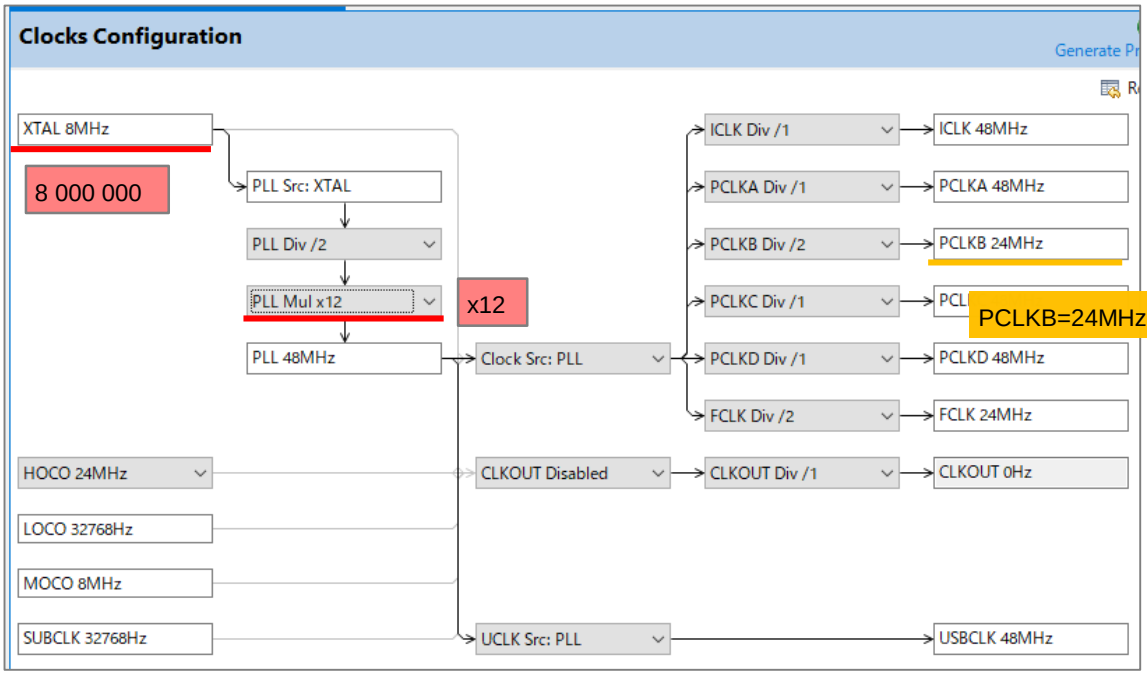
・RA4E1/RA4M3/RA4M2 の場合



PLL Src : XTAL に変更、PLL の分周比・逓倍率を/3, x25 選択。

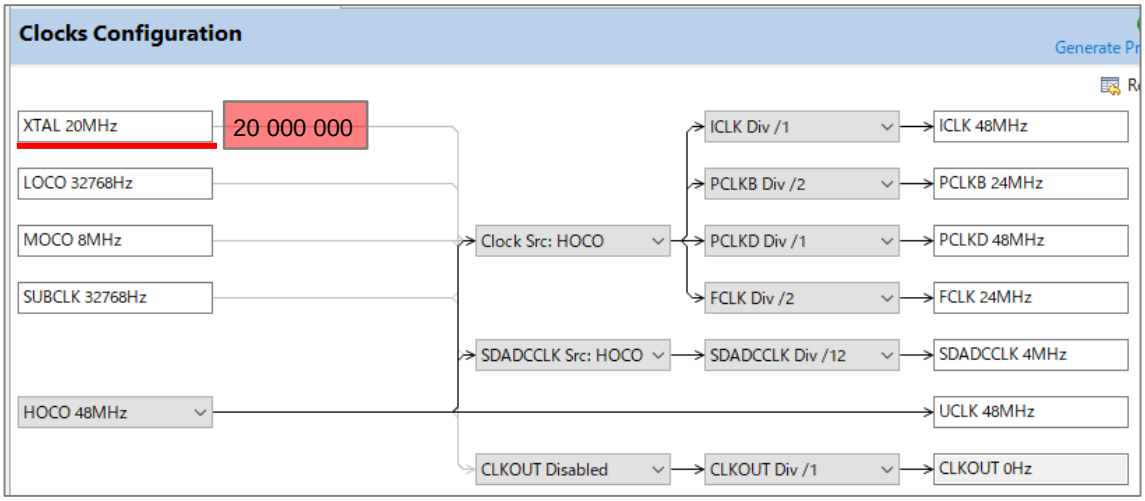
※画面のハードコピーは RA4E1 のものですが、他のマイコンでも同様です

・RA4M1 の場合



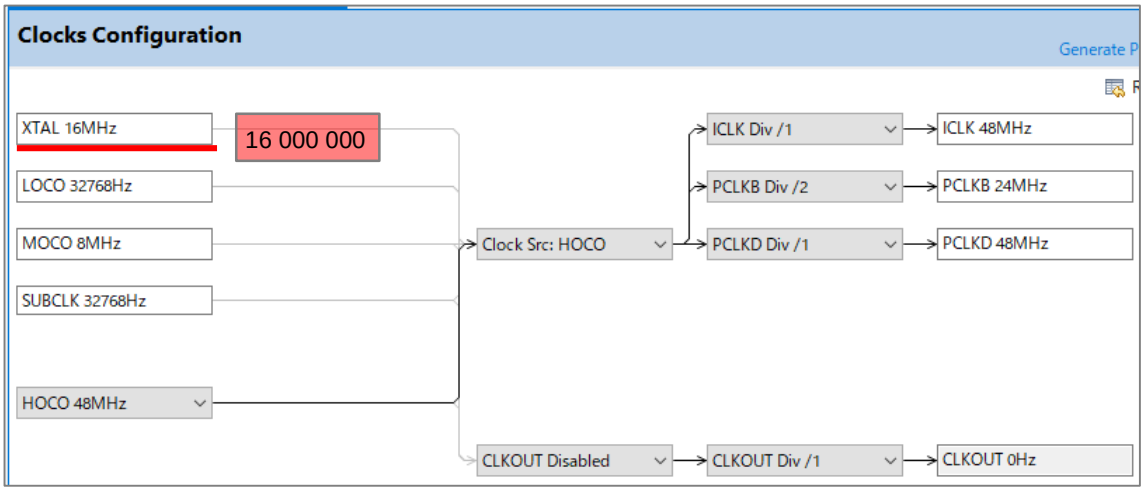
XTAL を 8MHz に、PLL の逓倍率を 12 に設定してください。

・RA2A1 の場合



XTAL を 20MHz に設定してください。(CAN のクロックとしては、XTAL が使用されます)

・RA2L1 の場合

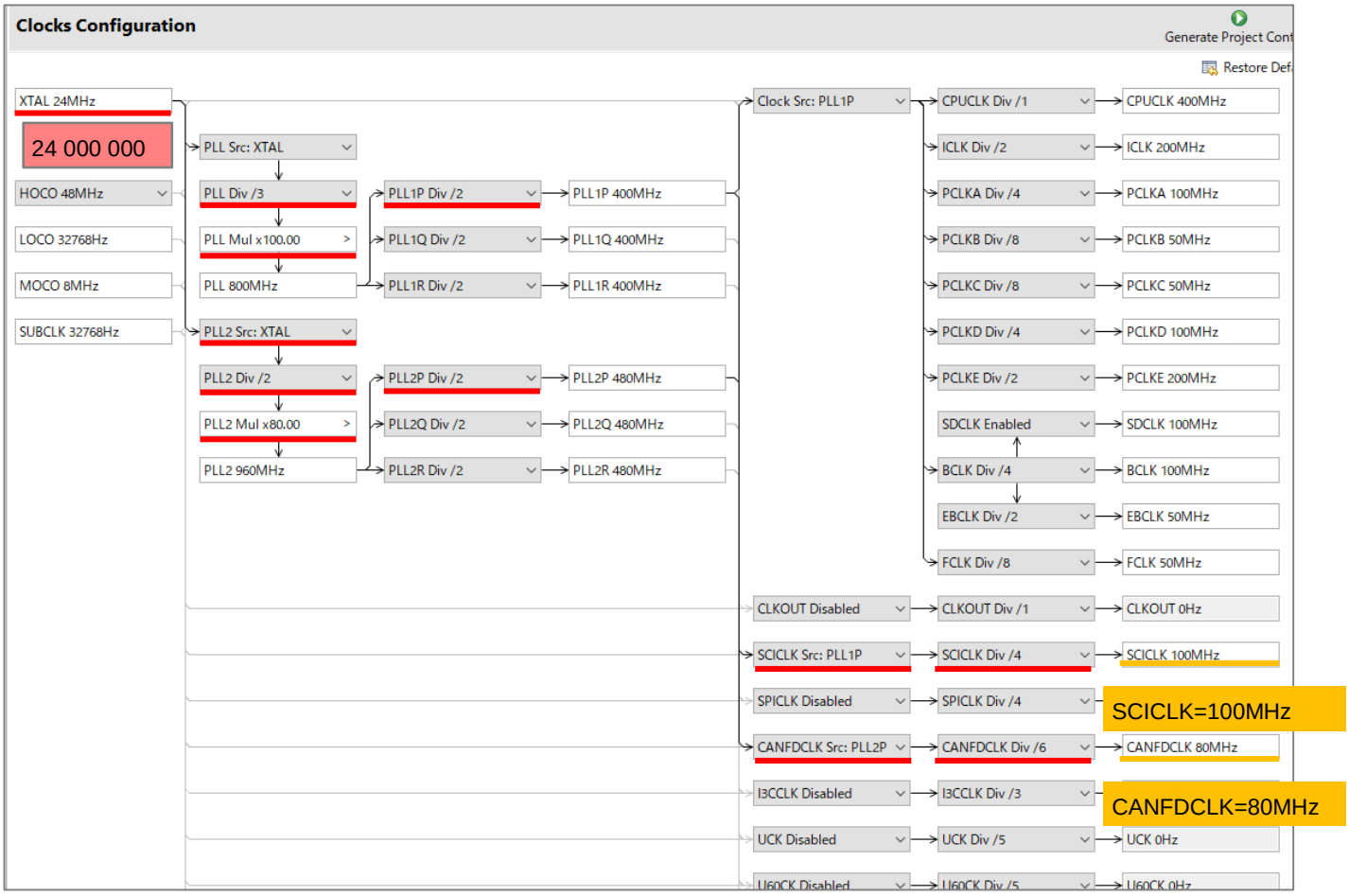


XTAL を 16MHz に設定してください。(CAN のクロックとしては、XTAL が使用されます)

RA2A1(HSBRA2A1F64), RA2L1(HSBRA2L1F100, HSBRA2L1F64)は、PLL を搭載していません。

CPU クロック(ICLK)としては、HOCO(48MHz)で動作し、CAN のクロックは XTAL となります。

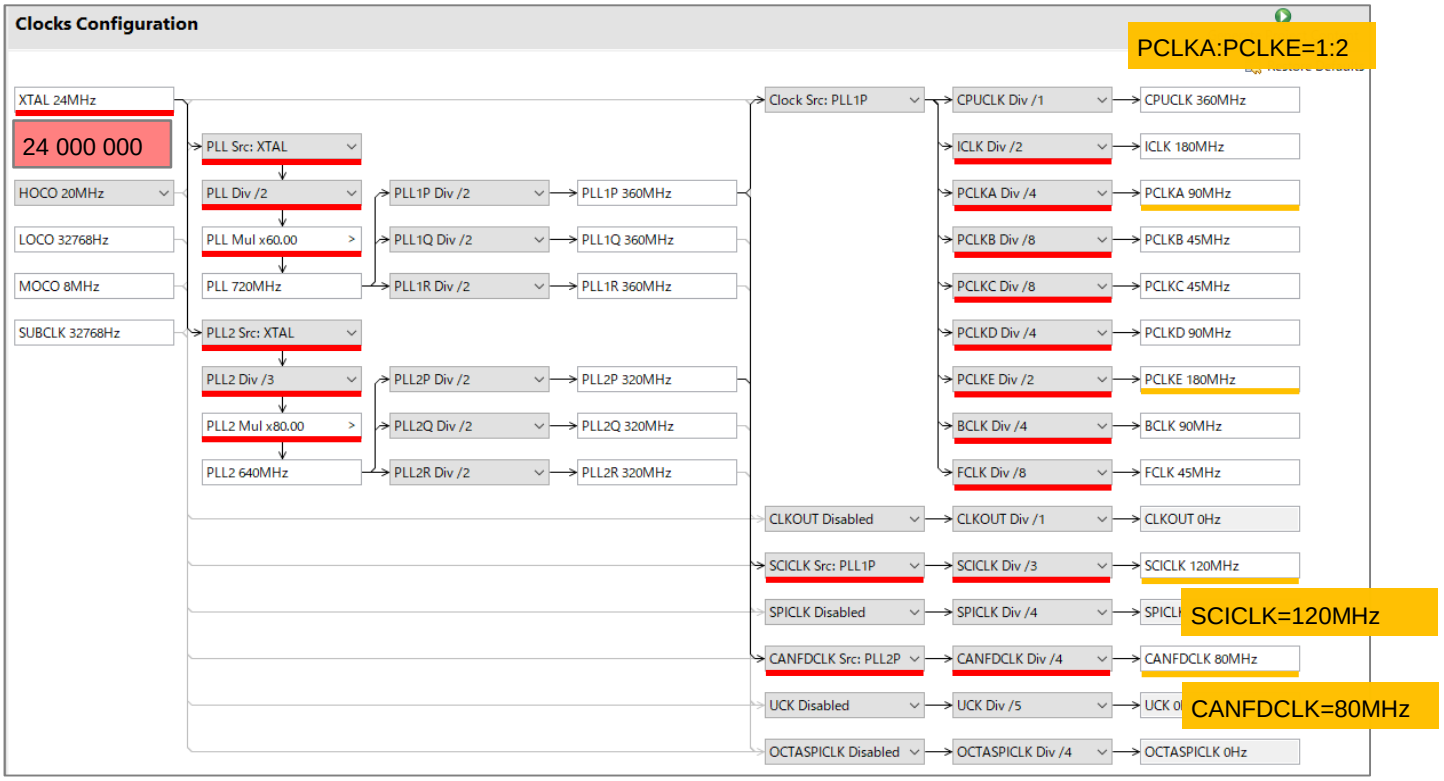
・RA8M1/RA8T1 の場合



XTAL は 24MHz、PLL の分周比・逓倍率を調整し、CANFDCLK=80MHz となる様に設定してください。

また、サンプルプログラムでは SCI の機能を使っているので、（SCICLK Disable ->??MHz 周波数はどの値で問題ありません）SCICLK を有効化してください。

・RA8E1 の場合



XTAL は 24MHz、PLL の分周比・逓倍率を調整し、CANFDCLK=80MHz となる様に設定してください。

また、サンプルプログラムでは SCI の機能を使っているので、（SCICLK Disable ->??MHz 周波数はどの値で問題ありません）SCICLK を有効化してください。CAN-FD 使用時は、PCLKA:PCLKE=1:2 である必要があるので、PCLKA の分周比を大きくして PCLKA:PCLKE=1:2 の周波数となる様に設定してください。

※上記設定は、CPUCLK=360MHz（最大動作周波数）かつ、CANFDCLK=80MHz に設定する設定例です
ICLK, PCLKB などは、最大動作周波数より遅い設定です
（CANFDCLK を落として PCLKA や PCLKB の周波数を上げる等、設定は自由です。）

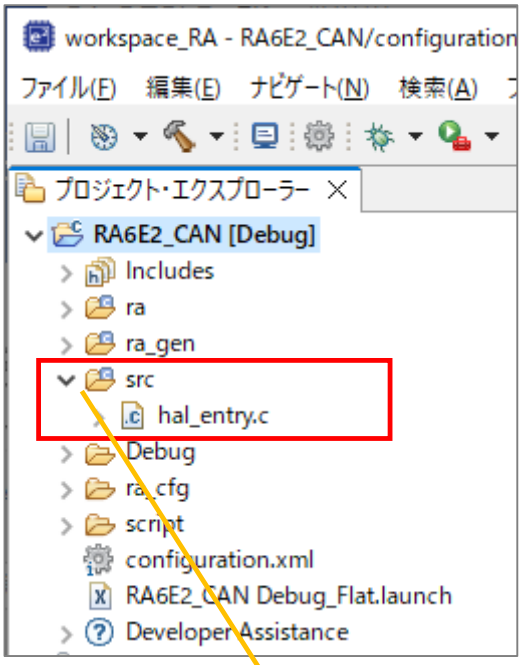
※PCLKB=60MHz(RA6M3, RA6M2, RA6M1), PCLKB=50MHz(RA6E1, RA6M4, RA4M3, RA4M2, RA4E1), PCLKB=24MHz(RA4M1), XTAL=20MHz(RA2A1), XTAL=16MHz(RA2L1), CANFDCLK=40MHz(RA6M5, RA6T3, RA6T2, RA6E2, RA4T1, RA4E2), CANFDCLK=80MHz(RA8E1, RA8M1, RA8T1)以外の値に設定した際は、src/settings/board.h 内の、CAN クロックの定義を変える必要があります

```
//CLK[MHz]
#define ICLK                180
#define PCLKB               45
#define XTAL                24
#define CAN_CLOCK           80          //CANFDCLK(=80MHz)をCANクロックとして使用
#define CANFD_CK_DIV_BASE   1           //CANFD:CANCLOCK/1 を CAN クロックに設定
```

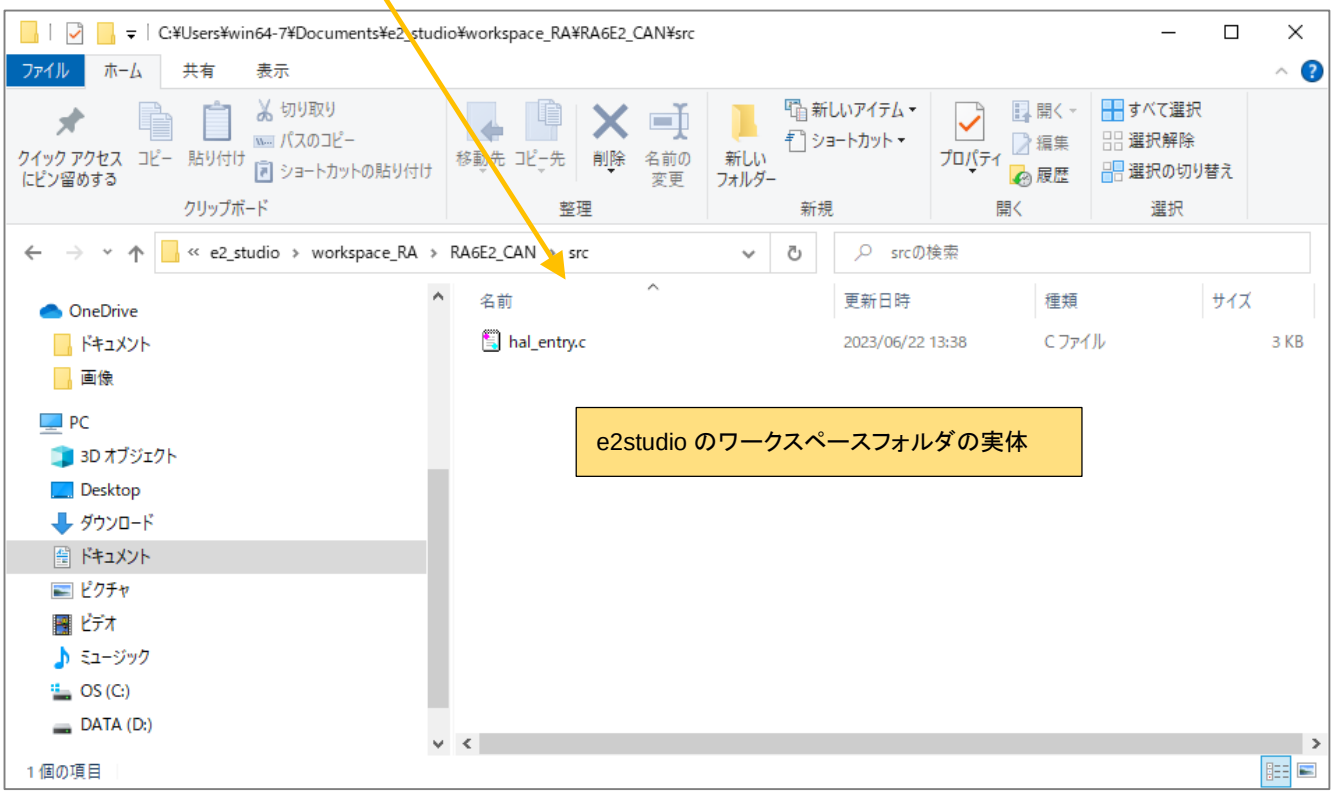
GUI で設定後、Create Project Content ボタンを押すと、クロック設定のソースコードが自動生成されますが、この後割り込み設定を行い、最後にコード生成を実行させますので、ここではコード生成はせず先に進んで良いです。

(3)ソースをプロジェクトに追加（コピー）

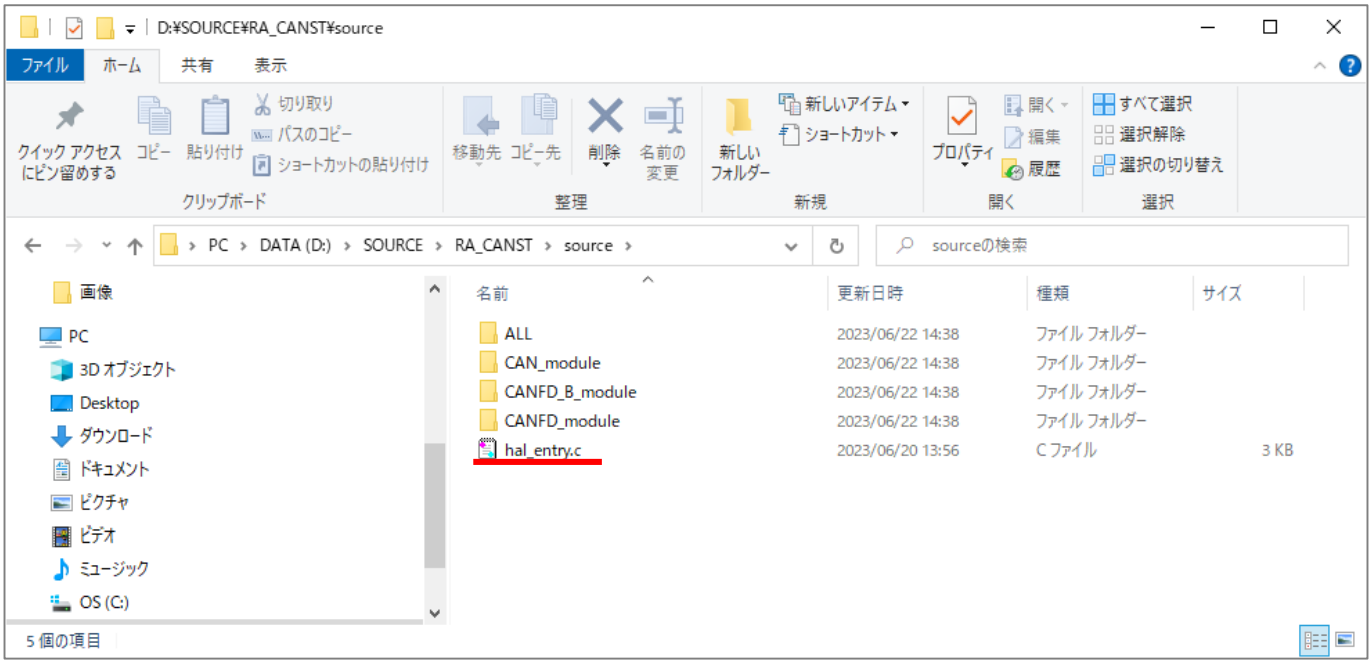
プログラムは、ソースファイルの形で CD 内に格納されています。



ソースは、プロジェクト・エクスプローラの src フォルダ以下に追加します。
ソースの追加は、e2studio 上からも行えますが、Wndows のエクスプローラ上でコピーするという手も使えます。



(a)hal_entry.c の上書きコピー

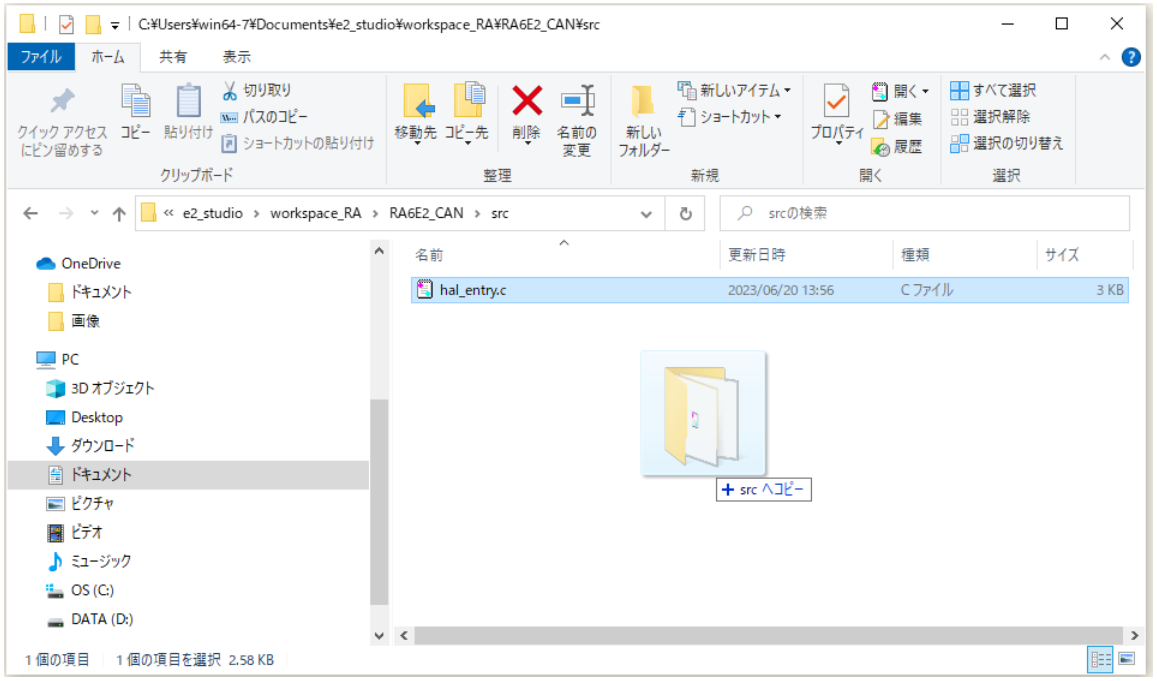


CD 内の、SOURCE¥RA_CANST¥source¥hal_entry.c を先程の src フォルダに上書き。

(b)マイコンタイプ毎の設定

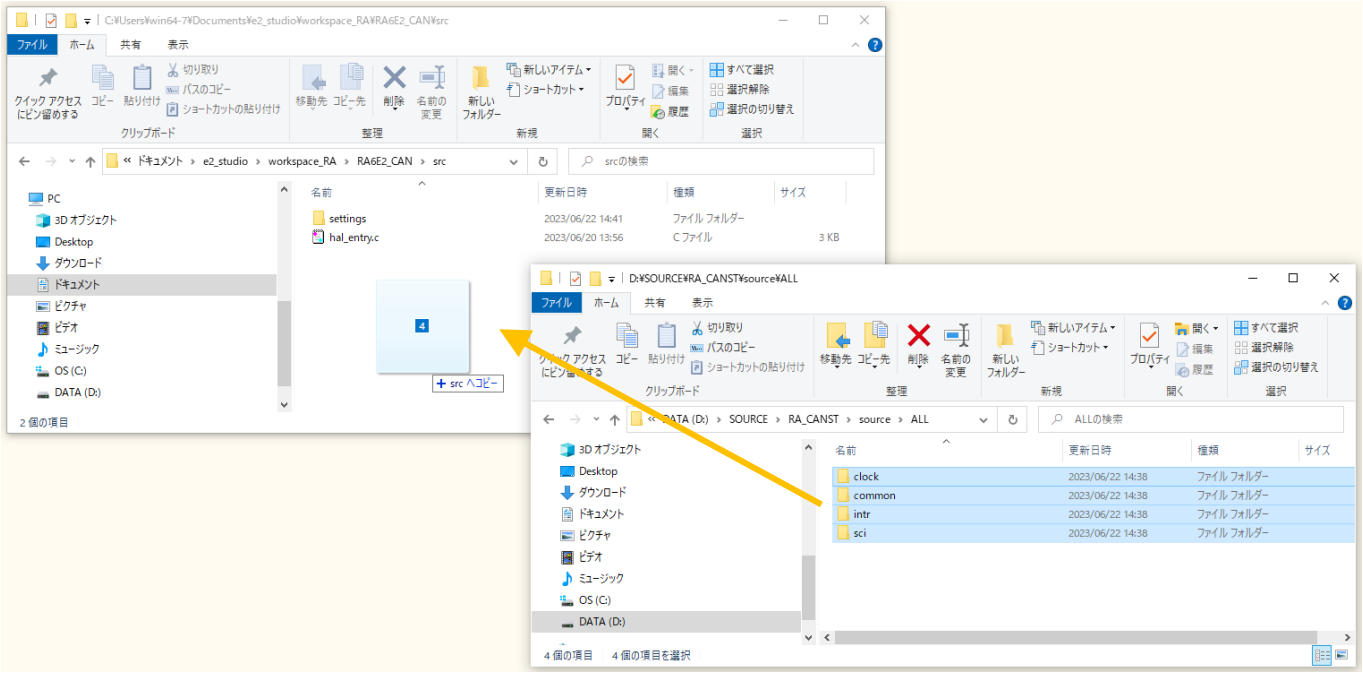
SOURCE¥RA_CANST¥mcu¥RA*xxx*¥settings
（*xxx* はマイコンタイプ名で分かれます）

上記フォルダを、src フォルダにコピー。



(c)共通で使用するソースをコピー

SOURCE¥RA_CANST¥source¥ALL 以下のフォルダをコピー。



(d)モジュール毎のソースをコピー

[CAN モジュールの場合]
SOURCE¥RA_CANST¥source¥CAN_module¥

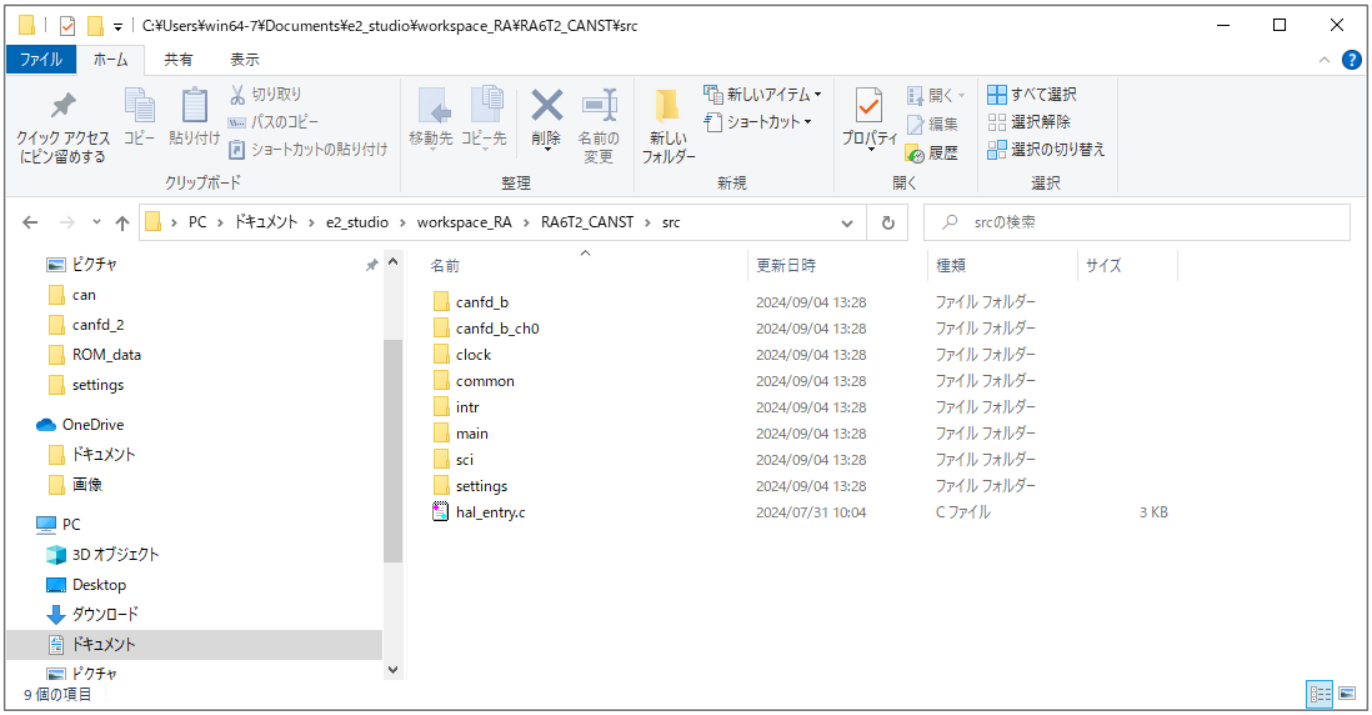
[CANFD モジュールの場合]
SOURCE¥RA_CANST¥source¥CANFD_module¥

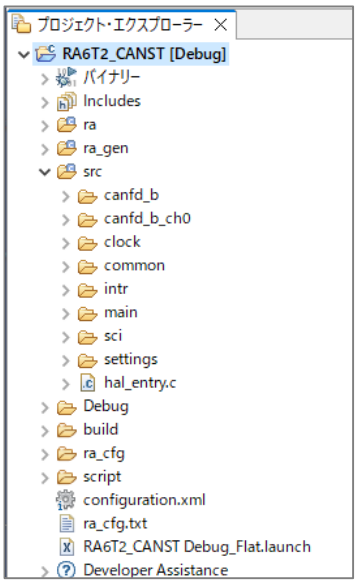
[CANFD_B モジュールの場合]
SOURCE¥RA_CANST¥source¥CANFD_B_module¥

[CANFD_2 モジュールの場合]
SOURCE¥RA_CANST¥source¥CANFD_2_module¥

以下のフォルダをコピー

CANFD_B モジュールの場合コピー後は以下の様になります。





ワークスペースフォルダに、ファイルやフォルダをコピーした場合、e2studio のプロジェクト・エクスプローラからも、リアルタイムにファイルやフォルダが見える様になるはずです。

ーsrc フォルダに追加するソースコードー

・CAN モジュール対応マイコン

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
can	source¥CAN_module¥can
can_ch0	source¥CAN_module¥can_ch0
can_ch1	source¥CAN_module¥can_ch1
main	source¥CAN_module¥main

・CANFD モジュール対応マイコン

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
canfd	source¥CANFD_module¥canfd
canfd_ch0	source¥CANFD_module¥canfd_ch0
canfd_ch1	source¥CANFD_module¥canfd_ch1
main	source¥CANFD_module¥main

・CANFD_B モジュール対応マイコン

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
canfd_b	source¥CANFD_B_module¥canfd_b
canfd_b_ch0	source¥CANFD_B_module¥canfd_b_ch0
main	source¥CANFD_B_module¥main

・CANFD_2 モジュール対応マイコン

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
canfd_2	source¥CANFD_2_module¥canfd_2
canfd_b_ch0	source¥CANFD_2_module¥canfd_2_ch0
canfd_b_ch1	source¥CANFD_2_module¥canfd_2_ch1
main	source¥CANFD_2_module¥main

・全てのマイコンで共通

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
intr	source¥ALL¥intr
clock	source¥ALL¥clock
common	source¥ALL¥common
sci	source¥ALL¥sci

・全てのマイコンで共通

コピー先フォルダ名 [Workspace_folder]¥[Project]¥src 以下	コピーするファイルの格納先 CD 内の¥SOURCE¥RA_CANST¥
settings	mcu¥RA***xxx***¥settings

RA***xxx*** の部分はマイコン種毎。

上記フォルダに加え、CD 内の¥SOURCE¥RA_CANST¥source¥hal_entry.c を [Workspace_folder]¥[Project]¥src フォルダに上書きしてください。

can~と main のフォルダは、

CAN_module

CANFD_module

CANFD_B_module

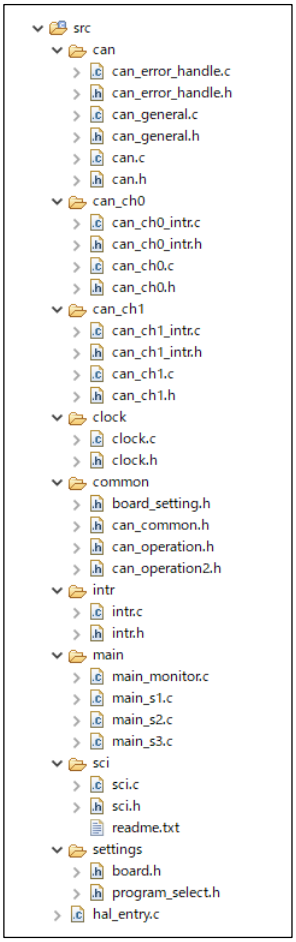
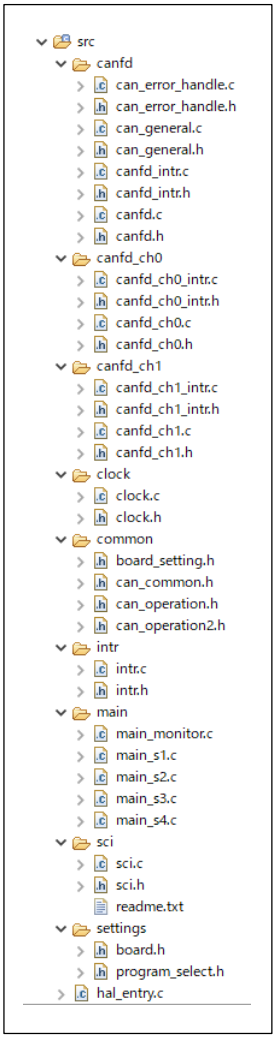
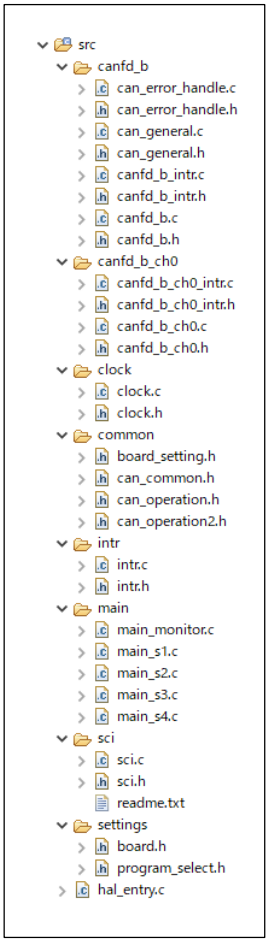
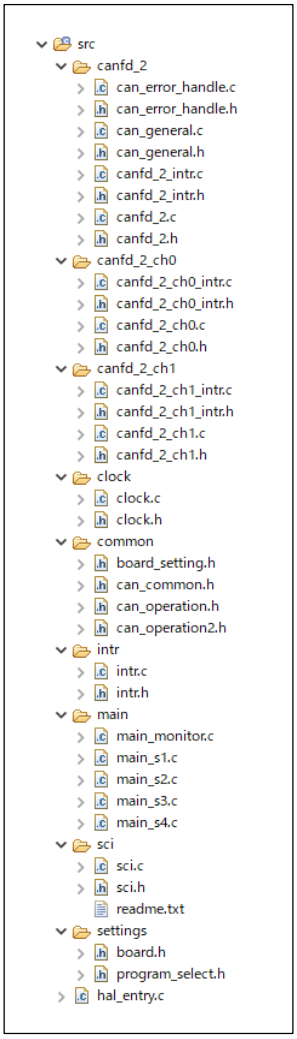
CANFD_2_module

以下を丸ごとコピー。

また、ALL 以下は、どのモジュールでも丸ごとコピー。

settings のフォルダは、マイコンタイプ毎に mcu¥RA***xxx*** 以下からコピーを行ってください。

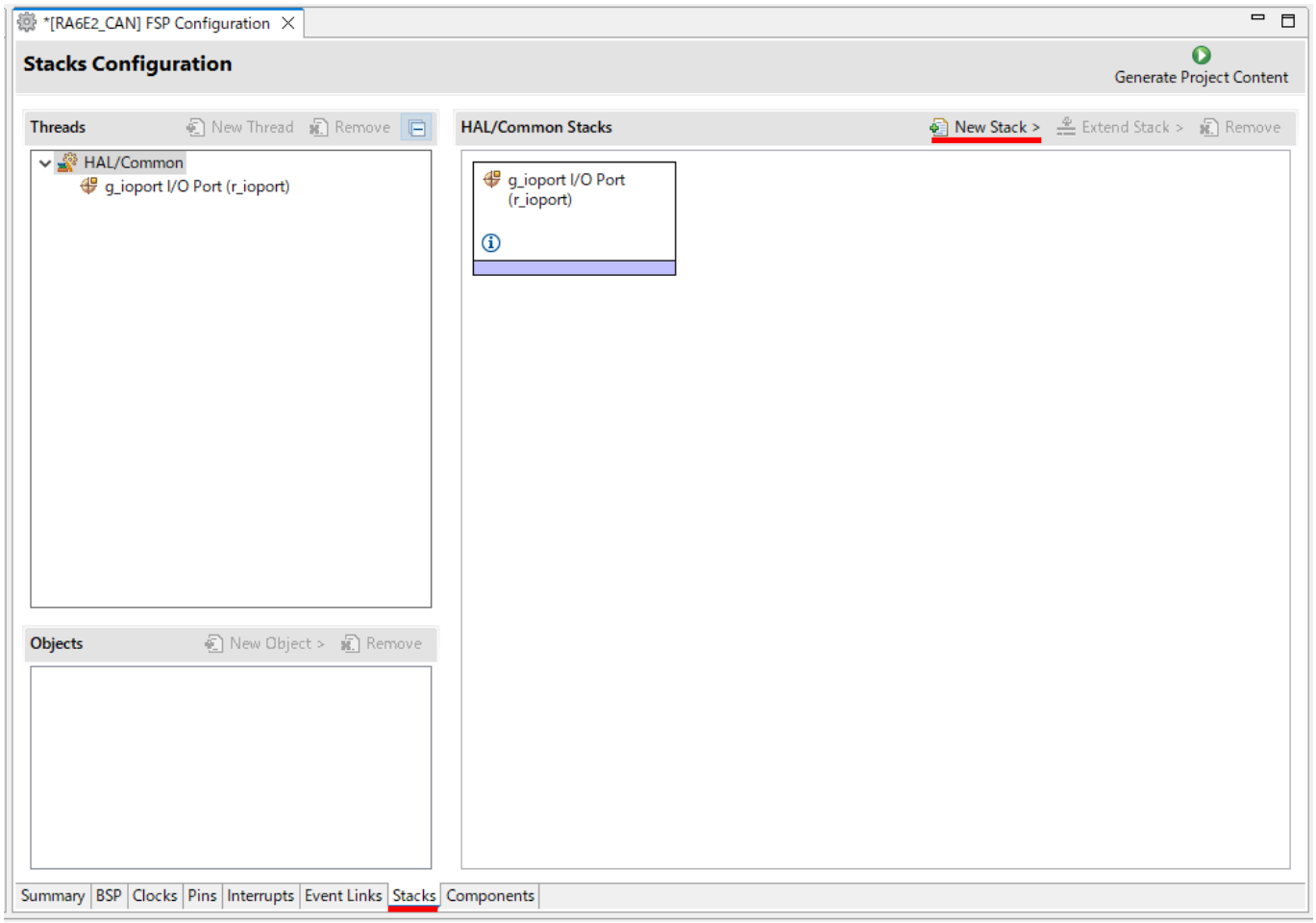
コピー後のプロジェクト・エクスプローラの階層は以下の様になります。

CAN モジュール 搭載マイコン	CANFD モジュール 搭載マイコン	CANFD_B モジュール 搭載マイコン	CANFD_2 モジュール 搭載マイコン
			

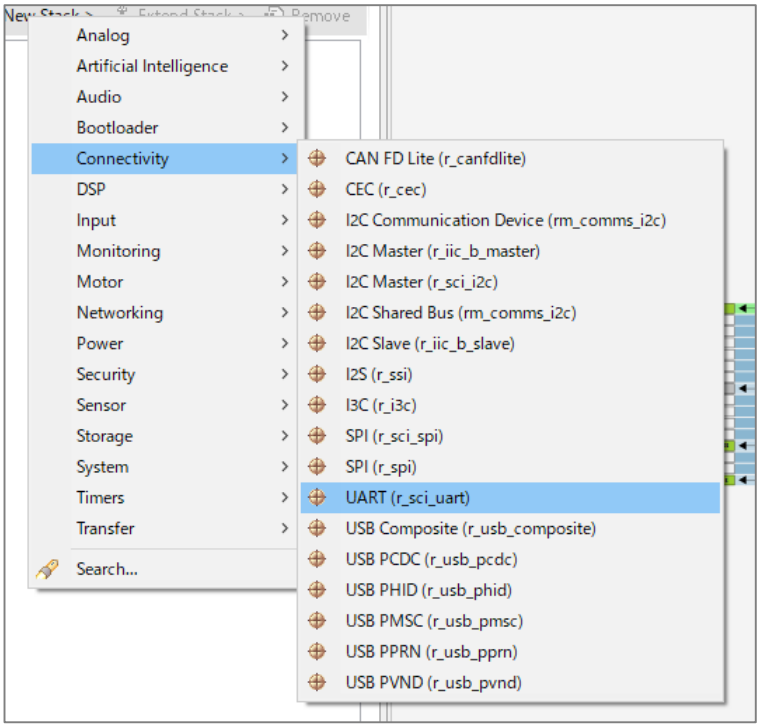
※CAN モジュール対応マイコンで、CAN が 1ch（CAN0 しかない）マイコンでも、can_ch1 フォルダはコピーしてください

(4)UART の設定

PC と通信を行うため、UART(SCI)の設定を行います。

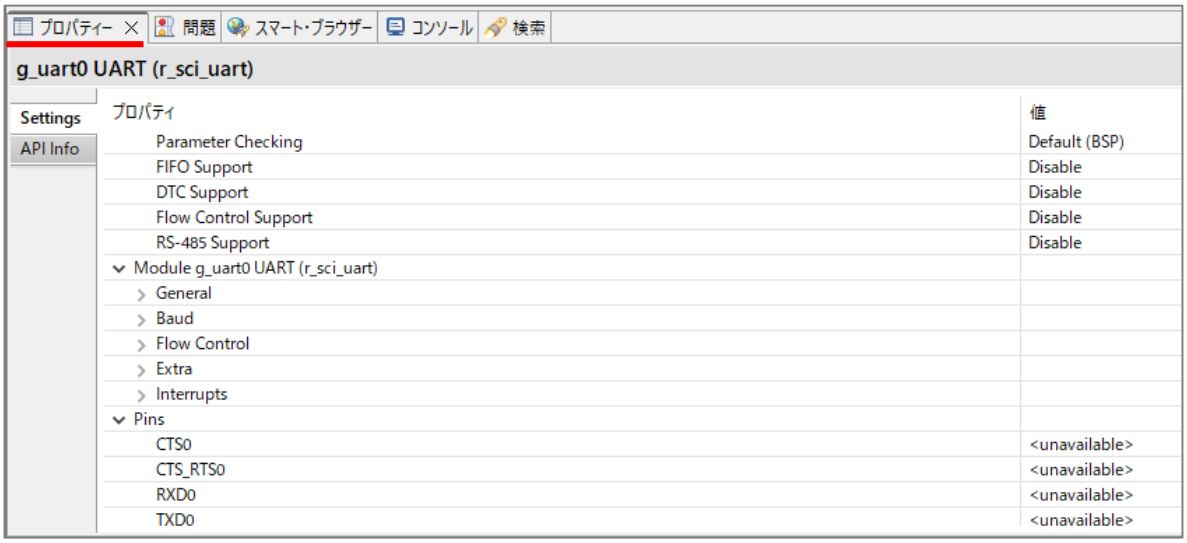


「Stacks」タブ、「New Stack」



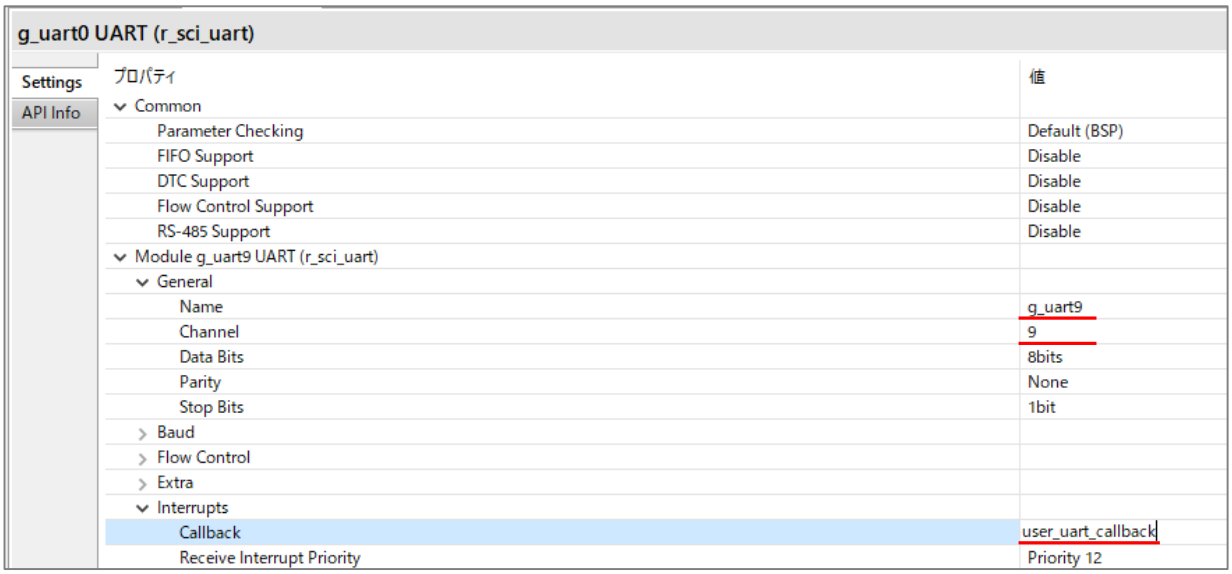
Connectivity – UART

を追加。



※RA6T2 等では、(r_sci_b_uart)となります

プロパティを選択。

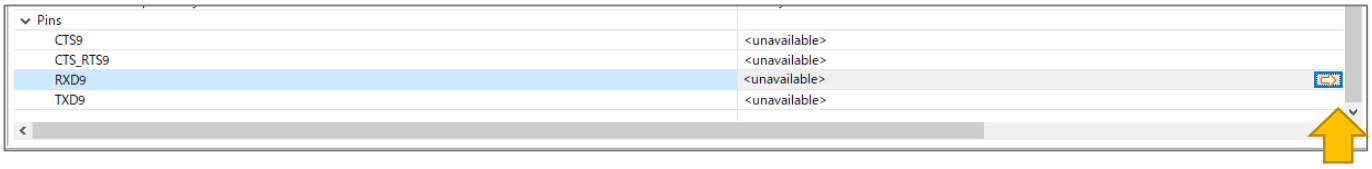


デフォルトから変更が必要な箇所

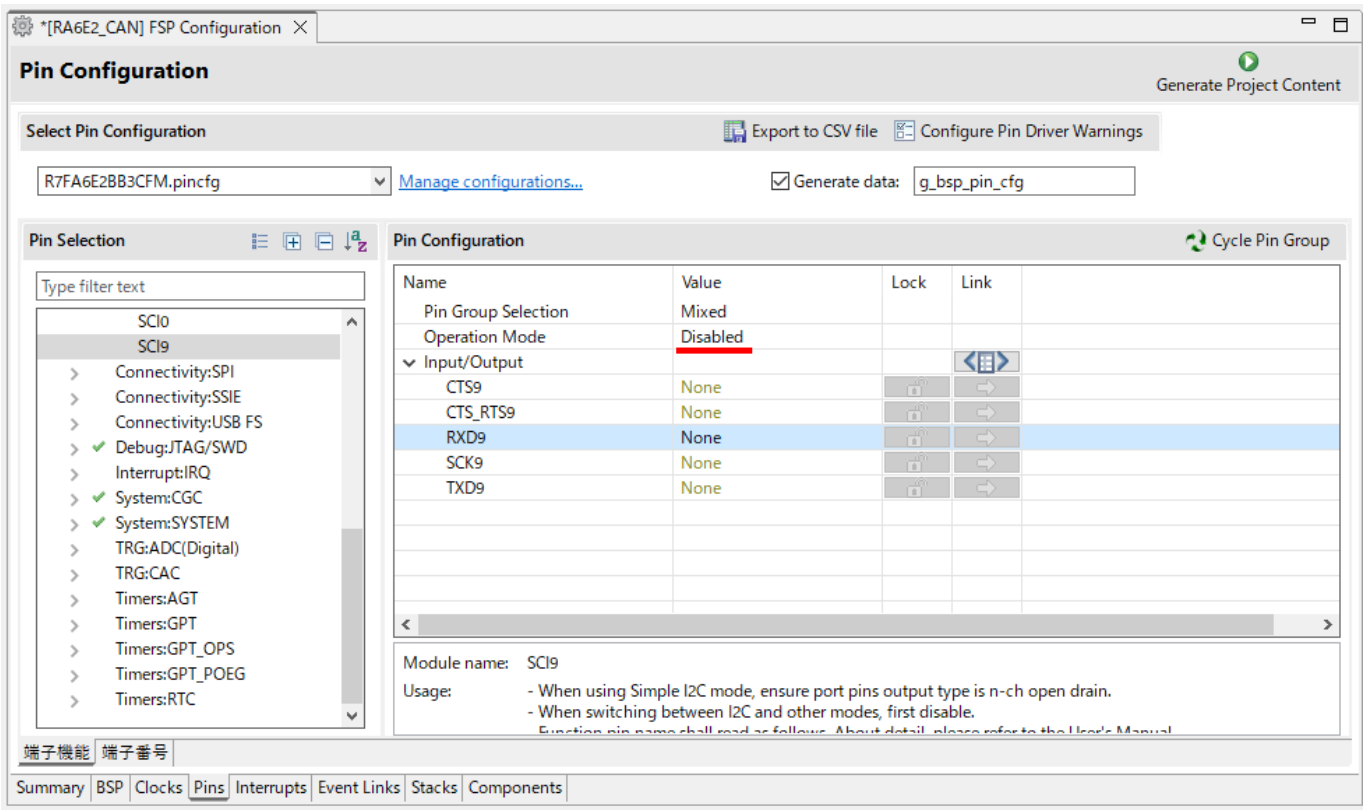
プロパティ	
General - Name	g_uart9
General – Channel	9
Interrupts - Callback	user_uart_callback

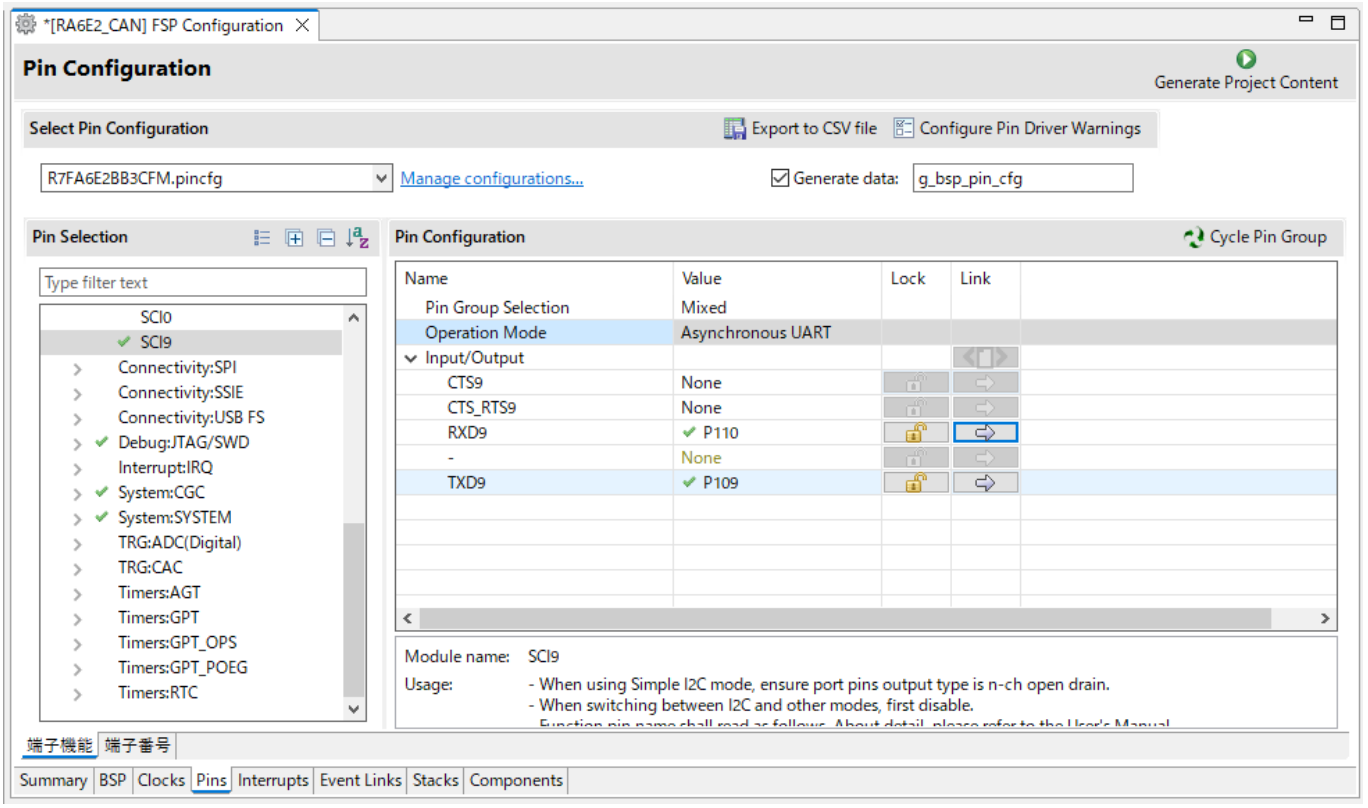
上記を設定してください。

※Baud 以下に速度設定値がありデフォルトは 115200 です（任意の速度に変更しても問題ありません）



Pins の RXD9,TXD9（どちらか）をクリックして→アイコンをクリック





Operation Mode を「Asynchronous UART」に設定。

端子は、

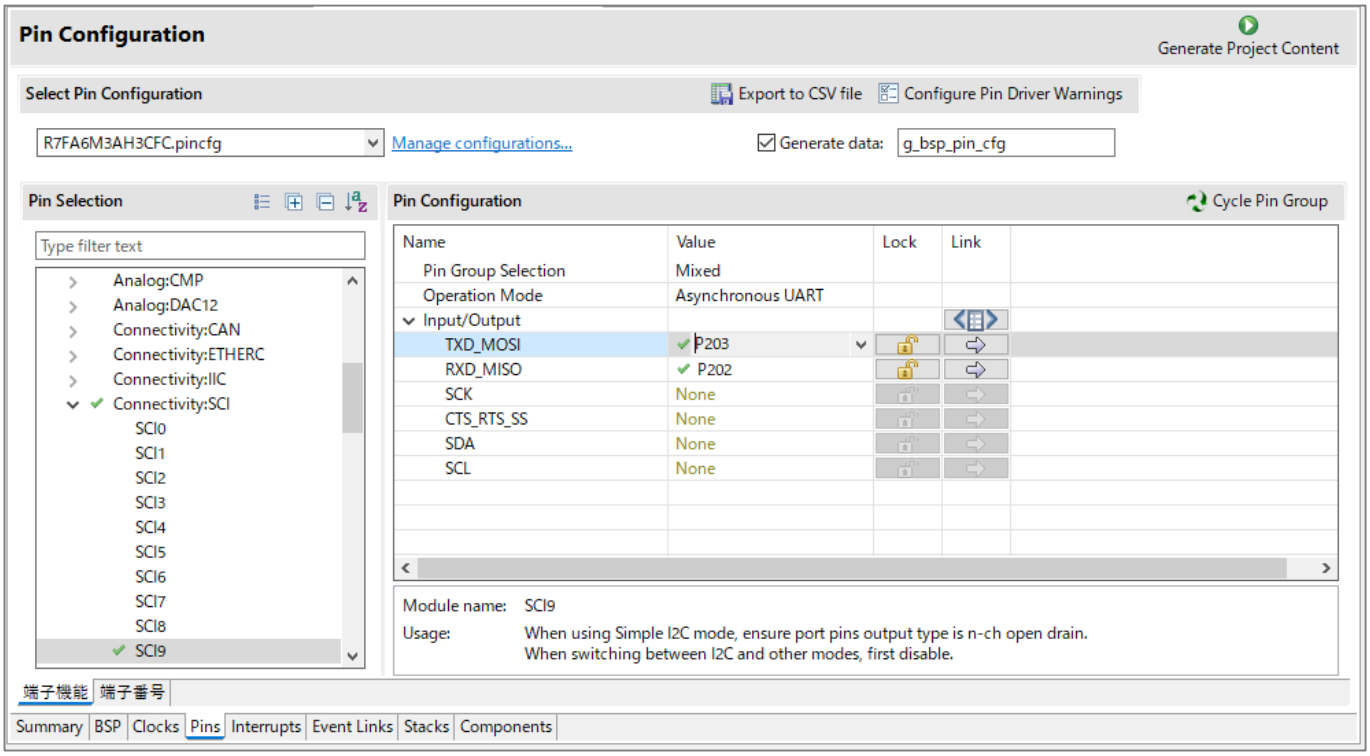
[参考　HSBRA6E2F64 取扱説明書]

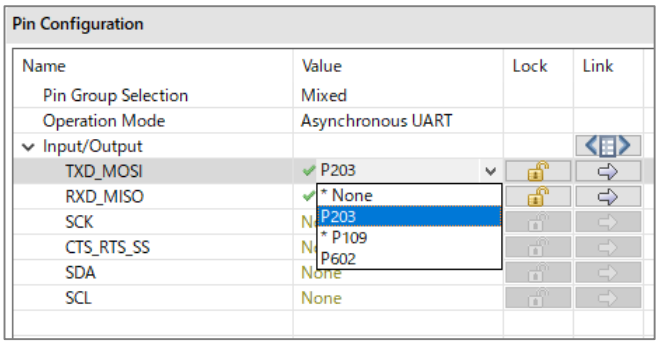
表 2-2 エミュレータインタフェース信号表 (J6)

No	マイコンピン番号	信号名	No	マイコンピン番号	信号名
1	32	P300/SWCLK	2	-	VSS
3	-	(NC)	4	-	(NC)
5	34	**P109/TXD9**	6	-	(NC)
7	(33)(*1)	(P108/SWDIO)	8	-	VCC
9	(33)(*2)	(P108/SWDIO)	10	-	(NC)
11	35	**P110/RXD9**	12	-	VSS
13	25	*RES	14	-	VSS

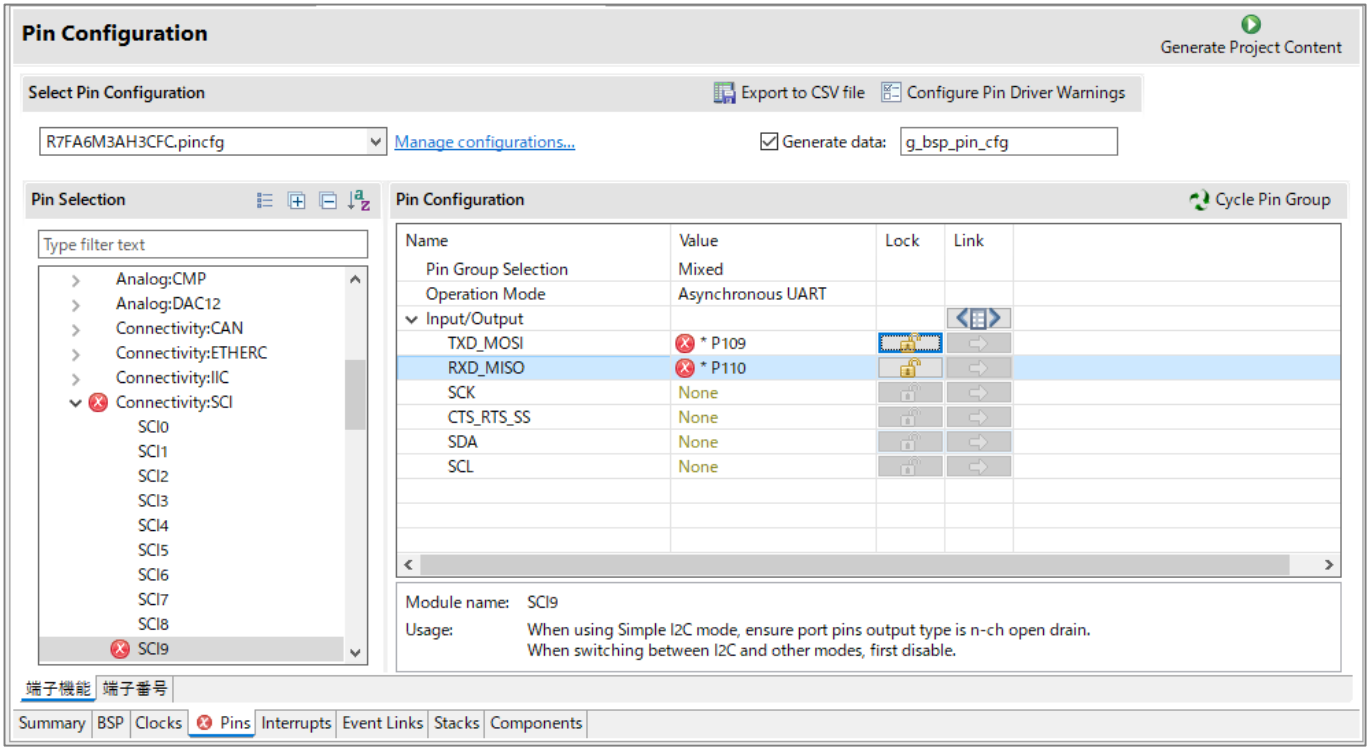
14P コネクタの 5 番ピンが TXD9 の設定(P109)、11 番ピンが RXD9 の設定(P110)と合っていれば問題ありません。

例えば、RA6M3 では、デフォルトでは端子は下記の設定となります。





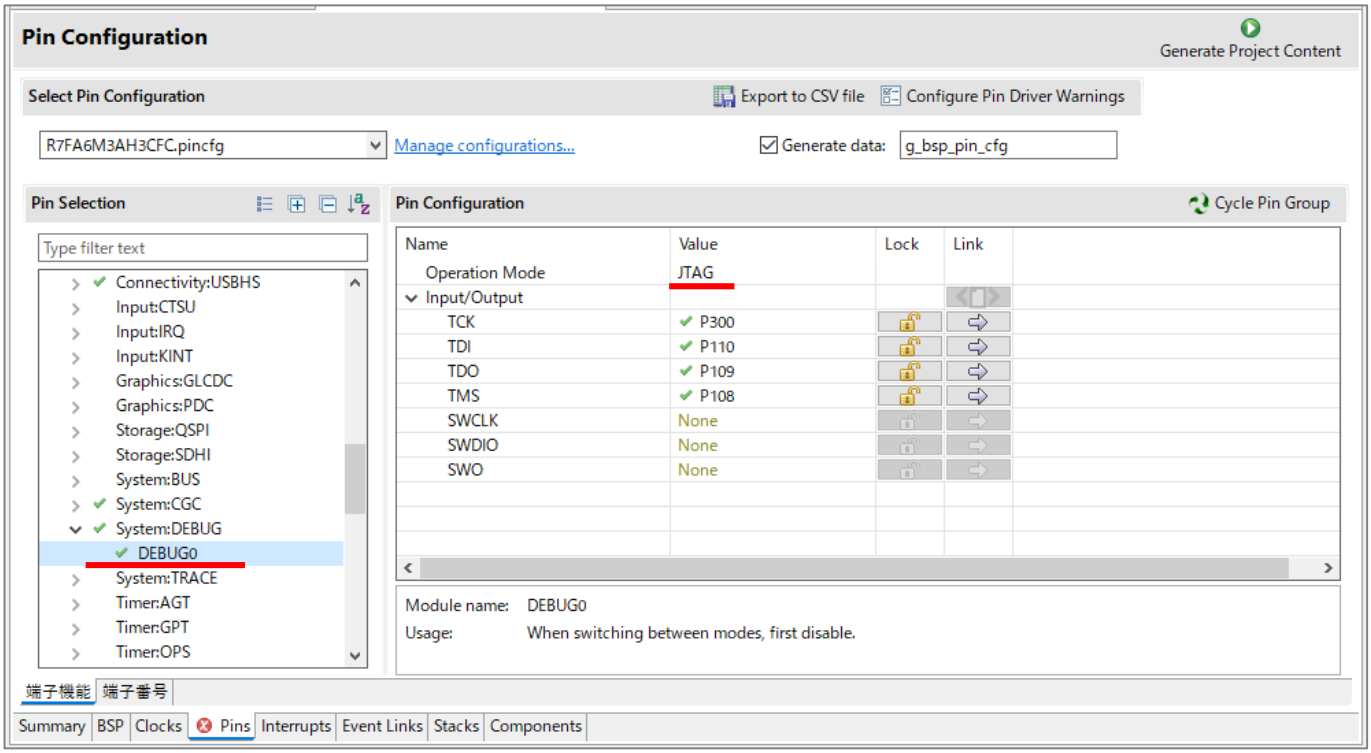
この場合、端子部をクリックすると、TXD9 の割り当てを別な端子に変更が可能です。



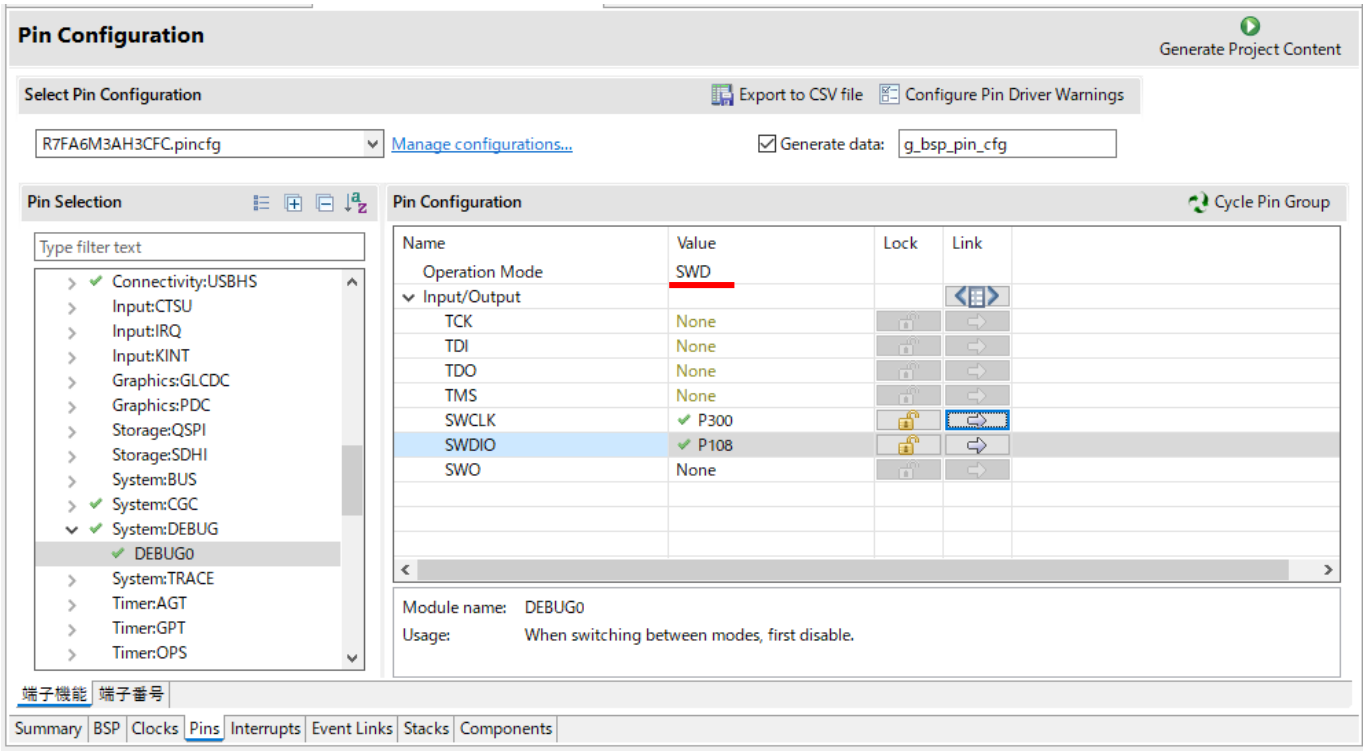
変更は可能ですが、エラーとなります（端子の競合）。

－TXD, RXD 設定端子－

	RA8x	RA6T2	左記以外の RA
TXD9	P209	PB03	P109
RXD9	P208	PA15	P110



DEBUG の項目を選択。



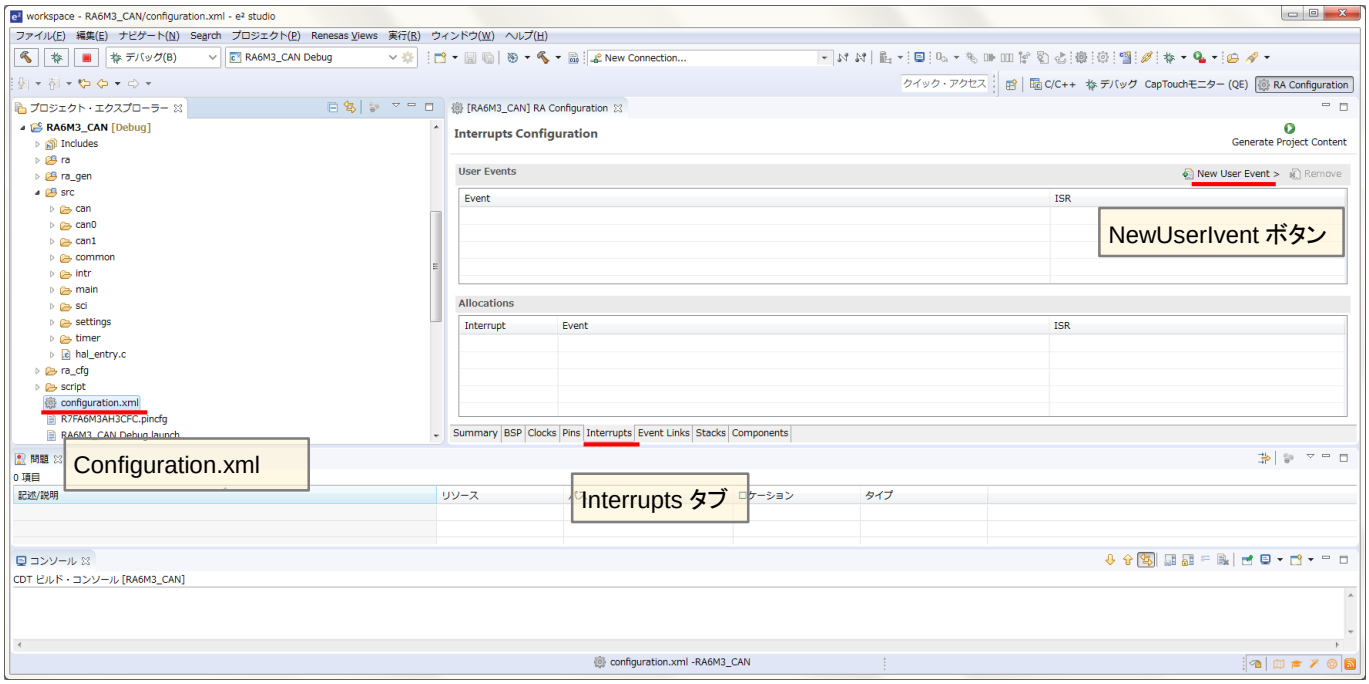
Operation Mode を JTAG から SWD に変更。SWCLK-P300, SWDIO-P108 を選択。

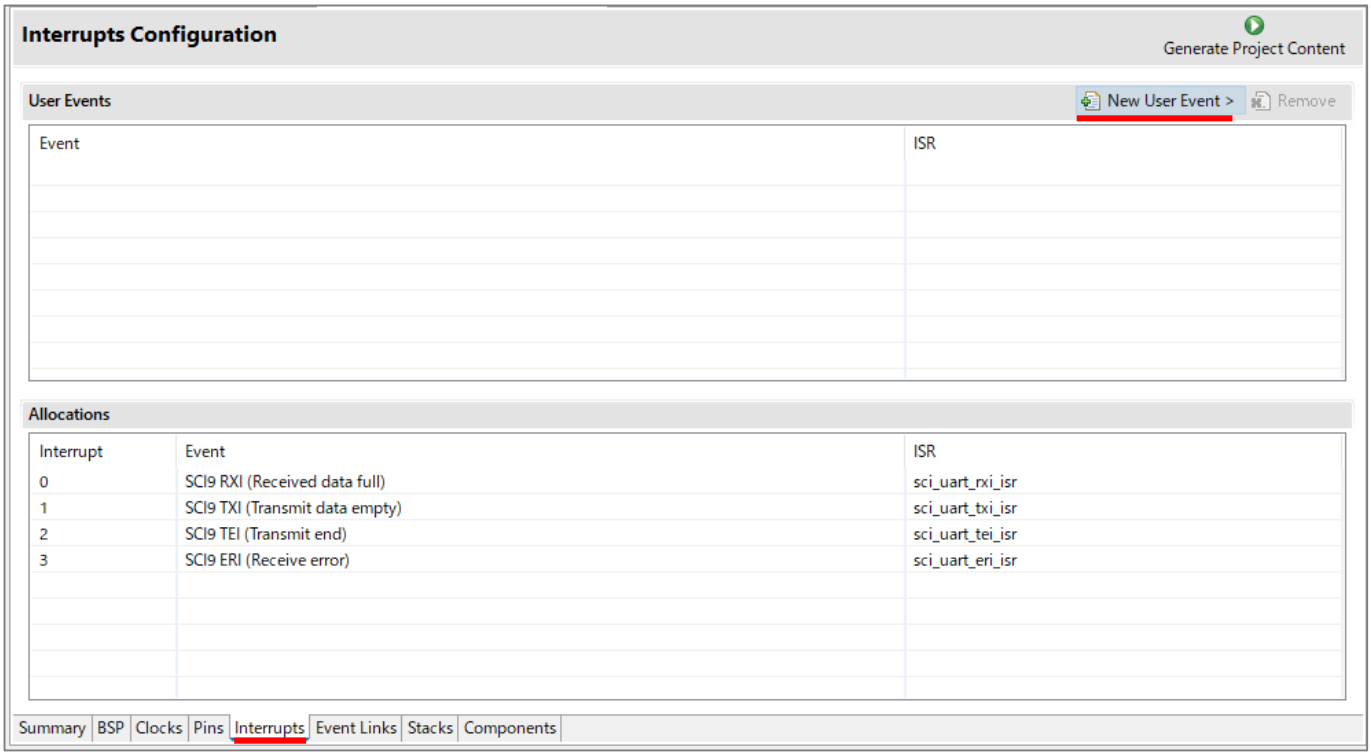
上記変更で、端子の競合エラーは解消されます。

（※JTAG に非対応のマイコンの場合は変更不要です）

(4)割り込み設定

FSP では、割り込みが GUI 上で設定できる様になっています。

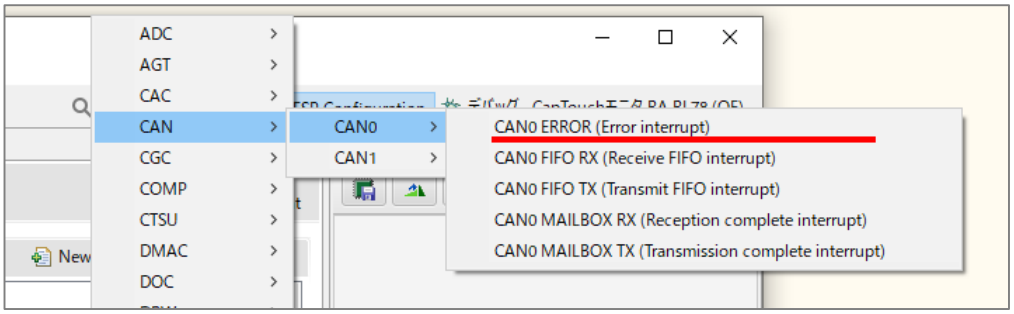




UART 設定後は、上記の様に UART(SCI)関連の割り込みが登録された状態になっています。

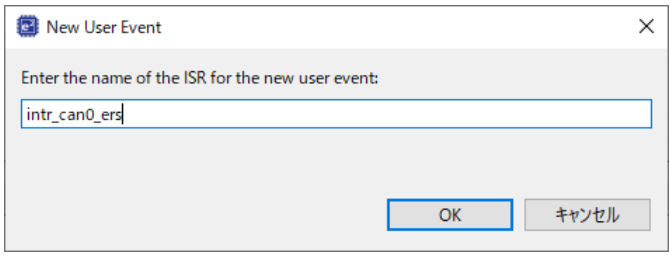
「New User Event」ボタンを押してください。

－CAN モジュール搭載マイコンの場合－

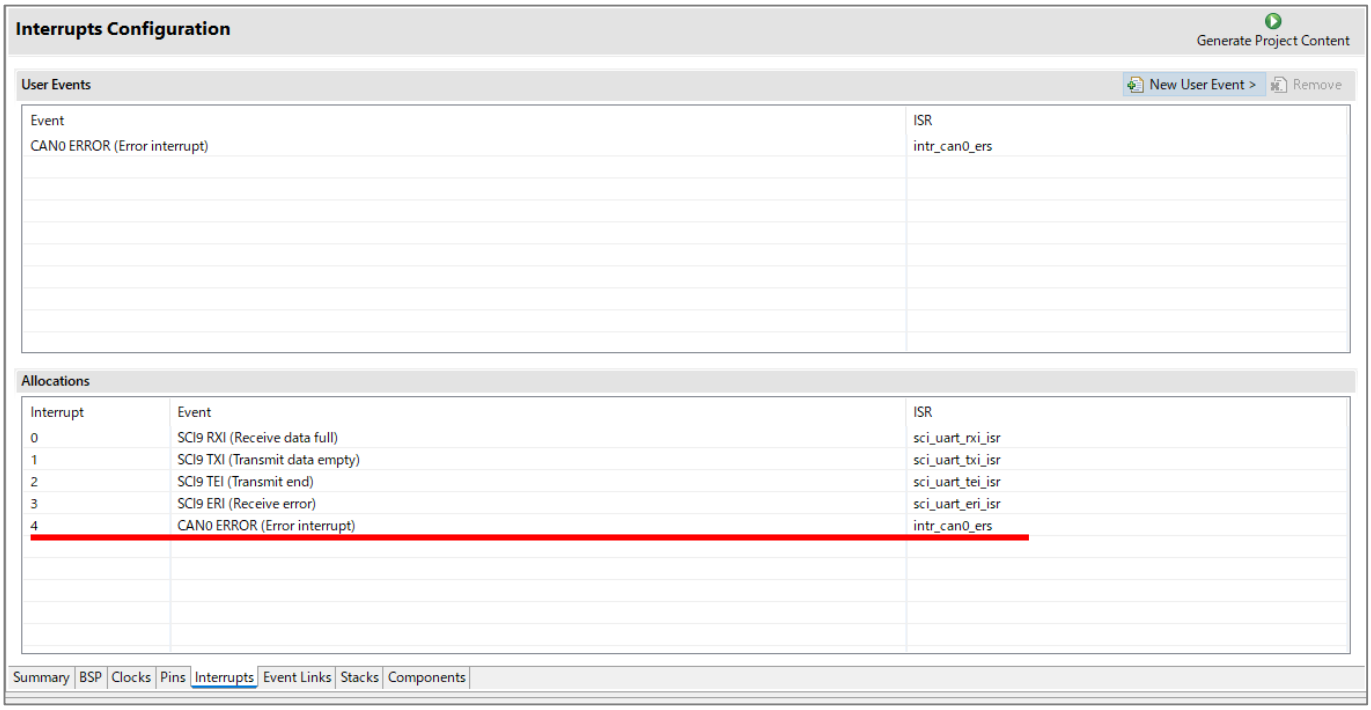


機能毎に割り込みを選べるようになっていますので、CAN 関連の割り込みをここで追加します。

CAN – CAN0 – CAN0 ERROR を選択します。



名称を入力するボックスが出ますので、
intr_can0_ers
と入力してください。ここで、入力した名称は関数名となります。
（CAN0 でエラーが生じた際に、intr_can0_ers()関数が呼ばれる様になる）



割り込みの追加を行ったので、割り込み番号 4 番に、追加した intr_can0_ers が登録されました。

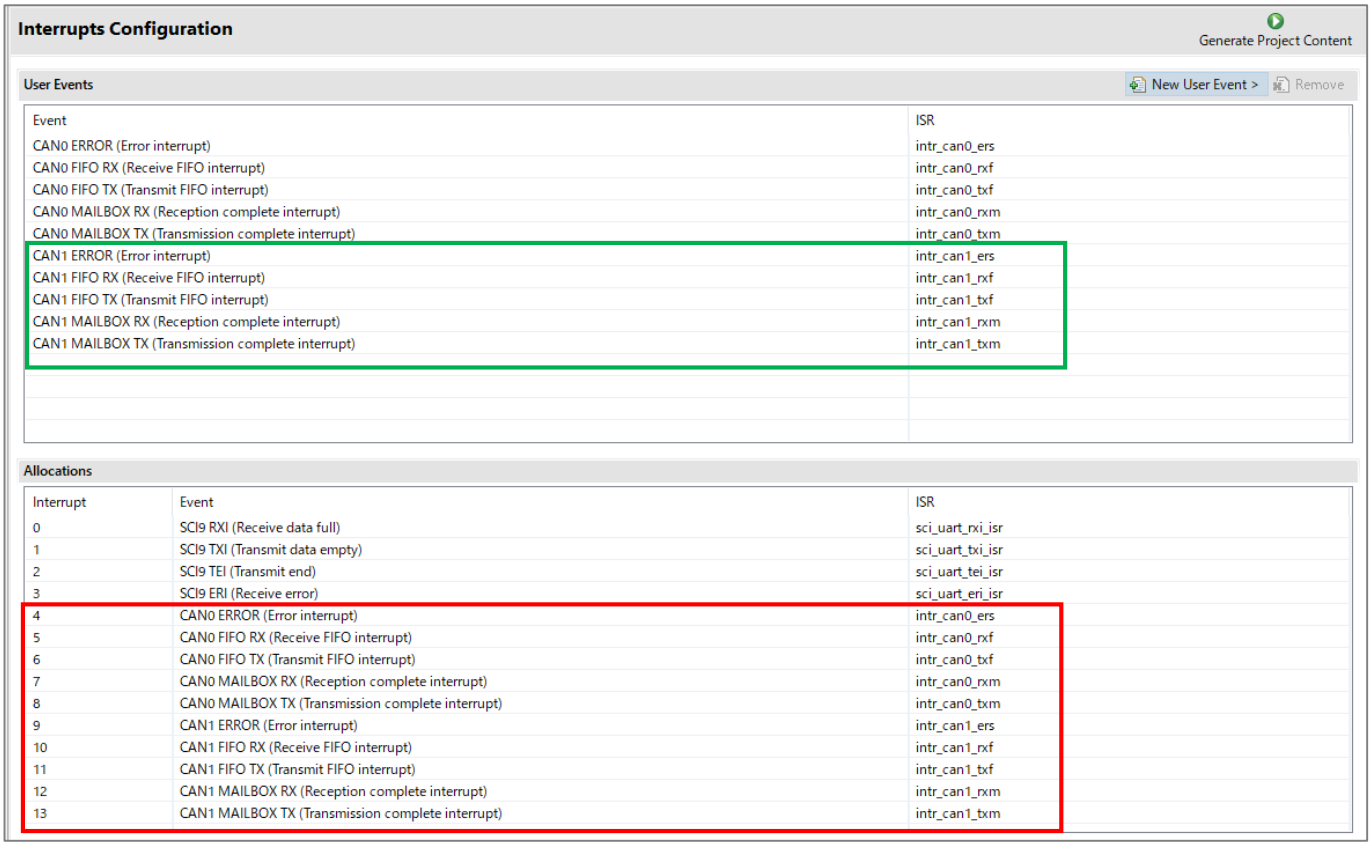
割り込み番号（ここでは 4）は、追加した順や、（RA2L1 等のマイコンでは）割り込み機能毎に決められた番号となりますが、何番に割り振られても問題ありません。（割り込み番号は、特に意味を持ちません。）

CAN モジュール対応マイコンの場合は、以下の割り込みを追加定義してください。

グループ	モジュール	種別	定義名
CAN	CAN0	CAN0 ERROR	intr_can0_ers
CAN	CAN0	CAN0 FIFO RX	intr_can0_rxf
CAN	CAN0	CAN0 FIFO TX	intr_can0_txf
CAN	CAN0	CAN0 MAILBOX RX	intr_can0_rxm
CAN	CAN0	CAN0 MAILBOX TX	intr_can0_txm
CAN	CAN1	CAN1 ERROR	intr_can1_ers (*1)
CAN	CAN1	CAN1 FIFO RX	intr_can1_rxf (*1)
CAN	CAN1	CAN1 FIFO TX	intr_can1_txf (*1)
CAN	CAN1	CAN1 MAILBOX RX	intr_can1_rxm (*1)
CAN	CAN1	CAN1 MAILBOX TX	intr_can1_txm (*1)

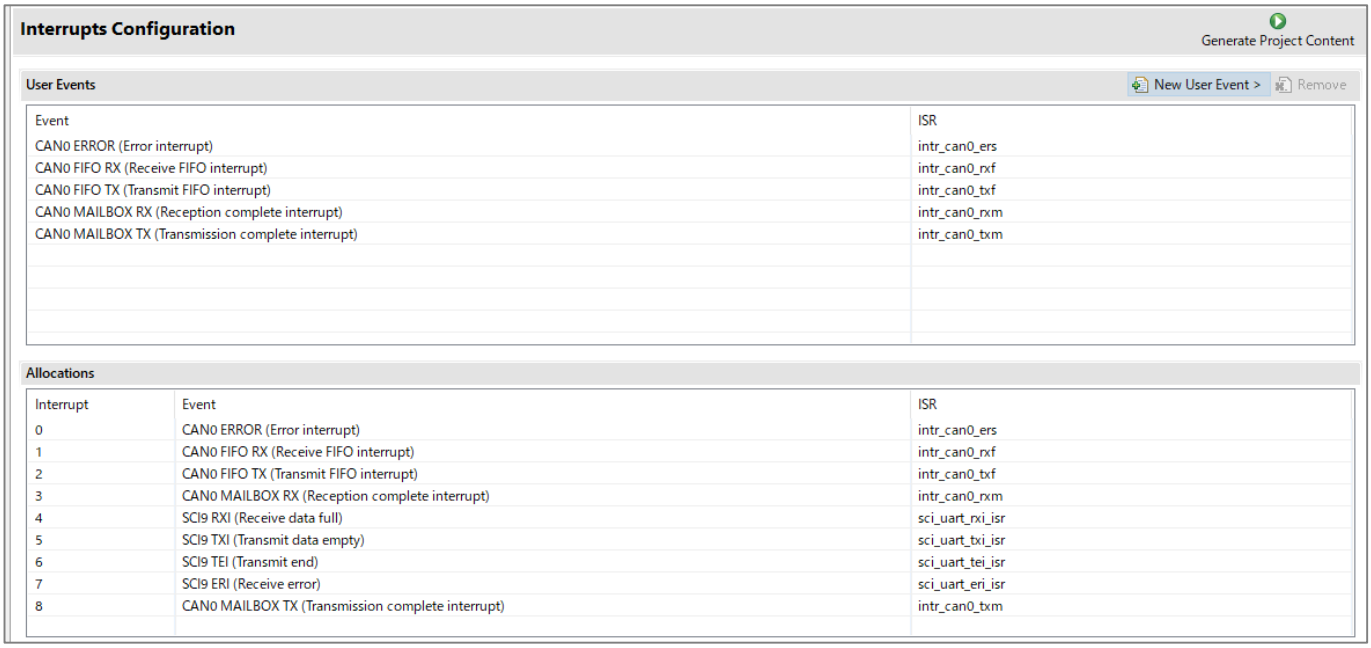
(*1)CAN を 2ch 搭載しているマイコン（RA6M3 等）のみ定義

※RA2L1 等、割り込み番号が機能と紐づけされているタイプのマイコンがあり、その場合割り込み番号が追加順にはなりません



CAN モジュール対応マイコンでは追加後は上記の様になります。CAN が 1ch しかないマイコンでは、緑枠部分なしとなります。割り込み番号 0~13 は追加順によっては番号が上記とは異なるケースがありますが、特に問題はありません。（割り込み機能と、関数名の対応が設定されていれば問題ありません。）

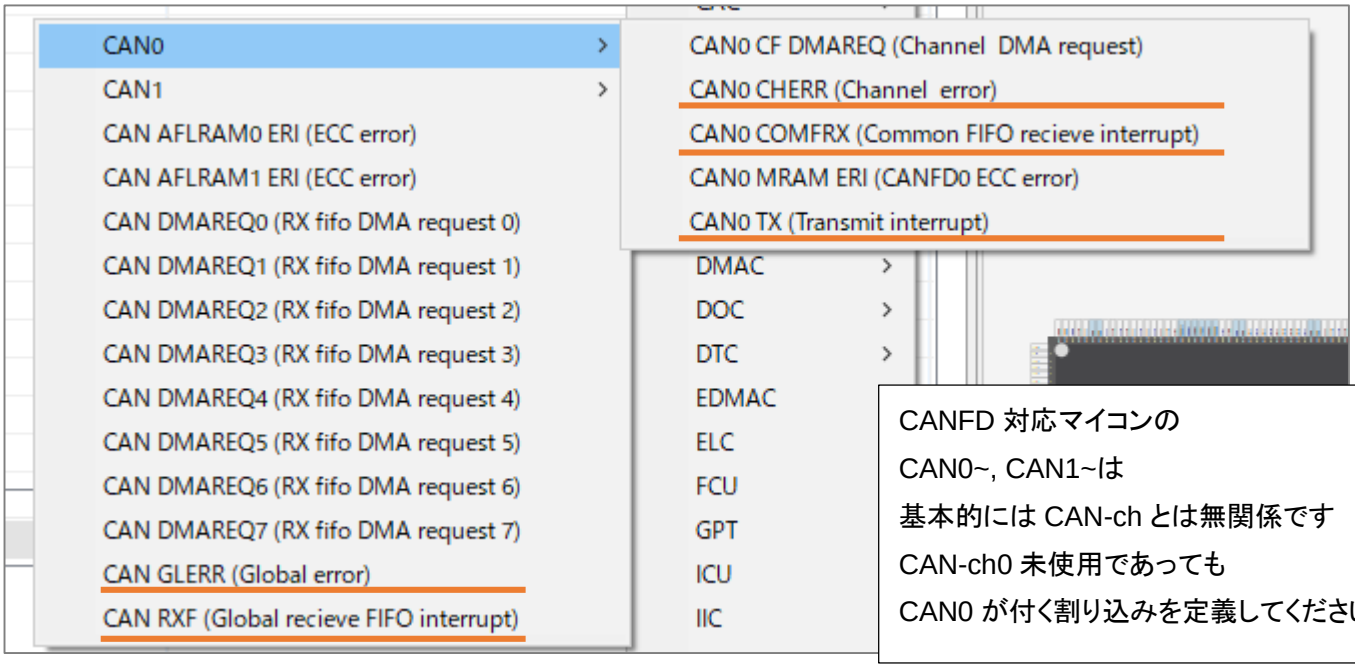
・RA2L1 の場合



RA2L1 の割り込み設定は、機能毎に割り当てられる割り込み番号がグループ化されており、割り込み番号は追加順の連番には割り当てされません。

－CANFD モジュール搭載マイコンの場合－

CANFD モジュール対応マイコンの場合は、以下の割り込みを追加定義してください。

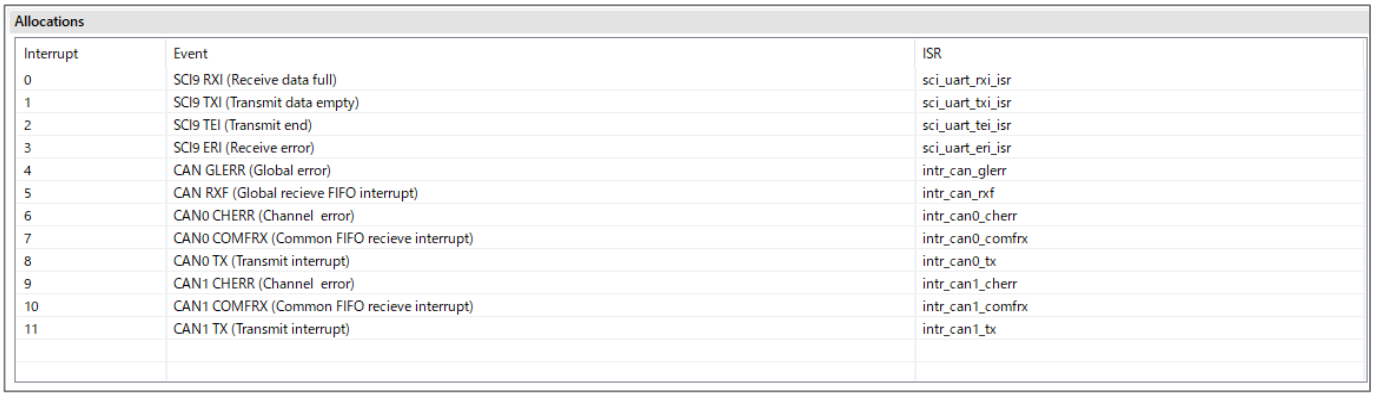


CANFD 対応マイコンの
CAN0~, CAN1~は
基本的には CAN-ch とは無関係です
CAN-ch0 未使用であっても
CAN0 が付く割り込みを定義してください

グループ	モジュール	種別	定義名
CAN		CAN GLERR	intr_can_glerr
CAN		CAN RXF	intr_can_rxf
CAN	CAN0	CAN0 CHERR	intr_can0_cherr
CAN	CAN0	CAN0 COMFRX	intr_can0_comfrx
CAN	CAN0	CAN0 TX	intr_can0_tx
CAN	CAN1	CAN1 CHERR	intr_can1_cherr
CAN	CAN1	CAN1 COMFRX	intr_can1_comfrx
CAN	CAN1	CAN1 TX	intr_can1_tx

※CANn CHERR, CANn COMFRX, CANn TX は、CAN - CANn の階層の下にあります

追加後は、以下の様になります。

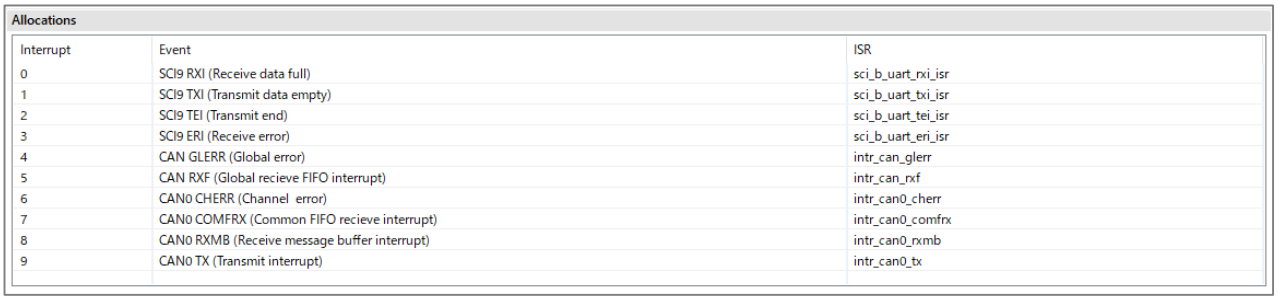


※表示順が変わる事は問題ありません

－CANFD_B モジュール搭載マイコンの場合－

グループ	モジュール	種別	定義名
CAN		CAN GLERR	intr_can_glerr
CAN		CAN RXF	intr_can_rxf
CAN	CAN0	CAN0 CHERR	intr_can0_cherr
CAN	CAN0	CAN0 COMFRX	intr_can0_comfrx
CAN	CAN0	CAN0 RXMB	intr_can0_rxmb
CAN	CAN0	CAN0 TX	intr_can0_tx

追加後は、以下の様になります。



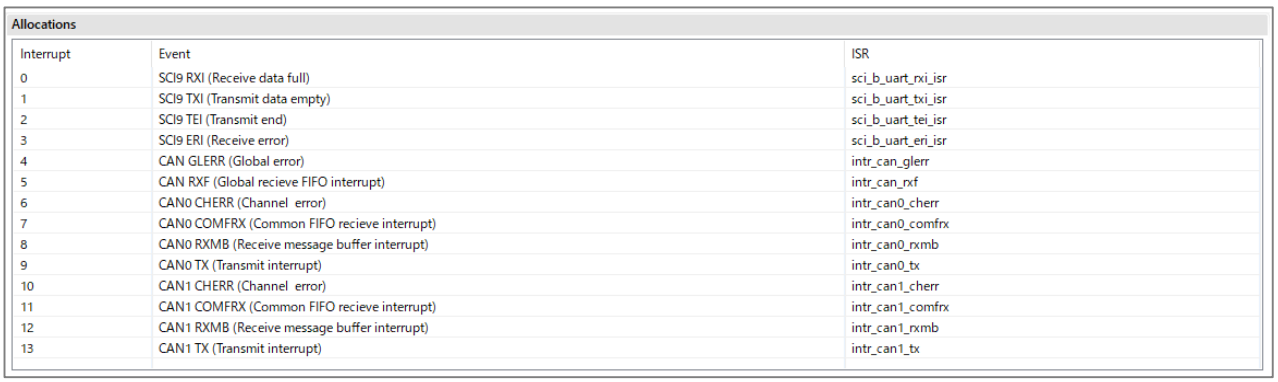
Allocations

Interrupt	Event	ISR
0	SCI9 RXI (Receive data full)	sci_b_uart_rxi_isr
1	SCI9 TXI (Transmit data empty)	sci_b_uart_txi_isr
2	SCI9 TEI (Transmit end)	sci_b_uart_tei_isr
3	SCI9 ERI (Receive error)	sci_b_uart_eri_isr
4	CAN GLERR (Global error)	intr_can_glerr
5	CAN RXF (Global recieve FIFO interrupt)	intr_can_rxf
6	CAN0 CHERR (Channel error)	intr_can0_cherr
7	CAN0 COMFRX (Common FIFO recieve interrupt)	intr_can0_comfrx
8	CAN0 RXMB (Receive message buffer interrupt)	intr_can0_rxmb
9	CAN0 TX (Transmit interrupt)	intr_can0_tx

－CANFD_2 モジュール搭載マイコンの場合－

グループ	モジュール	種別	定義名
CAN		CAN GLERR	intr_can_glerr
CAN		CAN RXF	intr_can_rxf
CAN	CAN0	CAN0 CHERR	intr_can0_cherr
CAN	CAN0	CAN0 COMFRX	intr_can0_comfrx
CAN	CAN0	CAN0 RXMB	intr_can0_rxmb
CAN	CAN0	CAN0 TX	intr_can0_tx
CAN	CAN1	CAN1 CHERR	intr_can1_cherr
CAN	CAN1	CAN1 COMFRX	intr_can1_comfrx
CAN	CAN1	CAN1 RXMB	intr_can1_rxmb
CAN	CAN1	CAN1 TX	intr_can1_tx

追加後は、以下の様になります。



Allocations

Interrupt	Event	ISR
0	SCI9 RXI (Receive data full)	sci_b_uart_rxi_isr
1	SCI9 TXI (Transmit data empty)	sci_b_uart_txi_isr
2	SCI9 TEI (Transmit end)	sci_b_uart_tei_isr
3	SCI9 ERI (Receive error)	sci_b_uart_eri_isr
4	CAN GLERR (Global error)	intr_can_glerr
5	CAN RXF (Global recieve FIFO interrupt)	intr_can_rxf
6	CAN0 CHERR (Channel error)	intr_can0_cherr
7	CAN0 COMFRX (Common FIFO recieve interrupt)	intr_can0_comfrx
8	CAN0 RXMB (Receive message buffer interrupt)	intr_can0_rxmb
9	CAN0 TX (Transmit interrupt)	intr_can0_tx
10	CAN1 CHERR (Channel error)	intr_can1_cherr
11	CAN1 COMFRX (Common FIFO recieve interrupt)	intr_can1_comfrx
12	CAN1 RXMB (Receive message buffer interrupt)	intr_can1_rxmb
13	CAN1 TX (Transmit interrupt)	intr_can1_tx

Allocations

Interrupt	Event	ISR
0	SCI9 RXI (Receive data full)	sci_uart_rxi_isr
1	SCI9 TXI (Transmit data empty)	sci_uart_txi_isr
2	SCI9 TEI (Transmit end)	sci_uart_tei_isr
3	SCI9 ERI (Receive error)	sci_uart_eri_isr
4	CAN0 ERROR (Error interrupt)	intr_can0_ers
5	CAN0 FIFO RX (Receive FIFO interrupt)	intr_can0_rxf
6	CAN0 FIFO TX (Transmit FIFO interrupt)	intr_can0_txf
7	CAN0 MAILBOX RX (Reception complete interrupt)	intr_can0_rxm
8	CAN0 MAILBOX TX (Transmission complete interrupt)	intr_can0_txm
9	CAN1 ERROR (Error interrupt)	intr_can1_ers
10	CAN1 FIFO RX (Receive FIFO interrupt)	intr_can1_rxf
11	CAN1 FIFO TX (Transmit FIFO interrupt)	intr_can1_txf
12	CAN1 MAILBOX RX (Reception complete interrupt)	intr_can1_rxm
13	CAN1 MAILBOX TX (Transmission complete interrupt)	intr_can1_txm

割り込み定義時に設定した名称（赤枠部）は、割り込み関数名に対応します。

src¥can_ch0¥can_ch0_intr.c

```
/*-----------------------------------------------------------------------
    割り込み関数
-----------------------------------------------------------------------*/
void intr_can0_ers(void);
void intr_can0_rxf(void);
void intr_can0_txf(void);
void intr_can0_rxm(void);
void intr_can0_txm(void);

/*-----------------------------------------------------------------------
    関数本体
-----------------------------------------------------------------------*/
//CANエラー検出
#ifdef INTR_CAN0_ERS
void intr_can0_ers(void)
```

例えば CAN モジュール搭載マイコン向けのプログラムでは、can_ch0_intr.c 内に、割り込み関数

intr_can0_ers

intr_can0_rxf

intr_can0_txf

intr_can0_rxm

intr_can0_rxm

が定義されています。関数名と、FSP で設定する名称が同じくなる様に設定してください。（FSP 上の定義名称を変更する場合、ソースコード内の関数名を変更してください。）

※関数名に関して

関数名は、FSP での設定と、xxx_intr.c（割り込み関数の実体を定義したソースファイル）内の関数名が一致していれば、任意の名称で構いません。本サンプルプログラムでは、ハードウェアマニュアル内の、割り込みコントローラ(ICU)項のイベントテーブルの「名称」を基にしています。

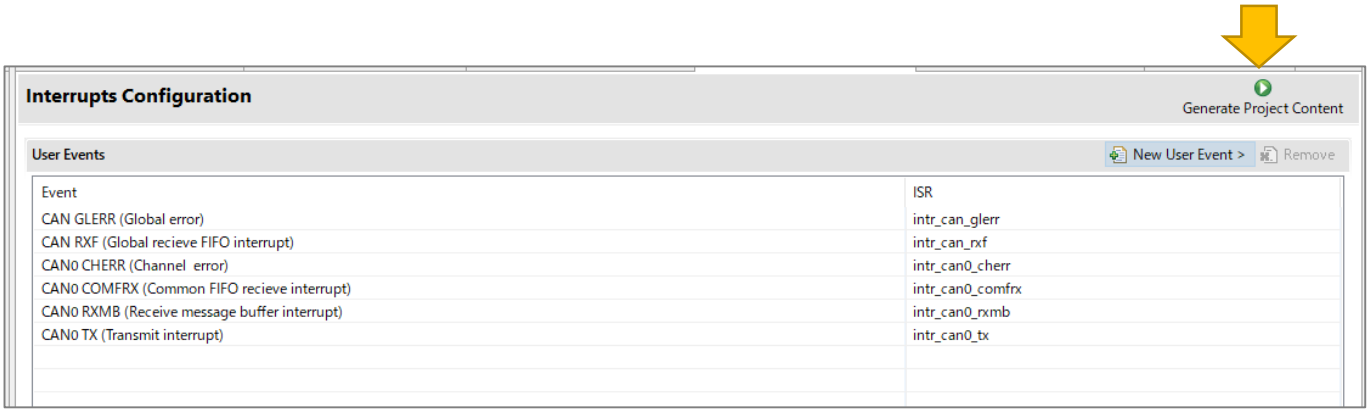
名称　　CAN0_ERS

↓

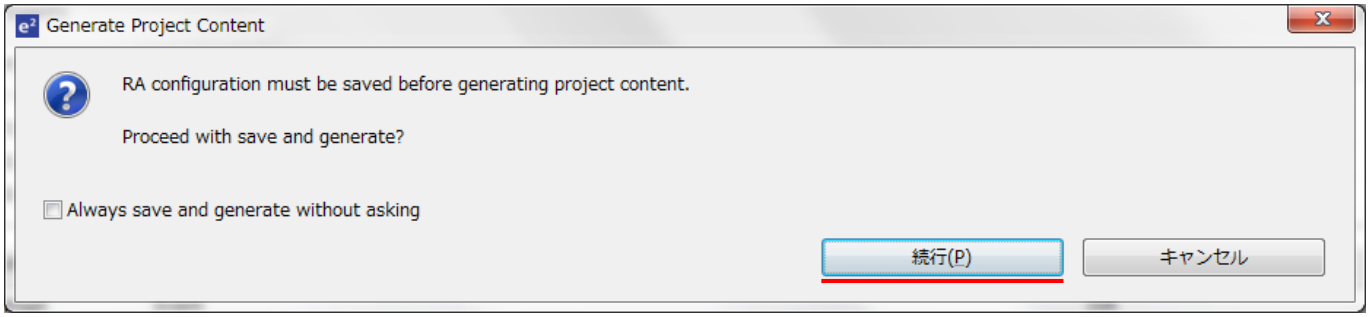
割り込み関数名　intr_can0_ers

（先頭に intr_を付与、大文字を小文字に変更）

一連の設定が終わった後、



Generate Project Content のボタンを押して、プログラムコードの生成を行ってください。



上記の画面が出た場合は、「続行」してください。

この操作で、クロック設定や割り込み設定等が反映されたソースコードが生成されます。

（src 以下は、ユーザのプロラムコードを格納する場所なので、書き換え等は行われません。ra_gen 以下等の FSP が生成するコードが生成、上書きされます。）

以上で、プロジェクトの新規作成は完了ですので、3.2 章の手順でプログラムのビルドを行ってください。

5. サンプルプログラムの説明(共通部分)

5.1. サンプルプログラムの選択

サンプルプログラムは、

SAMPLE1	割り込みを使用しないシンプルな送受信プログラム データフレームのみ取り扱う CAN パケットのみ取り扱う
SAMPLE2	**割り込みを使用**する送受信プログラム データフレームのみ取り扱う CAN パケットのみ取り扱う
SAMPLE3	割り込みを使用する送受信プログラム データフレームと**リモートフレームを取り扱う** CAN パケットのみ取り扱う
SAMPLE4	割り込みを使用する送受信プログラム データフレームとリモートフレームを取り扱う **CANFD パケットを取り扱う** ※SAMPLE4 は CANFD 対応マイコンのみ
SAMPLE_MONITOR	モニタプログラム 既存の CAN バスに流れているデータをモニタしたり、CAN バスにデータを送信するプログラム 割り込みを使用 データフレームとリモートフレームを取り扱う CAN パケット及び CANFD 対応マイコンでは CANFD パケットを取り扱う

※**太字**は前の SAMPLE***n*** に対し、新たに取り扱う項目

CANFD 対応マイコンでは 5 種類、CANFD 非対応のマイコンでは SAMPLE4 を除く 4 種類のプログラムが用意されています。

SAMPLE_MONITOR は、CAN バスのモニタ用プログラムです。自作のプログラムで思った様なデータが出力されているか等を確認する用途向けです。

settings カテゴリ(CS+), settings フォルダ(e2studio)内の、

program_select.h

```
/*---------------------------------------------------------------------
定数定義
---------------------------------------------------------------------*/

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
//#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム，CANFD版 ※CAN-FD対応マイコンのみ
//#define SAMPLE4

//MONITOR
//CANパケット送受信のモニタ用プログラム
//#define SAMPLE_MONITOR

//ーーここまで プログラム選択
```

の#define 文を 1 つのみコメントアウトが外れた状態に設定してください。

```
void main(void)
{

#ifdef SAMPLE1
    main_s1();
#endif

#ifdef SAMPLE2
    main_s2();
#endif

#ifdef SAMPLE3
    main_s3();
#endif

#ifdef SAMPLE4
    main_s4();
#endif

#ifdef SAMPLE_MONITOR
    main_monitor();
#endif

}
```

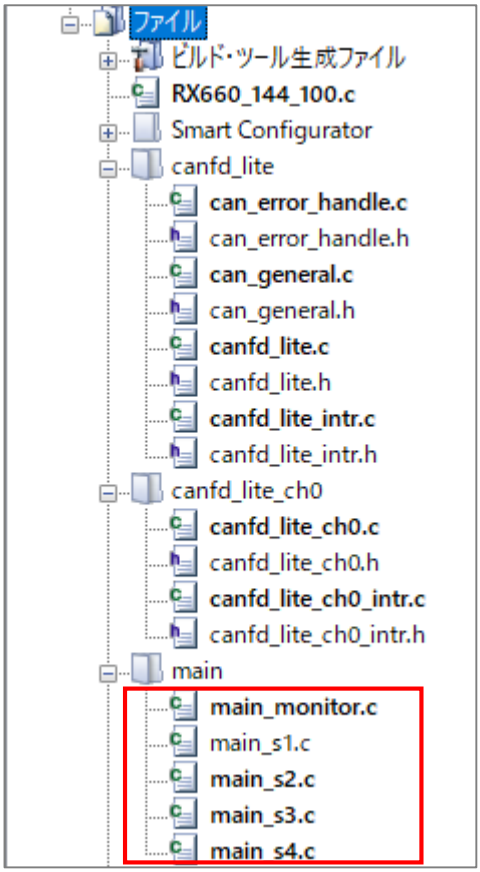
メイン関数は上記の様になっており、

main_s1()

main_s2()

main_s3()

main_s4()
main_monitor()
のいずれかが実行されるようになっています。



main_s1() ~ main_monitor()は、main のカテゴリ（フォルダ）以下に、ソースの実体があります。

5.2. クロック設定

サンプルプログラムの初期化（クロックの設定等）は、

－RX－
スマートコンフィグレータで生成されたコード
－RA－
FSP で生成されたコード

を使っています（初期化のコードを書き下してはいません）。

RX/RA マイコンでは、リセット後低速オンチップオシレータ(LOCO)，中速オンチップオシレータ(MOCO)が選択されていますので、
・メインクロックの発振
・PLL の起動（PLL 搭載マイコンのみ）
・クロックの切り替え
・RTC クロックの起動（RTC 搭載ボードのみ） ※CAN 動作には使用しません
を行い、マイコンが本来の速度（最大動作周波数）で動作するよう設定します。

クロック設定はマイコンボード毎に異なります。

－RX－

対象マイコンボード	搭載 水晶振動子 (MainOSC)	PLL 逓倍	CPU クロック	周辺クロック (PCLKB)	CAN クロック
HSBRX71M176 HSBRX72M176 HSBRX72N176 HSBRX71M100 HSBRX72N144 HSBRX72N100	24MHz	24×**10** (=240MHz)	240MHz	60MHz	PCLKB =60MHz
HSBRX64MC HSBRX65N176 HSBRX651F176 HSBRX65N144A HSBRX65N100A HSBRX651F144A HSBRX651F100A HSBRX65N144 HSBRX65N100 HSBRX651F144 HSBRX651F100 HSBRX66N176 HSBRX66N144 HSBRX66N100	24MHz	24×**10** (=240MHz)	120MHz	60MHz	PCLKB =60MHz
HSBRX660-144H HSBRX660-100B	24MHz	24×**10** (=240MHz)	120MHz	60MHz	CANFDCLK =60MHz

HSBRX671F144 HSBRX671F100	24MHz	24×**10** (=240MHz)	120MHz	60MHz	PCLKB =60MHz
HSBRX72T144	24MHz	24/3×**25** (=200MHz)	200MHz	50MHz	PCLKB =50MHz
HSBRX66T144	24MHz	24/3×**20** (=160MHz)	160MHz	40MHz	PCLKB =40MHz
HSBRX66T100A HSBRX66T100B	20MHz	20/2×**16** (=160MHz)	160MHz	40MHz	PCLKB =40MHz
HSBRX231F100 SmartRX!!!	8MHz	8/2×**13.5** (=54MHz)	54MHz	27MHz	MainOSC =8MHz
HSBRX23E-B100	8MHz	8/2×**8** (=32MHz)	32MHz	32MHz	PCLKB/2 =16MHz
HSBRX24U144 HSBRX24U100 HSBRX24T100B	10MHz	10×**8** (=80MHz)	80MHz	40MHz	PCLKB/2 =20MHz
HSBRX261-100	8MHz	8×**8** (=64MHz)	64MHz	32MHz	CANFDCLK =32MHz
HSBRX26T100	24MHz	24×**10** (=240MHz)	120MHz	60MHz	CANFDCLK =60MHz
HSBRX140F80	8MHz	8/2×**12** (=48MHz)	48MHz	24MHz	PCLKB/2 =12MHz

－RA－

対象マイコンボード	搭載 水晶振動子 (MainOSC)	PLL 逓倍	CPU クロック	周辺クロック (PCLKB)	CAN クロック
HSBRA8E1F144	24MHz	24/2×**60** (=720MHz)	360MHz	45MHz	CANFDCLK =80MHz
HSBRA8M1F176 HSBRA8T1F176	24MHz	24/3×**100** (=800MHz)	400MHz	50MHz	CANFDCLK =80MHz
HSBRA6M5F176	24MHz	24/3×**25** (=200MHz)	200MHz	50MHz	CANFDCLK =40MHz
HSBRA6M3F176 HSBRA6M2F144 HSBRA6M1F100 HSBRA6T1F100	24MHz	24/2×**20** (=240MHz)	120MHz	60MHz	PCLKB =60MHz
HSBRA6E1F100 HSBRA6M4F144	24MHz	24/3×**25** (=200MHz)	200MHz	50MHz	PCLKB =50MHz
HSBRA6E2F64 HSBRA6T3F64	24MHz	24/3×**20** (=160MHz)	160MHz(*1)	40MHz	CANFDCLK =40MHz
HSBRA6T2F100	24MHz	24×**10** (=240MHz)	240MHz	60MHz	CANFDCLK =40MHz
HSBRA4E1F64 HSBRA4M3F144 HSBRA4M2F100	24MHz	24/3×**25** (=200MHz)	100MHz	50MHz	PCLKB =50MHz
HSBRA4E2F64 HSBRA4T1F64	24MHz	24/3×**20** (=160MHz)	80MHz(*1)	40MHz	CANFDCLK =40MHz
HSBRA4M1F100	8MHz	8/2×**12** (=48MHz)	48MHz	24MHz	PCLKB =24MHz
HSBRA2A1F64	20MHz	-	48MHz	24MHz	MainOSC =20MHz
HSBRA2L1F100 HSBRA2L1F64	16MHz	-	48MHz	24MHz	MainOSC =16MHz

※PLL 逓倍の項の太字は逓倍率設定値

(*1)CANFDCLK を優先させるため最大動作周波数で駆動する設定ではありません

5.3. 初期化の流れ

CAN モジュールを初期化して、CAN 通信可能となるまでの流れを説明します。初期化部分は、搭載 CAN モジュール種別毎に多少異なります。下記表に示しているのは、実行する関数名です。

本節では大まかな流れのみ説明します。細かな点は、モジュール毎のソフトウェアマニュアルで解説します。

CANFD	CANFD_2	CANFD_B CANFD_Lite RSCAN	CAN	備考
can_reset	can0_reset can1_reset	can_reset		(*1)
can_init can0_init can1_init	can_init can0_init can1_init	can_init can0_init	can0_init can1_init can2_init	(*2)
can_receive_buf_conf	can0_receive_buf_conf can1_receive_buf_conf	can_receive_buf_conf		(*3)
			can0_mbox_set_send can1_mbox_set_send can2_mbox_set_send	(*4)
can0_receive_rule_set can1_receive_rule_set	can0_receive_rule_set can1_receive_rule_set	can0_receive_rule_set	can0_mbox_set_receive can1_mbox_set_receive can2_mbox_set_receive	(*5)
can_operate can0_operate can1_operate	can0_operate can1_operate	can_operate can0_operate	can0_operate can1_operate can2_operate	(*6)

例えば CANFD_Lite モジュールの初期化のコード例は以下となります。

```
//CAN-FDリセット
can_reset();

//初期化
can_init();
can0_init();

//受信バッファ設定
can_receive_buf_conf();

//受信ルール設定
// 受信ルール番号, バッファモード, IDフォーマット, データ/リモートフレーム, ID
can0_receive_rule_set(0, CAN_RULE_RXBUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);

//動作モードに移行
can_operate();
can0_operate();
```

can0/can1/can2 と数字の付いているのは、ch 依存の処理です。当該 ch を使用しない場合は実行の必要はありません。

(*1)CAN モジュールのリセット

CANFD, RSCAN モジュール系で必要です。

CANFD_2 モジュールでは、ch 毎に分かれています

(*2)CAN の初期化

CAN モジュールは、ch 毎に独立しているので、使用する ch の can***n***_init を実行してください。CANFD, RSCAN 系モジュールは、can_init の後、使用する ch の can***n***_init を実行してください。

(*3)受信バッファ数の設定

CANFD, RSCAN モジュール系で必要です。

CANFD_2 モジュールでは、ch 毎に分かれています。

(*4)送信メールボックスの設定

CAN モジュール系で、送信メールボックスを使用して送信する際に必要です。CAN モジュール系で FIFO を使用して送信する際は本関数の実行は不要です。

(*5)受信ルールの設定

→5.5 節で概要を説明

受信する ID や IDE, RTR の設定、受信先などを指定します。CAN のモジュール種別毎に設定項目等が異なりますので、詳細はモジュール毎の説明書で解説します。

CAN モジュールで、FIFO を使用して受信する場合は、本関数の実行は不要です。

can0_mbox_set_receive では、メールボックス毎の受信ルール設定を行います。

CANFD, RSCAN 系の can0_receive_rule_set では受信先の設定も行います。

(*6)動作開始設定

CANFD_2, CAN モジュールでは、ch 毎の関数。CANFD, CANFD_B, CANFD_Lite, RSCAN モジュールでは、can_operate の後で ch 毎の can***n***_operate を実行してください。

(*6)の後で、受信可能となります。

5.4. CAN の送信

CAN のデータ送信は、以下の関数で行います。

CAN モジュール種別		備考
CAN	can0_mbox_send can1_mbox_send can2_mbox_send	メールボックスを使用して送信
CAN	can0_mboxfifo_send can1_mboxfifo_send can2_mboxfifo_send	メールボックス FIFO を使用して送信
CANFD CANFD_2 CANFD_B CANFD_LIte RSCAN	can0_txbuf_send can1_txbuf_send	送信メッセージバッファを使用した送信
CANFD CANFD_2 CANFD_B CANFD_LIte	can0_txqueue_send can1_txqueue_send	TX キューを使用した送信
CANFD CANFD_2 CANFD_B CANFD_LIte	can0_cfifo_send can1_cfifo_send	共通 FIFO を使用した送信
RSCAN	can0_srfifo_send	送受信 FIFO を使用した送信

例えば CANFD_Lite モジュールの送信バッファで送信するコード例は以下となります。

```
can_message s_msg;
s_msg.id = 0x00000000;         //CANのID
s_msg.rtr = CAN_DATA_FRAME;    //データフレームを送信
s_msg.ide = CAN_ID_FORMAT_EID;  //拡張IDを使用
s_msg.dlc = 1; //送信データ1バイト
s_buf = 0;      //送信バッファ0を使用

can0_txbuf_send(s_buf, &s_msg);
```

送信では、CAN モジュール系ではメールボックスか FIFO を使用して送信することが出来ます。CANFD 系モジュールでは、送信バッファ、TX キュー、共通 FIFO を使用しての送信、RSCAN では送信バッファ、送受信 FIFO を使用しての送信が可能です。

CANFD のデータ送信は、以下の関数で行います。

CAN モジュール種別	データ送信関数	備考
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd0_txbuf_send canfd1_txbuf_send	送信メッセージバッファを使用した送信
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd0_txqueue_send canfd1_txqueue_send	TX キューを使用した送信

CAN モジュール種別	データ送信関数	備考
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd0_cfifo_send canfd1_cfifo_send	共通 FIFO を使用した送信

関数名が can0~から canfd0~の様に変わります。canfd と付いているのが、CANFD パケットを扱う送信関数です。

5.5. CAN の受信ルール設定

CAN では、予め設定した受信ルールにマッチしたメッセージが、CAN モジュール内のメールボックス、受信バッファや FIFO に格納されます。メールボックス等に格納されているメッセージを取り出す関数が、5.6 で説明している受信関数です。

受信ルールにマッチしないメッセージは、メールボックス等に保存される事なく捨てられますので、予め受信したいメッセージの受信ルール設定が必要となります。

(1)CAN モジュールでのメールボックスの受信ルール設定

```
can0_mbox_set_receive(0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
```

メールボックス 0（第 1 引数）の受信ルールを設定します。
受信対象とするのは
・拡張 ID(29bit)のメッセージ（第 2 引数: CAN_ID_FORMAT_EID）
・データフレーム（第 3 引数: CAN_DATA_FRAME）
・ID=0x00000000（第 4 引数）
とします。「拡張 ID フォーマット」で、「データフレーム」の「ID=0x00000000」のデータを受信した際に、メールボックス 0 にメッセージが格納されます（格納されたメッセージの読み出し処理は、5.6 節で説明。）
DLC（データバイト数）は任意の値のデータを受信します。（CAN の機能としては、受信データの DLC に制約を掛ける事も可能です。本サンプルプログラムでは、DLC は任意の値を受信するように設定しています。）

メールボックスは、32 個あります（※FIFO を使用した際は 24 個）。最大 32 個の受信ルールの設定が可能です。

can0 の部分は、CAN-ch1 の場合は can1 に変ります（ch 毎に設定は独立です）。

can***n***_mbox_set_receive() では、最大 32 個のルールまで設定可能ですが、
・任意の ID を受信対象としたい
・ID=0x00000000 ~ 0x00000FFF の範囲を
という場合は、

```
can0_mbox_set_receive_mask(0, 0x00000000);
```
→メールボックス 0~3 に対して、全ての ID を受信する設定とする

```
can0_mbox_set_receive_mask(4, 0xFFFFF000);
```
→メールボックス 4~7 に対して、末尾 3 バイトが任意の値の ID を受信する

上記のマスク設定関数と組み合わせる事で、受信範囲を広げる事が出来ます。
（マスク設定は、4 つのメールボックス毎の設定となります。メールボックス 0~3 で一種類のマスク設定。4~7, 8~11, ...も同様）

(2)CAN モジュールでの FIFO 受信ルール設定

CAN モジュール系の受信 FIFO は、
・拡張 ID, データフレーム, ID は任意
・拡張 ID, リモートフレーム, ID は任意
を受信する様に設定しています。受信ルール設定コマンドは不要です。
※標準 ID のメッセージを受信したい場合は、
・can_operation.h 内で標準 ID のみ取り扱う様に設定する
#define CAN_ID_TYPE CAN_ID_FORMAT_SID
・can_ch***n***.c 内の FIFO マスクレジスタの設定を変更する
などの方法があります。

(3)CANFD 系、RSCAN モジュールでの受信ルール設定

can0_receive_rule_set(0, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);

ルール番号 0（第 1 引数）の受信ルールを設定します。
メッセージの格納先は、CAN_RULE_RXFIFO0（第 2 引数: RXFIFO(0)）とします。
受信対象とするのは
・拡張 ID(29bit)のメッセージ（第 3 引数: CAN_ID_FORMAT_EID）
・データフレーム（第 4 引数: CAN_DATA_FRAME）
・ID=0x00000000（第 5 引数）
とします。「拡張 ID フォーマット」で、「データフレーム」の「ID=0x00000000」のデータを受信した際に、RXFIFO(0)にメッセージが格納されます（格納されたメッセージの読み出し処理は、5.6 節で説明。）
DLC（データバイト数）は任意の値のデータを受信します。（CAN の機能としては、受信データの DLC に制約を掛ける事も可能です。本サンプルプログラムでは、DLC は任意の値を受信するように設定しています。）

メッセージの格納先は、RXFIFO, CFIFO, SRFIFO, RXBUF（受信バッファ）などの指定が可能です。

設定可能ルール数は、CAN モジュール種によって変ります。受信ルール設定時の注意点として、ルール番号に隙間が出来ないように設定してください。（0, 1, 2, 4, 5 の受信ルールを設定した場合は、0, 1, 2 が有効となり、4, 5 は、無効です。0 番から隙間が生じないようにルールを設定してください。）

設定可能ルール数には、限りがありますので、1 つのルールで

・任意の ID を受信したい
can0_receive_rule_mask_set(0, 1, 1, 0x00000000);
→ルール番号 0 の受信ルールで任意の ID を受信する様に設定する
※can***n***_receive_rule_set()関数とセットで使用

第 1 引数: ルール番号

第 2 引数: 標準 ID, 拡張 ID 区分マスク(0:どちらのデータも受信する, 1:can***n***_receive_rule_set()関数の設定を引き継ぐ)

第 3 引数: データフレーム, リモートフレーム区分マスク(0:どちらのデータも受信する, 1:can***n***_receive_rule_set()関数の設定を引き継ぐ)

第 4 引数: ID マスク(0b0: 任意の ID を受信する, 0b1: 1:can***n***_receive_rule_set()関数の設定を引き継ぐ)

・データフレームとリモートフレームを 1 つのルールで受信したい

can0_receive_rule_mask_set(0, 1, 0, 0xFFFFFFFF);

→ルール番号 0 の受信ルールでデータフレームとリモートフレームを受信する様に設定する

can***n***_receive_rule_mask_set()関数にて、ID や IDE(拡張, 標準 ID 区分), RTR(データフレーム, リモートフレーム区分)のマスク設定を行う事が可能です。

5.6. CAN の受信

CAN のデータ受信は以下の関数で行います。

CAN モジュール種別	データ受信関数	備考
CAN	can0_mbox_receive can1_mbox_receive can2_mbox_receive	メールボックスを使用して受信
CAN	can0_mboxfifo_receive can1_mboxfifo_receive can2_mboxfifo_receive	メールボックス FIFO を使用して受信
CANFD CANFD_2 CANFD_B CANFD_LIte RSCAN	can0_rxbuf_receive can1_rxbuf_receive	受信メッセージバッファを使用した受信
CANFD CANFD_2 CANFD_B CANFD_LIte RSCAN	can_rxfifo_receive (*1) can0_rxfifo_receive (*2) can1_rxfifo_receive (*2)	受信 FIFO を使用した受信 (*1)CANFD モジュール (*2)CANFD_B, CANFD_2, CANFD_Lite モジュール
CANFD CANFD_2 CANFD_B CANFD_LIte	can_cfifo_receive (*1) can0_cfifo_receive (*2) can1_cfifo_receive (*2)	共通 FIFO を使用した受信 (*1)CANFD モジュール (*2)CANFD_B, CANFD_2, CANFD_Lite モジュール
RSCAN	can0_srfifo_receive	送受信 FIFO を使用した受信

例えば CANFD_Lite モジュールの受信バッファで受信するコード例は以下となります。

```
int r_ret;
can_message r_msg;
char *id_format[2];
id_format[0] = "SID";
id_format[1] = "EID";

r_ret = can0_rxbuf_receive(0, &r_msg); //受信バッファ0のデータを受信

if (r_ret != CAN_RET_NODATA)  //受信バッファにデータが格納されている場合
{
  sci_write_str("CAN data received,");
  sci_write_str(" id_type=");
  sci_write_str(id_format[r_msg.ide]);
  sci_write_str(" id=0x");
  sci_write_uint32_hex(r_msg.id);
  sci_write_str(" rtr=0x");
  sci_write_uint8_hex(r_msg.rtr);
  sci_write_str(" dlc=");
  sci_write_uint8(r_msg.dlc);
  sci_write_str(" data=0x");
  for (int i=0; i<(r_ret & 0xf); i++) sci_write_uint8_hex(r_msg.data[i]);
  sci_write_str(" ts=0x");
  sci_write_uint16_hex(r_msg.ts);
  sci_write_str("¥n");
}
```

CAN は、複数の受信方式があり、受信ルール設定時に設定した受信先にメッセージが格納されます。格納されたメッセージを本関数にて、指定した変数にコピーします。

CANFD のデータ受信は以下の関数で行います。

CAN モジュール種別	データ受信関数	備考
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd0_rxbuf_receive canfd1_rxbuf_receive	受信メッセージバッファを使用した受信
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd_rxfifo_receive (*1) canfd0_rxfifo_receive (*2) canfd1_rxfifo_receive (*2)	受信 FIFO を使用した受信 (*1)CANFD モジュール (*2)CANFD_B, CANFD_2, CANFD_Lite モジュール
CANFD CANFD_2 CANFD_B CANFD_LIte	canfd_cfifo_receive (*1) canfd0_cfifo_receive (*2) canfd1_cfifo_receive (*2)	共通 FIFO を使用した受信 (*1)CANFD モジュール (*2)CANFD_B, CANFD_2, CANFD_Lite モジュール

関数名が、can0~から canfd0 に変わります。

5.7. CAN メッセージ構造体に関して

本サンプルプログラムでは、CAN のメッセージを構造体で取り扱っています。

構造体定義

```
//CANメッセージ構造体
typedef struct{
    unsigned long id;
    unsigned char rtr;
    unsigned char ide;
    unsigned char dlc;
    unsigned char data[8];
    unsigned short ts;
} can_message;

//CANFDメッセージ構造体
typedef struct{
    unsigned long id;
    unsigned char rtr;
    unsigned char ide;
    unsigned char fdf;
    unsigned char brs;
    unsigned char esi;
    unsigned char dlc;
    unsigned char data[64];
    unsigned short ts;
} canfd_message;
```

型	メンバ	備考
unsigned long	id	CAN-ID 標準 ID では 0~0x7FF(11bit) 拡張 ID では 0~0x1FFFFFFF(29bit)
unsigned char	rtr	CAN_DATA_FRAME(0) （データフレーム） CAN_REMOTE_FRAME(1) （リモートフレーム） のいずれか
unsigned char	ide	CAN_ID_FORMAT_SID(0) （標準 ID） CAN_ID_FORMAT_EID(0) （拡張 ID） のいずれか
unsigned char	dlc	データバイト数 1~8 : 1~8 バイト 9: 12 バイト 10: 16 バイト 11: 20 バイト 12: 24 バイト 13: 32 バイト 14: 48 バイト 15: 64 バイト （9~15 は基本的には CANFD フレームで使用）
unsigned char	data[]	データ CAN フレームでは最大 8 バイト CANFD フレームでは最大 64 バイト

型	メンバ	備考
unsigned short	ts	タイムスタンプ 受信側で受信時のタイミングに応じて付与する 2 バイトのデータ
unsigned char	fdf	CAN_DATA_FORMAT(0) （CAN フレーム） CANFD_DATA_FORMAT(1) （CANFD フレーム） のいずれか
unsigned char	brs	CAN_BITRATE(0) （ビットレート固定） CANFD_BITRATE(1) （データ部分のビットレート可変） のいずれか
unsigned char	esi	CANFD_ERROR_ACTIVE(0) （エラーアクティブ） CANFD_ERROR_PASSIVE(1) （エラーパッシブ） のいずれか

CAN データを取り扱う送受信関数等では、本構造体を引数として指定する仕様としています。

5.8. 割り込みに関して

SAMPLE2 以降のプログラムでは、「エラー」「受信」「送信」の 3 種類の割り込みを有効化しています。

・エラー発生時の割り込み
エラー内容の表示
エラー履歴の変数への格納

・データ受信時の割り込み
受信リングバッファへのデータコピー
受信データの表示
受信データがリモートフレームの場合リモートフレーム応答データの送信

・データ送信後の割り込み
送信済みであることの表示

割り込み処理は、
xxx_intr.c
内に記載されています。（xxx はモジュール種別や CAN-ch を示す文字列）

割り込み関数の最後に、コールバック関数を呼ぶように設定しており、コールバック関数はメイン関数
main_s2.c
main_s3.c
main_s4.c
main_monitor.c
内に記載してあります。
（xxx_intr.c 内には、SAMPLEn に依存しない処理、SANPLEn に依存する処理は、main_xxx.c 内に記載。）

※詳細は各モジュール毎のソフトウェア編マニュアルで説明

5.9. 受信リングバッファに関して

SAMPLE1 では、受信リングバッファは未使用です。SAMPLE1 以外では、データ受信時に受信割り込みルーチンにて、受信データを受信リングバッファにコピーする仕様としています。

受信リングバッファの定義

```
#define CAN_CH 2                //CAN-ch0, CAN-ch1を取り扱う場合
#define CAN_RECV_BUF_SIZE 16    //リングバッファのサイズ

#ifdef CANFD_MODE
canfd_message g_can_recv_buf[CAN_CH][CAN_RECV_BUF_SIZE] = {0}; //CANFDパケット取り扱い時
#else
can_message g_can_recv_buf[CAN_CH][CAN_RECV_BUF_SIZE] = {0};  //CANパケットのみ取り扱い時
#endif
```

受信リングバッファにデータを格納するコード例

（割り込み関数内で、このような処理を行っています）

```
int ret;
can_message msg;
int ch = 0; //CAN-ch0の処理

ret = can0_rxfifo_receive(0, &msg); //RXFIFOの受信処理

if (ret > 0)//can0_rxfifo_receiveがエラーではない場合

//受信リングバッファ書き込みインデックスを進める
g_can_recv_buf_index1[can_ch]++;
if (g_can_recv_buf_index1[can_ch] >= CAN_RECV_BUF_SIZE) g_can_recv_buf_index1[can_ch] = 0;

//受信リングバッファに格納
g_can_recv_buf[can_ch][g_can_recv_buf_index1[can_ch]] = msg;
```

受信リングバッファからデータを読み出すコード例

（main_s2.c 内の can_interrupt_rx_callback など）

```
int ret;
can_message msg;
int ch = 0; //CAN-ch0の処理

while(1)
{
  ret = can_read_data(ch, &msg);  //受信リングバッファから読み出しを行う関数
  if (ret != CAN_RET_NODATA)
  {
    //リングバッファ内に受信データあり
    can_receive_data_print(ch, &msg);  //受信データがある場合は、表示関数にデータを渡す
  }
  else
  {
    //受信データなし
    break;
  }
}
```

上記の様な処理を行うと、受信リングバッファが空になるまで、受信リングバッファからの読み出しを行います。

※受信リングバッファは、
#define CAN_RECV_BUF_SIZE 16
サイズが 16 の構造体配列として定義していますが、格納可能なのは 15 メッセージまでとなります。
（インデックスの取り扱いを工夫する事により、16 メッセージの取り扱いも可能ですが、処理が煩雑となるため、利用可能なサイズが 1 減るのは許容しています。16 を変えると、サイズの変更は可能です。）
（上記定数は、can_operation.h 内で定義）

6. サンプルプログラムの動作説明

サンプルプログラムで、CAN のモジュールに依存する部分は、「CAN モジュール編」「RSCAN モジュール編」「CANFD(CANFD_B, CANFD_2, CANFD_Lite)モジュール編」の別冊のマニュアルで説明しています。使用する CAN のモジュールに合わせて、当該マニュアルを参照してください。

6.1. サンプルプログラム(SAMPLE1)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示

を行うサンプルプログラムとなっています。

プログラムをマイコンボードに書き込み起動すると、端末に、下記の表示が出力されます（マイコンボードによっては多少表示が異なります）。

マイコンボード(1) HSBRX72M176

マイコンボード(2) HSBRX24U-144

の場合の表示例を以下に示します。

```
HSBRX72M CAN Starter kit program boot.
Copyright (C) 2018-2024 HokutoDenshi. All Rights Reserved.

SAMPLE1: CAN Data frame send/receive simple program.

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

Command Usage:以下の部分が、本ボードに対するコマンドです。端末から PC のキーボードでコマンドを入力すると、入力されたコマンドに従い、動作を行います。

```
HSBRX24U CAN Starter kit program boot.
Copyright (C) 2021-2024 HokutoDenshi. All Rights Reserved.

SAMPLE1: CAN Data frame send/receive simple program.

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])
```

2 台のマイコンボードに、SAMPLE1 のプログラムを書き込んで起動すると、接続先の PC の端末には上記メッセージが表示されます。(Teraterm 等の端末を 2 つ開いてください)

本サンプルプログラムでは、端末のキーボードから、数字の 0~3 を押すと CAN のデータフレームを CAN バスに送出する様になっています。

マイコンボード(1)の端末で、"0"を押した場合、マイコンボード(1)がデータとして 0x01(1 バイト)を送信します。

```
CAN0 data send,     MB=0x00 ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

マイコンボード(2)では、送信とほぼ同時にデータを受信します。

```
CAN0 data received, buf=04 ret=1 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
ts=0x51C5
```

マイコンボード(2)の端末で、"1"を押した場合、マイコンボード(2)がデータとして 0x01 0x23(2 バイト)を送信します。

```
CAN0 data send,     buf=01 ret=0 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123
```

このとき、マイコンボード(1)側ではデータを受信します。

```
CAN0 data received, MB=0x09 ret=2 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123
ts=0x9EBD
```

端末では以下の情報を表示します。

MB=　使用メールボックス番号(CAN モジュール系)

buf=　使用バッファ番号(RSCAN, CANFD モジュール系)

ret=　送信時:エラーが無ければ 0, 受信時:受け取ったデータバイト数

id_type= ID タイプ(EID:拡張フォーマット, SID:標準フォーマット)

id=　CAN ID

rtr=　　データフレーム(0). リモートフレーム(1)区分
dlc=　　データバイト数(DLC)　※実際に送受信したバイト数ではなく CAN パケットに含まれる DLC 値
data=　送受信データ
ts=　　タイムスタンプ（受信側が回しているタイマカウンタ値）

タイムスタンプは、受信側でデータの受信タイミング（複数のデータを受信した際、時系列的にどの順番であったか）を判断するために付与しているデータとなります。（CAN の送信データの中にはタイムスタンプの情報は含まれません）

・0~3 コマンド

CAN メッセージの送信
を行います。

コマンド	ID	IDE	RTR	DLC	データ
0	0x0000000	1	0	1	0x01
1	0x0000001	1	0	2	0x01 0x23
2	0x0000002	1	0	4	0x01 0x23 0x45 0x67
3	0x0000003	1	0	8	0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF

ID はコマンド 0~3 に応じて変更。IDE は、標準では拡張 ID(29bit)なので 1 を送ります（後述の's'コマンドで 0:標準 ID(11bit)に変更可能です）。RTR は「データフレーム」なので 0 固定です。DLC（送信バイト数）は、コマンドに応じて変わります。データは、表に記載の値（固定値）です。

※評価用スイッチを押すと、コマンド 0 と同じ動作（CAN メッセージの送信）を行います
ボード毎の評価スイッチに関しては、
CAN_STARTER_KIT_RXRA_MCU_BOARD_REV_x_x_x_.pdf
を参照してください。

・z コマンド

z コマンドを入力すると、端末に接続されているマイコンボード上の LED が 3 回点滅します。

```
LED blink test(for board identify).
```

本キットでは、2 台以上のマイコンボードを取り扱うため、端末とマイコンボードの関係が判らなくなるケースもあります。その際、z コマンドで端末とマイコンボードの接続を確認してください。
（ボード上に LED が搭載されていないボードは、起動時のメッセージに z コマンドが表示されません。）

LED の位置や LED 使用に必要なジャンパ設定などは
CAN_STARTER_KIT_RXRA_MCU_BOARD_REV_x_x_x_.pdf
を参照してください。

・s コマンド

拡張フォーマットと標準フォーマットの切り替え

を行います。トグル動作となり、s コマンドを入力する度にフォーマットが拡張→標準→拡張と変化します。

拡張フォーマット（起動時のデフォルト）は、ID を 29bit で送り、標準フォーマットは ID を 11bit で送ります。また、CAN パケットに含まれる IDE(1bit)の値が

拡張フォーマット IDE=1

標準フォーマット IDE=0

変わります。

マイコンボード(1)の端末で、"s"を押した場合、端末に下記メッセージが表示され、フォーマットが切り替わった事が判ります。

```
send format -> SID
```

その後、"0"を押した場合、マイコンボード(1)が標準フォーマットでデータを送出します。

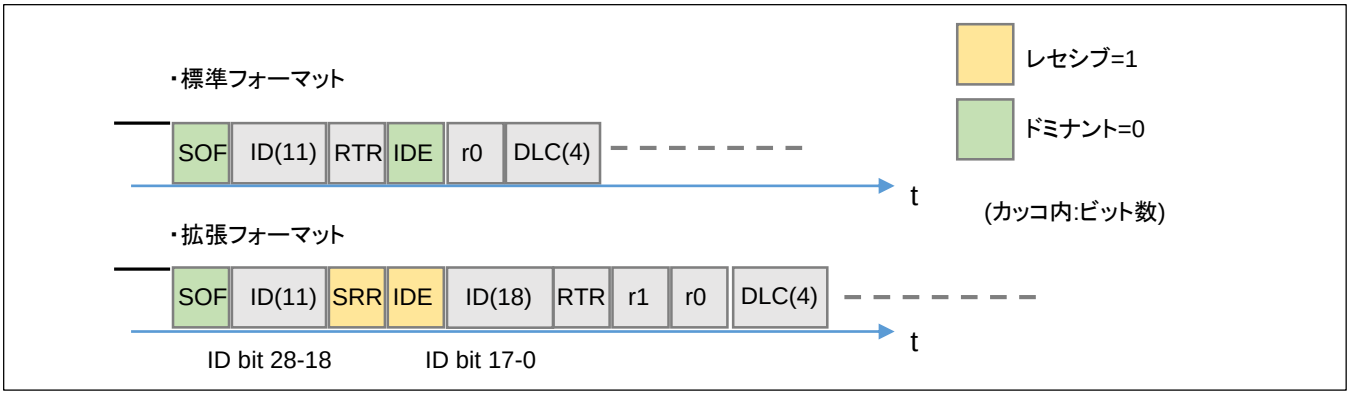
```
send format -> SID
CAN0 data send,     MB=0x00 ret=0 id_type=SID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

マイコンボード(2)では、データを受信します。

```
CAN0 data received, buf=00 ret=1 id_type=SID id=0x00000000 rtr=0x00 dlc=1 data=0x01
ts=0xBB5A
```

送信、受信側共、id_type の行が SID の表示になり、標準フォーマットでのデータ送受信である事が判ります。

ーフォーマットの違いー



ID=0x0000002 の場合

bit	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
標準フォーマット																			0	0	0	0	0	0	0	0	0	1	0
拡張フォーマット	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

前半で送信　　後半で送信

拡張フォーマットでは、ID が 2 分割で送信されます。IDE ビットの値も異なります。

・a コマンド

送信・送信待機となっている状態の停止

を行います。

1 コマンドでデータ送信

```
CAN0 data send,     buf=01 ret=0 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123
```

このとき、通信相手のボードが接続されていない、接続先のボードがリセット状態であったり、CAN 通信プログラムが実行されていない場合は、通信の ACK（通信相手がデータを受信した旨の応答）が返ってこない状態となり、データ送信コマンドを実行しているボードは、データの再送を繰り返します。（連続的にデータ送信を繰り返す状態となります。）

a コマンドを入力

```
send abort!
```

この状態で、a コマンドを使用するとデータ送信を繰り返す状態から、データ送信を止める事が出来ます。

通信相手が居ない状態で、a コマンドは以下の様に作用します。

```
CAN0 data send,     buf=01 ret=0 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123 (1)
CAN0 data send,     buf=01 ret=-3 (2)
send abort! (3)
CAN0 data send,     buf=01 ret=0 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123 (4)
```

(1) 1 コマンドを入力

→（通信相手が居ない場合）送信が繰り返される状態となる

(2) 1 コマンドを入力

→指定した送信バッファが使用中のためエラーコード-3 を返した

(3) a コマンドを入力

→送信操作の中止

(4) 1 コマンドを入力
→再度送信が可能となる（この時点では送信バッファが未使用のため）

・S コマンド
ステータスレジスタの表示
を行います。
※小文字 s と大文字 S は別なコマンドとなります

－CAN モジュール－

```
-----
CAN0 status register

Transmit error counter (TECR)    : 0
Receice error counter (RECR)     : 18

Error status flag (EST)          : 0 -> NO error detected
Error passive status flag (EPST) : 0 -> NOT error passive
Bus-off status flag (BOST)       : 0 -> NOT Bus-off

-----
CAN2 status register

Transmit error counter (TECR)    : 128
Receice error counter (RECR)     : 0

Error status flag (EST)          : 1 -> error detected
Error passive status flag (EPST) : 1 -> error passive
Bus-off status flag (BOST)       : 1 -> Bus-off
```

( )内はマイコンのレジスタの名称です。
複数 ch の CAN を有効化している場合は、ch 毎のステータスが表示されます。

－RSCAN モジュール－

```
-----
CAN0 status register

Transmit error counter (TEC)       : 246
Receice error counter (REC)        : 0

Communication status flag (COMSTS) : 1 -> communication ready
Receive status flag (RECSTS)       : 0 -> idle
Transmit status flag (TRMSTS)      : 0 -> idle
Bus-off status flag (BOSTS)        : 0 -> NOT bus-off
Error-passive status flag (EPSTS)  : 1 -> error-passive
Channel stop status flag (CSLPSTS) : 0 -> NOT channel stop mode
Channel halt status flag (CHLTSTS) : 0 -> NOT channel halt mode
Channel reset status flag (RSTST)  : 0 -> NOT channel reset mode
```

モジュール毎に、持っているステータスレジスタが異なりますので、表示内容は CAN モジュールのケースとは異なります。

－CANFD モジュール系－

```
-----
CAN0 status register

Transmit error counter (TEC)      : 12
Receice error counter (REC)       : 0

ESI status flag (RESI)           : 0 -> NOT received CANFD message with ESI flag
Communication status flag (CRDY)  : 1 -> communication ready
Receive status flag (RECST)       : 0 -> idle
Transmit status flag (TRMST)      : 0 -> idle
Bus-off status flag (BOST)        : 0 -> NOT bus-off
Error-passive status flag (EPST)  : 0 -> NOT error-passive
Channel sleep status flag (SLPST) : 0 -> NOT channel sleep mode
Channel halt status flag (HLTST)  : 0 -> NOT channel halt mode
Channel reset status flag (RSTST) : 0 -> NOT channel reset mode

-----
CANFD0 status register

Successful occurrence counter (SC) : 6
Error occurrence counter (EC)      : 0

Transmitter delay compensation violation flag (TDCV) : 0 -> NO violation is present
Successful occurrence counter overflow flag (SOCV)   : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCV)        : 0 -> NOT overflow
Transmitter delay compensation result status (TDCR)  : 0x00
```

CANFD 系では、RSCAN と似ていますが、CANFD の表示が増えている事などが異なります。
※上記表示例は RX マイコンの場合です、RA マイコンの場合はレジスタ名等が変わります

S コマンドは、現在の CAN モジュールのステータスレジスタを表示しますので、CAN のステータスを確認する場合に使用してください。

・E コマンド

エラーステータスレジスタの表示

を行います。

－CAN モジュール－

```
-----
CAN0 Error flag

Stuff error flag (SEF)                : 0 -> NO Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 0 -> NO Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 0 -> NO Bus error detected
Error warning flag (EWIF)             : 0 -> NO error-warning detected
Error passive flag (EPIF)             : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected


-----
CAN2 Error flag

Stuff error flag (SEF)                : 0 -> NO Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 1 -> Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 1 -> Bus error detected
Error warning flag (EWIF)             : 1 -> Error-warning detected
Error passive flag (EPIF)             : 1 -> Error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected
```

( )内はマイコンのレジスタの名称です。

どのようなエラーが生じているかの表示を行います。上記表示例では、CAN-ch2 側で ACK エラーが生じていますので、接続相手が居ない、2 台のボードで通信速度が合っていない等の原因が考えられます。

ーRSCAN モジュールー

```
-----
CAN error flag register

GERFLL = 0x00000000

Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow
is detected
FIFO message lost flag (MES)                  : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                         : 0 -> NO DLC error is detected

-----
CAN0 error flag register

C0ERFLL = 0x00002507

ACK delimiter error flag (ADERR)              : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)  : 1 -> dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                         : 0 -> NO CRC error is detected
ACK error flag (AERR)                         : 1 -> ACK error is detected
Form error flag (FERR)                        : 0 -> NO form error is detected
Stuff error flag (SERR)                       : 1 -> stuff error is detected
Arbitration-lost flag (ALF)                    : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                           : 0 -> NO bus-lock is detected
Overload flag (OVLF)                          : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                   : 0 -> NO bus-off recovery is detected
Bus-off entry flag (BOEF)                     : 0 -> NO bus-off entry is detected
Error-passive flag (EPF)                      : 1 -> error-passive is detected
Error-warning flag (EWF)                      : 1 -> error-warning is detected
Bus-error flag (BEF)                          : 1 -> bus-error detected
```

RSCAN モジュールでは、グローバル（ch 非依存）のエラーフラグレジスタと、ch 依存のエラーフラグレジスタがあります。

ーCANFD 系モジュールー

```
-----
CAN error flag register

GEESR = 0x00000000

CANFD payload overflow flag (PODF)           : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLDF) : 0 -> NO Transmit history buffer overflow
is detected
FIFO message lost flag (MLDF)                : 0 -> NO FIFO message lost is detected
DLC error flag (DEDF)                       : 0 -> NO DLC error is detected

-----
CAN0 error flag register

C0CHESR = 0x00000000

ACK delimiter error flag (ADEDF)              : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0EDF)  : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1EDF) : 0 -> NO recessive bit error is detected
CRC error flag (CEDF)                         : 0 -> NO CRC error is detected
ACK error flag (AEDF)                         : 0 -> NO ACK error is detected
Form error flag (FEDF)                        : 0 -> NO form error is detected
Stuff error flag (SEDF)                       : 0 -> NO stuff error is detected
Arbitration-lost flag (ALDF)                   : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLDF)                          : 0 -> NO bus-lock is detected
Overload flag (OLDF)                          : 0 -> NO overload is detected
Bus-off recovery flag (BORDF)                  : 0 -> NO bus-off recovery is detected
Bus-off entry flag (BOEDF)                     : 0 -> NO bus-off entry is detected
Error-passive flag (EPDF)                      : 0 -> NO error-passive is detected
Error-warning flag (EWDF)                      : 0 -> NO error-warning is detected
Bus-error flag (BEDF)                         : 0 -> NO bus-error is detected

-----
CANFD0 error flag register

C0FDSTS = 0x00000000

Transmitter delay compensation violation flag (TDCV) : 0 -> NO violation
Successful occurrence counter overflow flag (SCOV)   : 0 -> NOT overflow
Error occurrence counter overflow flag (ECOV)        : 0 -> NOT overflow
```

CANFD 系では、グローバル、ch 依存に加え CANFD のエラーフラグレジスタが追加で表示されます。

・H コマンド

エラーステータスレジスタの累積表示

を行います。

S コマンドで見られるのは、現在のステータスですが、H コマンドは過去に発生したエラー(*1)含め表示します。

(*1)エラー割り込みが有効な SAMPLE2 以降で有効

エラー割り込み発生時に、グローバル変数にエラー履歴を追加します

エラー割り込みが有効化されていない SAMPLE1 では、E コマンド実行時にエラー履歴を更新します

－CAN モジュール－

```
-----
CAN0 error flag register [error history]

Stuff error flag (SEF)                : 1 -> Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 0 -> NO Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 1 -> Bus error detected
Error warning flag (EWIF)             : 0 -> NO error-warning detected
Error passive flag (EPIF)             : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected

-----
CAN2 error flag register [error history]

Stuff error flag (SEF)                : 0 -> NO Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 1 -> Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 1 -> Bus error detected
Error warning flag (EWIF)             : 0 -> NO error-warning detected
Error passive flag (EPIF)             : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected
```

赤字部分が追加で表示される以外は、S コマンドの表示例と変わりません。

現時点で解消されているエラーに関しても（エラーが保存されていれば）表示されます。

※エラー履歴変数の更新はエラー割り込み発生時やエラー表示（E コマンド実行）時なので、必ずしも全てのエラーが保存されるわけではありません（エラー割り込み発生後に追加で発生したエラーなどは、保存対象から外れるケースがあります）

・C コマンド

エラーステータスのクリア

を行います。

ーCAN モジュールー

E コマンドでエラーの確認

```
-----
CAN0 Error flag

Stuff error flag (SEF)                : 0 -> NO Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 1 -> Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 1 -> Bus error detected
Error warning flag (EWIF)             : 1 -> Error-warning detected
Error passive flag (EPIF)             : 1 -> Error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected
```

C コマンドでエラークリア

```
CAN Error register / occurrence counter clear
```

再度 E コマンドでエラーの確認

```
-----
CAN0 Error flag

Stuff error flag (SEF)                : 0 -> NO Stuff error detected
Form error flag (FEF)                 : 0 -> NO Form error detected
Ack error flag (AEF)                  : 0 -> NO Ack error detected
CRC error flag (FEF)                  : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)        : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                 : 0 -> NO Bus error detected
Error warning flag (EWIF)             : 0 -> NO error-warning detected
Error passive flag (EPIF)             : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)            : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                   : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                  : 0 -> NO Bus-Lock detected
```

C コマンドでは、エラーレジスタのクリアを行います。

－CANFD 系モジュール－

S コマンドでステータスの確認

```
-----
CAN0 status register

Transmit error counter (TEC)      : 116
Receice error counter (REC)       : 0

ESI status flag (RESI)           : 0 -> NOT received CANFD message with ESI flag
Communication status flag (CRDY)  : 1 -> communication ready
Receive status flag (RECST)       : 0 -> idle
Transmit status flag (TRMST)      : 0 -> idle
Bus-off status flag (BOST)        : 0 -> NOT bus-off
Error-passive status flag (EPST)  : 0 -> NOT error-passive
Channel sleep status flag (SLPST) : 0 -> NOT channel sleep mode
Channel halt status flag (HLTST)  : 0 -> NOT channel halt mode
Channel reset status flag (RSTST) : 0 -> NOT channel reset mode

-----
CANFD0 status register

Successful occurrence counter (SC) : 12
Error occurrence counter (EC)      : 255

Transmitter delay compensation violation flag (TDCV) : 0 -> NO violation is present
Successful occurrence counter overflow flag (SOCV)   : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCV)        : 1 -> overflowed
Transmitter delay compensation result status (TDCR)  : 0x00
```

C コマンド入力後

```
-----
CANFD0 status register

Successful occurrence counter (SC) : 0
Error occurrence counter (EC)      : 0

Transmitter delay compensation violation flag (TDCV) : 0 -> NO violation is present
Successful occurrence counter overflow flag (SOCV)   : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCV)        : 0 -> NOT overflow
Transmitter delay compensation result status (TDCR)  : 0x00
```

CANFD 系モジュールでは、通信成功、通信エラーカウンタ、それらのオバーフローフラグもクリアします。

※（SAMPLE1 ではエラー割り込みが無効化されていますが、SAMPLE2 以降のサンプルプログラムで）C コマンドはエラー割り込み回数のカウンタもクリアします

※H コマンドで表示される履歴に関しては C コマンドでは初期化されません

・c コマンド

送信元 ch の切り替え

を行います。

※CAN が 2ch 以上有効になっているケースで使用可能なコマンド

CAN が 2ch 以上有効になっているボードでは、"c"コマンドで送信するチャンネルを変更する事ができます。

0 コマンドを入力

```
CAN0 data send,     MB=0x00 ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

→CAN-ch0 からデータが送信されます

c コマンドを入力

```
CAN send channel -> CAN2
```

→送信元の CAN-ch が切り替わります

0 コマンドを入力

```
CAN2 data send,     MB=0x00 ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

→CAN-ch2 からデータが送信されます

なお、受信は、全ての有効化されたチャネルで有効です。"c"は送信元を変更するコマンドです。

※RX72M(HSBRX72M176)では、ch0, ch2 がデフォルトで有効化されているので、"c"コマンドを入力する度に、ch0 と ch2 が切り替わります

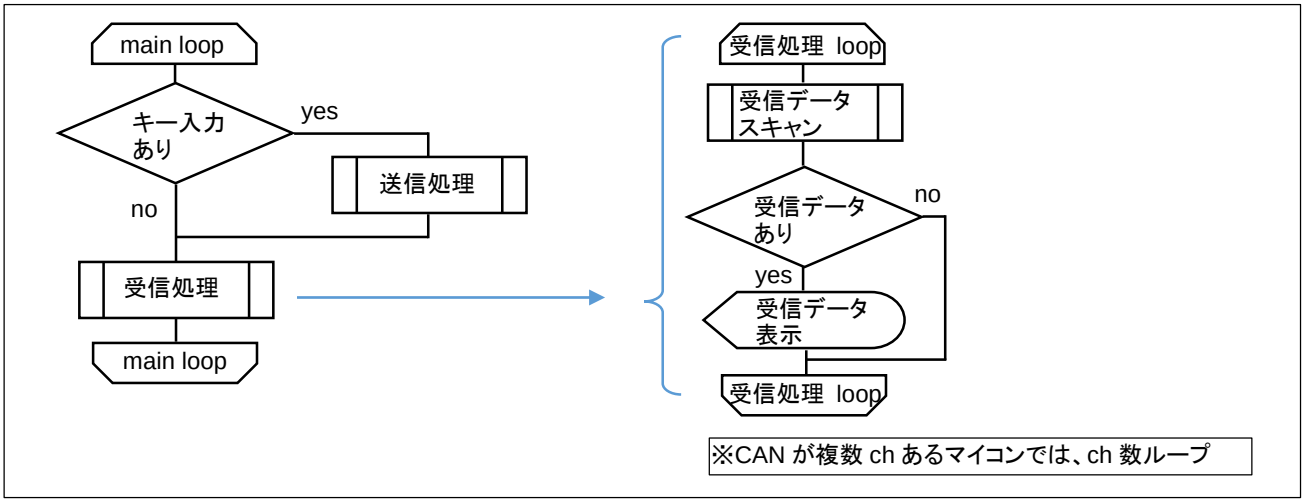
（起動後には、有効化されている ch の内一番若い番号の ch がコマンドによる送信対象 ch に設定されます）

※CAN-ch0 と CAN-ch2 を同一 CAN バス上に接続した場合は、CAN-ch2 で送信したデータを、CAN-ch0 が受信します

```
CAN2 data send,     MB=0x00 ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
CAN0 data received, MB=0x08 ret=1 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

※基本的にはボード 1 が送信したデータはボード 2 が受信しますが、CAN-ch0 と CAN-ch2 をケーブルで接続した場合には、CAN-ch2 が送信したデータを CAN-ch0 が受信します。（同一ボードの、別 ch の動作は独立しています。同一ボードの別 CAN-ch 間での通信も成立します。）

本サンプルプログラムでは、各種初期化を行った後はプログラムをループさせておりループ内で送信の処理と受信データの有無の確認を行っています。メインループの簡易フローチャートを以下に示します。



本サンプルプログラムでは、受信データの有無をポーリング処理によって確認しています。
・常に受信データの確認を行っている（受信データの確認にリソースが消費されている）
・送信等別な処理が入ると、受信データのスキャン頻度が変わる
といったデメリットがあります。

次のサンプルプログラム(SAMPLE2)では、割り込みを使用しており、受信と送信後の処理を割り込みで行っています。

6.2. サンプルプログラム(SAMPLE2)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示

を行うサンプルプログラムです。見た目上の動作としては、SAMPLE1 と変わりませんが、割り込みを使用して処理を行っている点が異なります。

マイコンボード(1) HSBRX72M176

マイコンボード(2) HSBRX24U-144

の場合の表示例を以下に示します。

```
HSBRX72M CAN Starter kit program boot.
Copyright (C) 2018-2024 HokutoDenshi. All Rights Reserved.

SAMPLE2: CAN Data frame send/receive program(with interrupt).

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

```
HSBRX24U CAN Starter kit program boot.
Copyright (C) 2021-2024 HokutoDenshi. All Rights Reserved.

SAMPLE2: CAN Data frame send/receive simple program(with interrupt, use FIFO).

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])
```

マイコンボード(1)の端末で、"2"を押した場合、マイコンボード(1)がデータとして 0x01 0x23 0x45 0x67（4 バイト）を送信します。

```
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
CAN0 send finished.(FIFO)
```

マイコンボード(2)では、送信とほぼ同時にデータを受信します。

```
CAN0 data received, id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123 ts=0x74A0
```

マイコンボード(2)の端末で、"3"を押した場合、マイコンボード(2)がデータとして 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF（8 バイト）を送信します。

```
CAN0 data send,    ret=0 id_type=EID id=0x00000003 rtr=0x00 dlc=8 data=0x0123456789ABCDEF
CAN0 send finished.(SRFIFO)
```

このとき、マイコンボード(1)側ではデータを受信します。

```
CAN0 data received, id_type=EID id=0x00000003 rtr=0x00 dlc=8 data=0x0123456789ABCDEF
ts=0x195E
```

見た目上の動作として、SAMPLE1 との相違点は
send finished
という表示が追加されている位です。

これは、送信が「完了」した事を示しています。送信完了＝送信先の相手が ACK を返したという事です。SAMPLE1 での送信動作は、送りっぱなし（送信が完了したか、ACK を返す相手が居なくて送信動作を繰り返しているかは表示されない）ですが、SAMPLE2 では送信完了時の割り込みにより、後処理（このプログラムでは画面表示ですが、実際のアプリケーションでは、送信完了後に別な処理を入れるといった使い方ができます）を入れています。

SAMPLE1 との相違点として、

CAN モジュールでは、
受信：メールボックス FIFO＋メールボックス
送信：メールボックス FIFO
を使用する様にしています。

RSCAN モジュールでは、
受信：受信 FIFO
送信：送受信 FIFO
としています。SAMPLE1 では、受信には受信バッファを使用していましたが、SAMPLE2 では FIFO を使用する様に変更しています。これは、割り込みの機能と FIFO が紐付けされているためです。

（※受信に受信バッファを使用する設定も可能ですが、その場合受信動作は割り込みではなくポーリングによる受信となります）

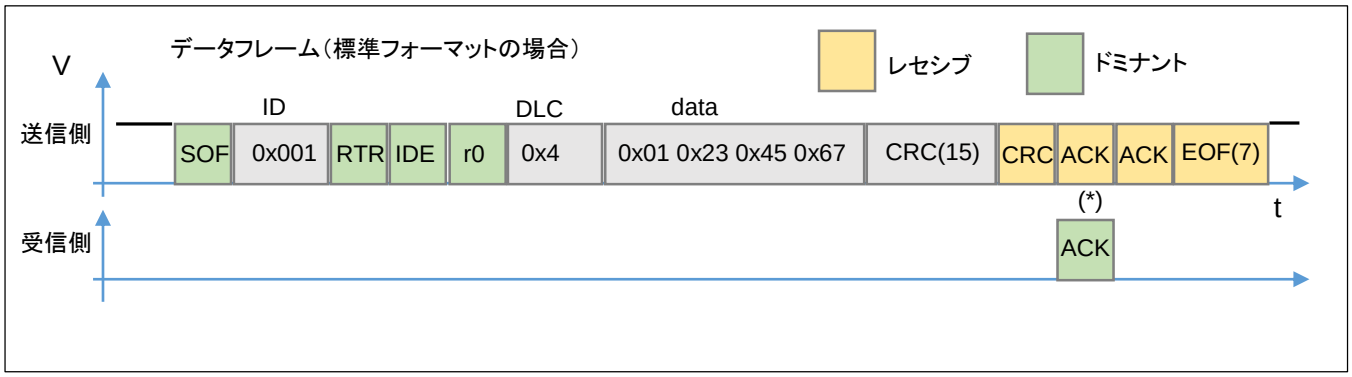
CANFD モジュールでは、
受信：受信 FIFO
送信：共通 FIFO
としています。RSCAN 同様受信バッファでの受信を選択すると、割り込みでの受信はできません。

CANFD_B, CANFD_2, CANFD-Lite モジュール系では、
受信：受信 FIFO
送信：共通 FIFO
としています。これらのモジュールは、受信バッファでの受信時、割り込みの使用はできます。

※受信、送信方式は、can_operation2.h で定義されています（変更可能です）

SAMPLE2 では、送信後のメッセージ(send finished)が表示されますが、対向機（通信相手）が居ない場合は表示されません。送信後のメッセージは、CAN でつながる送信元以外のボードが ACK を返した場合に表示されます。

送信側が 2 コマンドでデータを送出した場合、



CAN のデータを受信する通信相手が居る場合は、(*)のタイミングで受信側が 0（ドミナント)を送出します。このとき、送信側は、(*)のタイミングで 0（ドミナント）を受信した場合、自分が送出したデータが相手に伝わったと認識して、データ送信を完了します。(*)のタイミングが 1（レセシブ）の場合は、送信側は送信したデータが受信されなかったと認識して、データ送信を繰り返します。

SAMPLE2 では、エラー割り込みを有効化しているので、(*)のタイミングで相手からの ACK が返ってこない場合、ACK エラーとなり、エラー割り込みが掛かります。

〇通信相手が居ない場合（2 コマンドでデータ送信）

```
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
*****
- can0_interrupt_error_callback() -
Stuff error flag (SEF)                 : 0 -> NO Stuff error detected
Form error flag (FEF)                  : 0 -> NO Form error detected
Ack error flag (AEF)                   : 1 -> Ack error detected
CRC error flag (FEF)                   : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                 : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                 : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)         : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                  : 1 -> Bus error detected
Error warning flag (EWIF)              : 0 -> NO error-warning detected
Error passive flag (EPIF)              : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)             : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                    : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                   : 0 -> NO Bus-Lock detected
*****
```

※CAN モジュールの場合の表示例

SAMPLE1 では、ACK が返ってこない場合、端末には何も表示されず、データ再送を繰り返す動作となりますが、SAMPLE2 では、エラー割り込みが掛かり、エラー内容が表示されます。また、このエラー割り込み処理の中で送信を中断します。

エラー割り込みが掛かった際、エラー割り込みの回数をグローバル変数に保存しておき、累積 3 回エラー割り込みが掛かると、エラー割り込みを無効化します。

〇「通信相手が居ない状態でデータ送信コマンド（2 コマンド）を、3 回入力した場合

```
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
*****                                                          3 回目の送信コマンド
- can0_interrupt_error_callback() -
Stuff error flag (SEF)                 : 0 -> NO Stuff error detected
Form error flag (FEF)                  : 0 -> NO Form error detected
Ack error flag (AEF)                   : 1 -> Ack error detected
CRC error flag (FEF)                   : 0 -> NO CRC error detected
Bit1 error flag (BE1F)                 : 0 -> NO Bit1 error detected
Bit0 error flag (BE0F)                 : 0 -> NO Bit0 error detected
Ack delimiter error flag (ADEF)         : 0 -> NO Ack delimiter error detected
Bus error flag (BEIF)                  : 1 -> Bus error detected
Error warning flag (EWIF)              : 0 -> NO error-warning detected
Error passive flag (EPIF)              : 0 -> NO error-passive detected
Bus-off enter flag (BOEIF)             : 0 -> NOT Bus-off enter
Bus-off recovery flag (BORIF)          : 0 -> NOT Bus-off recover
Overrun flag (ORIF)                    : 0 -> NO overrun detected
Overload frame transmission flag (OLIF) : 0 -> NO overload frame transmission detected
Bus-Lock flag (BLIF)                   : 0 -> NO Bus-Lock detected
-----
So many errors occured. -> ch error interrupt disabled.
-----
*****
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
CAN0 data frame send,    ret=0 id_type=EID id=0x00000002 rtr=0x00 dlc=4 data=0x01234567
CAN0 data frame send,    ret=-4
                                                  4~8 回目の送信コマンド
```

エラー割り込みが 3 回掛かった時点で、エラー割り込みは無効化され、4 回目に送信コマンドを送った場合は、SAMPLE1 と同様、データ送信を繰り返す動作となります。

SAMPLE2 では、送信に FIFO を使用しているので、データ送信が完了しない状態でも、他のメッセージを送信する事が可能です。CAN モジュールの場合、FIFO は 4 段なので 4 つまでのメッセージを送信待機状態で溜めることができます。5 個目のメッセージで、-4（FIFO フル）のエラーを返します。

※FIFO の段数は、CAN モジュール種別で変わります

（この状態で"a"コマンドを入力すると、データ送信を停止します。FIFO に溜まっているキューもクリアされます。）

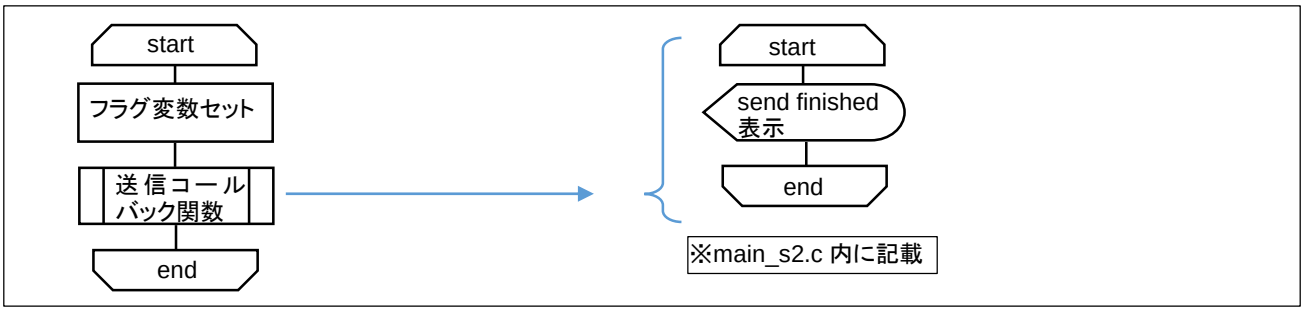
※エラーは累積 3 回でエラー割り込みを無効化していますが、"C"コマンドで累積回数をクリアする事が出来ます

※エラー割り込みを無効化する回数（3 回）は、can_operation.h 内で定義されています

－エラー割り込みを 3 回で無効化するようにしている理由－

自局の送信起因でエラー割り込みが生じた場合は、送信を停止するなどで、エラー割り込みの発生要因を取り除く事が出来ます。それに対し、通信相手が通信速度が合っていないデータを送信してきた場合など、自分側ではエラーの発生要因を取り除く事が出来ない場合、エラー割り込みの無限ループとなります。エラー割り込みの無限ループを回避するために、累積 3 回でエラー割り込みを無効化する様にしています。

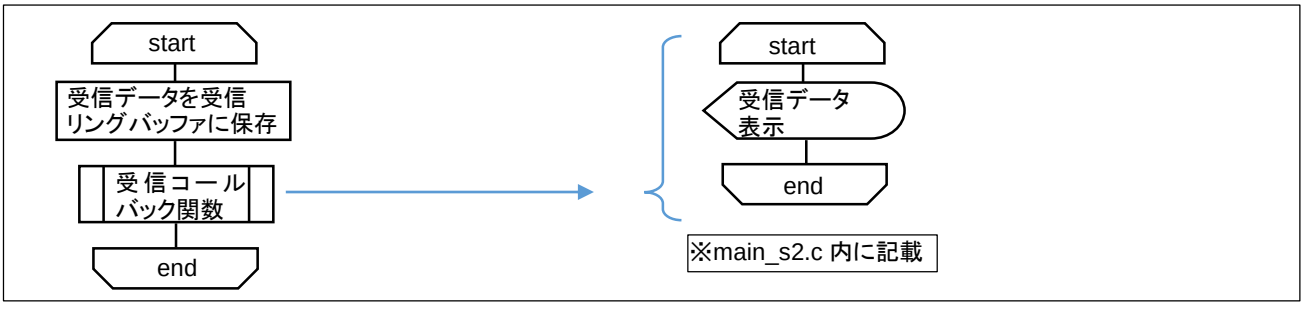
－送信割り込みの簡易フローチャート－



送信割り込みは、[module_name]_ch[can_ch]_intr.c 内に記載しており、コールバック関数は main_s2.c 内に記載しています。

SAMPLE2 では、データ受信時にも、割り込みを使用して受信データの処理を行っています。

－受信割り込みの簡易フローチャート－



受信割り込みは、[module_name]_ch[can_ch]_intr.c 内に記載しており、コールバック関数は main_s2.c 内に記載しています。

受信割り込み関数内では、受信データをユーザが用意した受信用のリングバッファに保存する処理を行っています。また、受信関数の最後にコールバック関数を呼び出す様にしており、コールバック関数内で受信データの表示処理を実行しています。SAMPLE2 では、受信のポーリング処理は無くなり、データを受信したタイミングで効率よく受信データを処理できます。

6.3. サンプルプログラム(SAMPLE3)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示
・端末からの指示でリモートフレームの送信
・リモートフレーム受信時にレスポンスを送信

を行うサンプルプログラムです。SAMPLE2 に対して、リモートフレームの送信と応答の機能を追加したものです。

マイコンボード(1) HSBRX72M176

マイコンボード(2) HSBRX24U144

の場合の表示例を以下に示します。

```
HSBRX72M CAN Starter kit program boot.
Copyright (C) 2018-2024 HokutoDenshi. All Rights Reserved.

SAMPLE3: CAN [Data frame] send/receive program(with interrupt).
            [Remote frame] request/response program(with interrupt).

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  qwer: Remote frame send(data request)
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

```
HSBRX24U CAN Starter kit program boot.
Copyright (C) 2021-2024 HokutoDenshi. All Rights Reserved.

SAMPLE3: CAN [Data frame] send/receive program(with interrupt, use FIFO).
            [Remote frame] request/response program(with interrupt, use FIFO).

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  qwer: Remote frame send(data request)
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])
```

SAMPLE3 では、新たに q~r のコマンドが追加されています。これは、リモートフレーム送信コマンドです。

・q~r コマンド

リモートフレーム要求メッセージの送信

を行います。

コマンド	ID	IDE	RTR	DLC
q	0x0000000	1	1	1
w	0x0000001	1	1	2
e	0x0000002	1	1	4
r	0x0000003	1	1	8

マイコンボード(1)の端末で、"q"を押した場合、マイコンボード(1)が ID=0x0000000, 相手に 1 バイトのデータを送るよう要求します（リモートフレーム送信を行います）。

```
++
CAN0 remote frame send,  ret=0 id_type=EID id=0x00000000 rtr=0x01 dlc=1
CAN0 send finished.(FIFO)
CAN0 data received, id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0xA1 ts=0x7913
--
```

(1)

(4)

マイコンボード(2)では、受信したパケットがリモートフレームであると認識して、データを送信します。

```
++
CAN0 data received, id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0xCE3E
CAN0 response data send, id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0xA1 ret=0
CAN0 send finished.(SRFIFO)
--
```

(2)

(3)

画面表示は一瞬で処理されますので、時系列が判り難いですが、
(1)リモートフレームの送信(ID=0x0000000, DLC=0x1)
(2)リモートフレームの受信(RTR=1)
(3)データの送信（リモートフレームの応答）
(4)データの受信
の順で行われます。

リモートフレームのレスポンスとして送信されるデータは、
0xA1A2A3A4A5A6A7A8
です（要求バイト数に応じて先頭から送信，プログラム内で固定）。

本サンプルプログラムでは、（割り込みルーチン内で処理される）データ受信時、データフレームだと、端末に表示するのみ。リモートフレームでは、端末表示と、データ送信（応答）を行います。

標準フォーマットでリモートフレームを受信した際は、標準フォーマットのデータフレームを送信します。拡張フォーマットのリモートフレームに対しては、拡張フォーマットで返信を返します。リモートフレーム応答の ID は、受信した ID と同じものを使用します。

端末表示は
++
--
で区切られますが、これは表示を見易くするための表示です。処理の先頭に+++、一連の処理の終了後---が表示されます。

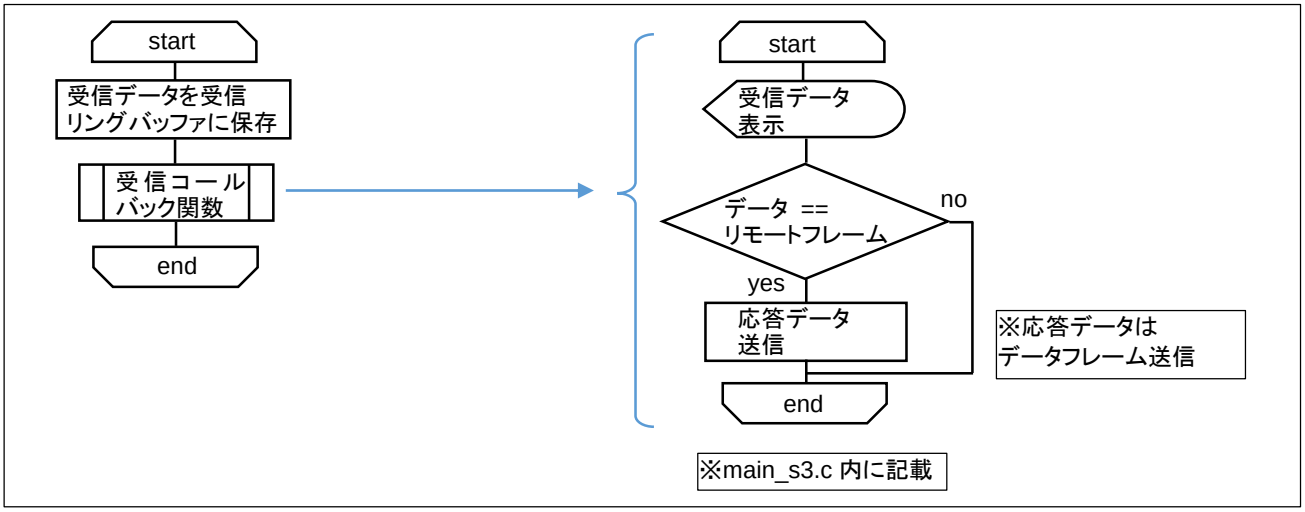
リモートフレーム送信時に、DLC 値が小さい場合で送信相手が直ぐにデータを返送してきた場合、送信完了割り込みより、受信割り込みが先に生じる場合もあります。

```
++
CAN0 remote frame send,  ret=0 id_type=SID id=0x00000000 rtr=0x01 dlc=1
CAN0 data received, id_type=SID id=0x00000000 rtr=0x00 dlc=1 data=0xA1 ts=0x4041
--
CAN0 send finished.(SRFIFO)
--
```

その場合は、++ ～ --で、一連のメッセージが区切られない場合もあります。（送信完了割り込みと受信割り込みの順番が異なっていても、処理としては正しく処理が行われています。）

※SAMPLE3 のプログラムを書き込んだマイコンボードを 3 台以上同一バスに接続し、q~r のコマンドを使用すると、リモートフレームを送信したボード以外の全てのボードがデータを返しますので、表示が見難くなります。

－受信割り込みの簡易フローチャート－



受信割り込み関数は、SAMPLE2 と SAMPLE3 で変わりません（SAMPLE に依存せず同じソースコードです）。コールバック関数は、main_s3.c 内に記載されており、この中で受信したデータがリモートフレームの場合、応答データを送信する処理を入れています。

6.4. サンプルプログラム(SAMPLE4)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示
・端末からの指示でリモートフレームの送信
・リモートフレーム受信時にレスポンスを送信

を行うサンプルプログラムです（機能的には SAMPLE3 と同じ）。SAMPLE3 に対して、CANFD 対応を行ったものです。

※SAMPLE4 は、CANFD 対応マイコン（RA6M5, RA6T2, RX660 等）のみ用意されています

マイコンボード(1) HSBA6M5F176

マイコンボード(2) HSBRA6T2F100

の場合の表示例を以下に示します。

```
HSBRA6M5F176 CAN Starter kit program boot.
Copyright (C) 2022-2024 HokutoDenshi. All Rights Reserved.

SAMPLE4: CANFD [Data frame] send/receive program(with interrupt, use FIFO).
              [Remote frame] request/response program(with interrupt, use FIFO).

CAN ID mode -> EID

CANFD setting:

  Please input in ( ) value, Enter to use default value

  CANFD speed [Mbps](2-8(3:3.33Mbps, 6,7:6.67Mbps), default:2)) >
```

(1)

起動すると、CANFD 各種設定を端末から入力します。
最初に(1)CANFD の速度を聞かれますので、端末から 2~5 の数値を入力します。
[Enter]を押すとデフォルト値の 2(2Mbps)が設定されます。

※ボードにより最大 8(8Mbps)の入力が可能ですが、ボードに搭載している、CAN トランシーバ IC が最大 5Mbps（TJA1044 搭載ボード）の場合は、8Mbps では通信が行えません。
※CANFDCLK の値により、選択可能なビットレートは変わります

```
  CAN transceiver delay compensation(y/n, default:n) >        (2)
```

次は、(2)トランシーバの遅延補償を行うかの選択です。通信レートが 4Mbps 以下であれば、n（行わない）が良いと思います。5Mbps より高いビットレートであれば、y（行う）が良いです。

```
CAN transceiver delay compensation(y/n, default:n) >y

  CAN transceiver secondary sampling point(d/o, d:delay+offset, o:offset,
default:d) >    (3)
```

(2)で y を選択した場合、(3)トランシーバの遅延補償の方式として、d（ディレイ測定値＋固定オフセット値）か o（固定オフセット値のみ）かを選択します。

```
CAN transceiver secondary sampling point delay(0-255, default:0) >        (4)
```

次に、(4)トランシーバ遅延補償の固定オフセット値(0~255)を入力します。

```
  CAN transceiver RX edge filter(y/n, default:n) >        (5)
```

次に、(5)RX のエッジフィルタの有効／無効を入力します。デフォルトは n（無効）です。

なお、(2)で n（トランシーバの遅延補償を行わない）を選択した場合は、(3)(4)の選択肢は表示されず、(2)の次に(5)の入力となります。

CANFD では、単純な速度設定以外に、トランシーバの遅延補償等のパラメータ設定があり、これらの値を変更した際にどうなるか（通信が通るかどうか）を見られるように、起動後に各種パラメータの設定が行えるようにしています。（なお、通常 CAN の速度設定は、can_operaton.h 内で選択するようになっています。通常 CAN の速度変更に関しては、ヘッダファイルを編集して、プログラムを再ビルドしてください。CANFD の速度設定は、ボードのリセットや電源再投入で値を変えられるようになっています。）

メニューで設定されるパラメータ

	設定内容	CANFD, CANFD_B, CANFD_2 での レジスタ値(RA6M5, RA6T2 他)	CANFD-Lite でのレジスタ値 (RX660 他)	デフォルト値 (括弧内レジスタ値)
(1)	CANFD 速度	CFDCnDCFG.DTSEG1 CFDCnDCFG.DTSEG2	CANFD0.DBCR.TSEG1 CANFD0.DBCR.TSEG2	2 [2Mbps]
(2)	トランシーバ遅延補償	CFDCnFDCFG.TDCE	CANFD0.FDCFG.TDCE	n(0) [no]
(3)	トランシーバ遅延補償方式	CFDCnFDCFG.TDCOE	CANFD0.FDCFG.SSPC	d(0) [delay+offset]
(4)	遅延固定オフセット値	CFDCnFDCFG.TDCO	CANFD0.FDCFG.TDCO	0(0)
(5)	RX エッジフィルタ	CFDCnFDCFG.REFE	CANFD0.FDCFG.REFE	n(0) [n:no]

メニュで選択した項目は上表のレジスタ値に反映されます。

※とりあえず動作確認する場合は、[Enter]でデフォルト値を選択して先に進めてください。

※2 台のボードで通信速度は合わせる必要があります

－設定不可の速度を選択した場合－

```
Command Usage:
  0123: Data frame send
(中略)
  (Push-SW: Data frame send [=keyboard 0])

**********
CAN bitrate parameter setting error!
**********
```

コマンド一覧の後で、ビットレート設定エラーのメッセージが表示されます。このメッセージが表示された場合は、ビットレートの選択値を変えてください。
（もしくは、CANFDCLK の周波数、board.h 内の CANFDLCK 定義値を変えてください。現状の CANFDCLK の値では設定不可のビットレートが選択されています。）

・端末 1 から'3'を入力（データ送信）

```
++
CAN1 data frame send,       ret=0 id_type=EID id=0x00000003 rtr=0x00 dlc=15
data=0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223
2425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
CAN1 send finished.(CFIFO)
--
```

・サンプルプログラムのキーボードから入力可能なコマンド（SAMPLE4 での変更）

コマンド	ID	IDE	RTR	DLC （送信,要求バイト数）	備考
0	0x0000000	1	0	1(1)	データフレーム送信
1	0x0000001	1	0	8(8)	データフレーム送信
2	0x0000002	1	0	10(16)	データフレーム送信
3	0x0000003	1	0	15(64)	データフレーム送信
q	0x0000000	1	1	1(1)	リモートフレーム送信
w	0x0000001	1	1	8(8)	リモートフレーム送信
e	0x0000002	1	1	10(16)	リモートフレーム送信
r	0x0000003	1	1	15(64)	リモートフレーム送信

0~3 がデータ送信、q~r がリモートフレームデータ要求である事は SAMPLE3 と変わりないです。送信、要求するデータサイズが、SAMPLE4 では 1,8,16,64 バイトとなっています。CAN では 1 パケットのデータサイズは最大 8 バイトですが、CANFD では 64 バイトに拡張されていますので、SAMPLE4 では送信データサイズを変えています。
（SAMPLE4 での送信データは、0x00, 0x01, 0x02, ...の様に単純にバイト毎のインクリメント値です。）

```
++
CAN0 data received, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 esi=0x00
dlc=15
data=0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223
2425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F ts=0x61A7
--
```

データを受信した側では、上記の様な表示となります。fdf と brs と esi の表示が SAMPLE3 に対して追加されています。

・端末 1 から'r'を入力（リモートフレームデータ要求）

```
++
CAN1 remote frame send,  ret=0 id_type=EID id=0x00000003 rtr=0x01 dlc=15    (1)
CAN1 send finished.(CFIFO)
CAN1 data received, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 esi=0x00
dlc=15
data=0xFFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0DFDEDDDCDB
DAD9D8D7D6D5D4D3D2D1D0CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0 ts=0x516D    (3)
--
```

相手先に対して、リモートフレーム要求を出した場合です。動作の時系列は(1)→(2)→(3)となります。

```
++
CAN0 data received, id_type=EID id=0x00000003 rtr=0x01 fdf=0x00 brs=0x00 esi=0x00
dlc=15 ts=0xB49F
CAN0 response data send, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01
dlc=15
data=0xFFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0DFDEDDDCDB
DAD9D8D7D6D5D4D3D2D1D0CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0 ret=0    (2)
CAN0 send finished.(CFIFO)
--
```

リモートフレーム要求は、CAN パケット（ビットレートが最初から最後まで同じ）となります。そのため、受信側では **fdf=0x00, brs=0x00** となっています。リモートフレーム要求に対する応答は、**fdf=0x01, brs=0x01** となっておりCANFD パケットでの応答としています。
（リモートフレーム応答のデータは、0xff からのバイト毎のデクリメントしたデータ、0xff, 0xfe, 0xfd, ...としています）

データ送信時は（データ送信、リモートフレーム応答）、CANFD パケットとし、データ送信バイト数（CANFD での最大 64 バイト）を変更したものが SAMPLE4 であり、その他の動作（割り込みで受信処理を行う等）は、SAMPLE3 と同様です。

6.5. サンプルプログラム(SAMPLE_MONITOR)

・CAN の通信速度を選択（1Mbps, 500kbps, 250kbps, 125kbps(*1)）
・端末からの指示でデータフレームの送信（任意の ID のデータを受信）
・データフレームの受信、端末への表示
・端末からの指示でリモートフレームの送信
・リモートフレーム受信時にレスポンスを送信（起動時は OFF）

を行うサンプルプログラムです。基本的には、既存の CAN バスに接続して、どのようなデータが流れているかの簡易モニタリングを行うサンプルプログラムです。

(*1)CANFDCLK=80MHz の場合、125kbps は選択できません
（CANFDCLK=80MHz のマイコンで、125kbps とする場合は、CANFDCLK の周波数を落とすか、CAN のクロック分周比を 2 以上に設定する必要があります。）

マイコンボード(1) HSBX72M176

マイコンボード(2) HSBRX26T100

の場合の表示例を以下に示します。

```
CAN Starter kit program boot.
Copyright (C) 2018-2024 HokutoDenshi. All Rights Reserved.

MONITOR: CAN [Data frame] send/receive program(with interrupt).
            [Remote frame] request(/response program)(with interrupt).

CAN ID mode -> EID

CAN speed setting:

  Please input in ( ) value, Enter to use default value

  CAN speed (1-4, 1:1Mbps, 2:500kbps, 3:250kbps, 4:125kbps, default:1)) >  (1)
```

起動すると、(1)CAN の通信速度を聞かれますので、1~4 から選択してください。数字を入力せずにリターンキーを押すと、1Mbps が選択されます。

```
Input summary:

  CAN speed [kbps] > 1000

Command Usage:
  d: Data frame send(CAN, DLC=8(default))
  r: Remote frame send(data request, DLC=8(default))
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  b: DLC byte set
  R: Remote frame response/NOT response(toggle)
  m: message clear
  c: send CAN ch change
```

SAMPLE1~4 から変更しているコマンドに関して説明します。

・d コマンド

データフレームの送信

を行います。従来の 0~3 コマンドの代わりとなるコマンドです。DLC の値は、b コマンドで変更可能です（起動時の DLC 値は 8）。

送信データは、0x00, 0x01, 0x02, ...となります。ID は、0x0000000 です。（変更時は、main_monitor.c を編集）

```
CAN0  data  frame  send,          ret=0  id_type=EID  id=0x00000000  rtr=0x00  dlc=8
data=0x0001020304050607
CAN0 send finished.(FIFO)
```

・r コマンド

リモートフレームの送信

を行います。従来の q~r コマンドの代わりとなるコマンドです。DLC の値は、b コマンドで変更可能です。

```
CAN0 remote frame send,  ret=0 id_type=EID id=0x00000000 rtr=0x01 dlc=8
CAN0 send finished.(FIFO)
[送信したリモートフレームに対し応答がある場合はこの部分に受信データが表示されます]
```

・b コマンド

DLC 値を変更します。

```
DLC set (0-8)>7
  DLC = 7

CAN0  data  frame  send,          ret=0  id_type=EID  id=0x00000000  rtr=0x00  dlc=7
data=0x00010203040506
CAN0 send finished.(FIFO)
```

b コマンドの後で、DLC 値を聞かれます。7[リターン]を入力すると、DLC の値が 7 になります。（その後、d コマンドを入力すると、7 バイトのデータを送信します。）

・R コマンド

リモートフレーム応答を有効化します。

起動後は、リモートフレームには応答しません。

```
Remote frame response flag change:
  CAN0 : response
  CAN1 : Not-response [not valid ch]
  CAN2 : Not-response
```

1 回 R コマンドを入力すると、CAN-ch0 がリモートフレーム応答に設定されます。

何度か R コマンドを入力すると

```
Remote frame response flag change:
  CAN0 : Not-response
  CAN1 : response [not valid ch]
  CAN2 : Not-response
```

→CAN-ch1 が応答有効状態であるが、CAN-ch1 は現時点では無効化されているために意味なし

```
Remote frame response flag change:
  CAN0 : Not-response
  CAN1 : Not-response [not valid ch]
  CAN2 : response
```

→CAN-ch2 が応答有効状態

```
Remote frame response flag change:
  CAN0 : response
  CAN1 : Not-response [not valid ch]
  CAN2 : response
```

→CAN-ch0 と CAN-ch2 が応答有効状態

R コマンドを入力する度に、response/Not-response が切り替わります。

応答有効状態で、リモートフレームを受信すると、応答（データフレーム）を送信します。

```
CAN0 data received, id_type=EID id=0x00000000 rtr=0x01 dlc=8 ts=0xBFAA
CAN0 response data send, id_type=EID id=0x00000000 rtr=0x00 dlc=8 data=0xFFFEFDFCFBFAF9F8
ret=0
CAN0 send finished.(FIFO)
```

送信する EID（フォーマット）, ID, DLC は、リモートフレーム要求のデータの値。送信データは、0xff からの減算値となります。

・m コマンド
端末のクリア
を行います。

次に受信するデータを確認したい場合等に入力してください。

次に CANFD 対応マイコンでの表示例を示します。

```
CAN Starter kit program boot.
Copyright (C) 2022-2024 HokutoDenshi. All Rights Reserved.

MONITOR: CANFD/CAN [Data frame] send/receive program(with interrupt, use FIFO).
                   [Remote frame] request(/response program)(with interrupt, use FIFO).

CAN ID mode -> EID

CAN speed setting:

  Please input in ( ) value, Enter to use default value

  CAN speed (1-4, 1:1Mbps, 2:500kbps, 3:250kbps, 4:125kbps, default:1)) >1
```

起動すると、CAN の通信速度を聞かれるのは、CANFD 非対応のマイコンと同様です。

```
Input summary:

  CAN speed [kbps] > 1000

CANFD setting:

  Please input in ( ) value, Enter to use default value

  CANFD speed [Mbps](2-7(3:3.33Mbps,7:7.5Mbps)[CANFDCLK=60MHz], default:2)) >2

  CAN transceiver delay compensation(y/n, default:n) >n

  CAN transceiver RX edge filter(y/n, default:n) >n

Input summary:

  CANFD speed [kbps] > 2000
  CAN transceiver delay compensation > n
  [not use] CAN transceiver secondary sampling point > delay+offset
  [not use] CAN transceiver secondary sampling point delay > 0
  CAN transceiver RX edge filter > n
```

次に、SAMPLE4 同様に、CANFD の速度や遅延補償などのパラメータを聞かれますので、入力してください。
（リターンで入力を進めると、デフォルト値（2Mbps, 遅延補償なし）が入力されます。）

```
Command Usage:
  d: Data frame send(CAN, DLC=8(default))
  D: Data frame send(CANFD, DLC=8(default))
  r: Remote frame send(data request, DLC=8(default))
  z: LED blink test(for board identify)
  s: send format EID <-> SID
  a: send abort
  S: Status register print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  b: DLC byte set
  R: Remote frame response/NOT response(toggle)
  f: Remote frame response format CAN/CANFD(toggle)
  m: message clear
```

CANFD 非対応のプログラムとの相違点を示します。

・D コマンド

CANFD のデータフレームの送信

を行います。

```
CAN0 data frame send,    ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 dlc=8
data=0x0001020304050607
```

FDF=BRS=1 で、データ部分のビットレートが CANFD 速度設定で設定した値での送出となります。

・b コマンド

b コマンド自体の動作は変わりませんが、DLC 値として 15 までの値が設定可能です。

```
DLC set (0-15)>15
  DLC = 15

CAN0 data frame send,    ret=-2   d コマンド                                    D コマンド
CAN0 data frame send,     ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01
dlc=15
data=0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F2021222324252627
28292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
```

なお、DLC=9 以上の値に設定した状態で、d コマンドで CAN のメッセージを送信しようとするとエラー(-2:引数エラー)を返します。、CANFD のメッセージは、DLC=9~15 でも送信可能です。

・f コマンド

リモート応答時の CAN/CANFD メッセージの切り替え

を行います。

```
Remote frame response format change:
        format -> CANFD(FDF=1)
```

f コマンドを入力すると、リモートフレームに対するレスポンスが、CANFD フォーマットになります。

```
Remote frame response format change:
        format -> CAN(FDF=0)
```

もう一度 f コマンドを入力すると、CAN フォーマットとなります。トグルで CAN←→CANFD が切り替わります。
（起動時のデフォルトは、リモートフレーム応答を行いませんので、リモートフレーム応答を行いたい場合は、R コマンドで応答を有効化してください。）

リモートフレームは、常に CAN フォーマットでの送信となり、リモートフレーム内のデータには CAN フォーマットで返送すべきか CANFD フォーマットで返送すべきかの情報が含まれませんので、コマンドで切り替えるようにしています。

（本モニタプログラムでは、CANFD 対応マイコンでも、CANFD 非対応のバスのデータを拾ったり、データを送出したりすることができます。）
※CAN バス上に 1 つでも CANFD 非対応のデバイスが存在する場合、CANFD フォーマットのデータを流すと、バスエラー等が発生します

7. サンプルプログラムの補足説明

7.1. サンプルプログラム(SAMPLE1)

〇CAN モジュール

RX600/700/RA 系マイコンに搭載されている CAN モジュールでは、「メールボックス」というものを利用してデータの送受信を行います。メールボックスは CAN データ（メッセージ）の送受信に使用する箱の様なもので、送信時はメールボックスにデータを格納した後で送信、受信時はメールボックスにデータが格納される形となります。メールボックスは 32 個あり、32 個のメールボックスとして使用するか、24 個のメールボックス＋4 段の送信 FIFO＋4 段の受信 FIFO として使用するか選択可能です。SAMPLE1 では、FIFO は未使用で、32 個のメールボックスとして使用しています。

本サンプルプログラムでの ID とメールボックスの設定を以下に示します。

・送信

メールボックス番号	フォーマット(IDE)	ID	RTR	送受信区分
0~3	送信時に指定	送信時に指定	送信時に指定	送信

コマンド 0~3 がメールボックス 0~3 に対応します。

・受信

メールボックス番号	フォーマット(IDE)	ID	RTR	送受信区分
4	標準(SID)	0x000	0	受信
5	標準(SID)	0x001	0	受信
6	標準(SID)	0x002	0	受信
7	標準(SID)	0x003	0	受信
8	拡張(EID)	0x0000000	0	受信
9	拡張(EID)	0x0000001	0	受信
10	拡張(EID)	0x0000002	0	受信
11	拡張(EID)	0x0000003	0	受信

SAMPLE1 では、メールボックス 4~11 を使う設定です。
（表にない、フォーマット、ID、RTR の組み合わせに関しては、受信しません。）
（RTR=0 を指定しているので、データフレームのみ受信します。）
（受信しない＝受信データがメールボックスに格納されない。データ受信時に ACK は返します。）

〇RSCAN モジュール

RX200 系マイコンに搭載されている、RSCAN モジュールは、送信バッファ、送受信 FIFO バッファを使用してメッセージの送信を行います。受信バッファ、受信 FIFO、送受信 FIFO を使用してメッセージの受信を行います。本サンプルプログラムでは、送信バッファと受信バッファを使用しています。

・送信

フォーマット(IDE)	ID	RTR	送信元
送信時に指定	送信時に指定	送信時に指定	送信バッファ 0~3

・受信ルール設定

受信ルール番号	フォーマット(IDE)	ID	RTR	受信バッファ番号
0	標準(SID)	0x000	0	0
1	標準(SID)	0x001	0	1
2	標準(SID)	0x002	0	2
3	標準(SID)	0x003	0	3
4	拡張(EID)	0x0000000	0	4
5	拡張(EID)	0x0000001	0	5
6	拡張(EID)	0x0000002	0	6
7	拡張(EID)	0x0000003	0	7

RACAN, CANFD 系モジュールでは、「受信ルール」を設定する形で、ルールにマッチしたデータが、受信ルールで定められた格納先に格納されます。

RSCAN モジュールは、16 個の受信ルールが設定可能で、ルールにマッチした（ID などが一致した）データの格納先を設定できますが、本サンプルプログラムでは、受信バッファに割り当てています。（受信バッファは、16 個あります。受信ルールと、受信バッファは自由に紐付けできます。本サンプルプログラムでは、受信ルール番号と受信バッファの同じ番号のものを 1:1 で紐付けさせていますが、本来設定は自由です。）

〇CANFD 系モジュール

RA6M5 マイコンに搭載されている、CANFD モジュール、RA6T2 の CANFD_B モジュール、RX660 の CANFD-Lite モジュール、及び RA8 の CANFD モジュール（本マニュアルでは CANFD_2 と記載）は、送信メッセージバッファ、送信キュー、共通 FIFO を使用してメッセージの送信。受信メッセージバッファ、受信 FIFO、共通 FIFO を使用してメッセージの受信を行います。本サンプルプログラムでは、送信メッセージバッファと、受信メッセージバッファを使用しています。

・CANFD モジュール(RA6M5)送信

CAN-ch	フォーマット(IDE)	ID	RTR	送信元
CAN0	送信時に指定	送信時に指定	送信時に指定	送信バッファ 0~3
CAN1	送信時に指定	送信時に指定	送信時に指定	送信バッファ 64~67

CANFD 系では、受信ルールをアクセプタンスフィルタリスト(AFL)という名称で呼んでいますが、要は受信ルールの事です。

・CANFD モジュール(RA6M5)AFL 設定

アクセプタンスフィルタリスト番号(0-15) (ページ番号:0-7)	フォーマット(IDE)	ID	RTR	受信メッセージバッファ番号 n=0,1:CAN チャネル
0, (page=0+4n)	標準(SID)	0x000	0	0+16×n
1, (page=0+4n)	標準(SID)	0x001	0	1+16×n
2, (page=0+4n)	標準(SID)	0x002	0	2+16×n
3, (page=0+4n)	標準(SID)	0x003	0	3+16×n
4, (page=0+4n)	拡張(EID)	0x0000000	0	4+16×n
5, (page=0+4n)	拡張(EID)	0x0000001	0	5+16×n
6, (page=0+4n)	拡張(EID)	0x0000002	0	6+16×n
7, (page=0+4n)	拡張(EID)	0x0000003	0	7+16×n

CANFD では、アクセプタンスフィルタリスト（16 ルール×8 ページ）で設定したルールにマッチした（ID 等が一致した）データの格納先を最大 8 個設定できます(*1)。本サンプルプログラムでは、データの格納先を受信メッセージバッファに割り当てています。（受信メッセージバッファは、32 個あり、アクセプタンスフィルタのルールと、受信メッセージバッファは自由に紐付けできます。）

本サンプルプログラムでは、アクセプタンスフィルタリストは CAN-ch0 側はページ 0-3（設定可能ルール数 16x4=64 ルール）、CAN-ch1 側はページ 4-7 を割り当てています。受信バッファは、CAN-ch0 側は 0~15、CAN-ch1 側は 16~31 を使用しています。（本サンプルプログラムでは、アクセプタンスフィルタリストと受信バッファを、CAN-ch0 と CAN-ch1 で半分ずつ割り当てていますが、本来割り当ては自由です。）

・CANFD_B モジュール(RA6T2 等), CANFD-Lite モジュール(RX660 等)送信

フォーマット(IDE)	ID	RTR	送信元
送信時に指定	送信時に指定	送信時に指定	送信バッファ 0~3

・CANFD_B モジュール(RA6T2 等), CANFD-Lite モジュール(RX660 等)AFL 設定

アクセプタンスフィルタリスト番号	フォーマット(IDE)	ID	RTR	受信メッセージバッファ番号
0, (page=0)	標準(SID)	0x000	0	0
1, (page=0)	標準(SID)	0x001	0	1
2, (page=0)	標準(SID)	0x002	0	2
3, (page=0)	標準(SID)	0x003	0	3
4, (page=0)	拡張(EID)	0x0000000	0	4
5, (page=0)	拡張(EID)	0x0000001	0	5
6, (page=0)	拡張(EID)	0x0000002	0	6
7, (page=0)	拡張(EID)	0x0000003	0	7

CANFD_B, CANFD-Lite では、アクセプタンスフィルタリスト（16 ルール×2 ページ）で設定したルールにマッチした（ID 等が一致した）データの格納先を最大 2 個設定できます(*1)。

本サンプルプログラムでは、データの格納先を受信メッセージバッファに割り当てています。（受信メッセージバッファは、32 個あり、アクセプタンスフィルタのルールと、受信メッセージバッファは自由に紐付けできます。）

・CANFD_2 モジュール(RA8M1 等)送信

CAN-ch	フォーマット(IDE)	ID	RTR	送信元
CAN0	送信時に指定	送信時に指定	送信時に指定	送信バッファ 0~3
CAN1	送信時に指定	送信時に指定	送信時に指定	送信バッファ 0~3

CANFD_2 モジュールでは送信バッファは、ch 毎に独立しているので、CAN-ch に拘わらず、送信バッファ 0~3 を使います。

・CANFD_2 モジュール(RA8M1 等)AFL 設定

アクセプタンスフィルタリスト番号	フォーマット	ID	RTR	受信メッセージバッファ番号
0	標準(SID)	0x000	0	0
1	標準(SID)	0x001	0	1
2	標準(SID)	0x002	0	2
3	標準(SID)	0x003	0	3
4	拡張(EID)	0x0000000	0	4
5	拡張(EID)	0x0000001	0	5
6	拡張(EID)	0x0000002	0	6
7	拡張(EID)	0x0000003	0	7

CANFD_2 では、アクセプタンスフィルタリスト（16 ルール×1 ページ／ch 毎）で設定したルールにマッチした（ID 等が一致した）データの格納先を最大 2 個設定できます(*1)。本サンプルプログラムでは、データの格納先を受信メッセージバッファに割り当てています。（受信メッセージバッファは、16 個／ch あり、アクセプタンスフィルタのルールと、受信メッセージバッファは自由に紐付けできます。）

(*1)マイコンの機能としては、とあるメッセージを複数の受信先にコピーして持たせる事が可能ですが、本サンプルプログラムでは受信先は 1 箇所に限定しています。

－サンプルプログラムのキーボードから入力するデータ送信コマンド－

コマンド	ID	DLC 送信バイト数	送信データ	CAN 使用メール ボックス番号	RSCAN CANFD_Lite CANFD_B CANFD_2 送信バッファ 番号	CANFD 送信バッファ 番号 n=0,1:CAN チャネル
0	0x0000000	1	0x01	0	0	0+n×64
1	0x0000001	2	0x0123	1	1	1+n×64
2	0x0000002	4	0x01234567	2	2	2+n×64
3	0x0000003	8	0x0123456789ABCDEF	3	3	3+n×64

サンプルプログラムでは、0~3 のキーボード入力に対し、上記の ID、送信バイト数(DLC)、送信データを送出します。CAN モジュールでは、0~3 のメールボックス番号を使用します。RSCAN, CANFD_Lite, CANFD_B, CANFD_2 モジュールでは、4 つある送信バッファの内 0~3 の番号のバッファを使用しています。CANFD モジュールでは、CAN ch0 側は 0~3 の番号のバッファ、CAN ch1 側は、64~67 の番号のバッファを使用しています。

データの受信に関しては、標準・拡張フォーマット区分(IDE)及び、ID に応じたメールボックス、受信ルールが適用されます。例えば、拡張フォーマットで ID=0x0000002 のデータを受信した際、

メールボックス 10(MB=0x0A)にデータが格納される（CAN モジュール）
受信ルール 6 にマッチし、受信バッファ 6 にデータが格納される（RSCAN モジュール）
アクセプタンスフィルタ 6 にマッチし、受信メッセージバッファ 6 にデータが格納される（CANFD(CAN-ch0), CANFD_B, CANFD_2, CANFD-Lite モジュール系）
となります。

7.2. サンプルプログラム(SAMPLE2)

SAMPLE2 では、送受信の方式を SAMPLE1 から変更しています。
（can_operation2.h で送受信の方式を選べるようになっています。）

CAN モジュールでは、
受信：メールボックス FIFO を使用（フォーマット=EID の場合）
　　メールボックスを使用（フォーマット=SID の場合）
送信：メールボックス FIFO を使用

RSCAN モジュールでは、
受信：受信 FIFO(RXFIFO)を使用
送信：送受信 FIFO(SRFIFO)を使用

CANFD 系モジュールでは、
受信：受信 FIFO(RXFIFO)を使用
送信：共通 FIFO(CFIFO)を使用

としています。

－送受信方式に関して－

	CAN モジュール	RSCAN モジュール	CANFD モジュール	CANFD_B CANFD_2 CANFD_Lite モジュール
受信方式	**メールボックス**	受信バッファ(*1)	受信バッファ(*1)	受信バッファ
	メールボックス FIFO	**受信 FIFO**	**受信 FIFO**	**受信 FIFO**
		送受信 FIFO(*2)	共通 FIFO	共通 FIFO(*2)
送信方式	メールボックス	送信バッファ	送信バッファ	送信バッファ
	メールボックス FIFO	**送受信 FIFO**(*2)	**送受信 FIFO**	**共通 FIFO**(*2)
			送信キュー	送信キュー

(*1)割り込みによる受信はできません
(*2)送受信 FIFO，共通 FIFO は 1 本のため、送信または受信のどちらか一方での利用のみ可

※**太字**は SAMPLE2 のデフォルト設定（can_operation2.h で変更可能）

※メールボックス、送信バッファ、受信バッファは 1 つのメッセージのみ格納可能
　FIFO，送信キューには複数のメッセージの格納が可能

受信、送信には複数の手法があり、詳細は各モジュールのソフトウェア編マニュアルで説明します。

○CAN モジュール

・送信

メールボックス番号	フォーマット(IDE)	ID	RTR	送受信区分
24~27(FIFO)	送信時に指定	送信時に指定	送信時に指定	送信(FIFO)

・受信

メールボックス番号	フォーマット(IDE)	ID	RTR	送受信区分
4	標準(SID)	0x000	0	受信
5	標準(SID)	0x001	0	受信
6	標準(SID)	0x002	0	受信
7	標準(SID)	0x003	0	受信
28~31(FIFO)	拡張(EID)	任意	任意	受信(FIFO)

SAMPLE2 では、FIFO を使用する設定です。FIFO を使用すると、メールボックス番号 24~27 が送信 FIFO。28~31 が受信 FIFO となります。FIFO は、送信 FIFO，受信 FIFO 共に 4 段（4 つのメッセージを格納可能）となります。

受信は、拡張 ID のデータを FIFO で受信します。標準 ID のデータは、メールボックスで受信します。（SAMPLE2 では、FIFO とメールボックスによる受信の 2 本立てです。）

※受信 FIFO では、任意の ID を受信する設定なので、ID=0x0~0x3 以外のデータも受信します。

（CAN モジュールの拡張 ID 受信時のみ。他のモジュールでは、（拡張 ID，標準 ID とも）ID=0x0~0x3 を受信する設定です。）

RSCAN モジュール、CANFD 系モジュールでは、受信ルール、AFL の受信先が「受信バッファ」→「受信 FIFO」に変わります。

○RSCAN モジュール

・送信

フォーマット(IDE)	ID	RTR	送信元
送信時に指定	送信時に指定	送信時に指定	SRFIFO(0)

送受信 FIFO は 1 本のみなので、送信時は SRFIFO(0)を使用する設定です。

・受信ルール設定

受信ルール番号	フォーマット(IDE)	ID	RTR	受信先
0	標準(SID)	0x000	0	RXFIFO(0)
1	標準(SID)	0x001	0	RXFIFO(0)
2	標準(SID)	0x002	0	RXFIFO(0)
3	標準(SID)	0x003	0	RXFIFO(0)
4	拡張(EID)	0x0000000	0	RXFIFO(0)
5	拡張(EID)	0x0000001	0	RXFIFO(0)
6	拡張(EID)	0x0000002	0	RXFIFO(0)
7	拡張(EID)	0x0000003	0	RXFIFO(0)

RXFIFO は、2 本ありますが、RXFIFO(0)で受信する設定です。

〇CANFD 系モジュール

・CANFD モジュール(RA6M5)送信

CAN-ch	フォーマット (IDE)	ID	RTR	送信元
CAN0	送信時に指定	送信時に指定	送信時に指定	CFIFO(0)
CAN1	送信時に指定	送信時に指定	送信時に指定	CFIFO(3)

CANFD モジュールでは、共通 FIFO は 6 本(RXFIFO(0)~RXFIFO(5))ありますが、CAN-ch0 は CFIFO(0), CAN-ch1 は RXFIFO(3)で受信する設定です。

・CANFD モジュール(RA6M5)AFL 設定

CAN-ch	アクセプタンスフィルタリスト番号(0-15) (ページ番号:0-7)	フォーマット (IDE)	ID	RTR	受信先 n=0,1:CAN チャネル
CAN0	0, (page=0)	標準(SID)	0x000	0	RXFIFO(0)
CAN0	1, (page=0)	標準(SID)	0x001	0	RXFIFO(0)
CAN0	2, (page=0)	標準(SID)	0x002	0	RXFIFO(0)
CAN0	3, (page=0)	標準(SID)	0x003	0	RXFIFO(0)
CAN0	4, (page=0)	拡張(EID)	0x0000000	0	RXFIFO(0)
CAN0	5, (page=0)	拡張(EID)	0x0000001	0	RXFIFO(0)
CAN0	6, (page=0)	拡張(EID)	0x0000002	0	RXFIFO(0)
CAN0	7, (page=0)	拡張(EID)	0x0000003	0	RXFIFO(0)
CAN1	0, (page=4)	標準(SID)	0x000	0	RXFIFO(1)
CAN1	1, (page=4)	標準(SID)	0x001	0	RXFIFO(1)
CAN1	2, (page=4)	標準(SID)	0x002	0	RXFIFO(1)
CAN1	3, (page=4)	標準(SID)	0x003	0	RXFIFO(1)
CAN1	4, (page=4)	拡張(EID)	0x0000000	0	RXFIFO(1)
CAN1	5, (page=4)	拡張(EID)	0x0000001	0	RXFIFO(1)
CAN1	6, (page=4)	拡張(EID)	0x0000002	0	RXFIFO(1)
CAN1	7, (page=4)	拡張(EID)	0x0000003	0	RXFIFO(1)

CANFD モジュールでは、受信 FIFO は 8 本(RXFIFO(0)~RXFIFO(7))ありますが、CAN-ch0 は RXFIFO(0), CAN-ch1 は RXFIFO(1)で受信する設定です。

HSBRA6M5F176 のボードでは、CAN1 が端子に出力されていますので、CAN-ch1 がデフォルトで有効化されています。(送信は、共通 FIFO の CFIFO(3)を使い、受信は受信 FIFO の RXFIFO(1)を使うのがデフォルトです。)

・CANFD_B モジュール(RA6T2 等), CANFD-Lite モジュール(RX660 等)送信

フォーマット (IDE)	ID	RTR	送信元
送信時に指定	送信時に指定	送信時に指定	CFIFO(0)

共通 FIFO は 1 本のみなので、送信時は CFIFO(0)を使用する設定です。

・CANFD_B モジュール(RA6T2 等), CANFD-Lite モジュール(RX660 等)AFL 設定

アクセプタンスフィルタリスト番号	フォーマット(IDE)	ID	RTR	受信先
0, (page=0)	標準(SID)	0x000	0	RXFIFO(0)
1, (page=0)	標準(SID)	0x001	0	RXFIFO(0)
2, (page=0)	標準(SID)	0x002	0	RXFIFO(0)
3, (page=0)	標準(SID)	0x003	0	RXFIFO(0)
4, (page=0)	拡張(EID)	0x0000000	0	RXFIFO(0)
5, (page=0)	拡張(EID)	0x0000001	0	RXFIFO(0)
6, (page=0)	拡張(EID)	0x0000002	0	RXFIFO(0)
7, (page=0)	拡張(EID)	0x0000003	0	RXFIFO(0)

RXFIFO は、2 本ありますが、RXFIFO(0)で受信する設定です。

・CANFD_2 モジュール(RA8M1 等)送信

CAN-ch	フォーマット(IDE)	ID	RTR	送信元
CAN0	送信時に指定	送信時に指定	送信時に指定	CFIFO(0)
CAN1	送信時に指定	送信時に指定	送信時に指定	CFIFO(0)

CANFD_2 モジュールでは、共通 FIFO は ch 毎に 1 本で、送信時は CFIFO(0) を使用する設定です。

・CANFD_2 モジュール(RA8M1 等)AFL 設定

CAN-ch	アクセプタンスフィルタリスト番号(0-15)	フォーマット(IDE)	ID	RTR	受信先
CAN0	0	標準(SID)	0x000	0	RXFIFO(0)
CAN0	1	標準(SID)	0x001	0	RXFIFO(0)
CAN0	2	標準(SID)	0x002	0	RXFIFO(0)
CAN0	3	標準(SID)	0x003	0	RXFIFO(0)
CAN0	4	拡張(EID)	0x0000000	0	RXFIFO(0)
CAN0	5	拡張(EID)	0x0000001	0	RXFIFO(0)
CAN0	6	拡張(EID)	0x0000002	0	RXFIFO(0)
CAN0	7	拡張(EID)	0x0000003	0	RXFIFO(0)
CAN1	0	標準(SID)	0x000	0	RXFIFO(0)
CAN1	1	標準(SID)	0x001	0	RXFIFO(0)
CAN1	2	標準(SID)	0x002	0	RXFIFO(0)
CAN1	3	標準(SID)	0x003	0	RXFIFO(0)
CAN1	4	拡張(EID)	0x0000000	0	RXFIFO(0)
CAN1	5	拡張(EID)	0x0000001	0	RXFIFO(0)
CAN1	6	拡張(EID)	0x0000002	0	RXFIFO(0)
CAN1	7	拡張(EID)	0x0000003	0	RXFIFO(0)

CANFD_2 モジュールでは、AFL 設定が CAN-ch に独立しており、CAN-ch0, CAN-ch1 とも同じ設定（どちらも RXFIFO(0)で受信する設定)です。

－サンプルプログラムのキーボードから入力するデータ送信コマンド－

コマンド	ID	DLC 送信バイト数	送信データ	CAN 使用メール ボックス番号	RSCAN CANFD_Lite CANFD_B CANFD_2 送信 FIFO	CANFD 送信 FIFO
0	0x0000000	1	0x01	24~27(FIFO)	SRFIFO(0) CFIFO(0)	CFIFO(n) n=0:CAN-ch0 n=3:CAN-ch1
1	0x0000001	2	0x0123			
2	0x0000002	4	0x01234567			
3	0x0000003	8	0x0123456789ABCDEF			

－CANFD モジュールの受信割り込みに関して－

SAMPLE2 では割り込みを使用しますが、CANFD モジュールに関して受信先に応じて呼ばれる割り込み関数は以下の様になります。

受信先	割り込み先
RXFIFO(0)	CAN_RXF
RXFIFO(1)	CAN_RXF
RXFIFO(2)	CAN_RXF
RXFIFO(3)	CAN_RXF
RXFIFO(4)	CAN_RXF
RXFIFO(5)	CAN_RXF
RXFIFO(6)	CAN_RXF
RXFIFO(7)	CAN_RXF

RXFIFO は、8 本ありますが、どの RXFIFO で受信した場合でも同じ割り込み関数が呼ばれます。

本サンプルプログラムのデフォルトの受信先は、RXFIFO としていますが、受信先としては共通 FIFO を選ぶこともできます（can_operation2.h で選択）。

受信先として、共通 FIFO を選んだ場合の割り込み関数は以下の様になります。

受信先	割り込み先
CFIFO(0)	CAN0_COMFRX
CFIFO(1)	CAN0_COMFRX
CFIFO(2)	CAN0_COMFRX
CFIFO(3)	CAN1_COMFRX
CFIFO(4)	CAN1_COMFRX
CFIFO(5)	CAN1_COMFRX

CFIFO(0)~CFIFO(5)で受信した場合の割り込み先は、CAN0_COMFRX と CAN1_COMFRX に分かれます。
※CAN の物理 ch と CAN0_COMFRX と CAN1_COMFRX には関連がありません。
※本サンプルプログラムでは、CAN-ch0 は、CFIFO(0)~CFIFO(2)。CAN-ch1 は、CFIFO(3)~CFIFO(5)を使う事で、CAN-ch0 は CAN0_COMFRX の割り込み、CAN-ch1 は CAN1_COMFRX の割り込みに紐付けています。
（CAN-ch1 の受信に、CFIFO(1)を割り当てた場合は、CAN-ch1 の受信割り込関数は CAN0_COMFRX となります。）

7.3. サンプルプログラム(SAMPLE3)

本サンプルプログラムでは、受信ルールにリモートフレームが追加されています。送信に関しては、SAMPLE2 と変わりません。

〇CAN モジュール

CAN モジュールでのメールボックスは以下のものを設定しています。

・受信

メールボックス番号	フォーマット(IDE)	ID	RTR（データフレーム・リモートフレーム区分）	送受信区分
4	標準(SID)	0x000	データフレーム(0)	受信
5	標準(SID)	0x001	データフレーム(0)	受信
6	標準(SID)	0x002	データフレーム(0)	受信
7	標準(SID)	0x003	データフレーム(0)	受信
8	標準(SID)	0x000	リモートフレーム(1)	受信
9	標準(SID)	0x001	リモートフレーム(1)	受信
10	標準(SID)	0x002	リモートフレーム(1)	受信
11	標準(SID)	0x003	リモートフレーム(1)	受信
28~31	拡張(EID)	任意	データフレーム(0)	受信(FIFO)
	拡張(EID)	任意	リモートフレーム(1)	

FIFO での受信時は、ID フォーマット(IDE)、データフレーム・リモートフレーム区分(RTR)に関して、2 種類のフィルタの設定が可能です。
(1)拡張 ID-データフレーム
(2)拡張 ID-リモートフレーム
(3)標準 ID-データフレーム
(4)標準 ID-リモートフレーム
(1)~(4)の組み合わせの内 2 種類を受信 FIFO で受信する設定が行えます。本サンプルプログラムでは、(1)(2)を FIFO で受信するように設定しているので、標準 ID に関してはメールボックスで受信する設定としています。
（拡張 ID を使用しない様に設定すれば、(3)(4)を FIFO で受信する様にします。）

送信は、SAMPLE2 同様送信 FIFO を使う設定です。データフレームの送信、リモートフレームの送信、リモートフレームに応答するデータフレームの送信全て同じ送信 FIFO が送信元となります。

RSCAN, CANFD 系では、SAMPLE2 と同様に受信は、受信 FIFO。送信は、送受信 FIFO, 共通 FIFO を使う設定です。

RSCAN, CANFD 系モジュールの受信ルール設定は以下です。

〇RSCAN モジュール

・受信ルール

受信ルール番号	フォーマット(IDE)	ID	RTR（データフレーム・リモートフレーム区分）	受信先
0	標準(SID)	0x000	データフレーム(0)	RXFIFO(0)
1	標準(SID)	0x001	データフレーム(0)	RXFIFO(0)
2	標準(SID)	0x002	データフレーム(0)	RXFIFO(0)
3	標準(SID)	0x003	データフレーム(0)	RXFIFO(0)
4	標準(SID)	0x000	リモートフレーム(1)	RXFIFO(0)
5	標準(SID)	0x001	リモートフレーム(1)	RXFIFO(0)
6	標準(SID)	0x002	リモートフレーム(1)	RXFIFO(0)
7	拡張(EID)	0x003	リモートフレーム(1)	RXFIFO(0)
8	拡張(EID)	0x0000000	データフレーム(0)	RXFIFO(0)
9	拡張(EID)	0x0000001	データフレーム(0)	RXFIFO(0)
10	拡張(EID)	0x0000002	データフレーム(0)	RXFIFO(0)
11	拡張(EID)	0x0000003	データフレーム(0)	RXFIFO(0)
12	拡張(EID)	0x0000000	リモートフレーム(1)	RXFIFO(0)
13	拡張(EID)	0x0000001	リモートフレーム(1)	RXFIFO(0)
14	拡張(EID)	0x0000002	リモートフレーム(1)	RXFIFO(0)
15	拡張(EID)	0x0000003	リモートフレーム(1)	RXFIFO(0)

〇CANFD 系モジュール

・AFL（CANFD モジュール(RA6M5)）

アクセプタンスフィルタ番号 (n=0,1, CAN-ch)	フォーマット (IDE)	ID	RTR（データフレーム・リモートフレーム区分）	受信先 CAN-ch0 : n=0 CAN-ch1: n=1
0, (page=n×4)	標準(SID)	0x000	データフレーム(0)	RXFIFO(n)
1, (page=n×4)	標準(SID)	0x001	データフレーム(0)	RXFIFO(n)
2, (page=n×4)	標準(SID)	0x002	データフレーム(0)	RXFIFO(n)
3, (page=n×4)	標準(SID)	0x003	データフレーム(0)	RXFIFO(n)
4, (page=n×4)	標準(SID)	0x000	リモートフレーム(1)	RXFIFO(n)
5, (page=n×4)	標準(SID)	0x001	リモートフレーム(1)	RXFIFO(n)
6, (page=n×4)	標準(SID)	0x002	リモートフレーム(1)	RXFIFO(n)
7, (page=n×4)	標準(SID)	0x003	リモートフレーム(1)	RXFIFO(n)
8, (page=n×4)	拡張(EID)	0x0000000	データフレーム(0)	RXFIFO(n)
9, (page=n×4)	拡張(EID)	0x0000001	データフレーム(0)	RXFIFO(n)
10, (page=n×4)	拡張(EID)	0x0000002	データフレーム(0)	RXFIFO(n)
11, (page=n×4)	拡張(EID)	0x0000003	データフレーム(0)	RXFIFO(n)
12, (page=n×4)	拡張(EID)	0x0000000	リモートフレーム(1)	RXFIFO(n)
13, (page=n×4)	拡張(EID)	0x0000001	リモートフレーム(1)	RXFIFO(n)
14, (page=n×4)	拡張(EID)	0x0000002	リモートフレーム(1)	RXFIFO(n)
15, (page=n×4)	拡張(EID)	0x0000003	リモートフレーム(1)	RXFIFO(n)

・AFL（CANFD_B モジュール(RA6T2 等), CANFD_2 モジュール(RA8M1 等),CANFD-Lite モジュール(RX660 等)）

アクセプタンス フィルタ番号	フォーマット	ID	RTR （データフレーム・ リモートフレーム 区分）	受信先
0, (page=0)	標準(SID)	0x000	データフレーム(0)	RXFIFO(0)
1, (page=0)	標準(SID)	0x001	データフレーム(0)	RXFIFO(0)
2, (page=0)	標準(SID)	0x002	データフレーム(0)	RXFIFO(0)
3, (page=0)	標準(SID)	0x003	データフレーム(0)	RXFIFO(0)
4, (page=0)	標準(SID)	0x000	リモートフレーム(1)	RXFIFO(0)
5, (page=0)	標準(SID)	0x001	リモートフレーム(1)	RXFIFO(0)
6, (page=0)	標準(SID)	0x002	リモートフレーム(1)	RXFIFO(0)
7, (page=0)	標準(SID)	0x003	リモートフレーム(1)	RXFIFO(0)
8, (page=0)	拡張(EID)	0x0000000	データフレーム(0)	RXFIFO(0)
9, (page=0)	拡張(EID)	0x0000001	データフレーム(0)	RXFIFO(0)
10, (page=0)	拡張(EID)	0x0000002	データフレーム(0)	RXFIFO(0)
11, (page=0)	拡張(EID)	0x0000003	データフレーム(0)	RXFIFO(0)
12, (page=0)	拡張(EID)	0x0000000	リモートフレーム(1)	RXFIFO(0)
13, (page=0)	拡張(EID)	0x0000001	リモートフレーム(1)	RXFIFO(0)
14, (page=0)	拡張(EID)	0x0000002	リモートフレーム(1)	RXFIFO(0)
15, (page=0)	拡張(EID)	0x0000003	リモートフレーム(1)	RXFIFO(0)

ーサンプルプログラムのキーボードから入力するデータ送信コマンドー

コマンド	ID	DLC 送信バイト数	送信データ	CAN 使用メール ボックス番号	RSCAN CANFD_Lite CANFD_B CANFD_2 送信 FIFO	CANFD 送信 FIFO
0	0x0000000	1	0x01	24~27(FIFO)	SRFIFO(0) CFIFO(0)	CFIFO(n) n=0:CAN-ch0 n=3:CAN-ch1
1	0x0000001	2	0x0123			
2	0x0000002	4	0x01234567			
3	0x0000003	8	0x0123456789ABCDEF			
q	0x0000000	1				
w	0x0000001	2				
e	0x0000002	4				
r	0x0000003	8				

リモートフレーム送信コマンド(q,w,e,r)に関しても、送信元は SAMPLE2 と同じです。

7.4. サンプルプログラム(SAMPLE4)

SAMPLE4 は、CANFD を使用する設定です。

CANFD を使用するメリットとしては、
・データが高速に送信可能
CANFD は最初は、CAN の速度設定に基づき通信を開始します。データ送信時に、CANFD ビットレートに変更してデータ送信を行う方式です。CAN ビットレート(max 1Mbps)に対し、CANFD ビットレート(max 8Mbps)となり、高速な通信が可能です。
・1 回のデータ転送で多くのデータを送信可能
CAN は、1 つの CAN メッセージで最大 8 バイトのデータ送信となりますが、CANFD では最大 64 バイトとなります。
などが上げられます。

その一方で、CAN ネットワークを構成するデバイス（同一の CAN バスに接続されるデバイス）で、1 つでも CANFD 非対応のデバイスがつながっている場合は、CANFD のメッセージのやり取りが出来ません。（CANFD 非対応のネットワークに CANFD 対応デバイスを接続して、CAN メッセージのみを送信するように運用する事は可能。）

#define CANFD_MODE
定義時は CANFD を使用、未定義時は未使用となります。（SAMPLE4 では、can_operation2.h 内で本定数を定義しています。）

・CANFD_MODE 未設定時
CAN モード時（CANFD 未使用）は、CANFD の速度設定(DBRP, DTSEG1, DTSEG2, DSJW)は行わない。CAN の速度設定は、分周比(NBRP)を指定したビットレート値となる様に計算して算出します。

・CANFD_MODE 設定時
CANFD モード時は、CANFD(DBRP, DTSEG1, DTSEG2, DSJW)と CAN(NBRP, NTSEG1, NTSEG2, NSJW)の速度設定を行いますが、分周比は DBRP=NBRP=1（board.h で定義した値、本サンプルプログラムでは 1）となります。

※CANFD の速度設定パラメータ(DBRP, DTSEG1, DTSEG2, DSJW)は、テーブル参照方式で値を決めています。ビットレートが高速（概ね 5Mbps 以上）の場合、タイミングがシビアになりますので、ユーザ側で調整できる様にという考え方です。CAN の速度パラメータ(NBRP, NTSEG1, NTSEG2, NSJW)は、それ程シビアに調整する必要がないので、プログラムで値を計算する方式としています。
（can_general.c 内で値を決めています）

CAN（データ最大 8 バイト）に対し CANFD（データ最大 64 バイト）となりますので、メッセージバッファの数は CANFD 非対応の~SAMPLE3 から変更しているケースがあります。
（CAN のメッセージのやり取りで使用する専用の RAM の容量は決まっており、CAN と同じバッファ数を確保した場合、RAM が溢れてメッセージのデータ化けが生じることがあります。）

送信元、受信先に関しては、SAMPLE3 と同一です。

8. ヘッダファイルに関して

ヘッダファイルで、CAN の通信速度や送受信方式、どの CAN-ch, CAN 端子を使用するかなどを設定する事が可能です。

8.1. program_select.h

格納先:

RX: ¥SOURCE¥RX_CANST¥[ProjectFolder]¥src¥usr_src¥settings¥

RA: ¥SOURCE¥RA_CANST¥mcu¥[MCU Name]¥settings¥

```
/*----------------------------------------------------------------------
定数定義
----------------------------------------------------------------------*/

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
//#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム，CANFD版 ※CAN-FD対応マイコンのみ
//#define SAMPLE4

//MONITOR
//CANパケット送受信のモニタ用プログラム
//#define SAMPLE_MONITOR

//ーーここまで プログラム選択
```

CANFD 対応マイコンのみ SAMPLE4 が有効

ビルド対象とするサンプルプログラム(SAMPLEn)を選択する定義ファイルです。

#define SAMPLEn

の行を 1 つのみ、コメントアウト(//)を外して有効化してください。

RX660, RX261, RX26T, RA8E1, RA8M1, RA8T1, RA6M5, RA6T2, RA6T3, RA6E2, RA4T1, RA4E2 マイコン（CANFD 対応）の場合のみ、SAMPLE4 が有効です。その他のマイコンの場合は、SAMPLE4 は選択してもエラーとなります。

8.2. board.h

格納先:

RX: ¥SOURCE¥RX_CANST¥[ProjectFolder]¥src¥usr_src¥settings¥　　[ProjectFolder]:RX72M_176 等

RA: ¥SOURCE¥RA_CANST¥mcu¥[MCU Name]¥settings¥　　[MCU Name]:RA6M5 等

```
#ifndef BOARD_H__
#define BOARD_H__

/*----------------------------------------------------------------------
インクルード
----------------------------------------------------------------------*/
#include "board_setting.h"

/*----------------------------------------------------------------------
定数定義
----------------------------------------------------------------------*/
//ボードタイプ選択
#define BOARD_TYPE HSBRX651F176
//#define BOARD_TYPE HSBRX65N176

//LEDのポート設定
#define LED_PORT_NUM  1
#define LED1_PORT_PDR  PORTA.PDR.BIT.B1
#define LED1_PORT  PORTA.PODR.BIT.B1

#define LED_ON  1
#define LED_OFF  0

//SWのポート設定
#define SW_PORT_NUM  1
#define SW1_PORT_PDR  PORTA.PDR.BIT.B2
#define SW1_PORT  PORTA.PIDR.BIT.B2

#define CAN_MACRO_TYPE  CAN_MACRO_TYPE_CAN
#define CAN_CH  2

#define CAN0_VALID  1//CAN0有効
#define CAN1_VALID  0//CAN1無効

#define ICLK  120
#define PCLKB  60
#define XTAL  24
#define CAN_CLOCK  PCLKB

//CANのクロックソース
#define CAN_CLOCK_SOURCE    CAN_CLOCK_PCLK//PCLK(B)をCANクロックとして使用

#define MCU_TYPE  MCU_TYPE_RX65_2M

#define PIN_NO  176
```

例示しているファイルは RX のものです

使用するマイコンボードを選択する(どちらかを有効にする)
※選択するようになっていないボードもあります

LED や SW の設定

どのタイプの CAN マクロ搭載しているかの定義
※搭載マイコンで決まるので変更不可

マイコンが持っている CAN の ch 数

CAN-ch0 のみ使用する設定(*1)

CAN1_VALID 1 とすれば CAN-ch1 を有効化できます

CAN_CLCOK には CAN のクロックソース MHz 単位を指定(*2)

CAN のクロックソース選択(*3)

マイコンの種別

```
//CAN 端子設定（使用端子のコメントアウトを外す）

//CAN-ch0
#define CAN0_TX_P32
//#define CAN0_TX_PD1

#define CAN0_RX_P33
//#define CAN0_RX_PD2

//CAN-ch1
//#define CAN1_TX_P14
//#define CAN1_TX_P54

//#define CAN1_RX_P15
//#define CAN1_RX_P55

//割り込みモニタ端子設定

//エラー割り込み PF4 でモニタ
//(PF4->J1-3)
#define CAN_ERS_PORT_PDR  PORTF.PDR.BIT.B4
#define CAN_ERS_PORT_PODR  PORTF.PODR.BIT.B4

//CAN0 受信FIFO割り込み PF3 でモニタ
//(PF3->J1-4)
#define CAN0_RXF0_PORT_PDR  PORTF.PDR.BIT.B3
#define CAN0_RXF0_PORT_PODR  PORTF.PODR.BIT.B3

//CAN0 送信FIFO割り込み PF2 でモニタ
//(PF2->J1-5)
#define CAN0_TXF0_PORT_PDR  PORTF.PDR.BIT.B2
#define CAN0_TXF0_PORT_PODR  PORTF.PODR.BIT.B2

//CAN0 メールボックス受信割り込み PF1 でモニタ
//(PF1->J1-6)
#define CAN0_RXM0_PORT_PDR  PORTF.PDR.BIT.B1
#define CAN0_RXM0_PORT_PODR  PORTF.PODR.BIT.B1

//CAN0 メールボックス送信割り込み PJ3 でモニタ
//(PJ3->J1-7)
#define CAN0_TXM0_PORT_PDR  PORTJ.PDR.BIT.B3
#define CAN0_TXM0_PORT_PODR  PORTJ.PODR.BIT.B3

//CAN1 受信FIFO割り込み PJ5 でモニタ
//(PJ5->J1-8)
#define CAN1_RXF1_PORT_PDR  PORTJ.PDR.BIT.B5
#define CAN1_RXF1_PORT_PODR  PORTJ.PODR.BIT.B5

//CAN1 送信FIFO割り込み PF5 でモニタ
//(PF5->J1-9)
#define CAN1_TXF1_PORT_PDR  PORTF.PDR.BIT.B5
#define CAN1_TXF1_PORT_PODR  PORTF.PODR.BIT.B5

//CAN1 メールボックス受信割り込み P00 でモニタ
//(P00->J1-10)
#define CAN1_RXM1_PORT_PDR  PORT0.PDR.BIT.B0
#define CAN1_RXM1_PORT_PODR  PORT0.PODR.BIT.B0

//CAN1 メールボックス送信割り込み P01 でモニタ
//(P01->J1-11)
#define CAN1_TXM1_PORT_PDR  PORT0.PDR.BIT.B1
#define CAN1_TXM1_PORT_PODR  PORT0.PODR.BIT.B1
```

CTXD0, CRXD0 で使用する端子のコメントアウトを外します(*4)

CAN-ch1 を使用する際は、使用端子のコメントアウトを外してください

割り込み発生時に反転させるポート設定（デバッグ向け）

```
//CANの速度設定をマニュアルで行う場合（デフォルト未使用）
//#define CAN_SPEED_PARAM_MANUAL_SETTING    //定義時はCANの速度設定レジスタに直接値を代入する

//-> 1Mbps設定 : CANベースクロック20MHz(=PCLKB/3), 1ビットタイム=20Tq, 50ns x 20 = 1us (1Mbps), サンプリングポイント=75%
//SS(1) TSEG1(14) TSEG2(5)
#define CAN_BCR_BRP         (3-1)   //分周比:3
#define CAN_BCR_TSEG1       (14-1)  //TSEG1=14
#define CAN_BCR_TSEG2       (5-1)   //TSEG2=5
#define CAN_BCR_SJW         (1-1)   //SJW=1
//CAN_SPEED_PARAM_MANUAL_SETTING 未定義時は、can_operation.h 内の速度設定値を使用して各パラメータを計算で求める

#endif
```

CAN の速度パラメータ設定値(*5)
（#define CAN_SPEED_PARAM_MANUAL_SETTING
のコメントアウトを外した際に、この値が使用されます
通常は、can_operation.h 内の速度定義値を基に計算された値が使用されます）

board.h には、マイコンボード固有の定義が記載されています。
（ボードに搭載している LED のポート名や、CAN で使用するポート、クロック周波数など）
当社製のマイコンボードを使用する際には、基本的には変更の必要はありません。外付けの CAN ドライバ回路を追加する場合や、当社製以外のボードを使用する際には、変更の必要があります。重要な設定を以下に示します。

(*1)有効化する CAN の ch 設定
#define CAN0_VALID　1
#define CAN1_VALID　0
とすると、CAN-ch0 は有効となり、CAN-ch1 は未使用の設定となります。

(*2)CAN のクロック周波数の設定
#define CAN_CLOCK 60
CAN のクロックソースとして、60MHz のクロックを入力する場合。この値と、can_operation.h（後述）内の速度設定値を使用して、CAN の速度パラメータ（クロック分周比(BPR), TSEG1, TSEG2）の値が計算されます。CAN マクロに入力される、クロック周波数の値を MHz 単位で指定します。

－CANFD 対応マイコンに関して－

```
//CLK[MHz]
#define ICLK 120
#define PCLKB 60
#define XTAL 24
#define CANFD_CLOCK     60              //CANFDクロック 60MHz
#define CAN_CLOCK CANFD_CLOCK //CANFDCLKをCANクロックとして使用
#define CANFD_CK_DIV_BASE   1 //CANFD:CANCLOCK/1 を CAN クロックに設定
```

CANFD 対応マイコンでは、クロックの分周比 CANFD_CK_DIV_BASE を定義しています。（CANFD 有効時：この値がビットレート調整レジスタの分周比(BRP)として設定されます。）

CANFD パケット取り扱い時、CAN ビットレート調整の分周比と、CANFD ビットレート調整の分周比は同じ値とする事が求められています。CANFD 使用時は、この値が CAN と CANFD のビットレート調整の分周比に設定されます。

CANFDCLK=80MHz 時は、分周比=1 の場合は、

1/80MHz = 12.5ns(1Tq)

1bit 最大 384Tq : 384 x 12.5ns = 4.8us, 1 / 4.8us = 208.3kbps

一番遅いビットレートとして、208kbps となります。(125kbps には設定不可)

CANFD パケットを取り扱う場合で、遅いビットレートを使用する際は、この分周比の値を調整してください。

※CANFD パケットを取り扱わない場合(SAMPLE1~SAMPLE3 など)、CANFD_MODE 定数が未定義の場合は、この分周比設定は使用されず、分周比を含めて計算値が設定されるので、CANFD_CK_DIV_BASE 値は使用されません(CANFD_CK_DIV_BASE を変更しなくても、208kbps 以下の遅いビットレート設定が可能です。)

(*3)CAN のクロックソース選択

CANFD 対応マイコンは、CAN のクロックとして CANFDCLK として独立したクロックが用意されています。

#define CAN_CLOCK_SOURCE　　CAN_CLOCK_CANFDCLK //CANFDCLK を選択 [通常はこちらを選択]

#define CAN_CLOCK_SOURCE　　CAN_CLOCK_XTAL //XTAL を選択

CAN のクロックソースとして、XTAL を選択することも可能ですが、通常は CANFDCLK を選択してください。

その他のマイコンは、PCLKB(もしくは PCLKB/2)か XTAL(外付け水晶振動子のクロック)のどちらかを選択する形となります。

#define CAN_CLOCK_SOURCE　　CAN_CLOCK_PCLK //PCLKB(もしくは PCLKB/2)を選択

#define CAN_CLOCK_SOURCE　　CAN_CLOCK_XTAL //XTAL を選択

PCLKB が、HOCO(マイコン内蔵オシレータ)から生成されている場合は、XTAL 側を選んでください。また、PCLKB を分周しても目的のビットレートに調整できない場合等も考えられますので、必要に応じてクロックソースを選択してください。

(RX231 は最大動作周波数 54MHz で動かした際、PCLKB=27MHz となり 1Mbps には設定できない。)

(基本的には、XTAL < PCLKB となるケースがほとんどですので、高速な PCLKB を選んだ方が調整可能な速度の幅が広がるなどのメリットがあります。)PCLKB 側を選んだ場合、CAN のクロックソースが PCLKB/2 になるマイコンもあります。マイコンのハードウェアマニュアルで、CAN のクロックソースが PCLKB となるか PCLKB/2 となるかは確認してください。PCLKB/2 となるマイコンでは、(*2)の設定値は、PCLKB/2 の周波数値を設定してください。

(*4)使用 CAN ポートの設定

どの CTX, CRX 端子を使用するかは、複数の端子から選べる様になっています。

#define CAN0_TX_P32

#define CAN0_RX_P33

上記の様に定義すると、CAN-ch0 の CTX は P32、CAN-ch0 の CRX は P33 を使用する設定となります。

(*5)CAN の速度パラメータ設定

デフォルトでは、

//#define CAN_SPEED_PARAM_MANUAL_SETTING

上記文はコメントアウトされているため、board.h 内の速度パラメータ

BRP:クロック分周比

TSEG1, TSEG2:1bit を何 Tq で構成するか

SJW:再同期ジャンプ幅

の定義は使用されません。これらのパラメータを自分で定義したい場合は、

#define CAN_SPEED_PARAM_MANUAL_SETTING

のコメントアウトを外してください。

```
//CANの速度設定をマニュアルで行う場合（デフォルト未使用）
//#define CAN_SPEED_PARAM_MANUAL_SETTING    //定義時はCANの速度設定レジスタに直接値を代入する

//-> 1Mbps設定 : CANベースクロック20MHz(=PCLKB/3), 1ビットタイム=20Tq, 50ns x 20 = 1us (1Mbps), サンプリングポイント=75%
//SS(1) TSEG1(14) TSEG2(5)
#define CAN_BCR_BRP         (3-1)   //分周比:3
#define CAN_BCR_TSEG1       (14-1)  //TSEG1=14
#define CAN_BCR_TSEG2       (5-1)   //TSEG2=5
#define CAN_BCR_SJW         (1-1)   //SJW=1
//CAN_SPEED_PARAM_MANUAL_SETTING 未定義時は、can_operation.h 内の速度設定値を使用して各パラメータを計算で求める

#endif
```

CANFD 対応マイコンの場合は、2 種類の速度パラメータ（最初に使用される通信レートとデータ送信の際に使用される通信レート）の定義があります。

```
//CANの速度設定をマニュアルで行う場合（デフォルト未使用）
//#define CAN_SPEED_PARAM_MANUAL_SETTING    //定義時はCANの速度設定レジスタに直接値を代入する

//CANの速度設定パラメータ
//-> 1Mbps設定 : CANベースクロック60MHz(=CANFDCLK/1), 1ビットタイム=60Tq, 16.7ns x 60 = 1us (1Mbps), サンプリングポイント=75%
//SS(1) TSEG1(44) TSEG2(15)
#define CAN_BCR_BRP         (1-1)   //分周比:1
#define CAN_BCR_TSEG1       (44-1)  //TSEG1=44
#define CAN_BCR_TSEG2       (15-1)  //TSEG2=15
#define CAN_BCR_SJW         (3-1)   //SJW=3
//CAN_SPEED_PARAM_MANUAL_SETTING 未定義時は、can_operation.h 内の速度設定値を使用して各パラメータを計算で求める

//CANFDの速度設定パラメータ
//-> 2Mbps設定 : CANベースクロック60MHz(=CANFDCLK/1), 1ビットタイム=30Tq, 33.3ns x 30 = 0.5us (2Mbps), サンプリングポイント=76.7%
//SS(1) TSEG1(22) TSEG2(7)
#define CANFD_BCR_BRP       (1-1)   //分周比:1
#define CANFD_BCR_TSEG1     (22-1)  //TSEG1=22
#define CANFD_BCR_TSEG2     (7-1)   //TSEG2=7
#define CANFD_BCR_SJW       (1-1)   //SJW=1
//CANFD_SPEED_PARAM_MANUAL_SETTING 未定義はキーボードから入力した速度設定値を使用してテーブルで各パラメータを設定
```

最初に使用される通信レート（公称レート）の設定パラメータ

データ送信時の通信レートの設定パラメータ

基本的には、CAN_BCR_BRP と CANFD_BCR_BRP の値は同一値としてください。

board.h は、マイコンのボードに依存する定義を行っているヘッダファイルです。(CD に含まれているものは、ボードに合わせて設定済みですので、基本的には CD からコピー後に変更する必要はありません。外付けの CAN ドライバボードを接続して、使用 CAN ポートを増やす際などに編集してください。)

8.3. board_setting.h

格納先:

RX: ¥SOURCE¥RX_CANST¥source¥ALL¥common¥

RA: ¥SOURCE¥RA_CANST¥source¥ALL¥common¥

例示しているファイルは RX のものです

```
#ifndef BOARD_SETTING_H__
#define BOARD_SETTING_H__

/*-----------------------------------------------------------------------
定数定義
-----------------------------------------------------------------------*/

//定数値定義

//マイコンボードタイプ定義 [以下のうちいずれかを選択]
//RX71M
#define HSBRX71M176 1
#define HSBRX71M100 2

 (中略)
//CANマクロ種別
#define CAN_MACRO_TYPE_CAN        1 //メールボックスを使用するタイプ
#define CAN_MACRO_TYPE_RSCAN      2 //RX200系など
#define CAN_MACRO_TYPE_CANFD      3 //RA6M5
#define CAN_MACRO_TYPE_CANFD_B    4 //RA6T2など
#define CAN_MACRO_TYPE_CANFD_LITE 5 //RX660,RX26Tなど
#define CAN_MACRO_TYPE_CANFD_2    6 //RA6M1,RA8T1など

//MCUタイプ
#define MCU_TYPE_RX71M   1
#define MCU_TYPE_RX65_2M 2
#define MCU_TYPE_RX65_1M 3
#define MCU_TYPE_RX64M   4

 (後略)
```

マイコンボード名、CAN マクロのタイプ、マイコン種などの名称が定義されているファイルです。基本的には修正の必要はありません。

8.4. can_common.h

格納先:

RX: ¥SOURCE¥RX_CANST¥source¥ALL¥common¥

RA: ¥SOURCE¥RA_CANST¥source¥ALL¥common¥

```
#ifndef CAN_COMMON_H__
#define CAN_COMMON_H__

/*----------------------------------------------------------------------
    インクルード
----------------------------------------------------------------------*/
#include "board.h"

/*----------------------------------------------------------------------
    定数定義
----------------------------------------------------------------------*/
//通信速度, [kbps]単位
#define CAN_BPS_1M                  1000    //1M bps
#define CAN_BPS_500K                500     //500k bps
#define CAN_BPS_250K                250     //250k bps
#define CAN_BPS_125K                125     //125k bps

//通信速度（データ）, [kbps]単位
#define CANFD_BPS_12M               12000   //12Mbps, テスト用
#define CANFD_BPS_10M               10000   //10Mbps, テスト用
#define CANFD_BPS_8M                8000    //8M bps
#define CANFD_BPS_7_5M              7500    //7.5M bps
#define CANFD_BPS_6_6M              6667    //6.6M bps
#define CANFD_BPS_6M                6000    //6M bps
#define CANFD_BPS_5M                5000    //5M bps
#define CANFD_BPS_4M                4000    //4M bps
#define CANFD_BPS_3_3M              3333    //3.3M bps
#define CANFD_BPS_3M                3000    //3M bps
#define CANFD_BPS_2M                2000    //2M bps

//パラメータ有効/無効設定
#define ENABLE                      1
#define DISABLE                     0

//CANFD 第2サンプルポイント設定
#define DELAY_OFFSET                0       //測定値＋オフセット値
#define OFFSET_ONLY                 1       //オフセット値のみ

//ID
#define CAN_ID_FORMAT_SID           0       //標準ID(11bit)フォーマットを使用
#define CAN_ID_FORMAT_EID           1       //拡張ID(29bit)フォーマットを使用
#define CAN_SID_MASK                0x7FF
#define CAN_EID_MASK                0x1FFFFFFF
#define CAN_EID2_MASK               0x3FFFF

//RTR
#define CAN_DATA_FRAME              0
#define CAN_REMOTE_FRAME            1

//FDF
#define CAN_DATA_FORMAT             0       //CANデータフォーマット
#define CANFD_DATA_FORMAT           1       //CANFDデータフォーマット

//BRS
#define CAN_BITRATE                 0       //CANビットレート
#define CANFD_BITRATE               1       //CANFDビットレート
```

CAN の速度値設定

CANFD の速度値設定

```
//ESI
#define CANFD_ERROR_ACTIVE          0       //エラーアクティブ
#define CANFD_ERROR_PASSIVE         1       //エラーパッシブ

//クロックソース
#define CAN_CLOCK_PCLK              0
#define CAN_CLOCK_CANFDCLK          0
#define CAN_CLOCK_XTAL              1

//受信ルール
#define CAN_RULE_RXBUF              0x0     //受信バッファで受信
#define CAN_RULE_RXFIFO0            0x0001  //RXFIFO0で受信
#define CAN_RULE_RXFIFO1            0x0002  //RXFIFO1で受信
#define CAN_RULE_RXFIFO2            0x0004  //RXFIFO2で受信
#define CAN_RULE_RXFIFO3            0x0008  //RXFIFO3で受信
#define CAN_RULE_RXFIFO4            0x0010  //RXFIFO4で受信
#define CAN_RULE_RXFIFO5            0x0020  //RXFIFO5で受信
#define CAN_RULE_RXFIFO6            0x0040  //RXFIFO6で受信
#define CAN_RULE_RXFIFO7            0x0080  //RXFIFO7で受信
#define CAN_RULE_CFIFO0             0x0100  //CFIFO0で受信
#define CAN_RULE_CFIFO1             0x0200  //CFIFO1で受信
#define CAN_RULE_CFIFO2             0x0400  //CFIFO2で受信
#define CAN_RULE_CFIFO3             0x0800  //CFIFO3で受信
#define CAN_RULE_CFIFO4             0x1000  //CFIFO4で受信
#define CAN_RULE_CFIFO5             0x2000  //CFIFO5で受信
#define CAN_RULE_SRFIFO0            0x0010  //SRFIFO0で受信

//CANの受信方法
#define CAN_RX_RXBUF                0x01
#define CAN_RX_RXFIFO               0x02
#define CAN_RX_CFIFO                0x04
#define CAN_RX_SRFIFO               0x08
#define CAN_RX_MBOX                 0x10
#define CAN_RX_MBOXFIFO             0x20

//CANの送信方法
#define CAN_TX_TXBUF                0x01
#define CAN_TX_TXQUEUE              0x02
#define CAN_TX_CFIFO                0x04
#define CAN_TX_SRFIFO               0x08
#define CAN_TX_MBOX                 0x10
#define CAN_TX_MBOXFIFO             0x20

//CAN-ch
#define CAN_CH0                     0
#define CAN_CH1                     1
#define CAN_CH2                     2

//送信完了フラグ
#define CAN_TX_FLAG_CFIFO               0x0100      //共通FIFOで送信完了
#define CAN_TX_FLAG_SRFIFO              0x0100      //送受信FIFOで送信完了（共通FIFOと送受信FIFO
は同じ番号を割り振る）
#define CAN_TX_FLAG_TXBUF               0x0200      //送信バッファで送信完了
#define CAN_TX_FLAG_TXBUF_WITH_ABORT    0x0300      //送信バッファで with abort
#define CAN_TX_FLAG_TXBUF_ABORTED       0x0400      //送信バッファで aboated
#define CAN_TX_FLAG_TXQUEUE             0x0500      //送信キューで送信完了
#define CAN_TX_FLAG_MBOX                0x0600      //送信メールボックスで送信完了
#define CAN_TX_FLAG_MBOX_WITH_ABORT     0x0700      //送信メールボックスで送信アボート
#define CAN_TX_FLAG_MBOX_REG_TIMEOUT    0x0800      //送信メールボックスでレジスタ書き換えタイム
アウト
#define CAN_TX_FLAG_MBOXFIFO            0x0900      //送信メールボックスFIFOで送信完了
```

```
//戻り値
#define CAN_RET_SUCCESS             0       //成功
#define CAN_RET_NODATA             -1       //データなし
#define CAN_RET_ARGUMENT_ERROR      -2       //引数エラー
#define CAN_RET_IN_USE             -3       //使用中エラー
#define CAN_RET_OVERFLOW           -4       //FIFOフルエラー
#define CAN_RET_MODE_ERROR         -5       //関数が実行可能なモードではない
#define CAN_RET_VALUE_ERROR        -6       //値が不正
#define CAN_RET_TIMEOUT            -7       //タイムアウト
#define CAN_RETFLAG_LOST_DATA      0x80     //失われたデータあり（戻り値とorを取る）

/*-----------------------------------------------------------------------
    構造体
-----------------------------------------------------------------------*/
//CANメッセージ構造体
typedef struct{
    unsigned long id;
    unsigned char rtr;
    unsigned char ide;
    unsigned char dlc;
    unsigned char data[8];
    unsigned short ts;
} can_message;

//CANFDメッセージ構造体
typedef struct{
    unsigned long id;
    unsigned char rtr;
    unsigned char ide;
    unsigned char fdf;
    unsigned char brs;
    unsigned char esi;
    unsigned char dlc;
    unsigned char data[64];
    unsigned short ts;
} canfd_message;

//CANFD設定構造体
typedef struct{
    unsigned short speed;   //CANFD 通信速度[kbps]
    unsigned char tdce;     //CANFD トランシーバ遅延補償
    unsigned char ssp;      //CANFD 第2サンプリングポイント 測定値+オフセット
    unsigned char tdco;     //CANFD トランシーバ遅延補償 オフセット値
    unsigned char refe;     //RXエッジフィルタ
} canfd_param;

#endif
```

CAN のメッセージ構造体

CANFD のメッセージ構造体

CANFD 設定構造体

プログラムで使用する定数や構造体を定義しているファイルです。

8.5. can_operation.h

格納先:

RX: ¥SOURCE¥RX_CANST¥source¥ALL¥common¥

RA: ¥SOURCE¥RA_CANST¥source¥ALL¥common¥

```
#ifndef CAN_OPERATION_H__
#define CAN_OPERATION_H__

/*----------------------------------------------------------------------
    インクルード
----------------------------------------------------------------------*/
#include    "can_common.h"
#include    "program_select.h"

#if defined(SAMPLE1) || defined(SAMPLE2) || defined(SAMPLE3) || defined(SAMPLE4)
#include    "can_operation2.h"
/*
 * サンプルプログラムでは、SAMPLEnで受信方法や割り込み使用有無を変更するので、SAMPLE依存の定義は
can_operation2.hに追い出す
 */
#endif

/*----------------------------------------------------------------------
    定数定義
----------------------------------------------------------------------*/
//通信速度
#define CAN_SPEED       CAN_BPS_1M

//下記を設定
//CAN_BPS_1M(1000)      //1M bps
//CAN_BPS_500K(500)     //500k bps
//CAN_BPS_250K(250)     //250k bps
//CAN_BPS_125K(125)     //125k bps ※CANFDCLK=80MHz時 かつCANFD動作時(CANFD_MODE定義時)指定不可
//クラシカルCAN動作時（CANFD_MODE未定義時）
//→1000の1/n(n:整数)の数値のみ指定可能
//
//CANFD動作時（CANFD_MODE定義時）
//#define CAN_SPEED 400  （400kbpsの意）の様に任意の速度を指定する事も可能
//任意の値が指定可能だが、1Tqの倍数とならない場合は速度誤差が生じる
//[例]CANクロック40MHzの場合は、1Tq=25ns
//1Tq=25ns時の指定可能な下限は、104kbps程度（100kbpsは指定不可）
//→1Tq=25nsを大きな値とすれば、100kbps未満の値も指定可能となる

//通信速度（データ） ※CANFD対応マイコンのみ CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_SPEED     CANFD_BPS_2M

//いずれかを設定
//CANFD_BPS_8M(8000)    //8Mbps   ※要CANトランシーバ回路外付け
//CANFD_BPS_7_5M(7500)  //7.5Mbps ※要CANトランシーバ回路外付け
//CANFD_BPS_6_6M(6600)  //6.6Mbps ※要CANトランシーバ回路外付け
//CANFD_BPS_6M(6000)    //6Mbps   ※要CANトランシーバ回路外付け
//CANFD_BPS_5M(5000)    //5Mbps
//CANFD_BPS_4M(4000)    //4Mbps
//CANFD_BPS_3_3M(3300)  //3.3Mbps
//CANFD_BPS_3M(3000)    //3Mbps
//CANFD_BPS_2M(2000)    //2Mbps
//CANFD_BPS_1M(1000)    //1Mbps
//※CANFD_SPEEDは任意の値設定不可（タイミングパラメータはテーブル指定のため、定義値のみ指定可能）

//CANFD トランシーバ遅延補償 ※CANFD対応マイコンのみ CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_TDCE      DISABLE

//CANFD 第2サンプルポイント ※CANFD対応マイコンのみ CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_SSP       DELAY_OFFSET
```

SAMPLE1~SAMPLE4 は、サンプルプログラム毎に割り込みや送受信方法を変更しているので、SAMPLE1~SAMPLE4 の動作は、can_operation2.h 側で定義する

CAN の速度値設定(*1)

CANFD の速度値設定(*2)

CANFD のパラメータ設定(*3)

```
//CANFD トランシーバ遅延補償 オフセット値 ※CANFD対応マイコンのみ CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_TDCO      0

//CANFD トランシーバRXエッジフィルタ ※CANFD対応マイコンのみ CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_REFE      DISABLE

//ID設定
#define CAN_ID_TYPE     CAN_ID_FORMAT_EID

//いずれかを設定
//CAN_ID_FORMAT_SID     //標準ID(11bit)フォーマットを使用
//CAN_ID_FORMAT_EID     //拡張ID(29bit)フォーマットを使用

//レジスタ書き換えウェイト（CAN_MACRO_TYPE_CANタイプのマイコンで使用）
#define CAN_ABORT_WAIT          750e-6/((1.0/(ICLK*1e6))*11)    //約750usのウェイト（11命令/loopで計算）
#define CAN_LOOP_TIMEOUT        1000                            //レジスタ書き換えタイムアウト（回数）

//デバッグ用
//#define SCI_DEBUG                     //定義時SCI表示によるデバッグを行う
#define CAN_INTERRUPT_DEBUG             //定義時割り込みを端子でモニタする

//受信バッファサイズ
#define CAN_RECV_BUF_SIZE         16    //can_message構造体のバッファ数のメモリを消費

//割り込み優先度，RA(RA_INTR_TYPE1):0-15を指定（0:最高優先度，15:最低優先度），RA(RA_INTR_TYPE2):0-3を指定（0:最高優先度，3:最低優先度），RX:1-15を指定（15:最高優先度，1:最低優先度）
#define CAN_RX_INTERRUPT_PRIORITY       14  //RXFIFO/CFIFO/SRFIFO/受信メールボックスの受信割り込み優先度
#define CAN_TX_INTERRUPT_PRIORITY       14  //送信割り込み優先度
#define CAN_ERROR_INTERRUPT_PRIORITY    14  //エラー割り込み優先度
#define CAN_INTERRUPT_PRIORITY      14  //共通化されている場合の割り込み優先度

//送受信方法の選択,割り込みの使用（SAMPLEプログラム実行時はSAMPLEnに応じて選択）
#if defined(SAMPLE1) || defined(SAMPLE2) || defined(SAMPLE3) || defined(SAMPLE4)
/*
 * SAMPLEnの動作を変える場合は
 * can_operation2.h 内の定義を変更
 */
#else

//CAN, RX600, RX700, RA6M2, RA2L1などCANマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CAN)

//CANの受信方法（いずれか）
#define CAN_RX_METHOD       (CAN_RX_MBOXFIFO | CAN_RX_MBOX)     //メールボックス(FIFO)+メールボックスを使用（デフォルト）(*1)
//#define CAN_RX_METHOD       CAN_RX_MBOX                     //メールボックスを使用
//#define CAN_RX_METHOD       CAN_RX_MBOXFIFO                 //メールボックス(FIFO)を使用(*2)

//CANの送信方法（いずれか）
#define CAN_TX_METHOD       CAN_TX_MBOXFIFO         //メールボックス(FIFO)を使用
//#define CAN_TX_METHOD     CAN_TX_MBOX             //メールボックスを使用

#define CAN_RX_INTERRUPT    ENABLE                  //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                  //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                  //エラー割り込み

//(*1)拡張IDはFIFOで受信、標準IDはメールボックスで受信
//(*2)メールボックスFIFOを選択した場合は、拡張IDのみ受信

#endif //CAN_MACRO_TYPE
```

ID 設定(*4)
CAN_ID_FORMAT_SID を選択すると、標準 ID のみ取り扱います
CAN_ID_FORMAT_EID を選択すると、拡張 ID と標準 ID の両方を取り扱います

デバッグ設定(*5)
定義時は、SCI(UART)や端子デバッグが有効になります

受信リングバッファのサイズ

割り込み優先度(*6)
RX では 14, RA では 1 を設定しています

SAMPLE1~SAMPLE4 では、以下の設定は使用しません
（can_operation2.h 内の設定を使用）

CAN マクロタイプ毎の定義
CAN マクロの場合

CAN の受信方法の選択(*7)

CAN の送信方法の選択(*8)

CAN の割り込みの有効化(*9)

```
//RSCAN, RX200などRSCANマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_RSCAN)

//CANの受信方法（いずれか）
//#define CAN_RX_METHOD     CAN_RX_RXBUF          //受信バッファを使用
#define CAN_RX_METHOD       CAN_RX_RXFIFO         //受信FIFOを使用
//#define CAN_RX_METHOD     CAN_RX_SRFIFO         //送受信FIFOを使用（*1）

//CANの送信方法（いずれか）
//#define CAN_TX_METHOD     CAN_TX_TXBUF          //送信バッファを使用
#define CAN_TX_METHOD       CAN_TX_SRFIFO         //送受信FIFOを使用（*1）

//(*1)送受信FIFOは受信or送信のどちらか一方（送信&受信をSRFIFOで行う事はできない）

#define CAN_RX_INTERRUPT    ENABLE                //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                //エラー割り込み

#endif //CAN_MACRO_TYPE

//CAN-FD, RX660, RX26T, RA6M5, RA6T2, RA8M1などCAN-FDマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD) || (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_B)
|| (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_LITE) || (CAN_MACRO_TYPE ==
CAN_MACRO_TYPE_CANFD_2)

//CANの受信方法（いずれか）
//#define CAN_RX_METHOD     CAN_RX_RXBUF          //受信バッファを使用
#define CAN_RX_METHOD       CAN_RX_RXFIFO         //受信FIFOを使用
//#define CAN_RX_METHOD     CAN_RX_CFIFO          //共通FIFOを使用（*2）

//CANの送信方法（いずれか）
//#define CAN_TX_METHOD     CAN_TX_TXBUF          //送信バッファを使用
//#define CAN_TX_METHOD     CAN_TX_TXQUEUE        //送信キューを使用
#define CAN_TX_METHOD       CAN_TX_CFIFO          //共通FIFOを使用（*2）

//(*2)CANFD_B, CANFD_LITE, CANFD_2では共通FIFOは受信or送信のどちらか一方（送信&受信をCFIFOで行う
事はできない）

#define CAN_RX_INTERRUPT    ENABLE                //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                //エラー割り込み

#define CANFD_MODE                                //CANFD対応マイコンではデフォルトCANFDモードと
する※CANFD未使用時はコメントアウト

#endif //CAN_MACRO_TYPE

#endif //SAMPLEn

#if (CAN_MACRO_TYPE != CAN_MACRO_TYPE_CANFD)      //CANFDはCFIFO 6本なので送信と受信の両方に使
用可能
#if (CAN_TX_METHOD == CAN_TX_CFIFO) && (CAN_RX_METHOD == CAN_RX_CFIFO)
#error "Error: CFIFO select error"
/*
 * CFIFOは1本（CFIFO(0)のみ）なので、「送信」「受信」のどちらかで使用可能（送信と受信のどちらかを
CFIFO以外の方法としてください）
 */
#endif
#endif
```

CAN マクロタイプ毎の定義
RSCAN マクロの場合

CAN の受信方法の選択(*10)

CAN の受信方法の選択(*11)

CAN の割り込みの有効化(*9)

CAN マクロタイプ毎の定義
CANFD 系マクロの場合

CAN の受信方法の選択(*12)

CAN の受信方法の選択(*13)

CAN の割り込みの有効化(*9)

CANFD の有効化(*14)
※CANFD 対応マイコンでのみ有効

SAMPLE1~SAMPLE4 時は未使用の範囲終了

以下は、SAMPLE1~SAMPLE4 でも使用される

定義がエラーの場合のトラップ

```
#if (CAN_TX_METHOD == CAN_TX_SRFIFO) && (CAN_RX_METHOD == CAN_RX_SRFIFO)
#error "Error: SRFIFO select error"
/*
 * SRFIFOは1本なので、「送信」「受信」のどちらかで使用可能（送信と受信のどちらかをSRFIFO以外の方法としてください）
 */
#endif

//FIFO番号
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_RSCAN)

#define CAN0_RX_RXFIFO_NO   0       //CAN0の受信をRXFIFO0で行う （0~1の値を設定） [CAN_RX_METHOD -> CAN_RX_RXFIFO 指定時に使用
#define CAN0_RX_SRFIFO_NO   0       //CAN0の受信をSRFIFO0で行う （0のみ） [CAN_RX_METHOD -> CAN_RX_SRFIFO 指定時に使用](*1)(*2)
#define CAN0_TX_SRFIFO_NO   0       //CAN0の送信をSRFIFO0で行う （0のみ） [CAN_TX_METHOD -> CAN_TX_SRFIFO 指定時に使用](*1)(*2)

//(*1)SRFIFO0は、送信または受信での排他利用
//(*2) 0のみ指定可能（SRFIFO/CFIFOを複数使用できるタイプのマイコンと関数仕様を共通化するため、SRFIFO番号を指定する様にしているため）

#elif (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_B) || (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_LITE) || (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_2)

//RXFIFO[0] ~ RXFIFO[1]
//CFIFO[0]

#define CAN0_RX_RXFIFO_NO   0       //CAN0の受信をRXFIFO0で行う （0~1の値を設定） [CAN_RX_METHOD -> CAN_RX_RXFIFO 指定時に使用
#define CAN0_RX_CFIFO_NO    0       //CAN0の受信をCFIFO0で行う （0のみ） [CAN_RX_METHOD -> CAN_RX_CFIFO 指定時に使用](*1)(*2)
#define CAN0_TX_CFIFO_NO    0       //CAN0の送信をCFIFO0で行う （0のみ） [CAN_TX_METHOD -> CAN_TX_CFIFO 指定時に使用](*1)(*2)
#define CAN0_TX_TXQUEUE_NO  0       //CAN0の送信をTXQUEUE0で行う （0のみ） [CAN_TX_METHOD -> CAN_TX_TXQUEUE 指定時に使用](*2)

#define CAN1_RX_RXFIFO_NO   0       //CAN1の受信をRXFIFO0で行う （0~1の値を設定） [CAN_RX_METHOD -> CAN_RX_RXFIFO 指定時に使用
#define CAN1_RX_CFIFO_NO    0       //CAN1の受信をCFIFO0で行う （0のみ） [CAN_RX_METHOD -> CAN_RX_CFIFO 指定時に使用](*1)(*2)
#define CAN1_TX_CFIFO_NO    0       //CAN1の送信をCFIFO0で行う （0のみ） [CAN_TX_METHOD -> CAN_TX_CFIFO 指定時に使用](*1)(*2)
#define CAN1_TX_TXQUEUE_NO  0       //CAN0の送信をTXQUEUE0で行う （0のみ） [CAN_TX_METHOD -> CAN_TX_TXQUEUE 指定時に使用](*2)

//(*1)CFIFO0は、送信または受信での排他利用
//(*2) 0のみ指定可能（CFIFO，TXQUEUEを複数使用できるタイプのマイコンと関数仕様を共通化するため、CFIFO，TXQUEUE番号を指定する様にしているため）

#elif (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD)

//RXFIFO(0) ~ RXFIFO(7) : 全ch共通
//CFIFO(0) ~ CFIFO(5) : 全ch共通
//TXQUEUE(0) ~ TXQUEUE(3) : ch毎に独立
#define CAN0_RX_RXFIFO_NO   0       //CAN0受信をRXFIFO0で行う （0~7の値を設定）[CAN_RX_METHOD -> CAN_RX_RXFIFO 指定時に使用](*1)
#define CAN0_RX_CFIFO_NO    1       //CAN0受信をCFIFO1で行う（0~2の値を設定，TXとは別な値とする）[CAN_RX_METHOD -> CAN_RX_CFIFO 指定時に使用](*2)
#define CAN0_TX_CFIFO_NO    0       //CAN0送信をCFIFO0で行う（0~2の値を設定） [CAN_TX_METHOD -> CAN_TX_CFIFO 指定時に使用](*2)
#define CAN0_TX_TXQUEUE_NO  0       //CAN0の送信をTXQUEUE0で行う(0~3の値を設定） [CAN_TX_METHOD -> CAN_TX_TXQUEUE 指定時に使用]
```

定義がエラーの場合のトラップ

RSCAN の場合の使用 FIFO 番号(*15)

CANFD_B, CANFD_2, CANFD_Lite の場合の使用 FIFO 番号(*16)

CANFD の場合の使用 FIFO 番号(*17)

```
#define CAN1_RX_RXFIFO_NO   1       //CAN1受信をRXFIFO1で行う （0~7の値を設定） [CAN_RX_METHOD -
> RXFIFO 指定時に使用](*1)
#define CAN1_RX_CFIFO_NO    4       //CAN1受信をCFIFO4で行う（3~5の値を設定，TXとは別な値とする）
[CAN_RX_METHOD -> CAN_RX_CFIFO 指定時に使用](*2)
#define CAN1_TX_CFIFO_NO    3       //CAN1送信をCFIFO3で行う（3~5の値を設定） [CAN_TX_METHOD ->
CAN_TX_CFIFO 指定時に使用](*2)
#define CAN1_TX_TXQUEUE_NO  0       //CAN1の送信をTXQUEUE0で行う(0~3の値を設定）[CAN_TX_METHOD
-> CAN_TX_TXQUEUE 指定時に使用]
//(*1)
//CAN0_RXFIFO_NO，CAN1_RXFIFO_NOは0-7の値に設定してください、CAN0_RXFIFO_NOとCAN1_RXFIFO_NOは別
な値としてください
//(*2)
//CAN0_TX_CFIFO_NO は0-2に設定してください
//CFIFOで受信する場合 CAN0_RX_CFIFO_NO は0-2(*3)に設定してください （TXとRXで番号が重複しない様設
定）
//CAN1_TX_CFIFO_NO は3-5に設定してください
//CFIFOで受信する場合 CAN1_RX_CFIFO_NO は3-5(*3)に設定してください （TXとRXで番号が重複しない様設
定）
//(*3)
//CAN-ch0/1端子と受信設定のCFIFO0-5は自由に割り当て可能ですが、割り込みの処理でCFIFO0-2がCAN0の割
り込み，CFIFO3-5がCAN1の割り込みに紐づいてるために、
//ch0(0-2)，ch1(3-5)の制約で運用しています（割り込み処理のプログラムに手を加えれば、受信時のCFIFO番
号とchの縛りはなくなります）
//※送信側は、CFIFO0-2がCAN-ch0端子，CFIFO3-5がCAN-ch1端子からの送信となります

#endif  ///CANマクロタイプ

//通信カウンタオーバフロー割り込みを抑止する（CANFD系）
/*
通信カウンタ(SOC)は通信成功時にインクリメントされ、0xFF(=255)を超えた場合に
チャネルエラー割り込みが発生する
通信カウンタのオーバフローはエラーではないので
下記変数定義時は、オーバフロー割り込みを生じさせない事とする
*/
//#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE    通信成功カウンタオーバフロー設定(*18)

//エラー割り込みで割り込みを無効化する閾値
#define CAN_ERROR_INTERRUPT_DISABLE_THRESHOLD   3   //3回目以降のエラー割り込み発生時、エラー割り
込みを無効化する
                                                エラー割り込み無効化閾値(*19)

//割り込みコールバック関数の有効化
#define CAN_USE_RX_CALLBACK_FUNCTION        //受信割り込み関数のコールバック関数を使用する
#define CAN_USE_TX_CALLBACK_FUNCTION        //送信割り込み関数のコールバック関数を使用する
#define CAN_USE_ERROR_CALLBACK_FUNCTION     //エラー割り込み関数のコールバック関数を使用する

#endif                                          コールバック関数の有効化(*20)
```

operation.h 内には、速度設定や送受信方式、使用する FIFO 番号、割込み使用の有無などの動作に関わる定義が記載されています。SAMPLE_MONITOR や本ソースコードを流用する場合は、このファイル内の定義が使われます。SAMPLE1~SAMPLE4 のサンプルプログラムでは、サンプルプログラム毎に設定値を変更したいので、別ファイル(can_operation2.h)内で同様の定義を行っています（SAMPLE1~SAMPLE4 では、can_opetation2.h 内の定義値が使用されます。）

(*1)CAN の速度設定

#define CAN_SPEED　CAN_BPS_1M

1Mbps 設定とする場合は上記です。CAN_BPS_1MB(1000), CAN_BPS_500K(500), CAN_BPS_250K(250), CAN_BPS_125K(125)などの値が指定可能です。（4 値は定数として予め定義済みです）

#define CAN_SPEED　100

など、数値を指定することも出来ます。単位は kbps です（上記は 100kbps 設定）。

　CAN のクロック周波数との兼ね合いで、必ずしも任意の速度値が設定出来る訳ではありません。

※CANFD 対応マイコンでは

#define CANFD_MODE

定義を行い、CANFD を有効化した場合は、クロック分周比は board.h 内で定義された値が使用されます。

CANFD_MODE 未定義時は、クロック分周比は速度設定値から計算された値が使用されます。

(*2)CANFD の速度設定（CANFD 対応マイコンで CANFD 使用時のみ使用）

#define CANFD_SPEED CANFD_BPS_2M

　CANFD（データ通信レート）の速度設定を行います。

CANFD_BPS_8M　8Mbps

CANFD_BPS_7_5M　7.5Mbps

CANFD_BPS_6_6M　6.6Mbps

CANFD_BPS_6M　6Mbps

CANFD_BPS_5M　5Mbps

CANFD_BPS_4M　4Mbps

CANFD_BPS_3_3M　3.3Mbps

CANFD_BPS_3M　3Mbps

CANFD_BPS_2M　2Mbps

CANFD_BPS_1M　1Mbps

　予め定義済みの上記値のみ指定可能です。CANFD の通信速度は、現状任意の値を指定可能にはなっていません。また、CANFDCLK の値により指定可能な速度値は変わります。

CANFDCLK [MHz]	8Mbps	7.5Mbps	6.6Mbps	6Mbps	5Mbps	4Mbps	3.3Mbps	3Mbps	2Mbps	1Mbps
80	○		○		○	○	○		○	○
66(66.667)			○				○			
60		○	○	○	○	○	○	○	○	○
50					○		○		○	○
48	○			○		○		○	○	○
40	○		○		○	○	○		○	○
33(33.333)							○			
32	○					○			○	○
30		○		○	○		○	○	○	○
25					○					○
24						○		○	○	○

　上表にない組み合わせで使用したい場合は、can_general.c 内のテーブルを拡張してください。

(*3)CANFD パラメータ

・トランシーバ遅延補償

#define CANFD_TDCE　DISABLE　//有効

#define CANFD_TDCE　ENABLE　//無効

4Mbps 以下で使用する場合は、DISABLE, 5Mbps 以上で使用する場合は、ENABLE が推奨です。

・第 2 サンプルポイント

#define CANFD_SSP　MEASURE_OFFSET　//測定値+オフセット値

#define CANFD_SSP　OFFSET_ONLY　//オフセット値のみ

・トランシーバ遅延補償オフセット値

#define CANFD_TDCO　0

0~255(0xFF)までの値が指定可能です

・トランシーバ RX エッジフイルタ

#define CANFD_REFE　DISABLE //有効

#define CANFD_REFE　ENABLE //無効

SAMPLE4, SAMPLE_MONITOR では、起動時に対話的にこれらのパラメータの入力が求められます。（対話入力を使用した場合は、本ファイルの定義は使用されません。）

(*4)ID 設定

#define CAN_ID_TYPE　CAN_ID_FORMAT_EID

CAN の ID として、拡張 ID(29bit)と標準 ID(11bit)の両方を取り扱います。

#define CAN_ID_TYPE　CAN_ID_FORMAT_SID

CAN の ID として標準 ID(11bit)のみを取り扱います。

(*5)デバッグ定義

#define SCI_DEBUG

定義を有効化した場合、割り込みに入ったタイミング等で、SCI 端末にデバッグメッセージを出力します。

#define CAN_INTERRUPT_DEBUG

定義を有効化した場合、割り込みに入ったタイミングで I/O 端子を反転させます、

(*6)割り込み優先度の設定

・受信割り込み

・送信割り込み

・エラー割り込み

の優先度を設定します。

#define CAN_RX_INTERRUPT　14

#define CAN_TX_INTERRUPT　14

#define CAN_ERROR_INTERRUPT　14

RX では、1~15 の値（1:最低優先度, 15:最高優先度）
RA（96 の割り込みが使用可能なマイコン）では、0~15（0:最高優先度, 15:最低優先度）
RA（32 の割り込みが使用可能なマイコン）では、0~3（0:最高優先度, 3:最低優先度）
の値が指定可能です。

(*7)CAN の受信方法
・FIFO とメールボックスの両方を使用（デフォルト）
#define CAN_RX_METHOD　(CAN_RX_MBOXFIFO | CAN_RX_MBOX)
・メールボックスを使用
#define CAN_RX_METHOD　CAN_RX_MBOX
・FIFO を使用
#define CAN_RX_METHOD　CAN_RX_MBOXFIFO

CAN モジュールでは、メールボックスと FIFO が使用可能です。最初の指定は、FIFO とメールボックスの両方を使う指定です。FIFO を使用する場合、FIFO はフィルタリングが 2 条件設定可能ですが、
(a)拡張 ID, データフレーム
(b)拡張 ID, リモートフレーム
(c)標準 ID, データフレーム
(d)標準 ID, リモートフレーム
の 4 通りの設定は出来ないので、本サンプルプログラムでは(a)(b)を FIFO で受信する設定としています。（(c)(d)はメールボックスで受信する設定。）（受信に FIFO のみを使用する場合は、ソースコードを変更すれば、(a)~(d)の内の 2 条件を設定可能です。）
受信・送信で FIFO を使用しない場合は、0~31 のメールボックスが利用可能。FIFO を使用する場合は、0~23 のメールボックスが利用可能です。
FIFO の段数は 4 段となりますので、4 つまでのメッセージであれば、受信操作を行わなくても CAN モジュール内の FIFO バッファに溜めておく事ができます。

(*8)CAN の送信方法
・FIFO を使用（デフォルト）
#define CAN_TX_METHOD　CAN_TX_MBOXFIFO
・メールボックスを使用
#define CAN_TX_METHOD　CAN_TX_MBOX

CAN モジュールの送信方法としても、FIFO とメールボックスが使用可能です。FIFO の段数は 4 段ですので、送信完了にならない場合でも 4 つまでのメッセージは FIFO に溜めておく事が出来ます。メールボックスを使用する場合は、データ送信前に次の送信を行うと、送信関数がエラーを返します。

(*9)割り込みの有効化
#define CAN_RX_INTERRUPT　ENABLE
#define CAN_TX_INTERRUPT　ENABLE
#define CAN_ERROR_INTERRUPT　ENABLE

受信割り込み、送信割り込み、エラー割り込みの有効・無効の設定です。DISABLE の場合は、割り込みを無効化します。

(*10)RSCAN の受信方法
・受信バッファで受信
#define CAN_RX_METHOD　CAN_RX_RXBUF
・RXFIFO で受信（デフォルト）
#define CAN_RX_METHOD　CAN_RX_RXFIFO
・SRFIFO で受信
#define CAN_RX_METHOD　CAN_RX_SRFIFO

RSCAN モジュールの受信方法は 3 種類の中からの選択となります。受信割り込みを使用する場合は、RXFIFO か SRFIFO からの選択となります。

(*11)RSCAN の送信方法
・送信バッファで送信
#define CAN_TX_METHOD　CAN_TX_TXBUF
・SRFIFO で送信（デフォルト）
#define CAN_TX_METHOD　CAN_TX_SRFIFO

RSCAN モジュールの送信方法は、2 種類のどちらかの選択です。SRFIFO は 1 本しかないため、受信で SRFIFO を使用した場合は、送信では使用できません。（SRFIFO は受信か送信の排他利用となります。）
FIFO、受信バッファの合計で 16 のバッファを割り当て可能ですが、本プログラムでは、FIFO を使用する場合、受信 8 段、送信 8 段の設定としています。

(*12)CANFD 系モジュールの受信方法
・受信バッファで受信
#define CAN_RX_METHOD　CAN_RX_RXBUF
・RXFIFO で受信（デフォルト）
#define CAN_RX_METHOD　CAN_RX_RXFIFO
・CFIFO で受信
#define CAN_RX_METHOD　CAN_RX_CFIFO

※RSCAN の SRFIFO（送受信 FIFO）と CANFD 系の CFIFO（共通 FIFO）は、名称が異なるだけで同じような機能です

CANFD 系モジュールの受信方法は 3 種類の中からの選択となります。
（CANFD モジュールの制約：受信割り込みを使用する場合は、RXFIFO か CFIFO からの選択となります。）

(*13)CANFD 系モジュールの送信方法
・送信バッファで送信
#define CAN_TX_METHOD　CAN_TX_TXBUF
・TX キューで送信
#define CAN_TX_METHOD　CAN_TX_TXQUEUE
・CFIFO で送信（デフォルト）
#define CAN_TX_METHOD　CAN_TX_CFIFO

CANFD 系モジュールの送信方法は、3 種類の内からの選択です。

(*14)CANFD の使用設定（※CANFD 対応マイコンのみ有効）
#define CANFD_MODE
CANFD_MODE 定数定義時は、CANFD を使用する設定を行います。未定義時は、CANFD 対応マイコンで CAN の機能のみ使用する設定を行います。
（SAMPLE4, SAMPLE_MONITOR で、本定数が定義されます。）

(*15)RSCAN の使用 FIFO 番号
・RXFIFO 番号
#define CAN0_RX_RXFIFO_NO　0
RXFIFO は 2 本となりますので、0 または 1 が指定可能です。
（RXFIFO0/RXFIFO1 の両方を使用する場合は、FIFO と受信バッファでトータル 16 に収まるように、FIFO 段数の調整を行ってください。）
・SRFIFO 番号（受信時の使用）
#define CAN0_RX_SRFIFO_NO　0
・SRFIFO 番号（送信時の使用）
#define CAN0_TX_SRFIFO_NO　0
RSCAN モジュールでは、SRFIFO は 0 番のみとなります。（0 以外の数値は有効にはなりません。）

(*16)CANFD_B, CANFD_2, CANFD_Lite の使用 FIFO 番号
・RXFIFO 番号
#define CAN0_RX_RXFIFO_NO　0
#define CAN1_RX_RXFIFO_NO　0　//CANFD_2 モジュールのみ
RXFIFO は 2 本となりますので、0 または 1 が指定可能です。
（RXFIFO0/RXFIFO1 の両方を使用する場合は、FIFO と受信バッファでモジュール搭載のバッファメモリ容量に収まるように、FIFO 段数の調整を行ってください。）
CAN0 は CAN-ch0, CAN1 は CAN-ch1 の設定です。

・CRFIFO 番号（受信時の使用）
#define CAN0_RX_CFIFO_NO　0
#define CAN1_RX_CFIFO_NO　0　//CANFD_2 モジュールのみ
・CFIFO 番号（送信時の使用）
#define CAN0_TX_CFIFO_NO　0
#define CAN1_TX_CFIFO_NO　0　//CANFD_2 モジュールのみ
　CANFD_B, CANFD_2, CANFD_Lite モジュールでは、CFIFO は 0 番のみとなります。（0 以外の数値は有効にはなりません。）

・TX キュー番号
#define CAN0_TX_TXQUEUE　0
#define CAN1_TX_TXQUEUE　0　//CANFD_2 モジュールのみ
　CANFD_B, CANFD_2, CANFD_Lite モジュールでは、TX キューは 0 番のみとなります。（0 以外の数値は有効にはなりません。）

(*17)CANFD の使用 FIFO 番号
・RXFIFO 番号
#define CAN0_RX_RXFIFO_NO　0
#define CAN1_RX_RXFIFO_NO　1
　RXFIFO は 8 本となりますので、0~7 が指定可能です。
※同一の番号とすると、CAN の物理 ch とメッセージ格納先(RXFIFO(n))の対応が取れなくなりますが、設定上 NG ではありません

・CFIFO 番号
#define CAN0_TX_CFIFO_NO　0
#define CAN0_RX_CFIFO_NO　1
#define CAN1_TX_CFIFO_NO　3
#define CAN1_RX_CFIFO_NO　4
　CFIFO は 6 本となりますので、0~5 が指定可能です。番号の割り振りは自由ですが、TX と RX で同じ番号を使用する事はできません。
　CFIFO0~2 が CAN0-COMFRX（CFIFO 受信割り込み）、3~5 が CAN1-COMFRX に紐付けされているので、CAN-ch0 では 0~2 を、CAN-ch1 では 3~5 を使用すると、CAN の物理 ch と受信割り込みの対応が判りやすくなります。（CANFD モジュールでは、CFIFO は 6 本あるので送信と受信の両方で使用しても問題ありません。）

・TX キュー番号
#define CAN0_TX_TXQUEUE_NO　0
#define CAN1_TX_TXQUEUE_NO　0
　CANFD モジュールでは、TX キューは ch あたり 4 本となりますので、0~3 の番号が指定可能です。（ch 毎に独立しているので同じ番号で問題ありません。）CANFD_B, CANFD_2, CANFD_Lite モジュールでは 0 のみ。

(*18)通信成功カウンタオーバフロー割り込み設定
#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE
　デフォルト（上記コメントアウト）の場合は、全てのエラー割り込みを有効化しますので、通信成功カウンタオーバフローの場合も割り込みが生じます（エラー割り込み有効化時）。
　通信成功カウンタオーバフローは、基本的にはエラーではないため、上記有効（定数が定義された状態）とすると、通信成功オーバフロー時に割り込みが掛からない様に設定します。
（デフォルトでは、起動後 256 回通信が成功すると、割り込みが掛かります。）

(*19)エラー割り込み無効化閾値設定
#define CAN_ERROR_INTERRUPT_DISABLE_THRESHOLD　3
　デフォルトでは、エラー割り込みが 3 回連続で入ると、エラー割り込み自体を無効化します。
（端末から'C'コマンドを入力すると、エラー発生回数がリセットされます。）
　エラーの発生要因が自ボードの場合（送信したが ACK が返らないなど）、エラー発生要因を取り除く事が出来ますが、通信相手起因の場合（通信相手が、速度が合わない CAN パケットを連続で送信してくるなど）、エラー割り込みの無限ループに突入します。
　その場合は、連続で発生したエラーが本定数で設定した値に達した際に、当該エラー割り込み自体を無効化する処理を行います。

(*20)割り込みコールバック関数の有効化
#define CAN_USE_RX_CALLBACK_FUNCTION
#define CAN_USE_TX_CALLBACK_FUNCTION
#define CAN_USE_ERROR_CALLBACK_FUNCTION
　定義時、受信割り込み関数、送信割り込み関数、エラー割り込み関数が呼ばれた際、割り込み関数を抜ける直前に既定のコールバック関数を呼び出します。
　割り込み関数自体は、SAMPLEn に依存しない形の内容としています。
（受信割り込み関数では、受信データを受信リングバッファにコピー。送信割り込み関数では、送信結果をフラグ変数にコピー。エラー割り込み関数では、エラー内容をフラグ変数にコピー。）
　SAMPLEn に固有の処理は、コールバック関数内に記載
（SAMPLE2 であれば、受信データの表示、SAMPLE3 であればリモートフレームに対する応答など）
する形としています。

8.6. can_operation2.h

格納先:

RX: ¥SOURCE¥RX_CANST¥source¥ALL¥common¥

RA: ¥SOURCE¥RA_CANST¥source¥ALL¥common¥

SAMPLE1~SAMPLE4 は、サンプルプログラム毎に割り込みや送受信方法を変更しているので、SAMPLE1~SAMPLE4 の動作は、本ファイルで定義する

CAN の SAMPLE1(*1)

CAN の SAMPLE2~3(*2)

```
#ifndef CAN_OPERATION2_H__
#define CAN_OPERATION2_H__

/*----------------------------------------------------------------------
    インクルード
----------------------------------------------------------------------*/
#include    "can_common.h"
#include    "program_select.h"

/*----------------------------------------------------------------------
    定数定義
----------------------------------------------------------------------*/
#if defined(SAMPLE4)
#define CANFD_MODE  //SAMPLE4ではCANFDモードとする
#endif

//送受信方法の選択,割り込みの使用（SAMPLEプログラム実行時はSAMPLEnに応じて選択）

//CAN, RX600, RX700, RA6M2, RA2L1などCANマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CAN)

#if defined(SAMPLE1)

//CANの受信方法
#define CAN_RX_METHOD       CAN_RX_MBOX             //メールボックスを使用（デフォルト）
//#define CAN_RX_METHOD       CAN_RX_MBOXFIFO       //メールボックス(FIFO)を使用(*1)

//CANの送信方法
#define CAN_TX_METHOD       CAN_TX_MBOX             //メールボックスを使用（デフォルト）
//#define CAN_TX_METHOD       CAN_TX_MBOXFIFO       //メールボックス(FIFO)を使用

//SAMPLE1では割り込みは使用しない
#define CAN_RX_INTERRUPT    DISABLE                 //受信割り込み
#define CAN_TX_INTERRUPT    DISABLE                 //送信割り込み
#define CAN_ERROR_INTERRUPT DISABLE                 //エラー割り込み

#elif defined(SAMPLE2) || defined(SAMPLE3)

//CANの受信方法
#define CAN_RX_METHOD       (CAN_RX_MBOXFIFO | CAN_RX_MBOX)     //メールボックス(FIFO)+メールボックスを使用（デフォルト）(*2)
//#define CAN_RX_METHOD       CAN_RX_MBOX                       //メールボックスを使用
//#define CAN_RX_METHOD       CAN_RX_MBOXFIFO                   //メールボックス(FIFO)を使用(*1)

//CANの送信方法
#define CAN_TX_METHOD       CAN_TX_MBOXFIFO         //メールボックス(FIFO)を使用（デフォルト）
//#define CAN_TX_METHOD       CAN_TX_MBOX           //メールボックスを使用

//(*1)メールボックスFIFOを選択した場合は、拡張IDのみ受信
//(*2)拡張IDはFIFOで受信、標準IDはメールボックスで受信

//SAMPLE2~SAMPLE3では割り込みを使用する
#define CAN_RX_INTERRUPT    ENABLE                  //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                  //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                  //エラー割り込み

#endif //SAMPLEn
#endif //CAN_MACRO_TYPE
```

```
//RSCAN, RX200などRSCANマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_RSCAN)

#if defined(SAMPLE1)                                    RSCAN の SAMPLE1(*3)

//CANの受信方法
#define CAN_RX_METHOD       CAN_RX_RXBUF            //受信バッファを使用(デフォルト)
//#define CAN_RX_METHOD       CAN_RX_RXFIFO//受信FIFOを使用
//#define CAN_RX_METHOD       CAN_RX_SRFIFO    //送受信FIFOを使用(*1)

//CANの送信方法
#define CAN_TX_METHOD       CAN_TX_TXBUF            //送信バッファを使用(デフォルト)
//#define CAN_TX_METHOD       CAN_TX_SRFIFO//送受信FIFOを使用(*1)

//SAMPLE1では割り込みは使用しない
#define CAN_RX_INTERRUPT    DISABLE                 //受信割り込み
#define CAN_TX_INTERRUPT    DISABLE                 //送信割り込み
#define CAN_ERROR_INTERRUPT DISABLE                 //エラー割り込み

#elif defined(SAMPLE2) || defined(SAMPLE3)          RSCAN の SAMPLE2~3(*4)

//CANの受信方法
#define CAN_RX_METHOD       CAN_RX_RXFIFO//受信FIFOを使用(デフォルト)
//#define CAN_RX_METHOD       CAN_RX_RXBUF//受信バッファを使用
//#define CAN_RX_METHOD       CAN_RX_SRFIFO//送受信FIFOを使用(*1)

//CANの送信方法
#define CAN_TX_METHOD       CAN_TX_SRFIFO//送受信FIFOを使用(デフォルト)(*1)
//#define CAN_TX_METHOD       CAN_TX_TXBUF//送信バッファを使用

//(*1)送受信FIFOは1本なので、受信「または」送信のいずれかで使用可

//SAMPLE2~SAMPLE3では割り込みを使用する
#define CAN_RX_INTERRUPT    ENABLE                  //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                  //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                  //エラー割り込み

#endif //SAMPLEn
#endif //CAN_MACRO_TYPE


//CAN-FD, RX660, RX26T, RA6M5, RA6T2, RA8M1などCAN-FDマクロ搭載マイコン
#if (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD) || (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_B)
|| (CAN_MACRO_TYPE == CAN_MACRO_TYPE_CANFD_LITE) || (CAN_MACRO_TYPE ==
CAN_MACRO_TYPE_CANFD_2)

#if defined(SAMPLE1)                                CANFD 系の SAMPLE1(*5)

//CANの受信方法
#define CAN_RX_METHOD       CAN_RX_RXBUF            //受信バッファを使用（デフォルト）
//#define CAN_RX_METHOD       CAN_RX_RXFIFO           //受信FIFOを使用
//#define CAN_RX_METHOD       CAN_RX_CFIFO            //共通FIFOを使用(*1)

//CANの送信方法
#define CAN_TX_METHOD       CAN_TX_TXBUF            //送信バッファを使用（デフォルト）
//#define CAN_TX_METHOD       CAN_TX_TXQUEUE          //送信キューを使用
//#define CAN_TX_METHOD       CAN_TX_CFIFO            //共通FIFOを使用(*1)

//(*1)共通FIFOは1本なので、受信「または」送信のいずれかで使用可

//SAMPLE1では割り込みは使用しない
#define CAN_RX_INTERRUPT    DISABLE                 //受信割り込み
#define CAN_TX_INTERRUPT    DISABLE                 //送信割り込み
#define CAN_ERROR_INTERRUPT DISABLE                 //エラー割り込み
```

```
#elif defined(SAMPLE2) || defined(SAMPLE3) || defined(SAMPLE4)

//CANの受信方法(CAN_RX_RXFIFO, CAN_RX_CFIFO, CAN_RX_RXBUFのいずれか)
#define CAN_RX_METHOD       CAN_RX_RXFIFO           //受信FIFOを使用（デフォルト）
//#define CAN_RX_METHOD       CAN_RX_CFIFO          //共通FIFOを使用(*1)
//#define CAN_RX_METHOD       CAN_RX_RXBUF          //受信バッファを使用

//CANの送信方法(CAN_TX_TXBUF, CAN_TX_TXQUEUE, CAN_TX_CFIFOのいずれか)
#define CAN_TX_METHOD       CAN_TX_CFIFO            //共通FIFOを使用（デフォルト）(*1)
//#define CAN_TX_METHOD       CAN_TX_TXBUF          //送信バッファを使用
//#define CAN_TX_METHOD       CAN_TX_TXQUEUE        //送信キューを使用

//(*1)共通FIFOは1本なので、受信「または」送信のいずれかで使用可

//SAMPLE2~SAMPLE4では割り込みを使用する
#define CAN_RX_INTERRUPT    ENABLE                  //受信割り込み
#define CAN_TX_INTERRUPT    ENABLE                  //送信割り込み
#define CAN_ERROR_INTERRUPT ENABLE                  //エラー割り込み

#endif //SAMPLEn
#endif //CAN_MACRO_TYPE

#endif
```

CANFD 系の SAMPLE2~4(*6)

(*1)CAN モジュールでの SAMPLE1 の設定

・受信　メールボックスを使用

・送信　メールボックスを使用

・割り込み　未使用

(*2)CAN モジュールでの SAMPLE2~3 の設定

・受信　FIFO＋メールボックスを使用

・送信　FIFO を使用

・割り込み　使用

(*3)RSCAN モジュールでの SAMPLE1 の設定

・受信　受信バッファを使用

・送信　送信バッファを使用

・割り込み　未使用

(*4)RSCAN モジュールでの SAMPLE2~3 の設定

・受信　RXFIFO を使用

・送信　SRFIFO を使用

・割り込み　使用

(*5)CANFD 系モジュールでの SAMPLE1 の設定

・受信　受信バッファを使用

・送信　送信バッファを使用

・割り込み　未使用

(*6)CANFD 系モジュールでの SAMPLE2~4 の設定

・受信　RXFIFO を使用

・送信　CFIFO を使用

・割り込み　使用

　SAMPLE1~SAMPLE4 で、送受信方法や割り込み使用有無を変更したい場合は、can_operation2.h を編集してください。

9. 共通ソースファイルに関して

CAN モジュール種別に依存しない処理が記載されているファイルです。

9.1. can_general.c 関数仕様

dlc2byte

概要:DLC 値変換関数

宣言:

unsigned short dlc2byte(unsigned char dlc)

説明:

・DLC 値のデータバイト数への変換

を行います

引数:

dlc: DLC 値(0-15)

戻り値:

データバイト数(0-8, 12, 16, 20, 24, 32, 48, 64)

補足:

引数(dlc)	戻り値
0~8	0~8
9	12
10	16
11	20
12	24
13	32
14	48
15	64

単純に、上記値を返す関数です。CAN の DLC 値に対応するデータバイト数を返します。

can_timing_parameter

概要:CAN のタイミングパラメータ設定関数

宣言:

int can_timing_parameter(unsigned short can_speed, float can_clock, float sampling_point, unsigned short *brp, unsigned char *sjw, unsigned char *tseg1, unsigned char *tseg2, float *tol_result)

説明:

・CAN のタイミングを決めるレジスタ値の算出

を行います

引数:

can_speed: CAN の速度　kbps 単位での指定（500kbps の場合は 500）

can_clock: CAN モジュールのクロック周波数　MHz 単位での指定（60MHz の時は 60）

sampling_point: サンプリング位置の指定（75%の位置を指定する場合は 0.75）

*brp: 計算された分周比設定値　BRP レジスタに戻り値を指定

*sjw: 計算された SJW 値　SJW レジスタに戻り値を指定

*tseg1: 計算された TSEG1 値　TSEG1 レジスタに戻り値を指定

*tseg2: 計算された TSEG2 値　TSEG2 レジスタに戻り値を指定

*tol_result: 計算された設定パラメータの速度誤差

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

CAN_RET_VALUE_ERROR(-6): 計算結果エラー

使用例:

ret = can_timing_parameter(1000, (float)CAN_CLOCK, 0.75f, &brp, &sjw, &tseg1, &tseg2, &tol_result);

1Mbps(1000kbps), サンプリングポイント 75%に設定する場合

brp, sjw, tseg1, tseg2 を CAN のレジスタに設定してください。tol_result は未使用でも構いません。

補足:

CANFD 対応マイコンの場合は TSEG1=2~256, TSEG2=2~128、CANFD 非対応のマイコンは TSEG1=4~16, TSEG2=2~8 の範囲でパラメータを算出します。

#define CAN_SPEED_TORR_MIN で CAN_SPEED_TORR_MIN を定義した場合は、最小誤差となる分周比を探します。未定義の場合は、1ppm 未満の分周比（できるだけ最小となる分周比）を選びます。

速度誤差が、0.5%未満に入らない場合は、CAN_RET_VALUE_ERROR(-6)を返します。

sampling_point は概ね 0.6 以上の値を指定する必要があります。

（sampling_point の値が小さいと TSEG1, TSEG2 の値の条件を満たさない）

CANFD 対応マイコンでは、計算値を（CAN の速度を決める）NBRP, NTSEG1, NTSEG2, NSJW レジスタに設定してください。

canfd_can_timing_parameter

概要：CANFD の CAN タイミングパラメータ設定関数

宣言：

int canfd_can_timing_parameter(unsigned short can_speed, float can_div_clock, float sampling_point, unsigned char *nsjw, unsigned char *ntseg1, unsigned char *ntseg2)

説明：

・CANFD 対応マイコンの CAN のタイミングを決めるレジスタ値(NTSEG1, NTSEG2, NSJW)の算出を行います

引数：

can_speed: CAN の速度　kbps 単位での指定（500kbps の場合は 500）
can_div_clock: CAN モジュールの分周後のクロック周波数　MHz 単位での指定（60MHz の時は 60）
sampling_point: サンプリング位置の指定（75%の位置を指定する場合は 0.75）
*nsjw: 計算された SJW 値　SJW レジスタに戻り値を指定
*ntseg1: 計算された TSEG1 値　TSEG1 レジスタに戻り値を指定
*ntseg2: 計算された TSEG2 値　TSEG2 レジスタに戻り値を指定

戻り値：

CAN_RET_SUCCESS(0): 正常終了
CAN_RET_ARGUMENT_ERROR(-2): 引数エラー
CAN_RET_VALUE_ERROR(-6): 計算結果エラー

使用例：

ret = can_timing_parameter(1000, 40.0f, 0.75f, &nsjw, &ntseg1, &ntseg2);
1Mbps(1000kbps), サンプリングポイント 75%に設定する場合
NBRP（分周比）は、別途決めた値を指定、CAN モジュールに入力しているクロック/分周比の値が 40MHz の場合、第二引数に 40 を与えてください
nsjw, ntseg1, ntseg2 を CAN のレジスタに設定してください。

補足：

本関数は分周比は計算しません。分周比は特定の値で使用する場合に使用する関数です。
CANFD 対応マイコンの場合、CANFD 側の分周比と CAN 側の分周比を同一としなければならないので、分周比固定でパラメータを計算する本関数を用意しています。

例えば、can_div_clock=80（80MHz)で、can_speed=125(125kbps)を指定した場合は、NTSEG1, NTSEG2 の値が許容範囲を超えるので CAN_RET_VALUE_ERROR(-6)を返します。
（can_timing_parameter()関数は分周比を含め計算するので 80MHz, 125kbps の様な値でも、パラメータを計算可能です。）

CANFD 対応マイコンで、CANFD の機能を使わない場合は、クロック分周比の縛りはありませんので、can_timing_parameter()関数を使い、CANFD の機能を使う場合は、クロック分周比を CAN 側だけで自由に決める事が出来ないので、canfd_can_timing_parameter()関数を使う事を想定した作りになっています。

canfd_timing_parameter

概要：CANFD の CANFD タイミングパラメータ設定関数

宣言：

int canfd_timing_parameter(unsigned short canfd_speed, unsigned char *dsjw, unsigned char *dtseg1, unsigned char *dtseg2)

説明：

・CANFD 対応マイコンの CANFD のタイミングを決めるレジスタ値(DTSEG1, DTSEG2, DSJW)の算出を行います

引数：

canfd_speed: CANFD の速度 kbps 単位での指定（5Mbps の場合は 5000）
*dsjw: 計算された SJW 値 SJW レジスタに戻り値を指定
*dtseg1: 計算された TSEG1 値 TSEG1 レジスタに戻り値を指定
*dtseg2: 計算された TSEG2 値 TSEG2 レジスタに戻り値を指定

戻り値：

CAN_RET_SUCCESS(0): 正常終了
CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

使用例：

```
#define CAN_CLOCK 80 //CAN のクロックが 80MHz の場合
ret = can_timing_parameter(CANFD_BPS_5M, &dsjw, &dtseg1, &dtseg2);
```

CANFD_BPS_5M(=5000), 5Mbps に設定する場合
dsjw, dtseg1, dtseg2 を CAN のレジスタに設定してください。

補足：

本関数は予め定義されたテーブル引きを行います
（定義済みの速度、CAN モジュールに入力された周波数の組み合わせで、設定値を拾う）

CANFD で 5Mbps 以上のビットレートの場合、タイミングパラメータがシビアになりますので、最適化値を適用できる様にテーブル化しています。
CAN_CLOCK 定数は整数値（66.666...MHz のときは切捨てで 66）としてください

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
80MHz	8Mbps	12.5 ns	6	3	70%
	6.6Mbps		8	3	75%
	5Mbps		11	4	75%
	4Mbps		14	5	75%
	3.3Mbps		17	6	75%
	2Mbps		29	10	75%
	1Mbps		59	21	75%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
66.667MHz	6.6Mbps	15 ns	6	3	70%
	3.3Mbps		14	5	75%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント	備考
60MHz	12Mbps	16.67 ns	2	2	60%	テスト用
	10Mbps		3	2	66.7%	テスト用
	7.5Mbps		5	2	75%	
	6.6Mbps		5	3	66.7%	
	6Mbps		6	3	70%	
	5Mbps		8	3	75%	
	4Mbps		10	4	73.3%	
	3.3Mbps		12	5	72.2%	
	3Mbps		14	5	75%	
	2Mbps		21	8	73.3%	
	1Mbps		44	15	75%	

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
50MHz	5Mbps	20 ns	6	3	70%
	3.3Mbps		10	4	73.3%
	2Mbps		17	7	72%
	1Mbps		36	13	74%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
48MHz	8Mbps	20.83 ns	3	2	66.7%
	6Mbps		5	2	75%
	4Mbps		8	3	75%
	3Mbps		11	4	75%
	2Mbps		17	6	75%
	1Mbps		35	12	75%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
40MHz	8Mbps	25 ns	2	2	60%
	6.6Mbps		3	2	66.7%
	5Mbps		5	2	75%
	4Mbps		6	3	70%
	3.3Mbps		8	3	75%
	2Mbps		14	5	75%
	1Mbps		29	10	75%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
33.333MHz	3.3Mbps	30 ns	6	3	70%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
32MHz	8Mbps	31.25 ns	2	1	75%
	4Mbps		5	2	75%
	2Mbps		11	4	75%
	1Mbps		23	8	75%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
30MHz	7.5Mbps	33.33 ns	2	1	75%
	6Mbps		2	2	60%
	5Mbps		3	2	66.7%
	3.3Mbps		5	3	66.7%
	3Mbps		6	3	70%
	2Mbps		10	4	73.3%
	1Mbps		21	8	73.3%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
25MHz	5Mbps	40 ns	2	2	60%
	1Mbps		17	7	72%

分周後のクロック	通信速度	1Tq	DTSEG1	DTSEG2	サンプルポイント
24MHz	4Mbps	41.67 ns	3	2	66.7%
	3Mbps		5	2	75%
	2Mbps		8	3	75%
	1Mbps		17	6	75%

定義されているテーブルは上記です。上記以外のクロック周波数、通信速度の場合はテーブルを拡張する必要があります。

can_raad_data

概要:リングバッファデータ取得関数

宣言:

int can_read_data(unsigned int buf_num, can_message *msg)　　[CANFD 未使用時]
int can_read_data(unsigned int buf_num, canfd_message *msg)　[CANFD 使用時]

説明:

・リングバッファに保存されてるデータの取得
を行います

引数:

buf_num: リングバッファのバッファ番号の指定

*msg: 読み出しデータを格納する CAN メッセージ構造体　[CANFD 未使用時]

*msg: 読み出しデータを格納する CAN メッセージ構造体　[CANFD 使用時]

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RETFLAG_LOST_DATA(0x80): 未読み出しで上書きされたデータあり
（残っているデータで一番古いものを返す）

CAN_RET_NODATA(-1): 未読み出しのデータなし

使用例:

```
can_message msg;
while (can_read_data(CAN_CH0, &msg) != CAN_RET_NODATA)
{
    //msg の読み出し結果に応じた処理
}
```

補足:

本サンプルプログラムでは、リングバッファは CAN の ch 数分確保しています。

can_read_data(CAN_CH0, &msg)で、CAN-ch0 で受信したデータの読み出しが可能です。

リングバッファのサイズは can_operation.h 内で CAN_RECV_BUF_SIZE(16),16 に定義されています。

（リングバッファのサイズは 16 ですが、格納可能なデータは 15 までとなります）

can_raad_data_size

概要:リングバッファデータ数取得関数

宣言:

```
int can_read_data_size(unsigned int buf_num)
```

説明:

・リングバッファに保存されてるデータ数の取得

を行います

引数:

buf_num: リングバッファのバッファ番号の指定

戻り値:

0: リングバッファに未読み出しのデータなし

1 以上: リングバッファに溜まっているメッセージ数

can_read_buf_clear

概要:リングバッファクリア関数

宣言:

void can_read_buf_clear(unsigned int buf_num)

説明:

・リングバッファに保存されてるメッセージのクリア

を行います

引数:

buf_num: リングバッファのバッファ番号の指定

戻り値:

なし

補足:

リングバッファに溜まっているメッセージを全て読み出し済みとします

10.その他

10.1.CAN ポート

RX マイコンでは、複数の端子からCAN で使用するポートを割り当てる様になっています。本サンプルプログラムでは以下の**赤太字で記載**している端子を設定しています。

・CAN モジュール系(RX600/700)

	ch0		ch1		ch2	
	CTX0	CRX0	CTX1	CRX1	CTX2	CRX2
HSBRX71M176 HSBRX72M176 HSBRX72N176 HSBRX66N176 HSBRX72N144 HSBRX66N144	P32 **PD1**	P33 **PD2**	P14 P44	P15 P55	**P66**	**P67**
HSBRX71M100	**P32** PD1	**P33** PD2	P14 P44	P15 P55		
HSBRX72T144 HSBRX66T144	**PA0** P23 PA6 PB5 PC5 PD7 PF2	**PA1** P22 PA7 PB6 PC6 PE0 PF3				
HSBRX64MC	**P32** PD1	**P33** PD2	**P14** P44	**P15** P55	**P66**	**P67**
HSBRX65xxxx HSBRX671F144 HSBRX671F100	**P32** PD1	**P33** PD2	P14 P44	P15 P55		
HSBRX66T100A HSBRX66T100B	**P23** PA0 PB5 PD7	**P22** PA1 PB6 PE0				

・RSCAN モジュール系(RX200/RX100)

	CTXD0	CRXD0
HSBRX24Uxxx HSBRX24T100B	**PA0** PF2	**PA1** PF3
HSBRX231F100 SmartRX!!!	**P14** P54	**P15** P55
HSBRX23E-B100	P14 **PC0**	P15 **PC1**
HSBRX140F80	P14 **P54**	P15 **P55**

・CANFD_Lite モジュール系(RX660/RX261)

	CTX0	CRX0
HSBRX660-144H HSBRX660-100B	**P14** P32 P54 PD1	**P15** P33 P55 PD2
HSBRX261-100	P14 P32 **P54** PD1	P15 P33 **P55** PD2

・CANFD_Lite モジュール系(RX26T)

	CTX0	CRX0
HSBRX26T100	**P23** P92 PA0 PB3 PB5 PD7	**P22** P93 PA1 PB4 PB6 PE0

・CAN, CANFD, CANFD_B, CANFD_2 モジュール系(RA)

	ch0		ch1	
	CTX0	CRX0	CTX1	CRX1
RA8E1	P103 P203 P312 **P401** P704 P815	P102 P202 P311 **P402** P705 P814	P209 P415 **P512** P609	P208 P414 **P511** P610
HSBRA8M1F176 HSBRA8T1F176	P103 **P203** P312 P401 P704 P815	P102 **P202** P311 P402 P705 P814	P209 P415 **P512** P609	P208 P414 **P511** P610
HSBRA6Mxxx HSBRA4M3F144	P103 P203 P401 P704	P102 P202 P402 P705	P109 P512 **P609**	P110 P511 **P610**
HSBRA6E1F100 HSBRA6T1F100(*1)	**P103** P401	**P102** P402		
HSBRA6E2F64 HSBRA6T3F64 HSBRA4E2F64 HSBRA4T1F64	**P103** P401 P109	**P102** P402 P110		
HSBRA6T2F100(*2)	**PB09** PA11 PB04 PB06 PB13 PD01	**PB08** PA12 PB03 PB05 PB12 PD00		
HSBRA4Mxxx	**P103** P109 P401	**P102** P110 P402		
HSBRAE1F64	**P103** P401	**P102** P402		
HSBRA2A1F64	P110 **P304** P407 P409	**P303** P408		
HSBRA2L1F100 HSBRA2L1F64	**P103** P109 P401	**P102** P110 P402		

(*1)当該ボードは、ジャンパで使用するポートを選べるようになっていますが、サンプルプログラムでは P103/P102 側を有効化ています。

(*2)当該ボードは、ジャンパで使用するポートを選べるようになっていますが、サンプルプログラムでは PB09/PB08 側を有効化ています。

※P109/P110 は SCI9 で UART 通信ポートとして使用していますので、CAN ポートとして、P109/P110 に割り当てを行う場合は、14P コネクタ経由での UART 通信は行えません

例えば、RX65 系で CAN の ch1 を使用したい場合は、P14,P15（または、P44, P45）端子に、CAN のトランシーバ回路（北斗電子製品では、「CAN ドライバボード」）を接続し、プログラムで端子を CAN ポートに設定する事により、ch1 の CAN ポートが使用できます。

RX65 系（144 or 100 ピン）で CAN の ch1 を使用する場合、
SOURCE¥RX_CANST¥RX65_144_100¥src¥usr_src¥settings¥board.h

```
#define CAN0_VALID 1//CAN0有効
#define CAN1_VALID 1//CAN1有効  [変更]

#define ICLK 120
#define PCLKB 60
#define XTAL 24
#define CAN_CLOCK PCLKB

//CANのクロックソース
#define CAN_CLOCK_SOURCE    CAN_CLOCK_PCLK//PCLK(B)をCANクロックとして使用

#define MCU_TYPE MCU_TYPE_RX65_2M

#if (BOARD_TYPE == HSBRX651F144A)
#define PIN_NO 144
#endif

#if (BOARD_TYPE == HSBRX651F100A)
#define PIN_NO 100
#endif

//CAN 端子設定（使用端子のコメントアウトを外す）

//CAN-ch0
#define CAN0_TX_P32
//#define CAN0_TX_PD1

#define CAN0_RX_P33
//#define CAN0_RX_PD2

//CAN-ch1
#define CAN1_TX_P14 //[変更]
//#define CAN1_TX_P54

#define CAN1_RX_P15 //[変更]
//#define CAN1_RX_P55
```

上記、[変更]の行（3 箇所）を変更（P14, P15 に外付けの CAN ドライバ回路を接続する場合)して、プログラムを「リビルド」してください。

10.2.デバッガの接続に関して

デバッガ（ルネサス E1, E20, E2, E2lite）を接続する際は、USB-ADAPTER-RX14 上の 14P コネクタにデバッガを接続してください。（※E1, E20 は RX のみ）

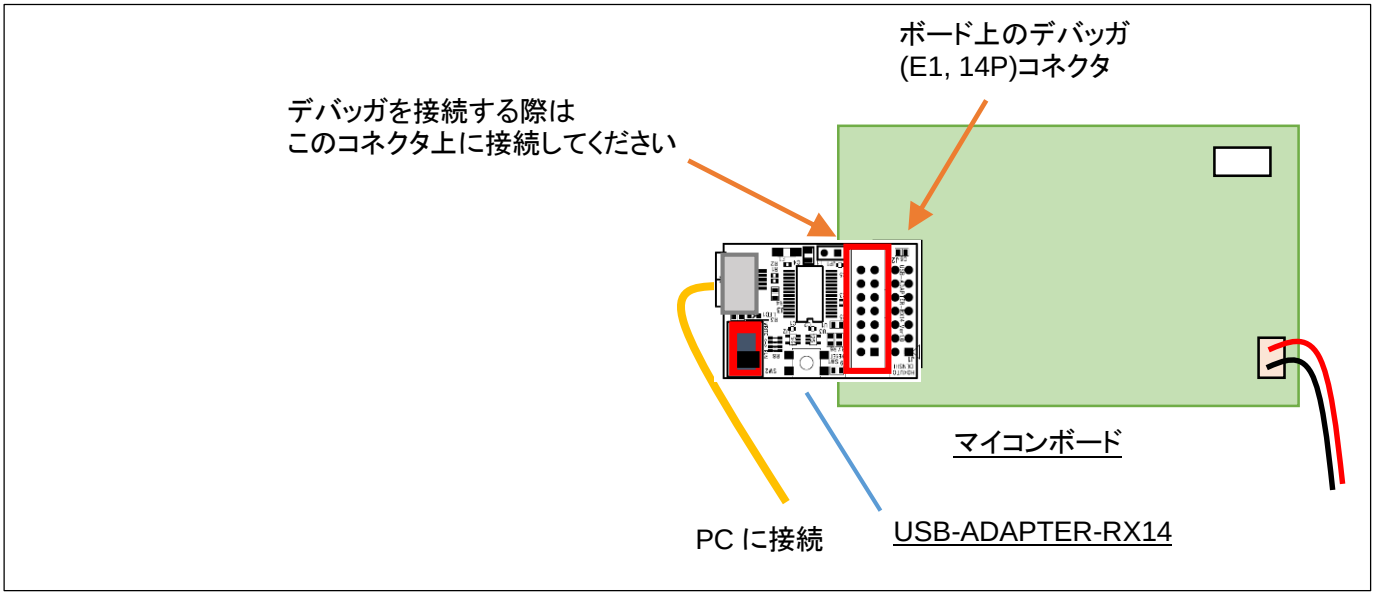


図 10-1 デバッガ接続

USB-ADAPTER-RX14 は、デバッガ接続用の信号をスルーで 14P コネクタに接続していますので、USB-ADAPTER-RX14 上の 14P コネクタにデバッガを接続する事により、デバッガを使用したデバッグが可能です。

－RA－

RA のボードでは、E2, E2Lite のどちらを使用するかによって、ジャンパピンの設定が必要です。
（詳細はマイコンボードの取扱説明書を参照してください）

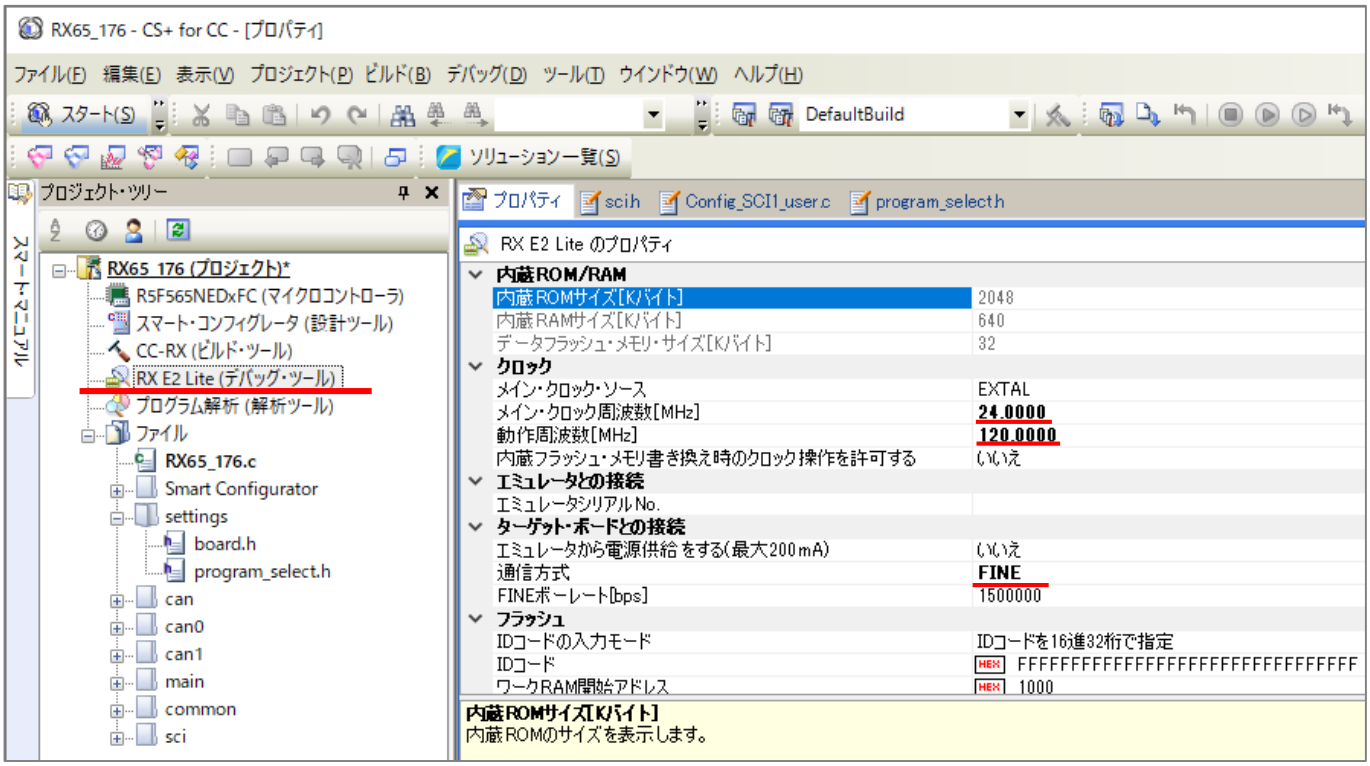
10.3.デバッガ(RX, CS+)

・デバッガの選択

E1, E20 使用時は、
RX E1(Serial)
RX E20(Serial)
を選択してください。(JTAG 接続と、USB-ADAPTER-RX14 は同時に使用できません。)

E2, E2Lite の場合は、通信方式:FINE を選択してください。

・デバッグツール設定



通信方式が選べる場合(RX600/700/RX26T 系マイコンと、E2, E2Lite デバッガの組み合わせ)では、**FINE** を選択してください。

「メイン・クロック・ソース」と「メイン・クロック周波数」は、ボードに合わせて設定してください。(CD 内のサンプルプロジェクトは、予め設定済みです)

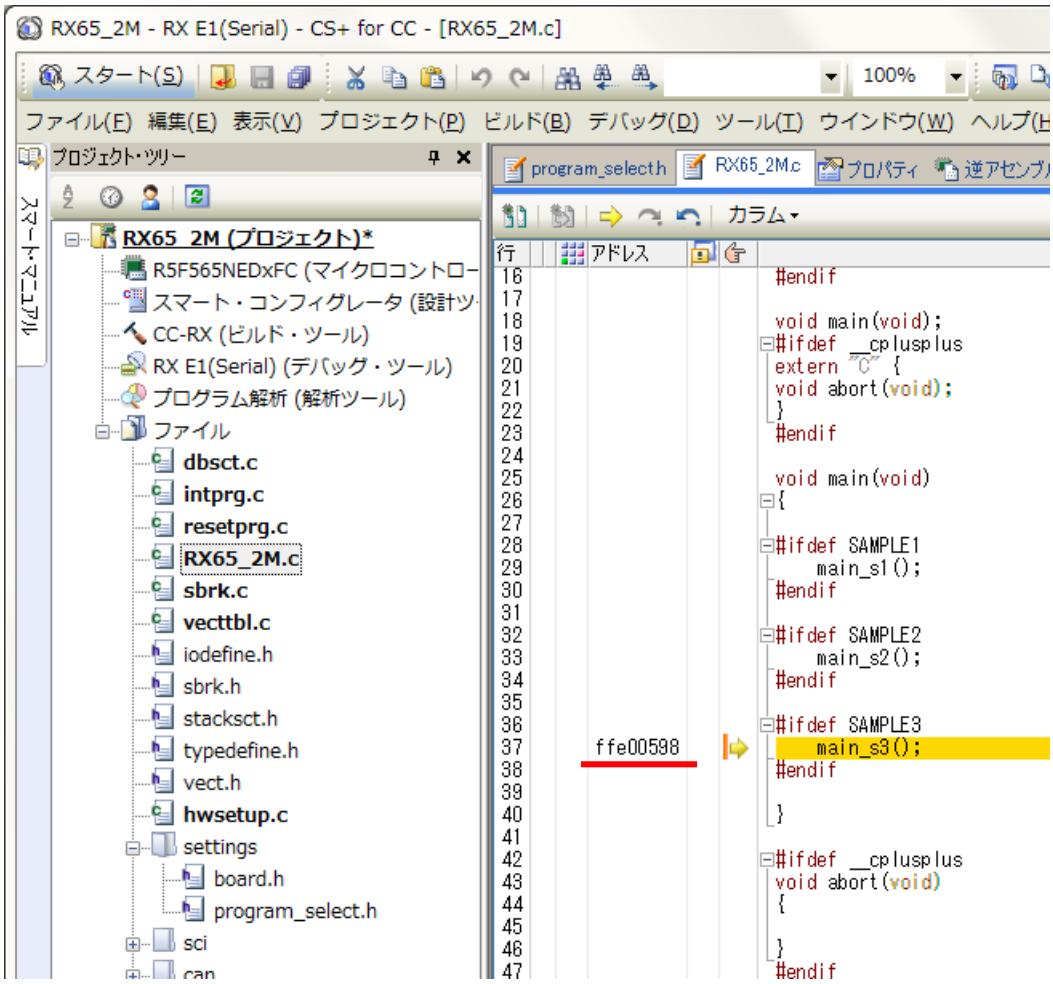
メイン・クロック・ソースは **EXTAL** を選択し、メイン・クロック周波数は、ボード搭載の水晶振動子の値を入力してください。

	メイン・クロック周波数 [MHz]	動作周波数 [MHz]
HSBRX71Mxxx HSBRX72M176 HSBRX72Nxxx	24	240
HSBRX72T144	24	200
HSBRX64MC HSBRX65xxxx HSBRX66Nxxx HSBRX671Fxxx	24	120
HSBRX261-100	8	64
HSBRX660-xxxy HSBRX26T100	24	120
HSBRX66T144	24	160
HSBRX66T100A/B	20	160
HSBRX24Uxxx HSBRX24T100B	10	80
HSBRX231F100 SmartRX!!!	8	54
HSBRX23E-B100	8	32
HSBRX140F80	8	48

デバッガと、USB-ADAPTER-RX14 を経由した PC との通信は同時に使用可能です。

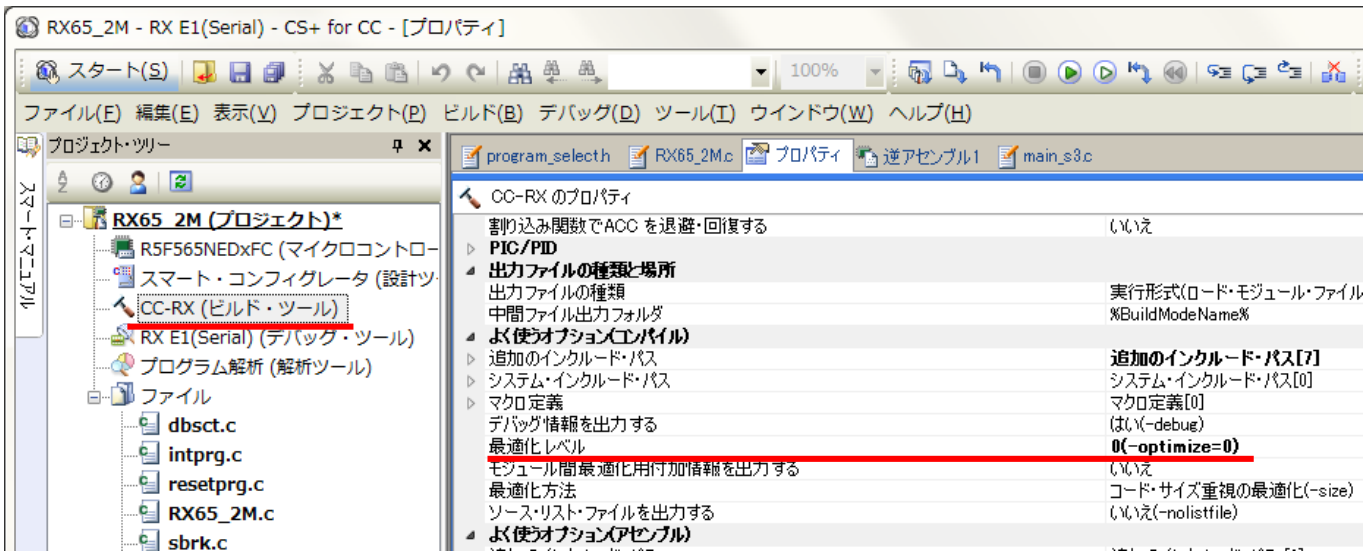
ビルド後、

デバッグ－デバッグツールへダウンロード



接続が成功した場合は、アドレスのところに、数値が入り、オレンジのライン（プログラムが停止しているソースの行）が表示されます。 デバッガを使用した場合、特定の場所でプログラムの実行を停止したり、1 行ずつ実行したり、変数をモニタしたりする事が可能です。デバッグ機能の詳細は、CS+のマニュアルを参照ください。

・ビルドの設定



デバッグ時、「モニタしたい変数の値が思ったように表示されない」「ステップ実行時ソースの上の行から順番に実行されない」といった時は、

ビルド・ツール

　最適化レベル　**0(-optimize=0)**

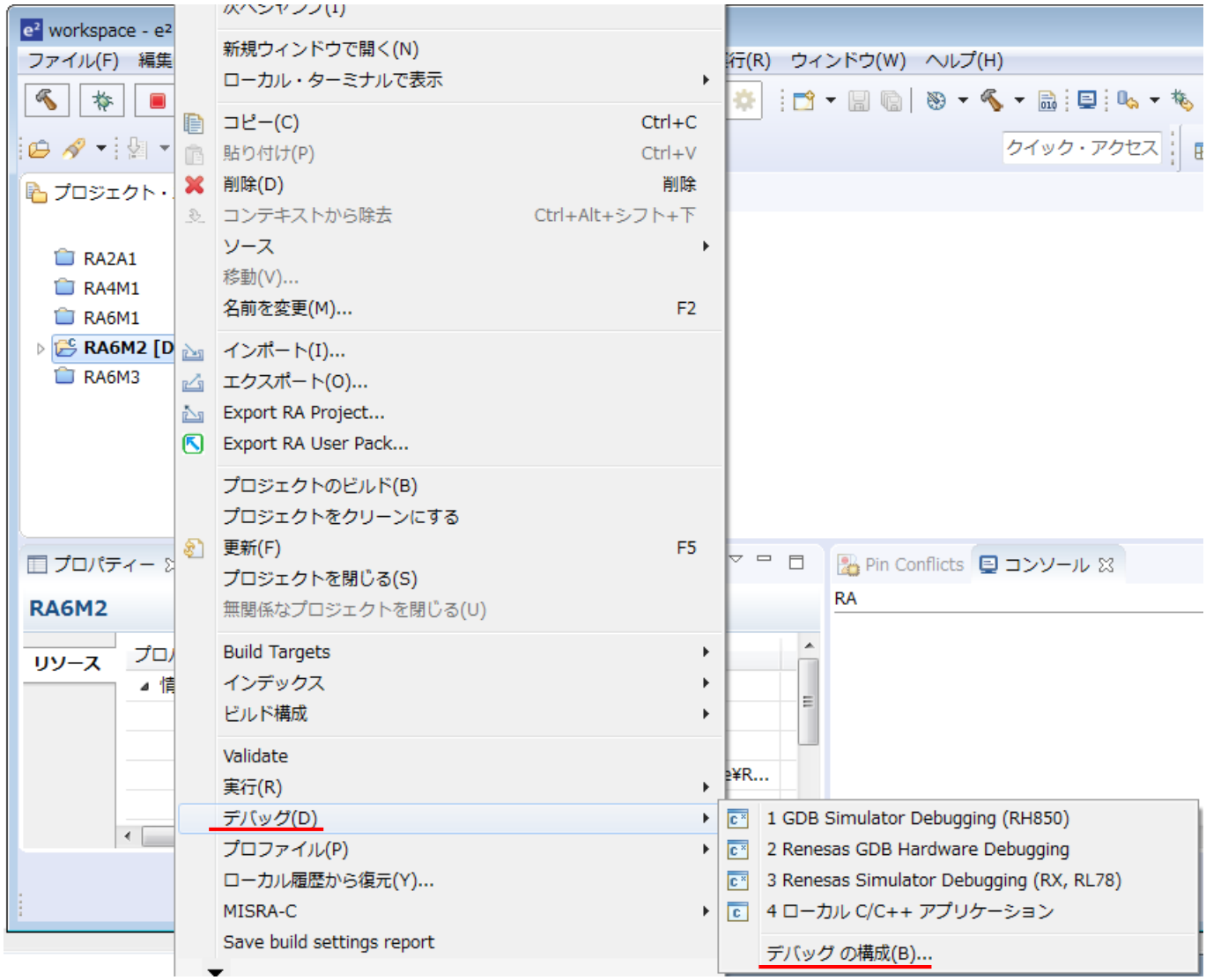
に変更してみてください。

（本オプション変更後は、プロジェクトのビルド及び、マイコンボードへの再ダウンロードが必要）

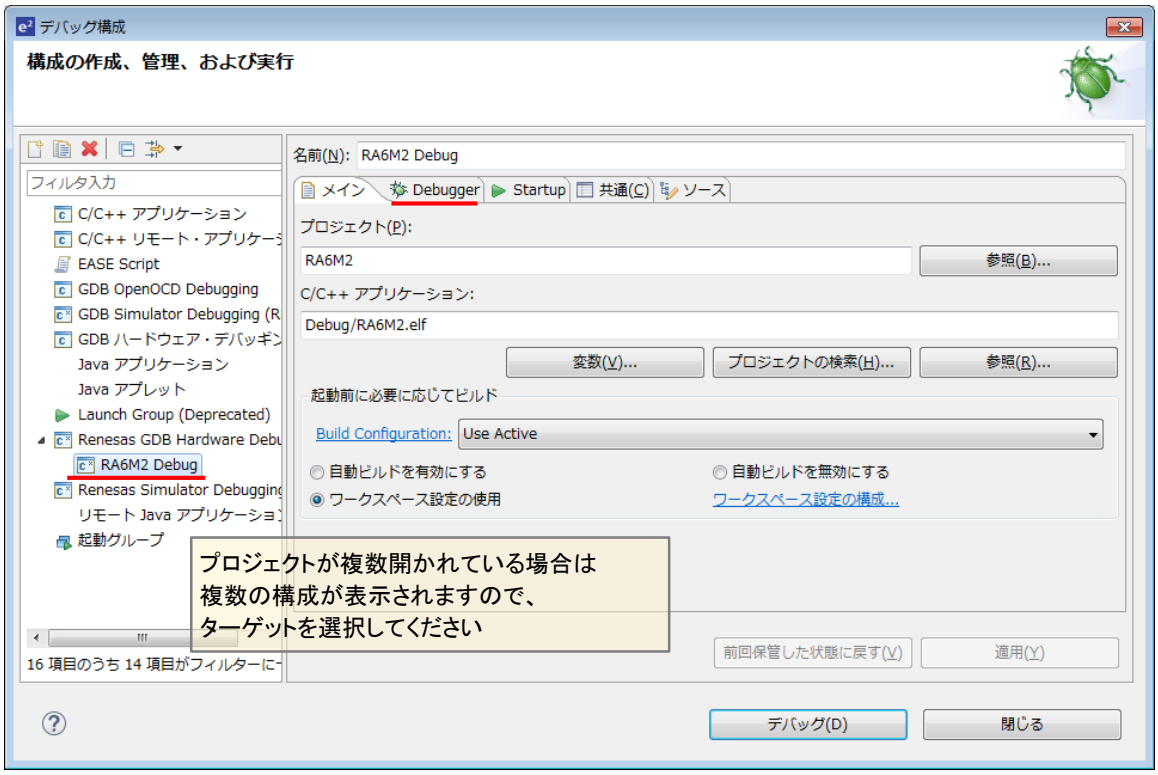
コンパイル時に最適化が行われず、変数やソースの実行順が追いやすくなるケースがあります。

10.4.デバッガ(e2studio)

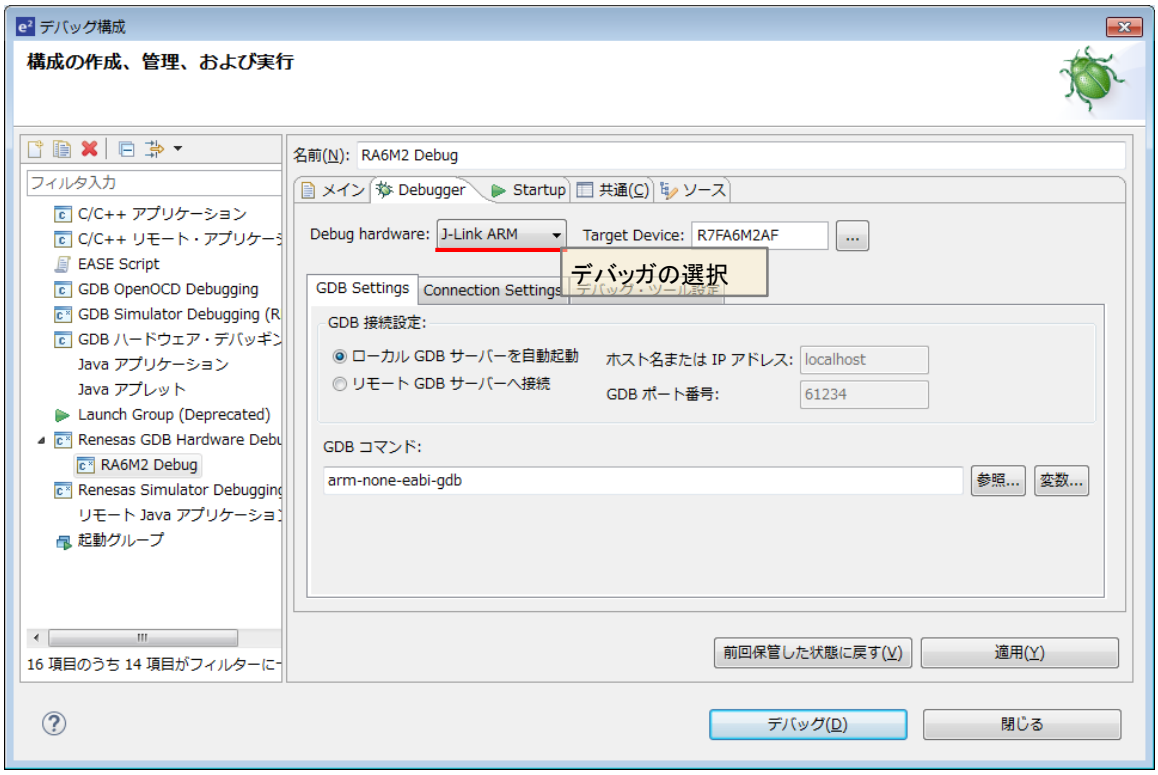
・デバッガの選択



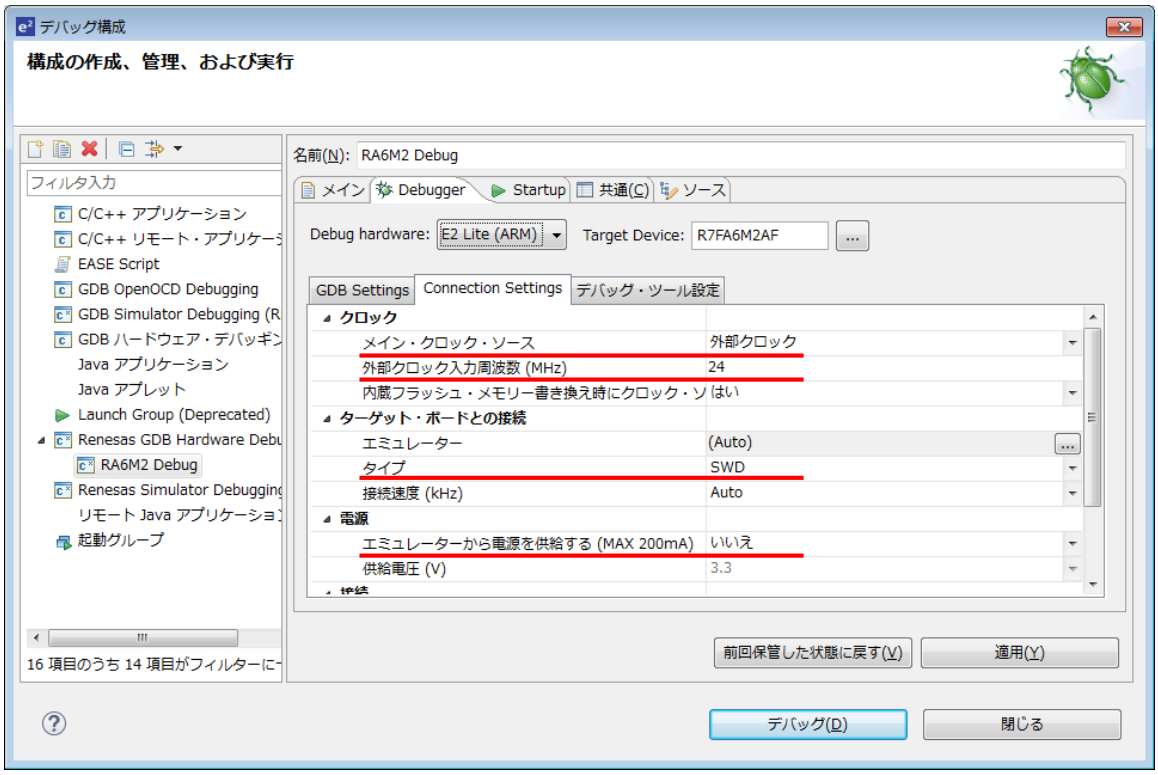
プロジェクトを右クリック、
デバッグ　ー　デバッグの構成



Debugger タブ



Debug hardware: の部分で、使用中のデバッガを選択してください。



メイン・クロック・ソース 「外部クロック」を選択してください（HSBRA6xxxxx, HSBRA4Mxxxxx の場合）

「内部クロック」を選択してください（HSBRA2xxxxx の場合）

外部クロック入力周波数(MHz) 「8」（HSBRA4M1F100 の場合）

「24」（HSBRA4M1F100 以外の RA6, RA4 マイコンボードの場合）

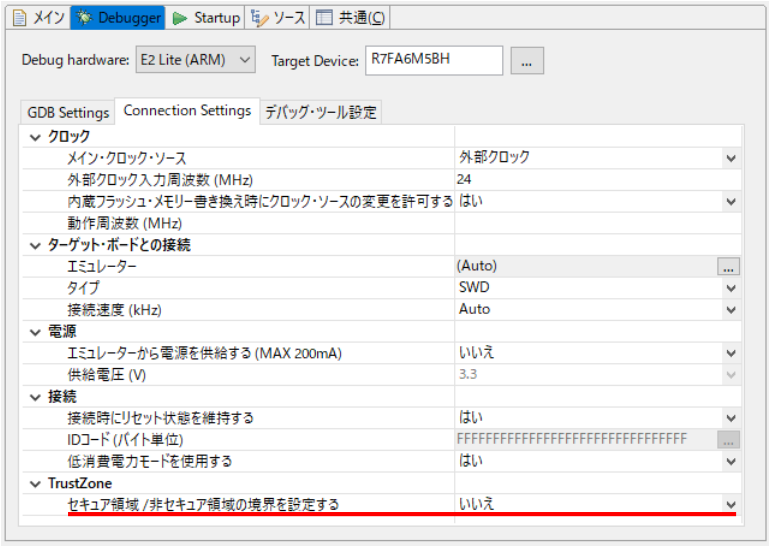
,

タイプ 「SWD」（USB-ADAPTER-RX14 と併用する場合）

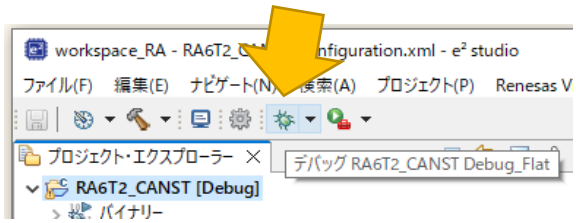
エミュレータから電源を供給する 「いいえ」

※選択するエミュレータによっては、選択項目が異なります

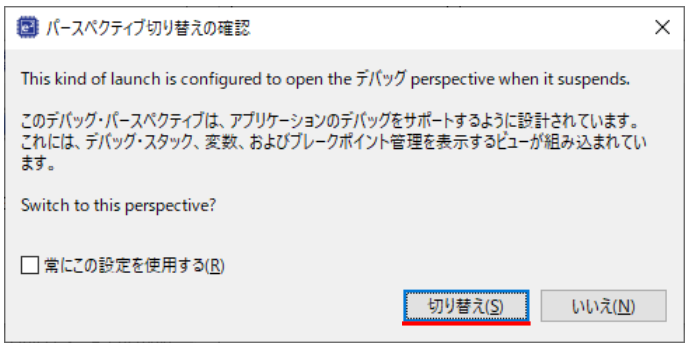
※TrustZone 対応マイコン(RA6M5, RA6M4, RA6T2, RA4M3 等）の場合



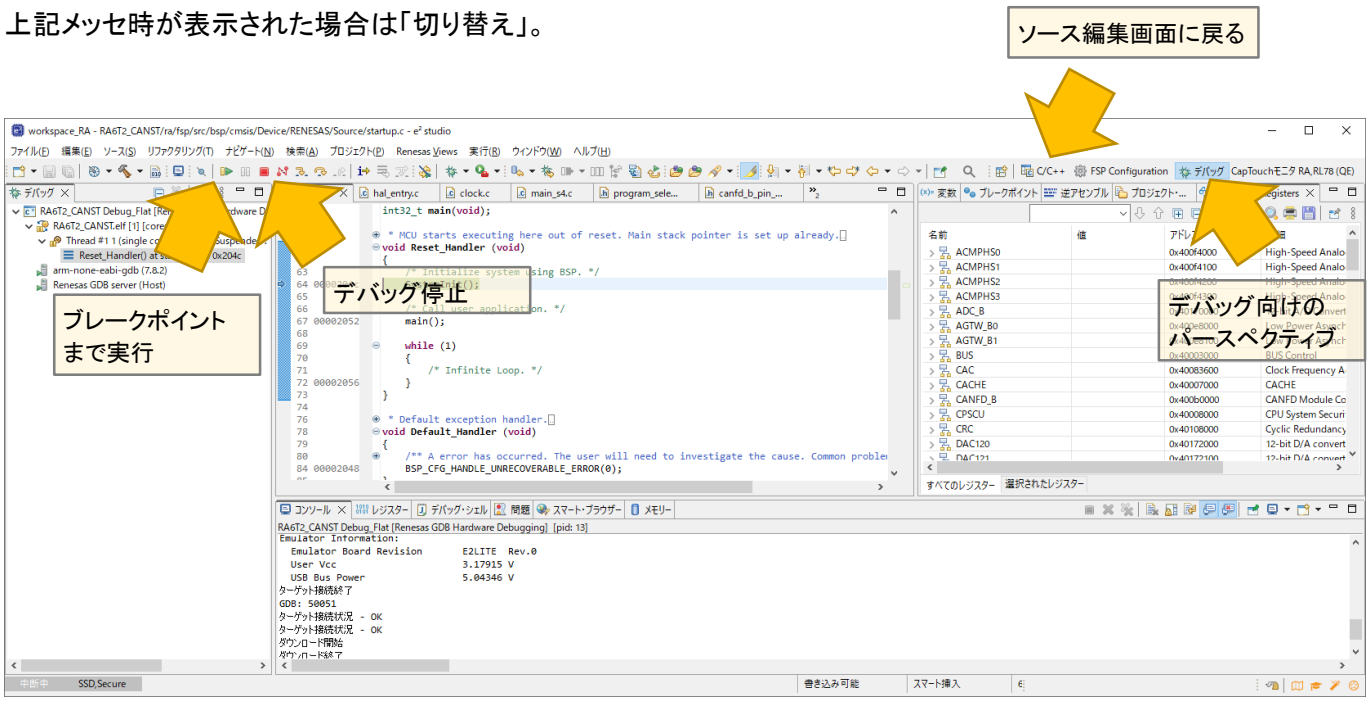
チップのセキュリティ領域設定を変更する必要がない場合は、上記項目を「いいえ」にしてください。



デバッグアイコンをクリック

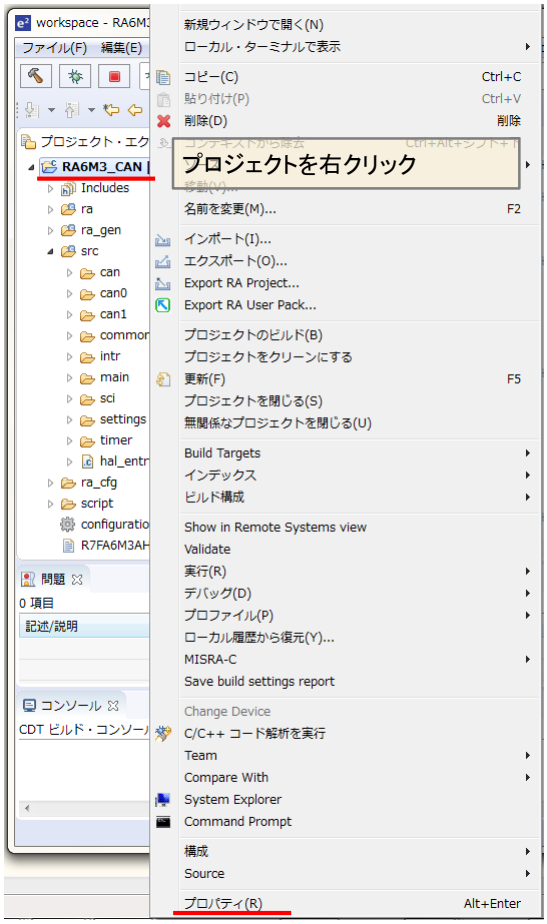


上記メッセ時が表示された場合は「切り替え」。

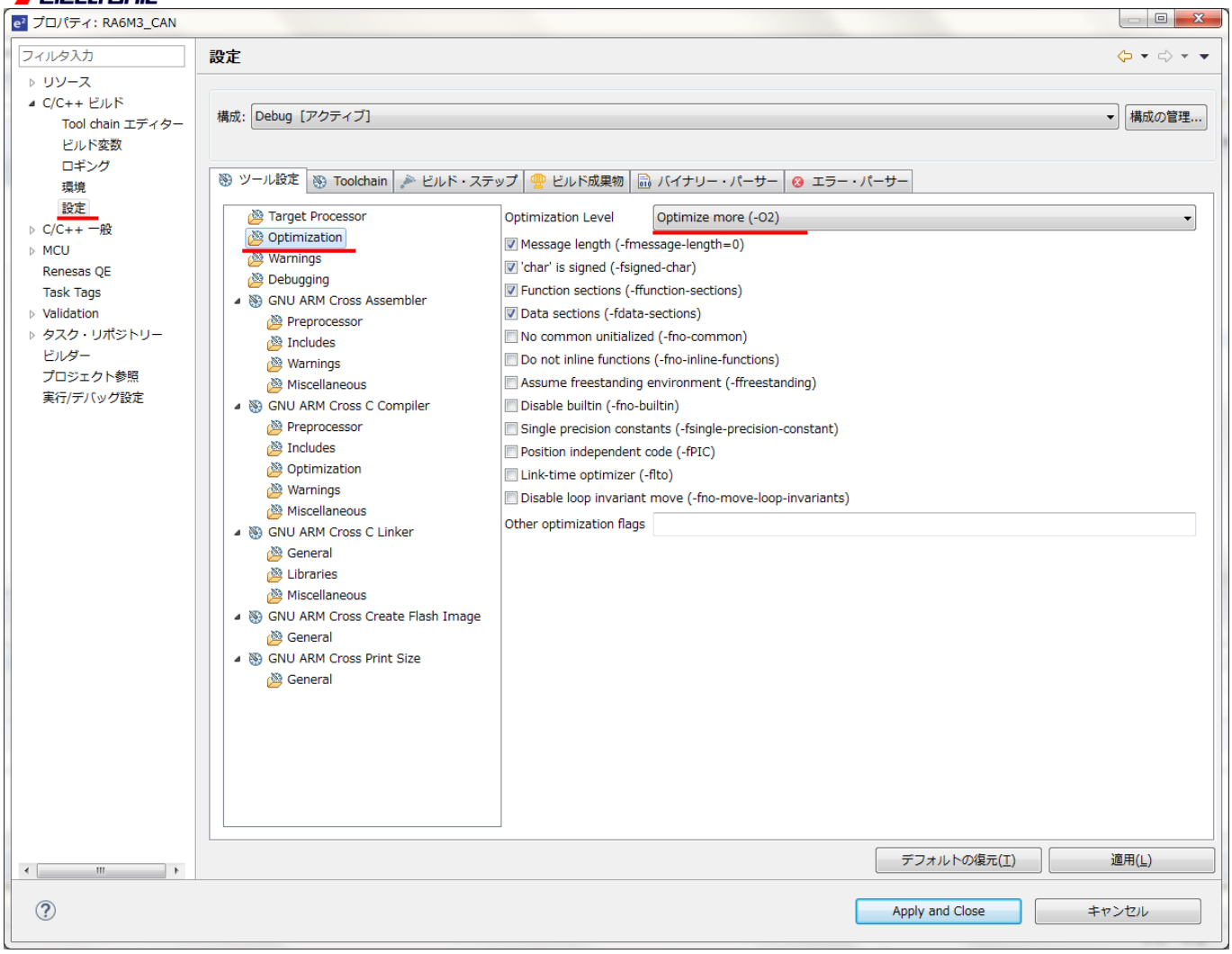


マイコンボードにプログラムがダウンロードされ、ブレークポイントの設定、ステップ実行やレジスタや変数値のモニタができます。

・ビルドの設定



プロジェクトのプロパティを開き、



C/C++ビルド ー 設定 ー ツール設定タブ ー Optimization

Optimization Level

で最適化の設定が変えられます。デバッグ時には、最適化を切る(-O0 を選択する)と、プログラムが追いかけ易くなるケースがあります。

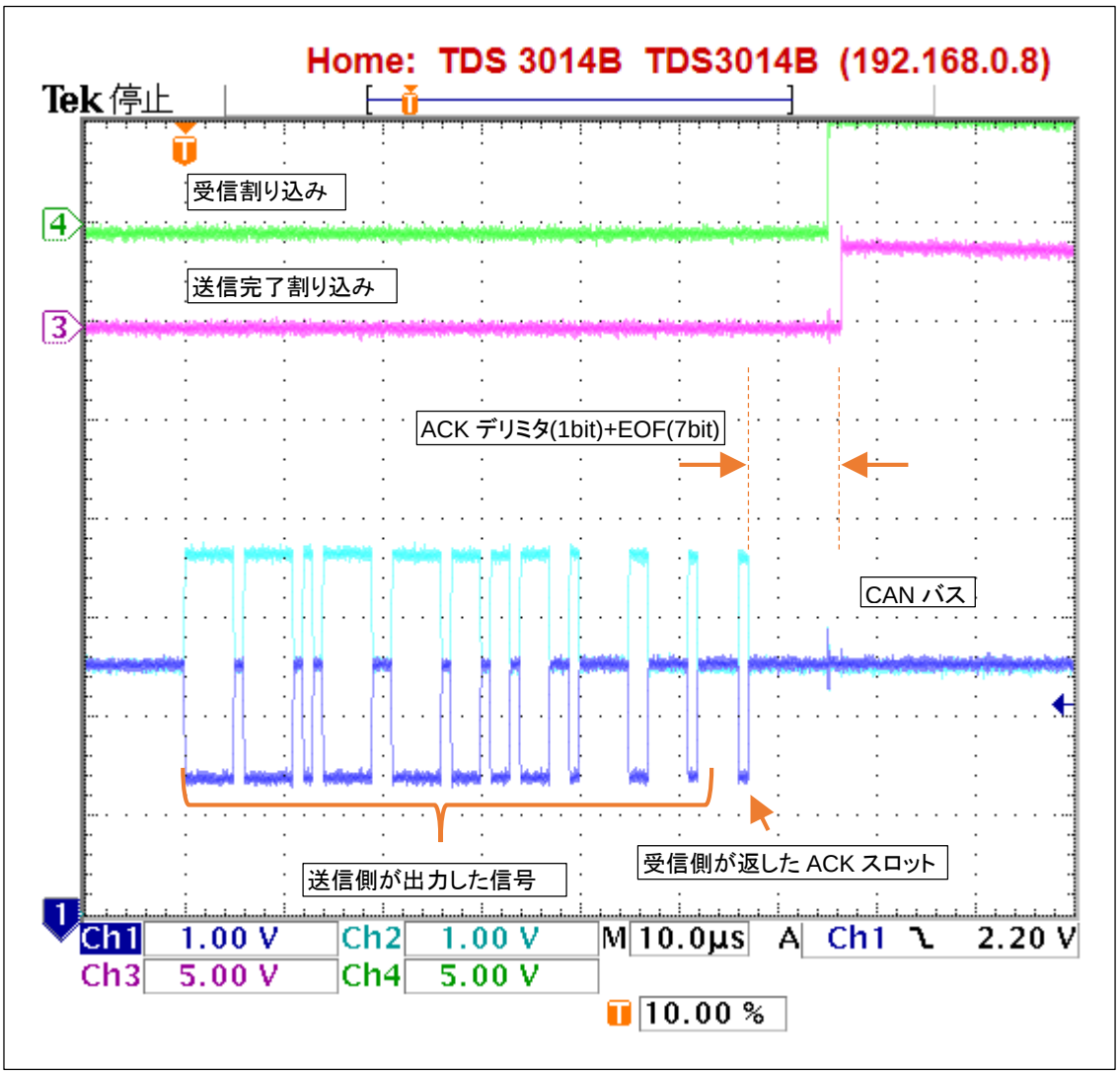
10.5.割り込みタイミングのデバッグに関して

受信、送信完了の割り込みがどのタイミングで生じているかをモニタするのが、定数定義

#define CAN_INTERRUPT_DEBUG

です。定義している場合、割り込みのタイミングで特定の I/O ポートを反転させます。

・データフレーム送信波形例（標準フォーマット）



上記の波形では、ACK スロットの後、8 ビット（1Mbps で 8 ビット→8us）後に、受信及び送信完了割り込みが観測されます。

※サンプルプログラムでは、割り込み発生時 I/O ポートを反転させているので、立下り波形となる場合もあります

・割り込みのモニタ機能で使用する I/O ポート

マイコンボード		CAN0				CAN1				CAN2			
	エラー	FIFO 受信	FIFO 送信	MBOX 受信	MBOX 送信	FIFO 受信	FIFO 送信	MBOX 受信	MBOX 送信	FIFO 受信	FIFO 送信	MBOX 受信	MBOX 送信
HSBRX71M176 HSBRX72M176 HSBRX72N176 HSBRX66N176	P96 J1-1	PD3 J1-2	P97 J1-3	PD4 J1-4	PD5 J1-5	PG0 J1-6	PD6 J1-7	PG1 J1-8	PD7 J1-9	P60 J1-10	P61 J1-11	P62 J1-13	P63 J1-13
HSBRX72N144 HSBRX66N144	P91 J1-1	P92 J1-2	P93 J1-3	PD0 J1-4	PD3 J1-7	PD4 J1-8	PD5 J1-9	PD6 J1-10	PD7 J1-11	P60 J1-12	P61 J1-13	P62 J1-14	P63 J1-15
HSBRX71M100 HSBRX72N100 HSBRX66N100	PE0 J1-1	PE3 J1-4	PE4 J1-5	PE5 J1-6	PE6 J1-7	PA1 J1-8	PA2 J1-9	PA7 J1-10	PA0 J1-11				
HSBRX72T144 HSBRX66T144	P90 J1-2	P92 J1-3	P63 J1-4	P94 J1-5									
HSBRX64MC	P65 J2-5	P64 J2-6	P63 J2-7	P62 J2-8	P61 J2-9	PE7 J2-11	PE6 J2-12	PE5 J2-13	PE4 J2-14	PE3 J2-15	PE2 J2-16	PE1 J2-17	PE0 J2-18
HSBRX65N176 HSBRX651F176	PJ3 J1-7	PJ5 J1-8	PF5 J1-9	P00 J1-10	P01 J1-11	P02 J1-12	P03 J1-13	P05 J1-14	P07 J1-17				
HSBRX651F144(A) HSBRX651F100(A) HSBRX671F144 HSBRX671F100	PE0 J2-5	PE1 J2-6	PE2 J2-7	PE3 J2-8	PE4 J2-9	PE5 J2-10	PE6 J2-12	PE7 J2-13	PA0 J2-17				
HSBRX65N144(A) HSBRX65N100(A)	PE0 J1-41	PE1 J1-42	PE2 J1-43	PE3 J1-44	PE4 J1-45	PE5 J2-3	PE6 J2-5	PE7 J2-6	PA0 J2-10				
HSBRX66T100A HSBRX66T100B	P80 J1-3	P11 J1-4	P10 J1-5	PE5 J1-6	P00 J1-7								

マイコンボード	送受信 FIFO 受信	グローバル 受信 FIFO	チャネル 送信	チャネル エラー	グローバル エラー
HSBRX24U144 HSBRX24U100 HSBRX24T100B	P91 J1-1	P92 J1-2	P93 J1-3	P94 J1-4	P95 J1-5
HSBRX231F100	P55 J1-5	P54 J1-6	P53 J1-7	P52 J1-8	P51 J1-9
HSBRX231E-B100	P12 J1-1	P13 J1-2	P14 J1-3	P15 J1-4	P16 J1-5
HSBRX140F80	PE2 J1-1	PE1 J1-2	PE0 J1-3	PD2 J1-4	PD1 J1-5
SmartRX	PA6 J1-1	PA4 J1-2	PA3 J1-3	PA1 J1-4	PE4 J1-5

マイコンボード	チャネル エラー	共通 FIFO 受信	グローバル エラー	受信 FIFO	チャネル 送信	受信バッファ
HSBRX660-144H	PE1 J1-1	PE0 J1-2	P64 J1-3	P63 J1-4	P62 J1-5	P61 J1-6
HSBRX660-100B	P07 J1-1	P05 J1-2	P06 J1-3	P03 J1-4	P04 J1-5	PJ3 J1-6
HSBRX261-100	PA0 J3-1	PA1 J3-2	PA2 J3-3	PA3 J3-4	PA4 J3-5	PA5 J3-6
HSBRX26T100	P82 J1-1	P81 J1-2	P80 J1-3	P11 J1-4	P10 J1-5	PE5 J1-6

マイコンボード	CAN0					CAN1				
	エラー	FIFO 受信	FIFO 送信	MBOX 受信	MBOX 送信	エラー	FIFO 受信	FIFO 送信	MBOX 受信	MBOX 送信
HSBA6M3F176	P408 J4-2	P409 J4-3	P410 J4-4	P411 J4-5	P412 J4-6	P301 J3-1	P302 J3-2	P303 J3-3	P304 J3-4	P305 J3-5
HSBRA6E1F100 HSBRA6M4F144 HSBRA6M2F144 HSBRA4M3F144 HSBRA4M2F100 HSBRA4M1F100	P408 J1-2	P409 J1-3	P410 J1-4	P411 J1-5	P412 J1-6	P413 J1-7	P414 J1-8	P415 J1-9	P708 J1-10	P709 J1-11
HSBRA6M1F100	P408 J1-2	P409 J1-3	P410 J1-4	P411 J1-5	P412 J1-6	P413 J1-7	P414 J1-8	P415 J1-9	P708 J1-10	P406 J1-24
HSBRA2A1F64	P012 J2-3	P013 J2-4	P014 J2-5	P015 J2-6	P502 J2-7					
HSBRA2L1F100 HSBRA2L1F64	P204 J1-3	P205 J1-4	P206 J1-5	P207 J1-6	P208 J1-7					
HSBRA6T1F100	P600 J3-2	P601 J3-3	P602 J3-4	P610 J3-5	P609 J3-6					
HSBRA4E1F64	P013 J2-1	P014 J2-2	P015 J2-3	P500 J2-4	P100 J2-5					

マイコンボード	CAN0 CFIFO 受信	CAN0 送信	CAN1 CFIFO 受信	CAN1 送信	RXFIFO	グローバル エラー	CAN0 チャネル エラー	CAN1 チャネル エラー
HSBA6M5F176	P408 J4-2	P409 J4-3	P410 J4-4	P411 J4-5	P412 J4-6	P413 J4-7	P414 J4-8	P415 J4-9

マイコンボード	送信	RXFIFO	受信バッファ	CFIFO 受信	グローバル エラー	チャネル エラー
HSBRA6T2F100	PB02 J1-1	PB01 J1-2	PB00 J1-3	PC05 J1-4	PC04 J1-5	PA07 J1-6
HSBRA6E2F64 HSBRA6T3F64 HSBRA4E2F64 HSBRA4T1F64	P013 J1-2	P014 J1-3	P015 J1-4	P006 J1-5	P008 J1-6	P500 J1-7

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2020.4.30	—	初版発行
REV.1.1.0.0	2020.9.15	P5, 8, 12, 17, 42, 53, 62, 86, 90	RX72N, RX66N 搭載ボードに関して追記
REV.1.2.0.0	2020.12.22	P7, 28, 35, 36, 44, 55, 63, 87	RA6T1, RA2L1 搭載ボードに関して追記
REV.1.3.0.0	2021.3.18	P7, 26, 29, 35, 36, 45, 64, 88, 97	RA6M4, RA4M3 搭載ボードに関して追記
		P63, 65	表記の追加、変更
REV.1.4.0.0	2021.6.9	P7, 29, 35, 37, 45, 64, 88, 97	RA6M2 搭載ボードに関して追記
REV.1.5.0.0	2021.9.14	P7,11,13,16,19,30, 31,33,39,40,49, 52,57,58,59,64, 69,70,72,73,74,76, 80,88,89,93,94,96, 97,98,101,102,103 ,106	RA6M5 搭載ボードに関して追記
REV.1.6.0.0	2022.4.14	P5,8,11,48,49 62,76,108,113	RX671, RX66T(100A)搭載ボードに関して追記
REV.1.7.0.0	2022.5.26	P7,30,37,40,41,49, 76	RA4E1 搭載ボードに関して追記
REV.1.8.0.0	2022.8.9	P4-9,12,13-15, 17,19-23,29-40, 45-52,58-61,66, 75-79	CANFD 対応、RA6T2, RX660 搭載ボードに関して追記 表記の追加、変更 サンプルプログラムの説明を CAN モジュール種別毎に分離（別マニュアル化）
REV.1.9.0.0	2022.9.29	P5,9,58,60,66,75	RX660 搭載ボードに関して記載を変更
REV.1.10.0.0	2023.2.1	P7,31,35,44,45,48, 49,58,61,75	RA6E1 搭載ボードに関して追記
REV.1.11.0.0	2023.6.23	P6-8,10,13-15,19,22,23,27-86,88,90,91,95,99, 100,104,105	RX140, RX26T, RA6E2, RA4E2, RA6T3, RA4T1 搭載ボードに関して追記 全体的に構成を見直し
REV.1.12.0.0	2024.1.24	P6,9,13,36,88,90, 95	RX23E-B 搭載ボードに関して追記
REV.1.13.0.0	2024.10.15		RA8 搭載ボードに関して追記 全体的に構成を見直し
REV.1.14.0.0	2024.11.29	P5,10,13,36,97, 158,192,197,206	RX261 搭載ボードに関して追記
REV.1.15.0.0	2025.1.22	P7,13,57,66,67,95, 160,195	RA8E1 搭載ボードに関して追記

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:https://www.hokutodenshi.co.jp

商標等の表記について

- 全ての商標及び登録商標はそれぞれの所有者に帰属します。
- パーソナルコンピュータを PC と称します。

ルネサス エレクトロニクス RX, RA マイコン搭載
HSB シリーズマイコンボード 評価キット

CAN スターターキット RX/RA
CAN スターターキット SmartRX
ソフトウェア編　マニュアル

株式会社 北斗電子
 2025 年 1 月 22 日改訂 REV.1.15.0.0 (250122)