CAN スタータキット RXV2
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 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.1.2.0.0

Hokuto Electronic

―目　次―

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 RXV2」「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）

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 から徐々にコード・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。

フォルダ			内容
SOURCE¥RX_CANST¥	RX65_2M¥		RX65 2MB 版向け プロジェクトフォルダ
	RX65_1M¥		RX65 1MB 版向け プロジェクトフォルダ
	RX71M¥		RX71M 176pin 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
	RX64M¥		RX64M 向け プロジェクトフォルダ
	RX231¥		RX231 向け プロジェクトフォルダ
	RX24U¥		RX24U/RX24T 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
	RX66T_100B¥		RX66T 100pin 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
	RX72T_144¥		RX72T 144pin 向け プロジェクトフォルダ
	RX72M¥		RX72M 176pin 向け プロジェクトフォルダ
	source¥can		CAN モジュール向け共通フォルダ
	source¥can0		CAN モジュール ch0
	source¥can1		CAN モジュール ch1
	source¥can2		CAN モジュール ch2
	source¥common		ボード設定
	source¥main_can		メイン関数(RSCAN 向け)
	source¥main_rscan		メイン関数(CAN 向け)
	source¥rscan		RSCAN モジュール
	source¥sci		SCI(UART)ドライバ
	source¥timer		コンペアマッチタイマドライバ
BINARY¥			コンパイル済み mot ファイル 格納フォルダ
DOCUMENT¥	CAN_STARTER_KIT_RXV2_REV_X_s.pdf		キット説明書
	CAN_STARTER_KIT_RXV2_Software_ REV_X_s.pdf		本資料

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 Ver8.01 以降を予めインストール願います。

マイコンボードへのプログラムの書き込みには、ルネサスエレクトロニクス RenesasFlashProgrammer が使用できますので、予めインストール願います。（CS+と E1 を使用してプログラムを書き込む際は、RenesasFlashProgrammer のインストールは必須ではありません）

1. サンプルプログラムの構成

1.1. CS+プロジェクトフォルダ

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
HSBRX65N176 HSBRX651F176 HSBRX65N144A HSBRX65N100A HSBRX651F144A HSBRX651F100A	RX65_2M
HSBRX65N144 HSBRX65N100 HSBRX651F144 HSBRX651F100	RX65_1M
HSBRX71M176	RX71M
HSBRX71M100	RX71M_100
HSBRX64MC	RX64M
HSBRX24U144 HSBRX24U100 HSBRX24T100B	RX24U
HSBRX231F100	RX231
SmartRX!!!	SmartRX
HSBRX66T100B	RX66T_100B
HSBRX66T144	RX66T_144
HSBRX72T144	RX72T_144
HSBRX72M176	RX72M

1.2. CAN モジュール種別

RX マイコンが搭載している、CAN モジュールは 2 タイプに別れています。

CAN モジュール	マイコン種	特徴
CAN	RX65 RX71M RX64M RX66T RX72T RX72M	・「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 ・最大 1Mbps
RSCAN	RX24U RX24T RX231	・現行の RX200 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

RX600/700 系のマイコンでは、「CAN モジュール」が搭載されており、RX200 系のマイコンでは RSCAN モジュールが搭載されています。この 2 タイプのモジュールは、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、CAN 系と RSCAN 系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。

なお、CAN モジュールと RSCAN モジュール間の通信は行えます。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。

1.3. ソースファイルフォルダ

SOURCE¥RX_CANST¥source

以下に、実際のソースファイルが格納されています。

フォルダ	説明
source¥can	CAN モジュール向け共通フォルダ
source¥can0	CAN モジュール ch0
source¥can1	CAN モジュール ch1
source¥can2	CAN モジュール ch2
source¥common	ボード設定
source¥main_can	メイン関数（RSCAN 向け）
source¥main_rscan	メイン関数（CAN 向け）
source¥rscan	RSCAN モジュール
source¥sci	SCI(UART)ドライバ
source¥timer	コンペアマッチタイマドライバ

青字のフォルダは、CAN モジュールを搭載したマイコンで使用されています。

緑字のフォルダは、RSCAN モジュールを搭載したマイコンで使用されています。

それ以外は、共通で使用されています。

各フォルダには、サンプルプログラム毎(SAMPLE*n*)にソースが分かれているものがあり、その場合はファイル名に

_s*n*

が付いています。

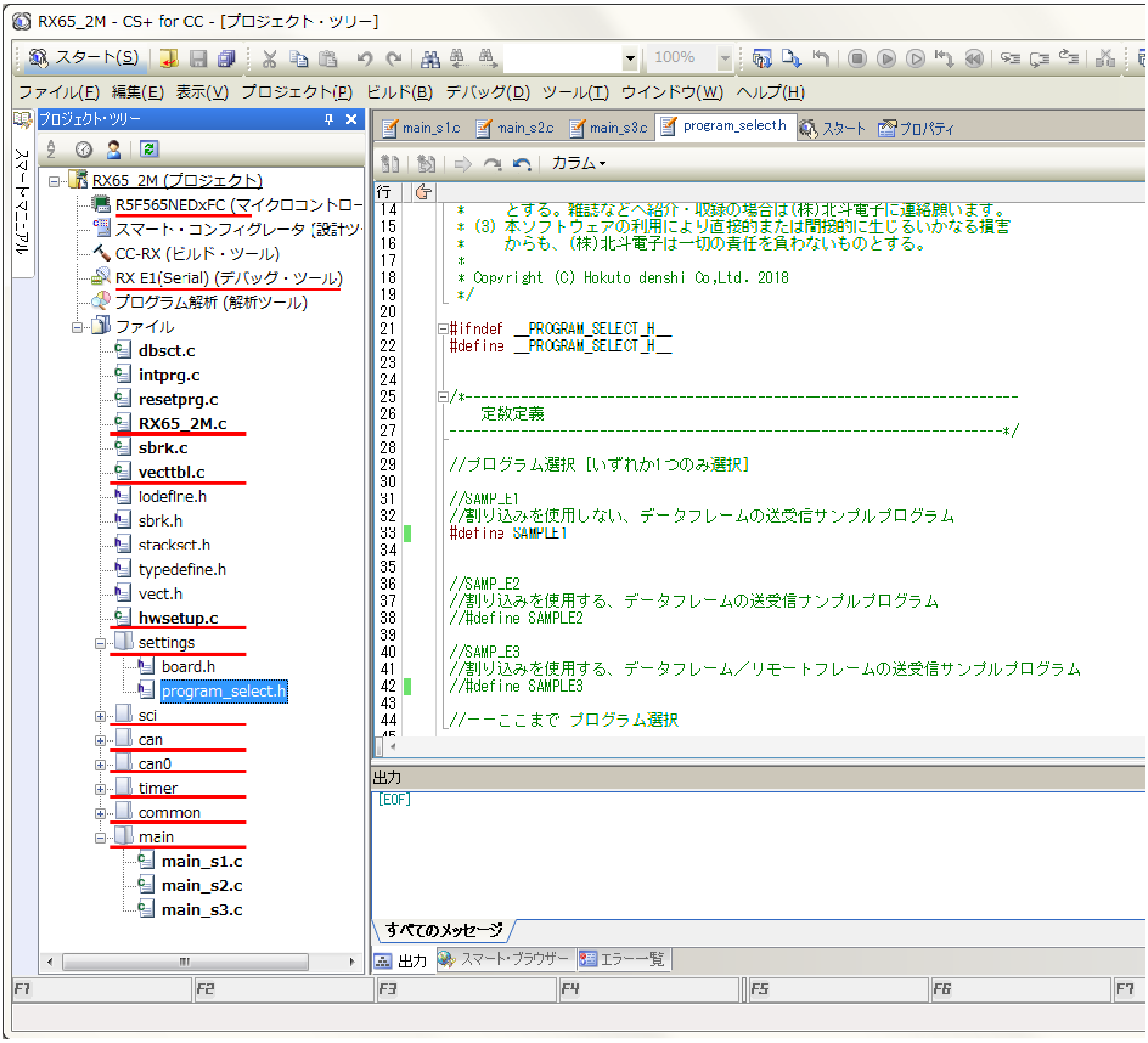
フォルダ	ファイル名
source¥can¥	can_s1.h can_s2.h can_s3.h
source¥can0¥	can_ch0_s1.c can_ch0_s1.h can_ch0_s2.c can_ch0_s2.h can_ch0_s3.c can_ch0_s3.h
source¥can1¥	（同上）
source¥can2¥	（同上）
source¥main_can	main_s1.c main_s2.c main_s3.c
source¥main_rscan	main_s1.c main_s2.c main_s3.c
source¥rscan	can_s1.c can_s1.h can_s2.c can_s2.h can_s3.c can_s3.h

サンプルプログラム毎に、別なファイルが読み込まれる様になっていますので、サンプルプログラムを変更する場合は、(SAMPLE*n*)に応じたファイルを変更してください。

1.4. 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_2M 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_2M 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）

・RX E1(Serial)（デバッグ・ツール）

デバッガを使用する場合は、使用するデバッガの種類に応じて変更してください。

JTAG を選択した場合は、デバッガと USB-ADAPTER-RX14 との同時使用はできませんので、JTAG/Serial の両方が選べるタイプのマイコンでは、Serial の方を選択してください。

デバッガを接続しない場合は変更する必要はありません。

・プロジェクト名.c（上記では RX65_2M.c）

[メイン関数呼び出し様テンプレート]

選択した(SAMPLEn)に対応するメイン関数を呼び出します。

・vecttbl.c

[ベクタテーブル定義ファイル]

このファイルで定義されている全ての割り込みの中から、本サンプルプログラムで使用する割り込みの部分をコメントアウトしています。

※RX66T, 72T, 72M 向けのソースでは、プロジェクトフォルダ以下の vecttbl.c は使用していません

・hwsetup.c

[ハードウェア初期化ファイル]

クロックの設定等、起動後に設定される基本的な設定を行っています。このファイルはマイコン種毎に別になっています。

※RX66T, 72T, 72M 向けのソースでは、プロジェクトフォルダ以下の hwsetup.c は使用していません

・settings カテゴリ

このカテゴリの下には、以下のファイルが配置されています。

board.h

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

//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
//RX64M
#define HSBRX64MC       3
//RX65
#define HSBRX65N176     4
#define HSBRX651F176    5
#define HSBRX65N144     6
#define HSBRX651F144    7
#define HSBRX65N144A    8
#define HSBRX651F144A   9
#define HSBRX65N100     10
#define HSBRX651F100    11
#define HSBRX65N100A    12
#define HSBRX651F100A   13
//RX231
#define HSBRX231F100    14
#define SMART_RX        15
//RX24U
#define HSBRX24U144     16
#define HSBRX24U100     17
//RX24T
#define HSBRX24T100B    18
//RX66T
#define HSBRX66T100B    19
#define HSBRX66T144     20
//RX72T
#define HSBRX72T144     21
//RX72M
#define HSBRX72M176     22

//ーーここまで マイコンボードタイプ定義

#define BOARD_TYPE HSBRX66T100B

//上記 BOARD_TYPE に応じて端子やクロック設定を選択する場合 AUTO_BOARD_SETTING を定義
//端子設定等をマニュアルで選択する場合、source¥common¥board_seting.h を書き換えてください
#define AUTO_BOARD_SETTING
```

マイコンのボード種を定義しているファイルです。BOARD_TYPE 定義を実際に使用するマイコンボードの型名に変更してください。

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

//ーーここまで プログラム選択
```

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。

・sci カテゴリ
SOURCE¥RX_CANST¥source¥sci 以下の
sci.c
sci.h
ファイルが含まれます。このファイルは、全プロジェクトで共通です。
USB-ADAPTER-RX14 を通して、PC と通信を行うプログラムが含まれます。
SCI1 を使用して、115,200bps, 8bit, ストップビット 1bit, パリティなしの条件で SCI(UART)の通信を行います。

・can カテゴリ[CAN モジュール系のみで使用]
SOURCE¥RX_CANST¥source¥can 以下の
can_s1.h
can_s2.h
can_s3.h
ファイルが含まれます。このファイルは、CAN モジュール系プロジェクトで共通です。
CAN モジュールのチャネル(ch0~ch2)に関係ない定義が記載されているファイルです。
サンプルプログラム毎に、ファイルが分かれています。program_select.h で選択した(SAMPLE***n***)に対応する
can_s***n***.h ファイルの中身が使用されます。

・can0 カテゴリ[CAN モジュール系のみで使用]
SOURCE¥RX_CANST¥source¥can0 以下の
can_ch0_s1.c
can_ch0_s1.h
can_ch0_s2.c
can_ch0_s2.h
can_ch0_s3.c
can_ch0_s3.h
ファイルが含まれます。このファイルは、CAN モジュール系で使用されます。
ch0, ch1, ch2 とチャンネルに応じたファイルがありますので、使用する CAN のポートに合わせて、プロジェクトに含めるようにしてください。
※RX65_2M, RX65_1M, RX66T_100B, RX66T_144, RX72T_144 のプロジェクトでは、can0 のみ、プロジェクトに含まれますが、ボード上に ch2 のポートが出ている、RX71M や RX72M では、ch2 が、また RX64M では、ch1, ch2 のファイルがプロジェクトに追加されています
(RX65 系マイコンで ch1 の CAN を使用する場合は、SOURCE¥RX_CANST¥source¥can1 以下のファイルをプロジェクトに追加してください)
このファイルも、サンプル毎(SAMPLEn)で別になっています。

・timer カテゴリ
SOURCE¥RX_CANST¥source¥timer 以下の
cmt.c
cmt.h
ファイルが含まれます。このファイルは、全プロジェクトで共通です。
マイコンのコンペアマッチタイマ(CMT)の機能を用い、シリアル(SCI1)の文字出力処理を定期的に実行しています。

・common カテゴリ
SOURCE¥RX_CANST¥source¥common 以下の

board_setting.h
ファイルが含まれます。このファイルは、全プロジェクトで共通です。

board_setting.h

```
//AUTO_BOARD_SETTINGを使用しない場合の設定

//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

//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

//マイコンタイプ
#define MCU_TYPE    MCU_TYPE_RX65_2M

//CANマクロ種別(RX6xx/RX71M系の"CAN_MACRO_TYPE_CAN", RX2xx系の"CAN_MACRO_TYPE_RSCAN"のいずれか)
#define CAN_MACRO    CAN_MACRO_TYPE_CAN

//CANのチャネル数
#define CAN_CH    2

//CANを有効にするチャネル(RX6xx/RX71M系でのみ使用)
#define CAN0_VALID  1        //CAN0 有効
#define CAN1_VALID  0        //CAN1 無効
#define CAN2_VALID  0        //CAN2 無効

//PCLKBのクロック周波数[MHz]
#define PCLKB    60

//ピン数
#define PIN_NO    176
```

settingsカテゴリのboard.hで

#define AUTO_BOARD_SETTING

を有効にしている場合は、編集する必要はありません。各種設定を手動で行う場合、board.h 内の #define AUTO_BOARD_SETTING をコメントアウトして、本ファイルで設定してください。

・mainカテゴリ

SOURCE¥RX_CANST¥source¥main_can 以下の

main_s1.c

main_s2.c

main_s3.c

ファイルが含まれます。このファイルは、CANモジュール系で使用されます。

このファイルも、サンプル毎(SAMPLEn)で別になっています。

※RSCAN系マイコンでは、SOURCE¥RX_CANST¥source¥main_rscan

以下のファイルが、本カテゴリの下に含められています

2. 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。

(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 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_2M では、[COPY 先]¥RX_CANST¥RX65_2M¥RX65_2M.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

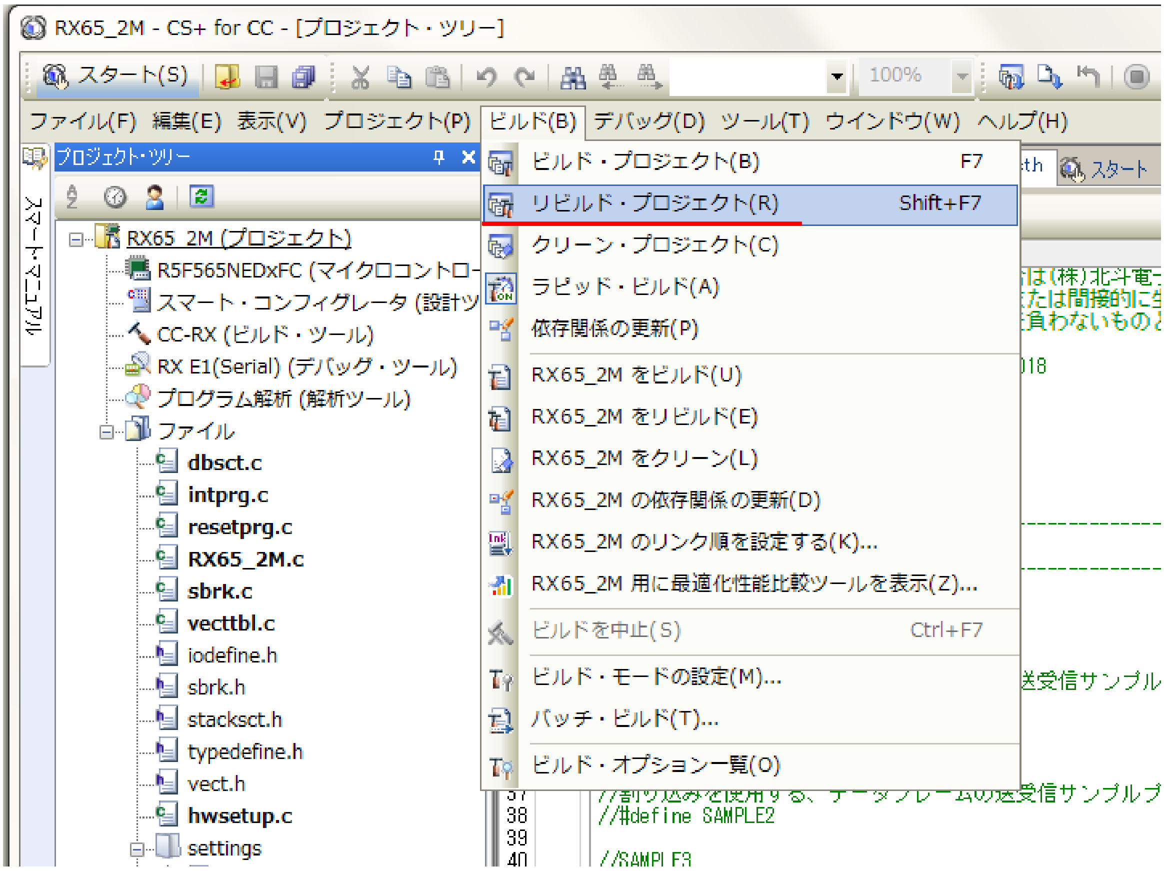
settings カテゴリに含まれる、program_select.h を開き、編集してください。

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

//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。

(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_2M では、[COPY 先]¥RX_CANST¥RX65_2M¥DefaultBuild¥RX65_2M.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、キットのマニュアルの 5 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 E1 デバッガ（E2, E2Lite, E20）をお持ちの場合は、デバッガを USB-ADAPTER-RX14 の 14P コネクタに接続し、ビルド後、

デバッグ－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。

3. サンプルプログラムの説明(共通部分)

3.1. 初期化設定

サンプルプログラムの初期化は、

hwsetup.c

で行っています。初期化部は主にクロックの設定です。（RX66T,72T,72M は、スマートコンフィグレータで初期化コードを生成しています）

RX マイコンでは、リセット後低速オンチップオシレータ(LOCO)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

・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）

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。

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
HSBRX65N176 HSBRX651F176 HSBRX65N144A HSBRX65N100A HSBRX651F144A HSBRX651F100A	24MHz	10(=240MHz)	120MHz	60MHz	PCLKB/5 =12MHz
HSBRX65N144 HSBRX65N100 HSBRX651F144 HSBRX651F100	24MHz	10(=240MHz)	120MHz	60MHz	PCLKB/5 =12MHz
HSBRX71M176	24MHz	10(=240MHz)	240MHz	60MHz	PCLKB/5 =12MHz
HSBRX71M100	24MHz	10(=240MHz)	240MHz	60MHz	PCLKB/5 =12MHz
HSBRX64MC	24MHz	10(=240MHz)	120MHz	60MHz	PCLKB/5 =12MHz
HSBRX24U144 HSBRX24U100 HSBRX24T100B	10MHz	8(=80MHz)	80MHz	40MHz	PCLKB/4 =10MHz
HSBRX231F100 SmartRX!!!	8MHz	13.5(8/2× **13.5**=54MHz)	54MHz	27MHz	8MHz (=MainOSC)
HSBRX66T100B	20MHz	16(20/2 × **16**=160MHz)	160MHz	40MHz	PCLKB/4 =10MHz
HSBRX66T144	24MHz	20(24/3 × **20**=160MHz)	160MHz	40MHz	PCLKB/4 =10MHz
HSBRX72T144	24MHz	25(24/3 × **25**=200MHz)	200MHz	50MHz	PCLKB/5 =10MHz

※CAN のクロック設定は、hwsetup.c ではなく、can0~can2, rscan 以下のソースで行っています

4. サンプルプログラムの説明(SAMPLE1)

4.1. プログラム仕様

・データフレームの送信
・データフレームの受信

を行うサンプルプログラムとする。

・CAN ID は標準フォーマット(11bit)とする（拡張フォーマットのデータは受信しない）
・送信に使用する ID は 0x0000~0x0007 までの 8 種
・受信する ID は、0x0000~0x0007 までの 8 種（それ以外の ID のデータは受信しない）
・リモートフレームは受信しない
・端末のキーボード入力を読み取り 0~7 の入力に応じて ID，送信データバイト数を変えて送信する

ーキーボードから入力したキーと送信データの関係ー

キーボードからの入力	ID	送信バイト数	送信データ
0	0x0000	1	0x 01
1	0x0001	2	0x 01 23
2	0x0002	3	0x 01 23 45
3	0x0003	4	0x 01 23 45 67
4	0x0004	5	0x 01 23 45 67 89
5	0x0005	6	0x 01 23 45 67 89 AB
6	0x0006	7	0x 01 23 45 67 89 AB CD
7	0x0007	8	0x 01 23 45 67 89 AB CD EF

4.2. 関数仕様

4.2.1. ユーザ関数

can*n*_init　[CAN モジュール系](n=0~2)
can_init　[RSCAN モジュール系]

概要：初期化関数
宣言：

int can0_init(void)　[ch0 向け]
int can1_init(void)　[ch1 向け]
int can2_init(void)　[ch2 向け]
int can_init(void)

説明：
・モジュールストップ解除
・端子設定
・通信設定

を行います

引数:

　　なし

戻り値:

　　0: 正常終了

初期化関数は、CAN を 1Mbps 設定で初期化します。

cann_mb_set　[CAN モジュール系](n=0~2)

概要:メールボックス設定関数

宣言:

　　int can0_mb_set(unsigned char mb, unsigned short id, unsigned char direction)　[ch0 向け]

　　int can1_mb_set(unsigned char mb, unsigned short id, unsigned char direction)　[ch1 向け]

　　int can2_mb_set(unsigned char mb, unsigned short id, unsigned char direction)　[ch2 向け]

説明:

・メールボックスの設定

を行います

引数:

　　mb: メールボックス番号(0~31)

　　id: ID を指定

　　direction: 送受信区分を設定

　　　CAN_DIRECTION_SEND(0)　送信時は 0 指定

　　　CAN_DIRECTION_RECEIVE(1)　受信時は 1 指定

戻り値:

　　0: 正常終了

　　-1: 引数エラー

　　-2: 送信、受信アボート処理失敗

can_receive_rule_set　[RSCAN モジュール系]

概要:受信ルール設定関数

宣言:

　　int can_receive_rule_set(unsigned char num, unsigned short id)

説明:

・受信ルール設定

を行います

引数:

num: 受信ルール番号(0~15)

id: ID を指定

direction: 送受信区分を設定

戻り値:

0: 正常終了

-1: 引数エラー

-2: 受信ルール設定不可の動作モード

CAN モジュール系では、「メールボックス」の設定。RSCAN モジュール系では「受信ルール」の設定を行う関数となっています。モジュールの種別により、設定項目が異なりますので、別な関数となっています。

can_start [RSCAN モジュール系]

概要:動作モード変更関数

宣言:

int can_start(void)

説明:

・CAN モジュールを動作モードに移行する

を行います

引数: なし

戻り値:

0: 正常終了

-2: 動作モード変更 NG

RSCAN モジュール系では、受信ルール設定完了後、本関数を呼び出す事により、CAN モジュールを動作モードに変更します。

can*n*_send [CAN モジュール系](n=0~2)

概要:データ送信関数

宣言:

int can0_send(unsigned char mb, unsigned char dlc, unsigned char *data) [ch0 向け]

int can1_send(unsigned char mb, unsigned char dlc, unsigned char *data) [ch1 向け]

int can2_send(unsigned char mb, unsigned char dlc, unsigned char *data) [ch2 向け]

説明:

・データフレームの送信

を行います

引数:

mb: 使用するメールボックスを指定します(0~31)

dlc: 送信バイト数を指定します(1~8)

*data: 送信するデータを指定します

戻り値:

0: 正常終了

-1: 引数チェックエラー

-2: レジスタ値設定タイムアウト

-3: メールボックスが受信に設定されている

-4: メールボックスが現在送信中

can_send [RSCAN モジュール系]

概要:データ送信関数

宣言:

int can_send(unsigned char num, unsigned short id, unsigned char dlc, unsigned char *data)

説明:

・データフレームの送信

を行います

引数:

num: 送信バッファ番号(0~3)

id: 送信する ID を指定します

dlc: 送信バイト数を指定します(1~8)

*data: 送信するデータを指定します

戻り値:

0: 正常終了

-1: 引数チェックエラー

-3: 送信バッファ使用中

CAN モジュール系は、メールボックスと送信 ID を既に設定済みなので、送信関数ではメールボックス番号の指定のみです。RSCAN モジュール系は、送信関数を呼び出す際に、ID を指定します。

cann_receive　　[CANモジュール系](n=0~2)

概要：受信関数

宣言：

int can0_receive(unsigned char mb, unsigned short *id, unsigned char *data, unsigned short *ts)
[ch0 向け]
int can1_receive(unsigned char mb, unsigned short *id, unsigned char *data, unsigned short *ts)
[ch1 向け]
int can2_receive(unsigned char mb, unsigned short *id, unsigned char *data, unsigned short *ts)
[ch2 向け]

説明：

・データの受信

を行います

引数：

mb: メールボックス(0~31)
*id: 受信した ID を格納するポインタ(unsigned short x1)
*data: 受信したデータを格納するポインタ（最大 unsigned char x8）
*ts: データ受信時のタイムスタンプ(unsigned short x1)

戻り値：

0: 受信データなし
1~8: 受信したデータのバイト数
0x1x: (b4=1)受信メールボックス更新中
0x2x: (b5=1)オーバライドフラグが立っている（受信操作前に上書きされたメッセージあり）
-1: 引数チェックエラー
-2: レジスタ設定タイムアウト
-3: メールボックスが受信用に設定されていない

本関数を呼び出すと、指定したメールボックスにメッセージが格納されているかを調べ、メッセージが無ければ 0 を返す。メッセージがあれば、引数にメッセージの中身をコピーして、受信したバイト数を関数の戻り値として返す。

can_receive　　[RSCAN モジュール系]

概要：受信関数

宣言：

int can_receive(unsigned char num, unsigned short *id, unsigned char *data, unsigned short *ts)

説明：

・データの受信

を行います

引数：

num: 受信ルール番号（＝受信バッファ番号）(0~15)

*id: 受信した ID を格納するポインタ(unsigned short x1)

*data: 受信したデータを格納するポインタ（最大 unsigned char x8）

*ts: データ受信時のタイムスタンプ(unsigned short x1)

戻り値：

0: 受信データなし

1~8: 受信したデータのバイト数

-1: 引数チェックエラー

-3: 受信ルールが設定されていない

関数の動作としては、CAN モジュール系と同じです。

CAN モジュールの受信関数は、メールボックス番号を指定し、RSCAN モジュールの受信関数は、受信ルール番号（＝受信バッファ番号で設定）を指定する様になっています。

4.3. プログラムで使用している変数・定数

4.3.1. 定数定義

・[CAN モジュール系]

```
#define CAN_DIRECTION_SEND  0
#define CAN_DIRECTION_RECEIVE  1
```

メールボックス設定時、送受信区分を指定する定数。

```
#define CAN_MESSAGE_NOT_OVERRIDE  0
#define CAN_MESSAGE_OVERRIDE  1
```

データ受信時、上書きされたメッセージの有無を示す定数

```
#define CAN_ABORT_WAIT  10000  //約 750us のウェイト
```

メールボックスの動作を変更する際のタイムアウト設定。

```
#define CAN_LOOP_TIMEOUT  1000  //レジスタ書き換えタイムアウト
```

レジスタ値の書き換えタイムアウト設定。

4.4. メールボックス、送受信バッファの設定

4.4.1. メールボックス設定(CAN モジュール系)

メールボックス 番号	ID	送受信区分
0	0x0000	送信
1	0x0001	送信
2	0x0002	送信
3	0x0003	送信
4	0x0004	送信
5	0x0005	送信
6	0x0006	送信
7	0x0007	送信
8	0x0000	受信
9	0x0001	受信
10	0x0002	受信
11	0x0003	受信
12	0x0004	受信
13	0x0005	受信
14	0x0006	受信
15	0x0007	受信

32 個あるメールボックスのうち、0~8 を送信用、9~15 を受信用に設定。メールボックス番号と ID をメールボックス設定の時点で紐付け。

4.4.2. 受信バッファ設定(RSCAN モジュール系)

受信ルール 番号	ID	受信バッファ 番号
0	0x0000	0
1	0x0001	1
2	0x0002	2
3	0x0003	3
4	0x0004	4
5	0x0005	5
6	0x0006	6
7	0x0007	7

8 個の受信ルールを設定。受信ルールに合致したメッセージは、受信バッファ 0~7 に格納。

4.5. 動作説明(CAN モジュール系)

実際に CAN の通信を行う際、4.2 記載の関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can0_init();

CAN の ch0 を初期化する。

(2)メールボックスの設定

```
//      メールボックス番号  ID  送受信区分
can0_mb_set(0, 0x000, CAN_DIRECTION_SEND);//送信
…
can0_mb_set(7, 0x007, CAN_DIRECTION_SEND);//送信

can0_mb_set(8, 0x000, CAN_DIRECTION_RECEIVE);//受信
…
can0_mb_set(15, 0x007, CAN_DIRECTION_RECEIVE);//受信
```

メールボックス 0~7 を送信設定。メールボックス 8~15 を受信設定する例。メールボックス設定時、ID は、メールボックス番号と紐付けされる。

(3)データの受信

```
for(i=8; i<16; i++)//MB[8]-[16]が受信設定
{
   r_ret = can0_receive(i, &r_id, &r_data[0], &r_ts);
}
```

r_ret が 0 ならば、受信したデータはなし。1~8 であれば、戻り値に対応するバイト数のデータを受信しています。

(4)データの送信

```
unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_mb=0;//使用するメールボックス番号
s_dlc=1;//送信バイト数
```

s_ret = **can0_send**(s_mb, s_dlc, &s_data[0]);

メールボックス 0 から（ID=0x0000）、0x01 を 1 バイト送信する。

4.6. 動作説明（RSCAN モジュール系）

実際に CAN の通信を行う際、4.2 記載の関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can_init();

CAN の初期化をする。

(2)受信ルールの設定

```
//      受信ルール番号   ID
can_receive_rule_set(0, 0x00);
…
can_receive_rule_set(7, 0x07);

can_start();
```

受信ルール番号 0~7 を受信設定する例。ID は、受信ルール番号と紐付けされる。ルール設定後、can_start を呼び出す。

(3)データの受信

```
for(i=0; i<8; i++)  //受信ルール 0~7 が設定済み
{
   r_ret = can_receive((unsigned char)i, &r_id, &r_data[0], &r_ts);
}
```

r_ret が 0 ならば、受信したデータはなし。1~8 であれば、戻り値に対応するバイト数のデータを受信しています。

(4)データの送信

```
unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_buf=0; //使用する送信バッファ番号
s_id=0x000; //ID
```

s_dlc=1; //送信バイト数

s_ret = **can_send**(s_buf, s_id, s_dlc, &s_data[0]);

送信バッファ 0 から ID=0x000、データ 0x01 を 1 バイト送信する。

4.7. 送信データ

送信関数呼び出し時、CAN のポートからは、以下のようなデータ列が出力され、CAN バス（CANH, CANL のライン）に流れます。

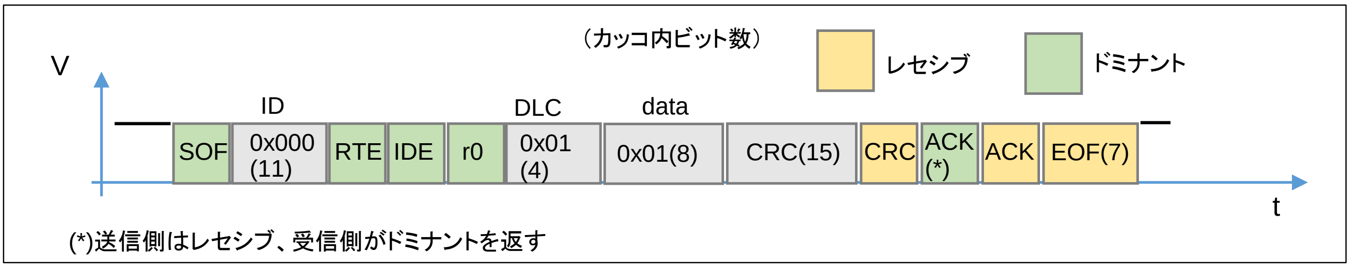


図 4-1 CAN データフレームのデータパケット

CAN モジュール系、RSCAN モジュール系どちらのモジュールも、出力するデータパケットは同じものとなります。
CAN モジュールが送信したデータを RSCAN モジュールが受信する事も問題なく行えます。
（ルネサス以外のマイコンや、CAN 規格に準拠した通信モジュールとの送受信も問題ないはずです）

4.8. データの受信

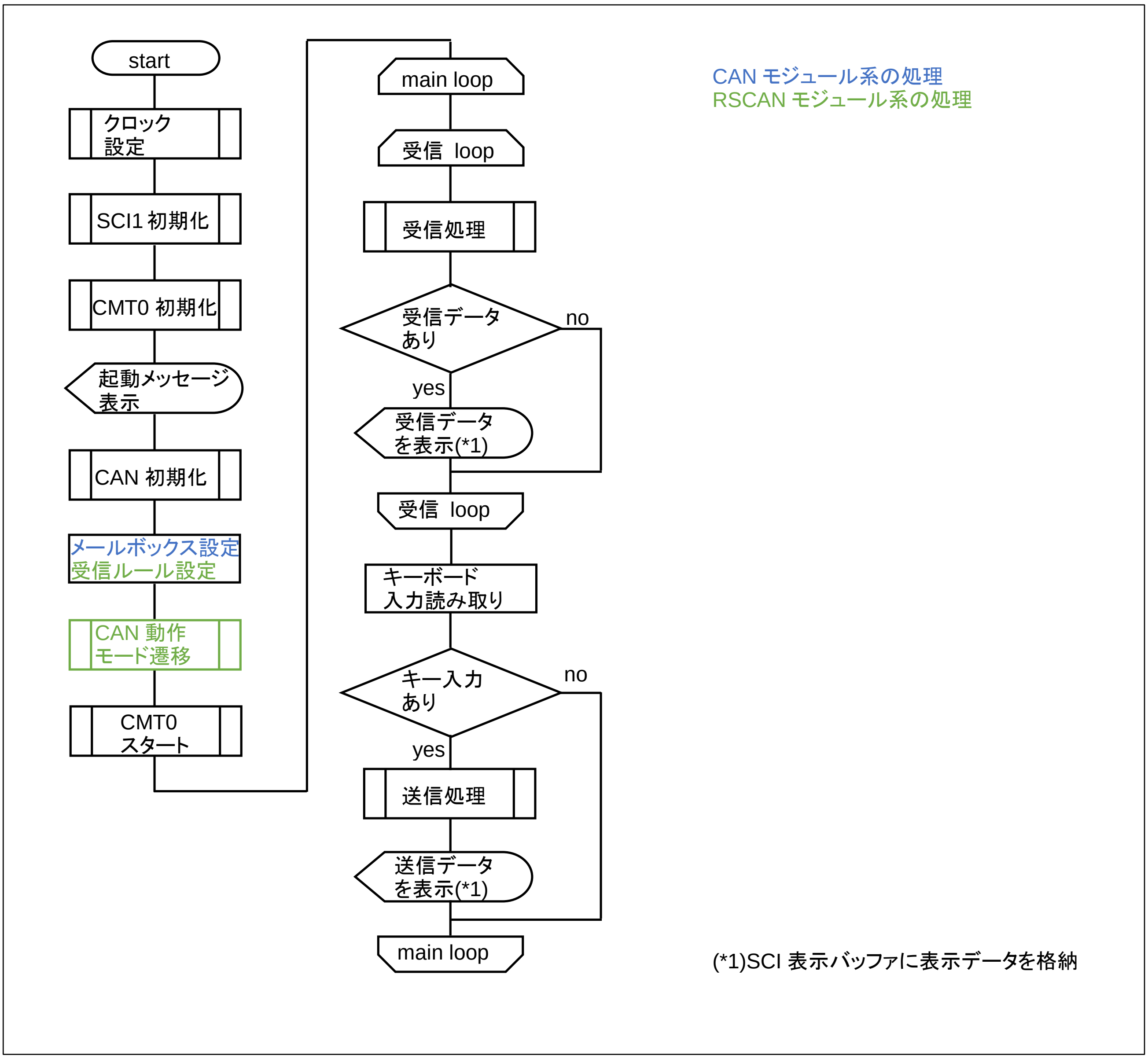
SAMPLE1 でのデータの受信に関しては、
メールボックス（CAN モジュール系）
受信バッファ（RSCAN モジュール系）
までのデータ格納に関しては、（初期化、メールボックス、受信ルール設定が済んでいれば）マイコンのハードウェアが行います。メールボックス、受信バッファにデータが格納されているかは、プログラムで受信関数を呼び出す事で確認を行っています。そのため、常に受信関数を呼び出して確認を行わないと、データの取りこぼしが生じる可能性があります。受信データの確認に CPU リソースを食うため、スムーズな手法とはいえないと考えます。

（なお、次のサンプルプログラム、SAMPLE2 ではデータ受信時に割り込みが入る様に設定しており、受信の処理が割り込みによって処理されます。）

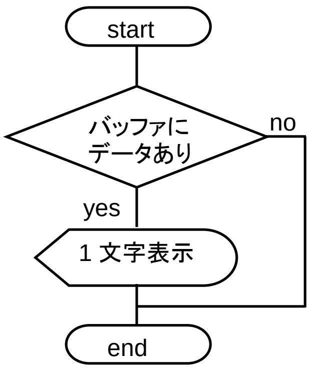
4.9. SAMPLE1 フローチャート

ー処理フローー

メイン関数　main_s1()



CMT0 割り込み関数 Intr_CMT0_CMI0() [100us に 1 回実行される]



画面の文字表示は、115,200bps でも 1 文字表示させるのに、87us 程度掛かるためプログラムのループ内では、バッファに格納（メモリにコピー）とし、一定間隔（100us）毎にバッファの内容を表示処理させています。

5. サンプルプログラムの説明(SAMPLE2)

5.1. プログラム仕様

・データフレームの送信
・データフレームの受信

を行うサンプルプログラムとします。

SAMPLE1 との相違点は、データ送信後の処理と、受信処理を割り込みを使用して行う事とします。

5.2. 関数仕様（SAMPLE2 での相違点）

5.2.1. ユーザ関数

関数の仕様としては、基本的には SAMPLE1 と同じとします。ただし、割り込みの設定等、関数の中身は相違点があります。

can*n*_send　　[CAN モジュール系](n=0~2)

概要：データ送信関数
宣言：

int can0_send(unsigned char mb, unsigned char dlc, unsigned char *data)　[ch0 向け]
int can1_send(unsigned char mb, unsigned char dlc, unsigned char *data)　[ch1 向け]
int can2_send(unsigned char mb, unsigned char dlc, unsigned char *data)　[ch2 向け]

説明：
・データフレームの送信
を行います

引数：
mb: 使用するメールボックスを指定します(0~31)
dlc: 送信バイト数を指定します(1~8)
*data: 送信するデータを指定します

戻り値：
0: 正常終了
-1: 引数チェックエラー
-2: レジスタ値設定タイムアウト
-3: メールボックスが受信に設定されている
-4: メールボックスが現在送信中
-5: 送信フラグがクリアされていない（SAMPLE2 追加仕様）

can_send [RSCAN モジュール系]

概要：データ送信関数
宣言：

int can_send(unsigned short id, unsigned char dlc, unsigned char *data)

説明：

・データフレームの送信

を行います
引数：

~~num: 送信バッファ番号(0~3)~~（SAMPLE2 では FIFO を使用するので送信バッファ番号は指定しない）

id: 送信する ID を指定します

dlc: 送信バイト数を指定します(1~8)

*data: 送信するデータを指定します

戻り値：

0: 正常終了

-1: 引数チェックエラー

-2 : FIFO フル（SAMPLE2 相違点）

can_receive [RSCAN モジュール系]

概要：受信関数
宣言：

int can_receive(unsigned short *id, unsigned char *data, unsigned short *ts)

説明：

・データの受信

を行います
引数：

~~num: 受信ルール番号（＝受信バッファ番号）(0~15)~~

（SAMPLE2 では FIFO を使用するので受信バッファ番号は指定しない）

*id: 受信した ID を格納するポインタ(unsigned short x1)

*data: 受信したデータを格納するポインタ（最大 unsigned char x8）

*ts: データ受信時のタイムスタンプ(unsigned short x1)

戻り値：

0~8: 受信したデータのバイト数（SAMPLE2 相違点）

0x2x: (b5=1)オーバライドフラグが立っている（受信操作前に上書きされたメッセージあり）（SAMPLE2 相違点）

-2: 受信 FIFO にデータなし（SAMPLE2 相違点）

5.3. プログラムで使用している変数・定数（SAMPLE2 での追加分）

5.3.1. 定数定義

#define INTERRUPT_DEBUG

割り込みデバッグ用定数。定義している場合は、CAN の割り込みが入った際 I/O ポートが反転し、オシロ等で割り込みのタイミングのモニタが可能

5.3.2. グローバル変数

・[CAN モジュール系]

unsigned short can0_error_flag;

レジスタ書き換えタイムアウトの際、フラグをセット。

unsigned long can0_mb_send_flag;

送信指示を発行した際にフラグを立て、送信完了時にフラグを落とす。

5.4. 使用する割り込み

5.4.1. RX700,600 系マイコン,CAN モジュール

RX72M, RX71M, RX65, RX64M 系マイコンでは、CAN モジュールの割り込みは、選択型割り込み B に割り当てられおり、メールボックス受信、送信時の割り込みは、RXMn, TXMn となります。これらの割り込みを、（割り当て先は任意に選択できますが）本プログラムでは割り込み番号 130~135 に割り当てる事とします。

選択型割り込み B 割り込み 要因番号	割り込み 要求元	名称	割り当て先 割り込み番号
52	CAN0	RXM0	130
53	CAN0	TXM0	131
56	CAN1	RXM1	132
57	CAN1	TXM1	133
60	CAN2	RXM2	134
61	CAN2	TXM2	135

※RX65N では、CAN2 はありません

RX72T, RX66T 系マイコンでは、CAN モジュールの割り込みは独立した割り込み番号が割り振られています。

割り込み 要求元	名称	割り込み番号
CAN0	RXM0	172
CAN0	TXM0	173

5.4.2. RX200 系マイコン,RSCAN モジュール

RX200 系マイコンでは、FIFO に割り込みが割り当てられているため、SAMPLE2 では、FIFO を使用してデータの送受信を行う事とします。

・RX24U

割り込み 要因番号	割り込み 要求元	名称
60	RSCAN	RXFINT
61	RSCAN	TXINT

・RX231

割り込み 要因番号	割り込み 要求元	名称
53	RSCAN	RXFINT
54	RSCAN	TXINT

5.5. 送受信 FIFO バッファの設定(RSCAN モジュール系)

CAN モジュール系のメールボックスの設定に関しては、SAMPLE1 と SAMPLE2 では同一です。

RSCAN モジュール系では、SAMPLE2 では受信には「受信 FIFO」。送信には「送受信 FIFO」を送信の設定で使用する事とします。

合計 16 個のバッファが使用できますので、本サンプルプログラムでは、
受信 FIFO 8 バッファ
送受信 FIFO(送信に使用) 8 バッファ
を割り当てる事とします。

5.5.1. 受信ルール設定(RSCAN モジュール系)

受信ルール 番号	ID	受信 FIFO 割り当て
0	0x0000	FIFO(0)
1	0x0001	FIFO(0)
2	0x0002	FIFO(0)
3	0x0003	FIFO(0)
4	0x0004	FIFO(0)
5	0x0005	FIFO(0)
6	0x0006	FIFO(0)
7	0x0007	FIFO(0)

SAMPLE2 での受信ルールでは、データを受信した際に、受信 FIFO の 0 番側にデータが格納される様設定します。

5.6. 動作説明（CAN モジュール系）

実際に CAN の通信を行う際、関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can0_init();

CAN の ch0 を初期化する。

(2)メールボックスの設定

```
//      メールボックス番号  ID  送受信区分
can0_mb_set(0, 0x000, CAN_DIRECTION_SEND);//送信
…

can0_mb_set(8, 0x000, CAN_DIRECTION_RECEIVE);//受信
…
```

メールボックス 0~7 を送信設定。メールボックス 8~15 を受信設定する例。メールボックス設定時、ID は、メールボックス番号と紐付けします。

(3)データの送信

```
unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_mb=0;//使用するメールボックス番号
s_dlc=1;//送信バイト数
s_ret = can0_send(s_mb, s_dlc, &s_data[0]);
```

メールボックス 0 から（ID=0x0000）、0x01 を 1 バイト送信する。

(i1)受信割り込み関数[SAMPLE1 との相違点]

```
for(i=0; i<32; i++)
{
    if(CAN0.MCTL[i].BIT.RX.NEWDATA != 1) continue;
    ret = can0_receive(i, &id, &data[0], &ts);
    [受信データの表示]
}
```

全メールボックス(0~31)のうち、新規受信データーがあるメールボックスからの読み出し、画面表示を行う。

(i2)送信割り込み関数[SAMPLE1 との相違点]

送信後のレジスタクリア、送信時に設定したフラグ変数をクリアします。
本プログラムでは、送信後に特定のタスクを実行する様にはなっていませんが、送信完了（CAN バス上でデータパケットを受信し、ACK を返す機器が存在する）時に処理を行う場合はこの部分にコードを記載します。

5.7. 動作説明（RSCAN モジュール系）

実際に CAN の通信を行う際、4.2 記載の関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can_init();

CAN の初期化をする。

(2)受信ルールの設定

// 受信ルール番号 ID
can_receive_rule_set(0, 0x00);
…

can_start();

受信ルール番号 0~7 を受信設定する例。ID は、受信ルール番号と紐付けされる。

(3)データの送信

unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_id=0x000; //ID
s_dlc=1; //送信バイト数
s_ret = **can_send**(s_id, s_dlc, &s_data[0]);

送信 FIFO から、ID=0x000、データ 0x01 を 1 バイト送信する。

(i1)受信割り込み関数[SAMPLE1 との相違点]

```
while(RSCAN.RFSTS0.BIT.RFEMP != 1)
{
    ret = can_receive(&id, &data[0], &ts);
    [受信データの表示]
}
```

割り込みフラグクリアと、受信 FIFO にデータが存在する内は FIFO からの読み出しと表示を行う。

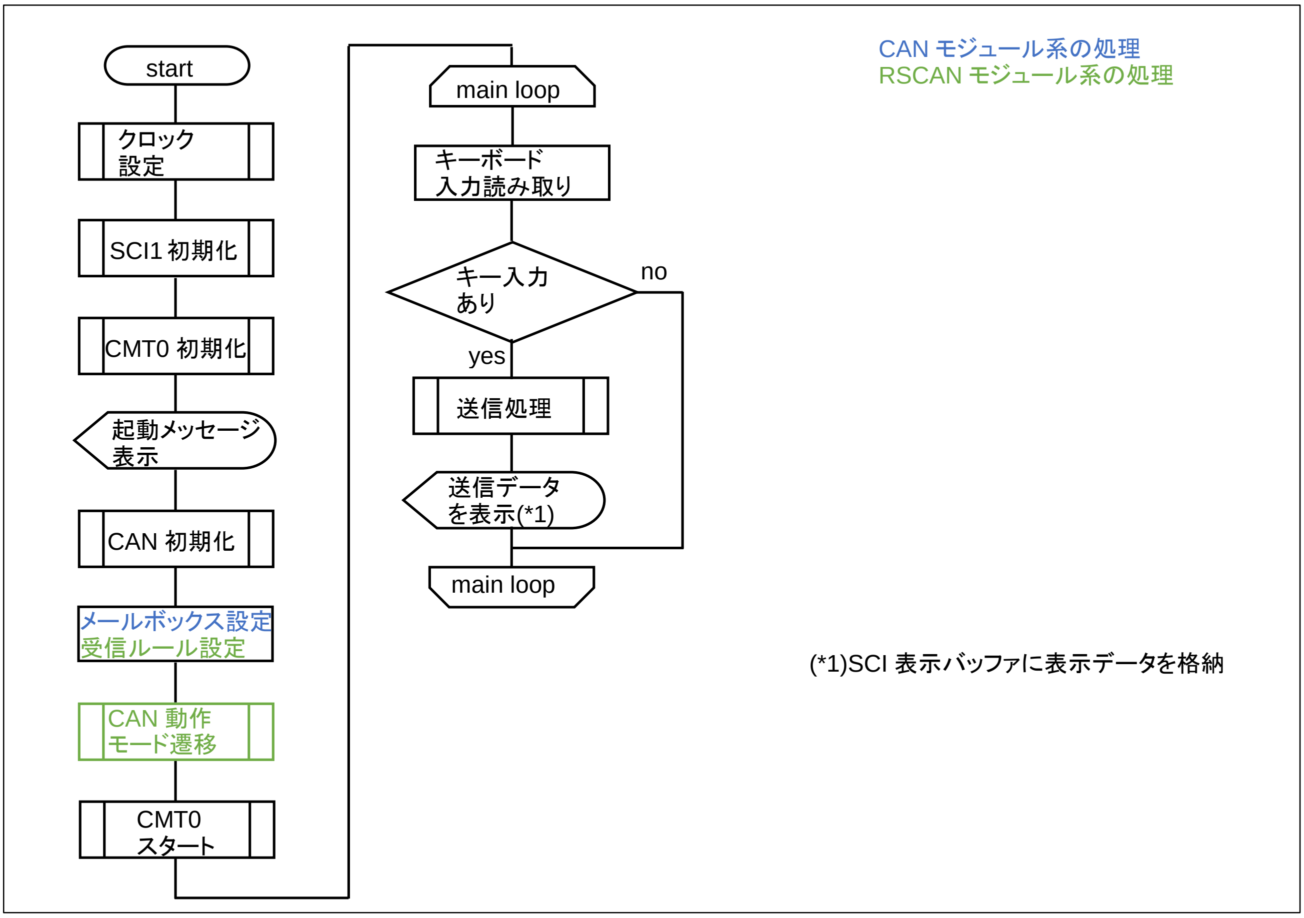
(i2)送信割り込み関数[SAMPLE1 との相違点]

割り込みフラグクリアと、画面表示を行う。

5.8. SAMPLE2 フローチャート

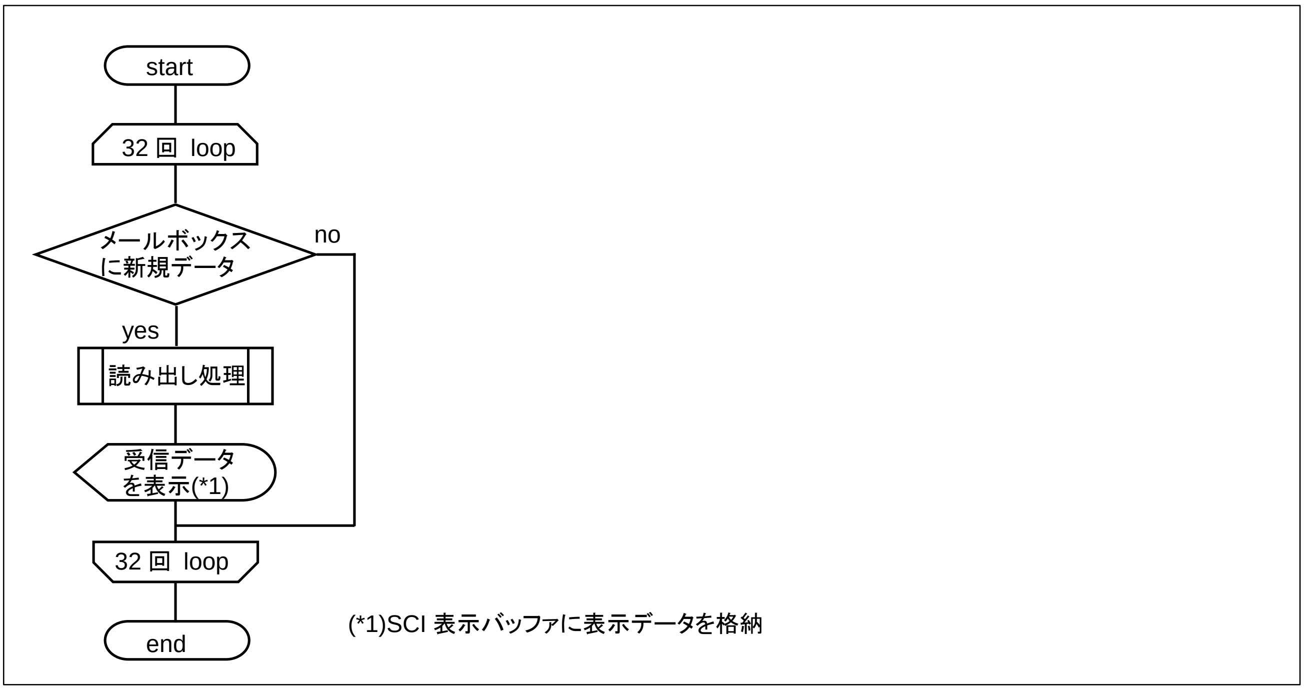
ー処理フローー

メイン関数 main_s2()

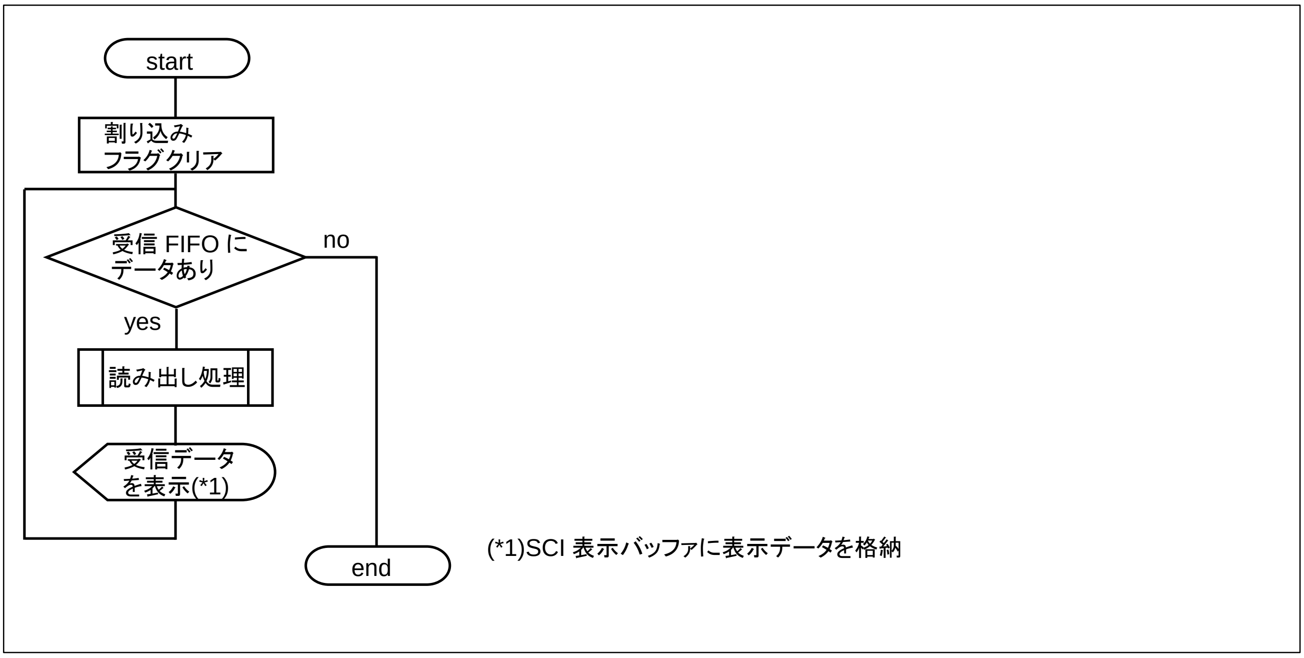


受信割り込み関数（CAN モジュール系）

Intr_CAN0_RXM0() [ch0]　Intr_CAN1_RXM1() [ch1]　Intr_CAN2_RXM2() [ch2]

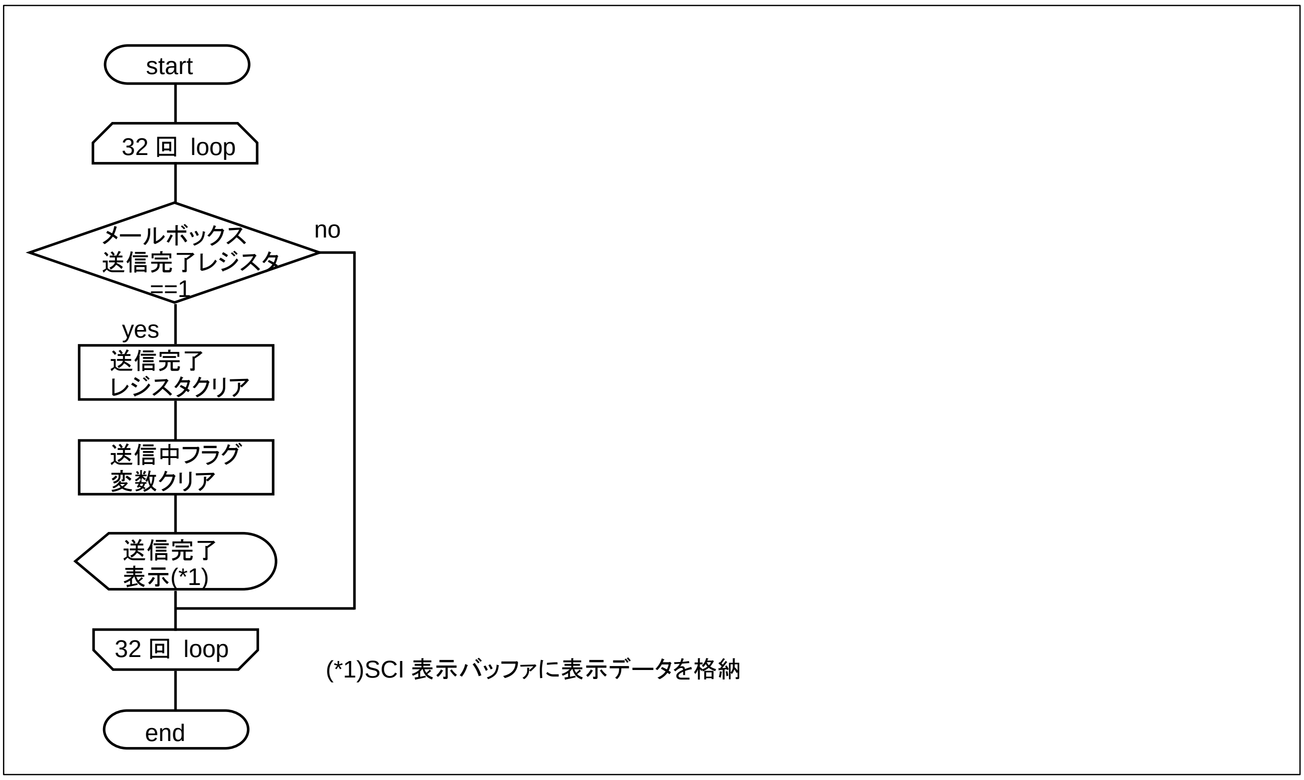


受信割り込み関数（RSCAN モジュール系） Intr_RSCAN_RXFINT()



送信完了割り込み関数（CAN モジュール系）

Intr_CAN0_TXM0() [ch0]　Intr_CAN1_TXM1() [ch1]　Intr_CAN2_TXM2() [ch2]



送信完了割り込み関数（RSCAN モジュール系） Intr_RSCAN_TXINT()

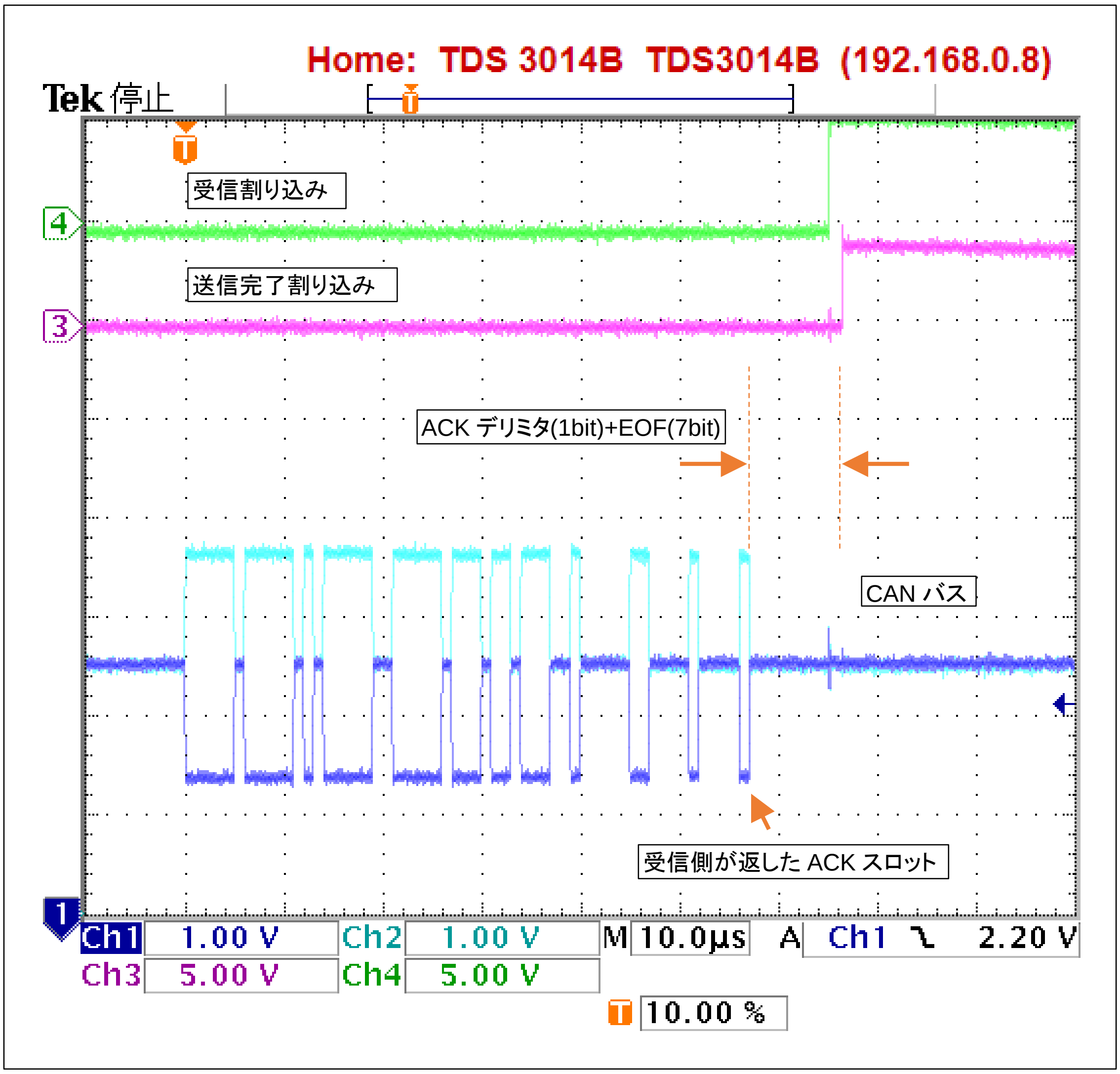
5.9. 割り込みタイミングのデバッグに関して

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 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 ポートを反転させせます。

・データフレーム送信波形例



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 受信	CAN0 送信	CAN1 受信	CAN1 送信	CAN2 受信	CAN2 送信
HSBRX65N176 HSBRX651F176	PE0(J1-49)	PE1(J1-50)	PE2(J1-51)	PE2(J1-52)		
HSBRX65N144(A) HSBRX651F144(A) HSBRX65N100(A) HSBRX651F100(A)	PE0(J1-41)	PE1(J1-42)	PE2(J1-43)	PE3(J1-44)		
HSBRX71M176 HSBRX72M176	PE0(J1-15)	PE1(J1-16)	PE2(J1-17)	PE3(J1-18)	PE4(J1-19)	PE5(J1-20)
HSBRX71M100	PE0(J1-1)	PE1(J3-8)	PE2(J3-7)	PE3(J1-4)	PE4(J1-5)	PE5(J1-6)
HSBRX64MC	PE0(J2-18)	PE1(J2-17)	PE2(J2-16)	PE3(J2-15)	PE4(J2-14)	PE5(J2-13)
HSBRX66T100B	PE0(J1-13)	PE1(J1-12)				
HSBRX72T144 HSBRX66T144	PE0(J4-13)	PE1(J4-14)				

マイコンボード	CAN 受信	CAN 送信
HSBRX24U144 HSBRX24U100 HSBRX24T100B	P91(J1-1)	P92(J1-2)
HSBRX231F100	P53(J1-7)	P52(J1-8)

（カッコ内　拡張 I/O ポート番号）

SAMPLE2（及び SAMPLE3）では、上記ポートを割り込みのモニタに割り当てています。

6. サンプルプログラムの説明(SAMPLE3)

6.1. プログラム仕様

・データフレームの送信
・データフレームの受信
・リモートフレームの送信
・リモートフレームを受信した際応答(データ送信を行う)する

を行うサンプルプログラムとします。

SAMPLE2 との相違点は、リモートフレームに対応している事です。

ーキーボードから入力したキーと送信データの関係ー

・データフレーム送信

キーボードからの入力	ID	送信バイト数(DLC)	送信データ
0	0x0000	1	0x 01
1	0x0001	2	0x 01 23
2	0x0002	3	0x 01 23 45
3	0x0003	4	0x 01 23 45 67
4	0x0004	5	0x 01 23 45 67 89
5	0x0005	6	0x 01 23 45 67 89 AB
6	0x0006	7	0x 01 23 45 67 89 AB CD
7	0x0007	8	0x 01 23 45 67 89 AB CD EF

・リモートフレーム送信

キーボードからの入力	ID	送信要求バイト数(DLC)
q	0x0000	1
w	0x0001	2
e	0x0002	3
r	0x0003	4
t	0x0004	5
y	0x0005	6
u	0x0006	7
i	0x0007	8

・リモートフレーム応答

0xA1 A2 A3 A4 A5 A6 A7 A8

を、要求元の送信要求バイト数に応じて返答

(DLC=3 のときは、0xA1 A2 A3 を返す)

※本サンプルプログラムは、ID=0x000 ~ 0x0007 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)しますので、本サンプルプログラムが動作するボードを 3 台(以上)同一バスに接続すると、2 台(以上)が同時に応答します CAN バス上では、1 つのリモートフレームに続き 2 つ(以上)のデータが流がれますので、多少動作が見難くなるかと思います

※SAMPLE3 を 3 台以上同時に接続する場合は、ボード毎に応答する ID を変更する事を推奨致します

6.2. 関数仕様（SAMPLE3 での相違点）

6.2.1. ユーザ関数

can*n*_mb_set　[CAN モジュール系](n=0~2)

概要：メールボックス設定関数

宣言：

int can0_mb_set(unsigned char mb, unsigned short id, unsigned char rtr, unsigned char direction)

[ch0 向け]

int can1_mb_set(unsigned char mb, unsigned short id, unsigned char rtr, unsigned char direction)

[ch1 向け]

int can2_mb_set(unsigned char mb, unsigned short id, unsigned char rtr, unsigned char direction)

[ch2 向け]

説明：

・メールボックスの設定

を行います

引数：

mb: メールボックス番号(0~31)

id: CAN-ID を指定

rtr: データフレーム、リモートフレーム区分を指定（受信設定のときのみ有効）

CAN_RTR_DATA(0)　データフレームでは 0 指定

CAN_RTR_REMOTE(1)　リモートフレームでは 1 指定

direction: 送受信区分を設定

CAN_DIRECTION_SEND(0)　送信時は 0 指定

CAN_DIRECTION_RECEIVE(1)　受信時は 1 指定

戻り値：

0: 正常終了

-1: 引数エラー

-2: 送信、受信アボート処理失敗

can*n*_send　[CAN モジュール系](n=0~2)

概要：データ送信関数

宣言：

int can0_send(unsigned char mb, unsigned char rtr, unsigned char dlc, unsigned char *data)

[ch0 向け]

int can1_send(unsigned char mb, unsigned char rtr, unsigned char dlc, unsigned char *data)

[ch1 向け]

int can2_send(unsigned char mb, unsigned char rtr, unsigned char dlc, unsigned char *data)

[ch2 向け]

説明:
　・データフレーム／リモートフレームの送信
を行います

引数:
　　mb: 使用するメールボックスを指定します(0~31)
　　rtr: データフレーム、リモートフレーム区分を指定
　　　CAN_RTR_DATA(0)　　　データフレームでは 0 指定
　　　CAN_RTR_REMOTE(1)　 リモートフレームでは 1 指定
　　dlc: 送信バイト数、要求バイト数を指定します(1~8)
　　*data: 送信するデータを指定します（リモートフレームでは未使用）
戻り値:
　　0: 正常終了
　　-1: 引数チェックエラー
　　-2: レジスタ値設定タイムアウト
　　-3: メールボックスが受信に設定されている
　　-4: メールボックスが現在送信中
　　-5: 送信フラグがクリアされていない

can_send　　[RSCAN モジュール系]

概要:データ送信関数
宣言:
　　int can_send(unsigned short id, unsigned char rtr, unsigned char dlc, unsigned char *data)
説明:
　・データフレームの送信
を行います
引数:
　　id: 送信する ID を指定します
　　rtr: データフレーム、リモートフレーム区分を指定
　　　CAN_RTR_DATA(0)　　　データフレームでは 0 指定
　　　CAN_RTR_REMOTE(1)　 リモートフレームでは 1 指定
　　dlc: 送信バイト数、要求バイト数を指定します(1~8)
　　*data: 送信するデータを指定します（リモートフレームでは未使用）
戻り値:
　　0: 正常終了
　　-1: 引数チェックエラー
　　-2 : FIFO フル

can*n*_receive　　[CAN モジュール系](n=0~2)

概要：受信関数

宣言：

int can0_receive(unsigned char mb, unsigned short *id, unsigned char *rtr, unsigned char *data, unsigned short *ts)　[ch0 向け]

int can1_receive(unsigned char mb, unsigned short *id, unsigned char *rtr, unsigned char *data, unsigned short *ts)　[ch1 向け]

int can2_receive(unsigned char mb, unsigned short *id, unsigned char *rtr, unsigned char *data, unsigned short *ts)　[ch2 向け]

説明：

・データの受信

を行います

引数：

mb: メールボックス(0~31)

*id: 受信した ID を格納するポインタ(unsigned short x1)

*rtr: データフレーム、リモートフレーム区分(unsigned char x1)

*data: 受信したデータを格納するポインタ（最大 unsigned char x8）

*ts: データ受信時のタイムスタンプ(unsigned short x1)

戻り値：

0: 受信データなし

1~8: 受信したデータのバイト数

0x1x: (b4=1)受信メールボックス更新中

0x2x: (b5=1)オーバライドフラグが立っている（受信操作前に上書きされたメッセージあり）

-1: 引数チェックエラー

-2: レジスタ設定タイムアウト

-3: メールボックスが受信用に設定されていない

can_receive　　[RSCAN モジュール系]

概要：受信関数

宣言：

int can_receive(unsigned short *id, unsigned char *rtr, unsigned char *data, unsigned short *ts)

説明：

・データの受信

を行います

引数：

*id:　受信した ID を格納するポインタ(unsigned short x1)

*rtr:　データフレーム、リモートフレーム区分(unsigned char x1)

*data:　受信したデータを格納するポインタ（最大 unsigned char x8）

*ts:　データ受信時のタイムスタンプ(unsigned short x1)

戻り値：

0~8:　受信したデータのバイト数

0x2x: (b5=1)オーバライドフラグが立っている（受信操作前に上書きされたメッセージあり）

-2:　受信 FIFO にデータなし

6.3. プログラムで使用している変数・定数（SAMPLE3 での追加分）

6.3.1. 定数定義

```
#define CAN_DATA_FRAME  0
#define CAN_REMOTE_FRAME  1
```

データフレームとリモートフレーム識別用定数。

6.4. メールボックス設定（CAN モジュール系）

6.4.1. メールボックス設定(CAN モジュール系）

メールボックス番号	ID	RTR	送受信区分
0	0x0000	0	送信
1	0x0001	0	送信
2	0x0002	0	送信
3	0x0003	0	送信
4	0x0004	0	送信
5	0x0005	0	送信
6	0x0006	0	送信
7	0x0007	0	送信
8	0x0000	0	受信
9	0x0001	0	受信
10	0x0002	0	受信
11	0x0003	0	受信
12	0x0004	0	受信
13	0x0005	0	受信
14	0x0006	0	受信
15	0x0007	0	受信
16	0x0000	1	受信
17	0x0001	1	受信
18	0x0002	1	受信
19	0x0003	1	受信
20	0x0004	1	受信
21	0x0005	1	受信
22	0x0006	1	受信
23	0x0007	1	受信

16~23 をリモートフレーム受信用に割り当て。

※0~7 の送信用メールボックスは、設定の段階では RTR（データ／リモートフレーム値）を 0 に設定していますが、データ／リモートフレームに応じて送信時に RTR 値を書き換えます

6.5. 動作説明（CAN モジュール系）

実際に CAN の通信を行う際、関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can0_init();

CAN の ch0 を初期化する。

(2)メールボックスの設定

// メールボックス番号 ID RTR 送受信区分
can0_mb_set(0, 0x000, CAN_DATA_FRAME, CAN_DIRECTION_SEND);//送信
…
can0_mb_set(8, 0x000, CAN_DATA_FRAME, CAN_DIRECTION_RECEIVE);//データフレーム受信
…
can0_mb_set(16, 0x000, CAN_REMOTE_FRAME, CAN_DIRECTION_RECEIVE);//リモートフレーム受信
…

0~7 を送信用、8~15 をデータフレーム受信用、16~23 をリモートフレーム受信用に設定。

(3)データの送信

unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_mb=0;//使用するメールボックス番号
s_rtr=CAN_DATA_FRAME;//データフレーム
s_dlc=1;//送信バイト数
s_ret = **can0_send**(s_mb, s_rtr, s_dlc, &s_data[0]);

メールボックス 0 から（ID=0x0000）、0x01 を 1 バイト送信（データフレーム）する。

(i1)受信割り込み関数

for(i=0; i<32; i++)
{
    if(CAN0.MCTL[i].BIT.RX.NEWDATA != 1) continue;
    ret = **can0_receive**(i, &id, &rtr, &data[0], &ts);
    [データフレームの場合：受信データの表示]
    [リモートフレームの場合：データ表示，データ送信]
    ret2 = **can0_send**(mb, CAN_DATA_FRAME, dlc, &remote_frame_response_data[0]);

}

全メールボックス(0~31)のうち、新規受信データがあるメールボックスを確認し、データフレームであれば、画面表示を行う。リモートフレームであれば、画面表示を行った後、データの送信を行う。データ送信に関しては、通常のデータフレーム送信と同じ関数(can***n***_send)で行う。但し、送信バイト数はリモートフレーム要求元から送信されてきた値(DLC)を使用する。

(i2)送信割り込み関数

送信後のレジスタクリア、送信時に設定したフラグ変数をクリアします。

6.6. 動作説明(RSCAN モジュール系)

実際に CAN の通信を行う際、関数をどのような流れで呼び出すかを以下で説明します。

(1)初期化を行う

can_init();

CAN の初期化をする。

(2)受信ルールの設定

// 受信ルール番号 ID
can_receive_rule_set(0, 0x00);
…
can_start();

受信ルール番号 0~7 を受信設定する例。ID は、受信ルール番号と紐付けされる。

(3)データの送信

unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ
s_id=0x000; //ID
s_rtr=CAN_DATA_FRAME;
s_dlc=1; //送信バイト数
s_ret = **can_send**(s_id, s_rtr, s_dlc, &s_data[0]);

送信 FIFO から ID=0x000、データ 0x01 を 1 バイト送信する。

(i1)受信割り込み関数

while(RSCAN.RFSTS0.BIT.RFEMP != 1)
{
　ret = **can_receive**(&id, &rtr, &data[0], &ts);
　[データフレームの場合：受信データの表示]
　[リモートフレームの場合：データ表示，データ送信]
　ret2 = **can_send**(id, CAN_DATA_FRAME, dlc, &remote_frame_response_data[0]);
}

割り込みフラグクリアと、受信 FIFO にデータが存在する内は FIFO からの読み出しと表示を行う。リモートフレームの際はデータの送信を行う。データ送信に関しては、通常のデータフレーム送信と同じ関数(can_send)で行う。但し、送信バイト数はリモートフレーム要求元から送信されてきた値(DLC)を使用する。

(i2)送信割り込み関数

割り込みフラグクリアと、画面表示を行う。

6.7. 送信データ

データ送信関数にリモートフレームを送信するよう指定して呼び出した時、CAN のポートからは、以下のようなデータ列が出力され、CAN バス（CANH, CANL のライン）に流れます。

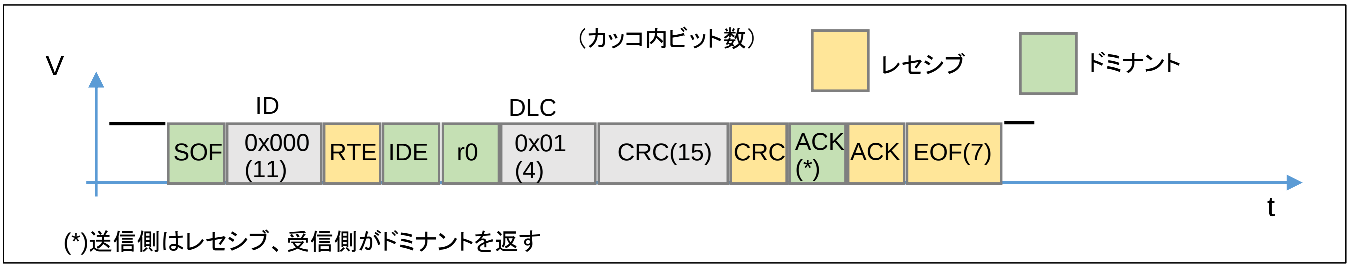


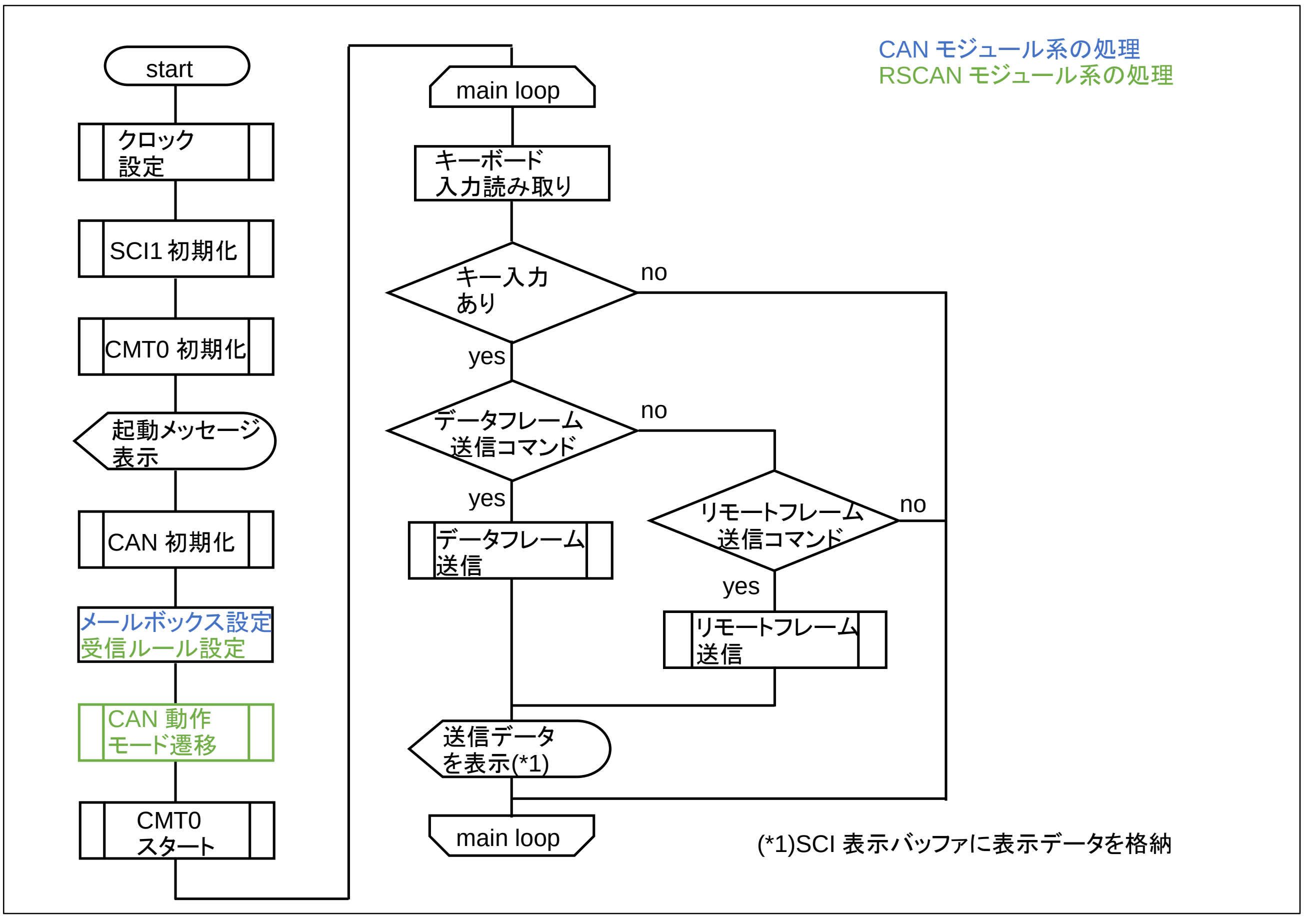
図 6-1 CAN リモートフレームのデータパケット

RTE のデータがレセシブ(1)となり、データパケットでデータとなっていた箇所が詰められた形となります。DLC は、送信側が受信側に送って欲しいデータバイト数を埋め込んだ形で送信します。

6.8. SAMPLE3 フローチャート

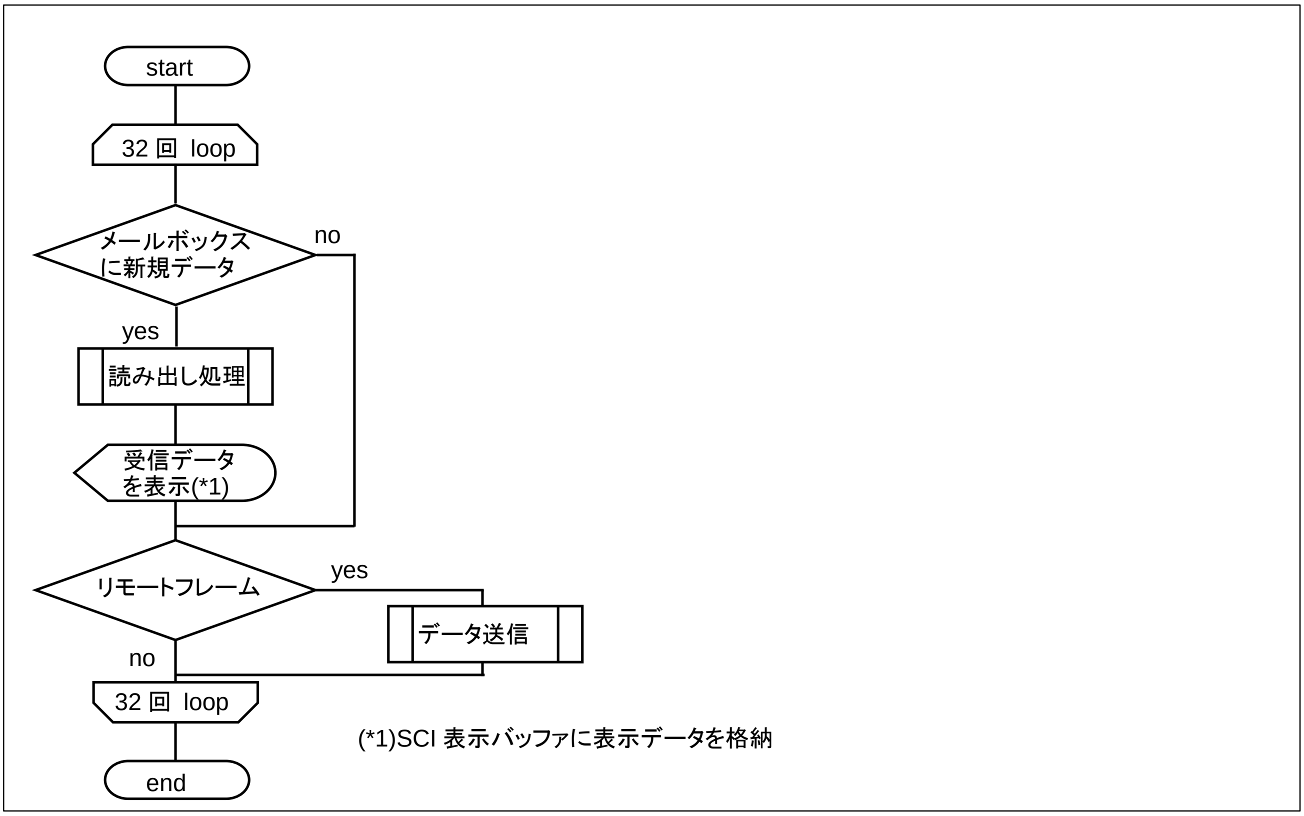
ー処理フローー

メイン関数 main_s3()

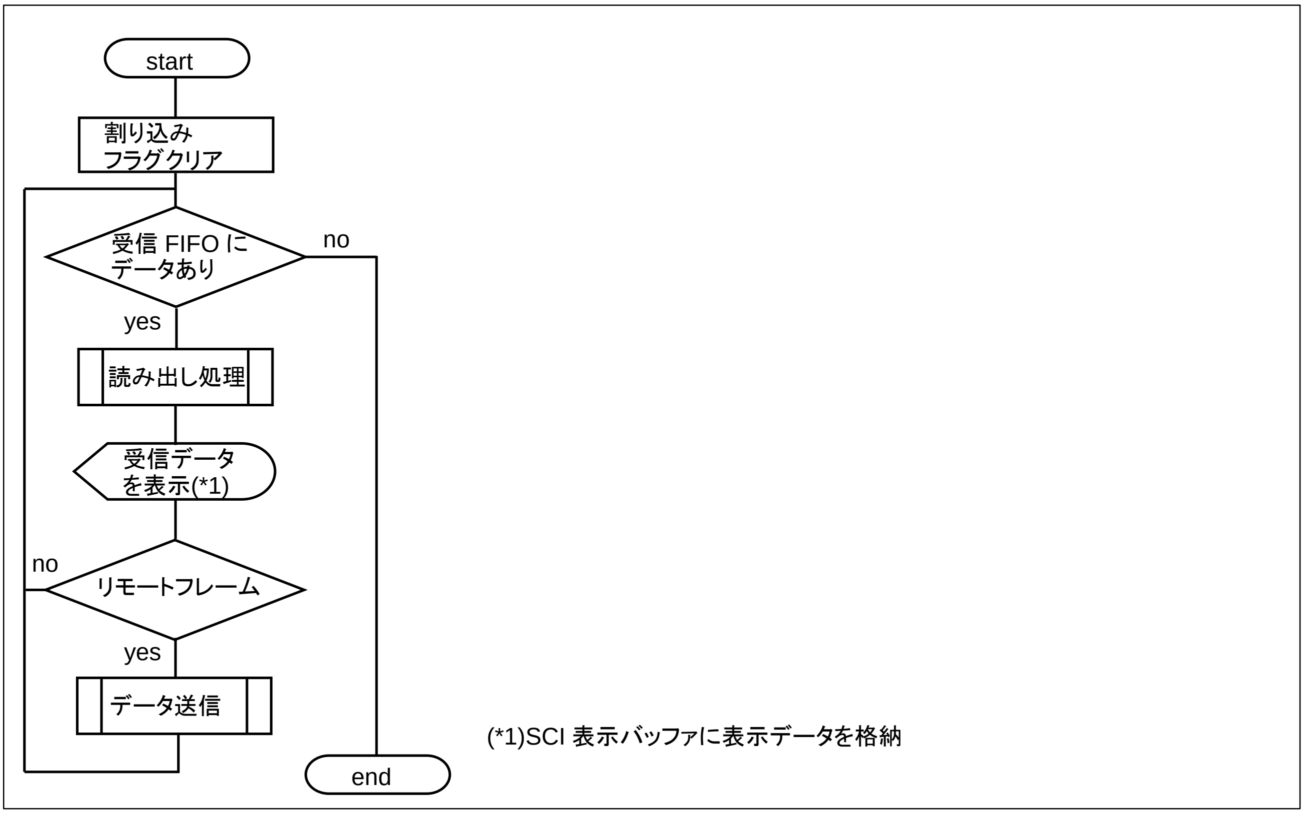


受信割り込み関数（CAN モジュール系）

Intr_CAN0_RXM0() [ch0]　Intr_CAN1_RXM1() [ch1]　Intr_CAN2_RXM2() [ch2]



受信割り込み関数（RSCAN モジュール系） Intr_RSCAN_RXFINT()



7. その他

7.1. 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
HSBRX65xxxx	**P32** PD1	**P33** PD2	P14 P44	P15 P55		
HSBRX71M176 HSBRX72M176	P32 **PD1**	P33 **PD2**	P14 P44	P15 P55	**P66**	**P67**
HSBRX71M100	**P32** PD1	**P33** PD2	P14 P44	P15 P55		
HSBRX64MC	**P32** PD1	**P33** PD2	**P14** P44	**P15** P55	**P66**	**P67**
HSBRX66T100B	**P23** PA0 PB5 PD7	**P22** PA1 PB6 PE0				
HSBRX72T144 HSBRX66T144	**PA0** P23 PA6 PB5 PC5 PD7 PF2	**PA1** P22 PA7 PB6 PC6 PE0 PF3				

・RSCAN モジュール系(RX200)

	CTXD0	CRXD0
HSBRX24Uxxx HSBRX24T100B	**PA0** PF2	**PA1** PF3
HSBRX231F100 SmartRX!!!	**P14** P54	**P15** P55

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 系で CAN の ch1 を使用する場合、
SOURCE¥RX_CANST¥source¥can1¥ch1_s**n**.h（**n**=1~3, SAMPLE1~3 に対応）

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

#if (MCU_TYPE == MCU_TYPE_RX65_2M) ||   (MCU_TYPE == MCU_TYPE_RX65_1M)

//#define CAN1_P54_P55
//#define CAN1_P14_P15

#elif (MCU_TYPE == MCU_TYPE_RX71M)

//#define CAN1_P54_P55
//#define CAN1_P14_P15

#elif (MCU_TYPE == MCU_TYPE_RX64M)

#define CAN1_P14_P15
//#define CAN1_P54_P55

#endif
```

使用するポートに合わせて、CAN1_Pxx_Pxx の定義を有効にしてください。

CAN モジュール系では、
SOURCE¥RX_CANST¥source¥can0¥ch0_s**n**.h
SOURCE¥RX_CANST¥source¥can1¥ch1_s**n**.h
SOURCE¥RX_CANST¥source¥can2¥ch2_s**n**.h

RSCAN モジュール系では、
SOURCE¥RX_CANST¥source¥rscan¥can_s**n**.h

（**n**=1~3, SAMPLE1~3 に対応）

内で、どのポートを CAN で使用するか定義していますので、ポートの変更やチャネルの追加は、このファイルを変更してください。

※ポート指定のヘッダファイル内では、例えば
(CTX, CRX) = (P54, P55)
(CTX, CRX) = (P14, P15)
の組（ポート名、端子番号の近い組）で予め定義していますが、
(CTX, CRX) = (P54, P15)
の様な組み合わせでも使用可能です。そのような使い方を行いたい際は、ヘッダファイル内に定義を追加してください。

7.2. 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。

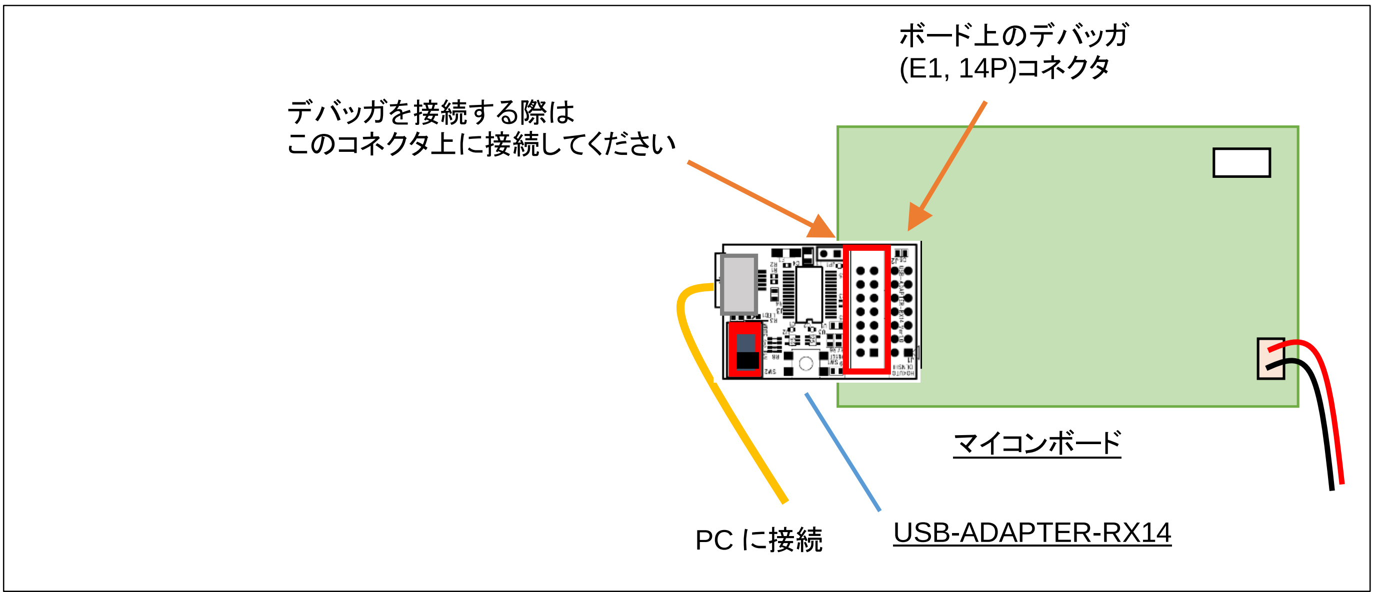
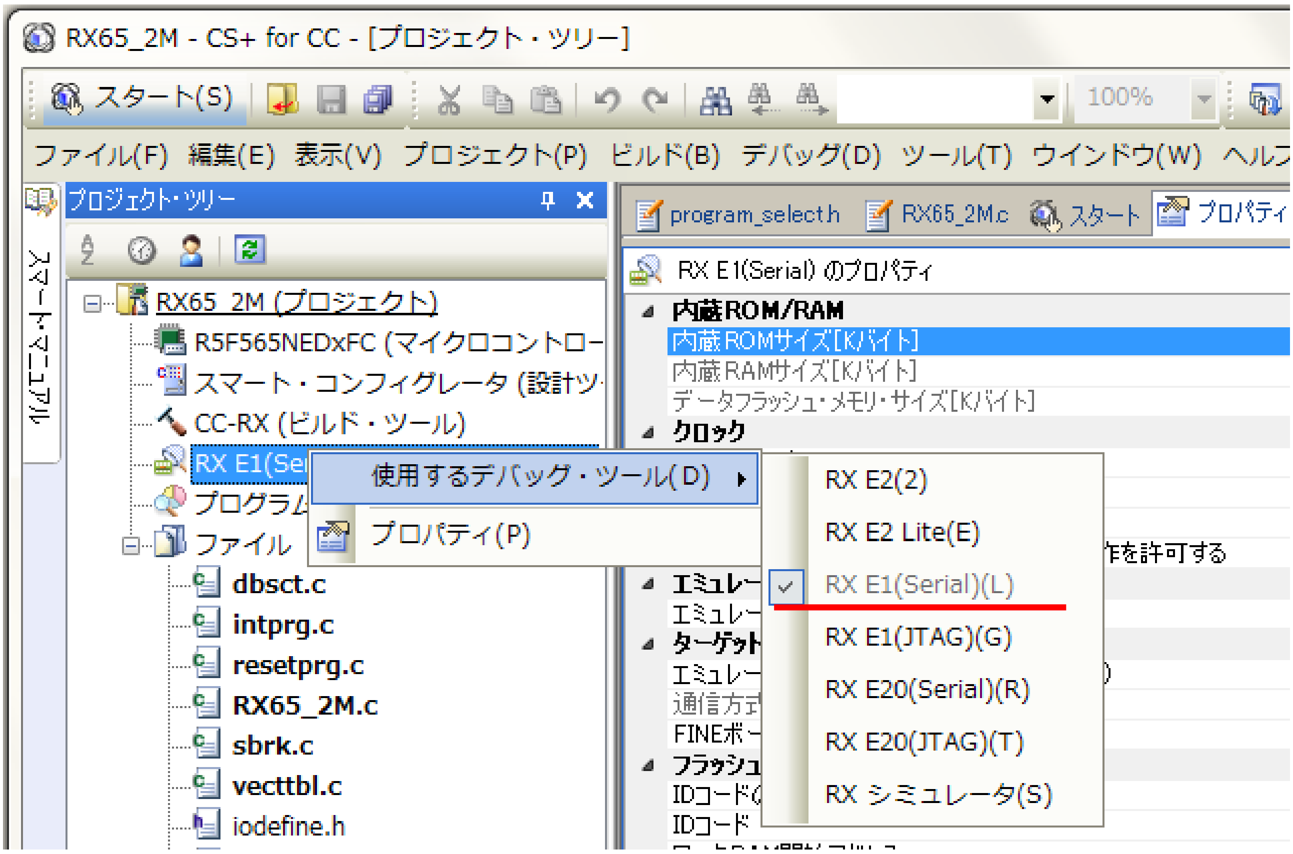


図 7-1 デバッガ接続

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。

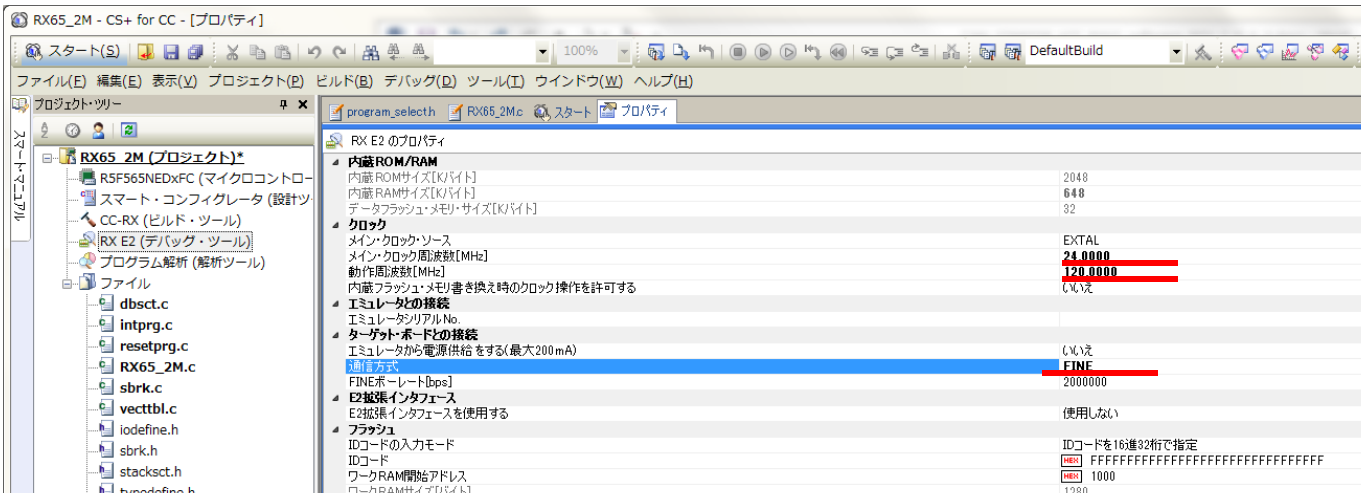
・デバッガの選択



使用するデバッガに応じて、選択してください。

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。)

・デバッグツール設定



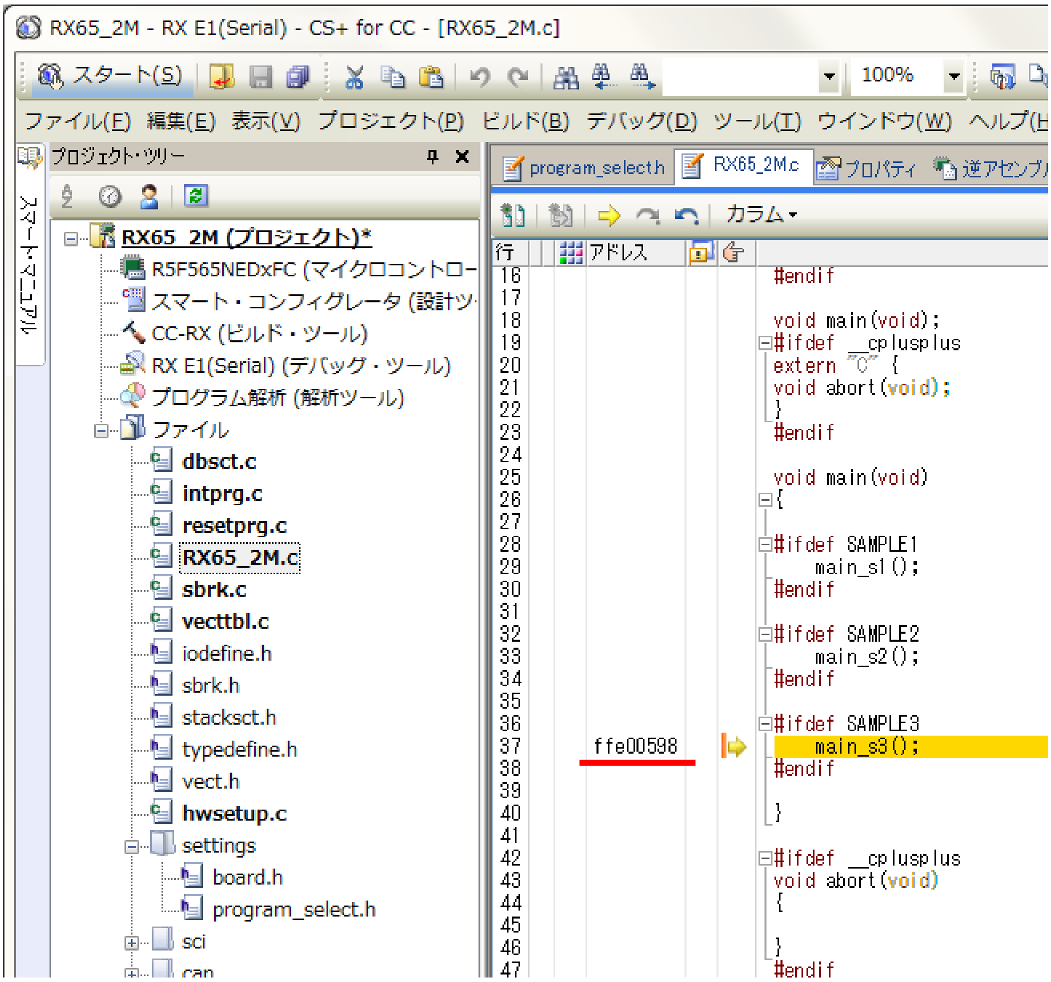
メイン・クロック・ソースは **EXTAL** を選択し、メイン・クロック周波数は、ボード搭載の水晶振動子の値を入力してください。

	メイン・クロック周波数 [MHz]	動作周波数 [MHz]
HSBRX65xxxx	24	120
HSBRX71Mxxx HSBRX72M176	24	240
HSBRX64MC	24	120
HSBRX24Uxxx HSBRX24T100B	10	80
HSBRX231F100 SmartRX!!!	8	54
HSBRX66T100B	20	160
HSBRX66T144	24	160
HSBRX72T144	24	200

通信方式が選べる場合（RX600/700 系マイコンと、E2 系デバッガの組み合わせ）では、**FINE** を選択してください。

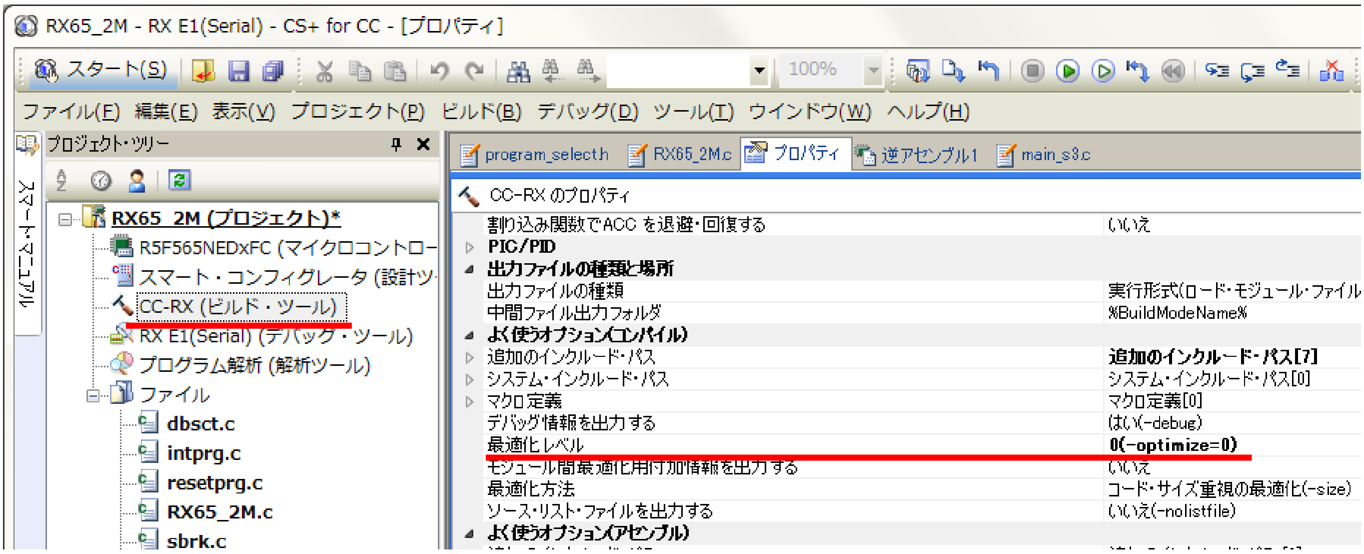
デバッガと、USB-ADAPTER-RX14 を経由した PC との通信は同時に使用可能です。

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。

8. 付録

8.1. キットに関するご意見・ご要望に関して

本製品は、サンプルプログラムを通して、実際に CAN バスを駆動して、CAN の動作を学べるキットとして開発を行っています。

本キットで CAN の基本的な動作を学んだ次のステップとして

「この様なサンプルプログラムを付属させて欲しい」

といった要望がございましたら、

support@hokutodenshi.co.jp

まで、

件名:「CAN スタータキット RXV2 に関する要望」

といった形で、ご意見・ご要望をメールにて送付頂けないでしょうか。

頂いたご要望は、本キットのバージョンアップや製品開発の参考にさせて頂きます。

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	2018.6.29	—	初版発行
REV.1.1.0.0	2019.6.21	P5-8 P11,12,14 P27 P35 P40-41 P44 P54 P55 P56 P59	RX66T/72T を追加 RX66T/72T を追加 誤記訂正 RX66T/72T を追加 記載を修正(関数名変更) RX66T/72T を追加 記載を修正(関数名変更) RX66T/72T を追加 説明を追加 RX66T/72T を追加
REV.1.2.0.0	2020.3.3	P5,7,8,12, 14,35,43, 55,59	RX72M を追加

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:http://www.hokutodenshi.co.jp

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

CAN スタータキット RXV2
CAN スタータキット SmartRX
ソフトウェア編 マニュアル

株式会社北斗電子