CAN スタータキット RX/RA
CAN スタータキット SmartRX
CAN モジュール編
ソフトウェアマニュアル

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.1.9.0.0

ー目　次ー

注意事項 **1**

安全上のご注意 **2**

概要 **4**

1.　ソースファイル構成 **5**

2.　サンプルプログラムの説明(SAMPLE1) **6**

2.1.　プログラム仕様 6
2.2.　メールボックスの設定 7
2.3.　動作説明 7
2.4.　データの受信 10
2.5.　SAMPLE1 フローチャート 11

3.　サンプルプログラムの説明(SAMPLE2) **12**

3.1.　プログラム仕様 12
3.2.　使用する割り込み 12
3.2.1.　RX700,600 系マイコン CAN モジュール 12
3.2.2.　RA 系マイコン CAN モジュール 13
3.3.　動作説明（CAN モジュール系） 14
3.4.　SAMPLE2 フローチャート 16
3.5.　割り込みタイミングとメッセージ表示に関して 18

4.　サンプルプログラムの説明(SAMPLE3) **19**

4.1.　プログラム仕様 19
4.2.　メールボックス設定 20
4.3.　動作説明 21
4.4.　SAMPLE3 フローチャート 23

5.　サンプルプログラムで使用している関数の説明 **25**

5.1.　関数仕様 25
5.2.　プログラムで使用している変数・定数 29
5.2.1.　グローバル変数 29
5.2.2.　定数定義 29
5.3.　CAN マクロのメールボックスに関して 31
取扱説明書改定記録 32
お問合せ窓口 32

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。

従来のマニュアルでは、複数のモジュールの動作を併記していましたが、本バージョンからモジュール毎に分割を行う事と致しました。本書は、「CAN モジュール編」のマニュアルです。CAN モジュール搭載マイコンの場合、本書を参照してください。

1. ソースファイル構成

・CAN モジュール向け

フォルダ	ファイル	説明
source¥CAN_module¥can		CAN モジュール向け共通フォルダ
	can.h	各種定義ファイル
source¥CAN_module¥can0		CAN モジュール ch0
	can_ch0.h	ch0 関数用ヘッダ
	can_ch0.c	ch0 関数
	can_ch0_pin_def.h	ch0 端子設定
	can_ch0_s1.c	SAMPLE1 向け割り込み関数定義（中身は空）
	can_ch0_s2.c	SAMPLE2 向け割り込み関数定義
	can_ch0_s3.c	SAMPLE3 向け割り込み関数定義
source¥CAN_module¥can1		CAN モジュール ch1
	can_ch0.h	ch1 関数用ヘッダ
	can_ch0.c	ch1 関数
	can_ch0_pin_def.h	ch1 端子設定
	can_ch0_s1.c	SAMPLE1 向け割り込み関数定義（中身は空）
	can_ch0_s2.c	SAMPLE2 向け割り込み関数定義
s	can_ch0_s3.c	SAMPLE3 向け割り込み関数定義
source¥CAN_module¥can2		CAN モジュール ch2
	can_ch0.h	ch2 関数用ヘッダ
	can_ch0.c	ch2 関数
	can_ch0_pin_def.h	ch2 端子設定
	can_ch0_s1.c	SAMPLE1 向け割り込み関数定義（中身は空）
	can_ch0_s2.c	SAMPLE2 向け割り込み関数定義
	can_ch0_s3.c	SAMPLE3 向け割り込み関数定義
source¥CAN_module¥main		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

※RA 向けには can2 フォルダが存在しません

2. サンプルプログラムの説明(SAMPLE1)

2.1. 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。

・CAN ID は拡張フォーマット(29bit)とする（標準フォーマットのデータも受信する）(*1)
・送信に使用する ID は 0x0000000~0x0000003 までの 4 種
・受信する ID は、0x0000000~0x0000003 までの 4 種（それ以外の ID のデータは受信しない）
・送信データは"s"コマンドで拡張フォーマットと標準フォーマットの切り替えが可能
・複数の CAN ch を持っているボードは、全ポート受信を行い、送信は"c"コマンドで ch の変更を行う
・データフレームのみ受信（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 0~3 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
・プッシュスイッチが付いているボードでは、プッシュスイッチを押すと 1 バイト送信する（キーボードの 0 と等価）
・LED が付いているボードはデータを受信する度に LED の点灯・消灯が切り替わる

－キーボードから入力したキーと送信データの関係－

キーボードからの入力	ID	送信バイト数	送信データ
0	0x0000000	1	0x 01
1	0x0000001	2	0x 01 23
2	0x0000002	4	0x 01 23 45 67
3	0x0000003	8	0x 01 23 45 67 89 AB CD EF

(*1)can_operation.h の定義で、標準フォーマットのみ取り扱う様に変更可能

2.2. メールボックスの設定

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

CAN モジュールでは、32 個のメールボックスが使用できます。本サンプルプログラムでは、0~7 を送信用、8~15 を受信用に設定しています。

※can_operation.h で標準フォーマットを選択した場合は、メールボックスの 4~7, 12~15 は未使用となります。

2.3. 動作説明

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。

※CAN の ch1, ch2 を使用する場合は、この部分で
can1_init();
can2_init();
を実行する。

(2)メールボックスの設定

メールボックス 0~8 を送信用のメールボックスに設定します。

//引数:メールボックス番号　フォーマット区分　ID
can0_mb_set_send(0, CAN_ID_FORMAT_SID, mb_id[0]); //送信用，標準フォーマット, mb_id[0]=0x0000000
can0_mb_set_send(1, CAN_ID_FORMAT_SID, mb_id[1]); //送信用，標準フォーマット, mb_id[1]=0x0000001
…
can0_mb_set_send(4, CAN_ID_FORMAT_EID, mb_id[4]); //送信用，拡張フォーマット, mb_id[4]=0x0000000
…

メールボックス 8~15 を受信用のメールボックスに設定します。

can0_mb_set_receive(8, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, mb_id[8]); //受信用，標準フォーマット, mb_id[0]=0x0000000
…
can0_mb_set_receive(12, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, mb_id[12]); //受信用，拡張フォーマット, mb_id[12]=0x0000000
…
can0_mb_set_receive(15, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, mb_id[15]); //受信用，拡張フォーマット, mb_id[15]=0x0000003

メールボックス番号：0~31 のメールボックス番号
フォーマット区分:標準フォーマット(CAN_ID_FORMAT_SID)，拡張フォーマット(CAN_ID_FORMAT_EID)を指定
データフレーム／リモートフレーム区分：データフレーム(CAN_DATA_FRAME)，リモートフレーム(CAN_REMOTE_FRAME)を指定
ID：ID 値を指定

can*n*_mb_set_send()は、メールボックスを送信用に設定する関数で、can*n*_mb_set_receive()は、受信用に設定する関数です。引数に ID を与え、この時点で、メールボックス番号と ID 値の紐付けを行います。

上記では、メールボックス 0~7 を送信設定（0~3:標準フォーマット, 4~7:拡張フォーマット）。メールボックス 8~15 を受信設定（8~11:標準フォーマット, 12~15:拡張フォーマット）としています。

(3)データの受信

```
for(i=8; i<=15; i++)  //メールボックス[8]-[15]が受信設定
{
   //引数:メールボックス番号  フォーマット区分  データフレーム／リモートフレーム区分  ID  データ  タイムスタンプ
   r_ret = can0_receive(i, &r_ide, &r_rtr, &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。

メールボックス番号: 受信したいメールボックス番号
フォーマット区分: 受信した IDE 値
　標準フォーマット(CAN_ID_FORMAT_SID), 拡張フォーマット(CAN_ID_FORMAT_EID)
データフレーム／リモートフレーム区分: 受信した RTR 値
　データフレーム(CAN_DATA_FRAME), リモートフレーム(CAN_REMOTE_FRAME)
ID: 受信した ID 値
データ: 受信データ
タイムスタンプ: タイムスタンプ値（受信側で付与）

&r_ide ~ &r_ts に受信したデータが入ります。

can*n*_receive()で指定するメールボックス番号は、事前に can*n*_mb_set_receive()で受信設定を行っておく必要があります。

(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

//引数:メールボックス番号  データフレーム／リモートフレーム区分  送信バイト数  データ
s_ret = can0_send(0, CAN_DATA_FRAME, 1, &s_data[0]);
```

メールボックス番号: 送信に使用するメールボックス番号
データフレーム／リモートフレーム区分: 受信したデータフレーム(CAN_DATA_FRAME), リモートフレーム(CAN_REMOTE_FRAME)
送信バイト数: 1~8
データ: 送信データ

メールボックス 0 から、データ 0x01 を 1 バイト送信する。

can*n*_send()で指定するメールボックス番号は、事前に can*n*_mb_set_send()で送信メールボックスとして設定を行っておく必要があります。

2.4. データの受信

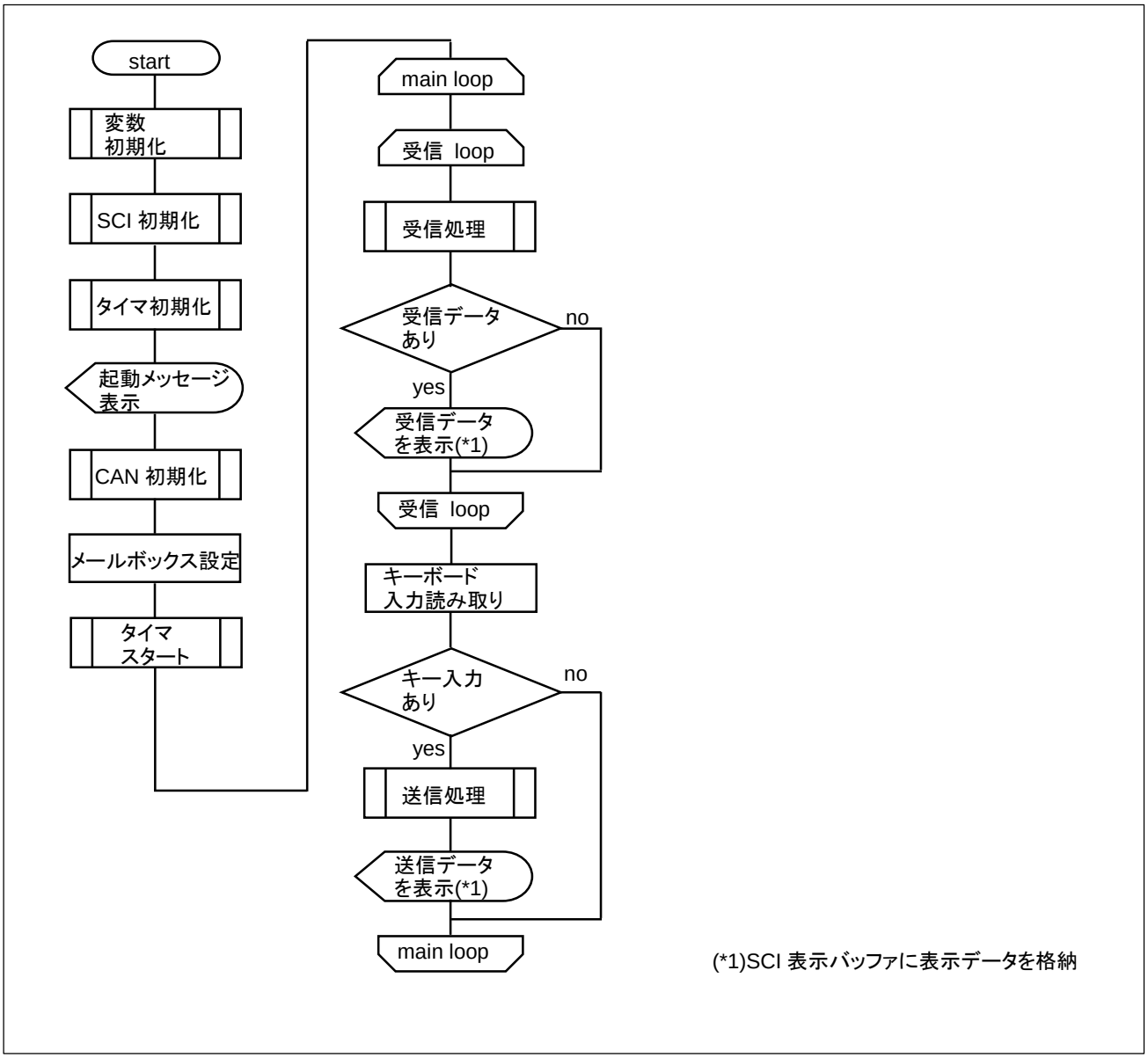
SAMPLE1 でのデータの受信に関しては、メールボックスまでのデータ格納に関しては、（初期化、メールボックス、受信ルール設定が済んでいれば）マイコンのハードウェアが行います。メールボックス、受信バッファにデータが格納されているかは、プログラムで受信関数を呼び出す事で確認を行っています（ポーリングによる受信処理）。そのため、常に受信関数を呼び出して確認を行わないと、データの取りこぼしが生じる可能性があります。受信データの確認に CPU リソースを食うため、スムーズな手法とはいえないと考えます。

（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。）

2.5. SAMPLE1 フローチャート

－処理フロー－

メイン関数　main_s1()



3. サンプルプログラムの説明(SAMPLE2)

3.1. 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。

3.2. 使用する割り込み

3.2.1. RX700,600 系マイコン CAN モジュール

RX71M, RX72M, RX72N, RX64M, RX65/RX671, RX66N 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

※RX65/RX671 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

3.2.2. RA 系マイコン CAN モジュール

RA マイコンでは、CAN モジュールの割り込みは、ICU イベントリンク設定で割り込み番号の割り当てを行います。「ソフトウェア編」3.2 章の GUI 上での設定で割り込み番号が割り当て済みです。

・RA6, RA4, RA2A1（※CAN1 がないマイコンもあります）

割り込み 要求元	名称	対応する 割り込み番号	呼び出される 割り込み関数名
CAN0	CAN0_RXM	4	intr_can0_rxm
CAN0	CAN0_TXM	5	intr_can0_txm
CAN1	CAN1_RXM	6(*1)	intr_can1_rxm
CAN1	CAN1_TXM	7(*1)	intr_can1_txm

(*1)CAN-ch1 が存在するマイコンのみ

・RA2L1

割り込み 要求元	名称	対応する IESLR 番号	呼び出される 割り込み関数名
CAN0	CAN0_RXM	3	intr_can0_rxm
CAN0	CAN0_TXM	0	intr_can0_txm

割り当て番号を変えた場合は、board_settings.h の対応する定義部分を変更してください。

board_settings.h（変更例）

```
//割り込み番号
#define INTR_CAN0_RXM  4
#define INTR_CAN0_TXM  5
#define INTR_CAN1_RXM  6
#define INTR_CAN1_TXM  7
```

3.3. 動作説明

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。SAMPLE1 と同一な点は説明を省略します。

(1)初期化を行う

can0_init();　　※必要に応じて can1_init(); can2_init();

(2)メールボックスの設定

can0_mb_set_send　　※ch0 のメールボックス設定（送信用）
can0_mb_set_receive　　※ch0 のメールボックス設定（受信用）

(3)データの送信

can0_send　　※ch0 送信

(1)~(3)の処理はメイン関数で処理され、SAMPLE1 と同一です。

(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, &ide, &rtr, &id, &data[0], &ts);
    [受信データの表示]
}
```

データを受信した際、受信全メールボックス(0~31)のうち、新規受信デ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。

受信割り込み関数、送信割り込み関数は

can_ch***n***_s2.c (n=0,1,2 : CAN の ch 番号）　　（SAMPLE2 の割り込み関数は_s2.c 内で定義）

内で定義されています。

・割り込み関数名

	受信割り込み関数	送信割り込み関数
RX	Intr_CAN0_RXM0_s(void)	Intr_CAN0_TXM0_s(void)
RA	intr_can0_rxm_s(void)	intr_can0_txm_s(void)

※CAN1(ch1)の場合は、Intr_CAN1_RXM1_s(void)の様に、0 の部分が 1 になります、ch2 も同様です
※元々の割り込み関数は、Intr_CAN0_RXM0(void)の様に_s がない形で定義されています、_s 付きの関数は

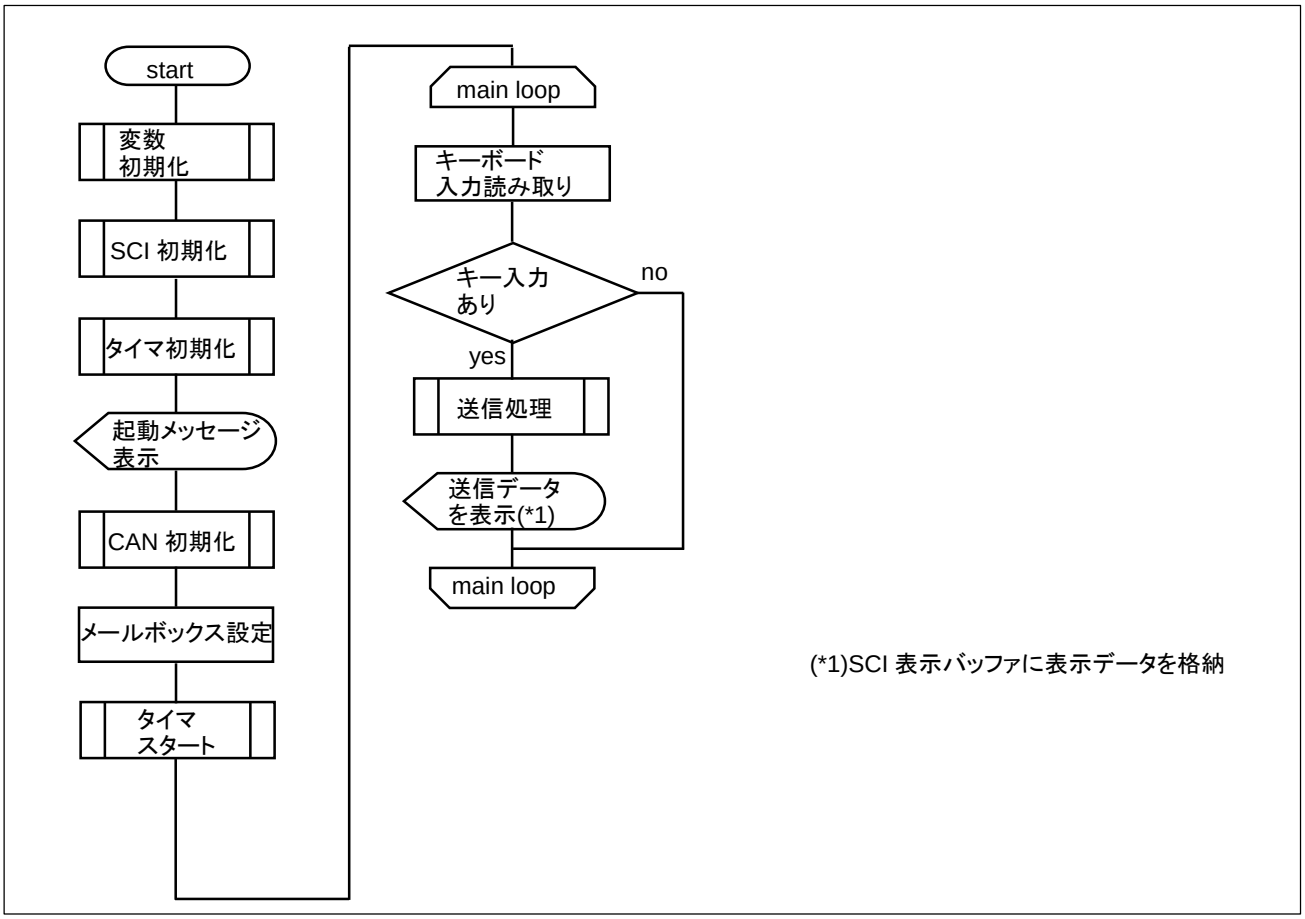
```
Intr_CAN0_RXM0(void)　　//システムで呼び出される割り込み関数
{
//この部分にユーザ関数外で処理する内容を追加するケースを想定して分けている
　　Intr_CAN0_RXM0_s();　　//ユーザ定義の割り込み関数
}
```

の様にワンクッション入れて呼び出しています。（現状、特段深い意味合いはありません。割り込み関数名が変わった場合でも、修正量を小さくする程度の意味合いです。）

3.4. SAMPLE2 フローチャート

－処理フロー－

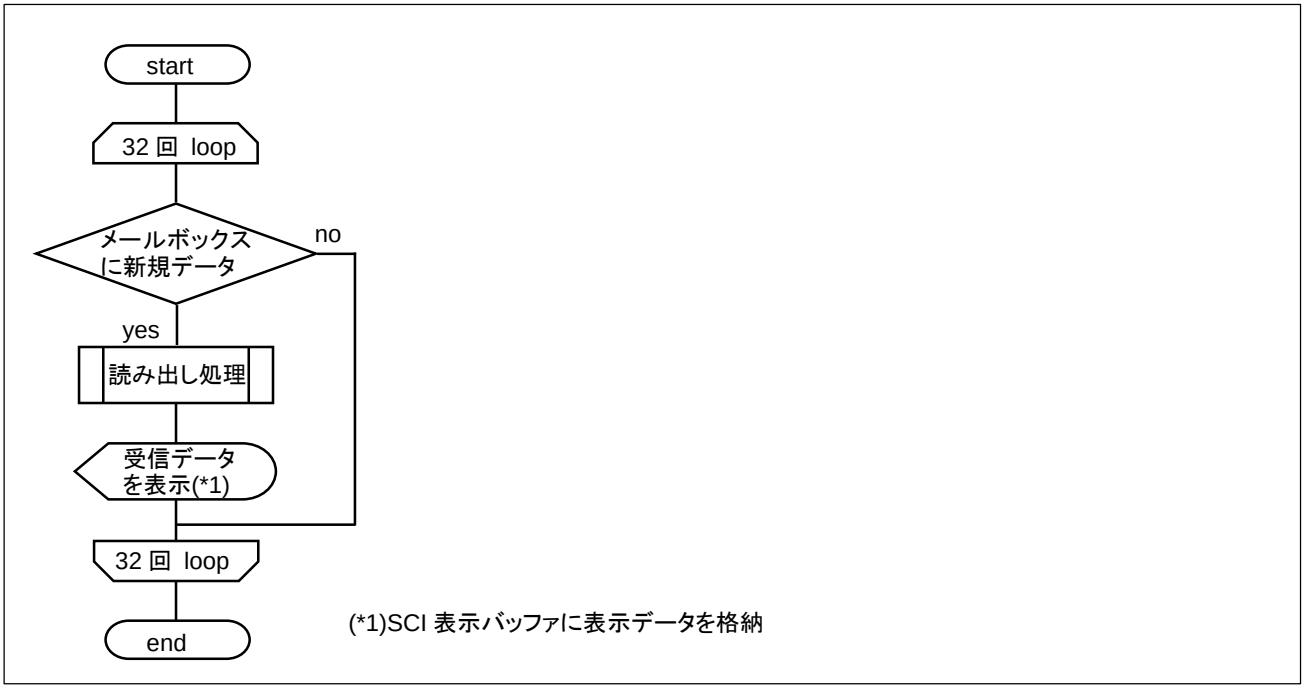
メイン関数　main_s2()



受信割り込み関数

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

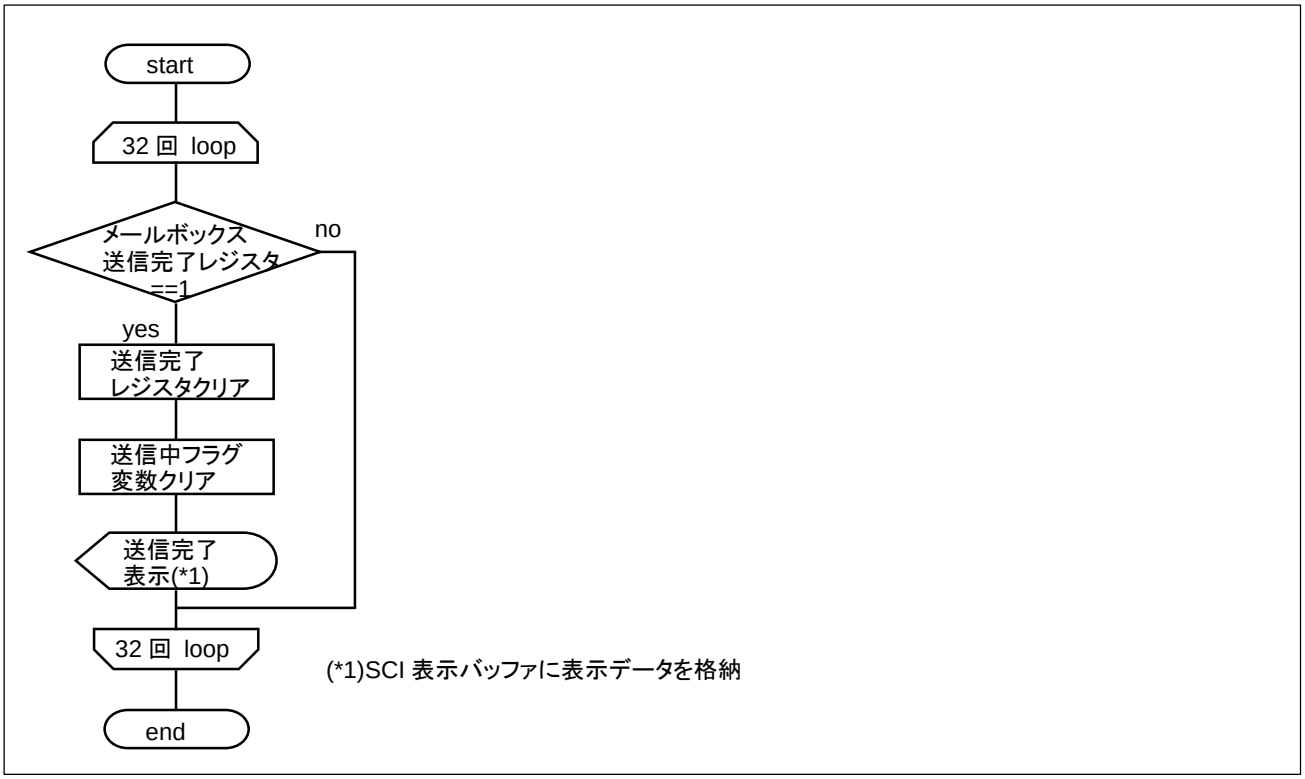
intr_can0_rxm() [ch0]　intr_can1_rxm() [ch1]　[RA]



送信完了割り込み関数

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

intr_can0_txm() [ch0]　intr_can1_txm() [ch1]　[RA]



3.5. 割り込みタイミングとメッセージ表示に関して

データ送信時

(1)データ送信（メイン関数内、can***n***_send, can_fifo_send）
(2)送信データ表示（メイン関数内）
(3)"send finished"表示（割り込み関数内）

上記動作の流れとなりますが、場合によっては(2)の処理が終わらない内に(3)の割り込み（CAN0 send finished の表示）が入ってくる場合もあります。

・期待される表示（(2)完了後に**(3)**）

```
CAN0 data frame send,       MB=0x04 ret=0 id_type=EID id=0x00000000 data=0x01
```

橙字が「メイン関数で表示される部分(2)」で、赤字が「割り込み関数で表示される部分(3)」

・期待と異なる表示（(2)の表示処理が終わらない内に**(3)**の割り込みで表示が出力されてしまう）

```
CAN0 data frame send,       MB=0x04 CAN0 send finished,         MB=0x04
```

上記の様な表示となる事を抑止するため、一連のメッセージ

CAN0 data frame send,　　MB=0x04 ret=0 id_type=EID id=0x00000000 data=0x01

を UART(SCI)の表示バッファへのコピーが終わるまで、一時的に割り込みを禁止するコードを入れています。（VER1.9 で追加）

対象は、割り込みでデータを受信する
main2.c
main3.c
です。

[RX] clrpsw_i();　～メッセージ表示のコード～　setpsw_i();
[RA] __disable_irq();　～メッセージ表示のコード～　__enable_irq();

4. サンプルプログラムの説明(SAMPLE3)

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
・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。

－キーボードから入力したキーと送信データの関係－

・データフレーム送信
キーボード 0~3（SAMPLE1 と同一）

・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	0x0000000	1
w	0x0000001	2
e	0x0000002	4
r	0x0000003	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=4 のときは、0xA1 A2 A3 A4 を返す）

※本サンプルプログラムは、ID=0x0000000 ~ 0x0000003 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 台以上同時に接続させて動作させる場合は、ボード毎に応答する ID を変更する事を推奨致します

4.2. メールボックス設定

メールボックス番号	フォーマット	データフレーム／リモートフレーム	ID	送受信区分
0	標準(SID)		0x000	送信
1	標準(SID)		0x001	送信
2	標準(SID)		0x002	送信
3	標準(SID)		0x003	送信
4	拡張(EID)		0x0000000	送信
5	拡張(EID)		0x0000001	送信
6	拡張(EID)		0x0000002	送信
7	拡張(EID)		0x0000003	送信
8	標準(SID)	データフレーム	0x000	受信
9	標準(SID)	データフレーム	0x001	受信
10	標準(SID)	データフレーム	0x002	受信
11	標準(SID)	データフレーム	0x003	受信
12	拡張(EID)	データフレーム	0x0000000	受信
13	拡張(EID)	データフレーム	0x0000001	受信
14	拡張(EID)	データフレーム	0x0000002	受信
15	拡張(EID)	データフレーム	0x0000003	受信
16	標準(SID)	リモートフレーム	0x000	受信
17	標準(SID)	リモートフレーム	0x001	受信
18	標準(SID)	リモートフレーム	0x002	受信
19	標準(SID)	リモートフレーム	0x003	受信
20	拡張(EID)	リモートフレーム	0x0000000	受信
21	拡張(EID)	リモートフレーム	0x0000001	受信
22	拡張(EID)	リモートフレーム	0x0000002	受信
23	拡張(EID)	リモートフレーム	0x0000003	受信

16~23 をリモートフレーム受信用に割り当て。

4.3. 動作説明

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。SAMPLE2 と同一な点は説明を省略します。

(1)初期化を行う

can0_init();

(2)メールボックスの設定

//引数：メールボックス番号　フォーマット区分　ID
can0_mb_set_receive(16, CAN_ID_FORMAT_SID, **CAN_REMOTE_FRAME**, mb_id[16]); //受信用，標準フォーマット, mb_id[16]=0x0000000

メールボックス番号 16~23 をリモートフレーム受信用に設定。

(3)データの送信[SAMPLE2 との相違点]

//引数：メールボックス番号　**データフレーム／リモートフレーム区分**　送信バイト数　データ
s_ret = **can0_send**(s_mb, **s_rtr**, s_dlc, &s_data[0]);

SAMPLE1~2 では、データフレーム固定のところが、SAMPLE3 では、コマンド(0~3, **q~r**)に応じて、送信する値を変更しています。

(1)~(3)の処理はメイン関数で処理されます。

(i1)受信割り込み関数[SAMPLE2 との相違点]

```
for(i=0; i<32; i++)
{
    if(CAN0.MCTL[i].BIT.RX.NEWDATA != 1) continue;
    ret = can0_receive(i, &ide, &rtr, &id, &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、データの送信を行います。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)で行います。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)を使用します。

(i2)送信割り込み関数[SAMPLE2 との相違点]

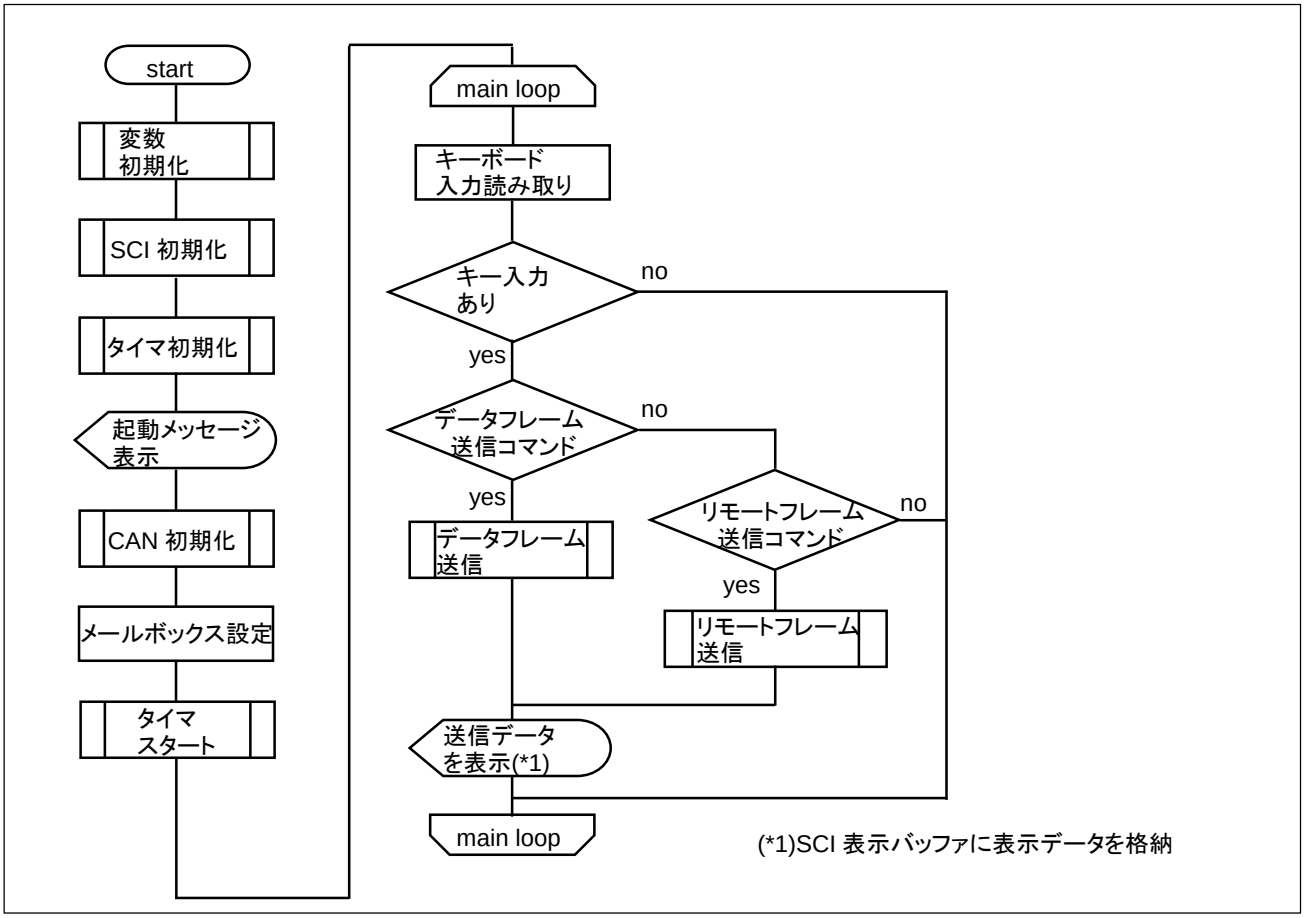
if(can0_remote_frame_request != 1) sciPrintBuf("--¥n");

リモートフレームの場合はデータの区切りが判る様表示を追加しています。

4.4. SAMPLE3 フローチャート

－処理フロー－

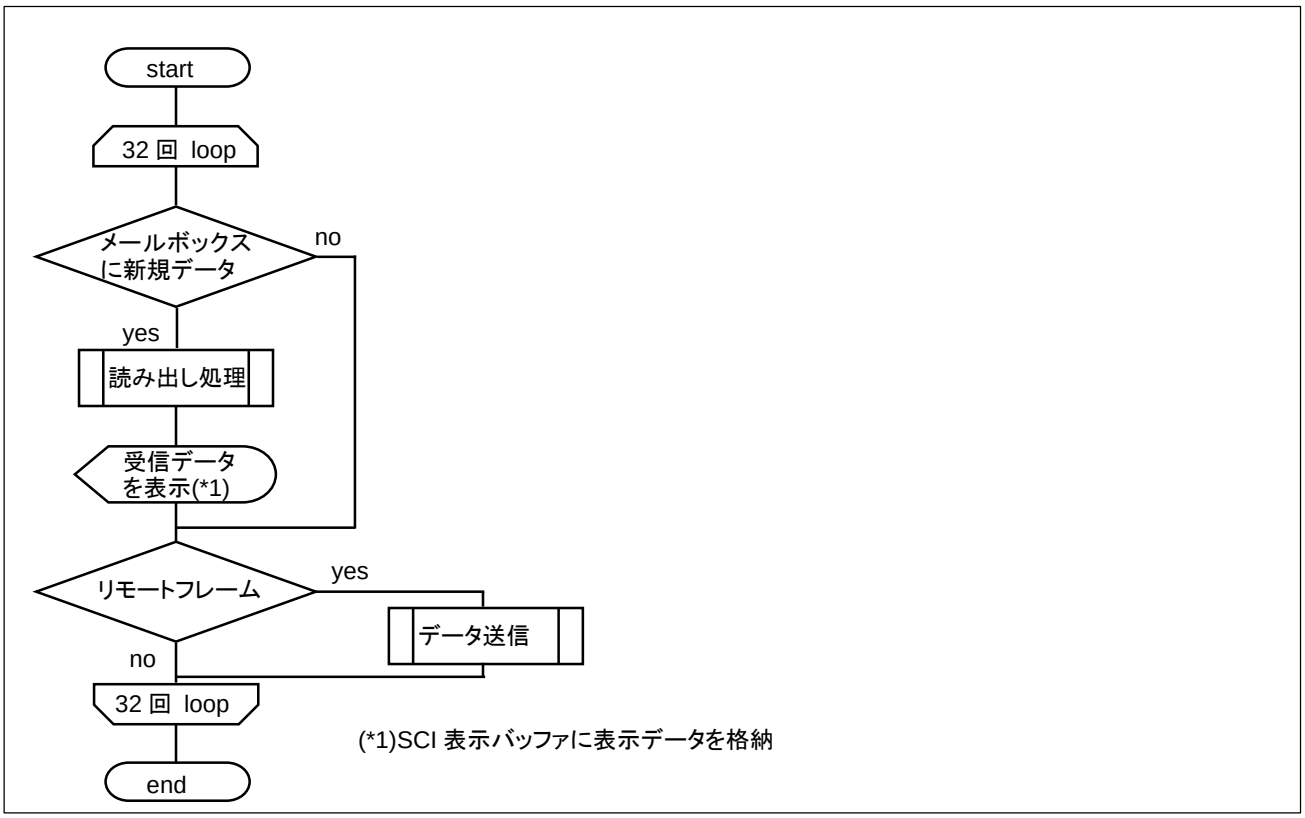
メイン関数　main_s3()



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]　[RX]

Intr_can0_rxm() [ch0]　Intr_can1_rxm() [ch1]　[RA]



5. サンプルプログラムで使用している関数の説明

5.1. 関数仕様

can*n*_init　　(n=0~2)

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]　[RX のみ]

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(*1)で初期化します。

(*1)通信速度は、can_operation.h の定義で変更可能です
通信速度の異なるボード同士は通信が行えません

can*n*_mb_set_send (n=0~2)

概要:メールボックス設定関数

宣言:

int can0_mb_set_send(unsigned char mb, unsigned char ide, unsigned long id) [ch0 向け]

int can1_mb_set_send(unsigned char mb, unsigned char ide, unsigned long id) [ch1 向け]

int can2_mb_set_send(unsigned char mb, unsigned char ide, unsigned long id) [ch2 向け] [RX のみ]

説明:

・メールボックスを送信用として設定

を行います

引数:

mb: メールボックス番号(0~31)

ide: 標準／拡張フォーマット区分

CAN_ID_FORMAT_SID(0) 標準フォーマット(ID=11bit)

CAN_ID_FORMAT_EID(1) 拡張フォーマット(ID=29bit)

id: ID を指定

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*_mb_set_receice (n=0~2)

概要:メールボックス設定関数

宣言:

int can0_mb_set_receive(unsigned char mb, unsigned char ide, unsigned char rtr, unsigned long id) [ch0 向け]

int can1_mb_set_receive (unsigned char mb, unsigned char ide, unsigned char rtr, unsigned long id) [ch1 向け]

int can2_mb_set_receive (unsigned char mb, unsigned char ide, unsigned char rtr, unsigned long id) [ch2 向け] [RX のみ]

説明:

・メールボックスを受信用として設定

を行います

引数:

mb: メールボックス番号(0~31)

ide: 標準／拡張フォーマット区分

CAN_ID_FORMAT_SID(0) 標準フォーマット(ID=11bit)

CAN_ID_FORMAT_EID(1) 拡張フォーマット(ID=29bit)

rtr: データフレーム／リモートフレーム区分

CAN_DATA_FRAME(0)　　データフレーム（データ送信）
CAN_REMOTE_FRAME(1)　リモートフレーム（相手にデータ要求）

id: ID を指定

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　　(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)

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

int can2_send(unsigned char mb, unsigned char rtr, unsigned char dlc, unsigned char *data)　[RX のみ]

説明:

・データの送信

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

CAN_DATA_FRAME(0)　　データフレーム（データ送信）

CAN_REMOTE_FRAME(1)　リモートフレーム（相手にデータ要求）

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 モジュール系は、メールボックス番号とフォーマット区分(IDE)、送信 ID の対応が既に設定済みなので、送信関数では、ID と IDE は引数として与えない仕様としています。

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 char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

int can1_receive(unsigned char mb, unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

int can2_receive(unsigned char mb, unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts) [RX のみ]

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)

*ide: 標準／拡張フォーマット区分を格納するポインタ(unsigned char x1)

CAN_ID_FORMAT_SID(0)　標準フォーマット(ID=11bit)

CAN_ID_FORMAT_EID(1)　拡張フォーマット(ID=29bit)

*rtr: データフレーム／リモートフレーム区分を格納するポインタ(unsigned char x1)

CAN_DATA_FRAME(0)　　データフレーム（データ送信）

CAN_REMOTE_FRAME(1)　リモートフレーム（相手にデータ要求）

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

（※標準フォーマットでも unsigned long 型のポインタでデータを受け取ってください）

*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（DLC=0 のデータはデータなしとして扱う）

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。

5.2. プログラムで使用している変数・定数

5.2.1. グローバル変数

unsigned short can*n*_error_flag; (n=0~2)

レジスタ書き換えタイムアウトの際セットされるエラーフラグ変数。

unsigned long can*n*_mb_send_flag; (n=0~2)

データ送信中であることを示すフラグ変数。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。

unsigned long can*n*_remote_frame_request; (n=[], 0~2)

リモートフレームであることを示すフラグ変数。リモートフレーム送信前にフラグを立て、リモートフレームの処理完了時にフラグを落とす。

5.2.2. 定数定義

can¥can.h
で定義

```
#define  CAN_BPS_1M  1        //通信速度  1Mbps
#define  CAN_BPS_500K  2        //通信速度  500kbps
#define  CAN_BPS_250K  3        //通信速度  250kbps
#define  CAN_BPS_125K  4        //通信速度  125kbps
```

通信速度設定。common¥can_operation.h 内で速度を変更する際に指定。
（数値は、クロック分周比と対応していますので、定義値を任意の数値に変更できる訳ではありません）

```
#define CAN_ID_FORMAT_SID  0        //標準フォーマット(ID=11bit)
#define  CAN_ID_FORMAT_EID  1        //拡張フォーマット(ID=11bit)
```

標準／拡張フォーマットール定義値。

```
#define CAN_DATA_FRAME0        //データフレーム
#define CAN_REMOTE_FRAME1        //リモートフレーム
```

データフレーム／リモートフレーム定義値。

```
#define CAN_CLOCK_PCLK  0        //PCLK(B)を CAN のクロックソースとして使用
#define CAN_CLOCK_XTAL  1        //XTAL を CAN のクロックソースとして使用
```

使用するクロックソース定義。

```
#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  750e-6/((1.0/(ICLK*1e6))*19)        //約 750us のウェイト（19 命令/loop で計算）
```

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。

```
#define CAN_REG_REWRITE_TIMEOUT  0x0010
```

レジスタ値の書き換えタイムアウト時のエラーコード定義。

```
#define CAN_INTERRUPT_DEBUG
```

定義時、デバッグ向けに、割り込み時にポートを反転させる。

5.3. CAN マクロのメールボックスに関して

本サンプルプログラムでは、CAN マクロ向けのメールボックス設定関数、送信関数は

can0_mb_set_send(unsigned char mb, unsigned char ide, unsigned long id)　[ch0 向け]
can0_send(unsigned char mb, unsigned char rtr, unsigned char dlc, unsigned char *data)　[ch0 向け]

となっています。メールボックスの構成としては、

IDE	RTR		SID[10]	SID[9]	SID[8]	SID[7]	SID[6]
SID[5]	SID[4]	SID[3]	SID[2]	SID[1]	SID[0]	EID[17]	EID[16]
EID[15]	EID[14]	EID[13]	EID[12]	EID[11]	EID[10]	EID[9]	EID[8]
EID[1]	EID[6]	EID[5]	EID[4]	EID[3]	EID[2]	EID[1]	EID[0]
				DLC[3]	DLC[2]	DLC[1]	DLC[0]
DATA0							
DATA1							
DATA2							
DATA3							
DATA4							
DATA5							
DATA6							
DATA7							
TS[15:8]							
TS[7:0]							

IDE: 標準／拡張フォーマット区分
RTR: データフレーム／リモートフレーム区分
SID: 標準 ID(11bit), 拡張 ID 18~28bit
EID: 拡張 ID 0~17bit
DATA: データ（最大 8 バイト）
TS: タイムスタンプ(16bit)
　となっており、TS を除く太字の部分はデータ送信時に使用されます（送信データに含まれる）。
（※IDE=0, 標準フォーマットの場合は、EID の部分は送信に使用されません）
（※上記のメールボックスが 32 個あります）

本サンプルプログラムでは、送信時に送信関数で赤字の部分を引数として与える使用としており、紫の部分はメールボックス設定時に値をセットしています。

どのタイミングで、メールボックスに値をセットするかは自由です。

・送信関数で（送信時）IDE, RTR, ID(SID, EID), DATA の全てをセットする
・送信関数では DATA のみセットする（他のパラメータは事前にセットしておく）
・DTC の機能を用いてメールボックスにデータ転送する

等、実際のアプリケーションでは、都合の良い手法を採用して頂ければと思います。

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.1.8.0.0	2022.8.9	—	ソフトウェア編マニュアルから分離、独立
REV.1.9.0.0	2023.6.23	P13,15 P18	RA での割り込み番号の変更 メッセージの表示乱れ対策に割り込み禁止コードの追加 タイマ割り込み関数の削除

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

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 スタータキット RX/RA
CAN スタータキット SmartRX
CAN モジュール編 ソフトウェアマニュアル

株式会社 北斗電子
 2023 年 6 月 23 日改訂 REV.1.9.0.0 (230623)