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.1.10.0.0

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。

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、途中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.c	グローバル変数定義、割り込み設定（RX のみ）
	can.h	グローバル変数定義ヘッダ
	can_error_handle.c	エラー処理関数
	can_error_handle.h	エラー処理関数ヘッダ
	can_general.c	タイミングパラメータ設定関数,リングバッファアクセス関数
	can_general.h	上記ヘッダ
	can_intr.c(*1)	エラー割り込み関数
	can_intr.h(*1)	エラー割り込み関数ヘッダ
source¥CAN_module¥can_ch0		CAN モジュール ch0
	can_ch0.c	ch0 向け各種関数定義
	can_ch0.h	ch0 関数ヘッダ
	can_ch0_intr.c	ch0 割り込み関数
	can_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥CAN_module¥can_ch1		CAN モジュール ch1
	can_ch1.c	ch1 向け各種関数定義
	can_ch1.h	ch1 関数ヘッダ
	can_ch1_intr.c	ch1 割り込み関数
	can_ch1_intr.h	ch1 割り込み関数ヘッダ
source¥CAN_module¥can_ch2		CAN モジュール ch2
(*2)	can_ch2.c(*2)	ch2 向け各種関数定義
	can_ch2.h(*2)	ch2 関数ヘッダ
	can_ch2_intr.c(*2)	ch2 割り込み関数
	can_ch2_intr.h(*2)	ch2 割り込み関数ヘッダ
source¥CAN_module¥main		メイン関数
	main_monitor.c	SAMPLE_MONITOR 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

(*1)(*2)RX 向けソースにのみ存在（RA 向けには、当該フォルダ、ファイルは存在しません）

(*1)RX の CAN モジュールでは、エラー割り込みはグループ化されており、ch 毎に独立していないため、エラー割り込みは ch 毎のフォルダ以下のファイルではなく、can フォルダ以下のファイルとして持たせています。
(*2)RA は、現状 ch2 をサポートするマイコンが存在しないため、ch2 向けのフォルダは用意していません。

・共通

フォルダ	ファイル	説明
source¥ALL¥common		設定定義フォルダ
	board_setting.c	ボード名定義
	can_common.h	CAN で使用する定数定義
	can_operation.c	動作に関わる定数定義
	can_operation2.c	動作に関わる定数定義（SAMPLE1~3 の動作）
source¥ALL¥sci		端末表示(UART)フォルダ
	sci.c	端末表示(UART)ソース
	sci.h	端末表示(UART)ヘッダ

・共通（RA のみ）

フォルダ	ファイル	説明
source¥ALL¥clock		クロック設定フォルダ
	clock.c	ボード名定義
	clock.h	CAN で使用する定数定義
source¥ALL¥intr		割り込み定義フォルダ
	inrt.c	割り込み優先度と有効化処理
	intr.h	上記ヘッダ

・マイコンボード毎の設定

フォルダ	ファイル	説明
settings(*1)		ボード毎の設定フォルダ
	board.h	ボード毎の定義
	program_select.h	サンプルプログラム選択

(*1)

RX では、

[プロジェクト名]¥src¥usr_src¥settings

RA では、

mcu¥[MCU 名]¥settings

2. 初期化

CAN モジュールを初期化して、通信可能となるまでの流れに関して説明します。

(1)CAN モジュール初期化

```
can0_init();
can1_init();
can2_init();
can_init(); //エラー割り込みを使用する場合に必要[RX のみ]
```

can0_init() は、CAN-ch0 の初期化です。使用する、CAN-ch のみ、can***n***_init()（***n***=0~2、CAN の ch 番号）で初期化を行ってください。以下の関数でも、can***n*** の ***n*** は CAN の ch 番号を示しますので、ch に応じた can***n***_....を使い分けてください。RX マイコンでエラー割り込みを使用する際は、can_init()を実行してください。

(2)メールボックスの設定

```
can0_mbox_set_send(0);
```

CAN-ch0 のメールボックス 0 を送信メールボックスとして設定する場合。

※CAN モジュールのリセット後の値は、送信メールボックスの設定となりますので、省略可能。
（一度受信用に設定したメールボックスを送信用に変更したい場合等は設定要。）

```
can0_mbox_set_receive(4, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
```

CAN-ch0 のメールボックス 4 を
・標準 ID(11bit, IDE=0 のデータ)
・データフレーム（RTR=0 のデータ）
・ID=0x000 のデータ
を受信する様に設定する。

(3)動作モードへの移行

```
can0_operate();
```

CAN-ch0 を動作モードに移行させる。
以降、CAN メッセージの送受信が行えるようになります。

データの送受信は、「メールボックス」を使用する方法と、「メールボックス FIFO（以下 FIFO)」を使用する方法があります。

（FIFO は、First In First Out（先入れ先出し方式）のバッファで、CAN のメッセージを 4 つまで溜めておく事ができます。）

送受信方法は、can_operation.h（SAMPLE1~3 の動作は、can_operation2.h）内で選択可能です。

・受信方法

```
#define CAN_RX_METHOD      CAN_RX_MBOX          //メールボックスを使用(*1)
#define CAN_RX_METHOD      CAN_RX_MBOXFIFO    //FIFO を使用
#define CAN_RX_METHOD      (CAN_RX_MBOXFIFO | CAN_RX_MBOX) //メールボックス+FIFO を使用(*2)
```

・送信方法

```
#define CAN_TX_METHOD      CAN_TX_MBOX              //メールボックスを使用(*1)
#define CAN_TX_METHOD      CAN_TX_MBOXFIFO         //FIFO を使用(*2)
#define CAN_TX_METHOD      (CAN_TX_MBOXFIFO | CAN_TX_MBOX) //メールボックス+FIFO を使用
```

(*1)SAMPLE1 のデフォルト
(*2)SAMPLE2~3 のデフォルト

FIFO のみを使用する場合は、(2)のメールボックスの設定の手順は不要となります。

※受信、送信のどちらかで FIFO を使用する設定とした場合、FIFO が有効となります。
FIFO 有効時も、メールボックスは利用可能です。
FIFO 有効時、メールボックス 0~23 が使用できます。FIFO 無効時（受信、送信ともメールボックスを使う設定）は、メールボックス 0~31 が使用できます。

3. データの送受信

3.1. データの送信

CAN のメッセージを送信するには、以下の様に行います。

```
int ret;
can_message msg;

msg.id = 0x00000001;
msg.ide = CAN_ID_FORMAT_EID;
msg.rtr = CAN_DATA_FRAME;
msg.dlc = 2;
msg.data[0] = 0x01;
msg.data[1] = 0x02;
```

(1)メールボックスを使用した送信

```
ret = can0_mbox_send(0, &msg);
```

0 は、メールボックス番号です。初期時に、can0_mbox_set_send()で送信用メールボックスとして設定したメールボックス番号を指定してください。受信用に設定したメールボックスを使用しての送信はできません。

(2)FIFO を使用した送信

```
ret = can0_mboxfifo_send(&msg);
```

拡張 ID, ID=0x0000001, データフレーム, 0x01, 0x02 の 2 バイトを送信する場合、上記の関数呼び出しとなります。msg は CAN のメッセージ構造体で、この構造体を引数として CAN のメッセージのやり取りを行います。

関数の戻り値は、0(=CAN_RET_SUCCESS):正常終了, 0 以外:エラーとなります。(詳細は関数仕様の項を参照ください。)

3.2. 受信条件の設定

CAN のメッセージを受信すると、データはメールボックスに格納されますが、受信したデータが無条件でメールボックスに格納される訳ではありません。予め、受信するデータの条件設定が必要です。

```
can0_mbox_set_receive(4, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
```

標準 ID フォーマット(CAN_ID_FORMAT_SID)で、データフレーム(CAN_DATA_FRAME)で、ID が 0x000 のデータを受信した際に、メールボックス 4 にデータを格納する。

```
can0_mbox_set_receive(9, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000001);
```

拡張 ID フォーマット(CAN_ID_FORMAT_EID)で、データフレーム(CAN_DATA_FRAME)で、ID が 0x00000001 のデータを受信した際に、メールボックス 9 にデータを格納する。

ここで、設定した条件にマッチしたデータを受信した際、設定したメールボックスにデータが格納されます。

3.3. データの受信

CAN のメッセージを受信するには、以下の様に行います。

```
can_message msg;
```

(1)メールボックスを使用した受信

```
ret = can0_mbox_receive(4, &msg);
```

4 番のメールボックスのデータを取り出す。

※can0_mbox_set_receive()で、予め 4 番のメールボックスの受信条件の設定が必要です。

ret が、-1(=CAN_RET_NODATA)の場合、メールボックス 4 にはデータが受信されていません。ret が 1 以上の場合は、受信データが msg 構造体にコピーされています。(ret の値は、受信したデータバイト数となります。)

CAN のデータ受信に関しては、受信条件にマッチした CAN のメッセージを受信すると、メールボックスへの格納が行われます(CAN モジュールのハードウェアが行うので、メールボックスの格納までは、ユーザプログラムの関与なしに行われます)。アプリケーションプログラムでは、メールボックス(要はレジスタですが)に格納されているデータを参照(can0_mbox_receive()ではデータコピー)する形となります。

メールボックスへの格納は、can0_mbox_set_receive()で設定した「ID 値」、「拡張・標準 ID 区分（IDE 値）」、「データフレーム・リモートフレーム区分（RTR 値）」の 3 値が一致した場合に行われます。DLC 値（データバイト数）は、任意の値のデータがメールボックスに格納されます。（DLC 値によるフィルタリングは、マイコンの機能としては可能ですが、本サンプルプログラムでは未使用です。）

(2)FIFO を使用した受信

```
ret = can0_mboxfifo_receive(&msg);
```

FIFO を使用する場合は、受信の条件設定はありませんが、本サンプルプログラムでは

・ID 値は任意の値
・拡張 ID(IDE=1)
・データフレーム、リモートフレーム両方(RTR=0,1)
・DLC 値は任意の値

のデータを受信した際に、受信 FIFO にデータ格納が行われます。

※標準 ID のデータは、FIFO では受信しません
（can_operation.h で
#define CAN_ID_TYPE　　CAN_ID_FORMAT_SID
と定義して、標準 ID のみ取り扱う様に設定した場合は、「標準 ID(IDE=0)」「データフレームとリモートフレーム両方(RTR=0/1)」のデータを受信した際に、受信 FIFO にデータ格納が行われます）

※受信に FIFO を使う設定で、標準 ID のデータを受信したい場合は、メールボックスでの受信としてください
（もしくは can の初期化のソースコードを変更してください）
受信 FIFO を使う場合、
・拡張 ID，データフレーム(*)
・拡張 ID，リモートフレーム(*)
・標準 ID，データフレーム
・標準 ID，リモートフレーム
の、4 条件の内、2 条件の受信設定が可能です。本サンプルプログラムでは、(*)の 2 条件を受信条件に設定しています。

受信 FIFO を使用する場合、受信関数呼び出し時に、FIFO に複数のデータが格納されているケースもあり得ます。全てのデータを取り出す場合は、受信 FIFO が空になるまでループ処理を行ってください。

－コード例－

```
int i;
can_message msg[4];
i = 0;
while(0)
{
   ret = can0_mboxfifo_receive(&msg[i++]);
   if (ret == CAN_RET_NODATA) break;
}
```

4. 割り込み

CAN モジュールの割り込みは、以下の様になっています。

4.1. マイコンに拠る割り込みの相違

4.1.1. 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 に割り当てられています。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 130~139 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	呼び出される 割り込み関数名	備考
50	CAN0	RXF0	128	Intr_CAN0_RXF0	受信 FIFO
51	CAN0	TXF0	129	Intr_CAN0_TXF0	送信 FIFO
52	CAN0	RXM0	130	Intr_CAN0_RXM0	受信メールボックス
53	CAN0	TXM0	131	Intr_CAN0_TXM0	送信メールボックス
54	CAN1	RXF1	132	Intr_CAN0_RXF1	受信 FIFO
55	CAN1	TXF1	133	Intr_CAN0_TXF1	送信 FIFO
56	CAN1	RXM1	134	Intr_CAN0_RXM1	受信メールボックス
57	CAN1	TXM1	135	Intr_CAN0_TXM1	送信メールボックス
58	CAN2	RXF2	136	Intr_CAN0_RXF2(*1)	受信 FIFO
59	CAN2	TXF2	137	Intr_CAN0_TXF2(*1)	送信 FIFO
60	CAN2	RXM2	138	Intr_CAN0_RXM2(*1)	受信メールボックス
61	CAN2	TXM2	139	Intr_CAN0_TXM2(*1)	送信メールボックス

(*1)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	呼び出される 割り込み関数名	備考
CAN0	RXF0	170	Intr_CAN0_RXF0	受信 FIFO
CAN0	TXF0	171	Intr_CAN0_TXF0	送信 FIFO
CAN0	RXM0	172	Intr_CAN0_RXM0	受信メールボックス
CAN0	TXM0	173	Intr_CAN0_TXM0	送信メールボックス

エラー割り込みは、グループ BE0 の割り込みが割り当てられています。

割り込み 要求元	名称	割り込み番号
ICU	GROUPBE0	106

グループ	番号	割り込み 要求元	名称	呼び出される 割り込み関数名	備考
BE0	0	CAN0	ERS0	Intr_CAN0_ERS0	エラー
	1	CAN1	ERS1	Intr_CAN0_ERS1	エラー
	2	CAN2	ERS2	Intr_CAN0_ERS2	エラー

CAN のどの ch で発生したエラーも全て、106 番の割り込みで処理されます。呼ばれる関数は、ch 毎に別となります。

RX では、受信・送信、FIFO・メールボックスで独立。エラーは、グループ BE0 割り込みでグルーピング（エラー割り込みに関しては、割り込みベクタが独立はしていない）という形です。

4.1.2. 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、FSP の割り込み設定(Interrupts Configuration)で割り込み名に対し、呼び出される関数名を定義します。

割り込み 要求元	名称	呼び出される 割り込み関数名	備考
CAN0	CAN0_ERS	intr_can0_ers	エラー
CAN0	CAN0_RXF	intr_can0_rxf	受信 FIFO
CAN0	CAN0_TXF	intr_can0_txf	送信 FIFO
CAN0	CAN0_RXM	intr_can0_rxm	受信メールボックス
CAN0	CAN0_TXM	intr_can0_txm	送信メールボックス
CAN1	CAN1_ERS	intr_can1_ers(*1)	エラー
CAN1	CAN1_RXF	intr_can1_rxf(*1)	受信 FIFO
CAN1	CAN1_TXF	intr_can1_txf(*1)	送信 FIFO
CAN1	CAN1_RXM	intr_can1_rxm(*1)	受信メールボックス
CAN1	CAN1_TXM	intr_can1_txm(*1)	送信メールボックス

(*1)CAN-ch1 が存在するマイコンのみ

※FSP の設定で関数名の設定を上記以外として場合は、割り込み関数
can_ch***n***_intr.c, can_ch***n***_intr.h
内の関数名を変更する必要があります。

基本的には、CAN-ch 毎に「エラー」「受信 FIFO」「送信 FIFO」「受信メールボックス」「送信メールボックス」の 5 種類の割り込みとなります。

SAMPLE1 では、割り込みは未使用。SAMPLE2~では、割り込みを使用しています。

4.2. エラー割り込み

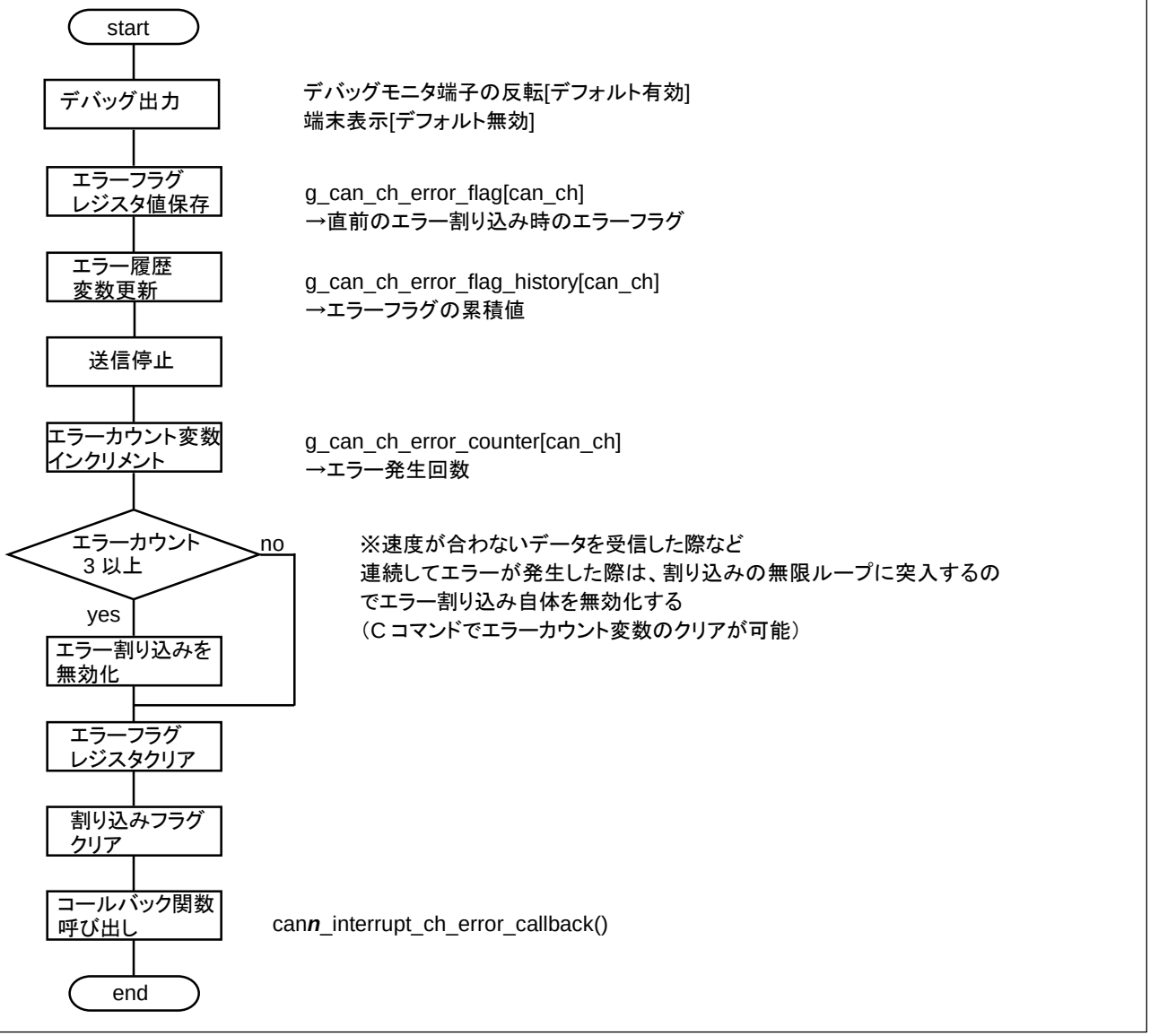
・エラー割り込み対象
　バスロック
　オーバロードフレーム送信
　オーバラン
　バスオフ復帰
　バスオフ開始
　エラーパッシブ
　エラーワーニング
　バスエラー

・関数名
RX: intr_CAN***n***_ERS***n***
RA: intr_can***n***_ers
(***n*** = CAN-ch 番号)

　RX では、
can¥can_intr.c
　RA では、
can_ch***n***¥can_ch***n***_intr.c （***n***=0,1)
内に割り込み処理のソースコードが記載されています。

－処理内容－
・エラーフラグのグローバル変数への保存
・累積エラーフラグ変数の更新
・送信停止
・エラーカウント数のインクリメント
・割り込みが累積 3 回を超えた場合エラー割り込みの無効化
・エラーフラグクリア
・割り込みフラグクリア
・コールバック関数の呼び出し
→can***n***_interrupt_ch_error_callback() を呼び出しますので、ユーザ側で処理したい内容はコールバック関数内に記載してください。(コールバック関数は、本サンプルプログラムではメイン関数のファイル(main_s***x***.c)に記載しています。)

ーフローチャートー


start
デバッグ出力
エラーフラグ
レジスタ値保存
エラー履歴
変数更新
送信停止
エラーカウント変数
インクリメント
エラーカウント
3 以上
no
yes
エラー割り込みを
無効化
エラーフラグ
レジスタクリア
割り込みフラグ
クリア
コールバック関数
呼び出し
end
デバッグモニタ端子の反転[デフォルト有効]
端末表示[デフォルト無効]
g_can_ch_error_flag[can_ch]
→直前のエラー割り込み時のエラーフラグ
g_can_ch_error_flag_history[can_ch]
→エラーフラグの累積値
g_can_ch_error_counter[can_ch]
→エラー発生回数
※速度が合わないデータを受信した際など
連続してエラーが発生した際は、割り込みの無限ループに突入するの
でエラー割り込み自体を無効化する
（C コマンドでエラーカウント変数のクリアが可能）
can*n*_interrupt_ch_error_callback()

4.3. 受信 FIFO 割り込み

・関数名

RX: intr_CAN***n***_RXF***n***
RA: intr_can***n***_rxf

－処理内容－

・受信データのリングバッファへのデータコピー

→g_can_recv_buf[can_ch][index]に受信データをコピーします（can_ch は、CAN の物理 ch, index はデフォルトでは 0~15 です。index はユーザが意識する必要はありません。）

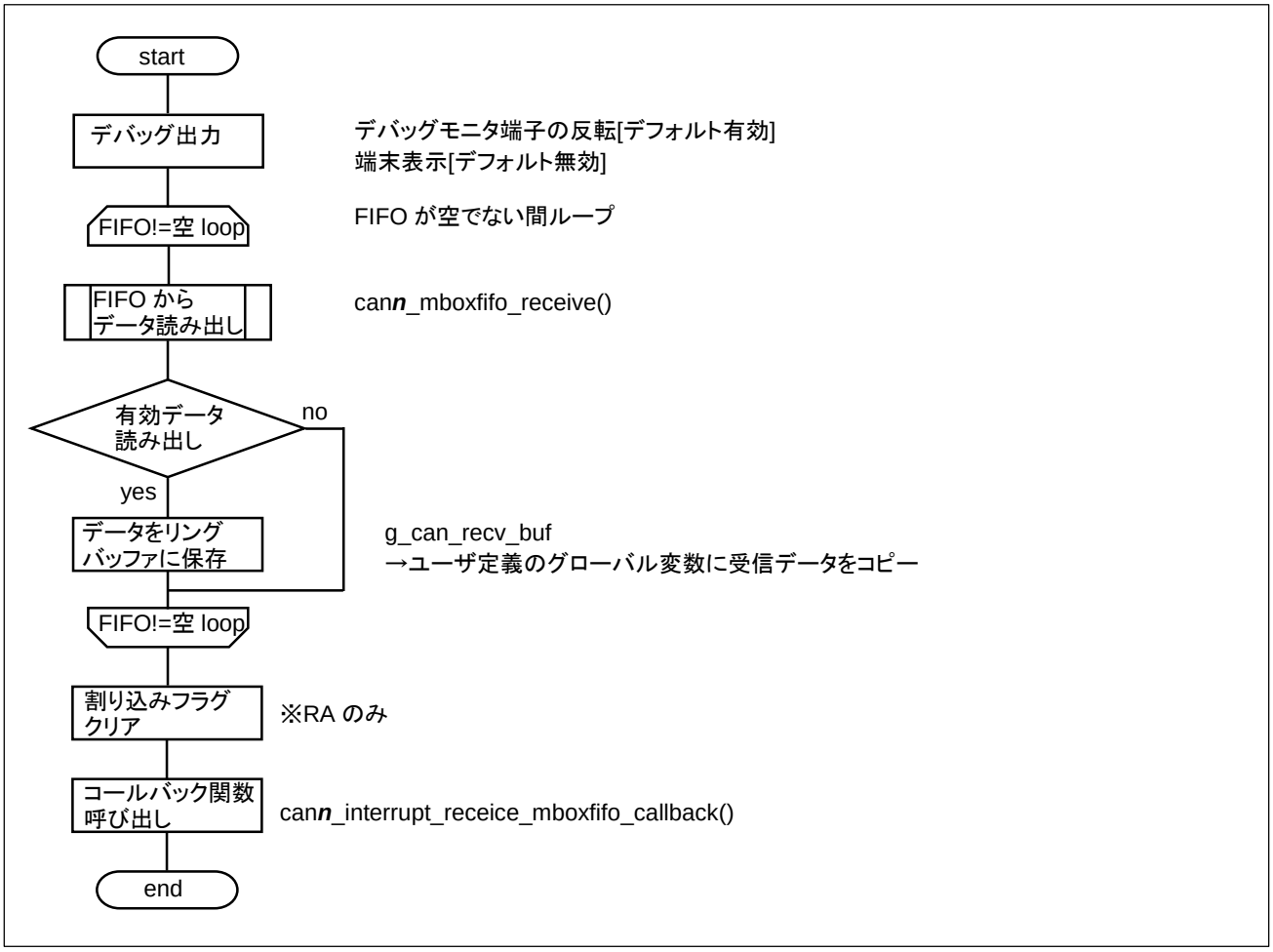
※リングバッファに蓄えられている受信データを取り出す際は、can_read_data()関数を使用します

・割り込みフラグクリア

・コールバック関数の呼び出し

→can***n***_interrupt_receive_mboxfifo_callback()

－フローチャート－



割り込み関数内では、受信データを cann_mboxfifo_receive()関数で読み出しを行い、ユーザプログラムで定義した、受信データ用のリングバッファにコピーします。

リングバッファは、CAN の物理 ch に対応させており、
g_can_recv_buf[CAN_CH0]　:CAN-ch0 で受信したデータを格納
g_can_recv_buf[CAN_CH1]　:CAN-ch1 で受信したデータを格納
g_can_recv_buf[CAN_CH2]　:CAN-ch2 で受信したデータを格納
としています。（デフォルトでは、バッファのサイズは 16 で、15 個までのデータを保持できます）

CAN-ch0 のリングバッファのデータを取り出す際は、

```
can_message msg;
int ret;
ret = can_read_data(CAN_CH0, &msg);    // g_can_recv_buf[CAN_CH0]  のデータを取り出し
```

とします。

4.4. 送信 FIFO 割り込み

・関数名

RX: intr_CAN***n***_TXF***n***
RA: intr_can***n***_txf

－処理内容－

・送信フラグ変数のセット

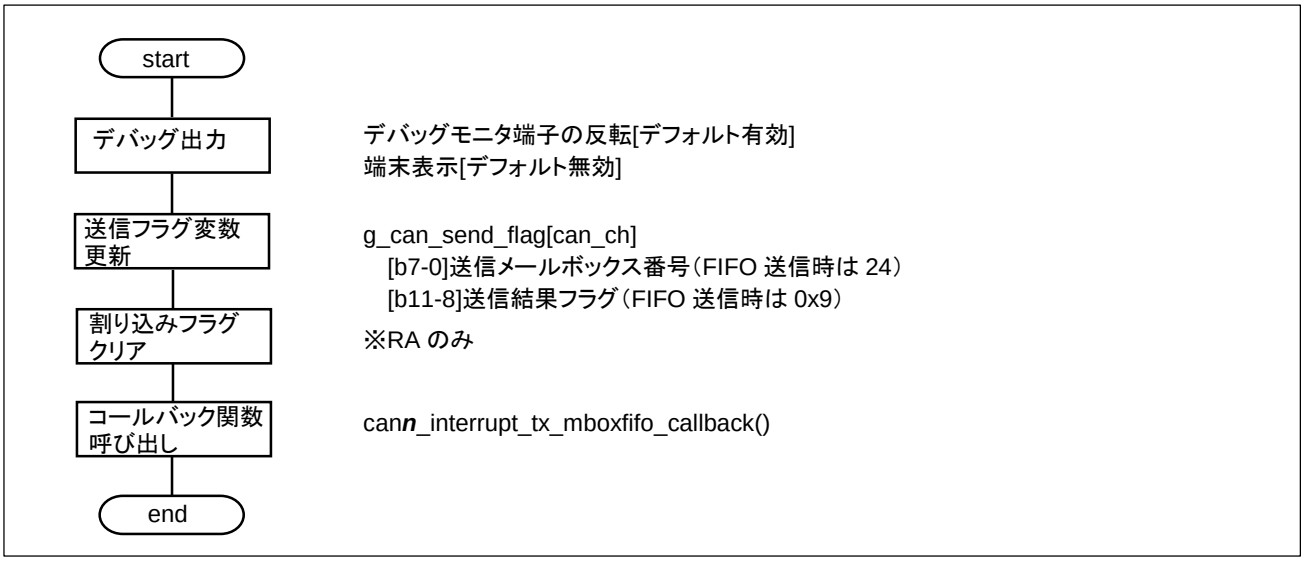
→g_can_send_flag[can_ch]に、送信済みであるフラグをセットします

・割り込みフラグクリア

・コールバック関数の呼び出し

→can***n***_interrupt_tx_mboxfifo_callback()

－フローチャート－



4.5. メールボックス受信割り込み

・関数名

RX: intr_CAN***n***_RXM***n***
RA: intr_can***n***_rxm

－処理内容－

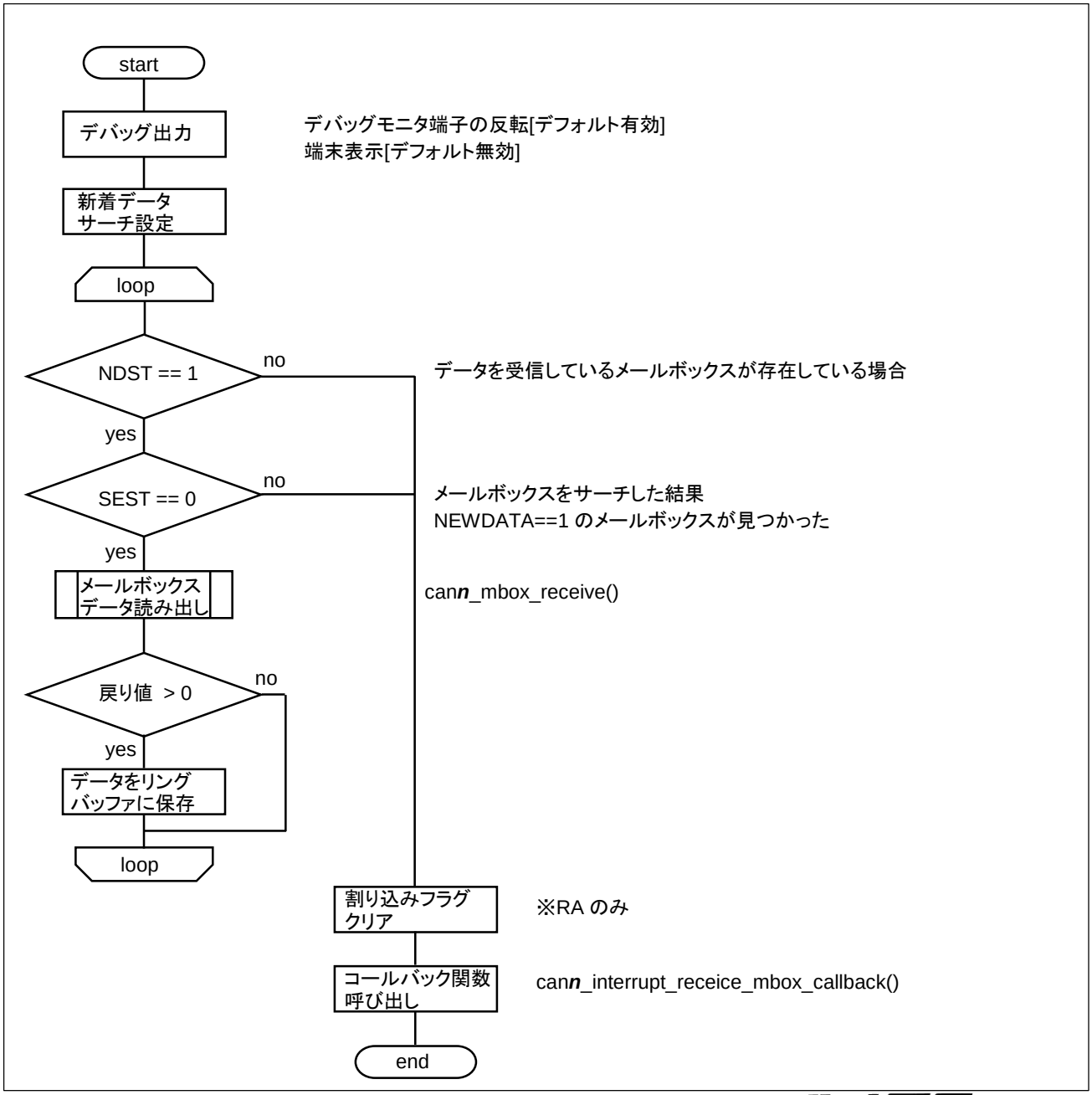
・受信データのリングバッファへのデータコピー

・割り込みフラグクリア

・コールバック関数の呼び出し

→can***n***_interrupt_receive_mbox_callback()

－フローチャート－



4.6. メールボックス送信割り込み

・関数名

RX: intr_CAN***n***_TXM***n***
RA: intr_can***n***_txm

－処理内容－

・CAN レジスタの SENTDATA フラグを落とす

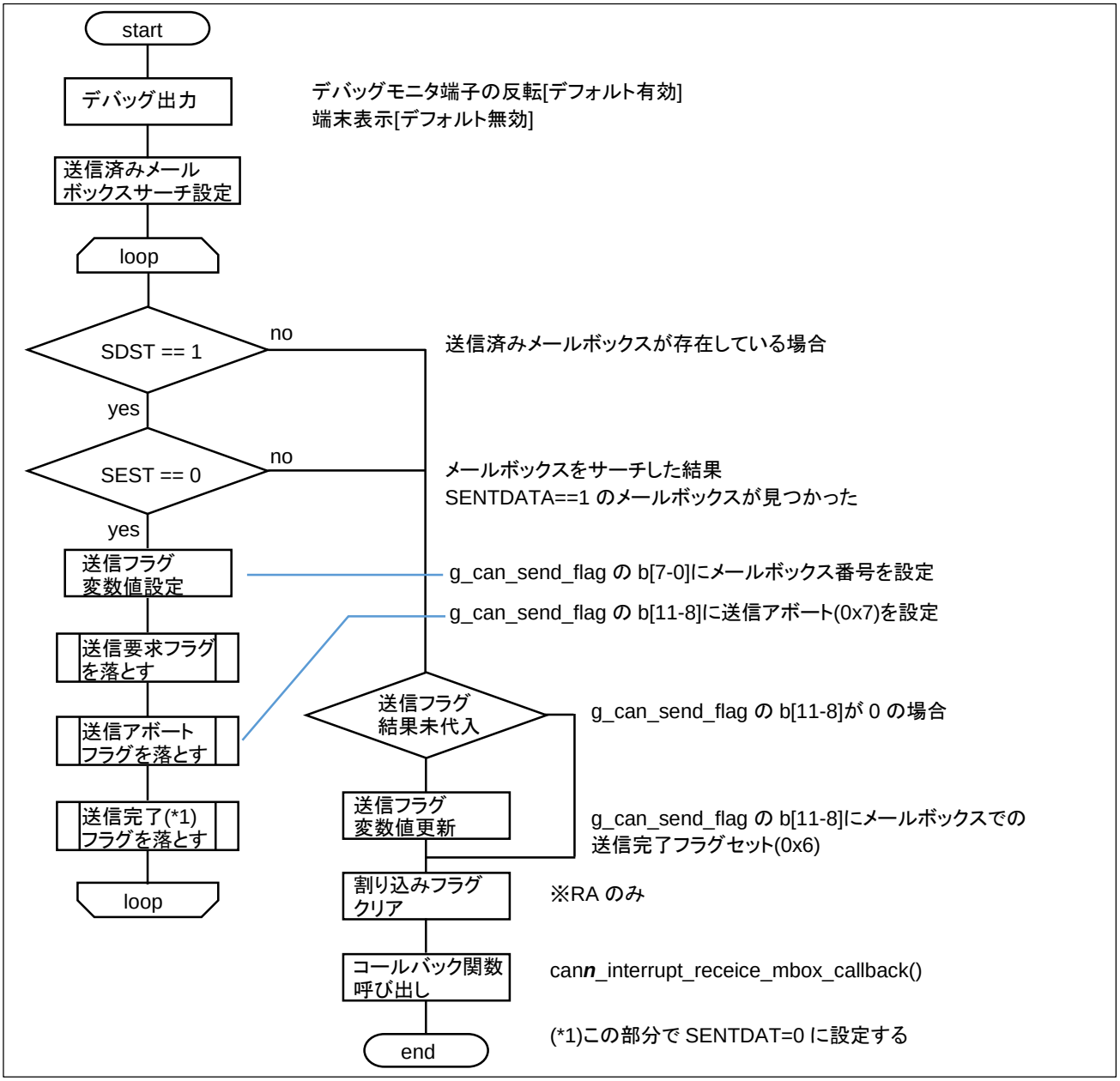
・送信フラグ変数のセット

・割り込みフラグクリア

・コールバック関数の呼び出し

→can***n***_interrupt_tx_mbox_callback()

－フローチャート－



5. サンプルプログラムの説明(SAMPLE1)

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。

・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）
・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）
・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(*2)
・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(*2)
・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(*2)
・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）(*2)
・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(*2)

－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)これらの動作は、CAN のモジュール種別に拘わらず共通ですので、ソフトウェア編マニュアルを参照してください

5.2. 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	送信時に指定	送信時に指定	送信
1	送信時に指定	送信時に指定	送信
2	送信時に指定	送信時に指定	送信
3	送信時に指定	送信時に指定	送信
4	標準(SID)	0x000	受信
5	標準(SID)	0x001	受信
6	標準(SID)	0x002	受信
7	標準(SID)	0x003	受信
8	拡張(EID)	0x0000000	受信
9	拡張(EID)	0x0000001	受信
10	拡張(EID)	0x0000002	受信
11	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、0~3 を送信用、4~11 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 8~11 は未使用となります。

5.3. 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 0~3 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
can0_mbox_set_send(0);
can0_mbox_set_send(1);
…
can0_mbox_set_send(3);
```

メールボックス 4~11 を受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, ID フォーマット, データフレーム/リモートフレーム, ID

can0_mbox_set_receive(4, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
…
can0_mbox_set_receive(7, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000003);
can0_mbox_set_receive(8, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
…
can0_mbox_set_receive(11, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000003);
```

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*_mbox_set_send()は、メールボックスを送信用に設定する関数で、can*n*_moxb_set_receive()は、受信用に設定する関数です。

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

(3)CAN モジュールを動作状態に移行

```
can0_operate();
can1_operate();   //CAN-ch1 使用時
can2_operate();   //CAN-ch2 使用時
```

(4)データの受信

```
for(i=4; i<=11; i++)  //メールボックス[4]-[11]が受信設定
{
   //引数:メールボックス番号   CAN メッセージ構造体
   r_ret = can0_mbox_receive(i, &r_msg);
}
```

r_ret が-1(CAN_RET_NODATA)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
CAN メッセージ構造体:
- r_msg.id : 受信した ID 値
- r_msg.rtr : 受信した RTR 値(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)
- r_msg.ide : 受信した IDE 値(拡張 ID, 標準 ID 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)
- r_msg.dlc : 受信した DLC 値(データバイト数)
- r_msg.data[ ] : 受信データ
- r_msg.ts : タイムスタンプ値(受信側で付与)

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

(5)データの送信

```
can_message s_msg;

s_msg.id = 0x0000001;
s_msg.rtr = CAN_DATA_FRAME;
s_msg.ide = CAN_ID_FORMAT_EID;
s_msg.data[0] = 0x01; //送信データ
s_msg.data[1] = 0x23;
…
s_msg.dlc = 2;

//引数:メールボックス番号   CAN メッセージ構造体
s_ret = can0_mbox_send(1, &s_msg);
```

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
CAN メッセージ構造体:
　送信する ID 値, RTR 値, IDE 値, データ, DLC 値をセット

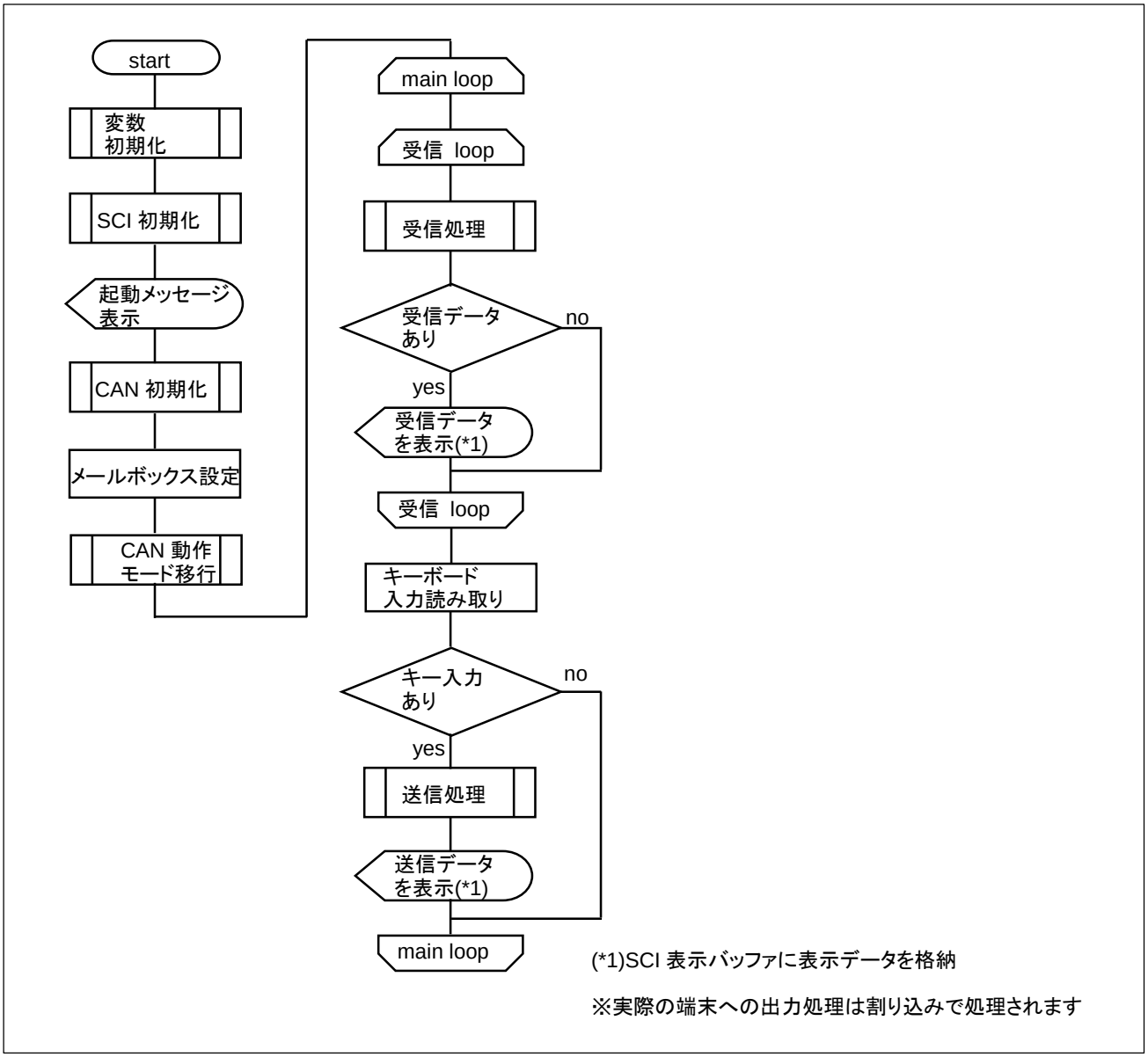
メールボックス 1 から、拡張 ID フォーマットで、データ 0x01 0x02 を 2 バイト送信する場合上記の様な関数呼び出しとなります。

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。)

5.4. SAMPLE1 フローチャート

メイン関数　main_s1()



6. サンプルプログラムの説明(SAMPLE2)

6.1. プログラム仕様

・データフレームの送信（FIFO を使用）
・データフレームの受信（拡張 ID のデータは FIFO を使用、標準 ID のデータはメールボックスを使用）

を行うサンプルプログラムとします。

SAMPLE1 との相違点として、FIFO を使用する事と、割り込みによる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。

6.2. 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
4	標準(SID)	0x000	受信
5	標準(SID)	0x001	受信
6	標準(SID)	0x002	受信
7	標準(SID)	0x003	受信

送信に FIFO、拡張 ID フォーマットの受信に FIFO を使用しますので、標準 ID の受信のみ、メールボックス設定を行っています。（標準 ID の場合は上記の ID のみ、拡張 ID の場合は任意の ID のデータを受信します。）

6.3. 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-ch0 の初期化、CAN-ch1, CAN-ch2 使用時は必要に応じて can1_init(); can2_init();
can_init();　　※RX のみ必要

(2)メールボックスの設定

FIFO で送信、受信を行う場合はメールボックスの設定は不要です。SAMPLE2 では、標準 ID のデータをメールボックスで受信するので、その設定をここで行います。

can0_mbox_set_receive(4, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x0000000)**;**
…
can0_mbox_set_receive(7, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x0000003)**;**

メールボックス 4~7 を受信設定とします。

※SAMPLE2 では FIFO を使う設定なので、メールボックスとしては 0~23 が使用可能です

(3)CAN モジュールを動作状態に移行

can0_operate();

(4)データの受信

SAMPLE1 では、この部分でデータの受信処理がありましたが、SAMPLE2 ではありません。
（受信処理は割り込みに追い出す形です。）

(4)データの送信

//引数：CAN メッセージ構造体
s_ret = **can0_mboxfifo_send**(&s_msg);

SAMPLE1 に対し、FIFO で送信する関数に変わります。

メイン関数(main_s2.c 内の main()）で行っているのは上記となります。

6.4. 割り込み処理

割り込み関数内での処理は、4 章に記載した処理が実行されます。

main_s2.c 内には、割り込みコールバック関数が記載されており、割り込み関数の最後でコールバック関数が呼ばれます。ここでは、コールバック関数の処理に関して記載します。

(1)エラー割り込み

can***n***_interrupt_ch_error_callback();

n=0,1,2 のどの ch で発生したエラーでも、同じ子関数
can_interrupt_ch_error_callback(CAN_CH0);　//CAN-ch0 の割り込みの場合(CAN_CH0=0)
を、ch 番号を引数にして呼ぶ事としています。（エラー処理の実体は、1 つの関数にまとめています）

エラー割り込みコールバック関数では、エラー割り込み関数内で保存したエラーレジスタの画面表示を行います。

(2)受信割り込み

can***n***_interrupt_receive_mboxfifo_callback();　//FIFO で受信
can***n***_interrupt_receive_mbox_callback();　//メールボックスで受信

FIFO で受信（拡張 ID のデータ）したケースとメールボックスで受信（標準IDのデータ）したケースで、呼び出されるコールバック関数は異なりますが、どちらの関数でも

can_interrupt_rx_callback(CAN_CH0);　//CAN-ch0 の割り込みの場合

同じ子関数を呼び出す事としています。can_interrupt_rx_callback()内では、受信データの表示を行っています。

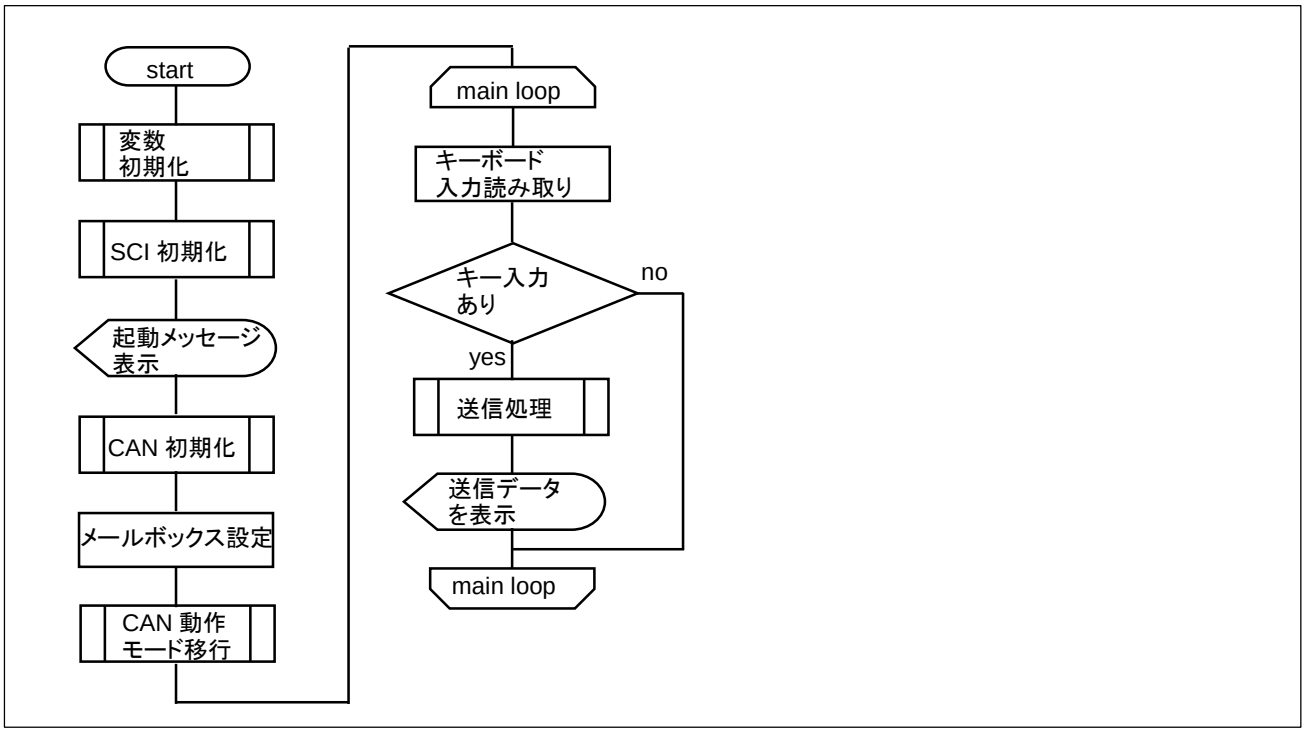
(3)送信割り込み

can***n***_interrupt_tx_mboxfifo_callback();　//FIFO で送信
→can_interrupt_tx_callback(CAN_CH0); // CAN-ch0 の割り込みの場合

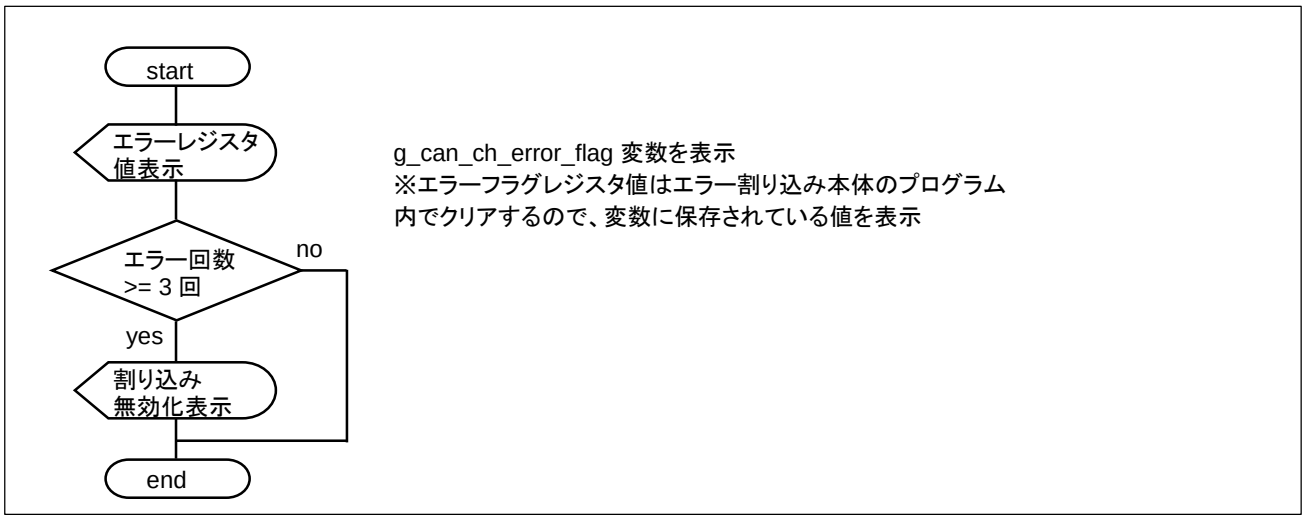
受信同様、コールバック関数内で子関数 can_interrupt_tx_callback()を呼び出す処理としており、送信完了のメッセージ
CAN0 send finished.(FIFO)
を表示する処理を行います。

6.5. SAMPLE2 フローチャート

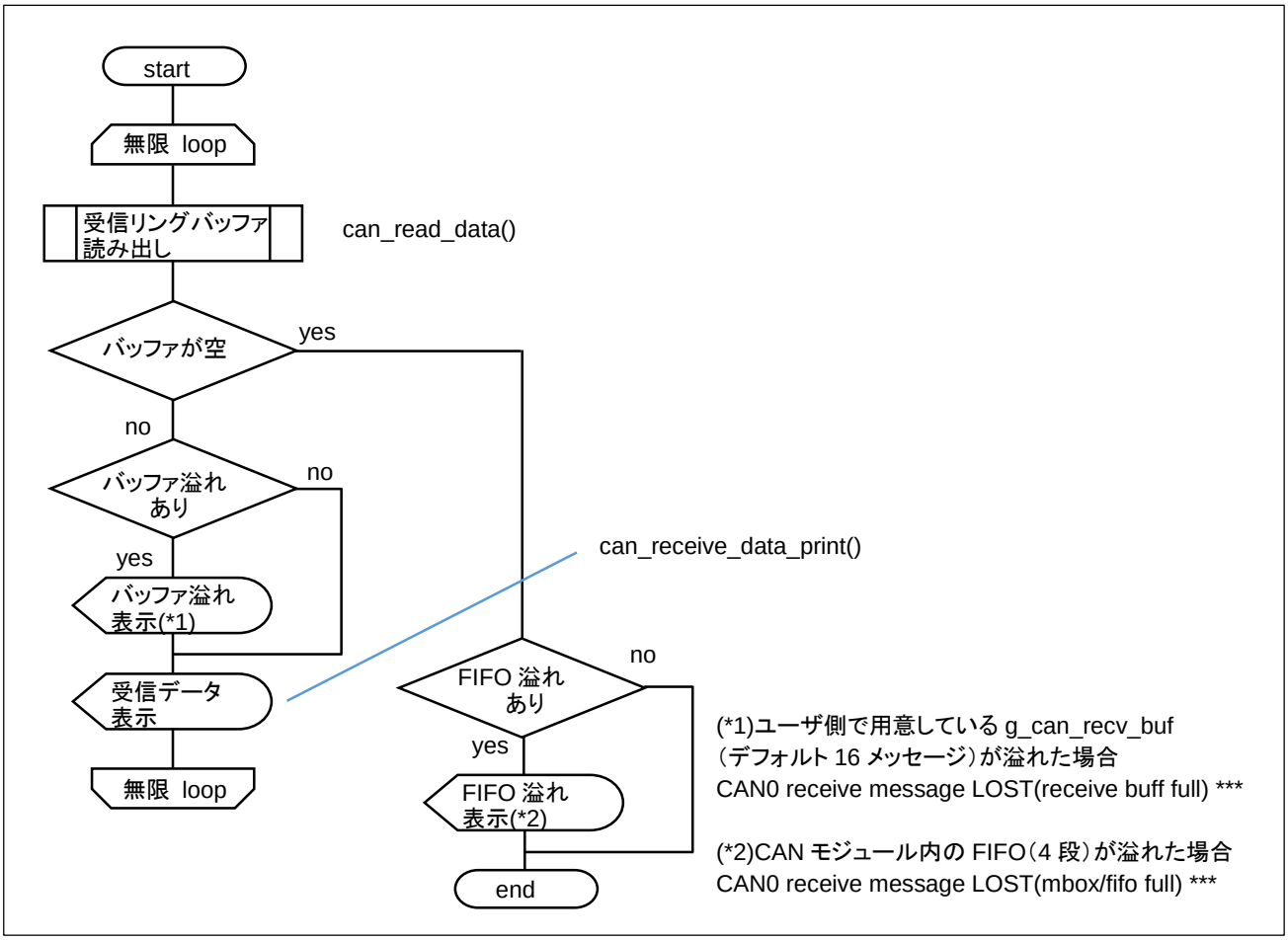
メイン関数　main_s2()



エラー割り込みコールバック関数　can_interrupt_ch_error_callback()



受信割り込みコールバック関数から呼ばれる関数　can_interrupt_rx_callback()



受信コールバック関数は、

can***n***_interrupt_receive_mboxfifo_callback()　FIFO 受信コールバック関数　(***n***=0,1,2: CAN-ch)

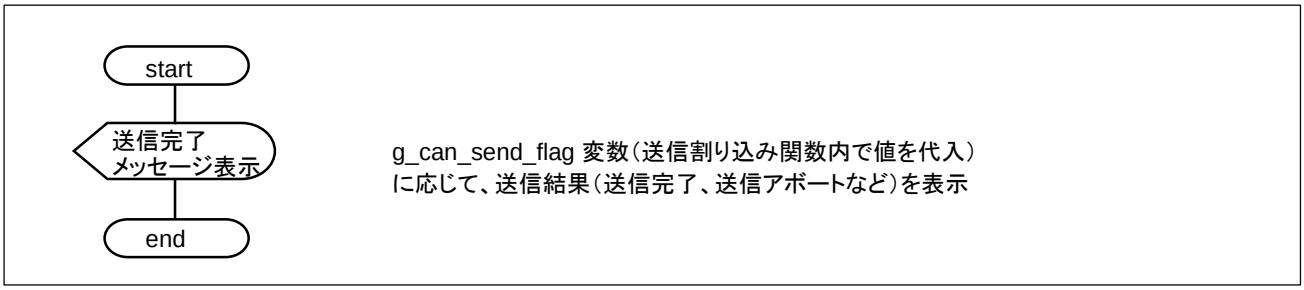
can***n***_interrupt_receive_mbox_callback()　メールボックス受信コールバック関数

から

can_interrupt_rx_callback()

を呼ぶ形にしています。

送信割り込みコールバック関数から呼ばれる関数　can_interrupt_tx_callback()



送信コールバック関数も同様に、

can***n***_interrupt_tx_mboxfifo_callback()　FIFO 送信コールバック関数　(***n***=0,1,2: CAN-ch)
can***n***_interrupt_tx_mbox_callback()　メールボックス送信コールバック関数
　から

最終的に can_interrupt_tx_callback() を呼ぶ形にしています。

7. サンプルプログラムの説明(SAMPLE3)

7.1. プログラム仕様

・データフレームの送信（FIFO を使用）
・データフレームの受信（拡張 ID のデータは FIFO を使用、標準 ID のデータはメールボックスを使用）
・リモートフレームの送信（FIFO を使用）
・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。

・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）の応答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 を変更する等の方法があります。

7.2. 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
4	標準(SID)	データフレーム	0x000	受信
5	標準(SID)	データフレーム	0x001	受信
6	標準(SID)	データフレーム	0x002	受信
7	標準(SID)	データフレーム	0x002	受信
8	標準(SID)	リモートフレーム	0x000	受信
9	標準(SID)	リモートフレーム	0x001	受信
10	標準(SID)	リモートフレーム	0x002	受信
11	標準(SID)	リモートフレーム	0x003	受信

メールボックス 4~8 を受信用に割り当て。

7.3. 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_init();　　※RX のみ必要

(2)メールボックスの設定

can0_mbox_set_receive(4, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x0000000)**; //SAMPLE2 と同じ**
…
can0_mbox_set_receive(7, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x0000003)**; //SAMPLE2 と同じ**
can0_mbox_set_receive(8, CAN_ID_FORMAT_SID, CAN_REMOTE_FRAME, 0x0000000)**;**
…
can0_mbox_set_receive(11, CAN_ID_FORMAT_SID, CAN_ REMOTE _FRAME, 0x0000003)**;**

メールボックス 8~11 を受信設定とします。

(3)CAN モジュールを動作状態に移行

can0_operate();

(4)データの送信

・データフレームの送信（0~3 コマンド）
s_msg.rtr = CAN_DATA_FRAME;
s_ret = **can0_mboxfifo_send**(&s_msg);

・リモートフレームの送信（q~r コマンド）
s_msg.rtr = CAN_REMOTE_FRAME;
s_ret = **can0_mboxfifo_send**(&s_msg);

データフレームを送る関数と、リモートフレームを送る関数は同一です。引数として指定する can_message 構造体のメンバの値
s_msg.rtr = CAN_DATA_FRAME または CAN_REMOTE_FRAME
によって、送信されるメッセージの形式が変わります。
リモートフレームの場合、
s_msg.data[]
値は使用されません。
リモートフレームの場合、相手に何バイトのデータを送って欲しいかは、送信元が要求するので
s_msg.dlc
の値は指定する必要があります。

7.4. 割り込み処理

SAMPLE2 から変えているのは、受信割り込みとなります。

(1)受信割り込み

can***n***_interrupt_receive_mboxfifo_callback(); //FIFO で受信
can***n***_interrupt_receive_mbox_callback(); //メールボックスで受信

上記コールバック関数から

can_interrupt_rx_callback(CAN_CH0); //CAN-ch0 の割り込みの場合

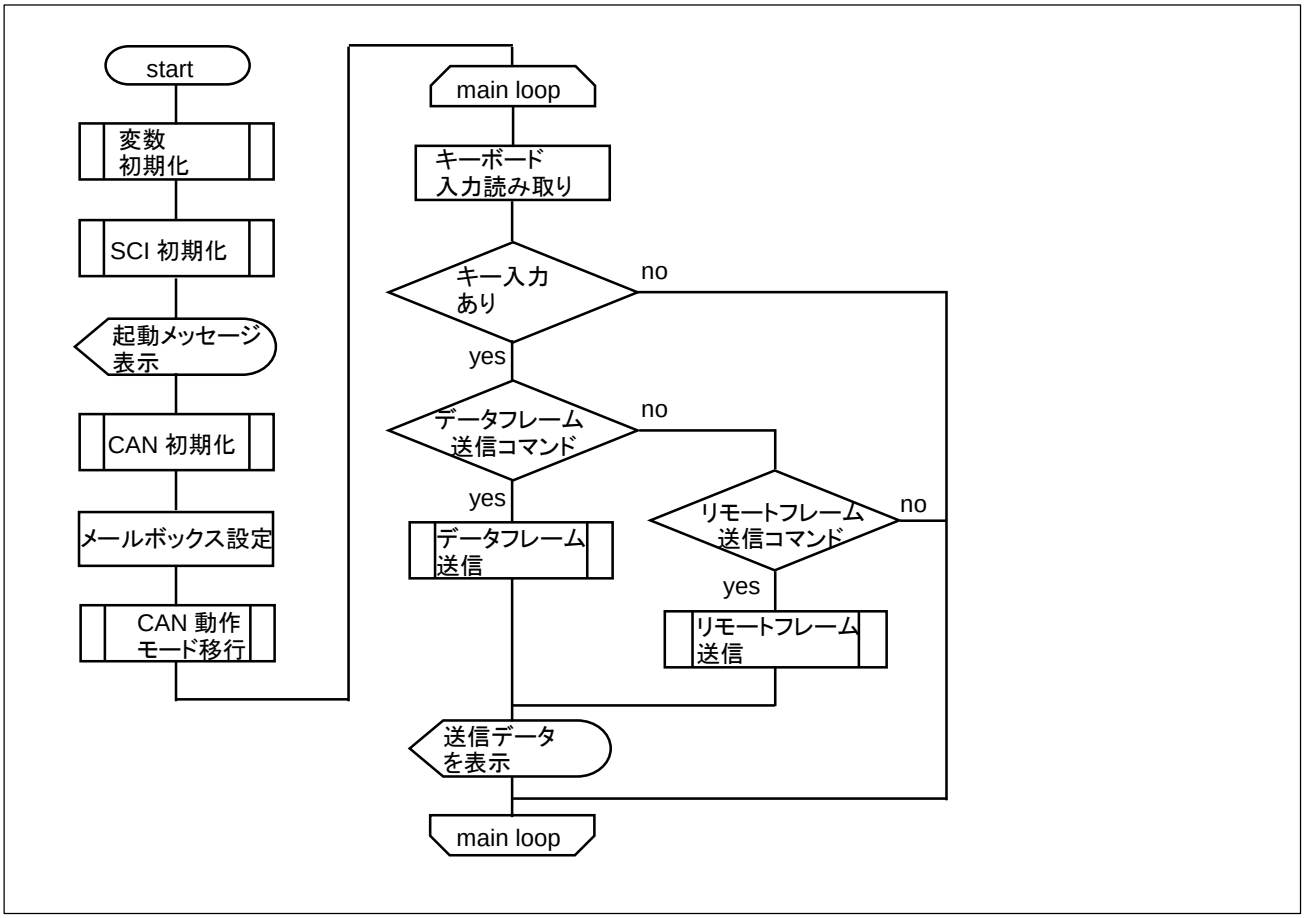
が呼ばれるのは同一。この関数内で、

・受信データの表示（SAMPLE2 と同じ）
・（受信データがリモートフレームの場合）応答データの送信

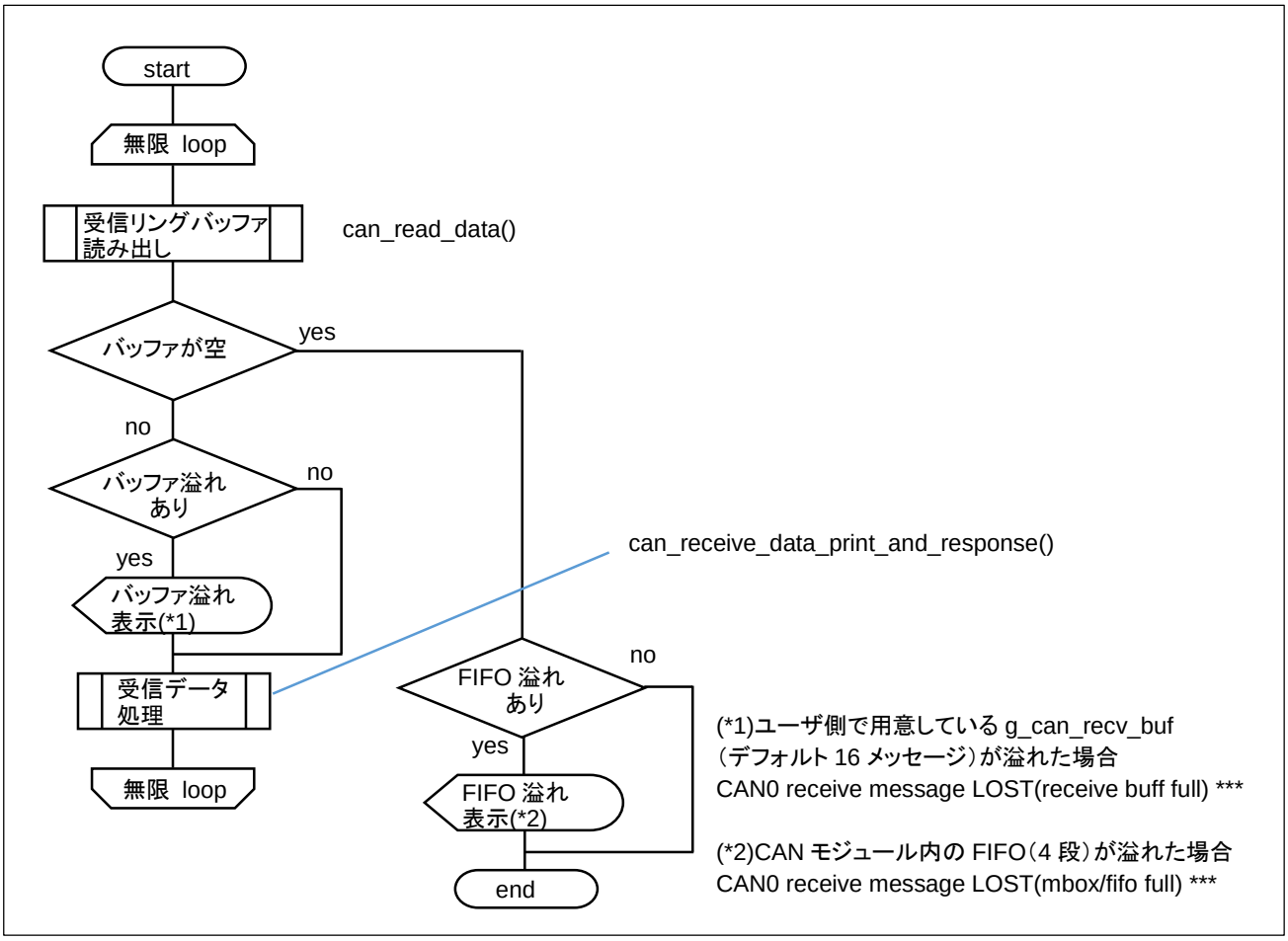
を行っています。

7.5. SAMPLE3 フローチャート

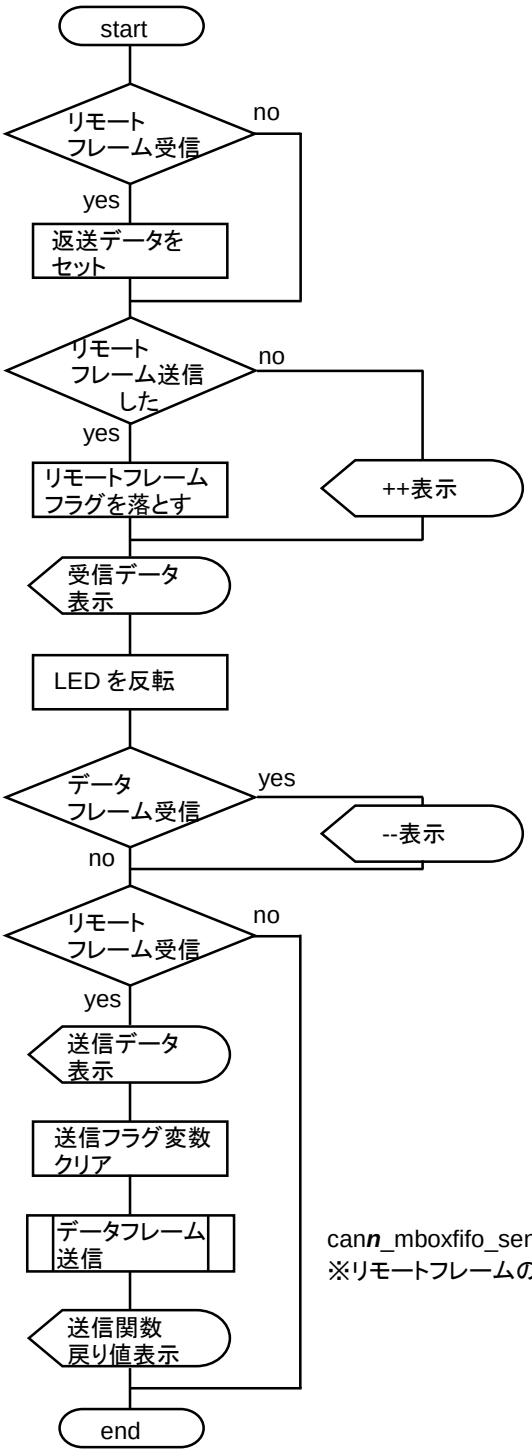
メイン関数　main_s3()



受信割り込みコールバック関数 can_interrupt_rx_callback() ※SAMPLE2 から変更あり



受信データ処理関数　can_receive_data_print_and_response()



一連のメッセージの開始を表す表示
（自身がリモートフレームを送信していない場合、データを受信したのが、一連の処理の開始点となる）

一連のメッセージの終了を表す表示
（通信相手がデータフレームを送ってきた場合は、その時点でメッセージの終了
通信相手が、自身のリモートフレーム要求に応答する形でデータフレームを送ってきた場合、その時点で一連の処理の終了となる）

can***n***_mboxfifo_send()
※リモートフレームの応答はデータフレームでの送信

8. サンプルプログラムで使用している関数の説明

8.1. 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:

CAN_RET_SUCCESS(0): 正常終了
CAN_RET_VALUE_ERROR(-6): 端子設定または速度設定に誤りがある

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_init

概要:グループ割り込み初期化関数(RX のみ)

宣言:

int can_init(void)　[RX のみ]

説明:

・エラー割り込みの有効化

・割り込み関数の登録

を行います

引数:

なし

戻り値:

0: 正常終了

補足:

RX の場合は、エラー割り込みがグループ BE0 にグルーピングされており、グループ割り込みの有効化と、グループ割り込みの割り込み要因ごと(エラー発生 ch ごと)の割り込み関数の登録を、本関数にて行います。

RX でエラー割り込みを使用する場合に、本関数を実行してください。

cann_operate (n=0~2)

概要:動作モード移行関数

宣言:

int can0_operate(void) [ch0 向け]

int can1_operate (void) [ch1 向け]

int can2_operate (void) [ch2 向け] [RX のみ]

説明:

・動作モードへの移行

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

補足:

cann_init() の後で実行してください。

can*n*_mbox_set_send　　(n=0~2)

概要:メールボックス設定関数

宣言:

int can0_mbox_set_send(unsigned char mb)　[ch0 向け]

int can1_mbox_set_send(unsigned char mb)　[ch1 向け]

int can2_mbox_set_send(unsigned char mb)　[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

FIFO を使用しない場合は、0~31、FIFO 使用時は 0-23 の数値が有効です。

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

CAN_RET_TIMEOUT(-7): 送信、受信アボート処理失敗

can*n*_mbox_set_receice　　(n=0~2)

概要:メールボックス設定関数

宣言:

int can0_mbox_set_receive(unsigned char mb, unsigned char ide, unsigned char rtr, unsigned long id) [ch0 向け]

int can1_mbox_set_receive (unsigned char mb, unsigned char ide, unsigned char rtr, unsigned long id) [ch1 向け]

int can2_mbox_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受信ルール設定

を行います

引数:

mb: メールボックス番号(0~31)（FIFO 使用時は 0~23）

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:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

CAN_RET_TIMEOUT(-7): 送信、受信アボート処理失敗

cann_mbox_set_receice_mask　(n=0~2)

概要:受信ルールマスク設定関数

宣言:

int can0_mbox_set_receive_mask(unsigned char mb, unsigned long id_mask)　[ch0 向け]

int can1_mbox_set_receive_mask(unsigned char mb, unsigned long id_mask)　[ch1 向け]

int can2_mbox_set_receive_mask(unsigned char mb, unsigned long id_mask)　[ch2 向け] [RX のみ]

説明:

・受信用メールボックスの ID のマスク条件設定（FIFO 使用時は 0~23）

を行います

引数:

mb: メールボックス番号(0~31)（FIFO 使用時は 0~23）

id_mask: ID のマスク値を指定

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

CAN_RET_MODE_ERROR(-5) 本関数が実行可能なモードではない

補足:

id_mask = 0x1FFFFF00 を引数に与えて実行した場合は、

ID の bit7-0 が任意の ID のデータを受信する様になります

具体例として、cann_mbox_set_receive() で、id = 0x00000030 を指定して、本コマンドで

id_mask = 0x1FFFFFF8 を指定した場合、

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
id	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
id_mask	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
受信 ID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	x	x	x

x=0 or 1

受信する ID としては、0x00000030~ 0x00000037 の範囲となります。
id_mask = 0x00000000 とした場合は、全ての ID が受信対象となります。

ID のマスクは、32 個のメールボックスに対して、8 個のマスクを定義可能になっています。メールボックス 0~3, 4~7, 8~11, ..., 28~31 の 8 通りです。(メールボックス 0 と 1 で別々のマスクを定義することが出来ません。)
本関数で、mb = 0 を指定した場合した場合、メールボックス 0~3 のマスク設定が変更されます。mb = 1, mb = 2, mb = 3 を指定した場合も同様です。本関数で、mb = 4 でマスク設定後、mb = 5 でマスク設定した場合は、後で設定した mb = 5 で設定したマスク値が、メールボックス 4~7 に適用されます。
1 つのメールボックスで複数の ID のデータを受信したい場合、本関数の実行が必要です。
本関数は、can***n***_operate() の前に実行する必要があります。

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

概要:データ送信関数

宣言:

int can0_mbox_send(unsigned char mb, can_message *msg) [ch0 向け]
int can1_mbox_send(unsigned char mb, can_message *msg) [ch1 向け]
int can2_mbox_send(unsigned char mb, can_message *msg) [ch2 向け] [RX のみ]

説明:

・メールボックスを使用してデータの送信
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)(FIFO 使用時は 0~23)
msg: 送信データを設定した CAN メッセージ構造体
msg.ide:
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)
msg.rtr:
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)
msg.id: 送信する ID
msg.dlc: 送信バイト数を指定します(1~8)
msg.data[ ]: 送信するデータを指定します

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

CAN_RET_IN_USE (-3): メールボックス使用中

CAN_RET_MODE_ERROR(-5): メールボックスが受信に設定されている

CAN_RET_TIMEOUT(-7): レジスタ値設定タイムアウト

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

概要:データ送信関数

宣言:

int can0_mboxfifo_send(can_message *msg) [ch0 向け]

int can1_mboxfifo_send(can_message *msg) [ch1 向け]

int can2_mboxfifo_send(can_message *msg) [ch2 向け] [RX のみ]

説明:

・メールボックス FIFO を使用してデータの送信

を行います

引数:

msg: 送信データを設定した CAN メッセージ構造体

msg.ide:

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)

msg.rtr:

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)

msg.id: 送信する ID

msg.dlc: 送信バイト数を指定します(1~8)

msg.data[ ]: 送信するデータを指定します

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

CAN_RET_OVERFLOW(-4): FIFO フル

CAN_RET_MODE_ERROR(-5): FIFO が有効なモードではない

補足:

FIFO は 4 段となりますので、未送信のデータが 4 つ溜まっている状態で本関数を呼び出すと、CAN_RET_OVERFLOW となります。

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

概要:受信関数

宣言:

int can0_mbox_receive(unsigned char mb, can_message *msg) [ch0 向け]
int can1_mbox_receive(unsigned char mb, can_message *msg) [ch1 向け]
int can2_mbox_receive(unsigned char mb, can_message *msg) [ch2 向け] [RX のみ]

説明:

・メールボックスに格納されているデータの受信
を行います

引数:

mb: メールボックス(0~31)（FIFO 使用時は 0~23）
msg: 受信データを格納する CAN メッセージ構造体
msg.ide:
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)
msg.rtr:
CAN_DATA_FRAME(0) データフレーム
CAN_REMOTE_FRAME(1) リモートフレーム
msg.id: 受信した ID
msg.dlc: データバイト数(1-8)
※実際に受信したデータバイト数ではなく、CAN メッセージの DLC フィールドに含まれる値
msg.data[ ]: 受信データ
msg.ts: タイムスタンプ（受信側で付与したデータ）

戻り値:

1~8: 受信したデータのバイト数（DLC の値）
CAN_RET_NODATA(-1): 受信データなし
CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー
CAN_RET_MODE_ERROR(-5): メールボックスが受信用に設定されていない
CAN_RET_TIMEOUT(-7): メッセージ更新タイムアウト，レジスタ設定タイムアウト
CAN_RETFLAG_LOST_DATA(0x8x): (b7=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）

補足:

戻り値が 0x82 の場合は、受信バイト数は 2 で、受信操作前に上書きされたデータが存在する事を示します

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

概要:受信関数

宣言:

int can0_mboxfifo_receive(can_message *msg) [ch0 向け]

int can1_mboxfifo_receive(can_message *msg) [ch1 向け]

int can2_mboxfifo_receive(can_message *msg) [ch2 向け] [RX のみ]

説明:

・メールボックス FIFO に格納されているデータの受信

を行います

引数:

msg: 受信データを格納する CAN メッセージ構造体

msg.ide:

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)

msg.rtr:

CAN_DATA_FRAME(0) データフレーム

CAN_REMOTE_FRAME(1) リモートフレーム

msg.id: 受信した ID

msg.dlc: データバイト数(1-8)

※実際に受信したデータバイト数ではなく、CAN メッセージの DLC フィールドに含まれる値

msg.data[ ]: 受信データ

msg.ts: タイムスタンプ(受信側で付与したデータ)

戻り値:

1~8: 受信したデータのバイト数(DLC の値)

CAN_RET_NODATA(-1): 受信データなし

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

CAN_RET_MODE_ERROR(-5): FIFO が有効なモードではない

CAN_RET_TIMEOUT(-7): メッセージ更新タイムアウト, レジスタ設定タイムアウト

CAN_RETFLAG_LOST_DATA(0x8x): (b7=1)オーバライドフラグが立っている

(受信操作前に破棄されたメッセージあり)

補足:

戻り値が 0x82 の場合は、受信バイト数は 2 で、受信操作前に破棄されたデータが存在する事を示します

CAN のデータ受信は、メールボックス, FIFO のどちらを使用した場合でも、メールボックスまたは FIFO へのデータ格納は、CAN モジュールのハードウェアが行います。メールボックスや FIFO に格納されているデータの読み出しを行うのが、受信関数となります。

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

概要:送信停止関数

宣言:

int can0_abort(void) [ch0 向け]
int can1_abort(void) [ch1 向け]
int can2_abort(void) [ch2 向け] [RX のみ]

説明:

・送信中、送信待機データの破棄

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了
CAN_RET_TIMEOUT(-7): レジスタ値変更タイムアウトし

補足:

データ送信時、ACK を返す通信相手が居ない場合など、データ送信が完了しない場合は送信待機状態となり再度データ送信を試みます。データ送信を何度も繰り返す動作となります。その様な際、本関数で送信を停止する事が可能です。

8.2. プログラムで使用しているグローバル変数

can_message g_can_recv_buf[CAN_CH][CAN_RECV_BUF_SIZE]

受信データ用リングバッファ
受信割り込み関数では、受信データを本変数にコピーする処理が行われます。
CAN_CH : CAN の ch 数（board.h で定義）
CAN_RECV_BUF_SIZE(=16): リングバッファの段数（can_operation.h で定義）

unsigned short g_can_recv_buf_index1[CAN_CH]
unsigned short g_can_recv_buf_index2[CAN_CH]

受信データ用リングバッファの書き込み、読み出しインデックス変数
※受信データ用リングバッファ格納の際、index1 がインクリメント、
受信データ用リングバッファ読み出し(can_read_data()実行）時 inde2 がインクリメントされますので、ユーザ側で意識する必要がない変数です。

unsigned short g_can_recv_buf_override[CAN_CH]

受信データ用リングバッファが溢れた際に 1 にセットされます。

unsigned short g_can_message_lost_flag[CAN_CH]

CAN の受信データが失われた際（FIFO フルやメールボックス上書き）に 1 にセットされます。

unsigned short g_can_speed = CAN_SPEED

CAN の通信速度を保持する変数です。
CAN_SPEED: can_operation.h 内で、1000(1000kbps)の値が定義されています。

volatile int g_can_send_flag[CAN_CH]

送信フラグ変数。送信割り込み関数内で、送信時に使用したメールボックス番号や送信結果フラグがセットされます。

unsigned long g_can_ch_error_flag[CAN_CH]

エラーフラグ保持変数。エラー割り込み関数内で、エラーフラグレジスタ値を本変数に保存します。

unsigned long g_can_ch_error_flag_history[CAN_CH]

エラー履歴変数。累積のエラーを本変数に保存する。H コマンドで本変数を参照します。
※エラー履歴をクリアしたい場合は、本変数に 0 を代入してください

unsigned long g_can_ch_error_counter[CAN_CH]

エラーカウント変数。デフォルトでは、本変数値が 3 を超えると、エラー割り込みを無効化します。C コマンドで本変数値をクリアすることができます。

unsigned short g_can_remote_frame_request[CAN_CH]

SAMPLE3 で、CAN の一連のメッセージを
+++
一連のメッセージのやり取り

で囲むための変数です。リモートフレーム送信時に 1 にセットされ、データ受信時にクリアされます。

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
REV.1.10.0.0	2024.10.15		バージョンアップに伴い大幅に刷新

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
 2024 年 10 月 15 日改訂 REV.1.10.0.0 (241015)