CAN スタータキット RX/RA
CAN スタータキット SmartRX
RSCAN 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、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、「RSCAN モジュール編」のマニュアルです。RSCAN 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

・RSCAN モジュール向け

フォルダ	ファイル	説明
source¥RSCAN_module¥rscan		RSCAN モジュール向け共通フォルダ
	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
	rscan.c	ch に依存しない処理関数，グローバル変数定義
	rscan.h	上記ヘッダ
	rscan_intr.c	グローバル割り込み関数
	rscan_intr.h	グローバル割り込み関数ヘッダ
source¥RSCAN_module¥rscan_ch0		ch0 向けフォルダ
	rscan_ch0.c	ch0 向け関数
	rscan_ch0.h	ch0 関数ヘッダ
	rscan_ch0_intr.c	ch0 割り込み関数
	rscan_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥RSCAN_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

※RSCAN は CAN の ch が 2ch 以上となるマイコンは存在しませんが、ch 依存の処理と、全体の処理を分けています

・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（SAMPLE1~4 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

・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) [プロジェクト名]¥src¥usr_src¥settings

2. 初期化

RSCAN 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)RSCAN モジュール初期化

```
can_reset();
can_init();
can0_init();
```

RSCAN モジュールのストップ解除を行い、グローバルリセットモードへの遷移、端子設定等を行います。

(2)受信ルール・受信バッファ数設定

```
can_receive_buf_conf();
```

受信ルール・受信バッファ数の設定を行います。

(3)受信ルール設定

```
can0_receive_rule_set(0, CAN_RULE_RXBUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME,
0x00000000);
```

CAN-ch0 の受信ルール 0 番に
・受信先は受信バッファ(CAN_RULE_RXBUF)
・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。（※RSCAN では、CAN-ch0 しか扱いませんので、使用する関数名は、常に can0_receive_rule_set()です）

(4)動作モードへの移行

```
can_operate();
can0_operate();
```

RSCAN モジュール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。

3. データの送受信

データの受信は、「受信バッファ」「受信 FIFO」「送受信 FIFO」と 3 種類の方法があります。データの送信は、「送信バッファ」「送受信 FIFO」と 2 種類の方法があります。

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

送受信方法は、can_operation.h（SAMPLE1~4 の動作は、can_operation2.h）内で選択可能です。

・受信方法（いずれか 1 つを有効化）

```
#define CAN_RX_METHOD      CAN_RX_RXBUF      //受信バッファを使用(*1)
#define CAN_RX_METHOD      CAN_RX_RXFIFO     //受信 FIFO を使用(*2)
#define CAN_RX_METHOD      CAN_RX_SRFIFO     //送受信 FIFO を使用
```

・送信方法（いずれか 1 つを有効化）

```
#define CAN_TX_METHOD      CAN_TX_TXBUF      //送信バッファを使用(*1)
#define CAN_TX_METHOD      CAN_TX_SRFIFO     //送受信 FIFO を使用(*2)
```

(*1)SAMPLE1 のデフォルト

(*2)SAMPLE2~3 のデフォルト

※RSCAN モジュールの機能としては、複数の受信方式、複数の送信方式を同時に使用可能ですが本サンプルプログラムでは、受信と送信でそれぞれ 1 種類を選択するようにしています。

	RSCAN モジュール
受信バッファ	最大 16
受信 FIFO	RXFIFO(0)~(1) [2 本]　最大 16 段
送受信 FIFO	SRFIFO(0) [1 本]　最大 16 段

受信バッファ、受信 FIFO、送受信 FIFO でトータル 16 のメッセージバッファを振り分ける事となります。

送受信 FIFO は 1 本となりますので、「受信」または「送信」の一方のみで使用可能です。

本サンプルプログラムでは、FIFO の段数は 8 に設定しています。

・受信バッファと FIFO 段数の設定

受信方式	送信方式	消費メッセージサイズ				
		RXFIFO(0)	RXFIFO(1)	SRFIFO(0)	受信バッファ	合計
受信 FIFO	送受信 FIFO	8	0	8	0	16
	送信バッファ	8	0	0	0	8
送受信 FIFO	送信バッファ	0	0	8	0	8
受信バッファ	送受信 FIFO	0	0	8	8	16
	送信バッファ	0	0	0	16	16

本サンプルプログラムでは、どの受信方式と送信方式を選んだ場合でも、メッセージサイズが 16 を超えない様に設定しています。例えば、RXFIFO(0), RXFIFO(1), SRFIFO(0)の 3 種類の FIFO を使用するなどの場合は、FIFO 段数を 4 段に設定するなど、トータルで 16 を超えない様に設定してください。

（※メッセージ数の設定をオーバした場合、受信データで受信ルールが上書きされる等の、意図しないデータ破壊が起きます。メッセージ数の設定オーバに関しては、マイコン側にチェックする手段がないため、プログラムを作成する人が計算する必要があります。）

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。

・CAN

```
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)送信バッファを使用した送信

```
ret = can0_txbuf_send (0, &msg);
```

0 は、送信バッファ番号です。

RSCAN モジュールは CAN-ch0 のみとなりますが、関数名には can0~と 0 が付く形となります。（他の複数 ch をサポートしている、モジュールとの互換性を持たせるため、関数名は can0~としています。）

(2)送受信 FIFO を使用した送信

```
ret = can0_srfifo_send(0, &msg);
```

0 は、FIFO 番号です。送受信 FIFO を使用した場合、複数のメッセージを FIFO に溜めておく事が可能です。
RSCAN モジュールの送受信 FIFO は 1 本のため、常に第 1 引数は 0 で使用してください。（複数本の FIFO を持つマイコンとの関数インタフェースを揃えるため、引数に FIFO 番号を持たせる形としています。）

(1)~(2)は、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です。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. 受信ルール設定

CAN のメッセージを受信するためには、受信ルールの設定が必要です。

```
can0_receive_rule_set(0, CAN_RULE_RXBUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME,
0x00000000);
```

0 番の受信ルールとして、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、RXBUF(CAN_RULE_RXBUF)にデータを格納する。

受信ルールで指定するのは、
・ルール番号
・受信先
・標準/拡張 ID 区分(IDE)
・データフレーム/リモートフレーム区分(RTR)
・ID
です。

```
can0_receive_rule_set(6, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME,
0x00000002);
```

6 番の受信ルールに、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 0x00000002 のデータを受信した際に、RXFIFO(0)(CAN_RULE_RXFIFO0)にデータを格納するルールを設定。

ルール番号は 0 から初めて、空きが出ない様に設定してください。0,1,3,4 番のルールを設定した場合、0,1 番のルールは有効ですが、3,4 番のルールは有効にはなりません。

	RSCAN モジュール
ルール番号	0~15
受信先	CAN_RULE_RXBUF(*1) CAN_RULE_RXFIFO0 CAN_RULE_RXFIFO1 CAN_RULE_SRFIFO0

※RSCAN モジュールの動作としては受信条件にマッチしたデータを複数の受信先（箇所）にコピーする事が可能ですが、本サンプルプログラムではデータの格納先は 1 箇所としています

ルール番号は、0~15 の最大 16 ルールの設定が可能です。

(*1)受信先として受信バッファを指定した場合、受信バッファの番号は以下が使用されます

使用可能な受信バッファ数は、送信に送受信 FIFO を使用した場合は 8、使用しない場合は 16 となります。

ルール番号	使用される受信バッファ番号 （受信バッファ数：条件により 8,16）
0-7	0-7
8-15	8-15（受信バッファが 16 個使用可能な場合） 0-7（受信バッファが 8 個の場合）

受信バッファが 16 個使用できる場合は、ルール番号=受信バッファ番号となります。受信バッファが 8 個使用できる場合は、ルール番号 8 の場合は、受信バッファ 0 を使用します。

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)受信バッファを使用した受信

```
ret = can0_rxbuf_receive(0, &msg);
```

0 番の受信バッファのデータを取り出す。

※can0_receive_rule_set()で、予め 0 番の受信バッファへの受信条件の設定が必要です。

ret が、-1(=CAN_RET_NODATA)の場合、受信バッファ 0 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、設定した受信ルールにマッチした CAN のメッセージを受信すると、can***n***_receive_rule_set() で指定した格納先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、受信バッファへの格納までは、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、受信バッファ（要はレジスタですが）に格納されているデータを参照（can0_rxbuf_receive()ではデータコピー）する形となります。

受信バッファへの格納は、can0_receive_rule_set()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受信バッファに格納されます。（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)受信 FIFO を使用した受信

```
ret = can0_rxfifo_receive(0, &msg);
```

RXFIFO(0)に格納されているメッセージを、msg 構造体にコピーします。0 は、RXFIFO 番号です。0~1 の値が指定可能です。

can0_receive_rule_set()で、受信先を RXFIFO に設定した場合、上記関数でデータ受信が可能です。FIFO には複数のデータを格納できますので、FIFO のデータを全て読み出す際は、ret=-1(=CAN_RET_NODATA)になるまで複数回上記関数を実行してください。

－コード例－

```
int i;
can_message msg[8];
i = 0;
while(0)
{
   ret = can0_rxfifo_receive(0, &msg[i++]);
   if (ret == CAN_RET_NODATA) break;
}
```

(3)送受信 FIFO を使用した受信

```
ret = can0_srfifo_receive(0, &msg);
```

関数としては、受信 FIFO を使用する場合の関数と使用方法は変わりません。第一引数の 0 は、SRFIFO(0)のデータを読み出す場合です。0 のみ指定可能です（第一引数は常に 0 で使用してください）。
※受信 FIFO 受信関数と関数のインタフェースを同一とするため、FIFO 番号を引数に持たせています。

4. 割り込み

RSCAN モジュールの割り込みは、以下の様になっています。

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
RSCAN	COMFRXINT	52(*1), 59(*2)	Intr_RSCAN_COMFRXINT	共通 FIFO 受信
RSCAN	RXFINT	53(*1), 60(*2)	Intr_RSCAN_RXFINT	受信 FIFO
RSCAN	TXINT	54(*1), 61(*3)	Intr_RSCAN_TXINT	チャネル送信
RSCAN	CHERRINT	55(*1), 64(*2)	Intr_RSCAN_CHERRINT	チャネルエラー
RSCAN	GLERRINT	56(*1), 65(*2)	Intr_RSCAN_GLERRINT	グローバルエラー

(*1)RX231 などの割り込み番号
(*2)RX24U などの割り込み番号

基本的には、RSCAN の割り込みは、グローバル割り込みとチャネル割り込みに大別されます。グローバル割り込みは
・受信 FIFO
・グローバルエラー
となり、チャネル割り込みは
・送信
・チャネルエラー
・共通 FIFO 受信
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。

4.1. 受信 FIFO 割り込み

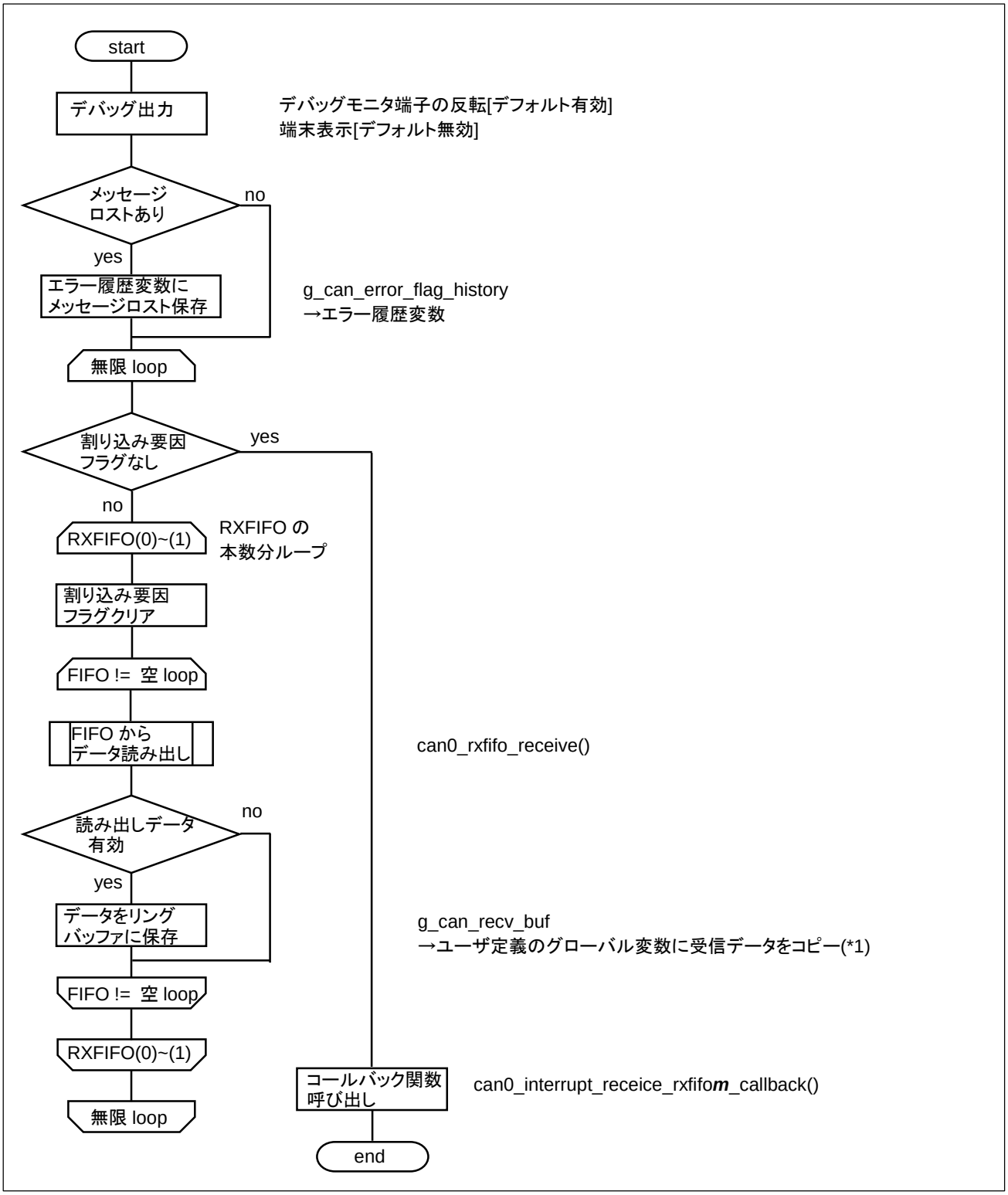
－関数名－
Intr_RSCAN_RXFINT

－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, RSCAN の場合 can_ch=0 です, 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_interrupt_receive_rxfifo***m***_callback()
（***m***: RXFIFO(m), RXFIFO 番号）

ーコールバック関数ー

割り込みコールバック関数	受信先	備考
can_interrupt_receive_rxfifo0_callback()	RXFIFO(0)で受信	
can_interrupt_receive_rxfifo1_callback()	RXFIFO(1)で受信	

ーフローチャートー



(*1)リングバッファへのコピー

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（RSCAN モジュールでは CAN-ch0 のみ）
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）

受信元	格納先
RXFIFO(0)	g_can_recv_buf[CAN_CH0]
RXFIFO(1)	g_can_recv_buf[CAN_CH0]

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);　//CAN-ch0 向けのリングバッファからの読み出し
```

とします。

4.2. グローバルエラー割り込み

－エラー割り込み対象－

・送信履歴オーバーフロー

・FIFO メッセージロスト

・DLC エラー

－関数名－

Intr_RSCAN_GLERRINT

－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

・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

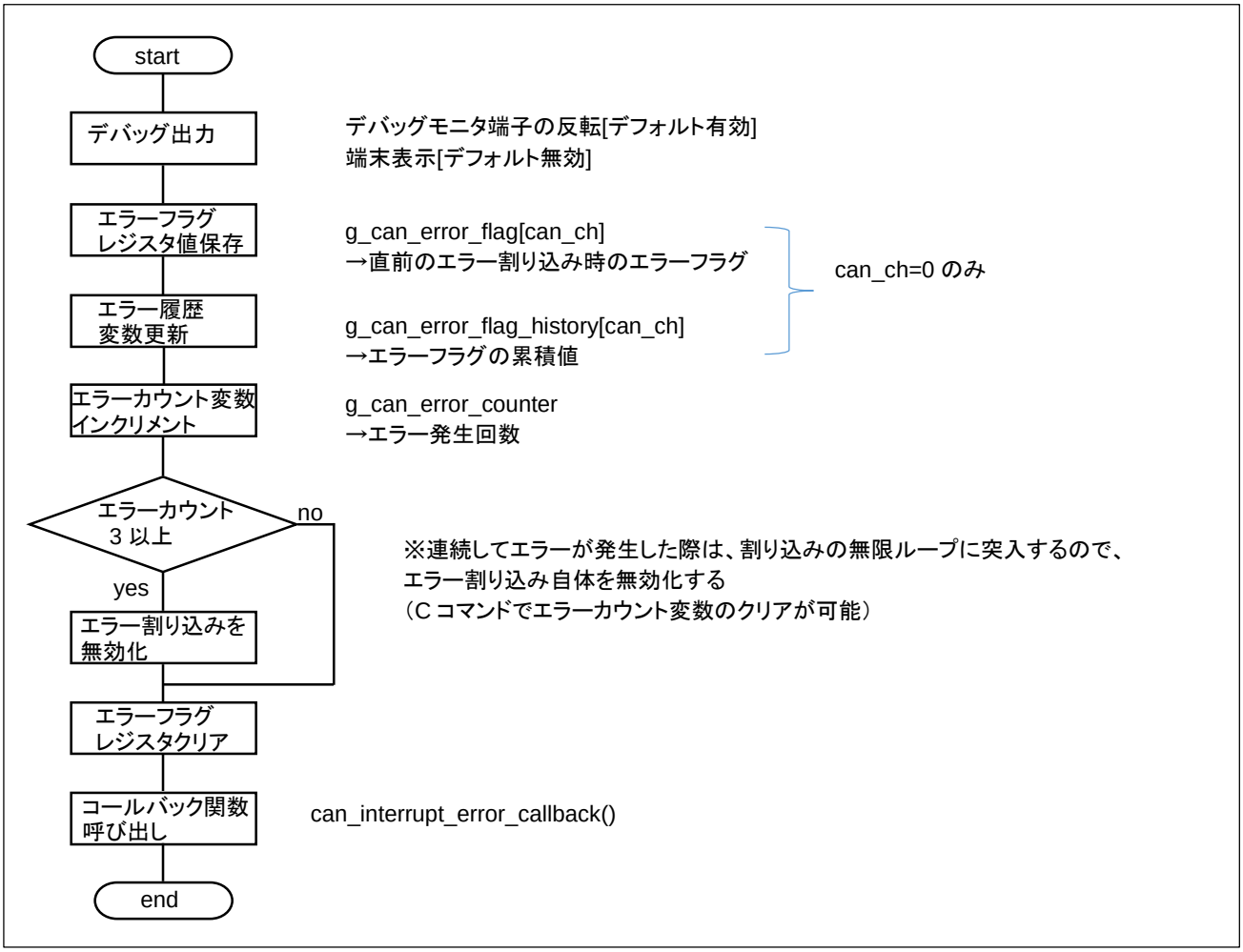
・コールバック関数の呼び出し

→can_interrupt_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。）

－コールバック関数－

割り込みコールバック関数	備考
can_interrupt_error_callback()	

ーフローチャートー


start
デバッグ出力
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]
エラーフラグ
レジスタ値保存
g_can_error_flag[can_ch]
→直前のエラー割り込み時のエラーフラグ
can_ch=0 のみ
エラー履歴
変数更新
g_can_error_flag_history[can_ch]
→エラーフラグの累積値
エラーカウント変数
インクリメント
g_can_error_counter
→エラー発生回数
エラーカウント
3 以上
no
yes
※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）
エラー割り込みを
無効化
エラーフラグ
レジスタクリア
コールバック関数
呼び出し
can_interrupt_error_callback()
end

4.3. 送信割り込み

－関数名－

Intr_RSCAN_TXINT

－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

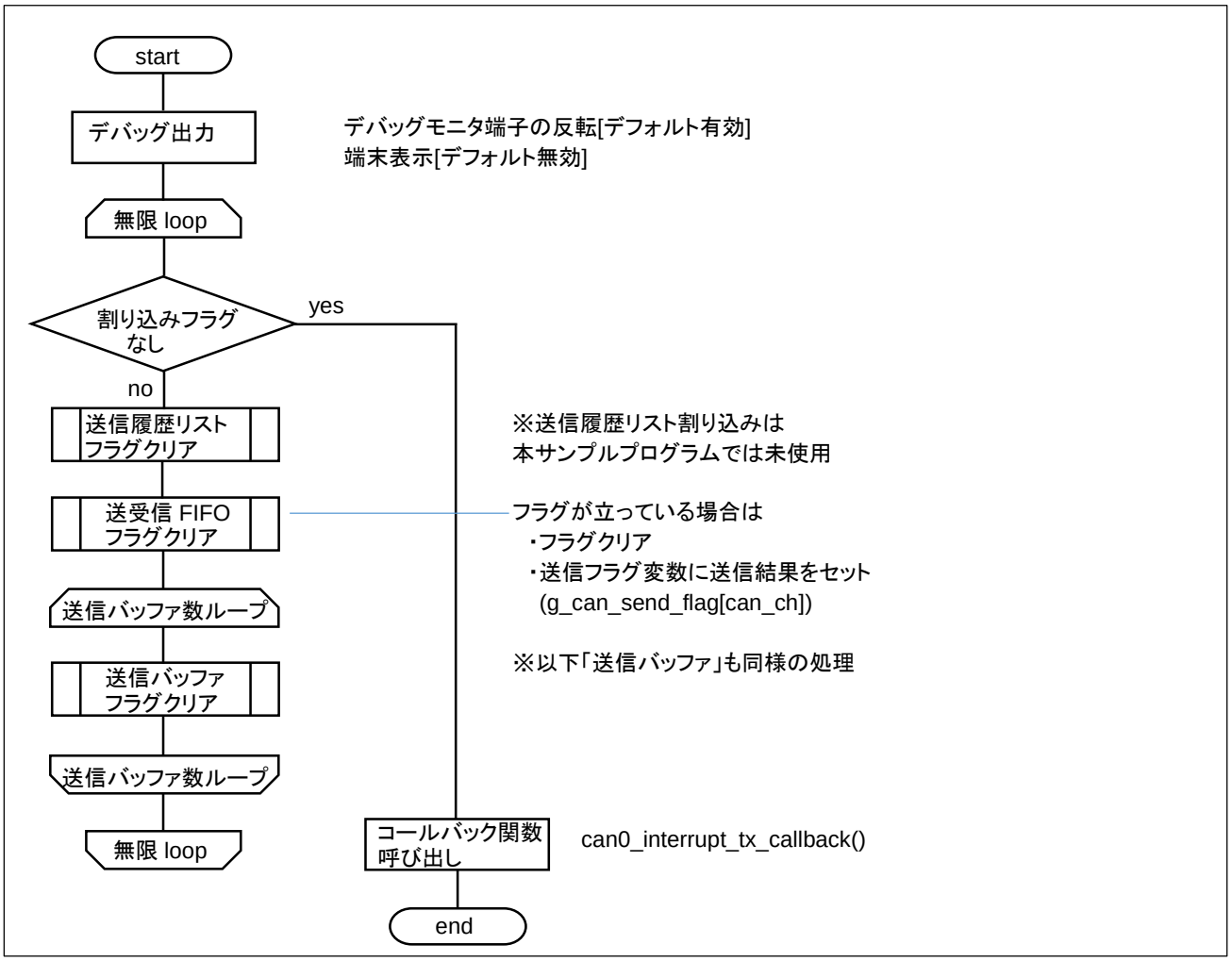
・コールバック関数の呼び出し

→can0_interrupt_tx_callback()

－コールバック関数－

割り込みコールバック関数	備考
can0_interrupt_tx_callback()	

－フローチャート－



4.4. チャネルエラー割り込み

－エラー割り込み対象－
・アービトレーションロスト
・バスロック
・オーバロードフレーム
・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

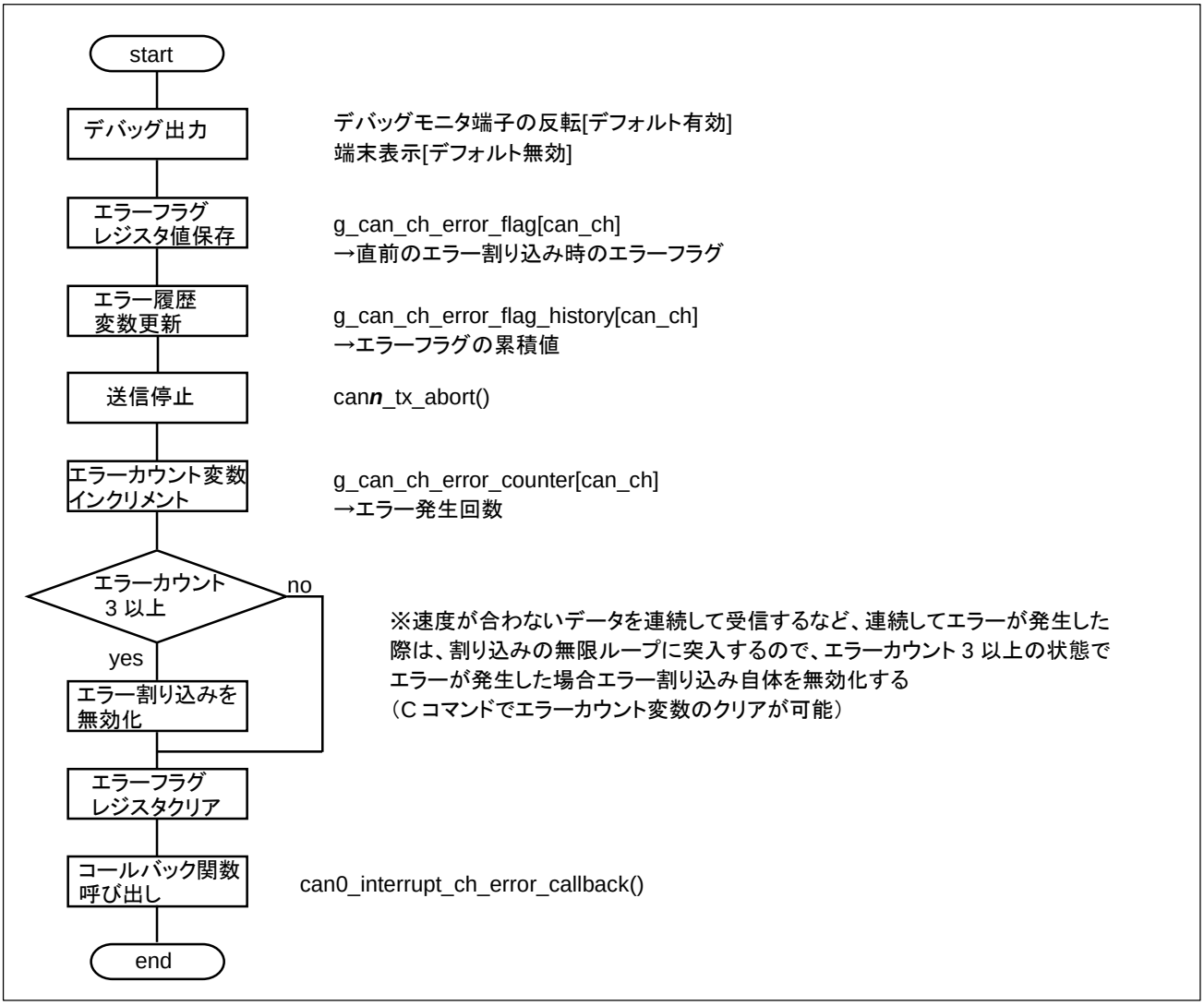
－関数名－
Intr_RSCAN_CHERRINT

－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
・コールバック関数の呼び出し
→can0_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。）

－コールバック関数－

割り込みコールバック関数	備考
can0_interrupt_ch_error_callback()	

ーフローチャートー


start
デバッグ出力
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]
エラーフラグ
レジスタ値保存
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
エラー履歴
変数更新
g_can_ch_error_flag_history[can_ch]
→エラーフラグの累積値
送信停止
can*n*_tx_abort()
エラーカウント変数
インクリメント
g_can_ch_error_counter[can_ch]
→エラー発生回数
エラーカウント
3 以上
no
yes
※速度が合わないデータを連続して受信するなど、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、エラーカウント 3 以上の状態でエラーが発生した場合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）
エラー割り込みを
無効化
エラーフラグ
レジスタクリア
コールバック関数
呼び出し
can0_interrupt_ch_error_callback()
end

4.5. 送受信 FIFO 受信割り込み

－関数名－

Intr_RSCAN_COMFRXINT

－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

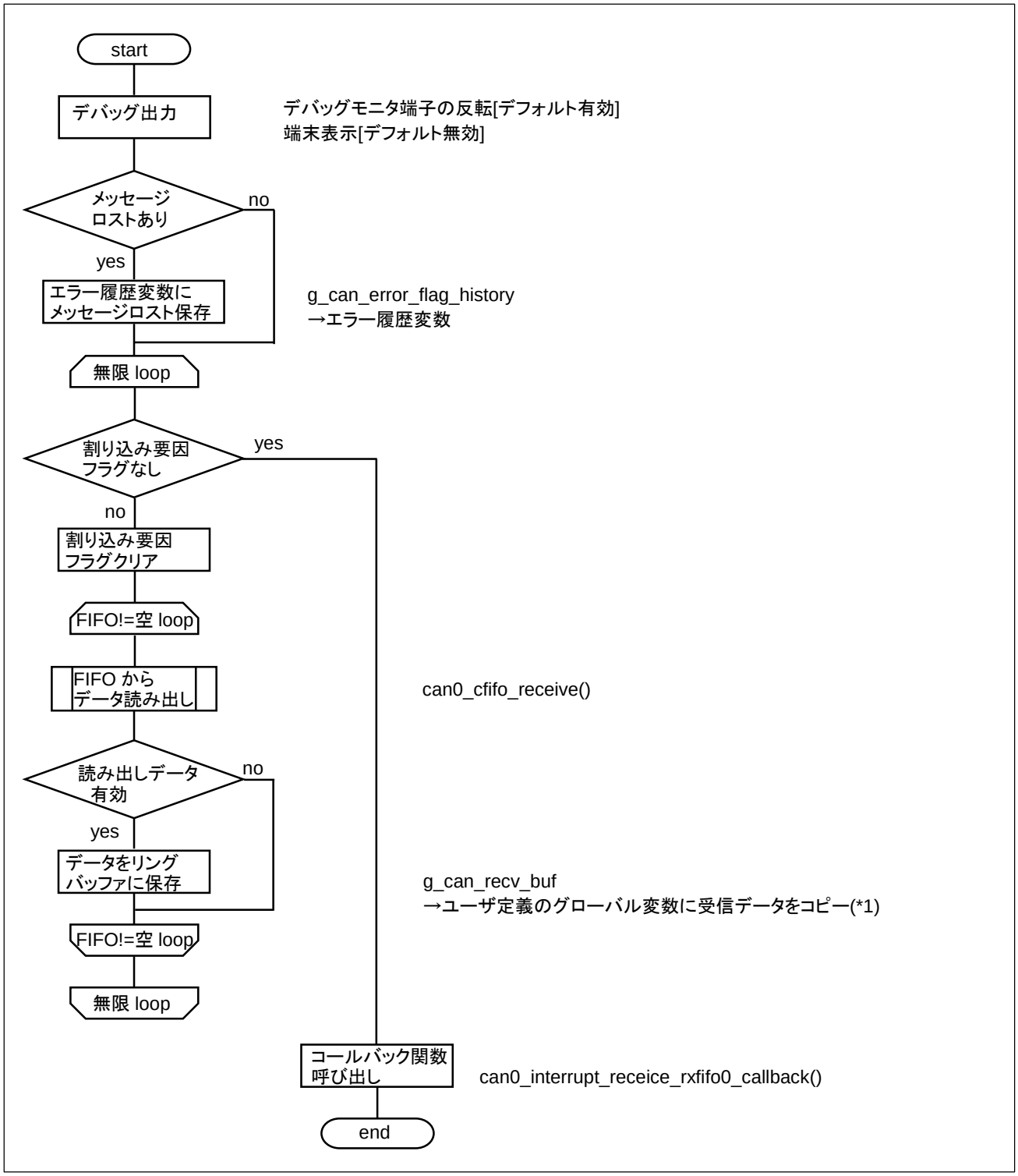
・コールバック関数の呼び出し

→can0_interrupt_receive_srfifo0_callback()

－コールバック関数－

割り込みコールバック関数	受信先	備考
can0_interrupt_receive_srfifo0_callback()	SRFIFO(0)で受信	

ーフローチャートー



(*1)リングバッファへのコピー

リングバッファは、
g_can_recv_buf[CAN_CH0]　:CAN-ch0 で受信したデータを格納（RSCAN は CAN-ch0 のみ）
です。

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）
・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
・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

送信は送信バッファ、受信は受信バッファを使う事とします。

5.2. 受信ルール設定

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

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

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

can_reset(); //RSCAN モジュールのリセット
can_init(); //RSCAN モジュールの初期化
can0_init(); //CAN-ch0 の初期化
can_receive_buf_conf(); //受信バッファの設定

CAN の初期化をする。

(2)受信ルールの設定

//引数: 受信ルール番号 バッファ／FIFO 区分 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
can0_receive_rule_set(0, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
can0_receive_rule_set(1, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000001);
…
can0_receive_rule_set(4, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
…
can0_receive_rule_set(7, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000003);

上表に記載している 8 つの受信ルールを定義する例です。

(3)動作モードへの移行

can_operate();　　//全体の動作設定
can0_operate();　　//CAN-ch0 の動作設定

受信ルール設定後、can_operate, can0_operate を呼び出す。
※動作モード移行後の受信ルール設定はできません

(4)データの受信

```
for(i=0; i<8; i++)  //受信ルール 0~7 が設定済み
{
   //引数:       受信バッファ番号   CAN メッセージ構造体
   r_ret = can0_rxbuf_receive((unsigned char)i, &r_msg);
```

受信バッファ番号: SAMPLE1 では can0_receive_rule_set()で設定した受信ルール番号と同じ
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）

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。

(4)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:　　送信バッファ番号　CAN メッセージ構造体
s_ret = **can0_txbuf_send**(0, &s_msg);

送信バッファ番号:　送信に使用する送信バッファ番号
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

送信バッファ 0 から ID=0x001、データ 0x01, 0x23 の 2 バイト送信する。

送信バッファ番号は、コマンド 0~3 に応じて送信バッファ 0~3 を使う事としています。

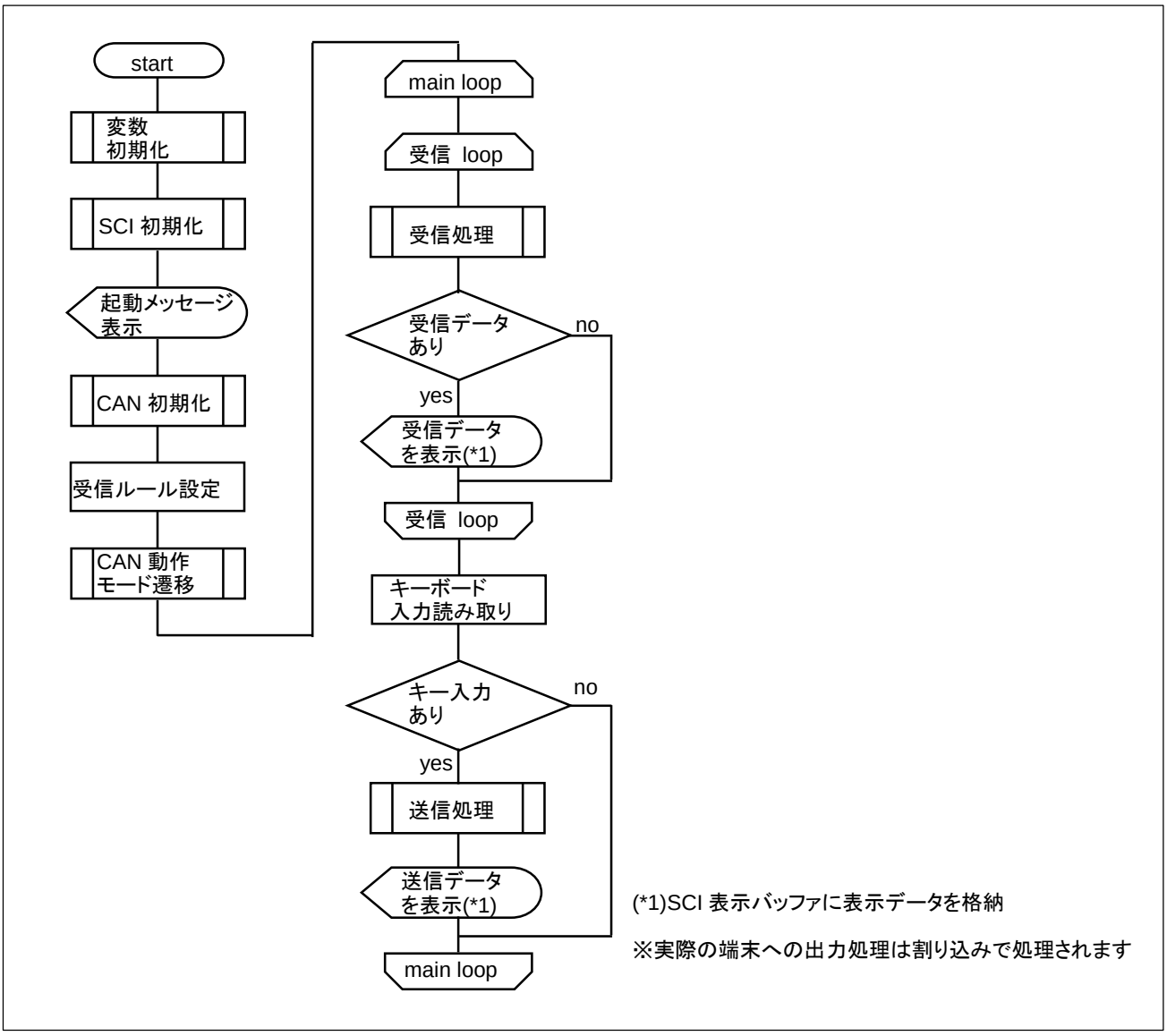
5.4. データの受信に関して

SAMPLE1 でのデータの受信に関しては、受信メッセージバッファまでのデータ格納に関しては、（初期化、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。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。）

5.5. 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
・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、データ送信後の割り込み処理と、受信処理を割り込みを使用して行う事とします。

SAMPLE1 では、受信バッファ・送信バッファを使用していましたが、SAMPLE2 では受信には「受信 FIFO」、送信には「共通 FIFO」を送信の設定で使用する事とします。
※どの送受信方式を選択するかは、can_operation2.h 内の定義で変更可能

6.2. 受信ルール設定

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

6.3. 送信設定

送信は送受信 FIFO を送信で使用する設定です。

送信： SRFIFO(0)

RSCAN モジュールでは、16 個のメッセージバッファが使用できますが、SAMPLE2 では、受信には RXFIFO(0)8 段、送信には SRFIFO(0)8 段を使用する設定です。(16 個のメッセージバッファを全て使用する設定です。

6.4. 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

[SAMPLE1 と同様]

(2)エラー割り込みの有効化

can_error_interrupt_enable(); //グローバルエラー割り込み有効化
can_ch_error_interrupt_enable(0); //チャネルエラー割り込み有効化(CAN-ch0)

SAMPLE2~では、送信割り込み、受信割り込みの他、エラー割り込みも有効化しています。
（エラー割り込みを使用しない場合は、実行不要です）

(3)受信ルールの設定[SAMPLE1 との相違点]

//引数: 受信ルール番号 バッファ／FIFO 区分 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
can0_receive_rule_set(0, **CAN_RULE_RXFIFO0**, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000); //CAN-ch0

…
受信ルール設定時、受信バッファではなく RXFIFO(0)を使うように設定。

(4)動作モードへの移行

[SAMPLE1 と同様]

(5)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。）

(5)データの送信[SAMPLE1 との相違点]

//引数: FIFO 番号 CAN メッセージ構造体
s_ret = **can0_srfifo_send**(0, &s_msg);

常に第一引数の FIFO 番号は 0 で使用する事となります
（SRFIFO 送信関数の FIFO 番号の引数自体を無くす形でも問題ありませんが、関数のインタフェース共通化のために残す形としました。）

6.5. 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)グローバルエラー割り込み

can_interrupt_error_callback();

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)チャネルエラー割り込み

can0_interrupt_ch_error_callback();

子関数
can_interrupt_ch_error_callback(CAN_CH0);　//CAN-ch0 の割り込みなので(CAN_CH0=0)

を、ch 番号を引数にして呼ぶ事としています。
※RSCAN モジュールは、CAN-ch0 のみですが、他のモジュールの割り込みコールバック関数に合わせて、can0_interrupt_ch_error_callback()の中で、can_interrupt_ch_error_callback(CAN_CH0)を呼ぶ様にしています

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_interrupt_receive_rxfifo0_callback();　//RXFIFO(0)で受信の際に呼ばれるコールバック関数

子関数
can_interrupt_rx_callback(CAN_CH0);　//CAN-ch0 の割り込みなので引数に CH0 を与える

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。

※受信に送受信 FIFO を使うよう設定した場合でも同じ子関数を呼び出します。

can_interrupt_receive_rxfifo1_callback();　//RXFIFO(1)で受信の際に呼ばれるコールバック関数
→SAMPLE2 では中身は空（RXFIFO(1)では受信設定を行っておらず、基本的には呼ばれる事がないコールバック関数。）

(3)送信割り込み

can0_interrupt_tx_callback();　//送信完了時に呼ばれるコールバック関数
→can_interrupt_tx_callback(CAN_CH0); //CAN-ch0 の割り込みなので引数に CH0 を与える

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.(SRFIFO)
を表示する処理を行います。

※送信に、送信バッファを使うよう設定した場合でも同じ子関数を呼び出します。

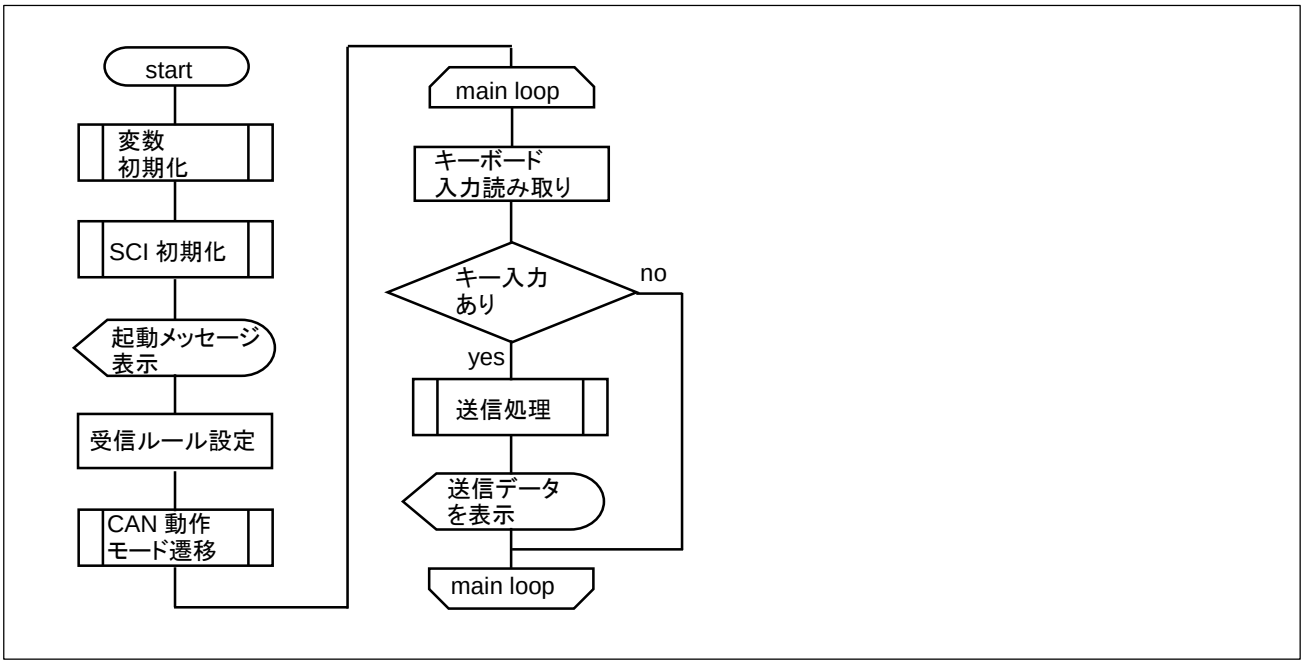
受信割り込みは、「受信 FIFO」「送受信 FIFO（受信）」で、割り込み関数が別となります。
（受信バッファでの受信時には割り込みは使用できない）

それに対し、送信割り込みは、「送受信 FIFO（送信）」「送信バッファ」いずれの送信方式でも、同じ割り込み関数(Intr_RSCAN_TXINT)が呼ばれます。
（割り込み関数内で、どの送信方式の割り込みフラグが立っているかを、フラグ変数に保存していますので、send finished の表示時に使用した送信方式が表示されます。）

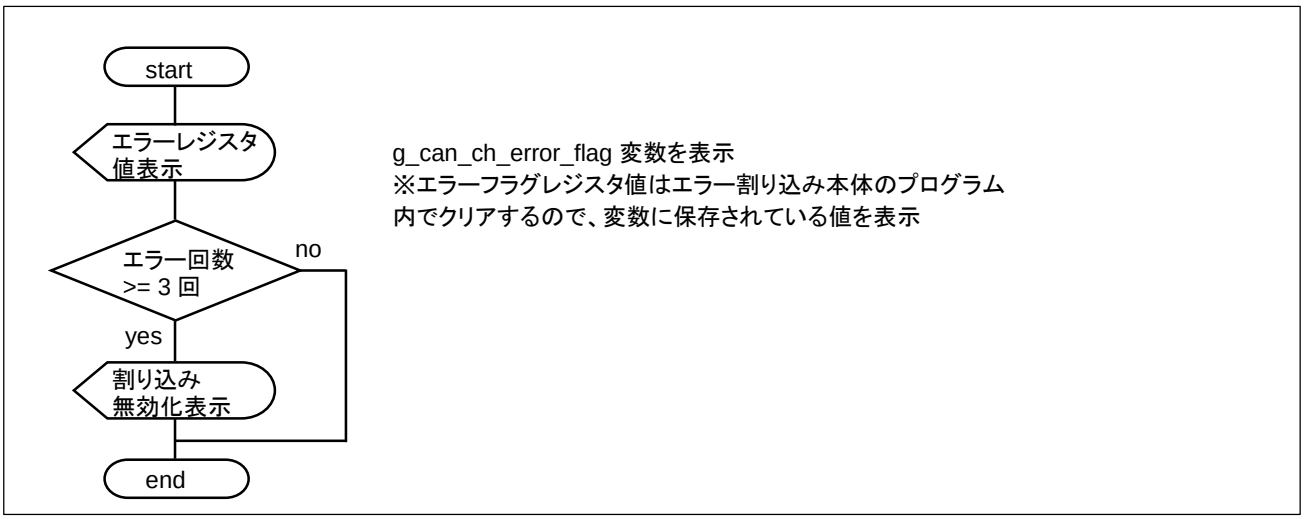
6.6. SAMPLE2 フローチャート

－処理フロー－

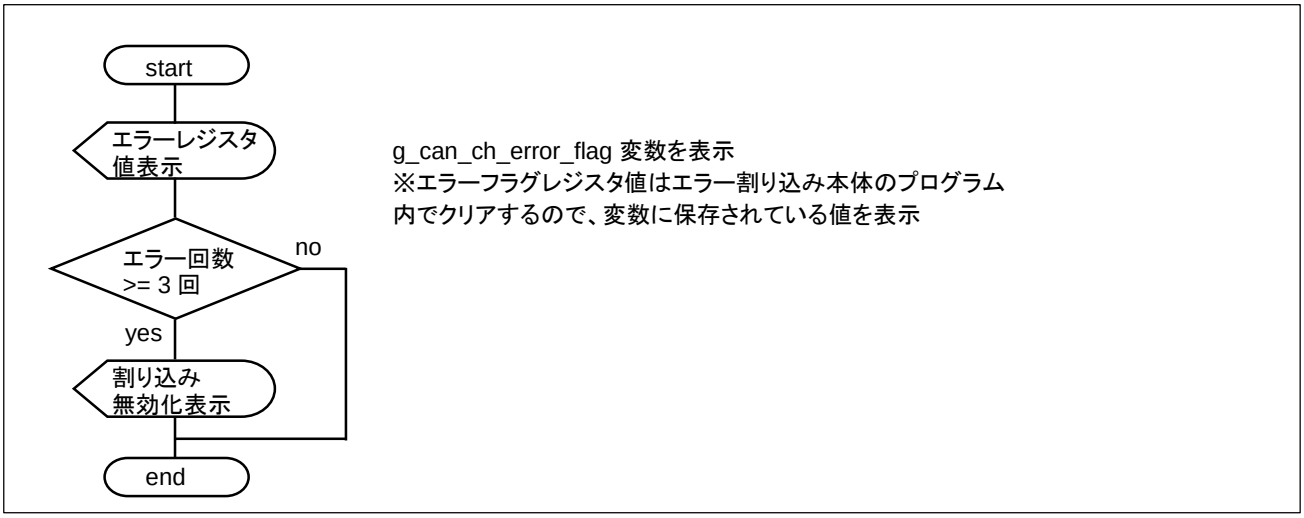
メイン関数　main_s2()



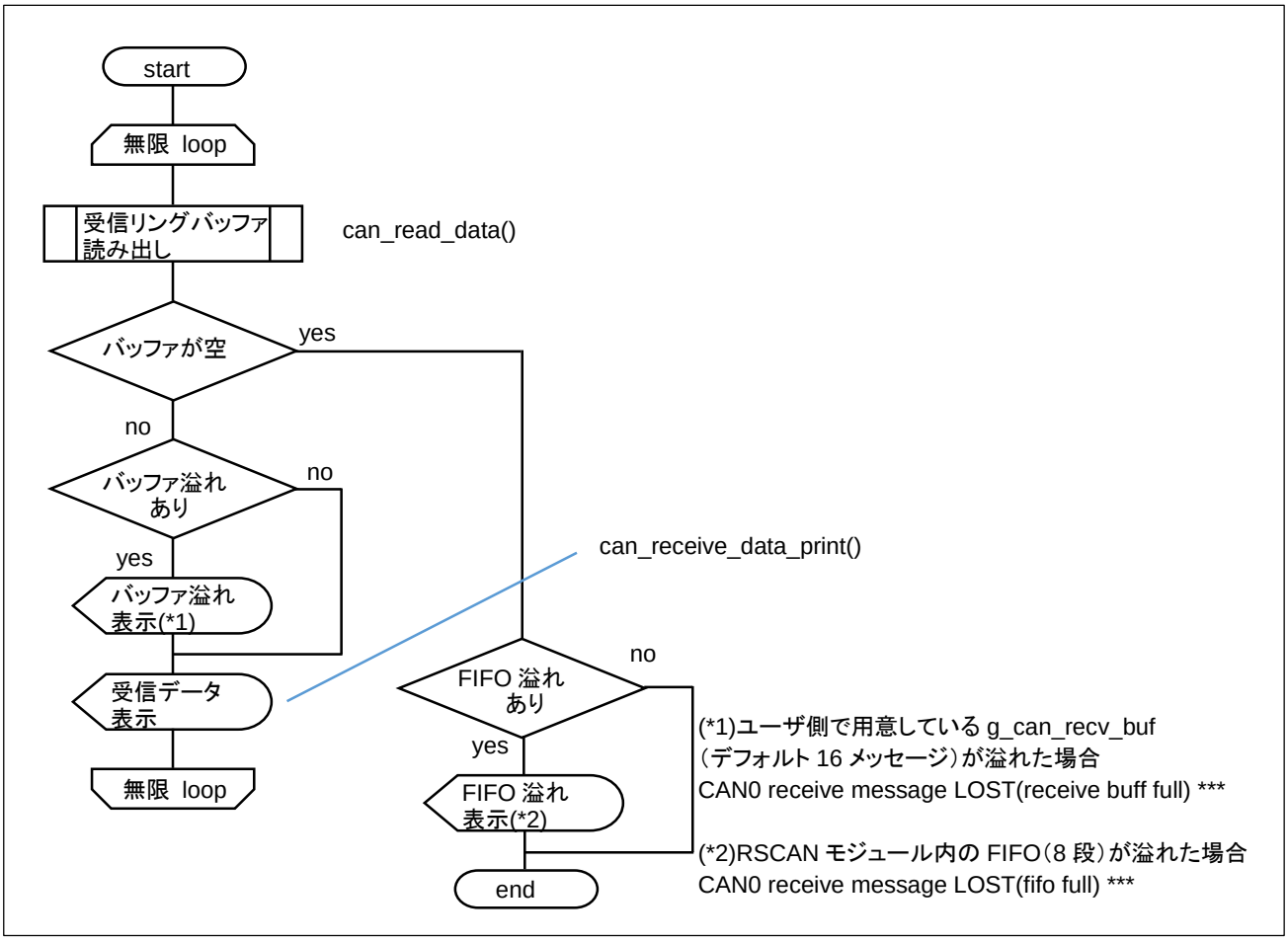
グローバルエラー割り込みコールバック関数　can_interrupt_error_callback()



チャネル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()



割り込み関数(*1)	割り込みコールバック関数(*2)	割り込みコールバック関数から呼ばれる関数(*3)	備考
Intr_RSCAN_RXFINT()	can_interrupt_receive_rxfifo0_callback()	can_interrupt_rx_callback()	RXFIFO(0)で受信
	can_interrupt_receive_rxfifo1_callback()		RXFIFO(1)で受信
Intr_RSCAN_COMFRXINT()	can0_interrupt_receive_srfifo0_callback()	can_interrupt_rx_callback()	SRFIFO(0)で受信

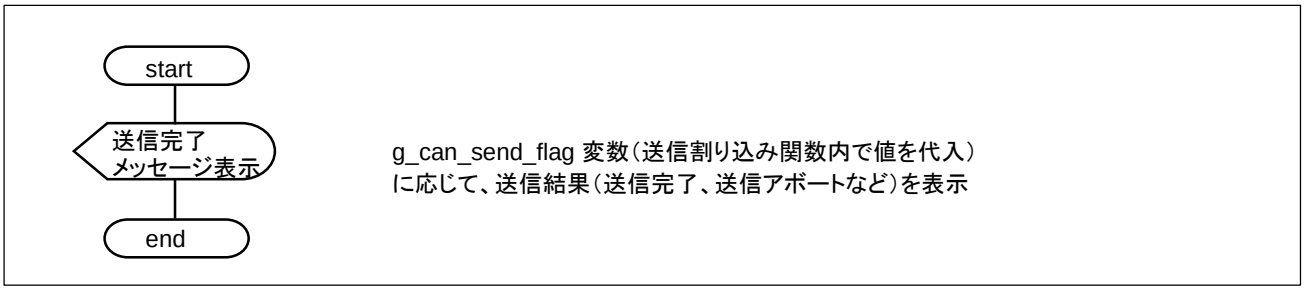
(*1)割り込み関数は
rscan_intr.c
rscan_ch0_intr.c
内に記載されています。
(*2)(*3)main_s2.c 内に記載しています。

割り込み関数に対して、どの FIFO 番号で受信したかにより呼び出す割り込みコールバック関数を分けています。受信 FIFO で受信した場合と送受信 FIFO で受信した場合で、どちらもコールバック関数から最終的に can_interrupt_rx_callback()（受信データを表示する子関数）を呼び出す事としています。

(*3)の欄で空白の部分は、処理は空になっています。（処理を追加する場合は、main_s2.c 内に記載してください。）

受信ルール設定時に、受信ルールをどの FIFO 番号に割り当てるかで受信時に呼び出されるコールバック関数が変わります。
受信ルール設定は、ID（や IDR, RTR）毎に、自由に受信先を設定できますので、ID=0x30 のデータは RXFIFO(0)で受信して、ID=0x31 のデータは、RXFIFO(1)で受信するような設定も問題ありません。
（RXFIFO と SRFIFO（受信）を同時に使用する事も可能ですが、その場合は、ソースコードの変更が必要です。）

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()



割り込み関数	割り込みコールバック関数	割り込みコールバック関数から呼ばれる関数	備考
Intr_RSCAN_TXINT()	can0_interrupt_tx_callback()	can_interrupt_tx_callback()	

送信コールバック関数も受信コールバック関数と同様、最終的に can_interrupt_tx_callback()を呼ぶ形にしています。（送信の場合は、送信方式に拘わらず、最初に呼ばれる割り込み関数は同一です。）

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
・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. 受信ルール設定

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

リモートフレーム向けの受信ルールが追加されている形となります。

※受信ルール 0~7 は変更せず 8~15 に追加する形でも問題ありませんが、can_operaion.h 内で標準 ID のみ取り扱う様に変更した場合、拡張 ID のルールが消える形となります。8~15 にリモートフレームの条件を追加した場合で標準 ID のみ使用する設定だと、ルール番号 4~7 が抜ける状態となり、受信ルール番号 8 以降が有効になりません。受信ルール設定(can0_receive_rule_set)を行う際は番号が途中で抜けることの無い様に設定してください。

7.3. 送信設定

SAMPLE2 から変更ありません。

7.4. 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

[SAMPLE1 と同様]

(2)エラー割り込みの有効化

[SAMPLE1 と同様]

(2)受信ルールの設定[SAMPLE2 との相違点]

//引数:　　受信ルール番号　バッファ／FIFO 区分　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
…
can0_receive_rule_set(4, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, **CAN_REMOTE_FRAME**, 0x00000000); //CAN-ch0
…

受信ルール番号 4~7, 12~15 をリモートフレーム受信向けに追加。

(4)動作モードへの移行

[SAMPLE1 と同様]

(5)データの送信[SAMPLE2 との相違点]

```
can_message s_msg;

s_msg.id = 0x0000001;
s_msg.rtr = CAN_REMOTE_FRAME;
s_msg.ide = CAN_ID_FORMAT_EID;
s_msg.dlc = 2;

//引数:　　FIFO 番号　CAN メッセージ構造体
s_ret = can0_srfifo_send(0, &s_msg);
```

SAMPLE1~SAMPLE2 では、s_msg.rtr は常に、CAN_DATA_FRAME で送信を行っていましたが、SAMPLE3 では、入力されたコマンド(q~r)により REMOTE_FRAME で送信します。

リモートフレーム送信時は、データフレーム時に設定していた送信データ
s_msg.data[]
は使用しません。

DLC 値
s_msg.dlc
は、リモートフレームのメッセージに含まれる（相手に返送して欲しいデータバイト数）ため、設定が必要です。

7.5. 割り込み処理

ここでは、main_s3.c 内に記載している、コールバック関数に関して記載します。

(1)グローバルエラー割り込み

[SAMPLE2 と同様]

(2)チャネルエラー割り込み

[SAMPLE2 と同様]

(3)受信割り込み

コールバック関数から、

can_interrupt_rx_callback(CAN_CH0);　//RSCAN モジュールでは常に CAN_CH0

が呼ばれるのは、SAMPLE2 と変わりません。can_interrupt_rx_callback()関数の中身が以下の様に変わります。

－SAMPLE2－
・受信データの表示

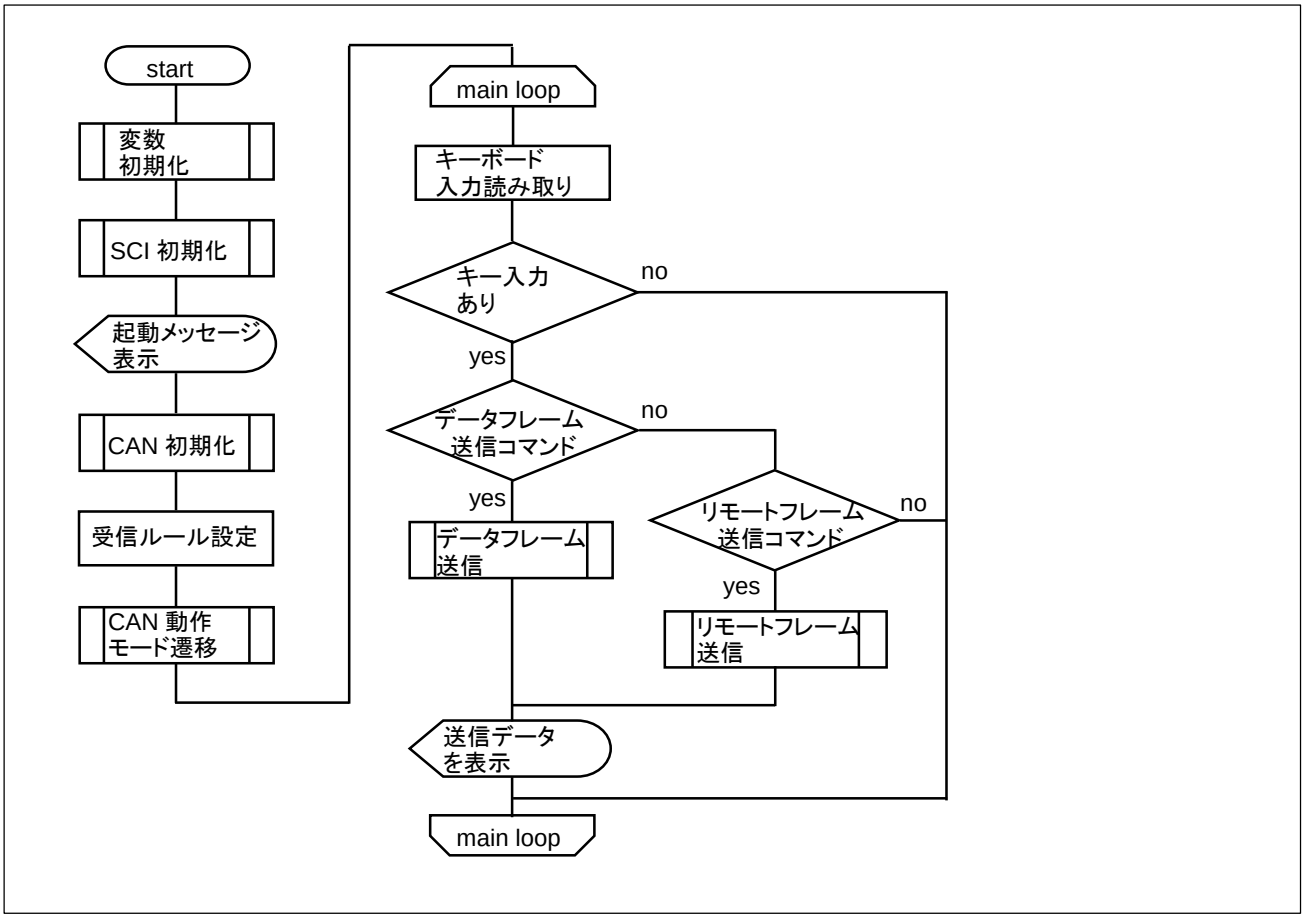
－SAMPLE3－
・受信データの表示
・受信データがリモートフレームの場合データフレームの送信

(4)チャネルエラー割り込み

[SAMPLE2 と同様]

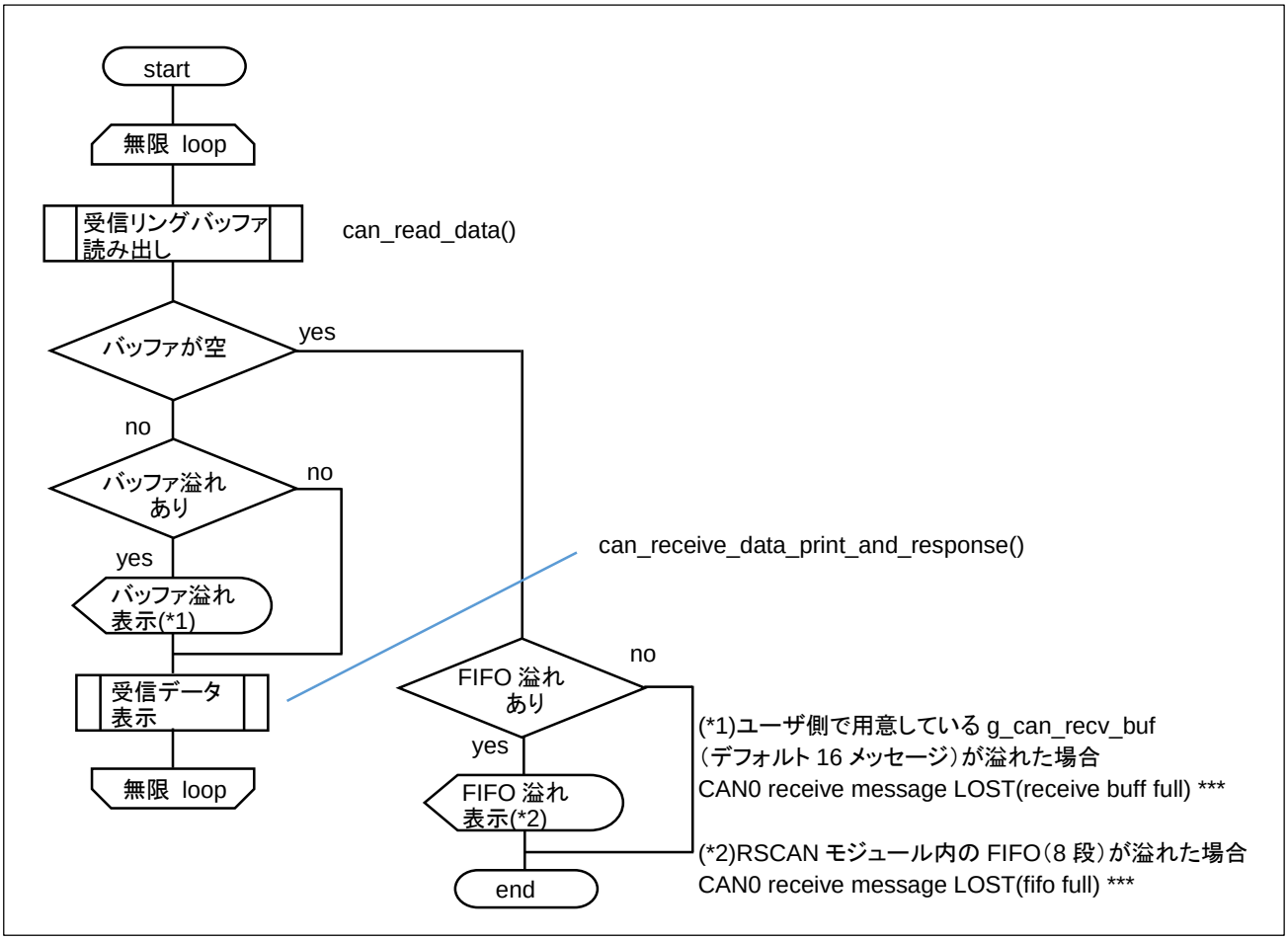
7.6. SAMPLE3 フローチャート

メイン関数　main_s3()

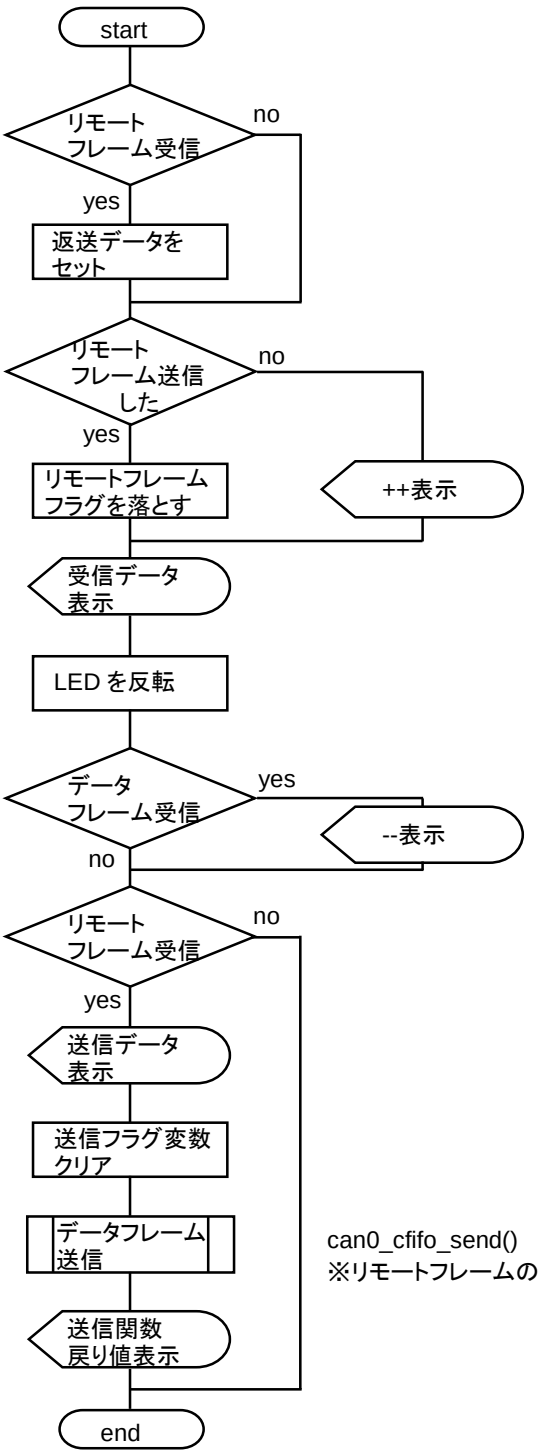


※エラー割り込み、送信割り込みコールバック関数から呼ばれる関数は SAMPLE2 から変更なし

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() ※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）

can0_cfifo_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_reset

概要:初期化関数

宣言:

int can_reset(void)

説明:

・モジュールストップ解除
・グローバルリセットモードへの遷移

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

can_init
can0_init

概要:初期化関数

宣言:

int can_init(void)
int can0_init(void)　[ch0 向け]

説明:

・端子設定
・通信速度設定
・FIFO 設定
・割り込み設定
・チャネルリセットモードへの遷移

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

補足:

can_init()実行後、can0_init()を実行してください。can_reset()後に実行してください。

can_receive_buf_conf

概要:受信メッセージバッファ数・受信ルール数設定関数

宣言:

int can_receive_buf_conf(void)

説明:

・受信メッセージバッファ数設定

・受信ルール数設定

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS (0): 正常終了

補足:

can0_init() 後、can0_receive_rule_set()実行前に実行してください。

can0_receive_rule_set

概要:受信ルール設定関数

宣言:

int can0_receive_rule_set(unsigned char num, unsigned short mode, unsigned char ide, unsigned char rtr, unsigned long 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

mode: 受信先

CAN_RULE_BUF(0x0)　　受信バッファで受信

CAN_RULE_RXFIFO0(0x0001)　　受信 FIFO(0)で受信

CAN_RULE_RXFIFO1(0x0002)　　受信 FIFO(1)で受信

CAN_RULE_SRFIFO0(0x0010)　送受信 FIFO(0)で受信

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_MODE_ERROR(-5): 受信ルール設定不可の動作モード

補足:

受信ルール番号は、0-15 の範囲での指定が可能です

ルール番号は、0 から始め途中に空きの出ない様に設定してください

0,1,2,8,9,10 のルールを設定した場合は、0~2 は有効ですが、8~10 は有効にはなりません

（ルール番号 2 の後にルール番号 1 の受信ルールを設定する事は問題ありません

最終的に設定したルール番号の途中に未設定のルール番号が生じないようにしてください）

can0_receive_rule_mask_set

概要:受信ルールマスク設定関数

宣言:

int can0_receive_rule_mask_set(unsigned char num, unsigned char ide_mask, unsigned char rtr_mask, unsigned long id_mask);

説明:

・受信ルールのマスク設定

を行います

引数:

num: 受信ルール番号

ide_mask: 標準／拡張フォーマット区分のマスク

rtr: データフレーム／リモートフレーム区分のマスク

id: ID マスクを指定

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): 受信ルール設定不可の動作モード

使用例:

can0_receive_rule_mask_set(0, 1, 1, 0x00000000);

→受信ルール 0 番のルールに対し、全ての ID を受信する様に設定する

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

→受信ルール 0 番のルールに対し、標準 ID(SID)、拡張 ID(EID)どちらのデータも受信する様に設定する

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

→受信ルール 0 番のルールに対し、データフレーム・リモートフレーム両方のデータを受信する様に設定する

can0_receive_rule_set(0, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME,
0x00000030); //ID=0x00000030 のデータのみ受信する設定

can0_receive_rule_mask_set(0, 1, 1 0x1FFFFFF8);

→受信ルール 0 番のルールに対し ID=0x00000030~0x00000037 の範囲の ID を受信する様に設定する

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

この例では、マスク値で 0 を指定した b2-0 が任意の値となりますので ID=0x00000030~0x00000037 の範囲を指定する事となります

引数で 0 を指定したところは、比較対象から外すルールが適用されます

補足:

1 つの受信ルールで複数の ID を受信したい場合や、SID/EID の区分なく受信したい場合に本関数を実行してください

必要がなければ本関数の実行は不要です

can_operate
can0_operate

概要:動作モード変更関数

宣言:

int can_operate(void)

int can0_operate(void)

説明:

・RSCAN モジュールの動作モード(グローバルオペレーションモード, チャネル通信モード)への遷移

・FIFO の有効化(FIFO 使用時)

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

補足:

本関数実行後は、受信ルールの変更・追加(can0_receive_rule_set(), can0_receive_rule_mask_set())は実行不可です

can0_txbuf_send

概要:データ送信関数

宣言:

int can0_txbuf_send(unsigned char num, can_message *msg);

説明:

・送信バッファを使用してデータの送信

を行います

引数:

num: 送信バッファ番号(0-3)

msg: 送信データを設定した CAN メッセージ構造体

msg.id: 送信する ID

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.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.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

msg.dlc: 送信 DLC 値を指定します(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): 送信バッファ使用中

can0_srfifo_send

概要:データ送信関数

宣言:

int can0_srfifo_send(unsigned char fifo_no, can_message *msg);

説明:

・送受信 FIFO を使用してデータの送信
を行います

引数:

fifo_no: FIFO 番号(0)

msg: 送信データを設定した CAN メッセージ構造体

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

補足:

fifo_no=0 のみ指定可能です

can0_rxbuf_receive

概要:受信関数

宣言:

int can0_rxbuf_receive(unsigned char num, can_message *msg)

説明:

・受信バッファを使用したデータの受信

を行います

引数:

num: 受信バッファ番号(0-15)

msg: 受信データを格納する CAN メッセージ構造体

msg.id: 受信した ID

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.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.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(受信データに含まれる DLC 値)

CAN_RET_NODATA(-1): 受信バッファにデータが格納されていない

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー、受信バッファ数オーバー

補足:

can_receive_buf_conf()で確保した受信バッファ数を超える num を指定した場合は CAN_RET_ARGUMENT_ERROR(-2)となります

can0_rxfifo_receive

概要:受信関数

宣言:

int can0_rxfifo_receive(unsigned char fifo_no, can_message *msg)

説明:

・受信 FIFO を使用したデータの受信
を行います

引数:

fifo_no: 受信 FIFO 番号(0-1)

msg: 受信データを格納する CAN メッセージ構造体

戻り値:

1~8: 受信したデータのバイト数（受信データに含まれる DLC 値）

CAN_RET_NODATA(-1): 受信バッファにデータが格納されていない

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

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

can0_srfifo_receive

概要:受信関数

宣言:

int can0_srfifo_receive(unsigned char fifo_no, can_message *msg)

説明:

・送受信 FIFO を使用したデータの受信
を行います

引数:

fifo_no: 共通 FIFO 番号（0 のみ）

msg: 受信データを格納する CAN メッセージ構造体

戻り値:

1~8: 受信したデータのバイト数（受信データに含まれる DLC 値）

CAN_RET_NODATA(-1): 受信バッファにデータが格納されていない

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

CAN_RET_MODE_ERROR(-5): 送受信 FIFO が受信モードに設定されていない

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、受信バッファ，受信 FIFO，送受信 FIFO のどの方式を使用した場合でも、受信バッファまたは 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。受信バッファや 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。

can0_tx_abort

概要:送信停止関数

宣言:

int can0_abort(void)

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

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）, RSCAN モジュールでは CAN_CH=1
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]
unsigned long g_can_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。

CAN のエラーレジスタと、グローバルエラー用に 2 つ用意されています。

unsigned long g_can_ch_error_flag_history[CAN_CH]
unsigned long g_can_error_flag_history[CAN_CH]

エラー履歴変数。上記変数の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_error_counter
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	2022.8.10	—	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		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
RSCAN 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)