CAN スタータキット RX/RA
CAN スタータキット SmartRX
CANFD 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.1.12.0.0

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、「CANFD モジュール編」のマニュアルです。CANFD, CANFD_B, CANFD_2, CANFD_Lite 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

・CANFD モジュール向け

フォルダ	ファイル	説明
source¥CANFD_module¥canfd		CANFD 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
	canfd.c	ch に依存しない処理関数，グローバル変数定義
	canfd.h	上記ヘッダ
	canfd_intr.c	グローバル割り込み関数
	canfd_intr.h	グローバル割り込み関数ヘッダ
source¥CANFD_module¥canfd_ch0		ch0 向けフォルダ
	canfd_ch0.c	ch0 向け関数
	canfd_ch0.h	ch0 関数ヘッダ
	canfd_ch0_intr.c	ch0 割り込み関数
	canfd_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥CANFD_module¥canfd_ch1		ch0 向けフォルダ
	canfd_ch1.c	ch1 向け関数
	canfd_ch1.h	ch1 関数ヘッダ
	canfd_ch1_intr.c	ch1 割り込み関数
	canfd_ch1_intr.h	ch1 割り込み関数ヘッダ
source¥CANFD_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
	main_s4.c	SAMPLE4 メイン関数

・CANFD_B モジュール向け

フォルダ	ファイル	説明
source¥CANFD_B_module¥canfd_b		CANFD_B 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
	canfd_b.c	ch に依存しない処理関数，グローバル変数定義
	canfd_b.h	上記ヘッダ
	canfd_b_intr.c	グローバルエラー割り込み関数
	canfd_b_intr.h	グローバルエラー割り込み関数ヘッダ
source¥CANFD_B_module¥canfd_b_ch0		ch0 向けフォルダ
	canfd_b_ch0.c	ch0 向け関数
	canfd_b_ch0.h	ch0 関数ヘッダ
	canfd_b_ch0_intr.c	ch0 割り込み関数
	canfd_b_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥CANFD_B_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
	main_s4.c	SAMPLE4 メイン関数

・CANFD_2 モジュール向け

フォルダ	ファイル	説明
source¥CANFD_2_module¥canfd_2		CANFD_B 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
	canfd_2.c	ch に依存しない処理関数，グローバル変数定義
	canfd_2.h	上記ヘッダ
	canfd_2_intr.c	グローバルエラー割り込み関数
	canfd_2_intr.h	グローバルエラー割り込み関数ヘッダ
source¥CANFD_2_module¥canfd_2_ch0		ch0 向けフォルダ
	canfd_2_ch0.c	ch0 向け関数
	canfd_2_ch0.h	ch0 関数ヘッダ
	canfd_2_ch0_intr.c	ch0 割り込み関数
	canfd_2_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥CANFD_2_module¥canfd_2_ch1		ch1 向けフォルダ
	canfd_2_ch1.c	ch1 向け関数
	canfd_2_ch1.h	ch1 関数ヘッダ
	canfd_2_ch1_intr.c	ch1 割り込み関数
	canfd_2_ch1_intr.h	ch1 割り込み関数ヘッダ
source¥CANFD_2_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
	main_s4.c	SAMPLE4 メイン関数

・CANFD_Lite モジュール向け

フォルダ	ファイル	説明
source¥CANFDLite_module¥canfd_lite		CANFD_B 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
	canfd_lite.c	ch に依存しない処理関数，グローバル変数定義
	canfd_lite.h	上記ヘッダ
	canfd_lite_intr.c	グローバルエラー割り込み関数
	canfd_lite_intr.h	グローバルエラー割り込み関数ヘッダ
source¥CANFDLite_module¥canfd_lite_ch0		ch0 向けフォルダ
	canfd_lite_ch0.c	ch0 向け関数
	canfd_lite_ch0.h	ch0 関数ヘッダ
	canfd_lite_ch0_intr.c	ch0 割り込み関数
	canfd_lite_ch0_intr.h	ch0 割り込み関数ヘッダ
source¥CANFDLite_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
	main_s4.c	SAMPLE4 メイン関数

・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

・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

CANFD 系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)CANFD, CANFD_B, CANFD_Lite モジュール初期化

```
can_reset();
can_init();
can0_init();   //CAN-ch0 使用時
can1_init();   //CAN-ch1 使用時
```

can0_init() は、CAN-ch0 の初期化です。使用する、CAN-ch のみ、can***n***_init()（***n***=0~1、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、使用する ch に応じた can***n***_....を使い分けてください。

(1')CANFD_2 モジュール初期化

```
can0_reset();
can1_reset();
can_init();
can0_init();
can1_init();
```

CANFD_2 モジュールでは、can_reset()が ch 毎に独立しています。使用しない ch の can***n***_reset(), can***n***_init() は不要です。

(2)エラー割り込みの有効化 [オプション]

```
can_error_interrupt_enable();
can_ch_error_interrupt_enable(0);   //CAN-ch0 のエラー割り込みを有効化
can_ch_error_interrupt_enable(1);   //CAN-ch1 のエラー割り込みを有効化
```

エラー割り込みを有効化する場合、can_error_interrupt_enable()（グローバルエラー割り込みの有効化）、can_ch_error_interrupt_enable(CAN-ch)（ch 毎のエラー割り込みの有効化）を実行してください。

(3)CANFD, CANFD_B, CANFD_Lite モジュール受信バッファ・受信ルール数設定

```
can_receive_buf_conf();
can0_receive_rule_set();
```

can1_receive_rule_set();

受信バッファ、受信ルール数の設定を行います。

(3')CANFD_2 モジュール受信バッファ・受信ルール数設定

can0_receive_buf_conf();
can1_receive_buf_conf();
can0_receive_rule_set();
can1_receive_rule_set();

CANFD_2 では、can_receive_buf_conf が ch 毎に独立となります。

(4)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。

(5) CANFD, CANFD_B, CANFD_Lite モジュール動作モードへの移行

can_operate();
can0_operate();
can1_operate();

CAN-ch0, CAN-ch1 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。

(5') CANFD_2 モジュール動作モードへの移行

can0_operate();
can1_operate();

CAN-ch0, CAN-ch1 を動作モードに移行させる。CANFD_2 モジュールでは、can_operate はありません。ch 毎の動作モード移行関数のみがある構成です。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」「共通 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」「送信キュー」「共通 FIFO」と 3 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_CFIFO        //共通 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_TXQUEUE      //送信キューを使用
#define CAN_TX_METHOD       CAN_TX_CFIFO        //共通 FIFO を使用(*2)
```

(*1)SAMPLE1 のデフォルト
(*2)SAMPLE2~4 のデフォルト
※CANFD 系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。

	CANFD	CANFD_B CANFD_Lite	CANFD_2
受信バッファ(*3)	最大 32 全 ch 共通	最大 32	最大 16 ch 毎
受信 FIFO(*4)	RXFIFO(0)~(7) [8 本] 全 ch 共通	RXFIFO(0)~(1) [2 本]	RXFIFO(0)~(1) [2 本] ch 毎
共通 FIFO(*5)	CFIFO(0)~(5) [6 本] 全 ch 共通	CFIFO(0) [1 本]	CFIFO(0) [1 本] ch 毎
TX キュー(*6)	TXQUEUE(0)~(3) [4 本] ch 毎	TXQUEUE(0) [1 本]	TXQUEUE(0) [1 本] ch 毎

(*3)CANFD 系モジュールで使用可能な RAM サイズが決まっています
FIFO の使用段数や、CANFD パケットを取り扱うか否かで、使用可能な受信バッファ数は変わります
※CANFD モジュールでは、受信バッファで受信した場合、受信割り込みの使用はできません

CANFD_B, CANFD_Lite, CANFD_2 系のモジュールでは、共通 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。

(*4)(*5)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。
(*6)CANFD モジュールでは、キューの段数は 16、その他のモジュールでは 4 です。

ー受信バッファ数に関してー

受信バッファ、FIFO は同じメモリ領域を使用するので、FIFO 使用有無によって、使用可能な受信バッファの数は異なります。また、CANFD 未使用（CAN メッセージのデータバイト数最大 8）と CANFD 使用（同 64）でも変わります。
FIFO は、本サンプルプログラムでは、1 つの FIFO あたり 8 段に設定しています（メモリサイズを超過しない限り変更は可）。
CANFD モジュール系では、受信 FIFO、共通 FIFO を受信に使用、受信バッファの 3 種類を併用する事も可能ですが、本サンプルプログラムでは、受信は 1 つの方法。送信も 1 つの方法を選択する形を取っています。
（受信方式に、受信 FIFO を選んだ場合は、受信バッファは使用しない設定。）

		受信バッファ数		
CANFD 使用	送信に 共通 FIFO を使用	CANFD	CANFD_B CANFD_Lite	CANFD_2
×	×	32（全 ch 共通）	32	16（ch 毎）
×	○	32（全 ch 共通）	24	16（ch 毎）
○	×	32（全 ch 共通）	16	16（ch 毎）
○	○	32（全 ch 共通）	8	8（ch 毎）

受信方式に、受信バッファを選ばなかった場合は、受信バッファ数は 0 に設定されます。

※CANFD 系のモジュールでは、受信バッファ数や FIFO 段数を CANFD 系モジュールが使用可能な RAM サイズに収める必要があります。設定が RAM サイズを超過した場合でもエラーを検出する方法がなく、動作がおかしくなる（受信データが化ける、受信するはずのデータを受信しない）といった挙動となるので、注意が必要です。

3.1. データの送信

CAN, CANFD 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;
```

・CANFD

```
int ret;
canfd_message fdmsg;

msg.id = 0x00000001;
msg.ide = CAN_ID_FORMAT_EID;
msg.rtr = CAN_DATA_FRAME;
msg.fdf = CANFD_DATA_FORMAT; //CANFD フォーマットでの送信
msg.brs = CANFD_BITRATE;  //データ部の通信速度を CANFD のビットレートにする
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); //CAN フォーマットでの送信
ret = canfd0_txbuf_send (0, &fdmsg); //CANFD フォーマットでの送信
```

0 は、送信バッファ番号です。

※CAN-ch1 からの送信の場合は、can1_~となります

(2)TX キューを使用した送信

```
ret = can0_txqueue_send(0, &msg);
ret = canfd0_txqueue_send(0, &fdmsg);
```

0 は、TX キュー番号です。can0_~は CAN-ch0 から CAN フォーマットでの送信、canfd0_~は CAN-ch0 から CANFD フォーマットでの送信となります。送信バッファは、1 つのメッセージのみ取り扱う（前回呼び出し時のメッセージが送信完了となった後で、次のメッセージを受け入れ可能な）形ですが、TX キューは複数のメッセージを溜めておく事が出来ます。

(3)共通 FIFO を使用した送信

```
ret = can0_cfifo_send(0, &msg);
ret = canfd0_cfifo_send(0, &fdmsg);
```

0 は、FIFO 番号です。共通 FIFO を使用した場合、TX キュー同様複数のメッセージを FIFO に溜めておく事が可能です。

(1)~(3)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。fdmsg は、CANFD メッセージ構造体で、CANFD で使用するメンバが追加されている構造体です。

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（アクセプタンスフィルタリスト:AFL）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。

	CANFD モジュール	CANFD_B モジュール CANFD_Lite モジュール	CANFD_2 モジュール
ルール番号 （ch 毎）	0~63	0~31	0~15
AFL	0~15×8 ページ（全 ch）	0~15×2 ページ	0~15（ch 毎）
受信先	CAN_RULE_RXBUF(*1) CAN_RULE_RXFIFO0~ CAN_RULE_RXFIFO7 CAN_RULE_CFIFO0~ CAN_RULE_CFIFO5	CAN_RULE_RXBUF(*1) CAN_RULE_RXFIFO0 CAN_RULE_RXFIFO1 CAN_RULE_CFIFO0	CAN_RULE_RXBUF(*1) CAN_RULE_RXFIFO0 CAN_RULE_RXFIFO1 CAN_RULE_CFIFO0

※CANFD 系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（CANFD モジュールでは 8 箇所、それ以外のモジュールは 2 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、本サンプルプログラムで定義している番号で、マイコン側は 16 ルールを複数ページで構成する形となっています。

CANFD モジュールでは、CAN-ch0 側は page=0~3, CAN-ch1 側は page=4~7 を割り当てています。

CAN-ch0 のルール番号 16 は page=1, AFL=0

CAN-ch1 のルール番号 0 は、page=4, AFL=0

を使用しています。

CANFD_B, CANFD_Lite モジュールでは、ルール番号 16~31 は、page=1 の AFL=0~15 の割り当てです。

CANFD_2 モジュールでは、page=0 のみとなります。

(*1)受信先として受信メッセージ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

－CANFD モジュール－

CAN-ch	ルール番号	CANFD モジュールで使用される 受信バッファ番号 （受信バッファ：0-32, ch 共有）
CAN0	0-15 16-31 32-47 48-63	0-15 0-15 0-15 0-15
CAN1	0-15 16-31 32-47 48-63	16-31 16-31 16-31 16-31

受信バッファは最大 32 となり、CAN-ch0 側で 0-15、CAN-ch1 側で 16-31 を割り当てる設定です。（CAN0, CAN1 のいずれかのみを使用する場合は、プログラムを変更して、片方の ch で 32 個のバッファを使用する様にする事も可能です。）受信ルールは、64 ルール設定可能ですが、受信バッファ数の制約により、受信ルール 0 と受信ルール 16 は同じ受信バッファ 0 にデータが格納されます。（受信バッファを独立で使用したい場合は、設定するルール数は 16 ルールまでとしてください。）

－CANFD_B, CANFD_Lite モジュール－

使用可能な受信バッファ数は、CANFD 使用有無や共通 FIFO 使用有無によって 8~32 の範囲で変わります。設定可能な受信ルール数は 32 となります。

CAN-ch	ルール番号	CANFD モジュールで使用される 受信バッファ番号 （受信バッファ数：条件により 8,16,32）
CAN0	0-31	ルール番号 % 設定した受信バッファ数

%: 余剰

受信バッファが 32 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 30 の場合は、受信バッファ 6 を使用します。（ルール番号 6, 14, 22, 30 で受信バッファ 6 を共有します。）

－CANFD_2 モジュール－

使用可能な受信バッファ数は、共通 FIFO 使用有無によって 8~16 の範囲で変わります。設定可能な受信ルール数は 16 となります。

CAN-ch	ルール番号	CANFD モジュールで使用される 受信バッファ番号 （条件により 8,16, ch 毎）
CAN0	0-15	ルール番号 % 設定した受信バッファ数
CAN1	0-15	ルール番号 % 設定した受信バッファ数

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;
canfd_message fdmsg;
```

(1)受信バッファを使用した受信

```
ret = can0_rxbuf_receive(0, &msg);
ret = canfd0_rxbuf_receive(0, &fdmsg);
```

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/fdmsg 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(can***n***_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、can***n***_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[CANFD モジュール]

```
ret = can_rxfifo_receive(0, &msg);
ret = canfd_rxfifo_receive(0, &fdmsg);
```

RXFIFO(0)に格納されているメッセージを、msg/fdmsg 構造体にコピーします。0 は、RXFIFO 番号です。0~7 の値が指定可能です。

CANFD モジュールでは、CAN-ch0, CAN-ch1（CAN の物理 ch）に拘わらず、RXFIFO が構成されているので、can**0**, canfd**0** の 0 が付かない形で関数を用意しています。（どちらの物理 ch で受信したデータも、同じ関数で受信する形となります。どちらの CAN の物理 ch で受信したかを明確にしたい場合は、can***n***_receive_rule_set()の時点で、CAN-ch0 のデータを RXFIFO(0)、CAN-ch1 のデータを RXFIFO(1)で受信するなど、FIFO 番号を別にしてください。

can***n***_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。

ーコード例ー

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

(2')受信 FIFO を使用した受信[CANFD_B, CANFD_2, CANFD_Lite モジュール]

```
ret = can0_rxfifo_receive(0, &msg);
ret = canfd0_rxfifo_receive(0, &fdmsg);
```

CANFD_2 モジュールの場合は、CAN-ch0 の RXFIFO と CAN-ch1 の RXFIFO が別になっているので、can**0**_~, canfd**0**_~の様に ch 番号が関数に含まれます。CAN-ch1 のデータを受信する場合は、can**1**_~の関数を使用してください。

第一引数の 0 は、RXFIFO(0)の指定ですが、このタイプのモジュールでは 0 または 1 が指定可能です。

※CANFD_B, CANFD_Lite モジュールでは、CAN-1ch のみ（CAN0 のみ）です。
本来は、CANFD_B, CANFD_Lite モジュールでは RXFIFO は ch 非依存の構成となっているので、関数名に 0 が付かないのが、あるべき構成かとも思いますが、本サンプルプログラムでは、CANFD_B, CANFD_Lite モジュールの受信関数名は関数名に ch 番号を含める形としています

(3)共通 FIFO を使用した受信[CANFD モジュール]

```
ret = can_cfifo_receive(0, &msg);
ret = canfd_cfifo_receive(0, &fdmsg);
```

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、CFIFO(0)のデータを読み出す場合です。0~5 の値が指定可能です。

(3')共通 FIFO を使用した受信[CANFD_B, CANFD_2, CANFD_Lite モジュール]

```
ret = can0_cfifo_receive(0, &msg);
ret = canfd0_cfifo_receive(0, &fdmsg);
```

これらのモジュールでは、関数名に can**0**_~の様に CAN-ch 番号が入ります。引数 2 つ持つ形の関数ですが、第一引数は 0 のみを指定可能です。常に第一引数は 0 で本関数を使用してください。
※CANFD モジュールなど複数本の共通 FIFO を使用可能なマイコン向けのプログラムとの互換性を持たせるため、FIFO 番号を第一引数に指定する形としています

4. 割り込み

CANFD 系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. CANFD モジュール[RA6M5]

・グローバル割り込み

ch 非依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
RXFIFO 受信	CANFD	CAN_RXF	Intr_can_rxf
グローバルエラー	CANFD	CAN_GLERR	Intr_can_glerr
DMA(RX)	CANFD	CAN_RF_DMAREQ0 … CAN_RF_DMAREQ7	※未使用

呼び出される関数名は、FSP の割り込み設定でユーザが指定するものです。本サンプルプログラムでは、表に記載の名称の関数名を指定して、関数定義を行っています。DMA 関連の割り込みは、本サンプルプログラムでは使用していません。

・ch 割り込み

ch 依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
チャネル送信	CANFD	CAN0_TX	Intr_can0_tx
チャネルエラー	CANFD	CAN0_CHERR	Intr_can0_cherr
共通 FIFO 受信	CANFD	CAN0_COMFRX(*1)	intr_can0_comfrx
DMA(TX)	CANFD	CAN0_CF_DMAREQ	※未使用
チャネル送信	CANFD	CAN1_TX	Intr_can1_tx
チャネルエラー	CANFD	CAN1_CHERR	Intr_can1_cherr
共通 FIFO 受信	CANFD	CAN1_COMFRX(*1)	intr_can1_comfrx
DMA(TX)	CANFD	CAN1_CF_DMAREQ	※未使用

(*1)CAN-ch0 が CAN0_COMFRX で処理される訳ではなく、CFIFO(0)~CFIFO(2)が CAN0_COMFRX で、CFIFO(3)~CFIFO(5)が CAN1_COMFRX で処理されます。
（CAN-ch1 のデータを CFIFO(1)で受信する様に設定した場合は、CAN-ch1 のデータ受信の際に、CAN0_COMFRX の割り込みが呼ばれます）

4.1.2. CANFD_B モジュール[RA6T2, RA6T3, RA6E2, RA4T1, RA4E2]

・グローバル割り込み

ch 非依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
RXFIFO 受信	CANFD	CAN_RXF	intr_can_rxf
グローバルエラー	CANFD	CAN_GLERR	intr_can_glerr
受信バッファ受信	CANFD	CAN0_RXMB	intr_can0_rxmb
DMA(RX)	CANFD	CAN_RF_DMAREQ0 CAN_RF_DMAREQ1	※未使用

CANFD モジュールに対し、受信バッファでの割り込みが使用可能になっています。

※受信バッファの割り込みは、名称は CAN0~となっていますが、分類はグローバル割り込みの定義です

・ch 割り込み

ch 依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
チャネル送信	CANFD	CAN0_TX	intr_can0_tx
チャネルエラー	CANFD	CAN0_CHERR	intr_can0_cherr
共通 FIFO 受信	CANFD	CAN0_COMFRX	intr_can0_comfrx
DMA(TX)	CANFD	CAN0_CF_DMAREQ	※未使用

4.1.3. CANFD_2 モジュール[RA8E1, RA8M1, RA8T1]

・グローバル割り込み

ch 非依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
RXFIFO 受信	CANFD	CAN_RXF	intr_can_rxf
グローバルエラー	CANFD	CAN_GLERR	intr_can_glerr
受信バッファ受信	CANFD	CAN0_RXMB	intr_can0_rxmb
受信バッファ受信	CANFD	CAN1_RXMB	intr_can1_rxmb
DMA(RX)	CANFD	CAN0_RF_DMAREQ0 CAN0_RF_DMAREQ1 CAN1_RF_DMAREQ0 CAN1_RF_DMAREQ1	※未使用

・ch 割り込み

ch 依存の割り込みです。

割り込み種別	割り込み 要求元	名称	呼び出される 割り込み関数名
チャネル送信	CANFD	CAN0_TX	Intr_can0_tx
チャネルエラー	CANFD	CAN0_CHERR	Intr_can0_cherr
共通 FIFO 受信	CANFD	CAN0_COMFRX	intr_can0_comfrx
DMA(TX)	CANFD	CAN0_CF_DMAREQ	※未使用
ECC	CANFD	CAN0_MRAM_ERI	※未使用
チャネル送信	CANFD	CAN1_TX	Intr_can1_tx
チャネルエラー	CANFD	CAN1_CHERR	Intr_can1_cherr
共通 FIFO 受信	CANFD	CAN1_COMFRX	intr_can1_comfrx
DMA(TX)	CANFD	CAN1_CF_DMAREQ	※未使用
ECC	CANFD	CAN1_MRAM_ERI	※未使用

4.1.4. CANFD_Lite モジュール[RX660, RX26T]

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
CANFD	RFDREQ0	104	※未使用	受信 FIFO-DMA
CANFD	RFDREQ1	105	※未使用	受信 FIFO-DMA

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
CANFD0	CFDREQ0	106	※未使用	共通 FIFO-DMA

選択型割り込み A 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
96	CANFD	EC1EI	-	※未使用	1 ビット ECC エラー
97	CANFD	EC2EI	-	※未使用	2 ビット ECC エラー
98	CANFD	ECOVI	-	※未使用	ECC オーバフロー

割り込み 要求元	名称	割り込み番号
ICU	GROUPBL2	107

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
BL2	1	CANFD0	CHEI	Intr_CANFD0_CHEI	チャネルエラー
	2	CANFD0	CFRI	Intr_CANFD0_CFRI	共通 FIFO 受信
	3	CANFD	GLEI	Intr_CANFD_GLEI	グローバルエラー
	4	CANFD	RFRI	Intr_CANFD_RFRI	受信 FIFO
	5	CANFD0	CHTI	Intr_CANFD0_CHTI	チャネル送信
	6	CANFD	RMRI	Intr_CANFD_RMRI	受信バッファ

DMA, ECC エラー割り込みは、本サンプルプログラムでは未使用です。

使用する CAN の割り込みは、グループ BL2 に割り当てられています。

プログラムで使用する割り込みは全て、107 番のグループ割り込みで処理されます。呼ばれる関数は、割り込み要因毎に別となります。

4.1.5. CANFD_Lite モジュール[RX261]

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
CANFD	RFRI	186	Intr_CANFD_RFRI	受信 FIFO
CANFD	GLEI	187	Intr_CANFD_GLEI	グローバルエラー
CANFD	RMRI	188	Intr_CANFD_RMRI	受信バッファ
CANFD	RFDREQ0	189	※未使用	受信 FIFO-DMA
CANFD	RFDREQ1	190	※未使用	受信 FIFO-DMA
CANFD	EC1EI	191	※未使用	1 ビット ECC エラー
CANFD	EC2EI	192	※未使用	2 ビット ECC エラー
CANFD	ECOVI	193	※未使用	ECC オーバフロー

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
CANFD0	CHTI	194	Intr_CANFD0_CHTI	チャネル送信
CANFD0	CHEI	195	Intr_CANFD0_CHEI	チャネルエラー
CANFD0	CFRI	196	Intr_CANFD0_CFRI	共通 FIFO 受信
CANFD0	CFDREQ0	197	※未使用	共通 FIFO-DMA

DMA, ECC エラー割り込みは、本サンプルプログラムでは未使用です。

RX261 の場合、使用する CAN の割り込みは、独立した割り込み番号を割り当てられています。

基本的には、CANFD 系の割り込みは、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

・受信バッファ（CANFD モジュールを除く）

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。

4.2. 受信 FIFO 割り込み

－関数名－

RX: Intr_CANFD_RFRI
RA: intr_can_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_rxfifo***m***_callback()

（***n***: CANFD_2 モジュールのみ CAN-ch、他のモジュールでは n はなし(can_interrupt...)）

（***m***: RXFIFO(m), RXFIFO 番号）

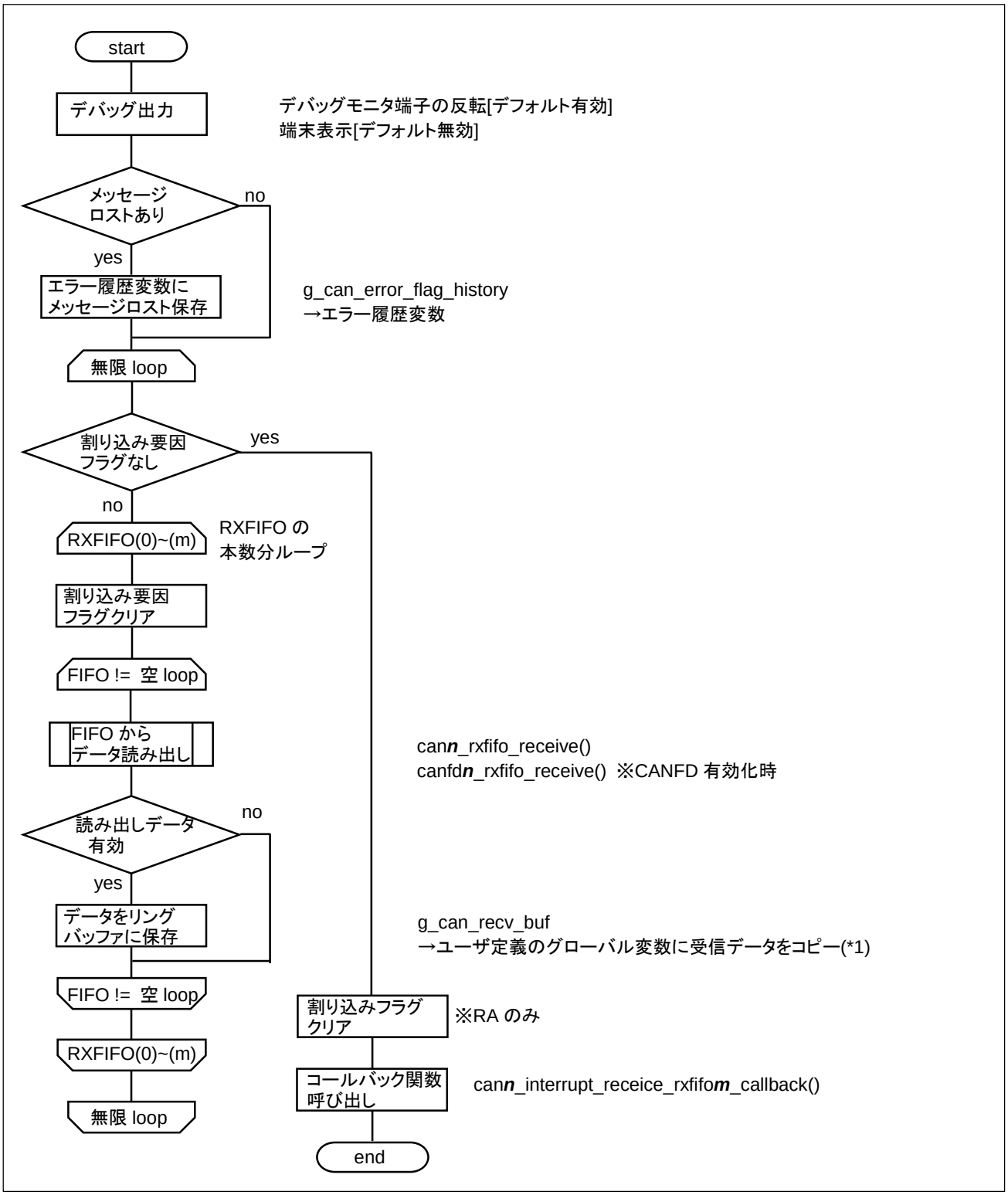
－コールバック関数（CANFD, CANFD_B, CANFD_Lite モジュール）－

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	
can_interrupt_receive_rxfifo2_callback()	RXFIFO(2)で受信	CANFD モジュールのみ
can_interrupt_receive_rxfifo3_callback()	RXFIFO(3)で受信	CANFD モジュールのみ
can_interrupt_receive_rxfifo4_callback()	RXFIFO(4)で受信	CANFD モジュールのみ
can_interrupt_receive_rxfifo5_callback()	RXFIFO(5)で受信	CANFD モジュールのみ
can_interrupt_receive_rxfifo6_callback()	RXFIFO(6)で受信	CANFD モジュールのみ
can_interrupt_receive_rxfifo7_callback()	RXFIFO(7)で受信	CANFD モジュールのみ

－コールバック関数（CANFD_2 モジュール）－

割り込みコールバック関数	CAN-ch	受信先	備考
can0_interrupt_receive_rxfifo0_callback()	0	RXFIFO(0)で受信	
can0_interrupt_receive_rxfifo1_callback()	0	RXFIFO(1)で受信	
can1_interrupt_receive_rxfifo2_callback()	1	RXFIFO(2)で受信	
can1_interrupt_receive_rxfifo3_callback()	1	RXFIFO(3)で受信	

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]
メッセージ
ロストあり
no
yes
エラー履歴変数に
メッセージロスト保存
g_can_error_flag_history
→エラー履歴変数
無限 loop
割り込み要因
フラグなし
yes
no
RXFIFO(0)~(m)
RXFIFO の
本数分ループ
割り込み要因
フラグクリア
FIFO != 空 loop
FIFO から
データ読み出し
can*n*_rxfifo_receive()
canfd*n*_rxfifo_receive() ※CANFD 有効化時
読み出しデータ
有効
no
yes
データをリング
バッファに保存
g_can_recv_buf
→ユーザ定義のグローバル変数に受信データをコピー(*1)
FIFO != 空 loop
RXFIFO(0)~(m)
無限 loop
割り込みフラグ
クリア
※RA のみ
コールバック関数
呼び出し
can*n*_interrupt_receice_rxfifo*m*_callback()
end

(*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
g_can_recv_buf[CAN_CH1]　:CAN-ch1 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）

－CANFD_B, CANFD_Lite モジュール－

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

－CANFD モジュール－

受信元	格納先
RXFIFO(0)	g_can_recv_buf[CAN_CH0]
RXFIFO(1)	g_can_recv_buf[CAN_CH1]
RXFIFO(2)	
RXFIFO(3)	
RXFIFO(4)	
RXFIFO(5)	
RXFIFO(6)	
RXFIFO(7)	

格納先が空白のところは、受信処理のみ行いリングバッファへのコピーは行いません。（動作を変更する場合は、canfd_intr.c を編集してください。）

－CANFD_B, CANFD_Lite モジュール－

受信元	格納先
CAN-ch0, RXFIFO(0)	g_can_recv_buf[CAN_CH0]
CAN-ch0, RXFIFO(1)	g_can_recv_buf[CAN_CH0]
CAN-ch1, RXFIFO(0)	g_can_recv_buf[CAN_CH1]
CAN-ch1, RXFIFO(1)	g_can_recv_buf[CAN_CH1]

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);　//CAN-ch0 向けのリングバッファからの読み出し
```

とします。

4.3. グローバルエラー割り込み

－エラー割り込み対象－

・CANFD ペイロードオーバフロー

・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－

RX: Intr_CANFD_GLEI
RA: intr_can_glerr

－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

・割り込みフラグクリア

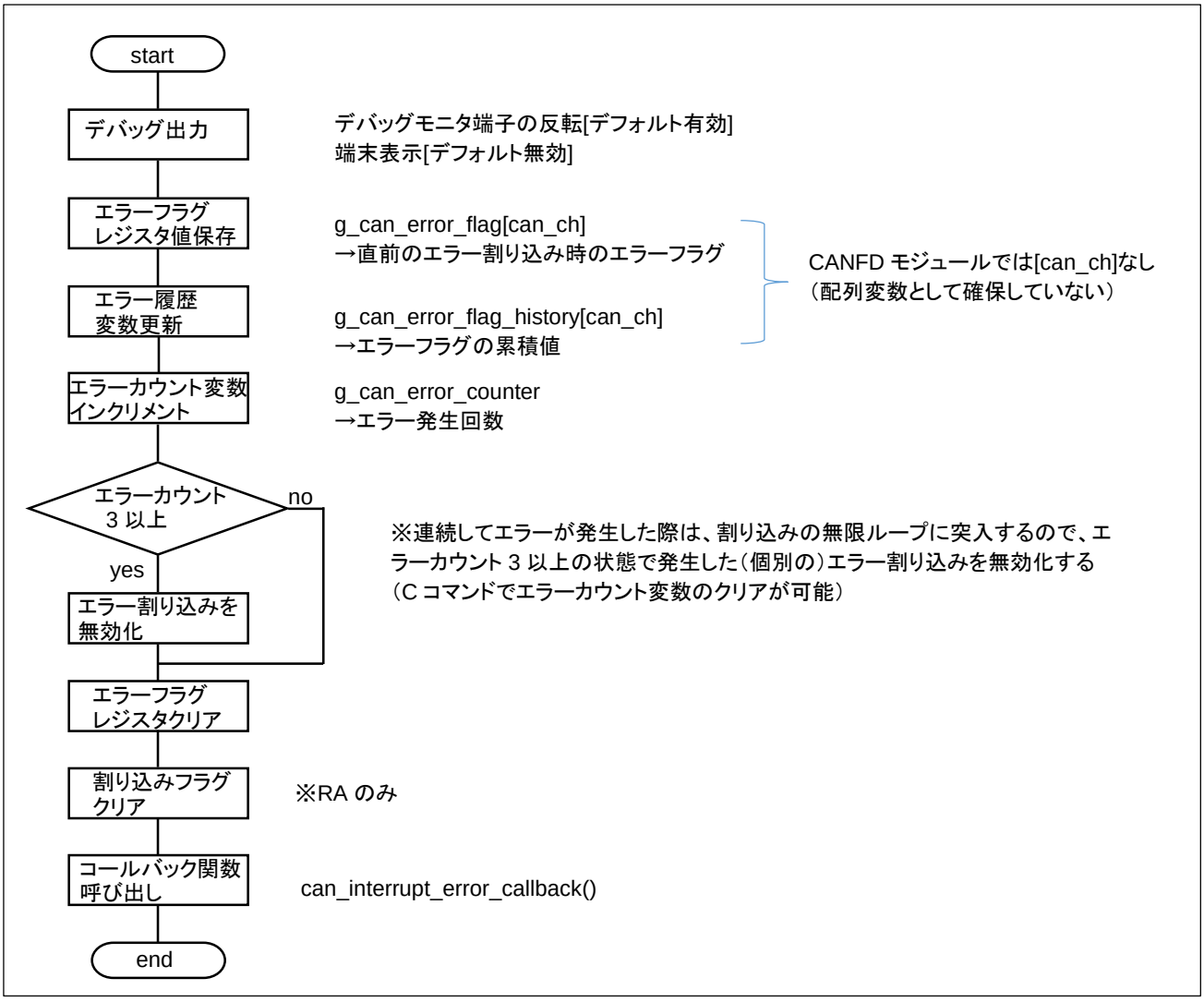
・コールバック関数の呼び出し

→can_interrupt_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()	

－フローチャート－


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
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
CANFD モジュールでは[can_ch]なし
（配列変数として確保していない）
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、エラーカウント 3 以上の状態で発生した（個別の）エラー割り込みを無効化する
（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
割り込みフラグ
クリア
※RA のみ
コールバック関数
呼び出し
can_interrupt_error_callback()
end

4.4. 受信バッファ割り込み

－関数名－

RX: Intr_CANFD_RMRI
RA: intr_can***n***_rxmb
(***n*** = CAN-ch 番号)

－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_rxbuf_callback()

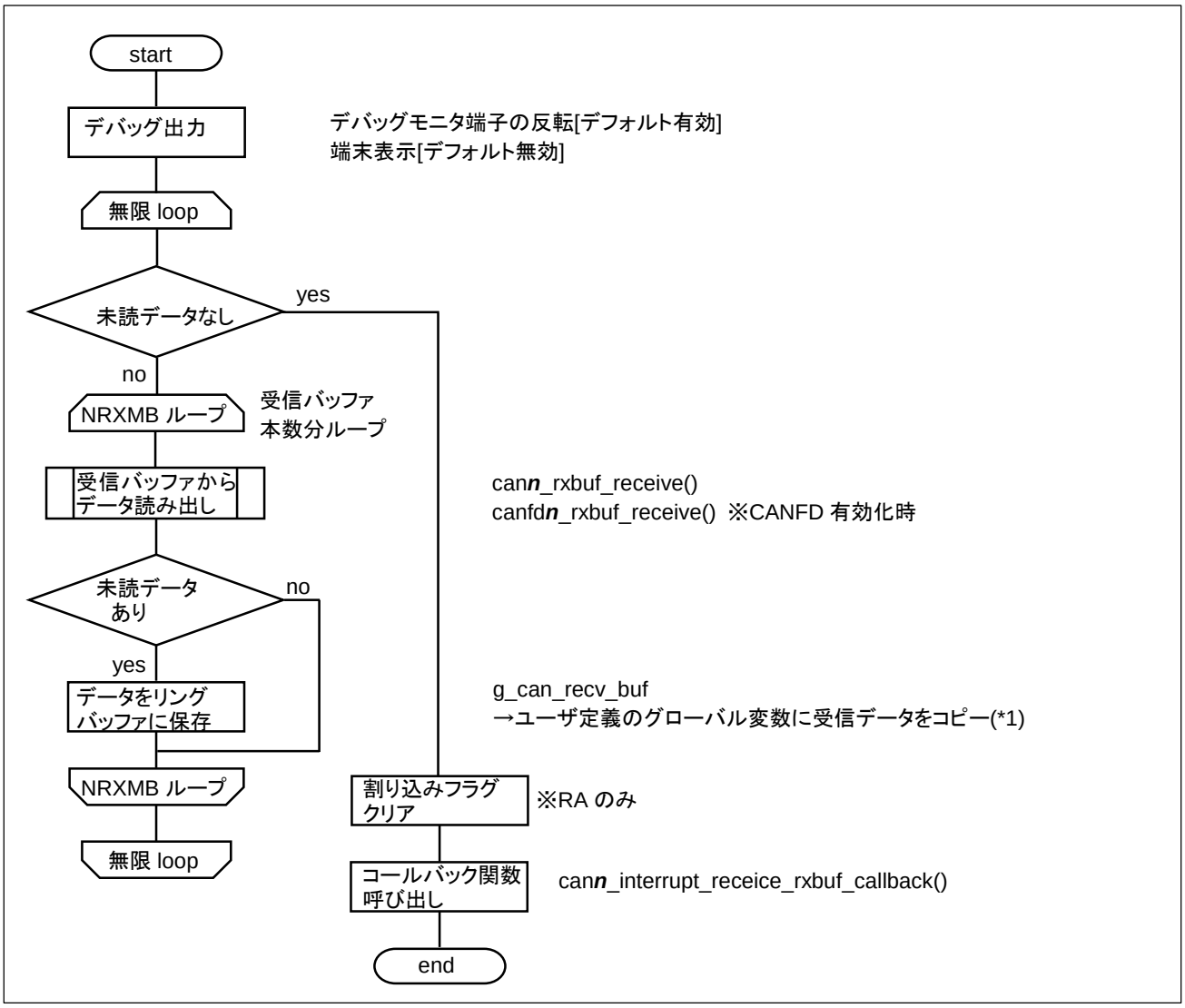
－コールバック関数(Intr_CANFD_RMRI, intr_can0_rxmb)－

割り込みコールバック関数	CAN-ch	備考
can0_interrupt_receive_rxbuf_callback()	0	

－コールバック関数(intr_can1_rxmb)－

割り込みコールバック関数	CAN-ch	備考
can1_interrupt_receive_rxbuf_callback()	1	

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]
無限 loop
未読データなし
yes
no
NRXMB ループ
受信バッファ
本数分ループ
受信バッファから
データ読み出し
can*n*_rxbuf_receive()
canfd*n*_rxbuf_receive() ※CANFD 有効化時
未読データ
あり
no
yes
データをリング
バッファに保存
g_can_recv_buf
→ユーザ定義のグローバル変数に受信データをコピー(*1)
NRXMB ループ
無限 loop
割り込みフラグ
クリア
※RA のみ
コールバック関数
呼び出し
can*n*_interrupt_receice_rxbuf_callback()
end

4.5. 送信割り込み

－関数名－

RX: Intr_CANFD0_CHTI
RA: intr_can***n***_tx

－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_callback()

－コールバック関数(Intr_CANFD0_CHTI, intr_can0_tx)－

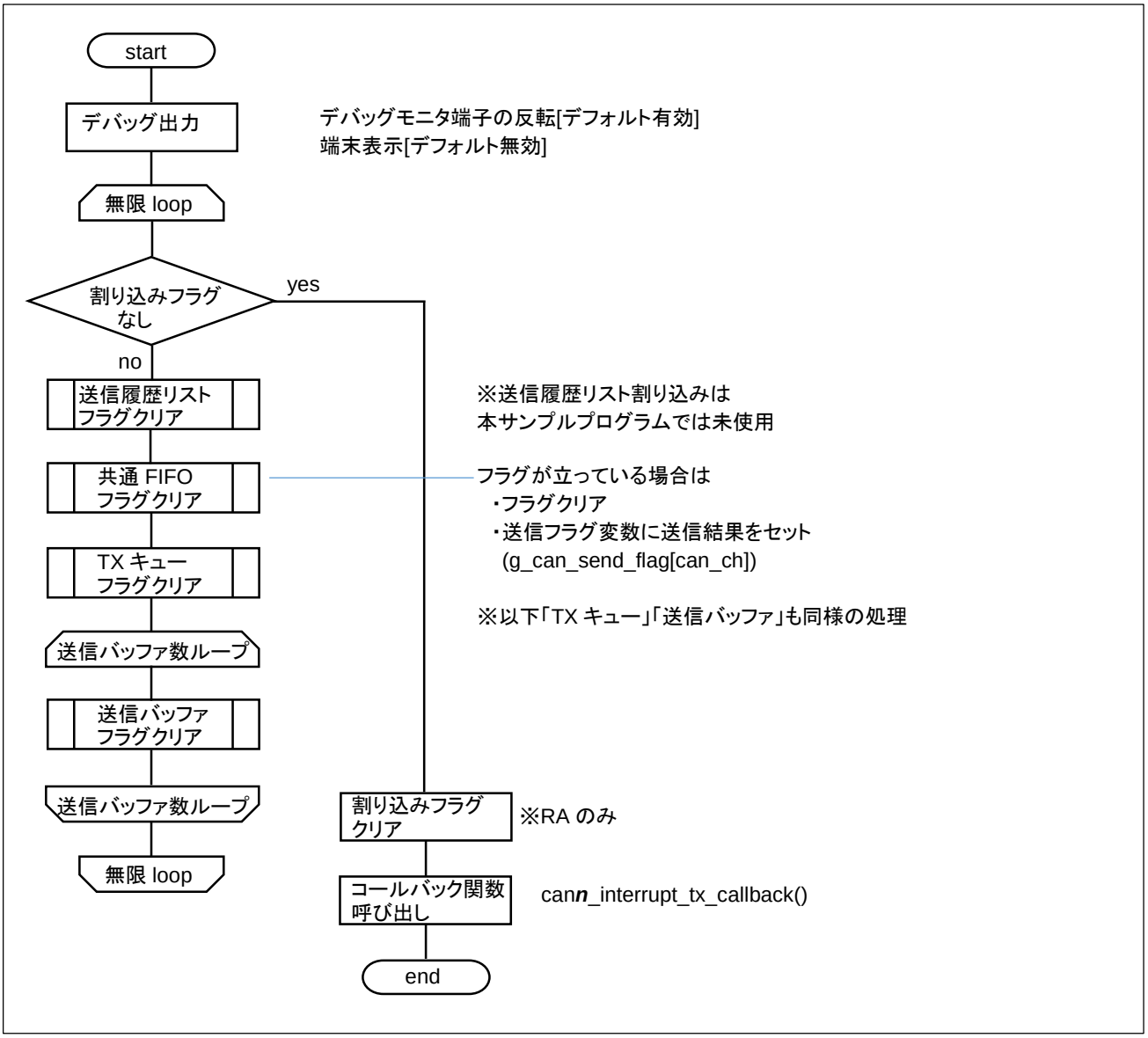
割り込みコールバック関数	備考
can0_interrupt_tx_callback()	

－コールバック関数(intr_can1_tx)－

割り込みコールバック関数	備考
can1_interrupt_tx_callback()	

※CANFD モジュールでは、CAN の物理 ch とは無関係に、CFIFO(0)~CFIFO(2)での送信時、intr_can0_tx()の割り込み関数、CFIFO(3)~CFIFO(5)の送信時 intr_can1_tx()の割り込み関数が呼ばれます。

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]
無限 loop
割り込みフラグ
なし
yes
no
送信履歴リスト
フラグクリア
※送信履歴リスト割り込みは
本サンプルプログラムでは未使用
共通 FIFO
フラグクリア
フラグが立っている場合は
・フラグクリア
・送信フラグ変数に送信結果をセット
(g_can_send_flag[can_ch])
TX キュー
フラグクリア
※以下「TX キュー」「送信バッファ」も同様の処理
送信バッファ数ループ
送信バッファ
フラグクリア
送信バッファ数ループ
割り込みフラグ
クリア
※RA のみ
無限 loop
コールバック関数
呼び出し
can*n*_interrupt_tx_callback()
end

4.6. チャネルエラー割り込み

－エラー割り込み対象－
・トランシーバ遅延補償（CANFD 有効時のみ）
・通信完了カウンタオーバフロー(*1)
・通信エラーカウンタオーバフロー
・送信アボート
・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

(*1)本来エラーとすべき事象ではありません。通信成功の度に、通信完了カウンタがインクリメントされ、255 を超えるとチャネルエラー割り込みが生じます。can_operation.h 内で、
#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE
を有効とすると、（デフォルトはコメントアウト）、通信完了カウンタオーバフローは無効化されます

－関数名－
RX: Intr_CANFD0_CHEI
RA: intr_can***n***_cherr

－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。）

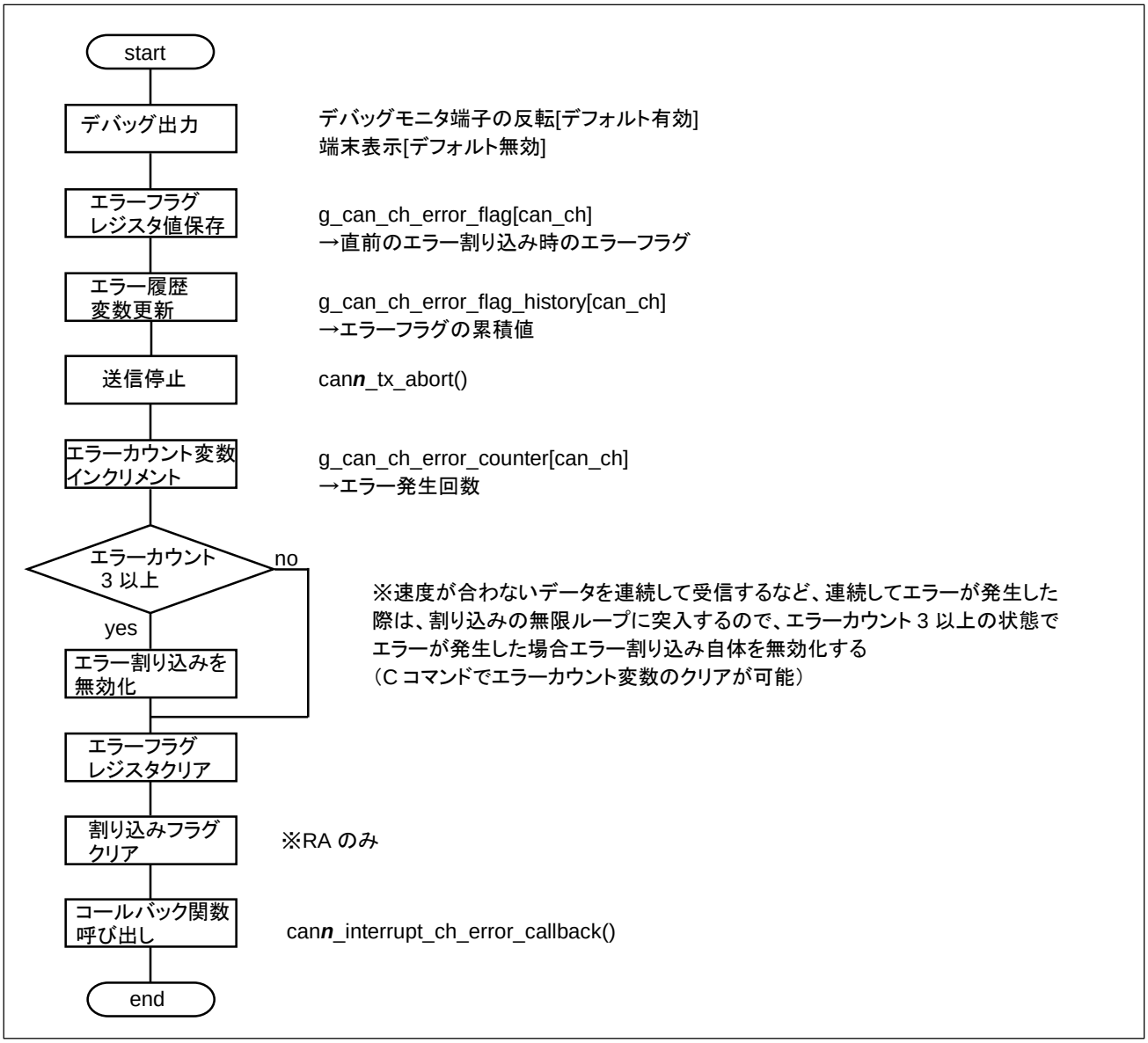
ーコールバック関数(Intr_CANFD0_CHEI, intr_can0_cherr)ー

割り込みコールバック関数	CAN-ch	備考
can0_interrupt_ch_error_callback()	0	

ーコールバック関数(intr_can1_cherr)ー

割り込みコールバック関数	CAN-ch	備考
can1_interrupt_ch_error_callback()	1	

ーフローチャートー



4.7. 共通 FIFO 受信割り込み

－関数名－

RX: Intr_CANFD0_CGRI
RA: intr_can***n***_comfrx

－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_cfifo***m***_callback()

（***n***: CANFD_2 モジュールのみ CAN-ch、他のモジュールでは n はなし(can_interrupt...)）

（***m***: CXFIFO(m), CFIFO 番号）

－コールバック関数(Intr_CANFD0_CGRI, intr_can0_comfrx)（CANFD_B, CANFD_Lite モジュール）－

割り込みコールバック関数	受信先	備考
can0_interrupt_receive_cfifo0_callback()	CFIFO(0)で受信	

－コールバック関数(intr_can0_comfrx)（CANFD モジュール）－

割り込みコールバック関数	受信先	備考
can0_interrupt_receive_cfifo0_callback()	CFIFO(0)で受信	
can0_interrupt_receive_cfifo1_callback()	CFIFO(1)で受信	
can0_interrupt_receive_cfifo2_callback()	CFIFO(2)で受信	

－コールバック関数(intr_can1_comfrx)（CANFD モジュール）－

割り込みコールバック関数	受信先	備考
can1_interrupt_receive_cfifo3_callback()	CFIFO(3)で受信	
can1_interrupt_receive_cfifo4_callback()	CFIFO(4)で受信	
can1_interrupt_receive_cfifo5_callback()	CFIFO(5)で受信	

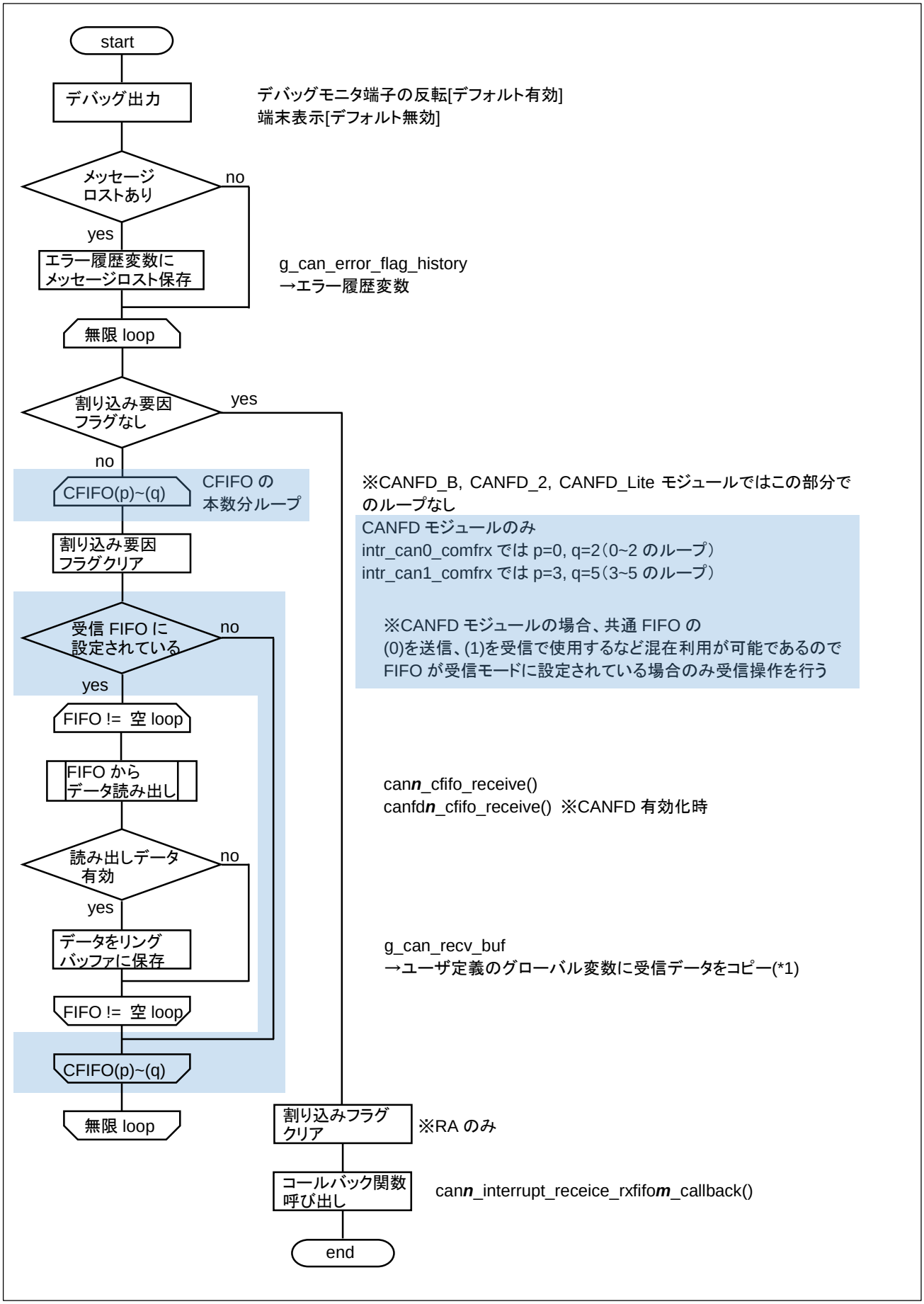
※CANFD モジュールの場合は、CAN-ch0, CAN-ch1 と intr_can0_comfrx, intr_can1_comfrx は関連がありません（CAN-ch0 の受信ルールで受信先を CFIFO(3)に設定した場合、CAN-ch0 でデータ受信時 intr_can1_comfrx() → can1_interrupt_receive_cfifo3_callback() が呼ばれます。）

－コールバック関数(intr_can0_comfrx)（CANFD_2 モジュール）－

割り込みコールバック関数	CAN-ch	受信先	備考
can0_interrupt_receive_cfifo0_callback()	0	CFIFO(0)で受信	

－コールバック関数(intr_can1_comfrx)（CANFD_2 モジュール）－

割り込みコールバック関数	CAN-ch	受信先	備考
can1_interrupt_receive_cfifo0_callback()	1	CFIFO(0)で受信	


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]
メッセージ
ロストあり
no
yes
エラー履歴変数に
メッセージロスト保存
g_can_error_flag_history
→エラー履歴変数
無限 loop
割り込み要因
フラグなし
yes
no
CFIFO(p)~(q)
CFIFO の
本数分ループ
※CANFD_B, CANFD_2, CANFD_Lite モジュールではこの部分でのループなし
CANFD モジュールのみ
intr_can0_comfrx では p=0, q=2（0~2 のループ）
intr_can1_comfrx では p=3, q=5（3~5 のループ）
※CANFD モジュールの場合、共通 FIFO の
(0)を送信、(1)を受信で使用するなど混在利用が可能であるので
FIFO が受信モードに設定されている場合のみ受信操作を行う
割り込み要因
フラグクリア
受信 FIFO に
設定されている
no
yes
FIFO != 空 loop
FIFO から
データ読み出し
can*n*_cfifo_receive()
canfd*n*_cfifo_receive() ※CANFD 有効化時
読み出しデータ
有効
no
yes
データをリング
バッファに保存
g_can_recv_buf
→ユーザ定義のグローバル変数に受信データをコピー(*1)
FIFO != 空 loop
CFIFO(p)~(q)
無限 loop
割り込みフラグ
クリア
※RA のみ
コールバック関数
呼び出し
can*n*_interrupt_receice_rxfifo*m*_callback()
end

(*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
g_can_recv_buf[CAN_CH1]　:CAN-ch1 で受信したデータを格納
という構成です。

－CANFD_B, CANFD_Lite モジュール－

受信元	格納先
CFIFO(0)	g_can_recv_buf[CAN_CH0]

－CANFD モジュール－

受信元	格納先(*1)
CFIFO(0)	
CFIFO(1)	g_can_recv_buf[CAN_CH0]
CFIFO(2)	

受信元	格納先(*2)
CFIFO(3)	
CFIFO(4)	g_can_recv_buf[CAN_CH1]
CFIFO(5)	

格納先が空白のところは、受信処理のみ行いリングバッファへのコピーは行いません。（動作を変更する場合は、canfd_ch0_intr.c, canfd_ch1_intr.c を編集、または下記定数を変更してください。）

(*1)#define CAN0_RX_CFIFO_NO　1　（デフォルト値）の場合　（本定数は 0~2 の値が有効です）
(*2)#define CAN1_RX_CFIFO_NO　4　（デフォルト値）の場合　（本定数は 3~5 の値が有効です）
（これらの定数は、can_operation.h 内で定義されています）
※CFIFO(0), CFIFO(3)は送信向けに割り当てているので、受信では CFIFO(1)と CFIFO(4)を使っています。
※CAN-ch0 の処理を CFIFO(0)~CFIFO(2)、CAN-ch1 の処理を CFIFO(3)~CFIFO(5)で行うという思想でプログラムを構成していますが、マイコンの機能としてはその様な制約はなく、CAN の物理 ch と CFIFO 番号の組み合わせは自由です。

－CANFD_B, CANFD_Lite モジュール－

受信元	格納先
CAN-ch0, CFIFO(0)	g_can_recv_buf[CAN_CH0]
CAN-ch1, CFIFO(0)	g_can_recv_buf[CAN_CH1]

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
・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

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

・CAN-ch0

受信ルール番号	フォーマット	ID	データ格納先 受信メッセージ 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

・CAN-ch1

受信ルール番号	フォーマット	ID	データ格納先 受信メッセージ バッファ番号 [CANFD モジュール]	データ格納先 受信メッセージ バッファ番号 [CANFD_2 モジュール]
0	標準(SID)	0x000	16	0
1	標準(SID)	0x001	17	1
2	標準(SID)	0x002	18	2
3	標準(SID)	0x003	19	3
4	拡張(EID)	0x0000000	20	4
5	拡張(EID)	0x0000001	21	5
6	拡張(EID)	0x0000002	22	6
7	拡張(EID)	0x0000003	23	7

CANFD 系モジュールでは、アクセプタンスフィルタリスト(AFL)で、ルールにマッチした（ID 等が一致した）データの格納先を設定する形です。本サンプルプログラム(SAMPLE1)では、データの格納先を受信メッセージバッファに割り当てています。

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(); //CANFD 系モジュールのリセット[CANFD, CANFD_B, CANFD_Lite モジュール]
can0_reset(); //CANFD_2 モジュールのリセット[CANFD_2 モジュール](*1)
can1_reset(); //CANFD_2 モジュールのリセット[CANFD_2 モジュール](*1)
can_init(); //CANFD 系モジュールの初期化
can0_init(); //CAN-ch0 の初期化（CAN-ch0 使用時）(*1)
can1_init(); //CAN-ch1 の初期化（CAN-ch1 使用時）(*1)
can_receive_buf_conf(); //受信メッセージバッファの設定[CANFD, CANFD_B, CANFD_Lite モジュール]
can0_receive_buf_conf(); //受信メッセージバッファの設定[CANFD_2 モジュール](*1)
can1_receive_buf_conf(); //受信メッセージバッファの設定[CANFD_2 モジュール](*1)

CAN の初期化をする。
(*1)どちらか先でも構いません、使用する ch のみ実行要

(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
can*n*_receive_rule_set(0, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
can*n*_receive_rule_set(1, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000001);
…
can*n*_receive_rule_set(4, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
…
can*n*_receive_rule_set(7, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000003);

n=0 or 1（CAN-ch に応じた値）

(3)動作モードへの移行

can_operate(); //全体の動作設定[CANFD, CANFD_B, CANFD_Lite モジュール]
can0_operate(); //CAN-ch0 の動作設定(*2)
can1_operate(); //CAN-ch1 の動作設定(*2)

受信ルール番号 0~7 を受信設定する例。受信ルール設定後、can_operate, can*n*_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

(*2)どちらが先でも構いません（使用する ch のみ実行）

(4)データの受信

```
for(i=0; i<=7; 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 = can n_rxbuf_receive((unsigned char)i, &r_msg);
```

受信バッファ番号：SAMPLE1 では can***n***_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。

※CAN-ch0, CAN-ch1 の両方の受信処理を行いたい場合は、can0_buf_receive(), can1_buf_receive()を呼び出してください。

※CANFD モジュールでは
can1_buf_receive(0, &r_msg);
で受信バッファ番号 16 のデータを読み出します。
can0_buf_receive()は受信バッファ 0~15 から、can1_buf_receive()は受信バッファ 16~31 から読み出しを行います。

(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

CAN-ch0 で、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。

※CAN-ch1 から送信する場合は、can1_txbuf_send()

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。

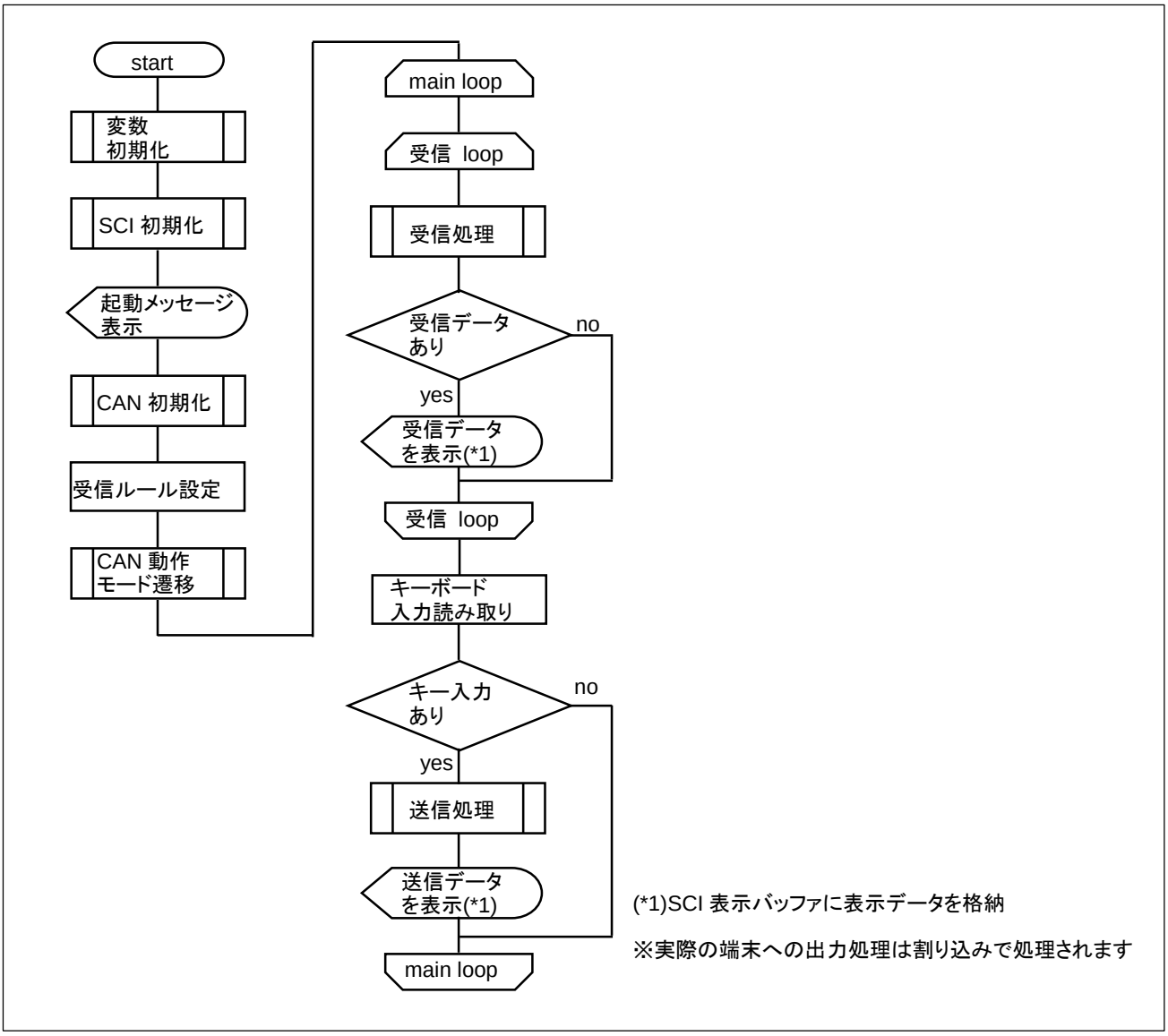
5.4. データの受信に関して

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「送受信 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)

・CAN-ch1

受信ルール番号	フォーマット	ID	データ格納先 [CANFD モジュール]	データ格納先 [CANFD_2 モジュール]
0	標準(SID)	0x000	RXFIFO(1)	RXFIFO(0)
1	標準(SID)	0x001	RXFIFO(1)	RXFIFO(0)
2	標準(SID)	0x002	RXFIFO(1)	RXFIFO(0)
3	標準(SID)	0x003	RXFIFO(1)	RXFIFO(0)
4	拡張(EID)	0x0000000	RXFIFO(1)	RXFIFO(0)
5	拡張(EID)	0x0000001	RXFIFO(1)	RXFIFO(0)
6	拡張(EID)	0x0000002	RXFIFO(1)	RXFIFO(0)
7	拡張(EID)	0x0000003	RXFIFO(1)	RXFIFO(0)

CANFD モジュールでは、RXFIFO は全 ch で共通。CANFD_2 モジュールの場合は、ch 毎に独立です。
（CANFD_B, CANFD_Lite モジュールは、CAN-ch1 なし）

6.3. 送信設定

送信は共通 FIFO を送信で使用する設定です。

・CAN-ch0
CFIFO(0)

・CAN-ch1
CFIFO(3)　[CANFD モジュール]
CFIFO(0)　[CANFD_2 モジュール]

CANFD モジュールでは、CFIFO(0)~CFIFO(2)を CAN-ch0 で使用、CFIFO(3)~CFIFO(5)を CAN-ch1 で使用という形を本サンプルプログラムでは採用しています（6 本の CFIFO を全 ch 共有で使用する仕様）。CANFD_2 モジュールでは、そもそも CFIFO(0)のみ使用可能、ch 毎に CFIFO は独立となっているので、CAN-ch0, CAN0-ch1 共 CFIFO(0)を使う設定です。

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)
can_ch_error_interrupt_enable(1);　//チャネルエラー割り込み有効化(CAN-ch1)

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

can1_receive_rule_set(0, **CAN_RULE_RXFIFO1**, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000); //CAN-ch1 **[CANFD モジュール]**

can1_receive_rule_set(0, **CAN_RULE_RXFIFO0**, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000); //CAN-ch1 **[CANFD_2 モジュール]**

…

受信ルール設定時、受信バッファではなく CAN-ch0：RXFIFO(0), CAN-ch1：RXFIFO(x)を使うように設定。
（CANFD モジュール x=1, CANFD_2 モジュール x=0）

(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_cfifo_send**(0, &s_msg); //CAN-ch0

s_ret = **can1_cfifo_send**(3, &s_msg); //CAN-ch1 **[CANFD モジュール]**
s_ret = **can1_cfifo_send**(0, &s_msg); //CAN-ch1 **[CANFD_2 モジュール]**

CAN-ch1 の送信時は、CANFD と CANFD_2 で使用する FIFO 番号が違いますが、使用する関数は同じです。
※CANFD_B, CANFD_2, CANFD_Lite モジュールでは、常に FIFO 番号 0 で使用する事となります
（CFIFO 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

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

n=0,1 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_interrupt_receive_rxfifo0_callback();　//RXFIFO(0)で受信(CAN-ch0)[CANFD, CANFD_B, CANFD_Lite モジュール]
can_interrupt_receive_rxfifo1_callback();　//RXFIFO(1)で受信(CAN-ch1)[CANFD モジュール]
can0_interrupt_receive_rxfifo0_callback();　//RXFIFO(0)で受信(CAN-ch0)[CANFD_2 モジュール]
can1_interrupt_receive_rxfifo0_callback();　//RXFIFO(0)で受信(CAN-ch1)[CANFD_2 モジュール]

どの受信コールバック関数からでも、

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。

※受信に共通 FIFO、受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。
(受信バッファの受信割り込みは、CANFD モジュールでは使えません)

(3)送信割り込み

can***n***_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.(CFIFO)
を表示する処理を行います。

※送信に、送信キュー、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。

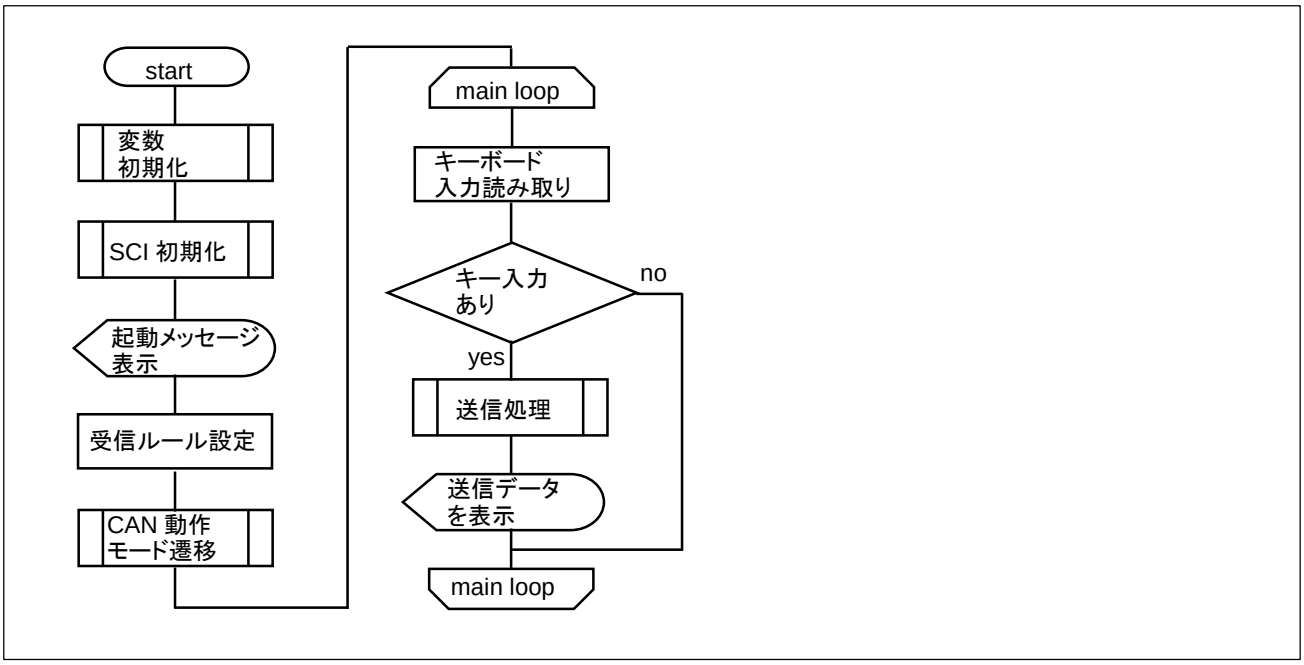
受信割り込みは、「受信 FIFO」「共通 FIFO（受信）」「受信バッファ」で、割り込み関数が別となります。
（受信バッファ割り込みは CANFD モジュール以外で使用可能）

それに対し、送信割り込みは、「共通 FIFO（送信）」「送信キュー」「送信バッファ」いずれの送信方式でも、同じ割り込み（RX:Intr_CANFD0_CHTI(), RA: intr_can***n***_tx()）関数が呼ばれます。
（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。）

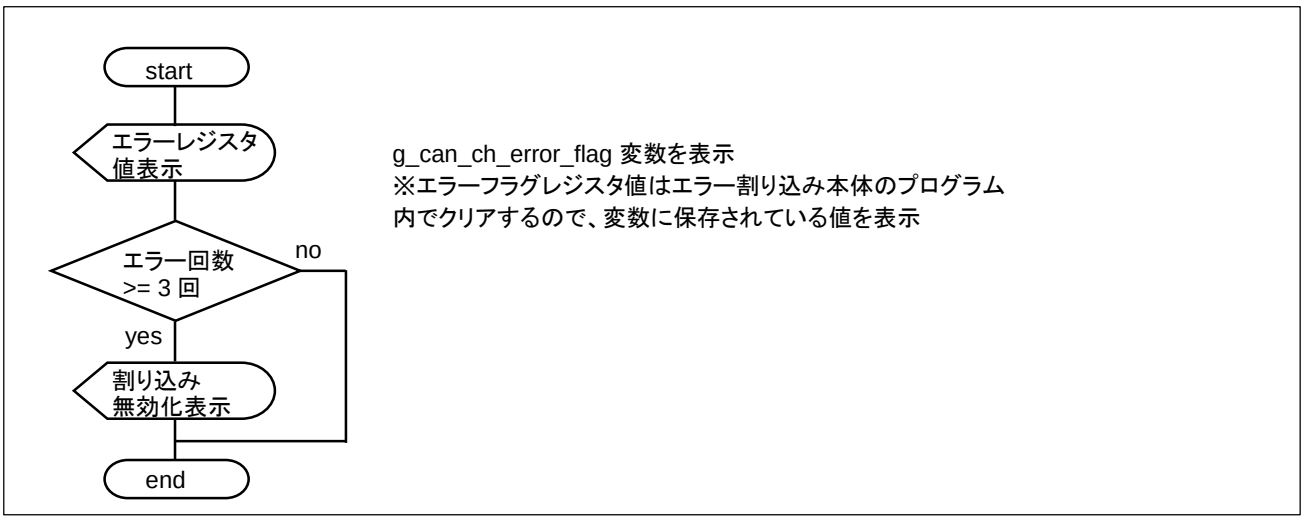
6.6. SAMPLE2 フローチャート

－処理フロー－

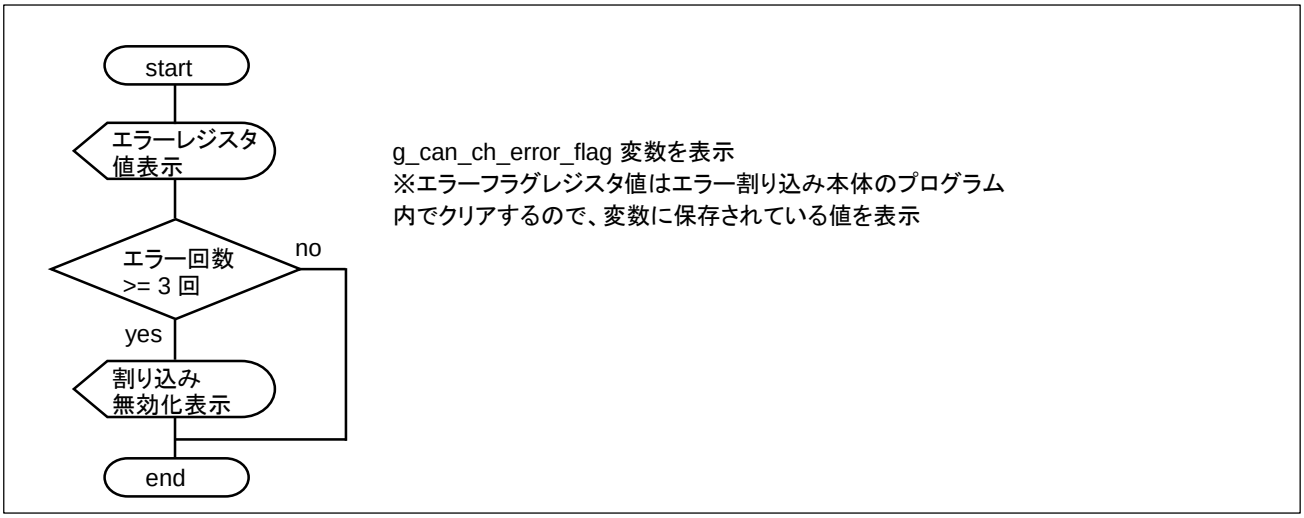
メイン関数　main_s2()



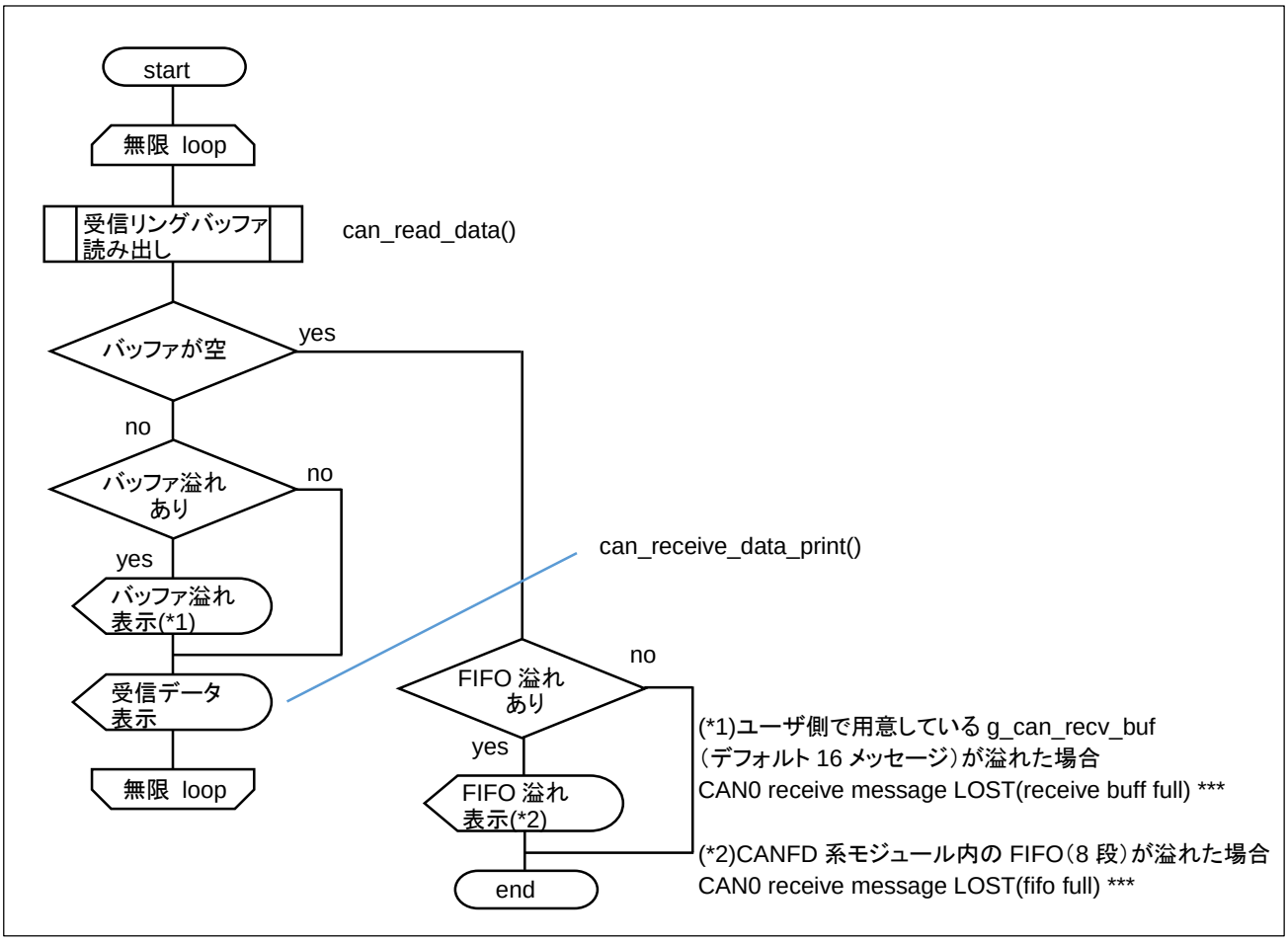
グローバルエラー割り込みコールバック関数　can_interrupt_error_callback()



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()



－CANFD モジュール－

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_can_rxf()	can_interrupt_receive_rxfifo0_callback()	can_interrupt_rx_callback()	RXFIFO(0)で受信
	can_interrupt_receive_rxfifo1_callback()	can_interrupt_rx_callback()	RXFIFO(1)で受信
	can_interrupt_receive_rxfifo2_callback()		RXFIFO(2)で受信
	can_interrupt_receive_rxfifo3_callback()		RXFIFO(3)で受信
	can_interrupt_receive_rxfifo4_callback()		RXFIFO(4)で受信
	can_interrupt_receive_rxfifo5_callback()		RXFIFO(5)で受信
	can_interrupt_receive_rxfifo6_callback()		RXFIFO(6)で受信
	can_interrupt_receive_rxfifo7_callback()		RXFIFO(7)で受信
intr_can0_comfrx() (*4)	can0_interrupt_receive_cfifo0_callback()		CFIFO(0)で受信
	can0_interrupt_receive_cfifo1_callback()	can_interrupt_rx_callback()	CFIFO(1)で受信
	can0_interrupt_receive_cfifo2_callback()		CFIFO(2)で受信
intr_can1_comfrx()(*4)	can1_interrupt_receive_cfifo3_callback()		CFIFO(3)で受信
	can1_interrupt_receive_cfifo4_callback()	can_interrupt_rx_callback()	CFIFO(4)で受信
	can1_interrupt_receive_cfifo5_callback()		CFIFO(5)で受信

(*1)割り込み関数は
canfd_intr.c
canfd_intr_ch0.c
canfd_intr_ch1.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。
(*4)CAN の物理 ch（CAN-ch0, CAN-ch1）とは無関係です。CFIFO(0)~CFIFO(2)を can0~, CFIFO(3)~CFIFO(5)を can1~という関数名としています。

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。
CANFD モジュールでは、RXFIFO(0)で CAN-ch0, RXFIFO(1)で CAN-ch1 のデータを受信する様に設定しているので、これらの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。

受信に共通 FIFO を使用する様に変更した場合（can_operation2.h で変更可）、CFIFO(1)で CAN-ch0, CFIFO(4)で CAN-ch1 のデータを受信する様に設定しているので、これらのコールバック関数から最終的に受信データを表示する子関数を呼び出しています。
※CFIFO(0)は CAN-ch0 の送信用、CFIFO(3)は CAN-ch1 の送信用に設定しているので、受信は CFIFO(1), CFIFO(4)を使う設定です

(*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。）

※CANFD モジュールの受信先設定は、全 ch 共通なので、受信ルール設定時にどの CAN-ch の受信ルールをどの 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(4)で受信して、ID=0x31 のデータは、RXFIFO(5)で受信するような設定も問題ありません。
（RXFIFO と CFIFO（受信）を同時に使用する場合は、ソースコードの変更が必要です。）

－CANFD_B モジュール－

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_can_rxf()	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_can0_comfrx()	can0_interrupt_receive_cfifo0_callback()	can_interrupt_rx_callback()	CFIFO(0)で受信
intr_can0_rxmb()	can0_interrupt_receive_rxbuf_callback()	can_interrupt_rx_callback()	受信バッファで受信

CANFD_B モジュールでは、RXFIFO(1)は未使用。他の受信先の場合は、受信データ表示の子関数を呼び出します。

－CANFD_2 モジュール－

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_can_rxf()	can0_interrupt_receive_rxfifo0_callback()	can_interrupt_rx_callback()	RXFIFO(0)で受信(*1)
	can0_interrupt_receive_rxfifo1_callback()		RXFIFO(1)で受信(*1)
	can1_interrupt_receive_rxfifo0_callback()	can_interrupt_rx_callback()	RXFIFO(0)で受信(*2)
	can1_interrupt_receive_rxfifo1_callback()		RXFIFO(1)で受信(*2)
intr_can0_comfrx()	can0_interrupt_receive_cfifo0_callback()	can_interrupt_rx_callback()	CFIFO(0)で受信(*1)
intr_can1_comfrx()	can1_interrupt_receive_cfifo0_callback()	can_interrupt_rx_callback()	CFIFO(0)で受信(*2)
intr_can0_rxmb()	can0_interrupt_receive_rxbuf_callback()	can_interrupt_rx_callback()	受信バッファで受信(*1)
intr_can1_rxmb()	can1_interrupt_receive_rxbuf_callback()	can_interrupt_rx_callback()	受信バッファで受信(*2)

(*1)CAN-ch0 で受信

(*2)CAN-ch1 で受信

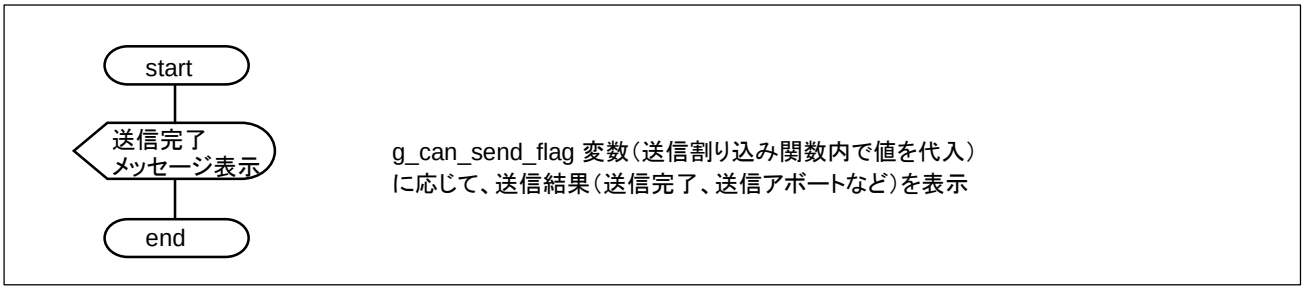
CANFD_2 モジュールの場合は、RXFIFO(1)は未使用。他の受信先の場合は、受信データ表示の子関数を呼び出します。

－CANFD_Lite モジュール－

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_CANFD_RFRI()	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_CANFD0_CFRI()	can0_interrupt_receive_cfifo0_callback()	can_interrupt_rx_callback()	CFIFO(0)で受信
Intr_CANFD_RMRI()	can0_interrupt_receive_rxbuf_callback()	can_interrupt_rx_callback()	受信バッファで受信

CANFD_Lite モジュールは、割り込み関数の関数名が変わりますが、CANFD_B モジュールと同一の構成です。

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_can0_tx()	can0_interrupt_tx_callback()	can_interrupt_tx_callback()	CAN-ch0 で送信
intr_can1_tx()	can1_interrupt_tx_callback()	can_interrupt_tx_callback()	CAN-ch1 で送信

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

・CAN-ch0

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)

・CAN-ch1

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 [CANFD モジュール]	データ格納先 [CANFD_2 モジュール]
0	標準(SID)	データフレーム	0x000	RXFIFO(1)	RXFIFO(0)
1	標準(SID)	データフレーム	0x001	RXFIFO(1)	RXFIFO(0)
2	標準(SID)	データフレーム	0x002	RXFIFO(1)	RXFIFO(0)
3	標準(SID)	データフレーム	0x003	RXFIFO(1)	RXFIFO(0)
4	標準(SID)	リモートフレーム	0x000	RXFIFO(1)	RXFIFO(0)
5	標準(SID)	リモートフレーム	0x001	RXFIFO(1)	RXFIFO(0)
6	標準(SID)	リモートフレーム	0x002	RXFIFO(1)	RXFIFO(0)
7	標準(SID)	リモートフレーム	0x003	RXFIFO(1)	RXFIFO(0)
8	拡張(EID)	データフレーム	0x0000000	RXFIFO(1)	RXFIFO(0)
9	拡張(EID)	データフレーム	0x0000001	RXFIFO(1)	RXFIFO(0)
10	拡張(EID)	データフレーム	0x0000002	RXFIFO(1)	RXFIFO(0)
11	拡張(EID)	データフレーム	0x0000003	RXFIFO(1)	RXFIFO(0)
12	拡張(EID)	リモートフレーム	0x0000000	RXFIFO(1)	RXFIFO(0)
13	拡張(EID)	リモートフレーム	0x0000001	RXFIFO(1)	RXFIFO(0)
14	拡張(EID)	リモートフレーム	0x0000002	RXFIFO(1)	RXFIFO(0)
15	拡張(EID)	リモートフレーム	0x0000003	RXFIFO(1)	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(can***n***_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_cfifo_send(0, &s_msg); //CAN-ch0
```

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);　//CAN-ch0 の割り込みの場合

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

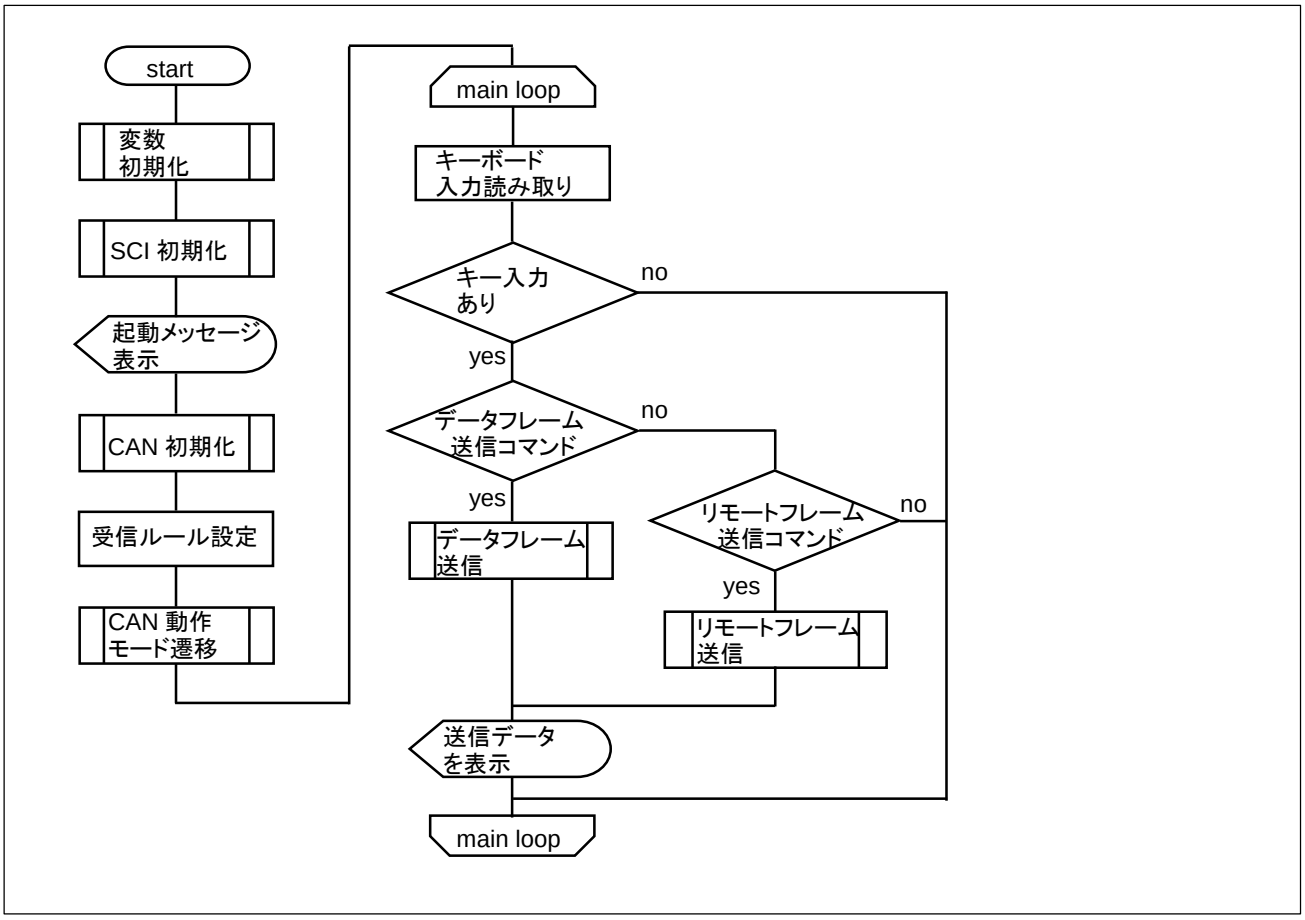
－SAMPLE3－
・受信データの表示
・受信データがリモートフレームの場合データフレームの送信

(4)チャネルエラー割り込み

[SAMPLE2 と同様]

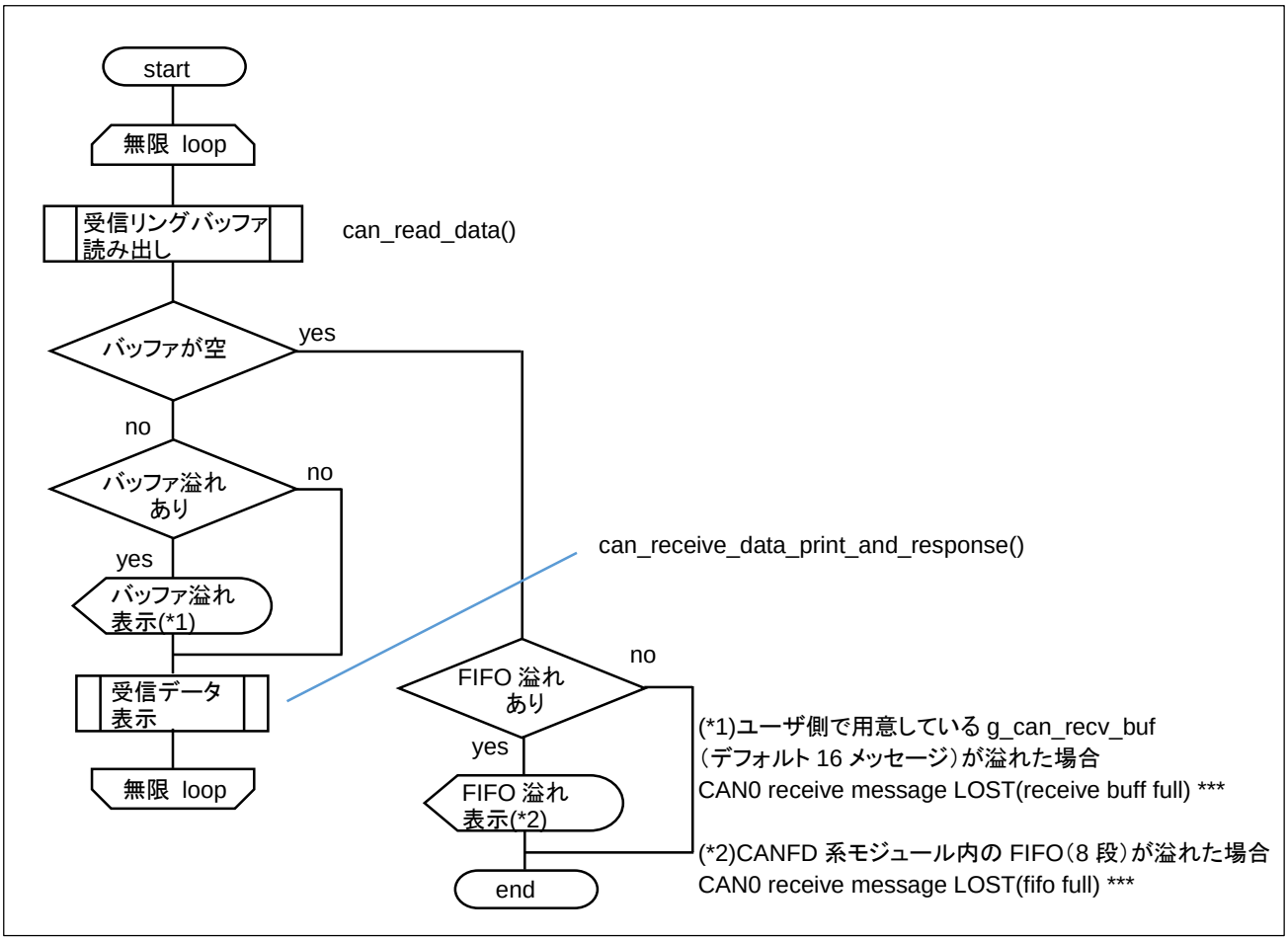
7.6. SAMPLE3 フローチャート

メイン関数　main_s3()

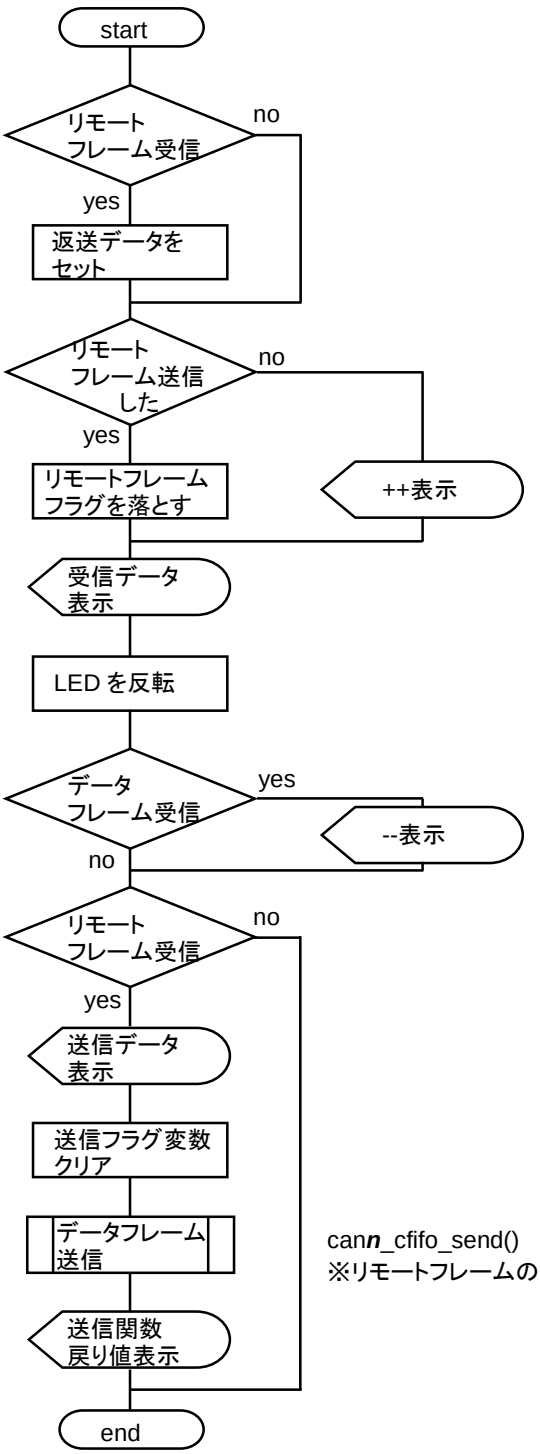


※エラー割り込み、送信割り込みコールバック関数から呼ばれる関数は SAMPLE2 から変更なし

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）

can***n***_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の説明(SAMPLE4)

8.1. プログラム仕様

・データフレーム（CANFD フレーム）の送信
・データフレーム（CANFD 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（CANFD フレーム）を行う）する

を行うサンプルプログラムとします。動作としては、SAMPLE3 と同様ですが、赤字の部分を CANFD フレームを使って送受信します。（リモートフレームの送信のみ、従来の CAN フレームでの通信となります。…CANFD の仕様）

－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

キーボードからの入力	ID	送信バイト数(DLC)
0	0x0000000	1
1	0x0000001	8
2	0x0000002	16
3	0x0000003	64

送信データは、0x00, 0x01, 0x02, 0x03.....（0 からインクリメントしたデータ）で、最大 64 バイトです。

・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	8
e	0x0000002	16
r	0x0000003	64

・リモートフレーム応答
0xFF, 0xFE, 0xFD, 0xFC.....（0xFF からデクリメントしたデータ）
を、要求元の送信要求バイト数に応じて返答
（DLC=8 のときは、0xFF FE FD FC FB FA F9 F8 を返す）

CANFD では、データのパケットサイズは最大 64 バイトに拡張されています。SAMPLE4 では、最大 64 バイトのデータを取り扱う様にしています。

SAMPLE4 では、起動時に CANFD の通信速度を聞かれます。リターンキーで応答するとデフォルトの 2Mbps に設定されます。
※main_s4.c 内の `#define CANFD_PARAMETER_SETTING` の行をコメントアウトした場合、起動時のパラメータの対話入力は省略され、can_operation.h 内で定義された速度他のパラメータが有効になります

```
CANFD setting:

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

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

SAMPLE4 のプログラムを起動すると、端末に CANFD の速度の入力が促されます。数値を入力することも可能ですが、とりあえずリターンで先へ進めても問題ありません。

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

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

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

Input summary:

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

リーターンを押すとデフォルト値

・CANFD 通信速度　2Mbps

・レシーバ遅延補償　なし

・RX エッジフィルタ　未使用

の設定となります。

※CANFD 使用時は、「CANFD のビットレート」と「CAN のビットレート」の 2 種類の通信速度が存在します。ココで設定しているのは、CANFD のビットレートの方です。CAN のビットレートは、can_operation.h 内で定義されています。（サンプルプログラムの CAN ビットレートのデフォルト値は、1Mbps）

CANFD のビットレートは、CANFDCLK の周波数によって設定できる値が決まってきます。例えば CANFDCLK = 60MHz の場合、8Mbps には設定不可となります。（クロック周波数とビットレートの関係は、CAN スタータキットの取扱説明書に記載があります。）

設定不可のビットレートを入力した場合は

```
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])

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

コマンド一覧表示の後で、上記表示が出ますので、その場合はリセットして別な速度を入力してください。

（マイコンのクロック周波数を変更しなければ、どのマイコンでもデフォルト値の 2Mbps には設定可能です。）

8.2. 受信ルール設定

受信ルールの設定は、SAMPLE3 に同じ。

CANFD 未使用の場合は、メッセージのデータ部分は最大 8 バイトです。CANFD メッセージの場合は、最大 64 バイトとなります。

CANFD_B, CANFD_2, CANFD-Lite モジュールでは、バッファメモリの制約が厳しく、用意されているバッファメモリは 1 メッセージのデータサイズを 64 バイトとすると、16 メッセージ分です。

受信メッセージバッファを 16 個確保すると、受信メッセージバッファのみで全てのバッファメモリを使い切るので、SAMPLE4 では受信メッセージバッファを
・送信に共通 FIFO を使用した場合　8 個
・送信を送信バッファまたは送信キューを使う場合　16 個
としています。

・CANFD モードでのメッセージ数設定[CANFD_B, CANFD_2, CANFD-Lite モジュール]

受信方式	送信方式	消費メッセージサイズ				
		RXFIFO(0)	RXFIFO(1)	CFIFO(0)	受信バッファ	合計
受信 FIFO	共通 FIFO	8	0	8	0	16
	送信バッファ/送信キュー	8	0	0	0	8
共通 FIFO	送信バッファ/送信キュー	0	0	8	0	8
受信バッファ	共通 FIFO	0	0	8	8	16
	送信バッファ/送信キュー	0	0	0	16	16

※合計が 16 以下である必要があります（データサイズ 64 バイトの場合）

FIFO 段数の設定と受信メッセージバッファ数の設定を調整して、メモリ容量をオーバしないようにしてください。

※CANFD フレームを取り扱う場合でも、データサイズを 64 バイトではなくもっと小さな値に制限した場合、16 を超えるメッセージを取り扱う事も可能です（例えば 32 バイトにした場合は 32 メッセージまで）（その場合、データサイズの設定を超えるパケットは受信できません）

CANFD モジュールでは、メッセージ数はデータサイズ 64 バイトの場合でも 512 メッセージとなりますので、FIFO 段数の設定や受信メッセージバッファ数の設定に余裕があります。CANFD_B, CANFD_2, CANFD_Lite モジュールでは、意外に少ない容量ですので、注意が必要です。

（※メッセージ数の設定をオーバした場合、受信データで AFL テーブ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。）

8.3. 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。SAMPLE3 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]

(3)受信ルールの設定

[SAMPLE3 と同様]

(4)動作モードへの移行

[SAMPLE1 と同様]

(5)データの送信[SAMPLE3 との相違点]

canfd_message s_msg; //データを取り扱う構造体が CANFD 向けのものに変わる

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

//引数:       FIFO 番号   CAN メッセージ構造体
s_ret = canfd0_cfifo_send(0, &s_msg); //CAN-ch0  データフレームの送信
```

CANFD メッセージを送信する関数は canfd0_cfifo_send となり、関数が変わります。

従来の CAN メッセージには存在しない、fdf, brs というメンバが増えています。また、data も従来は data[0] ~ data[7]でしたが、CANFD メッセージ構造体では data[0] ~ data[63]となっています。

追加となっている FDF と BRS の項目ですが、FDF=1 の時は CANFD フォーマットであることを示しており、BRS=1 の時は、データフィールドを高速で通信する事を示しています。（CAN スタータキット取扱説明書の 2.5 章 CANFD のデータパケットの項を参照）

リモートフレーム送信の際は、

```
s_msg.id = 0x0000001;
s_msg.rtr = CAN_REMOTE_FRAME;
s_msg.ide = CAN_ID_FORMAT_EID;
s_msg.fdf = CAN_DATA_FORMAT;
s_msg.brs = CAN_BITRATE;
s_msg.dlc = 10;

//引数:       FIFO 番号   CAN メッセージ構造体
s_ret = canfd0_cfifo_send(0, &s_msg); //CAN-ch0  リモートフレームの送信
```

FDF=0(CAN_DATA_FORMAT), BRS=0(CAN_BITRATE)として、メッセージ送信してください。

CANFD 有効時は、DLC 値として 9~15 の値の指定が可能です。

DLC 値	データバイト数
9	12
10	16
11	20
12	24
13	32
14	48
15	64

データフレーム送信時は、DLC に応じたバイト数のデータを送信します。リモートフレーム送信時は、データパケット内に DLC の値が埋め込まれて送信されますので、データを受信した側が CANFD 有効であれば、DLC 値に応じたバイト数のデータを返送するはずです。

8.4. 割り込み処理

ここでは、main_s4.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

[SAMPLE2~3 と同様]

(2)チャネルエラー割り込み

[SAMPLE2~3 と同様]

(3)受信割り込み

コールバック関数から、呼ばれる can_interrupt_rx_callback()関数で扱う CAN メッセージ構造体が CANFD メッセージ構造体に変わります。

－SAMPLE3－
can_message msg;

－SAMPLE4－
canfd_message msg;

受信データの表示とリモートフレーム応答を行っている関数 can_receive_data_print_and_response()でも同様に canfd_messege 構造体でデータを処理しています。受信メッセージの表示時に、FDF, BRS などの情報を表示している点と、リモートフレーム応答時に、CANFD メッセージ送信関数を使っている点などは異なりますが、基本的には SAMPLE3 と変わりません。

(4)チャネルエラー割り込み

[SAMPLE2~3 と同様]

8.5. SAMPLE4 フローチャート

データ送信、受信の関数が CANFD 版に変わっているだけで、フローチャートとしては SAMPLE3 に同じ。

9. サンプルプログラムで使用している関数の説明

9.1. 関数仕様

can_reset
can*n*_reset　　(n=0~1)

概要:初期化関数

宣言:

int can_reset(void)　　[CANFD, CANFD_B, CANFD_Lite モジュール]
int can0_reset(void)　[ch0 向け]　[CANFD_2 モジュール]
int can1_reset(void)　[ch1 向け]　[CANFD_2 モジュール]

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

補足:

CANFD, CANFD_B, CANFD_Lite モジュールでは、can_reset()を実行
CANFD_2 モジュールでは使用する CAN-ch の can***n***_reset()を実行してください

can_init
can*n*_init　　(n=0~1)

概要:初期化関数

宣言:

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

説明:

・端子設定
・通信速度設定
・FIFO, TX キュー設定
・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、使用する ch の can***n***_init()を実行してください。can_reset(), can***n***_reset()後に実行してください。

can_receive_buf_conf
can*n*_receive_buf_conf (n=0~1)

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) [CANFD, CANFD_B, CANFD_Lite モジュール]
int can0_receive_buf_conf(void) [CANFD_2 モジュール]
int can1_receive_buf_conf(void) [CANFD_2 モジュール]

説明:

・受信メッセージバッファ数設定
・受信ルール(AFL)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:

CANFD, CANFD_B, CANFD_Lite モジュールでは、can_receive_buf_conf()を実行

CANFD_2 モジュールでは使用する CAN-ch の can***n***_ receive_buf_conf()を実行してください

can*n*_init() 後、can*n*_receive_rule_set()実行前に実行してください。

can*n*_receive_rule_set (n=0~1)

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);

int can1_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_RXFIFO7(0x0080) 受信 FIFO(7)で受信

CAN_RULE_CFIFO0(0x0100) 共通 FIFO(0)で受信

…

CAN_RULE_CFIFO5(0x2000) 共通 FIFO(5)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
CANFD モジュール: 0-63
CANFD_B, CANFD_Lite モジュール: 0-31
CANFD_2 モジュール: 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）
受信先、RXFIFO(2)~RXFIFO(7), CFIFO(1)~CFIFO(5)は CANFD モジュールでのみ設定可能です

can*n*_receive_rule_mask_set (n=0~1)

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);
int can1_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

cann_operate (n=0~1)

概要:動作モード変更関数

宣言:

int can_operate(void) [CANFD, CANFD_B, CANFD_Lite モジュールのみ]

int can0_operate(void)

int can1_operate(void)

説明:

・CANFD 系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

・AFL のロック

・FIFO の有効化（FIFO 使用時）

・TX キューの有効化（TX キュー使用時）

を行います

引数:

なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

補足:

CANFD, CANFD_B, CANFD_Lite モジュールでは、can_operate()の後、使用する ch の can***n***_operate()を実行してください

CANFD_2 モジュールでは、使用する ch の can***n***_operate()を実行してください

本関数実行後は、受信ルールの変更・追加（can***n***_receive_rule_set(), can***n***_receive_rule_mask_set()）は実行不可です

can_operate 実行（CANFD モジュール全体を動作モードに変更）後 ch 毎の動作モードを変更(can*n*_operate)を実行してください。

can*n*_txbuf_send (n=0~1) [CAN メッセージ送信関数]

canfd*n*_txbuf_send (n=0~1) [CANFD メッセージ送信関数]

概要:データ送信関数

宣言:

```
int can0_txbuf_send(unsigned char num, can_message *msg);
int can1_txbuf_send(unsigned char num, can_message *msg);
int canfd0_txbuf_send(unsigned char num, canfd_message *msg);
int canfd1_txbuf_send(unsigned char num, canfd_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

msg: 送信データを設定した CAN メッセージ構造体 [CAN メッセージ送信関数]

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: 送信データを設定した CANFD メッセージ構造体 [CANFD メッセージ送信関数]

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.fdf:

CAN_DATA_FORMAT(0) CAN フレームのフォーマットで送信

CANFD_DATA_DORMAT(1) CANFD フレームのフォーマットで送信

msg.brs:

CAN_BITRATE(0) データフィールドの通信速度は通常

CANFD_BITRATE(1) データフィールドの通信速度は高速

msg.dlc: 送信 DLC 値を指定します(1~15)

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

補足:

CANFD_B, CANFD_2, CANFD_Lite モジュールでは、num=0~3 を指定してください

CANFD モジュールでは、num=0~7, 32~39 を指定してください

（CANFD モジュールで CAN-ch1 から送信する際は、num=0~7→送信バッファ 64~71, num=32~39→送信バッファ 96~103 が使用されます）

cann_txqueue_send (n=0~1) [CAN メッセージ送信関数]
canfdn_txqueue_send (n=0~1) [CANFD メッセージ送信関数]

概要:データ送信関数

宣言:

int can0_txqueue_send(unsigned char queue_no, can_message *msg);
int can1_txqueue_send(unsigned char queue_no, can_message *msg);
int canfd0_txqueue_send(unsigned char queue_no, canfd_message *msg);
int canfd1_txqueue_send(unsigned char queue_no, canfd_message *msg);

説明:

・送信キューを使用してデータの送信
を行います

引数:

queue_no: 送信キュー番号
msg: 送信データを設定した CAN メッセージ構造体 [CAN メッセージ送信関数]
msg: 送信データを設定した CANFD メッセージ構造体 [CANFD メッセージ送信関数]

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): TX キューフル

補足:

CANFD_B, CANFD_2, CANFD_Lite モジュールでは、queue_no=0 のみ指定可能です
CANFD モジュールでは、queue_no=0~3 を指定してください

cann_cfifio_send (n=0~1) [CAN メッセージ送信関数]
canfdn_cfifo_send (n=0~1) [CANFD メッセージ送信関数]

概要:データ送信関数

宣言:

int can0_cfifo_send(unsigned char fifo_no, can_message *msg);
int can1_cfifo_send(unsigned char fifo_no, can_message *msg);
int canfd0_cfifo_send(unsigned char fifo_no, canfd_message *msg);
int canfd1_cfifo_send(unsigned char fifo_no, canfd_message *msg);

説明:

・共通 FIFO を使用してデータの送信

を行います

引数:

fifo_no: FIFO 番号

msg: 送信データを設定した CAN メッセージ構造体 [CAN メッセージ送信関数]

msg: 送信データを設定した CANFD メッセージ構造体 [CANFD メッセージ送信関数]

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:

CANFD_B, CANFD_2, CANFD_Lite モジュールでは、fifo_no=0 のみ指定可能です

CANFD モジュールでは、fifo_no=0~5 を指定してください

can*n*_rxbuf_receive (n=0~1) [CAN メッセージ受信関数]

canfd*n*_rxbuf_receive (n=0~1) [CANFD メッセージ受信関数]

概要:受信関数

宣言:

```
int can0_rxbuf_receive(unsigned char num, can_message *msg)
int can1_rxbuf_receive(unsigned char num, can_message *msg)
int canfd0_rxbuf_receive(unsigned char num, canfd_message *msg)
int canfd1_rxbuf_receive(unsigned char num, canfd_message *msg)
```

説明:

・受信メッセージ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

msg: 受信データを格納する CAN メッセージ構造体 [CAN メッセージ受信関数]

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）

msg: 受信データを格納する CAN メッセージ構造体 [CANFD メッセージ受信関数]
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.fdf:
CAN_DATA_FORMAT(0)　CAN フレームのフォーマットで受信
CANFD_DATA_DORMAT(1)　CANFD フレームのフォーマットで受信
msg.brs:
CAN_BITRATE(0)　データフィールドの通信速度は通常
CANFD_BITRATE(1)　データフィールドの通信速度は高速
msg.esi:
CANFD_ERROR_ACTIVE(0)　エラーアクティブ
CANFD_ERROR_PASSIVE(1)　エラーパッシブ
msg.dlc: DLC 値(1-15)
※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 [can***n***_rxbuf_receive]（受信データに含まれる DLC 値）
1~15: 受信したデータのバイト数に相当する DLC 値 [canfd***n***_rxbuf_receive]
（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(), can***n***_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

CANFD モジュールでは、can1_rxbuf_receive(), canfd1_rxbuf_receive()では、指定した num+16 の受信バッファの読み出しを行います（CAN-ch0 は 0~15, CAN-ch1 は 16~31 の受信バッファにアクセスします）

can_rxfifo_receive		[CAN メッセージ受信関数]
can*n*_rxfifo_receive	(n=0~1)	[CAN メッセージ受信関数]
canfd_rxfifo_receive		[CANFD メッセージ受信関数]
canfd*n*_rxfifo_receive	(n=0~1)	[CANFD メッセージ受信関数]

概要：受信関数

宣言：

int can_rxfifo_receive(unsigned char fifo_no, can_message *msg) [CANFD モジュール]

int can0_rxfifo_receive(unsigned char fifo_no, can_message *msg)
[CANFD_2, CANFD_B, CANFD_Lite モジュール]

int can1_rxfifo_receive(unsigned char fifo_no, can_message *msg) [CANFD_2 モジュール]

int canfd_rxfifo_receive(unsigned char fifo_no, canfd_message *msg) [CANFD モジュール]

int canfd0_rxfifo_receive(unsigned char fifo_no, canfd_message *msg)
[CANFD_2, CANFD_B, CANFD_Lite モジュール]

int canfd1_rxfifo_receive(unsigned char fifo_no, canfd_message *msg) [CANFD_2 モジュール]

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

msg: 受信データを格納する CAN メッセージ構造体 [CAN メッセージ受信関数]

msg: 受信データを格納する CAN メッセージ構造体 [CANFD メッセージ受信関数]

戻り値：

1~8: 受信したデータのバイト数 [can***x***_rxfifo_receive](x=0,1,空白)（受信データに含まれる DLC 値）

1~15: 受信したデータのバイト数に相当する DLC 値 [canfd***x***_rxfifo_receive]
（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

CANFD モジュールでは、can_rxfifo~, その他のモジュールでは、can0_rxfifo の様に関数名が変わります
fifo_no は、CANFD モジュールでは 0~7, その他のモジュールでは 0~1 です

can_cfifo_receive		[CAN メッセージ受信関数]
can*n*_cfifo_receive	(n=0~1)	[CAN メッセージ受信関数]
canfd_cfifo_receive		[CANFD メッセージ受信関数]
canfd*n*_cfifo_receive	(n=0~1)	[CANFD メッセージ受信関数]

概要:受信関数

宣言:

int can_cfifo_receive(unsigned char fifo_no, can_message *msg) [CANFD モジュール]
int can0_cfifo_receive(unsigned char fifo_no, can_message *msg)
[CANFD_2, CANFD_B, CANFD_Lite モジュール]
int can1_cfifo_receive(unsigned char fifo_no, can_message *msg) [CANFD_2 モジュール]
int canfd_cfifo_receive(unsigned char fifo_no, canfd_message *msg) [CANFD モジュール]
int canfd0_cfifo_receive(unsigned char fifo_no, canfd_message *msg)
[CANFD_2, CANFD_B, CANFD_Lite モジュール]
int canfd1_cfifo_receive(unsigned char fifo_no, canfd_message *msg) [CANFD_2 モジュール]

説明:

・共通 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
msg: 受信データを格納する CAN メッセージ構造体 [CAN メッセージ受信関数]
msg: 受信データを格納する CAN メッセージ構造体 [CANFD メッセージ受信関数]

戻り値:

1~8: 受信したデータのバイト数 [can***x***_cfifo_receive](x=0,1,空白)（受信データに含まれる DLC 値）
1~15: 受信したデータのバイト数に相当する DLC 値 [canfd***x***_cfifo_receive]
（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): 共通 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
CANFD モジュールでは、can_cfifo~, その他のモジュールでは、can0_cfifo の様に関数名が変わります
fifo_no は、CANFD モジュールでは 0~5, その他のモジュールでは 0 のみ指定可能です

CAN のデータ受信は、受信バッファ, 受信 FIFO, 共通 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。

can*n*_tx_abort (n=0~1)

概要:送信停止関数

宣言:

int can0_abort(void) [ch0 向け]
int can1_abort(void) [ch1 向け]

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。

9.2. 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]　※CANFD 未使用時
canfd_message g_can_recv_buf[CAN_CH][CAN_RECV_BUF_SIZE]　※CANFD 使用時

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）
SAMPLE1~3 では、can_message 構造体。SAMPLE4 では、canfd_message 構造体となります。

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。

canfd_param g_canfd_setting

CANFD のパラメータを保持する変数です。
g_canfd_setting.speed : CANFD のビットレート
g_canfd_setting.tdce: トランシーバ遅延補償
g_canfd_setting.tdco: 第 2 サンプリングポイント設定
g_canfd_setting.refe: RX エッジフィルタ
のメンバを持ちます。

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_canfd_ch_error_flag[CAN_CH]
unsigned long g_can_error_flag [CANFD モジュール]
unsigned long g_can_error_flag[CAN_CH] [CANFD_B, CANFD_2, CANFD_Lite モジュール]

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 CANFD のエラーレジスタ用、グローバルエラー用に 3 つ用意されています。

unsigned long g_can_ch_error_flag_history[CAN_CH]
unsigned long g_canfd_ch_error_flag_history[CAN_CH]
unsigned long g_can_error_flag_history [CANFD モジュール]
unsigned long g_can_error_flag_history[CAN_CH] [CANFD_B, CANFD_2, CANFD_Lite モジュール]

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	2022.8.19	—	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	RA の割り込み番号の変更、特定のマイコン型名からモジュール名への変更 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
REV.1.11.0.0	2024.11.29	P25	RX261 の記載を追加
REV.1.12.0.0	2025.1.21	P23	RA8E1 の記載を追加

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

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

株式会社 北斗電子
 2025 年 1 月 21 日改訂 REV.1.12.0.0 (250121)