LIN・CAN スタータキット RL78/F24
RS-CANFD_Lite ソフトウェア編
マニュアル

ルネサス エレクトロニクス社　RL78/F24(QFP-100 ピン)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.1.0.1.0

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、「LIN・CAN スタータキット RL78/F24」付属 CD に含まれる、CAN 動作のサンプルプログラムの解説を行う資料となります。

・付属 CD（本マニュアルに関連する内容のみ抜粋）

フォルダ			内容
	RL78_F24_100_CAN¥ (*3)		CAN サンプルプログラム [本書で説明する内容]
	RL78_F24_100_CAN_MONITOR¥ (*3)		CAN の通信データを確認するためのサンプルプログラム
	OTHERS¥RL78_F24_64_CAN¥ (*4)		HSBRL78F24-64 を通信相手として CAN のサンプルプログラムの動作を見る場合のサンプルプログラム
	OTHERS¥RL78_F15_CAN_S3¥ (*4)		HSBRL78F15-100 を通信相手として CAN のサンプルプログラムの動作を見る場合のサンプルプログラム(*1)
	OTHERS¥RL78_F13_CAN_S3_64¥ (*4) OTHERS¥RL78_F14_CAN_S3_64¥ (*4) OTHERS¥RL78_F15_CAN_S3_64¥ (*4)		HSBRL78F13-64 HSBRL78F14-64 HSBRL78F15-64 を通信相手として CAN のサンプルプログラムの動作を見る場合のサンプルプログラム(*1)
BINARY¥	RA		RA マイコン搭載ボードをを通信相手として CAN のサンプルプログラムの動作を見る場合のバイナリ(*2)
	RX		RX マイコン搭載ボードをを通信相手として CAN のサンプルプログラムの動作を見る場合のバイナリ(*2)
	RL78¥HSBRL78F24-100 (*3) RL78¥OTHERS¥ (*4)		SOURCE に含まれるサンプルプログラムの mot ファイル
DOCUMENT¥	HSBRL78F24-100_REV_X_s.pdf		マイコンボード取扱説明書
	LIN_CAN_KIT_RL78F24_software_CAN_REV_X_s.pdf		サンプルプログラム解説書(1)
	LIN_CAN_KIT_RL78F24_software_LIN_REV_X_s.pdf		サンプルプログラム解説書(2)
	LIN_CAN_KIT_RL78F24_REV_X_s.pdf		本資料

(*1)CANFD には非対応、CAN の 4 種類のサンプルプログラムの内 SAMPLE3 のプロジェクトになっています（ソースコードが含まれるプロジェクトです）

(*2)CAN スタータキット RX/RA 対応マイコンボードを CAN の対向機として使用する場合、本フォルダに含まれるバイナリ（mot ファイル）をマイコンボードに書き込んで、本キットのマイコンボード(HSBRL78F24-100)と通信を行わせてください

(*3)本キット付属の HSBRL78F24-100 向けのプロジェクト、バイナリです。

(*4)通信対向機として使用可能な当社製マイコンボード向けのプロジェクト、バイナリです。

1. サンプルプログラムの動作

付属 CD に収録されているサンプルプログラム

SOURCE¥RL78_F24_100_CAN
（CS+プロジェクト）

の動作に関して説明します。

1.1. サンプルプログラム動作時の接続形態

本キット付属のサンプルプログラムで、CAN の動作を見る場合の接続を以下に示します。

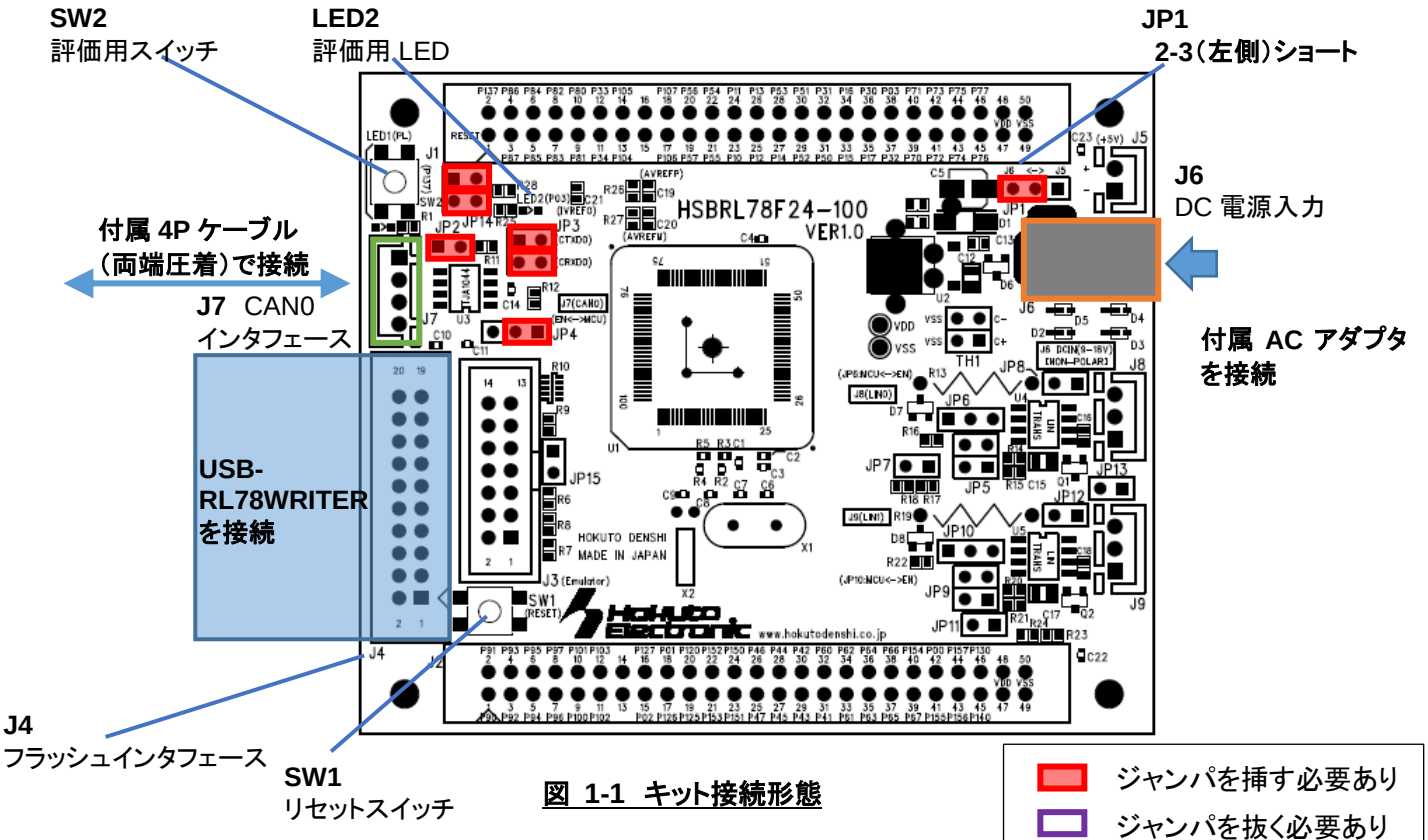


図 1-1 キット接続形態

・電源は、付属の AC アダプタを J6 に接続する
（本サンプルプログラムでは、LIN 電源は使用しませんので、J6 以外からの電源印加でも問題ありません）
・J7 と「他のボード」は、付属 4P ケーブルで接続する

－ジャンパ設定－

No	設定	用途	備考
JP1	2-3 ショート（左側）	ボード VDD 選択	J6 から入力した電源を元にボード上のレギュレータで 5V を生成し、ボード VDD に供給
JP2	ショート	CAN0-TERM	CAN0 の終端抵抗を有効化

JP3-A	ショート	CAN0-TX	CAN0 トランシーバ IC の出力をマイコンに接続
JP3-B	ショート	CAN0-RX	CAN0 トランシーバ IC の出力をマイコンに接続
JP4	1-2 ショート	CAN0-STB	マイコンの出力を CAN0 トランシーバ IC に接続 （マイコンから STB 制御）
JP14-A	ショート	SW-EN	評価用スイッチを使用する
JP14-B	ショート	LED-EN	評価用 LED を使用する

1.1.1. HSBRL78F24-100 を 2 台使用する場合

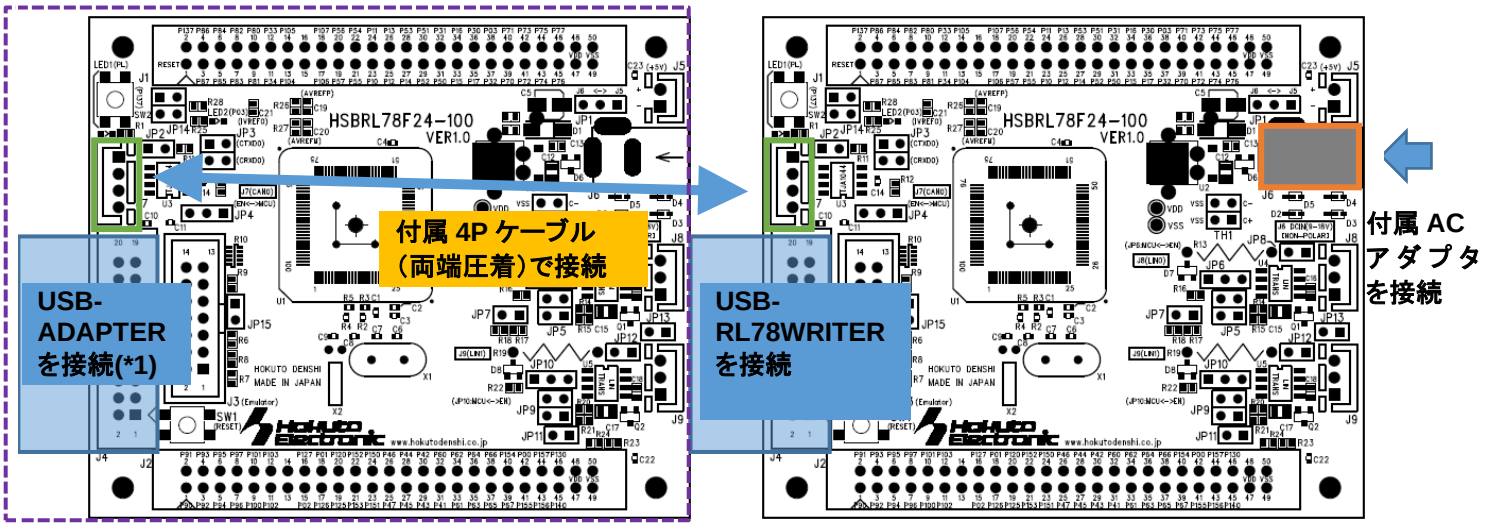


図 1-2 キット接続形態(1)

2 台のボードで CAN の動作を見る場合は、
・どちらか 1 台のボードに電源を供給する
（もう一方のボードには、CAN ケーブルを経由して給電されます）
・CAN を動作させるためのジャンパは図 5.1 の設定とする
(*1)UART(SCI)で情報を表示しますので、20P コネクタに通信モジュールを接続する事を推奨致します。
当社製品では、
USB-RL78WRITER
USB-ADAPTER
が使用可能です。市販の USB-Serial 変換機器でも可。

2 台目のボードを、HSBRL78F15-100 とする事も可能です。その場合は、HSBRL78F15-100 側には、CD の BINARY¥RL78¥OTHERS¥HSBRL78F15-100¥ RL78_F15_CAN_S3.mot（バイナリ、mot ファイル）を書き込んで下さい。
（HSBRL78F15-100 向けのソースコードは、SOURCE¥OTHERS¥RL78_F15_CAN_S3（プロジェクトフォルダ）に含まれます。）

1.1.2. 当社製 RL78 の CAN 対応 64pin ボードを使用する場合

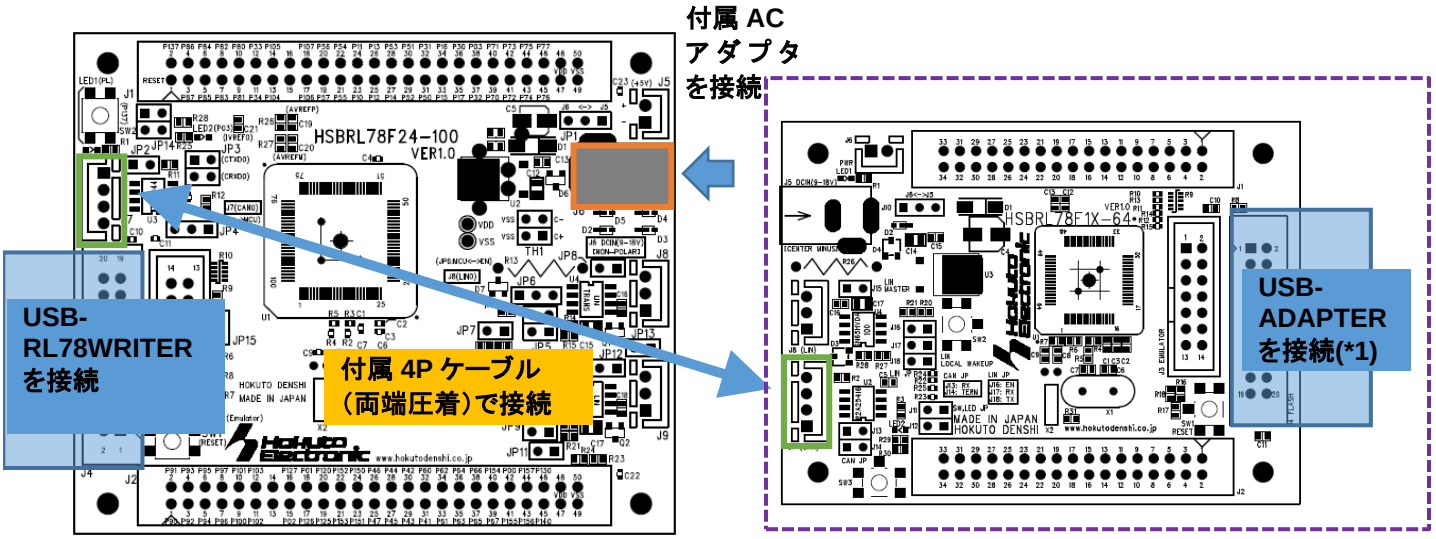


図 1-3 キット接続形態(2)

通信相手として、当社製の
HSBRL78F24-64(*2)
HSBRL78F13-64(*3)
HSBRL78F14-64(*3)
HSBRL78F15-64(*3)
を接続して動作を確認する事が可能です。

(*2)のボードに書き込むプログラムは、CD の SOURCE¥OTHERS¥RL78_F24_64_CAN （ソースコードを含むプロジェクト）、ビルド済みの mot ファイルは BINARY¥RL78¥OTHERS¥HSBRL78F24-64 以下に含まれています。
SAMPLE4（CANFD パケットを使うプログラム）の動作を見る場合は、SAMPLE4 のプログラムを HSBRL78F24-64 に書き込んでください。SAMPLE1~SAMPLE3（CAN パケットを扱うプログラム）の動作を見る場合は、SAMPLE3（SAMPLE1~SAMPLE2 上位互換）のプログラムを書き込んでください。

(*3)のボードに書き込むプログラムは、CD の SOURCE¥OTHERS¥RL78_F1x_CAN_S3_64(x=3,4,5) （ソースコードを含むプロジェクト）、ビルド済みの mot ファイルは BINARY¥RL78¥OTHERS¥HSBRL78F1x-64(x=3,4,5) 以下に含まれています。
※RL78/F1x の 64pin のボード向けのサンプルプログラムは SAMPLE3 相当の動作です（SAMPLE1, SAMPLE2 のサンプルプログラムと、SAMPLE3 は相互に通信可能です）RL78/F1x の 64pin のボードでは、CANFD(SAMPLE4)の通信は行えません。

・どちらか 1 台のボードに電源を供給する
（もう一方のボードには、CAN ケーブルを経由して給電されます）
・CAN を動作させるためのジャンパは図 5.1 の設定とする
(*1)SCI(UART)で情報を表示しますので、20P コネクタに通信モジュールを接続する事を推奨致します。
キット付属の USB-RL78-WRITER の他、USB-ADAPTER も使用可能です。

1.1.3. 当社製「CAN スタータキット RX/RA」の組み合わせ

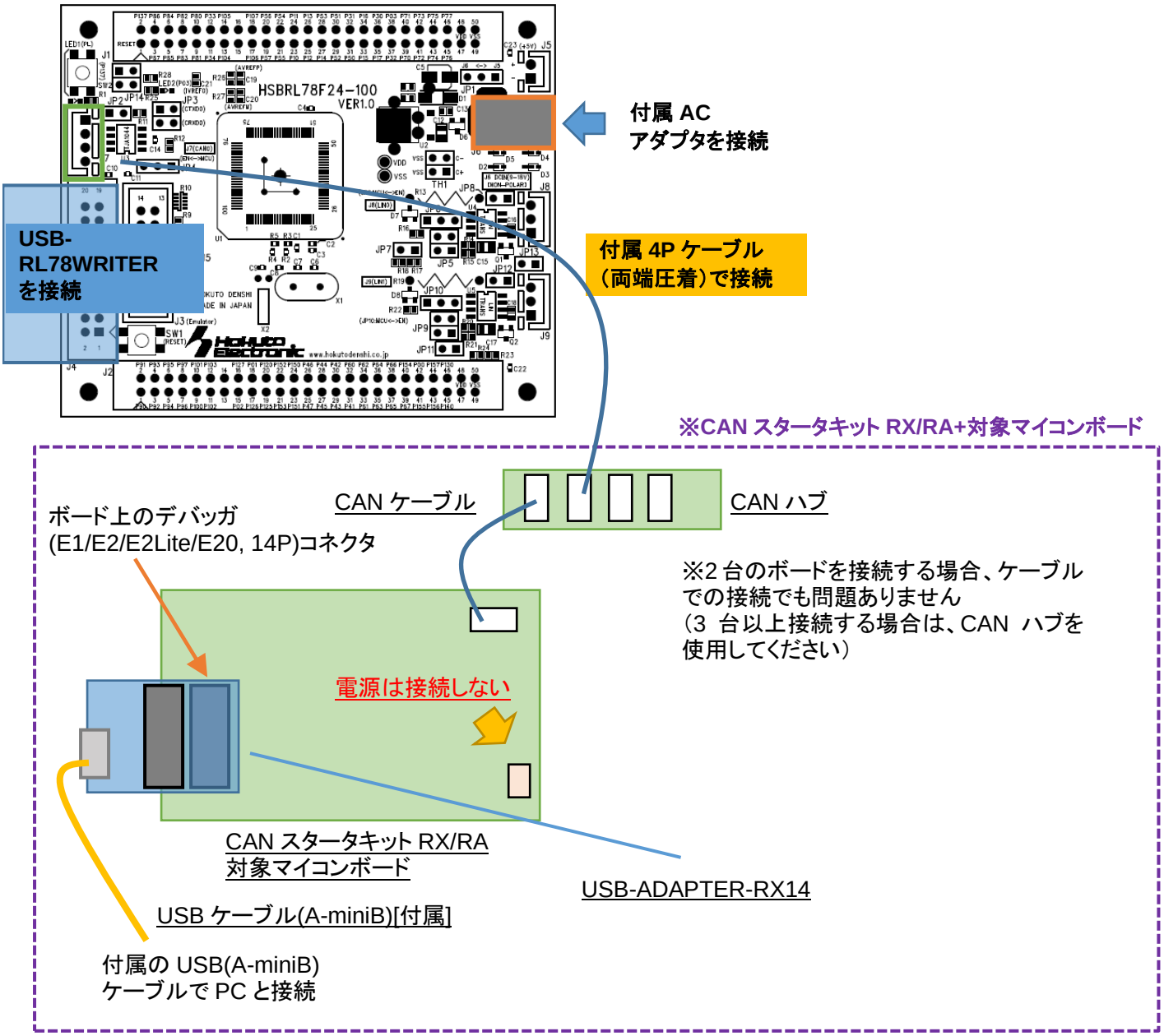


図 1-4 キット接続形態(3)

・どちらか 1 台のボードに電源を供給する
（もう一方のボードには、CAN ケーブルを経由して給電されます）
・CAN を動作させるためのジャンパは図 5.1 の設定とする

SAMPLE1~SAMPLE4 のサンプルプログラムは、当社製「CAN スタータキット RX/RA」と同じ動作となる様に作成されており、「CAN スタータキット RX/RA」と組み合わせた RX や RA マイコンを搭載したマイコンボードとの通信が可能です。（※SAMPLE4 は、CANFD 対応マイコンのみ対応）

「CAN スタータキット RX/RA」について

当社製の CAN スタータキットで、
RX マイコン（RX140, RX231, RX23E-B, RX24T, RX24U, RX64M, RX651, RX660, RX671, RX66N, RX66T, RX71M, RX72M, RX72N, RX72T)及び、
RA マイコン(RA2A1, RA2L1, RA4E1, RA4E2, RA4M1, RA4M2, RA4M3, RA4T1, RA6E1, RA6E2, RA6M1, RA6M2, RA6M3, RA6M4, RA6M5, RA6T1, RA6T2, RA6T3)
等のマイコンに対応したキットです。

LIN・CAN スタータキット RL78/F24 の SAMPLE1~SAMPLE4 は、「CAN スタータキット RX/RA」のサンプルプログラムと同等の動作です。

「CAN スタータキット RX/RA」のサンプルプログラムが書き込まれた、RX, RA マイコンボードと通信を行う事が可能です。

※3 台以上のマイコンボードを接続する場合は、終端抵抗は 2 箇所（できるだけ端に位置する 2 箇所のマイコンボード）のジャンパを挿して終端抵抗を ON するようにしてください
HSBRL78F24-100 のボードでは、JP2 が CAN0 の終端抵抗となります。

1.2. サンプルプログラムの構成

サンプルプログラムは、単純な送受信プログラムから、徐々に機能追加していく流れになっています。

・SAMPLE1
送信バッファを使用したデータフレームの送信
受信バッファを使用したデータフレームの受信
（割り込みは未使用）

・SAMPLE2
送受信 FIFO を使用したデータフレームの送信
受信 FIFO を使用したデータフレームの受信
割り込み処理を使用

・SAMPLE3
SAMPLE2 の機能に加えて、
リモートフレームの送信
リモートフレームの応答

・SAMPLE4

SAMPLE3 の CANFD 対応版

SAMPLE1~SAMPLE3 では、CANFD パケットは取り扱いません。（CANFD 非対応のマイコンボードと通信を行う場合は、SAMPLE1~3 を使用してください。）

SAMPLE1~SAMPLE4 の切り替えは、

settings¥program_select.h

の書き換えによって行います。

サンプルプログラムのビルドに関しては、3.2 章で説明していますので、そちらを参照ください。

1.3. 対向機として HSBRL78F24-64 を接続した場合

接続形態 1.1.2 で示す、HSBRL78F24-64 を接続して通信を行わせる場合の動作例です。

1.3.1. サンプルプログラム(SAMPLE1)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示

を行うサンプルプログラムとなっています。

プログラムをマイコンボードに書き込み起動すると、端末に、下記の表示が出力されます。

マイコンボード(1) HSBRL78F24-100

```
HSBRL78F24-100 CAN Starter kit program boot.
Copyright (C) 2024 HokutoDenshi. All Rights Reserved.

SAMPLE1: CAN [Data frame] send/receive simple program.
CAN ID mode -> EID

Command Usage:
   0123: Data frame send
   z: LED blink test(for board identify)
   s: send fornat EID <-> SID
   a: send abort
   S: Status resister print
   E: Error register print
   H: Error register history print
   C: Error register / occurrence counter clear
   (Push-SW: Data frame send [=keyboard 0])
```

HSBRL78F24-100 マイコンボードに、SAMPLE1 のプログラムを書き込んで起動すると、接続先の PC の端末には上記メッセージが表示されます。

※端末の通信速度は 115,200bps です

端末にキーボードからコマンドを入力する事により動作します・

本プログラムで使用可能なコマンドとしては、

0: 1 バイト送信(id=0x0)
1: 2 バイト送信(id=0x1)
2: 4 バイト送信(id=0x2)

3: 8 バイト送信(id=0x3)
z: ボード上の LED が点滅（複数台のボード使用時に、端末とボードの関係が判らなくなった場合に使用）
s: 標準 ID と拡張 ID をトグルで切り替える
a: 送信停止
S: ステータスレジスタの表示
E: エラーレジスタの表示
C: エラーレジスタと通信カウンタのクリア

となっています。

なお、マイコンボード上の SW2 を押す事により、コマンド"0"と同じ動作を行います。

マイコンボード(1) HSBRL78F24-100

マイコンボード(2) HSBRL78F24-64　※（別売）本キットには含まれません, SAMPLE1 のプログラム書き込み

の場合の表示例を以下に示します。

マイコンボード(1)の端末から、"0"を入力します。

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
```

マイコンボード(1)から、データフレーム送信を行い
ret=0（送信関数の戻り値 OK）
id_type=EID（ID は拡張(29bit)ID）
id=0x00000000（ID 値は、0x00000000）
rtr=0x00（データフレームを表す）
dlc=1（送信バイト数 1）
data=0x01（送信データ 0x01）
上記の内容で、送信した旨、表示されます。

マイコンボード(2)は、マイコンボード(1)が送信したデータを受信して以下の表示が出力されます。

```
CAN0 data received, buf=4 ret=1 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01 ts=0x5209
```

buf=4（受信バッファ 4 で受信）
ret=1（受信関数の戻り値 1=受信データバイト数）
id_type=EID（ID は拡張(29bit)ID）
id=0x00000000（ID 値は、0x00000000）

rtr=0x00（データフレームを表す）
dlc=1（受信バイト数 1…関数の戻り値と同値）
data=0x01（受信データ 0x01）
ts=0x5209（受信のタイムスタンプ値）

※タイムスタンプ値は、送信データには含まれません。受信側で、受信順に付与しているデータです。タイムスタンプ値は、受信のタイミングで値は変わります。

上の例が、マイコンボード(1)から(2)に対し、CAN で 1 バイトのデータを送った際の表示です。

マイコンボード(1)の SW2 を押した際にも、同様の動作（1 バイト送信＆受信）となります。

今度は、マイコンボード(2)側でコマンド"3"を試してみます。

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000003 rtr=0x00 dlc=8 data=0x0123456789ABCDEF
```

```
CAN0 data received, buf=7 ret=8 id_type=EID id=0x00000003 rtr=0x00 dlc=8 data=0x0123456789ABCDEF
```

送信元がマイコンボード(2)側となり、マイコンボード(1)で受信します。先程のコマンドとの相違点は、
dlc=8, data=0x123456789ABCDEF
送信データバイト数が 8 バイトになっている事と
id=0x00000003
ID が、0x00000003 になっている事、
buf=7
受信バッファ 7 で受信した事となります。

なお、ボード上の LED(LED2)は、データを受信する度に点灯/消灯がトグルで切り替わります。

"z"コマンドは、入力するとボード上の LED が点滅します。端末とボードの関係（2 つの端末を開いていて、どちらがどちらのマイコンボードに指示を出しているか）が判るようにするためのコマンドです）。

"s"コマンドは送信する ID を標準 ID に切り替えるコマンドです。拡張 ID(29bit)から標準 ID(11bit)に切り替わります。

マイコンボード(1)側で"s"コマンド、"1"コマンドを試してみます。

```
send format -> SID
CAN0 data frame send, ret=0 id_type=SID id=0x00000001 rtr=0x00 dlc=2 data=0x0123
```

```
CAN0 data received, buf=1 ret=2 id_type=SID id=0x00000001 rtr=0x00 dlc=2 data=0x0123 ts=0x4577
```

受信側で、ID フォーマットが SID で受信する様になります。

ID がどちらのタイプかは、
id_type=
の行で判るようになっています。
（"s"コマンドは、トグルで動作するコマンドなので、"s"コマンドを入力する度に、送信データの標準 ID←→拡張 ID が切り替わります。）

"a"コマンド

```
send abort!
```

送信を中断するコマンドです。受信相手が居ない場合で、データ再送を繰り返している状態などで、データ送信を止める事が出来ます。

"S", "E", "H", "C"のコマンドは、CAN のステータスを見るコマンドです。

"S"コマンド

```
-----
CAN0 status register

Transmit error counter (TEC)       : 0
Receice error counter (REC)        : 0

ESI status flag (ESIF)                    : 0 -> NOT received CANFD message with ESI flag
Communication status flag (COMSTS)   : 1 -> communication ready
Receive status flag (RECSTS)            : 0 -> idle
Transmit status flag (TRMSTS)           : 0 -> idle
Bus-off status flag (BOSTS)             : 0 -> NOT bus-off
Error-passive status flag (BOSTS)     : 0 -> NOT error-passive
Channel sleep status flag (CSLPSTS) : 0 -> NOT channel sleep mode
Channel halt status flag (CHLTSTS)   : 0 -> NOT channel halt mode
Channel reset status flag (CRSTSTS) : 0 -> NOT channel reset mode

-----
CANFD0 status register

Successful occurrence counter (SOC) : 14
Error occurrence counter (EOC)          : 0

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation is present
PNF mode status flag (PNSTS)                                    : 0b00 -> Nomal (not use PNF)
Successful occurrence counter overflow flag (SOCO)       : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)             : 0 -> NOT overflow
Transmitter delay compensation result status (TDCR)     : 0x00
```

各種ステータスの値を見ることが出来ます。

"E"コマンド

```
-----
CAN error flag register

GERFLL = 0x0000

CANFD payload overflow flag (CMPOF)                 : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow is detected
FIFO message lost flag (MES)                        : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                                : 0 -> NO DLC error is detected

-----
CAN0 error flag register

C0ERFLL = 0x0000

ACK delimiter error flag (ADERR)                    : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)    : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                               : 0 -> NO CRC error is detected
ACK error flag (AERR)                               : 0 -> NO ACK error is detected
Form error flag (FERR)                              : 0 -> NO form error is detected
Stuff error flag (SERR)                             : 0 -> NO stuff error is detected
Arbitration-lost flag (ALF)                         : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                                 : 0 -> NO bus-lock is detected
Overload flag (OVLF)                                : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                        : 0 -> NO bus-off recovery is detected
Bus-off entry flag (BOEF)                           : 0 -> NO bus-off entry is detected
Error-passive flag (EPF)                            : 0 -> NO error-passive is detected
Error-warning flag (EWF)                            : 0 -> NO error-warning is detected
Bus-error flag (BEF)                                : 0 -> NO bus-error is detected

-----
CANFD0 error flag register

C0FDSTSL = 0x0000

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation
Successful occurrence counter overflow flag (SOCO)       : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)            : 0 -> NOT overflow
```

各種エラーフラグ値を見ることが出来ます。

"H"コマンド

```
-----
CAN error flag register [error history]

CANFD payload overflow flag (CMPOF)                 : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow is detected
FIFO message lost flag (MES)                        : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                                : 0 -> NO DLC error is detected

-----
CAN0 error flag register [error history]

ACK delimiter error flag (ADERR)                    : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)    : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                               : 0 -> NO CRC error is detected
ACK error flag (AERR)                               : 0 -> NO ACK error is detected
Form error flag (FERR)                              : 0 -> NO form error is detected
Stuff error flag (SERR)                             : 0 -> NO stuff error is detected
Arbitration-lost flag (ALF)                         : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                                 : 0 -> NO bus-lock is detected
Overload flag (OVLF)                                : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                        : 0 -> NO bus-off recovery is detected
Bus-off entry flag (BOEF)                           : 0 -> NO bus-off entry is detected
Error-passive flag (EPF)                            : 0 -> NO error-passive is detected
Error-warning flag (EWF)                            : 0 -> NO error-warning is detected
Bus-error flag (BEF)                                : 0 -> NO bus-error is detected

-----
CANFD0 error flag register [error history]

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation
Successful occurrence counter overflow flag (SOCO)    : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)         : 0 -> NOT overflow
```

各種エラーフラグの履歴を見ることが出来ます。

"C"コマンド

```
CAN Error regiser / occurrence counter clear
```

エラーフラグと、通信カウンタのクリアを行います。

RL78/F24 マイコンに搭載されている、RS-CANFD_Lite モジュールは、「送信バッファ」「送受信 FIFO(SRFIFO)」を使用してデータの送信を行い、「受信バッファ」「受信 FIFO(RXFIFO)」「送受信 FIFO(SRFIFO)」を使用してデータの受信を行います。本サンプルプログラムでは、送信は「送信バッファ」、受信は「受信バッファ」を使用しています。（一番シンプルなデータの取り扱い方かと考えます。）（プログラム内の表記としては、受信 FIFO は RXFIFO、送受信 FIFO は SRFIFO と記載しています。受信 FIFO と RXFIFO、送受信 FIFO と SRFIFO は同じものを示しています）

本サンプルプログラムの送信バッファと、受信バッファの設定を以下に示します。

・送信バッファ

送信バッファ番号	キーボードからのコマンド
0	0
1	1
2	2
3	3

送信バッファは、4 本あり、キーボードからのコマンドに応じて使用する送信バッファを変えています。

・受信バッファ

受信ルール番号	フォーマット	ID	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

RL78/F24 では、最大 16 個の受信ルールが設定可能で、ルールにマッチした（設定した ID 等が一致した）データを受信し、受信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（受信バッファで受信）。（受信バッファは、最大 16 個ありますが、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、マイコンの仕様上は設定は自由です。）

データの受信に関しては、標準 ID・拡張 ID フォーマット区分(IDE)及び、ID に応じた受信ルールが適用されます。例えば、拡張フォーマットで ID=0x0000002 のデータを受信した際、

受信ルール 6 にマッチし、受信バッファ 6 にデータが格納される

となります。

※データを受信した際に、マッチする受信ルールがあるとデータが受信され、どのルールにもマッチしないとデータが捨てられるという動作となります。

・サンプルプログラムのキーボードから入力可能なデータ送信コマンド

コマンド	ID	送信バイト数(DLC)	送信データ	送信バッファ番号
0	0x0000000	1	0x01	0
1	0x0000001	2	0x0123	1
2	0x0000002	4	0x01234567	2
3	0x0000003	8	0x0123456789ABCDEF	3

サンプルプログラムでは、0~3 のキーボード入力に対し、上記の ID、送信バイト数(DLC)、送信データを送出します。0~3 の番号の送信バッファを使用しています。

※ID=0x0000002 の場合

bit	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
標準フォーマット																			0	0	0	0	0	0	0	0	0	1	0
拡張フォーマット	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

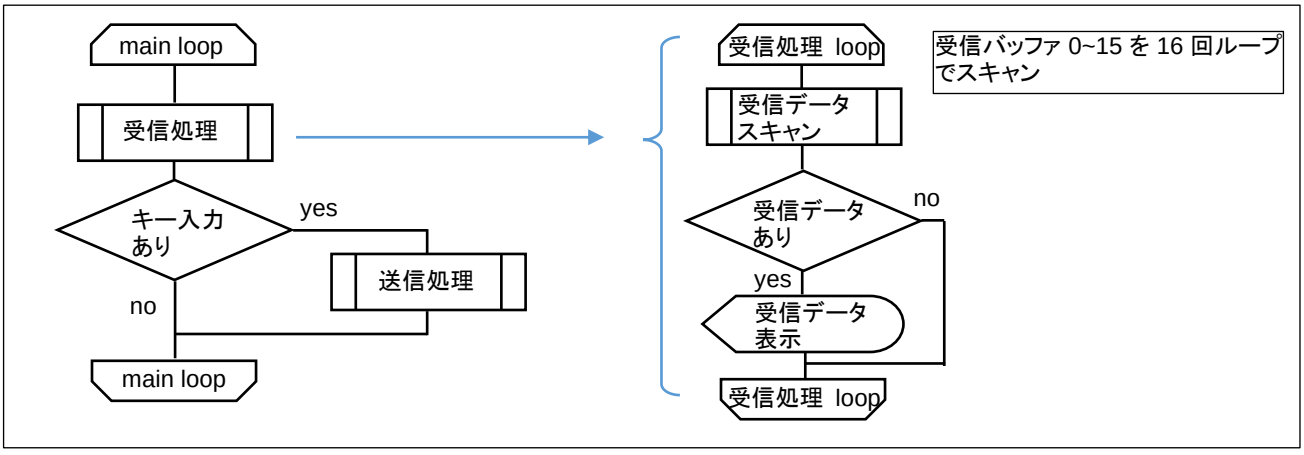
ID は、標準フォーマットでは、11bit で送信を行い、拡張フォーマットでは 29bit で送信します。(拡張フォーマットで送信する場合でも、下位の 11bit が先に送信されます)

"s"コマンドで、送信フォーマット、「標準フォーマット」←→「拡張フォーマット」の切り替えを行います。

評価用のプッシュスイッチ(SW2)はキーボードの"0"と同じ(データフレームで、1 バイト送信)効果となります。

また、1 回のデータ受信で、ボード上の LED(LED2)の点灯/消灯がトグルで切り替わります。

本サンプルプログラムでは、各種初期化を行った後はプログラムをループさせておりループ内で送信の処理と受信データの有無の確認を行っています。メインループの簡易フローチャートを以下に示します。



本サンプルプログラムでは、受信データの有無をポーリング処理によって確認しています。

・常に受信データの確認を行っている(受信データの確認にリソースが消費されている)

・送信等別な処理が入ると、受信データのスキャン頻度が変わる

といったデメリットがあります。

次のサンプルプログラム(SAMPLE2)では、受信と送信後の処理を割り込みで行っています。
（メインループに受信処理が含まれません）

1.3.2. サンプルプログラム(SAMPLE2)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示

を行うサンプルプログラムです。見た目上の動作としては、SAMPLE1 と大きくは変わりませんが、割り込みを使用して処理を行っている点が異なります。

マイコンボード(1) HSBRL78F24-100

マイコンボード(2) HSBRL78F24-64　※（別売）本キットには含まれません, SAMPLE2 のプログラム書き込みの場合の表示例を以下に示します。

マイコンボード(1)の端末で、"0"を押した場合、マイコンボード(1)がデータとして 0x01（1 バイト）を送信します。

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
CAN0 send finished.(SRFIFO)
```

送信完了を示す send finished の行が、SAMPLE1 との相違です。

マイコンボード(2)では、送信とほぼ同時にデータを受信します。

```
CAN0 data received, id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01 ts=0x58D6
```

次に、マイコンボード(2)の端末で"1"を押した場合です。

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123
CAN0 send finished.(SRFIFO)
```

(1)のマイコンボード(HSBRL78F24-100)側でデータを受信します。

```
CAN0 data received, id_type=EID id=0x00000001 rtr=0x00 dlc=2 data=0x0123 ts=0xF296
```

受信は受信 FIFO を使用しますので、SAMPLE1 で表示されていた buf=の表示（受信バッファ番号）はなくなります。

見た目上の動作として、SAMPLE1 との相違点は
send finished
という表示が追加されている位です。

※(SRFIFO)の表示は、送受信 FIFO で送信した事を示しています（送信に送信バッファを使った場合は、表示が多少変わります

send finished の表示ですが、これは、送信が「完了」した事を示しています。送信先の相手が ACK を返したという事です。SAMPLE1 での送信動作は、送りっぱなし（送信が完了したか、ACK を返す相手が居なくて送信動作を繰り返しているかは表示されない）ですが、SAMPLE2 では送信完了時の割り込みにより、送信処理が完了した事が判り、送信の後処理を入れています（このプログラムでは画面表示ですが、実際のアプリケーションでは、送信完了後に別な処理を入れるといった使い方ができます）。

※CAN ケーブルを抜いた場合、SAMPLE1 と SAMPLE2 で違いがあります。CAN ケーブルを抜いた（CAN の接続先が存在しない）場合、0~3 のデータ送信コマンドに対し、SAMPLE1, SAMPLE2 共、データを送出した表示は出力されます。SAPMLE2 では、CAN ケーブルを抜いて、受信するポートがない場合、

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
*****
- can0_interrupt_error_callback() -
CAN0 error: ACK
CAN0 error: Bus-error
*****
```

上記の表示となり、
「CAN0 send finished.(SRFIFO)」
の表示が出力されません。この場合、送信が成功していないので、データの再送を試みます。（データを何度も再送するという動作となります。）
データの再送を繰り返しているときに、CAN ケーブルを挿すと、（及び受信側の受信準備が出来ていると）、

```
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0x01
*****
- can0_interrupt_error_callback() -
CAN0 error: ACK
CAN0 error: Bus-error
*****
CAN0 send finished.(SRFIFO)
```

「CAN0 send finished.(SRFIFO)」の表示が出るはずです。これは、送出したデータに対し、誰か（送信元以外の CAN デバイス）が反応したという事です。SAMPLE1 では、送信完了有無は判りません(*1)が、SAMPLE2 では表示で判るようになっています。

(*1)SAMPLE1 では行っていませんが、送信後ポーリング処理にて送信ステータスを監視すると、送信完了有無を判断する事は可能です

RS-CANFD_Lite の CAN パケットの送受信では、複数の手法を選択可能ですが、SAMPLE2 では、下線の
受信:受信 FIFO または 送受信 FIFO または （受信バッファ）
送信:送受信 FIFO または 送信バッファ
としています。
※can_operation.h で変更可能です
※送受信バッファは ch 毎に 1 本しかないため、受信か送信のどちらか一方での使用となります
※送信に送受信 FIFO を選択した場合、送受信 FIFO の先に送信バッファがぶら下がる形となります（結局、送信時に送信バッファは使用されます）

・送信に送受信 FIFO を使用した場合

送信 ch	送受信 FIFO	送信バッファ
CAN0	0	0

送信 FIFO は 4 段（4 つのメッセージをバッファリング）に設定し、送受信 FIFO の 0 番を使用（RL78/F24 では 0 番しかありません）。送受信 FIFO の 0 番を送信バッファの 0 番に紐付けする設定としています。送受信 FIFO の段数は最大 16 段まで拡張可能ですが、トータルで使用できるメッセージバッファ（メモリ）内でやりくりする必要があります。

・受信

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

(*1)デフォルトは受信 FIFO で受信、can_operation.h で選択可能
受信 FIFO は 0~1 の 2 本あり、サンプルプログラムでは 0 番の受信 FIFO を使用しています。（1 番の受信 FIFO は未使用。）受信ルール設定で、受信データをどこに格納するか設定可能です。
（ID=0x00000002 は、受信バッファで受信して、ID=0x00000003 は、受信 FIFO で受信するといった設定もマイコンの機能としては可能です。）

受信 FIFO は 4 段の設定です（最大は 16 段）。トータル 16 あるメッセージバッファを、「受信 FIFO」「送受信 FIFO」「受信バッファ」で共用するので、どこにいくつ割り振るかはプログラム作成する段階で決める事となります。

CAN の割り込みは、全部で 8 本ありますが、本サンプルプログラムでは、

・CAN グローバル受信 FIFO（受信 FIFO で受信時）
・CAN グローバル受信バッファ（受信バッファで受信時）(*1)
・CAN グローバルエラー
・CAN0 チャネル送信（送受信 FIFO 送信時、送信バッファ送信時）
・CAN0 チャネルエラー
・CAN0 送受信 FIFO（送受信 FIFO で受信時）(*1)
（・CAN0 ウェイクアップ）※本サンプルプログラムでは未使用
（・CAN RAM ECC）※本サンプルプログラムでは未使用

の 6 本を使用しています。

(*1)本サンプルプログラムのデフォルト（受信 FIFO で受信）では使用しません。
※can_operation.h で受信先を変更した際に(*1)が使用されます

※全ての受信ルールにマッチしないデータを受信した場合、「ACK は返す」「受信データを受信 FIFO 等のメッセージバッファには格納しない」という動作となります（送信側は、送信完了と判断）

1.3.3. サンプルプログラム(SAMPLE3)

・端末からの指示でデータフレームの送信
・データフレームの受信、端末への表示
・端末からの指示でリモートフレームの送信
・リモートフレーム受信時にレスポンスを送信

を行うサンプルプログラムです。SAMPLE2 に対して、リモートフレームの送信と応答の機能を追加したものです。

マイコンボード(1) HSBRL78F24-100

マイコンボード(2) HSBRL78F24-64　※（別売）本キットには含まれません, SAMPLE3 のプログラム書き込み

```
HSBRL78F24-100 CAN Starter kit program boot.
Copyright (C) 2024 HokutoDenshi. All Rights Reserved.

SAMPLE3: CAN [Data frame] send/receive program(with interrupt, use RXFIFO).
                [Remote frame] request/response program(with interrupt, use RXFIFO).

CAN ID mode -> EID

Command Usage:
   0123: Data frame send
   qwer: Remote frame send(data request)
   z: LED blink test(for board identify)
   s: send fornat EID <-> SID
   a: send abort
   S: Status resister print
   E: Error register print
   H: Error register history print
   C: Error register / occurrence counter clear
   (Push-SW: Data frame send [=keyboard 0])
Command Usage:
   0123: Data frame send
   qwer: Remote frame send(data request)
   z: LED blink test(for board identify)
   s: send fornat EID <-> SID
   c: send CAN ch change
   (Push-SW: Data frame send [=keyboard 0])
```

SAMPLE3 では、新たに q~r のコマンドが追加されています。これは、リモートフレーム送信コマンドです。

・サンプルプログラムのキーボードから入力可能なコマンド（SAMPLE3 追加分）

コマンド	ID	要求バイト数 (DLC)
q	0x0000	1
w	0x0001	2
e	0x0002	4
r	0x0003	8

端末で、"q"を押した場合、CAN0 が ID=0x0000000 の相手に 1 バイトのデータを送るよう要求します（リモートフレーム送信を行います）。

```
++
CAN0 remote frame send, ret=0 id_type=EID id=0x00000000 rtr=0x01 dlc=1      (1)
CAN0 send finished.(SRFIFO)                                                (1')
CAN0 data received, id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0xA1 ts=0xFBAD  (4)
--
```

```
++
CAN0 data received, id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x96DC     (2)
CAN0 response data send, id_type=EID id=0x00000000 rtr=0x00 dlc=1 data=0xA1 ret=0  (2')
CAN0 send finished.(SRFIFO)                                                (3)
--
```

(1)マイコンボード(1)から DLC=1 でリモートフレームのパケットを送信
(2)マイコンボード(2)がリモートフレームを受信
(3)マイコンボード(2)がリモートフレームに対する応答…通常のデータフレームを送信
(4)マイコンボード(1)がマイコンボード(2)が送ったデータフレームを受信

動作としては上記の様になります。(1')は(1)の送信完了割込み時の表示。(2')も同様です。

リモートフレームのレスポンスとして送信されるデータは、
0xA1A2A3A4A5A6A7A8
です（要求バイト数に応じて先頭から送信，プログラム内で固定）。

※リモートフレームは、RTR=1 となる、データフレームとは異なるパケットです
本サンプルプログラムでは、リモートフレームを受信した際に、リモートフレーム内に含まれる DLC 値（リモートフレーム送信元が、相手に要求するデータバイト数）に応じた、データフレームを返送する動作としています。
（リモートフレーム受信時に、データフレームを返送するかどうかは、自由です。）

※「+++」「---」は一連のメッセージの区切りのために表示させています

SAMPLE3 では、リモートフレームを受信するために、受信ルールが増えています。

・受信ルール（受信 FIFO または送受信 FIFO 使用）

受信ルール番号	受信 ch	フォーマット	ID	RTR 区分	受信 FIFO 番号	送受信 FIFO 番号(*1)
0	CAN0	標準(SID)	0x000	0	0	0
1	CAN0	標準(SID)	0x001	0	0	0
2	CAN0	標準(SID)	0x002	0	0	0
3	CAN0	標準(SID)	0x003	0	0	0
4	CAN0	標準(SID)	0x000	1	0	0
5	CAN0	標準(SID)	0x001	1	0	0
6	CAN0	標準(SID)	0x002	1	0	0
7	CAN0	標準(SID)	0x003	1	0	0
8	CAN0	拡張(EID)	0x0000000	0	0	0
9	CAN0	拡張(EID)	0x0000001	0	0	0
10	CAN0	拡張(EID)	0x0000002	0	0	0
11	CAN0	拡張(EID)	0x0000003	0	0	0
12	CAN0	拡張(EID)	0x0000000	1	0	0
13	CAN0	拡張(EID)	0x0000001	1	0	0
14	CAN0	拡張(EID)	0x0000002	1	0	0
15	CAN0	拡張(EID)	0x0000003	1	0	0

・受信ルール（受信バッファ使用）

受信ルール番号	受信 ch	フォーマット	ID	RTR 区分	受信バッファ番号(*1)
0	CAN0	標準(SID)	0x000	0	0
1	CAN0	標準(SID)	0x001	0	1
2	CAN0	標準(SID)	0x002	0	2
3	CAN0	標準(SID)	0x003	0	3
4	CAN0	拡張(EID)	0x0000000	0	4
5	CAN0	拡張(EID)	0x0000001	0	5
6	CAN0	拡張(EID)	0x0000002	0	6
7	CAN0	拡張(EID)	0x0000003	0	7
8	CAN0	拡張(EID)	0x0000000	1	8
9	CAN0	拡張(EID)	0x0000001	1	9
10	CAN0	拡張(EID)	0x0000002	1	10
11	CAN0	拡張(EID)	0x0000003	1	111

SAMPLE1, SAMPLE2 では、RTR が 0(CAN_DATA_FRAME)のみでしたが、SAMPLE3 では、1(CAN_REMOTE_FRAME)が増えています。

受信ルール 8~16 が増えていますが、8~16 に RTR=1 の条件を追加している訳ではなく、4~7, 12~15 に RTR=1 の条件を追加しています。これは、プログラムの定数設定(can_operation.h)で標準 ID のみ取り扱う様にした場合のためです。標準 ID のみ取り扱う様にすると、受信ルールの拡張(EID)の項がなくなります。このとき、受信ルールが 0, 1, 2, 3, 8, 9, 10, 11 の様に（途中が欠番）なり、その場合、受信ルールの 0~3 は有効ですが、受信ルール 8~11 が無効（受信したデータが受信ルール 8~11 にマッチしない）となります。

拡張(EID)が欠番となっても、途中欠番が出ない様に受信ルール番号を割り振るようにしています。

受信ルール設定時は、番号の若い順から埋めていき、途中で欠番が出ない様にしてください。

(*1)デフォルトでは未使用です。can_operation.h 内で、受信に送受信 FIFO, 受信バッファを使うように設定する事が可能。

受信に受信バッファを使う場合ですが、SID のリモートフレームは受信しない設定となっています。

メッセージバッファは、トータル 16 本で、送信に「送受信 FIFO」を使用した場合、FIFO 段数を 4 段としているので、メッセージバッファが 4 つ消費されます。そのため、受信バッファは。残り 12 本の中でやりくりする必要があります。
（送信に「送信バッファ」を使用した場合は、受信バッファとして 16 本使用可能です。）

※マイコン機能としては、1 つの受信ルールで、RTR=0/1 の両方，フォーマット=SID/EID 両方, ID が 0x1???（??? 全てにマッチする）といった設定が可能ですので、より幅広いデータを受信ルール設定する場合は、ルール設定のプロラムを見直してみてください
（全ての CAN データを受信する様に設定している
SOURCE¥HSBRL78_F24_100_MONITOR プロジェクトでは、ID に関しては全ての ID を受信するマスク設定としています）

1.3.4. サンプルプログラム(SAMPLE4)

SAMPLE3 に対して、CANFD パケットでのデータ送信を行う様に変更したのが、SAMPLE4 です。

マイコンボード(1) HSBRL78F24-100

マイコンボード(2) HSBRL78F24-64　※（別売）本キットには含まれません, SAMPLE4 のプログラム書き込み

```
HSBRL78F24-100 CAN Starter kit program boot.
Copyright (C) 2024 HokutoDenshi. All Rights Reserved.

SAMPLE4: CANFD [Data frame] send/receive program(with interrupt, use RXFIFO).
                [Remote frame] request/response program(with interrupt, use RXFIFO).

CAN ID mode -> EID

CANFD setting:

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

  CANFD speed [Mbps](2-5, 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
```

ボードを起動すると、通信ビットレートや受信回路の遅延補償等のパラメータを聞いてきます。

とりあえず、リターンキーで先へ進めます。CANFD ビットレートは、2Mbps、遅延補償、RX エッジフィルタ未使用の設定がデフォルトです。入力項目が、summary として表示された後にコマンド一覧が出力されます。

```
Command Usage:
  0123: Data frame send
  qwer: Remote frame send(data request)
  z: LED blink test(for board identify)
  s: send fornat EID <-> SID
  a: send abort
  S: Status resister print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])
```

※CANFD パラメータの入力後に受信待機の状態となります。

SAMPLE3 との相違は、データ送信部分のビットレートが高速になる事と、最大 64 バイトまでのデータが送れる事です（CAN は、8 バイトまで）。SAMPLE4 では、0-3, q-r のコマンドの送信バイト数が変更になっています。

－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

キーボードからの入力	ID	送信バイト数(DLC)
0	0x0000000	1(DLC=1)
1	0x0000001	8(DLC=8)
2	0x0000002	16(DLC=10)
3	0x0000003	64(DLC=15)

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(DLC=1)
w	0x0000001	8(DLC=8)
e	0x0000002	16(DLC=10)
r	0x0000003	64(DLC=15)

・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 で通信する 2 者のボード間では、ビットレートを同じ値としてください。(ビットレートが異なる場合は、通信が通りません。)

端末で、"3"を押した場合、CAN0 が ID=0x0000003 の相手に 64 バイトのデータを送ります。(データフレーム送信を行います)。

```
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 dlc=15
data=0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223242526272
8292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
CAN0 send finished.(SRFIFO)
--
```

```
++
CAN0 data received, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 esi=0x00 dlc=15
data=0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20212223242526272
8292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F ts=0xD257
--
```

受信側は 64 バイトのデータを受信します。(CAN に対して、サイズの大きなデータを送れるようになりますので、何かと使い勝手が良くなると思います。)

fdf=(1:CANFD パケット), brs=(1:高速ビットレート), esi=(ESI フラグ設定有無)の表示が、追加されています。

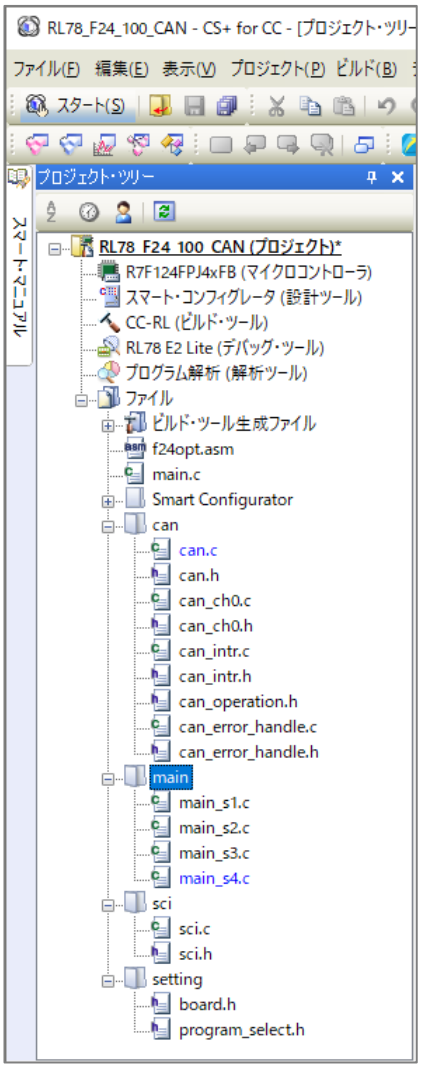
端末で、"r"を押した場合、CAN0 が ID=0x0000003 の相手に 64 バイト要求のリモートフレームを送ります。
(リモートフレームは、CANFD 使用時でも CAN パケットになります)
(リモートフレームの応答のデータフレームは、CANFD パケットとしています)

```
++
CAN0 remote frame send, ret=0 id_type=EID id=0x00000003 rtr=0x01 dlc=15
CAN0 send finished.(SRFIFO)
CAN0 data received, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 esi=0x00 dlc=15
data=0xFFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0DFDEDDDCDBDAD9D8
D7D6D5D4D3D2D1D0CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0 ts=0x47D4
--
```

```
++
CAN0 data received, id_type=EID id=0x00000003 rtr=0x01 fdf=0x00 brs=0x00 esi=0x00 dlc=15 ts=0x980C
CAN0 response data send, id_type=EID id=0x00000003 rtr=0x00 fdf=0x01 brs=0x01 dlc=15
data=0xFFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0DFDEDDDCDBDAD9D8
D7D6D5D4D3D2D1D0CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0 ret=0
CAN0 send finished.(SRFIFO)
--
```

2. ソースファイル構成とプロジェクトのビルドに関して

2.1. プロジェクト・ツリー



プロジェクトのツリー構成は上記の様になっており、それぞれのフォルダの内容は以下です。

main.c	メイン関数	main_s1~main_s4 へのエントリ関数
Smart Configurator	スマートコンフィグレータ生成コード	
can	RS-CANFD_Lite の CAN のソース	
main	メイン関数本体	SAMPLE1~SAMPLE4 でファイルが別
sci	UART 表示関数	
setting	設定ファイル	

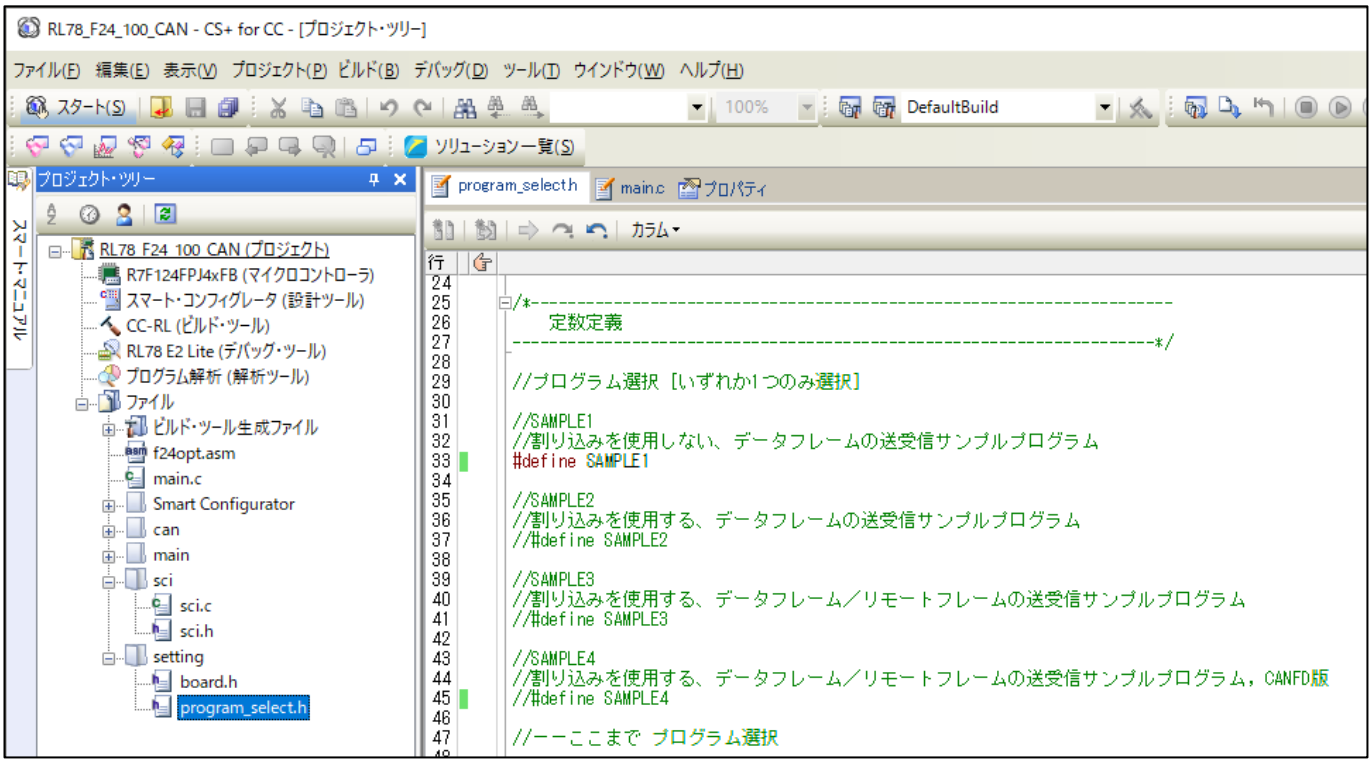
ソースファイルの構成は、

RL78_F24_100_CAN¥　（プロジェクトフォルダ）以下に
　smc_gen¥　　スマートコンフィグレータ生成コード
　usr_src¥　　本キット向けのソールフォルダ

usr_src 以下は以下のファイル構成です。

フォルダ	ファイル	説明
can¥		CAN ソースフォルダ
	can.c	ch に依存しない処理関数
	can.h	can.c 向けヘッダファイル
	can_ch0.c	ch0 関数
	can_ch0.h	ch0 関数用ヘッダファイル
	can_error_handle.c	エラー処理
	can_error_handle.h	エラー処理ヘッダファイル
	can_intr.c	割り込み処理
	can_intr.h	割り込み処理ヘッダファイル
	can_operation.h	速度や標準/拡張 ID の定義
main¥		メイン関数フォルダ
	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
sci¥		UART 処理ソースフォルダ
	sci.c	UART(SCI)処理関数
	sci.h	sci.c 向けヘッダファイル
setting¥		設定値格納フォルダ
	board.h	ボード固有定義ファイル
	program_select.h	SAMPLE1~SAMPLE4 選択ヘッダファイル

2.2. プロジェクトのビルド



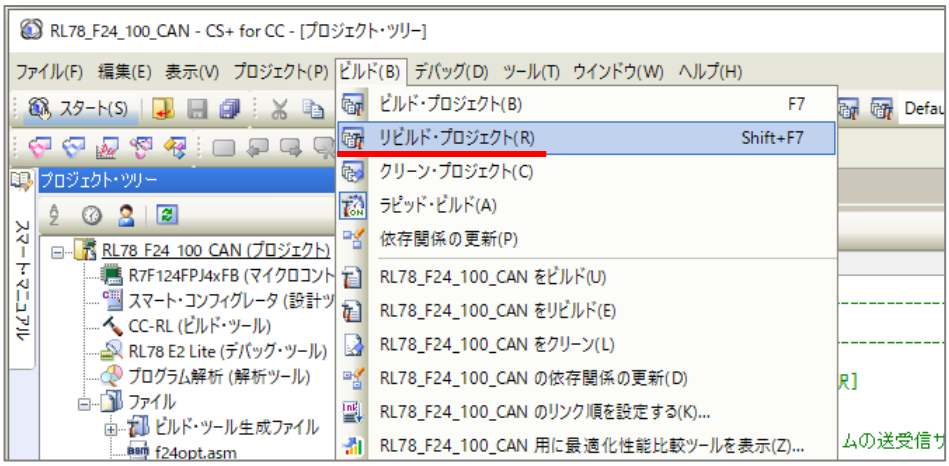
program_select.h が、SAMPLE1~SAMPLE4 の 4 種類のサンプルプログラムの切り替えを行うファイルです。

program_select.h

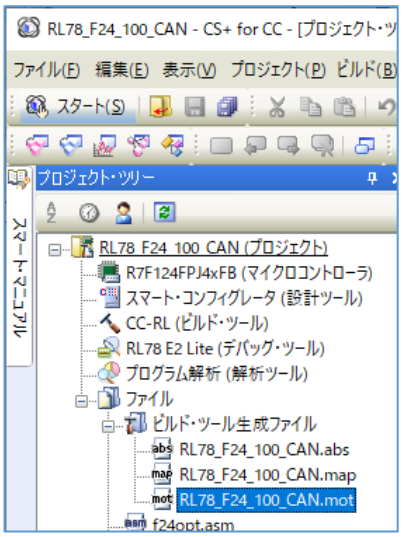
```
/*------------------------------------------------------------------------
定数定義
------------------------------------------------------------------------*/

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
//#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム，CANFD版
//#define SAMPLE4

//ーーここまで プログラム選択
```

#define SAMPLEn
という記載がありますので、1 つのみ有効（コメントアウトを外す）としてください。

※上記を変更した際は、プログラムの「リビルド・プロジェクト」を実行してください
（通常のビルドですと、変更箇所のビルドのみとなり、プログラム全体の定数設定が更新されない事があります。）



ビルドが成功すると、

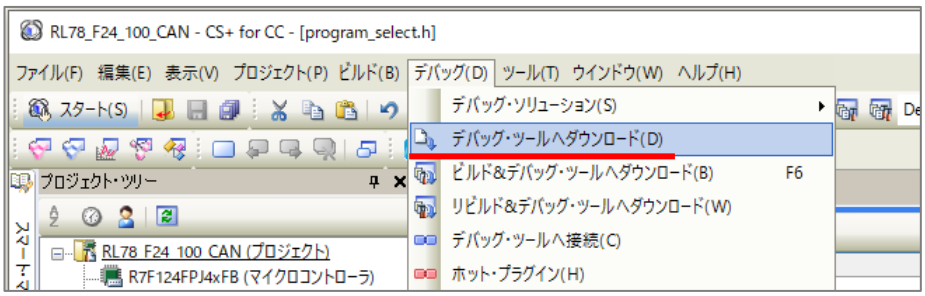


RL78_F24_100_CAN.mot というファイルが、プロジェクトフォルダの下の DefaultBuild の中に生成されます。

RL78_F24_100_CAN¥ （プロジェクトフォルダ）
　DefaultBuild¥RL78_F24_100_CAN.mot

このファイルが、ビルドで作成したファイルとなります。

E2 または E2Lite エミュレータをお持ちであれば、E2, E2Lite をボードの J3（14P コネクタ）に接続して、



デバッグ – デバッグ・ツールへダウンロード

を行ってください。エミュレータがない場合は、RL78_F24_100_CAN.mot を、RFP(RenesasFlashProgrammer)を使って HSBRL78F24-100 ボードに書き込んでください。書き込みの手順は、別マニュアル「LIN・CAN スタータキット RL78/F24 取扱説明書」の 4 章を参照してください。
（キット付属の USB-RL78WRITER を使って書き込む事ができます。）

プログラム書き込み後は、1.3 章に記載されている動作となります。

3. サンプルプログラムに含まれる関数の使用の流れ

3.1. 初期化

```
can_reset();
```

CAN モジュールの初期化を行います。

```
can_interrupt_setup();
can_error_interrupt_enable();
can_ch_error_interrupt_enable(0);
```

CAN の割り込み設定を行います（割り込み使用時）。

```
can0_init();
```

CAN の ch0 の初期化を行います。

※RL78/F24 は CAN は 1ch のみです
サンプルプログラムでは、ch 依存の処理（can0_等"0"が付く関数）と ch 非依存の処理（can_～など数字を含まない関数）を分けています。

```
can_receive_rule_conf();
```

CAN 受信ルール数設定を行います。設定可能な受信ルールは最大 16 なので、本関数内では受信ルール数=16 の設定を行っています。

[参考]受信ルール数は、全体で最大いくつと決められています。RS-CANFD 系の CAN モジュールで、複数 ch の CAN を取り扱う場合、ch 毎に受信ルール数を割り振る必要があります。RL78/F24 では ch0 しかありませんので、使用可能な 16 ルールを全て ch0 に割り振っています。本来わざわざ関数化して実行する意味はありませんが、複数 ch のサポートを念頭に独立した関数で処理しています。

```
can_receive_buf_conf(16);
```

受信バッファの設定。受信バッファで受信する場合のみ、実行が必要。RS-CANFD_Lite 系では、受信は
・受信バッファ
・受信 FIFO
・送受信 FIFO
のいずれか（もしくは複数）で行います。全体のバッファは RL78/F24 では、16 本あり 16 本のバッファを、「受信バッファ」「受信 FIFO」「送受信 FIFO」で分け合う形となります。

受信バッファを使わない場合は、本関数の実行は不要。受信 FIFO 等にバッファを振り分ける場合は、他で使用するバッファの本数と受信バッファ数の合計が 16 以内となる様に設定してください。

```
can_operate();
```

CAN モジュールを実行状態に遷移させます。

```
can0_receive_rule_set(0, CAN_RULE_RXBUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x0033);
```

受信ルール設定を行います。
受信ルールにマッチしたデータは受信して、マッチしないデータは受信しない（捨てられる）という動作となります。

1 つ目の引数はルール番号。受信ルール数は 16 としているので、0~15 の値が指定可能です。
2 つ目の引数は、受信先。
CAN_RULE_RXBUF(0x0000) 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_SRRXFIFO0(0x0100) 送受信 FIFO(0)で受信
3 つ目の引数は、ID 区分。
CAN_ID_FORMAT_SID(0) 標準 ID（11bit の ID）のデータ
CAN_ID_FORMAT_EID(1) 拡張 ID（29bit の ID）のデータ
4 つ目の引数は、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
CAN_REMOTE_FRAME(1) リモートフレーム
5 つ目の引数は、ID 値。

上記の場合、0 番の受信ルールとして、
「拡張 ID」で「データフレーム」「ID 値が 0x0033」のデータを受信した際、「0 番の受信バッファ(*1)」にデータを格納する、という動作となります。

(*1)1 つ目の引数のルール番号=受信バッファ番号という処理としています
（RS-CANFD_Lite では、「受信ルールの番号」と「受信バッファの番号」の縛りはありませんが、本サンプルプログラムでは 1:1 対応としています）

※RS-CANFD_Lite では、受信ルールに適合したデータを、「受信バッファ」と「受信 FIFO(0)」の両方に格納するといった使い方が可能ですが、本サンプルプログラムではデータ格納先を 1 箇所としています

※RS-CANFD_lite では、受信ルールとして、1 つのルールで「標準 ID」及び「拡張 ID」の両方といった設定や、ID が 0x01??（0x0100~0x01FF）のものという設定も可能です。

can0_receive_rule_set()関数では、「両方」や「??（ワイルドカード）」の設定はサポートしていません。
「両方」や「??（ワイルドカード）」設定を行う場合は、GAFLMi レジスタ（マスクレジスタ）に値を設定してください。

受信ルールは、最大 16 個のルールを定義可能です。ルールを定義する場合は、0 番から欠番の出ない様にルールを設定してください。例えば、0,1,2,5,6,7 のルールを設定すると、有効となるのは 0,1,2 のみとなります。

※RS-CANFD_Lite では、受信の最小バイト数(DLC)の指定（設定した DLC 以上のデータのみ受信する）設定が可能ですが、本サンプルプログラムでは DLC はどの値のパケットも受信するように設定しています

```
can0_operate();
```

CAN-ch0 を実行状態とします。

3.2. データ構造体

3.2.1. CAN メッセージ構造体

```
//CANメッセージ構造体
typedef struct{
  unsigned long id;
  unsigned char rtr;
  unsigned char ide;
  unsigned char dlc;
  unsigned char data[8];
  unsigned short ts;
} can_message;
```

CAN のデータは、ID や RTR（データフレーム／リモートフレーム）等をパッケージした構造体でデータをやり取りします。

メンバ	型	説明
id	unsigned long	ID（標準 ID, 11bit または拡張 ID, 29bit）
rtr	unsigned char	CAN_DATA_FRAME(0), CAN_REMOTE_FRAME(1)のどちらか
ide	unsigned char	CAN_ID_FORMAT_SID(0), CAN_ID_FORMAT_EID(1)のどちらか
dlc	unsigned char	データバイト数(1~8)
data	unsigned char[]	送受信データ（最大 8 バイト）
ts	unsigned short	タイムスタンプ（受信データで使用）

実際の CAN のデータの送信を行う際は、本構造体にデータをセットしてから、送信関数を実行するという流れになります。また、受信の際は、構造体にデータが入ってくるという動作となります。

3.2.2. CANFD メッセージ構造体

```
//CANメッセージ構造体
typedef struct{
  unsigned long id;
  unsigned char rtr;
  unsigned char ide;
  unsigned char fdf;
  unsigned char brs;
  unsigned char esi;
  unsigned char dlc;
  unsigned char data[64];
  unsigned short ts;
} can_message;
```

メンバ	型	説明
id	unsigned long	ID（標準 ID, 11bit または拡張 ID, 29bit）
rtr	unsigned char	CAN_DATA_FRAME(0), CAN_REMOTE_FRAME(1)のどちらか
ide	unsigned char	CAN_ID_FORMAT_SID(0), CAN_ID_FORMAT_EID(1)のどちらか
fdf	unsigned char	CAN_DATA_FORMAT(0), CANFD_DATA_FORMAT(1)のどちらか
brs	unsigned char	CAN_BITRATE(0), CABFD_BITRATE(1)のどちらか
esi	unsigned char	CANFD_ERROR_ACTIVE(0), CANFD_ERROR_PASSIVE(1)のどちらか
dlc	unsigned char	データバイト数に応じた DLC 値(1~15) DLC=1~8： データバイト数 1~8 DLC=9,10,11,12,13,14,15： データバイト数:12,16,20,24,32,48,64
data	unsigned char[]	送受信データ（最大 64 バイト）
ts	unsigned short	タイムスタンプ（受信データで使用）

CANFD の場合、FDF, BRS, ESI が追加されており、データサイズも 64 バイトになっています。

3.3. データの送信

3.3.1. 送信バッファを使用したデータ送信

```
can_message msg;

msg.id = 0x0033;
msg.rtr = CAN_DATA_FRAME;
msg.ide = CAN_ID_FORMAT_EID;
msg.dlc = 8;
msg.data[0] = 0x01;
msg.data[1] = 0x23;
msg.data[2] = 0x45;
msg.data[3] = 0x67;
msg.data[4] = 0x89;
msg.data[5] = 0xAB;
msg.data[6] = 0xCD;
msg.data[7] = 0xEF;

can0_txbuf_send(0, &msg);
```

can0_txbuf_send() は、CAN パケットを送信する関数で、送信バッファを使用してデータの送信を行います。

1 つ目の引数は、送信バッファ番号。0~3 の値が指定可能です。
2 つ目の引数は、CAN メッセージ構造体です。

msg.ide
　CAN_ID_FORMAT_SID(0)　標準 ID（11bit の ID）のデータ
　CAN_ID_FORMAT_EID(1)　拡張 ID（29bit の ID）のデータ

msg.rtr
　CAN_DATA_FRAME(0)　データフレーム
　CAN_REMOTE_FRAME(1)　リモートフレーム

msg.id
　ID 値。

msg.dlc
　DLC 値（＝送信バイト数）。

CANFD パケットの送信時は、

```
canfd_message msg;

//ID や RTR の設定は CAN パケット送信時と同じ
msg.fdf = CANFD_DATA_FORMAT;
msg.brs = CANFD_BITRATE;
msg.esi = CANFD_ERROR_PASSIVE
msg.dlc = 15;
msg.data[0] = 0x01;
...
msg.data[63] = 0xFF;

canfd0_txbuf_send(0, &msg);
```

CANFD 固有の設定が追加となり、関数名が canfd0_txbuf_send()となります。
クラシカル CAN では、dlc 値はデータフレーム送信時・リモートフレーム要求時、1~8（1~8 バイト）の値が指定可能。
CANFD フレームでは、1~15（1~8, 12, 16, 20, 24, 32, 48, 64 バイト）の値が指定可能です。

3.3.2. 送受信 FIFO を使用したデータ送信

```
can_message msg;

can0_srfifo_send(&msg);
```

```
canfd_message msg;

canfd0_srfifo_send(&msg);
```

送信バッファを使用した送信の際は、1 つ目の引数として送信バッファの番号を指定しましたが、送受信 FIFO での送信時は、引数はメッセージ構造体のみとなります。

3.4. データの受信

3.4.1. 受信バッファを使用したデータ受信

```
can_message msg;
int ret;

ret = can_rxbuf_receive(0, &msg);

if (ret > 0)
{
  //受信データに応じた処理
```

受信の場合は、
1 つ目の引数は、受信バッファ番号です。
can0_receive_rule_set()で受信ルールを割り当てた番号（本サンプルプログラムでは、ルール番号=受信バッファ番号としている）を指定してください。
2 つ目の引数は、CAN メッセージ構造体です。
戻り値は、受信データに含まれる DLC 値（＝受信したデータバイト数）となります。
戻り値が CAN_RET_NODATA(-1)の時は、データを受信していません。

```
can_message msg;
int ret;
int i;

for (i=0; i<16; i++)
{
  ret = can_rxbuf_receive(i, &msg);

  if (ret > 0)
  {
    //受信データに応じた処理
```

0~15 番の受信バッファを定義している場合、ループ等で受信したい番号を全てスキャンしてください。

受信は、マイコンの CAN モジュールがハード的に受信動作を行って受信バッファにコピーしてあるデータを、can_rxbuf_receive()が取りに行くというイメージです。

can_rxbuf_receive()で、受信データを取りに行く前にデータを受信した場合は、古いデータは新しいデータで上書きされます。（関数の戻り値をみれば、データの上書きが生じたかは判ります。）

※受信バッファは最大で 16 本使用できますが、ルール毎に格納される受信バッファ番号は決まりますので、別ルール（ID が異なる等）のデータを連続して受信した際には、それぞれ別々の受信バッファに格納されます。それに対し、同一ルール（ID、RTR、IDE が同一）のデータを連続して受信した際にはデータの上書きが発生します
（ポーリングでデータ受信処理する場合には、上書きが生じるかどうかはポーリングの周期次第です。）

CANFD メッセージの受信は、

```
canfd_message msg;
int ret;

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

関数名に fd が付く事と引数に CANFD 構造体を指定する事以外は変わりません。戻り値は、DLC 値で 9 以上の値の場合はデータバイト数ではありません。（戻り値は、あくまで DLC 値となります。）

3.4.2. 受信 FIFO を使用したデータ受信

```
can_message msg;
int ret;

ret = can_rxfifo_receive(0, &msg);

if (ret > 0)
{
  //受信データに応じた処理
```

1 つ目の引数は、受信 FIFO 番号です。0 か 1 を指定可能です。
can0_receive_rule_set()で受信ルールを指定した際に、データの格納先を RXFIFO(0)か RXFIFO(1)かを指定する様になっていますので、その番号と同じ番号を指定する必要があります。（本サンプルプログラムでは、RXFIFO(0)のみ使用しています。）

CANFD の場合は、メッセージ構造体と関数名が変わります。

```
canfd_message msg;
int ret;

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

受信 FIFO は、本サンプルプログラムでは、4 段に設定されていますので、4 メッセージまででしたら FIFO に蓄えておくことができます。2 つ以上のメッセージを取り出す際は、can_rxfifo_receive()/canfd_rxfifo_receive()関数を複数回呼び出してください。戻り値、ret == CAN_RET_NODATA(-1)となったら、全てのメッセージの取り出しが完了です。

```
can_message msg;
int ret;

while(1)
{
  ret = can_rxfifo_receive(0, &msg);

  if (ret == CAN_RET_NODATA) break;

  //受信データに応じた処理
}
```

FIFO に複数のデータが受信済みの場合、全てのデータを取り出したい場合は、例えば上記の様な処理となります。

3.4.3. 送受信 FIFO を使用したデータ受信

```
can_message msg;
int ret;

ret = can0_srfifo_receive(&msg);

if (ret > 0)
{
  //受信データに応じた処理
```

送受信 FIFO を使用した受信の場合は、引数はメッセージ構造体のみとなります。

[参考]受信バッファ、受信 FIFO を使用した受信の関数名は can_となっており、送受信 FIFO を使用した関数は can0_となっています。RL78/F24 では CAN は 1ch なので、区別する必要はありませんが、複数 ch をサポートするマイコンの場合は、受信バッファ、受信 FIFO は ch 非依存、送受信 FIFO は ch 依存となるので、この様な関数名としています。

CANFD の場合は、メッセージ構造体と関数名が変わります。

```
canfd_message msg;
int ret;

ret = canfd0_srfifo_receive(&msg);

if (ret > 0)
{
  //受信データに応じた処理
```

複数のデータが FIFO に溜まっている場合の処理は、受信 FIFO と同様の処理としてください。

4. 割り込み処理

SAMPLE1 では、割り込みを使っていません。SAMPLE2~SAMPLE4 では、割り込みを有効にしています。

・使用している割り込み

割り込み関数名	ベクタ名	
intrcangrvc_interrupt	INTRCANGRVC	(1)受信バッファ割り込み（グローバル）
intrcan0err_interrupt	INTRCAN0ERR	(2)チャネルエラー割り込み（ch 毎）
intrcan0cfr_interrupt	INTRCAN0CFR	(3)送受信 FIFO 受信割り込み（ch 毎）
intrcan0trm_interrupt	INTRCAN0TRM	(4)チャネル送信割り込み（ch 毎）
intrcangrfr_interrupt	INTRCANGRFR	(5)受信 FIFO 割り込み（グローバル）
intrcangerr_interrupt	INTRCANGERR	(6)エラー割り込み（グローバル）

割り込みには、ch 非依存（グローバル）割り込みと、ch 毎の割り込みがありますが、RL78/F24 の場合は 1ch しかありませんので、グローバルと ch 毎の区分はあまり気にしなくても良いかと思います。

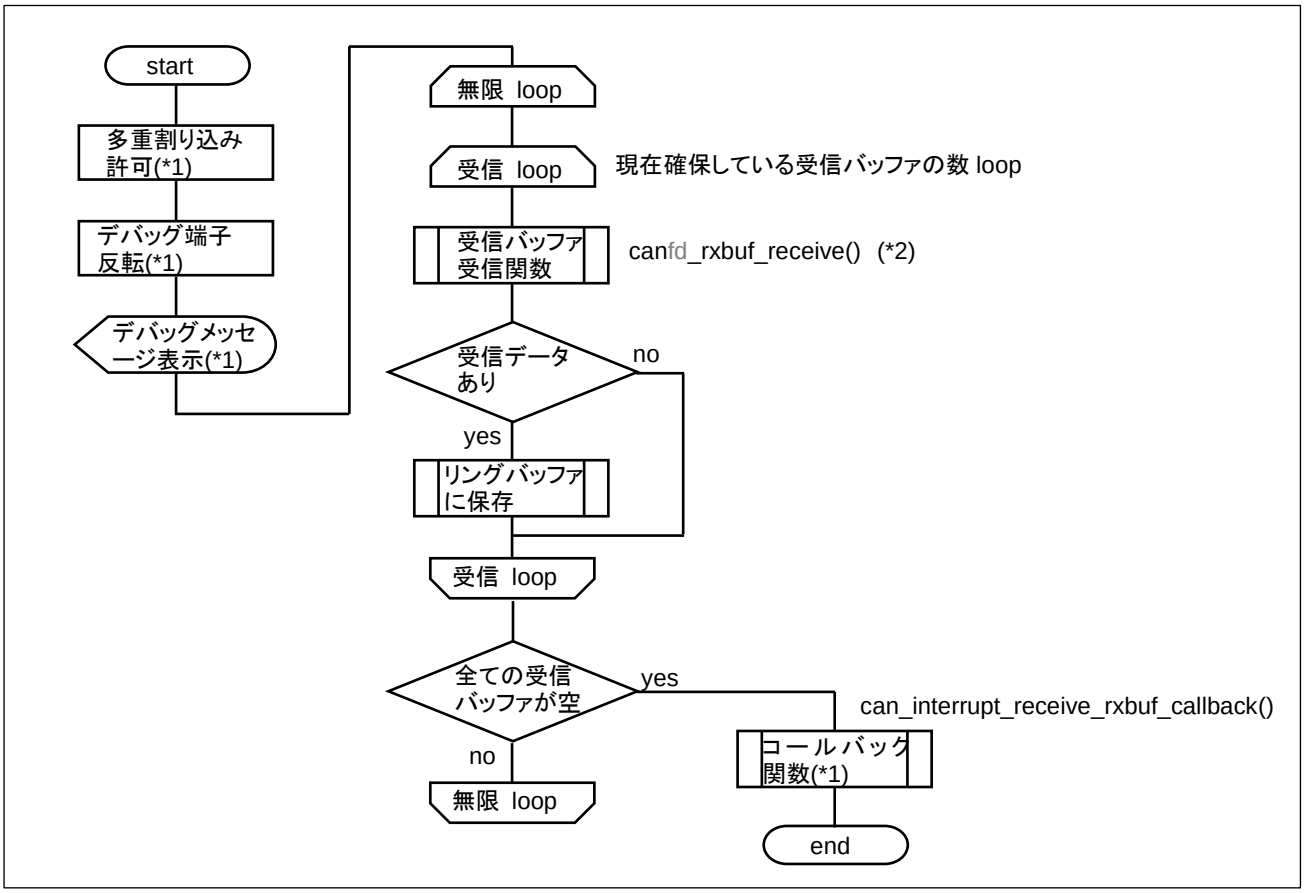
なお、マイコンの CAN 割り込み機能としては、上記の他に
(7)ウェイクアップ割り込み
(8)RAM ECC 割り込み
がありますが、この 2 つは本サンプルプログラムでは未使用です。

割り込み関数本体は、
can_intr.c
内に記載されています。

4.1. 受信バッファ割り込み

受信割り込みでは、有効にした受信バッファ番号をスキャンして受信データがある場合、ユーザが確保したリングバッファにデータをコピーします。

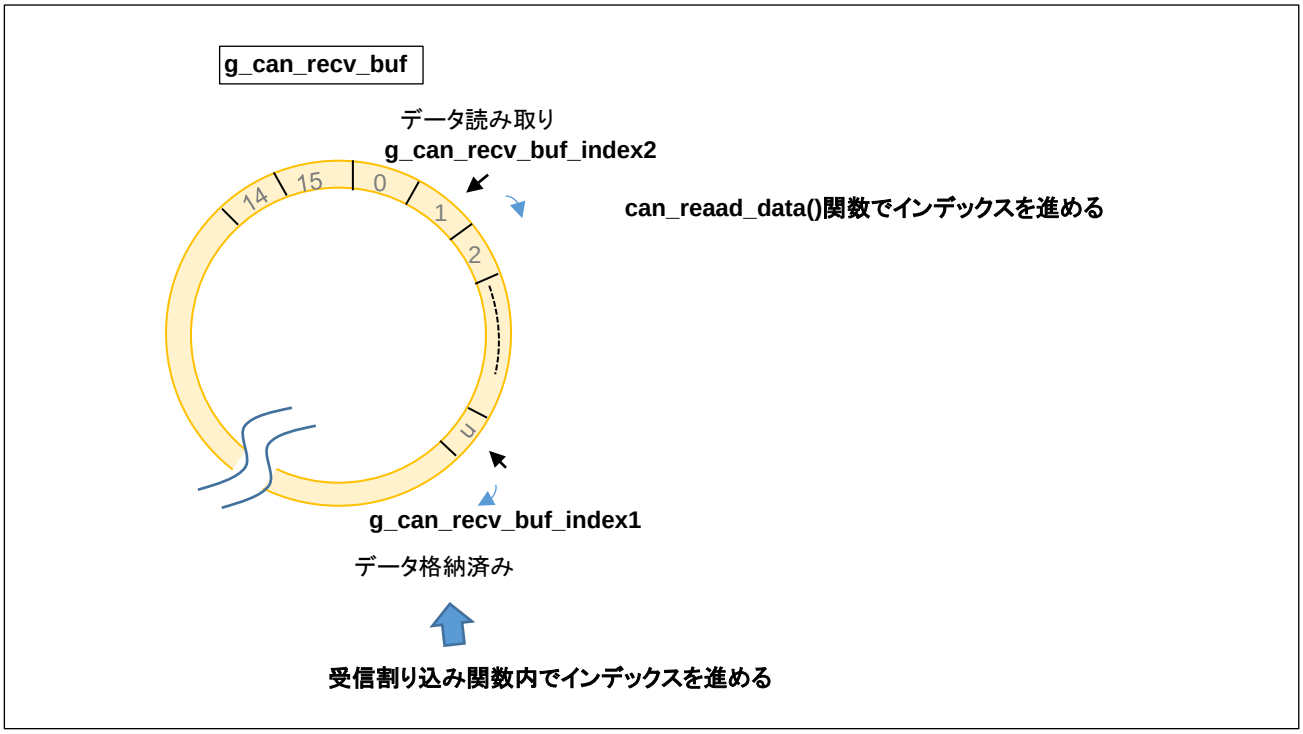
受信バッファ割り込み関数 intrcangrvc_interrupt()



(*1)定数定義有無により「実行」または「実行スキップ」
(*2)CANFD 有効時と無効時で実行する関数を別にしています

※SAMPLE1 では、受信に受信バッファを使っていますが、割り込みは有効化していないので、割り込み関数内の処理をメイン関数(main_s1())内で書き下しています
※SAMPLE2~4 では、デフォルトでは RXFIFO を使って受信するので、本関数は実行されませんが、定数定義で受信バッファで受信する様に設定すれば、本関数が使われます。

ー受信リングバッファのイメージー



受信リングバッファ(g_can_recv_buf)は、デフォルトでは配列数 16 の、can_message 構造体です。
（CANFD 有効時は、canfd_message 構造体。CAN-ch 毎に変数を確保しているので、
g_can_recv_buf[1][16]の 2 次元配列ですが、g_can_recv_buf[0][n]最初のインデックスは CAN-ch 番号で、
RL78/F24 の場合は常に 0 で使われます。）
（16 は、can.h 内で定義しているので、好きな値に変更可能です。）

受信バッファ割り込みで、データを受信すると、受信リングバッファにデータを格納して、格納済みインデックス(g_can_recv_buf_index1)を進めます（インクリメントします）。インデックスが 15 まで進むと、次は 0 に戻ります。

受信リングバッファからデータを取り出す関数、can_read_data()を実行すると、データ読み取りインデックス(g_can_recv_buf_index2)が示すバッファに溜まっているデータを取り出し、インデックスを進めます。

読み取りインデックスが格納済みインデックスに追いついた場合(g_can_recv_buf_index2 == g_can_recv_buf_index1)、読み出しのデータはないと判断されます。

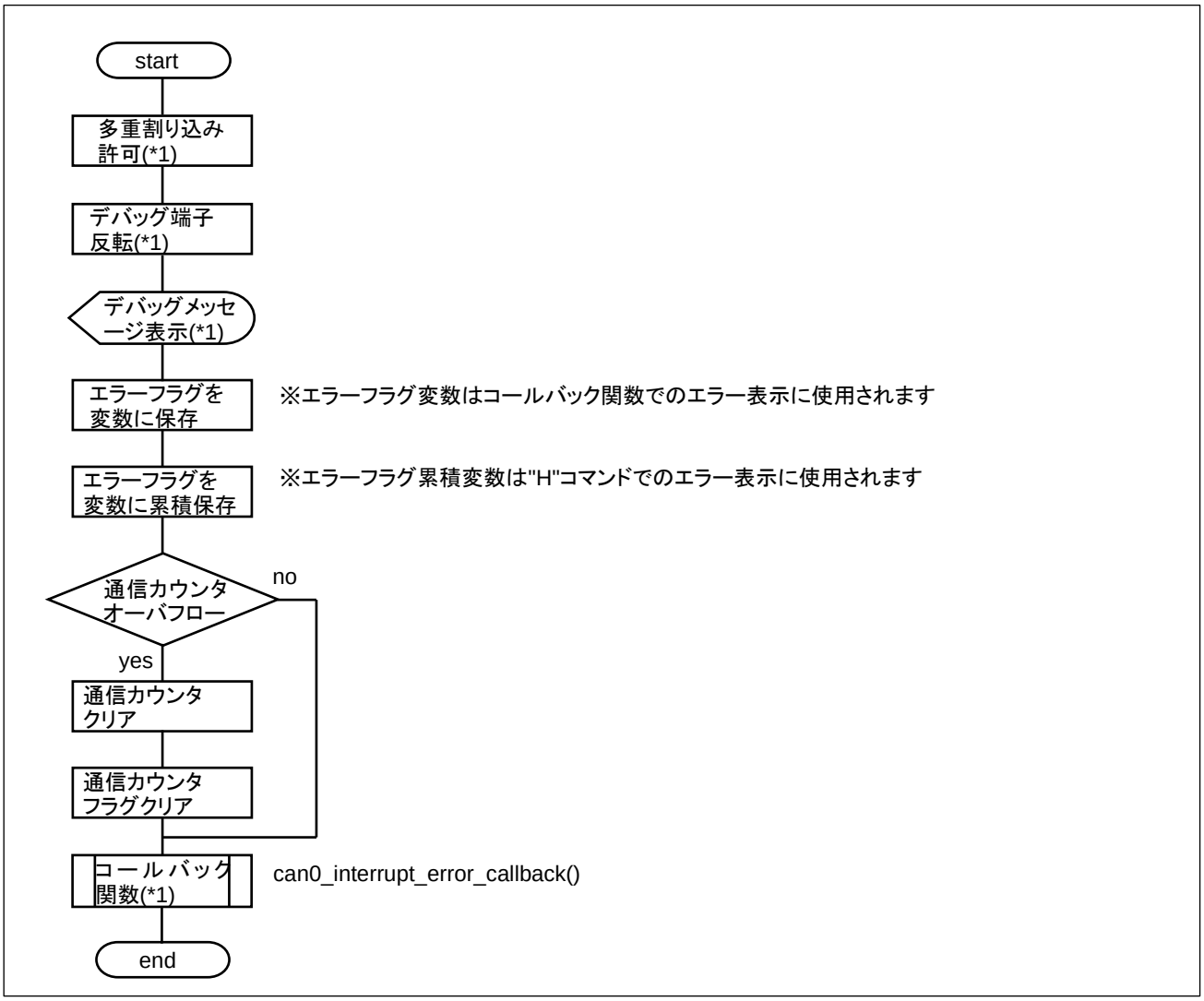
格納済みインデックスが、読み取りインデックスを追い越してしまうと、古いデータは捨てられます。

※本サンプルプログラムで使用しているリングバッファは、16 段の構成ですが、15 個のデータしか保持できません。（1 段無駄があります、処理を単純化するための制約です。気になる方は 16 個のデータを保持できる様に改造してください。）インデックスを uint_32 として、インデックスを 0 に戻さずに単純インクリメントとして、余剰を取れば、単純計算で 100us 毎にデータを受信しても丸 5 日間はインデックス溢れが生じません。（かつ、確保したバッファを全て使用できます。）実装としてはそちらの方が素直かも知れません。

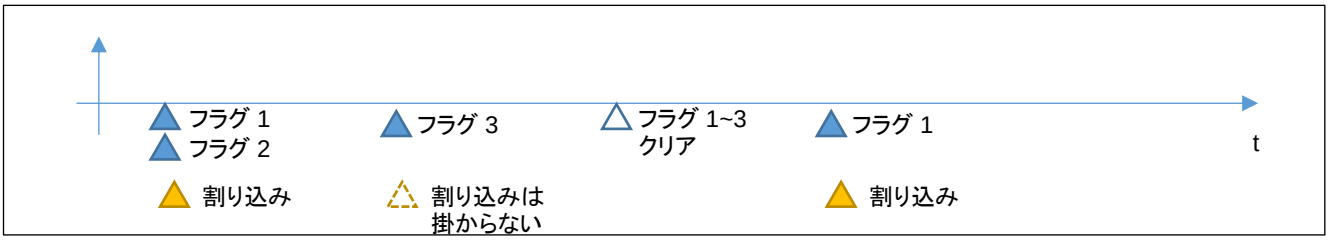
4.2. チャネルエラー割り込み

ch 毎のエラーを検出した際に、本割り込みに飛んできます。どのエラーに対して、割り込みを有効化するかは設定可能ですが、本サンプルプログラムでは全てのエラーに関して、割り込みを有効化しています。

チャネルエラー割り込み関数 intrcan0err_interrupt()



(*1)定数定義有無により「実行」または「実行スキップ」



本割り込みが掛る要因は複数ありますが、割り込みが掛る条件が、割り込みフラグが立っていない状態で、1 つ以上のフラグが立った際に、割り込みが掛ります。

例えば、当該ボードが CAN パケットを送信した際に通信相手が ACK を返さない場合、割り込みの要因となるエラーが発生して、エラー割り込みが掛りますが、この時は
「ACK エラー」「Bus エラー」
が同時に発生して、割り込みが掛ります。その後、時間差で
「エラーパッシブ」「エラーワーニング」「通信カウンタオーバフロー」
が発生しますが、その際は割り込みが掛りません。

・データ送信時に ACK を返す相手が居ない場合の実行例

```
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 dlc=1
data=0x00
*****
- can0_interrupt_error_callback() -
CAN0 error: ACK
CAN0 error: Bus-error
*****
```

→割り込み発生時は、ACK, Bus-error の 2 つのエラーフラグが立っている。

その後、"E"コマンドで、エラー一覧を見た場合

```
-----
CAN error flag register

GERFLL = 0x0000

CANFD payload overflow flag (CMPOF)          : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow is
detected
FIFO message lost flag (MES)                 : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                         : 0 -> NO DLC error is detected

-----
CAN0 error flag register

C0ERFLL = 0x0407

ACK delimiter error flag (ADERR)              : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)  : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                         : 0 -> NO CRC error is detected
ACK error flag (AERR)                         : 1 -> ACK error is detected
Form error flag (FERR)                        : 0 -> NO form error is detected
Stuff error flag (SERR)                       : 0 -> NO stuff error is detected
Arbitration-lost flag (ALF)                   : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                           : 0 -> NO bus-lock is detected
Overload flag (OVLF)                          : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                  : 0 -> NO bus-off recovery is detected
Bus-off entry flag (BOEF)                     : 0 -> NO bus-off entry is detected
Error-passive flag (EPF)                      : 1 -> error-passive is detected
Error-warning flag (EWF)                      : 1 -> error-warning is detected
Bus-error flag (BEF)                          : 1 -> bus-error detected
```

```
-----
CANFD0 error flag register

C0FDSTSL = 0x0100

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation
Successful occurrence counter overflow flag (SOCO)    : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)         : 1 -> overflowed
```

これは、エラー割り込みの時点では、「ACK エラー」「Bus エラー」のみで、ACK を返す相手が居ないのでデータの再送を繰り返す内に、エラーワーニング、エラーパッシブ状態となり、エラーカウンタがオーバフローする動作、時間差で他のエラーが出るというものです。

[参考]割り込み関数内での処理に関して

割り込み関数内では、次の割り込みのために、割り込み要因フラグをすべてクリアしてから、割り込みルーチンを抜けるのが基本的なフローとなります。（受信割り込みでは、そのようなフローとしています。）

ACK を返す相手が居ないというエラー等の場合は、割り込みルーチン内で割り込みフラグをクリアして割り込みルーチンを抜けると、直ぐに同じエラーで割り込みが掛ります。割り込みがループで何度も呼ばれる状態となります。そのため、本サンプルプログラムでは、割り込み関数内で割り込みフラグのクリア処理は行わずに割り込みルーチンを抜ける様にしています。

本サンプルプログラムでは、

(1)エラー要因を取り除く（ACK を返す相手が居ない状態で再送を繰り返している場合は、「ACK を返す相手を接続する」もしくは、「"a"コマンドで送信を取りやめる」）

(2)"C"コマンドでエラーフラグをクリアする

上記のフローで、再度エラー割り込みが掛る状態となります。

－通信成功カウンタオーバフローに関して－

エラー割り込みの中で、通信成功カウンタオーバフロー(SOCO)に関しては、エラーではないので割り込みルーチン内で、フラグとカウンタクリアの処理を入れています。通信成功カウンタオーバフローは、256 でオーバフローするので、256 回通信が成功した時点で、割り込みが掛ります。次の 256 回目でも同様です。これは本来エラーではありません。

can_intr.h 内で

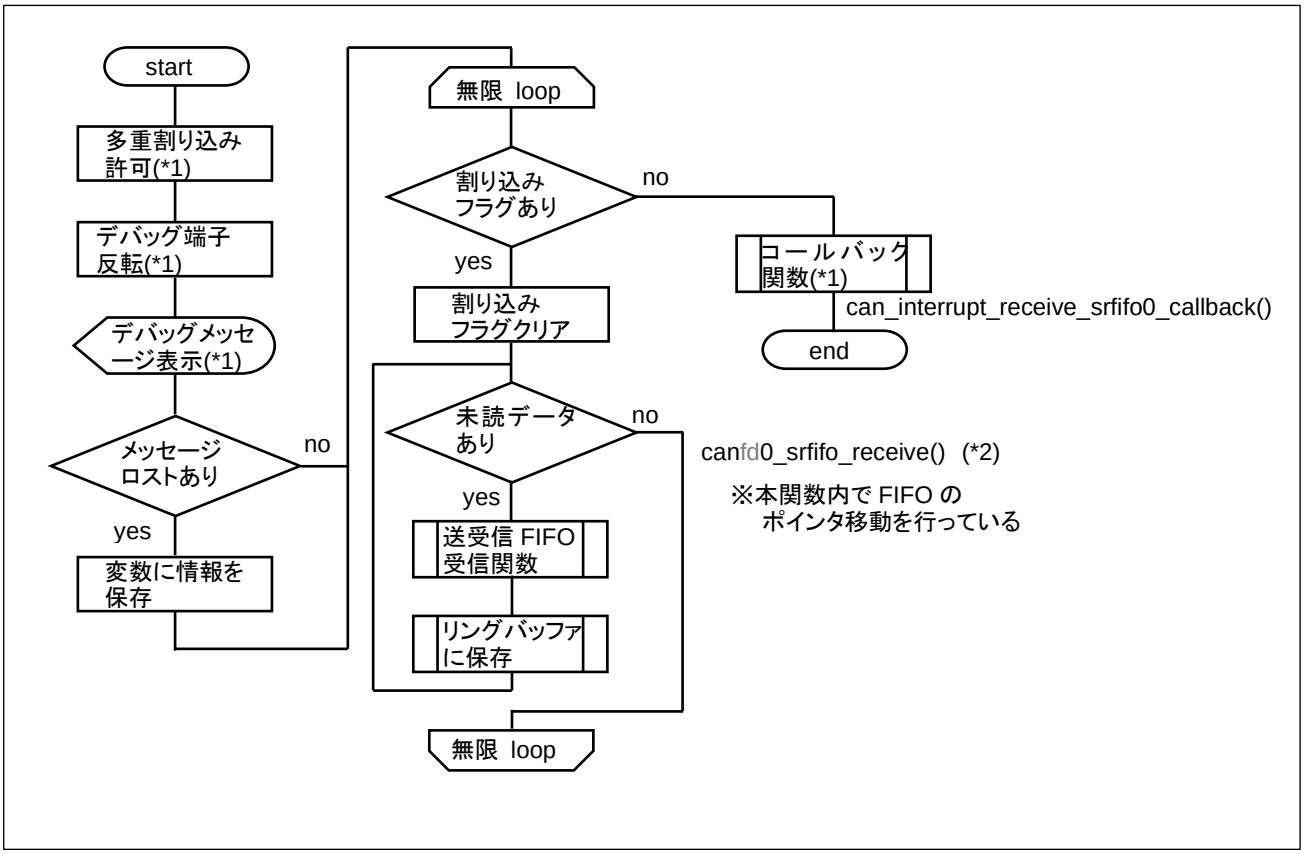
```
//通信カウンタオーバフロー割り込みを抑止する
/*
通信カウンタ(SOC)は通信成功時にインクリメントされ、0xFF(=255)を超えた場合に
チャネルエラー割り込みが発生する
通信カウンタのオーバフローはエラーではないので
下記変数定義時は、オーバフロー割り込みを生じさせない事とする
*/
//#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE
```

#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE を有効化すれば、通信成功カウンタオーバフローは割り込み対象から外す事が出来ます。

4.3. 送受信 FIFO 受信割り込み

送受信 FIFO に格納済みのデータを、ユーザが確保したリングバッファにデータをコピーします。

送受信 FIFO 受信割り込み関数 intrcan0cfr_interrupt()



(*1)定数定義有無により「実行」または「実行スキップ」
(*2)CANFD 有効時と無効時で実行する関数を別にしています

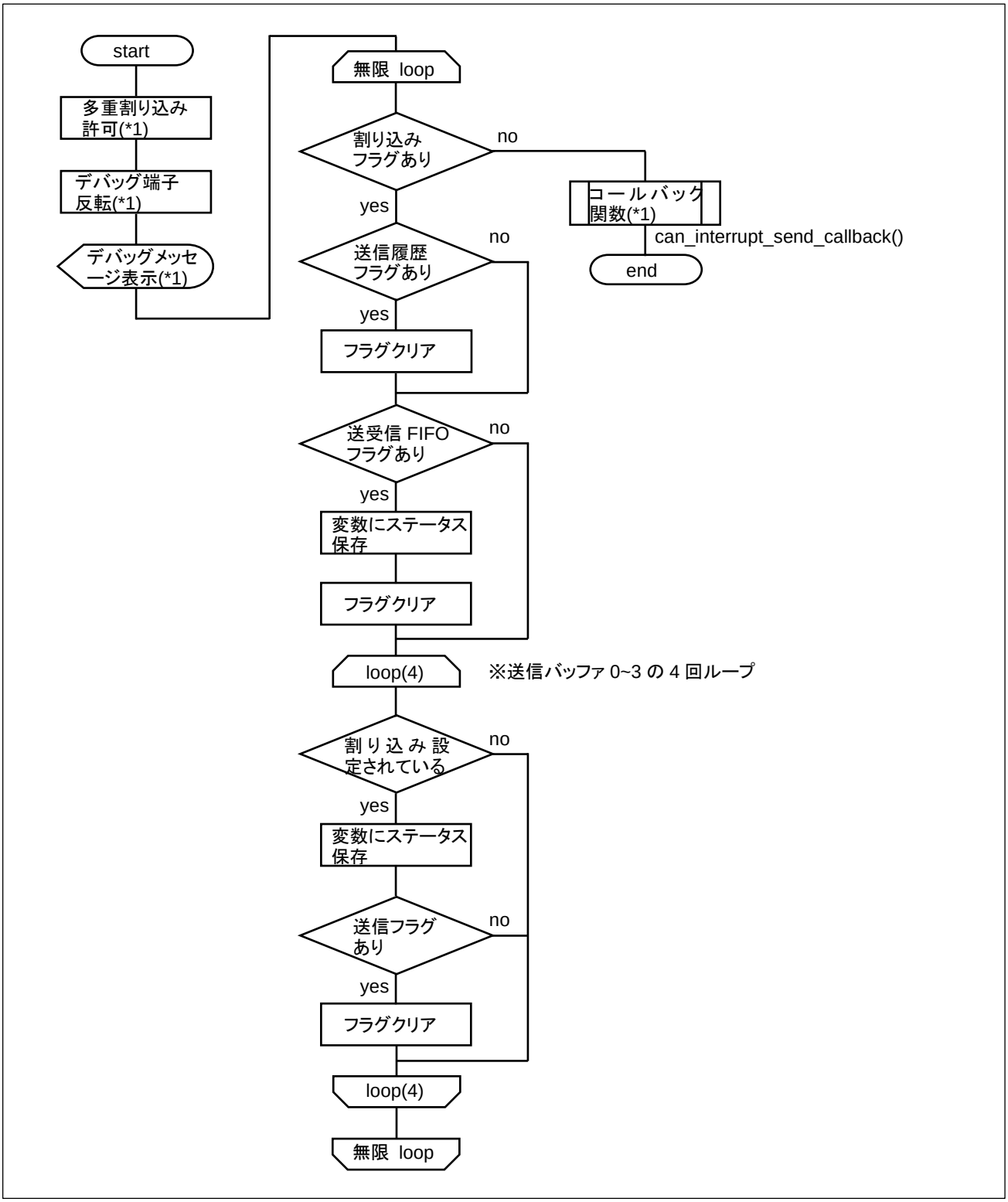
メッセージロスト（本サンプルプログラムでは FIFO を 4 段で構成していますが、データ読み出し（送受信 FIFO 受信関数呼び出し）前に、5 個以上のデータを受信した際には、5 個目以降のデータは捨てられるという動作となります。その際、メッセージロストフラグが立ち、グローバルエラー割り込みの対象となるのですが、本割り込み関数内でメッセージロストフラグを落としてしまうので、グローバルエラー割り込みは掛かりません。その代わりに、メッセージロストフラグの履歴をエラー履歴変数に残します。エラー履歴は、"H"コマンドで確認が可能です。

※FIFO 段数は最大 16 段とすることが出来ます（can_ch0.h 内で設定）

4.4. チャネル送信割り込み

送信完了した際に掛かる割り込みです。

チャネル送信割り込み関数　intrcan0trm_interrupt()



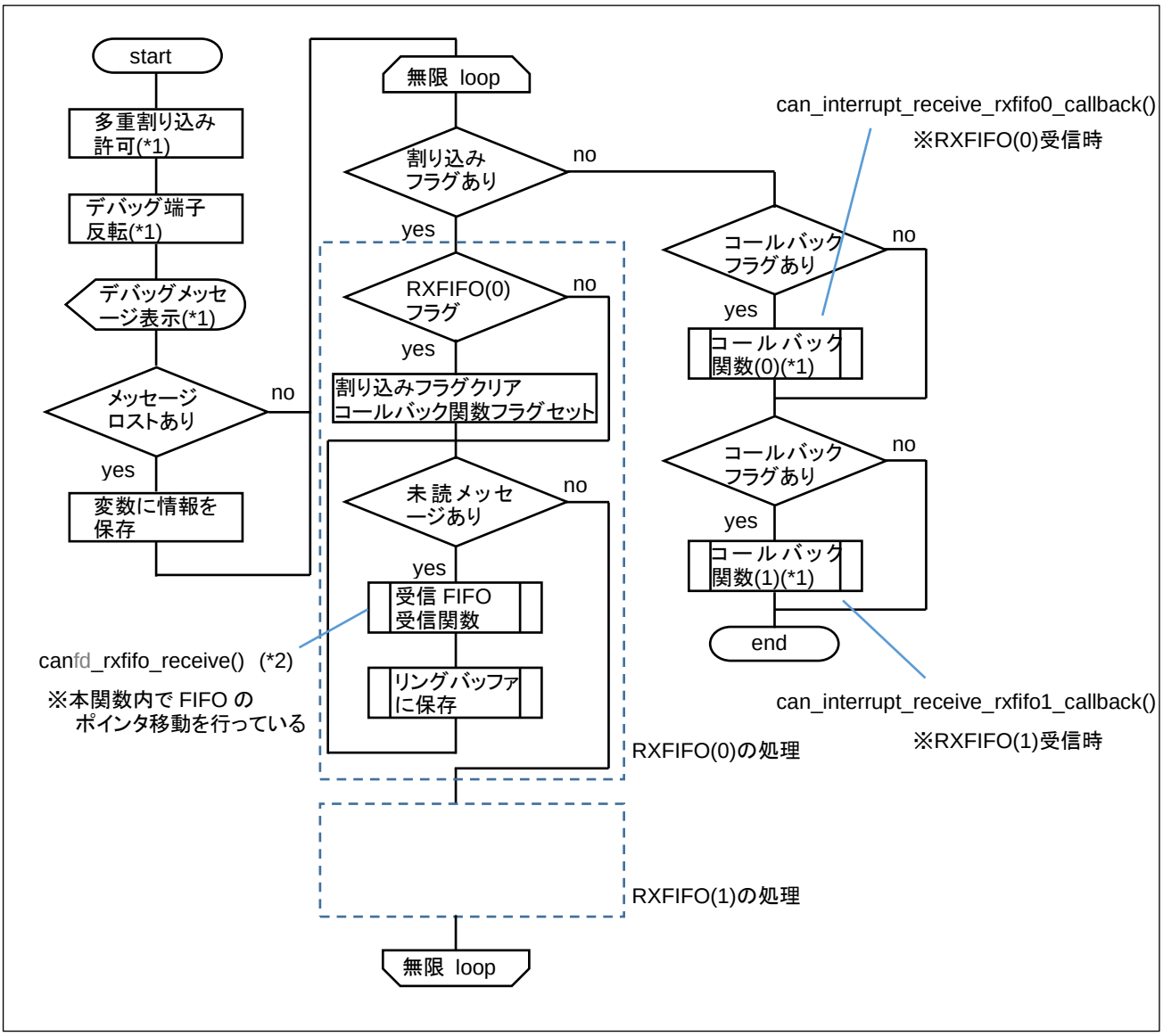
(*1)定数定義有無により「実行」または「実行スキップ」

送信割り込み関数は、「送受信 FIFO」を使用した送信と、「送信バッファ」を使用した送信、どちらも同じ割り込み関数に飛んできます。

4.5. 受信 FIFO 割り込み

受信 FIFO に格納済みのデータを、ユーザが確保したリングバッファにデータをコピーします。

受信 FIFO 割り込み関数　intrcangrfr_interrupt()



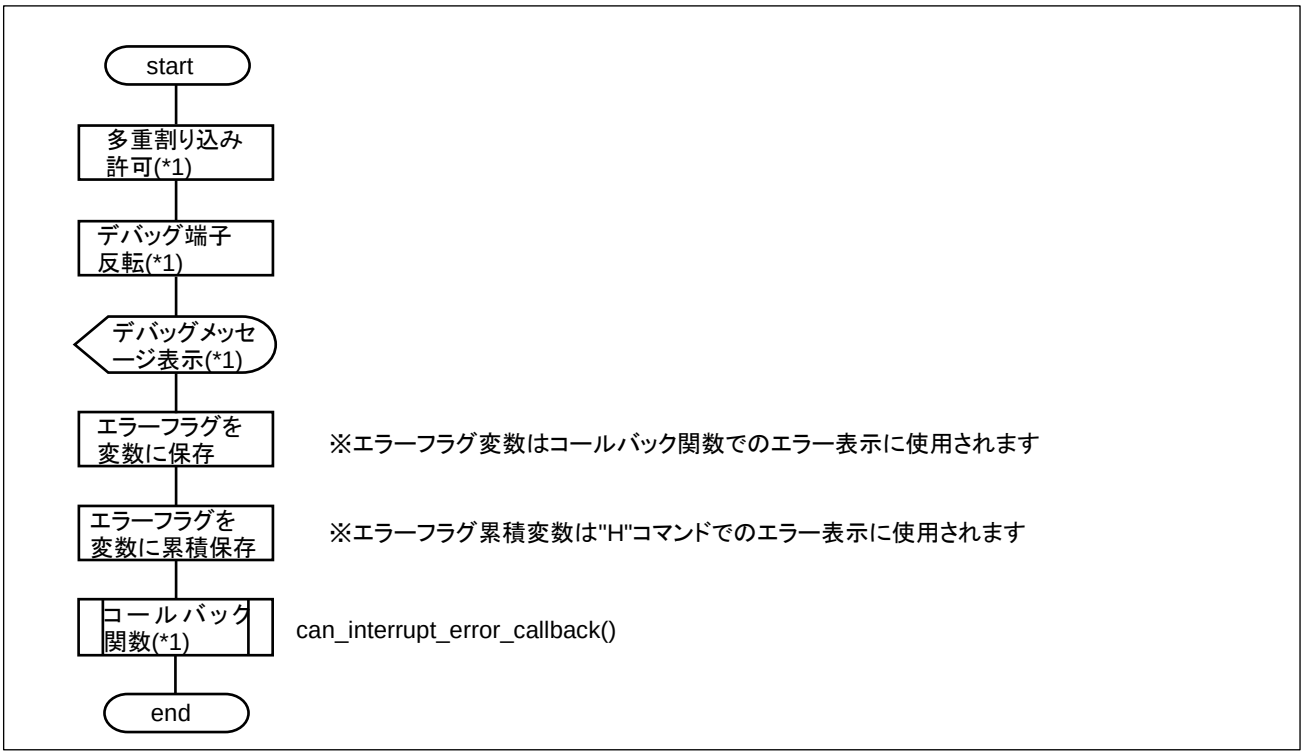
(*1)定数定義有無により「実行」または「実行スキップ」
(*2)CANFD 有効時と無効時で実行する関数を別にしています

RXFIFO(0)と RXFIFO(1)は同じ割り込み関数に飛んできますので、RXFIFO(0)と RXFIFO(1)の両方の受信処理を本割り込み関数内で行っています。

4.6. エラー割り込み

ch 非依存のエラーを検出した際に、本割り込みに飛んできます。どのエラーに対して、割り込みを有効化するかは設定可能ですが、本サンプルプログラムでは全てのエラーに関して、割り込みを有効化しています。

エラー割り込み関数 intrcangerr_interrupt()



(*1)定数定義有無により「実行」または「実行スキップ」

5. サンプルプログラムの説明(共通部分)

5.1. クロックと通信速度

サンプルプログラムの初期化（クロックの設定等）は、スマート・コンフィグレータ生成コードを使用しています。

対象マイコンボード	搭載 水晶振動子 (fMX)	PLL 逓倍	CPU クロック	周辺クロック (fCLK)	CAN クロック (fCAN)
HSBRL78F24-100	8MHz	8×10 (=80MHz)	40MHz	40MHz	fCLK =40MHz

RL78/F24 は、CAN のクロックソース(fCAN)として

fCLK(40MHz)

fMX(8MHz)

のいずれかを選択可能です。本サンプルプログラムでは、fCLK を選択しています。

5.1.1. CAN（CANFD 未使用時）の速度設定

1Mbps 設定時、fCAN(=40MHz)を 4 分周して fCANTQ=10MHz としてビットクロックとしています。1Tq=100ns で、10Tq で 1 ビットとする設定です。

CAN は、送信側と受信側がそれぞれ自分自身のクロックで動作しますので（送信側のクロックで生成された波形を、受信側のクロックでサンプリングする）、CAN のタイミングのベースとなるクロックは、周波数精度の良い水晶振動子ベースのクロックを使用する事が推奨されます。

PLL の入力を水晶振動子クロック(fMX)とした場合、fCLK, fMX のどちらを使用しても水晶振動子ベースのクロックとなります。

本キットのサンプルプログラムでは、CAN の通信速度は 1Mbps に設定してあります。1Mbps では、1 ビットの時間が 1us となります。（定義ファイルで、500kbps, 250kbps, 125kbps が選択可能です）

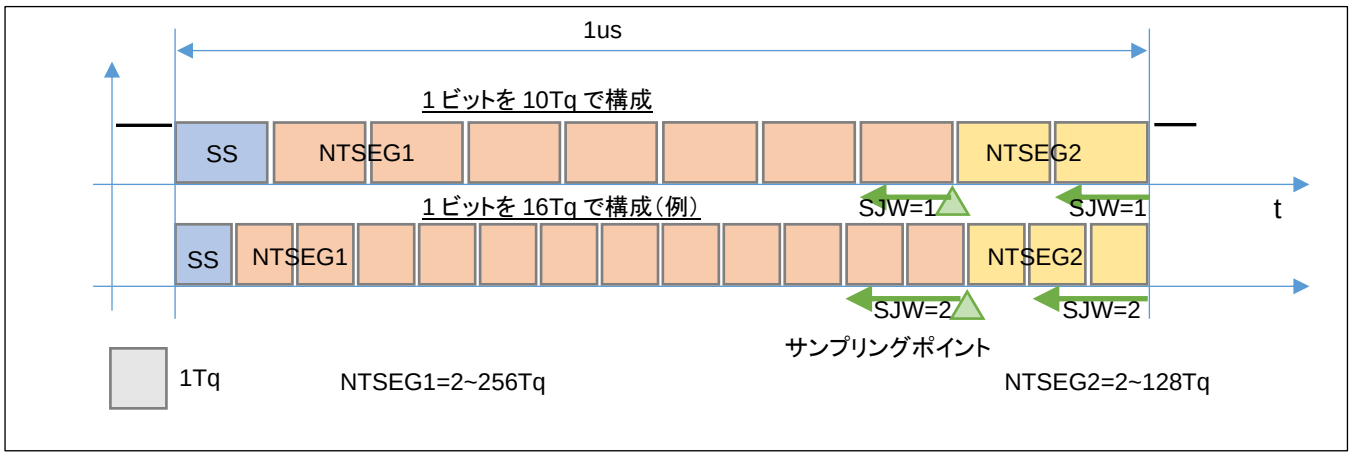


図 5-1 CAN データの 1 ビット

CAN の 1 ビットデータは、複数の Tq（Time Quantum, データの 1 ビットより短いタイミング基準）で構成されます。
1 ビットは、8Tq~385Tq で設定する必要があり、NTSEG1>NTSEG2>=SJW を満たす必要があります。
1 ビットを 1us とするためには、10Tq で 1 ビットを構成する場合、1Tq=100ns（10MHz のクロック源が必要）となります。

※サンプリングポイント=(1+NTSEG1)/(1+NTSEG1+NTSEG2)
※SJW:リシンクロナイゼーション ジャンプ幅（セグメントを延長・短縮する補正幅）

本ボードの fCAN は、fCLK(40MHz)としていますので、1Mbps 時 1 ビットを 40Tq で構成する事も可能ですが、C0NCFGL レジスタの NBRP=3（4 分周）として、10Tq で 1 ビットを構成する設定としています。（500kbps, 250kbps, 125kbps 設定時は、1 ビット 10Tq の設定は維持し BPR の値（分周比）を調整しています。）

・本サンプルプログラムでの設定値（1Mbps 設定時）

マイコン種	搭載水晶振動子 fMX	クロックソース fCAN（周波数）	分周比 [BRP]（分周後のクロック）	1Tq	NTSEG1	NTSEG2	SJW	サンプルポイント
RL78/F24	8MHz	fCLK(40MHz)	4(10MHz) (*1)	100ns	7	2	1	80%

(*1)500kbps 設定時は 8, 250kbps 設定時は 16, 125kbps 設定時は 32 に設定しています

クロックソース（周波数）	分周[BRP]（分周後のクロック）	通信速度	1Tq	DTSEG1	DTSEG2	SJW	サンプルポイント
CANFDCLK (40MHz)	4(10MHz)	1Mbps	100ns	7	2	1	80%
	8(5MHz)	500kbps	200ns				
	16(2.5MHz)	250kbps	400ns				
	32(1.25MHz)	125kbps	800ns				

5.1.2. CANFD 使用時の速度設定

CANFD では、通信速度が高速になる分高速なクロックをベースとする（1Tq の時間を小さくする）必要があります。
CANFD においても、最初は従来の CAN の通信速度で通信を開始するので、CANFD の場合は
・CANFD での TSEG1, TSEG2(DTSEG1, DTSEG2)
・CAN での TSEG1, TSEG2(NTSEG1, NTSEG2)
の 2 系統の速度設定を行う必要があります。

例えば、CANFD の通信速度を 5Mbps、CAN の通信速度を 1Mbps に設定する場合、CANFD での分周比設定と、CAN での分周比設定は別々の値を指定できますので、分周比で 1:5 の差を付けるという手法が考えられます。

(1)8Tq で 1 ビットを構成する

	通信速度	クロックソース（周波数）	分周比[BRP]（分周後のクロック）	1Tq	TSEG1	TSEG2	サンプルポイント
CANFD	5Mbps	fCAN (40MHz)	1(40MHz)	25ns	5	2	80%
CAN	1Mbps		5(8MHz)	125ns	5	2	80%

この場合、TSEG1 と TSEG2 の設定値は CANFD と CAN で同じ値となります。

但し、この様な設定は、推奨されていない設定となります。CANFD パケットにおいては、データの途中で通信速度が変化するので、CANFD と CAN のデータを同じクロックソースで取り扱う事がハードウェアマニュアルで推奨されています。

(2)分周比を CAN と CANFD で同一とする

	通信速度	クロックソース（周波数）	分周比[BRP]（分周後のクロック）	1Tq	TSEG1	TSEG2	サンプルポイント
CANFD	5Mbps	fCAN (40MHz)	1(40MHz)	25ns	5	2	80%
CAN	1Mbps				29	10	75%

TSEG1 と TSEG2 の値で、通信速度を変える設定で、本キットでは、こちらの設定を採用します。

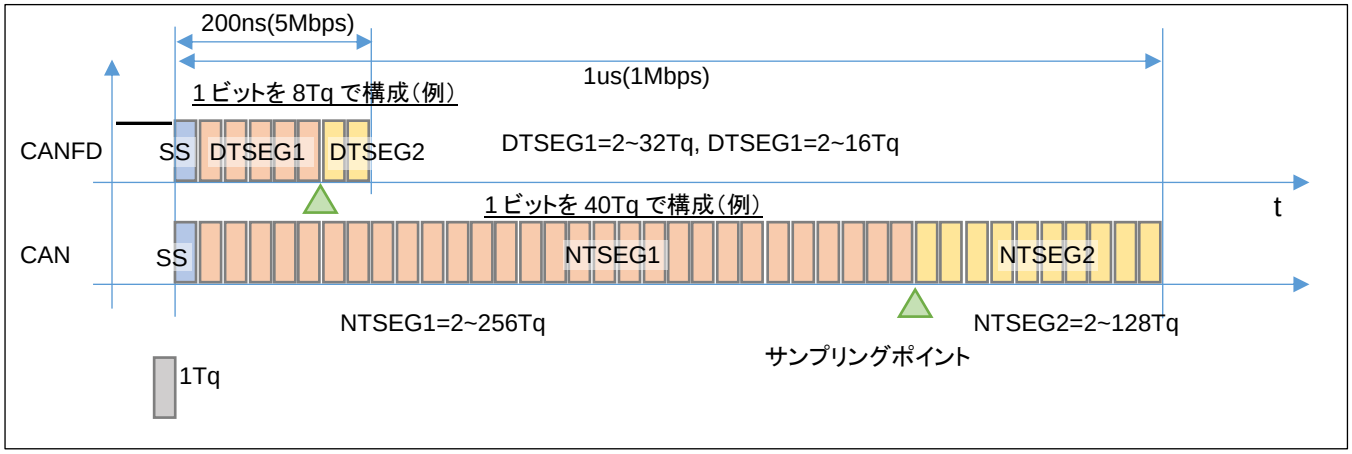


図 5-2 CANFD/CAN データの 1 ビット

CANFD の 1 ビットは、5Tq~49Tq で設定する必要があり、DTSEG1>=DTSEG2>=SJW を満たす必要があります。

CAN の 1 ビットは、8Tq~385Tq で設定する必要があり、NTSEG1>NTSEG2>=SJW を満たす必要があります。

CANFD と CAN の 1Tq を同じ値とする事が推奨されています。

遅延補償を有効にする場合は、fCAN を分周しない事が求められています。

・本サンプルプログラムでの CANFD 速度設定値

クロックソース（周波数）	分周[DBRP]（分周後のクロック）	通信速度	1Tq	DTSEG1	DTSEG2	SJW	サンプルポイント
CANFDCLK (40MHz)	1(40MHz)	5Mbps	25ns	5	2	1	80%
		4Mbps		6	3	1	70%
		3.3Mbps		8	3	1	75%
		2Mbps		14	5	1	75%

CANFD の速度設定は、テーブル化されています。

・本サンプルプログラムでの CANFD 速度設定値（CANFD 有効時）

NTSEG1, NTSEG1, SJW の値は、can_timing_parameter()関数で計算されます。サンプリングポイントは、75%に近い値となる様、SJW は NTSEG2 の 1/4 程度の値となる様に調整されます。

[参考]プログラムでは下記の値が選ばれます

クロックソース（周波数）	分周[DBRP]（分周後のクロック）	通信速度	1Tq	DTSEG1	DTSEG2	SJW	サンプルポイント
CANFDCLK (40MHz)	1(40MHz)	1Mbps	25ns	29	10	2	75%
		500kbps		59	20	5	75%
		250kbps		119	40	10	75%
		125kbps		239	80	20	75%

5.2. ヘッダファイルの設定

5.2.1. can.h

can.h

```
/*-----------------------------------------------------------------------
定数定義
-----------------------------------------------------------------------*/
#if defined(SAMPLE4)
#define CANFD_MODE  //SAMPLE4ではCANFDモードとする
#endif

//通信速度, [kbps]単位
#define CAN_BPS_1M    1000 //1M bps
#define CAN_BPS_500K  500  //500k bps
#define CAN_BPS_250K  250  //250k bps
#define CAN_BPS_125K  125  //125k bps

//通信速度（データ）, [kbps]単位
#define CANFD_BPS_5M    5000  //5M bps
#define CANFD_BPS_4M    4000  //4M bps
#define CANFD_BPS_3_3M  3333  //3.3M bps
#define CANFD_BPS_2M    2000  //2M bps

//パラメータ有効/無効設定
#define ENABLE  1
#define DISABLE  0

//CANFD 第2サンプルポイント設定
#define CANFD_MEASURE_OFFSET  0  //測定値＋オフセット値
#define CANFD_OFFSET_ONLY     1  //オフセット値のみ

//ID
#define CAN_ID_FORMAT_SID  0  //標準ID(11bit)フォーマットを使用
#define CAN_ID_FORMAT_EID  1  //拡張ID(29bit)フォーマットを使用

//RTR
#define CAN_DATA_FRAME    0  //データフレーム
#define CAN_REMOTE_FRAME  1  //リモートフレーム

//FDF
#define CAN_DATA_FORMAT    0  //CANデータフォーマット
#define CANFD_DATA_FORMAT  1  //CANFDデータフォーマット

//BRS
#define CAN_BITRATE    0//CANビットレート
#define CANFD_BITRATE  1//CANFDビットレート

//ESI
#define CANFD_ERROR_ACTIVE   0//エラーアクティブ
#define CANFD_ERROR_PASSIVE  1//エラーパッシブ
```

(a)動作モード定義

青枠内はプログラムで使用している定数値なので変更しないでください

can.h（続き）

```
//受信ルール
#defineCAN_RULE_RXBUF     0x0     //受信メッセージバッファ
#defineCAN_RULE_RXFIFO0   0x0001  //RXFIFO0
#defineCAN_RULE_RXFIFO1   0x0002  //RXFIFO1
#defineCAN_RULE_SRFIFO0   0x0100  //SRFIFO0

//CANの受信方法
#define CAN_RX_RXBUF   0
#define CAN_RX_SRFIFO  1
#define CAN_RX_RXFIFO  2

//CANの送信方法
#define CAN_TX_TXBUF   0
#define CAN_TX_SRFIFO  1

//CAN-ch
#define CAN_CH0  0  //RL78/F24ではch0のみ

//送信完了フラグ
#define CAN_SEND_FLAG_SRFIFO               0x0010  //送受信FIFOで送信完了
#define CAN_SEND_FLAG_SENDBUF              0x0020  //送信バッファで送信完了
#define CAN_SEND_FLAG_SENDBUF_WITH_ABORT   0x0030  //送信バッファで with abort
#define CAN_SEND_FLAG_SENDBUF_ABORTED      0x0040  //送信バッファで aboated

//戻り値
#define CAN_RET_SUCCESS          0  //成功
#define CAN_RET_NODATA          -1  //データなし
#define CAN_RET_ARGUMENT_ERROR  -2  //引数エラー
#define CAN_RET_IN_USE          -3  //使用中エラー
#define CAN_RET_OVERFLOW        -4  //FIFOフルエラー
#define CAN_RET_MODE_ERROR      -5  //関数が実行可能なモードではない
#define CAN_RET_VALUE_ERROR     -6  //値が不正
#define CAN_RETFLAG_LOST_DATA   0x80  //失われたデータあり（戻り値とorを取る）

//デバッグ用
//#define SCI_DEBUG  //定義時SCI表示によるデバッグを行う

//受信バッファサイズ
#define CAN_RECV_BUF_SIZE  16  //can_message 構造体のバッファ数のメモリを消費
```

青枠内はプログラムで使用している定数値なので変更しないでください

(b)SCI デバッグ表示の選択

(c)受信バッファサイズの選択

(a)動作モード定義

#define CANFD_MODE

本定数定義時は、CANFD が有効となります。受信データ等、基本的なデータは CANFD 構造体側が選択され、CANFD パケットとして処理されます。

SAMPLE1~3 では未定義、SAMPLE4 では定義となります。

(a)SCI デバッグ表示の選択

```
//#define SCI_DEBUG
```

SCI_DEBUG 定義時

割り込み時等に端末にメッセージが追加で表示される様になります。

SCI_DEBUG 未定義時（デフォルト）

割り込み時等に端末にメッセージが追加で表示されません

(b)受信リングバッファサイズの設定

```
#define CAN_RECV_BUF_SIZE   16
```

CAN_RECV_BUF_SIZE は、受信のリングバッファの容量です。（SAMPLE2 以降、受信割り込みルーチン内で受信データを受信リングバッファに保存します。）

5.2.2. can_operation.h

can_operation.h

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

//速度設定
#define CAN_SPEED  CAN_BPS_1M

//いずれかを設定
//CAN_BPS_1M(1000)//1M bps
//CAN_BPS_500K(500)//500k bps
//CAN_BPS_250K(250)//250k bps
//CAN_BPS_125K(125)//125k bps
//※
//クラシカルCAN動作時（CANFD_MODE未定義時）
//→1000の1/n(n:整数)の数値のみ指定可能
//
//CANFD動作時（CANFD_MODE定義時）
//#define CAN_SPEED 400 （400kbpsの意）の様に任意の速度を指定する事も可能
//任意の値が指定可能だが、1Tq=25ns(1/40MHz)の倍数とならない場合は速度誤差が生じる
//1Tq=25ns時の指定可能な下限は、104kbps程度（100kbpsは指定不可）
//→1Tq=25nsを大きな値とすれば、100kbps未満の値も指定可能となる

//通信速度（データ）※CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_SPEED  CANFD_BPS_2M

//いずれかを設定
//CANFD_BPS_5M(5000)//5M bps
//CANFD_BPS_4M(4000)//4M bps
//CANFD_BPS_3_3M(3300)//3.3M bps
//CANFD_BPS_2M(2000)//2M bps
//※
//CANFD_SPEEDは任意の値設定不可（タイミングパラメータはテーブル指定のため、定義値のみ指定可能）

//CANFD トランシーバ遅延補償 ※CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_TDCE  DISABLE

//CANFD 第2サンプルポイント ※CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_SSP  CANFD_MEASURE_OFFSET

//CANFD トランシーバ遅延補償 オフセット値 ※CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_TDCO  0

//CANFD トランシーバRXエッジフィルタ ※CANFD_PARAMETER_SETTING 未定義時有効
#define CANFD_REFE  DISABLE

//ID設定
#define CAN_ID_TYPE  CAN_ID_FORMAT_EID

//いずれかを設定
//CAN_ID_FORMAT_SID//標準ID(11bit)フォーマットを使用
//CAN_ID_FORMAT_EID//拡張ID(29bit)フォーマットを使用
```

(a)通信速度の選択

(b)通信速度の選択(CANFD)

(c)CANFD 通信パラメータ

(d)ID 設定

can_operation.h（続き）

```
//受信と送信方法の選択（SAMPLE1-SAMPLE4のサンプルプログラム毎に選択）

#if defined(SAMPLE1)

//SAMPLE1の設定

//受信：受信バッファ，割り込みは未使用
#define CAN_RX_METHOD      CAN_RX_RXBUF
#define CAN_RX_INTERRUPT   DISABLE

//送信：送信バッファ，割り込みは未使用
#define CAN_TX_METHOD      CAN_TX_TXBUF
#define CAN_TX_INTERRUPT   DISABLE

//エラー：エラー割り込み未使用
#define CAN_ERROR_INTERRUPT   DISABLE

#elif defined(SAMPLE2) || defined(SAMPLE3) || defined(SAMPLE4)

//SAMPLE2-SAMPLE4の設定

//受信：受信FIFO，割り込みを使用
#define CAN_RX_METHOD      CAN_RX_RXFIFO//デフォルト（他の受信方法を選択しても問題ありません）
//#define CAN_RX_METHOD   CAN_RX_SRFIFO
//#define CAN_RX_METHOD   CAN_RX_RXBUF
#define CAN_RX_INTERRUPT   ENABLE

//送信：送受信FIFO，割り込みを使用
#define CAN_TX_METHOD      CAN_TX_SRFIFO//デフォルト（他の送信方法を選択しても問題ありません）※受信と送信の両方をSRFIFOとする事は不可
//#define CAN_TX_METHOD   CAN_TX_TXBUF
#define CAN_TX_INTERRUPT   ENABLE

//エラー：エラー割り込み使用
#define CAN_ERROR_INTERRUPT   ENABLE

#else

//SAMPLE1-4未定義時

#define CAN_RX_METHOD              CAN_RX_RXFIFO
#define CAN_RX_INTERRUPTENABLE

#define CAN_TX_METHOD              CAN_TX_SRFIFO
#define CAN_TX_INTERRUPTENABLE

#define CAN_ERROR_INTERRUPTENABLE

#endif

#if (CAN_RX_METHOD == CAN_RX_SRFIFO) && (CAN_TX_METHOD == CAN_TX_SRFIFO)

#error "Error: SRFIFO select error"

#endif
//受信FIFO番号
#define CAN0_RX_RXFIFO_NO  0//CAN0 受信を RXFIFO で行う場合の FIFO 番号(0-1 を指定)
[CAN_RX_METHOD -> CAN_RX_RXFIFO 時使用]
```

SAMPLE1 での設定

(e)受信方法の選択

(f)受信割り込みの使用

(g)送信方法の選択

(h)送信割り込みの使用

(i)エラー割り込みの使用

SAMPLE2~4 での設定

(e)受信方法の選択

(f)受信割り込みの使用

(g)送信方法の選択

(h)送信割り込みの使用

(i)エラー割り込みの使用

SAMPLE1~4 以外での設定

(j)受信 FIFO 番号の選択

(a)通信速度の選択（CAN のビットレート）
#define CAN_SPEED　CAN_BPS_1M

基本的には、1Mbps~125kbps の 4 種類が選択可能です。

#define CAN_SPEED 200 //200kbps に設定する場合、kbps 単位で値を設定

CANFD 未使用時は、1Mbps を基本として、1Mbps の 1/n の速度であれば予め定義されている 4 種類以外の値でも指定可能です。（CANFD 未使用時は、分周比を調整して通信速度を決定します。）

CANFD 使用時は、104kbps 以上 1Mbps 以下の値を指定可能ですが、25ns(1Tq)で割り切れない数値の場合は、速度誤差が生じます。（CANFD 使用時は、1Tq=25ns をベースに通信速度を決定します。）
（CANFD 使用時で、100kbps 以下の速度設定としたい場合は、1Tq=25ns の変更が必要です。その場合、
（1Tq=50ns 以上となり）CANFD ビットレート 5Mbps 等は選択できなくなります。）

※CANFD を使用するかどうかは、
program_select.h 内で
#define CANFD_MODE
が有効化されている場合(SAMPLE4)は、CANFD を使用。CANFD_MODE 定数未定義時は、CANFD 未使用となります。

(b)通信速度の選択（CANFD のビットレート）
#define CANFD_SPEED　CANFD_BPS_2M

CANFD は、SAMPLE4 で有効になりますが、SAMPLE4 のデフォルトの動作は、起動時にビットレートをキーボードから入力する動作となりますので、ここで設定した値は使用されません。main_s4.c 内で、対話入力を無効化（#define CANFD_PARAMETER_SETTING をコメントアウト）した場合、ここでの設定が有効になります。

基本的には、5M, 4M, 3.3M, 2M の 4 種類が選択可能です。（その他の速度としたい場合は、DTSEG の設定値を計算して設定を行う必要があります。）

(c)CANFD 通信パラメータ

CANFD は、SAMPLE4 で有効になりますが、SAMPLE4 のデフォルトの動作は、起動時に通信パラメータをキーボードから入力する動作となりますので、ここで設定した値は使用されません。（対話入力を無効化した場合に、ここでの設定が有効になります。）

#define CANFD_TDCE　DISABLE

トランシーバ遅延補償 「DISABLE」（無効）または「ENABLE」（有効）

#define CANFD_SSP　CANFD_MEASURE_OFFSET

第 2 サンプリングポイント 「CANFD_MEASURE_OFFSET」（測定値＋オフセット値）または「CANFD_OFFSET_ONLY」（オフセット値のみ）

#define CANFD_TDCO　0

トランシーバ遅延補償オフセット値「0～255」

#define CANFD_REFE　DISABLE

トランシーバ RX エッジフィルタ 「DISABLE」（無効）または「ENABLE」（有効）

(d)ID 設定

#define CAN_ID_TYPE　CAN_ID_FORMAT_EID

「CAN_ID_FORMAT_EID」CAN の ID を拡張 ID(29bit)と標準 ID(11bit)両方取り扱う（拡張 ID と標準 ID 両方のデータを受信）（デフォルトは拡張 ID で送信）
「CAN_ID_FORMAT_SID」CAN の ID を標準 ID(11bit)のみ取り扱う（標準 ID のデータしか受信しない）

(e)受信方法の選択

#define CAN_RX_METHOD　CAN_RX_RXBUF

「CAN_RX_RXBUF」受信バッファで受信（SAMPLE1 のデフォルト）、「CAN_RX_RXFIFO」受信 FIFO で受信（SAMPLE2~4 のデフォルト）、「CAN_RX_SRFIFO」送受信 FIFO で受信の 3 種類の受信方法から選択。

※RS-CANFD_Lite の機能としては、1 つの受信データを複数の受信先に格納する事も可能ですが、本サンプルプログラムでは、複数の受信先の指定はサポートされていません。

(f)受信割り込みの使用

#define CAN_RX_INTERRUPT　DISABLE

「DISABLE」（無効）または「ENABLE」（有効）
SAMPLE1 では「DISABLE」、SAMPLE2~4 では「ENABLE」の設定です。

(g)送信方法の選択

```
#define CAN_TX_METHOD    CAN_TX_TXBUF
```

「CAN_TX_TXBUF」送信バッファを使用して送信（SAMPLE1 のデフォルト）、「CAN_TX_SRFIFO」送受信 FIFO を使用して送信（SAMPLE2~4 のデフォルト）、のどちらか。

※送受信 FIFO は、「受信」「送信」どちらかの用途で使用可能です。受信を「送受信 FIFO」で受信する様に設定した場合は、送信を「送受信 FIFO」で行う事はできません。

(h)送信割り込みの使用

```
#define CAN_TX_INTERRUPT    DISABLE
```

「DISABLE」（無効）または「ENABLE」（有効）
SAMPLE1 では「DISABLE」、SAMPLE2~4 では「ENABLE」の設定です。

(i)エラー割り込みの使用

```
#define CAN_ERROR_INTERRUPT    DISABLE
```

「DISABLE」（無効）または「ENABLE」（有効）
SAMPLE1 では「DISABLE」、SAMPLE2~4 では「ENABLE」の設定です。

(j)受信 FIFO 番号の選択

```
#define CAN0_RX_RXFIFO_NO    0
```

受信方法として、受信 FIFO を選択した場合に使用されます。「0」RXFIFO(0)を使用、または「1」RXFIFO(1)を使用のどちらかです。

※サンプルプログラムでは RXFIFO(0)を使う設定です（RXFIFO(1)は未使用です）

5.2.3. can_intr.h

can_intr.h

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

//4種類の割り込みの使用対応
#if (CAN_RX_INTERRUPT == ENABLE)      //受信割り込みの設定
#if (CAN_RX_METHOD == CAN_RX_RXFIFO)
#define CAN_RXFIFO_INTERRUPT  ENABLE  //RXFIFO割り込みを使用
#elif (CAN_RX_METHOD == CAN_RX_SRFIFO)
#define CAN_SRFIFO_INTERRUPT  ENABLE  //SRFIFO割り込みを使用
#elif (CAN_RX_METHOD == CAN_RX_RXBUF)
#define CAN_RXBUF_INTERRUPT  ENABLE   //RXBUF割り込みを使用
#endif//CAN_RX_METHOD
#endif//CAN_RX_INTERRUPT

#if (CAN_TX_INTERRUPT == ENABLE)      //送信割り込みの設定
#define CAN_TXBUF_INTERRUPT  ENABLE  //TXBUF割り込みを使用（SRFIFOを使用して送信した場合でも、送信割り込みは「送信バッファ」割り込みとなる）
#endif//CAN_TX_INTERRUPT

#if (CAN_ERROR_INTERRUPT == ENABLE)         //エラー割り込みの設定
#define CAN_CH_ERROR_INTERRUPT  ENABLE      //チャネルエラー割り込みを使用
#define CAN_GLOBAL_ERROR_INTERRUPT  ENABLE //グローバルエラー割り込みを使用
#endif//CAN_ERROR_INTERRUPT

//割り込み優先度
#define CAN_RXBUF_INTERRUPT_PRIORITY1//受信バッファの受信割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）
#define CAN_CH_ERROR_INTERRUPT_PRIORITY1//チャネルエラー割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）
#define CAN_SRFIFO_INTERRUPT_PRIORITY1//SRFIOの受信割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）
#define CAN_TXBUF_INTERRUPT_PRIORITY1//送信割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）
#define CAN_RXFIFO_INTERRUPT_PRIORITY1//RXFIOの受信割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）
#define CAN_GLOBAL_ERROR_INTERRUPT_PRIORITY1//グローバルエラー割り込み優先度，0-3を指定（0:最高優先度，3:最低優先度）

//CANの多重割り込みを許可
//#define CAN_MULTIPLE_INTERRUPT_EN
/*
CANの割り込み同士で多重割り込みを使用する場合は、上記割り込み優先度に差を付ける必要がある
*/

//通信カウンタオーバフロー割り込みを抑止する
/*
通信カウンタ(SOC)は通信成功時にインクリメントされ、0xFF(=255)を超えた場合に
チャネルエラー割り込みが発生する
通信カウンタのオーバフローはエラーではないので
下記変数定義時は、オーバフロー割り込みを生じさせない事とする
*/
//#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE
```

どの受信割り込みを使用するか、基本的に変更不要

送信割り込みの有効化、基本的に変更不要

2 種類のエラー割り込み
どちらかを無効化するのでなければ、変更不要

(a)割り込み優先度

(b)多重割り込み許可

(c)通信カウンタ割り込み無効化

can_intr.h（続き）

```
//割り込みコールバック関数の有効化
#define CAN_USE_SEND_CALLBACK_FUNCTION//送信割り込み関数のコールバック関数を使用する
#define CAN_USE_RECEIVE_CALLBACK_FUNCTION//受信割り込み関数のコールバック関数を使用する
#define CAN_USE_ERROR_CALLBACK_FUNCTION//エラー割り込み関数のコールバック関数を使用する
                                        (d)割り込みコールバック関数の有効化

//デバッグ用
#define CAN_INTERRUPT_DEBUG//定義時端子による割り込みデバッグを行う
                                        (e)端子デバッグの有効化
```

(a)割り込み優先度の設定

```
#define CAN_RXBUF_INTERRUPT_PRIORITY    1
#define CAN_CH_ERROR_INTERRUPT_PRIORITY    1
#define CAN_SRFIFO_INTERRUPT_PRIORITY    1
#define CAN_TXBUF_INTERRUPT_PRIORITY    1
#define CAN_RXFIFO_INTERRUPT_PRIORITY    1
#define CAN_GLOBAL_ERROR_INTERRUPT_PRIORITY    1
```

受信バッファ，チャネルエラー，送受信 FIFO（受信），送信，受信 FIFO，エラーの 6 種類の割り込みの優先度の設定。「0~3」の値が指定可能です。（0:最高優先度，3:最低優先度）

(b)多重割り込み許可

```
//#define CAN_MULTIPLE_INTERRUPT_EN
```

定義を有効にすると、割り込み関数の先頭で、多重割り込み許可(PSW.IE=1)を実行します。

※例えば、優先度 3 の割り込み処理中に優先度 0~2 の割り込みが入った場合、かつ多重割り込みを有効にしている場合、後から実行された割り込みが先に処理されます。多重割り込みを有効にする場合は、本定数を定義する事に加え、割り込み優先度の値に差を付けて（例えば受信を 2，エラーを 1 など）おく必要があります。
（優先度 0 に設定した場合、かつ多重割り込みを有効化した場合は、後から実行された割り込みの優先度が 0 の場合（同じレベルの割り込み）でも多重割り込みで処理されます。）

(c)通信カウンタ割り込み無効化

```
//#define CAN_SOC_OVERFLOW_INTERRUPT_DISABLE
```

通信成功カウンタ（CAN のパケットの送受信に成功した場合、カウンタがインクリメントされる）がオーバフローした場合の割り込みの有効、無効を制御します。デフォルトは有効で、本定数定義時は無効化します。

通信成功カウンタは、8 ビットカウンタなので 256 回通信が成功するとオーバフローして、チャネルエラー割り込みが入ります。本来、通信成功カウンタのオーバフローは、エラーではないので、無効化しても問題ありません。

(d)割り込みコールバック関数の有効化

割り込み関数が呼ばれた際に、ユーザ側で処理を追加したい場合、本定数を定義すると、特定のコールバック関数が呼ばれる様になります。

#define CAN_USE_SEND_CALLBACK_FUNCTION

上記定義有効、かつ送信割り込みが有効の場合

void can_interrupt_send_callback(void)

#define CAN_USE_RECEIVE_CALLBACK_FUNCTION

上記定義有効、かつ受信 FIFO 割り込みが有効の場合
void can_interrupt_receive_rxfifo0_callback(void)　RXFIFO(0)の割り込み時
void can_interrupt_receive_rxfifo1_callback(void)　RXFIFO(1)の割り込み時

上記定義有効、かつ送受信 FIFO（受信）割り込みが有効の場合
void can0_interrupt_receive_srfifo0_callback(void)

上記定義有効、かつ受信バッファ割り込みが有効の場合
void can_interrupt_receive_rxbuf_callback(void)

#define CAN_USE_ERROR_CALLBACK_FUNCTION

上記定義有効、かつエラー割り込みが有効の場合

void can0_interrupt_error_callback(void)　チャネルエラー割り込み時
void can_interrupt_error_callback(void)　エラー割り込み時（グローバルエラー）

それぞれ、上記のコールバック関数が呼ばれます。これらの関数は、ユーザ作成のプログラムコード内に作成してください。

(e)端子デバッグ有効化
#define CAN_INTERRUPT_DEBUG

本定数有効化時は、割り込みルーチンの先頭で特定の端子を反転させる処理を行います。オシロ等で、割り込みのタイミングを確認したい場合に有効化してください。（デフォルト：有効）

5.2.4. board.h

board.h

```
//CANのポート(CAN0)
#define CAN0_TX_P10
#define CAN0_RX_P11
#define CAN0_EN_P12

//LEDのポート設定
#define LED_PORT_NUM     1
#define LED1_PORT_INIT  PM0_bit.no3=0;
#define LED1_PORT       P0_bit.no3

#define LED_ON  1
#define LED_OFF  0

//SWのポート設定
#define SW_PORT_NUM  1
//#define SW1_PORT_INIT  //入出力モードの切り替え不要
#define SW1_PORT  P13_bit.no7

#define CAN_MACROCAN_MACRO_TYPE_RSCANFD_LITE  //CANモジュール種別（プログラム上は未使用）

#define CAN_CH  1  //マイコンが持つCAN最大チャネル数

#define CAN0_VALID 1  //CAN0を使用する（使用する場合1，未使用の場合0）

#define ICLK  40  //CPUクロック [MHz]
#define FCLK  40  //ベースクロック(fCLK) [MHz]
#define FCAN  40  //CANのベースクロックfCAN=fCLK/1 [MHz]
#define XTAL  8  //搭載XTALの値[MHz]
#define CAN_CLOCK  FCLK  //FCLK:FCLK側をCANクロックとして使用，(XTAL:XTAL側をCANクロックとして使用)（"FCLK"か"XTAL"のいずれか）
#define CAN_CK_DIV_BASE  4  //1Mbps時 fCAN/4(=10MHz)をCANクロックに設定
#define CANFD_CK_DIV_BASE  1  //CANFD動作時 fCAN/1(=40MHz)をCANクロックに設定

//割り込みモニタ端子
//J2-1(P90)受信バッファ受信割り込み
//J2-2(P91)CAN0 送受信FIFO受信割り込み
//J2-3(P92)CAN0 チャネル送信割り込み
//J2-4(P93)受信FIFO割り込み
//J2-5(P94)エラー割り込み（グローバルエラー,チャネルエラー）
#define INT1_PORT_INITPMC9_bit.no0=0;PM9_bit.no0=0;
#define INT2_PORT_INITPMC9_bit.no1=0;PM9_bit.no1=0;
#define INT3_PORT_INITPMC9_bit.no2=0;PM9_bit.no2=0;
#define INT4_PORT_INITPMC9_bit.no3=0;PM9_bit.no3=0;
#define INT5_PORT_INITPMC9_bit.no4=0;PM9_bit.no4=0;

#define INT1_PORTP9_bit.no0
#define INT2_PORTP9_bit.no1
#define INT3_PORTP9_bit.no2
#define INT4_PORTP9_bit.no3
#define INT5_PORTP9_bit.no4
```

CAN の使用端子の設定

LED の使用ポートの設定

SW の使用ポートの設定

(a)クロックの設定

割り込みデバッグ端子の設定

HSBRL78F24-100 を使用する限りでは基本的には変更不要

board.h は、ボード固有の設定なので、HSBRL78F24-100 を使用する限りでは、基本的には変更不要です。

(a)クロック設定

```
#define CAN_CLOCK    FCLK
```

CAN のベースクロック(fCAN)の設定。「FCLK」か「XTAL」のどちらか。CANFD 未使用時は、XTAL を選択する事も可能です。CANFD 使用時は、PLL ベースの FCLK を選択する事となると思います。基本は、FCLK で問題ありません。

```
#define CAN_CK_DIV_BASE    4
```

CANFD 未使用時の、1Mbps 選択時の基本分周比です。FCLK=40MHz で、10Tq で 1bit を構成する場合、4 となります。（40MHz/4=10MHz, 10MHz, 10Tq → 1us...1Mbps）

```
#define CANFD_CK_DIV_BASE    1
```

CANFD 使用時の基本分周比です。CANFD 使用時は、40MHz/1=40MHz, 1Tq=25ns を基本的なクロックとしています。

5.2.5. prohram_select.h

program_select.h

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

//プログラム選択 [いずれか1つのみ選択]

//SAMPLE1
//割り込みを使用しない、データフレームの送受信サンプルプログラム
//#define SAMPLE1

//SAMPLE2
//割り込みを使用する、データフレームの送受信サンプルプログラム
//#define SAMPLE2

//SAMPLE3
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム
//#define SAMPLE3

//SAMPLE4
//割り込みを使用する、データフレーム／リモートフレームの送受信サンプルプログラム, CANFD版
#define SAMPLE4

//ーーここまで プログラム選択
```

2.2 章に記載がありますが、4 種類のサンプルプログラム(SAMPLE1~SAMPLE4)の選択ファイルです。

#define SAMPLEn

の行を 1 つのみ有効化してください。

5.3. 初期化の処理

初期化の処理としては、以下の関数で以下の処理を行っています。

・CAN モジュールのリセット
can_reset()

CAN モジュールのイネーブル
CAN クロックの供給
グローバルリセットモード遷移
受信リングバッファのクリア

・CAN の割り込み設定
can_interrupt_setup()

割り込み優先度の設定
割り込みフラグクリア
割り込みマスク解除

・ch 毎の初期化処理
can*n*_init()　　(n=0)

ポート設定
チャネルリセットモード遷移
クロック分周比、TSEG 値設定
受信 FIFO 設定（必要に応じて）
送受信 FIFO 設定（必要に応じて）

・エラー割り込みの有効化
can_error_interrupt_enable()
can_ch_error_interrupt_enable(0)

グローバルエラー割り込みの有効化
チャネルエラー割り込みの有効化

・受信ルール設定
can_reveive_rule_conf()

受信ルール数の定義

RL78/F24 では定義可能な受信ルールは最大 16 なので、本関数では受信ルール数を 16 に設定しています。

本関数では、AFL（アクセプタンス・フィルタ・リスト）の設定を行っています。本サンプルプログラムでは、PNF は未使用としています。

・受信バッファ設定

can_receve_buf_conf()

受信バッファ数の定義

割り当て可能なバッファ数 16 から、送信と受信に使用する FIFO の数を差し引いた値を受信バッファ数としています。

SAMPLE1 では、FIFO 未使用なので、受信バッファに 16 割り当て。SAMPLE2~4 では、受信 FIFO と送受信 FIFO をそれぞれ 4 段ずつ使用しているので、受信バッファ数は 8 です。（SAMPLE2~4 で、送信に送受信 FIFO、受信に受信バッファを使用した際の受信バッファ数は 12 です。）

・CAN 動作

can_operate()

CAN モジュールを動作モードに移行させます。

・受信ルール設定

can*n*_receive_rule_set　（n=0)

受信ルールの設定

(1)受信手段（受信 FIFO，送受信 FIFO，受信バッファ）
(2)標準・拡張 ID 区分
(3)RTR（データフレームかリモートフレームか）
(4)ID 値

の設定を行います。(2)(3)(4)は完全一致したデータのみ受信します。RS-CANFD_Lite モジュールの動作としては、1 つのルールで「RTR=0/1 の両方受信する」「ID の上位 7 ビットを比較対象とし下位 4 ビットはどんな値でも受信する」等の動作が可能ですが、本関数では設定した値と一致する事を受信の条件としています。

受信ルールは、16 ルールまでです。

使用例:

can0_receive_rule_set(0, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x123);

CAN0 で(RL78/F24 では CAN0 のみ)、ルール番号 0 に、「標準 ID」「データフレーム」「ID 値は 0x123」の条件のデータを受信した際、RXFIFO の 0 番にメッセージを格納する、というルールを設定する。

※ルール設定時は、0 番から埋めていき途中で欠番が出ない様に設定してください
OK: ルール 0,1,2,3 を設定
NG: ルール 0,4,5,6 を設定(この場合はルール 4,5,6 は有効になりません)
(ルール 0, ルール 3, ルール 2, ルール 1 の順で設定する事は問題ありません。最終的に途中欠番がなければ OK です)

・CAN 動作(ch 毎)
can*n*_operate() (n=01)

CAN ch-0 を動作モードに移行させます。

上記実行後、データの送信及び受信が可能になります。

5.4. 端末から使用可能なコマンド

・データフレーム送信コマンド(0,1,2,3)

データフレームを送信します。

コマンド	ID	送信バイト数	送信データ
0	0x00000000	1	0x01
1	0x00000001	2	0x0123
2	0x00000002	4	0x01234567
3	0x00000003	8	0x0123456789ABCDEF

SAMPLE4 では、送信バイト数が、1, 8, 16, 64 バイト、送信データが 0x00 からインクリメントした値となります。

・リモートフレーム送信コマンド(q,w,e,r)

リモートフレームを送信します。

コマンド	ID	DLC 値
q	0x00000000	1
w	0x00000001	2
e	0x00000002	4
r	0x00000003	8

SAMPLE3~4 で有効です。
SAMPLE4 では、DLC 値が、1, 8, 10, 15 となります。

・ボード識別コマンド(z)

接続されているボードの LED が点滅します。端末とどのボードがつながっているかを識別するコマンドです。

・送信 ID フォーマット変更コマンド(s)

送信データの ID フォーマットが、拡張 ID←→標準 ID トグルで切り替わります。

・データ送信停止コマンド(a)

ACK を返す相手が居ない場合、データ送信が繰り返されますが、本コマンドでデータ送信を停止します。

```
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 dlc=1 data=0x00
*****                                                          0 コマンドを入力
- can0_interrupt_error_callback() -
CAN0 error: Stuff
CAN0 error: Bus-error                  ACK を返す相手が居ないので Bus-error となる
*****
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 c  0 コマンドを入力
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 c  0 コマンドを入力
++
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 c  0 コマンドを入力
++
CAN0 data frame send, ret=-4   0 コマンドを入力
++                                                  送信キューに 4 データ溜まって
CAN0 data frame send, ret=-4   0 コマンドを入力      いるので、データ送信が受け付
send abort!                                         けられずにエラーとなる
++                             a コマンドを入力
CAN0 data frame send, ret=0 id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 c  0 コマンドを入力
```

送信に送受信 FIFO（4 段）を使用して、0 コマンドを連続で入力した場合の表示例を示します。

"a"コマンドを入力する前は、データ送信が繰り返されている状態で、さらに送信コマンドを入力したので FIFO キューが満杯な状態となっています。"a"コマンドにより、データ送信が停止され、FIFO キューも空になります。

・ステータスレジスタ表示コマンド(S)

```
-----
CAN0 status register

Transmit error counter (TEC)       : 0
Receice error counter (REC)        : 56

ESI status flag (ESIF)                    : 0 -> NOT received CANFD message with ESI flag
Communication status flag (COMSTS)   : 1 -> communication ready
Receive status flag (RECSTS)             : 0 -> idle
Transmit status flag (TRMSTS)            : 1 -> in transmission
Bus-off status flag (BOSTS)              : 1 -> bus-off
Error-passive status flag (BOSTS)     : 0 -> NOT error-passive
Channel sleep status flag (CSLPSTS) : 0 -> NOT channel sleep mode
Channel halt status flag (CHLTSTS)   : 0 -> NOT channel halt mode
Channel reset status flag (CRSTSTS) : 0 -> NOT channel reset mode

-----
CANFD0 status register

Successful occurrence counter (SOC) : 0
Error occurrence counter (EOC)          : 255

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation is present
PNF mode status flag (PNSTS)                                    : 0b00 -> Nomal (not use PNF)
Successful occurrence counter overflow flag (SOCO)       : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)             : 1 -> overflowed
Transmitter delay compensation result status (TDCR)     : 0x01
```

現状のステータスレジスタを表示します。

・エラーフラグ表示コマンド(E)

```
-----
CAN error flag register

GERFLL = 0x0000

CANFD payload overflow flag (CMPOF)                 : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow is detected
FIFO message lost flag (MES)                        : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                                : 0 -> NO DLC error is detected

-----
CAN0 error flag register

C0ERFLL = 0x011F

ACK delimiter error flag (ADERR)                    : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)    : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                               : 0 -> NO CRC error is detected
ACK error flag (AERR)                               : 0 -> NO ACK error is detected
Form error flag (FERR)                              : 0 -> NO form error is detected
Stuff error flag (SERR)                             : 1 -> stuff error is detected
Arbitration-lost flag (ALF)                         : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                                 : 0 -> NO bus-lock is detected
Overload flag (OVLF)                                : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                        : 1 -> bus-off recovery is detected
Bus-off entry flag (BOEF)                           : 1 -> bus-off entry is detected
Error-passive flag (EPF)                            : 1 -> error-passive is detected
Error-warning flag (EWF)                            : 1 -> error-warning is detected
Bus-error flag (BEF)                                : 1 -> bus-error detected

-----
CANFD0 error flag register

C0FDSTSL = 0x0100

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation
Successful occurrence counter overflow flag (SOCO)    : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)         : 1 -> overflowed
```

現状のエラーフラグを表示します。

・エラーフラグ履歴表示コマンド(H)

```
-----
CAN error flag register [error history]

CANFD payload overflow flag (CMPOF)                 : 0 -> NO CANFD payload overflow is detected
Transmit history buffer overflow flag (THLES) : 0 -> NO Transmit history buffer overflow is detected
FIFO message lost flag (MES)                        : 0 -> NO FIFO message lost is detected
DLC error flag (DEF)                                : 0 -> NO DLC error is detected

-----
CAN0 error flag register [error history]

ACK delimiter error flag (ADERR)                    : 0 -> NO ACK delimiter error is detected
Bit 0 error (dominant bit error) flag (B0ERR)   : 0 -> NO dominant bit error is detected
Bit 1 error (recessive bit error) flag (B1ERR) : 0 -> NO recessive bit error is detected
CRC error flag (CERR)                               : 0 -> NO CRC error is detected
ACK error flag (AERR)                               : 0 -> NO ACK error is detected
Form error flag (FERR)                              : 0 -> NO form error is detected
Stuff error flag (SERR)                             : 1 -> stuff error is detected
Arbitration-lost flag (ALF)                         : 0 -> NO arbitration-lost is detected
Bus-lock flag (BLF)                                 : 0 -> NO bus-lock is detected
Overload flag (OVLF)                                : 0 -> NO overload is detected
Bus-off recovery flag (BORF)                        : 1 -> bus-off recovery is detected
Bus-off entry flag (BOEF)                           : 1 -> bus-off entry is detected
Error-passive flag (EPF)                            : 1 -> error-passive is detected
Error-warning flag (EWF)                            : 1 -> error-warning is detected
Bus-error flag (BEF)                                : 1 -> bus-error detected

-----
CANFD0 error flag register [error history]

Transmitter delay compensation violation flag (TDCVF) : 0 -> NO violation
Successful occurrence counter overflow flag (SOCO)    : 0 -> NOT overflow
Error occurrence counter overflow flag (EOCO)         : 1 -> overflowed
```

過去に立ったエラーフラグを表示します。

※エラーフラグ履歴が更新されるタイミングは

・エラー割り込み

・エラー表示（"E"コマンド）

・受信 FIFO キュー溢れで受信割り込みが入った場合

です。エラーフラグが立って、エラー割り込みが掛らないケースでは履歴変数にはエラーの情報は反映されません。

（エラー割り込みが掛る条件に関しては、4.2 章参照）

・エラーフラグ・通信カウンタクリアコマンド(C)

CAN Error regiser / occurrence counter clear

エラーフラグと、通信カウンタをクリアします。

一度、エラー割り込みが入った場合、基本的にはエラーフラグはクリアされないので、再度別なエラーが発生した際はエラー割り込みが掛りません。

本コマンドで、エラーフラグはクリアされるので、本コマンド入力後はエラー割り込みが掛る様になります。(エラー発生の要因が取り除かれていない場合、本コマンド入力後直ぐにエラー割り込みが掛ります。)

6. サンプルプログラムの説明(SAMPLE1)

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。

本サンプルプログラムでは、割り込みは使用しません。

・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(*2)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(*3)はデータを受信する度に LED の点灯・消灯が切り替わる

－キーボードから入力したキーと送信データの関係－

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

(*1)can_operation.h の定義で、標準フォーマットのみ取り扱う様に変更可能
(*2)SW2 がデータ送信用スイッチ
(*3)LED2 が受信 LED

6.2. 受信ルールの設定

受信ルール番号	フォーマット	ID	受信バッファ番号(*1)
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

(*1)受信バッファ番号は関数の引数としては指定しませんが、この値が設定されます
（受信バッファ番号=受信ルール番号、として設定されます）

本サンプルプログラムでは、受信ルールで設定した、ルールにマッチした（ID 等が一致した）データの格納先は、受信バッファに割り当てています。受信バッファは、最大 16 個設定可能で、本サンプルプログラムでは 16 個の設定です。本来、受信ルール番号と受信バッファ番号は自由に紐付けできますが、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番号は 1:1 対応としています。

※RS-CANFD_Lite モジュールの機能としては、受信ルールにマッチしたデータの格納先を最大 2 個（受信 FIFO(0)と受信バッファ等）設定できますが、本サンプルプログラムでは 1 個に制限しています

※本サンプルプログラムでは、受信バッファ番号は 0-15 までの 16 個の受信バッファの使用が可能です
受信ルールも、0-15 までの 16 ルールの設定が可能です
※送信で FIFO（段数 4）を使用した際は、受信バッファは 12 個となりますので、受信ルール 11-15 の受信先を受信バッファに設定する事はできません

6.3. 送信バッファの設定

送信バッファは、ch 毎に 4 つあり送信バッファ 0~3 が使用可能です。

データ送信中は、同一の送信バッファに次のデータを格納する事はできません。

・送信バッファ

送信バッファ番号	キーボードからのコマンド
0	0
1	1
2	2
3	3

SAMPLE1 では、キーボードの"0"（1 バイト送信、ID=0）は、送信バッファ 0 で送信します。

キーボードからの送信の場合、送信間隔が十分空くので、連続で同じ送信バッファを使用しても問題ありませんが、プログラムで連続でデータを送信する場合、送信完了を待つか、別な番号の送信バッファを使用してください。

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。

(1)受信ルールの設定

初期化の処理内（通信開始前）に設定します。

```
//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);
```

(2)データの受信

```
for(i=0; i<=receive_buf_num; i++)  //receive_buf_num=16,  受信バッファ数
{
   //引数:       受信バッファ番号   CAN メッセージ構造体
   r_ret = can_rxbuf_receive((unsigned char)i, &r_rmsg);
```

受信バッファ番号： can0_receive_rule_set()呼び出し時に設定された番号
（ここでは全受信バッファをスキャンしています）

CAN データ構造体： r_msg(.id, .rtr, .ide, .dlc, .data[], .ts)

r_ret が-1 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。

(3)データの送信

//引数:　　バッファ番号　CAN データ構造体
s_ret = **can0_txbuf_send**(0, &s_msg);

バッファ番号:　送信に使用する送信バッファ番号
CAN データ構造体:　s_msg(.id, .rtr, .ide, .dlc, .data[])
データ:　送信データ

※バッファ番号は 0~3 の値を設定してください
（連続でデータを送信する場合は、別な番号を設定してください。データ送信完了後に本関数を呼び出す場合は、同じ番号でも問題ありません。）

6.5. データの受信

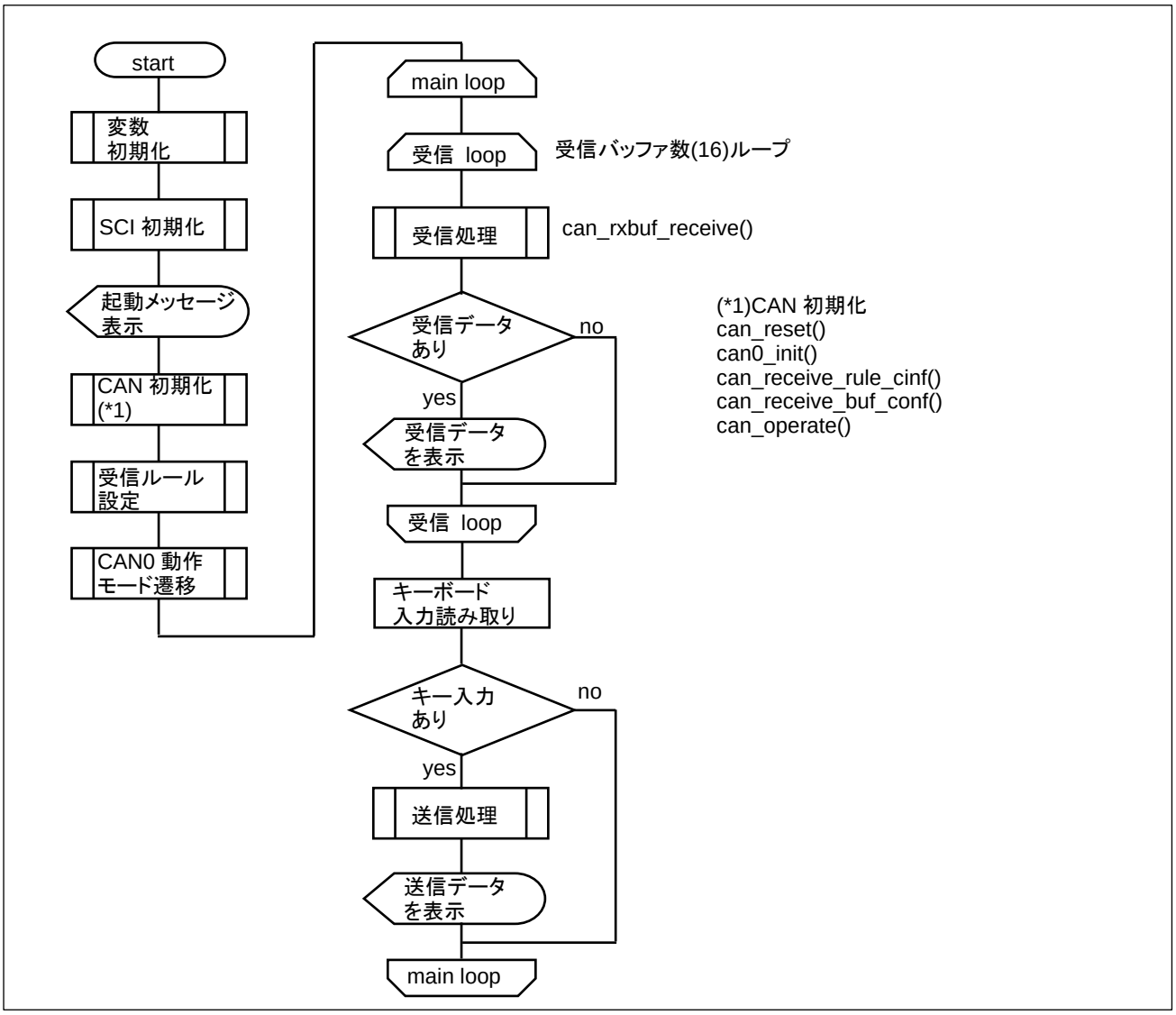
SAMPLE1 でのデータの受信に関しては、受信メッセージバッファまでのデータ格納に関しては、（初期化、受信ルール設定、受信バッファ設定が済んでいれば）マイコンのハードウェアが行います。受信バッファにデータが格納されているかは、プログラムで受信関数(can_rxbuf_receive)を呼び出す事で確認を行っています。そのため、常に受信関数を呼び出して確認を行わないと、データの取りこぼしが生じる可能性があります。受信データの確認に CPU リソースを食うため、スムーズな手法とはいえないと考えます。

（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。）

6.6. SAMPLE1 フローチャート

－処理フロー－

メイン関数　main_s1()



7. サンプルプログラムの説明(SAMPLE2)

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

を行うサンプルプログラムとします。

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

7.2. 送受信方式

・受信
受信 FIFO　※デフォルト
送受信 FIFO　選択可
受信バッファ　選択可

・送信
送受信 FIFO　※デフォルト
送信バッファ　選択可

・送受信 FIFO は、1 本しかないので、送受信 FIFO を受信で使用した場合、送信で送受信 FIFO は使えない

という条件があります。

送信側の割り込みは、「送信完了」の割り込みで、「データ送信完了後」に「送信したデータに対し、データを受信した側が ACK を返した」場合に入ります。

受信は 3 種類の受信方式のどの方式で受信したかで呼ばれる割り込みが異なります。送信は、送受信 FIFO を使った場合でも送信バッファを使った場合でも、同じ割り込みとなります。

どの受信方式を選択するかは、受信ルール設定 can0_receive_rule_set()で受信先を設定しますので、ルール毎に受信先を変えることは、RS-CANFD_Lite の機能としては可能ですが、本サンプルプログラムでは、can_operation.h でどの受信方式を使うか 1 つ選ぶ形となります。

どの送信方式を使用するかは、送信関数
送受信 FIFO を使った送信関数　can0_srfifo_send()
送信バッファを使った送信関数　can0_txbuf_send()

で使い分け可能ですが、本サンプルプログラムでは、can_operation.h でどの送信方式を使うか 1 つ選ぶ形としています。

受信方式、送信方式をデフォルトから変更する場合は、can_operation.h を編集してください。

※本サンプルプログラムでは、can_intr.h 内で、どの受信割り込みを有効化するか 1 つ選ぶ形としているので、複数の受信方法（＝受信割り込み）を有効にする場合は、can_intr.h を編集してください

7.3. FIFO の設定

FIFO は First In First Out の構成のバッファで、複数の CAN メッセージを格納できるバッファです。

受信 FIFO は、RXFIFO(0)~RXFIFO(1)の 2 本あります。

本サンプルプログラムでは、RXFIFO(0)を使う様にしています。RXFIFO(1)は未使用です。

送受信 FIFO は SRFIFO(0)の 1 本です。

送受信 FIFO での送信を有効にした場合（デフォルト）、使用可能な送信バッファは減少します。（送信バッファの 0 番を送受信 FIFO との紐付けとしており、使用不可となります。その場合でも、送信バッファの 1~3 は送信バッファとして、使用可能です。）

・使用 FIFO

FIFO	用途	FIFO 段数
RXFIFO(0)	受信	4
RXFIFO(1)	未使用	0
SRFIFO(0)	送信	4

FIFO の段数（1 つの FIFO にいくつのメッセージを格納可能か）は、0（FIFO を使用しない）から 16 段まで設定可能です。但し、全体でメッセージバッファは 16 個となりますので、上記 3 本の FIFO 段数と受信メッセージバッファ数を全て合算して 16 以下にする必要があります。

SAMPLE2 では、FIFO で 8 個のメッセージバッファを消費しているので、（受信バッファは動作に使用していませんが、）受信バッファ数は 8 個となります。FIFO と受信バッファの両方を用いる際は、FIFO 段数と受信バッファ数の合計に注意願います。※合計が 16 を超えると、CAN のデータ受信時に意図しないメモリ破壊を引き起こします。レジスタ値の上書きが生じて、CAN 全体の動作が異常になるので設定には注意が必要です。

7.4. 受信ルールの設定

SAMPLE2 では、FIFO を使って受信しますので、受信のルールの設定が変わります。

・受信ルール

受信ルール番号	フォーマット	ID	受信 FIFO 使用時 [デフォルト]	送受信 FIFO 使用時 [選択可]	受信バッファ 使用時 [選択可]
0	標準(SID)	0x000	RXFIFO(0)	SRFIFO(0)	0
1	標準(SID)	0x001	RXFIFO(0)	SRFIFO(0)	1
2	標準(SID)	0x002	RXFIFO(0)	SRFIFO(0)	2
3	標準(SID)	0x003	RXFIFO(0)	SRFIFO(0)	3
4	拡張(EID)	0x0000000	RXFIFO(0)	SRFIFO(0)	4
5	拡張(EID)	0x0000001	RXFIFO(0)	SRFIFO(0)	5
6	拡張(EID)	0x0000002	RXFIFO(0)	SRFIFO(0)	6
7	拡張(EID)	0x0000003	RXFIFO(0)	SRFIFO(0)	7

7.5. 送信バッファの設定

・送信バッファ

送信バッファ 番号	送受信 FIFO 使用時 [デフォルト]	送信バッファ 使用時(*1) [選択可]
0	SRFIFO(0)	コマンド"0"に対応
1	未使用	コマンド"1"に対応
2	未使用	コマンド"2"に対応
3	未使用	コマンド"3"に対応

送信に送受信 FIFO を使う場合、送信バッファとしては 0 番を使用する設定です。

※未使用となっているところは、送信バッファの設定はされていますので、送信バッファとして使用可能です

(*1)送信バッファで送信するように設定した場合、SAMPLE1 と同じですが、送信割り込み処理は追加されています

7.6. 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)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);

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

送受信 FIFO を使って受信する場合は、**CAN_RULE_RXFIFO0→CAN_RULE_SRFIFO0** に変わります。

(2)データの送信

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

送受信 FIFO を使用する関数で送信。
（送信バッファを使用して送信する様に設定した場合は、SAMPLE1 と変わりません）

(1)~(2)の処理はメイン関数内で処理されます。

(i1)受信割り込み関数[SAMPLE1 との相違点]

intrcangrfr_interrupt()（can_intr.c 内）

```
while(1)
{
   if ( (RFSTS0 | RFSTS1) & 0x0008) == 0) break;   //受信割り込みルーチンを抜ける

   //RXFIFO(0)の受信処理（CAN-ch0 側）
   RFSTS0 &= ~0x0008;   //b3:RFIF,割り込みフラグクリア
```

```
    while((RFSTS0 & 0x0001) == 0)    //FIFO 内に未読メッセージあり
    {
        ret = can_rxfifo_receive(0, &msg);       //受信 FIFO を使った RXFIFO(0)受信
        [受信データをリングバッファにコピー]
    }
}
```

受信 FIFO には複数のデータが格納されているかもしれないので、FIFO が空になるまでデータの読み出し(can_rxfifo_receive の実行)を行います。

受信 FIFO の受信割り込みは、RXFIFO(0)~RXFIFO(1)のいずれかにデータが格納された時に割り込み関数に飛んできますが、割り込み関数内の処理を実行中に、新しいデータを受信した場合注意が必要です。

※RXFIFO(0)と RXFIFO(1)は、同じ割り込みに飛んできます

受信割り込み関数内の処理を実行中に次のデータを受信した場合で、割り込み処理を抜ける条件に関して考えてみます。

シンプルな処理としては、
・RXFIFO(0)の読み出し
・RXFIFO(0)の割り込みフラグを落とす
・RXFIFO(1)の読み出し
・RXFIFO(1)の割り込みフラグを落とす
・割り込みルーチンを抜ける
というような、処理とすることが考えられますが、この様な処理とした場合問題が起こります。シンプルな処理を行った場合の動作を示します。

・ケース 1

時系列	割り込みフラグ			アクション	プログラム上の処理
	RXFIFO		INTRCANGRFR		
	(0)	(1)			
1	○	-	↑	RXFIFO(0)でデータ受信	→割り込み関数に飛んでくる
2	○	○		RXFIFO(1)でデータ受信	RXFIFO(0)の読み出し処理
3	-	○			RXFIFO(0)のフラグを落とす
4	-	○			RXFIFO(1)の読み出し処理
5	-	-			RXFIFO(1)のフラグを落とす
6	-	-			割り込み関数を抜ける
7	○	-	↑	RXFIFO(0)でデータ受信	→割り込み関数に飛んでくる
8	-	-			RXFIFO(0)の読み出し処理

上記のケースは問題ありません。

・ケース 2

時系列	割り込みフラグ			アクション	プログラム上の処理
	RXFIFO		INTRCANGRFR		
	(0)	(1)			
1	-	○	↑	RXFIFO(1)でデータ受信	→割り込み関数に飛んでくる
2	-	○			RXFIFO(0)読み出し処理→データなし
3	○	○		RXFIFO(0)でデータ受信	RXFIFO(1)読み出し処理
4	○	-			RXFIFO(1)のフラグを落とす
5	○	-			割り込み関数を抜ける(*1)
6	○	-			
7	○	-		RXFIFO(0)でデータ受信	(*2)
8	○	○		RXFIFO(1)でデータ受信	(*2)

上記のケースは 2 つ問題があります。
(*1)データが残っているのに割り込み関数を抜ける
→この場合、割り込み関数を抜けた後で直ぐに次の割り込みが発生すれば特に問題ではありませんが、割り込みは発生しません
(*2)以降データを受信した場合でも割り込みは発生しない

受信 FIFO 割り込み(INTRCANGRFR)の割り込みが成立する条件が、「RXFIFO(0)~(1)のいずれかの割り込みフラグが立っている」であれば問題ないのですが、「RXFIFO(0)~(1)の割り込みフラグが全て立っていない状態から、いずれかの割り込みフラグが立つ」となっています。そのため、RXFIFO(0)~(1)の割り込みフラグが残っている状態で割り込み関数を抜けた場合、次の割り込みが入ってくる事がない、という動作となります。

必ず、RXFIFO(0), RXFIFO(1)の両方のフラグが立っていないタイミングがある事を確認した後で、割り込みルーチンを抜ける必要があります。

現状の割り込みルーチンの処理は以下の流れとなります。

・ケース 3

時系列	割り込みフラグ			アクション	プログラム上の処理
	RXFIFO		INTRCANGRFR		
	(0)	(1)			
1	○	-	↑	RXFIFO(0)でデータ受信	→割り込み関数に飛んでくる
2	○	-			2 つのフラグを確認する処理
3	-	-			RXFIFO(0)のフラグを落とす
4	-	-			RXFIFO(0)読み出し処理
5	-	-			RXFIFO(1)の処理→データなし
6	-	-			2 つのフラグを確認する処理(*3)
7	-	○	↑	RXFIFO(1)でデータ受信	
8	-	○			(*3)で RXFIFO(0), RXFIFO(1)どちらのフラグも立っていないので、割り込み関数を抜ける
9	-	○			→割り込み関数に飛んでくる

上記の様なケースは問題ありません。8 番のところで一度割り込み関数を抜けますが、INTRCANGRFR の割り込みフラグは立っているので、割り込み関数を抜けた後、再度割り込み関数に飛んできます（2 回目の割り込み関数の処理で RXFIFO(1)のデータを処理できます）。
※ポイントは RXFIFO(0)~(1)の 2 つのフラグが両方落ちているタイミングがあるかどうかです

(i2)送信割り込み関数[SAMPLE1 との相違点]

※割り込み関数（「送受信 FIFO 送信」と「送信バッファ」で共通）

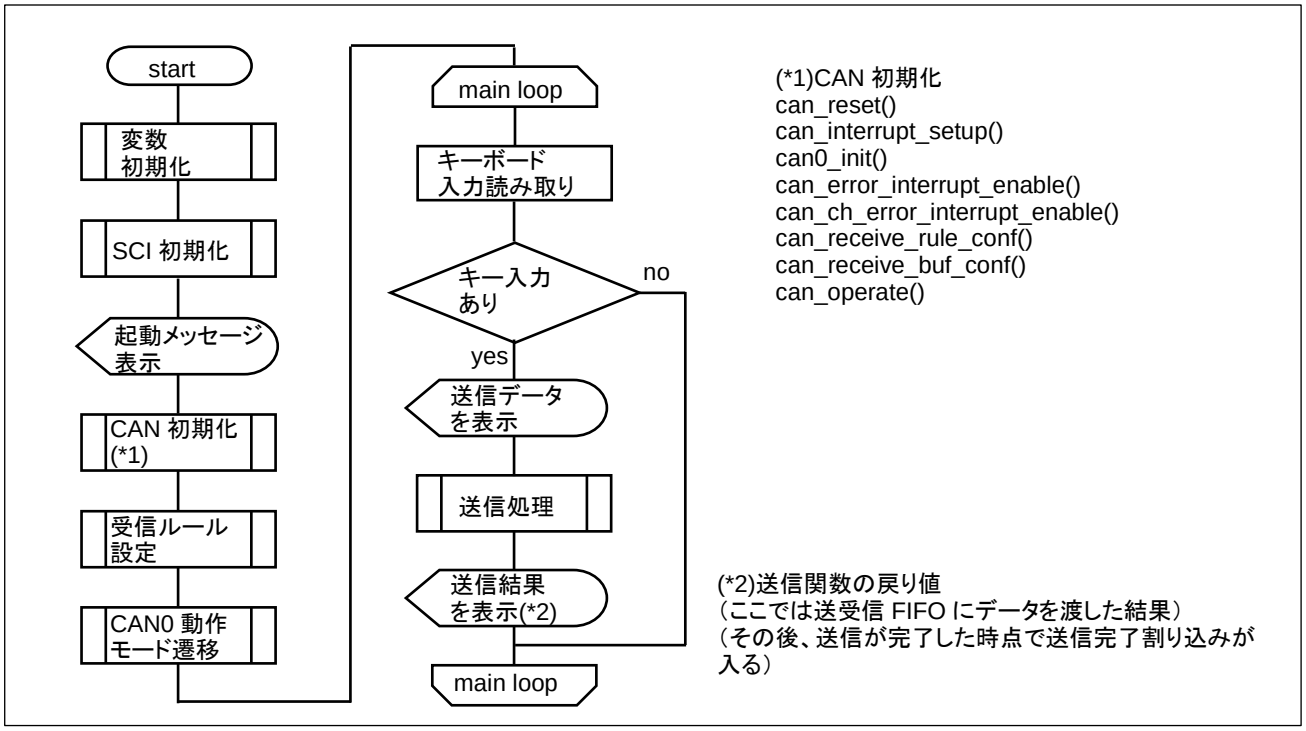
intrcan0trm_interrupt()

```
//送受信FIFO割り込み
if((CFSTS0 & 0x0010) != 0)     //b4 CFTXIF==1  送信割り込み要求あり
{
   [送受信 FIFO で送信済みとなった事を示すフラグをグローバル変数に保存]
   CFSTS0 &= ~0x0010;    //b4:CFTXIF,  割り込みフラグクリア
}
```

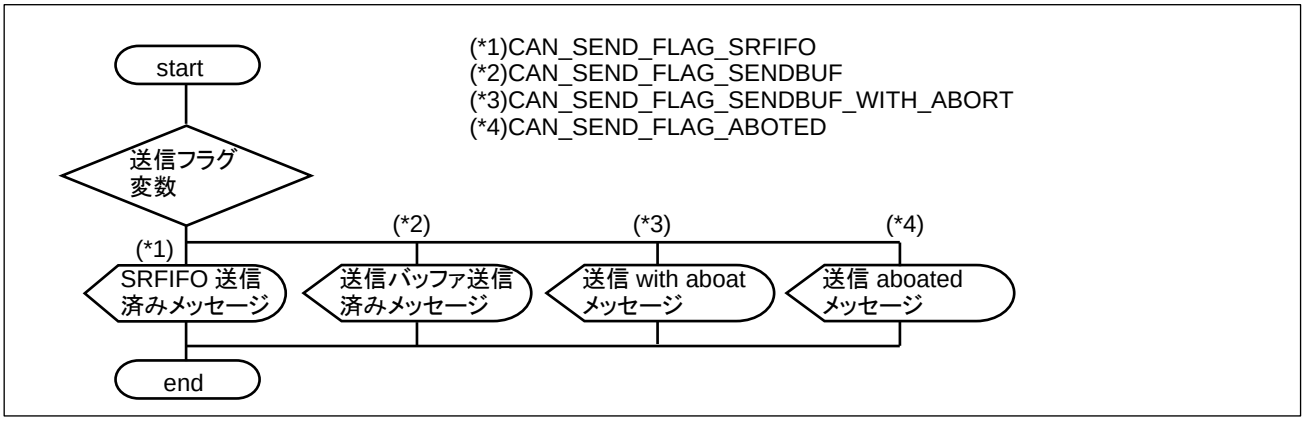
7.7. SAMPLE2 フローチャート

－処理フロー－

メイン関数　main_s2()

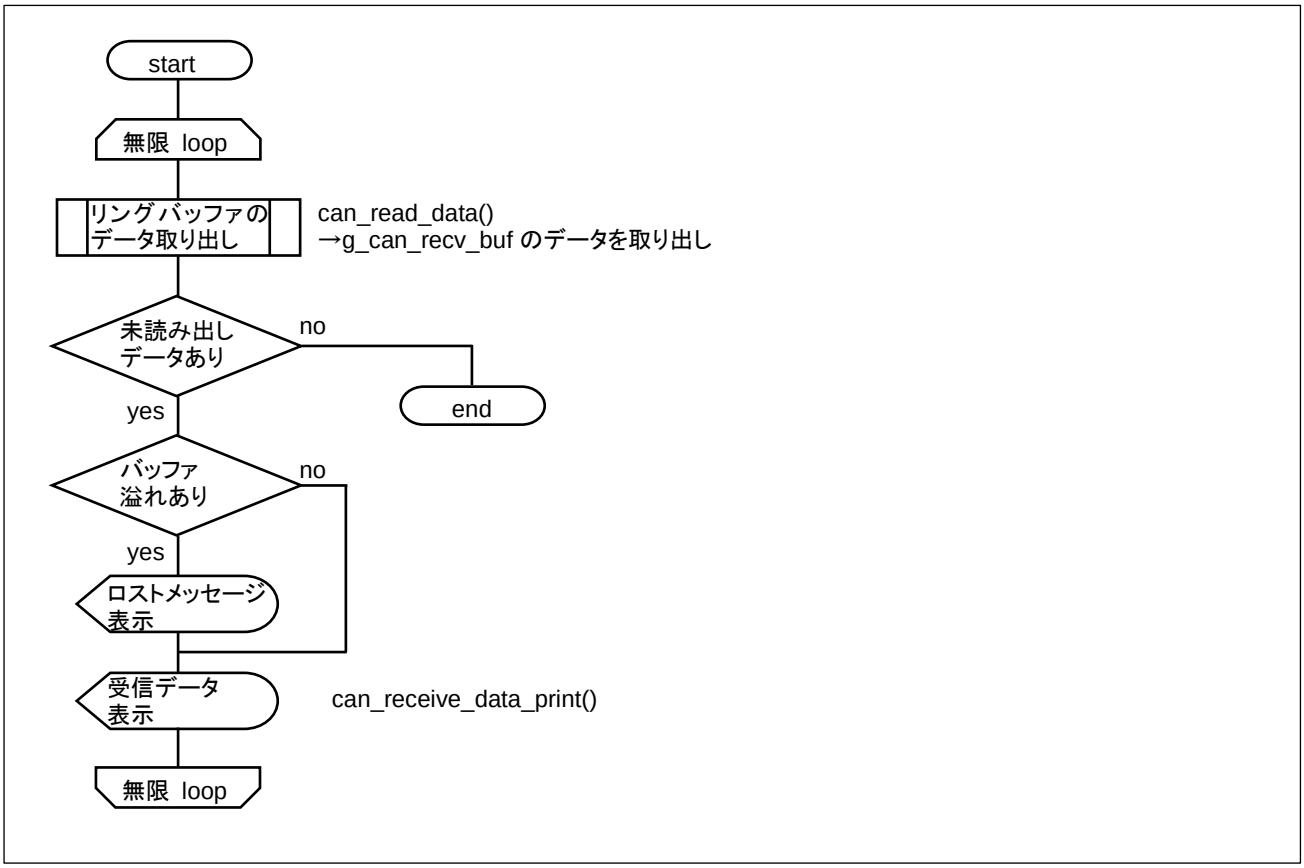


送信コールバック関数 can_interrupt_send_callback() （main_s2.c 内）



送信コールバック関数は、送信割り込みの後で呼ばれます。送信フラグは、送信割り込み関数内で設定され、本関数内では送信結果の表示を行います。

受信コールバック関数 can_interrupt_receive_rxfifo0_callback() （main_s2.c 内）



受信コールバック関数は、受信 FIFO 割り込みの後に呼ばれます。ここでは、リングバッファに溜まっている受信データの表示を行っています。
（受信 FIFO 割り込み関数内で、FIFO に溜まっているデータをリングバッファへコピーしています。）

8. サンプルプログラムの説明(SAMPLE3)

8.1. プログラム仕様

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

を行うサンプルプログラムとします。

SAMPLE2 との相違点は、リモートフレームに対応している事です。

－キーボードから入力したキーと送信データの関係－

・データフレーム送信
キーボード 0~3（SAMPLE1 と同一）

・リモートフレーム送信

キーボードからの入力	ID	送信要求バイト数(DLC)
q	0x0000000	1
w	0x0000001	2
e	0x0000002	4
r	0x0000003	8

・リモートフレーム応答
0xA1 A2 A3 A4 A5 A6 A7 A8
を、要求元の送信要求バイト数に応じて返答
（DLC=4 のときは、0xA1 A2 A3 A4 を返す）

※本サンプルプログラムは、ID=0x0000000 ~ 0x0000003 でリモートフレームを受信すると、応答（データフレームを送信）しますので、本サンプルプログラムが動作するボードを 3 台（以上）同一バスに接続すると、2 台（以上）が同時に応答します　CAN バス上では、1 つのリモートフレームに続き 2 つ（以上）のデータが流がれますので、多少動作が見難くなるかと思います

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

使用する割り込みと FIFO 設定は、SAMPLE2 と同じです。

8.2. 受信ルール設定

・受信ルール

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	受信 FIFO 使用時 [デフォルト]	送受信 FIFO 使用時 [選択可]	受信バッファ使用時(*1) [選択可]
0	標準(SID)	データフレーム	0x000	RXFIFO(0)	SRFIFO(0)	0
1	標準(SID)	データフレーム	0x001	RXFIFO(0)	SRFIFO(0)	1
2	標準(SID)	データフレーム	0x002	RXFIFO(0)	SRFIFO(0)	2
3	標準(SID)	データフレーム	0x003	RXFIFO(0)	SRFIFO(0)	3
4	標準(SID)	リモートフレーム	0x000	RXFIFO(0)	SRFIFO(0)	-(*1)
5	標準(SID)	リモートフレーム	0x001	RXFIFO(0)	SRFIFO(0)	-(*1)
6	標準(SID)	リモートフレーム	0x002	RXFIFO(0)	SRFIFO(0)	-(*1)
7	標準(SID)	リモートフレーム	0x003	RXFIFO(0)	SRFIFO(0)	-(*1)
8 (4)(*1)	拡張(EID)	データフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	4
9 (5)(*1)	拡張(EID)	データフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	5
10 (6)(*1)	拡張(EID)	データフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	6
11 (7)(*1)	拡張(EID)	データフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	7
12 (8)(*1)	拡張(EID)	リモートフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	8
13 (9)(*1)	拡張(EID)	リモートフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	9
14 (10)(*1)	拡張(EID)	リモートフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	10
15 (11)(*1)	拡張(EID)	リモートフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	11

(*1)受信バッファを使うように選択することが出来ますが、標準 ID でのリモートフレーム受信は行いません、受信ルール番号は 0-11 が使用されます。

※送信に送受信 FIFO(0)（4 段）を使用した場合、受信バッファに割り当てられるのは、12 個までとなりますので、受信バッファ 12~は使用できません

※送信に送信バッファを使用した場合は、16 個の受信バッファが使用可能です

can_operation.h で標準 ID のみ取り扱う様に設定した場合、受信ルール番号の 8~15 が未設定となります。
（その場合でも、受信ルール番号の途中欠番が生じないように、前半に標準 ID の設定、後半に拡張 ID の設定を行っています。）

8.3. 送信バッファの設定

・送信バッファ

送信バッファ番号	送受信 FIFO 使用時 [デフォルト]	送信バッファ使用時 [選択可]
0	SRFIFO(0)	コマンド"0", "q"に対応
1	未使用	コマンド"1", "w"に対応
2	未使用	コマンド"2", "e"に対応
3	未使用	コマンド"3", "r"に対応

8.4. 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)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);
…
```

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。

(2)データの送信

```
//引数:　　CAN メッセージ構造体
s_msg.rtr = CAN_REMOTE_FRAME;
s_ret = can0_srfifo_send(&s_msg);
```

SAMPLE2 では、データフレーム固定のところが、SAMPLE3 では、コマンド(0~3, q~r)に応じて、送信する値を変更しています。(0~3 の時は s_msg.rtr =0, q~r の時は s_msg.rtr =1)

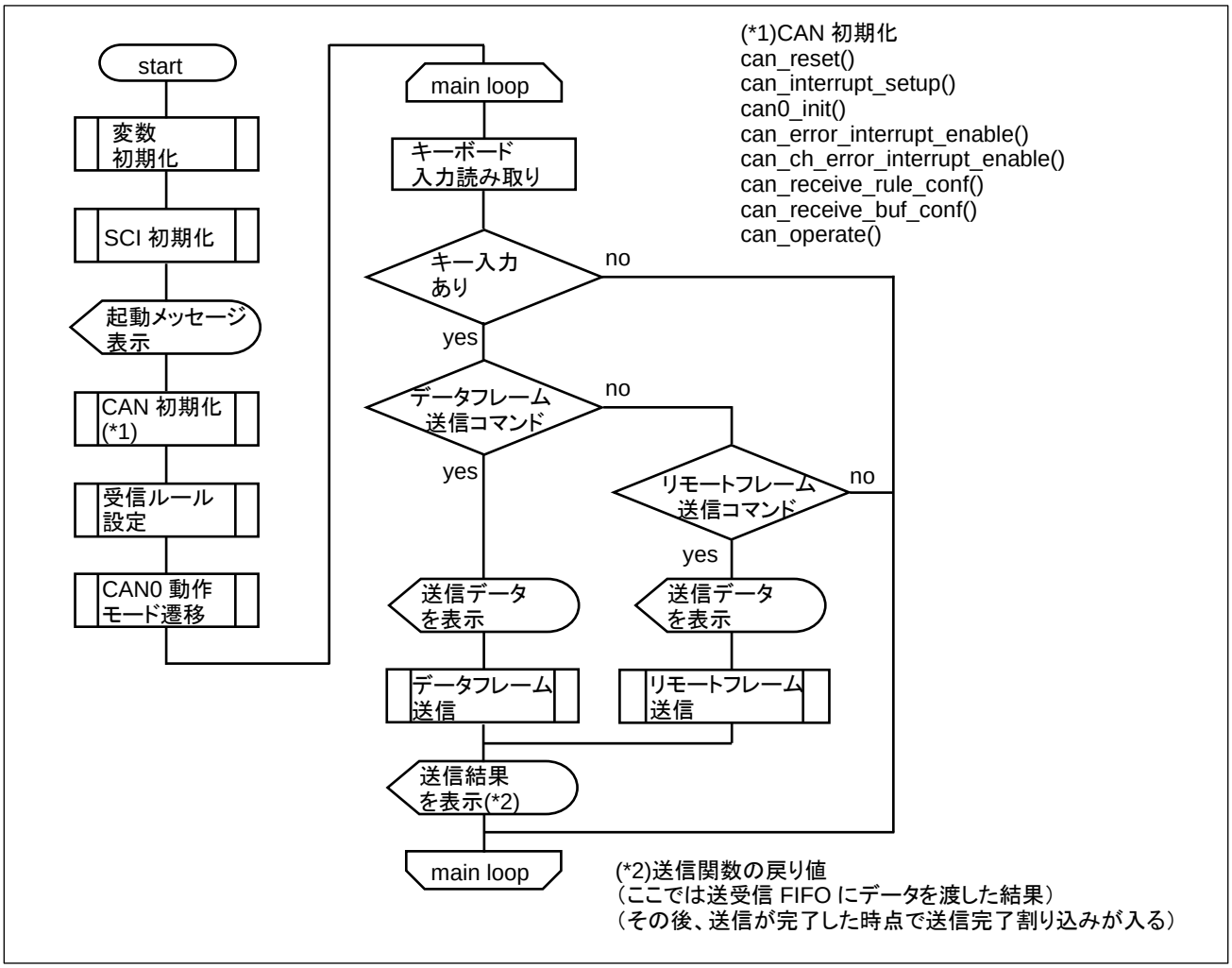
上記(1)(2)は、メイン関数内での処理となります。

割り込み関数は、SAMPLE2 と同じです。

8.5. SAMPLE3 フローチャート

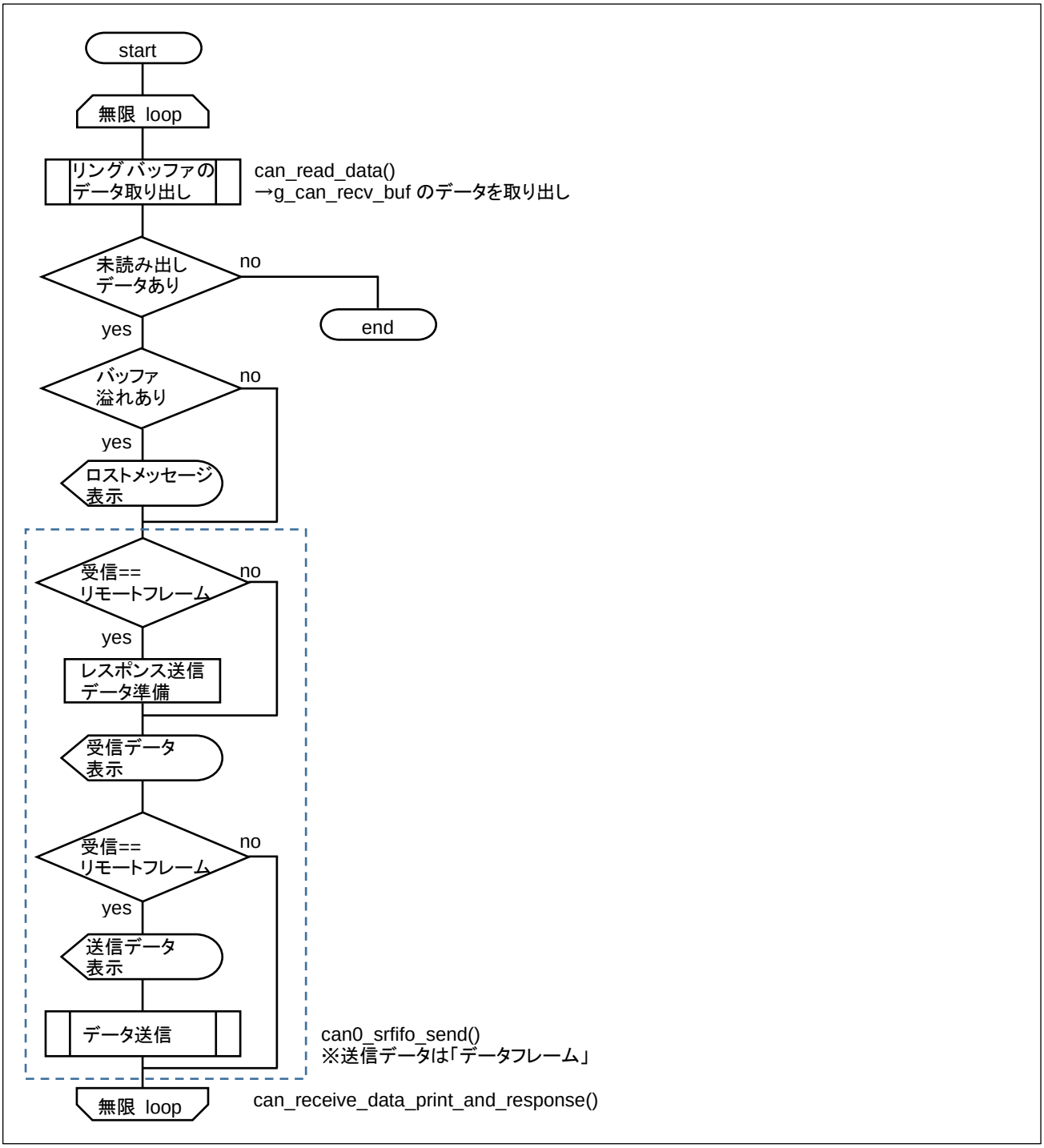
－処理フロー－

メイン関数　main_s3()



送信コールバック関数は、SAMPLE2 と同じ。

受信コールバック関数 can_interrupt_receive_rxfifo0_callback() （main_s3.c 内）



受信コールバック関数は、受信 FIFO 割り込みの後に呼ばれます。ここでは、リングバッファに溜まっているデータの表示とリモートフレームに対する応答を行っています。
（受信 FIFO 割り込み関数内で、FIFO に溜まっているデータをリングバッファへコピーしています。）

9. サンプルプログラムの説明(SAMPLE4)

9.1. 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 DLC (バイト数)
0	0x0000000	1 (1)
1	0x0000001	8 (8)
2	0x0000002	10 (16)
3	0x0000003	15 (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 DLC (バイト数)	送信要求バイト数(DLC)
q	0x0000000	1 (1)	1
w	0x0000001	8 (8)	8
e	0x0000002	10 (16)	16
r	0x0000003	15 (64)	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 のプログラムをボードに書き込んで起動すると、通信速度を聞かれますので、通信を行う 2 台のボードで同じ値を設定してください。

※通信速度設定が異なるボード同士は通信が通りません

```
HSBRL78F24-100 CAN Starter kit program boot.
Copyright (C) 2024 HokutoDenshi. All Rights Reserved.

SAMPLE4: CANFD [Data frame] send/receive program(with interrupt, use RXFIFO).
                [Remote frame] request/response program(with interrupt, use RXFIFO).

CAN ID mode -> EID

CANFD setting:

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

  CANFD speed [Mbps](2-5, default:2)) >
```

通信速度の入力

CANFD speed [Mbps](2-5, default:2)) >

のプロンプトに対し、2~5 の数値をキーボードから入力するか、[Enter]を押して先に進んでください。
（[Enter]を押した場合は、デフォルト値 2Mbps 設定となります。）

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

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

[Enter]

[Enter]

とりあえず動作を見る場合、[Enter]で先へ進めて問題ありません。

※設定項目に関しては

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

CAN トランシーバでの遅延補償を有効とするか(y/n)

```
  CAN transceiver secondary sampling point(d/o, d:delay+offset, o:offset, default:d) >
```

[遅延補償を有効にする場合](d)ディレイ＋オフセット値, (o)オフセット値のみ

```
  CAN transceiver secondary sampling point delay(0-255, default:0) >
```

[遅延補償を有効にする場合](0-255)オフセット値

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

RX のエッジフィルタを有効化するか(y/n)

上記の設定が行えます。

```
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

Command Usage:
  0123: Data frame send
  qwer: Remote frame send(data request)
  z: LED blink test(for board identify)
  s: send fornat EID <-> SID
  a: send abort
  S: Status resister print
  E: Error register print
  H: Error register history print
  C: Error register / occurrence counter clear
  (Push-SW: Data frame send [=keyboard 0])
```

CAN 通信速度の入力が終了すると、入力項目の一覧(Input summary:)が表示され、コマンド一覧(Command Usage:)が表示されます。

この状態は、通信が行える状態（受信待機）となります。

※起動後、CANFD のパラメータ設定の入力を完了していない状態では、データを受信できませんので注意願います

※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

9.2. 受信ルール設定

SAMPLE3 と同じです。

9.3. 送信バッファの設定

SAMPLE3 と同じです。

9.4. 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。SAMPLE3 と同一な点は説明を省略します。

(1)データの送信

送信関数に渡すメッセージ構造体が、CANFD メッセージ構造体となります。

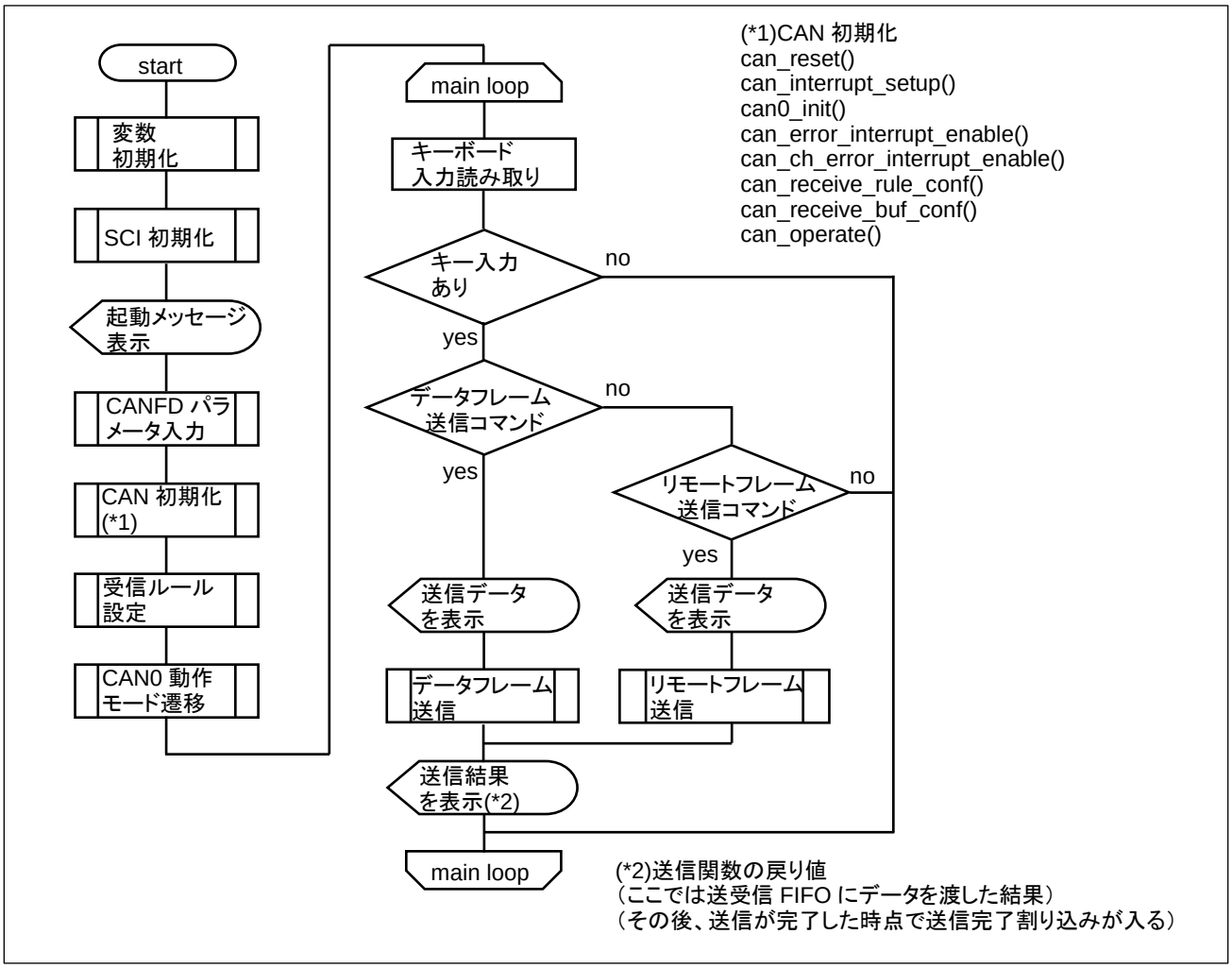
```
//引数:       CANFD メッセージ構造体
s_msg.dlc = 15;
s_msg.fdf = CANFD_DATA_FORMAT;
s_msg.brs = CANFD_BITRATE;
s_msg.esi = CANFD_ERROR_ACTIVE;
s_ret = can0_srfifo_send(&s_msg);
```

DLC 値が、最大 15（データバイト数 64）になります。（CAN フレームでは、最大 8）
FDF, BRS, ESI のパラメータが増えています。

9.5. SAMPLE4 フローチャート

－処理フロー－

メイン関数　main_s4()



送信コールバック関数、受信コールバック関数は、SAMPLE3 と同じ。（但し、受信コールバック関数は、CANFD フレームを扱う様に修正。）

10.サンプルプログラムで使用している関数・変数の説明

10.1.関数仕様(can.c)

can_interrupt_setup

概要:割り込み設定関数
宣言:
　　int can_interrupt_setup(void);

説明:
　・割り込み優先度の設定
　・割り込みフラグのクリア
　・割り込みマスク解除
を行います

引数:
　　なし
戻り値:
　　CAN_RET_SUCCESS(0): 正常終了

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
　・クロック設定
　・CAN リセットモード遷移
　・受信リングバッファ初期化
を行います

引数:
　　なし
戻り値:
　　CAN_RET_SUCCESS(0): 正常終了

can_receive_rule_conf

概要: 受信ルール数設定関数
宣言:
　　int receive_rule_conf(void);

説明:
　・受信ルール数の設定
を行います

グローバルリセットモード（can_reset()の後, can_operate()の前）時に実行してください。

引数:
　　なし
戻り値:
　　CAN_RET_SUCCESS(0): 正常終了
　　CAN_RET_MODE_ERROR(-5): 実行可能なモードではない

　受信ルール数として、16 ルールを設定します。
※RL78/F24 は、CAN が 1ch なので、独立した関数で設定する必要はありませんが、CAN を複数 ch サポートするマイコンでは、本関数で ch 毎のルール数の振り分けを行う事を想定しています。

can_receive_buf_conf

概要: 受信バッファ数設定関数
宣言:
　　int receive_buf_conf(unsigned short buf_num);

説明:
　・受信バッファ数設定

を行います

グローバルリセットモード（can_reset()の後, can_operate()の前）時に実行してください。

引数:
　　buf_num: 受信バッファ数（最大 16）、FIFO にメッセージバッファを割り振った場合は、設定可能数が 16 から減少します

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): 実行可能なモードではない

補足:

FIFO にメッセージバッファを割り振って、受信バッファに割り当て可能なバッファ数を超えた場合でも、本コマンドではエラーとはなりません。

can_operate

概要:動作モード変更関数

宣言:

int can_operate(void);

説明:

・CAN モジュールを動作モードに移行する処理

を行います

引数: なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

dlc2byte

概要:DLC 値バイト数変換得関数

宣言:

unsigned short dlc2byte(unsigned char dlc);

説明:

・DLC 値をデータバイト数に変換

を行います

引数:

dlc: DLC 値

戻り値:

データバイト数, DLC=1~8 戻り値 1~8, DLC=9,10,11,12,13,14,15 戻り値=12,16,20,24,32,48,64

can_timinig_parameter

概要:タイミングパラメータ計算関数

宣言:

int can_timing_parameter(float tq_ns, unsigned short can_speed, float sampling_point, unsigned char *nsjw, unsigned char *ntseg1, unsigned char *ntseg2);

説明:

・CAN のタイミングパラメータ(NTSEG1, NTSEG2, NSJW 値)の計算

を行います

引数:

tq_ns: 1Tq の値[ns]

can_speed: 通信速度の値[kbps]

sampling_point: サンプリングポイント(0.75 等、先頭から 75%の位置をサンプリングポイントに設定)

*nsjw: NSJW レジスタ値(計算値)

*ntseg1: NTSEG1 レジスタ値(計算値)

*ntseg2: NTSEG2 レジスタ値(計算値)

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

CAN_RET_VALUE_ERROR(-6): 計算結果エラー

補足:

can_timing_parameter(25.0f, 500, 0.75f, &nsjw, &ntseg1, &ntseg2);

1Tq=25ns(fCAN=40MHz, 分周比=1, CANFD 時のデフォルト)、500kbps, サンプリングポイント、75%の場合。nsjw, ntseg1, ntseg2 に格納された値を、レジスタに反映させてください。SJW 値は、TSEG2 の 1/4 程度の値に設定します。

canfd_timinig_parameter

概要:タイミングパラメータ計算関数

宣言:

int can_timing_parameter(unsigned short can_speed, unsigned char *dsjw, unsigned char *dtseg1, unsigned char *dtseg2);

説明:

・CANFD のタイミングパラメータ(DTSEG1, DTSEG2, DSJW 値)の計算

を行います

引数:

can_speed: 通信速度の値[kbps], 2000, 3300, 4000, 5000 のいずれか

*dsjw: DSJW レジスタ値（計算値）

*dtseg1: DTSEG1 レジスタ値（計算値）

*dtseg2: DTSEG2 レジスタ値（計算値）

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_ARGUMENT_ERROR(-2): 引数エラー

補足:

1Tq=25ns（fCAN=40MHz, 分周比=1, CANFD 時のデフォルト）の場合のレジスタ値を返します。

テーブル参照なので、速度は can.h で定義された値のみ受け付けます。

can_receive_rule_set

概要:受信ルール設定関数

宣言:

int can_receive_rule_set(unsigned char num, unsigned char mode, unsigned char receive_buf_num、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_RXBUF(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(0x0100) 送受信 FIFO(0)で受信

receive_buf_num: 受信先の受信バッファ番号（受信バッファで受信する場合のみ使用）

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): 引数チェックエラー

- CAN_RET_MODE_ERROR(-5): 受信ルール設定不可の動作モードとなっている

補足:

can_ch0.c 内に、can0_receive_rule_set()という関数がありますが、RL78/F24 は CAN-1ch のみなので、どちらの関数を使用しても同等です

本来、can0_receive_rule_set→CAN-ch0 向けの受信ルール関数

can_receive_rule_set→can*n*_receive_rule_set から呼ばれる関数(非ユーザ関数)

という想定です ※基本的には、can0_receive_rule_set()を使用する想定

※本関数を使用した場合は、ルール番号=受信バッファ番号の制約はありません

can_rxbuf_receive
canfd_rxbuf_receive

概要:データ受信関数(受信バッファ使用)

宣言:

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

int canfd_rxbuf_receive(unsigned char num, canfd_message *msg);

説明:

・受信バッファに格納されているデータの受信

を行います

引数:

num: 受信バッファ番号(0~15)

*msg: CAN メッセージ構造体/CANFD メッセージ構造体

戻り値:

1~15: 受信した DLC 値 ※can_rxbuf_receive では 1-8

CAN_RET_NODATA(-1): 受信データなし

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

can_rxfifo_receive
canfd_rxfifo_receive

概要:データ受信関数(受信 FIFO 使用)

宣言:

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

int can_rxfifo_receive(unsigned char fifo_no, canfd_message *msg);

説明:

・受信 FIFO に格納されているデータの受信

を行います

引数:

fifo_no: 受信 FIFO 番号(0~1)

*msg:CAN メッセージ構造体/CANFD メッセージ構造体

戻り値:

1~15: 受信した DLC 値 ※can_rxfifo_receive では 1-8

CAN_RET_NODATA(-1): 受信データなし

CAN_RET_ARGUMENT_ERROR(-2): 引数チェックエラー

CAN_RETFLAG_LOST_DATA(0x8x): b7=1 のとき、受信 FIFO のメッセージロストフラグが立っている（受信関数呼び出し前に破棄されたデータあり）

補足:

戻り値が 0x82 の場合、受信データは 2 バイトで、受信 FIFO フル時に受信したために受信 FIFO に格納されなかったデータがあることを示します。

can_read_data

概要:受信リングバッファ読み出し関数

宣言:

int can_read_data(unsigned char ch, can_message *msg);

int can_read_data(unsigned char ch, canfd_message *msg); ※#define CANFD_MODE 定義時

説明:

・受信リングバッファからのデータ読み出し

を行います

引数:

ch: CAN-ch（RL78/F24 の場合は必ず 0）

*msg: CAN メッセージ構造体/CANFD メッセージ構造体

戻り値:

CAN_RET_SUCCESS(0) : 正常終了

CAN_RET_NODATA(-1) : 受信データなし

CAN_RETFLAG_LOST_DATA(0x80) : リングバッファ溢れのため捨てられたデータがある事を示す（本関数呼び出し前に破棄されたデータあり）

補足:

戻り値が 0x80 の場合、リングバッファが溢れて捨てられた古いデータが存在します。リングバッファが溢れた場合には古いデータから捨てられる動作となります。

can_read_data_size

概要:受信リングバッファ格納データ数取得関数
宣言:
　　int can_read_data_size(unsigned char ch);

説明:
　・受信リングバッファに格納されているデータ数の取得
を行います
引数:
　　ch: CAN-ch（RL78/F24 の場合は必ず 0）
戻り値:
　　格納されているメッセージの数

can_read_buf_clear

概要:受信リングバッファ格納データクリア関数
宣言:
　　void can_read_buf_clear(unsigned char ch);

説明:
　・受信リングバッファに格納されているデータのクリア
を行います
引数:
　　ch: CAN-ch（RL78/F24 の場合は必ず 0）
戻り値:
　　なし
　　CAN_RETFLAG_LOST_DATA(0x80) : リングバッファ溢れのため捨てられたデータがある事を示す
　　（本関数呼び出し前に破棄されたデータあり）
補足:
　　戻り値が 0x80 の場合、リングバッファが溢れて捨てられた古いデータが存在します。リングバッファが溢れた場合には古いデータから捨てられる動作となります。

10.2.関数仕様(can0.c)

CAN の ch0 に依存する関数は、本ソース内に含まれます。

can0_init

概要:CAN-ch0 初期化関数
宣言:
　　int can0_init(void);

説明:
　・ポートの設定
　・チャネルスリープ解除
　・チャネルリセットモード遷移
　・速度や通信パラメータの設定
を行います
引数:　なし
戻り値:
　　CAN_RET_SUCCESS(0): 正常終了
　　CAN_RET_VALUE_ERROR(-6): 端子設定エラー（PIOR と使用端子の設定）
補足:
　　can_reset()の実行後、can0_init()を実行してください。

can0_operate

概要:動作モード変更関数
宣言:
　　int can0_operate(void);

説明:
　・CAN-ch0 を動作モードに移行する処理
　・送信バッファの設定
　・FIFO の有効化
を行います
引数:　なし
戻り値:
　　CAN_RET_SUCCESS(0): 正常終了

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 char mode, unsigned char ide, unsigned char rtr, unsigned long id);

説明:

・CAN-ch0 の受信ルール設定

を行います

引数:

num: 受信ルール番号

mode: 受信先

ide: 標準／拡張フォーマット区分

rtr: データフレーム／リモートフレーム区分

id: ID を指定

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_MODE_ERROR(-5): 受信ルール設定不可の動作モードとなっている

補足:

本関数では、受信バッファを使用して受信する場合、ルール番号=受信バッファ番号として設定します。

（can_receive_rule_set()関数では、そのような制約はありません）

can0_txbuf_send
canfd0_txbuf_send

概要:データ送信関数（送信バッファ使用）

宣言:

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

int canfd0_txbuf_send(unsigned char num, canfd_message *msg);

説明:

・データ送信

を行います

引数:

num: 送信バッファ番号(0-3)

*msg: CAN/CANFD 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_IN_USE(-3): 送信バッファ使用中

can0_srfifo_send
canfd0_srfifo_send

概要:データ送信関数(送受信 FIFO 使用)

宣言:

int can0_srfifo_send(can_message *msg);

int canfd0_srfifo_send(canfd_message *msg);

説明:

・データ送信

を行います

引数:

*msg: CAN/CANFD 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

can0_srfifo_receive
canfd0_srfifo_receive

概要:データ受信関数(送受信 FIFO 使用)

宣言:

int can0_txbuf_send(can_message *msg);

int canfd0_txbuf_send(canfd_message *msg);

説明:

・送受信 FIFO に格納されているデータの受信

を行います

引数:

*msg: CAN/CANFD メッセージ構造体

戻り値:

1-15 : 受信したDLC値 ※can0_srfifo_receiveでは1-8

CAN_RET_NODATA(-1): 受信FIFOにデータなし

CAN_RETFLAG_LOST_DATA(0x8x) : b7=1のとき、受信FIFOのメッセージロストフラグが立っている（受信関数呼び出し前に破棄されたデータあり）

補足:

戻り値が 0x82 の場合、受信データは 2 バイトで、送受信 FIFO フル時に受信したために受信 FIFO に格納されなかったデータがあることを示します。

10.3.関数仕様(can_intr.c)

割り込み関数は、本ソース内に含まれます。

割り込み処理は、4 章で説明していますので、そちらを参照下さい。

10.4.関数仕様(can_error_handle.c)

エラー表示等の関数は、本ソース内に含まれます。

can_ch_status_register_print

概要:ステータスレジスタ表示関数

宣言:

void can_ch_status_register_print(unsigned char ch);

説明:

・ステータスレジスタの表示

を行います

引数:

ch: CAN-ch（RL78/F24 の場合は必ず 0）

戻り値: なし

can_ch_error_flag_register_print

概要:エラーフラグレジスタ表示関数

宣言:

void can_ch_error_flag_register_print(unsigned char ch);

説明:

・エラーフラグレジスタ(ch 依存)の表示

を行います

引数:

ch: CAN-ch(RL78/F24 の場合は必ず 0)

戻り値: なし

can_ch_error_flag_register_history_print

概要:エラーフラグレジスタ履歴表示関数

宣言:

void can_ch_error_flag_register_history_print(unsigned char ch);

説明:

・エラーフラグレジスタ(ch 依存)の履歴表示

を行います

引数:

ch: CAN-ch(RL78/F24 の場合は必ず 0)

戻り値: なし

補足:

起動後にフラグが立ったエラーレジスタを保存しておき、本コマンドで表示させています

エラーレジスタの保存は、

・エラー割り込み

・can_ch_error_flag_register_print()関数実行時

・FIFO 受信割り込み関数(送受信 FIFO)内で FIFO フルを検出した場合

に行われます。(エラー割り込みを伴わないで、エラーフラグが立った場合は、履歴に反映されません。)

can_ch_error_flag_register_clear

概要:エラーフラグレジスタクリア関数

宣言:

void can_ch_error_flag_register_clear(unsigned char ch);

説明:

・エラーフラグレジスタ(ch 依存)のクリア

を行います

引数:

ch: CAN-ch(RL78/F24 の場合は必ず 0)

戻り値: なし

補足:

エラーフラグクリア後に発生したエラーは、エラー割り込みが掛ります。

エラー発生要因が取り除かれていない場合、フラグクリア後、直ぐにエラーフラグが立ち、エラー割り込みが掛ります。

can_error_flag_register_print

概要:エラーフラグレジスタ表示関数

宣言:

void can_error_flag_register_print(void);

説明:

・エラーフラグレジスタの表示

を行います

引数: なし

戻り値: なし

can_error_flag_register_history_print

概要:エラーフラグレジスタ履歴表示関数

宣言:

void can_error_flag_register_history_print(void);

説明:

・エラーフラグレジスタの履歴表示

を行います

引数: なし

戻り値: なし

補足:

起動後にフラグが立ったエラーレジスタを保存しておき、本コマンドで表示させています

エラーレジスタの保存は、

・エラー割り込み

・can_error_flag_register_print()関数実行時

・FIFO 受信割り込み関数（受信 FIFO）内で FIFO フルを検出した場合

に行われます。（エラー割り込みを伴わないで、エラーフラグが立った場合は、履歴に反映されません。）

can_error_flag_register_clear

概要:エラーフラグレジスタクリア関数

宣言:

void can_error_flag_register_clear(void);

説明:

・エラーフラグレジスタのクリア

を行います

引数: なし

戻り値: なし

補足:

エラーフラグクリア後に発生したエラーは、エラー割り込みが掛ります。

can_ch_error_interrupt_enable

概要:エラー割り込み有効化関数

宣言:

int can_ch_error_interrupt_enable(unsigned char ch);

説明:

・エラー割り込み（ch 依存）の有効化

を行います

引数:

ch: CAN-ch（RL78/F24 の場合は必ず 0）

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): 設定可能なモードではない

can_error_interrupt_enable

概要:エラー割り込み有効化関数

宣言:

int can_error_interrupt_enable(void);

説明:

・エラー割り込みの有効化

を行います

引数: なし

戻り値:

CAN_RET_SUCCESS(0): 正常終了

CAN_RET_MODE_ERROR(-5): 設定可能なモードではない

10.5.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_message g_can_recv_buf[CAN_CH][CAN_RECV_BUF_SIZE] ※CANFD_MODE 定義時

SAMPLE2~SAMPLE4 で使用。CAN メッセージを保存するリングバッファ。SAMPLE2~SAMPLE4 では、受信割り込み関数内で、受信データを本変数に保存。(ユーザ関数内では、本変数を読み出す形となります。)

CAN_CH = 1(RL78/F24 では、CAN は 1ch のみ)
CAN_RECV_BUF_SIZE = 16(デフォルト設定値)

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

リングバッファの書き込み、読み出しインデックス変数。g_can_recv_buf_index1 が書き込み位置、g_can_recv_buf_index2 が読み出し位置を示す。

※リングバッファのサイズは、デフォルトで 16 ですが、保存できるデータ数は 15 個までとなります。リングバッファをフルに使用したい場合は、本インデックス変数の運用を見直してみてください。

unsigned short g_can_recv_buf_override[CAN_CH]

リングバッファのデータ溢れが起こった事を保存する変数。

※FIFO のデータ溢れとリングバッファのデータ溢れは、別物です。本変数は、リングバッファのデータ溢れの状態を保存します。（データ溢れの場合 FIFO では、新しいメッセージを受信しない。リングバッファでは、古いメッセージを捨てるという動作となります。）

canfd_param g_canfd_setting

CANFD のパラメータを保存、設定時に使用する変数です。

g_canfd_setting.speed CANFD の速度
g_canfd_setting.tdce トランシーバ遅延補償
g_canfd_setting.tdcoc トランシーバ遅延補償の手法
g_canfd_setting.sdco トランシーバ遅延補償のオフセット値
g_canfd_setting.refe 受信フィルタ使用可否
のメンバを持っています。

volatile int g_can_send_flag[CAN_CH]

送信フラグを保存する変数。送信割り込み関数内でセットされます。ユーザプログラム内で、本変数を参照する事により、送信完了時のフラグを確認可能です。

unsigned short g_can_ch_error_flag[CAN_CH]
unsigned short g_canfd_ch_error_flag[CAN_CH]
unsigned short g_can_error_flag
unsigned short g_can_ch_error_flag_history[CAN_CH]
unsigned short g_canfd_ch_error_flag_history[CAN_CH]
unsigned short g_can_error_flag_history;

エラーフラグレジスタを保存する変数です。

unsigned short g_can_remote_frame_request[CAN_CH]

リモートフレーム要求の際にセットされる変数。画面表示

unsigned long g_packet_flag

一連のパケットであることを示すフラグ変数。(*2) [RL78/F14, F13: 未定義]

CAN0_CAN1_CANBUS_SHARE 定数が定義されている場合（CAN0 と CAN1 を同一の CAN バスに接続）、(*2) の変数が使用され、一連の動作の先頭に
++

一連の動作の終わりに、

--

が表示されます。(リモートフレームの受信からデータフレーム送信完了までの一連の動作)
画面表示を行うためだけの変数で、画面表示の見た目にしか寄与しません。

10.6.受信ルール設定に関して

```
                                                (1)                 (2)                 (3)
can0_receive_rule_set(0, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
can0_receive_rule_set(1, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000001);
can0_receive_rule_set(2, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000002);
can0_receive_rule_set(3, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000003);
can0_receive_rule_set(4, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_REMOTE_FRAME, 0x00000000);
can0_receive_rule_set(5, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_REMOTE_FRAME, 0x00000001);
can0_receive_rule_set(6, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_REMOTE_FRAME, 0x00000002);
can0_receive_rule_set(7, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, CAN_REMOTE_FRAME, 0x00000003);
can0_receive_rule_set(8, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
can0_receive_rule_set(9, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000001);
can0_receive_rule_set(10, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000002);
can0_receive_rule_set(11, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000003);
can0_receive_rule_set(12, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_REMOTE_FRAME, 0x00000000);
can0_receive_rule_set(13, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_REMOTE_FRAME, 0x00000001);
can0_receive_rule_set(14, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_REMOTE_FRAME, 0x00000002);
can0_receive_rule_set(15, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_EID, CAN_REMOTE_FRAME, 0x00000003);
```

本サンプルプログラムでは、can0_receive_rule_set() 関数で

(1)ID 区分　CANID_FORMAT_SID(0)か CANID_FORMAT_EID(1)
(2)データ／リモートフレーム区分　CAN_DATA_FRAME(0)か CAN_REMOTE_FRAME(1)
(3)ID 値

を設定し、(1)~(3)の全てが合致した場合のみ、受信するようになっています。受信ルールは 16 個という上限がありますので、「どのような ID 値であっても受信したい」とか、1 つのルールで SID(標準 ID)と EID(拡張 ID)の両方を受信したいという事があるかと思います。

その様な場合は、can_receive_rule_set() 関数に手を加えてください。この関数内のマスクレジスタが(1)~(3)を比較対象とするかどうかを決めています。can_receive_rule_set() 関数ではマスクビットを 0b1 に設定しているので、一致した場合のみルールにマッチするようになっています。

can.c, can_receive_rule_set()関数内

```
//GAFLMiL
gafl_reg[2] = 0xffff;//b15-0 GAFLIDM[15:0], 対応するIDビットを比較対象とする

//GAFLMiH
gafl_reg[3] = 0xdfff;//b15 GAFLIDEM, b14 GAFLRTRM, b12-0 GAFLIDM[28:16], 対応
する IDE, RTR, ID ビットを比較対象とする, b13:GAFLIFL1=0 受信ラベルビット 1 は設定しない
```

（サンプルプログラムでは ID 決め打ちで受信を行っていますが、受信ルールの設定次第で変えることが可能です。）

また、本サンプルプログラムでは DLC 値に関しては受信制限を設けていない（DLC がどの値であっても受信する）としていますが、受信する DLC 値に関して制約を掛ける事（DLC=8 以上のメッセージしか受信しない）も可能です。

受信ルールにマッチしたデータがデータ格納先（第 2 引数、上記では RXFIFO の 0 番）に格納され、複数のルールがマッチする場合は、番号の若い方のルールが優先されます。

ルール番号 7 番を欠番とした場合、ルール番号 8 以降には（条件が整っていても）マッチしません。ルール番号は若い順に欠番が出ない様に設定してください。

10.7.割り込みのポートデバッグに関して

受信、送信完了の割り込みがどのタイミングで生じているかをモニタするのが、can_intr.h 内の定数定義

#define CAN_INTERRUPT_DEBUG

です。定義している場合、割り込みのタイミングで特定の I/O ポートを反転させます。

・割り込みのモニタ機能で使用する I/O ポート

マイコンボード	受信バッファ受信 (INTRCANGRVC)	送受信 FIFO 受信 (INTRCAN0CFR)	チャネル送信 (INTRCAN0TRM)	受信 FIFO 受信 (INTRCANGRFR)	エラー (INTRCAN0ERR) (INTRCANGERR)
HSBRL78F24-100	P90(J2-1)	P91(J2-2)	P92(J2-3)	P93(J2-4)	P94(J2-5)

割り込みが入ってきた際、上記ポートが反転（L→H もしくは H→L）となりますので、オシロスコープ等で観測可能です。

11.CAN パケットの観測に関して

CD 内に含まれる、

RL78_F24_100_CAN_MONITOR

プロジェクトの mot ファイルを、マイコンボードに書き込んで動作させてみてください。

```
HSBRL78F24-100 CAN Starter kit program boot.
Copyright (C) 2024 HokutoDenshi. All Rights Reserved.

Monitor: CANFD [Data frame] send/receive program(with interrupt, use RXFIFO).
                [Remote frame] request program(with interrupt, use RXFIFO).

CAN ID mode -> EID

CAN speed setting:

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

   CAN speed (1-4, 1:1Mbps, 2:500kbps, 3:250kbps, 4:125kbps, default:1)) >1

Input summary:

   CAN speed [kbps] > 1000

CANFD setting:

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

   CANFD speed [Mbps](2-5, 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
```

CAN のビットレート
1Mbps, 500kbps, 250kbps, 125kbps 以外の速度の場合は、
ソースコードを変更してビルドし直す必要があります

CANFD のビットレートや通信条件
SAMPLE4 と同じ選択項目です

```
Command Usage:
   d: Data frame send(CAN, DLC=8)
   D: Data frame send(CANFD, DLC=15)
   r: Remote frame send(data request, DLC=8)
   s: send fornat EID <-> SID
   a: send abort
   S: Status resister print
   E: Error register print
   H: Error register history print
   C: Error register / occurrence counter clear
```

起動後は、CAN と CANFD のビットレートを聞かれますので、CAN バスの通信速度に合わせて設定してください。CANFD を取り扱わない場合は、CANFD 側のビットレートは適当で構いません。

CAN のパケットを受信すると、端末に表示されます。CAN/CANFD、ID, RTR, IDE, DLC 等は任意のデータ（どの ID のデータでも）受信して、表示します。

```
CAN0 data received, id_type=EID id=0x00000000 rtr=0x00 fdf=0x01 brs=0x01 esi=0x00 dlc=1 data=0x00 ts=0x1569
CAN0 data received, id_type=EID id=0x00000003 rtr=0x01 fdf=0x00 brs=0x00 esi=0x00 dlc=15 ts=0x9D58
```

送信系のコマンドは、3 種類で、

d: データフレーム（CAN パケット, 8 バイト）を送信
D: データフレーム（CANFD パケット, 64 バイト）を送信
r: リモートフレーム（CAN パケットで RTR=1, DLC=8）を送信

上記動作を行います。ID=0x00000000 で、送信データは 0x00 からのインクリメントデータです。

ID や送信データに関しては、can_monitor.c 内に定義されているので、ソースを変更してプログラムをビルドすれば、変更可能です。

パケットの受信、端末への画面表示に関しては、UART で 115,200bps なので、CAN バスに多数のデータが流れている場合は、表示の部分で抜けが出ます。

通信速度に関しては、任意の速度という訳には行かないのですが、データに関しては任意のデータを受信する事ができますので、通信相手がどのようなプロトコルで送信しているか不明な場合は、簡易 CAN モニタとして使用可能です。

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2024.3.28	—	初版発行
REV.1.0.1.0	2024.4.4	P4	誤記訂正

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 RL78 マイコン搭載
HSB シリーズマイコンボード 評価キット

LIN・CAN スタータキット RL78/F24
RS-CANFD_Lite ソフトウェア編　マニュアル

株式会社 北斗電子
 2024 年 4 月 4 日改訂 REV.1.0.1.0 (240404)