LIN・CAN スタータキット RL78/F15
RS-CAN_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/F15(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.1.0.0.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

警告 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

注意 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプの点灯中に電源を切ったり、パソコンをリセットをしないでください。

製品の故障や、データ消失の原因となります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

概要

本書は、「LIN・CAN スタータキット RL78/F15」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

フォルダ			内容
SOURCE¥	RL78F15_LIN¥		LIN（SLAVE からのデータ送信） サンプルプログラム(*1)
	RL78F15_LIN2¥		LIN（MASTER からのデータ送信） サンプルプログラム(*1)
	RL78F15_CAN¥		CAN（CAN0→CAN1 データ送信） サンプルプログラム(*1)
	RL78_F15_CAN_S1¥		CAN 送信バッファ、受信バッファ 使用データフレーム送受信 サンプルプログラム(*2)
	RL78_F15_CAN_S2¥		CAN 送受信 FIFO、受信 FIFO 使用データフレーム送受信 サンプルプログラム(*2)
	RL78_F15_CAN_S3¥		CAN 送受信 FIFO、受信 FIFO 使用データフレーム/リモートフレーム送受信サンプルプログラム(*2)
	64PIN		HSBRL78F13-64 HSBRL78F14-64(*3) HSBRL78F15-64(*3) 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(*2)

(*1)に関しましては、LIN_CAN_KIT_RL78F15_software_REV_X_s.pdf を参照ください。
本資料は、(*2)のプログラムの動作に関して解説しているものです。

(*3)搭載マイコンが、RL78/F14, F13 になりますので、RL78/F15 と一部相違があります
[RL78/F14, F13: RL78/F15 との相違点を記載]

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

RL78_F15_CAN_S1 (SAMPLE1)
RL78_F15_CAN_S2 (SAMPLE2)
RL78_F15_CAN_S3 (SAMPLE3)

の動作を示します。

1.1. サンプルプログラム動作時の接続形態

1.1.1. キット付属ボード(HSBRL78F15-100)を 1 台使用する場合

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。

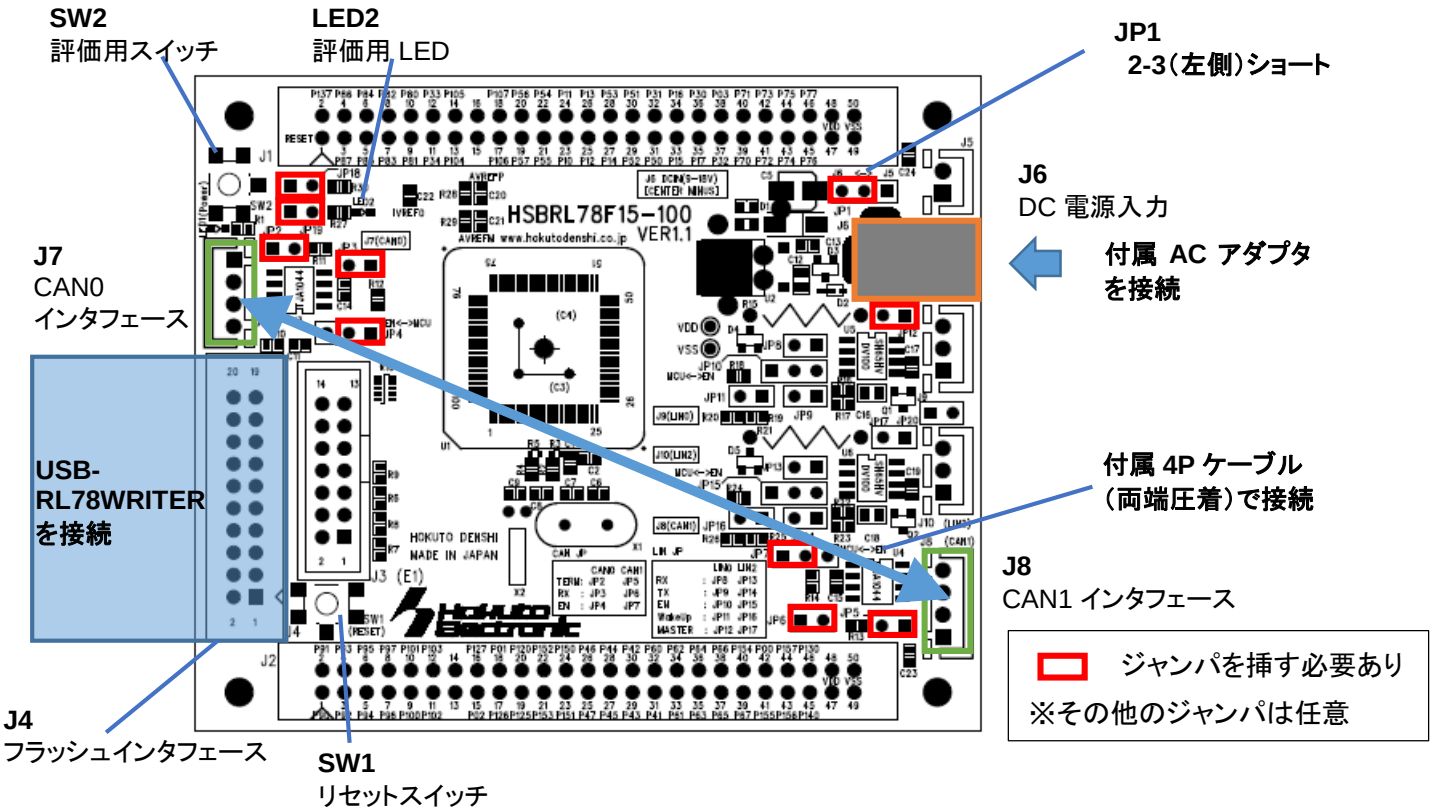


図 1-1 キット接続形態(1)

・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 J8 は、付属 4P ケーブルで接続する

ージャンパ設定ー

No	設定	用途	備考
JP1	2-3 ショート	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	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
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）
JP5	ショート	CAN1-TERM	CAN1 の終端抵抗を有効化
JP6	ショート	CAN1-TX	CAN1 トランシーバ IC の出力をマイコンに接続
JP7	1-2 ショート	CAN1-STB	マイコンの出力を CAN1 トランシーバ IC に接続（マイコンから STB 制御）

1.1.2. HSBRL78F15-100 を 2 台使用する場合

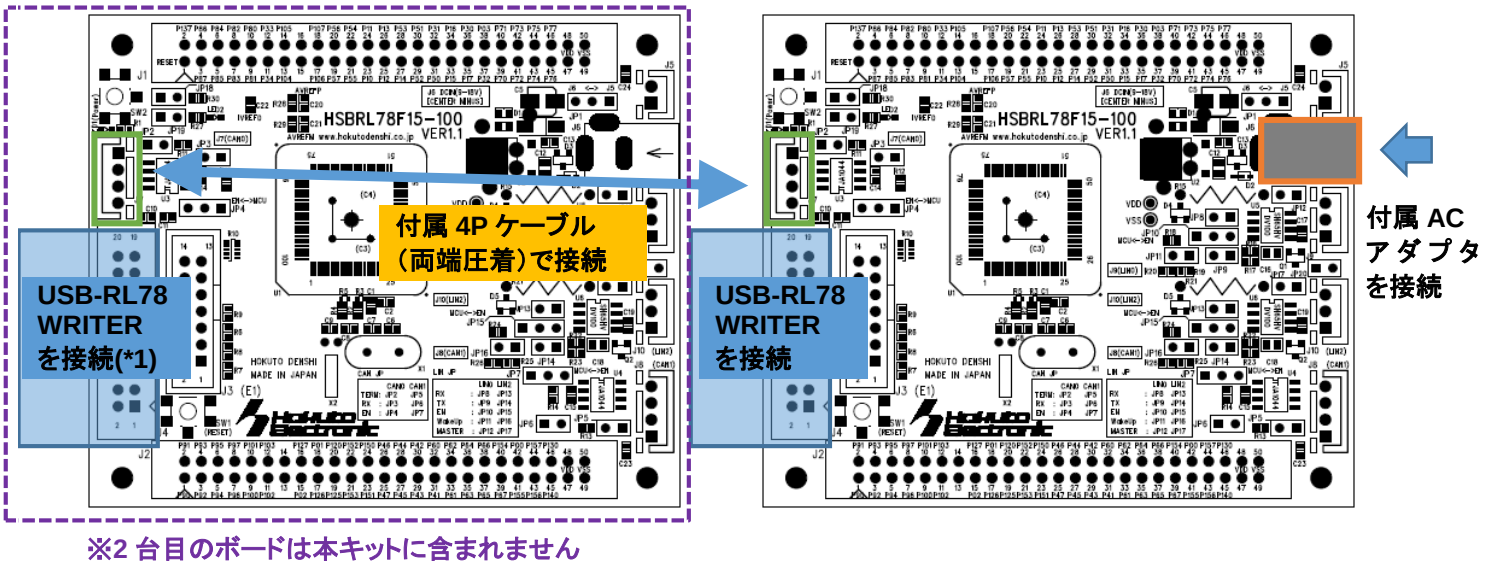


図 1-2 キット接続形態(2)

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）

・接続 CAN ポートは任意

（上記では CAN0 同士を接続していますが、CAN1 同士でも、CAN0-CAN1 の組み合わせでも可）

・CAN を動作させるためのジャンパは 2.1.1 と同様に設定する

(*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-RL78WRITER の他、USB-ADAPTER も使用可能です。

1.1.3. 当社製 RL78 の CAN 対応 64pin ボード(*1)を使用する場合

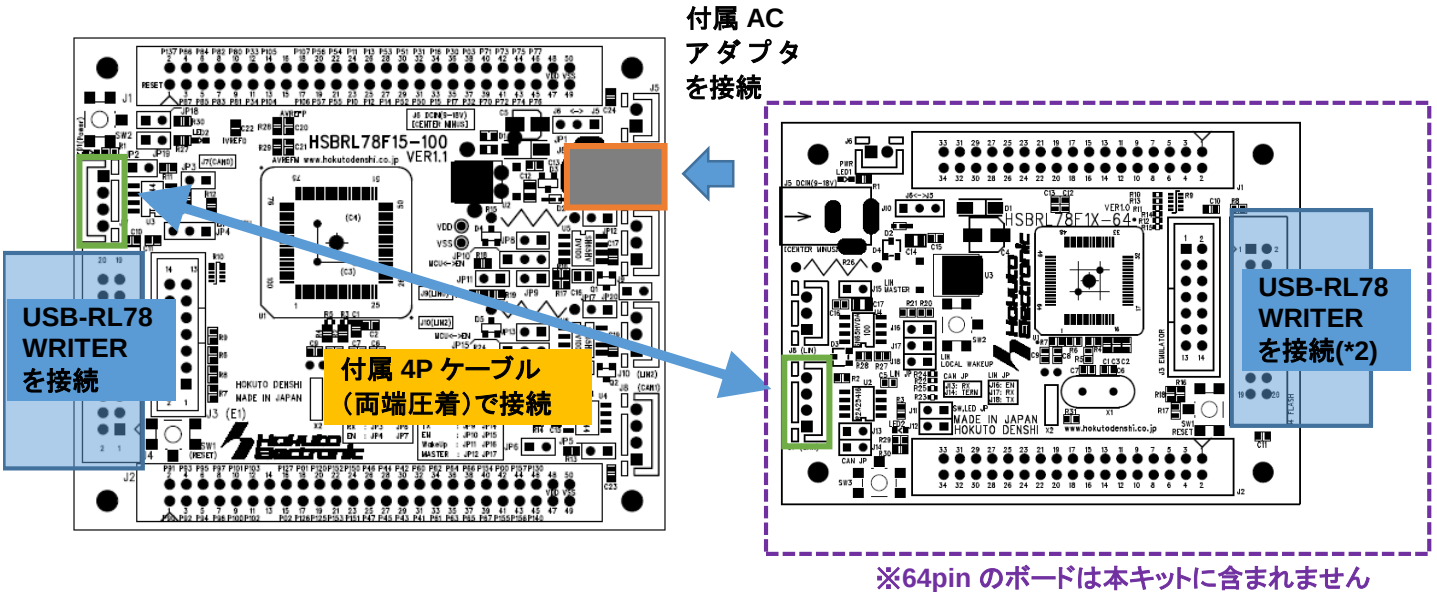


図 1-3 キット接続形態(3)

(*1)通信相手として、当社製の
HSBRL78F13-64
HSBRL78F14-64
HSBRL78F15-64
を接続して動作を確認する事が可能です。(*1)のボードに書き込むプログラムは、CD の SOURCE¥64PIN 以下に含まれています。※64pin のボード向けのサンプルプログラムは SAMPLE3 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）

・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）
・接続 CAN ポートは任意
（上記では CAN0 を接続していますが、CAN1 でも可）
・CAN を動作させるためのジャンパは 2.1.1 と同様に設定する
(*2)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-RL78WRITER の他、USB-ADAPTER も使用可能です。

1.1.4. 当社製「CAN スタータキット RX/RA」の組み合わせ

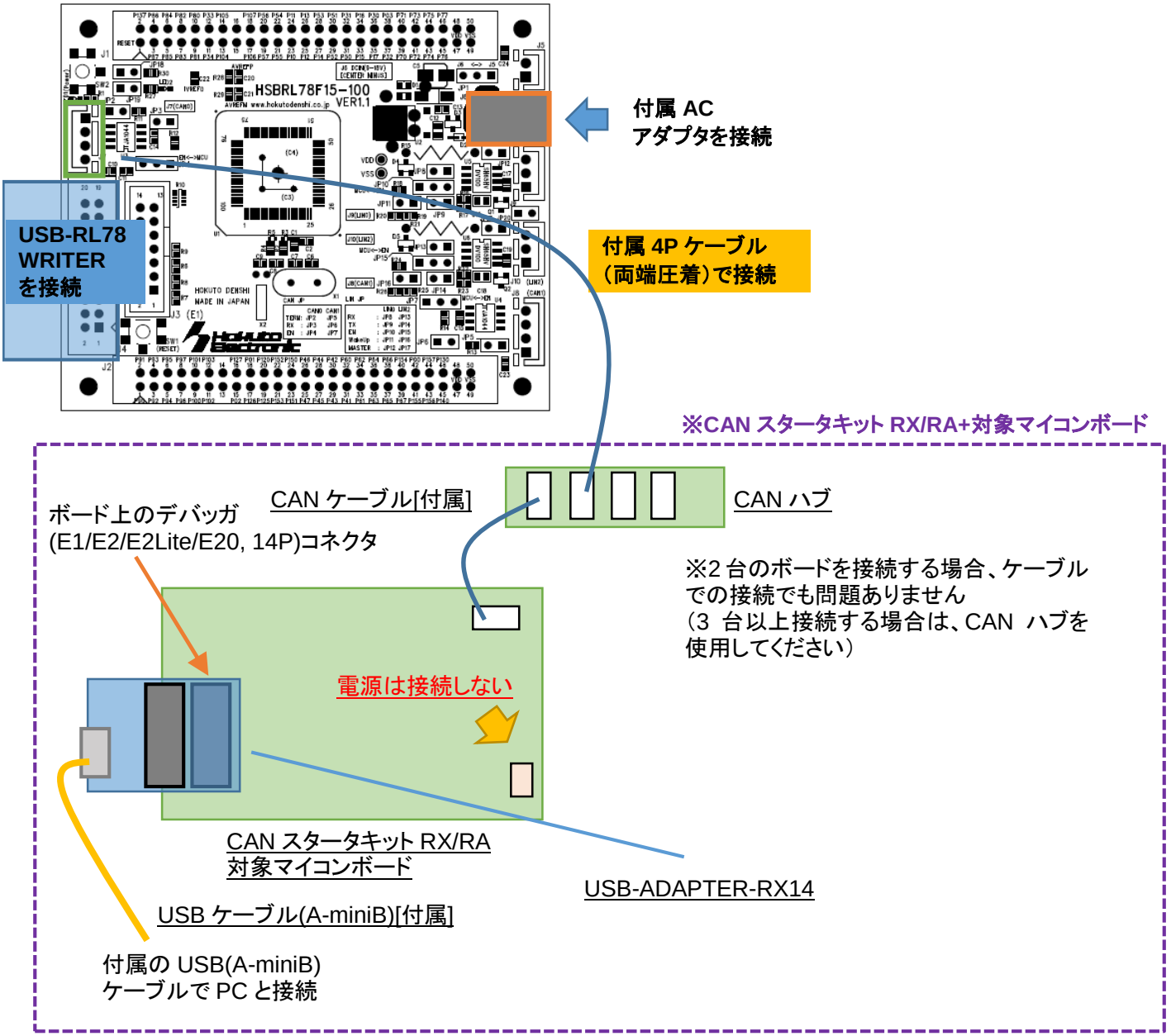


図 1-4 キット接続形態(4)

・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）
・接続 CAN ポートは任意
（上記では CAN0 を接続していますが、CAN1 でも可）
・CAN を動作させるためのジャンパは 2.1.1 と同様に設定する

SAMPLE1~SAMPLE3 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。

「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（RX231, RX24U, RX24T, RX64M, RX651, RX660, RX671, RX66N, RX66T, RX71M, RX72M, RX72N, RX72T)及び、
RA マイコン(RA2A1, RA2L1, RA4E1, RA4M1, RA4M2, RA6M1, RA6M2, RA6M3, RA6M4, RA6M5, RA6T1, RA6T2)
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/F15 の SAMPLE1~SAMPLE3 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
HSBRL78F15-100 のボードでは、JP2 が CAN0 の終端抵抗、JP5 が CAN1 の終端抵抗となります。

1.2. 1 台の HSBRL78F15-100 ボードでの動作

接続形態 2.1.1 で示す、1 台のボードで CAN0-CAN1 間の通信を行わせる場合の動作です。

1.2.1. 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) HSBRL78F15-100

```
HSBRL78F15-100 CAN Starter kit program boot.
Copyright (C) 2022 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
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

HSBRL78F15-100 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

本サンプルプログラムでは、端末のキーボードから、数字の 0~3 を押すと CAN のデータフレームを CAN バスに送出する様になっています。

マイコンボード端末で、"0"を押した場合、CAN0 ポートからデータが送信され、CAN1 ポートがデータを受信します。データとしては 0x01（1 バイト）を送信・受信します。

```
--PACKET START--
CAN0 data frame send, id_type=EID id=0x00000000 data=0x01 ret=0
CAN1 data received, buf=20 ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01
ts=0xD3D3
--PACKET END--
```

CAN0 data frame send の行がデータの送信を行っている表示です。ID タイプは拡張 ID（EID, 29 ビットの ID）で、ID 値は 0x00000000、RTR 値は 0x00（RTR 値は後述）、データ(data)は 0x01（1 バイト）、送信時（データ送信関数）の戻り値は ret=0（正常終了）です。

CAN1 data received はデータの受信。データの受信に使用した受信バッファ(buf)は、20 番。ret は受信関数の戻り値で、受信したデータバイト数(1)。ID タイプは拡張 ID(EID)で、ID 値は 0x00000000、RTR 値は 0x00、受信したデータ(data)は 0x01。ts は受信時のタイムスタンプデータです。
タイムスタンプは、受信側でデータの受信タイミング（複数のデータを受信した際、時系列的にどの順番であったか）を判断するために付与しているデータとなります。（CAN の送信データの中にはタイムスタンプの情報は含まれません）

送信データと受信データ、及び ID タイプと ID 値が一致するはずです。

端末では以下の情報を表示します。

ret= 送信時：エラーが無ければ 0, 受信時：受け取ったデータバイト数
id_type= ID タイプ（EID:拡張フォーマット, SID:標準フォーマット）
id= CAN ID
rtr= RTR 値（0:データフレーム, 1:リモートフレーム）(*1)
buf= 使用受信バッファ番号
data= 送受信データ
ts= タイムスタンプ（受信側が回しているタイマカウンタ値）

(*1)SAMPLE1~SAMPLE2 ではデータフレームしか取り扱わない様にしているので、必ず 0 です
（リモートフレームは受信しないように設定）

※--PACKET START--, --PACKET END--は一連の動作である事が判り易い様に表示しているものです

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
c: 送信元 ch をトグルで切り替える

となっています。

端末から、c と入力すると、送信元が CAN0 から CAN1 に切り替わります。

```
CAN send channel -> CAN1
```

端末から 2 を入力する事で 4 バイト送信(ID=0x2, データ=0x01234567)となります。

```
--PACKET START--
CAN1 data frame send, id_type=EID id=0x00000002 data=0x01234567 ret=0
CAN0 data received, buf=6 ret=4 id_type=EID id=0x00000002 rtr=0x00
data=0x01234567 ts=0x6D30
--PACKET END--
```

※最初と異なり、CAN1 から CAN0 にデータが送られている

コマンド s は、標準 ID への切り替え

```
send format -> SID
```

コマンド 3 では、ID=0x3 で 8 バイト送信(0x12345678ABCDEF)となりますが、

```
--PACKET START--
CAN1 data frame send, id_type=SID id=0x00000003 data=0x0123456789ABCDEF ret=0
CAN0 data received, buf=3 ret=8 id_type=SID id=0x00000003 rtr=0x00
data=0x0123456789ABCDEF ts=0x9294
--PACKET END--
```

送信、受信とも SID(標準 ID)となっています。

c コマンドと s コマンドはトグルなので、もう一度入力すると、送信元 CAN1→CAN0、標準 ID→拡張 ID に変わります。

RL78/Fx 系マイコンに搭載されている、RS-CAN_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	送信 ch	キーボードからのコマンド
0	CAN0	0
1	CAN0	1
2	CAN0	2
3	CAN0	3
4	CAN1	0
5	CAN1	1
6	CAN1	2
7	CAN1	3

送信バッファは、CAN0 側が 0~3、CAN1 側が 4~7 で、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。[RL78/F14,F13: ch は CAN0 のみ、送信バッファは 0~3]

・受信

受信ルール番号	受信 ch	フォーマット	ID	受信バッファ番号
0	CAN0	標準(SID)	0x000	0
1	CAN0	標準(SID)	0x001	1
2	CAN0	標準(SID)	0x002	2
3	CAN0	標準(SID)	0x003	3
4	CAN0	拡張(EID)	0x0000000	4
5	CAN0	拡張(EID)	0x0000001	5
6	CAN0	拡張(EID)	0x0000002	6
7	CAN0	拡張(EID)	0x0000003	7
0+offset1(*1)	CAN1	標準(SID)	0x000	0+offset2(*2)
1+offset1(*1)	CAN1	標準(SID)	0x001	1+offset2(*2)
2+offset1(*1)	CAN1	標準(SID)	0x002	2+offset2(*2)
3+offset1(*1)	CAN1	標準(SID)	0x003	3+offset2(*2)
4+offset1(*1)	CAN1	拡張(EID)	0x0000000	4+offset2(*2)
5+offset1(*1)	CAN1	拡張(EID)	0x0000001	5+offset2(*2)
6+offset1(*1)	CAN1	拡張(EID)	0x0000002	6+offset2(*2)
7+offset1(*1)	CAN1	拡張(EID)	0x0000003	7+offset2(*2)

RL78/F15 では、最大 40 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 32 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。)[RL78/F14,F13: 受信ルールと受信バッファは最大 16]

(*1)offset1=20　受信ルールは、CAN0/CAN1 トータルで 40 個です。プログラム上は、CAN0 で 0~19、CAN1 でも 0~19 で設定しますが、実際には CAN1 側は 20 のオフセットが付いた値（受信ルール番号 1 は、実際は受信ルール番号 21 で設定）となります。

(*2)offset2=16　受信バッファは、CAN0/CAN1 トータルで 32 個です。CAN0 で 0~15、CAN1 でも 0~15 で設定しますが、実際には CAN1 側は 16 のオフセットが付いた値（受信バッファ番号 1 は、実際は受信バッファ番号 17 で設定）となります。

データの受信に関しては、標準・拡張フォーマット区分(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 CAN0 側が受信した際、

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 n:CAN ch
0	0x0000000	1	0x01	0+n×4
1	0x0000001	2	0x0123	1+n×4
2	0x0000002	4	0x01234567	2+n×4
3	0x0000003	8	0x0123456789ABCDEF	3+n×4

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。CAN ch0 側は 0~3 の番号の送信バッファ、CAN ch1 側は、4~7 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）

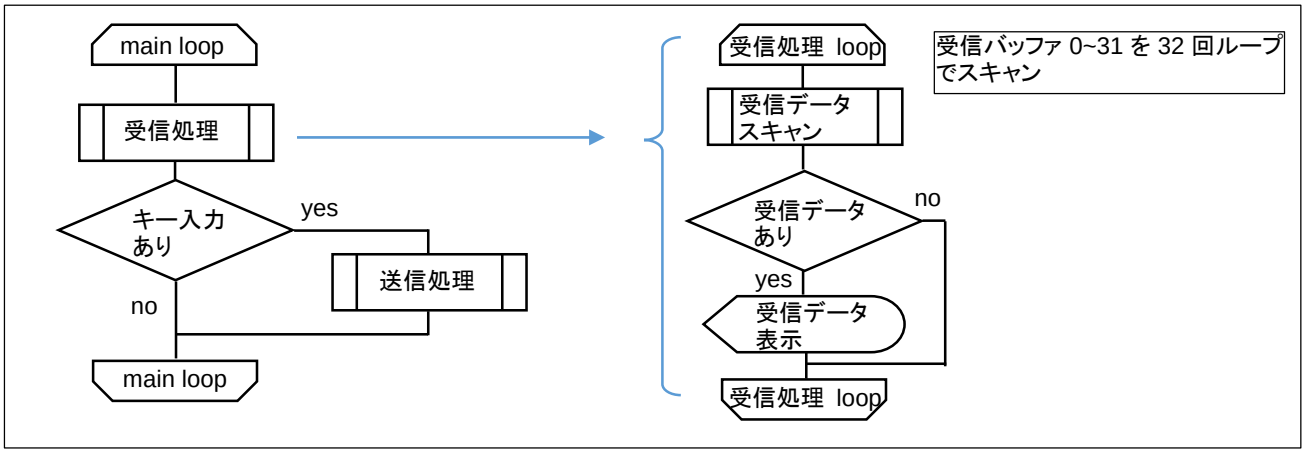
"s"コマンドで、送信フォーマット、「標準フォーマット」←→「拡張フォーマット」の切り替えを行います。

また、キーボードから入力可能なコマンドとして"z"がありますが、このコマンドを使用すると、ボード上の LED が一瞬点滅します。PC 上の端末と、物理的にどのマイコンボードがつながっているか、対応が判らなくなった際に使用可能なコマンドです。

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）

1.2.2. 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。

```
HSBRL78F15-100 CAN Starter kit program boot.
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

SAMPLE2: CAN [Data frame] send/receive program(with interrupt, use FIFO).
CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  z: LED blink test(for board identify)
  s: send fornat EID <-> SID
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

端末で、"2"を押した場合、CAN0→CAN1 にバイト送信します。(SAMPLE1 と動作としては変わりません。)

```
--PACKET START--
CAN0 data frame send, id_type=EID id=0x00000002 data=0x01234567 ret=0
CAN1 data received, ret=4 id_type=EID id=0x00000002 rtr=0x00 data=0x01234567
ts=0xE880
CAN0 send finished.(SRFIFO)
--PACKET END--
```

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

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 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。
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 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-CAN_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.h で変更可能です
※受信に受信バッファを選択した場合は、割り込みを使って受信処理を行う事は出来ません
※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
CAN1	1	4

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、CAN0 は送受信 FIFO の 0 番。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 32 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。
[RL78/F14,F13: 送受信 FIFO は 0 のみで 1 本]

・受信

受信ルール番号	受信 ch	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	CAN0	標準(SID)	0x000	0	0	0
1	CAN0	標準(SID)	0x001	0	0	1
2	CAN0	標準(SID)	0x002	0	0	2
3	CAN0	標準(SID)	0x003	0	0	3
4	CAN0	拡張(EID)	0x0000000	0	0	4
5	CAN0	拡張(EID)	0x0000001	0	0	5
6	CAN0	拡張(EID)	0x0000002	0	0	6
7	CAN0	拡張(EID)	0x0000003	0	0	7
0(+20)	CAN1	標準(SID)	0x000	1	1	0(+16)
1(+20)	CAN1	標準(SID)	0x001	1	1	1(+16)
2(+20)	CAN1	標準(SID)	0x002	1	1	2(+16)
3(+20)	CAN1	標準(SID)	0x003	1	1	3(+16)
4(+20)	CAN1	拡張(EID)	0x0000000	1	1	4(+16)
5(+20)	CAN1	拡張(EID)	0x0000001	1	1	5(+16)
6(+20)	CAN1	拡張(EID)	0x0000002	1	1	6(+16)
7(+20)	CAN1	拡張(EID)	0x0000003	1	1	7(+16)

(*1)デフォルトは受信 FIFO で受信、can.h で選択可能

(*2)

受信 FIFO は 0~3 の 4 本あり、サンプルプログラムでは CAN0 の受信に 0 番、CAN1 の受信に 1 番を割り当てています。[RL78/F14,F13: 受信 FIFO は 0~1 の 2 本]

受信 FIFO は 0 番, 1 番共に、4 段の設定です。トータル 40 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。[RL78/F14,F13: メッセージバッファは 16]

受信 FIFO は、CAN0/CAN1 の縛りはなし。送受信 FIFO を受信に使用する場合は、CAN0 と送受信 FIFO の 0 番、CAN1 と送受信 FIFO の 1 番が紐付けとなります。

CAN の割り込みは、全部で 10 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（受信 FIFO で受信時）
・CAN0 送受信 FIFO 受信
・CAN1 送受信 FIFO 受信 [RL78/F14,F13 では存在しない]
・CAN0 チャネル送信（送受信 FIFO 送信時、送信バッファ送信時）
・CAN1 チャネル送信（送受信 FIFO 送信時、送信バッファ送信時） [RL78/F14,F13 では存在しない]

の 5 本を使用しています。受信 FIFO で受信（サンプルプログラムのデフォルト）とした場合、CAN0/CAN1 どちらで受信した場合でも「CAN グローバル受信」の割り込みとなります。送受信 FIFO 受信と送信割り込みは ch 毎に別な割り込みとなっています。

※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）

1.2.3. 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。

```
HSBRL78F15-100 CAN Starter kit program boot.
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

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

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
  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）。

```
--PACKET START--
CAN0 remote frame send, id_type=EID id=0x00000000 dlc=1 ret=0   (1)
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x78A1   (2)
CAN1 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0   (3)
CAN0 send finished.(SRFIFO)   (1')
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x7ADA   (4)
CAN1 send finished.(SRFIFO)   (3')
--PACKET END--
```

(1)CAN0 から ID=0 に対して、DLC=1 でリモートフレームのパケットを送信
(2)CAN1 がリモートフレームを受信
(3)CAN1 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)CAN0 が CAN1 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。(3')も同様です。

本サンプルプログラムでは、多重割込みを禁止していますので、表示上(1')は(3)の後となっていますが、実際の時系列は、下記のようになります。

```
--PACKET START--
CAN0 remote frame send, id_type=EID id=0x00000000 dlc=1 ret=0
CAN0 send finished.(SRFIFO)
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x1BE9
CAN1 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0
CAN1 send finished.(SRFIFO)
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x1EA4
--PACKET END--
```

※r_cg_userdefine.h　内の定義で多重割り込みを許可する設定が可能です
※別なマイコンボードと通信を行った際、表示が乱れる（メッセージの途中で、別なメッセージが割り込む）ケースがありますので、デフォルトでは多重割込みを行わないコードとしています

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）。

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)	受信バッファ番号(*1)
0	CAN0	標準(SID)	0x000	0	0	0	0
1	CAN0	標準(SID)	0x001	0	0	0	1
2	CAN0	標準(SID)	0x002	0	0	0	2
3	CAN0	標準(SID)	0x003	0	0	0	3
4	CAN0	標準(SID)	0x000	1	0	0	4
5	CAN0	標準(SID)	0x001	1	0	0	5
6	CAN0	標準(SID)	0x002	1	0	0	6
7	CAN0	標準(SID)	0x003	1	0	0	7
8	CAN0	拡張(EID)	0x0000000	0	0	0	8
9	CAN0	拡張(EID)	0x0000001	0	0	0	9
10	CAN0	拡張(EID)	0x0000002	0	0	0	10
11	CAN0	拡張(EID)	0x0000003	0	0	0	11
12	CAN0	拡張(EID)	0x0000000	1	0	0	12
13	CAN0	拡張(EID)	0x0000001	1	0	0	13
14	CAN0	拡張(EID)	0x0000002	1	0	0	14
15	CAN0	拡張(EID)	0x0000003	1	0	0	15
0(+20)	CAN1	標準(SID)	0x000	0	1	1	0(+16)
1(+20)	CAN1	標準(SID)	0x001	0	1	1	1(+16)
2(+20)	CAN1	標準(SID)	0x002	0	1	1	2(+16)
3(+20)	CAN1	標準(SID)	0x003	0	1	1	3(+16)
4(+20)	CAN1	標準(SID)	0x000	1	1	1	4(+16)
5(+20)	CAN1	標準(SID)	0x001	1	1	1	5(+16)
6(+20)	CAN1	標準(SID)	0x002	1	1	1	6(+16)
7(+20)	CAN1	標準(SID)	0x003	1	1	1	7(+16)
8(+20)	CAN1	拡張(EID)	0x0000000	0	1	1	8(+16)
9(+20)	CAN1	拡張(EID)	0x0000001	0	1	1	9(+16)
10(+20)	CAN1	拡張(EID)	0x0000002	0	1	1	10(+16)
11(+20)	CAN1	拡張(EID)	0x0000003	0	1	1	11(+16)
12(+20)	CAN1	拡張(EID)	0x0000000	1	1	1	12(+16)
13(+20)	CAN1	拡張(EID)	0x0000001	1	1	1	13(+16)
14(+20)	CAN1	拡張(EID)	0x0000002	1	1	1	14(+16)
15(+20)	CAN1	拡張(EID)	0x0000003	1	1	1	15(+16)

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。

※なお、このルールは ch 毎に適用されます

受信ルール、0~19 を CAN0 向け、20~39 を CAN1 向けとした場合、0~19 内に欠番が合ったとしても、CAN1 側のルールである 20~は有効です。（なお、21 が欠番だった場合は、22~は無効となります。）

1.3. HSBRL78F15-100 と他のボードを接続した場合での動作

接続形態 2.1.3 で示す、HSBRL78F15-100 と HSBRL78F15-64 のボードで通信を行わせる場合の動作です。

※HSBRL78F15-100 は CAN0 側を接続（CAN1 側は未接続）

1.3.1. サンプルプログラム(SAMPLE1)

マイコンボード(1) HSBRL78F15-100（CAN0 ポート）

マイコンボード(2) HSBRL78F15-64　　※（別売）本キットには含まれません

の場合の表示例を以下に示します。

```
HSBRL78F15-100 CAN Starter kit program boot.
Copyright (C) 2022 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
  c: send CAN ch change
  (Push-SW: Data frame send [=keyboard 0])
```

```
HSBRL78F15-64 CAN Starter kit program boot.
Copyright (C) 2022 HokutoDenshi. All Rights Reserved.

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

CAN ID mode -> EID

Command Usage:
  0123: Data frame send
  qwer: Remote frame send(data request) ※通信相手が SAMPLE1 の時は反応しません
  z: LED blink test(for board identify)
  s: send fornat EID <-> SID
  (Push-SW: Data frame send [=keyboard 0])
```

通信相手の（ここでは HSBRL78F15-64）マイコンボードに、SAMPLE3 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。（※HSBRL78F15-64 向けには SAMPLE3 のプログラムを用意しています。SAMPLE1 と SAMPLE3 のプログラムは通信可能です。「CAN スタータキット RX/RA では、SAMPLE1~SAMPLE3 のプログラムが含まれます）。

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。

```
--PACKET START--
CAN0 data frame send, id_type=EID id=0x00000000 data=0x01 ret=0
--PACKET END--
```

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, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x8B2F
--
```

次に、マイコンボード(2)の端末で、"1"を押した場合、マイコンボード(2)がデータとして 0x01 0x23（2 バイト）を送信します。

```
++
CAN0 data frame send, id_type=EID id=0x00000001 data=0x0123 ret=0
CAN0 send finished.(SRFIFO)
--
```

このとき、マイコンボード(1)側ではデータを受信します。

```
--PACKET START--
CAN0 data received, buf=5 ret=2 id_type=EID id=0x00000001 rtr=0x00 data=0x0123
ts=0x140E
--PACKET END--
```

基本的には 1 台で動作せた場合と変わる事はないのですが、(1)のボードから(2)のボードに対して、データが送られている様子。また、その逆の(2)から(1)に対してのデータを送っている様子が直感的に判り易いかと思います。

1.3.2. サンプルプログラム(SAMPLE2)

マイコンボード(1) HSBRL78F15-100（CAN0 ポート）

マイコンボード(2) HSBRL78F15-64　　※（別売）本キットには含まれません

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。

```
--PACKET START--
CAN0 data frame send, id_type=EID id=0x00000000 data=0x01 ret=0
CAN0 send finished.(SRFIFO)
--PACKET END--
```

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, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x0843
--
```

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, id_type=EID id=0x00000001 data=0x0123 ret=0
CAN0 send finished.(SRFIFO)
--
```

(1)のマイコンボード(HSBRL78F15-100)側でデータを受信します。

```
--PACKET START--
CAN0 data received, ret=2 id_type=EID id=0x00000001 rtr=0x00 data=0x0123
ts=0x3402
--PACKET END--
```

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。

1.3.1. サンプルプログラム(SAMPLE3)

マイコンボード(1) HSBRL78F15-100（CAN0 ポート）

マイコンボード(2) HSBRL78F15-64　　※（別売）本キットには含まれません

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、"q"を押した場合、マイコンボード(1)がリモートフレームを送信。DLC=0x01（1 バイト）を要求します。

```
--PACKET START--
CAN0 remote frame send, id_type=EID id=0x00000000 dlc=1 ret=0
CAN0 send finished.(SRFIFO)
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0xF8B2
--PACKET END--
```

(1)

(4)

データを受信するところが SAMPLE2 との相違です。

マイコンボード(2)では、リモートフレームの受信及びデータフレームの応答を送信します。

```
++
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0xD121
CAN0 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0
CAN0 send finished.(SRFIFO)
--
```

(2)

(3)

動作は一瞬で完了しますが、
(1)リモートフレームの送信、送信完了
(2)リモートフレームの受信
(3)データフレームの送信
(4)データフレームの受信
の順番での、通信のやり取りとなります。

[参考] HSBRL78F15-100 の CAN0/CAN1 両方と他のボードを接続した場合

1 つの CAN バスに、HSBRL78F15-100 の 2 つの CAN ポートと、他のマイコンボードの合計 3 つのポートを接続した場合の SAMPLE3 の動作例を示します。

マイコンボード(1) HSBRL78F15-100（CAN0 ポート,CAN1 ポート）

マイコンボード(2) HSBRL78F15-64　　※（別売）本キットには含まれません

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。

```
--PACKET START--
CAN0 data frame send, id_type=EID id=0x00000000 data=0x01 ret=0
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x80A9
CAN0 send finished.(SRFIFO)
--PACKET END--
```

```
++
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x149B
--
```

HSBRL78F15-100 CAN0 が送信したデータを、HSBRL78F15-100 CAN1 と HSBRL78F15-64 が受信します。

マイコンボード(2)の端末で、"0"を押した場合。

```
++
CAN0 data frame send, id_type=EID id=0x00000000 data=0x01 ret=0
CAN0 send finished.(SRFIFO)
--
```

```
--PACKET START--
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x1ADE
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0x01 ts=0x1ADF
--PACKET END--
```

HSBRL78F15-100 の CAN0 と CAN1 の両方が受信します。

マイコンボード(1)の端末で、"q"（リモートフレーム要求）を押した場合。

```
--PACKET START--
CAN0 remote frame send, id_type=EID id=0x00000000 dlc=1 ret=0 (1)
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x946C (2)
CAN1 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0 (3)
CAN0 send finished.(SRFIFO) (1')
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x96A4 (4)
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x96F5 (5)
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x96F6 (6)
CAN1 send finished.(SRFIFO) (3')
--PACKET END--
```

```
++
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x7D31 (7)
CAN0 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0 (8)
++
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0xA0C1 (9)
--
CAN0 send finished.(SRFIFO) (8')
--
```

(1)HSBRL78F15-100 CAN0　リモートフレーム要求
　に対し、
(2)HSBRL78F15-100 CAN1　リモートフレーム受信
(7)HSBRL78F15-64　リモートフレーム受信
　の 2 者が受信し、
(3)HSBRL78F15-100 CAN1　データフレーム送信
(8)HSBRL78F15-64　データフレーム送信
　となり、
(4)(9)HSBRL78F15-100 CAN1 が送信したデータフレームを受信
(5)(6)HSBRL78F15-64 が送信したデータフレームを受信
　という動作です。

　2 者がデータフレームを送信しているので、表示が錯綜していますが、規定の動作となります。

※(4)(5)はこの動作例では、どちら（HSBRL78F15-100 CAN1 と HSBRL78F15-64）が送ったデータを受信したものなのかは区別できません

マイコンボード(2)の端末で、"q"（リモートフレーム要求）を押した場合。

```
++
CAN0 remote frame send, id_type=EID id=0x00000000 dlc=1 ret=0
CAN0 send finished.(SRFIFO)
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x2985
--
++
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x4A97
--
```

```
--PACKET START--
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x7A27
CAN0 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x01 dlc=1 ts=0x7A27
CAN1 response data send, id_type=EID id=0x00000000 data=0xA1 ret=0
CAN1 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x7C9F
CAN0 data received, ret=1 id_type=EID id=0x00000000 rtr=0x00 data=0xA1 ts=0x7EAF
CAN0 send finished.(SRFIFO)
CAN1 send finished.(SRFIFO)
--PACKET END--
```

HSBRL78F15-100 の CAN0, CAN1 の両方がレスポンスを返します。その結果、HSBRL78-F15-64 では 2 つの応答を受け取り、HSBRL78F15-100 の CAN0, CAN1 もそれぞれ逆側のポートが送信したレスポンスデータを受け取っています。

上記の様に、CAN0/CAN1 を同一 CAN バスに接続して、さらに別なボードを同一の CAN バスに接続すると、リモートフレーム応答を返すデバイスが複数台となり、表示が混雑しますが、一つずつメッセージを追いかけると動作が判るかと思います。

2. ソースファイル構成

フォルダ	ファイル	説明
RL78_F15_CAN_S*n*¥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
	canfd_ch0.c	ch0 関数
	canfd_ch0.h	ch0 関数用ヘッダファイル
	canfd_ch1.c	ch1 関数
	canfd_ch1.h	ch1 関数用ヘッダファイル
	can_intr_s*n*.c	SAMPLE*n* に対応した割込み処理
	can_operation.h	速度や標準/拡張 ID の定義
RL78_F15_CAN_S*n*¥board		ボード固有定義フォルダ
	board.h	ボード固有定義ファイル
RL78_F15_CAN_S*n*¥sci		SCI ソースフォルダ
	sci.c	SCI(UART)処理関数
	sci.h	sci.c 向けヘッダファイル

n: 1~3, SAMPLE1~3 に対応

2.1. ヘッダファイルの設定

(1) can.h

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

//通信速度
#define CAN_BPS_1M      1   //1M bps
#define CAN_BPS_500K    2   //500k bps
#define CAN_BPS_250K    4   //250k bps
#define CAN_BPS_125K    8   //125k bps

//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
#define CAN_REMOTE_FRAME    1

//受信ルール
#defineCAN_RULE_BUF         0x0   //受信メッセージバッファ
#defineCAN_RULE_RXFIFO0     0x0001   //RXFIFO0
#defineCAN_RULE_RXFIFO1     0x0002   //RXFIFO1
#defineCAN_RULE_RXFIFO2     0x0004   //RXFIFO2
#defineCAN_RULE_RXFIFO3     0x0008   //RXFIFO3
#defineCAN_RULE_SRFIFO0     0x0010   //SRFIFO0
#defineCAN_RULE_SRFIFO1     0x0020   //SRFIFO1

//CAN の受信方法
#define RXBUF                       0
#define SRFIFO                      1
#define RXFIFO                      2

//3 種類の受信方法から 1 つを選択（3 行の内いずれかを有効にしてください）※SAMPLE2-3 で設定が有効（SAMPLE1 は受信バッファを使用）
#define CAN_RX_METHOD               RXFIFO          //受信 FIFO を使用[デフォルト]  ※いずれを有効化
//#define CAN_RX_METHOD             SRFIFO          //送受信 FIFO を使用  ※いずれかを有効化(*1)
//#define CAN_RX_METHOD             RXBUF           //受信バッファを使用  ※いずれかを有効化(*2)

//CAN の送信方法
#define TXBUF                       0
//#define SRFIFO                    1//定義済み

//2 種類の送信方法から 1 つを選択（2 行の内どちらかを有効にしてください）※SAMPLE2-3 で設定が有効（SAMPLE1 は送信バッファを使用）
#define CAN_TX_METHOD               SRFIFO          //送受信 FIFO を使用[デフォルト]  ※どちらかを有効化(*1)
//#define CAN_TX_METHOD             TXBUF           //送信バッファを使用  ※どちらかを有効化
```

グレーの部分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

(a)受信方法の選択

(b)送信方法の選択

(a)は受信方法の選択です。「受信 FIFO」「送受信 FIFO」「受信バッファ」のいずれか 1 つを有効化（コメントアウトを外す）してください。(b)は送信方法の選択です。「送受信 FIFO」「送信バッファ」のどちらかを有効化してください。

(1) can.h（続き）

```
#if (CAN_RX_METHOD == SRFIFO) && (CAN_TX_METHOD == SRFIFO)

#error "Error: SRFIFO select error"

//SRFIFO は ch 毎に 1 本
//送信か受信かどちらか一方でしか使用できない

#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(0-3 を指定)
[CAN_RX_METHOD -> RXFIFO 時使用]
#define CAN1_RX_RXFIFO_NO   1          //CAN1 受信を RXFIFO で行う場合の FIFO 番号(0-3 を指定)
[CAN_RX_METHOD -> RXFIFO 時使用]

/*
(*1)CAN の受信に送受信 FIFO を使った場合、送信で送受信 FIFO を使用する事はできません
(*2)CAN の受信に受信バッファを使った場合、割り込みでの処理はできません
*/

//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
```

送受信バッファは ch 毎に 1 本なので、
受信:「送受信バッファ」
送信:「送受信バッファ」
の設定はできません。

(c)受信 FIFO 番号の選択

(d)端子デバッグの使用

(c)は、受信方法として受信バッファを指定した場合、使用する FIFO 番号(0~3)を指定してください。（CAN0 と CAN1 で重複しないように定義）

(d)は定数を定義すると割込み時に、端子が反転する事で、割り込みの有無やタイミングをデバッグに使用できます。

(2) can_operation.h

```
//速度設定
#define CAN_SPEED   CAN_BPS_1M

//いずれかを設定
//CAN_BPS_1M   //1M bps
//CAN_BPS_500K   //500k bps
//CAN_BPS_250K   //250k bps
//CAN_BPS_125K   //125k bps


//ID 設定
#define 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)ID フォーマットの選択

(a)通信速度は、1Mbps~125kbps の 4 種類が選択可能です。（それ以外の速度は、ご自身で分周比等を計算して設定してください。(b)の ID フォーマットは、CAN_ID_FORMAT_EID（デフォルト）とすると、標準 ID と拡張 ID の両方のデータを受信します。CAN_ID_FORMAT_SID とすると、標準 ID のデータしか受信しません。

(3) r_cg_userdefine.h

```
//CAN0 と CAN1 を接続（表示上だけの処理）
#define CAN0_CAN1_CANBUS_SHARE          (a)表示フォーマットの選択
/*
CAN0 と CAN1 を同一の CAN バスに接続する際に定義
一連の処理を
--PACKET START--

--PACKET END--
表示する様になります
（表示上だけの処理です）
*/

//多重割り込みを許可
//#define CAN_MULTIPLE_INTERRUPT_EN     (b)多重割り込みの許可

//使用する割込み
#define RXFIFO_INTERRUPT_USE//受信 FIFO の割り込みを使用する
#define SRFIFO_INTERRUPT_USE//送受信 FIFO 割り込みを使用する
#define TX_BUF_INTERRUPT_USE//送信バッファの割り込みを使用する   (c)割り込み使用有無

//フラグ変数
extern unsigned short can_remote_frame_request[];
extern int g_packet_flag;
```

本ファイルは、RL78 のコード生成で自動生成されるファイルで、プロジェクトフォルダ
RL78_F15_CAN_S*n*
の直下に存在します。

(a)の表示フォーマットは、一連の処理を
--PACKET START--
--PACKET END--
で囲むか、SAMPLE3 で
++
--
で囲むかを変更するものです。CAN0 と CAN1 を同一の CAN バスに接続した場合、どこからどこまでが一連の処理なのか、判りやすく表示するための設定です。有効にした場合、最初のアクション（データ送信や受信）から 100ms 後に、--PACKET END--が表示されるという非常に単純な処理です。

(b)の多重割り込みは、CAN の割り込みルーチン内で優先度 0 の多重割り込みを許可するものです。有効にすると、割り込みのタイミングは判り易いですが、表示が乱れる（メッセージの途中で別なメッセージが割り込む）ケースがあるので、デフォルトでは無効化しています。

(c)の割り込み使用有無は、割込み使用時に有効化してください。SAMPLE1 では割り込みを使用しないので無効化。SAMPLE2~3 では有効化しています。

(4) board.h

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

//CAN マクロ種別
#define CAN_MACRO_TYPE_CAN  1
#define CAN_MACRO_TYPE_RSCAN  2
#define CAN_MACRO_TYPE_CANFD  3
#define CAN_MACRO_TYPE_CANFD_B  4
#define CAN_MACRO_TYPE_CANFD_LITE  5
#define CAN_MACRO_TYPE_RSCAN_LITE  6

//MCU タイプ
#define MCU_TYPE_RL78_F13  13
#define MCU_TYPE_RL78_F14  14
#define MCU_TYPE_RL78_F15  15

//RL78/F15 100pin

#define  MCU_TYPE  MCU_TYPE_RL78_F15

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

//CAN のポート(CAN1)
#define CAN1_TX_P61
#define CAN1_RX_P60
#define CAN1_EN_P62

//LED のポート設定
#define LED_PORT_NUM  1
#define LED1_PORT_PDR  PM0_bit.no3
#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_PDR//入出力モードの切り替え不要
#define SW1_PORT  P13_bit.no7

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

プログラムで使用している
定数定義（変更しない）

CAN で使用する I/O ポート定義
（HSBRL78F15-100 で使用する場合は変更しない）

サンプルプログラムで使用する LED の I/O ポート定義
（HSBRL78F15-100 で使用する場合は変更しない）

サンプルプログラムで使用する SW の I/O ポート定義
（HSBRL78F15-100 で使用する場合は変更しない）

(4) board.h（続き）

```
#define CAN_CH  2  //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 CAN1_VALID  1//CAN1 を使用する

#define ICLK  32  //CPU クロック[MHz]
#define FCLK2  16  //CAN のベースクロック(fCLK/2)[MHz]
#define XTAL  8  //搭載 XTAL の値[MHz]
#define CAN_CLOCK  FCLK2  //FCLK2 を CAN クロックとして使用
#define CK_DIV_BASE  2  //1Mbps 時  FCLK2/2 を CAN クロックに設定

//割り込みモニタ端子
//J2-1(P90)受信 FIFO 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)CAN1  送受信 FIFO 受信割り込み
//J2-5(P94)CAN1  チャネル送信割り込み
#define  INT_MONITOR_DIGITAL_PORT  ADPC=0x1;
#define INT1_PORT_PDR  PM9_bit.no0
#define INT2_PORT_PDR  PM9_bit.no1
#define INT3_PORT_PDR  PM9_bit.no2
#define INT4_PORT_PDR  PM9_bit.no3
#define INT5_PORT_PDR  PM9_bit.no4
#define INT1_PORT_PODR  P9_bit.no0
#define INT2_PORT_PODR  P9_bit.no1
#define INT3_PORT_PODR  P9_bit.no2
#define INT4_PORT_PODR  P9_bit.no3
#define INT5_PORT_PODR  P9_bit.no4
```

RI78/F15 の場合は変えない

2 つある CAN ポートの内、両方有効にしない場合は変更

クロック定義
（HSBRL78F15-100 に合わせています）

割り込みを I/O 端子でデバッグする場合に定義

board/board.h の定義は、ボード固有の値を定義しているファイルです。当社製の HSBRL78F15-100 以外のボードに適用する場合や、CAN0/CAN1 のどちらか一方を有効化したい場合等に編集してください。

3. サンプルプログラムの説明(共通部分)

3.1. 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、コード生成を使用しています。

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)
HSBRL78F15-100	8MHz	8×**8** (=64MHz)	32MHz	32MHz	fCLK/2 =16MHz

RL78/F15 は、CAN のクロックソース(fCAN)として
fCLK/2(16MHz)
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/2 を選択しています。

1Mbps 設定時、fCAN(=16MHz)を 2 分周して fCANTQ=8MHz としてビットクロックとしています。1Tq=125ns で、8Tq 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/2, 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）

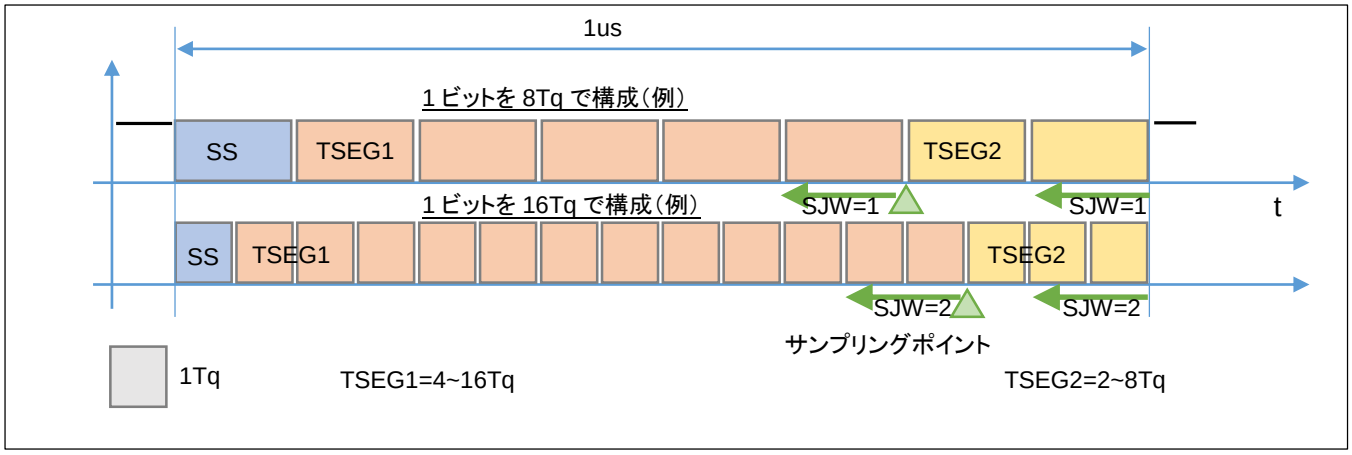


図 3-1 CAN データの 1 ビット

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~25Tq で設定する必要があり、TSEG1>TSEG2>=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、8Tq で 1 ビットを構成する場合、1Tq=125ns（8MHz のクロック源が必要）となり、25Tq で 1 ビットを構成する場合、1Tq=40ns（25MHz 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+TSEG1)/(1+TSEG1+TSEG2)
※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/2(16MHz)としていますので、1Mbps 時 1 ビットを 16Tq で構成する事も可能ですが、C0CFGL/C1CFGL レジスタの BRP=1（2 分周）として、8Tq 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 8Tq 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	TSEG1	TSEG2	SJW	サンプルポイント
RL78/F15	8MHz	fCLK/2 (16MHz)	2(8MHz) (*1)	125ns	5	2	1	75%

(*1)500kbps 設定時は 4, 250kbps 設定時は 8, 125kbps 設定時は 16 に設定しています

3.2. 初期化の処理

初期化の処理としては、

・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
グローバルリセットモード遷移
CAN クロックの供給

・ch 毎の初期化処理
can*n*_init()　（n=0, 1)

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）(*1)
送受信 FIFO 設定（必要に応じて）(*1)
送信割込み設定（必要に応じて）(*2)

(*1)can.h 内の設定により、受信（受信 FIFO，送受信 FIFO，受信バッファ）、送信（送受信 FIFO，送信バッファ）の使用区分が変わります（使用するブロックを有効化します）
(*2)r_cg_userdefine.h 内の割り込み使用設定によって有効化
※CAN1 のみ有効化したい場合は、can1_init()のみ実行

・受信ルール設定
reveive_rule_conf()

受信ルール数の定義

CAN0/CAN1 の両方を有効化すると、CAN0/CAN1 それぞれ 20 ルールを割り振ります。どちらか一方の ch を有効化すると、有効化した方に 40 ルールを割り振ります。（CAN-ch の有効化は、board.h 内で定義）
[RL78/F14,F13: CAN0 に 16 ルール定義]

・受信バッファ設定
receve_buf_conf()

受信バッファ数の定義

受信バッファは最大 32。受信バッファ数と FIFO のメッセージバッファ数の合計が 40 以下。という条件内で設定する必要があります。FIFO の使用有無に応じて、残っているメッセージバッファを受信バッファに割り振ります。
[RL78/F14,F13: 受信バッファ最大 16、受信バッファと FIFO バッファの合計で 16 以下]

CAN0/CAN1 両方有効で、受信 FIFO（4 段）、送受信 FIFO（4 段）を使用する場合、受信バッファ数は 24 となります。(40-8x2 = 24)

・受信ルール設定
can*n*_receive_rule_set　　（n=0, 1）

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-CAN_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、CAN0/CAN1 両方使用時、ch 毎に 20 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 側で、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。

※CAN0/CAN1 有効時、ルール番号は 0-19 が有効です
※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）
[RL78/F14,F13: can0_receive_rule_set()のみ、ルール番号は 0-15 が有効です]

・CAN 動作

can_operate()

CAN モジュールを動作モードに移行させます。

・CAN 動作（ch 毎）

can*n*_operate()　（n=0, 1)

CAN0/CAN1（指定のチャネル）を動作モードに移行させます。[RL78/F14,F13: can0_operate()のみ]

・割り込みの設定

intr_setup()

割り込み優先度の設定（優先度 0[最高優先度]に設定）
割り込みフラグクリア
割り込みマスク解除

CAN の割り込みを使用する場合は、本関数を呼び出してください。

以上が、初期化の部分となります。各関数を呼び出す順番は上記の通りとしてください。ch 毎の処理は、CAN0/CAN1 どちらを先に実行しても構いません。

4. サンプルプログラムの説明(SAMPLE1)

4.1. プログラム仕様

・データフレームの送信
・データフレームの受信

を行うサンプルプログラムとします。

・CAN ID は拡張フォーマット(29bit)とする（標準フォーマットのデータも受信する）(*1)
・送信に使用する ID は 0x0000000~0x0000003 までの 4 種
・受信する ID は、0x0000000~0x0000003 までの 4 種（それ以外の ID のデータは受信しない）
・送信データは"s"コマンドで拡張フォーマットと標準フォーマットの切り替えが可能
・複数の CAN ch を持っているボード(*2)は、全ポート受信を行い、送信は"c"コマンドで ch の変更を行う
・データフレームのみ受信（リモートフレームは受信しない）
・端末のキーボード入力を読み取り 0~3 の入力に応じて ID，送信データバイト数を変えて送信する
・プッシュスイッチが付いているボード(*3)では、プッシュスイッチを押すと 1 バイト送信する（キーボードの 0 と等価）
・LED が付いているボードはデータを受信する度に LED の点灯・消灯が切り替わる

－キーボードから入力したキーと送信データの関係－

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

(*1)can_operation.h の定義で、標準フォーマットのみ取り扱う様に変更可能
(*2)HSBRL78F15-100 は、CAN0/CAN1 の 2ch の CAN ポートを持っています
(*3)SW2 がデータ送信用スイッチです

4.2. 受信ルールの設定

・CAN-ch0

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

・CAN-ch1

受信ルール番号 (*1)	フォーマット	ID	受信バッファ番号 (*2)
0 (20)	標準(SID)	0x000	16
1 (21)	標準(SID)	0x001	17
2 (22)	標準(SID)	0x002	18
3 (23)	標準(SID)	0x003	19
4 (24)	拡張(EID)	0x0000000	20
5 (25)	拡張(EID)	0x0000001	21
6 (26)	拡張(EID)	0x0000002	22
7 (27)	拡張(EID)	0x0000003	23

(*1)関数(can1_receive_rule_set)の引数としては 0-7 を指定します、内部的にはオフセット 20 が付いたルール番号となります

(*2)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
（CAN0 側は受信バッファ番号=受信ルール番号、CAN1 側は受信バッファ番号＝受信ルール番号+16 が設定されます）

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、CAN-ch 共有で最大 32 個設定可能で、本サンプルプログラムでは 32 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(*2)。

※RS-CAN_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 FIFO0 と送受信 FIFO0 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

[参考]CAN-ch0 を使用しない様に設定した場合は、CAN-ch1 の受信ルール番号、受信バッファ番号は 0~の割付となります。

[RL78/F14,F13: CAN0 のみ、受信ルール番号 0~15]

4.3. 送信バッファの設定

送信バッファは、ch 毎に 4 つあり CAN0 側が送信バッファ 0~3、CAN1 側が 4~7 を使用可能です。

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	送信 ch	キーボードからのコマンド
0	CAN0	0
1	CAN0	1
2	CAN0	2
3	CAN0	3
4	CAN1	0
5	CAN1	1
6	CAN1	2
7	CAN1	3

SAMPLE1 では、キーボードの"0"（1 バイト送信、ID=0）は、CAN0 側は送信バッファ 0、CAN1 側は送信バッファ 4 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。

[RL78/F14,F13: CAN0 のみ、送信バッファ 0~3 を使用]

4.4. 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。

(2)受信ルールの設定

3.2 章で示す受信ルールの設定です。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
can n_receive_rule_set(0, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000);
can n_receive_rule_set(1, CAN_RULE_BUF, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000001);
…
can n_receive_rule_set(4, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000000);
…
can n_receive_rule_set(7, CAN_RULE_BUF, CAN_ID_FORMAT_EID, CAN_DATA_FRAME, 0x00000003);
```

n=0 or 1（CAN-ch に応じた値）[RL78/F14,F13: can0_receive_rule_set()のみ]

(2)データの受信

・CAN0 側

```
for(i=0; i<=RMNB; i++)  //RMNB=32,  受信バッファ数  [RL78/F14,F13:  ループ数は 16]
{
  //引数:　　受信バッファ番号　フォーマット区分　データフレーム／リモートフレーム区分　ID　データ　タイムスタンプ
  r_ret = can_buf_receive((unsigned char)i, &r_ide, &r_rtr, &r_id, &r_data[0], &r_ts);
```

受信バッファ番号：　can*n*_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）
フォーマット区分：　受信した標準フォーマット(CAN_ID_FORMAT_SID)，拡張フォーマット
　(CAN_ID_FORMAT_EID)
データフレーム／リモートフレーム区分：　受信したデータフレーム(CAN_DATA_FRAME)，リモートフレーム
　(CAN_REMOTE_FRAME)
ID：　受信した ID 値
データ：　受信データ
タイムスタンプ：　タイムスタンプ値（受信側で付与）

r_ret　が 0 ならば、受信したデータはなし。1~8 であれば、戻り値に対応するバイト数のデータを受信しています。

※受信バッファに格納されているデータが、CAN0 で受信したものか CAN1 で受信したものかは受信バッファを見ても判断が付きません（受信ルールで CAN0/CAN1 と受信バッファ番号を紐付けしている）
　ここでは、受信バッファ番号の前半(0-15)が CAN0、後半(16-31)が CAN1 として受信データを表示しています。（受信ルール設定時、その様に設定しているため）

(4)データの送信

```
unsigned char s_data[8]={0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF}; //送信データ

//引数:　　バッファ番号　フォーマット区分　データフレーム／リモートフレーム区分　ID　送信バイト数　データ
s_ret = can0_buf_send(0, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x0000, 1, &s_data[0]);
```

バッファ番号：送信に使用する送信バッファ番号
フォーマット区分：標準フォーマット(CAN_ID_FORMAT_SID)，拡張フォーマット(CAN_ID_FORMAT_EID)
データフレーム／リモートフレーム区分：データフレーム(CAN_DATA_FRAME)，リモートフレーム(CAN_REMOTE_FRAME)
送信バイト数：1~8
データ：送信データ

　CAN-ch0 で、送信バッファ 0 から ID=0x000、データ 0x01 を 1 バイト送信する。

※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。）
※can1_buf_send の場合は、送信バッファ番号 0~3 を指定した場合、実際には送信バッファ 4~7 で送信されます
[RL78/F14,F13: can0_buf_send()のみ]

4.5. データの受信

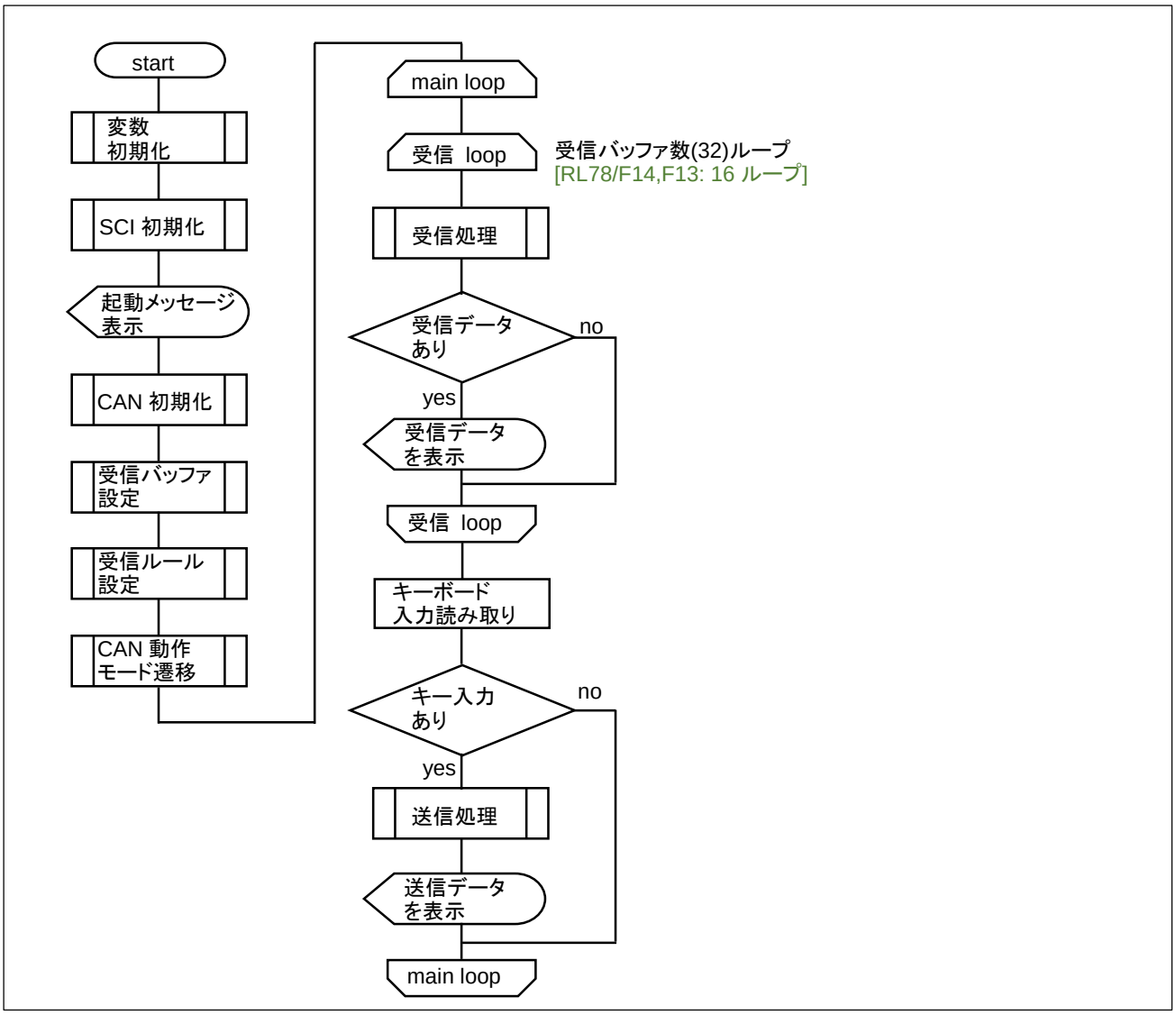
　SAMPLE1 でのデータの受信に関しては、受信メッセージバッファまでのデータ格納に関しては、（初期化、受信ルール設定、受信バッファ設定が済んでいれば）マイコンのハードウェアが行います。受信バッファにデータが格納されているかは、プログラムで受信関数(can_buf_receive)を呼び出す事で確認を行っています。そのため、常に受信関数を呼び出して確認を行わないと、データの取りこぼしが生じる可能性があります。受信データの確認に CPU リソースを食うため、スムーズな手法とはいえないと考えます。

（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。）

4.6. SAMPLE1 フローチャート

－処理フロー－

メイン関数　main_s1()



5. サンプルプログラムの説明(SAMPLE2)

5.1. プログラム仕様

・データフレームの送信
・データフレームの受信

を行うサンプルプログラムとします。

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。

5.2. 使用する割り込み

・受信
受信 FIFO　※デフォルト
送受信 FIFO　選択可
受信バッファ　選択可であるが、割り込みによる受信処理不可

・送信
送受信 FIFO　※デフォルト
送信バッファ　選択可

割り込みで受信の処理を行う場合は、「受信 FIFO」または「送受信 FIFO」のどちらかを選択します。

・受信 FIFO 割り込みは CAN0/CAN1 で分かれていない（どちらの ch で受信しても同じ割込み）
・送受信 FIFO の受信割り込みは CAN0/CAN1 で別な割り込みとなるが、1 本/ch 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。

・送受信 FIFO の送信と送信バッファによる送信は同じ割り込み（CAN0/CAN1 は別割り込み）

送受信系の割り込みは、全部で 5 本となります。
[RL78/F14,F13: 送受信系の割り込みは全部で 3 本]

・CAN 使用割り込み一覧

割り込み 要求元	割り込み要因	割り込み条件	呼び出される 割り込み関数名
CAN	INTCANGRFR	受信 FIFO	intcangrfr_interrupt(*1)
CAN0	INTCAN0CFR	送受信 FIFO 受信	intcan0cfr_interrupt(*2)
CAN0	INTCAN0TRM	チャネル送信	intcan0trm_interrupt(*3)
CAN1	INTCAN1CFR	送受信 FIFO 受信	intcan1cfr_interrupt(*4)
CAN1	INTCAN1TRM	チャネル送信	intcan1trm_interrupt(*5)

割り込みの関数は、
RL78_F15_CAN_S2¥can¥can_intr_s2.c
内に含まれています。

受信に受信 FIFO を使った場合（デフォルト）、CAN0/CAN1 どちらの受信時でも(*1)の割り込み関数が実行されます。

受信に送受信 FIFO を使った場合、CAN0 の受信時は(*2)が、CAN1 の受信時は(*4)の割り込み関数が実行されます。

送信の割り込みは、送受信 FIFO を使った場合、送信バッファを使った場合共に、CAN0 の送信時は(*3)、CAN1 の送信時は(*5)の割り込み関数が実行されます。[RL78/F14,F13: (*1)~(*3)]

5.3. 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(3)の 4 本あり、CAN0/CAN1 の物理 ch と FIFO の関連付けは自由です。

本サンプルプログラムでは、CAN0 で受信したデータは RXFIFO(0)に、CAN1 で受信したデータは RXFIFO(1)に格納するようにしています。

※RS-CAN_Lite モジュールの構成上、受信ルールで FIFO の格納先を設定するので、ID=0x123 のデータは CAN0/CAN1 どちらで受信した場合でも、RXFIFO(0)に格納。ID=0x124 のデータは、RXFIFO(1)に格納するといった使い方も可能です

送受信 FIFO は SRFIFO(0)~SRFIFO(1)の 2 本で、CAN0 と SRFIFO(0)、CAN1 と SRFIFO(1)が対応しています。

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 番と 4 番が送受信 FIFO との紐付けとなり、使用不可となります。送信バッファの 1~3, 5~7 は送信バッファとして、使用可能です。）[RL78/F14,F13: RXFIFO(0~1)の 2 本, SRFIFO(0)の 1 本]

・使用 FIFO

FIFO	用途	FIFO 段数
RXFIFO(0)	CAN0 受信	4
RXFIFO(1)	CAN1 受信	4
RXFIFO(2)	未使用	0
RXFIFO(3)	未使用	0
SRFIFO(0)	CAN0 送信	4
SRFIFO(1)	CAN1 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 32 段まで設定可能です。但し、全体でメッセージバッファは 40 となりますので、上記 6 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 40 以下にする必要があります。

[RL78/F14,F13: FIFO 段数は 16 段まで、全体でメッセージバッファは 16]

SAMPLE2 では、FIFO で 16 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 24 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 40 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。

[参考]

本サンプルプログラムでは、送受信 FIFO の SRFIFO(0)は CAN0 の送受信、SRFIFO(1)は CAN1 の送受信に使用しています。RL78/F15 マイコンの実動作を見ると、CAN0 の受信を SRFIFO(1)で行う事も可能な様です（受信動作においては、物理 ch と SRFIFO の 0/1 は必ずしも対応していない）。但し、送信に関しては、SRFIFO の 0/1 と物理 ch が対応しており、ハードウェアマニュアルの記載上も、SRFIFO の 0/1 と物理 ch が 1:1 対応の様に見受けられますので、サンプルプログラムや本マニュアルでは、SRFIFO(0)=CAN0, SRFIFO(1)=CAN1 の 1:1 対応としています。

5.4. 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。

・CAN-ch0

受信ルール番号	フォーマット	ID	受信 FIFO 使用時	送受信 FIFO 使用時	受信バッファ 使用時(*1)
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

・CAN-ch1

受信ルール番号	フォーマット	ID	受信 FIFO 使用時	送受信 FIFO 使用時	受信バッファ 使用時(*1)
0 (20)	標準(SID)	0x000	RXFIFO(1)	SRFIFO(1)	16
1 (21)	標準(SID)	0x001	RXFIFO(1)	SRFIFO(1)	17
2 (22)	標準(SID)	0x002	RXFIFO(1)	SRFIFO(1)	18
3 (23)	標準(SID)	0x003	RXFIFO(1)	SRFIFO(1)	19
4 (24)	拡張(EID)	0x0000000	RXFIFO(1)	SRFIFO(1)	20
5 (25)	拡張(EID)	0x0000001	RXFIFO(1)	SRFIFO(1)	21
6 (26)	拡張(EID)	0x0000002	RXFIFO(1)	SRFIFO(1)	22
7 (27)	拡張(EID)	0x0000003	RXFIFO(1)	SRFIFO(1)	23

(*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、割り込みでの受信はできません

5.5. 送信バッファの設定

・送信バッファ

送信バッファ 番号	送信 ch	送受信 FIFO 使用時	送信バッファ 使用時(*1)
0	CAN0	SRFIFO(0)	ｺﾏﾝﾄﾞ"0"に対応
1	CAN0	未使用	ｺﾏﾝﾄﾞ"1"に対応
2	CAN0	未使用	ｺﾏﾝﾄﾞ"2"に対応
3	CAN0	未使用	ｺﾏﾝﾄﾞ"3"に対応
4	CAN1	SRFIFO(1)	ｺﾏﾝﾄﾞ"0"に対応
5	CAN1	未使用	ｺﾏﾝﾄﾞ"1"に対応
6	CAN1	未使用	ｺﾏﾝﾄﾞ"2"に対応
7	CAN1	未使用	ｺﾏﾝﾄﾞ"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（CAN0 の送信に使用）、4 番（CAN1 の送信に使用）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
※但し、割り込み処理は追加されています

5.6. 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); //CAN-ch0

can1_receive_rule_set(0, **CAN_RULE_RXFIFO1**, CAN_ID_FORMAT_SID, CAN_DATA_FRAME, 0x00000000); //CAN-ch1

…

受信ルール設定時、受信バッファではなく CAN-ch0：受信 FIFO(0), CAN-ch1：受信 FIFO(1)を使うように設定。（受信割り込み機能が、FIFO と関連しているためです）

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。
[RL78/F14,F13: can0_receive_rule_set()のみ]

(2)データの送信

//引数:　　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　送信バイト数　データ

s_ret = **can*n*_srfifo_send**(s_format, CAN_DATA_FRAME, s_id, s_dlc, &s_data[0]);

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。
[RL78/F14,F13: n=0 のみ]

(i1)受信割り込み関数[SAMPLE1 との相違点]

(a)受信 FIFO を使う場合（デフォルト）

intcangrfr_interrupt()

```
while(1)
{
   //RXFIFO(0)の受信処理（CAN-ch0 側）
   while((RFSTS0 & 0x0001) == 0)   //FIFO 内に未読メッセージあり
   {
      r_ret = can_rxfifo_receive(0, &ide, &rtr, &id, &data[0], &ts);      //受信 FIFO を使った RXFIFO(0)受信
      [受信データの表示]
   }
   RFSTS0 &= ~0x0008;//b3:RFIF,割り込みフラグクリア

   //RXFIFO(1)の受信処理（CAN-ch1 側）
   while((RFSTS1 & 0x0001) == 0)   //FIFO 内に未読メッセージあり
   {
      r_ret = can_rxfifo_receive(1, &ide, &rtr, &id, &data[0], &ts);      //受信 FIFO を使った RXFIFO(1)受信
      [受信データの表示]
   }
   RFSTS1 &= ~0x0008;//b3:RFIF,割り込みフラグクリア
…（RXFIFO(2), RXFIFO(3)の受信処理）

   if ( (RFSTS0 | RFSTS1 | RFSTS2| RFSTS3) & 0x0008) == 0) break;   //受信割り込みルーチンを抜ける(*1)
}
```

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。

(*1)受信 FIFO の受信割り込みは、RXFIFO(0)~RXFIFO(3)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。

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、(*1)の処理がない場合
（RXFIFO(0)~RXFIFO(3)の一連の受信処理の後、割り込み関数を抜ける）どうなるかを考えてみます。

・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				INTCANGRFR		
	(0)	(1)	(2)	(3)			
1	○	-	-	-	↑	RXFIFO(0)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
3	○	○	-	-		RXFIFO(1)でデータ受信	
4	-	○	-	-			RXFIFO(0)読み出し完了、フラグを落とす
5	-	○	-	-			RXFIFO(1)読み出し処理
6	-	-	-	-			RXFIFO(1)読み出し完了、フラグを落とす
7	-	-	-	-			割り込み関数を抜ける
8	○	-	-	-	↑	RXFIFO(0)でデータ受信	→割り込み関数に飛んでくる
9	○	-	-	-			RXFIFO(0)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				INTCANGRFR		
	(0)	(1)	(2)	(3)			
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(1)読み出し処理
4	○	○	-	-		RXFIFO(0)でデータ受信	
5	○	-	-	-			RXFIFO(1)読み出し完了、フラグを落とす
6	○	-	-	-			割り込み関数を抜ける(*2)
7	○						(*2)
8	○	-	-	-		RXFIFO(0)でデータ受信	(*3)
9	○	○	-	-		RXFIFO(1)でデータ受信	(*3)

上記のケースは 2 つ問題があります。
(*2)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
(*3)以降データを受信した場合でも割り込みは発生しない

INTCANGRFR の割り込みが成立する条件が、「RXFIFO(0)~(3)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)~(3)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、前頁の(*1)の処理がない場合、RXFIFO(0)~(3)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。

これを回避する方法として、ウォッチドッグ処理の様に一定時間以上 RXFIFO の割り込みフラグが立ちっぱなしになっている場合はクリアするという方法もありますが、本サンプルプログラムでは割り込み関数を抜ける条件として、RXFIFO(0)~(3)が全てクリアされているという事としています。割り込み関数の最後でフラグがクリアされていなければ、最初から受信処理を再度実行します。

なお、(*1)の処理を入れた場合、ですが

・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				INTCANGRFR		
	(0)	(1)	(2)	(3)			
1	○	-	-	-	↑	RXFIFO(0)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
3	-	-	-	-			RXFIFO(0)読み出し完了、フラグを落とす
4	-	-	-	-			RXFIFO(1)~(3)の処理
5	-	-	-	-			(*1)4 つのフラグを確認する処理
6	-	○	-	-	↑	RXFIFO(1)でデータ受信	
7	-	○	-	-			割り込み関数を抜ける
8	-	○	-	-			→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。7 番のところで一度割り込み関数を抜けますが、INTCANGRFR の割り込みは立つので、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)~(3)の 4 つのフラグが全部落ちているタイミングがあるかどうかです

（どのタイミングでデータを受信しても問題ない様に考えているつもりですが、抜けがあるかもしれません。CAN の受信割り込みの処理は、割り込み成立条件に多少注意が必要ですので留意願います。）

(b)送受信 FIFO を使う場合

intcan0cfr_interrupt()

```
//CAN-ch0 側
while((CFSTS0 & 0x0001) == 0)
{
   r_ret = can0_srfifo_receive(&ide, &rtr, &id, &data[0], &ts);       //送受信 FIFO を使った受信
   [受信データの表示]
}
CFSTS0 &= ~0x0008;//b3:CFRXIF,割り込みフラグクリア
```

intcan1cfr_interrupt()

```
//CAN-ch1 側
while((CFSTS1 & 0x0001) == 0)
{
   r_ret = can1_srfifo_receive(&ide, &rtr, &id, &data[0], &ts);       //送受信 FIFO を使った受信
   [受信データの表示]
}
CFSTS1 &= ~0x0008;//b3:CFRXIF,割り込みフラグクリア
```

共通 FIFO を使う場合も、FIFO が空になるまでデータの読み出しを行います（受信 FIFO を使う場合と同様です）。

CAN-ch0 と CAN-ch1 の送受信 FIFO を使った受信割り込みは、別関数となっています。（受信 FIFO を使った場合は、CAN-ch0, CAN-ch1 どちらの受信でも同じ割り込み関数に飛んできます。）

[RL78/F14,F13: intcan0cfr_interrupt()のみ]

(i2)送信割り込み関数[SAMPLE1 との相違点]

(a)送受信 FIFO を使う場合
(b)送信バッファを使う場合

※割り込み関数(「送受信 FIFO 送信」と「送信バッファ」で共通)

intcan0trm_interrupt()　//CAN0 側

```
//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
{
   [送信済みの表示]  →"CAN0 send finished.(SRFIFO)"
}
CFSTS0 &= ~0x0010;   //b4:CFTXIF,  割り込みフラグクリア

//送信バッファ割り込み
n=0~3 で 4 回ループ
{
   switch(TMSTSn) //n=0~3
   {
      case 0x4:
         [送信済みの表示]  →"CAN0 send finished.(buf=n, status=OK)"
         break;
      case 0x6;
          [送信済みの表示]  →"CAN0 send finished.(buf=n, status=with abort)"
         break;
       case 0x2;
          [送信済みの表示]  →"CAN0 send finished.(buf=n, , status=aborted)"
         break;
   }
   TMSTSn &= 0x6;   //TMTRF クリア
}
```

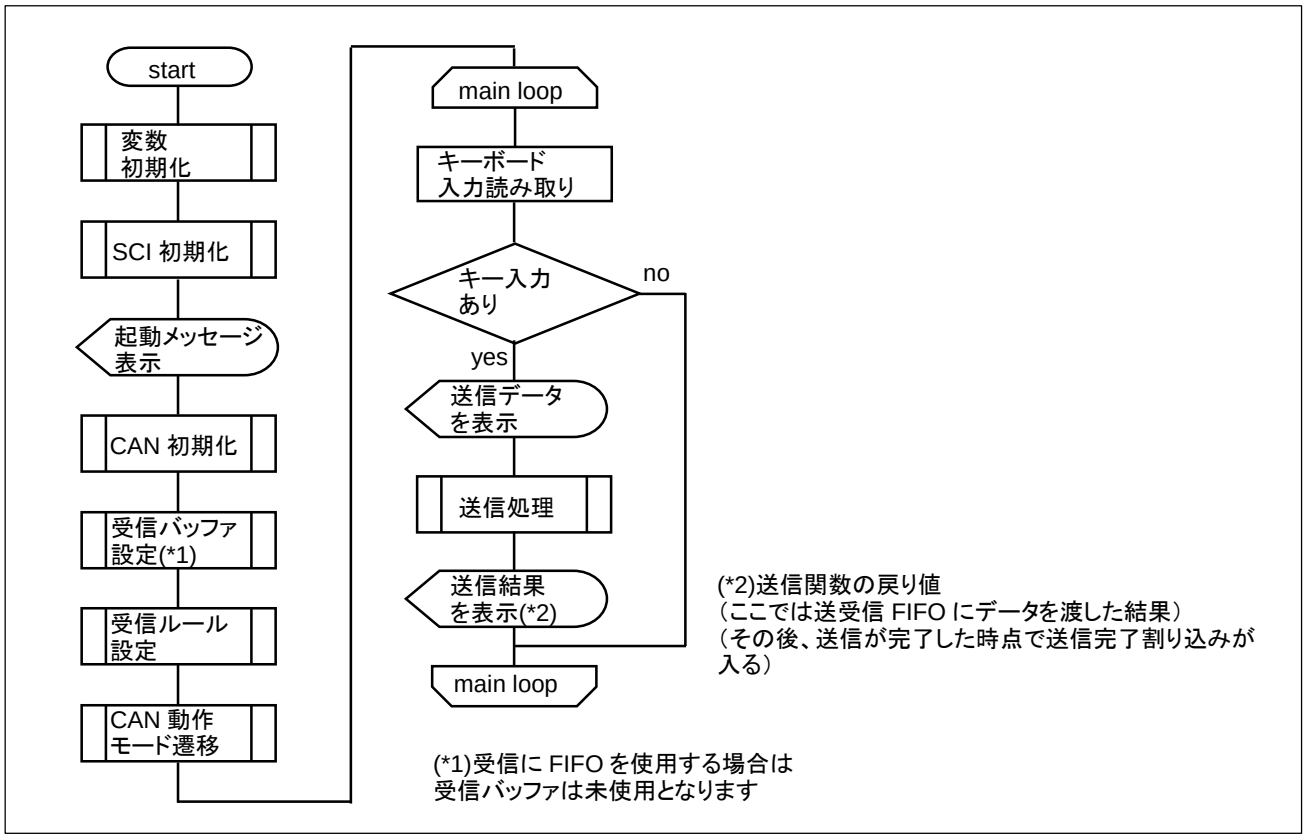
送信の割り込みは、確認用のメッセージの表示とフラグクリアのみです。送信バッファの割り込み処理は送信結果も表示します。

intcan1trm_interrupt()　//CAN1 側(処理内容は CAN0 側と同じです)
[RL78/F14,F13: intcan0trm_interrupt()のみ]

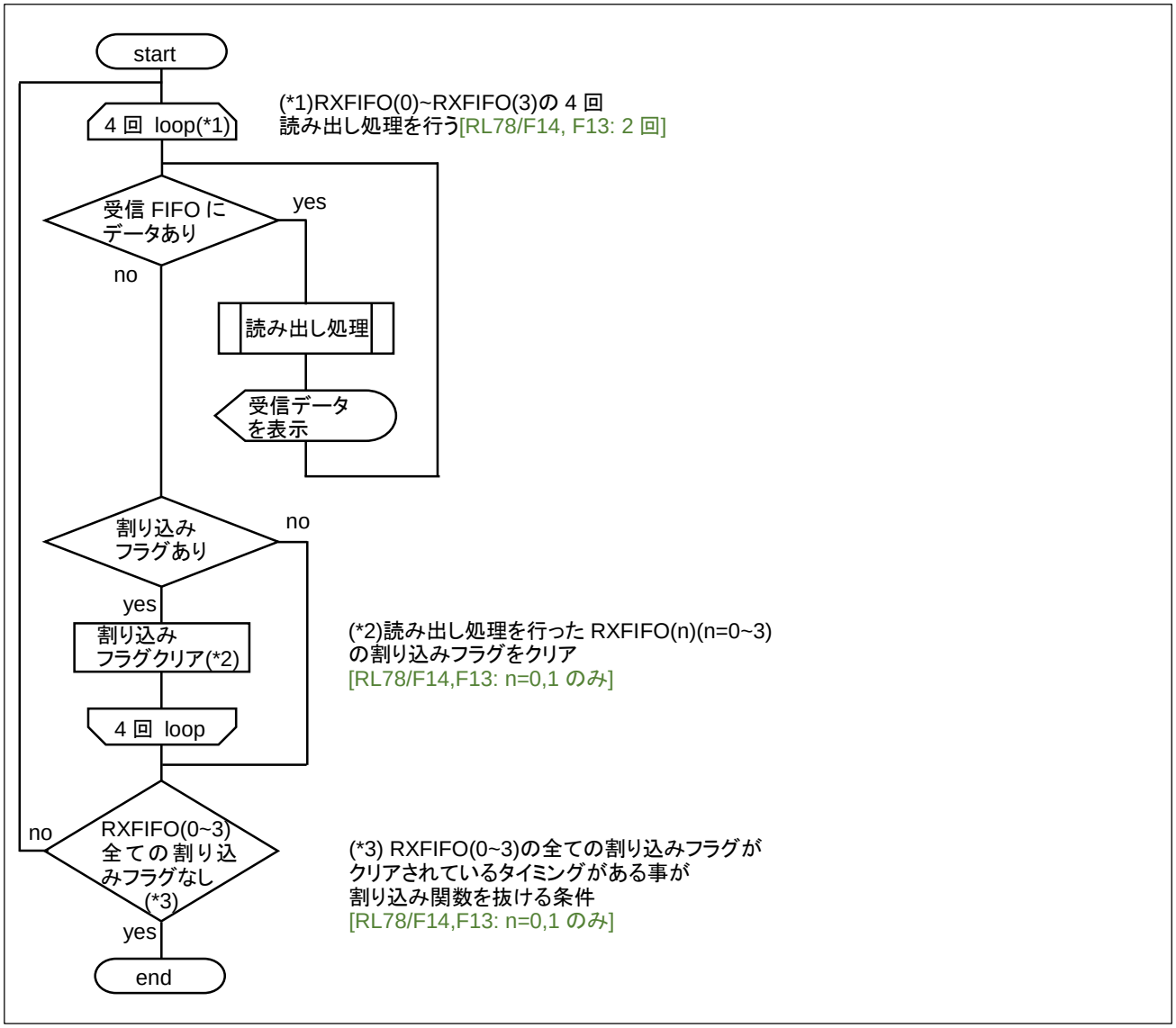
5.7. SAMPLE2 フローチャート

－処理フロー－

メイン関数　main_s2()



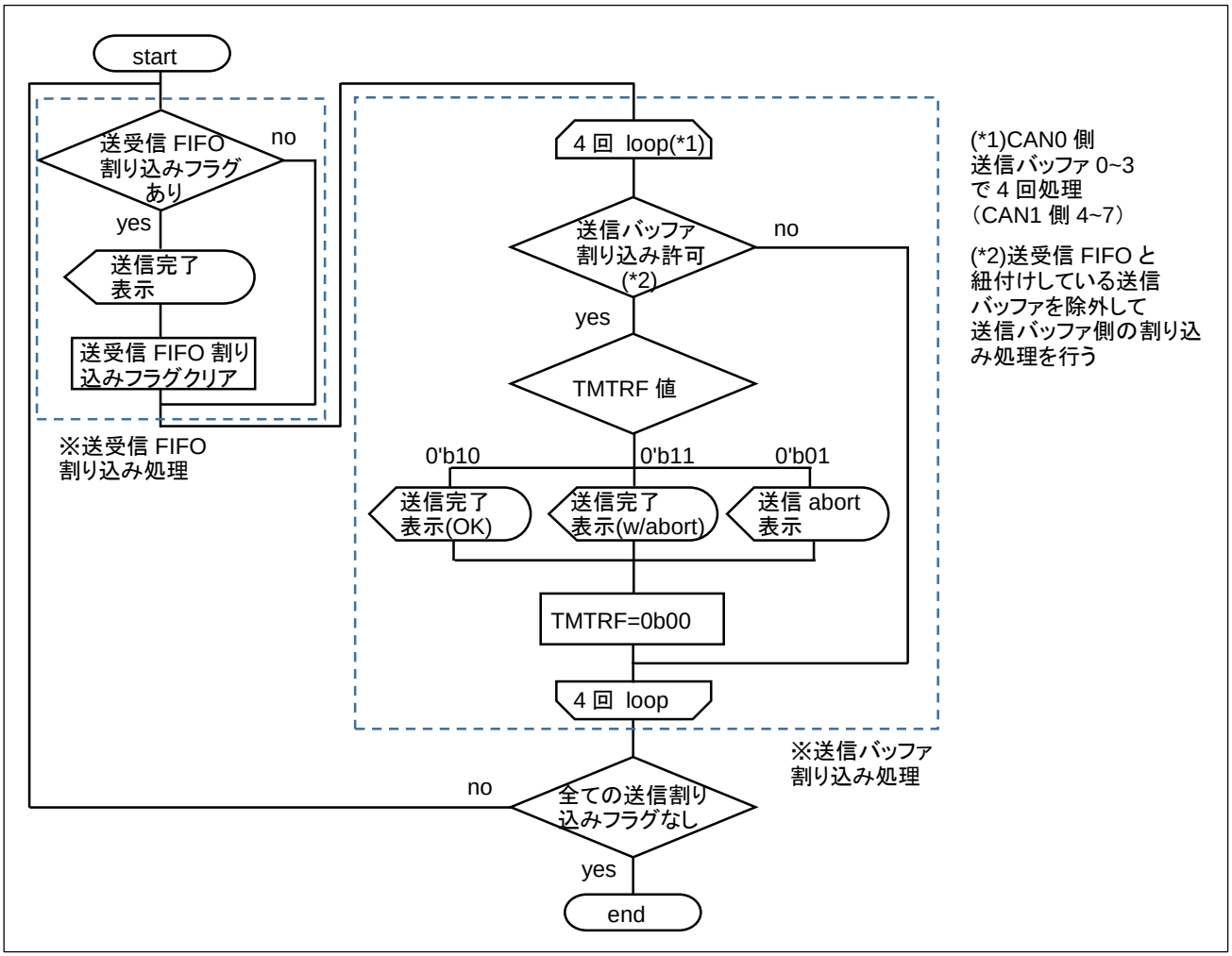
受信割り込み関数（受信 FIFO） intcangrfr_interrupt()



受信割り込み関数（送受信 FIFO） intcan0cfr_interrupt(), intcan1cfr_interrupt()

SRFIFO(0)と SRFIFO(1)は別な割り込み関数となります。

送信完了割り込み関数 intcan0trm_interrupt(), intcan1trm_interrupt()



6. サンプルプログラムの説明(SAMPLE3)

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
・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。

6.2. 受信ルール設定

・CAN-ch0

受信ルール番号	フォーマット	データフレーム／リモートフレーム	ID	受信 FIFO 使用時	送受信 FIFO 使用時	受信バッファ使用時(*1)
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)	4
5	標準(SID)	リモートフレーム	0x001	RXFIFO(0)	SRFIFO(0)	5
6	標準(SID)	リモートフレーム	0x002	RXFIFO(0)	SRFIFO(0)	6
7	標準(SID)	リモートフレーム	0x003	RXFIFO(0)	SRFIFO(0)	7
8	拡張(EID)	データフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	8
9	拡張(EID)	データフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	9
10	拡張(EID)	データフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	10
11	拡張(EID)	データフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	11
12	拡張(EID)	リモートフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	12
13	拡張(EID)	リモートフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	13
14	拡張(EID)	リモートフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	14
15	拡張(EID)	リモートフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	15

・CAN-ch1

受信ルール番号	フォーマット	データフレーム／リモートフレーム	ID	受信 FIFO 使用時	送受信 FIFO 使用時	受信バッファ使用時(*1)
0 (21)	標準(SID)	データフレーム	0x000	RXFIFO(0)	SRFIFO(0)	16
1 (22)	標準(SID)	データフレーム	0x001	RXFIFO(0)	SRFIFO(0)	17
2 (23)	標準(SID)	データフレーム	0x002	RXFIFO(0)	SRFIFO(0)	18
3 (24)	標準(SID)	データフレーム	0x003	RXFIFO(0)	SRFIFO(0)	19
4 (25)	標準(SID)	リモートフレーム	0x000	RXFIFO(0)	SRFIFO(0)	20
5 (26)	標準(SID)	リモートフレーム	0x001	RXFIFO(0)	SRFIFO(0)	21
6 (27)	標準(SID)	リモートフレーム	0x002	RXFIFO(0)	SRFIFO(0)	22
7 (28)	標準(SID)	リモートフレーム	0x003	RXFIFO(0)	SRFIFO(0)	23
8 (29)	拡張(EID)	データフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	24
9 (30)	拡張(EID)	データフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	25
10 (31)	拡張(EID)	データフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	26
11 (32)	拡張(EID)	データフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	27
12 (33)	拡張(EID)	リモートフレーム	0x0000000	RXFIFO(0)	SRFIFO(0)	28
13 (34)	拡張(EID)	リモートフレーム	0x0000001	RXFIFO(0)	SRFIFO(0)	29
14 (35)	拡張(EID)	リモートフレーム	0x0000002	RXFIFO(0)	SRFIFO(0)	30
15 (36)	拡張(EID)	リモートフレーム	0x0000003	RXFIFO(0)	SRFIFO(0)	31

(*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、割り込みでの受信はできません
　受信バッファを使った場合リモートフレームの受信・画面表示は行いますが、データフレーム応答は返しません

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。）

6.3. 送信バッファの設定

・送信バッファ

送信バッファ番号	送信 ch	送受信 FIFO 使用時	送信バッファ使用時
0	CAN0	SRFIFO(0)	コマンド"0", "q"に対応
1	CAN0	未使用	コマンド"1", "w"に対応
2	CAN0	未使用	コマンド"2", "e"に対応
3	CAN0	未使用	コマンド"3", "r"に対応
4	CAN1	SRFIFO(1)	コマンド"0", "q"に対応
5	CAN1	未使用	コマンド"1", "w"に対応
6	CAN1	未使用	コマンド"2", "e"に対応
7	CAN1	未使用	コマンド"3", "r"に対応

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。SAMPLE2 と同一な点は説明を省略します。

(2)受信ルールの設定[SAMPLE2 との相違点]

//引数:　　受信ルール番号　バッファ／FIFO 区分　フォーマット区分　データフレーム／リモートフレーム区分　ID
…

can0_receive_rule_set(4, CAN_RULE_RXFIFO0, CAN_ID_FORMAT_SID, **CAN_REMOTE_FRAME**, 0x00000000); //CAN-ch0
…

can1_receive_rule_set(4, CAN_RULE_RXFIFO1, CAN_ID_FORMAT_SID, **CAN_REMOTE_FRAME**, 0x00000000); //CAN-ch0
…

受信ルール番号 4~7, 12~15 をリモートフレーム受信向けに追加。[RL78/F14,F13: can0_receive_rule_set()のみ]
[RL78/F14,F13: can0_receive_rule_set()のみ]

(3)データの送信

//引数:　　バッファ番号　フォーマット区分　データフレーム／リモートフレーム区分　ID　送信バイト数　データ
s_ret = can***n***_srfifo_send(s_format, **s_rtr**, s_id, s_dlc, &s_data[0]);

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_rtr=0, q~r の時は s_rtr=1)
[RL78/F14,F13: n=0 のみ]

(i1)受信割り込み関数[SAMPLE2 との相違点]

(a)受信 FIFO を使う場合（デフォルト）

intcangrfr_interrupt()

```
while(1)
{
   //RXFIFO(0)の受信処理（CAN-ch0 側）
   while((RFSTS0 & 0x0001) == 0)   //FIFO 内に未読メッセージあり
   {
      r_ret = can_rxfifo_receive(0, &ide, &rtr, &id, &data[0], &ts);      //受信 FIFO を使った RXFIFO(0)受信
      [受信データの表示]
      if(rtr == CAN_REMOTE_FRAME)
      {
         dlc = (unsigned char)(r_ret & 0xf); //受信した DLC の値を送信の DLC の値として使用
         [送信データの表示]
         ret2 = can0_srfifo_send(ide, CAN_DATA_FRAME, id, dlc, &remote_frame_response_data[0]);
      }

   }
   RFSTS0 &= ~0x0008;//b3:RFIF,割り込みフラグクリア
   …
}
```

受信したデータの RTR 値を見て、データフレーム(RTR=CAN_DATA_FRAME(0))の場合は、SAMPLE2 と同じ処理（画面表示）、リモートフレーム(RTR=CAN_REMOTE_FRAME(1))の場合は、データフレームの送信を行います。

上記は、受信に受信 FIFO を使った場合ですが、送受信 FIFO 使った場合でも同様の処理となります。

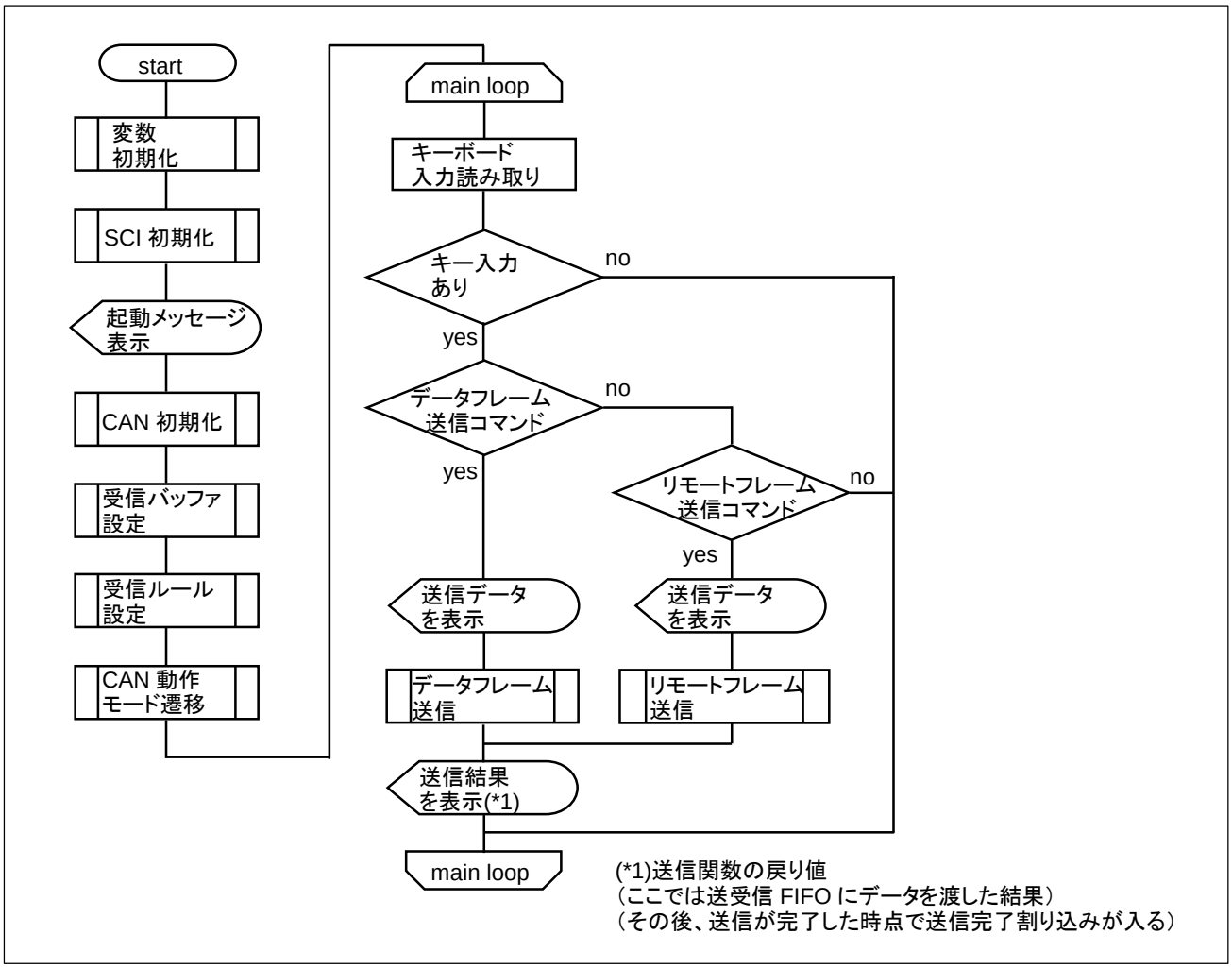
(i2)送信割り込み関数

送信割り込み関数の処理に関しては、SAMPLE2 と同様です。

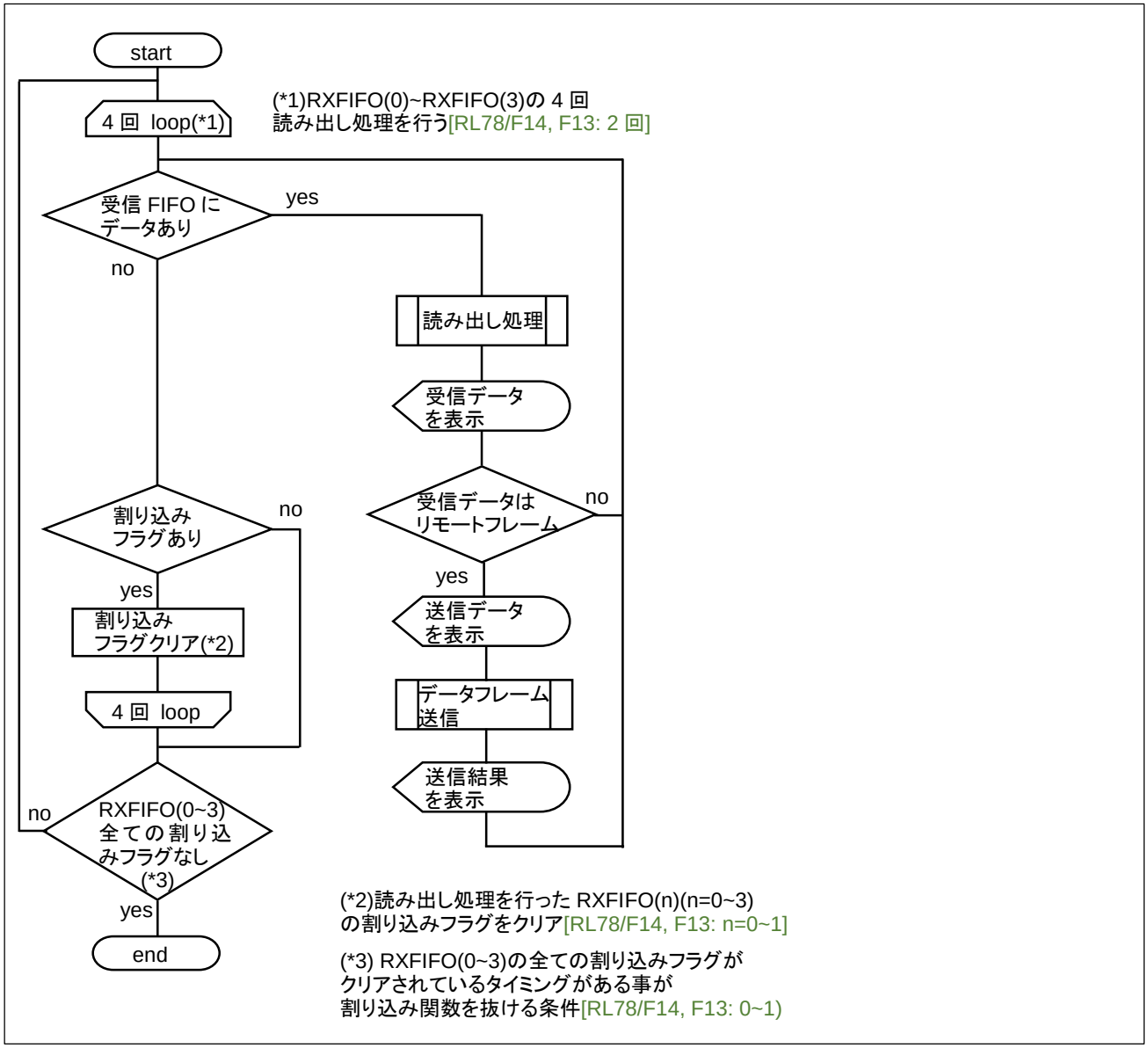
6.5. SAMPLE3 フローチャート

－処理フロー－

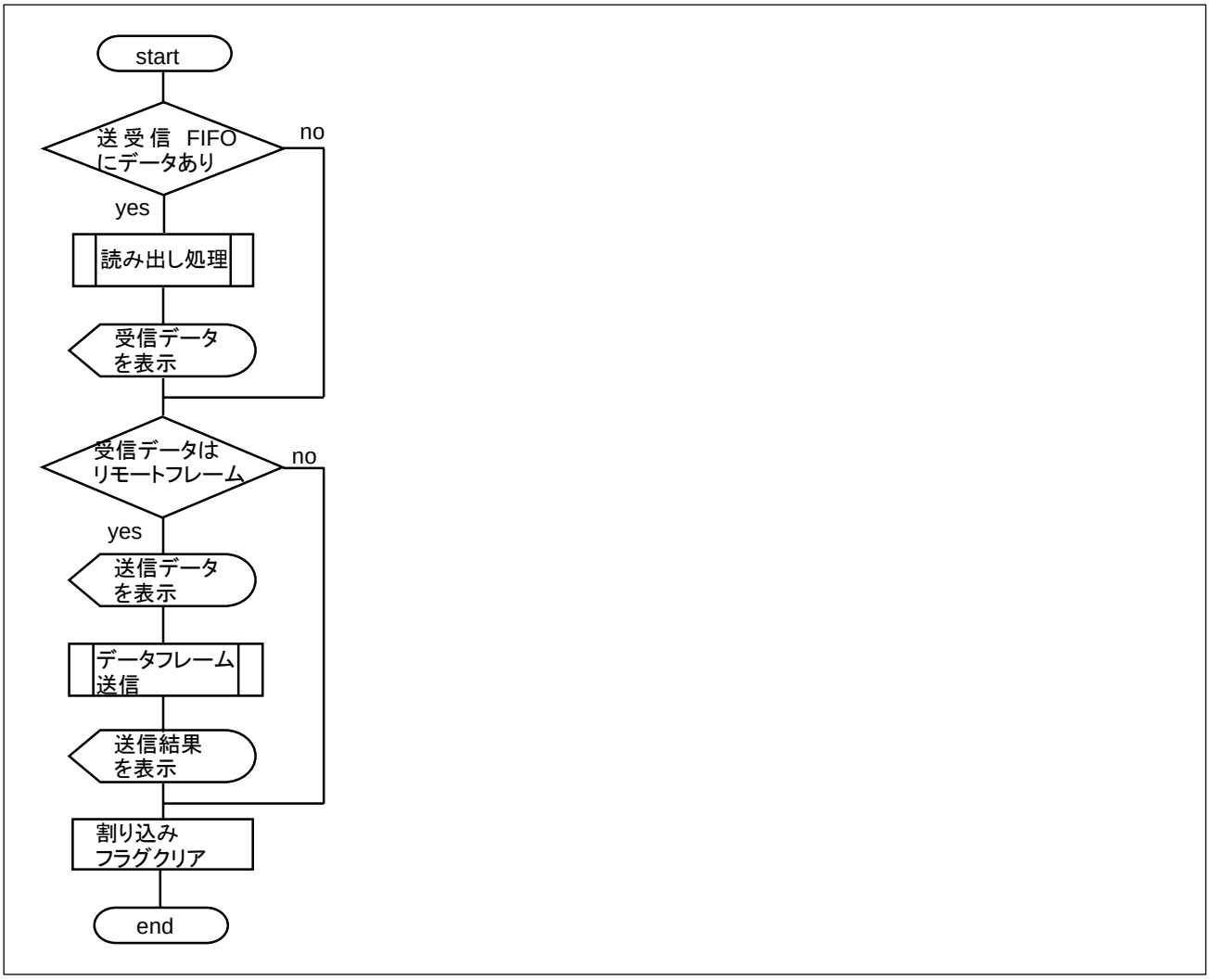
メイン関数　main_s3()



受信割り込み関数（受信 FIFO） intcangrfr_interrupt()



受信割り込み関数（送受信 FIFO） intcan0cfr_interrupt(), intcan1cfr_interrupt()



7. サンプルプログラムで使用している関数の説明

7.1. 関数仕様

can_reset

概要:初期化関数
宣言:
　　int can_reset(void)

説明:
　・モジュールストップ解除
　・クロック設定
　・CAN リセットモード遷移
を行います

引数:
　　なし
戻り値:
　　0: 正常終了

cann_init　　(n=0~1)

概要:チャネル初期化関数
宣言:
　　int can0_init(void)　[ch0 向け]
　　int can1_init(void)　[ch1 向け] [RL78/F14, F13: 当該関数なし]

説明:
　・チャネルリセットモード遷移
　・端子設定
　・通信速度設定
を行います

引数:
　　なし
戻り値:
　　0: 正常終了

初期化関数は、CAN を 1Mbps 設定(*1)で初期化します。

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

can_reset()の実行後、can*n*_init()を実行してください。

can_reset()は全体の処理、can*n*_init()は ch 毎の処理です。

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

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

引数:
なし
戻り値:
0: 正常終了

CAN0 と CAN1 の両方有効にした場合は、CAN0/CAN1 それぞれ 20 ルールを設定します(*1)。どちらか一方の ch のみ有効化した場合は、有効化した ch 側に 40 ルールを設定します。[RL78/F14, F13: CAN に 16 ルール]
(*1)マイコンの機能としては、トータル 40 という制約があるだけで、CAN0 側に 30, CAN1 側に 10 ルールといった設定も可能です

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

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

引数:

buf_num

戻り値:

0: 正常終了

-1: 引数エラー（32 以上のバッファを設定しようとしている）

メッセージバッファは、トータルで 40 あります。FIFO の段数と受信バッファ合計で 40 以下となる様に設定してください。（FIFO を使用しない場合でも、受信バッファ数は最大 32 です。）[RL78/F14, F13: 合計で 16 以下]

can*n*_receive_rule_set　　(n=0~1) [RL78/F14, F13: n=0 のみ]

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);

int can1_receive_rule_set(unsigned char num, unsigned char mode, unsigned char ide, unsigned char rtr, unsigned long id);

説明:

・受信ルール設定

を行います

引数:

num: 受信ルール番号

mode: 受信先

CAN_RULE_BUF(0x0)　　受信バッファで受信

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

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

CAN_RULE_RXFIFO2(0x0004)　　受信 FIFO(2)で受信 [RL78/F14, F13: 指定不可]

CAN_RULE_RXFIFO3(0x0008)　　受信 FIFO(3)で受信 [RL78/F14, F13: 指定不可]

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

CAN_RULE_SRFIFO1(0x0020)　　送受信 FIFO(1)で受信 [RL78/F14, F13: 指定不可]

ide: 標準／拡張フォーマット区分

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

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

rtr: データフレーム／リモートフレーム区分

CAN_DATA_FRAME(0)　　データフレーム（データ送信）

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

id: ID を指定

戻り値:

0: 正常終了

-1: 引数エラー

-2: 受信ルール設定不可の動作モード

補足:

num は、1ch 使用時 0-39, 2ch（CAN0 及び CAN1）使用時 0-19 を指定可能です[RL78/F14, F13: 0-15]

但し、mode=0（受信バッファを使用）とした場合は、1ch 使用時 receive_buf_conf()で指定した値まで

2ch 使用時 receive_buf_conf()で指定した値の半分まで指定可能です

mode=0（受信バッファを使用）時、

・1ch 使用時、または CAN0 側

受信ルール番号=num

受信バッファ番号=num

で設定します。

・2ch 使用時の CAN1 側

受信ルール番号=num+20

受信バッファ番号=num+受信バッファ数/2

で設定します。

※2ch 使用時、受信バッファ数 32 の場合、

使用可能な受信ルール　CAN0 側: 0~19, CAN1 側: 20~29

使用可能な受信バッファ　CAN0 側: 0~15, CAN1 側: 16~31

となり、

num=16~19（CAN0 側:受信ルール番号 16~19, CAN1 側:受信ルール番号 36~39）には、受信バッファの割り当ては出来ません。

（マイコンの機能としては、その様な使い方は可能です）

※can0_receive_rule_set(),can1_receive_rule_set()を使用した場合は、受信バッファの前半の半分は CAN0 向け、後半の半分は CAN1 向けに設定するという制約が入ります。

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 reveive_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

receive_buf_num: 受信バッファ番号(mode=0 指定時のみ有効)

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:

0: 正常終了

-1: 引数エラー

-2: 受信ルール設定不可の動作モード

CAN0 向けの can0_receive_rule_set(), CAN1 向けの can1_receive_rule_set()では、ルール番号と受信バッファ番号を計算した値に割り振る様にしていますが、本関数は引数として指定した値を加工せず、受信ルール番号、受信バッファ番号に割り付けます。num は 0~39 の値、receive_buf_num は、確保した受信バッファ数の範囲で自由に設定可能です。[RL78/F14, F13: num は 0~15]

※マイコンの機能としては、受信ルールに適合したメッセージを「受信 FIFO(RXFIFO(0))」と「受信バッファ(2)」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の関数では受信先は 1 箇所に制限しています。
(複数個所に受信したメッセージを格納したい場合は、関数の改変が必要です。)

※マイコンの機能としては、ID の部分一致や IDE=0/1 両方を 1 つのルールでマッチさせるという事も可能ですが、本サンプルプログラムでは、ID, IDE, RTR は完全一致としています。(DLC は比較対象から外しています。)(D, IDE, RTR の部分一致や DLC の完全一致のルールを生成する場合は、関数の改変が必要です。)

can_operate

can*n*_operate　　(n=0~1) [RL78/F14, F13: n=0 のみ]

概要:動作モード変更関数

宣言:

　　int can_operate(void)

　　int can0_operate(void)

　　int can1_operate(void)

説明:

　・CAN モジュールを動作モードに移行する処理

　・チャネ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:

　　0:　正常終了

　　-2:　動作モード変更 NG

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

intr_setup

概要:割り込み設定関数

宣言:

　　int intr_setup(void)

説明:

　・割り込みの設定

　・割り込みの有効化

を行います

引数:　なし

戻り値:

　　0:　正常終了

　本関数は、ch 毎の動作モード変更(can*n*_operate)後の実行を想定していますが、CAN モジュールに対する操作と、本関数（割り込み機能に対する操作）は、無関係なので、どのタイミングで実行しても問題はありません。

RXFIFO_INTERRUPT_USE 定数定義時は、受信 FIFO の割り込み(INTCAN0CFR)有効化
SRFIFO_INTERRUPT_USE 定数定義時は、送受信 FIFO 受信割り込み(INTCAN0/1CFR)有効化
TX_BUF_INTERRUPT_USE 定数定義時は、送信割り込み(INTCAN0/1TRM)有効化
を行います。(有効化したい割り込みを定数定義で与える)[RL78/F14, F13: 0 側のみ]

各割り込みの、割り込み優先度は 0(最高優先度)となります。

can*n*_buf_send (n=0~1) [RL78/F14, F13: n=0 のみ]

概要:データ送信関数
宣言:
int can0_buf_send(unsigned char num, unsigned char ide, unsigned char rtr, unsigned long id, unsigned char dlc, unsigned char *data);
int can1_buf_send(unsigned char num, unsigned char ide, unsigned char rtr, unsigned long id, unsigned char dlc, unsigned char *data);

説明:
・送信バッファを使用してデータの送信
を行います

引数:
num: 送信バッファ番号(0~3)(*1)
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: 送信する ID を指定します
dlc: 送信バイト数を指定します(1~8):1~8 バイト [cann_buf_send]
*data: 送信するデータを指定します
戻り値:
0: 正常終了
-1: 引数チェックエラー
-3: 送信バッファ使用中

補足:

(*1) can1_buf_send()側では、num=0~3 を指定した場合、4~7 に変換されます（CAN1 側では、送信バッファ 4~7 が使用されます）

cann_srfifo_send (n=0~1) [RL78/F14, F13: n=0 のみ]

概要:データ送信関数

宣言:

int can0_srfifo_send(unsigned char ide, unsigned char rtr, unsigned long id, unsigned char dlc, unsigned char *data)

int can1_srfifo_send(unsigned char ide, unsigned char rtr, unsigned long id, unsigned char dlc, unsigned char *data)

説明:

・送受信 FIFO を使用してデータの送信

を行います

引数:

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: 送信する ID を指定します

dlc: 送信バイト数を指定します(1~8):1~8 バイト [cann_cfifo_send]

*data: 送信するデータを指定します

戻り値:

0: 正常終了

-1: 引数チェックエラー

-2: FIFO フル

補足:

CAN ch0 は SRFIFO(0)で送信を行います

CAN ch1 は SRFIFO(1)で送信を行います

can_buf_receive

概要:受信関数

宣言:

int can_buf_receive(unsigned char num, unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

説明:

・受信バッファを使用したデータの受信

を行います

引数:

num: 受信バッファ番号(0~31) [RL78/F14, F13: 0~15]

*ide: 標準／拡張フォーマット区分を格納するポインタ(unsigned char [1])

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

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

*rtr: データフレーム／リモートフレーム区分を格納するポインタ(unsigned char [1])

CAN_DATA_FRAME(0) データフレーム（データ送信）

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

*id: 受信した ID を格納するポインタ(unsigned long [1])

*data: 受信したデータを格納するポインタ（最大 unsigned char [8]）

*ts: データ受信時のタイムスタンプ(unsigned short [1])

戻り値:

0: 受信データなし（DLC=0 のデータはデータなしとして扱う）

1~8: 受信したデータのバイト数

-1: 引数チェックエラー

can_rxfifo_receive

概要:受信 FIFO 受信関数

宣言:

int can_rxfifo_receive(unsigned char fifo_no, unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

説明:

・受信 FIFO を使用したデータの受信

を行います

引数:

fifo_no: 受信 FIFO 番号(0~3) [RL78/F14, F13: 0~1]

*ide: 標準／拡張フォーマット区分を格納するポインタ(unsigned char [1])

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

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

*rtr: データフレーム／リモートフレーム区分を格納するポインタ(unsigned char [1])

CAN_DATA_FRAME(0) データフレーム（データ送信）

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

*id: 受信した ID を格納するポインタ(unsigned long [1])

*data: 受信したデータを格納するポインタ（最大 unsigned char [8]）

*ts: データ受信時のタイムスタンプ(unsigned short [1])

戻り値:

0: 受信データなし（DLC=0 のデータはデータなしとして扱う）

1~8: 受信したデータのバイト数 [cann_rxfifo_receive]

0x2x: 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）

-1: 引数チェックエラー

-2: 受信 FIFO にデータなし

補足:

戻り値が 0x22 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*n*_srfifo_receive (n=0~1) [RL78/F14, F13: n=0 のみ]

概要:送受信 FIFO 受信関数

宣言:

int can0_srfifo_receive(unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

int can1_srfifo_receive(unsigned char *ide, unsigned char *rtr, unsigned long *id, unsigned char *data, unsigned short *ts)

説明:

・共通 FIFO を使用したデータの受信

を行います

引数:

*ide: 標準／拡張フォーマット区分を格納するポインタ(unsigned char [1])

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

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

*rtr: データフレーム／リモートフレーム区分を格納するポインタ(unsigned char [1])

CAN_DATA_FRAME(0) データフレーム（データ送信）

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

*id: 受信した ID を格納するポインタ(unsigned long [1])

*data: 受信したデータを格納するポインタ（最大 unsigned char [8]）

*ts: データ受信時のタイムスタンプ(unsigned short [1])

戻り値:

0: 受信データなし（DLC=0 のデータはデータなしとして扱う）

1~8: 受信したデータのバイト数

0x2x: 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）

-1: 引数チェックエラー

-2: 受信 FIFO にデータなし

補足:

戻り値が 0x22 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。

7.2. プログラムで使用している変数・定数

7.2.1. グローバル変数

unsigned long can_remote_frame_request[]

リモートフレームであることを示すフラグ変数。リモートフレーム送信前にフラグを立て、リモートフレームの処理完了時にフラグを落とす。（画面表示で使用する変数です。）(*1)

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

-- PACKET START--

一連の動作の終わりに、

-- PACKET END--

が表示されます。「-- PACKET END--」の表示は、「-- PACKET START--」から単純に 100ms 経過後です。

CAN0_CAN1_CANBUS_SHARE 定数が未定義の場合、SAMPLE3 では、(*1)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。（リモートパケットの受信から送信完了までの一連の動作）

どちらの変数も、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。

7.2.2. 定数定義

・can¥can.h で定義

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

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

```
#define CAN_ID_FORMAT_SID  0      //標準フォーマット(ID=11bit)
#define  CAN_ID_FORMAT_EID  1     //拡張フォーマット(ID=29bit)
```

標準／拡張フォーマットール定義値。(IDE)

```
#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
```

データフレーム／リモートフレーム定義値。(RTR)

```
#define  CAN_RULE_BUF  0x0        //受信バッファモード
#define  CAN_RULE_RXFIFO0  0x0001      //受信 FIFO0 で受信
#define  CAN_RULE_RXFIFO1  0x0002      //受信 FIFO1 で受信
#define  CAN_RULE_RXFIFO2  0x0004      //受信 FIFO2 で受信 [RL78/F14, F13:  使用不可]
#define  CAN_RULE_RXFIFO3  0x0008      //受信 FIFO3 で受信 [RL78/F14, F13:  使用不可]
#define  CAN_RULE_SRFIFO0  0x0010      //送受信 FIFO0 で受信
#define  CAN_RULE_SRFIFO1  0x0020      //送受信 FIFO1 で受信 [RL78/F14, F13:  使用不可]
```

受信データ格納先定義。

```
#define RXBUF  0        //受信バッファで受信
#define SRFIFO  1       //送受信バッファで受信もしくは送信
#define RXFIFO  2       //受信 FIFO で受信
#define TXBUF  0        //送信バッファで送信
```

送受信方法定数値。

```
#define CAN_RX_METHOD  RXFIFO        //受信FIFOで受信
#define CAN_RX_METHOD  SRFIFO        //送受信 FIFO で受信
#define CAN_RX_METHOD  RXBUF         //受信バッファで受信
```

データ受信方法の定義（いずれかの定義を有効化する）。

```
#define CAN_TX_METHOD  SRFIFO        //送受信 FIFO で送信
#define CAN_TX_METHOD  TXBUF         //送信バッファで受信
```

データ送信方法の定義（いずれかの定義を有効化する）。

```
#define CAN0_RX_RXFIFO_NO  0         //CAN-ch0受信に使用する受信FIFO番号
#define CAN1_RX_RXFIFO_NO  1         //CAN-ch1受信に使用する受信FIFO番号
```

受信 FIFO で使用する FIFO 番号の定義。0~3 が使用可能。[RL78/F14, F13: 0~1 が使用可能]

```
#define CAN_INTERRUPT_DEBUG
```

定義時、デバッグ向けに、割り込み時にポートを反転させる。

・can¥can_operation.h で定義

```
#define CAN_SPEED  CAN_BPS_1M
```

通信速度定義。"CAN_BPS_1M", CAN_BPS_500K", "CAN_BPS_250K", CAN_BPS_125K"のいずれかを定義。

```
#define ID_TYPE  CAN_ID_FORMAT_EID
```

ID を標準 ID(CAN_ID_FORMAT_SID)か、拡張 ID(CAN_ID_FORMAT_EID)のどちらを使用するかを定義。CAN_ID_FORMAT_SID を指定した場合、受信は標準 ID のみ、送信は標準 ID。CAN_ID_FORMAT_EID を指定した場合、受信は拡張 ID と標準 ID の両方。（受信ルールの設定に従う）送信は、起動後は拡張 ID。"s"コマンドで、標準 ID に切り替えての送信が可能となります。

・r_cg_userdefine.h で定義

#define CAN0_CAN1_CANBUS_SHARE [RL78/F14, F13: 未定義]

定数定義時は、CAN0 と CAN1 が同じ CAN バスに接続されている事を想定した画面表示となります。

CAN0 と CAN1 を同じ CAN バスに接続した場合、SAMPLE3 で、CAN0 からリモートフレームを送信すると、

・CAN0 リモートフレーム送信
・CAN0 送信完了
・CAN1 リモートフレーム受信
・CAN1 データフレーム送信（リモートフレームに対する応答の送信）
・CAN1 送信完了
・CAN0 データフレーム受信

の複数のパケットが飛び交い、画面表示が混乱します。本定数定義時は、画面表示の全体を「-- PACKET START --」「-- PACKET END --」で囲み、一連の動作であることを判りやすくします（画面表示上の問題だけで、CAN の動作には影響を与えません）。

複数台のボードで動作確認を行う場合、（HSBRL78F15-100 の CAN0 と CAN1 を接続しない場合、）未定義として問題ないかと考えます。

//#define CAN_MULTIPLE_INTERRUPT_EN

多重割り込みの許可。（デフォルトはコメントアウトで無効化）

定義時は、多重割り込みを許可します。CAN の受信 FIFO 割り込み関数内の処理実行中であっても、送信完了割り込みを掛けたい、という時に有効j化します。
（定義時の利点は、割り込みが入るタイミングが判り易くなる点です。欠点は、場合によって画面に表示されるメッセージが分断されて見難くなる点です。）

#define RXFIFO_INTERRUPT_USE //受信FIFOの割り込みを使用する
#define SRFIFO_INTERRUPT_USE //送受信FIFO割り込みを使用する
#define TX_BUF_INTERRUPT_USE //送信バッファの割り込みを使用する

割り込みの使用を定義する定数です。SAMPLE1 では（割り込みを使用しないので）未定義。SAMPLE2,3 では、定義、としています。

7.3. 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、can*n*_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 40 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。[RL78/F14, F13: 受信ルール上限 16]

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。

（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。

7.4. 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、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 FIFO 受信 (INTCANGRFR)	CAN0 送受信 FIFO 受信 (INTCAN0CFR)	CAN0 チャネル 送信 (INTCAN0TRM)	CAN1 送受信 FIFO 受信 (INTCAN1CFR)	CAN1 チャネル 送信 (INTCAN1TRM)
HSBRL78F15-100	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。

8. 対向機向けのサンプルプログラムに関して

CD 内には、HSBRL78F15-64, HSBRL78F13-64, HSBRL78F14-64 向けの SAMPLE3 のサンプルプログラムが含まれます。

当社 64pin のボードで CAN 通信を試す場合、

SOURCE¥64PIN

以下のサンプルプログラムを参照してください。

なお、RL78/F15 と RL78/F14,F13 では相違が何点かあり、RL78/F14,F13 では、以下の様になっています。

	RL78/F15	RL78/F14, F13
CAN 物理 ch	2(CAN0. CAN1)	1(CAN0)
受信ルール数	40	16
受信メッセージバッファ	40	16
受信バッファ	32	16

※RL78/F15 では、FIFO を 8 段(4 段×2ch)確保しても、受信バッファは 32 個(16 個×2ch)確保できましたが、RL78/F14, 13 では、FIFO を 4 段(4 段×1ch)確保すると、受信バッファは 12 個しか確保できません。(単純に 2ch→1ch で半分という訳でもありません。)

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	2022.9.8	—	初版発行

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/F15
RS-CAN_Lite ソフトウェア編　マニュアル

株式会社 北斗電子
 2022 年 9 月 8 日改訂 REV.1.0.0.0 (220908)