LIN・CAN スタータキット RL78/F15

LIN 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)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、LIN 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)
	RL78F15_LIN3¥		LIN MASTER/SLAVE 通信サンプルプログラム(*2)
	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(*3)
	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(*3)
	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(*3)
	64PIN		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(*3)

(*1)に関しましては、LIN_CAN_KIT_RL78F15_software_REV_x_x_x_x_s.pdf を参照ください。

(*3)に関しましては、LIN_CAN_KIT_RL78F15_software_RS-CAN_Lite_REV_x_x_x_x_s.pdf

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。

RL78F15_LIN3 は、RL78F15_LIN1, RL78F15_LIN2 で分かれていた MASTER/SLAVE の処理を統合したサンプルプログラムとなります。

RL78F15_LIN1, RL78F15_LIN2 では、動作のシーケンスが見える様に意識して構成していますが、フラグ変数を多用しており、プログラム全体の見通しが悪くなる傾向がありました。本書で説明する RL78F15_LIN3 は、シンプルに通信が行える事を考えてプログラムを再構成したものです。通信を行うという観点で不要な処理はユーザに見えない様にしているので、アプリケーションに組み込んで使用する場合は、RL78F15_LIN3 をベースにユーザ側の処理を構築するのが良いと思います。

1. サンプルプログラムの動作確認

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、LIN の動作を見る場合の接続を以下に示します。

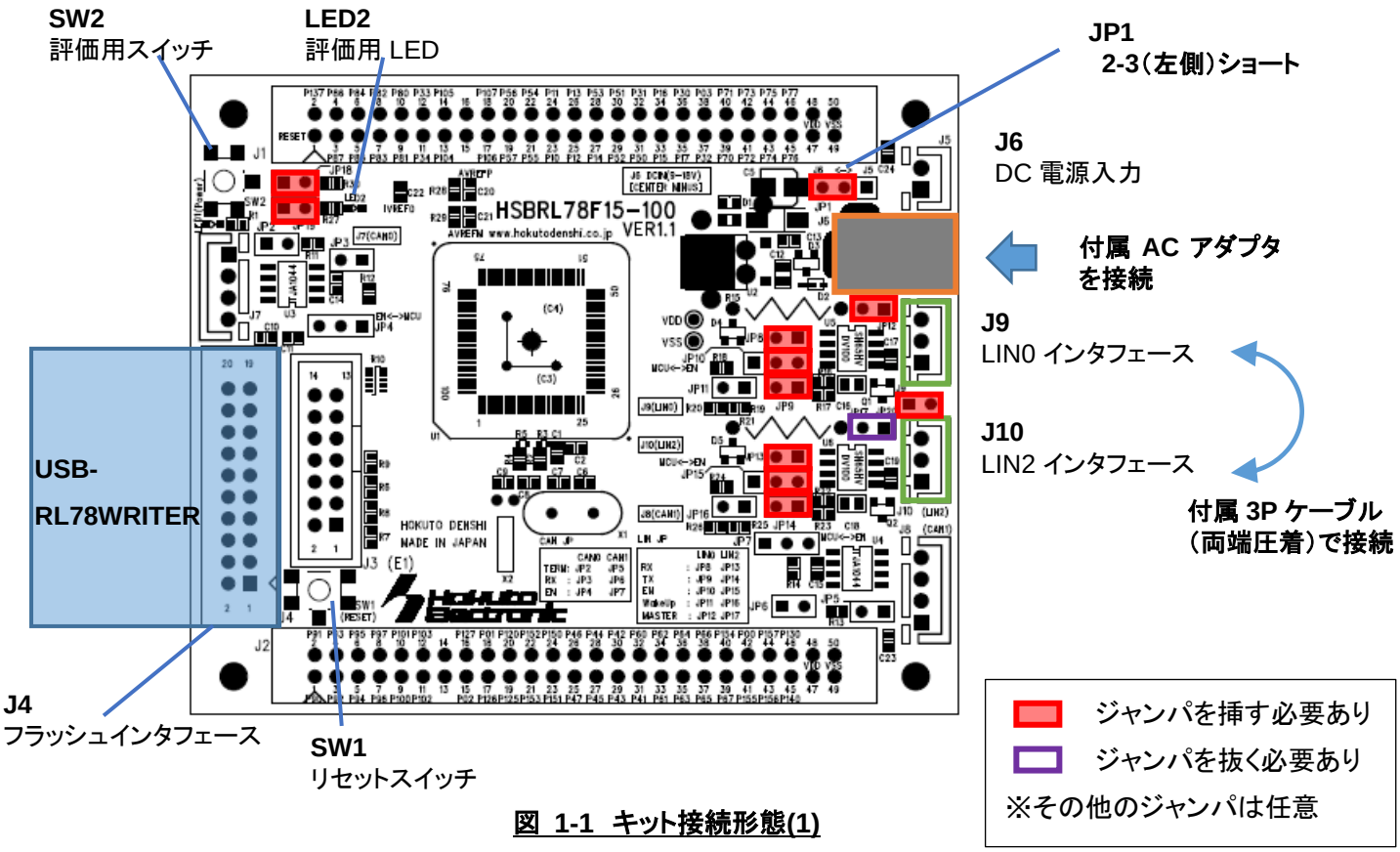


図 1-1 キット接続形態(1)

・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)
・J9 と J10 は、付属 4P ケーブルで接続する
・JP12 はショート(LIN0 側を MASTER に設定)
・JP17 はオープン(LIN2 側は SLAVE に設定)
・JP8~10, JP13~15 はショート(JP10, JP15 は上図では 2-3 ショートに設定していますが、1-2 ショートでも問題ありません)
・JP20 はショート(LIN0/LIN2 は同じ電源を共有)

1.1.2. HSBRL78F15-100 を 2 台使用する場合

RL78F15_LIN3 プロジェクトを変更せずに動作を見る場合、2 台のボードに同じプログラムを書き込み、片方を MASTER、もう一方を SLAVE として動作させてください。

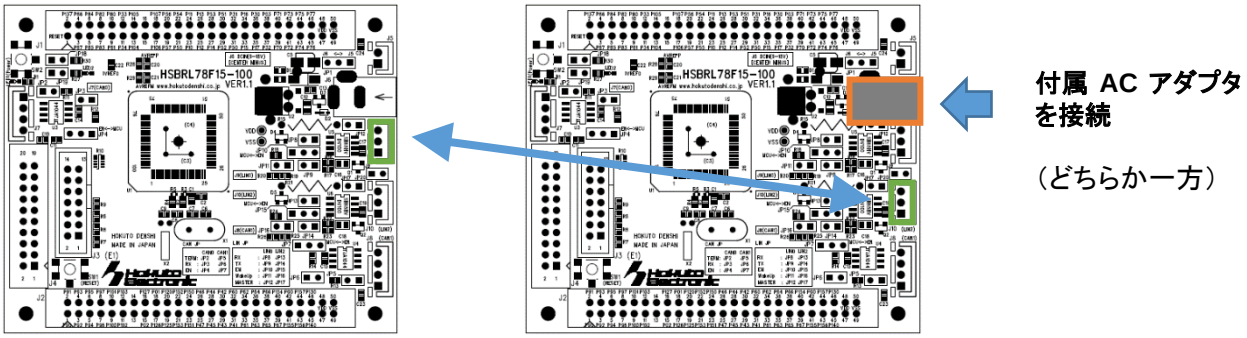


図 1-2 キット接続形態(2)

※ジャンパの設定は、1.1.1 の設定と同じ

1.2. サンプルプログラムの動作確認

USB-RL78WRITER（スライドスイッチ SCI 側）を PC と接続し、PC で 115,200bps で端末を開いてください。

端末には、下記の表示が出力されます。

```
Copyright (C) 2023 HokutoDenshi. All Rights Reserved.

LIN Starter kit RL78/F15 Sample program 3.

Command:
  0 : MASTER header/response send(ID=0x30)
  1 : SLAVE header send(ID=0x31)
  2 : SLAVE header send(ID=0x32)
  3 : SLAVE header send(ID=0x33)
  4 : SLAVE header send(ID=0x34)
  5 : SLAVE header send(ID=0x35)
  6 : SLAVE header send(ID=0x36)
  7 : SLAVE header send(ID=0x37)
  8 : SLAVE header send(ID=0x38)
  9 : SLAVE header send(ID=0x39)
  a-p : MASTER header/response send(ID=0x30) data[2]=0x00~0x0f
>
```

LIN ネットワーク上に LIN-ID=0x32~0x39 のデバイスが存在しない場合、応答なし

1.1.1 の 1 台のボードで動作を行った場合、端末の表示は以下の様になります。

キーボードから"0"を入力すると、

```
--
LIN header/response data send : LIN-ID=0x30 data=0x0000000100800000

LIN data receive(ch2) : LIN-ID=0x30 data=0x0000000100800000 sum=0x8D
```

MASTER 側(ch0)が、ヘッダ・レスポンス送信を行います。(1 行目)
SLAVE 側(ch2)が、MASTER 側から送信されたデータを受信します。(2 行目)

1 行目の data と 2 行目の data は同じ表示となるはずです。

data は、

1	2	3	4	5	6	7	8
0x00	0x00	0x00	0x1	0x00	0x80	0x00	0x00
送信時間(4 バイト) ※ボードを起動してからの秒数				P0 ポート	P13 ポート	0x00 固定	0x00 固定

上記の様になっています。1~4 バイト目は、ボードを起動してから 1 秒毎にインクリメントされる数値です。5 バイト目は、ポート 0 のレジスタ値(P0)、6 バイト目はポート 13 のレジスタ値(P13)で、7,8 バイト目は 0x00 固定です。(ボードの LED2 が P03 につながっているので、5 バイト目の bit3 が LED2 の点灯情報。SW2 が P137 につながっているので、6 バイト目の bit7 が SW2 を押しているかどうかの情報となります。)

キーボードから"1"を入力すると、

```
--
LIN header send : LIN-ID=0x31

LIN data receive(ch0) : LIN-ID=0x31 data=0x0000000200800000 sum=0xCB
```

MASTER 側(ch0)が、LIN-ID=0x31 のヘッダ送信を行います。(1 行目)
SLAVE 側(ch2)が、MASTER 側から送信された ID が自分自身の ID と一致したので、レスポンスデータを送信します。
MASTER 側(ch0)が、SLAVE 側から送信されたデータを受信して表示します。(2 行目)

※データの内容は、MASTER 側から送るものと同じです

キーボードから"2"を入力すると、

```
--
LIN header send : LIN-ID=0x32
```

MASTER 側から、LIN-ID=0x32 のヘッダ送信を行いますが、応答する SLAVE が居ないので、レスポンスが返ってくる事はありません。
（別途、LIN バスに、ID=0x32 のデバイスを接続した場合は、応答が返ってくるはずです。）

MASTER と SLAVE のボードを別にした場合は、LIN header send 及び LIN data receive(ch0)は、MASTER 側のボード、LIN data receive(ch2)は SLAVE 側のボードに表示されます。

SLAVE 側から送信されるデータの 6 バイト目の bit7 が 0（SW2 が押されている）場合は、LED2 が点灯します。
（SW2 を押した状態(*)で、キーボードから"1"を押して、MASTER 側がデータを受信したタイミングで LED2 の点灯・消灯が切り替わります。）
(*)SLAVE 側の送信データ更新は 1 秒間隔なので、正確には SW2 を押して最大 1 秒待った後にデータを受信する必要があります。

1.3. HSB_LIN_COMM を接続する場合

HSB_LIN_COMM は、LIN デバッグ用の外部デバイスです。SLAVE デバイスとして動作(*)し、MASTER からのヘッダ送信により、レスポンスデータを返送します。

(*)ジャンパ設定により MASTER デバイスとして動作させる事も可能です

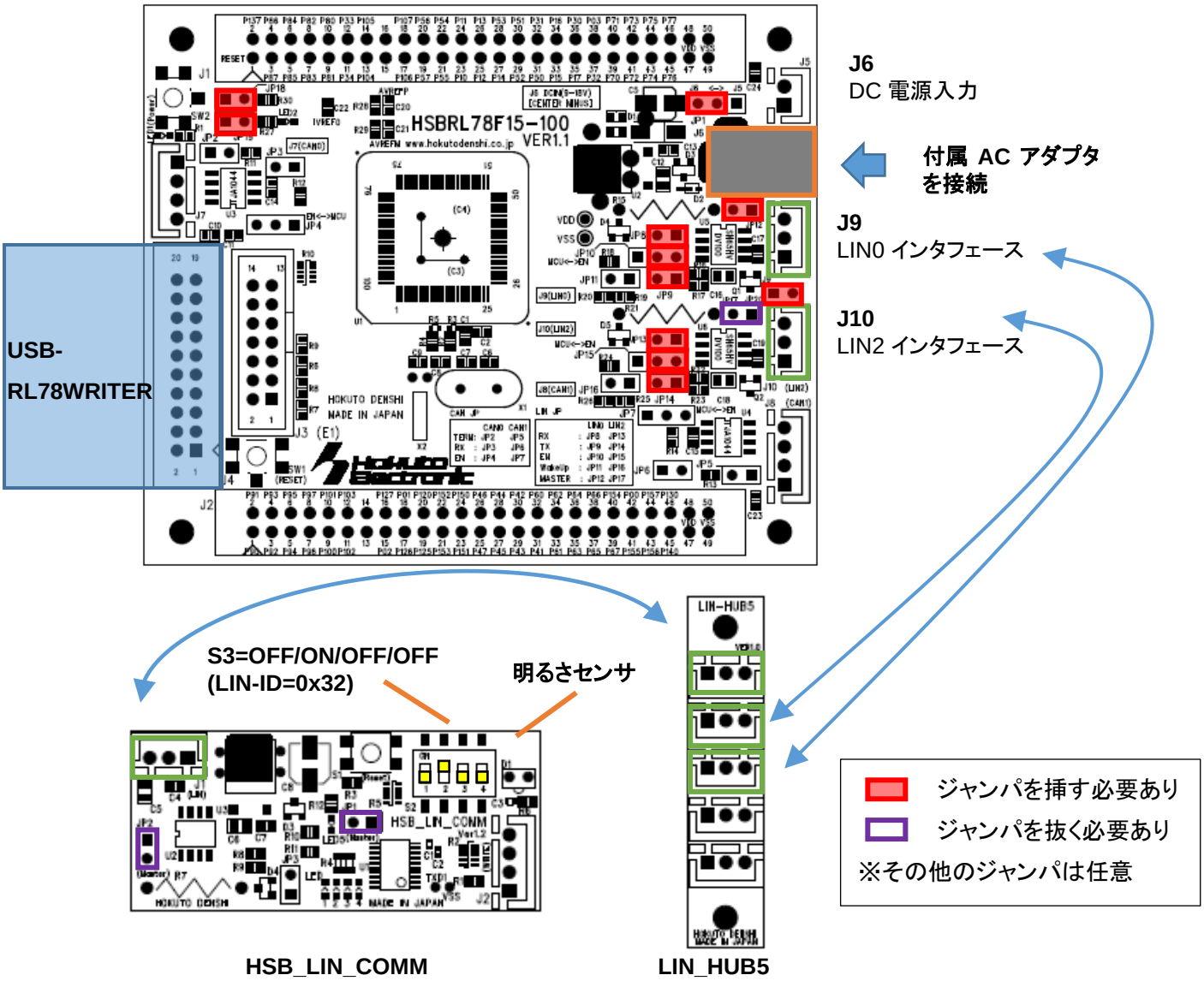


図 1-3 HSB_LIN_COMM との接続

※J10(LIN-ch2)を未接続として、HSBRL78F15-100(J9)と HSB_LIN_COMM の LIN コネクタを 1:1 接続としても良いです

・HSB_LIN_COMM の S3 は、OFF/ON/OFF/OFF に設定（LIN-ID=0x32 設定）
・HSB_LIN_COMM の JP1 と JP3 はオープン（SLAVE 設定）

キーボードから"0"を入力すると、

```
--
LIN header/response data send : LIN-ID=0x30 data=0x0000000500800000
LIN data receive(ch2) : LIN-ID=0x30 data=0x0000000500800000 sum=0x89
```

ch0 から送信したデータを ch2 が受信（HSBRL78F15-100 単体で動かした場合と同じ）

キーボードから"1"を入力すると、

```
--
LIN header send : LIN-ID=0x31
LIN data receive(ch0) : LIN-ID=0x31 data=0x0000000A00800000 sum=0xC3
```

ch0 が送信したヘッダを ch2 が受信してレスポンスデータを送信、そのデータを ch0 が受信（HSBRL78F15-100 単体で動かした場合と同じ）

キーボードから"2"を入力すると、

```
--
LIN header send : LIN-ID=0x32
LIN data receive(ch2) : LIN-ID=0x32 data=0x01CD040002640000 sum=0x94
LIN data receive(ch0) : LIN-ID=0x32 data=0x01CD040002640000 sum=0x94
```

ID=0x32 のデバイス(HSB_LIN_COMM)がレスポンスデータを送信、そのデータを ch0 と ch2 が受信。（ch0 と ch2 が受信するデータは同じ）※HSBRL78F15-100 単体と異なる

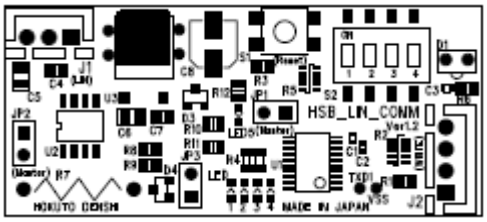
```
--
LIN header/response data send : LIN-ID=0x30 data=0x000000000000000
LIN data receive(ch2) : LIN-ID=0x30 data=0x0000000000000000 sum=0x0F

--
LIN header/response data send : LIN-ID=0x30 data=0x000001000000000
LIN data receive(ch2) : LIN-ID=0x30 data=0x0000010000000000 sum=0x0E

--
LIN header/response data send : LIN-ID=0x30 data=0x000002000000000
LIN data receive(ch2) : LIN-ID=0x30 data=0x0000020000000000 sum=0x0D

--
LIN header/response data send : LIN-ID=0x30 data=0x00000F0000000000
LIN data receive(ch2) : LIN-ID=0x30 data=0x00000F0000000000 sum=0x00
```

'a'コマンド

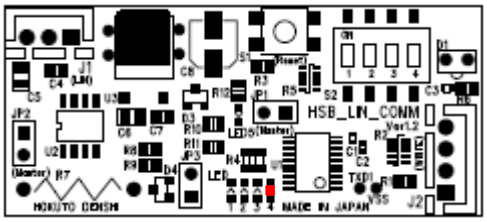
'b'コマンド

'c'コマンド

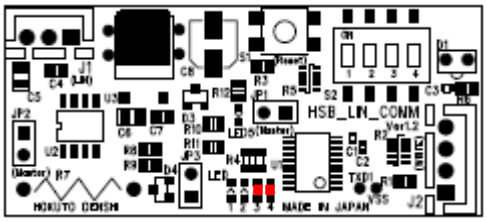
'p'コマンド



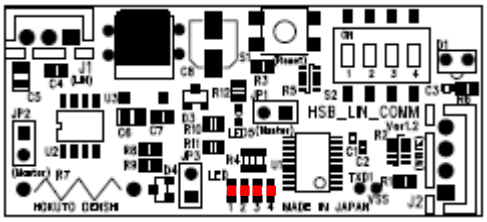
'a'コマンド
→LED1~4=OFF/OFF/OFF/OFF



'b'コマンド
→LED1~4=OFF/OFF/OFF/ON



'c'コマンド
→LED1~4=OFF/OFF/ON/ON



'p'コマンド
→LED1~4=ON/ON/ON/ON

'a'~'p'のコマンドは、送信データの 3 バイト目のデータが変わります。'a'が data[2]=0x00, 'b'が data[2]=0x01, …, data[2]=0x0f となります。HSB_LIN_COMM は受信した 3 バイト目のデータにより LED の点灯状態が変わる仕様なので、'a'~'p'コマンドで、HSB_LIN_COMM ボードの LED の点灯状態を LIN 通信経由で制御できます。

HSB_LIN_COMM ボードは、(出荷時のプログラムで)ID=0x31~0x34 の設定が可能で、DIP-SW と明るさセンサの情報をレスポンスデータに乗せて送信するような機能を持っています。
(HSB_LIN_COMM ボードは、LIN の通信相手として使用可能なボードです。)

ー[参考]HSB_LIN_COMM が送信するデーターー

```
--
LIN header send : LIN-ID=0x32
LIN data receive(ch2) : LIN-ID=0x32 data=0x01CD040002640000 sum=0x94
LIN data receive(ch0) : LIN-ID=0x32 data=0x01CD040002640000 sum=0x94
```

1	2	3	4	5	6	7	8
0x01	0xCD	0x04	0x00	0x02	0x64	0x00	0x00
100ms 毎にインクリメントされるシリアル番号		DIP-SW の情報	LED の情報	明るさセンサの情報		0x00 固定	0x00 固定

HSB_LIN_COMM は、LIN の対向機（LIN の通信相手）として使用可能なボードです。

HSB_LIN_COMM がデータを受信した際は、3 バイト目のデータに応じて LED の点灯状態を変えられる様になっています。

HSB_LIN_COMM が送信するデータは、DIP-SW の情報や明るさセンサの情報等です。

2. LIN 通信の概要

2.1. 接続形態

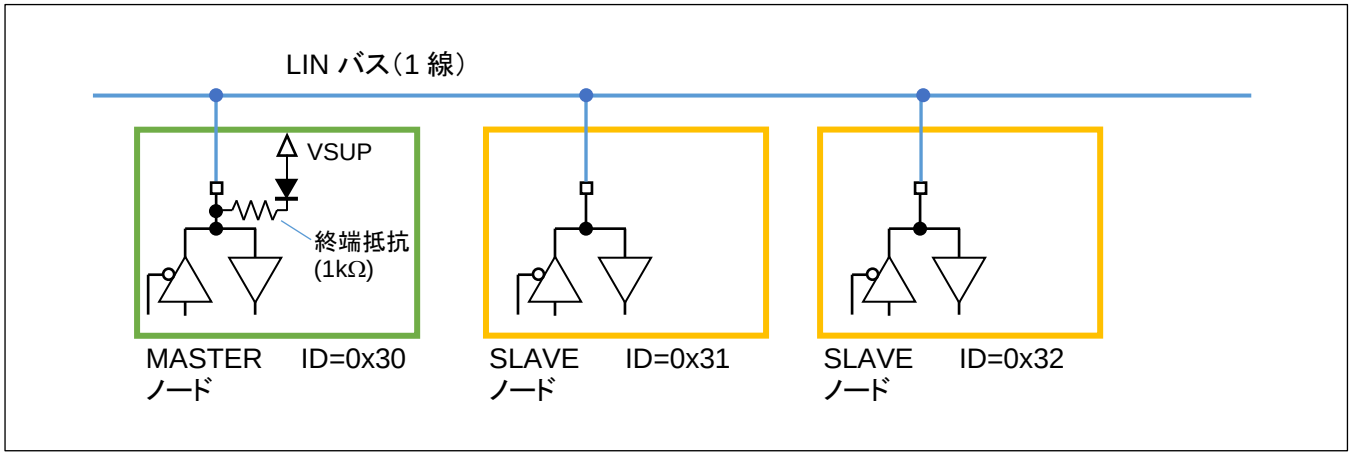


図 2-1 LIN 接続形態

VSUP は
基本的には自動車の
バッテリの電源(12~14V)となります

LIN は、1 線の LIN バスのラインに複数のノードがぶら下がる形となり、以下のような形態を取ります。

・LIN バスは 1 線の信号ラインとなる
・1 本の信号線で、データの送受信を行う
・MASTER ノードは、1 つのバスに 1 つのみ
・SLAVE ノードは、1 つのバスに複数存在してよい
・バスの終端は MASTER ノードで行う
・各ノードは重複しない ID を有する

2.2. LIN の物理層

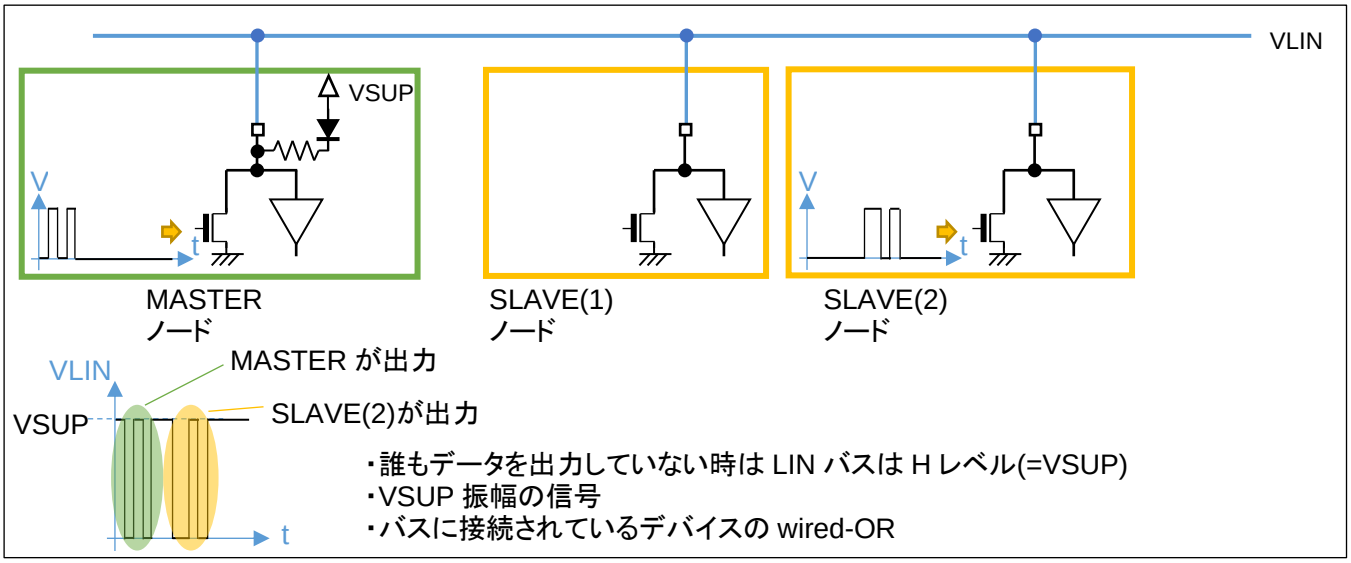


図 2-2 LIN 物理層

LIN バスの信号は、各ノードの出力の wired-OR（いずれかのノードが L を出力すると、LIN バスは L）となります。そのため、各ノードが自由なタイミングで出力を始めると、信号衝突が起こるため、出力を行うタイミングは MASTER が制御する事となります。CAN と異なり、データの衝突は許されていません。また、LIN バスの振幅は、VSUP 振幅となります。

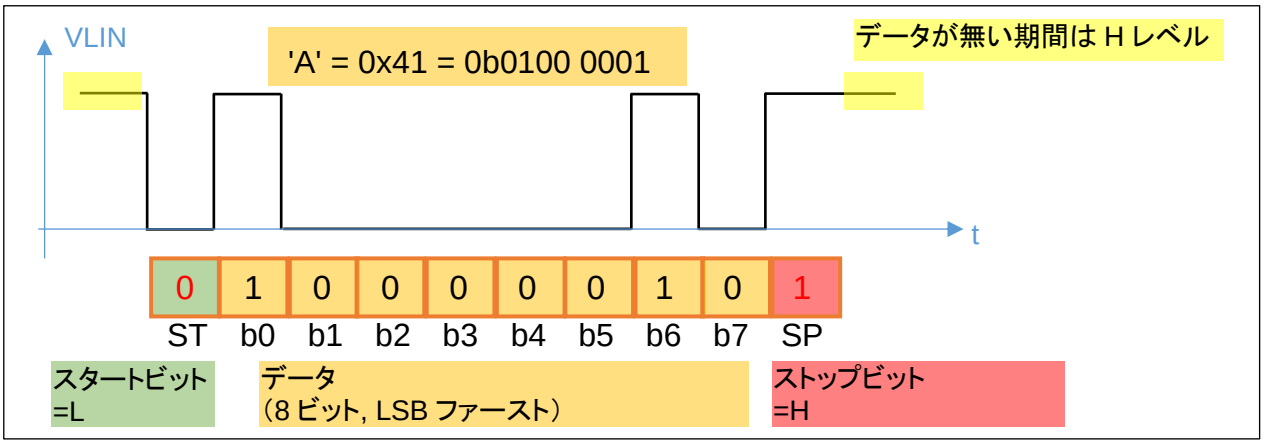


図 2-3 LIN の 1 バイト

LIN で 1 バイト(A=0x41)を送信する場合の波形は上記の様になります。スタートビット=0、ストップビット=1 が付与され、データは LSB（下位ビット）から順に送信されます。このフォーマットは、UART（調歩同期式通信）と同様です。

また、LIN の通信速度は一般的には、~20kbps です。

2.3. LIN のデータパケット

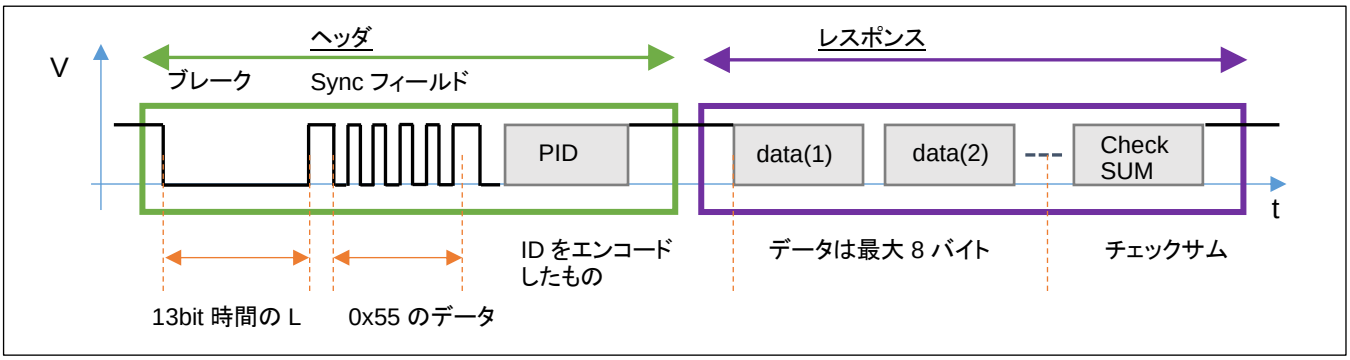


図 2-4 LIN データパケット

LIN のデータパケットは、ヘッダとレスポンスから構成され、以下の特徴を持ちます。

・ヘッダを送出するのは MASTER ノードのみ

・レスポンスは、MASTER か SLAVE ノードが送出する

・SLAVE ノードは、ヘッダに含まれる ID が自ノードの ID と一致した場合に、レスポンスを返す事ができる

・レスポンスデータは、データとチェックサムで構成される

・バスに流れているデータは、どのノードでも受信可能

ヘッダは、ブレーク（13bit 時間（以上）の L）、Sync フィールド（0x55 のデータ）、PID(Protected ID)から構成されます。

ブレークは信号開始の合図。Sync フィールドは、受信側に通信速度を同期させる役割（受信側は Sync フィールドのパルス幅を測定して、送信側の通信速度に合わせる事ができる）があり、PID は ID をエンコード（ID にチェックサムを付与したデータ）となっています。

・MASTER レスポンス送信

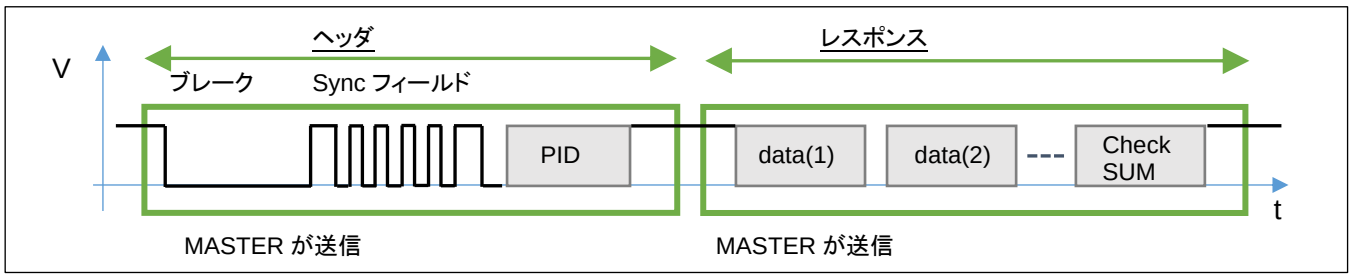


図 2-5 MASTER レスポンス送信

MASTER レスポンス送信では、ヘッダ、レスポンスとも MASTER ノードが送信を行います。SLAVE ノードでは、このデータを受信可能です（SLAVE 側は受信動作のみ）。

・SLAVE レスポンス送信

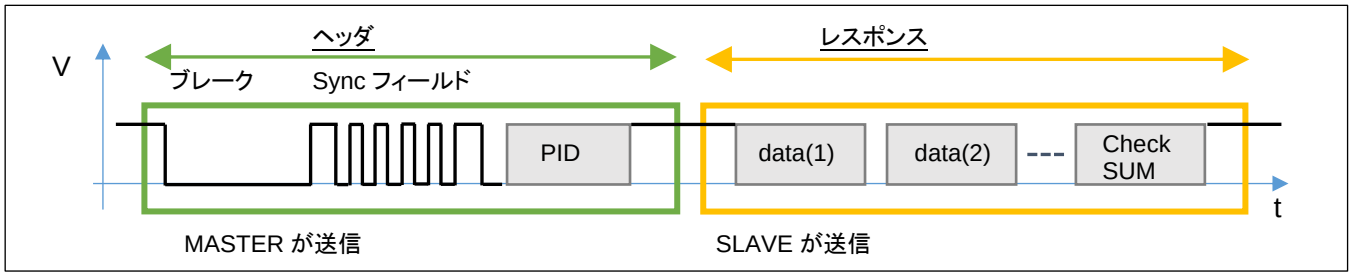


図 2-6 SLAVE レスポンス送信

SLAVE レスポンス送信では、ヘッダは MASTER ノードが出力しますが、レスポンスは SLAVE ノードが送信を行います。SLAVE ノードでは、ヘッダに含まれる PID（ID コードをエンコードしたもの）が自分自身の ID である場合に、レスポンス送信を行います。MASTER 側はレスポンスデータを受信し、SLAVE がどのようなデータを送信したのかを知ることができます。

3. サンプルプログラムに含まれる関数の使用法

3.1. MASTER 動作

LIN インタフェースを MASTER に設定して使用する場合の使用法を示します。

3.1.1. 初期化

```
lin_init(0, LIN_MASTER, 0x30);
```

1 つ目の引数は、LIN の ch を指定します。RL78/F15 の場合は、0~2 を指定可能です。
2 つ目の引数は、LIN_MASTER(0)か LIN_SLAVE(1)を指定します。この定数は、lin.h で定義されています。
3 つ目の引数は、LIN の ID を指定します。0x0~0x3f の値が指定可能です。
※MASTER 設定の場合、ここで設定した ID はグローバル変数に保持されますが、動作で使用される事はありません

3.1.2. MASTER ヘッダ・レスポンス送信

```
unsgned char data[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

lin_master_header_response_send(0, 0x30, data, 8);
```

MASTER 側から、ヘッダ＋レスポンスデータを送信する場合、

1 つ目の引数は、LIN の ch を指定します。
2 つ目の引数は、ヘッダに含まれる LIN の ID を指定します。
3 つ目の引数は、送信データ（最大 8 バイト）を指定します。
4 つ目の引数は、送信バイト数（1~8）を指定します。

3.1.3. MASTER ヘッダ送信（SLAVE レスポンス要求）

```
lin_master_header_send(0, 0x31, 8);
```

MASTER 側から、ヘッダを送信する場合、

1 つ目の引数は、LIN の ch を指定します。
2 つ目の引数は、ヘッダに含まれる LIN の ID を指定します。（データを送信して欲しい SLAVE の ID）
3 つ目の引数は、受信予定のバイト数（1~8）を指定します。

※CAN のリモートフレーム要求の場合、送信元が DLC（データバイト数）をパケットに埋め込んで送信します。受信側（レスポンスデータを返送する側）は、受信した DLC に基づき、（基本的には）DLC のバイト数のレスポンスデータを送信します。

それに対し、LIN の場合は、MASTER から送信するヘッダには、データのバイト数の情報が含まれません。上記関数の受信予定バイト数（3 番目の引数）は、チェックサムの計算に用いられます。この値と、SLAVE 側が返送するバイト数が合っていないと、チェックサムエラーとなり、受信データは捨てられます。

LIN の場合は、パケット毎にデータバイト数を変えるのは、色々弊害があるので、データバイト数（パケット長）は、lin.h 内で、LIN_RESPONSE_DATA_SIZE 定数で定義しています（固定化しています）。

（通信の途中で、レスポンスデータのバイト数を変える場合は、MASTER 側と SLAVE 側で何らかの手段でデータバイト数の値の情報を共有する仕組みの追加が必要になります。

3.1.1 の初期化は、最初に 1 回（もしくは、ID や MASTER/SLAVE 種別を変更するタイミングで）行ってください。初期化後は、3.1.2, 3.1.3（MASTER レスポンス送信、MASTER ヘッダ送信）を任意のタイミングで実行して構いません。

3.2. SLAVE 動作

LIN インタフェースを SLAVE に設定して使用する場合の使用法を示します。

3.2.1. 初期化

```
lin_init(2, LIN_SLAVE, 0x31);
```

1 つ目の引数は、LIN の ch を指定します。RL78/F15 の場合は、0~2 を指定可能です。
2 つ目の引数は、LIN_MASTER(0)か LIN_SLAVE(1)を指定します。この定数は、lin.h で定義されています。
3 つ目の引数は、LIN の ID を指定します。0x0~0x3f の値が指定可能です。

3.2.2. レスポンスデータ設定

```
unsgned char data[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

lin_response_data_set(2, data, 8);
```

1 つ目の引数は、LIN の ch を指定します。
2 つ目の引数は、送信データ（最大 8 バイト）を指定します。
3 つ目の引数は、送信バイト数（1~8）を指定します。

SLAVE がデータ(レスポンスデータ)を送信できるのは、自分自身の LIN-ID を含むヘッダが LIN バスに流れてきた場合(自分自身の LIN-ID を受信した場合)のみです。ヘッダ送信が行えるのは、MASTER デバイスのみですので、SLAVE 側は自らのタイミングでの送信はできません。

本関数は、受信したヘッダに自分自身の ID が含まれている際にレスポンスとして返信するデータを設定します。なお、実際のデータの送信処理は、受信割り込み関数内で行われます。

※レスポンスデータ設定関数 lin_response_data_set()呼び出し直後(lin_response_data_set()から処理が戻る前)に受信割り込みが入った場合、前回 lin_response_data_set()で設定したデータが送信されます

3.2.3. 受信待機

```
lin_slave_header_receive(2);
```

1 つ目の引数は、LIN の ch を指定します。

指定した ch の受信をスタンバイします。本関数は、3.2.1 の初期化後 1 回実行してください。

3.2.1 の初期化は最初に 1 回行ってください。3.2.2 のレスポンスデータ設定と 3.2.3 の受信待機はどちらが先でも問題ありません。

3.2.2 のレスポンスデータ設定は、任意のタイミングで何度でも実行可能です。

3.3. レスポンスデータの受信(MASTER/SLAVE 共)

データの受信は、受信割り込み関数内で行われ、受信データは、受信バッファのグローバル変数(g_lin_recv_buf)に保持されます。

MASTER 動作の場合、ヘッダ送信を行ってレスポンスが返ってきた際にデータを受信します。

SLAVE 動作の場合、自分自身の LIN-ID 以外のレスポンスデータ(MASTER がレスポンス送信したデータと、他の SLAVE がレスポンス送信したデータ)を受信します。

(MASTER, SLAVE 共、自分自身が送信したデータは、受信しません。)

受信データを保持する、g_lin_recv_buf は lin_message 構造体としており、

```
typedef struct{
    unsigned char id;
    unsigned char data[8];
    unsigned char sum;
    unsigned char size;
} lin_message;
```

LIN-ID とレスポンスデータ、チェックサム、データバイト数を保持します。

受信バッファのサイズは、lin.h 内で LIN_RECV_BUF_SIZE（=デフォルト値 16）として定義されています。

受信バッファに保持されているデータの読み出しは、

```
int ret;
lin_message lin_msg;

ret = lin_read_data(0, &lin_msg);
```

の様に行います。

受信データがある場合、ret=0 となります。受信しているデータが無い場合、ret=2 となります。
（ret != 2 の間 lin_read_data()を複数回実行すると、格納されている全てのデータを取り出せます。）
※ret == 1 の場合は、バッファが溢れて捨てられたデータがある事を示しています、バッファが溢れた場合古いデータから破棄されます（ret == 1 の場合でも取り出したデータは有効です）

現状のサンプルプログラムでは、受信割り込み関数では受信バッファにデータを格納する処理としています。受信時に何かアクションを行わせたい場合は、受信コールバック関数を有効化して、受信コールバック関数内（lin***n***_interrupt_receive_callback(), n=0~2, ch 番号）内に処理を追加してください。※詳細は 5.3 節参照

4. サンプルプログラムの動作

サンプルプログラムは、2 種類の動作モードを用意してあり、

lin_operatopn.h

```
//動作モード（どちらかを選択）
//#define LIN_OPERATION_MODE  LIN_OPERATION_TIMER    //下記のインターバル時間毎にヘッダ送信
#define LIN_OPERATION_MODE  LIN_OPERATION_UART    //UARTからのコマンドでヘッダ送信
```

デフォルトは、キーボードからのコマンド

0: MASTER レスポンス送信
1~9: SLAVE に対しヘッダ送信
a-q: MASTER レスポンス送信（データの 3 バイト目 0x00~0x0f で送信）

で動作します。動作モードを LIN_OPERATION_TIMER 側に設定した場合は、（デフォルトでは）1 秒毎に

1. MASTER レスポンス送信
2. LIN-ID=0x31 のヘッダ送信
3. LIN-ID=0x32 のヘッダ送信
4. LIN-ID=0x33 のヘッダ送信
5. LIN-ID=0x34 のヘッダ送信

1~5 を繰り返す、動作となります。（#define LIN_OPERATION_MODE　LIN_OPERATION_UART での、キーボードの、0,1,2,3,4 を 1 秒毎に入力した動作）

送信データは、

・シリアル NO（4 バイト） RTC でカウントした 1 秒毎にインクリメント
・P0 レジスタの値（1 バイト）
・P13 レジスタの値（1 バイト）
・フィルデータ（2 バイト） 0x00 0x00

の 8 バイトです。

－送信データ（MASTER, SLAVE 共同じ）－

1	2	3	4	5	6	7	8
0x00	0x00	0x00	0x01	0x00	0x80	0x00	0x00
送信時間（4 バイト） ※ボードを起動してからの秒数				P0 ポート	P13 ポート	0x00 固定	0x00 固定

ー送信データ（MASTER 側 a~p コマンド）ー

1	2	3	4	5	6	7	8
0x00	0x00	0x0X	0x00	0x00	0x00	0x00	0x00
0x00 固定	0x00 固定	'a': X=0 'b': X=1 … 'p': X=f	0x00 固定	0x00 固定	0x00 固定	0x00 固定	0x00 固定

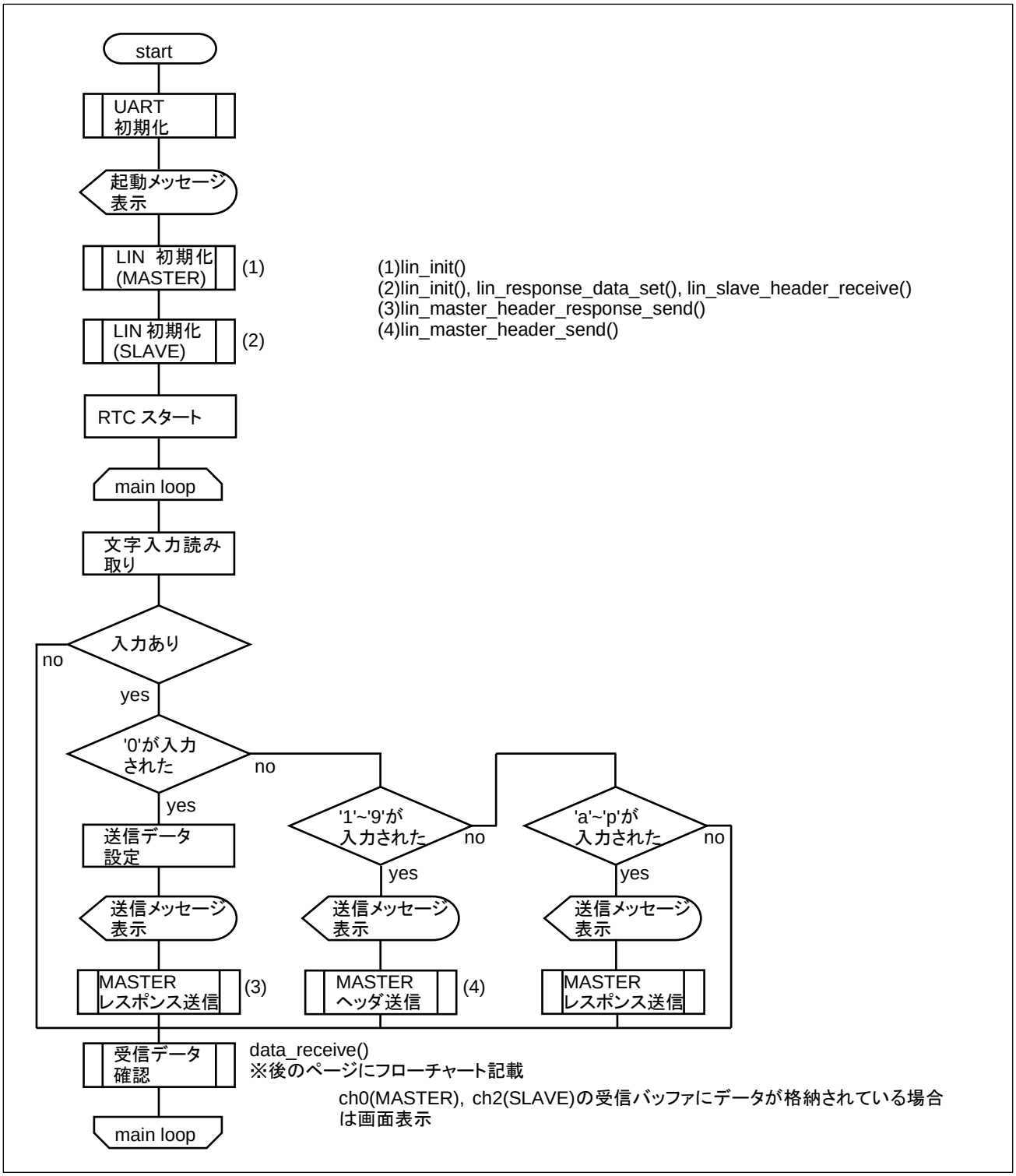
※a~p コマンドは、SLAVE として HSB_LIN_COMM を接続した場合向けのコマンドです

MASTER 側は、送信のタイミングで、送信データを設定します。

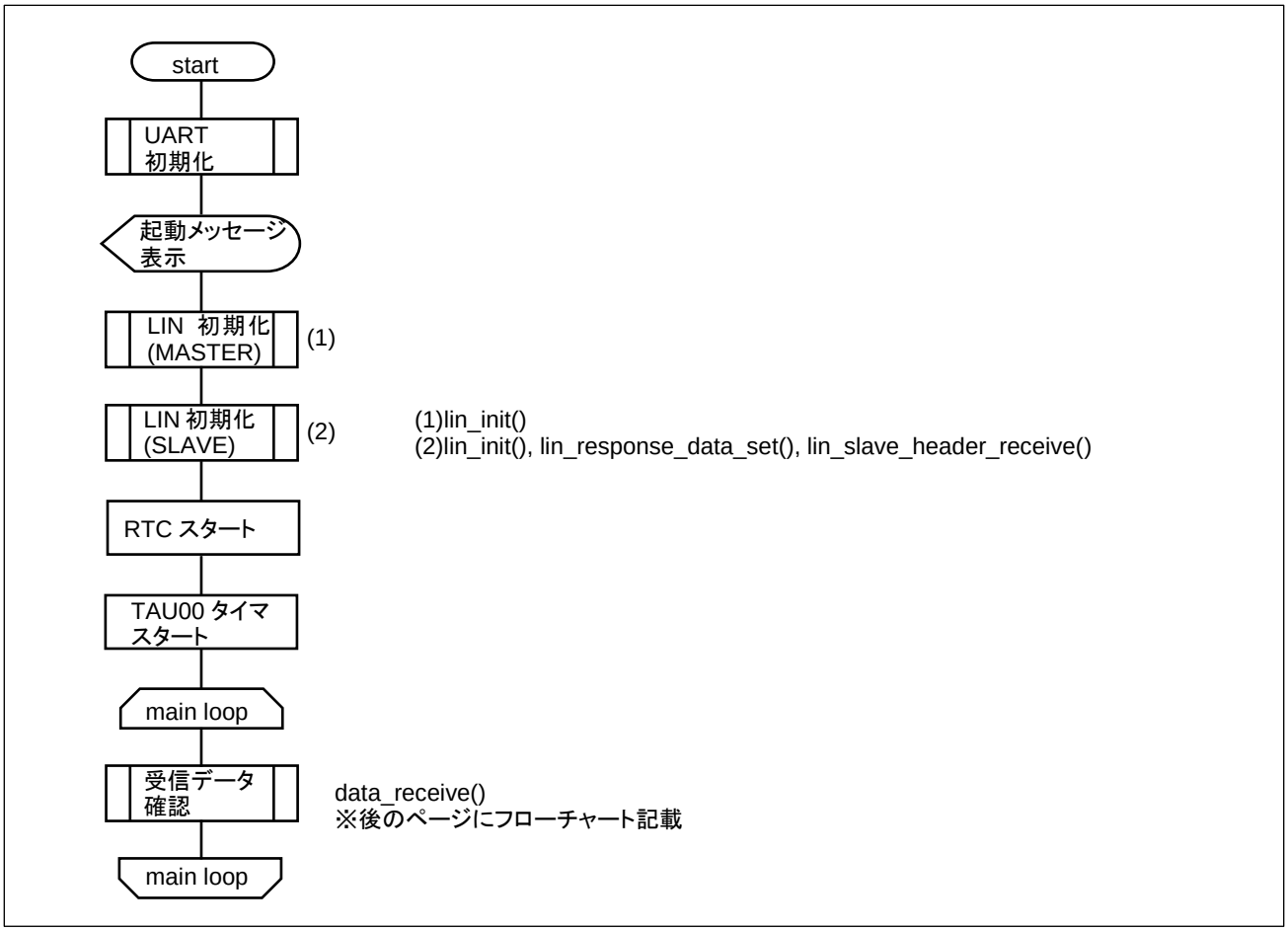
SLAVE 側は、RTC の 1 秒毎のタイミングで、送信データを設定します。

－処理フロー－

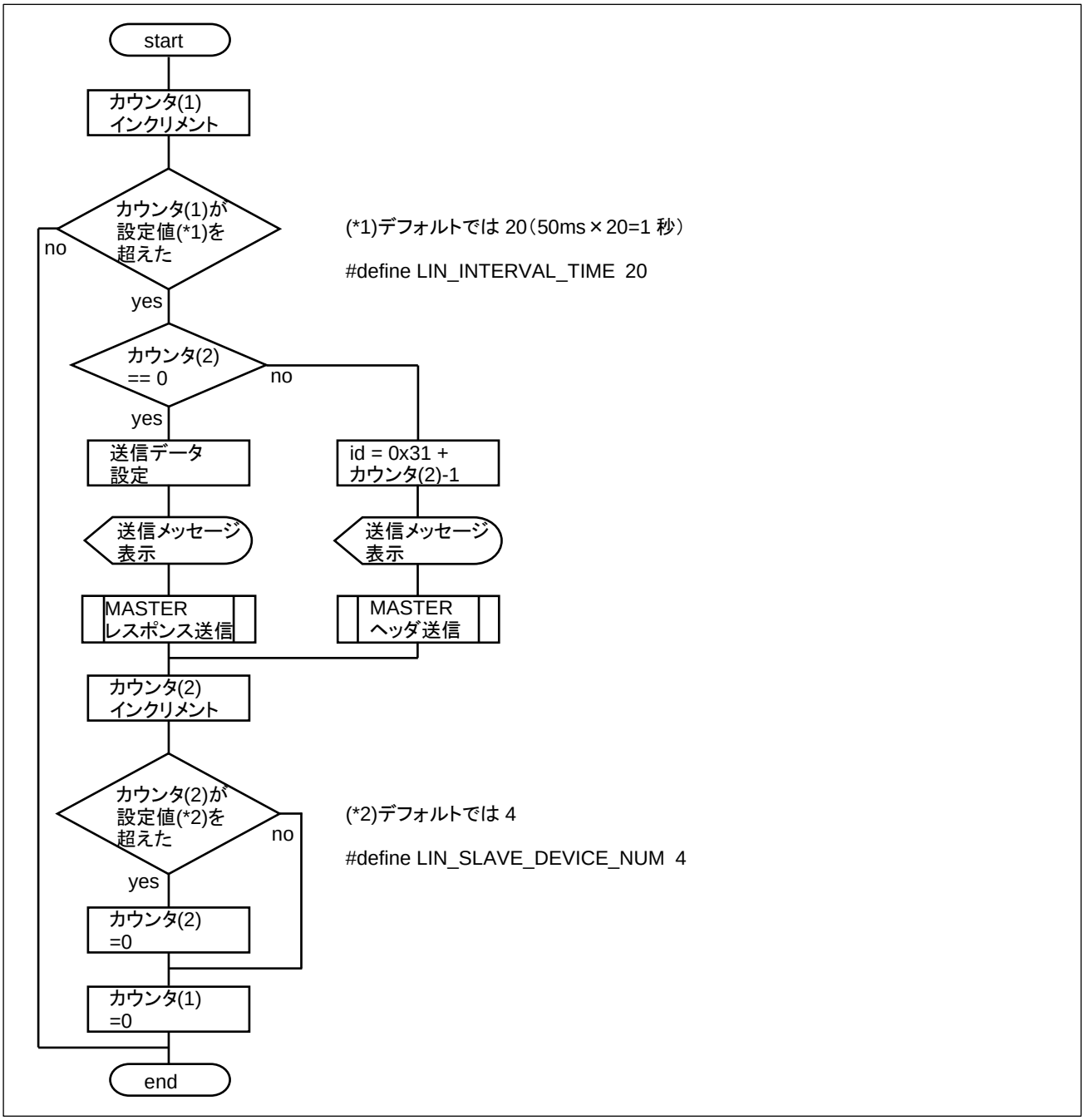
メイン関数　r_main.c 内 main()　※キーボードからのコマンドで動作させる場合



メイン関数　main()　※タイマで動作せる場合[#define LIN_OPERATION_MODE LIN_OPERATION_UART]



タイマ(TAU00)処理　timer.h 内　timer_tau00()　50ms 毎の定期処理　※タイマで動作させる場合のみ使用



MASTER レスポンス送信

MASTER ヘッダ送信(LIN-ID=0x31)

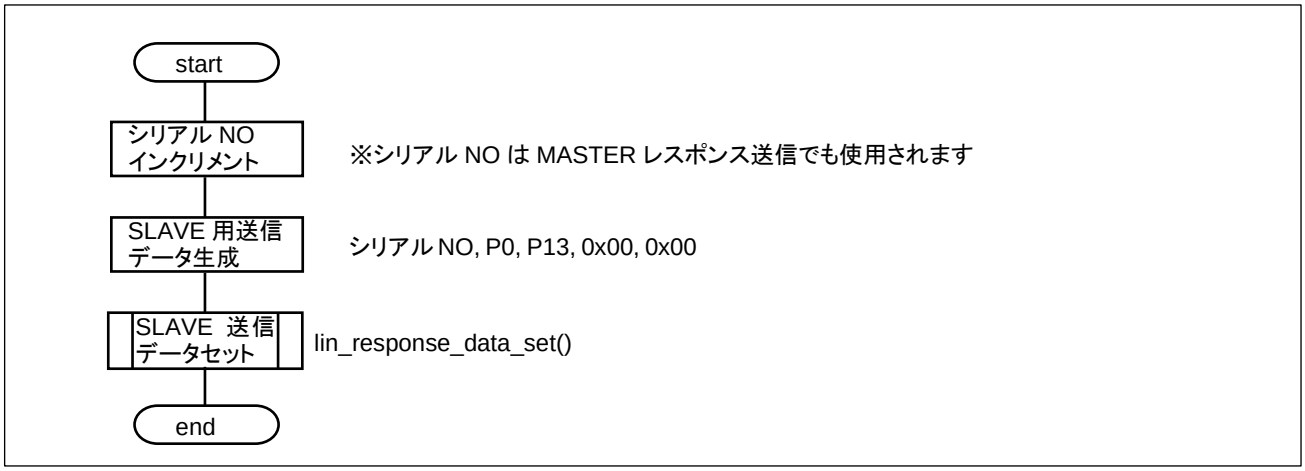
MASTER ヘッダ送信(LIN-ID=0x32)

MASTER ヘッダ送信(LIN-ID=0x33)

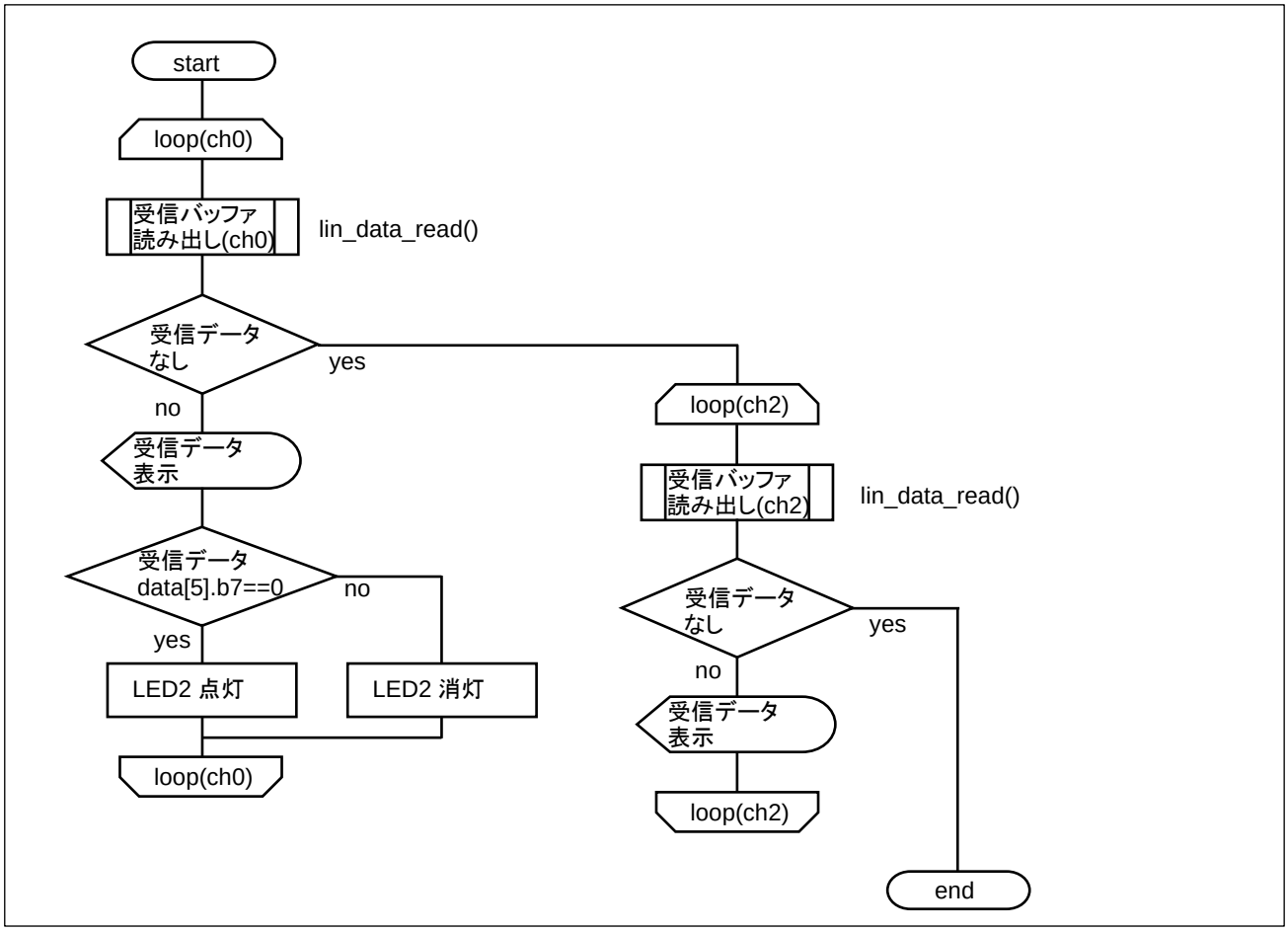
MASTER ヘッダ送信(LIN-ID=0x34)

上記動作を、1 秒間隔で繰り返します。

1 秒毎の定期処理　r_cg_rtc_user.c 内　r_rtc_callback_constperiod()



受信データ確認　r_main.c 内　data_receice()



受信データ処理は、メイン関数のループ内で呼び出されます。LIN-ch0 側(MASTER)の受信バッファが空になるまで受信データの表示を行い、次に LIN-ch2 側(SLAVE)の受信バッファが空になるまで受信データの表示を行います。

5. 関数仕様

・ソースファイル構成

フォルダ	ファイル	説明
RL78_F15_LIN3¥usr_src¥lin		LIN ソースフォルダ
	lin.c	LIN 通信処理関数
	lin.h	lin.c 向けヘッダファイル
	lin_operation.h	速度や ID 等の定義
RL78_F15_LIN3¥usr_src¥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
RL78_F15_LIN3¥usr_src¥timer		タイマ処理フォルダ
	timer.c	定期処理関数
	timer.h	timer.c 向けヘッダファイル

5.1. ヘッダファイルで定義している定数

5.1.1. lin.h の定数定義

lin.h

```
/*-----------------------------------------------------------------------
    定数定義（ユーザ設定箇所）
-----------------------------------------------------------------------*/
//定義時端末にデバッグ表示を行う
//#define SCI_DEBUG

//レスポンスデータサイズ
#define LIN_RESPONSE_DATA_SIZE  8    //レスポンスデータのサイズを送信側と受信側で同じ値とする

//受信バッファサイズ
#define LIN_RECV_BUF_SIZE  16    //lim_message構造体のバッファ数×ch分のメモリを消費

//割り込みコールバック関数の有効化
//#define LIN_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 LIN_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 LIN_USE_STATUS_CALLBACK_FUNCTION  //ステータス割り込み関数のコールバック関数を使用
する

/*-----------------------------------------------------------------------
    定数定義
-----------------------------------------------------------------------*/
//モード
#define LIN_MASTER  0
#define LIN_SLAVE  1

//LIN-ch数
#define LIN_CH_NUM  3    //RL78/F15の場合3ch
#define LIN_CH0  0
#define LIN_CH1  1
#define LIN_CH2  2
```

lin.h（続き）

```
//TXフラグ
#define LIN_TX_FLAG  0x1    //送信中
#define LIN_TX_RESPONSE_FLAG  0x2    //マスタレスポンス送信

//RXフラグ
#define LIN_RX_READY  0x1    //レスポンス受信待機

//速度設定(bps)
#define LIN_SPEED_2400  1
#define LIN_SPEED_9600  2
#define LIN_SPEED_10417  3
#define LIN_SPEED_19200  4

//速度設定(LBRPxx)
#define LIN_SPEED_LBRP0  104    //2400bps, 9600bps, 19200bps向け
#define LIN_SPEED_LBRP1  96    //10417bps向け

//速度設定（システムクロック選択ビット）
#define LIN_SPEED_2400_LCKS  2    //fc(LBRP0:1/8)
#define LIN_SPEED_9600_LCKS  1    //fb(LBRP0:1/2)
#define LIN_SPEED_10417_LCKS  3    //fd(LBRP1:1/2)
#define LIN_SPEED_19200_LCKS  0    //fa(LBRP0:1/1)

//動作モード
#define LIN_OPERATION_TIMER  0    //タイマ動作
#define LIN_OPERATION_UART  1    //UARTコマンド入力に応じて動作

/*-----------------------------------------------------------------------
    構造体
-----------------------------------------------------------------------*/
typedef struct{
    unsigned char id;
    unsigned char data[8];
    unsigned char sum;
    unsigned char size;
} lin_message;

（後略）
```

#define SCI_DEBUG

定義時端末表示がデバッグモードとなります。

```
--
LIN header/response data send : LIN-ID=0x30 data=0x0000000401800000
[intr(ch2)=recv][intr(ch0)=send][intr(ch0)=send][intr(ch2)=recv]
LIN data receive(ch2) : LIN-ID=0x30 data=0x0000000401800000 sum=0x89

--
LIN header send : LIN-ID=0x31
[intr(ch2)=recv][intr(ch0)=send][intr(ch2)=send][intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x31 data=0x0000000901800000 sum=0xC3
```

MASTER ヘッダ・レスポンス送信

MASTER ヘッダ送信
（SLAVE レスポンス送信）

赤字の行が表示される様になり、処理中にどの種類の割り込みが入っているかが判る様になります。
MASTER ヘッダ・レスポンス送信の場合は、MASTER 側がヘッダを送出した時点で、
・ch2(SLAVE)の受信割り込み
・ch0(MASTER)の送信割り込み

が、ほぼ同時に発生し、MASTER がレスポンス送信を終えた時点で、
・ch0(MASTER)の送信割り込み
・ch2(SLAVE)の受信割り込み
が発生しています。

－意図的にエラーを起こさせた場合－

MASTER ヘッダ・レスポンス送信

```
LIN header/response data send : LIN-ID=0x30 data=0x0000000201800000
[intr(ch2)=recv][intr(ch0)=send][intr(ch2)=stat LST2=0x88 LEST2=0x80][intr(ch0)=send]
```

例えば、ch2(SLAVE)側を受信待機せずに、ch0(MASTER)側からデータ送信した場合、ch2(SLAVE)側はデータ受信してない（LIN data receive(ch2)の表示が出ない）ものの、
[intr(ch2)=recv]
となっているので、ch2 側には受信割り込みが発生している事が判ります。その後、
[intr(ch2)=stat LST2=0x88 LEST2=0x80]
ch2(SLAVE)側はステータス割り込みが入り、その際のエラーステータス（LEST2=0x80, レスポンス準備エラー）が判ります。

－LIN のケーブルを抜いた場合－

MASTER ヘッダ・レスポンス送信

```
LIN header/response data send : LIN-ID=0x30 data=0x0000000101800000
[intr(ch0)=send][intr(ch0)=send]
```

なお、ch0 と ch2 をつなぐケーブルを抜いて、ch0(MASTER)側からデータ送信した場合、表示は上記の様になります。デバッグ表示 OFF の時の表示は「意図的にエラーを起こさせた場合」と同一ですが、デバッグ表示を行わせると、違いが判る様になります。（この例では、ch2 側は全く反応していません。）

本プログラムでは、チェックサムエラーとなった場合はデータを受信バッファに格納せずに捨てる仕様としています。そのような場合でも、受信割り込み（ヘッダ受信時）やステータス割り込み（エラー発生時）は発生するはずなので、意図しない動作となった場合、デバッグ表示を有効にすると、エラー時のエラーステータスレジスタの値等の確認が可能となります。

#define LIN_RESPONSE_DATA_SIZE 8

レスポンスデータのデータサイズ（バイト数）
LIN は、パケット毎にデータサイズを変更するのに適していないプロトコルですので、本定数でデータサイズを固定化しています。

#define LIN_RECV_BUF_SIZE 16

受信バッファ(lin_message 構造体）の配列数
sizeof(lin_message)×LIN_CH_NUM(3)×LIN_RECV_BUF_SIZE　のメモリを消費します。

※現状のプログラムでは、LIN-ch0, LIN-ch2 を使用するプログラムになっていますが、未使用の LIN-ch1 用の受信バッファも確保しています（g_lin_recv_buf[**LIN_CH_NUM**][LIN_RECV_BUF_SIZE]: g_lin_recv_buf の 1 つ目の[ ]の要素は LIN-ch 番号）（メモリ消費量を抑えたい場合は、g_lin_recv_buf の 1 つ目の[ ]の要素を LIN-ch 番号ではなく、使用 ch 毎に番号を割り振る等変更してください）

※デフォルトでは LIN_RECV_BUF_SIZE=16 としていますが、保持できるデータ数は 15 となります
（未読み出しの状態で、16 個目のデータを受信した場合、1 つ目のデータは（この時点では上書きはされないものの）参照不可となります）
→受信バッファはリングバッファの構成を取っており、2 つのインデックス g_lin_recv_buf_index1（書き込みインデックス）と g_lin_recv_buf_index2（読み出し済みインデックス）から構成されています。g_lin_recv_buf_index1 == g_lin_recv_buf_index2 の場合、格納データが無いと判断しているアルゴリズムのため、実際に使用可能なデータ数は LIN_RECV_BUF_SIZE-1 となっています。
※バッファメモリをフルに活用したい場合は、index1 と index2 の運用を見直してみてください

// #define LIN_USE_SEND_CALLBACK_FUNCTION
// #define LIN_USE_RECEIVE_CALLBACK_FUNCTION
// #define LIN_USE_STATUS_CALLBACK_FUNCTION

割り込みコールバック関数の有効化。

LIN_USE_SEND_CALLBACK_FUNCTION を定義した場合、送信割り込み時に、
lin***n***_interrupt_send_callback(n=0~2)関数が呼ばれます。
LIN_USE_RECEIVE_CALLBACK_FUNCTION を定義した場合、受信割り込み時に、
lin***n***_interrupt_receive_callback(n=0~2)関数が呼ばれます。
LIN_USE_STATUS_CALLBACK_FUNCTION を定義した場合、ステータス割り込み時に、
lin***n***_interrupt_status_callback(n=0~2)関数が呼ばれます。
（n: 0~2 は、LIN-ch 毎）
割り込み時にユーザ処理を追加したい場合、有効化して、コールバック関数の中身を記載してください。

```
#define LIN_SPEED_LBRP0 104
#define LIN_SPEED_LBRP1 96
```

通信速度は、lin_operation.h で 2400, 9600, 10417, 19200bps を選択できるようになっています。

上記値は、LBRP0/1（速度設定レジスタ）に設定している値です。予め定義されている、4 種類の速度以外の速度値とする場合は、レジスタ設定値を計算して設定してください。

```
#define LIN_SPEED_2400_LCKS 2
#define LIN_SPEED_9600_LCKS 1
#define LIN_SPEED_10417_LCKS 3
#define LIN_SPEED_19200_LCKS 0
```

2400bps の場合は「LBRP0 の 1/8」(=fc)、9600bps の場合は「LBRP0 の 1/2」(=fb)、10417bps の場合は「LBRP1 の 1/2」(=fd)、19200bps の場合は「LBRP0 の 1/1」(=fa)をベースクロックとして選択する設定です。

LIN の通信速度（1bit の時間）は、

1/32MHz（fCLK） ×分周比(LCKS) ×プリスケーラ設定値(LBRP)×16 サンプリング

で決まります。1bit 時間は、

・LBRP0 側を使用
19200bps: 1/32MHz ×1(fa) × 104 × 16 = 52[us]
9600bps: 1/32MHz ×2(fb) × 104 × 16 = 104[us]
2400bps: 1/32MHz ×8(fc) × 104 × 16 = 416[us]

・LBRP1 側を使用
10417bps: 1/32MHz ×2(fd) × 96 × 16 = 96[us]

上記設定値となります。

任意の速度とする場合は、LBRP0 または LBRP1 レジスタ値と、LCKS の値（どちらのプリスケーラを使用するかと分周比）を調整してください。

本節で説明していない定数は、プログラム動作で使用されている定数値で基本的には変更の必要はありません。

lin_message 構造体は、

・LIN-ID（1 バイト）

・データ（最大 8 バイト）

・チェックサム（拡張チェックサム、1 バイト）

・データサイズ（1~8 の値、1 バイト）

で構成される、LIN の受信メッセージを扱う構造体です。

5.1.2. lin_operation.h の定数定義

lin_operation.h

```
*---------------------------------------------------------------------------
    定数定義（ユーザ設定箇所）
---------------------------------------------------------------------------*/
//使用CH
#define LIN_USE_CH0    //LIN-ch0を使用する
//#define LIN_USE_CH1    //LIN-ch1を使用する
#define LIN_USE_CH2    //LIN-ch2を使用する

//速度設定（いずれかを選択）
//#define LIN_SPEED  LIN_SPEED_2400    //2400bps
#define LIN_SPEED  LIN_SPEED_9600    //9600bps
//#define LIN_SPEED  LIN_SPEED_10417    //10417bps
//#define LIN_SPEED  LIN_SPEED_19200    //19200bps

//割り込み優先度
#define LIN_INTERRUPT_PRIORITY  1    //0:最高優先度～3:最低優先度

//LIN ID
#define LIN_MASTER_ID  0x30  //MASTER動作時に設定するID
#define LIN_SLAVE_ID  0x31  //SLAVE動作時に設定するID

//SLAVEレスポンスを要求するID（MASTER動作時のみ使用）
#define LIN_SLAVE_RESPONSE_START_ID  0x31    //SLAVEレスポンス送信を要求する要求先 0x31,
0x32, 0x33, 0x34 ....

//レスポンス要求するSLAVEデバイス数（MASTER動作時のみ使用）
#define LIN_SLAVE_DEVICE_NUM  4    //0x31 ~ 0x34 に対してレスポンス要求する

//動作モード（どちらかを選択）
//#define LIN_OPERATION_MODE  LIN_OPERATION_TIMER    //下記のインターバル時間毎にヘッダ送信
#define LIN_OPERATION_MODE  LIN_OPERATION_UART    //UARTからのコマンドでヘッダ送信

//動作インターバル時間[x50ms]（MASTER動作時のみ使用）
#define LIN_INTERVAL_TIME  20    //1秒毎にヘッダを出す
/*
  MASTERの場合、
  50×LIN_INTERVAL_TIME[ms]毎に、
  (1)ID=0x30（マスタレスポンス送信）
  (2)ID=0x31（スレーブレスポンス送信要求）
  (3)ID=0x32（スレーブレスポンス送信要求）
  (4)ID=0x33（スレーブレスポンス送信要求）
  (5)ID=0x34（スレーブレスポンス送信要求）
  を繰り返す（(LIN_SLAVE_DEVICE_NUM+1)×50×LIN_INTERVAL_TIME[ms]で1周）
*/
```

```
#define LIN_USE_CH0
// #define LIN_USE_CH1
#define LIN_USE_CH2
```

使用する LIN-ch を有効にしてください。使用しない LIN-ch はコメントアウトして、定数を未定義としてください。

```
// #define LIN_SPEED  LIN_SPEED_2400
#define LIN_SPEED  LIN_SPEED_9600
// #define LIN_SPEED  LIN_SPEED_10417
// #define LIN_SPEED  LIN_SPEED_19200
```

速度設定です。いずれか 1 行のみ有効化（コメントアウトを外す）してください。予め定義されている 4 種類以外の速度を使用する場合は、lin.h 側含め修正が必要です。

```
#define LIN_INTERRUPT_PRIORITY  1
```

LIN の割り込み優先度の設定です。0（最高優先度）～3（最低優先度）のいずれかの値としてください。

LIN の割り込みは、「送信割り込み」「受信割り込み」「ステータス割り込み」の 3 種類ありますが、本プログラムでは 3 種類の割り込みに上記で設定した値を適用します。（3 種類の割り込みの優先度を変えたい場合は、lin.c の修正が必要です。）

※割り込み優先度を 0 にすると、多重割り込み（受信割り込み処理の最中に送信割り込みが入る等）が掛る状態となります（→現状のプログラムでは特に問題はないと思いますが、割り込み処理内にユーザの処理を追加する場合はご留意ください。）

```
#define LIN_MASTER_ID  0x30
#define LIN_SLAVE_ID  0x31
```

サンプルプログラムでは、ch0 を MASTER, ID=0x30。ch2 を SLAVE, ID=0x31 に設定しています。LIN の ID はこの定数値を使用しています。

```
#define LIN_SLAVE_RESPONSE_START_ID  0x31
```

キーボードからのコマンドで動作するモードの場合の、コマンド 1~9。タイマ動作するモードの SLAVE に対しヘッダ送信を行う ID の開始値です。コマンド 1:ID=0x31、コマンド 2:ID=0x32。タイマ動作の場合 ID=0x31 から ID をインクリメントさせながら、ヘッダ送信します。

#define LIN_SLAVE_DEVICE_NUM 4

タイマ動作するモードの SLAVE に対しヘッダ送信を行う ID の数です。(コマンドモードの時は未使用です。)4 の場合、ID=0x31~ID=0x34 の 4 つの SLAVE に対してヘッダ送信を行います。

// #define LIN_OPERATION_MODE LIN_OPERATION_TIMER
#define LIN_OPERATION_MODE LIN_OPERATION_UART

サンプルプログラムの動作モードを決める定数です。どちらか一方を有効化してください。
LIN_OPERATION_UART は、UART 経由でコマンド入力で動作(デフォルト)、LIN_OPERATION_TIMER は、タイマモードで一定時間毎に、MASTER ヘッダ・レスポンス送信、MASTER ヘッダ送信…、を繰り返します。

#define LIN_INTERVAL_TIME 20

タイマ動作モード時のヘッダの送信間隔です。50ms(基本周期)の何倍かを指定する値です。(20 の場合、50ms×20 の 1 秒間隔でのヘッダ送信。)

9600bps 設定の場合、8 バイトのレスポンスデータ送信で一連のパケット(ヘッダ＋レスポンス)の時間は、約 13ms になります。LIN バスのケーブル長が長い場合(MASTER と SLAVE の距離が離れている場合)や、SLAVE が RL78/F15 ではなく別デバイスの場合等、13ms という時間は多少長くなる事はあり得ます。9600bps の場合は、この値を 1 に設定しても問題ないと思います。

2400bps 設定の場合は、8 バイトのレスポンスデータ送信で 53ms 程度となますので、最低でも 2 以上の値とする必要があります。

5.2. lin.c 内で定義している関数

lin_init

概要:初期化関数

宣言:

int lin_init(unsigned char ch, unsigned char mode, unsigned char id)

説明:

・ch の有効化
・割り込み設定
・通信速度、通信条件

の設定を行います

引数:

ch: LIN の ch 番号(0~2)

mode: MASTER/SLAVE 区分　LIM_MASTER(0)、または LIN_SLAVE(1)

id: LIN-ID(0x0~0x3f)

戻り値:

0: 正常終了

-1: 引数エラー

補足:

id は MASTER モードの時未使用です。SLAVE 時は、本関数で設定した値と受信したヘッダに含まれる ID 値が一致した場合、レスポンス送信を行います

lin_response_data_set

概要:レスポンスデータ設定関数

宣言:

int lin_response_data_set(unsigned char ch, unsigned char *data, unsigned char size)

説明:

・SLAVE 側での送信データの設定

を行います

引数:

ch: LIN の ch 番号(0~2)

*data: レスポンスデータ（1~8 バイト）

size: データバイト数(1~8)

戻り値:

0: 正常終了

-1: 引数エラー

補足:

SLAVE に設定した ch で実行するコマンドです

SLAVE 側の ch がレスポンス送信を行う予定がない場合（受信のみを行うケース）でも、本コマンドで

受信データサイズの設定が必要です（その場合は*data はダミーで構いません）

SLAVE 側で自分自身の ID を含むヘッダを受信した場合、本コマンドで設定したレスポンスデータを送信します

lin_master_header_send

概要:ヘッダ送信関数

宣言:

int lin_master_header_send(unsigned char ch, unsigned char id, unsigned char size)

説明:

・MASTER からヘッダの送信

を行います

引数:

ch: LIN の ch 番号(0~2)

id: ヘッダに含める LIN-ID(0x0~0x3f)

size: 受信予定のデータバイト数(1~8)

戻り値:

0: 正常終了

1: 送信中（指定した ch で送信中）

-1: 引数エラー

補足:

本コマンドで設定した size と実際に受信したバイト数が合わない場合チェックサムエラーとなり、受信データは

捨てられます（SLAVE 側で実行した lin_response_data_set()で指定した size と合わせる様にしてください）

実際に LIN バスに流れる ID は、指定した id 値にパリティが付与されたもの(PID)となります

本コマンドで指定する id は、パリティを付与しない 0~0x3f の ID を指定してください

lin_master_header_response_send

概要:ヘッダ・レスポンス送信関数

宣言:

int lin_master_header_response_send(unsigned char ch, unsigned char id, unsigned char *data, unsigned char size)

説明:

・MASTER からヘッダ・レスポンス送信の送信

を行います

引数:

ch: LIN の ch 番号(0~2)

id: ヘッダに含める LIN-ID(0x0~0x3f)（MASTER デバイスの ID）

*data: 送信データ（1~8 バイト）

size: 送信のデータバイト数(1~8)

戻り値:

0: 正常終了

1: 送信中（指定した ch で送信中）

-1: 引数エラー

補足:

実際に LIN バスに流れる ID は、指定した id 値にパリティが付与されたもの(PID)となります

本コマンドで指定する id は、パリティを付与しない 0~0x3f の ID を指定してください

lin_slave_header_receive

概要:ヘッダ受信待機関数

宣言:

int lin_slave_header_receive(unsigned char ch)

説明:

・SLAVE 側のヘッダ受信待機状態への遷移

を行います

引数:

ch: LIN の ch 番号(0~2)

戻り値:

0: 正常終了

-1: 引数エラー

補足:

SLAVE デバイスの場合、lin_init()で初期化後、本関数でヘッダの受信待機状態にしてください

本関数実行前はヘッダの受信準備が整っていない状態です

lin_init() → lin_response_data_set() → 本関数　の順番で実行されることを想定しています

初期化後 lin_response_data_set()実行前にヘッダを受信した場合、意図しないデータを返送する事を抑止するために、本関数を設けています（意図しないデータを返送するよりエラーとして処理する）

lin_response_data_set()実行前に本コマンドを実行しても問題ありません

※lin_init()では、レスポンスデータは 0x00×LIN_RESPONSE_DATA_SIZE に設定されていますので、lin_response_data_set()未実行の場合は、受信したヘッダに SLAVE 自身の ID が含まれていた場合 0x00×LIN_RESPONSE_DATA_SIZE のレスポンスデータを送信します

lin_status_clear

概要:エラーステータスクリア関数

宣言:

int lin_status_clear(unsigned char ch)

説明:

・エラーステータスのクリア

を行います

引数:

ch: LIN の ch 番号(0~2)

戻り値:

0: 正常終了

-1: 引数エラー

補足:

ステータス・レジスタ(LSTn)

エラー・ステータス・レジスタ(LESTn)

のクリアを行います

lin_read_data

概要:受信データ読み出し関数

宣言:

int lin_read_data(unsigned char ch, lin_message *msg)

説明:

・受信バッファに格納されているデータの読み出し

を行います

引数:

ch: LIN の ch 番号(0~2)

*msg: lin_message 構造体

lin_message 構造体

unsigned char id;	受信したデータのID
unsigned char data[8];	受信データ
unsigned char sum;	チェックサム値
unsigned char size;	データバイト数

戻り値:

0: 正常終了(データの読み出し完了)

1: 読み出し前に上書きされたデータあり(データの読み出し完了)

2: 未読み出しのデータなし

-1: 引数エラー

補足:

データを受信したタイミングで、受信割り込み関数内で、受信バッファ(g_lin_recv_buf)にデータが格納されます
本関数は、受信バッファに格納されているデータの読み出しを行います
未読み出しの(LIN_RECV_BUF_SIZE-1)数のデータが溜まっている状態で次のデータを受信すると、一番古いデータが上書きさます その状態で本関数を呼び出すと、格納されているデータで一番古いデータが読み出され、関数の戻り値は失われたデータがある事を示す 1 を返します

lin_read_data_size

概要:受信データ数読み出し関数

宣言:

lin_read_data_size(unsigned char ch)

説明:

・受信バッファに格納されているメッセージ数の情報
を返します

引数:

ch: LIN の ch 番号(0~2)

戻り値:

0: 未読み出しのデータなし

>0: 受信バッファに格納されているメッセージ数

-1: 引数エラー

lin_read_buf_clear

概要:受信データクリア関数

宣言:

int lin_read_buf_clear(unsigned char ch)

説明:

・受信バッファに格納されているデータをクリア(読み出し済みにセット)
を行います

引数:

ch: LIN の ch 番号(0~2)

戻り値:

0: 正常終了

-1: 引数エラー

－ローカル関数（外部からの呼び出し不可）－

lin_port_init

概要：ポート初期化関数

宣言：

void lin_port_init(unsigned char ch)

説明：

・LIN 通信で使用する I/O ポートの初期化

を行います

引数：

ch: LIN の ch 番号(0~2)

戻り値： なし

補足：

プログラムでは、下記の端子を使用する設定としています

別な端子を使用する場合や EN（LIN ドライバイネーブル端子）を使用しない場合は、本関数内の記載を適宜変更ください

	ch0	ch1	ch2	備考
LTXDn	P13	P10	P155	
LRXDn	P14	P11	P154	
EN	P53		P156	H 出力設定

5.3. 割り込みコールバック関数

割り込みコールバック関数は、ユーザ記載のコード内で定義してください。

lin*n*_interrupt_send_callback (n=0~2)

概要:送信割り込みコールバック関数

宣言:

void lin0_interrupt_send_callback(void)
void lin1_interrupt_send_callback(void)
void lin2_interrupt_send_callback(void)

説明:

・送信割り込み処理の終わりでコールされます

引数: なし
戻り値: なし
補足:

LIN-ch 毎に別関数となっています
使用時は、#define LIN_USE_SEND_CALLBACK_FUNCTION を有効化してください

lin*n*_interrupt_receive_callback (n=0~2)

概要:受信割り込みコールバック関数

宣言:

void lin0_interrupt_receive_callback(void)
void lin1_interrupt_receive_callback(void)
void lin2_interrupt_receive_callback(void)

説明:

・受信割り込み処理の終わりでコールされます

引数: なし
戻り値: なし
補足:

LIN-ch 毎に別関数となっています
使用時は、#define LIN_USE_RECEIVE_CALLBACK_FUNCTION を有効化してください

lin***n***_interrupt_status_callback (n=0~2)

概要:ステータス割り込みコールバック関数

宣言:

void lin0_interrupt_status_callback(void)

void lin1_interrupt_status_callback(void)

void lin2_interrupt_status_callback(void)

説明:

・ステータス割り込み処理の終わりでコールされます

引数: なし

戻り値: なし

補足:

LIN-ch 毎に別関数となっています

使用時は、#define LIN_USE_STATUS_CALLBACK_FUNCTION を有効化してください

5.4. 使用しているグローバル変数

g_lin_id

用途：ch 毎の LIN-ID を保持

宣言：

unsigned char g_lin_id[LIN_CH_NUM];

説明：

SLAVE 動作時、ヘッダに含まれる ID と自局の ID（＝g_lin_Id）が一致しているかの判定に使用します

g_lin_mode

用途：ch 毎の動作モード(MASTER/SLAVE)を保持

宣言：

unsigned short g_lin_mode[LIN_CH_NUM];

説明：

MASTER 動作時、LIN_MASTER(0)
SLAVE 動作時、LIN_SLAVE(1)
受信割り込み関数内で動作モードで処理を分ける際などに使用します

g_lin_tx_flag

用途：送信状態を表すフラグ変数

宣言：

unsigned short g_lin_tx_flag[LIN_CH_NUM];

説明：

送信中 LIN_TX_FLAG(0x1)
MASTER 動作時、レスポンスデータ送信予定 LIN_RESPONSE_FLAG(0x2)
の OR を取ります ※詳しくは 6.5 節参照

g_lin_rx_flag

用途：受信状態を表すフラグ変数

宣言：

unsigned short g_lin_rx_flag[LIN_CH_NUM];

説明：

レスポンスデータ受信待機中 LIN_RX_READY (0x1)、それ以外 0 を取ります ※詳しくは 6.5 節参照

g_lin_recv_buf

用途：受信バッファ

宣言：

lin_message g_lin_recv_buf[LIN_CH_NUM][LIN_RECV_BUF_SIZE];

説明：

受信割り込み時、有効なレスポンスデータを受信すると本変数にデータを格納します

g_lin_send_buf

用途：送信バッファ（SLAVE 向け）

宣言：

unsigned char g_lin_send_buf[LIN_CH_NUM][8];

説明：

SLAVE レスポンス送信の際、本バッファに格納されているデータを送信します

g_lin_recv_buf_index1

用途：受信バッファの書き込みインデックス

宣言：

unsigned short g_lin_recv_buf_index1[LIN_CH_NUM];

説明:

受信バッファにデータを格納する際、本インデックスが示す変数にデータが格納されます

g_lin_recv_buf_index2

用途:受信バッファの読み出しインデックス

宣言:

unsigned short g_lin_recv_buf_index2[LIN_CH_NUM];

説明:

受信バッファからデータを読み出す際、本インデックスが示す変数のデータを読み出します
読み出し時、g_lin_recv_buf_index2 == g_lin_recv_buf_index1 の場合、未読データがない事を示します

g_lin_recv_buf_override

用途:受信バッファの上書きフラグ

宣言:

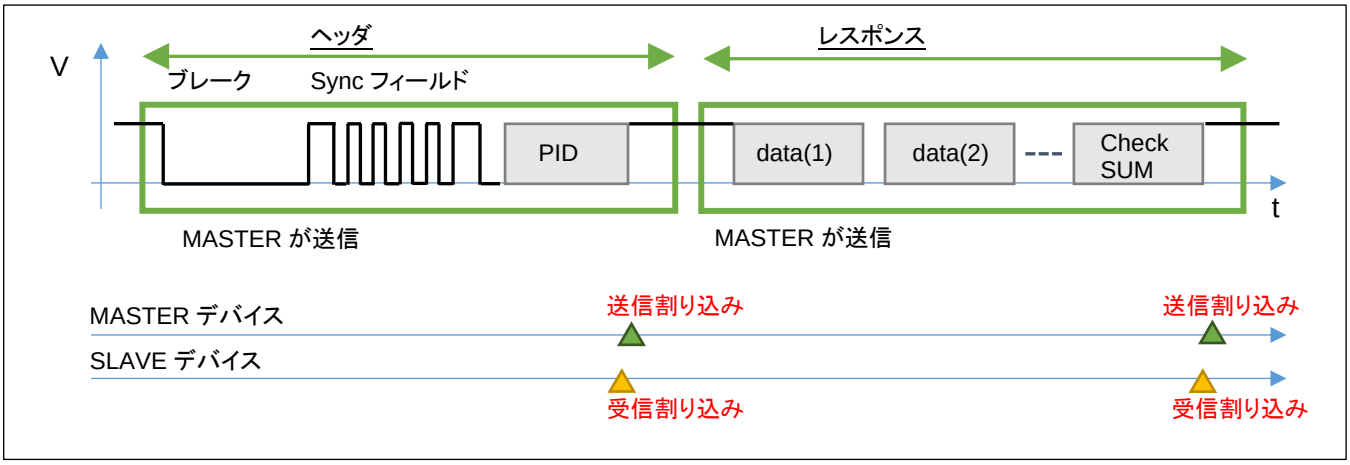
unsigned short g_lin_recv_buf_override[LIN_CH_NUM];

説明:

受信バッファ書き込み時にバッファに空きがない（未読データで埋まっている）場合、格納されている中で最も古いデータから上書きされます
データの上書きが生じた際、本変数が 1 となります
（データの読み出し時、データの上書きが生じた事を表す戻り値を返すと共に、本変数は 0 クリアされます）

6. 割り込み処理

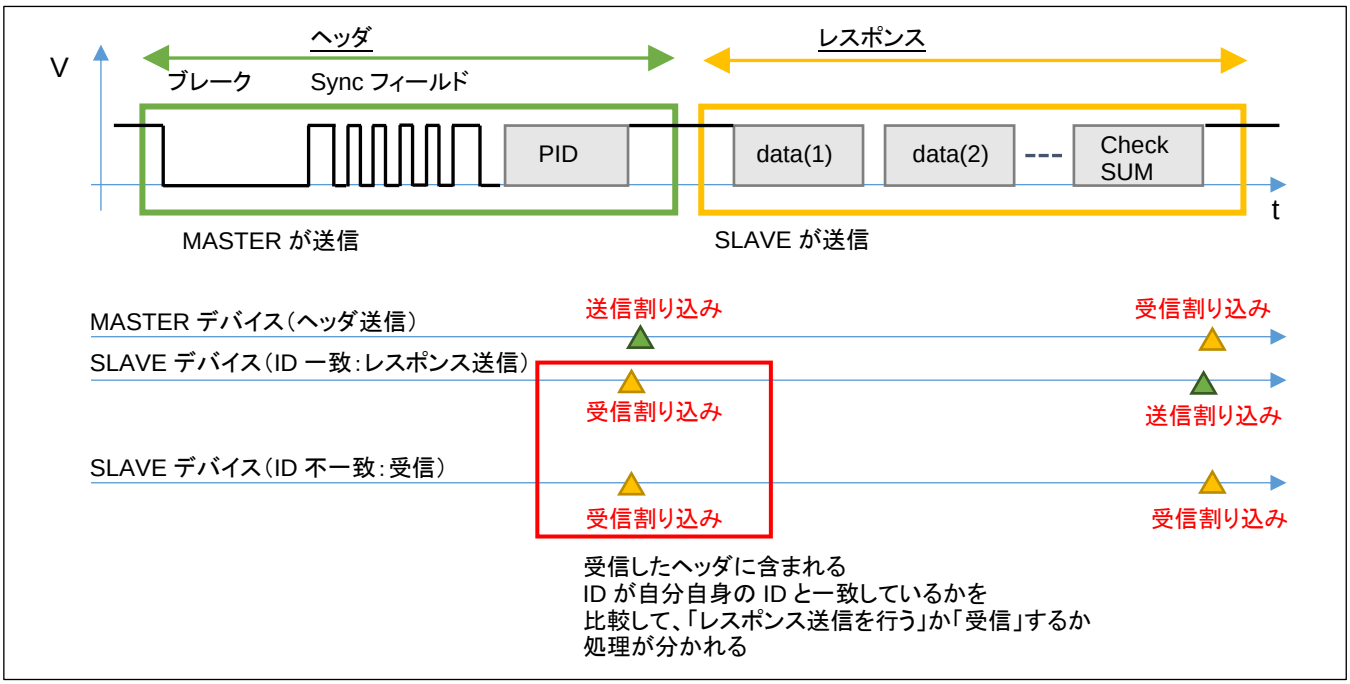
6.1. MASTER レスポンス送信動作



MASTER レスポンス送信の場合は、送信(MASTER)側で、送信割り込みが「ヘッダ送信完了」のタイミングと「レスポンス送信完了」のタイミングで 2 回発生します。

SLAVE 側は、同様に受信割り込みが「ヘッダ受信完了」と「レスポンス受信完了」のタイミングで 2 回発生します。

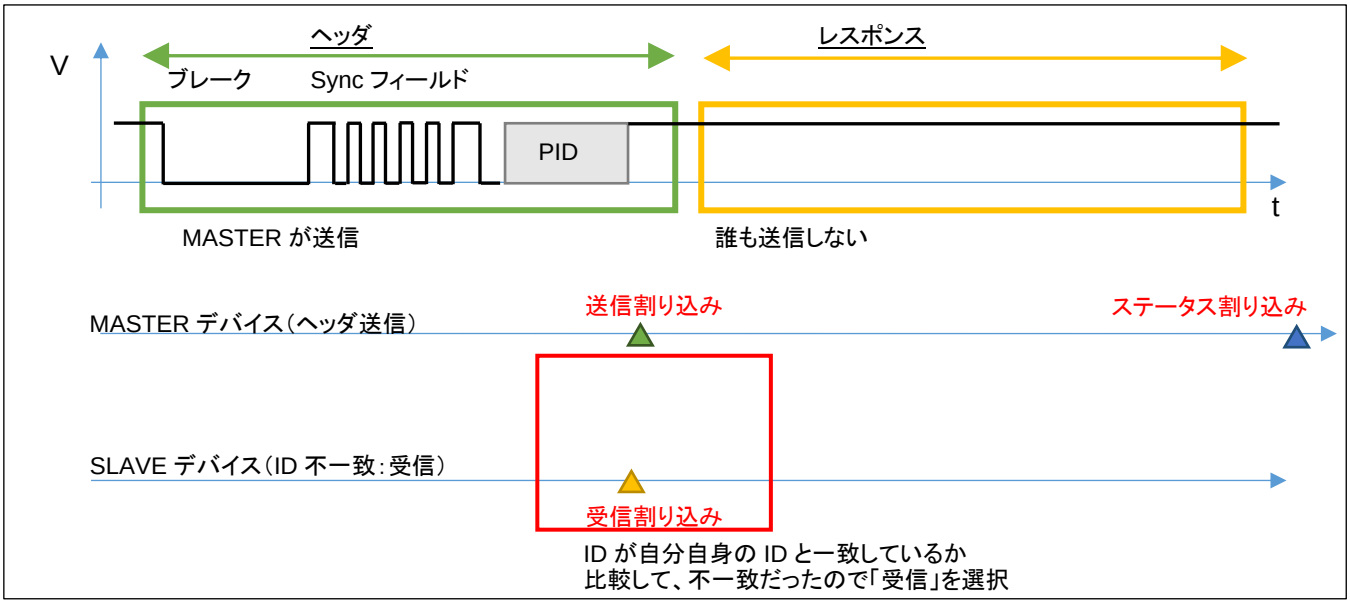
6.2. MASTER ヘッダ送信動作



MASTER ヘッダ送信の際には、上記の様な割り込みの流れとなります。

SLAVE デバイスは、最初の受信割り込みで「レスポンス送信を行う」かどうかを判断する事になります。

6.3. MASTER ヘッダ送信動作（応答する SLAVE なし）

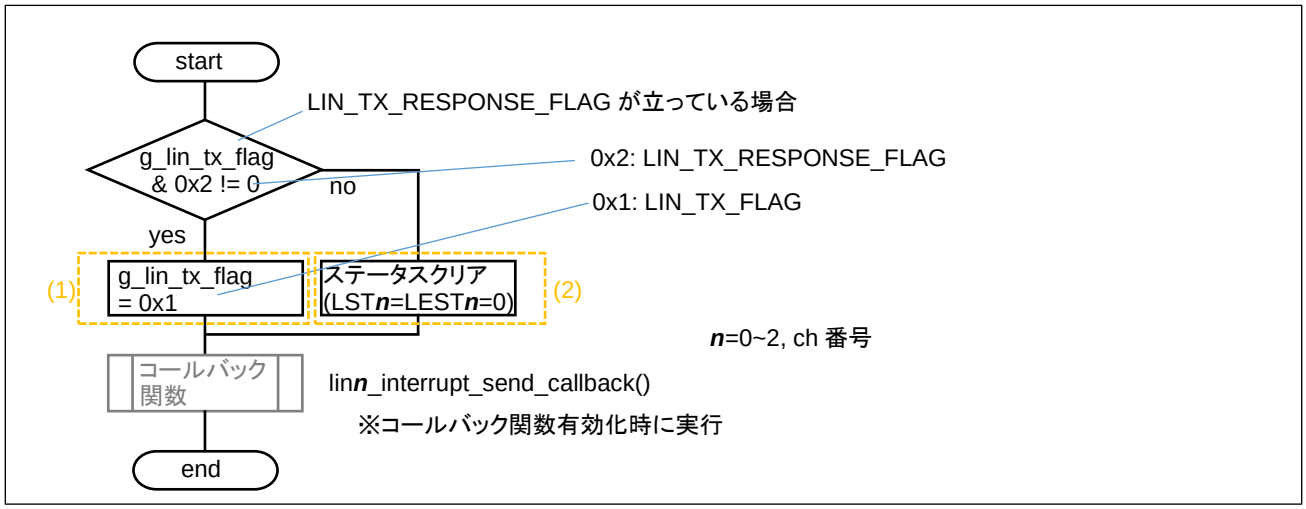


MASTER ヘッダ送信で応答する SLAVE が居ない場合、送信側ではステータス割り込みが入ります。
（エラーステータス LESTx=0x04, タイムアウトエラーとなります）
※ステータス割り込みでエラーステータスをクリアするので、レスポンスがない場合でも動作の継続が可能です

6.4. 割り込み関数

6.4.1. 送信割り込み

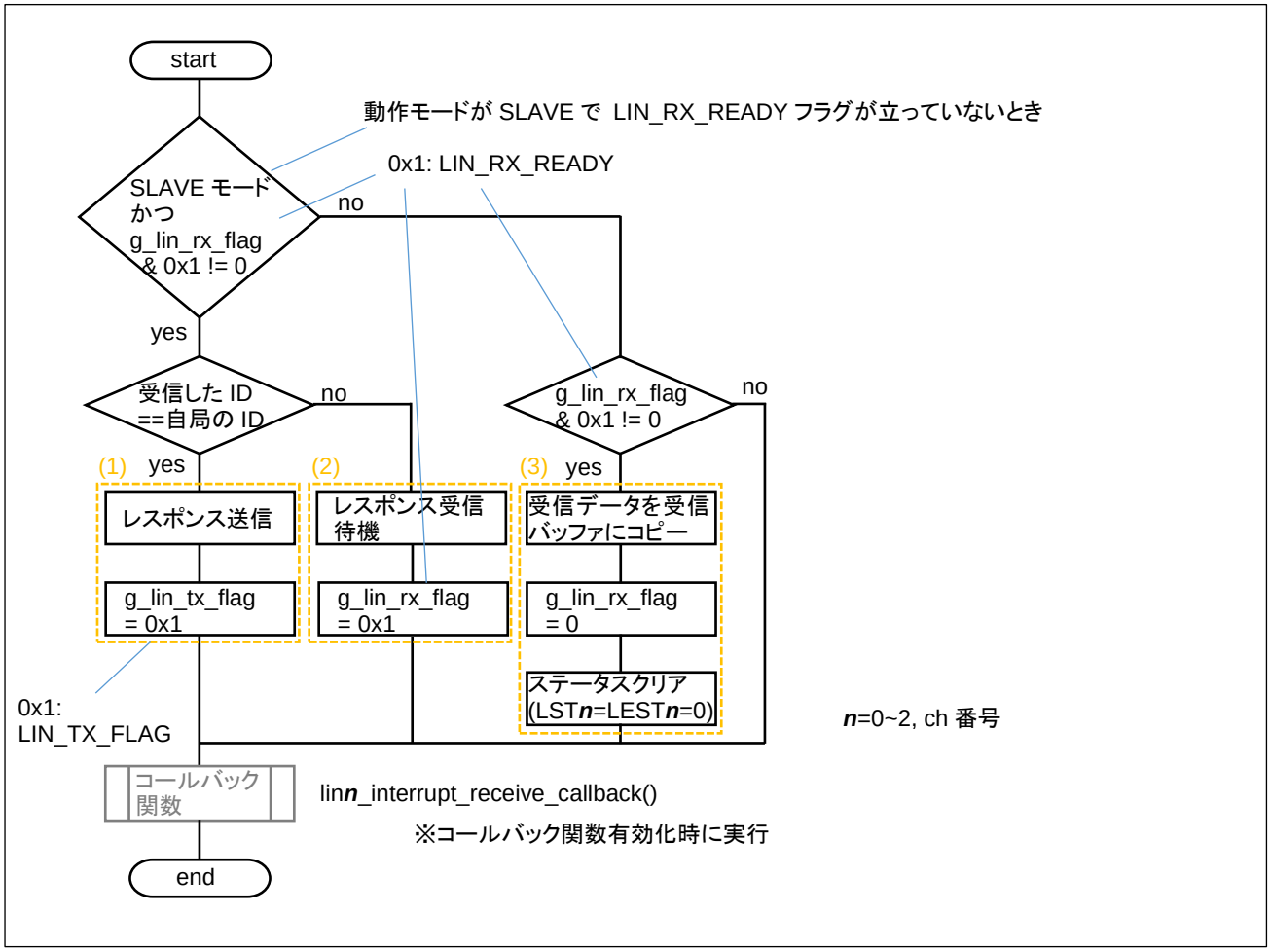
lin.c 内 lin***n***_interrupt_send()（n=0~2, ch 番号）



・MASTER レスポンス送信の 1 回目（ヘッダ送信完了のタイミング）は(1)側
・MASTER レスポンス送信の 2 回目（レスポンス送信完了のタイミング）は(2)側
・SLAVE レスポンス送信の（レスポンス送信完了のタイミング）は(2)側
が実行されます。

6.4.2. 受信割り込み

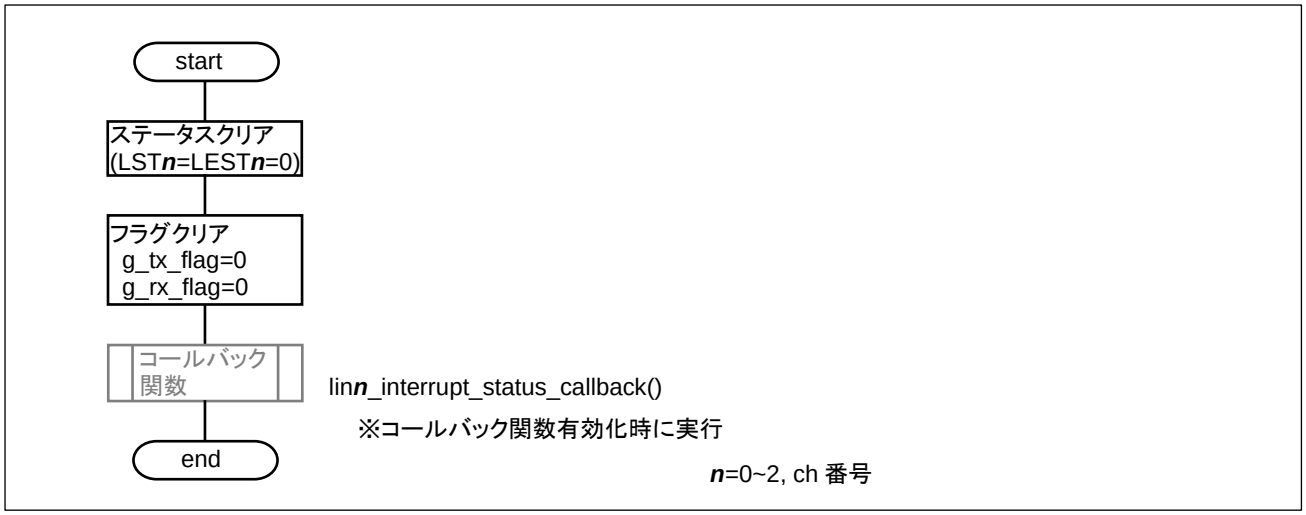
lin.c 内 lin***n***_interrupt_receive()（n=0~2, ch 番号）



・MASTER ヘッダ送信時のレスポンス受信割り込みは(3)
・SLAVE のヘッダ受信時にヘッダに含まれる ID が自局の ID と一致したときは(1)
・SLAVE のヘッダ受信時にヘッダに含まれる ID が自局の ID と一致しなかったときは(2)
・SLAVE のレスポンス受信時は(3)
が実行されます。

6.4.3. ステータス割り込み

lin.c 内 lin***n***_interrupt_status()（n=0~2, ch 番号）



ステータス割り込みでは、ステータスレジスタと、フラグ変数をクリアします。

6.5. フラグ変数に関して

割り込みで処理内容を変えるため、プログラムでは 2 種類のフラグ変数

g_lin_tx_flag[ch]
g_lin_rx_flag[ch]

を使用しています。フラグ変数は、LIN-ch 毎に独立です。

g_lin_tx_flag は
bit0: LIN_TX_FLAG(0x1) 送信中を示すフラグ
bit1: LIN_TX_RESPONSE_FLAG(0x2) MASTER レスポンス送信を示すフラグ

g_lin_rx_flag は
bit0: LIN_RX_READY(0x1) レスポンス受信を待機するフラグ

を持ちます。

6.5.1. MASTER レスポンス送信の際の MASTER 側

(1)lin_master_header_response_send()内

[g_lin_tx_flag の、bit0(LIN_TX_FLAG)が立っている場合は、送信中としてエラーを返す]

・g_lin_tx_flag の、bit0(LIN_TX_FLAG)と bit1(LIN_TX_RESPONSE_FLAG)フラグを立てる

(2)送信割り込み関数内(1)[lin***n***_interrupt_send()]

[g_lin_tx_flag の、bit1(LIN_TX_RESPONSE_FLAG)が立っている場合]
→2 回入る送信割り込みの 1 回目、ヘッダ送信が終わったタイミング
・bit1(LIN_TX_RESPONSE_FLAG)フラグを落とす

(2)送信割り込み関数内(2)[lin***n***_interrupt_send()]

[g_lin_tx_flag の、bit1(LIN_TX_RESPONSE_FLAG)が立っていない場合]
→2 回入る送信割り込みの 2 回目、レスポンス送信が終わったタイミング
・g_lin_tx_flag の bit0(LIN_TX_FLAG)フラグを落とす

6.5.2. MASTER レスポンス送信の際の SLAVE 側

(1)受信割り込み関数内(1)[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っていない場合]
→2 回入る受信割り込みの 1 回目、ヘッダ受信のタイミングでの割り込み
→ヘッダに含まれる ID でレスポンス送信を行うか、レスポンス受信を行うか判断するが MASTER レスポンス送信の場合は必ず IDが不一致となるはずである
・レスポンス受信を行うので、g_lin_rx_flag の bit0(LIN_RX_READY)フラグを立てる

(2)受信割り込み関数内(2)[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っている場合]
→2 回入る受信割り込みの 2 回目、レスポンス受信のタイミングでの割り込み
・データ受信を行う（受信データを受信バッファにコピー）
・レスポンス受信が完了したので、g_lin_rx_flag の bit0(LIN_RX_READY)フラグを落とす

6.5.3. MASTER ヘッダ送信の際の MASTER 側

(1)lin_master_header_send()内

[g_lin_tx_flag の、b0(LIN_TX_FLAG)が立っている場合は、送信中としてエラーを返す]

・g_lin_tx_flag の、b0(LIN_TX_FLAG)フラグを立てる
・レスポンス受信を行うので、g_lin_rx_flag の bit0(LIN_RX_READY)フラグを立てる

(2)送信割り込み関数内[lin***n***_interrupt_send()]

[g_lin_tx_flag の、bit1(LIN_TX_RESPONSE_FLAG)が立っていない場合]
→ヘッダ送信が終わったタイミング（レスポンス送信の予定なし）
・g_lin_tx_flag の bit0(LIN_TX_FLAG)フラグを落とす（以降送信の予定がないので、送信中フラグを落とす）

(2)受信割り込み関数内[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っている場合]
→レスポンス受信のタイミングでの割り込み
・データ受信を行う（受信データを受信バッファにコピー）
・レスポンス受信が完了したので、g_lin_rx_flag の bit0(LIN_RX_READY)（受信待機）フラグを落とす

6.5.4. MASTER ヘッダ送信の際、レスポンス送信を行う SLAVE 側

(1)受信割り込み関数内[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っていない場合]
→ヘッダ受信のタイミングでの割り込み
→ヘッダに含まれる ID でレスポンス送信を行うか、レスポンス受信を行うか判断するが ID が一致してレスポンス送信を行うケース
・レスポンス送信を行うので、g_lin_tx_flag の bit0(LIN_TX_FLAG)を立てる

(2)送信割り込み関数内[lin***n***_interrupt_send()]

→レスポンス送信が終了したタイミング
・g_lin_tx_flag の bit0(LIN_TX_FLAG)フラグを落とす（送信が終わったので、送信中フラグを落とす）

6.5.5. MASTER ヘッダ送信の際、レスポンス受信を行う SLAVE 側

(1)受信割り込み関数内(1)[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っていない場合]
→2 回入る受信割り込みの 1 回目、ヘッダ受信のタイミングでの割り込み
→ヘッダに含まれる ID でレスポンス送信を行うか、レスポンス受信を行うか判断するが ID が不一致だったので、レスポンス受信を行うケース
・レスポンス受信を行うので、g_lin_rx_flag の bit0(LIN_RX_READY)フラグを立てる

(2)受信割り込み関数内(2)[lin***n***_interrupt_receive()]

[g_lin_rx_flag の bit0(LIN_RX_READY)が立っている場合]
→2 回入る受信割り込みの 2 回目、レスポンス受信のタイミングでの割り込み
・データ受信を行う(受信データを受信バッファにコピー)
・レスポンス受信が完了したので、g_lin_rx_flag の bit0(LIN_RX_READY)フラグを落とす

以上、各ケースでのフラグ変数の変化を示します。

送信割り込み関数では、
g_lin_tx_flag の bit1(LIN_TX_RESPONSE)フラグ

g_lin_tx_flag & 0x2 != 0 (LIN_TX_RESPONSE フラグが立っている)	g_lin_tx_flag & 0x2 == 0 (LIN_TX_RESPONSE フラグが立っていない)
g_lin_tx_flag の LIN_TX_RESPONSE フラグを落とす ※MASTER のみ	g_lin_tx_flag を 0 にする(送信完了) ※MASTER, SLAVE 共

受信割り込み関数では、
g_lin_rx_flag の bit0(LIN_RX_READY)フラグ

g_lin_rx_flag==1 (LIN_RX_READY フラグが立っている)	g_lin_rx_flag==1 (LIN_RX_READY フラグが立っていない)
データ(レスポンスデータ)受信処理 g_lin_rx_flag を 0 にする(受信完了) ※MASTER, SLAVE 共	ヘッダに含まれる ID により処理を分ける (レスポンス送信 or 受信待機) ※SLAVE のみ

という、フラグによる処理分けを行っています。

・MASTER 動作、SLAVE 動作
・MASTER, SLAVE のどちらがレスポンスデータを送信するか
・SLAVE の場合ヘッダに含まれる ID が自局の ID と一致するか

により、処理が分かれるので、多少複雑になっています。

7. まとめ

lin.c に含まれる処理は、MASTER 動作時と SLAVE 動作時で、フラグ変数による場合分けは残っていますが、できるだけ簡素に使用できる様に構成したものです。

メイン関数や、タイマ割り込み関数はサンプルプログラムの位置付け、lin フォルダ内のファイル（lin.c, lin.h, lin_operation.h（一部）は、独立して使用できるように組み立てています。

取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2023.10.16	—	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/F15(QFP-100 ピン)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
LIN ソフトウェア編　マニュアル

株式会社 北斗電子
 2023 年 10 月 16 日改訂 REV.1.0.0.0 (231016)