LIN・CAN スタータキット RL78/F25

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/F25(QFP-100 ピン/QFP-80 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。

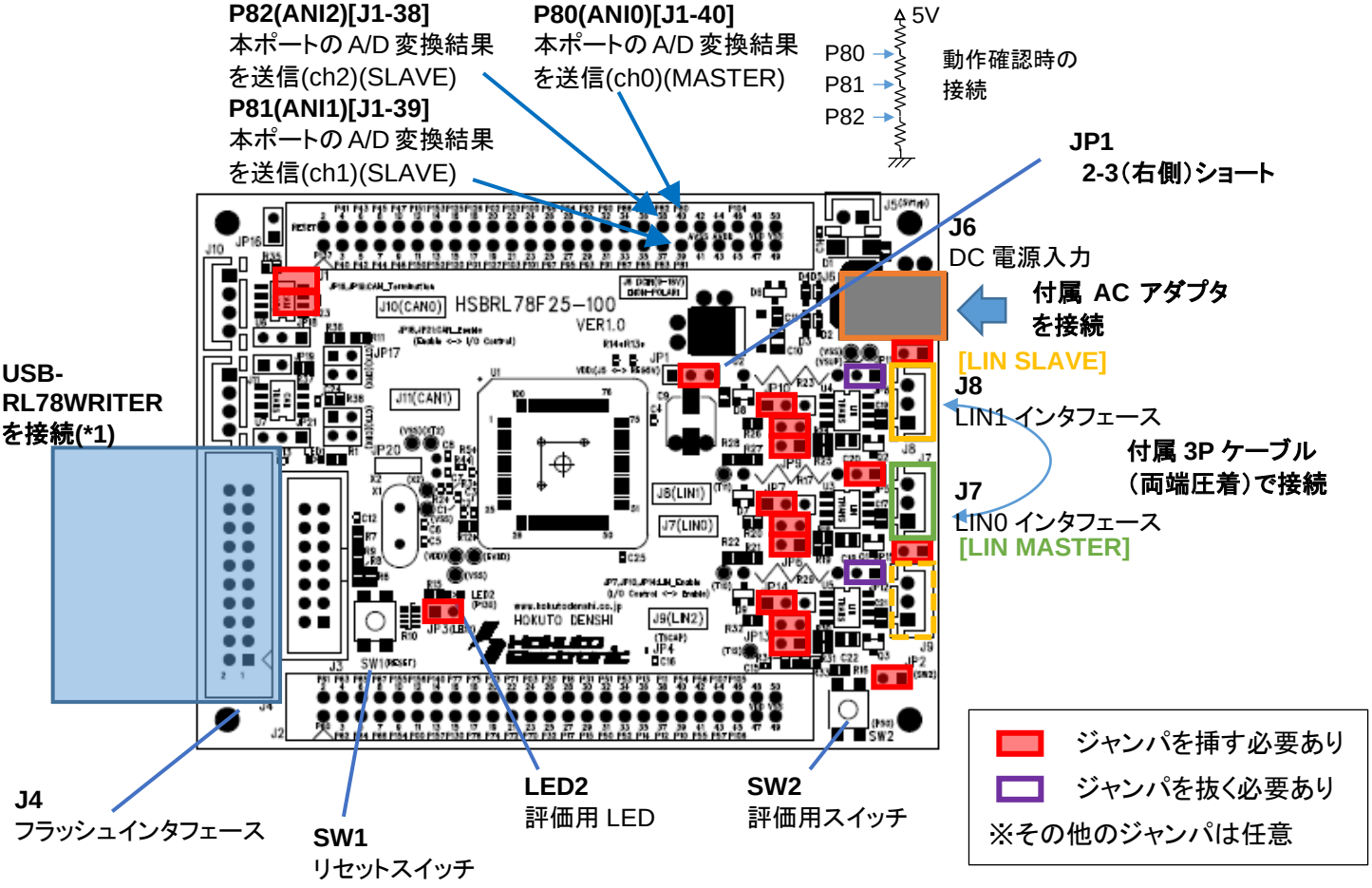
概要

本書は、「LIN・CAN スタータキット RL78/F25」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。

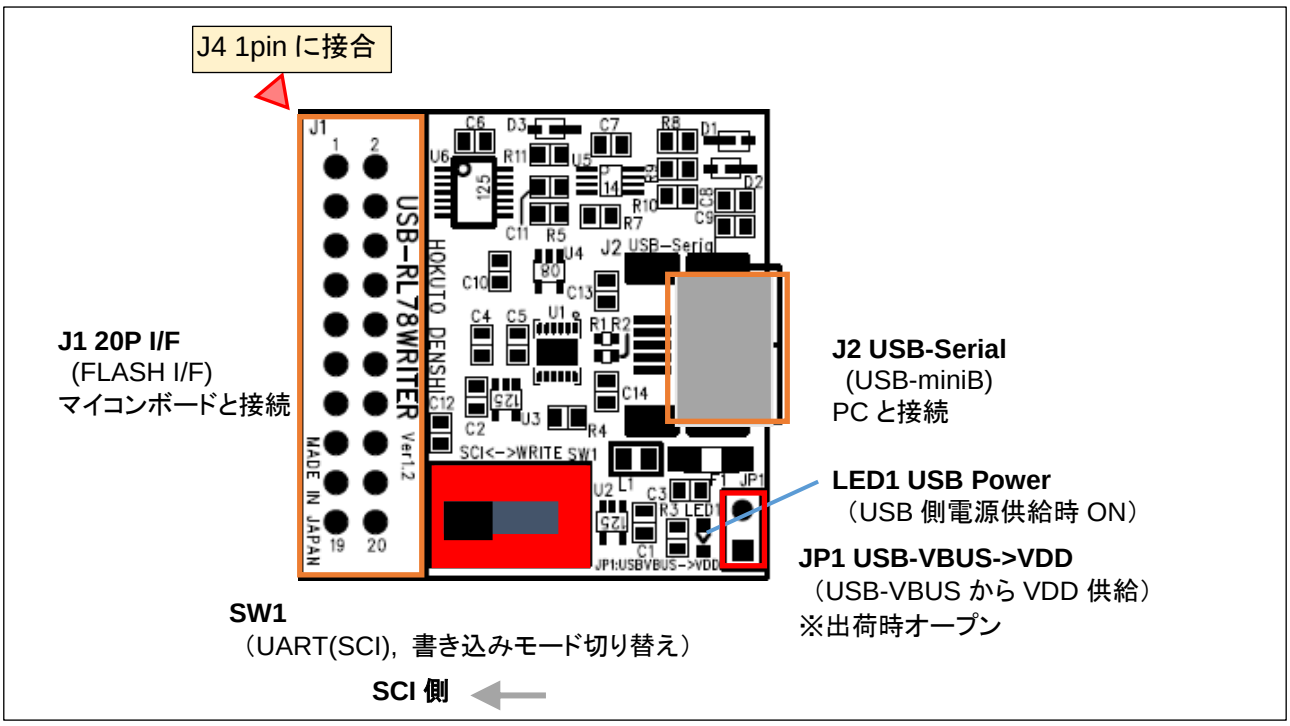
・付属 CD（本マニュアルに関連する内容のみ抜粋）

フォルダ			内容
SOURCE¥	RL78_F25_100_LIN¥		LIN サンプルプログラム HSBRL78F25-100 向け
	RL78_F25_80_LIN¥		LIN サンプルプログラム HSBRL78F25-80 向け
BINARY¥	RL78_LIN		上記プロジェクトをビルドした バイナリ （mot ファイル）

1. サンプルプログラムの動作



(*1)キット付属の、USB-RL78WRITER を使用する際は、SW1 を SCI 側に設定してください



通信アダプタとしては、USB-RL78WRITER 以外では、当社製 USB-ADAPTER などが使用可能です。

サンプルプログラムでは J7 のコネクタが LIN MASTER デバイスとして動作し、J8, J9 が LIN SLAVE デバイスとして動作するようになっています。J7 と J8、もしくは J7 と J9 をケーブルで接続してください。

サンプルプログラムは、「UART モード」（デフォルト）と「タイマモード」の 2 種類の動作モードを用意しています。タイマモードは一定時間毎に、マスタからヘッダを送信する動作。UART モードは、MASTER からのヘッダ送信をキーボードからの入力で行う動作モードです。

```
Copyright (C) 2026 HokutoDenshi. All Rights Reserved.

LIN Starter kit RL78/F25 Sample program.

Command:
  t : operation mode -> timer mode
  u : operation mode -> uart mode

>
```

J4 に接続した通信アダプタにつながる端末（仮想 COM ポート）には、起動時下記の表示が出力されます。端末は、115,200bps（, 8bit, パリティなし）で開いてください。
※端末は、teraterm などの COM ポートが開けるソフトを使ってください。
キーボードから、t を入力するとタイマモード、u を入力すると UART モードでの動作となります。

1.1. UART モード

キーボードから u を入力した場合です。

```
Operation start with uart mode.

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
  S : status print
  C : status clear
  M : message clear
```

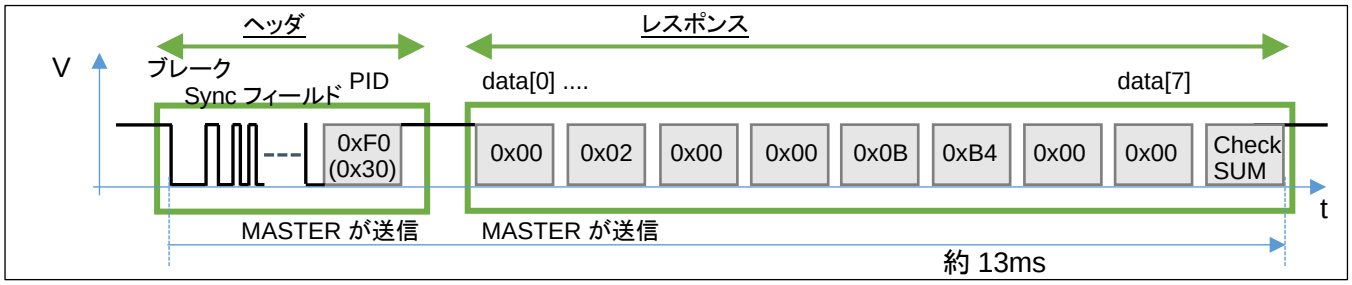
LIN ネットワーク上に LIN-ID=0x32~0x39 のデバイスが存在しない場合、応答なし

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）
S: LIN の ch 毎のステータスを表示
C: ステータス（エラー）をクリア
M: 画面のメッセージ消去

で動作します。

端末の画面で、キーボードから"0"を入力すると（リターンは不要）、LIN MASTER のポート(J7)からヘッダ・レスポンス通信が送信されます。



ヘッダ・レスポンスの送信元は、両方 MASTER になります。

その際端末には下記表示が出力されます。（※J7(LIN0)と J8(LIN1)を接続した場合）

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000200000BB40000 (*1)
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000200000BB40000 sum=0x4D (*2)
```

--は一連のメッセージの開始を示す表示。（**赤字**は注釈です。端末には表示されません。）
(*1)は、送信したことを示す表示。
(*2)は、LIN-ch1 側がデータを受信した表示。

(ch0)は、LIN-ch0, MASTER 側の動作に起因する表示です。(ch1)は、LIN-ch1, SLAVE 側の動作に起因する表示です。

[intr .....] の表示は割り込みが生じた際に表示されます。

ヘッダには、MASTER に割り当てた LIN ID(0x30)が含まれます。0x30 はパリティを付与すると、0xF0 になりますので、実際に送出されるデータは 0xF0 です。データの最後に付加されるチェックサムはデータに応じて計算される値（1 バイト）です。

レスポンスデータは、本プログラムでは 8 バイト固定としており、以下のデータフォーマットとなっています。

－LIN-ch0 送信データ（MASTER 側 0 コマンドで送出されるデータ）－

1	2	3	4	5	6	7	8
0x00	0x02	0x00	0x00	0x0B	0xB4	0x00	0x00
送信時間（2 バイト） ※ボードを起動してからの秒数		ボード上 LED	ボード上 SW	P80/ANI0 端子の A/D 変換値 (12bit) ※4096 の約 3/4 の値		0x00 固定	0x00 固定

・1~2 バイト目　1 秒毎にインクリメントされる数値

0x0000～0xFFFF

※ボードに搭載されているサブクロック(32.768kHz)でマイコンのリアルタイムクロックを動かして 1 秒毎にカウンタを増やす。ボード起動後の経過秒数。

・3 バイト目　ボード上 LED の状態

0x00 : LED2 は消灯

0x01 : LED2 は点灯

・4 バイト目　ボード上の SW の状態

0x00 : SW2 は押されていない

0x01 : SW2 は押されている

・5~6 バイト目　端子の A/D 変換値

0x0000 ~ 0x0FFF, 12bit の A/D 変換値

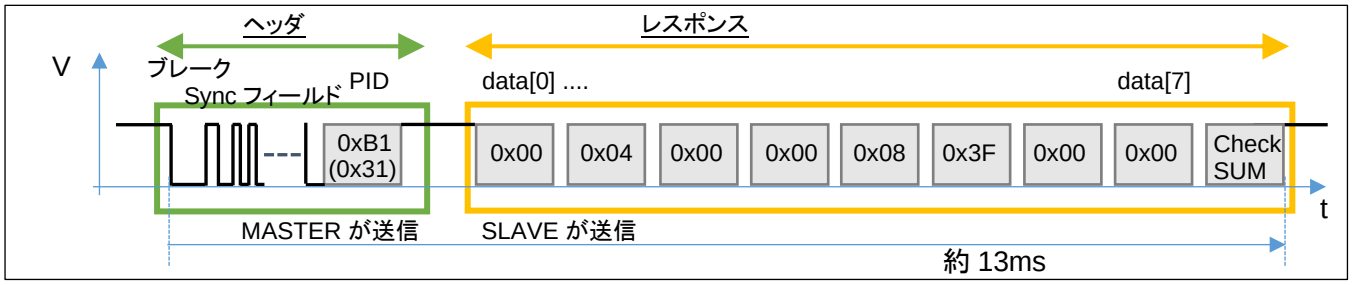
P80/ANI0 端子の A/D 変換値(*2)

※A/D 変換値は、この動作例では 5V-R-□-R-R-R-GND（□のポイントが P80 の接続）なので、大体 4096×3/4=3072(=0xC00)付近の値となります

・7~8 バイト目　0x00 固定

キーボードから、"1"~"9"を入力すると SLAVE デバイスに対するヘッダ送信を行います。

・送信した ID に合致する SLAVE デバイスが存在する場合



この場合の端末の表示は以下の様になります。

```
--
LIN header send(ch0) : LIN-ID=0x31 (*1)
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=send]
LIN response data send finished(ch1) : LIN-ID=0x31 data=0x00040000083F0000 (*2)
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x31 data=0x00040000083F0000 sum=0x03 (*3)
```

(*1)は、ヘッダを送信したことを示す表示。

(*2)は、LIN-ch1 側がレスポンスデータを送信した際の表示。

(*3)は、そのレスポンスデータを LIN-ch0 が受信した表示。

－LIN-ch1 送信データ（SLAVE 側　1 コマンドで返送されるデータ）－

1	2	3	4	5	6	7	8
0x00	0x04	0x00	0x00	0x08	0x3F	0x00	0x00
送信時間（2 バイト） ※ボードを起動して からの秒数		ボード上 LED	ボード上 SW	P81/ANI1 端子の A/D 変換値 (12bit) ※4096 の約 1/2 の値		0x00 固定	0x00 固定

※A/D 変換値は、この動作例では 5V-R-R-□-R-R-GND（□のポイントが P81 の接続）なので、大体 4096/2=2048 付近の値となります

なお、J7(LIN-ch0)と J9(LIN-ch2)を接続した場合は、以下の様な表示例となります。

```
--
LIN header send(ch0) : LIN-ID=0x32
[intr(ch2)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch2)=send]
LIN response data send finished(ch2) : LIN-ID=0x32 data=0x00090000042A0000
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x32 data=0x00090000042A0000 sum=0x96
```

－LIN-ch2 送信データ（SLAVE 側 2 コマンドで返送されるデータ）－

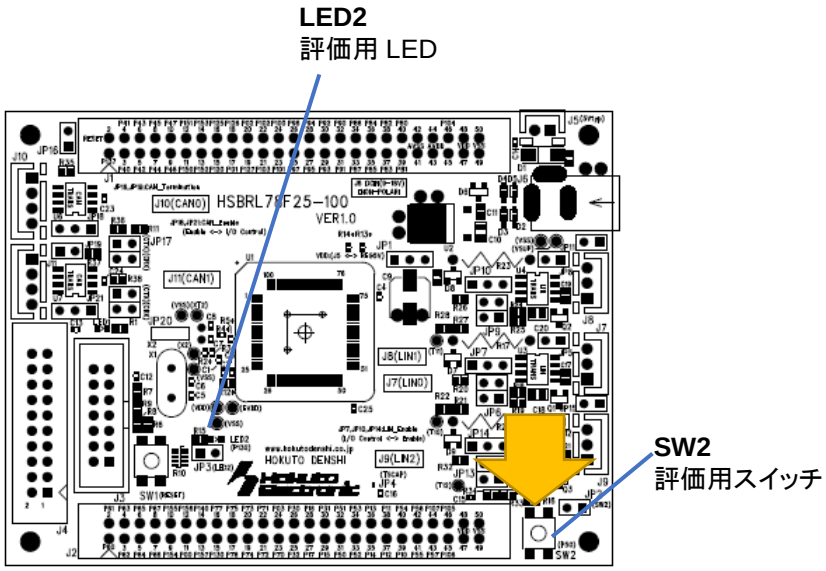
1	2	3	4	5	6	7	8
0x00	0x09	0x00	0x00	0x04	0x2A	0x00	0x00
送信時間（2 バイト） ※ボードを起動してからの秒数		ボード上 LED	ボード上 SW	P82/ANI2 端子の A/D 変換値 (12bit) ※4096 の約 1/4 の値		0x00 固定	0x00 固定

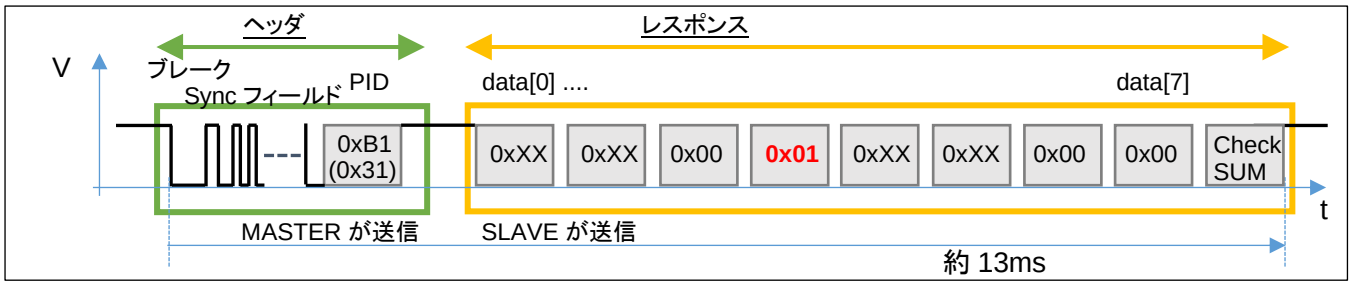
※A/D 変換値は、この動作例では 5V-R-R-R-□-R-GND（□のポイントが P82 の接続）なので、大体 4096/4=1024 付近の値となります

※5,6 バイト目の A/D 変換値に関しては、この動作例では、P81 に約 2.5V、P82 に約 1.25V が印加される回路を接続しているので、この様な値になります。

[参考]ボードの LED2 が点灯・消灯する条件に関して

・SW2 を押した状態で MASTER 側から SLAVE(ID=0x31)にヘッダを送信した場合





LIN-ch1(SLAVE)が返す、レスポンスデータの 4 バイト目が 0x01 になります。そのデータを MASTER 側が受信した場合、ボード上の LED(LED2)が点灯する動作となります。

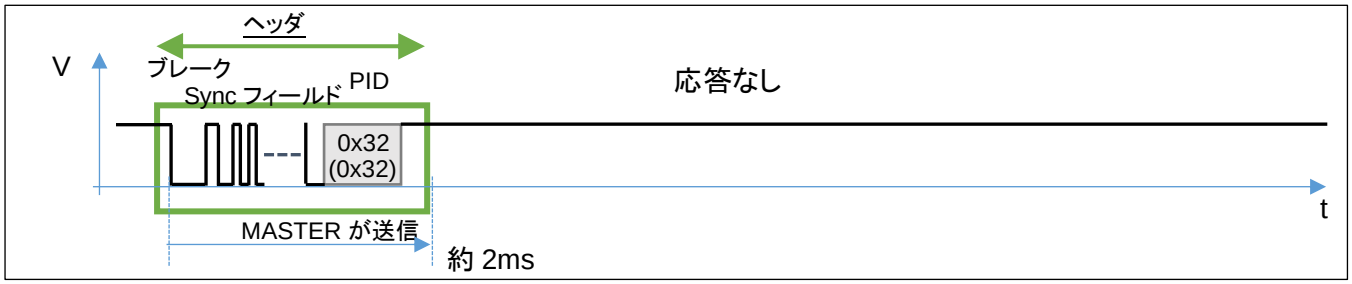
※SW2 を押さない状態で、MASTER 側から SLAVE(ID=0x31)にヘッダを送信した場合は、4 バイト目が 0x00 になり、そのデータを MASTER 側が受信した場合、ボード上の LED(LED2)が消灯する動作となります。

（MASTER 側は受信データの 4 バイト目のデータにより、ボード上の LED2 の点灯・消灯を切り替えます。）

（J7(LIN-ch0)と J9(LIN-ch2)を接続した場合は、SLAVE(ID=0x32)にヘッダを送信した場合同じ動作となります。）

（2 台のボードを接続した場合は、SLAVE 側のボードの SW2 の状態が、MASTER 側のボードの LED2 に反映されます。）

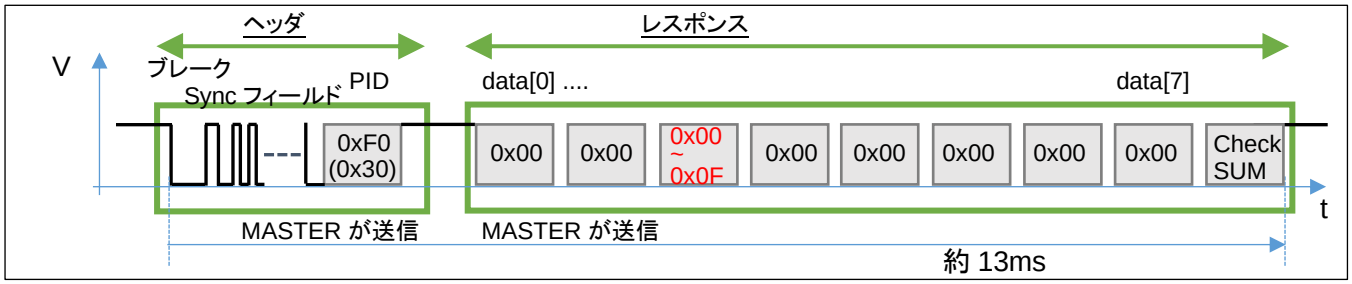
・送信した ID に合致する SLAVE デバイスが存在しない場合




```
--
LIN header send(ch0) : LIN-ID=0x32
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=stat LST0=0x08 LEST0=0x04]
```

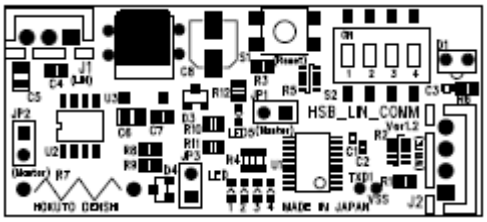
LIN バス上に MASTER が送信したヘッダ含まれる ID に合致する SLAVE デバイスが存在する場合、SLAVE はレスポンスデータを送信します。ID に合致する SLAVE デバイスが存在しない場合は、ヘッダ送信に対する応答が無い状態となります。

[参考]キーボードから、"a"~"p"を入力すると、MASTER ヘッダ・レスポンス送信となりますが、

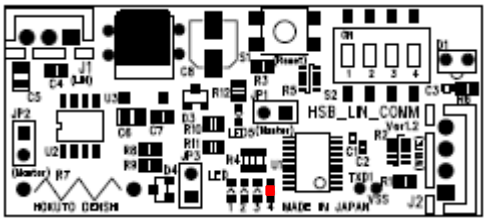


送信するデータの 3 バイト目が、a:0x00. b:0x01, c:0x02 ... p:0x0F と変化します。これは、LIN バス上に、HSB_LIN_COMM（別売の LIN 通信モジュール）を接続した場合向けのコマンドです。

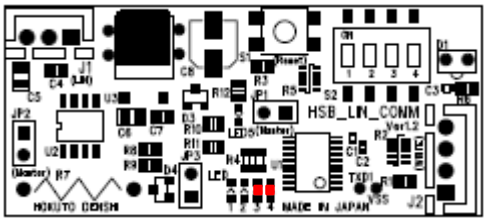
・HSB_LIN_COMM ボード（別売オプション）



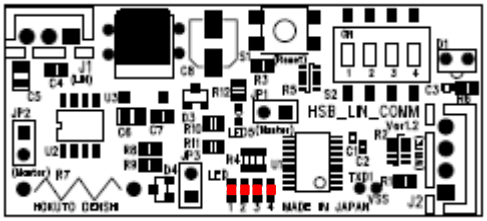
'a'コマンド
→LED1~4=OFF/OFF/OFF/OFF



'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

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、LIN バスに HSB_LIN_COMM ボードを接続した場合、'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。

－送信データ（MASTER 側 a~p コマンド）－

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

UART モードのその他のコマンドとしては、以下のコマンドがあります。

・S コマンド（ステータス表示）

```
LIN : status print
  LIN0 : LST0=0x00 LEST0=0x00
  LIN1 : LST1=0x00 LEST1=0x00
  LIN2 : LST2=0x00 LEST2=0x00
```

LST は、LIN のスタータスレジスタ。LEST はエラーステータスレジスタの値です。

・C コマンド（ステータスクリア）

```
LIN : status print
  LIN0 : LST0=0x00 LEST0=0x00
  LIN1 : LST1=0x80 LEST1=0x00
  LIN2 : LST2=0x00 LEST2=0x00
LIN : status clear
LIN : status print
  LIN0 : LST0=0x00 LEST0=0x00
  LIN1 : LST1=0x00 LEST1=0x00
  LIN2 : LST2=0x00 LEST2=0x00
```

S コマンド
→ステータスが 0x00 ではない

C コマンド（スタータスクリア）

S コマンド
→ステータスがクリアされる

上記は、スタータスが 0x00 ではない場合の、S コマンド、C コマンド、S コマンドの実行例です。C コマンドにより、ステータスとエラーステータスのレジスタがクリアされます。

・M コマンド
→画面のクリア
新しくコマンドを入力する場合、どこまでが前のコマンドの表示結果なのか区切りが判り難い際に使用します。

1.2. タイマモード

起動後、キーボードから t を入力した場合です。

動作モードを「タイマモード」にした場合は、（デフォルトでは）1 秒毎に MASTER 側 ch(LIN-ch0, J7)が

1. MASTER レスポンス送信(*1)
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。（UART モードでの、キーボードの、0,1,2,3,4,0...を 1 秒毎に入力した動作）

(*1)の送信データは、UART モードと同じです、

送信する ID や、ヘッダ送信を行う回数、送信間隔は lin_operation.h 内で変更可能です。

J7(LIN-ch0)と J8(LIN-ch1)を接続した場合（ID=0x31 の SLAVE デバイスが LIN バス上に存在、ID=0x32～0x34 のデバイスは存在しない）の端末の表示例は以下の様になります。

```
>t

Operation start with timer mode.


--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000100000BD00000
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000100000BD00000 sum=0x32

--
LIN header send(ch0) : LIN-ID=0x31
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=send]
LIN response data send finished(ch1) : LIN-ID=0x31 data=0x0002000008410000
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x31 data=0x0002000008410000 sum=0x03
```

LIN-ch0 がデータ送信

LIN-ch1 がデータ送信

表示例（続き）

```
--
LIN header send(ch0) : LIN-ID=0x32
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=stat LST0=0x08 LEST0=0x04]

--
LIN header send(ch0) : LIN-ID=0x33
[intr(ch1)=stat LST1=0x88 LEST1=0x08]
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=stat LST0=0x08 LEST0=0x04]

--
LIN header send(ch0) : LIN-ID=0x34
[intr(ch1)=stat LST1=0x88 LEST1=0x08]
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=stat LST0=0x08 LEST0=0x04]

--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000600000BB40000
[intr(ch1)=stat LST1=0x88 LEST1=0x08]
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000600000BB40000 sum=0x49
```

ID=0x32 のデバイスはつながっていないので誰も反応しない

ID=0x33 のデバイスはつながっていないので誰も反応しない

ID=0x34 のデバイスはつながっていないので誰も反応しない

LIN-ch0 がデータ送信

以下繰り返し。

1.3. MASTER 側動作の説明

LIN では、MASTER が通信動作の肝となります。MASTER 側からヘッダを送信しないと、通信は行われません。

MASTRE 側では、

・ヘッダ・レスポンス送信を行う
・ヘッダ送信を行う

のいずれかを行います。

1.3.1. ヘッダ・レスポンスデータ送信

LIN-ch0 からヘッダ・レスポンス送信を行う処理です。

main_sample.c 内、送信処理（抜粋）

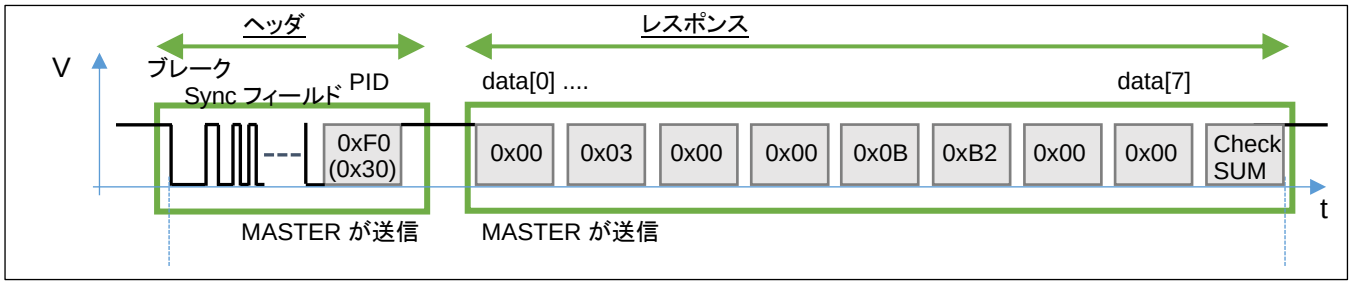
```
lin_send_data[0] = (unsigned char)((g_counter & 0x0000FF00UL) >> 8);
lin_send_data[1] = (unsigned char)(g_counter & 0x000000FFUL);
lin_send_data[2] = led;//LED2が点灯している時は0x01
lin_send_data[3] = sw;//SW2が押されている時は0x01
lin_send_data[4] = (unsigned char)((g_adc_val[0] & 0x0000FF00UL) >> 8);
lin_send_data[5] = (unsigned char)(g_adc_val[0] & 0x000000FFUL);
lin_send_data[6] = 0x00;//0x00固定
lin_send_data[7] = 0x00;//0x00固定

lin_master_header_response_send(LIN_CH0, LIN_MASTER_ID, lin_send_data, LIN_RESPONSE_DATA_SIZE);
```

LIN-ch0(J7)から、ID=0x30(LIN_MASTER_ID)で、設定したデータ(lin_send_data)を、8 バイト（=LIN_RESPONSE_DATA_SIZE)送信します。

その際、端末には下記表示が出ます。

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000300000BB20000 (*1)
[intr(ch1)=recv]
[intr(ch0)=send] (*2)
LIN header send finished(ch0) (*3)
[intr(ch0)=send] (*4)
LIN response data send finished(ch0) (*5)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000300000BB20000 sum=0x4E
```

※LIN-ch0 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません



(*1)ヘッダに続いてレスポンスデータを送信

ヘッダ・レスポンスの送信元は、両方 MASTER になります。

(*2)ヘッダ送信の割り込み

LIN の割り込みは、3 種類あり割り込みが入った際に [ ] で囲まれたメッセージが表示されます。

[intr(ch0)=send] 送信割り込み

[intr(ch0)=recv] 受信割り込み

[intr(ch0)=stat] ステータス割り込み

割り込みが入った際のメッセージは、デバッグ用に表示しているもので

lin_operation.h

```
//定義時端末にデバッグ表示を行う
#define SCI_DEBUG
```

lin_operation.h 内で、

#define SCI_DEBUG

を有効にした場合に表示されます。(デフォルト有効化)

上記をコメントアウトすると、[ ]で囲まれたメッセージは表示されなくなります。

(*3)ヘッダ送信メッセージ

送信割り込みのコールバック関数内で表示しているメッセージです。

(*4)レスポンスデータの送信割り込み

ヘッダ・レスポンス送信時は、送信割り込みが 2 回発生します。

(*5)レスポンスデータ送信メッセージ

送信割り込みのコールバック関数内で表示しているメッセージです。

1.3.1. ヘッダ送信

LIN-ch0 からヘッダ送信を行う処理です。

main_sample.c 内、送信処理(抜粋)

```
//SLAVE レスポンス送信要求（SLAVEに対してヘッダ送信）
lin_slave_id = (unsigned char)(LIN_SLAVE_RESPONSE_START_ID + xc - '0' - 1);
//'1'->0x31 ... '9'->0x39に変換
lin_master_header_send(LIN_CH0, lin_slave_id, LIN_RESPONSE_DATA_SIZE);//送信元はLIN-CH0
```

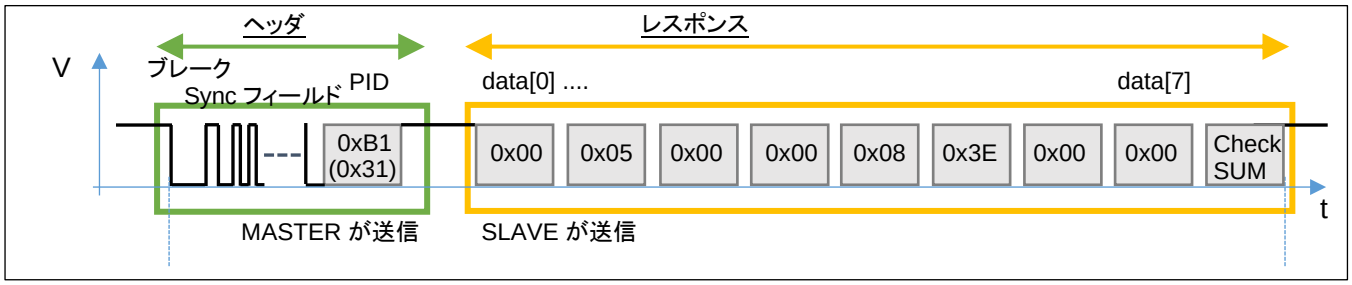
LIN-ch0(J7)から、lin_slave_id の ID 値で、ヘッダ送信を行います。SLAVE から送信されるデータバイト数は、8 バイト(=LIN_RESPONSE_DATA_SIZE)期待です。

(lin_slave_id には、キーボードからの入力に応じて、0x31～0x39 が代入されます。)

端末には下記表示が出ます。

```
--
LIN header send(ch0) : LIN-ID=0x31 (*1)
[intr(ch1)=recv]
[intr(ch0)=send] (*2)
LIN header send finished(ch0) (*3)
[intr(ch1)=send]
LIN response data send finished(ch1) : LIN-ID=0x31 data=0x00050000083E0000
[intr(ch0)=recv] (*4)
LIN data receive(ch0) : LIN-ID=0x31 data=0x00050000083E0000 sum=0x03 (*5)
```

※LIN-ch0 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません



(*1)ヘッダを送信

→ヘッダに含まれる ID は 0x31 です

(*2)ヘッダ送信の割り込み

(*3)ヘッダ送信メッセージ

送信割り込みのコールバック関数内で表示しているメッセージです。

(*4)受信割り込み

SLAVE が返したレスポンスデータを受信した際に割り込みが入ります。

lin_intr.c 内、受信割り込み処理（抜粋）

```
lin_message msg;

msg.id = lin_id_decode(LIDB0);
msg.data[0] = LDB01;
msg.data[1] = LDB02;
msg.data[2] = LDB03;
msg.data[3] = LDB04;
msg.data[4] = LDB05;
msg.data[5] = LDB06;
msg.data[6] = LDB07;
msg.data[7] = LDB08;
msg.sum = LCBR0;
msg.size = LDFC0 & 0xf;

lin_data_enqueue(LIN_CH0, &msg);//受信データを受信バッファに格納する
```

lin_massage 構造体に
データをコピー

lin_id_decode()は、実際に受信した ID のデータ(LIN-ID に対してエンコードされている)を、通常の LIN-ID に変換しています。

lin_data_enqueue()は、ユーザ側で用意した受信バッファにデータを格納する関数です。本プログラムでは、16 段の受信バッファを用意して、受信割り込み内では受信バッファに格納する処理としています。
(受信割り込みを抜けた後で、受信データを処理しています。)
(※実際のプログラムでは、この部分をユーザ側で行いたい処理に変更して良いです。)

lin_message 構造体定義(lin.h 内)

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

lin_message 構造体は、LIN-ID, データ(最大 8 バイト)、チェックサム、データサイズを保持する変数です。

(*5)受信データの表示

main_sample.c 内、受信データ表示処理(抜粋)

```
void received_data_print(void)
{
    //LIN受信データ処理関数

    int ret;
    lin_message lin_msg;
    unsigned short i;

    //LIN-ch0受信データ処理
    while (1)
    {
        //MASTER側受信処理 (SLAVEのレスポンスデータを受信)
        ret = lin_data_dequeue(LIN_CH0, &lin_msg);
        if (ret != LIN_RET_NODATA)
        {
            //LIN受信データあり
            sci_write_str("LIN data receive(ch0) : LIN-ID=0x");
            sci_write_uint8_hex(lin_msg.id);
            sci_write_str(" data=0x");
            for (i=0; i<LIN_RESPONSE_DATA_SIZE; i++)
            {
                sci_write_uint8_hex(lin_msg.data[i]);
            }
            sci_write_str(" sum=0x");
            sci_write_uint8_hex(lin_msg.sum);
            sci_write_str("¥n");
        }
        else
        {
            //受信データなし
            break;
        }
    }
```

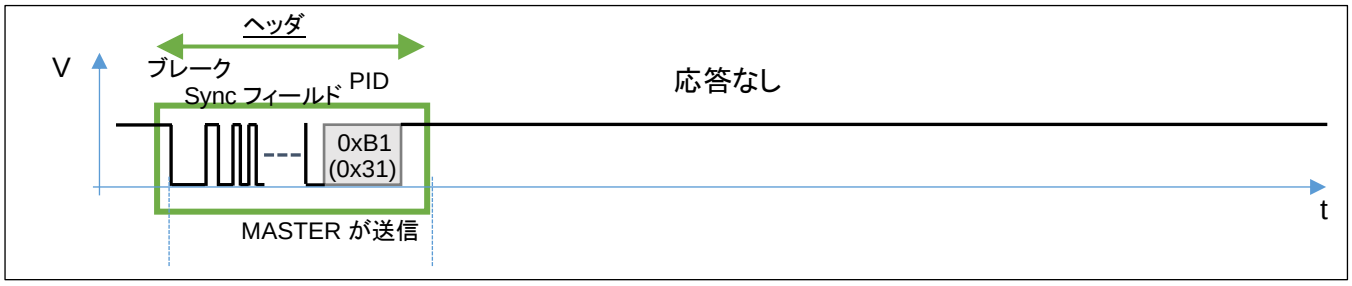
メイン関数内では、受信データ表示関数（received_data_print()）をループ内で呼び出しており、受信バッファに格納したデータを取り出す処理
lin_data_dequeue()
を実行しています。受信データが空になるまで、取り出しと表示のループとなります。

なお、ヘッダ送信を行った場合でレスポンスを返す SLAVE が居ない場合の動作は以下の様になります。

端末には下記表示が出ます。

```
--
LIN header send(ch0) : LIN-ID=0x31 (*1)
[intr(ch0)=send] (*2)
LIN header send finished(ch0) (*3)
[intr(ch0)=stat LST0=0x08 LEST0=0x04] (*4)
```

※LIN-ch0 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません



(*1)ヘッダを送信
→ヘッダに含まれる ID は 0x31 です
(*2)ヘッダ送信の割り込み
(*3)ヘッダ送信メッセージ
(*4)スタータス割り込み

このとき、LIN-ch0 では、ステータス割り込みが入り、

[intr(ch0)=stat LST0=0x08 LEST0=0x04]

ステータスレジスタ(LST0)=0x08
エラーレジスタ(LEST0)=0x04
となっています。それぞれのレジスタの意味は、

LST（ステータスレジスタ）

	b7 HTRC	b6 D1RC	b5 ―	b4 ―	b3 ERR	b2 ―	b1 FRC	b0 FTC
LST	0	0	0	0	**1**	0	0	0

→エラーがある

LEST（エラーステータスレジスタ）

	b7 RPER	b6 ―	b5 CSER	b4 ―	b3 FER	b2 FTER	b1 PBER	b0 BER
LST	0	0	0	0	0	**1**	0	0

→レスポンスタイムアウトエラー

である事が判ります。

SLAVE ID=0x31 に対してヘッダ送信を行ったが、誰も応答せずにタイムアウトでエラーとなります。

ちなみに、ステータス割り込み内で、エラーとステータスはクリアされますので、タイムアウトエラーとなった場合でも、動作が継続（次のヘッダ送信は可能）となります。

1.4. SLAVE 側動作の説明

SLAVE 側は、LIN の場合自発的にデータを送信する事は出来ません。

SLAVE 側でできる事は、

・LIN バス上を流れているデータを受信する（MASTER がレスポンスデータを送信した場合）
・LIN バス上を流れているデータを受信する（自分以外の SLAVE がレスポンスデータを送信した場合）
・MASTER が送信したヘッダに、自分自身の ID が含まれていた場合にレスポンスデータを送信する

となります。

1.4.1. ヘッダ・レスポンスデータ受信

MASTER がヘッダ及びレスポンスデータを送信した場合の端末の表示例。

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000300000BB20000
[intr(ch1)=recv] (*1)
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv] (*2)
LIN data receive(ch1) : LIN-ID=0x30 data=0x000300000BB20000 sum=0x4E (*3)
```

※LIN-ch1 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません

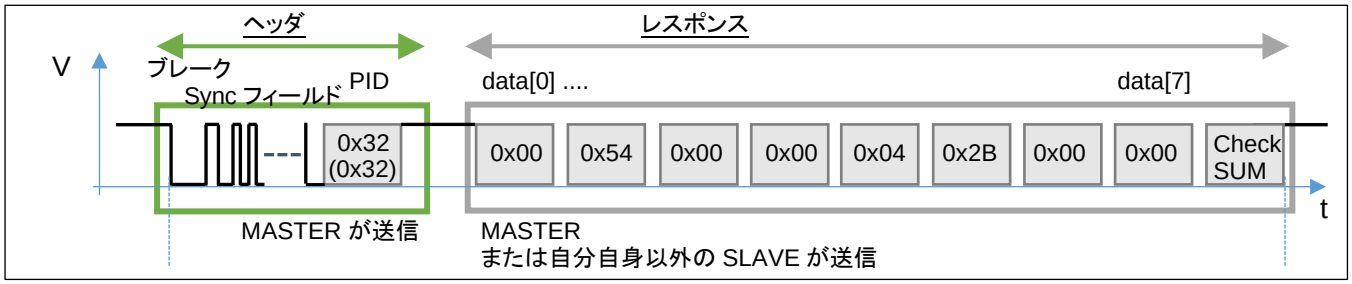
自分自身以外の SLAVE がレスポンスデータを送信した場合の表示例。
（MASTER, LIN-ch0 は、ID=0x32 のヘッダ送信）

```
--
LIN header send(ch0) : LIN-ID=0x32
[intr(ch2)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=recv] (*1)
[intr(ch2)=send]
LIN response data send finished(ch2) : LIN-ID=0x32 data=0x00540000042B0000
[intr(ch0)=recv]
[intr(ch1)=recv] (*2)
LIN data receive(ch0) : LIN-ID=0x32 data=0x00540000042B0000 sum=0x4A
LIN data receive(ch1) : LIN-ID=0x32 data=0x00540000042B0000 sum=0x4A (*3)
```

※LIN-ch1 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません

※別売の LIN-HUB を使い、J7(LIN-ch0)と J8(LIN-ch1)と J9(LIN-ch2)を接続した場合

ID=0x32 のデバイス（LIN-ch2)がレスポンスデータを送信し、LIN-ch1 がデータを受信。



(*1)ヘッダの受信割り込み

lin_intr.c 内、受信割り込み処理（抜粋、一部変更）

```
id = lin_id_decode(LIDB1);

if (id == g_lin_id[LIN_CH1])
{
    //IDが自局のものなのでレスポンス送信を行う
    LDB11 = g_lin_send_buf[LIN_CH1][0];
    LDB12 = g_lin_send_buf[LIN_CH1][1];
    LDB13 = g_lin_send_buf[LIN_CH1][2];
    LDB14 = g_lin_send_buf[LIN_CH1][3];
    LDB15 = g_lin_send_buf[LIN_CH1][4];
    LDB16 = g_lin_send_buf[LIN_CH1][5];
    LDB17 = g_lin_send_buf[LIN_CH1][6];
    LDB18 = g_lin_send_buf[LIN_CH1][7];

    LDFC1 |= 0x10;//(b4)RFT=1, レスポンスフィールド:送信

    LTRC1 = 0x02;//レスポンス送信
}
else
{
    //IDが自局のもの以外はレスポンス受信を行う
    LDFC1 &= ~0x10;//(b4)RFT=0, レスポンスフィールド:受信

    LTRC1 = 0x02;//レスポンス受信
}
```

本ケースではこちらの処理

受信割り込み関数内で、ヘッダに含まれる ID を確認して、自分自身の ID と異なる場合は、受信する動作としています。（MASTER が MASTER レスポンス送信で自身の ID=0x30 をヘッダ送信した場合も、SLAVE の ID（0x31 以外）をヘッダ送信した場合も同じ処理となります。）

[参考]
※この部分で
(a)ID が自分自身の ID と一致
(b)ID が MASTER の ID と一致
(c)それ以外（他の SLAVE に向けたヘッダ送信）
の 3 パターンの条件分岐とする事も可能です。(b)の場合は受信処理、(c)の場合は受信しない、といった形の処理もできます。

(*2)レスポンスデータの受信割り込み

lin_intr.c 内、受信割り込み処理（抜粋）

```
lin_message msg;

msg.id = lin_id_decode(LIDB1);
msg.data[0] = LDB11;
（中略）
msg.data[7] = LDB18;
msg.sum = LCBR1;
msg.size = LDFC1 & 0xf;

lin_data_enqueue(LIN_CH1, &msg);//受信データを受信バッファに格納する
```

lin_massage 構造体に
データをコピー

この処理は、2.3.1 の MASTER がデータを受信した場合と同じ処理となります。

ユーザが用意した受信バッファに受信データをコピーします。

(*3)受信データの表示

main_sample.c 内、受信データ表示処理（抜粋）

```
void received_data_print(void)
{
    //LIN受信データ処理関数

    int ret;
    lin_message lin_msg;
    unsigned short i;

（中略）

    //LIN-ch1受信データ処理
    while (1)
    {
        //SLAVER側受信処理（MASTERレスポンスデータを受信）（外部に別なSLAVEデバイスを接続した場合は、自分自身以外のSLAVEが送信したデータも受信する）
        ret = lin_data_dequeue(LIN_CH1, &lin_msg);
        if (ret != LIN_RET_NODATA)
        {
            //LIN受信データあり
            sci_write_str("LIN data receive(ch1) : LIN-ID=0x");
            sci_write_uint8_hex(lin_msg.id);
            sci_write_str(" data=0x");
            for (i=0; i<LIN_RESPONSE_DATA_SIZE; i++)
            {
                sci_write_uint8_hex(lin_msg.data[i]);
            }
            sci_write_str(" sum=0x");
            sci_write_uint8_hex(lin_msg.sum);
            sci_write_str("¥n");
        }
        else
        {
            //受信データなし
            break;
        }
```

メイン関数内では、受信データ表示関数（received_data_print()）をループ内で呼び出しており、受信バッファに格納したデータを取り出す処理

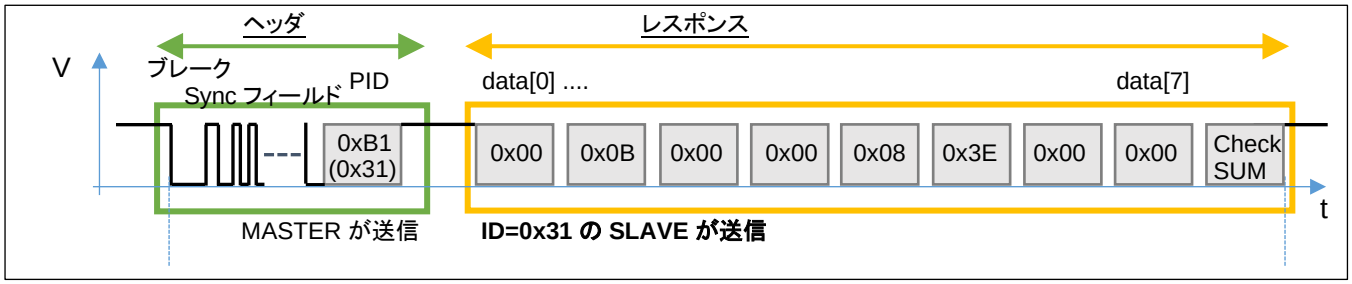
lin_data_dequeue()
を実行しています。受信データが空になるまで、取り出しと表示のループとなります。この処理は、MASTER 側でレスポンスデータを受信した場合と同じです。

1.4.2. ヘッダ受信・レスポンスデータ送信

MASTER 側が、ID=0x31（LIN-ch1 の ID）でヘッダ送信した場合の端末の表示例。

```
--
LIN header send(ch0) : LIN-ID=0x31
[intr(ch1)=recv] (*1)
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=send] (*2)
LIN response data send finished(ch1) : LIN-ID=0x31 data=0x000B0000083E0000 (*3)
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x31 data=0x000B0000083E0000 sum=0xFC
```

※LIN-ch1 に関連するメッセージを強調表示，赤字の部分はメッセージには表示されません



SLAVE 側は、MASTER が送信したヘッダを受信して、ヘッダに含まれる ID が自分のものだった場合に、レスポンスデータを返すという動作となりますので、下記のような動作となります。

・ヘッダは受信待機しておく（初期化の際）
・レスポンスデータを予め決めて設定しておく

ヘッダ受信後に、レスポンスデータを決めても良いですが、本サンプルプログラムではレスポンスデータは定期的（RTC タイマを使用して 1 秒に 1 回）に返信するデータ（8 バイト）を決めて更新しておく動作としています
※ヘッダ受信後はできるだけ速やかにレスポンスデータを送信したいので、そのような処理としています

intr.c 内、1 秒毎に実行されるタイマ割り込み処理（抜粋）

```
//RTC(1s) スレーブの送信データ更新
void timer_rtc(void)
{
    unsigned char lin_slave_response_data[LIN_RESPONSE_DATA_SIZE];
    unsigned char led = 0x00;
    unsigned char sw = 0x00;

    //カウンタ変数のインクリメント
    g_counter++;

    //LEDとSWの状態を送信データに乗せる
    if (LED_PORT == LED_ON) led = 0x01;
    if (SW_PORT == SW_ON) sw = 0x01;

    //LIN SLAVEレスポンスデータの設定(ch1)
    //送信データ: ss ss LED SW aa aa 0x00 0x00 , ss ss:1秒毎にインクリメントされる値（16ビット）,
LED:P130(LED2)の値, SW:P50(SW2)の値, aa aa:P81/ANI1のA/D変換値, 0x00 0x00:00固定
    lin_slave_response_data[0] = (unsigned char)((g_counter & 0x0000FF00UL) >> 8);
    lin_slave_response_data[1] = (unsigned char)(g_counter & 0x000000FFUL);
    lin_slave_response_data[2] = led;//LED2が点灯している時は0x01
    lin_slave_response_data[3] = sw;//SW2が押されている時は0x01
    lin_slave_response_data[4] = (unsigned char)((g_adc_val[1] & 0x0000FF00UL) >> 8);
    lin_slave_response_data[5] = (unsigned char)(g_adc_val[1] & 0x000000FFUL);
    lin_slave_response_data[6] = 0x00;//0x00固定
    lin_slave_response_data[7] = 0x00;//0x00固定

    lin_response_data_set(LIN_CH1, lin_slave_response_data, LIN_RESPONSE_DATA_SIZE);
    //レスポンス要求が来た場合に返すデータ, SLAVE側でセットするデータ, LIN-CH1
```

lin_response_data_set()関数の実行は、(*1)～(*3)とは別に、定期的に実行されている処理となります。

(*1)ヘッダの受信割り込み

lin_intr.c 内、受信割り込み処理（抜粋、一部変更）

```
id = lin_id_decode(LIDB1);

if (id == g_lin_id[LIN_CH1])
{
    //IDが自局のものなのでレスポンス送信を行う
    LDB11 = g_lin_send_buf[LIN_CH1][0];
    LDB12 = g_lin_send_buf[LIN_CH1][1];
    LDB13 = g_lin_send_buf[LIN_CH1][2];
    LDB14 = g_lin_send_buf[LIN_CH1][3];
    LDB15 = g_lin_send_buf[LIN_CH1][4];
    LDB16 = g_lin_send_buf[LIN_CH1][5];
    LDB17 = g_lin_send_buf[LIN_CH1][6];
    LDB18 = g_lin_send_buf[LIN_CH1][7];

    LDFC1 |= 0x10;//(b4)RFT=1, レスポンスフィールド:送信

    LTRC1 = 0x02;//レスポンス送信
}
else
```

本ケースではこちらの処理

ヘッダの受信後に受信した ID を調べて、自分自身の ID と一致した場合は、レスポンス送信の処理となります。

(*2)レスポンス送信の割り込み

(*1)で、レスポンスデータの送信を行ったので、送信割り込みが発生します。

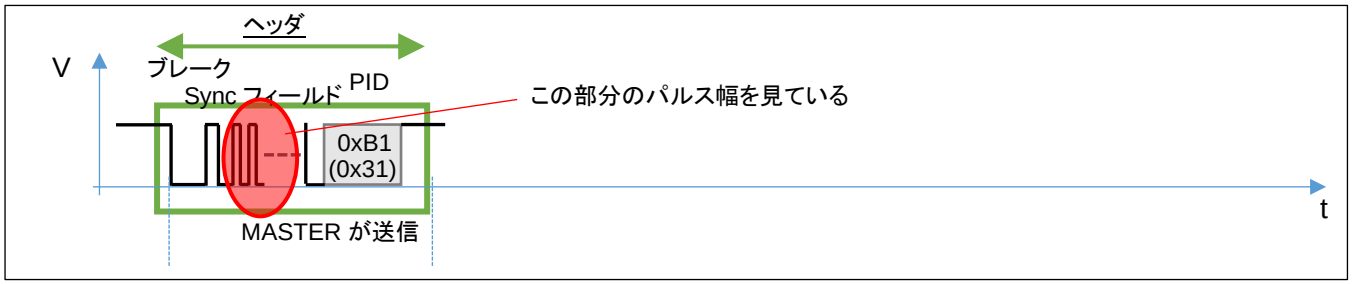
(*3)送信割り込みのコールバック関数内でのメッセージ表示

main_sample.c 内、送信コールバック関数

```
void lin1_interrupt_send_callback(void)
{
    unsigned char send_data[LIN_RESPONSE_DATA_SIZE];
    int i;

    for (i=0; i<LIN_RESPONSE_DATA_SIZE; i++)
    {
        send_data[i] = g_lin_send_buf[LIN_CH1][i];
        //g_lin_send_buf は割り込み優先度と多重割り込み許可により更新される可能性があるので最初にコピーしておく
    }

    sci_write_str("LIN response data send finished(ch1) : LIN-ID=0x");
    sci_write_uint8_hex(g_lin_id[LIN_CH1]);
    sci_write_str(" data=0x");
    for (i=0; i<LIN_RESPONSE_DATA_SIZE; i++)
    {
        sci_write_uint8_hex(send_data[i]);
    }
    sci_write_str("¥n");
}
```

送信割り込みが発生した時点での送信データ(=lin_response_data_set()で設定したデータ)を、端末に表示させています。

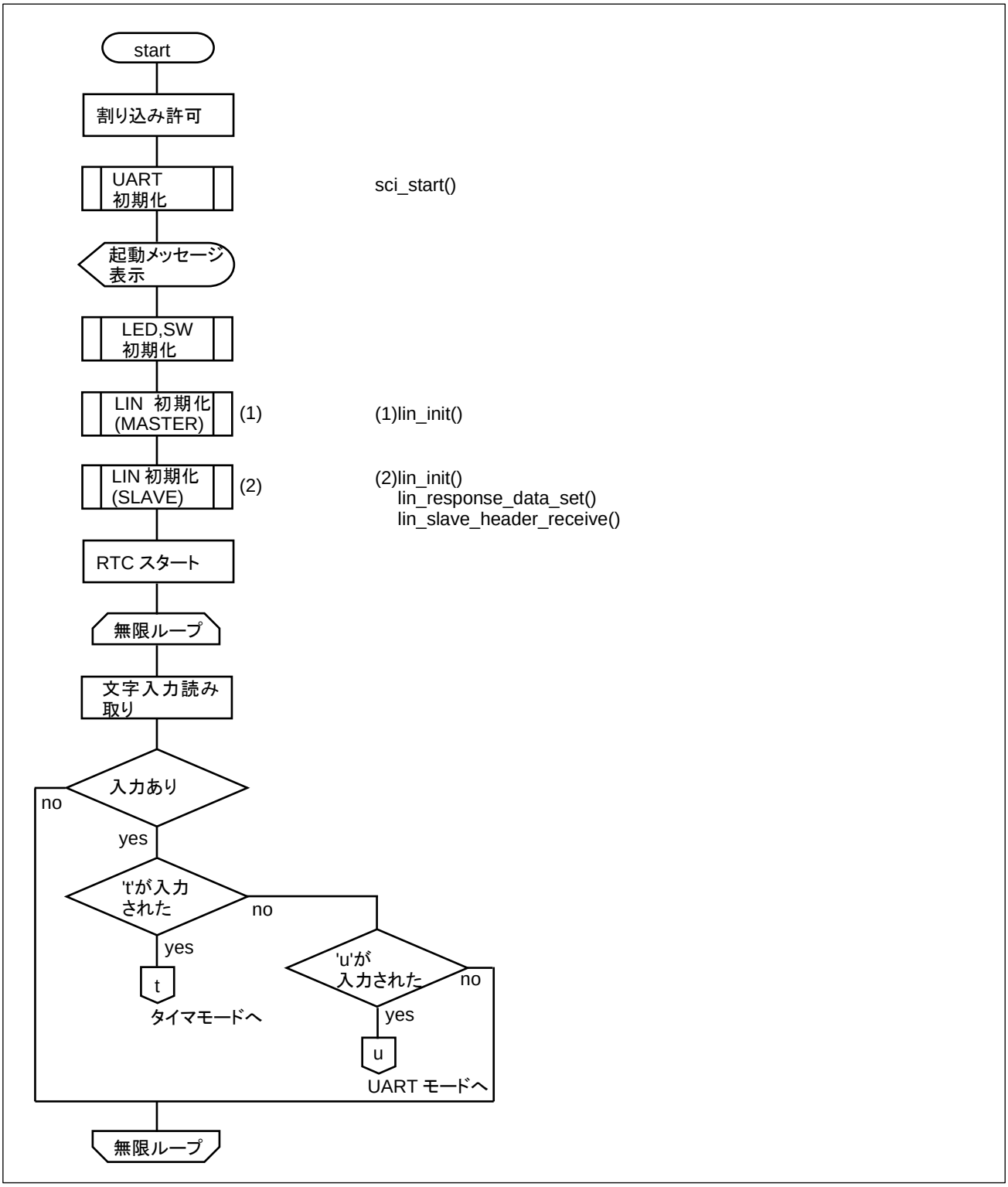
1.4.3. SLAVE 側の速度設定に関して



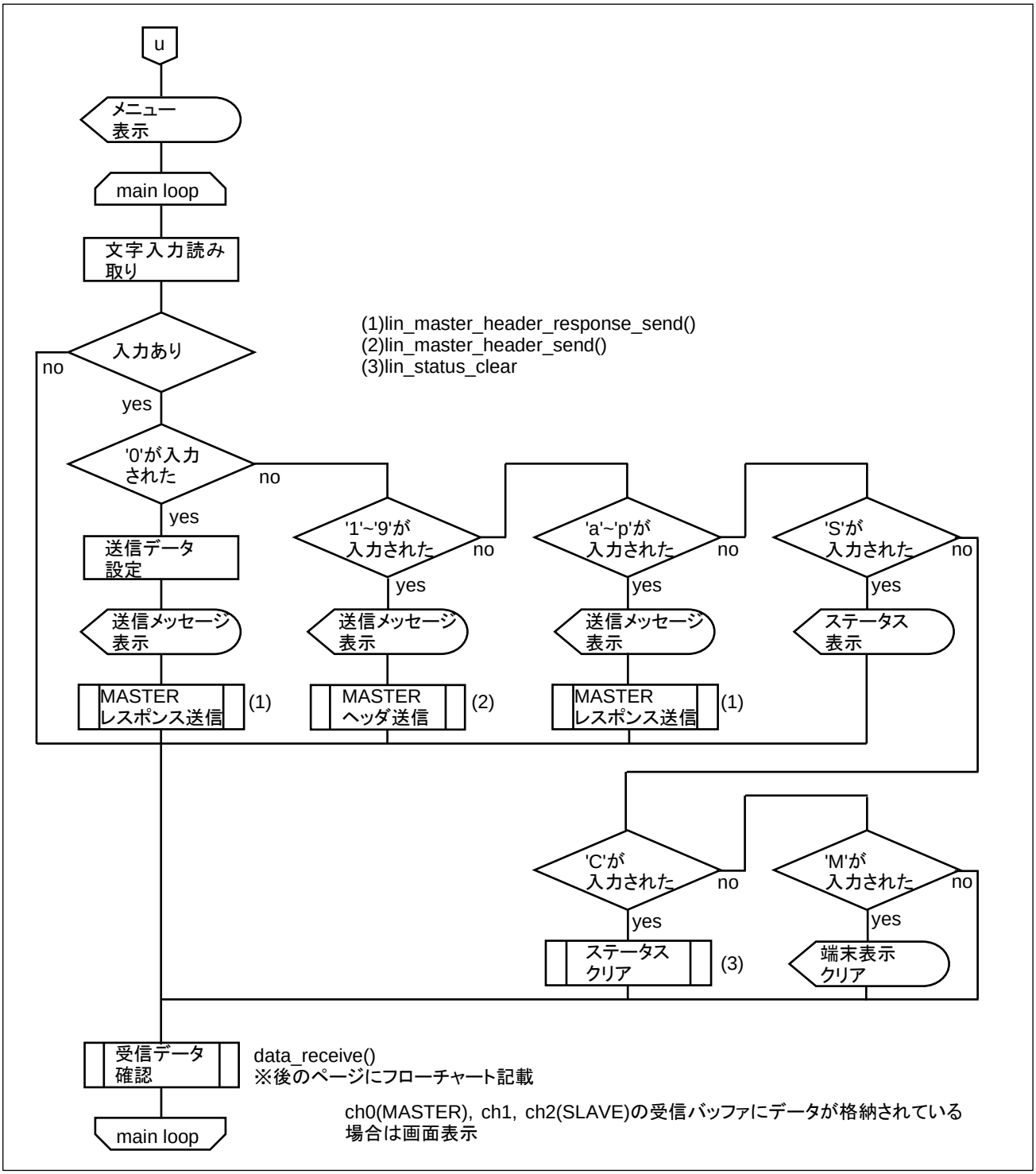
SLAVE 側は、オート・ボーレート設定としているので、MASTER が送ってくるヘッダ内の SYNC フィールドの 0x55(L/H の繰り返し)の信号に合わせて通信速度が決定されます。通信速度は、2.4kbps~20kbps の範囲のデータを受信可能です。(パケット毎にビットレートが異なっていても受信可能です。)

1.5. 処理のフロー

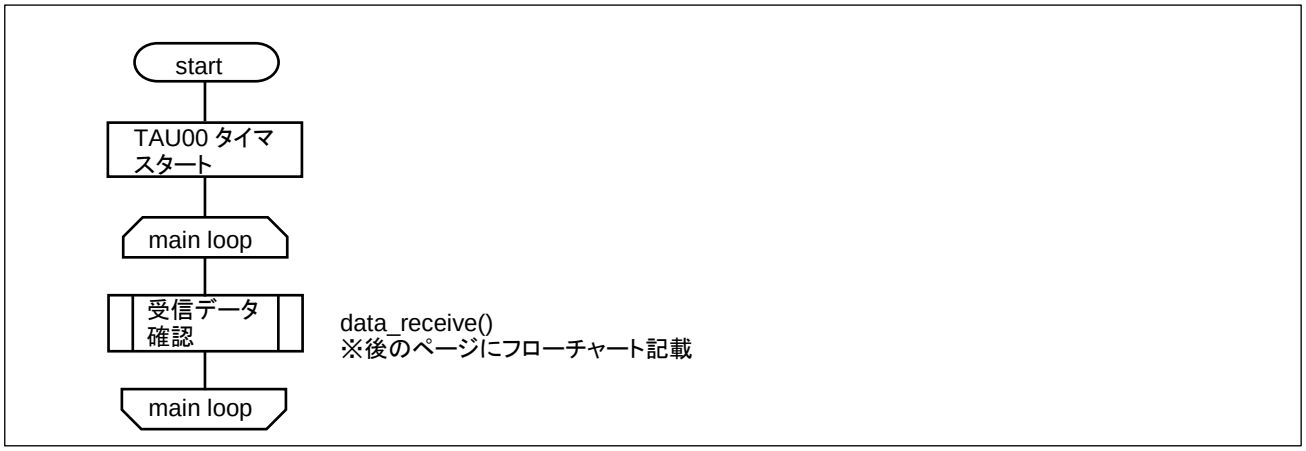
メイン関数　main_sample.c 内 main_sample()



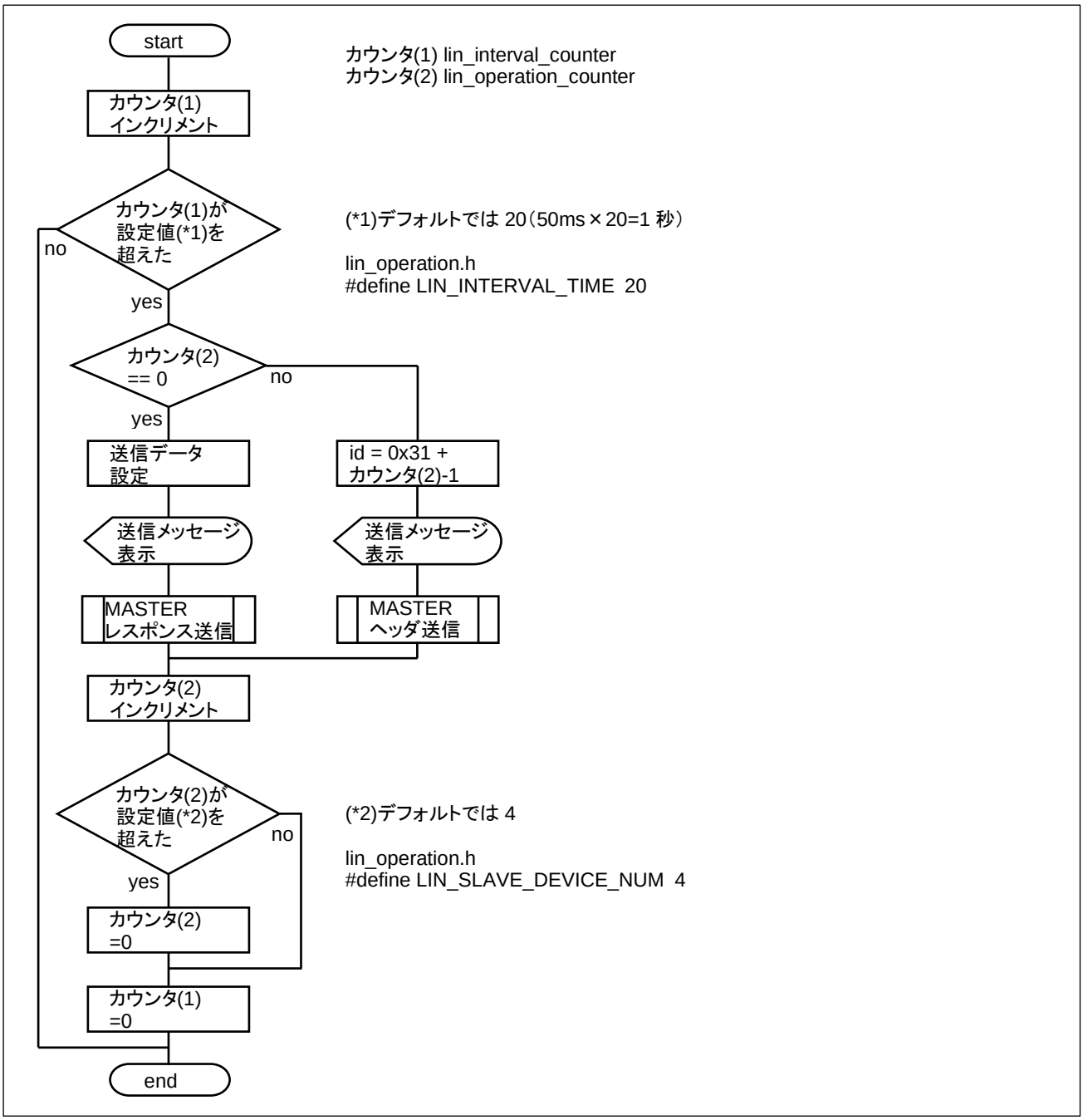
メイン関数　main_sample.c 内 main_sample()　※「UART モード」キーボードからのコマンドで動作させる場合



メイン関数　main_sample.c 内 main_sample()　※「タイマモード」タイマで一定時間毎にヘッダ送信させる場合



タイマ(TAU00)処理　intr.c 内　timer_tau00()　50ms 毎の定期処理　※「タイマモード」の場合のみ使用



MASTER レスポンス送信(LIN-ID=0x30)

MASTER ヘッダ送信(LIN-ID=0x31)

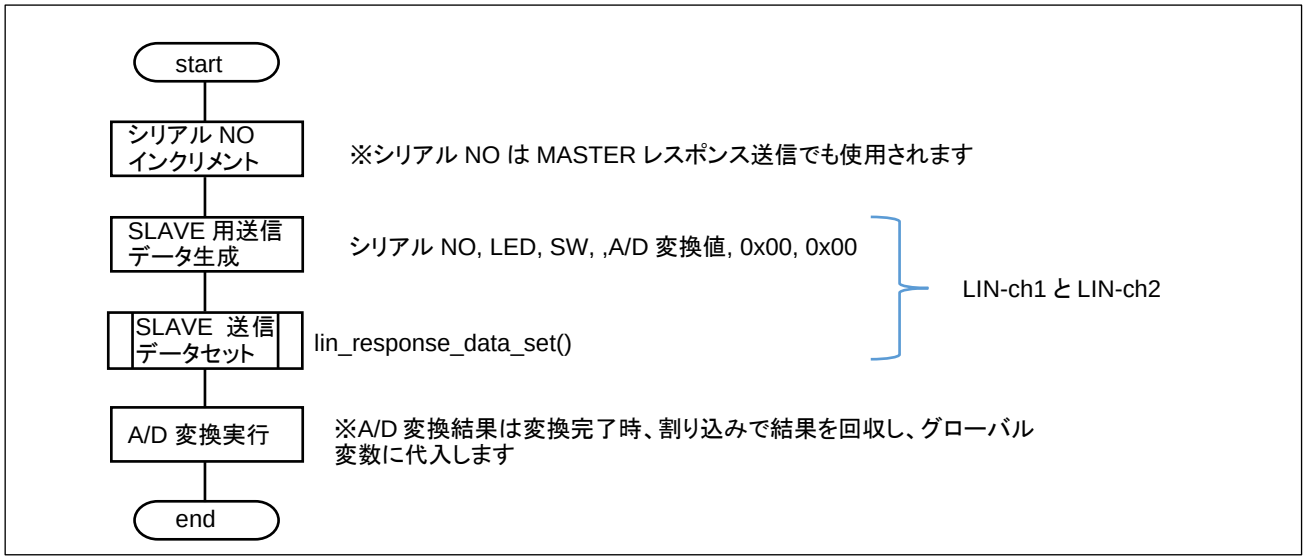
MASTER ヘッダ送信(LIN-ID=0x32)

MASTER ヘッダ送信(LIN-ID=0x33)

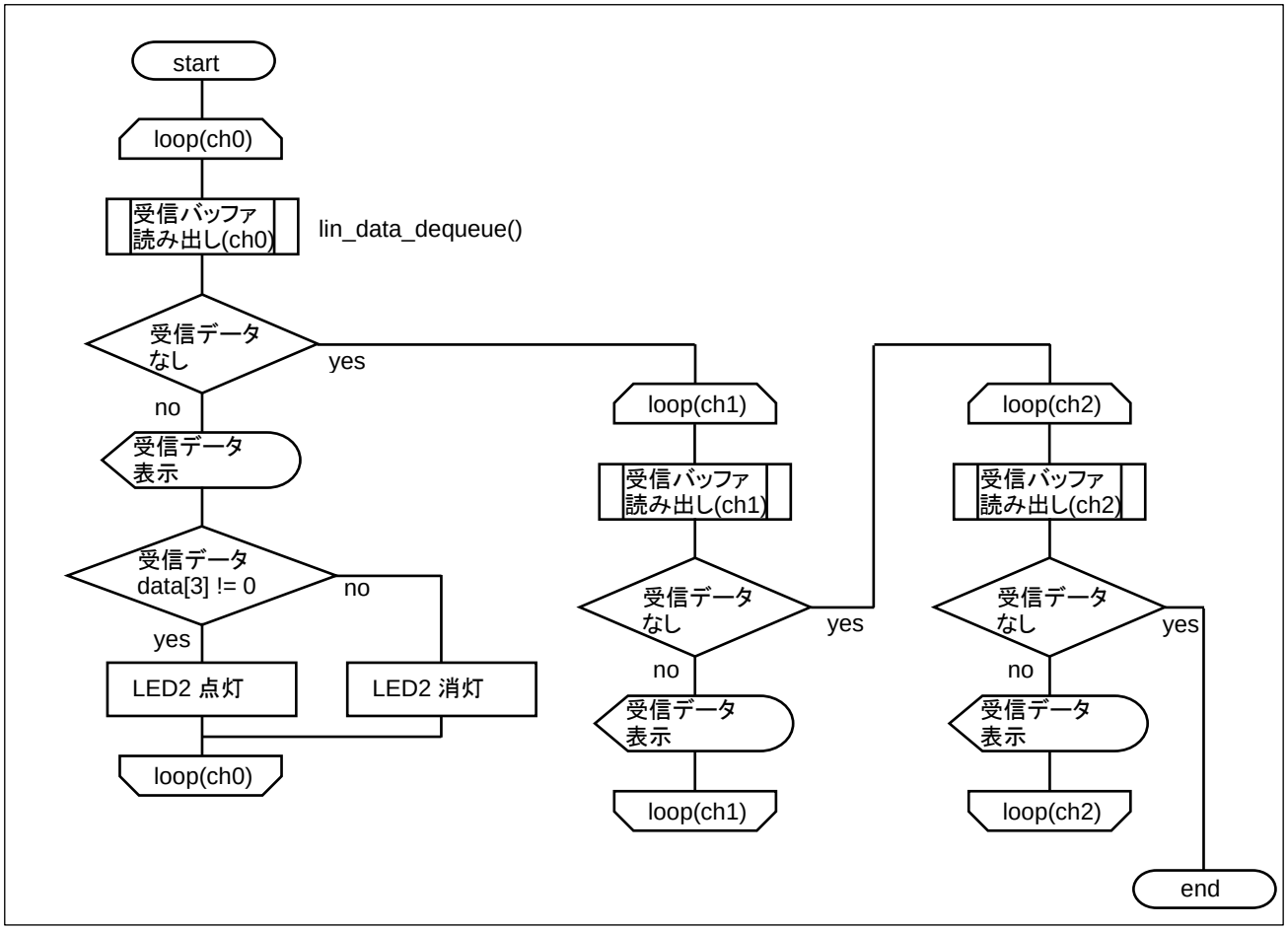
MASTER ヘッダ送信(LIN-ID=0x34)

上記動作を、1 秒間隔で繰り返します。

1 秒毎の定期処理　intr.c 内　timer_rtc()



受信データ確認　main_sample.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(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-ch1(SLAVE), LIN-ch2(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。

2. サンプルプログラムに含まれる関数の使用の流れ

2.1. 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。

2.1.1. 初期化

```
lin_init(LIN_CH0, LIN_MASTER, 0x30);
```

1 つ目の引数は、LIN の ch を指定します。RL78/F25 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

2.1.2. 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(LIN_CH0, 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。

2.1.3. MASTER ヘッダ送信（SLAVE レスポンス要求）

```
lin_master_header_send(LIN_CH0, 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。

※[参考]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。

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）と、SLAVE 側が返送するバイト数が合っていないとエラーとなり、データを受信しません。

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。

初期化(lin_init)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、MASTER レスポンス送信(lin_master_header_response_send)、MASTER ヘッダ送信(lin_master_header_send)を任意のタイミングで実行して構いません。

2.2. 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。

2.2.1. 初期化

```
lin_init(LIN_CH1, LIN_SLAVE, 0x31);
```

1 つ目の引数は、LIN の ch を指定します。RL78/F25 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。

2.2.2. 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(LIN_CH1, 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

2.2.3. 受信待機

```
lin_slave_header_receive(LIN_CH1);
```

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、、初期化(lin_init()実行)後 1 回実行してください。

初期化, lin_init()は最初に 1 回行ってください。レスポンスデータ設定(lin_response_data_set())と受信待機(lin_slave_header_receive())はどちらが先でも問題ありません。

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

2.3. 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、リングバッファ構造のグローバル変数(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。)

※本来はそのようなケースはないはずですが、MASTER に設定したポートは「他のデバイスが送信したヘッダ」と「他のデバイスが送信したヘッダに対するレスポンス」に関しては受信しません

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。

g_lin_recv_buf は 2 次元の配列となっており、g_lin_recv_buf[ch][index]の構成。index 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。（ch は、RL78/F25 の場合 3）

g_lin_recv_buf 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_data_dequeue(LIN_CH0, &lin_msg);
```

の様に行います。

受信データがある場合、
ret == LIN_RET_SUCCESS(0)
となります。受信しているデータが無い場合、
ret == LIN_RET_NODATA(-1)
となります。
（ret != LIN_RET_NODATA(-1)の間 lin_data_dequeue()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 == LIN_RETFLAG_LOST_DATA(0x80)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 == LIN_RETFLAG_LOST_DATA(0x80)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 3.3 節参照

3. 関数仕様

・ソースファイル構成

フォルダ	ファイル	説明
usr_src¥intr		割り込み処理フォルダ
	intr.c	TAU00, RTC, A/D 変換割り込み関数
	intr.h	割り込み関数ヘッダファイル
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_intr.c	LIN 割り込み処理関数
	lin_intr.h	LIN 割り込み関数ヘッダファイル
	lin_operation.h	速度や ID 等の動作に関わる定義
usr_src¥main		メイン関数フォルダ
	main_monitor.c	モニタプログラムメイン関数（5.3 節）
	main_sample.c	通常のサンプルプログラム
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
usr_src¥setting		設定ファイルフォルダ
	board.h	ボード定義ヘッダファイル
	program_select.h	プログラム選択ヘッダファイル

3.1. ヘッダファイルで定義している定数

3.1.1. lin.h の定数定義

lin.h

```
/*-----------------------------------------------------------------------
    定数定義（ユーザ設定箇所）
-----------------------------------------------------------------------*/
//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決めてしまう（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 //2のべき乗の値を指定すると有利（処理速度が若干速くなります）

//受信リングバッファが溢れた場合の動作
#define LIN_RECV_BUF_OVERRIDE //定義時はバッファフルで受信時古いデータが捨てられる動作，未定義時はバッファフルで受信時新しいデータが格納されない動作

//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
```

lin.h（続き）

```
/*-----------------------------------------------------------------------
    定数定義
-----------------------------------------------------------------------*/
//インデックスの計算アルゴリズム
#if (LIN_RECV_BUF_SIZE & (LIN_RECV_BUF_SIZE - 1)) != 0
#define LIN_RECV_BUF_INDEX_CALC_MOD //LIN_RECV_BUF_SIZEを2のべき乗以外の値とした場合は本定数を定義（未定義の場合は論理積、定義の場合は余剰で計算）
#endif

//モード
#define LIN_MASTER 0
#define LIN_SLAVE 1

//LIN-ch数
#define LIN_CH_NUM 3 //RL78/F25の場合3ch
#define LIN_CH0 0
#define LIN_CH1 1
#define LIN_CH2 2

//TX&RXフラグ
#define LIN_INT_REST_1 1 //あと割り込みが1回来る事を期待
#define LIN_INT_REST_2 2 //あと割り込みが2回来る事を期待

//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 130 //2400bps, 9600bps, 19200bps向け 1/40MHz×130×1(LCKS:fa)×16 = 52us(=1/19200)
#define LIN_SPEED_LBRP1 120 //10417bps向け 1/40MHz×120×16×2(LCKS:fd)×16 = 96us(=1/10417)

//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_PROMPT 0 //起動時にキーボードからの入力で選択
#define LIN_OPERATION_TIMER 1 //タイマ動作
#define LIN_OPERATION_UART 2 //UARTコマンド入力に応じて動作

//ヘッダ送信とレスポンス送信を別々に行うフレームセパレートモード時に定義（通常はヘッダ送信とレスポンス送信を1回の送信要求で処理）
//#define LIN_FRAME_SEPARATE_MODE //コメントアウト

//戻り値
#define LIN_RET_SUCCESS 0 //成功
#define LIN_RET_NODATA -1 //データなし
#define LIN_RET_ARGUMENT_ERROR -2 //引数エラー
#define LIN_RET_IN_USE -3 //送信中
#define LIN_RET_OVERFLOW -4 //リングバッファフル
#define LIN_RET_PIOR_ERROR -5 //端子設定(PIOR)エラー
#define LIN_RETFLAG_LOST_DATA 0x80 //失われたデータあり（戻り値とorを取る）

//TRUE/FALSE
#define TRUE    1
#define FALSE   0
```

#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。

2 のべき乗の値を設定した場合、インデックスの計算が若干速くなります。

#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、割り込みコールバック関数は ch 毎に別です）

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にユーザ処理を記載してください。

```
#define LIN_SPEED_LBRP0 130
#define LIN_SPEED_LBRP1 120
```

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
```

LCKS	fx	分周比	プリスケーラ
0	fa	1/1	LBRP0
1	fb	1/2	LBRP0
2	fc	1/8	LBRP0
3	fd	1/2	LBRP1

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、

1/40MHz(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: 1/40MHz ×1 × 130 × 16 = 52[us], (LCKS=0, fa : 1 ×LBPR0(130))
9600bps: 1/40MHz ×2 × 130 × 16 = 104[us], (LCKS=1, fb : 2 ×LBPR0(130))
2400bps: 1/40MHz ×8 × 130 × 16 = 416[us], (LCKS=2, fc : 8 ×LBPR0(130))

・LBRP1 側を使用
10417bps: 1/40MHz ×2 × 120 × 16 = 96[us], (LCKS=3, fd : 2 ×LBPR1(120))

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。

ここで設定する速度は、MASTER 側の速度を決めます。（SLAVE 側は速度は自動調整です。）

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。

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

lin_message 構造体は、

・LIN-ID（1 バイト）

・データ（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。

3.1.2. 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_SPEEDLIN_SPEED_2400 //2400bps
#define LIN_SPEEDLIN_SPEED_9600 //9600bps
//#define LIN_SPEEDLIN_SPEED_10417 //10417bps
//#define LIN_SPEEDLIN_SPEED_19200 //19200bps

//割り込み優先度，0:最高優先度～3:最低優先度
#define LIN_SEND_INTERRUPT_PRIORITY 1 //送信割り込み優先度
#define LIN_RECEIVE_INTERRUPT_PRIORITY 1 //受信割り込み優先度
#define LIN_STATUS_INTERRUPT_PRIORITY 2 //ステータス割り込み優先度

//多重割り込みを許可
//#define LIN_MULTIPLE_INTERRUPT_EN

//LIN ID
#define LIN_MASTER_ID 0x30 //MASTER動作時に設定するID
#define LIN_SLAVE_ID1 0x31 //SLAVE動作時に設定するID
#define LIN_SLAVE_ID2 0x32 //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
#define LIN_OPERATION_MODELIN_OPERATION_PROMPT //起動時にキーボードからの入力で動作モードを選択
//#define LIN_OPERATION_MODELIN_OPERATION_TIMER //下記のインターバル時間毎にヘッダ送信
//#define LIN_OPERATION_MODELIN_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）
*/

//定義時端末にデバッグ表示を行う
#define SCI_DEBUG

//定義時割り込みを端子でモニタする
#define LIN_INTERRUPT_DEBUG
```

```
#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。

※MASTER の場合本速度設定が有効になります
SLAVE の場合、ヘッダに含まれる sync フィールドのパルス幅に従い通信速度が決定されます

```
#define LIN_SEND_INTERRUPT_PRIORITY  1
#define LIN_RECEIVE_INTERRUPT_PRIORITY  1
#define LIN_STATUS_INTERRUPT_PRIORITY  2
```

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、それぞれの割り込みの優先度を個別に設定可能です。（LIN-ch0~LIN-ch2 の ch 毎の割り込み優先度に差を付けたい場合は、lin_intr.c の変更が必要です。）

```
// define LIN_MULTIPLE_INTERRUPT_EN
```
[コメントアウト]

LIN の割り込みで多重割り込みを有効にしたい場合はコメントアウトを外してください。

```
#define LIN_MASTER_ID  0x30
#define LIN_SLAVE_ID1  0x31
#define LIN_SLAVE_ID2  0x32
```

サンプルプログラムでは、LIN-ch0 を MASTER, ID=0x30。LIN-ch1 を SLAVE, ID=0x31。LIN-ch2 を SLAVE, ID=0x32、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（UART モード）の場合の、コマンド 1~9。タイマ動作モードの 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

タイマ動作モードの SLAVE に対しヘッダ送信を行う ID の数です。（UART 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_PROMPT
// #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。いずれかを有効化してください。
LIN_OPERATION_PROMPT（デフォルト）は、起動時に動作モードをキーボードから入力して決定します。
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。LIN_OPERATION_UART は、UART モードで動作、

#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=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/F25 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、この値を最小で 1 に設定（50ms 毎にヘッダ送信を行う設定）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。

#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(ch0) : LIN-ID=0x30 data=0x000200000BB40000
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000200000BB40000 sum=0x4D

--
LIN header send(ch0) : LIN-ID=0x31
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=send]
LIN response data send finished(ch1) : LIN-ID=0x31 data=0x00040000083E0000
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x31 data=0x00040000083E0000 sum=0x04
```

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、

・ch1(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

・ch1(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－

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000100000BD20000
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch1)=stat LST1=0x88 LEST1=0x80]
[intr(ch0)=send]
LIN response data send finished(ch0)
```

MASTER ヘッダ・レスポンス送信

例えば、ch1(SLAVE)側を受信待機せずに、ch0(MASTER)側からデータ送信した場合、ch1(SLAVE)側はデータ受信してない（LIN data receive(ch1)の表示が出ない）ものの、

[intr(ch1)=recv]

となっているので、ch1 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(ch1)=stat LST1=0x88 LEST1=0x80]
ch1(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（LEST1=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－

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000500000BB40000
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
```

MASTER ヘッダ・レスポンス送信

なお、ch0 と ch1 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。赤字の行を除いた、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、ch1 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。（デフォルトでは、デバッグ表示を有効化しています。）

```
#define  LIN_INTERRUPT_DEBUG
```

定義時割り込みを端子でモニタできる様になります（詳細は 5 章参照）。

3.2. 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:

LIN_RET_SUCCESS(0): 正常終了
LIN_RET_ARGUMENT_ERROR(-2): 引数エラー
LIN_RET_PIOR_ERROR(-5): LIN 端子設定エラー（PIOR 設定が正しくない）

補足:

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
MASTER 時は、ここで指定した ID はグローバル変数に保存されますので設定した ID 値の参照は可能ですが本サンプルプログラムでは使用していません

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(0~1)

*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:

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_ARGUMENT_ERROR(-2): 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:

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_ARGUMENT_ERROR(-2): 引数エラー

LIN_RET_IN_USE(-3): 送信中（指定した ch で送信中）

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エラーとなり、データ受信は行われ
ません（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:

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_ARGUMENT_ERROR(-2): 引数エラー

LIN_RET_IN_USE(-3): 送信中（指定した ch で送信中）

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:

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_ARGUMENT_ERROR(-2): 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→lin_init()では、受信待機状態にはしないという設計思想です

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:

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_ARGUMENT_ERROR(-2): 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_data_enqueue

概要：受信データを受信バッファに格納する関数

宣言：

int lin_data_enqueue(unsigned int buf_num, lin_message *msg)

説明：

・受信バッファにデータの格納

を行います

引数：

buf_num: バッファ番号

*msg: lin_message 構造体

lin_message 構造体

unsigned char id;	データのID
unsigned char data[8];	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：

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_OVERFLOW(-4): 受信バッファオーバーフロー

補足：

データを受信したタイミングで実行される受信割り込み関数内で、本関数を使い受信バッファ(g_lin_recv_buf)にデータを格納しています

・LIN_RECV_BUF_OVERRIDE 定義時（デフォルト）

未読み出しの(LIN_RECV_BUF_SIZE)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れます

・LIN_RECV_BUF_OVERRIDE 未定義時

未読み出しの(LIN_RECV_BUF_SIZE)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、

受信データは受信バッファに格納せず LIN_RET_OVERFLOW を返します

本サンプルプログラムでは、受信バッファを 3 組（3×LIN_RECV_BUF_SIZE）用意して、buf_num=LIN-ch としています

lin_data_dequeue

概要：受信データ読み出し関数

宣言：

int lin_data_dequeue(unsigned int buf_num, 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：

buf_num: バッファ番号

*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：

LIN_RET_SUCCESS(0): 正常終了

LIN_RETFLAG_LOST_DATA(0x80): 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）

LIN_RET_NODATA(-1): 未読み出しのデータなし

補足：

未読み出しの(LIN_RECV_BUF_SIZE)数のデータが溜まっている状態で次のデータを受信した場合、受信バッファに格納されている一番古いいデータが上書きされます（LIN_RECV_BUF_OVERRIDE 定義時）
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 0x80 を返します（その場合でも、読み出しデータは有効です）

lin_in_queue_size

概要:受信データ数読み出し関数

宣言:

int lin_in_queue_size(unsigned int buf_num)

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:

buf_num: バッファ番号

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

lin_queue_clear

概要:受信データクリア関数

宣言:

void lin_queue_clear(unsigned int buf_num)

説明:

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

を行います

引数:

buf_num: バッファ番号

戻り値:

なし

lin_id_encode

概要:ID エンコード関数

宣言:

unsigned char lin_id_encode(unsigned char id)

説明:

・ID を PID に変換

を行います

引数:

id: LIN-ID

戻り値:

PID 値

lin_id_decode

概要:ID デコード関数

宣言:

unsigned char lin_id_decode(unsigned char encoded_id)

説明:

・PID を ID に変換

を行います

引数:

encoded_id: PID 値

戻り値:

ID 値

－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：

int 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：

LIN_RET_SUCCESS(0): 正常終了

LIN_RET_PIOR_ERROR(-5): LIN 端子設定エラー

補足：

LIN の端子は

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

－本キットでの使用端子－

	LTXDn	LRXDn	EN
LIN-ch0	P13	P14	P53
LIN-ch1	P120	P125	P01
LIN-ch2	P70	P71	P72

	LTXDn	LRXDn	備考
LIN-ch0	**P13** P42	**P14** P43	 PIOR44=1
LIN-ch1	P10 P106 **P120**	P11 P107 **P125**	 PIOR45=1, PIOR93=0 **PIOR45=1, PIOR93=1**
LIN-ch2	P155 **P70**	P154 **P71**	 **PIOR94=1**

LIN の使用端子は、一番上に記載している端子がデフォルトで、デフォルト以外の端子を選択する場合は、PIOR レジスタの設定が必要です。（**太字**が本キットでの使用端子）

※LIN-ch0 側は、PIOR レジスタの設定は不要です。

LIN-ch1 側は、PIOR45=1, PIOR93=1、LIN-ch2 側は PIOR94=1 の設定が必要です。

本サンプルプログラムの端子定義は、board.h 内で定義されており、以下の様になっています。

```
//LINのポート(LIN0)
#define LIN0_TX_P13
#define LIN0_RX_P14
#define LIN0_EN_P53

//LINのポート(LIN1)
#define PIOR45 1
#define PIOR93 1
#define LIN1_TX_P120
#define LIN1_RX_P125
#define LIN1_EN_P01

//LINのポート(LIN2)
#define PIOR94 1
#define LIN2_TX_P70
#define LIN2_RX_P71
#define LIN2_EN_P72
```

[参考]

本サンプルプログラムを当社製、HSBRL78F25-64 で使用する場合の端子設定。

```
//LINのポート(LIN0)
#define PIOR44 1
#define LIN0_TX_P42
#define LIN0_RX_P43
#define LIN0_EN_P32
```

HSBRL78F25-64 では、PIOR レジスタの設定が必要で、上記の様な端子設定とする必要があります。(PIOR44 の定数値を 1 に設定する事により、lin_init_port()関数内で、PIOR44=1 の設定が実行されます。)

#define PIOR44 1 の設定が無いにも拘らず
#define LIN0_TX_P42 を定義した場合、lin_init_port()関数は、戻り値-5(PIOR 設定と、使用端子のミスマッチ)のエラーを返します。

3.3. 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。
main_sample.c 内には、割り込みコールバック関数が定義されていますので、そちらの記載を参考にしてください。

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

3.4. 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 MASTER/SLAVE 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：

一連の処理であと何回送信割り込みが生じるか、残りの送信割り込みの期待値の回数がセットされます
MASTER 動作時、レスポンスデータ送信時　LIN_INT_REST_2(2)
MASTER 動作時、ヘッダのみ送信時, SLAVE 動作時レスポンスデータ送信時　LIN_INT_REST_1(1)
※詳しくは 4.5 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：

一連の処理であと何回受信割り込みが生じるか、残りの受信割り込みの期待値の回数がセットされます
MASTER 動作時、ヘッダのみ送信, SLAVE 動作時レスポンスデータ受信時　LIN_INT_REST_1(1)
※詳しくは 4.5 節参照

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

用途：受信リングバッファ

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_recv_buf_write_index

用途：受信リングバッファの書き込みインデックス

宣言:

uint64_t g_lin_recv_buf_write_index[LIN_CH_NUM];

説明:

受信リング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_read_index

用途:受信リングバッファの読み出しインデックス

宣言:

uint64_t g_lin_recv_buf_read_index[LIN_CH_NUM];

説明:

受信リング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_read_index == g_lin_recv_buf_write_index の場合、未読データがない事を示します

[参考]

受信バッファのインデックス変数は、uint64_t(64bit 符号なし)としています。

仮に、50ms 毎にデータを受信した場合で、32bit 変数を用いた場合($50 \times 10^{-3} \times 2^{32}/3600/24/365 = 6.8$)、6.8 年でインデックス溢れが生じます。6.8 年ですので、この場合は 32bit 変数でも十分かとは思います。

(64bit 変数では、2.9×10^{10} 年です。64bit 変数を使った場合、インデックスが溢れる事に関しては考えなくても良いという事となります。RL78(16bitCPU)で 64bit 変数を取り扱う場合、オーバヘッドも大きいですので、パフォーマンス重視の場合は、unsigned long 型(32bit 変数)に置き換えてください。)

g_lin_recv_buf_override

用途:受信リング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:

受信リング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)

g_lin_receive_id

用途：受信 ID 値の保存

宣言：

unsigned char g_lin_receive_id[LIN_CH_NUM];

説明：

受信した ID を保存する変数

補足：

通常のサンプルプログラム(main_sample.c)では未使用

（モニタプログラム(main_monitor.c）で表示用に使用）

g_lin_receive_timing

用途：受信時のタイミングパラメータの保存

宣言：

unsigned short g_lin_timing[LIN_CH_NUM];

説明：

SLAVE 時、オートボーレート設定で受信した信号のタイミングパラメータ（ビットレートカウンタ）を保存する変数

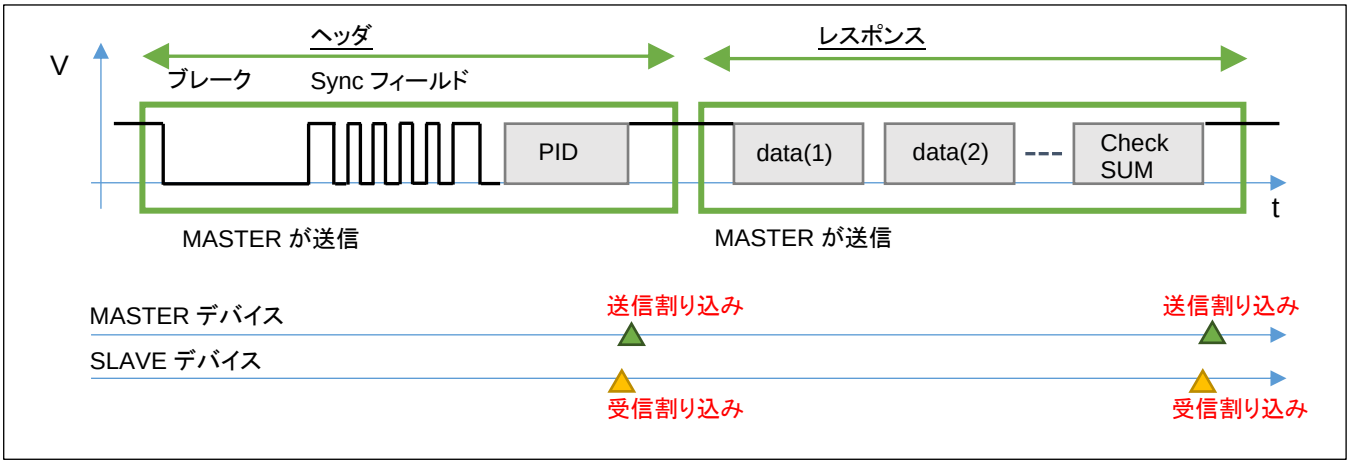
補足：

通常のサンプルプログラム(main_sample.c)では未使用

（モニタプログラム(main_monitor.c）で表示用に使用）

4. 割り込み処理

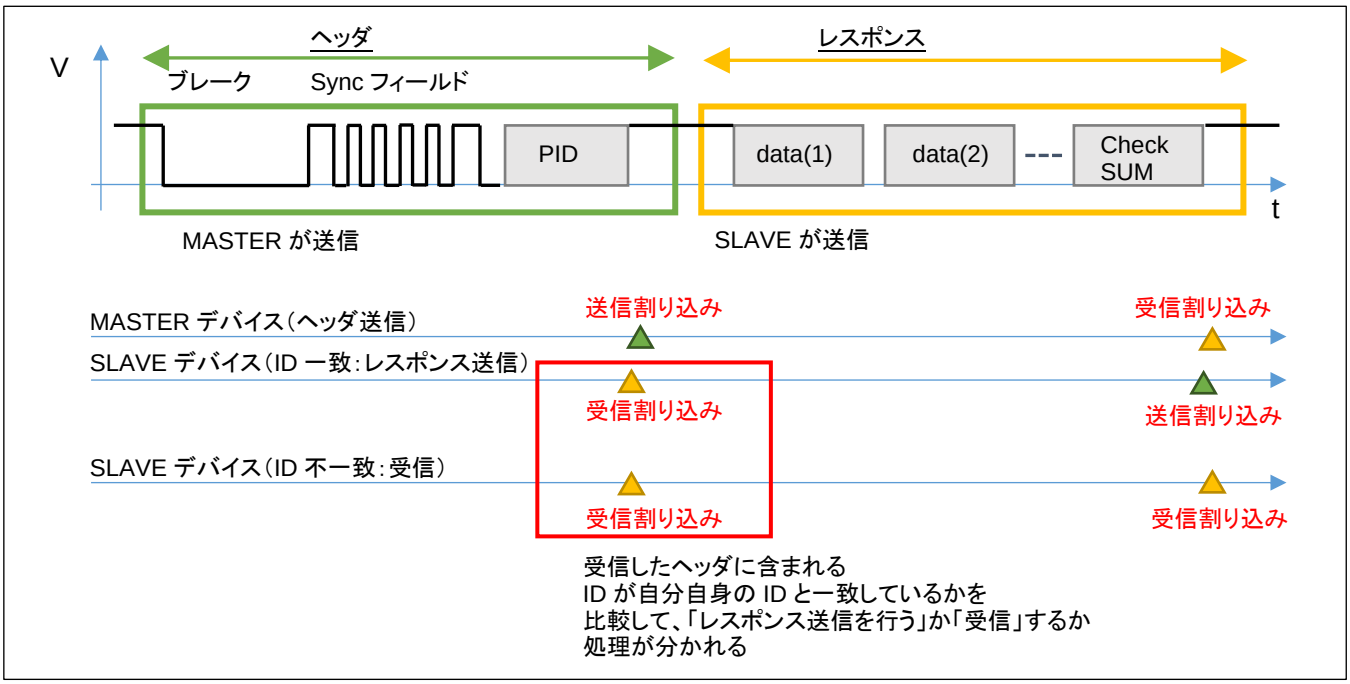
4.1. MASTER レスポンス送信動作



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。

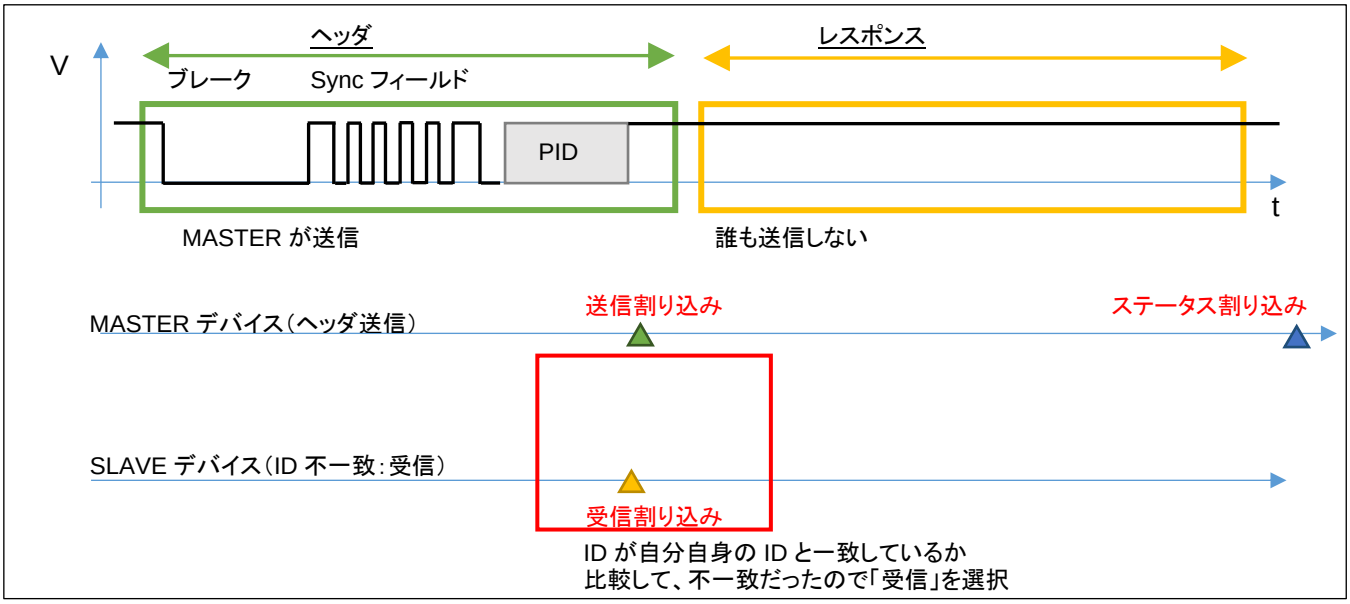
4.2. MASTER ヘッダ送信動作



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。

4.3. 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。

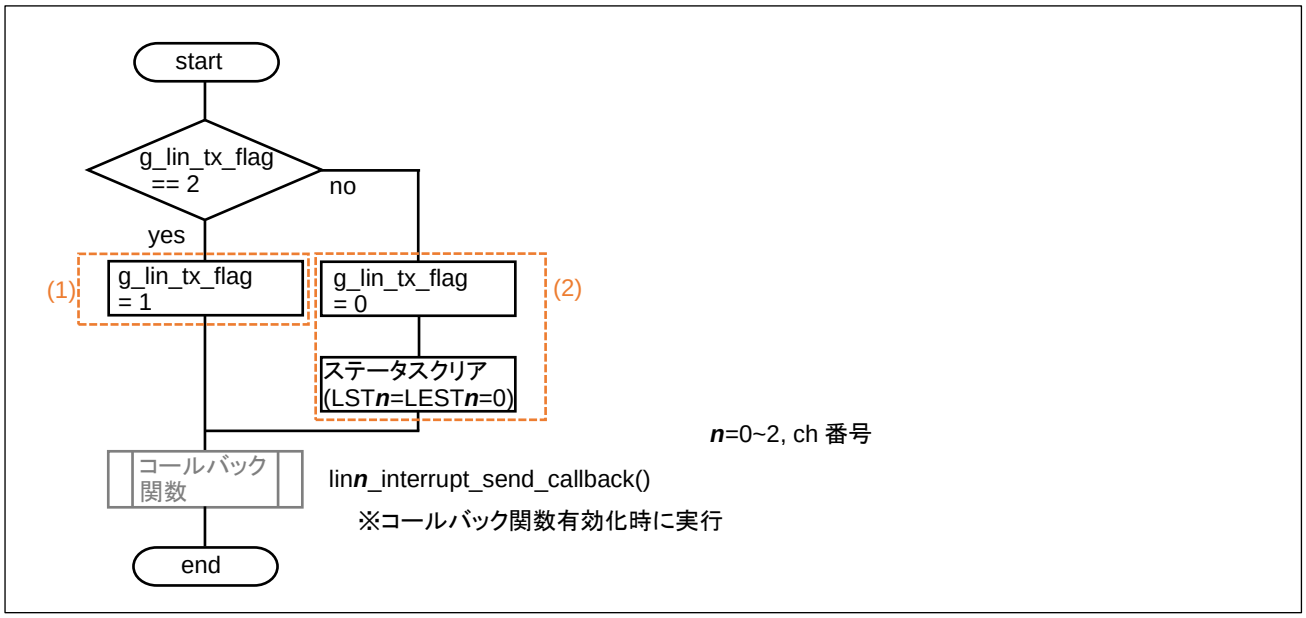
（エラーステータス LESTx=0x04，タイムアウトエラーとなります）

※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

4.4. 割り込み関数

4.4.1. 送信割り込み

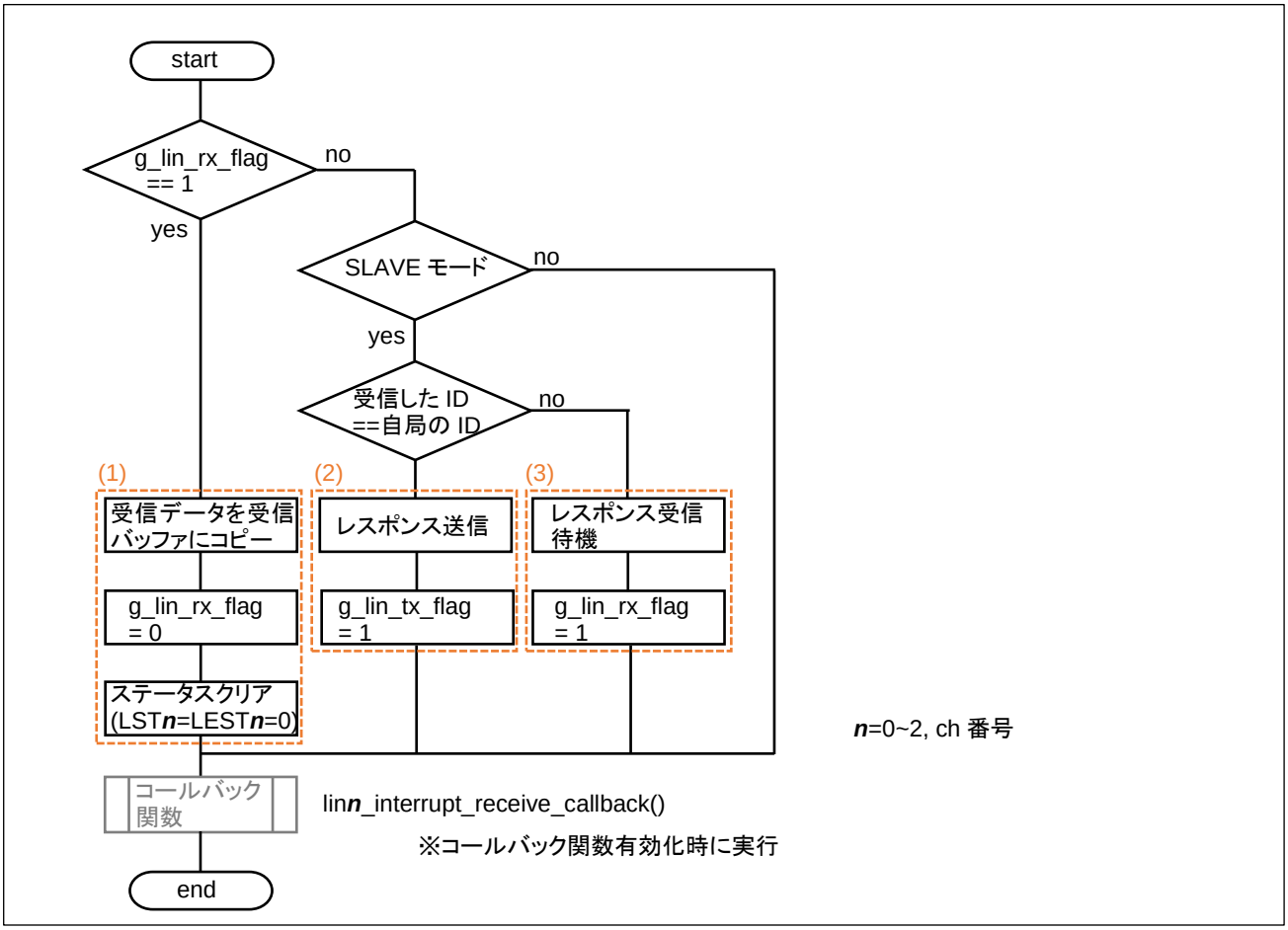
lin_intr.c 内 lin***n***_interrupt_send()（n=0~2, ch 番号）



・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。

4.4.2. 受信割り込み

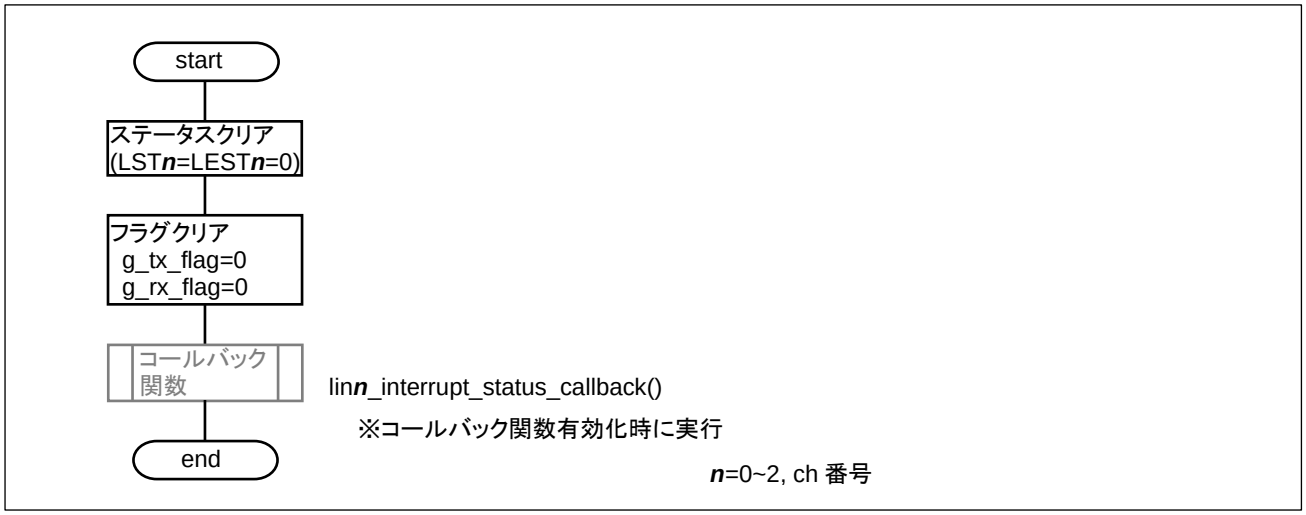
lin.c 内 lin***n***_interrupt_receive()（n=0~2, ch 番号）



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

4.4.3. 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。

4.5. 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。

どちらのフラグ共、本変数がセットされた時点で、一連の動作中にあと何回の割り込みが入るかを示しています。
LIN_INT_REST_1(1) あと 1 回割り込みが入る事が期待値
LIN_INT_REST_2(2) あと 2 回割り込みが入る事が期待値
（g_lin_tx_flag は 0,1,2 のいずれか、g_lin_rx_flag は 0 か 1 です。）

g_lin_tx_flag は、残りの送信割り込みの回数。g_lin_rx_flag は、残りの受信割り込みの回数です。どちらの変数も、割り込みで減算されて、0 になれば一連の動作完了です。

4.5.1. MASTER レスポンス送信の際の MASTER 側

(1)lin_master_header_response_send()

[g_lin_tx_flag が 0 でなければ、送信中としてエラーを返す]

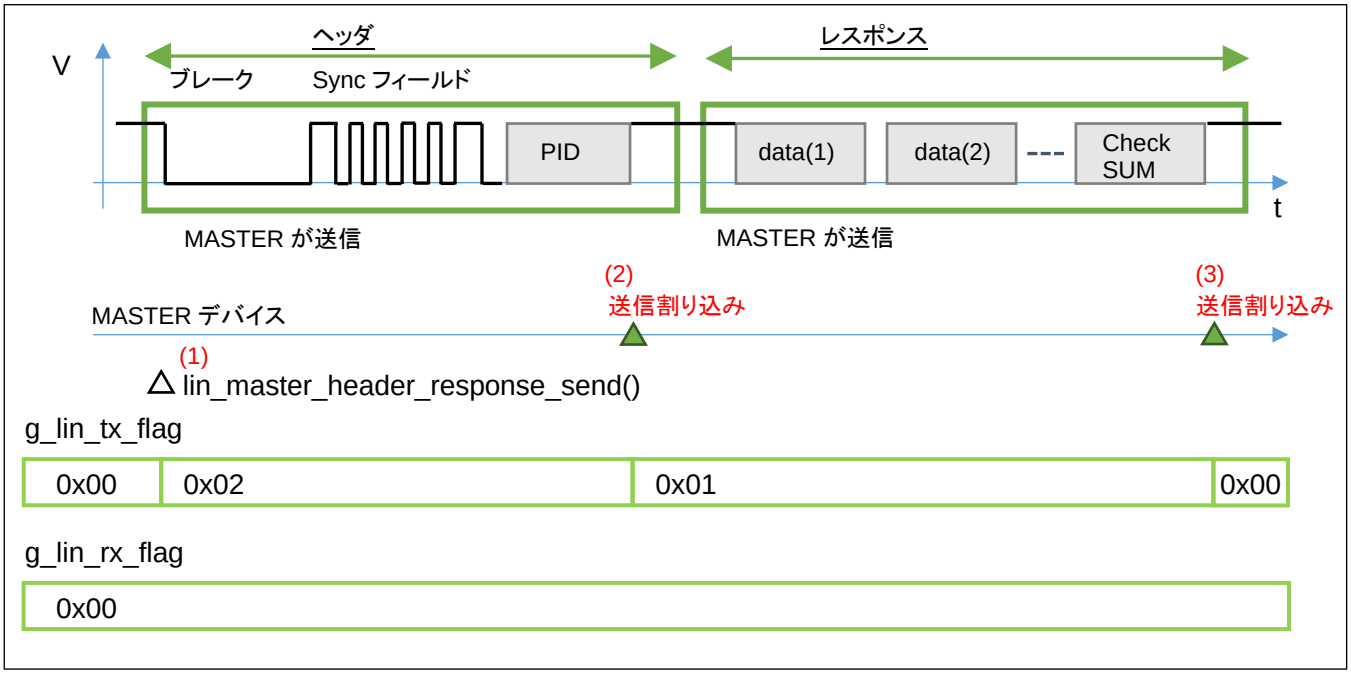
・g_lin_tx_flag を LIN_INT_REST_2(=2)にセット

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

〇g_lin_tx_flag == 2
→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
・g_lin_tx_flag を LIN_INT_REST_1(=1)にセット（あと 1 回送信割り込みに来るのが期待値）

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

〇g_lin_tx_flag == 1（g_lin_tx_flag == 2 以外）
→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 を 0 とする（これ以降送信割り込みには来ない事が期待値）



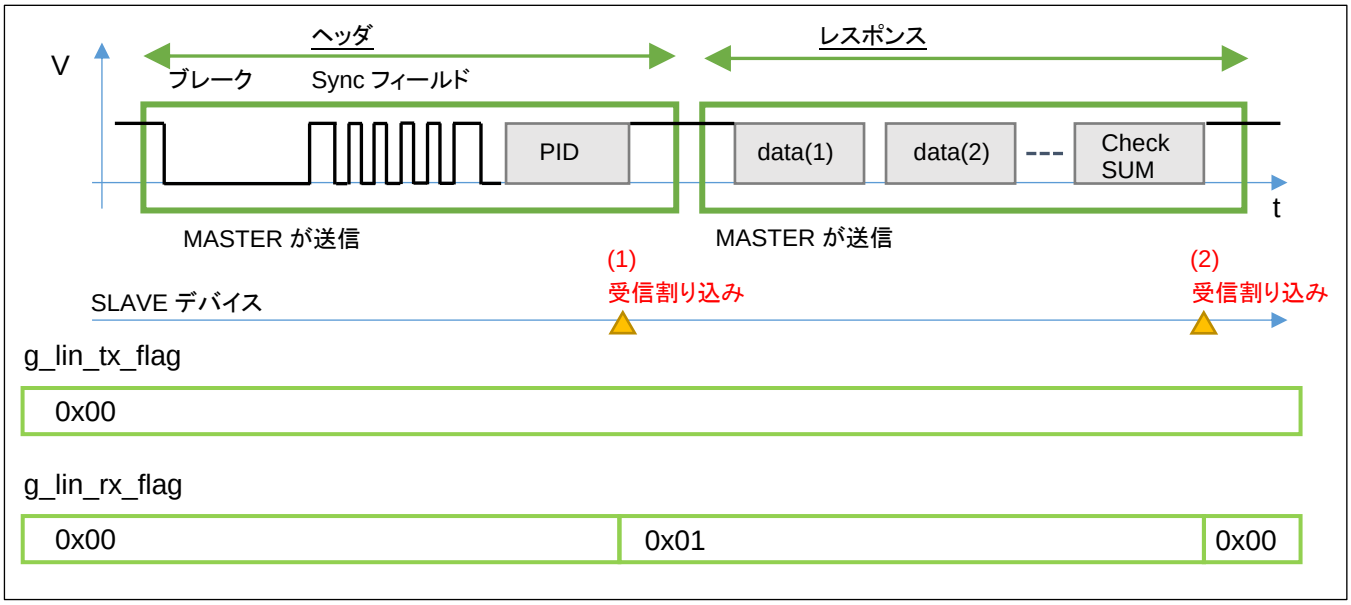
4.5.2. MASTER レスポンス送信の際の SLAVE 側

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

〇g_lin_rx_flag == 0（g_lin_rx_flag == 1 以外）
→ヘッダ受信のタイミングでの受信割り込み
→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 = LIN_INT_REST_1(=1)にセット（あと 1 回受信割り込みに来るのが期待値）

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

〇g_lin_rx_flag == 1
→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 を 0 とする（これ以降受信割り込みには来ない事が期待値）



4.5.3. MASTER ヘッダ送信の際の MASTER 側

(1)lin_master_header_send()

[g_lin_tx_flag が 0 でなければ、送信中としてエラーを返す]

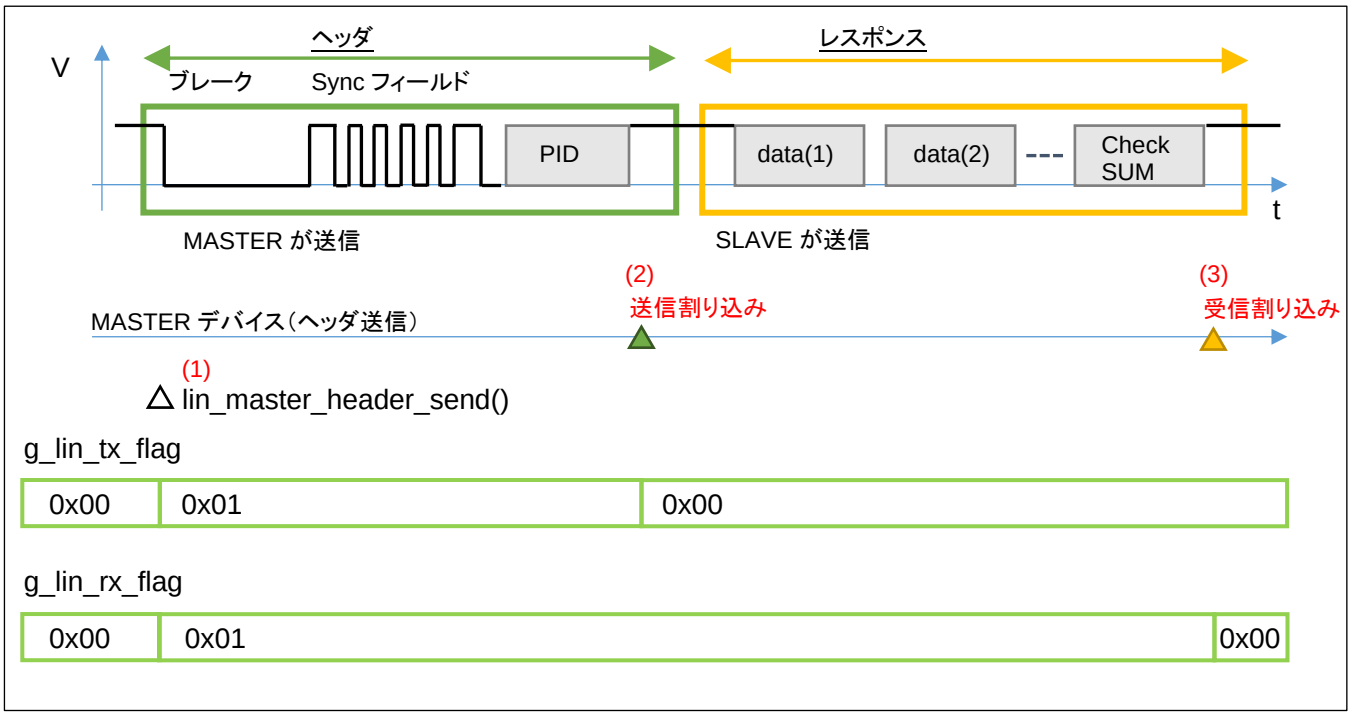
・g_lin_tx_flag = LIN_INT_REST_1(=1)にセット（あと 1 回送信割り込みに来るのが期待値）
・レスポンス受信を行うので、g_lin_rx_flag = LIN_INT_REST_1(=1)にセット（あと 1 回受信割り込みに来るのが期待値）

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

〇g_lin_tx_flag == 1（g_lin_tx_flag == 2 以外）
→1 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
・g_lin_tx_flag を 0 とする（これ以降送信割り込みには来ない事が期待値）

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

〇g_lin_rx_flag == 1
→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 を 0 とする（これ以降受信割り込みには来ない事が期待値）



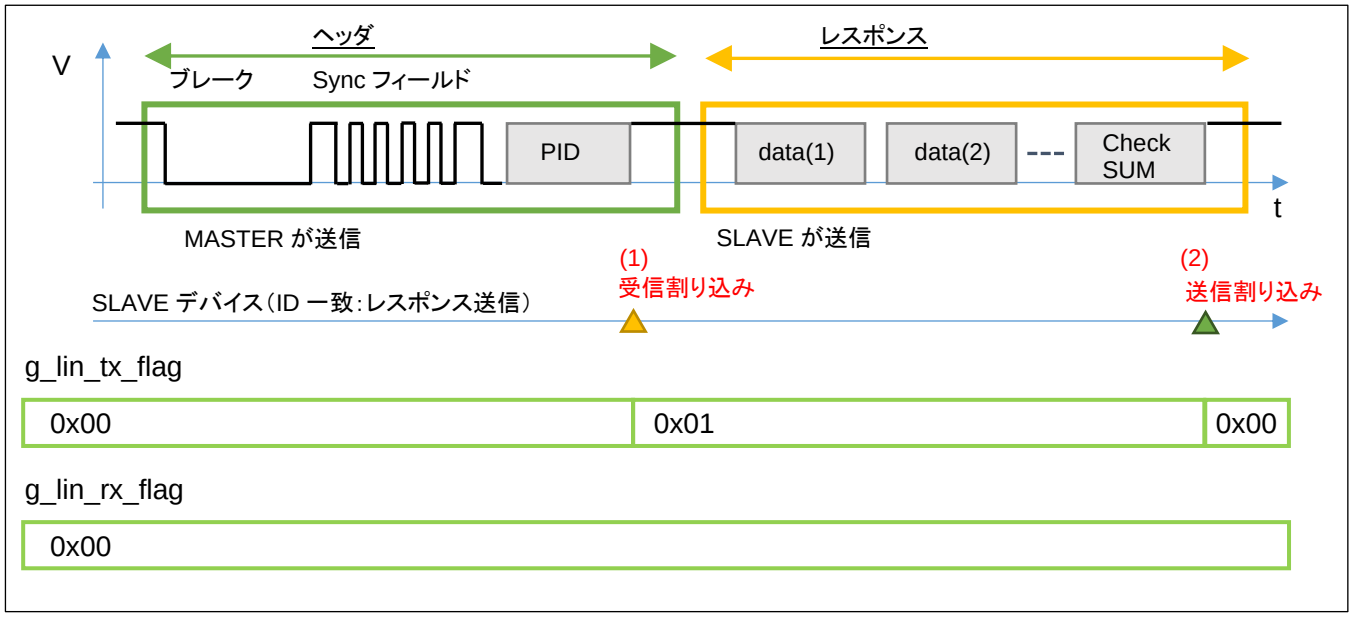
4.5.4. MASTER ヘッダ送信の際、レスポンス送信を行う SLAVE 側

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

〇g_lin_rx_flag == 0（g_lin_rx_flag == 1 以外）
→ヘッダ受信のタイミングでの受信割り込み
→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 = LIN_INT_REST_1(=1)にセット（あと 1 回送信割り込みに来るのが期待値）

(2)送信割り込み関数[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 を 0 とする（これ以降送信割り込みには来ない事が期待値）



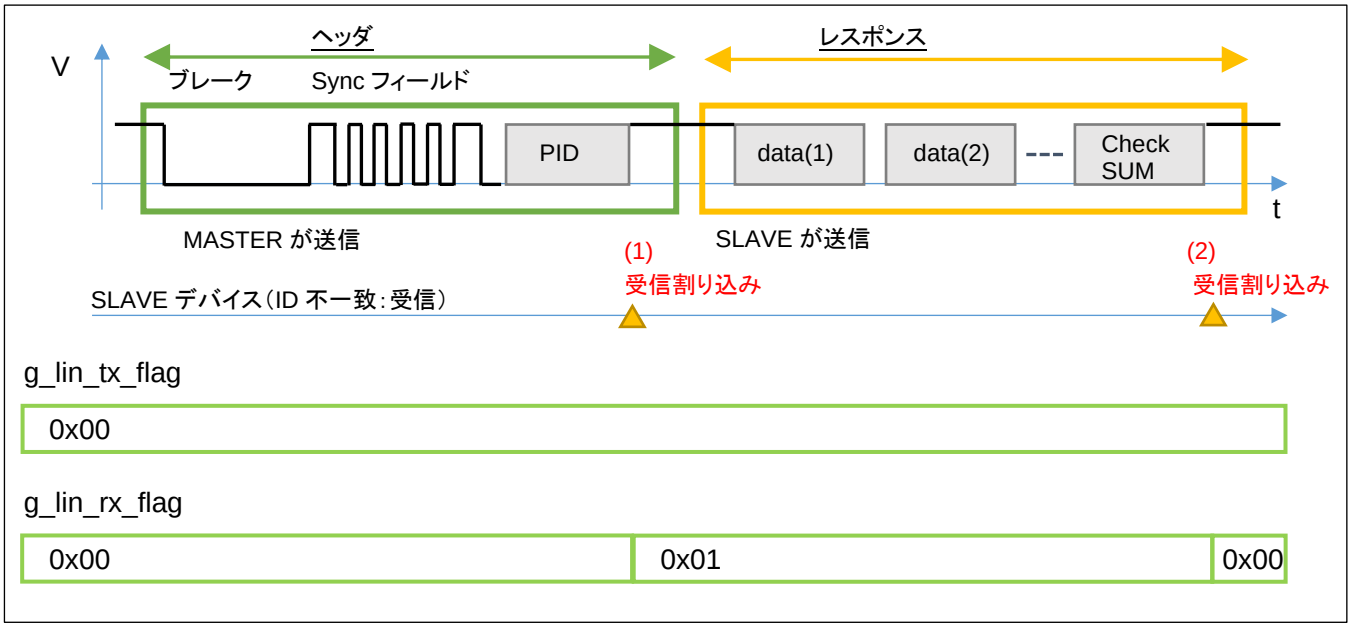
4.5.5. MASTER ヘッダ送信の際、レスポンス受信を行う SLAVE 側

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

〇g_lin_rx_flag == 0（g_lin_rx_flag == 1 以外）
→ヘッダ受信のタイミングでの受信割り込み
→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 がが不一致だったので、
<u>レスポンス受信を行うケース</u>
・レスポンス受信を行うので、g_lin_rx_flag = LIN_INT_REST_1(=1)にセット（あと 1 回受信割り込みに来るのが期待値）

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

〇g_lin_rx_flag == 1
→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 を 0 とする（これ以降受信割り込みには来ない事が期待値）



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

g_lin_tx_flag	状況	アクション
==2	MASTER レスポンス送信のヘッダ送信完了	g_lin_tx_flag=1 にセット
==1	MASTER レスポンス送信のレスポンス送信完了 MASTER ヘッダ送信のヘッダ送信完了 SLAVE レスポンス送信のレスポンス送信完了	g_lin_tx_flag=0 にセット

受信割り込み関数では、

g_lin_rx_flag

g_lin_rx_flag	状況	アクション
==1	MASTER ヘッダ送信のレスポンス受信完了 SLAVE レスポンス受信完了	g_lin_rx_flag=0 にセット
==0	SLAVE ヘッダ受信完了	SLAVE レスポンス送信時は g_lin_tx_flag=1 にセット SLAVE レスポンス受信時は g_lin_rx_flag=1 にセット

という、フラグによる処理分けを行っています。

割り込み発生時に、

・次に送信割り込みが生じる予定の場合

→g_lin_tx_flag=1 にセット

・次に受信割り込みが生じる予定の場合

→g_lin_rx_flag=1 にセット

・以降割り込みが生じる予定がない場合

→g_lin_tx_flag = g_lin_rx_flag=0 にセット

というルールです。

MASTER ヘッダ・レスポンス送信時は、送信割り込みが 2 回生じる予定なので、

→g_lin_tx_flag = 2 にセット

MASTER ヘッダ送信時は、送信割り込みが 1 回、受信割り込みが 1 回生じる予定なので、

→g_lin_tx_flag=1, g_lin_rx_flag=1 にセット

としています。

MASTER の場合、master_header_response_send(), master_header_send()内で初期値を設定。

SLAVE の場合、1 回目の受信割り込み内で初期値を設定。

5. プログラムのデバッグに関して

5.1. SCI デバッグ表示

lin.h 内の

#define SCI_DEBUG

を有効にすることにより、端末表示がデバッグモードとなります。

```
--
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x000500000BB60000
[intr(ch1)=recv]
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x000500000BB60000 sum=0x48
```

MASTER ヘッダ送信
（SLAVE レスポンス送信）

割り込みが入ったタイミングで[intr...]の表示が出力されます。

5.2. 割り込みのポートデバッグに関して

送信、受信、ステータス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、lin.h 内の定数定義

#define LIN_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

LIN0 送信 (INTLIN0TRM)	LIN0 受信 (INTLIN0RVC)	LIN0 ステータス (INTLIN0STA)
P90(J1-32)	P91(J1-31)	P92(J1-30)

LIN1 送信 (INTLIN1TRM)	LIN1 受信 (INTLIN1RVC)	LIN1 ステータス (INTLIN1STA)
P93(J1-29)	P94(J1-28)	P95(J1-27)

LIN2 送信 (INTLIN2TRM)	LIN2 受信 (INTLIN2RVC)	LIN2 ステータス (INTLIN2STA)
P96(J1-26)	P97(J1-25)	P02(J1-20)

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。

5.3. モニタプログラムに関して

CD 内の、
BINARY¥RL78_LIN¥RL78_F25_100_LIN_MONITOR.mot （HSBRL78F25-100 向け）
BINARY¥RL78_LIN¥RL78_F25_80_LIN_MONITOR.mot （HSBRL78F25-80 向け）
は、LIN の通信モニタ用のサンプルプログラムです。

LIN バスを流れているデータのモニタや、LIN バスに対してヘッダ、レスポンスの送信を行う事ができます。

LIN-ch0(J7) データ送信用ポート(MASTER)
・ヘッダレスポンス送信（m コマンド）
ID=0x30（デフォルト）
送信データバイト数 1-8 バイト
送信データ（0x01 02 03 04 05 06 07 08, ソース内に記載）
送信ビットレート 9,600bps（lin_operation.h 内で変更可）
・ヘッダ送信（q~o コマンド）
ID=0x31~39 のヘッダ送信

LIN-ch1(J8) データ受信用ポート(SLAVE)
LIN-ch2(J9) データ受信用ポート(SLAVE)
オートボーレート（2.4~20kbps の信号受信可）
受信データバイト数 1-8 バイト（事前に設定要）
レスポンスデータ（0xFF FE FD FC FB FA F9 F8, ソース内に記載）
レスポンスを返すか受信専用とするかは、キーボードからの入力で変更可（デフォルトは返さない）

```
Copyright (C) 2026 HokutoDenshi. All Rights Reserved.

LIN Starter kit RL78/F25 Monitor sample program.

Command:
  1-8 : MASTER/SLAVE data packet byte size set(default=8)
  m : MASTER header/response send (ID=0x30)
  q : MASTER header send (ID=0x31)
  w : MASTER header send (ID=0x32)
  e : MASTER header send (ID=0x33)
  r : MASTER header send (ID=0x34)
  t : MASTER header send (ID=0x35)
  y : MASTER header send (ID=0x36)
  u : MASTER header send (ID=0x37)
  i : MASTER header send (ID=0x38)
  o : MASTER header send (ID=0x39)
  z : LIN-ch1(SLAVE) response ON/OFF (default=OFF)
  x : LIN-ch2(SLAVE) response ON/OFF (default=OFF)
  S : status print
  C : status clear
  M : message clear

>
```

キーボードからのコマンド

1-8: データサイズ指定コマンド
起動時のデフォルトは 8 バイト。送信、及び受信のバイト数を本コマンドで指定する。

m: ヘッダ・レスポンス送信
ID=0x30 でヘッダ・レスポンス送信を行う

q-o: ヘッダ送信
ID=0x31~0x39 でヘッダ送信を行う

x,z: SLAVE レスポンスデータを返す・返さないの切り替え

S: ステータス,エラーレジスタの表示

C: ステータス,エラーレジスタのクリア

M: メッセージクリア

・LIN-ch0（送信動作）

LIN-ch0(J7)は、送信用のポートになりますので、LIN-SLAVE 機器を接続して下さい。通信レートは、デフォルトでは 9,600bps です。（変更時は、lin_operation.h を編集してください。）

・ID=0x32 のヘッダ送信, w コマンドの実行例（バス上に、レスポンスを返す SLAVE が存在する場合）

```
LIN header send(ch0) : LIN-ID=0x32
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=recv]
LIN data receive(ch0) : LIN-ID=0x32 data=0xFFFEFDFCFBFAF9F8 sum=0xE9
```

・ID=0x33 のヘッダ送信, e コマンドの実行例（バス上に、レスポンスを返す SLAVE が存在しない場合）

```
LIN header send(ch0) : LIN-ID=0x33
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=stat LST0=0x08 LEST0=0x04]
```

・ID=0x30 のヘッダ・レスポンス送信, m コマンドの実行例

```
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x0102030405060708
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
```

・データサイズ変更コマンド, 1 コマンドの実行例

```
LIN default data byte size -> 1
```

データサイズ（送信及び受信）が 1 バイトになります。

・ID=0x30 のヘッダ・レスポンス送信, m コマンドの実行例

```
LIN header/response data send(ch0) : LIN-ID=0x30 data=0x01
[intr(ch0)=send]
LIN header send finished(ch0)
[intr(ch0)=send]
LIN response data send finished(ch0)
```

送信バイト数が 1 バイトに変更されます。

・LIN-ch1（受信動作）

LIN-ch1(J8)は、受信用のポートになりますので、LIN バスに接続して下さい。通信レートは、2.4k-20k オート・ボーレートです。受信データバイト数は、起動後は 8 バイトです。（8 バイトのデータでなければ、受信完了しません。）1-8 コマンドで、受信データバイト数の変更が可能です。

・外部デバイスが ID=0x30 で 8 バイトのデータをヘッダ・レスポンス送信した場合

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x30 9615[bps]
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x0102030405060708 sum=0xEA
```

表示としては、ヘッダを受信したタイミングとレスポンス（データ）を受信したタイミングで表示が 2 行に分かれます。[intr...]の行は割り込み発生時の表示です。lin.h 内で SCI_DEBUG を未定義にした場合には、表示されなくなります。ヘッダ受信の際に、受信した信号のパルス幅から計算されるビットレートを表示します。

・外部デバイスが ID=0x32 でヘッダ送信した場合（レスポンスを返すスレーブが存在するケース）

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x32 9615[bps]
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x32 data=0xFFFEFDFCFBFAF9F8 sum=0xE9
```

このケースでは、ヘッダを送信したデバイスとレスポンスを送信したデバイスは別となりますが、表示としては、ヘッダ・レスポンス送信の場合と変わりません。

・外部デバイスが ID=0x31 でヘッダ送信した場合（レスポンスを返すスレーブが存在しないケース）

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x31 9615[bps]
```

ヘッダの受信情報のみ表示を行います。

・ステータスの確認, S コマンド

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x31 9615[bps]
LIN : status print
  LIN0 : LST0=0x00 LEST0=0x00
  LIN1 : LST1=0x80 LEST1=0x00
  LIN2 : LST2=0x00 LEST2=0x00
```

このケースでエラーを確認すると、LST b7(HTRC)=0b1 ヘッダ受信完了でレスポンスデータ待ちとなっている事が判ります。
（※この状態でヘッダを受信するとエラーとなります）

[参考]LIN-ch1 がヘッダ受信完了でレスポンスデータなしの状態で待機、その後通常の通信データを受け取った場合

```
[intr(ch1)=stat LST1=0x88 LEST1=0x08]
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x30 9615[bps]
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x30 data=0x0102030405060708 sum=0xEA
```

LIN-ch1 はエラー（LEST1=0x08→フレーミングエラー）となり、ステータス割り込み[intr(ch1=stat...]が発生します。ステータス割り込み内でエラーステータスはクリアされるので、その後ヘッダの受信割り込みに飛んできて、続いてレスポンスデータの受信を行います。
（エラー発生時はステータス割り込みに飛んできて、エラーがクリアされて動作継続するイメージです。）

・外部デバイスが ID=0x32 で送信データ 2 バイトでヘッダ・レスポンス送信した場合

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x32 9615[bps]
```

現状、受信バイト数が 8 バイトになっているので、ヘッダは受信しますが、データは受信しません。

・ステータスの確認, S コマンド

```
LIN : status print
  LIN0 : LST0=0x00 LEST0=0x00
  LIN1 : LST1=0xC0 LEST1=0x00
  LIN2 : LST2=0x00 LEST2=0x00
```

このケースでエラーを確認すると、LST b7(HTRC)=0b1, LST b6(D1RC)=0b1 ヘッダ受信完了、かつ 1 バイト目のデータ受信完了となっている事が判ります。(MASTER が 1 バイト目のデータを送信したのか、MASTER とは別な SLAVE が 1 バイト目のデータを送信したのかは判断ができませんが、レスポンスデータを送信したデバイスが存在している事は判ります。)
(※この状態でヘッダを受信するとエラーとなります(エラー発生から、ステータス割り込み、エラークリア、動作継続です))

・データサイズ変更コマンド, 2 コマンドの実行例

```
LIN default data byte size -> 2
```

・外部デバイスが ID=0x32 で送信データ 2 バイトでヘッダ・レスポンス送信した場合

```
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x32 9615[bps]
[intr(ch1)=recv]
LIN data receive(ch1) : LIN-ID=0x32 data=0xFFFE sum=0xCE
```

受信データサイズを変更すれば、8 バイト以外のデータも受信可能です。

※レスポンスデータを受信する際、受信前に RFDL レジスタにレスポンスデータサイズの指定が必要なため、任意のサイズのデータ受信はできません。LIN バスに流れているデータバイト数が不明な場合、1 コマンドで、データバイト数を 1 バイトに設定した場合、データバイト数が 1 の場合受信完了となり、1 より大きい場合は下記のチェックサムエラーとなります。

※受信待ちサイズより送信データバイト数が大きい場合

```
LIN default data byte size -> 1
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x32 9615[bps]
[intr(ch1)=stat LST1=0xC8 LEST1=0x20]
```

ステータス割り込みが発生し、エラー内容としては「チェックサムエラー」となる。

レスポンスデータが存在しているのか不明な場合は、ヘッダ受信後に S コマンドでステータスを確認するか、1 コマンドでデータバイト数を 1 に設定して、

・エラーが出ない場合: レスポンスデータが存在しない
・受信完了: 1 バイトのレスポンスデータあり
・チェックサムエラー: レスポンスデータは存在しているが、受信サイズが合っていない
(→2~8 コマンドでデータサイズを変更してみて、レスポンスデータサイズに合うサイズを探す)

という判断が可能です。

LIN-ch1(J8), LIN-ch2(J9)は、起動後は受信のみ（レスポンスデータの返信は行わない）ですが、SLAVE のレスポンス応答を行わせたい場合は、z, x コマンドでレスポンスデータを返す設定が有効になります。

z コマンド（LIN-ch1(J8)のレスポンス ON/OFF）

```
LIN-ch1 : SLAVE response -> ON,  LIN-ID=0x31
```

x コマンド（LIN-ch2(J9)のレスポンス ON/OFF）

```
LIN-ch2 : SLAVE response -> ON,  LIN-ID=0x32
```

もう一度 x コマンド

```
LIN-ch2 : SLAVE response -> OFF
```

z, x コマンドはトグルで、レスポンスを返す、返さないを切り替えます。

レスポンスデータは、0xFF FE FD FC FB FA F9 F8（8 バイトの場合）です。1-8 コマンドでデータバイト数を切り替えた場合は、先頭側が有効（2 の場合は 0xFF FE）となります。

レスポンスデータを返す様に設定した場合、LIN-ch1 は ID=0x31, LIN-ch2 は ID=0x32 となります。

z コマンドでレスポンスデータ返信を有効にした場合

```
LIN-ch1 : SLAVE response -> ON,  LIN-ID=0x31
[intr(ch1)=recv]
LIN header receive(ch1) : LIN-ID=0x31 9615[bps]
[intr(ch1)=send]
LIN response data send finished(ch1) : LIN-ID=0x31 data=0xFFFEFDFCFBFAF9F8
```

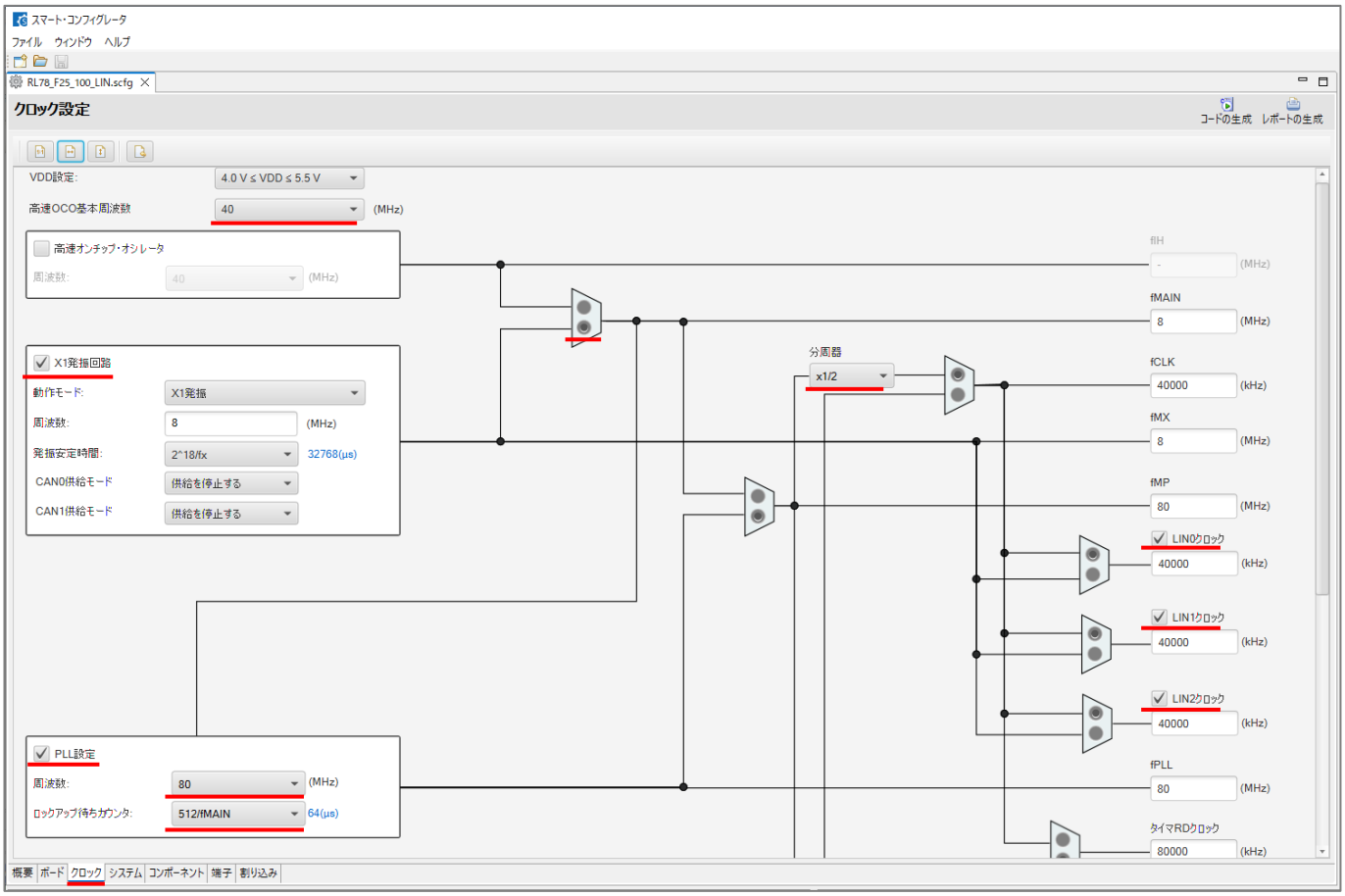
ヘッダが、0x31 の ID となっていた場合は、レスポンスデータを返す動作となります。

6. 開発環境に関して

6.1. スマート・コンフィグレータ

本プロジェクトでは、RL78 スマート・コンフィグレータを使用して、クロックの設定やタイマ、UART などの周辺機器設定を行っています。

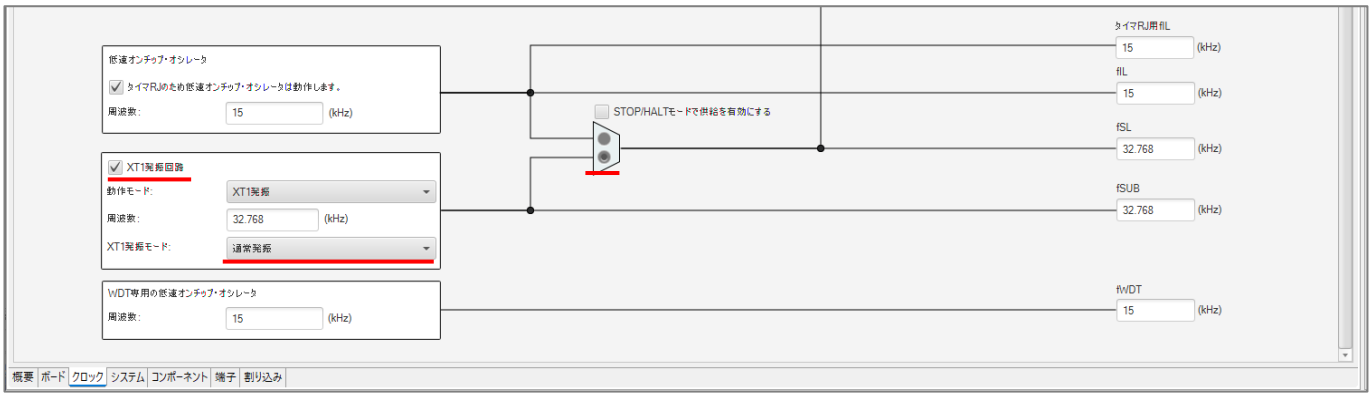
6.1.1. クロック設定



クロックタブを選択。

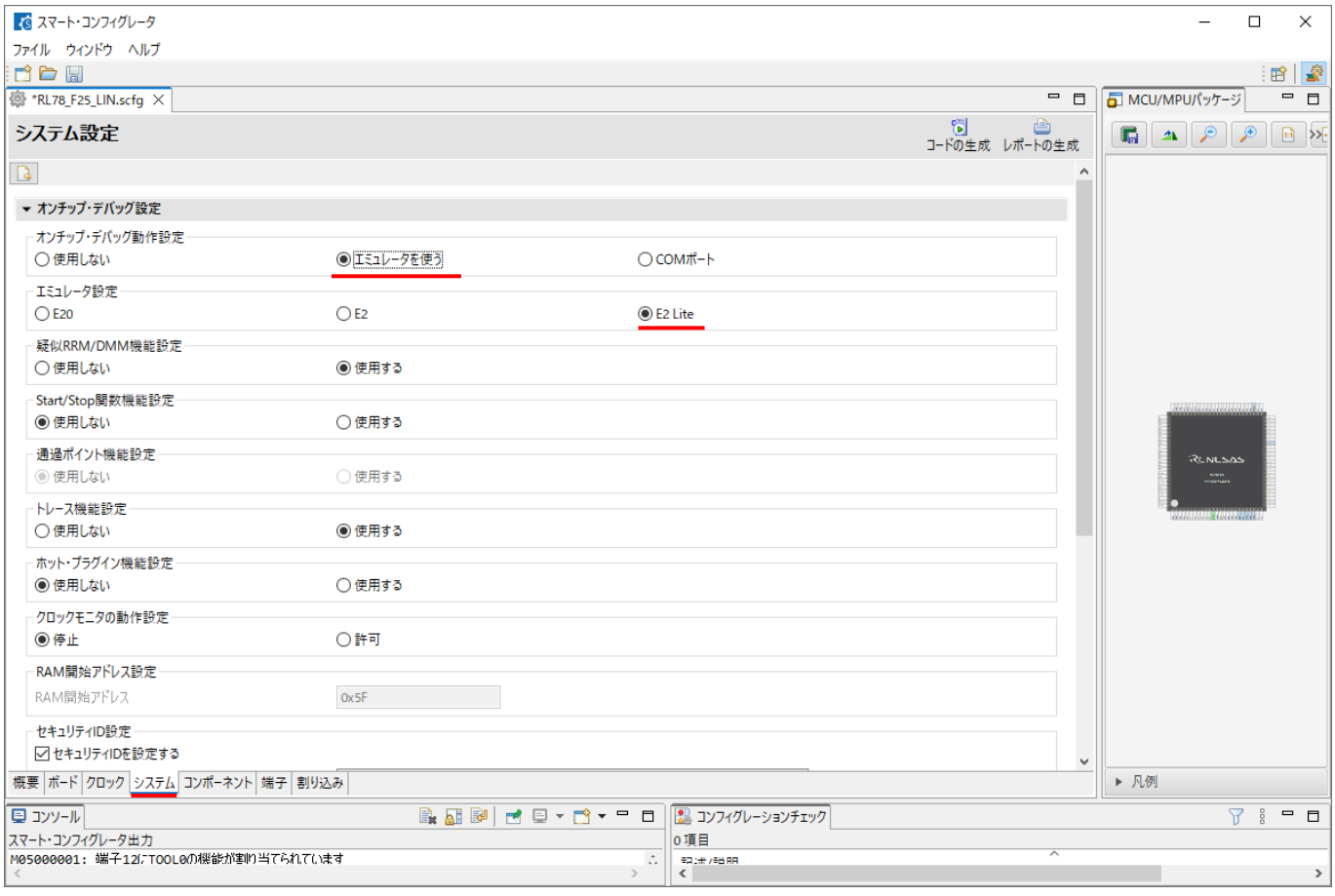
クロックソースは、X1（ボード搭載水晶振動子）とします。LIN は、速度設定があまシビアではない（最大ビットレートが 20kbps 程度）ですので、オンチップオシレータを使う設定でも問題はないかと思います。

本プロジェクトでは、
X1=8MHz を、PLL で 80MHz として、1/2 分周したクロック(40MHz)を CPU クロック(fCLK)としています。
LIN を使用する場合は、（使用する LIN-ch に応じた）LIN0 クロック～LIN2 クロックのチェックを入れてください。



本プロジェクトでは RTC（リアルタイムクロック）も使用していますので、XT1 発振回路を有効（通常発振）として、fSUB にサブクロックの 32.768kHz を与えています。

6.1.2. デバッガの使用



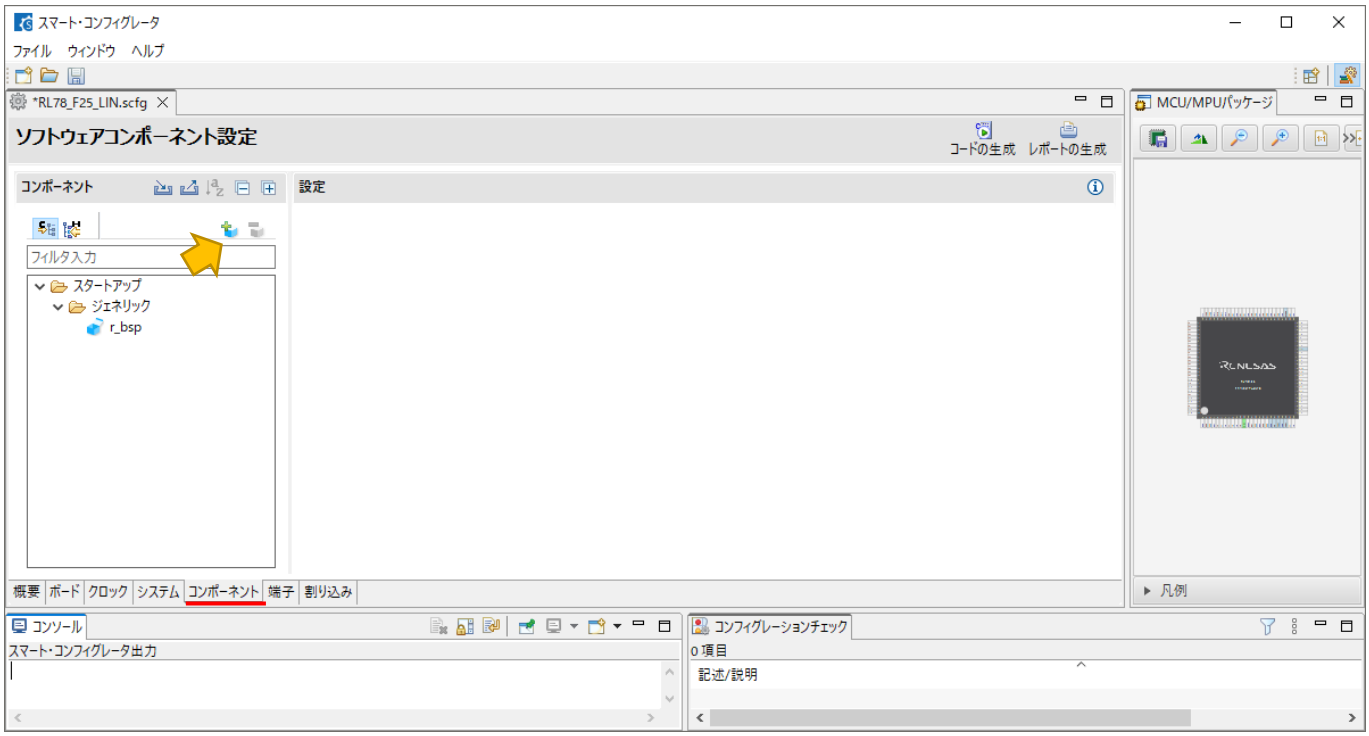
システムタブを選択。

デバッガ使用時は、

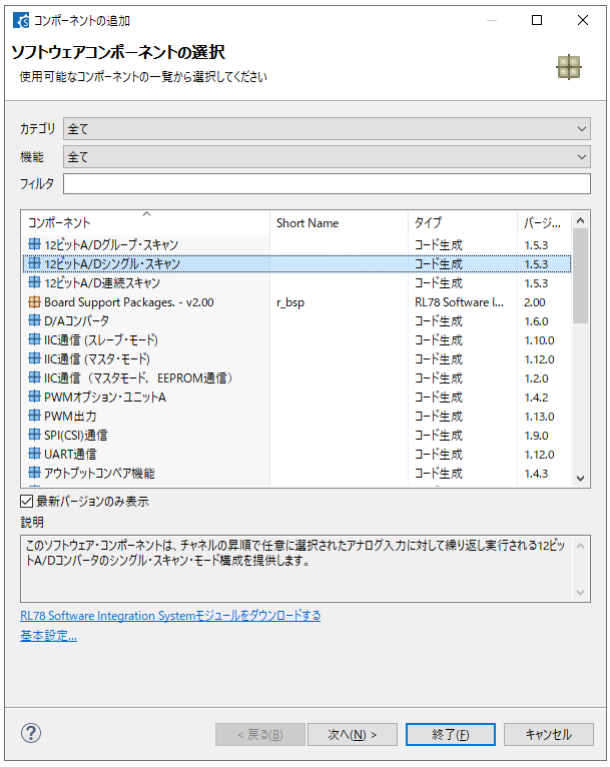
オンチップ・デバッグ動作設定　エミュレータを使う
エミュレータ設定　E2 Lite （お使いのデバッガを選択してください）

とします。（その他の設定は必要に応じて設定）

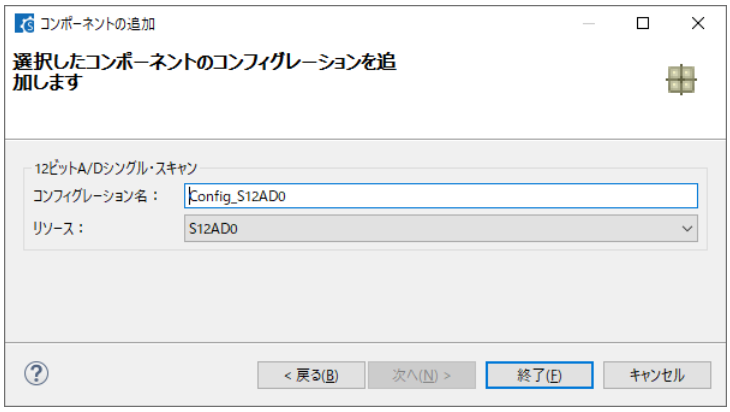
6.1.3. A/D 変換(S12AD0)



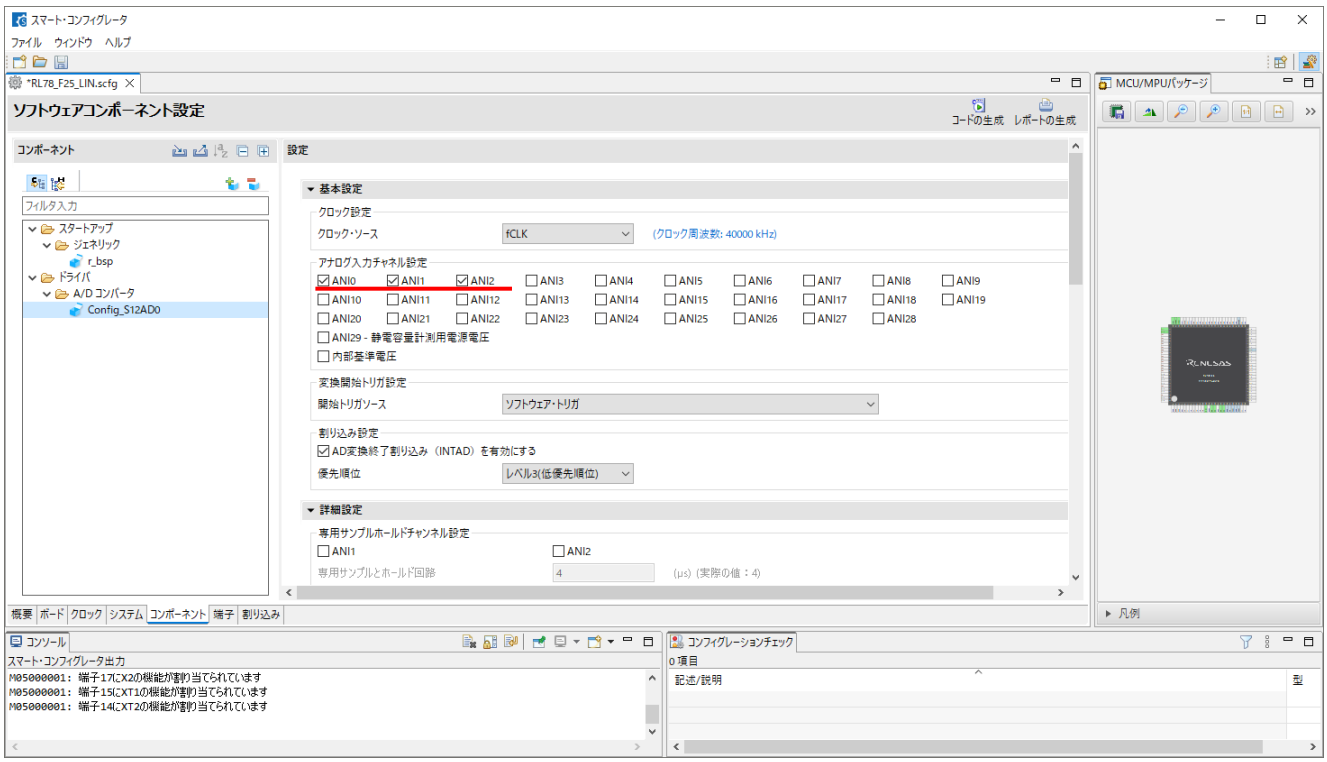
コンポーネントタブからコンポーネントの追加ボタンを押します。



12 ビット A/D シングル・スキャンを選択して次へ。



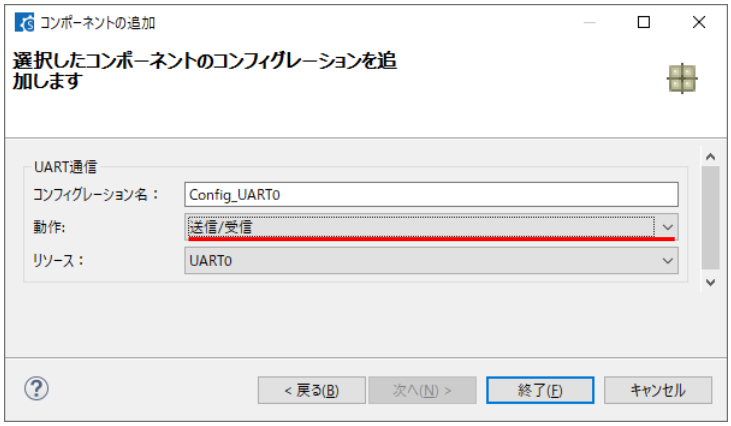
終了を押す。



ANI0, ANI1, ANI2 にチェックを入れる

6.1.4. UART 通信(UART0)

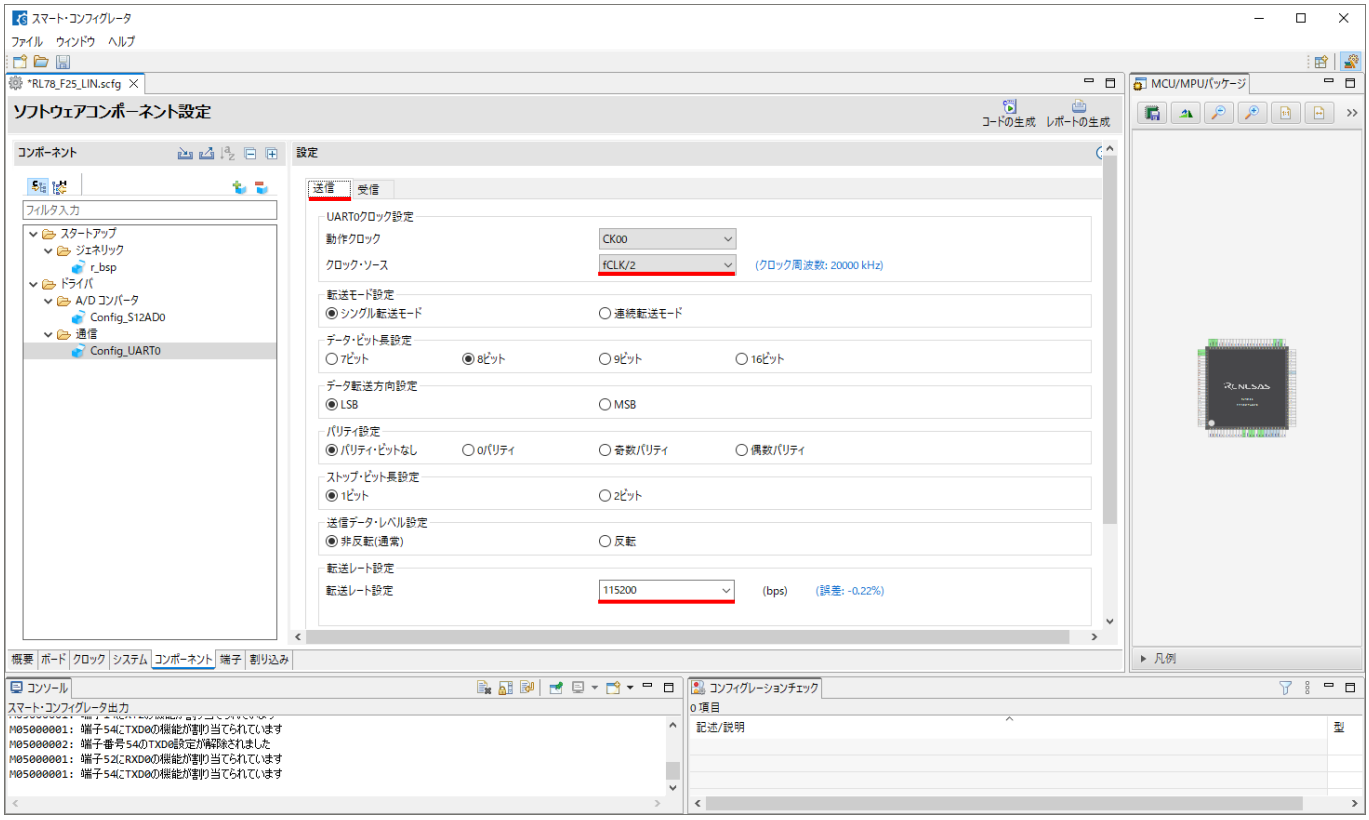
同様に、コンポーネントの追加から、UART 通信を選択して追加。



動作： 送信/受信

リソース： UART0

を選択して終了。



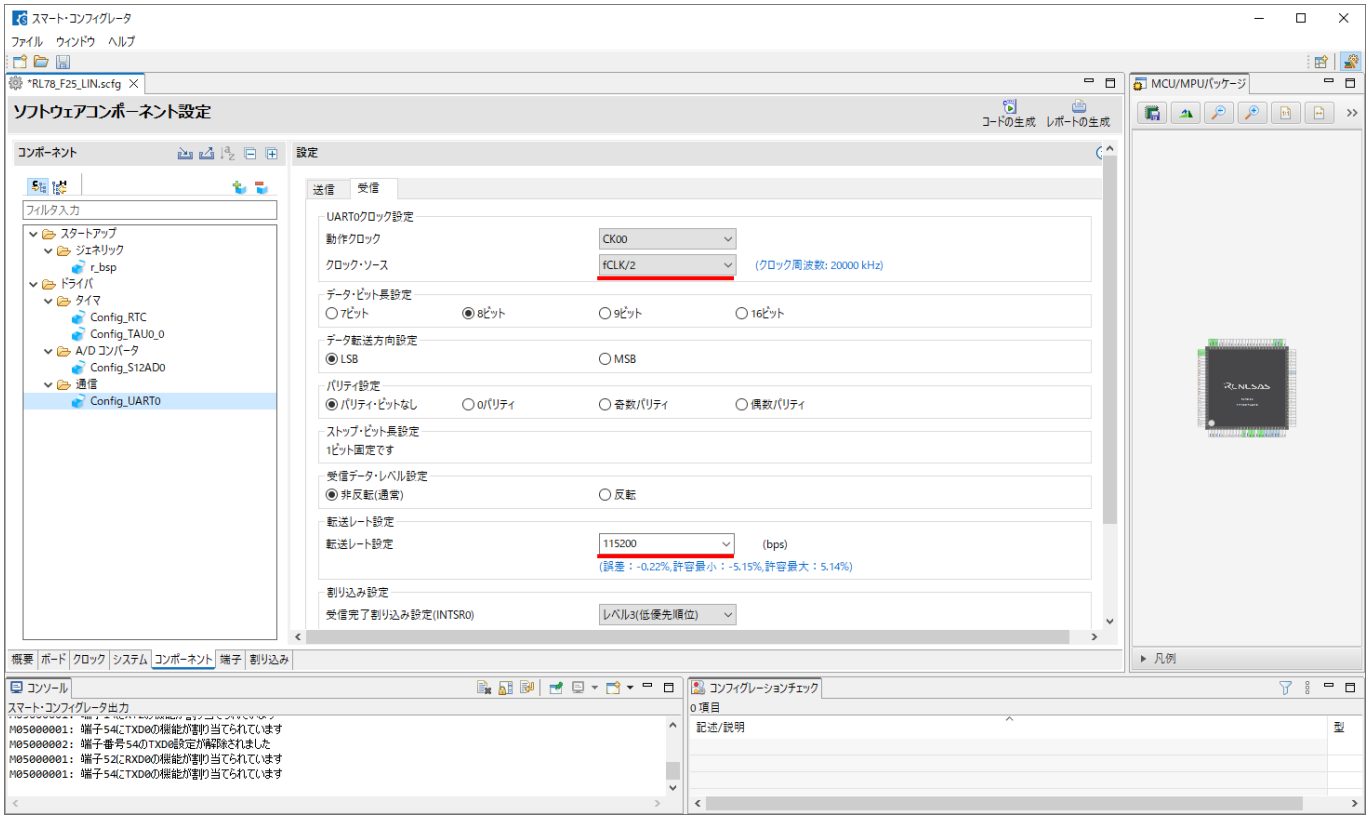
送信タブを選択

クロック・ソース fCLK/2

転送レート設定 115200 （手入力）

とします。転送レートは、端末の速度ですので、115,200bps 以外の速度を設定しても問題ありません。転送レートはプルダウンから選ぶようになっていますが、任意の値を手入力する事もできます。
（転送レートによっては、クロックソースの分周比を変える必要があります。エラーにならない分周比を選択してください。）

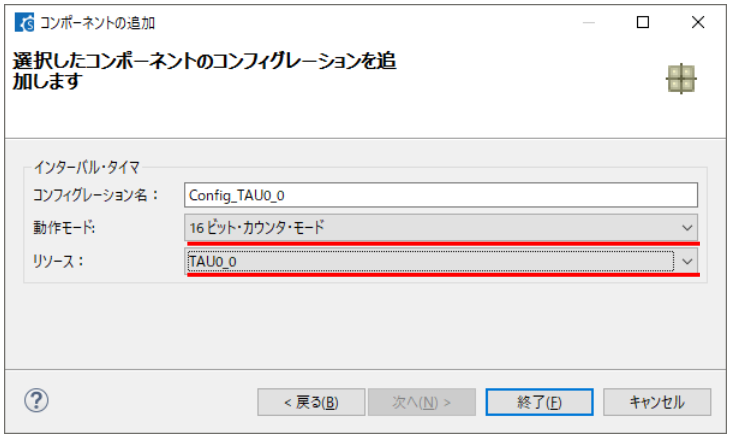
受信タブ側も同様の設定とします。



（RL78 は、UART で送信と受信で別々の転送レート設定が出来てしまいます。PC 上の端末ソフトは、送信と受信で同じ通信速度となりますので、PC と通信させる場合は、必ず送信と受信で同じ転送速度の値を入力してください。）

6.1.5. インターバル・タイマ(TAU0_0)

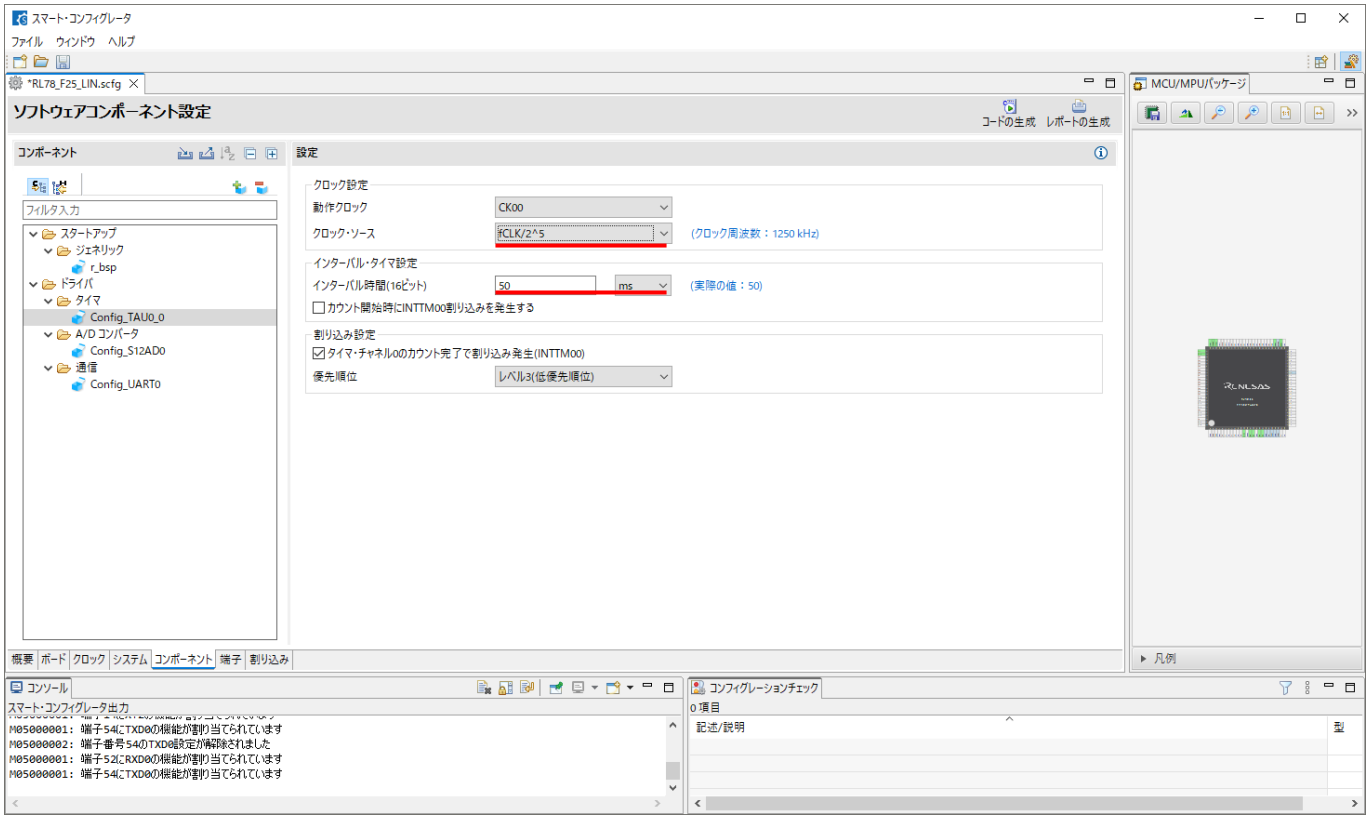
同様に、コンポーネントの追加から、インターバル・タイマを選択して追加。



動作モード　16 ビット・カウンタ・モード

リソース　TAU0_0

を選択して終了。



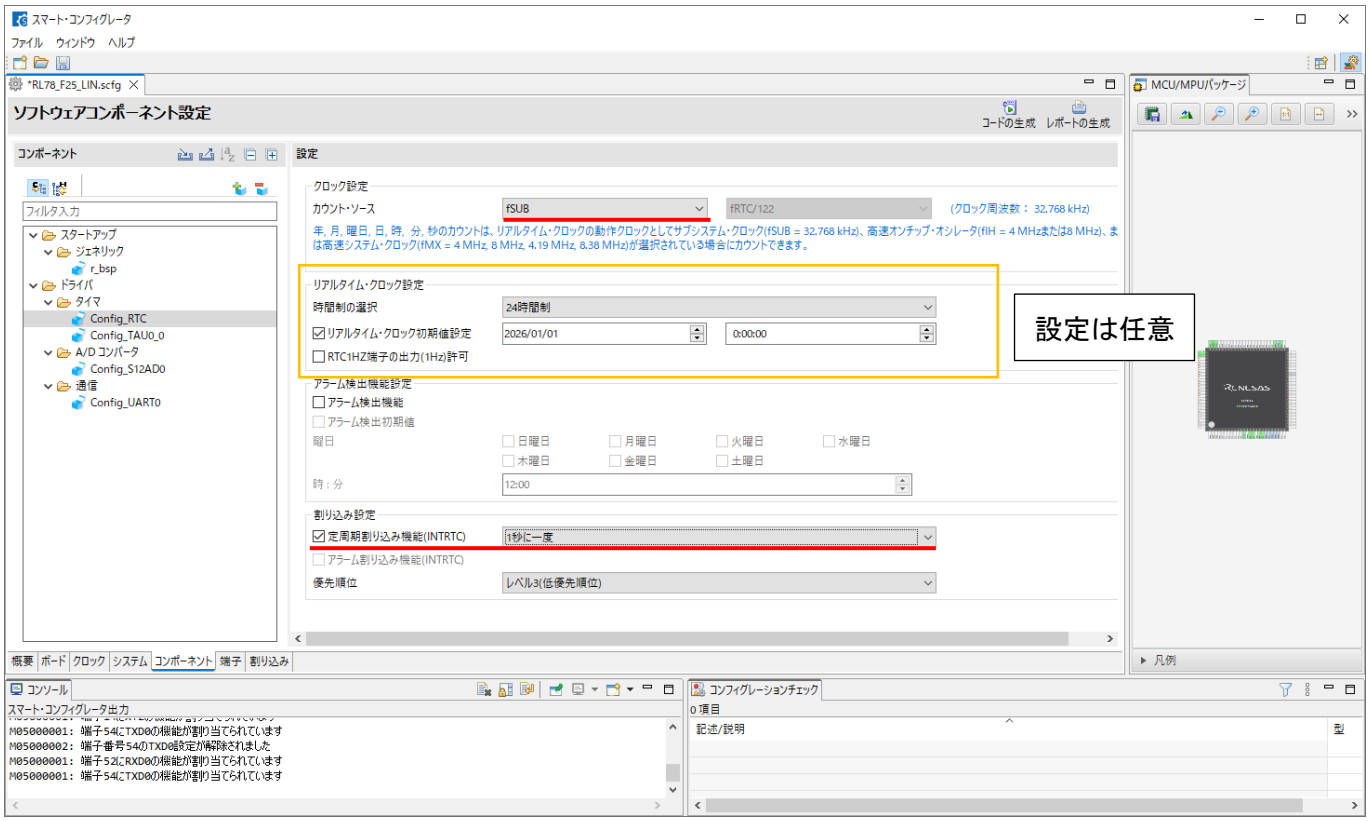
クロック・ソース　fCLK/2^5 （または、fCLK/2^5 よりも大きい分周比）

インターバル時間　50 ms

を選択。

6.1.6. リアルタイム・クロック(RTC)

同様に、コンポーネントの追加から、リアルタイム・クロックを選択して追加。



カウント・ソース　fSUB

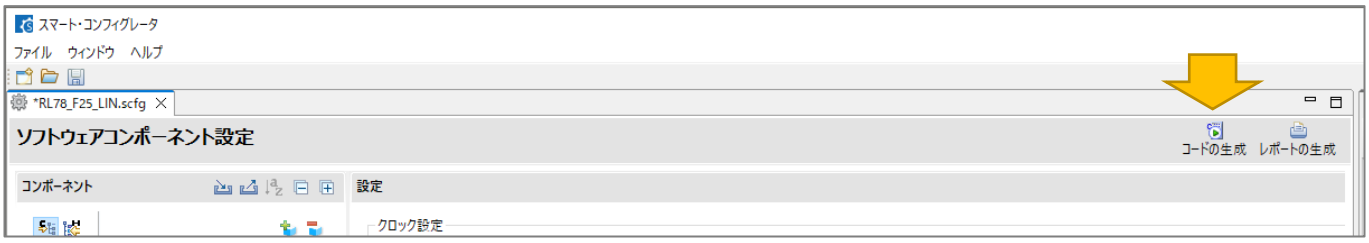
割り込み設定

　定期割り込み機能　チェック

　1 秒に 1 回

（リアルタイム・クロック設定の欄は、本プロジェクトではカレンダ機能は使用していませんので、設定は任意です。）

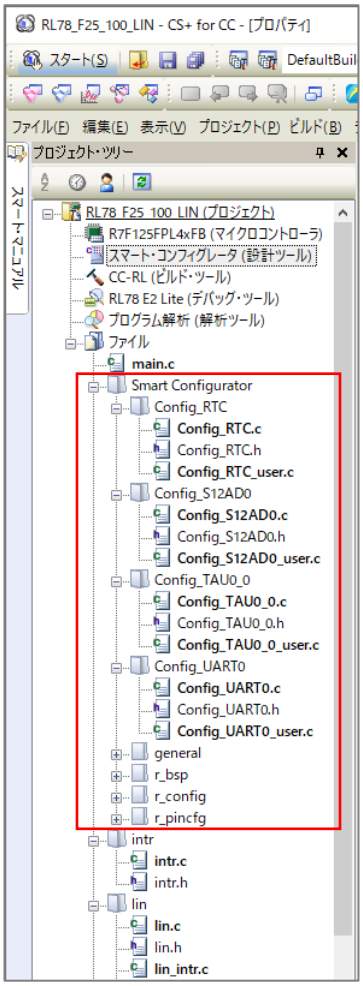
6.1.7. コード生成



一通り設定が終わった後、コード生成ボタンを押してください。

コード生成ボタンにより、GUI で設定した項目がソースコードに反映されます。

6.1.8. 生成コードへのユーザプログラムの追加



プロジェクト・ツリーの SmartConfigurator 以下が、スマート・コンフィグレータで生成したコードです。このファイルの中で、

Config_RTC_user.c

の様に、_user が付くファイルが、基本的にユーザ側で編集するファイルです。

・Config_RTC_user.c

```
/*
* Copyright (c) 2021 - 2026 Renesas Electronics Corporation and/or its affiliates
*
* SPDX-License-Identifier: BSD-3-Clause
*/

/***********************************************************************************************************
******************
* File Name       : Config_RTC_user.c
* Component Version: 1.9.0
* Device(s)       : R7F125FPL4xFB
* Description     : This file implements device driver for Config_RTC.
***********************************************************************************************************
*****************/
/***********************************************************************************************************
******************
Includes
***********************************************************************************************************
*****************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_RTC.h"
/* Start user code for include. Do not edit comment generated here */
#include "intr.h"
/* End user code. Do not edit comment generated here */

（中略）

/***********************************************************************************************************
******************
* Function Name: r_Config_RTC_callback_constperiod
* Description  : This function is real-time clock constant-period interrupt service handler.
* Arguments    : None
* Return Value : None
***********************************************************************************************************
*****************/
static void r_Config_RTC_callback_constperiod(void)
{
    /* Start user code for r_Config_RTC_callback_constperiod. Do not edit comment generated here */
    timer_rtc();
    /* End user code. Do not edit comment generated here */
}
```

追加するのは、赤字の行の 2 箇所です。

#include "intr.h"

timer_rtc();

追加する場合は、

/* Start user code

/* End user code

の間に追加してください。それ以外の場所に追加すると、コード生成の時に消されてしまいます。

・Config_S12AD0_user.c

```
/***********************************************************************************************
******************
Includes
***********************************************************************************************
*****************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
/* Start user code for include. Do not edit comment generated here */
#include "intr.h"
/* End user code. Do not edit comment generated here */

（中略）

/***********************************************************************************************
******************
* Function Name: r_Config_S12AD0_interrupt
* Description  : This function is INTAD interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************************
*****************/
static void __near r_Config_S12AD0_interrupt(void)
{
/* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */
    adc_fin();
/* End user code. Do not edit comment generated here */
}
```

・Config_TAU0_0_user.c

```
/***********************************************************************************************
******************
Includes
***********************************************************************************************
*****************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_0.h"
/* Start user code for include. Do not edit comment generated here */
#include "intr.h"
/* End user code. Do not edit comment generated here */

（中略）

/***********************************************************************************************
******************
* Function Name: r_Config_TAU0_0_interrupt
* Description  : This function is INTTM00 interrupt service routine.
* Arguments    : None
* Return Value : None
***********************************************************************************************
*****************/
static void __near r_Config_TAU0_0_interrupt(void)
{
    /* Start user code for r_Config_TAU0_0_interrupt. Do not edit comment generated here */
    timer_tau00();
    /* End user code. Do not edit comment generated here */
}
```

リアルタイムクロック、A/D 変換、インターバルタイマは、

#include "intr.h"

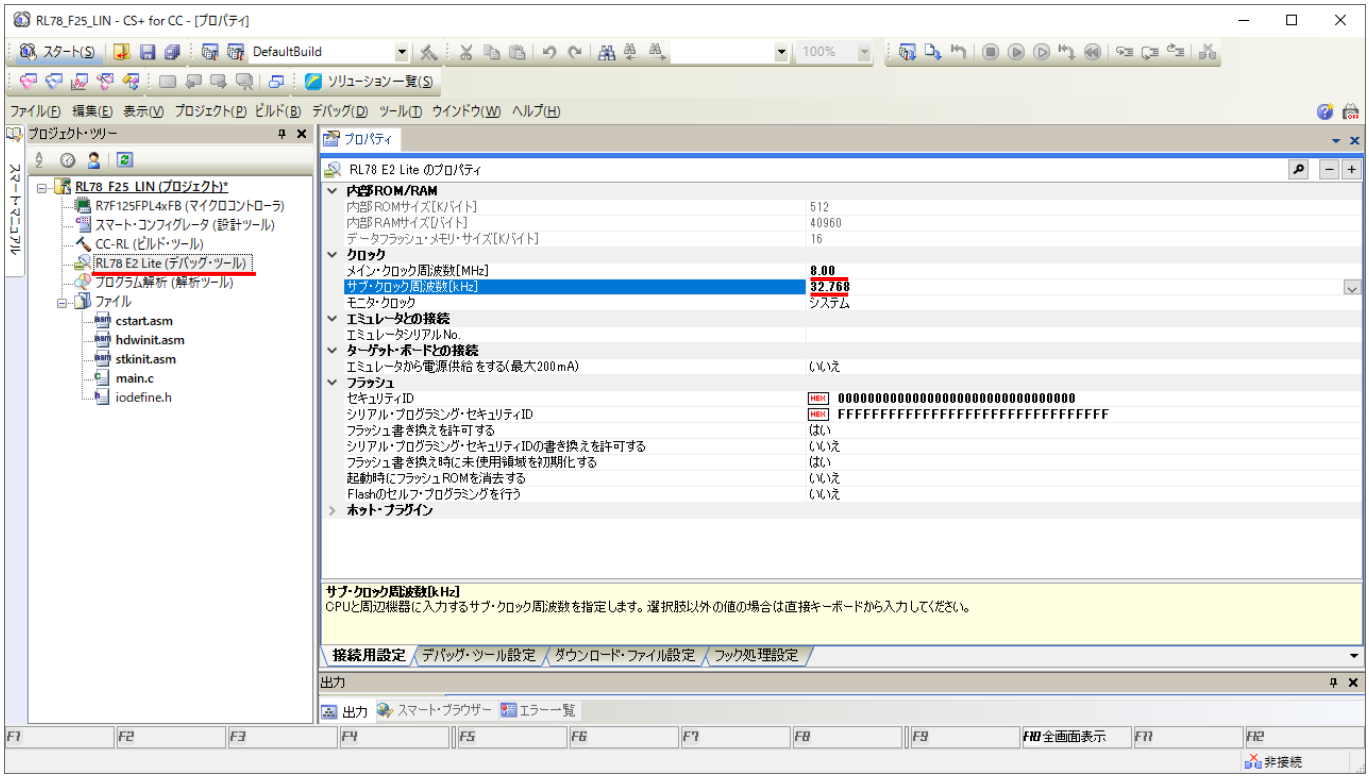
の追加と、intr.c 内に記載されている、割り込み処理の実体関数を追加する形です。

・Config_UART0_user.c

```
/***********************************************************************************************************
******************
Includes
***********************************************************************************************************
*****************/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_UART0.h"
/* Start user code for include. Do not edit comment generated here */
#include "sci.h"
/* End user code. Do not edit comment generated here */

（中略）

/***********************************************************************************************************
******************
* Function Name: r_Config_UART0_callback_sendend
* Description  : This function is a callback function when UART0 finishes transmission.
* Arguments    : None
* Return Value : None
***********************************************************************************************************
*****************/
static void r_Config_UART0_callback_sendend(void)
{
    /* Start user code for r_Config_UART0_callback_sendend. Do not edit comment generated here */
    intr_sci_send_end();
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
******************
* Function Name: r_Config_UART0_callback_receiveend
* Description  : This function is a callback function when UART0 finishes reception.
* Arguments    : None
* Return Value : None
***********************************************************************************************************
*****************/
static void r_Config_UART0_callback_receiveend(void)
{
    /* Start user code for r_Config_UART0_callback_receiveend. Do not edit comment generated here */
    intr_sci_receive_end();
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
******************
* Function Name: r_Config_UART0_callback_error
* Description  : This function is a callback function when UART0 reception error occurs.
* Arguments    : err_type -
*                    error type info
* Return Value : None
***********************************************************************************************************
*****************/
static void r_Config_UART0_callback_error(uint8_t err_type)
{
    /* Start user code for r_Config_UART0_callback_error. Do not edit comment generated here */
    intr_sci_receive_error();
    /* End user code. Do not edit comment generated here */
}
```

UART は、4 行追加する必要があります。

6.2. デバッグ・ツールの設定



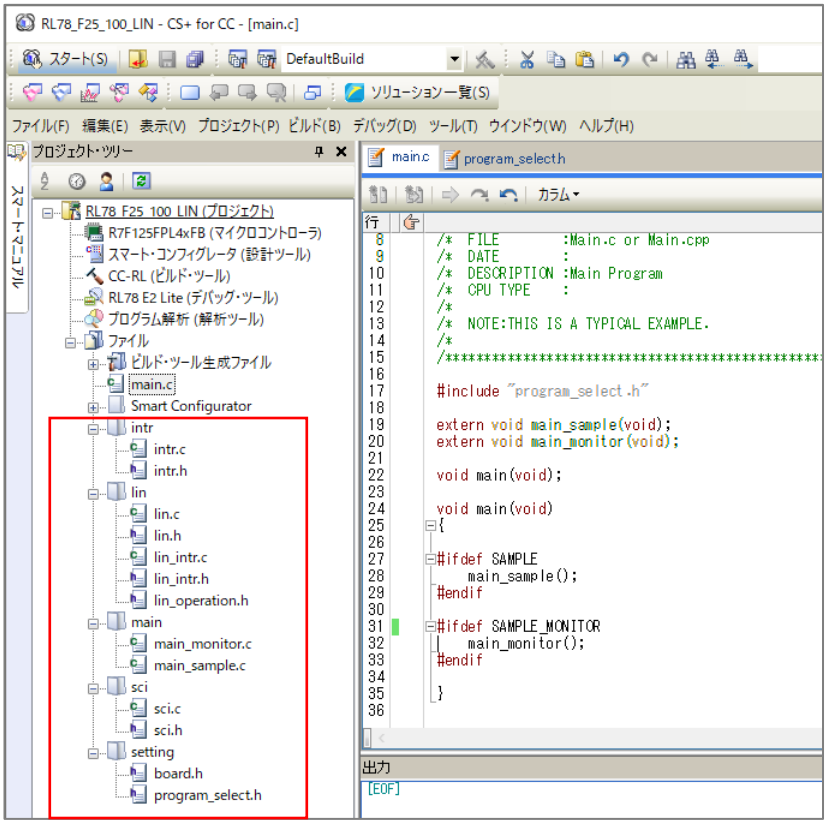
デバッグ・ツールで、E2 か E2Lite（お使いのデバッガに応じて）を選択。

メイン・クロック周波数　8.00 を選択

サブ・クロック周波数　32.768 を選択

してください。

6.3. ユーザ作成コード



赤線内が本プロジェクトで追加してるソースです。

ソースファイルは、

[プロジェクトフォルダ]¥usr_src

以下に格納されています。

（プロジェクトを新規に作成する場合は、プロジェクトツリーの「ファイル」以下に、ユーザ作成ソースがぶら下がる様に追加してください。）

「通常のサンプルプログラム」と「モニタプログラム（5.3 節）」は、main 以下で

main_monitor.c　モニタプログラム

main_sample.c　通常のサンプルプログラム

と分かれています。

これらは、program_select.h で切り替えています。

・program_select.h

```
/*---------------------------------------------------------------------
定数定義
---------------------------------------------------------------------*/
//プログラム選択 [どちらか1つのみ選択]


//通常のLINのサンプルプログラム
#define SAMPLE

//MONITOR
//LINパケット送受信のモニタ用プログラム
//#define SAMPLE_MONITOR

//ーーここまで プログラム選択
```

#define SAMPLE //通常のサンプルプログラム

#define SAMPLE_MONITOR //モニタプログラム

どちらか一方の定義を有効にして、プログラムの「リビルド」を行って下さい。

（この定義を変更した場合、通常のビルドではなく「リビルド」を行ってください。）

6.4. ミラー領域に関して（far と near）

RL78 では、アドレス上に「ミラー領域」という短い命令でアクセス可能な領域が設定されています。

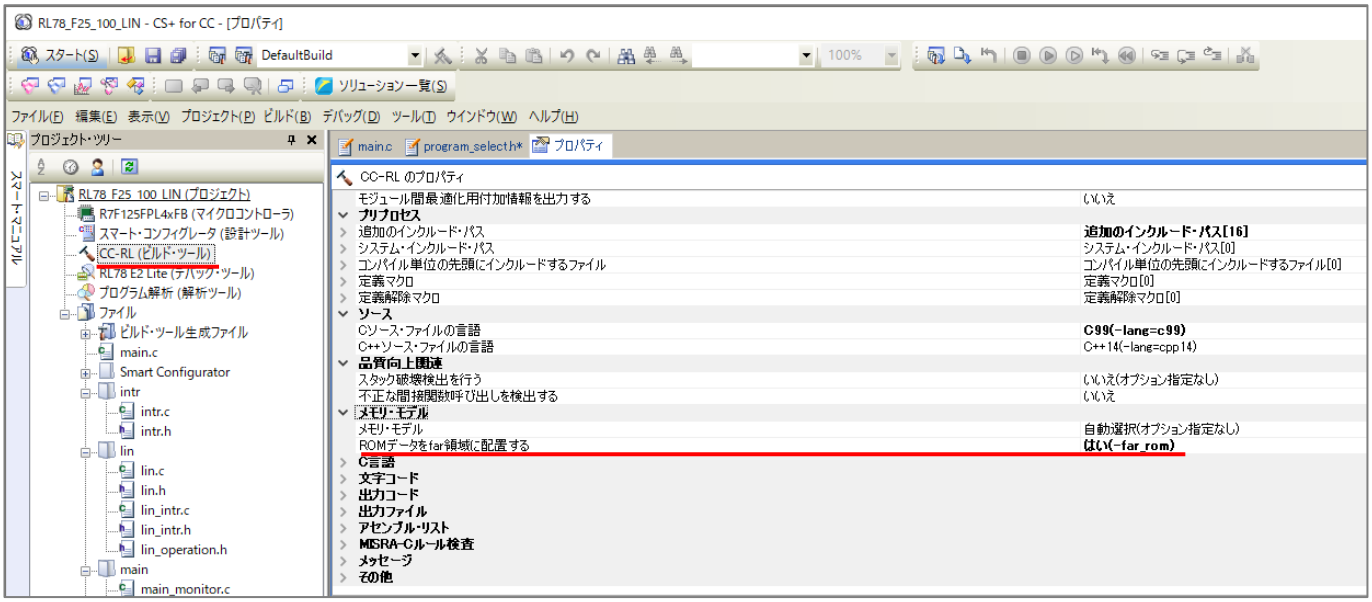
RL78/F25 では
F5000H ~ F5EFFH
で、サイズは 3.75kB です。RL78/F25 は ROM が 512kB と RL78 の中では大きなサイズの ROM を搭載しているのですが、ミラー領域は他の RL78 マイコンに対してサイズが小さいものとなっています。

一般的に、初期値ありの変数や文字列定数
int a = 3;
sci_write_str("LIN Starter kit RL78/F25 Monitor sample program.¥n¥n");
が、ミラー領域に割り当てられ、ミラー領域に割り当てられた変数は短い命令で高速にアクセス可能となります。

但し、RL78/F25 では、ミラー領域のサイズが小さいため、初期値ありの変数が多いとミラー領域に割り当てられる変数がオーバフローします（リンク時にエラーとなります）。

この場合、高速にアクセスできる利点を捨てて、通常の ROM 領域（ここでは far 領域）に変数を配置する事で、オーバフローを回避する事ができます。

・データを far 領域に配置する設定



CC-RL（ビルド・ツール）「コンパイル・オプション」タブ
メモリ・モデル
ROM データを far 領域に配置する　はい

（初期値なしの変数は RAM に割り当てられます。初期値ありの変数は、ROM と RAM の両方に割り当てられます。最初は ROM から読み出して、RAM にコピーされ、その後は RAM 上の変数がアクセスされます。この、初期値ありの変数が割り当てられる ROM 領域として、通常はミラー領域から優先的に割り当てされます。）

（単純に言えば、ミラー領域を使った場合は near で CPU が高速にメモリアクセスできる領域、far はアクセス速度が低下する領域です。）

（配置データを工夫して、一部を near、残りを far に配置するという事も可能ですが、オーバフローした場合上記設定で、一括で far 領域を使用する様に設定するのが楽です。）

※なお、本キットの CAN のサンプルプログラムは 3.75kB のミラー領域にデータが入りませんでした。それに対し、LIN のサンプルプログラムは

・RL78_F25_100_LIN.map

```
.constf
                              000000f0  000008ee       7ff   2
```

使用領域が、0x7FF（=2047 バイト）となっており、上記設定を行わなくてもミラー領域に定数(const)が収まります。

現状の LIN のサンプルプログラムでは、far 領域を使わなくても問題はないのですが、ユーザがコードを追加した結果溢れてしまう事も考えられますので、最初から far 領域を使う設定としています。

※map ファイルで見た際に、.const が near 領域の定数、.constf が far 領域の定数となります。

（「ROM データを far 領域に配置する　はい」を指定した場合、.const に変わって.constf が使われます。）

RL78/F25 でリンク時に.const のオーバフローとなった場合は、ROM データを far 領域に配置する設定を行ってみてください。

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）, lin_intr.c 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	2026.8.3	—	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/F25(QFP-100 ピン, QFP-80 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/F25
LIN ソフトウェア編　マニュアル

株式会社 北斗電子
 2026 年 8 月 3 日改訂 REV.1.0.0.0 (260803)