LIN・CAN スタータキット RL78/F15
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.1.1.0.0

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、サンプルプログラムの解説を行う資料となります。

キット付属のマイコンボード(HSBRL78F15-100)は、ボード上に LIN トランシーバを 2ch、CAN トランシーバを 2ch 搭載しており、LIN, CAN をそれぞれ 2ch 使用する事ができるボードとなっています。

本書で説明するサンプルプログラムは、

・LIN（SLAVE レスポンス）
・LIN（MASTER レスポンス）
・CAN

の通信を行うものです。

プログラムは、通信を行うコードがシンプルで判り易くなるよう構成しています。エラー等の処理は含まれていませんので、実際のアプリケーションを構築する際は、必要な処理を追加してください。

サンプルプログラム CD

製品に付属しているサンプルプログラム CD の内容を下記に示します。

フォルダ			内容
SOURCE¥	RL78_F15_LIN¥		LIN（SLAVE からのデータ送信） サンプルプログラム(*1)
	RL78_F15_LIN2¥		LIN（MASTER からのデータ送信） サンプルプログラム(*1)
	RL78_F15_CAN¥		CAN（CAN0→CAN1 データ送信） サンプルプログラム(*1)
	RL78_F15_CAN_S1¥		CAN 送信バッファ、受信バッファ 使用データフレーム送受信 サンプルプログラム(*2)
	RL78_F15_CAN_S2¥		CAN 送受信 FIFO、受信 FIFO 使用データフレーム送受信 サンプルプログラム(*2)
	RL78_F15_CAN_S3¥		CAN 送受信 FIFO、受信 FIFO 使用データフレーム/リモートフレーム送受信サンプルプログラム(*2)
	64PIN		HSBRL78F13-64 HSBRL78F14-64 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(*2)
DOCUMENT¥	HSBRL78F15-100_REV_X_s.pdf		マイコンボード取扱説明書
	LIN_CAN_KIT_RL78F15_software_REV_X_s.pdf		サンプルプログラム解説書(1)
	LIN_CAN_KIT_RL78F15_software_ RS-CAN_Lite_REV_X_s.pdf		サンプルプログラム解説書(2)
	LIN_CAN_KIT_RL78F15_REV_X_s.pdf		本資料

サンプルプログラムは、ルネサスエレクトロニクス CS+(CS+forCC)向けのプロジェクトで作成されています。
CS+ for CC Ver8.08 以降を予めインストール願います。

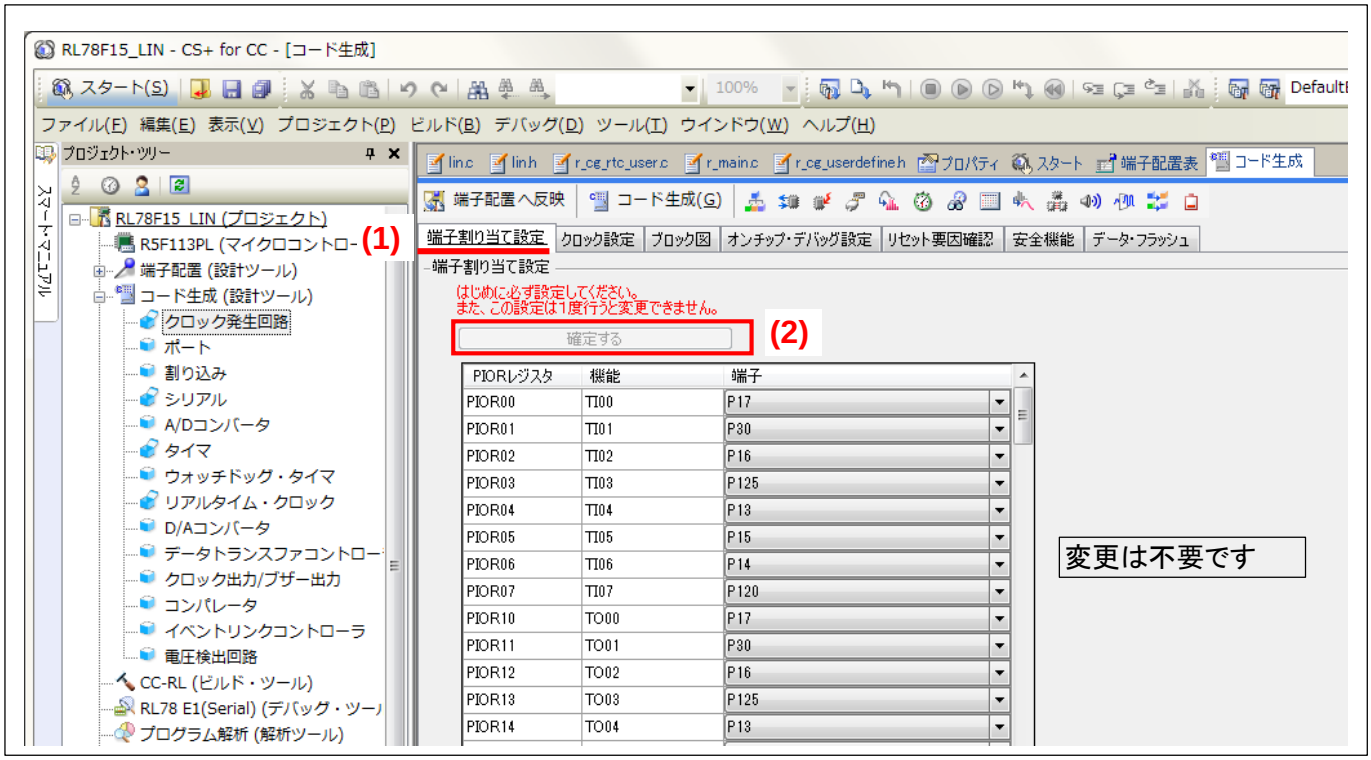
(*1)のサンプルプログラムは、サンプルプログラム解説書(1)
(*2)のサンプルプログラムは、サンプルプログラム解説書(2)
で説明されています。

1. サンプルプログラム共通部分

サンプルプログラムは、CS+のコード生成機能を活用し、基本的な部分のプログラムコードを生成しています。
新規に、お客様側でプロジェクトを作成する際は、本章を参考にして頂きたく。CD に含まれるサンプルプログラムを動かす上では、本章の説明は読み飛ばして頂いて構いません。
CD に含まれるサンプルプログラムに関しては、2 章以降で説明しています。

1.1. 端子設定

RL78/F15 で、CS+のコード生成機能を使用する際に、端子の割り当て（PIOR レジスタ）の設定を最初に行う必要がありますが、この設定は初期値を使用しています。



(1)端子割り当て設定タブ
(2)「確定する」
を押して設定を確定させる

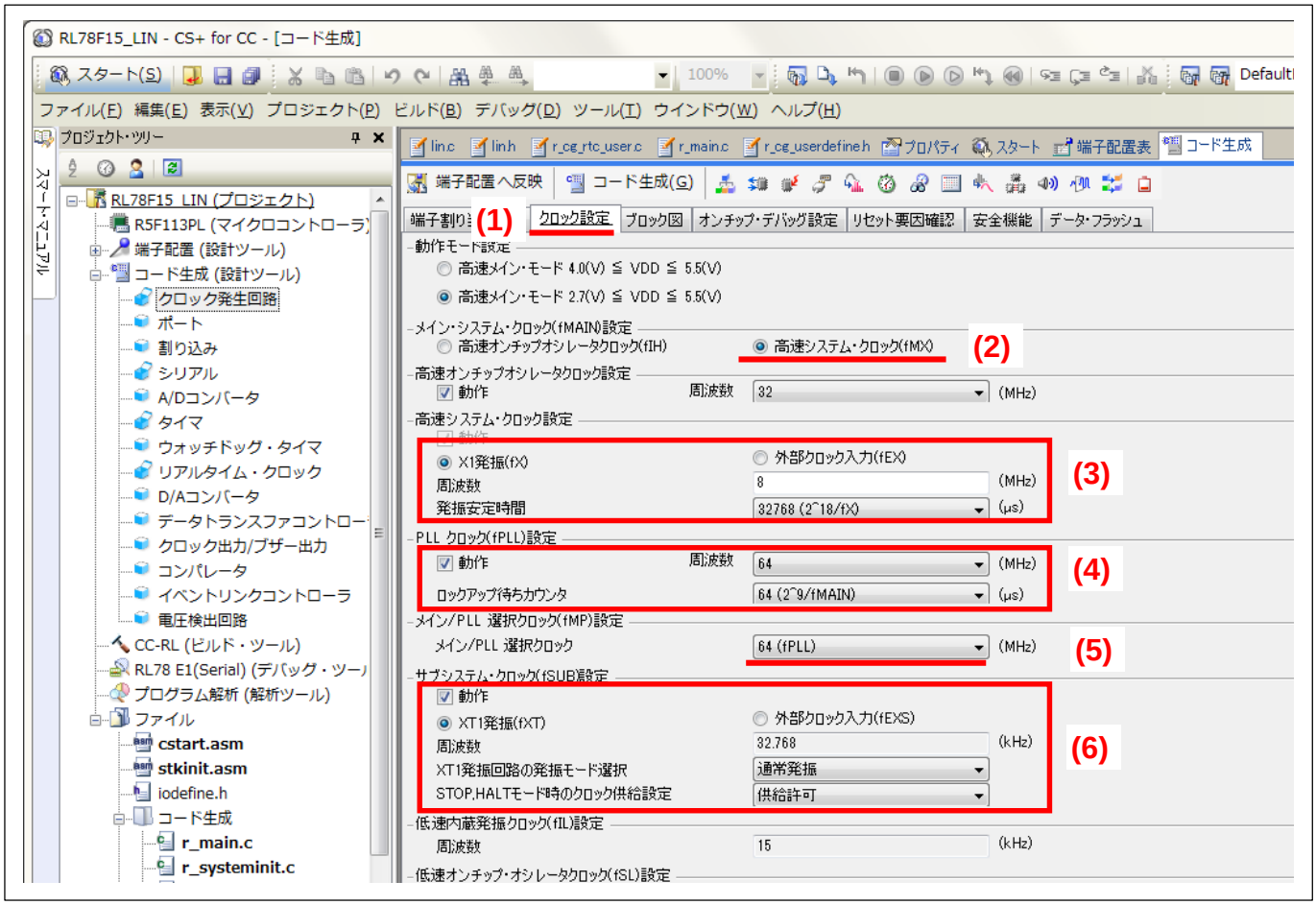
－本サンプルプログラムで使用している端子－

PIOR レジスタ	機能	端子	備考
PIOR40	**RXD0**/SI00/SDA00	P16	SCI0 の端子設定（プログラムの動作の表示に使用）
PIOR40	**TXD0**/SO00	P15	SCI0 の端子設定（プログラムの動作の表示に使用）
PIOR44	LTxD0	P13	LIN ch0 の端子設定
PIOR44	LRxD0	P14	LIN ch0 の端子設定
PIOR46	CTxD0	P10	CAN ch0 の端子設定
PIOR46	CRxD0	P11	CAN ch0 の端子設定

新規にプロジェクトを作成する場合は、上記の端子は別な端子に設定しないでください。その他の端子は割り当てを変更頂いても問題ありません。

1.2. クロック設定

LIN MASTER や、CAN のプログラムでは通信速度（ビットレート）を正確に決めるため、水晶振動子ベースのクロックを使用しています。キット付属ボードには、8MHz の水晶振動子が搭載されており、この 8MHz を源泉とし、マイコン内蔵の PLL 回路で、32MHz を生成し、ボードのクロックとしています。



設定項目で、設定が必要なものを示します。

(1)クロック設定タブ

(2)メイン・システムクロック(fMAIN)設定　「高速システム・クロック(fMX)」 を選択

(3)高速システム・クロック設定

「X1 発振」 を選択

周波数　「8」 を入力

(4)PLL クロック(fPLL)設定

動作　「チェック」

周波数　「64」 MHz を選択

(5)メイン/PLL 選択クロック(fMP)設定　「64(fPLL)」 を選択

(6)サブシステム・クロック(fSUB)設定

動作　「チェック」

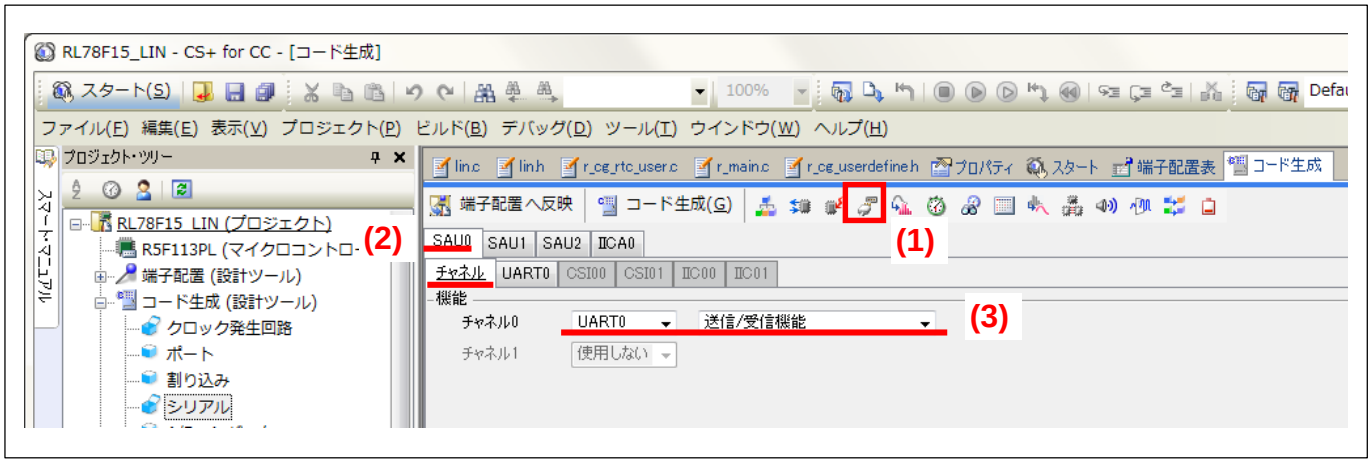
「XT 発振(fXT)」 選択

周波数　「32.768」 kHz を設定

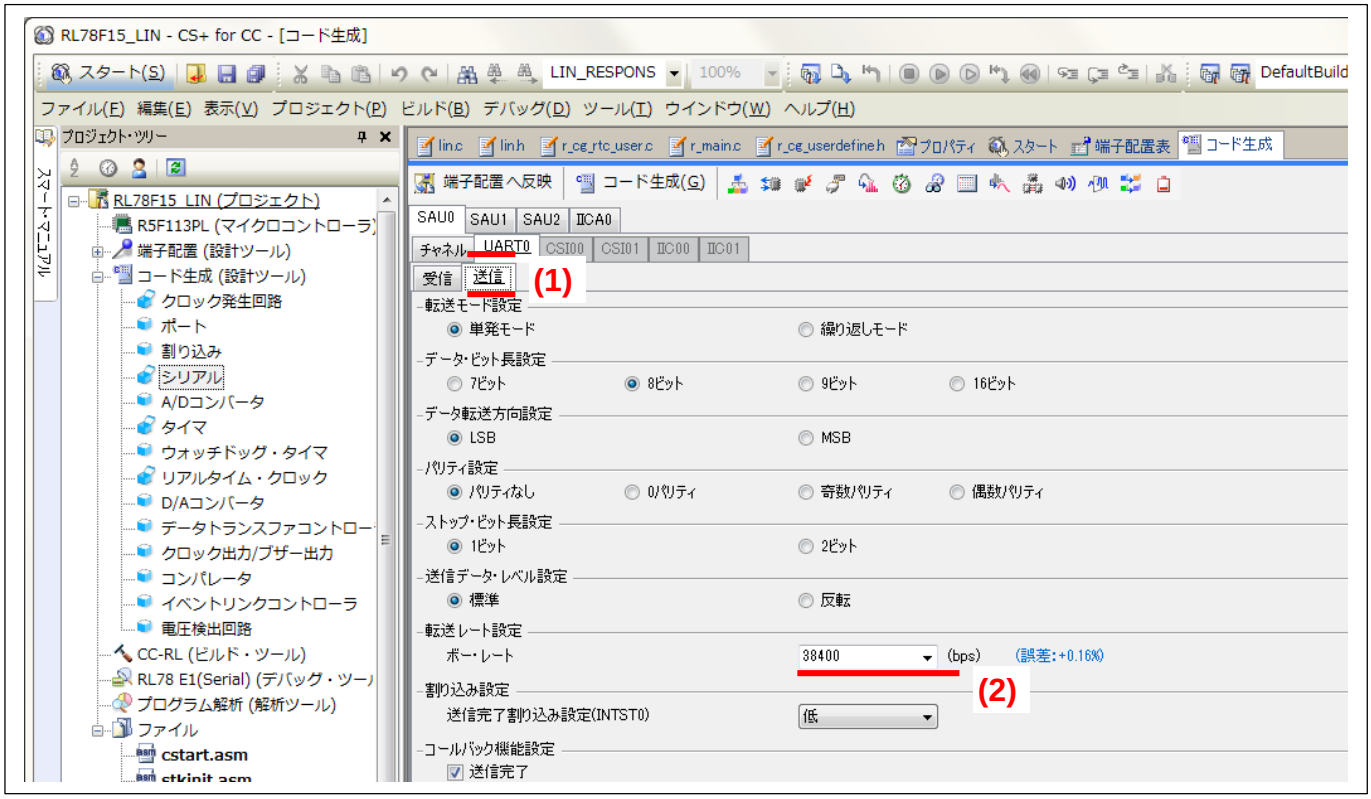
XT1 発振回路の発振モード選択　「通常発振」 を選択

1.3. シリアル設定

プログラムの動作のモニタに、UART(SCI0)を使用していますので、シリアルの設定を行っています。



設定項目は
(1)シリアル設定ボタン
(2)SAU0 タブ　チャンネルタブ
(3)チャネル 0　「UART0」「送信/受信機能」 を選択

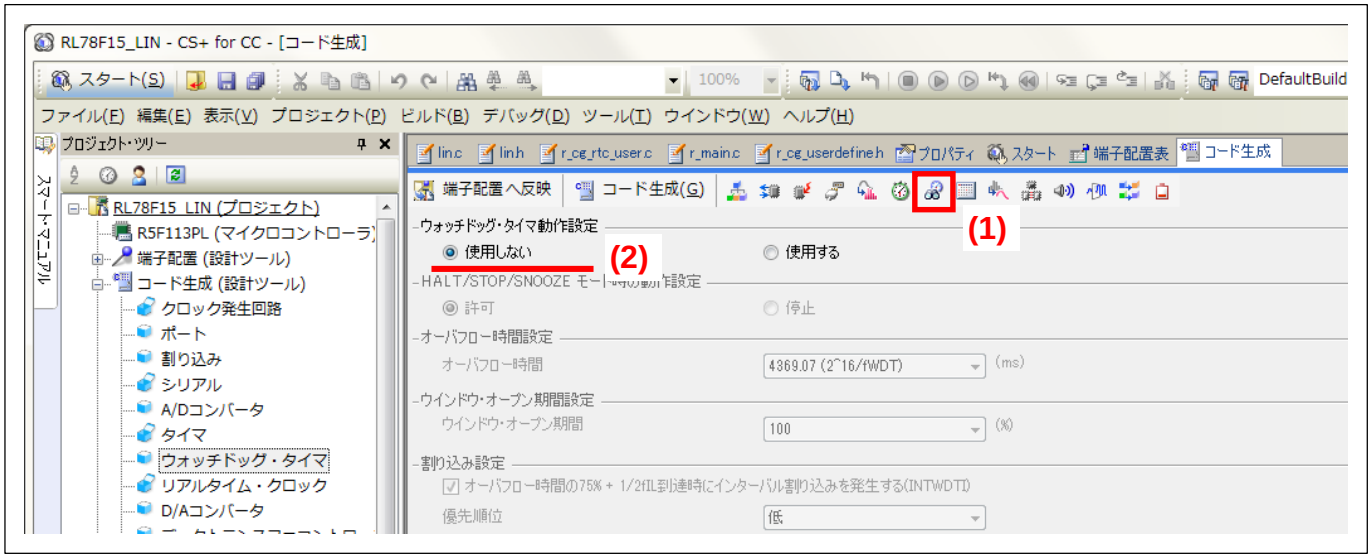


設定項目は
(1)UART0 タブ　送信タブ
(2)ボーレート　「38400」を選択

※(2)以外はデフォルト値で問題ありません
※サンプルプログラムでは、「受信」機能は使用していませんが、ボーレートは「送信」側と同じ値に設定してください

1.4. ウォッチドッグ設定

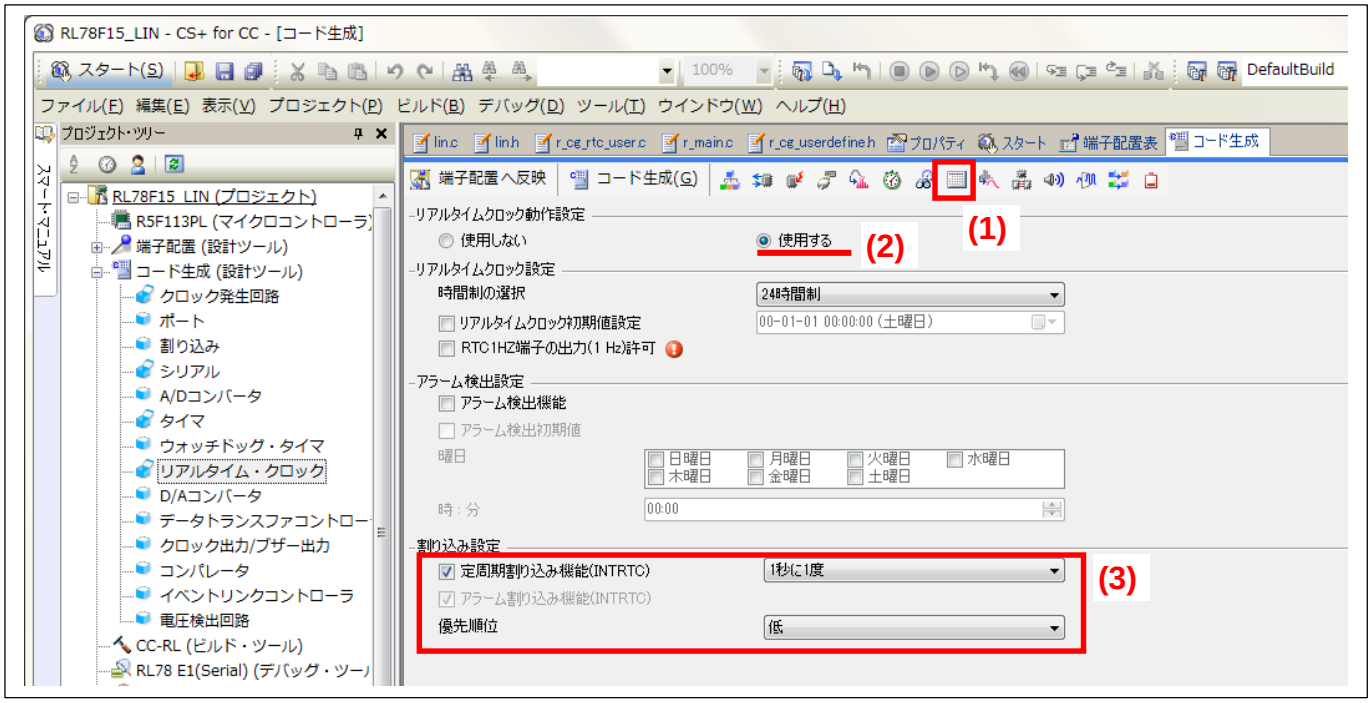
サンプルプログラムでは、ウォッチドッグタイマは未使用なので、使用しない設定を行います。



設定項目は
(1)ウォッチドッグ・タイマボタン
(2)ウォッチドッグ・タイマボタン　「使用しない」 を選択

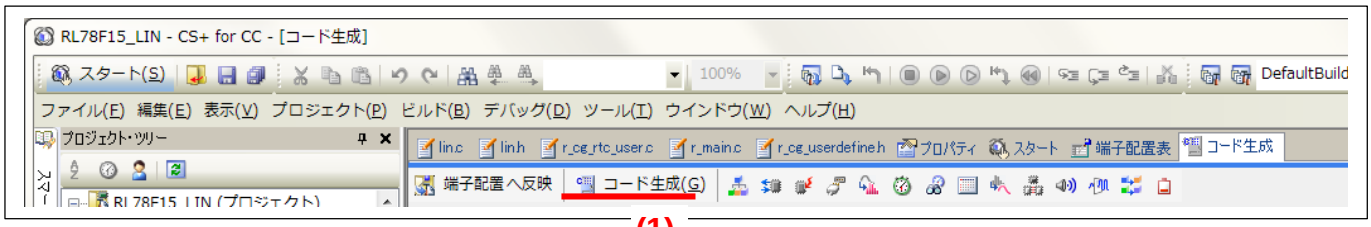
1.5. リアルタイムクロック設定

サンプルプログラムでは、リアルタイムクロックでデータ送出を制御しているので、1 秒に 1 回定期割り込みが入るよう設定ています。



設定項目は
(1)リアルタイムクロックボタン
(2)リアルタイムクロック動作設定　「使用する」 を選択
(3)割り込み設定
　　定期割り込み機能(INTRTC)　「チェック」　「1 秒に 1 度」 を選択

1.6. コード生成



(1)

各種設定後、最後に「コード生成」のボタンを押してください。

GUI で設定した各種設定が、プログラムコードに反映されます。

－共通で行う設定－

端子設定	CTXD0 等の端子が、ボードに合うよう設定	
クロック設定	ボード搭載の メインクロック(8MHz)を使用 サブクロック(32.768kHz)を使用	通信レートの源泉として、8MHz の水晶振動子を使用します
UART	SCI0 を使用(38,400bps)	速度は変更可
ウォッチドッグ	使用しない	
リアルタイムクロック	1 秒毎に定期割り込みが入るよう設定	

上記、各サンプルプログラムで共通で使用している部分は、CS+のコード生成機能により生成できます。

2. LIN サンプルプログラム

2.1. プログラム仕様

・LIN の MASTER からヘッダ送信
・SLAVE からレスポンス送信
・MASTER からレスポンス送信

を行うサンプルプログラムとする。

2.2. 関数仕様

2.2.1. ユーザ関数

lin0_init　　（LIN ch0 向け）
lin2_init　　（LIN ch2 向け）

概要：初期化関数
宣言：
　　int lin0_init(unsigned char mode, unsigned char id)
　　int lin2_init(unsigned char mode, unsigned char id)
説明：
　・端子設定
　・割り込み設定
　・レジスタの設定
　・フラグクリア
　・LIN 動作モード遷移
を行います
引数：
　　mode: モード(MASTER, SLAVE)を指定
　　　　LIN_MASTER(0)　MASTER モードとして初期化
　　　　LIN_SLAVE(1)　　SLAVE モードとして初期化
　　id: LIN-ID を指定
戻り値：
　　0: 正常終了
　　-1: エラー

MASTER の場合は、
　ボーレート　9,600bps
　ブレーク幅　13bit
で設定を行います。

SLAVE の場合は、
　オートボーレート（MASTER 側から送信されるヘッダの SYNC フィールドにボーレートを同期させる）
で設定します。

lin0_data_set　　（LIN ch0 向け）
lin2_data_set　　（LIN ch2 向け）

概要：データ設定関数
宣言：
　　int lin0_data_set(unsigned char direction, unsigned char id, unsigned char *data, unsigned char length)
　　int lin2_data_set(unsigned char direction, unsigned char id, unsigned char *data, unsigned char length)
説明：
　・送信データの設定
を行います
引数：
　　direction: 送信・受信を設定
　　　　LIN_SEND(1)　　送信を指定
　　　　LIN_RECEIVE(2)　　受信を指定
　　id: LIN-ID を指定
　　*data: 送信データを設定
　　length: データ長(0~8)を設定
戻り値：
　　0: 正常終了
　　-1: エラー

・LIN の ch が MASTER に初期化されており、direction に「送信」を指定した場合
　id　ヘッダに含まれる ID を指定
　*data　送信データを指定
　length　送信バイト数を指定

・LIN の ch が MASTER に初期化されており、direction に「受信」を指定した場合
　id　ヘッダに含まれる ID を指定（レスポンスを要求する SLAVE の ID を指定）
　*data　未使用
　length　受信バイト数を指定

・LIN の ch が SLAVE に初期化されており、direction に「送信」を指定した場合
　id　未使用
　*data　送信データを指定
　length　送信バイト数を指定

・LIN の ch が SLAVE に初期化されており、direction に「受信」を指定した場合
　id　受信 ID を指定（ここで指定した ID がヘッダに含まれていた場合データを受信する）
　*data　未使用
　length　受信バイト数を指定

lin0_header_start　　（LIN ch0 向け）
lin2_header_start　　（LIN ch2 向け）

概要：ヘッダ送信、受信設定関数
宣言：
　　int lin0_header_start(void)
　　int lin0_header_start(void)
説明：
　・MASTER:ヘッダを送信する
　・SLAVE:ヘッダ受信設定を行う
を行います
引数：　なし
戻り値：
　　0: 正常終了

LIN の ch が MASTER に設定されている場合、本関数を呼び出したタイミングでヘッダを送出します。
SLAVE に設定されている場合、ヘッダの受信設定（受信待機）を行います。ヘッダを受信したタイミングで割り込みが掛かりますので、ヘッダ受信後の処理は割り込みルーチンで実行します。

lin0_response_read　　（LIN ch0 向け）
lin2_response_read　　（LIN ch2 向け）

概要：レスポンス読み出し関数
宣言：
　　int lin0_response_read(unsigned char *id, unsigned char *data, unsigned char *sum)
　　int lin2_response_read(unsigned char *id, unsigned char *data, unsigned char *sum)
説明：
　・レスポンスデータの読み出し
を行います
引数：
　*id: ヘッダに含まれる ID が格納されます
　*data: 受信したデータが格納されます（8 バイト以上の領域を確保してください）
　*sum: 受信したチェックサムデータが格納されます
戻り値：
　　0: 正常終了

LIN のデータレジスタから読み出しを行います。データの受信が完了した事を示すフラグの確認後に、本関数を呼び出してください。

lin0_status_clear　　（LIN ch0 向け）
lin2_status_clear　　（LIN ch2 向け）

概要:ステータスクリア関数
宣言:
　　int lin0_status_clear(void)
　　int lin2_status_clear(void)
説明:
　・ステータスレジスタのクリア
　・エラーフラグレジスタのクリア
を行います
引数:　なし
戻り値:
　　0:　正常終了

LIN のステータス、エラーのレジスタ値のクリアを行います。

2.2.2. 内部関数

ユーザ関数から内部的に呼び出される関数です。

lin_id

概要:PID 変換関数
宣言:
　　unsigned char lin_id(unsigned char id)
説明:
　・指定した ID を PID（プロテクト ID，パリティ情報を含んだ実際にヘッダに埋め込まれる ID データ）に変換
を行います
引数:
　　id: ID 値（0x00 ~ 0x3F が有効です）
戻り値:
　　PID 値

lin_rev_id

概要:PID 逆変換関数
宣言:
　　unsigned char lin_rev_id(unsigned char encoded_id)
説明:
　・PID(プロテクト ID，パリティ情報を含んだ実際にヘッダに埋め込まれる ID データ)を ID に変換します
を行います
引数:
　　encoded_id: PID 値
戻り値:
　　ID 値

2.2.3. 割り込み関数

lin0_interrupt_send (vect=INTLIN0TRM)　　(LIN ch0 向け)
lin2_interrupt_send (vect=INTLIN2TRM)　　(LIN ch2 向け)

概要:送信割り込み関数
宣言:
　　static void __near lin0_interrupt_send(void)
　　static void __near lin2_interrupt_send(void)
説明:
　・送信完了時
に呼び出されます
引数:　なし
戻り値:　なし

lin0_interrupt_receive (vect=INTLIN0RVC)　　(LIN ch0 向け)
lin2_interrupt_receive (vect=INTLIN2RVC)　　(LIN ch2 向け)

概要:送信割り込み関数
宣言:
　　static void __near lin0_interrupt_receive(void)
　　static void __near lin2_interrupt_receive(void)
説明:
　・レスポンス受信完了時
に呼び出されます
引数:　なし
戻り値:　なし

lin0_interrupt_status (vect=INTLIN0STA)　　（LIN ch0 向け）
lin2_interrupt_status (vect=INTLIN2STA)　　（LIN ch2 向け）

概要:ステータス割り込み関数
宣言:
　　static void __near lin0_interrupt_status (void)
　　static void __near lin2_interrupt_status(void)
説明:
　・エラー発生時
に呼び出されます
引数:　なし
戻り値:　なし

2.3. プログラムで使用している変数・定数

2.3.1. 定数定義(lin.h)

・モード設定
```
#define  LIN_MASTER  0
#define LIN_SLAVE  1
```

・動作制御
```
#define LIN_NO_OPERATION  0
#define LIN_RESPONSE_SET  1
```

・動作ステータス
```
#define LIN_0_IDLE  0
#define LIN_1_HEADER_PROCESS  1
#define LIN_2_RESPONSE_PROCESS  2
#define LIN_3_DATA_READ_PROCESS  3
```

・方向
```
#define LIN_SEND  1
#define LIN_RECEIVE  2
```

上記定義値は、条件分岐等で数値の代わりに使用される名称の定義です。

```
#define  LIN0_EN_USE
```

LIN0 の EN を P53 で制御する場合に定義

#define LIN0_WAKEUP_USE

LIN0 の LOCAL WAKEUP を P52 で制御する場合に定義（サンプルでは未定義）

#define LIN2_EN_USE

LIN2 の EN を P156 で制御する場合に定義

#define LIN2_WAKEUP_USE

LIN2 の LOCAL WAKEUP を P157 で制御する場合に定義（サンプルでは未定義）

#define LIN_FLAME_LENGTH 4

データのフレーム長を定義（0~8)、サンプルプログラムではデータ長を 4 バイトとしている

2.3.2. 定数定義(r_cg_userdefine.h)

#define LIN_CH0_ID0x30
#define LIN_CH2_ID0x32

LIN の ID を定義する

#define INTERRUPT_DEBUG

割り込みのデバッグ定義
本定数を定義している場合、割り込みのタイミングで、P90~P95 のポートを反転させる

2.3.3. グローバル変数

unsigned char g_lin0_mode （LIN ch0 向け）
unsigned char g_lin2_mode （LIN ch2 向け）

現在の LIN ch の動作モード(MASTER, SLAVE)を格納
LIN_MASTER(0)
LIN_SLAVE(1)
のいずれか。

unsigned char g_lin0_id　　（LIN ch0 向け）
unsigned char g_lin2_id　　（LIN ch2 向け）

ch の ID を格納。SLAVE 設定時、かつ送信設定時、この ID がヘッダに含まれていると、レスポンスを返す。

unsigned char g_lin0_accept_id　　（LIN ch0 向け）
unsigned char g_lin2_accept_id　　（LIN ch2 向け）

受信対象とする ID を設定。SLAVE 設定時、かつ受信設定時、この ID がヘッダに含まれていると、データを受信する。

unsigned char g_lin0_operation　　（LIN ch0 向け）
unsigned char g_lin2_operation　　（LIN ch2 向け）

レスポンス動作を保持
LIN_NO_OPERATION(0)
LIN_RESPONSE_SET(1)
のいずれか。
LIN_RESPONSE_SET(1)が設定されているときは、
MASTER 送信　ヘッダ送信完了割り込みでレスポンス送信
SLAVE 送信　ヘッダ受信割り込みでレスポンス送信
動作を行う。

unsigned char g_lin0_status_flag　　（LIN ch0 向け）
unsigned char g_lin2_status_flag　　（LIN ch2 向け）

現在のステータスを保持。
LIN_0_IDLE(0)　　アイドル状態
LIN_1_HEADER_PROCESS(1)　　ヘッダ処理中
LIN_2_RESPONSE_PROCESS(2)　　レスポンス処理中
LIN_3_DATA_READ_PROCESS(3)　データ受信完了
のいずれか。このフラグ変数で、データ受信完了をモニタする。

2.4. 動作説明

実際に LIN の通信を行う際、2.2 記載の関数をどのような流れで呼び出すかを以下で説明します。

2.4.1. SLAVE レスポンス送信

(1)初期化を行う

lin0_init(LIN_MASTER, LIN_CH0_ID);　//LIN0 MASTER として初期化
lin2_init(LIN_SLAVE, LIN_CH2_ID);　　//LIN2 SLAVE として初期化
(LIN_CH0_ID : 0x30, LIN_CH2_ID : 0x32)

(2)SLAVE(ch2)側で送信するデータをレジスタに設定する

lin2_data_set(LIN_SEND, dummy, lin_buf2, LIN_FLAME_LENGTH);

lin_buf2[0]~[3]に送信するデータを代入し、lin2_data_set 関数で SLAVE 側から送信する準備をする。

(3)SLAVE(ch2)側でヘッダ受信設定をする

lin2_header_start();

(4)MASTER(ch0)側でレスポンス受信設定を行う

lin0_data_set(LIN_RECEIVE, id, dummy_a, LIN_FLAME_LENGTH);

id は受信側の（レスポンスを返して欲しい）ID を指定。

(5)MASTER(ch0)側からヘッダを送信する

lin0_header_start();

(i1)ch0　送信割り込み　(lin0_interrupt_send)

フラグ変数の変更

(i2)ch2　受信割り込み　(lin2_interrupt_receive)

ヘッダの ID を確認、自局のものであればレスポンス送信

(i3)ch2 送信割り込み (lin0_interrupt_send)

フラグ変数の変更

(i4)ch0 受信割り込み (lin0_interrupt_receive)

フラグ変数の変更
g_lin0_status_flag に LIN_3_DATA_READ_PROCESS(3) をセット

(6)MASTER(ch0)側でレスポンス受信したデータを読み出す

```
switch(g_lin0_status_flag)
{
   case LIN_3_DATA_READ_PROCESS:
      lin0_response_read(&lin_id0, lin_buf0, &lin_sum0);
      break;
}
```

g_lin0_status_flag を確認し、データ受信が完了している場合、読み出しデータは有効となるので、データ読み出し関数でデータを取得する。lin_buf0, lin_sum0 に、受信データとチェックサムが格納される。(lin_id0 には、ID が格納されるが、自分自身で送信したものであるので、あまり意味はありません)

※(1)~(6)はユーザがプログラムで設定するものです
(i1)~(i4)は、割り込み関数で処理されるものです

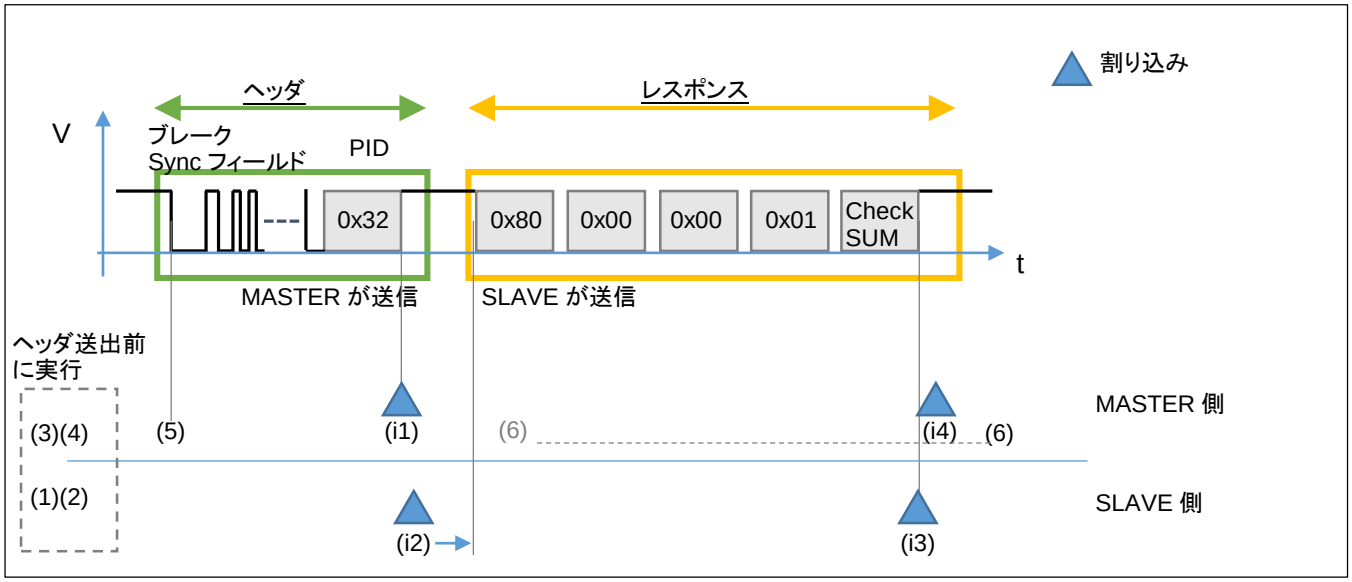


図 2-1 LIN SLAVE レスポンスでのデータパケット

各処理を時系列で示すと、図 2-1 の様になります。
(5)を呼び出した際、MASTER 側からヘッダの送信が行われます。
(i2)[SLAVE 側受信割り込み]をトリガに、SLAVE 側はレスポンス送信を行います。
(6)の処理はループ等で定期的に行い、(i4)でフラグがセットされた後、有効データが読み出せます。

2.4.2. MASTER レスポンス送信

(1)初期化を行う

lin0_init(LIN_MASTER, LIN_CH0_ID);　//LIN0 MASTER として初期化
lin2_init(LIN_SLAVE, LIN_CH2_ID);　　//LIN2 SLAVE として初期化
(LIN_CH0_ID : 0x30, LIN_CH2_ID : 0x32)

(2)SLAVE(ch2)側で受信設定する

lin2_data_set(LIN_RECEIVE, LIN_CH0_ID, dummy_a, LIN_FLAME_LENGTH);

id に受信する ID を設定する。（ここでは、ch0 の ID, 0x30 を指定）

(3)SLAVE(ch2)側でヘッダ受信設定をする

lin2_header_start();

(4)MASTER(ch0)側で送信するデータをレジスタ設定する

lin0_data_set(LIN_SEND, id, data, LIN_FLAME_LENGTH);

data[0]~[3]に送信するデータを代入し、id は送信側の ID を指定。（ここでは 0x30）

(5)MASTER(ch0)側からヘッダを送信する

lin0_header_start();

(i1)ch0　送信割り込み　(lin0_interrupt_send)

レスポンス送信セット
フラグ変数の変更

(i2)ch2 受信割り込み (lin2_interrupt_receive)

ヘッダの ID を確認、受信対象のものであれば受信を行う

(i3)ch0 送信割り込み (lin0_interrupt_send)

フラグ変数の変更

(i3)ch2 受信割り込み (lin2_interrupt_receive)

フラグ変数の変更
g_lin2_status_flag に LIN_3_DATA_READ_PROCESS(3) をセット

(6)SLAVE(ch2)側でレスポンス受信したデータを読み出す

```
switch(g_lin2_status_flag)
{
   case LIN_3_DATA_READ_PROCESS:
      lin2_response_read(&lin_id2, lin_buf2, &lin_sum2);
      break;
}
```

g_lin2_status_flag を確認し、データ受信が完了している場合、読み出しデータは有効となるので、データ読み出し関数でデータを取得する。lin_id2, lin_buf0, lin_sum0 に、ID と受信データとチェックサムが格納される。

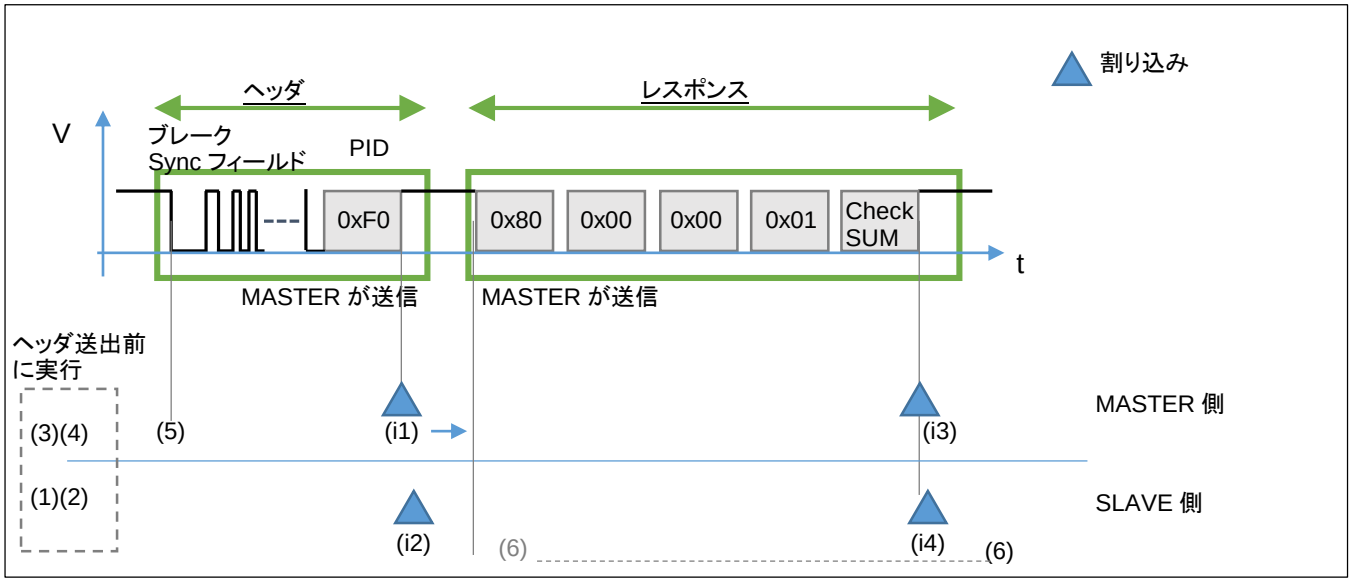


図 2-2 LIN MASTER レスポンスでのデータパケット

各処理を時系列で示すと、図 2-2 の様になります。
(5)を呼び出した際、MASTER 側からヘッダの送信が行われます。
(i1)[MASTER 側送信割り込み]をトリガに、MASTER 側はレスポンス送信を行います。
(6)の処理はループ等で定期的に行い、(i4)でフラグがセットされた後、有効データが読み出せます。

2.5. サンプルプログラムフローチャート

2.5.1. SLAVE レスポンス送信

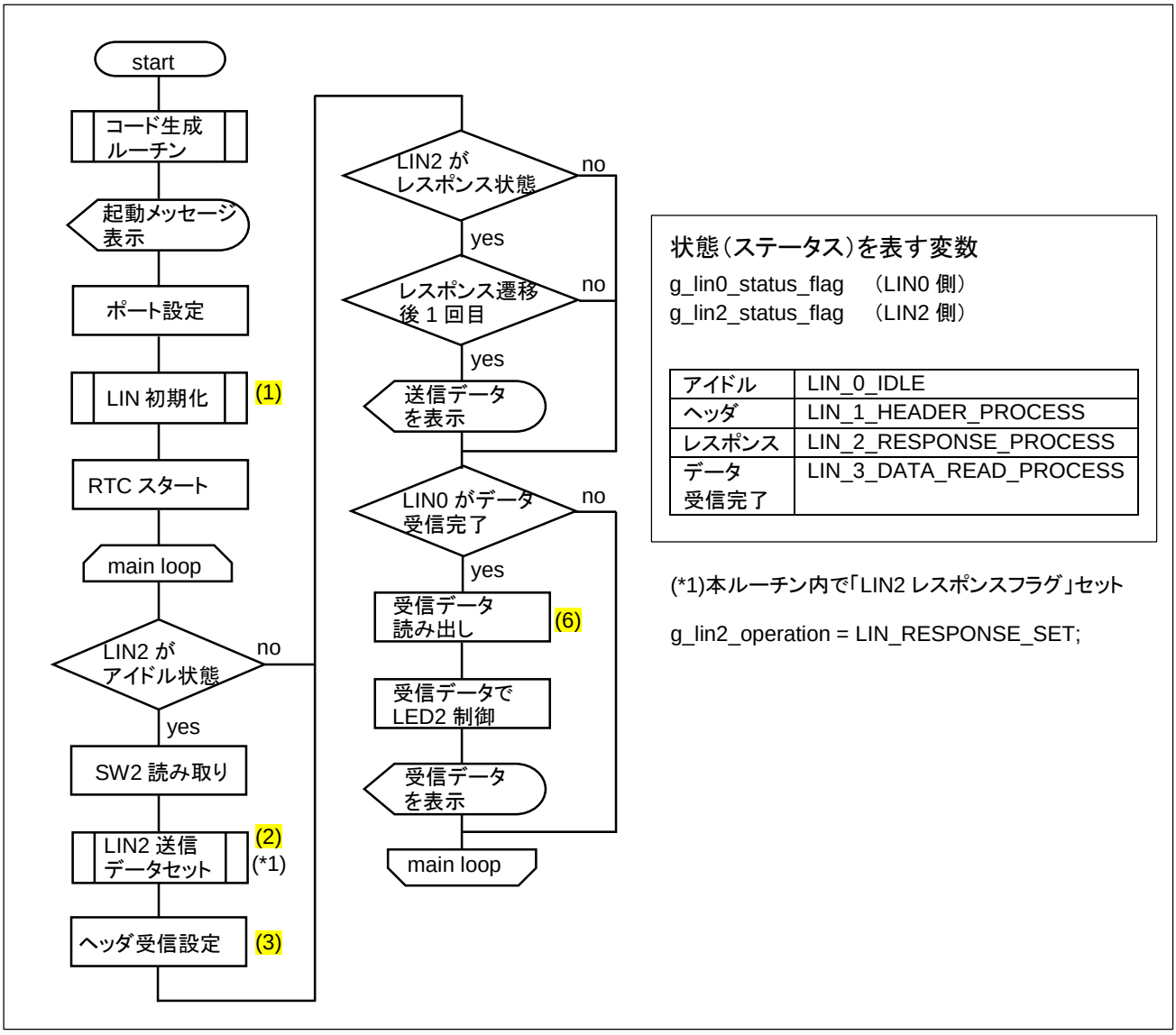
付属 CD

SOURCE¥RL78F15_LIN

以下に収録されています。

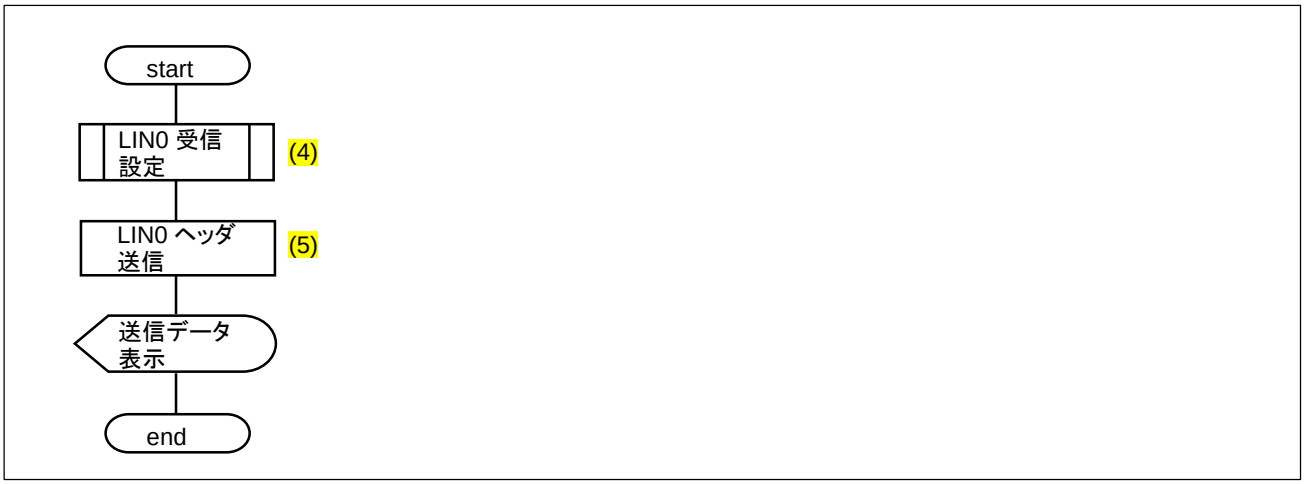
－処理フロー－

メイン関数　main()

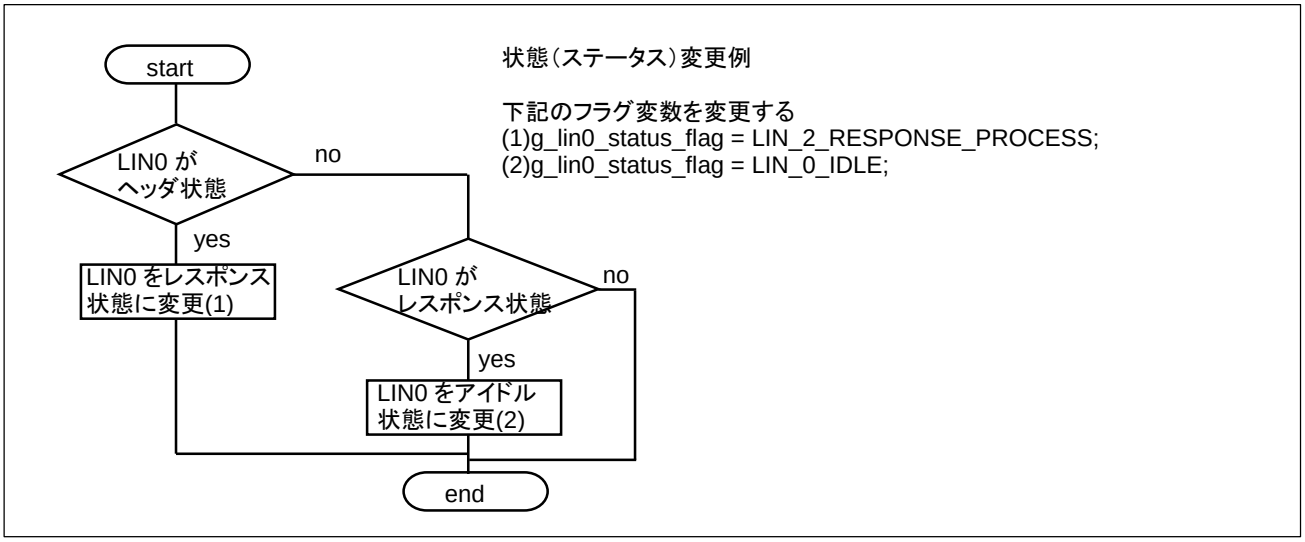


※コード生成ルーチン部は、CS+のコード生成機能で生成した、クロックの初期化やシリアルの初期化を行う

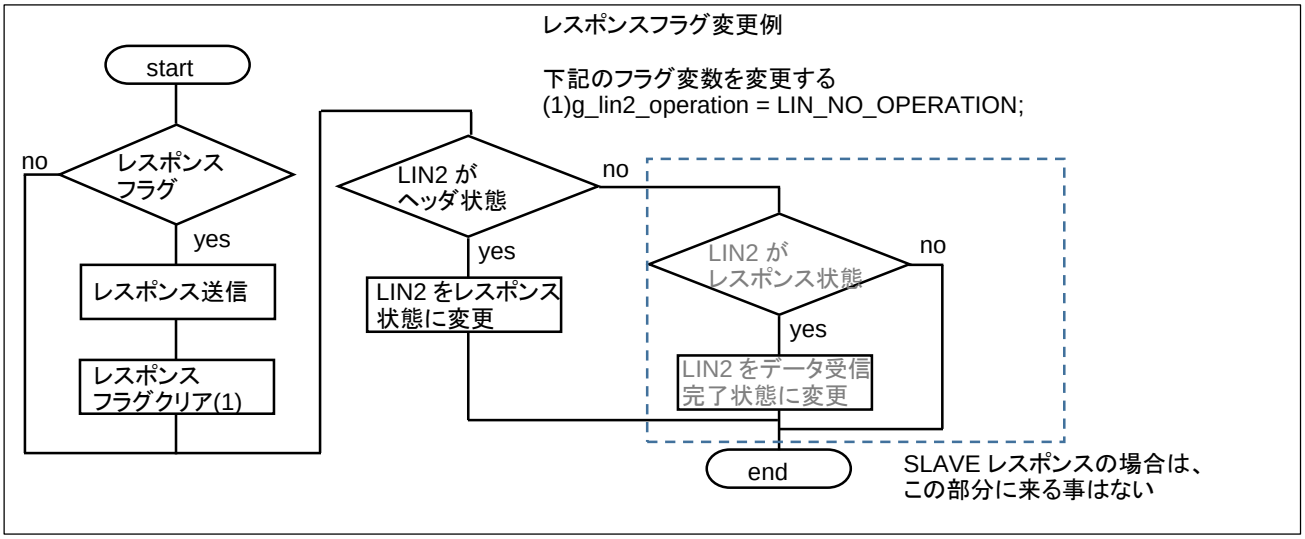
RTC 割り込み関数 r_rtc_callback_constperiod() [1 秒に 1 回実行される]



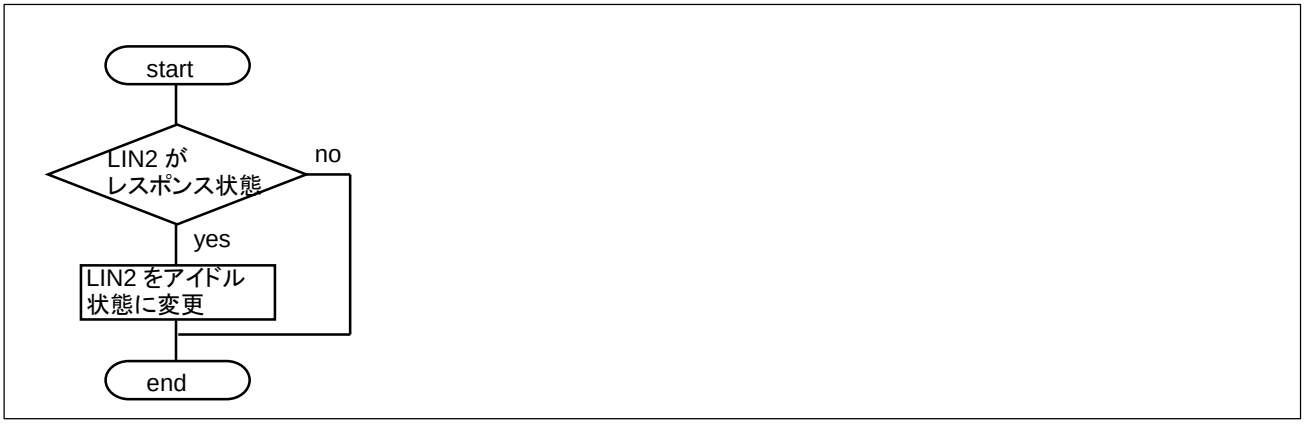
LIN0 送信割り込み関数 lin0_interrupt_send() [MASTER，受信モード時] (i1)



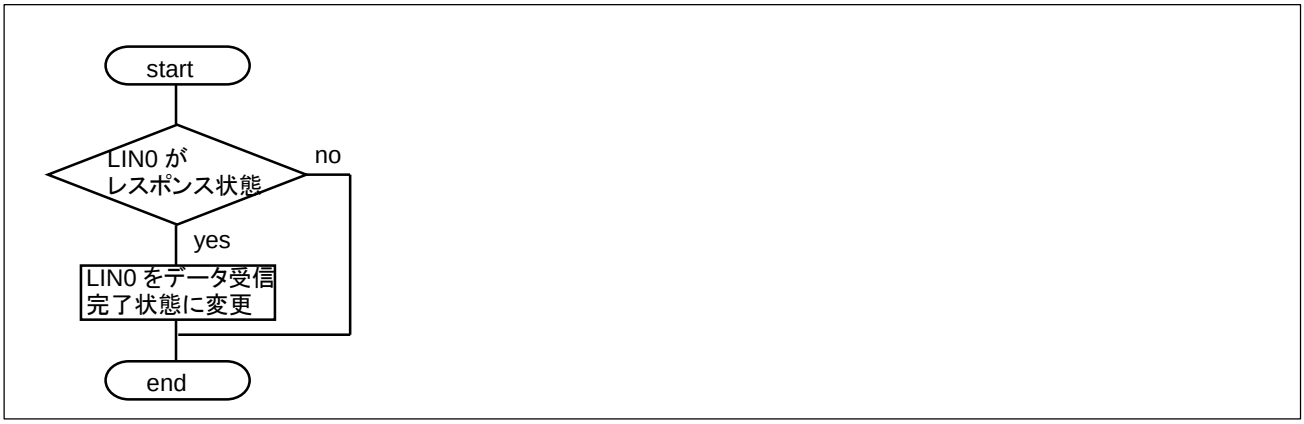
LIN2 受信割り込み関数 lin2_interrupt_receive() [SLAVE，送信モード時] (i2)



LIN2 送信割り込み関数　lin2_interrupt_send()　[SLAVE，送信モード時]　(i3)



LIN0 受信割り込み関数　lin0_interrupt_receive()　[MASTER，受信モード時]　(i4)



2.4.1「SLAVE レスポンス送信」で説明しているプログラム動作の内

メイン関数で、(1)(2)(3)(6)

RTC 割り込み関数で、(4)(5)

LIN0 送信割り込み関数　(i1)

LIN2 受信割り込み関数　(i2)

LIN2 送信割り込み関数　(i3)

LIN0 受信割り込み関数　(i4)

の処理を行っています。

2.5.2. MASTER レスポンス送信

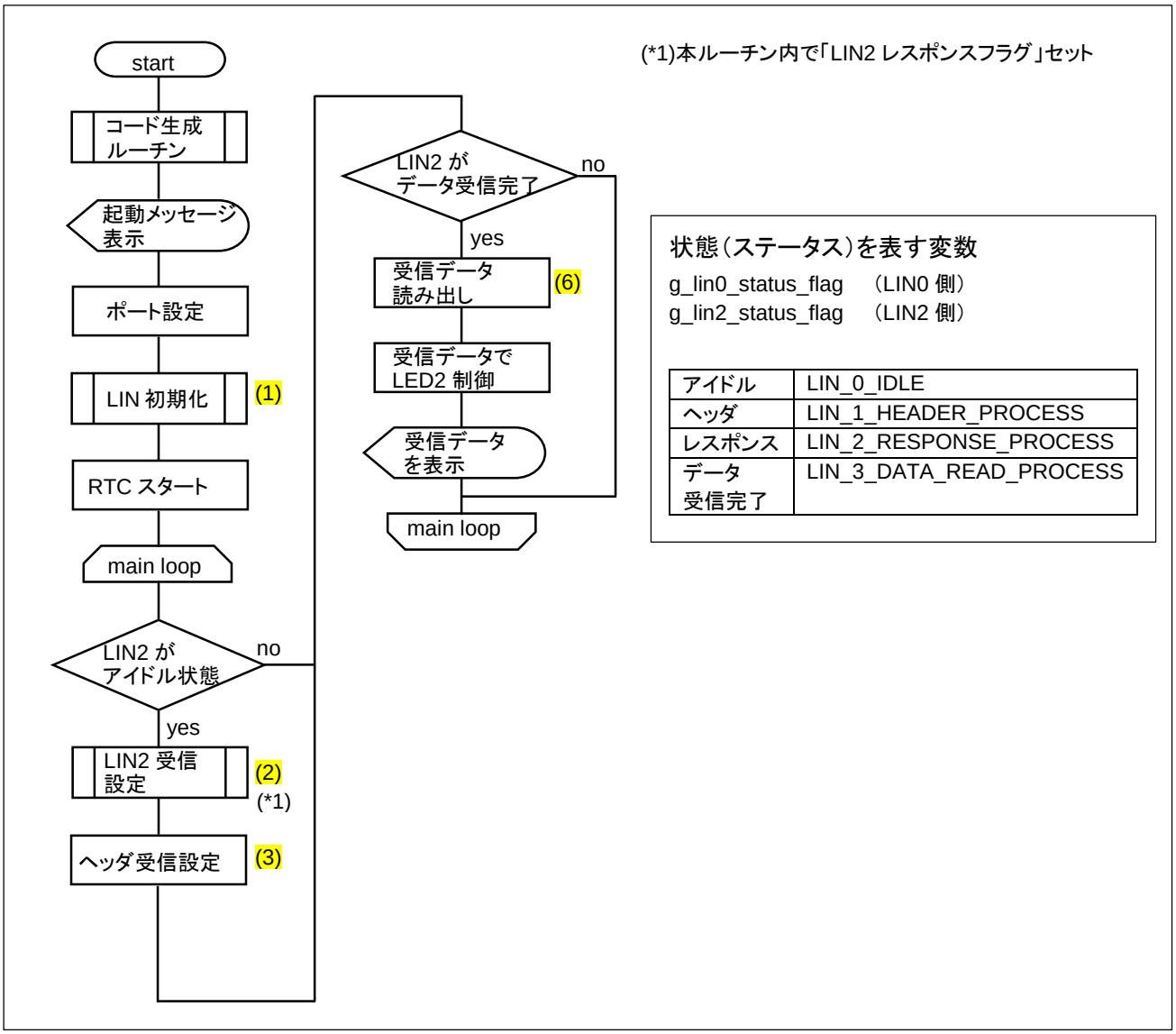
付属 CD

SOURCE¥RL78_F15_LIN2

以下に収録されています。

ー処理フローー

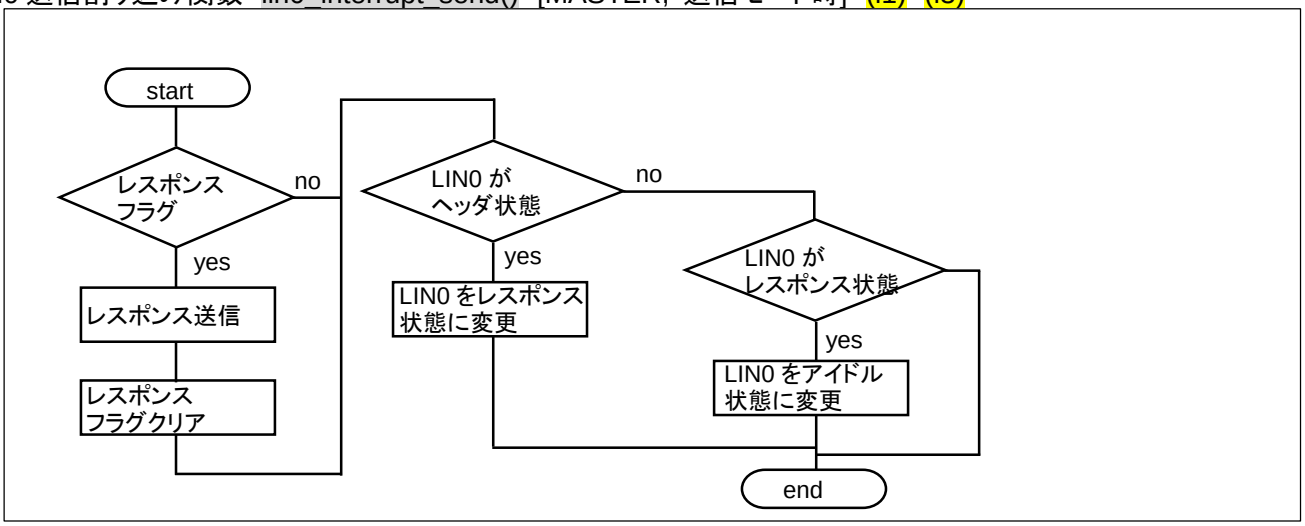
メイン関数　main()



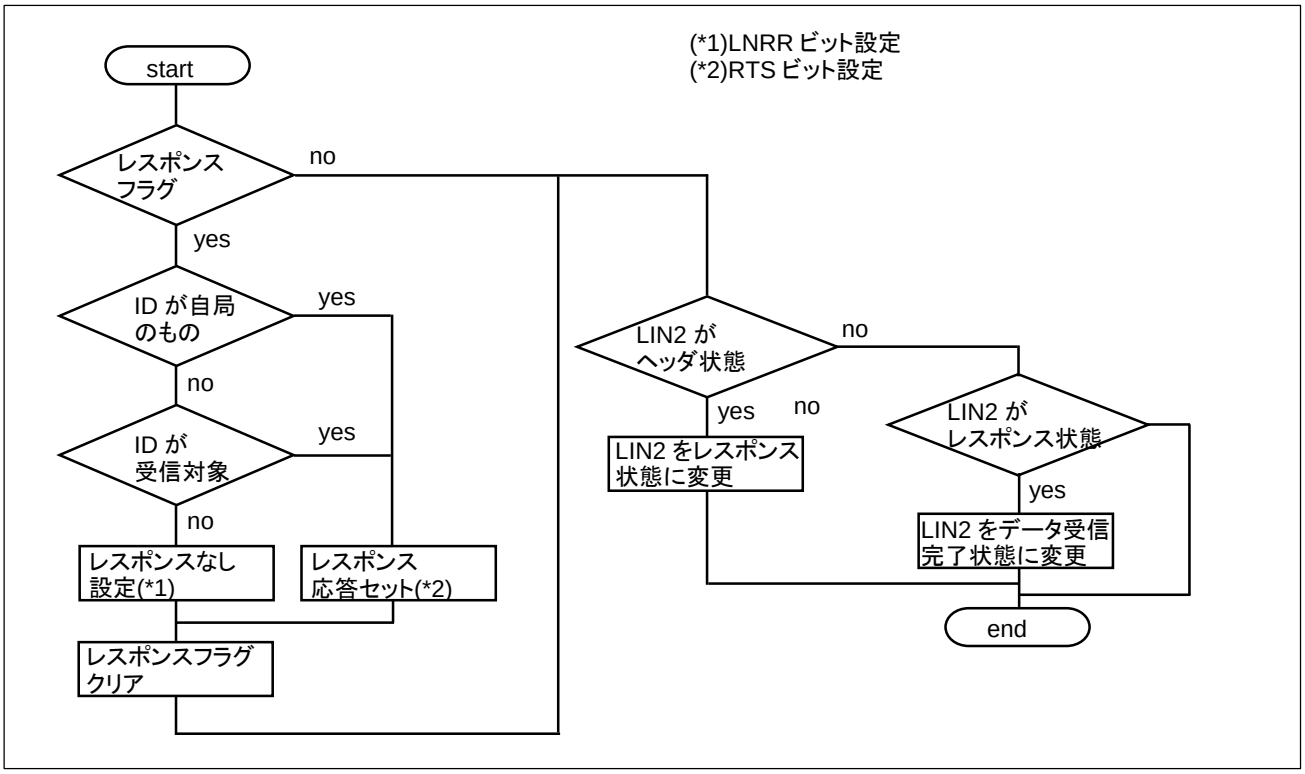
※コード生成ルーチン部は、CS+のコード生成機能で生成した、クロックの初期化やシリアルの初期化を行う

RTC 割り込み関数 r_rtc_callback_constperiod() [1 秒に 1 回実行される]

LIN0 送信割り込み関数 lin0_interrupt_send() [MASTER，送信モード時] (i1) (i3)



LIN2 受信割り込み関数 lin2_interrupt_receive() [SLAVE，受信モード時] (i2) (i4)



2.4.2「MASTER レスポンス送信」で説明しているプログラム動作の内

メイン関数で、(1)(2)(3)(6)

RTC 割り込み関数で、(4)(5)

LIN0 送信割り込み関数 (i1)(i3)

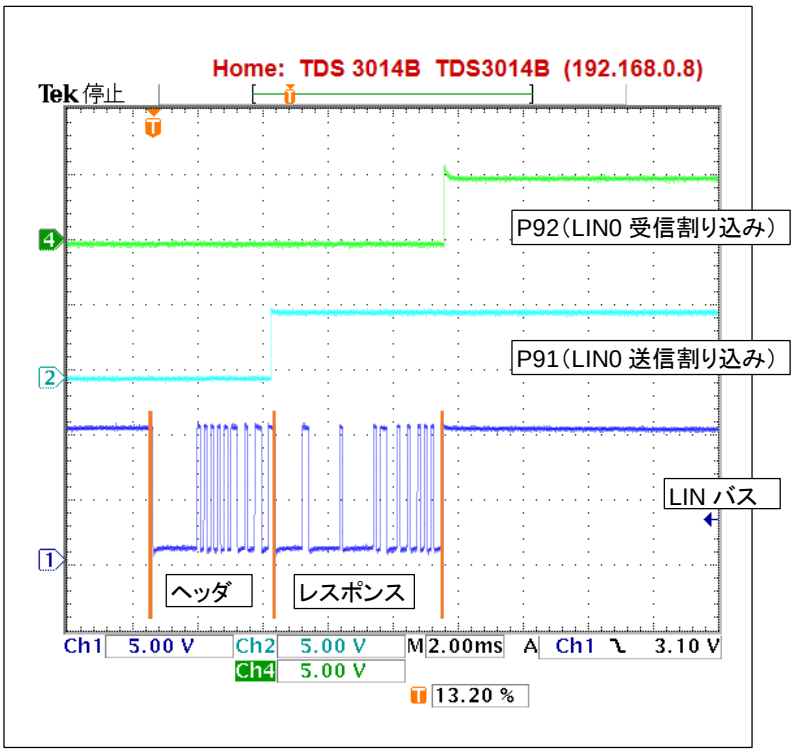
LIN2 受信割り込み関数 (i2)(i4)

の処理を行っています。

2.6. 割り込みタイミングのデバッグに関して

LIN では、合計 6 種（送信、受信、ステータス）×2ch の割り込みがあり、プログラムの動作を見る際、どの割り込みが、どのタイミングで発生しているかを知りたいときがあります。

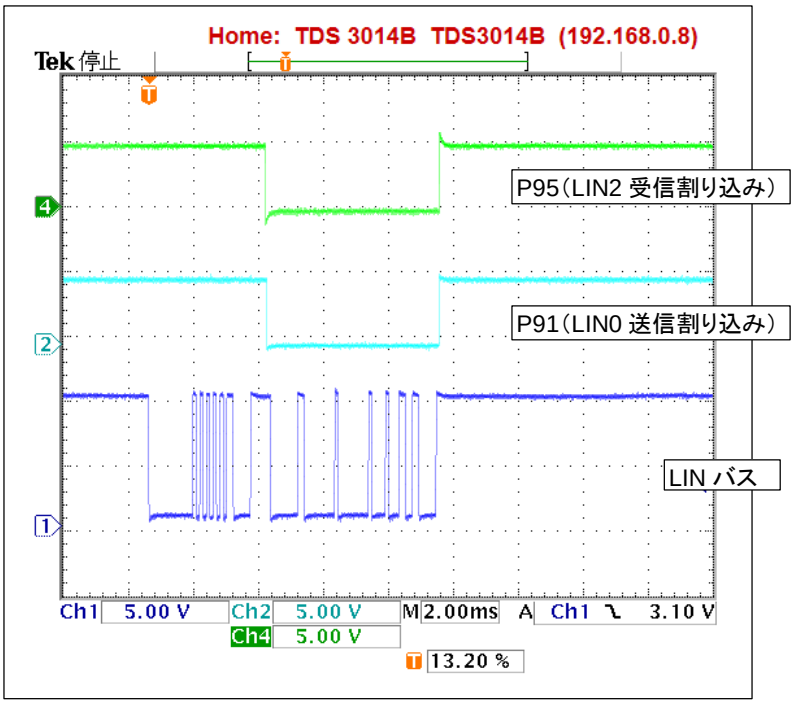
(1)LIN SLAVE レスポンスの波形例



ヘッダの送信終了時に、LIN0 の送信割り込み。レスポンス受信完了のタイミングで LIN0 の受信割り込みが入っている事が確認できる
（1 回のパケットで 1 回割り込みが入る）

P90(J2-1)	LIN0 送信割り込み
P91(J2-2)	LIN0 受信割り込み
P92(J2-3)	LIN0 ステータス割り込み
P93(J2-4)	LIN2 送信割り込み
P94(J2-5)	LIN2 受信割り込み
P95(J2-6)	LIN2 ステータス割り込み

(2)LIN MASTER レスポンスの波形例



ヘッダの送信終了時とレスポンスの送信完了時に、LIN0 の送信割り込み。ほぼ同タイミングで、LIN2 の受信割り込みが入っている事が確認できる。
（1 回のパケットで 2 回割り込みが入る）

INTERRUPT_DEBUG 定数を有効にしている場合、割り込みが入るタイミングでポート(P90~P95)の信号が反転するので、どのタイミングで割り込みが生じているかモニタ可能です。

3. CAN サンプルプログラム

3.1. プログラム仕様

・CAN0 から送信
・CAN1 で受信

を行うサンプルプログラムとする。

3.2. 関数仕様

3.2.1. ユーザ関数

can_init

概要:初期化関数
宣言:
　　int can_init(void)
説明:
　・端子設定
　・レジスタの設定
　・CAN 動作モード遷移
を行います
引数:　なし
戻り値:
　　0:　正常終了

　通信レート 1Mbps
　CAN0　送信先 ID=0x31,　受信 ID=0x30
　CAN1　送信先 ID=0x30,　受信 ID=0x31
で設定

can0_send　　（CAN ch0 向け）
can1_send　　（CAN ch1 向け）

概要:データ送信関数
宣言:
　　int can0_send(unsigned char *data, unsigned char length)
　　int can1_send(unsigned char *data, unsigned char length)
説明:
　・データの送信
を行います
引数:
　　*data: 送信データを設定
　　length: データ長(0~8)を設定
戻り値:
　　0: 正常終了
　　-1: エラー

can0_receive　　（CAN ch0 向け）
can1_receive　　（CAN ch2 向け）

概要:受信関数
宣言:
　　can0_receive(unsigned char *data, unsigned short *id, unsigned short *label, unsigned short *time_stamp)
　　can1_receive(unsigned char *data, unsigned short *id, unsigned short *label, unsigned short *time_stamp)
説明:
　・データの受信
を行います
引数:
　　*data: 受信したデータが格納されます（8 バイト以上の領域を確保してください）
　　*id: ID が格納されます
　　*label: ラベル情報が格納されます
　　*time_stamp: タイムスタンプが格納されます
戻り値:
　　0<: 受信バイト数
　　-1: 受信データなし

メールボックスに受信データが届いている場合は、0 以上の値を返します。
初期設定で、メールボックスは上書きするよう設定していますので、複数回データを受信しているときは、最近のものを返します。

3.3. プログラムで使用している変数・定数

3.3.1. 定数定義

#define CAN0_USE

CANch0 を使用する場合に定義

#define CAN0_EN_USE

CAN0 EN(STB)信号を、P12 経由でマイコンから制御する場合に定義

#define CAN1_USE

CANch1 を使用する場合に定義

#define CAN1_EN_USE

CAN1 EN(STB)信号を、P62 経由でマイコンから制御する場合に定義

#define CAN0_ID 0x30
#define CAN1_ID 0x31

CAN の ID 設定（受信側 ID）

#define CAN0_LABEL 0xE30
#define CAN1_LABEL 0xE31

データに付与するラベルの設定

3.4. 動作説明

実際に CAN の通信を行う際、3.2 記載の関数をどのような流れで呼び出すかを以下で説明します。
データの送信元は、CAN0 で、受信側は CAN1 です。

(1)初期化を行う

can_init();

(2)CAN0 側からデータを送信する

can0_send(buf, 4);

(3)CAN1 側でデータを受信する

ret = can1_receive(can_buf, &can_id, &can_label, &can_time_stamp);

ret（can1_receive()の戻り値）を確認し、0 以上であればデータ受信が完了しています（メールボックスにデータが届いています）。

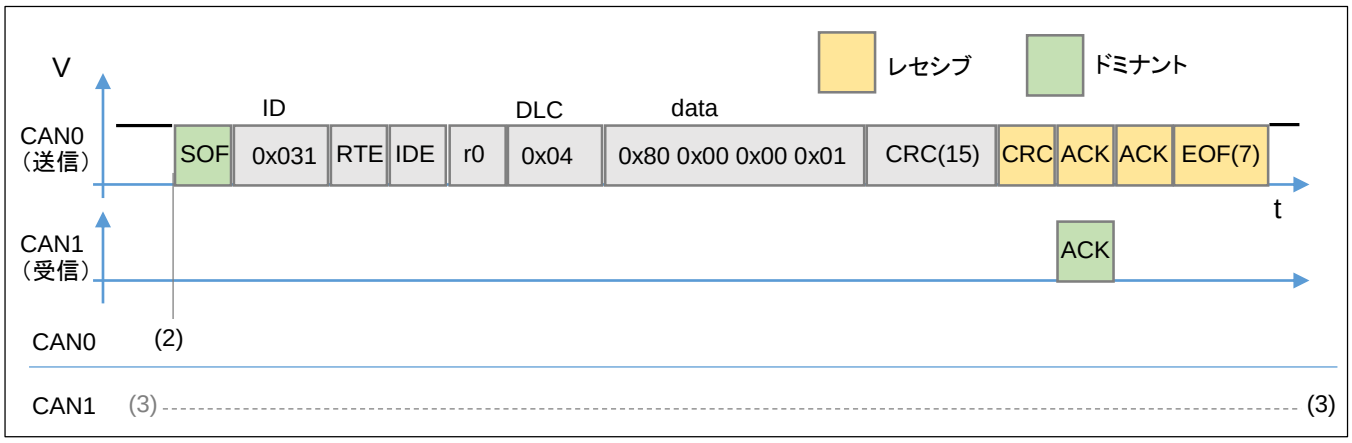


図 3-1 CAN0→CAN1 データ送信のデータパケット

(2)を呼び出したタイミングデータが送出され（もし、CAN バスにつながる他のユニットがバスを使用中の場合は、調停が行われます）ます。

CAN の場合は、CAN1 側を初期化していれば、CAN1 の受信ルールにマッチしたデータが来た場合、CAN1 の ACK は自動的に返します。（LIN の場合は、割り込みでレスポンスを返すよう、プログラム上の処理が必要でしたが、CAN の場合は、データのやり取りはマイコンが処理を行ってくれます）

プログラム的には、CAN1 側のメールボックスを定期的に確認し、有効データがあれば読み出しを行うという処理となります。

3.5. サンプルプログラムフローチャート

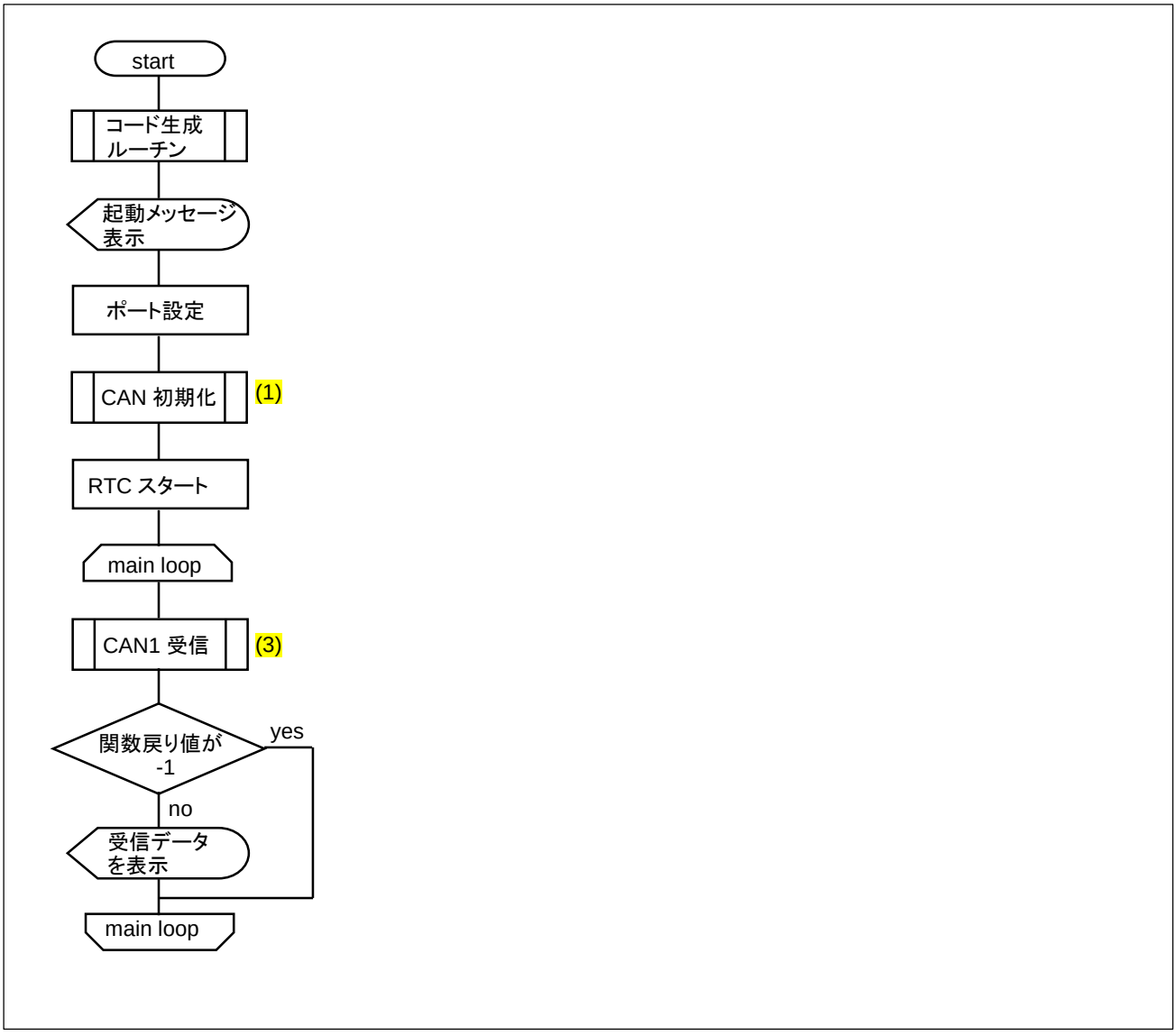
付属 CD

SOURCE¥RL78_F15_CAN

以下に収録されています。

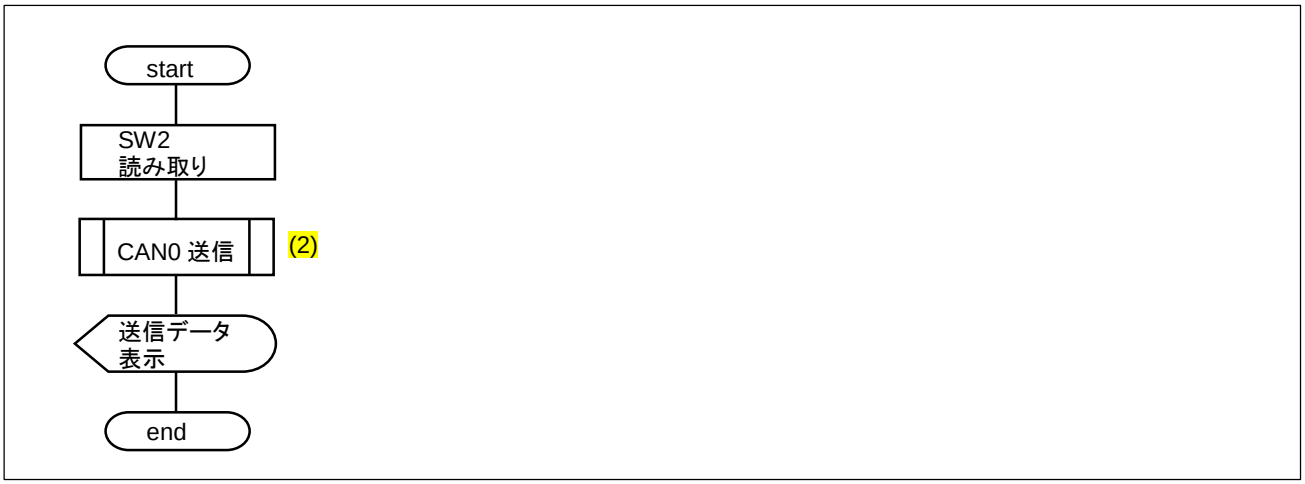
－処理フロー－

メイン関数　main()



※コード生成ルーチン部は、CS+のコード生成機能で生成した、クロックの初期化やシリアルの初期化を行う

RTC 割り込み関数　r_rtc_callback_constperiod()　[1 秒に 1 回実行される]



3.4 で説明しているプログラム動作の内

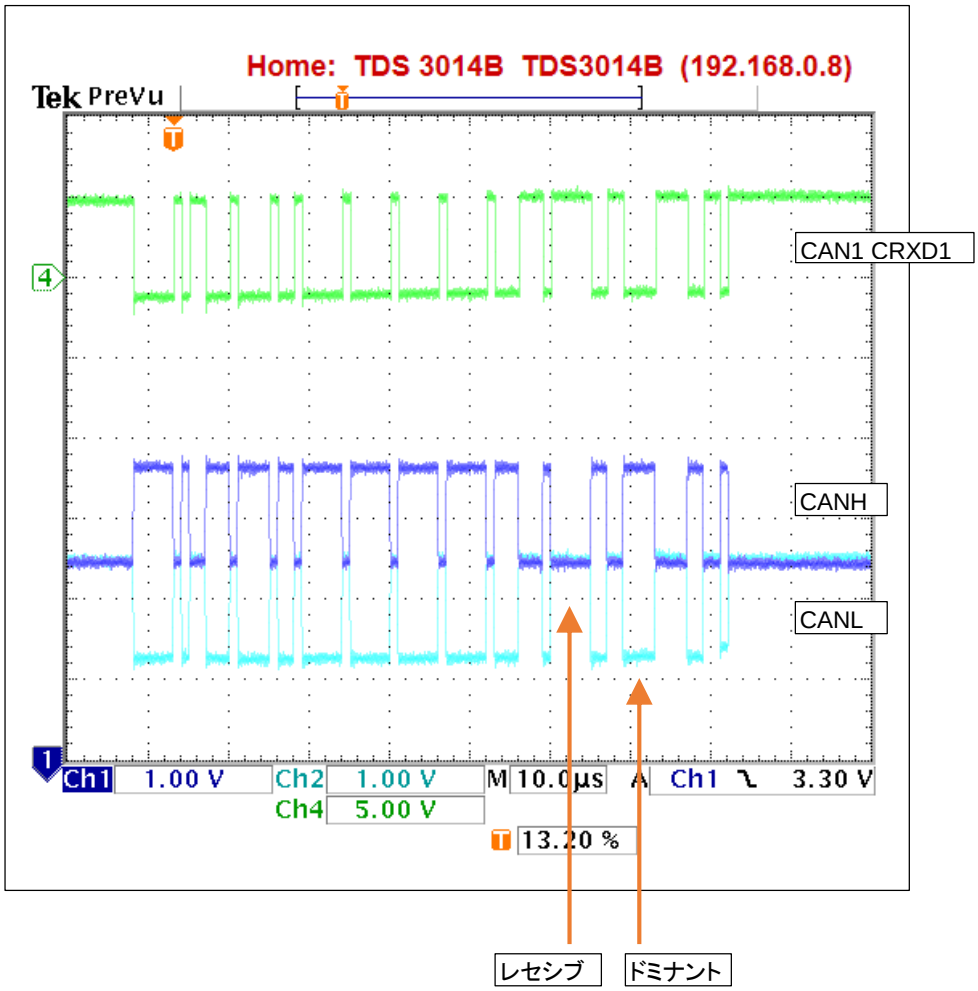
メイン関数で、(1)(3)

RTC 割り込み関数で、(2)

の処理を行っています。（本サンプルでは、CAN の割り込みは使用していません）

3.6. 波形例

下記に、CAN で通信を行った際の、CAN バスと、CAN トランシーバを通した後の波形を示します。



CANH, CANL は CAN バスの波形で、ドミナント、レセシブの 2 値を取ります。

CAN1 CRXD1 は、CAN トランシーバ IC を通し、CMOS レベル（L/H）に変換された後の波形となります。

4. デバッガを使用したデバッグに関して

本キットには、デバッガは付属しておりません。デバッガを使用してデバッグを行う場合は、別途

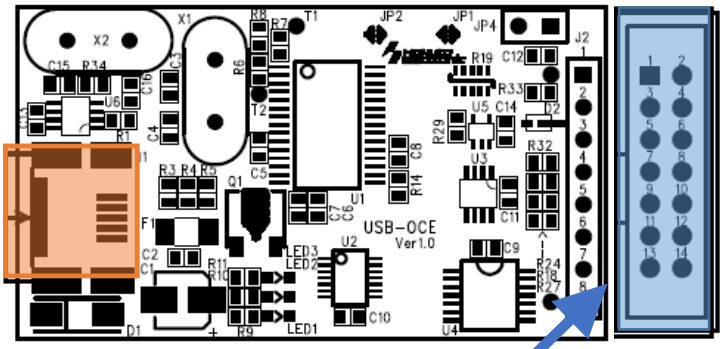
ルネサスエレクトロニクス製　E1（または、E2, E2Lite, E20）
北斗電子製　USBOCE-RL78-3-14

のいずれかをご用意ください。

・USBOCE-RL78-3-14 を接続する場合

USBOCE 経由で
マイコンボードに給電する場合 JP4 ショート

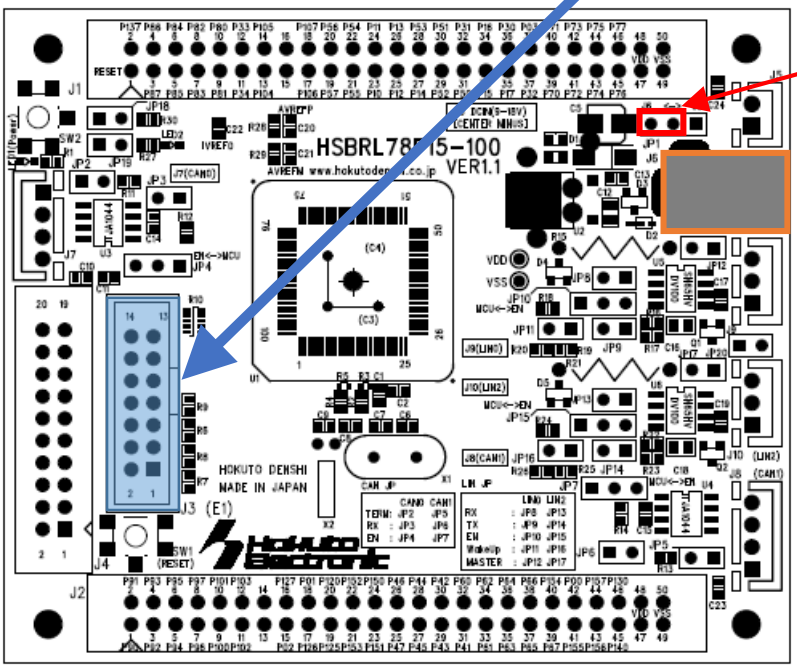
USBOCE-RL78-3-14



USB-miniB
ケーブルで PC と
接続

14P コネクタ同士
を USBOCE 付属
ケーブルで接続

HSBRL78F15-100



JP1
電源選択ジャンパ

付属 AC アダプタ
を接続

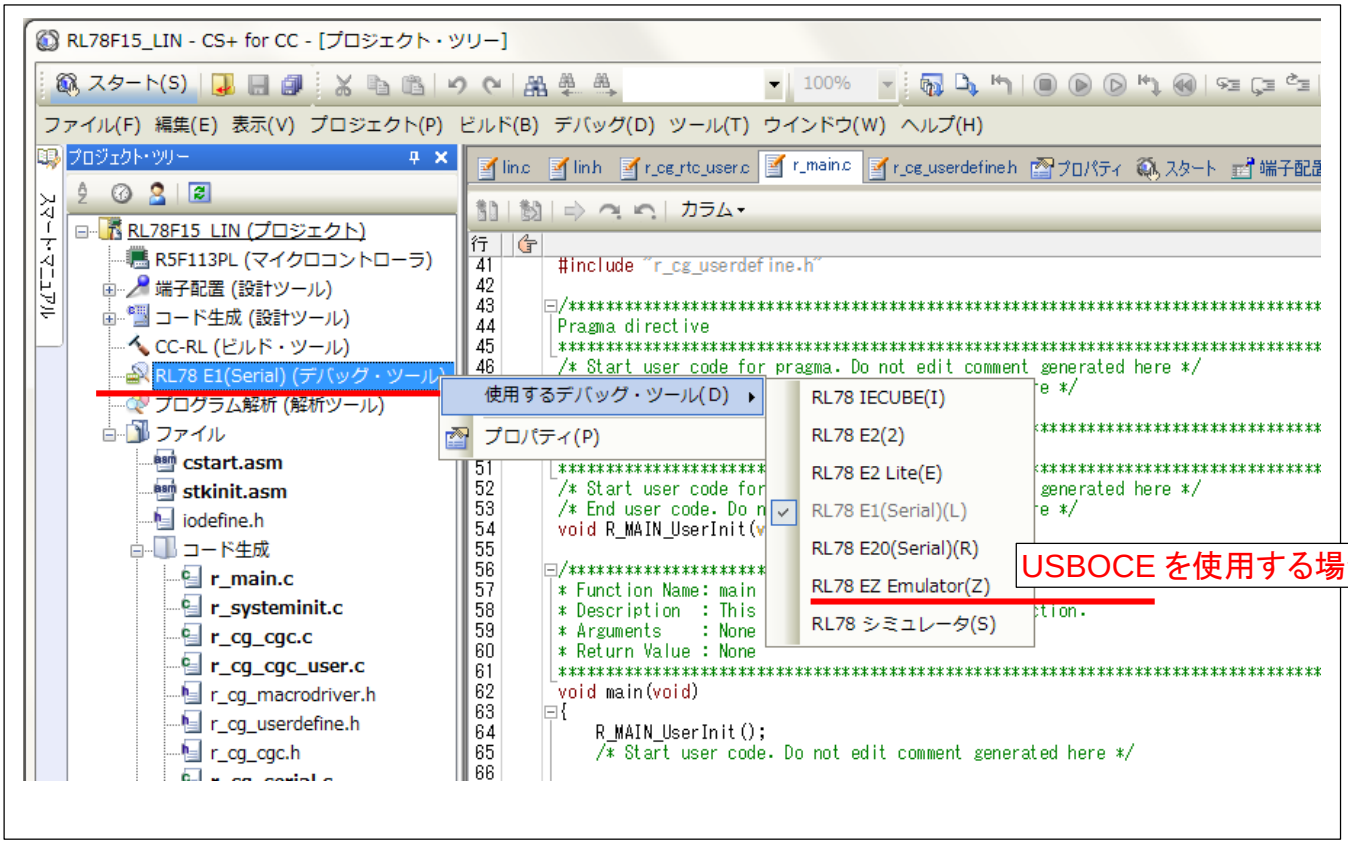
※LIN を使用する場合は、AC アダプタで給電してください

※LIN を使用しない場合は、USBOCE からの給電でも使用できます

デバッガは、マイコンボード J3(14P)コネクタに接続してください。

※USBOCE からマイコンボード VDD に給電することも可能ですが、マイコンボードに対する電源の供給元は 1 箇所となるようにしてください（USBOCE から給電する場合は、マイコンボード JP1 を 1-2 ショート（右側）に設定してください、USBOCE から給電した場合マイコンボード VDD は+5V となります）

デバッガ接続は、サンプルプロジェクトを開き

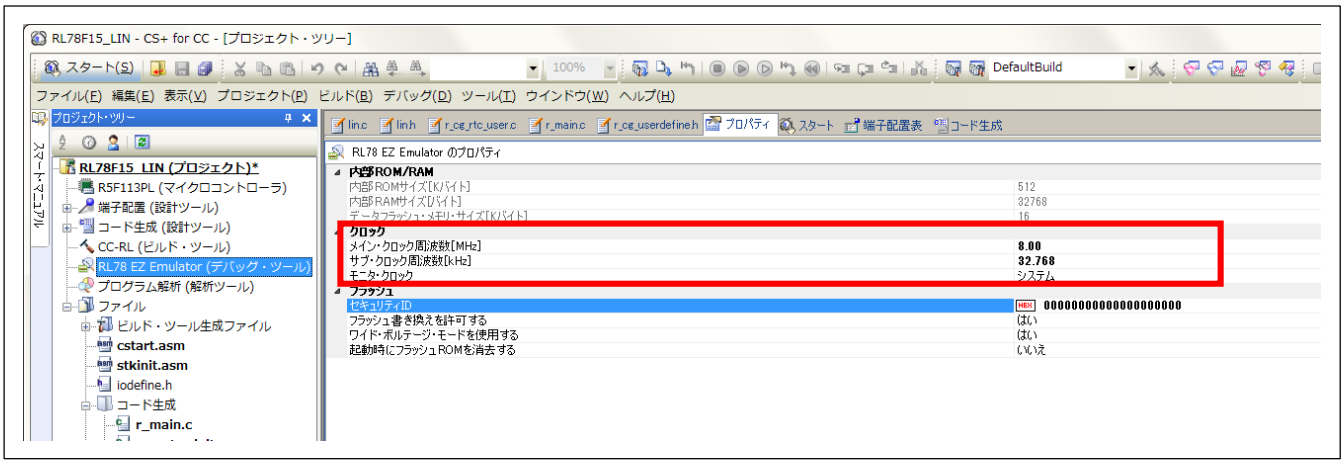


プロジェクト・ツリー内の
デバッグツール（右クリック）
　使用するデバッグ・ツール

ルネサスエレクトロニクス製のデバッガを使用する場合は、ご使用するデバッガに合わせて選択ください。
北斗電子製 USBOCE を使用する場合は、「RL78 EZ Emulator」を選択してください。

※「RL78 シミュレータ」では、マイコンボードにプログラムを転送する事ができませんのでご注意ください

・デバッグツール設定

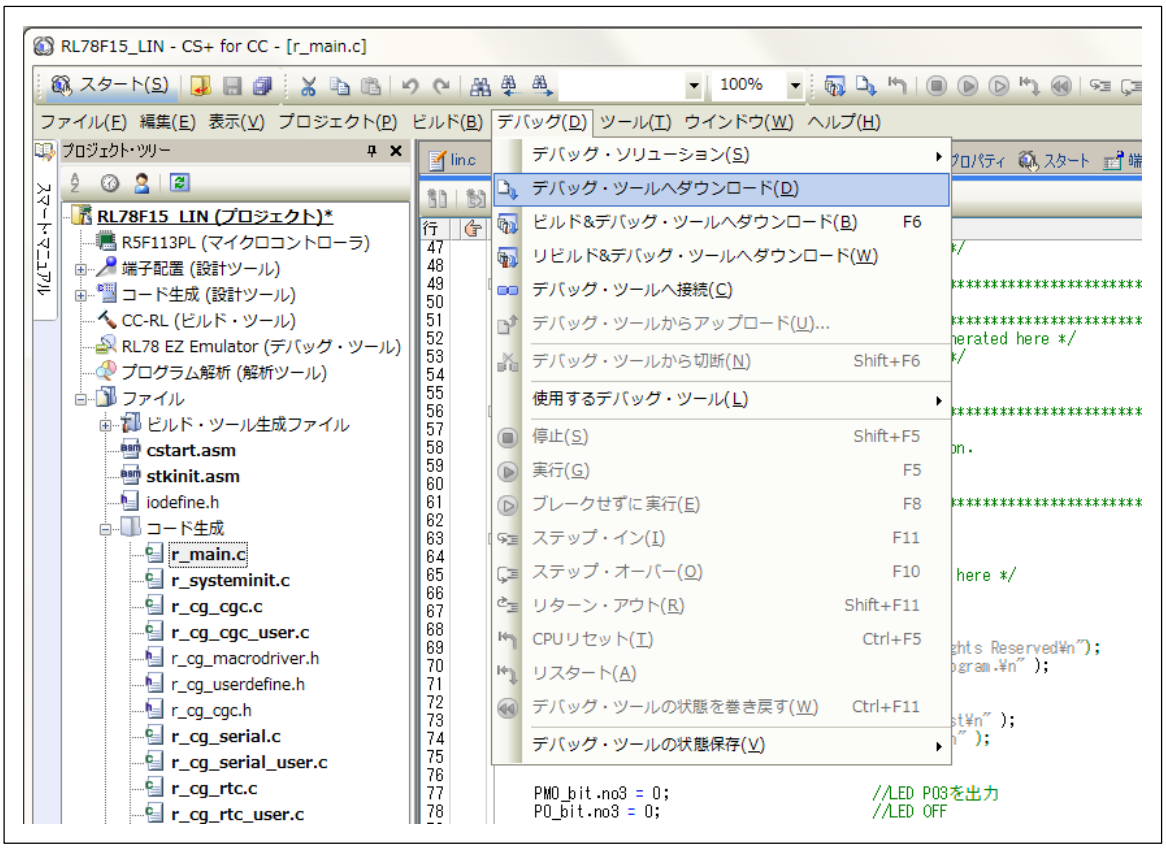


メインクロック周波数　8.00 [MHz]を選択

サブクロック周波数　32.768 [kHz]を選択

してください。（※画面は、RL78 RZ Emulator を選択した場合の例ですが、E1 を使用する場合も同様の設定を行ってください）

・デバッグツール接続



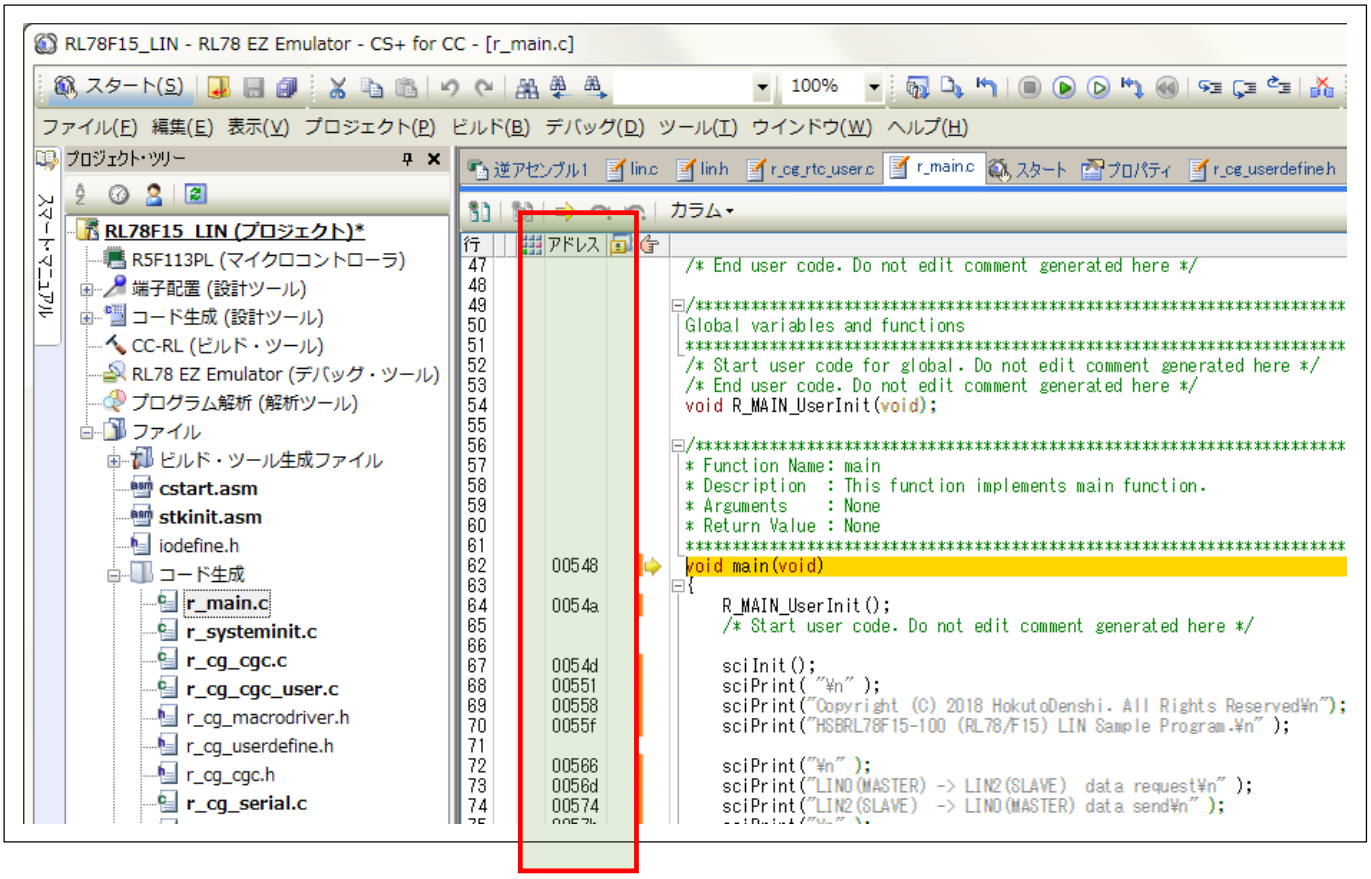
デバッグメニュー

　デバッグ・ツールへダウンロード

で「デバッガに接続」、及び「プログラムをマイコンボードに転送」できます

※プログラムソースを変更した場合は「ビルド&デバッグ・ツールへダウンロード」を選択してください

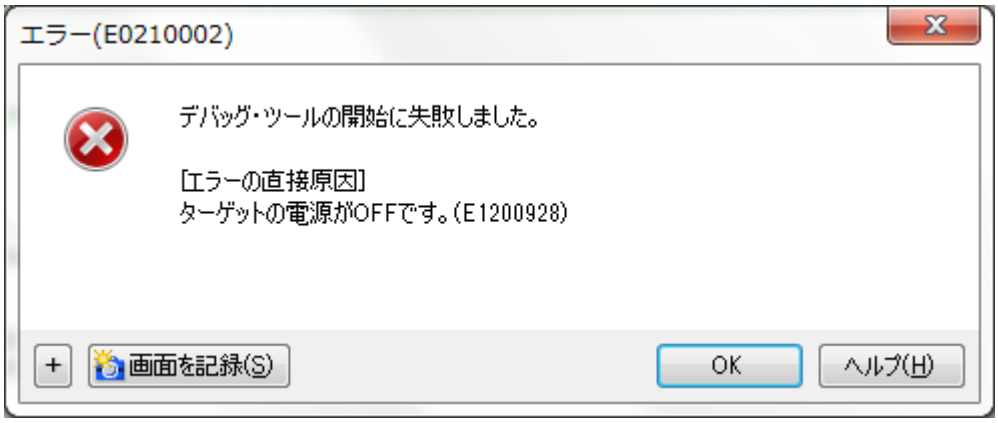
・デバッグ



接続が成功した場合は、アドレスのところに、数値が入り、オレンジのライン（プログラムが停止しているソースの行）が表示されます。

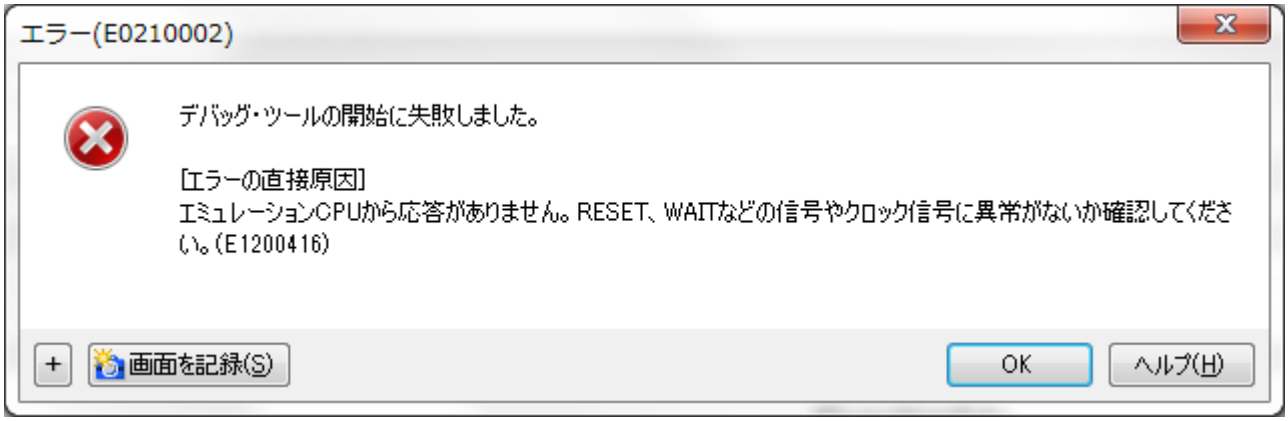
デバッガを使用した場合、特定の場所でプログラムの実行を停止したり、1 行ずつ実行したり、変数をモニタしたりする事が可能です。デバッグ機能の詳細は、CS+のマニュアルを参照ください。

・USBOCE の接続でエラーとなる場合



USBOCE の接続で、上記エラーが出た場合は、マイコンボードに給電されているかを確認してください。電源の問題ではないときは、PC に USBOCE 以外のデバッガ（ルネサス E1 等）が接続されていないかをご確認ください。USBOCE 以外のデバッガが接続されている場合は、接続を外してください。

・接続エラーとなった場合



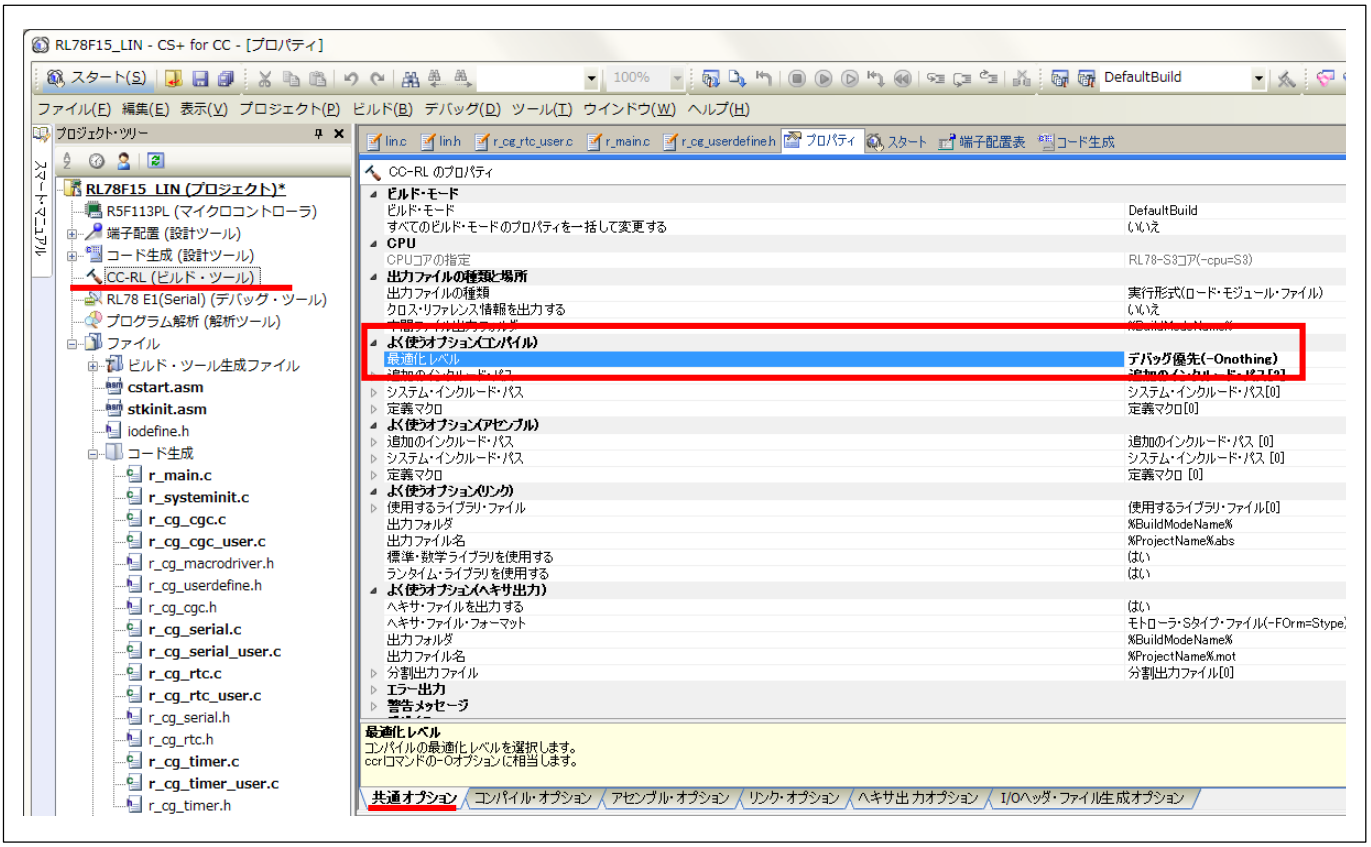
マイコンボードボードとの接続に問題がないにも拘わらず上記エラーとなった場合は、下記を設定してみてください。

デバッグ・ツール　プロパティ

　起動時にフラッシュ ROM を消去する　「はい」

オプションバイトのデバッガ設定ビットが、「オンチップ・デバッグ禁止」になっている場合、デバッガ接続が失敗します。その際は、デバッガ接続時に、オプションバイト含めフラッシュ ROM を消去すると、接続可能となります。

・デバッグ時変数値が思ったように表示されない



デバッグ時、モニタしたい変数の値が思ったように表示されないときは、

ビルド・ツール　プロパティ　　共通オプション　タブ

　　最適化レベル　「デバッグ優先」

に変更してみてください。

（本オプション変更後は、プロジェクトのビルド及び、マイコンボードへの再ダウンロードが必要）

コンパイル時に最適化が行われず、変数の変化が追いやすくなるケースがあります。

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	2018.04.06	—	初版発行
REV.1.1.0.0	2022.8.31	P5	CD の内容を拡充

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
ソフトウェア編　マニュアル

株式会社 北斗電子
 2022 年 8 月 31 日改訂 REV.1.1.0.0 (220831)