お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

H8/300H Tiny シリーズ　H8/36049 グループ

LIN (Local Interconnect Network) マスタ編

要旨

LIN (Local Interconnect Network) アプリケーションノート H8/300H Tiny シリーズ　H8/36049 グループ マスタ編は，H8/300H Tiny シリーズ　H8/36049 グループマイコンの内蔵周辺機能を使用し，LIN 通信プロトコルによる通信を行なう使用例，設定例を記しており，ユーザにてソフトウェア設計およびハードウェア設計の際，ご参考として役立てていただけるようにまとめたものです。

なお，本アプリケーションノートに掲載されているプログラム，回路等の動作は確認しておりますが，実際にご使用になる場合は，必ず動作確認の上ご使用くださいますようお願いいたします。

動作確認デバイス

H8/300H Tiny シリーズ　H8/36049F

目次

1. LIN 通信システム概要

本アプリケーションノートに掲載している LIN 通信用ソフトウェアライブラリサンプル (以下 LIN2.0 ライブラリと称す) を組み込んだシステムによる LIN 通信の概要を示します。

1.1 LIN バスへの接続

LIN バスによりネットワーク接続されたシステムに LIN バスインタフェース回路 (または LIN トランシーバ) を介して接続することにより，マスタノードとしてヘッダフレームの送信，レスポンスフレームの送受信等の LIN 通信を行ないます。

1.1.1 システム構成

図 1 に LIN バスネットワークシステム構成例を示します。

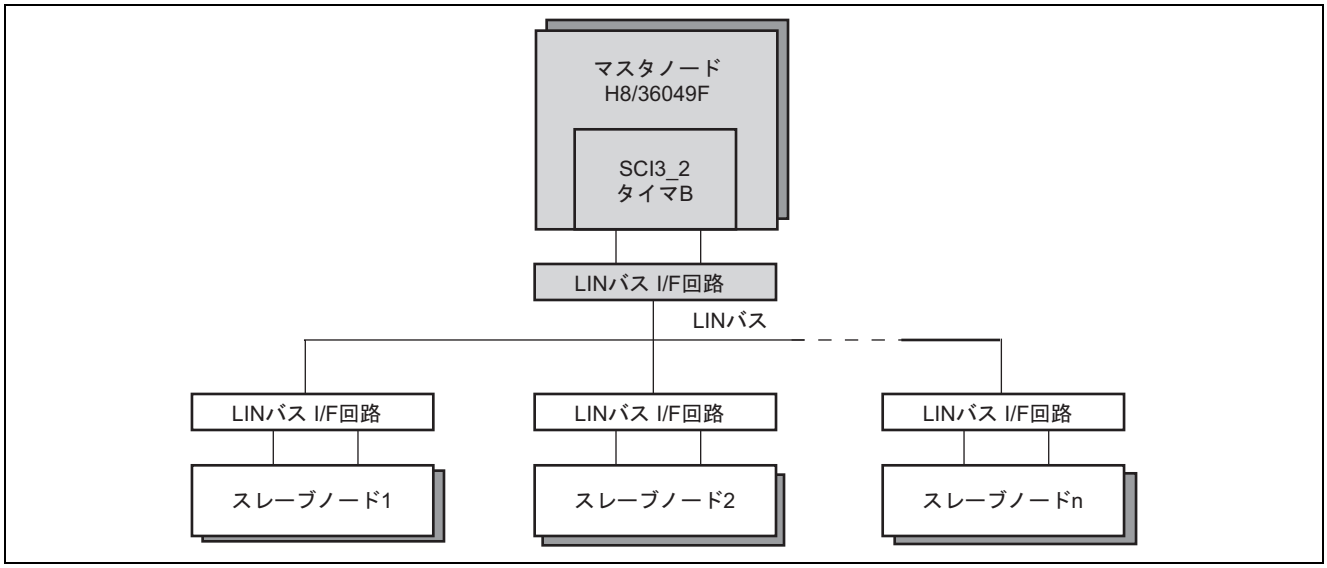

図 1　システム構成

1.1.2　使用リソース

表 1 に H8/36049F で使用するリソースを示します。

表 1　マスタノードが使用する CPU リソース

機能		端子機能 (端子 No.)	用途	説明
I/O ポート		P60 (42pin)	LIN トランシーバ制御	I/O ポート (High/Low) 出力により LIN トランシーバ有効/無効切り替え (ユーザによりリセット時 High/出力に設定)
SCI3 (channel-2)	送信	TXD_2 (72pin)	ヘッダ送信 レスポンスフレーム送信 WakeUp 信号出力	調歩同期式モード データ長 8 ビット パリティビット無し 1 ストップビット (スタートビット付加) LSB ファースト
	受信	RXD_2 (71pin)	レスポンスフレーム受信	
			通信エラー検出	モジュール内エラー検出機能
タイマ B1		—	Break デリミタ期間計測 WakeUp 信号期間計測	通信速度に合わせた Break 期間を計測

1.2　LIN 通信概要

LIN 通信プロトコルにより送受信される各フレームの概要を示します。

1.2.1　無条件フレーム (Unconditional Frame)

無条件フレームは，信号値更新の有無にかかわらず，必ず送受信されるフレームです。

ヘッダに対するレスポンス送信ノードはマスタノード/スレーブノードのいずれかになります。また，レスポンス受信ノードもマスタノード/スレーブノードのいずれかになります。

図 2 に無条件フレームのシーケンス内容を示します。

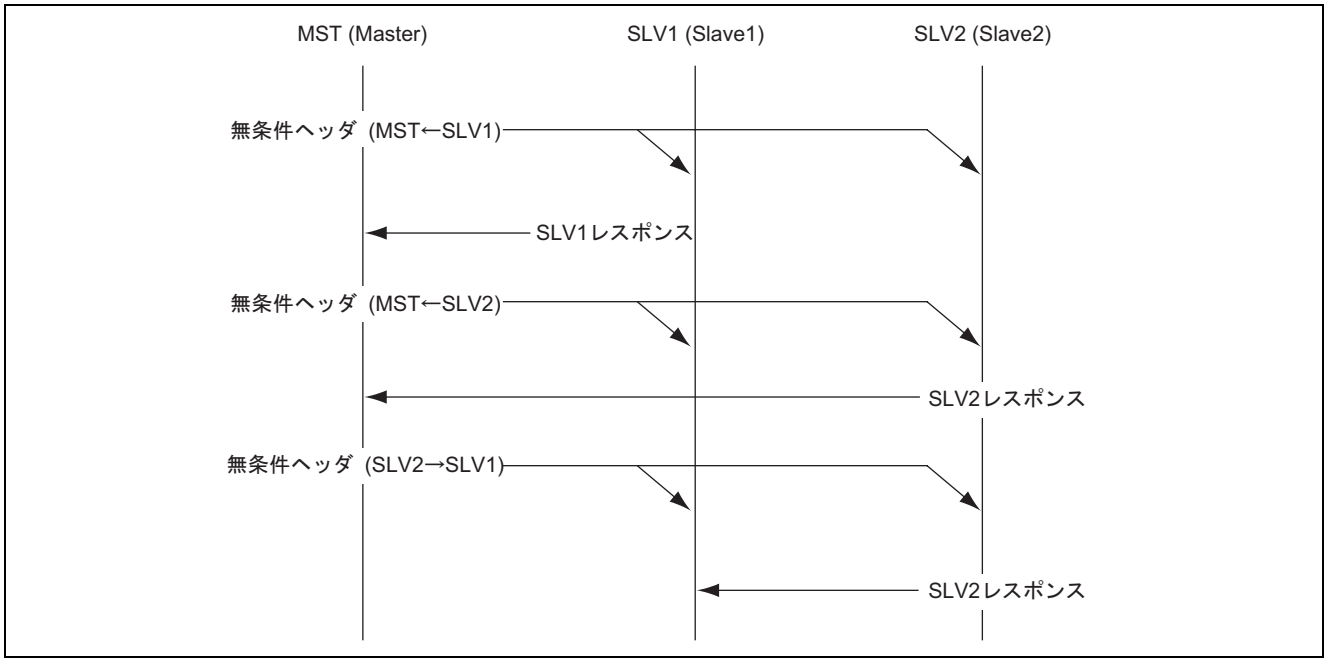

図 2　無条件フレームのシーケンス内容

1.2.2　**イベントトリガフレーム** (Event Triggered Frame)

イベントトリガフレームは，マスタノードがスレーブノードに対し，信号値更新の有無を確認するために送受信されるフレームです。

ヘッダに対するレスポンス送信ノードは信号値に更新があったスレーブノードからのみとなります。このとき，複数のスレーブノードがレスポンスを送信する場合があり，その場合にはコリジョンが発生します。コリジョン発生時にはマスタノードが一旦すべてのスレーブノードに対して，無条件フレームにより信号値の確認を行ないます。また，レスポンス受信ノードはマスタノードのみとなります。

図 3 にイベントトリガフレームのシーケンス内容を示します。

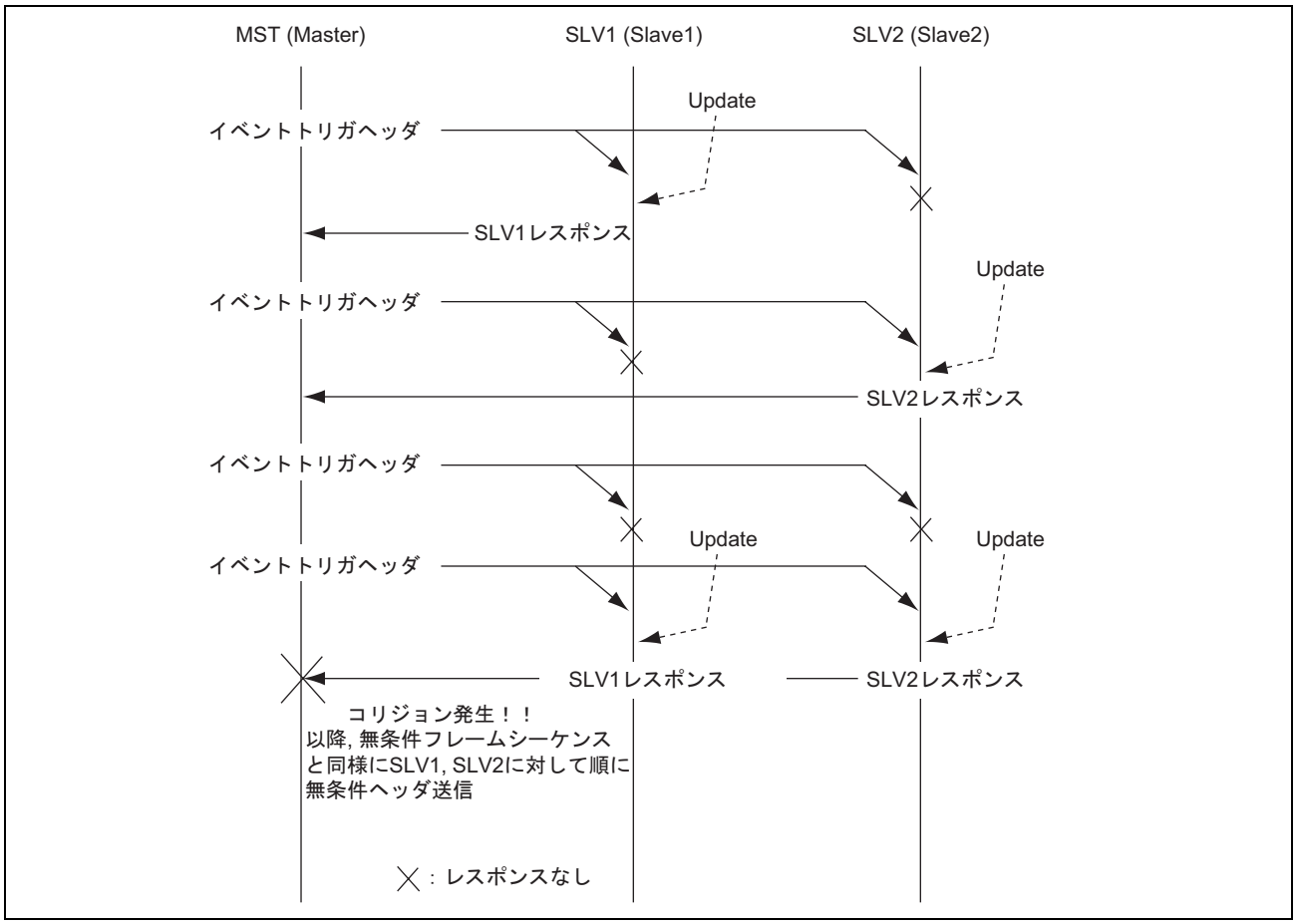

図 3　イベントトリガフレームのシーケンス内容

1.2.3　**散発フレーム** (Sporadic Frame)

散発フレームは，マスタノードが管理する信号値に更新があった場合に，関連するスレーブノードすべてに対して，信号値の通知を行ないます。ヘッダに対するレスポンス送信ノードは必ずマスタノードのみからとなります。

図 4 に散発フレームのシーケンス内容を示します。

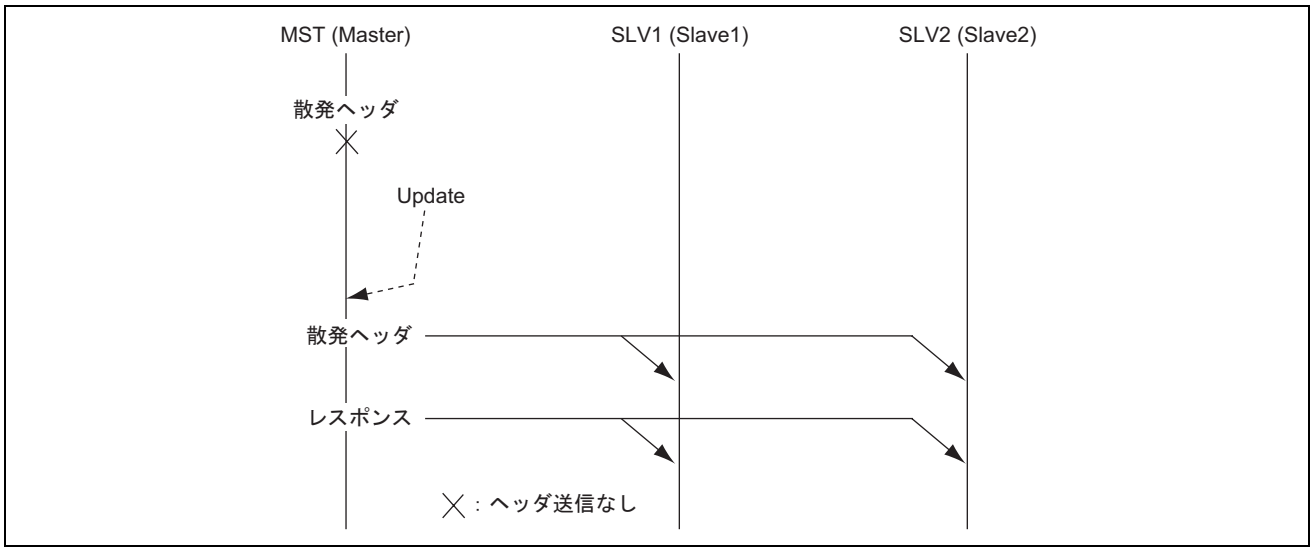

図 4　**散発フレームのシーケンス内容**

1.2.4　**マスタリクエストフレーム** (Master Request Frame)

マスタリクエストフレームは，マスタノードからスレーブノードに対して，ノード設定フレーム，ノード診断フレームの送信を行ないます。ヘッダに対するレスポンスは必ずマスタノードのみからとなります。

図 5 にマスタリクエストフレームのシーケンス内容を示します。

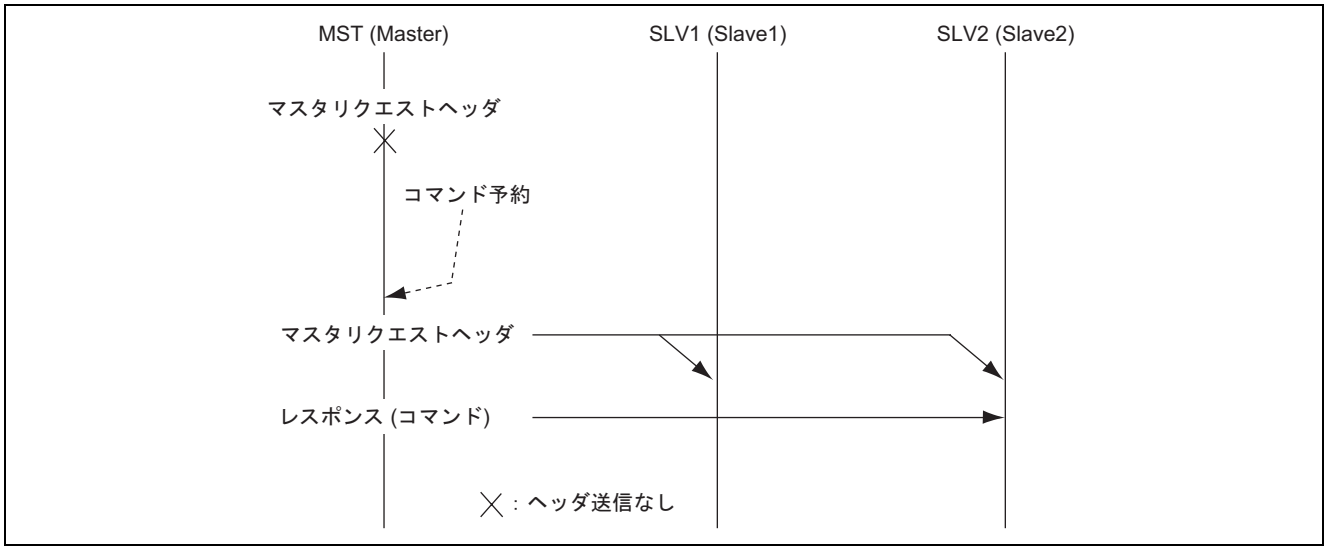

図 5　マスタリクエストフレームのシーケンス内容

1.2.5　**スレーブレスポンスフレーム** (Slave Response Frame)

スレーブレスポンスフレームは，マスタノードからスレーブノードに対して，ノード設定フレームに対するレスポンス，ノード診断フレーム有無の確認/受信を行ないます。ヘッダに対するレスポンスは必ずスレーブノードのみからとなります。このとき複数のスレーブノードが反応するクラスタ構成であってはなりません。また，スレーブノードは応答するレスポンスがない場合には，レスポンスの送信を行ないません。

図 6 にスレーブレスポンスフレームのシーケンス内容を示します。

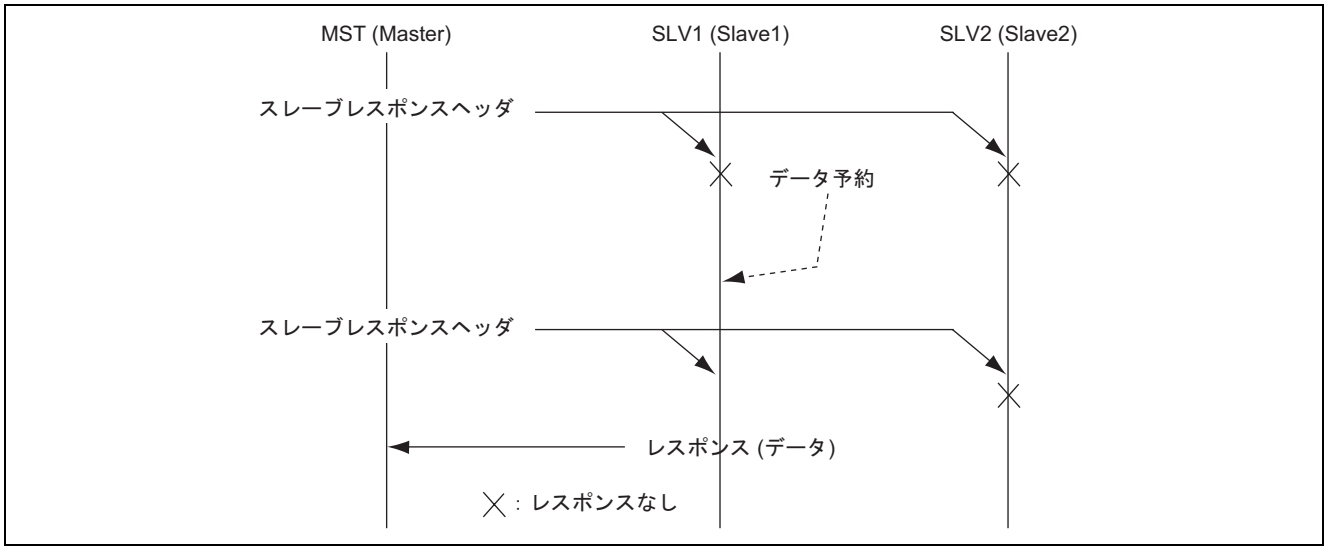

図 6　**スレーブレスポンスフレームのシーケンス内容**

2. LIN2.0 ライブラリ仕様

ライブラリをユーザアプリケーションプログラムに組み込むことにより H8/36049F の内蔵機能を使用し，マスタノードとして LIN 通信を行ないます。

2.1 ファイル構成

- **36049s.h (Ver.1.00)**
  H8/36049 グループ用内蔵 I/O レジスタ定義ファイルです。

- **sci_drv36049.c (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) SCI3 機能設定, および通信制御を行なう SCI3 ドライバの C ソースファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **sci_drv36049.h (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) SCI3 機能設定，および通信制御を行なう SCI3 ドライバのヘッダファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **tmr_drv36049.c (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) タイマ B 機能設定, カウント制御を行なうタイマ B ドライバの C ソースファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **tmr_drv36049.h (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) タイマ B 機能設定, カウント制御を行なうタイマ B ドライバのヘッダファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **Lin_Drv36049.c (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) 機能設定, 通信制御を行なう LIN ドライバの C ソースファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **Lin_Drv36049.h (Ver.1.00)**
  H8/36049F における LIN 通信用 (マスタ) 機能設定, 通信制御を行なう LIN ドライバのヘッダファイルです。このファイルはユーザが使用する CPU 環境により自由に変更/改造することができます。このファイルは LIN2.0 ライブラリとは切り離されているためユーザアプリケーションプログラムコンパイル時に組み込む必要があります。

- **Lin_Master_Cnf.c(Ver.1.00)**
  マスタノード専用の定義ファイルです。クラスタ内で扱われる信号，フレーム，スケジュールなどの定義がされています。このファイルはユーザが構築するクラスタ環境により生成しますが，通常はコンフィグレータを使用して生成します。

- **Lin_Com_Cnf.h (Ver.1.00)**
  マスタノード定義ファイル (Lin_Master_Cnf.c) にインクルードするヘッダファイルです。

- **lin20.h (Ver.1.00)**
  LIN2.0 ライブラリのヘッダファイルです。このヘッダファイルをユーザアプリケーションプログラムへインクルードする必要があります。

- **lin20.lib (Ver.1.00)**
  LIN2.0 ライブラリ本体です。このファイルをユーザアプリケーションプログラムとリンクする必要があります。

2.2　ROM/RAM 使用容量

(H8S, H8/300 SERIES C/C++ COMPILER (V.6.00.03.000) 使用)

ROM/RAM 使用容量は，定義内容により変化するため，ここでは LIN2.0 ライブラリ (lin20.lib) の使用量のみ示します。

- **ROM 容量**: 13356 Byte
- **RAM 容量**: 234 Byte*

【注】 ヒープ領域は含みません。「2.2.1　**ヒープ領域」参照**

2.2.1　ヒープ領域

LIN2.0 ライブラリでは初期化時にヒープ領域から動的にバッファを確保します。そのためアプリケーション開発時にはヒープ領域を用意して頂く必要があります。以下では最低限必要とするヒープ領域を示します。また，ヒープ領域を消費する項目を示します。

1. 最低限必要なヒープ (RAM) 領域
   — インタフェース管理用 RAM バッファ　　: 20 byte
   — 未処理診断フレーム送信用 FIFO バッファ: 9 byte (1 段確保時)
   — 未処理診断フレーム受信用 FIFO バッファ: 9 byte (1 段確保時)
   上記により，最低 38 byte のヒープ領域が必要です。

2. ヒープ (RAM) 領域を消費する項目
   — 未処理診断フレーム送信用 FIFO バッファ
   — 未処理診断フレーム受信用 FIFO バッファ
   上記はユーザがコンフィグレータで FIFO 段数を指定することができます。指定できる段数は送信用，受信用共に 1～65535 段です。

   消費するヒープ領域の計算方法は以下となります。

   計算式: 送信用 (または受信用) FIFO 段数 ×9 byte (FIFO1 段消費数)

   例: 送信用 FIFO バッファ 30 段，受信用 FIFO バッファ 20 段確保時

   【30 (段) × 9 (byte) 】+【20 (段) × 9 (byte) 】= 450 (byte)

【注】 ヒープ領域が確保できない場合，LIN **システム初期化時にエラーとなります。**

2.3　API 関数

LIN2.0 ライブラリでスレーブノードが使用可能な API 関数について説明します。

図 7 に API 関数の説明形式を示します。

機能の概要を示します。

ライブラリ関数の型 (リターン値および引数) を示します。

説　明	ライブラリ関数の機能を説明します。
リターン値	正常：ライブラリ関数が正常終了したときの値です。 異常：ライブラリ関数が異常終了したときの値です。
引　数	引数の意味を説明します。
例	呼び出し手順を説明します。
備　考	補足説明, または使用上の注意事項です。

図 7　API 関数の説明形式

2.3.1　API 一覧

以下の表 2 にスレーブノードが使用可能な API 関数 (全 36 関数) の一覧を示します。

表 2　API 関数一覧

API 関数名	用　途
l_sys_init	LIN システム初期化
l_ifc_init	インタフェース初期化
l_ifc_ioctl	I/O ドライバ登録
l_ifc_connect	LIN バス接続
l_ifc_disconnect	LIN バス切断
l_sch_set	スケジュール設定
l_sch_tick	スケジュール実行
l_flg_tst	フラグテスト
l_flg_clr	フラグクリア
l_bool_rd	1 bit 信号読み出し
l_u8_rd	2～8 bit 信号読み出し
l_u16_rd	9～16 bit 信号読み出し
l_bytes_rd	バイト配列信号読み出し
l_bool_wr	1 bit 信号書き込み
l_u8_wr	2～8 bit 信号書き込み
l_u16_wr	9～16 bit 信号書き込み
l_bytes_wr	バイト配列信号書き込み
l_ifc_goto_sleep	スリープコマンド予約
l_ifc_wake_up	ウェイクアップ信号出力
l_ifc_tx	1 フレーム送信
l_ifc_rx	1 フレーム受信
l_ifc_read_status	ステータス取得
ld_is_ready	ノード設定状態確認
ld_check_response	レスポンス状態取得
ld_assign_frame_id	フレーム ID アサイン
ld_read_by_id	ノードプロパティ読み出し
ld_assign_NAD	NAD 値アサイン
ld_conditional_change_NAD	条件付き NAD 値アサイン
ld_put_raw	未処理診断フレーム送信
ld_get_raw	未処理診断フレーム取得
ld_raw_tx_status	未処理診断送信状態取得
ld_raw_rx_status	未処理診断受信状態取得
ld_send_message	処理済診断フレーム送信
ld_receive_message	処理済診断フレーム受信
ld_tx_status	処理済診断送信状態取得
ld_rx_status	処理済診断受信状態取得

2.3.2　コア API

システム初期化

l_bool　l_sys_init(void)	
説　明	LIN システムの初期化をします。
リターン値	正常に初期化できたとき: 0 初期化に失敗したとき: 1
引　数	なし
例	l_bool　ret ret = l_sys_init();
備　考	これ以降，説明する API 関数をコールするよりも先に当 API 関数をコールしてください。当 API はリセット時に一度だけコールします。

インタフェース初期化

void l_ifc_init(l_u8　ifc_name)	
説　明	LIN インタフェースの初期化をします。
リターン値	なし
引　数	ifc_name　　インタフェース名
例	ifc_init(0);
備　考	「l_sys_init」API 関数と「l_ifc_ioctl」API 関数コール後に，当 API 関数をコールしてください。 コール前に上記以外の API 関数をコールした場合の動作は不定です。 インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。

l_u16　l_ifc_ioctl(l_u8　ifc_name, l_ioctl_op op, void* hand)

説　明　　**各ノードで使用する** I/O **ドライバを登録します。**

リターン値　　**全てのドライバが登録されたとき**: 0
一部のドライバが登録されなかったとき: **登録されなかった数**

引　数

ifc_name		**インタフェース名**
op		**オペレーションコード**
hand		**登録するドライバのハンドラ　ポインタ**

例

```
const T_Lib_Slave_Handle Slave_handle = {
        Lin_Drv_Init,
        Lin_Drv_BreakOut,
        Lin_Drv_BreakFinish,
        Lin_Drv_SendSync,
        Lin_Drv_SendPid,
        Lin_Drv_SendPidFinish,
        Lin_Drv_First_SendData,
        Lin_Drv_SendData,
        Lin_Drv_First_RecvReq,
        Lin_Drv_RecvData,
        Lin_Drv_SendRecvFinish,
        Lin_Drv_LinBus_Enable,
        Lin_Drv_LinBus_Disable,
        Lin_Drv_WakeUp,
        Lin_Drv_WakeUpFinish
};
l_u16    ret;
ret = l_ifc_ioctl( 0, LIN_ENTRY_SLAVE _DRV, &Slave_handle );
```

備　考　　**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**
オペレーションコードは以下の二つから指定してください。
マスタノードドライバ登録: LIN_ENTRY_MASTER_DRV
スレーブノードドライバ登録: LIN_ENTRY_SLAVE_DRV
当 API **は「**l_ifc_init**」**API **関数をコールする前に呼び出してください。**

LIN バス接続

l_bool l_ifc_connect(l_u8 ifc_name)

説　明	LIN **バスと接続します。**
リターン値	**接続に成功したとき**: 0
	接続に失敗したとき: 1
引　数	ifc_name　　**インタフェース名**
例	l_bool　ret;
	ret = l_ifc_connect(0);
	if(ret) {
	/* Lin Bus Connect Failed */
	}
備　考	LIN **通信を行なう場合には当** API **関数で接続してからスケジュール実行をしてください。**
	インタフェース名は 0 **しか設定できません。**0 **でない値は設定しないでください。**

LIN バス切断

l_bool l_ifc_disconnect(l_u8　ifc_name)

説　明	LIN **バスを切断します。**
リターン値	**切断に成功したとき**: 0
	切断に失敗したとき: 1
引　数	ifc_name　　**インタフェース名**
例	l_bool　ret;
	ret = l_ifc_disconnect(0);
	if(ret) {
	/* Lin Bus Disconnect Failed */
	}
備　考	LIN **通信を終了する場合には当** API **関数で切断してください。**
	インタフェース名は 0 **しか設定できません。**0 **でない値は設定しないでください。**

スケジュール設定

void l_sch_set(l_u8　ifc_name, l_schedule_handle schedule, l_u8　entry)

説　明	**次回のスケジュール実行タイミングで実行するスケジュールを設定します。**
リターン値	**なし**
引　数	ifc_name　**インタフェース名** schedule　**スケジュール名** entry　**エントリ番号**
例	l_sch_set (0, & Lin_Sch_Schedule1, 3);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **スケジュール名は** LIN **コンフィグレータで定義したスケジュール名です。**LIN **コンフィグレータではユーザが定義したスケジュールの実体を生成します。実体は出力されたファイル内に存在します。ここでは定義したスケジュールの実体に【**&**】をつけて，実体のアドレスを設定してください。**NULL **など不定なアドレスは設定しないようにしてください。エントリ番号は，設定したスケジュールの何番目から実行を行なうかを設定します。設定したスケジュール内で指定できる番号を設定してください。不定な値を設定した場合の動作は保証できません。また，**0 **を指定した場合は，**1 **を指定した場合と同様にスケジュールの先頭から実行します。詳細は「**3.　**参考文献」を参照してください。**

スケジュール実行

l_u8　l_sch_tick(l_u8　ifc_name)

説　明	1 **スロット分のスケジュールを実行します。**
リターン値	**スケジュール実行を行なったとき**: **次回実行予定のエントリ番号** 1 **スロット消化のみでフレーム送信しなかったとき**: 0
引　数	ifc_name　**インタフェース名**

例	l_u8　entry;
	entry = l_sch_tick (0);
	if(entry) {
	/* Something is done by using "Entry". */
	}

備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **当** API **は** LIN **コンフィグレータで設定したタイムベース間隔でコールする必要があります。** **この間隔を守れなかった場合の動作は保証できません。詳細は「**3.　**参考文献」を参照してください。**

フラグのテスト

l_bool　l_flg_tst(l_flag_handle flag_name)

説　明	**フラグのテストを行ないます。**
リターン値	**フラグの値**: 0 or 1
引　数	flag_name　　**フラグ名**

例	l_bool　ret;
	ret = l_flg_tst(&Lin_Frm_FrameU1_flg);
	if(ret) {
	　　/* Something is done. */
	}
	else {
	　　/* Something is done. */
	}

備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **フラグ名はユーザが定義したフラグの名前です。** **定義したフラグのアドレスを代入します。**

フラグのクリア

void l_flg_clr(l_flag_handle flag_name)

説　明	**フラグのクリアを行ないます。**
リターン値	**なし**
引　数	flag_name　　**フラグ名**
例	l_flg_clr (&Lin_Frm_FrameU1_flg);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **フラグ名はユーザが定義したフラグの名前です。** **定義したフラグのアドレスを代入します。**

信号値読み出し

l_bool l_bool_rd(l_signal_handle sig_name)

説　明	1 **ビット信号の読み出しを行ないます。**
リターン値	**信号値**: 0 or 1
引　数	sig_name　　**信号名**
例	l_bool value; value = l_bool_rd (& Lin_Sig_Test0);
備　考	**信号名はユーザが定義した信号の名前です。** **定義した信号のアドレスを代入します。** 1 bit **信号でない信号は当** API **で読み出さないでください。** **読み出した場合の動作は保証外です。**

信号値読み出し

l_u8 l_u8_rd(l_signal_handle sig_name)

説　明	2～8 **ビット信号の読み出しを行ないます。**
リターン値	**信号値**: 0～255
引　数	sig_name　　**信号名**
例	l_u8 value; value = l_u8_rd (&Lin_Sig_Test3);
備　考	**信号名はユーザが定義した信号の名前です。** **定義した信号のアドレスを代入します。** 2～8 **ビットの信号でない信号は当** API **で読み出さないでください。** **読み出した場合の動作は保証外です。**

信号値読み出し

l_u16 l_u16_rd(l_signal_handle sig_name)	
説　明	9～16 **ビット信号の読み出しを行ないます。**
リターン値	**信号値**: 0～65535
引　数	sig_name　　**信号名**
例	l_u16 value; value = l_u16_rd (&Lin_Sig_Test7);
備　考	**信号名はユーザが定義した信号の名前です。** **定義した信号のアドレスを代入します。** 9～16 **ビットの信号でない信号は当** API **で読み出さないでください。** **読み出した場合の動作は保証外です。**

信号値読み出し

void l_ifc_init(l_u8	**ifc_name)**
説　明	**バイト配列信号の読み出しを行ないます。**
リターン値	**なし**
引　数	sig_name　　**信号名** start　　**読み出し開始バイト位置** count　　**読み出しバイト数** data　　**信号値格納用バッファ**: 1～8 byte
例	l_u8　data[8]; l_bytes_rd (& Lin_Sig_Test13, 1, 2);
備　考	**信号名はユーザが定義した信号の名前です。** **定義した信号のアドレスを代入します。** **バイト配列信号でない信号は当** API **で読み出さないでください。** **読み出した場合の動作は保証外です。** **また，読み出しバイト数が定義した信号のサイズを超えないようにしてください。** **エラー発生時には読み出しは行ないません。このときのバッファの中身は不定となります。**

信号値書き込み

void l_bool_wr(l_signal_handle sig_name, l_bool sig)

説　明	1 ビット信号の書き込みを行ないます。	
リターン値	なし	
引　数	sig_name	**信号名**
	sig	**信号値**: 0 or 1
例	l_bytes_wr(& Lin_Sig_Test1, 1);	
備　考	信号名はユーザが定義した信号の名前です。	
	定義した信号のアドレスを代入します。	
	1 bit 信号でない信号は当 API で書き込まないでください。	
	書き込んだ場合の動作は保証外です。	

信号値書き込み

void l_u8_wr(l_signal_handle sig_name, l_u8 sig)

説　明	2〜8 ビット信号の書き込みを行ないます。	
リターン値	なし	
引　数	sig_name	**信号名**
	sig	**信号値**: 0〜255
例	l_u8_wr(&Lin_Sig_Test4, 123);	
備　考	信号名はユーザが定義した信号の名前です。	
	定義した信号のアドレスを代入します。	
	2〜8 ビットの信号でない信号は当 API で書き込まないでください。	
	書き込んだ場合の動作は保証外です。	

信号値書き込み

void l_u16_wr(l_signal_handle sig_name, l_u16 sig)

説　明	9〜16 **ビット信号の書き込みを行ないます。**	
リターン値	**なし**	
引　数	sig_name	**信号名**
	sig	**信号値**: 0〜65535
例	l_u16_wr(&Lin_Sig_Test8, 12345);	
備　考	**信号名はユーザが定義した信号の名前です。**	
	定義した信号のアドレスを代入します。	
	9〜16 **ビットの信号でない信号は当** API **で書き込まないでください。**	
	書き込んだ場合の動作は保証外です。	

信号値書き込み

void l_bytes_wr(l_signal_handle sig_name, l_u8 start, l_u8 count, const l_u8* const data)

説　明	**バイト配列信号の書き込みを行ないます。**	
リターン値	**なし**	
引　数	sig_name	**信号名**
	start	**書き込み開始バイト位置**
	count	**書き込みバイト数**
	data	**信号値格納用バッファ**: 1〜8 byte
例	l_u8　data[8] = { 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE, 0xF0 };	
	l_bytes_wr(& Lin_Sig_Test15, 0, 8);	
備　考	**信号名はユーザが定義した信号の名前です。**	
	定義した信号のアドレスを代入します。	
	バイト配列信号でない信号は当 API **で書き込まないでください。**	
	書き込んだ場合の動作は保証外です。	
	また，書き込みバイト数が定義した信号のサイズを超えないようにしてください。	
	エラー発生時には書き込みは行ないません。	

スリープコマンド

void l_ifc_goto_sleep(l_u8 ifc_name)

説　明	**スリープコマンドの予約を行ないます。**
リターン値	**なし**
引　数	ifc_name　　**インタフェース名**
例	l_ifc_goto_sleep(0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **当** API **コール時にはスリープコマンドは送信しません。次回のマスタリクエストフレーム送信時に送信します。また，他のフレームが予約されていた場合でも，優先的にスリープコマンドを送信します。このとき既に予約された他のフレームは次回のマスタリクエストフレーム送信時に送信します。当** API **を** (**予約後，スリープコマンド送信前に**) **連続コールした場合，**2 **回目以降のコールは無視されます。**

ウェイクアップ信号

void l_ifc_wake_up(l_u8 ifc_name)

説　明	**ウェイクアップ信号の出力を行ないます。**
リターン値	**なし**
引　数	ifc_name　　**インタフェース名**
例	l_ifc_wake_up(0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **当** API **コール時にウェイクアップ信号を出力します。**

フレーム送信

void l_ifc_tx(l_u8 ifc_name

説　明　**フレームの送信を行ないます。**
リターン値　**なし**
引　数　ifc_name　**インタフェース名**
例

```
vodi tx_isr( void )
{
        l_ifc_tx( 0 );
}
```

備　考　**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**
通常，当 API **はシリアルの送信割り込み関数内でコールされます。**
コールされる場所は H/W **構成に依存します。**

フレーム受信

void l_ifc_rx(l_u8 ifc_name)

説　明　**フレームの受信を行ないます。**
リターン値　**なし**
引　数　ifc_name　**インタフェース名**
例

```
vodi rx_isr( void )
{
        l_ifc_rx( 0 );
}
```

備　考　**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**
通常，当 API **はシリアルの受信割り込み関数内でコールされます。**
コールされる場所は H/W **構成に依存します。**

ステータス取得

l_u16 l_ifc_read_status(l_u8 ifc_name)

説　明	**ステータス値の取得を行ないます。**
リターン値	**ステータス値**: **「**3.　**参考文献」参照**
引　数	ifc_name　　**インタフェース名**
例	l_u16　status; status = l_ifc_read_status(0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**

2.3.3　ノード設定 API

ノード設定確認

l_bool ld_is_ready(l_u8 ifc_name)

説　明	ノード設定の状況を確認します。
リターン値	ノード設定 API コール OK: 1 ノード設定 API コール NG: 0
引　数	ifc_name　インタフェース名
例	lif(ld_is_ready (0)) { }
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。 マスタリクエストフレームの後にスレーブレスポンスフレームがスケジュールに登録されている場合には，スレーブレスポンスフレームを実行後に設定されます。マスタリクエストフレーム送信後にスケジュールを変更し，スレーブレスポンスフレームが送信されなかった場合にはスレーブレスポンスフレームが送信されるまで設定されません。マスタリクエストフレームとスレーブレスポンスフレームは連続して (推奨) 実行してください。

レスポンス確認

l_u8 ld_check_response(l_u8 ifc_name, l_u8* rsid, l_u8* error_code)

説　明	レスポンスの状況を確認します。
リターン値	レスポンスの状況: 「3.　参考文献」参照
引　数	ifc_name　インタフェース名 rsid　レスポンス ID 格納用バッファ error_code　エラーコード格納用バッファ
例	l_u8　rtn, rsid, error_code; rtn = ld_check_response (0, &rsid, &error_code);
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。

フレーム ID アサイン

void ld_assign_frame_id(l_u8 ifc_name, l_u8 nad, l_u16 supplier_id, l_u16 message_id, l_u8 pid)

説　明	**フレーム** ID **をアサインするコマンドを予約します。**	
リターン値	**なし**	
引　数	ifc_name	**インタフェース名**
	nad	**アサインするノードの** NAD **値**
	supplier_id	**アサインするノードのサプライヤ** ID
	message_id	**アサインするノードのメッセージ** ID
	pid	**アサインするフレーム** ID **のプロテクト** ID
例	ld_assign_frame_id (0, 0x23u, 0x1234u, 0x4567u, 0x61u);	
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **当** API **コール時には送信しません。次回のマスタリクエストフレーム送信時に送信します。このときスリープコマンドが予約されているときには，スリープコマンドを優先します。戻り値がないためエラーチェックは行なっていません。エラーチェックは当** API **コール側で行なってください。**	

ノードプロパティ読み込み

void ld_read_by_id(l_u8 ifc_name, l_u8 nad, l_u16 supplier_id, l_u16 function_id, l_u8 id, l_u8* const data)

説　明	**プロパティを読み込むコマンドを予約します**	
リターン値	**なし**	
引　数	ifc_name	**インタフェース名**
	nad	**読み込むノードの** NAD **値**
	supplier_id	**読み込むノードのサプライヤ** ID
	function_id	**読み込むノードの機能** ID
	data	**読み込んだデータ格納用バッファ**
例	l_u8　data[8]; ld_read_by_id (0, 0x23, 0x1234, 0x6789, 1, data);	
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **当** API **コール時には送信しません。次回のマスタリクエストフレーム送信時に送信します。このときスリープコマンドが予約されているときには，スリープコマンドを優先します。データ格納用バッファのバッファサイズは必ず** 8 byte **としてください。戻り値がないためエラーチェックは行なっていません。エラーチェックは当** API **コール側で行なってください。**	

NAD 値アサイン

void ld_assign_NAD(l_u8 ifc_name, l_u8 nad, l_u16 supplier_id, l_u16 function_id, l_u8 new_NAD)

説　明	NAD 値をアサインするコマンドを予約します。	
リターン値	なし	
引　数	ifc_name	インタフェース名
	nad	アサインするノードの NAD 値
	supplier_id	アサインするノードのサプライヤ ID
	function_id	アサインするノードの機能 ID
	new_NAD	アサインする新しい NAD 値
例	ld_assign_NAD (0, 0x23, 0x1234, 0x5678, 0x15);	
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。 当 API コール時には送信しません。次回のマスタリクエストフレーム送信時に送信します。このときスリープコマンドが予約されているときには，スリープコマンドを優先します。戻り値がないためエラーチェックは行なっていません。エラーチェックは当 API コール側で行なってください。	

条件付き NAD 値変更

void ld_conditional_change_NAD(l_u8 ifc_name, l_u8 nad, l_u8 id, l_u8 byte, l_u8 mask, l_u8 invert, l_u8 new_NAD)

説　明	条件成立時に NAD 値をアサインするコマンドを予約します	
リターン値	なし	
引　数	ifc_name	インタフェース名
	nad	アサインするノードの NAD 値
	id	アサインするノードのプロパティ値
	byte	アサインするノードの読み込むプロパティ値のバイト位置
	mask	読み込んだプロパティ値をマスクする値
	invert	読み込んだプロパティ値を排他する値
	new_NAD	条件成立時にアサインする新しい NAD 値
例	ld_conditional_change_NAD (0, 0x23, 1, 2, 0x55, 0xAA, 0x15);	
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。 当 API コール時には送信しません。次回のマスタリクエストフレーム送信時に送信します。このときスリープコマンドが予約されているときには，スリープコマンドを優先します。戻り値がないためエラーチェックは行なっていません。エラーチェックは当 API コール側で行なってください。	

2.3.4　診断フレーム用 API

未処理診断フレーム送信予約

void ld_put_raw(l_u8 ifc_name, const l_u8* const data)

説　明	未処理診断フレームを送信 FIFO バッファへ予約します。	
リターン値	なし	
引　数	ifc_name	インタフェース名
	data	送信データバッファ
例	l_u8　data[8] = { 0x20u, 0x06u, 0xb1u, 0xffu, 0x7fu, 0x00u, 0x00u, 0x20u }; ld_put_raw (0, data);	
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。 当 API コール時には送信しません。次回のマスタリクエストフレーム送信時に送信します。このときスリープコマンドまたはノード設定コマンドが予約されているときには，スリープコマンド，ノード設定コマンドを優先します。FIFO バッファに空きがない場合にはコールしても予約しません。戻り値がないためエラーチェックは行なっていません。エラーチェックは当 API コール側で行なってください。	

未処理診断フレームデータ取得

void ld_get_raw(l_u8 ifc_name, l_u8* const data)

説　明	未処理診断フレームのデータを FIFO バッファから取得します。	
リターン値	なし	
引　数	ifc_name	インタフェース名
	data	取得データ格納用バッファ
例	l_u8　data[8]; ld_get_raw (0, data);	
備　考	インタフェース名は 0 しか設定できません。0 でない値は設定しないでください。 当 API コール時に FIFO バッファの古いものから順に 1 フレーム分取得します。FIFO バッファが空の場合にはコールしても取得しません。戻り値がないためエラーチェックは行なっていません。エラーチェックは当 API コール側で行なってください。	

未処理診断フレーム送信確認

l_u8 ld_raw_tx_status(l_u8 ifc_name)

説　明	**未処理診断フレームの送信** FIFO **バッファの状況を確認します。**
リターン値	FIFO **バッファに空きがない場合**: LD_QUEUE_FULL FIFO **バッファが空の場合**: LD_QUEUE_EMPTY **転送エラー発生時**: LD_TRANSFER_ERROR
引　数	ifc_name　　**インタフェース名**
例	l_u8　rtn; rtn = ld_raw_tx_status (0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**

未処理診断フレーム受信確認

l_u8 ld_raw_rx_status(l_u8 ifc_name)

説　明	**未処理診断フレームの受信** FIFO **バッファの状況を確認します。**
リターン値	FIFO **バッファにデータ有り**: LD_DATA_AVAILABLE **転送エラー発生時**: LD_TRANSFER_ERROR
引　数	ifc_name　　**インタフェース名**
例	l_u8　rtn; rtn = ld_raw_rx_status (0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**

処理済診断フレーム送信

void ld_send_message(l_u8 ifc_name, l_u16 length, l_u8 NAD, const l_u8* const data)

説　明	**処理済み診断フレームの送信を予約します。**	
リターン値	**なし**	
引　数	ifc_name	**インタフェース名**
	length	**送信データ長**
	NAD	**送信先ノードの** NAD **値**
	data	**送信データバッファ**
例	l_u8　data[5] = { 0x12, 0x34, 0x56, 0x78, 0x9A }; ld_send_message (0, 5, 0x23, data);	
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **戻り値がないためエラーチェックは行なっていません。エラーチェックは当** API **コール側で行なってください。当** API **を送信完了前に再度コールした場合の動作は保証できません。**	

処理済診断フレーム受信

void ld_receive_message(l_u8 ifc_name, l_u16* length, l_u8* NAD, l_u8* const data

説　明	**処理済み診断フレームの受信を予約します。**	
リターン値	**なし**	
引　数	ifc_name	**インタフェース名**
	length	**受信データ長格納用バッファ**
	NAD	**送信元ノードの** NAD **値**
	data	**受信データ格納用バッファ名**
例	l_u8　data[100], nad; l_u16　length = 100; ld_receive_message (0, &length, &nad, data);	
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。** **受信データ長は予約時に許容受信データ長を格納してから設定します。** **戻り値がないためエラーチェックは行なっていません。エラーチェックは当** API **コール側で行なってください。当** API **を受信完了前に再度コールした場合の動作は保証できません。**	

処理済診断フレーム送信確認

l_u8 ld_tx_status(l_u8 ifc_name)

説　明	**処理済み診断フレームの送信状況を確認します。**
リターン値	**送信完了**: LD_COMPLETED
	送信中: LD_IN_PROGRESS
	送信エラー発生: LD_FAILED
引　数	ifc_name　　**インタフェース名**
例	l_u8　rtn;
	rtn = ld_tx_status(0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**

処理済診断フレーム受信確認

l_u8 ld_rx_status(l_u8 ifc_name)

説　明	**処理済み診断フレームの受信状況を確認します。**
リターン値	**受信完了**: LD_COMPLETED
	受信中: LD_IN_PROGRESS
	受信エラー発生: LD_FAILED
引　数	ifc_name　　**インタフェース名**
例	l_u8　rtn;
	rtn = ld_rx_status(0);
備　考	**インタフェース名は** 0 **しか設定できません。**0 **でない値は設定しないでください。**

2.4　LIN2.0 ライブラリ API 関数の使い方

以下では LIN2.0 ライブラリ API 関数の使用例を示します。

2.4.1　LIN システムの初期化

LIN2.0 ライブラリ API 関数を使用する前に LIN システムの初期化を行ないます。

以下の例では，マイコンリセット時に LIN システムの初期化をしています。

【注】 LIN の API コール箇所を反映します。

```
extern unsigned char lin_SomeCotrol_init( void );
__entry(vect=0) void PowerON_Reset(void)
{
    set_imask_ccr(1);

   _INITSCT();
// _CALL_INIT();            // Remove the comment to use global class object.
// _INIT_IOLIB();           // Remove the comment mark to use SIM I/O.
// errno=0;                 // Remove the comment mark to use errno.
// srand(1);                // Remove the comment mark to use rand().
// _s1ptr=NULL;             // Remove the comment mark to use strtok().
   HardwareSetup();    // Remove the comment mark to use Hardware Setup.

   set_imask_ccr(0);

   /* ......Something to do */

   if( l_sys_init() ) {
      /* LIN System Initialize failed */
      sleep();
   }
   else {
      if( lin_SomeCotrol_init() ) {
         /* SomeSensor Initialize failed */
         sleep();
      }
   }

   /* ......Something to do */

   main();

// _CLOSEALL();             // Remove the comment mark to use SIM I/O.
// _CALL_END();             // Remove the comment mark to use global class object.
   sleep();
}
```

```
/* マスタドライバエントリ用定義 */
const T_Lib_Master_Handle Master_handle = {
   Lin_Drv_Init,
   Lin_Drv_BreakOut,
   Lin_Drv_BreakFinish,
   Lin_Drv_SendSync,
   Lin_Drv_SendPid,
   Lin_Drv_SendPidFinish,
   Lin_Drv_First_SendData,
   Lin_Drv_SendData,
   Lin_Drv_First_RecvReq,
   Lin_Drv_RecvData,
   Lin_Drv_SendRecvFinish,
   Lin_Drv_LinBus_Enable,
   Lin_Drv_LinBus_Disable,
   Lin_Drv_WakeUp,
   Lin_Drv_WakeUpFinish
};

/* Cluster Initialize */
extern T_Schedule Lin_Sch_Schedule1;     /* ユーザが定義したスケジュール */
unsigned char lin_SomeCotrol_init( void )
{
   unsigned char rtn;

   rtn = 0;
   if( l_ifc_ioctl( 0, LIN_ENTRY_MASTER_DRV, &Master_handle ) ) {
      /* The init of the LIN Master Driver failed */
      rtn = 1u;
   }
   else {
      l_ifc_init(0);   /* Interface Initialize */
      if( l_ifc_connect(0) ) {
         /* Connection of the LIN interface failed */
         rtn = 1u;
      }
      else {
         /* Schedule Setting */
         l_sch_set( 0, &Lin_Sch_Schedule1, 0 );
         lin_schedule_start();
      }
   }
   return rtn;
}

void lin_schedule_start( void )
{
   MSTCR1.BIT.MSTTW = 0;           /* モジュールスタンバイ解除 */
   TW.TCRW.BIT.CCLR = 0;           /* フリーラン */
   TW.TCRW.BIT.CKS = 3;            /* 8/ */
   TW.TIERW.BIT.OVIE = 1u;
   TW.TCNT = (0xFFFFu-2500u);
   TW.TMRW.BIT.CTS = 1u;           /* start Count Up */
}
```

2.4.2　スケジュール実行

LIN システムではスケジュール実行 API 関数を定期的にコールする必要があります。

以下の例では，タイマ W にて，1 ms カウントアップ機能を作成し，スケジュール実行用関数 (メイン処理) 内で，スケジュール実行タイムベース間隔でスケジュール実行 API 関数をコールしています。ここでは，タイムベースを 500 ms に定義しています。

【注】 LIN の API コール箇所を反映します。

```
static unsigned short tw_counter = 0;
/************************************/
/* タイマ W 1ms 割り込み関数     */
/************************************/
__interrupt(vect=21)
void tw_isr_1ms( void )
{
   UB dummy;

   TW.TCNT = (0xFFFFu-2500u);
   dummy = TW.TSRW.BYTE;
   TW.TSRW.BYTE = 0;

   /* Something to do */

   tw_counter++;

   /* Something to do */

   return;
}

/************************************/
/* 1ms カウンタ取得関数        */
/************************************/
unsigned short lin_get_tw_counter( void )
{
   unsigned short c;

   /* 1ms timer interrupt disable */
   c = tw_counter;
   /* 1ms timer interrupt enable */
   return c;
}

/************************************/
/* 1ms カウンタクリア関数      */
/************************************/
void lin_clr_tw_counter( void )
{
   /* timer w interrupt disable */
   TW.TIERW.BIT.OVIE = 0;

   tw_counter = 0;

   /* timer w interrupt enable */
   TW.TIERW.BIT.OVIE = 1u;

   return;
}
```

```
/**********************************/
/* LIN スケジュール関数           */
/**********************************/
void lin_schedule_exe( void )
{
   l_u8 entryno;

   if( LIN_TIME_BASE <= lin_get_tw_counter() ) {
      lin_clr_tw_counter();
      /* ......Something to do */

      entryno = l_sch_tick( 0 );

      /* ......Something to do */
   }

   return;
}
```

2.4.3　**アプリケーション**

ここでは初期化, スケジュール実行以外にアプリケーションからコールされる LIN2.0 ライブラリ API 関数についてのサンプルコードを示します。API 関数コールによって取得したデータの使用方法はアプリケーションに依存するためここでは特に記載しません。LIN バス上で送受信されるデータ (フレーム) の内容は各ノードの状態や周辺デバイスなどから取得したデータです。どのようなデータを送受信するか，データをどのように処理するかは LIN のシステム構成によります。

```
#include "36014s.h"
#include "Lin_Drv36014.h"
#include "lin20.h"
void lin_application( void );
/*************************/
/* メイン関数            */
/*************************/
void main(void)
{
   while( 1 ) {
      /* ......Something to do */

      lin_application();

      /* ......Something to do */
   }
}
```

```
/***********************************/
/* LIN アプリケーション関数        */
/***********************************/
extern l_flg   Lin_Sig_Status_Slv1_flg;        /* ユーザが定義したフラグ */
extern T_Signal   Lin_Sig_Status_Slv0;         /* ユーザが定義した信号 */
extern T_Signal   Lin_Sig_Command;             /* ユーザが定義した信号 */
extern T_Schedule Lin_Sch_Schedule2;           /* ユーザが定義したスケジュール */
void lin_application( void )
{
   l_u16 signal;
   l_u8  rsid, error_code, ret_res;
   l_u8  data[8];
   union {
      l_u16 Word;
      struct {
         l_u16 lastpid       :8;
         l_u16         :4;
         l_u16 gotosleep     :1;
         l_u16 overrun       :1;
         l_u16 txsuccese     :1;
         l_u16 errorrsp      :1;
      } Bit;
   } status;

   /* Lin Schedule Cyclic Excute */
   lin_schedule_exe();

   if( l_flg_tst( &Lin_Sig_Status_Slv1_flg ) ) {     /* フラグの状態を確認 */
      l_flg_clr( &Lin_Sig_Status_Slv1_flg );         /* フラグの状態をクリア */
            signal = l_u16_rd( &Lin_Sig_Status_Slv0 );     /* 信号値を取得 */
         l_u16_wr( &Lin_Sig_Command, signal );       /* 読み込んだ信号値を設定 */
   }

   /* Read Status */
   status.Word = l_ifc_read_status( 0 );

   if( status.Bit.errorrsp ) {
      /* Something Error Response Processing   */
   }

   if( status.Bit.lastpid == 0x34u ) {
      l_sch_set( 0, &Lin_Sch_Schedule2, 2 );   /* 実行スケジュールを再設定 */
   }
```

```
    if( ld_is_ready( 0 ) ) {
       ret_res = ld_check_response( 0, &rsid, &error_code );
       switch( ret_res ) {
       case LD_NEGATIVE:
          /* Something is done */
          ld_read_by_id( 0, 0x23u, 0x1234u, 0x4321u, 0, data );
          break;
       case LD_SUCCESS:
          /* Something is done */
          break;
       case LD_NO_RESPONSE:
          /* Something is done */
          ld_assign_frame_id( 0, 0x20u, 0x1234u, 0x5678u, 0x61u );
          break;
       case LD_OVERWRITTEN:
          /* Something is done */
          break;
       default:
          /* Something is done */
          break;
       }
    }

    switch( ld_tx_status( 0 ) ) {
    case LD_COMPLETED:
       /* Something is done */
       break;
    case LD_IN_PROGRESS:
       /* Something is done */
       break;
    case LD_FAILED:
       /* Something is done */
       break;
    default:
       /* Something is done */
       break;
    }

    if( status.Bit.gotosleep ) {
       /* Something Sleepmode Processing */
    }
 }
```

3. 参考文献

- LIN Specification Package Revision 2.0 http://www.lin-subbus.org
- LIN Protocol Specification Revision 2.0 http://www.lin-subbus.org
- LIN Diagnostic and Configuration Specification Revision 2.0 http://www.lin-subbus.org
- LIN Application Program Interface Specification Revision 2.0 http://www.lin-subbus.org
- LIN Physical Layer Specification Revision 2.0 http://www.lin-subbus.org
- H8/36049 グループ ハードウェアマニュアル RJJ09B0046-0200Z

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2006.01.31	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。