NonOS 対応 LIN 通信ミドルウェア Slave

ユーザーズマニュアル

Ver.1.00　2007 年 10 月 31 日

株式会社サニー技研

TOPPERS/NonOS 対応 LIN 通信ミドルウェア

Toyohashi Open Platform for Embedded Real-Time Systems/
NonOS 対応 LIN 通信ミドルウェア Slave パッケージ

目次

1. 概要 1
1.1 関連文書 1
2. システム要件 2
2.1 ハードウェア要件 2
2.2 ソフトウェア要件 2
2.3 開発環境 2
3. LIN スレーブモジュールの構成 3
3.1 ファイル構成 3
3.2 LIN バッファ 4
3.2.1 config 7
3.2.2 size 7
3.2.3 state 7
3.2.4 snd_flag 7
3.2.5 rcv_flag 7
3.2.6 err_flag 7
3.2.7 SndSlot(0-7) 8
3.2.8 RcvSlot(0-7) 9
3.3 LIN バッファステータス 10
3.3.1 送信データ時 10
3.3.2 受信データ時 10
3.3.3 受信データ時（オーバーライト） 10
3.3.4 送信データ時（エラー） 11
3.3.5 受信データ時（エラー） 11
4. LIN スレーブモジュールの動作 12
5. ユーザプログラムへの組み込み 13
5.1 動作環境 13
5.2 slin.h 13
5.3 slin_def.h 14
5.4 slin_sfr.h 14
5.5 slin_frm.h 15
5.6 slin_frm.c 17
5.7 割り込みベクタテーブルの設定 17
6. ＡＰＩ 18
6.1 LIN の動作モードと API について 18
6.2 初期設定 20
6.2.1 ノード ID の設定 20
6.2.2 l_sys_init() 20
6.2.3 l_ifc_init() 20
6.3 通信設定 21
6.3.1 l_ifc_connect() 21
6.3.2 l_ifc_disconnect() 21
6.3.3 l_wakeup() 22

6.3.4 l_sleep() 22
6.3.5 l_run() 23
6.4 送信 24
6.4.1 l_set_sndreq() 24
6.4.2 l_stop_sndreq() 24
6.4.3 l_read_sndsts() 25
6.4.4 l_clr_sndsts() 25
6.4.5 l_set_snddata() 26
6.5 受信 27
6.5.1 l_set_rcvreq() 27
6.5.2 l_stop_rcvreq() 27
6.5.3 l_read_rcvsts() 28
6.5.4 l_clr_rcvsts() 28
6.5.5 l_get_rcvdata() 29
7. 送信処理手順 30
7.1 送信要求 30
7.1.1 API を使用する場合 30
7.1.2 API を使用しない場合 30
7.2 送信完了確認 31
7.2.1 API を使用する場合 31
7.2.2 API を使用しない場合 31
8. 受信処理手順 32
8.1 受信要求 32
8.1.1 API を使用する場合 32
8.1.2 API を使用しない場合 32
8.2 受信完了確認 33
8.2.1 API を使用する場合 33
8.2.2 API を使用しない場合 34
9. エラー 35
9.1 エラー検出 35
10. 使用上の注意事項 36
10.1 周辺機能の使用に関する制限事項 36
10.2 端子の使用に関する制限事項 36
10.3 割り込みに関する制限事項 37
10.4 LIN トランシーバ 37

1. 概要

本ミドルウェアは、**LIN** 仕様 **1.3** に準拠した、**NonOS** 対応 **LIN** スレーブミドルウェアです。

1.1 関連文書

表 1-1 関連文書一覧表

仕様書	Version
LIN Protocol Specification	1.3

2. システム要件

2.1 ハードウェア要件

本ミドルウェアは、以下の MCU を実装ターゲットとしています。

表 2-1 ハードウェア要件一覧表

項目	内容
MCU	ルネサステクノロジ M32C/85
発振子	セラミック発振子または水晶発振子
クロック周波数	32MHz
シリアル I/O	UART0
タイマ※1	タイマ A0
割込み	UART0 受信完了、UART0 送信完了、 タイマ A0

2.2 ソフトウェア要件

本ミドルウェアは、以下の要件を固定としています。

表 2-2 ソフトウェア要件一覧表

項目	内容
通信速度	9600bps

2.3 開発環境

本ミドルウェアは、以下の C コンパイラに対応しています。

表 2-3 開発環境

コンパイラバージョン	オプション（コード生成に影響するもののみ）
ルネサステクノロジ **NC308WA Ver.5.20 Release 02**	最適化オプションなし

3．LIN スレーブモジュールの構成

3.1 ファイル構成

本ミドルウェアは、以下のファイルで構成されます。これらのファイル名は、全て固定とします。メインプログラムなどのプログラムは、ファイル名は自由です。

表 3-1 ファイル一覧表

ファイル名	内容	ユーザ編集
slin.c	本ミドルウェアのプログラム本体です。 LIN 通信にて使用する割り込みプログラム,サブルーチンが記述されています。ユーザプログラムと共に使用する必要があります。	必要なし
slin.h	本ミドルウェア用の定義ファイルです。本ミドルウェアで使用する、#define 定義、LIN バッファの共用体の定義を行っています。	必要なし
slin_sfr.h	本ミドルウェア用 SFR 設定ファイルです。	必要あり
slin_def.h	スロット数などのパラメータを設定します。システムに応じて編集する必要があります。	必要あり
slin_frm.h	LIN 通信のフレーム定義を行います。必要に応じて編集する必要があります。	必要あり
slin_frm.c	slin_frm.h で定義されているフレームを LIN バッファに割り当てます。必要に応じて編集する必要があります。	必要あり

3.2 LIN バッファ

LIN バッファは本ミドルウェアの使用するバッファ領域です。
バッファ領域は以下の内容で構成されています。詳細については以下を参考にしてください。
（なお、表 1-2 は送受信スロットとも、８スロット使用時のものとします。）

表 3-2 LIN バッファレジスタ

オフセット	レジスタ名	ビット	内容	初期値	動作開始	値	R	W
+0	config	b0:b3	Master ID number	0	保持	0h – Fh	○	○
		b4:b7	Slave ID number	0	保持	0h – Fh	○	○
+1	size	b0:b3	Transmit slot number	不定値(注 2)	保持	0h – 8h	○	×
		b4:b7	Receive slot number	不定値(注 2)	保持	0h – 8h	○	×
+2	state	b0:b1	LIN state	0	不定	0: Reset 1: Sleep 2: Run 3: Wakeup	○	×
		b2:b7	Reserve	0	–	–	–	–
+3	snd_flag	b0	transmit slot0	0	保持	0b: not yet 1b: completed	○	○ (注 1)
		b1	transmit slot1	0	保持			
		b2	transmit slot2	0	保持			
		b3	transmit slot3	0	保持			
		b4	transmit slot4	0	保持			
		b5	transmit slot5	0	保持			
		b6	transmit slot6	0	保持			
		b7	transmit slot7	0	保持			
+4	rcv_flag	b0	receive slot0	0	保持	0b: not yet 1b: completed	○	○ (注 1)
		b1	receive slot1	0	保持			
		b2	receive slot2	0	保持			
		b3	receive slot3	0	保持			
		b4	receive slot4	0	保持			
		b5	receive slot5	0	保持			
		b5	receive slot6	0	保持			
		b7	receive slot7	0	保持			
+5	err_flag	b0	UART_ERROR	0	保持	0b: no error 1b: error occur	○	○ (注 1)
		b1	WAKE_UP_ERROR	0	保持			
		b2	BIT_ERROR	0	保持			
		b3	CHECKSUM_ERROR	0	保持			
		b4	IDENTIFIER_PARITY_ERROR	0	保持			
		b5	SLAVE_NOT_RESPONDING_ERROR	0	保持			
		b6	INCONSISTENT_SYNCH_FIELD_ERROR	0	保持			
		b7	NO_BUS_ACTIVITY	0	保持			

(注 1)　“0”だけ書き込み可(“1”を書き込まないでください)。

(注 2)　slin_def.h で設定された値がセットされます。

オフセット	レジスタ名		ビット	内容	初期値	動作開始	値	R	W
+6	SndSlot(0)	slot	b0:b7	slot number	固定値(注 3)	保持	00h – 07h	○	×
+7		com	b0:b1	cmd	00b	保持	00b: idle 01b: transmit 10b: receive 11b: invalid	○	○
			b2:b3	status	00b	保持	00b: idle 01b: invalid 10b: run 11b: end	○	○ (注 1)
			b4:b6	reserve	0	–	0	–	–
			b6	error	0	保持	0b: no error 1b: error occur	○	○ (注 1)
			b7	Don't care	0	–	–	–	–
+8		id	b0:b5	id	不定値(注 2)	保持	00h – 3Fh	○	○
			b6:b7	Don't care	不定値	–	–		
+9		data(0)	b0:b7	data(0)	0	保持	00h – FFh	○	○
+10		data(1)	b0:b7	data(1)	0	保持	00h – FFh	○	○
+11		data(2)	b0:b7	data(2)	0	保持	00h – FFh	○	○
+12		data(3)	b0:b7	data(3)	0	保持	00h – FFh	○	○
+13		data(4)	b0:b7	data(4)	0	保持	00h – FFh	○	○
+14		data(5)	b0:b7	data(5)	0	保持	00h – FFh	○	○
+15		data(6)	b0:b7	data(6)	0	保持	00h – FFh	○	○
+16		data(7)	b0:b7	data(7)	0	保持	00h – FFh	○	○
+17		checksum	b0:b7	checksum	0	保持	00h – FFh	○	×
+6 + 12*1	SndSlot(1)								
+6 + 12*2	SndSlot(2)								
+6 + 12*3	SndSlot(3)								
+6 + 12*4	SndSlot(4)								
+6 + 12*5	SndSlot(5)								
+6 + 12*6	SndSlot(6)								
+6 + 12*7	SndSlot(7)								

(注 1)　"0"だけ書き込み可("1"を書き込まないでください)。

(注 2)　slin_frm.c で設定された値がセットされます。

(注 3)　各スロットのスロット番号がセットされます。

オフセット	レジスタ名		ビット	内容	初期値	動作開始	値	R	W
+6 + 12*8	RcvSlot(0)	slot	b0:b7	slot number	固定値(注 3)	保持	00h – 07h	○	×
+7 + 12*8		com	b0:b1	cmd	00b	保持	00b: idle 01b: transmit 10b: receive 11b: invalid	○	○
			b2:b3	status	00b	保持	00b: idle 01b: invalid 10b: run 11b: end	○	○ (注 1)
			b4:b6	reserve	0	–	0	–	–
			b6	error	0	保持	0b: no error 1b: error occur	○	○ (注 1)
			b7	over write	0	保持	0b: no error 1b: error occur	○	○ (注 1)
+8 + 12*8		id	b0:b5	id	不定値(注 2)	保持	00h – 3Fh	○	○
			b6:b7	Don't care	不定値	–	–	–	–
+9 + 12*8		data(0)	b0:b7	data(0)	0	保持	00h – FFh	○	×
+10 + 12*8		data(1)	b0:b7	data(1)	0	保持	00h – FFh	○	×
+11 + 12*8		data(2)	b0:b7	data(2)	0	保持	00h – FFh	○	×
+12 + 12*8		data(3)	b0:b7	data(3)	0	保持	00h – FFh	○	×
+13 + 12*8		data(4)	b0:b7	data(4)	0	保持	00h – FFh	○	×
+14 + 12*8		data(5)	b0:b7	data(5)	0	保持	00h – FFh	○	×
+15 + 12*8		data(6)	b0:b7	data(6)	0	保持	00h – FFh	○	×
+16 + 12*8		data(7)	b0:b7	data(7)	0	保持	00h – FFh	○	×
+17 + 12*8		checksum	b0:b7	checksum	0	保持	00h – FFh	○	×
+6 + 12*9	RcvSlot(1)								
+6 + 12*10	RcvSlot(2)								
+6 + 12*11	RcvSlot(3)								
+6 + 12*12	RcvSlot(4)								
+6 + 12*13	RcvSlot(5)								
+6 + 12*14	RcvSlot(6)								
+6 + 12*15	RcvSlot(7)								

(注 1) "0"だけ書き込み可("1"を書き込まないでください)。

(注 2) slin_frm.c で設定された値がセットされます。

(注 3) 各スロットのスロット番号がセットされます。

3.2.1 config

・Master ID number
マスタノードのノードＩＤを指定します。本ライブラリでは使用していません。
・Slave ID number
自ノードのノードＩＤを指定します。本ライブラリでは使用していません。

3.2.2 size

・Transmit slot number
RESPONSE 送信用に使用するスロット数を設定します。
・Receive slot number
RESPONSE 受信用に使用するスロット数を設定します。

3.2.3 state

・LIN state
本ミドルウェアのステータスを示す、読みこみ専用レジスタです。ステータスの詳細については、モード説明にて記載します。

3.2.4 snd_flag

・transmit slot0-7
各スロットの RESPONSE 送信完了を示すフラグです。送信完了したスロット番号と同じビットがセットされます。

3.2.5 rcv_flag

・receive slot0-7
各スロットの RESPONSE 受信完了を示すフラグです。受信完了したスロット番号と同じビットがセットされます。

3.2.6 err_flag

エラーの種類を示すフラグです。
・UART_ERROR
UART0 通信に関するエラー（フレーミングエラー・オーバーランエラー）が発生したときにセットされます。
・WAKE_UP_ERROR
スリープ時にウェイクアップフレームを正しく送信もしくは受信できなかった時にセットされます。
・BIT_ERROR
送信したデータと受信したデータとが一致しないときにセットされます。
・CHECKSUM_ERROR
計算したチェックサム値と受信したチェックサム値とが異なる時にセットされます。

チェックサムエラーが発生したとき、そのフレームは受信スロットには書き込まれません。

・IDENTIFIER_PARITY_ERROR

IDENT-FIELD のパリティビットに異常があるときにセットされます。

・SLAVE_NOT_RESPONDING_ERROR

フレームのタイムアウト発生時にセットされます。

・INCONSISTENT_SYNCH_FIELD_ERROR

SYNCH FIELD の誤差が許容範囲外の時にセットされます。

・NO_BUS_ACTIVITY

SYNCH BREAK が 25000 ビットタイム検出されない時にセットされます。

3.2.7 SndSlot(0-7)

送信用スロットです。スロット数は、Transmit slot number により決まります。

・slot : slot_number

各スロットのスロット番号です。

・com : cmd

送信要求フラグです。RESPONSE を送信する時の条件は次の通りです。

・IDENT FIELD 受信時にスロットに書き込まれている id の値と
受信した IDENTIFIER が一致

・cmd=1(transmit)

以上、2 つの条件が揃ったときに RESPONSE を送信します。

・com : status

送信ステータスフラグです。RESPONSE を送信開始すると”2”(run)になり、RESPONSE を送信完了すると”3”(end)となります。

・com : error

データおよびチェックサム送信時エラーです。UART0 エラー、ビットエラー又は送信データ受信タイムアウトの時”1”となります。error のクリアは、API の l_clr_sndsts を実行または、直接 0 書き込みをおこなってください。

・id

RESPONSE を送信する IDENTIFIER を設定します。パリティビットは、送信する際に本ミドルウェアが自動的に計算するので設定する必要はありません。

・data(0-7)

送信するデータを格納します。送信データが８バイトに満たないときも、常に８バイト分確保する必要があります。

・checksum

チェックサムは、送信する際に本ミドルウェアが自動的に計算するので設定する必要はありません。dlc が 0 のとき、チェックサム値は、0FFH を送信します。

3.2.8 RcvSlot(0-7)

受信用スロットです。スロット数は、Receive slot number により決まります。

・slot : slot_number

各スロットのスロット番号です。

・com: cmd

受信要求フラグです。データを受信する時の条件は次の通りです。

- CHECKSUM まで全て受信完了時にスロットに書き込まれている IDENTIFIER と受信した IDENTIFIER が一致
- cmd=2(receive)
- status=idle

以上、３つの条件が揃ったときに受信したメッセージがスロットに書き込まれます。

・com : status

受信ステータスフラグです。スロットに書き込まれている id の値と同じ IDNTIFIER を受信してからデータ、チェックサムを受信するまでの間 ”2” (run) となり、書き込みが終了すると ”3” (end) となります。”3” (end) のときは受信メッセージはスロットに書き込まれないので、オーバーライトエラーとなるので、できるだけ早く受信メッセージを読み込み、status を”0”(Idle)にしてください。

・com : error

データおよびチェックサム受信時エラーです。UART0 エラー、チェックサムエラー又は受信データタイムアウトの時”1”となります。error のクリアは、API の l_clr_rcvsts を実行または、直接 0 書き込みをおこなってください。

・com : over write

オーバーライトエラーです。status=3(end)の時に、受信すると over write は”1”となり、データの上書きはおこないません。over write のクリアは、API の l_clr_rcvsts を実行または、直接 0 書き込みをおこなってください。

・id

RESPONSE を受信する IDENTIFIER を設定します。パリティビットは、受信する際に本ミドルウェアが自動的に計算するので設定する必要はありません。

・data (0-7)

受信したデータが格納されます。受信データが８バイトに満たないときも、常に８バイト分確保する必要があります。

・checksum

受信したチェックサムが格納されます。dlc が 0 のとき、チェックサム値は、0FFH とします。

3.3 LIN バッファステータス

LIN バッファの com の status についてタイミング図を示します。
以下の図を参考にし、送受信完了の判断をおこなってください。

3.3.1 送信データ時

Ident Field が送信スロットの Ident Field と一致した場合、status は図のように変化します。

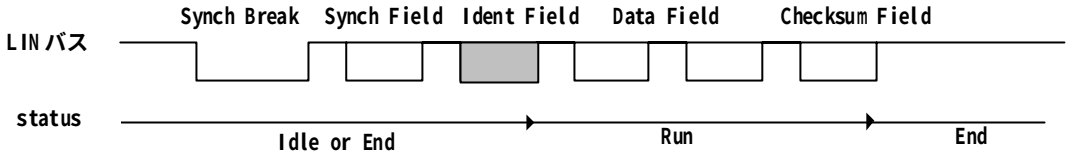


図 3-1 ステータス変化タイミング（送信データ時）

3.3.2 受信データ時

Ident Field が受信スロットの Ident Field と一致した場合、status は図のように変化します。

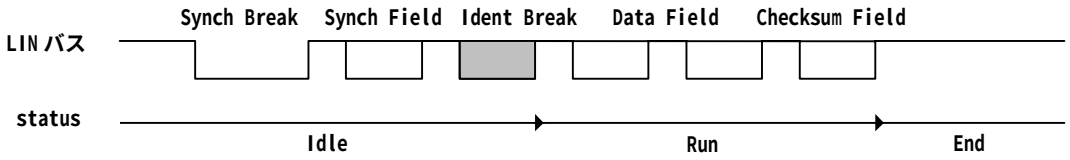


図 3-2 ステータス変化タイミング（受信データ時）

3.3.3 受信データ時（オーバーライト）

Ident Field が受信スロットの Ident Field と一致し、status が End の場合、over write は図のように変化します。

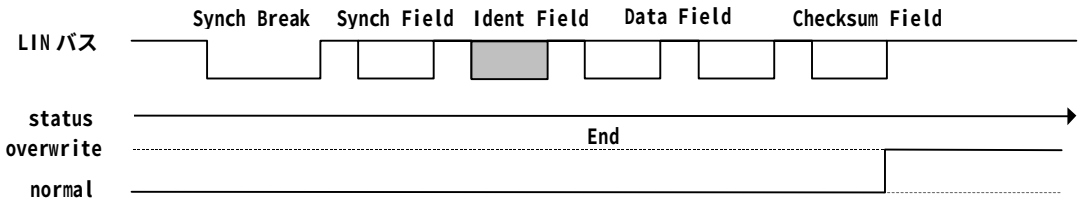


図 3-3 オーバーライトフラグ変化タイミング

3.3.4 送信データ時（エラー）

Ident Field が送信スロットの Ident Field と一致し、UART0 エラー、送信データが不一致もしくはデータ受信できずタイムアウトになった場合、status およびエラーフラグは図のように変化します。

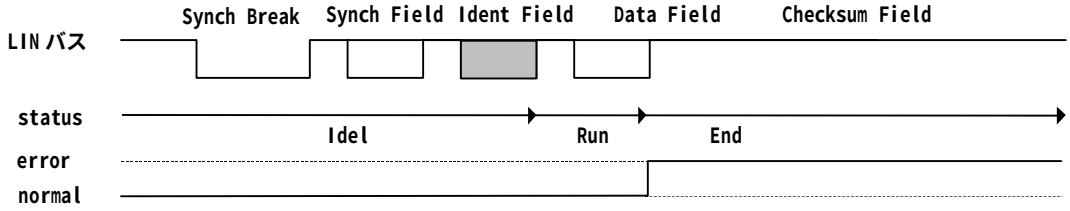


図 3-4 エラーフラグ変化タイミング（送信データ時）

3.3.5 受信データ時（エラー）

Ident Field が受信スロットの Ident Field と一致し、UART0 エラー、チェックサムエラーもしくは受信データタイムアウトの時、status および error は図のように変化します。

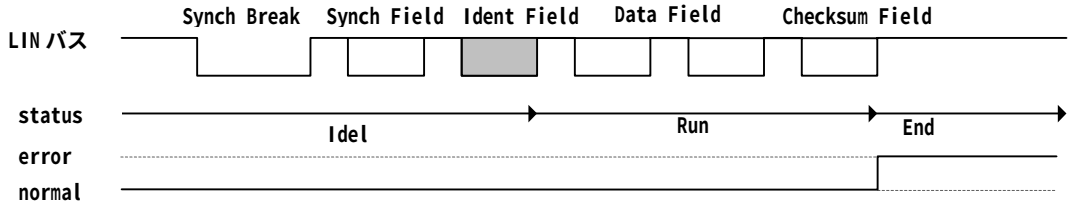


図 3-5 エラーフラグ変化タイミング（受信データ時）

4. LIN スレーブモジュールの動作

LIN スレーブモジュールでは、以下のようなフレーム動作をおこないます。

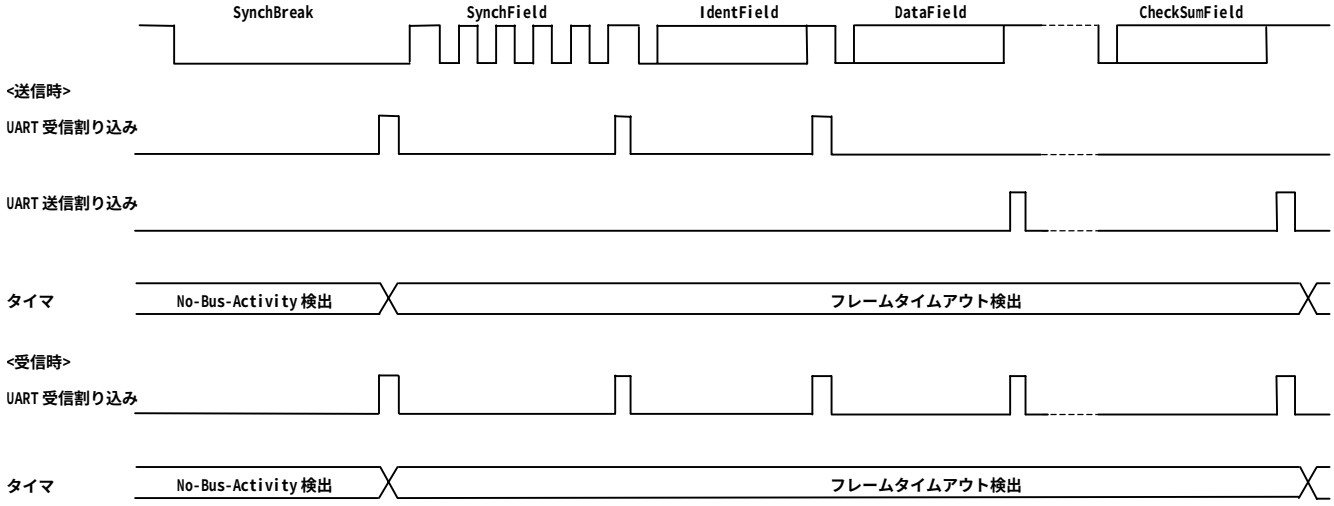


図 4-1 フレーム送受信の動作概要

5．ユーザプログラムへの組み込み

本ミドルウェアを使用する場合、以下の設定をおこなってください。

5.1 動作環境

本ミドルウェアは以下の設定により使用可能です。

MCU	・・・	M32C/85
クロック	・・・	セラミック発振子 8M z（PLL　32MHz）
通信速度	・・・	9600bps
ノード数	・・・	最大 15 ノード（+ 1 マスターノード）

5.2 slin.h

本ミドルウェア用定義ファイルです。インクルードをおこなうことにより LIN バッファへのアクセス、LIN API が使用可能となります。

例）

```
#include “slin.h”
```

5.3 slin_def.h

LIN 通信パラメータ設定ファイルです。インクルードを行うことにより SFR 領域内のレジスタの LIN 用ラベル名が使用可能となります。

表 5-1 slin_def.h

シンボル	内容	設定
LIN_SND_NUM	送信スロット数	0h～8h
LIN_RCV_NUM	受信スロット数	0h～8h

5.4 slin_sfr.h

本ミドルウェア用 SFR 設定ファイルです。

表 5-2の項目については、必要により編集をおこなってください。

表 5-2 slin_sfr.h

パラメータ	内容	設定
LIN_UT_INTLEVEL	UART0 送信割り込み優先レベル設定値	0〜7(デフォルト 5)
LIN_UR_INTLEVEL	UART0 受信割り込み優先レベル設定値	0～7(デフォルト 6)
LIN_TIMER_INTLEVEL	Timer 割り込み優先レベル設定値	0～7(デフォルト 5)

5.5 slin_frm.h

LIN 通信のフレーム定義をおこないます。フレーム名を使って、lin_buffer をアクセスする場合は、インクルードする必要があります。

例）

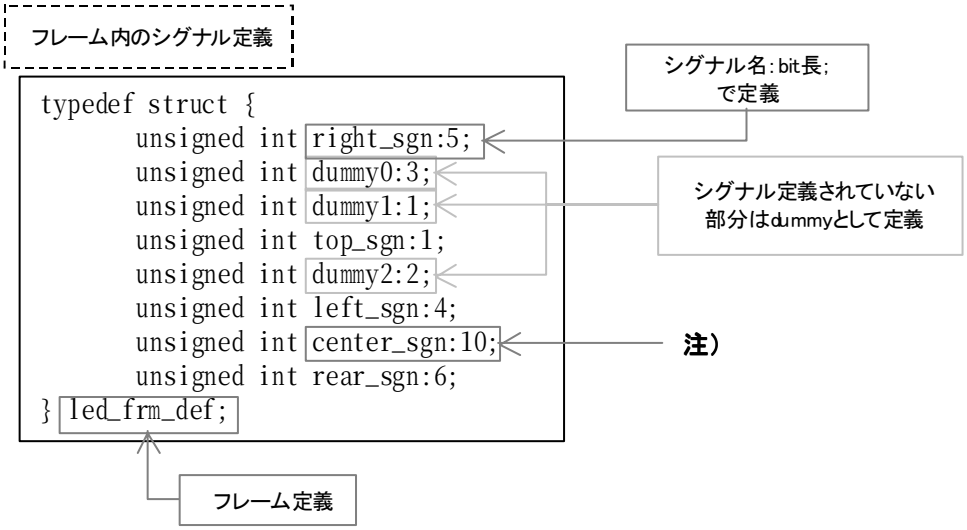
```
#include “slin_frm.h”
```

<slin_frm.h の記述例>

```
/* transmit frame */
extern slib_slot_def *info_frm1;

/* receive frame */
extern slib_slot_def *led_frm;

/* dlc */
extern unsigned char lin_dlc_buffer[64];

typedef struct {
          unsigned int right_sgn:5;
          unsigned int dummy0:3;
          unsigned int dummy1:1;
          unsigned int top_sgn:1;
          unsigned int dummy2:2;
          unsigned int left_sgn:4;
          unsigned int center_sgn:10;
          unsigned int rear_sgn:6;
} led_frm_def;
```

※この部分については
下記を参照ください

※ 上記の<slin_frm.h の記述例>の※部分では、フレーム内のシグナル定義をおこないます。
以下にその構成例を示します。



注)：8bit を超えるシグナルは偶数アドレスに配置するようにしてください。

また、型が異なるビットフィールドが出現した場合は、型が異なるビットフィールド毎に

struct{ }

でくくって下さい。以下に、その構成例を示します。

```
typedef struct {
  struct{
    unsigned char signal1:4;
    unsigned char signal2:3;
    unsigned char dummy0:1;
  }s1;
  struct{
    unsigned int signal3:9;
    unsigned int dummy1:7;
  }s2;
  struct{
    unsigned char signal4:1;
    unsigned char dummy2:7;
  }s3;
} TransmitFrame_def;
```

5.6 slin_frm.c

各スロットの ID,DLC の設定および、slin_frm.h で定義されているフレーム名を LIN バッファの各スロットに割り当てます。

表 5-3 slin_frm.c

シンボル	内容	設定
LIN_SNDFRMID_TBL[]	各送信スロットの ID	0h~3Fh
LIN_RCVFRMID_TBL[]	各受信スロットの ID	0h~3Fh
lin_dlc_buffer[64]	各 ID に対応する DLC	0h~8h

5.7 割り込みベクタテーブルの設定

割り込みベクタテーブルに割り込み処理ルーチンを設定してください。

表 5-4 割り込みベクタ設定

シンボル	内容
LS_LinSndInt	UART0 の送信割り込み
LS_LinRcvInt	UART0 の受信割り込み
LS_DetectTimeOut	No-Bus-Activity 検出、ﾌﾚｰﾑﾀｲﾑｱｳﾄ検出用タイマ割り込み

6. ＡＰＩ

本ミドルウェアを使用する場合の送受信方法、ライブラリとのメッセージの受け渡し方法について記述します。

6.1 LIN の動作モードと API について

本ミドルウェアでは、次の API を持っております。

表 6-1 API 名

分類	API 名	機能	動作モード移行	
初期設定	l_sys_init	ライブラリ初期化	RESET	
	l_ifc_init	LIN ハードウェア初期化	-	
LIN 通信設定	l_ifc_connect	LIN バス接続	SLEEP	
	l_ifc_disconnect	LIN バス切断	RESET	
	l_wakeup	WAKEUP 動作実行	WAKEUP	
	l_sleep	SLEEP 状態へ移行	SLEEP	注 1)
	l_run	RUN 状態へ移行	RUN	注 1)
送信	l_set_sndreq	送信スロット送信許可	-	注 1)
	l_stop_sndreq	送信スロット送信禁止	-	注 1)
	l_read_sndsts	送信ステータスリード	-	注 1)
	l_clr_sndsts	送信ステータスクリア	-	注 1)
	l_set_snddata	送信データセット	-	注 1)
受信	l_set_rcvreq	受信スロット受信許可	-	注 1)
	l_stop_rcvreq	受信スロット受信禁止	-	注 1)
	l_read_rcvsts	受信ステータスリード	-	注 1)
	l_clr_rcvsts	受信ステータスクリア	-	注 1)
	l_get_rcvdata	受信データセット	-	注 1)

注 1)　これら API は、LIN API Revision1.3 仕様には含まれない、API です。

またLINバッファのstateに示される動作モードとAPI及びその他通信イベントの関係は次の図のとおりとなっております。始めは必ず**l_sys_init 、l_ifc_init** を発行する必要があります。

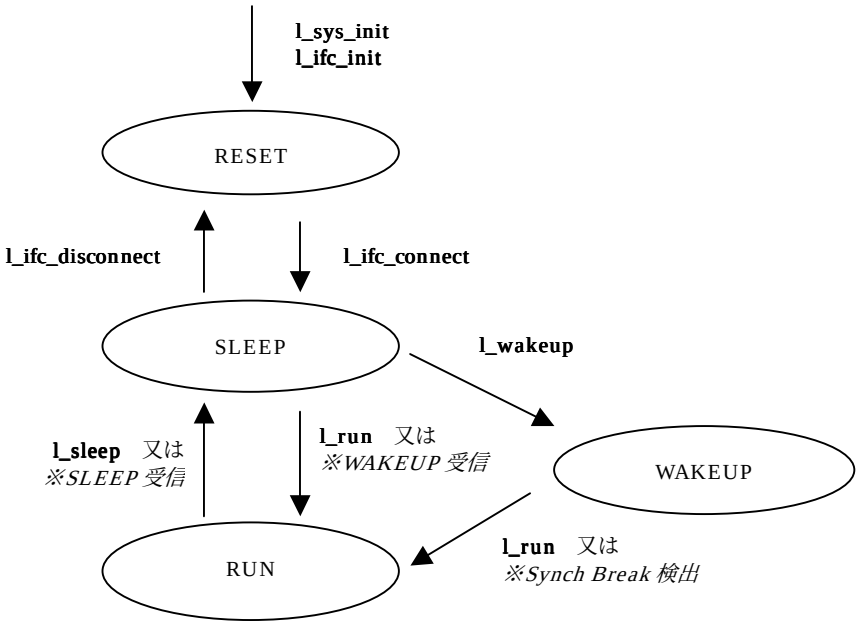


図 6-1 state 状態遷移図

※*斜体*はLINバス上のメッセージを表します。各意味は次の通りです。

SLEEP 受信　　：LIN バスから SLEEP メッセージを検出する。

WAKEUP 受信　：LIN バスから WAKEUP を受信する。

Synch Break 検出 ：LIN バスから正しく Synch Break を検出する。

6.2 初期設定

6.2.1 ノード ID の設定

LIN バッファの config の下位 4bit に、LIN バスに接続される LIN マスタのノード ID、上位 4bit に自ノードのノード ID を設定してください。本ライブラリは、現在使用していません。

例）

```
lb_masterid = 0;                /* Master ID = 0 */
lb_slaveid = 1;                 /* Slave ID = 1 */
```

6.2.2 l_sys_init()

“ l_sys_init() ”を実行すると、本ミドルウェアの初期化がおこなわれます。この API を呼ばない限り、他の API を使用することはできません。必ずユーザプログラムの初期設定ルーチンにて、はじめに実行してください。この API 呼び出し後、本ミドルウェアは RESET 状態となります。

この API は、戻り値として以下を返します。

E_OK：初期化完了

E_NG：初期化失敗

初期化が失敗したときは、オシレータの設定と本ミドルウェアが異なっていること確認してください。

例）

```
if (l_sys_init() == E_NG)
   ;/* 異常処理 */
else
   ;/* 正常処理 */
```

6.2.3 l_ifc_init()

“ l_ifc_init() ”を実行すると、本ミドルウェアが使用するハードウェアの初期化がおこなわれます。この API を呼ばない限り、LIN バスに参加することができません。LIN 通信をおこなう前に、必ず実行してください。

例）

```
l_ifc_init()
```

6.3 通信設定

6.3.1 l_ifc_connect()

“ l_ifc_connect() ”を実行すると、LIN バスに接続され、LIN 通信をおこなうことができます。この API を呼ばない限り、LIN 通信をおこなうことができません。この API 呼び出し後、本ミドルウェアは SLEEP 状態となり、Wakeup フレーム受信待機となります。実際に LIN 通信をおこなう為には、Wakeup フレームを受信するか、”l_wakeup()″ または”l_run()”の API を呼び出す必要があります。

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、すでに LIN バスに接続されている可能性があります。

例）

```
if (l_ifc_connect() == E_NG)
    ;/* 異常処理 */
else
    ;/* 正常処理 */
```

6.3.2 l_ifc_disconnect()

“ l_ifc_disconnect() ”を実行すると、LIN バスから切断され、LIN 通信をおこなうことができなくなります。この API 呼び出し後、本ミドルウェアは RESET 状態となります。再度 LIN バスに接続するためには、“ l_ifc_connect() ”を呼び出してください。

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、本ミドルウェアが動作中か、すでに LIN バスから切断されている可能性があります。

例）

```
if (l_ifc_disconnect() == E_NG)
    ;/* 異常処理 */
else
    ;/* 正常処理 */
```

6.3.3 l_wakeup()

“l_wakeup()”を実行すると、ウェイクアップフレームを送信し、WAKEUP 状態となります。この状態から RUN 状態になるには、Synch Field を受信するもしくは、”l_run()”の API を呼び出す必要があります。

また、この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、本ミドルウェアが SLEEP 状態でない可能性があります。この API は、本ミドルウェアが SLEEP 状態でのみ有効です。

例）

```
if (l_wakeup() == E_NG)
  ;/* 異常処理 */
else
  ;/* 正常処理 */
```

6.3.4 l_sleep()

“l_sleep()”を実行すると、SLEEP 状態となります。

また、この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、本ミドルウェアが RUN 状態でない可能性があります。この API は、本ミドルウェアが RUN 状態でのみ有効です。

例）

```
if (l_sleep() == E_NG)
  ;/* 異常処理 */
else
  ;/* 正常処理 */
```

6.3.5 l_run()

“l_run()”を実行すると、RUN 状態となります。

また、この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、本ミドルウェアが SLEEP 状態または、WAKEUP 状態でない可能性があります。

この API は、本ミドルウェアが SLEEP 状態または、WAKEUP 状態で有効です。

例）

```
if (l_run() == E_NG)
    ;/* 異常処理 */
else
    ;/* 正常処理 */
```

6.4 送信

6.4.1 l_set_sndreq()

“l_set_sndreq()”を実行することにより、指定したスロットを送信許可にすることができます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットがすでに送信許可である可能性があります。

例）

```
if(l_set_sndreq(port0_frm) == E_NG)
   ;/* 異常処理 */
else
   ;/* 正常処理 */
```

6.4.2 l_stop_sndreq()

“l_stop_sndreq()”を実行することにより、指定したスロットを送信禁止にすることができます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットがすでに送信禁止である可能性があります。

例）

```
if(l_stop_sndreq(port0_frm) == E_NG)
   ;/* 異常処理 */
else
   ;/* 正常処理 */
```

6.4.3 l_read_sndsts()

“l_read_sndsts()”を実行することにより、指定したスロットの状態を読むことが出来ます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

LIN_STS_IDLE：アイドル

LIN_STS_RUN：送信中

LIN_STS_END：送信完了

例）

```
if(l_read_sndsts(port0_frm) ==LIN_STS_END )
   ;/* 処理*/
```

6.4.4 l_clr_sndsts()

“l_clr_sndsts()”を実行することにより、指定したスロットの送信ステータスおよび送信完了フラグをクリアすることができます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットが送信中である可能性があります。

例）

```
while(l_clr_sndsts(port0_frm) == E_NG)
   ;/* ウェイト処理 */
```

6.4.5 l_set_snddata()

“l_set_snddata()”を実行することにより、指定したスロットに送信データをセットすることが出来ます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

(l_u8) data_buffer[8]： データバッファ（8 バイト）

例）

```
l_set_snddata(port0_frm, message_buffer);
```

6.5 受信

6.5.1 l_set_rcvreq()

“ l_set_rcvreq() ”を実行することにより、指定したスロットを受信許可にすることができます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットがすでに受信許可である可能性があります。

例）

```
if(l_set_rcvreq(port0_frm) == E_NG)
    ;/* 異常処理 */
else
    ;/* 正常処理 */
```

6.5.2 l_stop_rcvreq()

“ l_stop_rcvreq() ”を実行することにより、指定したスロットを受信禁止にすることができます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットがすでに受信禁止である可能性があります。

例）

```
if(l_stop_rcvreq(port0_frm) == E_NG)
    ;/* 異常処理 */
else
    ;/* 正常処理 */
```

6.5.3 l_read_rcvsts()

“l_read_rcvsts()”を実行することにより、指定したスロットの状態を読むことが出来ます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

LIN_STS_IDLE：アイドル

LIN_STS_RUN：受信中

LIN_STS_END：受信完了

例）

```
if(l_read_rcvsts(port0_frm) ==LIN_STS_END )
   ;/* 処理*/
```

6.5.4 l_clr_rcvsts()

“l_clr_rcvsts()”を実行することにより、指定したスロットの受信ステータスおよび受信完了フラグをクリアすることができます。一度受信完了したスロットを再度受信設定するためには、この API を実行する必要があります。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットが受信完了でない可能性があります。

例）

```
while(l_clr_rcvsts(port0_frm) == E_NG)
   ;/* ウェイト処理 */
```

6.5.5 l_get_rcvdata()

“l_get_rcvdata()”を実行することにより、指定したスロットから受信データをリードすることが出来ます。

この API は、以下の引数を与える必要があります。

(slib_slot_def) *lin_frame： LIN フレーム構造体（slin_frm.h で設定）

(l_u8) data_buffer[8]： データバッファ（8 バイト）

この API は、戻り値として以下を返します。

E_OK：完了

E_NG：失敗

失敗したときは、そのスロットが受信エラーの可能性があります。

例）

```
while(l_get_rcvdata(port0_frm, message_buffer) == E_NG)
   ;/* ウェイト処理 */
```

7．送信処理手順

本ミドルウェアを使用して LIN メッセージの RESPONSE を送信する手順を説明します。

7.1 送信要求

7.1.1 API を使用する場合

l_set_sndreq()、l_set_snddata()を使用して、RESPONSE 送信要求をすることができます。

例）Data0=11h, Data1=22h, Data2=33h, Data3=44h

を、led_frm に設定する場合

```
l_u8 data_buf[8];
data_buf[0] = 0x11;
data_buf [1] = 0x22;
data_buf [2] = 0x33;
data_buf [3] = 0x44;
l_set_snddata(led_frm, data_buf);          // 送信データを led_frm にセット
l_set_sndreq(led_frm);                     //  送信許可
```

7.1.2 API を使用しない場合

直接 LIN バッファにアクセスして、RESPONSE 送信要求をすることができます。なお、データを LIN バッファに直接書き込む際は、送信データを送信スロットに書き込み中に送信対象スロットの ID を受信すると、書き込み途中のデータが送信される為、割り込み禁止にしてから行ってください。

例）Data0=11h, Data1=22h, Data2=33h, Data3=44h

を、led_frm に設定する場合

```
asm(“FCLR I”);                             // 割り込み禁止
led_frm->data[0] = 0x11;
led_frm->data[1] = 0x22;
led_frm->data[2] = 0x33;
led_frm->data[3] = 0x44;
led_frm->com.BIT.cmd = LIN_REQ_SND;
asm(“FSET I”);                             // 割り込み許可
```

7.2 送信完了確認

7.2.1 API を使用する場合

l_read_sndsts を使用して、送信完了を確認することができます。

例）フレーム名 led_frm,speed_frm,power_frm の送信完了を確認する場合

```
if(l_read_sndsts(led_frm) == LIN_STS_END){
        ; /* 送信完了処理 */
        l_clr_sndsts(led_frm);
}
if(lb_read_sndsts(&speed_frm) == LIN_STS_END){
        ; /* 送信完了処理 */
        l_clr_sndsts(speed_frm);
}
if(lb_read_sndsts(power_frm) == LIN_STS_END){
        ; /* 送信完了処理 */
        l_clr_sndsts(power_frm);
}
```

7.2.2 API を使用しない場合

各スロットの com を使用して、送信完了を確認することができます。

例）フレーム名 led_frm,speed_frm,power_frm の送信完了を確認する場合

```
if(led_frm->com.BIT.status == LIN_STS_END){
        ; /* 送信完了処理 */
        led_frm->com.BIT.status = LIN_STS_IDLE;
}
if(speed_frm->com.BIT.status == LIN_STS_END){
        ; /* 送信完了処理 */
        speed_frm->com.BIT.status = LIN_STS_IDLE;
}
if(power_frm->com.BIT.status == LIN_STS_END){
        ; /* 送信完了処理 */
        power_frm->com.BIT.status = LIN_STS_IDLE;
}
```

8．受信処理手順

本ミドルウェアを使用して LIN メッセージの RESPONSE を受信する手順を説明します。

8.1 受信要求

8.1.1 API を使用する場合

l_set_rcvreq()を使用して、RESPONSE 受信要求をすることができます。

例）led_frm に設定する場合

```
l_set_rcvreq(led_frm);                          //  受信許可
```

8.1.2 API を使用しない場合

直接 LIN バッファにアクセスして、RESPONSE 受信要求をすることができます。

例）led_frm に設定する場合

```
led_frm->com.BIT.cmd = LIN_REQ_RCV;
```

8.2 受信完了確認

8.2.1 API を使用する場合

l_read_rcvsts を使用して、受信完了を確認することができます。

例）フレーム名 led_frm,speed_frm,power_frm の受信完了を確認する場合

```
l_u8   data_buf[8];
if(l_read_rcvsts(led_frm) == LIN_STS_END){
        l_get_rcvdata(led_frm,data_buf);          // 受信データの格納
        l_clr_rcvsts(led_frm);
        ; /* 受信完了処理 */
}
if(lb_read_rcvsts(speed_frm) == LIN_STS_END){
        l_get_rcvdata(speed_frm,data_buf);        // 受信データの格納
        l_clr_rcvsts(speed_frm);
        ; /* 受信完了処理 */
}
if(lb_read_rcvsts(power_frm) == LIN_STS_END){
        l_get_rcvdata(power_frm,data_buf);        // 受信データの格納
        l_clr_rcvsts(power_frm);
        ; /* 受信完了処理 */
}
```

8.2.2 API を使用しない場合

各スロットの com を使用して、受信完了を確認することができます。

例）フレーム名 led_frm,speed_frm,power_frm の受信完了を確認する場合

```
l_u8    data_buf[8];
if(led_frm->com.BIT.status == LIN_STS_END){
        for(i=0; i<led_frm->dlc; i++)                   // 受信データ数
                data_buf[i] = led_frm->data[i];         // 受信データの格納
        led_frm->com.BIT.status = LIN_STS_IDLE;
        ; /*  受信完了処理  */
}
if(speed_frm->com.BIT.status == LIN_STS_END){
        for(i=0; i<speed_frm.dlc; i++)                  // 受信データ数
                data_buf[i] = speed_frm->data[i];       // 受信データの格納
        speed_frm->com.BIT.status = LIN_STS_IDLE;
        ; /*  受信完了処理  */
}
if(power_frm->com.BIT.status == LIN_STS_END){
        for(i=0; i<power_frm.dlc; i++)                  // 受信データ数
                data_buf[i] = power_frm->data[i];       // 受信データの格納
        power_frm->com.BIT.status = LIN_STS_IDLE;
        ; /*  受信完了処理  */
}
```

9. エラー

9.1 エラー検出

本ミドルウェアでは、以下のエラーを検出することが出来ます。

- UART_ERROR
- WAKE_UP_ERROR
- BIT_ERROR
- CHECKSUM_ERROR
- IDENTIFIER_PARITY_ERROR
- SLAVE_NOT_RESPONDING_ERROR
- INCONSISTENT_SYNCH_FIELD_ERROR
- NO_BUS_ACTIVITY_ERROR

それぞれのエラーを検出するタイミングを以下に示します。

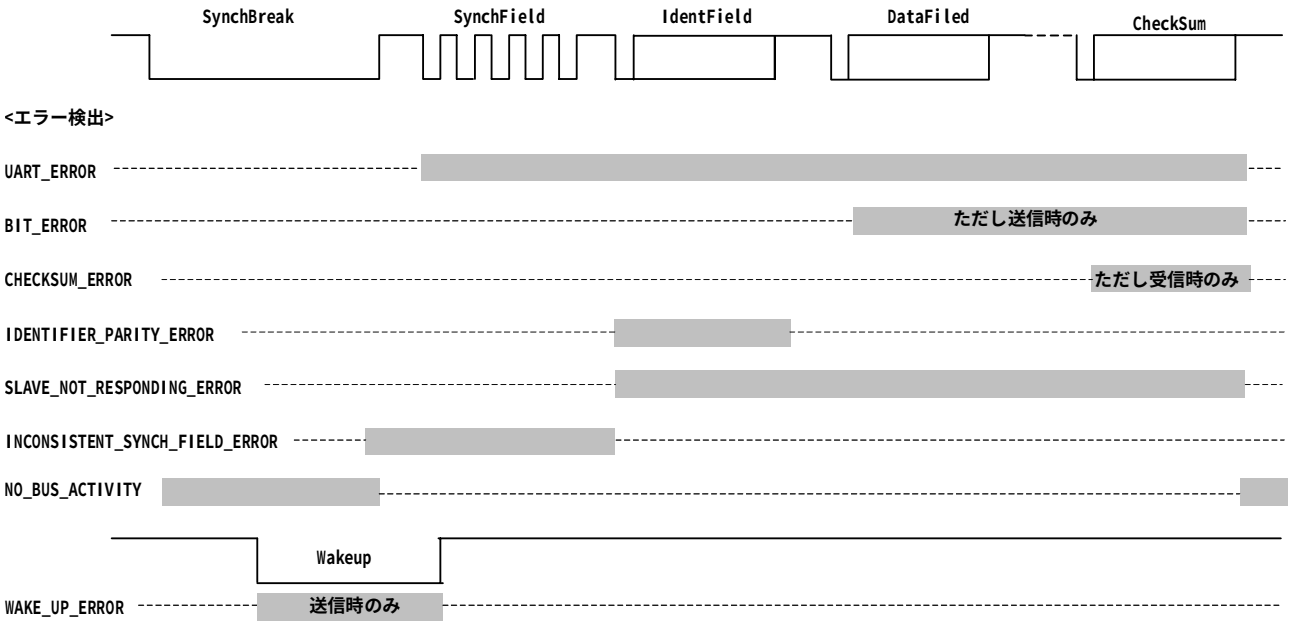


図 9-1 エラー検出時の動作概要

10．使用上の注意事項

本ミドルウェアの使用上の注意事項について説明します。ご使用になる場合は必ず制限事項に従い使用してください。

10.1 周辺機能の使用に関する制限事項

以下の機能は本ミドルウェアに使用します。ユーザプログラムでは使用している H/W と競合しないよう選択してください。

表 10-1 使用周辺機能

機能	用途
UART0 送受信	送信時に TxD 波形を送信するために使用します。 受信時に RxD 波形を受信するために使用します。 Synch Break を検出するために使用します。 SLEEP 時、ウェイクアップメッセージの判定をおこないます。
TA0	No-Bus-Activity の計測およびメッセージの Time Out を検出するために使用します。

10.2 端子の使用に関する制限事項

以下の端子は本ミドルウェアに使用します。他の用途での使用はしないでください。

表 10-2 端子使用制限

端子名	用途
RxD	LIN メッセージを受信します。 UART0 受信端子として使用します。
TxD	LIN メッセージを送信します。 UART0 送信端子として使用します。

10.3 割り込みに関する制限事項

UART0 受信割り込み及びタイマ割り込みは本ミドルウェアに使用します。本ミドルウェアを使用するときはベクタテーブルを以下のように設定をおこなう必要があります。

表 10-3 割り込み番地設定

割り込み要因	割り込み飛び先番地
UART0 送信	LS_LinSndInt
UART0 受信	LS_LinRcvInt
TA0	LS_DetectTimeOut

本ミドルウェアは UART0 受信割り込み、タイマ割り込みを使用して通信をおこないます。そのためユーザプログラムの割り込み処理で、割り込み禁止時間が 1 ビットタイムを超える場合、本ミドルウェアに異常が発生し、通信異常により通信が停止する可能性があります。よって、ユーザプログラムの割り込み禁止時間が 1 ビットタイムを超える場合は、ユーザプログラムの割り込み優先レベルを、本ミドルウェアで使用している割り込み処理の割り込み優先レベルより低く設定し、割り込み許可フラグ（I フラグ）を"1"（割り込み許可）に設定して下さい。なお、本ミドルウェアで使用している割り込み処理の割り込み優先レベルについては、「5.4 slin_sfr.h」を参照ください。

10.4 LIN トランシーバ

LIN トランシーバとは、以下のように接続する必要があります。

LIN トランシーバ Tx：MCU の TxD と接続

LIN トランシーバ Rx：MCU の RxD と接続

改訂履歴

Version	変更内容	日付	作成者
Ver.1.00	・新規作成	2007/10/31	サニー技研