

# AVR286 : LIN/UART制御器用LINファームウェア基礎

## LIN特徴

局所相互連結網(LIN:Local Interconnect Network)は分配された車載応用でのメカトロクス節点(ノード)の制御を効率的に支援する直列通信規約です。LINバスの主な特性は次のとおりです。

- ・ 複数の従装置を持つ単一主装置の概念
- ・ 一般的なUART通信に基づく安価なシリコン実装
- ・ 従装置節点(ノード)に於けるチップ上発振器での自己同期
- ・ 予め計算可能な単一伝播時間での確定的な信号送信
- ・ 安価な単線実装
- ・ 20kビット/sまでの速度

## 1. Atmel® LIN/UART制御器

LIN/UART制御器はATmega32/64M1, C1やATtiny167のようないくつかのAVR® マイクロコントローラで利用可能です。

### 1.1. 要点

- ・ LIN 2.xのハードウェア実装(LIN 1.x互換)
- ・ 小さくCPU効率的で、LIN2.x仕様の“LIN作業の流れの概念”に基く独立した主/従ルーチン
- ・ 自動LIN先頭部処理と無関係なLINフレームの濾過
- ・ 自動LIN応用処理
- ・ 拡張されたLIN異常検出と合図
- ・ ハードウェア フレーム時間超過検出
- ・ “データ内中断”支援能力
- ・ 正しく完全なフレームを保証するための自動再同期
- ・ 完全な柔軟性を持つ拡張フレーム支援能力

### 1.2. 制御器概要

LIN/UART制御器はLINソフトウェア応用構造に可能な限り近く合うように設計されています。LINソフトウェア応用は多数の従作業と1つの主作業の独立した作業として開発されています。制御器はこの分割から成ります。主節点(ノード)での主作業と従作業間の繋がりはIDOKの(割り込み)フラグです。主作業について、この事象はこの作業の終りを表し、従作業についてはこの作業の始まりを表します。

従作業が識別子の存在を知らされると、応答で何を行うかを知るため、最初にそれを分析します。ハードウェア フラグは60(\$3C)から63(\$3F)までの特定識別子の1つの存在を識別します。

## 2. ファームウェア基礎

### 2.1. 説明

この応用記述はATmega32/64,M1/C1とATtiny167のLIN/UART制御器を使うためのファームウェア基礎(ドライバ)を提供します。それは読者がデバイスの構造とLIN/UART制御器に精通していると仮定します。LIN仕様v1.xまたはv2.x([www.lin-subbus.org](http://www.lin-subbus.org))の最低限の知識はこの資料の内容の理解も必要とされます。

この資料はAtmel AVRマイクロコントローラでのそれらの応用またはそれらのLIN階層構築の開発を手助けするようにソフトウェア開発者のために書かれています。ドライバはLIN/UART制御器のプログラミングを簡単化するように設計されています。

それらの応用を素早く構築する開発者を手助けするためにいくつかの一般的な情報と助言も検討されます。加えて、この資料はそれらのドライバの使い方を説明するための応用例を提供します。

### 2.2. 制限

- ・ **診断フレーム** : ドライバ上に置かれるソフトウェアのLINライブラリによって処理される



8ビット **AVR**<sup>®</sup>  
マイクロコントローラ

## 応用記述

本書は一般の方々の便宜のため有志により作成されたもので、Atmel社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 8123A-03/08, 8123AJ2-04/21

- ・ 休止と起き上がり：応用依存機能
- ・ LINハードウェア検査目的用構成設定：含まれません。

## 2.3. 用語

ID：LIN識別子  
Tx：送信  
Rx：受信

## 2.4. コード書き様式

下で説明されるコード書き様式はファームウェアを理解するのに重要です。

- ・ 定義された定数は大文字を使います。

```
#define CONF_LINBRR 25
```

- ・ C-マクロ関数は先頭文字を大文字として使います。

```
#define Is_lin_header_ready() ((LINSIR & (1<<LIDOK)) ? TRUE: FALSE)
```

- ・ 関数は小文字を使います。

```
void lin_get_response (unsigned char *l_data)
```

## 2.5. デバイス特定I/Oヘッダ ファイル

この資料で記述されるファームウェアはコンパイラによって提供されるデバイス特定I/Oヘッダ ファイルに基づきます。

1. AVRGCC：“iom64m1.h”、“iom64c1.h”、“iom32m1.h”または“iom32c1.h”
2. IAR：“iom64m1.h”、“iom64c1.h”、“iom32m1.h”または“iom32c1.h”
3. ～

デバイス特定I/Oヘッダ ファイルに於いて、全てのレジスタの名前、アドレス、そして内容(レジスタビット)が定義されます。

## 3. 定義

最初に応用は以下の値を定義しなければなりません。

```
#define FOSC 8000 // kHzでのデバイス周波数
#define LIN_BAUDRATE 19200 // ビット/秒でのLINボーレート
```

### 3.1. ビット時間

ビット時間定義のCONF\_LINBRRは通信用のボーレートを構成設定します。この定義はFOSCとLIN\_BAUDRATEによって制御されます。最後に於いて、1つのCONF\_LINBRR値だけがLIN/UART制御器のLINボーレートレジスタを書くのに記憶されます。

```
#ifndef FOSC
#error FOSCを定義しなければなりません。
#elif FOSC == 16000 // デバイス周波数=16MHzの場合
#ifndef LIN_BAUDRATE
#error You must define LIN_BAUDRATE
#elif LIN_BAUDRATE == 19200
#define CONF_LINBRR 25 // (0x19) 19.2kbps、誤差=0.2%
#elif LIN_BAUDRATE == 9600
#define CONF_LINBRR 51 // (0x33) 9.6kbps、誤差=0.2%
#elif LIN_BAUDRATE == 4800
#define CONF_LINBRR 103 // (0x67) 4.8kbps、誤差=0.2%
#elif LIN_BAUDRATE == 2400
#define CONF_LINBRR 207 // (0xCF) 2.4kbps、誤差=-0.2%
#else
#error 利用可能でないLIN_BAUDRATE値
#endif
#elif FOSC == 8000 // デバイス周波数=8MHzの場合
#ifndef LIN_BAUDRATE
#error You must define LIN_BAUDRATE
#elif LIN_BAUDRATE == 19200
#define CONF_LINBRR 12 // (0x0C) 19.2kbps、誤差=0.2%
#elif LIN_BAUDRATE == 9600
#define CONF_LINBRR 25 // (0x19) 9.6kbps、誤差=0.2%
```

```

#elif LIN_BAUDRATE == 4800
#define CONF_LINBRR 51 // (0x33) 4.8kbps、誤差=0.2%
#elif LIN_BAUDRATE == 2400
#define CONF_LINBRR 103 // (0x67) 2.4kbps、誤差=-0.2%
#else
#error 利用可能でないLIN_BAUDRATE値
#endif
#else
#error 利用可能でないFOSC値
#endif

```

この方法はCプロセッサの条件付コンパイルを使います。この方法はCでのコード行と比べて時間とコードバイトを節約します。他のFOC定義(例:FOSC == 4000)に対して新しいCONF\_LINBRR値の組を追加することが非常に容易です。CONF\_LINBRR値は以下の式を使って計算することです。

$$\text{CONF\_LINBRR} = \lceil \text{丸め} \{ (\text{FOSC} \times 1000) / (32 \times \text{LIN\_BAUDRATE}) \} \rceil - 1$$

## 3.2. 構成設定

ビット時間定義のCONF\_LINBRRは通信用のボーレートを構成設定します。この定義はFOCとLIN\_BAUDRATEによって制御されます。

```

// LIN規約
#define LIN_1X (1<<LIN13)
#define LIN_2X (0<<LIN13)

// LIN命令
#define LIN_CMD_MASK ((1<<LCMD1) | (1<<LCMD0))
#define LIN_RX_HEADER ((0<<LCMD1) | (0<<LCMD0))
#define LIN_ABORT ((0<<LCMD1) | (0<<LCMD0))
#define LIN_TX_HEADER ((0<<LCMD1) | (1<<LCMD0))
#define LIN_RX_RESPONSE ((1<<LCMD1) | (0<<LCMD0))
#define LIN_TX_RESPONSE ((1<<LCMD1) | (1<<LCMD0))

// LIN割り込み要求フラグ
#define LIN_ERROR (1<<LERR)
#define LIN_IDOK (1<<LIDOK)
#define LIN_RXOK (1<<LRXOK)
#define LIN_TXOK (1<<LTXOK)

// LIN ID印
#define LIN_1X_ID_MASK ((1<<LID3) | (1<<LID2) | (1<<LID1) | (1<<LID0))
#define LIN_1X_LEN_MASK ((1<<LID5) | (1<<LID4))
#define LIN_2X_ID_MASK ((1<<LID5) | (1<<LID4) | LIN_1X_ID_MASK)

// ATmega32/64/M1/C1特有 // ATtiny167特有
#define LIN_PORT_IN PIN_D // ATmega32/64/M1/C1特有
#define LIN_PORT_DIR DDRD // ATmega32/64/M1/C1特有
#define LIN_PORT_OUT PORTD // ATmega32/64/M1/C1特有
#define LIN_INPUT_PIN 4 // ATmega32/64/M1/C1特有
#define LIN_OUTPUT_PIN 3 // ATmega32/64/M1/C1特有
#define LIN_PORT_IN PINA // ATtiny167特有
#define LIN_PORT_DIR DDRA // ATtiny167特有
#define LIN_PORT_OUT PORTA // ATtiny167特有
#define LIN_INPUT_PIN 0 // ATtiny167特有
#define LIN_OUTPUT_PIN 1 // ATtiny167特有

```

この定義の方法はレジスタアドレスの変更とレジスタビットの割り当てを許容します。

## 4. C-マクロ

C-マクロは一般的にCに於いてコードの小さな断片を定義するのに使われます。事前処理段階中に、各C-マクロ呼び出しはC-マクロ定義に対応することによってインラインで置き換えられます。C-マクロがパラメータを持つ場合、それらは展開中にC-マクロ本体内で交換され、従ってC-マクロは模擬的なC-関数にできます。これを行う通常理由は、関数呼び出しの付随作業が性能に於いて重要な影響を持つほど十分に軽いコードで単純な場合に関数呼び出しの付随作業を避けるためです。

### 4.1. LIN 1.x用C-マクロ

```

// LIN 1.x規約使用
#define Lin_lx_set_type() (LINCRCR |= LIN_1X)
#define Lin_lx_enable() (LINCRCR = LIN_1X | (1<<LENA) | (0<<LCMD2))

```

```

#define Lin_lx_set_id(id)      {LINIDR &= ~LIN_1X_ID_MASK; ¥
                               LINIDR |= id & LIN_1X_ID_MASK;}
#define Lin_lx_set_len(len)   {LINIDR &= ~LIN_1X_LEN_MASK; ¥
                               LINIDR |= (((len+4)<<2) & LIN_1X_LEN_MASK);}
#define Lin_get_len()         (LINDLR & (0x0F << LRXDLO) )
// Lin_set_rx_len(len) - LIN 1.xでの応答用自動設定
// Lin_set_tx_len(len) - LIN 1.xでの応答用自動設定

```

## 4.2. LIN 2.x用C-マクロ

```

// LIN 2.x規約使用
#define Lin_2x_set_type()     ( LINCRL |= LIN_2X )
#define Lin_2x_enable()      ( LINCRL = LIN_2X | (1<<LENA) | (0<<LCMD2) )
#define Lin_2x_set_id(id)    {LINIDR &= ~LIN_2X_ID_MASK; ¥
                               LINIDR |= id & LIN_2X_ID_MASK;}
// Lin_2x_set_len(len) - LIN 2.1で利用不可
// Lin_get_len() ----- LIN 2.1で利用不可
#define Lin_set_rx_len(len)  ( LINDLR = len & (0x0F << LRXDLO) )
#define Lin_set_tx_len(len)  ( LINDLR = len << LTXDLO )

```

## 4.3. 共用されるC-マクロ

```

// 何れかの規約の使用
#define Lin_set_type(ltype)   (( ltype == LIN_2X ) ? ¥
                               Lin_2x_set_type() : Lin_lx_set_type() )
#define Lin_get_type()       ( LINCRL & (1<<LIN1X) )
#define Lin_set_len(len)     ( LINDLR = (len<<LTXDLO) | (len & (0x0F<<LTXDLO) ) )

// その他のC-マクロ
#define Lin_get_error_status() ( LINERR )
#define Lin_set_baudrate(br)  {LINBRRH = (unsigned char)((unsigned short)br)>>8); ¥
                               LINBRRL = (unsigned char) ((unsigned short)br);}
#define Lin_sw_reset()       ( LINCRL = 1<<LSWRES )
#define Lin_full_reset()     { Lin_sw_reset(); Lin_clear_enable_it(); ¥
                               LINBRRL = 0x00; LINBRRH = 0x00; }

// 割り込み処理
#define Lin_get_it()          ( LINSIR & ((1<<LERR) | (1<<LIDOK) | (1<<LTXOK) | (1<<LRXOK)) )
#define Lin_set_enable_it()  ( LINENIR = (1<<LENERR) | (1<<LENIDOK) | (1<<LENTXOK) | (1<<LENRXOK) )
#define Lin_clear_enable_it() ( LINENIR = (0<<LENERR) | (0<<LENIDOK) | (0<<LENTXOK) | (0<<LENRXOK) )
#define Lin_clear_err_it()   ( LINSIR = 1<<LERR )
#define Lin_clear_idok_it()  ( LINSIR = 1<<LIDOK )
#define Lin_clear_txok_it()  ( LINSIR = 1<<LTXOK )
#define Lin_clear_rxok_it()  ( LINSIR = 1<<LRXOK )

// LIN命令
#define Lin_abort()           (LINCRL &= ~LIN_CMD_MASK)
#define Lin_rx_header()      (LINCRL &= ~LIN_CMD_MASK)
#define Lin_tx_header()      {LINCRL &= ~LIN_CMD_MASK; LINCRL |= LIN_TX_HEADER ;}
#define Lin_rx_response()    {LINCRL &= ~LIN_CMD_MASK; LINCRL |= LIN_RX_RESPONSE;}
#define Lin_tx_response()    {LINCRL &= ~LIN_CMD_MASK; LINCRL |= LIN_TX_RESPONSE;}

// LIN調査
#define Is_lin_header_ready() ( (LINSIR & (1<<LIDOK)) ? 1 : 0 )
#define Is_lin_rx_response_ready() ( (LINSIR & (1<<LRXOK)) ? 1 : 0 )
#define Is_lin_tx_response_ready() ( (LINSIR & (1<<LTXOK)) ? 1 : 0 )

// IDとデータの処理
#define Lin_get_id()          ( LINIDR & LIN_2X_ID_MASK )
#define Lin_clear_index()    ( LINSSEL = (0<<LAINC) | (0x00<<LINDX0) )
#define Lin_get_data()       ( LINDAT )
#define Lin_set_data(data)   ( LINDAT = data )

```

#### 4.4. C-マクロでの注意

長いC-マクロが頻度良すぎで使われる場合、生成されるコードの大きさが不利にされます(各C-マクロ呼び出しは対応するC-マクロ定義によってインラインで置き換えられます)。その時はそれをC-関数で簡潔な形にされるのが適切です。

### 5. C-関数

#### 5.1. LIN/UART制御器初期化

この関数はLIN制御器と必要ならばLIN割り込みを初期化します。

関数には2つの引数が渡されます。

1. 'unsigned char l\_type' : 構成により、'l\_type'は'LIN\_1X'または'LIN\_2X'のどちらかです。
2. 'unsigned long b\_rate' : LINボーレートレジスタ(LINBRR)値

関数は以下を返します。

'unsigned char' :       0 : 初期化失敗、LIN形式が一致しません。  
                  0以外 : 初期化遂行

警告 : なし

##### 5.1.1 原型

```
extern unsigned char lin_init (unsigned char l_type, unsigned long b_rate);
```

##### 5.1.2 関数

```
unsigned char lin_init (unsigned char l_type, unsigned long b_rate)
{
    // TxLINとRxLINでのプルアップ(ビット アドレス指定を使うために1ずつ)
    LIN_PORT_DIR &= ~(1<<LIN_INPUT_PIN );
    LIN_PORT_DIR &= ~(1<<LIN_OUTPUT_PIN);
    LIN_PORT_OUT |= (1<<LIN_INPUT_PIN );
    LIN_PORT_OUT |= (1<<LIN_OUTPUT_PIN);

    Lin_full_reset();
    Lin_set_baudrate(b_rate);

    if (l_type == LIN_1X) {
        Lin_1x_enable();
    } else if (l_type == LIN_2X) {
        Lin_2x_enable();
    } else {
        return 0;
    }

    // LINが割り込み駆動なら、以下の2行を許可してください。
    // Lin_set_enable_it();
    // asm ("sei");
    return 1;
}
```

##### 5.1.3. 注意と備考

LIN/UART制御器は規約の使用者独自化を許します。

- 受信したフレーム内でチェックサム(CHECKSUM)領域の検出、CHECKSUMの送信、またはフレーム時間超過を取り去ることが可能です。
- これらの機能が不十分な場合、LIN/UART制御器は使用者の最大自由化を許すために純粋なUARTとして初期化することもできます。その時に先頭部を管理するための自動機能と応答を管理するための"手動"形態(UART動作形態)への切り替えを維持することが可能です。
- 同じ応答で送受信を混ぜるシステムも可能です。

## 5.2. LIN先頭部送信

この関数は主節点(ノード)の主作業でのLIN先頭部送出を命令します。

関数には3つの引数が渡されます。

1. 'unsigned char l\_type' : 構成により、'l\_type'は'LIN\_1X'または'LIN\_2X'のどちらかです。
2. 'unsigned long l\_id' : LIN識別子値。'LIN\_1X'の場合、符号化された長さがLIN識別子内に運ばれます。
3. 'unsigned long l\_len' : 応答で運ばれるデータバイト数、(符号化されない)真の長さ。この情報はそれが'l\_id'で符号化されるため、'LIN\_1X'で使われません。

関数は以下を返します。

'unsigned char' : 0 : 初期化失敗、LIN形式が一致しません。  
0以外 : 先頭部進行中の先頭部送出

警告 : なし

### 5.2.1. 原型

```
extern unsigned char lin_tx_header (unsigned char l_type, unsigned char l_id, unsigned char l_len);
```

### 5.2.2. 関数

```
unsigned char lin_tx_header (unsigned char l_type, unsigned char l_id, unsigned char l_len)
{
    Lin_abort(); // 制御器が未だ'lin_tx_response'または'lin_rx_response'動作の場合に有用です。それらの動作走行
                // 時にLINレジスタ書き込みが禁止され、IDを制御器に設定できないことに注意してください。
                // ("データ中の中断"の動きを参照)

    if (l_type == LIN_1X) {
        Lin_1x_set_id(l_id);
        Lin_1x_set_len(l_len);
    } else if (l_type == LIN_2X) {
        Lin_2x_set_id(l_id); // LIN 2.xでは運ばれた長さなし
    } else {
        return 0;
    }

    Lin_tx_header(); // 命令設定
    return 1;
}
```

### 5.2.3. 注意と備考

送信中に異常がないのを確認することが重要です。

## 5.3. LIN応答受信

この関数は主または従の節点(ノード)の従作業でのLIN応答の受信を命令します。

関数には2つの引数が渡されます。

1. 'unsigned char l\_type' : 構成により、'l\_type'は'LIN\_1X'または'LIN\_2X'のどちらかです。
2. 'unsigned long l\_len' : 応答で予測されるデータバイト数、(符号化されない)真の長さ。この情報はそれが'l\_id'で符号化されるため、'LIN\_1X'で使われません。

関数は以下を返します。

'unsigned char' : 0 : 初期化失敗、LIN形式が一致しません。  
0以外 : 受信進行中の応答

警告 : 受信の最後で、LIN/UART制御器のデータ緩衝部は(読み込みで)空にされることが必要です('Lin\_get\_response()'関数参照)。

### 5.3.1. 原型

```
extern unsigned char lin_rx_response (unsigned char l_type, unsigned char l_len);
```

### 5.3.2. 関数

```
unsigned char lin_rx_response (unsigned char l_type, unsigned char l_len)
{
    if (l_type == LIN_1X) {
        Lin_1x_set_type(); // 変更が必要
    }
}
```



```

} else if (l_type == LIN_2X) {
    Lin_2x_set_type();           // 変更が必要
    Lin_set_rx_len(l_len);
} else {
    return 0;
}

Lin_rx_response();           // 命令設定
return 1;
}

```

### 5.3.3. 注意と備考

受信中に異常がないのを確認することが重要です。

### 5.4. LIN応答送信

この関数は主または従の節点(ノード)の従作業でのLIN応答の送出を命令します。

関数には3つの引数が渡されます。

1. 'unsigned char l\_type' : 構成により、'l\_type'は'LIN\_1X'または'LIN\_2X'のどちらかです。
2. 'unsigned long l\_data' : 内部SRAM配列ポインタ。この配列は送信するデータバイトを含みます。
3. 'unsigned long l\_len' : 応答で運ばれるデータバイト数、(符号化されない)真の長さ。この情報はそれが'l\_id'で符号化されるため、'LIN\_1X'で使われません。

関数は以下を返します。

'unsigned char' :       0 : 初期化失敗、LIN形式が一致しません。  
                  0以外 : 応答送信進行中

警告 : なし

#### 5.4.1. 原型

```
extern unsigned char lin_tx_response (unsigned char l_type, unsigned char *l_data, unsigned char l_len);
```

#### 5.4.2. 関数

```

unsigned char lin_tx_response (unsigned char l_type, unsigned char *l_data, unsigned char l_len)
{
    unsigned char i;

    if (l_type == LIN_1X) {
        Lin_1x_set_type();           // 変更が必要
    } else if (l_type == LIN_2X) {
        Lin_2x_set_type();           // 変更が必要
        Lin_set_tx_len(l_len);
    } else {
        return 0;
    }

    Lin_clear_index();               // データ処理
    for (i = 0; i < l_len; i++) {
        Lin_set_data(*l_data++);
    }

    Lin_tx_response();               // 命令設定
    return 1;
}

```

### 5.4.3. 注意と備考

送信中に異常がないのを確認することが重要です。

## 5.5. 受信データ読み込み

この関数はLIN応答が受信された時に受信データ緩衝部を読み(空に)します。この関数は'lin\_rx\_response()'関数の付加物です。関数には1つだけの引数が渡されます。

1. 'unsigned char \*l\_data' : 内部SRAM配列ポインタ。この配列は受信したデータバイトを含みます。

関数は何も返しません。

警告: なし

### 5.5.1. 原型

```
extern void lin_get_response (unsigned char *l_data);
```

### 5.5.2. 関数

```
void lin_get_response (unsigned char *l_data)
{
    unsigned char i, l_len;

    l_len = Lin_get_len();
    Lin_clear_index();
    for (i = 0; i < l_len; i++) {
        (*l_data++) = Lin_get_data();
    }
}
```

### 5.5.3. 注意と備考

(なし)

## 6. ファームウェア一括

2つのファイルが配給されます。

### 6.1. 'lin\_drv.h'ファイル

"lin\_drv.h"ファイルは主クロック周波数、使われるLINポーレート、それとデバイス特有I/Oヘッダファイルが定義される"config.h"ファイルをインクルードします。

"lin\_drv.h"ファイルでは以下を得ます。

1. 定義
  - 'ビット時間'定義
  - '構成設定'定義
2. C-マクロ
  - 'LIN 1.x' C-マクロ
  - 'LIN 2.x' C-マクロ
  - 共用マクロ
3. 関数の原型
  - 'lin\_init()'
  - 'lin\_tx\_header()'
  - 'lin\_rx\_response()'
  - 'lin\_tx\_response()'
  - 'lin\_get\_response()'

### 6.2. 'lin\_drv.c'ファイル

"lin\_drv.c"ファイルは"lin\_drv.h"ファイルをインクルードします。

"lin\_drv.c"ファイルでは以下のC-関数得るだけです。

1. 'lin\_init()'
2. 'lin\_tx\_header()'
3. 'lin\_rx\_response()'
4. 'lin\_tx\_response()'
5. 'lin\_get\_response()'



## 7. LIN通信管理

この節はポーリング法に基づくLIN通信管理を示します。割り込みに基づく管理はそれ自身のLINメッセージオブジェクト記述子を構築するのに必要です。しかし下で検討される多くの項目は割り込み法が望まれる場合に再使用することができます。

### 7.1. LIN状態/割り込みレジスタの使い方

LIN通信を管理するには、LIN状態/割り込みレジスタ(LINSIR)の読み込みだけが必要です。このレジスタが全ての割り込み元を収容することに注意してください。このレジスタを考察してください。

7	6	5	4	3	2	1	0	
LIDST2	LIDST1	LIDST0	LBUSY	LERR	LIDOK	LTXOK	LRXOK	LINSIR

4つの事象だけが有り得ます。

1. ID送信完了/ID受信：ID(LIDOK)フラグが挙がると、これは2つのことを意味します(両方共に重要)。
  - ・ LIN先頭部段階が実行され、先頭部送信が完了された(主装置動作)、または先頭部が受信された(従装置動作)。
  - ・ 節点(ノード)が受信したLIN IDに関係するなら、可能な限り素早くLIN応答段階が起こされなければなりません。
2. 応答受信完了：Rx(LRXOK)フラグが挙がると、これは今やLIN応答受信が完了されて、異常が全く検出されなかったことを意味します。
3. 応答送信完了：Tx(LTXOK)フラグが挙がると、これは今やLIN応答送信が完了されて、異常が全く検出されなかったことを意味します。
4. 異常：異常(LERR)フラグが挙がると、この発生源はLIN異常レジスタ(LINERR)に含まれます。それは異常が起きる時に注意することが重要です。異常の訳は度々それに依存します。

### 7.2. LIN従装置例

```
//-----
// 表題 : lin_slave_example.c
//-----

//----- インクルード
#include "config.h"
#include "lin_drv.h"

//----- 宣言
#define LIN_ID_0    0x12
#define LIN_ID_1    0x13

#define LEN_0       1
#define LEN_1       2

unsigned char lin_motor_control[LEN_0];
unsigned char lin_node_status[LEN_1];

volatile unsigned char lin_slave_err_nb = 0;

//----- 関数

//. . . . .
//
// "lin_slave_example.c"のlin_id_task関数
//
// 受信したLIN IDは検査されて節点(ノード)が管理しなければならないものと比較されなければなりません。
//
void lin_id_task (void)
{
    switch (Lin_get_id()) {
        case LIN_ID_0:
            lin_rx_response(LIN_2X, LEN_0);
            break;
        case LIN_ID_1:
            lin_tx_response(LIN_2X, lin_node_status, LEN_1);
    }
}
```

```

        break;
default:
    // ID: 不在/拒否
    break;
}
}

```

```

//. . . . .
//
// "lin_slave_example.c"のlin_rx_task関数
//
// - 応答データ保存
// - 応用信号更新
//
void lin_rx_task (void)
{
    unsigned char l_temp;

    lin_get_response (lin_motor_contol);
    // 応用信号の更新
    l_temp = lin_motor_contol[0] & 0x03;
    if((l_temp == 0x01) || (l_temp == 0x03)) {
        PORTB |= 1<<PORT0;
    } else {
        PORTB &= ~(1<<PORT0);
    }
    if(l_temp == 0x02) {
        PORTB |= 1<<PORT1;
    } else {
        PORTB &= ~(1<<PORT1);
    }
}

//. . . . .
//
// "lin_slave_example.c"のlin_tx_task関数
//
// - 次の送信の準備を整えるために応用信号で緩衝部配列を更新
//
void lin_tx_task (void)
{
    // 応用信号の更新
    lin_node_status[0] = PINB & 0x03;
    lin_node_status[1] = lin_slave_err_nb;
}

//. . . . .
//
// "lin_slave_example.c"のlin_err_task関数
//
// - LIN異常なら、節点(ノード)異常番号を増加
//
void lin_err_task (void)
{
    // 全域変数更新
    lin_slave_err_nb++;
}

```

```

//. . . m a i n ( ) . . . . .
//
// "lin_slave_example.c"のmain関数
//
// '終了無し'で以下を行い繰り返す:
// - LIN_ID_0(受信応答1バイト)なら
//   • PORT-B.0 = DC電動機命令⇒時計回り
//   • PORT-B.1 = DC電動機命令⇒反時計回り
// - LIN_ID_1(送信応答2バイト)なら
//   • Byte[0] = 電動機状態
//   • Byte[1] = 節点(ノード)異常番号
//
int main (void)
{
    // 電動機制御用ポートB設定
    DDRB |= 1<<PORTB0; DDRB |= 1<<PORTB1;
    PORTB &= ~(1<<PORTB0); PORTB &= ~(1<<PORTB1);

    // LIN初期化
    lin_init((unsigned char)LIN_2X, ((unsigned short)CONF_LINBRR));

    // 終了無き繰り返し
    while (1) {
        switch (Lin_get_it()) {
            case LIN_IDOK:
                lin_id_task();
                Lin_clear_idok_it();
                break;
            case LIN_RXOK:
                lin_rx_task();
                Lin_clear_rxok_it();
                break;
            case LIN_TXOK:
                lin_tx_task();
                Lin_clear_txok_it();
                break;
            case LIN_ERROR:
                lin_err_task();
                Lin_clear_err_it();
                break;
            default:
                break;
        }
    }
    return 0;
}

```

プログラムの大きさ(コンパイラ任意選択:-Os) : コード=816 フラッシュ バイト、データ=4 RAMバイトです。

### 7.3. LIN主装置例

先頭部を送るために実時間作業が追加されなければなりません。例えば計時器0がそれらを送るのに使われます。さもなければプログラムはLIN従装置例のように見えます。

```
//-----
// 表題 : lin_master_example.c
//-----

//----- インクルード
#include "config.h"
#include "lin_drv.h"
#include <avr/interrupt.h>           // AVR-GCCライブラリ使用

//----- 宣言
#define LIN_ID_0    0x12
#define LIN_ID_1    0x13

#define LEN_0       1
#define LEN_1       2

#define MAGIC_NUMBER    77

unsigned char lin_motor_command[LEN_0];
unsigned char lin_slave_node_status[LEN_1];

volatile unsigned char lin_master_err_nb = 0;
volatile unsigned char rtc_tics = 0;

//----- 関数

//. . . . .
//
// "lin_master_example.c"のlin_id_task関数
//
// 受信したLIN IDは調査されて節点(ノード)が管理しなければならないものと比較されなければなりません。
//
void lin_id_task (void) {
    switch (Lin_get_id()) {
        case LIN_ID_0:
            lin_tx_response(LIN_2X, lin_motor_command, LEN_0);
            break;
        case LIN_ID_1:
            lin_rx_response(LIN_2X, LEN_1);
            break;
        default:
            // ID: 不在/拒否
            break;
    }
}

//. . . . .
//
// "lin_master_example.c"のlin_rx_task関数
//
// - 応答データ保存
// - 応用信号更新
//
```

```

void lin_rx_task (void)
{
    lin_get_response (lin_slave_node_status);

    // 従節点(ノード)に於いて命令がOKで異常がなければ
    // - PORTB[6] = 0
    // - さもなければ、PORTB[6] = 1
    if (lin_slave_node_status[0] != lin_motor_command[0]) {
        PORTB |= 1<<PORT6;
    } else if (lin_slave_node_status[1] != 0) {
        PORTB |= 1<<PORT6;
    } else {
        PORTB &= ~(1<<PORT6);
    }
}

//. . . . .
//
// "lin_master_example.c"のlin_tx_task関数
//
// - 次の送信の準備を整えるために応用信号で緩衝部配列を更新
//
void lin_tx_task (void)
{
    // 応用信号(Low活性スイッチ)の更新
    lin_motor_command[0] = (~PINB) & 0x03;
}

//. . . . .
//
// "lin_master_example.c"のlin_err_task関数
//
// - LIN異常なら、節点(ノード)異常番号を増加
//
void lin_err_task (void)
{
    // 全域変数更新
    lin_master_err_nb++;
}

//. . . . .
//
// "lin_master_example.c"のOCR0A割り込み処理ルーチン
//
// AVR-GCC宣言(既定):
//     ISR(TIMERO_COMPA_vect)
// IAR宣言:
//     #pragma vector TIMERO_COMPA_vect
//     __interrupt void timer0_compa_vect(void)
//
// rtc_tic値に従って適切なLIN先頭部が送られます。
//
ISR(TIMERO_COMPA_vect)
{
    rtc_tics++;

    if((rtc_tics & 0x01) == 0x01) {
        lin_tx_header((unsigned char)LIN_2X, (unsigned char)LIN_ID_0, 0);
    } else {
        lin_tx_header((unsigned char)LIN_2X, (unsigned char)LIN_ID_1, 0);
    }
}

```

```

//. . . m a i n ( ) . . . . .
//
// "lin_master_example.c"のmain関数
//
// '終り無し'で以下を行い繰り返す:
// - LIN_ID_0(送信応答1バイト)なら
//   ・ PORT-B.0 = DC電動機命令⇒時計回り
//   ・ PORT-B.1 = DC電動機命令⇒反時計回り
// - LIN_ID_1(受信応答2バイト)なら
//   ・ Byte[0] = 電動機状態
//   ・ Byte[1] = 従節点(ノード)異常番号
// 計時器0は各線頭部が20ms間隔で送られることを保証します。
//
int main (void)
{
    // 電動機制御用ポートB設定
    // PORTB0とPORTB1はに内部プルアップとでLow活性を切り換えます。
    DDRB &= ~(1<<PORTB0); DDRB &= ~(1<<PORTB1);
    PORTB |= 1<<PORTB0; PORTB |= 1<<PORTB1;

    // 電動機状態フラグ用ポートB設定
    DDRB |= 1<<PORTB6; PORTB &= ~(1<<PORTB6);

    // LIN初期化
    lin_init((unsigned char)LIN_2X, ((unsigned short)CONF_LINBRR));

    // 計時器0と割り込みの初期化(出力なし)
    // 計時器0セット
    TIMSK0 = 0; TCCR0B = 0; TCCR0A = 0; TCNT0 = 0; OCR0A = 0;
    TIFR0 = ((1<<OCF0A) | (1<<TOV0));
    // 出力なし、動作形態2、10ms割り込み
    TCCR0A = 0x02; OCR0A = MAGIC_NUMBER; TCCR0B = 0x05;
    // 計時器0比較A割り込み許可
    TIMSK0 |= 1<<OCIE0A; asm ("sei");

    // 終り無き繰り返しNo End Loop
    while (1) {
        switch (Lin_get_it()) {
            case LIN_IDOK:
                lin_id_task();
                Lin_clear_idok_it();
                break;
            case LIN_RXOK:
                lin_rx_task();
                Lin_clear_rxok_it();
                break;
            case LIN_TXOK:
                lin_tx_task();
                Lin_clear_txok_it();
                break;
            case LIN_ERROR:
                lin_err_task();
                Lin_clear_err_it();
                break;
            default:
                break;
        }
    }
    return 0;
}

```

プログラムの大きさ(コンパイラ任意選択:-Os) : コード=946 フラッシュ バイト、データ=5 RAMバイトです。

#### 7.4. 構成設定ヘッダ ファイル

```
//-----  
// "lin_slave_example.c"または"lin_master_example.c"用config.h  
//-----  
  
//----- インクルード  
#include <avr/io.h> // AVR-GCCライブラリ使用  
  
//----- 宣言  
// "lin_master_example.c"に対して外部クリスタル発振器を使用  
#define FOSC 8000 // kHzで  
#define LIN_BAUDRATE 19200 // ビット/秒で
```





## 本社

### *Atmel Corporation*

2325 Orchard Parkway  
San Jose, CA 95131  
USA  
TEL 1(408) 441-0311  
FAX 1(408) 487-2600

## 国外営業拠点

### *Atmel Asia*

Unit 1-5 & 16, 19/F  
BEA Tower, Millennium City 5  
418 Kwun Tong Road  
Kwun Tong, Kowloon  
Hong Kong  
TEL (852) 2245-6100  
FAX (852) 2722-1369

### *Atmel Europe*

Le Krebs  
8, Rue Jean-Pierre Timbaud  
BP 309  
78054 Saint-Quentin-en-  
Yvelines Cedex  
France  
TEL (33) 1-30-60-70-00  
FAX (33) 1-30-60-71-11

### *Atmel Japan*

104-0033 東京都中央区  
新川1-24-8  
東熱新川ビル 9F  
アトメル ジャパン株式会社  
TEL (81) 03-3523-3551  
FAX (81) 03-3523-7581

## 製品窓口

### ウェブサイト

[www.atmel.com](http://www.atmel.com)

### 技術支援

[avr@atmel.com](mailto:avr@atmel.com)

### 販売窓口

[www.atmel.com/contacts](http://www.atmel.com/contacts)

### 文献請求

[www.atmel.com/literature](http://www.atmel.com/literature)

お断り: 本資料内の情報はAtmel製品と関連して提供されています。本資料またはAtmel製品の販売と関連して承諾される何れの知的所有権も禁反言あるいはその逆によって明示的または暗示的に承諾されるものではありません。Atmelのウェブサイトに位置する販売の条件とAtmelの定義での詳しい説明を除いて、商品性、特定目的に関する適合性、または適法性の暗黙保証に制限せず、Atmelはそれらを含むその製品に関連する暗示的、明示的または法令による如何なる保証も否認し、何ら責任がないと認識します。たとえばAtmelがそのような損害賠償の可能性を進言されたとしても、本資料を使用できない、または使用以外で発生する(情報の損失、事業中断、または利益の損失に関する制限なしの損害賠償を含み)直接、間接、必然、偶然、特別、または付随して起こる如何なる損害賠償に対しても決してAtmelに責任がないでしょう。Atmelは本資料の内容の正確さまたは完全性に関して断言または保証を行わず、予告なしでいつでも製品内容と仕様の変更を行う権利を保留します。Atmelはここに含まれた情報を更新することに対してどんな公約も行いません。特に別の方法で提供されなければ、Atmel製品は車載応用に対して適当ではなく、使用されるべきではありません。Atmel製品は延命または生命維持を意図した応用での部品としての使用に対して意図、認定、または保証されません。

© Atmel Corporation 2008. 不許複製 Atmel®、ロコとそれらの組み合わせ、AVR®とその他はAtmel Corporationの登録商標または商標またはその付属物です。他の用語と製品名は一般的に他の商標です。

© HERO 2021.

本応用記述はAtmelのAVR286応用記述(doc8123.pdf Rev.8123A-03/08)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意識されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はリンクとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。