本ドキュメントは Cypress (サイプレス) 製品に関する情報が記載されております。本ドキュメントには、「MB」から始まるシリーズ名，品名およびオーダ型格が記載されておりますが、これらはすべて「CY」から始まるシリーズ名，品名およびオーダ型格として、新規および既存のお客様に引き続き提供してまいります。

オーダ型格の調べ方について

1. www.cypress.com/pcnにアクセスしてください。
2. SEARCH PCNS フィールドに、オーダ型格などのキーワードを入力し、「Apply」をクリックしてください。
3. 該当するタイトル(Title)をクリックしてください。
4. 「Affected Parts List」ファイルを開いてください。
   当該ファイルに記載されている各種変更情報をご利用ください。

詳しいお問い合わせ先

Cypress 製品およびそのソリューションの詳細につきましては、お近くの営業所へお問い合わせください。

サイプレスについて

サイプレスは、世界で最も革新的な車載や産業機器、スマート家電、民生機器および医療機器製品向けに、最先端の組み込みシステム ソリューションを提供するリーディング カンパニーです。サイプレスのマイクロコントローラーや、アナログ IC、ワイヤレスおよび USB ベースのコネクティビティ ソリューション、高い信頼性と高性能を提供するメモリ製品は、各種機器メーカーの差異化製品の開発と早期市場参入を支援します。サイプレスは、ベストクラスのサポートと開発リソースをグローバルに提供することで、彼らが従来市場を破壊しまったく新しい製品カテゴリを歴史的なスピードで市場投入できるよう支援します。詳細はサイプレスのウェブサイト (japan.cypress.com) をご覧ください。

AN204336

16-bit Microcontroller F²MC-16FX ファミリ MB96600 シリーズ LIN の使用方法

関連製品ファミリ	シリーズ	製品
F²MC-16FX ファミリ	MB96610	MB96F612R, MB96F612A, MB96F613R, MB96F613A, MB96F615R, MB96F615A
	MB96620	MB96F622R, MB96F622A, MB96F623R, MB96F623A, MB96F625R, MB96F625A
	MB96630	MB96F633R, MB96F633A, MB96F635R, MB96F635A, MB96F636R, MB96F637R
	MB96640	MB96F643R, MB96F643A, MB96F645R, MB96F645A, MB96F646R, MB96F647R
	MB96650	MB96F653R, MB96F653A, MB96F655R, MB96F655A, MB96F656R, MB96F657R
	MB96670	MB96F673R, MB96F673A, MB96F675R, MB96F675A
	MB96680	MB96F683R, MB96F683A, MB96F685R, MB96F685A
	MB96690	MB96F693R, MB96F693A, MB96F695R, MB96F695A, MB96F696R
	MB966A0	MB96F6A5R, MB96F6A5A, MB96F6A6R
	MB966B0	MB96F6B5R, MB96F6B5A, MB96F6B6R
	MB966C0	MB96F6C5R, MB96F6C5A, MB96F6C6R

本アプリケーションノートでは、F²MC-16FX Family MB96600 シリーズの LIN の使用方法を記述します。

Contents

1 はじめに 1
2 LIN 概要 1
2.1 LIN とは 1
2.2 LIN の仕様 4
2.3 LIN 通信の流れ 6
2.4 エラー発生時のマスタとスレーブの通信 7
3 ハードウェア環境概要 8
3.1 ハードウェア環境詳細 8
4 ソフトウェア概要 11
4.1 LIN ドライバ概要 11
4.2 LIN ドライバ設定 14
4.3 LIN マスタサンプルプログラム 15
4.4 LIN スレーブサンプルプログラム 28
改版履歴 39
セールス, ソリューションおよび法律情報 40

1 はじめに

このアプリケーションノートは F²MC-16FX を使用した LIN マスタと LIN スレーブの制御方法を記述しています。LIN 制御を行う様々なアプリケーションに適用することができます。

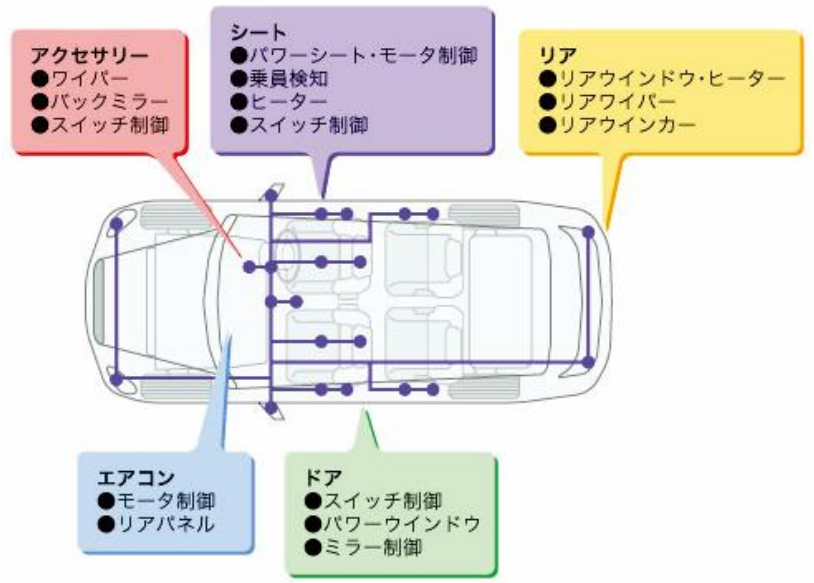
2 LIN 概要

2.1 LIN とは

LIN とは Local Interconnect Network (ローカル・インターコネクト・ネットワーク) の略称で、車載 LAN の通信プロトコルの一種です。制御系車載 LAN として最も普及している CAN よりも安価に構築できることを目的として、1999 年に LIN コンソーシアムが提唱しました。その後、数回のバージョンアップを重ねたのち、診断機能などを追加した LIN2.0 が 2003 年に発表されました。また 2006 年には LIN2.1 にバージョンアップしています。

LIN の用途について説明していきます。車の多機能化に伴い、車の中にもネットワークが欠かせない存在になってきました。現在、車載 LAN は、走行や車体に関する制御系およびカーナビやオーディオなどの機器をつなぐ情報系の 2 種類に大別され、用途に応じて異なる車載 LAN が使用されています。とりわけ、ボディ系に分類される電動ミラーやパワーウィンドウといった車体装備は、それほど高速できめ細かい制御までは必要とされません。その分、コストもかけられません。こういった箇所に LIN が用いられています。

図 1 車の LIN アプリケーション例



このようにして用いられる LIN の特徴を 5 つのポイントにまとめて紹介します。

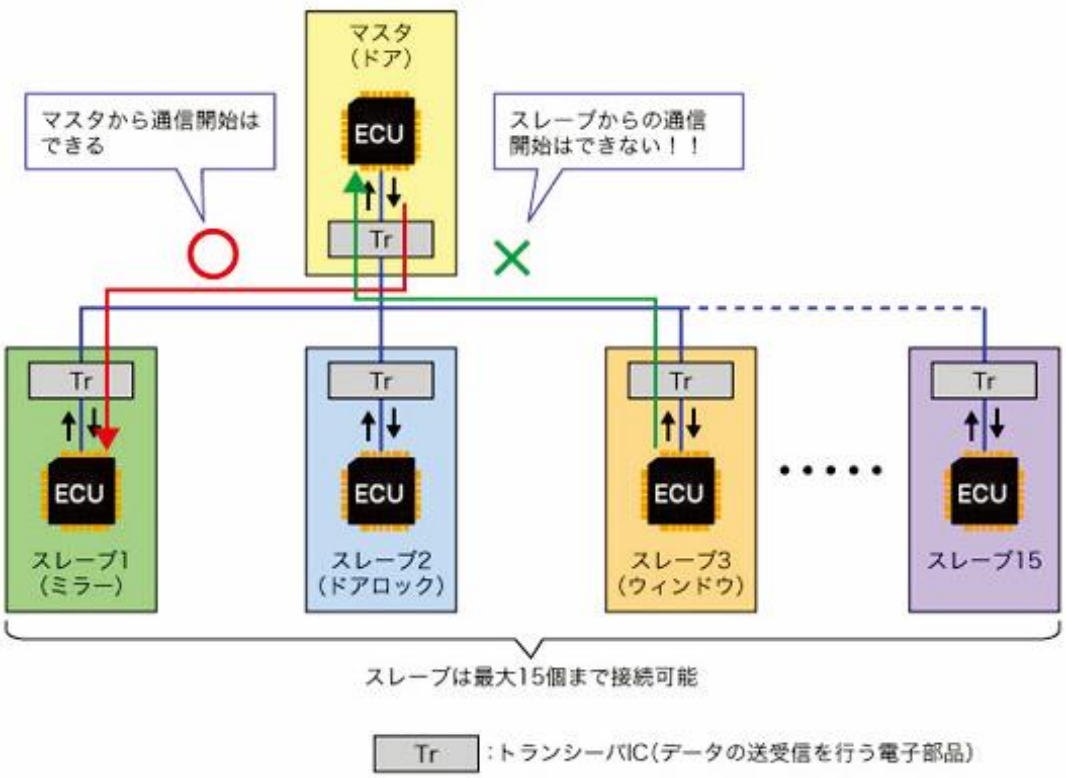
① シングルマスタ通信である

LIN には 2 種類の通信ノードがあります。1 つは「マスタ」(送り元) です。すべての通信開始の管理を行います。もう 1 つがスレーブ (送り先) です。マスタが出す命令に対して応答します。LIN 通信は必ずマスタから始まり、スレーブから通信を開始することはありません。そして LIN ではマスタとなる通信ノードは決まっています。この方式を「シングルマスタ方式」と呼びます。

② 最大 15 ノードのスレーブをバス型配線でつなぐ

LIN のネットワーク構成 (トポロジ) はバス型です。シングルマスタ方式の LIN では、スレーブはマスタから命令が来たときのみ通信を行うため、バス上で信号が衝突することはありません。1 つのマスタに対し、最大 15 ノードまでスレーブを接続することができます。

図 2 LIN の主なネットワーク構成



③　配線はワイヤ 1 本ですむ

車内の各 ECU は、トランシーバ IC (データの送受信を行う電子部品) を介して LIN ネットワークに接続され、マスタからスレーブまで各 ECU はバス接続されます。このバスのケーブルとしては、ごく普通のメタル線 1 本が用いられます。CAN はメタル線 2 本をより合わせて 1 対とした、より対線を使います。FlexRay は 2 本のより対線を用います。このため LIN では、より対線を使用する CAN や FlexRay と比べて、ネットワークの配線数が 1 本ですむというメリットがあります。

通信距離は最長 40 m です。LIN は、CAN と組み合わせて使われることがありますが、その場合、CAN は基幹ネットワークとして、LIN はその支線ネットワークとして使われることが多いです。

④　通信速度は最大 20 k ビット/秒

LIN の仕様で通信速度は、1～20k ビット/秒の範囲内で規定されています。実際に車内 LAN として使用される LIN の通信速度は、各自動車メーカーのシステム仕様により異なりますが、2400 ビット/秒, 9600 ビット/秒, 19200 ビット/秒のいずれかが一般的です。

⑤　通信エラーは検出のみ、後の処理はアプリケーション依存

LIN では、送受信が正常に行われたかどうかの情報を基に通信エラーを検出しています。ただしエラー検出後の処理についての規定がありません。そこで LIN は、アプリケーションに合わせてエラー処理を任意に設計します。CAN や FlexRay はエラーカウンタとよばれるカウンタの値によって通信状態を管理することが仕様で決められていますが、LIN では、エラーが発生した場合は、単に次の命令を待つといったシンプルなエラー処理をすることも可能です。

2.2 LIN の仕様

ここでは LIN の仕様について簡単に説明します。

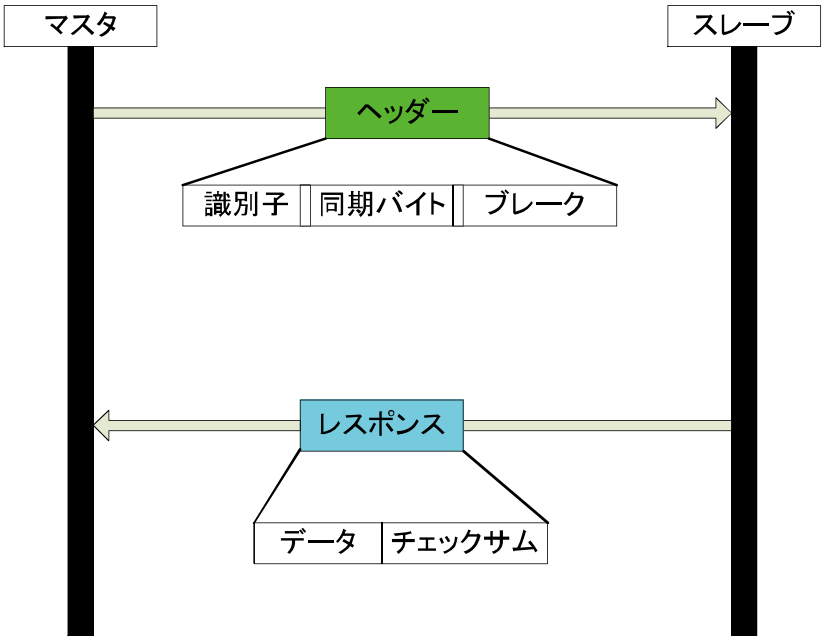
詳細な仕様について知りたい場合は、LIN コンソーシアムのホームページ (http://www.lin-subbus.org/) にアクセスしていただき、氏名やメールアドレスなどを登録のうえ、仕様書を入手してください。

2.2.1 フレーム構成について

LIN の基本的な通信の単位であるフレームについて説明します。

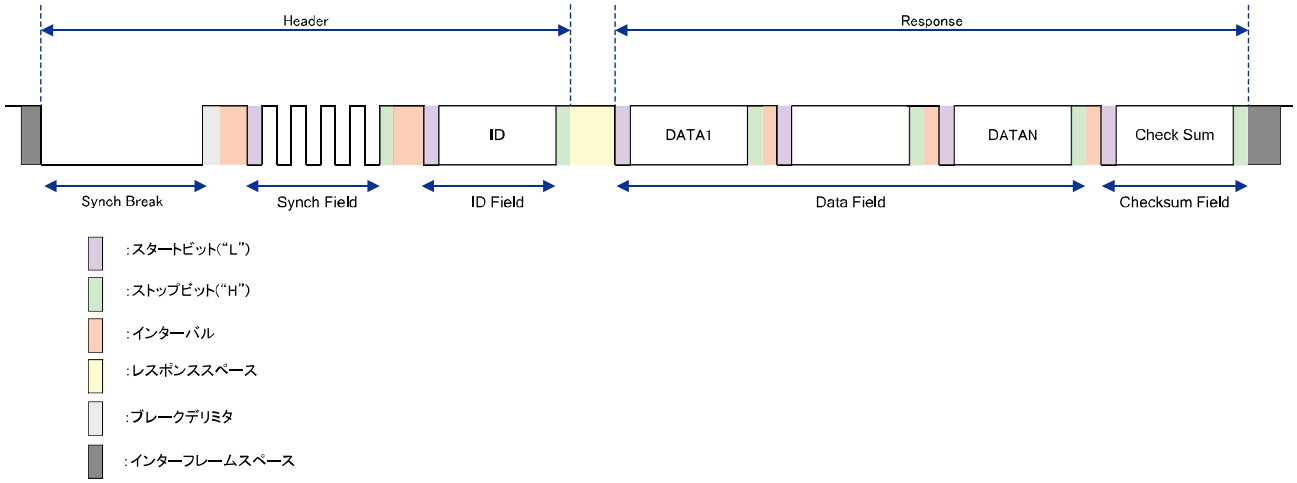
LIN のフレームは「ヘッダー」と「レスポンス」によって構成されています。基本的な通信の流れとしては、図 3 LIN 通信の基本的なフローのようにマスタがスレーブにヘッダーを送信し、送られてきたヘッダーの内容に従ってスレーブは処理を行い、マスタにレスポンスを送るという手順になります。

図 3 LIN 通信の基本的なフロー



ヘッダーはさらに、Sync break (ブレーク), Sync field (同期バイト), ID field (識別子) の 3 つのフィールドから、レスポンスは Data field (データ), Checksum field (チェックサム) の 2 つのフィールドで構成されます。

図 4 LIN のフレーム構成



1. Sync break (ブレーク)

ヘッダーの先頭領域にある Sync break は、新しいフレームの開始を示す可変長のフィールドです。最少 13～16 ビットの “0” (固定値ゼロ) から成ります。一般的なブレーク長は 13 ビットです。

2. Sync field (同期バイト)

ブレークに続く Sync field は、マスタとスレーブが同期をとるための 10 ビット長の固定長のフィールドです。同期バイトの構成は、1 ビットのスタートビット (“0”), 8 ビットのデータビット, 1 ビットのストップビット (“1”) から成ります。8 ビットのデータビットは、固定値 “0x55” (2 進数表現では“0x01010101”) です。マスタが送ってきた同期バイト中の 0x55 をスレーブが正常に受信できていれば同期がとられます。

3. ID field (識別子)

同期バイトに続いてヘッダーの最終領域にある「識別子」は、フレームの種類や目的を指定する 10 ビット固定長のフィールドです。識別子は “0”～“63” (6 ビット) の値をとります。この識別子は、マスタが個々のスレーブを指定するためにも使います。スレーブはマスタから送られてきた識別子の値によって、どのタイプのフレームが送られてきたのか、それが自分当てかどうなのかを判断して、それに応じたレスポンスをマスタに返します。なお、識別子のフィールドは、“0”～“63” (6 ビット) に続いて、2 ビットのパリティビットがあります。その前後に同期バイトと同様に、1 ビットのスタートビットと 1 ビットのストップビットがあるため、全体で 10 ビット長となります。

4. Data field (データ)

レスポンスの先頭にある「データ」は、文字通りデータを転送する可変長のフィールドです。

あらかじめ取り決められたバイト数 (1～8 バイト) のデータを伝えます。ヘッダーの同期バイトと同様に、1 ビットのスタートビットと 1 ビットのストップビットが 1 バイトのデータの前後にあるため、1 バイトデータにつき 10 ビットで構成されます。このため、データの全フィールド長は「バイト数×10 ビット」となります。

5. Checksum field (チェックサム)

データに続く「チェックサム」は、データ確認用の 10 ビット固定長のフィールドです。データ受信側は送られてきたデータとチェックサムを比べて、データに誤りがないかどうかを確認します。チェックサムのフィールド長は、これも同期バイトと同様に 8 ビットのチェックサムにスタートビットおよびストップビットが加わって 10 ビットとなります。

2.3 LIN 通信の流れ

一般的な LIN 通信では、1 つのマスタが複数個のスレーブと通信します。バス型のトポロジをとる LIN は、1 本のワイヤにマスタと全てのスレーブが接続するため、マスタが送ったヘッダーの電気信号はワイヤを伝わって全てのスレーブに届きます。各スレーブはヘッダーの識別子 (ID) をみて、自身にあてたヘッダーであれば、その内容に従ったレスポンスをマスタに返します。送られてきたヘッダーが他のスレーブ当てなら、無視します。こうすることで、マスタと各スレーブは 1 対 1 の通信を実現しています。

実際の通信のやりとりを説明していきます。いま、スレーブ 1 からスレーブ 15 まで、それぞれに機能が割り当てられています。マスタはまずスレーブ 1 との通信を行ってモータを回し (図 5 LIN の主なネットワーク構成および図 6 通常通信時のマスタとスレーブとの通信シーケンス例の (1))、次にスレーブ 3 との通信を行ってセンサの情報を取得します (同 (2))。その後、スレーブ 2 と通信してモータを回します (同 (3))。マスタは再びスレーブ 3 からセンサ情報を取得し (同 (4))、最後にスレーブ 15 との通信でランプを点灯させます (同 (5))。この一連の通信では、マスタとスレーブ 3、マスタとスレーブ 2 の通信は連動していて、マスタはスレーブ 3 との通信で取得したセンサ情報を基に、次のスレーブ 2 との通信でモータを回す処理を行っています。このように実際の通信では、マスタと複数のスレーブが 1 対 1 の通信を繰り返しています。

図 5 LIN の主なネットワーク構成

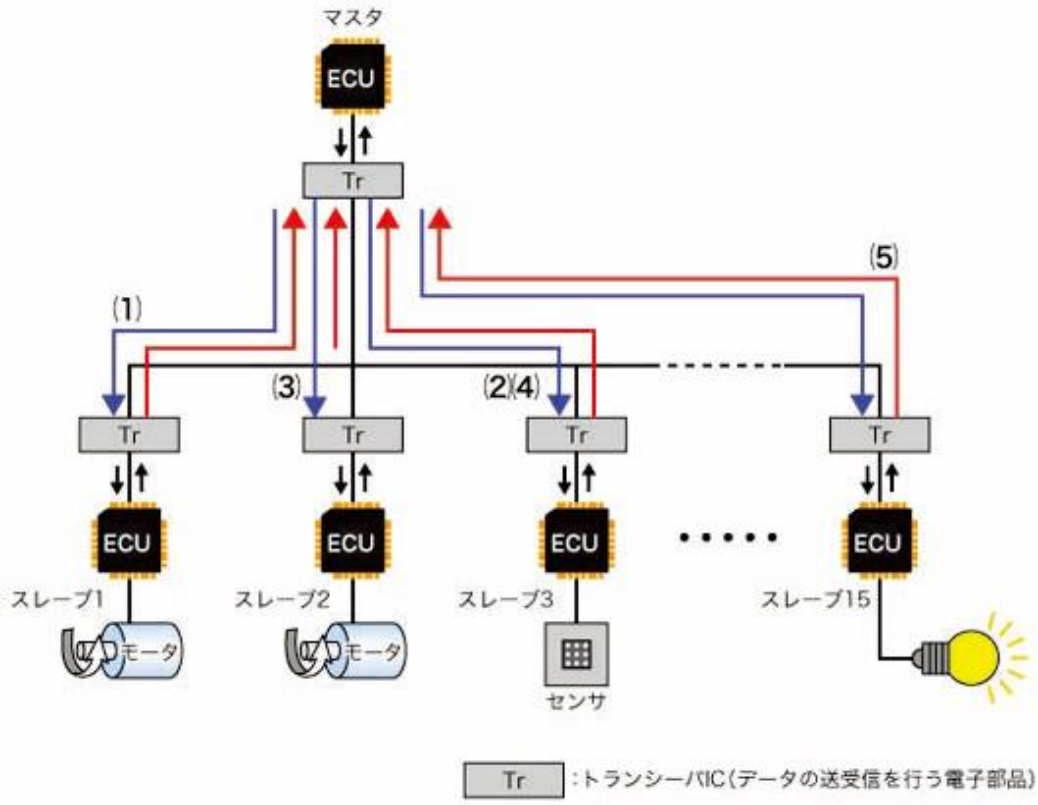
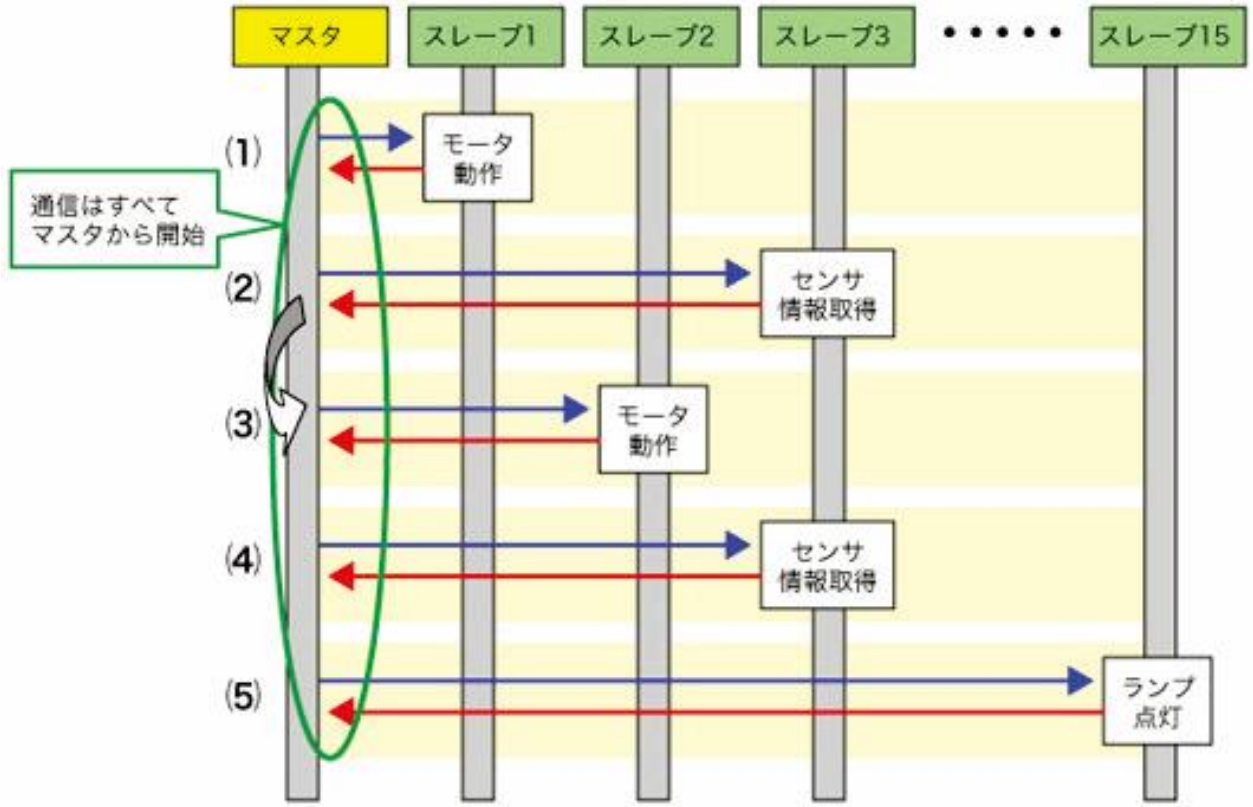


図 6 通常通信時のマスタとスレーブとの通信シーケンス例



2.4 エラー発生時のマスタとスレーブの通信

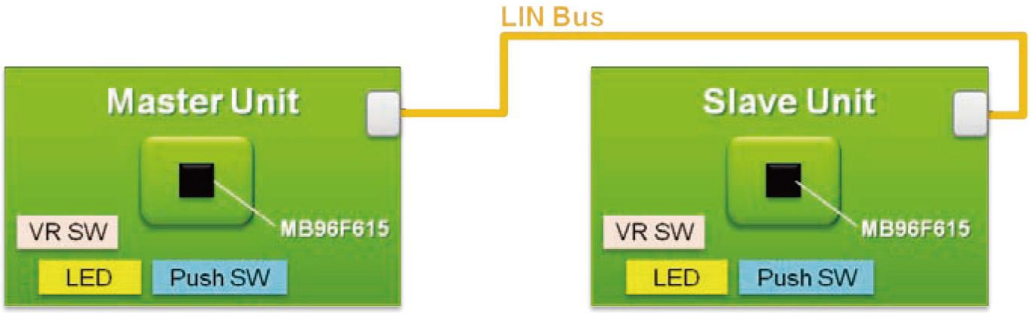
LIN のエラー管理は、プロトコルによる決まりがないため、アプリケーションに依存します。そのため、設計時に、エラーの検出方法やエラー処理後の処理を考慮する必要があります。ただし LIN の仕様にはプロトコルによる決まりこそありませんが、エラー発生時におけるシステム設計の例が「Status Management」の章で紹介されています。その紹介例では、スレーブが自身のステータスをマスタに報告することでエラー管理を行っています。その仕組みは次のようになっています。

基本的なマスタの動作としては、現在のスレーブとの通信が終わったら、次のスレーブにヘッダーを送信するだけです。一方、スレーブの動作としてはヘッダーを受信時とレスポンス送信時にエラーチェックを行います。受信時にはチェックサムなどをチェックします。送信時には、送信データとモニタリングしているバスのデータを見比べてチェックしています。こうすることで、自身のステータスを認識し、結果をレスポンスの中に入れてマスタに送信します。マスタはレスポンスによりスレーブの状態を知り、不具合があればスレーブの初期化を行います。こうすることで、エラー状態を完全にクリアします。

3 ハードウェア環境概要

以下に LIN 通信用環境の構成を示します。マスタユニット、スレーブユニットともに、入力用に Push SW および Volume SW、出力用に LED をそれぞれ搭載しています。LIN サンプルソフトでは、これらの SW の入力情報を LIN バス経由で対向ユニットに送信し、LED 制御を行います。

図 7 LIN 通信ハードウェア概要

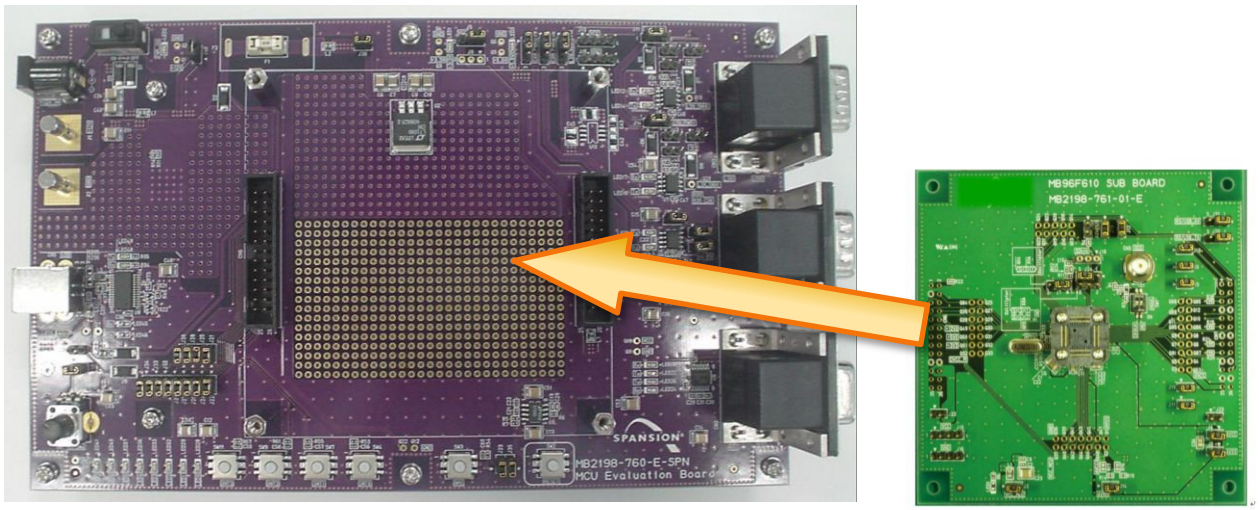


3.1 ハードウェア環境詳細

LIN 通信制御には、以下の評価環境を使用します。

- サイプレス製 MCU MB96F615
- サイプレス製 MB96600 シリーズ評価用メインボード：MB2198-760-E
- サイプレス製 MB96610 シリーズ評価用ドーターボード：MB2198-761-01-E

図 8 LIN 通信ハードウェア環境 (MB2198-760-E (左図), MB2198-761-01-E (右図))



3.1.1 LIN/IF

MB96600 シリーズ評価用メインボードには LIN I/F が 2 チャネル搭載されています。LIN_PWR ジャンパ設定により、LIN トランシーバ (MAX13020) の VBAT 供給元を内部電源か外部電源か選択できます。(デフォルトは内部電源設定です。) また、ユニットをマスタノードとして使用する場合は、JP46 をショートさせる必要があります。LIN I/F 回路構成について以下に示します。

図 9 LIN I/F 回路構成

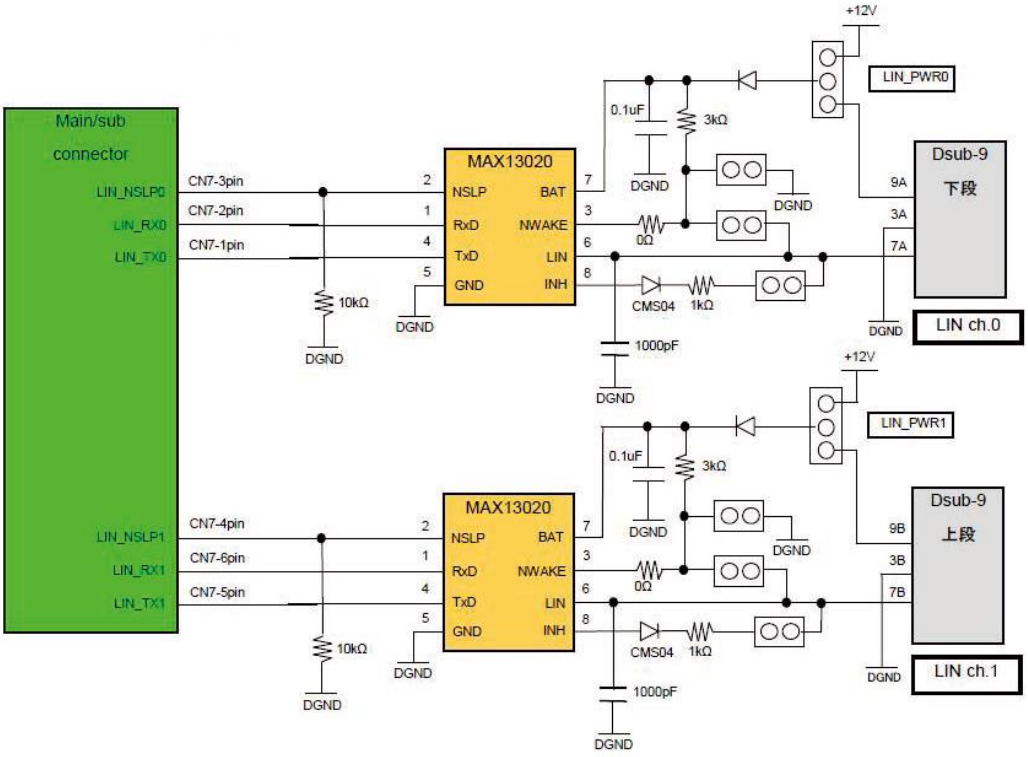
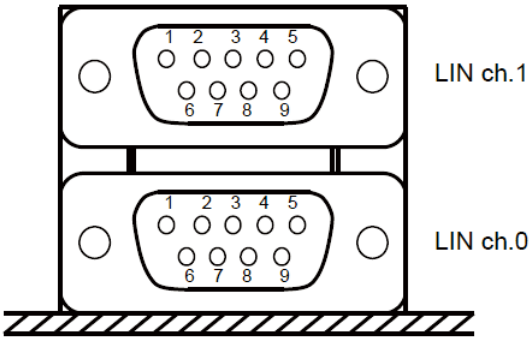


図 10 LIN I/F ピン配置 (D-SUB9)

ピン番号	信号	機能
1	-	-
2	-	-
3	GND	共通グランド
4	-	-
5	-	-
6	-	-
7	LIN	LIN bus I/O
8	-	-
9	VCC	VBAT 電源供給

Note: ”-“ は解放(未接続)を意味します。

図 11 LIN I/F コネクタ

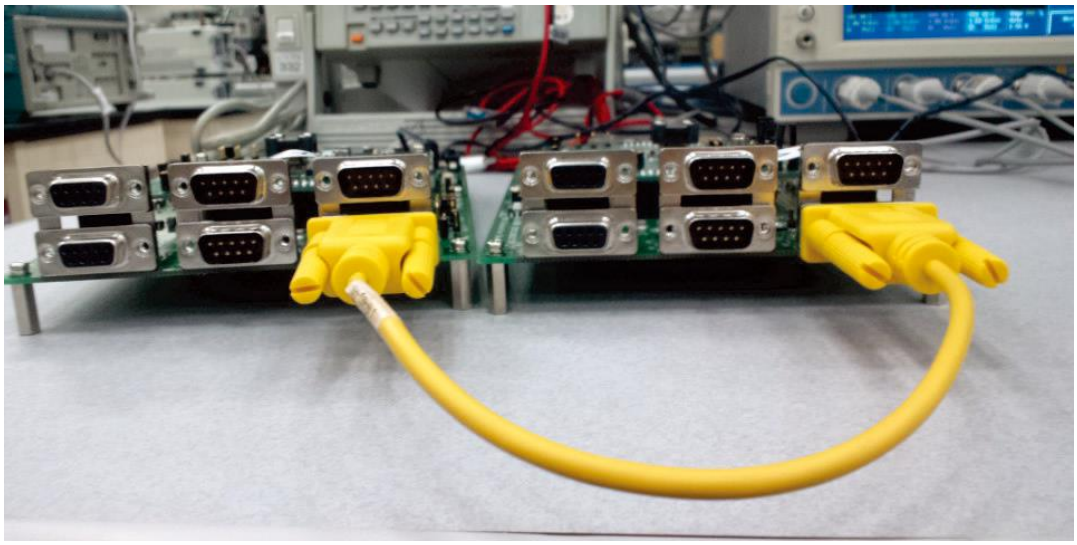


実際の LIN 通信環境を以下に示します。今回使用する LIN 通信サンプルソフトでは、LINUSART のチャネル 7 を使用します。MB96610 シリーズの評価環境では、LIN-USART チャネル 7 はメインボードの LIN ch0 (図 11 LIN I/F コネクタの下部) に接続されています。

図 12 LIN 通信環境



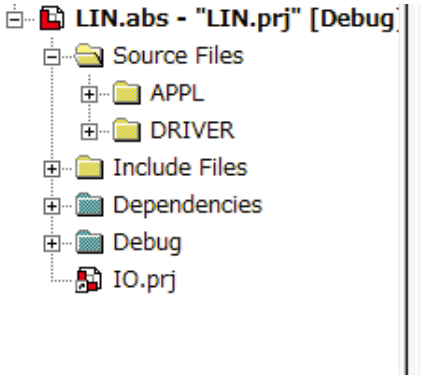
図 13 LIN コネクタ接続



4 ソフトウェア概要

LIN サンプルソフトウェアは、主に 2 つの階層に分かれています。1 つ目が LIN 通信制御を行う LIN ドライバ部、2 つ目がアプリケーション部です。

図 14 プロジェクトフォルダ



4.1 LIN ドライバ概要

本 LIN ドライバでは、以下のようなことが実現可能です。

- LIN マスタ／スレーブの切り替えが可能
- ボーレートの変更が可能 (2400／9600 (初期値)／19200bps)
- 容易に LIN メッセージの追加が可能
- タイマを使った Sync field での同期制御 (LIN スレーブのみ)
- Wake up 出力 (LIN スレーブのみ)
- LIN マスタ⇔LIN スレーブ間の SW 制御, LED 制御, アナログ制御

また、本サンプルソフトで使用しているリソースを以下に示します。

図 15 LIN 通信での使用リソース

	MB96610(MB2198-761-01-E)	備考
スライドボリューム(VR)	VR1(P06_3/AN3)	-
LED	LED0-1(P6_0-P6_1) LED2-3(P3_0-P3_1) LED4-7(P1_4-P1_7)	-
Reset SW	RSTX(SW2)	-
SW	P05_6/INT4_R(SW4)	-
	P00_3/INT11(SW7)	-
	P00_4/INT12(SW9)	-
LIN-USART(受信)	LIN-USART7(SIN7_R/P00_2)	LIN受信用
LIN-USART(送信)	LIN-USART7(SOT7_R/P00_1)	LIN送信用
LIN-USART(クロック)	LIN-USART7(SCK7_R/P00_0)	LINトランシーバ制御用
リロードタイマ	チャネル1	周期計測用
フリーランタイマ	チャネル1	各種タイムアウト検出用
アウトプットコンペア	チャネル6	各種タイムアウト検出用
インプットキャプチャ	チャネル9	sync field検出用

4.1.1 LIN 通信条件

表 1 LIN 通信条件に LIN サンプルソフトでの LIN 通信条件をまとめます。

表 1 LIN 通信条件

項目	設定値
通信速度	2400/9600(初期値)/19200bps
原発振	4MHz
周辺クロック(CLKP2) for LIN Clock	16MHz(PLL4 逓倍設定)
Sync Break 長	13bit (受信は 11bit 固定)
データ長	8bit
データビットフォーマット	LSB ファースト
データバイト数	8Byte

次に、表 2 LIN メッセージ概要 (ID&DATA) に LIN 通信ソフトウェアで使用する LIN ID および各 ID のデータを示します。偶数の LIN ID はマスタからスレーブへのデータ転送、奇数の LIN ID はスレーブからのマスタへのデータ転送としてアサインされています。

表 2 LIN メッセージ概要 (ID&DATA)

ID	データ内容	データ通信方向	Data Format							
			byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte 1	byte 0
0x00	Slaveアナログ値取得命令 (byte0:0x55)	Master → Slave(H+R)	予備	予備	予備	予備	予備	予備	AD値 (VR)	0x55
0x01	Slaveアナログ値(byte2:AD値)	Master → Slave(H) Master ← Slave(R)	予備	予備	予備	予備	予備	AD値 (VR)	予備	予備
0x02	LED01表示命令(byte0:0x55) Master SW4カウント数	Master → Slave(H+R)	予備	予備	予備	予備	予備	予備	SW4値 (Master)	0x55
0x03	Slave SW4カウント数	Master → Slave(H) Master ← Slave(R)	予備	予備	予備	予備	予備	SW4値 (Slave)	予備	予備
0x04	LED23表示命令(byte0:0x55) Master SW7カウント数	Master → Slave(H+R)	予備	予備	予備	予備	SW7値 (Master)	予備	予備	0x55
0x05	Slave SW7カウント数	Master → Slave(H) Master ← Slave(R)	予備	予備	予備	SW7値 (Slave)	予備	予備	予備	予備

※H：ヘッダーR：レスポンス

各、LIN ID におけるデータ内容は以下になります。

○LIN ID<0x00>

byte 0：LIN マスタから LIN スレーブへの Volume SW 情報取得命令 (0x55)

byte 1：LIN マスタの Volume SW 情報 (Analog 値)

○LIN ID<0x01>

byte 2：LIN スレーブの Volume SW 情報 (Analog 値)

○LIN ID<0x02>

byte 0：LIN マスタから LIN スレーブへの LED01 表示命令 (0x55)

byte 1：LIN マスタの SW4 カウント情報

○LIN ID<0x03>

byte 2：LIN スレーブの SW4 カウント情報

○LIN ID<0x04>

byte 0：LIN マスタから LIN スレーブへの LED23 表示命令 (0x55)

byte 3：LIN マスタの SW7 カウント情報

○LIN ID<0x05>

byte 4：LIN スレーブの SW7 カウント情報

4.2 LIN ドライバ設定

本サンプルプログラムは LIN マスタ, LIN スレーブ双方に対応しています。LIN マスタ, LIN スレーブの切り替えは lindmsg.h 中の define 定義で切り替えることが可能です。

〇LIN マスタとして使用する場合：#define LIN_MASTER 1

〇LIN スレーブとして使用する場合：#define LIN_MASTER 0

図 16 LIN マスタ⇔LIN スレーブの切り替え (lindbmsg.h)

```
/*============================================================*/
/*Function definition                                         */
/*        BASE_TIME:          5[ms]                           */
/*        LIN_MASTER:         0:SLABE NODE, 1:MASTER NODE     */
/*        LINUART_CH:         7                               */
/*        EXTERNAL_INT:       10                              */
/*        FREE_RUN_TIMER_CH   1                               */
/*        OUTPUT_COMPARE_CH   6                               */
/*============================================================*/
#define   BASE_TIME           5
#define   LIN_MASTER          1
#define   LINUART_CH          7
#define   EXTERNAL_INT        10
#define   FREERUN_TIMER_CH    1
#define   OUTPUT_COMPARE_CH   6
```

通信ボーレートの切り替えも同様に lindmsg.h 中の define 定義で切り替えることが可能です。

〇2400 bps の場合：#define BAUD_RATE 2400

〇9600 bps の場合：#define BAUD_RATE 9600

〇19200 bps の場合：#define BAUD_RATE 19200

図 17 ボーレートの切り替え

```
/*=========================================================*/
/*        BAUDRATE Definition(2400/9600/19200)             */
/*=========================================================*/
#define   BAUD_RATE       9600
#define   BAUD_RATE_ADJUST    1
```

4.3 LIN マスタサンプルプログラム

本サンプルプログラムの LIN 通信のフロー図を図 18 LIN 通信フローチャート (メインルーチン) および図 19 LIN 通信フローチャート (LIN 受信割込みルーチン) に示します。はじめに、マイコンの初期化, LIN-USART の初期化, タイマの初期化を行います。そして、LIN マスタとして、LIN バス接続処理, スケジュールセットを行います。その後、プログラムはループ状態に入ります。ループの中では、一定の周期でヘッダーの送信, レスポンスの送受信が行っています。ヘッダーとなる sync break, sync filed, ID field の送信, レスポンスの送受信は LIN-USART の受信割込みによって処理しています。送受信した LIN ID に応じてアプリケーション処理を行っています。

図 18 LIN 通信フローチャート (メインルーチン)

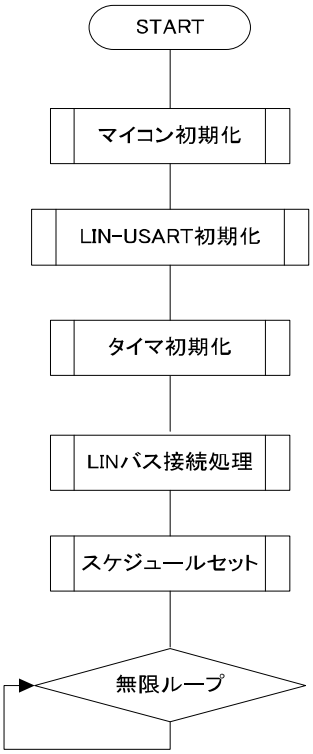


図 19 LIN 通信フローチャート (LIN 受信割込みルーチン)

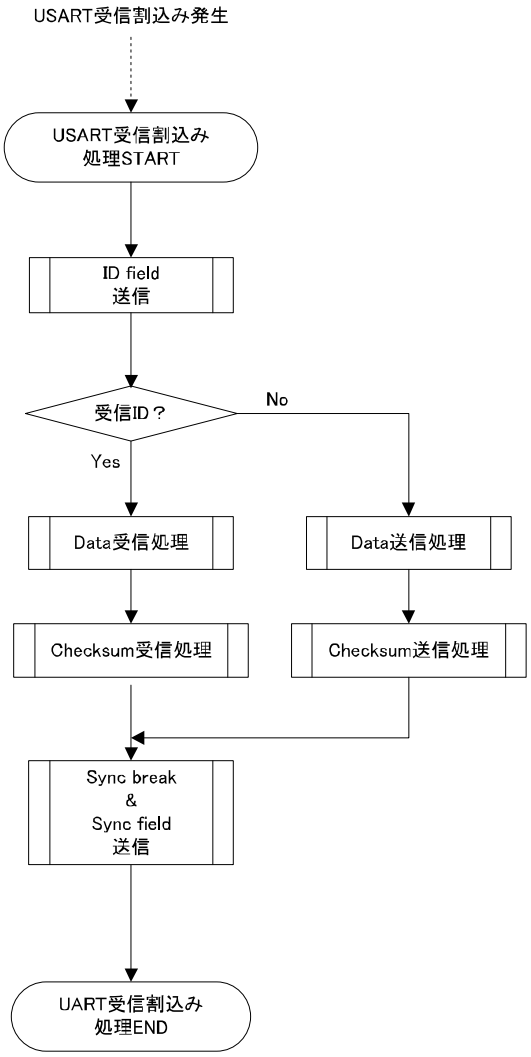
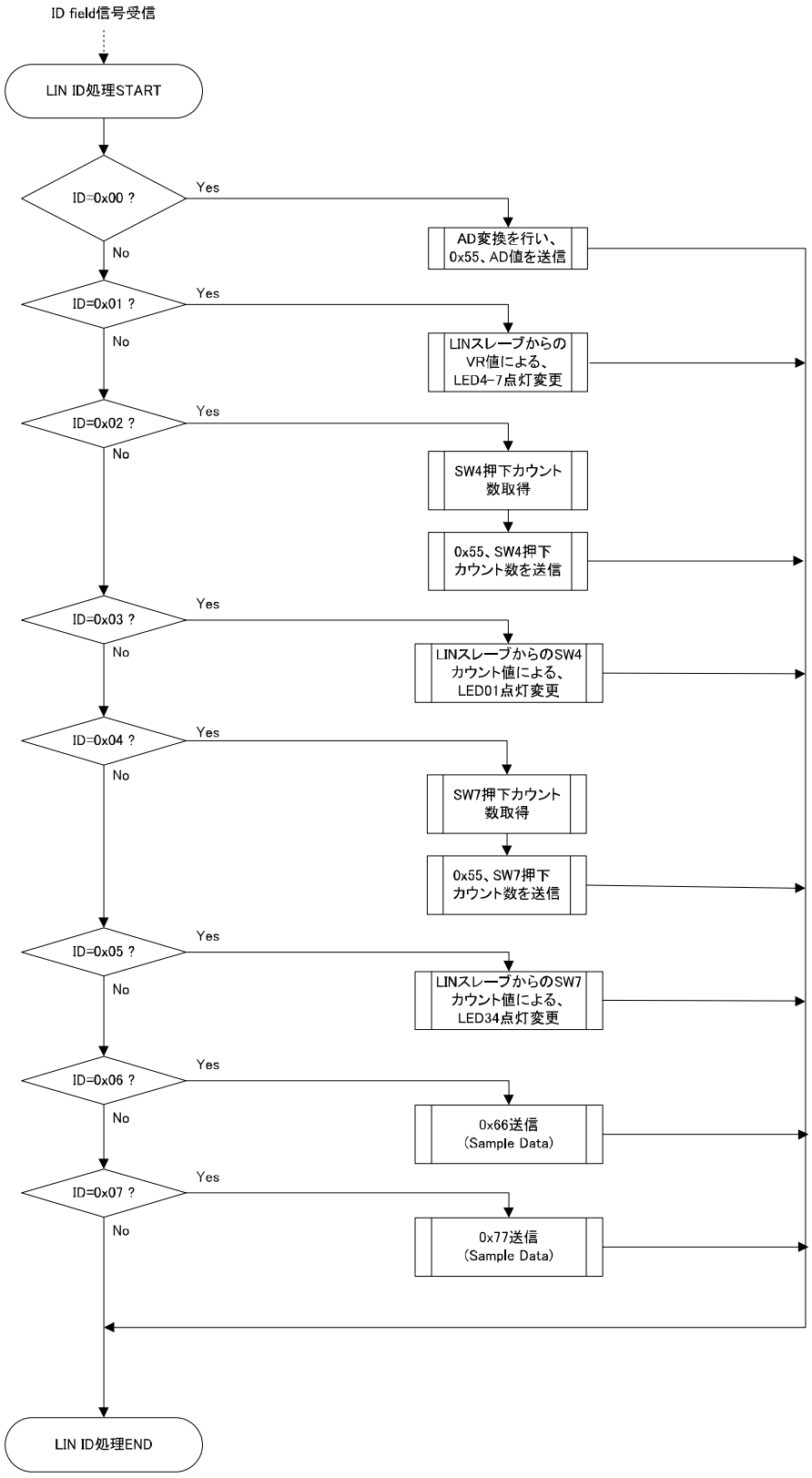


図 20 LIN 通信フローチャート (LIN ID に応じたアプリ処理)



LIN 通信時おける LIN プロトコルでの本サンプルプログラムの動作ポイントを以下に示します。

LIN マスタとなりますので、まずは、LIN バス接続処理、スケジュール登録を行ないます。

図 21 LIN バス接続処理

```
void      main(void)
{
          （省略）
          l_ifc_connect(hLIN_NORMAL_WAKEUP);          ←LIN バス接続処理
          l_sch_set(hSchedule1, Schedule1_DATA00);    ←スケジュールセット
```

LIN の ID 設定、スケジュール設定を行っている部分の一部について、以下に示します。

サンプルプログラムでは、1 つのスケジュールテーブルと 8 個の ID を使用しています。実際に、データ送受信用に使用しているのは、ID 0x00～ID 0x06 となります。

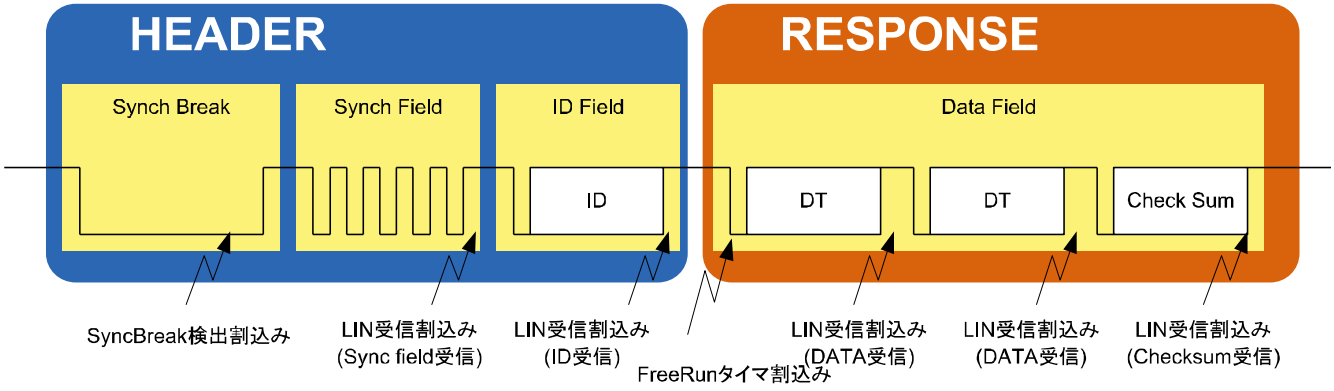
図 22 LIN 送受信 ID の登録-lindbmaster.h

```
typedef   enum {
          Schedule1_DATA00 = 0,                       ←8 個のID登録
          Schedule1_DATA01,
          Schedule1_DATA02,
          Schedule1_DATA03,
          Schedule1_DATA04,
          Schedule1_DATA05,
          Schedule1_DATA06,
          Schedule1_DATA07,
          （省略）
#define             Schedule1Count   8                ←8 個のID登録
          （省略）
__far const   l_u8   Schedule1_IdList[Schedule1Count] =   ←8 個のID登録
              {       ID_00, ID_01, ID_02, ID_03, ID_04, ID_05, ID_06, ID_07        };
```

```
l_u8*    __far const        LinTxDataPtr[64] = {        ←送信レスポンス登録
/*       0  1  2  3  4  5  6  7  8  9  */
/* 0*/    ucDATA00, 0, ucDATA02, 0, ucDATA04, 0, ucDATA06, 0, 0, 0,
/*10*/    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          （省略）
l_u8*    __far const        LinRxDataPtr[64] = {        ←受信レスポンス登録
/*       0  1  2  3  4  5  6  7  8  9  */
/* 0*/    0, ucDATA01, 0, ucDATA03, 0, ucDATA05, 0, ucDATA07, 0, 0,
```

本サンプルプログラムは、図 23 各種割込み処理の動作ポイントのように、複数の割込み処理により動作します。LIN プロトコルの各フィールドでの本サンプルソフトの処理をみていきます。

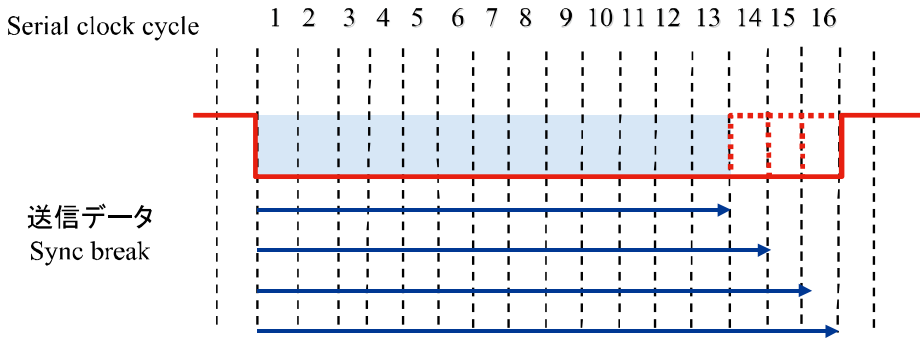
図 23 各種割込み処理の動作ポイント



① Sync break

sync break では sync break 信号 (13～16 ビットの Low 信号) の送信を行います。サンプルプログラムでは、13 ビットの sync break 送信を行っています。同時に受信処理も行なっており、バスが 11 ビットタイム以上 ”0” になると、sync break 検出割込み (LBD) が発生します。Sync break 検出後、sync break 検出フラグのクリアを行います。

図 24 sync break データ設定



LBR	LIN Synch break生成ビット
0	影響なし
1	LIN Synch break 生成

LBL0	LBL1	LIN Synch break長選択ビット
0	0	13ビット分
1	0	14ビット分
0	1	15ビット分
1	1	16ビット分

図 25 sync break 割込み制御

```
__interrupt void    _LinUartRx(void)
{
        if ((ssr & 0xE0) != 0) {                          ←エラーチェック
        (省略)
        } else if (IO_UART3_ESCR3_bit_LBD == SET) {       ←sync break 検出
        #if         (LIN_MASTER==1)
                    IO_UART3_ESCR3_bit_LBD = CLEAR;       ←sync break 検出フラグクリア
        (省略)
    }else{
        l_ifc_rx(data);                          ←受信処理
```

② Sync field

sync break 検出後、sync field の送受信を行います。マスタは sync field として 0x55 の送信を行い、同時に自身が送信したデータが 0x55 かどうかチェックします。

図 26 sync field 制御

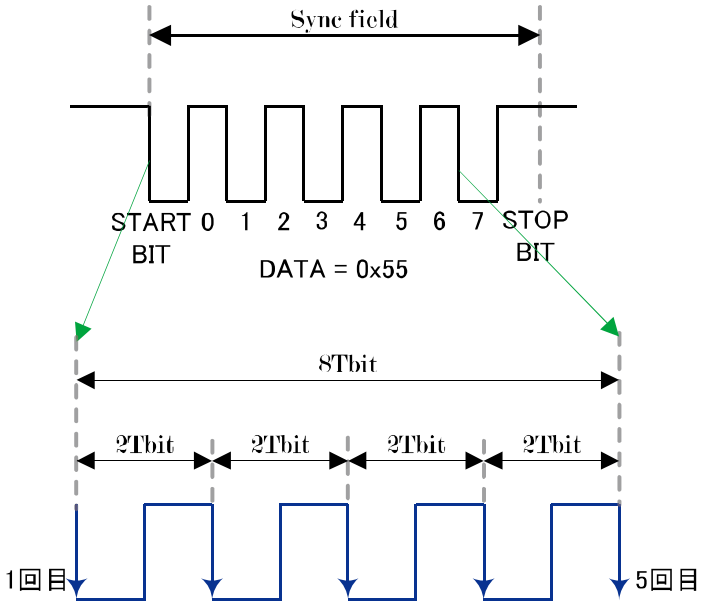


図 27 受信判定処理のように受信割込み関数の中で呼ばれる l_ifc_rx (l_ifc_handle rx_data) で、Sync field の受信制御が行われます。l_ifc_rx (l_ifc_handle rx_data) では、LIN ステータスに応じて sync break 受信, sync field 受信, ID 受信, DATA 送受信, WAKEUP 送信処理が行われます。

図 27 受信判定処理

```
void      l_ifc_rx(l_ifc_handle rx_data)
{
  switch(ucLinStatus){
    case   LIN_TRANSMIT:                                  ← DATA FIELD 送信
           （省略）
    case   LIN_DATA_RECEPTION:                            ← DATA FIELD 受信
           （省略）
    case   LIN_ID_RECEPTION:                              ← ID FIELD 受信待ち
           （省略）
    case   LIN_WAIT_SYNCH_FIELD:                          ←Sync field 受信待ち
           （省略）
    case   LIN_MS_WAIT_SYNCH_BREAK:                       ←Sync break 受信待ち
#if        (LINUART_CH==3)
                IO_UART3_ESCR3_byte = 0x00;               ←ESCR レジスタのクリア
           (省略)
                ucLinStatus = LIN_WAIT_SYNCH_FIELD;       ←Sync  field受信待ち状態へ
           (省略)
    case   LIN_WAKEUP_TRANSMIT:                           ←WAKEUP 送信状態
           (省略)
  }
}
```

図 28 sync field 受信判定処理にて、データが 0x55 である事を確認したら、ID field 受信待ち状態となります。その後、マスタは ID の送信処理を行ないます。

図 28 sync field 受信判定処理

```
void    l_ifc_rx(l_ifc_handle rx_data)
{
        (省略)
        case    LIN_WAIT_SYNCH_FIELD:           ←Sync field 受信待ち
                if (rx_data == SYNCH_FIELD_CHAR) {   ←Sync field0x55 かをチェック
        (省略)
                ucLinStatus = LIN_ID_RECEPTION;   ←ID 受信待ち状態へ
                l_ifc_tx(ucLinMsScheduleCurrentId);   ←ID 送信処理
        (省略)
```

なお、sync field (0x55) 送信は図 29 USART 送信起動処理のように送信起動処理の関数 l_ifc_tx (l_ifc_handle tx_data) の中で行われます。Sync field 送信の場合は tx_data として 0x55 がセットされます。

図 29 USART 送信起動処理

```
void    l_ifc_tx(l_ifc_handle tx_data)
{
#if     (LINUART_CH == 3)
        IO_UART3_RDR3 = tx_data;          ←送信データをレジスタに設定
        (省略)
```

③ ID field

ID 送信処理は、sync field 受信による LIN 受信割込みの中で行われます。この時の LIN ステータスは LIN_WAIT_SYNCH_FIELD です。LIN ステータスを LIN_ID_RECEPTION に切り替えた後に、ID スケジュールテーブルに登録された ID が送信されます。

ID 受信も同様に LIN 受信割込みの中で処理が行われます。ID 受信判定処理では、取得した ID が送信用 ID もしくは受信用の ID かの判定およびパリティチェックを行い、送信用ならステータスを送信準備状態へ遷移させて送信データをバッファにコピーします。受信用であれば、DATA 受信待ち状態へ遷移させ、スレーブからのレスポンス (データ) 受信に備えます。

図 30 ID 受信判定処理

```
void    l_ifc_rx(l_ifc_handle rx_data){
        (省略)
    case  LIN_ID_RECEPTION:                ← ID FIELD 受信待ち状態
        ucCurrentId.byte   = rx_data;          ←受信した ID の格納
        if( ucCurrentId.fields.parity != ucRightParity[ucCurrentId.fields.id] ) { ←Parity チェック
        l_flg_tst(hBIT_ERR);                   ←エラー処理
        l_flg_clr(hBIT_ERR);
        (省略)
        } else if( LinRxDataPtr[ucCurrentId.fields.id] != 0 ) {←受信 ID ならば
            ucLinStatus = LIN_DATA_RECEPTION;      ←DATA 受信待ち状態
        (省略)
            vSetLinFreerunTimersCompare(ucRxCount);  ←フリーランタイマセット
        } else if ( LinTxDataPtr[ucCurrentId.fields.id] != 0 ) {←送信 ID ならば
        (省略)
                                                ↓送信データをバッファにコピー
            vLinWordCopy(ucUartTxBuffer, LinTxDataPtr[ucCurrentId.fields.id], ucTxCount);
            vSetLinFreerunTimersCompare(hTINFRAME_SPACE_IND); ←フリーラン
        }                                                      タイマセット
        (省略)
    }
}
```

④ DATA field

DATA field での DATA 送信、受信処理について説明していきます。

まず、DATA 送信についてですが、ID field にて受信した ID が送信用の ID であった場合に、フリーランタイマの割込みによってタイムアウト処理のための vTimeoutCheckTask 関数が呼ばれます。この関数は、フリーランタイマで設定したタイムアウト値を検出した時に呼ばれる関数で、この場合では、ヘッダー受信からレスポンス送信までのタイムアウト値 (レスポンススペース) を検出すると呼ばれます。

vTimeoutCheckTask 関数の中では、ステータス情報によって送信前処理、初期化処理等を切り分けていて、ステータスが送信前状態の場合に、DATA の 1byte 目の送信を行います。

図 31 タイムアウト検出処理

```
void      vTimeoutCheckTask(void){
    (省略)
    if ( uiIntDemandCounter == 0 ) {
        switch ( ucLinStatus ) {
        case      LIN_PRETRANSMIT:          ← 送信前状態
            ucLinStatus = LIN_TRANSMIT;     ← 状態遷移:DATA FIELD 送信状態
            ucSaveData = ucUartTxBuffer[0]; ← 送信データ 1byte 取得
            l_ifc_tx(ucUartTxBuffer[0]);    ← データ送信処理
            (省略)
        case      LIN_UART_INITIAL:
                  (省略)
        case      LIN_ID_RECEPTION:
                  (省略)
        case      LIN_DATA_RECEPTION:
                  (省略)
        case      LIN_TRANSMIT:
                  (省略)
        case      LIN_WAIT_SYNCH_FIELD_START:
                  (省略)
    }
}
```

1byte 目のデータを送信すると、自身が送信した DATA を受信することで、受信割込みが発生します。すると、ID field での動作と同様に、受信判定処理の関数_ifc_rx (l_ifc_handlerx_data) が呼ばれ、図 32 DATA 送信処理のように DATA FIELD 送信状態のもと 2byte 目以降のデータの送信処理が行われ、同様の処理を繰り返します。今回の LIN 通信では DATA バイト数は 8 に設定しているため、8byte 分の DATA の送信が終わると、最後に Checksum の送信を行い、送信処理が終了します。

図 32 DATA 送信処理

```
void      l_ifc_rx(l_ifc_handle rx_data){
    switch(ucLinStatus){
    case   LIN_TRANSMIT:        ← DATA FIELD 送信状態
           if ( ucTxCurrentIndex < ucTxCount ){          ←送信 DATA が残っていれば
           (省略)
           l_ifc_tx(ucUartTxBuffer[ucTxCurrentIndex]);   ←送信処理
           (省略)
           } else if ( ucTxCurrentIndex == ucTxCount ){  ←送信 DATA を全て送信していれば
           (省略)
           l_ifc_tx(((unsigned char)~uiTxCheckSum));     ←Checksum 送信処理
           (省略)
           }
    case   LIN_DATA_RECEPTION:
           (省略)
    case   LIN_ID_RECEPTION:
           (省略)
    case   LIN_WAKEUP_TRANSMIT:
           (省略)
    }
}
```

ID 受信処理にて取得した ID が受信用であった場合には、ステータスを DATA 受信状態に遷移させ、スレーブからの DATA 受信を待ちます。スレーブからの DATA 受信による受信割込みが発生すると、図 33 DATA 受信処理のように受信処理の関数 l_ifc_rx (data) の中で受信処理が行われます。DATA 受信の場合も 2byte 目以降の DATA 送信と同様に、1byte のデータを受信するごとに l_ifc_rx (l_ifc_handle rx_data) で受信処理を行い、8byte 分の全ての DATA を受信したら、チェックサムエラーがなければ受信成功フラグをセットし、受信処理が終了します。

図 33 DATA 受信処理

```
void      l_ifc_rx(l_ifc_handle rx_data){
    switch(ucLinStatus){
    case   LIN_TRANSMIT:
          (省略)
    case   LIN_DATA_RECEPTION:          ← DATA FIELD 受信状態
          if ( ucRxCurrentIndex >= ucRxCount ) {  ← 全てのデータを受信していたら
              if ( (uiRxCheckSum + rx_data) == 0xFF ) {← Checksum 計算正常なら
                  (省略)
                  flagsLinTxRx.bit.SucceedReception = SET; ← 受信成功フラグセット
                  memcpy( &ucUartRxFixedBuffer[0], &ucUartRxBuffer[0], ucRxCount );
                  (省略)                          ↑受信データコピー
              } else {                            ← Checksum エラーなら
                  l_flg_tst(hCHECKSUM_ERR);       ← エラー処理
          } else {                        ← 受信データがまだ残っていれば
              ucUartRxBuffer[ucRxCurrentIndex] = rx_data;  ← バッファに受信データ格納
              (省略)
    case   LIN_ID_RECEPTION:
          (省略)
    case   LIN_WAKEUP_TRANSMIT:
          (省略)
    }
}
```

main.c にある vBaseTimeTask 関数の中で、データ送受信が完了したかどうかのチェックを定期的に行っています。すべてのデータ受信が完了した (flagsLinTxRx.bit.SucceedReception が SET された) 時点でこの関数が呼ばれると、受信完了処理として submain. c にある sub_control 関数が呼ばれ、図 20 LIN 通信フローチャート (LIN ID に応じたアプリ処理) のように受信した LIN ID に応じたアプリケーション動作が行われます。

4.4 LIN スレーブサンプルプログラム

本サンプルプログラムの LIN 通信のフロー図を図 34 LIN 通信フローチャートおよび図 35 LIN 通信フローチャート (LIN 受信割込み) に示します。はじめに、マイコンの初期化、LIN-USART の初期化、タイマの初期化を行います。そして、LIN スレーブとして、LIN バス接続処理を行います。その後、プログラムはループ状態に入ります。ループ処理の中では、一定の周期でデータの送受信が完了できたかどうかを監視しており、データ受信完了時に ID に応じたアプリケーション処理を行っています。LIN スレーブとして動作するための sync break 検出、ID 受信、データの送受信は LIN-USART の受信割込み処理の中で行っています。また sync filed におけるボーレート調整は、図 36 LIN 通信フローチャート (インプットキャプチャ割込み) のようにインプットキャプチャの割込み処理の中で行っています。

図 34 LIN 通信フローチャート

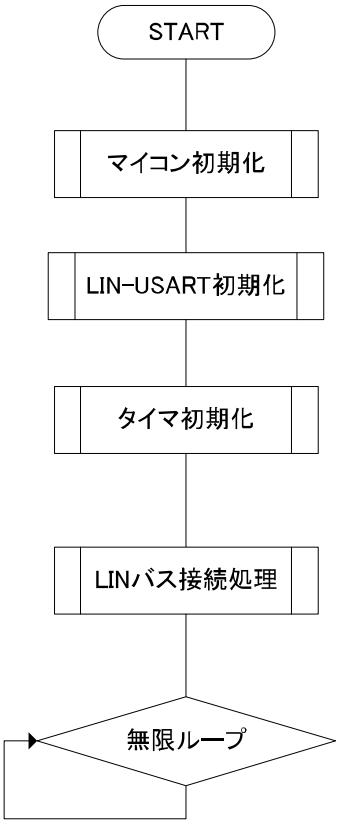


図 35 LIN 通信フローチャート (LIN 受信割込み)

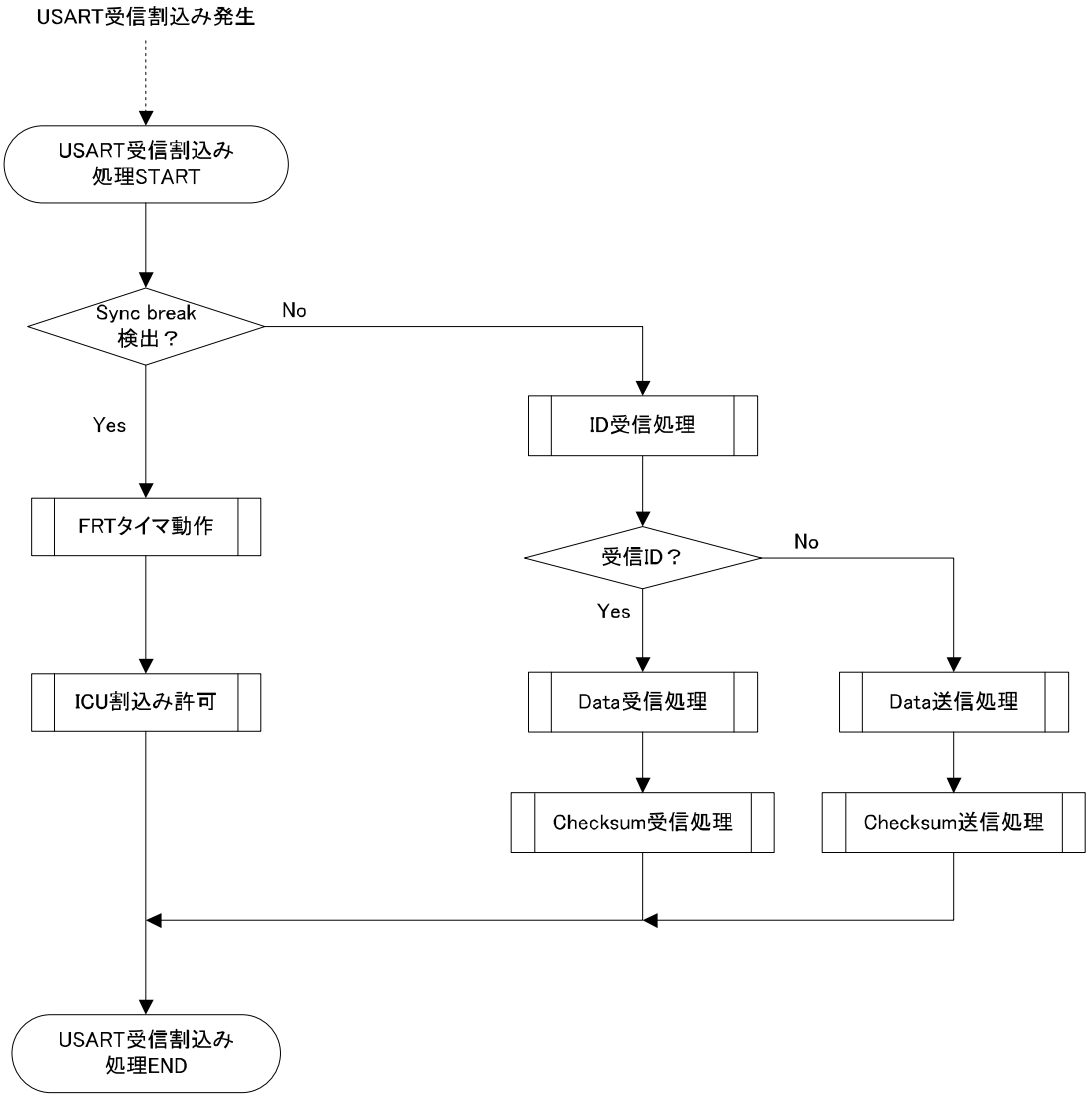


図 36 LIN 通信フローチャート (インプットキャプチャ割込み)

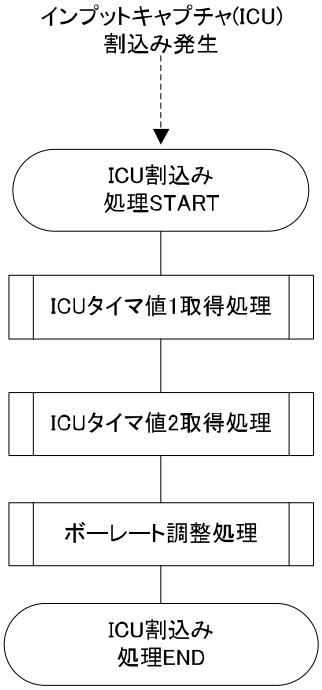
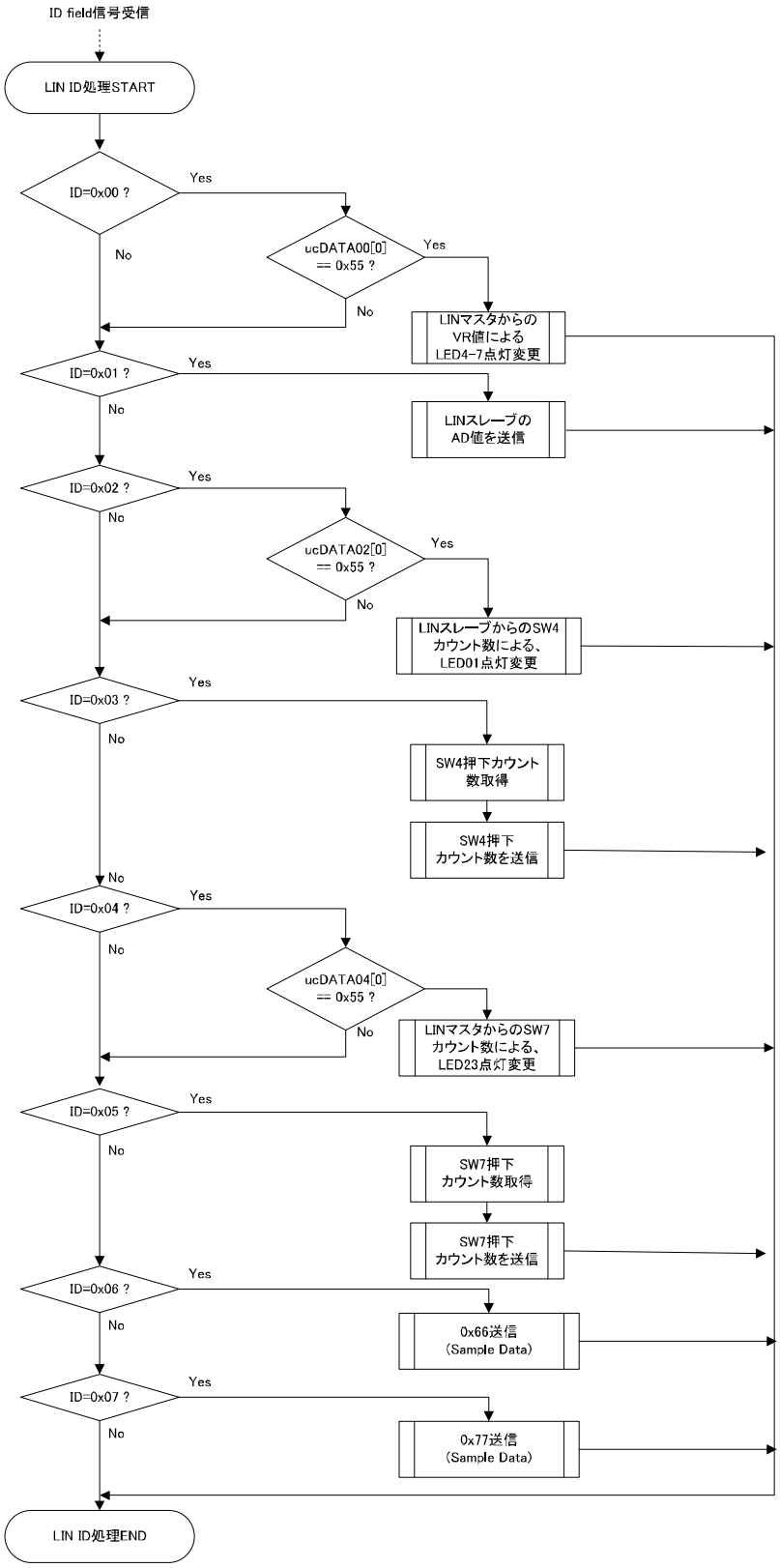
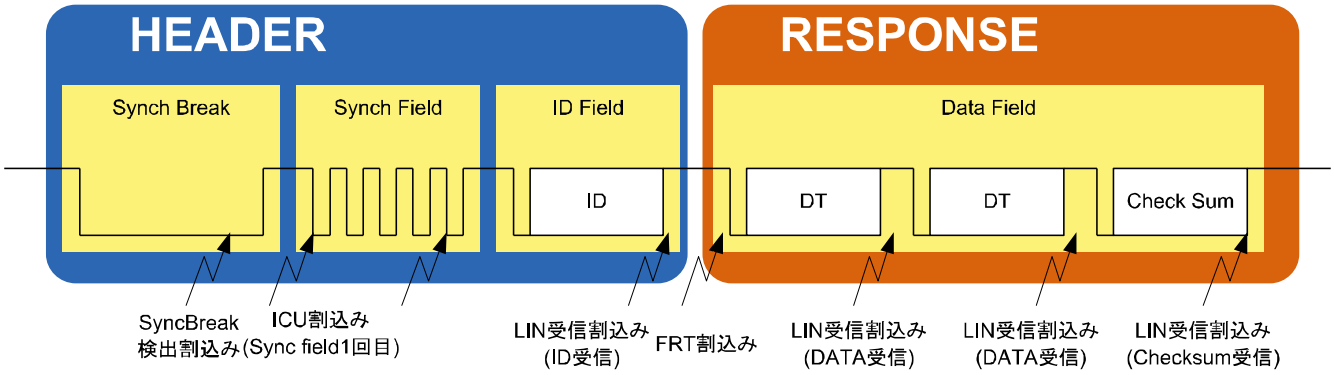


図 37 LIN 通信フローチャート (LIN ID 応じたアプリ処理)



続いて本サンプルプログラムについて説明します。スレーブとしての LIN 通信において、LIN プロトコルでの本サンプルプログラムの動作ポイントを以下に示します。本サンプルソフトは、図 38 各種割込み処理の動作ポイントのように、複数の割込み処理により LIN スレーブとして動作しています。LIN フレームの各フィールドでの本サンプルソフトの処理をみていきます。

図 38 各種割込み処理の動作ポイント



① Sync break

Sync break では LIN マスタから sync break 信号 (13～16 ビットの Low 信号) を受信し、バスが 11 ビットタイム以上 ”0” になると、sync break 検出割込みが発生します。Sync break 検出割込みを検出すると、sync break 割込み禁止設定およびインプットキャプチャ割込みを許可し、sync field 開始待ち状態へと遷移します。

図 39 sync break 検出割込み制御

```
__interrupt void   _LinUartRx(void)
{
        if ((ssr & 0xE0) != 0) {                          ←エラーチェック
        (省略)
        } else if (ESCR_LBD == SET) {                     ←synch break 検出
        ESCR_LBD = CLEAR;                                 ←synch break 検出フラグクリア
        (省略)
        vSetLinFreerunTimersCompare(hTHEADER_MAX_IND);  ←複合タイマ(FRT)値セット
        ucLinStatus = LIN_WAIT_SYNCH_FIELD_START; ←状態遷移:synch field 開始待ち
        (省略)
        T00CR1_IE = SET;          ←インプットキャプチャ割込み許可
}
```

② Sync field

LIN スレーブは sync break 検出後、sync field にてインプットキャプチャを用いてボーレートを測定し補正を行っています。本サンプルソフトでは、インプットキャプチャを、両エッジ検出、インプットキャプチャの入力ソースを LIN チャネル (Ch7) に設定しています。インプットキャプチャの割込みを許可設定および両エッジ検出に設定すると、エッジ検出時にインプットキャプチャの割込みが入ります。両エッジでのタイマ値およびオーバーフローした回数を測定し、8 で割ることでボーレートを算出し調整を行っています。

図 40 sync field におけるインプットキャプチャ動作

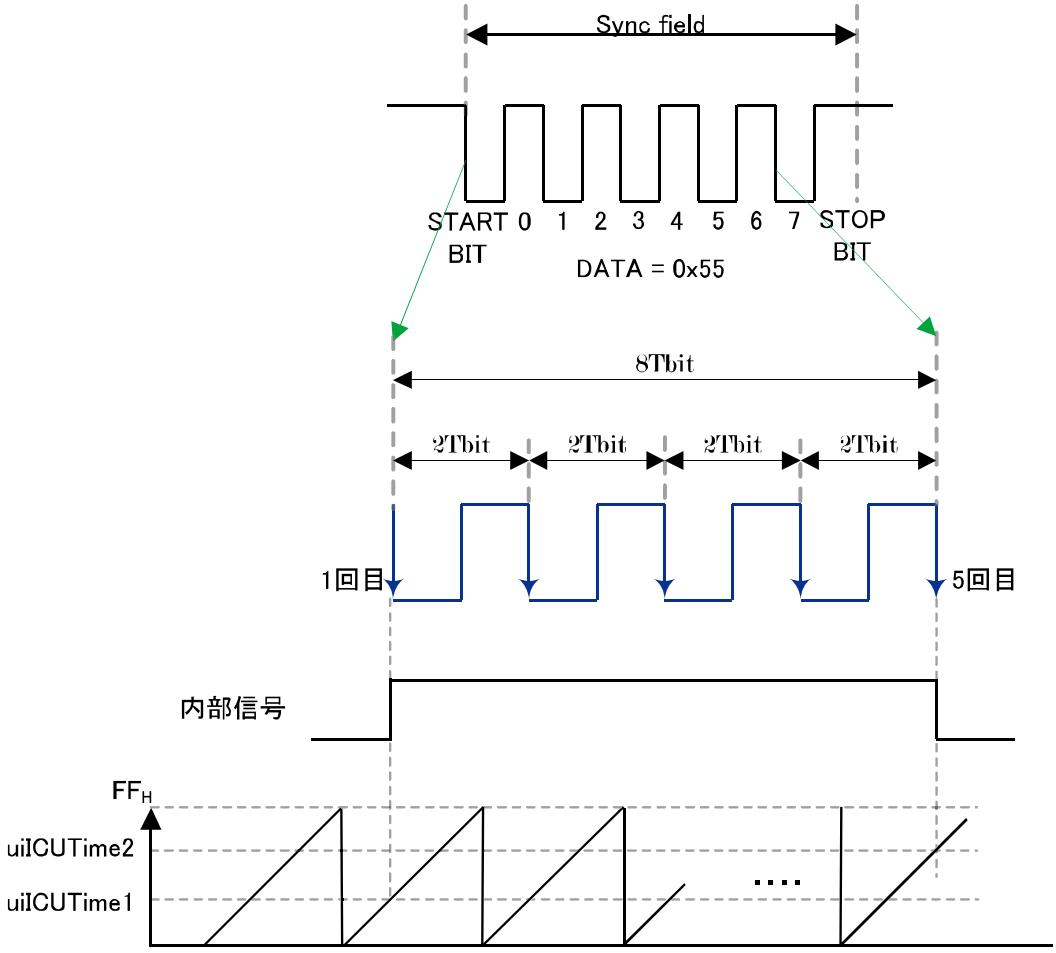


図 41 インプットキャプチャ (ICU) 割込み制御

```
__interrupt void   _LinICU (void)
{
        if (ucLinStatus == LIN_WAIT_SYNCH_FIELD_START){
                uiICUTime1 = IO_IPCP9;                  ←synch field 開始待ち
                (省略)                                  ←タイマ値取得
                ucLinStatus =   LIN_WAIT_SYNCH_FIELD_END;
        else if(ucLinStatus == LIN_WAIT_SYNCH_FIELD_END){   ←状態遷移:
                (省略)                                      synch field 終了待ち
                uiICUTime2= IO_IPCP9;
                /*   adjust Boud Rate   */  ←タイマ値取得
                (省略)
                vEnableLinUartReception();          ←ボーレート調整処理
                ucLinStatus = LIN_ID_RECEPTION;     ←LIN-USART 割込み許可
        }                                           ←状態遷移:ID 受信
}
```

③　ID field

ID 受信処理は、LIN-USART の割込み関数_LinUART (void) の中で行われます。割込み発生時にエラーの発生がなく、割込み要因が sync break での割込みでなければ受信処理が行われます。

図 42 LIN-USART 受信割込み制御

```
__interrupt void   _LinUART (void)
{
    (省略)
    if ((ssr & 0xE0) != 0) {                ←エラーチェック
            (省略)
    } else if (ESCR_LBD == SET) {           ←sync break での割込かどうかをチェック
            (省略)
    }else{
          l_ifc_rx(data);                   ←受信処理
    }
}
```

図 43 受信判定処理のように受信判定処理の関数 l_ifc_rx (l_ifc_handle rx_data) の中では、ステータスによって ID 受信、DATA 送信、DATA 受信、WAKEUP 送信処理を切り分けています。通常シーケンスでは、インプットキャプチャの 2 回目の割込み処理の中にて状態を ID FIELD 受信待ち状態に遷移させているため、ID 受信処理が行われます。ID 受信処理では、取得した ID が送信用 ID もしくは受信用の ID かの判定およびパリティチェックを行い、送信用ならステータスを送信準備状態へ遷移させて送信データをバッファにコピーします。受信用であれば、DATA 受信待ち状態へ遷移させ、マスタからのレスポンス (データ) 受信に備えます。

図 43 受信判定処理

```
void      l_ifc_rx(l_ifc_handle rx_data){
    switch(ucLinStatus){
    case   LIN_TRANSMIT:                          ← DATA FIELD 送信状態
             （省略）
    case   LIN_DATA_RECEPTION:                    ← DATA FIELD 受信状態
             （省略）
    case   LIN_ID_RECEPTION:                      ← ID FIELD 受信待ち状態
           ucCurrentId.byte   = rx_data;          ←受信した ID の格納
           if( ucCurrentId.fields.parity != ucRightParity[ucCurrentId.fields.id] ) { ←Parity チェック
             （省略）                             ←エラー処理
           else if( LinRxDataPtr[ucCurrentId.fields.id] != 0 ) {   ←受信 ID ならば
              ucLinStatus = LIN_DATA_RECEPTION;   ←状態遷移：DATA 受信待ち状態
              (省略）
              vSetLinFreerunTimersCompare(ucRxCount);  ←8/16bit 複合タイマセット
           } else if ( LinTxDataPtr[ucCurrentId.fields.id] != 0 ) {
              ucLinStatus = LIN_PRETRANSMIT;
             （省略）                             ↓送信データをバッファにコピー
              vLinWordCopy(ucUartTxBuffer, LinTxDataPtr[ucCurrentId.fields.id], ucTxCount);
              vSetLinFreerunTimersCompare(hTINFRAME_SPACE_IND);  ←8/16bit 複合
           }                                                     タイマセット

             （省略）
    case   LIN_WAKEUP_TRANSMIT:                   ← WAKEUP 送信状態
             （省略）
    }
}
```

④ DATA field

DATA field での DATA 送信、受信処理について説明していきます。

まず、DATA 送信についてですが、ID field にて受信した ID が送信用の ID であった場合に、フリーランタイマの割込みによって**エラー! スイッチの指定が正しくありません。**のような vTimeoutCheckTask 関数が呼ばれます。この関数は、フリーランタイマで設定したタイムアウト値を検出した時に呼ばれる関数で、この場合では、ヘッダー受信からレスポンス送信までのタイムアウト値(レスポンススペース) を検出すると呼ばれます。vTimeoutCheckTask 関数の中では、ステータス情報によって送信前処理、初期化処理等を切り分けていて、ステータスが送信前状態の場合に、DATA の 1byte 目の送信を行います。

図 44 タイムアウト検出処理

```
void        vTimeoutCheckTask(void){
      (省略)
      if ( uiIntDemandCounter == 0 ) {
            switch ( ucLinStatus ) {
            case        LIN_PRETRANSMIT:          ← 送信前状態
                  ucLinStatus = LIN_TRANSMIT;     ← 状態遷移:DATA FIELD 送信状態
                  ucSaveData = ucUartTxBuffer[0]; ← 送信データ 1byte 取得
                  l_ifc_tx(ucUartTxBuffer[0]);    ← データ送信処理
                  (省略)
            case        LIN_UART_INITIAL:
                        (省略)
            case        LIN_ID_RECEPTION:
                        (省略)
            case        LIN_DATA_RECEPTION:
                        (省略)
            case        LIN_TRANSMIT:
                        (省略)
            case        LIN_WAIT_SYNCH_FIELD_START:
                        (省略)
      }
}
```

1byte 目のデータを送信すると、自身が送信した DATA を受信することで、受信割込みが発生します。すると、ID field での動作と同様に、受信判定処理の関数_ifc_rx(l_ifc_handle rx_data)が呼ばれ、図 45 DATA 送信処理のように DATA FIELD 送信状態のもと 2byte 目以降のデータの送信処理が行われ、同様の処理を繰り返します。今回の LIN 通信では DATA バイト数は 8 に設定している為、8byte 分の DATA の送信が終わると、最後に Checksum の送信を行い、送信処理が終了します。

図 45 DATA 送信処理

```
void      l_ifc_rx(l_ifc_handle rx_data){
    switch(ucLinStatus){
    case   LIN_TRANSMIT:        ← DATA FIELD 送信状態
           if ( ucTxCurrentIndex < ucTxCount ){          ←送信 DATA が残っていれば
           (省略)
           l_ifc_tx(ucUartTxBuffer[ucTxCurrentIndex]);   ←送信処理
           (省略)
           } else if ( ucTxCurrentIndex == ucTxCount ){  ←送信 DATA を全て送信していれば
           (省略)
           l_ifc_tx(((unsigned char)~uiTxCheckSum));     ←Checksum 送信処理
           (省略)
           }
    case   LIN_DATA_RECEPTION:
           (省略)
    case   LIN_ID_RECEPTION:
           (省略)
    case   LIN_WAKEUP_TRANSMIT:
           (省略)
    }
}
```

ID 受信処理にて取得した ID が受信用であった場合には、ステータスを DATA 受信状態に遷移させ、LIN マスタからの DATA 受信を待ちます。LIN マスタからの DATA 受信による LIN 受信割込みが発生すると、図 46 DATA 受信処理のように受信処理の関数 l_ifc_rx(l_ifc_handle rx_data の中で受信処理が行われます。DATA 受信の場合も 2byte 目以降の DATA 送信と同様に、1byte のデータを受信する毎に l_ifc_rx(data)で受信処理を行い、8byte 分の全ての DATA を受信したら、チェックサムエラーがなければ受信成功フラグをセットし、受信処理が終了します。

図 46 DATA 受信処理

```
void      l_ifc_rx(l_ifc_handle rx_data){
    switch(ucLinStatus){
    case   LIN_TRANSMIT:
          (省略)
    case   LIN_DATA_RECEPTION:          ← DATA FIELD 受信状態
          if ( ucRxCurrentIndex >= ucRxCount ) {  ← 全てのデータを受信していたら
              if ( (uiRxCheckSum + rx_data) == 0xFF ) {← Checksum 計算正常なら
                  (省略)
                  flagsLinTxRx.bit.SucceedReception = SET; ← 受信成功フラグセット
                  memcpy( &ucUartRxFixedBuffer[0], &ucUartRxBuffer[0], ucRxCount );
                  (省略)                          ↑受信データコピー
              } else {                            ← Checksum エラーなら
                  l_flg_tst(hCHECKSUM_ERR);       ← エラー処理
          } else {                        ← 受信データがまだ残っていれば
              ucUartRxBuffer[ucRxCurrentIndex] = rx_data;  ← バッファに受信データ格納
              (省略)
    case   LIN_ID_RECEPTION:
          (省略)
    case   LIN_WAKEUP_TRANSMIT:
          (省略)
    }
}
```

最後に、受信した ID に応じたアプリケーション動作ですが、main.c にある vBaseTimeTask 関数の中でデータ送受信が完了したかどうかのチェックを定期的行っています。すべてのデータ受信が完了した(flagsLinTxRx.bit.SucceedReception が SET された)時点でこの関数が呼ばれると、受信完了処理として submain.c にある sub_control 関数が呼ばれ、図 37 LIN 通信フローチャート (LIN ID 応じたアプリ処理)のようにアプリケーション動作が行われます。

改版履歴

ドキュメント名: 16-bit Microcontroller F^{2}MC-16FX ファミリ MB96600 シリーズ LIN の使用方法

ドキュメント番号: 002-04337

Revision	ECN	変更者	発行日	変更内容
**	-	KHAS	11/08/2012	1 版: 初版 英語版の 002-04336 Rev. **を翻訳した日本語版に該当します
*A	5623642	KHAS	02/08/2017	サイプレスフォーマットに変換しました。内容の更新はしておりません。

セールス, ソリューションおよび法律情報

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店、および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

製品

ARM® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
タッチ センシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス／RF	cypress.com/wireless

PSoC® ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

サイプレス開発者コミュニティ

フォーラム | WICED IOT Forums | Projects | ビデオ | ブログ | トレーニング | Components

テクニカルサポート

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2011-2017. 本書面は、Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社（以下、「Cypress」という。）に帰属する財産である。本書面（本書面に含まれ又は言及されているあらゆるソフトウェア又はファームウェア（以下、「本ソフトウェア」という。）を含む）は、アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき、Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し、また、本段落で特に記載されているものを除き、Cypress の特許権、著作権、商標権又はその他の知的財産権のライセンスを一切許諾していない。本ソフトウェアにライセンス契約書が伴っておらず、かつ、あなたが Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意をしていない場合、Cypress は、あなたに対して、（1）本ソフトウェアの著作権に基づき、（a）ソースコード形式で提供されている本ソフトウェアについて、Cypress ハードウェア製品と共に用いるためにのみ、組織内部でのみ、本ソフトウェアの修正及び複製を行うこと、並びに（b）Cypress のハードウェア製品ユニットに用いるためにのみ、（直接又は再販売者及び販売代理店を介して間接のいずれかで）エンドユーザーに対して、バイナリーコード形式で本ソフトウェアを外部に配布すること、並びに（2）本ソフトウェア（Cypress により提供され、修正がなされていないもの）に抵触する Cypress の特許権のクレームに基づき、Cypress ハードウェア製品と共に用いるためにのみ、本ソフトウェアの作成、利用、配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス（サブライセンスの権利を除く）を付与する。本ソフトウェアのその他の使用、複製、修正、変換又はコンパイルを禁止する。

適用される法律により許される範囲内で、Cypress は、本書面又はいかなる本ソフトウェアに関しても、明示又は黙示をとわず、いかなる保証（商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない）も行わない。適用される法律により許される範囲内で、Cypress は、別途通知することなく、本書面を変更する権利を留保する。Cypress は、本書面に記載のあるいかなる製品又は回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報（あらゆるサンプルデザイン情報又はプログラムコードを含む）は、参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計し、プログラムし、かつテストすることは、本書面のユーザーの責任において行われるものとする。Cypress 製品は、兵器、兵器システム、原子力施設、生命維持装置若しくは生命維持システム、蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム、汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分として用いるため、又はシステムの不具合が人身傷害、死亡若しくは物的損害を生じさせることになるその他の使用（以下、「本目的外使用」という。）のためには、設計、意図又は承認されていない。重要な構成部分とは、装置又はシステムのその構成部分の不具合が、その装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できる、機器又はシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ、若しくは本目的外使用に関連するいかなる請求、損害又はその他の責任についても、Cypress はその全部又は一部をとわず一切の責任を負わず、かつ、あなたは Cypress をそれら一切から免除するものとし、本書により免除する。あなたは、Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求、費用、損害及びその他の責任（人身傷害又は死亡に基づく請求を含む）から Cypress を免責補償する。

Cypress、Cypress のロゴ、Spansion、Spansion のロゴ及びこれらの組み合わせ、WICED、PSoC、CapsSense、EZ-USB、F-RAM、及び Traveo は、米国及びその他の国における Cypress の商標又は登録商標である。Cypress の商標のより完全なリストは、cypress.com を参照のこと。その他の名称及びブランドは、それぞれの権利者の財産として権利主張がなされている可能性がある。