
TLIN (Toppers - Local Interface Network)

アプリケーション・ガイド

Revision 1.0

2007 年 3 月 1 日

株式会社 OTSL



Edition History

発行日	バージョン	作成者・加筆者	改版の内容
2006/03/01	1.0	日野	初版

目次

1. はじめに	7
2. LIN 仕様について	8
2.1. LIN CONSORNIUM	8
2.2. LIN SPECIFICATION	8
3. TLIN REFERENCE CODE の使い方	10
3.1. 独自アプリケーションの開発	10
3.1.1. スケジュールおよび通信フレームの設計	11
3.1.1.1. LDF の入手	11
3.1.1.2. LDF から C ソースコードへの変換	12
3.1.2. アプリケーション・プログラムの書き方	19
3.1.2.1. include ファイル	19
3.1.2.2. その他一般的な定義文	21
3.1.3. 初期化	22
3.1.3.1. システムの初期化	23
3.1.3.2. エンティティ・アクセサの初期化	23
3.1.3.3. フレーム管理タスクとスタック・タスクの起動	24
3.1.3.4. TLIN の初期化	24
3.1.3.5. インタフェースの初期化と接続	25
3.1.4. 通信駆動方式	26
3.1.4.1. 独立駆動方式	26
3.1.4.2. 直接(毎回)駆動方式	29
3.1.5. タスクのプライオリティ設定	29
3.2. 適用マイコンに応じたカスタマイズ	30
付録 A サンプル・アプリケーションの仕様	31
1. ターゲット・システム	31
2. ノード間情報通信図	31
3. 情報通信マトリクス	32
4. メッセージの説明	33
5. 機能概要	34
5.1. S810-CLG2 の場合	34

5.2. S810-CLG3, S810-CLG3-85 の場合	35
6. スケジュール・テーブル	36
6.1. スケジュール・テーブルの種類	36
6.2. メッセージ・スケジュール	37
付録- B LDF 仕様書翻訳(参考)	3

図表索引

図 1 LIN Consortium	8
図 2 出荷 CDROM のフォルダ構造	10
図 3 LIN バス・システム開発の情報の流れ	11
図 4 Lin Plan の GUI 画面の例	13
図 5 TLinLDF_DEMO_MASTER.c	17
図 6 TLinLDF_DEMO.h	18
図 7 独立駆動方式	28
図 8 直接(毎回)駆動方式	29
図 9 サンプル・アプリケーションのノード間情報通信図	31
図 10 サンプル接続試験の状況	31

1. はじめに

この文書は、『**TLIN Reference Code**』を元に、どのようにLIN 通信プログラムを構築するのかを述べるものである。

TLIN Reference Code は、LIN の仕様を理解するための参照用に設計されたものであり、本書に示す手順に従って実際の車載 ECU(電子制御装置)に組み込まれた場合に直接・間接的に発生する如何なる不具合や事故および ECU 設計を行った個人・企業がこうむる如何なる不利益に対しても、如何なる責任も保障も行わない。

2. LIN 仕様について

2.1. LIN Consortium

LIN プロトコルなどの国際的な標準となっている規格を調べる場合には、その規格を管理している委員会などの組織が発行している仕様書を、まずはじめに参照することが大事である。LIN 規格の場合では、LIN Consortium と呼ばれる委員会が、この規格の監修・発行を行っている。（下図参照）

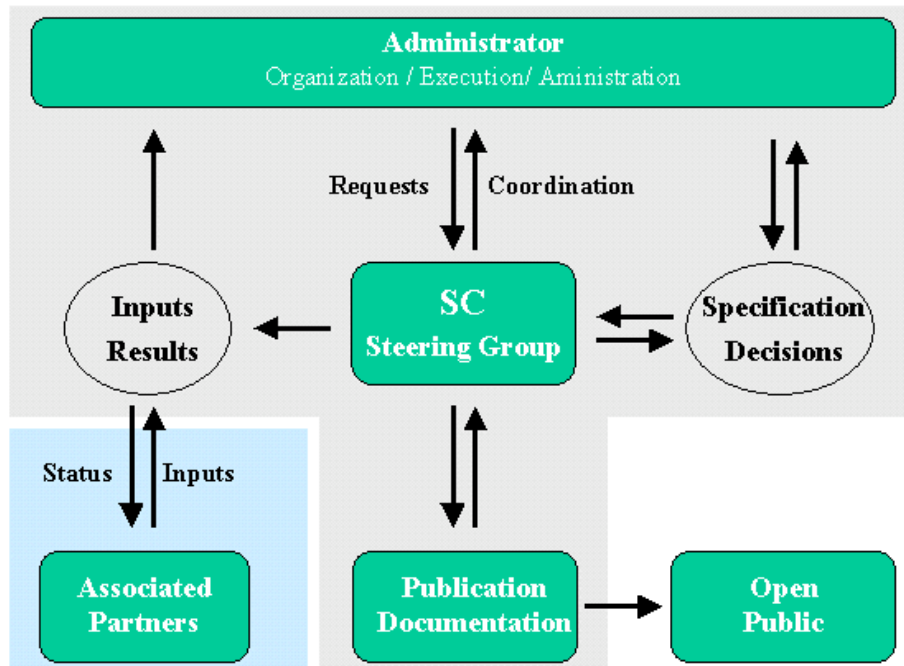


図 1 LIN Consortium

Administrator は、現在 Freescale 社（ドイツ法人）と Volcano Communications Technologies 社が担当している。上の図で、SC とあるのは、Steering Comitee の頭文字である。Steering Comitee は、現在、Audi AG, BMW AG, DaimlerChrysler AG, Volcano Communications Technologies AB, Volkswagen AG, Volvo Car Corporation などのドイツの自動車関係各社が担当している。（2006 年 12 月現在）

Assoiciated Partners は、Steering Comitee から事前に情報を得ることができ、またドラフトの内容についての提言を行うことができ、仕様の策定、改修などに、外部から協力する。

詳細は、LIN Consortium のウェブサイト参照いただきたい¹。

2.2. LIN Specification

標題に、“LIN 仕様”と書かずに、“LIN Specification”と書いたのには意味がる。

¹ <http://www.LIN-subbus.org/>

現在、LIN 仕様書の日本語訳は発行されておらず、各自動車メーカーやサプライヤなどが、独自に翻訳して、社内で参照しているのが現状のようである。しかし、初めて新しい仕様を調べる場合は、まず仕様の監修元が発行する仕様書を原語（大抵の場合英語）で読むことをお勧めする。仕様の作成者の思想や、微妙なニュアンスは原本からしか読み取れない場合が多い。

疑問が発生した場合には、必ず原文の仕様書に戻るということを強調しておきたい。**TLIN Reference Code**（以下 **TLIN** と略す）は、この原本の理解と解釈を助けるためのツールであることを、ここで強調しておきたい。

一度原書に目を通した後は、仕事の効率化のために、日本語訳を参照するのは構わないと思われるが、仕様の読み取りにおいて、微妙な表現にぶつかり、実際のインプリメンテーションでの処理の方法に迷った場合は、原本に戻るべきである。それでも、不明であれば、LIN Consortium に問い合わせるのが、市場における Compliance（互換性、相互接続性）を確保する最良の方法である。

さて、現在 LIN Consortium は、ウェブサイトからダウンロードできる形で、LIN Spec 1.3 と LIN Spec 2.0 を公開しています。最新版は、LIN Spec 2.0 であり、「新規に LIN を導入するインプリメントは、Spec 2.0 の参照を推奨する」と述べている。しかし、日本の自動車業界においては、現在 LIN Spec 1.3 をベースにして、ECU 設計を行っている。

現在の状況については、**JASPAR**² (Japan Automotive Software Platform Architecture) や **AUTOSAR**³ (AUTomotive Open System ARchitecture) などに問い合わせることをお勧めする。



² <https://www.jaspar.jp/>

³ <http://www.autosar.org/>

3. TLIN Reference Code の使い方

ここでは、**TLIN Reference Code** を実際のアプリケーションに適用するための手順を述べる。

TLIN Reference Code (以下 **TLIN** と略す) は、デモ・プログラムを含めて、完結されたアプリケーションの形で提供される。**TLIN** は、ビルド環境を **Build** フォルダとして提供している。**Build** フォルダの下に **Master** フォルダはマスタ・ノードのビルド環境を、**Slave** フォルダはスレーブ・ノードのビルド環境を提供する。それぞれ、**demo-master.c** および **demo-slave.c** のサンプル・コードが含まれている。(図 2 参照)

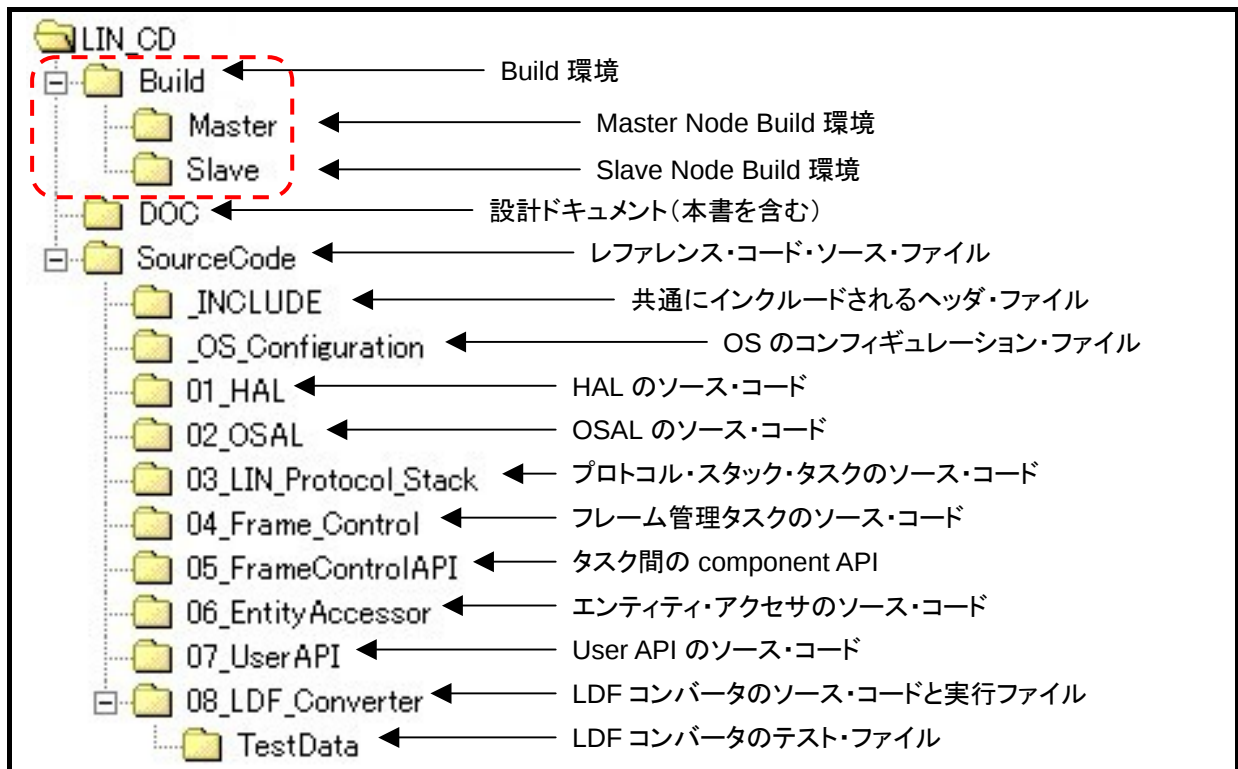


図 2 出荷 CDROM のフォルダ構造

ユーザ独自のアプリケーション構築には、この **Build** フォルダをコピーして使うことを推奨する。**SourceCode** フォルダは、**Build** フォルダで使用する各ファイルを、機能ごとに分類している。これらは、原本管理フォルダと思っていただきたい。実際のビルドは、**Build** フォルダ以下だけで行えるように配置している。

3.1. 独自アプリケーションの開発

独自のアプリケーション開発のためには、**Build** フォルダに含まれるサンプル・アプリケーション – **demo-master.c** および **demo-slave.c** を参照していただくのが簡単である。サンプル・アプリケーションの処理内容は、巻末の付録-A を参照いただきたい。また、具体的なビルド方法については、“**TLIN** レファレンス・コード ビルド・ガイド”を参照のこと。ここでは、一般的な処理の手順を紹介す

る。

3.1.1. スケジュールおよび通信フレームの設計

3.1.1.1. LDF の入手

LIN 通信を用いるシステム設計の第一歩は、まず、バス・アドミニストレータから、自ノードの入出力情報を記載した **LDF** (LIN Description File) を受け取ることから始まる。LDF が無いと設計は始まらない。

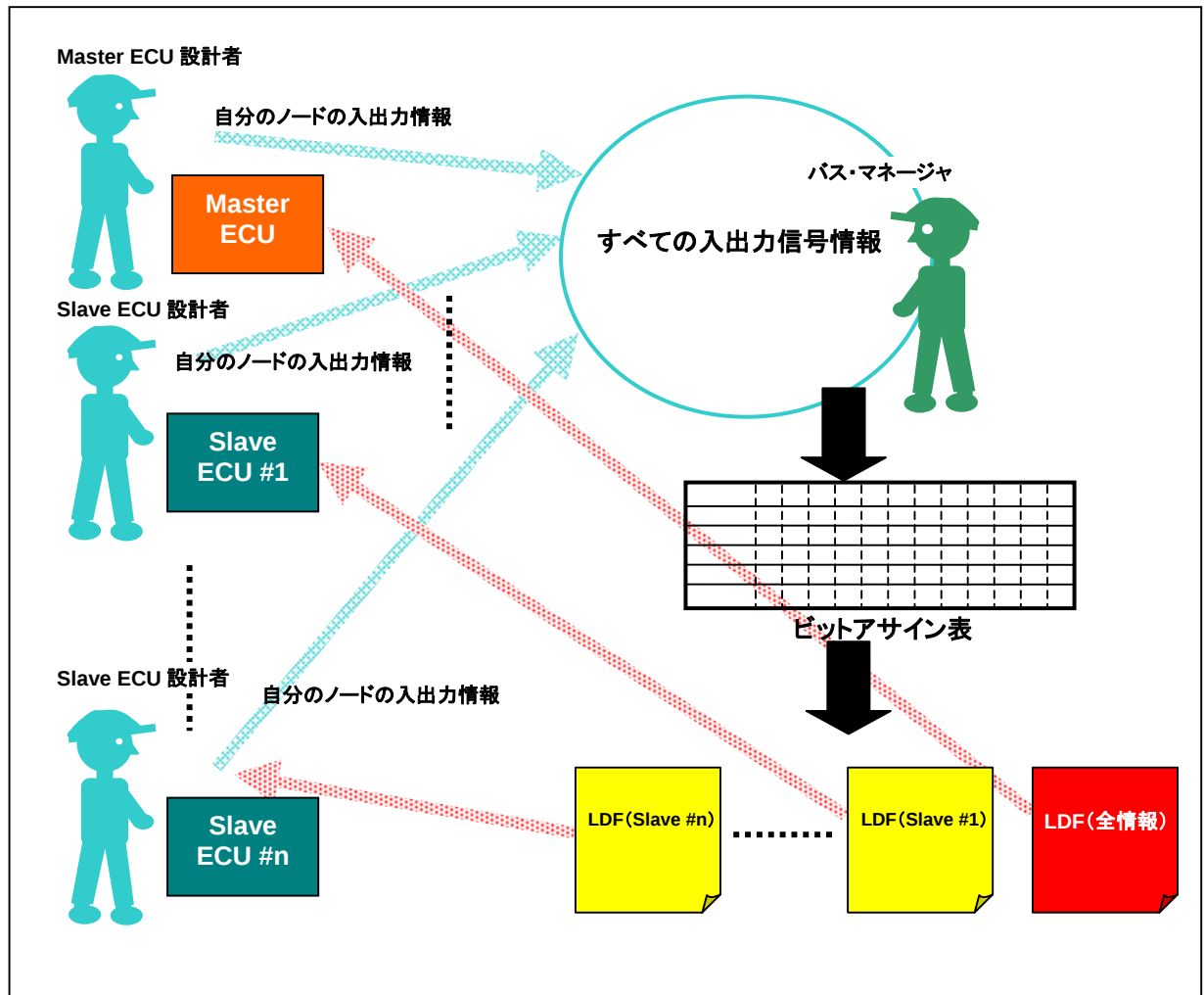


図 3 LIN バス・システム開発の情報の流れ

LIN バス通信を行なうシステムには、必ずバス・マネージャ (**BUS Manager**: 通称・バスマネ) を設けなければならない。(LIN Consortium Specification の規定である) バスマネは、各 ECU ノード設計者から、入出力仕様を聞き出し、全ての情報を得た上で、メッセージ・フレームとスケジュール・テーブルを作成する。これを元にビット・アサイン表を作成する。

作成された情報は、通常 LDF と呼ばれるフォーマットで、各 ECU 設計者に配布される。ECU 設計者は、この LDF を元にして自ノードの通信仕様を作成する。LDF については、著者の翻訳を付録-B として掲載するが、前述したように、あくまで参考として取り扱っていただきたい。原文は、LIN Consortium Specification 1.3 に含まれるものであり、付録-B は、著者の解釈により日本語化したものであり、付録-B の記載内容によりコンプライアンスの議論を行なうべきではないし、著者はこのようなトラブルに関して一切関知しないし、責任を持たない(持てない)ことを銘記しておく。

3.1.1.2. LDF から C ソースコードへの変換

バスマネから受け取った LDF は、そのままでは C ソースコードにはならない。また、場合によっては、バスマネは、LDF でなく、ビット・アサイン表などの情報を電子ファイルなど(場合によっては紙面で)供給してくる場合がある。

LDF からの変換

LDF で情報が得られた場合は、LDF 変換ツールを用いて簡単に C ソースコードに変換可能である。TLIN の場合は、LDF ツールと呼ばれる Windows 版 LDF 変換ツール(LinSrcGenerator.exe)または、OSEC oil ベースの SG(System Generator)が供給される。

現時点では、どちらが供給されるか決定されていない。

これらのツールの使い方については、それぞれの説明書を参考のこと。いずれのツールも LDF から、メッセージ・フレームやスケジュール・テーブルの ID を定義するヘッダ(XXXXX.h)と、テーブル本体(XXXXX.c)が生成される。これらのファイルを、TLIN ソースコードと、自作のアプリケーション・ソース・コードとともにコンパイルすることにより、メッセージ・フレームやスケジュール・テーブルの内部に立ち入らずに LIN 通信を行なうプログラムを生成できる。

TLIN の供給 CD の Build フォルダには、サンプルとして、TLinLDF_DEMO_MASTER.c(マスタ)、TLinLDF_DEMO_SLAVE.c(スレーブ)のテーブル・ファイルと、TLinLDF_DEMO.h(マスタ/スレーブ共通)ファイルのサンプルが含まれている。

ビット・アサイン表からの変換

バスマネがビット・アサイン表しかくれない場合(現実的には、このケースが多い)や、試作のために、自分でビット・アサイン表を作成した場合には、2つの手段がある。

① 自作 LDF を作成する

ビット・アサイン表を見ながら、エディタで自分で LDF を作る。付録-B として LDF の仕様を掲載したのは、このケースを想定したものである。

この方法は、LDF の理解を確実にする。LDF は、文法的には簡単な言語であるので、C プログラマならば、容易に理解できると推定する。また、他の市販ツールへの入力形式として一般的であるので、市販シミュレータなどを使用する場合に便利である。

② 市販ツールを使う

LDF の文法を理解して、エディタで自作するのが面倒な場合には、市販の LDF 作成スールなどを用いればよいであろう。例えば、TTTech (Time Triggered Technology⁴) 社の LIN Plan⁵などを利用する手がある。LIN Plan は、バスマネ用のツールであり、全てのノードの入出力情報を元に、GUI を用いて効果的にスケジュールとメッセージ・フレームの構成を行なうツールである。

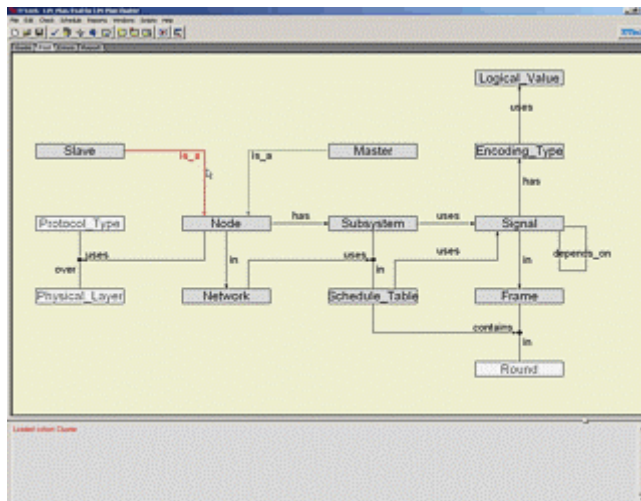


図 4 Lin Plan の GUI 画面の例

③ 自分で C ソース・コードを入力する

慣れてくれば、最も早い手法である。

図 5, 図 6 に、付録-B で説明するサンプル・アプリケーションと、そのビット・アサイン表を掲載している。このサンプル・アプリケーションで用いているビット・アサイン表に対応するソース・コードである。両方見比べていただくと、テーブルの意味付けが容易に理解していただけると考える。

⁴ <http://www.tttech.de/index.htm>

⁵ <http://www.tttech.de/products/software/linplan/overview.htm>

```

/*****
*/
/** File:   TLinLDF.c                               */
/** Update: Rev0000 (2006.02.15) VirtualCode & Comment      */
/** Update: Rev0001 (2006.12.20) VirtualCode & Comment      */
/*****
*/

#include "TLinTypes.h"
#include "TLinEntity.h"
#include "TLinLDF_DEMO.h"

const TLinSigRcd gCTLinSignalTable[] =
{
/* flg      size  val */
  { 0,    0,    0 },      // 0: null object

  { 0,    0,    0 },      // 1: T_SIGHND_DM4      1bit
  { 0,    0,    0 },      // 2: T_SIGHND_DM2      1bit
  { 0,    3,    0 },      // 3: T_SIGHND_DM5_8    4bit
  { 0,    0,    0 },      // 4: T_SIGHND_DM3      1bit

  { 0,    6,    0 }      // 5: T_SIGHND_DS2_8    7bit
};

const TLinSigLnk gCTLinSignalOffsetTable[] =
{
/* hnd_sig      offset */
  { T_SIGHND_NULL,    0 },      // 0: Null
  { T_SIGHND_DS2_8,    8 },      // 1: T_FRMHND_DIP_SLV1 - 9
  { T_SIGHND_DM4,    0 },      // 2: T_FRMHND_DIP_MST1 - 9
  { T_SIGHND_DM2,    8 },      // 3:
  { T_SIGHND_DM5_8,   32 },      // 4:
  { T_SIGHND_DM3,    56 }      // 5:
};

const TLinFrmRcd gCTLinFrameTable[] = {
/* brseponse  id  sig_num  sig_base  size */
  { 0,    0,    0,    0,    0 },      // 0: Null
  { 0,    0x12,  1,    1,    2 },      // 1: T_FRMHND_DIP_SLV1
  { 0,    0x13,  1,    1,    2 },      // 2: T_FRMHND_DIP_SLV2
  { 0,    0x14,  1,    1,    2 },      // 3: T_FRMHND_DIP_SLV3
  { 0,    0x15,  1,    1,    2 },      // 4: T_FRMHND_DIP_SLV4
  { 0,    0x16,  1,    1,    2 },      // 5: T_FRMHND_DIP_SLV5
  { 0,    0x17,  1,    1,    2 },      // 6: T_FRMHND_DIP_SLV6
  { 0,    0x18,  1,    1,    2 },      // 7: T_FRMHND_DIP_SLV7
  { 0,    0x19,  1,    1,    2 },      // 8: T_FRMHND_DIP_SLV8
  { 0,    0x1A,  1,    1,    2 },      // 9: T_FRMHND_DIP_SLV9
  { 1,    0x31,  4,    2,    8 },      // 10: T_FRMHND_DIP_MST1
  { 1,    0x32,  4,    2,    8 },      // 11: T_FRMHND_DIP_MST2
  { 1,    0x33,  4,    2,    8 },      // 12: T_FRMHND_DIP_MST3
  { 1,    0x34,  4,    2,    8 },      // 13: T_FRMHND_DIP_MST4
  { 1,    0x35,  4,    2,    8 },      // 14: T_FRMHND_DIP_MST5
  { 1,    0x36,  4,    2,    8 },      // 15: T_FRMHND_DIP_MST6
  { 1,    0x37,  4,    2,    8 },      // 16: T_FRMHND_DIP_MST7
  { 1,    0x38,  4,    2,    8 },      // 17: T_FRMHND_DIP_MST8
  { 1,    0x39,  4,    2,    8 }      // 18: T_FRMHND_DIP_MST9
};

const TLinSchRcd gCTLinSchduleRecordTable[] = {
/* hnd_frm      delay */

```

```

{ T_FRMHND_NULL,          0 }, // 0:
{ T_FRMHND_DIP_SLV1,     12 }, // 1: Mode_1
{ T_FRMHND_DIP_MST9,     26 }, // 2:
{ T_FRMHND_DIP_SLV2,     12 }, // 3:
{ T_FRMHND_DIP_MST8,     26 }, // 4:
{ T_FRMHND_DIP_SLV3,     12 }, // 5:
{ T_FRMHND_DIP_MST7,     26 }, // 6:
{ T_FRMHND_DIP_SLV4,     12 }, // 7:
{ T_FRMHND_DIP_MST6,     26 }, // 8:
{ T_FRMHND_DIP_SLV5,     12 }, // 9:
{ T_FRMHND_DIP_MST5,     26 }, // 10:
{ T_FRMHND_DIP_SLV6,     12 }, // 11:
{ T_FRMHND_DIP_MST4,     26 }, // 12:
{ T_FRMHND_DIP_SLV7,     12 }, // 13:
{ T_FRMHND_DIP_MST3,     26 }, // 14:
{ T_FRMHND_DIP_SLV8,     12 }, // 15:
{ T_FRMHND_DIP_MST2,     26 }, // 16:
{ T_FRMHND_DIP_SLV9,     12 }, // 17:
{ T_FRMHND_DIP_MST1,     26 }, // 18:
{ T_FRMHND_DIP_SLV1,     10 }, // 19: Mode_2
{ T_FRMHND_DIP_MST1,     16 }, // 20:
{ T_FRMHND_DIP_SLV9,     10 }, // 21:
{ T_FRMHND_DIP_MST9,     16 }, // 22:
};

const TLinSchTbl gCTlinSchduleTable[] = {
/* frm_num frm_baseHnd */
{ 0,      0 }, // 0:
{ 18,     1 }, // 1: T_SCHHND_Mode1
{ 4,      19 }, // 2: T_SCHHND_Mode2
};

const TLinResRcd gCTlinResponseRecord[] = {
/* hnd_frm */
{ T_FRMHND_NULL }, // 0:
{ T_FRMHND_DIP_SLV1 }, // 1:
{ T_FRMHND_DIP_SLV2 }, // 2:
{ T_FRMHND_DIP_SLV3 }, // 3:
{ T_FRMHND_DIP_SLV4 }, // 4:
{ T_FRMHND_DIP_SLV5 }, // 5:
{ T_FRMHND_DIP_SLV6 }, // 6:
{ T_FRMHND_DIP_SLV7 }, // 7:
{ T_FRMHND_DIP_SLV8 }, // 8:
{ T_FRMHND_DIP_SLV9 }, // 9:
{ T_FRMHND_DIP_MST1 }, // 10:
{ T_FRMHND_DIP_MST2 }, // 11:
{ T_FRMHND_DIP_MST3 }, // 12:
{ T_FRMHND_DIP_MST4 }, // 13:
{ T_FRMHND_DIP_MST5 }, // 14:
{ T_FRMHND_DIP_MST6 }, // 15:
{ T_FRMHND_DIP_MST7 }, // 16:
{ T_FRMHND_DIP_MST8 }, // 17:
{ T_FRMHND_DIP_MST9 }, // 18:
};

const TLinResTbl gCTlinResponseTable[] = {
/* frm_num h_baseHnd */
{ 0,      0 },
{ 18,     1 },
};

const TLinDiagRcd gCTlinDiagTable[] = {

```

```

/* bUseDiag  bResponse  u8Adr  aSendData[8]  aRecvData[8]
*/
{ 1,      1,      0x20,  {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}},
{ 1,      0,      0x40,  {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}}
};

/* data allocation (RAM) */
TlinSigRcd    gTlinSignalTable[6];
TlinSigLnk    gTlinSignalOffsetTable[6];
TlinFrmRcd    gTlinFrameTable[19];
TlinSchRcd    gTlinSchduleRecordTable[23];
TlinSchTbl    gTlinSchduleTable[3];
TlinResRcd    gTlinResponseRecord[19];
TlinResTbl    gTlinResponseTable[2];
TlinDiagRcd    gTlinDiagTable[2];
TlinEntityRcd gTlinEntityRcd;

/* Initialize Entity (RAM initialize) */
void tlinEntityInit_ldf( void )
{
    int i;
    /* gTlinSignalTable <= gCTlinSignalTable */
    for( i = 0; i < 6; ++i ) {
        gTlinSignalTable[i].flag = gCTlinSignalTable[i].flag;
        gTlinSignalTable[i].size = gCTlinSignalTable[i].size;
        gTlinSignalTable[i].val  = gCTlinSignalTable[i].val;
    }

    /* gTlinSignalOffsetTable <= gCTlinSignalOffsetTable */
    for( i = 0; i < 6; ++i ) {
        gTlinSignalOffsetTable[i].hnd_sig =
gCTlinSignalOffsetTable[i].hnd_sig;
        gTlinSignalOffsetTable[i].offset = gCTlinSignalOffsetTable[i].offset;
    }

    /* gTlinFrameTable <= gCTlinFrameTable */
    for( i = 0; i < 19; ++i ) {
        gTlinFrameTable[i].bResponse = gCTlinFrameTable[i].bResponse;
        gTlinFrameTable[i].id        = gCTlinFrameTable[i].id;
        gTlinFrameTable[i].sig_num   = gCTlinFrameTable[i].sig_num;
        gTlinFrameTable[i].sig_base  = gCTlinFrameTable[i].sig_base;
        gTlinFrameTable[i].size      = gCTlinFrameTable[i].size;
    }

    /* gTlinSchduleRecordTable <= gCTlinSchduleRecordTable */
    for( i = 0; i < 23; ++i ) {
        gTlinSchduleRecordTable[i].hnd_frm =
gCTlinSchduleRecordTable[i].hnd_frm;
        gTlinSchduleRecordTable[i].delay  =
gCTlinSchduleRecordTable[i].delay;
    }

    /* gTlinSchduleTable <= gCTlinSchduleTable */
    for( i = 0; i < 3; ++i ) {
        gTlinSchduleTable[i].frm_num      = gCTlinSchduleTable[i].frm_num;
        gTlinSchduleTable[i].frm_baseHnd = gCTlinSchduleTable[i].frm_baseHnd;
    }

    /* gTlinResponseRecord <= gCTlinResponseRecord */
    for( i = 0; i < 19; ++i ) {
        gTlinResponseRecord[i].hnd_frm = gCTlinResponseRecord[i].hnd_frm;
    }
}

```

```

/* gTlinResponseTable <= gTlinResponseTable */
for( i = 0; i < 2; ++i ) {
    gTlinResponseTable[i].frm_num = gCTlinResponseTable[i].frm_num;
    gTlinResponseTable[i].h_baseHnd = gCTlinResponseTable[i].h_baseHnd;
}

/* gTlinDiagTable <= gCTlinDiagTable */
for( i = 0; i < 2; ++i ) {
    gTlinDiagTable[i].bUseDiag = gCTlinDiagTable[i].bUseDiag;
    gTlinDiagTable[i].bResponse = gCTlinDiagTable[i].bResponse;
    gTlinDiagTable[i].u8Addr = gCTlinDiagTable[i].u8Addr;
}

/* gTlinEntityRcd */
gTlinEntityRcd.num_sig_rcd = 6;
gTlinEntityRcd.num_sig_lnk = 6;
gTlinEntityRcd.num_frm_rcd = 19;
gTlinEntityRcd.num_sch_rcd = 23;
gTlinEntityRcd.num_sch_tbl = 3;
gTlinEntityRcd.num_res_rcd = 19;
gTlinEntityRcd.num_res_tbl = 2;
gTlinEntityRcd.tbl_sig_rcd = gTlinSignalTable;
gTlinEntityRcd.tbl_sig_lnk = gTlinSignalOffsetTable;
gTlinEntityRcd.tbl_frm_rcd = gTlinFrameTable;
gTlinEntityRcd.tbl_sch_rcd = gTlinSchduleRecordTable;
gTlinEntityRcd.tbl_sch_tbl = gTlinSchduleTable;
gTlinEntityRcd.tbl_res_rcd = gTlinResponseRecord;
gTlinEntityRcd.tbl_res_tbl = gTlinResponseTable;
gTlinEntityRcd.num_diag_rcd = 2;
gTlinEntityRcd.tbl_diag_tbl = gTlinDiagTable;
}

```

图 5 TLinLDF DEMO MASTER.c

```

/*****
*/
/** File:   TLinLDF.h                               ***/
/** Update: Rev0000 (2006.02.15) VirtualCode & Comment      ***/
/*****
*/
#define T_IFCHND_1      1
#define T_RESHND_1      1

#define T_SIGHND_NULL    0 // 0:
#define T_SIGHND_DM4     1 // 1: 1bit
#define T_SIGHND_DM2     2 // 2: 1bit
#define T_SIGHND_DM5_8   3 // 3: 4bit
#define T_SIGHND_DM3     4 // 4: 1bit
#define T_SIGHND_DS2_8   5 // 5: 7bit

#define T_FRMHND_NULL    0 // 0:
#define T_FRMHND_DIP_SLV1 1 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV2 2 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV3 3 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV4 4 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV5 5 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV6 6 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV7 7 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV8 8 // 1: Slave Dip switch
#define T_FRMHND_DIP_SLV9 9 // 1: Slave Dip switch
#define T_FRMHND_DIP_MST1 10 // 2: Master Dip switch
#define T_FRMHND_DIP_MST2 11 // 2: Master Dip switch
#define T_FRMHND_DIP_MST3 12 // 2: Master Dip switch
#define T_FRMHND_DIP_MST4 13 // 2: Master Dip switch
#define T_FRMHND_DIP_MST5 14 // 2: Master Dip switch
#define T_FRMHND_DIP_MST6 15 // 2: Master Dip switch
#define T_FRMHND_DIP_MST7 16 // 2: Master Dip switch
#define T_FRMHND_DIP_MST8 17 // 2: Master Dip switch
#define T_FRMHND_DIP_MST9 18 // 2: Master Dip switch

#define T_SCHHND_NULL    0
#define T_SCHHND_Mode1   1
#define T_SCHHND_Mode2   2

```

6 TLinLDF DEMO.h

3.1.2. アプリケーション・プログラムの書き方

前項の説明の手順で、LIN 通信のテーブル・ファイルが作成できれば、いよいよアプリケーション・プログラムの作成となる。本項では、サンプルの demo-master.c を参照しながら説明する。

3.1.2.1. include ファイル

コンフィギュレーション

インクルードするファイルは、自作のものを含めて、特に手順は無いが、唯一 TlinCfg.h は、先頭に記述されることを推奨する。

このファイルは、ターゲット・ボードの選定他、種々のコンフィギュレーション情報を記載したものである。コンフィギュレーションの実際の内容については、別書“TLIN レファレンス・コード プログラム・ビルド・ガイド”に詳細を記載してあるので、参照されたい。

これを先頭に記載してないと動作しない訳ではないが、このほうが安全である。

```
/*#####  
#  
Copyright(C) 2006, 2007 OTSL Inc. All Rights Reserved. This is UNPUBLISHED  
PROPRIETARY SOURCE CODE of OTSL Inc., No part of this file may be  
copied, modified, sold, and distributed in any form or by any means without  
prior explicit permission in writing from OTSL Inc.  
<NAME> demo.c  
<Layer> Demo & Evaluation Program for TLIN Master node.  
<DESCRIPTION>  
<NOTES>  
<BUGS>  
<Author>  
<Version> 1.6.2_ST  
<Date> 2007.02.26  
  
#####*/  
/  
/*=====
```

Include Configuration File

```
=====*/  
#include "TlinCfg.h"  
#ifndef __NO_OS__  
    #include <mr30.h>  
#endif
```

次行の `#ifndef __NO_OS__` に続く 3 行は、OS を使用し、OS のシステム・コールをアプリケーションが使用する場合に必要な OS のヘッダ・ファイルである。例では、ルネサス製 MR30 を使用した場合の記述であり、この部分は自分が使用する OS によって変更される。

SFR 定義

以下は、使用するマイコンの SFR 定義のインクルードである。例では、M16C6N4 (S810-CLG2), M16C6NK (S810-CLG3), M32C85 (S810-CLG3-85) の切り替えを行なっている。またそれぞれの SFR 定義ファイル・sfr6n4.h, sfr6nk.h, sfr32c85.h を添付供給してい

る。これ以外のマイコンを使う場合は、メーカーのウェブ・サイトからダウンロードするか、自作すればよい。

```
/*=====
   Include SFR definitions according to board type
   =====*/
#ifdef __CLG2__
    #include "sfr6n4.h"
#endif
#ifdef __CLG3__
    #include "sfr6nk.h"
#endif
#ifdef __CLG3_85__
    #include "sfr32c85.h"
#endif
```

TLIN 型定義

TLIN では独自のデータ型を定義しているため、その定義ファイルをインクルードする必要がある。例えば、エンティティ(フレーム中の送受信情報)の書き換えを行なう API 関数を用いるためには、このヘッダは必須である。

```
/*=====
   基本変数型, 定義マクロ, 構造体等の定義ファイル
   一応読み込んだ方が安全です。
   =====*/
#include "TLinTypes.h"
#include "TLinDefs.h"
#include "TLinTypedef.h"
```

API およびエンティティ・アクセサ

API 関数およびエンティティ・アクセサ(API 関数の一部)を用いる場合に必要。必須である。

```
/*=====
   API とエンティティ・アクセサの定義ファイル
   =====*/
#include "TLinAPI.h"
#include "TLinEntity.h"
```

OSAL, HAL, スタック内部変数の定義ファイル

OSAL や HAL を直接呼ぶ場合に必要。それ以外では不要である。

```
/*=====
   Include HAL / OSAL / LIN Protocol Stack definitions
   OSAL / HAL や、スタックの内部関数を直接使わない場合は不要
   =====*/
#include "TLIN_HAL_if.h"
#include "TLIN_OSAL_if.h"
#include "tlin_stack_defs.h"
```

SG 作成ヘッダ

SG (System Generator) や LDF コンバータ・ツールの出力ヘッダ。自作した場合は、自作のヘッダとなる。必須である。

```
/*=====
SG(ジェネレータ)作成のサンプル・ヘッダ
シグナル, フレーム, スケジュールの定義
インタフェース・ハンドルの定義
=====*/
#include "TLinLDF_DEMO.H"
```

3.1.2.2. その他一般的な定義文

一般的に、マクロ定義, プロトタイプ宣言, 外部参照宣言, グローバル変数定義などが記述される。(もちろん自作ヘッダ・ファイル)にまとめるのも自由) 以下は、参考として掲載するにすぎない。

```
/*=====*/
/* Define 宣言 */
/*=====*/
#define ALL_OUTPUT ( unsigned char )(0xff) /* 出力定義 */
#define OUTPUT ( unsigned char )(1) /* 出力定義 */
#define INPUT ( unsigned char )(0) /* 入力定義 */

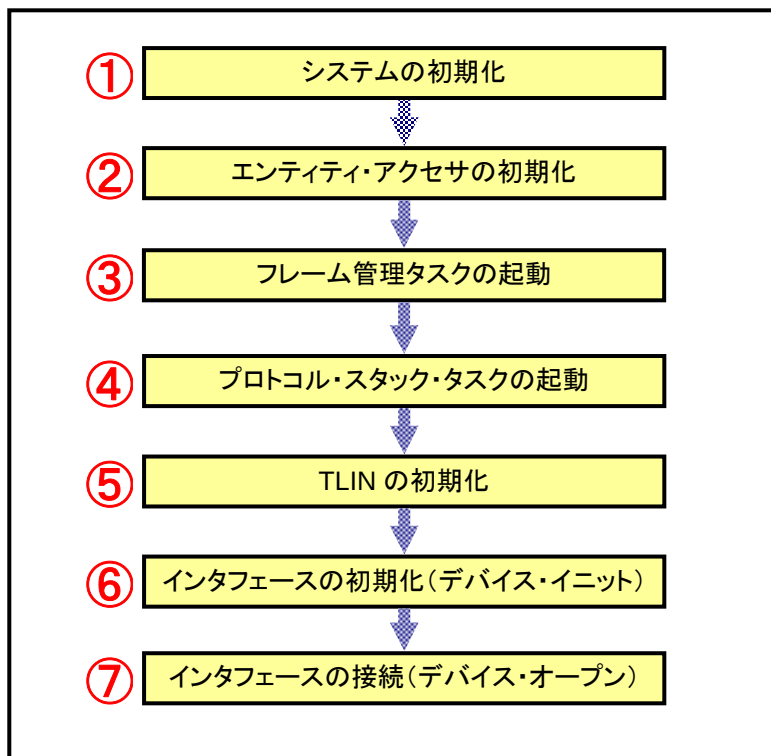
/*=====*/
/* Prototype 宣言 */
/*=====*/
void main (void);
void task2(void);
void task3(void);
/*=====*/
/* Extern 宣言 */
/*=====*/
extern T_TLIN_FRMSTM_CNT g_tlin_frmstm_info_01; /* FRMSTM Control 構造体の実体 01 */
/*=====*/
/* Gloval Values and Tables */
/*=====*/

(途中省略)

/*=====
DIP Switch の状態変数
=====*/
int DIP1_STATUS;
int DIP2_STATUS;
int DIP3_STATUS;
int DIP4_STATUS;
int DIP5_STATUS;
int DIP6_STATUS;
int DIP7_STATUS;
int DIP8_STATUS;
```

3.1.3. 初期化

TLIN アプリケーションでは、以下の手順で初期化を行なう。



3.1.3.1. システムの初期化

最初に行なうのは、マイコンおよびシステムの初期化である。下のサンプルでは、マイコンの初期化部分が見られないが、これは、MR30 RTOS の準備する初期化ルーチン CRT0MR.A30 に初期化をまかせているためである。

```
/*=====
main ( )
=====*/
void main()
{
    int i;
    TLinU08 is_ok;
    TLinU08 led_data;

    #ifdef __CLG2__
        p0 = 0x0ff;          /* P0 output latch <--- H */
        pd0 = INPUT;         /* Dip Switch Input */
        p1 = 0;              /* Port 0 <--- all OFF */
        pd1 = ALL_OUTPUT;    /* LED port direction */
    #else
        p4 = 0x80;           /* Dot only ON */
        LED_data1 = 5;
        LED_data2 = 0;
        LED_data3 = 0;
        LED_data4 = 0;

        pd4 = ALL_OUTPUT;

        p3 = 0xfe;           /* LED1 select */
        pd3 = ALL_OUTPUT;    /* LED selector port direction */

        pd2 = INPUT;         /* Dip Switch Input */
    #endif

    DIP1_STATUS = 0;
    DIP2_STATUS = 0;
    DIP3_STATUS = 0;
    DIP4_STATUS = 0;
    DIP5_STATUS = 0;
    DIP6_STATUS = 0;
    DIP7_STATUS = 0;
    DIP8_STATUS = 0;
}
```

①

3.1.3.2. エンティティ・アクセサの初期化

この処理は、現在 API 仕様に記述されていない⁶。しかし、必ず必要である。

```
/*=====
/* Entity Accesor Init */
/*=====
_tlin_Ent_Initialize();
```

②

⁶ ROM 化を行なうために、この初期化が必要であることを後ほど議論が出たため、処理関数は組み込んであるが、仕様書の改版が遅れているため。次リリースでは、API 仕様書に含まれる予定である。

3.1.3.3. フレーム管理タスクとスタック・タスクの起動

ここまでの時点では、TLINを構成する2つのタスク・フレーム管理タスク(ID_tlin_frmcon_tsk)とスタック・タスク(ID_tlin_stack_tsk)は、まだ起動してはいけない。エンティティ・アクセサの初期化を行なわない状態で、タスク起動をかけると、エンティティへのアクセスが開始され、動作保証ができない。

また、この2つのタスクは、必ずこの順番に起動しなければならない。(フレーム管理タスクが先)スタック・タスクが割り込み検知を行い、フレーム管理タスクへメッセージを投げるからである。一方、フレーム管理タスクは、アプリケーションから指示が来ない限り、スタック・タスクへメッセージ送出行なわない。

また、これらは、次項に述べる TLIN の初期化 API 発行の前に行なわねばならない。

```
/* ***** */
/* Fram Control Task Start */
/* ***** */
sta_tsk( ID_tlin_frmcon_tsk, 1);

/* ***** */
/* Stack Task Start */
/* ***** */
sta_tsk( ID_tlin_stack_tsk, 1 );
```

③

④

3.1.3.4. TLIN の初期化

TLIN の初期化は、tlin_sys_init API コールによって行なわれる。この API コールは、フレーム管理タスクによって受理され、フレーム管理タスクから、スタック・タスクへのメッセージとして実現される。したがって、この API コールは、タスクが起動された後で、全ての API コールに先立ち、一回のみ実行されるべきである。

```
/* ***** */
/* API : Sys_init */
/* ***** */
is_ok = tlin_sys_init();
```

⑤

3.1.3.5. インタフェースの初期化と接続

インタフェースの初期化は、API コール *tlin_ifc_init* によって行なわれる。これにより、LIN 通信を行なうマイコン内部ハードウェア(タイマ, ポート)が初期化される。しかし、これらは初期化されるのみであり、まだ動作をしていない。つまり、バスに他のノードからの信号が現れても、まったく反応しない状態である。

API コール *tlin_ifc_connect* が実行されて、全ての通信のためのハードウェアが動作を開始する。

```

/*****
/* API : IFC_init
/*****
interface_handle = T_IFCHND_1;
response_handle = T_RESCHND_1;
tlin_ifc_init(interface_handle, response_handle, TLIN_BOOL_TRUE);

/*****
/* API : IFC_connect
/*****
is_ok = tlin_ifc_connect( interface_handle );
```

以上の手順は、一連の動作として、サンプル・ソース・コードの記述を変更せずに、そのまま使用することをお勧めする。

3.1.4. 通信駆動方式

前項までの処理で、TLIN は既に動作可能状態である。

スレーブ・ノードの場合は、これ以上何もすることはない。エンティティ・アクセス用の API コールを用いて、自ノード独自の処理結果をエンティティに書き込み、またはエンティティを読み出し、その値によって定義された動作をするのみである。

通信のタイミングは、すべて TLIN スタックにまかせ、アプリケーションは通信のタイミングをまったく意識する必要は無い。

マスタ・ノードの場合は、自分でスケジュールをコントロールする(つまり、各ノードに対するポーリングを行なう)必要がある。

まず行なう必要があるのは、使用するスケジュール・テーブルを TLIN スタックに教えることである。これは、API コール `tlin_ifc_sch_set` によって行なわれる。これで、いつでもスケジュールの開始が可能になる。

スケジュール・テーブルを切り替える場合には、切り替えるスケジュール・テーブルのハンドルをセットして同じ API コール `tlin_ifc_sch_set` を行なう。タイミングを関知する必要は無く、TLIN スタックは、実行中のメッセージ・フレームが終了した時点で、指定されたスケジュール・テーブルの先頭のフレームからスケジューリングを開始する。

```
/* ***** */
/*  API : Schedule_set          */
/* ***** */
schedule_handle = T_SCHHND_Model1;
tlin_ifc_sch_set(interface_handle, schedule_handle, 1);
```

3.1.4.1. 独立駆動方式

サンプル・コードでは、以下のように別タスク・ID_task3 を起動している。

```
/* ***** */
/*  Scheduling Start          */
/* ***** */
sta_tsk( ID_task3, 1 );    /* LED Dynamic Display */
```

ID_task3(task3())関数の OS 上の ID)は、以下のような記述になっている。

```
/*=====
task3( )
    LED1 - LED4 Dynamic Drive & tick
=====*/
void task3(void)
{
    int i;

    digit = 3;
```

```

while(1)
{
    dly_tsk( 3 );

    digit++;
    if(digit >= 4)
    {
        digit = 0;
    }

    #ifndef __CLG2__
    switch ( digit )
    {
        case 0:
            p3 = 0xff;
            p4 = dot_seg | led_encode[LED_data1];
            p3 = 0xfe;
            break;
        case 1:
            p3 = 0xff;
            p4 = dot_seg | led_encode[LED_data2];
            p3 = 0xfd;
            break;
        case 2:
            p3 = 0xff;
            p4 = dot_seg | led_encode[LED_data3];
            p3 = 0xfb;
            break;
        default:
            p3 = 0xff;
            p4 = dot_seg | led_encode[LED_data4];
            p3 = 0xf7;
            break;
    }
    #endif

    /* tick */
    /* tick */
    if( digit == 0 )
    {
        tlin_ifc_sch_tick(interface_handle);
    }
}
}

```

重要なのは、最後の `tlin_ifc_sch_tick` 関数のみである。TLIN は、この `tlin_ifc_sch_tick` がコールされるたびに、スケジュール・テーブルにあるメッセージ・フレームを次々に実行する。ただし、あるメッセージ・フレームを実行中に来た `tlin_ifc_sch_tick` は、保留され、次のメッセージ・フレームへの起動予約として認識され、現在実行中のメッセージ・フレームが終了（指定されたスロット時間が経過）した時点で、間を置かず次のメッセージ・フレームを実行する。

メッセージ・フレーム実行中に受け取った `tlin_ifc_sch_tick` の保留は、2 回目以降何度受け取っても、次の 1 回分の保留として受け取る。従って、最小スロット時間よりも短い間隔で `tlin_ifc_sch_tick` を発行してやれば、アプリケーションは、スケジュール管理についても、まったく関与せず、自らの処理を行い、エンティティの更新を行えばよい訳である。

`task3()` では、ほぼ 10msec 毎に `tlin_ifc_sch_tick` を発行するようになっている。この結果、サン

プル・アプリケーションでは、通信には、まったく関与せずに、ディップ・スイッチの値を読み込んで送信エンティティに書き込み、受信エンティティを読み出しては LED に表示する処理を繰り返すのみとなっている。TLIN の設計思想は、このような独立駆動方式である。

この処理を模擬的に図示すると以下ようになる。

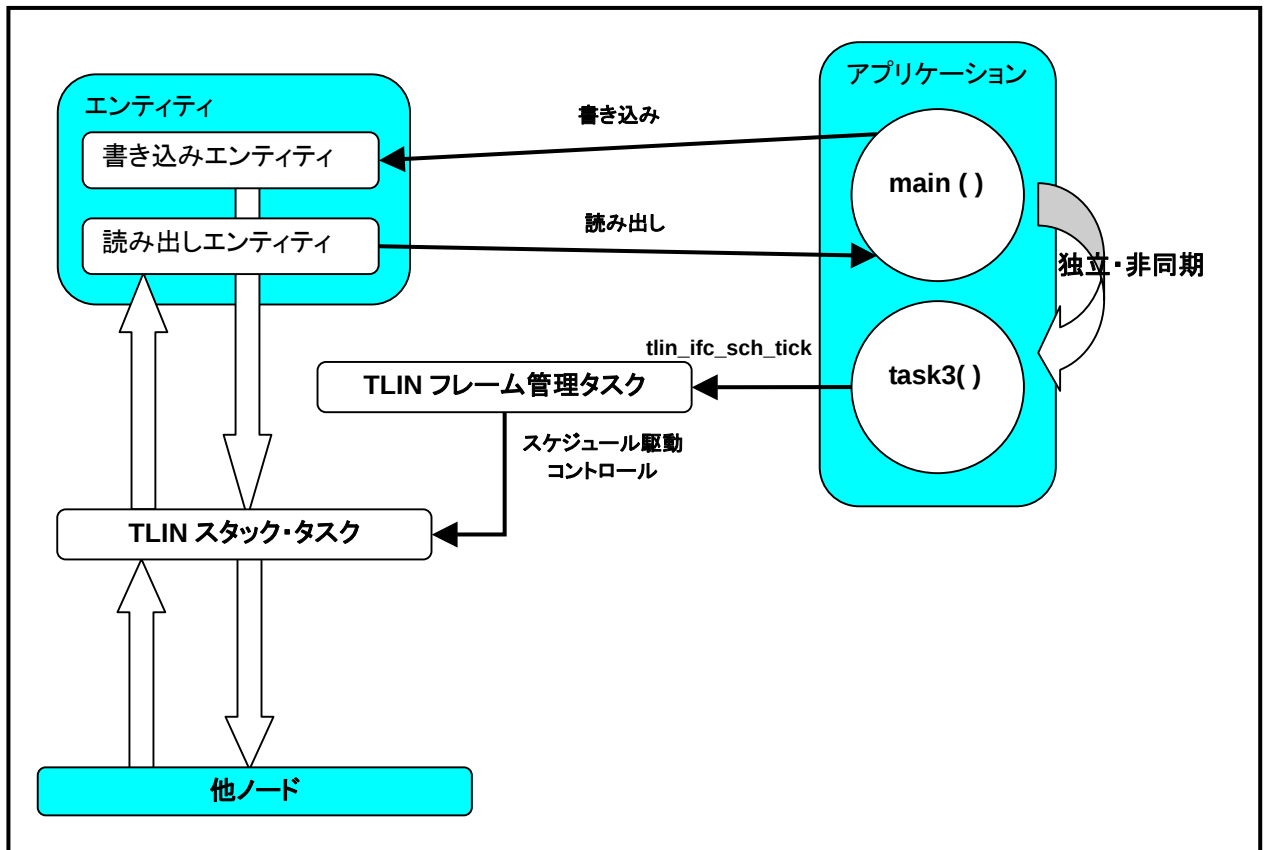


図 7 独立駆動方式

3.1.4.2. 直接(毎回)駆動方式

直接駆動方式とは、アプリケーション(main)タスクが、毎回 `tlin_ifc_sch_tick` を発行する方法である。以下のような手順になる。

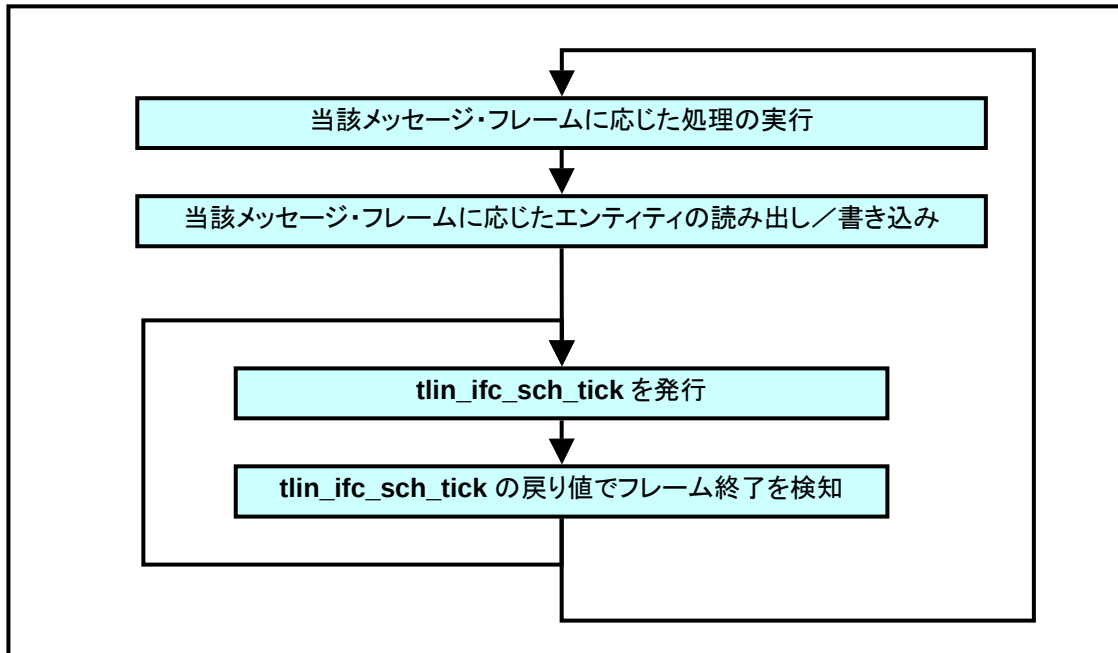


図 8 直接(毎回)駆動方式

これらは、時と場合により使い分ければよい。しかし、通常は独立駆動方式を取ることを推奨する。

3.1.5. タスクのプライオリティ設定

述べるまでもないが、OS を使用する際には、各タスクにプライオリティを設定する。TLIN においては、プライオリティを以下のように設定することを推奨する。

スタック・タスク > フレーム管理タスク > アプリケーション・タスク

3.2. 適用マイコンに応じたカスタマイズ

適用マイコンに応じたカスタマイズについては、“TLIN ビルド・ガイド”のコンフィギュレーションを参照のこと。

本書で参考として挙げたマイコンや異なるボードを用いる場合は、HAL を自作する必要がある。ここで HAL と呼ぶのは、以下のソース・コードである。

- TLIN_HAL.c
- m16c_uart0.c
- m16c_uart2.c

付録 A サンプル・アプリケーションの仕様

1. ターゲット・システム

サニー技研製・S810-CLG2, S-810-CLG3, S810-CLG3-85 の 3 品種に適応する。

2. ノード間情報通信図

サンプル・アプリケーションは、マスタとスレーブの最も簡単なシステム構成において、お互いの DIP Switch 情報を読み取り、LED に表示する。

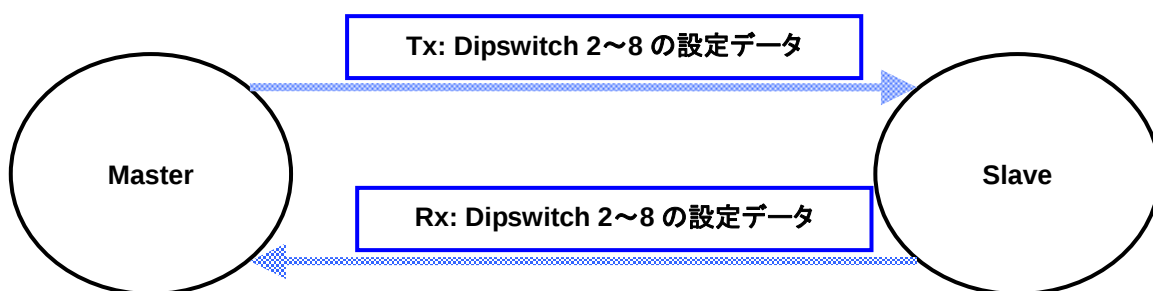


図 9 サンプル・アプリケーションのノード間情報通信図

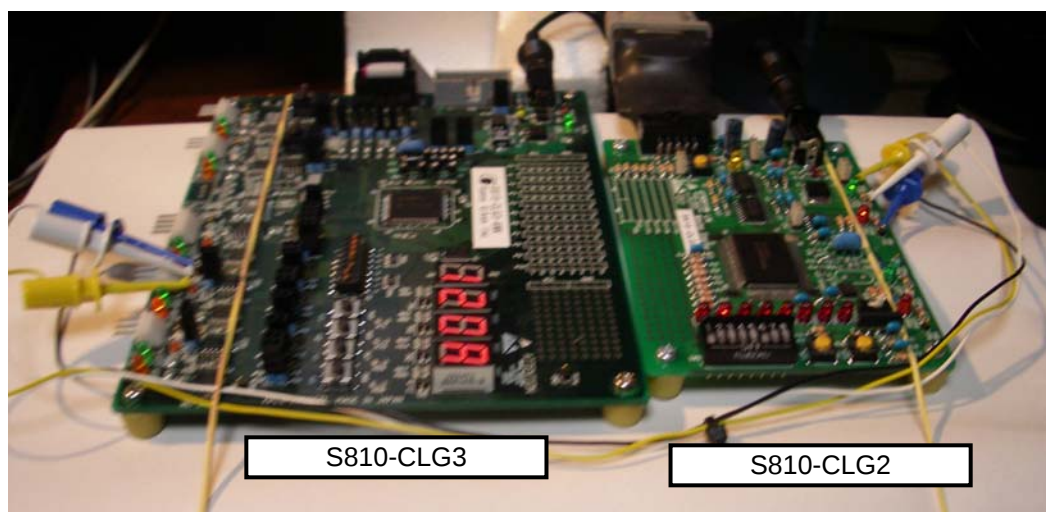


図 10 サンプル接続試験の状況

3. 情報通信マトリクス

以下にノード間通信の情報マトリクスを示す。

表 1 サンプルアプリケーションの情報通信マトリクス

ID パリティ無し	ID パリティ込み	メッセージ名	値	メッセージの 送信先	メッセージ長 (ms)	サイズ (バイト)
0x12	0x92	T_FRMHND_DIP_SLV1	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x13	0xD3	T_FRMHND_DIP_SLV2	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x14	0x14	T_FRMHND_DIP_SLV3	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x15	0x55	T_FRMHND_DIP_SLV4	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x16	0xD6	T_FRMHND_DIP_SLV5	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x17	0x97	T_FRMHND_DIP_SLV6	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x18	0xD8	T_FRMHND_DIP_SLV7	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x19	0x99	T_FRMHND_DIP_SLV8	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x1A	0x1A	T_FRMHND_DIP_SLV9	Slave の Dip Switch2～8 の値	Slave ---> Master	8	2
0x31	0xB1	T_FRMHND_DIP_MST1	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x32	0x32	T_FRMHND_DIP_MST2	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x33	0x73	T_FRMHND_DIP_MST3	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x34	0xB4	T_FRMHND_DIP_MST4	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x35	0xF5	T_FRMHND_DIP_MST5	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x36	0x76	T_FRMHND_DIP_MST6	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x37	0x37	T_FRMHND_DIP_MST7	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x38	0x78	T_FRMHND_DIP_MST8	Master の Dip Switch2～8 の値	Master ---> Slave	14	8
0x39	0x39	T_FRMHND_DIP_MST9	Master の Dip Switch2～8 の値	Master ---> Slave	14	8

4. メッセージの説明

表 2 メッセージの説明

メッセージ名	信号名	Event Trans	ビット位置 MSB - LSB	ビット長	適用	値
T_FRMHND_DIP_SLV1	T_SIGHND_DS2_8	Yes	6 ~ 0 (14 ~ 8)	7	Slave の Dip Switch 2 ~ 8 の値	00 ~ 7F
T_FRMHND_DIP_SLV2	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV3	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV4	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV5	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV6	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV7	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV8	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_SLV9	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST1	T_SIGHND_DM2	Yes	0 (8)	1	Master の Dip Switch 2 の値	0 / 1
	T_SIGHND_DM3	Yes	0 (56)	1	Master の Dip Switch 3 の値	0 / 1
	T_SIGHND_DM4	Yes	0 (0)	1	Master の Dip Switch 4 の値	0 / 1
	T_SIGHND_DM5_8	Yes	3 ~ 0 (35 ~ 32)	4	Master の Dip Switch 5 ~ 8 の値	0000 ~ 1111
T_FRMHND_DIP_MST2	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST3	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST4	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST5	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST6	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST7	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST8	↑	↑	↑	↑	↑	↑
T_FRMHND_DIP_MST9	↑	↑	↑	↑	↑	↑

5. 機能概要

5.1. S810-CLG2 の場合

動作の確認のために、ボード上の LED と DIP Switch に以下の機能を与えている。

1) ハート・ビート機能

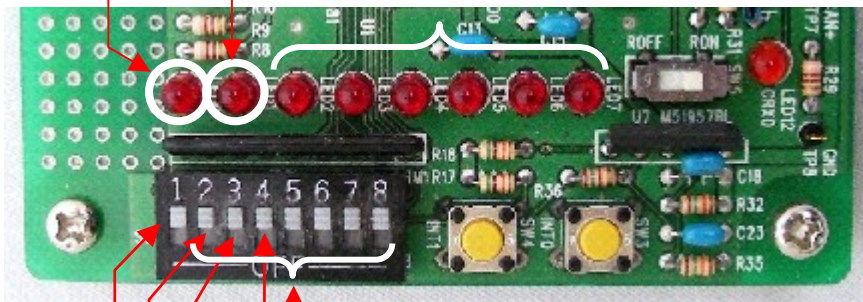
動作中、LED0 を 1 秒間隔で点滅させる。

2) ステータス表示

LED1 でステータスを表示する

- 点灯: Active
- 消灯: Sleep

3) 通信相手の DIP Switch 3 ~ 8 の設定状態を、LED2 ~ LED7 で表示する



4) スリープ可否の設定

- DIP Switch 1 = ON : 不可
- DIP Switch 1 = OFF : 可

5) ウェイクアップ要因の発生(ウェイクアップ信号発生要求)

DIP Switch 2 を、ON ⇒ OFF ⇒ ON と変化させることにより、1 回要求を発生する。

6) スケジュール・テーブルの切り替え(マスタのみ)

DIP Switch 3 を、ON ⇒ OFF ⇒ ON と変化させるたびに、スケジュール・テーブルを切り替える。

8) 強制スリープ(マスタのみ)

DIP Switch 4 を、ON ⇒ OFF ⇒ ON と変化させるたびに、強制スリープを要求する。ただし、DIP Switch 1 は OFF でなければならない。

9) 通信相手ノードの LED2 ~ LED7 に表示させるデータの設定

DIP Switch 2 ~ DIP Switch 8 の設定データが、相手ノードの LED1 ~ LED7 に表示される。

5.2. S810-CLG3, S810-CLG3-85 の場合

動作の確認のために、ボード上の LED と DIP Switch に以下の機能を与えている。

1) ハート・ビート機能

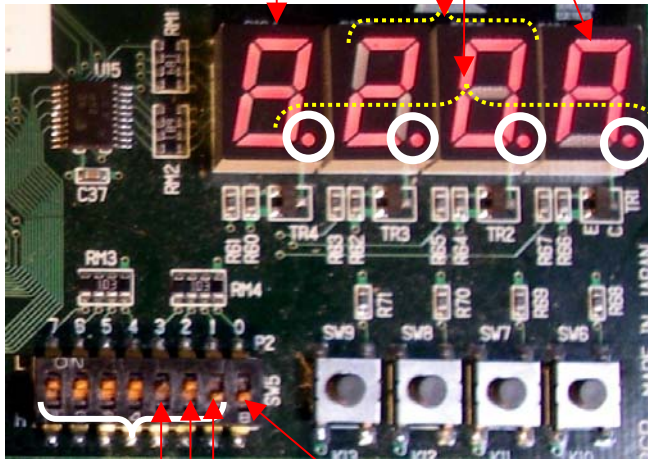
動作中、**ドットセグメント**を 1 秒間隔で点滅させる。また **LED4** を 1 秒毎にカウントアップする。

2) ステータス表示

LED1 でステータスを表示する

- **A** : Active
- **5(S)** : Sleep

3) 通信相手の DIP Switch 2 ~ 7 の設定状態を、**LED2 ~ LED7** で表示する



4) スリープ可否の設定

- **DIP Switch 0 = ON** : 不可
- **DIP Switch 0 = OFF** : 可

5) ウェイクアップ要因の発生(ウェイクアップ信号発生要求)

DIP Switch 1 を、ON ⇒ OFF ⇒ ON と変化させることにより、1 回要求を発生する。

6) スケジュール・テーブルの切り替え(マスタのみ)

DIP Switch 2 を、ON ⇒ OFF ⇒ ON と変化させるたびに、スケジュール・テーブルを切り替える。

8) 強制スリープ(マスタのみ)

DIP Switch 3 を、ON ⇒ OFF ⇒ ON と変化させるたびに、強制スリープを要求する。ただし、DIP Switch 0 は OFF でなければならない。

9) 通信相手ノードの LED2 ~ LED7 に表示させるデータの設定

DIP Switch 1 ~ DIP Switch 7 の設定データが、相手ノードの LED1 ~ LED7 に表示される。

6. スケジュール・テーブル

6.1. スケジュール・テーブルの種類

スケジュール・テーブル スケジュール名称	デフォルト・ テーブル 1: デフォルト 0: その他	スケジュール・タイプ 0: オペレーション 1: ダイアグノスティクス 2: コンフィギュレーション 3: 検査	スケジュール・テーブルを 切り替える方法／タイミング
Schedule1	1	0	Default schedule
Schedule2	0	0	DIP Switch 3 を、H ⇒ L ⇒ H と変化させる。もう一度 DIP Switch 3 を、H ⇒ L ⇒ H と変化させるとまた Schedule 1 に戻る。(トグルで動作する)

6.2. メッセージ・スケジュール

スケジュール名称	メッセージ名	メッセージ・スロット(ms)
Schedule1	T_FRMHND_DIP_SLV1	12
	T_FRMHND_DIP_MST9	26
	T_FRMHND_DIP_SLV2	12
	T_FRMHND_DIP_MST8	26
	T_FRMHND_DIP_SLV3	12
	T_FRMHND_DIP_MST7	26
	T_FRMHND_DIP_SLV4	12
	T_FRMHND_DIP_MST6	26
	T_FRMHND_DIP_SLV5	12
	T_FRMHND_DIP_MST5	26
	T_FRMHND_DIP_SLV6	12
	T_FRMHND_DIP_MST4	26
	T_FRMHND_DIP_SLV7	12
	T_FRMHND_DIP_MST3	26
	T_FRMHND_DIP_SLV8	12
	T_FRMHND_DIP_MST2	26
	T_FRMHND_DIP_SLV9	12
	T_FRMHND_DIP_MST1	26
Schedule2	T_FRMHND_DIP_SLV1	10
	T_FRMHND_DIP_MST1	16
	T_FRMHND_DIP_SLV9	10
	T_FRMHND_DIP_MST9	16





1. はじめに

本書は、LIN 仕様の一部である。

1.1 本書の目的

本書には、LIN ツールが認識する LIN 記述言語の構文と意味について記載する。

2. 改訂履歴

Rev	著者	日付	改訂内容
1.0	VCT-IHt	1999 年 7 月 2 日	本仕様初公開
1.1	VCT-IHt	1999 年 12 月 14 日	構文の改善、誤り訂正(1箇所)
1.11	VCT-IHt	2000 年 2 月 11 日	第 8 章削除
1.2	VCT-IHt	2000 年 8 月 28 日	イベント・トリガード・フレームの定義を追加
1.2	VCT-IHt	2000 年 11 月 13 日	オプション的なフレーム長の定義を追加
draft2	VCT-IHt		診断フレーム処理を追加
			イベント・トリガード・フレームの定義を変更
1.3	VCT-IHt	2002 年 12 月 8 日	例を訂正

2.1 Rev1.0 と Rev1.1 との相違点

- ・ 文法上、LIN プロトコルおよび言語のバージョン番号を `char_string` 型となるように変更
- ・ `group_offset` に関する記述の中の誤りを訂正
- ・ 符号化タイプの文法を改善
- ・ 変更に従って例を更新

2.2 Rev1.1 と Rev1.11 との相違点

- ・ エミュレーション制御ファイルの記述を削除
- ・ スケジュール・テーブル例中の誤りを訂正

2.3 Rev1.11 と Rev1.2 との相違点

- ・ イベント・トリガード・フレームの定義をオプションとして追加
- ・ オプション的なフレーム長の定義を追加
- ・ イベント・トリガード・フレームの定義を変更
- ・ 診断アドレスの定義を追加
- ・ 診断フレームの定義を追加

2.4 Rev1.2 と Rev1.3 との相違点

- ・ 例を訂正

3. 参考文献

参照符号	文献	Doc.nr	Rev／日付
[1]	LIN プロトコル仕様		1.3
[2]	LIN API 推奨実施例		1.3



4. 用語

4.1 略語

LIN	Local Interconnect Network (ローカル・インターコネクト・ネットワーク)
TBD	To Be Defined (未定)
Tool	LIN analyzer/emulator (LIN アナライザ／エミュレータ)

5. 適用範囲

本書に記述する言語は、「LIN 記述ファイル(LDF)」を作成するために使用される。LIN 記述ファイルは、LIN ネットワークを完全に記述するファイルであり、ネットワークの監視に必要なすべての情報も含んでいる。この情報を用いれば、(Tool がサポートする場合)Tool のユーザ・インターフェース(例えば、エミュレーション・ノードやスケジュール・テーブルを選択して)を介して制御される、制限されたエミュレーションを設定することができる。

LIN Tool のユーザ・インターフェースに関しては、構文および意味のいずれも規定されていない場合は、Tool ベンダ独自での実施が許されている。

さらに、LIN 記述ファイルは、LIN ネットワークの一部となる電子制御ユニット用のソフトウェアを記述する際の、コンポーネントの 1 つとして利用することもできる。従来、アプリケーション・プログラマー・インターフェース(API :Application Programmer's Interface)は、異なるアプリケーション・プログラム内部から LIN ネットワークにアクセスするための一定の方法を確保するために、推奨実施例として定義されてきた(参考文献[2]参照)。しかし、アプリケーション・プログラムの機能的な動作については、LIN 記述ファイルによる指定がなされていない。

6. 構文の概要

構文は、modified BNF(Bachus-Naur 形式)を用いて記述される。BNF を下表にまとめる。

シンボル	意味
::=	::=の左側の名称は、右側の構文を用いて表現される。
<>	後に規定するオブジェクトをマークするのに用いる。
	垂直バーは、選択を意味する。垂直バーの左右いずれかが選択される。
Bold	太字体 のテキストは、予約ワードまたは強制的な区切りのいずれかであるため、予約済みである。
[]	角括弧内のテキストは、1 回または複数回出現する。
()	いくつかのオプション的な節をグループ化するのに用いる。
char_string	引用符 “ ” で囲われた任意の文字列
identifier	一般的に名称オブジェクトに使用される識別子。識別子は、標準的な C 言語の変数宣言規則に従うものとする。
integer	整数。10 進法(第 1 桁目は 1~9 を表す)または 16 進法(数値の前に 0x を付す)で表すことができる。
real_or_integer	実数または整数。実数は常に固定小数点の 10 進法で表される。

この構文を使用するファイル中では、どこにコメントを加えてもかまわない。コメント構文は、C++の規則と同じであって、//から行末までと、デリミタ/*および*/で区切られた部分は、すべて無視される。

7. LIN 記述ファイルの定義

<LIN_description_file> ::=

LIN_description_file ;

<LIN_protocol_version_def>

<LIN_language_version_def>

<LIN_speed_def>

<Node_def>

(<Diag_addr_def>)

<Signal_def>

(<Diag_signal_def>)

<Frame_def>

(<Event_triggered_frame_def>)

(<Diag_frame_def>)

<Schedule_table_def>

(<Signal_groups_def>)

(<Signal_encoding_type_def>)

(<Signal_representation_def>)

7.1 LIN プロトコル・バージョン番号の定義

<LIN_protocol_version_def> ::=

LIN_protocol_version = char_string ;

「0.01」～「99.99」の範囲の値をとるものとする。

7.2 LIN 言語バージョン番号の定義

<LIN_language_version_def> ::=

LIN_language_version = char_string ;

「0.01」～「99.99」の範囲の値をとるものとする。

7.3 LIN 通信速度の定義

<LIN_speed_def> ::=

LIN_speed = real_or_integer kbps ;

1.00～20.00k ビット／秒の範囲の値をとるものとする。

7.4 ノードの定義

<Node_def> ::=

Nodes {

Master:<node_name>,<time_base> **ms** ,<jitter> **ms** ;

Slaves:<node_name>([,<node_name>]) ;

}

<node_name> ::= identifier

node_name で表されるすべての識別子は、サブクラス Nodes において一意であるものとする。

予約ワード Master の後ろの node_name で表される識別子は、マスター・ノードを規定する。

<time_base> ::= real_or_integer

time_base 値は、最大許容フレーム伝送時間を生成するために、マスター・ノードで用いた時間基準を規定する。
なお、time_base は、ミリ秒で規定するものとする。

<jitter> ::= real_or_integer

jitter 値は、time_base の開始点からフレーム・ヘッダ送信開始点(ブレイク信号の立下りエッジ)までの最大遅延と最小遅延の差を規定する。なお、jitter は、ミリ秒で規定するものとする。(time_base および jitter の使用方法に関するより詳細な情報については、サブクラス Schedule_tables の定義を参照のこと。)

7.5 ノード診断アドレスの定義

<Diag_addr_def> ::=

Diagnostic_addresses {

[<node_name>:<diag_addr>;]

}

<node_name> ::= identifier

node_name で表されるすべての識別子は、サブクラス Nodes において規定される node_name 識別子の 1 つと等しいものとする。

<diag_addr> ::= integer

diag_addr は、1~255 の範囲内で、識別されたノードの診断アドレスを規定する。(診断アドレス 0 は、将来適用されるアドレスとして予約済みである。)

7.6 シグナルの定義

<Signal_def> ::=

Signals {

[< signal_name >:< signal_size>,<init_value>,<published_by>
[,<subscribed_by>];]

}

<signal_name> ::= identifier

signal_name で表されるすべての識別子は、サブクラス Signals において一意であるものとする。

<signal_size> ::= integer

signal_size は、1～16 ビットの範囲で、シグナルのサイズを規定する。

<init_value> ::= integer

init_value は、シグナルを含むフレームを受信するまで、すべての受信ノードで使用されるシグナル値を規定する。アプリケーション・プログラムがシグナルを更新するまでは、同じ初期シグナル値が(スケジュール・テーブルに一致する)送信ノードから送信されるものとする。

<published_by> ::= identifier

<subscribed_by> ::= identifier

published_by および subscribed_by で表される識別子は、サブクラス Nodes において規定される node_name 識別子の 1 つと等しいものとする。

7.7 フレームの定義

<Frame_def> ::=

Frames {

 [<frame_name>:<frame_id>,<published_by>(<frame_size>) {

 [<signal_name>,<signal_offset>;]

 }]

}

 <frame_name> ::= identifier

frame_name で表されるすべての識別子は、サブクラス Frames において一意であるものとする。

 <frame_id> ::= integer

frame_id は、0～63 の範囲で、フレーム ID 番号を規定する。frame_id は、サブクラス Frames におけるすべてのフレームについて一意であるものとする。

フレーム・サイズは、LIN プロトコル仕様に従って、frame_id を継承する。

 <published_by> ::= identifier

published_by で表される識別子は、サブクラス Nodes において規定される node_name 識別子の 1 つと等しいものとする。

 <frame_size> ::= integer

frame_size は、オプション的な識別子であって、1～8 バイトの範囲でフレームのサイズを規定する。

frame_size の規定がない場合、フレームのサイズは、LIN プロトコル仕様の規定する通り、frame_id から派生するものとする。

 <signal_name> ::= identifier

signal_name で表される識別子は、サブクラス Signals において規定される signal_name 識別子の 1 つと等しいものとする。

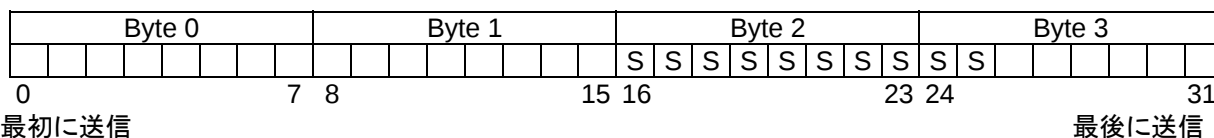
1 つのフレーム定義内のシグナルはすべて、そのフレームに対して published_by 識別子で規定されたのと同じノードから送信される。

 <signal_offset> ::= integer

signal_offset 値は、フレーム内のシグナルの最下位ビット位置を規定する。この値は、0～(8*frame_size-1)の範囲の値をとるものとする。なお、シグナルの最下位ビットは、最初に送信されるビットである。

例:

シグナル「S」(サイズ=10 ビット)は、オフセット 16 ビットの 4 バイト長フレームに設定する。(「S」の LSB はオフセット 16 ビットの位置、また、MSB はオフセット 25 ビットの位置となる。)



最初に送信

最後送信

シグナル設定規則:

- ・ サイズが 8 ビット以上のすべてのシグナルについて、(最下位ビットの)バイト位置合わせを強制的に行う。
- ・ 8 ビットより短いシグナルは、1 つのバイトにのみ含まれるものとする(これにより、「短い」シグナルがバイト境界にまたがることは不可能となる)。

7.8 イベント・トリガード・フレームの定義

`<Event_triggered_frame_def> ::=`

Event_triggered_frames {

`[<event_trig_frm_name>:<frame_id>[,<frame_name>];]`

}

`<event_trig_frm_name> ::= identifier`

event_trig_frm_name で表されるすべての識別子は、サブクラス Event_triggered_frames において一意であり、サブクラス Frames において定義されるいかなる識別子とも異なるものとする。

`<frame_id> ::= integer`

frame_id は、0~63 の範囲で、フレーム ID 番号を規定する。frame_id は、サブクラス Frames および Event_triggered_frames のすべてのフレームについて一意であるものとする。

`<frame_name> ::= identifier`

frame_name で表されるすべての識別子は、サブクラス Frames において規定される識別子の 1 つと等しいものとする。

Frame_name 識別子リストによって規定されるすべてのフレームは、ネットワークの異なるノードから発行されるものとする。

注: サブクラス Frames の frame_name によって特定されるフレームは、イベント・トリガード・フレームにマッピングする際には、レイアウト上の制限が追加される。イベント・トリガード・フレームは、予約済みの最初のバイトを有しており、このバイトには、サブクラス Frames において規定されるフレームの完全なフレーム ID (ID およびパリティ・ビット) が含まれるものとする。

スレーブ・タスクは、イベント・トリガード・フレームの直前の送信から、フレーム内容が更新されている場合に限って、イベント・トリガード・フレーム ID (サブクラス Event_triggered_frames において規定される) に応答するものとする。

注: イベント・トリガード・フレームとサブクラス Frames において規定されるそれぞれのフレームのデータ内容は、両フレームがネットワーク上で観測される際には、等しいものとする。同時に2つ以上のノードが、イベント・トリガード・フレームに応答している場合には、バス衝突が発生するものと考えられる。この場合、マスター ECU が、サブクラス Frames において規定されるフレーム ID によりマッピングされたすべてのフレームに対して、1つのスレーブ ECU を割り当てなければならない。

7.9 診断フレームの定義

<Diag_frame_def> ::=

```
Diagnostic_frames {
  MasterReq : 60 {
    MasterReqB0,0;
    MasterReqB1,8;
    MasterReqB2,16;
    MasterReqB3,24;
    MasterReqB4,32;
    MasterReqB5,40;
    MasterReqB6,48;
    MasterReqB7,56;
  }
  SlaveResp : 61 {
    SlaveRespB0,0;
    SlaveRespB1,8;
    SlaveRespB2,16;
    SlaveRespB3,24;
    SlaveRespB4,32;
    SlaveRespB5,40;
    SlaveRespB6,48;
    SlaveRespB7,56;
  }
}
```

MasterReq および SlaveResp という予約済みのフレーム名によって、診断フレームが識別される。

MasterReq フレームは、LIN プロトコル仕様において規定する固定フレーム ID(60)および固定サイズ(8 バイト)を有している。このフレームは、マスター・ノードからのみ送信可能である。

SlaveResp フレームは、LIN プロトコル仕様において規定する固定フレーム ID(61)および固定サイズ(8 バイト)を有している。このフレームは、直前の MasterReq フレームによって選択されたスレーブ・ノードからのみ送信可能である。

スレーブ・ノードの選択はサブクラス Diagnostic_addressess において規定するスレーブの診断アドレスに基づく。

MasterReqB0~MasterReqB7 の予約済みシグナル名により、MasterReq フレームにおいて 8 ビット長整数として使用されるシグナルを特定する。

SlaveRespB0~SlaveRespB7 の予約済みシグナル名により、SlaveResp フレームにおいて 8 ビット長整数として使用されるシグナルを特定する。

MasterReqB0 および SlaveRespB0 は、LIN プロトコル仕様第 3.2 項において規定するように使用するものとする。

(MasterReq および SlaveResp フレームにおける)予め定義された診断シグナルの設定記述は、正規のフレーム記述構文に従う。(signal_name、signal_offset)

7.10 スケジュール・テーブルの定義

<Schedule_table_def> ::=

```
Schedule_tables {  
    [<schedule_table_name> {  
        [<frame_name> delay <frame_time> ms ;]  
    }  
}
```

<schedule_table_name> ::= identifier

schedule_table_name で表されるすべての識別子は、サブクラス Schedule tables において一意であるものとする。

<frame_name> ::= identifier

frame_name で表される識別子は、サブクラス Frames において規定される frame_name 識別子の 1 つと等しいものとする。

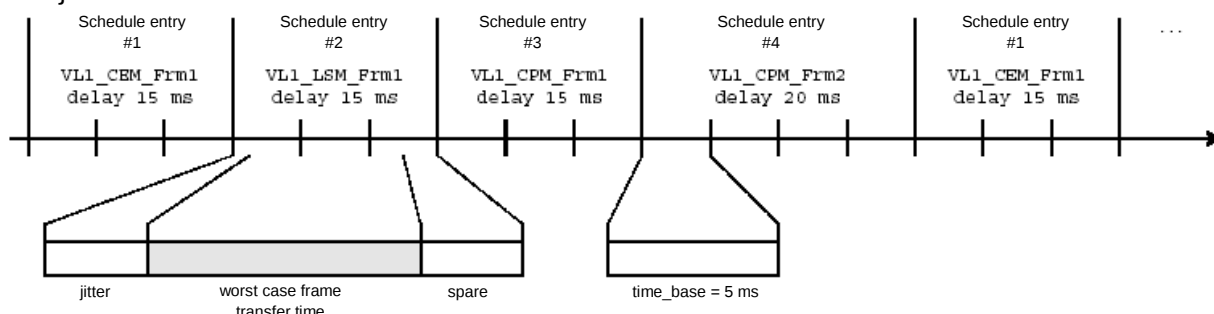
<frame_time> ::= real_or_integer

frame_time は、2 つの隣接するフレーム間の時間間隔を規定する。この時間は、最大許容フレーム伝送時間よりも長く、マスター・ノードの time_base 値の正確な倍数であるものとする。なお、frame_time 値は、ミリ秒で規定するものとする。

スケジュール・テーブルの選択は、マスター・アプリケーション・プログラムによって制御されるものとする。テーブル間の切り替えは、フレーム時間(現在の送信フレーム)経過直後に行わなければならない。

例:

```
Schedule_tables {
  VL1_ST1 {
    VL1_CEM_Frm1 delay 15 ms;
    VL1_LSM_Frm1 delay 15 ms;
    VL1_CPM_Frm1 delay 15 ms;
    VL1_CPM_Frm2 delay 20 ms;
  }
}
```



スケジュール・エントリーごとに規定される delay は、jitter およびワーストケースのフレーム伝送時間よりも長いものとする(参考文献[1]参照)。

注:異なるスケジュール・テーブル間での使用方法ならびに切り替え方法は、アプリケーション・プログラムによって異なる問題ではあるが、適切なメカニズムについては参考文献[2]に記述がある。

7.11 シグナルグループの定義

シグナルグループのサブクラスは、LIN ファイルのオプションである。

```
<Signal_groups_def> ::=
Signal_groups {
  [<signal_group_name>:<group_size> {
    [<signal_name>,<group_offset> ;]
  }]
}
```

<signal_group_name> ::= identifier

signal_group_name で表される識別子は、サブクラス Signal_group において一意であり、サブクラス Signals において規定されるいかなる signal_name 識別子とも異なるものとする。

<group_size> ::= integer

group_size は、1~(8*frame_size)ビットの範囲で、シグナルのサイズを規定する。

<signal_name> ::= identifier

signal_name で表される識別子は、サブクラス Signals において規定される signal_name 識別子の 1 つと等しいものとする。

<group_offset> ::= integer

group_offset 値は、グループ内のシグナルの最下位ビット位置を規定するものであり、0～(group_size-1)の範囲の値をとる。また、最下位ビットは最初に送信されるビットである。グループ内の未使用ビット位置には、ゼロを設定する。

1 つのシグナルグループは、常に、1 つのシグナルフレームに属するシグナルから構成される。グループの定義により、Tool において 17 ビット以上の 1 つのシグナルを表すことが可能となる(このときの Tool は、依然、予め定義された特定の信号符号化タイプを使用している)。

7.12 信号符号化タイプの定義

信号符号化タイプのサブクラスは、LIN ファイルのオプションである。

```
<signal_encoding_type_def> ::=
  Signal_encoding_types {
    [<signal_encoding_type_name> {
      [<logical_value> |
       <physical_range> |
       <bcd_range> |
       <ascii_range>]
    }
  }
```

```
<signal_encoding_type_name> ::= identifier
```

signal_encoding_type_name で表されるすべての識別子は、サブクラス Signal encoding types において一意である。

```
<logical_value> ::= logical_value, <signal_value>(<text_info>) ;
<physical_range> ::= physical_value, <min_value>, <max_value>, <scale>,
<offset>(<text_info>) ;
<bcd_value> ::= bcd_value ;
<ascii_value> ::= ascii_value ;
<signal_value> ::= integer
<min_value> ::= integer
<max_value> ::= integer
<scale> ::= real_or_integer
<offset> ::= real_or_integer
<text_info> ::= char_string
```

signal_value、min_value および max_value は、0～65535 の範囲の値をとるものとする。また、max_value は、min_value 以上の値をとるものとする。信号符号化タイプ情報は、モニタリング中に、生データを論理値やスケール化物理値および/または予め定義された文字列に置き換えるために、Tool で使用可能である。生データが、最小値および最大値で規定される範囲内にある場合、物理値は以下のように算出される。

物理値=スケール*生データ+オフセット

例:

```
Signal_encoding_types {
    1BitDig {
        logical_value,0,"off";
        logical_value,1,"on";
    }
    Temp {
        physical_value,0,250,0.5,-40,"degree";
        physical_value,251,253,1,0,"undefined";
        logical_value,254,"out of range";
        logical_value,255,"error";
    }
}
```

7.13 シグナル表示の定義

シグナル表示のサブクラスは LIN ファイルのオプションである。

```
<Signal_representation_def> ::=
Signal_representation {
    [<signal_encoding_type_name>:<signal_or_group_name>
    ([,<signal_or_group_name>]);]
}
```

<signal_encoding_type_name> ::= identifier

signal_encoding_type_name で表される識別子は、サブクラス Signal_encoding_types において規定される signal_encoding_type_name 識別子の 1 つと等しいものとする。

<signal_or_group_name> ::= <signal_name> | <signal_group_name>

<signal_name> ::= identifier

<signal_group_name> ::= identifier

signal_name で表される識別子は、サブクラス Signals において規定される signal_name 識別子の 1 つと等しいものとする。

signal_group_name で表される識別子は、サブクラス Signal_groups において規定される signal_group_name 識別子の 1 つと等しいものとする。

8. 例

8.1 LIN 記述ファイル

```
// This is a LIN description example file
// Issued by Istvan Horvath
LIN_description_file ;
LIN_protocol_version = "1.3";
LIN_language_version = "1.3";
LIN_speed = 19.2 kbps;

Nodes {
    Master:CEM,5 ms, 0.1 ms;
    Slaves:LSM,CPM;
}

Signals {
    RearFogLampInd:1,0,CEM,LSM;
    PositionLampInd:1,0,CEM,LSM;
    FrontFogLampInd:1,0,CEM,LSM;
    IgnitionKeyPos:3,0,CEM,LSM,CPM;
    LSMFuncIllum:4,0,CEM,LSM;
    LSMSymbolIllum:4,0,CEM,LSM;
    StartHeater:3,0,CEM,CPM;
    CPMReqB0:8,0,CEM,CPM;
    CPMReqB1:8,0,CEM,CPM;
    CPMReqB2:8,0,CEM,CPM;
    CPMReqB3:8,0,CEM,CPM;
    CPMReqB4:8,0,CEM,CPM;
    CPMReqB5:8,0,CEM,CPM;
    CPMReqB6:8,0,CEM,CPM;
    CPMReqB7:8,0,CEM,CPM;
    ReostatPos:4,0,LSM,CEM;
    HeadLampBeamLev:4,0,LSM,CEM;
    FrontFogLampSw:1,0,LSM,CEM;
    RearFogLampSw:1,0,LSM,CEM;
    MLSSoff:1,0,LSM,CEM;
    MLSHeadLight:1,0,LSM,CEM;
    MLSPosLight:1,0,LSM,CEM;
    HBLShortHigh:1,0,LSM,CEM;
    HBLShortLow:1,0,LSM,CEM;
    ReoShortHigh:1,0,LSM,CEM;
    ReoShortLow:1,0,LSM,CEM;
    LSMHWPartNoB0:8,0,LSM,CEM;
    LSMHWPartNoB1:8,0,LSM,CEM;
    LSMHWPartNoB2:8,0,LSM,CEM;
    LSMHWPartNoB3:8,0,LSM,CEM;
    LSMSWPartNo:8,0,LSM,CEM;
    CPMOutputs:10,0,CPM,CEM;
    HeaterStatus:4,0,CPM,CEM;
    CPMGlowPlug:7,0,CPM,CEM;
    CPMFanPWM:8,0,CPM,CEM;
    WaterTempLow:8,0,CPM,CEM;
```

```

WaterTempHigh:8,0,CPM,CEM;
CPMFuelPump:7,0,CPM,CEM;
CPMRunTime:13,0,CPM,CEM;
FanIdealSpeed:8,0,CPM,CEM;
FanMeasSpeed:8,0,CPM,CEM;
CPMRespB0:1,0,CPM,CEM;
CPMRespB1:1,0,CPM,CEM;
CPMRespB2:1,0,CPM,CEM;
CPMRespB3:1,0,CPM,CEM;
CPMRespB4:1,0,CPM,CEM;
CPMRespB5:1,0,CPM,CEM;
CPMRespB6:1,0,CPM,CEM;
CPMRespB7:1,0,CPM,CEM;
}
Frames {
  VL1_CEM_Frm1:32,CEM,3 { //The length of this frame is redefined to 3
    RearFogLampInd,0;
    PositionLampInd,1;
    FrontFogLampInd,2;
    IgnitionKeyPos,3;
    LSMFuncIllum,8;
    LSMSymbolIllum,12;
    StartHeater,16;
  }
  VL1_CEM_Frm2:48,CEM {
    CPMReqB0,0;
    CPMReqB1,8;
    CPMReqB2,16;
    CPMReqB3,24;
    CPMReqB4,32;
    CPMReqB5,40;
    CPMReqB6,48;
    CPMReqB7,56;
  }
  VL1_LSM_Frm1:33,LSM {
    ReostatPos,0;
    HeadLampBeamLev,4;
    FrontFogLampSw,8;
    RearFogLampSw,9;
    MLSSOff,10;
    MLSHeadLight,11;
    MLSPosLight,12;
    HBLSortHigh,16;
    HBLShortLow,17;
    ReoShortHigh,18;
    ReoShortLow,19;
  }
  VL1_LSM_Frm2:49,LSM,5 { //The length of this frame is redefined to 5
    LSMHWPartNoB0,0;
    LSMHWPartNoB1,8;
    LSMHWPartNoB2,16;
    LSMHWPartNoB3,32;
    LSMSWPartNo,40;
  }
}

```

```

VL1_CPM_Frm1:50,CPM {
    CPMOutputs,0;
    HeaterStatus,10;
    CPMGlowPlug,16;
    CPMFanPWM,24;
    WaterTempLow,32;
    WaterTempHigh,40;
    CPMFuelPump,56;
}
VL1_CPM_Frm2:34,CPM {
    CPMRunTime,0;
    FanIdealSpeed,16;
    FanMeasSpeed,24;
}
VL1_CPM_Frm3:51,CPM {
    CPMRespB0,0;
    CPMRespB1,8;
    CPMRespB2,16;
    CPMRespB3,24;
    CPMRespB4,32;
    CPMRespB5,40;
    CPMRespB6,48;
    CPMRespB7,56;
}
}
Schedule_tables {
    VL1_ST1 {
        VL1_CEM_Frm1 delay 15 ms;
        VL1_LSM_Frm1 delay 15 ms;
        VL1_CPM_Frm1 delay 20 ms;
        VL1_CPM_Frm2 delay 20 ms;
    }
    VL1_ST2 {
        VL1_CEM_Frm1 delay 15 ms;
        VL1_CEM_Frm2 delay 20 ms;
        VL1_LSM_Frm1 delay 15 ms;
        VL1_LSM_Frm2 delay 20 ms;
        VL1_CEM_Frm1 delay 15 ms;
        VL1_CPM_Frm1 delay 20 ms;
        VL1_CPM_Frm2 delay 20 ms;
        VL1_LSM_Frm1 delay 15 ms;
        VL1_CPM_Frm3 delay 20 ms;
    }
}
Signal_groups {
    CPMReq:64 {
        CPMReqB0,0;
        CPMReqB1,8;
        CPMReqB2,16;
        CPMReqB3,24;
        CPMReqB4,32;
        CPMReqB5,40;
        CPMReqB6,48;
        CPMReqB7,56;
    }
}

```

```

    }
    CPMResp:64 {
        CPMRespB0,0;
        CPMRespB1,8;
        CPMRespB2,16;
        CPMRespB3,24;
        CPMRespB4,32;
        CPMRespB5,40;
        CPMRespB6,48;
        CPMRespB7,56;
    }
}

Signal_encoding_types {
    1BitDig {
        logical_value,0,"off";
        logical_value,1,"on";
    }
    2BitDig {
        logical_value,0,"off";
        logical_value,1,"on";
        logical_value,2,"error";
        logical_value,3,"void";
    }
    Temp {
        physical_value,0,250,0.5,-40,"degree";
        physical_value,251,253,1,0,"undefined";
        logical_value,254,"out of range";
        logical_value,255,"error";
    }
    Speed {
        physical_value,0,65500,0.008,250,"km/h";
        physical_value,65501,65533,1,0,"undefined";
        logical_value,65534,"error";
        logical_value,65535,"void";
    }
}

Signal_representations {
    1BitDig:RearFogLampInd,PositionLampInd,FrontFogLampInd;
    Temp:WaterTempLow,WaterTempHigh;
    Speed:FanIdealSpeed,FanMeasSpeed;
}

```