
TLIN レファレンス・コード

(LIN コンソシアム仕様に基づくレファレンス・コード)

プログラム・ビルド・ガイド

Version 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
2007/01/31	1.0	日野	初版

目次

1. はじめに	7
2. BUILD 環境	8
2.1. 対象開発ボード	8
2.2. コンパイル環境	9
2.2.1. M3T-NC30WA	9
2.2.2. REAL TIME OS	9
3. フォルダ構造	11
3.1. 出荷 CDROM の内容	11
3.1.1. SOURCECODE フォルダ	11
3.1.2. DOC フォルダ	11
3.1.3. BUILD フォルダ	11
4. ファイル構造	13
4.1. ソース・コード・ファイル	13
4.1.1. APPLICATION TASK	13
4.1.2. FRAME CONTROL TASK	14
4.1.3. ENTITY ACCESSOR	14
4.1.4. OSAL	14
4.1.5. HAL	14
4.1.6. PROTOCOL STACK TASK	15
5. コンフィギュレーション	17
5.1. HAL のコンフィギュレーション	17
5.1.1. TLINCFG.H	17
5.1.2. TLIN_HAL_DEFS.H	20
5.1.3. TLIN_STACK_DEFS.H	22
5.1.4. TLIN_MASTER_RAM.C, TLIN_SLAVE_RAM.C	23
5.2. OS のコンフィギュレーション	25
5.2.1. MY_APPLICATION_NAME.CFG	25
5.2.2. コンフィグレーションファイルの定義項目	27
6. ビルド・オペレーション	33
6.1. ビルドのためのコマンド・ライン処理	33

図表索引

図 1	S810-CLG2 ボード概観	8
図 2	ボードと共に提供されるケーブル類	9
図 3	出荷 CDROM のフォルダ構造	11
図 4	ソース・コード・ファイルの構成 (ヘッダ・ファイルは非表示)	13
図 5	TLinCfg.h	19
図 6	TLIN_HAL_defs.h	21
図 7	tlin_stack_defs.h	23
図 8	tlin_master_ram.c	24
図 9	tlin_slave_ram.c	24
図 10	smp.cfg の記述例	27

1. はじめに

この文書は、『Toppers プロジェクトの一環として供給される、LIN コンソシアム仕様に基づくレファレンス・コード』（以降、『*TLIN Reference Code*』と記す）のソース・コードを実際のアプリケーションと結合させて、実行可能コードを生成する手順（ビルド手順）およびコンフィギュレーションの概要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. Build 環境

2.1. 対象開発ボード

TLIN Reference Code を動作させる環境は、一般的な 16 ビットマイコンもしくは、それ以上のマイコンであれば、メーカーや機種を問わない。ターゲット・マイコンは、最低以下の仕様を満たすものであること。

- プログラム・メモリ領域 64K バイト以上
- データ・メモリ領域(スタックを含む)10K バイト以上
- UART(Transmitter/Receiver のセット): 1
- 標準的な 16 bit timer¹ : 2
- BUS monitor 用外部割込み端子: 1

しかし、ビルド手順は、そのマイコンの開発環境に依存するものであり、手順や手法もそれぞれ異なる。本書では、株式会社サニー技研²製 M16C/6N CAN/LIN Gateway「S810-CLG2³」または「S810-CLG3」、「S810-CLG3-85」を用いて説明を行う。

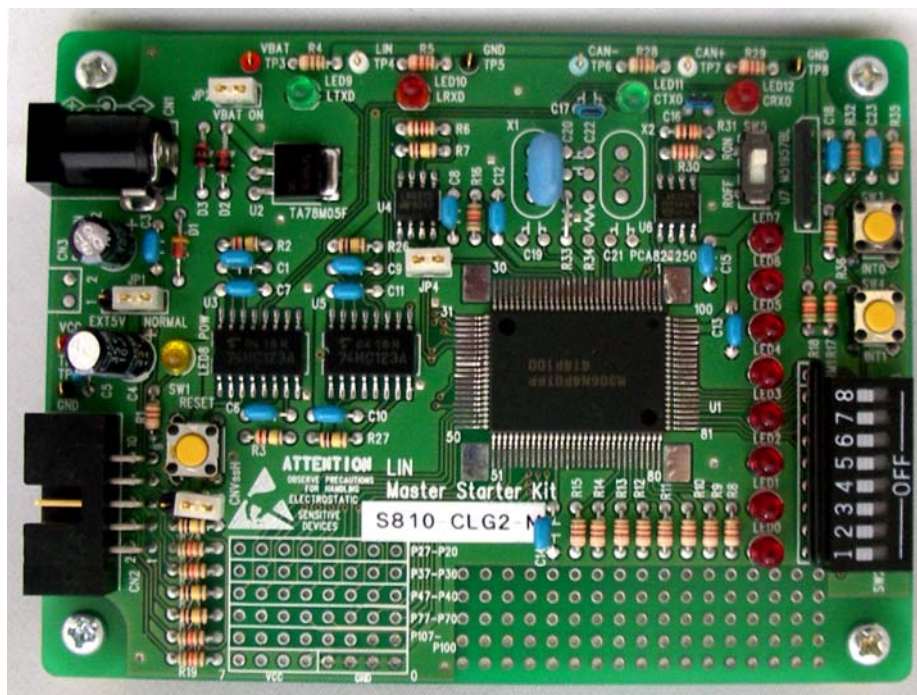


図 1 S810-CLG2 ボード概観

¹ 現在市場に出ているマイコンのタイマには、多くの機能が内蔵されているが、本インプリメンテーションにおいては、レフェレンス・コードにより汎用性を与えるために、プリミティブな機能のみを使用する方針で行った。

² <http://www.sunnygiken.co.jp/>

³ 現在サニー技研社では S810-CLG3 をメインに販売している。本レファレンス・コードは、S810-CLG2、S810-CLG-3、S810-CLG3-85 の全てに対応して設計されている。コンフィギュレーションの項で詳細を説明する。

図 2 に添付されるケーブル類を示す。



図 2 ボードと共に提供されるケーブル類

詳細については、サニー技研・S810-CLG2 に添付される CDROM の説明書を参照すること。

2.2. コンパイル環境

本書での説明は、S810-CLG2 に標準添付されるルネサス・テクノロジー社製の C コンパイラ・M3T-NC30WA を使用する。

2.2.1. M3T-NC30WA

S810-CLG2 に標準添付される C コンパイル環境である。リリース時期によりバージョンが異なる可能性はあるが、本書の評価においては、『M3T-NC30WA V.5.10 Release 1 (エントリー版)』を用いた。

また、IE(統合化開発環境)・TM も含まれるが、本書の説明においては使用しない。統合開発環境は便利であるが、詳細な設定が見えないところで行われるため、敢えて使用を避けた。

2.2.2. Real Time OS

RTOS(Real Time Operating System)は、特に問わない。

実際のところ、TLIN Reference Code は OSEC 環境で供給されるが、RTOS は特に問わない。実際のところ M16C ファミリーでは MR30 OS が良く使われ、知られていているため、MR30 を例に挙げたに過ぎない。

MR30 は、 μ ITRON⁴ 仕様に従って 16 ビットマイクロプロセッサ M16C/60, 30, 20, 10 シリーズ用に開発された、リアルタイム・オペレーティングシステムである。本書で使用的是、Version 3.30 である。

μ ITRON 仕様 V.3.0 では、各システム・コールは、レベル R、レベル S、レベル E、レベル C に分けられている。MR30 は、 μ ITRON 仕様 V.3.0 で規定されたシステム・コールのうち、レベル R、レベル S の全部と、レベル E の一部をインプリメントしている。詳細は、ルネサス・テクノロジー社の M3T-Mr30 ユーザマニュアルを参照のこと。

⁴ 本レファレンス・コードは、使用 OS を μ ITRON 3.0 に特定するものではない。実際のところ、リアルタイム OS のシステム・コールの非常に基本的なもののみを用いて記述しており、OS に依存する処理は OSAL に記述を集めているため、 μ ITRON であれば、Version を問わず動作するはずである。また、他の OS でも動作する。実際 OSEC でも動作を確認している。

3. フォルダ構造

3.1. 出荷 CDROM の内容

LIN レファレンス・コードの出荷 CDROM は、以下のようなフォルダ構造になっている。

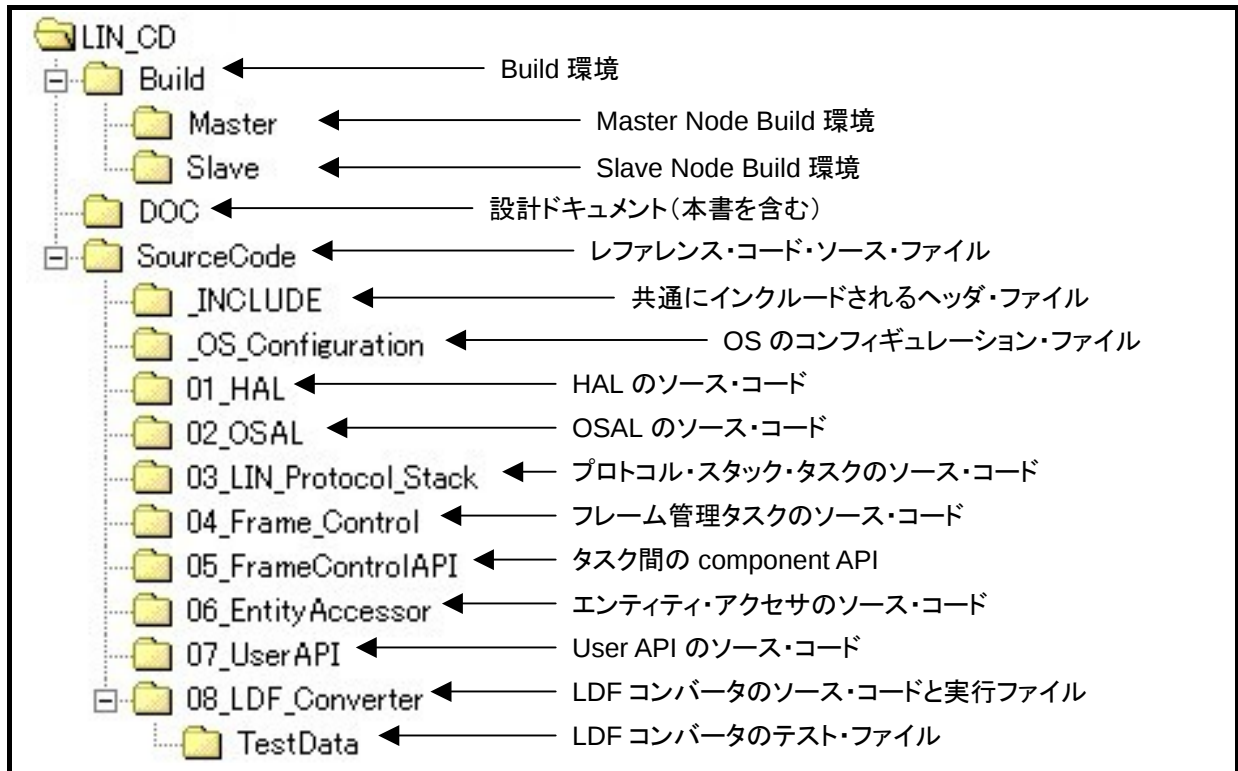


図 3 出荷 CDROM のフォルダ構造

3.1.1. SourceCode フォルダ

全てのソース・コードは、SourceCode フォルダの下に含まれる。SourceCode のフォルダは、モジュールごとにサブフォルダに分けて格納されている。(図 3 参照)

3.1.2. DOC フォルダ

また、ドキュメントは、全て DOC フォルダに格納されている。

3.1.3. Build フォルダ

Build フォルダには、Source フォルダの中でモジュールごとに分納されていたソース・コード(.c)やヘッダ(.h)およびコンフィギュレーション・ファイル(smp.cfg)などを、マスタ・ノード構築に必要なファイル(Master フォルダ)と、スレーブ・ノード構築に必要なファイル(Slave フォルダ)に分けて格納

している。また、M3T-MR30 のツール cfg30 (RTOS 環境コンフィギュレータ) により作成されたアセンブラ用のソース・コード (.a30) およびインクルード・ファイル (.inc) などが含まれる。

さらに、コンフィギュレータが作成したメイク・ファイル (makefile) の他、既にビルドされた結果生成されたアセンブラ・ソース・リスト (.LST)、リローケータブル・オブジェクト・コード (.R30) と、実行形式オブジェクト・ファイル (.x30) が含まれている。

サンプルをそのまま S810-CLG2 にダウンロードして実行するのならば、S810-CLG2 の CDROM に添付されているデバッガ・KD30 でダウンロードして、Go コマンドを実行するだけである。

なお、SourceCode フォルダに含まれるソース・コードと Build フォルダに含まれるソース・コードは、まったく同じものである。ユーザが自分独自のアプリケーションを追加して、新しくビルドする場合は、root の下に適当なフォルダ (例えば MyApp など) を作成し、Build フォルダの内容をそのままコピーしてから、自分のアプリケーション・ソースを作成した後、新しくビルドすることを推奨する。

なお、Build フォルダの Master フォルダと Slave フォルダのソース・コードは、ほとんど共通である。ただし、いくつかのファイルは、マスタ・ノードとスレーブ・ノードのコンフィギュレーションの違いにより異なっていたり、いくつかのファイルは、片方にのみ含まれていたりする。

次節にマスタ、スレーブ、それぞれのファイル構成を説明する。

4. ファイル構造

以下に、ビルド環境で使用されるファイル(即ち Build フォルダに含まれるファイル)の構成を示す。

4.1. ソース・コード・ファイル

図 4 に Build フォルダ中のソース・コード・ファイルの構成を示す。

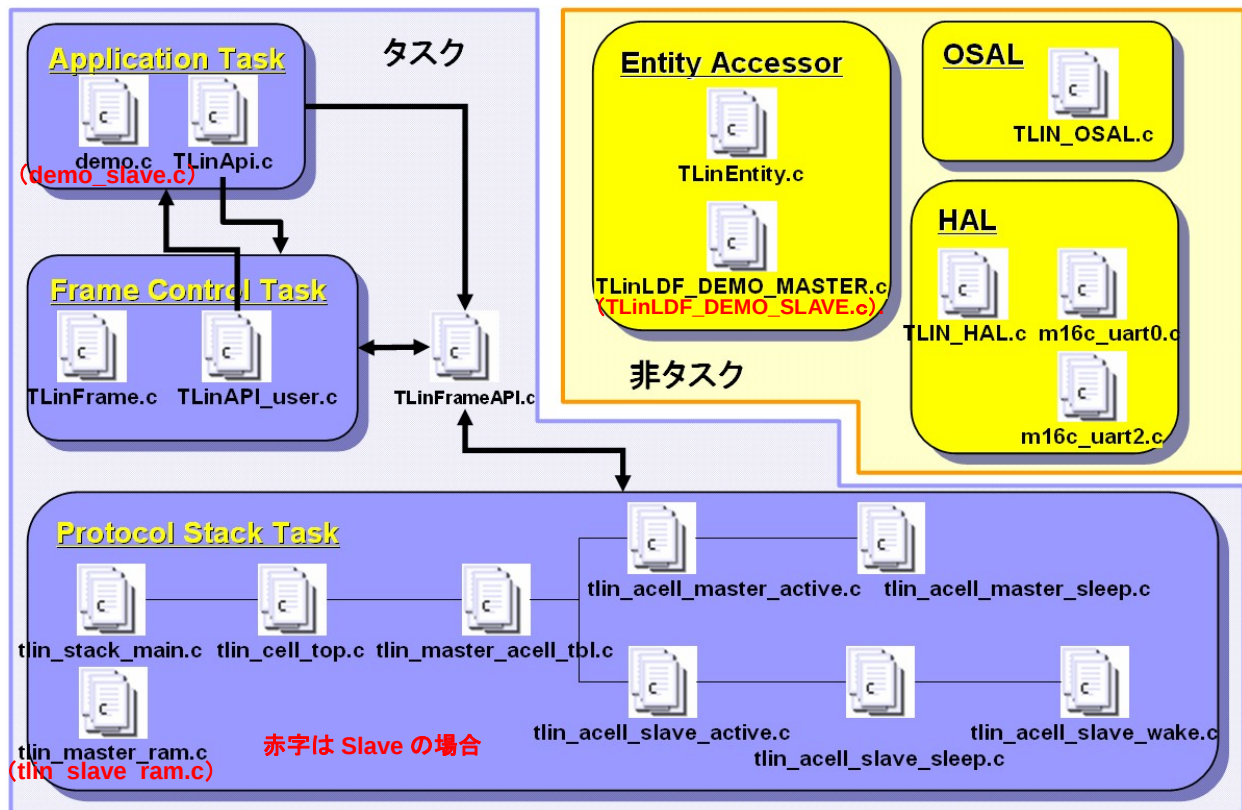


図 4 ソース・コード・ファイルの構成(ヘッダ・ファイルは非表示)

4.1.1. Application Task

レファレンス・コードのサンプル・プログラム(Application Task)のソース・コードは、

マスタ・ノードの場合(Master フォルダ): `demo.c`
スレーブ・ノードの場合(Slave フォルダ): `demo_slave.c`

である。このソースに `main()` 関数が記述されており、レファレンス・コードのプログラムは、この `main()` 関数のプログラムを実行する。Frame Control Task も、Protocol Stack Task も、この Application Task の `main()` 関数の中で起動(start task)しない限り起動されない。いったん下位タスクが起動されると、どのタスクに実行権が移されるかは、OS にゆだねられる。

ユーザ独自のアプリケーション・プログラムは、`demo.c` または `demo_slave.c` を書き直すか、もしくは、これらのファイルを削除して、新しく独自のアプリケーション・ソース・ファイルを作成する。

4.1.2. Frame Control Task

Frame Control Task は、TLinFrame.c ファイルひとつに、ほぼ集約される。このファイルは、Master と Slave でまったく同じものを使用する。

TLinAPI_user.c は、Application Task からユーザ・リクエストのために用意した API (Application Program Interface) 関数を記述したものである。

なお、図 4 では、Frame Control Task の外側に配置されている TLinFrameAPI.c は、Frame Control Task と Protocol Stack Task 間のメッセージ・パッシングによる通信を行う関数を記述している。Frame Control Task と Protocol Stack Task の両方から使用される(関数 call される)。

4.1.3. Entity Accessor

Entity Accessor は、TLinEntity.c と LDF ツールが自動作成するテーブル・ファイルにより構成される。

TLinEntity.c は、Entity Accessor の本体である。

一方テーブル・ファイルは、LDF ツールが作成するファイルであり、ユーザが指定したファイル名に依存して異なる。レファレンス・コードのサンプルでは、Master : TLinLDF_DEMO_Master.c, Slave : TLinLDF_DEMO_Slave.c が含まれている。この他に、図 4 では表示していないが、LDF ツールにより生成される TLinLDF_DEMO.h (Master と Slave で共通) が含まれる。

TLinLDF_DEMO_Master.c と TLinLDF_DEMO_Slave.c は、同じ LDF から生成した場合でも異なるものになる。マスタ・ノードは、全てのメッセージ情報とスケジュール・テーブル情報を持つ必要があるが、スレーブ・ノードは、自身が関係するメッセージ情報のみを持つためである。

テーブル・ファイルは、ユーザが自分で手作業により直接編集しても動作に支障はない。しかし、間違えて論理的に不合理なパラメータ設定を行った場合には動作が保障されない。データ・マスタ・シートもしくはビット・アサイン表を原本として、LIN-Plan などを用いて LDF ファイルを生成し、その LDF ファイルを入力として、LDF ツールにより生成するのが安全である。

4.1.4. OSAL

OSAL (Operating System Abstraction Layer) は、TLIN_OSAL.c に記述されている。レファレンス・コードでデフォルトに使用している M3T-MR30 (μITRON3.0) を使用する場合には、ユーザは関知する必要はない。

4.1.5. HAL

HAL (Hardware Abstraction Layer) は、TLIN_HAL.c, m16c_uart0.c, m16c_uart2.c の 3 つのファイルで構成される。m16c_uart0.c, m16c_uart2.c の 2 つは、マイコン内蔵の UART の入出力制御を、UART#0 と UART#2 のそれぞれについて記述したものである。

タイマ制御および割り込みを用いたイベントの検出と、メッセージ・パッシングは TLIN_HAL.c に記述されている。

出荷時の HAL は、S810-CLG2 を用いた場合に合わせて記述されている。異なる開発ボードを用いる場合（例えば S810-CLG3 など）を用いる場合や、同じボード（S810-CLG2）の場合でも、ポート接続を変えたり、異なる内蔵タイマを使用するなどのハードウェアや接続の変更を行う場合にのみ、これらのファイルの記述を変更する。

ただし、HAL の変更は十分に注意して行わないと、生成された実行コードが、まったく動作しない場合がある。

4.1.6. Protocol Stack Task

Protocol Stack Task は、以下のファイルが、Master フォルダと Slave フォルダに共通に含まれる。

共通部分

- `tlin_stack_main.c` : Protocol Stack のメイン関数。イベント検出と、その振り分け（イベントに対応する action cell を call する）を行う。
- `tlin_cell_top.c`⁵ : API イベント（Frame Control から発行されるリクエスト・イベント）の処理を行う。
- `tlin_acell_tbl.c` : 外部イベント（割り込みに起因するイベント）の action cell が記述されている。

マスタ動作部分

- `tlin_acell_master_active.c` : マスタのアクティブ状態の action cell が記述されている。
- `tlin_acell_master_sleep.c` : マスタのスリープ状態の action cell が記述されている。

スレーブ動作部分

- `tlin_acell_slave_active.c` : スレーブのアクティブ状態の action cell が記述されている。
- `tlin_acell_slave_sleep.c` : スレーブのスリープ状態の action cell が記述されている。
- `tlin_acell_slave_wake.c` : スレーブのウェイク状態の action cell が記述されている。

以上、ここまでが、Master フォルダと Slave フォルダに共通に含まれる。マスタ動作部分とスレーブ動作部分は、実行コードとしては、両方必要ないが、マスタ／スレーブのマルチ・スレッド（ひとつの ECU が、マスタ・ノードとスレーブ・ノードの両方の動作を、ひとつのマイコンで実現する）をサポートするために、両方ともに含まれる構造にした。

マルチ・スレッドが必要ない場合には、ユーザが必要ないグループ（マスタ動作部分またはスレー

⁵ このファイルは、以前のリリース（version 1.5 より前）においては、`tlin_master_celltop.c` としていた。しかし、マスタのみで使用されると勘違いされる恐れがあるために、`master` を取り除き、`tlin_celltop.c` と変更した。

ブ動作部分)を取り除く事は可能であるが、共通部分のコードから、これらの取り除いた部分を call しているため取り除く必要がある。(そうしないと LINK Error が発生する。)

Master フォルダのみに含まれるファイル

- tlin_master_ram.c

Slave フォルダのみに含まれるファイル

- tlin_slave_ram.c

上のそれぞれのファイルは、内蔵タイマと検出イベントの結び付け、インタフェース・ハンドルと HAL の通信チャンネルの結び付けを定義するものである。共通のものを使用しても構わないが、たいていの場合、スレーブ・ノードが使用するリソースの方が、マスター・ノードの使用するリソースに比べて少ないため、リソース割付の最適化のために、別々にすることを推奨する。

5. コンフィギュレーション

5.1. HAL のコンフィギュレーション

5.1.1. TLinCfg.h

以下のコンフィギュレーションを行う。図 5(TLinCfg.h のソース行)に示した番号(赤い丸の囲み番号)にしたがって説明する。

① NO_OS : OS 使用するか否かのコンパイル・スイッチ

OS を使用するインプリメンテーションを行なう場合にこのコンパイル・スイッチを有効にする。通常は、コメントアウトしておく。

② CLG2 , CLG3 , CLG3-85 : ターゲットボードのコンパイル・スイッチ

使用するターゲット・ボードとして S810-CLG2, S810-CLG3, S810-CLG3-85 の 3 種を選ぶコンパイルスイッチ。いずれか、ひとつのみを有効にすること。これにより HAL の MCU 機種依存／ボード依存の部分が変更される。

CLG2 : S810-CLG2(MCU: M16C6N4)

CLG3 : S810-CLG3(MCU M16C6NK)

CLG3-85 : S810-CLG3-85(MCU: M32C85)

③ SYNDEL_SOFT_LOOP : Synch Break Delimiter の長さ設定

マスタ動作の Synch Break Delimiter の時間設定は、ソフトウェア・ループで実現している。このパラメータにより、微調を行う。

④ MAXBYTE_DT_CTL : TMAX_BYTE タイマの動作設定

UART 送信行毎に、TMAX_BYTE タイマによる時間監視を行うか否かを指定する。このマクロは、コンパイル・スイッチとして動作する。このマクロが定義されていると、TMAX_BYTE タイマによる監視が組み込まれる。デフォルトでは、監視を行う。

⑤ SYN_BRK_DT_CTL : UART 送信中の Synch Break の監視

このスイッチが定義されていれば、UART の送受信中の Synch Break 監視を停止する。CPU 負荷が重い場合に、負荷軽減のために設けた。しかし、これを有効にすると、通信の途中に何らかの理由（例えばマスタ ECU がノイズなどのために強制リセットされた等）により、フレームの途中で発生する Synch Break の検出ができなくなるので、通常はコメントアウトしておくことが望ましい。

⑥ MAX_IFC_HNDL_NUM：マルチ・スレッド動作時のハンドルの総数

マルチ・スレッドのインプリメンテーションを行う場合、ハンドルの数を記述する。

```
#ifndef _TLINCFG_H_
#define _TLINCFG_H_

/*=====*/
/* OS 使用可否のスイッチ */
/* このスイッチが定義されていると、No OS 用のインプリとなる。 */
/* ===== */
/* マスタは、No OS に対応していません。マスタの場合は */
/* このスイッチを有効にしないでください。 */
/* ===== */
// #define __NO_OS__
#ifndef __NO_OS__
    #include "id.h"
    #define __OS_ID__
#endif

/*=====*/
/* Board version */
/* 以下のボードのいずれかに対応しています。 */
/* サニー技研製：S810-CLG2 */
/* サニー技研製：S810-CLG3 */
/* サニー技研製：S810-CLG3-85 */
/* 以下のコンパイルスイッチの対応ボードのみを有効にしてください。 */
/* ===== */
#define __CLG2__
// #define __CLG3__
// #define __CLG3_85__

/*=====*/
/* Specification Switch */
/* LIN Consortium 仕様と、T 社の独自仕様の切り替えスイッチ */
/* ※ 本リリース版では、LIN Consortium 仕様のみを掲載しています。 */
/* 以下のコンパイル・スイッチは、CONSORTIUM_SPEC のみでご使用 */
/* ください。(T_SPEC を有効にすると、コードが正しく生成されません) */
/* ===== */
#define CONSORTIUM_SPEC
// #define T_SPEC
```

①

```

/*=====*/
/* Synch Delimiter 計測ループ回数 */
/* ヘッダの Synch Delimiter は、現在ソフトウェア・ループで実現して */
/* います。CPU の動作クロックによっては調整が必要となります。 */
/*=====*/
#define SYNDEL_SOFT_LOOP (300) ③
    /* 現在は、M16C/6N@16MHz 動作で、4 bit@9600bps の設定です。 */

/*=====*/
/* 1 バイト送信の時間監視タイマのアクティベート・コントロール */
/* このスイッチが定義されていれば、1 バイト送出ごとに、タイムアウト */
/* 監視を行いません。ハードウェア(UART or LIN Transceiver)の故障に */
/* よるデッドロック防止になりますが、これを入れなくても、スロット */
/* 時間が経過するとプログラムはエラー(スロット・タイムアウト)を */
/* 検出します。 */
/* 通信時の CPU 負荷軽減のためには、このスイッチは OFF(定義しない) */
/* をお勧めします。 */
/*=====*/
#define MAXBYTE_DT_CTL ④

/*=====*/
/* UART 送信中の Synch Break 監視。 */
/* このスイッチが定義されていれば、UART の送受信中の Synch Break 監視 */
/* を停止する。 */
/*=====*/
// #define SYN_BRK_DT_CTL ⑤

/*=====*/
/* OS リソースの ID 定義 */
/*=====*/

    (途中省略)

/*****
 * Interface Handle
 *****/
#ifdef __CLG2__
#define MAX_IFC_HNDL_NUM (1) /* 取り扱えるインタフェースの数 */
#endif

#ifdef __CLG3__
#define MAX_IFC_HNDL_NUM (2) /* 取り扱えるインタフェースの数 */
#endif

#ifdef __CLG3_85__
#define MAX_IFC_HNDL_NUM (2) /* 取り扱えるインタフェースの数 */
#endif

```

図 5 TLinCfg.h

5.1.2. TLIN_HAL_defs.h

以下のコンフィギュレーションを行う。図 6(TLIN_HAL_defs.h のソース行)に示した番号(赤い丸の囲み番号)にしたがって説明する。

⑦ HAL_CHANNEL_MAX : HAL で管理するチャンネルの数

内容的には、⑥のスレッド数(ハンドル数)と同じであるが、HALにおいてはチャンネル数となる。
⑥MAX_IFC_HNDL_NUM の定義数と一致させなければ動作が保証できない。

⑧ HAL_TIMER_MAX : HAL で使用する仮想タイマの本数

HAL においては、実タイマ・A1 一本を使用して、時間測定を行うタイマを最大 12 本使用できる。
実際に使用する仮想タイマの本数を定義する。

5.1.4 tlin_master_ram.c および 5.1.5 tlin_slave_ram.c で定義されるイベントと仮想タイマの割付の結果を反映したものでなければならない。この数を少なく抑えるほど、タイマ A1 割り込み時の CPU 負荷が小さくなる。

⑨ HAL_SOURCE_CLK : マイコンの源振クロック周波数(MHz)

マイコンに供給される源振クロックの周波数。デフォルトでは、S81-CLG2 の M16C6N4FG の振動子に合わせて 16(MHz)となっている。

⑩ タイマ A1(仮想タイマ実体)の割り込み周期とSynch Break Detect タイマ A2 の割り込み周期とカウント数

- HAL_TIMER_TICK ; 仮想タイマの元となるハードウェア・タイマ A1 の割り込み周期を指定する。整数で指定し、単位は μ sec となる。デフォルトは、500 μ sec になっている。
- HAL_SYMBRK_TIMER_TICK および HAL_PHYERR_TIMER_TICK :
Synch Break Detect および Physical Bus Error 検出に使用されるハードウェア・タイマ A2 の割り込み周期を定義する。タイマ A2 は、Synch Break Detect と、Physical Bus Error 検出では異なる割り込み周期を使用できるインプリメンテーションを用いた。HAL_SYMBRK_TIMER_TICK は Synch Break Detect 検出時の設定、HAL_PHYERR_TIMER_TICK は Physical Bus Error 検出時の設定となる。
- HAL_SYMBRK_TICK, HAL_PHYERR_TICK : それぞれ、Synch Break Detect 検出および Physical Bus Error 検出時の割り込み回数のカウント数を定義する。すなわち

$\text{HAL_SYMBRK_TIMER_TICK} \times \text{HAL_SYMBRK_TICK} = \text{Synch Break Detect 検出時間}$
--

HAL_PHYERR_TIMER_TICK × HAL_PHYERR_TICK = Physical Bus Error 検出時間

となる。(単位は、μ sec)

```
#ifndef __TLIN_HAL_DEFS_H__
#define __TLIN_HAL_DEFS_H__

/**** BUS の信号の論理レベル ****/
#define BUS_RECESSIVE 1
#define BUS_DOMINANT 0

/**** HAL で使用するチャンネル数 ****/
#define HAL_CHANNEL_MAX 1 ⑦

/* 仮想タイマ本数 (スレーブ) */
#define HAL_TIMER_MAX 8 } ⑧
/* 仮想タイマ本数 (マスタ) */
// #define HAL_TIMER_MAX 9

/**** HAL で使用するタイマの割り込みレベル ****/
#define HAL_TIMER_INT_LEVEL 0x01 /* Freerun Timer */
#define HAL_SYNBK_TIMER_INT_LEVEL 0x02 /* Synch Break Detect Timer */

/**** CPU の源振クロック周波数(MHz) ****/
#define HAL_SOURCE_CLK 16 /* source clock MHz */ ⑨

#define HAL_TIMER_TICK 500 /* Freerun Timer tick 500usec */
#define HAL_SYMBRK_TIMER_TICK 1000 /* SynBRK Detect Timer tick 1msec */
#define HAL_PHYERR_TIMER_TICK 8190 /* Physical Bus Error Timer tick 8.19msec */ ⑩

#define HAL_SYNBK_TICK 1 /* 1msec */
#define HAL_PHYERR_TICK 318 /* 2604.42msec */

#endif
```

図 6 TLIN HAL defs.h

5.1.3. tlin_stack_defs.h

以下のコンフィギュレーションを行う。図 7(tlin_stack_defs.h のソース行)に示した番号(赤い丸の囲み番号)にしたがって説明する。

⑪ LIN 仕様で規定される時間パラメータの定義

⑩で定義される HAL_TIMER_TICK を単位時間とする記述になる。つまり、
定義時間(μ sec) = HAL_TIMER_TICK × (以下のマクロ定義値)
となる。

- TICK_MaxByte : TMAX_BYTE 時間の定義(LIN 仕様の規定外の時間)
- TICK_MaxHead : T_{HEADER_MAX} の定義
- TICK_MaxFrame : T_{FRAME_MAX} の定義
- TICK_MaxSlot : T_{SLOT} の定義。LDF の delay パラメータに対応する。
- TICK_SynBRKSent : Synch Break の Dominant 区間の定義
- TICK_WUPDEL : T_{WUDEL} の定義
- TICK_WURTY : T_{WURTY} の定義
- TICK_T3BRK : T_{T3BRK} の定義
- TICK_NOACT : T_{TIME_OUT} の定義

```

#ifndef __TLIN_STACK_DEFS_H__
#define __TLIN_STACK_DEFS_H__
#include "TLinTypes.h"

#define TLIN_ACELL_OK 1

/* 500usec/tick */
#define TICK_MaxByte      7      /* 3.5ms          */
#define TICK_MaxHead     10     /* 5.0ms:49 bit  */
#define TICK_MaxFrame     37     /* 175 bit:8byte  */
#define TICK_MaxSlot     41     /* MaxFrame+2msec */
#define TICK_SynBRKSent   3      /* 1.5 msec       */
#define TICK_WUPDEL       70     /* 35 msec        */
#define TICK_WURTY        100    /* 50 msec        */
#define TICK_T3BRK        3000   /* 1500 msec      */
#define TICK_NOACT        5200   /* 2.6 sec        */

#define TICK_MSTInit      800     /* 400msec        */
#define TICK_MSTStart     1000   /* 500msec        */
#define TICK_SLVInit      800     /* 400msec        */
#define TICK_SLVStart     1200   /* 600msec        */

#define TICK_COMERR       20000  /* 10 sec         */

/* Synch Field & Wake-up Pulse definition */
#define PT_SYNC           0x55    /* sync byte      */
#define PT_WAKEUP         0x80    /* wakeup         */

/* Message Receive waiting time */
#define MAX_MSG_WAIT_TICK 65535

/* Synch Break width adjustment for Slave Node */
#define TICK_SLAVE_ADJUST 2      /* Synch Break Detect Time */

#endif

```

図 7 tlin_stack_defs.h

5.1.4. tlin_master_ram.c, tlin_slave_ram.c

この2つのファイルは、時間測定イベントと仮想タイマ(TID_EVN_TIM0～TID_EVN_TIM11)の association(関連付け)を定義する。

構造体 TLIN_EVT_TM_ASGN tlin_tim_asn_tbl_01[] のメンバの第 1 フィールドが、イベント名称(TLinDefs.h で定義)であり、第 2 フィールドが associate される仮想タイマとなる。テーブルの終了は、{ 0, 0xffff } でなければならない。

```

/*-----
Timer Configuration
-----*/
TLIN_EVT_TM_ASGN tlin_tim_asn_tbl_01[] = {
    EVT_MaxByte,    TID_EVN_TIM0,    /* Timeout: MaxByte */
    EVT_SynBRKSend, TID_EVN_TIM1,    /* Timeout: SynBRK send */
    EVT_MaxHead,    TID_EVN_TIM2,    /* Timeout: MaxHead */
    EVT_MaxFrame,   TID_EVN_TIM3,    /* Timeout: MaxFrame */
    EVT_MaxSlot,    TID_EVN_TIM4,    /* Timeout: MaxSlot */
    EVT_WUPDEL,     TID_EVN_TIM5,    /* Timeout: Wake-up DEL */
    EVT_BUS_NO_ACT, TID_EVN_TIM6,    /* Timeout: Bus No Act */
    0,              0xffff,
};

TLIN_EVT_TM_ASGN tlin_tim_asn_tbl_02[] = {
    EVT_MaxByte,    TID_EVN_TIM0,    /* Timeout: MaxByte */
    EVT_MaxHead,    TID_EVN_TIM1,    /* Timeout: MaxHead */
    EVT_MaxFrame,   TID_EVN_TIM2,    /* Timeout: MaxFrame */
    EVT_WURTY,      TID_EVN_TIM3,    /* Timeout: Wake-up Retry*/
    EVT_T3BRK,      TID_EVN_TIM4,    /* Timeout: T3BRK */
    EVT_BUS_NO_ACT, TID_EVN_TIM5,    /* Timeout: Bus No Act */
    0,              0xffff,
};

void * p_tlin_tim_asn_tbl[] = {
    &tlin_tim_asn_tbl_01,
    &tlin_tim_asn_tbl_02,
    0,
};

```

图 8 tlin_master_ram.c

```

/*-----
Timer Configuration
-----*/
TLIN_EVT_TM_ASGN tlin_tim_asn_tbl_02[] = {
    EVT_MaxByte,    TID_EVN_TIM0,    /* Timeout: MaxByte */
    EVT_SynBRKSend, TID_EVN_TIM1,    /* Timeout: SynBRK send */
    EVT_MaxHead,    TID_EVN_TIM2,    /* Timeout: MaxHead */
    EVT_MaxFrame,   TID_EVN_TIM3,    /* Timeout: MaxFrame */
    EVT_MaxSlot,    TID_EVN_TIM4,    /* Timeout: MaxSlot */
    EVT_WUPDEL,     TID_EVN_TIM5,    /* Timeout: Wake-up DEL */
    EVT_BUS_NO_ACT, TID_EVN_TIM6,    /* Timeout: Bus No Act */
    0,              0xffff,
};

TLIN_EVT_TM_ASGN tlin_tim_asn_tbl_01[] = {
    EVT_MaxByte,    TID_EVN_TIM0,    /* Timeout: MaxByte */
    EVT_MaxHead,    TID_EVN_TIM1,    /* Timeout: MaxHead */
    EVT_MaxFrame,   TID_EVN_TIM2,    /* Timeout: MaxFrame */
    EVT_WURTY,      TID_EVN_TIM3,    /* Timeout: Wake-up Retry*/
    EVT_T3BRK,      TID_EVN_TIM4,    /* Timeout: T3BRK */
    EVT_BUS_NO_ACT, TID_EVN_TIM5,    /* Timeout: Bus No Act */
    0,              0xffff,
};

void * p_tlin_tim_asn_tbl[] = {
    &tlin_tim_asn_tbl_01,
    &tlin_tim_asn_tbl_02,
    0,
};

```

图 9 tlin_slave_ram.c

5.2. OS のコンフィギュレーション

OS のコンフィギュレーションは、使用する OS により異なるものである。ここでは、レファレンス・コードで使った M3T-MR30 の場合を簡単に説明する。

5.2.1. *My_application_name.cfg*

M3T-MR30 の場合は、.cfg ファイルでコンフィギュレーションを定義する。上記の *My_application_name.cfg* の斜体の部分 (*My_application_name*) は、ユーザが自分で好きな名称を用いることができる。

レファレンス・コードでは、マスタ、スレーブ共に smp.cfg とした。ここで注意すべきことは、最終的に実行形式のオブジェクト・ファイルは、Application Task の記述ファイル名 (demo) ではなく、OS コンフィギュレーション・ファイルの名称が使われ smp.x30 となることである。

```
//*****
//
//  COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
//  AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
//
//  MR30 System Configuration File.
//  $Id: smp.cfg,v 1.8 2003/08/22 12:59:49 muraki Exp $
//*****

// System Definition
system{
    stack_size      = 1024;
    priority        = 20;
    system_IPL      = 4;
    message_size    = 32;
    timeout         = YES;
    task_pause      = NO;
};

//System Clock Definition
clock{
    mpu_clock       = 8MHz;
    timer           = A4;
    IPL             = 4;
    unit_time       = 1ms;    // ms
    initial_time    = 0:0:0;
};

//Task Definition
task[1]{
    entry_address =      main();
    stack_size    =      100;
    priority      =      10;
    initial_start =      ON;
};
task[2]{
    entry_address =      task2();
    stack_size    =      50;
    priority      =      9;
};
task[3]{
    entry_address =      task3();
```

```

        stack_size    =    50;
        priority      =    9;
};
task[4]{
    entry_address =    tlin_stack_tsk();
    stack_size    =    200;
    priority      =    3;
};
task[5]{
    entry_address =    tlin_frmcon_tsk();
    stack_size    =    200;
    priority      =    4;
};
// event flag
flag[1]{
    name          =    hal_event;
};
flag[2]{
    name          =    stack_event;
};
flag[3]{
    name          =    fc_event;
};

// Mailbox
mailbox[1]{
    name          =    tlin_stack_mbx;
    buffer_size   =    100;
};
mailbox[2]{
    name          =    tlin_fc_mbx;
    buffer_size   =    100;
};
mailbox[3]{
    name          =    tlin_api_mbx;
    buffer_size   =    100;
};

// Semaphore
semaphore[1]{
    name          =    tlin_entity_sem;
    initial_count =    1;
};

// interrupt handler
interrupt_vector[15]{
    os_int        =    YES;
    entry_address =    HAL_UART_Tx_ch1();
};
interrupt_vector[16]{
    os_int        =    YES;
    entry_address =    HAL_UART_Rx_ch1();
};
interrupt_vector[17]{
    os_int        =    YES;
    entry_address =    HAL_UART_Tx_ch0();
};
interrupt_vector[18]{
    os_int        =    YES;
    entry_address =    HAL_UART_Rx_ch0();
};
// timer A1
interrupt_vector[22]{

```

```

        os_int      =      YES;
        entry_address =      HAL_Timer_Intr();
};
// timer A2
interrupt_vector[23]{
        os_int      =      YES;
        entry_address =      HAL_UART_synBRKdt_timer_Intr_ch0();
};
// timer A3
interrupt_vector[24]{
        os_int      =      YES;
        entry_address =      HAL_UART_synBRKdt_timer_Intr_ch1();
};
// INT1
interrupt_vector[30]{
        os_int      =      YES;
        entry_address =      HAL_UART_synBRKdt_port_Intr_ch1();
};
// INT2
interrupt_vector[31]{
        os_int      =      YES;
        entry_address =      HAL_UART_synBRKdt_port_Intr_ch0();
};

```

図 10 smp.cfg の記述例

5.2.2. コンフィグレーションファイルの定義項目

コンフィグレーションファイルでは以下の項の定義をおこなう。

- システム定義
- システムクロック定義
- 最大項目定義
- タスク定義
- イベントフラグ定義
- セマフォ定義
- メールボックス定義
- 固定長メモリープール定義
- 可変長メモリープール定義
- 周期起動ハンドラ定義
- アラームハンドラ定義
- 割り込みベクタ定義

smp.cfg では、この全てを使って定義している訳ではない。使用しない OS リソースは定義文を記述する必要はない。以下用いた定義文についてのみ説明する。

【システム定義】

<< 形式 >>

```

// System Definition
system{
    stack_size      = システムスタックサイズ ;

```

```

priority      = 優先度の最大値 ;
message_size  = メッセージサイズ ;
system_IPL    = OS 割り込み禁止レベル ;
timeout       = タイムアウト ;
task_pause    = タスクポーズ ;
};

```

<< 内容 >>

1. システムスタックサイズ (バイト)

【 定義形式 】数値

【 定義範囲 】1 以上

システムコール処理および割り込み処理で使用するスタックサイズの合計を定義する。

2. 優先度の最大値 (最低優先度の値)

【 定義形式 】数値

【 定義範囲 】1 ~ 255

MR30 のアプリケーションプログラムの使用する優先度の最大値を定義する。すなわち使用している優先度の最も大きい値を設定する。

3. メッセージサイズ

【 定義形式 】数値

【 定義範囲 】16 or 32

メールボックスのメッセージサイズを指定します。メッセージが 16 ビットのデータの場合は、16 を指定し、32 ビットのデータの場合は、32 を指定する。指定を省略した場合は、16 が設定される。

4. OS 割り込み禁止レベル

【 定義形式 】数値

【 定義範囲 】0 ~ 7

システムコール内での IPL の値、すなわち OS 割り込み禁止レベルを設定する。

5. タイムアウト

【 定義形式 】シンボル

【 定義範囲 】YES or NO

tslp_tsk、twai_flg、twai_sem、trcv_msg を使用している場合は、YES を設定し、使用していない場合は、NO を設定する。

6. タスクポーズ

【 定義形式 】シンボル

【 定義範囲 】YES or NO

PD30 の OS デバッグ機能であるタスクポーズ機能をご使用になる場合は、YES を設定し、使用していない場合は、NO を設定する。

【システムクロック定義】

<< 形式 >>

```
// System Clock Definition
clock{
    mpu_clock      = MPU のクロック;
    timer          = システムクロックに使用するタイマ;
    IPL            = システムクロック割り込み優先レベル;
    unit_time      = システムクロックの単位時間;
    initial_time   = システム時刻の初期値;
};
```

<< 内容 >>

1. MPU のクロック (MHz MHz)

【 定義形式 】周波数

【 定義範囲 】なし

M16C の MPU 動作クロックの周波数を MHz 単位で定義する。0 を定義した場合は、システムクロック割り込みハンドラや OS 依存割り込みハンドラが全く使用できなくなる。

2. システムクロックに使用するタイマ

【 定義形式 】シンボル

【 定義範囲 】

- ・M16C/60 シリーズ
A0～A4, B0～B5, OTHER, NOTIMER
- ・M16C/30 シリーズ
A0～A2, B1～B2, OTHER, NOTIMER
- ・M16C/20 シリーズ
A0～A7, B0～B5, X0～X2, OTHER, NOTIMER
- ・M16C/10 シリーズ
OTHER, NOTIMER

システムクロックに使用するハードウェア・タイマを定義する。上記のシリーズごとのタ

イマの設定範囲はコンフィグレータではチェックしていないので、マイコンにないタイマを指定しないように注意する必要がある。

M16C/10 シリーズでタイマを指定する場合は OTHER を指定して、使用するタイマの設定をスタートアップで行うこと。システムクロックを使用しない場合は、"NOTIMER"を定義する。

3. システムクロック割り込み優先レベル

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義の OS 割り込み禁止レベル)

システムクロック用タイマ割り込みの優先レベルを定義する。1 ～ OS 割り込み禁止レベルまでの値を設定する。

システムクロックの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられない。

4. システムクロックの単位時間 (ms)

【 定義形式 】時間

【 定義範囲 】 $\frac{1}{\text{MPUのクロック (mpu_clock)}}$ ～ $\frac{32 \times 65535}{\text{MPUのクロック (mpu_clock)}}$

システムクロックの単位時間 (システムクロック割り込み発生間隔)を ms 単位で定義する。最小値はユーザーズマニュアルの計算式で計算できるが、計算値が 0.001ms より小さくなる場合は、最小値は 0.001ms となる。コンフィグレーションファイル中で 0.001ms より小さい値を設定すると、コンフィグレータが以下のエラーを返す。

cfg30 Error : illegal clock.unit_time --> <0> near line YY (smp.cfg)

なお、コンフィグレーションファイルで時間間隔を設定する場合には、OS のシステムクロック割り込みハンドラの処理時間よりも大きくする必要がある。この時間は、マイコンの種類や動作条件に依存する。

5. システム時刻の初期値

【 定義形式 】時刻

【 定義範囲 】0 : 0 : 0 ～ 0x7FFF : 0xFFFF : 0xFFFF

システム時刻の初期値を定義する。システム時刻を用いた機能 (set_tim、get_tim、アラームハンドラ)を使用しない場合は、この項目を設定する必要はない。本定義をおこなわないことにより自動的にシステムクロック割り込みハンドラの処理が最適化される。

【最大項目数定義】

この定義は、ROM 化を行う場合のみ設定する項目である。ここでの設定は、複数のアプリケーションの中で、各定義の最大数を定義する。

<< 形式 >>

```
// Max Definition
maxdefine{
    max_task      = 最大タスク定義数;
    max_flag      = 最大イベントフラグ定義数;
    max_mbx       = 最大メールボックス定義数;
    max_sem       = 最大セマフォ定義数;
    max_mpl       = 最大固定長メモリプール定義数;
    max_cyh       = 最大周期起動ハンドラ定義数;
    max_alh       = 最大アラームハンドラ定義数;
};
```

【タスク定義】

<< 形式 >>

```
// Tasks Definition
task[ID 番号]{
    entry_address = タスクの開始アドレス;
    stack_size    = タスクのユーザースタックサイズ;
    priority      = タスクの初期優先度;
    context       = 使用するレジスタ;
    initial_start = 初期起動状態;
};
:
:
```

【イベントフラグ定義】

この定義は、イベントフラグ機能を使用する場合に必ず設定する項目である。

<< 形式 >>

```
// Eventflag Definition
flag[ID 番号]{
    name          = 名前;
};
:
:
```

【セマフォ定義】

この定義は、セマフォ機能を使用する場合に必ず設定する項目である。

<< 形式 >>

```
// Semaphore Definition
semaphore[ID 番号]{
    name          = 名前;
    initial_count = セマフォのカウンタ初期値;
};
      :
      :
```

【メールボックス定義】

この定義は、メールボックス機能を使用する場合に必ず設定する項目である。

<< 形式 >>

```
// Mailbox Definition
mailbox[ID 番号]{
    name          = 名前;
    buffer_size   = メールボックスの最大メッセージ数;
};
      :
      :
```

【割り込みベクタ定義】

<< 形式 >>

```
// Interrupt Vector Definition
interrupt_vector[ベクタ番号]{
    os_int         = OS 依存割り込みハンドラ;
    entry_address  = 割り込みハンドラの開始アドレス;
};
      :
      :
```

割り込みベクタ番号は 0～63、247～255 の範囲まで記述できる。ただし、そのベクタ番号が有効か否かは使用しているマイコンに依存する。コンフィギュレータは、ここで指定した割り込みの割り込み制御レジスタ(IPL 等)や、割り込み要因等の初期設定のコードは生成しない。初期設定はスタートアップファイル中もしくは、開発されるアプリケーションにあわせて作成する必要がある。

6. ビルド・オペレーション

前節までのコンフィギュレーション設定を正しく行えば、統合開発環境を使用している場合は、単純に Build メニューをクリックすれば自動的に実行形式ファイルを生成してくれる。

DOS 環境で行う場合には、いくつかのコマンドを入力する必要がある。

6.1. ビルドのためのコマンド・ライン処理

M3T-NC30WA および M3T-MR30 が正しくインストールされてしまえば、以下の手順でビルドが行われる。

- ① まず、ビルドする対象のソース・コードを、ひとつのフォルダに集める。ここでは、`D:¥TMC_LIN_REF¥>My_App¥Master` を自分の home folder とする。
- ② 次に、WINDOWS のスタート・メニューから、コマンド・プロンプトを起動する。
- ③ ①でソース・コードを格納したフォルダに移動する。
- ④ 以下のように、MR30 のコンフィギュレータを実行する。ここで、コンフィギュレーション・ファイルは、`my_app.cfg` とする。

```
D:¥TMC_LIN_REF¥>My_App¥Master> cfg30 -mV my_app.cfg
```

- ⑤ コンフィギュレータが正しく終了すると、ソース・コード・フォルダ内に `makefile` が作成される。
- ⑥ コマンド・ライン・プロンプトで、以下のように、入力する。

```
D:¥TMC_LIN_REF¥>My_App¥Master> make
```
- ⑦ 以上でビルド作業は終了である。 `My_app.x30` が生成されたことが画面に示される。