TRAVEO™ T2G ファミリの LIN 使用方法

本書について

適用範囲と目的

AN225346 は、TRAVEO™ T2G ファミリ MCU の Local Interconnect Network (LIN) の使い方を説明します。TRAVEO™ T2G の LIN ブロックは、シリアルインタフェースプロトコル LIN と UART に対応します。LIN ブロックは、CPU 処理を削減する LIN フレームの自動送信に対応します。

関連製品ファミリ

TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4 シリーズ

対象者

このドキュメントは、TRAVEO™ T2G ファミリの Local Interconnect Network (LIN) ドライバを使用するすべての人を対象とします。

目次

目次

1 はじめに

LIN は、車載ネットワークで使われる、決定的な低コストのシリアル通信プロトコルです。LIN ブロックには LIN モードと UART モードがあります。このアプリケーションノートは、TRAVEO™ T2G ファミリ MCU の専用 LIN ブロックを用いて、マスタとスレーブの通信の操作方法を説明します。本アプリケーションノートでは、LIN バスが、常にアクティブ状態であることを想定しており、ウェイクアップとスリープモードに関しては、取り扱っていません。

このアプリケーションノートで説明されている内容と、使われる用語をご理解いただくため Architecture Technical Reference Manual (TRM)の Local Interconnect Network (LIN) 章を参照してください。

2 概要

2.1 LIN システム接続図

LIN プロトコルは、1 つのマスタと複数のスレーブからなり、通信のため単線式バスを使用します。図 1 に 2 つの LIN ノードを持つ LIN クラスタの基本的な構成を示します。

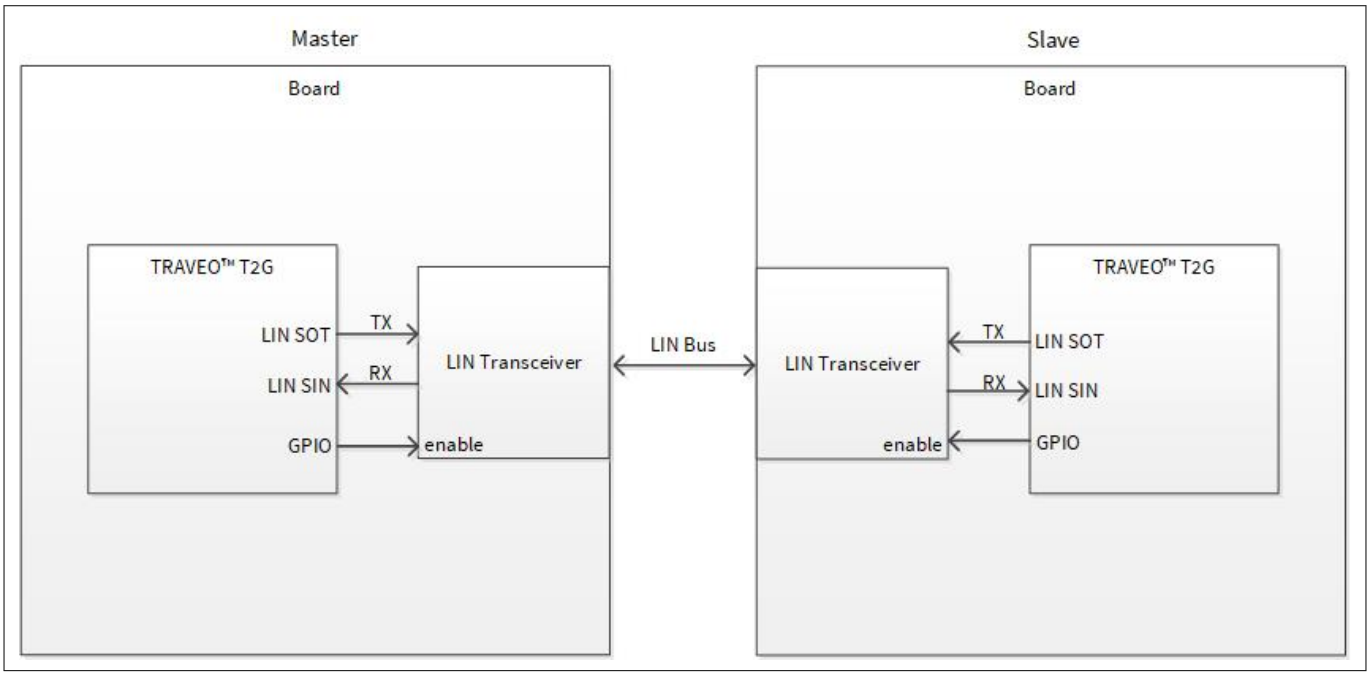

図 1 LIN マスタ/スレーブ接続例

2.2 メッセージフレームフォーマット

図 2 の様に LIN メッセージフレームは、以下のヘッダと応答から構成されます。

- ヘッダ: マスタによってのみ送信され Break 領域, Sync 領域, および保護識別子 (PID) 領域から構成されます。
- 応答: マスタまたは、スレーブから送信され、最大 8 つのデータ領域とチェックサム領域から構成されます。

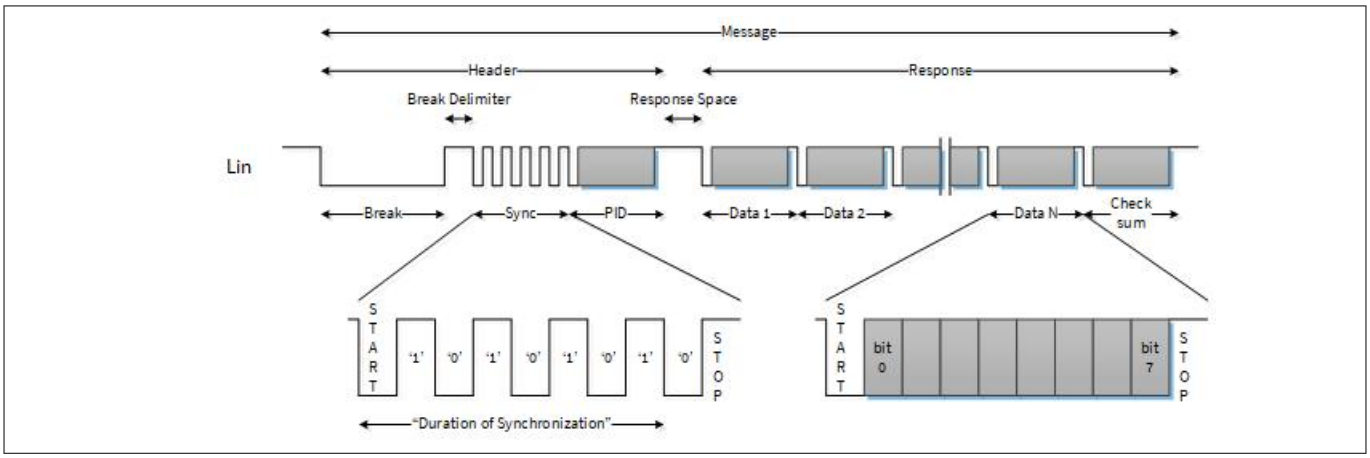

図 2 LIN メッセージフレームフォーマット

LIN メッセージフレームフォーマットの詳細は Architecture TRM を参照してください。

2.3 ボーレート設定

ボーレートは、PERI クロックから分周され、各チャネルに設定できます。PERI クロックはペリフェラル クロック分周器を介して LIN ブロックに入力されます。ボーレートは、ペリフェラル クロック分周器の値により設定されます。さらに LIN チャネルには、16 の固定信号オーバーサンプリング係数があります。したがって、ボーレートは式 1 のように計算されます。

式 1

$$Baud\ Rate = \frac{PERI\ clock}{16 \times Divider\ value}$$

式 2 は、PERI クロック 24 MHz で、必要なボーレート 20 kbps (20 kHz) の時の分周設定値の計算例を示します。

式 2

$$Divider\ value = \frac{PERI\ clock}{16 \times Baud\ Rate}$$

$$= \frac{24\ MHz}{16 \times 20\ kHz} = 75$$

PERI クロック, ペリフェラル クロック分周器, および分周設定値の詳細は Architecture TRM の Clocking System 章を参照してください。

3 LIN 通信例

ここでは、サンプルドライバライブラリ (SDL) を使用して LIN 通信を実装する方法について説明します。このアプリケーションノートのコードは SDL の一部です。SDL についてはその他の参考資料を参照してください。

SDL には、設定部とドライバ部があります。設定部は、目的の操作のためのパラメータ値を設定します。ドライバ部は、設定部のパラメータに基づいて各レジスタを設定します。システムに応じて設定部を設定できます。

LIN は、周期設定した時間で動作ができるよう、LIN マスタは、参照タイマにより定期的に起動するスケジューラを持ち、バス動作を制御します。あらゆるフレームは、あらかじめ定義されたスロットによって送信されます。各 LIN フレームは、マスタヘッダから始まります。

さらに LIN マスタは、スケジュール表を持ち、この表はタイムスロットに分割されます。すべてのタイムスロットの応答フレームが送受信されることで、スケジュールは完了します。スケジューラを再トリガすることによって、スケジュール表は繰り返し実行されます。しかし、マスタはスケジュール表を別のものと取り替える柔軟性も持ちます。

スケジュール表では、フレーム ID, メッセージタイプ, データ長, 応答で使われるチェックサムのタイプのような各タイムスロットの通信設定が、あらかじめ決められています。メッセージタイプは、応答の送信者を定義します。複数のスレーブがある場合、メッセージタイプは、応答を送信するスレーブを定義します。LIN 通信では、クラシックモードとエンハンスモードの 2 種類のチェックサムタイプに対応します。クラシックモードでは、PID 領域は、チェックサムの計算に含まれず、データ領域での計算のみを含みます。一方、エンハンスモードでは、PID 領域とデータ領域の両方が、チェックサムの計算に含まれます。チェックサムタイプは、LIN_CH_CTL レジスタの CHECKSUM_ENHANCED ビットにより選択されます。チェックサムタイプの詳細は、Registers TRM を参照してください。

表 1 に、スケジュール表の例を示します

表 1 スケジュール表の例

タイムスロット	ID	メッセージタイプ	データ長	チェックサムタイプ
1	0x01	スレーブ応答	8	エンハンス
2	0x02	マスタ応答	8	エンハンス
3	0x10	スレーブ応答	1	エンハンス
4	0x11	マスタ応答	1	エンハンス
5	0x20	Slave-to-Slave	-	エンハンス

この例では、スケジュール表は、1 から 5 の 5 個のタイムスロットからなります。

タイムスロット 1 のメッセージタイプは、スレーブ応答でデータ長は 8 です。したがって、ヘッダがスケジューラのトリガによって送られるとき、LIN スレーブは、8 バイトの応答データをマスタへ送ります。

タイムスロット 4 では、マスタ応答でデータ長は 1 バイトです。マスタは、ヘッダとともに 1 バイトのデータをスレーブへ送ります。

タイムスロット 5 は、Slave-to-Slave 応答を定義します。この場合、応答は専用のスレーブノードの間でのみ行われ、マスタは応答を無視できます。

図 3 に、表 1 に示したスケジュールしたがって、マスタとスレーブ間の LIN 通信の例を示します。

3 LIN 通信例

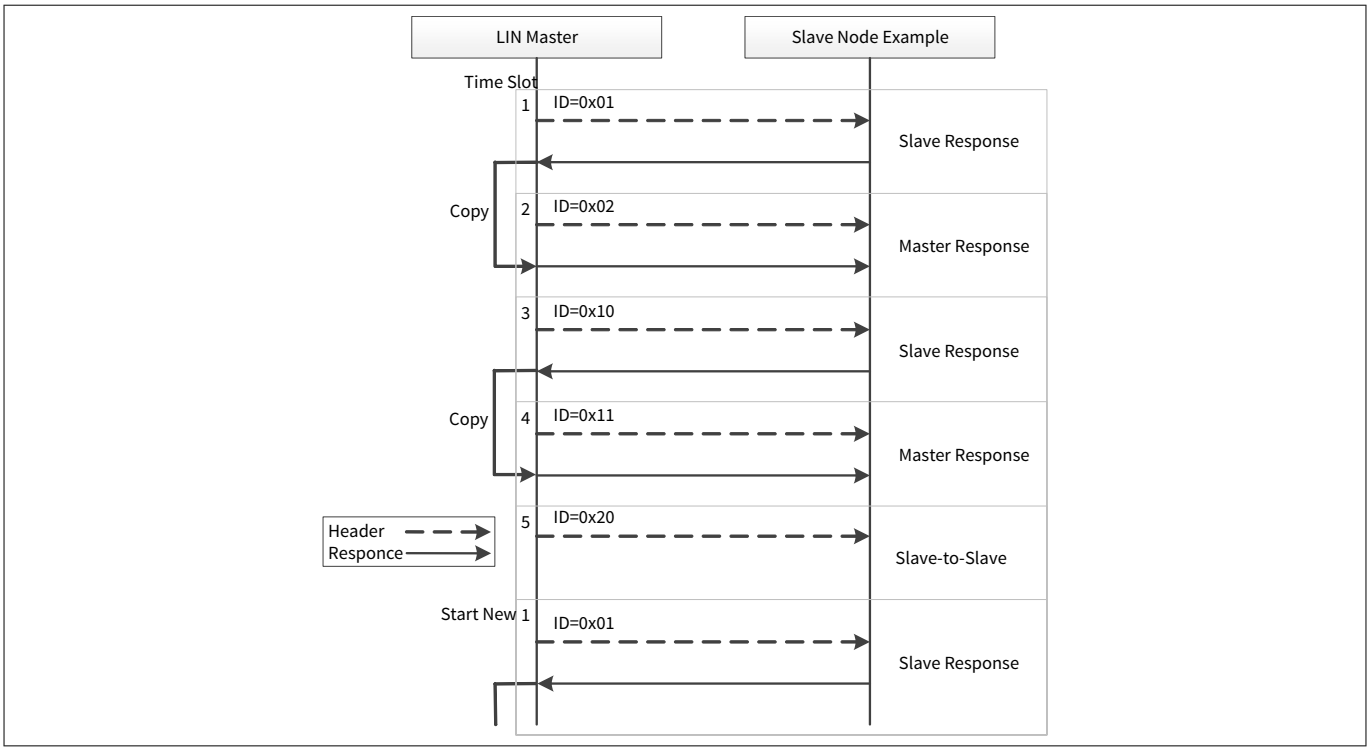

図 3　　LIN マスタとスレーブ間の通信

1. マスタは、スケジューラの起動後、ID = 0x01 のヘッダを送ります。
2. スレーブがヘッダを受信した後、スレーブは、スケジュール表によって、8 バイトの応答をマスタに送ります (タイムスロット 1)。
3. マスタが応答を受信した時、タイムスロット 1 のフレームは完了し、そしてマスタは次のスケジューラの起動を待ちます。
4. スケジューラが起動したとき、マスタは ID = 0x02 のヘッダを送ります。
5. マスタがヘッダを送信した後、マスタは 8 バイトの応答をスレーブに送ります (タイムスロット 2)。そして、マスタは、次のスケジューラの起動を待ちます。
6. この動作を、最後のタイムスロット 5 まで繰り返します。
7. タイムスロット 5 の動作が完了した後、タイムスロット 1 を始める次のスケジューラが起動します。

3.1　　LIN メッセージ通信

SDL では、ヘッダ/応答の送受信のような、異なったメッセージタイプに対応するために、LIN マスタまたは LIN スレーブの動作モードの処理は、次のコマンドによって行われます。

- `LIN_CMD_TX_HEADER`: このコマンドは、ヘッダを送信するために、マスタが使用します。
- `LIN_CMD_TX_RESPONSE`: このコマンドは、応答を送信するために、マスタまたはスレーブが使用します。
- `LIN_CMD_RX_RESPONSE`: このコマンドは、応答を受信するために、マスタまたはスレーブが使用します。

これらのコマンドは、表 1 内のメッセージタイプに対応し設定されます。詳細は、セクション 4 およびセクション 5 を参照してください。

3.2　　イベント生成

LIN ブロックは、送信完了, 受信完了, およびエラー検出などの割込みイベントを生成します。各 LIN チャネルは専用の割込み信号と独自の割込みレジスタ (`LIN_CH_INTR`, `LIN_CH_INTR_SET`, `LIN_CH_INTR_MASK`, および `LIN_CH_INTR_MASKED`) を持ちます。この通信例では、INTR_MASK が割込み生成を制御し、`LIN_CH_CMD.INTR_MASKD` レジスタが割込みソースをチェックします。

3 LIN 通信例

表 2 に、SDL でマスタとスレーブが検出した送受信イベントを示します。

表 2　　送受信イベントリスト

送受信イベント	マスタ	スレーブ
TX_HEADER_DONE	✓	-
RX_HEADER_DONE	-	✓
TX_RESPONSE_DONE	✓	✓
TX_WAKEUP_DONE	✓	✓
RX_RESPONSE_DONE	✓	✓
RX_BREAK_WAKEUP_DONE	✓	✓
RX_HEADER_SYNC_DONE	-	✓

表 3 はマスタとスレーブにより、検出されるエラーイベントリストを示します。

表 3　　エラーイベントリスト

エラーイベント	マスタ	スレーブ
RX_NOISE_DETECT	✓	✓
TIMEOUT	✓	✓
TX_RESPONSE_BIT_ERROR	✓	✓
RX_HEADER_SYNC_ERROR	-	✓
RX_RESPONSE_FRAME_ERROR	✓	✓
RX_RESPONSE_CHECKSUM_ERROR	✓	✓
TX_HEADER_BIT_ERROR	✓	-

関連する割込みレジスタは、これらのイベントに対応するビットを備えます。対応するビットを設定またはクリアすることによって、ソフトウェアはイベントの生成を制御できます。
各割込みレジスタとイベントの詳細は、Architecture TRM と Registers TRM を参照してください。

4 マスタの操作例

表 1 を使った LIN マスタの操作例を示します。SDL では、コマンドを使用して次のステートマシンを管理できます。図 4 に、LIN マスタステートマシンの動作を示します。

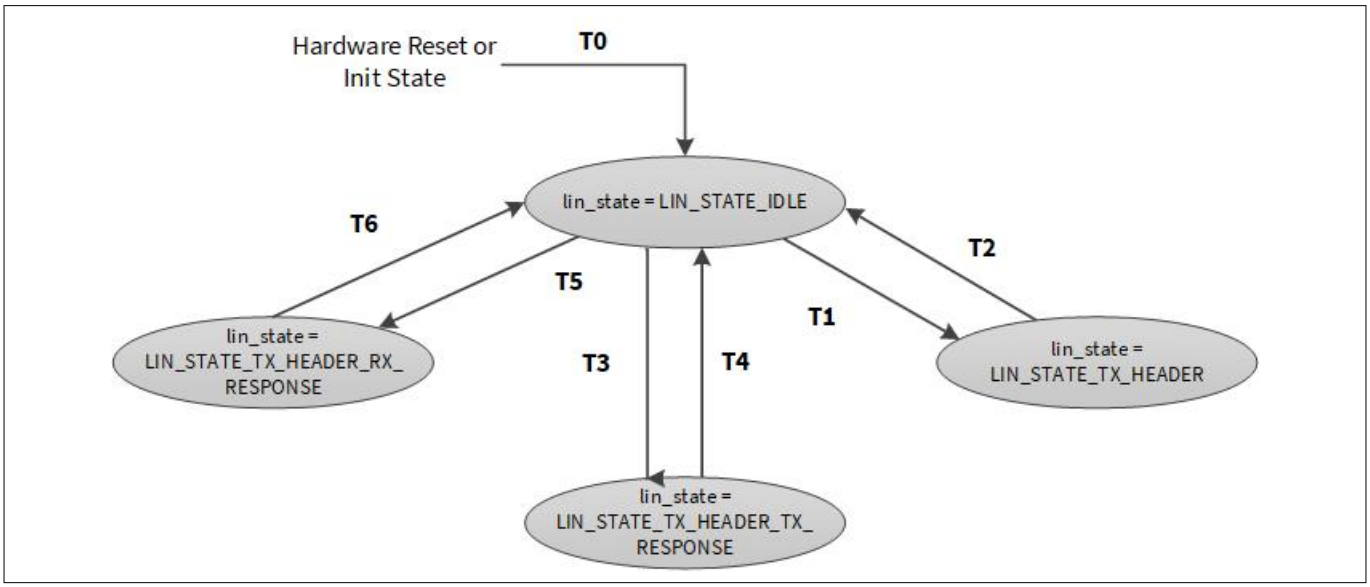

図 4 LIN マスタステートマシン

LIN マスタステートマシンは、以下の 4 つの状態があります。

- `LIN_STATE_IDLE`: これは、初期化後のデフォルト状態です。LIN マスタ IRQ ハンドラが完了すると、この状態になります。
- `LIN_STATE_TX_HEADER_RX_RESPONSE`: これは、メッセージタイプがスレーブ応答の状態です。マスタは、ヘッダを送信し、スレーブからの応答を待ちます。
- `LIN_STATE_TX_HEADER_TX_RESPONSE`: これは、メッセージタイプがマスタ応答の状態です。マスタは、ヘッダと応答をスレーブに送信します。
- `LIN_STATE_TX_HEADER`: これはメッセージタイプが Slave-to-Slave の状態です。マスタは、ヘッダのみ送信します。

ソフトウェアは、スケジュール表のメッセージタイプによって状態を決定し、その時の状態によって、コマンドシーケンスを設定します。

表 4 に、メッセージタイプ, 状態, およびコマンドシーケンスの関係を示します。

表 4 LIN マスタのメッセージタイプ, 状態, およびコマンドシーケンス設定の対応

メッセージタイプ	状態	コマンド			
		TX_HEADER	RX_HEADER	TX_RESPONSE	RX_RESPONSE
スレーブ応答	LIN_STATE_TX_HEADER_RX_RESPONSE	1	0	0	1
マスタ応答	LIN_STATE_TX_HEADER_TX_RESPONSE	1	0	1	0
Slave-to-Slave	LIN_STATE_TX_HEADER	1	0	0	0

以下は、これらのプロセスを実行するための初期化と割込み制御の例です。

4 マスタの操作例

4.1 LIN マスタの初期化

図 5 に、LIN マスタ初期化のフロー例を示します。

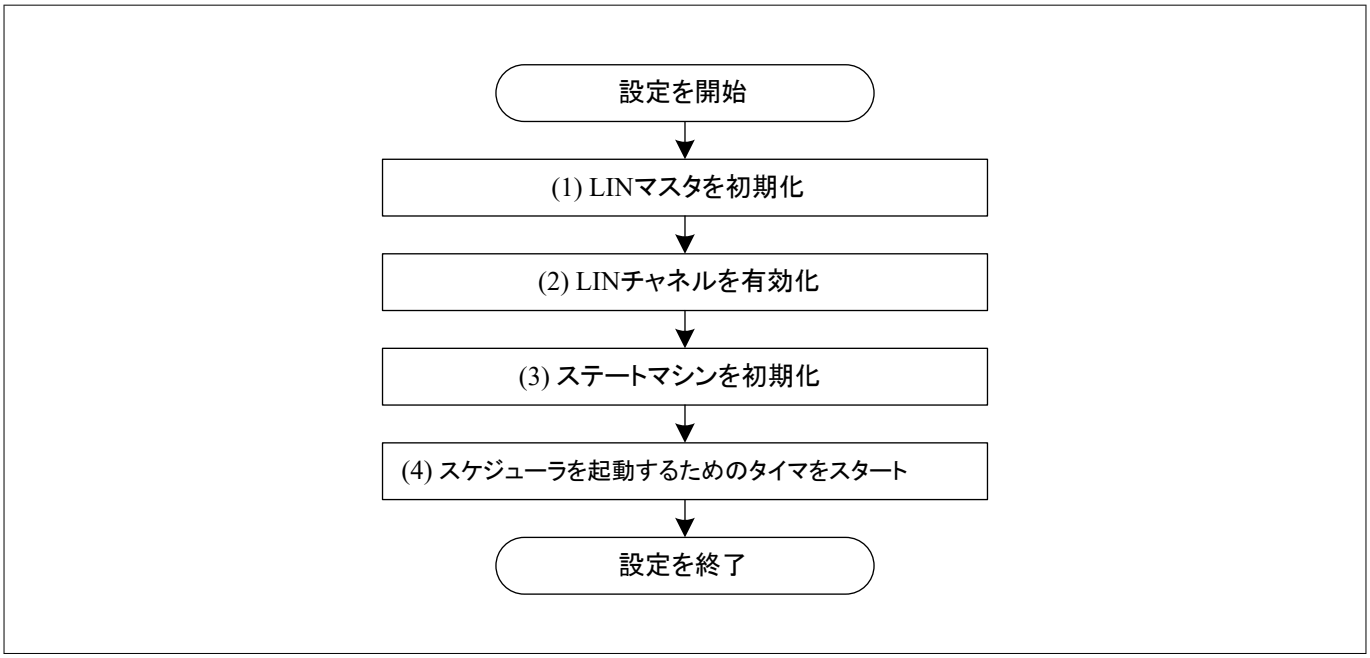

図 5 LIN マスタ初期化フロー例

1. LIN マスタを初期化してください。
2. LIN チャネルを有効にしてください。
   ポート設定完了後、ソフトウェアにより外部 LIN トランシーバを有効にします。この設定手順の 4 において `LIN_CH_CTL0.AUTO_EN` を“0”に設定するため、この例では、外部 LIN トランシーバを制御しません。この場合、ソフトウェアはレジスタビット `TX_RX_STATUS.EN_OUT` を介して EN ピンを制御します。設定されたチャネルの `LIN_EN_OUT` ピンが MCU で使用できない場合は、トランシーバの EN ピンも通常の GPIO 出力で制御できます。
3. ソフトウェアステートマシンを初期化してください。
   現在の状態を `lin_state = LIN_STATE_IDLE` に設定してください。
4. スケジューラを始めるために、タイマを始動してください。
   この設定でスケジューラを始めるとき、自動的に通信が始まります。

クロック設定、ポート設定、および割込みコントローラー設定の詳細は、Architecture TRM と Registers TRM を参照してください。

4.1.1 ユースケース

ここでは、以下のパラメータを使用した LIN マスタ初期化の使用例について説明します。

- マスタ/スレーブノード: マスタノード
- LIN インスタンス: LIN0_CH0
- ボーレート: 19231 Hz

4.1.2 設定と例

表 5 に、LIN マスタ初期化用の SDL の設定部のパラメータを示します。

4 マスタの操作例

表 5　　LIN マスタ初期化パラメータリスト

パラメータ	説明	設定値
CLK 用		
CY_LINCH0_PCLK	周辺クロック番号	PCLK_LIN0_CLOCK_CH_EN0
LIN 用		
config.bMasterMode	マスタまたはスレーブモード	true (マスタモード)
config.bLinTransceiverAutoEnable	LIN トランシーバ自動有効	true (有効)
config.u8BreakFieldLength	ビット時間のブレイク/ウェイクアップ長 (1 を減算)	13ul (13-1 = 12 bit)
config.enBreakDelimiterLength	ブレイクデリミタ長	LinBreakDelimiterLength1bits (1 bit)
config.enStopBit	ストップビット時間	LinOneStopBit (1 bit)
config.bFilterEnable	受信フィルタ	true
CY_LINCH0_TYPE	使用する LIN チャネル番号	LIN0 の channel 0

Code Listing 1 に、設定部での LIN マスタを初期化するためのサンプルプログラムを示します。

4 マスタの操作例

Code Listing 1 CYT2 シリーズ: 設定部での LIN 初期化例 (マスタ)

```
int main(void)
{
    :
    /* LIN baudrate setting */
    {
        /* Note:
         * LIN IP does oversampling and oversampling count is fixed 16.
         * Therefore LIN baudrate = LIN input clock / 16.
         */
         Cy_SysClk_PeriphAssignDivider(CY_LINCH0_PCLK, CY_SYSCLK_DIV_16_BIT, 0ul); /*Configure
the Baud Rate Clock*1*/
         /*Configure the Baud Rate Clock*1*/
         Cy_SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, 0ul, 259ul); // 80 MHz / 260 /
16 (oversampling) = 19231 Hz
Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, 0ul);   /*Configure the Baud Rate Clock*1*/
    }

    :
    /* Initialize LIN */
    {
    stc_lin_config_t config =    /*Configure LIN Master parameters*/
        {
            .bMasterMode = true,
            .bLinTransceiverAutoEnable = true,
            .u8BreakFieldLength = 13ul,
            .enBreakDelimiterLength = LinBreakDelimiterLength1bits,
            .enStopBit = LinOneStopBit,
            .bFilterEnable = true
        };

/* (1)Initialize LIN Master based on above structure (See Code Listing 3) */
        /* (2)Enable LIN_CH0 (See Code Listing 3)  */
        Lin_Init(CY_LINCH0_TYPE, &config);

lin_state = LIN_STATE_IDLE;      /*(3)Initialize the state machine*/
    }

    /* Start scheduling */
    SchedulerInit();             /*(4)Start the Timer (See Code Listing 2)*/
}
```

*1: 詳細は、Architecture TRM の Clocking System セクションを参照してください。

Code Listing 2 に、SchedulerInit の例を示します。

4 マスタの操作例

Code Listing 2 SchedulerInit の例

```
static void SchedulerInit(void)
{
    Cy_SysTick_Init(CY_SYSTICK_CLOCK_SOURCE_CLK_CPU, SYSTICK_RELOAD_VAL);
    Cy_SysTick_SetCallback(0ul, LIN0_TickHandler);
    Cy_SysTick_Enable();                          /*(4)Start the timer*/
}
```

Code Listing 3 に、ドライバ部で LIN を構成するためのサンプルプログラムを示します。
次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- pstcLin->unCTL0 は、Registers TRM に記載されている LINx_CHy_CTL0 レジスタです。他のレジスタも同様に記述されます。'x'は LIN インスタンス番号を示し、'y'はチャネル番号を示します。
- パフォーマンス改善策:

レジスタ設定のパフォーマンスを向上させるために、SDL は完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドは、ビット書き込み可能なバッファで事前に生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
ctl0.stcField.u1BIT_ERROR_IGNORE = 0ul;
ctl0.stcField.u1PARITY = 0ul;
ctl0.stcField.u1PARITY_EN = 0ul;
pstcLin->unCTL0.u32Register = ctl0.u32Register;
```

- レジスタの共用体と構造体の詳細については、hdr/rev_x/ip の下の"cyip_lin.h"を参照してください。

4 マスタの操作例

Code Listing 3 Lin_Init

```
/*******************************************************************************
* Function Name: Lin_Init
*******************************************************************************/
cy_en_lin_status_t Lin_Init( volatile stc_LIN_CH_t* pstcLin, const stc_lin_config_t *pstcConfig)
{
    cy_en_lin_status_t status = CY_LIN_SUCCESS;

    /* Check if pointers are valid */
    if ( ( NULL == pstcLin )          ||    /*Check if parameter values are valid.*/
         ( NULL == pstcConfig ) )
    {
        status = CY_LIN_BAD_PARAM;
    }
    else if (pstcConfig->bMasterMode &&
    ((LIN_MASTER_BREAK_FILED_LENGTH_MIN > pstcConfig->u8BreakFieldLength) ||
              (LIN_BREAK_WAKEUP_LENGTH_BITS_MAX < pstcConfig->u8BreakFieldLength)))
    {
        status = CY_LIN_BAD_PARAM;
    }
    else if (LIN_BREAK_WAKEUP_LENGTH_BITS_MAX < pstcConfig->u8BreakFieldLength)
    {
        status = CY_LIN_BAD_PARAM;
    }
    else
    {
        un_LIN_CH_CTL0_t ctl0 = { 0ul };

        /* Stop bit length */
        ctl0.stcField.u2STOP_BITS = pstcConfig->enStopBit; /*(1)Initialize the LIN with
parameter*/
        /* LIN Transceiver Auto Enable by Hardware */
        ctl0.stcField.u1AUTO_EN = pstcConfig->bLinTransceiverAutoEnable; /*(1)Initialize the
LIN with parameter*/
        /* Break field length */
        ctl0.stcField.u5BREAK_WAKEUP_LENGTH = pstcConfig->u8BreakFieldLength - 1ul; /
*(1)Initialize the LIN with parameter*/
        /* Break Delimiter Length: Bit8-9 */
        /* This field effect only master node header transmission. */
        ctl0.stcField.u2BREAK_DELIMITER_LENGTH = pstcConfig->enBreakDelimiterLength; /
*(1)Initialize the LIN with parameter*/
        /* Mode of Operation: Bit 24: 0 -> LIN Mode, 1 -> UART Mode */ /*(1)Initialize the LIN
with parameter*/
        ctl0.stcField.u1MODE = 0ul;    /*(1)Initialize the LIN with parameter*/
        /* Enable the LIN Channel */
        ctl0.stcField.u1ENABLED = 1ul; /*(2)Enable LIN_CH0*/
        /* Filter setting */
        ctl0.stcField.u1FILTER_EN = pstcConfig->bFilterEnable;    /*(1)Initialize the LIN with
parameter*/
        /* Other settings are set to default */
        ctl0.stcField.u1BIT_ERROR_IGNORE = 0ul /*(1)Initialize the LIN with parameter*/
        ctl0.stcField.u1PARITY = 0ul;    /*(1)Initialize the LIN with parameter*/
```

```
        ctl0.stcField.u1PARITY_EN = 0ul;    /*(1)Initialize the LIN with parameter*/
        pstcLin->unCTL0.u32Register = ctl0.u32Register;   /*(1)Initialize the LIN with
parameter*/

    }
    return status;
}
```

4.2 LIN マスタの LIN 通信フロー例

LIN 通信が開始されると、マスタスケジューラハンドラは割込みによって起動します。図 6 に、マスタスケジューラハンドラがどのように動作するかの例を示します。

4 マスタの操作例

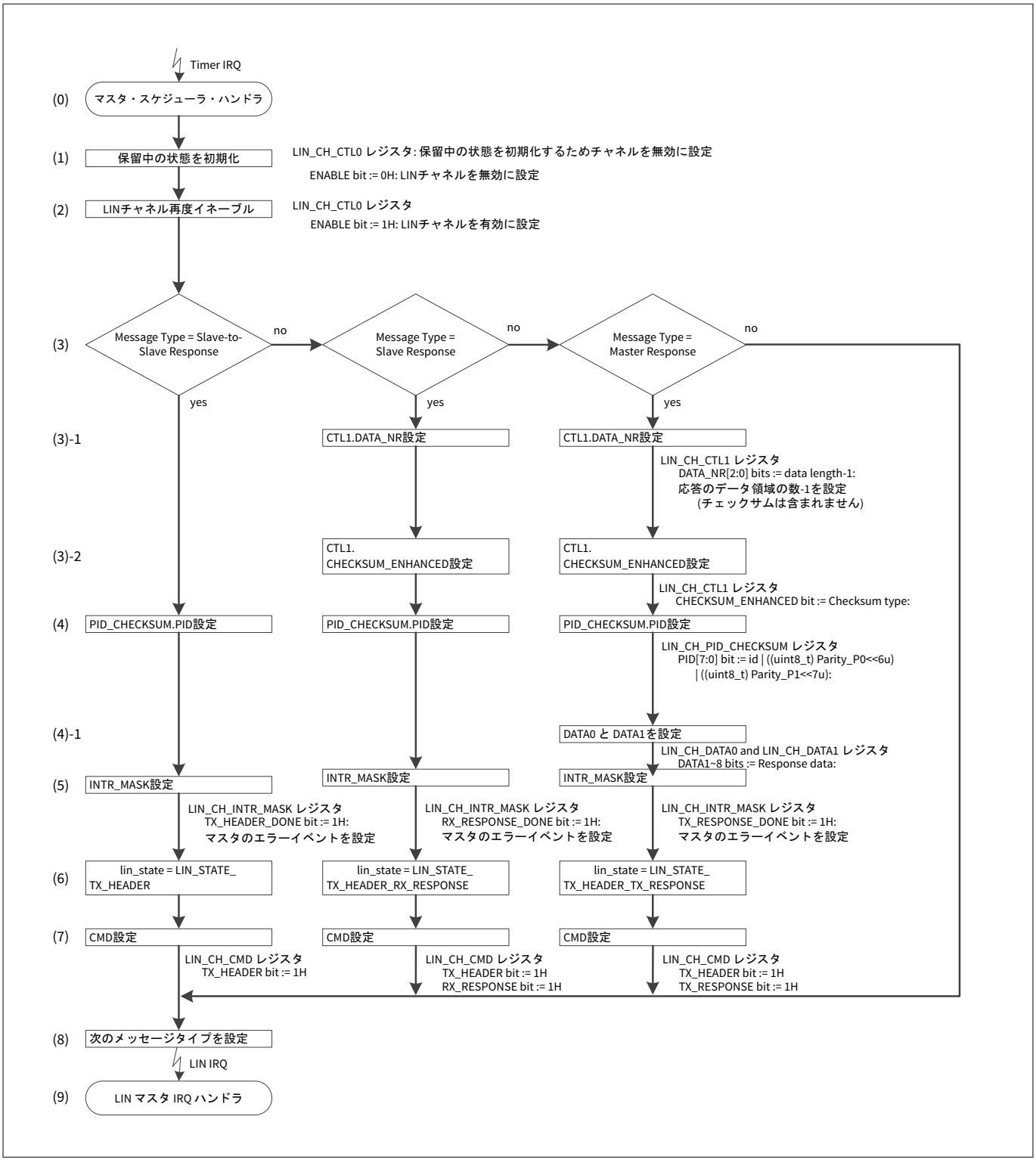

図 6　　マスタスケジューラハンドラの例

以下は、スケジューラのためのアプリケーションソフトウェアの操作です。

(0) タイマ IRQ は、LIN マスタのマスタスケジュールハンドラを起動させます。

(1) `LIN_CH_CTL0.ENABLE` を“0”に設定することにより、保留中の状態を初期化してください。すべての非保持された MMIO レジスタ (例えば、`LIN_CH_STAUS`, `LIN_CH_CMD`, および `LIN_CH_INTR` レジスタ) `LIN_CH_CTL0.ENABLE` を“0”に設定することによりデフォルト値にリセットされます。初期化されるレジスタの詳細は Registers TRM を参照してください。

(2) LIN チャネルを再度有効にしてください。

(3) 次のフレームのメッセージタイプをチェックしてください。これは、現在のスケジューラで特定されたメッセージタイプです。もしメッセージタイプが、マスタ応答かスレーブ応答であれば、それに応じて応答のデータ長 (3)-1 とチェックサムタイプ (3)-2 を設定してください。

(4) ヘッダの PID 領域を設定してください。LIN_CH_PID_CHECKSUM.PID[7]は parity [1]であり、LIN_CH_PID_CHECKSUM.PID[6]は parity [0]、そして LIN_CH_PID_CHECKSUM.PID[5:0]は ID です。ソフトウェアにより PID 領域パリティビット P[1]と P[0]を計算する必要があります。パリティは以下の様に計算されます。

P[0] = (ID[4] ^ ID[2] ^ ID[1] ^ ID[0])

P[1] = ! (ID[5] ^ ID[4] ^ ID[3] ^ ID[1])

- マスタ応答の場合: LIN マスタは要求されたデータ長の応答データをデータレジスタ (DATA 0/1) に設定してください。(4)-1

(5) LIN_CH_INTR_MASK レジスタは、ケースによってイベント割込みを可能にしてください。

- Slave-to-Slave 応答:

  TX_HEADER_DONE を“1”に設定してください。

  エラー検出ビットを“1”に設定してください。
- スレーブ応答:

  RX_RESPONSE_DONE を“1”に設定してください。

  エラー検出ビットを“1”に設定してください。

  –
- マスタ応答:

  TX_RESPONSE_DONE を“1”に設定してください。

  エラー検出ビットを“1”に設定してください。

システムによって必要な検出ビットを設定することが必要です。

(6) 現在のメッセージタイプによって、状態を設定してください。

- Slave-to-Slave 応答:

  lin_state を LIN_STATE_TX_HEADER に設定してください。
- スレーブ応答:

  lin_state を LIN_STATE_TX_HEADER_RX_RESPONSE に設定してください。
- マスタ応答:

  lin_state を LIN_STATE_TX_HEADER_TX_RESPONSE に設定してください。

(7) ケースごとの状態によって、コマンドシーケンスを設定してください。

- Slave-to-Slave 応答:

  LIN_CH_CMD.TX_HEADER を“1”に設定してください。
- スレーブ応答:

  LIN_CH_CMD.TX_HEADER を“1”に設定してください。

  LIN_CH_CMD.RX_RESPONSE を“1”に設定してください。
- マスタ応答:

  LIN_CH_CMD.TX_HEADER を“1”に設定してください。

  LIN_CH_CMD.TX_RESPONSE を“1”に設定してください。(ヘッダ送信後、応答は送られます。)

(8) 表 1 によって、次のスケジューラ起動のためにメッセージタイプを設定してください。

(9) スケジューラ (タイマ割込み) から戻り、表 2 のように設定された LIN の割込み発生を待ちます。

4.2.1 ユースケース

ここでは、メッセージタイプを判別して LIN マスタ通信を実行する例について説明します。

- マスタ/スレーブノード: マスタノード
- LIN インスタンス: LIN0_CH0
- 通信操作: 表 1 とセクション 4 を参照してください。

4.2.2 設定と例

表 6 に、LIN 通信 (LIN マスタ) 用 SDL の設定部のパラメータを示します。

表 6 LIN 通信パラメータのリスト

パラメータ	説明	設定値
LIN 用		
msgContext[]	ID/ Message タイプ	0x01ul / LIN_RX_RESPONSE 0x02ul / LIN_TX_RESPONSE 0x10ul / LIN_RX_RESPONSE 0x11ul / LIN_TX_RESPONSE 0x20ul / LIN_TX_HEADER
	チェックサムタイプ	LinChecksumTypeExtended
	データ長	8ul または 1ul
CY_LINCH0_TYPE	使用する LIN チャネル番号	LIN0 の channel 0

Code Listing 4 に、設定部での LIN を通信するためのサンプルプログラムを示します。

4 マスタの操作例

Code Listing 4 CYT2 シリーズ: 設定部での LIN 通信例 (マスタ)

```
lin_message_context msgContext[] =
{
{0x01ul, LIN_RX_RESPONSE, LinChecksumTypeExtended, 8ul,},     /*Set msgContext 表 1 */
    {0x02ul, LIN_TX_RESPONSE, LinChecksumTypeExtended, 8ul,},     /*Set msgContext 表 1 */
    {0x10ul, LIN_RX_RESPONSE, LinChecksumTypeExtended, 1ul,},     /*Set msgContext 表 1 */
    {0x11ul, LIN_TX_RESPONSE, LinChecksumTypeExtended, 1ul,},     /*Set msgContext 表 1 */
    {0x20ul, LIN_TX_HEADER, LinChecksumTypeExtended, 8ul,},     /*Set msgContext 表 1 */
};
    :
int main(void)
{
    :

/* Master schedule handler */
static void LIN0_TickHandler(void)
{
    /* Disable the channel for clearing pending status */
    Lin_Disable(CY_LINCH0_TYPE);     /* 1)Initializes the current pending state for LIN0_CH0
Code Listing 5 */

    /* Re-enable the channel */
    Lin_Enable(CY_LINCH0_TYPE);     /* (2)Re-enables the LIN channel Code Listing 6 */

    switch(msgContext[scheduleIdx].responseDirection) /* (3)Checks the message type */
    {
    case LIN_TX_RESPONSE:
        /* Response Direction = Master to Slave */
        /* (3)-1 Configure the data length of the response field Code Listing 7 */
        /* (3)-2 Configure the checksum type Code Listing 8 */
        Lin_SetDataLength(CY_LINCH0_TYPE, msgContext[scheduleIdx].dataLength);

        Lin_SetChecksumType(CY_LINCH0_TYPE, msgContext[scheduleIdx].checksumType);



        /* (4)Configure the PID field Code Listing 9 */
        Lin_SetHeader(CY_LINCH0_TYPE, msgContext[scheduleIdx].id);

        /* (4)-1 Configure the data register Code Listing 10 */
        Lin_WriteData(CY_LINCH0_TYPE, msgContext[scheduleIdx].dataBuffer,
msgContext[scheduleIdx].dataLength);


        /* (5)Enables the event interrupt Code Listing 11 */
        Lin_SetInterruptMask(CY_LINCH0_TYPE, LIN_INTR_TX_RESPONSE_DONE |
LIN_INTR_ALL_ERROR_MASK_MASTER);


        /* (6)Configure the lin_state */
        lin_state = LIN_STATE_TX_HEADER_TX_RESPONSE;
```

4 マスタの操作例

```
            /* (7)Configure the command sequence Code Listing 12 */
            Lin_SetCmd(CY_LINCH0_TYPE, LIN_CMD_TX_HEADER_TX_RESPONSE);
            break;

        case LIN_RX_RESPONSE:
            /* Response Direction = Slave to Master */
            /* (3)-1 Configure the data length of the response field Code Listing 7 */
            Lin_SetDataLength(CY_LINCH0_TYPE, msgContext[scheduleIdx].dataLength);

    /* (3)-2 Configure the checksum type Code Listing 8 */
            Lin_SetChecksumType(CY_LINCH0_TYPE, msgContext[scheduleIdx].checksumType);

    /* (4)Configure the PID field Code Listing 9 */
            Lin_SetHeader(CY_LINCH0_TYPE, msgContext[scheduleIdx].id);

    /* (5)Enables the event interrupt Code Listing 11 */
            Lin_SetInterruptMask(CY_LINCH0_TYPE, LIN_INTR_RX_RESPONSE_DONE |
    LIN_INTR_ALL_ERROR_MASK_MASTER);

            /* (6)Configure the lin_state */
            lin_state = LIN_STATE_TX_HEADER_RX_RESPONSE;

    /* (7)Configure the command sequence Code Listing 12 */
            Lin_SetCmd(CY_LINCH0_TYPE, LIN_CMD_TX_HEADER_RX_RESPONSE);
            break;

        case LIN_TX_HEADER:
            /* Response Direction = Slave to Slave */
            /* (4)Configure the PID field Code Listing 9 */
            Lin_SetHeader(CY_LINCH0_TYPE, msgContext[scheduleIdx].id);
            /* (5)Enables the event interrupt Code Listing 11 */
            Lin_SetInterruptMask(CY_LINCH0_TYPE, LIN_INTR_TX_HEADER_DONE |
    LIN_INTR_ALL_ERROR_MASK_MASTER);


            /* (6)Configure the lin_state */
            lin_state = LIN_STATE_TX_HEADER;
            /* (7)Configure the command sequence Code Listing 12 */
            Lin_SetCmd(CY_LINCH0_TYPE, LIN_CMD_TX_HEADER);
            break;
        default:
            break;
        }

        /* (8)Configure the message type for the next scheduler activation */
        scheduleIdx = (scheduleIdx + 1ul) % (sizeof(msgContext) / sizeof(msgContext[0ul]));
    }
```

Code Listing 5 から Code Listing 12 に、ドライバ部で LIN を通信する例を示します。

4 マスタの操作例

Code Listing 5 Lin_Disable

```
/*******************************************************************************
 ** \brief Disable LIN channel.
*******************************************************************************/
cy_en_lin_status_t Lin_Disable(volatile stc_LIN_CH_t* pstcLin)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if (NULL == pstcLin)                    /* Check if parameter values are valid */
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        pstcLin->unCTL0.stcField.u1ENABLED = 0ul;    /* (1)Disable the LIN */
    }
    return ret;
}
```

Code Listing 6 Lin_Enable

```
/*******************************************************************************
 ** \brief Enable LIN channel.
*******************************************************************************/
cy_en_lin_status_t Lin_Enable(volatile stc_LIN_CH_t* pstcLin)       /* Check if parameter
values are valid */
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if (NULL == pstcLin)
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        pstcLin->unCTL0.stcField.u1ENABLED = 1ul;        /* (2)Enable the LIN */
    }
    return ret;
}
```

4 マスタの操作例

Code Listing 7 Lin_SetDataLength

```
/*******************************************************************************
 ** \brief Setup LIN response field data length
*******************************************************************************/
cy_en_lin_status_t Lin_SetDataLength(volatile stc_LIN_CH_t* pstcLin, uint8_t length)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if ((NULL == pstcLin) ||        /* Check if parameter values are valid */
        (length > LIN_DATA_LENGTH_MAX) ||
        (length < LIN_DATA_LENGTH_MIN))
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        /* (3)-1 Configure the data length of the response field */
        pstcLin->unCTL1.stcField.u3DATA_NR = length - 1ul;
    }
    return ret;
}
```

Code Listing 8 Lin_SetChecksumType

```
/*******************************************************************************
 ** \brief Setup LIN checksum type setting
*******************************************************************************/
cy_en_lin_status_t Lin_SetChecksumType(volatile stc_LIN_CH_t* pstcLin, en_lin_checksum_type_t
type)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if (NULL == pstcLin)              /* Check if parameter values are valid */
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        pstcLin->unCTL1.stcField.u1CHECKSUM_ENHANCED = type;  /* (3)-2 Configure the checksum
type */
    }
    return ret;
}
```

4 マスタの操作例

Code Listing 9 Lin_SetHeader

```
/*******************************************************************************
 ** \brief Setup LIN header for master tx header operation
*******************************************************************************/
cy_en_lin_status_t Lin_SetHeader(volatile stc_LIN_CH_t* pstcLin, uint8_t id)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    uint8_t TempPID;
    uint8_t Parity_P1, Parity_P0;
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    uint8_t TempPID;
    uint8_t Parity_P1, Parity_P0;       /* Check if parameter values are valid */
    if ((NULL == pstcLin) ||
        (LIN_ID_MAX < id))
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        /* Calculate the Parity bits P0 & P1 */
        Parity_P0 = ((id) ^ (id>>1ul) ^
                     (id>>2ul) ^ (id>>4ul)) & 0x01ul;
        Parity_P1 = (~((id>>1ul) ^ (id>>3ul) ^
                       (id>>4ul) ^ (id>>5ul))) & 0x01ul;
        /* Assign the Parity bits and the header values in to the TempPID */
        TempPID = id | ((uint8_t) Parity_P0<<6ul) | ((uint8_t) Parity_P1<<7ul);
        /* Write the TempID value in to the TX_HEADER register */
        /* (4)Configure the PID field */
        pstcLin->unPID_CHECKSUM.stcField.u8PID = TempPID;
    }
    return ret;
}
```

4 マスタの操作例

Code Listing 10 Lin_WriteData

```
/*******************************************************************************
 ** \brief Write response data.
*******************************************************************************/
cy_en_lin_status_t Lin_WriteData( volatile stc_LIN_CH_t* pstcLin, const uint8_t *au8Data,
uint8_t u8DataLength )
{
    cy_en_lin_status_t status = CY_LIN_SUCCESS;
    un_LIN_CH_DATA0_t data0 = { 0ul };
    un_LIN_CH_DATA1_t data1 = { 0ul };
    uint8_t u8Cnt;
    /* Check if NULL pointer */       /* Check if parameter values are valid */
    if( ( NULL == pstcLin ) ||
        ( NULL == au8Data ) )
    {
        status = CY_LIN_BAD_PARAM;
    }
    /* Check if data length is valid */       /* Check if parameter values are valid */
    else if( LIN_DATA_LENGTH_MAX < u8DataLength )
    {
        status = CY_LIN_BAD_PARAM;
    }
    /* Check if the bus is free */       /* Check if parameter values are valid */
    else if( 0ul == pstcLin->unSTATUS.stcField.u1TX_BUSY )
    {

        /* Write data in to the temp variables */
        for( u8Cnt = 0ul; u8Cnt < u8DataLength; u8Cnt++ )
        {
            if( 4ul > u8Cnt )
            {
                data0.au8Byte[u8Cnt] = au8Data[u8Cnt];
            }
            else
            {
                data1.au8Byte[u8Cnt - 4ul] = au8Data[u8Cnt];
            }
        }
        /* Write data to HW FIFO */
        /* (4)-1 Configure the data register (DATA 0) */
        pstcLin->unDATA0.u32Register = data0.u32Register;
        /* (4)-1 Configure the data register (DATA 1) */
        pstcLin->unDATA1.u32Register = data1.u32Register;
    }
    else
    {
        status = CY_LIN_BUSY;
        /* A requested operation could not be completed */
    }
    return status;
}
```

4 マスタの操作例

Code Listing 11 Lin_SetInterruptMask

```
/*******************************************************************************
 ** \brief Setup interrupt source to be accepted.
*******************************************************************************/
cy_en_lin_status_t Lin_SetInterruptMask(volatile stc_LIN_CH_t* pstcLin, uint32_t mask)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if (NULL == pstcLin)                              /* Check if parameter values are valid */
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
    pstcLin->unINTR_MASK.u32Register = mask;           /* (5)Enables the event interrupt */
    }
    return ret;
}
```

Code Listing 12 Lin_SetCmd

```
/*******************************************************************************
 ** \brief Setup LIN operation command
*******************************************************************************/
cy_en_lin_status_t Lin_SetCmd(volatile stc_LIN_CH_t* pstcLin, uint32_t command)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    un_LIN_CH_CMD_t cmdReg = pstcLin->unCMD;
    if (NULL == pstcLin)                          /* Check if parameter values are valid */
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else if (((command & (LIN_CH_CMD_TX_HEADER_Msk | LIN_CH_CMD_RX_HEADER_Msk))
                == (LIN_CH_CMD_TX_HEADER_Msk | LIN_CH_CMD_RX_HEADER_Msk))   ||
            (((command & LIN_CH_CMD_TX_WAKEUP_Msk) != 0ul) &&
             ((command & (LIN_CH_CMD_TX_HEADER_Msk |
                          LIN_CH_CMD_TX_RESPONSE_Msk |
                          LIN_CH_CMD_RX_HEADER_Msk |
                          LIN_CH_CMD_RX_RESPONSE_Msk)) != 0ul)))
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else if (((cmdReg.stcField.u1TX_HEADER != 0ul) && (command & LIN_CH_CMD_RX_HEADER_Msk) !=
0ul) ||
             ((cmdReg.stcField.u1RX_HEADER != 0ul) && (command & LIN_CH_CMD_TX_HEADER_Msk) !=
0ul) ||
             ((cmdReg.stcField.u1TX_WAKEUP != 0ul) &&
                           ((command & (LIN_CH_CMD_TX_HEADER_Msk |
                                        LIN_CH_CMD_TX_RESPONSE_Msk |
                                        LIN_CH_CMD_RX_HEADER_Msk |
                                        LIN_CH_CMD_RX_RESPONSE_Msk)) != 0ul)))
    {
        ret = CY_LIN_BUSY;
    }
    else
    {
    pstcLin->unCMD.u32Register = command;         /* (7)Configure the command sequence */
    }
    return ret;
}
```

4.3 LIN マスタ割込み処理の例

スケジューラによって設定された割込みが発生したとき、LIN マスタ IRQ ハンドラは起動します。図 7 に、LIN マスタ IRQ ハンドラがどのように動作するかの例を示します。

4 マスタの操作例

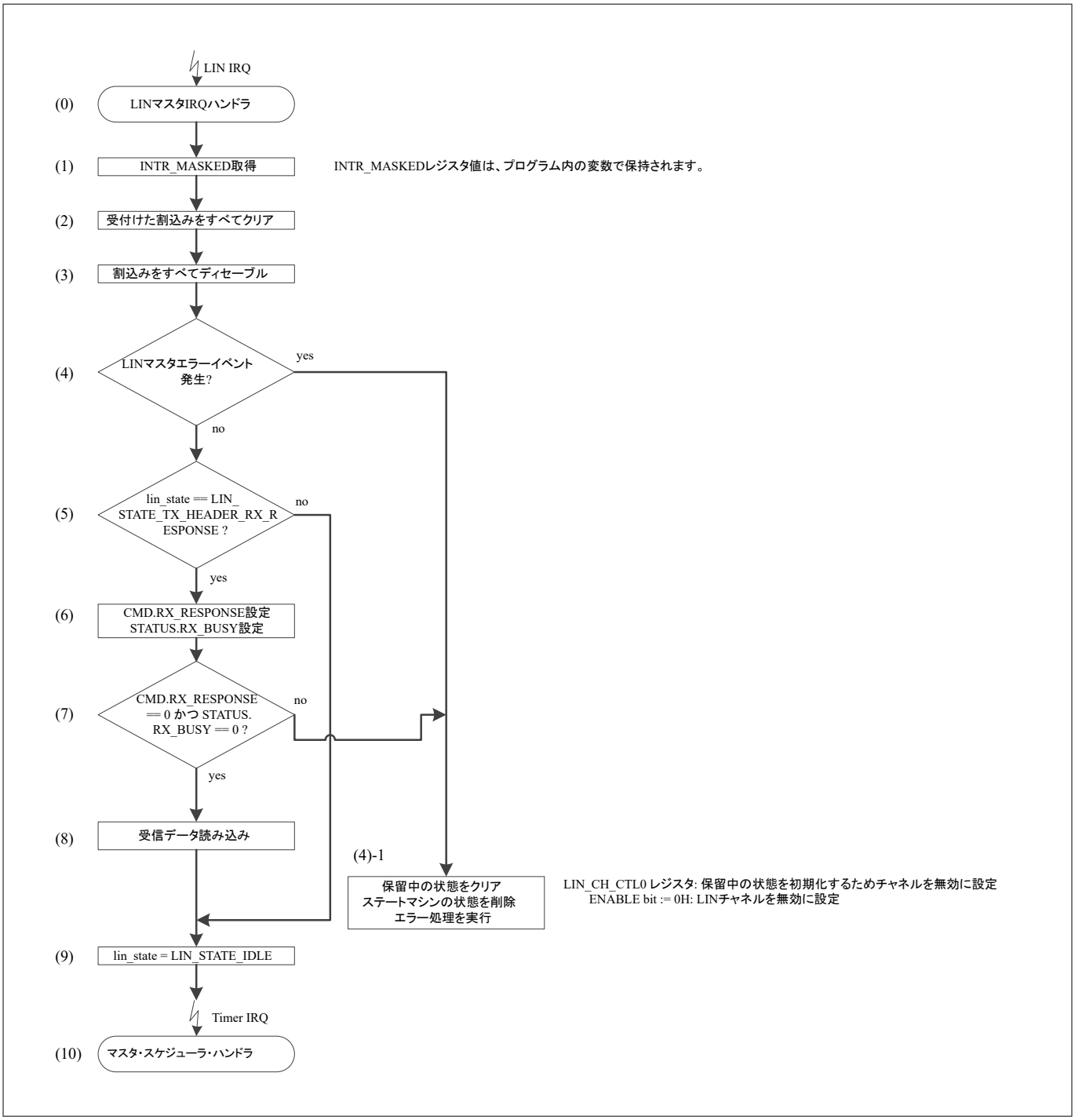

図 7　　LIN マスタ IRQ ハンドラの例

以下は、LIN マスタ IRQ ハンドラのためのアプリケーションソフトウェア操作です。

(0) LIN IRQ は、LIN マスタ IRQ ハンドラを起動させます。

(1) `LIN_CH_MASKED` レジスタから割込み情報を取得してください。

(2) 受け入れたすべての割込みをクリアしてください。

(3) 割込み処理の間、ほかの割込み発生を防ぐため、すべての割込みを無効にしてください。

(4) エラーが発生しているかを確認し、発生していたら (4)-1 に進んでください。

(4)-1 LIN_CH_CTL0.ENABLE が“0”に設定することで保留中の状態の状態をクリアし、LIN ブロック内のハードウェア内部ステートマシンとソフトウェアステートマシンの状態をクリアしてください。その後、エラー操作を実行してください。

(5) 通信エラーが検出されなければ、図 6 のスケジュールハンドラ (6) で決定されるソフトウェアステートマシンの状態 (lin_state) を確認してください。

- 状態が、LIN_STATE_TX_HEADER_RX_RESPONSE でない場合
- 、(9)に進んでください。

(6) 状態が、LIN_STATE_TX_HEDER_RX_RESPONSE であれば、LIN_CH_CMD.RX_RESPONSE と LIN_CH_STATUS.RX_BUSY の状態を取得してください。

(7) LIN_CH_CMD.RX_RESPONSE と LIN_CH_STATUS.RX_BUSY のビット領域を確認してください。

- ハードウェアは、コマンドシーケンスが成功で完了すると LIN_CH_CMD.RX_RESPONSE を"0"に設定し (エラーが検出されると"0"に設定しない)、前のコマンドシーケンスが成功で完了、またはエラーが検出された場合は、LIN_CH_STATUS.RX_BUSY は“0”に設定します。したがって、両方のビットが”0”に設定されているとき、受信は正しく完了します。
- LIN_CH_CMD.RX_RESPONSE または LIN_CH_STATUS.RX_BUSY が “1” ならば、受信は正しく完了していません。この場合、(4)-1 に進んでください。

(8) DATA0 と DATA1 レジスタから受信データを読んでください。

(9) 状態を LIN_STATE_IDLE に設定してください。

(10) LIN マスタ IRQ ハンドラを終了し、次のスケジューラ起動を待ってください。

4.3.1 ユースケース

ここでは、LIN マスタハンドラが割込み要因を決定し、次に割込み要因をクリアして、現在の状態の処理を実行する例について説明します。

- システム割込みソース: LINCH0 (IDX: 69)
- CPU 割込みマッピング: IRQ3
- CPU 割込み優先度: 3
- 通信操作: 表 1 とセクション 4 を参照してください。

4.3.2 設定と例

表 7 に、SDL の LIN マスタ割込みハンドラの設定部のパラメータを示します。

表 7 LIN マスタ割込みハンドラパラメータのリスト

パラメータ	説明	設定値
割込み用		
irq_cfg.sysIntSrc	システム割込みインデックス番号	CY_LINCH0_IRQN
irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
irq_cfg.isEnabled	CPU 割込み許可	true (0x1)
LIN 用		
CY_LINCH0_TYPE	使用する LIN チャネル番号	LIN0 の channel 0

Code Listing 13 に、設定部での LIN 割込みのサンプルプログラムを示します。

4 マスタの操作例

Code Listing 13 CYT2 シリーズ: 設定部での LIN 割込み例 (マスタ)

```
int main(void)
{

    :
    __enable_irq(); /* Enable global interrupts. */
    :
    /* Register LIN interrupt handler and enable interrupt */
    {
        cy_stc_sysint_irq_t irq_cfg;
        irq_cfg = (cy_stc_sysint_irq_t){
            .sysIntSrc  = CY_LINCH0_IRQN,
            .intIdx     = CPUIntIdx3_IRQn,
            .isEnabled  = true,
        };
        Cy_SysInt_InitIRQ(&irq_cfg);     /* Set the parameters to interrupt structure*1 */
        Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, LIN0_IntHandler);     /* Set the system
interrupt handler*1 */
        NVIC_SetPriority(CPUIntIdx3_IRQn, 0ul);     /* Set priority*1*/
        NVIC_EnableIRQ(CPUIntIdx3_IRQn);      /* Interrupt Enable*1 */
    }
    :
/* LIN0 IRQ Handler */
static void LIN0_IntHandler(void)
{
    uint32_t maskStatus;
    cy_en_lin_status_t apiResponse;
    /* (1)Acquire interrupt information Code Listing 14 */
    Lin_GetInterruptMaskedStatus(CY_LINCH0_TYPE, &maskStatus);
    /* (2)Clear all accepted interrupt Code Listing 15 */
    Lin_ClearInterrupt(CY_LINCH0_TYPE, maskStatus);  /* Clear all accepted interrupt */


/* (3)Disable all interrupt to prevent occurrence of different interrupt during interrupt
handling Code Listing 11 */
    Lin_SetInterruptMask(CY_LINCH0_TYPE, 0uL);       /* Disable all interrupt */




     /* (4)Check if an error occurred */
    if ((maskStatus & LIN_INTR_ALL_ERROR_MASK_MASTER) != 0ul)
    {
        /* Wait for next tick. */
        lin_state = LIN_STATE_IDLE;
        /* Disable the channel to reset LIN status */
        /* (4)-1 Clear the currently pending state  */
        Lin_Disable(CY_LINCH0_TYPE);       /* (4)-1 Clear the currently pending state Code
Listing 5 */
        /* Re-enable the channel */
        Lin_Enable(CY_LINCH0_TYPE);
    }
```

```
    else
    {
        switch(lin_state)
        {
        /* (5)Current state is not LIN_STATE_TX_HEADER_RX_RESPONSE */
        case LIN_STATE_TX_HEADER:
            /* Tx header complete with no error */
            break;
            /* (5)Current state is not LIN_STATE_TX_HEADER_RX_RESPONSE */
        case LIN_STATE_TX_HEADER_TX_RESPONSE:
            /* Tx response complete with no error */
            break;
            /* (6)Current state is LIN_STATE_TX_HEDER_RX_RESPONSE */
        case LIN_STATE_TX_HEADER_RX_RESPONSE:

            /* (7)Check the bit fields. */
            /* Tx header and rx response complete with no error */
            while(1)
            {
            /* (8)Read the received dataCode Listing 16. */
                apiResponse = Lin_ReadData(CY_LINCH0_TYPE, msgContext[scheduleIdx].dataBuffer);
                if(apiResponse == CY_LIN_SUCCESS)
                {
                    break;
                }
            }
            /* For testing
             * Set rx data to tx data. Rx ID + 1 => Tx ID
             */
            memcpy(msgContext[scheduleIdx + 1ul].dataBuffer,
msgContext[scheduleIdx].dataBuffer, LIN_DATA_LENGTH_MAX);
            break;
        default:
            break;
        }
        lin_state = LIN_STATE_IDLE;        /* (9) Set the state to LIN_STATE_IDLE. */
    }
}
```

*1 詳細は、Architecture TRM の CPU interrupt handing セクションを参照してください。

Code Listing 14 から Code Listing 16 に、ドライバ部での LIN 割込みのプログラム例を示します。

4 マスタの操作例

Code Listing 14 Lin_GetInterruptMaskedStatus

```
/*******************************************************************************
 ** \brief Return interrupt masked status.
*******************************************************************************/
cy_en_lin_status_t Lin_GetInterruptMaskedStatus(volatile stc_LIN_CH_t* pstcLin, uint32_t
*status)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if ((NULL == pstcLin) ||        /* Check if parameter values are valid */
        (NULL == status))
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        *status = pstcLin->unINTR_MASKED.u32Register;     /* (1)Acquire interrupt information */
    }
    return ret;
}
```

Code Listing 15 Lin_ClearInterrupt

```
/*******************************************************************************
 ** \brief Clear interrupt status.
*******************************************************************************/
cy_en_lin_status_t Lin_ClearInterrupt(volatile stc_LIN_CH_t* pstcLin, uint32_t mask)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if (NULL == pstcLin)        /* Check if parameter values are valid */
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        pstcLin->unINTR.u32Register = mask;    /* (2)Clear interrupt status */
    }
    return ret;
}
```

4 マスタの操作例

Code Listing 16 Lin_ReadData

```
/*******************************************************************************
 ** \brief Read response data.
*******************************************************************************/
cy_en_lin_status_t Lin_ReadData( volatile stc_LIN_CH_t* pstcLin, uint8_t  *u8Data )
{
    cy_en_lin_status_t status = CY_LIN_SUCCESS;
    uint8_t u8Cnt;
    uint8_t u8Length;
    /* Check if pointers are valid */
    if( ( NULL == pstcLin )           ||        /* Check if parameter values are valid */
        ( NULL == u8Data ))
    {
        status = CY_LIN_BAD_PARAM;
    }
    /* Check if the response is received successfully */
    else if( ( 0ul == pstcLin->unCMD.stcField.u1RX_RESPONSE ) &&
             ( 0ul == pstcLin->unSTATUS.stcField.u1RX_BUSY ) )
    {
        u8Length = pstcLin->unCTL1.stcField.u3DATA_NR + 1ul;
        /* Copy the data in to u8Data array */
        un_LIN_CH_DATA0_t data0 = pstcLin->unDATA0;
        un_LIN_CH_DATA1_t data1 = pstcLin->unDATA1;        /* (8)Read response data */
        for ( u8Cnt = 0ul; u8Cnt < u8Length; u8Cnt++ )
        {
            if( 4ul > u8Cnt )
            {
                u8Data[u8Cnt] = data0.au8Byte[u8Cnt];
            }
            else
            {
                u8Data[u8Cnt] = data1.au8Byte[u8Cnt - 4ul];
            }
        }
    }
    else
    {
        status = CY_LIN_BUSY;
    }
    return status;
}
```

5 スレーブの操作例

LIN スレーブの実装例を示します。LIN スレーブは、マスタのように動作する LIN プロトコルアナライザから、スケジュール表に従い情報を送受信します。LIN スレーブ IRQ ハンドラは、テーブルを持ちます。メッセージフレーム ID 処理表の例として表 8 を参照してください。この情報は、図 11 で使われます。LIN スレーブは、LIN マスタからヘッダを受信します。表 8 の様にヘッダを受信すると、受け取った PID と一致する応答領域が送受信されます。これらの異なるメッセージタイプに対応するために、Architecture TRM の LIN Slave Command Sequence 表にあるように、LIN スレーブ動作の取扱いは、コマンドシーケンスによって処理されます。

表 8 LIN スレーブのメッセージフレーム ID 処理表

ID	メッセージタイプ	データ長	チェックサムタイプ
0x01	マスタ応答	8	エンハンス
0x02	スレーブ応答	8	エンハンス
0x10	マスタ応答	1	エンハンス
0x11	スレーブ応答	1	エンハンス

この例では、ソフトウェアはステートマシンを用い、コマンドシーケンスの設定を管理します。図 8 に LIN スレーブのステートマシンを示します。T0 から T6 の矢印は、状態遷移のトリガです。

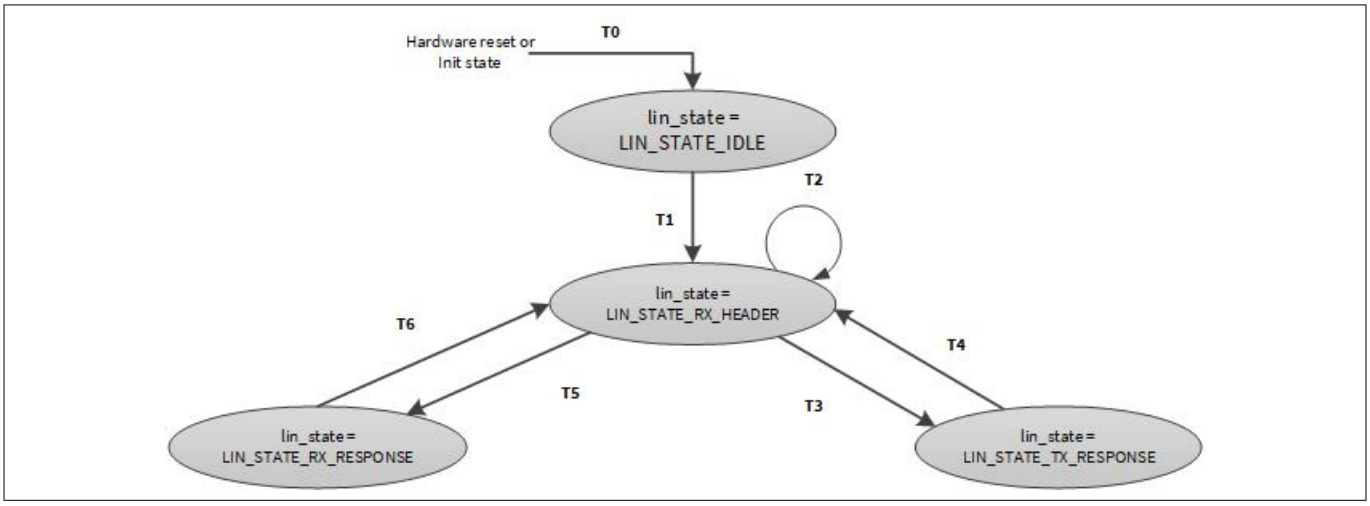

図 8 LIN スレーブステートマシン

LIN スレーブステートマシンは以下の 4 状態を持ちます。

1. `LIN_STATE_IDLE`: これは、初期化後のデフォルト状態です。スレーブは、LIN バスでいかなる情報も送受信しません。
2. `LIN_STATE_RX_HEADER`: これは、スレーブで LIN break 検出の準備ができている状態です。スレーブは、ヘッダ受信の完了を待ちます。
3. `LIN_STATE_RX_RESPONSE`: これは、メッセージタイプがマスタ応答の状態です。スレーブは、マスタからの応答を待ちます。
4. `LIN_STATE_TX_RESPONSE`: これは、メッセージタイプがスレーブ応答の状態です。スレーブは、マスタへの応答を送信します。メッセージタイプが Slave-to-Slave の場合、スレーブは他のスレーブに応答を送信します。

ソフトウェアは、表 8 のメッセージタイプによって状態を判断し、その時の状態によってコマンドシーケンスを設定します。表 9 に、メッセージタイプ, 状態, コマンドシーケンスの関係を示します。

表 9 LIN スレーブのメッセージタイプ, 状態, およびコマンドシーケンスの関係

メッセージタイプ	状態	TX_HEADER	RX_HEADER	TX_RESPONSE	RX_RESPONSE
スレーブ応答	LIN_STATE_TX_RESPONSE	0	1	1	1
マスタ応答	LIN_STATE_RX_RESPONSE	0	1	0	1

以下は、これらのプロセスを実行するための初期化と割込み制御の例です。

5.1 LIN スレーブの初期化

図 9 に、LIN スレーブ初期化のフロー例を示します。

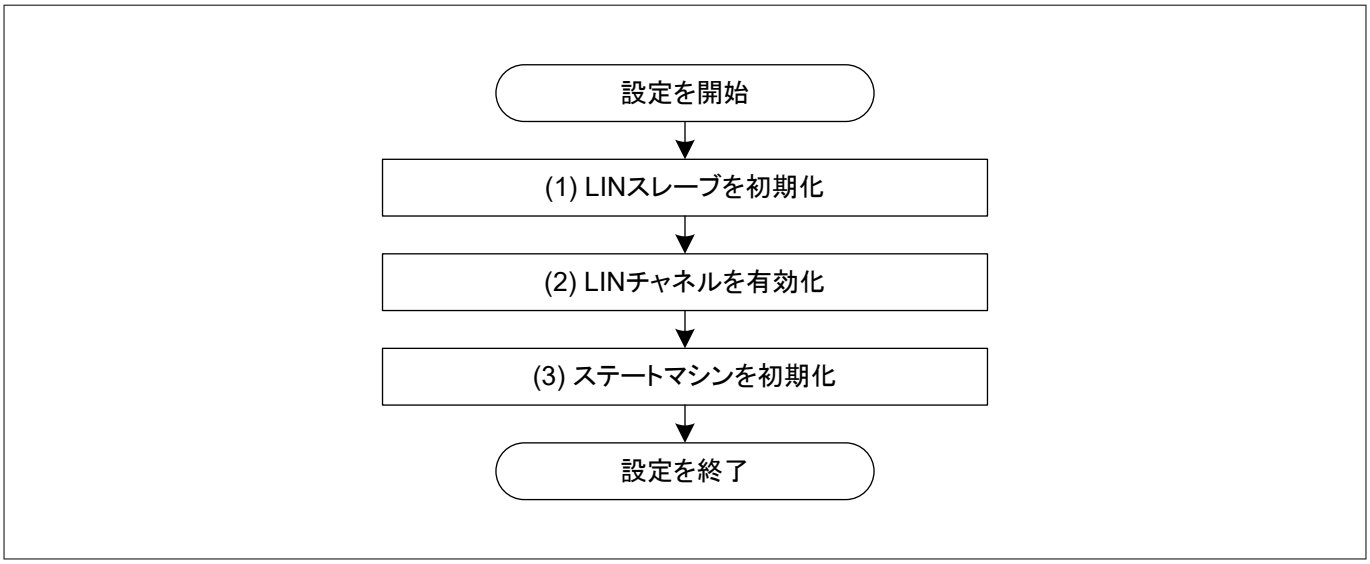

図 9 LIN スレーブ初期化フロー例

1. LIN スレーブを初期化してください。
2. LIN チャネルを有効にしてください。
   ポート設定完了後、ソフトウェアにより外部 LIN トランシーバを有効にします。設定手順 (1) において LIN_CH_CTL0.AUTO_EN が「0」に設定するため、この例では、外部 LIN トランシーバを制御しません。この場合、ソフトウェアはレジスタビット TX_RX_STATUS.EN_OUT を介して EN ピンを制御します。設定されたチャネルの LIN_EN_OUT ピンが MCU で使用できない場合は、トランシーバの EN ピンも通常の GPIO 出力で制御できます。
3. ソフトウェアステートマシンを初期化してください。
   現在の状態を lin_state = LIN_STATE_IDLE に設定してください。

クロック設定、ポート設定、および割込みコントローラー設定の詳細は、Architecture TRM と Registers TRM を参照してください。

5.1.1 ユースケース

ここでは、以下のパラメータを使用した LIN 初期化の使用例について説明します。

- マスタ/スレーブノード: スレーブノード
- LIN インスタンス: LIN0_CH0
- ボーレート: 19231 Hz

5.1.2 設定と例

表 10 に、LIN マスタ初期化用の SDL の設定部のパラメータを示します。

表 10 LIN マスタ初期化パラメータリスト

パラメータ	説明	設定値
CLK 用		
CY_LINCH0_PCLK	周辺クロック番号	PCLK_LIN0_CLOCK_CH_EN0
LIN 用		
lin_config.bMasterMode	マスタまたはスレーブモード	false (スレーブモード)
lin_config.bLinTransceiverAutoEnable	LIN トランシーバ自動有効	true (有効)
lin_config.u8BreakFieldLength	ビット時間のブレイク/ウェイクアップ長 (1 を減算)	11ul (11-1 = 10 bit)
lin_config.enBreakDelimiterLength	ブレイクデリミタ長	LinBreakDelimiterLength1bits (1 bit)
lin_config.enStopBit	ストップビット時間	LinOneStopBit (1 bit)
lin_config.bFilterEnable	受信フィルタ	true
CY_LINCH0_TYPE	使用する LIN チャネル番号	LIN0 の channel 0

Code Listing 17 に、設定部での LIN スレーブを初期化するためのサンプルプログラムを示します。

Code Listing 17 CYT2 シリーズ: 設定部での LIN 初期化例 (スレーブ)

```
/* Configure LIN Slave parameters */
static const stc_lin_config_t lin_config =
{
    .bMasterMode = false,
    .bLinTransceiverAutoEnable = true,
    .u8BreakFieldLength = 11ul,
    .enBreakDelimiterLength = LinBreakDelimiterLength1bits,
    .enStopBit = LinOneStopBit,
    .bFilterEnable = true
};

    :
int main(void)
{
    :
    /* LIN baudrate setting */
    /* Note:
     * LIN IP does oversampling and oversampling count is fixed 16.
     * Therefore LIN baudrate = LIN input clock / 16.
     */
    /* Configure LIN Slave parameters */
    Cy_SysClk_PeriphAssignDivider(CY_LINCH0_PCLK, CY_SYSCLK_DIV_16_BIT, 0u);
    /* Configure LIN Slave parameters */
    Cy_SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, 0ul, 259ul); // 80 MHz / 260 / 16
(oversampling) = 19231 Hz
    /* Configure LIN Slave parameters */
    /* Configure the Baud Rate Clock*1 */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, 0ul);

    /* Initialize LIN */
    /* (1)Initialize LIN Master based on above structure (See Code Listing 3) */
    /* (2)Enable LIN_CH0 (See Code Listing 3) */
    Lin_Init(CY_LINCH_TYPE, &lin_config);
    lin_state = LIN_STATE_IDLE;

    /* LIN operation */
    /* (3)Initialize the state machine */
    lin_state = LIN_STATE_RX_HEADER;
    :
}
```

*1: 詳細は、Architecture TRM の Clocking System セクションを参照してください。

5.2 LIN スレーブ割込み処理の例

マスタからのヘッダによって、割込みが設定された時、LIN スレーブ IRQ ハンドラは起動します。

図 10 に LIN スレーブ IRQ ハンドラがどのように動作するかの例を示します。このフローは Code Listing 18 で使用されます。

5 スレーブの操作例

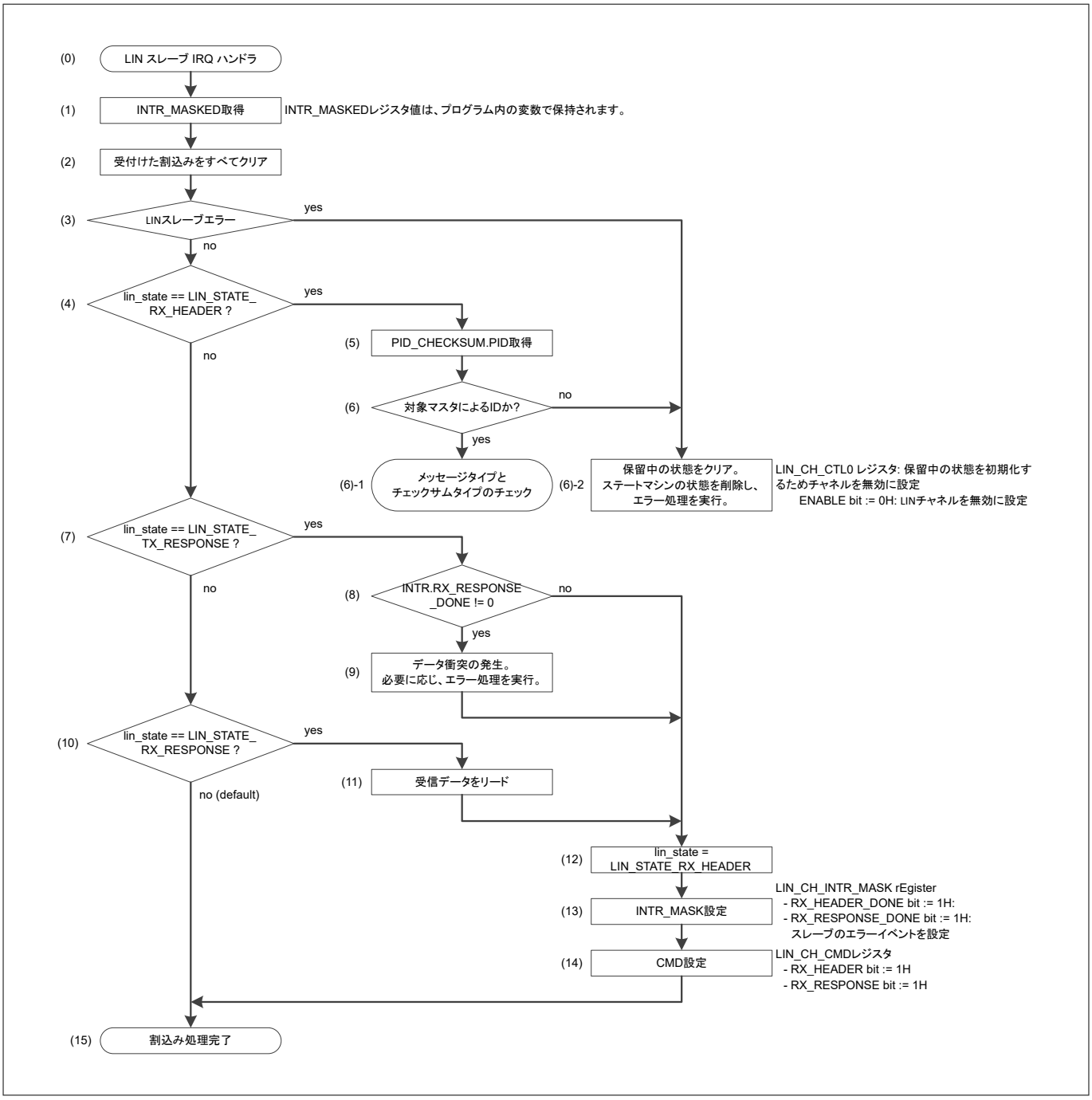

図 10　　LIN スレーブ IRQ ハンドラの例

以下に LIN スレーブ IRQ ハンドラのアプリケーションソフトウェア操作を示します。

(0) LIN IRQ で LIN スレーブ IRQ ハンドラを起動させます。

(1) `LIN_CH_MASKED` レジスタから割込み情報を取得してください。

(2) 割込み状態を初期化するため、すべての割込みフラグをクリアしてください。

(3) 通信エラーの発生を確認し、エラーが検出されたら (6)-2 に進んでください。

(4) 通信エラーがなければ、ステートマシンの状態 (lin_state) を確認してください。

- ステートマシンの状態が、`LIN_STATE_RX_HEADER` であれば、(5) に進んでください。
- ステートマシンの状態が、`LIN_STATE_RX_HEADER` でなければ、(7) に進んでください。

(5) `LIN_CH_PID_CHECKSUM.PID` から受信した PID の値を取得してください。

5 スレーブの操作例

(6) 現在の ID をチェックしてください。現在の ID が表 8 になければ、(6)-2 に進んでください。

- ID が表 8 にあったら、(6)-1 に進んでください。
- (6)-1 図 11 の (0) に進んでください。
- (6)-2 LIN_CH_CTL0.ENABLE を“0”に設定することにより、保留中の状態をクリアし、ハードウェア内部ステートマシンとソフトウェアステートマシンの状態を削除してください。その後、システムに応じて適切なフェイル操作を行ってください。

(7) ステートマシンの状態 (lin_state) を確認してください。

- 状態が、LIN_STATE_TX_RESPONSE であれば、(8) に進んでください。そうでなければ、(10) に進んでください。

(8) INTR.RX_RESPONSE_DONE の状態をチェックしてください。

フレーム応答 (データ領域とチェックサム領域) が受信された (CMD.RX_RESPONSE が完了した) 時、LIN_CH_INTR.RX_RESPONSE_DONE は、ハードウェアにより“1”に設定されます。

LIN_CH_INTR.RX_RESPONSE_DONE が“0”ならデータ衝突がありません。この場合、(12) に進んでください。

LIN_CH_INTR.RX_RESPONSE_DONE が“1”ならデータ衝突が発生しています。この場合、(9) に進んでください。

(9) システムごとのデータ衝突時の操作を行い (12) に進んでください。

(10) ステートマシンの状態 (lin_state) を確認してください。

- 状態が、LIN_STATE_RX_RESPONSE であれば、(11) に進んでください。そうでなければ、(15) に進んでください。

(11) DATA0 と DATA1 から受信データを読んでください。

(12) 状態を、LIN_STATE_RX_HEADER に設定してください。

(13) LIN_CH_INTR_MASK レジスタにより、割込みイベントを有効にしてください。

- RX_HEADER_DONE を“1”に設定してください
- RX_RESPONSE_DONE を“1”に設定してください。
- エラー検出ビットを“1”に設定してください。
- システムに従い、必要なエラー検出ビットの設定が必要です。

(14) コマンドシーケンスを設定してください。

- LIN_CH_CMD.RX_HEADER を“1”に設定してください。
- LIN_CH_CMD.RX_RESPONSE を“1”に設定してください。

(15) LIN スレーブ IRQ ハンドラから戻り、表 2 のように設定された、LIN の割込みの発生を待ってください。

図 11 に、メッセージタイプとチェックサムタイプの動作が、どのように実行されるかを示します。このフローは、図 10 の (6)-1 と Code Listing 20 からジャンプする場合に使用されます。

5 スレーブの操作例

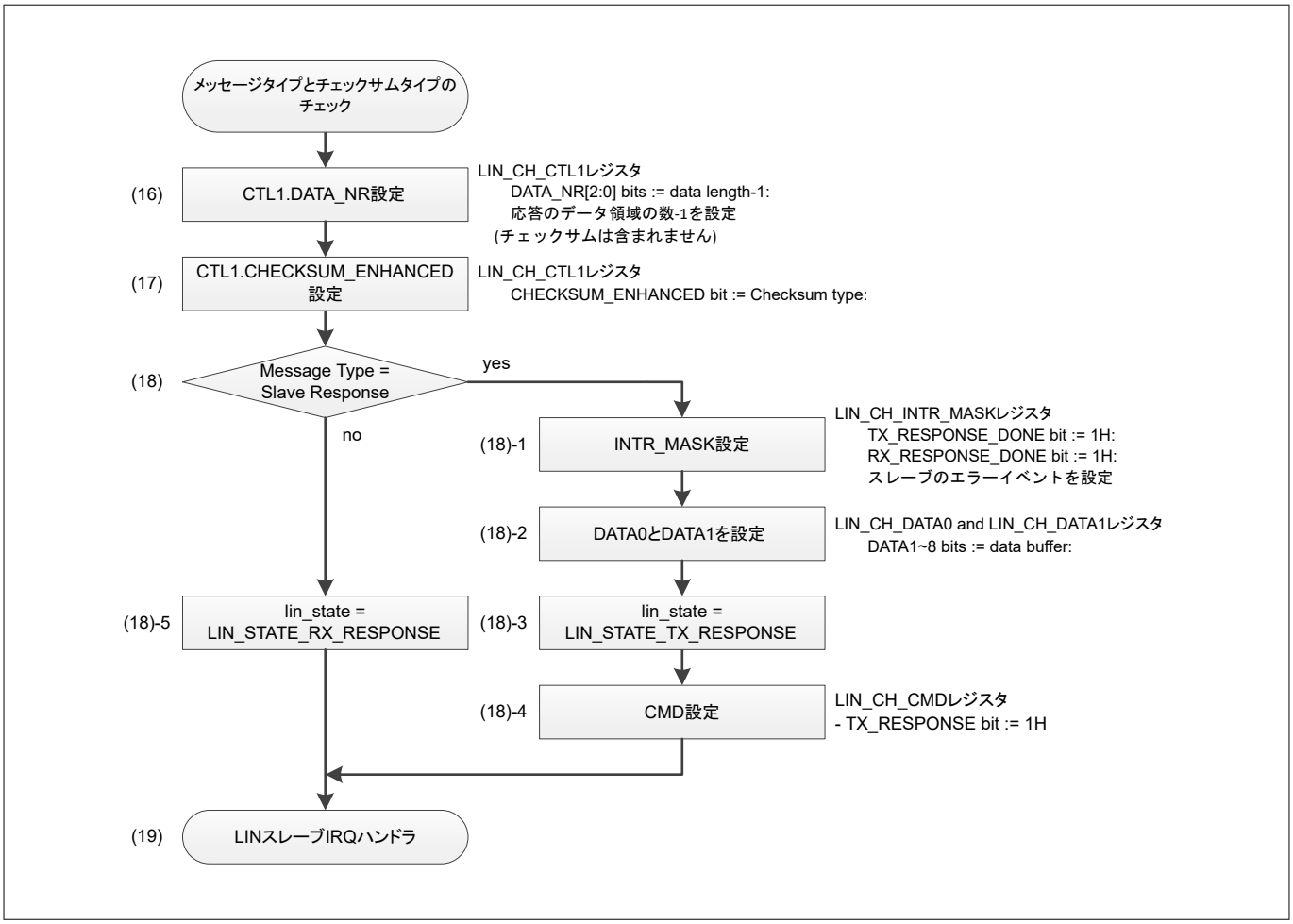

図 11　　メッセージタイプとチェックサムタイプの LIN スレーブチェックの例

メッセージタイプとチェックサムタイプのチェックのフローを、下記に示します。

(16) 表 8 に従って、応答のデータ長を設定してください。

(17) 表 8 に従って、チェックサムタイプを設定してください。

(18) 表 8 に従って、メッセージタイプを確認してください。

- メッセージタイプが `LIN_TX_RESPONSE` の場合
- (18)-1. LIN_CH_INTR_MASK レジスタによるイベント割り込みを有効にします。
- システムに応じて、必要なエラー検出ビットを設定する必要があります。
  - `RX_RESPONSE_DONE` を“1”に設定してください。
  - `TX_RESPONSE_DONE` を“1”に設定してください。
  - エラー検出ビットを“1”に設定してください。
- (18)-2. LIN スレーブは、必要なデータ長の応答データをデータレジスタ (DATA 0/1) に書き込みます。
- (18)-3. lin_state を `LIN_STATE_TX_RESPONSE` に設定してください。
- (18)-4. 状態によって、コマンドシーケンスを設定してください。`LIN_CH_CMD.TX_RESPONSE` を“1”に設定してください。
  - 現在のメッセージタイプが LIN_TX_RESPONSE でない場合
- (18)-5. lin_state を `LIN_STATE_RX_RESPONSE` に設定してください。

(19) LIN スレーブ IRQ ハンドラから戻り、次の割込みイベントを待ってください。

5.2.1 ユースケース

ここでは、LIN スレーブハンドラが割込み要因を決定し、次に割込み要因をクリアして、現在の状態の処理を実行する例について説明します。

- システム割込みソース: LINCH0 (IDX: 69)
- CPU 割込みマッピング: IRQ3
- CPU 割込み優先度: 3
- 通信操作: セクション 5 を参照してください。

5.2.2 設定と例

表 11 に、SDL の LIN スレーブ割込みハンドラの設定部のパラメータを示します。

表 11 LIN スレーブ割込みハンドラパラメータのリスト

パラメータ	説明	設定値
割込み用		
lin_irq_cfg.sysIntSrc	システム割込みインデックス番号	CY_LINCH0_IRQN
lin_irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
lin_irq_cfg.isEnabled	CPU 割込み許可	true (0x1)
LIN 用		
CY_LINCH0_TYPE	使用する LIN チャネル番号	LIN0 の channel 0

Code Listing 18 に、設定部での LIN 割込みのサンプルプログラムを示します。

5 スレーブの操作例

Code Listing 18 CYT2 シリーズ: 設定部での LIN 割込み例 (スレーブ)

```
#define CY_LINCH_IRQN           CY_LINCH0_IRQN
    :
static const cy_stc_sysint_irq_t lin_irq_cfg =  /* Configure interrupt structure parameters*1 */
{
    .sysIntSrc  = CY_LINCH_IRQN,
    .intIdx     = CPUIntIdx3_IRQn,
    .isEnabled  = true,
};
    :
int main(void)
{
    :
    __enable_irq(); /* Enable global interrupts. */   /* Enable global interrupt*1 */
    :
    Cy_SysInt_InitIRQ(&lin_irq_cfg);   /* Set the parameters to interrupt structure*1 */
    Cy_SysInt_SetSystemIrqVector(lin_irq_cfg.sysIntSrc, LIN0_IntHandler);   /* Set the system
interrupt handler*1 */
    NVIC_SetPriority(CPUIntIdx3_IRQn, 3ul);   /* Set priority*1 */
    NVIC_EnableIRQ(CPUIntIdx3_IRQn);    /* Interrupt Enable*1 */
    :

/* LIN0 IRQ Handler */
void LIN0_IntHandler(void)
{
    uint32_t maskStatus;
    /* (1)Acquire interrupt information (See Code Listing 14) */
    Lin_GetInterruptMaskedStatus(CY_LINCH0_TYPE, &maskStatus);
    /* (2)Clear all interrupt flags (See Code Listing 15) */
    Lin_ClearInterrupt(CY_LINCH0_TYPE, maskStatus); /* Clear all accepted interrupt */
    cy_en_lin_status_t apiResponse;

    /* (3)Check if an error occurred */
    if ((maskStatus & CY_LIN_INTR_ALL_ERROR_MASK_SLAVE) != 0ul)
    {
        /* There are some error */
        /* Handle error if needed. */
        /* Disable the channel to reset LIN status */
        Lin_Disable(CY_LINCH0_TYPE);
        /* Re-enable the channel */
        Lin_Enable(CY_LINCH0_TYPE);
        /* Re enable header RX */
        lin_state = LIN_STATE_RX_HEADER;
        Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_RX_HEADER_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);
        Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_RX_HEADER_RX_RESPONSE);
    }
    else
    {
        bool acceptedId = false;
        uint8_t id, parity;
        switch(lin_state)
```

```
        {
        case LIN_STATE_RX_HEADER:    /* (4)Current state is LIN_STATE_RX_HEADER */
            /* Rx header complete with no error */
            Lin_GetHeader(CY_LINCH0_TYPE, &id, &parity);    /* (5)Get the received PID value
(See Code Listing 19) */
            /* Analyze ID */    /* (6)Check the current ID */
            for (uint8_t I = 0ul; I < (sizeof(msgContext) / sizeof(msgContext[0ul])); i++)
            {
                if (id == msgContext[i].id)
                {
                    currentMsgIdx = I;
                    acceptedId = true;
                    break;
                }
            }
            if (acceptedId)
            {
                /* Setup checksum type and data length */
                Lin_SetDataLength(CY_LINCH0_TYPE, msgContext[currentMsgIdx].dataLength);
                Lin_SetChecksumType(CY_LINCH0_TYPE, msgContext[currentMsgIdx].checksumType);
                if (msgContext[currentMsgIdx].responseDirection == LIN_TX_RESPONSE)
                {
                    Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_TX_RESPONSE_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);
                    Lin_WriteData(CY_LINCH0_TYPE, msgContext[currentMsgIdx].dataBuffer,
msgContext[currentMsgIdx].dataLength);
                    lin_state = LIN_STATE_TX_RESPONSE;
                    Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_TX_RESPONSE);
                }
                else
                {
                    lin_state = LIN_STATE_RX_RESPONSE;
                }
            }
            else
            {
                /* Message to be ignored */
                /* Disable the channel to reset LIN status */
                Lin_Disable(CY_LINCH0_TYPE);      /* (6)-2 Clear the currently pending state
(See Code Listing 5) */
                /* Re-enable the channel */
                Lin_Enable(CY_LINCH0_TYPE);
                /* Re enable header RX */
                lin_state = LIN_STATE_RX_HEADER;
                Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_RX_HEADER_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);
                Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_RX_HEADER_RX_RESPONSE);
            }
            break;
        case LIN_STATE_TX_RESPONSE:           /* (7)Current state is LIN_STATE_TX_RESPONSE */
            /* Tx response complete with no error */
            /* Check if RX_DONE interrupt occurs or not */
            /* If RX_DONE interrupt occurs, response collision occurs */
```

```
            if ((maskStatus & CY_LIN_INTR_RX_RESPONSE_DONE) != 0ul)    /* (8)Check the
condition of INTR.RX_RESPONSE_DONE */
            {
                /* Data collision occurs */
                /* Do error handling if needed */
            }
            {
                /* Data collision occurs */            /* (9)Run the data collision operation */
                /* Do error handling if needed */
            }
            /* Re enable header RX */
            lin_state = LIN_STATE_RX_HEADER;            /* (12)Configure the state to
LIN_STATE_RX_HEADER */
            /* (13)Enable event interrupt for RX_HEADER (See Code Listing 11) */
            Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_RX_HEADER_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);


            /* (14)Configure the Command Sequence for RX_HEADER (See Code Listing 12) */
            Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_RX_HEADER_RX_RESPONSE);
            break;

        case LIN_STATE_RX_RESPONSE:            /* (10)Current state is LIN_STATE_RX_RESPONSE */
            /* Rx response complete with no error */
            while(1)
            {
            /* (11)Read the reception data from DATA0 and DATA1 (See Code Listing 16) */
                apiResponse = Lin_ReadData(CY_LINCH0_TYPE,
msgContext[currentMsgIdx].dataBuffer);
                if(apiResponse == CY_LIN_SUCCESS)
                {
                    break;
                }
            }
            /* For testing
             * Set rx data to tx data. Rx ID - 1 => Tx ID
             */
            memcpy(msgContext[currentMsgIdx - 1].dataBuffer,
msgContext[currentMsgIdx].dataBuffer, CY_LIN_DATA_LENGTH_MAX);
            /* Re enable header RX */
            /* (12)Configure the state to LIN_STATE_RX_HEADER.*/
            lin_state = LIN_STATE_RX_HEADER;
            /* (13)Enable event interrupt for RX_HEADER (See Code Listing 11).*/
            Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_RX_HEADER_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);

            /* (14)Configure the Command Sequence for RX_HEADER (See Code Listing 12).*/
            Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_RX_HEADER_RX_RESPONSE);
            break;
        default:
            break;
        }
```

```
    }
}
```

*1: 詳細は、Architecture TRM の CPU interrupt handing セクションを参照してください。

Code Listing 19 に、ドライバ部での LIN 割込みのプログラム例を示します。

Code Listing 19 Lin_GetHeader

```
/*******************************************************************************
 ** \brief Return received LIN header
*******************************************************************************/
cy_en_lin_status_t Lin_GetHeader(volatile stc_LIN_CH_t* pstcLin, uint8_t *id, uint8_t *parity)
{
    cy_en_lin_status_t ret = CY_LIN_SUCCESS;
    if ((NULL == pstcLin) ||        /* Check if parameter values are valid */
        (NULL == id)      ||
        (NULL == parity))
    {
        ret = CY_LIN_BAD_PARAM;
    }
    else
    {
        /* Store received ID and parity bits */
        uint8_t temp = pstcLin->unPID_CHECKSUM.stcField.u8PID;        /* (5)Return received LIN
header */
        *parity = (temp >> 6ul);
        *id = (temp & LIN_ID_MAX);
    }
    return ret;
}
```

Code Listing 20 に、設定部でのメッセージタイプとチェックサムタイプの操作方法のサンプルプログラムを示します。

5 スレーブの操作例

Code Listing 20 CYT2 シリーズ: メッセージタイプとチェックサムタイプの操作方法例

```
/* LIN0 IRQ Handler */
void LIN0_IntHandler(void)
{

    :
        /* Setup checksum type and data length */
        /* (16)Configure the data length of the response (See Code Listing 7) */
        Lin_SetDataLength(CY_LINCH0_TYPE, msgContext[currentMsgIdx].dataLength);
        /* (17)Configure the checksum type (See Code Listing 8) */
        Lin_SetChecksumType(CY_LINCH0_TYPE, msgContext[currentMsgIdx].checksumType);


        /* (18)Check the current message type */
        if (msgContext[currentMsgIdx].responseDirection == LIN_TX_RESPONSE)

            {
        /* 18)-1 Enable event interrupt (See Code Listing 11) */
                Lin_SetInterruptMask(CY_LINCH0_TYPE, CY_LIN_INTR_TX_RESPONSE_DONE |
CY_LIN_INTR_RX_RESPONSE_DONE | CY_LIN_INTR_ALL_ERROR_MASK_SLAVE);


        /* (18)-2 Write the response data (See Code Listing 10) */
                Lin_WriteData(CY_LINCH0_TYPE, msgContext[currentMsgIdx].dataBuffer,
msgContext[currentMsgIdx].dataLength);

                lin_state = LIN_STATE_TX_RESPONSE;        /* (18)-3 LIN_STATE_TX_RESPONSE */

        /* (18)-4 TX_RESPONSE bit = 1H (See Code Listing 12) */
                Lin_SetCmd(CY_LINCH0_TYPE, CY_LIN_CMD_TX_RESPONSE);
            }
            else
            {
        /* (18)-5 LIN_STATE_RX_RESPONSE */
                lin_state = LIN_STATE_RX_RESPONSE;
            }
        }
        else
        {
    :
```

6 用語集

用語	説明
LIN	Local Interconnect Network
LIN トランシーバ	LIN バスは、enable 機能を含む 3 ピンのインタフェースにより外付けのトランシーバと接続され、マスタとスレーブの機能をサポートします。
GPIO	General Purpose Input/Output
AUTOSAR	AUTomotive Open System Architecture
ヘッダ	マスタによってのみ送信され Break 領域、Sync 領域保護識別子 (PID) 領域から構成されます。詳細は、Architecture TRM の LIN Message Frame Format 章を参照してください。
応答	マスタとスレーブから送信され最大 8 つのデータ領域とチェックサム領域から構成されます。詳細は、Architecture TRM の LIN Message Frame Format 章を参照してください。
MMIO	Memory Mapped I/O
PID	保護識別子。Protected Identifier の略
PERI clock	PERipheral Interconnect clock
メッセージタイプ	メッセージタイプは応答が、マスタ, スレーブのどちらから送られるかを示します。
マスタ応答	マスタがヘッダを送信し応答も送信することを示します。このタイプは、スレーブを制御するために使われます。詳細は、Architecture TRM の LIN 章の LIN Message Transfer セクションを参照してください。
スレーブ応答	マスタがヘッダを送信します。そしてスレーブが応答を送信しマスタがそれを受信します。このタイプは、スレーブの状態を確認するのに用いることができます。詳細は、Architecture TRM の LIN 章の LIN Message Transfer セクションを参照してください。
Slave-to-Slave	マスタがヘッダを送信します。スレーブが応答を送信し、もう一つのスレーブがその応答を受信します。詳細は、Architecture TRM の LIN 章の LIN Message Transfer セクションを参照してください。
データ長	LIN_CH_CTL1 レジスタ DATA_NR [2:0] ビットにより応答のデータ領域の数が設定されます。(チェックサムは含まれません)
チェックサムタイプ	クラシックモードとエンハンスモードの 2 種類のチェックサムのタイプがサポートされます。クラシックモードでは PID 領域にチェックサムの計算結果が含まれ、エンハンスモードでは、PID 領域にチェックサムの計算結果が含まれません。
ISR	Interrupt Service Routine
IRQ	Interrupt ReQuest

7 関連ドキュメント

以下は、TRAVEO™ T2G ファミリシリーズのデータシートとテクニカルリファレンスマニュアルです。これらのドキュメントを入手するには、テクニカルサポートにご連絡ください。

- データシート
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller TRAVEO™ T2G Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller TRAVEO™ T2G Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller TRAVEO™ T2G Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller TRAVEO™ T2G Family
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller TRAVEO™ T2G Family
- Body Controller Entry ファミリ
  - TRAVEO™ T2G Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - TRAVEO™ T2G Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - TRAVEO™ T2G Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- Body Controller High ファミリ
  - TRAVEO™ T2G Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - TRAVEO™ T2G Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT4BF
  - TRAVEO™ T2G Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D ファミリ
  - TRAVEO™ T2G Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - TRAVEO™ T2G Automotive Cluster 2D Registers Technical Reference Manual (TRM)

8 その他の参考資料

さまざまな周辺機器にアクセスするためのサンプルソフトウェアとしてのスタートアップを含むサンプルドライバライブラリ (SDL) が提供されます。SDL は、公式の AUTOSAR 製品でカバーされないドライバの顧客へのリファレンスとしても機能します。SDL は自動車規格に適合しないため、製造目的で使用できません。このアプリケーションノートのプログラムコードは SDL の一部です。SDL の入手については、テクニカルサポートに連絡してください。

***重要**:*

改訂履歴

版数	発行日	変更内容
**	2019-07-18	これは英語版 002-25346 Rev. **を翻訳した日本語版 Rev. **です。英語版の改訂内容: New application note
*A	2020-07-27	これは英語版 002-25346 Rev. *A を翻訳した日本語版 Rev. *A です。英語版の改訂内容: Changed target parts number (CYT2/CYT4 series) Added target parts number (CYT3 series)
*B	2021-04-26	これは英語版 002-25346 Rev. *B を翻訳した日本語版 Rev. *B です。英語版の改訂内容: Added example of SDL Code and description. MOVED TO INFINEON TEMPLATE.
*C	2024-12-04	これは英語版 002-25346 Rev. *C を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Template update; no content update

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-12-04

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-jyl1683100159465

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。