参考資料

Application Note

MSPM0 デバイスのローカル相互接続ネットワーク (LIN) 自動ボーレート検出メカニズム

Kalyan Gajjala, Gaurang Gupta, Ashwini Gopinath *EP-MSP*

概要

このアプリケーション ノートでは、MSPM0 UART および UNICOMM UART レシーバの LIN ノードによって提供される自動ボーレート検出メカニズムの概要について詳しく説明します。ここに示す情報は、同期方式によって必要なクロック周波数の許容誤差を達成するデバイスに、特に関連しています。MSPM0 UART および UNICOMM UART モジュールでは、ハードウェアおよびソフトウェア ソリューションの組み合わせを使用してボーレートの自動検出が行われます。

このドキュメントではまず、LIN (Local Interconnect Network) プロトコルで規定されている許容誤差要件の概要を示します。次に、MSPM0 UART/UNICOMM UART モジュールが、可変ボーレートを確実に検出しながら、これらの許容誤差制限を満たすように設計されていることを説明します。このアプリケーション ノートには、同期プロセスの詳細が説明されています。クロックが不正確になる可能性のあるシステムでも、高精度のデータ受信が可能になります。組込みシステムの設計者がプロジェクト内で自動ボーレート検出を効果的に活用できるように、統合の詳細、実用的な実装手法、および特定用途向けの検討事項を解説します。

目次

商標

1 LIN プロトコルの概要

LIN (Local Interconnect Network) は、車載アプリケーションに特化して設計されたシリアル通信プロトコルです。低コストの標準 UART インターフェイスを活用し、ソフトウェア ソリューションによる導入を可能にします。LIN プロトコルでは、1 ～ 20Kb/S の速度が許容されます。

標準的な LIN フレームは、図 1-1 に示すように、ヘッダーと応答の 2 つの部分に分けられます。ヘッダーは、ブレーク フィールド、同期フィールド、保護識別子 (PID) フィールドで構成されています。応答はデータとチェックサムで構成されます。

バイト間スペースとは、1 バイトのストップ ビットの終了から、次のバイトのスタート ビットまでの時間です。応答スペースとは、PID フィールドから最初のデータ バイトまでの時間です。

すべてのバイト フィールド (ブレーク フィールドを除く) は、最下位ビット (LSB) が最初に、最上位ビット (MSB) が最後に送信されます。

このアプリケーション ノートで以降に使用される一部の用語表記 (利便性のため LIN 仕様からの引用として使用):

- 公称ビット レート: F_{Nom}
- 同期前のレスポンダ ノードの公称ビット ノードからの偏差:F_{TOL_UNSYNC} (< ±14%)
- 同期後のレスポンダ ノードの偏差:F_{TOL_SYNC} (< ±1.5%)

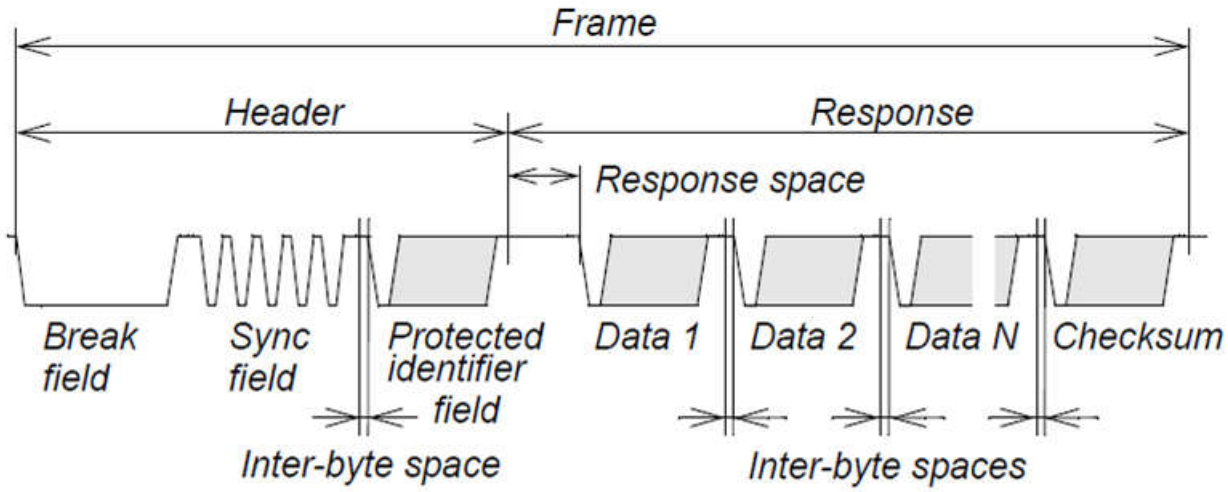


図 1-1. LIN フレーム

1.1 ブレーク フィールド

ブレーク フィールドは LIN フレームの開始を示し、常にコマンダ ノードから送信されます。ブレーク フィールドでは、ドミナント状態 (ゼロ) が少なくとも 13 公称ビット時間継続し、その後少なくとも 1 公称ビット時間のブレーク デリミタが続きます。これを次の図に示します。レスポンダ ノードは、9.5 ビット時間以上連続してドミナント ビットが検出された場合にブレーク フィールドを検出でき、ブレーク デリミタが実際に 1 ビットの時間長であることを検証する必要はありません。

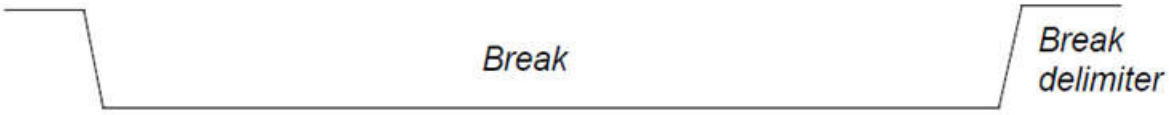


図 1-2. ブレーク フィールド

1.2 同期バイト フィールド

同期フィールドは、16 進数値 **0x55** に固定された 1 バイトであり、LIN バス上のデバイスがマスタ ノードとクロック同期することを可能にします。値 0x55 は 1 と 0 が交互に出現するバイナリ パターンを表し、予測可能な信号遷移 (エッジ) を生成します。

その仕組みは次のとおりです。

- LIN コマンダは、この交互パターンを送信します。
- レスポンダ ノードは、これらの信号遷移間の時間を測定して、マスタの正確なビット タイミングを判定します。
- 次に、レスポンダ ノードはこの測定値を使用してボーレートを調整し、レスポンダのボーレートと完全に同期するようにします。

図 1-3. 同期フィールド

1.3 PID のフィールド

LIN の PID (保護識別子) は、各 LIN フレーム ヘッダー内の 8 ビット フィールドで、メッセージを一意に識別します。次の図に示します。

6 ビットの識別子 (ID) が含まれており、64 の固有のメッセージ タイプが可能です。残りの 2 ビットは、ID ビットから計算されたパリティ ビット (P0 および P1) で、送信中のエラーを検出するのに役立ちます。

P0 = ID0 XOR ID1 XOR ID2 XOR ID4

P1 = NOT(ID1 XOR ID3 XOR ID4 XOR ID5)

PID は常にコマンダ ノードから送信されます。すべてのレスポンダ ノードはこれを使用して、メッセージに応答するか、無視するか、受信のみを行うかを決定します。

パリティ ビットは、ID が正しく受信されたかどうかをノードが確認できるようにすることで、データの信頼性を確保するのに役立ちます。

図 1-4. PID のフィールド

1.4 データ

データ フィールドには、センサ読み取り値や制御コマンドなど、コマンダ ノードとレスポンダ ノード間で実際にやり取りされる情報が格納されます。

データ フィールドには通常、実際のペイロードを表す 1 ～ 8 バイトが含まれています。フレームに含まれるデータ バイト数は、そのフレームの PID によって定義され、コマンダ ノードとレスポンダ ノードの間で合意されています。

コマンダ ノードがヘッダーを送信してデータを要求すると、レスポンダ ノードはデータ フィールドに関連データを入力して返信します。

データの後には常にチェックサム バイトが続きます。このバイトは、すべてのノードにおいて、データが正しく受信され、送信中に破損していないことを確認するのに役立ちます。

図 1-5. データ

1.5 チェックサム

チェックサムは常に LIN フレームの応答部分の最後のバイトとして、データを送信するノードから送信されます。チェックサムの計算では、すべてのデータ バイト、またはすべてのデータ バイトと保護識別子を加算します (256 をモジュロとした加算を使用します。そのため、合計が 255 を超えると、値は先頭に戻って加算されます)。次に合計を反転します。これにより、すべてのバイトにチェックサムを加算した結果が 0xFF に等しくなります。

2 初期ボーレート設定

UART の機能ブロック図については、MSPM0 デバイス固有のユーザー ガイドを参照してください。MSPM0 UART / UNICOMM UART への入力クロックは、CLKDIV レジスタのビットフィールドを使用して、さらに分周できます。この分周クロックは、機能クロックまたは UART クロックと呼ばれます。IP のクロック速度については、該当するデバイスのデータシートを参照してください。

ボーレート除数は、16 ビット整数 (IBRD) と 6 ビット分数部 (FBRD) で構成される 22 ビットの数値です。これら 2 つの値によって形成される数値は、ボーレート ジェネレータで使用され、ビット サンプル周期が決定されます。小数ボーレート除数を使用すると、UART はすべての標準ボーレートを非常に正確に生成できます。

16 ビットの整数は UART 整数ボーレート除数 IBRD レジスタにロードされ、6 ビットの小数部は UART 小数ボーレート除数 FBRD レジスタにロードされます。

ボーレート除数 (BRD) は、次の式を使用して計算できます。

BRD = 機能クロック / (オーバーサンプリング x ボーレート)

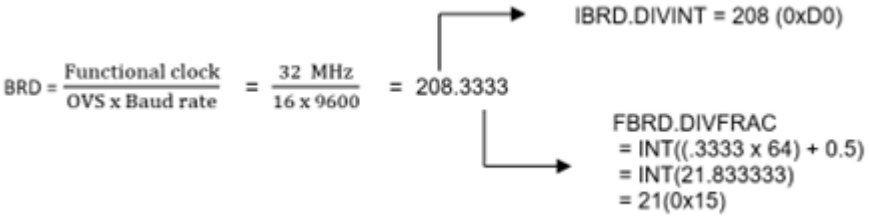
機能クロックは、CLKSEL および CLKDIV によって設定される UART クロック制御ロジックのクロック出力です。オーバーサンプリングは、CTL0 レジスタの高速オーバーサンプリング イネーブル (HSE) ビットによって選択され、16、8、3 のいずれかのオーバーサンプリングを選択できます。

- IBRD = INT(BRD):ボーレート除数の整数部分を含みます
- FBRD = INT ((BRD – INT(BRD))*64+0.5):ボーレート除数の残余の小数部部分を含みます

BRD の整数部は IBRD レジスタにロードされます。6 ビットの小数値は、FBRD レジスタにロードする必要があります。

以下の例は、9600 bit/s のボーレートの IBRD.DIVINT および FBRD.DIVFRAC を簡単に計算する方法を示しています。

- 機能クロック = 32MHz
- オーバーサンプリング = 16
- ボーレート = 9600 bit/s



3 LIN プロトコル MSPM0 UART / UNICOMM UART の実装

MSPM0 / UNICOMM の UART モジュールには、LIN (Local Interconnect Network) プロトコルの実装をサポートするための専用ハードウェア機能が組み込まれています。これらのハードウェア拡張は、ソフトウェアのオーバーヘッドを低減し、LIN 通信に必要な高精度のタイミング制御を実現するために特別に設計されています。LIN (Local Interconnect Network) プロトコルをサポートするため、UART モジュールに次のハードウェア拡張機能が実装されています。

- UART クロックで駆動される 16 ビット アップカウンタ (LINCNT) (UART クロックの生成方法の詳細については、TRM を参照してください)。
- カウンタ オーバーフロー時の割り込み機能 (CPU_INT.IMASK.LINOVF)。
- 16 ビット キャプチャ レジスタ (LINC0)。次の 2 つの構成可能なモードを備えています。
  - RXD 立ち下がりエッジでの LINCNT 値のキャプチャ。キャプチャ時の割り込み機能
  - LINCNT の比較。一致時の割り込み機能
- 16 ビット キャプチャ レジスタ (LINC1) を構成可能
- RXD 立ち上がりエッジでの LINCNT 値のキャプチャ。キャプチャ時の割り込み機能。
- Rx 立ち上がりエッジ (RXPE) および Rx 立ち下がりエッジ (RXNE) に対する割り込み機能。

MSP デバイスの UART モジュールは、LIN コマンダおよび LIN レスポンダとして使用できます。以下のセクションでは、LIN プロトコルをサポートするための拡張機能について詳しく説明します。

注

LINC0_CAP と LINC0_MATCH の両方が同時に 1 に設定された場合、MATCH 動作が CAP 動作を無効にするため、LINC0 レジスタは MATCH モードで動作します。

UNICOMM ベースのデバイスでは、CLKDIV をゼロでない値に設定する必要がある場合は、CLKDIV を構成する前に LINC0 MMR を構成する必要があります。(詳細については、デバイス エラッタを参照してください)。

3.1 LIN 送信

コマンダ モード:MSPM0 UART / UNICOMM UART は、ブレーク、SYNC、PID を生成し、PID をデコードした後にデータを送受信します。

3.1.1 ブレーク フィールド

ブレーク フィールドは、LIN フレームの開始を示す LIN 通信の重要な部分です。MSP デバイスでブレーク フィールドを生成するには、LCRH.BRK ビットを使用する必要があります。UART.LCRH レジスタの BRK ビットを 1 に設定すると、UART が連続的な Low 信号を出力するよう強制され、ブレーク フィールドが生成されます。

この時間は、ソフトウェア / アプリケーションによって制御されます。

TXDATA/FIFO にデータを入力する前に、LCRH.BRK ビットを設定する必要があります。ブレーク フィールドの継続時間は、少なくとも 13 ビット時間のドミナント状態が続き、その後に 1 ビット時間のブレーク デリミタが続きます。

ブレーク フィールドが送信されたら、LCRH.BRK ビットをクリアする必要があります。

実装シーケンス:

- LCRH.BRK ビットをアサート
- ソフトウェアを使用して、所定の時間ドミナント状態を維持
- デリミタのために LCRH.BRK ビットをクリアし

注

適切なブレーク フィールドが生成されるように、LCRH.BRK ビットの設定は TXDATA/FIFO 動作の前に行う必要があります。

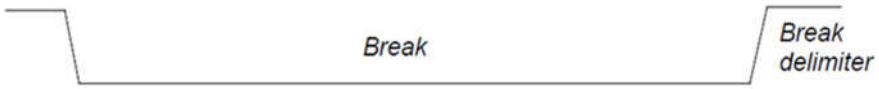


図 3-1. ブレーク フィールドの図示

または、比較モードで LINCNT と LINC0 を使用して、LIN ブレーク フィールドのタイミングを正確に設定することもできます。LINC0 は、ブレーク フィールドの継続時間と一致するよう設定でき、LCRH.BRK を設定してブレーク フィールドの送信を開始した後、LINCNT を有効にできます。LINC0 ISR で LCRH.BRK がクリアされると、ブレーク フィールドは終了します。さらに、GPTIMER などの他の IP からの割り込みを、この目的で使用できます。

3.1.2 同期フィールド

ブレーク フィールドの送信と LCRH.BRK ビットのデアサート後、同期フィールド送信フェーズを開始するには、同期フィールド (0x55) を UART TXFIFO にロードする必要があります。

3.1.3 PID のフィールド

同期フィールド送信が完了した後、保護識別子 (PID) を TXDATA/FIFO にロードし、同期フィールドと PID 送信シーケンスの間に必要なバイト間隔を維持する必要があります。

3.1.4 データ フィールド

保護識別子 (PID) の送信完了後、LIN フレームでの送信に指定された必要なデータ バイトを TXDATA/FIFO に入力できます。

次のコード シーケンスは、LIN フレーム ヘッダーのブレーク フィールド、同期フィールド、保護識別子 (PID) をシーケンシャルに送信するための実装プロトコルを示しています。

```
/* Transmit BREAK, SYNC byte, and PID */
DL_UNICOMMUART_enableLINSendBreak(uart);//initiate the process to send the break field by setting LCRH.BRK bit
delay_cycles(LIN_BREAK_LENGTH); /* Send break field conforming to the timing specifications, calculated based on the baud rate */
DL_UNICOMMUART_disableLINSendBreak(uart);//abort the break field transmission by clearing the LCRH.BRK bit
DL_UNICOMMUART_transmitDataBlocking(uart, LIN_SYNC_BYTE);//load the UART TXDATA/FIFO with SYNC field (0x55)
delay_cycles(LIN_INTER_BYTE_SPACE);//provide the inter-byte space between SYNC field and PID field
DL_UNICOMMUART_transmitDataBlocking(uart, messageTable[tableIndex].msgID);//Send the PID, in this case 0x0D
```

図 3-2. ブレーク フィールド、同期フィールド、データ バイトを送信するためのソフトウェア シーケンス

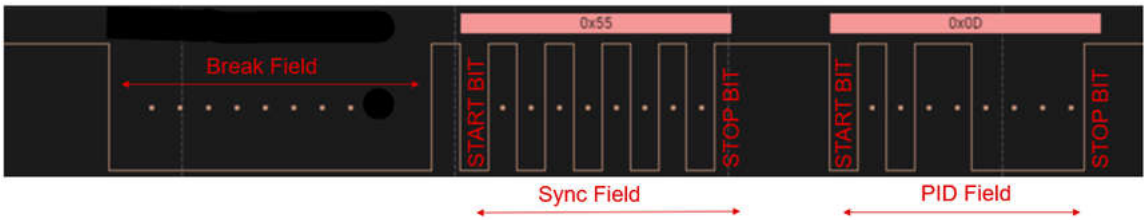


図 3-3. LIN コマンダによって送信されたブレーク フィールド、同期フィールド、PID フィールド

3.1.5 チェックサム

LIN プロトコルは、反転チェックサム計算を利用した 8 ビット エラー検出メカニズムを実装しています。チェックサム計算には、0xFF を超える値に対してバイトの逐次加算とキャリー管理が行われ、最終的な合計に対して 1 の補数演算が行われます。

以下の 2 つのチェックサム方式が指定されています。

1. クラシック チェックサム: 計算にはデータ バイトのみが含まれます
2. 拡張チェックサム: 計算には保護識別子 (PID) とデータ バイトを連結したものが含まれます

チェックサム アルゴリズム:

- バイトの逐次加算

- 合計が 0xFF を超えた場合のキャリー加算
- 結果の反転 (1 の補数)

実装例:

保護識別子:0x0D

データ フィールド:[0xAB, 0xBC, 0xCD, 0xDE, 0xEF]

拡張チェックサム計算シーケンス:

ステップ 1:0x0D

初期値 = 0x0D

ステップ 2:0xAB を追加

0x0D + 0xAB = 0xB8

(合計 < 256 の場合に 255 を減算する必要はありません)

ステップ 3:0xBC を追加

0xB8 + 0xBC = 0x174

合計 ≥ 256 (0x100) の場合、255 (0xFF) を減算します

0x174 - 0xFF = 0x75

ステップ 4:0xCD を追加

0x75 + 0xCD = 0x142

合計 ≥ 256 の場合、255 を減算します

0x142 - 0xFF = 0x43

ステップ 5:0xDE を追加

0x43 + 0xDE = 0x121

合計 ≥ 256 の場合、255 を減算します

0x121 - 0xFF = 0x22

ステップ 6:0xEF を追加

0x22 + 0xEF = 0x111

合計 ≥ 256 の場合、255 を減算します

0x111 - 0xFF = 0x12

最後のステップ:結果の反転

チェックサム = 0xFF - 0x12 = 0xED

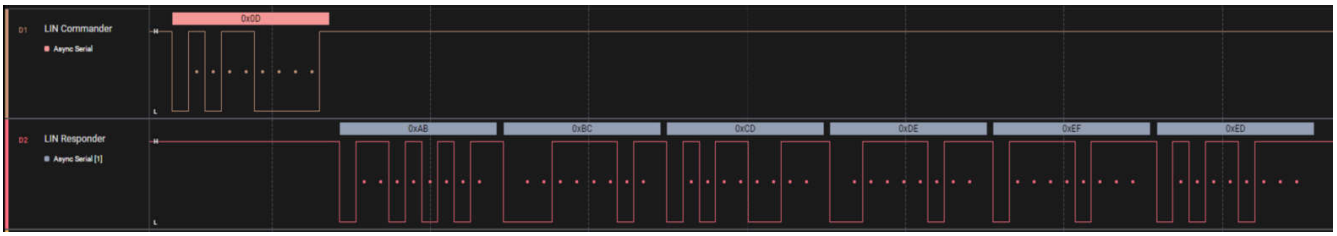


図 3-4. チェックサムの送信

以下に示すコード セクションは、LIN チェックサムの計算と送信のプロセスを示しています。

```c
void sendLINResponderTXMessage( UNICOMM_Inst_Regs *uart, uint8_t tableIndex,
    uint8_t *msgBuffer, LIN_table_record_t *responderMessageTable)
{
    uint8_t locIndex;
    uint8_t checksum;
    LIN_word_t tempChksum;

    /* Disable LIN RX */
    DL_UART_Extend_disableInterrupt(uart, DL_UART_EXTEND_INTERRUPT_RX);

    tempChksum.word = responderMessageTable[tableIndex].msgID;
    tempChksum.word = tempChksum.byte[0] + tempChksum.byte[1];

    for (locIndex = 0; locIndex < responderMessageTable[tableIndex].msgSize;
         locIndex++) {
        DL_UART_Extend_transmitDataBlocking(uart, msgBuffer[locIndex]);
        tempChksum.word += msgBuffer[locIndex];
    }
    /* Calculate and send checksum */
    checksum = tempChksum.byte[0];
    checksum += tempChksum.byte[1];
    checksum = 0xFF - checksum;

    DL_UART_Extend_transmitDataBlocking(uart, checksum);
    while (DL_UART_Extend_isBusy(uart)) {
        ;
    }

    DL_UART_Extend_receiveDataBlocking(uart);

    /* Enable LIN RX */
    DL_UART_Extend_clearInterruptStatus(uart, DL_UART_EXTEND_INTERRUPT_RX);
    DL_UART_Extend_enableInterrupt(uart, DL_UART_EXTEND_INTERRUPT_RX);
}
```

図 3-5. チェックサムを送信するためのソフトウェア シーケンス

3.2 LIN 受信

レスポンダ モード:MSPM0 UART / UNICOMM UART は、ブレーク検出を待ち、PID をデコードした後にデータを送受信します。

LIN コマンダは各フレームの開始時にブレーク フィールドと同期フィールドを送信します。LIN レスポンダのソフトウェア ドライバが BREAK-SYNC を適切に検出し、ボーレートの調整や誤差の判定に必要なタイミング パラメータを測定できるよう、ハードウェアが追加されています。

3.2.1 ブレーク フィールド検出

LIN フレームを受信するには、カウンタおよび比較モード機能を利用した正確なブレーク フィールド検出が必要です。

構成シーケンス:

1. カウンタの初期化
   - LIN カウンタをリセット (UARTx.LINCNT = 0)
2. 比較モードの構成
   - カウンタ比較一致モードをアサート (UARTx.LINCTL.LINC0_MATCH = 1)
   - UARTx.LINC0 を 9.5 x Tbit スレッショルド値で設定
   - LINC0 一致割り込みを有効化 (CPU_INT.IMASK.LINC0 = 1)
3. カウンタ制御パラメータ (UARTx.LINCTL)
   - RXD Low 状態カウントを有効化 (LINCTL.CNTRXLOW = 1)
   - 立ち下がりエッジ カウンタのリセットを設定 (LINCTL.ZERONE = 1)
   - カウンタ動作を有効化 (LINCTL.CTRENA = 1)
4. 検出機能
   - RX 立ち上がりエッジ割り込み機能 (CPU_INT.IMASK.RXPE = 1)
   - RXPE 割り込みが起動すると、ソフトウェアは LINCNT を直接読み取ることで、ブレーク フィールドのタイミングを確認できます。

オプション:ユーザーは、LIN カウンタのオーバーフロー割り込みを有効化 (CPU_INT.IMASK.LINOVF = 1) することで、ブレーク フィールドが長すぎて 16 ビット カウンタがオーバーフローしたことを検出できます。このタイムアウトは、**$t_{Timeout}$ = 216 / UART クロック**として計算できます

3.2.2 同期フィールドの検証

LIN フレーム ヘッダーの正確なタイミング精度を確保し、コマンダのボーレート パラメータを決定するためには、同期フィールドの検証が重要です。検証シーケンスが成功することにより、複数の基準 (ブレーク フィールド検出、正確な通信タイミング、正確な保護識別子 (PID) の受信、全体的なフレーム同期の整合性) を満たすことが確認されます。

同期フィールドは、事前定義された 0x55 バイトのパターン (01010101) で構成され、特に以下のために設計されています。

- 交互ビット パターンによる高精度のタイミング リファレンス
- 4 つの明確なビット時間測定の機会
- 検証のための確定的な遷移間隔

この構造化パターンにより、受信ノードは以下のことが可能になります。

- 実際の通信パラメータの決定
- 必要に応じたボーレート調整の実行
- マスタ タイミング リファレンスとの同期

ブレーク フィールドが検証されると、システムは RX 立ち上がりエッジで LINCNT 値をキャプチャする LINC1 キャプチャ レジスタを利用して、同期フィールド測定を開始します。LINCNT カウンタは立ち下がりエッジでリセットされ、RX LOW 状態でインクリメントするように構成されています。LINC1 キャプチャ動作と RX 立ち上がりエッジ割り込みは、各立ち上がりエッジ遷移でトリガされます。割り込みサービス ルーチン中に、ソフトウェアは LINC1 レジスタ値により個別のビット タイミング パラメータを分析し、タイミング仕様を検証し、必要に応じてボーレート調整を実行します。

図 3-6. 同期フィールド - 0x55

3.3 LIN トランシーバ

このアプリケーション ノートでは、TLIN2029A-Q1 評価基板 (EVM) を外部 LIN トランシーバとして使用しています。次の図に、MSPM0 コマンダおよびレスポンダと TLIN2029A-Q1 トランシーバとの接続方法のブロック図を示します。

トランシーバと LIN コマンダ / レスポンダ間の回路図接続の詳細については、『TLIN2029-Q1 EVM ユーザー ガイド』を参照してください。

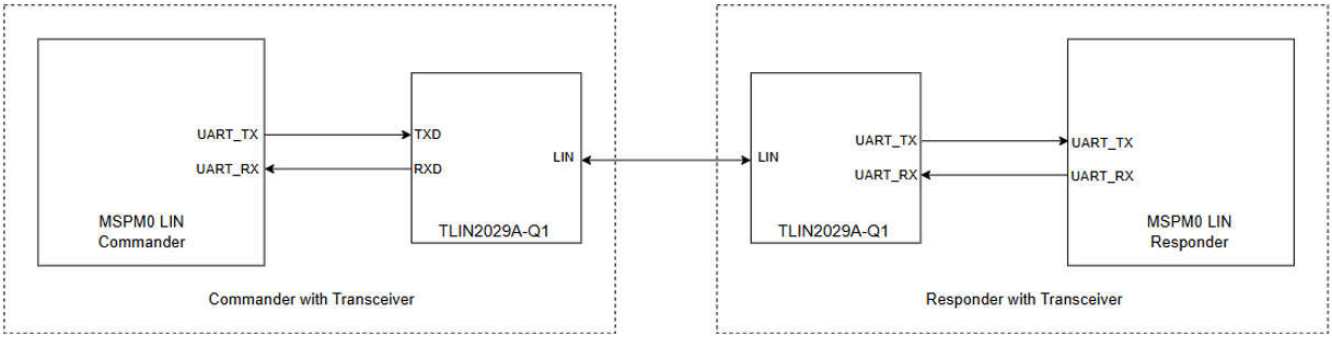


図 3-7. TLIN2029A-Q1 トランシーバと接続された MSPM0 コマンダおよびレスポンダのブロック図

4 自動ボーレート検出

LIN における自動ボーレート検出とは、LIN フレームが開始されるたびに、レスポンダ ノードがコマンダのボーレートを自動的に認識して調整するプロセスです。

コマンダ ノードは各フレームの開始時に特別な同期フィールドを送信します。このフィールドは常にバイト 0x55 (バイナリ 01010101) です。

レスポンダ ノードは、MSPM0 / UNICOMM レジスタを利用して、同期バイトのビット時間 (Tbit) を測定することにより、現在のボーレートを計算できます (セクション 3.2.2 を参照)。

4.1 MSPM0 UART / UNICOMM UART を使用してビット幅を測定する手順

レスポンダ ノードは、同期バイトの各立ち上がりエッジでのビット時間 (Tbit) を測定することで、現在のボーレートを計算できます。立ち上がりエッジは、次の図に示すように、1、3、5、7、STOP ビット時間の間隔で利用できます。

自動ボーレート検出で、同期フィールドのビット タイミングを計算するために、MSPM0/UNICOMM では以下のレジスタを使用できます。

1. 有効なブレーク フィールドを検出した後、LIN カウンタを 0 (LINCNT = 0) に初期化します。
2. RX 立ち上がりエッジでの割り込みを有効にします (CPU_INT.IMASK.RXPE = 1)
3. LIN カウント制御 (LINCTL) レジスタを設定します
   a. RX 立ち下がりエッジでの LIN カウンタ クリアを有効化 (LINCTL.ZERONE = 1)
   b. RX の Low 信号時のカウントを有効化 (LINCTL.CNTRXLOW = 1)
   c. 立ち上がり RX エッジでの LIN カウンタのキャプチャを有効化 (LINCTL.LINC1CAP = 1)
   d. LIN カウンタを有効化 (LINCTL.CTRENA = 1)

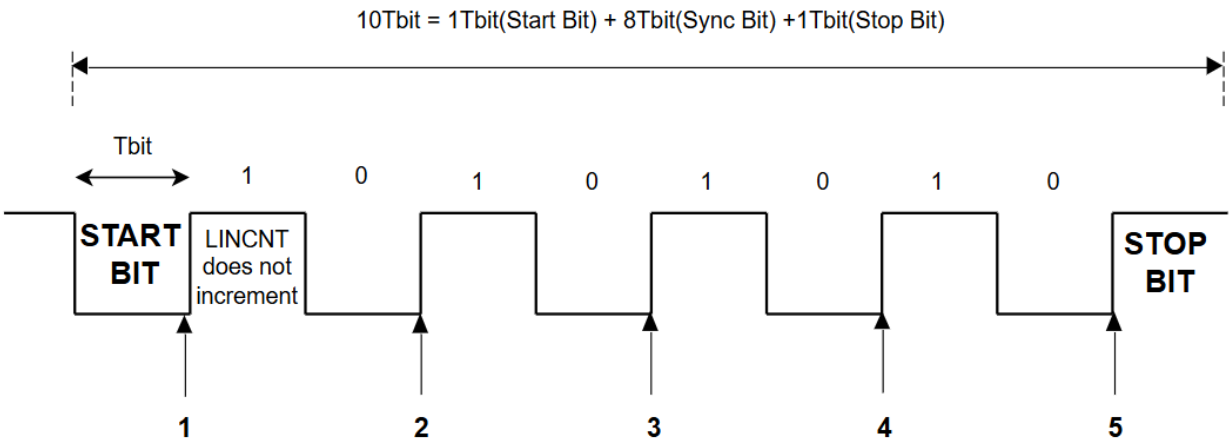


図 4-1. LIN 同期フィールドの検証

同期フィールド中に RX ラインの各立ち上がりエッジで実行されるアクションは、次のとおりです。

1. LIN カウンタは各立ち下がり RX エッジで 0 にリセットされ、T ビット時間にわたって RX が Low になるとカウントを開始します。
2. LINCNT の値は、各 RX 立ち上がりエッジで LINC1 レジスタによってキャプチャされます。
3. RX 立ち上がりエッジ割り込みが、5 回繰り返しトリガされます (RXPE)。
   - RX 立ち上がりエッジ割り込みサービス ルーチン (ISR) が繰り返されるごとに、LINC1 キャプチャ レジスタが読み取られます。LIN カウンタは RX が Low のときのみカウントするように構成されるため、キャプチャされた値は T ビット時間を表します。

4.2 正しいボーレートの計算

同期の前にレスポンダのクロックが 32MHz (公称周波数、F_{Nom}) で動作している場合、コマンダ ノードは 9600 ボーレートの同期フィールドを送信します。ソフトウェアは、次の手順で正しいボーレート除数を算出できます。

5 つの LINC1 キャプチャ値の平均値を求めます。

$$\text{Average Bit Time (Tbit)} = \frac{\text{Total Bit time}}{5} = \frac{16665}{5} = 3333 \text{ functional clock cycles} \quad (1)$$

計算された Tbit は、各ビット時間における機能クロック サイクル数を示します。同期後にボーレートの許容誤差 (F_{TOL_SYNC}) を維持するため、計算された値で IBRD/FBRD レジスタを更新する必要があります。

次のフローチャートの例は、Tbit 時間が算出された後に、IBRD (整数ボーレート除数) および FBRD (小数ボーレート除数) の値を決定するための簡単なアプローチを示しています。

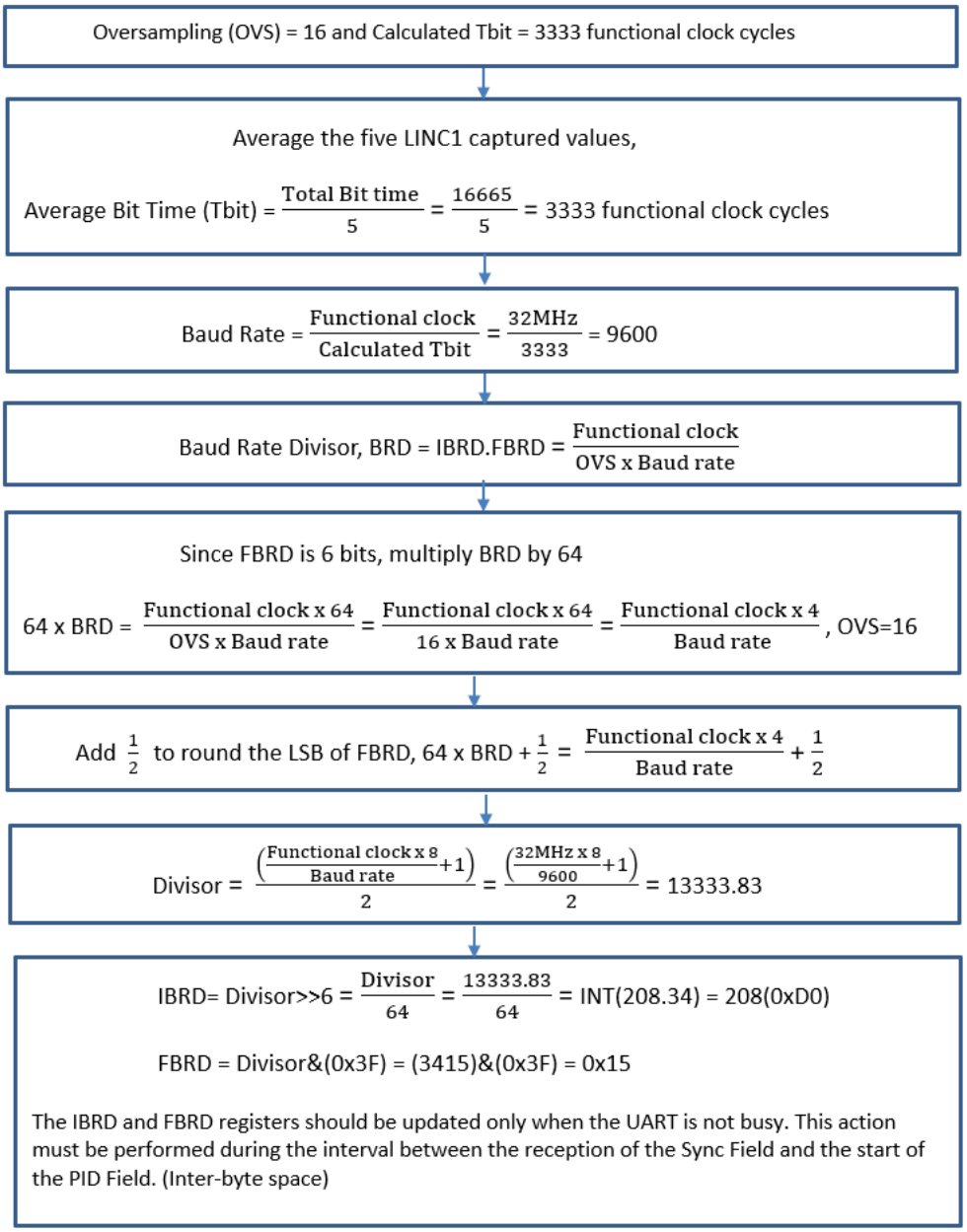


図 4-2. 公称周波数 (32MHz) での LIN ボーレート除数計算のフローチャート

4.2.1 レスポンダ ノードの水晶振動子誤差

同期前にレスポンダのクロックが公称レートより 14% 低速で動作している場合、具体的にはボーレート 9600 において目的の 32MHz ではなく 27.52MHz で動作している場合、計算される Tbit は、正しいクロックに基づいて想定される 3333 サイクルではなく、(誤差のあるクロックに基づいて) 2867 クロック サイクルになります。

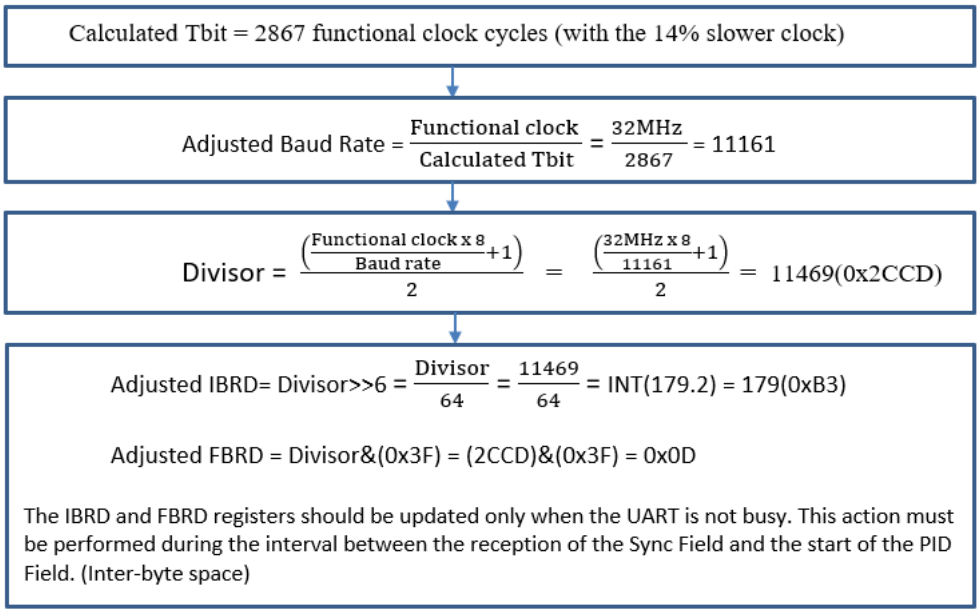


図 4-3. 14% 低速なクロック (27.52MHz) での LIN ボーレート除数計算のフローチャート

同期前にレスポンダのクロックが公称レートより 14% 高速で動作している場合、具体的にはボーレート 9600 において目的の 32MHz ではなく 36.48MHz で動作している場合、計算される Tbit は、正しいクロックに基づいて想定される 3333 サイクルではなく、(誤差のあるクロックに基づいて) 3800 クロック サイクルになります。

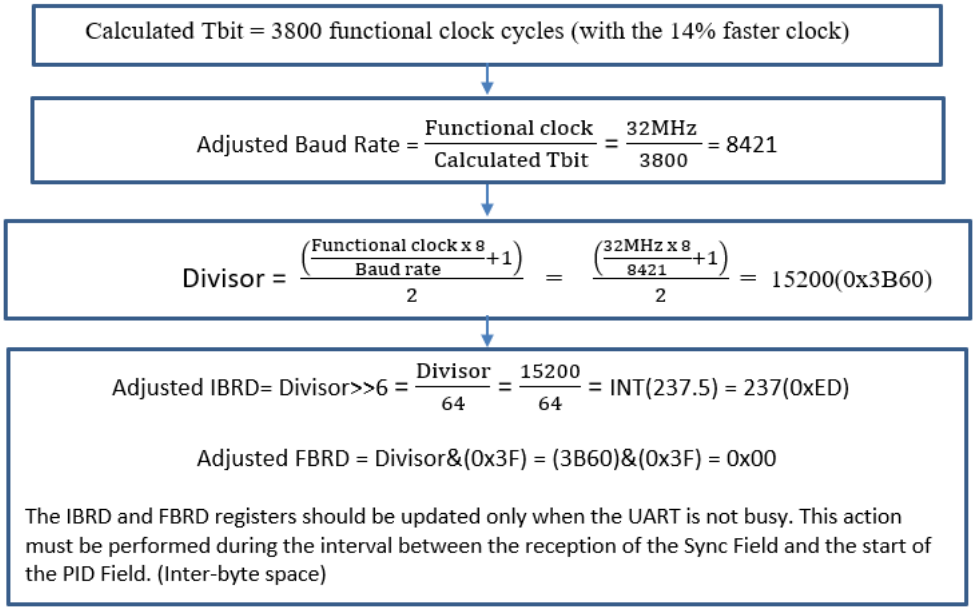


図 4-4. 14% 高速なクロック (36.48MHz) での LIN ボーレート除数計算のフローチャート

以下のソフトウェア コードは、立ち上がりエッジ ISR 内のボーレート検出と設定を示しています。

```
case DL_UART_EXTEND_IIDX_RXD_POS_EDGE:
          /* Signals the positive edge of a sync field segment. */
       if (gStateMachine == LIN_STATE_SYNC_FIELD_POS_EDGE)
         {
              gBitTimes[gNumCycles].posEdge =
                  DL_UART_Extend_getLINRisingEdgeCaptureValue(LIN_0_INST);
              /* Validation check of the timing of the sync field segment.
               * Finding an invalid sync bit stores each bit time to
               * calculate new baud rate */
              if (gBitTimes[gNumCycles].posEdge > ((gLin0TbitWidthVar * 95) / 100) &&
                  gBitTimes[gNumCycles].posEdge < ((gLin0TbitWidthVar * 105) / 100))
                {
                 gNumCycles++;
                }
               else if (!gFirstSyncBit)
                 {
                  gTotalBitTime = gTotalBitTime + gBitTimes[gNumCycles].posEdge;
                  gNumSyncErrors++;
                 }
               else
                {
```

```
          gFirstSyncBit = false;
        }
          /* Only 5 segments of a sync field. */
       if ((gNumSyncErrors + gNumCycles) ==LIN_RESPONDER_SYNC_CYCLES)
        {
          DL_UART_Extend_enableInterrupt(LIN_0_INST, DL_UART_EXTEND_INTERRUPT_RX);
          DL_UART_Extend_disableInterrupt(
              LIN_0_INST, DL_UART_EXTEND_INTERRUPT_RXD_NEG_EDGE);

          /* Track new and previous baud rate for validation when
           * increasing baud rate. Ensures that resets to deal with
           * overrun errors happen on the appropriate frame.
           * Reset all variables relevant to sync field */
          if (gNumCycles == LIN_RESPONDER_SYNC_CYCLES)
            {
              gPrevBaudRate = gCurrBaudRate;
              gAutoBaudUsed = false;
            }
          gNumCycles     = 0;
          gNumSyncErrors = 0;
          gTotalBitTime  = 0;
          gFirstSyncBit  = true;

          /* If 4 sync errors are detected, update baud rate given
          * autobaud is enabled*/
        }
       else if ((gNumSyncErrors == AUTO_BAUD_THRESHOLD) &&AUTO_BAUD_ENABLED)
        {
          averageBitTime   = gTotalBitTime / gNumSyncErrors;
          measuredBaudRate = LIN_0_INST_FREQUENCY / averageBitTime;

          // Wait for UART Busy bit to go LOW
          while(DL_UART_isBusy(LIN_0_INST));

          gLinResponseLapseVar = LIN_0_INST_FREQUENCY / (2 * measuredBaudRate);
          gLin0TbitWidthVar = averageBitTime;
          // Configure new calculated baud rate
         DL_UART_configBaudRate(LIN_0_INST,LIN_0_INST_FREQUENCY,measuredBaudRate);
         DL_UART_Extend_setLINCounterCompareValue(LIN_0_INST,
              gLin0TbitWidthVar * LIN_0_TBIT_COUNTER_COEFFICIENT);


          gPrevBaudRate = gCurrBaudRate;
          gCurrBaudRate = measuredBaudRate;
          gAutoBaudUsed = true;
```

```
            gStateMachine = LIN_STATE_SYNC_FIELD_NEG_EDGE;
          }
        else
          {
            gStateMachine = LIN_STATE_SYNC_FIELD_NEG_EDGE;
          }
      }
      break;
```

5 同期後のボーレートの偏差

同期前にレスポンダのクロックが公称レートより 14% 低速で動作している場合、機能クロックは目的の 32MHz ではなく 27.52MHz になります。同期後、IBRD および FBRD レジスタの値は、それぞれ 0xB3 および 0x0D となります。

調整後ボーレート = $\frac{\text{Functional clock}}{\text{OVS x IBRD.FBRD}} = \frac{27.52\text{MHz}}{16 \times 179.13}$ = 9601.1

想定されるボーレートは 9600 で、対応する想定ビット時間 (T ビット) は 104.16µs です。

同期後に実際に計算されたボーレートは 9601.1 で、実際の T ビット は 104.15µs です。

$$\text{Percentage of error in Baud Rate post synchronization} = \frac{(\text{Expected Tbit} - \text{Actual Tbit})}{\text{Actual Tbit}} \times 100$$

$$= \frac{(104.16 - 104.15)}{104.16} \times 100 = 0.01\%$$

同期後、レスポンダ ノードのボーレート偏差は 0.01% であり、これは LIN 仕様の許容誤差制限 (FTOL_SYNC < ±1.5%) 内に十分収まっています。

6 参考資料

1. MSPM0 G シリーズ 80MHz マイコン テクニカルリファレンス マニュアル (改訂版 C)
2. LIN-2.2 仕様書
3. TLIN2029-Q1 EVM ユーザー ガイド
4. MSPM0Gx51x CAN-FD インターフェイス搭載ミックスド シグナル マイコン データシート (改訂版 B)

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月