VECTOR >

はじめての FlexRay

ビギナー向け資料「はじめてシリーズ」

vector.com

目次

本稿はベクター・ジャパン執筆による原稿に基づき、2011 年に@IT MONOist に連載された記事「次世代車載ネットワーク FlexRay 入門」より転載、加筆、修正したものです。

@IT MONOist
http://monoist.atmarkit.co.jp

発行元:ベクター・ジャパン株式会社

1. 車載ネットワーク FlexRay とは？

車載ネットワーク通信プロトコル「FlexRay」。本稿では、その仕様概要を解説します。

自動車には多くの「ECU（Electronic Control Unit：電子制御装置）」が搭載され、それらがネットワークに接続されて互いに制御情報を通信することで、より高度な機能を実現しています。

この車載ネットワークでは「CAN（Controller Area Network）」や「LIN（Local Interconnect Network）」などが標準的な通信プロトコル（規格）として知られていますが、より優れた高速性と信頼性を実現する技術が「FlexRay」です。

FlexRay は、自動車内のエレクトロニクス化が進むに従って表面化してきたさまざまな問題点（例えば通信量の増加など）を解決する技術として、またいわゆる"X-by-Wire"を実現するための基幹技術として策定された通信プロトコルであり、高い柔軟性と信頼性、高速性を特長としています。

本稿第 1 章では、まず FlexRay が策定された背景、通信プロトコルの主な特長とそれぞれの概要について解説します。

1.1 なぜ新しい通信プロトコルが求められたか？

自動車の電子制御化が進むに従い、さまざまなニーズが生まれてきました。例えば以下のような事柄です。

> ECU 数の増加に伴う、車載ネットワークの通信量の増加、複雑化への対応
> 商品性の向上、システムの最適化、コスト削減のための統合的な協調制御
> 機能や性能の向上のための X-by-Wire システムへの移行

1.1.1 車載ネットワークの通信量の増加、複雑化への対応

1997 年以前、パワートレイン（駆動系）向け車載ネットワークに接続されている ECU（通信ノード）の数は 6～8 程度でした。ところがその後、上級車に搭載されている ECU の数は 100 近くにもなり、通信量が膨大になりました。またネットワークが複雑化した結果、設計の難易度が増し、品質や機能の低減も懸念されました。これらの問題は CAN/LIN の通信量／通信速度では解決が難しい場合もあり、より多くのデータを通信できるネットワークプロトコルが必要となりました。

1.1.2 商品性の向上、システムの最適化、コスト削減のための統合的な協調制御

以前、各 ECU は個別に機能分担されて制御されていましたが、さらなる商品性の向上、システムの最適化、コスト削減に伴い、分散配置された ECU 同士がネットワークを介することで車両を統合的に協調制御するシステムが求められました。

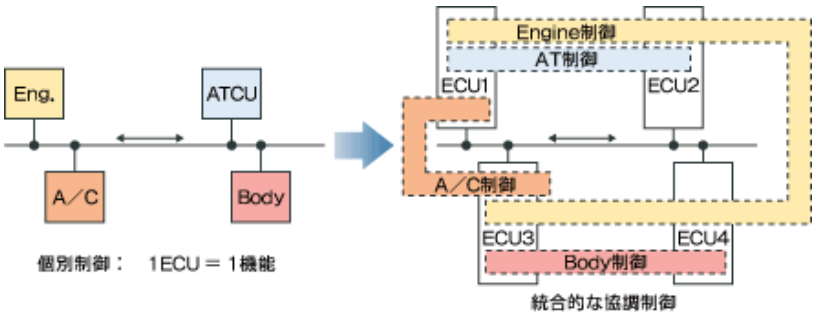

図 1 個別制御から協調制御へ

このようなシステムでは、各 ECU ができるだけ高速かつ同期して通信することが求められます。一方、例えば CAN での通信フレームは送信要求が発生したときに送信され、その送信時間が必ずしも決まっていません。フレームが同時に送信され、フレーム同士の衝突が発生する場合もあり、これは送信遅延につながります。つまり、その送信遅延は通信の負荷状況（ネットワークの混雑状況など）に依存するため、協調制御では課題があります。一部必要なデータが「ネットワークが混んでいたため遅れました……」となっては都合が悪いのです。

1.1.3 機能や性能の向上のための X-by-Wire システムへの移行

X-by-Wire は、機械的な伝達機構で動作している自動車の操作部分をエレクトロニクス部品に置き換える、自動車の構造そのものを根本的に変える技術です。操作部分とアクチュエータを電気的に接続すれば済むため、機械的な部品が大幅に削減され、自動車の設計やレイアウトの自由度は大幅に高まります。例えば、X-by-Wire をステアリングやブレーキに導入した場合、コラムシャフト、油圧機構、パーキング用のケーブルなどの伝達機構が必要なくなります。また、これらを電子制御化することにより、他機構との統合制御に組み込むことがより簡単になり、安全性や快適性の大幅な向上が期待できます。

この X-by-Wire を実現する際、高い信頼性／通信速度を持つ通信仕様が求められます。例えば、通信不良のためにステアリングやブレーキが正常に動作しなければ重大な事故につながってしまいます。CAN では信頼性、通信速度、ともに十分とはいえません。

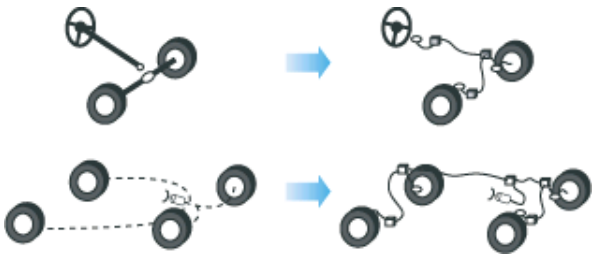

図 2 ステアリング by Wire（上）とブレーキ by Wire（下）

これらのニーズに対応する車載通信プロトコルへの要件を整理すると以下になります。

- **速い通信速度**
- **送信遅延時間が保証されていること（決定論的な送信）**
- **同期通信**
- **高い信頼性**
- **障害耐性**
- **高い柔軟性**

FlexRay は、これらの要件を満たすことができる車載通信プロトコルです。では次に、FlexRay がどのような経緯で策定されたのかを紹介します。

1.2 これまでの経緯

欧州における高信頼性、高速性を備えた通信プロトコルの研究開発は 1980 年代より行われており、一時期、「TTP/C（Time Triggered Protocol for SAE Class. C Applications）」という通信プ ロトコルが EU プロジェクトにも取り上げられるほど有望視されていました。しかし、この TTP/C では信頼性を重視するあまり、コスト面や柔軟性を気にする自動車メーカーの意見を十分に反映せず、プロジェクトメンバーであった Daimler 社や BMW 社が離脱。2000 年、標準化団体 FlexRay コンソーシアムを設立し、BMW 社独自の通信プロトコル「Byteflight（バイトフライト）」を参考にして FlexRay 仕様の策定を進めました。

その後、FlexRay と TTP/C は競合関係にありましたが、2005 年、Volkswagen 社など TTP/C 陣営の主要メンバーが FlexRay コンソーシアムに加入したことで、FlexRay に一本化されました。

そして 2006 年、BMW 社が SUV 車「X5 モデル」にて世界で初めて FlexRay を採用しました。その後、BMW 社「7 シリーズ」「5 シリーズ」モデル、Audi 社「A8」モデルなど、採用車種が拡大しました。

1.3 FlexRay コンソーシアム

通信プロトコルを 1 企業で業界標準化することは難しいため、前述のとおり、複数の企業が標準化団体（FlexRay コンソーシアム）を 2000 年に結成し、通信プロトコルの策定を進めました。この FlexRay コンソーシアムには 7 社のコアメンバー（BMW 社、Bosch 社、Daimler 社、Freescale Semiconductor 社、General Motors 社、NXP Semiconductors 社、Volkswagen 社）を中心に大手自動車メーカー、半導体メーカー、ツールメーカーなど計 100 社以上が加盟しました。

コンソーシアム活動は 2009 年末に終了しています。標準化団体活動の成功例として評価されており、「AUTOSAR」などほかの標準化団体のベンチマークともなっています。

1.4 FlexRay プロトコルの特長

まず、FlexRay プロトコルの主な特長を簡単に紹介します。

- **通信方式**:
  TDMA（Time Division Multiple Access／時分割多重アクセス）。Fixed TDMA と Flexible TDMA の 2 種類に対応
- **ネットワークトポロジー**:
  バス型、スター型、それらの混合型など、さまざまなトポロジーに対応
- **フォールトトレランス（障害耐性）**:
  ネットワークの二重化による冗長性／可能な限り通信を継続するコンセプト
- **ノード構造**:
  ホスト ― 通信コントローラ ― バスドライバ ― バスガーディアンによる相互監視
- **優れた通信同期機能／共通の時間軸**:
  グローバルタイムとさまざまな補正機能
- **高速通信**:
  最大 10Mbps

1.4.1 Fixed TDMA 通信／Flexible TDMA 通信

TDMA（時分割多重アクセス）は、通信時間を一定時間ごと（「タイムスロット」もしくは「スロット」と呼ばれます）に分割することで、多重化通信を実現する「タイムトリガー」通信方式です。

FlexRay ではこの TDMA 通信方式を採用し、各フレームの送信タイミングや順番は事前に定義されます。よって、フレーム同士の衝突は起こらず、また特定のフレームによってネットワークが独占されることもなくなるため、期待どおりのタイミングで通信を行い、通信負荷をある一定以内に保つことができます。一方、CAN のように何らかの事象＝イベントの発生によってフレーム送信が行われる通信方式（イベントトリガー方式：CSMA/CA）の場合、フレームの衝突や調停（衝突したフレームのどちらを先に送信させるかを調整すること）によって、常に期待したタイミングで通信を行えるとは限りません。

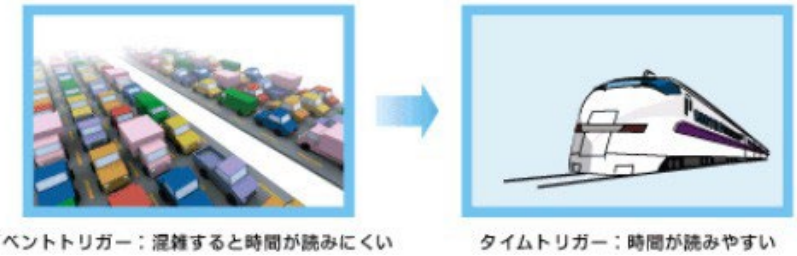

図 3（左）イベントトリガー方式と（、右）タイムトリガー方式のイメージ

また、FlexRay の TDMA には「Fixed TDMA」と「Flexible TDMA」の 2 種類があることが大きな特長です。

1.4.1.1 Fixed TDMA 通信

→ 決まったフレームを決まったタイミングで必ず送信（＝最大送信遅延時間を保証）

→ 送信タイミング（スロットの長さ）は、送信フレームの有無にかかわらず一定

→ 安全性重視のアプリケーションに適している

1.4.1.2 Flexible TDMA 通信

→ 決まったフレームをトリガーに基づいて送信

→ 送信「順番」は、事前に定義（フレームの衝突は発生しない）

→ 送信タイミング（スロットの長さ）は、送信フレームの有無によって変わる

→ 高い応答性が求められるアプリケーションに適している

それぞれの特長を生かすことにより、一定の送信周期を保証する、もしくは迅速なレスポンスを実現するなど、アプリケーション志向の通信を柔軟に設計できます（FlexRay の「flex」は Flexibility＜柔軟性＞から由来しているといわれています）。

図 4

Fixed TDMA 通信の例：

・発車予定やその時間は事前に決まっている
・電車の長さや発車間隔は一定

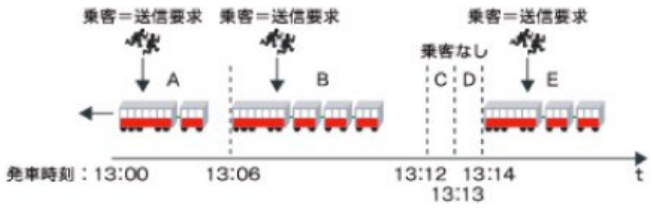

図 5

Flexible TDMA 通信の例：

・発車時間は乗客次第。順番のみ事前に決まっている
・電車の長さも一定ではない

1.4.2 ネットワークトポロジー

ネットワークトポロジー（構成）は「バス型」から「パッシブ・スター型」「アクティブ・スター型」および、それらの「混合（ハイブリッド）型」まで、多様な形が可能です。これにより、自由度の高い車載ネットワーク設計ができます。

バス型は、CAN などにも採用されている一般的なトポロジーです。コスト効率が良い半面、バスの長さは主に電気信号の反射と伝播遅延により制限され、ネットワーク上のある個所で発生した障害によりネットワーク全体が影響を受ける可能性が大きい、といった懸案もあります。

スター型には"パッシブ"と"アクティブ"の 2 種類があります。パッシブ・スター型は、基本的にバス型と同じ考え方です。一方、アクティブ・スター型は「スター・カプラー」と呼ばれるゲートウェイのような役割を行う装置に各ノードが接続されたトポロジーです。アクティブ・スター型では、バス長をより長くすることができるため、大きいネットワークを構築できます。また、ネットワーク上のある個所で障害が発生した場合のネットワーク全体に与える影響度も抑えることができるため、より安全重視のアプリケーション向きといえます。

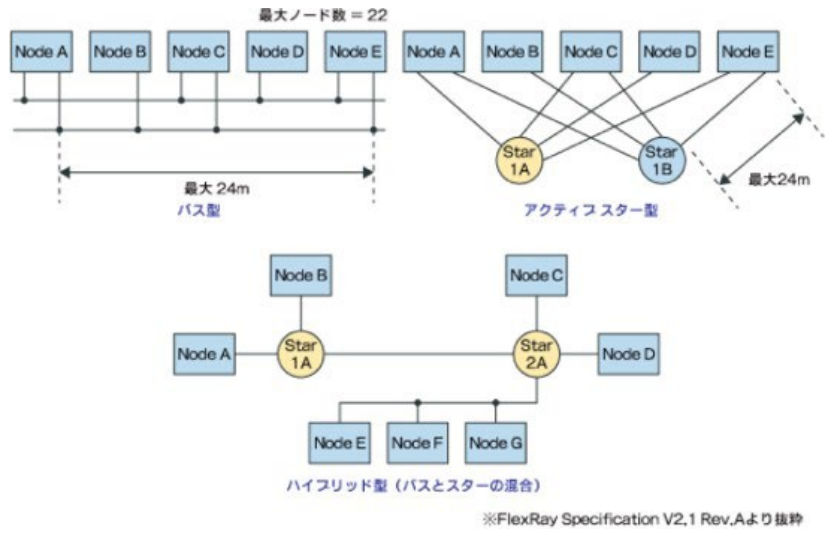

図 6 FlexRay のネットワークトポロジー

1.4.3 フォールトトレランス（障害耐性）

ネットワークの二重化（A チャンネル／B チャンネル）に対応することで、冗長性を持たせています。つまり、仮に一方のネットワーク線に断線などの障害が発生しても、残りのネットワーク系統により正常な通信が維持されることになります。ただし、この二重化は必須ではなく、要求される信頼性やコストに従って、ネットワークごと、ノードごとに柔軟に設計できます。

また、FlexRay では可能な限り通信を継続するコンセプトに基づいているため、エラーによって通信を中断する条件は通信プロトコルでは規定しておらず、アプリケーションで判断します。つまり、致命的なエラーを検出した場合も、通信コントローラが停止状態（halt）には遷移しますが、プロトコルが「勝手に」通信を中止させることはしません。その決定はアプリケーションで行われることになります。より安全性、信頼性が求められるシステムに適した仕様といえます。

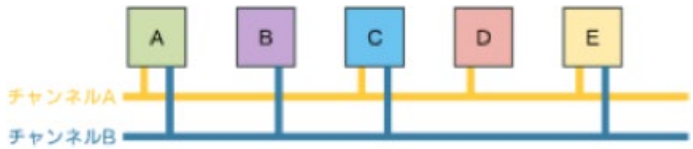

図 7（例）ネットワーク二重化：
ノード A、C、E は 2 つのチャンネルに接続。
ノード B はチャンネル B にのみ、ノード D はチャンネル A のみに接続

1.4.4 ノード構造

FlexRay ノードの構造は、複数の「論理ブロック」と呼ばれる機能体（以下）から構成され、それぞれが多岐にわたる情報を共有し、相互監視を行っています。

1.4.4.1 ホスト（マイクロコントローラ）：

アプリケーションの一部として、通信コントローラにアプリケーションの制御情報、設定情報、データを提供することや、バスドライバの動作モードを制御することを担当します。1 ノードに 1 つ装備されます。

1.4.4.2 通信コントローラ（CC）：

通信プロトコルの処理を担当します。ホストへシステム状態の情報、受信したデータを提供し、バスドライバに送信データや送信許可信号を提供します。1 ノードに 1 つ装備されます。

1.4.4.3 バスドライバ（BD）：

チャンネルへの物理的なアクセスを担当します。通信コントローラへ受信したデータを、ホストへエラー情報を提供します。1 チャンネルに 1 つ装備されます（ネットワーク二重化の場合は 1 ノードに 2 つ装備）。

1.4.4.4 バスガーディアン（BG）：

バスドライバによるネットワークへの送信を監視します。通信コントローラとは別のタイムスケジュールを持っており、常にバスドライバのネットワークへのアクセスをモニタします。そして、不適切なタイミングでの送信要求を検知した場合、バスドライバに対して送信を停止させることができます。1 チャンネルに 0～1 つ装備されます（オプション）。

これらにより、高い安全性、信頼性を実現します。

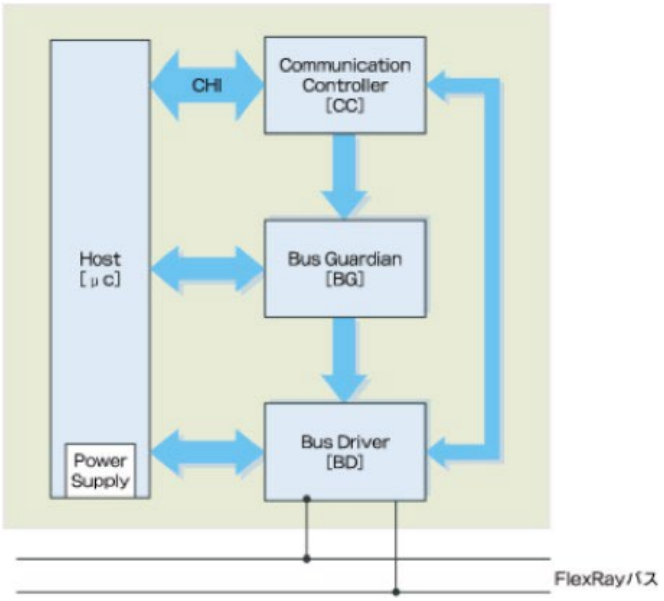

図 8 FlexRay ノード構造：各機能体の相互監視

1.4.5 優れた通信同期機能

「マルチマスタ・クロック同期」と呼ばれる、任意ノードのクロック（処理のタイミングを合わせるために用いられる信号）に基づく同期方法と、さまざまな補正機能を備えることによって、ネットワーク上のすべてのノードが同じタイミングで通信を行うことができます。

まず通信開始時に、タイミングに関して基準となる複数ノードから「グローバルタイム」が与えられます。このグローバルタイムとは、絶対的な時間ではなく、あくまでも「ネットワーク内における時間に関する共通の“ものさし”（具体的には、サイクルの長さと開始時間）」です。これを用いて各ノードがタイミングを合わせることで同期を図ります。また通常、温度変化、電圧変動、水晶発振器の製造誤差などにより、各ノードが持つ時間（ローカルタイム）は異なっており、通信開始時にタイミングを合わせたとしても徐々に差が出てくるため、通信開始後の補正が必要となります。FlexRay　では、各ノードが常にグローバルタイムとローカルタイムとの差を測定し、補正することで同期通信を実現しています。

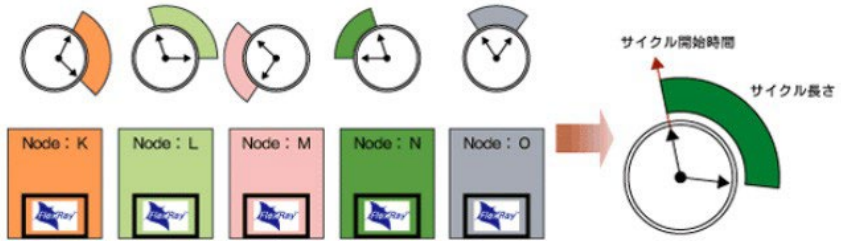

図 9 グローバルタイムとローカルタイム：共通のタイムベースに対してクロック同期を実施

1.4.6 高速通信

FlexRay の最大通信速度は 10Mbps であり、CAN（1Mbps）の 10 倍です。

2. FlexRay プロトコルの概要（その 1）

第 2 章では、FlexRay プロトコルのうち「通信構造／バスアクセス」「タイミング」「フレームフォーマット」「通信コントローラの状態遷移」の概要を説明します。

2.1 通信構造／バスアクセス

2.1.1 サイクル

FlexRay 通信は「サイクル」と呼ばれる、周期を繰り返す“時分割通信方式（TDMA）”です。このサイクルは FlexRay 通信で基本となる最も大きい単位です。その概要を以下に示します。

> サイクルの長さは、ネットワーク全体（FlexRay では通常「クラスタ」と呼びます）で共通の値として決められます。

> 1 つのサイクルは、以下 4 つのセグメントで構成されます。各セグメントの長さもクラスタで共通の値として定義されます。

- スタティックセグメント（必須）
- ダイナミックセグメント（オプション）
- シンボルウィンドウセグメント（オプション）
- NIT（Network Idle Time）セグメント（必須）

> 各ノードは、サイクルの繰り返す回数を「サイクルカウンタ」として 0 からカウントします。このカウンタ数はある回数でリセットされ、これを「サイクルカウンタワードラップ」といいます。プロトコルバージョン V2.1 におけるサイクルカウンタワードラップは定数＝64 と規定されており、サイクルカウンタは 0 を基点に 63 まで増加した後、0 に戻ることを繰り返します（0、1、2、……、62、63、0、1、2、……）。一方、プロトコルバージョン V3.0 以降では 8～64 間の各偶数値をサイクルカウンタワードラップとして任意に設定できます。

図 10 通信構造―「サイクル」

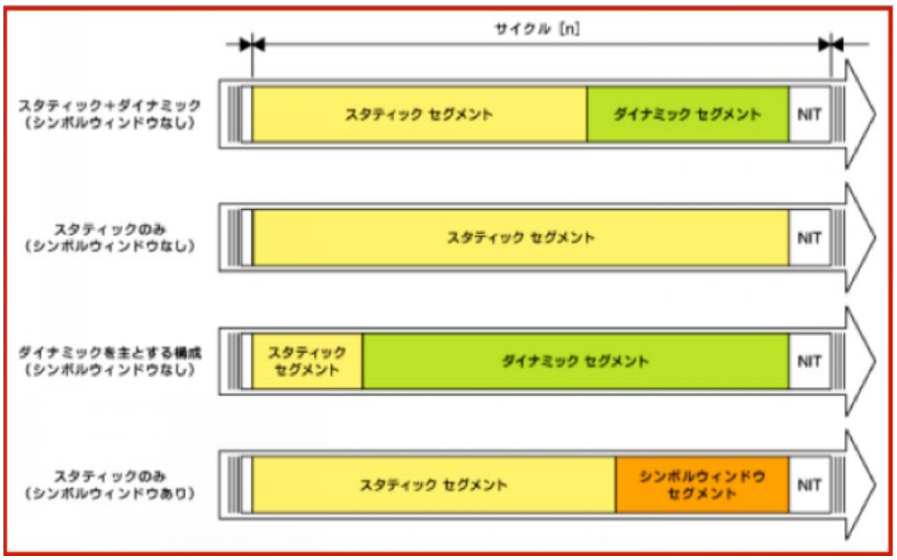

図 11 サイクル構成のバリエーション：通信構成を柔軟に設計できる

2.1.2 スタティックセグメント

Fixed TDMA 通信を行うための領域です。第 1 章でも紹介したように、決まったフレームを決まったスロットタイミングで送信するため、一定の送信周期を保証でき、安全性重視のアプリケーションに適しています。

各サイクルの最初に必ず設定されます。

- 「スロット」で分けられます（スタティックセグメントのスロットを「スタティックスロット」と呼びます）。1 つのスタティックスロットが経過するたびにスロットカウンタが 1 つずつ増えます（1、2、3、.....）。
- 各スタティックスロットの長さはクラスタ共通で、すべて同じ長さです。
- 各スタティックスロットに対して送信ノードが割り当てられます。プロトコルバージョン V2.1 では、1 スタティックスロットに 1 ノードを割り付けます。一方、プロトコルバージョン V3.0 以降では特定のスロット以外であれば 1 つのスタティックスロットに複数のノードを割り当てることが可能です。
- 送信タイミングがあらかじめ決まっているため、バスガーディアンによる保護が可能です。

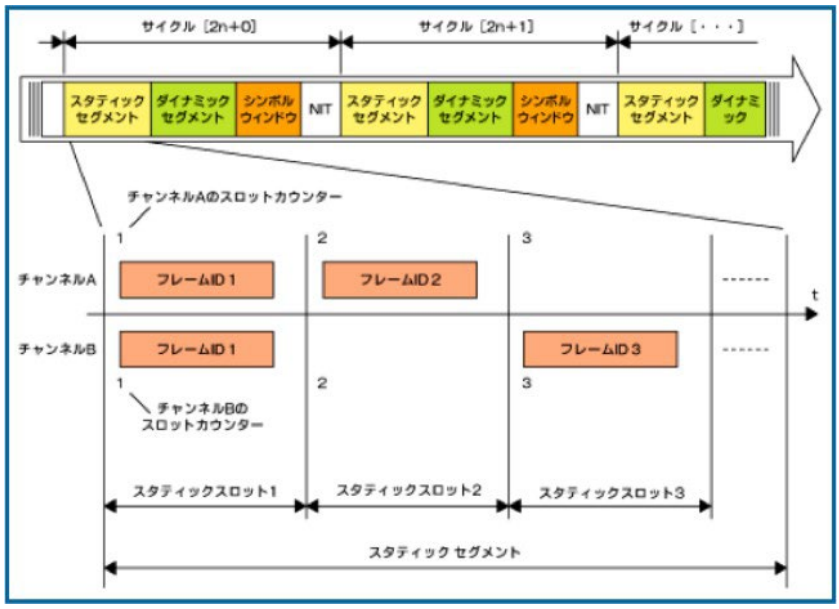

図 12 スタティックセグメントの構造

2.1.3 ダイナミックセグメント

Flexible TDMA 通信を行う領域です。送信の順番は事前に定義されますが、送信自体は要求に基づいて行われるため、高い応答性が求められるアプリケーションに適しています。

- スタティックセグメントの次に設定されますが、必須ではありません。
- 「ミニスロット」で細分化されます。1 つのミニスロットが経過するたびにスロットカウンタが 1 ずつ増えます（1、2、3、.....）。
- 送信要求があった場合のみ、複数のミニスロットで構成されるダイナミックスロットとなります。ダイナミックスロットの長さはフレーム長に応じて可変です。
- 複数の ECU から同タイミングで送信要求があった際も、事前に定義された送信順に従って送信されるため、フレームの衝突およびそれに伴う送信遅延などは発生しません。
- 各チャンネルの各ダイナミックスロットに対して、別々に送信フレームが割り当てられます。
- 送信要求がなかった場合、1 ミニスロット分が未使用のまま、次のミニスロットにスロットカウンタが進みます。ミニスロットがいわば「プレイスホルダ（場所取り）」の役割を果たします。
- 送信要求に応じて、2 つのチャンネル（A／B チャンネル）はそれぞれ独立してスロットカウンタを進めます。よって、同じタイミングに異なるスロットカウンタ値を取り得ます。
- 要求に基づいた送信であるため、バスガーディアンによる保護はありません。

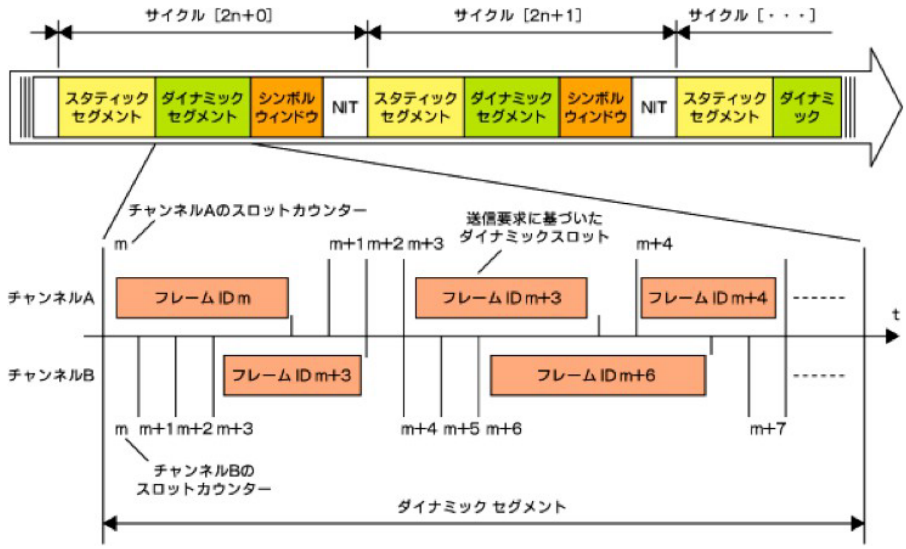

図 13 ダイナミックセグメントの構造

2.1.4 シンボルウィンドウセグメント

ダイナミックセグメントの次に設定されますが、必須ではありません。FlexRay プロトコルにて「シンボル」と呼んでいる特定のビットパターンを送信する領域です。

プロトコルバージョン V2.1 では、送信できるシンボルは「MTS（Media Access Test Symbol）」のみです。この MTS は各ノードのバスガーディアンが正常かどうかをチェックするために使用されます。プロトコルバージョン V3.0 以降では MTS に加えて「WUDOP（Wakeup During Operation Pattern）」と呼ばれるシンボルも送信できます。

2.1.5 NIT（Network Idle Time）セグメント

各サイクルの最後には、必ず「NIT セグメント」が設定されます。同期に必要な各パラメータの計算を実施する領域、かつ同期のための補正を行うためのいわゆる“のりしろ”となる領域であるため、データの送信などは一切ありません。ネットワーク上ではアイドル時間となります。

2.1.6 サイクルマルチプレキシング

前述のスタティック／ダイナミックセグメントに関する図（図 12、13）では、通信例として 1 つのサイクルのみを示しましたが、各サイクルに異なる通信スケジュールを定義することができます。これによりセグメントを有効に活用できます。これを「サイクルマルチプレキシング」と呼びます。

これは、各フレームに対して“何サイクルに 1 回送信するか”を示す「サイクルカウンタフィルター」を設定することにより実現されます。例えば、あるフレームがサイクルカウンタフィルター＝2 によってスロット 3 に設定され、最初のサイクル（＝サイクル 0）から送信される場合、このフレームが送信されるタイミングはサイクル 0、2、4、6、……でのスロット 3 となります。サイクル 1、3、5、7、……でのスロット 3 には別のフレームを定義できます。

図 14 サイクルマルチプレキシングの例:

フレーム c がサイクルカウンタフィルター＝2 によってスロット 3 に設定され、最初のサイクル 0 から送信されるため、その送信タイミングはサイクル 0、2、4、6、……となる。同じスロットのサイクル 1、3、5、7、……では別フレームが送信できる

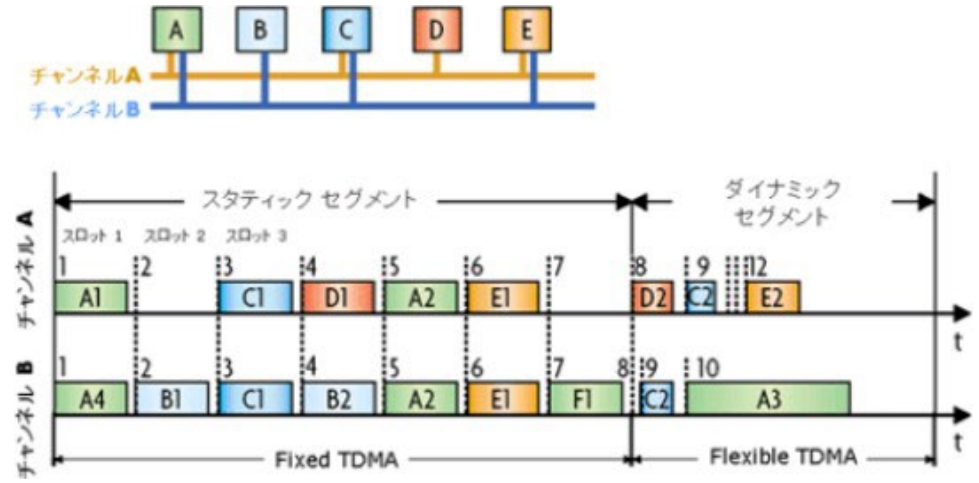

図 15 FlexRay 通信の例

2.2 タイミング

FlexRay はタイムトリガー通信方式であり、この方式で重要なのは各ノード共通の“時間軸 — タイムベース”です。FlexRay ではクラスタ内の時間における共通の“物差し”である「グローバルタイム」が与えられ、このグローバルタイムと各ノードが持つ固有の「ローカルタイム」の差を測定・補正することで同期通信を実現しています。

これらタイムベースは、先に紹介したサイクルのほかに、「マクロティック（Macrotick）」「マイクロティック（Microtick）」といった 3 つの“単位”に基づいています。それらの関係は図 16 のようになります。

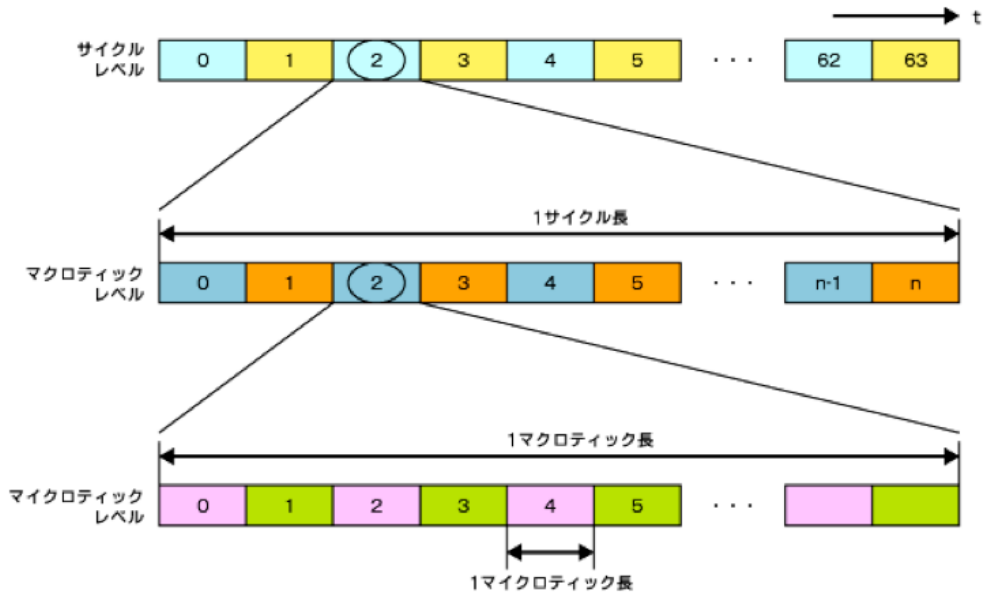

図 16 タイミングの階層

2.2.1 サイクル

FlexRay 通信における最も大きな単位です。偶数のマクロティックから成り立っています。

2.2.2 マクロティック（Macrotick）

「グローバルタイム 」の最小単位であり、クラスタ全体で共通の値です。複数のマイクロティックから構成されます。

2.2.3 マイクロティック（Microtick）

サンプルクロックから得られる「ローカルタイム」の最小単位です。サンプルクロックは発振器の周波数などノード固有の要因によって決まるため、当然、このマイクロティックの長さも各ノードで異なってきます。

2.3 フレームフォーマット

次に、これまで紹介してきたスロットで送信されるフレームの具体的な構造や主な種類を紹介します。

フレームは「ヘッダー（Header）」「ペイロード（Payload）」「トレーラー（Trailer）」の 3 つのセグメントから構成されており、この構造はスタティックセグメントで送信されるフレーム（＝スタティックフレーム）とダイナミックセグメントにて送信されるフレーム（＝ダイナミックフレーム）で共通です。

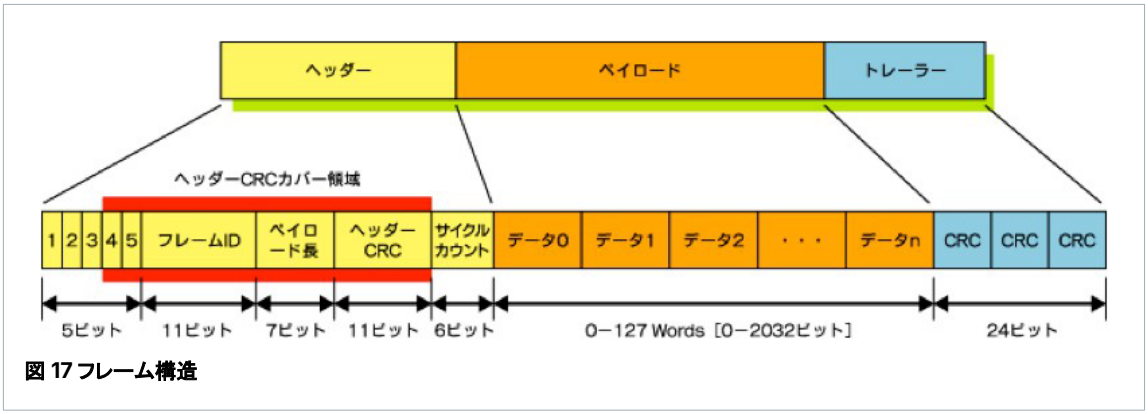

図 17 フレーム構造

それぞれのセグメントに格納されるデータは以下になります。

- ヘッダー（40 ビット）
- リザーブ（Reserved）（1 ビット）：今後の拡張のための予約ビットです。通常「0」が設定されます。
- ペイロードプリアンブルインジケーター（Payload Preamble Indicator）（1 ビット）：ペイロードの一部を特別な目的に使用するかどうかを示します。
- Null フレームインジケーター（Null Frame Indicator）（1 ビット）：フレームが Null フレームかどうかを示します。
- 同期フレームインジケーター（Sync Frame Indicator）（1 ビット）：フレームが同期フレームかどうかを示します。
- スタートアップフレームインジケーター（Startup Frame Indicator）（1 ビット）：フレームがスタートアップフレームかどうかを示します。
- フレーム ID（Identifier）（11 ビット）：フレーム ID を定義します。取り得る値は 1～2047。0 は無効です。
- ペイロード長（Payload Length）（7 ビット）：ペイロード部分の長さを単位「Word」で示します（※ 1Word＝バイト）。
- ヘッダーCRC（Header CRC）（11 ビット）：ヘッダー中の同期フレームインジケーター、スタートアップフレームインジケーター、フレーム ID、ペイロード長部分を保護する CRC（Cyclic Redundancy Check／巡回冗長検査）コードです。
- サイクルカウント（Cycle Count）（6 ビット）：サイクルカウンタ数を示します。取り得る値は 0～63 です。
- ペイロード（0～2032 ビット）
- データが格納される部分です。最大 127Words（＝254 バイト）のデータを定義できます。
- トレーラー（24 ビット）
- フレーム CRC（Frame CRC）：ヘッダー全体とペイロードを保護する CRC です。

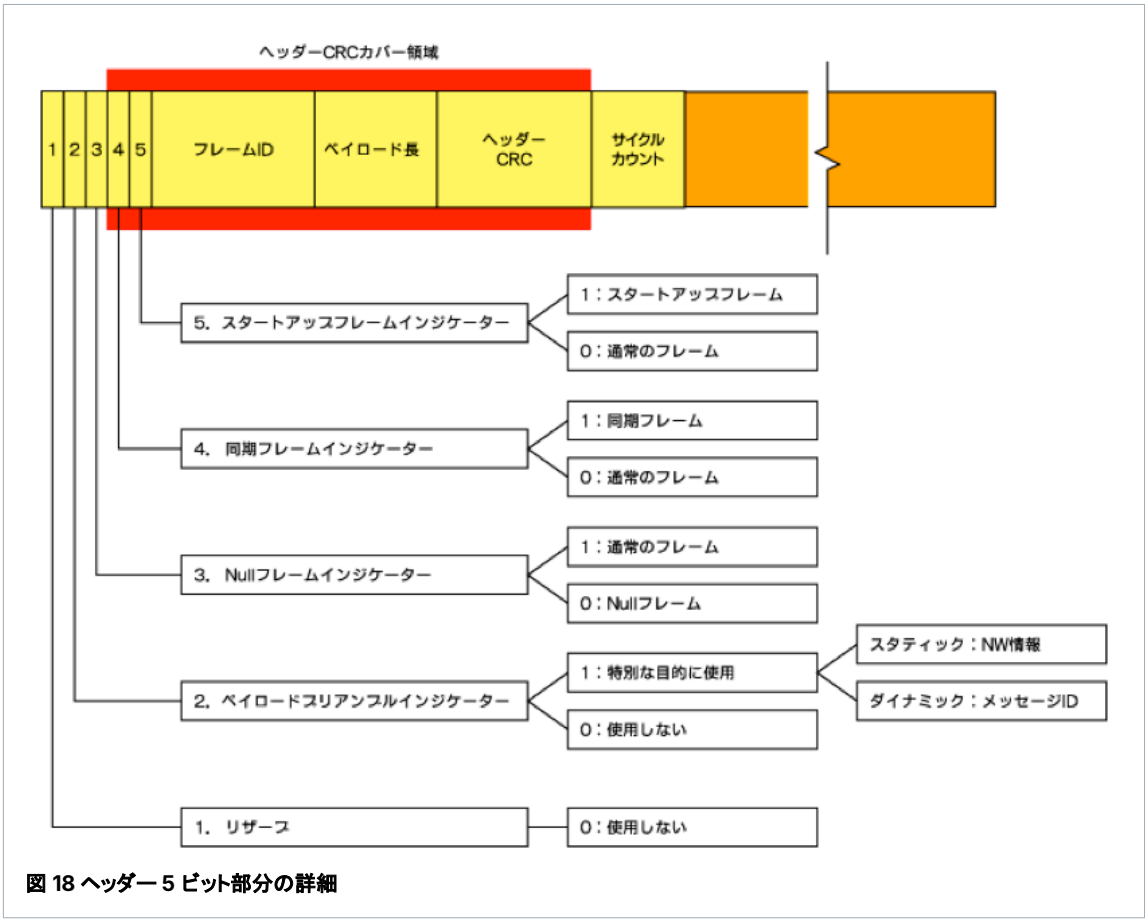

図 18 ヘッダー 5 ビット部分の詳細

2.3.1 ペイロードプリアンブルインジケーター

ペイロードの一部を特別な目的に使用するかどうかを示すビットとなり、その用途はスタティックフレームとダイナミックフレームで異なります。

スタティックフレームではペイロードセグメントの最初の 0～12 バイトにネットワークマネジメントの情報を、ダイナミックフレームではペイロードセグメントの最初の 2 バイトにメッセージ ID を定義できます。

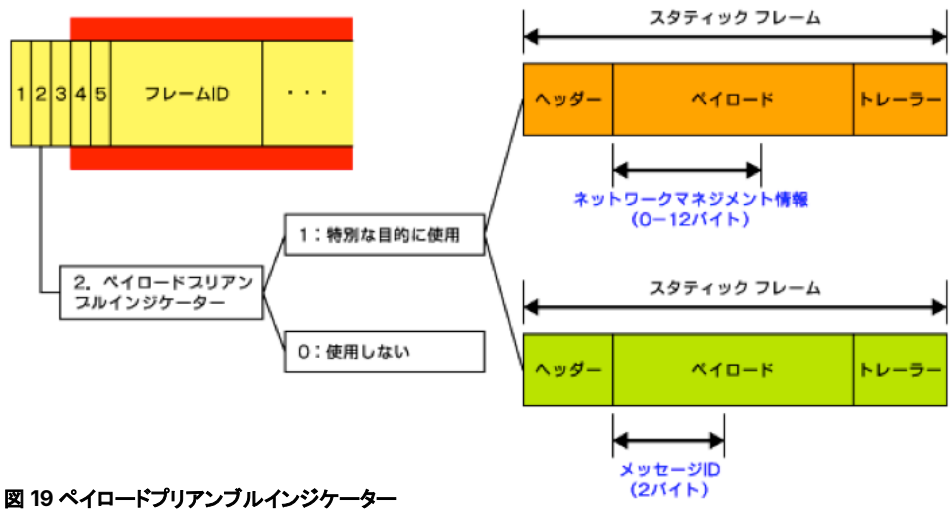

図 19 ペイロードプリアンブルインジケーター

2.3.2 Null フレーム

ペイロードに使用できるデータを載せていないフレーム、つまり“データが使われない”フレームです。フレームのヘッダーセグメントの Null フレームインジケーターが「0」のとき、そのフレームは Null フレームとなり、ペイロードデータはすべて「0」となります。このフレームは、例えばサイクルマルチプレキシングの未使用スロットでの送信など、送るべきデータがないとき、またはコントローラでのフィルター条件に合致した場合、スタティックセグメントでのみ送信されます。

2.3.3 同期フレーム

同期補正を実行する際に使用されるフレームです。

2.3.4 スタートアップフレーム

クラスタ全体の通信開始時に、同期を図るために使用するフレームです。

2.4 通信コントローラの状態遷移（POC：Protocol Operation Control）

第 1 章で紹介したように、FlexRay ノードにはホスト、通信コントローラ、バスドライバといった複数の機能体が搭載されており、その中の通信コントローラにはプロトコルを制御する「プロトコルオペレーションコントロール（POC）」というメカニズムが備わっています。ここでは、この POC の状態がどのように遷移するのかを簡単に説明します。

各プロセスと POC との関係は次の図 20 で表せます（POC とアプリケーションの一部であるホストとの通信を行うインターフェイスをコントローラホストインターフェイス（CHI）といいます）。

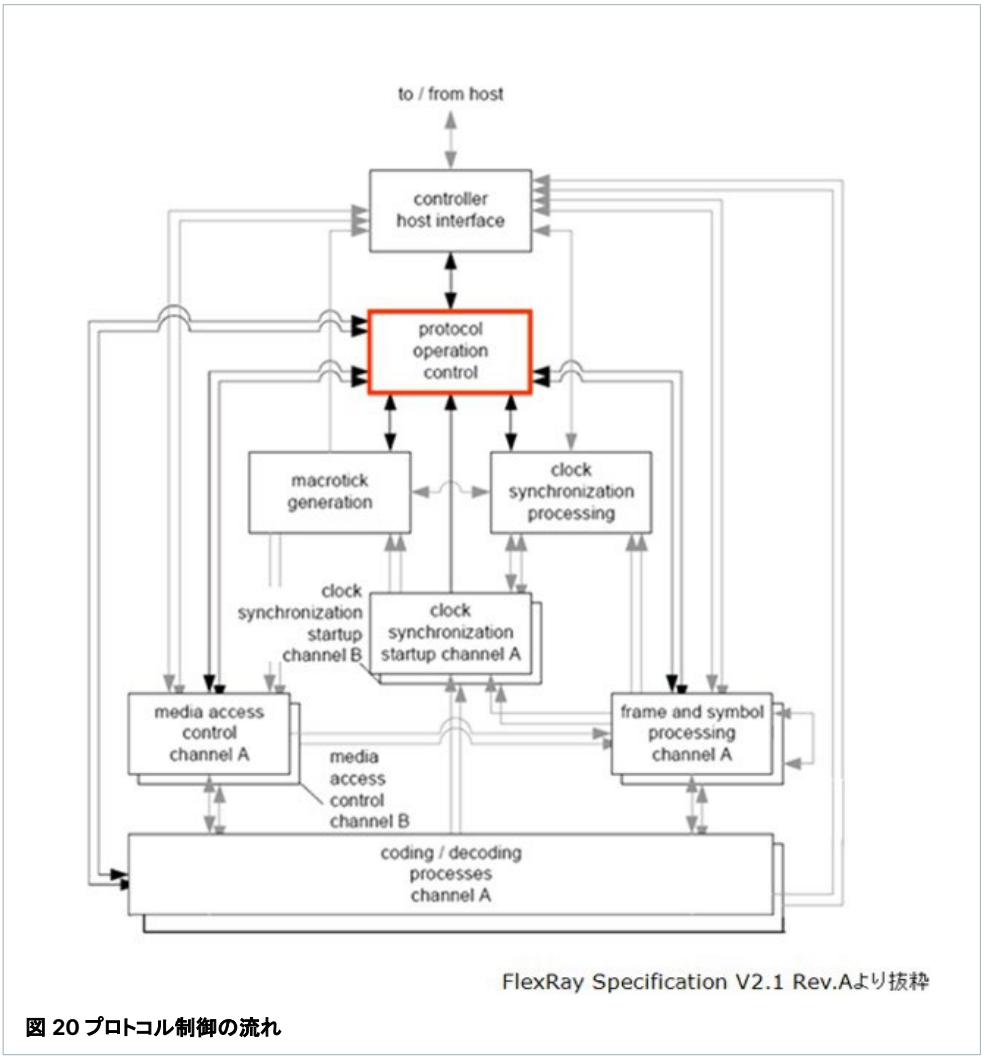

図 20 プロトコル制御の流れ

この POC が取り得る状態や遷移条件などの関連性を示した図が図 21 になります。通常、これらの状態はホストから CHI を経由して変更されますが、図 21 の（*）部分は POC 自身によって、（**）部分は POC または CHI によって変更されます。

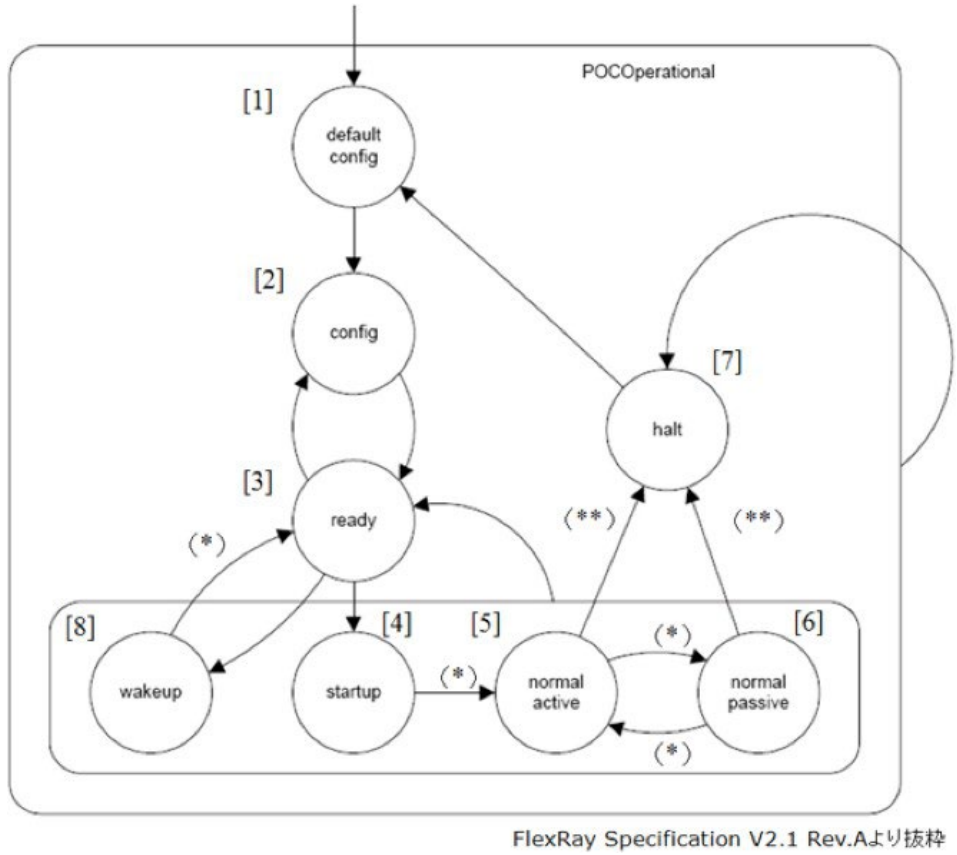

図 21 POC の状態遷移

[1]初期コンフィグ（Default Config）：通信コントローラの初期状態です。

[2]コンフィグ（Config）：ホストによって通信に必要なすべてのパラメータが設定された状態です。

[3]レディ（Ready）：通信開始に向けた待機状態です。

[4]スタートアップ（Startup）：通信の準備を行う状態です。

[5]ノーマルアクティブ（Normal Active）：通常の通信を実行する状態です。

[6]ノーマルパッシブ（Normal Passive）：何らかのエラーが発生したときの一時的なエラー状態です。その後、問題がなければノーマルアクティブに戻ります。

[7]ホールト（Halt）：致命的なエラーが発生した場合に遷移する状態です。

[8]ウェイクアップ（Wakeup）：ウェイクアップ状態です。

ここで懸案となるのが“致命的なエラーが発生した状態”であるホールト（halt）に遷移する条件です。FlexRay ではプロトコルの判断により通信を中止することはなく、エラーの種類やノーマルアクティブからノーマルパッシブに遷移する条件、ホールトに遷移する条件などはあくまでもアプリケーションで行われます。POC は実際のエラー処理を行います。

3. FlexRay プロトコルの概要（その 2）

第 3 章では、FlexRay プロトコルの他の項目、「エンコーディング／デコーディング」「同期方法」「スタートアップ」について説明します。

3.1 エンコーディング／デコーディング

フレームやシンボルといった通信データをノードが送信する際に、どのようにビット列にエンコーディング（符号化）され、また受信ノードにてどのようにデコーディング（復号）されて、信頼性の高い、確実な通信が実現されているのかを紹介します。

3.1.1 エンコーディング

下の図 22 は、ダイナミックフレームをエンコーディングした際のビット列になります。「DTS」という部分以外はスタティックフレームも同じ構造を持っています。

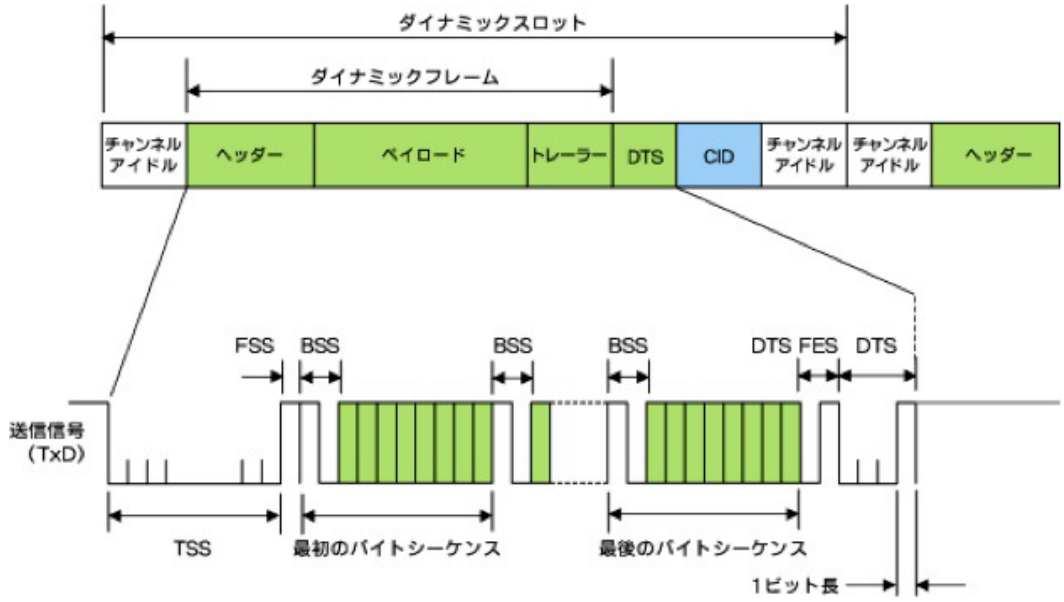

図 22 フレームのエンコーディング

※CID（Channel Idle Delimiter）：連続した High ビット（11 ビット）。フレームの後、次のチャンネルアイドルまでの「マージン」部分となる

- TSS（Transmission Start Sequence）：

  フレームの先頭を示す連続した Low ビット（3～15 ビット）。アクティブ・スターや受信ノードにて、この TSS を短くする（切り捨てる）ことによって、バスドライバの処理などで発生する遅延時間をできるだけ小さくする

- FSS（Frame Start Sequence）：

  TSS の終了を示す 1 ビット－High（論理'1'）

> BSS（Byte Start Sequence）:

- 8 ビットのデータが続くことを示す 2 ビットーHigh／Low（論理'10'）。受信ノードでは、この'10'の立ち下がりエッジを利用してタイミングの補正を行う
- 通信データの各バイトは、この BSS に続く 8 ビットで送信される。BSS（2 ビット）とそれに続く 8 ビットの計 10 ビットがバイトエンコーディングの基本的な単位となる

> FES（Frame End Sequence）:

フレームの最後を示す 2 ビットーLow／High（ 論理'01'）

> DTS（Dynamic Trailing Sequence）:

ダイナミックフレームの FES 直後に付加される。スタティックフレームでは、このシーケンスは付加されない。可変の Low／High で Low の長さでフレーム長を調整する

3.1.2 デコーディング

大きく 2 つのプロセスに分けられます。

① 受信信号サンプリング／ノイズフィルター実施

② ビットクロック調整／ビット値の決定

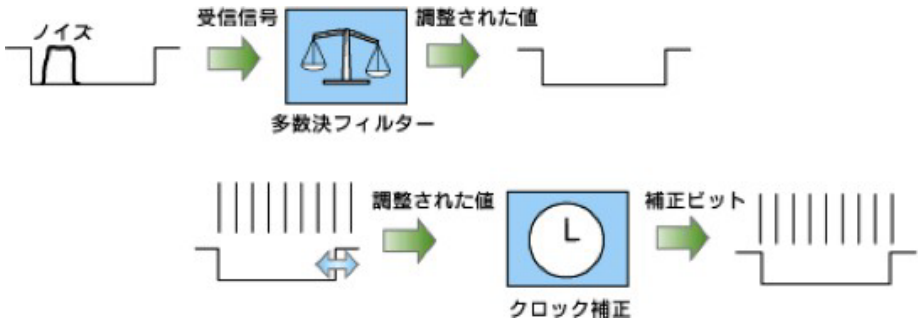

図 23 デコーディングの概観:
サンプリング→多数決フィルターで値を調整→ビットクロックを調整→ビット値を決定

① 受信信号サンプリング／ノイズフィルター実施

まず通信コントローラが受け取った受信（RxD）信号に対して、各ノードのサンプルクロックを基に 1 ビット当たり 8 個のサンプルを取ります。FlexRay でのバス信号はノイズを低減するための差動信号になっていますので、このままでは大きなノイズまで取得してしまいます。そのため、“多数決（Majority Voting）”フィルターという処理を実施し、ノイズを取り除きます。

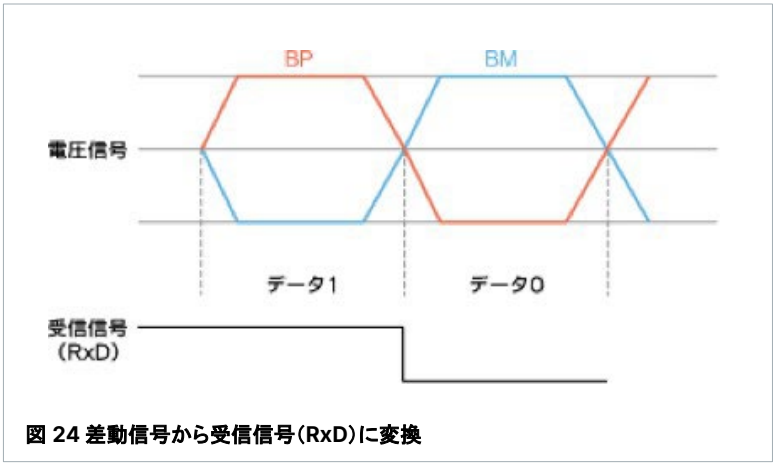

図 24 差動信号から受信信号（RxD）に変換

※FlexRay の差動信号はバス・プラス（BP）とバス・マイナス（BM）と呼ばれる 2 つの信号線間で測定される

- サンプルクロックごとに多数決フィルター用のサンプルを 5 つ取る。サンプルクロックの立ち上がりエッジ（図の赤丸部分）の値をサンプル値として取得し、FIFO（ファーストインファーストアウト）で保持する
- 5 つのサンプル中で‘0’と‘1’、多い方を「フィルター後の値」として採用する。これによりノイズを除去する
- 受信信号が変化に合わせてフィルター後の値も変化するが、2 サンプルクロック分の遅延を伴う

② ビットクロック調整／ビット値の決定

多数決フィルターにて値を調整した後、今度はビットレベルのタイミングについて調整を行い、最終的なビット値を決めます。

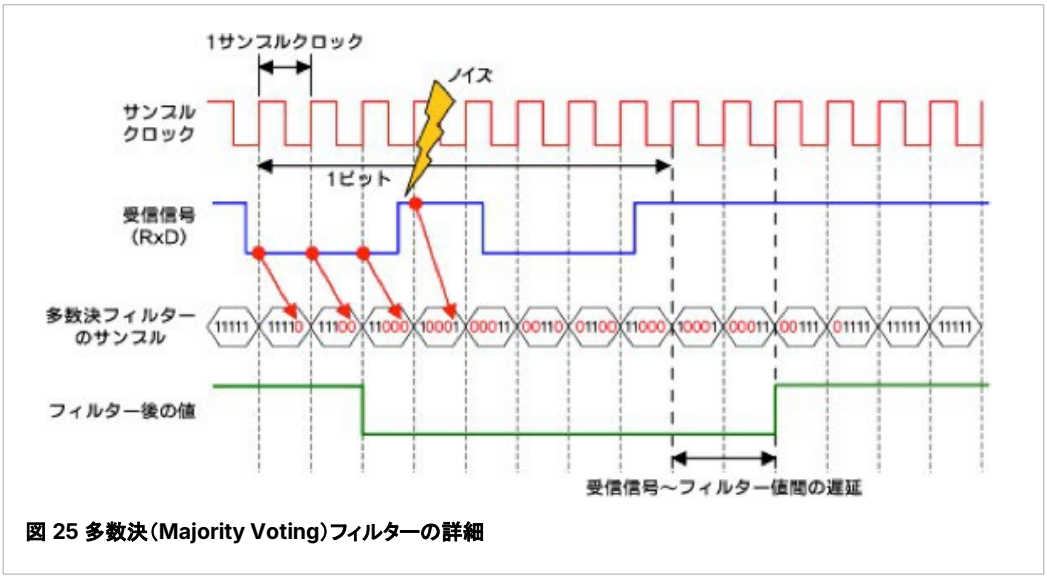

図 25 多数決（Majority Voting）フィルターの詳細

ビットクロック調整はビット列の BSS 部分など、決まったタイミングで必ず発生する立ち下がりエッジ(1→0)を基準として行います。各ノードでは、サンプルクロックを基に 1 ビット当たり 8 つのサンプルをサンプルカウンタでカウントしており、立ち下がりエッジを検出してから次のクロックでサンプルカウンタを"2"にリセットします。これにより、立ち下がりエッジを検出したクロックでカウンタが"1"にリセットされる場合と同等の動作となり、タイミングが修正されます。この後、サンプルカウンタが"5"のときの"多数決フィルター後の値"をビット値として採用します。

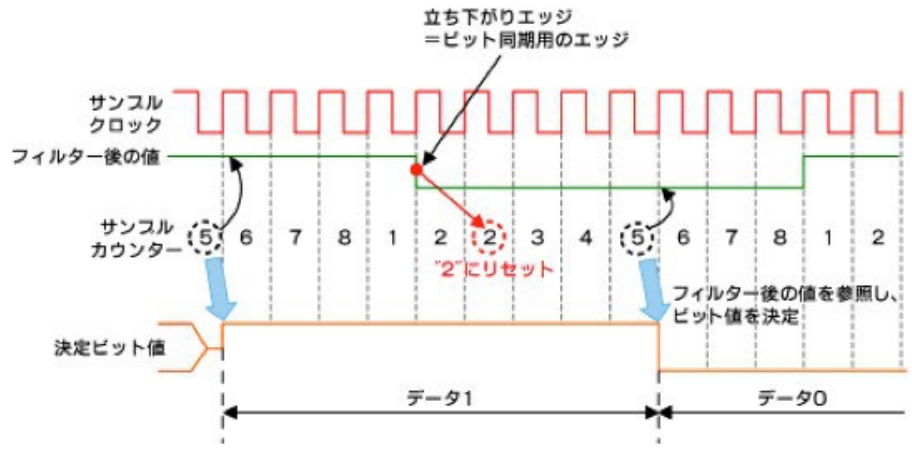

図 26 ビットクロックの調整／ビット値の決定の例

> ビット同期用のエッジを検出してから次のクロックを"2"にリセットして調整
> サンプルカウンタが"5"のときの多数決フィルターで調整された値を確認
> それを参照してビット値を決定

3.2 同期方法

高い信頼性を実現し、統合的な協調制御に高レベルで適応するため、FlexRay はグローバルタイムの導入や幾つかの補正方法を装備し、優れた通信同期機能を実現しています。その具体的な方法を紹介します。FlexRay では冗長による保全性を確保するため、同期タイミングについて 1 つのノードに依存せず、「同期ノード」と呼ばれる複数のノードのタイミングに、同期ノードではない他ノードが同期します。このタイミングがクラスタにおける「グローバルタイム」となるのです。また、ここでの“タイミングを合わせる”とは、具体的には各ノードで「共通のサイクル開始時間」「共通のサイクル長さ」を持つことになります。

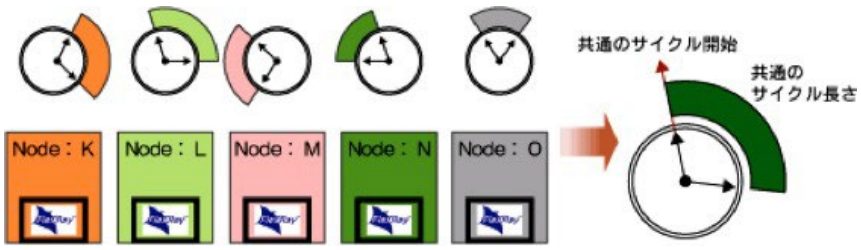

図 27 グローバルタイムとローカルタイム

各ノードがサイクル開始時間とサイクル長さを合わせるために使うのが「同期フレーム」であり、この同期フレームを送信するノードが「同期ノード」です。

同期ノードの主な特性:

> 1 つのクラスタに定義できる同期ノードの数：2～15
> 各同期ノードは、サイクルごとに決められた 1 つのスタティックスロットで 1 つの同期フレームを送信。つまり、同期フレームは常にスタティックフレーム
> 2 チャンネル（A／B チャンネル）を装備したシステムであれば、両チャンネルの同じスロットにて同期フレームを送信

3.2.1 同期プロセス

まず、各ノードではローカルクロックとマイクロティックカウンタ、マクロティックカウンタ、サイクルカウンタ、スロットカウンタといった各カウンタ値により、自身のタイミング＝ローカルタイムを知ることができます。また、同期フレームは送信タイミングが決まっているスタティックセグメントにて送信されるフレームであるため、その送信タイミングは予測することができ、これらを使ってローカルタイムにおけるサイクル開始時間やサイクル長を認識できます。

それに対して、同期ノードが実際に送信するすべての同期フレームのサイクル開始時間とサイクル長を測定し、これとローカルタイムを比べることにより自身の「ずれ」が分かります。そして、これから補正すべき値を計算し、補正を実施します。

補正の方法は以下の 2 通りがあります。

> レート補正：サイクルの長さを合わせる
> オフセット補正：サイクル開始時間を合わせる

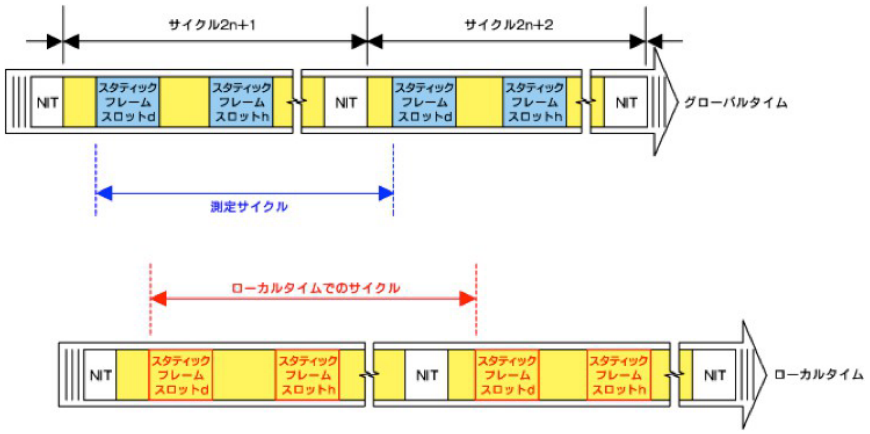

図 28 同期プロセス－2 つのタイムベース

3.2.2 レート補正

レート補正は他ノードとサイクルの長さを合わせる補正です。各ノードは、予測される同期フレームの送信間隔やマクロティックカウンタから自身のサイクル長を把握します。一方、同期フレームは、各サイクルの同じスロットで送信されるため、各同期フレーム到着時間の間隔を測定することで、同期ノードのサイクル長を取得できます。

例えば、図 29 にて、2n＋1 サイクルにおけるスタティックスロット d で送信される同期フレーム d と、次のサイクル（2n＋2 サイクル）におけるスタティックスロット d で送信される同期フレーム d の到着時間の間隔を測定することによって、同期フレーム d を送信する同期ノードについてのサイクル長が測定できます。同様な測定を同期フレーム h やその他のすべての同期フレームに対して、スロットごとに行います。同期フレームの数、つまり同期ノードの数が多ければ多いほど、測定するサイクル長も多くなり、精度が向上します。

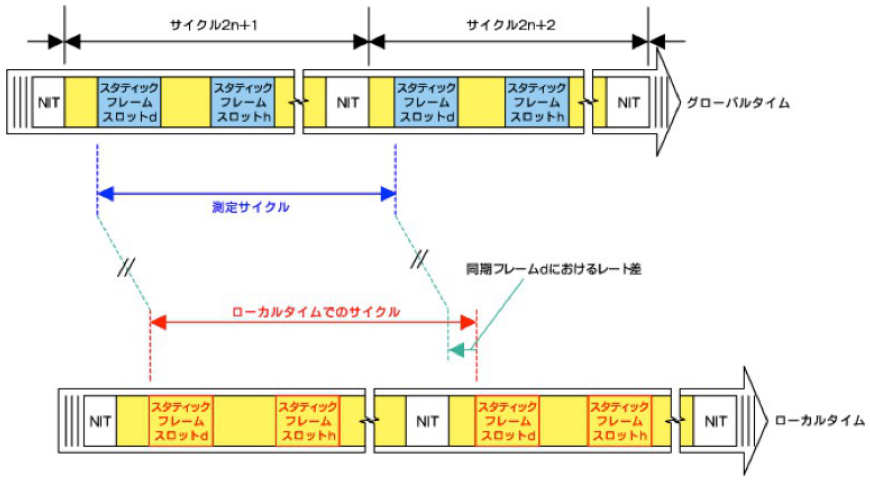

図 29 レート補正－測定例

これら測定値との差分を基に、ノード自身がサイクル長に関する補正値を計算します。その計算は NIT セグメントにて、2 サイクルに 1 回行われます。これは先の同期フレーム d の例の通り、同期フレームの間隔を測定するには 2 サイクルが必要だからです。その後、計算した補正値に基づいて補正を実施します。これはサイクルを通じて行います。具体的には、1 マクロティック当たりのマイクロティック数を調整することでサイクル長を調整します。

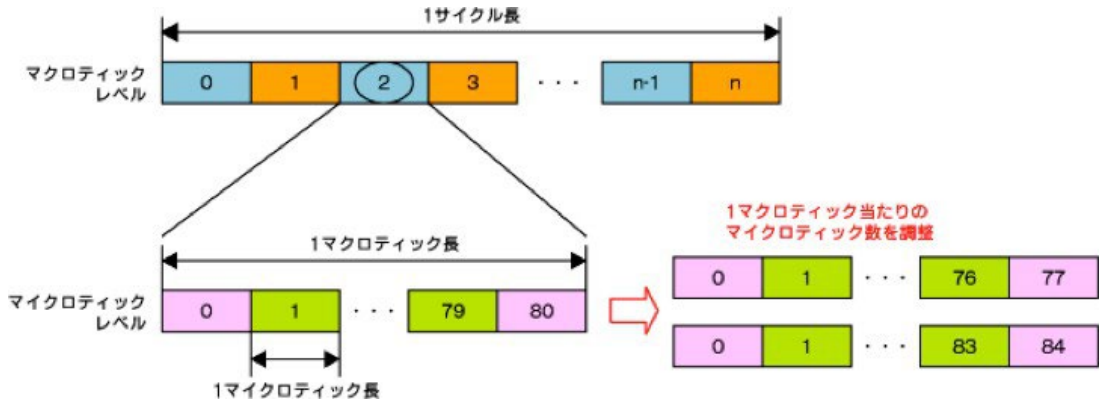

図 30 レート補正の実施方法イメージ

3.2.3オフセット補正

オフセット補正は、他ノードとサイクルの開始時間を合わせる補正です。予測した同期フレームの到着時間と実際の同期フレームの到着時間の差＝“オフセット”を測定し、ノードが自身のサイクル開始時間の偏差を把握します。

例えば図 31 にて、スタティックスロット d で送信される同期フレーム d の実際の到着時間と予測した到着時間の差が、つまり同期フレーム d を送信するノードとのサイクル開始時間差になります。レート補正と同様、他のすべての同期フレームについてもスロットごとにオフセット値を測定するため、同期フレームの数が多いほど精度が安定します。

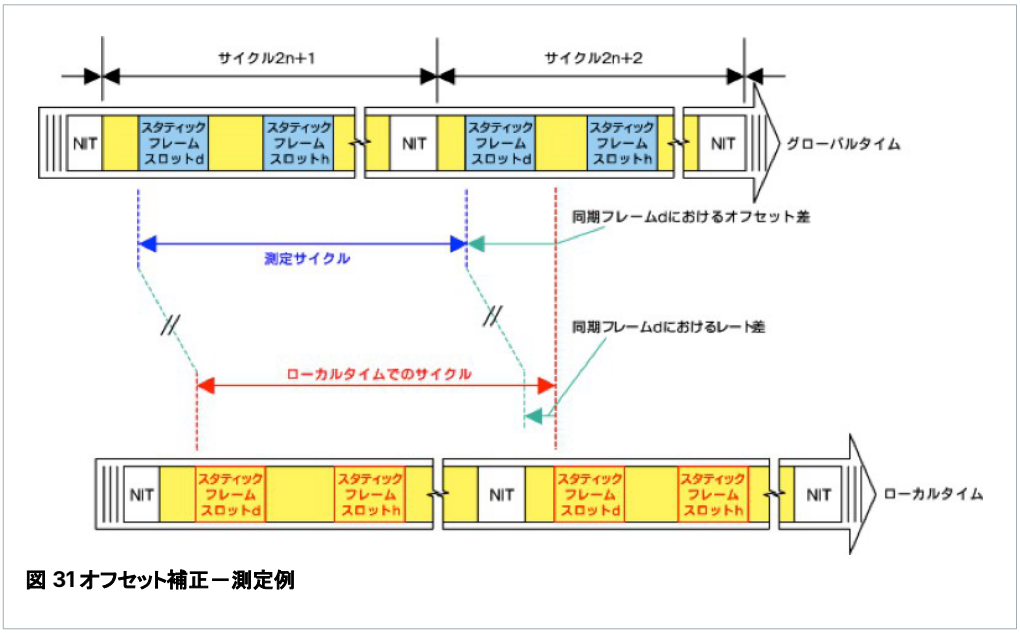

図 31 オフセット補正－測定例

測定したオフセットに基づいてその補正値を計算しますが、これをサイクルごとの NIT セグメントにて行います。補正の実施は 2 サイクルに 1 回、奇数サイクルのときに NIT セグメントを短縮または延長することによって行います。

図 32 オフセット補正の実施方法イメージ

3.2.4 補正値の決定

では具体的に、測定結果からどのように補正値を決めるのでしょうか？レート補正とオフセット補正、それぞれに対して同期ノードの数だけ測定結果が得られた後、各補正値は FTM（Fault Tolerant Midpoint）アルゴリズムという方法を使用して決定されます。

表 1 の例を基に説明します。

例えば、8 つの同期ノードが存在するクラスタ内の非同期ノード X におけるレート補正値が表 1 の通りだとします。各同期ノードが同期フレームを送信するスタティックスロットはそれぞれ d、h、m、o、r、u、y、z です。

スロット	レート補正 測定値 （単位：マイクロティック）
d	−9
h	1
m	−8
o	2
r	12
u	6
y	−9
z	−2

表 1 ノード X におけるレート補正値

また FlexRay プロトコルでは“パラメータ k”という値がクラスタ内の同期ノードの数によって、表 2 のように規定されています。このパラメータは、補正値を計算する際、すべての測定値のうちの最大値と最小値からこの値の数だけ除外することを意味します。つまり、補正値の算出に極端に大きい／小さい値は使わないことで、より安定した補正値が得られます。

同期ノード数	パラメータk
1～2	0
3～7	1
7以上	2

表 2 同期ノードの数とパラメータ k

上記の例の場合、同期ノード数は「8」であるため、パラメータ k＝2 となります。

これらの測定値の例から FTM アルゴリズムを使って、以下のように実際の補正値を決めます。

1. 各測定値を降順（大きい順）で並べ替える
2. 測定値のうち、一番大きい値と一番小さい値からパラメータ k の数（この場合は 2）だけ破棄する

3. 残りの値を合計し、2 で割る
4. その結果が補正値となる

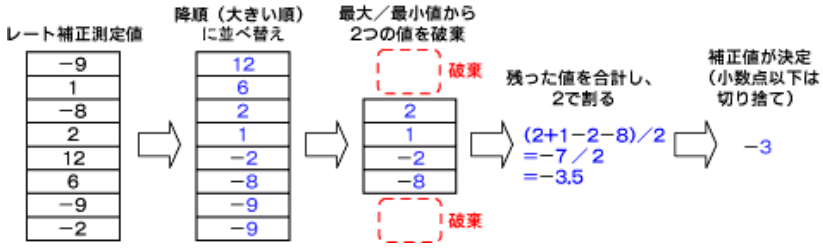

図 33 FTM アルゴリズムによる補正値の算出例

3.2.5 補正のまとめ

レート／オフセット補正の測定、計算、実施タイミングをまとめると以下のようになります。

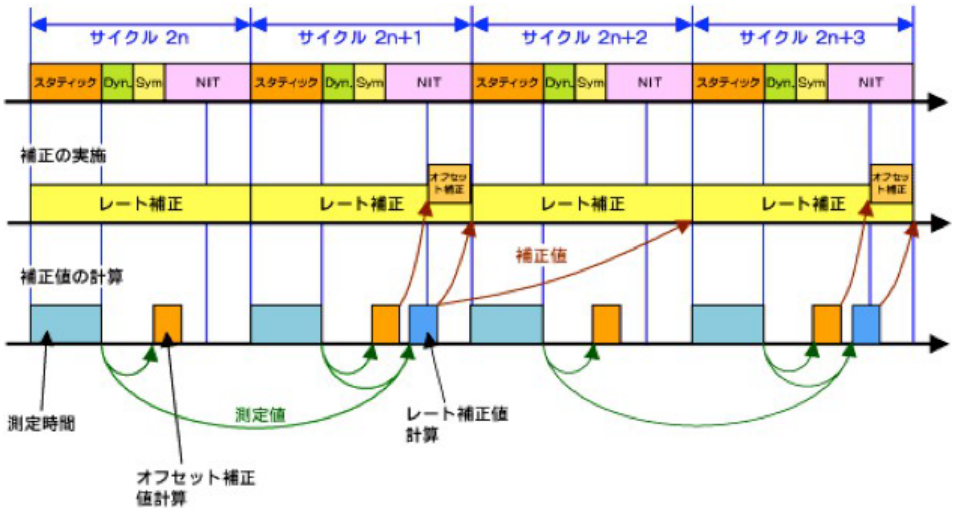

図 34 同期方法の概観図（※FlexRay Specification V2.1 Rev.A から引用／編集）

> 測定時間：

レート補正、オフセット補正、ともにスタティックセグメントにて同期フレームを測定

> レート補正：

- 補正値は 2 サイクルに 1 回、奇数サイクルにて計算される
- 補正は補正値を計算した奇数サイクルの次の偶数サイクル、およびその次の奇数サイクルにおいて、サイクル中に実行される

> オフセット補正：
  - 補正値はサイクルごとに計算される
  - 補正は奇数サイクルの NIT セグメントにて実行される

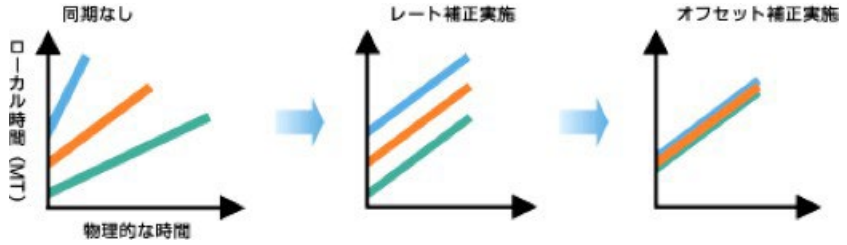

図 35 補正実施のイメージ

※オレンジ色の棒線がグローバルタイムでの 1 サイクルレート補正でサイクル長（＝傾き）を合わせ、オフセット補正でサイクルの開始点を合わせる

3.3 スタートアップ

第 2 章の「POC（Protocol Operation Control）の状態遷移」でも示した通り、通信開始に向けた準備段階、通信開始までのプロセスを FlexRay では“スタートアップ”と呼びます。このスタートアップにて各ノードがどのようにタイミングを合わせて通信を開始するのかを説明します。

通信中における“同期方法”では複数の同期ノードが基準となって同期を図っていましたが、スタートアップ時でも“コールドスタートノード（Coldstart Node）”という複数の特別なノードが中心になって同期を図ります。またコールドスタートノードが送信し、送信開始に向けたタイミング合わせのために使用されるフレームを“スタートアップフレーム”といいます。コールドスタートノードは必ず同期ノードであり、スタートアップフレームは必ず同期フレームです。

主な特性を以下に示します。

> スタートアップのためには、1 つのクラスタに少なくても 2 つのコールドスタートノードが必要

> リーディングコールドスタートノード（Leading Coldstart Node）：最初に起動するコールドスタートノード

> フォローイングコールドスタートノード（Following Coldstart Node）：リーディングコールドスタートノードに同期を取るコールドスタートノード

> これらリーディングコールドスタートノードやフォローイングコールドスタートノードとなるノードは事前に定義されず、実行時に決まる

> 2 チャンネル（A／B チャンネル）を装備したシステムの場合、スタートアップは両チャンネル同時に実施される

大まかなスタートアップ手順は、以下の通りです。

① 複数のコールドスタートノードのうちの 1 つが“CAS（Collision Avoidance Symbol）”と呼ばれるシンボル – いわゆる“合図”を送ります。これがリーディングコールドスタートノードです。

② その後、リーディングコールドスタートノードは最初のスタートアップフレームを送信します。その後、計 4 サイクルにわたって、スタートアップフレームを送信し続けます。

③ 1 つ以上のフォローイングコールドスタートノードは、リーディングコールドスタートノードが送信したスタートアップフレームを検出し、サイクル 0～3 の間に同期を試みた後、サイクル 4～6 中にスタートアップフレームを送信します。

④　その他のノードが送信された最低 2 つのスタートアップフレームに対して、自身を同期させて、通信に参加します。

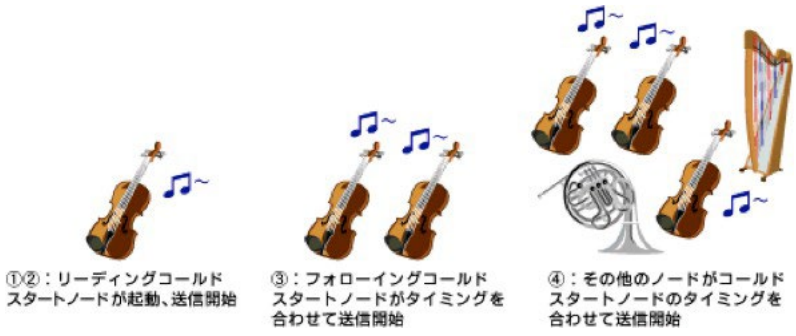

図 36 スタートアップ手順のイメージ

3.3.1 スタートアップの具体例

簡単な例を基にスタートアップの手順や各ノードの動作をもう少し詳しく紹介します。

まず、ノードの構成は以下の通りとします。コールドスタートノード数＝2 と最小構成です。

> ノード A：コールドスタートノード（リーディングコールドスタートノード）
> ノード B：コールドスタートノード（フォローイングコールドスタートノード）
> ノード C：通常のノード（コールドスタートノードではない）

ノード A→ノード B →ノード C の順番で通信を開始する具体的なフローは以下の通りです。

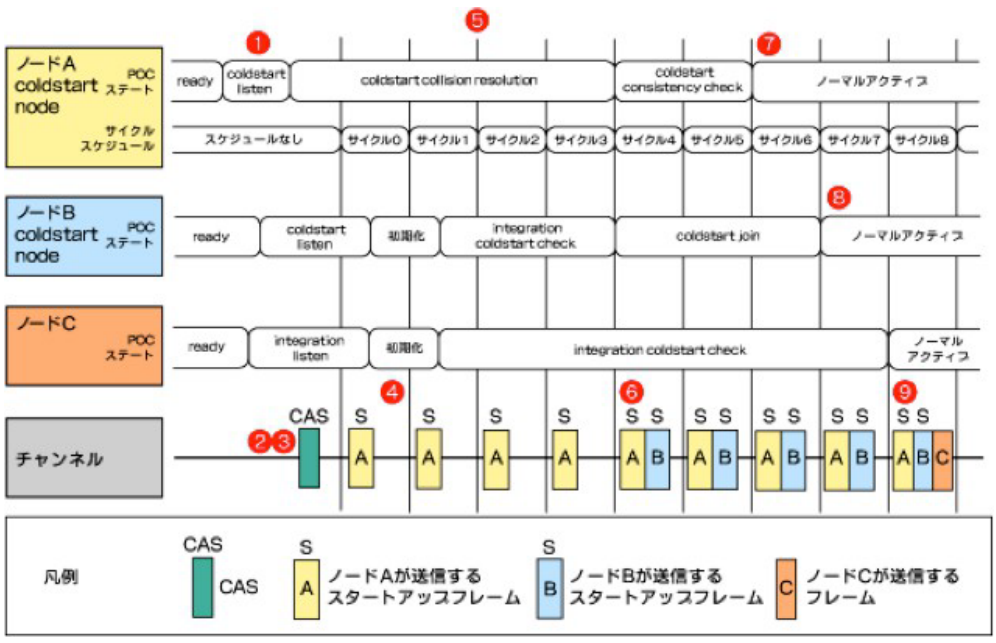

図 37 スタートアップの具体例（※FlexRay Specification V2.1 Rev.A から抜粋／編集）

① ノード A が自身のコントローラのステートをスタートアップに移行します。またバスの通信状況を確認し、通信がされていないことを確認します。

② ノード A はリーディングコールドスタートノードとなり、CAS を送信します。

③ この CAS により、他のノード B と C はノード A を検知します。

④ CAS 送信から特定時間が経過後、通信サイクルがサイクル 0 から始まります。ノード A はサイクル 0 にて最初のスタートアップフレームを送信します。その後、計 4 サイクルにわたって（サイクル 0～3）、スタートアップフレームを送信し続けます。

⑤ FlexRay プロトコルでは、コールドスタートノードと定義されているすべてのノードがリーディングコールドスタートとなって CAS を送信できるため、ノード B もほぼ同時に CAS を送信する可能性があります。仮にノード A が CAS 送信後にノード B が送信する CAS を受信したとき、ノード A が直ちに①の“バスの通信状況を確認する”状態に戻り、ノード B もスタートアップを中断することで、衝突を避けます。

⑥ ノード B ではノード A が送信したスタートアップフレームを検出した後、サイクル 0～3 の間に初期化やノード A にタイミングを合わせ、サイクル 4 でスタートアップフレームの送信を開始します。つまり、ノード B がフォローイングコールドスタートノードとなります。

⑦ ノード A はサイクル 4～5 での 2 つのスタートアップフレームからクロックの補正を行い、何もエラーがなかった場合、サイクル 6 からノーマルアクティブ＝通常の通信状態になります。

⑧ ノード B がスタートアップフレーム送信を開始してから 3 サイクル（サイクル 4～6）の間に、クロック補正について何もエラーがなく、他のコールドスタートノードであるノード A を正しく認識できている場合、サイクル 7 から通常の通信状態になります。

⑨ ノード C でもノード A が送信したスタートアップフレームを検出した後、初期化などを行い、ノード A と B から送信される 2 つのスタートアップフレームにタイミングを合わせます。4 サイクル（サイクル 4～7）にわたってスタートアップフレーム×2 を受信した後、通常の通信状態となり、通常のフレームの送信を開始します。

この例から分かる通り、FlexRay の通信開始にはコールドスタートノードの役割が大きく、クラスタ設計上、クラスタ内におけるコールドスタートノードの“ 数”が重要となります。プロトコル仕様では、コールドスタートノード数＝2 を認めてはいますが、この場合、仮に何らかの原因により 1 つのコールドスタートノードが起動しなかったとき、2 つのコールドスタートノードがそろわないことになります。結果、クラスタ全体が通信を開始できなくなるため注意が必要です。

4. 欧州での FlexRay 採用事例と取り組み

第 4 章では「、欧州での採用事例」「開発効率化のためのさまざまな取り組み」などを紹介します。

4.1 欧州での採用事例

2006 年、世界で初めて BMW 社が SUV 車「X5 モデル」に FlexRay を採用して以来、複数の自動車メーカーが採用しています。ここでは、BMW 社における過去の採用事例をいくつか紹介します。

4.1.1 BMW 社 X5 ― アダプティブドライブ

2006 年に発表された BMW X5 では、アダプティブドライブ（Adaptive Drive）と呼ばれる電子ダンパー制御システムに FlexRay を採用しました。

このアダプティブドライブシステムでは、走行速度やステアリング角度、前後方向の加速度、ホイールの加速度、車高などの各種データを基に、スタビライザーバーの旋回モータとショックアブソーバーの電磁バルブを制御します。これにより、走行状況に応じたボディロールとダンピングを制御することができ、優れた乗り心地と安全性、操作性といった通常、相反するニーズを両立させています。

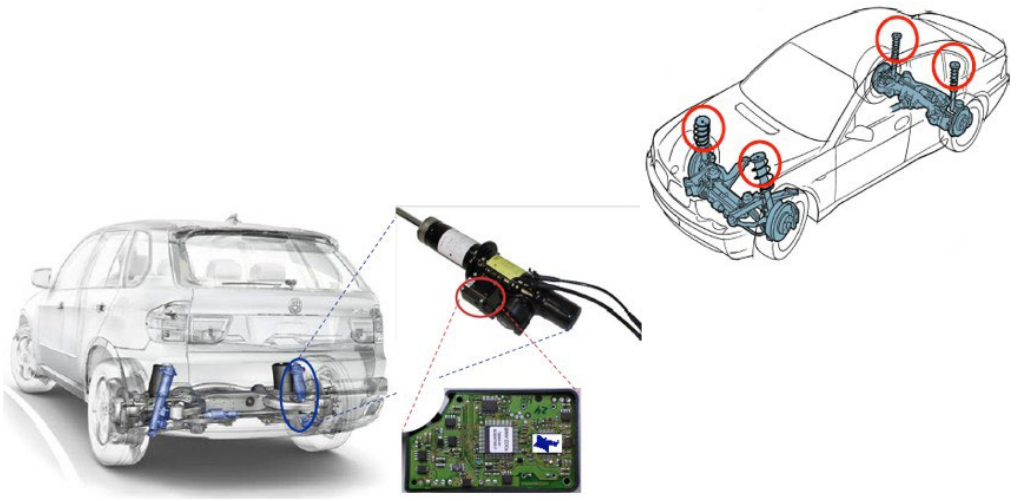

図 38 BMW 社「X5 モデル」アダプティブドライブ

図 39 は、このアダプティブドライブシステムの FlexRay トポロジーです。中央の制御モジュール ECU とそれらに接続されるダンパーモジュール（サテライト ECU）の計 5 ノードから構成されています。個々の車輪の励起を各ダンパーモジュールが制御し、ピッチとロールが発生した場合は高レベルのアルゴリズムを搭載した中央の制御モジュールがシャーシ全体の制御システムと連携して車両自体をコントロールします。このような統合的な協調制御ではデータ通信量が増加し、高い応答性が求められるため、従来の CAN では対応が難しく、FlexRay が必要となりました。

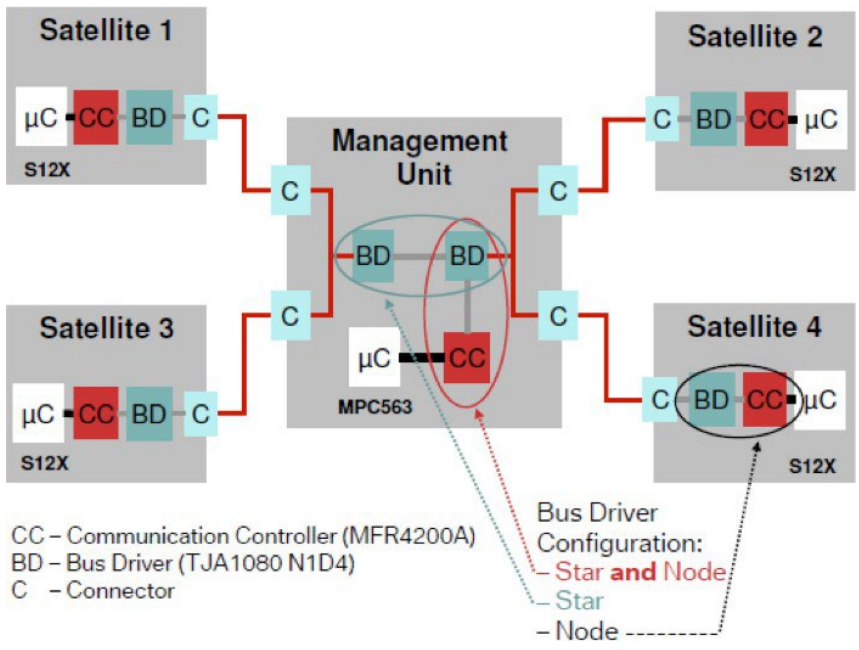

図 39 アダプティブドライブ FlexRay トポロジー
※Vector FlexRay Symposium 2007、Goals and Architecture of FlexRay at BMW より

4.1.2 BMW 社 7 シリーズ ― バックボーンネットワーク

2008 年に発表された「BMW 7 シリーズ」モデルでは、FlexRay の採用規模がさらに拡大しています。ドライバーアシスタンス、シャーシ、パワートレインなどの ECU と接続し、ノード数は 13 です。トポロジーはスター型とバス型が混在したハイブリッド型で、CAN や MOST（モスト）といった他の通信規格ネットワークとのゲートウェイも含まれています。通信速度は 10Mbit/s、シングルチャンネル（A チャンネルのみ）です。

また、FlexRay を車両の“バックボーン（Backbone）ネットワーク”として活用しています。これは、車両上の幹線道路のようなもので、バックボーンネットワークを通してシャーシやパワートレインなど複数のドメイン間にまたがるデータ処理を行うため、通信データ量は膨大となり、FlexRay による高速通信が必要となりました。これによって、より効率的で洗練された協調制御を実現しています。

なお、2009 年に発表された Audi 社の「A8」モデルでも、同様に FlexRay をバックボーンネットワークとして採用しています。

参考 BMW 7 シリーズモデル（2008 年）
※BMW Japan 社 Web サイトより

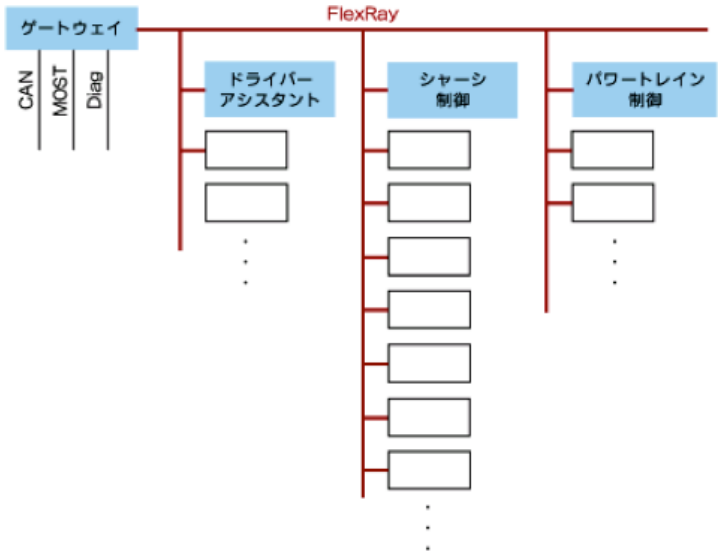

図 40「BMW 7 シリーズ」モデルの FlexRay トポロジー概観

4.1.3 BMW 社 5 シリーズ ― インテグラルアクティブステアリング

2010 年に発表された「BMW 5 シリーズ」モデルでは、インテグラルアクティブステアリング（Integral Active Steering：前後輪統合制御ステアリング）と呼ばれるシステムの協調制御に FlexRay が使われています。

このインテグラルアクティブステアリングは、フロントホイールの切れ角を車速度に応じて変えるシステムにリアホイールのステアリング機能を組み合わせたもので、いわゆる 4 輪操舵システムです。時速 60km/h 未満では、リアホイールがフロントホイールと反対方向に操舵されるため、低速走行における取り回しや俊敏性がよくなります。一方、60km/h 以上では、リアホイールがフロントホイールと同じ方向に素早く操舵されることにより、走行安定性が向上し、より快適な運転を実現します。

参考 BMW 5 シリーズモデル（2010 年）
※BMW Japan 社 Web サイトより

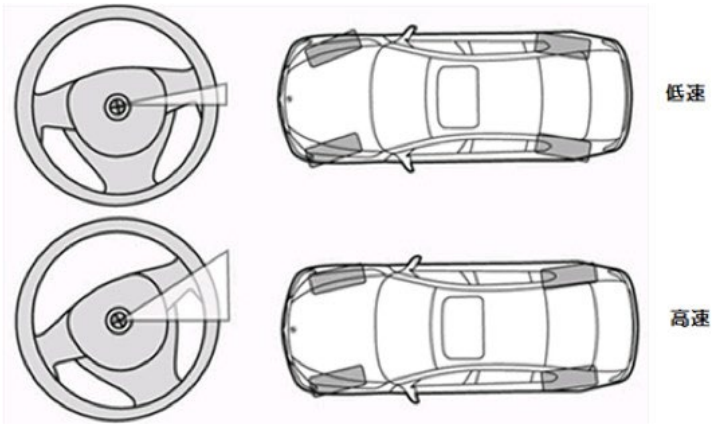

図 41「BMW 5 シリーズ」モデル インテグラルアクティブステアリングの概観
※BMW Japan 社 Web サイトより

このシステムの実現には、ヨーレートセンサー、横加速度センサー、ステアリング切れ角センサーといった各センサーや操舵アクチュエータが高速かつ安定して協調動作する必要があります。そのため、BMW 5 シリーズモデルでは、それらを制御する ECU が FlexRay で接続され、互いに通信を行っています。

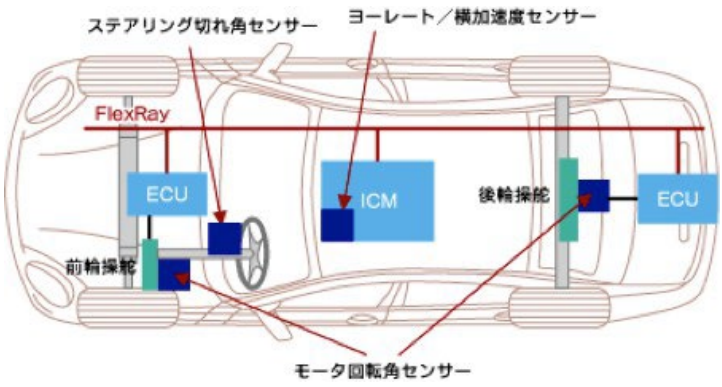

図 42 インテグラルアクティブステアリング FlexRay ネットワーク
※ICM（Integrated Chassis Management：インテグレーテッドシャーシマネジメント）

4.2 さまざまな取り組み

これまで紹介してきたように、FlexRay は高い信頼性と統合的な協調制御に適応する優れた機能を装備しています。一方、定義すべき通信パラメータの数が 60 以上にもなり、かつそれらを厳密に関連付ける必要があるなど、その仕様は複雑です。よって、“通信ネットワークをどのように効率的に設計すべきか”が大きな課題となっています。

以下、この課題に対する欧州での取り組みを幾つか紹介します。

4.2.1 パラメータセット

FlexRay 通信パラメータの考えられる組み合わせは膨大な数となり、通信ネットワークを開発するたびに、それらを評価し、決めていくことは非常に手間の掛かることです。よって、あるパラメータの組み合わせ“パラメータセット”を事前に評価、定義し、開発ごとにそれらをテンプレートとして適用することで開発の効率化を図る取り組みが行われています。

次の図 43 は、ある欧州自動車メーカーにおけるパラメータセットの一部です。同社ではパラメータセットを 1 種類定め、複数車種に搭載するすべての FlexRay クラスタにこれをベースとしたパラメータを適用することで開発の効率化を実現しています。

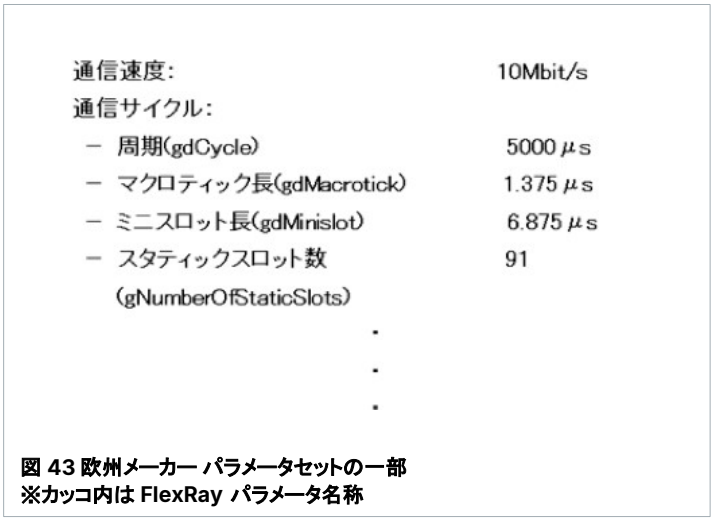

図 43 欧州メーカー パラメータセットの一部
※カッコ内は FlexRay パラメータ名称

このように、ある特定のパラメータセットを毎回使用することで開発の効率性を高めることはできますが、その反面、開発の柔軟性は失われ、車種ごとに最適化設計を行うことも難しくなります。開発の柔軟性と効率性のトレードオフの中で、さまざまな要因を考慮しながら“パラメータセット”をどの程度用意するべきなのかを見極める必要があります。例えば、日本における標準化団体「JasPar(Japan Automotive Software Platform and Architecture:ジャスパー)」においても同様に、パラメータセットを評価、定義する試みが行われ、ドメインごとに複数種類のパラメータセットを用意しました。

4.2.2 データベースフォーマット「FIBEX」

車両開発の際、複雑な FlexRay の通信仕様やその膨大な通信パラメータをどのように複数の部門間、企業間で伝達、運用していくかが大きな課題となります。例えば、紙の仕様書によって展開する場合、そこには仕様に関する誤解釈や設定ミスが生まれる潜在的な要因が大きくあり、また各開発者にとっても手間の掛かるものとなります。

この課題を解決する効果的な方法が標準的なデータベースフォーマットの活用です。ファイルフォーマットによって通信仕様を定義し、共有することで、開発中における伝達ミス、設定ミスが生じる可能性を大幅に減らすことができます。また、標準的なフォーマットであることから、多くの汎用的な開発ツールがこのデータベースファイルに対応し

ており、後工程でファイルをそのまま使うことができるため、工数の削減にもつながります。これは FlexRay だけではなく、CAN/LIN など他の車載ネットワークに関連した開発でも同様に有効ですが、複雑な FlexRay ではより効果的です。

図 44 データベースの活用

FlexRay では、「FIBEX（Field Bus Exchange Format：ファイベックス）」と呼ばれるフォーマットが標準的なデータベースフォーマットの一つとして活用されています。FIBEX は、欧州標準化団体「ASAM（Association for Standardisation of Automation and Measuring Systems）」にて策定された、XML スキーマベースのフォーマットです。FlexRay 専用フォーマットではなく、CAN、LIN、MOST といった他の通信プロトコルにも対応しています。既に多くの FlexRay 関連製品が FIBEX に対応しているため、FlexRay のデータマネジメントシステムやツールチェーンを比較的簡単に、低コストで構築することができます。

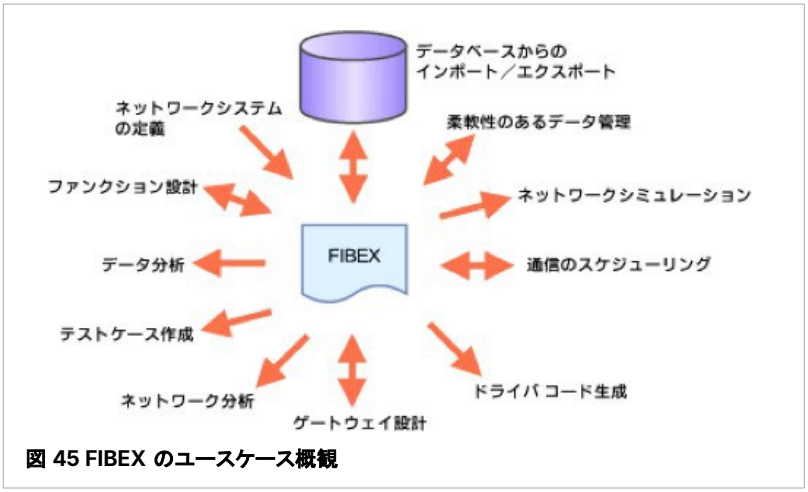

図 45 FIBEX のユースケース概観

4.2.3 開発ツールの活用

通信仕様の複雑さや求められる信頼性が高いことから、FlexRay ネットワークの開発では、十分な機能を持った開発ツールを使って工数の削減、品質の向上を図ることが、これまで以上に重要となります。ここでは、欧州の自動車メーカー4 社（Daimler 社、BMW 社、Audi 社、Volvo 社）にて行われている試みを紹介します。

開発期間の短縮、開発工数の削減などのため、実際のネットワークやノードが完成する前にシミュレーション環境を構築して、ネットワークやノード仕様の妥当性を確認することがよく行われています。このとき、シミュレーション用の開発ツールにネットワークやノードの仕様を組み込む必要があり、その工程における作業工数や品質管理などは無視できない課題です。

一方、自動車メーカーでは、例えば以下のような詳細事項をそれぞれ独自の共通仕様として定め、車両に搭載するすべてのノードに適用しています。

> いつ、どのような条件でノードがパワーオフになるか？ 送信を開始、停止するか？
> 特定のフレームやシグナルはどのような方法で、どの程度の頻度で送信されるか？
> 送受信の際に分割・統合される長いフレームはどのように処理されるか？

このような共通仕様を、毎回、シミュレーション用の開発ツールに手動で組み込むことは、無駄な作業といえるかもしれません。

これに対し、上記の自動車メーカーでは独自の共通仕様に適応した開発ツールの自動モデリングパッケージを用意し、関連サプライヤーに展開、そして FIBEX ファイルなどのデータベースファイルを基に開発ツールのシミュレーション実行モデルを自動的に生成することによって、開発の効率化、品質向上を実現しています。

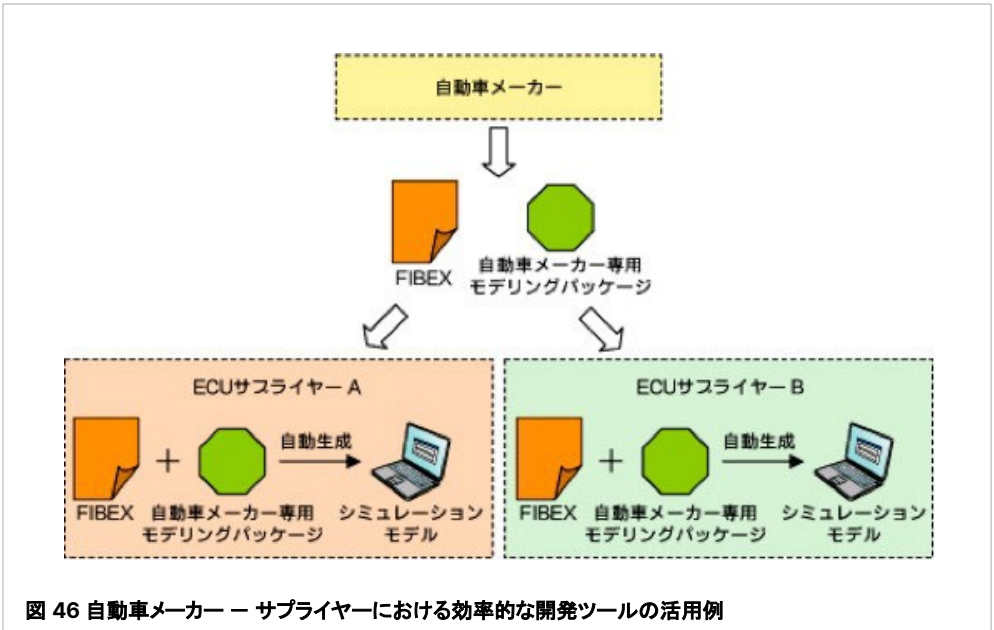

図 46 自動車メーカー ― サプライヤーにおける効率的な開発ツールの活用例

以上、4 章にわたって FlexRay が策定された背景から、その仕様の概要、採用事例などを紹介してきました。本稿が皆さまの FlexRay の理解に少しでも参考になれば幸いです。

5. お問い合わせ窓口

技術関連のお問い合わせは、ベクターサポートまでお寄せください。

✓ サポートリクエスト（Web サイト経由のサポート問い合わせフォーム）
www.vector.com/jp/ja/support-downloads/support/

【お問い合わせ時のお願い】

お問い合わせの際、サポート開始前に製品のバージョン（例：CANape バージョン xx.x など）、シリアルナンバーなどをお伺いいたします。お手数ではございますが、事前にご確認のうえ、お問い合わせくださいますようお願い申し上げます。

【サポート対応について】

夜間／休日に頂いたお問い合わせは翌営業日の対応となります。また、内容によりご回答までにお時間を頂く場合がございます。なお、不具合が発生した際にご使用されていた関連プログラム一式のコピーを、検証のためご提供いただくようお願いする場合がございます。何卒ご理解ご協力のほど、よろしくお願い申し上げます。

✓ **お見積もり／保守契約関連のお問い合わせ**

営業部　TEL：（東京）03-4586-1808　E-mail：sales@jp.vector.com

VECTOR >

MORE INFORMATION
Visit our website for:

> News
> Products
> Demo software
> Support
> Training and workshops
> Contact addresses

vector.com