**TJ 向け勉強資料**

**改版履歴**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版数 | 日付 | 作成者 | 記載箇所 | 内容 | 備考 |
| 0.1 | 2023/04/20 | 本城 |  | 新人向けに新規作成 |  |
|  |  |  |  |  |  |

目次

[**第1章** **組み込み知識** 4](#_Toc135410362)

[**1-1** **マイコン** 4](#_Toc135410363)

[**1-2** **組み込み開発の流れ** 4](#_Toc135410364)

[**1-3** **組み込み通信** 5](#_Toc135410365)

[**1-4** **MISRA-C** 9](#_Toc135410366)

[**1-5** **ベクタテーブル** 16](#_Toc135410367)

[**1-6** **アセンブラ** 17](#_Toc135410368)

[**1-7** **ARMアセンブラ** 20](#_Toc135410369)

[**1-8** **メモリ** 21](#_Toc135410370)

[**第2章** **CANの基礎知識** 23](#_Toc135410371)

[**2-1** **CANの特徴** 23](#_Toc135410372)

[**2-2** **CANプロトコル** 26](#_Toc135410373)

[**2-3** **ダイアグ通信** 32](#_Toc135410374)

[**第3章** **AUTOSAR** 34](#_Toc135410375)

[**3-1** **標準化** 34](#_Toc135410376)

[**3-2** **構造** 35](#_Toc135410377)

[**3-3** **ツールチェーン** 36](#_Toc135410378)

[**第4章** ソフトウェアの更新 37](#_Toc135410379)

[**4-1** **OTA/リプログラミング** 37](#_Toc135410380)

1. **組み込み知識**

　組み込み開発とは、特定の機能を実現するために機械や装置等に組み込まれているシステム開発のことである。「特定の機能を実現する」ところで、PCなどの汎用的なシステムとは対照的である。

1. **マイコン**

マイコンは機器の動作のかなめとなる、ハードウェアの制御を行っている。構成要素として、CPU、メモリ、周辺機能などで構成されている。あらかじめ用意された命令の組み合わせ(プログラム)がメモリに記憶されており、その命令を順次読みだして実行する。

**マイコン**

命令に従って

「入力-加工(演算)-出力」

を繰り返す

**入力**

**装置**

**出力**

**装置**

**CPU**

命令を読み出し

解読して実行する

**メモリ**

命令(プログラム)

データを記憶する

**周辺機能**

入出力装置と

CPUをつなぐ

スイッチ

センサ　など

モータ

LED　など

1. **組み込み開発の流れ**

　組み込みシステムの開発は、開発環境と動作環境が異なるので、一般的なソフトウェア開発とは開発フローが異なる。組み込み開発は大まかに以下の流れで進んでいる。

1. 要件定義・必要機能の抽出
2. システムの設計 | ハードウェア・ソフトウェア
3. 実装
4. クロスデバッグ
5. 環境試験

* クロスデバッグ

　組み込み開発では、プログラムを記述してビルドする環境と、そのプログラムが実行される環境が異なることが多い。このように開発環境と実行環境が異なることを、クロス開発環境と呼ぶ。デバッグにあたっても、動作環境であるハードウェアで実行したものを、開発環境のハードウェアで結果を表示して、チェックする必要がある。このようなクロス環境でのデバッグが、クロスデバッグである。

1. **組み込み通信**

* SPI通信

　SPI通信はデバイスどうしを接続するのによく使用される、同期的シリアル通信の一つである。以下の4つの信号線を使用して通信を行う。

|  |  |
| --- | --- |
| **信号線** | **概要** |
| SIMO | Slave In Master Outの略。\*マスターモード時はデータ出力ピンになり、\*スレーブモード時はデータ入力ピンになる。 |
| SOMI | Slave Out Master Inの略。マスターモード時はデータ入力ピンになり、スレーブモード時はデータ出力ピンになる。 |
| SCLK | Serial Clockの略。マスターモード時はクロック出力ピンになり、スレーブモード時はクロック入力ピンになる。 |
| SS | Slave Selectの略。Lowアクティブの信号になり、マスターは複数あるスレーブ機器の中で、通信したいスレーブ機器のSS端子をLowに制御し通信を行う。スレーブ機器が一つしか接続されていない場合には、スレーブ側のSS端子をLowに固定にすることで、配線を省くことが可能である。 |

\*マスター/スレーブ ： 複数の機器や装置、ソフトウェア、システムなどが連携して動作する際に、制御する側をマスター、される側をスレーブと呼ぶ。

**SPI**

**Master**

SCLK

SIMO

SOMI

**SPI**

**Slave**

SCLK

SIMO

SOMI

**SPI**

**Slave**

SCLK

SIMO

SOMI

**接続例**

接続するスレーブ機器の数SS端子に接続

* I2C

　I2C通信とはInter-Integrated Circuitの略で、デバイス同士を接続するのによく利用される同期式シリアル通信の一つである。一般的に速度はSPI通信に劣りますが、I2C通信は2本の信号線ですべてのデータの送受信を行う。そのため信号線は少なくて済む。

以下がI2Cの信号線である。

|  |  |
| --- | --- |
| **信号線** | **概要** |
| SDA | Serial Dataの略。SCLに同期し、データの転送を行う。 |
| SCL | Serial Clockの略。マスター側からSCLをスレーブに出力を行う。 |

**I2C**

**Master**

**SCL　　SDA**

**SCL　　SDA**

**I2C**

**Slave**

**SCL　　SDA**

**I2C**

**Slave**

**接続例**

**(VCC)**

**(SCL)**

**(SDA)**

* UART

UART通信はUniversal Asynchronous Receiver Transmitterの略で非同期式シリアル通信である。そのため、信号線は送信用のデータラインと、受信用のデータラインの2線で通信を行い、クロック用のラインは存在しない(データ転送のタイミングを揃えない)。

以下はUARTの信号線である。

|  |  |
| --- | --- |
| **信号線** | **概要** |
| TXD | Transmitter Dataの略。データを送信に使用する。 |
| RXD | Receiver Dataの略。データを受信に使用する。 |

**UART**

TX

RX

**UART**

TX

RX

**接続例**

* MIPI

　MIPI(Mobile Industry Processor Interface)の略で、デジタルインターフェースの一種であり、高速なデータ転送が可能である。車載システムにおいて、カメラやディスプレイなどの高速なデータ転送に使用されている。

以下は、物理層の規格である。

|  |  |
| --- | --- |
| **信号線** | **概要** |
| M-PHY | 高性能カメラやメモリ、チップ間アプリケーションなどを使用する場合の、性能重視の双方向パケット通信やネットワークで使われる。 |
| D-PHY | カメラまたはディスプレイ側に適用され、低速の単方向ストリーミングに使われる。 |
| C-PHY | D-PHYと同様にカメラまたはディスプレイ側に適用されますが、クロック方式や送信振幅が異なる。 |
| A-PHY | ADAS（先進運転支援システム）やADS（自動運転システム）、カメラやカーナビなど、ほかのサラウンドアプリケーションを含む自動車アプリケーションに適用される。 |

　以下は、通信規格である。

|  |  |
| --- | --- |
| **信号線** | **概要** |
| CSI-2 | Camera Serial Interface 2の略。映像データの転送に特化したプロトコルで、単方向データ転送を行う。 |
| CCI | Camera Control Interfaceの略。カメラモジュールの制御インターフェースで、双方向データ転送を行う。 |

**接続例**

SoC

MIPI

RX

Freeze

detection

Bypass

SEL

Open-LDI

Tx

Display

From Camera

MIPI

CSI-2

INIT

Open-LDI

**CrossLink-NX、又はその他Lattice社FPGA**

* LVDS

　Low Voltage Differential Signalingの略で、低電圧差動信号のことを指し、ノイズに強く、高速なデータ転送が可能である。車載システムにおいて、ディスプレイなどの信号伝送に使用される。

　以下は、LVDSの特徴を示す。

**・大量のデータ転送が可能**

　　振幅を小さくすることで、大量のデータを伝送することができる。

data

data

data

3.3V

350mV

data

data

data

振幅を

小さくする

データを

多くする

**・データを安定的に伝送する**

以降CANに説明記載。

1. **MISRA-C**
2. ディレクティブとルール

　すべてのガイドラインは、ディレクティブとルールに分類される。

* ディレクティブ(Directive)

適合のチェックを実行するために必要な厳密な定義を提供することが不可能なガイドラインである。チェックを実行するには、設計書や要件所などで提供されているような追加的な情報が必要になる。

例）

|  |  |
| --- | --- |
| Dir2.1 | すべてのソースファイルは、コンパイルエラーなしにコンパイルされなければならない。 |

Dir2.1への適合はソースコードのみからは判断できないため、コンパイル結果を確認する。

* ルール(Rule)

　要件が厳密に定義されているガイドラインである。他の情報を必要とせずともソースコードがルールに適合しているかをチェックすることができる。

例）

|  |  |
| --- | --- |
| Rule9.1 | 自動記憶域期間を持つオブジェクトは、設定する前に値を読取ってはならない。 |

初期化前の自動記憶域期間を持つオブジェクトを参照していないか確認する。

1. ガイドラインのカテゴリ

MISRA C:2012のガイドラインは、必須(Mandatory)、必要(Required)、推奨(Advisory)の三つにカテゴリ分けされている。

|  |  |
| --- | --- |
| 必須ガイドライン | 適合していなければならず、逸脱は認められない。 |
| 必要ガイドライン | 適合していなければならない。適合できない場合には、正式な逸脱手順が必要となる。また、組織やプロジェクトによって、「必要」を「必須」として取り扱っても良い。 |
| 推奨ガイドライン | 無理のない範囲で可能な限り適合すべきである。適合していない場合には正式な逸脱手順を必要としないが、非適合であることを文書化するなど、その代わりとなる対応をすべきである。また、組織やプロジェクトによって、「推奨」を「必須」や「必要」として取り扱っても良い。 |

1. 逸脱の手続き

　プログラムにあるガイドラインに「非適合」な記述があっても、適切な逸脱手続きを粉うことで、MISRA Cを遵守していると言うことができる。

逸脱は、次の2種類に分けられている。

|  |  |
| --- | --- |
| プロジェクト全体の逸脱  (Project Deviation) | プロジェクトに共通した状況で生じる逸脱で、通常は開発初期に定められている。 |
| 特定箇所の逸脱  (Specific Deviation) | 個別に単一のファイル内の単一箇所で生じる。 |

逸脱記録には次の内容を記録することが推奨されている。

* 逸脱するガイドラインの項目
* 逸脱が許される状況
* プロジェクト全体の逸脱は「モジュールXのすべてのコード」のように記述することができるが、特定の逸脱は逸脱するコードの場所を特定できるように記述する
* 逸脱する場合としない場合のリスクを比較する
* 安全性の保証に対する説明
* 逸脱により非安全な状態が発生しないことなどの説明、追加で行うレビューやテストの説明
* 逸脱に起因する潜在的結果とそれに対して取られた緩和措置

1. MISRA C適用時の注意点

　自動車の機能安全規格ISO26262-6:2018にMISRA Cという言葉が登場するように、必須ではなくても、MISRA Cを活用することでISO 26262に適合しやすくなる。その一方で、「MISRA Cを適用すれば、ISO 26262のいくつかの要求はパスできる」と誤解されているケースが多い。

　C言語規格やコンパイラや静的解析ツールについての理解が不十分のまま「…⇒実装⇒コンパイル⇒MISRA Cチェック⇒レビュー⇒…」といった流れで機械的に開発していては、十分に安全な対応にならない。

C言語規格では未定義/未規定/処理系定義といった問題が生じやすい箇所(リスク)について、約200件もの要求事項が存在している。MISRA C 2012ではDir1.1やRule1.3としてまとめられているため、MISRA CのみではC言語規格で明示されているリスクを取り除くのは不可能であり、多くの残留リスクについて別途対処する必要がある。

1. QACツール

Quality Assurance Compilerの略で、主に、組み込みソフトウェア開発で使用される、ソフトウェアの品質を向上させるための静的解析ツールである。QACの開発元である英国PRQA社は、MISRA C:1998,MISRA C:2004の査定委員会の一員であるため、適切な解釈に基づいたサポートがされている。

1. ディレクトリ一部

* Dir4.6(推奨)

基本的な数値型の代わりに、サイズと符号属性を示すtypedefを用いるべきである。

|  |
| --- |
| 例えば、32ビットのC90処理系では、次のような定義が適切。  ※main関数の場合、戻り値の型としてtypedefよりもintを使ってよい  typedef signed char int8\_t;  typedef signed short int16\_t;  typedef signed int int32\_t;  typedef signed long int32\_t;  typedef unsigned char uint8\_t;  typedef unsigned short uint16\_t;  typedef unsigned int uint32\_t;  typedef unsigned long uint32\_t;  typedef float float32\_t;  typedef double float64\_t;  typedef long double float64\_t; |

* Dir4.9(推奨)

関数により関数形式マクロが置き換え可能な場合、関数を用いるべきである。関数形式マクロの方が高速に動作する可能性が高いが、関数と関数形式マクロでは以下の違いがあるため、関数を使用すべきである。

* 関数は実引数と戻り値の型をチェックすることができるが、関数形式マクロはチェックできない。
* 関数は実引数を1回だけ評価するが、関数形式マクロは複数回評価することがある。

|  |
| --- |
| 例）  以下に示す関数funcと関数形式マクロPOWの実引数に++aを与えた場合、  “func”の場合 ：呼び出し前にaのインクリメントが1回だけ行われる  “POW”の場合：((++a)\*(++a))と展開され、aのインクリメントが2回行われる  このようにインクリメントが2回行われることは意図しない結果である可能性が高い。  uint16\_t func( uint8\_t x )  {  return x \* x;  }  #define POW(X) ( ( X ) \* ( X ) ) |

インクリメント：数値に1を加えること

* Dir4.12(必要)

動的メモリ割り当ては、用いてはならない。以下の問題があることを十分に理解する。

* メモリリークやメモリ不足による、割り当ての失敗やシステム停止の発生
* 使用方法やフラグメンテーションによる、メモリ割り当てや解放の処理時間のばらつき

1. ルール一部

* Rule2.1(必要)

プロジェクトは、到達しないコードを含んではならない。

　必要なコードが誤って実行されなくなってしまうことを防止する。到達しないコード(実行されないコード)は以下である。

* プログラマによる勘違い、ロジック検討不足、コーティングミス
* デバッグ用コードの書き残し
* コピー＆ペースト多用によるミス
* break、return位置ミス

　防衛的プログラミングとして、到達しないカード(実行されないコード)を書く場合は、逸脱の手続きが必要なる。

|  |
| --- |
| 例）到達しないelseの場合  else  {  ;　/\*処理は不要である\*/  }  「何もしない」というようなコメントを記載することでRule2.1、Rule15.7に適合できる。 |

* Rule2.3(推奨)

プロジェクトは未使用の型宣言を含むべきではない。宣言された型名が全く使用されていないということはコーティングミスの可能性がある。レビュー時や再利用時に混乱招くので余計なことは記述するべきではない。

* Rule4.2(推奨)

　3文字表記は、用いるべきではない。3文字表記とは、ISO646に従うキーボードで、そこに備わっていない文字を使いたいときに、2つの疑問符(??)とそれに続く記号で表現するように設けられたものである。意図しない??には注意すること。

|  |  |  |  |
| --- | --- | --- | --- |
| 置換元文字 | | | 置換先文字 |
| 3文字表記 | Iso646.h | 2文字表記 |
| ??= |  | %: | # |
| ??( |  | <: | [ |
| ??/ |  |  | \ |
| ??) |  | >: | ] |
| ??’ | xor |  | ^ |
| ??< |  | <% | { |

* 識別子

　Rule5番台の識別子の命名に関するルールであり、ルール文には「識別可能」「固有」という用語が用いられている。双方ともに、複数の識別子に重複がないことを示している。

　C言語の識別子には、名前空間および有効範囲と結合という区分において、それぞれ以下の分類がある。

|  |  |  |
| --- | --- | --- |
| 名前空間 | 有効範囲 | 結合 |
| (1) ラベル名 | (1) 関数 | (1) 外部結合 (external linkage) |
| (2) 構造体・共用体・列挙体のタグ | (2) ファイル | (2) 内部結合 (internal linkage) |
| (3) 構造体・共用体のメンバ | (3) ブロック | (3) 無結合(no linkage) |
| (4) その他すべて識別子 | (4) 関数プロタイプ |  |

識別子における区別できる先頭文字数までの文字数について、C言語規格では、以下の表に定める文字数を満たさなければならない。

|  |  |  |
| --- | --- | --- |
|  | C90 | C99 |
| 内部識別子 | 31文字 | 63文字 |
| マクロ名 | 31文字 | 63文字 |
| 外部識別子 | 6文字 | 31文字 |

* Rule5.1(必要)

　外部識別子は、識別可能でなければならない。

|  |
| --- |
| 例）利用する処理系が31文字までの外部識別子を識別可能な場合の例  int32\_t engine\_exhaust\_gas\_temperature\_raw;  int32\_t engine\_exhaust\_gas\_temperature\_scaled;  /\*非適合\*/ 識別子は、先頭から32文字目で初めて区別できるため識別可能ではない。 |

* Rule5.5(必要)

　識別子は、マクロ名と識別可能でなければならない。このルールでは、前処理前に存在しているマクロの名前が、前処理の後に存在する識別子と識別可能であることが必要とされている。

|  |
| --- |
| 例）ある識別子が、どれかのマクロ名と同じと誤認識されてしまうことを防止する。  #define Sum( a, b )( ( a ) + ( b ) )  int16\_t Sum;  関数形式マクロ名および識別子の両方においてSumを定義している。 |

* Rule6.1(必要)　型

　ビットフィールドは、適切な型で宣言しなければならない。未定義または処理系定義になる宣言を意図しない動作を防止する。

|  |
| --- |
| 例）  typedef unsigned int U16;  struct s{  unsigned int b1:2;　/\* C90 C99 ともに適合 \*/    int b2:2;　/\* C90 C99 ともに非適合 \*/    u6 b3:2;　/\* C90 C99 ともに適合 \*/  signed long b4:2;　/\* C90 非適合、C99 適合 \*/  unsigned char b5:2;　/\* C90 C99 ともに非適合 \*/  \_Bool b6:1;　/\* C90 非適合、 C99 適合 \*/  }; |

* Rule8.2(必要)

関数の型は、名前付き仮引数を持ったプロトタイプ形式に記載しなければならない。関数の引数の型や戸数の誤りをコンパイル時に検出しやすくする目的がある。

|  |
| --- |
| 例）仮引数がない場合はvoidと明記すること  extern int16\_t func1 ( uint16\_t n );　/\* 適合 \*/  extern int16\_t func2 ( uint16\_t );　/\* 非適合 \*/  extern int16\_t func3 ( n );　/\* 非適合 \*/  extern int16\_t func4 ( ); 　/\* 非適合 \*/  extern int16\_t func5 ( void );　 /\* 適合 \*/ |

* Rule8.9(推奨)

　オブジェクトの識別子が単一の場合のジャン藪納だけに現れる場合、そのオブジェクトをブロック有効範囲で定義すべきである。

|  |
| --- |
| 例）  適合  void func ( void )  {  uint16\_t i = 0; /\* 適合 \*/  for ( i = 0; i < N; i++ )  {  ...  }  }  非適合  uint16\_t i = 0;　 /\* 非適合 \*/  void func ( void )  {  for ( i = 0; i < N; i++ )  {  ...  }  } |

* Rule8.11(推奨)

　外部結合をもつ配列を宣言するときには、明示的にサイズを指示すべきである。外部結合をもつ配列への意図しない領域外のアクセスを防止する。

|  |
| --- |
| 例）  非適合  extern uint16\_t a[ ];  適合  extern uint16\_t a[ 5 ];  extern uint16\_t a[ ] = { 0, 1, 2, 3, 4 }; |

* Rule8.12(必要)

　列挙子の並び内では、暗黙的に指定された列挙定数の値は固有でなければならない。列挙子リストの設定値が意図しない値となるリスクを軽減する。

　C言語では、列挙子リストの値の設定は以下のようになる。

* 列挙子リストの先頭が値を持たない時、その場は”0”となる。
* 列挙子リストの途中が値を持たない時、その時には直前の値に加えた値が設定される。

|  |
| --- |
| 例）ec3とec4の値が5で同一のため非適合  enum etag1  {  ec1 = 3,  ec2,  ec3,  ec4 = 5 /\* 非適合 \*/  } |

* 本質型モデル

　型変換に起因する問題や、変数などが適切な型で宣言されていないことに起因する問題に対応するルールを規定するために、本質型の概念を導入している。

　本質型のカテゴリはBoolean, character, signed, unsigned, enum<i>, floatingの6つに分けられる。

|  |  |
| --- | --- |
| 本質型のカテゴリ | カテゴリに属する型 |
| Booleanブール | \_Bool |
| character文字 | char |
| signed  符号付 | signed char  signed short  signed int  signed long  signed long long |
| unsigned  符号なし | unsigned char  unsigned short  unsigned int  unsigned long  unsigned long long |
| eunm<i>列挙 | named enum |
| floating  浮動小数点 | float  double  long doule |

* Rule10.2(必要)

本質的な文字型の式は、加算及び減算で(本ルールで定める)不適切な使用をしてはならない。文字型の演算に制限を与えるで、意図が分かりにくい減算や誤りを防止する。

|  |
| --- |
| /\* 文字 (8 ビット ) \*/  char cha;  /\* 符号付き（8 ビット）\*/  int8\_t s8a;  /\* 符号なし（8 ビット）\*/  uint8\_t u8a;  /\* 符号付き（16 ビット）\*/  int16\_t s16a;  /\* 符号なし（16 ビット）\*/  uint16\_t u16a;  /\* 浮動小数点（32 ビット）\*/  float32\_t f32a;  /\* 列挙 \*/  enum enuma { A1, A2, A3 } ena;  例）文字の加算  ‘ 0 ‘ + f32a　/\* 非適合 \*/  ‘ a ‘ + 3　　 /\* 適合：結果’ d ‘ \*/  ‘ a ‘ + u8a /\* 適合：結果は文字型 \*/ |

* ポインタ型の変換

　ポインタ型の型変換ルールは、ポインタを次のように分類して規定している。

* オブジェクト型へのポインタ
* 関数へのポインタ
* 不完全型へのポインタ
* voidへのポインタ
* 空ポインタ定数、すなわち値を0又はそれをvoid\*でキャストしたもの

※不完全型：オブジェクトサイズを確定する情報が不十分な型(例：struct structa;)

* Rule11.9(必要)

マクロNULLは、整数の空ポインタ定数の勇逸の許可された形式である。空ポインタ定数にと記載することを禁止することで可読性を向上させるのが目的である。空ポインタとする意図で記述した場合、0で初期化されたと誤読されてしまうため。

1. **ベクタテーブル**

ベクタテーブルとは、マイコンなどの組み込みシステムにおいて、割り込みや例外などの発生時に、処理するためのアドレスや関数ポインタなどを格納するテーブルのことを言う。「

一番初めのプログラムの実行を開始する」という処理には、以下の2種類がある。

|  |  |
| --- | --- |
| 実行開始番地 | 概要 |
| 固定にするCPU | 多くが0番地(アドレス空間中の最も小さいアドレス)から実行を始める。また、0番地で「次に実行する番地は○○番地」と書き換えれば、開始番地を可変したと同様な効果が得られる。 |
| 可変にするCPU | 「ベクタテーブル」という部分に開始番地を書き込む。一般には、アドレス空間中の最も大きなアドレス部分に置く。 |

（予約領域）

　　　：

（予約領域）

（予約領域）

浮動小数点例外

（予約領域）

未定義命令例外

（予約領域）

アクセス例外

特権命令例外

（予約領域）

リセット

ノンマスカブル割り込み

（予約領域）

（予約領域）

FFFFFF80h

：

FFFFFFECh

FFFFFFE8h

FFFFFFE4h

FFFFFFE0h

FFFFFFDCh

FFFFFFD8h

FFFFFFD4h

FFFFFFD0h

FFFFFFCCh

FFFFFFFCh

FFFFFFF8h

FFFFFFF4h

FFFFFFF0h

MSB

LSB

最も大きなアドレス部分に、

一番初めに実行する

プログラムの先頭番地を置く

**ベクタテーブル例**

通常のプログラム

NMI処理プログラム

10000000h

00000000h

アドレス空間

00000000h

10000000h

**CPU**

ベクタテーブル

**(2)ベクタテーブルに書かれた**

**NMIの開始番地を読む**

**(3)読んだ番地にあるNMIの**

**プログラムを実行する**

**(1)NMIが発生**

**ノンマスカブル割り込み発生時流れ**

1. **アセンブラ**

アセンブラとは、アセンブリ言語で記述されたコンピュータプログラムを、コンピュータが直接解釈・実行できる機械語（マシン語）による表現に変換するソフトウェアである。また、アセンブリ言語とは、マシン語の数値があらわす命令にネックネーム（英語を短縮したようなもの）を付け、ニックネームを使ってプログラミングする手法である。このニックネームのことを「**ニーモニック**」と呼ぶ。

　以下は、アセンブラの命令一覧である。コンピュータにできることは、「メモリの読み書き」「I/Oとの入出力」「CPU内部で演算」「プログラムのながれの制御」の4つのみである。

・メモリの読み書きをおこなう命令

|  |  |  |
| --- | --- | --- |
| ニーモニック | 意味 | 機能 |
| LD | LoaD | メモリからCPUへデータを読み込む |
| ST | STore | CPUからメモリにデータを書き出す |
| LAD | Load ADress | CPUのレジスタに値を設置する |
| PUSH | PUSH | スタックデータを書き出す |
| POP | POP | スタックからデータを読み込む |
| SVC | Super Visor Call | I/Oとの入出力を行う |

・CPU内部での演算を行う命令

|  |  |  |
| --- | --- | --- |
| ニーモニック | 意味 | 機能 |
| ADDA | ADD Arithmetic | 算術加算 |
| ADDL | ADD Logical | 論理加算 |
| SUBA | SUBtract Arithmeti | 算術減算 |
| SUBL | SUBtract Logical | 論理減算 |
| AND | AND | 論理積(AND演算) |
| OR | OR | 論理和(OR演算) |
| XOR | eXclusive OR | 排他的論理和(XOR演算) |
| CPA | Com Pare Arithmetic | 算術比較 |
| CPL | Com Pare Logical | 論理比較 |
| SLA | Shift Left Arithmetic | 算術左シフト |
| SRA | Shift Right Arithmetic | 算術右シフト |
| SLL | Shift Left Logical | 論理左シフト |
| SRL | Shift Right Logical | 論理右シフト |

・プログラムの流れを制御する命令

|  |  |  |
| --- | --- | --- |
| ニーモニック | 意味 | 機能 |
| JPL | Jump on Plus | 演算結果がプラスの場合に分岐 |
| JMI | Jump on Mlnus | 演算結果がマイナスの場合に分岐 |
| JNZ | Jump on Non Zero | 演算結果がゼロで内の場合に分岐 |
| JZE | Jump on ZEro | 演算結果がゼロの場合に分岐 |
| JOV | Jump on Overflow | 演算結果が桁あふれの場合に分岐 |
| JUMP | JUMP | 無条件で分岐する |
| CALL | CALL | 主プログラムから副プログラムを呼び出す |
| RET | RETurn | 副プログラムから主プログラムに戻る |

* レジスタ

　CPU内部には「レジスタ」がある。レジスタは、命令やデータを格納する入れ物であり、データを演算する役割もある。アセンブラのプログラムでは、レジスタを名前で指定する。以下は、Cortex-M3のレジスタである。

|  |  |
| --- | --- |
| 汎用レジスタ | 特殊レジスタ |
| 汎用レジスタ(R0) | xPSR |
| 汎用レジスタ(R1) | PRIMASK |
| ：(R2～R11) | FAULTMASK |
| 汎用レジスタ(R12) | BASEPRI |
| スタックポインタ(R13) | CONTROL |
| リンクレジスタ(R14) |
| プログラムカウンタ(R15) |

|  |  |  |
| --- | --- | --- |
| 汎用レジスタ | 汎用レジスタ | データ処理や関節アドレッシングに用いる |
| スタックポインタ(SP) | スタックの場所を示す |
| リンクレジスタ(LR) | サブルーチン(関数が呼ばれた)の場合の戻り先アドレスが入る。 |
| プログラムカウンタ(PC) | 現在実行されている命令のアドレスをす。 |
| 特殊レジスタ | プログラムステータス  レジスタ(xPSR) | Cortex-M3の状態を示す。内部は3つに分かれている。演算結果の状態(ゼロ、負値等々)を示すアプリケーションPSR、割り込みの状態を示す割り込みPSR、Thumb-2命令の実行状態を示す実行PSRである。 |
| 割り込みマスクレジスタ | PRIMASK,FAULTMASK,BASEPRI。割り込み(例外)を無効にするために使う。 |
| 制御レジスタ(CONTROL) | 特権レベルとスタックポインタの選択を行うために使う。 |

　Cortex-M3プロセッサは、汎用レジスタのほかに特殊レジスタを持っている。特殊レジスタは主に割り込み処理に使用する。

　プロセッサには13個の汎用レジスタがある。R0～R7を下位ビット、R8～R12を上位レジスタと呼ぶ。

|  |  |
| --- | --- |
| 下位レジスタ(R0～R7) | 汎用レジスタを指定するすべての命令(Thumb-2命令の中の16bit命令、32bit命令)でアクセス可能である。 |
| 上位レジスタ(R8～R12) | 汎用レジスタを指定するすべての32bit命令によってアクセス可能である。しかし改レジスタと異なり、16bit命令ではアクセスできない。初期値は不定である。 |

1. **ARMアセンブラ**

　Advanced RISC Machinesの略でARM Holdingsによって開発されたマイクロプロセッサのアーキテクチャである。ARMは、低出力かつ高性能な特徴を持ち、主にモバイルデバイスや組み込みシステムなどの省電力要求や高性能要求を満たすために設計されている。

　ARMアーキテクチャは、RISC(Reduced Instruction Set Computer)アーキテクチャの一種である。一つ一つの命令が実行する動作は簡単だが、高速に実行させるため、組み合わせて一連の動作を効率良く実行するという概念で生まれた。以下がRISCの特徴である。

* 命令は基本的で、操作が単純であることからデコードや実行の効率性を高め、高速な命令処理を実現できる。
* メモリへのデータアクセスは専用のロードとストア命令を使用して行う。算術演算や論理演算はレジスタ間で行われ、メモリとのデータ転送は明示的な命令によって行う。
* 多くの操作がレジスタ間で行われる。データはメモリからレジスタにロードされ、レジスタ上で演算が行われる。これにより、メモリへのアクセス回数を減らし、処理速度を向上させる。
* 命令の実行を複数のステージに分割し、各ステージで並列に命令を処理するパイプライン処理が採用されている。これにより、命令の効率的な実行が可能となり、高速な処理を実現する。

ARMプロセッサはARM1からARM11と呼ばれていたが、性能や機能に関係なくCortexというファミリアで統一し、用途別に以下のアーキテクチャを定義した。

* Cortex-A

　高機能で電力効率のよいCortex-Aプロセッサシリーズは、ADAS、自動車運転、IVI、デジタルコックピット・アプリケーションなどの先進運転システムの複雑なコンピューティングニーズをサポートしている。

* Cortex-R

　リアルタイムの確定的制御機能を備えたArm Cortex-Rプロセッサ市リースは、バッテリー管理システム、車載充電器、トラクションモータなど、パワートレイン及び動作制御ドメインコントローラ向けの、信頼性が高く、安全で高機能のコンピューティング・ソリューションを提供する。

* Cortex-M

　ARMの小型で低電力のプロセッサスイートは、制約が多くエネルギー効率の良いマイクロ君トローラ(MCU)に最適である。ボディエレクトロニクス、センサ、ゲートウェイ、Soc管理などのオートモーティブ・アプリケーションに最適である。

1. **メモリ**

　メモリは情報を保存する記憶装置である。CPUによる試算結果屋プログラムの実行ファイルなどを保存する場所になる。以下は、メモリの分類である。

**RAM**

DRAM

SRAM

**ROM**

EPROM

PROM

マスクROM

**ハイブリット**

NVRAM

フラッシュ

メモリ

E2PROM

* RAM

　Random Access Memoryの略。データの読み書きを頻繁に行うことができるメモリである。電源を供給しているときのみ保持できる揮発性のメモリで、プロセッサが実行中のプログラムや演算データを一時的に保存しておく、主記憶装置として使用される。

* ROM

　Read Only Memoryの略。データの読み込みは頻繁に行えるが、データの書き込みは頻繁に行うことができないメモリである。電源が切れていても、データが保持される不揮発性の特徴を持つ。実行するプログラムの保存や、保持しておきたいデータの保存などに使用される。

* ハイブリット

　RAMとROMの両方の特養を有したメモリで、頻繁に書き込みができる上に、電源が切れてもデータが保持されるとう特徴を持つ。

* DRAM

　Dynamic RAMの略で、値段は安く、記憶容量も大きいが、データのアクセス速度が遅いという特徴がある。DRAMは1秒周期で、電源が安定した状態でもデータが消えてしまうことがあるため、DRAMコントローラというもので、周期的にメモリを初期化する必要がある。

* SRAM

　Static RAMの略で、アクセスの速度が高速だが、値段が高価で記憶容量も小さい(kBオーダ)という特徴がある。主にキャッシュメモリとして使用される。

* PROM

　Programmable ROMの意味で、プログラム可能なROMのことである。デバイスプログラマというツールを使用して、PROMのピンに電流を流すことで、データを書き込むことができる。一度データを書き込むと、変更はできず、値段はマスクROMよりも高額になる。

* EPROM

　Erasable Programmable ROMの略で、PROMのデータ消去機能が付いたROMである。書き込んだデータを消去することができ、再びデータを書き込むことができる。しかし、PROMよりも効果となる。

* E2PROM

　Electrically Erasable Programmable ROMの略。電気的にデータを消去できるROMである。機能はPROMに似ているが、電気的にデータを消せるため、データの一部だけを変更することができる。

* NVRAM

　Non Volatile RAMの略で、不揮発性を持つRAMのことである。SRAMにバッテリーによる電力バックアップを搭載することで、RAMでありながら、電源オフ時においてもデータを保持できるようになる。

* フラッシュメモリ

　E2PROMの一種だが、データの消去をブロックごとにすることにより、構造を簡略化したROM である。結果、値段も安くなり、電源が切れてもデータが保持され、データアクセスも早いという良い特徴を複数有している。

1. **CANの基礎知識**

CANとは、自動車など機械の内部で、電子回路や各装置を横断的に接続するための通信規格の一つ。Robert Bosch社によって、1986年に仕様が公開され、1994年には、国際規格(ISO11898)となり、現在最も普及している車載ネットワークプロトコルである。

1. **CANの特徴**

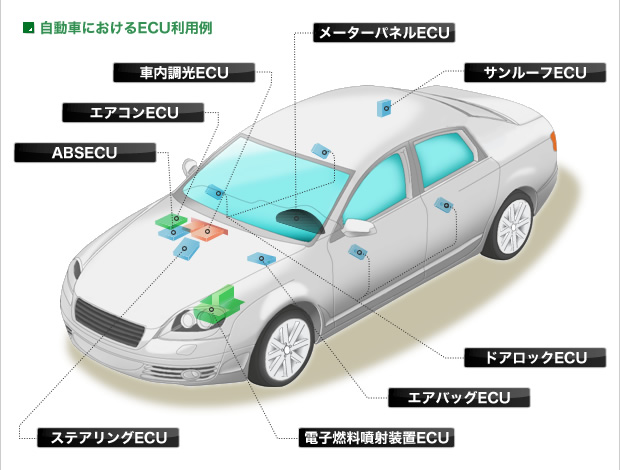
* ライン型構造

一般的にネットワークに接続される通信規格のことを「ノード」といい、車載ネットワークの場合は\*ECUがノードとなる。ライン型は、単純にバスに各ノードを接続していくことでネットワークが構成できるために、ネットワークがシンプルで設計も容易に行える。他の構造としてスター型、リング型がある。

ノード(ECU)

バス

\*ECU ：　Electronic Control Unitの略。エアバッグ、ATやCVTなどのトランスミッション、車線維持システム、車間距離制御システムなどの電子制御をするコンピュータである。現在は多い車では100個以上搭載されている。



* マルチ・マスター方式

ライン型で接続される各ノードに平等なアクセスが可能な方式である。メリットは以下の3つである。

1. 各ノードが均一私用で設計できる。
2. 各ノードに優劣が無いため、イベント指向通信に向いている。
3. ノードの追加接続が容易である。

* CSMA/CA

Carrier Sense Multiple Access with Collision Avoidanceの略。複数のノードから同時にバスにデータが送信される場合には、優先順位が高いものを衝突させずに送信させるような仕組みである。

* IDを使用したメッセージ・アドレッシング

各ノード間でデータをやり取りする方法で、データの中にIDを付加して送信する方式。IDによって、どのようなデータ化、自分が使用するデータかを判断できる。

* 耐ノイズ性に優れた物理層

　パワートレイン系に使用されるHigh Speed CAN(CAN-C)では、「2線式作動電圧方式」で通信している。2本線(ツイストペア)を使用し、それぞれの線に流れる電圧の差の有無によってデータを送信する方式。これにより、外部からノイズが混入した場合でも、それぞれの線に混入するノイズの電圧はほぼ同一となるため、電圧の差が有るかないかに影響を受けにくい仕組みになっている。

(一般的なデジタル信号波形)

(CANにおける信号波形)

5V

電圧レベルで判断

“1”と誤判定

1-0判断

2.5V

1.5V

3.5V

外部からのノイズ

電圧の差は同じ

* 優れたエラー検出機構

さまざまなエラー検出メカニズムを実装しており、ほぼ100%に近い確率で、各種エラーを検出できる。例えば、送信ノード自身が、送信するデータとバス上に流れたデータがあっているかを確認し、違いがある場合にはエラー検出することができる。

* データの一貫性

1台のノードが受信に失敗した場合、データを受信した全ノードがデータを破棄し、全ノードが受信に成功するまで、これを繰り返す。つまり、ある制御を行う場合、全ノードにおいて使用するデータは、同一になるので、CANではデータの一貫性を保つことができる。

* シグナルコーティング

CANでは、NRZ(Non Return to Zero)方式で、送信したいデータを変換して送信している。NRZ方式では、たとえば送信するデータが「0001100」の場合、連続する「0」の部分は「0」のまま、「1」の部分は、「1」のままとなり、\*ドミナント状態や\*レセシブ状態が連続することになる。

送信したいデータ

NRZ方式

dom

dom

dom

dom

dom

rec

rec

rec

rec

0

0

0

0

0

1

1

1

1

dom:dominant(ドミナント)

rec:recessive(レセシブ)

\*ドミナント ： 送信される2進数データの「0」を指す。優性を意味する。

\*レセシブ ： 送信される2進数データの「1」を指す。劣性を意味する。

※別ノードから同時に送信された場合は、ドミナントが優先される。

* 通信速度

　CANでは、1回に送信できるデータ量は最大8bytes、最大通信速度は1Mbpsとなっており、車載ネットワークのほとんどの分野で使用可能である。通信を正常に行う上で、各ノード間の通信速度を同一に保つことは重要である。

* ビットスタッフィングルール

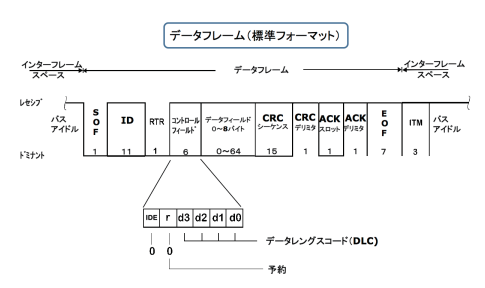
　バス上で同じ状態が5回連続した場合、それまで送信されていた状態と反対の状態ビット(スタッフビット)を1つ挿入する仕組みのことである。この仕組みにより、\*同期を行えるようになる。

\*同期 ： CANでは、各ノード間のシステムクロックの違いを補正しており、信号がレセシブからドミナントへ変化するとき(1-0変化時)に同期を行っている。

1. **CANプロトコル**
2. データフレーム

　CANにおいて、実際にいろいろなデータを含むものを「データフレーム」といい、送信ノードから制御パラメータ等のデータを送信する役割を果たす。データフレームには「標準フォーマット(11ビットID)」と「拡張フォーマット(29ビットID)」という細部で異なる2種類の形式がある。

* 標準フォーマット



* バスアイドル

　通信が行われていない時、バスがレセシブとなっていること。

* SOF

　ノードからデータフレームが送信されるとき、最初に送信される部分は開始を表すためにドミナントとする。この部分を「SOF(Start Of Frame)」という。

* ID

　データ内容や送信ノードを識別するために使用され、通信調停の優先順位を決定する働きをしている。標準フォーマットでは11ビット長でIDが構成され、範囲は0x0～0x7FFとなり、2048種類の識別が可能である。

* RTR

　Remote Transmission Requestの略。データフレームとリモートフレームを識別するために使用される。データフレームの場合はドミナントとなっている。

* コントロールフィールド

　1ビット長のIDE(Identifier Extension)、予約ビットrと4ビット長のデータレングスコード(DLC:Data Length Code)から構成される。標準フォーマットの場合は、ともにドミナントである。

* データレングスコード(DLC)

　コントールフィールドに続くデータフィールドにおいて、何バイトのデータが送信されるかを表している。DLCの背低範囲は0～8となっており、結果としてデータフィールドで送信されるデータは1バイト単位で0～8バイトまで送信できる。

* データフィールド

　送信されるデータの部分で、DLSによって設定されたデータ長となる。どのような形式でデータを割りあえるかについて設計者が決定できるようになっている。例えば、1バイトのデータを割り当てる場合では、「そのまま1バイトとして使用」「4ビットをそれぞれ1ビットずつ使用」「残り４ビットをまとめて使用」「8ビットをそれぞれ1ビットずつ使用」などが可能である。

* CRCシーケンス

　15ビット長であり、送信ノードがSOF、ID、コントールフィールド、データフィールドの送信値より演算してCRCシーケンスで演算結果を送信している。これは受信ノードが送信ノードと同様に演算して、その結果を比較することで正常に受信できたかの判断を行うことができる。

* CRCデリミタ

　CRCシーケンスの終了を表す区切り記号で、1ビット長のレセシブとなっている。また、CRCシーケンスとCRCデリスタを併せて「CRCフィールド」という。

* ACKスロット

　Acknowledgementの略。1ビット長で、送信ノードはこの部分でレセシブの送信を行います。ただし、受信ノードがCRCフィールド部分まで正常に受信できた場合は、ACKスロットのタイミングでドミナントを確認応答として送信することになっている。

　CANでは、ドミナントとレセシブが同時に送信された場合はドミナントが優先されドミナントとなるため、正常に通信を行っているCANネットワークにおいてACKスロットはドミナントとなっている。しかし、ACKスロットは1ビット長しか持たないために、CANネットワーク上の受信ノードがすべて正常に受信できたかの判断には使用できない。

* ACKデリミタ

　ACKスロットの終了を表す区切り記号で、1ビット長のレセシブとなっている。ACKスロットと併せて「ACKフィールド」と呼ぶ。

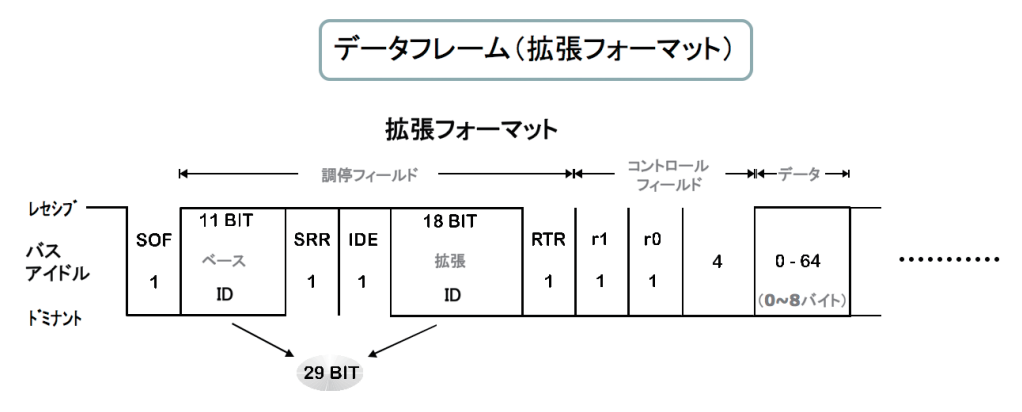
* EOF

　End Of Frameの略。データフレームの終わりに送信される。7ビット長のレセシブとなっている。ビットスタッフィングルールは、バスアイドル中やEOFは適用外になる。

* ITM

　Intermissionの略で、データフレームには含めない。3ビット長のレセシブとなっており、ITM終了後にバスアイドルになる。

* 拡張フォーマット



* ベースID

　標準フォーマットのIDを指す。11ビット長である。

* SRR

　Substitute Remote Request Bitの略で、1ビット長のレセシブとなっている。

* IDE

　Identifier Extensive Bitの略で、1ビット長のレセシブとなっている。

* 拡張ID

　18ビット長で、ベースIDと併せて29ビット長となり、これによりIDを表す。拡張フォーマットの29ビット長IDの範囲は0x0～0x1FFFFFFFとなり、536870912種類を使用することが可能である。主に、\*SAE J1939で使用されている。

\*SAE J1939 ： トラック・バスの制御とネットワーク通信のために開発されたプロトコルで、現在は建設業や農業機械などのディーゼルエンジン搭載車両や、船用電子機器などにも幅広く利用されている。

1. リモートフレーム

　リモートフレームは、データフレームの要求に使用され、基本構造はデータフレームからデータフィールドを除いたもの(もしくはデータフレームでDLCを0、データフィールドが0バイトのもの)と同一になっています。リモートフレームのIDは要求するデータフレームのIDを設定し、リモートフレームのDLCは要求するデータフレームのDLCを設定する。データフレームとはRTRが異なりレセシブとなっていることから、データフレームとリモートフレームの識別を行える。

　この方式を使用すると、ノード内マイコンのリソースを送受信に占有されずに必要な時だけリソースを使用でき、バス占有率も下がる。しかし、頻繁なデータのやり取りおいては必要なデータフレームと同じだけリモートフレームが必要になるため、かえってバス占有率が上がってしまうこともある。そのため現在では、リモートフレームはほとんど使用されなくなり、データフレームを定期的に各ノードからバスに送信する方式が使われるようになった。

受け取り

選択

送信要求

準備

リモート

受信

受け取り

選択

送信

準備

データ

受信

ノードA

ノードB

ID 4

RTR = 1

ID 4

RTR = 0

データ

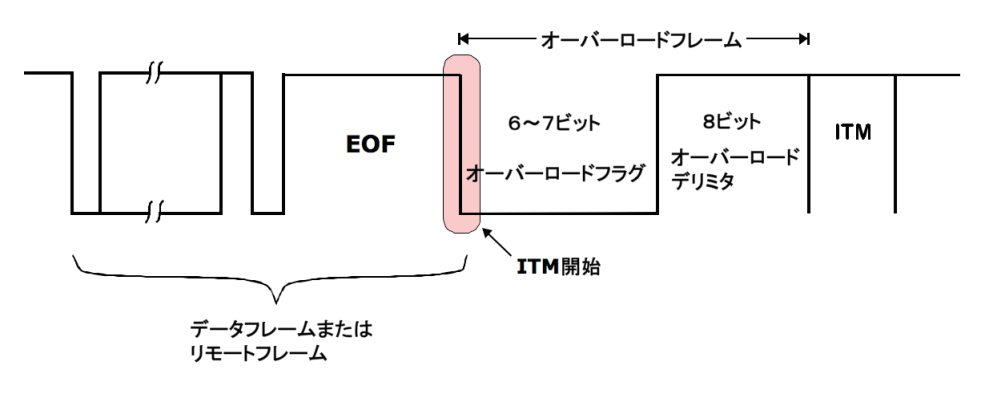
リモートフレーム

データフレーム

1. オーバーロードフレーム

　CANコントローラが前回のフレームの処理をまだ完了していない時に、次のフレームン開始を遅延させるために用いられる。データフレーム受信中に処理が間に合わないノードは、オーバーロードフレームを送信し、次に送信されるデータフレームの送信開始を遅らせることができます。

　オーバーロードフレームは、「オーバーロードプラグ」と「オーバーロードデリミタ」から構成されています。



* オーバーロードプラグ

　6ビットのドミナントから構成され、インターミッション(ITM)の最初の2ビット以内から送信が開始される。オーバーロードフラグを受信したノードは、即座にオーバーロードフラグを繰り返す。これにより、最初のオーバーロードフラグとそれにより送信されるオーバーロードフラグが重なる部分が生じ、結果としてオーバーロードフラグのビット長は7ビットになる。

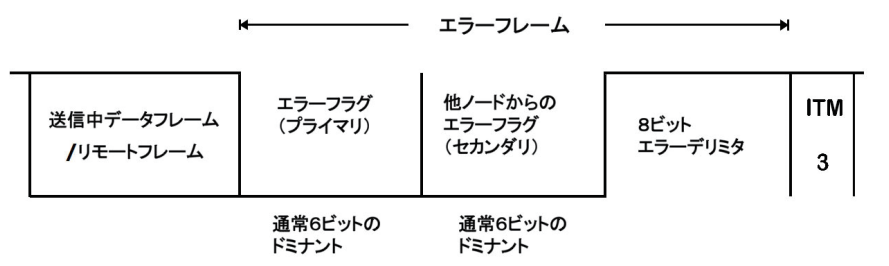
* オーバーロードデリミタ

　8ビットのレセシブから構成され、オーバーロードフレームの区切りを示す。オーバーロードフレームが送信されることにより、バスアイドル開始位置を遅らせることができ、その間に処理が間に合わないノードは処理を行うことができる。

1. エラーフレーム

　通信中に各種エラーが発生した時に送信されるフレームで、ネットワークに接続されているノードに異常を知らせる役割を果たす。

　エラーフレームは、「エラーフラグ」と「エラーデリミタ」から構成される。これらは、ビットスタッフィングルールに違反、または固定フォーム部分を破壊する形で、直近の送信を中断させる。



* エラーフラグ

　エラー発生を他ノードに知らせる目的で使用される。エラーフラグは、通常6ビットのドミナントから構成され、ビットスタッフィングルール違反を発生させる。

　すべてのCANコントローラは、エラー発生状態により内部のエラーカウンターの値を増加させる。これにより、現在ノードがどのような状態であるかを知ることができ、ハードウェア故障など深刻な状態のノードからエラーフラグが連続して送信され、結果として通信が行えなくなることを防ぐ仕組みになっている

* エラーデリミタ

　8ビットのレセシブからなり、エラーフレームを終了させる。データ、またはリモートフレームの終端における1ビットのACKデリミタと7ビットのEOFの部分と同じ、8ビットの連続したレセシブから構成されている。

1. **ダイアグ通信**

　ECUと診断テスターとの間で行われるオフボード診断のための通信方式である。統一された呼称はなく、自動車メーカーやECUサプライヤによって、ダイアグ通信、診断通信、診断データ通信、故障診断通信など様々な名称で呼ばれる。

　ダイアグ通信による基本的な診断手順は以下に示す。

1. 診断テスターから、「ECUへの指示」を伝えるリクエストメッセージを送信する
2. 指示を実行できる場合、ECUは指示された動作を行い「指示を実行したことと、その結果」を示す、ポジティブレスポンスメッセージを送信する
3. 何らかの事情で指示を実行できない場合、ECUは「指示を実行できないことと、その理由」を示す、ネガティブレスポンスメッセージを送信する

ECU 1

リクエストメッセージ**→**

**←**レスポンスメッセージ

(ポジティブ/ネガティブ)

**リクエストメッセージとレスポンスメッセージ**

※リクエストメッセージとレスポンスメッセージを合わせて診断メッセージという。

　ECUが診断テスターからの指示を実行できないケースとして、単純にリクエストメッセージの形式やデータ値が誤っていて指示を実行できない場合や、安全確保のために指示を実行できない場合（車両が走行中である等）などがある。診断テスターは、ECUからネガティブレスポンスを受信したら、これを受けて一連の診断の処理を中止する。

**62 03 8F 00 64**

**例）水温を読みだす**

ECU 1

ECU 1

**22 03 8F**

水温の読み出しを指示

水温を報告

　一般的なCAN通信の場合、1つのCAN メッセージに載せられるデータは裁断8byteである。8byte以上の長い診断メッセージの場合、分割し、複数のCANメッセージを使って伝送する方式(マルチフレーム)としてISO 15765-2の規格が採用されている。

　以下はマルチフレームの流れとFF、FC、CFのフォーマットである。

Sender

Receiver

FF:Frist Frame

FC:Flow Control

CF:Consecutive Frame

**マルチフレーム**

FF

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte#1 | | Byte#2 | Byte#3 | Byte#4 | Byte#5 | Byte#6 | Byte#7 | Byte#8 |
| Bit#7-#4 | Bit#3-#0 | Bit#7-#0 |
| 1(FF) | FF\_DL | N\_Data | | | | | | |

FC

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte#1 | | Byte#2 | Byte#3 | Byte#4 | Byte#5 | Byte#6 | Byte#7 | Byte#8 |
| Bit#7-#4 | Bit#3-#0 |
| 3(FC) | FS | BS | STmin | Reserved | | | | |

CF

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte#1 | | Byte#2 | Byte#3 | Byte#4 | Byte#5 | Byte#6 | Byte#7 | Byte#8 |
| Bit#7-#4 | Bit#3-#0 |
| 2(CF) | SN | Data | | | | | | |

1. **AUTOSAR**

　AUTOSAR (AUTomotive Open System Architecture)は、自動車業界で使用されているソフトウェアアーキテクチャの標準化プラットフォームである。

1. **標準化**

　従来の開発では、ハードウェアに応じてアプロケーションを大きく変える必要があったが、アプリケーションとハードウェアの間にAUTOSARに準拠した車載ソフトウェアがあれば、ハードウェアに合わせてアプリケーションを変える必要がなくなった。これによりアプリケーションの再利用が可能となり、効率的な開発が実現可能となった。

**アプリケーション**

**土台(ハード＋基本ソフト)**



**アプリケーション**

**ハードウェア**

**従来の開発**

**AUTOSAR準拠の開発**

　標準化によって、以下のようなメリットがある。

* アプリケーション間(ECU内、ECU間)で通信プロトコルを意識せずに連携可能
* OEMとECUサプライヤの役割分担を明確化し、連携・分業しやすくなる。

1. **構造**

　AUTOSAR Classicでは、車載アプリケーションとマイコンを切り離して開発するためのソフトウェアプラットフォーム使用を規定している。AUTOSARの使用では、アプリケーションとマイコンを切り離すために、RTE、BSWというレイヤを規定しており、BSWには機能毎に多くのモジュール使用が規定されている。さらに、AUTOSARプラットフォームを使用して開発する手法(メソドロジ)や、アプリケーション等のモデル定義に関しても詳細に規定されている。

アプリケーション層

ランタイム環境

OS

システム

サービス

オンボードデバイス

抽象化

マイクロコントローラ

ドライバ

メモリ

サービス

メモリハードウェア抽象化

メモリ

ドライバ

コミュニ

ケーション

サービス

通信ハードウェア抽象化

通信ドライバ

IO

ドライバ

I/O

ハード

ウェア

抽象化

複雑な

ドライバ

マイクロコントローラ

モデリングツール

構成

ツール

**設定**

1. **ツールチェーン**

　AUTOSAR Classic ECUの開発に、以下の実績のある汎用ツールチェーンを使用する。

System Design

PREEvision

CANseleStudio

Architecture and Communication Design

Diagnostic Specification

Application Software Development

PREEvision

vVIRTUALtarget pro

SWC Design

SWC Execution and Test

DaVinci Developer Classic

ECU Software Integration

DaVinci Configurator C;assic

TA.Design/TA.Simulation

BSW / RTE Configuration

Timing Analysis

vVIRTUALtarget basic

Virtual Integation

System Verification

CANape

CANoe & CANoe.DiVa

ECU Calibration

Verification of Network Communication

and Diagnostic Behavior

Application Software Verification

CANoe & VT System

vVIRTUALtarget pro

SWC Verification within Real ECU

SWC Verification in Virtual Enviroment

ECU Software Verification

CANoe.AMD

ECU Monitoring and Debugging

TA.Inspection

Timing Verification

AUTOSAR ECU / Syastem

SWC 2

SWC 1

SWC 3

AUTOSAR ECU / Syastem

SWC 2

SWC 1

SWC 3

AUTOSAR ECU / Syastem

RTE

SWC 1

SWC 3

Basic Software (BSW)

**MICROSAR**

|  |  |
| --- | --- |
| PREEvision | AUTOSARに準拠したシステム及びソフトウェア設計の作成をサポートする。 |
| CANdelaStudio | AUTOSAR ECUの診断使用を定義することができる。 |
| DaVinci Developer Classic | ECUのAUTOSAR SWCを設計・微調整ができる。 |
| vVIRTUALtarget pro | ECUのAUTOSAR SWCの実装を検証する。 |
| DaVinci Configurator Classic | 基本的なソフトウェアモジュールとRTEは、DaVinci Configurator Classicで構成されている。 |
| vVIRTUALtarget basic | 仮想ECUとしてvVIRTUALtargetBasicを使用して、統合されたAUTOSARコードをPC上で実行できる。 |
| TA Tool Suite | 組み込みマルチコアリアルタイムシステムの設計、シミュレーション、検証のためのツール。 |
| CANoe.AMD/XCP | このCANoeオプションを使用すると、AUTOSAR ECUの内部メモリにアクセスできる。 |
| CANoe | CANoeを使用して、AUTOSAR ECUをテストし、AUTOSARネットワーク通信を検証する。 |
| CANape | このCANoeを使用して、AUTOSAR ECUのパラメータを適合する。 |

1. ソフトウェアの更新

　自動ブレーキなどの先進運転支援システムや自動運転技術などが開発・導入され、自動車が高度にデジタル化している。そのため、品質・セキュリティの受容性が高まり、最新の車載ソフトウェアを効率よく更新することが必要である。

1. **OTA/リプログラミング**

Over The Airの略で無線通信を経由してデータを送受信することを指し、ソフトウェアの更新などを行うときに広く利用されている。以下は、OTAの更新方法である。

メーカー

OTAセンター

ECU

1. センターに登録
2. 無線で配信

③　自動更新

　車載のOTAリプログラミングでは、差分エンジンを使用している。差分エンジンは、ECUのプログラム書き換え高速化を実現する、差分書き換えパッケージである。以下は差分アップデートのメリットである

* 変更の発生した箇所だけ書き換えにより、書き換え時間を大幅に短縮
* 書き換えに用いる差分情報が新プログラムのサイズより大幅に小さく(最大99%削減)公衆回線を経由した配信等での通信量を削減
* 検知エンジンやIT機器にてOTAでの遠隔書き換えによるコストダウンや車両ユーザの利便性向上
* 車両工場における書き換えの高速化のよる生産性の向上

ダウンロード

**クラウド**

プログラム配信

差分圧縮データ

差分圧縮データ

新プログラム

新プログラム

旧プログラム

**車両**

プログラム受信

ソフトウェア更新

解答＆復元

解答

復元