組み込みマイコンの仕組みを理解しよう

第3回

FRマイコンの周辺機能

平石 郁雄

本連載では，富士通の32ビット・マイコン「FRファミリ」を例に，マイコンの仕組みや使い方について解説している．今回は，代表的な周辺機能であるA-Dコンバータと UART/SIOについて説明する．これらの機能を使用するためのサンプル・プログラムも示す．（編集部）

「マイコンに触れたことなんてないよ」と思っている読者もいるかもしれません．しかし，文明社会で生活しているのであれば，知らないうちにマイコンと触れ合っているはずです．今日もテレビを見たり，音楽を聴いたり，エアコンの温度を調整したりした記憶がありませんか．その時に，ボタンを押したりしたはずです．それが，マイコンとのコミュニケーションになるのです(**図1**)．ボタンを押す操作をマイコンが認識して，指示された動作を行っています．

● コミュニケーション手段はアナログ/ディジタル信号

マイコンが外界とコミュニケーションする場合，アナログまたはディジタルの電気信号を用いて行います(**図2**)．例えば，ボタンに使用されるキー・スイッチや加速度，温度，湿度を感知するセンサなどからアナログ信号が出力され，そのアナログ信号をマイコン内部にあるA-D(アナロ

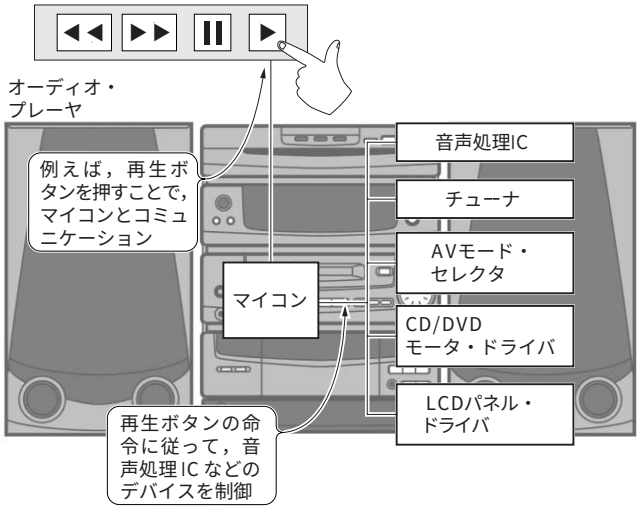


図1　マイコンとのコミュニケーション

例えばオーディオ・プレーヤの場合，再生ボタンを押したことにより，マイコンへの入力信号が変化する．その変化をマイコンが検知し，指示されたとおりに音声処理ICなどを制御する．また，再生を開始したことをLCDパネルなどに表示する．マイコンは人間とコミュニケーションしながら，指示されたとおりに動作する．

CAN　：Controller Area Network
I^2C　：Inter Integrated Circuit
LIN　：Local Interconnect Network
OSD　：On-screen Display
PPG　：Pulse Pattern Generater
PWC　：Pulse Width Counter
SIO　：Serial Input Output
UART：Universal Asynchronous Receiver Transmitter
UDC　：Up Down Counter
USB　：Universal Serial Bus

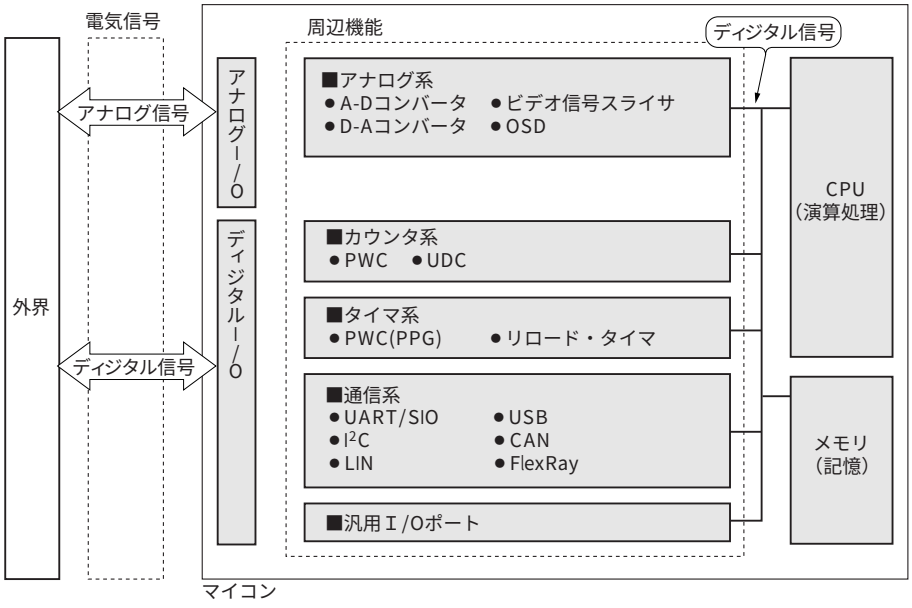
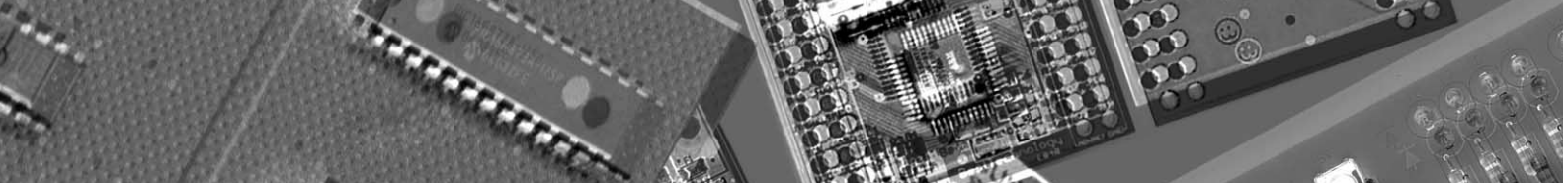


図2
周辺機能

周辺機能には，アナログ系，カウンタ系，タイマ系，通信系があり，アナログ信号またはディジタル信号を用いて外界にアクセスする．一般にCPUはアナログ信号を直接処理できないので，マイコン内部ではディジタル信号を用いて伝達する．

KEYWORD ── マイコン，FRファミリ，A-Dコンバータ，逐次比較型，比較回路，サンプル・ホールド回路，UART/SIO，RS-232-C，ボーレート，CAN，LIN，MOST，IDB-1394，FlexRay



グ-ディジタル）コンバータへ入力します．一般的なCPUはアナログ信号をそのまま処理することができないため，アナログ信号をディジタル信号へ変換してから処理します．

また，マイコンとLCD（液晶ディスプレイ）パネル・ドライバなどとのコミュニケーションにはディジタル信号が使用されます．UART（Universal Asynchronous Receiver Transmitter）/SIO（Serial Input Output）などの機能を用いて通信を行います．

A-Dコンバータや UART/SIOはマイコンの「周辺機能」と呼ばれます．本稿では，こうしたA-Dコンバータと UART/SIOについて解説します．

● 組み込みマイコンに使用される逐次比較型

A-Dコンバータの種類には，逐次比較型や直並列型，⊿-Σ型などがあります．組み込みマイコンの場合，コスト（回路規模）と性能の関係が重視され，一般に回路規模がコンパクトで変換時間が高速な逐次比較型を搭載する傾向があります．逐次比較型は，D-A（ディジタル-アナログ）変換回路と比較回路，A-D変換結果レジスタ，A-D制御回路，サンプル・ホールド回路によって構成されます（**図3**）．D-A変換回路からの出力電圧と入力電圧を比較することで，入力電圧値を測定します．例えば，A-D変換結果レジスタが2ビットで構成され，入力電圧が0～3［V］の範囲だったとすると，分解能は以下の通り1LSB＝1［V］になります（**図4**）．LSBとはLeast Significant Bitの略で，最下位ビットのことです．

$$1\mathrm{LSB} = \frac{(\text{最大電圧}-\text{最小電圧})}{(\text{変換結果最大値}-\text{変換結果最小値})} = \frac{(3[\mathrm{V}]-0[\mathrm{V}])}{(\mathrm{b}\,'11-\mathrm{b}\,'00)} = 1[\mathrm{V}] \quad \cdots\cdots(1)$$

2.0［V］や2.1［V］のような連続的な電圧値はb '10という値に量子化され，非連続的なディジタル値に変換されます．

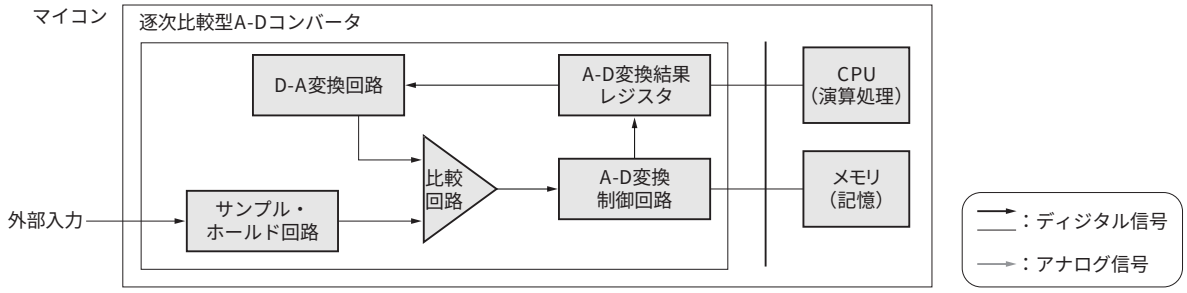
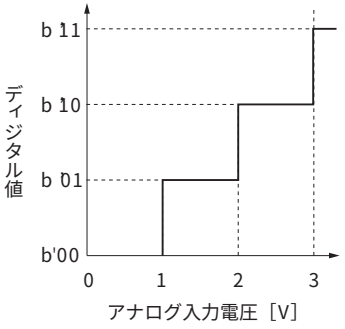


図3　逐次比較型A-Dコンバータ

まず，外部からの入力電圧をサンプル・ホールド回路でサンプリングし，その値を保持する．A-D制御回路によって，A-D変換結果レジスタの最上位ビット（MSB）が‘1’に設定される．その設定値に相当する電圧値がD-A変換回路から出力される．サンプル・ホールド回路に保持した電圧値とD-A変換回路からの電圧値を比較回路で比較し，入力電圧が大きければA-D変換結果レジスタの次のビットを‘1’にする．小さければ，前に設定したビットを‘0’に戻し，次のビットを‘1’に設定して比較する．この比較動作を最下位ビット（LSB）まで繰り返すことで，入力電圧値を測定する．

図4
2ビットA-Dコンバータの動作

2ビットのA-Dコンバータがあった場合，連続的なアナログ電圧は分解能1LSB＝1［V］の非連続なディジタル値へ変換される．例えば，外部から2.1［V］のアナログ電圧を入力した場合，まず，①のようにA-D変換結果レジスタの最上位ビットD1を‘1’に変更する．D-A変換回路から2.0［V］の電圧が出力され，比較回路でアナログ入力電圧2.1［V］と比較する．2.0［V］以上なので，D1は‘1’を維持する．次に，②のようにD0を‘1’に変更する．D-A変換回路から3［V］の電圧が出力され，比較回路でアナログ入力電圧2.1［V］と比較する．3.0［V］以下なので，③のようにD0は‘0’に変更される．その結果，A-D変換結果レジスタにはb '10のディジタルの測定値が残る．



（**a**）アナログ入力電圧とディジタル値の関係図

（**b**）A-D変換結果レジスタ

（**c**）2.1［V］を入力したときの動作例

● 実際の A-D コンバータを見てみよう

実際のマイコンに搭載された A-D コンバータはもう少し複雑です．その一例を**図 5** に示します．組み込みマイコンはコスト制約が厳しいので，D-A 変換回路や比較回路，A-D 制御回路，サンプル・ホールド回路において，各アナログ外部入力を共有して使用します．

また，A-D コンバータのレジスタとしては，A-D 変換結果レジスタと A-D 変換制御レジスタ，アナログ入力選択レジスタの 3 種類があります(**図 6**)．A-D 変換結果レジスタには，それぞれのアナログ外部入力の変換結果が 10 ビットのディジタル値として格納されます．A-D 変換制御レジスタでは，A-D 変換の起動や動作モードの設定，動作状況の確認を行います．そして，アナログ入力選択レジスタによって，変換するチャネルを選ぶことができます．

A-D コンバータを使う場合には，アナログ電圧の入力チャネルや A-D 変換の起動要因を設定します(**図 7**)．ノイ

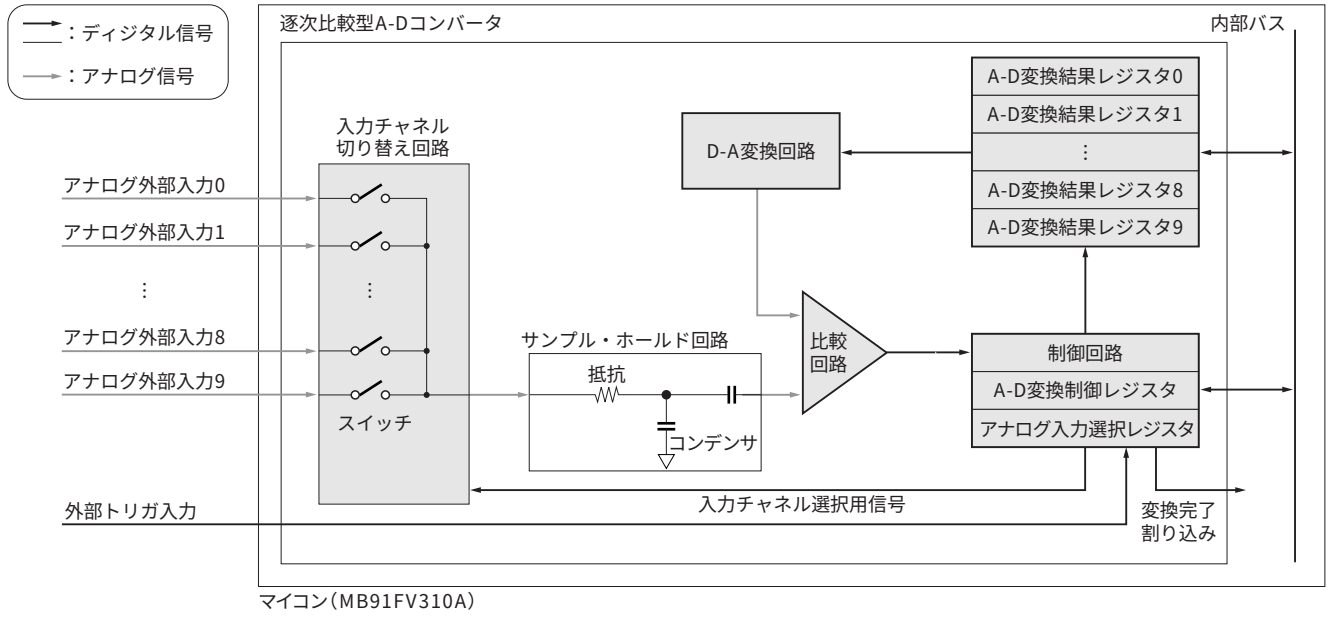
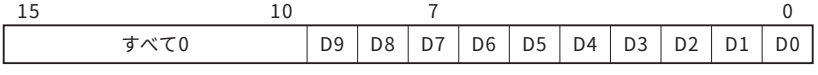
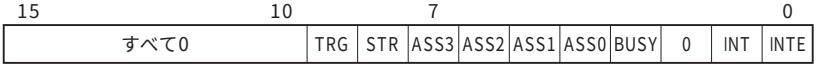


図 5　MB91FV310A の A-D コンバータ

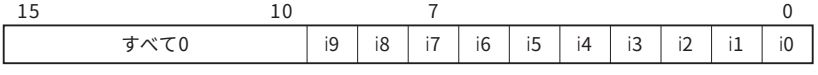
基本的な構成は，図 3 とほとんど変わらない．アナログ入力選択レジスタによって設定されたアナログ外部入力で時間をシェアしながら変換する．A-D コンバータの起動要因として，外部トリガとソフト・トリガがある．A-D 変換が完了すると，CPU に対して割り込みが発生する．



(**a**) A-D変換結果レジスタ(ADAT)0〜9



(**b**) A-D変換制御レジスタ(ADCT)



(**c**) アナログ入力選択レジスタ(ADCH)

図 6　MB91FV310A の A-D コンバータのレジスタ

それぞれのアナログ外部入力のチャネルごとに(a)のような A-D 変換結果レジスタがあり，10 ビットの変換結果を格納する．(b)の A-D 変換制御レジスタは動作モード設定や状況確認に使用できるレジスタ．TRG ビットは外部起動の設定，STR ビットはソフトウェア起動の設定，ASS3 〜 0 ビットは変換中のアナログ外部入力チャネルの状況確認，BUSY ビットは変換動作中の状況確認，INT は変換終了による割り込みフラグ，INTE は INT がセットされたときの割り込み動作許可に使用する．(c)のアナログ入力選択レジスタは，変換するチャネルを選択するために使用する．

1．アナログ電圧を入力するチャネルを選択する
- チャネル0〜9

2．A-D変換の起動要因を選択する
- ソフト・トリガ
- 外部トリガ

図 7　A-D コンバータ機能を使うときに決めること

使用するチャネルや起動要因を設定するだけで，簡単に A-D コンバータを使うことができる．ただし，実際にはノイズや誤差，インピーダンスなど，ハードウェア的な要因を考慮する必要がある．

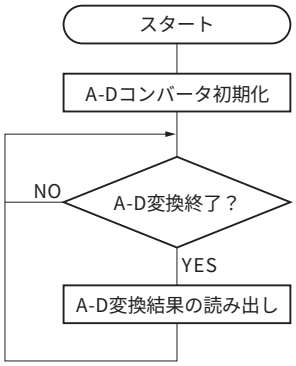


図8　A-Dコンバータのサンプル・プログラムのフロー図
割り込みなどを使用しない簡易的な例を示した．

リスト1　A-Dコンバータのサンプル・プログラム

```
/*********************************************************************/
/*   Sample program : ADC                                            */
/*   コメント : 割り込みを使用せずにA-D変換するサンプル・プログラム                  */
/*********************************************************************/

#include "_fr.h" /* 周辺機能のレジスタをC言語の変数として宣言・定義 */

/*********************************************************************/
/*   グローバル変数の定義                                                */
/*********************************************************************/
unsigned short     DATA0,DATA1; /* unsigned型の16ビット変数を宣言 */

/*********************************************************************/
/*   関数のプロトタイプ宣言                                              */
/*********************************************************************/
void     adc_init();    /* A-Dコンバータ初期化関数 */
void     adc_finish();  /* A-D変換終了関数 */

/*********************************************************************/
/*   Main関数                                                        */
/*********************************************************************/
void     main()
{
  adc_init();           /* A-Dコンバータ初期化関数の読み出し */
  IO_ADCT.bit.STR = 1; /* A-D変換制御レジスタでA-Dコンバータを起動 */
  while(1)              /* A-D変換終了待ち */
  {
    if(IO_ADCT.bit.INT==1){adc_finish();} /* A-D変換終了フラグ検出 */
  }
}

/*********************************************************************/
/*   ADC初期化関数                                                     */
/*********************************************************************/
void     adc_init()
{
  IO_ADCH = 0x03;        /* アナログ入力選択レジスタにて，アナログ外部入力1と2を入力許可 */
  IO_ADCT.bit.TRG = 0;   /* A-D変換制御レジスタにて，外部トリガでの起動禁止 */
  IO_ADCT.bit.INTE = 0; /* A-D変換制御レジスタにて，割り込み禁止 */
}

/*********************************************************************/
/*   ADC変換終了関数                                                   */
/*********************************************************************/
void     adc_finish()
{
  IO_ADCT.bit.INT = 0; /* A-D変換終了フラグをクリア */
  DATA0 = IO_ADAT0;     /* A-D変換結果レジスタ0のデータを変数に格納 */
  DATA1 = IO_ADAT1;     /* A-D変換結果レジスタ1のデータを変数に格納 */

  /* 通常は，ここに加速度の計算などを記載して処理する */
  if(DATA0!=DATA1)/* 入力電圧が一致するまで変換継続 */
  {
    IO_ADCT.bit.STR = 1;     /* A-Dコンバータを再起動 */
  }
}
```

ズや誤差を考える必要がある場合は，これまでに記載したことは重要です．しかし，そのようなことを気にする必要がない場合は，A-Dコンバータは簡単に使えます．ぜひ一度，試してみてください．

リスト1に，割り込みなどを使用しない簡易的なプログラムを記述します．処理の流れを**図8**に示します．ここではまず，A-Dコンバータ初期化関数で，アナログ入力チャネル設定と外部トリガ禁止，割り込み禁止などの初期設定を行います．次に，A-D変換を開始します．通常はA-D変換が完了するまで別の処理を行うのですが，ここではA-D変換終了フラグを検出し続けています．A-D変換終了フラグがセットされることにより，A-D変換終了関数を読み出します．忘れがちですが，A-D変換終了フラグをクリアして，変換結果を読み出します．そして，読み出した値に対して処理を行います．A-D変換を継続するのであれば，A-Dコンバータを再起動します．

● 最も一般的に使用される通信機能はUART/SIO

ほかのLSIとのコミュニケーションに使用される通信機能にはさまざまな種類がありますが，最も一般的に使用されているものとしてUART/SIOがあります（稿末のコラムを参照）．

UARTは2本の信号線を用いて通信を行う標準的な規格です．RS-232-Cのコネクタがあるパソコンであれば，マイコンとパソコンの間で通信することができます．しかし，通信フォーマットが同じであっても通信信号の電圧が違うため，電圧変換機能を持つICを間に挟む必要があります．また，最近のパソコンにはUSBコネクタしかないことが多く，その場合は，USB-シリアル変換ICや，そのようなICが入ったケーブルを使用する必要があります．

UARTでマイコンとパソコンを通信させる場合，2本の信号を利用してデータ送信とデータ受信を行います（**図9**）．デー

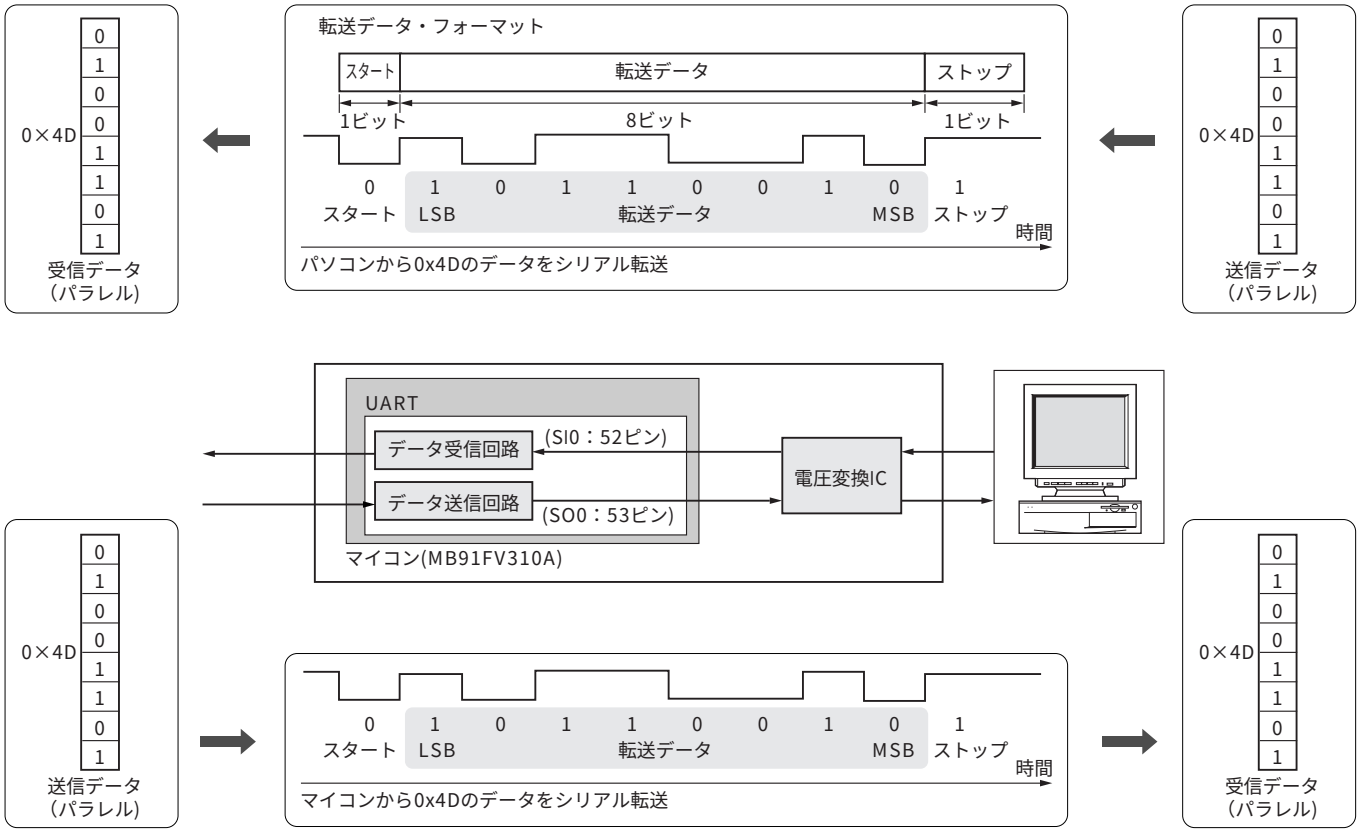


図9　UARTとパソコンの非同期通信例

マイコンとパソコンの間で非同期通信を行った場合の例．データ送信線1本とデータ受信線1本の合計2本の線で通信を行う．パソコンからの信号の電圧が12 [V] なので，電源変換ICによって3 [V] に変換して，マイコンへ入力している．転送フォーマットは，転送データが8ビット，ストップ・ビットが1ビットの例になる．全2重通信なので，パソコンからの送信データとマイコンからの送信データを同時に転送している．

タ転送では，スタート・ビット後に転送データをLSBファーストで送信し，最後にストップ・ビットを送信します．オシロスコープなどの測定器で観測するとビット・オーダリング（ビットの並び順）が逆のように見えるので，転送データの読み間違いで悩む初心者も多いようです．

● 実際のUART/SIO機能を見てみよう

UART/SIO機能の内部構成は，通信用クロックを生成する「通信クロック部」と，外部からのシリアル転送データを受信する「シリアル受信部」，外部にシリアル転送データを送信する「シリアル送信部」に分けることができます（**図10**）．これらの回路は，UART/SIO用のレジスタを用いて制御します（**図11**）．

実際にUART/SIO機能を使用する場合，考案しているシステムに応じて動作モードなどを決めることになります（**図12**）．例えばパソコンとマイコンを通信させる場合はRS-232-Cを使うので，動作モードに非同期のノーマル・モードを選択します．動作クロックは，外部で生成するよりも内部のクロックを使った方が楽なので，内部クロックを選択します．ストップ・ビットには1ビット，転送データ長は8ビット，転送方向はLSBファーストを選択します．

次に，通信のボーレートを19200 [bps] に設定します．ボーレート生成回路の動作としては，リロード・レジスタ（UTIMER）に設定した値がタイマ・レジスタ（UTIM）にロードされ，タイマ・レジスタは周辺クロック（周辺機能用の内部クロック）によってカウント・ダウンします．タイマ・レジスタがアンダ・フローすることによって，リロード・レジスタの値がタイマ・レジスタにロードされます．フリップフロップからの出力を16分周してボーレートとして使用します．これにより，ボーレートは以下の式で計算できます．

● UTIMCのUCC1＝0の場合

$$\text{通信ボーレート[bps]} = \frac{\text{(周辺クロック)}}{((2\times(\text{リロード・レジスタの設定値})+2)\times 16)}$$

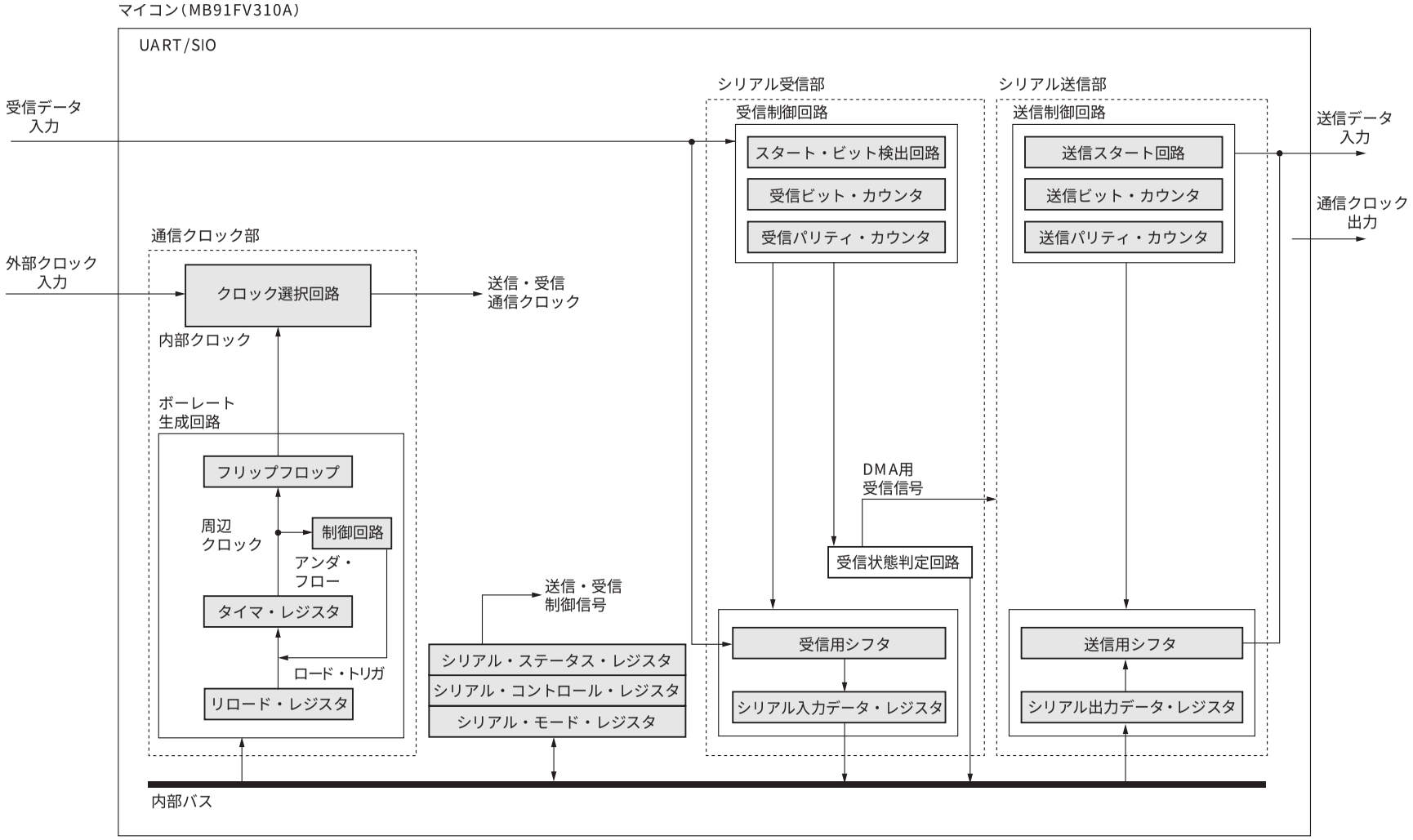


図10　MB91FV310Aの UART/SIO機能

UART/SIOの構成は，通信クロック部とシリアル受信部，シリアル送信部の三つに大きく分けることができる．クロック部では通信用のクロックを生成する．通信用のクロックを用いて，シリアル受信部とシリアル送信部がそれぞれ，シリアル転送データの送信，受信動作を行う．

15							8	7							0
b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

（**a**）タイマ・レジスタ（UTIM）0～4

15							8	7							0
b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

（**b**）リロード・レジスタ（UTIMER）0～4

7							0
UCC1	—	—	UTIE	UNDR	CLKS	UTST	UTCR

（**c**）タイマ・コントロール・レジスタ（UTIMC）0～4

7							0
MD1	MD0	—	—	CS0	—	—	—

（**d**）シリアル・モード・レジスタ（SMR）

7							0
PEN	P	SBL	CL	A/D	REC	RXE	TXE

（**e**）シリアル・コントロール・レジスタ（SCR）

7							0
PE	ORE	FRE	RDRF	TDRE	BDS	RIE	TIE

（**f**）シリアル・ステータス・レジスタ（SSR）

7							0
D7	D6	D5	D4	D3	D2	D1	D0

（**g**）シリアル入力データ・レジスタ（SIDR）

7							0
D7	D6	D5	D4	D3	D2	D1	D0

（**h**）シリアル出力データ・レジスタ（SODR）

図11　MB91FV310Aの UART レジスタ

(a)のタイマ・レジスタは，タイマ値の確認に使用する．(b)のリロード・レジスタには，通信ボーレートに応じた値を設定する．(c)のタイマ・コントロール・レジスタのUCC1は通信ボーレートの設定，UTIEはカウンタのアンダ・フロー割り込み設定，UNDRはカウンタのアンダ・フロー発生確認，CLKSはチャネル0とほかのチャネルのカスケード指定，UTSTはタイマの起動トリガ，UTCRはタイマ・レジスタのクリアに使用する．(d)のシリアル・モード・レジスタは動作モードを設定するレジスタ．MD1，MD0によって動作モードを設定し，CS0によって送信受信クロックを選択する．(e)のシリアル・コントロール・レジスタは通信プロトコルを設定するレジスタ．PENはパリティ設定，Pはパリティの偶数・奇数設定，SBLはストップ・ビット設定，CLは転送データ・サイズ設定，A/Dはマルチプロセッサ・モードのデータ形式設定，RECはエラー・フラグのクリア，RXEは受信許可設定，TXEは送信許可設定に使用する．(f)のシリアル・ステータス・レジスタは動作状況を確認することができるレジスタ．PEはパリティ・エラー，OREはオーバラン・エラー，FREはフレーミング・エラー，RDRFは受信データの有無，TDREは送信データ書き込み可能状態，BDSは転送方向，RIEは受信割り込み設定，TIEは送信割り込み設定に使用する．(g)のシリアル入力データ・レジスタには受信データが格納され，(h)のシリアル出力データ・レジスタには送信データが格納される．

●UTIMCのUCC1＝1の場合

$$\text{通信ボーレート[bps]} = \frac{(\text{周辺クロック})}{((2\times(\text{リロード・レジスタの設定値})+3)\times 16)}$$

UARTの誤差を±1%以内にする必要があるため，周辺クロックが20MHz，リロード・レジスタの値が31，UCC1＝1の条件で通信することになります．計算すると，約19230[bps]になります．通信ボーレートの設定を間違えて通信できないと悩む人も多いようです．通信できない場合には，この計算を見直しましょう．

1．動作モードを決める
- ●非同期のノーマル・モード　…普通の調歩同期通信を行うモード
- ●非同期のマルチプロセッサ・モード　…1台のホストUARTに複数のスレーブUARTを接続して調歩同期通信を行うモード
- ●クロック同期モード

2．動作クロックを決める
- ●内部クロック
- ●外部クロック

3．転送データのプロトコルを決める
- a）パリティ　有効/無効
- b）パリティの種類　偶数パリティ/奇数パリティ
- c）ストップ・ビット　1ストップ・ビット/2ストップ・ビット
- d）転送データ長　7ビット・データ/8ビット・データ
- e）転送方向　LSBファースト/MSBファースト

4．内部クロックを使用する場合，通信ボーレートを決める

図12　UART/SIO機能を使うときに決めること

初期ルーチンで設定する転送データ・フォーマットやボーレートなどを決定する．

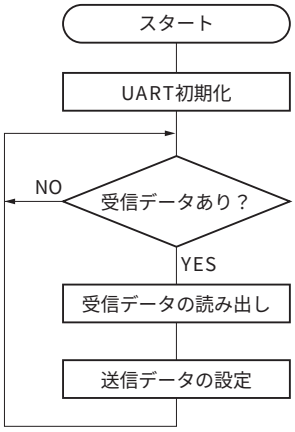
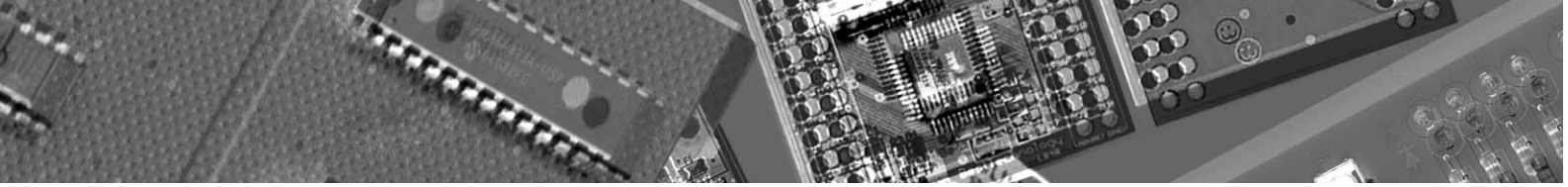


図13
UARTのサンプル・プログラムのフロー図

割り込みなどを使用しない簡易的な例を示した．

リスト2に，割り込みなどを使用しない簡易的なプログラムを記述します．処理の流れを**図13**に示します．まず，上記で決めた動作モードやボーレートなどをUART初期関数で設定します．次に，データを受信することでUART受信関数を読み出します．この関数では，受信したデータを読み出し，そのデータをそのまま送信しています．

ひらいし・いくお
富士通（株）電子デバイス事業本部 システムマイクロ事業部



リスト2　UARTのサンプル・プログラム

```
/*********************************************************************/
/*    Sample program : UART                                          */
/*    コメント : 割り込みを使用せずに通信するサンプル・プログラム                  */
/*********************************************************************/

#include "_fr.h" /* 周辺機能のレジスタをC言語の変数として宣言・定義 */

/*********************************************************************/
/*    グローバル変数の定義                                                */
/*********************************************************************/
char     DATA1;
/*********************************************************************/
/*    関数のプロトタイプ宣言                                              */
/*********************************************************************/
void     uart_init();        /* UART初期化関数 */
void     uart_receive();     /* UART受信関数 */

/*********************************************************************/
/*    Main関数                                                       */
/*********************************************************************/
void     main()
{
  uart_init();               /* UARTの初期化関数読み出し */
  while(1)                   /* UART受信待ち */
  {
    if(IO_SSR0.bit.RDRF==1) /* UARTの受信データ有無確認 */
    {
      uart_receive();        /* UART受信関数読み出し */
    }
  }
}

/*********************************************************************/
/*    UART初期化関数                                                  */
/*********************************************************************/
void     uart_init()
{
  IO_PORT.IO_PFR1.bit.UART0 = 1; /* 端子をUART0が使えるように設定 */
  IO_SMR0.bit.MD = 0;        /* 非同期，内部クロック */
  /* 通常はバイトで一括書き込みするが，説明のためにビットごとに設定 */
  IO_SCR0.bit.PEN = 0; /* パリティなし */
  IO_SCR0.bit.P = 0;   /* 偶数パリティ */
  IO_SCR0.bit.SBL = 0; /* 1ストップ・ビット */
  IO_SCR0.bit.CL = 1;  /* 転送データ長8ビット */
  IO_SCR0.bit.REC = 0; /* エラー・フラグ・クリア */
  IO_SCR0.bit.RXE = 1; /* 受信動作許可 */
  IO_SCR0.bit.TXE = 1; /* 送信動作許可 */

  IO_UTIM0 = 31;             /* リロード・レジスタに31を設定 */
  IO_UTIMC0.bit.UCC1 = 1;    /* UCC1 = 1 */
  IO_UTIMC0.bit.UTST = 1;    /* カウント開始 */
}

/*********************************************************************/
/*    UART受信関数                                                    */
/*********************************************************************/
void     uart_receive()
{
char     buf;
  DATA1 = IO_SIDR0;     /* 受信データの読み出し */
  /*通常はシステムに応じてエラー処理*/
  buf = IO_SSR0.byte;  /* エラー・フラグ読み出しだけで処理していない */
  IO_SIDR0 = DATA1;     /* 受信データをそのまま送信 */
}
```

コラム

自動車用マイコンは専用の通信制御機能を搭載

生原 浩士

UART以外にも，用途によりさまざまな通信規格が存在します．特に自動車用マイコンは，独特の通信制御機能を搭載することが少なくありません．自動車に搭載されるシステム間の通信には，車内LAN用として幾つかの規格が存在します．

● ECUが連携動作するために車内LANが必要

では，車内LANはなぜ必要なのでしょう？

最近の自動車には，CO_2排出量の少ないエンジン，コンパクトなボディに快適な車内空間を実現するエアコン，カー・オーディオやカーナビ，暗くなると自動的に点灯するヘッドライトなど，さまざまな機能があります．最新の自動車にはこれらの多くの安全・快適機能が搭載されており，その一つ一つの制御を行うためにマイコンが搭載されています．

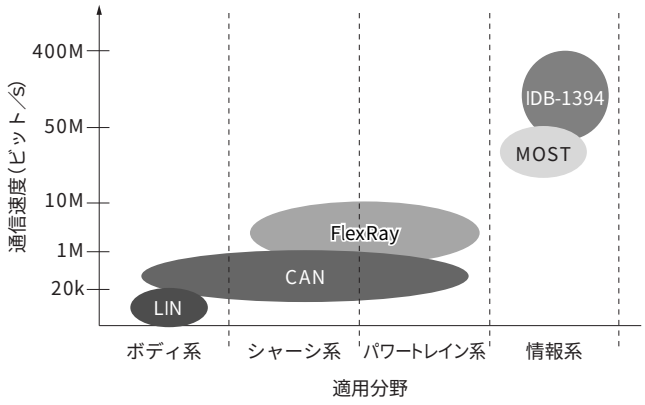
これらの安全・快適機能は，ECU（Electronic Control Unit）と呼ぶ電子装置が実現しています．ECUにはマイコンやセンサなどが搭載されており，車の各機能を電子制御します．マイコンはECU単独の機能を実現するだけでなく，複数のECUが結び付いて連携動作し，複雑なシステム全体の制御を行うケースもあります．

例えば，シフト・ギアをバック（リバース）に入れると，カーナビの画面がバック・モニタ・カメラの映像に切り替わり，同時にサイド・ミラーが下を向くといったようなケースです．この場合，ギア制御，モニタ制御，カメラ制御，ミラー制御を行う各ECUが連携します．そして，このような連携制御のために各ECUを通信ネットワークで結ぶ目的で車内LANが搭載されるのです．

● 用途に合わせて複数の車内LAN規格を利用

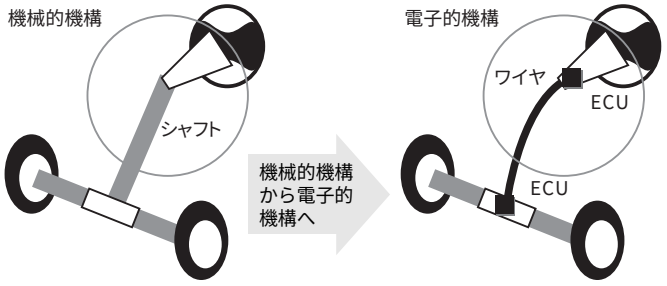
車内LANにはどのような種類と用途があるのでしょう？

車内LANの用途は大きく二つあります．一つは走行や車体に関係する「制御系」です．もう一つは，カーナビやカー・オーディオなどの機器をつなぐ「情報系」です．



図A　用途別に規格されている車内 LAN

制御系（ボディ系，シャーシ系，パワートレイン系）の車内 LAN は数十 k ビット/s の通信速度しか必要としないが，情報系ではオフィス用 LAN 並みの数十 M ビット/s の通信速度が求められる．ボディ系では現在，主に CAN が使用されており，その補助に低速で安価な LIN が使用されている．情報系では通信速度の高い MOST が使用され始めている．今後，制御系では FlexRay の普及が期待されている．情報系でもさらに高い通信速度を持った IDB-1394 の使用が検討されている．



（**a**）従来のステアリング　　（**b**）「Steering-by-Wire」によるステアリング

図B　Steering-by-Wire システムの例

Steering-by-Wire とは，シャフトやギアなどを用いた機械的機構の代わりに，ECU やモータ，アクチュエータなどを車内 LAN（ワイヤ）でつないだ電子的機構を用いたステアリングのことである．

このうち制御系は，さらに 3 系統に分かれます．すなわち，1）エンジンなどの「パワートレイン系」，2）サスペンションやステアリングなどの「シャーシ系」，3）エアコンやドアなどの「ボディ系」です．

これらの用途ごとに，数種類の車内 LAN が規格されています（**図A**）．例えば制御系では現在主に CAN（Controller Area Network）が使用されています．最近ではその補助に低速で安価な LIN（Local Interconnect Network）を組み合わせるケースも増えてきています．情報系では高速な MOST（Media Oriented Systems Transport）が使われ始め，さらに高い通信速度を持った IDB-1394 の使用も検討されています．

● 高い信頼性が求められる X-by-Wire

また，一部の CAN を置き換える次世代 LAN として，FlexRay が規格されています．FlexRay は「X-by-Wire（エックスバイワイヤ）」と呼ばれる制御系システムの実現手段として検討されています．X-by-Wire システムとは，従来のシャフトやギアなどの機械的機構の制御に替えて，モータやアクチュエータなどを制御するマイコンを車内 LAN でつなぎ，電子制御を行うシステムのことです．X-by-Wire によって，機械的機構を削減することができるため，車両を軽量化したり，車内空間を拡大したり，デザインの自由度を高めたりすることができます．

その一方で X-by-Wire によって**図B**のようなステアリング制御を実現しようとした場合，極めて高い安全性が求められます．このため，FlexRay の通信規格には通信方式や通信経路の冗長性などの面で，信頼性の高い通信を実現するためのさまざまな知恵が盛り込まれています．

これら通信制御のために，自動車用マイコンには車内 LAN 通信を制御するための CAN コントローラ，LIN コントローラ，FlexRay コントローラなどが搭載されており，通信制御をハードウェアで支援する機能が充実してきています．自動車では，マイコンによる個々の ECU 制御だけではなく，ECU の連携によるシステム全体の制御が不可欠です．そして，そのために必要な「車内ネットワーク制御機能」は，自動車用マイコンの最も重要な機能となってきています．

はいばら・ひろし
富士通（株）電子デバイス事業本部 自動車事業部