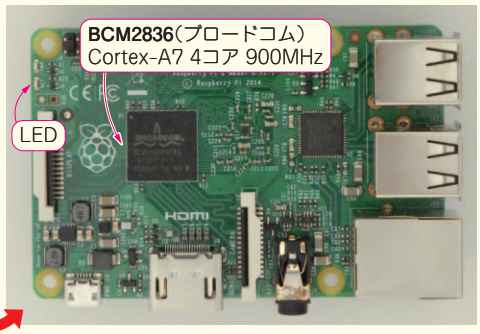
緊急速報!

Wi-Fi&Bluetooth無線付き!
1.2GHz/4コア/64ビットCortex-A57炸裂!

64ビット高性能ラズベリー・パイ3

森岡 澄夫

(a) ラズベリー・パイ3



コネクタ位置などはラズパイ2とほぼ同じ

(b) ラズベリー・パイ2

写真1 ラズベリー・パイ3はラズベリー・パイ2とそっくり

ラズベリー・パイ3が2016年2月に発表されました．残念ながら日本では技適の問題のために発売時期が少し遅れたようです(2016年4月時点では秋月電子通商などで入手可能)．

筆者はラズベリー・パイ生誕の地である英国に居住しており，発表直後に実機を入手して，動かすことができたので様子を速報します．特に初代と比べると桁違いに性能が向上しています．

特徴

● 外観はほとんど変わらない

ラズベリー・パイ3とラズベリー・パイ2の外観を**写真1**に示します．

搭載されているチップやコネクタの形状や，部品配置はそっくりで，一見では両者を区別できません．よく見ると，ラズベリー・パイ2でGPIOピン・ヘッダ近くにあった二つのLED(PWR，ACT)がラズベリー・パイ3では電源コネクタ付近に移動しています．ほかにはSoCの型番やシルク印刷のボード名が異なることが分かる程度です．

● 最も変わったのはCPU性能と無線通信機能

公開されている仕様のうち，変化のある項目を**表1**に示します．改良の力点が処理性能の向上にあるのは明らかで，値段がほとんど上がらないまま性能が急激に良くなってきています(わざわざラズベリー・パイ1を使う理由が思いつかないほど)．

ラズベリー・パイ1からラズベリー・パイ2ではCPUがマルチコア化しました． ラズベリー・パイ2からラズベリー・パイ3では64ビット・コアになりました．いずれの場合も，クロック周波数も上がっています．

ラズベリー・パイ3の発表時には「初代と比べ最大10倍速くなった」ともいわれました．筆者が評価した結果については後述します．

CPU以外に顕著に変わったのは，Wi-FiとBluetoothの二つの無線インターフェースが標準で搭載されたことです．ラズベリー・パイによるIoTデバイス・プロトタイピングも珍しくなくなってきたので，時流に合わせた変化でしょう．

無線通信機能については，速度評価のようなことはしていませんが，普通に使えています．

● ソフトウェアは概ねそのまま走る

緻密なテストを行ったわけではありませんが，ラズ

新人さんはココから！音声信号処理のススメ

編集部

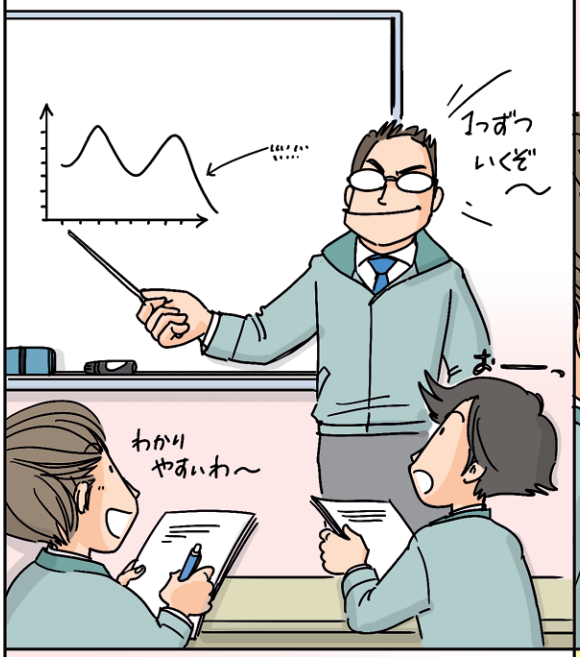
音声信号処理から始めるメリット

メリット① ウェアラブル端末とか新しいモノが作れる

メリット② 体験しながら基本を習得しやすい

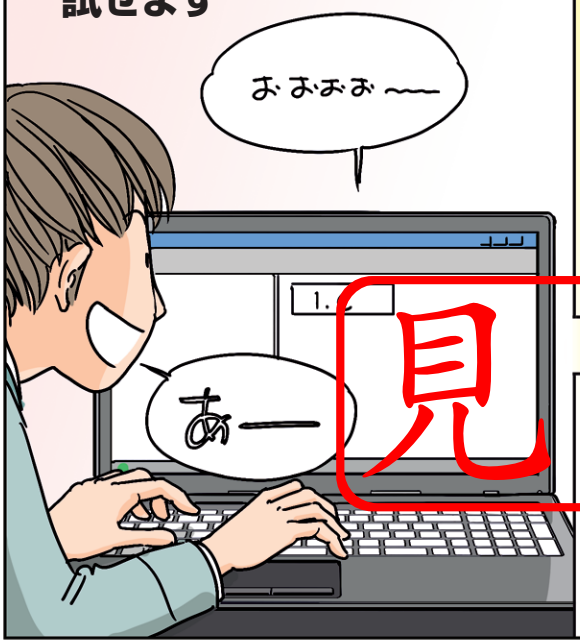
特集のコンセプト 新人さんもOK！

①なるべく細かく分けて一つずつ解説！



②exeを用意！全信号処理プログラムを試せます

③Cソースを用意！全信号処理を改造して試せます



④全プログラムは付録CD-ROMに収録

第1章 ファイルの読み書きから録音まで

音声データの基本操作

川村 新

1-1：wav音声ファイルの読み込み

収録フォルダ：44_WAV_reading

リスト1　wavファイルの読み込みプログラムDD_wav_reading.c（抜粋）

(a) wavファイル読み込み用変数の宣言部

```
unsigned short tmp1;// 2バイト変数
unsigned long  tmp2;// 4バイト変数
```

(b) wavヘッダ・ファイルの表示部

```
printf("Wave data is\n");
fseek(f1, 22L, SEEK_SET); // チャネル情報位置に移動
fread ( &tmp1, sizeof(unsigned short), 1, f1);
                    // チャネル情報読み込み 2バイト
ch=tmp1;            // 入力チャネル数の記録
fread ( &tmp2, sizeof(unsigned long), 1, f1);
                    // サンプリング周波数の読み込み 4バイト
Fs = tmp2;          // サンプリング周波数の記録
fseek(f1, 40L, SEEK_SET); // サンプル数情報位置に移動
fread ( &tmp2, sizeof(unsigned long), 1, f1);
                    // データのサンプル数取得 4バイト
len=tmp2/2/ch;      // 音声の長さの記録 (2バイトで1サンプル)

printf("Channel       = %d ch\n",  ch);
                    // 入力チャネル数の表示
printf("Sample rate   = %d Hz \n", Fs);
                    // 入力サンプリング周波数の表示
printf("Sample number = %d\n",     len);
                    // 入力信号の長さの表示
printf("\nPush any key\n");
getchar();
fseek(f1, 44L, SEEK_SET); // 音声データ開始位置に移動
```

(c) メイン・ループ内 Signal Processing 部

```
y[t]=s[t];          // 出力=入力
printf("16bit= %d\t Normalized= %f\n",input,s[t]);
                    // 観測信号を表示
```

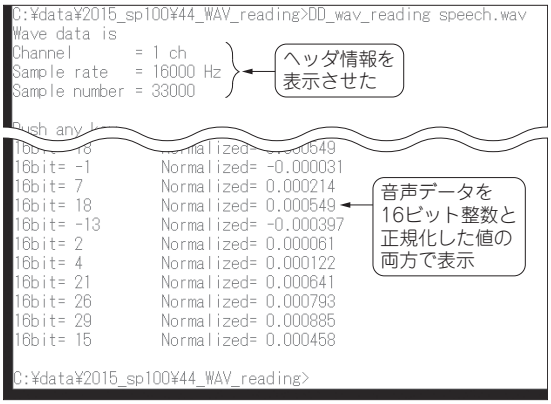


図1　wavファイルの読み込みプログラムの実行

音声処理でよく利用されるwavファイルを読み込み，テキスト・データとして表示するプログラムです．wavファイルを扱うための基本処理になります．

● 仕組み

wavファイルは，最初にサンプリング周波数やデータの数などのヘッダ情報が書かれています．その次に，音のデータが並んでいます．これらはもちろんすべてバイナリ・データ（0か1）で表現されています．

ここでは，ヘッダ情報を読み出して，主要部分を表示します．次に，バイナリ・データで書かれたデータを読み出し，テキスト・データとして表示します．

● プログラム

wavファイルを読み込むプログラムを**リスト1**に示します．

wavファイルのヘッダは，2バイトで書かれている部分と4バイトで書かれている部分があります．このため，読み込み用の変数として，2バイト用のtmp1と，4バイト用のtmp2を準備しています．

ヘッダ情報の場所は規格で決まっているので，fseek関数で該当箇所に移動し，データを読み込んで表示します．後半のデータ部も開始場所が決まっているので，そこから読み出しを開始します．

今回は16ビット・データが並んでいるという前提で読み出しを行っています．

● 実行結果

実行結果を**図1**に示します．ヘッダ情報として，1チャネルの音声で，サンプリング周波数が16000Hz，サンプル数が33000であることが表示されています．よって，音声信号は，33000/16000 = 2.0625秒であることが分かります．

続いて，各データについて，16ビット整数値（－32768 ～ 32767）と正規化した数（－1 ～ 1）の両方が表示されています．

第2章 はじめの一歩！ フィルタがわかれば何でもできる

基本フィルタ

川村 新

2-1：素通しフィルタ

収録フォルダ：00_Through

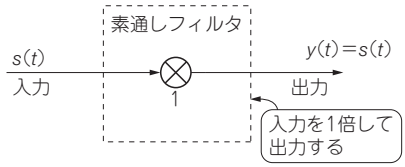


図1　素通しシステムの仕組み

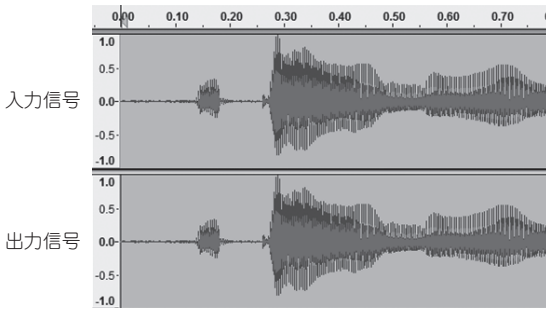


図2　入力信号がそのまま出力されている

最も簡単な音声処理の例として，あるwavファイルの信号を1倍して（つまりそのまま）出力します．出力wavファイルと入力wavファイルは一致します．

また，マイクから入力を得て，そのままスピーカから出力する素通しのリアルタイム・プログラムを作成します．

● 仕組み

入力をそのまま出力する素通しシステムを図1に示します．破線で囲まれた部分が素通しフィルタです．

中央の×マークは乗算器であり，矢印で入力される信号を定数倍して出力する役割を果たします．

● プログラム

素通しフィルタのプログラムを**リスト1**に示します．音声の入出力ファイルの取り扱いに関する説明は省きます．

変数宣言部において，時刻はtで管理し，0からスタートします．メイン・ループ内では，各時刻の処理が終了すると1ずつ増加させ，MEM_SIZEまで増加すると，再び0に戻るようにしています．入力データと出力データを格納する配列はMEM_SIZE＋1まで用意します．

メイン・ループ内では，入力として得たs[t]をそのまま出力y[t]に代入しています．これが各時刻における処理です．

リアルタイム・プログラム（RT_Through.c）でも入力データと出力データを同じ変数で定義していますので，宣言する変数と，メイン・ループ内の処理は全く同じです．

リスト1　素通しフィルタのプログラムDD_Through.c（抜粋）

（a）信号処理用変数の宣言部のプログラム

```
int       t       = 0;          // 時刻の変数
long int  t_out   = 0;          // 終了時刻計測用の変数
int       add_len= 0;           // 出力信号を延長するサンプル数
short     input, output;        // 読み込み変数と書き出し変数
double    s[MEM_SIZE+1]={0};    // 入力データ格納用変数
double    y[MEM_SIZE+1]={0};    // 出力データ格納用変数
```

（b）メイン・ループ内 Signal Processing 部のプログラム

```
y[t]=s[t];                      // 出力=入力
```

● 実行結果

実行結果を**図2**に示します．入力波形と出力波形が同一であることが確認できます．

リアルタイム・プログラムでは，マイク入力がそのまま出力されます．ハウリングを避けるために，必ずイヤホンを使用してください．

第3章 周波数領域で自由に信号処理できたらイッチョマエ

スペクトラム解析

川村 新

3-1：高速フーリエ変換FFT/逆FFT

収録フォルダ：15_1_FFT_Through

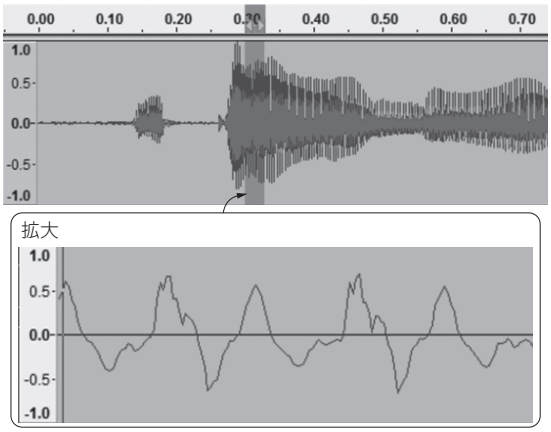


図1　音声の波形は短時間なら規則正しい繰り返しパターンになる

入力wavファイルに対して高速フーリエ変換（FFT：Fast Fourier Transform）を適用します．さらに逆FFT（IFFT：Inverse FFT）により，元の信号に復元します．

FFTによるプログラムができると，音声信号処理の幅が一気に広がります．

● 仕組み

▶周波数解析の基本は離散フーリエ変換

音声信号処理の主役級の周波数解析法は，離散フーリエ変換（DFT：Discrete Fourier Transform）です．N個の入力信号$s(t)$（$t=0, 1, \cdots, N-1$）に対するDFTは次のように定義されます．

$$X(k)=\sum_{t=0}^{N-1}s(t)\exp\left(-j\frac{2\pi k}{N}t\right) \quad \cdots\cdots (1)$$

ここで，jは虚数単位です．また，kは周波数番号を表します．k番目の周波数と実際の周波数F[Hz]は，次のように関係しています．

$$\frac{k}{N}=\frac{F}{F_s} \quad \cdots\cdots (2)$$

ここで，F_sはサンプリング周波数です．Nが大きいほど，周波数分解能は高くなりますが，その分，DFTに使用する$s(t)$のサンプル数は多くなります．

分析結果の$X(k)$から元の信号$s(t)$を得るには，次の逆DFT（IDFT：Inverse DFT）を利用します．

$$s(t)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)\exp\left(j\frac{2\pi k}{N}t\right) \quad \cdots\cdots (3)$$

▶離散フーリエ変換を高速に行う

FFTは離散フーリエ変換の高速算法です．よって，FFTの結果とDFTの結果は同じです．また，IFFTもIDFTの高速算法であり，結果は等しくなります．

FFTアルゴリズムの詳細は他の書籍に譲ります．音声に対するFFT分析は，フレームと呼ばれる短時間の区間ごとに順次実行します．これは，長時間では激しく変化する音声も，短時間（30ms程度）ならば特性がほとんど変動しないためです（**図1**）．

フレーム内の$s(t)$の始点と終点の値は通常異なりますが，このとき分析誤差が増大することが知られています．分析誤差を少なくするために，窓関数と呼ばれる関数を掛けて，フレーム内の$s(t)$の両端を滑らかにゼロにします．

また，FFT分析は，フレームの長さを半分ずつシフトして実施します．IFFTで出力を合成する場合，フレームが半分重なりますが，この部分は信号を加算することで出力を作成します．

● プログラム

FFTのプログラムを**リスト1**に示します．

今回のFFTは，$s(t)$の512サンプルを1フレームとして実行します．サンプリング周波数が16kHzなので，フレームの長さは32msに相当します．

FFTの結果は複素数になるので，実部をxr，虚部をxiの配列で表現しています．

● 実行結果

実行結果を**図2**に示します．入力波形と出力波形の

第4章 体感して信号処理のイメージをつかむ①

ボイス・チェンジャ

川村 新

4-1：再生速度の変更

収録フォルダ：50_speed_change_by_Fs

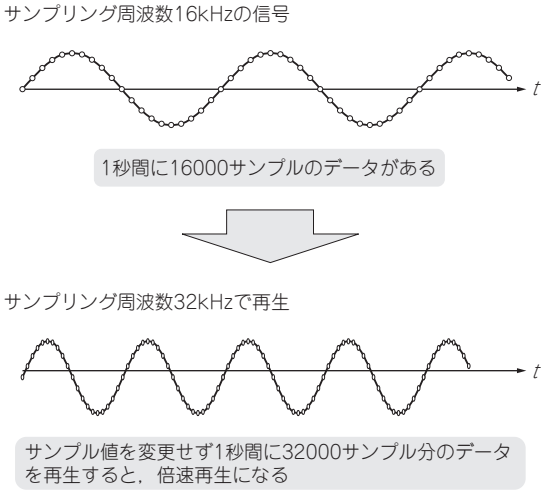


図1　16kHzサンプリングの信号を32kHzサンプリングで再生すると倍速再生になる

wavファイルのサンプリング周波数だけを変更して再生します．再生速度を変更する効果が得られます．

● 仕組み

あるサンプリング周波数でサンプリングした信号を，別のサンプリング周波数で再生すると，再生速度を変化させることができます．つまり，早回し再生や遅回し再生を実現できます．

サンプリング周波数をr倍すると，r倍の再生速度となります．例えば，16kHzサンプリング（1秒間に16000サンプル）の信号を倍の32kHzサンプリング（1秒間に32000サンプル）で再生した場合は，倍速再生になります（**図1**）．

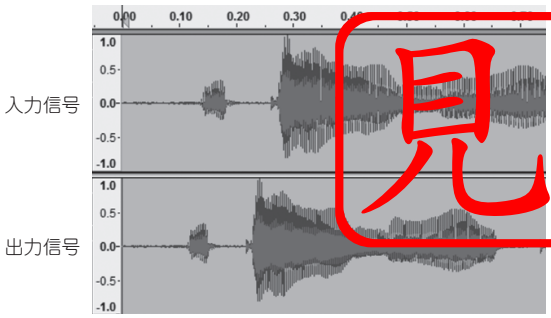


図2　サンプリング周波数を1.2倍にして再生…出力信号の再生時間が短縮されていることが分かる

リスト1　サンプリング周波数による再生速度の変更プログラム
DD_speed_change_by_Fs.c（抜粋）

```
(a)信号処理用変数の宣言部
double   y[MEM_SIZE+1]={0};        // 出力データ格納用変数
int      i;                        // 時刻の変数
double   speed_rate;               // 何倍速再生にするか

(b)出力wavファイルのヘッダ情報設定部
speed_rate    = 1.2;               // 何倍速再生にするか
Fs_out        = Fs * speed_rate;   // サンプリング・レート

(c)メイン・ループ部
x[t]=s[t];                         // x[t]を入力信号とする
```

● プログラム

サンプリング周波数による再生速度の変更のプログラムを**リスト1**に示します．

出力サンプリング周波数は，入力サンプリング周波数のspeed_rate倍になるように設定しています．単純に出力wavファイルのヘッダを書き換えるだけで速度変換が実現できます．

● 実行結果

実行結果を**図2**に示します．元の音声は，16kHzサンプリングです．出力では，サンプリング周波数を1.2倍にしています．この場合，波形の長さは1/1.2 ≒ 0.83倍になります．

試聴すると，早回し再生になっていることが確認できます．

第5章 体感して信号処理のイメージをつかむ②

エフェクト

川村 新

5-1：オート・パン…音源が左右に振られたような感覚が得られる

収録フォルダ：32_autopan

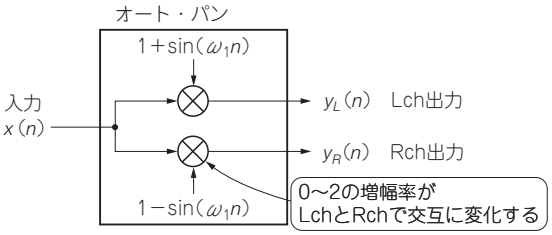


図1 オート・パン…音源を左右に振る

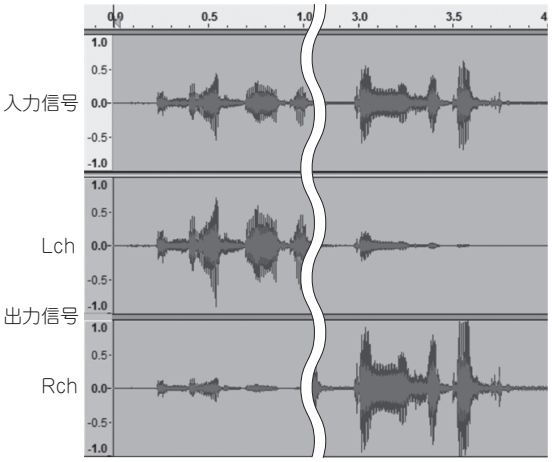


図2 LchとRchの信号振幅が互いに逆の変化になっている

モノラルの音声を入力し，ステレオ信号に変換し，さらに音源が左右に振られたようなエフェクトをかけます．

● 仕組み

オート・パンは，出力音量が増幅と減衰を交互に繰り返します．ただし，左チャネル（Lch）と右チャネル（Rch）で増幅の変化が互いに逆になります．このような信号を両耳で聞くと，音源が左右に振られたような感覚を得ることができます．

オート・パンのシステムを**図1**に示します．ここで，振幅の増幅を正弦波状に0～2で変化させます．ただし，LchとRchの増幅率は，1を中心に互いに逆の動きをします．

リスト1 オート・パンのプログラム DD_autopan.c（抜粋）

```
(a)信号処理用変数の宣言部
double   y_L[MEM_SIZE+1]={0}; // Lch出力データ格納用変数
double   y_R[MEM_SIZE+1]={0}; // Rch出力データ格納用変数
long int l = 0;               // 正弦波用の時刻管理
double   a, r;                // 正弦波パラメータ

(b)信号処理用変数の初期設定
r=0.2;                        // 正弦波の周波数

(c)メイン・ループ内 Signal Processing 部
a = 1.0 + sin(2.0*M_PI*r*l/(double)Fs);// 正弦波を更新
y_L[t] = a*s[t];   // 正弦波を入力に乗じたものをLch出力
a = 1.0 - sin(2.0*M_PI*r*l/(double)Fs);
                              // 負の正弦波を更新
y_R[t] = a*s[t];   // 正弦波を入力に乗じたものをRch出力
l=(l+1)%(10*Fs);   // 正弦波生成用の時刻
```

● プログラム

オート・パンのプログラムを**リスト1**に示します．

変数として，正弦波の周波数を`r`，乗算器の値として`a`を宣言しています．ここでは，周波数を`r=0.2`と設定しています．

ゆっくり変化させることがうまく音源を移動させるコツです．

● 実行結果

実行結果を**図2**に示します．Lchが大きいときRchは小さく，Rchが大きいときLchは小さくなっています．LchとRchの信号振幅が互いに逆の変化となっていることが分かります．

ヘッドホンで聞くと，音源が左右に移動しているように感じられると思います．

第6章 これからも応用範囲はいろいろ広いです

信号処理レベルアップ！ 適応フィルタ

川村 新

6-1：システム同定

収録フォルダ：71_SysId

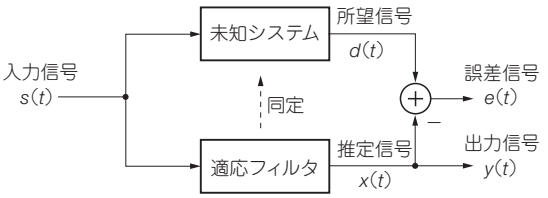


図1　未知システムと同じ構造を適応フィルタで作り出す…システム同定

リスト1　システム同定のプログラムDD_SysId.c（抜粋）

```
(a)冒頭の宣言部
#define  MEM_SIZE 16000      // 音声メモリのサイズ
#define  N        2000       // フィルタ次数

(b)信号処理用変数の宣言部
double   s[MEM_SIZE+1]={0}; // 入力データ格納用変数
double   y[MEM_SIZE+1]={0}; // 出力データ格納用変数
int      i;                 // forループ用
double   d;                 // 所望信号
double   x, e;              // 観測信号，誤差信号
double   g[N+1]={0};        // 未知系の係数
double   h[N+1]={0};        // 適応フィルタ係数
double   norm, mu;          // ノルムとステップ・サイズ

(c)変数の初期設定部
mu=0.05;                    // ステップ・サイズ
g[0]=1.0;g[N-1]=1.0;        // 未知システム(エコー)

(d)メイン・ループ内 Signal Processing 部
s[t] = input/32768.0;       // 音声の最大値を1とする(正規化)
x=0,d=0, norm=0;
for(i=0;i<N;i++){
    d = d + g[i] * s[(t-i+MEM_SIZE)%MEM_SIZE];
                                        // 所望信号作成
    x = x + h[i] * s[(t-i+MEM_SIZE)%MEM_SIZE];
                    // 適応フィルタ出力 = 所望信号のレプリカ
    norm=norm+s[(t-i+MEM_SIZE)%MEM_SIZE]*
          s[(t-i+MEM_SIZE)%MEM_SIZE]  //ノルムの計算
}
e = d - x;                  // 推定誤差
if(norm>0.00001){   // ノルムが一定値以上ならフィルタ係数を更新
    for(i=0;i<N;i++){  // NLMSアルゴリズムでフィルタ係数更新
        h[i] = h[i] + mu * s[(t-i+MEM_SIZE)%
                                MEM_SIZE] * e/norm;
    }
}
y[t] = x;                   // 推定信号を出力とする
```

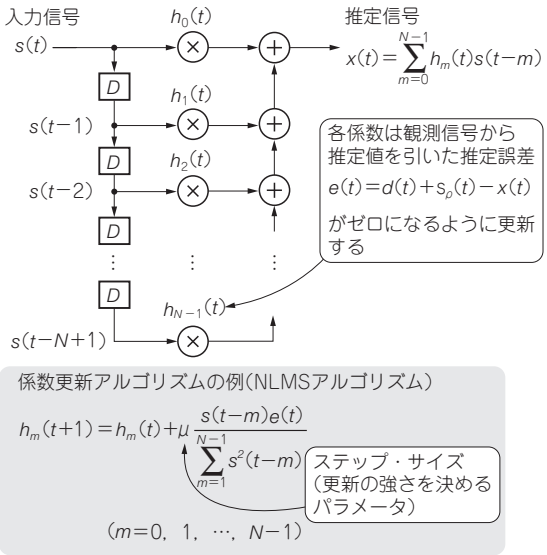



図2　適応フィルタとしてFIRフィルタを用いる

適応フィルタ（Adaptive Filter；出力と所望の信号を近づけるように自動的に設計変更を行えるフィルタ）で未知システム（入力と出力だけが観測可能で中身が全く分からないシステム）を推定することをシステム同定（Sysyem Identification）と呼びます．

● 仕組み

ある未知システムに信号を入力し，所望の信号が出力されたとします（図1）．例えば，カラオケなら，マイクに入る音声が入力で，スピーカから出るエコー付きの音声が所望の信号です．未知システムの中身は分かりません．

しかし，適応フィルタで所望の信号と同じ信号を作ることができれば，結果として未知システムと同じ構造を得たことになります．これがシステム同定です．

適応フィルタの構成を図2に示します．フィルタ係数を推定誤差が0に近づくように更新します．係数更新のための適応アルゴリズムとしては，NLMS

定番ZYBOボードによるハードウェア制御…その③

オリジナル回路用 デバイス・ドライバの動作メカニズム

鳥海 佳孝

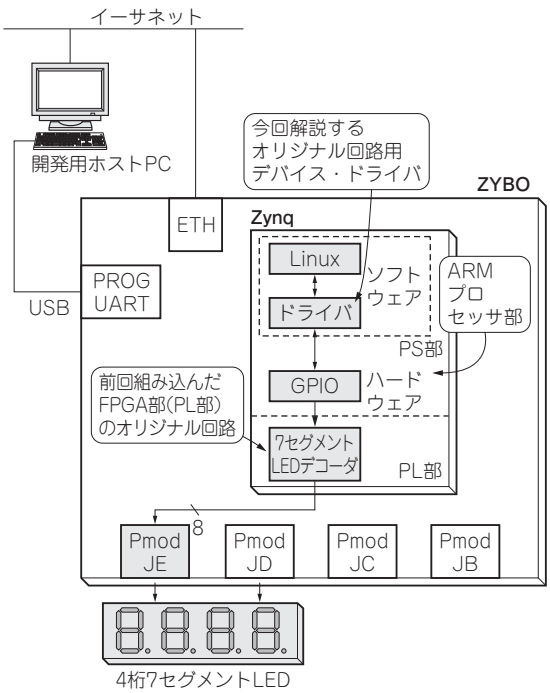


図1 今回解説すること…FPGA部オリジナル回路を動かすためのデバイス・ドライバ(GPIO)の動作メカニズム

このコーナでは，ARMプロセッサとFPGA(Field Programmable Gate Array)が1チップになったザイリンクスのZynqと，アルテラのSoC(Cyclone SoCやStratix SoC)を対象に，うまく使う方法やさまざまな話題を取り上げていきます．

2016年4月号と5月号の本コーナでは，FPGA部を使ったオリジナル回路の制御に，ダウンロードしたGPIOのデバイス・ドライバを使用していました(図1)．今回は，Linuxからオリジナル回路を動かすためのデバイス・ドライバの動作メカニズムについて説明します．Linuxデバイス・ドライバの全てを一から解説していては膨大な誌面が必要になるので，ここではまず本コーナで使用した基本的なGPIOのデバイス・ドライバに注目します．(編集部)

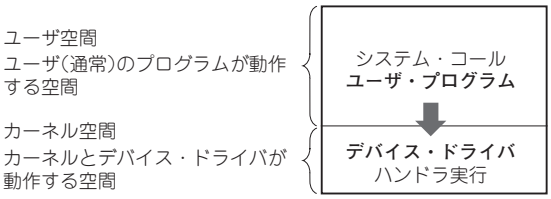


図2 デバイス・ドライバによるI/Oアクセス
ユーザ・プログラムによって実行されるシステム・コールに従ってカーネル空間で動作するデバイス・ドライバのハンドラが動作する

動作の仕組み

LinuxのようなOSにおいては，アドレス・マッピングされたI/Oを直接アクセスすることができないような仕掛けになっています．マイコン・システムとの，大きな違いの一つです．

● I/Oアクセスはデバイス・ドライバが基本

Linuxシステムにおける I/Oアクセスは，次のような方法が考えられます．

- mmapシステム・コールを用いて/dev/memを使ってI/Oアクセス
- デバイス・ドライバ使用してI/Oアクセス(図2)

mmapシステム・コールを用いた場合には，

- rootでないと実行できない
- 割り込みが使えない

などの問題があります．従って，LinuxでI/Oアクセスを行うときには，デバイス・ドライバを用いるのが王道です．

● デバイス・ドライバの呼び出し

Linuxシステムの場合，メモリ空間がユーザ空間とカーネル空間に分かれています．デバイス・ドライバはカーネル空間の中で動作しています．ユーザ空間で動作するプログラムがI/Oアクセスを行う場合は，実行されたシステム・コールに従って，デバイス・ドライバが動作します．この際，カーネルが該当するデバ

ARM FPGAザイリンクスZynqのLinux開発環境(2016年2月号)
Cでも書けるARM FPGAの回路設計環境(2016年3月号)
ARM FPGA用Linuxにデバドラを組み込む(2016年4月号)

個人で試せる！　ご購入はこちら

ダウンロード・データあります

キット発売開始！

生体（バイタル）センシング実験室

第7回　手軽で非接触OK！赤外線脈波心拍数計測

上田 智章

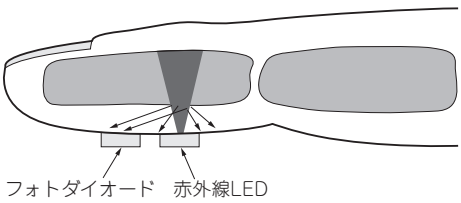


図1　赤外線LEDとフォトダイオードを利用して脈波を測定する

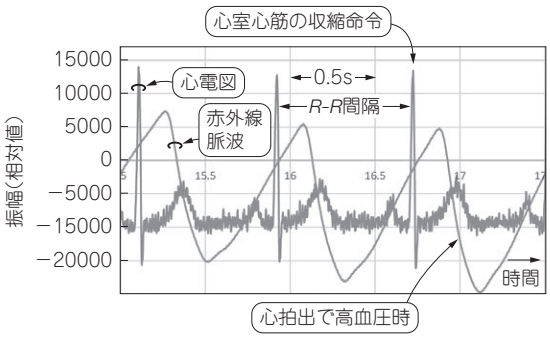


図2　心電図と赤外線脈波波形の時間関係

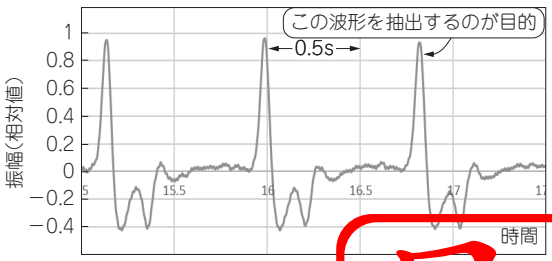


図3　赤外線脈波波形を2回微分したときのピーク間隔から心拍数が求められる

脈波波形からはR-R間隔を求められないため間隔をクリアするために2回微分した

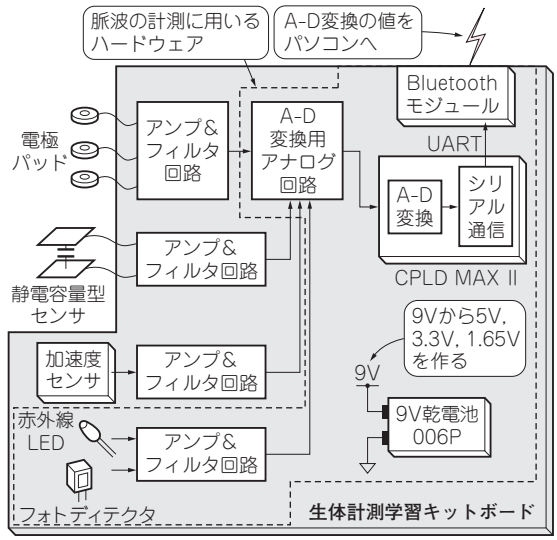


図4　連載を通して使用している生体計測学習キットボードと脈波計測に使う回路

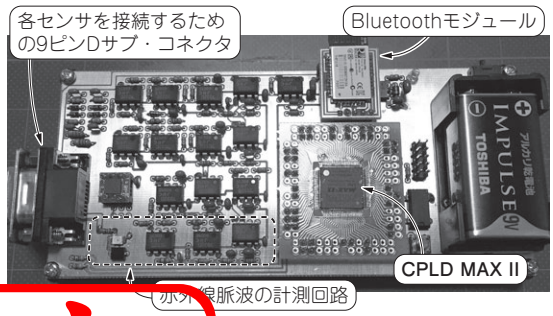


写真1　生体計測学習キットボードを用いた

● 今回やること…赤外線を使って測った脈波から心拍数を求める

図1に示すように，赤外線を皮膚に照射し，反射光の強度変化をフォトダイオードで測定することで脈波を測定できます．

図2に心電図と赤外線脈波の同時観測例を示します．心電図では鋭いR波のピーク時相を捕捉することでR-R間隔から心拍数を計算することができました．しかし，赤外線脈波はピーク位置がなだらかな変化しかなく，レベル変動も伴います．そこで，医療分野では**図3**に示すように脈波を2回微分して得られた加速度脈波を用います．心電図ほどではありませんが，鋭

第1回　心電図計測の予備知識…心臓のしくみ（2015年12月号）
第2回　バイタル・センシング実験ボード＆心電図取得用電極（2016年1月号）
第3回　心電図を取得する（2016年2月号）

さすがトルクMAXベクトル制御なら複雑な最新タイプもシンプルにサクッ！

5,000円モータ・キットで初体験！最新タイプ高効率モータIPMSM

大黒 昭宣

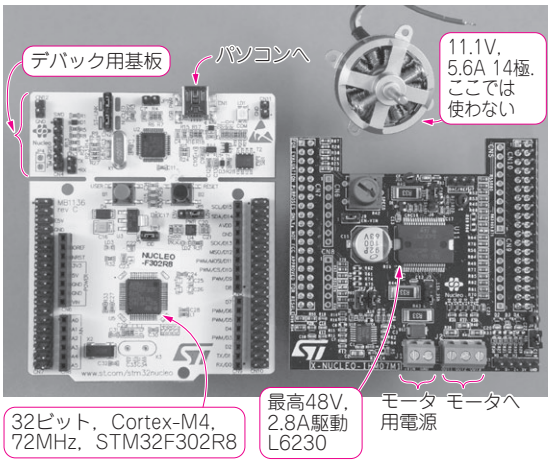


写真1　なんとDCブラシレス・モータ付きで5,000円！モータ制御入門キットP-NUCLEO-IHM001
2016年5月号で使い方を紹介

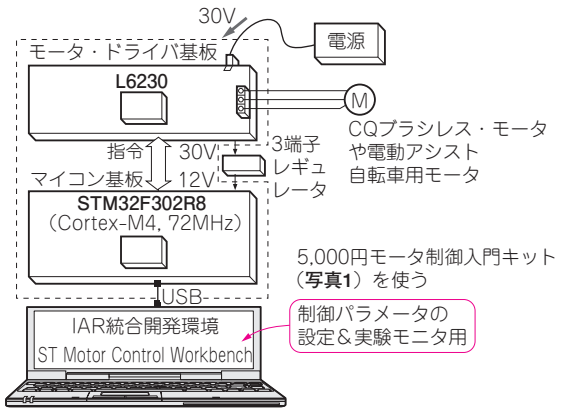


図1　実験のハードウェア構成

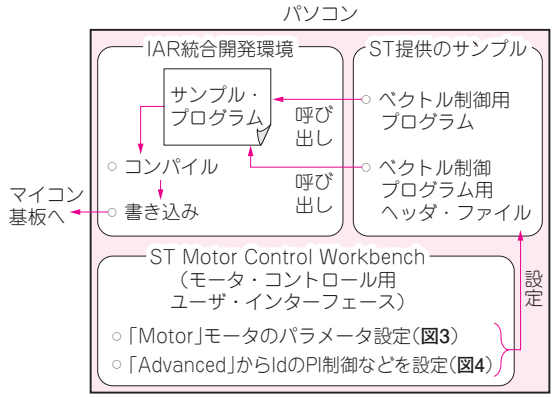


図2　ソフトウェア開発は定番統合開発環境IAR Embedded WorkbenchとSTマイクロ提供ソフトで行える

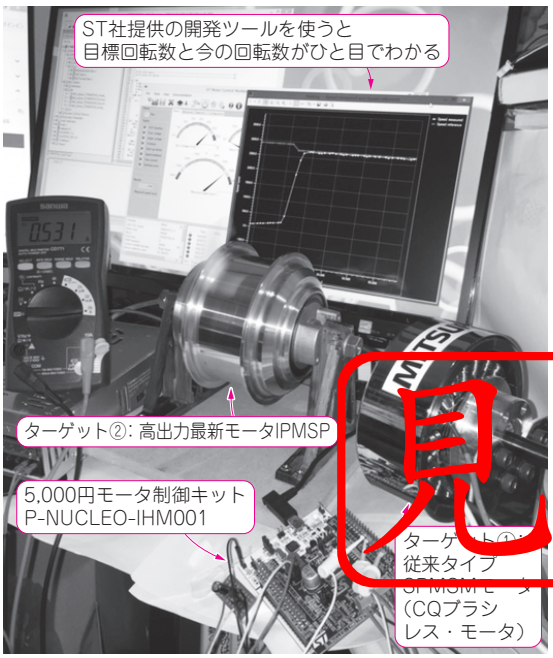


写真2　今回やること…制御は難しいが高出力が得られる最新モータIPMSMをベクトル制御を使ってできるだけ簡単に回してみる

昨年2016年5月号特集では，マイコン基板，モータ・ドライバ基板，DCブラシレス・モータが一つになった5,000円程度で入手できる入門キットP-NUCLEO-IHM001（STマイクロエレクトロニクス，写真1，コラム）を使って，電動カート向けDCブラシレス・モータ（CQブラシレス・モータ）をベクトル制御で駆動してみました．このモータは表面磁石型同期モータSPMSM（Surface Permanent Magnet Synchronous Motor）ですが，今回は埋め込み永久磁石同期モータIPMSM

ダウンロード・データあります

選び放題時代！小型モジュール＆iPhoneで試して合点！

高性能カメラ探偵団

第2回 パラメータ：SN比…暗いと目立ってくる！画像品質を表す代表値

チャート：グレー・スケール

エンヤ ヒロカズ

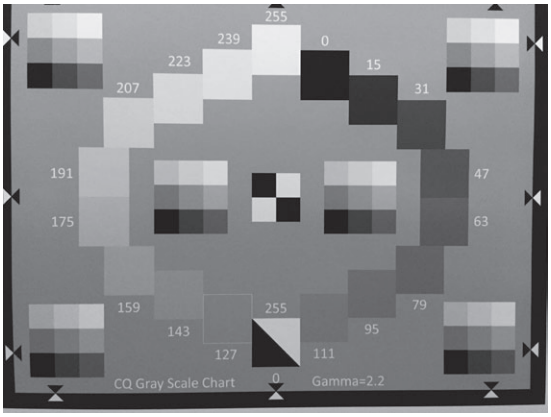


写真1　画像ノイズが測れるグレー・スケール・チャート
CQグレー・スケール・チャートENYA-01

前回（第1回，2016年5月号）はグレー・スケール・チャート（**写真1**）を紹介しました．

今回は同じグレー・スケール・チャートを用いて，画質を決める大きな要素であるノイズの測定方法を紹介します．

ターゲットのカメラは前回同様，OV5642カメラ・モジュール（http://www.csun.co.jp/SHOP/20120881201.html）とiPhoneです．

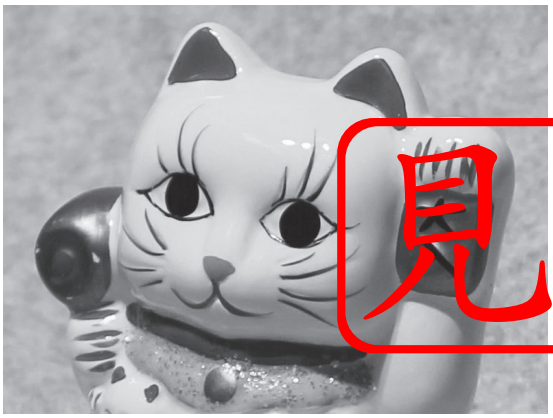
画像ノイズの基礎知識

● ザラザラした感じの画像になる

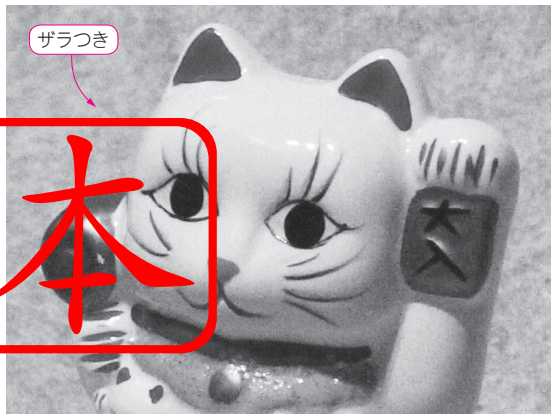
例えばオーディオ機器でノイズと言えば，サーというホワイト・ノイズやブーンという電源のハム・ノイズなど，実際の音楽演奏や会話の中に存在せず，機器内で発生したノイズのことを指します．

カメラの場合も同じで，実際の被写体には存在しないのに，撮影した画像に発生した映像成分のことをノイズと言います．**写真2**にノイズのありなしの一例を示します．実際の被写体となる**写真2**（a）は，まねき猫の表面が平たんでつやつやに光っています．ノイズを含んだ**写真2**（b）は，表面に細かい濃淡があり，ザラザラした感じの画像になっています．

これを模式的にチャートで表したものが**図1**になります．本来なら明るさに応じて画素単位で滑らかな階段状の線を描きますが，実際はノイズ成分として輝度が階段状に上下に変動しています［**図1**（b）］．



（a）ノイズなし



（b）ノイズあり

写真2　画像にノイズを含むとザラザラした感じになる

サンプル・プログラム付き!

高速ワンチップ・マイコンではじめる ソフトウェア無線

第9回　なんて不思議!? 1LSBより微弱な信号を受信して音が聞ける理由

高橋 知宏

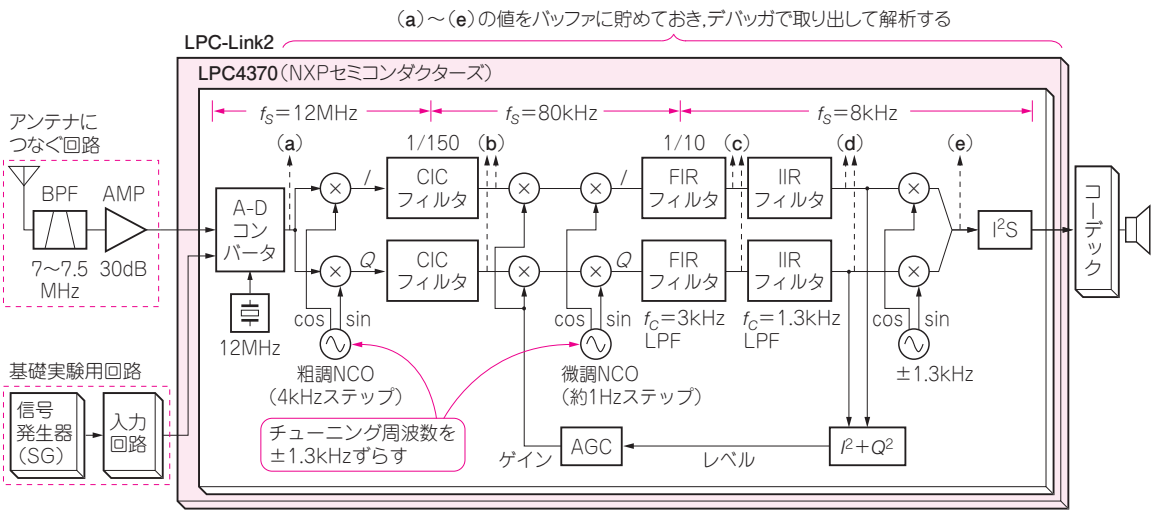


図1　今回の微小信号解析用7MHz帯アマチュア無線ソフトウェアSSB受信機の構成
それぞれの処理段階のデータを観察するために，(a)～(e)の箇所でデータを取り込む

本連載では，第5回～第8回で，7MHz帯アマチュア無線SSB(Single Side Band)を主な対象にしたソフトウェア受信機を試してきました．実際に受信してみると，想像していたよりも弱い信号でも受信することが可能です．

これまでの記事では，レベル不足対策に，30dB程度のアンプを使っていました．きちんとしたアンテナを使えば，実は，アンプを挿入しなくても受信できます．

アンテナからバンドパス・フィルタ(BPF)を通してA-Dコンバータ(ADC)に入力することでようとはソフトウェアだけで受信機を構成することができてしまうのです．たかだか12ビットのADCで，微弱なRF信号を取り込むことができてしまうのです．少し不思議に感じてしまうのですが，ADCと信号処理の性質からこのようなことが可能です．今回はこれを説明します．

今回実験で確認すること

● フルスケール0.5V/12ビットの1LSB＝122μV

LPC4370のADCは12ビットの分解能で，フルスケールは0.5Vです．0.5Vを2^{12}すなわち4096階調に分割するわけです．電圧をA-D変換した数値の1LSB(Least Significant Bit)当たりの電圧幅は，ざっくり0.5V/4096 = 122μVとなります．少なくとも122μVの変化がなければ，ADCで変換された数値は変化しないはずです．普通に考えるとADCでは1LSB未満の電圧変化は取り扱うことは不可能なように思えます．

● アンテナ入力レベル－100dBmのとき V_{P-P}＝6.3μV

一方，アンテナからの入力信号について考えてみます．一般にアンテナに入力される信号はごく微小で

第1回　AMラジオを作る(2015年10月号)
第2回　AMラジオで行う信号処理(2015年11月号)
第3回　短波ラジオを受信する(2015年12月号)

地球の裏側からもOK! 360° 見守りローバー君だぜ

ラズパイ式走るリモート探査カメラ

第8回 ラズパイ×モータ! 自走ローバー1号2号

森岡 澄夫

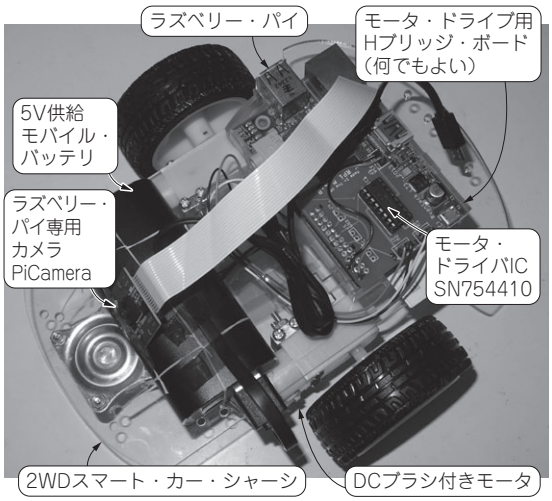


写真1 リモート・ローバー1号機
まっすぐ走るのは得意でない

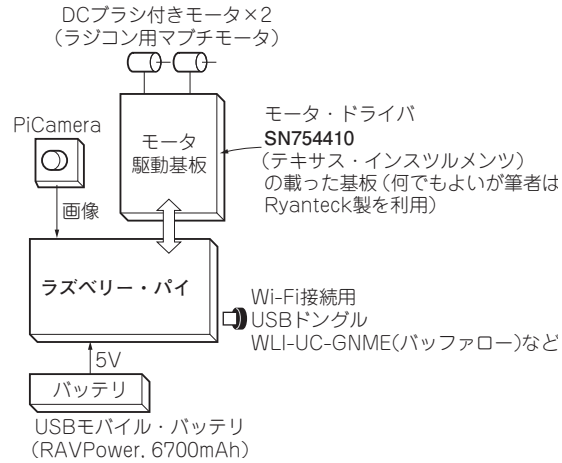


図1 いよいよモータを取りつけて自走ローバーづくりだ

今回から，第6回までに作成してきた画像伝送プラットホームを使い，本連載のメイン・テーマであるリモート・ローバーの作成に着手します．「PCからのリモート・コントロールによって走行する機能」を最初に実現し，それを改良していきます．

機能拡張では，ほかのハードウェアと組み合わせて高速I/O制御など，ラズベリー・パイの苦手分野を補強することも行います．

製作1…ラズベリー・パイでモーター制御バージョン

● 概要

まず，単純に走るだけのシンプルなリモート・ローバー(**写真1**，**図1**)の作成から始めましょう．最近，モータ駆動用の工作キットが安く入手できるようになりました．この上にラズベリー・パイを載せてモータをコントロールします．電源としては，スマホのチャージ用に市販されているUSB接続モバイル・バッテリを使います．

● パソコン操作で移動できる

これを自宅屋内で走らせているようすが**写真2**です．第6回までに作成してきた画像伝送プラットホーム(mjpg-streamerベース)を活用します(**図2**)．PCがローバーのリモコンとなり，ブラウザには車上のPiCamera(ラズベリー・パイ専用カメラ)からの動画像が表示されます．ブラウザのボタンでローバーを前

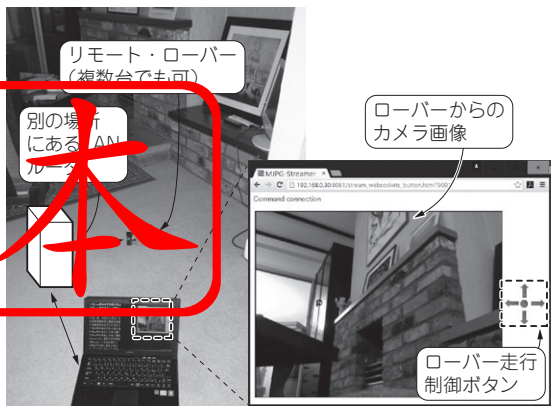


写真2 PCブラウザからリモート・ローバーを動かしているようす

目指せ高性能！ I²S & USBのクロック&データ同期入門

ラズパイ・オーディオの勘どころ

第4回　高品質再生向き！ USBアシンクロナス同期オーディオ

岡村 喜博

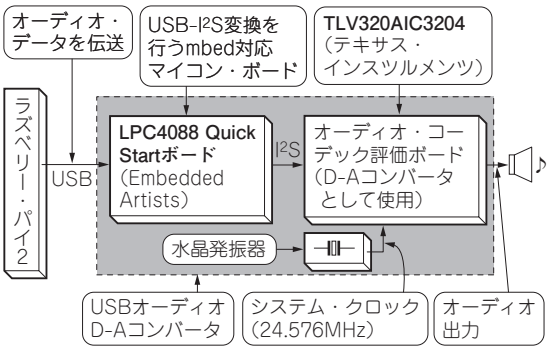


図1　実際に使うUSB接続オーディオ用D-Aコンバータ
前回の実験ではラズベリー・パイ2が決めた転送レートでオーディオ・データが送られていた．回路や使い方は前回（2016年5月号）や文献(3)参照

● 今回試すこと…USBアシンクロナス同期

前回（第3回，2016年5月号）は，ラズベリー・パイ2とオーディオ用D-Aコンバータを接続する方法としてUSBを紹介しました（**図1**）．

今回は，USBオーディオD-Aコンバータ側が要求するビット・レートに合わせてラズベリー・パイ2側からオーディオ・データを転送する（＝同期が可能になる）しくみをmbed対応マイコン・ボードに実装し，正常に動作するように修正します（**図2**）．

図2に今回のUSBオーディオD-Aコンバータを示します．ハードウェアは本連載第3回で使用したものと全く同じ構成で使用します．

USBの規格ではデータを転送する方式として四つの転送方式が用意されています．この中でオーディオ・データの転送にはアイソクロナス転送という方式が使用されます．この方式を採用することで，ラズベリー・パイ2で通常使用されるOSが標準的に持っているデバイス・ドライバを使用することができます（**図3**）．

さらに，ラズベリー・パイ2からD-Aコンバータにオーディオ・データを渡すためには転送の方式だけでなく，データの同期を考える必要があります．今回はアシンクロナス同期方式を利用しました．

USBオーディオで使われるアイソクロナス転送

● USBオーディオ・デバイスに求められること

USBスピーカなどのオーディオ機器は，USBオーディオ・デバイスとしてホスト側（ラズベリー・パイ2やパソコンなど）から認識されます．USBオーディオ・デバイスは，USBフラッシュ・メモリなどとは違い，リアルタイム転送とデータの同期が求められます．

特にデータの同期は，単にデータをとりこぼさない

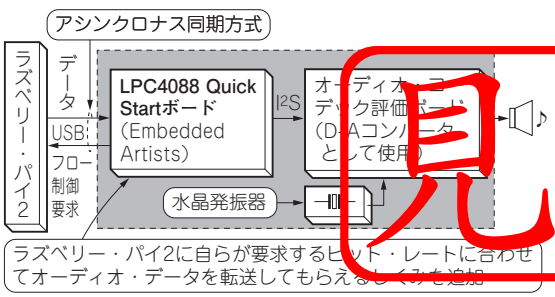


図2　今回やること…アシンクロナス方式によって同期をとる
自らが要求するビット・レートに合ったオーディオ・データをラズベリー・パイ2から転送してもらうしくみをUSB-I²S変換を行うmbed対応マイコン・ボードに追加する

図3　USBでのオーディオ・クラス，転送方式，同期方式の関係
今回は同期方式にアシンクロナス方式を採用

Windows/Mac/Linux対応でI/Oもサクッ！

オープンソースのブロック型言語 Pure Dataではじめる サウンド信号処理

青木 直史，藍 圭介

第7回 ネットワーク・プロトコルOSCを使ってiPhone操作でリモート再生

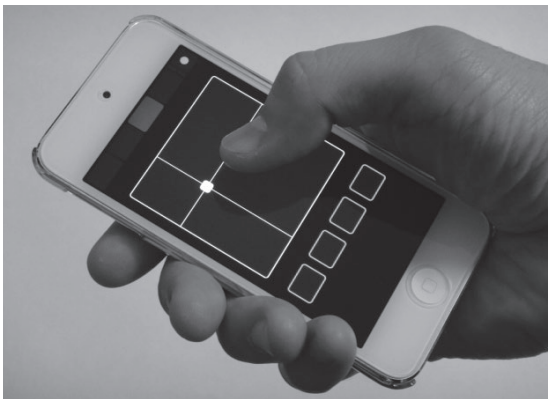


写真1　OSCという音響機器制御用ネットワーク・プロトコルを使ってスマホ（iPhone）からPure Data音源パソコンを鳴らしてみる

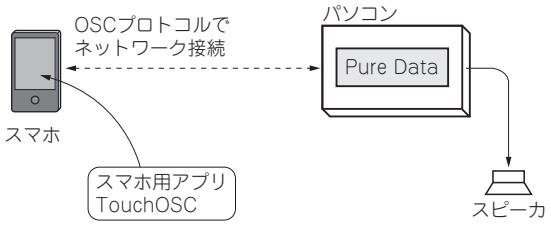


図1　今回の内容…iPhone操作でPCから音を鳴らす

PCだけでなく，さまざまな外部デバイスと連携して動作するシステムを手軽に作れることがPure Dataの魅力です．その一例として，前回は音響機器をコントロールするためのプロトコルであるMIDIを使って，MIDIコントローラからPure Dataのプログラムを動作させる方法を紹介しました．

今回は別のプロトコルとして，OSC（Open Sound Control）を紹介します．OSCは，カリフォルニア大学バークレー校のCNMAT（Center for New Music and Audio Technologies）という研究機関で開発されたプロトコルです．OSCはUDP/IPプロトコル上でネットワークに接続された音響機器をコントロールします．今回はOSCを使って，スマートフォンからPure Dataのプログラムを動作させてみます（写真1）．

MIDIの対抗馬！ネットワーク対応音響機器制御プロトコルOSC

● MIDIより自由度が高い

音響機器をコントロールするためのプロトコルといえば，やはりMIDIとの違いが気になるところですが，実は，OSCとMIDIは性格が全く異なります．

電子楽器をコントロールするため，ノート・オンやノート・オフといったメッセージの内容をあらかじめ定義しているのがMIDIの特徴ですが，OSCにはこうした決まりごとはありません．OSCには，どんなメッセージでもやり取りできる自由度の高さがあります．

● OSCメッセージの構造

OSCメッセージは，機能そのものを表す「アドレス」と機能の状態を表す「パラメータ」の二つをペアにして情報をやり取りすることが基本になっています．

例えば，スイッチに/pushというアドレスを割り当て，スイッチのONとOFFに1と0というパラメータを割り当てると，スイッチをONにしたときは/push 1，スイッチをOFFにしたときは/push 0というメッセージがやり取りされることになります．

iPhone操作でPure Dataのプログラムを動かす

● Pure DataもOSCに対応している

Pure DataもOSCに対応しているため，OSCを使うことでさまざまな外部デバイスと連携して動作するシステムを作ることができます．

もちろん，専用の音響機器にはOSCに対応したものも多数あり，具体的な外部デバイスとしてはこうしたものを利用することも選択肢の一つになります．

● スマホにはOSC用のアプリがある

しかし，もっと手軽な方法もあります．実は，

ネットワーク

さすが標準プロトコルは
本格オープンソース・ソフト完備で安心！

元祖DLNAホーム・ネットワーク
IoT時代に注目！ ×ラズパイ

第2回　DLNA/UPnPプロトコル＆通信メカニズム入門

平原 秀治

表1(1)　DLNAホーム・ネットワークを構成するプロトコル＆規定

項　目	規約/プロトコル
対応メディア形式	MPEG-2，MPEG-4 AVC/H.264，LPCM，MP3，AAC LC，JPEGなど
リンク保護	DTCP/IP
メディア転送	HTTP
メディア管理	UPnP AV 1.0など
検出/制御	UPnP Device Architecture 1.0
IPネットワーク	IPv4
物理接続	イーサネット(IEEE 802.3)，Wi-Fi (IEEE 802.11)

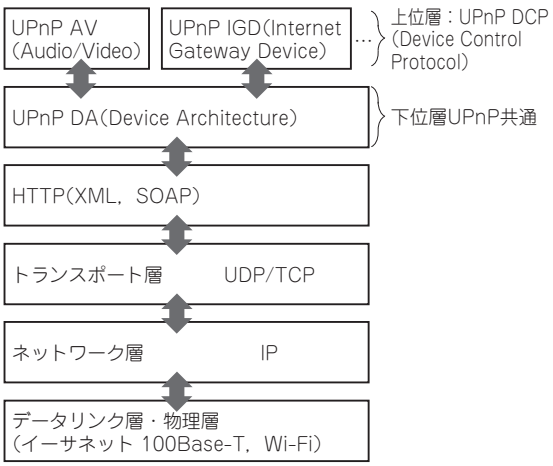


図1　DLNAの中核！UPnPプロトコルの階層構造

DLNAのネットワーク構造

● DLNAの目的：映像・音響コンテンツを利用するためのガイドラインを決める

DLNA(Digital Living Network Alliance)は，AV系のホーム・ネットワークの構成を規定します．映像・音響系のホーム・ネットワークを構成するさまざまなディジタル機器の相互接続を実現し，音楽/写真/動画といったディジタル・コンテンツを利用し合うための要件(以下，DLNAガイドライン)の策定を目的に設立された業界団体(http://www.dlna.org/)です．

DLNAは，自らが新たなプロトコル/仕様を策定するのではなく，標準の仕様を組み合わせて，使い方のガイドラインを示します．

● DLNAホーム・ネットワークを構成するプロトコル

表1に示すように，DLNAプロトコルの構成では，HTTPが，メディア転送として扱われています．UPnPより上位に位置づけられていますが，UPnPのプロトコル自体は，SOAPやXMLを使ってHTTPプロトコルでやりとりされています．

各規約やプロトコル・レイヤの上下関係はともかく，DLNAは，このように，HTTP，UDP，TCP/IPなどのプロトコルを組み合わせて使います．

中核となるプロトコルUPnP入門

● 構成要素…コントロール・ポイント＆デバイス

DLNAのガイドラインで，中核をなすプロトコルが，UPnPですので，ここではUPnPを中心に説明します(**図1**)．

UPnPでは，機器を発見し，指示を出す機能をコントロール・ポイントと呼びます(**図2**)．

コントロール・ポイントからの指示に従って動作する機能をデバイスと呼んでいます．

UPnPで定義するコントロール・ポイントやデバイスは，サービス機能単位ですので，通常TVやDVRなどの実機器と1対1の対応はせず，通常は一つの機器に複数のデバイスやコントロール・ポイントを持ちます．

特にデバイスという言葉は，UPnPでのサービス機能の意味か，実デバイスのどちらの意味で使っているか注意が必要です．

ダウンロード・データあります

パケットづくりではじめる ネットワーク入門

第11回 IPルータの課題…無限ループ対策

坂井 弘亮

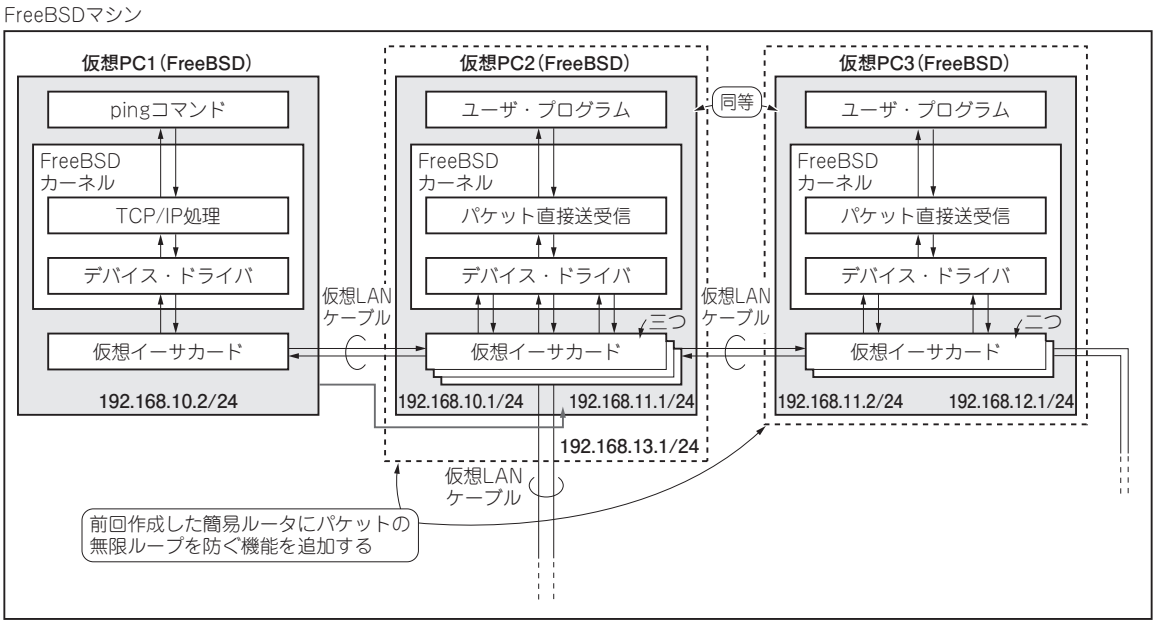


図1 パケットの無限ループ対策…TTLの減算機能とチェックサムの再計算機能を追加した簡易ルータの動作確認を行うためのネットワーク構成
3台のPC(仮想)間でネットワーク通信を行う

本連載はネットワーク上を流れるパケットを直接扱うようなツールを自作しつつ，ネットワークのしくみを勉強していきます．テーマは「自作」，「現物ベース」，「動く感動」の三つです．ネットワークにはEthernetとIPを想定しています．

前回までは簡易ルータに経路関連の機能を追加してきました．これによりパケット転送のための基本機能が備わりました．しかし実は経路の設定を誤ると，パケットがループしてしまうという問題があります．

● 今回行うこと

今回は，簡易ルータにループしたパケットを一定回数で破棄できるTTL(Time To Live)の減算機能と，チェックサムの再計算機能を追加します．TTLの減算機能とチェックサムの再計算機能を追加した簡易ルータの動作確認は，**図1**で示したネットワーク構成で行います．

本連載のプログラムのソースコードは以下の筆者のホームページからダウンロードできます．ライセンスはKL-01というもので，組み込み機器などでも自由に利用できます．

```
http://kozos.jp/books/interface/ether_net2/
```

◆今回作るもの◆

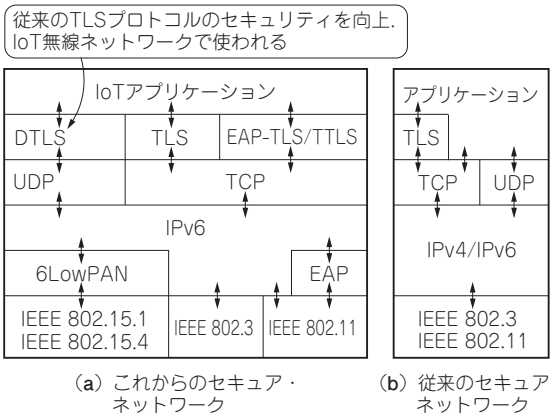
パケットの無限ループをなくすために，簡易ルータに機能を追加
- **一定回数で破棄できるTTLの減算機能**
- **チェックサムの再計算機能**

を追加

IoT時代の必読！ 押さえておこう！技術仕様の基本 インターネット・プロトコル教科書

第8回 IoT無線ネットワークに必須のセキュリティDTLS

笠野 英松



（a）これからのセキュア・ネットワーク　（b）従来のセキュア・ネットワーク

図1　今回紹介するプロトコル…IoT無線ネットワークで使われるセキュリティDTLS

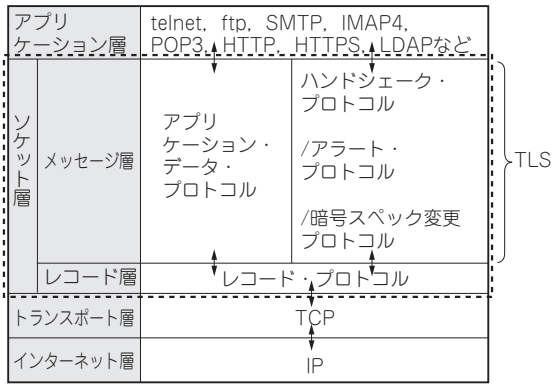


図2　DTLSのベースとなるネットワーク・セキュリティの基本プロトコルTLSの階層構造

最近では映像や音声，音楽，ゲームなど多くのアプリケーションがUDP（User Datagram Protocol）と呼ばれるプロトコルを使用することが多くなってきています．とくに，無線伝送が一般的なIoT/M2M（Internet of Things/Machine to Machine）では，TLS（Transport Layer Security）のセキュリティを向上させたDTLS（Datagram Transport Layer Security）を用います．今回はこのDTLSについて解説します．

● TCP/IPセキュリティ

図1はアプリケーションに直接関係するTCP/IPセキュリティのうちトランスポート関連の部分を示しています．従来はTLSだけでしたが，現在では，DTLSやEAP関連TLS（Extensible Authentication Protocol - Transport Layer Security）が出てきています．IoT/M2M関連では下位層にIEEE 802.11（無線LAN）＆EAPやIEEE 802.15.1/4＆6LowPAN，そしてIPv6を使用することになります．ここで，セキュリティの中心的なプロトコルがDTLSです．

ベースとなる基本セキュリティ・プロトコルTLS

DTLSプロトコルはTCP上で稼働するTLSプロトコルの「一つのバリエーション/差分」ということができます．そのため，DTLSを理解するためにはベースとなるTLSの仕様を押さえておくことが前提です．

● 階層構造

TLSは**図2**のような階層構造（2階層）を持っています．トランスポート層（TCP）の上にはTLSレコード・プロトコルがあり，上位層を暗号でカプセル化します．TLSの上位層としては，TLSハンドシェーク・プロトコルやアラート・プロトコル，暗号スペック変更プロトコル，アプリケーション・データ・プロトコルがあります．

● セキュリティ保護時のコネクション状態

TLSは特有のコネクション状態を保持します．このコネクション上でアプリケーションの各セッションを実行します．TLSコネクション状態は，暗号化アルゴリズムや圧縮アルゴリズム，認証アルゴリズムなどを表す．TLSレコード・プロトコルの動作環境で，以下のようなものがあります．

- 圧縮状態（compression state）：圧縮アルゴリズムのカレント状態．
- 暗号状態（cipher state）：暗号化アルゴリズムのカレント状態．
- MAC秘密鍵（MAC secret）：コネクションのため

第1回　ネットを使うときの隠れ常識！技術文書RFC（2015年9月号）
第2回　考えとかないとマズい！ IPv6プロトコルの全体像（2015年10月号）
第3回　IPv6プロトコルこれだけは…パケット/アドレス/ソケット（2015年11月号）

数mをリアルタイムに！クルマに使われる高信頼性バス！

制御＆監視向け！小型ネットワークCAN通信入門

第6回　最新テクノロジ：メニーECU時代は高速化必須！実験室で8MbpsのCAN FD

田代 有宏

今回は，CANプロトコルの後継として策定された新しいCAN FD（CAN with Flexible Data Rate）プロトコルについて紹介します．

CANの課題…通信速度1Mbpsじゃもう間に合わない

CANの普及に伴い自動車の電子制御化が進んでいます．それにより，ノード間でやり取りする情報も膨大になることで，CANバスの帯域の不足が課題となっています．

CANの通信速度は最大1Mbpsとなっていますが，安定的で安全な通信を考えると，実際の運用では500kbps以下で使うケースが多いです．

また，バスに掛かる負荷が高い状態だと遅延が大きくリアルタイムな制御が困難となるため，実質，30%程度までが最適と言われています．

現状は，帯域が不足した際には，複数のバスに分割することで負荷を分散して回避してますが，それも限界に近づいてきています．それに加え，さらなる電子制御化や利便性の向上には，より多くのデータを高速に扱うことができる通信プロトコルが必要となってきています．

これまでのCAN高速化の取り組み

● その1：FlexRay…最高10Mbpsまで対応できるが専用ネットワークを作らないといけない

10年前に今のCANをしのぐ全く新しい方式を採用した次世代の車載ネットワークとして，通信速度を最大10Mbpsに対応したFlexRayという通信プロトコルが注目を集めました．

残念ながらCANプロトコルとの差異が大きく，新たにネットワークを構築する必要があります．移行のために必要な条件も多いということから，日本国内では普及までには至らなかったというのが，これまでの状況でした．

● その2：TT-CAN…スケジューリングが難しくて通信速度が向上できなかった

CANバス上で，おのおののノードからメッセージが同時に送信されないように送信できるタイミングをスケジューリングすることで，通信帯域を拡大しようとするTT-CAN（Time Triggerd CAN）規格も登場しました．

ISO 11898-4として定義されているこの規格は，スケジューリングのために各ノード間の通信をかっちりと決める必要があることや，従来CANと同じ形式のフレーム構成であるため通信速度は同じということから速度が向上できず，普及拡大には至りませんでした．

最高8Mbps以上!?高速CAN FD通信プロトコル

● 規格

CAN FDは，CANの課題となっている帯域不足を解決するためにロバートボッシュ社によって提唱された新たな通信プロトコルです．

ISO 11898-1：2015として新規に制定されています．新しいISO規格では従来のCANのフォーマットを“Classical CAN frame format”，新しいCAN FDを“CAN Flexible Data Rate Frame format”として分けて定義しています．

● CAN FDとCANとの違い

CAN FD規格のベースは従来のCAN仕様なのですが，似て非なるものとなっています．

CANとCAN FDの機能面での主な違いは以下の通りです．

- 転送可能なデータ長を最大8バイト（CAN）から最大64バイト（CAN FD）に拡張
- CANの通信速度は最大1Mbpsだが，CAN FDでは2種類の通信速度が設定でき，データ領域の通信速度の高速化が可能（例えば2Mbps，4Mbps，…10Mbpsなど）．

これから10年使える技術！
標準AUTOSAR開発プラットホーム入門

安全に使い回す！車載ソフトウェアの世界

第6回　実際のプログラムの作成…ECUコンフィグレーション/インテグレーション

高田 光隆，鴫原 一人

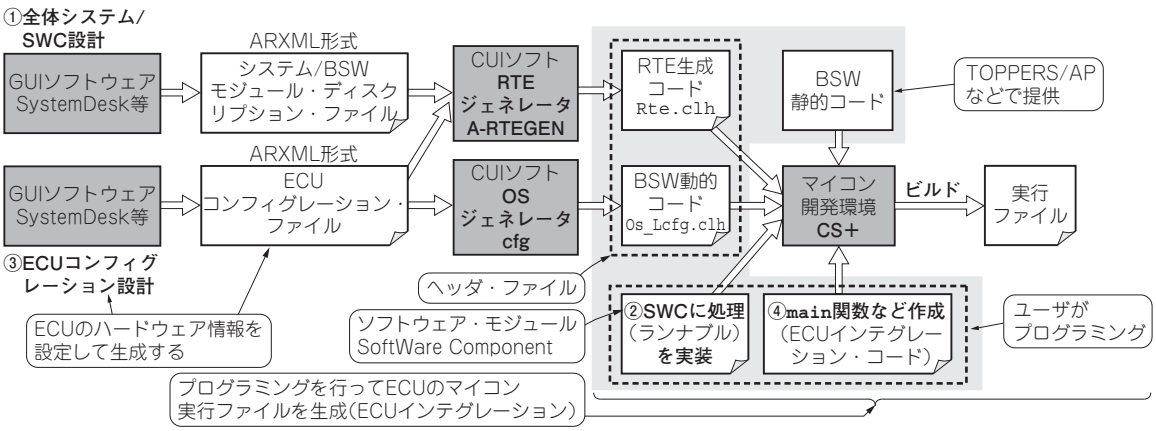


図1　標準AUTOSARによる車載ソフトウェア開発フロー…超シンプルな1ECUの場合
第5回図1再掲

● 今回やること

標準プラットホームAUTOSARを使った車載システム開発は，大きく次の3ステップで行います（**図1**，詳細は連載第2回参照）．

ステップ1：全体システム設計
ステップ2：ECUコンフィグレーション
ステップ3：ECUインテグレーション

前回は，「ステップ1：全体のシステム設計」を行いましたので，今回は，ステップ2/ステップ3という手順で，実際のECU用プログラムを作成します．

本連載のターゲット・ハードウェアは，クルマはラジコン・カーで，ECUは車載マイコンRH850ボードです．詳細については，ハードウェア構成は第3回を，ソフトウェア構成は第4回を参考にできます．

ステップ2：ECUコンフィグレーション…1ECU構成の場合

● やること…XML形式の設定ファイルを作る

1ECU構成の場合は，各ECUに機能割り当てなどを行うEcu Extractの作業はありません．

次の情報から，ECUのハードウェア情報などを記したECUコンフィグレーション・ファイルを作成します．

- システム/SoftWare Component設計の情報
- SoftWare Component開発で行ったランナブルやイベント，排他エリアに関する設定

一般的な組み込みアプリケーション開発の場合，リアルタイムOS（RTOS）を使って，アプリケーションをタスクに分解し，そのタスクをどのように起動させるかとか，タスクの優先度をどのように設定するかを決めます．

AUTOSAR開発でも同様のことをECUコンフィグレーションで行います．

● 設定に使うコンテナ/パラメータは定義されている

ボディ系のソフトウェア・モジュール（SoftWare Component）AddBodyControlを使ってECUコンフィグレーションの方法を見ていきます．

ECUコンフィグレーションでは，ARXML（XML）による記述で設定を行っています．設定には，AUTOSARの仕様書で定義されたコンテナ/パラメータを使います．

宇宙船の設計で考える?!

研究！モノづくりの最新コモンセンス「機能安全」

第9回　システムの安全性を高める多重化＆多様化

森本 賢一

図1　多重化で安全性を高める

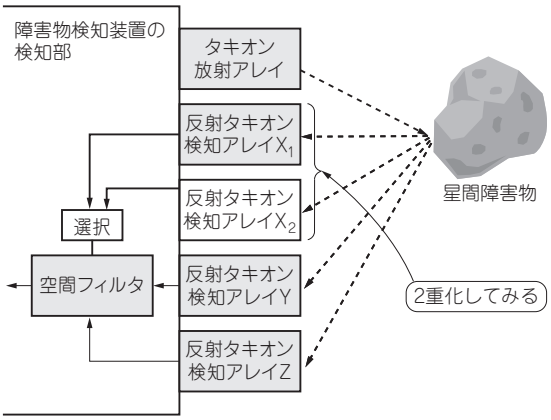


図2　宇宙船の障害物検知装置の検知アレイを2重化する
空間フィルタから先の処理は第7回の図2と同じ

● 今回解説すること

システムの危険側不動作確率*PFD*とは，電子部品の故障などにより，「万一の際に，きちんと動作しないかもしれない確率」を意味し，システムの完全性を評価する大切な数値です．設備のリスク・レベルに応じて必要な完全性のレベルが決まります．それをSILといいます．そのSILに応じてシステムが最低限達成しておくべき*PFD*が決まります[注1]．

必要な完全性を達成するためには，*PFD*の計算式にある，λ_{DU}を小さくすることが必要です（第8回で詳しく解説）．λとは部品故障率のことで，λ_S（安全側），λ_D（危険側），の分類に加え，λ_{DU}（危険側かつ検知できない），λ_{DD}（危険側かつ検知できる）に分類されます．このうちλ_{DU}が*PFD*の計算で重要な役割を果たします．λ_{DU}を小さくするために，安全メカニズム（自己診断などのメカニズム）をシステムの要件に追加してゆく必要があります．

危険側不動作確率*PFD*を低くする方法は大きく二つあります．

(1) 適切に検出して対策する安全メカニズムがあれば，その対象となる故障は，λ_{DU}ではなくλ_{DD}にカウントできることとなり，*PFD*の低減を図れる
(2) 多重化することで*PFD*の低減を図る

前回は(1)を紹介しましたので，今回は(2)を紹介します．

● *PFD*を小さくするための方策：多重化？

今回は，安全メカニズムの追加以外で，システムの危険側不動作確率*PFD*を小さくする手法として，多重化について説明します（**図1**）．細かい解説をしなくても，2重化，3重化すれば，システムの信頼性が上がることは感覚的に理解できると思います．多重化により片方が故障している間に故障側を修理するなど，障害への対処ができることも大きなポイントです．

しかし単に多重化するだけでは，極端には信頼性（完全性）が向上しません．

今回のターゲット…
障害物検知装置の2重化

● 基本思想…2台使えば失敗する確率を2乗で小さくできる？

前回（第8回，2016年5月号）取り上げた障害物検知装置について考えます．前回は，反射タキオン検知ア

注1：例えばSIL3では，*PFD*は10^{-4}のレベルです．これは，「万一の事態」が10,000回発生した場合，その対処を1回程度失敗する可能性があることを示します．

第1回　業界用語「機能安全」と「本質安全」（2015年8月号）
第2回　「リスク」「安全」…用語の定義（2015年9月号）
第3回　評価を繰り返して「安全」を目指す…リスク・マネジメント（2015年10月号）