江端さんのDIY奮闘記 EtherCATでホームセキュリティシステムを作る（7）：

# 実践！ご主人様とメイドがステッピングモーターを動かす

http://eetimes.jp/ee/articles/1601/28/news066.html

さて、今回は、前回に続きEtherCATの“2大技巧”の1つである「SyncManager」を解説したいと思います。その後、ステッピングモーターを動かすためのプログラムを作成してみましょう。

2016年01月28日 09時30分 更新

[江端智一, EE Times Japan]

FA（ファクトリオートメーション）を支える「EtherCAT」。この超高度なネットワークを、無謀にも個人の“ホームセキュリティシステム”向けに応用するプロジェクトに挑みます……!! ⇒「江端さんのDIY奮闘記 EtherCATでホームセキュリティシステムを作る」連載一覧

前回、IoT（モノのインターネット）が社会にパラダイムシフトを与えるためには、IoTに特有の特別な「エロ」を実現するアプリケーションが必要である、という主張を致しました。

ちょっと調べれば、GPSを使って出会いを演出するスマホアプリとか、ゴロゴロと出てきます。しかし、その手のものは、既に18年も前に専用端末として販売されていたのです（本当）。
当時、100万台以上も売れたらしいのです。しかし、それで「運命の人に出会えた」という話は、ついぞ聞いたことがありません*）。

*）そんな端末を持って、街の中をうろついている奴がいたら、私は、普通に「怖い」と思う。

ちょっと考えてみたのですが、「彼女／彼氏といるときに、ムード音楽を流すラジカセ」やら、「寝室の調光を少しずつ落していくライト」とか、「おたがいのエッチな気分を計測する装置」とか、

―― 陳腐だ。その程度のモノで、パラダイムシフトを起こせるものか

と自分でも思います。

「IoT」と「エロ」の組み合わせは、ちょっと、しきい値が高すぎたかもしれません。

しかし、「IoT」と「愛」くらいであれば、幾つか考えているものがあります。

「テンダー体重計（×スマート体重計）」です*）。

*）“Love me tender”の「テンダー」です。

私は、最近、EE Times Japanのサイトで、「ダイエット」について連載させて頂いています（世界を「数字」で回してみよう（ダイエット編））。

ダイエットを挫折させる大きな原因の1つに、「停滞期」という、「体重が低下していかないように見える期間」の存在があります。

「停滞期」がダイエットの意志を砕いているのであれば、体重計が、体重計の利用者であるユーザーに「停滞期」であることを知らせなければいいと思いませんか？

「テンダー体重計」は、ユーザーが、

1. 1日何回も体重を測ろうとしている
2. 1回の測定で何度も測り直す
3. 体重の表示が一番軽くなった瞬間に、体重計から降りようとする

などの行為を認めた場合、そのユーザーが「ダイエット中」であると判断します*）。

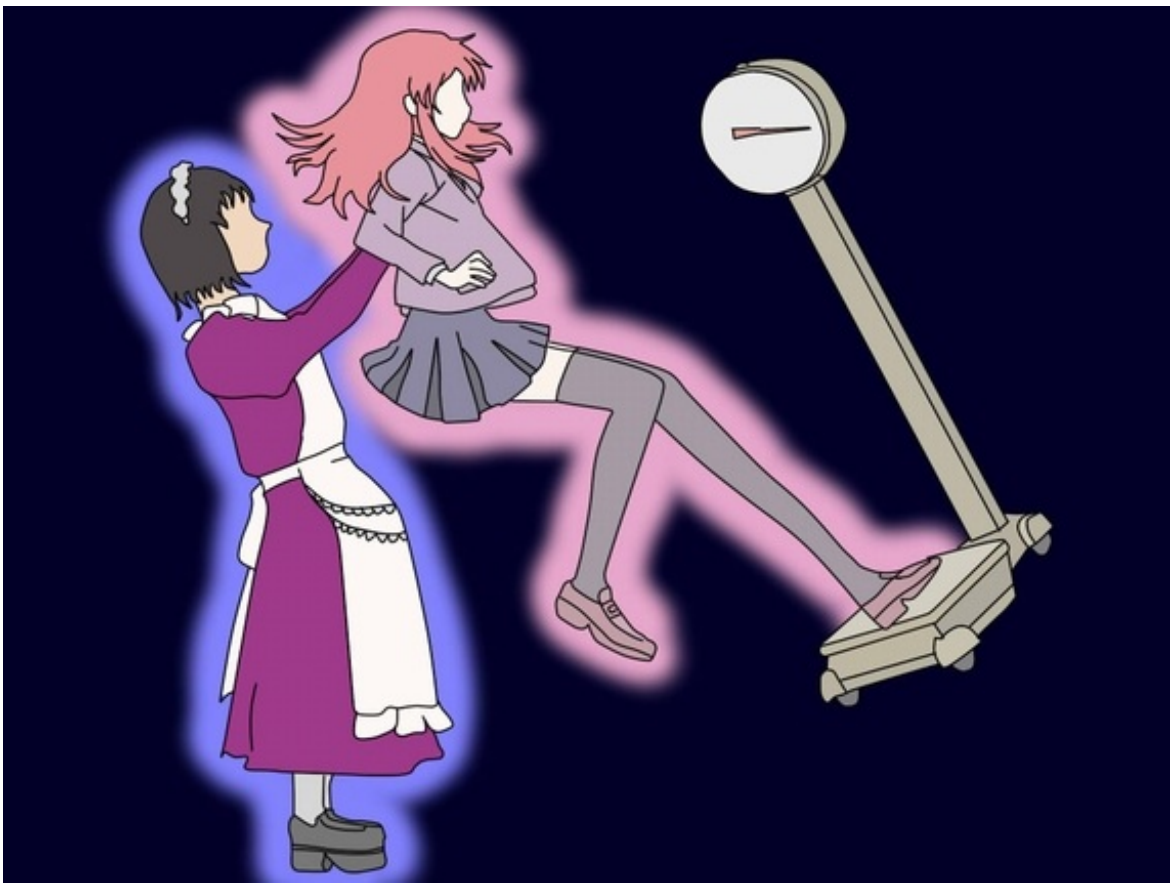
*）ちょっと高機能の体重計であればマイコンは内蔵されているでしょうから、この程度のプログラムなら簡単に実装できるハズです。

このような判断をした「テンダー体重計」は、

- 大きく体重が減った日であっても、あえて、控え目な数値を表示し、
- 停滞期に入った時にも、前日より少しだけ減らした数値を（間違っても、値数を増やすことがないように）表示する

ようにして、ユーザーにダイエットを諦めさせないように、動作します。

これこそが「IoT」と「愛」のコラボレーションと言えましょう。



この技術の適用分野は広いです。例えば、

- ヘルスケア分野では、身長計、血圧計、レントゲンなどの測定装置
- 運動分野では、ストップウォッチ、タイマーなどの計測装置
- 教育分野では、TOEICやTOEICや、そしてTOEICなどの採点システム*)

が考えられるでしょう。

*）関連記事：TOEICを斬る（前編） ～悪魔のような試験は、誰が生み出したのか～

「健康状態をモニタリングする」「運動能力を計測する」「英語能力を評価する」―― だけでは、足りない。

「健康になろう」「記録を伸ばそう」「英語の勉強をしよう」という気持ちにさせ、励ましてくれるモノ（Things）が、これからのIoTに求められると考えています。

「機械のような奴」という言葉が、「人の気持ちを理解する優しい奴」という意味に使われるようになる日 ―― それこそが、IoTが社会にパラダイムシフトを与える日になる、と私は思うのです。

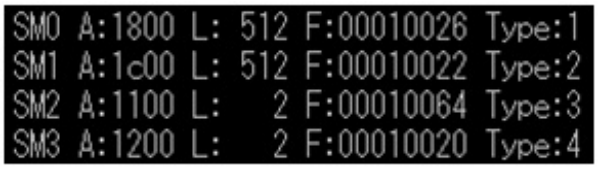
EtherCATの2大技巧「SyncManager」

こんにちは、江端智一です。

前回は、EtherCATを使ってステッピングモーターを回している動画、その回路図、slaveinfo.exeを使ったマスタのメモリマッピングの実体についてお話しました。

そして、EtherCATの2大技巧（と江端が勝手に呼んでいる）の1つ、FMMU（Fieldbus Memory Management Unit）の解説まで行いました。

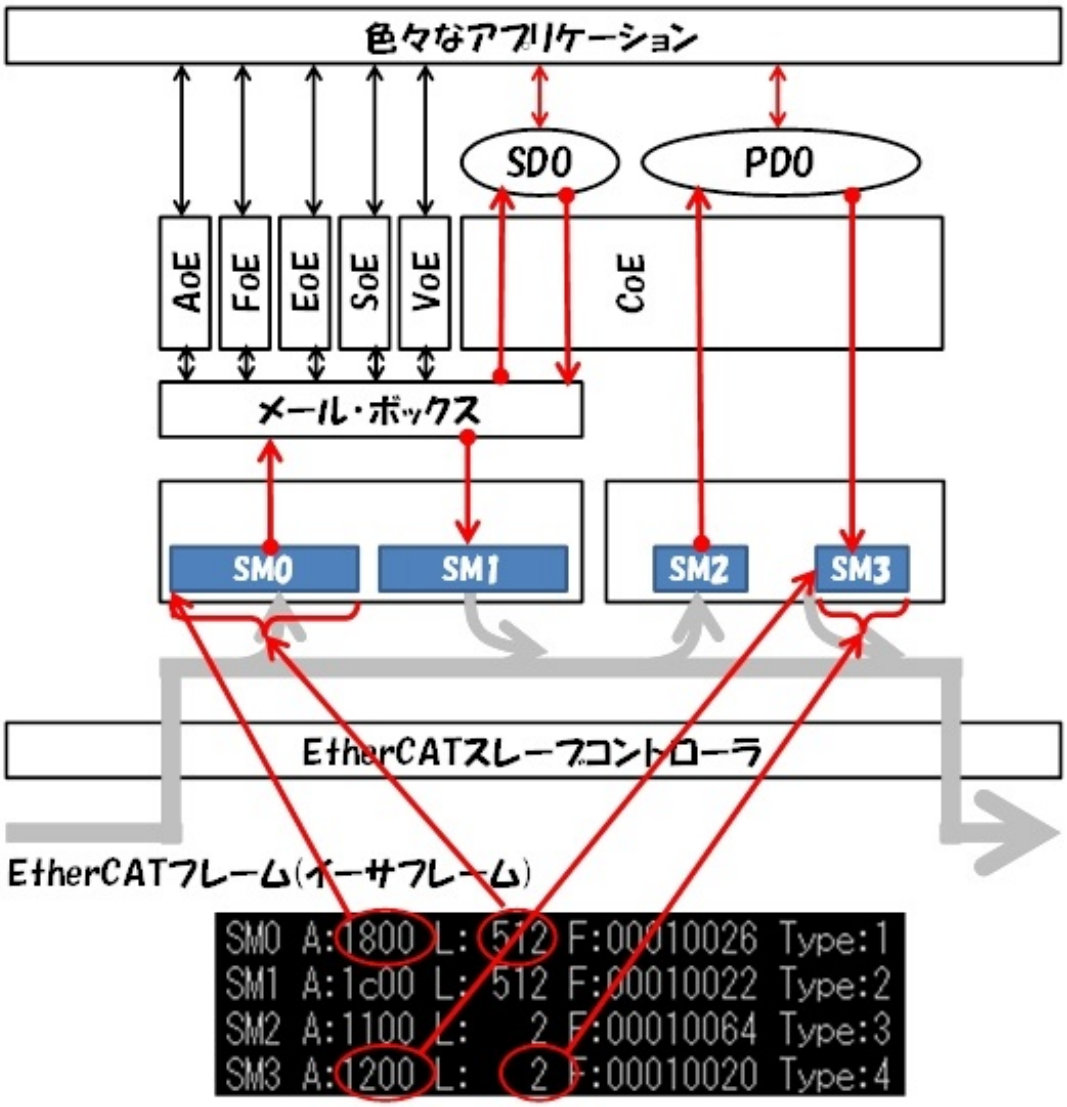
今回は、もう1つのSyncManager（以下、SMと言います）について説明します。




SMとは、（乱暴に言えば）メッセージを記載した用紙を一時的に置いておく机のようなものです。しかも、その机には（原則として）1枚の用紙しか置けません（以下、簡単に「バッファ」と言ってしまいます）。

ただ、SM0とSM1、そしてSM2とSM3では、置かれるメッセージの内容と性質が、まるっきり違います（SM3とSM4がミカン箱の机*）のサイズなのに対して、SM0、SM1の机は、その100倍以上も大きいのです）。

*）たとえが古過ぎるかな




[SM0]は、メイド（スレーブ）が、仕事が比較的暇な時にチェックする、ご主人様（マスタ）からの手紙の受信用のバッファで、[SM1]は、逆にメイドからご主人様へ送付する手紙の送信用のバッファになります。非同期の通信用に使います。[SM0][SM1]を取り扱う時のメイドの稼働イメージは「のんびり／ほのぼのモード」です。

第3回「「老人ホーム 4.0」がやって来る」イラスト

一方、[SM2]に到着するメッセージは、製造ラインを、ミリメートル、マイクロ秒単位の超絶高精度で制御するための、ご主人（マスタ）からの命令書用のバッファです。恐しく正確なタイミングで、最大毎秒8000回も届けられます。

メイド（スレーブ）は、そのメッセージを受けとるや否や、全力でその命令を実施し、その結果を、[SM3]のバッファからご主人様（マスタ）に返信します。

[SM2][SM3]は、周期プロセスデータ通信に使うもので、メイドの稼働イメージは「締め切り前/火事場モード」です。

第1回「EtherCATって結局なに？ ～「ご主人様」と「メイド」で説明しよう」イラスト

（なお、1つのスレーブの中に「のんびり/ほのぼのモード」と「締切前/火事場モード」の2人のメイドがいるわけではなく、同一人物が2つの仕事を行っています）

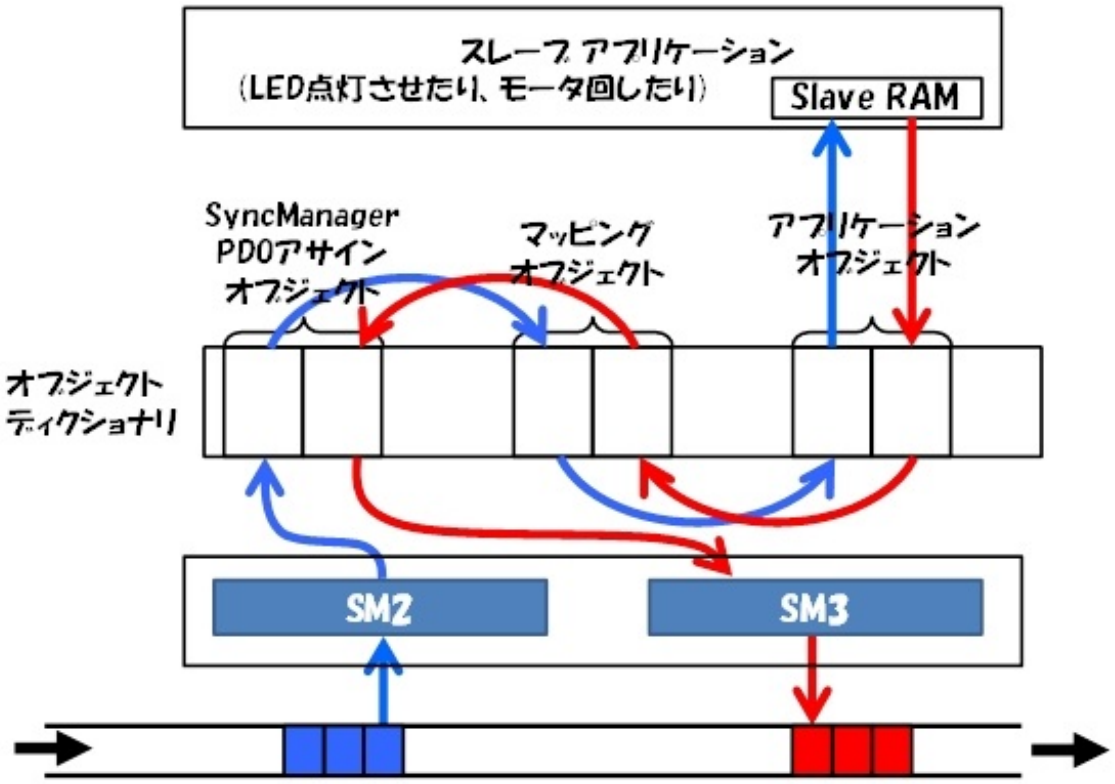
上図のケースでは、SyncManagerのサイズが、メールボックス通信用に512バイト、プロセス通信用に2バイトとなっているのが分かります（机のサイズとしては、256倍の差があります）。

SyncManagerからスレーブのアプリケーションに届くまで

ところで、それらのメッセージがSyncManagerからスレーブのアプリケーションに届くまでのプロセス（その逆方向も）は、壮絶に面倒くさいです。

しかし、私の後に、この壮絶な面倒くさい説明をしてくれる方が出てくるとも思えませんので、私が私のためだけに、以下の解説を残しておきたいと思います。

今回は、PDO用の[SM2][SM3]を使って、説明を試みます。




まず、3つのオブジェクト(SM PDOオブジェクト、マッピングオブジェクト、アプリケーションオブジェクト)が出てきます。オブジェクトと呼ばれていますが、実体は「単純な2列のテーブル」です。

正直、この絵を描いているうちに、次第に腹が立ってきました。

「スレーブの中で、メッセージをグルグルと回して、遊んどるんか!」と。

ところが、これも冷静に考えれば、結構、うまく考えられているのです。

例えば、1バイト=8bitの信号が、

- 赤色のパトランプの点滅用に1ビット
- 青色のパトランプの点滅用に1ビット
- 4段階の調光スイッチ用に2ビット
- ステッピングモーターを動かすために4ビット

が割り当てられているとします。

当然、これはデバイス単位でも管理しなければなりませんが、SyncManagerで、メッセージの中身を。バラバラにぶった切ることまでやると処理が遅くなってしまいます(そもそもSyncManagerは、メッセージ用紙を一時的に置いておく、単なる机ですし)。

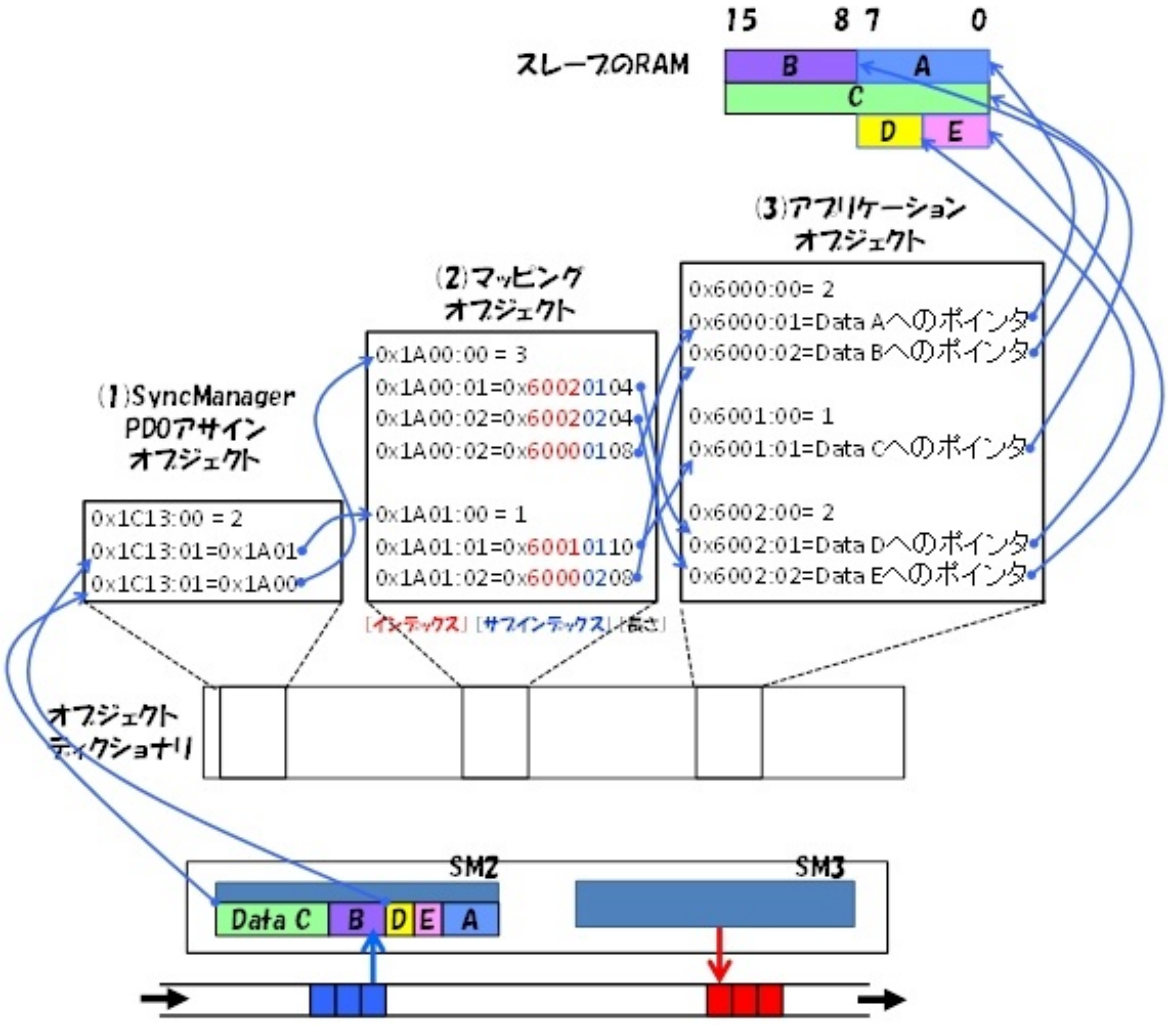
また、例えば、1つの出力信号を2つのアプリケーションで利用する(例えばパトランプとステッピングモーターの両方で同時に使う、など)とか、(イメージしにくいですが)1つの入力信号を2つ以上の異なる入力信号としてマスタに伝えたいような場合だってあるかもしれません。

とすれば、どこかで、ビットの塊を切り分けて整理しておくテーブルや、アプリケーションとアクセスさせるテーブルが必要になってくるはずです。

それぞれのオブジェクト（テーブル）の役割を、以下のように把握すれば分かりやすいと思います。

| 名称 | 役割 |
|---|---|
| (1)SyncManager PDOアサインオブジェクト | EtherNetのフレームから、直接ビット列を掴み出して置いておくテーブル |
| (2)マッピングオブジェクト | 掴み出したビット列を、アプリケーションが使えるビット単位にぶった切る為のテーブル |
| (3)アプリケーションオブジェクト | ぶった切ったビットを、アプリケーションに差し出すテーブル |

具体的には、こんな感じです。




よく分からないでしょう？ 大丈夫です。私も分からないです[*)]。

*）オブジェクトディクショナリの構成や、スレーブプログラムがどうやってオブジェクト（テーブル）の内容を参照するのかが、いまひとつピンとこない。

　いずれにしても、データの参照先が、テーブルでたらい回しにされて、EtherCATのフレームからスレーブのRAMにコピーされる、または、その逆が行われる、という感じに理解しておけば、十分と思っています。

　なぜなら、この辺の話は、（1）EtherCATスレーブの開発者の方の設計方法に依存する話であること、そして、（2）EtherCATマスタとしては、EtherCATスレーブの内部構造がどうであるかは「どーだっていい」からです。

　いろいろと記載致しましたが、SOEMのアプリケーションツールである”Slaveinfo.exe”を使うことによって

- マスタのどのアドレスに、スレーブ情報が割り当てられるか
- スレーブを動かす為には、マスタのアドレスのどのビットを書き換えれば良いのか

ということが分かります。

　それさえ分かれば、今回のステッピングモーターを動かすには十分なのです。

　実際のところ、FMMUやらSyncManagerの情報など、SOEMのアプリケーションプログラムやシステムのオペレータは知らなくても良いことだからです。それらの情報は、メイド（スレーブ）とご主人様（マスタ）の間で共有できていれば十分です*）。

*）ただ、マスタやスレーブの製品開発をするエンジニアは、これらを知らないと何も作れませんですけど。

　さて、今回の場合は、

「PCのメモリアドレス「01257D80」から最初の2バイト（のうち最初の8ビット）が、出力用、次の2バイト（のうち最初の8ビット）が入力用」

と分かれば、それで十分です。

EtherCATでステッピングモーターを回す

　では、ようやくここから、EtherCATでステッピングモーターを回すプログラムの作成に入ります。このプログラムはSOEMで動かすマスタ用のアプリケーションとなります。

　SOEMのアプリケーションを、スクラッチから作るのは大変なので、Microsoft Visual C++の中のsimple_test”プロジェクトの中にある、”simple_test.c”を改造して作ってしまいましょう。

　まず、前提として、「SOEMのデバッグ＆トレース環境の作り方」が完了しているものとします

（参考）。

Simple_testのプロジェクトを「スタートアッププロジェクトに設定」して、実行してみてください（simple_test.cは、まだ改造しません）。

この段階で、あなたの使っているDI/DOのEtherCATのスレーブのLEDが、ピカピカと点滅していれば問題ありません。

もし、点滅していなければ、先ほどの「slaveinfo.exeの表示（前回分ご参照）」に記載されている、メモリの先頭アドレスの値（例：「IOMap:01257D80」）と、スレーブ1の出力デジタルI/O用のアドレス（例：「Outputs 01257D84」）を調べてください。点滅していないのであれば、値が異なっているハズです。この差分を求めておいてください。

上記の例ですと、差分は、01257D84 - 01257D80 = 4 となります。

次に、simple_test.cの、最初の部分にある、”void CALLBACK RTthread（…）

の中にある、”IOmap[0]++;” の”0”を上記の値（例：”4”）に入れ替えてください。

（例："IOmap[0]++;"→IOmap[4]++;”）

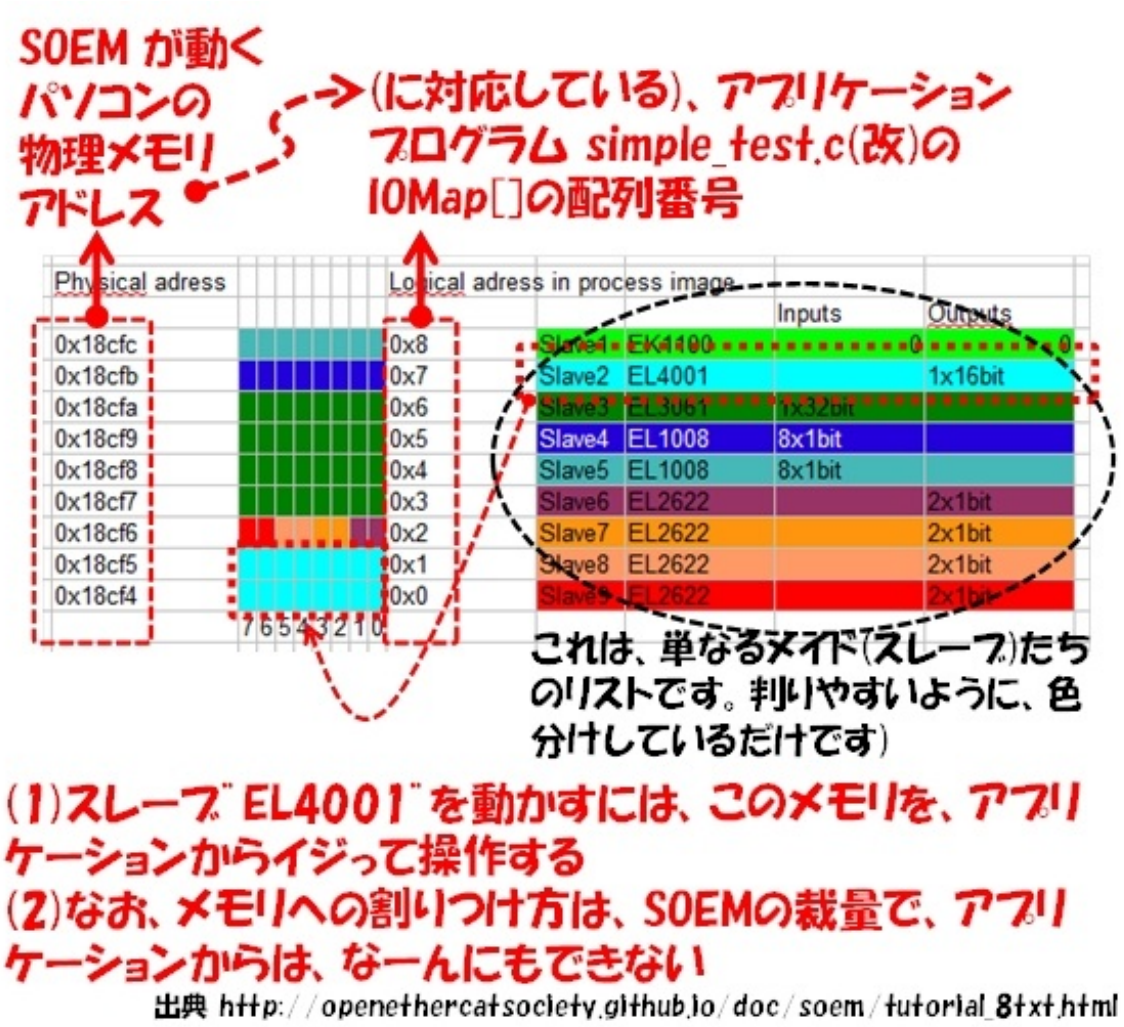
これで、再度コンパイルして実行すれば、LEDの点滅が確認できると思います。

これが先ほど申し上げた、「PCのメモリアドレス「01257D80」から最初の2バイト（のうち最初の8ビット）が、出力用、次の2バイト（のうち最初の8ビット）が入力用と分かれば、それで足るのです」の意味です。

“simple_test.c“が、EtherCATスレーブを制御するメモリの番地（例：”IOMap[4]”）さえ分かってしまえば、あとは、その” "IOMap[]"の中身を書き変えてやるだけで、EtherCATスレーブは動き出します。

何はともあれ、とにかく、何が何でも、この段階でスレーブのLEDのピカピカ点滅を確認してください。これが点滅しなかったら、ここから先に進めることができません。

さて、最後に、このマスタのアプリケーション（例：simple_test.c）で使われる”IOMap[]”の内容を（本当に、何度もしつこいですが）もう一度だけ、おさらいしてみましょう。




出典は[こちら]

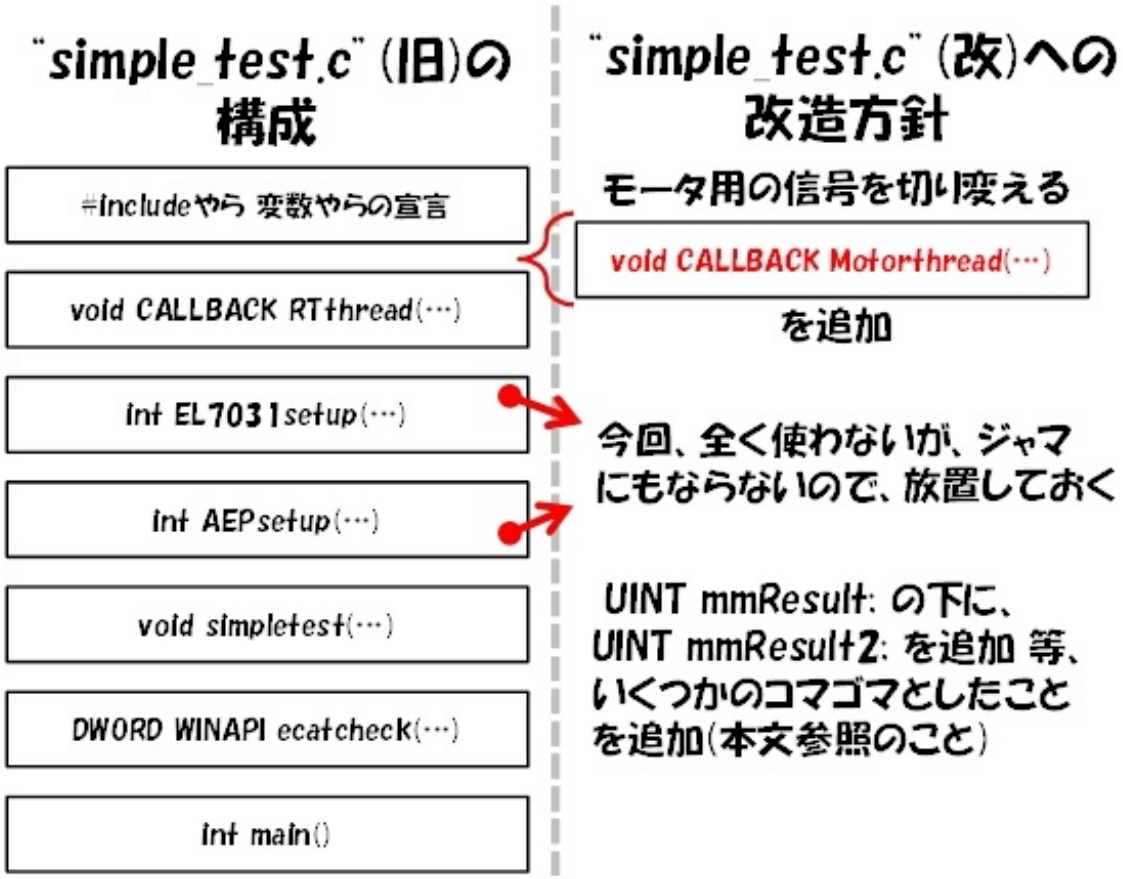
マスタとスレーブが協力して、うまいことマスタ(SOEM)のメモリを無駄に使わないように、パッキングしているのが分かります。

"simple_test.c"の改造

では、これより、"simple_test.c"の改造を開始します(改造前のsimple_test.cは、念のため、別のファイル名で保存しておいてください)。

といっても、要するに、EtherCATスレーブのIO出力を、"simple_test.c"から、コントロールできるようにすれば良いだけのことです(なお、ステッピングモーターの動作原理については、ここでは説明しませんので、がんばって勉強*)してくださいね)。

*)編集注:そんな皆さまに送りたい、EDN Japan関連記事。まずはここから「「ステッピングモーター」で学ぶエンコーダの活用法」




今回は「ステッピングモーターを動かせればいい」という観点のみに注力して、考え得る最小の改造とします。

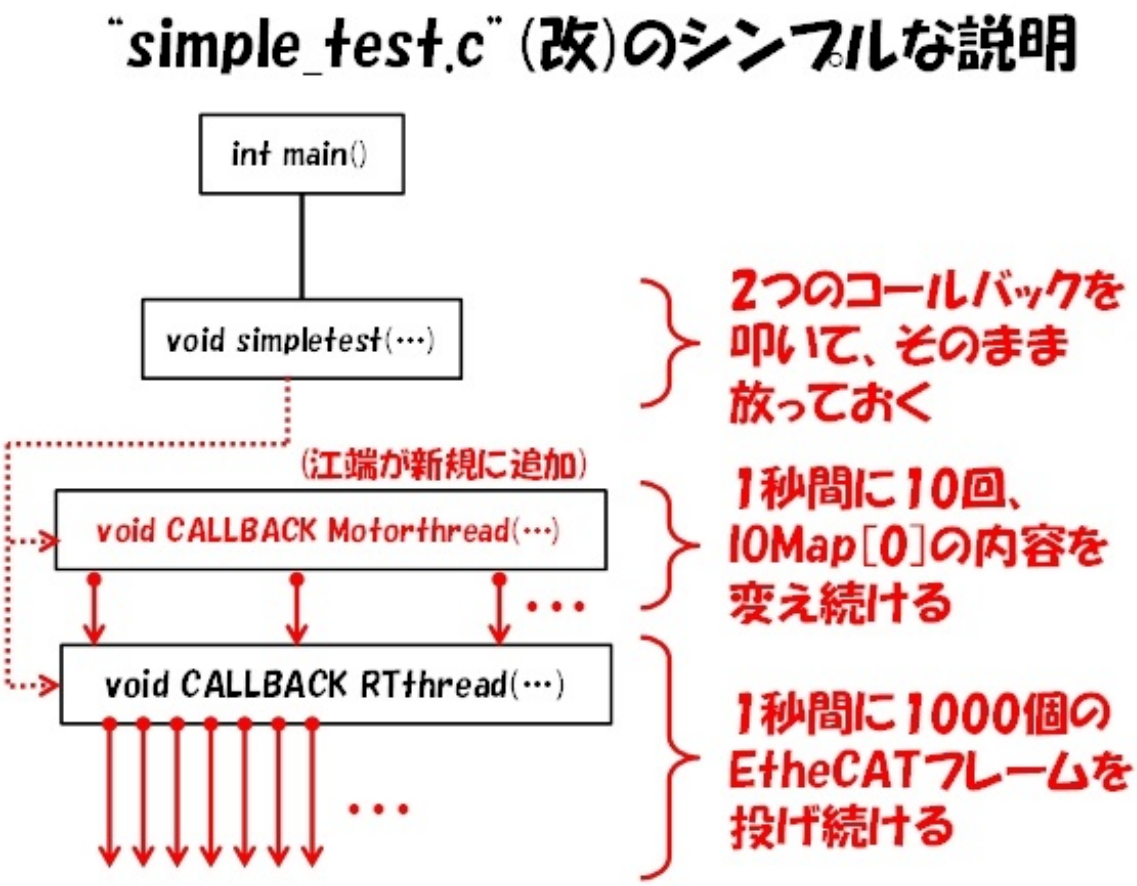
```
■37行目の下に、以下を追加
char stepping_Signal = 1; //00000001
/* Stepping motor value */
void CALLBACK Motorthread (UINT uTimerID, UINT uMsg, DWORD_PTR dwUser, DWORD_PTR dw1,  DWORD_PTR
dw2)
{
 if  (stepping_Signal == 1)
  stepping_Signal = 2;  // 00000010
 else if  ( stepping_Signal == 2)
  stepping_Signal = 4;  // 00000100
 else if  ( stepping_Signal == 4)
  stepping_Signal = 8;  // 00001000
 else if  ( stepping_Signal == 8)
  stepping_Signal = 1;  // 00000001
}
■41行目の” IOmap[0]++;”  を以下の様に変更
 IOmap[0] = stepping_Signal;   // ここがLEDを点滅させるパラメータ
 // 必要に応じてslaveinfoの情報を使って、”[0]” を, ”[2]”とか”[4]” とかに変更する
■128行目の”  UINT mmResult;” の下に、以下を追加
    UINT mmResult2;
■194行目の”  mmResult = timeSetEvent (1, 0, RTthread, 0, TIME_PERIODIC) ;” の下に、以下を追加
   mmResult2 = timeSetEvent (100, 0, Motorthread, 0, TIME_PERIODIC) ; // この”100” の値を小さくす
ると、モーターの速度が上がる
■252行目の”         timeKillEvent (mmResult) ;”の下に、以下を追加
        timeKillEvent (mmResult2) ;
```

これだけの変更で、とりあえず、モーターを動かす信号を作り出せます。

この”simple_test.c”では、一度キックすると、一定の時間間隔でずっと動き続ける、コールバ

ック関数が重要な役割をしています。

常にプロセスデータをEthernetに流し続ける処理と、IOMap[]の内容を替え続ける処理を、コールバック関数に丸投げしておくという方法を採用しています。




もっとも、SOEMやこのプログラムは、非リアルタイムOSであるWindows7の上で動いているので、厳密な意味でのリアルタイム制御はできません。

EtherCATを、産業用ロボットの製造ラインや、ドローン撃退システムに使うには、ベンダー（ベッコフ社さんなど）が提供するマスタを購入した方が良いと思います。

しかし、江端家のホームセキュリティシステムに使うには、Windows7の上で動くSOEMでも、十分すぎるスペックなのです。

ソースコードを変更して、LEDの点滅パターンを自由に変えられるようになったら、前回の最初にご紹介した物品や回路図を使って、EtherCATスレーブとの接続を行ってください。

このプログラムによって動くはずの、ステッピングモーターの動画を再掲しておきます。

（回路図はこちら）

では、あなたの家のパトランプやステッピングモーターも、上手く動くことをお祈りしております

。

大切なものは“目に見えるようにする”

　では、前回と今回の内容を併せてまとめます。

（1）SOEM（Simple Open EtherCAT Manager）と、DO/DO用のEtherCATスレーブを用いて、パトランプやステッピングモーターを動かしてみた
（2）その動かし方の手順の中で、実際に動いているマスタやスレーブを使ってIOMap、FMMU、SyncManager（SM）の説明を行った
（3）“simple_test.c”の改造手順の中で、EtherCATのマスタ用のプログラムの作り方を解説した

　以上です。

　さて、前回と今回の内容を振り返ってみると、EtherCATの2大技巧、FMMUとSyncManagerという、技術的に非常に面倒くさい内容の話と、SOEMのアプリケーションプログラミングの話だけになってしまいました。

　興味のない人には、圧倒的につまらない内容だったかもしれませんが、しかし、EtherCATを真剣に始めようと考えている人にとっては、「世界一役に立つ読み物の一つになった」という自負はあります。

　私が調べた限りにおいて、現実に「SOEMのプログラムやEtherCATのスレーブを使って、ステッピングモーターやパトランプを動かしてみた」、という記事や資料は見つけられませんでした（もし見つけていれば、こんなに苦労はしなかったと思う）。

　私は、この連載の執筆のために、5月から今日に至るまで、SOEM1.3.0のソースコードを全部読み（全部は理解できませんでしたが）、ステッピングモーターを7個購入し（その幾つかを過電流で破壊しました）、いろいろな人（初対面のハードウェアのエンジニアの方）に、恥しげもなく「教えてください」というメールを、一方的に送りつけてきました。

　どんなシステムの学習でもそうだと思いますが「1000回の座学は、たった1回の実証実験や、プロトタイプの製作に及ばない」と思うのです。

　例えば、

- 微分方程式の問題集を解くのではなく、階差式に落して、本物の数字を使ってエクセルで計算してみる
- 政府が発表する人口予測をそのまま、うのみにするのではなく、ネットに落ちている（信ぴょう性の低い）データを拾い集めてでも、自分でプログラミングしてみる
- 仮説を打ち立てたのであれば、その仮説を声高に主張するのではなく、その仮説に対する実証試験を行ってみる。実証試験が難しいのであれば（例えば、戦争の戦術を検証する為に、実際に戦争することはできない）、自分でシミュレーターを作って、コンピュータの中でシミュレーションしてみる

ということです。

　仮に、私にエンジニアとしての信条のようなものがあるとすれば、それは「動いているものだけが真実である」です。

　サン・テグジュペリ先生の「星の王子さま」の超有名なセリフ『大切なものは目には見えない』は、私には寝言にしか聞こえません。

　『大切なものなら、目に見えるようにする』のが、エンジニアである私の仕事です。

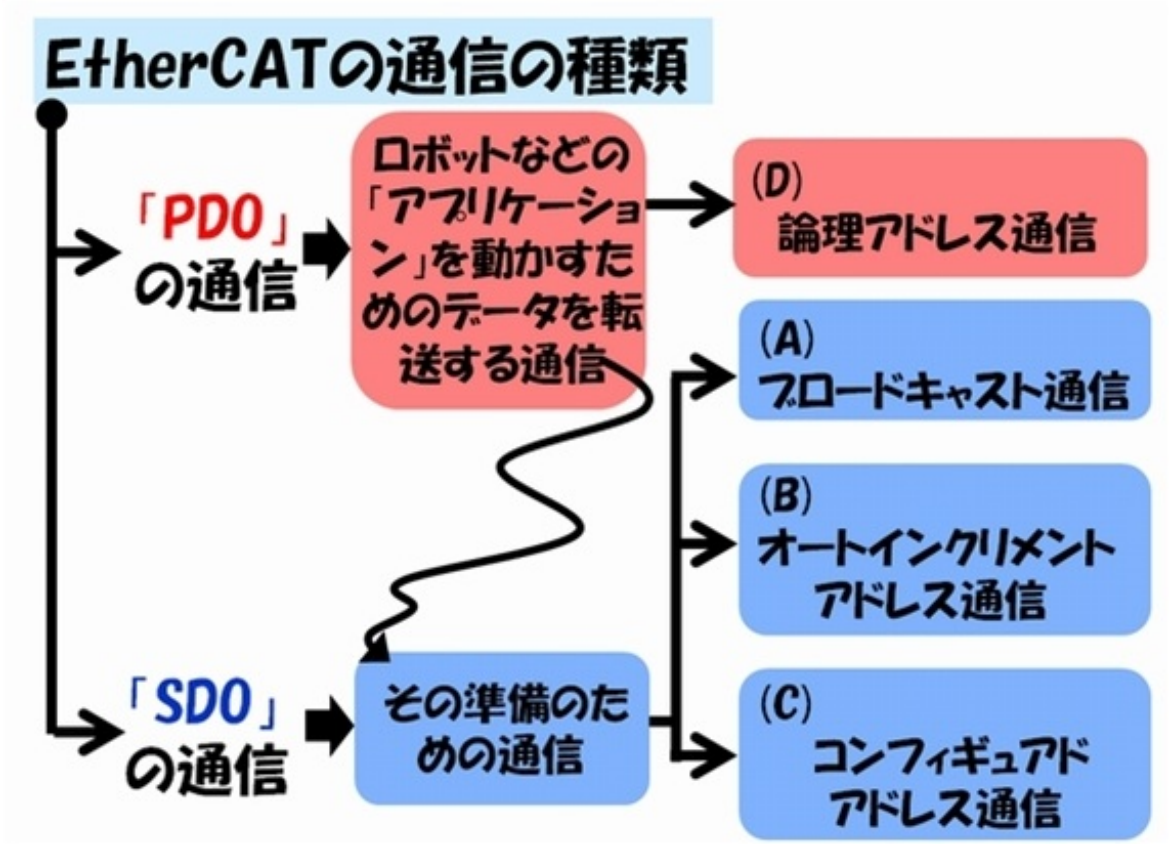
―― たとえ、それが「エロ」であろうが「愛」であろうが、です。

第5回の図版についての補足事項

　さて、2016年1月に、ベッコフオートメーションさんのオフィスにお伺いして、3時間近くにわたって、インタビューをさせていただきました（インタビューの内容は次々回に掲載予定です）。

　そこで、これまでの連載についてご指摘を頂きましたので、以下に記載します。

　第5回の「6万人のメイドが“合体”!? EtherCATの通信方式」では、以下の図を掲載しました。




　これが、読者の皆さんに誤解を与えるかもしれないとのご指摘と、以下のコメントを頂きました。

- 上記（B）オートインクリメントアドレス通信や、（C）コンフィギュアドアドレス通信は、スレーブ指定、アドレス指定とデータからなり、スレーブコントローラー内の特定のメモリをアクセスできます
- スレーブコントローラー内のメモリはレジスタエリアとSyncManagerバッファエリアからなりますが、その内、レジスタにアクセスするのはSDO通信とは呼びません
- メールボックスを経由してCoEのオブジェクトディクショナリにアクセスするものだけをSDO通信と呼びます

つまり、通信方式とアクセスできるオブジェクトは、原則として無関係ということです。（A）（B）（C）の通信は、SDOのためだけに行うものではない（他にもいっぱいある）という点にご注意頂くよう、よろしくお願い致します。

特別協力：

本連載では、スレーブの提供などでベッコフオートメーションにご協力いただいております。

---

Profile

江端智一（えばた ともいち）

日本の大手総合電機メーカーの主任研究員。1991年に入社。「サンマとサバ」を2種類のセンサーだけで判別するという電子レンジの食品自動判別アルゴリズムの発明を皮切りに、エンジン制御からネットワーク監視、無線ネットワーク、屋内GPS、鉄道システムまで幅広い分野の研究開発に携わる。

意外な視点から繰り出される特許発明には定評が高く、特許権に関して強いこだわりを持つ。特に熾烈（しれつ）を極めた海外特許庁との戦いにおいて、審査官を交代させるまで戦い抜いて特許査定を奪取した話は、今なお伝説として「本人」が語り継いでいる。共同研究のために赴任した米国での2年間の生活では、会話の1割の単語だけを拾って残りの9割を推測し、相手の言っている内容を理解しないで会話を強行するという希少な能力を獲得し、凱旋帰国。

私生活においては、辛辣（しんらつ）な切り口で語られるエッセイをWebサイト「こぼれネット」で発表し続け、カルト的なファンから圧倒的な支持を得ている。また週末には、LANを敷設するために

自宅の庭に穴を掘り、侵入検知センサーを設置し、24時間体制のホームセキュリティシステムを構築することを趣味としている。このシステムは現在も拡張を続けており、その完成形態は「本人」も知らない。

本連載の内容は、個人の意見および見解であり、所属する組織を代表したものではありません。

## 関連記事

電力という不思議なインフラ（前編）～太陽光発電だけで生きていけるか？～
太陽光発電のみで生活する――。これが現実になれば、私たちはもう原発やら電気代やらを心配することなく、夢のような生活を送ることができるでしょう。市販のソーラーパネルの「発電出力」だけを見れば、あながち不可能ではない気もしてしまいます。ですが、太陽光発電には大きな“落とし穴”があるのです。

トラブル遭遇時の初動方針は、「とにかく逃げる！」
どれだけ周到に準備をしたとしても完全には回避できない――。悲しいかな、トラブルとはそういうものです。悪天候でフライトがキャンセルされたり、怖い兄ちゃんが地下鉄に乗り込んできたり、“昼の”歓楽街でネーチャンにまとわりつかれたり……こういうものは、はっきり言って不可抗力です。実践編（海外出張準備編）の後編となる今回は、万が一トラブルに遭遇した場合の初動方針についてお話します。

ルネサス 自動運転車の頭脳となる次世代SoC発表
ルネサス エレクトロニクスは2015年12月2日、車載情報システム向けSoC「R-Carシリーズ」の第3世代品を発表した。2018年以降に市販される自動車への搭載を見込んだ製品群。第1弾製品として同日、サンプル出荷を開始した「R-Car H3」は“自動運転時代のSoC”と位置付けたハイエンド品で、最先端となるTSMCの16nm世代FinFET+プロセスを採用し、高性能な処理能力を盛り込んだ。

人間の脳が握る、デバイス低消費電力化の鍵
ウェアラブル機器に欠かせない要件の1つに、低消費電力がある。「第2回 ウェアラブルEXPO」のセミナーに登壇した日本IBMは、超低消費電力のコンピュータとして、人間の”脳”を挙げ、IBMが開発中の「超低消費電力脳型デバイス」について語った。