時間スケール，ハードとソフトの切り分け，協調モデル

組み込みソフトウェアの特徴

藤倉 俊幸

組み込みソフトウェア開発においてはさまざまな要素を考慮する必要がある．たとえば時間スケール一つとっても，人間の日常生活で使われる秒単位の操作と，ハードウェアの動くマイクロ秒単位の動作を埋めるために，くふうが必要とされる．
まずは組み込みソフトウェアの特徴について，さまざまな事情を見ていこう．（編集部）

組み込みソフトウェアを開発するには，ハードウェア開発者や関連する周辺分野の技術者と上手に連携する必要があります．そのためには，ソフトウェア開発者はハードウェアを中心とした周辺分野についてある程度は知識を持たなければなりません．そのうえで，RTOSやオブジェクト指向，形式的手法を使いこなしてソフトウェアを設計・実装する必要があります．

この特集では，組み込みソフトウェアを開発するうえで注意しなければならない雑学的周辺知識を交えながら，RTOSなしの場合から，RTOSを利用する場合のソフトウェア開発手法をまとめてみました．まず第1章では，組み込みソフトウェアの特徴についてレビューします．

1 時間スケールの多様さ

● 秒単位からナノ秒単位までをインターフェースする

普段，我々が使用している時間の単位は秒です．それに対して，組み込みソフトウェアが動作する環境の時間の単位は，だいたいミリ秒（ms）からマイクロ秒（μs）です．たとえば，「割り込み遅延時間は10μsである」とか「最長割り込みマスク時間は15μsである」などという言い方がされています．さらに，組み込みハードウェアの時間の単位は，ミリ秒からナノ秒（ns）のレベルです．

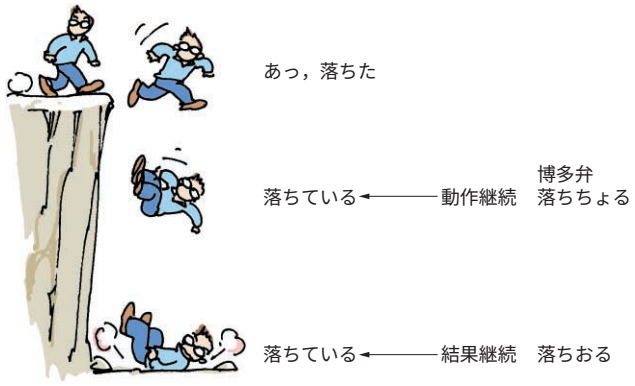


図1　動作継続と結果継続のちがい

組み込みソフトウェアは，制御対象の時間スケールと組み込みハードウェアの時間スケールとのインターフェースを取らなければなりません．すなわち，扱わなければならない時間の範囲が非常に広いのです．普通の文章では，表現しきれないほど広いのです．

● 瞬間か？　継続か？　日本語表現から見る動作の解釈

動詞には，瞬間動詞，継続動詞，状態動詞の種類があります．たとえば，「起こる」，「倒れる」，「見かける」などは瞬間動詞に分類されています．したがって，単純に考えれば瞬間動詞の動作時間が「一瞬」と認識される時間になります．標準語では，瞬間動詞に「～ている」を付けると動作継続を表す場合と，結果継続を表す場合があります（**図1**）．

どちらの意味になるかは，状況と動詞の種類に依存します．たとえば，パソコンを「起動している」と言うと，今ではほとんど動作継続と解釈されます．もし，パソコンが本当に一瞬で起動できれば，「起動している」といえば結果継続の意味になり，いつでも使えることを意味します．実際は，起動に時間がかかるので「起動している」ので少し待ってくださいの意味になることが多いのですが…．

さらに，「押す」のように結果を意識しない瞬間動詞の場合は，反復，習慣，経歴，経験などの意味になります．たとえば，エレベータのボタンを「押している」と言う場合には，何度も押していることを表したりもします．

日常生活と組み込みソフトウェアと組み込みハードウェアでは，瞬間を意識する時間スケールが異なるので，動作継続か結果継続か繰り返しかがあいまいになります．基本的には日常生活レベルで仕様書は書かれるので，イベントの順番などがあいまいになりやすく，これを明確にするためには，タイミング・チャートなどで補足しなければなりません．

KEYWORD ―ハードとソフトの切り分け，時間スケール，想定外

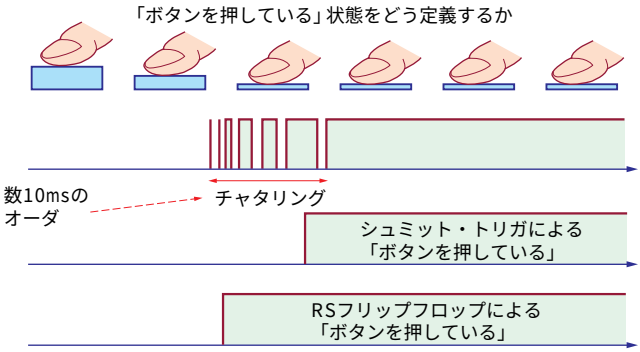


図2　ボタンを押したときのチャタリング

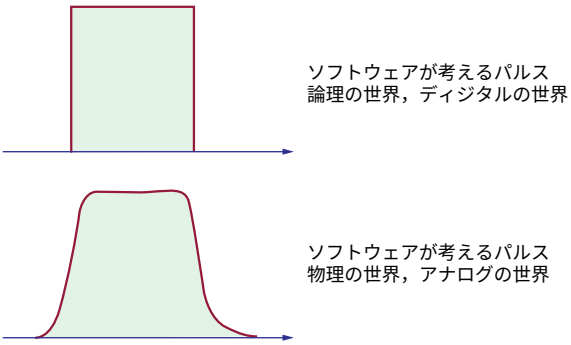


図3　ディジタルの世界のパルスとアナログ世界のパルス

● チャタリングを人間から見た場合とソフトウェアから見た場合の違い

ボタンを押した直後は，組み込みソフトウェアの時間スケールで見ると，接点でON/OFFを繰り返す現象が起こっています．ディジタル的に見ると，この過渡期的な状態はON/OFFを繰り返しているようにみえますが，アナログ的に見れば，しだいに電圧が変化しているようにみえます．この現象は，チャタリング(chattering)とか，接点はずみなどと呼ばれています(**図2**)．

この現象があると，ボタンを押された回数を数えている場合には，数えすぎることになります．ボタンでCPUに割り込みを掛けているような場合には，動作が不安定になります．日常生活レベルでは，ボタンを押すのは一瞬ですが，組み込みソフトウェアでは時間スケールが拡大されるため，本当の目的が伝わらないかもしれません．動作継続と結果継続の両方の可能性を検討しなければならないのです．

チャタリングを取り除くには，ソフトウェアで行う場合と，ハードウェアで行う場合があります．ハードウェアで行うには，フリップフロップを使う場合と，シュミット・トリガ回路を使う場合があります．フリップフロップは一定以上の幅のパルスがあればボタンを押されたと認識しますが，シュミット・トリガ回路の場合は電圧がしきい値以上になるとボタンを押されたと認識します．ハードウェアの時間スケールは，組み込みソフトウェアの時間スケールより拡大されて，一様だと思っていた時間間隔の中に，さらにイベント順などの順序構造が入ってきます．つまり，ハードウェアについて考える場合には，ソフトウェアより短いタイム・スケールでその違いを検討しなければなりません．

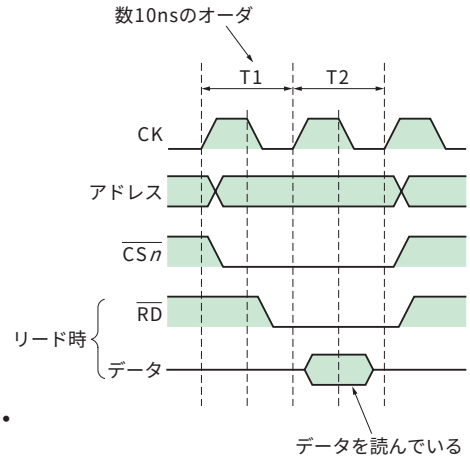
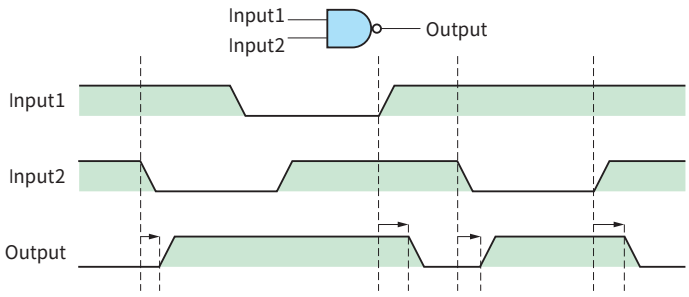


図4
バス・タイミング・チャートの例



C言語の論理式との違いに注意する必要がある．
実世界にはタイミングがある

Output = ! (Input1 && Input2)

図5　NANDゲートのタイミング・チャート

たとえば，**図2**では電圧の変化を垂直線で表現していますが，アナログの世界では**図3**のような曲線になりますし，ハードウェアどうしの動きを記述するバス・タイミング・チャートなどでは**図4**，**図5**のように傾斜した線で表現します．

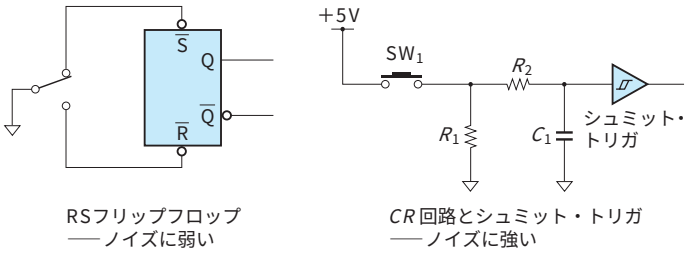
組み込みソフトウェアの仕様を記述するには，イベントの順など時間的な表現が重要です．これを正確に表すときには，自然言語はあまり頼りになりません．図や数式を使用して表現する必要があります．数式を毛嫌いする人が多いのですが，自然言語よりはずっと正確で簡潔です．また，文章で書くよりも短時間で済むというメリットもあります．文章は，あいまいなくせに，時間がかかる厄介(やっかい)なしろものです．

2 ハードウェアとソフトウェアの切り分け

● ソフトでやるべきか？　ハードでやるべきか？

ボタンのチャタリングを取り除くことは，ソフトウェアでも

Information ――経済産業省とIPA「2005年版 組込みソフトウェア産業実態調査」を発表
経済産業省と(独)情報処理推進機構は，「2005年版 組込みソフトウェア産業実態調査」を発表した．報告書は，http://www.ipa.go.jp/software/sec/download/report.php からダウンロードできる．



RSフリップフロップ
――ノイズに弱い

CR回路とシュミット・トリガ
――ノイズに強い

図6　チャタリング除去回路の例――どちらを採用すべきか？

ハードウェアでも行うことができます．どちらで行うか決めるのは，システム設計の仕事です．現在は，ハードウェア担当者が行うことがほとんどですが，本来ならば，ソフトウェア開発者もシステム設計に関与すべきです．

システム開発コストに含まれるソフトウェア開発費はかなりの比重を占めています．新しいボードを開発するとき，鉛フリー化にともなってボードを再設計したところが多かったと思いますが，このようなとき，ハードウェア技術者がソフトウェア技術者に希望を聞いても，ソフトウェア技術者からはあまり意見が出ないものです．RTOSやC++を使えるようにメモリを増やして欲しいとか，デバッグ用にシリアル・ポートを追加して欲しいとか，余ったI/Oポートを引き出しておいて欲しいとか，そこは割り込みにして欲しいとか，ソフトウェア技術者が言い出さないとだれも検討してくれません．たとえば，RTOSやC++を使えばソフトウェアの再利用性が向上し，並行開発もやりやすくなります．デバッグ/テスト工程をどのように短く，効率的にするか検討することは，開発期間短縮と信頼性向上には欠かせません．イベント・ドリブンなアプリケーションは，ポーリングだけでは実現できません．これらの意見は，システム開発コストの削減のために考慮されるべきものです．

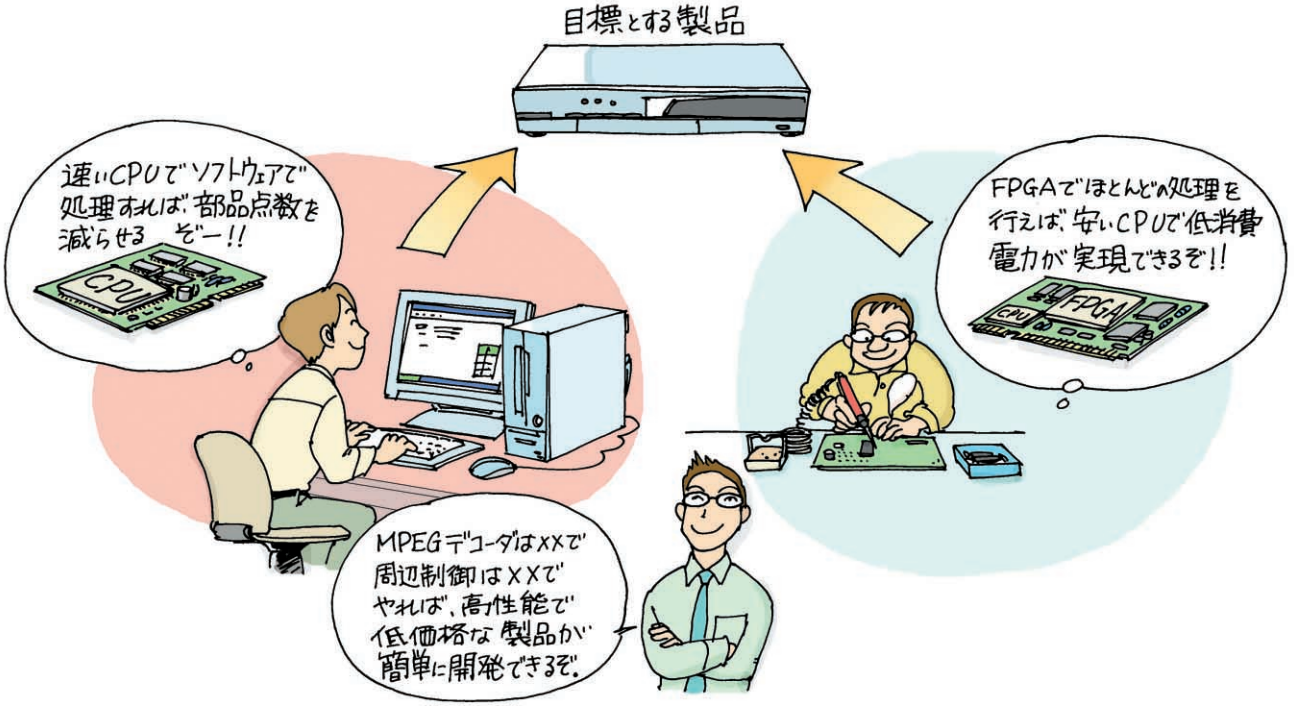
図6はチャタリング除去回路の例ですが，このように同じ目的を実現するためには複数の方法があります．どちらの方法を用いるか，ハードウェア技術者に任せっきりではなく，ソフトウェア技術者も積極的に意見を言うべきでしょう．

さらに，すでにハードウェア自身をVerilog-HDL（Verilog Hardware Description Language，**図7**）やVHDL（VHSIC Hardware Description Language，**図8**）などのソースコードとして記述する方向にあります．つまり，ハードウェアをコーディングする時代です．ハードウェアとソフトウェアの道具立ては似てきましたが，ソフトウェア技術者がハードウェアをコーディングできるわけではありません．ハードウェアに物申すソフトウェア技術者を増やさないと，システム設計は効率的に進まないでしょう．

システム設計にソフトウェア開発者が参加することで，ソフトウェアも含めた最適化が可能になります．また，最終的なシステム・テストでもソフトウェアの手法を応用できるなどのメリットを得られます．

● 時代の流れによる切り分けの変化

より広い意味でのシステム設計は，電気回路とソフトウェアの切り分けだけではなく，いわゆるメカとエレキとソフトウェアの切り分けを指します．ジェームス・ワットの蒸気機関では**図9**に示すような，回転数フィードバック機構が使われていました．ワットの時代では，メカのみで実現する以外の方法がな



New Products ――松下，ICカードの強度を従来より3倍以上向上するインレットを発売
松下電器産業(株)は，ICカードの強度を従来より3倍以上向上するインレット(フィルム上にアンテナとLSIを搭載したもの)のサンプル出荷を開始した．これにより，圧力や衝撃などに強い信頼性の高いICカードが実現できる．量産出荷は11月を予定している．

```
構造記述
module RSFF ( R, S, Q, Q_B );
    input R, S;
    output Q, Q_B;
    nor( Q , R, Q_B );
    nor( Q_B, S, Q );
endmodule


動作記述
module RSFF( R, S, Q, Q_B );
input R, S;
output Q, Q_B;
reg Q, Q_B;
always @( R or S )
    case({ R , S })
        1:begin Q <= 1; Q_B <= 0; end
        2:begin Q <= 0; Q_B <= 1; end
        3:begin Q <= 0; Q_B <= 0; end
    endcase
endmodule
```

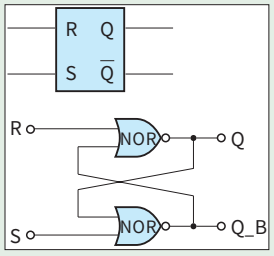


図7　Verilog-HDLで記述した例（NORの場合）

```
library IEEE;
use IEEE.std_logic_1164.all;

entity RS_FF is
    port ( R, S     : in  std_logic
           Q, Qnot  : out std_logic );
end RS_FF;

architecture STRUCTURE of RS_FF is

signal S1, S2 : std_logic;

begin
    S1      <= R nor S2;
    S2      <= S nor S1;
    Q       <= S1;
    Qnot    <= S1;
end STRUCTURE;
```

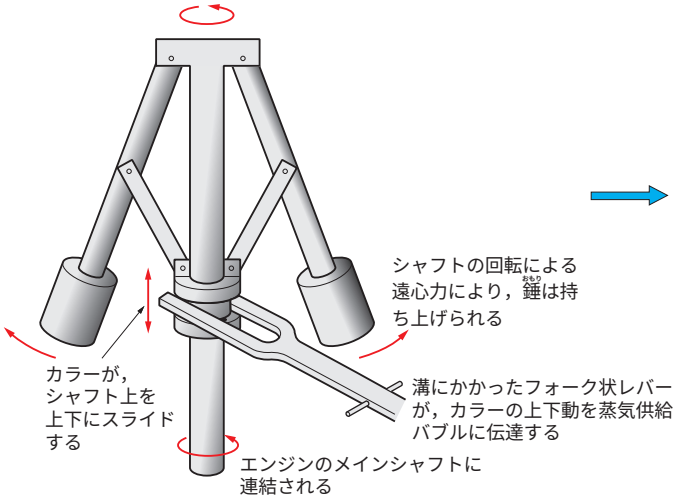
図8　VHDLで記述した例

かったのです．しかし，いまでは組み込みシステムを使うことで，よりきめ細かな制御が可能になっています．

20年ぐらい前の自動車は，ボンネットを開けてマニュアルを見ながら，ディストリビュータ（**図10**）のふたを開けたり，プラグを抜いてみたりして，楽しむことができました．しかし，最近の自動車はエンジンの周りにびっしりと機械が詰まっていて，素人には手が出せません．昔のように，ディストリビュータの角度を変えて，点火時期を遅らせてみるなどということはほとんど不可能になっています．

今は，電子進角装置（Electronic Spark Advancer）という組み込みシステムが，エンジン回転数（クランク角センサ），吸入空気量（エア・フロー・メータ）などエンジンの運転状態，エアコンを使っているかなどをセンサで検知し，点火時期の制御を行っています．ディストリビュータは回転センサ程度の位置付けであり，接点を紙やすりでピカピカにする楽しみはなくなりました．そのかわり，磨耗による点火タイミングの狂いなどは起きなくなりました．さらに，上り坂や，下り坂，急加速，冬と夏，暖気直後や大渋滞中など，状況に応じて，点火時期を制御できるようになっています．たとえば，始動直後は点火時期を遅らせたり，急な加速の際は点火時期を進めてパワーを出すなどです．機械だけではできなかったことが，できるようになりました．そして，さらに重要なのはこのような匠の技を，ソフトウェア資産として残すことができます．しかし，一方で，修理工場のおじさんがICEのような，デバッガのようなものを買わされて，それで調整することになります．組み込みソフトウェア開発者が，修理工場でバイトをする時代になるかもしれません．

ソフトウェアとハードウェアの切り分けは，非常に大きなテーマです．チャタリングを誰が取り除くかというローカルな問題から，修理工場のおじさんの運命まで左右しかねない大き



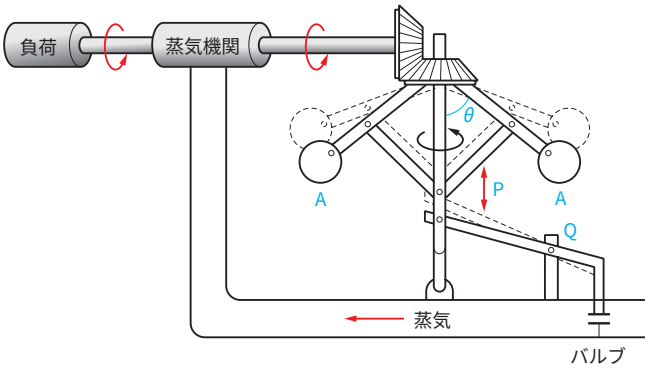
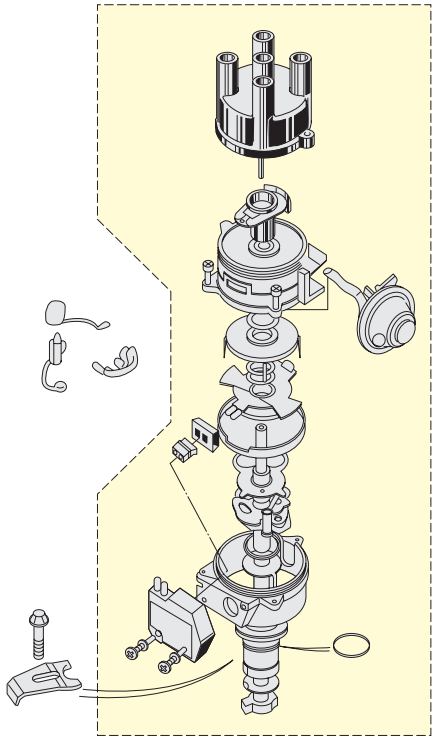


図9　機械系によるアナログ・フィードバック制御



点線内がディストリビュータ

図10
昔のディストリビュータ

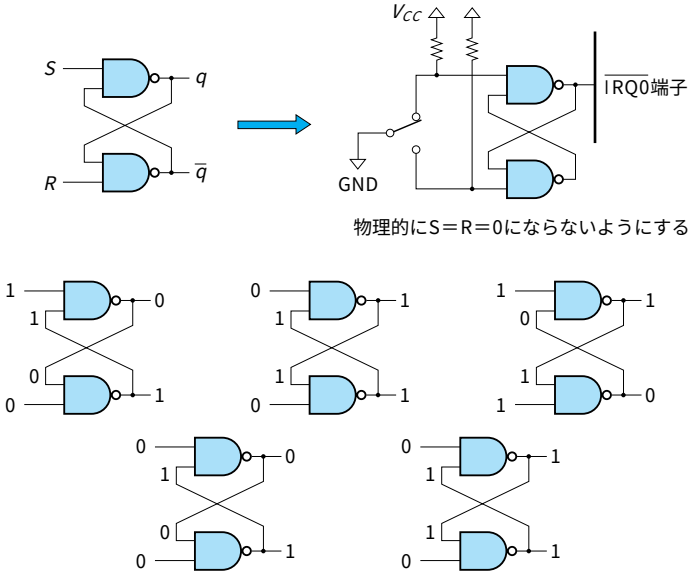


図11　RSフリップフロップの入力禁止

な問題まで，広範囲に影響を与えます．今のところ，開発コストと製品のパフォーマンスが切り分けのキーポイントになっています．開発コストは，製品のライフタイム全体の中でだれが何をするのかによって判断しなければなりません．ソフトウェアでやれば，ハードウェアの部品代が浮く，という時代ではないのです．パフォーマンスについては，設計の段階でどのようにパフォーマンスを見積もるかが重要になってきます．

3 想定外を想定する

● 想定外の状況が発生しても大丈夫なシステムとは

世の中には，何が起きても「想定内」にしてしまうタフな人が大勢います．良くできた組み込みソフトウェアも当然，タフでなければなりません．想定外の状況になっても，想定外になることは想定していた，よって，想定内なのです．

たとえば，**図11**のNANDゲートによるフリップフロップで，入力がS＝‘1’，R＝‘0’の場合，最初はqが‘1’でも‘0’でも，最終的な安定状態ではq＝‘0’となります．その後，R＝‘1’に変化しても，そのままq＝‘0’を保ちます．次に，R＝‘1’のままでS＝‘0’とすると，q＝‘1’となり安定化します．そして，S＝‘1’にしてもq＝‘1’が保存されます．つまり，S＝‘1’，R＝‘0’でq＝‘0’が確定し，S＝‘1’，R＝‘1’になってもq＝‘0’は保存されます．S＝‘0’，R＝‘1’でq＝‘1’が確定し，S＝‘1’，R＝‘1’になってもq＝‘1’は保存されます．これは，NANDの真理値表を傍らにおいて紙と鉛筆で確認できることです．

ところが，たとえば，S＝‘1’，R＝‘0’でq＝‘0’が確定した後で，S＝‘0’，R＝‘0’とするとq＝‘1’となりその後，S＝‘1’，R＝‘1’になってもq＝‘1’もq＝‘0’も取りうる可能性があり不定となってしまいます．すなわち，S＝R＝‘0’になることは，1ビットの情報を記憶するということに対しては想定外です．しかし，想定外であることがわかった瞬間，想定外ではなくなり対応可能となります．たとえば，**図11**に示したスイッチによって割り込みを発生させる回路のように，SとRをプルアップして，さらに，同時にGNDにならないようなスイッチを使用するのです．

想定外を想定するためには，すべての場合を尽くすくふうが必要です．そのためには，網羅的に状態の組み合わせを作ってくれるモデル・チェッキングのような形式的手法が有効です．

4 協調しながら同時に動かす

組み込みシステムの特徴の一つは，同時に動く複数のもので構成されるということです．そして，ただ同時に動くだけではなく，与えられた目的を実現するために協調して動きます．ここで，「動くもの」はCPUやデバイスなどのハードウェア，スレッドやプロセスなどのソフトウェアです．そして，協調して動くということは互いの動作を監視したり，タイミングを合わせたりするということです．

たとえば，本を読んでいて，あるページを読み終われば，次のページに進みます．そのためにはページをめくることになります．ページをどうやってめくるかについては，ほとんど意識しなくてもだれでもできることです．つまり，どうやって腕を伸ばすのか，どの指を使うのか，どうやってその指を曲げるの

Information ――T-Engineフォーラム，「ユビキタス食品情報基盤システム」の実証実験を開始
T-Engineフォーラムは，平成17年度よりコープさっぽろ，(株)三越，サミット(株)などと共同で，同フォーラムのユビキタスID技術を用いた「ユビキタス食品情報基盤システム」の実証実験を開始する．

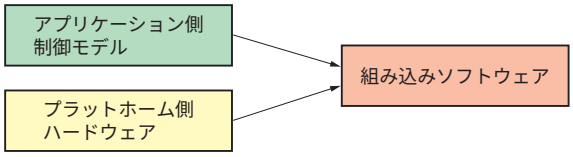


図12　二つの視点――アプリケーション側の制御モデルとプラットホーム側の制御モデル

か…我々は，これらの制御を同時に行っているのです．指を曲げるときには，内側の筋肉を収縮させると同時に外側の筋肉を弛緩させなければなりません．どこにどんな筋肉が付いているかまったく知らないけれども，協調しながら同時に動くように実装されています．「ページをめくりたい」と思っただけで，あるいはそんなことを考えることもなく，ただ本を読みたいと思うだけで自動運転されているのです．どうして自動運転できるのだろうと，考えすぎると何もできなくなってしまいますが．

このような組み込みシステムを作りたいものです．そのためには，協調しながら同時に動くものを扱う方法を学ばなければなりません．

5 制御モデルの作成を第一に考える

● 制御モデルの作成が重要である

実際に動くものを作ろうとすると，ハードウェアやソフトウェアが「わかった」だけでは，何もできません．たとえば，倒立振子を作ろうとしても力学モデルを作って，運動方程式を解いて，それに対して制御モデルを作って，それをソフトウェアで実現するとします．ソフトウェアで実現するときにはじめて，ハードウェアとソフトウェアの知識が役に立つのです．そして実装にあたっては，アプリケーション側の制御モデルと，プラットホーム側の制御モデルの二つの視点が必要になってきます（**図12**）．

組み込みソフトウェアの要求仕様には，制御モデル（**図13**）からの要求が入ってきます．たとえば，デッドライン要求などは制御モデルから出てきます．また，どのようなフィードバック系を作れば系が早く安定化するかは，制御モデルを見れば解答が書いてあり，これを使えば魔法のように解いてくれます．とは言っても過信してはいけません．メカやエレキの技術者が，ソフトウェア技術者に言えば何でもやってくれると誤解しているのと同じことになってしまいます．どのような制御モデルも最後の詰めは，試行錯誤で勝負するのです．そこから，制御技術者もソフトウェア技術者も学ぶのです．

業務系で言えば，ビジネス・ルールにあたる部分が，組み込みソフトウェアでは，物理モデルと制御モデルになります．このアプリケーション側からどのように要求を抽出するか，さらに，ソフトウェアでできることを逆に提案して要求自身を開発していくことがこれからは重要になってきます．

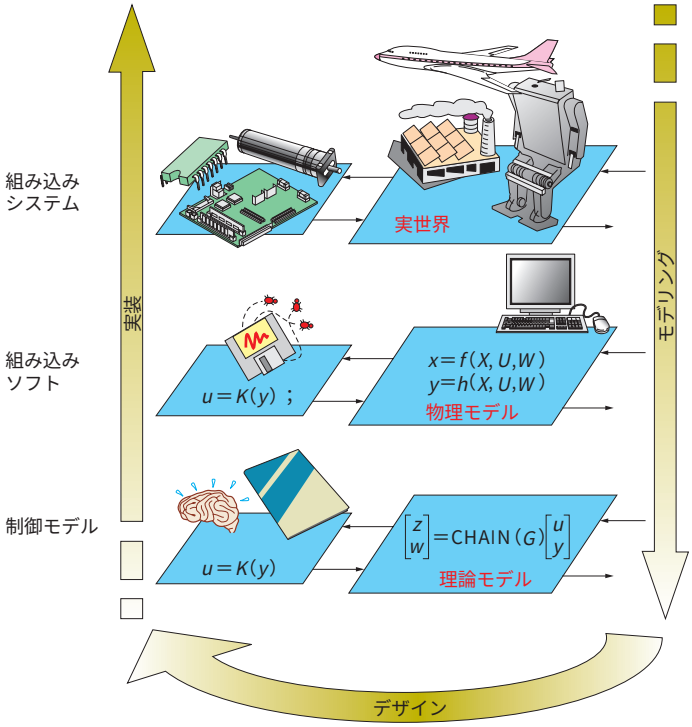


図13　制御モデル――最後の詰めは試行錯誤で勝負

● 制御技術者とも話をしよう

メカ・エレキ・ソフトのほかに，制御技術者といわれる人たちとも話ができなければなりません．制御技術者は，MATLABなどを使って普段からシミュレーションを行い，ソフトウェア面には詳しい人達です．したがって，メカやエレキの技術者よりは，話しやすいでしょう．

しかし，扱っている対象が微分方程式などの解析風のものが多く，変数はたいていfloatだったりします．それに実時間ではなくコンピュータの中に作り出した架空の時間スケールでモデルを動かしているなどの違いがあります．制御技術者と話をすると，普段，ハードウェア技術者たちがどんな気持ちでソフトウェア技術者と接しているか少しわかったりします．

結局，ソフトウェア技術者はいろいろなものの間に入ってインターフェースをとるのが仕事です．時間のギャップを埋めて，アナログとデジタルの間を取りもって，制御モデルをプラットホームにのっけて，などなど八面六臂（はちめんろっぴ）の日々が続きます．システム設計に一番近いところにいるのではないかと思います．

ふじくら・としゆき　組み込みコンサルタント