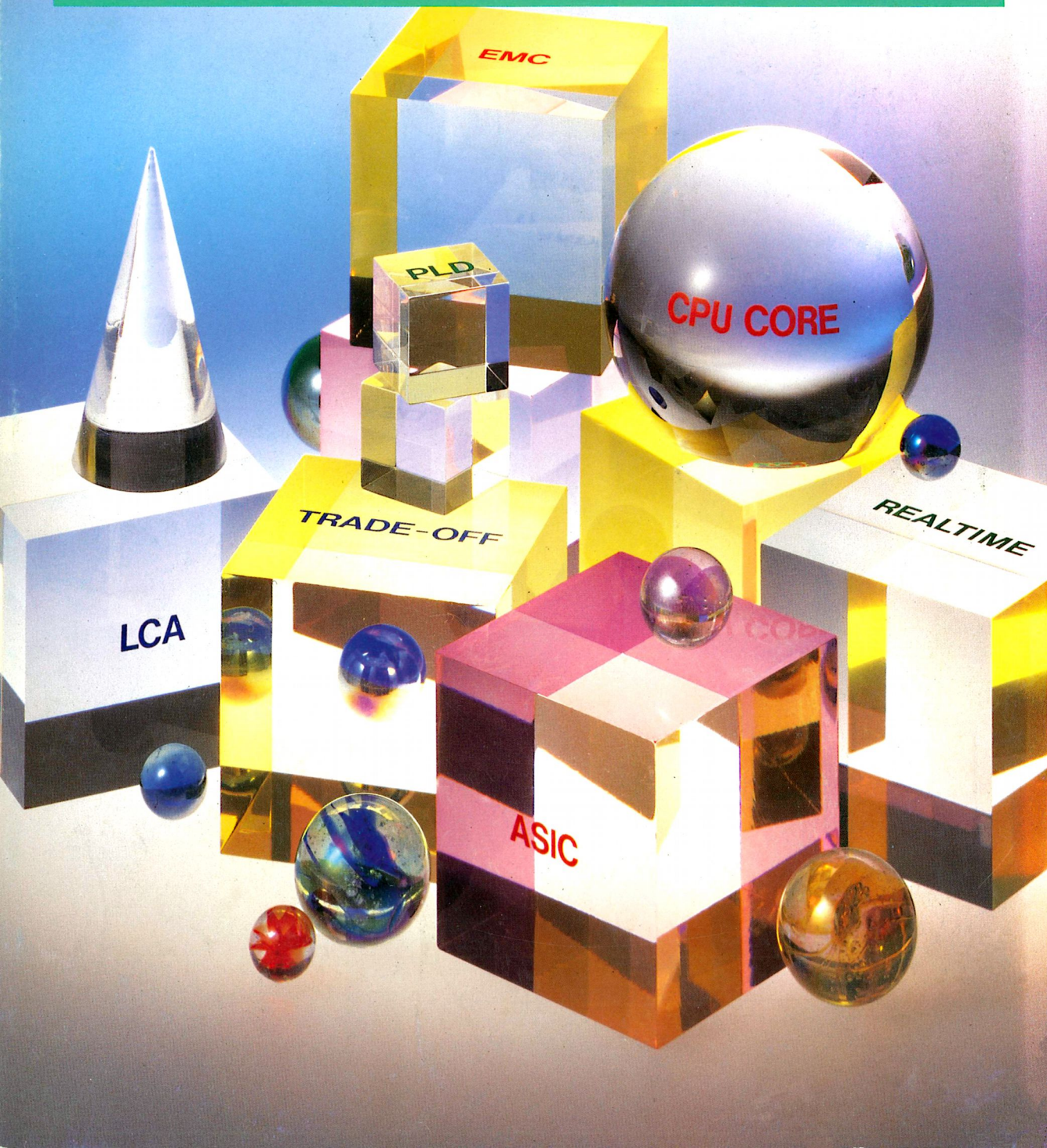
スキルアップ・シリーズ

SKILL UP

マイコン技術者スキルアップ事典

ハードに強いエンジニアになるためのデータバンク

長嶋洋一 著

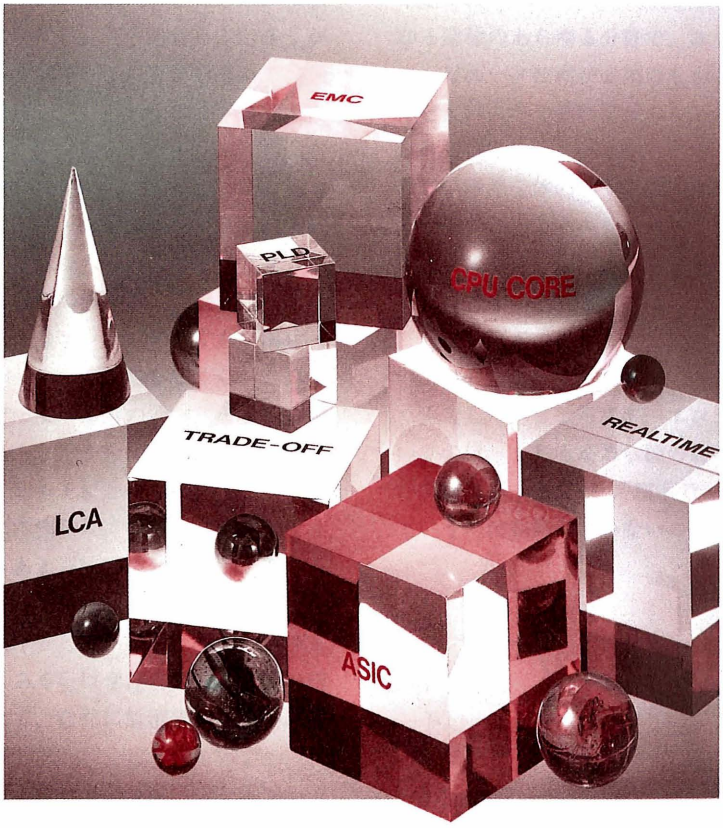


SKILL UP

マイコン技術者スキルアップ事典

ハードに強いエンジニアになるためのデータバンク

長嶋洋一 著



CQ出版社

＜プロローグ＞

フレッシュマンへのエール

●新人の「技術不安」

本書は，エレクトロニクスに関係するメーカへの就職を考えている学生・技術者の卵の皆さん，メーカに就職したばかりの新人エンジニアの皆さん，そして入社2～3年までの若手技術者の皆さんのために企画されました．エレクトロニクス技術・コンピュータ技術が激しく進歩するこの時代に，若手の皆さんの多くが体験する「技術不安」を解消して，自信をもってエンジニア人生に出帆してほしい，というのが本書の大きな目標です．

20世紀後半に人類史の表舞台に登場したコンピュータ技術は，現在では工業・商業ばかりでなく，人間のあらゆる活動になくてはならないものとなりました．その進歩のスピードと内容の複雑さ・巨大化は，すでに現場で活躍している先輩エンジニアですら，消化・吸収する前に新技術がどんどん登場してくる，と思うほどのプレッシャとなっているものです．そしてこの世界にこれから飛び込もう，または飛び込んでみたばかり，というフレッシュマンにとっては，なおさら直面する技術の膨大さに圧倒されてしまうことでしょう．「技術不安」になるのもある意味で当然ですし，もちろん筆者も同じような不安を体験しました．

しかし，漠然とした不安というのは，相手の正体をよく見極めることで，あるいは全体の姿をやや高い視点から眺めてみることで，自然に解消できるものだと思います．そこで本書では，

「マイコン・システム技術者の仕事地図を示す」
「エンジニアとしての生き方を考える」

という，ちょっと従来の技術書とは異なる視点を心がけてみました．具体的な「回路テクニック」とか「プログラム・リスト」はあまり多くありませんが，そういう技術情報を自分から効果的にアクセスしていく姿勢を身につけることを期待しているのです．さあ，つまらない「技術不安」にサヨナラして，前向きなエンジニアを目指そうではありませんか．

●専攻外・文系出身の意外な強み

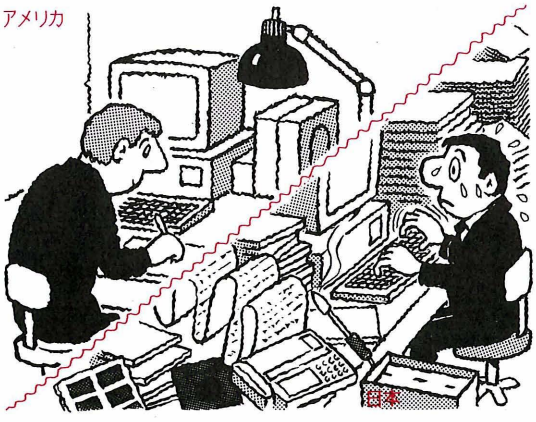
ここ数年来の社会傾向として，慢性的なエンジニア不足が続いています．大企業から中小企業まで，あらゆる業種のあらゆる分野で，エレクトロニクス技術やコンピュータ技術が必須のものとなってきたために，たとえば大学や専門学校の電気・電子・情報系の学生だけでは不足しているからです．そのため，専攻を問わずに理系全体の学生を広く求めたり，さらには文系の学生であってもエンジニアとして採用し，社内研修によってプロとして再教育するケースも多く見られてきています．

しかし筆者の意見としては，「エンジニア＝専門分野の出身」などという図式は，人材難の状況とは関係なく否定したいところです．エンジニアとしての適性というのは，もちろんある種の論理性や継続性，といった側面はありますが，「理系でないから向かない」などというのは誤解でしかないのです．というより，むし

ろ専攻出身でないことがメリットとなる点もある，とさえ思います．これからの時代，文系出身のコンピュータ技術者も当たり前になる，と確信をもって予言しておきましょう．

じつは筆者自身，大学での専攻は物理学(原子核物理)で，メーカに就職して電子関係のエンジニアとしてスタートした時点では，マイコンもディジタルもソフトもハードも，すべて白紙の状態でした．しかし物理屋というのは，「道具がなければまず道具を作ればいい」という，したたかな柔軟性(これを称して「物理する心」という)をもっています．知らない知識は臆せず先輩に聞き，知らないデバイス・装置は実験・分解して理解し，知らないコンピュータは触って覚え，知らない理論は文献から調べればいい，という「自由な感覚」ならもっているわけです．これは現在でも変わっていないらしく，コンピュータ言語はちょっと使わないと忘れてしまいますが，マニュアルさえ開けばすぐに現役バリバリにもどる，というぐあいで，「体得した一定の技術を後生大事に守る」よりも，「新しい技術の大海を漂い泳ぐことを楽しんでいる」のです．

ところが逆に，電気・電子・情報などの専攻の新入社員の中には，「ここは自分の専攻と違うのでわからない」というような，異分野への抵抗感をもつ人もいます．これは完全なマイナス面で，プロのエンジニアであれば，どんな新しい目標・新しい手法・新しい技術に直面していくかわからないのですから，自分のごく僅かな「過去の蓄積」など，忘れてしまったほうがいいのです．とくにコンピュータ技術などというのは日進月歩，どんどん新しくなっていきますから，むしろ「自分の専攻は役に立たないぞ」と自覚してスタートするぐらいのほうが，あとあとプラスになると思います．



●エンジニア冥利とは

まだ学生の皆さんにとっては，プロの技術者の実際の姿がよく見えないかもしれません．たしかに，研究室や実験室の中で，黙々と毎日コンピュータや計測機器に囲まれて研究・開発・設計の作業に従事しているエンジニアの姿というのは，一般にわかりにくいものでしょう．技術者がいつも格闘している実験ボードや試作機というのは，最終的に完成されたスマートな製品とは似ても似つかない手作りのバラックであったり，製品の基板上にある切手サイズ程度のカスタムLSIが，開発段階では1㍍四方の基板の3枚セットで，そのあちこちから修正の配線がわさわさと出ている「ジャングル」であったりします．

アメリカの研究者のスマートなオフィス(机の上に足を伸ばして，膝の上のキーボードを叩く)を想像するのも，たいていは大きな誤解です．実験室の地味な作業机には，多くの開発装置と実験ボードと部品が置かれ，ハンダごてやテスタが並んでいます．ソフト寄りの仕事場であれば，ファイルとプログラム・リストとフロッピが雑然と積み重なっている，という風景が普通のものでしょう．また，スケジュールに追われて残業が多いのも事実なのですが，「会社にいわれてやる」というよりも，「熱中するとついつい残業しても続けてしまう」という傾向が，エンジニアの多くに共通したところのようです．

エレクトロニクスにしてもコンピュータにしても，正しい設計をすれば確実にその通りに動いてくれるから，人間味というか偶然性の面白味がないじゃないか，といわれたことがありました．たしかに理屈はそうなのですが，残念ながら実際には，なかなか新システムはうまく動いてくれません．ほとんど人間的とも思えるほど，開発途上のシステムのきまぐれなトラブルはいろいろと登場してきます．正しい「道具」であるはずの計測器や開発用コンピュータの欠陥を，トラブル追求の最中に発見することさえあります(筆者の場合，ついついこのトラブルとの格闘を楽しんでしまいます)．

やがて，システム全体が自分のモノとなり，なにより「製品」という具体的な形となって，世の中に送り出されることになります．これが，技術者としての手応えを感じる頂点といえるかもしれません．そして，

新システムが完成した時点で，それを開発した自分自身が，エンジニアとしてより成長している，というところも大切です．つまり，「同じ機能のシステムを開発しなさい」というテーマがもう一度与えられたとすると，今度は１ステップ成長したその技術者によって，よりスマートで高性能でローコストな製品が，より短期間に美しく開発されることになるのです．開発対象とともに自分自身が成長していくところも，エンジニアという仕事の大いなる魅力だと思います．

●柔らかな頭と熱き心と

さて，それでは「エンジニアとして必要な姿勢」には，どんなものがあるでしょうか．なにしろ相手は深い深いエレクトロニクス技術・コンピュータ技術の世界ですから，膨大な技術情報に圧倒されずに，冷静に少しずつ理解していこう，という地道な姿勢はもちろん大切です．また，新しい概念には予備知識に頼らずに素直に接するとか，面倒がらずにマニュアルやデータブックを参照する，というのも，たとえば語学の初学者が「まめに辞書を引きなさい」といわれるのと同様の基本でしょう．電子回路とかコンピュータの動作というのは「規則通り」の世界ですから，理詰めで論理的に考える，という姿勢も大切です．

しかし，これら「当たり前の条件」以上に筆者が求めたいのが，「柔らかな頭と熱き心」という生き方です．いわゆる「工学系人間」というのは，上に述べたような真面目な点ではピカイチなのですが，いざというときやトラブルに直面したときの柔軟性とか，新しいシステムを検討する際の自由な発想とか，試行錯誤や新アイデアを楽しんで遊ぶ，といった余裕とかの面で，どうしても「固い」印象があります．理系アウトローの物理屋とか，文系出身(文学部の論理学専攻などは，コンピュータのプロとして素質十分)のエンジニアのほうが，そういう自由な発想の場では活躍した例も多く見かけています．この「柔らかな頭」を，フレッシュマンの誰にも期待したいところです．

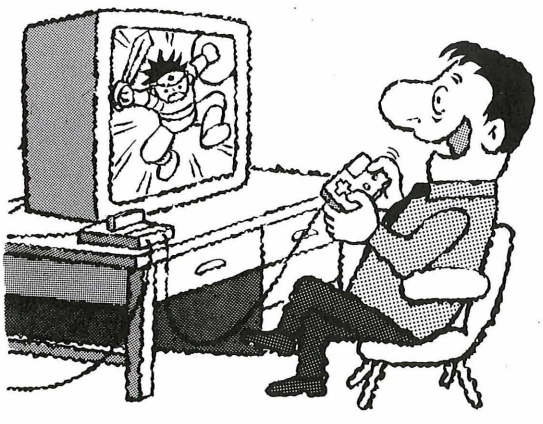
そしてもう一つのポイントが「熱き心」です．毎日ほとんど無言でコンピュータに向かっている技術者，というと，なんだか暗くて，ロボットのような冷たさをイメージしてしまうかもしれません．たしかに，会社から給料をもらうのと引き換えに自分の人生から一定の時間と労力を提供して，与えられた仕事を黙々とこなす「サラリーマン・エンジニア」もいるかもしれません．しかし筆者は，これではあまりに淋しい，もっと前向きに，エンジニアとしての人生そのものを楽しんでしまおうではないか，と提案したいのです．

たとえばマイコン応用のシステムを新しく開発する，という「業務」は，仕事と考えるとプレッシャでもありますが，こんな楽しい「ゲーム」はちょっとない，という見方だってあるのです．技術者である自分自身がロールプレイング・ゲームの主人公となって，会社のカネで部品や開発機材をふんだんに駆使して，個人の趣味ではとても出来ないような費用と期間を集中してシステムを完成に導いていく，と考えると，こんな楽しいゲームは他にないでしょう．事実，多くの技術者が仕事に熱中する原動力は，会社のためでもカネのためでもなく，彼ら自身が，子供のようにその仕事で「遊んでいる」ところにあると思います．大きな声でことさらいう必要はないのかもしれませんが，仕事をゲームとして楽しんでしまいましょう．

●本書の構成について

コンピュータ技術・エレクトロニクス技術の世界のフレッシュマン技術者のために，という壮大な目標をもった本書ですが，実際にはエンジニアといっても，多くの業種・多くの職種に対応した，いろいろな仕事があります．そこで，ある意味で最大公約数となる対象として，本書では「マイコン・システム技術者」というポイントを設定してみました．これは，ハードからソフトまでをこなして，専用の組み込み機器を開発するシステムハウス(一品料理の特注システムを受託開発する技術者集団)の技術者というのが，まさにそのものズバリです．しかし本書の扱ういろいろな技術領

仕事をゲーム感覚で楽しもう!?



域は，ソフトハウスのプログラマであっても，システム設計のSEであっても，量産ベース民生機器のハード開発者であっても，いずれもこれからの時代には必要となるものです.「自分の技術フィールドを限定することが，エンジニアとしての可能性の芽をつむことになる」という重要な視点を忘れずに，広い視野をもつための「事典」として活用していただければ，と思います.

全体の構成としては，まず＜イントロダクション＞として，「マイコン・システム」と「マイコン技術者」の定義からスタートし，概論として，その「仕事地図」をざっと眺めてみます．個々の技術的な内容については，最初はピンとこない部分もあるかもしれませんが，まずは「エンジニアの仕事」の全体像をつかんでみる，という作戦です．そして，つぎにマイコン技術者として必要ないろいろな技術項目について，「技術レベルのステップアップ」として，これもざっとチェックしていきます．個々の具体的な技術内容は＜事典＞のなかで取り上げられていきますから，これら個々の技術(縦糸)を別の視点からくくった横糸，と考えてみましょう.

そして本編である＜事典＞とは，マイコン技術者がいろいろな局面で必要になる技術として，まず最初に

(1) マイコン技術者のための基礎概論

という項目で予備的な基本技術をまとめ，ついで

(2) パソコン活用関連技術
(3) マイコン・システム実戦テクニック
(4) チップ関連技術
(5) 信号と信頼性の技術
(6) 情報収集テクニックとドキュメント技術
(7) ASIC技術
(8) 並列処理・分散処理とネットワーク

の8項目に分類して，それぞれ関連する技術ポイントを集めてみました．これらの項目はそれぞれ独立したオムニバス構成となっていますから，本書のどこから読んでも(つまみ食いでも)，それなりに役立つようになっています.

また＜教科書＞でなく＜事典＞としたのは，フレッシュマンにとって技術の全体に触れてみる材料として，さらに実務の現場においても参考になるように，という目標のためです．そこで，個々の内容が理論的に整然と網羅されている，というよりも，現場で役立つテクニックとか，あまり理論書には書かれない実際的な視点，といったものが多くなるように心がけています．中堅エンジニアにとっても，なにかの折りに本書をパラパラと眺めてみれば，その時点での自分の技術レベルの確認に利用していただけるでしょう.

●「エンジニア万歳！」に向かって

まだフレッシュマンの皆さんにとって，「エンジニアとしての人生目標」など想像がつかないかもしれません．エレクトロニクス技術やコンピュータ技術の歴史が若いために，お手本となるべき先輩技術者もまだ現役の人ばかりで，それら先輩エンジニアが，これからどういった「理想」の姿をわれわれに見せてくれるのかわからないのも事実でしょう.

ただ，次第に充実してきている「技術者のための支援環境」によって，これからのエンジニアの日常がより快適に，本人の能力をより効果的に引き出す方向に向かうことだけはたしかなようです．技術者がEWS(エンジニアリング・ワークステーション)ネットワークの環境によって周囲の世界と結びつき，知的なコンピュータ支援によって効率的に高度なシステムを開発していく…こんなシーンが未来予測として語られています．筆者などはこれに対して，実現されていく際にはまだかなりの曲折があるだろう，などと多少の経験から意地悪に見てしまいますが，基本的な傾向としては，技術者の環境の未来は明るいと思います．フレッシュマン技術者は，これまでの先輩技術者が望んでも得られなかった，高度で快適な開発環境を約束されているわけです.

さて，エンジニアとしてのスタート付近にいる皆さんのための＜プロローグ＞は，これでおしまいです．勇気と元気が湧いてきたでしょうか．ここからこの世界に飛び込んで，自分なりの「エンジニアの理想像」を模索しながら頑張るのは，あとは皆さん次第ということになります．10年選手として「エンジニア万歳！」と実感するようになった筆者からの，スタート時点でのエールは十分に贈ったつもりです．皆さんのこれからに期待し，先輩としてライバルとして，心から応援したいと思います.

マイコン技術者スキルアップ事典

CONTENTS

<プロローグ>

フレッシュマンへのエール ……3

<イントロダクション>

マイコン・システム技術者の世界 ……11

マイコン技術者の仕事地図 ——11

目標技術のステップアップ ——16

<コラム> マイコン技術者の徒弟制度 ——18

<飛び石コラム> おすすめBOOKS ① ——21

<Appendix> ある「技術勉強会」のメニュー ——22

事典編

1．マイコン技術者のための［基礎］概論 ……32

ディジタル技術の基本 ——32

<コラム> 解答1　符号付き乗算回路の例 ——38

<飛び石コラム> おすすめBOOKS ② ——39

アナログ技術の基本 ——41

マイコン・システムの3形態とトレードオフ ——46

ヒューマン・インターフェース考 ——53

2．パソコン活用システム関連技術 ……58

ハードウェア技術の基本 ——58

ソフトウェア技術の基本 ——60

外部システムとのインターフェース技術 ——67

<コラム> リアルタイム・システム ——70

<飛び石コラム> おすすめBOOKS ③ ——71

3．マイコン・システム実戦テクニック集 ……72

システム設計の実例 ——— 72
開発環境：ICE ——— 77
開発環境：モニタ ——— 79
ソフト・テクニック：ワンパターン処理 ——— 81
ソフト・テクニック：FIFO 処理 ——— 85

4．すべてはチップから：「チップ」関連技術 ……88

周辺 LSI ——— 88
＜コラム＞ 解答 2　可変クロック発生回路の例 ——— 90
メモリ ——— 95
1 チップ・マイコン ——— 96
セミカスタム IC ——— 98

5．「信号」技術から「信頼性」技術へ ……105

アナログ ⟷ ディジタル変換の考察 ——— 105
インターフェース信号の検討 ——— 108
信頼性の技術 ——— 110
＜コラム＞ 境界値分析 ——— 112
EMC 技術 ——— 113

6．情報収集テクニックとドキュメント技術 ……116

あるエンジニアの情報収集の例 ——— 116
技術情報の活用テクニック ——— 121
エンジニアのドキュメンテーション技術論 ——— 123
＜コラム＞ プロジェクト管理も重要な技術 ——— 127
＜飛び石コラム＞ おすすめ BOOKS ④ ——— 128

7．上級システム：ASIC 技術 ……129

ゲートアレイからスタンダードセルへ ——— 129
<コラム> ビットスライス CPU ——— 134
CPU コア内蔵「究極チップ」へ ——— 136
「究極チップ」の技術ポイント ——— 139
セミカスタム CPU からのアプローチ ——— 143
カスタム CPU への道 ——— 148

8．並列処理・分散処理とネットワーク ……150

ネットワーク化によるシステム性能向上 ——— 150
ボード・マイコンによる分散処理の例 ——— 153
チップによる並列・分散処理の例 ——— 158
ASIC 化による分散処理 ——— 162
<Appendix> 体験的 ASIC 開発入門 ——— 165

<エピローグ>
エンジニア万歳！ ……171

INDEX ——— 175

〈表紙デザイン〉アイドマ・スタジオ〈表紙撮影〉スタジオ DAX 〈本文イラスト〉榎本幸一郎

＜イントロダクション＞

マイコン・システム技術者の世界

マイコン技術者の仕事地図

●マイコン・システムとは

本書では「マイコン・システム技術者」を対象としていきますから，まず最初に「マイコン・システムとは」という定義について述べることにしましょう．

ひとことで定義すれば，ここで取り上げるマイコン・システムとは，一般にCPUとかMPUと呼ばれる，マイクロプロセッサ(またはマイクロコンピュータ)を使用したシステム，ということになります(**図0.1**)．

たとえば，パソコン(パーソナル・コンピュータ)は立派なマイコン・システムですが，エアコン・炊飯器・洗濯機などの家電製品やカメラ・自動車・電話機などでも，機能強化のためにマイコンを内蔵するのは常識となっています．そしてもちろん，事務処理，工場の自動化，通信機，ゲーム機などもマイコン技術の結晶です．「コンピュータ」＝「電子計算機」という名前の神通力はもはや過去のもので，現在ではあらゆるエレクトロニクス機器のたんなる「部品」として，マイクロプロセッサをふんだんに使うのが当たり前になっています．

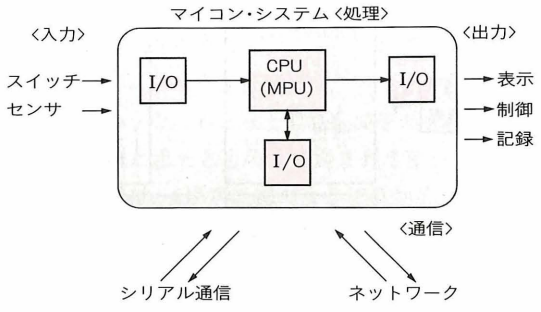
そして，CPUを使うという内部の条件とともに，「システム」として外から見る視点も重要です．マイコン・システムとは**図0.1**のように，全体の動作を制御する中央のCPUとともに，システムの外部(外界)とCPUとが情報交換するためのI/O(インターフェース)処理部分があり，システムとして「外界となんらかのやりとりをする」性質があります．パソコンのキーボードから入力するとか，CRTディスプレイから表示出力するとか，通信ポートを介して外部と通信する，といった動作が，ここでの外部インターフェースの典型的な例です．

また，簡単なマイコン・システムでは**図0.1**のように1個のCPUからなりますが，2個とか3個以上のCPUが分業・協力しながら，全体として一つのシステムを構成するような「マルチプロセッサ・システム」も多く，たとえば自動車では数個から十数個のマイコンが搭載されています．また，マイコンには荷が重い複雑・高速な処理を担当する専用のハードウェア(チップ)をもつもの，複数の独立したマイコン・システム同士が通信ネットワークを介して共同作業(分散処理)をするシステム，反対にたった1個のチップにすべての回路機能を盛り込んだ電卓のようなシステムもあります．なお，ここでの定義によれば，ワークステーションのような大規模システムまで含まれることになりますが，本書では，一人の技術者が相手にできるぐらいの規模を考えていくことにします．

●マイコン技術者の守備範囲

マイコン技術者の仕事の領域を具体的に考えてみるために，**図0.2**の例について見ていくことにしましょ

〔図0.1〕マイコン・システムとは



う．この例では，調査・研究からはじまって，システム全体の設計・開発段階，さらにシステムの各部分の具体的な設計・開発段階，そして製品の生産につながっていく試作・まとめの段階まで，時間とともにステップが推移していくものとしています．また，各ステップの中でも，いろいろな分野での作業が横に広がり，平行して進められることがわかると思います．

もちろんフレッシュマンが，いきなりこの仕事を全部任される，ということはありません．現役の技術者の大部分が，この領域のどこかの専門家として活躍しているわけです．また，どちらかというと大きな企業ほど，これらの仕事領域の細分化・専門化が徹底しているために，**図0.2**の全部の領域を一人のエンジニアが担当する，などということは希であるかもしれません．この意味で，「小さなシステムハウスの技術者のほうが，全体を見渡す仕事をすべてカバーすることになるために，むしろ技術的には面白いものだ」という意見もあります．

設計・開発の段階については，

(1) システム全体の設計・開発段階

(2) システムの各部分の具体的な設計・開発段階

と２段階に分けてありますが，図の左側の部分というのは，ソフトウェアであれば「上級 SE」の仕事に対応する，という見方によるものです．もちろん，この部分を担当するためには，具体的な設計・開発技術が十分に身についている必要がありますから，ここは中堅・ベテラン技術者の世界と考えられます．若手技術者の皆さんは，基礎的な技術力をアップしながら，この領域の仕事に進出していけるように，先輩のテクニックを吸収していくことになるでしょう．

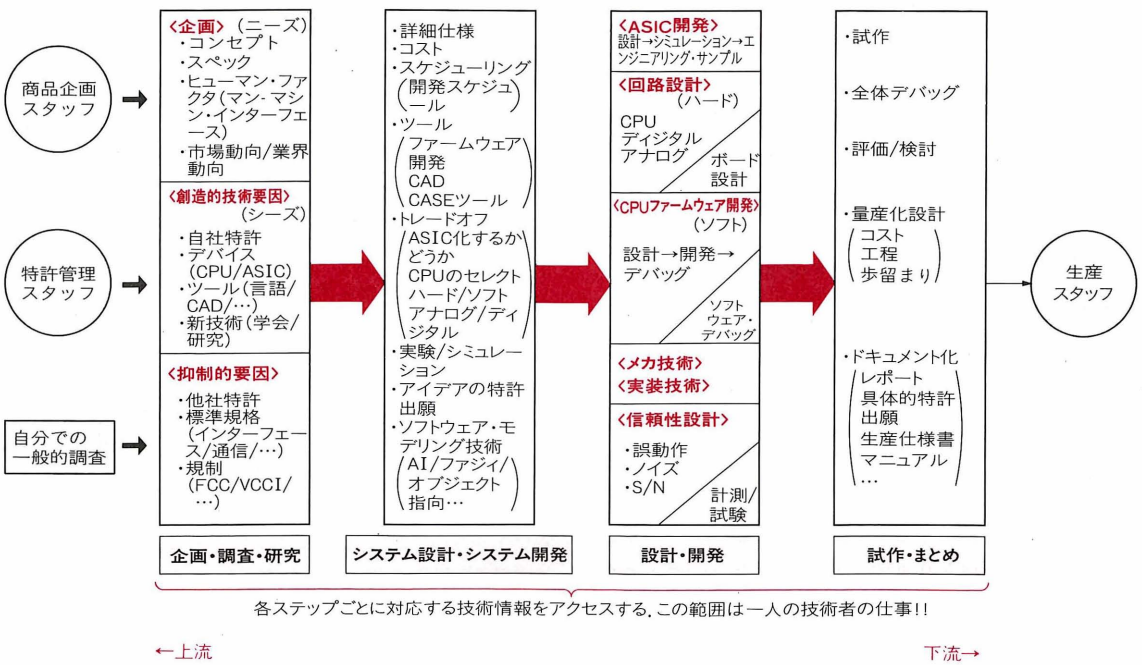
また，最終段階として「試作・まとめ」という呼び方をしていますが，これは前段の具体的な設計・開発作業と明確に区分されるものではありません．しかし，ここでは実際の製品として完成させる作業とともに，この開発によって得られた技術的な収穫を，つぎのステップでの活用に向けてまとめる，という重要な性格があるために，あえて独立のものとしてみました．

それでは，**図0.2**の仕事の流れについて検討していきます．

●企画・調査・研究

図0.2のステップのいちばん上流にあるのは，「企画・調査・研究」という仕事です．ここでは，具体的なシステム開発のテーマとは一見関係ないような，一

〔図0.2〕 マイコン・システム技術者の仕事(守備範囲)は？



般的な技術情報(学会の論文集・研究報告とか特許公報のような技術情報)を扱います.経験のあまりないフレッシュマンにとっては親しみのない,どこか縁のない世界のもののように思えるかもしれませんが,ベテラン技術者にとっては,直接は関係のない特許資料や学会発表の中から,将来の具体的実現のためのヒントをかぎつけるきっかけにもなるのです.

新聞や雑誌に「新技術」などと取り上げられる話題は多くありますが,たとえばあまりに高価で非現実的だったり,信頼性や精度の点で実用的でないような,「アイデア先行」の技術も多くあります.ところがプロの技術者は,マスメディアよりもかなり前から,これらの新技術の情報を入手して,さらにそれを現実的なシステムとするための実験・開発を進めることさえあるのです.

たとえば,「A 大学で基礎実験に成功した光センサが,1年後に B メーカの新製品コピー機のスキャナに応用されていた」というように,一般の目には見えない水面下で進むことになりますから,現代の企業戦略としても重要なものです.この「企画・調査・研究」活動は,新しい技術を前向きに取り入れていく技術者にとって,必須の業務といえるでしょう.

また,フレッシュマンが現場で直面するものの中に,各種の「**規格**」があります.これは,複数のメーカがそれぞれ勝手に独自のインターフェースやフォーマットを用いると,ユーザが混乱してしまうために,協定のもとに共通規格として,お互いに規定しているものです.メーカの技術者とすれば,新システムを開発する場合には,この規格というのは開発の自由度を制限する,抑制的な要因として関与してくることにもなります.しかし一方で,「規格」の適切な活用は,開発の効率化と信頼性の向上を同時に得ることにもつながります.たとえば,「まったく新しいアイデアの事務機器を開発する」という場合でも,入力装置や表示・印刷装置,あるいは外部との通信ケーブルや信号フォーマットをまったく独自のものにしたら,全体の設計・開発コストが巨大になってしまうだけでなく,すでに一般に出回っている機器が使えませんから,おそらくそんな「新製品」は市場に受け入れられないでしょう.この意味で,各種の規格をよく理解して活用していくことは,コンセプトのオリジナリティとは別の次元の問題として,重要なことなのです.

●ニーズ指向とシーズ指向

新しいシステムを生み出すための企画・調査という領域の仕事では,二つの方向性があります.

ひとつは,市場やユーザの希望・要望,つまり「新システムに求められるもの」という「**ニーズ**」を調査・分析して,それを満足するための新製品を企画・開発していく,という手法です.このためには,たとえば営業セクションを担当するスタッフとの情報交換の中から,自社の得意な技術によって実現可能なものを絞っていく,というような視点が重要となります.やはり,これもベテラン技術者の仕事でしょう.

もうひとつは,日頃から研究・開発している新しい技術(基礎技術を含む)の中から,そのユニークな技術をうまく応用することで,独自の付加価値をもった新製品の企画・開発を提唱していく,という自社開発技術(**シーズ**)からの発想もあります.このためには,目先の製品開発にふり回されずに,もっと長いスケールでじっくりと研究・開発に取り組む姿勢も必要となりますが,特許・ソフトなどの知的所有権意識の高揚している現代では,シーズ指向というのはかなり手堅い戦略であるともいえます.

エンジニアのシーズ指向のためには,つねに世の中の進歩をチェックするとともに,大学や研究機関の研究の先端状況を把握することが重要です.また,たとえば従来は1チップに入りきらない規模のためメリットの少なかったユニークな回路が,LSI メーカの微細化の進展によって実現されるといった,他社・他業界との関係によっても,状況はつねに変化していきます.技術者は実験室にこもるばかりが能ではなくて,このような情報収集のために,セミナや研究会などをチェックして歩き回ることも重要なのです.

●システム設計・開発

設計・開発のステップは,開発プロジェクトであればプロジェクト・リーダの仕事に相当するものです.その中には,開発作業の**スケジューリング**や**コスト検討**のような**プロジェクト管理**とか,アイデア段階での**特許出願**のような,ちょっと製品開発そのものの技術要因とは別種と思えるものまで含まれます(これらの技術については,<事典>編のなかで具体的にふれています).

そしてこの段階で一番重要なのが,後述する「**トレードオフ技術**」を活用した,システムを大きく切り分

けて構成していく作業になります．たとえば，ある仕様をもった製品システムは，ベテラン技術者にとっては，何通りも実現の方法が考えられる，というのが普通です．その中から，ソフトとハードをどう組み合わせ，どんなデバイスとどんなソフトウェア技術とを使って，さらにどんな開発手法で試作から生産まで進めていくか，という全体構想を，バランスよく構築していくものです．簡単な例でいえば，ある処理をソフトで行うのか専用ハードを設計するのか，という選択にしても，

- 性能・処理速度
- コスト(開発コスト・量産コスト)
- 開発期間
- 信頼性・規格

などの種々の要因が関係していますから，なかなか判断が難しいものなのです．そして，ここでのシステム構想が決定されてしまえば，もうあとでは変更がきかない場合が多いだけに，決定の際の責任も重大です．

また，「開発環境」とまとめていわれていますが，開発ツール，開発支援装置，開発言語，デバッグ手法などの構想を決定することも，この上流ステップでの重要なポイントとなります．もし開発環境の選択を間違うことになれば，ある意味でほとんどプロジェクトの失敗につながってしまいます．いくら各技術者が頑張っても，適切でない開発環境では，時間も労力も能力も，文字通り無駄使いになってしまうのです．開発環境は日々進歩していますから，この状況にアンテナを立てるのも，技術者の重要な仕事なのです．

●具体的設計・開発

一般的に「技術者」といえば，この具体的な設計・開発ステップの仕事が想像されるもののようです．いわゆるハード屋であれば，試作基板に部品をハンダ付けしてはシンクロのプローブを当てる，あるいはソフト屋であれば，毎日コンピュータの画面を眺め続けている，というようなイメージではないでしょうか．しかし，**最近のマイコン・システムというのは，どうもハードとソフトの境界があいまいになって，ハードだけのハード屋もソフトだけのソフト屋も少なくなってきました**．つまり，あらゆる技術者が，電子回路技術からソフト開発までをカバーしてしまう，逆にいえばハードからソフトまでを守備範囲としないと，満足な仕事ができないようなレベルになっている，という状況を理解しておく必要があります．

そして，最近のトレンドである**ASIC**(あるいはその前段の**PLD**や**LCA**．それぞれ＜事典＞編で詳しく解説があります)に関する技術の場合，ある機能を特定のハードウェアとして実現するために，システム開発者が「ソフト的にハードを設計する」という仕事がメインとなります．たとえば，ちょっとした事務機器の簡単なシステム開発でも，試作ボードに3台のパソコンと1台のICEを接続して，ASIC搭載ボード設計のために数種類のソフトを同時に駆使して作業を進める，といった風景になります．筆者は英語も満足に使いこなせない典型的日本人(？)ですが，ASICがらみのシステム開発のピーク時には，同時に数種類の言語を駆使するバイリンガル技術者に変身します．このメリハリは，初めて見た新人には驚異でしょうが，これからの時代，やがて誰もが体験していくことになるでしょう．

また，開発という仕事に対して，たんに目的とされる機能を設計して試作すればおしまい，というイメージしかもっていないとしたら，フレッシュマンの皆さんは考え方を変える必要があります．

一人の人間の頭が同時に考えられる物事の数には限りがありますが，マイコン・システムの場合，もっともシンプルなものでも，部品点数で数10個，LSIの内部に置かれたトランジスタに換算すれば数10万個分の複雑なシステムなのです．ソフトウェアにしても，ちょっとした処理でも何千～何万バイトの機械語の集まりであり，それらのすべてが完璧に「正しく」設計されることは，残念ながらあまり期待できません．そこで，試作段階ではこのミスを追求していくデバッグ作業がつきものとなるのです．つまり，**デバッグをいかにうまく行えるか**，がエンジニアの仕事のひとつのポイント(製品の開発スケジュールや品質を左右する要因)であり，**デバッグとは**まさに技術者の腕の見せどころ，あるいは先輩技術者からあらゆる**ノウハウを吸収する格好の機会**ともなります．

また，これも本来の設計・開発作業としては目立たないものですが，システムの完成度を高めるためのいくつかの重要な技術があります．たとえば，信号線や電源ラインに乗ったノイズによって誤動作しないようにする，あるいは他の電子機器に悪影響を与えないように，所定の電磁放射規制条件をクリアする(一定範囲内に抑える)といった設計技術では，実験のための**デー**

タ計測技術も必要となってきます．さらに基板サイズをコンパクト化する実装技術，故障を極力避ける（場合によっては回復する）ための**信頼性技術**なども，一見すると気づかないものですが，プロの仕事としては重要なものなのです．

●試作・まとめ

さて，設計・開発の最終段階を受けて，いよいよ「まとめ」のステップに到達します．ここではまず，ミスを検出する「デバッグ」よりも難しい，正しく動作することを確認するための「**検証**」作業があります．デバッグではエラーを出させてそれを修正していきますが，検証というのは当たり前のはずの動作をチェックするので，ややもすると「完全な検証」の追求が甘くなる可能性があり，デバッグ以上に論理的な作戦を立てる必要があります．

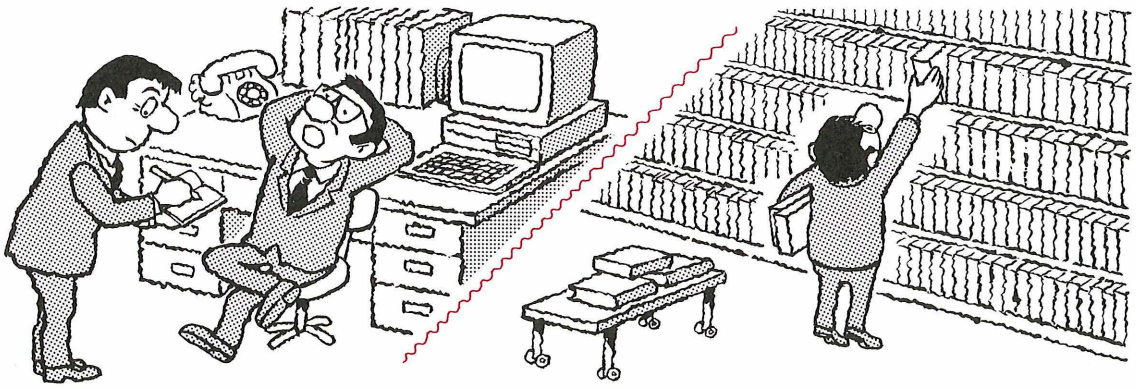
また，開発したシステムがある程度の数量の量産工程にバトンタッチされる場合には，コスト面や量産技術的な検討を加える必要もあります．生産部門への完全な仕様書を用意することや，生産スケジュールの検討・出荷検査仕様書などにも，開発部門がかなり関与します．さらに，ユーザのためのマニュアルの原稿も，小さなプロジェクトの場合には開発スタッフ自身が書くこともあります．この場合，開発の過程でいかにしっかりとメモをまとめておくか，で出来映えも内容も大きく変わってしまいますが，なかなか時間に追われた開発の修羅場では，その余裕がないのが実状のようです．

そして最後に，この「まとめ」の段階で重要なのが，設計・開発の途中で得られた技術・ノウハウを，その後の新たなテーマのための財産として整理することです．たとえば，新たに得られた技術は特許の出願という形で将来に備えたり，技術的なノウハウを研究開発レポートとしてまとめることなどです．また，新規に開発したPLDやASICなどのハード，基板やケースなどの技術資料，CPUソフトなどの「ドキュメント」をしっかりと残しておきます．というのも，開発終了後にバグが出た場合とか，将来モデル・チェンジやバージョンアップを検討するような場合には，開発担当者自身であっても「忘れてしまう」ことがほとんどだからです．

「立つ鳥あとを濁さず」といいますが，この「まとめ」段階をしっかり行う技術者が，長い目で見ると着実に成長するといわれます．終盤のドタバタの中でいつのまにかプロジェクトが終わる，というのがつねですが，若手技術者の皆さんは，ぜひ先輩のお手並みをチェックしてみてください．

このように，「技術者の仕事地図」と簡単にいってもかなりの広がりがあるものです．少しずつ，自分の足元を固めるとともに，ぜひ積極的に自分のフィールドを広げていってほしいと思います．

足を使うことが企画力や技術力を育てる



目標技術のステップアップ

●目標「技術レベル」のいろいろ

前節の「エンジニアの仕事地図」では，マイコン技術者のいろいろな業務のすがたを眺めるなかで，フレッシュマンが身につけていきたい各種の技術フィールドがあることに触れてきました．そして本節では，イントロダクションのもう一つの試みとして，エンジニアとして獲得していきたい技術(の段階)を，全体として別の視点からまとめてみたいと思います．

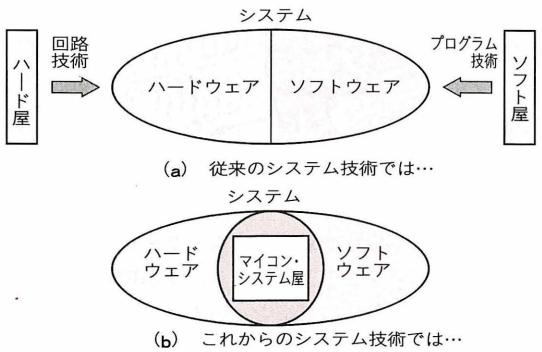
まず最初には，本書の全体にわたって重要なポイントとなっている「トレードオフ」技術のなかで，もっとも基本的な三つのトレードオフについて考えます．そしてつぎに，とくにフレッシュマンが基礎として身に付けていく「ハードウェア技術」，「ソフトウェア技術」のステップアップについて概観して，＜事典＞編の「基礎技術」の解説へとつながっていきます．さらに，プロのエンジニアとなっていくためのポイント技術として，

(1) 開発ツールの活用技術
(2) デバッグ技術
(3) モデリング技術
(4) スケジューリング技術

という四つの技術について，「マイコン技術者のステップアップ」という視点から考えていきます．

もちろん，まだまだ他にも重要な技術はありますが，これらはエンジニアとしての経験からしだいに身に付いてくる(スキルアップしてくる)，総合的な技術力のいろいろな側面なのです．したがって，いきなり全部を理解しようとする必要はありません．フレッシュマンが技術レベルをステップアップしていくにあたって，ポイントとなる「柔軟な視点」を紹介していると理解してください．

〔図0.3〕ハードとソフトのトレードオフ



●ソフトとハードのトレードオフ

それでは，マイコン技術者にとっての重要なポイントとして，筆者の持論である「三つのトレードオフ技術」をあげていきましょう．これは，フレッシュマンの皆さんが今後マイコン・システムについて考えていく上で，ときどき頭に置いて周囲を見なおしてほしい視点，ということになります．キーワードは「トレードオフ」，つまり技術者があらゆる場面で遭遇する，政策的・技術的な判断に関するものです．

第1のトレードオフは，「**ソフトとハードのトレードオフ**」です．**図0.3**のように，エレクトロニクスに関するシステムでは，昔は「回路を設計するハード屋」と「プログラムを作るソフト屋」という，二つの人種がいました．ところが，マイコン・システムの一般化と，システム開発を支援する環境の進歩から，ハードもソフトもカバーするような，いわば「マイコン・システム屋」という技術者が中心となって活躍する時代になりました．つまり，この部分はハード屋の仕事だとか，この問題はソフト屋の領分だ，というような考え方が通用しなくなってきたのです．そこで，システムを設計・開発していくエンジニアの重要なセンスとして，システムの機能のどの部分をハード化し，どの部分をソフトとして実現するか，という判断が大切になってきたのです．このトレードオフはクイズではありませんから，正解が一つとはかぎりません．技術者ごとに，ノウハウのレベルごとに判断が異なりうるので，それだけに「技術者の腕の見せどころ」ともなるのです．

●階層化のトレードオフ

第2のトレードオフは，「階層化のトレードオフ」です．半導体の集積度の進歩，あるいはコンピュータ・パワーの向上によって，現代のマイコン・システムというのは，細部まで眺めていると全体を把握しきれないほど，複雑・巨大なシステムとなっています．そこで必要となるのが，システムをいくつかの階層に分割して考える，という階層化の発想です．**図0.4**や**図0.5**のように，ハードにしてもソフトにしても，ある階層

から下はブラックボックスとして簡略化(部品化)する，という階層化によって，システムがわかりやすくなり，開発効率や信頼性も高くなるのです．ただし，この階層化の判断を誤ると，かえって複雑になって逆効果となったりします．階層の分割は「分割された階層間の適切な情報交換」の設計が必要ですから，ここでもノウハウ的な経験の蓄積が重要になります．

●コスト/スペック/スケジュール

第3のトレードオフは,「コスト/スペック/スケジュールのトレードオフ」です．二つでなく三つの要因がからんでいるために，おそらく世の中の大部分の技術者がこの点で悩んでいる，と筆者も確信をもっているポイントです．

コストとはもちろん費用ですが，これは製造コスト・部品原価だけでなく，開発(イニシャル)コストや人的(ランニング)コスト，あるいは知的所有権の将来的コスト，といった面もあります．もちろん基本的には「安いほどいい」のですが，残り2点とは相反する要因です．

スペックとは「仕様」のことで，システムの機能・能力・性能などと考えることができます．あきらかに，スペックを追求することはコストとスケジュールを圧迫することになります．

そしてスケジュールとは文字通りのものですが，技術者が一人で何もかも担当しているのでなければ,「他との関係の調整」という，なかなかやっかいな要因となります．当然，スケジュールの制限は他の二つの要因を制限してしまいます．このトレードオフをうまく解決できるようになることこそ，ベテラン技術者への登龍門だと思います．

●ハードウェア技術の基礎ステップ

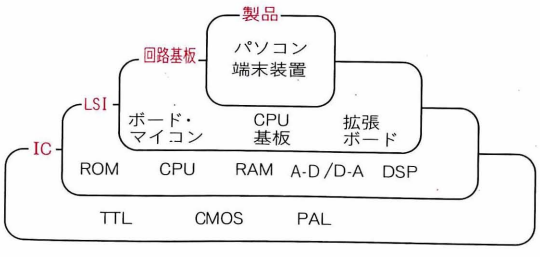
ここでは，マイコン・システムのハードウェア技術について，フレッシュマンがステップアップしていくための指針を考えていくことにします．なお，具体的な技術項目や具体的な課題例については，このあとの<事典>編で詳しく述べていきますから，ここでは「概論」ということになります．まったくの新人でもなければ当然すぎることばかりですが，たとえば「新人研修のときにこのポイントを確認させよう」というような使い方にも役立つものとして，あえて基礎の基礎から列記してあります．

まず最初のステップは,「基礎的な技術情報を身につける」ことです．算数から数学に進むためには九九や加減乗除を知らなければならないように，基本的な用語，各種部品の知識，ハンダ付けの技能(ソフト屋でも必須)，データブックの引き方(「このICはあの本で調べる」という情報)，回路図の約束事，カラー・コードやテスタの使い方，などは最低限身につけたいことです．

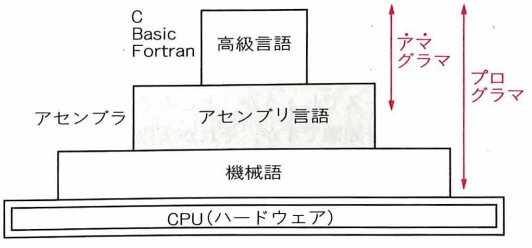
つぎのステップは，たとえば先輩から与えられた実験回路とか，製品の回路図，本に載っている回路図などを「読める」ようになることをめざしましょう．個々の部品の名前を当てるのではなくて，全体としてどこで何をやっているか(おおよその動作)が感覚としてわかる，ということです．これには，周囲の先輩をつかまえて質問攻めにするのが最適の方法です．

そのつぎには，正しく動くかどうかは別として,「自分で回路を設計してみる」ことです．ディジタル回路の実験というのは，間違えてもICから火や煙が出るくらいで，それほど人命に関わるような事故にはなりません．とにかくある機能(時計でもチャイムでも)を

〔図0.4〕ハードウェアの階層



〔図0.5〕プログラム言語の階層



自分の頭で実現しようと努力してみると，欠けている知識を痛感できます．そして，設計した回路を実際に手で作って，テスタやオシロなどを使いながら，動くところまで追求することが大切です．これによって「デバッグ」という技術の意味を理解することができます．

そして，この先にあるのは，もう実戦の世界となります．いきなり本番の仕事を全部任されるというよりは，たとえばチーム・リーダから一部の仕事をアシスタントとして与えられるわけですが，それでも立派なプロの仕事です．自分なりに精一杯工夫して，限られた条件の中で最高の性能をめざしてみましょう．実戦での試行錯誤の繰り返しこそ，スキルアップの最大の教師です．失敗を恐れることはありません．若手の失敗の可能性をちゃんとカバーしてくれているのが，先輩の技量だと思ってしまいましょう．

●ソフトウェア技術

つぎに，ソフトウェア技術のポイントについて考えてみましょう．すでに述べたように，ソフトとハードの両方について「自分のものにする」のがマイコン技術者にとっての基本的な目的ですから，どちらか一方でなく，ともに積極的にアプローチします．

まず最初のステップは，とにかく「コンピュータに慣れる」ことです(ファミコン世代のフレッシュマンにとって，「コンピュータ・アレルギー」の心配は不要でしょうか)．いろいろな理屈とか知識はさておいて，と

[コラム]
マイコン技術者の徒弟制度

「職人」が製品を生み出していた昔，若者は親方のところに弟子入りして，手とり足とりに技術を教えてもらいました．もっとも親方衆も頑固親父が多くて，「師匠の技を盗む」のがふつうでした．デザイン・道具・材料・製作技法などのあらゆるノウハウは，親方から弟子に無形のものとして継承され，この徒弟制度は日本ばかりでなく，たとえばヨーロッパの優れた工芸技術などでもまだまだ基本となっているようです．

さて，ところでエレクトロニクス技術とかコンピュータ技術の世界ではどうでしょうか．テクノロジーの進歩があまりにすさまじく，親方の技術レベルは弟子の世代の技術としては「過去の遺物」になってしまう世界のことです．徒弟制度などという言葉は無縁のものだ，と思われるでしょうか．筆者の感想ですが，なかなかどうして，この世界でも先輩の存在は非常に重要なものだ，とつくづく思うことばかりなのです．マイコン時代の徒弟制度，というのも，けっこう大切な視点のように思います．

●新人の伸びを左右するOJTと上司

まず，誰でも「新人」のときがあり，はじめてこの世界に触れる時期があります．新入社員研修とか，新人研修講座，さらにOJT (On the Job Training)と続くのが一般的なコースでしょうか．ところで，「研修」で習うのはいろいろな知識ですが，それが実際にフレッシュマンの技術として理解され，さらに利用できるように身についてくるのは，明らかにOJT以降のことでしょう．その際に先生となるのはそれぞれの職場の先輩で，まさにこの「師匠」のデキ次第で，その新人の伸びが左右されてしまうのです．

*

筆者の経験でも，新人のときのチームの先輩には大変に感謝しています．些細なこと，同じこと，単純なこと，とんでもないこと，とにかく「臆せず質問する」ウルサイ新人だったあのころ，よく付き合っていただいたものだ，と思い出すだけで恥ずかしくなります．相当に先輩の仕事の邪魔をしていたのだと，やっと分かってくる頃には，やがて自分が新人の面倒を見るようになるのです．筆者にとって，あのころの先輩はまさに「一生の師匠」のように思えます．

*

また，職場の上司も大切な「師匠」でした．技術の現場から次第に離れてマネジメント中心の中間管理職となっても，新しいツールを買ってくれとか，勉強のために学会に行きたいとか，この書籍は面白そうだから参考に買いたいとか，仕事に直結しない要求をいろいろ叶えてくれました．実際のコンピュータ技術としては時代から遅れてしまいがちなため，オフタイムに解説本を一生懸命に読んで話題を理解しようとしている努力の姿勢も，若手への刺激となりました．

*

そして実際に仕事を始めてみると，自分のチームの先輩だけでなく，他のセクションの技術者もまた，いろいろなヒントや知恵を与えてくれました．バグに悩まされ

にかくソフトウェアという概念に慣れてしまうのが，なんといっても先決なのです．「文系」の人であっても，ソフトについてはハンデはありませんから，自信をもってアタックしてください．たとえば，言語はBasicでテーマは「3日で何かゲーム1本を作ること」というと，なんだか遊んでいるみたいですが，これが立派な新人研修になります(実績は保証済みです)．

先輩技術者が特定のプログラミング言語のプロであると，その言語ばかりに限定されてしまうかもしれませんが，できればいろいろな種類の言語，いろいろな種類のソフトウェア環境を(つまみ食いでも)体験したいところです．インタプリタで遊んだらつぎはコンパイラ，そのつぎにはアセンブラ…という貪欲さを期待したいのです．まずは実際にソフトを体験するうちに，プログラミングのテクニックとか，ソフトウェア工学の理論的な勉強，というのも次第に気になってくることでしょう．

ソフトそのものに慣れるには，ハードウェアの心配のいらない(環境として提供される)パソコン上で実験するのがいちばんです．そしてたんなるプログラマのパソコン少年で終わらないためには，拡張スロットのボードで積極的に外界とインターフェースしてみたり，さらにはボード・マイコンという製品を使って，ボード上のCPUプログラムを作る実験に進みます．その先になると，ハードとの両輪になりますが，自作のCPU基板に自分のソフトを走らせる，という目標に向かってトライすることになります．たとえばLEDが1秒ごとに点滅する，というだけの簡単な動作であっても，それがオリジナルのCPUボードでコントロールできたとすれば，かなりのレベルといえます．

そしてプロのマイコン技術者と，アマチュアが一線を画すのが，「厳しい条件をソフトで解決する」，「信頼性の高いソフトウェア」といった点です．求められる機能仕様をハードを追加せずに，割り込みとかマルチタスクとか自作モニタによって，なんとかソフト的に実現したり，あるいはソフト的な誤動作対策の追求に悩まされるといった経験が，結果として新人を大きく成長させるのです．

●開発ツールの活用技術

技術者に「ツール」はつきものですが，マイコン技術者の場合には，仕事がハードからソフトまで，あるいはディジタルからアナログまで(時間スケールでいえばナノ秒オーダからマイクロ秒，ミリ秒を経て秒，分，時間のオーダまで)の領域にわたっていますから，関連するツールも多種多様です．ステップアップの指標としては，「**各種ツールを理解して，適切に活用できる**」という状態が明確な目標となります．

昔の「街の電気屋さん」というのは，テスタとハンダごてがあれば，たいていの家電製品を修理できたのですが，これは完全に過去の話となりました．現在では大手電気メーカのサービス・センタであっても，修理とは「CPUボード全体の交換」以外に方法がない，という状態になっています．テスタどころかシンクロスコープがあっても，動作がディジタルであれば見当がつかないわけです．各種の計測器を含めて，高価な

ている本人が気付かない単純なミスを，通りかかって一瞥するだけで指摘して立ち去ってしまう先輩が神様のように思えたり，ノイズ対策で悩んでいると，アナログに強い古参の先輩が画期的な解決策を授けてくれたり，といった経験は数多くありました．教科書に書かれていないこれらのノウハウを先輩から吸収すること，これはまさに徒弟制度そのもののプラス面だったと思います．

●「態度で示す」のが最高の教育

そして，新人もやがて中堅技術者に，さらに新人を教育する立場の「先輩」になっていきます．とても自分にはあの先輩のような立派な教育はできそうもない，とは思いながら，それでもメニューを作って実習させて，質問があれば仕事の手を休めてなるべく答え，そして何より，「態度で示す」雰囲気を自分にいい聞かせることに努める日々となるのです．先輩である自分自身が，本当に心からエンジニア生活をエンジョイしていれば，それが最大の教育であるように思います．技術者として自分を向上させることの喜び，新しい技術を自分のモノにして，さらにそれを具体的に世に出すことの面白さと手応え，そういう楽しさを，なるべく言葉に出さずに伝えたい，と思うようになりました．

*

技術者というと，エンジニアとしての自分の腕次第とか，実力の勝負というクールな面ばかりが強調されてしまいます．しかし，エンジニアとしての自分が現在あるのは，多くの先輩(師匠)のお陰なのだ，と理解するところまでは，ようやく分かってきたところです．会社との関係はある意味でギブ・アンド・テイクですが，先輩からはテイクばっかりの日々でした．このノウハウなり喜びを若手に惜しみなくギブしていくことで，やっと先輩への感謝をはたせるのだろう，と思っています．

開発ツールをいろいろ使うことは，プロの技術者のスタート・ラインとなります．

ハードウェアでは，**テスタ**で直流的(静的)に計測できるのは「ショートのチェック」程度で，**オシロ**や**シンクロ**，または**ロジック・アナライザ**によるディジタルの計測は常識となります．扱う信号によっては，**ストレージ・スコープ**とか**FFT アナライザ**といった装置も必要になりますが，機器の使い方の背景にある，信号そのものの物理的な理論を理解することも重要です．

CPU 関係ツールの本命は**ICE**(イン・サーキット・エミュレータ)，通信関係のツールならば**プロトコル・アナライザ**，ということになるのですが，場合によっては「使い慣れた CPU ボードを使って，必要な計測装置を作ってしまう」という姿勢が重要です．

ソフトウェア開発のツールは，なんといってもパソコンの機動力です．ソフト開発の場合には，パソコン→ ICE というレールが決まっていますが，同じ開発環境を使っても，ベテランと新人とではツールの活用度が違いますから，フレッシュマンは先輩の「技」をよくチェックしましょう．また，ハードのツールを自作することが重要な経験となるように，いつも ICE に頼るばかりでなくて，対象の程度によってはあえて ICE を使わずに，**デバッグ・モニタ**を自作してシステムに内蔵し，その環境でソフト開発を行ってみる，というような挑戦もいいことです．

●デバッグ技術

デバッグとは，システムのバグ(問題点)を追求し解決する，地味な作業です．当然のことですが，最初から原因の明らかなバグなどありません(わかっていれば対策する！)から，フレッシュマンにとっては困惑するばかりかもしれません．正しく設計して正しく試作したはずの基板が，どうして動かないのか，という気持ちです．

しかし，ちょっと慣れてくると，「**バグは必ずあるもの**」という真理を納得するようになります．中堅エンジニアともなると，試作基板にいきなり電源を入れたりはしませんが，その背景にはこんな悟りの境地があるのです．そして，いざバグに遭遇したら，過去の経験を総動員して，あらゆる視点から原因の可能性を追求します．適切なツールを活用したり，デバッグ用のハードやデバッグ用のルーチンを，(場合によっては設計の当初から)わざわざ作ることで「急がば回れ」を実践します．

実際の対象によって千差万別なため，なかなか図式的な「デバッグ技術」という教科書がない(工学的な理論は議論されている)のですが，ある意味でデバッグというのは，技術者の能力をフルに発揮する場です．ひとことでいえば「**経験とともに成長するのがデバッグ技術**」ということになります．筆者の場合は，バグやトラブルに直面すると，自分の力を試すいいチャンスだと思って楽しんでしまうこともあります．

●モデリング技術

マイコン・システムを設計・開発する際に，情報の解釈・データの形式・ソフトの動作などに，なんらかの「数学的」，「物理的」モデルを使うことがけっこうあります．たとえば，地上の重力の影響をプログラムで考慮するとか，2 次元の平面ディスプレイで 3 次元的な立体図形を表示するための座標変換処理，最近の流行ではバーチャル・リアリティ(仮想現実感)中の「世界モデル」などです．

この「モデリング技術」のようなテクニックは，マイコン・システムのハードやソフトの技術の獲得とはすこし異なるものです．つまり，自然界とか人間の感覚をシステム上でいかにモデリングするか，という技術は，着目するアイデアの部分は本人次第ですが，正しく表現するためには，しっかりとした理論的手法にしたがう必要があるのです．たいていのモデルは高校の数学程度の範囲で記述できるもので，筆者もよく物理や数学の教科書を開いて調べたことがあります．

モデリング技術を向上させるためには，日頃から周囲の現象を「マイコン・システムで表現するにはどうしたらいいか」といった視点で眺める，という姿勢が大切です．また，モデリングの実例を，いろいろな文献や論文で調べたり，実際にパソコンのソフトなどで実験してみることも有効です．

モデリングを考えているうちに，高校の授業で習った物理法則や数学公式の意味があらためて理解できた，などという経験はけっこう楽しいものです．

ここでモデリングのコツというか，ヒントを指摘しておくとすれば，自然界は「線形性」に支配されている，という物理学の大原則が大切な視点です．線形性，つまり比例関係によって力学的な微分方程式を記述すると，そこから自然に「重力加速度」，「単振動」，「時

定数の指数関数的特性(→共振回路の減衰振動)」などが出てくるわけで，これらの関数というのはけして数学のための作為的なものではないのです．興味のある人は，ちょっと考えてみてください．

●スケジューリング技術

トレードオフの話題のところでも述べましたが，マイコン技術とはちょっと異質のポイントとして，「**開発作業の進め方**」に関する，スケジュールとかチームプレイ，といった視点があります．スケジューリング技術，あるいは進んでいくと「**プロジェクト管理**」の技術として重要になるノウハウです．この章は「概論」ということで，これ以上の深入りはしませんが，スキルアップの後半戦では主役となる技術なので，頭の片隅に置いてみてください．職場の上司がたんに「管理職(マネジメント)」として技術の現場から離れているようでも，じつはこのスケジューリング技術を駆使している，その裏には，しっかりとした技術が存在していることに気づくかもしれません(p.127 の**コラム**参照)．

●「技術者の創造性」とは

さて，ここまでの技術というのは，どちらかというと「与えられる」スキルアップについてのものでした．ところでマイコン技術者は，世の中に新しいシステムを送り出すのですから，そこにはオリジナリティ，つまり「**独創性**」，「**創造性**」が求められていくことになります．まったく同じテーマを与えられても，エンジニアごとに生み出すシステムは異なりますし，同じ技術者でも成長していくにつれて，以前とは違う(より美しい)システムを思いつくはずです．

この技術者の創造性というのは，いつまでも尽きない課題なのですが，残念ながら筆者もここで系統的に意見を述べることはできません．まだまだ現在進行形で模索しているところで，これからも皆さんと共有する「宿題」ということになるようです．フレッシュマンがスキルアップを考えるときには，最後にはこの創造性の勝負となる，つまり先輩から「与えられる」技術でなく，自分自身の技術としてどこかで一人立ちしていくのだ，ということをしっかり心がけていってほしいと思います．

エレクトロニクスとかコンピュータといえば，ハイテクが際限なく進んでいる世界のようですが，逆にいえば**自分の技術を自由に展開できる空間が広がっている**，という楽しみのある世界なのです．自分の可能性を信じて，少しずつ着実に，そしてときには大胆にチャレンジしていきましょう．

[飛び石コラム] おすすめBOOKS ①

●心の社会
M. ミンスキー　著　　安西祐一郎　訳
産業図書

人工知能・認知科学の分野で世界中に翻訳され，古典的名著になろうとしている本です．人間の脳の中での働きについて，非常に多数の処理単位(エージェント)が別個に，かつ共同して複雑な仕事を実現している，というような考え方で，現在のニューラル・ネットワークとか，超並列コンピュータを考える際のヒントを与えてくれます．

●超マシン誕生 ── コンピュータ野郎たちの540日
トレイシー・キダー　著　　風間禎三郎　訳
ダイヤモンド社

コンピュータ黎明期の，夢とパワーあふれる若者たちのチャレンジ物語として，一気に読ませてしまう本です．内容はデータゼネラル社のスーパミニコン開発という，ちょっと前の時代ですが，エンジニア・スピリットを見事に伝えています．日本のこの手の「成功物語」が，なぜか企業の提灯持ち記事になる場合が多いのとは対照的です．

● aha！ ーひらめき思考(別冊サイエンス)
マーチン・ガードナー　著　　島田一男　訳
日本経済新聞社

世の中にパズルの本は数多くありますが，そんな中でもとびきりのパズル集として楽しめ，さらにコンピュータ技術者の「発想法トレーニング」としても重宝できる本です．著者が指摘するように，コンピュータ時代になると，問題を「計算して解く」部分は機械に任せて，人間の思考(ひらめき)がより重要になってくるのです．

[Appendix] ある「技術勉強会」のメニュー

以下のメモは，ある中堅エンジニアが何年か前に，若手技術者を対象に開催した技術「勉強会」での，「概論」の講演のための資料です．

いわゆる「研修会」というのは，上司から指示されて「仕事」として参加するものですが，この勉強会では，本人の自発的な参加希望を条件としました．そのため，ただ受け身で聞くばかりでなく，全員に事前に課題を与えて，毎回の冒頭に交代で発表させ，さらに最終レポート(または新規の特許出願)の提出を全員に義務づけました．具体的な実務に関したテーマでは，講師として中堅の技術者がそれぞれ各技術分野を深く解説しましたが，これは「講師」自身のプレゼンテーション研修でもありました．

ここに紹介するのは，冒頭3回分の一般的「概論」の一部(業界の具体的な事項に関する部分は＜省略＞扱い)ですが，エンジニアの姿勢，ハードウェアの概要，ソフトウェアの概要などを広く展望しています．

----------[勉強会・総論]----------

(1) イントロダクション

A．この勉強会の目的について

・若手の実力アップが求められている
　当社の開発部門の歴史：結果を待ちきれない体質
　年寄りの発想とは技術レベルが別になりつつある
・「講習会」でなく「勉強会」であることの重要性
　受け身では成長しない
　自分を変えられるか？
　自分の意見を持ち・主張すること
・たんなる専門馬鹿ではダメになる
　ハード屋もソフト屋も長続きしない
　　(技術とは進歩するもの：個々の人間の進歩のほうが遅い)
　当社の客観的な技術レベルは低いという認識
　当社は社員を成長させるのは下手→自分でやる！
　　(アメリカ：「引き抜き」されない技術者に問題あり)
・一点豪華主義のプロとなるためのネタを捜す
　研究者・学者を越えてもよい
　「この分野ならおまかせ」は武器となる
　趣味嗜好の多様化時代：自分の趣味に仕事を引き込む
・特許を書くのは技術者の仕事
　ワープロの活用：活字・コピー・書き直し
　論理思考のトレーニング
　アイデアだけでよい
　出願補助金は給料が安い技術職の合法的アルバイト
　レポート・報告書・図面も手書きの時代ではない

B．「いまやってみること----いくらでもある！」という話

・コンピュータを知らないで平気な技術者は去るべし
　パソコン・アレルギーは論外
　Basic でゲームを作る
　C に触ってみる
　Prolog/Lisp/Smalltalk とか･･･
　アセンブラでツールを作る
　パソコン・EWS などの一般情報を知っているか
　ミニコン・Unix
・日常業務の間隙を活用する
　実験室は部品の宝庫：こつこつと電子工作
　ソフトが走る合間の読書：会社の書籍をフル活用する
　　(定期購読誌・バックナンバ・書庫の文献)
　暇を見つけてゲーム・ソフトを作る
　こつこつとオリジナル装置を製作する
　他社製品の分解・解析
　メーカである会社の名前は案外に重い→活用するべき
　　(資料請求・サンプル評価)
・会社は向上への投資を惜しまない
　(日々向上しないのは技術者が無能である証明)
　英会話の能力があればバリバリ海外へ行ける
　アンテナを立てて学会・展示会・ショーなどへ出張
　勉強のための書籍は会社に買わせる
　去年と同じ仕事をしているのは能がない：環境の改善を考える
　　(必要性を証明して最新のツールを導入できないか)
　大学との共同研究のネタがないか
　フェア・市場調査に行きまくる
　特別な行動には十分な報告・レポートが条件→やりがいあり！

(2) ＜省略＞

(3) [コンピュータ]側からのアプローチ

G．コンピュータとは何なのか

・コンピュータの歴史：本を読めばわかるので省略
　ノイマン博士・ENIAC

PDP・VAX・Sun・NEWS・NeXT
アップル・IBM PC・Mac・ATARI・NES
PC80/98・MSX・ファミコン・TOWNS・AX
・コンピュータ技術：ハードとソフトの両面からなる
デバイス：真空管・TR・IC・LSI・DSP
大型機・ミニコン・TSS・パソコン・EWS・LAN
ノイマン方式・バッチ・TSS・OS
Fortran・Basic・C・Lisp
構造化・マルチタスク・ウィンドウ・ネットワーク

H．＜省略＞

I．コンピュータ科学・コンピュータ技術の進歩

・パーソナル化・マシンパワーの向上
パソコン→機能強化→かつてのミニコンを凌駕
ソフトの一部をハード化する
(DSP・FFT・ニューロ・ファジイ)
CPU の機能向上(CISC vs RISC)
・通信・ネットワーク化
LAN・ISDN・LA の分散処理化
情報の国際ネットワーク化
・マルチメディア化・マン-マシン・インターフェースの向上
AV・双方向・ウィンドウ・マウス・トラックボール
脳波バンドによる入力装置
・知識工学・人間工学のはてにあるものは何か
コンピュータ技術も結局は人間に回帰する

(4) ＜省略＞

－－－－－－[勉強会・ハードウェア概論] －－－－－－

(1) 当業界の回路方式の歴史＝電子回路技術の歴史
＜省略＞

(2) ハードウェア技術のおさらい

F．電子回路の特徴とポイント

・受動素子と能動素子
抵抗/コンデンサ/コイル
ダイオード/TR
AC/電池/水晶/LED/LCD
・アナログ回路
オペアンプ/コンパレータ
パッシブ・フィルタ/アクティブ・フィルタ
アナログ乗算器
・ディジタル回路
TTL/CMOS/HS-CMOS
ロジック IC/順序回路→乗算器を設計してみる
ROM/RAM/PAL/LCA
・ CPU 回路
ソフトウェアで回路を実現する
・アナログ・ディジタルの接点領域
S&H　A-D　D-A

G．さらにプロならば必要な視点

・システム設計技術の三つのポイント
ハードウェアとソフトウェアのトレードオフ
階層化の思想
時間スケールに関する視点
・ノイズ/EMC
停電/電源電圧低下
瞬間停電
AC ライン・ノイズ
静電気
誘導高周波信号
電磁放射
用語：EMC/FCC/マージン
「ノイズを出さないシステムは外からのノイズにも強い」
・信頼性設計
フェイルセーフ
ウォッチドッグ・タイマ
フォールトトレラント・システム
誤動作対策
(ハード・マージン向上/ソフト対策)
修復/対策技術
・テスタビリティ/デバッガビリティ
3種類のバグをいかに検出して退治するか
(仕様の不備/開発途上のミス/本当のバグ)
ハードウェアのバグ検出テクニック
(測定器/視点/ソフトがらみ)
デバッグ環境を作り込んでおくのがプロの技
(テスト・ピン/テスト・モード/テスト治具)
・開発のドキュメンテーション技術
企画検討
機能仕様書
基本システム構想
回路設計
ソフト開発
部品表/原価表
テスト/デバッグ記録
特許調査
詳細仕様書
ユーザーズ・マニュアル
サービス・マニュアル
開発レポート
・コスト/スケジュール/特許
仕様 vs コスト
仕様 vs 開発スケジュール(vs コスト)
仕様 vs 特許対策→方式変更もありうる
「全体として最適な方法は何か」という視点
スケジュールは与えられるものではない

(3) 具体的な回路の設計方法

H．アナログ方式

＜省略＞

I．ハイブリッド方式

＜省略＞

J．ディジタル方式

＜省略＞

K．システムとしての視点

＜省略＞

(4) 専用 LSI の設計方法

L．LSI 設計の基礎知識

・ディジタル時分割方式
　タイムスライス
　状態保持レジスタ群
　タイミング設計
・マイクロプログラム方式
　DSP
　処理制御信号の作り方
　マイクロプログラム・メモリ
・LSI の分類
　フルカスタム LSI
　セミカスタム LSI
　　(ゲートアレイ/スタンダードセル)
　ユーザ・プログラマブル LSI
　　(LCA/大規模 PLD)
　マクロセル/メガセルの登場
　DSP セル/コア CPU の登場：システム・オンチップ
　重要な視点：開発費用/開発期間/開発手法

M．LSI の設計風景

・いにしえの設計手法
　　[図面/タイミング・チャート] インターフェース
　儲かるのは→ NTT と宅急便
　代理店のデザイン・サポート部隊
・回路設計
　システム設計/ブロック分割
　タイミング設計
　CAD ツール
　データ変換/転送
　重要ポイント：テストを意識した回路設計
・テスト・プログラム設計
　テスト・プログラムの意味
　　(論理検証/タイミング検証/品質保証)
　テスト・プログラムの諸形式
　　(Wave/Vector/Graphic)
　データ変換/転送
・シミュレーション
　LSI シミュレーションの種類
　　(論理 Sim/仮 Sim/実 Sim)
　LSI シミュレーションの意味
　　(ブレッドボード(BB)の代行/回路の検証/テストの検証)

N．これからの専用システム開発について

・シリコン・コンパイラ化
　LSI 開発環境の統一→製造メーカを自由に選べる
　　(デザインハウス/シリコン・ファウンドリの分化)
　仕様記述のみで LSI 化：シリコン・コンパイラ
・評価システム/ソフト開発システム
　ES 評価システムの必要性
　コア CPU：ソフトの同時開発環境が重要
　BB の位置づけ：LCA 化がカギ？
・統合シミュレーション・システム
　今後のツールは EWS！
　仕様/条件の高級言語による記述
　ハード/ソフトの統合シミュレータ
　基板 CAD+ 3 次元メカ CAD →自動試作システム
　CASE ツールとの統合

－－－－－－[勉強会・ソフトウェア概論] －－－－－－

(1) コンピュータ/CPU のおさらい

A．コンピュータの種類と歴史

・メカ式コンピュータ
　歯車を組み合わせた計算器
　コンピュータは戦争とともに発展(弾道計算)
・アナログ・コンピュータ
　乗算→アナログに向いている(自然界は線形)
　非線形素子→指数/対数演算器
・ワイヤード・ロジック方式コンピュータ
　膨大な数のリレーによってディジタルを実現
　速度/信頼性が問題
・ノイマン方式コンピュータ
　ストアード・プログラム方式
　真空管→ TR → IC → LSI → VLSI → ULSI
　記憶装置の階層化：キャッシュ/バッファ/連想記憶
・並列コンピュータ
　CPU を複数並べる：1 次元/2 次元/3 次元
　並列処理のための記述言語
　CPU レベル→トランスピュータ
・データフロー方式コンピュータ
　非ノイマン型コンピュータ
　データ駆動方式とは
　NEC 画像処理用プロセッサ
　カシオ事務計算用コンピュータ
・ニューロ・コンピュータ
　コンピュータと人間の脳
　ニューロの実現方法：ソフト/ハード/バイオ

・ファジィ・コンピュータ
　ファジィの実現方法
　　(ソフト/メモリ/ハード/バイオ)
・光コンピュータ
　光ファイバの能力
　光 LSI による光コンピュータ(ニューロも含む)
・汎用コンピュータ
　IBM の大型機のような「メインフレーム」
　CPU の能力は巨大→ TSS
　階層性/互換性/信頼性
　　(走行を止めずに保守/増設)
・スーパ・コンピュータ
　科学技術計算/シミュレーション
　ベクトル演算/高精度浮動小数点演算
　専用の言語による最適化
・ミニコンピュータ
　DEC の PDP → VAX
　LA/FA から CAD/CAM 分野
　EWS に押されて消滅の危機
・パーソナル・コンピュータ
　8 ビット/16 ビット/32 ビット
　本当にパーソナルに普及する仕様とは？
・ワークステーション
　32 ビット/64 ビット CPU/RISC の活用
　ミニコンの複数端末→ 1 人 1 台の時代！
　CAD/ソフト開発/CG
　Sun/NEWS/VAX/各社
・オフィス・コンピュータ
　IBM は本当に使いやすいのか？
　「オフコン」→パソコン OA
　ネットワーク化：個々の端末を LAN で結ぶ
・ファクトリ・コンピュータ
　FA とは(NC → FA)
　信頼性の条件
・ラボラトリ・コンピュータ
　実験装置の自動制御
　データ収集/処理
・ゲーム・コンピュータ
　ゲームウォッチ/LCD ゲーム
　ぴゅう太/セガ/ファミコン/アタリ
　メガ・ドライブ/PC エンジン/ゲームボーイ
　PC8801/X68000/FM タウンズ

B．おもな CPU の歴史

・電卓用専用回路→ CPU の発想
　最初の CPU はなんと日本人が作った！
・インテル：8080/8085
　数値計算/データ処理向け
　あまり美しくないアーキテクチャ
　市場に先に参入してシェアを押さえる
・モトローラ：6800/6809
　制御系に向く
　エンジニア受けする美しいアーキテクチャ
　6809 の OS-9 は 8 ビットなのにマルチタスク
　後発に泣いた：モトローラの体質
　「ビッグ・マイナー」CPU
・ザイログ：Z80
　8080 のソフト上位互換/改良版
　8 ビットの帝王として君臨
　CP/M がディスク時代とマッチした
　「リトル・メジャー」CPU
　周辺 LSI ファミリは不振
・ロックウェル：6502
　アップル・コンピュータのオープン・アーキテクチャにのる
　ファミコンの心臓部
　小さいチップ/簡潔な命令セット/高パフォーマンス
　プリフェッチ機能/サイクル・スチール機能
・ビットスライス：2900
　コンピュータ機能を複数チップで最適に実現
　高速用途/専用機能向き
　高機能 CPU/ASIC に押されてイマイチ
・国内メーカの戦略と栄枯盛衰
　日電　日立　東芝　富士通　松下
　三菱　沖　三洋　シャープ　ソニー
　エプソン　リコー　ローム
・1 チップ・マイコン群
　4 ビット CPU：日本の得意分野
　　(機器組み込み/スレーブ CPU)
　8 ビット CPU：1 チップの主戦場
　　(6801/Z8/8049/8051)
　　(7801/6301/50740)
　　(16 ビット処理コア：日電/富士通/東芝/日立)
　16 ビット CPU：いよいよ本格化
　　(V25/68008/8096)
・16 ビット化/32 ビット化/64 ビット化
　インテル：8086/286/386/486
　モトローラ：68000/68020/68030
　NEC：独自の V シリーズ：V60/V70/V80
　国内各社：TRON チップで付和雷同
・高級言語対応：CISC vs RISC
　CISC の限界
　RISC の問題点
　SPARC/R3000/88000/80860
　日本のメーカ：独自にやらない(やれない)
・ASIC/コア CPU
　システム・オンチップの発想
　1 チップ・マイコン上にゲートアレイを
　スタンダードセル上に CPU コアを
　東芝の失敗例：スーパ・インテグレーション

リコーの成功例：ファミコン専用チップ
エプソンのアプローチ
モトローラの情けない対応例
国内大手電気メーカの重い腰

C．CPU の使われ方の例

・データの処理/計算/保管：電子手帳
2 進化 10 進数データ/10 進数命令
ストリング・サーチ命令/ブロック転送命令
・自動制御：FA
割り込み命令/タイマ割り込み/ポーリング
・センシング技術：自動車
センサのいろいろ/A-D コンバータ
アンチロック・ブレーキの例：8096
・情報の出力手段：LCD パネル
LED ダイナミック点灯/7 セグメント LED
LCD パネルの構成
・通信：FAX
イメージ・センサ/画像データ処理の特性
電話回線との接続/プロトコル/ネットワーク・リンク
・ネットワーク：SCSI/イーサネット
パソコンの外部信号バス
(GPIB/RS-232-C/SCSI)
EWS/ミニコンの LAN
・パソコン vs 機能特化製品群
ワープロ/電子手帳
多機能電話
ファミコン
・家電機器への組み込み
センサ入力に対する処理(エアコン/炊飯器/ガス警報器)
時間管理処理(タイマ・セット/周期的動作)
情報出力処理(メッセージを話す家電製品)

(2) ソフトウェアとは

D．ソフトウェアの種類と基本概念

・ハードウェア/ファームウェア/ソフトウェア
ファームウェアとは
組み込み機器のプログラムはファームウェア
・「ソフトが走る」とはどういうことか？
CPU が走る
プログラムが走る
ソフトとは無限ループのこと
・階層化→システム・ソフトウェア
階層化の必要性/システム・ソフトウェアとは
ブラックボックスとのインターフェース
・OS/DOS/BIOS
OS とは/DOS とは
BIOS とは
DOS コール/BIOS コール
・プログラム/タスク
プログラムとは/タスクとは
・ユーティリティ/ツール/ファイル
ユーティリティとは
ハードのツール/ソフトのツール
ソフトは「ファイル」という概念から
・モジュール/ルーチン/ライブラリ
ソフトのモジュール化
メイン・ルーチン/サブルーチン
ソフトのライブラリ化/ライブラリアン
・アルゴリズム/チューリング・マシンとは
フローチャート/アルゴリズム
コンピュータ理論の世界
・マルチタスク・システム
マルチタスクとは
当業界はマルチタスクの好例
・バッチ/TSS/リアルタイム
マルチユーザとは
バッチ・システム/TSS システム
コンピュータのリアルタイム性

E．OS のいろいろ

・OS/DOS の必要性
OS の歴史
CPU システムでの DOS の条件
・Unix
ミニコンの世界の標準
ファイル/ディレクトリの階層性
文字形式ファイルによる統一
2 派の対立と統合への展望
・DISC-Basic
DOS を意識させない DOS
Basic からの制御の限界
・CP/M
Z80 をメジャーにした DOS
ツールのいろいろ
・MS-DOS
パソコンの世界を制覇した DOS
CP/M がベースの 8086 用 DOS
途中のバージョンから Unix 風ファイル管理をサポート
MS-DOS の問題点
・OS-9
究極の 8 ビット CPU：6809 用の究極の OS
マルチタスク/オープン・システム
OS-9/68000 の人気
・OS/2
IBM の他社いじめ戦略：マイクロチャネル
80286 の問題点

・Windows
　マッキントッシュの先進性：マン-マシン・インターフェース
　オブジェクト指向の環境とは：各種ウィンドウ・システム
・TRON
　日の丸 OS＝TRON の目新しさ(無い)
　TRON の明日/CEC との関係
・ファミコン/Macintosh/FM タウンズ
　ファミコン：クイック・ディスク用 DOS
　FM タウンズ：CD-ROM だけが頼り

F．プログラム言語のいろいろ

・プログラム言語の分類
　インタプリタ言語/アセンブラ言語/コンパイラ言語
　クロス・アセンブラ/クロス・コンパイラ
　AI 言語
・機械語
　CPU が読むのは機械語
　インテル HEX フォーマット
・ASSEMBLER
　CPU ソフト開発の主流
　ニモニック記号
　マクロ・アセンブラ
・Fortran
　初代汎用言語
・Cobol
　事務/経理関係のプログラム言語
・PL/I
　事務/経理関係のプログラム言語
・Pascal
　計算用言語：(C に押され気味？)
・Basic
　ビギナー向け/インタプリタのメリット
　Basic コンパイラ
・FORTH
　制御向け言語/非常にコンパクト
・C
　Unix を記述/各種入り乱れて激戦
　C の問題点/欠点
・Lisp
　リスト処理向け言語
・Prolog
　AI 言語？
　知識表現/知識処理に向く
・Ada
　アメリカの軍用規格言語(ブームはほんの一時だけ)
・Smalltalk
　言語というより環境
　オブジェクト指向/ウィンドウ
・C＋＋
　C の欠点に対応して拡張

G．ソフトウェアの走行環境

・バッチ/TSS
　結果が欲しいような処理
　負荷に依存する処理速度：LSI 設計の実例
・パソコン
　通常のシステム：DOS からアプリケーションを呼ぶ
　コピー・プロテクト：独自 OS
　パソコンの暴走→プロテクト・モード/スーパバイザ・モード
・組み込み機器
　ファームウェアの走行環境
　リセットから CPU が ROM プログラムに従う
・エミュレーション
　機種の互換性とは
　ハードウェア互換性/ソフトウェア互換性
　エミュレーション・モード
　AX パソコンの失敗
・リモート・プロセシング
　CPU システムとの通信
　リモート・デバッガの例
・オンライン
　銀行のオンライン処理の例
　オンラインの注意点：信頼性/回復対策
・分散処理
　EWS/パソコンの LAN
　リアルタイム処理の分散化は可能か？→ NTT の例

H．プロのためのソフトウェア技術

・階層化/モジュール化：「ソフト部品」
　ソフト部品のメリット/注意点
・信頼性
　暴走しないシステム
　周囲を暴走させないシステム
　暴走からの復帰方法
　CPU が自分の発狂を知るには
　フォールトトレラント・システム
・フレキシビリティ
　製品の仕様変更に対応できるソフトとは
　多機種展開に対応したソフト開発
　組み込み機器の制御ソフトなんてみな同じ？
・保守性
　製品の寿命/何年もソフトを覚えていられるか
　ドキュメント化によるサポート
・ソフトウェアのライフサイクル
　企画フェーズ/システム設計フェーズ
　開発フェーズ/デバッグ/ROM 化
　市場デバッグ/バージョンアップ

・ソフトのコストとは
　製品コストとソフトの比率
　外注の危険性
　　(ノウハウのレベル/トンズラハウス/安易な変更)
・バグとは
　仕様の本質的なバグ
　プログラム上のミスによるバグ
　　(アルゴリズム/ミスタッチ/レジスタ共有)
　ソフトのインターフェースのバグ
　非常に頻度の低いバグ
　致命的バグ/許せるバグ
・デバッグとは
　デバッグ曲線
　あからさまなバグを検出するデバッグ
　想像を越えるバグを出現させるデバッグ
　市場デバッグ：最大のデバッグ
・ソフトウェア開発手法→ CASE
　効率的なソフト開発環境とは
　専用言語を作ってしまう
　専用コンパイラを作ってしまう
　CASE ツールとは
　アセンブラ＜高級言語＜自然言語＝仕様書
　CASE＋フル CAD システムの美しい世界

(3) CPU ソフトを走らせる

I．CPU の動作原理/アーキテクチャ

・ノイマン方式コンピュータとは
　CPU に適したシステム
　CPU の速度とメモリの速度と I/O の速度
・ CPU のアーキテクチャとは
　Z80/6809/V25 のアーキテクチャの例
・メモリ：ROM/RAM
　メモリのアクセス・タイム/CPU のウェイト信号
　SRAM のバックアップ
　DRAM のリフレッシュ
　疑似 SRAM のインターフェース
・バス：アドレス・バス/データ・バス
　バス・ラインの考え方
　3 ステート/ハイ・インピーダンス
　データ・バス/アドレス・バス/コントロール・バス
・システム・クロック
　CPU のクロックと命令サイクルの関係
　クロック発生回路
・プログラム・カウンタ
　命令ポインタ/インクリメント
・アキュムレータ
　ALU/演算処理
・レジスタ
　インテル系：多レジスタ・アーキテクチャ
　モトローラ系：ゼロページ・メモリ・アーキテクチャ
・アドレシング・モード
　インプライド・アドレシング
　レジスタ・アドレシング
　イミディエート・アドレシング
　ダイレクト・アドレシング
　レラティブ・アドレシング
　ゼロページ・アドレシング
　インデックス・アドレシング
　レジスタ・インダイレクト・アドレシング
　メモリ・インダイレクト・アドレシング
　レジスタ・インデックス・アドレシング
　インデックスト・インダイレクト・アドレシング
　インダイレクト・レジスタ・インデックスト・アドレシング
　エクステンデッド・アドレシング
・命令：オペコード/オペランド
　オペコードの例/オペランドの例
・フラグ/分岐
　フラグ・レジスタ
　フラグ・ビットによる分岐命令
・リセット/ベクトル・アドレス
　CPU をリセットすると･･･
　リセット・ベクタ方式 CPU
・割り込み(ハード/ソフト)
　ハードウェア割り込み
　ソフトウェア割り込み
・スタック/スタック・ポインタ
　スタックとは/スタック・ポインタとは
　FILO/FIFO：リング・バッファ
・入出力動作とは
　入出力命令
　I/O マップド I/O
　メモリ・マップド I/O
・パイプライン/プリフェッチ/キャッシュ
　CPU の高速化の要請
　命令パイプライン
　命令のプリフェッチ
　キャッシュ・メモリの搭載

J．周辺回路/周辺 LSI

・バス・バッファ
　245：双方向 3 ステート・バッファ
　374：3 ステート・ラッチ・バッファ
・デコーダ
　アドレス・デコードとは/負論理のチップ・セレクト信号
　138：3+3 入力 8 出力
　139：(2+1 入力 4 出力)2 系列
　PAL によるアドレス・デコード
　ゴースト(シャドウ)エリア

・タイマ
　CPU のソフトによる時間管理はデタラメ
　8253：古典的タイマ
・パラレル I/O
　データ・バスから信号を出力するには
　データ・バスに信号を取り込むには
　8255：古典的 PIA
　瞬間的バス・ファイトの問題
・シリアル I/O
　シリアル通信の種類
　シリアル信号を CPU のソフトで実現すると
　8251：古典的 UART → RS-232-C
・ MMU
　メモリ空間を拡張するには
　バンク・レジスタ
　ページング/セグメント方式
・割り込みコントローラ
　多重割り込みの必要性/プライオリティ
　Z80 ファミリ：デイジ・チェーン
　モトローラ系：ベクタ処理が簡単
　8259：プライオリティ可変コントローラ
・ DMAC
　CPU による単純転送の非効率さ
　DMA とは/DMAC の仕事
　DMA の欠点→分散処理へ
・ FDC
　FDD の制御処理
　765/8877：古典的 FDC
・ A-D・D-A
　S&H
　変換時間の考え方
　必要なビット精度/変換時間を見極めること
・ 1 チップ・マイコン
　マルチチップ CPU/1 チップ CPU
　チップ上のマスク ROM の使い方
・リセット IC
　リセット IC の機能/使い方
　　(パワーオン/電圧低下/レギュレータ)
・ウォッチドッグ・タイマ IC
　ウォッチドッグ・タイマとは
　ハードによる構成/専用 IC による構成

K．CPU プログラムの開発環境と開発手順

・機能仕様→システム検討
　トップダウン設計/ボトムアップ設計
　ニーズ指向/シーズ指向
・モジュール分割→フローチャート
　モジュールの切り分け方/モジュール結合
　モジュール間のインターフェース
・開発ツール
　オールインワン型/パソコン・ホスト型
　これからは EWS の時代
・エディタ
　エディタと英文ワープロとの違いは？
　エディタに要求されるもの
　「ソフトとは本質的にはエディタである」
・アセンブラ
　アセンブリ言語/アセンブラというツール
　マクロ機能/クロス・アセンブラ
・コンパイラ
　高級言語のメリット
　コンパイラのデメリット
・リンカ
　リロケータブル・オブジェクト・ファイル
　リンクとは/リンカを活用する
・ライブラリアン
　モジュールのライブラリ化のメリット
　ライブラリアンの機能
・ダウンローダ
　オブジェクト・フォーマット
　　(インテル HEX/拡張インテル/モトローラ)
・デバッガ
　パソコンのデバッガ/ハードウェアのデバッガ
・ ICE
　ICE の機能
　ICE によるリモート・デバッグ
・モニタ
　モニタの機能/モニタによるデバッグ
　モニタを自作する
・ ROM ライタ
　EPROM のプログラム/ROM ライタの活用
・ OTPROM/ピギーバック
　ワンタイム EPROM
　ピギーバック型 CPU
・ブレッド・ボード(BB)
　ブレッド・ボードとは/BB 開発のコスト
　BB による開発のメリットは
　コア CPU の BB による開発
・シミュレータ
　シミュレータによる開発のメリット
　これからの開発の姿

(4) 具体的な CPU プログラムでの実戦的検討

L．<省略>

M．<省略>

N．各種ルーチンの考え方/組み方

・全体の構成
　見やすいソフト構造とは
　作りやすい/修正しやすい構造
　2次災害を生まない構造
・ソフト処理とハードウェア割り込みとの関係
　ソフト処理で済むもの/割り込みを使用するもの
・初期化ルーチン
　最優先の処理は何か
　リセット時の処理のいろいろ
・メイン・ルーチン
　できれば単純ループがよい
　検査ルーチンとの2重化
・割り込みルーチン
　割り込みルーチンの位置付け
　非同期処理の注意点
　割り込み処理の鉄則
　通常ルーチンとのインターフェースのポイント
・処理プライオリティの付け方
　割り込みコントローラ
　ベクタ配置による自動的処理
　ソフトによる簡単な方法
　プライオリティの変更
・デバッグのための処理
　エラー・トラップ/エラー・メッセージ
　RS-232-Cの活用
　テスト・モード/隠しモード/裏モード
・暴走対策
　ノイズ関係→ハード屋と共同で
　ソフトは基本的に暴走に弱い
　冗長さによる対策
・ソフトを外注に出す場合の注意点
　ソフトハウスは当業界ソフトのノウハウを知らない
　自分でソフトを書けなければ外注管理能力はなし！
　時間をカネで買うことの認識

O．プログラム・テクニック/バグ出しテクニック

・ラベルの活用
　絶対アドレスは絶対に使わない
　アセンブラのラベル制限に注目
　コメント代わりにラベルを使う
・メモリ/レジスタの参照
　データは圧縮できる←→スピードは落ちる
　共通メモリ・アクセスの罠
　レジスタのスタックの注意点
・フラグ・ビットの活用
　ビット操作命令の活用
　CPUアーキテクチャに対応したワザがある
・間接アドレシング
　Z80の間接は最悪
　マニュアルをよく読む/CPUの個性をフルに活用する
・パラメータ・テーブル
　パラメータ比較の悪い例/良い例/もっと良い例
・ディシジョン・テーブル
　制御ソフトは場合分けのカタマリ
　テーブル・ジャンプの美しさ
　最後の処理を忘れずに
・マクロ/サブルーチンの活用
　似たような処理を並べない
　マクロのメリット/デメリット
　サブルーチンのメリット/デメリット
　パラメータの受け渡し
・BIOSを作る
　パソコンのグラフィックBIOS
　BIOSコールによるプログラム製作
・デバッグ・ルーチン
　ジョブ・モニタ/履歴記録
　条件アセンブル
　ステータスのディスプレイ
・他社に解読させないための方法
　パソコンのコピー・プロテクト/高額ソフトのプロテクト
　データ/パラメータの暗号化
　データ埋め込み
　逆アセンブル・トラップ
　バス信号線の暗号化
　ROM内蔵コアCPU化
　解読されて困るようなことはしない！(特許)
・自己診断機能
　メモリの検査(ROM/RAM)
　バージョン・チェック/周辺LSIの初期確認
　ソフト的ランタイム・カウンタ/イベント・カウンタ
・工場出荷検査機能
　特定の操作＋電源ON
　信号ループ
　LED点灯/SWスキャン
・隠し機能：リモート・デバッガ
　RS-232-Cの活用
　RAMエリアが大きいものは実現可能
・バグの出し方：境界値分析
　開発担当者のバグ出しはザル
　めくらめっぽう方式の欠点
　境界値分析の手法
　人間の手は2本という盲点
・パネル・スイッチ関係のバグ
　「考えられない組み合わせ」を考える
　導電ゴム接点の注意点
・誤動作対策関係のバグ
　ノイズで完全にリセットしていいのか
　UNDO/レジューム機能

事典編

1 マイコン技術者のための［基礎］概論

ディジタル技術の基本

●論理回路

ディジタル回路技術の基本となる「論理回路」は，何種類かの「ゲート」(**図1.1**)で構成されます．つまり，

- 1本の信号の反転(**否定**)であるインバータ
- 2入力1出力の論理積(**アンド**)
- 2入力1出力の論理和(**オア**)

の3種類を知っていれば，原理的にはあらゆる論理回路を構成できることになります．

また，これとは別に，ゲートアレイを設計する際によく使われる方法として，

- ［アンド＋インバータ］と等価なNAND(**ナンド**)
- ［オア＋インバータ］と等価なNOR(**ノア**)

のいずれか1種類だけを組み合わせることによっても，他の論理ゲートはすべて実現できます．

「排他的論理和」(**エクスクルーシブ・オア**)というゲートも基本的なものですが，この機能としては，

- 二つの入力の一致・不一致を検出する
- 一方の入力を他方の信号を反転するための制御信号とする

というような性質があり，演算回路などを構成する上でいろいろに活用されます．

そこで，**図1.2**のように，基本ゲートの組み合わせで，いろいろな機能のディジタル回路を設計してみましょう．TTLデータブックには，各ICの等価回路も掲載されていますから，真理値表から等価回路を設計して確認してみる，というのは最高の勉強となります．

●順序回路(同期回路)

論理回路が「入力が揃うと出力が決まる」という静的な回路であるのに対して，順序回路(同期回路)とは，

▷動作の基準となる時間信号(基準クロック・パルス)が存在して，

▷その1クロックごとに同期して状態が変化する「動的」な回路

と特徴づけることができます．CPUそのものを含めて，ディジタル回路としてなんらかの時間的動作を行うものは，すべて順序回路をもっていることになります．

順序回路の基本は，**フリップフロップ**(F/F)**によるトグル動作**にあります．**図1.3**にあるような，もっとも単純なSRフリップフロップや，実用的なJKフリップフロップとかTフリップフロップについては，回路図の上で動作を追ってみたり，実際にTTLの動作をテスタで確認してみましょう．この延長には，**図1.4**のような，F/Fを多段に重ねたカウンタ回路やシフト・レジスタ回路，あるいは複数個を並べたラッチ回路などがあります．全体のゲート規模は大きくなりますが，これらの回路の基本的な動作原理は個々のF/Fに分解して理解することができます．

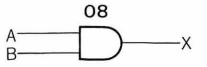
同期回路の考え方をさらに進めると，**図1.5**のように，クロック信号のタイミングに同期してシステムの各部分を動作させて，クロックとクロックの間(図では立ち上がりエッジ同士の間)での回路状態の遅延を吸収してしまう，という発想が重要になります．これは，ASICのような大規模な回路を設計する場合の定石的テクニックで，マージンを十分に確保した回路設計の秘訣ともいえるものです．

〔図1.1〕ゲート ── 基礎の基礎

ゲートとは，ディジタル論理回路を構成する基本的な論理単位のことで，通常は図のようなものをいう．論理演算の基本は2入力1出力(と反転)であり，これらのゲートを組み合わせることで，あらゆる回路を構成することができる．また，トランジスタ回路の動作はNANDそのものなので，TTLの最初の番号である**00**によって，他の基本ゲートもすべて作ることができる．

〈各種ゲート〉

●アンド(AND)

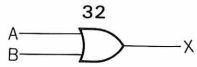


IN		OUT
A	B	X
L	L	L
H	L	L
L	H	L
H	H	H

・ともに“H”入力のときのみ出力は“H”

$X = A \cap B$

●オア(OR)

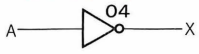


IN		OUT
A	B	X
L	L	L
H	L	H
L	H	H
H	H	H

・いずれかが“H”入力ならば出力は“H”
(ともに“L”入力のときのみ出力は“L”)

$X = A \cup B$

●(インバータ(INV) ノット(NOT))

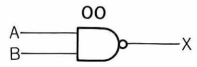


IN	OUT
A	X
L	H
H	L

・出力は入力の反対

$X = \overline{A}$

●ナンド(NAND)

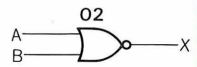


IN		OUT
A	B	X
L	L	H
H	L	H
L	H	H
H	H	L

・ともに“H”入力のときのみ出力は“L”

$X = \overline{A \cap B}$
$X = \overline{A} \cup \overline{B}$

●ノア(NOR)

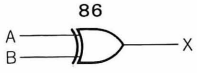


IN		OUT
A	B	X
L	L	H
H	L	L
L	H	L
H	H	L

・いずれかが“H”入力ならば出力は“L”
(ともに“L”入力のときのみ出力は“H”)

$X = \overline{A \cup B}$
$X = \overline{A} \cap \overline{B}$

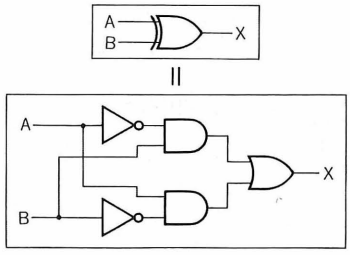
●エクスクルーシブ・オア(EXOR)



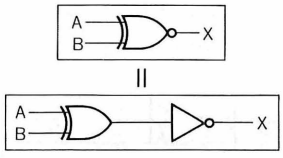
IN		OUT
A	B	X
L	L	L
H	L	H
L	H	H
H	H	L

・入力が異なるときは出力は“H”，入力が等しいときは出力は“L”

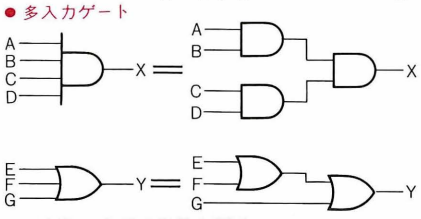
●エクスクルーシブ・オア(EXOR)



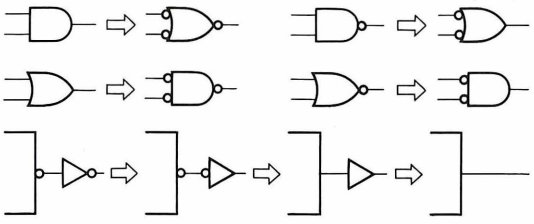
●エクスクルーシブ・ノア(EXNOR)



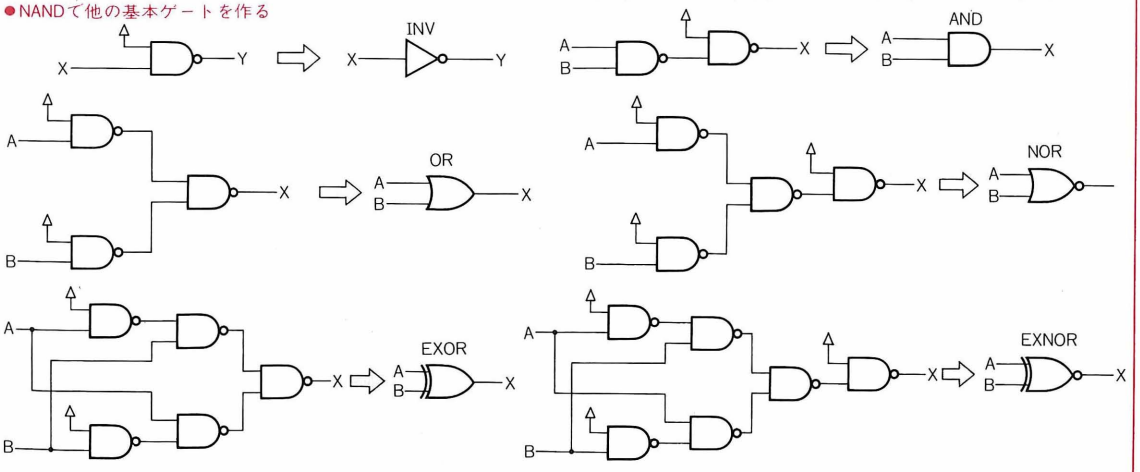
●多入力ゲート



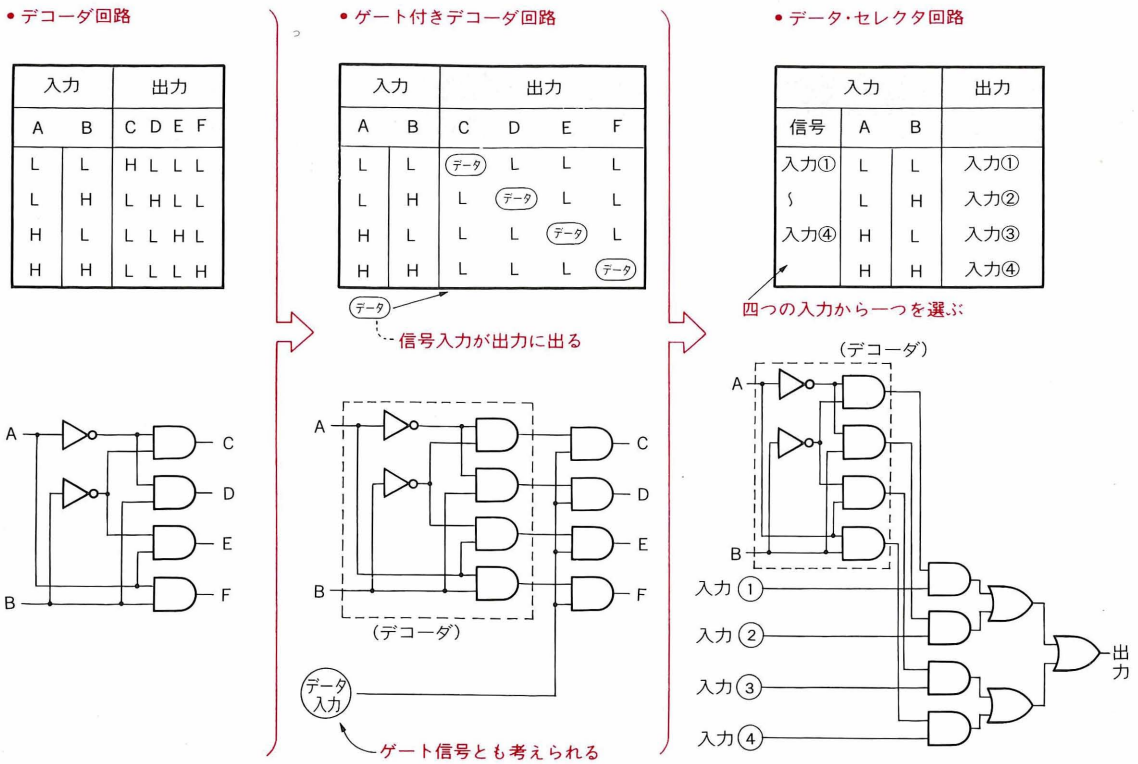
●正論理↔負論理の等価回路



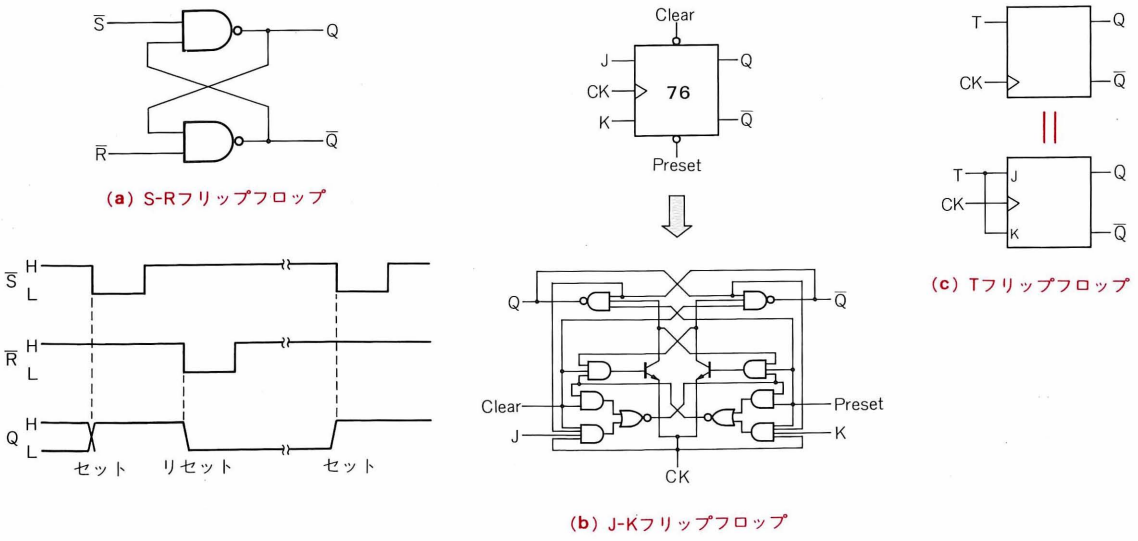
●NANDで他の基本ゲートを作る



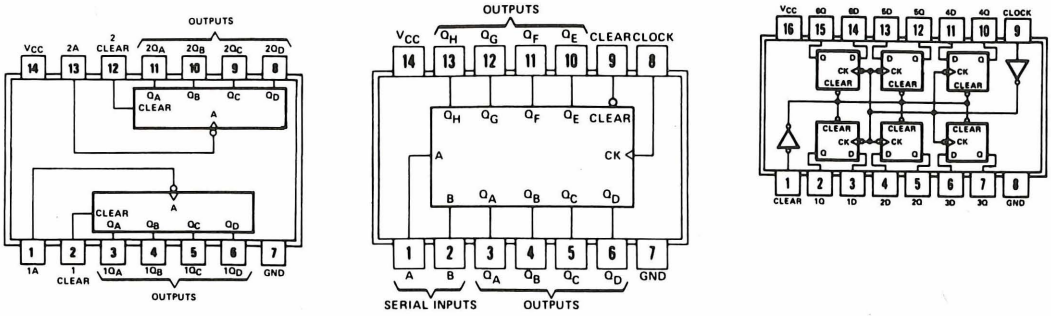
〔図1.2〕基本ゲートによる等価回路の設計例



〔図1.3〕フリップフロップ(F/F)

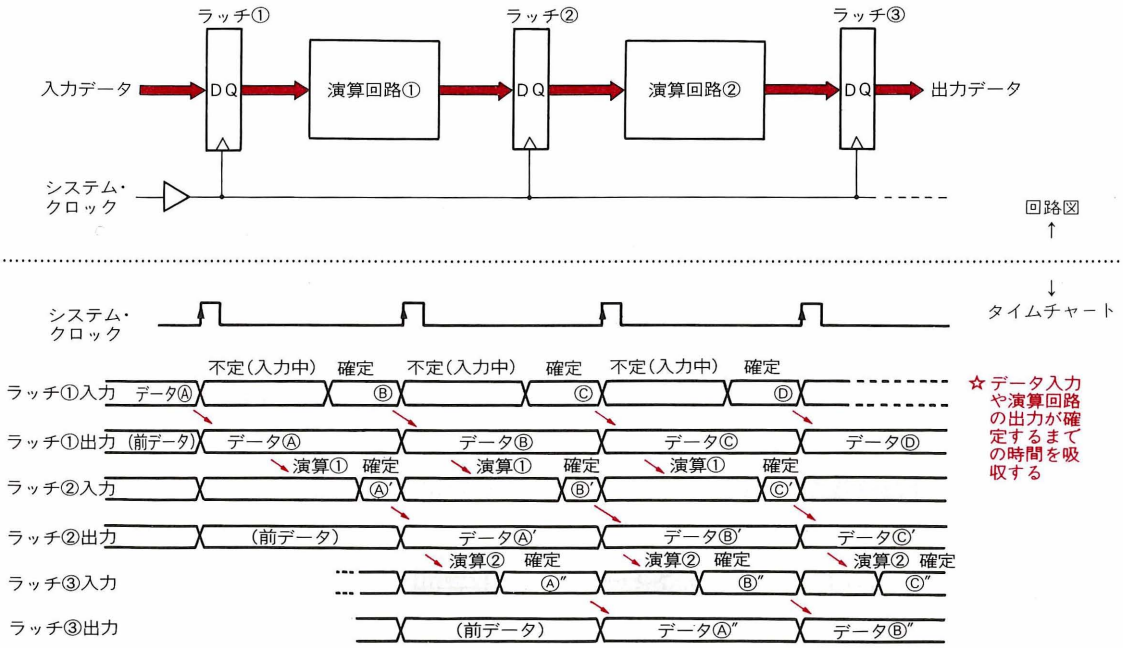


〔図1.4〕フリップフロップを多数ならべた機能回路の例



（通常，74393，74164 … の74は省略することが多い．本書でもその流儀にしたがう）

〔図1.5〕同期的なディジタル回路設計とは…



●演算回路

演算回路は，基本ゲートに分解すれば「高度に複雑な論理回路」ですが，加減乗除などの代表的な回路については，出会ったところでマスターしてしまいましょう．なお，**図1.6**のような2進数・8進数・16進数などの数値表現とか，**図1.7**のような，数値データの符号と補数の表現も確認しておきましょう．

演算回路の基本的要素である加算器は，**図1.8**の1ビットのハーフ・アダーで，二つをまとめてフル・アダーが構成できます．これを多段にわたって(キャリ信号をつぎつぎに繰り上げて)重ねていけば，原理的には任意のビット幅の加算器ができることになります．実際のASICやDSP回路では，キャリ信号が繰り上がる遅延時間が問題となるために，ルックアヘッド・キャリ回路によってスピードを稼ぎます．

また**図1.9**のように，2進数ではデータの2倍は1ビット・シフトに相当しますから，シフト・レジスタと加算器の組み合わせによって，乗算回路が実現できます．ここに補数変換のための符号反転回路(エクスクルーシブ・オア群)を加えると，「符号付き乗算回路」が構成できます．ちょっと課題として考えてみてください(**コラム 解答1** 参照)．

〔図1.6〕いろいろなデータ表現

(例)

10進法 … $100_{10} = 1\times10^2 + 0\times10^1 + 0\times10^0$

2進法 … $100_{10} = 1\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0 = 1100100_2$

8進法 … $100_{10} = 1\times8^2 + 4\times8^1 + 4\times8^0 = 144_8$

16進法 … $100_{10} = 6\times16^1 + 4\times16^0 = 64_{16}$

2進数 1100100_2 を8進数として見るには → (3ビット単位で見る) 1100100 → 144_8
(1 4 4)

2進数 1100100_2 を16進数として見るには → (4ビット単位で見る) 1100100 → 64_{16}
(6 4)

〔図1.7〕符号のついた整数の表現

① リニア(絶対値)表現

255	11111111	FF
↕	↕	↕
0	00000000	00
(10進)	(2進)	(16進)

② 2の補数

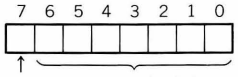
127	01111111	7F
↕	↕	↕
1	00000001	01
0	00000000	00
−1	11111111	FF
↕	↕	↕
−128	10000000	80

③ 1の補数

127	01111111	7F
↕	↕	↕
0	00000000	00
−1	11111110	FE
↕	↕	↕
−127	10000000	80

☆「ゼロ」が00hとFFhの2通りに表現されてしまう．

④ 符号(サイン)＋絶対値

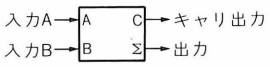


127	01111111	7F
↕	↕	↕
0	00000000	00
−1	10000001	81
↕	↕	↕
−127	11111111	FF

☆「ゼロ」が00hと80hの2通りに表現されてしまう．(255段階になってしまう)

〔図1.8〕加算器(アダー)の構成

〈ハーフ・アダー〉



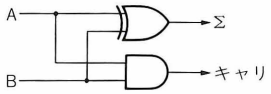
(a)

入力		出力	
A	B	Σ	キャリ
L	L	L	L
L	H	H	L
H	L	H	L
H	H	L	H

0＋0＝0
0＋1＝1
1＋0＝1
1＋1＝0，繰り上がりあり
} 2進数の加法

EXORと一致
ANDと一致

(b)



(c)

ハーフ・アダー(半加算器)の機能は(**a**)のようになる．真理値表は(**b**)のようになる．ここで出力Σは

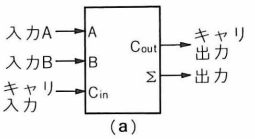
AとBのEXOR(排他的論理和)，

キャリ出力は

AとBのAND(論理積)

になっている．したがって，ハーフ・アダーの回路としては(**c**)のようになる．

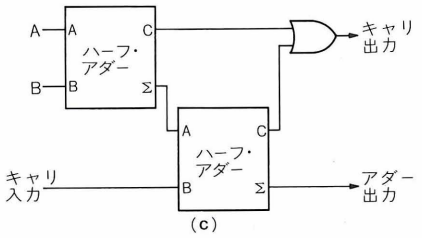
● 1ビットのフル・アダー



(a)

入力			出力		
A	B	Cin	Σ	Cout	
L	L	L	L	L	0＋0＋0＝0
L	L	H	H	L	0＋0＋1＝1
L	H	L	H	L	0＋1＋0＝1
L	H	H	L	H	0＋1＋1＝0　繰り上がり
H	L	L	H	L	1＋0＋0＝1
H	L	H	L	H	1＋0＋1＝0　繰り上がり
H	H	L	L	H	1＋1＋0＝0　繰り上がり
H	H	H	H	H	1＋1＋1＝1　繰り上がり

(b)

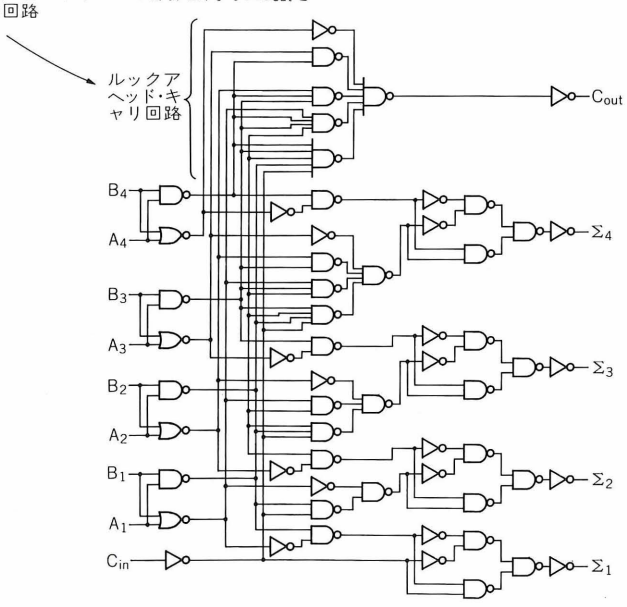


(c)

フル・アダー(全加算器)の機能は(**a**)のようになり，真理値表は(**b**)のようになる．つまり入力のA，B，キャリ入力という三つが対等の関係になる．フル・アダーは二つのハーフ・アダーとORゲートにより(**c**)のように構成できる．

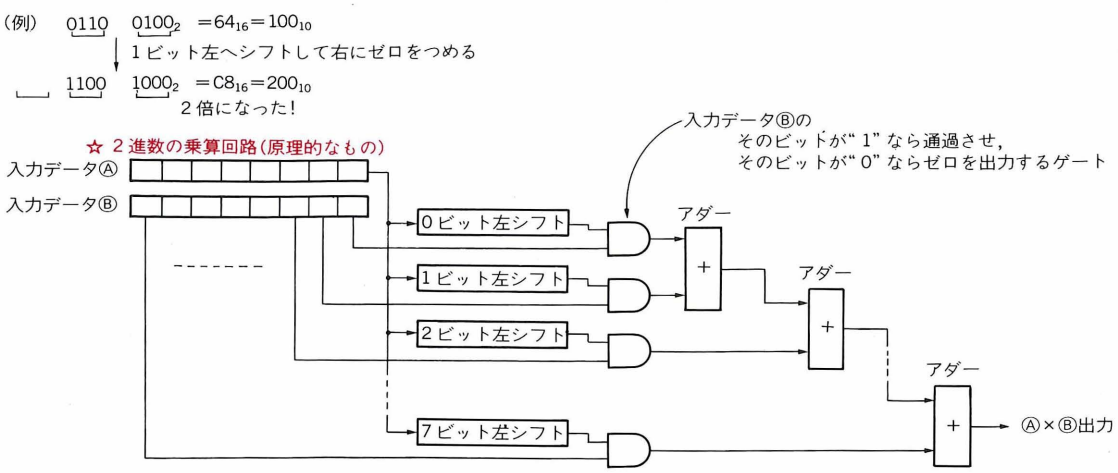
〈ルックアヘッド・キャリ〉

入力の組み合わせから，各ビットのΣの計算と同時にC_{out}を出力してしまうための回路



(4ビット・フル・アダー 283の等価回路)

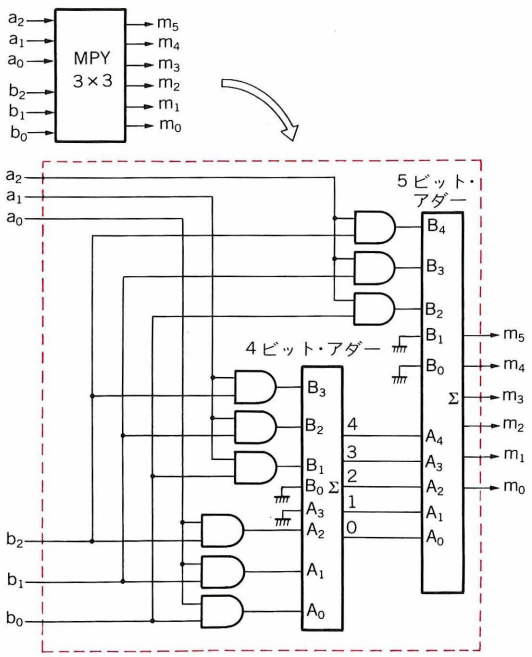
〔図1.9〕乗算器(マルチプライヤ)の構成



[コラム] 解答1

符号付き乗算回路の例

〔図A〕リニア表現の乗算回路(例:3ビット×3ビット)



図Aは,リニア表現の3ビット・データ同士を乗算する回路を,加算器によって構成した例です.それぞれ(0～7)の値をとる入力データの乗算結果(0～49)が,6ビット・データとして出力されます.

つぎに,このリニア乗算回路を使って,「1の補数」表現の4ビット・データ同士を乗算する回路を構成した例が,**図B**です.最上位ビットは「サイン・ビット」ですから,それ以下のビットのデータをサイン・ビットによって反転することで,符号以外の部分をリニア乗算します.符号部分の乗算というのは,ちょうどエクスクルーシブ・オア演算に相当します.そして乗算結果を,再び符号変換によって「1の補数」に戻しています.ここでは,ずらりと並んだエクスクルーシブ・オア・ゲート群が,「制御信号によってデータを反転するゲート」として活用されています.

コンピュータでよく使われる「2の補数」にするには,このデータにさらに,「結果が負の場合のみ,1を加える」という操作が必要になります.ところが,一般にこのような乗算回路の出力というのは,他のデータとさらに加算(累算)される場合が非常に多いために,**図C**のように,次段の加算器のキャリ入力として,この出力のMSB(サイン・ビット)を与えることで,そのまま2の補数形式に変換してしまいます.

[飛び石コラム] おすすめBOOKS ②

● SE スキルアップ NOTE ── 上級ソフトウェア技術者になるための実践計画

小暮裕明　著　　CQ 出版社

この本は，若手のソフトウェア技術者が，SE(システム・エンジニア)としてスキルアップしていくための指針として，10 年計画で各種の SE 技術を体得していく段階について解説しているものです．じつは，本書はこの本のハードウェア版として企画されたものなのですが，両方を読み比べてみるのも面白いかもしれません．

●情報と文化

情報文化フォーラム　編　　NTT 出版

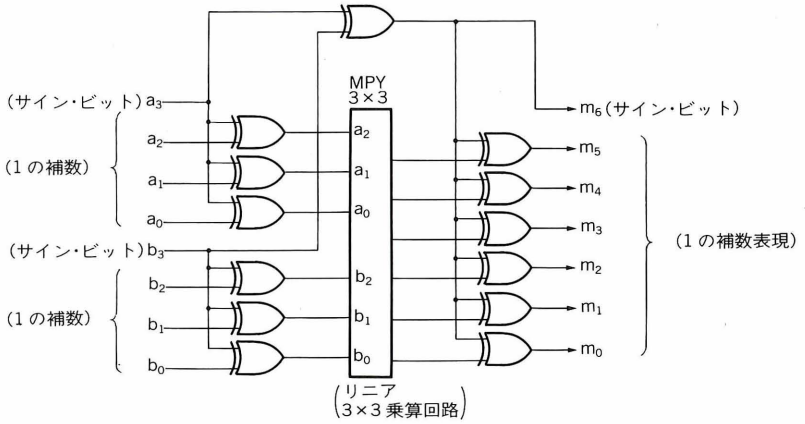
情報，生命，進化，文化，芸術，真理，哲学といった種々雑多なテーマを横断的に網羅してしまおう，という大胆・壮大な試みの本です．個々の分野では掘り下げが不足している不満もあるのですが，自然科学から情報科学，そして美学・哲学・論理学から文明論までを意欲的に追求する姿勢は，優れた読み物として称賛できます．

●人間工学 ── 装置設計者のための ──

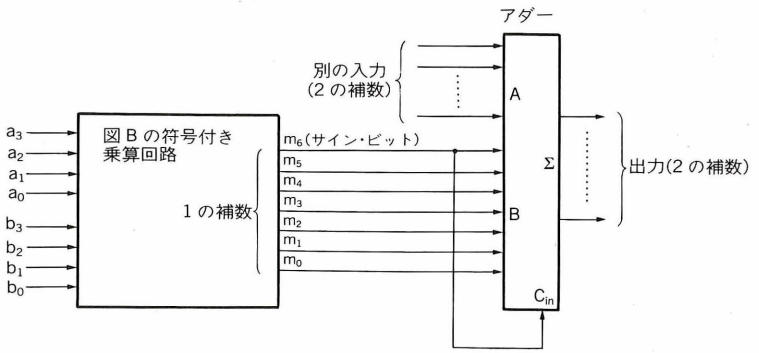
W.E. ウドソン　著　　青木和彦・野本明　共訳
コロナ社

マン-マシン・インターフェースとか，ヒューマン・インターフェースといった視点の歴史は非常に長いものです．この本はかなり古い(古書店で入手しました)ので，あるいは現在ではこの分野の別の本を捜す必要があるかもしれませんが，いずれにしても「人間工学」について知ってみることは，とても勉強になると思います．

〔図 B〕
符号付き乗算回路
(1 の補数，4 ビット×4 ビット)



〔図 C〕
図 B の符号付き乗算回路の利用法



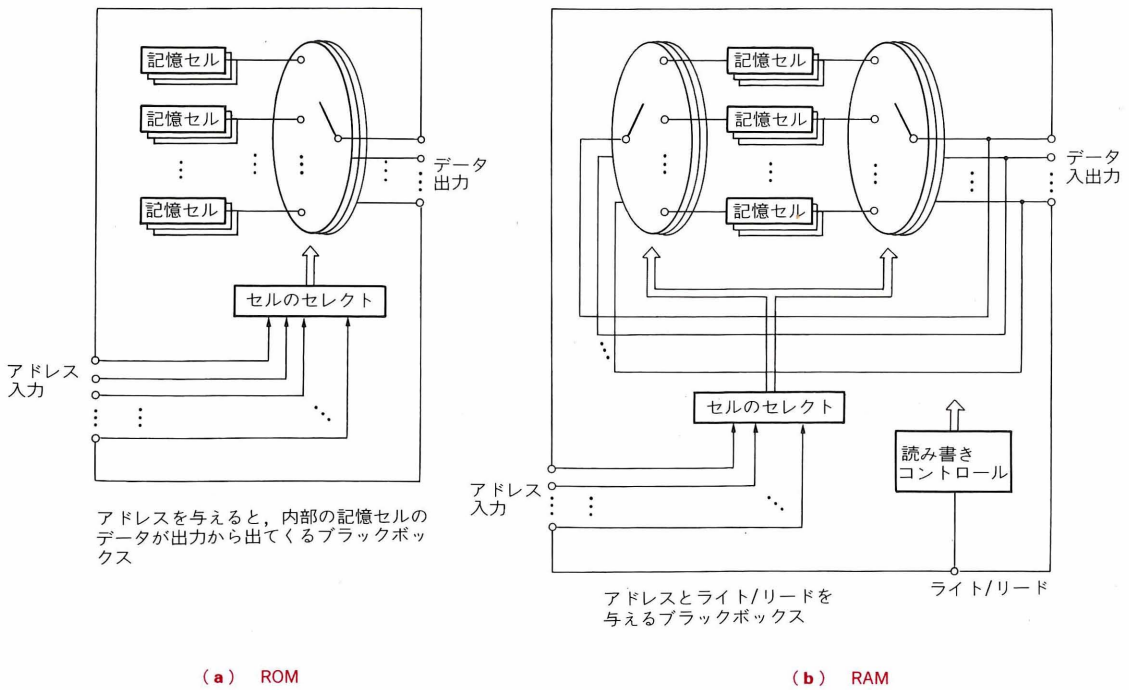
●記憶回路

先にふれたフリップフロップ回路は，入力データをクロックで確定させてつぎのクロックまで保持する，という意味では，もっとも基本的なメモリ(記憶)回路です．そしてF/Fを複数個並べたラッチ回路は，まさに1アドレスのメモリ回路そのものです．また，シフト・レジスタを複数ビット並べた，あるいはラッチを多段重ねたと見ることができるFIFO回路もメモリ回路の一種です．

マイコン・システムでは，**図1.10**のように，内部の記憶セルをブラックボックス(F/Fであるとは限らない！)にしたLSIとして，メモリ(RAMやROM)が中心的な活躍をします．ここではメモリの理解のために，やや意外な方法ですが，「ROMやRAMを単独で(CPU回路以外に)使う」ことを考えてみましょう．

非CPU回路にメモリICを使ってみようとすると，その情報容量の膨大さに驚かされます．たとえば8464というRAMは，374というラッチにして8192個分のデータを読み書きすることができ，27256というPROMは，8ビットのDIPスイッチを32768個を並べたことに相当します．もちろん，これらの情報は厳密に「同時に」扱うことはできませんが，「順序回路」のテクニックの応用課題としては格好のものです．

〔図1.10〕「メモリ」の考え方(ブラックボックス)



アナログ技術の基本

●アナログ技術の奥はとても深い

ここではアナログ回路技術の「ごく一部」について，基本技術のチェックポイントを考えましょう．「一部」と強調したのは，アナログ技術というのはディジタルよりも奥が深く，限られた誌面では十分に掘り下げきれないからです．エレクトロニクスの技術者にとって，アナログ技術のセンスというのは永遠の課題であり，ベテランのアナログのノウハウこそ，フレッシュマンにとって最大の目標であると思います．

●受動回路・能動回路

最初に必要なのは，アナログの基本となる「受動回路」の理解です．これは，抵抗とコンデンサとコイルの組み合わせに対して，電気信号がどういう振る舞いをするか，という「定性的な理解」のことです．いきなり電気系学科のような複素ベクトルや偏微分方程式などは不要で，まず最低限のアナログのセンスとしては，**図1.11**のような，アマチュア無線の試験用に小学生が覚える程度で十分です．

また，トランジスタによる「能動（アクティブ）回路」のノウハウは，これまでのエレクトロニクス技術者の領域では中心的なものでした．しかし筆者は，マイコン・システム技術のフレッシュマンにとって，この知識は「あと回しで結構」とあえていってしまいましょう．なぜなら，

- 技術的に獲得すべき，プライオリティのより高いものが他にも多い

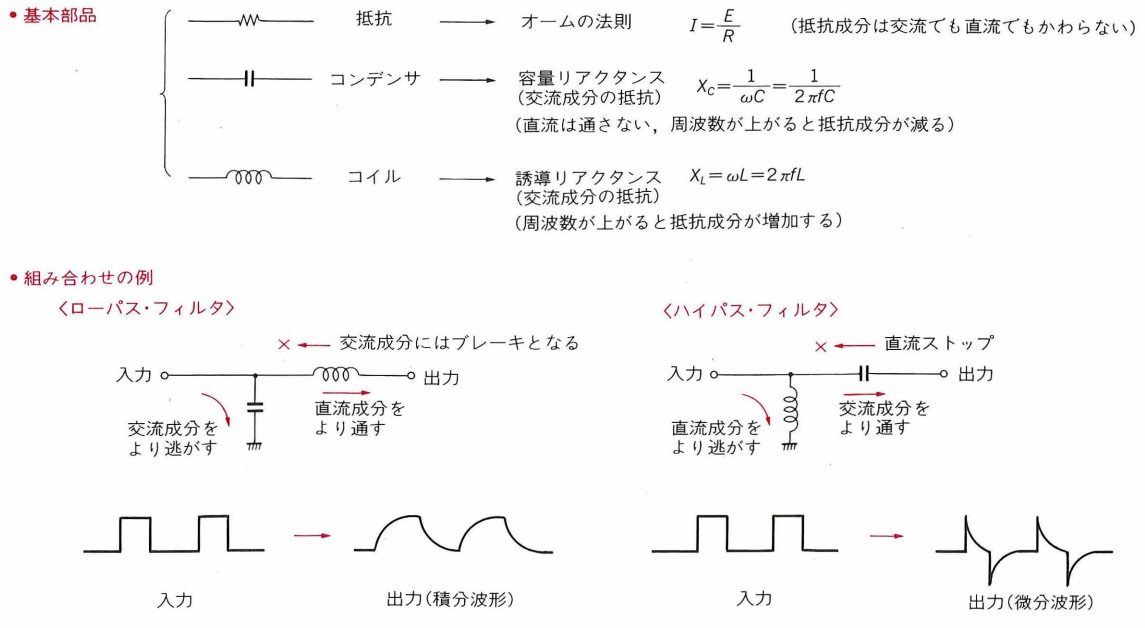
というのが第一ですが，

- 知らなくても，とりあえずは何とかなる

という第二の理由もあるからです．というのも，後述するOPアンプやアナログICの普及(高性能・ローコスト・多種化)によって，マイコン・システムでは，ディスクリートのトランジスタ回路を具体的に設計しなくても，まず当面はほとんど困らない，という状況になりつつあるからなのです．

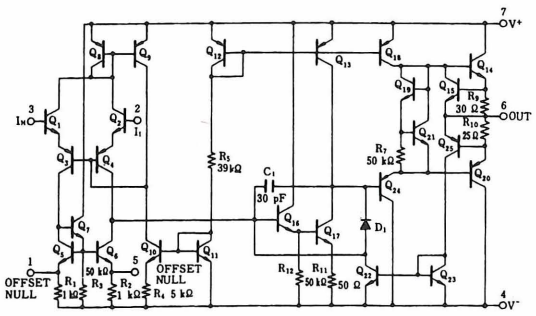
ディジタル回路で進んでいる「回路のブラックボックス化」という流れは，アナログ回路でも本流となっています．かつての「トランジスタ回路テクニック」を知らなくても，それぞれに要求される機能を1チップで実現してしまう，安くて高機能なアナログICがたくさんあります．たんなるバッファ・アンプ，たんなる電圧変換，たんなるインピーダンス変換回路であっても，理想特性のOPアンプを使えば，バイアス回路とかの余計な心配も不要で，全体の部品点数も減って

〔図1.11〕アナログ能動回路の基本



〔図1.12〕OP アンプの例(日本電気データブックより)

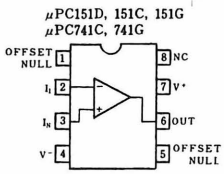
等価回路



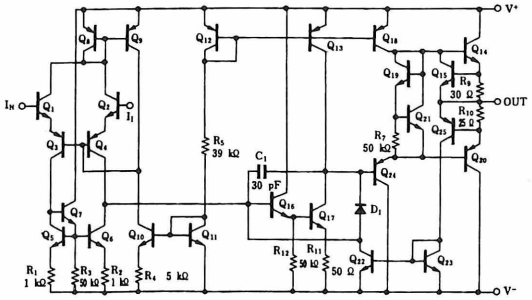
(a) 741(シングル)

端子接続 (Top View)

μPC151D, 151C, 151G
μPC741C, 741G



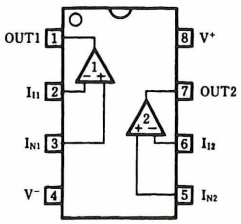
等価回路 (1/2回路)



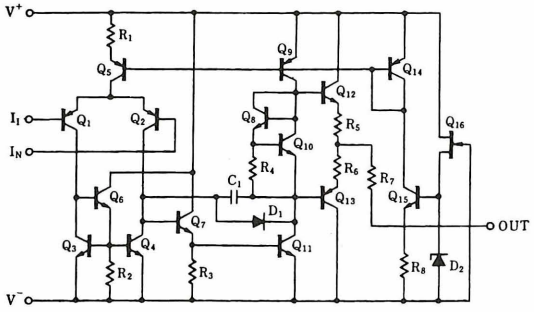
(b) 1458(デュアル)

端子接続 (Top View)

μPC251D, 251C, 251G/1458C, 1458G



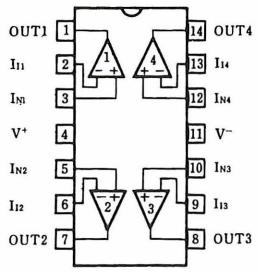
等価回路(1/4回路)



(c) 458(クワッド)

端子接続 (Top View)

μPC458D, 458C, 458G/4741C, 4741G



くれます．

誤解のないようにいいますが，もちろん「トランジスタ技術は不要」ということではありません．しかし，ディジタル技術よりもファジィなこの世界に不安をもつことなく，いずれあとでわかってくればいい，というぐらいの気持ちでいきましょう．

● OP アンプ回路

現在のアナログ回路の主役は，間違いなく各種のOP アンプ IC です．**図1.12**の等価回路のように，単一のトランジスタ増幅器の電気的特性の欠点を補うために，1個の OP アンプの中には多数のトランジスタが入っていて，いわゆる「増幅器の理想特性」をほぼ実現しています．1パッケージに2個とか4個のアンプがあったり，単一電源・低電圧電源・省電力タイプ・超高速タイプのものなど，種類もとても多くあります．これらの OP アンプを，まずは「完全にブラックボックスの便利な IC」として使ってしまえる，というのが重要なところでしょう．

●フィルタ回路での例

マイコン・システム技術の視点からすると，従来は OP アンプを複雑に組み合わせて構成していた回路機能が，ディジタル化によってかなりシンプルに実現されていきます．たとえば**図1.13**のように，

- 特性の揃った複数個の OP アンプ
- 理論的に要求される半端な(なかなかカラー・コードに一致しない)値の抵抗
- 高精度のコンデンサ(数値の要求は抵抗と同様)

を，多数(多段)並べた従来の「アクティブ・フィルタ回路」というものが，

- A-D 変換
- DSP によるディジタル・フィルタ
- D-A 変換

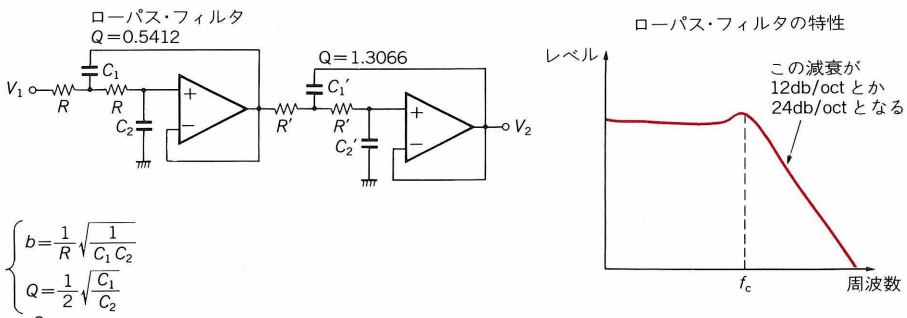
という構成によって実現できます．これによって，

- 事実上アナログでは不可能であった理想的なフィルタ特性が実現できる
- 無調整・コンパクトに実現できる(DSP のプログラム次第)

〔図1.13〕フィルタ回路の構成の例

(宮崎仁，「アクティブ・フィルタの実際」，『トランジスタ技術』，1991年8月号より)

〈アナログ方式〉



$$\begin{cases} b=\dfrac{1}{R}\sqrt{\dfrac{1}{C_1C_2}} \\ Q=\dfrac{1}{2}\sqrt{\dfrac{C_1}{C_2}} \end{cases}$$

$$\frac{C_1}{C_2}=4Q^2$$

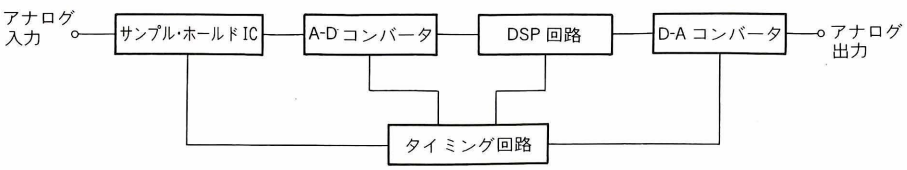
f_c＝1000(Hz)

$C_2=C_2'$＝1000(pF)，$b=2\pi fc$＝6283.2(rad/s)

C_1＝1172(pF)，C_1'＝6829(pF)

$R=R_2'$＝147.0(kΩ)

〈ディジタル方式〉



(実際にはアンチエイリアシングの LPF が入る)

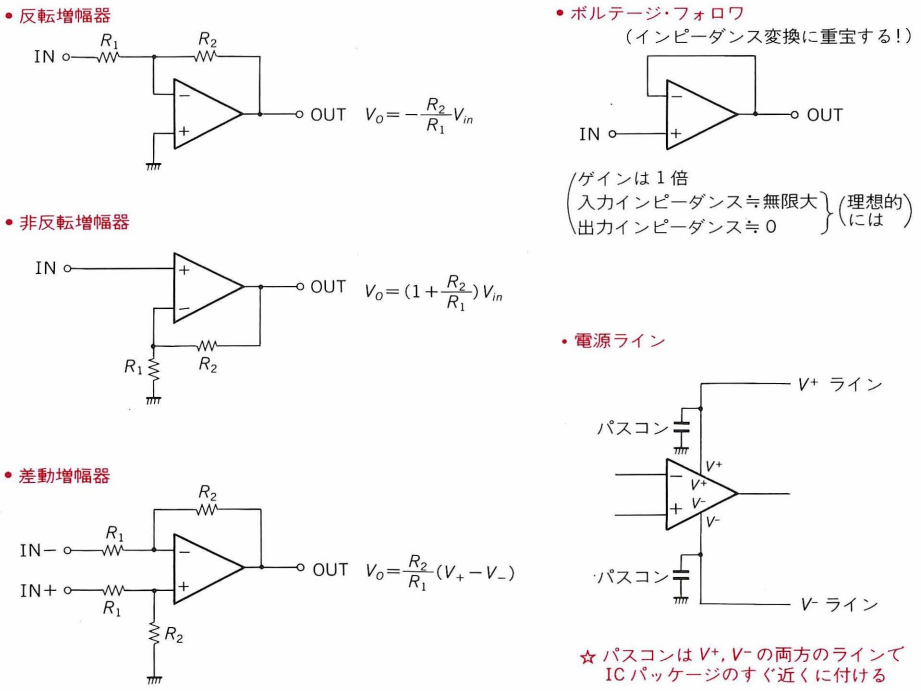
といったメリットが得られる時代になってきました．つまり，OPアンプの仕事が，システム回路の中心から，ディジタル・システムの「周辺部分」へと移っているわけです．

このような背景の中では，**図1.14**のようなOPアンプ回路の基本を知って，とりあえずアナログ信号のバッファ回路として活用できることが，まず必要になります．もちろん紙の上だけでなく，実際に手を使って実験しておくことが大切で，TTLなどのディジタル回路との違い，電源ラインの注意，ノイズなどのテクニックもあります．つぎのセンサの実験とともに，少しずつ慣れていきましょう．

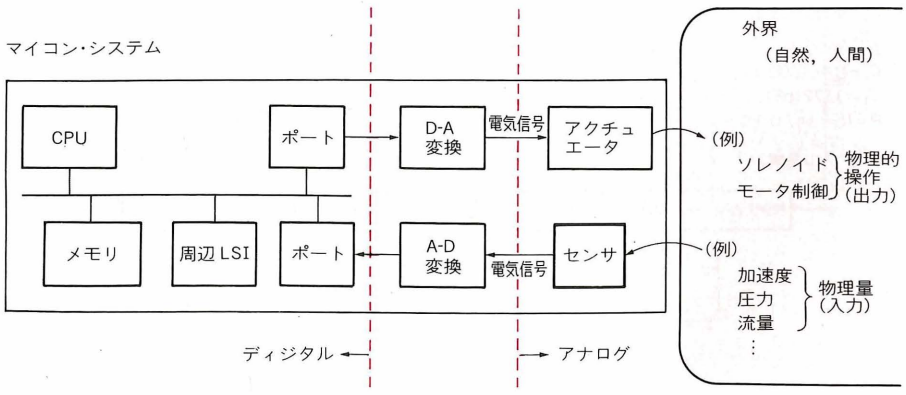
●センサ：外界との接点

マイコン・システムの中心のCPUはディジタルですが，人間の世界，あるいは自然界の物理現象はすべてアナログです．そこで**図1.15**のように，「アナログ↔ディジタル変換」という処理がつねに必要になります．

〔図1.14〕OPアンプ回路の基本



〔図1.15〕マイコン・システムと周囲との関係



そして，さらにその上流で，システムが外界のアナログ情報を取り込むために最初に必要なのがセンサということになります．

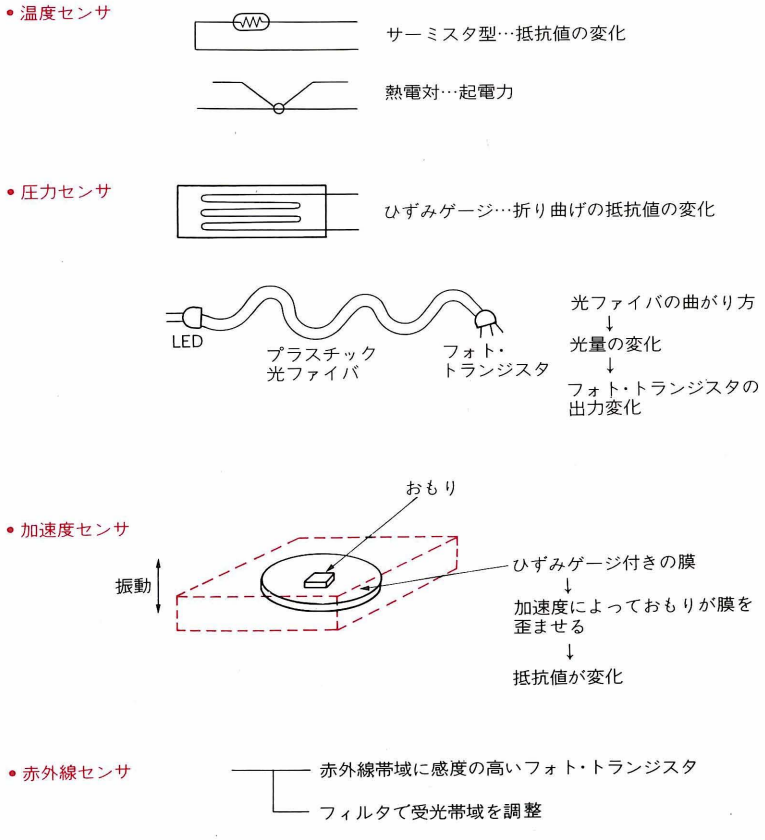
基本的には，ここでのセンサとは「いろいろな物理量を電気的な値に変換する」ものを指します．物理量というと大袈裟ですが，圧力とか温度とか明るさといった，人間の五感で検出できるもの，あるいは超音波や赤外線のようなものです．これらの情報がすべて電圧として出てくれるといいのですが，たとえば抵抗値とか周波数とかで得られるものも多く，この場合ディジタルで取り込むために，まず「アナログ-アナログ変換」（これを「シグナル・コンディショニング」ともいう）をする必要があります．

図1.16のように，センサの種類は文字通り（？）千差万別で，さらに同じ種類であっても，たとえば

●液体圧力計測専用の測定器に使われる圧力センサ
●スポーツ腕時計の水圧表示のための圧力センサ

の二つを比較すると，部品としてのセンサ単体の値段に1000倍くらいの開きがあります．その分，計測レンジ，精度，安定性，再現性，応答速度，寿命などの特性がいろいろと違うわけです．新しいLSIと同じように，機会があるたびに各種のセンサをサンプル実験してみることが，「センサのセンス」獲得の近道でしょう．

〔図1.16〕センサの例



マイコン・システムの3形態とトレードオフ

ここでは，マイコン・システムの具体的な形態として，ボード上にシステムが搭載されているものを考えていきます．いわゆる「ボード・マイコン」や，普通のパソコンがこの分類に入ります．これら以外に，最近の技術は「システム全体を1チップ上にまとめる」という，「システム・オン・チップ」を誕生させていますが，これについては［ASIC技術］の項で紹介します．

●マイコン・システムの3形態

なにか新しくマイコン・システムを開発する場合には，大きく3種類の形態が考えられます．これは**図1.17**のように，

- パソコンを使う(拡張ボードを使う)
- 市販のボード・マイコンを使う
- オリジナルのCPUボードを開発する

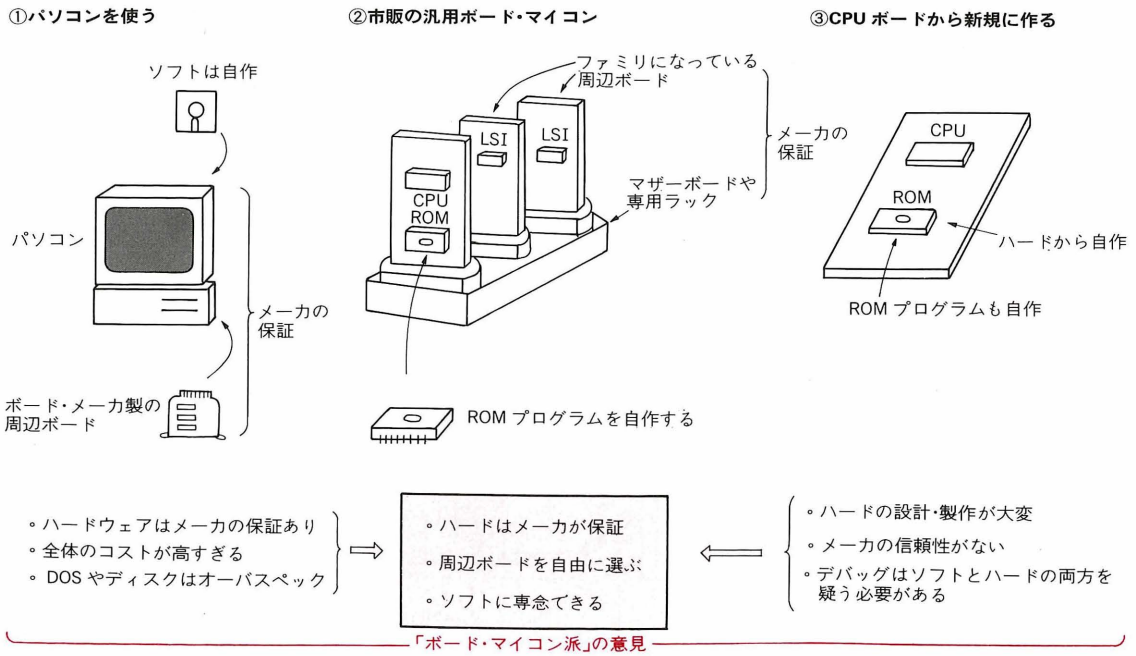
というもので，使用言語の種類を別にすれば，いずれもソフトウェアは自分で開発することになります．

あるテーマが与えられて，新しいマイコン・システムを開発していく場合には，この3種類のどのスタイルを採用していくか，という判断が必要になります．もちろんこれら3種類には，それぞれの長所と短所とがあって，いずれを採用するかは，それぞれのケースで何を優先するかによって異なります．たとえば，

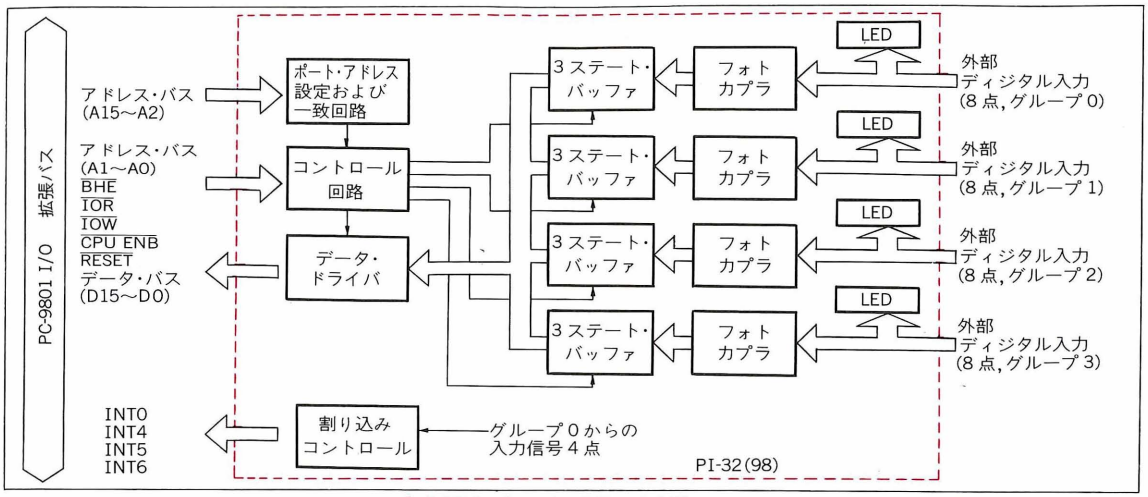
- 開発期間が非常に短い(確実に仕上げることを優先)
- 実験・設計・試作段階での初期開発費用をなるべく抑えたい
- 場合により同じシステムを量産する可能性もある(一品料理とは限らない)
- 大量生産時の低コスト実現が最優先
- システムが何度もバージョンアップされて長期間使用される見込み
- 海外(極端には宇宙)で使用されるので，無補修の信頼性が第一
- 悪環境なので，故障しても簡単に修理できるようにしたい

などのいろいろな条件が課せられるのが普通です．そこで，これらの制約要因に応じたトレードオフを考慮しながら，この3種類の形態も使い分けなければなりません．**図1.17**に書かれている「特徴」は，どちらかというと「ボード・マイコン派」の意見に重点を置いてみた視点の例になりますが，ちがう角度から見ると，

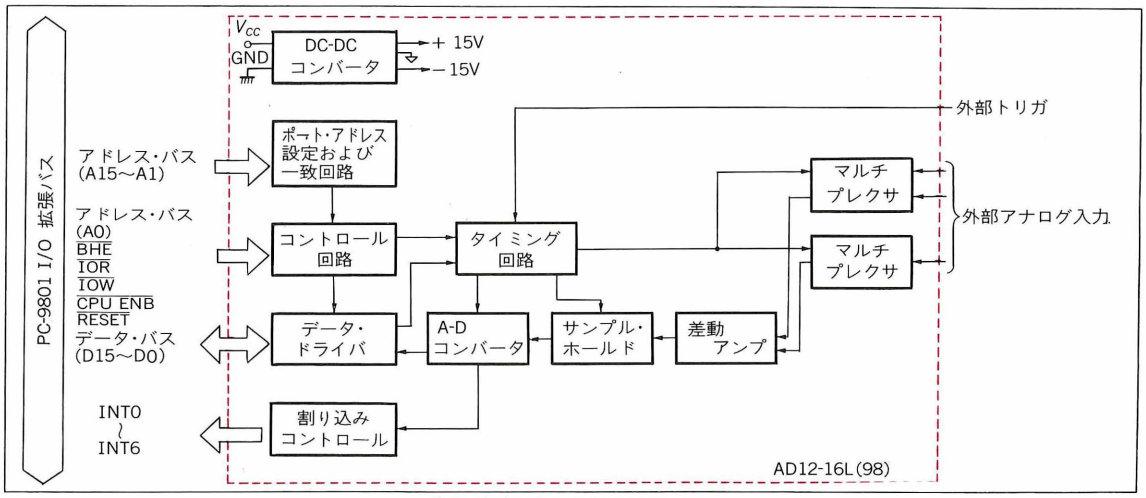
〔図1.17〕新しくマイコン・システムを開発する場合の3種類の形態



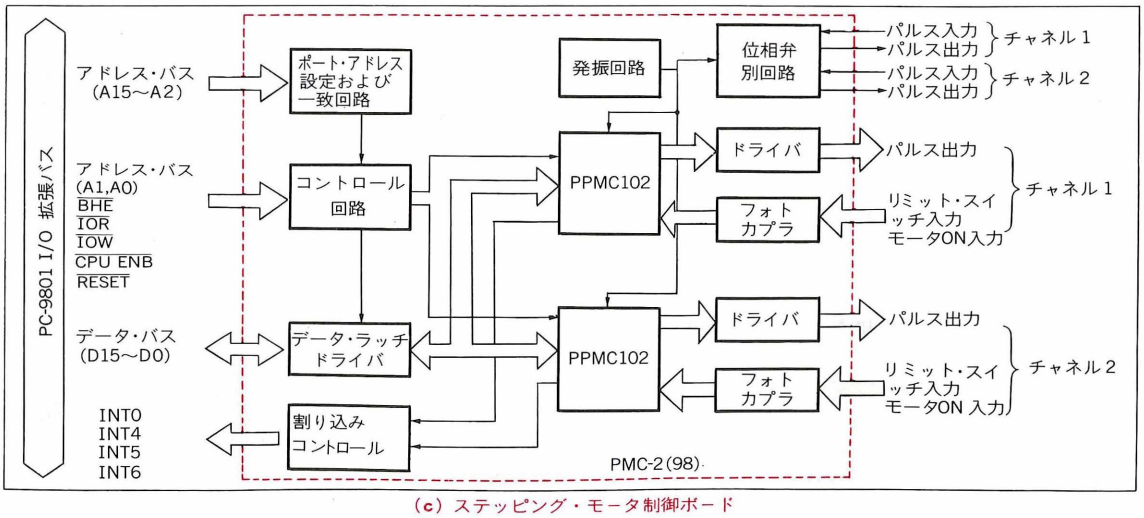
〔図1.18〕PC-9801 バスの周辺ボードの例(コンテック社データブックより)



(a) 32ビット・ディジタル入力ポート



(b) 多チャネルA-D コンバータ



(c) ステッピング・モータ制御ボード

3派それぞれにメリットとデメリットがあります.

●［パソコン＋ボード］システム

パソコン自体はプリンタとかRS-232-Cとかディスプレイなどのコネクタぐらいしかもっていませんから,一般の用途のために外部とインターフェースするためには,なんらかの拡張方法が必要となります.そこで,パソコンの拡張スロットに挿入する各種の拡張ボードが,多くの専業メーカから提供されています.

図1.18は,パソコンPC-9801シリーズ用のボードのデータブックから引用した例で,(a)は32ビットのディジタル入力ポート,(b)は多チャンネルのA-Dコンバータ,(c)はステッピング・モータの制御ボードのブロック図です.これらはほんの一例で,PC-9801シリーズであれば数社から数百種類のボードが提供されていて,わざわざ自作する必要がほとんどないほどです.この種のボードは,データシート,保証書,シリアル・ナンバがついていて,しっかり品質が保証されていることと,その代償として「かなり高価」(自分で作った場合の数倍から10倍以上)である,などが特徴です.実験室の試作用にはオーバスペックなのですが,組み込み機器やFAを意識してのことだと思えば,なんとか納得できます.

●［パソコン＋ボード］システムの長所と短所

このような方法で新しいマイコン・システムを構築するというのは,かなり大げさで,明らかにコストもかかる方法です.短所は,

- パソコン本体のコストがまず「部品」としては高い
- 拡張ボードのコストもかなり高い
- パソコンの図体が邪魔になり,機動性・可搬性に乏しい
- (DOS上の高級言語でソフト開発すると)アセンブラよりも遅い

などということになります.

しかし,パソコンを活用するための長所も多くあり,

- ハードウェアはほぼすべての部分に,メーカの信頼性の保証済み
- 同じシステムを複製する場合にとても簡単(ハードを買い増せばよい)
- パソコンの豊富な情報量(キーボード入力・ディスプレイ出力)を活用できる
- ディスク・ドライブ,ハード・ディスクなどの記憶媒体を活用できる
- ソフトウェア環境の充実により,短期間に開発可能
- バージョンアップなどのソフト修正が容易

などの点も捨てがたいものです.また,最近のノート・パソコンやブック・パソコンを活用すれば,機動性も大きく改善されます.

パソコンを「部品」として使う,という発想はなかなか面白いものですが,「お役所」からの仕事の受注の場合に活躍する,という裏話もあります.「カネに糸目をつけない」,「開発期間と信頼性を最優先」といったプロジェクトでは,この［パソコン＋ボード］をまず最初に検討してみるべきでしょう.

●「ボード・マイコン」によるシステム

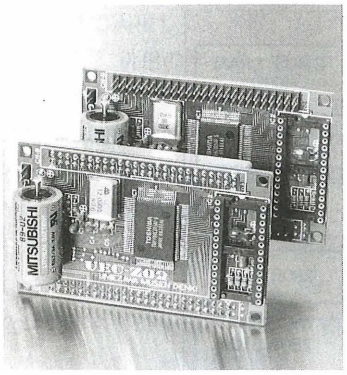
パソコンとは違って,ボード上にCPUと周辺LSIを搭載した,汎用の「ボード・マイコン」という製品を提供している専業メーカも多くあります.これには,メインCPUボードと各種周辺ボードとを組み合わせて使うシステムと,**図1.19**のように,1枚のボード上にコンパクトに各種機能をまとめたものとがあります.最近のCPUの発展にしたがった傾向としては,後者の1ボード汎用タイプのものが主流になりつつあるようで,**図1.20**にあるように,カード・サイズ(名刺サイズ)といっても,従来の8ビット・パソコンを軽く凌駕するくらいの,相当の能力を盛り込んでいます.

これらのボード・マイコンは,「CPUは68000シリーズ,共通バスはVMEバス」といった本格的なFA用ボードを別にすると,かなりのローコストで提供されています.そしてもちろん,ボード単体についての品質はメーカが保証しています.しかし,パソコンのDOSやコンパイラといった強力なソフトウェア環境とか,キーボード・ディスプレイなどの周辺入出力機器,さらにはディスク・ドライブのような外部記憶装置をもたないのが普通です.そこでマイコン・システム技術者としては,これらのソフトウェアとハードウェアの条件について,自分で解決して設計・開発ができなければなりません.開発のためのROMモニタ(後述)の付属したボードもありますが,場合によっては「モニタを自作する」必要もあるのです.

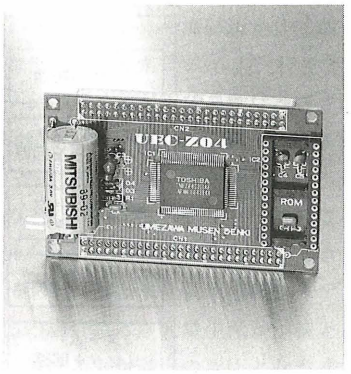
●ボード・マイコンによるシステムの長所と短所

上にあげた特徴から明らかですが,マイコン・システムとして汎用ボード・マイコンを採用する場合の長

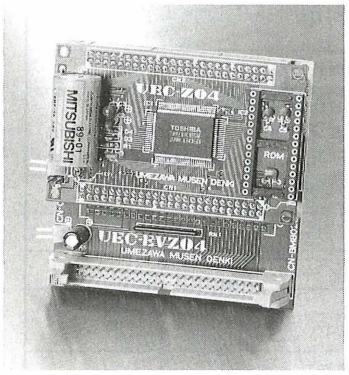
〔図1.19〕汎用ボード・マイコンの例(梅沢無線電機製)



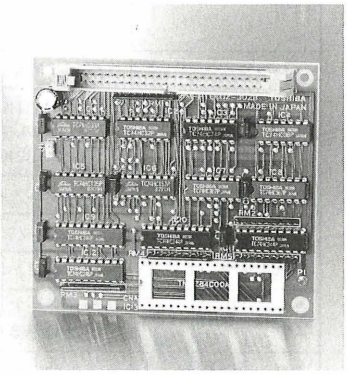
■UEC-Z01　9,100円(273)
UEC-Z02　9,800円(294)
・CPU84C015(Z80)
RAM 8Kバイト クロック 6MHz
システム・リセット バッテリ
バックアップ内蔵



■UEC-Z04　9,400円(282)
・CPU84C011(Z80)
RAM 8Kバイト クロック 4MHz
システム・リセット バッテリ
バックアップ内蔵



■UEC-EV01　22,000円(660)
UEC-EV02　22,000円(660)
UEC-EV04　22,000円(660)
開発ボード



■BM8027A　35,000円(1050)
015用アダプタ・ボード
■BM8026A　43,000円(1290)
011用アダプタ・ボード

所としては,
- ローコストである
- ボードの品質が保証されている
- 同じシステムを複製する場合に簡単(ボード・マイコンを買い増せばよい)
- システムの機能としてオーバスペックの無駄がない
- アセンブラで最適・高速なソフトウェアを開発できる

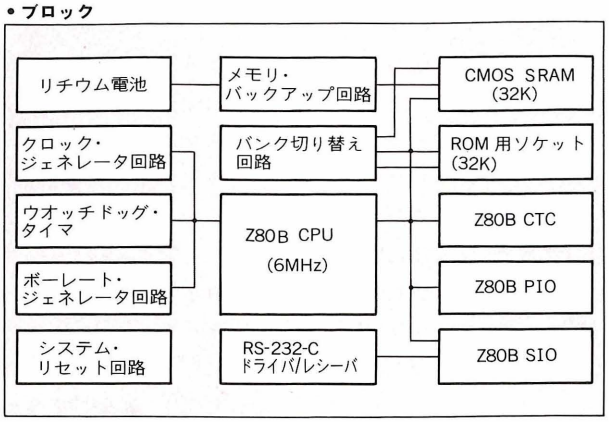
というような点がありますが，逆に短所としては,
- 周辺ハードウェアを自作する必要がある
- ソフトウェアの環境はかなり機械寄りである
- ソフトの改訂は普通はROMの交換となる
- ハード，ソフトの開発手法に工夫が必要となる

というような点があげられます.

［パソコン＋ボード］システムと比べてみると，エンジニア自身の技術力として，かなりの範囲の知識と力量が要求されています．ただし，技術的な壁をクリアしてこの方法でシステムを構築した場合には，たとえばパソコン応用のシステムの10倍以上のコスト・パフォーマンスを上げられるのが普通です．そのために，多くのシステムハウスが，各種の「開発支援ツール」を駆使して，この手法をフルに活用しています.

〔図1.20〕汎用ボード・マイコンの技術資料の例(梅沢無線電機製)

・ブロック



●基本仕様

項　　目	内　　容
CPU	・TMPZ84C015AF-6（東芝製） Z80B　CPU/PIO/SIO/CTC/CGC：クロック・ジェネレータ・コントローラ/ WDT：ウォッチドッグ・タイマ
クロック	・6.1440MHz（内蔵取り付け水晶は 12.2880MHz）
メ モ リ	・PROM　32K× 8 ビット(27256/27C256)　アドレス 0000H～7FFFH 32K バイト 基板埋め込み型超ローブロフイル丸ピン・ソケット付き（高さ 0.8mm） ・SRAM　32K× 8 ビット(TC55257AFL-10L)　アドレス 8000H～FFFFH 32K バイト
バックアップ	・リチウム・バッテリによるメモリ・バックアップ回路内蔵
バンク切替え	・メモリ・バンク切り替え回路内蔵（切替え専用 I/O ポート回路内蔵）
DRAM リフレッシュ	・64K のほかに，256K チップの DRAM リフレッシュが可能
暴走検出回路	・ウオッチドッグ・タイマ(WDT)回路内蔵 （CPU 暴走時にシステム・リセット動作可能）
スタンバイ機能	・各種動作モードをプログラム選択可能（4 種類）
リセット	・電圧検出システム・リセット回路内蔵　M51944B-ML(三菱製)使用
RS-232-C	・SIO の B チャネル側　RS-232-C　シリアル・ドライバ/レシーバ回路内蔵
ボーレート・クロック	・シリアル通信用に，ボーレート・ジェネレータ回路内蔵（75～19200bps）
入力/出力 インターフェース	・プログラマブル入出力ポート …… 8 ビット× 2 ポート ……(Z80B PIO) ・プログラマブル・カウンタ・タイマ ……(Z80B CTC) ・シリアル入出力コントローラ ……(Z80B SIO)
ゲートアレイ	・540ゲート（プロセス 2 μm，HC^{2}MOS シリコン・ゲート・メタル 2 層配線） ・内容（各種デコード，ボーレート発生回路など）内部回路は標準ロジック IC 換算にてすべて公開
使用コネクタ	・バス・コネクタ・ボード・マウント　ソケット（標準 2.54mm スタック・コネクタ）50極（金メッキ仕上げ）9150-4500SC（住友 3M 社製）　2 個
外形寸法	・54(縦)×85.5(横)×14(高さ)mm　40g(重量)
電　　源	・電源電圧　5V（±10%）単 1 電源 ・電源電流　標準 50mA（5V 標準動作時）
使用環境	・動作周囲温度 0 ～70℃，湿度90%以下（ただし結露なきこと）

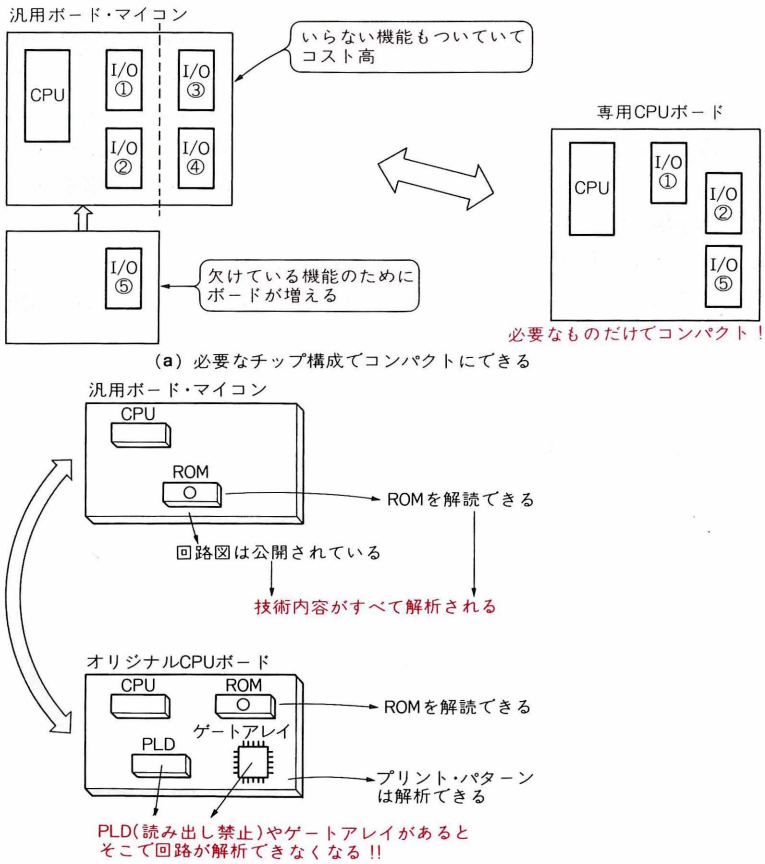
●オリジナルCPUボード・システム

第3の選択肢は，中核となるCPUボードまで，すべてオリジナル設計で実現してしまう，というものです．**図1.21**(a)のように，汎用のボード・マイコンというのは，メーカが多くのユーザを対象として，ある意味で「最大公約数」的な機能をもたせています．そこで，目的によってはボード上に不要な機能が無駄に置かれたり，わずかに欠けている機能を増設するためのもう1枚のボードを必要とする，というケースも出てきます．これに対して，オリジナル・ボードであれば，必要な機能だけをコンパクトにまとめたり，将来的な拡張機能の可能性を盛り込むこともできます．

また，**図1.21**(b)のように，ボード上の回路の「秘密保持」という側面も重要です．汎用ボード・マイコンは一般に回路図が公開されていますから，ボード上のCPUプログラムを格納したROMを解析すると，このシステムの中身がすべて明らかにされてしまいます．これはライバル・メーカや海賊版を作る者の存在が問題となる場合には，非常にデメリットとなります．ところがオリジナル・ボードであれば，まず基板パターンからの回路解析(4層とか6層の多層基板ではとくに効果的)という壁，さらにゲートアレイや読み出し禁止タイプのPLDを搭載するという壁によって，システムの解析を非常に困難にすることができます．

そして，家電機器とか自動車搭載システムなどの組み込み機器の場合には，大量生産が前提となって，なんといってもコストが最大の要因となります．そこで**図1.22**のように，1枚のボードに必要な機能を収めてしまうこと，場合によってはアナログまで混在させてしまうこと，あるいは基板の形状の条件(たとえばカメ

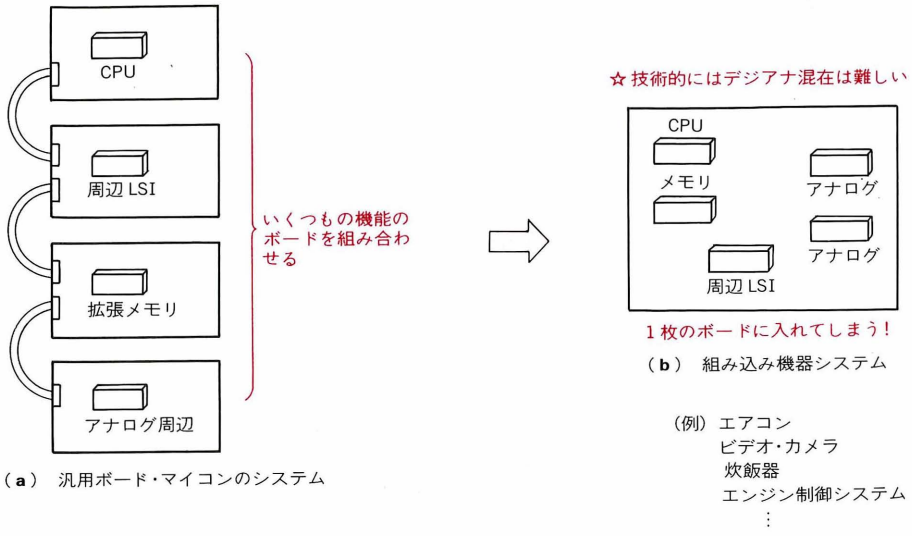
〔図1.21〕オリジナルCPUボードのメリット



(a) 必要なチップ構成でコンパクトにできる

(b) 回路上の秘密保持ができる

〔図1.22〕組み込み機器のCPUボード



ラ内部の，フィルム状のCPU基板)などがあると，これはオリジナルのCPUボードを開発するしかない，ということになります．

●オリジナル・ボードの長所と短所

まとめると，オリジナル・ボードの長所は，

- もっともローコストで量産製品向き
- 無駄のない，最適なハードとソフトを盛り込める
- システムの秘密を保持できる

といったものが上げられます．そして短所は，

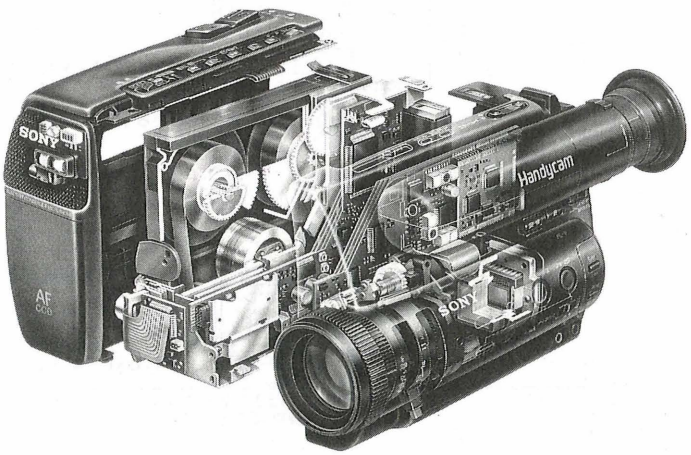
- ソフトに加えて，ハードウェアの開発も開発者の責任になる
- 初期投資として，開発期間・開発費用がもっとも多くかかる
- 信頼性は，ハードもソフトも自分で保証しなければならない

ということになります．筆者の個人的な意見として，あとひとつだけこの方法の「長所」を追加するとすれば，

- エンジニアにとっていちばん面白い(やり甲斐がある)

というところでしょうか．

組み込みのオリジナル・ボードの例
(ソニーのビデオ・カメラ，CCD TR55
ハンディカムの内部)



ヒューマン・インターフェース考

ここでは，マイコン・システムに共通の，外部から情報を入力したり外部へ情報を出力する，という部分に注目してみます．そして，多くの場合に対象となる「人間」を意識した「マン-マシン・インターフェース」とか，最近話題の「バーチャル・リアリティ」などの，やや未来物語の世界についても考えていきましょう．

●マイコン・システムの入力方式は？

パソコンの文字キーボードのように，「単語」「文章」という豊富な情報を扱えないマイコン・システムの場合，入力手段としてどのような方法があるでしょうか．まず最初に基本となるのは，**図1.23**のようなスイッチ・ボックスの検討です．フルキーボードはかなり高価で，キャラクタ表示装置も必要となる方法ですから，コンパクトなマイコン・システムとしては適当ではありません．しかし，パラメータの数だけずらりと小さなスイッチ群が並んでいる，というのも，あまり使いやすいものではありません．

そこで「少ないキーで効率的に操作する」という発想は重要です．このいい例がファミコンのコントローラで，基本的には上下左右の十文字キーと，たった二つのボタン・スイッチで，あらゆるゲームをコントロールできます．この場合，表示装置(マイコン・システムであればLCDパネルなど)のほうで，仮想的なカーソルを移動させる，という処理が同時に進行することが必要です．

●より人間に近づいた入力手段は？

スイッチといえばONとOFFの2値を検出するものですが，人間の情報を対象とするならば，たとえばピアノ演奏の鍵盤操作(強弱)のような，連続した物理量をアナログ検出することも必要になります．現在のところでは，これを正直にセンサからA-D変換する方法よりも，ジョイスティックとかマウスなどの，アナログ量を疑似的にディジタル化する入力装置のほうがコスト・パフォーマンスが高いようです．

これを押し進めていくと，人間の「ボディ・ランゲージ」(身振りや動作)をシステムが検出するような，高度な入力方法が重要な要素となっていくでしょう．これには**図1.24**のように，

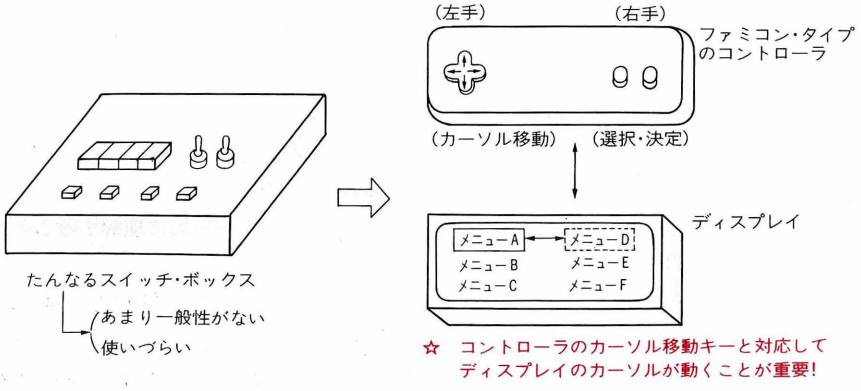
- 具体的に身体に各種のセンサを取り付ける方法
- CCDカメラや赤外線センサなどで，離れたところから検出する方法

などがあります．どれか1種類の高価な高精度センサを使おうとしても，人間の動作そのものがきわめてファジィですから，ローコストで低精度のセンサを複数種類，組み合わせる方法が有効でしょう．[チップ]の項にあるように，最近の1チップ・マイコンには，このような用途に最適の機能が内蔵されているものが多いので，機会があれば実験してみたいものです．

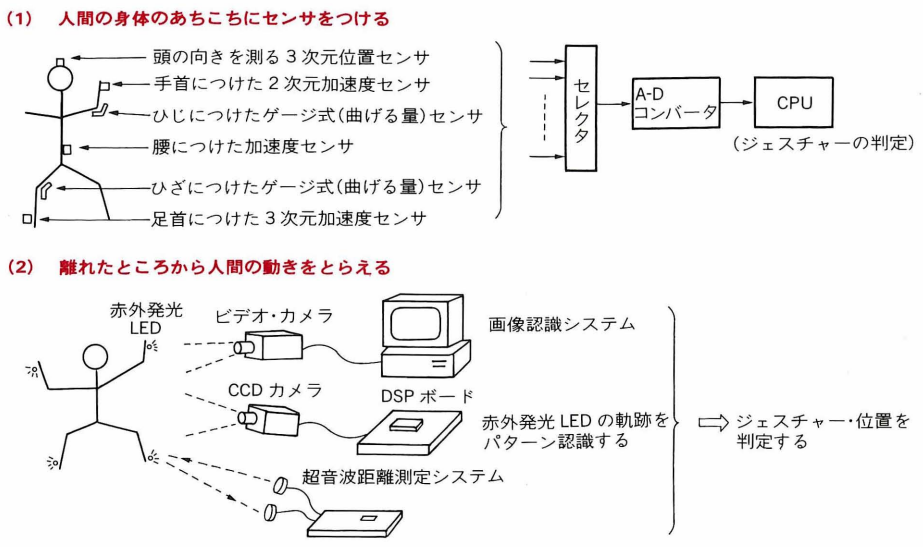
●システムの入力技術は永遠の課題

ここで技術的に問題となるのは，センサからの入力データの組み合わせから，人間のジェスチャーとしてシステムがその「意味を理解して判断」する，という点です．これは，パターン認識，意味理解，学習による成長，といったAI(人工知能)の重要課題と直結した

〔図1.23〕マイコン・システムの入力スイッチ例



〔図1.24〕人間の「ジェスチャー」を入力する方法



テーマで，多くの研究者が挑戦している先端の技術でもあります．フレッシュマン技術者としては，このアプローチに参加していく気概をもっていきたいものです．

また一方で，研究の歴史が長い入力手段の一つに「音声入力」もあります．これは「＊＊＊しなさい」とマイクに話し掛けると，システムがその音声メッセージを「認識」，「理解」，「判断」してくれる，というものです．技術的にはかなりのハードル(ほとんどが人間の発音の曖昧さに起因する)があって，なかなか汎用システムとしては難しい部分もあるようですが，いずれは入力方法のかなりの部分に活用されるでしょう．

そして究極のセンサといえば，「脳波とテレパシーのセンサ」です．機械が「以心伝心」するかどうかは疑問もありますが，未来を夢見るエンジニアとしては，頭のどこかに置いておきましょう．本当はそうでなくても，一見「人間の心がわかるような」システム，というブラックボックスを作って世間をアッといわせる，などというのはなかなか面白いテーマではないでしょうか．

●システムの出力手段：視覚的インターフェースの例

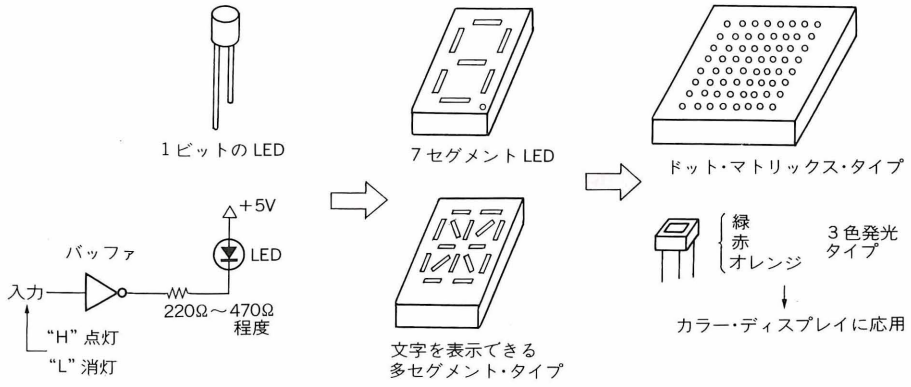
人間の五感の中では視覚がずば抜けて多量の情報を扱えるために，マイコン・システムの出力情報としては，**図1.25**のLED(発光ダイオード)，**図1.26**のLCD(液晶)などの低価格なデバイスが活用されています．また，パソコンの世界ではCRTがテキスト表示からグラフィック表示へ，モノクロからカラーへと，他の周辺装置よりも先に進歩していきました．昔は文字表示だけのゲームがあった，という歴史も，ファミコン世代には理解できないかもしれません．そしてコンピュータ自体の表示能力(解像度・描画速度)の向上によって，マウスやディジタイザ，タッチパネルやトラックボールによって画面上のポイントを指定して，対話的にインターフェースできるようなソフトへと，システムの動作そのものが進歩しています．

CPUパワーの向上と共に進んだソフトウェア技術としては，**図1.27**のように，それまでの「メニュー方式」ソフトから，階層の状態を視覚的に確認できる「ウィンドウ方式」，あるいは複数のプロセスを並行して走らせるマルチタスクへと進みました．このようないろいろな技術の進歩というのは，人間とコンピュータ・システムの間に置かれて，それぞれのスムースな活動を助けるための，マン-マシン・インターフェース技術の発展として，統一的に理解することができます．

●システムの出力技術も永遠の課題

この他の2次元ディスプレイとしては，ELパネルやプラズマ・ディスプレイがありますが，高性能とは

〔図1.25〕LED ディスプレイ

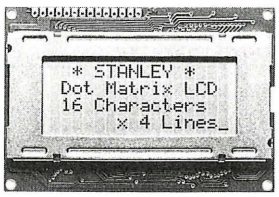


〔図1.26〕LCD ディスプレイの例(スタンレー電気)

GMD1620AL＊

GMD2420AL＊

GMD1640AL＊



●製品規格

単位：mm

品　名	表示容量(文字×行)	外形寸法(W×H×D)	有効表示寸法(W×H)	文字寸法(W×H)	ドット寸法(W×H)	Duty比	重量
GMD1610FL＊	16×1	80×36×15	64.5×13.8	3.11×5.77	0.59×0.79	1/16	40g
GMD1620AL＊	16×2	85×36×15	63.5×15.8	3.20×4.85	0.60×0.65	1/16	40g
GMD1640AL＊	16×4	87×60×15	61.8×25.2	2.95×4.15	0.55×0.55	1/16	60g
GMD2020AL＊	20×2	115×36×16	83.0×18.6	3.20×4.85	0.60×0.65	1/16	60g
GMD2040AL＊	20×4	98×60×15	76.0×25.2	2.95×4.15	0.55×0.55	1/16	70g
GMD2420AL＊	24×2	118×36×15	93.5×15.8	3.20×4.85	0.60×0.65	1/16	65g
GMD4020AL＊	40×2	182×33.5×16	154.5×15.8	3.20×4.85	0.60×0.65	1/16	100g

いえちょっとコストが高いものです．また，液晶シャッターを両目の視差を考慮して表示するHMD(ヘッド・マウント・ディスプレイ)の3次元表示も，まだまだこれからの技術です．

また，音声合成とか音響合成などの聴覚的な情報出力も，視覚表示の情報量に比べるとかなり少ないために，ちょっと単独では使えないようです．音楽演奏はできても，「再生」でなく「歌う」電子楽器がなかなか登場しないように，音声合成技術は発展途上段階ですから，しばらくの間は，いろいろな入力手段と組み合わせた上で，まだ視覚的な情報中心の時代が続くのかもしれません．

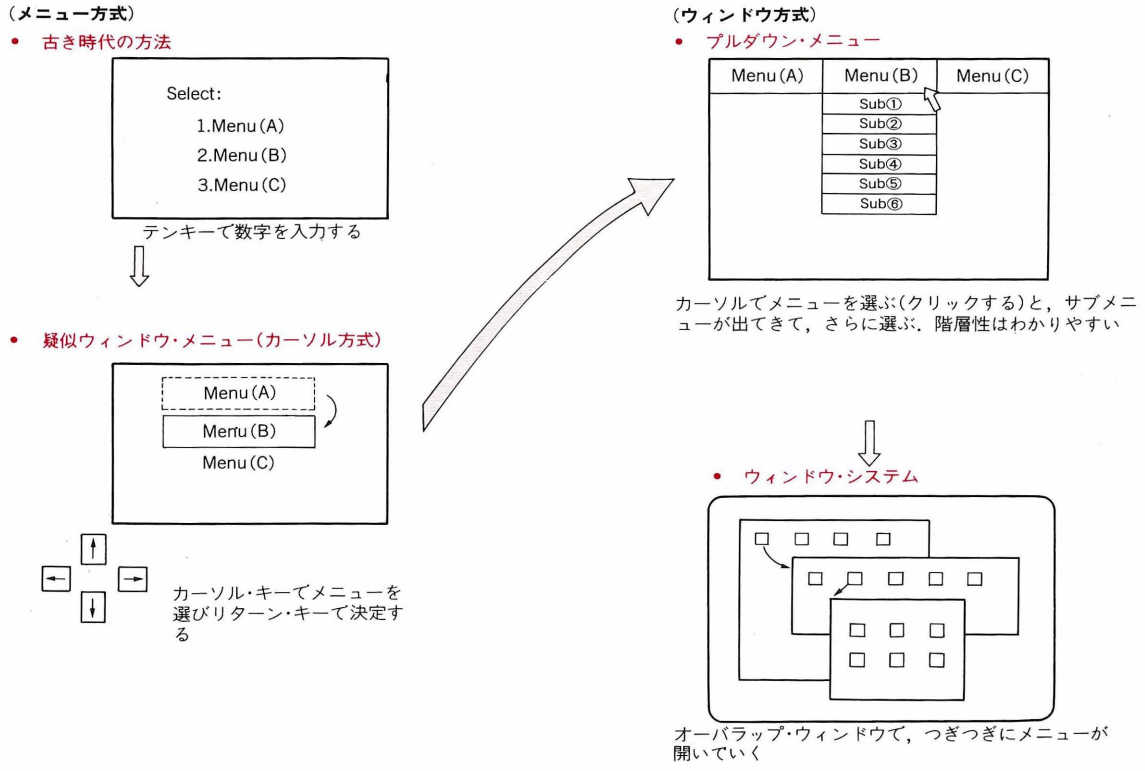
なお，たとえばデータ・グローブで動作を入力するのに対して，グローブの「手応え」をシステムから人間に物理的に返す，という「フォース・ディスプレイ」という情報出力技術もあります(**図1.28**)．これによって，人間は仮想的な物体の「質感」を実感するわけです．ゲームセンターのコックピットが動くものなども，同様の臨場感として有効でしょう．しかし，「頭に電極を突き刺して直接に脳に情報を伝える」「人間にテレパシーでメッセージを送る」というところまでは，たぶん行かないだろうと思います．

●ヒューマン・インターフェースとしてのバーチャル・リアリティ技術

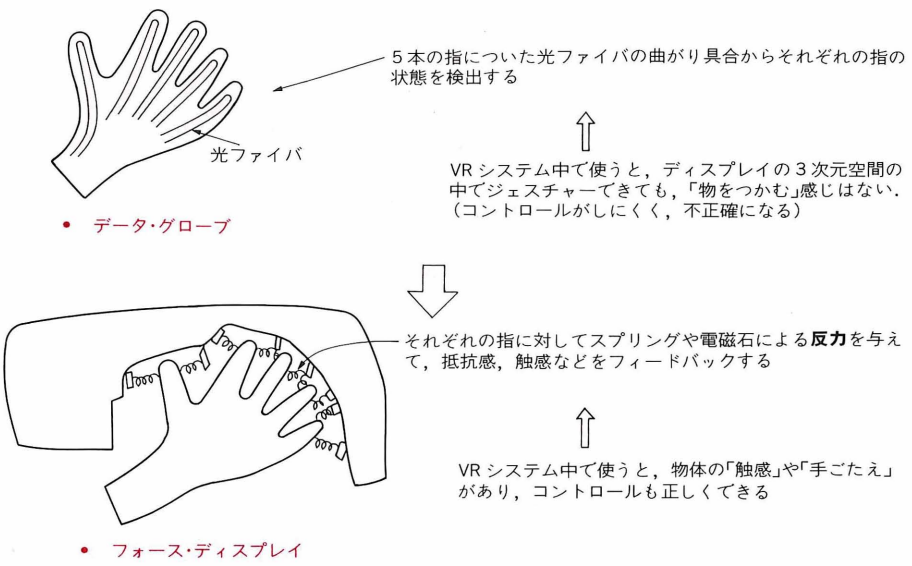
最後に，マイコン・システムと人間との接点(マン-マシン・インターフェース)の重要技術である，「ヒューマン・インターフェース」について考えてみましょう．マルチメディア技術の延長上にあるものとして，「仮想現実感」(バーチャル・リアリティ：VR，またはアーティフィシャル・リアリティ：AR)が流行していますが，これはまさに，ヒューマン・インターフェース技術の理想を追求しているのです．

この技術を検討するために，**図1.29**のように，マイコン・システムを，人間と「感覚される空間」との間に置いた，仮想的なインターフェースと想定してみます．人間は機械的なマイコン・システムを操作するのでなく，より自然なかたちで，この仮想現実空間に働きかけるのです．システム側ではこのインターフェースから必要な情報を抽出して，機械の形式の入力情報とします．また，マイコン・システムからの出力についても，機械の語法のままでは人間にとって理解が困難ですから，この仮想インターフェースが人間に優しいかたちに変換して提供することになります．このようなシステムは，まだまだ高価なセンサやEWSを駆使した大がかりなものですが，発想の部分については，マイコン・システムでもおおいに参考となっていくでしょう．

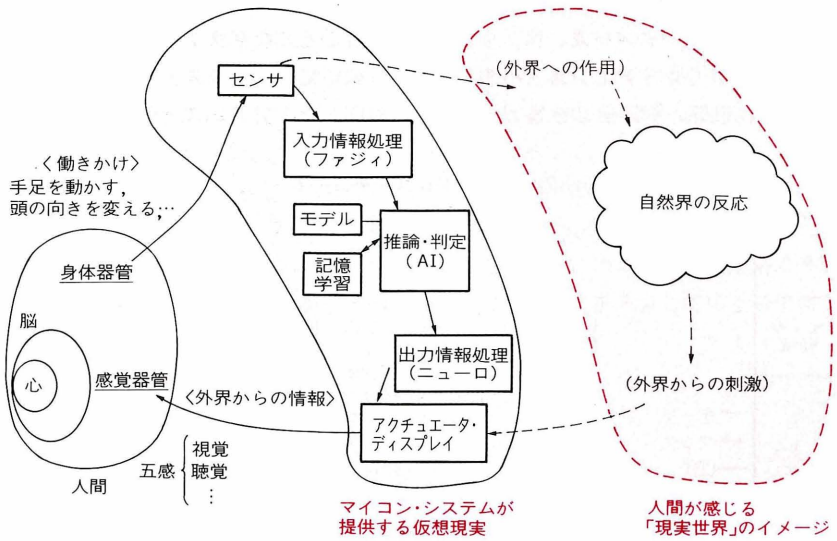
〔図1.27〕ソフトウェアのユーザ・インターフェースの進歩



〔図1.28〕フォース・ディスプレイ



〔図1.29〕マン-マシン・インターフェースとしての仮想現実感(バーチャル・リアリティ)



2

パソコン活用システム関連技術

ハードウェア技術の基本

ここでは，パソコンの拡張スロットに外部インターフェース用のボードを拡張していくための，もっとも基本的なハードウェア技術を確認していきます．

●アドレス・デコード

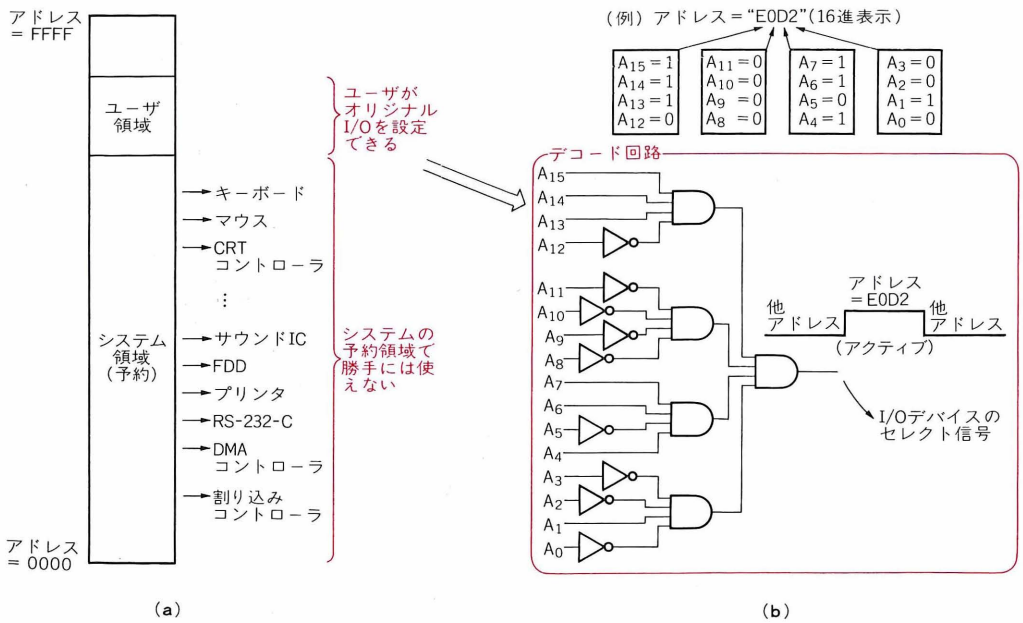
CPUチップ自身が外界とやりとりするための基本となるのは，

- 情報じたいをやりとりする**データ・バス**
- そのデータがどこにあるかを示す**アドレス・バス**

の2種類のバスです(それ以外の制御信号をまとめて「**コントロール・バス**」と呼ぶ場合もある)．そしてこのバスには，各種のメモリとか周辺LSIなどが接続されますから，これらのチップをそれぞれ区別して制御しなければなりません．そのための最初の準備として，**図2.1**(a)のように，CPUのアドレス空間(CPUによっては，メモリ空間とI/O空間とがある)に周辺LSIなどを配置した「アドレス・マップ」が規定されます．パソコンの場合には，最初からメーカによって予約されているアドレスが多いので，ユーザとして新たに周辺を増設する場合には，メーカから自由エリアとして提供されている「ユーザ領域」を増設周辺用に割り当てることになります．

続いて，アドレス・マップ上で決定したアドレスに対応した，アドレス・デコード回路が必要になります．

〔図2.1〕アドレス・デコード



これは基本的には**図2.1**(b)のように，アドレス・バスの信号から論理積を作って，該当する周辺チップにだけアクティブ信号(データ・バスの使用権，あるいはデータ・バス上の信号がそのチップに対するものであるというサイン)を送る，という考え方です．したがって，二つ以上の周辺チップが同じアドレスでアクティブにならないように，正しくアドレス・デコード回路を設計することが必要となります．周辺チップは一つだけのアドレスを占有するとは限らない(ある一定幅の領域をもつ)場合も多いので，デコード回路の設計は慎重に行います．

デコード回路を節約するために，実際に使用する範囲よりも広いアドレスでアクティブになるような回路とする場合も多い(このときの，実際には使わないアドレス領域を「ゴースト」とか「シャドー・エリア」という)のですが，あとでシステムの周辺を増設したような場合に，思わぬアドレス衝突のトラブルの原因となることもありますから，設計完了時点での最終的なアドレス・マップをきちんと記録しておきましょう．

●出力ポート(ラッチ出力)

パソコンから外部に信号を出力する場合，CPUのデータ・バス上の信号を確定させて外部に取り出すために，**図2.2**のようなラッチが必要です(これをラッチ出力という)．普通は174や374のような，エッジトリガ・タイプのDラッチを使用しますが，もしここに373のようなGラッチ(トランスペアレント・ラッチ)を使うと，ラッチ・パルス(ライト・パルス)がアクティブであるデータの切り替え時に，CPUの特性やデータの遅れによってはデータが変動する(不定)可能性があります．

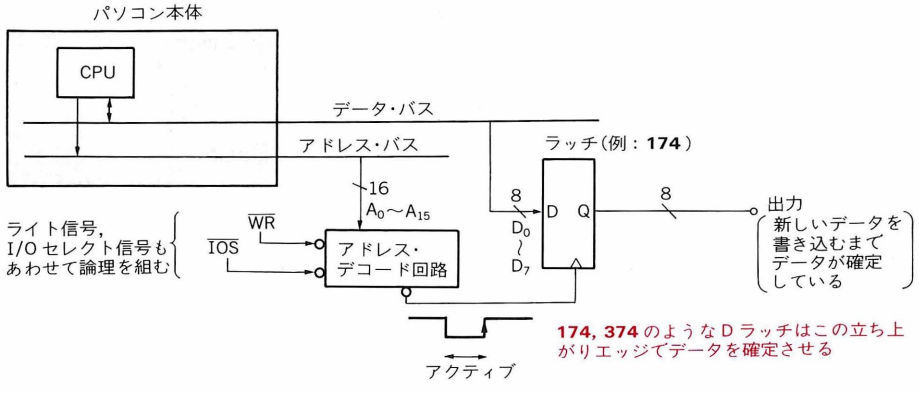
この図のような回路だけでは，スイッチのON/OFFのような単純な信号しか出ないように思うかもしれませんが，パソコンの処理スピードというのは相当なものですから，このようなラッチをON/OFFする繰り返しをスピーカから鳴らすだけで，たとえば楽器のようにメロディ(ピッチ)をつけることだってできるのです．また，ラッチの出力側に抵抗ネットワークをつけて，各ビットの出力の組み合わせに対応した一定の電圧が得られるようにすると，これは簡単なD-Aコンバータとなります．そこに，パソコンから刻々と変化するデータを送ると，かなりの音質(電話の音声よりは高品質)のPCM音声データを再生することもできます．このように，もっとも基本的なラッチ出力だけで，かなりのインターフェースが実現できるのです．

●入力ポート(3ステート入力)

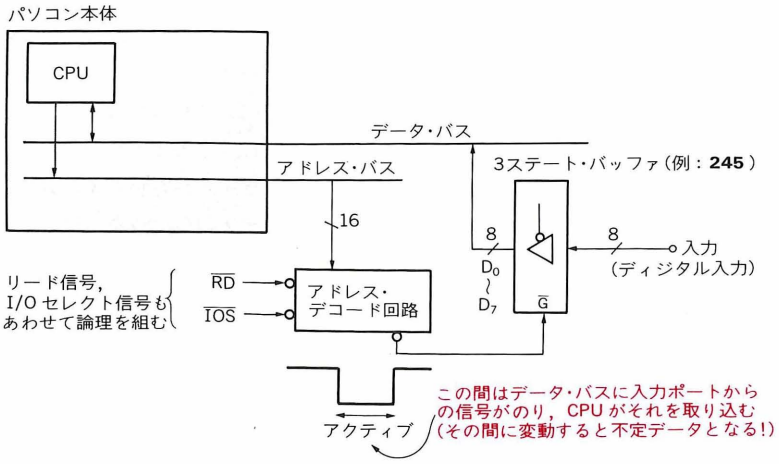
パソコンに外部から信号を入力する場合，CPUデータ・バスに外部からのディジタル信号を取り込むわけですが，このときは**図2.3**のような3ステート・バッファを使用します(これを3ステート入力という)．これには，普通は245や367などが使われます．データ・バスは普通はCPUなど他のチップが使っていますから，アドレス・デコード回路によってイネーブルされたときだけ出力するようにして，バス上の信号の衝突(バス・ファイト)を避けるわけです．

ただし，このような入力回路で外界の信号を取り込むと，データを入力している最中にそのデータが変化

〔図2.2〕パソコンからの出力ポートの例



〔図2.3〕パソコンへの3ステート入力ポートの例



した場合，CPUには不定データが与えられることがありますから，この方法は，静的なデータを取り込むための簡易的なものといえます．たとえば，スイッチ・パネルをこの方法で検出する場合には，接点がジャンプしている不定状態をたまたま入力してしまう確率もあるために，2回とか3回の状態検出結果が同じであるときに，はじめてその状態を入力として採用する，というようなソフトウェア(チャタリング防止)と組み合わせた方法をとります．

さらに，もっと変動するデータを取り込む場合には，パソコンにデータを与える外部機器の側でいったんラッチでデータを確定させ，さらに後述する「ハンドシェイク」(相互に確認して転送)などの方法によって，データを与えることが必要となります．

ソフトウェア技術の基本

●フローチャート

たとえば**図2.4**のようにパソコンを使って温度計測を行うシステムを例にとると，そのソフトウェア設計はまず，**図2.5**のようなフローチャートを考えることから始めることになります．この例では，左側の列に基本的な処理を並べて，さらに右側にはオプション的な処理を並べてあります．

このようなシステムの制御ソフトは，「基本処理」自体は意外に簡単に記述できるものなのですが，それ以外の部分，つまり

- 使用するユーザに状況を知らせる
- ユーザからの操作をやさしくする
- 本来の基本動作以外の例外的処理
- エラー・メッセージとエラー処理
- 確実なシステムの立ち上げとシャットダウン
- 動作状況の記録と再現
- 開発者のデバッグ作業のための隠し機能

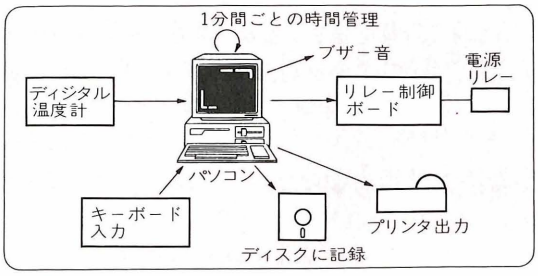
などのために，プロの仕事の場合，極端なときはソフトウェアの90％が占められるようなケースさえあります．

最近は，このようにフローチャートを手で描かなくても，対話的に簡単に作成できる「支援ツール」という一種のCADソフトウェアがあります．そして，作成したフローチャートの論理的な矛盾を検証・指摘してくれるものもあります．じつは正直なところ，筆者はほとんどフローチャートを描かないでプログラミングする，という悪い習慣をもっているのですが，これからのエンジニアは，このような支援環境を活用して高度なソフトウェアを構築していくことになると思います．

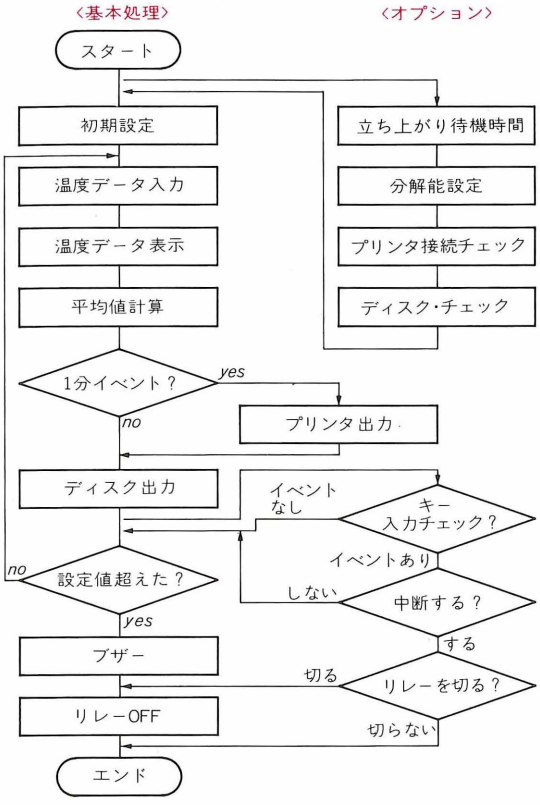
●使いやすいBasic

最近ではパソコンでBasicを使う人も少なくなったようですが，パソコンに付属してきた歴史もあり，実験室ではなかなか活躍しています．**リスト2.1**はさきの温度計測システムの場合のプログラム例ですが，このような簡単なものであれば，フローチャートとか紙の上での設計を抜きにして，いきなりパソコンに向か

〔図2.4〕パソコン応用の例――温度計測システム



〔図2.5〕処理プログラムのフローチャート例



〔リスト2.1〕パソコンによる温度計測システムのプログラム例

```
1000 'save "sample",a
1010    TEMP=&HE0D0:POWER=&HE0D2          'Port Address
1020    OUT POWER,&HFF
1030    OPEN "c:test.dat" FOR OUTPUT AS #1
1040    LIMIT=50
1050    SUM=0:I=0
1060    OLD$=LEFT$(TIME$,5)
1070 '##### Main Loop #####
1080    X1=INP(TEMP)
1090    X2=INP(TEMP)
1100    IF(X1<>X2) THEN GOTO 1080 ELSE PRINT X1;
1110    SUM=SUM+X1:I=I+1
1120    MEAN=INT(10*SUM/I)/10
1130    T$=LEFT$(TIME$,5)
1140    IF(T$<>OLD$) THEN GOSUB 1200      'Print Out
1150    WRITE #1,X1
1160    IF X1=<LIMIT THEN 1070
1170    OUT POWER,0
1180    BEEP:BEEP:BEEP:BEEP:BEEP
1190    CLOSE #1:END
1200 '##### Printer Output Routine #####
1210    LPRINT "Time = ";T$;"   ,   Temp =";MEAN
1220    OLD$=T$
1230    RETURN
```

〔リスト2.2〕機械語ルーチンの活用で高速化する(サブルーチンとして呼び出す例)

```
1000 'save "sample",a
1010    DEF SEG=&H8000              ' 機械語ルーチン領域の確保
1020    BLOAD "b:subrt.bin",0       ' 機械語ルーチンをLoad
1030    SUBRT=0                     ' 機械語ルーチン先頭Address
1040    TEMP=&HE0D0                 ' Port Address
1050 '##### Main Loop #####
1060    X1=INP(TEMP)                ' Data Input
1070    X2=INP(TEMP)
1080    IF(X1<>X2) THEN GOTO 1060 ' Normal Data ?
1090    CALL SUBRT(X1)              ' 機械語ルーチンをCall
1100    GOTO 1050
```

ってプログラミングしていけます.

Basic の最大の長所は,

●すぐに手直ししてテスト・ランできる

●バグがあってもエラー・メッセージが出て止まる(暴走しない)

という2点でしょう. つまり,「インタプリタ」という環境の中(お釈迦様の掌の上みたいなもの)で保護されているわけで, フレッシュマンがとにかくいろいろ実験してみるためのプラットフォームとしては, おすすめできると思います.

● Basic の高速化

Basic の欠点としてよくいわれるのは, その「遅さ」です. 最近のパソコンは非常に高速になったので, あまり目立ってこなくなってしまったのですが, 旧世代のパソコンでは顕著な問題点でした. そこで, このための対策として,

●機械語ルーチンの利用

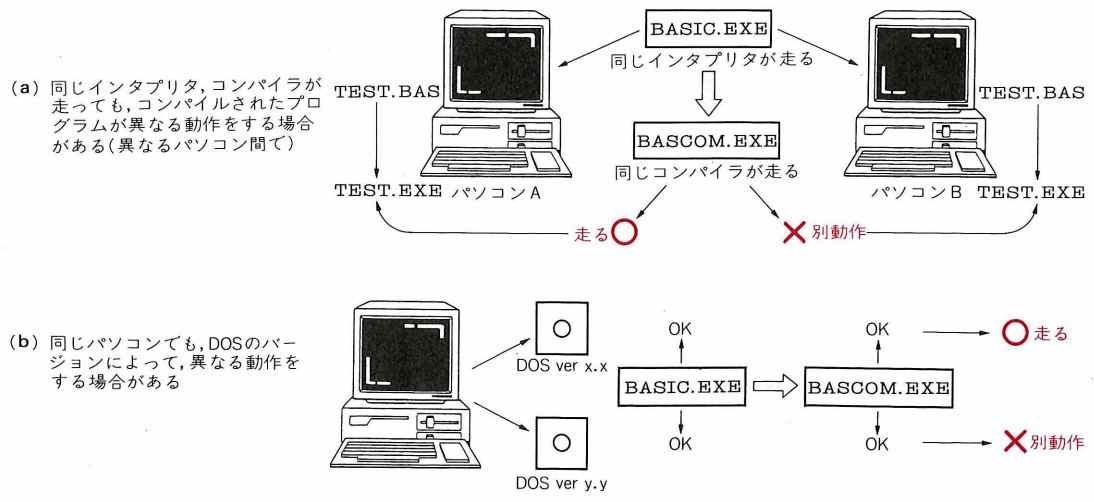
● Basic コンパイラの利用

というテクニックが重要なものとされました.

機械語ルーチンというのは, あらかじめアセンブラで作成したルーチンや, すでに完成しているルーチンなどの, マシン語による高速の処理を行うモジュールを, Basic のプログラム中から呼び出して, その処理部分については高速に実行するようにしたものです. **リスト2.2**の例はサブルーチン方式によるもので, Basic のプログラムとしてまずメモリ・エリアを確保して, そこに機械語ルーチンを読み込み, プログラム中のサブルーチンとしてコールしています. この他には, 機械語ルーチンを一種の関数として定義して呼び出す方法もあります.

Basic コンパイラというのは, Basic のプログラム

〔図2.6〕Basic コンパイラの注意点



をソース・プログラムとして，C言語と同様のコンパイラによってマシン語プログラムに変換するものです．今ほどC言語が普及していなかった時代には重宝したもので，**リスト2.3**のような手順で活用されました．ところが，現在のC言語ほど標準化が考えられていなかったために，

- DOSベースの同じインタプリタとコンパイラが走る異種のパソコン間で，コンパイルされた実行プログラムが異なった動作をする
- 同じパソコンでもDOSのバージョンによって異なった動作をする

といった問題点に泣かされたものです(**図2.6**)．

現在のフレッシュマンは，Basicコンパイラの世界に深入りする必要はなくなっていますから，自然にC言語に移行することをおすすめします．

● C言語

さて，ソフトウェアの基本的技術の最後は，C言語についてです．これはいまさら詳細にここで述べることもありませんから，**リスト2.4**をちらっとだけ眺めてください．ちなみに，このリストのCコンパイラは，もう現在では消えてしまった「あるマイナーなC」です(I/O操作の関数名で，コンパイラがわかる？)．

Unixという強力なOSを記述していることで明らかなように，C言語は大規模なソフトウェア・システムを構築する上で，またCPUにかなり近いレベルの動作を記述するためにも，非常に有効な言語です．この証明として，ソフトウェア製品の多くがCで書かれています(Cコンパイラ自体もCで開発されている)(**リスト2.5**)．ここに示したリスト例は，有名なソフトのいくつかについて，プログラム本体のダンプ・リストの一部を抜き出したものですが，いずれも，Cコンパイラのコピーライト・メッセージが残されています．

まさに，C言語は「プロのツール」なのです．また，最近の「オブジェクト指向」という技術に対応して，Cの機能拡張版として[C++]という言語も登場してきています．

〔リスト2.3〕Basicコンパイラ活用の手順

```
>basic

load "sample"
Ok
list
1000  'save "sample",a
1010    PRINT:PRINT " Start Time = ";TIME$
1020    FOR I=1 TO 10000
1030        J=SIN(I)
1040    NEXT I
1050    PRINT:PRINT "Finish Time = ";TIME$:PRINT
1060    IF INKEY$="" THEN 1060
Ok
run

 Start Time = 18:30:05

Finish Time = 18:30:59

Ok
system

>basicc

Personal Computer BASIC Compiler Version 1.0

Source Filename [.BAS]: sample
Object Filename [SAMPLE.OBJ]:
Source Listing [NUL.LST]:

20138 Bytes Available
19812 Bytes Free

    0 Warning Error(s)
    0 Severe  Error(s)

>link

Personal Computer Linker Version 2.4

Object Modules [.OBJ]: sample
Run File [SAMPLE.EXE]:
Source Listing [NUL.MAP]:
Libraries [.LIB]:

>sample

 Start Time = 18:36:19

Finish Time = 18:36:29

>
```

〔リスト2.4〕図2.5の例をC言語で書いてみる(リストのBasicプログラムと完全コンパチではない)

```
#include  <stdio.h>
#define   temp_port     0xe0d0
#define   power_port    0xe0d2
extern    unsigned _rax,_rcx,_rdx;

main(){
        char c;
        int fd,x1,x2;
        int limit=50;
        long sum=0L;
        int i=0;
        int j,k=0;
        unsigned old,t;
        float mean;

        _outb(0xff,power_port);
        fd=fopen("f:test.dat","w");
        old=time_check();
        for(;;){
                for(j=0;j++;j<100) {k++;}
                c=csts();
                if(c!=0){
                        fclose(fd);
                        exit();
                }
                x1=_inb(temp_port);
                x2=_inb(temp_port);
                if(x1==x2){
                        sum=sum+(long)x1;
                        i++;
                        mean=(float)sum/(float)i;
                        t=time_check();
                        if(old!=t){
                                old=t;
                                printf("¥nTime = %2d:%2d",t/256,t%256);
                                printf(" , Temp = %f",mean);
                                i=0;
                                sum=0L;
                        }
                        if((i%50)==0){
                                putw(x1,fd);
                        }
                        if(x1>limit){
                                putchar(0x07);
                                putchar(0x07);
                                putchar(0x07);
                                putchar(0x07);
                                putchar(0x07);
                                _outb(0x00,power_port);
                                fclose(fd);
                                exit();
                        }
                }
        }
}

time_check(){
        _rax=0x2c00;
        _doint(0x21);
        return(_rcx);
}
```

〔リスト2.5〕市販ソフトはC言語で書かれたものが多い①

（例）ワープロ「一太郎」（ジャストシステム）

```
>dump jxw.exe

0001EDB0  C3 FF 0F EB BE B0 03 0B-F6 74 02 FE C0 F9 C3 43   ....ｰ.....ﾀ..C
0001EDC0  6F 70 79 72 69 67 68 74-20 28 43 29 20 31 39 38   opyright (C) 198
0001EDD0  34 2C 20 31 39 38 35 2C-20 31 39 38 36 20 62 79   4, 1985, 1986 by
0001EDE0  20 50 68 6F 65 6E 69 78-20 53 6F 66 74 77 61 72    Phoenix Softwar
0001EDF0  65 20 41 73 73 6F 63 69-61 74 65 73 20 4C 74 64   e Associates Ltd

00078770  00 00 00 00 00 00 00 00-43 20 4C 69 62 72 61 72   ........C Librar
00078780  79 20 2D 20 28 43 29 43-6F 70 79 72 69 67 68 74   y - (C)Copyright
00078790  20 4D 69 63 72 6F 73 6F-66 74 20 43 6F 72 70 20    Microsoft Corp
000787A0  31 39 38 35 00 01 65 72-72 6F 72 20 32 30 30 31   1985..error 2001
```

（例）イーサネット通信ソフト「ＴＥＬＮＥＴ」（コンテック）

```
>dump telnet.exe

00011B80  00 00 00 00 00 00 00 00-43 20 4C 69 62 72 61 72   ........C Librar
00011B90  79 20 2D 20 28 43 29 43-6F 70 79 72 69 67 68 74   y - (C)Copyright
00011BA0  20 4D 69 63 72 6F 73 6F-66 74 20 43 6F 72 70 20    Microsoft Corp
00011BB0  31 39 38 35 00 01 65 72-72 6F 72 20 32 30 30 31   1985..error 2001

00011BD0  73 73 69 67 6E 6D 65 6E-74 0D 0A 00 0A 54 45 4C   ssignment....TEL
00011BE0  4E 45 54 28 46 41 2D 4C-41 4E 33 29 89 BC 91 7A   NET(FA-LAN3)仮想
00011BF0  83 5E 81 5C 83 7E 83 69-83 8B 20 20 20 20 20 20   ターミナル
00011C00  CA DE 2D BC DE AE DD 20-32 2E 33 31 20 20 42 79   ﾊﾞｰｼﾞｮﾝ 2.31  By
00011C10  20 43 4F 4E 54 45 43 20-43 6F 2E 2C 4C 74 64 2E    CONTEC Co.,Ltd.
```

（例）マクロアセンブラ「ＭＡＳＭ」（マイクロソフト）

```
>dump masm.exe

00019830  67 65 00 FF FF FF F0 15-B2 00 18 07 B0 4D 53 20   ge......ｲ...-MS
00019840  52 75 6E 2D 54 69 6D 65-20 4C 69 62 72 61 72 79   Run-Time Library
00019850  20 2D 20 43 6F 70 79 72-69 67 68 74 20 28 63 29    - Copyright (c)
00019860  20 31 39 38 38 2C 20 4D-69 63 72 6F 73 6F 66 74    1988, Microsoft
00019870  20 43 6F 72 70 11 00 00-00 00 00 00 00 70 74 72    Corp........ptr

0001A710  73 0A 43 6F 70 79 72 69-67 68 74 20 28 43 29 20   s.Copyright (C)
0001A720  4D 69 63 72 6F 73 6F 66-74 20 43 6F 72 70 20 31   Microsoft Corp 1
0001A730  39 38 31 2C 20 31 39 38-38 2E 20 20 41 6C 6C 20   981, 1988.  All
0001A740  72 69 67 68 74 73 20 72-65 73 65 72 76 65 64 2E   rights reserved.
```

（例）汎用エディタ「Ｂｒｉｅｆ」（日本システム）

```
>dump b.exe

0001B2A0  00 00 00 00 00 00 00 00-4D 53 20 52 75 6E 2D 54   ........MS Run-T
0001B2B0  69 6D 65 20 4C 69 62 72-61 72 79 20 2D 20 43 6F   ime Library - Co
0001B2C0  70 79 72 69 67 68 74 20-28 63 29 20 31 39 39 30   pyright (c) 1990
0001B2D0  2C 20 4D 69 63 72 6F 73-6F 66 74 20 43 6F 72 70   , Microsoft Corp

0001BBF0  00 00 F2 08 F6 08 FC 08-42 52 49 45 46 20 56 33   ........BRIEF V3
0001BC00  2E 30 20 4A 52 31 2E 30-33 20 53 44 43 20 53 6F   .0 JR1.03 SDC So
0001BC10  66 74 77 61 72 65 20 50-61 72 74 6E 65 72 73 20   ftware Partners
0001BC20  49 49 2C 20 4C 2E 50 00-43 6F 70 79 72 69 67 68   II, L.P.Copyrigh
0001BC30  74 20 28 43 29 20 31 39-39 30 2D 31 39 39 31 20   t (C) 1990-1991
0001BC40  53 44 43 20 53 6F 66 74-77 61 72 65 20 50 61 72   SDC Software Par
0001BC50  74 6E 65 72 73 20 49 49-2C 20 4C 2E 50 2E 20 41   tners II, L.P. A
0001BC60  6C 6C 20 72 69 67 68 74-73 20 72 65 73 65 72 76   ll rights reserv
0001BC70  65 64 00 95 CF 8D 58 95-94 95 AA 82 CD 94 6A 8A   ed.変更部分は破棄
```

〔リスト2.5〕市販ソフトはC言語で書かれたものが多い②

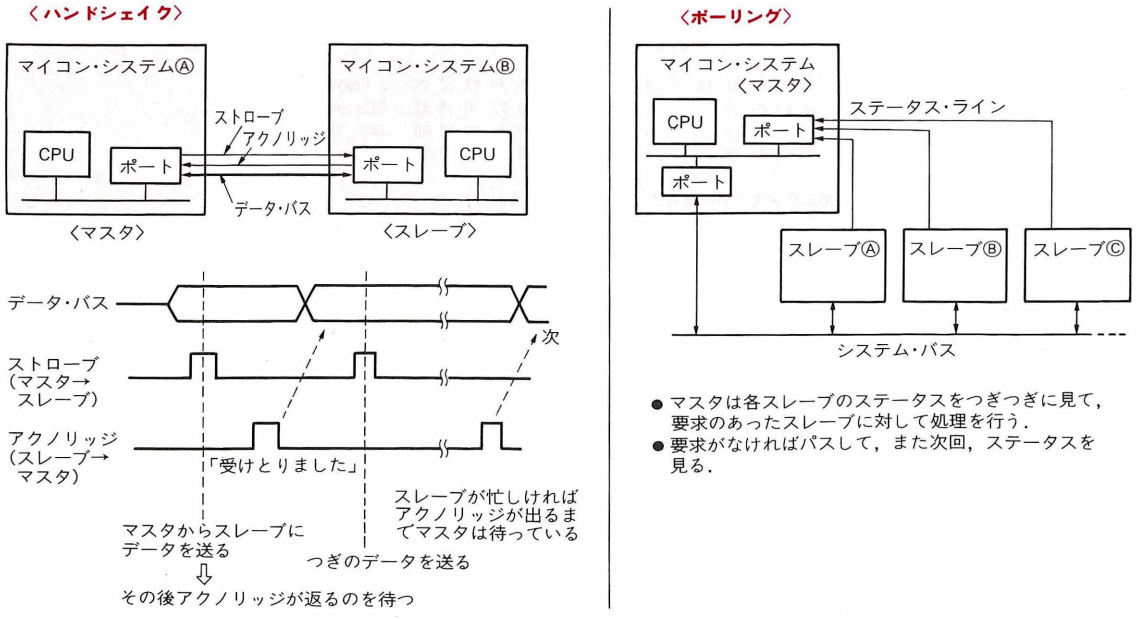
（例）実行ファイル圧縮ソフト「瞬間ＡＸＥ」（マイクロデータ）

```
>dump axe.exe

00004EF0  00 00 00 00 00 00 00 00-4D 53 20 52 75 6E 2D 54   ........MS Run-T
00004F00  69 6D 65 20 4C 69 62 72-61 72 79 20 2D 20 43 6F   ime Library - Co
00004F10  70 79 72 69 67 68 74 20-28 63 29 20 31 39 38 38   pyright (c) 1988
00004F20  2C 20 4D 69 63 72 6F 73-6F 66 74 20 43 6F 72 70   , Microsoft Corp

000066B0  3B 37 6D 20 20 20 8F 75-8A D4 82 60 82 77 82 64   ;7m   瞬間ＡＸＥ
000066C0  20 20 56 65 72 73 69 6F-6E 20 31 2E 32 31 20 20     Version 1.21
000066D0  1B 5B 33 32 3B 37 6D 20-20 20 43 6F 70 79 72 69   .[32;7m   Copyri
000066E0  67 68 74 28 43 29 20 31-39 39 30 20 20 62 79 20   ght(C) 1990  by
000066F0  53 45 41 20 2F 20 4D 49-43 52 4F 20 44 41 54 41   SEA / MICRO DATA
```

（例）高速ファイル転送ソフト「ＭＡＸＬＩＮＫ」（メガソフト）

```
>dump maxlink.exe

00003580  C3 21 23 B2 00 F0 FF B0-00 19 00 B0 43 20 4C 69   ﾃ!#ｲ...ｰ...-C Li
00003590  62 72 61 72 79 20 2D 20-28 43 29 43 6F 70 79 72   brary - (C)Copyr
000035A0  69 67 68 74 20 4D 69 63-72 6F 73 6F 66 74 20 43   ight Microsoft C
000035B0  6F 72 70 20 31 39 38 36-1F 00 82 6C 82 60 82 77   orp 1986..ＭＡＸ
000035C0  82 6B 82 68 82 6D 82 6A-81 69 85 50 85 4D 85 4F   ＬＩＮＫ（１．０
000035D0  85 58 94 C5 81 6A 20 85-62 85 8F 85 90 85 99 85   ９版）Ｃｏｐｙｒ
000035E0  92 85 89 85 87 85 88 85-94 85 47 85 62 85 48 20   ｉｇｈｔ（Ｃ）
000035F0  85 50 85 58 85 57 85 57-85 4C 85 58 85 4F 20 85   １９８８－９０Ｍ
00003600  6C 85 64 85 66 85 60 85-72 85 6E 85 65 85 73 20   ＥＧＡＳＯＦＴ
```

〔図2.7〕マイコン・システム間のインターフェース方式の例



- マスタは各スレーブのステータスをつぎつぎに見て，要求のあったスレーブに対して処理を行う．
- 要求がなければパスして，また次回，ステータスを見る．

外部システムとのインターフェース技術

ここでは，パソコンと拡張ボードによるシステムで，外部とインターフェースしていく方法について考えていきましょう．もっともこの技術はパソコンに限らず，ボード・マイコンやオリジナル・ボードによるシステムにも共通のものですから，マイコン技術者にとって必須の内容です．

パソコンの外部入出力インターフェース回路として，ラッチによる出力と，3ステート・バッファによる入力についてはすでに述べました．もちろん，電気的にはこの方法は，あらゆる外部インターフェースの基本です．しかし，マイコン・システムはCPUのソフトとともにありますから，このインターフェースが「時間的」に，すなわちCPUの動作とどう関係してくるか，が重要になります．

ここで取り上げるのは，一般のマイコン・システム同士のインターフェース技術でもっともポピュラな，「ハンドシェイク」と「ポーリング」です(**図2.7**)．

●ハンドシェイク

ハンドシェイクというのはその名前の通り，二つのシステムがお互いに手をつないだように，相手の状態をチェックして自分の動作を決定する，というものです．たとえばシステムAがシステムBにいくつかのデータを続けて送りたい場合に，

(1) システムAがまず最初のデータを信号ラインに乗せる

(2) つぎに，「データを書きました」という信号(ストローブ信号)をバスに送る

という動作だけでは，いつシステムBがそのデータをちゃんと読み込んで，つぎのデータを受け入れ可能であるか，わかりません．ここで「確実にデータが受け取られるであろう時間だけ，足踏みして待つ」というのでは，全体のデータ転送が遅くなってしまいます．そこでハンドシェイク方式では，

(3) システムBがデータ受け取り完了の信号(アクノリッジ信号)を送る

(4) システムAはこのアクノリッジを確認したら，つぎのデータを送る

(5) 以下，(2)に続く

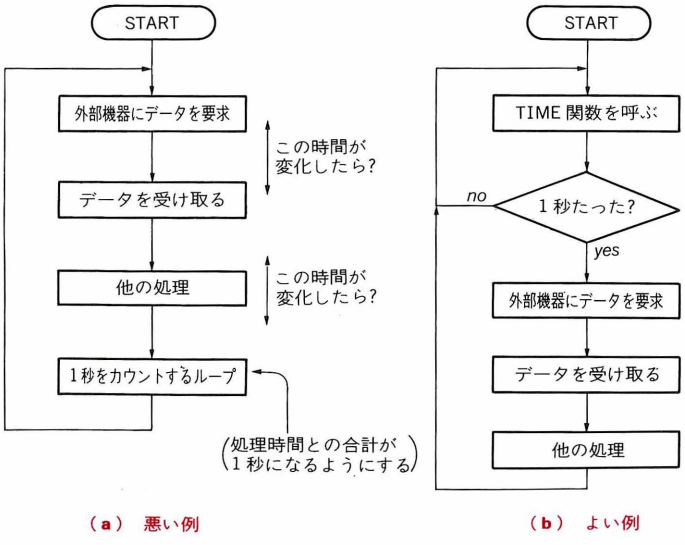
というようなやりとりを行います．これによって，確実にデータが送られ，さらに無駄な時間カウントなども不要となるわけです．

●ポーリング

ポーリングという方式は，

- データの送り側というよりも，むしろデータの受け側
- 複数のスレーブ装置を相手にして，つぎつぎにデー

〔図2.8〕ポーリングの必要性



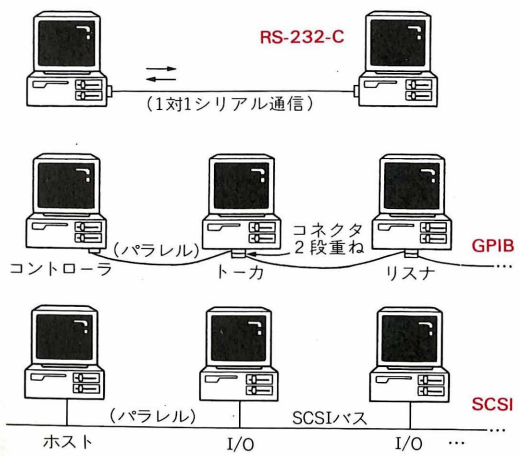
タ交換を行うマスタ装置

などの場合に必要な方式です．システムは自分の仕事をしながら，ときどき(ヒマなときに)周辺装置の指定された信号ライン(ステータス・ライン)をチェックします．そして，もし必要な要求信号が出ていたときには，それぞれ所定の対応処理を実行するというものです．

ここでは，ステータス・ラインのチェックを頻繁に行えば，相手のシステムをあまり待たせない，応答性のよいシステムとなりますが，その分自分の処理が遅くなる，という一種のトレードオフが発生します．逆にいえば，ハンドシェイクよりも簡易的なインターフェースであるために，情報密度(イベント密度)が高かったり，処理速度を重視したような要求の場合には，一定の限界が現れる可能性もあるのです．

どんなパソコンにもセントロニクス仕様のプリンタ・インターフェースがついていますが，パソコンからこのコネクタへのデータ転送は，通常はハンドシェイクで実行されます．また，相手側となるプリンタ内部のCPUシステムは，たいていの場合，ポーリングによってパソコンからの指令を取り込んでいます．遅いプリンタにパソコンが待たされるとか，パソコンからの中止命令をプリンタが受け付けるのは1ラインを打ち出してから，というのは，まさにこの事実を証明しているのです．

〔図2.9〕標準インターフェースの例



●ポーリングの例

パソコンがポーリングによって外部とインターフェースする簡単な例として，「1秒ごとに外部機器のデータを取り込む」事例を，**図2.8**のフローチャートを使って考えてみましょう．ここではパソコンから外部機器に「データ要求」の信号を出すと，外部機器からデータが返されるハンドシェイクを行っているものとします．

まず(a)は「悪い例」なのですが，

- 外部機器からデータを受け取る
- そのデータを処理する
- [1秒]－[上の処理時間]をカウントする(たとえばダミー・ループを回る)

というような流れです．これは一見正しい処理のように見えますが，外部機器からのデータの返答が遅れたり，データの内容によって処理ルーチンの処理時間が変化すると，「1秒ごとに」という動作が大きく狂ってきてしまいます．また，1秒という，パソコンにとっては非常に長い時間(おそらく何万ステップもの動作)を，ただ無駄にループしていて他の仕事をできない，という処理は非常に問題です．

そこで(b)のように，システムから「現在時間」を求めるTIME関数(BasicのTIME$関数とか，DOSのカレンダ時計機能とか，Cの時刻呼び出し関数など)を使います．ここでは1秒の経過がないときには，時刻呼び出しを繰り返している(ポーリング)ことになりますから，データ処理に関係しないバックグラウンド処理などは，ここのダミー・ループ内に置くことができます．

ポーリングのメリットとしては，このように時間的にCPUが無駄に足踏みをしないこと，多くの周辺システムと効率的にインターフェースすること，の2点が最大のものです．たいていのCPU周辺LSIでは，「ステータス・レジスタ」という内部情報を格納したポートをもっています．パソコンはここをまずチェックして(これがポーリング)，相手が待たせずに応答してくれる場合にのみ通信することで，CPU本来の能力を十分に生かすことができるのです．

パソコン同士のデータ通信の「標準インターフェース」としては，**図2.9**のようないろいろな方式がありますが，ここでも「ハンドシェイク」と「ポーリング」の通信方式が活躍しています．ここで個々に詳しく説明することは省略しますが，これらの標準インターフ

ェースに接する機会があれば，この視点からも検討してみましょう．

●割り込み

マイコン・システムが外部とインターフェースしていく上で，本当に効率よく高速の処理を実行していくためには，「割り込み」のテクニックは必須の技術です．とくに，Unix や一部の高級パソコンのようなマルチタスク環境のシステムでない，普通の CPU によるボード・マイコンやオリジナル・ボードでは，割り込みを極限まで活用することで，パソコンに負けないハイパフォーマンスを実現できます．

ところが，パソコンでの割り込みというのは非常にやっかいな側面をもっています．MS-DOS のように標準化された高級言語の環境と違って，個々のパソコンのハードウェアに限定された条件がシステム設計に持ち込まれて，たいていの場合にはアセンブラでプログラミングすることを余儀なくされます．また，それまで C 言語と簡単な周辺ハードの知識でシステム設計ができていたのに，本格的に割り込みを活用しようとすると，いきなり CPU や割り込みコントローラ LSI などに関する，理解に必要な技術情報がハードもソフトも増大して，フレッシュマンにとっては困惑するような世界が待ち構えています(**図2.10**)．

●「割り込み」攻略の作戦

パソコンの割り込みについては，システムが複雑な分だけボード・マイコンよりも活用できるまでに時間がかかり，逆にパソコンの割り込みを一度理解してしまえば，ボード・マイコンやオリジナル CPU ボードでの割り込みは自由自在になる，という一種の試金石となっています．このテーマだけのために書かれた書籍がパソコンごとに何種類もあるくらいですから，ここでは深入りせずに，「フレッシュマンがパソコンの割り込みをモノにしていく作戦」のステップについて，以下にまとめてみました．

*

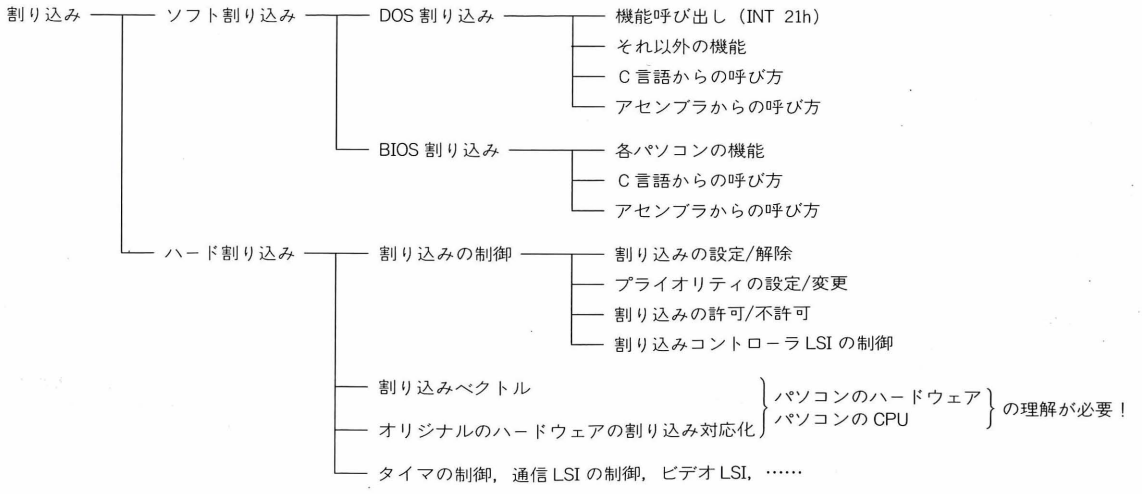
[A] アセンブラが使える

C コンパイラから進んで，MASM などのアセンブラで基本的なプログラミングができるようにします．まずは PC-9801 ならば，8086 という CPU を理解することが必要です．

[B] DOS 割り込み・BIOS 割り込みを使える

アセンブラのプログラムから，MS-DOS 上での「機能呼び出し」やパソコンごとの BIOS 呼び出しを実行して，キーボードからの入力とか CRT への出力を実行します．これらは「割り込み」といっても，ソフトウェア割り込みという，一種のサブルーチン・コールであることを理解します．これによって，DOS の実行

〔図2.10〕パソコンの割り込み周辺技術



プログラムの周辺の手続きも理解できるようになります．

［C］デバッガを使える

アセンブラと一緒に付いてくる，SYMDEBとかDEBUGといったデバッガを使えるようにします．これは，今後の割り込みの実験やデバッグで活躍するとともに，CPUの仕組みを確認する作業でもあります．

［D］割り込みアドレスを設定して呼び出してみる

DOSの「割り込みアドレス設定」機能呼び出しによって，自分の作ったプログラムの一部をベクタ・テーブルに登録して，プログラム中から対応するソフトウェア割り込みを発行して，そこにジャンプ(コール)してみます．以前の内容を退避するとか，他の割り込みを一時的に禁止するとか，のテクニックも理解します．

［E］タイマ割り込みで自分のルーチンを起動してみる

ここでは，パソコンごとのアーキテクチャにしたがって，割り込みコントローラとシステム・タイマの回路図を検討しながら，タイマの割り込みに対応したベクタ・テーブルに自分のルーチンを登録してみます．これで，ハードウェア割り込みとソフトウェア割り込みとが対等にリンクするとともに，かなり正確な時間管理を行えるようになります．

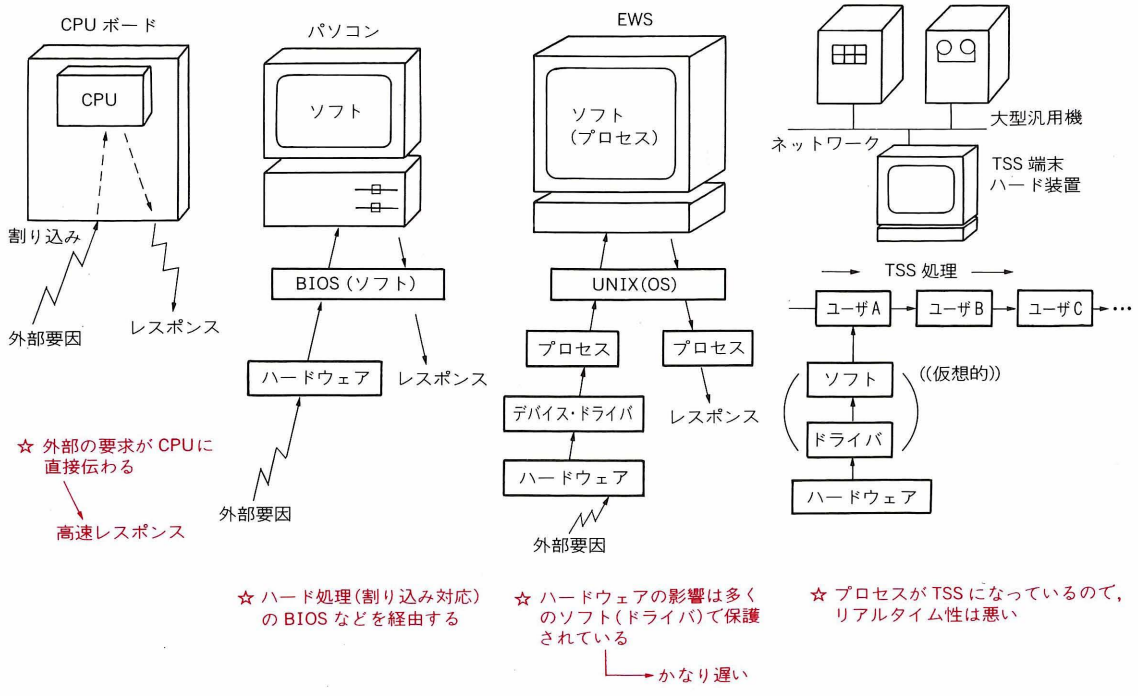
［F］拡張ボード(または内蔵UARTなど)からの割り込みルーチンを作ってみる

これで，パソコンの割り込み技術はほぼ手中に収めたことになります．実際にはパソコンの機種によって，システム予約のハードウェアとの競合とか，多重割り込みの処理で苦労することもあるでしょうが，ここまで来れば，もう自分のオリジナル・システムでも，楽に割り込みを活用できることになります．

［コラム］

リアルタイム・システム

〔図A〕コンピュータ・システムのリアルタイム性



[飛び石コラム] おすすめBOOKS ③

●解析　パワーサプライ
岡村廸夫　著　　CQ出版社

かずかずの名著をもつ著者による，アナログのセンス養成のための秘伝書として，フレッシュマンにぜひおすすめします．直接的には「スイッチング・レギュレータを中心とした電源設計技法」となっていますが，具体的に電源回路を設計しない技術者にとっても，あらゆるアナログ技術のエッセンスを勉強できる，格好の教科書だと思います．

●笑うコンピュータ　息子をハッカーにしないための10章
Karla Jennings 著　　邦つゆこ　訳
技術評論社

これは「読み物」です．ほぼ同時期に出てベストセラーになってしまった(掘り下げが甘いのでやや不満)「ウルトラマン研究序説」に比べて地味ですが，こちらのほうがずっと内容は豊富です．いい意味のハッカー物語であり，エンジニアとして共鳴するところも多いものです．コンピュータ技術者には，ユーモアのセンスも必要なのです．

●数学の問題　エレガントな解答をもとむ＝第1～3集
数学セミナーリーディングス　　日本評論社

これは「数学セミナー」という雑誌に連載されている記事をまとめた本で，かなり本格的な数学の問題を読者が解答する，というものです．しかし，数学を趣味とする人たちの，「エレガントな解法」へのこだわりと柔軟な発想というのは，コンピュータ技術者のための読み物としても，なかなか示唆的で楽しめます．

マイコン・システムとは現実の時間の中で動作するものですから，もちろん基本的にリアルタイム・システムです．ところが**図A**のように，単体のCPUボードよりもパソコン，パソコンよりもEWS，と高級なシステムになるほど，じつはリアルタイム性が失われてしまうのが一般的です．ちょっと不思議な感じですが，リアルタイムの応答性に優れているのが，もっともシンプルなボード・マイコンなのです．

これは，大型で複雑なシステムほど，周辺とのやりとり(割り込み，入出力，通信など)をCPUが直接に実行しないように，いろいろと階層的な周辺処理システムを介しているためです．逆にいえば，ハード的にもソフト的にも，CPUが周辺とリアルタイムに直接やりとりするような「野蛮な」方法では，大きなコンピュータ・システムを構成するには，信頼性とかソフトの保護性の面で，あまりにも危険なわけです．

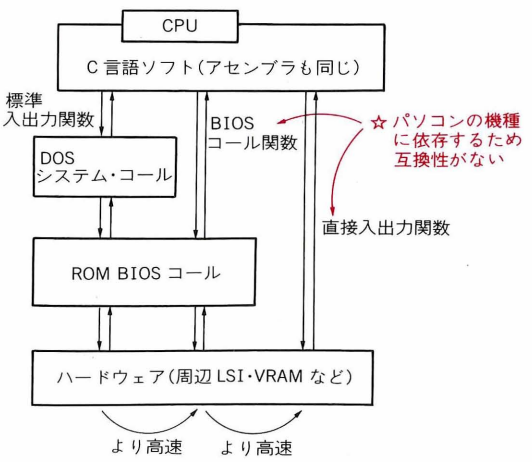
パソコンでも，まったく同様のことがいえます．周辺との入出力動作の方法としては，**図B**のように，

- DOSの機能呼び出しを利用する方法(C言語の入出力関数はここをコールする)
- パソコンのROM BIOSを呼び出す方法
- 直接的に周辺LSIやハードをコントロールしてしまう方法

など，いくつかの段階があります．高速な処理(レスポンス)を望むのであれば，余分なサービス処理や保護(退避)処理を省略して，アセンブラで直接に周辺LSIをコントロールすればいいのですが，あまり多用すると移植性のない場当たり的なソフトになってしまいます．

それでもスピードの要求によっては，複雑・丁寧で遅いprintf( )関数を使わずに直接，DOS割り込みの文字出力機能を呼んだり，パソコン固有のROM BIOSをコールしたり，さらには直接にVRAMにドット・イメージを転送するルーチンを書いてしまうこともあります．こういう方法でスピードを向上させるためには，既存のデバイス・ドライバをたんに使うだけでなく，場合によってはデバイス・ドライバを作ってしまうだけの力量が必要になるのです．

〔図B〕パソコンでの周辺ハードウェア処理



〈3〉 マイコン・システム実戦テクニック集

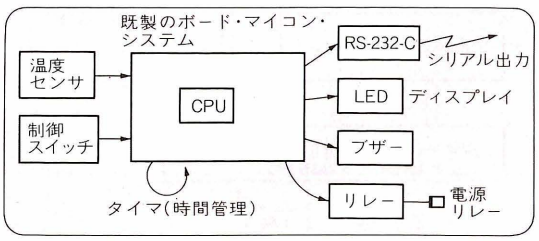
システム設計の実例

●システム構成のサンプル

パソコンほど大がかりでない，汎用のボード・マイコンやオリジナル・ボードを使ったシステムを設計していく例を，**図3.1**のような温度計測システムで考えていくことにしましょう．この図では，中心となるCPUボードの周囲に入出力機能が置かれています．しかし，実際にボード・マイコンを使う場合には，**図3.2**のように，マザーボードを介して複数のボードを使用する場合も多いのです．

汎用ボード・マイコンの場合には，メーカによっておおよそのメモリ・マップが与えられてしまっていますから，パソコンの拡張ボードの検討の例と似たようなものになります．ところがオリジナル・ボードであれば，基本的には設計するエンジニアの自由ですから，将来的な拡張性とか，回路部品の削減などに，それぞれのセンスとテクニックを生かす可能性が出てきます．

〔図3.1〕既製のボード・マイコンによる温度計測システム



● CPUメモリ・マップの検討

たとえば**図3.3**のように，CPUにROMとRAMが一つずつ，それにI/Oが二つ接続されるとしましょう．もっともシンプルなマッピングとしては，デコーダICの139を半分使って，メモリ空間を4等分に分割する，という方法があります．ところがこれでは，あとでI/Oを一つだけ増設しようとしても，メモリの容量を増やそうとしても，ちょっと大変です．

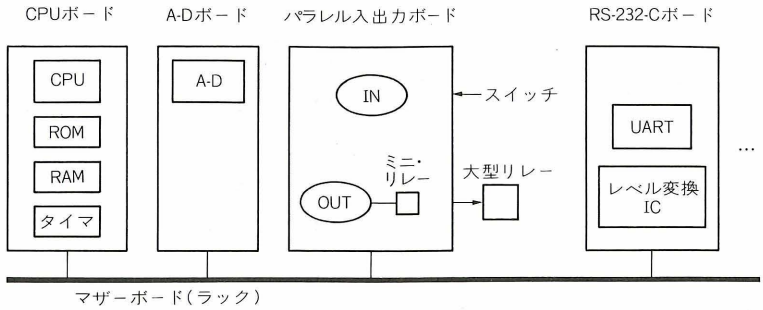
そこで，さらにデコーダの138を一つ加えてみると，将来的にさらに六つまでのI/Oと，倍の容量のメモリ増設にまでそのまま対応できることがわかります．また，もっと拡張したり，さらに上位アドレス(バンク・レジスタのデコード出力)の制御に対応するような設計も，おなじ部品で可能なのです．これは，設計する技術者のアイデア次第ということになります．

図3.4は，まったく同じ部品と回路条件で，いろいろなデコード回路が設計できることを示した，一種の「パズル」の例です．それぞれに，拡張性(予備)の取り方と，マップの配置に特徴がありますが，どれが正解というものではなく，ケース・バイ・ケースで検討されるものです．オリジナルのCPU回路を設計する場合には，せっかくのこのパズルを楽しんでしまいましょう．

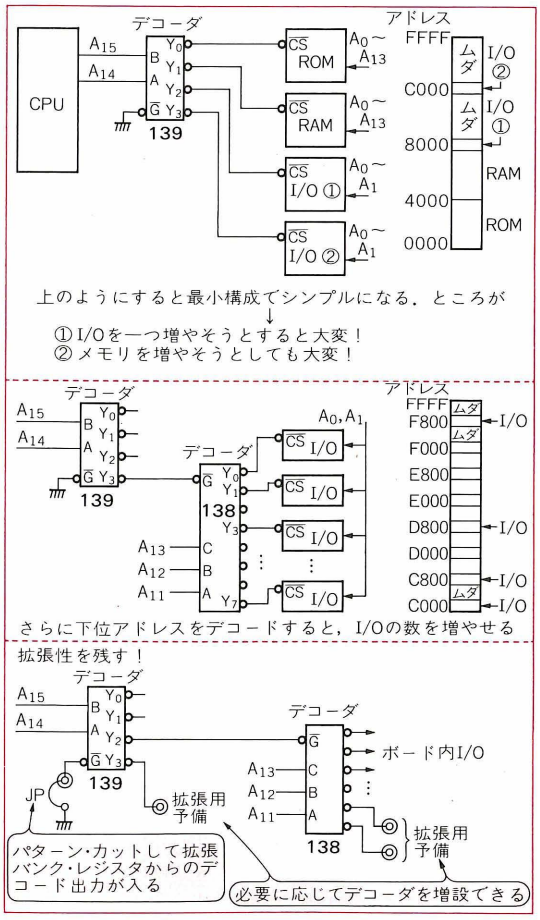
●ブザーを鳴らす

メモリ・マップに続いて，**図3.5**のように，周辺LSIやメモリ・チップを実際に配置して，全体のシステムを構成します．そしてさらに，ここではLED表示とブザーによる出力ポートの部分を自作する，という場合を**図3.6**に示しています．ブザーは，機械的に接点が振動するものは，ノイズ発生の心配があるので使いません．そこで，CPUからラッチに出力した1ビット・データを，ドライバIC(またはトランジスタ)を介して圧

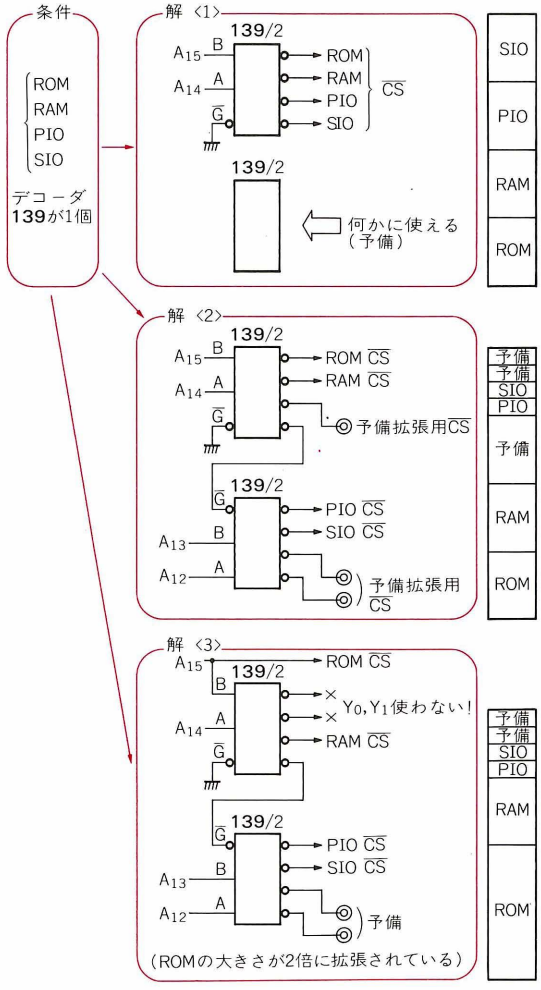
〔図3.2〕複数のボードによってシステムを組む



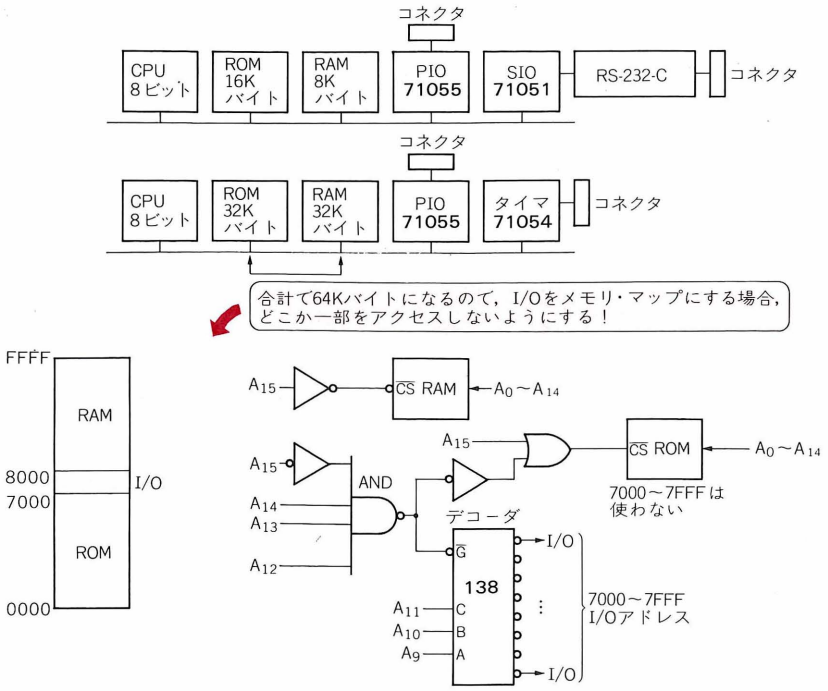
〔図3.3〕CPU メモリ・マップの考え方



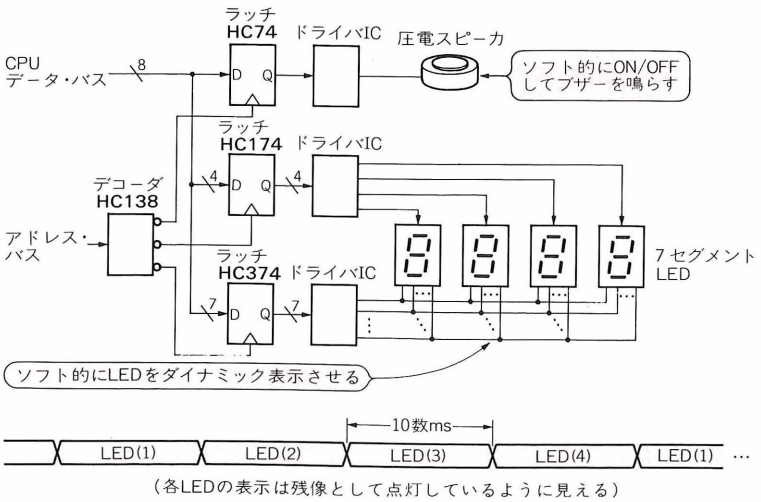
〔図3.4〕デコーダの使い方パズル



〔図3.5〕周辺LSIをならべてブロック図を検討し，メモリ・マップを設計する



〔図3.6〕自作基板の回路例(LED/ブザー・ボード)



（各LEDの表示は残像として点灯しているように見える）

電スピーカに供給して，ブザーの代用とします．たとえば250 Hzのピッチであれば，周期は4 msとなりますから，2 msごとにこのポートをON/OFFしてやればいいわけです．デューティを50％でなく，たとえば1 msと3 msにすると，音色まで変わります．

● LEDのダイナミック表示

ここのLEDは7セグメント・タイプの数字表示方式を使っています．「ダイナミック表示」はよく使うテクニックで，図のようにLEDのアノード側・カソード側ともに大電流ドライバを使って，各LEDをつぎつぎに点灯していきます．人間の視覚は細かいドットをかなり見極めます(分解能は良好)が，発光体の点滅(残像)には意外に鈍感で，このLEDダイナミック表示とか蛍光灯は，この現象をうまく活用しているわけです．簡単なシステムであれば，この点滅の周期は「メイン・ルーチンx回ごとに1回」のようにソフト的に設定しますが，処理によってメイン・ルーチンの1周の時間があまりばらつくと，チラつきとなって気になる場合がありますから，動作面での検討も重要です．

● ACラインの制御

この例では「電源リレー」という出力がありますが，マイコン・システムにとって商用AC電源ラインというのは，なるべく「お近づきになりたくない」狂暴なものです．ACラインは，コンセントを介して屋内にアンテナが張りめぐらされたようなものですし，電源スイッチをON/OFFしたときのインパルス・ノイズは強烈なものです．うっかりすると，リレー・ボード上のドライバICだけでなく，場合によってはCPUボード本体にストレスが加わって故障の原因となる危険性だってあります．

図3.7の例では，リレー出力ボード上のラッチ出力信号をフォト・カプラでアイソレート(分離・絶縁)して，それからドライバICを介してボード上の小型リレーをドライブしています．この場合，アドレス・デコーダ，データ・ラッチ，フォト・カプラの1次側のLEDの電源ラインまでをマザーボードから供給して，それ以降のドライバICとリレーの電源は別にすることが有効です．そしてこのリレー出力(電源側ばかりでなく，接地側も分離されていることに注意)を，さらに別の場所のAC電源リレーのボードに供給しています．

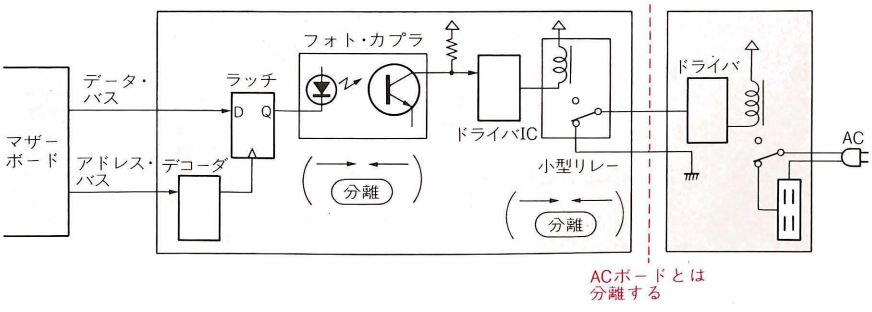
●ソフトの道具だて

以上のようなハードウェアの設計と試作がすすむと，並行してソフトウェアの開発ということになります．この程度の8ビットCPUボードであれば，アセンブラによる開発が手軽で確実ですが，アセンブラといっても，**リスト3.1**のような専用アセンブラ(開発ツール・メーカとかCPUメーカが専用に提供するもの)と，**リスト3.2**のような汎用クロス・アセンブラ(ソフト・メーカが，いろいろなCPUに共通の開発ソフトとして提供するもの)とがあります．汎用クロス・アセンブラは，一般的になったCPUしか対応していないのが難点ですが，マクロ機能やラベル文字数の制限などの点で格段に使いやすいものですから，アセンブラの選択というのも，重要な仕事のポイントになります．

●トップダウン・プログラミング

この例のような簡単なシステムの場合には，筆者はフローチャートもなくいきなりプログラミングすることが多いのですが，そんな場合に便利なテクニックを

〔図3.7〕AC電源をコントロールするときにはアイソレーションが重要



〔リスト3.1〕専用アセンブラによるソース・プログラム例

```
DTCHK:                                  ; DATA CHECK SEQUENCE
        LDA     DATA+1                  ; RX DATA
        BPL     DTLOW
        STA     DATA+2                  ; DATA STOCK
        RTS
DTLOW:                                  ; DATA < 128
        STA     DATA+3                  ; DATA BUFFER
        AND     #0F0H
        BEQ     D000                    ; 00-0F
        CMP     #010H
        BEQ     D010                    ; 10-1F
        CMP     #020H
        BEQ     D020                    ; 20-2F
        CMP     #030H
        BEQ     D030                    ; 30-3F
        LDA     #00H
        STA     DATA+4                  ; SEND PARAMETER
        BRA     DTEXT
D000:
        LDA     #4
        STA     DATA+4                  ; SEND PARAMETER
        BRA     DTEXT
D010:
        LDA     #7
        STA     DATA+4                  ; SEND PARAMETER
        BRA     DTEXT
D020:
        LDA     #5
        STA     DATA+4                  ; SEND PARAMETER
        BRA     DTEXT
D030:
        LDA     #6
        STA     DATA+4                  ; SEND PARAMETER
DTEXT:
        JSR     TXDAT                   ; RESULT DATA SEND
        RTS
```

〔リスト3.2〕汎用クロス・アセンブラによるソース・プログラム例

```
data_check_sequence:
        lda     rx_data
        bpl     data_under_128
        sta     data_stock
        rts
data_under_128:
        sta     data_buffer
        and     #0f0h
        &cp_eq  00h,rx_data_00_0f
        &cp_eq  10h,rx_data_10_1f
        &cp_eq  20h,rx_data_20_2f
        &cp_eq  30h,rx_data_30_3f
        &move   0,send_parameter
        bra     data_check_exit
rx_data_00_0f:
        &move   4,send_parameter
        bra     data_check_exit
rx_data_10_1f:
        &move   7,send_parameter
        bra     data_check_exit
rx_data_20_2f:
        &move   5,send_parameter
        bra     data_check_exit
rx_data_30_3f:
        &move   6,send_parameter
data_check_exit:
        jsr     result_data_send
        rts
```

〔リスト3.3〕「下書きなしのプログラム」の最初の状態

```
;----- Address Defines -----
work    equ     04000h          ; RAM Top Address
stack   equ     07fffh          ; RAM Tail Address
port1   equ     08000h          ; I/O (1)
port2   equ     0a000h          ; I/O (2)
port3   equ     0c000h          ; I/O (3)
port4   equ     0e000h          ; I/O (4)

;----- Starter Area -----
        org     00000h          ; RESET Vector Address
start:
        ld      sp,stack        ; Stack Pointer Set
        di                      ; Interrupt OFF
        call    initial         ; Initialize Routine
        jp      main_loop       ; Go to Main Loop !

;----- INT Area -----
        org     00038h          ; INT Vector Address
int:
        reti

;----- NMI Area -----
        org     00066h          ; NMI Vector Address
nmi:
        retn

;----- Initial Area -----
initial:                        ; Initialize Routine
        ret

;----- Main Program -----
main_loop:                      ; Main Program Loop
        jp      main_loop
```

〔リスト3.4〕まずはメイン・ループとサブルーチン名だけを並べる

```
;----- Main Program -----
main_loop:                      ; Main Program Loop
        call    sw_scan         ; Panel Switch Scan
        call    event_check     ; SW Event check
        call    code_select     ; Display Code Select
        call    led_display     ; LED Output
        jp      main_loop

sw_scan:
        ret

event_check:
        ret

code_select:
        ret

led_display:
        ret
```

紹介しましょう．**リスト3.3**は，どんなシステムでも最初に使うために，別にこれだけ作っておいてある「ひな型」プログラム(ここではZ80用)の例です．

基本的なハードウェアの構成に対応したI/Oのアドレス定義エリア，初期設定ルーチンをコールしてからメイン・ルーチンに飛んでいくリセット処理，そして割り込みルーチンのエリアが定義されています．といっても，それぞれのルーチンの中身はからっぽで，たんなるリターンだけ(メイン・ルーチンは無限ループ)のものです．

これをコピーしてきて，そのままアセンブルするとエラーもなく，何も起こりませんが，ROM化して実行可能です．ただし，このプログラムがあれば，白紙から打ち込むよりも簡単で，割り込みルーチンの記述忘れによる暴走，などというミスも回避できます．

ここに具体的な処理を構成していく場合には，たとえば**リスト3.4**のように，メイン・ルーチンから具体的な処理ルーチンをコールして，同時にコールされるルーチンの部分も，最初はリターンのみを置いて並べていきます．そして，それぞれの中身を実際に記述して，少しずつふくらませていくわけです．

この方法(構造化プログラミングのもっともシンプルなかたち)のメリットは，**図3.8**のように考えることができます．つまり，以前にOKだったプログラムをこのような方法でふくらませていって，もしエラーが起きれば，前回との差分のところに原因がある可能性が高い，という発想で，確実にシステムを大きくしていけるのです．つぎに述べる開発支援ツールとあわせて，この方法でソフト開発をしていくことにより，効率よくシステムを実現していけます．

開発環境：ICE

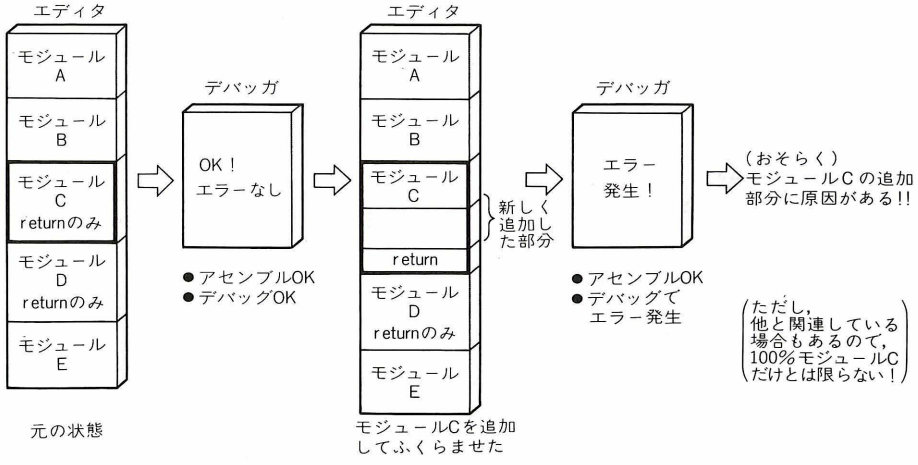
●マイコン開発支援ツール

ボード・マイコンやオリジナル・ボードの開発・デバッグには，「開発支援ツール」の活用が欠かせません．CPUボードに関わる技術者のまずほとんど全員，あるいはパソコンそのものを開発する技術者にとって，**開発ツール，ハード・デバッガ，ICE**などは，「なかったら仕事にならない」というほどのものです．マイコン・システムの発展の歴史というのは，開発ツール専業メーカという強力な裏方に支えられたものだった，といえるかもしれません．

ソフト上のシミュレーションによるテストでは，どうしても動作の再現性に限界があります．実機のCPUソケットにターゲットCPUがセットされ，プログラムROMソケットに目的ソフトの書かれたROMを挿入して，「せーの！」でリセットする…．これで一発で正常にシステムが走るのなら，デバッグは不要なのですが，そういうことはほぼ100%ありません．

そこで開発ツールの登場となります．ボード上のCPUソケットから「出張所」(エミュレータCPUボックス)まで信号線を延長して，すべての信号線を監視さ

〔図3.8〕段階的にモジュールをふくらませると，デバッグでの原因追求がやりやすい



れたCPUに動作を代行させる，というICEの仕組みは，少しずつ理解できるにつれて感心したものです．ところが時代が進むにつれて，CPUの高速化とバス・タイミングの複雑化のために，現在では離れた「出張所」にCPUを置くのでは，タイミングが微妙に合わなくなってきました．そこで，ボード上のCPUの位置のパッドにまで，エミュレーションCPUと周辺ロジックが進出してきました．この先となると，もうCPUチップ内にデバッグのためのロジックを内蔵してしまうしかないようです．

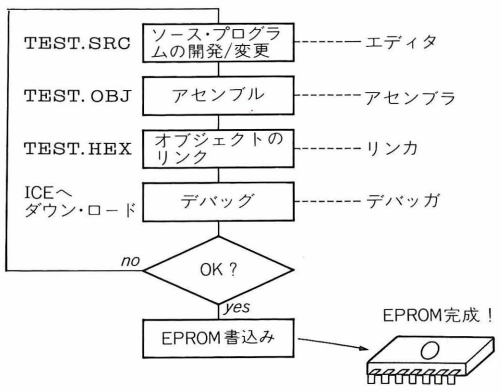
さて，同じツールを使って，同じ回路のCPUソフトをデバッグしたとしても，ベテラン技術者と若手とでは，歴然とした差が出ます．つまり，開発支援ツールというのはあくまで「道具」であって，これを生かすも殺すも，使っている本人次第となるのです．

●かつての開発支援ツール

ここで，開発ツールによるマイコン・システムのソフトウェア開発の手順を，**図3.9**を例にとって簡単に説明しましょう．これは，1台のマシンがすべての開発作業を支援するという，過去に主流であったタイプの専用マシンです．図のようにソース・プログラムのエディット，アセンブル，リンク，ICEへのダウンロード，そしてデバッグまでを実行します．デバッグの完了したプログラムは，これも筐体のパネル面にもっているROMライタで，EPROMへの書き込みまで済ませてしまいます．

このように書くと，この「開発支援ツール」が万能のように思えてしまいますが，現在ではCPUメーカのオリジナル1チップ・マイコンの専用機以外では，すっかりマイナーになっています．その原因は，「パソコンICE」の登場です．

〔図3.9〕開発ツールによるマイコン開発の手順



●パソコンICE

最近のマイコン開発ツールの主流は，なんといっても**図3.10**のような，パソコンをホストにした「パソコンICE」のスタイルです．パソコンの機能向上にともなって，この方式によるシステム開発効率は，相当なペースで向上しつづけているのです．

この方式のメリットとしては，すでに述べた「汎用クロス・アセンブラ」に対応していることと，パソコン用の汎用エディタが使えることが大きいものです(**図3.11**)．高級言語ではもちろんですが，アセンブラであればなおのこと，マクロ機能やモジュール化のテクニックを駆使しますから，

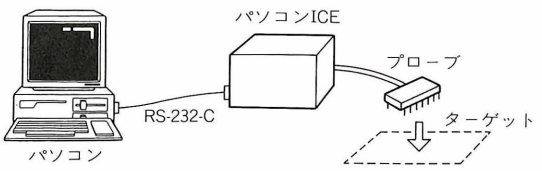
- 同時にいくつものファイルをオープンして編集
- カット&ペースト機能
- カーソル移動・スクロールのスピード
- キーのカスタマイズやマクロ・ファンクション

などの要因は重要なファクタとなります．この点で，パソコン用の汎用エディタの機能はとても強力で，開発効率に大きく影響するのです．

●パソコン開発環境の例

パソコンICEを活用して，開発環境の中心にパソコンを使った場合の例は，**図3.12**(a)のようになります．この図ではフロッピ・ディスクの図としてありますが，実際のプロの現場では，当然ながらすべてハード・ディスク以上のレベルです．「以上」というのは，たとえば筆者の愛用する環境の場合，10数Mバイトの RAMボードをすべてRAMディスクにして，開発ターゲット・プログラム(ソース・プログラムやオブジェ

〔図3.10〕パソコンをソフトにした開発システム



クト・プログラム)はもちろん，アセンブラ，リンカ，エディタなどのすべてのツールまで，完全にRAMディスク上で作業しますから，ハード・ディスク版での仕事は「かなり遅いなあ」と感じるスピードとなっています．

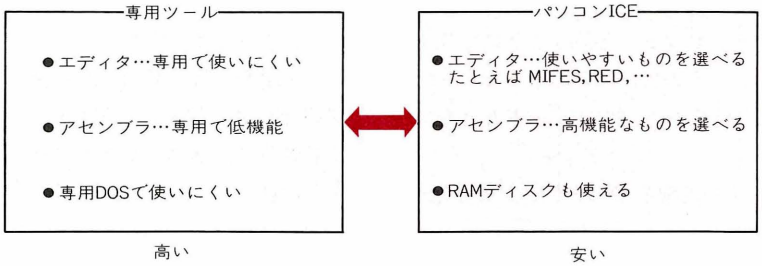
そしてもちろん，停電などのトラブルに備えて，1日数回はRAMディスクから100Mバイト以上のハード・ディスクにバック・アップをとり，さらに毎日，このハード・ディスクをまるまる，同じ容量の別のハード・ディスクにバック・アップしています．そして1週間に一度ずつ，このハード・ディスクの内容をすべて，大容量データ・ストリーマにバック・アップ(2本のテープに交互)しています．この開発環境の全体は，瞬間停電などに備えた「無停電電源装置」から電源を供給されているのも当然です(しかしそれでも，泣きたくなるようなトラブルは，ある日突然にやってくるものなのです)．

開発環境：モニタ

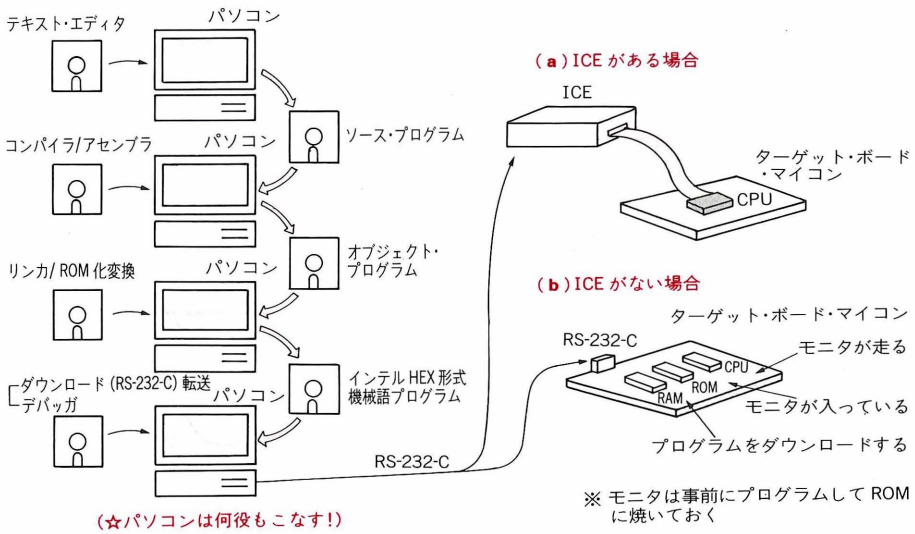
●パソコンとボード・マイコンの環境の違い

マイコン・システムのプラットフォームがパソコンからボード・マイコンに移ったときに当面するのは，「開発環境」，「デバッグ環境」の大きな差でしょう．パソコンであれば，DOSの環境のもとでソフトを開発して，DOSのもとで実行プログラムを起動して，正常に終了すればDOSに戻ってきました．つまり，お釈迦様

〔図3.11〕パソコンICEのメリット



〔図3.12〕ボード・マイコンのソフト開発手順の例



の掌のような，「すべてを見守ってくれる環境」の世界だったのです．

ところが，ボード・マイコンのCPUソフトというのは，電源が入る(リセットされる)とスタートして，どこにも戻らずに走り続ける無限ループの世界です．パソコンのDOSがデフォルトで用意していた，周辺LSIの初期設定から入力・表示装置のクリア，メイン処理プログラムの起動まで，すべて自分で設定してやらなければなりません(**図3.13**)．バグのために暴走したとしても，パソコンの「エラー・メッセージ」のようなものは出ませんから，エラー・メッセージの表示の部分も，自分で作り込んでやらなければならないのです．

●「モニタ」というプログラム

これは逆に見ると，DOS(Disk Operating System)というプログラムに相当する部分を自分で作る，ということに他なりません．OSというと大げさですが，もっと簡単には「モニタ」ソフト，ということになります．もともとモニタというのは，現在のようなパーソナルなOSがなかった時代から，CPUボードとともにありました．つまり，

- ボード上の周辺LSIの処理をBIOSとして共有化・標準化する
- プログラムを外部からボード上のRAMにダウンロードする
- プログラムを逆アセンブル表示する
- ブレークポイント動作などのデバッグ用ツール

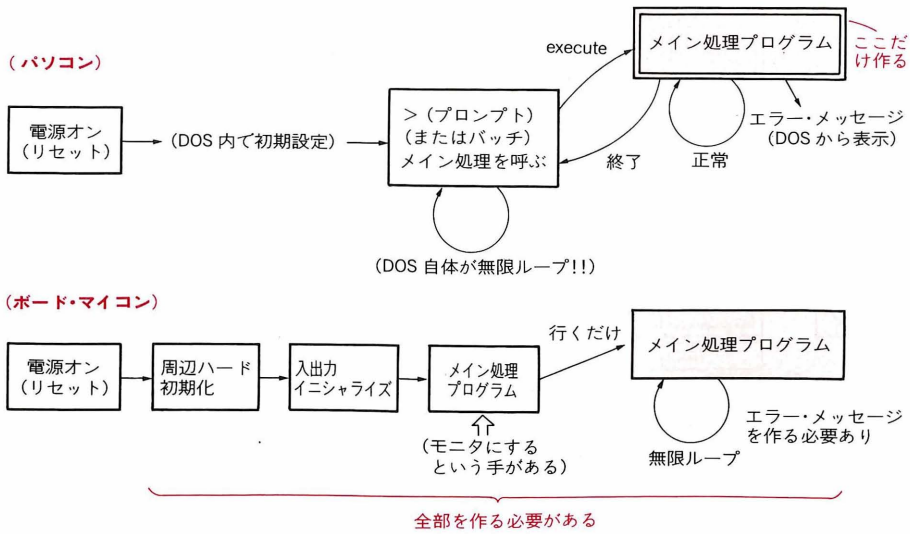
といった機能をもつプログラム(＝モニタ)をあらかじめ用意したのです．

最近のボード・マイコンには，**図3.14**のようにROMとして簡単なモニタを用意しているものも多く，ソフトの開発環境・デバッグ環境としては，かなり大きなプラス要因となります．そして最初は標準の内蔵モニタを使いますが，いずれはモニタを改良したり自作していきたいものです．

具体的な開発方法としては，**図3.12**(b)のように，デバッグ・モニタとRS-232-Cのインターフェースを標準装備している最近のボード・マイコンの場合には，パソコン上でクロス・アセンブラやクロス・コンパイラで開発したプログラムを，EPROMプログラム装置に転送するのと同様の「インテルHEX」形式などのデータとして，ボードのモニタに対して転送すればいいことになります．ただし，まったくのオリジナル・ボードの場合には，最初の開発環境となるべき「モニタ」部分(ROMに焼かれる部分)も，スタートから紙の上で設計して，具体的なプログラム開発の支援環境を作らなければなりません．

マイコン・システムの製作記事をよく見かけますが，この「モニタ部分の開発」というのは，決まりきった

〔図3.13〕パソコンとボード・マイコンの環境の差は？



ハードウェアなどよりも，ある意味で技術力の見せどころともいえるもので，それぞれの作者ごとの個性がよくあらわれています．実際の処理部分にあたる「プログラム」よりも，それを底辺で支えるBIOSとかモニタ，あるいはOSといった「システム・プログラム」に注意が向くようになったらプロだ，ともいわれています．地味なイメージの「モニタ」ですが，機会をみて注目してみましょう．

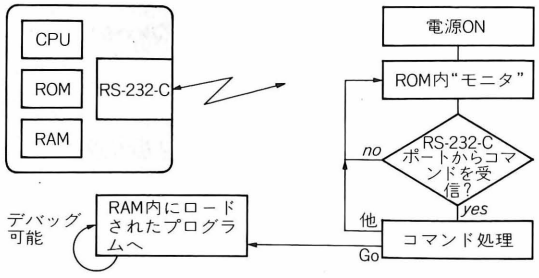
ソフト・テクニック：「ワンパターン処理」

ここでは，**図3.15**のように，10個のスイッチの状態変化(イベント)を検出して，10個のLEDにその状態を表示するような例について考えてみましょう．同じような処理を10個並べる「ワンパターン」の例というわけです．

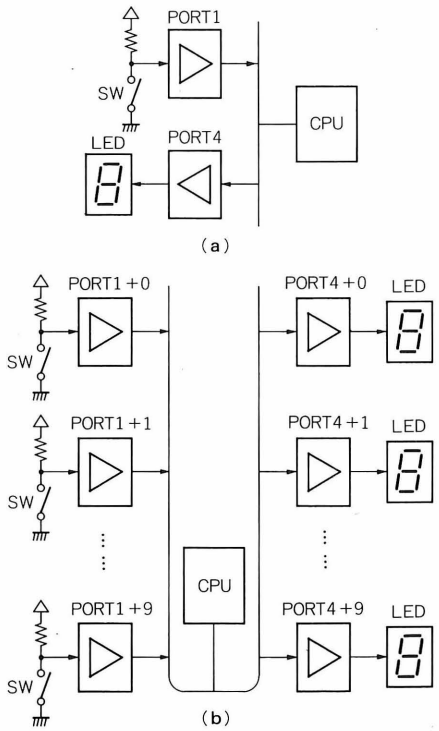
●ワンパターンにはマクロで対処

まず，**リスト3.5**のように，RAM領域に変数を定義して確保します．ここでは10個ではなく，最大16個に拡張できるように配置しています．**リスト3.6**が(ずるい？)マクロの例で，まずは冒頭にマクロ定義を置き

〔図3.14〕デバッグ・モニタROM付きボード・マイコン



〔図3.15〕10個のスイッチの状態を10個のLEDで表示する



〔リスト3.5〕変数領域に定義を並べていく

```
work    equ     04000h          ; RAM Top Address
sw_new  equ     04100h          ; SW Input [New] Status
sw_old  equ     04110h          ; SW Input [Old] Status
eve_flg equ     04120h          ; Event Flag
led_dat equ     04130h          ; LED Display Data
stack   equ     07fffh          ; RAM Tail Address
```

〔リスト3.6〕マクロを使った(ずるい？)方法

```
;----- MACRO Defines -----
macro   sw_check_led(no)
        ld      a,(port1+%no)   ; SW Input Port
        xor     a,0ffh          ; Inverting
        and     a,00000001b     ; Mask
        ld      b,a             ; Result
        ld      a,(sw_old+%no)
        ld      d,a
        call    event_check     ; SW Event check
        ld      a,d
        ld      (sw_old+%no),a
        call    code_select     ; Display Code Select
        ld      a,e             ; Code Load
        ld      (port4+%no),a   ; LED Display Port
        endmac

;----- Main Program -----
main_loop:                      ; Main Program Loop
        sw_check_led(0)
        sw_check_led(1)
        sw_check_led(2)
        sw_check_led(3)
        sw_check_led(4)
        sw_check_led(5)
        sw_check_led(6)
        sw_check_led(7)
        sw_check_led(8)
        sw_check_led(9)
        jp      main_loop
```

ます．ここでは，sw_scan と led_display に相当する部分は直接マクロ中に記述してしまい，レジスタだけを処理する残りの二つはそのままサブルーチン・コールしています．そして，スイッチの「前回値」は他のスイッチの処理の期間も保持する必要があるので，変数に退避します．スイッチ入力と LED 出力のポート・アドレスの部分は，マクロ・パラメータとして外部から与えるところがミソです．こうすると，メイン・ループ中にマクロ・パラメータを替えるための 10 行のマクロが並ぶだけで，目的とした処理を実行できるプログラムがとりあえず完成します．このように，アドレスとかのごく一部だけが変わるワンパターンの処理が並ぶ場合，まず短期間に「やっつける」ようなときにはマクロ・コールの利用というのは重宝します．この方法を「ずるい？」と呼ぶ理由は，**リスト3.6**のソース・プログラムを見ると非常に簡潔にできているようですが，実際にはマクロ展開されて，マクロ定義部分のプログラムがずらっと並ぶだけで，オブジェクト効率からすると冗長になっているからです．しかし筆者自身，締め切りギリギリの仕事とか，CPU の処理スピードにかなり余裕のあるときなど，「ずるいというのはウマイこと」とかいいながら，けっこうよくお世話になっています．

〔リスト3.7〕オフセット指定をＢレジスタで行った例

```
sw_scan:
        ld      hl,port1        ; SW Port
        ld      a,b             ; Select NO.
        add     a,l
        ld      l,a
        ld      a,(hl)          ; SW Input
        xor     a,0ffh          ; Inverting
        and     a,00000001b     ; Mask
        ld      (sw_new),a      ; Result --> [sw_new]
        ret

event_check:
        ld      a,0
        ld      (eve_flg),a     ; Event Flag
        ld      hl,sw_old       ; Old Status
        ld      a,b             ; Select NO.
        add     a,l
        ld      l,a
        ld      d,(hl)          ; Old Load
        ld      a,(sw_new)      ; New Load
        cp      a,d             ; New = Old ?
        ret     z               ; equal !
        ld      (hl),a          ; New --> Old
        ld      a,1             ; Event !
        ld      (eve_flg),a
        ret

code_select:
        ld      a,(eve_flg)     ; Flag Load
        cp      a,0             ; Event ?
        jr      z,_none
_event:
        ld      a,(sw_new)      ; Old(New) Load
        cp      a,0             ; ON ?
        jr      z,_off
_on:
        ld      a,00h           ; [ON] Code
        ld      (led_dat),a     ; Code Buffer
        ret
_off:
        ld      a,01h           ; [OFF] Code
        ld      (led_dat),a     ; Code Buffer
        ret
_none:
        ld      a,0ffh          ; [None] Code
        ld      (led_dat),a     ; Code Buffer
        ret

led_display:
        ld      hl,port4        ; LED Port
        ld      a,b             ; Select NO.
        add     a,l
        ld      l,a
        ld      a,(led_dat)     ; Code Load
        ld      (hl),a          ; LED Display
        ret
```

●ワンパターンにはサブルーチンで対処

さて，それでは「ずるくない」方法とはどうなるでしょうか．**リスト3.7**にある各サブルーチンは，Ｂレジスタにポート指定のパラメータが格納されている，とした場合の例です．これらのルーチンは，メイン・ルーチンを**リスト3.8**のようにすることで，何度も同じパターンのプログラムがオブジェクトとして繰り返される**リスト3.6**と違って，非常にコンパクトになります．

この**リスト3.7**と**リスト3.8**にあげた方法というのは，じつはサブルーチン・コールの使い方としては，やや中途半端なものです．オフセット指定のためにＢレジ

〔リスト3.8〕リスト3.7 のサブルーチンを呼ぶメイン・ループの例

```
;----- Initial Area -----
initial:                        ; Initialize Routine
        ld      b,0             ; Select NO. Start=[0]
        ret

;----- Main Program -----
main_loop:                      ; Main Program Loop
        call    sw_scan         ; Panel Switch Scan
        call    event_check     ; SW Event check
        call    code_select     ; Display Code Select
        call    led_display     ; LED Output
        inc     b               ; Next Select NO.
        ld      a,b
        cp      a,10            ; Over ?
        jp      nz,main_loop
        ld      b,0             ; If Over, Zero Re-Start!
        jp      main_loop
```

スタを予約してある，というのも，メイン・ルーチンにこのパラメータのインクリメントとか判定とかがある，というのも，あまり美しいものとはいえないでしょう．そこで，もう1段階，手を加えることになります．まずは変数のリストに，[sel_no]というような変数を追加しておいて，**リスト3.9**のようにメイン・ルーチンを変更するとしたら，どのようになるでしょうか．

サブルーチンの変更の例としては，**リスト3.10**のような形になります．ここでは，前の例のパラメータとしてBレジスタに置いたオフセット指定が，変数になっているわけです．そして，最後にあるnumber_setというサブルーチンで，この変数の変更・判定を行います．このようにすると，**リスト3.9**のメイン・ルーチンを見るかぎり，これら五つのサブルーチン以外に影響が及ばない，独立性の高い動作を実現できます．これはプログラムの作法としては重要なことで，**リスト3.9**のメイン・ルーチンにさらに新しい処理ルーチンを追加していくような場合を考えてみるとわかります．外部から見たときに，このメイン・ルーチンに並んだ五つのサブルーチンが相互に関連して何をやっているかが見えない，という情報のマスキングのためには，なるべく**レジスタを局所的に使う**，というのがポイントといえます．必要なパラメータの受け渡しは，面倒でもメモリ中に置いた変数を介するようにするわけです．

●モジュール分割の考え方

ここでは，ワンパターン処理を行うときの方法として，**マクロ・コール**と**サブルーチン・コール**という二つの実例を紹介しました．マクロ・コールが並ぶとソース・プログラムは非常に見やすくなり，規模が大きくなるほどその効用は身にしみてくるのですが，アセンブラがマクロを展開したアセンブル・ソース・リスト(またはデバッガで見られる逆アセンブル・リスト)を眺めてみると，同じ処理がずらっと並んでいて，とても冗長な印象をもちます．一方，サブルーチン・コールの手法によるものは，オブジェクトにしてもそのままですから，かぎられたメモリにプログラムを詰め込む必要のある1チップ・マイコンなどでは，安易にマクロを使わないで，なるべくサブルーチン・コールによってオブジェクト効率を上げるような注意が必要になります．

〔リスト3.9〕すっきりしたメイン・ループ

```
;----- Main Program -----
main_loop:                      ; Main Program Loop
        call    sw_scan         ; Panel Switch Scan
        call    event_check     ; SW Event check
        call    code_select     ; Display Code Select
        call    led_display     ; LED Output
        call    number_set      ; Select NO. Set
        jp      main_loop
```

〔リスト3.10〕リスト3.9から呼ばれるサブルーチンの例

```
sw_scan:
        ld      hl,port1        ; SW Port
        ld      a,(sel_no)      ; Select NO.
        add     a,1
        ld      l,a
        ld      a,(hl)          ; SW Input
        xor     a,0ffh          ; Inverting
        and     a,00000001b     ; Mask
        ld      (sw_new),a      ; Result --> [sw_new]
        ret

event_check:
        ld      a,0
        ld      (eve_flg),a     ; Event Flag
        ld      hl,sw_old       ; Old Status
        ld      a,(sel_no)      ; Select NO.
        add     a,1
        ld      l,a
        ld      d,(hl)          ; Old Load
        ld      a,(sw_new)      ; New Load
        cp      a,d             ; New = Old ?
        ret     z               ; equal !
        ld      (hl),a          ; New --> Old
        ld      a,1             ; Event !
        ld      (eve_flg),a
        ret

led_display:
        ld      hl,port4        ; LED Port
        ld      a,(sel_no)      ; Select NO.
        add     a,1
        ld      l,a
        ld      a,(led_dat)     ; Code Load
        ld      (hl),a          ; LED Display
        ret

number_set:
        ld      a,(sel_no)      ; Select Number
        inc     a               ; increment
        cp      a,10
        jr      nz,_exit        ; Overflow ?
        ld      a,0
_exit:
        ld      (sel_no),a      ; New NO. Set
        ret
```

ところがサブルーチン・コールにも欠点はあります．ソース・プログラムを論理の見やすさのために**サブルーチンに分割すればするほど，リターン・アドレスのスタックへのプッシュ/ポップという手間がかかるために，実行速度の点ではマクロよりも遅い**，という点を配慮する必要もあるのです．ソースがマクロで記述されていたとしても，実際のオブジェクト・プログラムではコールなしの機械語がズラッと並ぶために，長いけれどもじつは速い，というのがマクロの長所でもあるのです．このほかにも，**表3.1**にあるように，マクロの記述方法にはアセンブラごとの「方言」が大きい，というような特徴もあり，この二つの方法をどのように臨機応変に使い分けるか，というのは面白いポイントであるように思います．何かワンパターンの処理の繰り返しに出会ったときに，たいていの場合にはマクロ・コールでもサブルーチン・コールでも扱うことが可能ですから，まずはどちらかを使ってみて，さらに「別の方法だとどうなるか？」と考えるような癖をつけると，次第に適切な，美しい方法がピンとくるようになります．

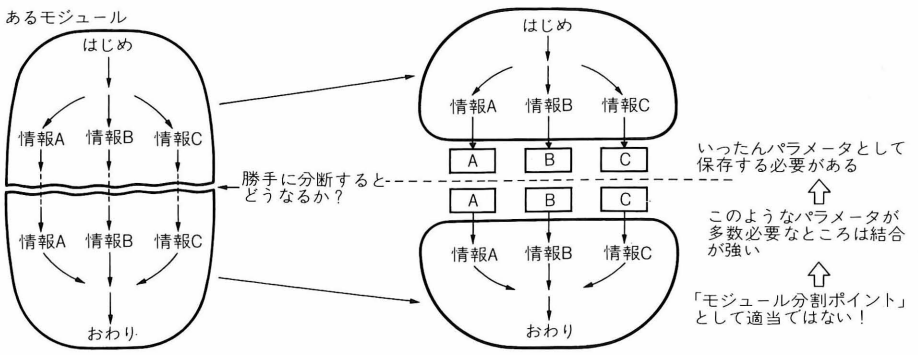
このモジュール分割の考え方は，理論的にも歴史の深い，モジュール間の結合度といった視点に展開できるものです．たとえば，ある一連のプログラム・モジュールを，まったくいいかげんな場所で二つに分断してしまい，その前後の情報として継続させる必要のあるものをすべてパラメータとして退避・引き渡しさせる，というような場合を想定してみましょう(**図3.16**)．分断された二つのモジュールは別々に呼ばれるので，その間に別のルーチンが入る場合もあり，レジスタなどに保持した情報は消滅してしまいます．このため，分断の前後で継承されるべき情報は，すべてパラメータとして転送して確保する必要が生じてきます．このパラメータがあまりに多いとすれば，つまりこの二つの部分の「結合度」が大きく，モジュール分割には適当な場所でなかった，ということになるわけです．この見方からすると，大きくなったモジュールを分割するための視点として，結合度が小さくなるポイントを選ぶ，という指導原理が見えてきます．

これは何もアセンブラにかぎったことではなくて，Cプログラムで長くなってきた関数を見やすくするために分割する場合とか，BIOSやDOSの項目としてパラメータ数の少ないように機能分割する場合とか，新しいサービス・ルーチン(ソフトウェア割り込み処理)を設計する場合など，およそある程度のソフトウェアを開発する際には直面することの多いテーマであるともいえるのです．先にあげたワンパターン処理に気づいたときに考えてみるのも，「どのようにモジュール(マクロ/サブルーチン)を設定すると結合度が小さくなって，スマートに処理を記述できるか」というポイントにほかならないのです．ソフトウェア技術としてはかなり基本的なテーマですので，この視点については頭のどこかに置いてみてください．

〔表3.1〕マクロ・コールとサブルーチン・コールの比較

	マクロ・コール	サブルーチン・コール
ソース・プログラムの姿	スッキリ	スッキリ
パラメータの渡し方	直接指定	レジスタ/変数による
オブジェクト・プログラムの姿	各マクロの展開されたリストがずらっと並ぶ	スッキリ
プログラム量	展開されただけ大きくなる	小さくてすむ
処理スピード	速い	いちいちサブルーチン・コールするので遅くなる
記述方法(方言)	アセンブラによって方言が大きい(異なる)	同じCPUなら，まずは共通

〔図3.16〕
モジュール間の
結合の考え方



ソフト・テクニック：「FIFO処理」

RS-232-Cなどのシリアル通信でよく使われるテクニックとして，ここでは「FIFOバッファ」の処理について考えてみましょう．これは**図3.17**のように，データを記憶する領域がリング状になっていて，別名「リング・バッファ」ともいいます．データを書き込むためのポインタとデータを読み出すためのポインタが1個ずつあり，それぞれリング上を同じ方向に回っていくために，先に記憶されたものが先に読み出されるものです．

● FIFOの動作とは

さて，**図3.17**を使って，FIFOの動作を確認しておきましょう．まず**図3.17**(a)の状態が最初となります．リードとライトのポインタは両方とも同じ位置を示しています．ここでのポインタの意味をもっと正確にしておくと，

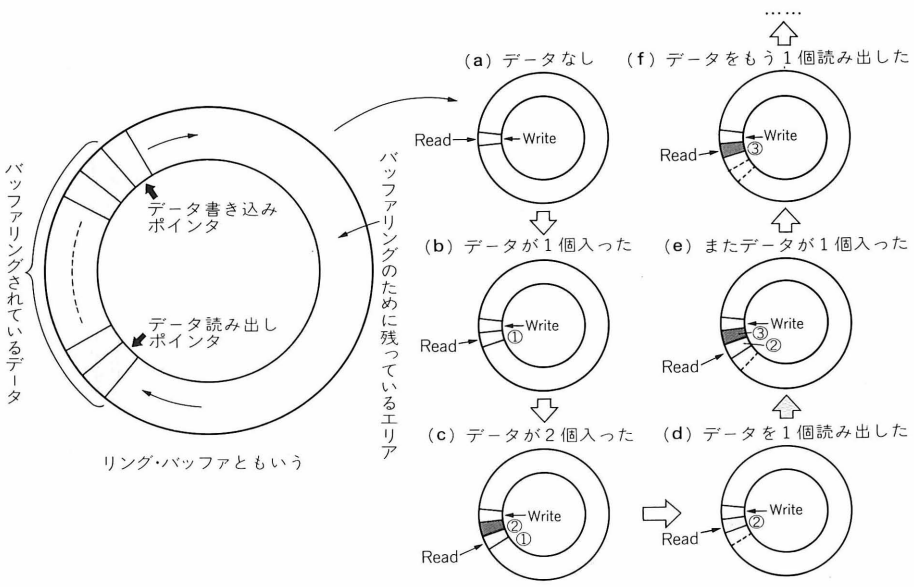
ライト・ポインタ：つぎにデータが書き込まれる位置
リード・ポインタ：つぎにデータが読み出される位置

となります．これとは別の意味にも定義できますが，ここでは上のようにしておきましょう．(a)の状態であるとき，すなわち二つのポインタの内容が一致しているときには，読み出しルーチンは何もなかったとしてリターンするようにします．

さて，FIFOにデータが一つ入力された，というのが**図3.17**(b)になります．この図の状態では，さきほどライト・ポインタのあった位置にデータが書き込まれ，ライト・ポインタは一つ進みました．まだ読み出しルーチンに来ていないので，リード・ポインタはそのままです．ここにさらにデータがもう一つ入力されると，同様の動作によって(c)の状態になります．読み出しルーチンが呼ばれるまでに入力されたデータは，このようにリングの進行方向につぎつぎに格納され，その先頭(最新データ)の一つ先をライト・ポインタが示しています．

ここで読み出しルーチンが呼ばれたとします．リード・ポインタは一番古いデータを示していましたから，そのデータが読み出され，リード・ポインタは(d)のようにつぎのデータに移っています．このようにして，さらに(e)，(f)というように動作が進行していくわけです．リード・ポインタが通過してしまった，つまり読み出されたデータというのはもう不用となってしまい，そのうちライト・ポインタが1周まわってきて，新しいデータを書き込むための予備エリアになってしまいます(データ自身は消えずに残っていますが，再びアクセスする手段がないだけです)．

〔図3.17〕FIFO(First In First Out)バッファとは

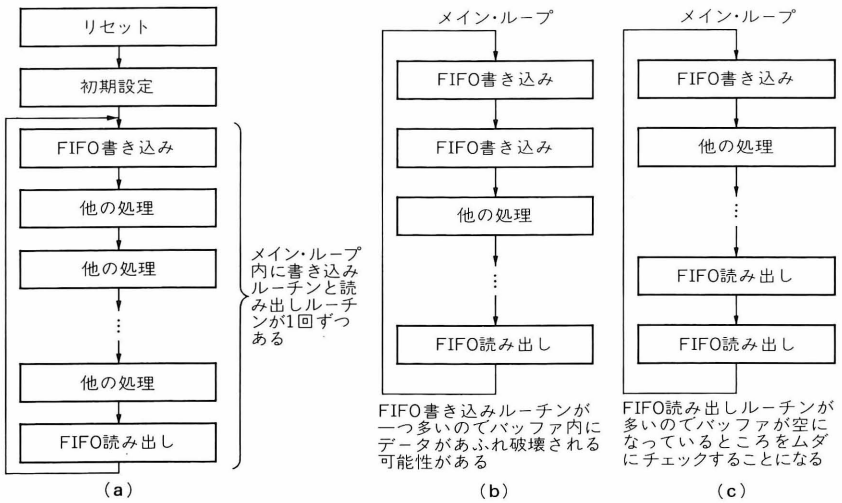


● FIFOを使うのはどのような場合か

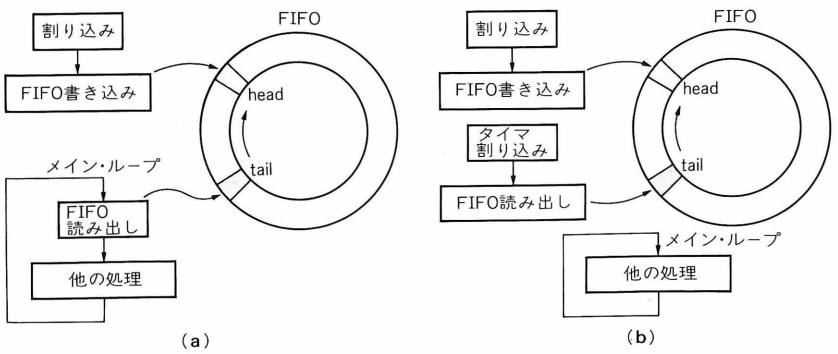
それでは，「どんなときにFIFOを使うのか」を確認しておきましょう．これには，FIFOを使うと問題のある例を考えるのが早道です．**図3.18**はそのような「よくない使い方」の例で，(a)ではメイン・ルーチンの中にFIFOバッファの書き込みルーチンと読み出しルーチンが1回ずつ含まれています．この場合，もしバッファに積むべきデータがあるときは，そのあとで必ず読み出しルーチンでデータを取り出しますから，別にFIFOの形式のバッファに積む必要がないだけです．さらに，(b)のようにメイン・ループ内にFIFO書き込みルーチンのほうを多くおいたものでは，しだいにバッファ内のデータがあふれてきて，リングを1周するとポインタが追い付いてデータが破壊されたり，(c)のように逆に読み出しルーチンが多いものでは，バッファが空になっているところを無駄にチェックすることになってしまいます．

これに対して**図3.19**は，よくあるFIFOバッファの使い方を示したものです．(a)の場合には，通信データの受信割り込み要求のような割り込みによってFIFO書き込みルーチンが呼ばれています．そして，読み出しルーチンはメイン・ループの中に置かれ，これらの二つのルーチンはそれぞれ別個のタイミングで勝手にFIFOバッファをアクセスしています．また(b)の場合には，同様の割り込みによってFIFO書き込みルーチンが呼ばれるとともに，タイマ割り込みのような一定

〔図3.18〕FIFOバッファのよくない使い方



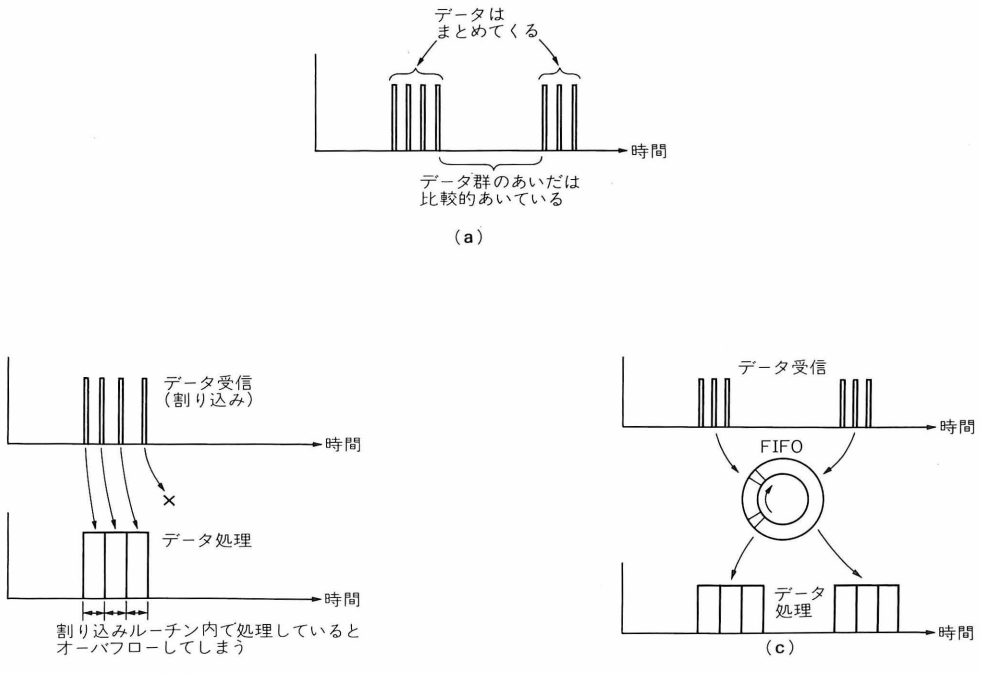
〔図3.19〕よくあるFIFOバッファの使い方



周期によって読み出しルーチンが呼ばれています．この場合には二つのFIFO処理ルーチンとは完全に別のところでメイン・ループが回っていて，それぞれの割り込みの発生時点でメイン・ループから抜け出して処理を行うことになります．

このような使い方の意味とメリットについて考えてみましょう．たとえばFIFOに積むデータは，通信ポートからの割り込みによって発生するとします．この場合，**図3.20**(a)のように受信データはある一定のかたまり(単語を構成する文字数とか，インテルHEX形式の1ライン分のデータなど)としてやってきて，それぞれのかたまりの間隔は比較的あいている(送信側でつぎのデータを用意する)のが普通です．そして，この1文字にあたるデータを解釈して対応する処理は，(b)のようにある一定の時間がかかるので，続けて何文字分もの処理を行うと，つぎの受信データにまで時間がオーバラップしてしまいます．ところで，よくある通信LSIのUARTなどの場合，ハードウェア内部にもっているのはダブル・バッファ，つまり2段バッファ程度が普通です．このため，受信割り込みルーチンの中で処理を行うと，データの密度が高いときには受信オーバフローというエラーが起きてしまいます．そこで，(c)のようにデータ受信のルーチンではFIFOにデータを積むだけの簡単な処理を行い，これとは別のルーチン(メイン・ルーチンとかタイマ割り込みルーチンなど)で，データの密度の低いときに処理を行う，という方法をとるわけです．つまりFIFOバッファとはその名のとおり，データ密度の疎密の差を吸収する，「緩衝」機能をもつことになるのです．この図からも明らかなように，データ読み出しルーチンをどのような頻度で呼ぶか，というのは想定されるデータ密度によって変わります．多少は経験からの慣れも必要かもしれませんが，適切なチューニングによって非常に強力なバッファとして活用できます．

〔図3.20〕FIFOバッファを使うケース



＜4＞

すべてはチップから：「チップ」関連技術

周辺 LSI

●周辺 LSI とは

マイコン・システムに欠かせないチップとして，各種の「CPU 周辺 LSI」があります．これはもともと，汎用 CPU の I/O 処理の周辺機能として，LSI メーカが CPU ごとの「ファミリ」として提供してきたものです．最近では，一般的な(似たような)機能をカバーするだけでは魅力に欠ける(付加価値として差別化できない)ためか，いろいろな特定機能対応汎用ハードウェア(ASSP)と位置づけ，各種通信規約対応 LSI やメカトロ関係 LSI などに，メーカ各社がユニークな機能の展開を競っています．それらの周辺 LSI は高密度の専用設計と大量生産によって，個別部品で組んだり，わざわざ ASIC 化するのに比べてかなり低価格に提供されています．

●「定番」周辺 LSI の例

周辺 LSI の代表格は，**図4.1**のような，インテル系 8 ビット CPU ファミリの周辺 LSI です．**8251** はシリアル通信(RS-232-C など)に使われる USART(Universal Synchronous Asynchronous Receiver-Transmitter)という LSI で，

- 通信データのシリアル-パラレル相互変換
- データの 2 重バッファリング(データ欠落を避けて，CPU の負担を軽減する)
- 各種プロトコルに対する専用の解釈・処理機能を搭載
- 各種動作モードをソフト的にプログラム可能

といった機能があります．このシリアル-パラレル変換を CPU のソフトやハードウェアで作ると，**図4.2**のような面倒なことになるのですが，この低価格の LSI を使えば，ブラックボックスとして非常にスマートに実現できてしまいます．

8253 は 3 本の 16 ビット・プログラマブル・カウンタを内蔵した，汎用タイマ LSI です．最近は 1 チップ・マイコンの多くがこのタイマ機能を内蔵しているので，やや活躍の場が減ってきていますが，CPU に対するタイミング信号だけでなく，単独の高性能カウンタとして，クロック発生器や一種のプログラマブル信号発生器としても重宝する LSI です．なお余談ですが，この「プログラマブル・カウンタ」という回路方式は，かなり幅広い範囲の周波数を発生できる反面，周波数帯域が高域になるほど精度(分解能)が悪くなる欠点があります．そして，まったく別の方式によるクロック発生回路で，この特性を正反対(周波数が高いほど高精度)にできるものもあります．興味のある方は，ちょっと考えてみてください(**コラム 解答 2** 参照)．

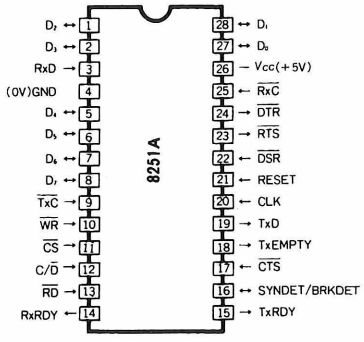
8255 は，基本的には 8 ビットの入出力ポートを内蔵したもので，374 と 245 が 3 個ずつ入っている LSI としても使えます．さらに CPU からのプログラムによって，ハンドシェイク動作をサポートしたり，1 ビット単位で入出力を切り替えたり，なかなか「すぐれもの」の機能をもつ LSI です．また，このほかにも，割り込みコントローラの **8259** とか，それぞれの LSI の CMOS 版などのラインアップも充実しています．

これまでの CPU 周辺 LSI には，上にあげたようないくつかの「定番」がありました．つまり，定番 LSI となって多くの設計者に採用されれば，LSI メーカは大量生産(工場が順調に稼働することが大切)のウマ味が得られ，ユーザも低価格の恩恵を得られる，というわけです．そこで，LSI メーカ各社は「つぎの定番」を狙って，高機能の周辺 LSI をつぎつぎに開発し，戦略的な低価格で発表しています．つぎにあげるのは，そう

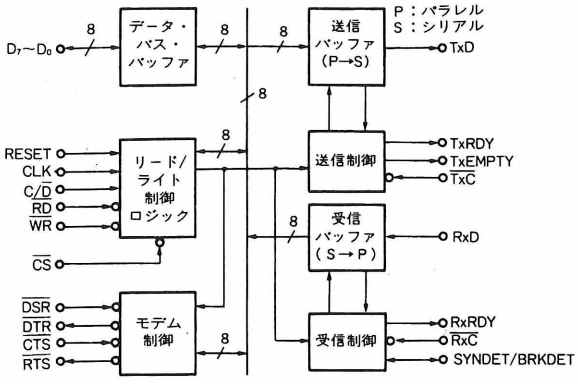
〔図4.1〕CPU 周辺 LSI

（『マイコン周辺 LSI 規格表』より）

■ ピン接続

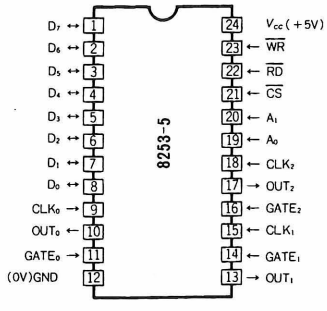


■ ブロック図

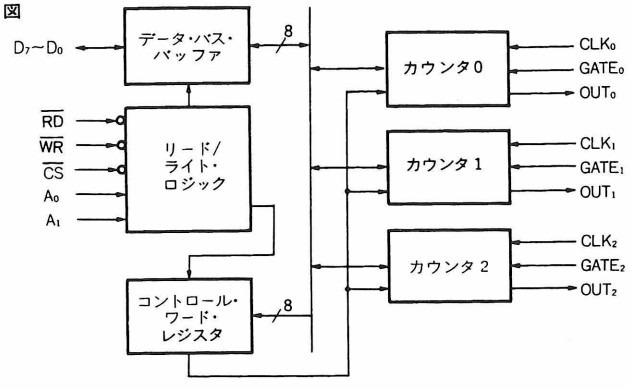


(a) 8251

■ ピン接続

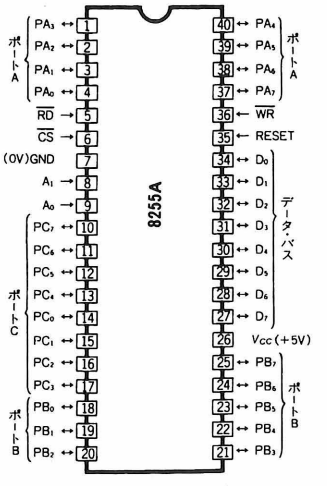


■ ブロック図

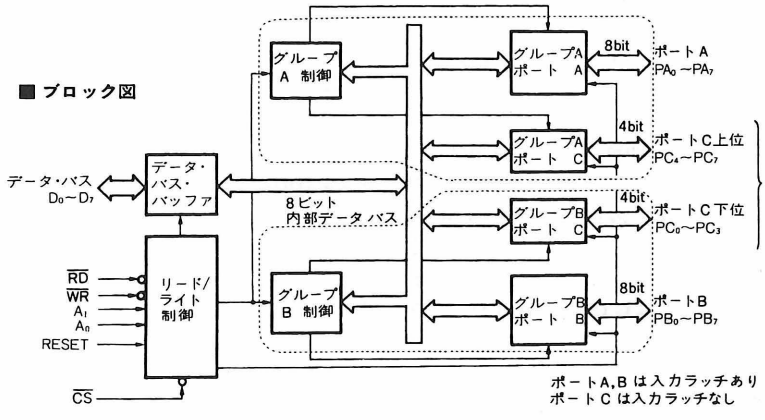


(b) 8253

■ ピン接続

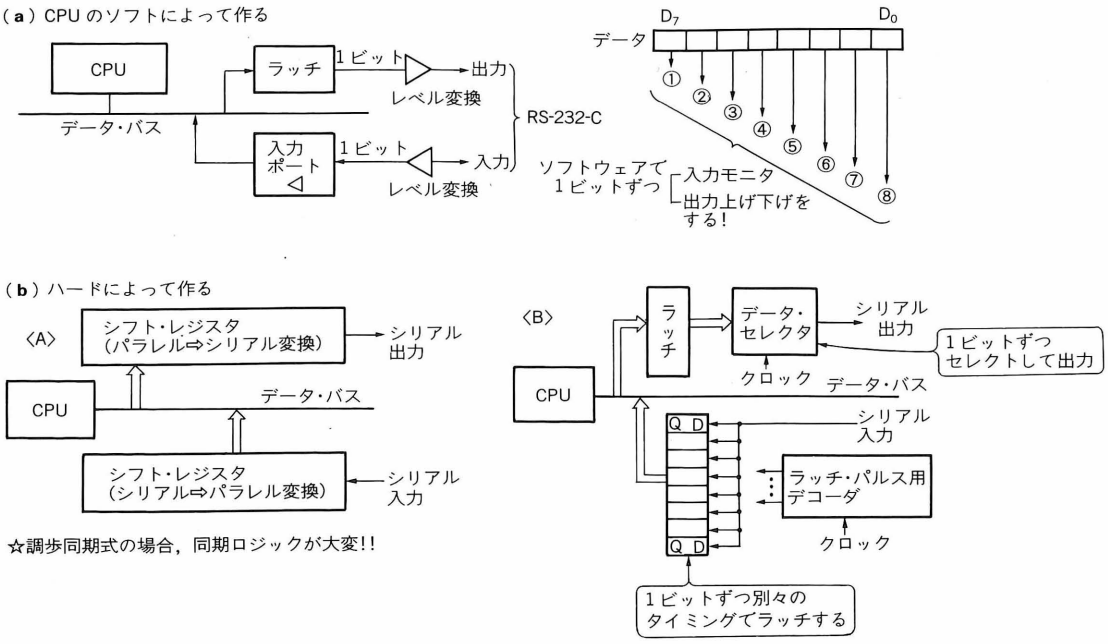


■ ブロック図



(c) 8255

〔図4.2〕RS-232-C をバカ正直に作ると…



［コラム］ 解答 2

可変クロック発生回路の例

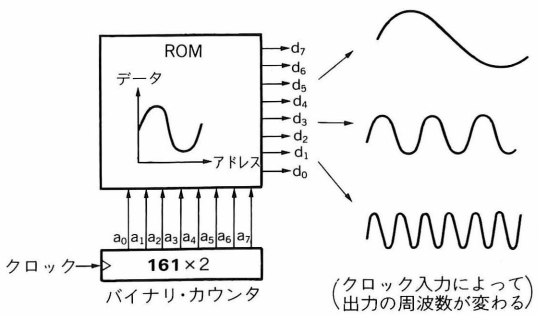
図Dは，ROM 内データとしていろいろな信号データを用意しておき，その ROM アドレスを 161 などのバイナリ・カウンタで発生することで，連続的に信号を発生しよう，という回路例です．この場合，出力信号の周波数というのは，バイナリ・カウンタに入力されるクロック信号によって決定されますから，このクロックをどのように「可変」にしてやるか，という部分が，出力信号の周波数の設定精度や分解能を決定することになります．

図Eは，8253 の内部にもある，「プログラム・カウンタ」(プログラマブル・カウンタともいう)という方式の可変クロック発生回路の例(実際は 16 ビット．ここでは簡単のために 4 ビット)です．アップ・カウンタによる例では，バイナリ・カウンタの出力がゼロから増加していって，あらかじめラッチに設定されている値と一致(コンパレータにより検出)すると，カウンタ値をクリア(リセット)するようにしています．また，ダウン・カウンタによる例では，あらかじめラッチによって設定された値をロードして，ここからダウン・カウントしていき，出力がゼロになると，ここで再び設定値をロードします．

この方式では，バイナリ・カウンタのマスタ・クロック入力を設定データで「割り算」(分周)していることになりますから，出力として高い周波数のデータを得たい場合には，設定データが小さな値となります．このため，分周の精度としては，割り算の分母が小さくなるほどデータ精度が悪くなって，出力周波数が高いほど，その誤差が増大するという特性になります．逆に，高精度の低速運転を制御するような場合には，この方式による可変クロック発生が最適です．

図Fは，「位相累算方式」とでもいえる(正式な名称は

〔図D〕ROM 読み出し方式のデータ発生回路

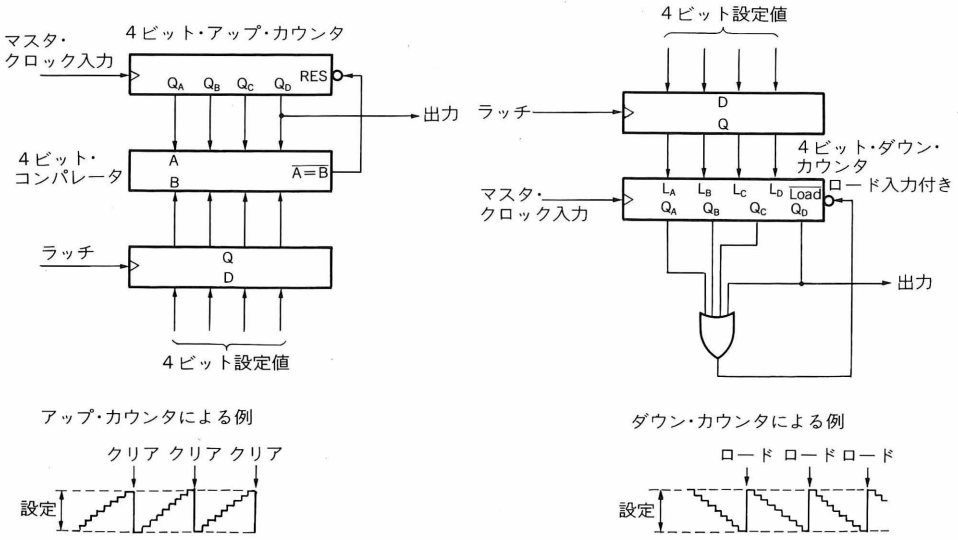


不明)方式による回路例です．ここでは，出力の周波数を設定するためのデータ・ラッチと，多ビットの加算器，そして多ビットのデータ累算用ラッチがあるだけです．マスタ・クロックは，データ累算用ラッチに周期的に与えられます．設定データは，一般には加算器のデータ幅よりもかなり小さく，このデータの下位に相当するように供給され，この上位部分にはゼロが入力されています．さて，まず，この回路の動作を理解してみましょう．

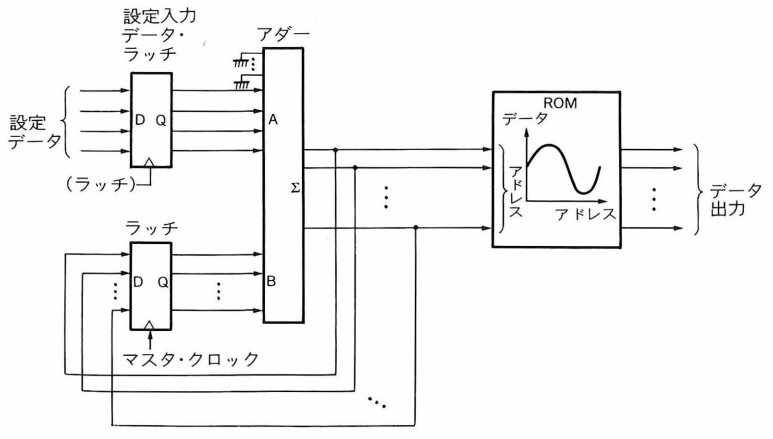
データ累算ラッチの出力と設定データとが加算されて，その出力が信号データROMに与えられるとともに，データ累算ラッチに入力されています．ここでマスタ・クロックが1発進むと，現在値と設定データが再び加算されるのです．信号データROMのアドレスとして使われるのは，一般にデータ幅のかなり上位のほうのビットなので，フルビットの設定データが加算されても，1回では，ほとんどROMアドレスは進みません．つまり，ここでの設定データというのは，ROMアドレスの「小数点以下」の小さなデータとして作用しています．そこで，周期的信号のごく小さな進み，ということで，この回路方式を「位相」の「累算」と名付けてみたわけです．

この方式では，設定データが大きいほど出力周波数が高くなりますから，設定データのビット幅としてフルに表現できる(周波数が高い)ほど，データ設定の精度が良好となります．つまり，プログラム・カウンタ方式とは逆の特性となっているわけです．また，この回路方式というのは，システム・クロックにしたがって回路全体を設計しやすいために，ディジタル信号処理回路の分野では定石的手法となっています．ここでは残念ながら深入りできませんが，「二つのラッチをRAMにする」という超重要テクニックが，この場合のヒントとなります．

〔図E〕
プログラム・カウンタの例



〔図F〕
位相累算方式の回路例



いったメーカ各社の「熾烈な競争」の一端です．

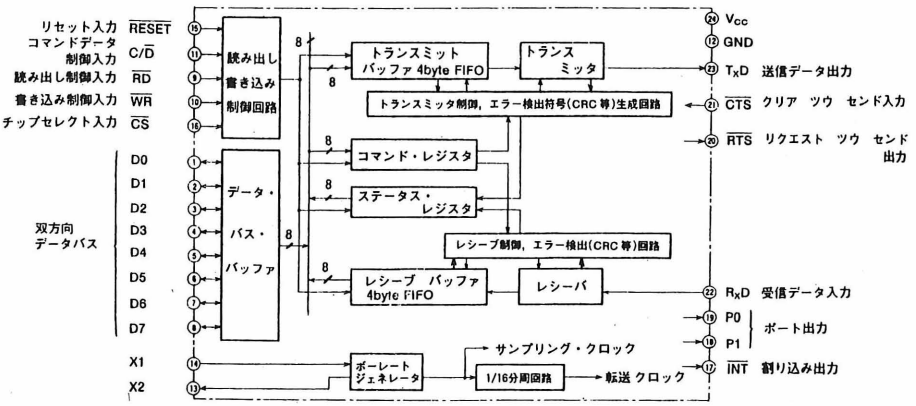
●各種の UART の例

図4.3にあるのは，「定番」8251 に対抗した各社の周辺 LSI のいろいろです．8251 は受信データ・バッファをダブル・バッファとして，CPU が自分の処理に忙しくて UART(Universal Asynchronous Receiver-Transmitter)からの割り込み要求への対応が遅れても，通信データを取りこぼさないようにしていました．三菱の M66230 では，なんとこれを 4 段にしています

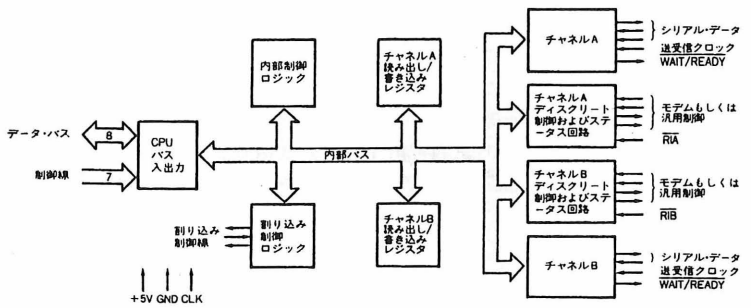
から，低速度の CPU でもかなりの通信速度まで対応できる，というメリットがあるわけです．

ザイログには Z80 SIO という 8251 に似た UART もありますが，この Z80 DART では，1 パッケージに 2 系列の UART をもっています．そしてリコーの RF5C59 になると，なんと UART を 4 系列も 1 パッケージに収めています．このチップは小型のフラット・パッケージですから，多系列の通信回線をもつシステムへの応用では，相当なコンパクト化が期待できる LSI といえるでしょう．

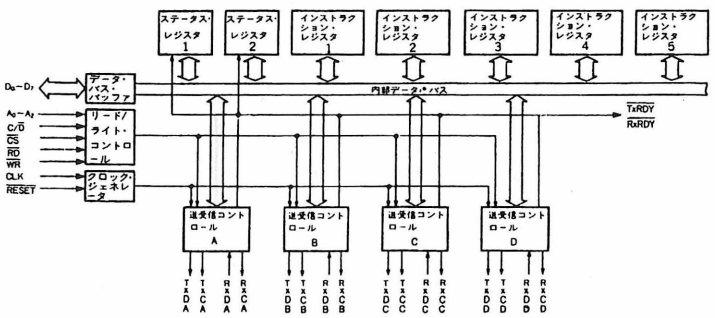
〔図4.3〕各社の UART の例
(『マイコン周辺 LSI 規格表』より)



(a) 三菱 M66230
(4段バッファ UART)



(b) ザイログ Z80 DART
(UART×2)



(c) リコー RF5C59
(UART×4)

●各種のPIOの例

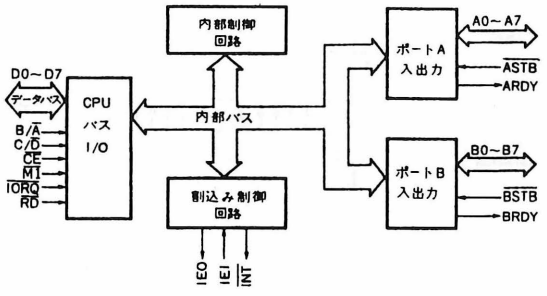
さらにすごい例をみましょう．**図4.4**は，各社のPIO(パラレルI/O)です．CPUの外部インターフェース・ポートとして，パラレルの入力ポートと出力ラッチ・ポートを増設するのは，ほとんどの分野のマイコン・システムにとって，もっとも基本的な周辺拡張の要求です．そこで，ここにはLSIメーカ各社の，涙ぐましいほどの参入が繰り返されています．シリアル変換のUART(各種プロトコルのサポート機能をもつ)と違って，PIOではプログラム設定による高度な付加機能はあまり見あたりません．そこで，各社は1パッケージにどれだけのビット数のポートを収めるか，という集積機能の勝負(そして，最後に残るコスト勝負)に出るしかないようです．

独立ハンドシェイク制御線付き8ビット2ポートのザイログ・Z80 PIOと，8ビット3ポートをもつインテル・8255(国内では日本電気・沖・東芝などがCMOS版でセカンド・ソース生産)が，しばらくの間は「定番」でした．これに対して，リコーのRF5C60では，8ビット5ポート＋5ビットという構成で，非常にコンパクトで多数ビットに対応した周辺拡張の可能性を提供しています．

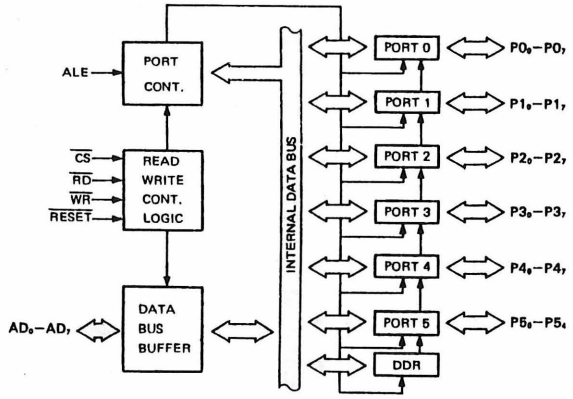
また三菱のM66500では，リコーとほぼ同じ規模のポート構成ですが，入力ポートにシュミット・トリガ入力(チャタリング防止に有効)とか，出力ポートに大電流駆動回路(高輝度LEDを直接ドライブ可能)を採用する，といった付加価値を搭載して，ユーザであるマイコン・システム設計エンジニアにアピールしています．

そして富士通のMB89363では，1チップになんと8255の機能を，小さなフラット・パッケージにまるまる2系列もってしまう(8ビット6ポート)という完全な「力ワザ」に出ています(三菱の同種LSIはシュリンクDIP版)．大手汎用LSIメーカならではの戦略ともいえますが，筆者はこのような競争を，いつも楽しんで眺めて(そして実際に活用して)います．

〔図4.4〕各社のPIOの例 ①

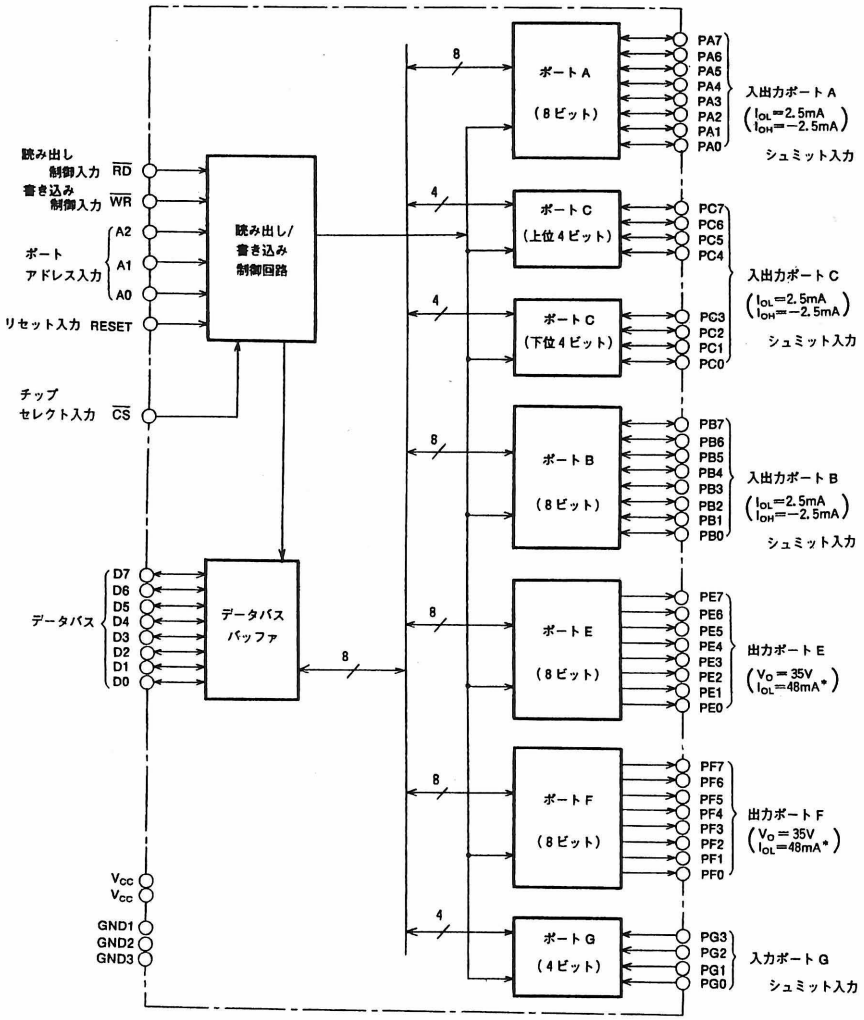


(a) ザイログ Z80 SIO

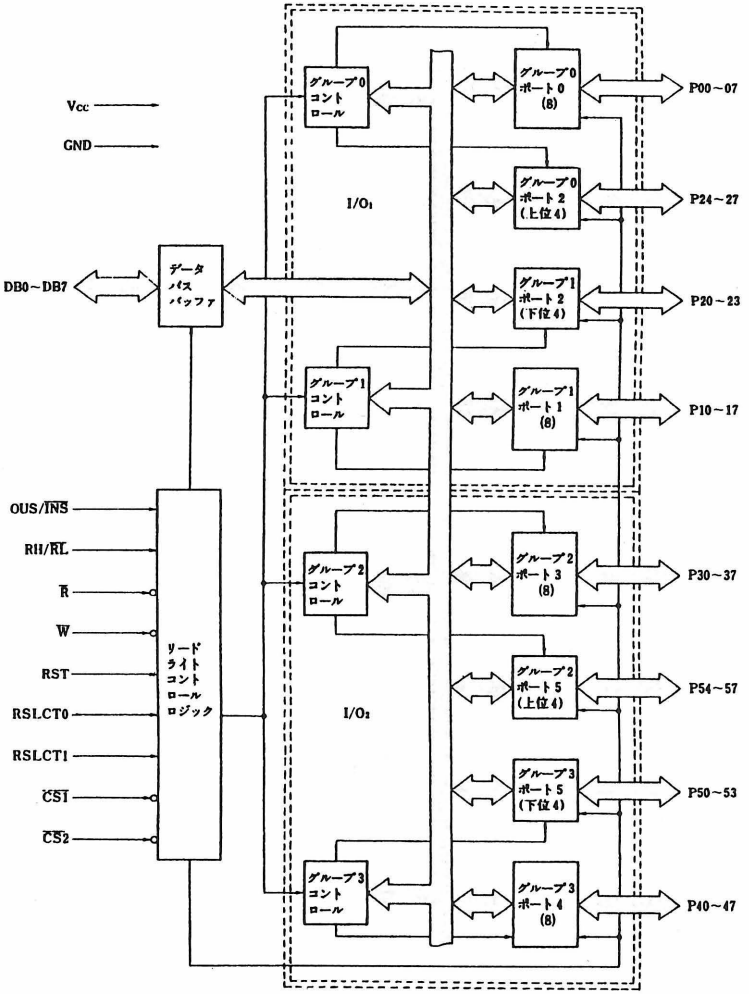


(b) リコー RF5C60 (8ビット×5+5ビット)

〔図4.4〕各社の PIO の例 ②



（c）三菱 M66500（8ビット×5＋4ビット）



（d）富士通 MB89363（8255×2＝8ビット×6）

メモリ

●メモリは最新情報が見逃せない

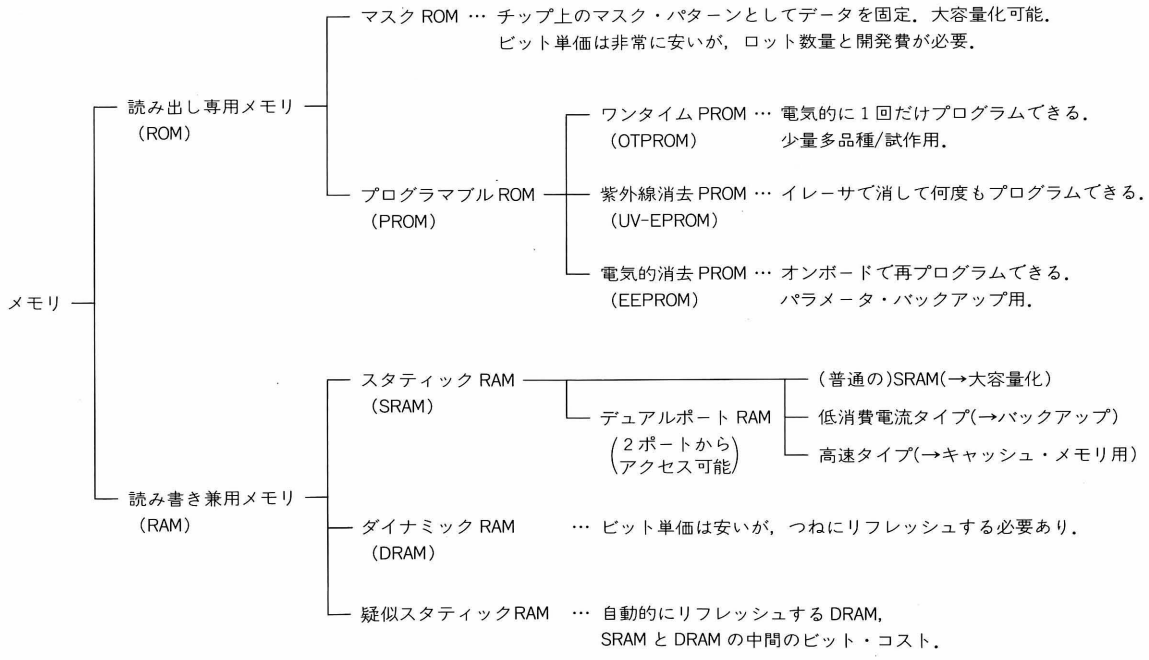
CPUと並んで「半導体技術の牽引役」である「メモリ」について考えてみます．一つのマイコン・システムにCPUやASICが1個か2個あるとすると，メモリ・チップはその数倍から十数倍の個数が使われる，というのが一般的です．そこで，LSIメーカ各社のミクロン・ルール(微細化)の進歩は，数量効果の大きい(設備投資を回収しやすい)メモリの製造ラインから始まります．この結果，各種LSIの中でも，メーカ各社の間でのメモリの新製品開発はもっとも競争が激しく，マイコン技術者にとっても，絶え間ない新製品の状況をチェックするのが必須の仕事となります．

一般的な電子技術に関する雑誌の記事でメモリが話題になるのは，飛躍的に性能を向上させたチップの実験・開発に成功した，といった海外での学会発表などの派手なニュースです．また，新聞の経済欄に登場するメモリの話題といえば，どのメーカのどの工場が新プロセスでの生産をスタートさせた，というような内容がほとんどです．しかし，マイコン技術者のところに日常的に飛び込んでくるメモリのニュースというのは，このどちらでもない，やや地味な情報(多くは営業マン経由の，「暫定情報」というコピー)です．つまり，「開発中」だったメモリがいつから受注可能になるとか，半年後にこの仕様のメモリを「発表予定」(計画中)とか，新製品の立ち上げ時によくある「機能制限情報」(データ・シートにあるうち，この機能は現在のところサポートされていません)などの刻々と変化する情報です．

しかし，これらの情報こそ，新しいメモリの機能を活用したいマイコン技術者にとって，もっとも重要なものなのです．たとえば，メーカA社とB社の情報しか入手していないエンジニアが実現できなかったあるシステムを，じつはこれより進んだメーカC社のメモリの情報を入手してうまく活用した別の会社のエンジニアが1年も早く完成した，という実話があります．情報の差が技術の差になってしまった好例です．

「先を見すえる」ということでは，メモリに関した面白い例があります．MS-DOS(8086)のパソコンでは，現在640Kバイトというメモリ空間の制限が致命的になっていますが，これは最初に設計したIBM・イ

〔図4.5〕メモリのいろいろ



ンテルのエンジニアにとっては，当時のメモリICから「想像できないほど十分に」巨大なメモリ空間(64Kバイトのセグメントが10個もある)だったのです．これに対して，モトローラのCPUはほぼ同じ時代から，メモリICの技術状況を別にして，当時としてはまったく夢のような「24ビットのリニアなメモリ空間(16Mバイト)」を想定した，スッキリとした設計を最初から貫いてきて，これがUnixやMacの系列を生み出しました．技術者として大切な視点をよく示したエピソードだと思いませんか．

●メモリ技術戦争のもうひとつの側面

新しいメモリは，基本的には性能(容量・速度・消費電力など)は向上していますが，量産が開始された最初は価格が高く，供給も不安定な場合があります．それなのに，なぜ新しいメモリを重視するかというと，進歩の裏で「陳腐化」が進んでしまうからです．「従来使っていたメモリをつねに採用すれば，回路の実験も不要で価格も安いだろう」というのは，プロとしては失格の甘い話なのです．

新しいメモリを早く軌道に乗せるために，あるいは旧式の生産ラインを改善していくために，LSIメーカは十分に価格のこなれた(競争と量産効果によって安くなった)旧機種のメモリICを，まだ市場が必要としているのに製造しなくなってしまう場合があるのです．このため，たとえば「半年後に出す新製品のために，すでに1～2年の実績のあるメモリを使う」というのは危険なこととなります．いざ生産に入ろうと思った矢先に「あと半年で製造中止」などと通告されかねません．そこで，半導体業界の動向(国内とは限らない)に情報アンテナを向けて，各種メモリの研究・開発・生産の状況をつねに知っておくことを心がけておきましょう．

また，**図4.5**のように，RAMといってもSRAMやDRAMに加えて，最近は疑似SRAMとかデュアルポートRAMでも，多くの種類のものが登場しています．ROMにしてもマスクROMとUVEPROMだけでなく，ワンタイムPROMやEEPROMがあり，メーカ各社はオリジナリティと付加価値の勝負を展開しています．ユニークなメモリは価格も高い(数量が出ないため？)のですが，その付加機能をうまく活用すると，逆にシステムのコスト・ダウンに貢献する可能性もあります．実際に使うかどうかは別として，いろいろなメモリを日頃から実験してみることも大切です．

1チップ・マイコン

●1チップ・マイコンとは

組み込み機器に幅広く活用されているCPUとして，あまり記事などでは目立たないながらもメジャーなのが，いろいろな「1チップ・マイコン」です．これは，CPUチップとともに，ROMやRAMなどのメモリ，シリアル・ポートやイベント・タイマやA-Dコンバータなどの周辺機能，DMAコントローラや割り込みコントローラなどを，すべて1パッケージに搭載してしまったタイプのCPUです．

1チップ・マイコンの登場してきた背景にはいろいろあります．半導体メーカの微細化(ミクロン・ルール)の進展から，チップ上にCPU以外の回路を詰め込むことが可能になったことが直接の要因ですが，それ以外にも，日本国内の原因として，

- 多くのチップを使いたくないセット・メーカからのコスト的な要求
- セカンド・ソースのマルチチップCPUを生産するだけでは「うまみ」が少ない(価格競争に負けると他社に流れてしまう)
- 次第にアメリカのオリジナル・メーカがセカンド・ソース権を与えなくなってきた

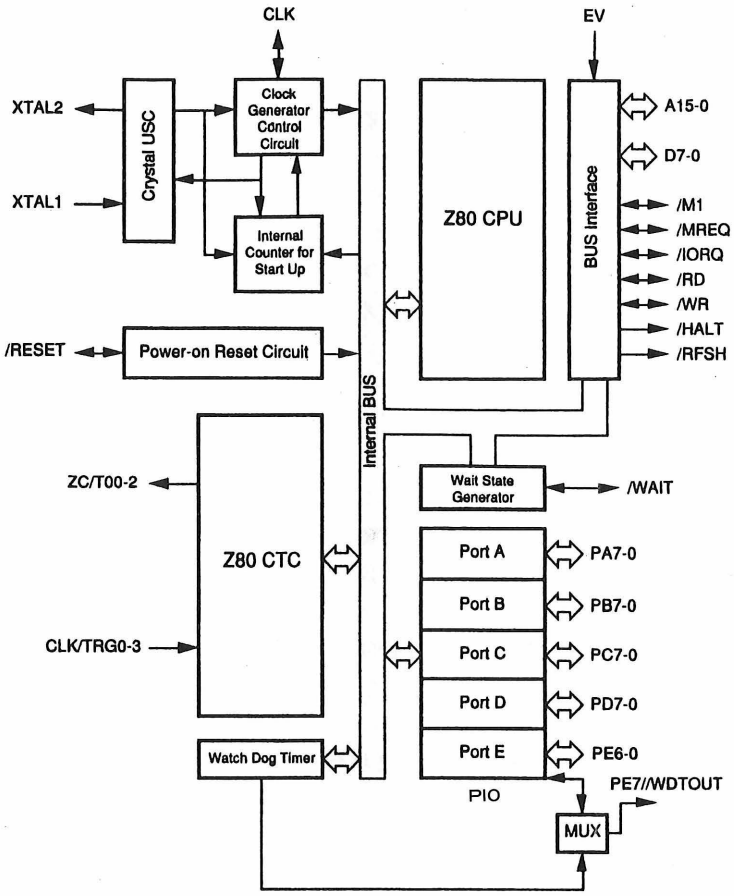
などの理由が考えられます．しかし，国内半導体メーカ各社の「オリジナリティ戦略」の結果という側面も大きいと思います．

どんな電子機器にもCPUが使われる，というテクノロジー文化が浸透してしまうと，CPUメーカの戦略としては，ローコスト競争とオリジナル機能競争しかありません．前者は生産数量と設備投資の勝負という簡単なものですから，力点としては後者の，いかに付加価値として機能を強化したCPUを提供するか，というアイデアの勝負となってきたわけです．

●1チップ・マイコンのバラエティの例

LSIメーカ各社が腕によりをかけてアイデアを展開している各種の1チップ・マイコンのバラエティについて，以下にその一例を示してみましょう(とても全体を把握しきれませんので，個々の機種名は書きません)．

〔図4.6〕 1チップ・マイコンの例
Z80＋周辺タイプ
(ザイログ)



- チップ上のメモリとしてマスクROMやRAMだけでなく，ワンタイムROMやEEPROMをもつもの
- CPUとして既存のチップと上位互換性をもたせたもの
- 8ビットながらCPU内部を16ビット構成として，高性能の乗算命令をサポートしたもの(オリジナルのCPUコア)
- メモリ空間を64Kバイトから数Mバイトに拡張したもの
- CPUも周辺LSIも，ポピュラな品種とまったくコンパチブルなものを一つのパッケージに入れたもの(後述のザイログの1チップがこの例)
- CPUのレジスタ・モデルとアセンブラのニモニックが同一なのに，じつはオリジナルCPUで，アドレシング・モードや割り込み機能が大幅に強化されているもの

具体的な例としてひとつだけ，**図4.6**のザイログのチップを眺めてみましょう．といってもじつは，これは東芝のチップをザイログがセカンド・ソース生産しているものです．ザイログのZ80をセカンド・ソース生産している東芝ですが，Z80をコアとして周辺まで1チップ化したのは東芝で，これが逆にザイログにライセンスされている，という面白い例なのです．Z80コンパチのCPUとともに，タイマ・SIO・PIOやクロック・ジェネレータ，さらにはリセット回路やWDT(ウォッチドッグ・タイマ)回路まで搭載されています．国内の各社から発表になっている「名刺サイズのボード・マイコン」の多くがこのチップを採用していますが，単体のZ80を使ったのでは，ハガキ大以上の大きさになってしまいますから，このチップが初めて登場したときのインパクトは相当のものでした．

● ROM化プログラムに対応した機能

パソコンはメイン・メモリをRAMにして，アプリケーション・プログラムはディスクからロードしてく

るという，(BIOS･ブート ROM を除いて)基本的には「クリーンな」コンピュータ・システムとなっています．ところが，制御用システムなどの「組み込み機器」の場合，プログラムは ROM 化され確定していて，ソフト的にロードすることのない「ファームウェア」(固定したソフトウェア)と呼ばれます．このようなシステムで効果的なのが 1 チップ・マイコンで，もともとは自動車・事務機器・家電製品などへの組み込み用に開発されて，ユーザである特定メーカ専用のマスク ROM を搭載したのです．

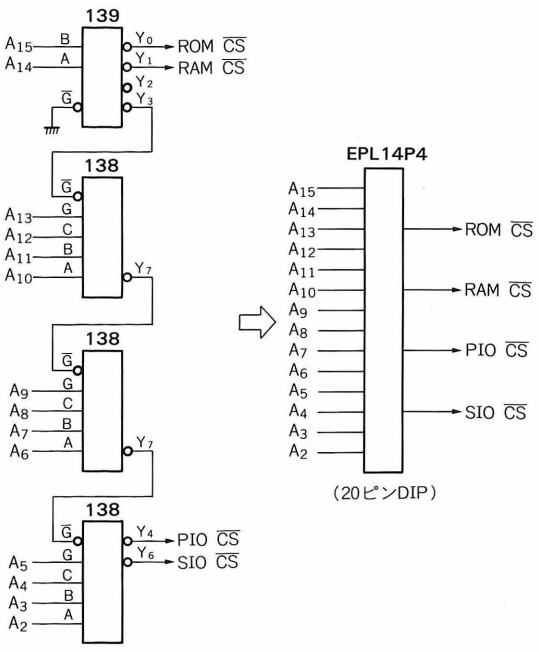
ところが現在では，1 チップ・マイコンは実験用・少量多品種製品などの，非量産ベースでも使用されています．この場合，チップ上のマスク ROM を焼くだけのロット数量になりませんから，この ROM 部分が無駄になります．しかし，たとえ EPROM を外部の拡張領域に別に置いても，チップ上の周辺 LSI 機能(タイマ・シリアル通信ポート・A-D コンバータなど)を別に用意するよりは，はるかにコスト的・スペース的に有効であるほどに，チップ上に提供される付加価値が豊富になってきたのです．

また，最終的にマスク ROM 化する場合(このタイプがもっともローコスト)でも，開発段階や試作のために，ROM 部分以外が完全に同一なファミリとして，

- 内蔵 ROM を取り去った外付けタイプ
- ROM をワンタイム EPROM として実験室でプログラムできるタイプ
- パッケージ上面に EPROM ソケットのあるピギーバック型

などのバリエーションが，たいていのメーカから出ています．これは，マスク化するつもりのないユーザにとっては好都合で，1 枚の CPU ボードの機能を一つのチップで容易に実現することができます．

〔図4.7〕PLD の利用例 (1)── デコーダとして



セミカスタム IC

● PLD(Programmable Logic Device)

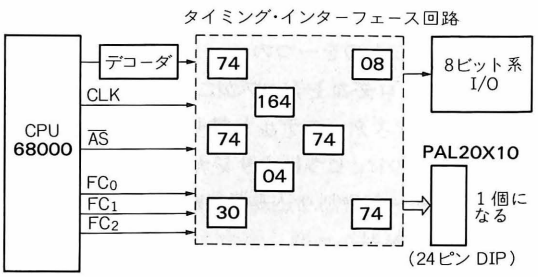
ここでは，ユーザの手元でハードウェアの一部をプログラムできる，という「セミカスタム IC」について考えていきましょう．まずは，PLD(PAL というのは商標)です．これはデバイスの一種なのですが，見方によっては，設計・開発作業を支援するツールとも考えることができます．

PLD の簡単な使い方の例としては，**図4.7**のような，CPU のアドレス･デコーダとしての活用がポピュラなものです．これは，デコーダ IC の 2 ～ 3 個分が 1 チップに収まるだけでなく，

- 自由なデコード論理を簡単に(論理式で)指定できる
- 仕様を変更してもハードが変わらない

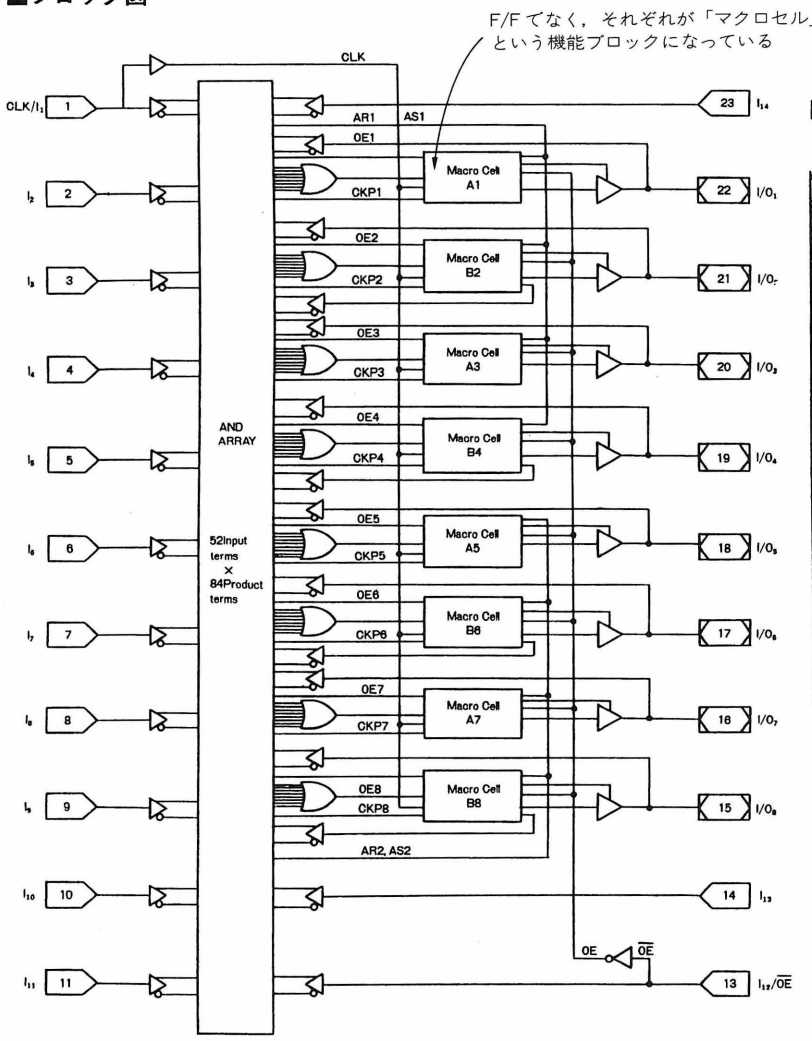
というメリットがあります．また，**図4.8**のように，フリップフロップなどの順序回路を内蔵したものでは，CPU 周辺のタイミング回路(クロック・ジェネレータや DRAM コントローラ)のロジック部分を簡潔な 1 チップにできる場合もあります．

〔図4.8〕PLD の利用例 (2)── CPU 周辺として

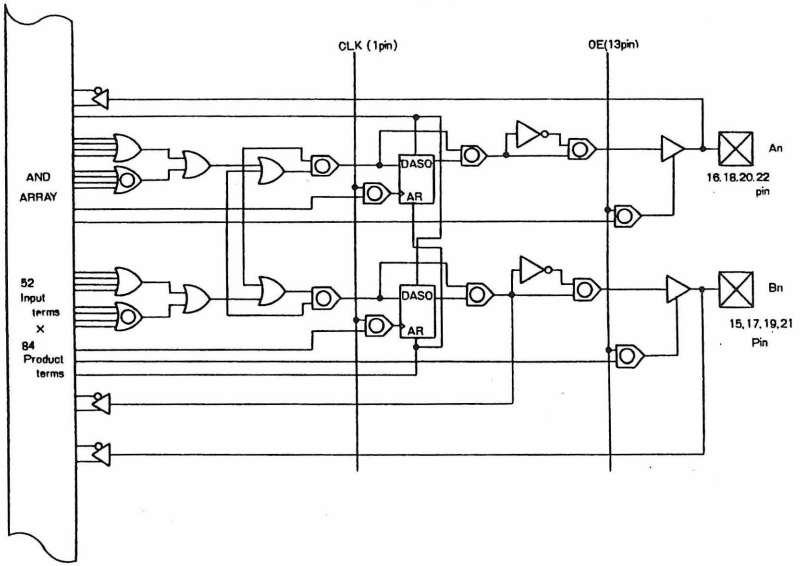


〔図4.9〕大規模・高機能 PLD の例(リコー・データブックより)

■ブロック図



■I/Oマクロセル ブロック図（これが８つある!!）



さらに，セキュリティ・ヒューズによって内部の秘密保持を行うタイプのものは，コピー・プロテクトや海賊版対策として，PLD内のデータを一種の「暗号」として活用したもので，技術的というよりも政策的な要請で採用されることがあります．

これはちょっと極端な例でしたが，一般的なPLDのメリットとしては，

- パソコン上の設計ツールによって，回路をソフト的に設計してロジック化できる
- 紫外線消去型PLDによって，ハンダ付けなしにいろいろな回路を実験できる
- 過去に設計した回路情報をファイル化して保存・再活用できる

などがあります．また，少量多品種・高付加価値製品の一つの「部品」として，試作だけでなく実際に生産に使ってしまう場合も増えています．

●高機能PLD

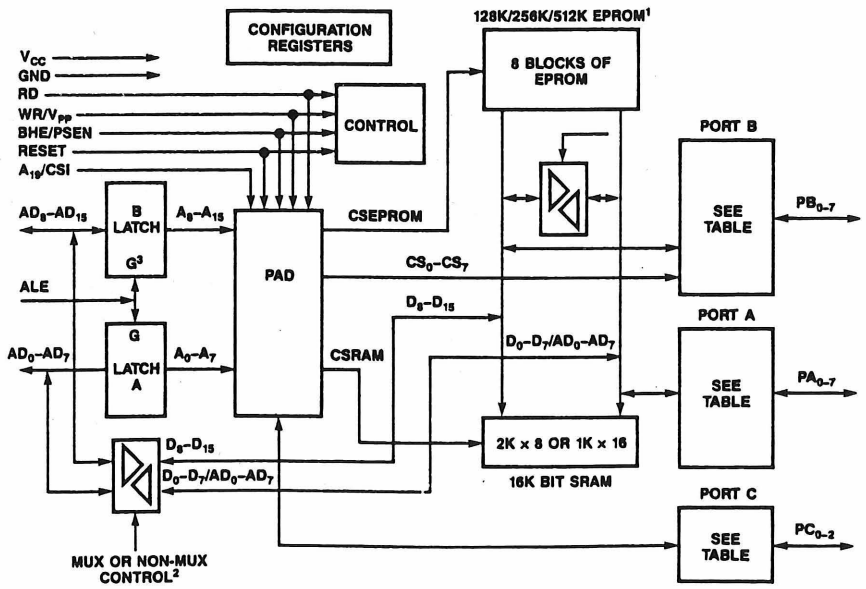
図4.9のように，最近は大規模なPLDや，ラッチなどの順序回路要素をもったPLDもかなりの種類が出回ってきました．この例では，従来のAND-ORアレイだけでなく，その出力にマクロセルというブロックをもっています．そして，

- フリップフロップと共通クロック信号をもって同期動作ができる
- AND-ORアレイに信号をフィードバックできる
- 出力の3ステート制御ができる(データ・バスに直結可能)

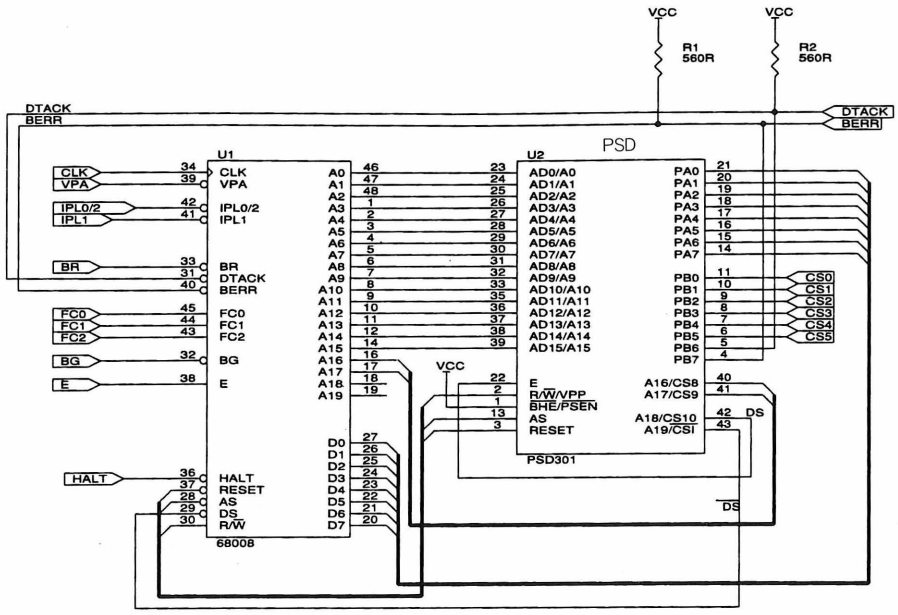
などの機能が強化されています．

また，WSI(ウエハスケール・インテグレーション)社のPSDというチップ(**図4.10**)は，アドレス・デコード回路をプログラマブルのEEPROMとして用意するとともに，RAMやタイミング回路などのCPU周辺回路をすべて1チップ化して，さらにバス幅などのCPUタイプごとにファミリを揃えたり，インテル系，モトローラ系の両方のCPUに対応できるように設計されている(**図4.11**)，という面白い展開をしています．「RAM内蔵の高機能PLD」というのか，「ユーザ・プログラマブルのASSP」というのか，そのアイデアのユニークさには脱帽です．

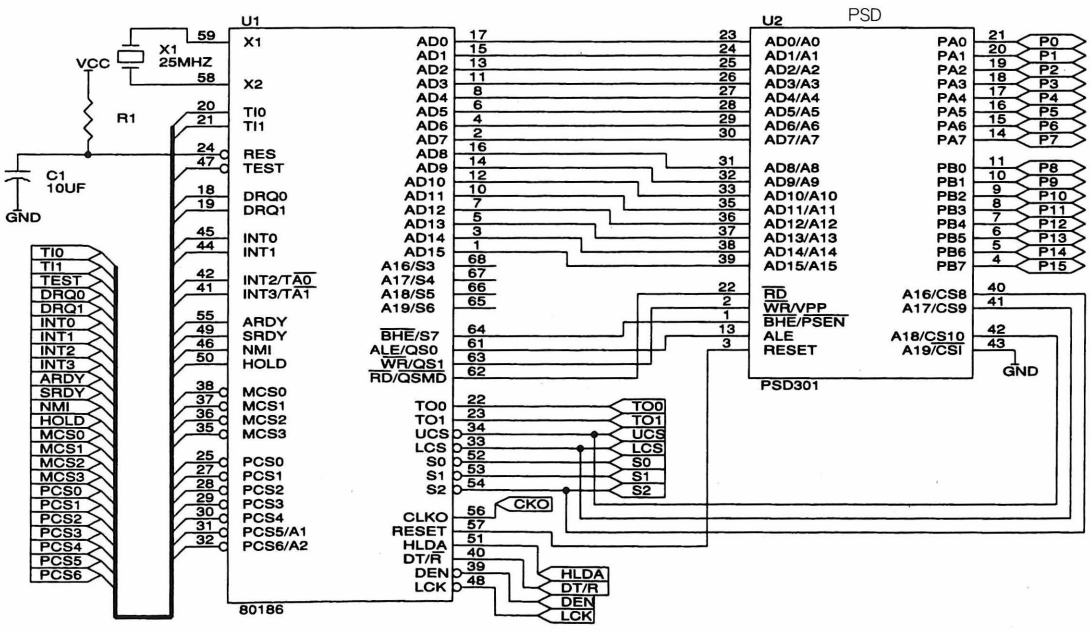
〔図4.10〕PSDの例(ウエハスケール・インテグレーション社データブックより)



〔図4.11〕 PSD による CPU 周辺 1 チップ化(ウエハスケール・インテグレーション社データブックより)



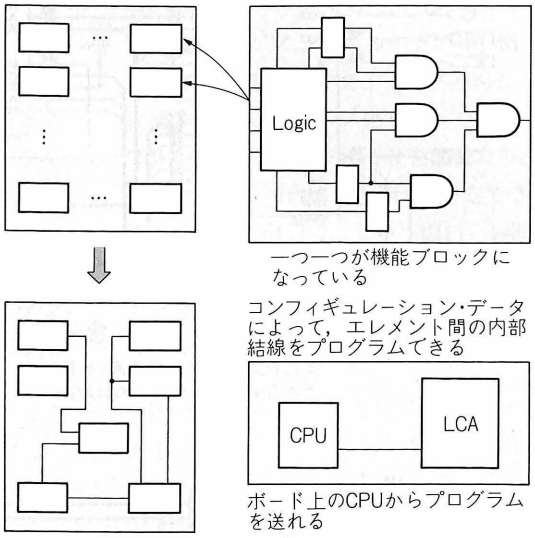
同じ PSD が，インテル系/モトローラ系の両方に対応できるようにあらかじめ設計されている



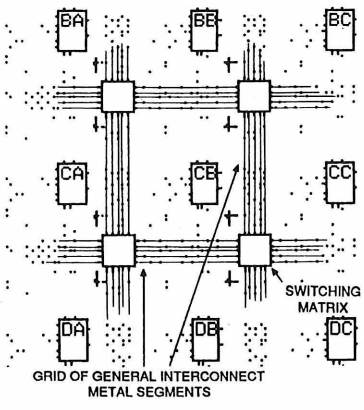
● LCA(Logic Cell Array)

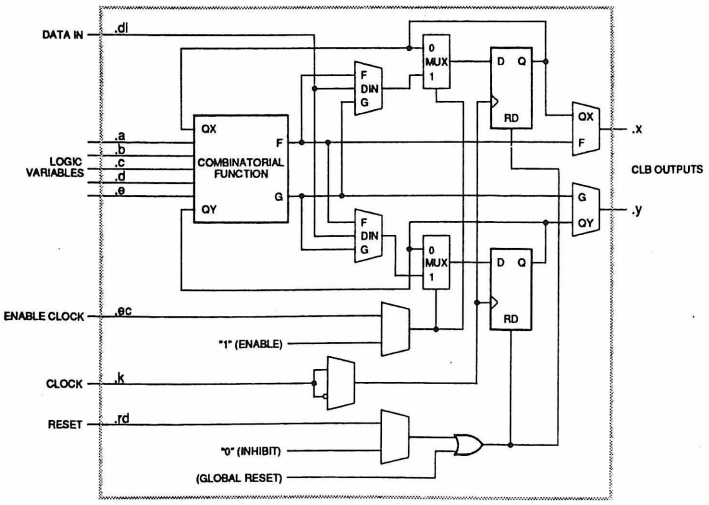
高機能PLDの延長としては，**図4.12**のような，内部の要素同士の結線情報を外部からロードして大規模な回路を実現できる，LCAというデバイスも活用されています．これは**図4.13**のように，チップ上には，CLBという基本単位のロジック回路ブロックが多数並んでいて，これらを組み合わせるとたいていの回路機能を構成できるようになっています．また，入出力ピンの部分にも，IOBという，ラッチや3ステート制御機能をもったブロックが置かれています．そして内部の多数のブロックを相互に結びつける配線ラインがあり，その交点の部分のスイッチ・マトリックスによって，チップ上の信号ラインを自在に結合できます．この「結線情報」は，LCA外部に置かれたEPROMからロードしたり，あるいはCPUから転送することで，自由にプログラム変更できるようになっています(**図4.14**)．

〔図4.12〕LCAのしくみ



〔図4.13〕LCAの例(ザイリンクスのデータブックより)

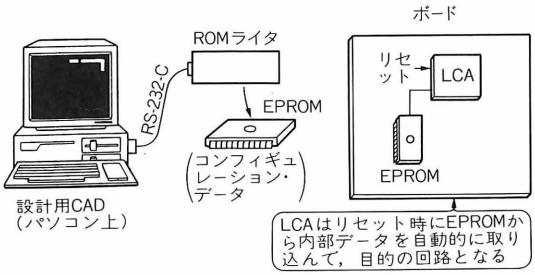




CLB：基本となる各ブロックのセル

最近のLCAは大規模なものが多く登場していて，ゲート規模では3000ゲートとか4000ゲート以上，ロジック部分のゲート遅延もナノ秒オーダなど，ほとんどASICのような世界になっています．また，設計を支援するツール(CADやシミュレータ)も，ASICの設計ツールとほぼ同じものが完備してきています．EWSレベルのコンピュータ周辺機器・高級なディジタル計測器など，性能のためには価格が多少高くてもいいという製品を中心に，すでにLCAは「部品」としてかなりの採用実績をあげています．また，回路の大部分を収めて，コンパクトにするためにLCAを活用する(**図4.15**)だけでなく，いずれはASIC化しようとする回路のブレッド・ボードにLCAを使い，ASICの欠点である開発期間を「つなぐ」という場面も多くあります．

〔図4.14〕LCAの開発



● LCAの採用とその限界

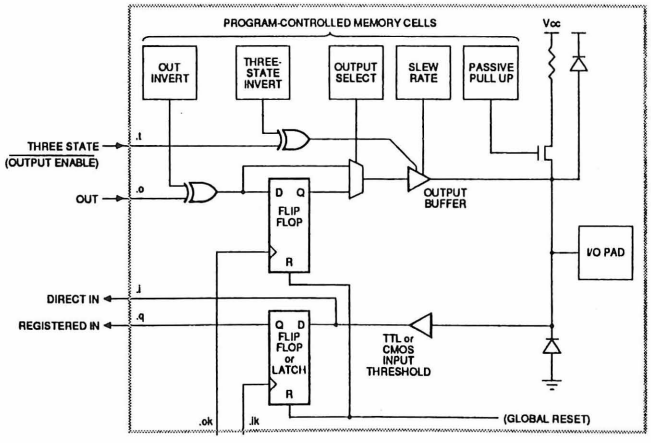
このように画期的なLCAですが，現在のところ短所もあります．

その第1は**コスト面**です．標準ロジックICで組むよりもかなり高価なチップなので，実装密度とか類似回路の展開(プログラムを少し変更して何種類も作る)などの，システム開発上の別のメリットがないと，なかなか採用できません．

第2に，メーカの宣伝している**ゲート密度**は理想的すぎて，実際には相当の時間をシミュレーションに割り当てても，なかなか有効ゲート数が上昇しません．つまり「4000ゲートのLCAといっても，実際には700ゲート分も入らない」といった例のような冗長度があります．

そして第3に，内部配線の影響による**ゲート遅延**のばらつきがあります．このため，高速でタイミングの微妙な部分では，ブラックボックスの不安があってなかなか使えない，という場合があります．

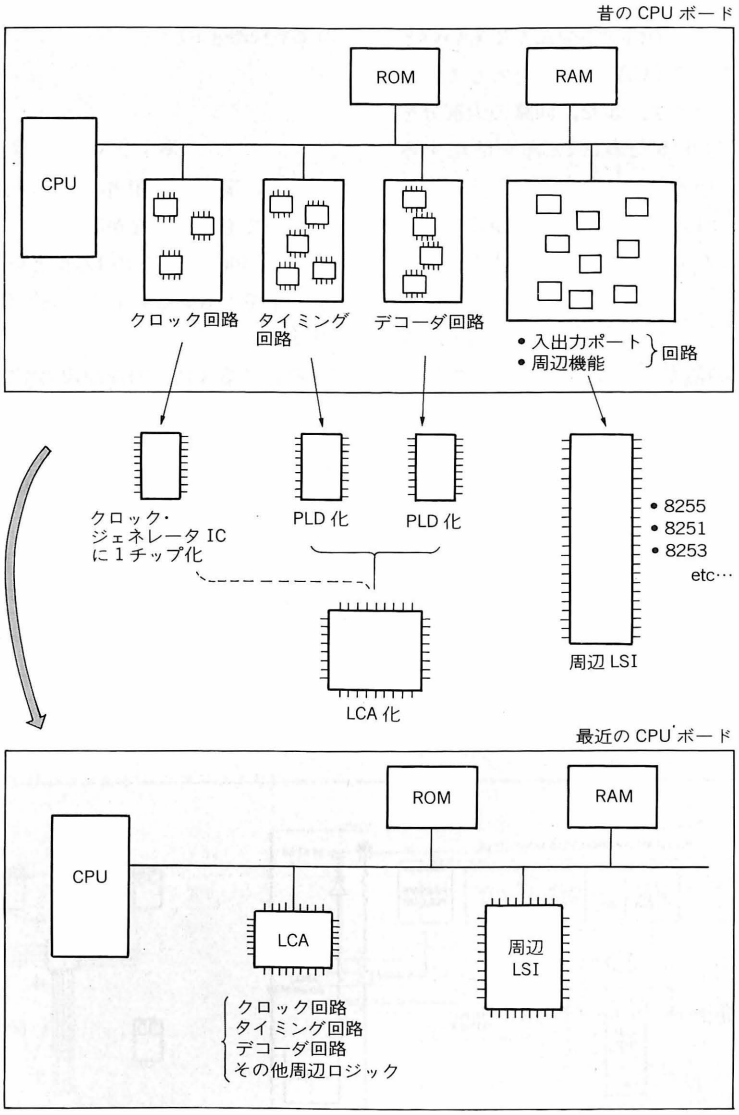
これは，しだいに開発ツールの機能向上によってカバーされていく部分もありますし，どうしても本物のASIC(ゲートアレイなど)にかなわない部分もあります．しかし，これからのマイコン技術者の「持ち駒」としては，すこし前のエンジニアには想像もつかなか



IOB：入出力ピン部分のセル

った「強力な材料」であるのは確かでしょう．実際に使える場面に遭遇するのがいつかは別として，PLDからLCAまでの，これらの「**ユーザ・プログラマブルIC**」を武器にしていくためにも，ここでも「チップの最新情報収集」という姿勢を大切にしたいものです．

〔図4.15〕PLDやLCAによるCPUボードのコンパクト化の進展



＜5＞ 「信号」技術から「信頼性」技術へ

アナログ←→ディジタル変換の考察

●アナログ←→ディジタル変換回路

マイコン・システムのCPU回路は，ソフトウェアという「論理」にしたがい，電圧レベルも＋5 VとGNDの2値をとる，ディジタルの世界です．一方，外界はアナログですから，マイコン・システムの周囲とのインターフェースでは，アナログからディジタルへのA-D変換回路と，ディジタルからアナログへのD-A変換回路は，ともに必須のものとなります．この技術を身につけていくためのステップとしては，

- 信号のもつ「物理量」，「時間」の二つの要因
- 変換の「離散化」，「量子化」という二つの原理

をまず理解し，そのうえで，これらを統合することで，

- 信号データの「アナログ」，「ディジタル」のそれぞれについて上の4種類の形態を理解すること

という最大のハードルを越えることができます．この先にあるのは，

- 実際にこの変換技術を活用した回路を設計できること

という目標になります．この基本的技術は，ハード屋，ソフト屋の区別なく，マイコン技術者のすべてに必要なものでしょう．

もっとも最後の実戦的な項目については，A-D回路

〔図5.1〕信号データの形態についての考察

（a）アナログかディジタルか

☆この2種類，というのは正しくない！
データ量の次元と，このデータ量の変化する時間軸の次元との2次元で考えると…

① データも時間もアナログ → （例）アナログ・テスタの針の動きをずっと見ている場合
② データはディジタル，時間はアナログ → （例）各ビットの重みに対応したアナログ・コンパレータの出力をずっと見ている場合
③ データはアナログ，時間はディジタル → （例）一定時間ごとにアナログ・テスタの数値を記録する場合（BBDも）
④ データも時間もディジタル → （例）データをA-Dコンバータで取り込んだ場合（量子化＋サンプリング）

（b）信号の物理的性質

① 電圧
② 電流
③ 抵抗
④ 光 ― 波長
　　― 光量
etc… ― パルス変調
　　⋮

（c）ディジタル信号の場合

① パラレル ← 高速のメリット
② シリアル ← 信号線が少ないメリット
　} 逆はデメリット
③ 多値ディジタル（各ビットが4値とか8値をとる）
④ 各種変調 ― PWM（パルス幅変調）
　　― PCM（パルス・コード化変調）
　　― PFM（パルス位相変調）
　　⋮

やD-A回路を，トランジスタやOPアンプやコンパレータを使って具体的に設計することは，必ずしも最初から必要ではありません．最近は高性能でローコストなサンプル・ホールドIC，A-Dチップ，D-Aチップがいろいろと供給されています．これは，ICメーカとしても市場性を認めて各社が参入し，また各セット・メーカの側でも，システムの付加価値のために多くのセンサとともに使用する，という相互作用によって，量産効果と競争原理が実現されてきているものです．そこで，まずはブラックボックスとして専用チップをどんどん利用し，しだいに本格的な変換回路を身につけていく，という作戦でも，プロとして十分に通用することでしょう．

●信号データの形態についての考察

それではまず，信号のもつ「物理量」と「時間」という要因について考えてみましょう．**図5.1**(a)のように，一般的に「信号データ」というのは，その値がアナログかディジタルか，と単純には分類できません．つまり，自然現象としてのデータは，必ず「時間」とともに記述されるものですから，データ(物理量)の次元と時間の次元の二つについて，それぞれアナログ(連続的)かディジタル(離散的)かで，合計4種類の組み合わせを考える，というのが正しい視点なのです．それぞれの「例」を参考にしてイメージしてみてください．

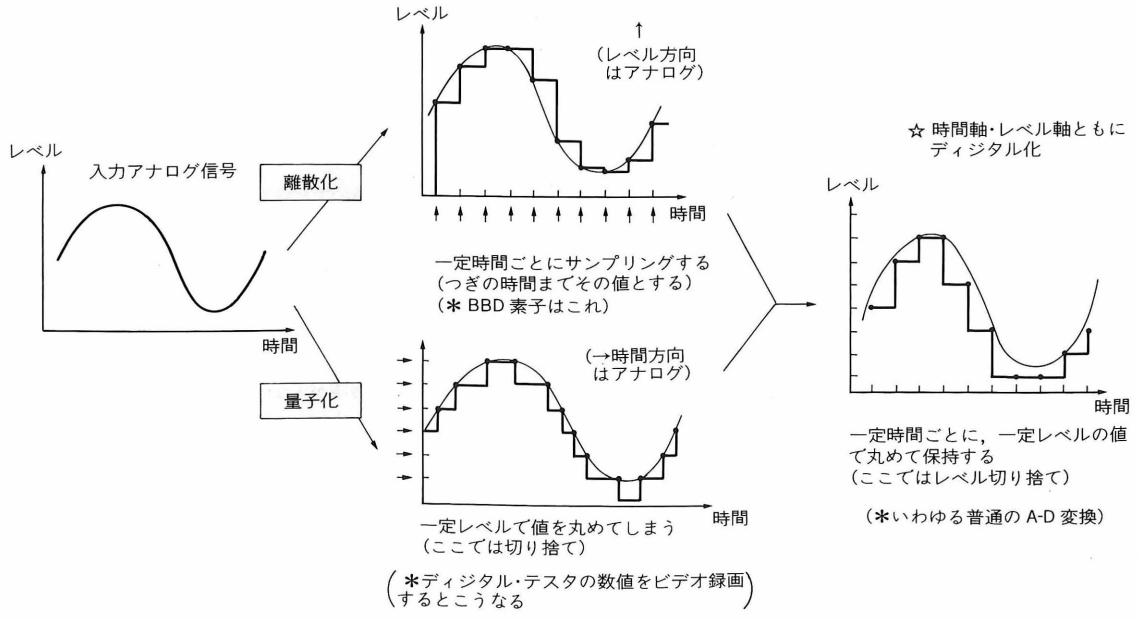
また，**図5.1**(b)のような，信号データそのものの物理的な性質による分類とか，**図5.1**(c)のように，ディジタル信号として表現する場合でも，じつはいろいろな方式があります．2値をとるのがディジタル，と思ったらおおまちがいで，たとえば3値(+Vと−VとGND)をとるディジタル信号というのは，2進数でなく3進数の論理にしたがうもので，身近な例では，通信に使う「モデム」が一種の多値信号を活用しています．

● A-D変換の「離散化」，「量子化」の例

それでは基本的な確認ですが，**図5.2**を使って，A-D変換によってアナログの入力がディジタルとなる様子を考えてみましょう．入力のアナログ信号でみると，「レベル」の軸は，電子回路では「電圧」という物理量であることが多く，もう一つの軸はもちろん「時間」で，ここでは両方とも連続しています．

ここで，「時間」軸の方向に関して，時間的にとびとびの地点(通常は等間隔)で代表させるもの(図の上の流れ)が「離散化」です．たとえば，カラオケの「電子エコー」に使われるBBD素子はこの動作を行うもので，クロック信号で切り刻まれた各瞬間の電圧を，多段のコンデンサが蓄積していくものです．この場合，それぞれの瞬間の電圧は連続的なアナログ電圧のまま

〔図5.2〕アナログ-ディジタル変換の原理



です．また，この電圧値を確実にディジタル化するために保持するものが，S&H(サンプル・ホールド)回路です．

また，逆に「レベル」軸のデータ量をとびとびの値で代表させるもの(図の下の流れ)が「量子化」です．たとえば，ディジタル表示の温度計というのは，実際には刻々と変化している温度を，たとえば1度とかの単位で丸めています．このディジタル温度計をじっと連続して眺めている人間，あるいはその表示をずっとビデオ撮影した，という状況を考えれば，理解できるでしょう．厳密にいえば，「レベルをある値に丸める」という作業のために必要な「変換時間」というものがありますが，ここでは単純化しています．

●情報圧縮による本質的な問題点

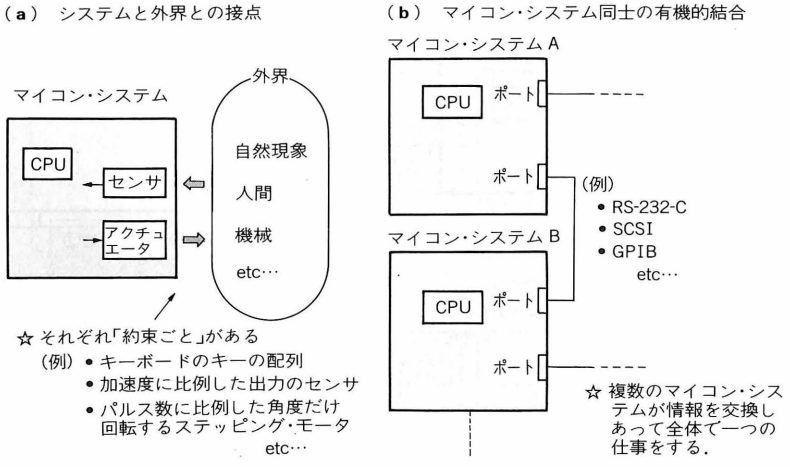
そして実際のA-D変換では，この「離散化」，「量子化」の両方が行われます．これが一般にいわれるA-D変換で，マイコン・システムの動作は時間的に不連続(システム・クロックにしたがう)ですから，当然，この分類となります．

ここで注意すべきことは，かなりの「情報の圧縮」が行われている点です．どんなA-Dコンバータにも必ずビット精度というものがあり，たとえば12ビット精度のA-Dであれば，一定範囲内の任意値(無限の種類)をとりうる入力データを，4096段階のどれかに限定することになります．この情報圧縮にともなう，実際のデータとの「量子化誤差」が問題となります．CDには16ビット形式のデータが記録されていますが，高級CDプレーヤでは「内蔵D-Aは18ビット精度」とか「20ビットのD-Aコンバータ搭載」などと競争しているのも，いわゆる「16ビットD-Aコンバータ」というチップの実質的な精度が，16ビットよりかなり悪い(じつは直線性も大問題)ためなのです．

また，どんなA-D変換器も時間ゼロでは変換できませんから，S&H回路によって変換時間ごとに入力データを確定させて，一定に保持します．これによって，入力データは時間的に切り刻まれる(サンプリング)ことになります．この「離散化」による情報圧縮は，サンプリング定理によって，いわゆる「折り返しノイズ」(サンプリング周波数の1/2の周波数以上の成分が原因となる本質的ノイズ)の要因となります．このために，ちょっと矛盾するようですが，ディジタル処理のためのA-D回路やD-A回路の周辺に，OPアンプを多用したローパス・フィルタ回路がずらりと並ぶ，といった風景も多く見られます．

ここで，最近の新方式として注目されているのが，「1ビット・オーバ・サンプリング方式」や「デルタ・シグマ方式」などの変換チップで，いずれもディジタル信号処理理論の研究と，電子回路技術やLSI設計技術の進展にともなって，各社から続々と発表されています．基本的には，たとえばオーディオ信号帯域を扱うのであれば，本質的に発生するノイズ成分を人間の可聴帯域よりもずっと高くしてしまおう，という原理によるもので，クロックとサンプリング周波数を非常

〔図5.3〕マイコン・システムの「インターフェース技術」



に高く設定したり，部分的に多値ディジタル量を活用して変換しています．未来のディジタル・オーディオ機器は，近くを飛んでいるコウモリを墜落させるようになるかもしれません．

インターフェース信号の検討

●マイコン・システムのインターフェース技術

マイコン・システムの基本的技術として，システムと外界との「インターフェース」を，ここであらためて検討してみます．これには，**図5.3**のように，

- システムと人間(や自然界)との接点としてのインターフェース
- 複数のシステムが互いに情報交換して有機的な結合を実現するもの

という二つの視点があります．

前者については，**図5.1**の例ですでに述べたように

- アナログとディジタルの視点(データ量と時間軸のそれぞれについて)
- 「電圧」，「電流」，「光」，「抵抗」といった物理量での分類
- ディジタルであればシリアル形式かパラレル形式か，多値形式か…

というような分類があります．「物理量」というと大げさですが，なにかのシステムでセンサを使ってデータを収集したい，というような具体的なテーマが与えられた瞬間，まず「なんのデータをどうやって検出しようか」と考える，まさにそのポイントです．

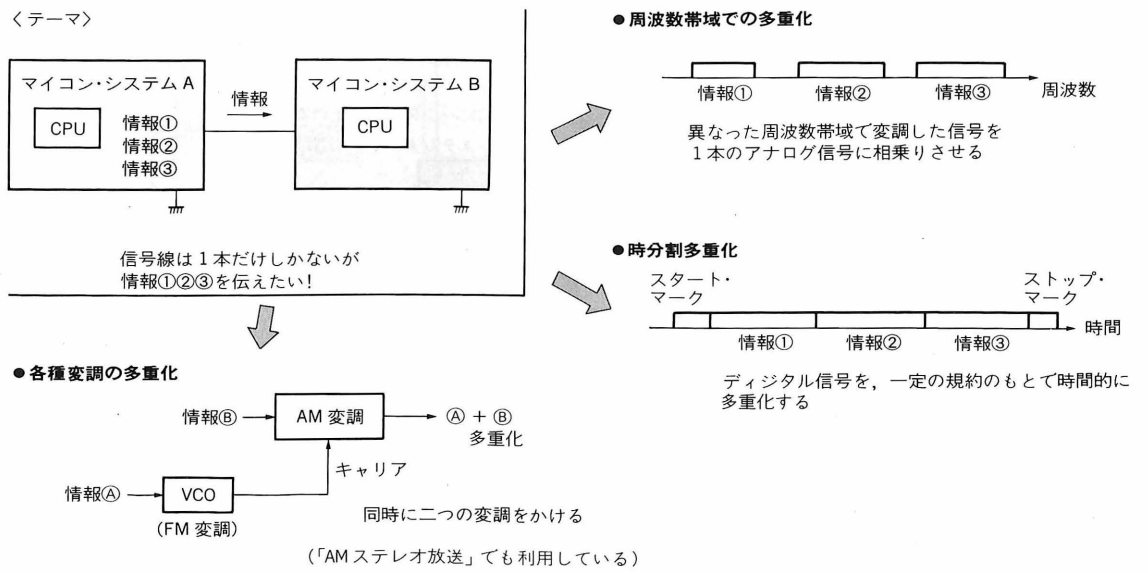
それから3番目の「信号の形式」も重要です．これは，たとえばシリアルは信号線が少なくて長い距離を結べる反面，情報の転送速度の面でパラレルよりも遅い，というように，基本的にトレードオフの問題ですから，「あらゆる面でパーフェクトのインターフェースというものは存在しない」という認識が大切でしょう．

◑インターフェース信号は格好の課題

現存するインターフェースを「そういうもの」として受け入れるだけでなく，もっと自由な発想で考えることも大切です．たとえば**図5.4**の例では，二つのマイコン・システムを結ぶ信号線が，たったの1本しかありません．ところが，ここに異なった3種類の情報をなんとか伝えたい，というのがテーマです．信号線が1本だと情報も1回線，というのでは面白くありません．そこで，たとえば

- 異なった周波数帯域の複数種類のアナログ信号を相乗りさせる
- ディジタル信号を時間的(時分割的)に多重化させる
- 周波数変調と振幅変調を同時に重ねられないか
- 2値でなく3値や4値のディジタル信号とならない

〔図5.4〕信号の多重化の例



か
など，いろいろな可能性を追求してみることになります．それぞれに深入りするとキリがない重要技術ですから，ここではあまり触れませんが，周波数帯域による多重化は光ファイバ通信の基本ですし，時分割多重化は最近のディジタル回線では常識です．

もちろん，送り手が情報を多重化した場合には，受け手はこれを正確に分離・復元しなければなりません．つまり，「符号化・復号化」，「信号多重化・信号分離」，あるいは「変調・復調」という処理は本質的にペアとなっていて，それぞれがハードとソフトの格好の課題となります．新しい「信号」に出会うたびに，このような背景となっている原理や技術を調べてみる，というのも，いい勉強になります．

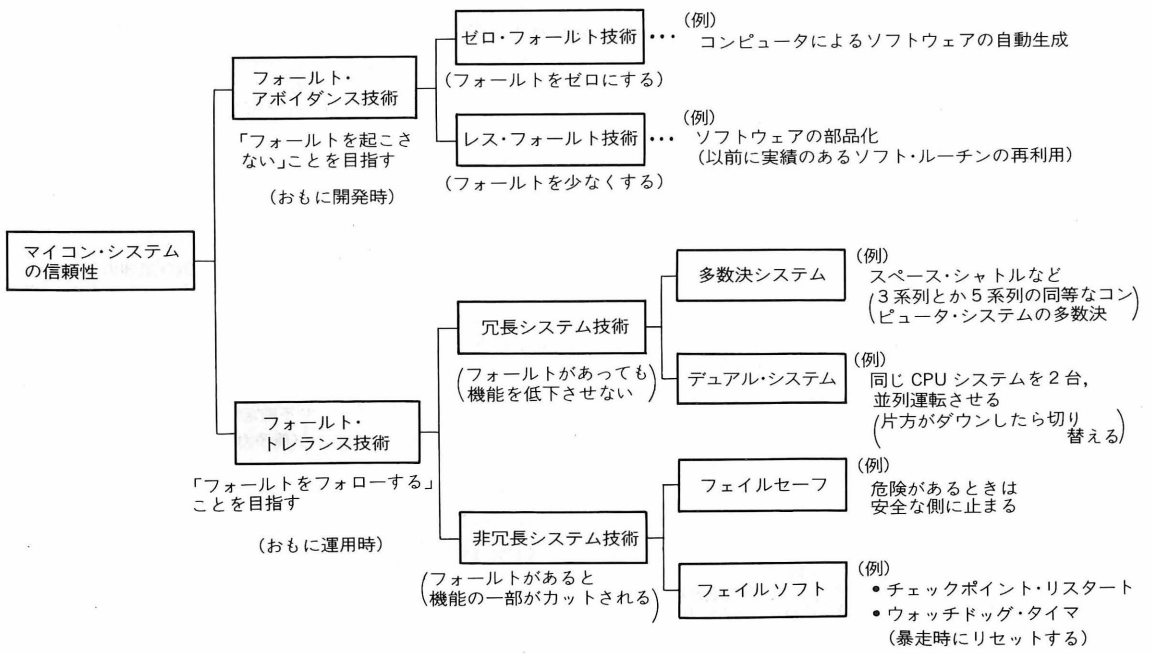
●信号の技術から信頼性の技術へ

インターフェース信号を例として，**図5.1**では信号の物理的な分類について少しだけ示しましたが，この「信号の考察」というのは，二つの意味でつぎの「信頼性の技術」につながる，重要なものです．

その第1は，「インパルス・ノイズ」，「電磁誘導」，「コモンモード・ノイズ」，「電波障害」などの，信頼性に関して登場するいろいろな現象が，すべて一種の電気信号（この場合，あまり歓迎されない）である点です．当然のことですが，ノイズ関係のトラブルに遭遇した場合に，相手をよく知らないのでは解析も対策もできません．いわば「上級アナログ技術」として，いろいろな信号（広くいえば「電気的現象」）について知識をもち，また応用のノウハウを蓄積することは，プロのマイコン技術者にとって長期的な目標なのです．

そして第2には，「信号」を理解するということが，エレクトロニクス技術の一つの基礎である「計測技術」と密接に結びついている点です．最近の計測器はディジタル表示のオートマチックが主流で，「オシロ・スコープの校正」とか「測定器・標準器のトリミング」といった，従来からの地味な計測技術のノウハウをあまり知らなくても，とりあえず通用してしまう便利な状況になっています．しかし実際の現場では，これらのセンスがあるかないかによって，システム性能やデバッグ効率が大きく左右される場合も多いのです．たとえば，「正しく調整していない電源装置と測定器を使って，規格取得のために製品の連続動作試験をしている」というような，笑えないエンジニアの「仕事」もあります．つまり，計測技術（「信号」を理解して正しく扱う）というのは，信頼性技術の基礎をささえる技術でもあるのです．

〔図5.5〕信頼性設計の考え方



マイコン・システムは，一種の「制御」(外界へのアクション)を行うものがほとんどですが，正しい制御というのは，正しい計測という前提があって，はじめて成立するものです．計測自動制御学会(SICE)という，かなり広範な対象を扱う学会がありますが，この学会誌の名前が『計測と制御』で，まさにこのポイントを示しています．基本のしっかりとした正しい計測があってこそ，そのデータを元にしたファジィ制御が意味をもつのであって，計測したデータそのものがファジィでは，システムは正しく動いてくれるはずがない，ということを理解しておきましょう．

信頼性の技術

●信頼性設計は「バグの研究」から

マイコン・システムを設計する場合，どれだけ信頼性のあるシステムをつくれるかがエンジニアの腕の見せどころとなります．もともと「信頼性」は，「正常に動作しているうちは気付かない」技術のために，なかなか表面に出てこない秘伝的ノウハウとなってしまいがちです．しかし，実際には信頼性の理論とか信頼性工学といった分野の歴史は(コンピュータの歴史以上に)長く，文献を調べてみると参考になる理論はいくつもあります．**図5.5**は筆者のまとめた「信頼性設計技術」の分類ですが，これはあくまで一例でしかありません．ここでは，信頼性の敵である「バグ」，「トラブル」に相当する概念を「フォールト」と呼んでいます．

全体を大きく二分するのは，

- フォールトを起こさないようにする技術(おもにシステムの開発段階で重要)
- フォールトは起きるものとして，それをフォローする技術(おもに稼働時に重要)

という視点です．これらはさらに，図のように細分化されていきますが，細かな項目のそれぞれが，いずれも重要な技術となっています．

また，**図5.6**にまとめたのは，このフォールトが起きてしまったときの「バグ」の分類です．なお，ここで「予選落ち」のバグというのは，たんに完成していない不良品のことですから，信頼性の検討の場合には論外となります．

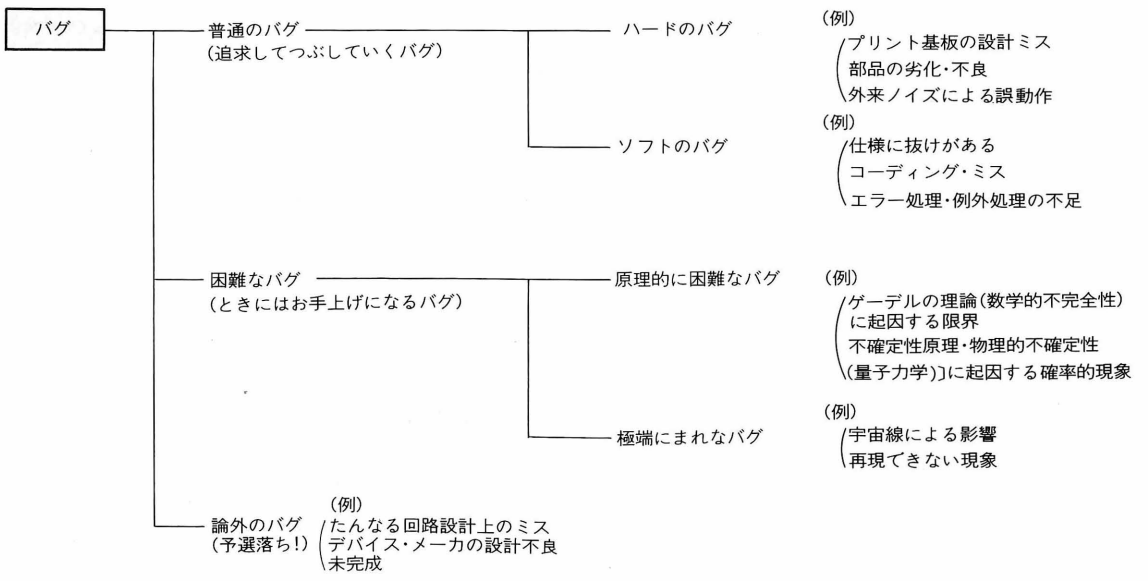
●トラブルからの復帰方法の考察

マイコン・システムにはハードとソフトの両面がありますから，信頼性設計にも両方からのアプローチが必要です．このうち，

- エラーをフォローして止まらない(暴走しない)
- エラーがあればリセットして復帰する

という部分については，つぎの「EMC技術」の項でも中心となりますので，ここでは関連するポイントを三

〔図5.6〕マイコン・システムの「バグ」のいろいろ



つだけ紹介しましょう．

大型のコンピュータ・システム，たとえば銀行のオンライン・システムでは，誤動作(データが化けたり，システムがダウンするトラブル)を避けるため，デュアル・システム(まったく同じシステムを二重にもつ)という対策を講じています．スペース・シャトルの制御系のマイコン・システムは，つねに5台が同じ処理を行って，偶発現象に対して結果が異なると多数決をとります．まだ汎用CPUチップではあまり見当りませんが，たとえば日本電気では，V60 以上のCPUで対応しているようです．

一般的な制御システムとしては，**WDT(ウォッチドッグ・タイマ)**を採用することが常識ですが，「はたしてエラーの際に，つねにリセットすればそれでいいのか」という問題は残ります．停電したら自分のキャッシュ・カードの中身がクリアされてしまった，というのは許されないでしょう．そこで，システムに何からの異常が発生したことを検知したとして，その結果をどう反映するか，という検討の課題があります．

また，どんなときにもシステムが安全な側に退避される(自動車であれば，アクセルを離す側，エンジンを切る側など)という，「フェイルセーフ」の発想も，信頼性設計の重要なポイントです．たとえば人工衛星のCPUソフトの場合，リセット・スイッチを押したりプログラムROMを変更したりできませんから，「絶対にエラーが起きない」ような設計だけが頼りとなります．もちろん設計したいとも思いませんが，原爆ミサイルの制御CPUシステムの設計エンジニアなどは，どんな神経をしていればいいのか，これは想像もつきません．

●「仕様的なバグ」のないシステム

マイコン・システムを小人数(場合によってはハードからソフトまで一人)で設計・開発していく場合に多いのは，じつは「仕様的なバグ」です．つまり，たとえば仕様書に100%したがった開発を外注先が実現したにもかかわらず起きたトラブル，といったもので，要は仕様を設計した段階で「すべてを見通せなかった」のが原因のバグのことです．

この種類のバグは人間である限りはしかたない，と開き直っていても解決しませんから，これに対しては，

- 構造化・階層化のテクニックでモジュール分割をキチンと行う
- モジュール仕様・モジュール間インターフェース仕様を明確に確認する
- 場合分けの検討は，**境界値分析(コラム参照)**などの手法を活用する
- ソフト上ですべての場合の可能性をシミュレーションしてみる
- 割り込みがつねに最悪のタイミングで連続している，という状況を仮定する

などの対策を地道に実行していくことが大切です．

また，最終的に仕様どおりに動作することの確認作業(検証)のポイントとして一つだけあげるとすれば，**「開発担当者がチェックしない」**という金言があります．開発している本人というのは，そのつもりがなくても，どうしてもクセや見落としが残ってしまいます．「市場デバッグが最大のデバッグ」という恐しい言葉もありますが，この意味では，エンジニアでなくて営業スタッフとか，学生や主婦とか，システムのことを知らない(予備知識のない)人に，「マニュアルだけを頼りに，なんでもいいから触ってもらう」というデバッグが有効です．

●自己診断プログラム

〈自己診断プログラム〉というのは，量産される製品の場合，すべてのハードウェアが外見上完成した段階で，チェッカにいちいちかける手間をなるべく省くものです．マイコン応用製品の場合，ハードウェアはいわばCPUの手足であるわけですから，CPUが自分で手足をあちこちチェックする，というのは容易なことです．

生産工場での効率を考えると，なるべくチェックの手間を少なくすることは，トータル・コストにも大きく貢献します．生産ラインの出口近くのテスト部分で，いちいちチェック用基板を挿入する，とかいうのではなく，よくある方法で，「ある特別の組み合わせでパネル・スイッチを押しながら電源ONする」という**チェック・ルーチン・エントリ方法**が便利です(いわば裏モード)．たとえば，温度アップと温度ダウンとか，スタートとストップといった，普通は同時に押さない組み合わせを押しつづけながらパワーオンする，というふうにします．またRS-232-Cコネクタに，TX DATAをそのままRX DATAに戻す特別のコネクタを差し込みながら上のことをする，というような手もあります．CPUはこのルーチンの冒頭で，RS-232-Cからある

[コラム]

境界値分析

●よくあるデバッグの例

図Gは，よくあるデバッグの例です．ある処理モジュールがあって，その機能仕様として，入力データを判定してそれぞれの処理に分岐する，と規定されているとします．このようなモジュールの動作を「検証」する場合に，どのような作戦をとったらいいか，というのが，ここでの課題です．

たとえば，「すべての入力データの組み合わせを試行してみる」というのも一つの方法ですが，この入力データが数ビット程度の整数であればともかく，たとえば「0から1の間の実数値」となったら，とたんに原理的に不可能となります．

この例の場合には，処理モジュールの内部動作として，はたしてどんなフローチャートを採用しているかが不明なのですから，検証も慎重でなければならないのです．**図H**(a)では，入力データと判定条件とをキチンと比較しているようですが，入力がたとえば［0，1，2，3，4］のいずれかをとるはず，という暗黙の期待があるようです．また，**図H**(b)のように，高速化のために「ジャンプ・ベクトル」を使っているかもしれません．この場合，このモジュールが「正常に」コールされていれば，確かに入力データは0から4までの整数値かもしれませんが，他のモジュールのバグとか，システムの誤動作によって入力データが別の値になったとしたら，このモジュールの処理の結果，ほぼ間違いなく暴走してしまいます．

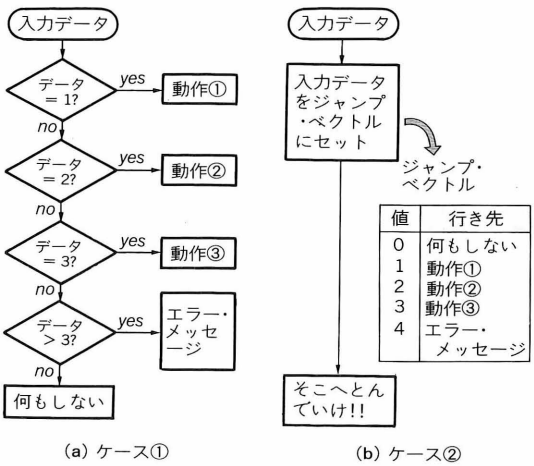
●極端な値と境界値の前後がポイント

このような検証の場合，処理モジュールの「入力データ」として期待されている値だけをテストしても十分ではありません．かといって，めくらめっぽうの値を入力するのも合理的ではありません．コツとしては，

- 入力データ範囲の最大値と最小値(場合によっては正負の無限大)
- 符号付きデータの場合，ゼロとプラス1・マイナス1
- 判定条件とされている値そのもの
- 判定条件とされている値のすぐ上の値とすぐ下の値
  (値が2なら，2.01とか1.99とかの値)
- 判定条件が等間隔の場合，その中間値とそのごくわずかに上下の値
  (中間値を判定に使っている場合があるため)
- ディジタル整数値であれば，オーバフロー・アンダフローする値
  (全ビット1とか，オール・ゼロ，サイン・ビット以外がすべて1か0の値など)

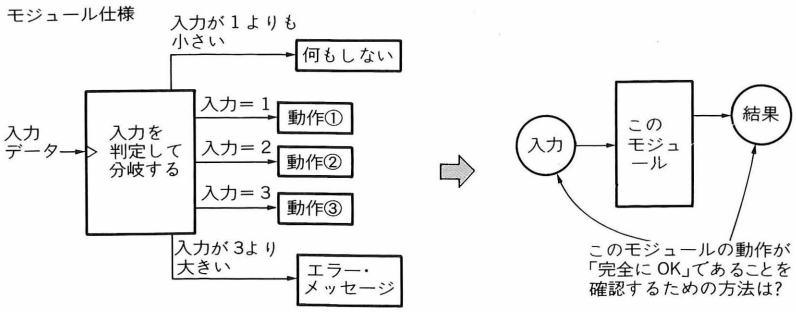
などの視点が重要です．「検証」の基本姿勢としては，モジュールの入力として，仕様に規定されていない，「想像もつかなかった」値が入力されうるものなのだ，という慎重さが大切だと思います．

〔図H〕図Gの処理モジュールの内部は…



(a) ケース①　　(b) ケース②

〔図G〕よくあるデバッグの例



暗号パターンを送信し，そのまま受信されるのを確認することで，このシリアル・ポートのハードウェア確認もしてしまえるわけです．

CPUはこのチェック・ルーチンでは，

① RAMのチェック
- RAMがささっているか
- データの読み書きができるか(ところどころのアドレスで，いくつかのデータのパターンで実行)

② ROMのチェック
- 特定アドレスのバージョン番号を確認

③ 周辺LSIのチェック
- できるものは割り込み対応なども

④ キー・スキャン，ディスプレイ・デバイスのチェック

などを実行し，OKなら所定のLEDを所定パターンで点滅するなどして知らせます．1秒もあれば相当のチェックができます．

EMC技術

● EMCとは

マイコン・システムの信頼性に関する話題として，最近ではEMC(Electro-Magnetic Compatibility)は常識となってきました．これには**図5.7**のような二つの側面があり，その両方を検討していく必要があります．

その第1は，システム自身から発生する電磁放射・ノイズによって外界に与える「妨害波」の影響です．輸出される電子機器では，FCC規格などの対策が重要になっていますが，この「外部に悪影響を与えない」というのは，今後ますますシビアになっていく技術です．日常的にも，パソコンでテレビの画面が乱れたり，ライタでパチンコ台がフィーバーしたとか，携帯電話で自動車が暴走したとか，という記事で話題になっています．

そして第2には，外部からの電磁波・ノイズによってシステムが影響されない「免疫性」(**イミュニティ**)があります．世の中のあらゆる電子機器にマイコンが内蔵されて，それぞれが数MHzのクロックで高速に動作しています．どこからどのような影響を受けるかわからない，という都市部の大気汚染(複合汚染)のような状況ですから，耐性を強化して自衛することもまた，システムの信頼性の面で重要な技術です．

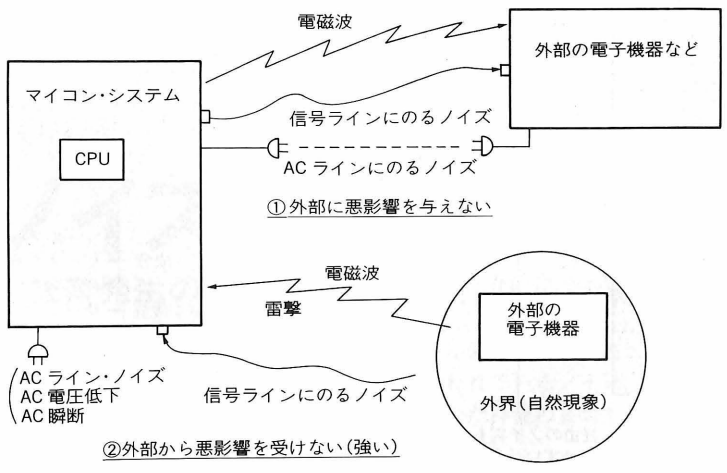
● EMCの要因は

このEMCを，**図5.8**のように，「影響を与えあう二つのシステムの関係」として考えると，

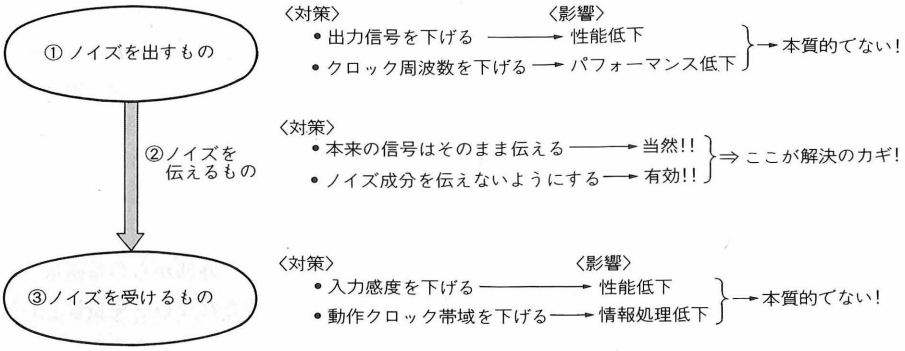
① ノイズを出すもの
② ノイズを伝えるもの
③ ノイズを受けるもの

という三つの要素がすべて揃っていないと，問題とならないことがわかります．ところが問題なのは，一般的に両者のシステム内部にも，それ自身がトレードオフをもっていて，EMCに対する解決方法が，逆にシス

〔図5.7〕マイコン・システムのEMCの二つの側面



〔図5.8〕EMCの三つの要素



テム側にとってはマイナス面をもつ対策となる場合が多いのです．たとえば

- 高い周波数の電磁放射を避けるためにCPUのクロックを大幅に下げる
- 高い周波数の電磁波に影響されないように受信機の感度を下げる

といった対策は，システムの機能そのものを低下させてしまいます．これでは何にもなりませんから，対策は残された「伝えるもの」に向かうのです．

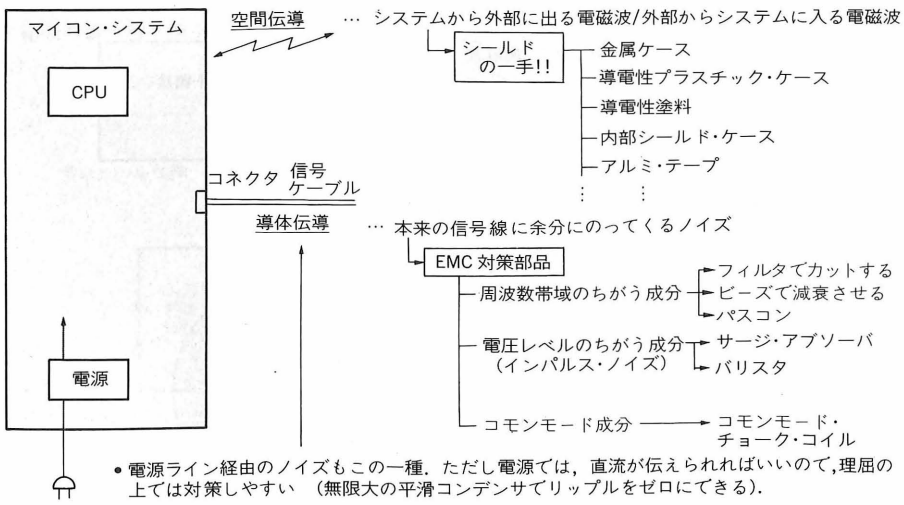
●ノイズの伝導経路と対策の整理

一言でノイズを伝えるもの，といっても複雑です．**図5.9**のようにノイズの伝送路としては，電磁波を伝える空間伝送路と，直接ノイズの伝播する導体伝送路があります．空間伝送路については，基本的には出すほうも入れないほうも，「シールド」という方針がメインです．シールドと簡単にいっても，目に見えない電磁波をふさぐというのは，やってみるとわかりますが，なかなか大変な技術なのです．

また，導体伝送路には，通信線・信号線のように，本来必要な信号も一緒に乗っているわけですから，ここからノイズだけを除外するのは大変です．あるいは電源ラインも導体伝送路ですが，ここでも電源を断つわけにはいきませんし，たいてい電源というのは共通にコンセントから取りますから，直流的にはつながっているだけになかなか面倒です．スイッチング電源のインバータ周波数も，最近ではAM放送用周波数に近いくらいに，どんどん高くなってきています．

導体伝導のノイズの場合には，必要な信号を通過させてノイズだけを遮断する，というアプローチが必要

〔図5.9〕ノイズの伝導経路と対策の例

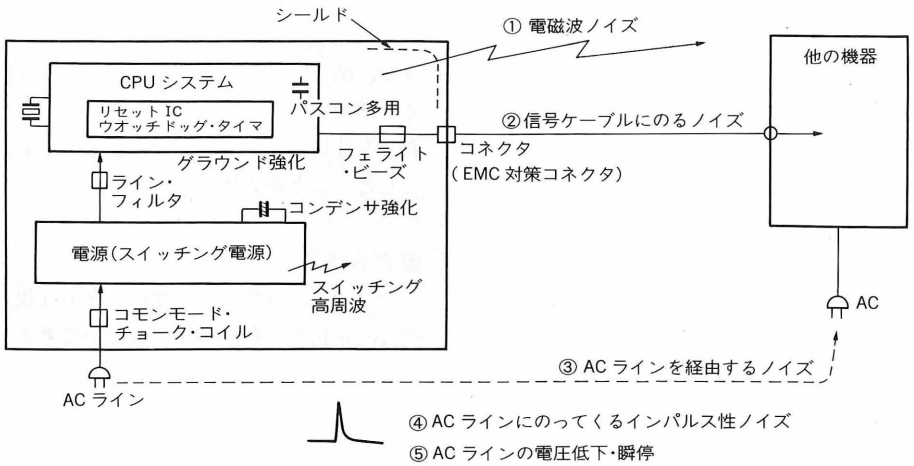


になります．つまり，いろいろな種類の「フィルタ」とか，一般に「EMC 対策部品」といわれる多くの部品が活躍します．ノイズと必要な信号とを区別する，なんらかの電気的性質に着目して，周波数帯域でのフィルタやビーズとか，電圧レベルでのサージ・アブソーバとか，コモンモード・チョークなどの部品を活用するわけです．ただし，これらの対策部品というのは，「万病に効く薬はない」のと同じで，この部品があればどんなノイズも大丈夫，などという特効薬はありません．ある部分は「現物合わせ」で，データを取りながら少しずつ目標性能に近づけていく…といった，地道な作業が必要になる場合が多いものです．

● EMC 対策はノウハウの展示会

この他の根本的な対策としては，基板のグラウンド・パターン，パスコン(バイパス・コンデンサ)，多層基板，ケース内の電源，グラウンド・ラインの引き回しなど，伝統技術のようなノウハウがいろいろあります(**図5.10**)．また，防いでも侵入してしまったノイズに対しては，誤動作という最悪のケースを避けるためのシステム対策が必要になります．つまり EMC 対策は，マイコン・システムにおいては「信頼性設計」とかなり結びついた技術なのです．この意味で，EMC 対策が十分になされたシステムは，誤動作対策の面でも強力(「**ノイズを出さないシステムはノイズに強い**」という格言)です．ハードウェアがただ動くだけでなく，よそに迷惑をかけない，あるいは他からの影響に強い，というシステムを作ることこそがプロの仕事なのだ，と肝に命じていきましょう．

〔図5.10〕マイコン・システムのノイズ対策のいろいろ



＜6＞

情報収集テクニックとドキュメント技術

あるエンジニアの情報収集の例

●技術情報とは

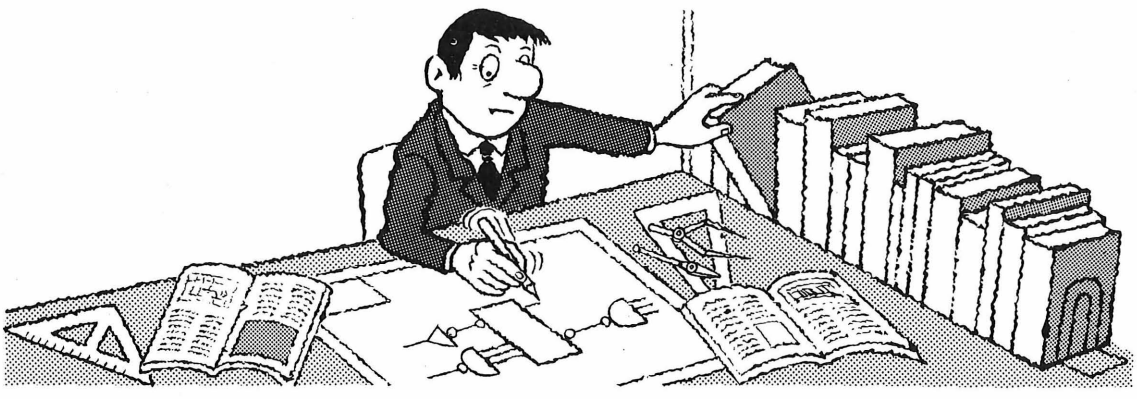
フレッシュマンが技術者として成長するためには，「技術情報の収集と活用」も一つの重要なポイントです．現代はコンピュータ時代ですから，あらゆる分野で「情報過多」になっています．日々刻々と進歩するマイコン技術の世界でも，もちろん関連する技術情報は数多くのメディアに溢れています．そしてフレッシュマンには「情報不安」，つまりどこかで大切な情報を逃してしまっているのではないか，というプレッシャがあるようです．

しかし，具体的なハードやソフトの技術を獲得するためには相応の経験を必要とするのに対して，情報の効果的な収集と活用は，心がければ今日からでも実行できることです．技術情報の特性を理解して，臆することなく，どんどん積極的に「情報の大海」に漕ぎだしましょう．

それでは，まず「技術情報」そのものを定義することから始めましょう．言葉どおりに考えれば，技術者が日常の業務で関係する情報は，すべて技術情報となります．そして情報の性格から大まかに分類してみると，マニュアル，データシート，規格書のように固定していて「参照される」ための情報，新製品，カタログ，報道のように「伝える」ことが目的の情報，そして論文・文献のように勉強・研究の対象となる情報など，特性によって，いくつかに分類できます．このような，個々の技術情報に対する位置付けは，後に述べる「活用テクニック」の重要な基準となりますから，自分なりに情報を分類できるところまで，まずは慣れてしまうことが大切です．

●ある技術者の情報源の例

それでは，今度は「情報源」という視点から，具体的な技術情報の収集ルートについて考えてみることにしましょう．**図6.1**は筆者の日常を見回してみて，技術情報収集に関係しそうな項目を表にしてみたものです．カッコ内には，具体的な名前の思い浮かんだものと活



用度のマークを付けてみましたが，この実名の列記にはかなりの抜けがあることをおことわりしておきます．では，順に簡単に説明していきましょう．

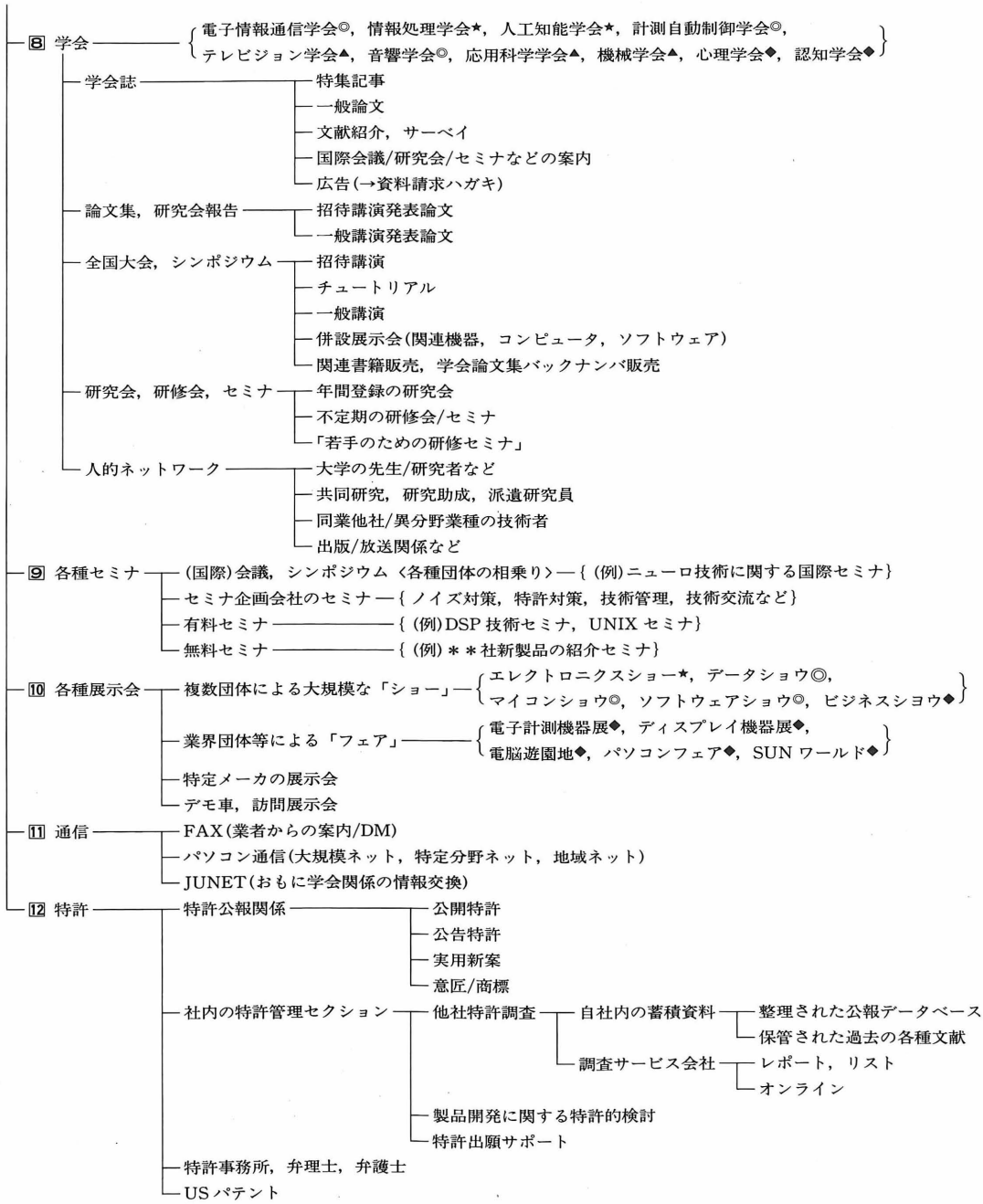
［1］の**社内情報**とは，会社内での社報・回覧・掲示などの，居ながらにして入手できる，もっとも身近なものです．ところが，先輩から継承される無形のノウハウなどに比べて，形になったものはあまり実際の役に立ちません．これは「身内」というフィルタを通る

〔図6.1〕ある技術者の情報源の一例 ①

ためなのか，情報を商品として提供するプロの仕事でないためなのか，とにかく「参考」程度として，社内の研修会とか勉強会だけでは不足なのだ，というぐらいに考えておきます．

［2］の**月刊誌**というのは，各種の情報源の中でもかなり重要な位置を占めています．特集・連載・評論などの記事本体とともに，資料請求ハガキを活用できる広告ページも，ベテラン技術者にとっては貴重な情報源です．たとえば，先月号からの変化のあったページをチェックすることで，広告そのもの以上の「業界の

〔図6.1〕ある技術者の情報源の一例 ②



傾向を推理する材料」になるのです．「差分をとって変化を知る」方法は，記事にも応用できます．たいていのエレクトロニクス総合雑誌は，ほぼ1年に一度の周期で，「カー・エレクトロニクス」，「ISDN 通信」，「ハイビジョン・衛星放送」，「ASIC 開発」，「CASE ツール」など特定の重要技術の特集を組みますが，これを前年の特集記事と比較してみると，中心となる技術そのものだけでなく，周辺の関連技術の進展の状況も整理して理解することができます．

［3］の**登録情報誌**では，定期購読システムのためにじっくりと(流行を追わずに)取材するという，月刊誌とはやや違った切り込みで編集されるために，それなりの収穫を得られるものです．ただし，かなり高いものですから，DM の宣伝に乗って個人で自宅購読することがベストであるか，というところは判断が分かれるでしょう．気になる点としては，たとえば「PC-9801 の将来は」という特集が同じ出版社の姉妹関係の各雑誌に一時期に集中して，ほぼ同じ記事を何度も目にするようなことがときどきあるなど，会社で購読しているものを回覧する，というパターンが無難なようです．

［4］の**無料情報誌**(**図6.2**)は，登録していれば定期的に送ってくれる，とても便利な情報ソースです．メーカから直接入手する最新情報(発表以前のもの)に比べるとやや古い，公表された新製品に限りますが，日頃あまりつき合いのないメーカの製品情報の入手に重宝します．ちょっとでも興味のあるものは「資料請求ハガキ」でどんどん請求し，必要なものはサンプルを取り寄せて実験し，使えないものはメーカの問い合わせ電話にも「また今度」と答える，というクールな対応をとることで，とことん活用できる情報チャネルです．実際の例ですが，たとえば数誌に登録していると，毎月数枚(計 20～30 件)のハガキを出すペースとなり，歩留まりとしては「100 件の資料請求がきっかけとなって，最終的に 2 件程度の採用につながれば十分な成果だ」，とでも割り切って活用できます．

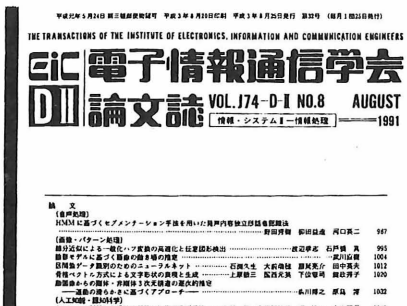
［5］の**新聞**については，筆者の場合には技術情報のソースとしては，まったく活用できていません．その理由は，「内容の掘り下げの浅さ」にあります．一般紙の科学欄・経済欄の記述があいまいであるのは仕方ないとして，業界紙であっても「メーカ側発表の引き写し」に徹しているのが，ちょっともの足りません．しかし新聞の情報は「報道」であって「解説」ではない，と割り切って何紙も読んで，相互の矛盾から「行間を読み」活用している人もいますから，ここからは個人差となるのでしょう．

〔図6.2〕さまざまな情報誌がある

［6］の**書籍**とは，雑誌以外のものを全部まとめています．もっとも正統的な技術情報のソースであり，理論とか勉強に立ちかえるとき，あるいは新しい技術について調べるときなど，つねにお世話になるものです．ハンドブックやデータブック，あるいはマニュアルや規格書といった書籍もまた「座右の書」としてつねに活用することになります．本書もまたこの分類に入りますが，はたして有益な情報を提供できているでしょうか．

［7］の**メーカ**とは，営業マンや代理店の担当者を経由して入手する，印刷される以前の最新の情報とか，公式なデータブックなどがメインとなります．また，一般に詳しい情報を公開しない ASIC 関連の技術情報については，それぞれのメーカから直接に入手することが基本です．新聞などに「新デバイス発表」と報道されるよりも半年とか1年前から，開発途中のテスト・サンプルを使って設計・開発を水面下で進めて，

〔図6.3〕学会誌の例


THE TRANSACTIONS OF THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

電子情報通信学会 論文誌 D-II

VOL. J74-D-II NO.8 AUGUST 1991

情報・システムII－情報処理

新デバイスが公式にデビューするのと同時に応用製品を発売する，などというのは，エンジニアとしてなかなか手応えのある仕事だと思います．

[8]の**学会**(**図6.3**)というのは，ちょっと異質な感じがするかもしれません．たしかに，具体的な製品情報とかの入手というよりは，先端の研究状況を知る，あるいは理論的な勉強をする場となります．しかし筆者の経験ですが，学会の全国大会のチュートリアルとかシンポジウムに参加することで，それまで何冊の本を読んでも理解できなかった概念がクリアになったり，目から鱗が落ちるように新アイデアが湧いてきたり…と，刺激の意味では，とても収穫の多い機会である，と思います．勇気を出して参加してみると，名前だけ知っていた学会の大御所先生と実際にお話しできてしまったり，関連領域の大学の先生と知り合いになったり(「共同研究」にでも発展できたらスゴイ！)，業界他社のエンジニアの友だちができたり，と人的ネットワークが広がります．

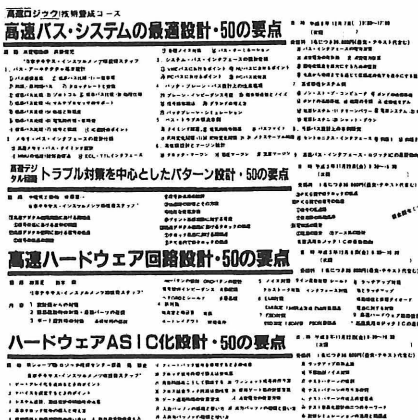
[9]の**各種セミナ**(**図6.4**)は，つぎの「展示会」に比べて，特定のテーマについての会議・講演会・セミナなどの勉強の機会として考えています．もっとも，最近では「セミナ屋」と呼ばれる専門の会社が，かなり高額なセミナ(似たようなものが多い)を定期的に開催しているためか，かつては無料が主流であったメーカ主催のセミナも，有料のものが多くなってしまい，めっきり参加することが減ってしまったのが実情です．有名な学者や評論家を(客寄せに)呼んで高額なセミナとするよりも，以前のようにメーカのエンジニアが講師をする，地味で確実な技術講習セミナを期待したいものです．

[10]の**各種展示会**とは，たとえばエレクトロニクスショーやデータショウ，マイコンショウなどが大きなものとして思いつきます．これはフレッシュマンの皆さんには，足にマメを作ってでも，1日中歩き回って，とにかくなるべく多くの情報を収集してみる格好のチャンスとして，おおいにおすすめします．カタログを両手に抱え，名刺をばらまき，どんどん質問することで，丸一日のショウで実験室にいる3ヵ月分くらいの技術情報を収集できます．

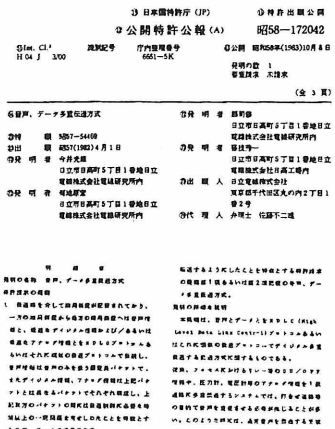
[11]の**通信**は，これからのトレンドとなるものでしょう．FAXとかパソコン通信，社内ネットワーク，そして国際ネットワークへと，世界の情報環境は着々と進んでいるところです．身近に機会があれば，どんどん活用してほしいと思います．ここではあまり深入りしませんが，情報そのもののソースとして活用するばかりでなく，通信・ネットワークに自分の力で参加(接続)すること自体が，技術的にかなりの勉強になる，という側面もあります．「コンピュータを制する者はエレクトロニクスを制する」という言葉は，これから21世紀に向けて，「コンピュータ」が「通信」や「ネットワーク」に置き換えられていくことになるでしょう．

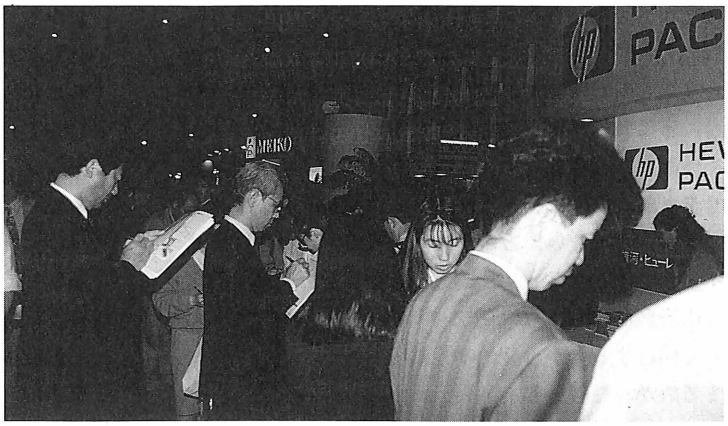
[12]の**特許**は，ソフトウェアの著作権も含めて，「知的所有権」として脚光を浴びている領域です．ここでは，特許公報(**図6.5**)などの情報を収集するだけでな

〔図6.4〕有料セミナの例



〔図6.5〕特許公報の例





各種ショウは技術情報の宝庫．丸一日歩きまわれば，収穫は大きい．毎年同じショウを見学し，差分をとることで，技術の大きな変化を読むこともできる．

く，新しいアイデアを特許出願するなど，「技術情報の発信」にもつながるものでなければなりません．具体的に詳しく触れることはできませんが，連日ニュースとなっている最近のホットな話題ですので，重要チェック項目として覚えておきましょう．

技術情報の活用テクニック

●情報の鮮度と寿命：プライオリティの視点

エンジニアが技術情報と一緒に生きていくためには，それぞれの情報をすべて後生大事に享受するのではなくて，情報の**鮮度と寿命**，そして**プライオリティ**という視点をもつことがたいせつです．

たとえば「トランジスタの発明秘話」というような情報は，読んでもおもしろいし，確かに一般的教養としては重要ですが，仕事をあとまわしにしてまで読むものではありません．筆者ならば出張の車中とか暇なときの読書とするでしょう．また「A社から新しいパソコンが出た」という情報自体は，たぶん半年後にはほとんど意味をもたない情報です．この類の技術情報というのは案外に多くて，「HS-CMOSのハイスピード版が出た」，「＊＊＊というツールのバージョンアップ」，「新オフィスの開設のおしらせ」など，頭の片隅に置くまでもない情報もたくさんあるのです．あとにも「情報の捨てかた」としてまとめますが，情報をどんどん取り入れるためには，受信した瞬間に，その情報をどの程度の重要度と判定するか，という感覚が必要になるのです．

この決断力というのは，同時にいくつもの仕事が押し寄せてきたときの行動決定の場でも発揮されますし，デバッグの際の試行錯誤の実験などでも必要なセンスなのです．何度かの失敗を恐れずに，情報を入手したときにプライオリティ判定を下して，全部をただ積み上げてしまわずに，

① 手を休めてでも読む
② ちらっと見たら捨てる
③ あとまわしにして保存する
④ 要点だけノートにメモする

というような分類を瞬間的に決定するトレーニングをしてみてはどうでしょうか．

●情報の収集・料理・廃棄

エンジニアにとって，技術情報とは生存環境のようなものです．そこで，ここではその「集めかた」，「食べかた」，「捨てかた」など，情報への対処法を考えてみることにしましょう．

まずは「**収集法**」です．情報過多の時代なので，過剰な情報は受け取らないように制限しよう，という考えかたもあるかもしれませんが，筆者はこれには反対です．制限したために，たまたま重要な情報を見逃すデメリットのほうが大きいと思うのです．ですから，資源保護の立場からはちょっと問題があるにしても，**「なるべく無節操に，できるだけ多くのルートから情報を受け入れる」**という姿勢を基本にしています．

なるべく心掛けて，多くの情報提供誌に登録し，雑誌の資料請求広告にハガキを出し，ショーやセミナなどの機会を活用し，職場で購読している雑誌は全部チェックするのです．この結果，数日の出張から帰ると，机の上にカタログのDMと雑誌類が山積みになって，

消化するのに半日を費やしたりしますが，これをがんばってかたづけるパワーをもちたいところです．情報を有効に料理するためには，内容は玉石混淆であっても，とにかく材料を集めることがたいせつなのです．

つぎに「**料理法**」です．ここでは前述の「**プライオリティ判定**」を駆使して，入手した情報をそのままいたずらにためない，という姿勢を堅持するようにします．それぞれの情報の重要度，保存/保管の必要な期間と場所(資料庫なのか机上なのか書類棚なのか)，チームで回覧するのか自分の手元に置くのか，読んだら捨てるのか，などなど…．

この作業をテキパキと進めるにはある程度の年季が必要ですが，こういう姿勢によって，技術情報を冷静に見る「**エンジニアの目**」が訓練されていくのです．情報を入力して，ここでフィルタにかけて重要な部分を抽出して，そして情報は廃棄されます．このフィルタの性能によって，同じ技術情報を与えても，そこからどれだけ有効なものを引き出せるかどうか当のエンジニアしだい，ということになるのです．

最後は「**廃棄方法**」です．技術資料をなるべく保存しておく，という姿勢は特許管理のセクションでは当然のことなのですが，それ以外の場合には，積極的な情報の捨てかた，というのも重要なことでしょう(とはいえ筆者も貧乏性のためか，ついつい資料がたまって反省する毎日です)．

情報にはそれぞれの鮮度があり，また寿命があります．定期的に自分の周囲の資料を見回して，新しい情報によって保存の意味を失ったものを廃棄していく，というのも技術者の仕事なのです．また，たとえば10ページのデータシートのうちのポイントとなる2ページだけを残して捨てるとか，記録としてブロック図のところだけをコピーしてファイルすれば十分などというように，情報を多量に入手したからには多量に捨てることでバランスをとることです．

●技術情報の二重ポインタ

技術情報のなかでもっとも身近なものは，データシート，規格書などの資料です．あるLSIの電気的特性とか，あるシステム・バスのコネクタ配置とか，電磁放射の規制値などのデータや，CPUの機械語命令とかエディタのコマンドなども同類でしょう．技術者が日々の作業でつねに接するもので，これらの情報は手元に置いて常時アクセスされます．

しかし，これらの情報は，自然に覚えてしまったものはしかたありませんが，「記憶すべき情報」ではありません．ここをしっかり理解しましょう．フレッシュマンはDOSの予約語とか抵抗のカラー・コードとかTTLの番号を「覚えなさい」といわれますが，これは日本語と同じようなもの，つまり最低限の常識であって，頭に詰め込むべき技術情報ではないのです．ここを誤解して，CPUのマニュアルとかコンピュータの教科書を丸暗記しようとする人がいますが，それはほとんど無駄なことです．

筆者は日本語くらいしか満足にできませんが，これまでにコンピュータ関係のいわゆる「言語体系」(CPUのアセンブラ，OSのコマンド，エディタ，コンパイラ，デバッガ，ASIC開発言語，CAD，…etc.)としては数十言語以上を使いこなしてきました．ところが，新しい環境を使いはじめると，以前のものは忘れるようにしていますから，常時スムーズなのは数言語程度なのです．それでは過去の言語は白紙になって忘れたかというと，マニュアルを開けばすぐにペラペラの現役に戻れるのです．

つまり，個々の基本的情報にどうアクセスするか，というポインタの情報だけをしっかり記憶していればいいのです．具体的な情報が必要になったときには，どのデータブックを参照すればいいか，どの書籍に解説が書いてあるか，どの棚のファイルに保存されているか，どのメーカの代理店に資料請求すればいいか，というような情報アクセスのためのポインタがあれば十分ということになります．

そしてさらにプロの技術者は，もうひとつ上の段階が必要とされることになります．上にあげたポインタというのは，いわば「技術情報図書館」の司書さんの仕事といえるでしょう．技術情報データベースを適切にアクセスするというのはもちろん重要なことですが，技術者は蓄積された過去の情報を引き出すことだけが仕事ではないはずです．新しいものを生み出していくためには，必要な情報を有効に活用しながら，既存の情報以上のものを創造していかなければなりません．

このためには，自分なりの「技術情報ポインタのデータベースとノウハウ」をいろいろな角度と条件からアクセスする，さらに高い次元のポインタが必要となるのではないでしょうか．いわば，技術情報への二重ポインタというわけです．

●行間を読む：情報の差分・補間・外挿

技術情報を取り込み，ラベル付けをして分類整理し，データベースとしてアクセスするだけなら，ほとんど自動化できるでしょう．さらに，デバイスＡと部品Ｂとソフトウェアを組み合わせたシステムを設計するという程度ならば，それぞれの情報データベースとリンクしたエキスパート・システムで，なんとかできそうな気がします．

では，エンジニアとしてそれ以上のものを創造していくためには，技術情報の扱いかたとしてどんなテクニックがあるでしょうか．ここでは筆者のアイデアとして，情報の**差分**，**補間**，**外挿**という提案をしてみましょう．皆さんがそれぞれ自分のフィールドなりにアレンジして，実際的な情報活用術の参考にしていただければと思います．

まず，**情報の差分**というテクニックです．ここでよく引き合いに出される例は，『トランジスタ技術』誌の分厚い広告ですが，定期的に供給されるような情報であれば，すべてに適用されます．はじめは圧倒されるほど多量の情報であっても，いつも接しているプロにとっては，前回の情報との「差」がわずかである技術情報は，案外にたくさんあります．そして，個々の情報そのものよりも，「どこが変化したか」，「どこが進歩したか」という「差分」というのは，より高度な技術動向の情報なのです．雑誌の全体的な解説記事として評論されるよりも何ヵ月も早く，現場のプロはこの変化を察知するのですが，それは「差分」情報によって臭ぎ分けているところが大きいのです．

また，**情報の補間**というのは，一見すると別々の技術情報の間の関係を補う視点です．たとえば，ある時期にＡ社とＢ社から，ジャンルの異なる新しいデバイスが発表になったとします．機械のデータベースはそれぞれの情報をため込むだけですが，エンジニアは何かを直感するときがあるものです．それは半導体製造技術の進歩状況であったり，水面下で新しく検討されている標準規格であったり，外国企業からのロイヤリティ譲渡だったりと，ケースによっていろいろです．また，補間される対象は，同じメーカからのいくつかの情報の場合もあります．まずは，こういう視点を頭の片隅に置いてみてください．

そして**情報の外挿**とは，文字どおりの技術的類推ということになります．一般に公開される情報というのは，まずまちがいなく，その発信者の技術レベルの一部でしかありません．あるメーカから新製品が発表されたときに，そのメーカの技術力としてはそこで精一杯ではなくて，もっと先を研究開発しているのですから，将来的なシステムをにらんだエンジニアとしては，入手できる情報をさらに外挿していくのは必須のことなのです．

たとえば，新しいプロセスのメモリICを発表したメーカでASICを作ろうとした場合，自分のスケジュールが半年とか1年先ならば，サブミクロンに進歩した状況を見越してシステム設計を行う，というような可能性を見ておく場合もあるのです．

エンジニアのドキュメンテーション技術論

●ドキュメンテーション技術とは

ここでは，21世紀までを視界に入れた「これからの

技術者」というものを考えていきます．じつはこれは，以前に筆者がまとめた一種の論文をベースにしているために，ちょっと表現が硬いものになっていますが，自分なりにポイントを系統的にまとめて暖めてきたものです．その重要ポイントとは，

●技術者を取り巻く環境が進歩・発展していく

●技術者自身の側にも大いなる変革が求められていく

という2点です．

一つめの「環境の進歩発展」とは，一言でいえば，技術者のあらゆる仕事のコンピュータ化，具体的には高度な作業支援システムを含む，いろいろな業務領域を統合したフルEWS化として実現されていくものです．また，二つめの技術者の変革とは，技術環境の推移とともに，「特定分野のエキスパート」から「総合的なエンジニアリング・プロデューサ」へと変化する，技術者自身の業務への要求といえると思います．

ここでは，技術者の直接の対象として密接に結びついた「ドキュメント」に注目して，上の二つの視点にもとづいて，これからの技術者が関係していく業務領域を6種類の「フェーズ」に分けて，それぞれの段階での技術環境や技術者の姿と，その具体的な結果であるドキュメントそのもの，あるいはドキュメント化する「ドキュメンテーション技術」について考察していきます．筆者なりの「近未来SF」として読み流していただければ結構ですが，実際にはこのような流れが着実に始まっている，という気がします．

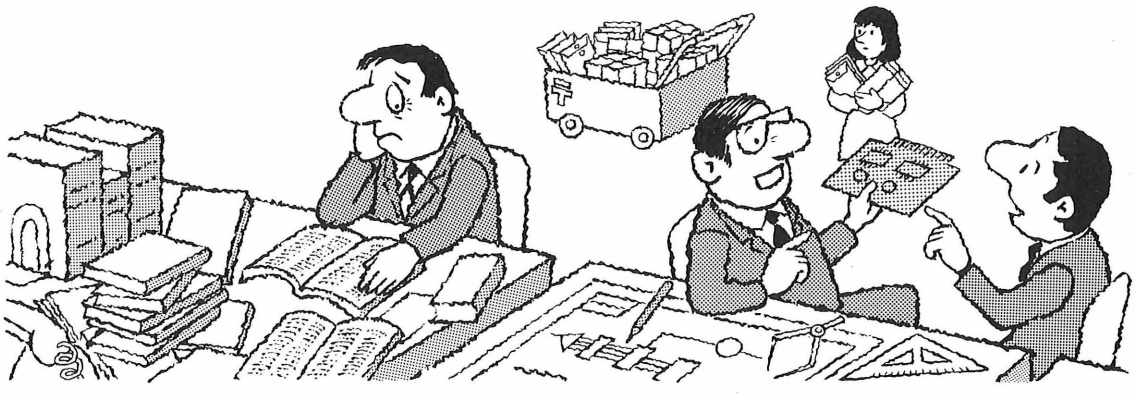
●企画フェーズ

技術者の仕事を単純化していえば，「新しい技術にもとづく製品・システム・サービスなどを，社会一般へ提供すること」です．ところで，新製品を世に発表する場合，いかにして対象となる人々に受け入れられやすい製品にするか，という「ニーズ」を考慮した企画戦略はとても重要です．このニーズ指向の部分に関する企画関係の業務を，ここでは最初の段階の「企画フェーズ」としています．

技術者がこの領域に関与する必要性は，これからの企画部門がワークステーション上に構築する「企画検討用データベース」という，一種のドキュメントを想定してみるとわかります．国内や海外から刻々と得られる市場調査データや，同業他社・他業種の動向の情報，あるいは消費傾向と関係する経済・文化・社会状況の分析データ，そして現在の事業展開情報や現行機種の資料などがEWS上に蓄積され，企画部門の業務は，このデータベースを対象とした詳細な分析・検討へと推移して，従来の新聞やカタログのコピー作業などは激減するはずです．

さらに，このデータベースをもとに「新製品企画のアイデア候補をいろいろと自動抽出するシステム」によって，同じコンセプトから複数のターゲットの製品ラインアップを同時に発表することが可能になりますが，これは「価値観の多様化時代」に適した有効な戦略です．このような状況になると，これまでのように専門分野の枠内にとどまった姿勢では，ニーズや企画情報が生かされないために，技術者自身の業務に対する発想の転換が必要となるわけです．

たとえば，システム設計の途中でLANを介して企画データベースにアクセスして，必要な市場ニーズ情報を自分のEWSに取り込んで，これを製品仕様に反映させながらシステム設計を行ったり，現行機種の仕



様に関するドキュメントをアクセスして，ソフト体系の上位互換性を確認しながら周辺機器との接続仕様を検討する，といった作業が普通になります．そしてさらに，企画データベース内の生データを自分の想定した分析手法によって検討し，新たなニーズ企画を創造・提案していく，というような柔軟な参加も可能になります．このような意味で，技術者が企画フェーズに関係することは，技術者自身の位置づけの向上とともに，より重要になっていくでしょう．

●研究フェーズ

「企画フェーズ」がニーズ指向の立場だったのに対して，この「研究フェーズ」とは，技術先導型のシーズ指向による企画分野として，技術者がより密接に関係しています．その対象は技術領域のあらゆる情報であり，これまでは独立した研究部門の技術者が専門に担当した分野なのですが，これからの技術者はこの研究フェーズも，業務の一端として関与していくことになります．

たとえば，現有する工業所有権や他社の特許状況に関する調査・分析データは，特許紛争の際の資料というよりも，製品企画・仕様検討のために個々の技術者が参照するドキュメントとして，EWS上にデータベース化されます．また，関連する技術分野の最新状況や，海外文献の自動翻訳による概説抄録，各種の学会論文や研究機関の研究報告なども，LANと接続されたファイル装置から必要な情報が検索できる環境となるでしょう．さらに，いろいろな種類の国際的な規格・規制・規則などの状況調査(改訂・変更が多く，つねにフォローしている必要がある)などは，試験・検証・保守といった業務に欠かせない技術情報であり，それぞれの作業環境となるEWSから随時アクセスされます．

さらに，新しいデバイス・装置・システムなどに関しては，提供側からのデータシートやマニュアルとともに，サンプルに対する評価試験の結果や，カタログ・データに載らない情報，実験から得られた経験則，使用上の注意や特殊なクセ，他の類似システムとの比較といった，担当技術者ならではの微妙な情報こそが重要です．従来は個々の技術者に頼ったこれらのノウハウが，共通の技術情報データベースに供給されることで，「個人では対応しきれない情報過多の時代」への解決策ともなるのです．

このように，研究フェーズの結果は，それ以降の各段階への基礎情報として参照され，さらには新しい技術によるシーズ企画の源泉となります．この意味で，「必要な情報の検索や分析がスムーズに行えるようなデータベース・システム」という，それぞれのメーカに固有の巨大なドキュメントの構築が鍵となります．

●設計フェーズ

従来の設計というと図面やコーディングを連想しますが，この「設計フェーズ」とは，EWSを支援ツールとした基本的なシステム構築作業で，その中心は，仕様検討・仕様書作成という，まさにドキュメント化の作業となります．ここでは，人から与えられた仕様にしたがって設計するだけ，という姿勢の技術者は消え去る運命にあります．

最近，企画→設計→開発→生産という流れの中で「コンセプトが散漫になる」といわれることがあります．これは各段階で情報が散逸する結果，製品の操作性の低下や機能の不備，ときには致命的なバグの誘因となるものです．しかしこの問題点は，技術者が検討・作成する開発仕様書がマニュアルの原稿でもある(いい替えれば「テクニカル・ライタを兼ねた技術者」による仕様検討)，という首尾一貫した業務形態が生まれることで克服されていきます．つまり，コンセプトを企画サイドとユーザの両方の視点から理解して，技術的状況をふまえて全体の機能仕様を客観的に検討し，これを「開発仕様書」というドキュメントの形に表現する，という技術者の業務が登場するのです．

もちろんこのためには，EWS上で対話的に確認しながら構築を支援する環境や，機能仕様の記述に使われる「記述形式」，「記述言語」の標準化が不可欠で，さらには「自然言語による記述から標準化言語の記述へ」の自動変換システムの実現も必要です．この開発仕様書の性格としては，

- 機能仕様・システム仕様の概略を技術者の立場から規定した骨格となるもの
- ハードウェアに関する部分をさらに具体化すると回路図になる
- ソフトウェアに関する部分を具体化するとフローチャートになる
- 動作機能に関する記述を一般ユーザ向けに整理するとマニュアルになる
- 個々の詳細な仕様条件を数値化すると検査仕様書になる

といったものです．そして，EWS上でこれらのドキュメント類を有機的に結びつけることによって，仕様変更・バグ対策・バージョンアップなど，ドキュメントを修正する際に，相互の矛盾や不統一を避けるための，有効な体系を実現することができます．

また，製品イメージや操作性を評価するための，プロトタイピング・ツールとしてシミュレータをEWS上で使うことで，具体的な開発段階に移る前に，あらかじめ全体像を把握できるメリットもあります．このように，設計フェーズはこれからの技術者の本領が発揮される，重要なステージとなります．

●開発フェーズ

設計段階のシステム構想を受けて，この「開発フェーズ」では具体的な開発作業を担当しますが，これは現在でも各種のツールによって少しずつ実現されてきているために，内容は比較的かんたんに想像できます．

最初に，機能的要求を実現するためのハードとソフトのトレードオフについて，対話的に検討するための支援ツールが起動されます．システム動作のどの部分をCPUソフト化して，どの部分をハード設計によって実現するか，という検討が，機能モデルの動作予測や開発期間・費用の予測データのもとで行われ，CPUの選択やASIC化を含めたハード構成の青写真が，最終的には「技術者の判定によって」決定されるのです．

つぎに，ハードの開発の面では，開発仕様書から全体の入出力信号に要求される仕様を抽出して，ここから「必要な部分のディジタル回路を搭載したカスタムLSI」を自動設計するCADシステムが起動されます．少量多品種製品の場合であれば，ASICでなくPLDやLCAを使った回路として，必要な論理設定プログラムが自動生成されます．すべての部品を含めた回路設計とプリント基板の設計も自動化され，設計ツールを操作する技術者の仕事は，EWS上に示された自動設計結果に対する「評価と指示」に比重が移ります．

ハード関係のドキュメントとしては，

- ファウンドリ(LSI製造会社)との接点となるLSIの設計データとテスト・データ
- 基板の回路パターン・データ
- 製造ラインの自動制御のための回路データ

などが出力されます．

また，ソフトの開発の面では，開発仕様書から機能仕様を自動抽出して，構造化設計に対応した基本フローチャートを自動生成するツールをまず起動し，対話的に修正しながら完成させます．さらに詳細な部分のソフトについては，高級言語(いずれは自然言語！)のレベルによって記述されたソース・プログラムが，そのまま仕様面でのドキュメントとなります．CPUプログラムの機械語レベルへの変換からROM化までは自動変換され，マスクROMはハード部分の部品データへとリンクされます．

このつぎにくる「試作機を使ってのデバッグ」という段階では，まず最初に，以上のハード関係ドキュメントとソフト関係ドキュメントを入力として，EWS上で全体の動作をシミュレーションするシステムを使用するようになります．これによって，機能的な論理チェックから詳細な動作タイミング・チェックまでが検証可能となり，実際に「手を出す」デバッグ作業のかなりの部分が省略されますから，これまでの意味でのデバッグというのは，相当に簡略化されるでしょう．

また，このような業務に並行して，技術者は「業務日報」に相当するドキュメントも，EWS上に常時，作成していきます．これはすべてのフェーズにおいて必須となり，

- のちの検証・仕様変更などの際に，各種ドキュメントを参照するための資料
- 特許問題や互換機などの著作権問題に際して，独自の技術状況を説明する証拠
- 客観的な評価がますます困難になる，技術者の勤務評価のための材料
- フレックス制・在宅勤務の時代に向けて，エンジニアの自己管理をサポート

というような意味を同時にもつ，重要なものです．

このように，製品開発の中核である開発フェーズの作業は，EWS環境(マシン単体だけでなく，ソフトやネットワークを含めた全体)の進展とともに，大幅にインテリジェント化されていくことになります．機械に使われるのでなく，機械を使いこなすエンジニア，という心構えが必要になりそうです．

●検証フェーズ

所定の設計に対するミスやバグを発見して修正する，という従来の意味でのデバッグは，環境の進歩とともに減っていきます．そして，この「検証フェーズ」では，仕様と異なる動作や論理的な矛盾を検証するデバッグを含む，信頼性試験・環境試験・機能試験などの

テスト作業の全体を行います．

まず，対象となる試作品や量産サンプルに対して，それぞれ該当する計測器・試験器などが接続されたり，所定のチェック・プログラムがロードされます．つぎに，具体的な試行・計測・記録・判定などが自動的に実行され，「評価データ」というドキュメントがEWSから視覚的に提示されます．

一方，たとえばノイズ試験であれば，データベース上のノイズ関係の国際規格(基準)と比較され，問題があれば警告されます．設計フェーズの開発仕様書から自動作成された「機能試験仕様書」や，開発フェーズの回路データから自動作成された「電気的試験仕様書」などの，電子化されたドキュメント類に対するチェックも同時に行われます．

これらの検証結果(というドキュメント)は，仕様変更や次期機種への応用，あるいは輸出規格取得のための資料ともなるので，定期的にまとめられてネットワーク内にファイリングされていきます．

●統括フェーズ

以上の段階によって物理的に完成した製品は，この「統括フェーズ」という技術者のドキュメント化作業によって，本当の意味での完成品になります．

まず最初の作業は，ユーザーズ・マニュアルとリファレンス・マニュアルの作成です．これは従来なら企画屋やテクニカル・ライタの分野でした．しかしこれ

[コラム]

プロジェクト管理も重要な技術

ここでは，「プロジェクト管理」という技術について考えてみます．これはフレッシュマンにとっては「技術」となかなか実感しにくく，もっとベテランになってから関係してくる「仕事」だ，と思われるかもしれません．たしかに，プロジェクト管理を実際に担当するのは，多くの実務経験のあとになるでしょう．しかし，これは若手だからといって目をつぶっていてほしくない，重要な技術だと思います．日常的に周囲を(興味をもって・批判的に)眺める一つの視点として，プロジェクト管理という考え方を加えてみてください．

●コストとスケジュール

筆者がお世話になった先輩技術者の口癖に，「コストとスケジュール」という，呪文のようにいつも登場するフレーズがありました．新しい開発テーマを検討する際に，まずはこの2点を考えよう，というわけですが，最初は個人的にけっこう抵抗があったのを覚えています．まだ若造だったこともあって，「技術者なんだから，まず新技術そのものの可能性を検討するべきでは？」などと考えていたのです．しかし数年すると，**結局はコスト，そしてポイントはスケジュール**なのだ，と身をもって実感するようになりました．

●スケジュール管理

技術者は，わざわざ「新技術」といわなくても，本能的に新しい技術を活用するものです．そして，技術者としての本能は，さらに目標となる機能仕様を越えた機能を，可能なかぎり盛り込んでしまいがちで，コストの制限を自主的に課さないと，ついつい過剰仕様の高性能化に走ってしまうのです．そして，この技術的こだわりを充実させるための条件として，スケジュール管理という「プロの技術」が効いてきます．期限の定められたシステム開発を担当したり，続々と新製品を開発する宿命をもったマイコン技術者は，時間的制約のあまりない「学者」，「研究者」とは違うのです．本人が納得するまで，などといっていると，際限なくスケジュールが伸びて，もちろん本人は徹底的に追求できたとしても，プロジェクトは失敗してしまいます．そこで，限られた期間に最大限，技術者の能力を注ぎこめるようにするための技術が，プロジェクト管理なのです．

すこし考えてみても，なにか具体的な開発テーマに対して，

- どのような個別技術が必要か
- 人員の割り当てと期間の見積り
- 開発環境やツールの想定
- いろいろな意味でのコスト

といった青写真を瞬間的に思い描く，というのは大変な仕事です．さらに，それをプロジェクトの具体的なスケジュール表，仕事分担，システム概略図などにブレークダウンする作業も，ちょっと新人には想像がつかないかもしれません．しかし，いずれは若手技術者もその境地に到達しなければならないのです．

からの環境では，基本的機能を記述した開発仕様書や，ソフト開発のドキュメントである自然言語のソース・プログラムなどによって，十分に機能説明の骨格は完備していますから，あとは技術者がユーザの視点を意識して，ドキュメントの切り貼りを行えば，容易に作成できます．ここでは，「説明文のフォーマットを用意して，キーワードの穴埋めを対話的に支援」するような，マニュアル作成ツールも使用されます．

一方，技術的な解説書であるリファレンス・マニュアルも，関連するドキュメントから必要な技術条項を検索して，マルチウィンドウ上で再構成するようなDTPツールの支援によって作成されます．開発プロジェクトの報告としてまとめられる「レポート」も，同じようなツールによって文書化されますが，新規な技術を採用した場合に必要な「特許出願」というドキュメント化とは，かなりの部分でオーバラップしますから，「特許出願原稿を兼ねた研究開発レポート」という新しい位置づけが生まれるでしょう．

さらに，不具合の改訂というよりも機能拡張のために，仕様変更やバージョンアップなどの作業が発生した場合，EWS上に保存されているフローチャートやソース・プログラムが，そのまま仕様変更の対象として検討できますから，従来の環境に比べて，2次災害(変更にともなって別の問題が発生)のリスクは，かなり改善されます．また，基本的システムを踏襲した後継機種(マイナ・チェンジ)を開発したり，新しい次期機種(モデル・チェンジ)の開発を検討する場合にも，これらの統括フェーズで整理された各種のドキュメントをベースとすることは，信頼性と効率の面で大きく貢献します．

このように，これからの技術者とは，それぞれの業務領域において，広範な視野をもって総合的な仕事を行う，いわば「**エンジニアリング・プロデューサ**」となるための自己形成が求められていくことになると思います．

[飛び石コラム] おすすめBOOKS ④

●マイクロコンピュータのプログラミング
石田晴久　編　　共立出版

進歩の激しいマイコン分野で，おそらく古書店にしかない10年以上昔のこの本を紹介するというのは，問題があるかもしれません．しかし，あらためて読み返してみると，「良書は歴史を超越する」という真理をこの本は証明しているように思います．伝説のTK-80の時代の技術とはいえ，本質は変わっていないのに驚かされます．

●ニューラルネットワークシミュレータ SONNET
若松真・安西祐一郎　著　岩波ソフトウェアライブラリ

これは「本」といっても，ニューラルネットを実験するためのソフトウェア・ライブラリに付属したマニュアル，ともいえます．多くの本でニューロの「説明」を読むよりも，本書によってとりあえず「走らせてみる」ほうが，はるかにニューロについての理解が進みます．このSONNETについては，パソコン通信 NIFTY-Serveの「人工知能フォーラム」でも，専門家が関連の話題をフォローしているようです．

●コンサルタントの秘密 ── 技術アドバイスの人間学
G.M. ワインバーグ　著　　木村泉　訳
共立出版

この本との出会いは，筆者にとってここ数年の中で最大のヒットとなりました．技術コンサルタントであり，優れた教育者である著者の鋭い指摘と人間観察の力にはただただ脱帽です．同じ著者の「スーパーエンジニアへの道」という本もあります．こちらは期待が過剰だったせいか，やや個人的には不満が残りましたが，むしろ後者からより多くの収穫を得る人もかなりいると思います．

●システムズエンジニアハンドブック
中原啓一・加藤榮護　　共編　　オーム社

この本は多くの著者による「教科書」として，システム・エンジニアリングの技術について集大成したものです．しかし勉強のための読み物としても，壮大な技術の背景の思想を垣間見ることができます．ただ，富士通関係者だけで執筆しているための内容の「偏り」がちょっと気になり，その分を割り引いて読む必要がありそうです．

＜7＞ 上級システム：ASIC技術

ゲートアレイから スタンダードセルへ

ここでは，大規模な回路をユーザ(マイコン・システム技術者)がLSIチップとして設計・開発してしまう「ASIC技術」の全般について，ステップごとに詳しく検討していきます．ASICは，従来は大量生産ベースの大メーカにしか関係のない世界であって，一品料理のシステムハウスには縁のないものでしたが，最近ではこの先入観を打破するユニークなASICも登場してきています．そこで，これからのマイコン技術者の強力な「持ち駒」として，あるいは半導体テクノロジーの上級技術として，誰もが視界に入れていく必要があるでしょう．

●ゲートアレイの流行

かつてのゲートアレイといえば，膨大な開発費用と，ロット数量としてかなりの個数(の保証)を必要としてきました．PLDやLCAを活用する技術があっても，この点でシステムハウスには，ちょっと縁のない(手の出せない)存在だったかもしれません．しかし，いろいろなASIC技術の進展で，状況はしだいに変化してきています．たとえば，ここ1年ほどの雑誌広告(開発途上の記事ではない！)をざっと見ても，

- ロット10個でも受注する試作専用ゲートアレイ
- ゲートアレイと完全互換のデータ体系をもつ大規模PLD
- 開発期間が2週間のゲートアレイ・メーカ
- 設計からレイアウトまでを受託するだけの専業LSIデザイン・ハウス
- マスク・パターンからLSIを製造するだけの専業ファウンドリ
- ファウンドリやプロセスをいろいろに選べる設計支援CAD

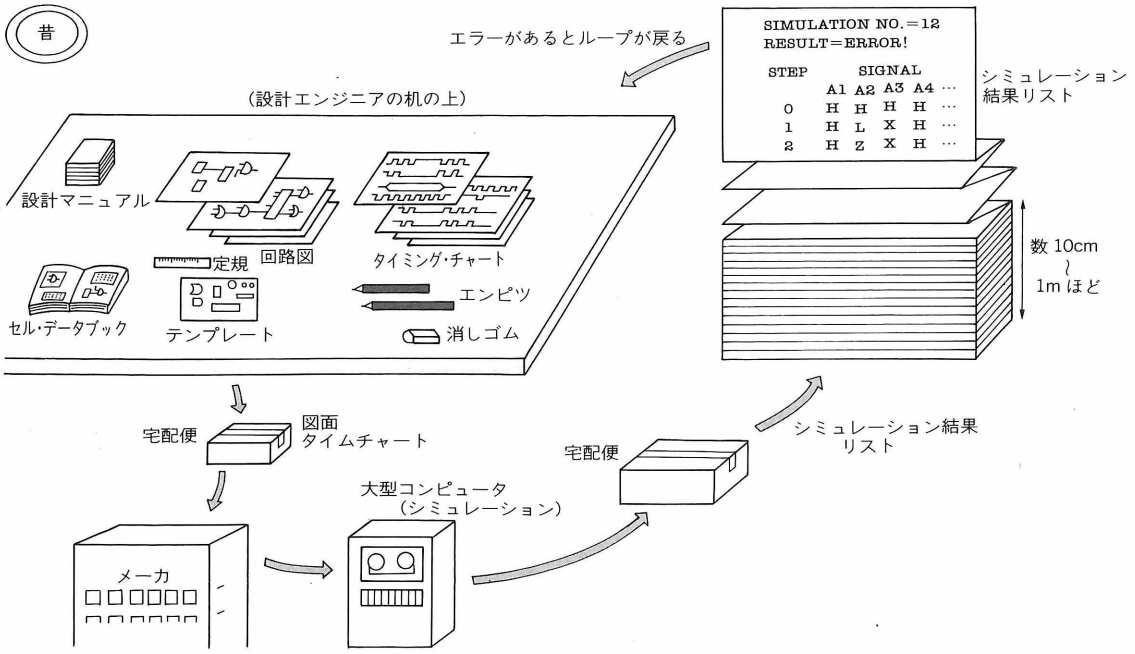
などといった，新しいものが登場してきました．これらの新技術はすべて，「小回りの効いたゲートアレイ」という状況にプラスとなっています．

●ゲートアレイの長所

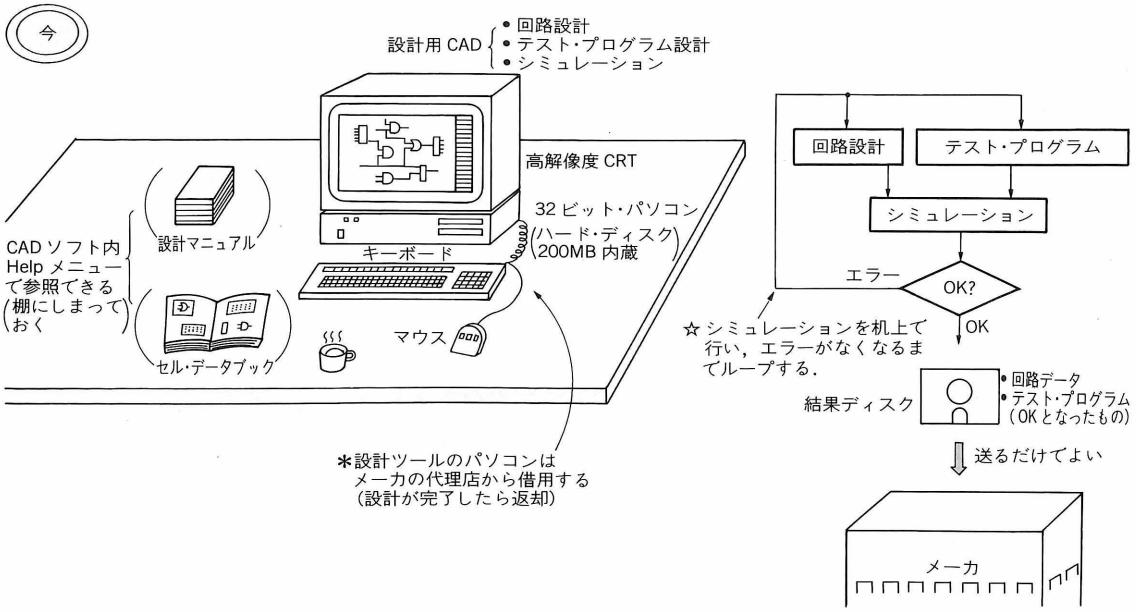
設計者のカスタム・ロジックを高密度に実現してしまうところがゲートアレイの最大のメリットです．もちろん知的所有権にからむ回路技術のノウハウもチップ内に隠蔽できますし，回路に要求される機能・性能・精度は，テスト・パターンとシミュレーションを納得するまで練りあげることで，理論的には「完全に」実現できます．そして最大のメリットである「コスト」の面では，最適な数千ゲート規模の場合，ゲート効率は90％程度と高く，適当なパッケージを選択すると，「ゲートあたり十数銭」(ピークでは「数銭」！)というような，驚異的なコストを実現できます．TTLでは「ゲートあたり数円」以上ですし，回路規模が大きくなれば電源や実装上のコストがさらに上昇しますから，ちょっと計算してみると，この差は非常に大きいのがわかるでしょう．CPU回路の周辺機能を1チップ化する場合には，QFP・PGAなどの採用によって，パッケージのピン数を多く取って，I/Oポートなどに活用できるメリットも大きいものです．

ゲートアレイの設計は，マイコン技術者本人が行うのが理想です．ASICの設計全般にいえることですが，設計の途中に他人が介在すればするほど，回路設計の最適化と設計思想の一貫性が失われていってしまいます．かつての「大型機のホストと電話回線でつなぐ」，「ミニコンを一晩ぶん回す」，「宅急便でシミュレーション・リストがドカッと届く」といった時代(**図7.1**)と違って，最近のゲートアレイ設計は，パソコン/EWS上

〔図7.1〕ゲートアレイ設計の今昔(昔)



〔図7.2〕ゲートアレイ設計の今昔(今)



〔図7.3〕スタンダードセルのセル・リストの例(セイコーエプソン)

●COMPLEX GATE

CELL NAME	FUNCTION
AN23	2-AND 2-WIDE 3-INPUT NOR GATE
AN14	3-AND 2-WIDE 4-INPUT NOR GATE
AN24	2-AND 2-WIDE 4-INPUT NOR GATE
AN34	2-AND 3-WIDE 4-INPUT NOR GATE
AN26	3-AND 2-WIDE 6-INPUT NOR GATE
AN36	2-AND 3-WIDE 6-INPUT NOR GATE
AN28	4-AND 2-WIDE 8-INPUT NOR GATE
AN48	2-AND 4-WIDE 8-INPUT NOR GATE
AN44	2-OR 2-AND 2-WIDE 4-INPUT NOR GATE
AN24A	2-LINE TO 1-LINE SELECTOR/MULTIPLEXER
AO23	2-AND 2-WIDE 3-INPUT OR GATE
AO24	2-AND 2-WIDE 4-INPUT OR GATE
AO26	3-AND 2-WIDE 6-INPUT OR GATE
AO36	2-AND 3-WIDE 6-INPUT OR GATE
AO28	4-AND 2-WIDE 8-INPUT OR GATE
AO48	2-AND 4-WIDE 8-INPUT OR GATE
AO24A	2-LINE TO 1-LINE SELECTOR/MULTIPLEXER
ON23	2-OR 2-WIDE 3-INPUT NAND GATE
ON14	3-OR 2-WIDE 4-INPUT NAND GATE
ON24	2-OR 2-WIDE 4-INPUT NAND GATE
ON34	2-OR 3-WIDE 4-INPUT NAND GATE
ON26	3-OR 2-WIDE 6-INPUT NAND GATE
ON36	2-OR 3-WIDE 6-INPUT NAND GATE
ON28	4-OR 2-WIDE 8-INPUT NAND GATE
ON48	2-OR 4-WIDE 8-INPUT NAND GATE
ON44	2-AND 2-OR 2-WIDE 4-INPUT NAND GATE
OA23	2-OR 2-WIDE 3-INPUT AND GATE
OA24	2-OR 2-WIDE 4-INPUT AND GATE
OA26	3-OR 2-WIDE 6-INPUT AND GATE
OA36	2-OR 3-WIDE 6-INPUT AND GATE
OA28	4-OR 2-WIDE 8-INPUT AND GATE
OA48	2-OR 4-WIDE 8-INPUT AND GATE

●LEVEL CLIP CELL

CELL NAME	FUNCTION
PLUP	LEVEL CLIP FOR POWER LEVEL
PLDW	LEVEL CLIP FOR GROUND LEVEL

●LATCH

CELL TYPE	FUNCTION	CELL NAME
LF*	LATCH	LFP/LFS/LFC/LFR
NLF*	NEGATIVE CLOCK LATCH	NLFP/NLFS/NLFC/NLFR

末尾P WITH PRESET　末尾C WITH CLEAR
末尾S WITH SET　末尾R WITH RESET

●FF-S

CELL TYPE	CELL NAME	WITH SET/RESET/SET & RESET
DF*	D-FLIP FLOP	DF/DFS/DFR/DFSR
TF*	T-FLIP FLOP	—/TFS/TFR/TFSR
JK*	JK-FLIP FLOP	JK/JKS/JKR/JKSR
SF*	SCAN FLIP FLOP	SF/SFS/SFR/SFSR
SJK*	SCAN JK-FLIP FLOP	SJK/SJKS/SJKR/SJKSR
NDF*	NEGATIVE CLOCK D-FF	NDF/NDFS/NDFR/NDFSR
NTF*	NEGATIVE CLOCK T-FF	—/NTFS/NTFR/NTFSR
NJK*	NEGATIVE CLOCK JK-FF	NJK/NJKS/NJKR/NJKSR
NSF*	NEGATIVE CLOCK SCAN D-FF	NSF/NSFS/NSFR/NSFSR

末尾S　WITH SET
末尾R　WITH RESET
末尾SR　WITH SET & RESET

■MSI CELL

MSI Cell Libraryには，頭文字Tから始まるTシリーズと頭文字Aから始まるAシリーズの2種類があります。頭文字に続く3文字(数字)はTTL74シリーズと機能等価な型名です。最後の1～2文字はサフィックスでオプション機能を示します。

T頭文字 T SERIES
A頭文字 ADVANCED SERIES
末尾C　WITHOUT CARRY
末尾E　WITHOUT ENABLE
末尾G　WITHOUT GATE
末尾H　HALF (2 BIT CNT)
末尾L　WITHOUT LOAD
末尾R　WITHOUT RESET
末尾T　TRANSPARENT LATCH
末尾V　REVERSE COUNTER
末尾W　DUAL (OCTAL REG)

●COUNTER CELL

CELL NAME	FUNCTION	SIMILITUDES
A161	SYNCHRONOUS 4-BIT BINARY COUNTER	T161E, A161H A161L, A161LE A161R, T161RE A161HE, A161V
A163	FULLY SYNCHRONOUS 4-BIT BINARY COUNTER	A163E, A163V
A191	SYNCHRONOUS 4-BIT UP/DOWN COUNTER	A191E A191CE, A191H A191LE, A191U A191V
A193	SYNCHRONOUS 4-BIT DUAL CLOCK BINARY UP/DOWN COUNTER	A193L, A193H A193HL
T669	SYNCHRONOUS 4-BIT BINARY UP/DOWN COUNTER	T669L
T177	ASYNCHRONOUS 4-BIT BINARY COUNTER	T177H, T177HV T177R, T177V T177VR
A390	ASYNCHRONOUS BCD UP COUNTER	—
A393	ASYNCHRONOUS 4-BIT BINARY UP COUNTER	T393, T393V

●ADDERS COMPARATORS

CELL NAME	FUNCTION	SIMILITUDES
T182	LOOK-AHEAD CARRY GENERATOR	—
T183	1-BIT FULL ADDER	—
A183	1-BIT FULL ADDER	—
T283	4-BIT FULL ADDER	T283H
A283	4-BIT FULL ADDER	—
T085	4-BIT MAGNITUDE COMPARATOR	T085G
T688	8-BIT MAGNITUDE COMPARATOR	T688H, T688HG
ADH	1-BIT HALF ADDER	—
ADL	1-BIT HALF ADDER	—

の一種の統合CADソフトなのです．回路入力のCADはプリント基板設計と似たものですし，テスト・パターンの編集には使い慣れたテキスト・エディタや，専用の波形CADを使います．これらのデータを用意すると，シミュレーション・ソフトはバッチで勝手に走って，エラーがあれば知らせてくれますから，論理シミュレーションまでなら，実験室の机上で済んでしまう規模になっているのです(**図7.2**)．

●個数問題は「社内汎用LSI」でクリア

ゲートアレイを使うための最大のハードルは，数量の条件です．そこで，当面する開発テーマの「専用LSI」でありながら，その後も「社内汎用LSI」として多量に使い回せるように，限られたゲート数の範囲内で，いろいろな便利機能を「隠し機能」として仕込んでおきます．このテクニックは，現実の経済的戦略というだけでなく，ある意味で，そのゲートアレイを「作品」としてしまう技術といえます．たとえば，あるCPU回路に対応して設計するのであれば，今後そのCPUを使って別のシステムを設計する人が，「このゲートアレイを活用すると便利だ」と思えるように，考えられる周辺サポート機能を盛り込んでおくわけです．

また，最初はカスタム開発したASICを，LSIメーカがその機能のメリットを評価して，「当社の汎用LSI(ASSP)として外販したい」と希望してくる場合もあります．こうなると，量産効果によってチップ単価は大幅に低下しますし，場合によってはロイヤリティ契約，というオイシイ話に展開する可能性すらあります．まさに技術者のアイデアと実力次第で，おおいに活躍できる世界だと思いませんか．

●ゲートアレイからスタンダードセルへ

ASICといえば，ゲートアレイとスタンダードセルが両横綱です．この両者の違いは，一部のメーカ内部では相当あるようですが，ユーザであるマイコン技術者の側では，あまりその差を考えなくてもいい時代に向かっています．極端にいえば，ASICを作る巨大LSIメーカが，「自分のオリジナル回路をチップにしてくれる，便利な外注」ぐらいに考えられる時代になろうとしているのです．

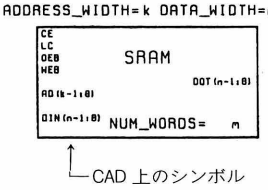
ゲートアレイとスタンダードセルの使い分けのトレードオフについては，「**ゲートアレイよりも集積度が高くて，大規模なロジックの場合にスタンダードセルを採用する**」というのが一つの目安です．具体例をあげると，複雑なTTL相当セル(**図7.3**)を多用する場合や，ROMやRAMなどのメモリ・ブロック，乗算器や累算器などの演算器ブロック，プログラマブル・タイマやシリアル入出力ポートなどの専用機能ブロック(メーカ製の高密度マスク)などを使う場合には，スタンダードセルによって設計していくことになります．

●コンパイルド・セル

最近では，メーカが提供する「出来あい」のセルばかりではなくて，コンパイルド・セルという高級なセル・タイプが登場しています．これは，従来はチップ上のランダムロジックとして，たとえばラッチを並べて小規模なメモリを構成していたものを，「xビット，yワード構成のRAM」とだけ指定すると，設計支援システム内で最適のレイアウト・パターンをもつセルを自動合成してくれる，というものです(**図7.4**)．もちろん，シミュレーション段階で他のセルと同様に扱うための「シミュレーション・モデル」も同時に作られますから，最適な構成のセルをオリジナル登録できる環境として，かなりの威力をもちます．このようなセル・コンパイラは，ALU(**図7.5**)やデュアルポートRAM(**図7.6**)など，複雑なセルになるほど，具体的にロジックで構成するよりも効果が出てきます．

なお，スタンダードセルの場合，ゲートアレイよりもシミュレーションやレイアウトの規模が大きくなるために，開発費用と開発期間が1桁はアップします．

〔図7.4〕コンパイルド・セルの例：SRAM(NCRデータブックより)



CAD上のシンボル

INPUT PARAMETER	ALLOWED RANGE	COMMENTS
Num_Words	16 - 1024 multiple of 16	The memory array is organized as "n" words of "n" bits each
Word_Width	1 - 16	

←ワード数の指定

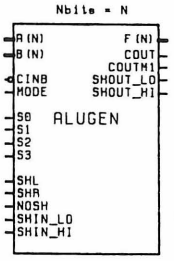
←1ワードのビット幅の指定

これは，設計の段階ではメーカ製のマスクを活用できるといっても，シミュレーションやチップのテストには，最終的にブロック内のゲート単位まで検証する必要があるため，配線部分を除くチップのハードウェア部分が標準化(共通化)されているゲートアレイに比べて，試作・製造のステップのかなりの部分が「オリジナル」となるためです．

●シー・オブ・ゲートとゲートアレイ分割

ゲートアレイの巨大なものとして，シー・オブ・ゲート(Sea Of Gates)という10万ゲートとか20万ゲート規模のものもあります(**図7.7**)．これと同等の機能は数万ゲート規模のスタンダードセルで実現できるのが普通ですから，どちらを選択するかは，かなり高度なトレードオフとなります．この規模では，1チップに収

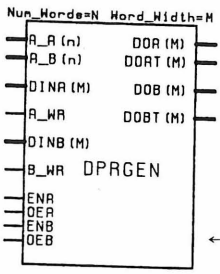
〔図7.5〕コンパイルド・セルの例：ALU(NCRデータブックより)



←CAD上のシンボル

INPUT PARAMETER	ALLOWED RANGE	EXPLANATION	
Nbits	2 - 32 even only	Number of bits in the input and output data buses.	←データ・バス幅を指定
Shift	0 or 1	Post-shift option (1=yes, 0=no)	←シフトのオプション指定

〔図7.6〕コンパイルド・セルの例：デュアルポートRAM(NCRデータブックより)



←CAD上のシンボル

INPUT PARAMETER	ALLOWED RANGE	EXPLANATION	
Num_Words	2-2048, even only	Number of data words in the RAM array. The number of address lines for each port will be n = $\log_2$ (Num_Words). Number of bits = Num_Words×Word_Width must be ≤ 16384.	←ワード数の指定
Word_Width	2 - 32	Number of bits in a data word. This effects the number of DINA, DINB, DOA, DOAT, DOB, and DOBT pins. Number of bits = Num_Words×Word_Width must be ≤ 16384.	←1ワードのビット幅の指定
PortArw	0 or 1	0: port "A" is read-only. 1: port "A" is read/write. If port "A" is read-only, then the DINA and A_WR pins are removed from the schematic symbol.	←ポートの動作指定

〔図7.7〕シー・オブ・ゲートの例(セイコーエプソン，カタログより)

■SLA10000シリーズ

特長＼機種名	SLA1020	SLA1024	SLA1029	SLA1034	SLA1039	SLA1049	SLA1060	SLA1073	SLA1081	SLA1152 *3	SLA1255 *3	
搭載ゲート数 *1 (ゲート使用効率)	20,216 (50%)	24,424 (50%)	29,120 (50%)	34,138 (47%)	39,644 (47%)	49,489 (47%)	60,653 (45%)	73,353 (45%)	81,320 (45%)	152,256 (40%)	254,743 (40%)	←チップ上の最大容量であってこの半分程度が実効ゲート数
テクノロジー	0.8μmシリコンゲートCMOS，2層Al，Sea of Gates											
I/Oレベル	TTL，CMOS											←I/Oセルで選択できる
遅延時間 内部ゲート *2	0.3ns (5.0V時標準)，0.5ns (3.0V時標準)											←ディスクリート部品では実現が困難！(ブレッドボードが作れない)
遅延時間 入力バッファ *2	$T_{PLH}=T_{PHL}$=1.2ns (5.0V時標準)，2.0ns (3.0V時標準)											
遅延時間 出力バッファ	$T_{PLH}=T_{PHL}$=3.5ns (5.0V時標準)，6.0ns (3.0V時標準) C_L=50pF											
TOTAL入出力端子数	120	132	144	156	168	188	208	228	240	328	424	
電源専用端子数	8	8	8	8	8	8	8	8	8	8	8	←ピン位置が指定される場合が多い
出力モード	ノーマル，オープンドレイン，3ステート，双方向，I_{OL}=2, 6, 12, 24mA											

*1 2入力NAND換算
*2 パワー NAND F.O.=2, AL=2mm
*3 開発中

大出力のセルはチップ上で大きなサイズとなる(ゲート数が多い)

めること(スペースまたはスピードのメリット)を追求したような話になりますから，コストよりも，どれだけ短期間に確実な設計と検証を行えるか，という開発手法の検討によって決定されることでしょう．

また，もし全体の回路規模が数万ゲート以上だとしても，それを1チップとするのが最善の方法とは限りません．あとで詳しく述べますが，ゲートアレイで数千ゲート規模，スタンダードセルで1万～2万ゲート規模，といわれるコスト効率のピークをうまく生かして，最適な規模の何チップかに分割するほうがローコストの場合が多く，ここがASICシステム設計の大きなポイントとなります．この場合，

- 各チップのゲート規模がローコストの最適エリアに入っているか
- 各チップの分割ポイントでの信号線の数が最適(ほぼ最小)になっているか
- 分割の前後でのタイミング信号の供給と，外部での信号遅延の影響

[コラム]
ビットスライスCPU

ここでは，**マイクロプログラム**方式という基本的技術の例として，「ビットスライスCPU」について考えてみましょう．あまり目につかない特殊なCPUのように思えるかもしれませんが，製品としてのビットスライスCPUは非常に少ないにしても，この考え方はどのCPUの内部にも共通しているものなのです．

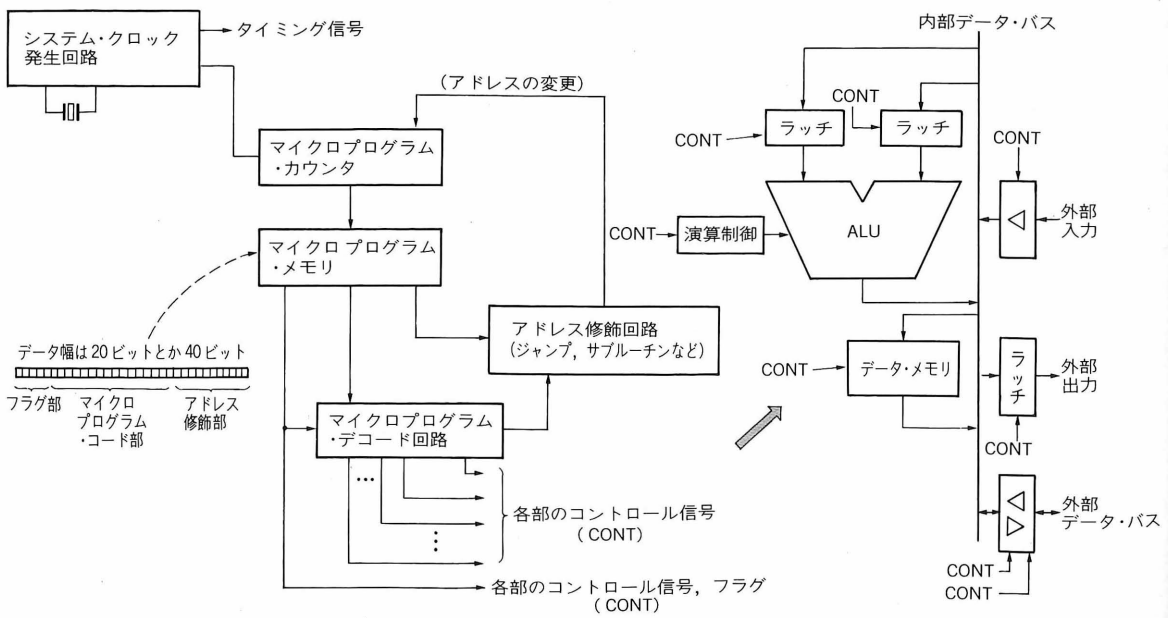
● CPUの動作をあらためて考える

一般的に，CPUの中身の構成と動作を考えてみると，

- システム・クロックにしたがって進むマイクロプログラム・カウンタ
- このカウンタ値によって読み出されるマイクロプログラム・メモリ
- このメモリ出力を動作命令と解釈する論理回路
- この動作命令によって所定の処理を行う演算回路
- このカウンタ値を動作命令によって変更させるアドレス修飾回路

などがあります(**図A**)．このような中心部があれば，そこから外部にプログラムをフェッチするためのアドレスを出し，外部のメモリから得られたプログラムにしたが

〔図A〕ビットスライスCPUの例



などの，普通の設計CADツールでは答えてくれないような要因が入ってきます．ここをなんとかクリアして，うまく効果的なチップ分割ができると，「美しい」システムが実現されるわけです．

● DSP →スタンダードセル

スタンダードセルに置き換えられることの多い分野に，ディジタル信号処理システムのハードウェア(一例としては，専用のDSPチップを使っていた，リアルタイムのデータ処理システム)があります．たとえば3次元の画像信号処理回路を，CPUによってソフトウェア的に実行しようとすると，100倍から1000倍も時間がかかってしまって実用にならないために，従来は専用ハードウェアを使ってきました．専用ハードの大きなボード上には，高速な乗算・累算のための演算器LSI(バイポーラの時代には放熱フィン付き!)が搭載され，多数の高速バイポーラ・メモリや，ハイスピードTTLのセレクタ，ラッチなどのタイミング回路が必要でした．

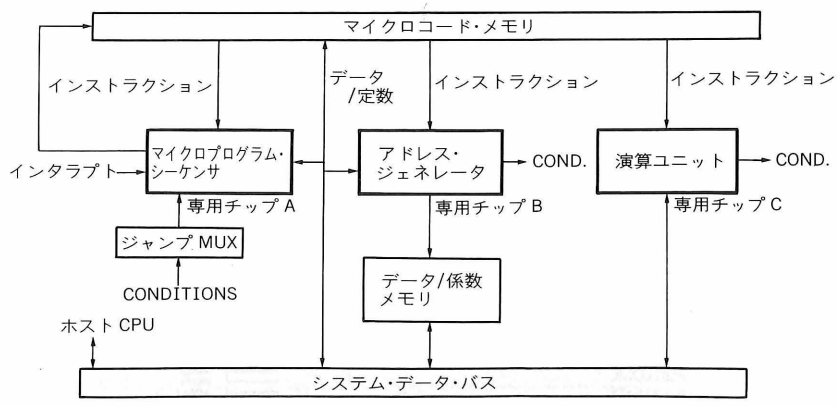
った動作をしていく，といった「ノイマン方式」のCPUの動作につながります．

このように見てみると，CPUそのものの内部にも，CPUと同じような原理で働く機能ブロックがあることに気づくでしょう．つまり，各種の機能ブロックのチップが別々に用意されていれば，それらを組み合わせて「オリジナルのCPUを構成する」ことが可能であることを意味しています．じつはこれがビットスライスCPUの考え方で，その歴史はZ80などの汎用CPUよりも長く，かなり古い技術なのです．現在のCPUは8ビットか16ビットか32ビットと相場が決まっていますが，4ビット単位，あるいは1ビット単位で，任意の(最適の)ビット幅のCPUを模索していた，というわけです．

ビットスライスCPUでは，一般のCPUの「1命令が××マイクロ秒」などというサイクル・タイムの中で，より多くのステップ数のマイクロプログラムを実行するのですから，論理回路もマイクロプログラム・メモリも，相当に高速なものを必要とします．また，普通のCPUが「機械語命令」として提供している機能そのものを，レジスタとかALUのデータ処理として，文字通りバイナリ・コード(完全な2進数)として設計していかなければなりませんから，そのハードウェアを理解するのは，普通のCPUを「使う」程度のものよりはるかに難しいことになります．しかし，ビット幅，演算機能，レジスタ・モデル，アドレシング・モードなどのCPU機能を，すべて自分の希望するように実現できるビットスライスCPUは，ASICが登場するまでは，ほとんど唯一の「究極の方法」だったのです．

現実にビットスライスCPUが活躍したのは，コスト的な条件から，汎用の大型計算機や航空機産業，宇宙産業とか軍用がほとんどでした．しかし，現在でもプロセスをバイポーラからCMOSに変えて，一部のメーカから細々と提供されています．また，**図B**(アナログ・デバイセズ社)のように，各ブロックの専用チップをこの技術によって組み合わせて，CPUのソフト処理では不可能な高速信号処理システムを構築するアプローチも展開されています．マイコン技術者としては，CPUそのものというよりも，DSPと同類の「大規模同期システムの設計技術」として，機会があれば体験してみたい一大技術です．筆者の場合，かつてアナログ・デバイセズ社が開催した技術セミナに参加して，この技術を詳しく知って「目から鱗が落ち」，やがて大規模ASIC設計の現場で非常に役立った経験がありました．

〔図B〕マイクロコード(マイクロプログラム)によるDSPシステムの考え方(アナログ・デバイセズ社技術セミナ資料より)



た．冗談話でなく，10 A とか 15 A を消費するこのボードの周囲では，冬場も暖房がいらなかったのです．

ところが，これを最近の技術によってスタンダードセル化すると，たとえば 1 万円の単体 DSP がチップ上の 2 ミリ四方の演算ブロックに吸収されてしまいます．高速メモリも高速ロジックも，チップ上のマクロセルと普通のゲート(ASIC 内のゲートは非常に高速で，単体 IC では最高速タイプに相当する)として収まります．そして，たとえば全体で 10 数万円の巨大ボードだったシステムが，1 個ウン千円の LSI を搭載したハガキ大の基板になり，場合によっては CMOS の電池駆動となる，というような大幅な変身を遂げることになるのです．

最近の例では，ハイビジョンの MUSE デコーダが典型的ですが，試作段階でのハード・ロジックによる巨大ラックと，ASIC 化されたシステム・ボードとの大きさ・コスト・消費電力の差は，なかなか示唆的です．そして，ハイビジョン・システムのチップ化の場合，さらに進んだ最近の世代では，チップ数も消費電力も減少させながら，性能を向上させつつ，よりローコストになっています．おそらく一般的な製品として家庭に出回るころには，現在でも 10 数チップになっている主要な部分を，ほとんど 1 チップ化してしまうところまで進歩していくことで，最終的な製品価格が 1 桁どころか 2 桁ちかく低下しているだろうと予言できます．

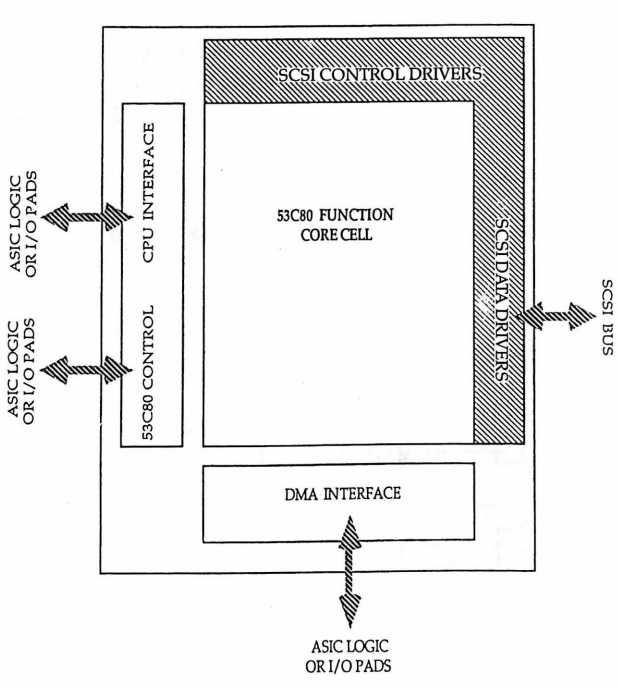
DSP を活用したシステムというのは，単純にデータ・バス上に乗算器を置いて，CPU が周辺回路として利用するようなスタティックな場合を除いて，ある意味で別の技術(ダイナミック回路)が必要となります．ここには一種の技術的飛躍があるのですが，ビットスライス CPU の基本にある「マイクロプログラム方式」(**コラム**参照)を使った回路技術を何かの機会に勉強すると，技術的な持ち駒がまた一つ増えます．筆者はこの部分を「ハードとソフトの技術の最後の接点」と考えていますが，フレッシュマンの皆さんも，いずれはここを突破するように，周辺の技術からトライしていきましょう．

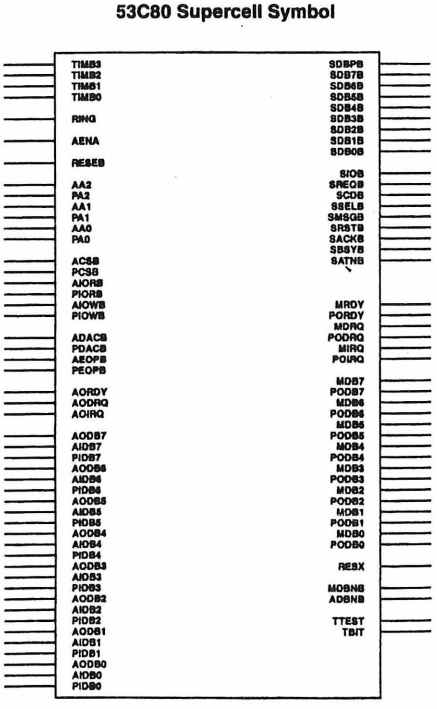
CPU コア内蔵「究極チップ」へ

●メガセル内蔵 ASIC

さて，スタンダードセルによって「ある機能ブロックを高密度にチップ上に搭載する」流れが本流となる

〔図7.8〕メガセルの例：SCSI インターフェースセル(NCR データブックより)





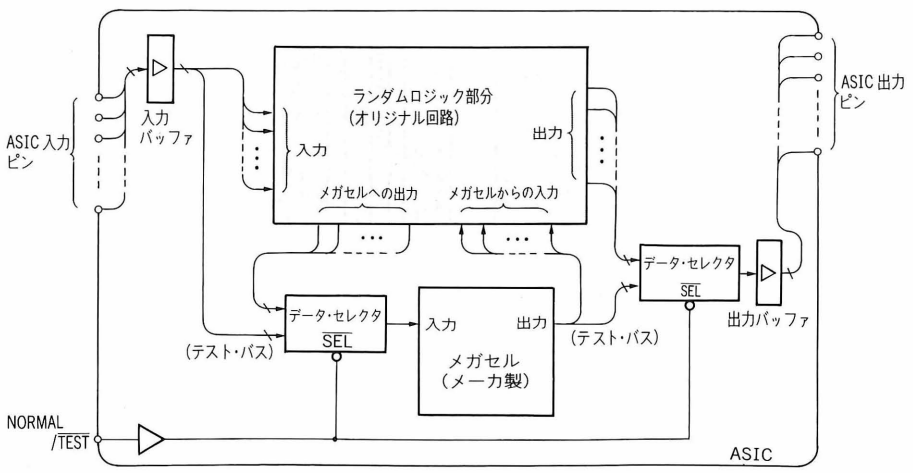
につれて，「システム・オン・チップ」の思想はますます進展していきました．コンパイルド・セルのメモリ・ブロックに続いて提供されてきたのは，CPU 周辺 LSI の機能をそのまま搭載してしまう，という「**メガセル**」のシリーズです．国内の半導体メーカは CPU のセカンド・ソースとして，各種の周辺 LSI ファミリも製造していますから，チップ上のマスク・パターンを使い回しできる利点があったのでしょう．現在では，シリアル通信ポート，DMA コントローラ，SCSI コントローラ(**図7.8**)などの機能ブロックが，LSI 設計上は一つのブラックボックスとして，CAD の回路図中に置けるようになっています．

この登場は，新しいゲートアレイの新ラインアップほど派手に PR されなかったものの，マイコン技術者には朗報でした．もともと ASIC の用途としては，CPU の周辺回路を 1 チップ化する目的がかなりの割合を占めています．ところが，UART，CTC，PIA などの機能を，標準論理 IC に相当するランダムロジックで組むと，意外とゲート数を食い，全体の規模が大きくなってしまいがちでした．ゲート数(つまりチップ単価)が上がってしまうのなら，汎用の周辺 LSI を別に使ったほうがいい，ということなってしまうのです．

そこで，メーカが十分に圧縮した(冗長な自動配線でない)レイアウト・マスクによる，コンパクトなメガセルというのは，ASIC のオリジナル設計のメリットと，専用ブロックの高機能をともに活用できる，きわめて強力な方法となりました．実際の設計の際には，メガセルのテストのための特別な「テスト・バス」を配線して，セルのテスト・モードでは外部ピンとメガセルとを直結するようにします(**図7.9**)．ただし，メガセルのテスト・パターンはメーカから提供されますから，これを機械的にテスト・プログラムの一部として挿入してやればいいことになります．この方法は，セル・コンパイラによって作られたメモリ・ブロックや，乗算器ブロックでも共通の手法となります．また，このようなテスト・バスは，ASIC 設計を階層的に行う場合の，各ブロックのテストの際にも活用できるものですから，全体のテスト効率の向上にも役立ちます．

このように，マクロセル，コンパイルド・セル，メガセルなどを活用したスタンダードセルは，たんなるランダムロジックのゲートアレイよりも，ある意味で簡単に大規模ロジックを実現できるものです．そして個々の回路技術そのものばかりでなく，**大規模回路を「階層的に」構築していく**，という技術的な発想法も大切です．つまり ASIC とは，個々のゲート回路の組み合わせから，ハードウェアとして集積度と複雑さが 1,000 倍から 10,000 倍のオーダで向上したシステム，ということができますが，単純にゲート数が増えていくのでは，とても人間の能力では全体の動作を正確に把握できません．それを効果的に補ってさらに大規模なシステムをチップ上に実現するために，これら各種の高級セルが階層化の進展とともに充実し，さらにソ

〔図7.9〕ASIC 内メガセルのテスト回路の例



フトウェア技術による開発支援環境も進歩してきたというわけです．

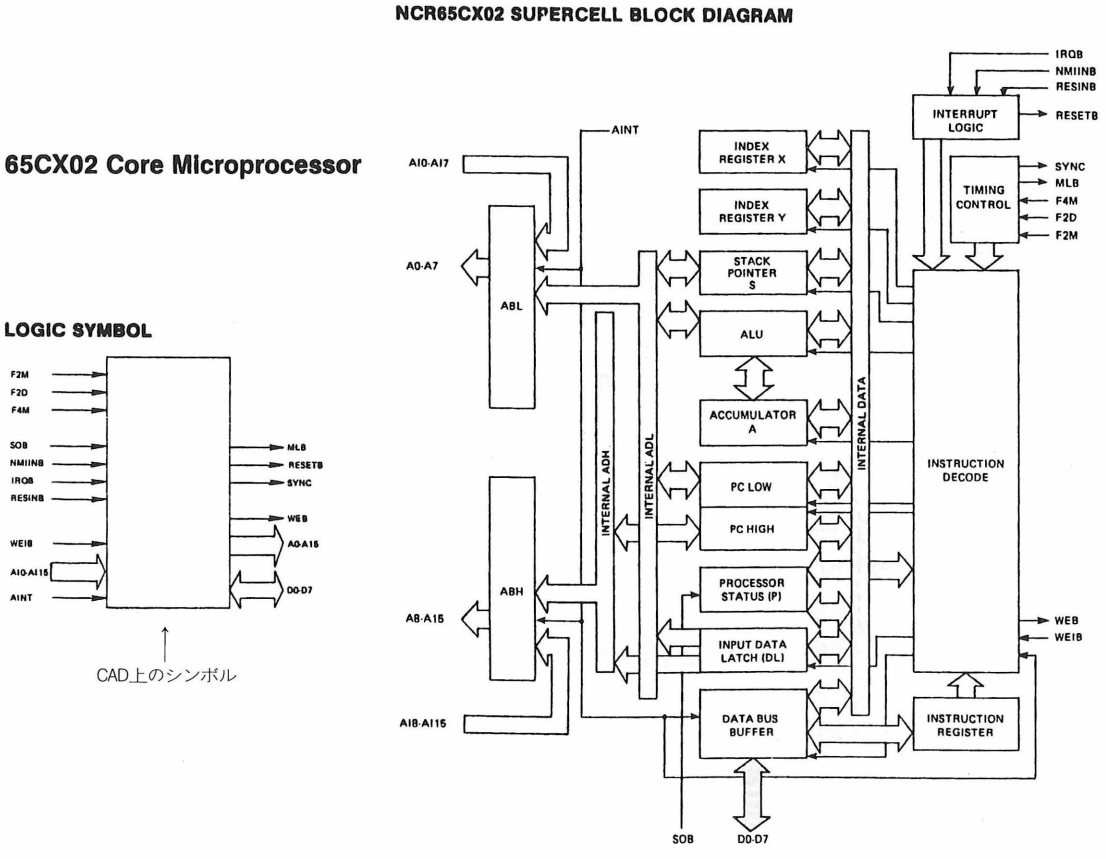
● CPU コア内蔵 ASIC：「究極チップ」へ

ASIC によるハードウェア技術の発展は，ついにはソフトの総本山である CPU へ到達することになります．チップ上に搭載するハードウェア要素が，たんなるゲート回路，複数のゲートを組み合わせたマクロセル，メモリなどのコンパイルド・セル，周辺 LSI 相当のメガセル，と発展してくれば，当然ながらそのつぎは，CPU そのもののセル(これを「CPU コア」と呼ぶ)が期待されるわけです．CPU コアという特別なメガセルをもった，ASIC 技術の頂点にあるこのタイプの LSI は，一般に「コア CPU タイプの ASIC」などと呼ばれています．しかし混乱を避けるために，ここでは仮に「究極チップ」とでも名付けて，「CPU コア」という主役と区別していくことにします．

ところで，マイコン技術者として，中規模のゲートアレイをパソコン・ベースの LSI 開発環境で設計したり，中規模のスタンダードセルを EWS ベースの開発環境で設計した人にとって，あるいはさらにメガセル方式の大規模 ASIC を設計した経験がある人にとって，このチップ上に CPU コアという新しいメガセルが加わった(**図7.10**)としても，それほど外面上の飛躍はありません．設計 CAD のソフト上では，CPU コアといってもたんなる一つの箱で，アドレス・バスやデータ・バスなど，ちょっと多めの入出力ラインがあるだけなのです(ただし，これは「普通の CPU ボード」をオン・チップ化したというレベルのシステム技術での話．さらに上のレベルについては後述)．

CPU コアの部分の選択肢としては，現在のところは各メーカが提供するライブラリから選ぶだけです．た

〔図7.10〕CPU コアの例：6502(NCR データブックより)



とえば，CPUコアのアーキテクチャに注文があるので一部改変したい，というのは当分は無理な条件で，いずれ将来的には許容されていく(おそらく各メーカのオリジナルCPUにおいて)ことでしょう．また，CPUコアのシミュレーション・テストについては，スタンダードセルのメガセルの部分で述べたのと同様の手法によって，ランダムロジック部分とは切り離して，テスト・バスによって外部からテストします．CPUコアのテスト本体は，通常はLSIメーカ側が行うことになっていますから，テスト・モードへのエントリ部分を，ハード(バス切り替え回路)とソフト(テスト・プログラム)の両方から揃えることになります．

「究極チップ」の技術ポイント

●「究極チップ」の技術的ハードル

この「究極チップ」(CPUコア内蔵タイプASIC)は，ハードウェア技術の点でASICの延長上にあるばかりでなく，もちろんCPUというソフトウェア技術の結晶でもあります．単体のチップである，汎用CPUや1チップ・マイコンであれば，アドレス・バスとかデータ・バスといった端子のデータシートが最初から揃っています．ところが，「究極チップ」の外部端子というのは，全体のピン配置(電源ピンの位置によってマージンが大きく変化する)や，個々の端子ごとの電気的特性(スレッショルド・レベル，ドライブ能力，耐ノイズ特性，負荷容量，スイッチング・スピード)などについても，設計するエンジニア自身が規定していかなければなりません．つまり「既存のCPUを使うだけ」の技術では不足で，ある意味で「CPUそのものが作れる」技術が必要になります．この意味では，製品のボード・マイコンを使うだけでなく，オリジナルのCPUボードを完全に手配線で作ったような経験とか，テスタ片手にアナログ回路をカット・アンド・トライで製作したセンスなどが生きてきます．

また，この「究極チップ」を設計するうえでの最大のポイントは，「システム内の多重(階層的)同期」です．もちろん，ある程度のゲートアレイ以上の規模のASICであれば，たいていは同期回路(ダイナミック回路)としてシステムを設計することになります．ところがこれに加えて「究極チップ」の場合には，「ディジタル回路の時間」と「CPUコアの時間」という，異なったスケールのタイミングを同期させることが必要となります．おそらく初めて設計した人は，意外なほど「遅い」CPU，というものを実感することになりますが，この感覚はちょっとイメージ的に理解しにくいので，つぎに例を示して考えてみましょう．

● ASICとCPUコアとの共存技術

最近のASICの「ゲート遅延」というのは，1ゲートあたり1 ns(ナノ・セック)を切るのが普通になっています．チップ上のRAMブロックのアクセス・タイムも，単体チップの高速RAMよりも速いぐらいで，乗算器ブロックも非常に高速です．このため，ASICと同等の回路をディスクリート基板として作って検証するはずの「ブレッド・ボード」が，Fシリーズとかの高速TTLを使っても，現実にチップ化された場合のスピードに追いつかない，といった現象も起きています．この場合，ブレッド・ボードのクロックを本番のASICよりも遅くして，検証する際の視点として時間軸を引き伸ばして解釈する(スローモーションを見る感覚)ことになります．ごく普通のASICであっても，チップ上のシステム・クロックは，たとえば100 ns(10 MHz)のオーダで設計することができて，高いパフォーマンスを稼げるのです．

さて，このようなASICシステムにCPUコアを組み入れた場合，LSIのテスト・パターンがどうなるかを考えてみましょう(**図7.11**)．たとえば，CPUコアが1サイクル1マイクロ秒(1,000 ns)でプログラムのフェッチやインストラクションの実行を行うとします．すると，このLSIのテスト・パターンとしては，システム・クロックの2倍の20 MHz (50 ns)を単位とするステップで表現したとして，電源が入って，CPUコアのリセット条件である10サイクルないし20サイクルの「リセット入力」期間を経過して，いざCPUがリセット後の最初のアドレス信号をアドレス・バスに送出するまでに，軽く100から200クロックが経過していることになります．この場合，LSIテスト・プログラムは，システム・クロックが200発ほど並んでいる間，CPUコアのリセット入力とクロック入力を除くすべての端子が，「不定」状態のマークでべったりと塗りつぶされていることになります．

この後，プログラムROMから得られる最初のデータを想定して，データ・バスにインストラクションのデータを与えます．それを受けてCPUコアがリセッ

ト・ルーチンに飛んで，また同様にメモリをアクセスしていきます．こうして，イニシャライズ・ルーチンのプログラムにしたがって，データ・バスからいちばん最初の周辺ブロック初期化データを出力する頃には，ハードウェアのテスト・パターンは，すでに数千ステップほど進んでしまっているのです．

●「究極チップ」の技術的ポイントの核心は

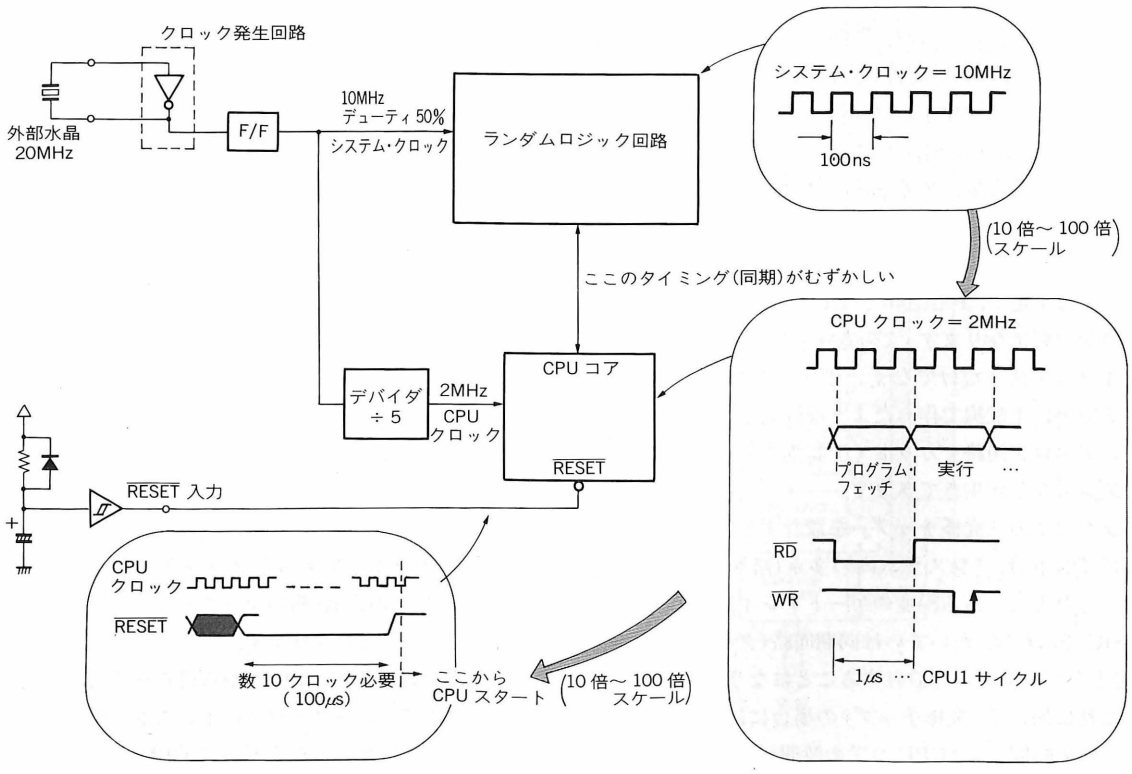
このように，ASICの中のランダムロジックに比べると，CPUコアというのは驚くほど遅い機能ブロックであるために，「究極チップ」として全体を同期動作させる場合には，そのタイミング回路とともに，テスト・パターン上での同期をとることに苦労するのです．もちろん，普通のCPU回路と同様に，実際に走るときには非同期なので，独立したものとして別々にテストする(片方をテストするときにはもう一方をDon't Careとする)という方法もあります．これは，前に述べた「普通のCPUボードをオン・チップさせた，というレベル」のシステムということになります．

しかし，なんといっても「究極チップ」のメリットは，チップ上のCPUとランダムロジックとを同期させるシステム・パフォーマンスにあります．この部分は，じつはASIC技術の一つの重要なポイント(ソフトとハードの融合点)なのではないか，と筆者は考えています．つまり，CPUと周辺ハードウェア回路が独立であるかぎり，そのシステムは「非同期待ち合わせ」，「共有情報の分散保持」というデメリットを本質的にもっていて，全体のシステム効率としては，ある一線を越えることができない宿命にあるのです．この点をさらに詳しく，具体的に考えてみましょう．

●非同期待ち合わせシステムの非効率

ここでは，同期的に動作するハードウェア回路の例として，一定周期でパイプライン処理を行っているディジタル画像処理回路と，同じチップ上のCPUコアとがインターフェースすることを考えます．CPUコアは，たとえばパネル・スイッチをスキャンしていて，ボタンが押されたことを検出すると，ハードウェアの演算処理パラメータを変更したくなります．ところが，信号処理は一定のペースが勝手に乱されるわけにはい

〔図7.11〕CPUコア内蔵ASICの時間スケールの例

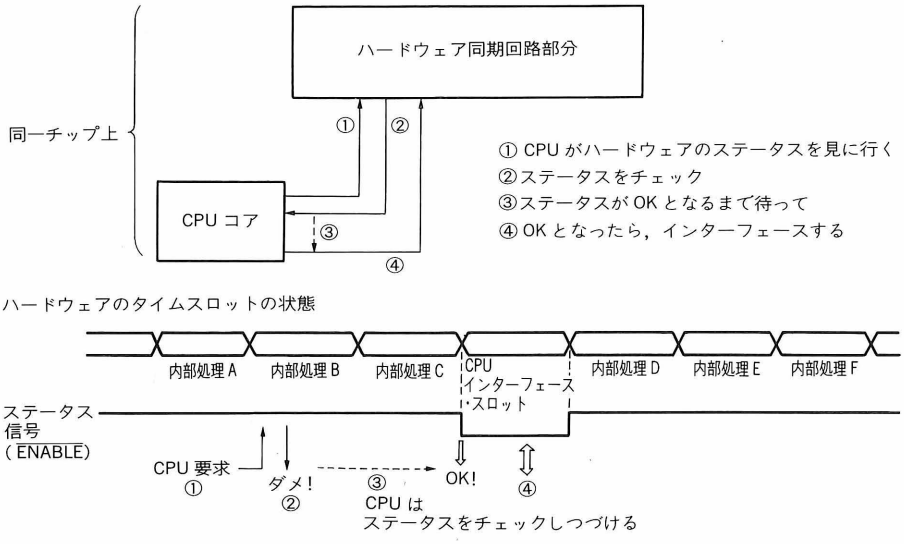


きませんから，たとえばハンドシェイク回路のフラグ(ウェイト信号として作用)が「書き込みOK」となるまで，CPUコアはそこで待たされます(**図7.12**)．

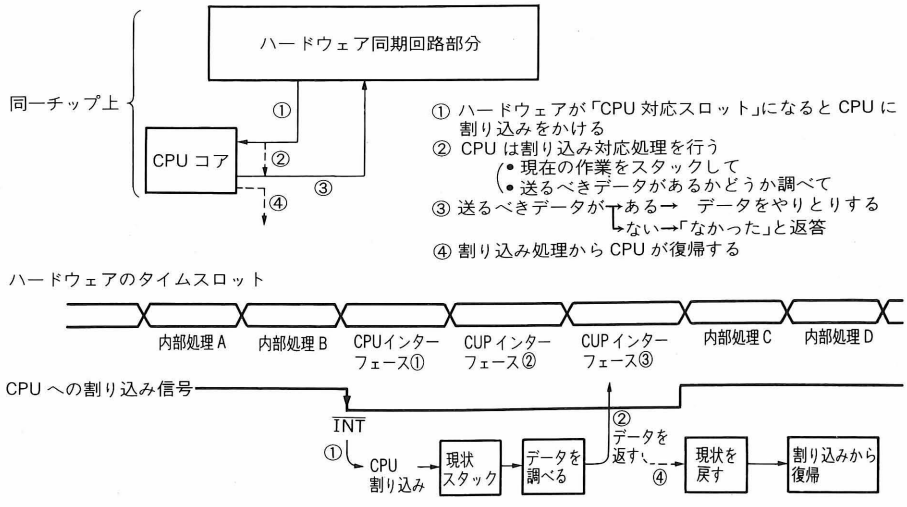
このように，いちいちCPUコアがハンドシェイクで足踏みしていては，全体の効率が上がりませんから，逆にハードウェア側が「データ変更可能」なタイミングに，CPUコアにデータ要求の割り込みをかける，という方法もあります．システム・クロック回路の中に，そういう処理タイムスロットを設計しておくわけです．すると今度は，CPUコアが処理を中断してデータを探しにいったり，場合によっては「今回は変更するデータがなかった」などという間抜けな返答をすることにもなります(**図7.13**)．

このような非同期インターフェースの定石として，上のような非効率を回避するためには，ダブル・ラッチによるタイミング調整回路を設けたり，DMA転送回路のような特別な回路ブロック，あるいはデュアルポートRAMのような面倒な(ゲートを食う)ハード

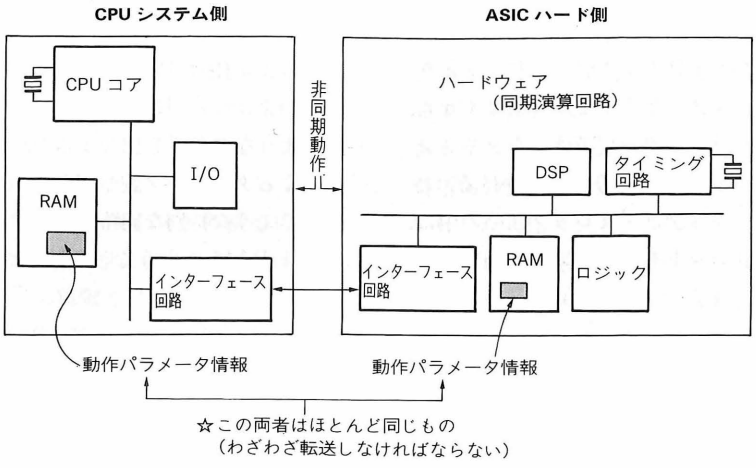
〔図7.12〕非同期待ち合わせの例 (1)



〔図7.13〕非同期待ち合わせの例 (2)



〔図7.14〕共有情報の分散保持の例

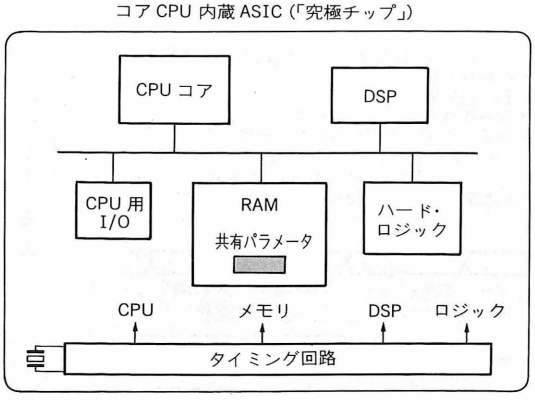


ウェアを採用しなければなりません．そしてこの場合でも，双方向インターフェースの場合には「同時アクセスの調停」，あるいはアクセス頻度の可能性によってはトリプル・ラッチ回路へと肥大して，かなりの規模のインターフェース回路を必要としてしまうのです．

●共有情報の分散保持という非効率

また，同期ハードウェア回路とやりとりされるデータというのは，通常は1個のデータではなく，複数のアドレスをもつレジスタ群とか，あるいは内部メモリに対するアクセスとなります．ところで，CPU コアのソフト上で作られてハード部分に転送されるこのデータは，CPU ワーク RAM 内にも同じものが並んでいるのが普通です．つまり，CPU システム側とハードウェア側のそれぞれに，ほぼ同じような情報が置かれていて，この両者を非同期インターフェース回路を経由して，苦労して転送してばかりいるのです(**図7.14**)．

〔図7.15〕「チップ上に情報を共有する」というアイデア



☆ チップ内の RAM に共有パラメータをもって CPU と DSP が交互にアクセスする

●身近になってきた「究極チップ」の技術

そこで，せっかく同じチップ上に CPU コアと同期ハードウェアとが同居しているのですから，「チップ上の RAM を共有する」と考えるとどうでしょうか(**図7.15**)．CPU コアのワーク RAM からハードウェアのパラメータ・メモリに「転送」するのでなくて，同じメモリをワーク RAM として，あるいはパラメータ・エリアとしてアクセスする，というわけです．これは，非同期回路をインターフェースするデュアルポート RAM ではありません．CPU コアとハードウェア回路の全体を「同期」させて，それぞれ「当然のように」，「自分のタイミングで」アクセスしているのに，全体は完全に同期している，というシステムなのです．余分な回路も，待ち合わせの無駄も，情報資源の重複もありません．

よく考えてみると，じつはこれは「ビットスライス CPU」の世界で昔から追求されてきた，基本的な発想なのです．わかってしまうと何でもないことなのですが，この技術をひとたびモノにしてみると，その威力と魅力にとりつかれることでしょう．

CPU コア内蔵 ASIC(究極チップ)によるマイコン・システムは，もちろん「システムの全部を1チップ化

する」という外見上の美しさもありますが，その動作の細部にまで設計思想を浸透させることができるために，ここで検討したような，おそらく想像以上の機能(付加価値)を得ることができるのです．CPUコアをサポートするメーカも，国内・海外ともにかなり増えてきましたから，チャンスがあれば，積極的に挑戦してみて欲しいと思います．

セミカスタムCPUからのアプローチ

●セミカスタムCPU

システムをチップ上にまとめるという目標に向けたもう一つの流れとして，ASICとは別の方向からのアプローチがあります．この一例としては，1チップ・マイコンの発展したかたちとして，最近のASSP(Application Specific Standard Product)の流行に乗った新しいCPUがあります(**図7.16**)．一種の1チップ・マイコン・ファミリとして，メインのCPU部分に加えてチップ上に搭載する機能ブロックとして，従来のメモリ(ROM・RAM)ばかりでなく，各種周辺LSIに相当する機能から「取捨選択して」,「組み合わせる」というものです．その昔，「電子ブロック」というオモチャがありましたが，それと同じ感覚で，一定のチップ・サイズの範囲に，それぞれのサイズをもつ機能ブロックをパズルのように並べて，希望する機能の1チップ・マイコンを手軽に実現できる時代になったというわけです．

この富士通の例は，可能な組み合わせをずらりとメーカ側で用意して，「どれでも好きなものをどうぞ」というものです．実際には，それぞれのチップをあらかじめ大量に用意していては効率が悪いですから，受注によって自動的に必要な専用メガセルを組み合わせてレイアウトするような，かなり自由度の高い設計環境が揃っているのだろうと思います．また日立や日本電気のアプローチとしては，内部にCPU専用バスをもった一種のメガセルASICとして，ほぼ同様の1チップ・マイコンのファミリを提供しています．こちらは比較してみると，より「正攻法」というべきでしょうか．

1チップ・マイコンを使ってマイコン・システムを設計していると，チップ上に搭載されている周辺機能の一部にいろいろと不満の出る場合がありました．たとえば，チップ上に汎用タイマを2本もっているのに，実際のシステムでは3本を必要とするために，わざわざ一般のCTCを1個，基板上に外付けする必要があるとか，UARTが別途に1チップ必要になる(内蔵UARTは1組だけ)とか，チップ上に搭載されているステッピング・モータ制御回路部分がまったく無駄になる(使わない)，などのケースです．実際，1チップ・マイコンの内蔵周辺機能というのは，おそらく「最大公約数的判断」から設定されていますから，ちょっと一般的用途から外れると，「過剰な仕様」と「不足する仕様」に直面することになるわけです．

そこで，たとえば

- チップ上には，CPU以外に汎用タイマが8本だけ欲しい
- CPUとシリアル通信ポートを4組だけもったチップ

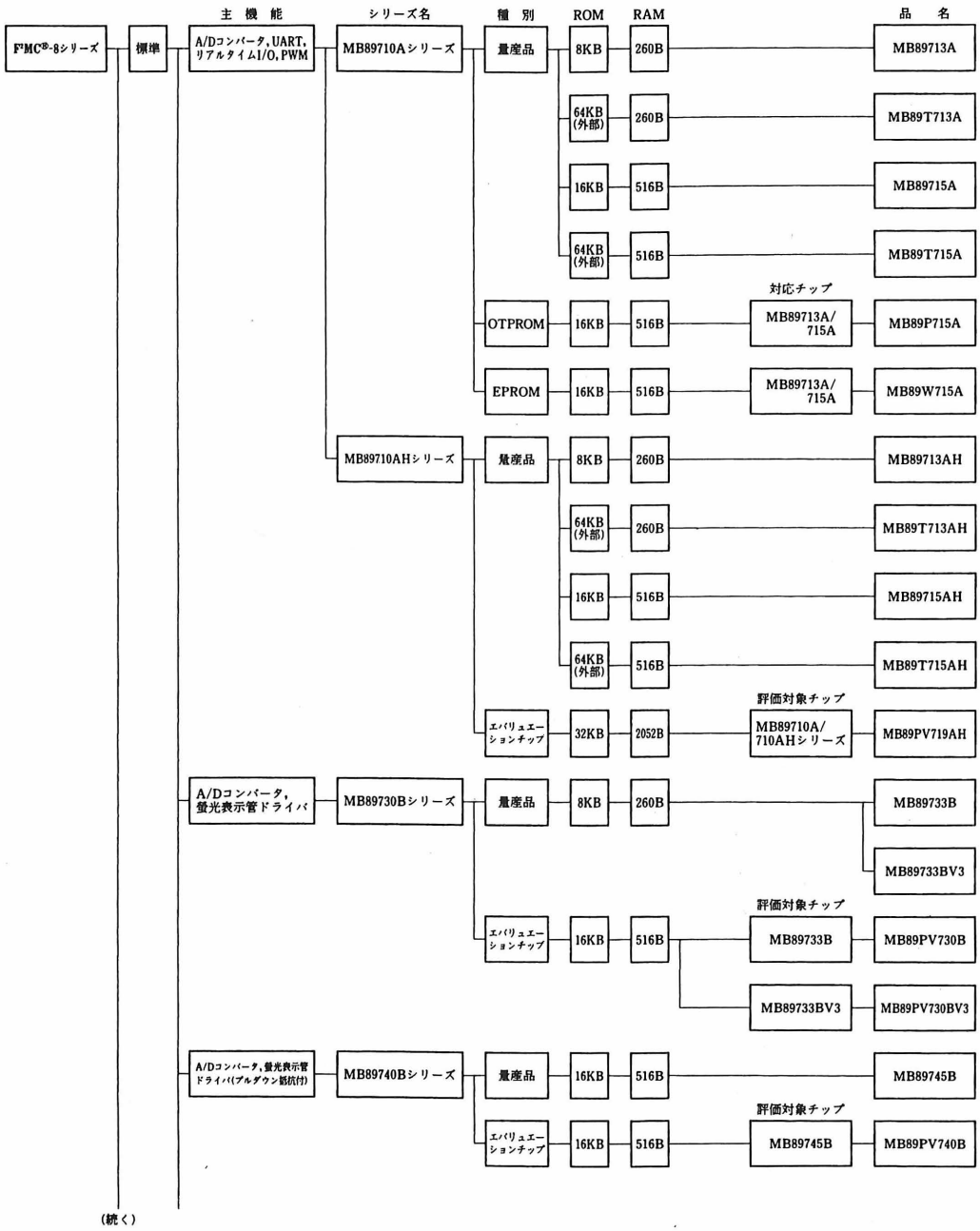
といった，従来ならばフルカスタムのLSIか，前節のCPUコア内蔵ASICの手法で実現するしかなかったオリジナルの1チップCPUが，このセミカスタムCPUによって得られることになったのです．メーカの提供する機能ブロックやCPUブロックは，メーカ側のレイアウトとシミュレーション・モデルにしたがったものですから，ASICのような設計作業とか，巨額の開発費用を必要とすることもありません．これが主流になる，というような状況にはなっていませんが，今後のひとつの流れを示唆しているタイプのCPUだと思います．

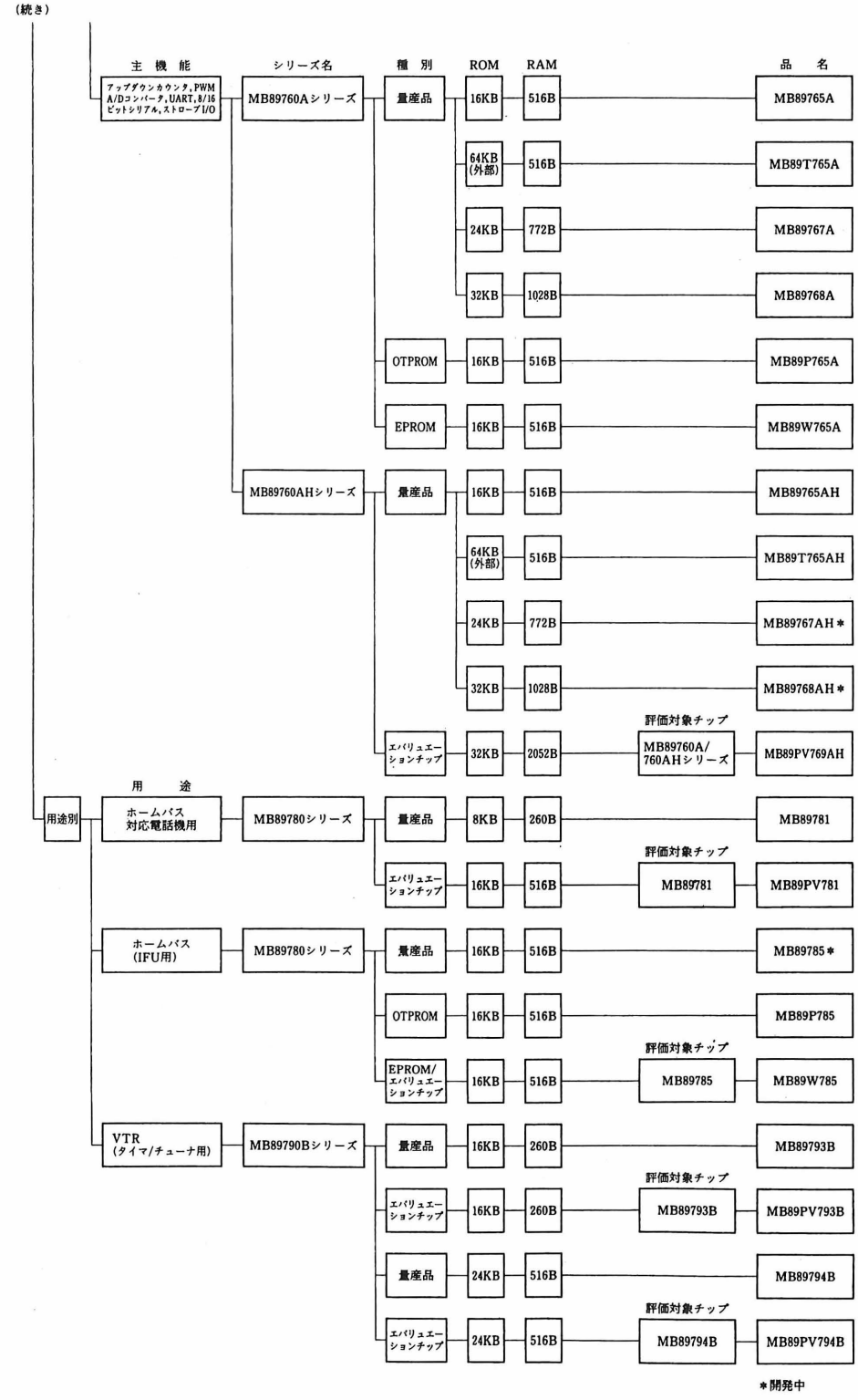
●ゲートアレイ内蔵CPU

マイコン技術者の要求の中に，この数年間にわたって変わっていないものがあります．「メーカから与えられる」だけの汎用CPUと，「ユーザ・オリジナル回路」を実現できるASICとを，同一チップ上に実現する「システム・オン・チップ」の希望です．このうち，一種のメガセルとしてCPUコアを搭載してしまうASICの「究極チップ」については，すでに前節で述べました．ところが，もう一つのCPU側からのアプローチの歴史は，意外にもASIC側とあまり変わらない時期からの古いものでした．

1チップ・マイコンのチップ上に，ASSPタイプCPUのようなメーカ提供の周辺機能ブロックを置く以外に，ごく簡単なゲートアレイを置くスペースを取れないか，という希望は根強いものです．ハードウェアとして本格的なランダムロジックを置くまでもない，

〔図7.16〕ASSP タイプ・CPU ファミリの例(富士通データブックより)




(続き)
主 機 能
シリーズ名
種 別
ROM
RAM
品 名
アップダウンカウンタ,PWM
A/Dコンバータ,UART,8/16
ビットシリアル,ストローブI/O
MB89760Aシリーズ
量産品
16KB
516B
MB89765A
64KB
(外部)
516B
MB89T765A
24KB
772B
MB89767A
32KB
1028B
MB89768A
OTPROM
16KB
516B
MB89P765A
EPROM
16KB
516B
MB89W765A
MB89760AHシリーズ
量産品
16KB
516B
MB89765AH
64KB
(外部)
516B
MB89T765AH
24KB
772B
MB89767AH*
32KB
1028B
MB89768AH*
評価対象チップ
エバリュエーションチップ
32KB
2052B
MB89760A/
760AHシリーズ
MB89PV769AH
用 途
用途別
ホームバス
対応電話機用
MB89780シリーズ
量産品
8KB
260B
MB89781
評価対象チップ
エバリュエーションチップ
16KB
516B
MB89781
MB89PV781
ホームバス
(IFU用)
MB89780シリーズ
量産品
16KB
516B
MB89785*
OTPROM
16KB
516B
MB89P785
評価対象チップ
EPROM/
エバリュエーションチップ
16KB
516B
MB89785
MB89W785
VTR
(タイマ/チューナ用)
MB89790Bシリーズ
量産品
16KB
260B
MB89793B
評価対象チップ
エバリュエーションチップ
16KB
260B
MB89793B
MB89PV793B
量産品
24KB
516B
MB89794B
評価対象チップ
エバリュエーションチップ
24KB
516B
MB89794B
MB89PV794B
*開発中

たとえば数百ゲート程度(TTLで数個，PLDなら2～3個ほど)の回路をもてるとしたら，CPUボードの細かい周辺部品がすべて吸収できますから，システム屋としてはぜひ欲しい付加価値なのです．ところがLSIメーカ側からすると，汎用品のCPUとカスタム設計の(得体の知れない)ゲートアレイとを混在させる，という発想は，設計環境やプロセスの関係で問題外のようでした．

●ユニークなCPUは第2グループから

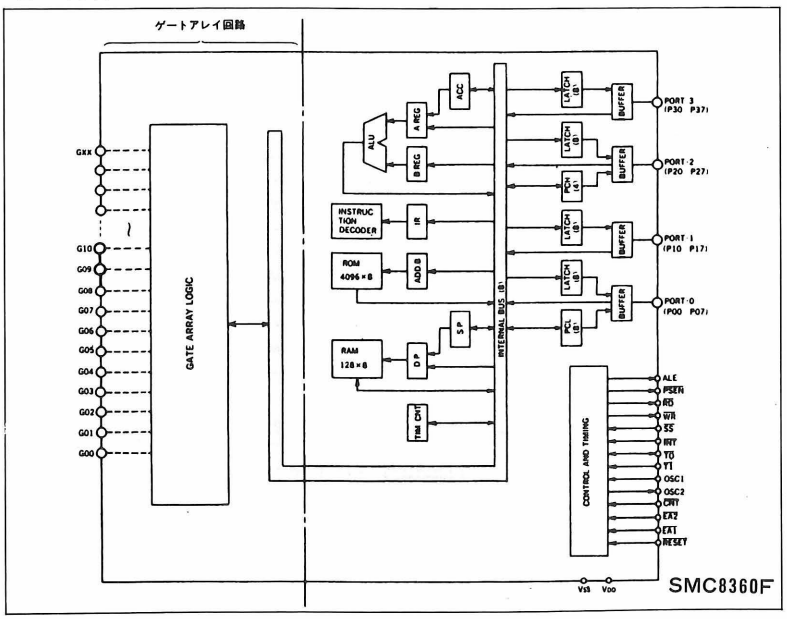
ただ数年前から1社，セイコーエプソンが1チップ・マイコンのラインアップの一部に，この機能(500ゲート程度のゲートアレイを内蔵)をもつCPUを提供していました(**図7.17**)．どうやら，汎用CPUやASICで先行した超大手メーカのほうが，やや古い設計環境(かつての中心は大型計算機ベースのバッチ処理)の関係で，あるいは大所帯の組織の大きさが災いしてか，機動力がなかったようにさえ思えます．ところが，ミニコンやEWS上のUnix環境の導入による，新

〔図7.17〕ゲートアレイ内蔵1チップCPU SMC8360F(セイコーエプソンのデータブックから)

■特　長

- ●CMOS LSI 8ビット並列処理CPU
- ●発振回路内蔵……50k～4MHz
- ●インストラクション実行時間 ……1μs Min
- ●インストラクションセット……104セット
- ●ROM容量……4K×8ビット
- ●RAM容量……128×8ビット
- ●ゲートアレイ規模 …… 2入力NAND×848ゲート
- ●ゲートアレイ……SLA6000シリーズ準拠
- ●入出力ポート……4ポート×8ビット
  ゲートアレイ部：33入出力端子(チップ)
  26入力端子(GFP5-8pin)
- ●外部プログラムROM拡張可能(ゲートアレイオプション)
- ●STOPモード解除回路(ゲートアレイオプション)
- ●割込み……外部割込み　INT, T1, T0のOR論理
  内部割込み　タイマ/カウンタのオーバーフロー
- ●サブルーチンネスティングレベル……8レベルMax
- ●動作モード……RUNモード，HALTモード，STOPモード
- ●電源……2.5～6.0V(周波数条件あり)
- ●パッケージ……QFP5-80pin(プラスチック)
  またはチップ

■ブロック図



しいLSI開発ツールの活用でASIC市場に参入した「第2グループ」メーカのほうが，このような面白いものをどんどん提供しています．

たとえば，ファミコンの心臓部となっているカスタムLSIは，「究極チップ」のパフォーマンスを最大限に活用していますが，じつはこれも，「第2グループ」のリコー製のチップ(CPUコアはApple-IIと同じ6502)であるのは，ファミコン・マニアの間では常識になっています．そして，すでにリコーの持ち駒のCPUコアは，スーパファミコンのLSIでは上位の16ビット・タイプ(65816)へと進化し，さらにオリジナル改良を加えたCPUコアも提供されているようです．

最近では，メモリLSIとしては大手でも，CPUやASICの分野では後発グループだったシャープから，Z80コア同様のタイプのチップと，なんと16ビットのV20コア(日本電気からライセンスを受けた？)をもつタイプが登場してきました．16ビットのほうはかなり大規模なチップ(どちらかというとSea of Gatesの中にCPUコアが浮かぶ雰囲気)で，比較できるクラスではありませんが，各半導体メーカの姿勢として，このようなユニークな方向に進んできているのは歓迎すべきことだと思います．

また，PLDやLCAで設計・検証した回路データを，そのまま無条件に変換してゲートアレイ化する，というメーカやデザイン・ハウスも登場していますが，この設計技術の展開として，実験・試作システムでLCAを使って検証済みの回路を，「CPUとLCA相当回路のゲートアレイとを搭載した」1チップ化によって実現するという，新しいセミカスタムCPUも発表されていくことでしょう．このように，LSI設計環境の進展と意欲的なメーカによって，これからはますます，美しいマイコン・システムを実現する可能性は増えていくことになります．あとは，古い技術を繰り返すだけで満足しないで，臆せずに新しい環境に飛び込んでいくエンジニア次第，ということなのでしょう．

● ASIC内蔵タイプCPUへ

汎用CPUのチップ上に，このようにゲートアレイというカスタム回路が搭載できるとすれば，ここからあと一歩で，「汎用CPUチップ上の一角に，大規模なスタンダードセルを！」という，これまた究極のシステム・オン・チップへと簡単に進んでいけそうです．ところが，システムの外見は「究極チップ」(CPUコア内蔵タイプASIC)と似てくるのですが，どうもこちらのほうの進展は，意外に難航しています(**図7.18**)．ここでは，ソフトウェア(CPU)からハードウェア(ASIC)への融合化の流れの最後の段階として，この「ASIC内蔵タイプCPU」(「究極チップ」との違いに注意！)のシステムについて考えてみましょう．

汎用CPUというチップは，もともと極限まで圧縮・最適化されたレイアウト・マスクをもっているLSIで，ASICのような冗長な設計手法とは別次元のもの，というのが従来のメーカ側の開発環境の前提だったようです．半導体の大手メーカの場合には，メモリやCPUなどの汎用LSIを大量生産する事業部と，ASICのようなユーザ個別対応設計の事業部(少量多品種)とでは，あらゆる意味でLSIの設計環境やビジネスの形態が異なっていた，という事情も大きいのです．このため，上に述べたようにASIC分野でユニークな展開を見せるメーカというのは，小回りのきく後発メーカ(第2グループ)に多く，後発のメリット(より新しいLSI設計環境：CPUといえどもたんなる一つのメガセルと見なせる)をフルに生かして健闘しています．見方を変えると，先発の大手メーカがあまり手を出さない，面倒な(いちいちユーザの要求に耳を貸す)ASIC分野からでなければ，なかなか半導体ビジネスに参入してこれなかったのです．

●マイコン技術者も進化する

前節の「究極チップ」の「CPUコアという名のメガセル」は，内部をブラックボックスとした一つの「シミュレーション・モデル」として，LSI設計CAD内で取り扱っています．そして，CPUとランダムロジックとが混在されることの問題点については，設計者であるユーザ自身が各種ツールを活用し，シミュレーションによって検証することが必要条件となります．レイアウト的な物理的問題という障害はあまり表面に出ることはなく，あくまで開発環境(ソフト)上の論理的な作業で済んでしまいます．

この点が，メーカ設計による汎用CPUチップ上に，ユーザのカスタムロジックを置くという立場では違ってくるところで，設計の信頼性の検証の部分がネックになるのです．つまり，汎用CPUチップは，いわばチップ上での「密度」が，ASICレベルの回路部分とは別格に濃密なため，ASICという冗長な「土台」の上にこの「異質な」マスクをポン，と簡単には置けないわけ

です．逆にいえば，従来の汎用CPUの高密度レイアウト・パターンを作り込んだチップ上に，ASICのような自由度をもつ設計環境をそのまま持ち込むことは困難だったのでしょう．

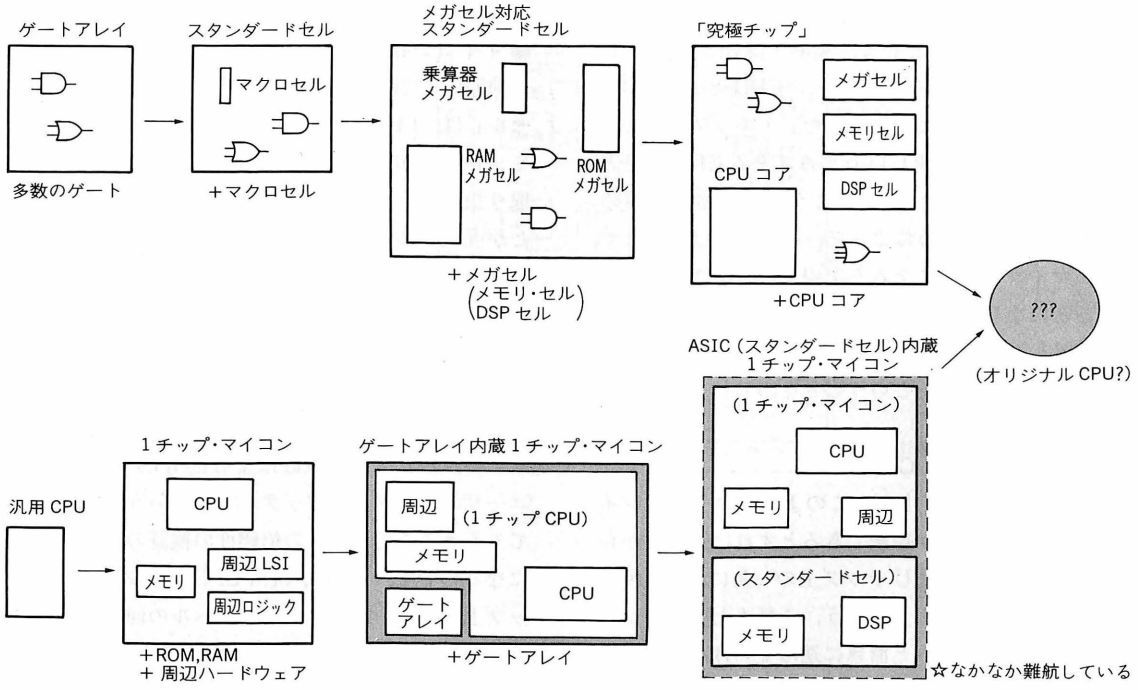
また，これとはまったく別のポイントもあります．CPUというのは，ディジタル・システムのまさに「中核」ですから，回路技術の知的所有権(特許，回路方式，マスクパターン，マイクロプログラムなど)の点で，いろいろと政策的に微妙なところがあるのも，この方式のアプローチの遅さに関係しているようです．セカンド・ソースのCPUはもちろんのこと，メーカ・オリジナルのCPUであっても，ASICのソフトウェア・モデルとして提供するCPUコアと，物理的に固定している汎用CPUのマスクとでは，不特定多数の設計ユーザに公開される技術情報であるだけに，扱いも別格となるのです．

どうやら，これからしばらくも「システム・オン・チップ」を実現するためには，LSIメーカが汎用CPUのチップ上にスタンダードセル用の自由スペースを確保するのを期待するよりは，ASIC側からのアプローチ(究極チップ)をとる必要があるようです．そこで，マイコン技術者としては，CPUをいろいろな視点から自由に眺められるとともに，ASIC技術を駆使してランダムロジックを設計できるようになることが望まれるわけです．この境地というのはまさに，ハード屋でもソフト屋でもなく，各種のCAD(ソフトとかツールというより「環境」そのもの)を自在に使いこなした，新しい「システム屋」の姿だろうと思います．メーカやASIC技術だけが進歩するのではなくて，エンジニアもまた，どんどん進化していくのです．

カスタムCPUへの道

いろいろな「システム・オン・チップ」を考えてきた旅路も，いよいよ最終段階となってしまいました．ここで筆者が描いてみたいのは，じつはLSIメーカ各社のエンジニアに，何年も前から希望している技術です．それは，一言でいえば「オリジナルCPUを作る」という，単純にして深遠な難題なのです．

〔図7.18〕システム・オン・チップの二つの流れ



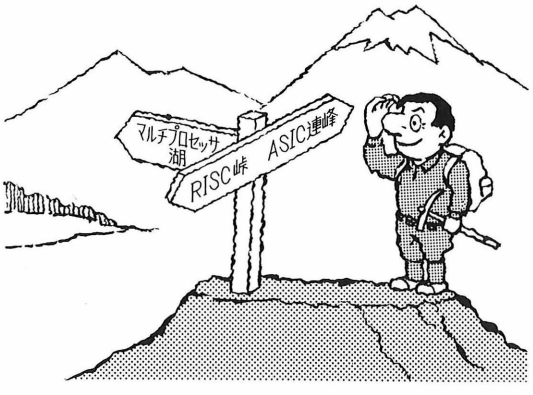
●カスタム CPU への道

じつは，現在のメガセル方式・CPU コア内蔵方式のスタンダードセルというのは，ほとんど技術的に「CPU を作る」ことを可能にしています．事実，ASIC の CPU コアを提供しているメーカは，セカンド・ソースの CPU であってもマスクのオリジナル版をもっていますし，もちろんメーカ・オリジナルの CPU コアについては，刻々とバージョンアップ・微細化を進めています．LSI 設計 CAD 上での CPU シミュレーション・モデルも，ほとんど毎月のようにバージョンを上げて完備しています．しかし，この手法のままで「オリジナル CPU を作る」となると，現在の「究極チップ」よりかなり冗長な，チップ上のサイズの大きなものになってしまいます．それはちょうど，ランダムロジックだけで，ビットスライス CPU を構成するようなものだからです．

ここで「本当のオリジナル CPU」として求めているのは，任意のビット幅・ワード数の RAM セルを自動生成するような，ASIC 用セル・コンパイレーション技術の延長上の，「CPU 設計セル・コンパイラ」なのです．つまり，CPU の ALU 機能，サイン・フラグの割り当て，具体的なレジスタ・モデル構成，命令の体系と種類，割り込みモード，アドレシング・モードの種類などの「CPU の個性」を，自由に選択・設定してシステムに与えるものです．そしてこの入力条件に対して，チップ・サイズ，クロック・スピード，タイミングのマージン，電源電圧などの条件の規定を対話的に確認して，最終的に OK となると，CPU コアのマスク・パターンとシミュレーション・モデルを，相当に**最適化**されたものとして**自動生成**してくれる，というようなものなのです．

ここでは，ASIC のミクロン・ルールの進展とは別の話として，メーカ製の(人間の手作業まで入る)高密度な CPU コアほどではないにしても，十分に最適化された圧縮が必須条件となります．つまり，オリジナル CPU そのものだけで ASIC のチップ上が占領されるのではなく，あくまでランダムロジックと混在するための一種のメガセルでなければなりません．そして，かなり虫の良い話ですが，これを「フルカスタム LSI」としてマニュアル配線するのではなくて，開発期間・開発費用の負担の軽い ASIC(セミカスタム LSI)として，基本的には自動設計の環境で実現してほしいのです．

こんな話はまったくの無理難題，夢物語でしかないでしょうか．現在のところは CPU そのものの基本特許の問題もありますから，たしかに困難かもしれません．半導体ビジネスの基本(メーカ側の設計による大量生産)にまるで反する性格のものですから，大手メーカほど感覚的に抵抗があるでしょう．しかし，CPU が特許の聖域であるのは，ある時期までの話です．そして，柔軟性のある ASIC 設計環境をセールス・ポイントにした一部の LSI メーカでは，確かに着々とこの世界へのアプローチを進めています．おそらく 21 世紀に入るかどうかの頃の早い時期に，ここで述べたようなシステム設計技術がかなり現実のものになっているでしょう．そしてそのとき，この技術を提供する LSI メーカ(と LSI デザイン・ハウス)は，現在の大手半導体メーカとはかぎりません．コンピュータ技術の進展の面白いところなのですが，優れた技術と自由で柔軟な発想，そして果敢なアプローチが「大きな組織の優位性」をひっくり返す，という可能性は，エンジニアの世界のあちこちにあるのです．



＜8＞

並列処理・分散処理とネットワーク

ネットワーク化による システム性能向上

ここでは，単独のマイコン・システムの発展形として，複数のシステムが共同して一つの仕事を実現する「分散処理」と，その技術的ポイントの一つである「並列処理」について，また，このために関係してくる「ネットワーク技術」について考えていきます．

●集中処理と分散処理

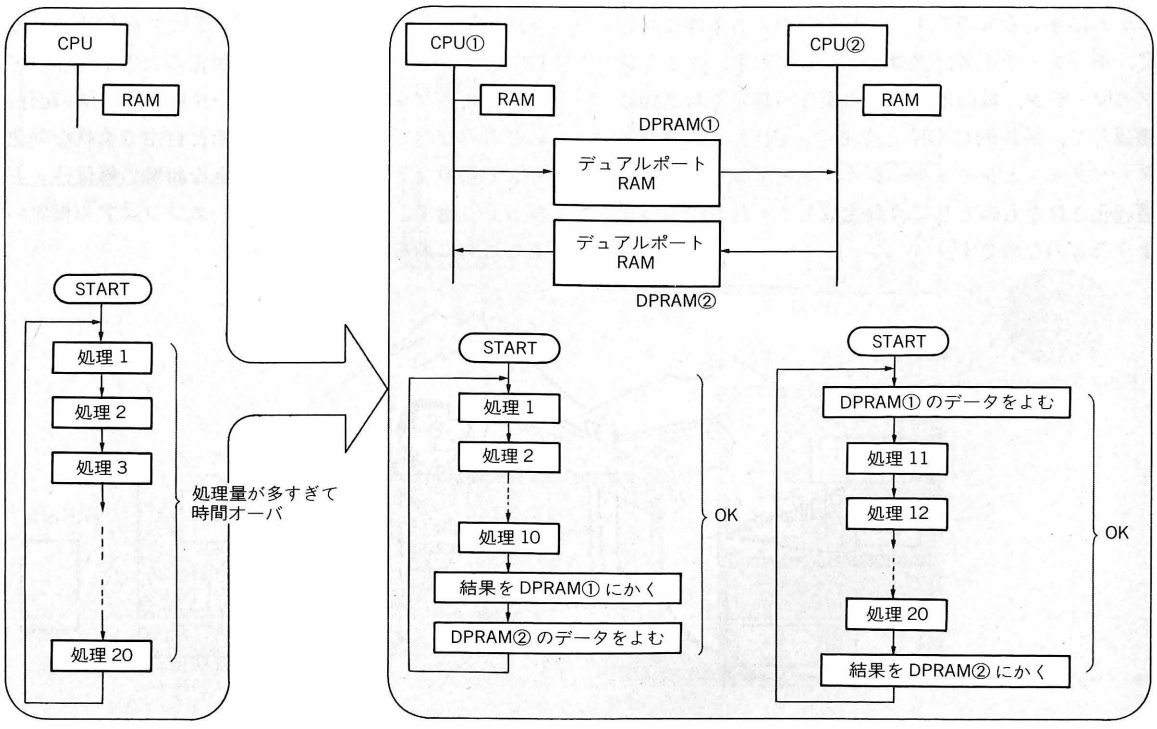
ここで考える「分散処理」の例としては，

- CPU 1個では厳しい処理を何個かの CPU で行うというチップ単位の分散処理
- 1枚のボードでは厳しい処理を数枚のボードで実現するボード単位の分散処理
- 1台のパソコンで無理な仕事を複数のパソコンをネットワーク化して行う分散処理

などがあります．

あらかじめ，これら分散処理のポイントを指摘しておくと，あえて「分散」させた各ブロックないしモジュールを，全体としてどう相互に結合させて処理を実現させていくか，というシステム技術が重要です．個々のブロックは，本書ですでに述べたようないろいろな

〔図8.1〕デュアルポート RAM を使ったモジュール分割の例



マイコン関連技術によって構成されていきますが，さらに分散処理システムでは，それぞれの有機的な結合(情報交換)によって動作していきます．各ブロックの動作は一種の同期がとれていなければなりませんし，情報交換のための具体的なハードウェア面での道具立て，あるいは交換される情報そのものの規定も設計しなければなりません．この手間は集中処理システムでは不要なものですから，最終的には相当のパフォーマンスが上がるように，いかに全体を統括していくか，という，ワン・ステップ高度なマイコン・システム技術が必要になってくるわけです．

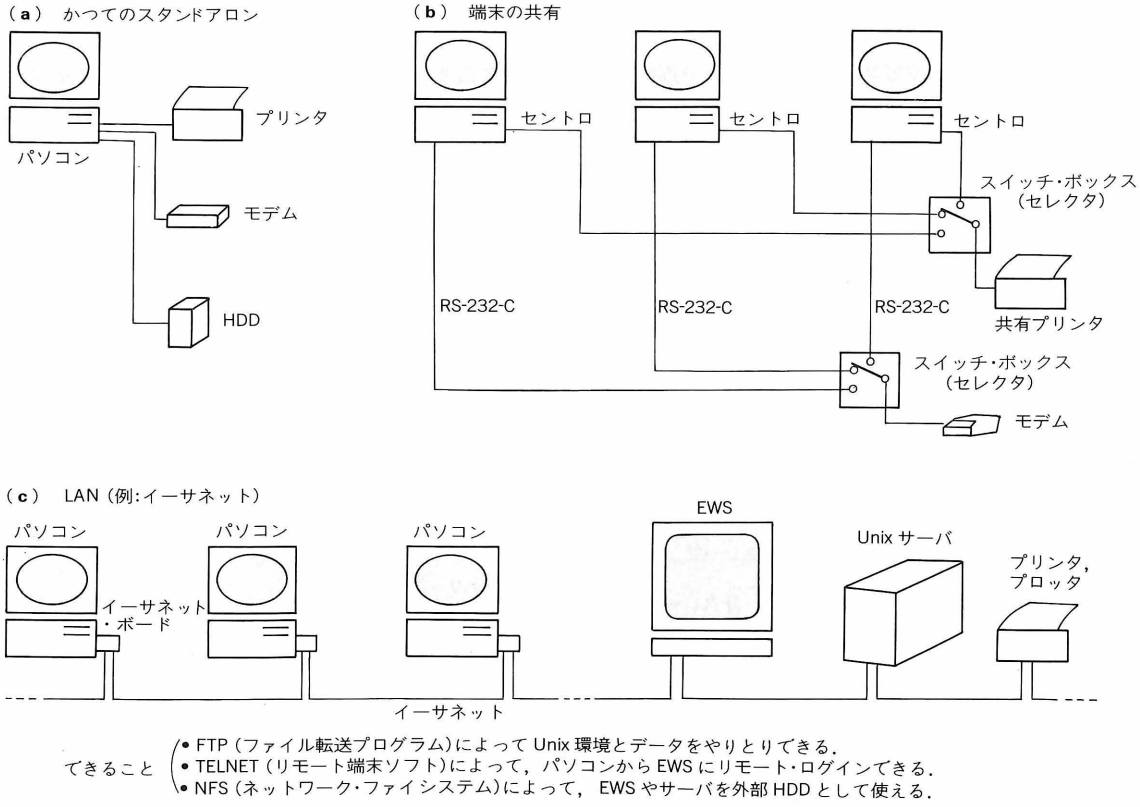
たとえばCPUを2個もった，もっとも簡単な分散処理システムの場合には，**図8.1**のように，デュアルポートRAMによって両方のCPUが情報交換を行う，という定石テクニックがあります．これは決して安いチップではないのですが，たんなるハンドシェイクのように通信部分を組んだのでは，その処理に時間がかかって，全体としてあまり処理を分散させるメリットが得られないために，デュアルポートRAMを使うわけです．

●パソコンによる分散処理：ネットワーク・システム

情報通信・ネットワーク化の時代を迎えて，マイコン・システムはスタンドアロンの機器組み込み用途ばかりでなく，複数の装置が有機的に結びついたネットワーク・システムを構成するようになってきています．この背景にはもちろん，各種の情報通信手段の進展と，CPUの進歩によるマイコン・システムの性能向上があります．ここでは，まずパソコン・ベースの技術から考えてみましょう．

これまでのパソコンは，「御本尊」としてシステムの中央に鎮座している姿が一般的でした．プリンタ，モデム，HDなどの各種装置はすべて「周辺機器」であり，パソコンごとのものでした．ところがやがて，セントロニクスやRS-232-Cなどのインターフェース規格を活用して，スイッチ・ボックス(セレクタ・ボックス)を使った端末の共有(プリンタ共有，モデム共有)の

〔図8.2〕パソコンによる分散処理



段階になりました(図8.2). 最近ではさらに，ウインドウ・ソフト環境を活用したパソコン専用LANの構築とか，さらにはイーサネットによるEWS環境とのリンクへと進んで，もはやパソコンは「システムの端末装置の一つ」へと変身しています．ネットワーク上にあるサーバによってファイルを共有(データ共有)することで，各自のマシンに同じようなデータが重複して散在している非効率が改善され，さらにUnix環境では仮想端末機能によって，システム全体の「マシン・パワー共有」へと進化しています(図8.2(c))．もし，すでにUnixネットワーク環境が稼働していれば，そこにパソコン(もっと安いXウィンドウ端末でもOK)を接続することで，すぐにEWSのパワーと資源を活用した高度な情報処理を行えるのです．

●ネットワーク・システム設計のポイント

パソコンLANのためのインターフェースを活用する場合，物理的な信号レベルは汎用ボードに任せてしまうとすれば(もちろん専用LSIでボードを組むとかなりの勉強になる)，その上の「プロトコル」の設計と，システム全体の「仕事量の分散のさせ方」がシステム開発のポイントとなります．たとえば，リアルタイム性を重視した制御目的のシステムであれば，汎用の通信プロトコルのサービス機能はやや冗長で遅いので，自分なりに高速でコンパクトな規約をオリジナル設計する，というようなアプローチになるわけです．また，ネットワークの形態(図8.3)についても，必要な動作に応じて設計できます．

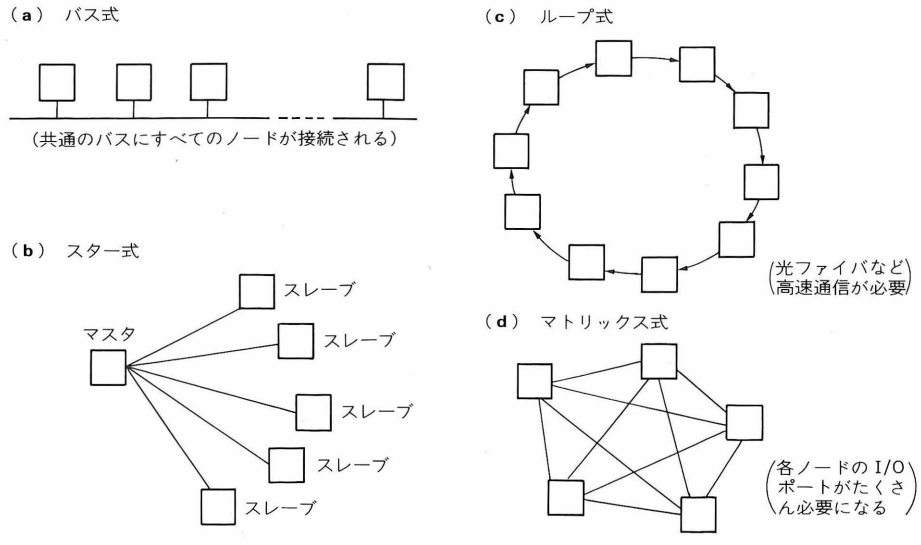
システム設計・プロトコル設計の際には，まずは「ネットワーク・システムが立ち上がるまで」の動作が重要です．単一のシステムのリセット処理と違って，複数のシステムがそれぞれ別個にリセットから立ち上がった後に，マスタとスレーブをそれぞれ認識して，ネットワークとしての初期設定の項目，たとえば

- 自分のID
- システム中での自分の位置付け(マスタかスレーブか)
- 他にどれだけの機器がアクティブなのか
- ネットワークへの参加方法(他のメンバとの通信方法)
- システムからの離脱方法の取得

などを完了しなければなりません．実際にネットワークが稼働状態にあるときよりも，この立ち上げ時の動作追跡のほうが難しいかもしれません．この点では，共有のサーバに対して勝手に個々のクライアントがアクセスするという，「**クライアント-サーバ方式**」のほうがまだ簡単なシステムですから，ここから挑戦してみるのがいいかもしれません．

まずはネットワーク環境を「使う」立場で十分に活用して，つぎにはシステム管理者としてネットワーク

〔図8.3〕ネットワークの形態のいろいろ



環境を「改良する」立場でさらに理解を進めて，やがてはネットワーク環境を自在に「構築する」という境地を目指してみたいものです．この技術こそ，21世紀のエンジニアの大きな活躍の舞台となることでしょう．

ボード・マイコンによる分散処理の例

●ボード・マイコンによる分散処理

パソコンによる分散処理システムほど大袈裟でなく，信頼性(誤動作対策やトラブル・リカバー処理の作り込み)の面で安心できて，もっと小回りがきいたシステムを構築したいとなったら，ボード・マイコン(またはオリジナル・ボード)を組み合わせた分散処理システムを検討してみましょう．

ボード・マイコンで分散処理システムを構築する目的は，開発効率の向上とコスト低減にあります．そのためには，直列に連なっている処理を分割していくつものボードに別プログラムを開発するよりも，共通のソフトウェア開発環境で共通のプログラムROMを用意する，というような作戦が有効です．つまり，並列な処理を分担するような分割を考えることになります．ここでは，「基板上のDIPスイッチの状態をCPUが自分で検出してIDを認識する」という定石テクニックがあります．

この分散処理・並列処理システムの設計指針は，ソフトウェアの構造化プログラミング技術と本質的に通じるものです．膨大な処理を階層的にモジュール(LANであれば各ノード)に分割するわけですが，ポイントとしては**「モジュール間を疎結合に」**ということになります．つまり，**分割された各処理単位の間でやりとりされる情報量がなるべく少なくなるように，**いいかえると各処理単位がなるべく完結した処理を担当するようにします．これはソフトでもLANでも明らかなことですが，転送される情報が多ければ，かんじんの処理に比べて転送処理の時間が増えてしまうからです．

●ボード・マイコンによる計測ネットワークの例

それでは，具体的な例で，この

- システム全体の仕事量を検討する
- それを分割してどのように情報転送させるか

という部分について考えてみます．ここでは，建物の構造解析のために，ビル内の100ヵ所(相互に数mずつ離れている)に振動センサを取り付けて，このデータを収集するとします(**図8.4**)．具体的な条件と検討例を，**図8.5**のように列記してみましょう．

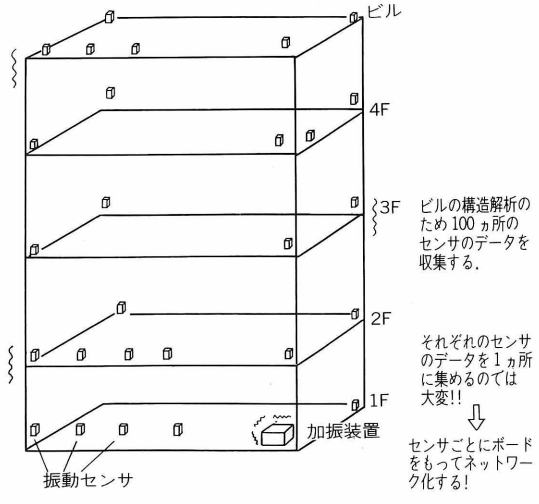
各センサ部分には，アナログ信号処理のための回路(センサ出力をある程度の電圧または電流に変換)があります．電圧データとして数10mの間隔をあけるのは，電圧降下やノイズの影響など，信頼性と精度の点で問題がありますから，電流ループ方式，あるいはV-Fコンバータによってデータを2値のディジタル信号とする方法も有効です．

データは8ビット精度として，各ポイントを10msごとに，つまり1秒間に100回ずつ計測したいとします．1回の計測は加震装置による1分間とすると，各ポイントのデータ量は6,000個，全体のデータは1回の計測ごとに60万個となります．各ポイントのデータ計測は，つぎの段階の解析(たとえばEWS上シミュレーション)処理のためにも，時間的に正確に同期していなければなりません．

●「集中処理」では困難

さて，このような仕事を1台の中央コンピュータで行えるでしょうか．まず，各ポイントのセンサ出力を順次スキャンして，A-D変換する場合のスピードを考えてみます．計測は1ポイントが10ms間隔でこれが100ポイントですから，すべてを1ヵ所で実行するた

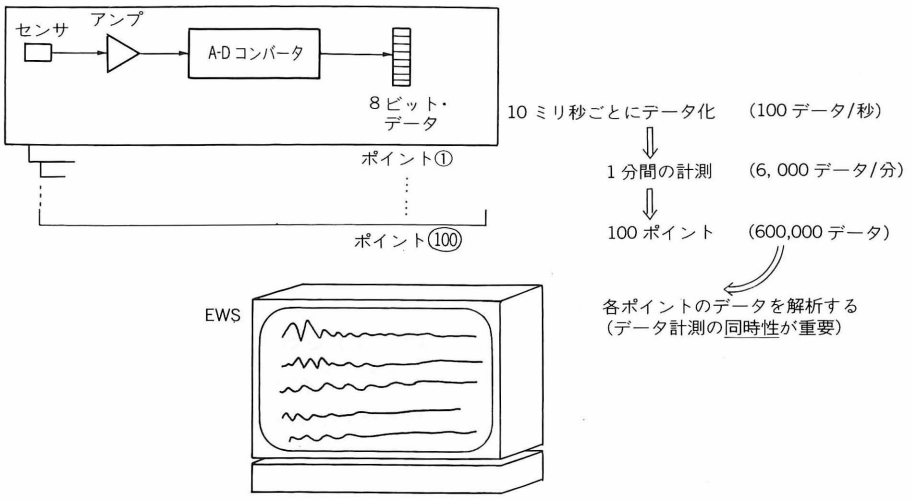
〔図8.4〕ビル内振動計測システムの例



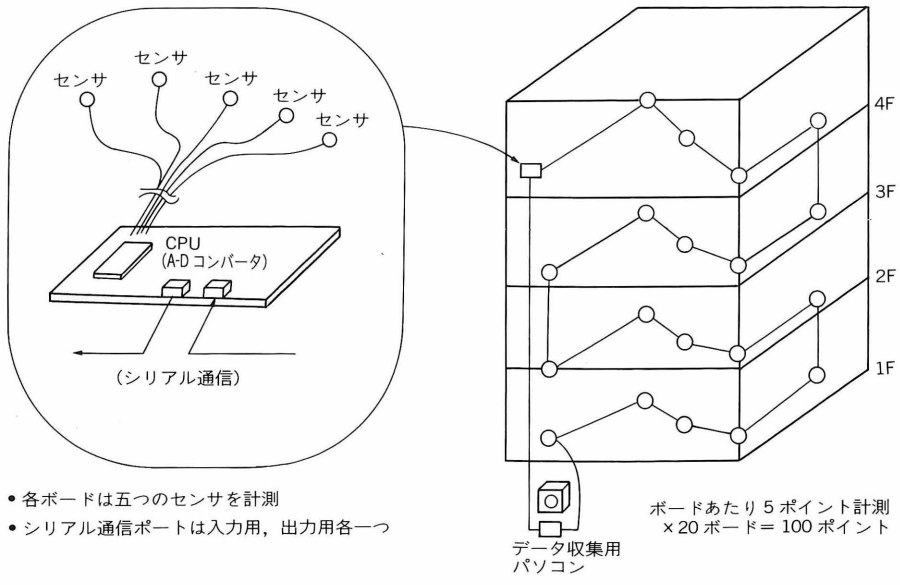
めには，各ポイントのデータ収集時間は100 μsということになります．やや高速のA-Dコンバータなら簡単にクリアできる変換時間ですが，これをパソコンのソフトとして実行させるには，A-D変換については，他の仕事をいっさいストップして必死に実行するような高速プログラムを，おそらくアセンブラで書かないと厳しいかもしれません．DOSのタイマ割り込みとかを強制的に禁止しておかないと，データ計測時間がばらつく恐れもありそうです．

また，アナログの入力ラインを切り替えるアナログ・スイッチが100回路の選択のために多段となり，その遅延を考慮したサンプル・ホールド回路を用意しなければなりません．電圧データや電流データでは，多段アナログ・スイッチのオン抵抗の影響や，長い信号ラインのコンデンサ成分の影響(データ信号が積分されやすくなる)も心配です．そして，それ以上に問題

〔図8.5〕データ計測の条件



〔図8.6〕ボード・マイコンによる実現例



- 各ボードは五つのセンサを計測
- シリアル通信ポートは入力用，出力用各一つ

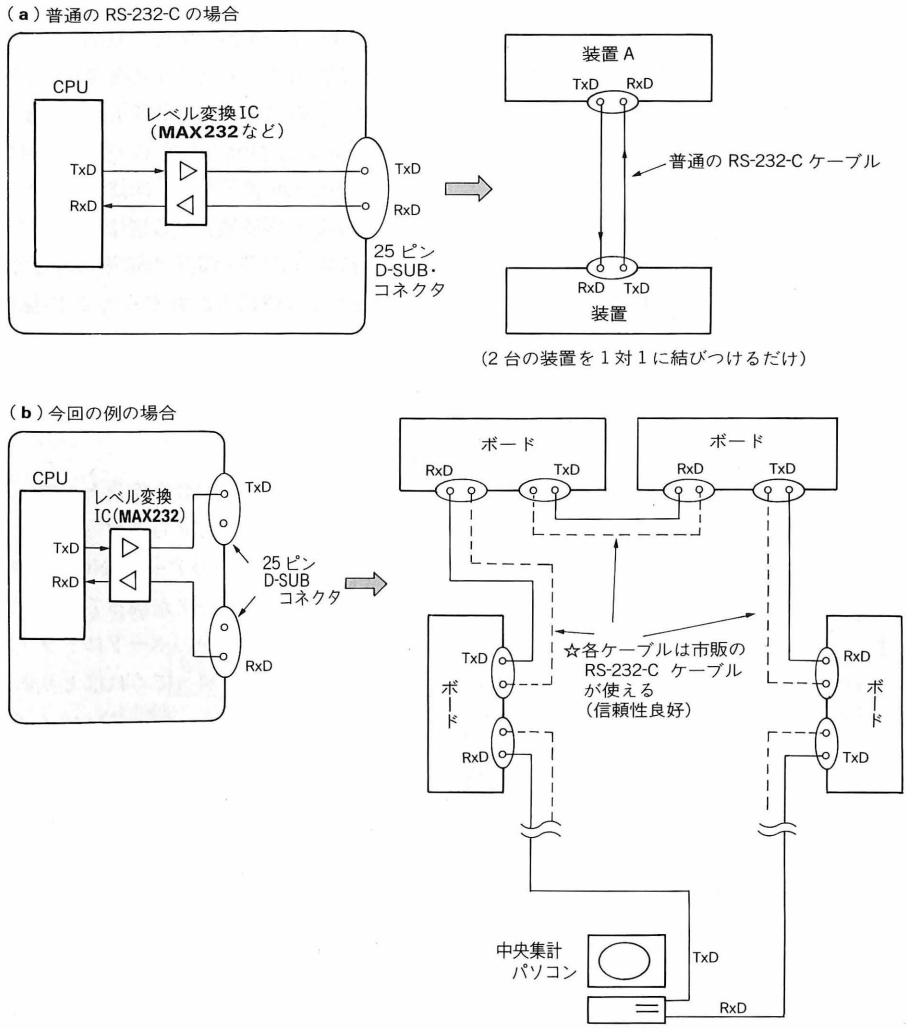
となるのは，全体の配線量の点です．中央のコンピュータから100本の入力信号ラインの配線がわさわさと出ている姿は，ちょっと想像したくありません．

●ボード・マイコンによる構成例

そこで，ボード・マイコンによって計測するという，分散処理システムが登場します(**図8.6**)．この場合，CPUとして1チップ・マイコンを使って，これに内蔵されているA-Dコンバータとシリアル通信ポートを活用するのが最適でしょう．各ボード上でセンサからの電圧データをA-D変換してディジタルのデータにして，さらにRS-232-Cなどのシリアル通信信号として相互に接続することで，システム全体の接続が画期的にシンプルになります．筆者であれば，ここで中央の集計装置であるパソコンから「スター状」のネットワークとするのではなくて，シリアル通信を使った「リング状」のLANとしてみたいところです．

1チップ・マイコンのA-Dコンバータは，5チャネルとか8チャネルとかの入力をもっています(自動車とかファクシミリなどのセンサは1台で何個か必要なのが普通なため)．そこで，たとえば1枚のボードで5ポイントのデータを収集して，全体として20枚のボードを用意すればよいことになります．この場合，各ボードは基板のパターンから部品，さらにはEPROM内

〔図8.7〕シリアル通信ポート部分



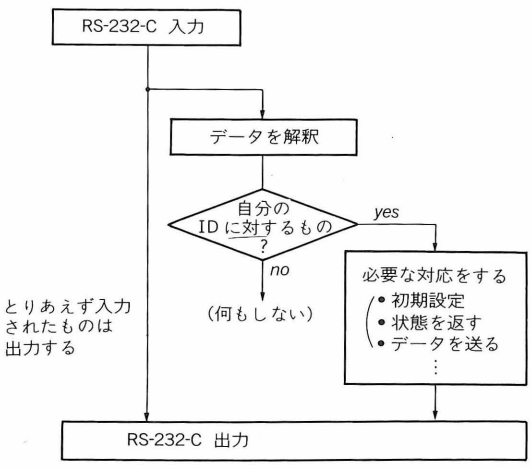
のプログラムまで，完全に同じものを作るようにします．各ボードのCPUはリセットされて立ち上がると，まず基板上のDIPスイッチ状態を調べて自分の端末番号を認識して，中央集計装置からの呼び掛けに対しては，自分宛てのものだけに対応することになります．

●ネットワーク構成のアイデア

情報交換のLANの部分については，普通のRS-232-C(**図8.7**(a))とはちょっと違った，ユニークな方法をとってみます．ケーブルのコネクタには，普通の25ピンのD-SUBコネクタを使うと，ケーブルとしては市販のものが活用できます(特別製のケーブルを何10本も自作する必要がない)．信号レベルについても，普通のRS-232-C用の変換ICを使います．ところが**図8.7**(b)のように，このコネクタをボードごとに2個用意して，送信ラインと受信ラインとを，別々のコネクタに接続するようにします．つまりCPUとしては1チャネルのシリアル通信ですが，2台の装置を接続して双方向に通信するのでなく，入力専用コネクタと出力専用コネクタをもつのです．そして，中央のパソコンの部分だけは，二つの通信ポートを使うか，あるいはコネクタ部分だけを外部で変更(拡張)して，パソコンからの出力が最初のボードの入力に，その出力がつぎのボードの入力に…とつぎつぎに接続して，最後のボードの出力をパソコンの入力に戻します．これで，リング状LANが完成します．

各CPUは自分の担当ポイントのセンサ出力をA-D変換してデータ化します．各ボードが厳密に「同時に」計測するためには，さらにシリアル信号ラインとは別に，1本だけ「スタート伝達ライン」を用意することになります．各ボードはこのライン(パソコンからスタートを送る)の変化をトリガとして，それぞれ自分のCPU内のタイマによって時間を管理して，データを計測します．水晶発振(クォーツ)ですから，パソコンのタイマと同じ正確さでしょう．

〔図8.8〕各ボードのシリアル通信処理のながれ



●それぞれのボード上での並列処理

こうなれば，あとは各ボード上のCPUソフトと中央のパソコンとの間で規定される，シリアル通信上の情報交換によって，全体の処理が決定されます．この例の場合，「データ計測」と「データ収集」，さらに「データ解析」の段階がきれいに分かれていますから，このプロトコル設計は容易です．ここで重要なのは，各ボードのシリアル通信処理の原則として，「入力情報はそのまま出力する」ということです．これによって，パソコンから出た情報は必ずLANを1周してすべてのボードに伝わり，さらにパソコンに戻ることで「LAN接続OK」をつねに確認できます．各ボードでは，こうして「通過」する情報を同時にチェックしていて，自分のIDを指定した情報に対しては，さらにその返答となる情報も出力するのです(**図8.8**)．情報にIDがつねに加わっていれば，パソコンはLAN中の任意のボードと「相互通信」(実際にはパソコンからの呼びかけに対するボードの返答)することが可能となります．

まずシステムの初期設定作業としては，それぞれのボードがリセットから立ち上がった頃を見はからって，パソコンからダミーのデータ(NOPコマンドでよい)を発します．このデータが戻ってこなければ接続が未完成，あるいはどこかのボードにトラブルがあるのです．発したデータが戻ってくればとりあえずLANが実現されていますから，つぎにはパソコンから各ボードに対して，順に「状態はいかが？」という問い合わせのコマンドを，それぞれのID番号を添えて送ります．各ボードはこの情報をバケツ・リレーして1周させ，さらに該当するボードからは「待機OK」の返答データが，確認のため自分のID番号を添えて返されます．これをすべてのボードに対して確認すると，システムの設定は完了です．

さて，「データ計測」の段階ですが，まずはパソコン

から「これから計測を開始する」というコマンドを送ると，すぐに1周して戻ってきます．各ボードはこのコマンドに対して，「スタート伝達ライン」の状態を監視するモードに入ります．そして計測環境(ここでは加震装置)と同期して，パソコンからのスタートが送られると，各ボードは1分間のデータ計測を行います．データはボード上のRAMに置かれます．1分と数秒たったところで，確認のためにパソコンは再度，各ボードの「状態OK」を調べます．ここでは各ボードが，「計測が正常に完了した」という返答を送るといいでしょう．

1回の計測ごとに，あるいは何回かをまとめてから，システムは「データ収集」の段階に移ります．ここでは，パソコンから「ID番号が××のボードは，××ポイントの計測データを順に送りなさい」というコマンドを送ればよいことになります．LANはつねに「素通し」の状態ですから，該当するボードからのデータがパソコンにつぎつぎと送られて，パソコン内のメモリ，あるいはファイルとして収集されます．言葉で書くと大変な作業のようですが，たとえば9600ボーの転送速度であれば，100ポイント1分間の全データをパソコンに回収するのに，計算上ではおよそ10分ほどかかることになります．この転送時間が長すぎるのか，あるいは計測装置の次回セッティングのバックグラウンドとして許容できるか，という判断は個々のケースによることになると思います．

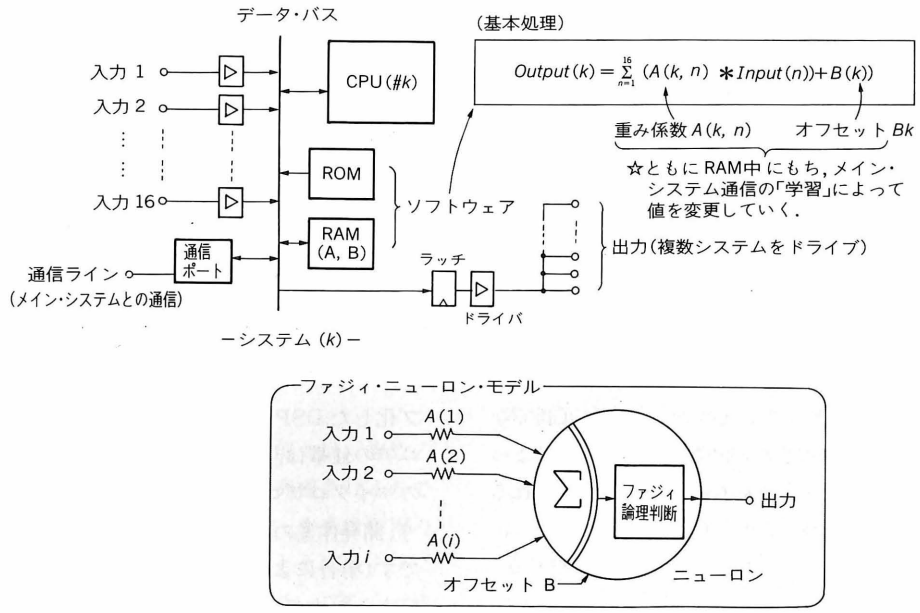
●分散・並列処理の例：ニューラル・ネットワーク

ボード・マイコンを活用した分散処理システムとしては，もっと挑戦的なテーマがあります(これはもしかすると，本書を読んだフレッシュマンの皆さんが，いきなり最先端の舞台に登場するかもしれない可能性です)．

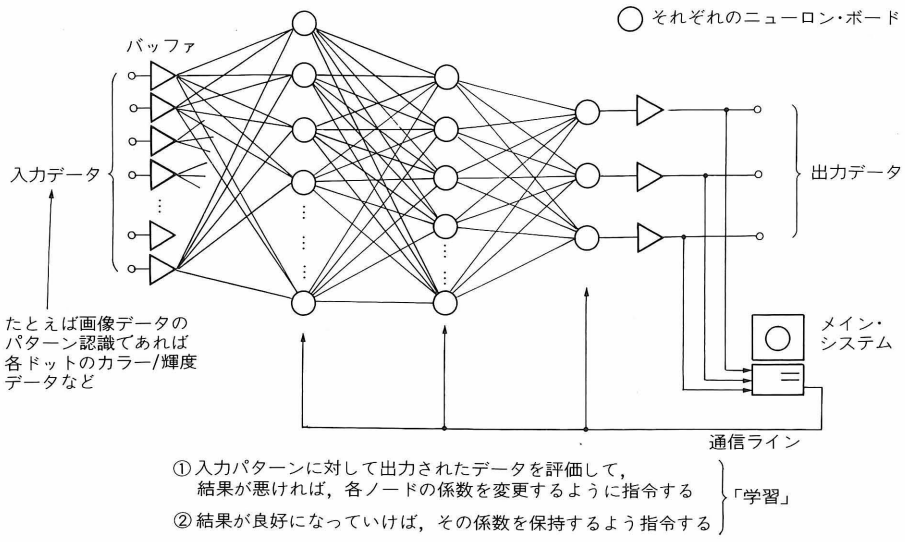
まず，各ボードのCPUソフトとして，**図8.9**のように，ごく簡単な自律システム(複数チャネルからの情報入力，ファジィ論理・判断，1チャネル・複数負荷への出力)を設定します．各ボードは簡単な「知識」を記憶するための係数データ・メモリをもち，自分の判断の結果が「良かったか悪かったか」(メイン・システムから通信ライン経由で伝えられる)を「学習」して，この知識を成長(問題のある判断を修正)させます．また，各ボードは「システム全体の中での自分の位置」を理解する必要はありません．自分に情報を与えてくれるいくつかの隣接システムと，自分の情報を受容するいくつかの隣接システムしか「見えない」のです．

そして，これらの自律的なボード・マイコンを，100

〔図8.9〕「ニューロン・ボード」の基本動作



〔図8.10〕「ニューロン・ボード」の相互接続の例



枚，できれば1,000枚のオーダで相互に結びつけます(**図8.10**)．具体的には，隣接するいくつかのボードの出力から情報入力して，結果の出力情報を隣接するいくつかのボードに供給するのです．それぞれの情報通信データとしては，アナログにしてもディジタルにしても，オン/オフの2値データでなく，一定範囲の幅をもったデータを伝送しなければなりません．

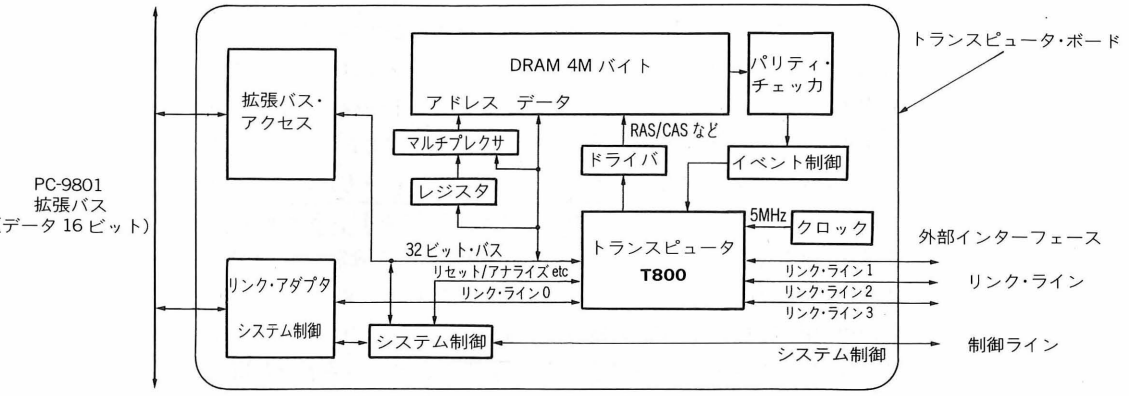
もうおわかりだと思いますが，このシステムというのは，「ハードウェアによるニューラル・ネットワーク」そのものなのです．現在のニューラル・ネットの研究というのは，EWS上でのソフトウェア・シミュレーションとか，パソコン用の専用アクセラレータ・ボード(DSPによる高速演算エンジン)によるものがほとんどです．いずれも非常に高価で，なかなか実験室で片手間にできるものではありません．そこで，いきなり100枚(1枚1万円のボードを200枚使うと，EWSやDSPボードとちょうど同じコスト帯になるのが面白い)の単位でなくても，雛型となるCPUソフトをいろいろ実験してみてはどうでしょうか．本当に冗談でなく，ボードに仕込まれたソフトウェア・モデルによっては，画期的なニューラル・ネットの挙動が得られるかもしれないのです．

チップによる並列・分散処理の例

●チップによる並列処理：トランスピュータ

映像・音声などのディジタル信号処理システムでは，多量のデータ演算処理を高速に実行する中心的プロセッサとして，汎用のDSPチップが活躍しています．かつてのDSPとは，たとえば24ビット同士の乗算と48ビットの加算・累算を数十ナノ秒で実行する，といった「演算専用チップ」でした．このためシステム全体の設計としては，DSPの周囲に高速のメモリやロジックを置いたボードを設計して，ちょうど「ビットスライスCPU」システムの場合のような，マイクロプログラム・メモリにしたがった動作を実現してやる必要がありました．ところが最近では，この周辺機能(ワーク・メモリ，データ・ラッチ，プログラム・シーケンサ，マイクロプログラム・メモリなど)をすべて1チップ化したDSPがいろいろと登場しています．エンジニアの仕事は，システムの具体的動作を決定するためのマイクロプログラムを設計してロードすることになり，開発作業の主舞台がソフトウェアに移行しているのです(場合によってはチップ内部のハード的構成を設定するための，LCAのコンフィギュレーション・デー

〔図8.11〕PC-9801用トランスピュータ・ボードの例(東洋通信機製)



タに相当した設計も行う).

このいずれの場合でも，これまでのDSPというのは，基本的に1チップで仕事をする孤独なLSIでした．ところで，これまでに述べてきたような「分散処理・並列処理によるシステム性能向上」という発想は，このDSPの世界にも適用できます．この，複数個のDSPチップの分散処理という視点から，二つの例を考えてみましょう．

まず，SGSトムソン・マイクロエレクトロニクス社インモス事業部の「トランスピュータ」というLSIについてです．このチップは，内部に強力な演算機能をもつ単体の高速CPUなのですが，トランスピュータ自身の動作を規定するプログラムを外部からロードしてやることで，一種のDSPとしてマイコン・システムの1ブロックに活用することができます．たとえば，パソコン拡張スロットへの挿入ボードとして，数値演算能力を飛躍的にアップさせる「エンジン基板」(**図8.11**)とか，あるいは「ファジィ演算ボード」などが実現されています．あらかじめコンパイラで開発したプログラムを，あたかもマイクロプログラムのようにボード上のトランスピュータに転送しておけば，あとはこのボードに入力データを与えると，すぐに演算結果データを返してくれます(パソコンのソフトで処理すると非常に時間がかかる).

しかし，トランスピュータの本当の能力は，複数のトランスピュータを使ったシステムで発揮されるのです．トランスピュータは，お互いのデータ交換専用の高速の通信ポートを装備していて，たとえば2チップを使用することによってほぼ2倍，4チップならほぼ4倍のデータ処理が，かなり容易に実現できるようになっています．これは，普通のDSPを2個とか3個並べようとしても，システムの基本的なところがまったく変更されてしまったり，共存できても全体の処理量がチップの個数に比例するどころか，データ交換の手間のためにあまり向上が期待できないのと対照的です．

トランスピュータ搭載の基本モジュールとして，マザーボードにどんどん追加して増設できるような超小型ボードが出ているのも，この特徴を生かそうという発想があるからです．データ通信ラインが複数系統あるために，相互に結合して分散処理・並列処理を実現する形態としても，ツリー状に結合させたり，スター状，ネットワーク状など任意の構成がとれます．すぐに思いつくことですが，前に述べたニューラル・ネットワークを実現するためにも，この機能は大きく役立とうとしています．

並列処理の記述のためには，トランスピュータ専用コンパイラ言語(Occam)によってプログラム開発を行いますが，これは通常の処理のためのCコンパイラによるプログラムと親和性がよく，容易にリンクできるようになっています．演算対象のモデリングとシステムのチューニングによりますが，32ビット・パソコンとトランスピュータを何十個か組み合わせたシステムでも，EWSやミニコンどころか，スーパ・コンピュータのレベルのベクトル演算処理ができる場合もある，といわれています．なんと，コスト・パフォーマンスは10倍オーダというよりは，100倍オーダの向上になる計算です．

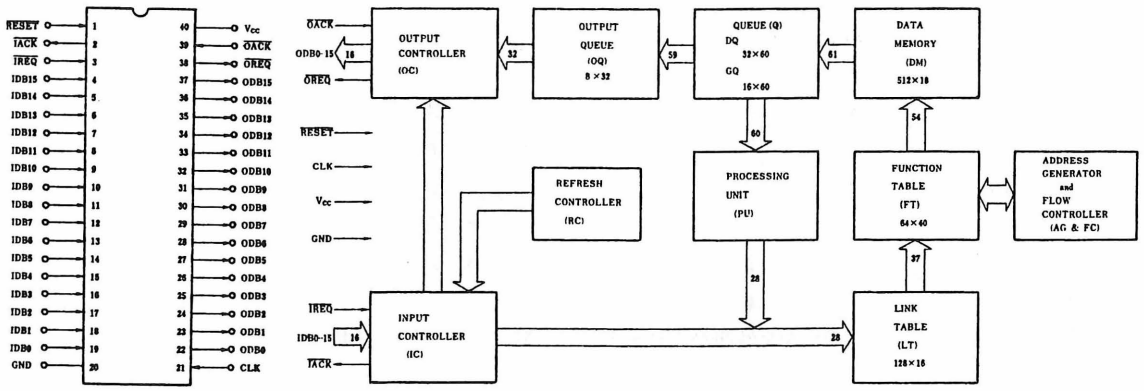
●チップによる並列処理：ImPP

日本電気のImPP(イメージ・パイプラインド・プロセッサ)という特殊な演算LSIがあります(**図8.12**)．これはすでに数年前から出回っているチップで，筆者はずっと興味をもって見守ってきたのですが，いまだに地味ながら頑張っています．じつは「非ノイマン方式」を採用しているのがこのLSIのユニークなところで，この点を，登場した当初から(実際にはチップが出る以前のアナウンスの段階から)注目してきたのです．

ノイマン方式の逐次処理の遅さを克服するためには，プログラムのフェッチのための内部バスと，実際の処理データの内部バスとを分離する，「ハーバード・アーキテクチャ」という方式が古くからあります(**図8.13**)．これは，従来から一部のDSP内部でも採用されており，さらに最近の高性能ハイエンドCPUでは，CISCでもRISCでも部分的にこの考え方を採用しています．ところがImPPでは「データフロー方式」を採用しているところがユニークなのです(最近，ようやくシャープからデータフロー方式による，かなり大がかりなチップ・セットが登場したところで，ずっと昔から1チップ化している日電はすごい！)．

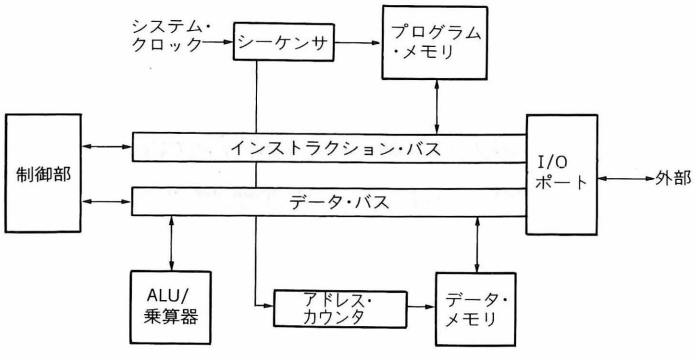
データフロー方式というのは**図8.14**のように，データ・バスを32ビットとか40ビット(シャープは64ビット)に幅広く設定して，1ワードのデータの中に，プログラムもデータも埋め込んでしまう，という方式です．ImPPは入力と出力が完全に同一形式のバス端子(配置もDIPの左右対称位置)となっていて，このデータ・ライン上に何個ものImPPを機械的に並べられるようになっています(**図8.15**)．

〔図8.12〕データフロー・プロセッサImPP(日本電気のデータブックより)



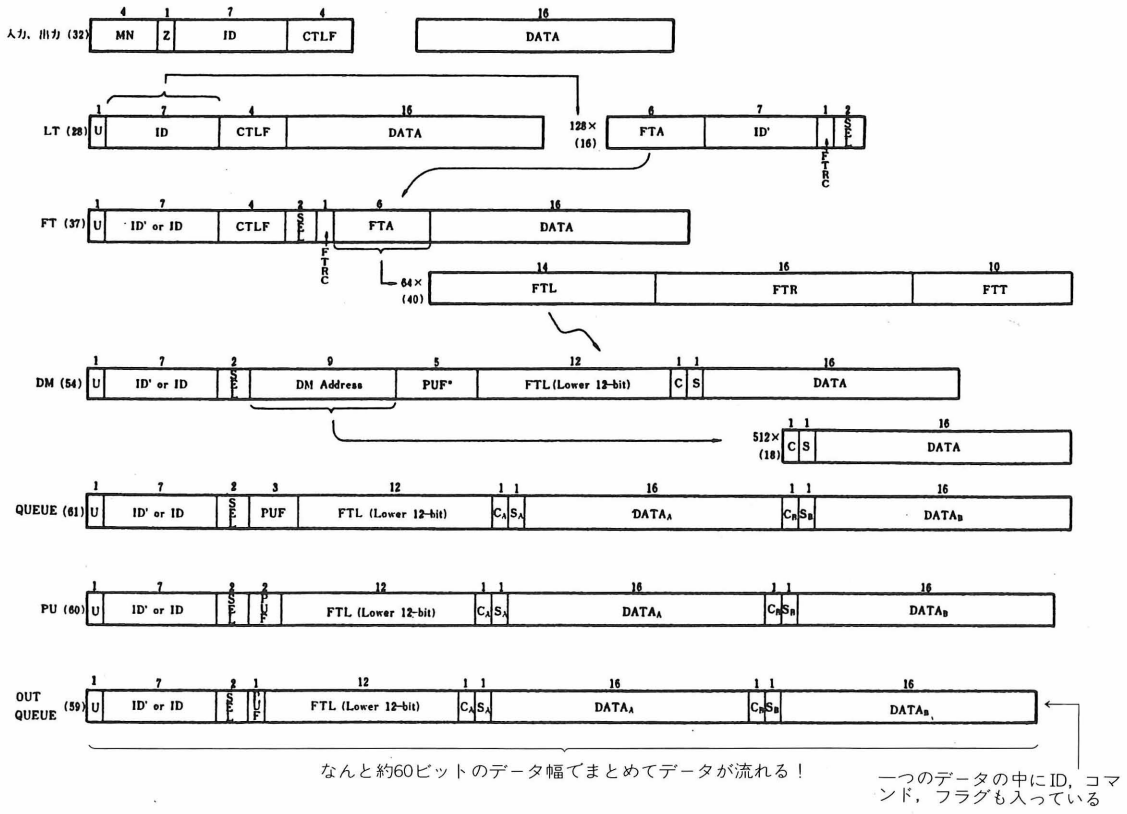
(a) μPD7281ピン配置　　(b) μPD7281ブロック図

〔図8.13〕「ハーバード・アーキテクチャ」の例(DSPチップの例)

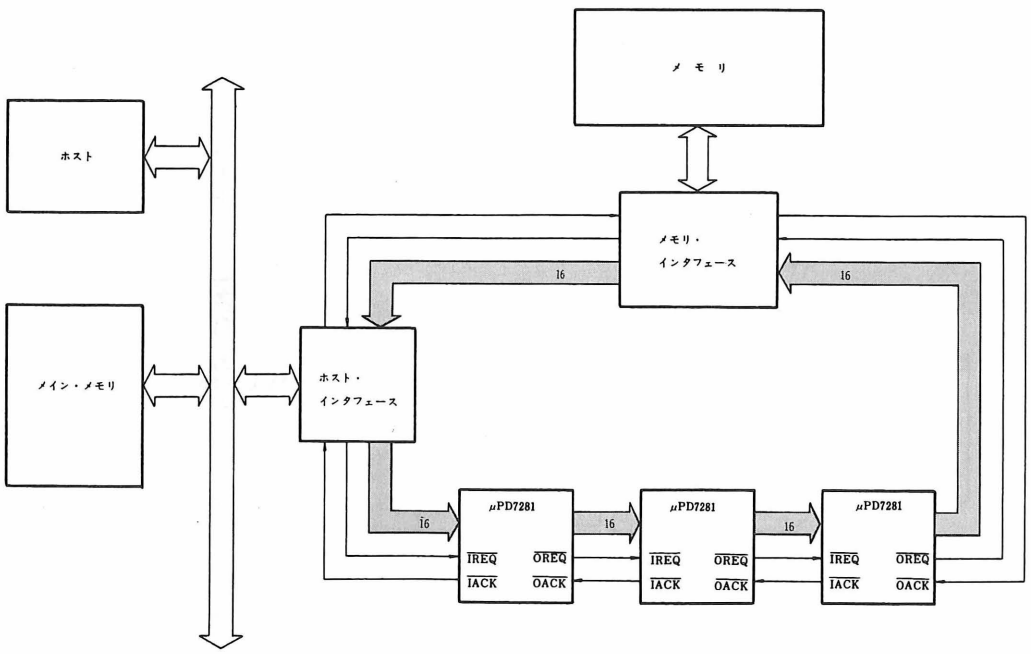


☆インストラクション・バスとデータ・バスが分かれている

〔図8.14〕「データフロー」の例(日本電気データブックより)



〔図8.15〕ImPP の並べ方はかんたん！(日本電気データブックより)



● ImPP の動作

ImPP の動作を簡単に紹介すると，システム・クロックに相当する 1 サイクルごとに，処理されるデータと処理プログラムとのセットが，一気にチップに入力されます．ImPP はプログラム中の ID を自分の ID(リセット時に獲得できるようになっている)と比較して，自分の担当すべき処理だけを実行して，次段へと出力します．データは，自分を処理するためのプログラムをともなって勝手に進んで行き，処理すべきデータの揃った ImPP では演算が実行されます．二つの入力データに対して演算処理する場合，片方のデータが届いた場合，もう一方が揃うまで自動的に「待ち合わせ」をします．ノイマン方式の場合のような「フローチャート」というものはなく，処理されるデータが揃ったところから，全体の順番とは関係なく処理が進行していきます．つまり，本質的に逐次処理であるノイマン方式で「並列処理」を実行させるためには，ソフト的に並列動作させるための努力が必要でしたが，データフロー方式の場合には，処理されるデータの流れ(データフロー)を記述することで，そのまま並列処理が実現されてしまうのです．そして，たとえば ImPP チップを 2 個から 4 個に単純に増やすと，全体の処理能力がほぼ 2 倍となるところは，トランスピュータの特徴とも似ています．

アイデアは素晴らしいのに，残念ながらこの ImPP は目立ったモデル・チェンジもなく，その後も細々と一部で画像処理をしているようです．おそらくユーザである設計者側の発想の転換が難しいのと，他の DSP に比べてそれほど画期的に高性能でもなく，アプリケーションの分野が限定されている(名前のとおり，画像処理のような連続・単調なものでないとパフォーマンスが高くならない)ために，なかなかメジャーとはならないのでしょう．しかし，このアイデアが 21 世紀には実を結ぶのでは，という期待をもたせる LSI です．

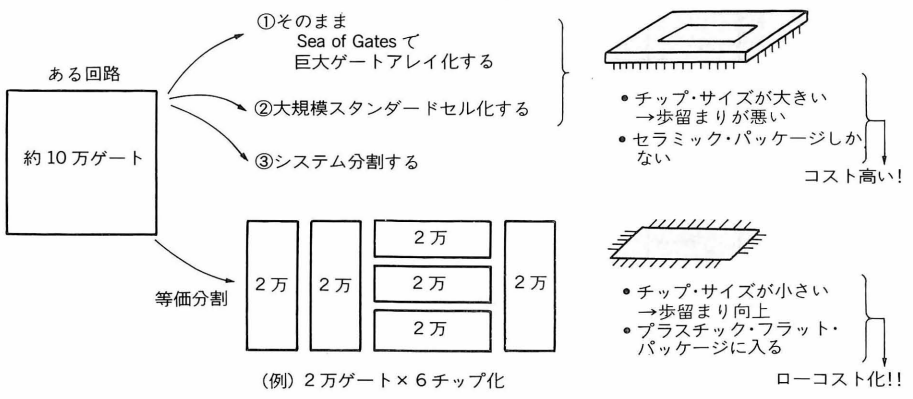
ASIC 化による分散処理

● ASIC における分散処理

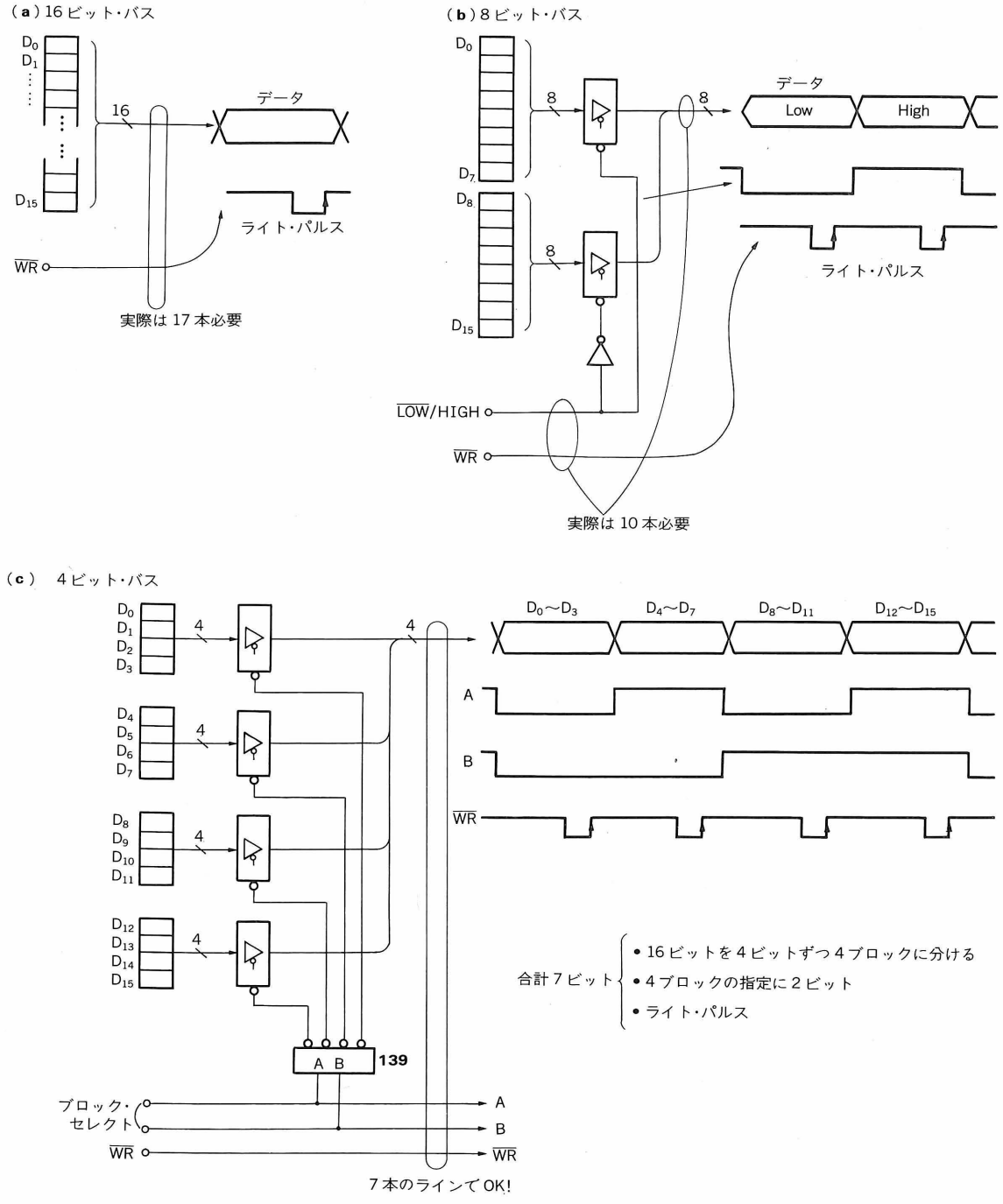
「ある大規模な処理を分割して分散処理する」という発想は，パソコンや CPU ボードばかりでなく，ハードウェアの先端である ASIC システムでも，もちろん適応できるものです．とくに，ASIC の世界では「数量効果」，「最適チップ・サイズ効果」が効いてくるために，分散処理の考え方をどう適用するかが，システム・パフォーマンスとコスト・ダウンの大きなポイントになります(図8.16)．

たとえば，ある処理のための専用 ASIC をラフにシステム設計したところ，約 10 万ゲート規模の回路であったとします．これを 1 チップ化して設計したとすると，ゲートアレイにしてもスタンダードセルにしても，メーカの提供するラインアップのいちばん上位のランクで，一般的に「ゲート単価」は高めになります．これは，チップ・サイズが大きくなって歩留まりが低下するためで，さらにパッケージも安価なプラスチック版に入らずに，高価なセラミック・パッケージ版(あるいはこの中間の，サーディップ・パッケージ版)となって，ますますコスト的に不利になります．具体的な数

〔図8.16〕ASIC のシステム分割は重要！



〔図8.17〕16ビット・データの転送方法のいろいろ



字をあげて考えてみると，開発費用を別にして，ゲート30銭としてチップ単価を3万円としましょう．

● ASICによるシステム分割の例

ところが，もしこのシステムをうまく分割できたとしたら，どうなるでしょうか．たとえば，システム分割にともなうインターフェース回路が増えるので，2万ゲートで6チップ化する(ぴったり5チップは無理)とします．このランクのスタンダードセルはメーカの歩留まりも良好のため，「ゲート単価」がいちばん安いランクとなり，さらにパッケージも，安価なプラスチック版QFPに入れることができます．ゲート10銭とすると各チップが2,000円，この場合の全体のコスト・パフォーマンスは2.5倍の向上，という皮算用になります．

6チップのうち共通のものがあったり，一部のチップを単独で別の用途(社内汎用的)に使用できるように設計すればもちろん，このままでも，LSIメーカを相手としたビジネスとして考えると，チップ全体の数量は増えます．この数量効果によって，さらに部品としてのコストは低下する方向に働きます．つまり，分割されたチップ間の情報転送という設計ポイントさえクリアできれば，全体としては非常に有効な作戦となるのです．

● ASICの分散処理における情報転送

ASICの場合には，分散処理・並列処理・ネットワーク化のためのインターフェース部分について，ソフト面だけでなくハードウェアそのものが与えられるだけでなく，最適に設計できます．たとえば，16ビット単位のデータ転送が必要だとしても，パッケージのピン数を食われたくない場合には，ソフトとハードの両面からうまく設計して，8ビットの転送バスとすることが可能です．この場合，データ・バスとして4ビットを4回に分けてブロック転送するとすれば，何ブロック目かの識別に2ビット，あと1ビットをマスタからの書き込みパルスとすると，残った1ビットによって，スレーブからのアクノリッジを返送する，効率的なハンドシェイク動作とすることもできます(**図8.17**)．

また，交換されるデータ量がかなり多い場合とか，待ち合わせをしないで高速のデータ転送を必要とする場合には，ASICのコンパイルド・セルとして用意されているデュアルポートRAMやトリプル・ポートRAM，といったメガセルを活用します．コストとの関係となりますが，ピン数の多いパッケージを採用できれば，ある意味ではCPUボード同士の結線よりも多くの信号ラインが使えるようになる，というのもASICの利点なのです．

このように，分散処理システムの考え方というのは，システムの規模が大きくなってきたときの解決策としては，なかなか強力なものです．さらに，いったん完成したシステムの処理量があとから増大してきた場合に，それぞれのノードの処理がほぼ同じもの(並列処理)であれば，分散処理のノード数を増設することで吸収してしまう，というような解決方法にもつながるものです．集中処理システムの限界がいろいろと出現し，一方でネットワーク化がますます進歩してきた流れを考えると，コンパクトなシステムを中心に開発するマイコン技術者にとっても，アンテナをはってつねにフォローしていきたい技術だと思います．

[Appendix]

体験的 ASIC 開発入門

これからのマイコン・システム技術者は，各種のツール(開発支援環境)を駆使して仕事をしていくことになる，という意見の参考として，あまり詳しく状況を知らないフレッシュマン技術者のために，先端のシステム開発環境がすでに稼働している「ASIC の開発」の様子を紹介しましょう．

ここでは，LSI メーカの実際の開発環境から代表的なケースを想定して，エンジニアの現場の様子を「限りなくノンフィクションに近いフィクション」として，お伝えしてみたいと思います．その仕事の大変さにびびってしまうか，それともぜひやってみたいと思うか，はたして皆さんはどちらでしょうか．

●予備検討・LSI メーカ調査

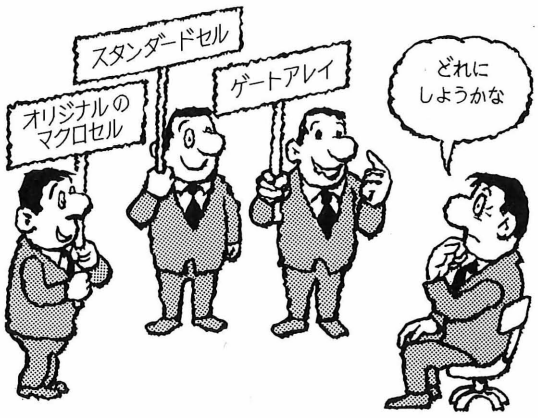
ある電子機器メーカに入社して 5 年目の A 君は，これまでに CPU を搭載した制御システムを 2 機種ほど開発したことがあり，ハード・ロジックの技術も CPU ソフトウェア開発も，中堅としてこなしてきました．そして，今回ついに，ある映像処理システム開発のために，中心となる信号処理用の ASIC を設計することになりました．このシステムは，従来は汎用 DSP チップと多くの TTL を組み合わせたボードとして，かなり大規模なものになっていました．A 君は全体のシステム設計からすべてを任され，制御用の CPU 以外のディジタル回路部分と DSP 演算処理部分を，新しい ASIC として 1 チップ化することを提案して，会社から承認されたところなのです．

A 君はまず，従来の映像処理システムの仕様書と回路図から，どのくらいのデータ処理能力が必要で，どのくらいの規模・スピードのディジタル回路であるかを検討しました．そして，ASIC 化する回路部分のブロック図レベルのシステム検討を開始するとともに，ASIC メーカ各社の代理店に連絡をとって，各社の状況の調査に入りました．

初めてのことなので，最初は LSI をゲートアレイとスタンダードセルのどちらで実現するのかもわかりませんでしたが，システム検討の結果，乗算器と演算用高速メモリをチップ上に搭載する必要があることから，スタンダードセルを採用することに決めました．メーカ各社のセールス・エンジニアと話をしてみると，カタログでは威勢のよかった B 社はあまりスタンダードセルの実績がなかったり，大手の C 社は乗算器などのメガセルがあまり得意でなかったりと，候補メーカが意外に脱落していきました．そして，最終的に D 社と E 社が残りました．

D 社はゲートアレイの老舗として有名で，「スタンダードセルも世界的に実績がある」と強気でした．E 社はどちらかというと後発メーカなのですが，「オリジナルのマクロセルまで作ることができますよ」という柔軟な最新の開発環境を自慢していました．A 君は上司と相談して会社のトップに決定を求め，これまでの取り引き実績から，今回は D 社によって開発することが決定されました．

A 君はシステム検討からより具体的なシステム設計を進めて，おおよそのチップ上の機能ブロックと所用ゲート規模が見えてきました．D 社のセールス・エンジニアとのミーティングが重ねられ，見込みベースでのチップ単価が提示され，開発期間・開発費用・開発スケジュールが検討されました．そしてついに，LSI メーカとの技術的な秘密保持契約と ASIC 開発契約が締結され，開発費用(8 ケタ！)の半金がメーカに渡されました．もう，後には引けません．

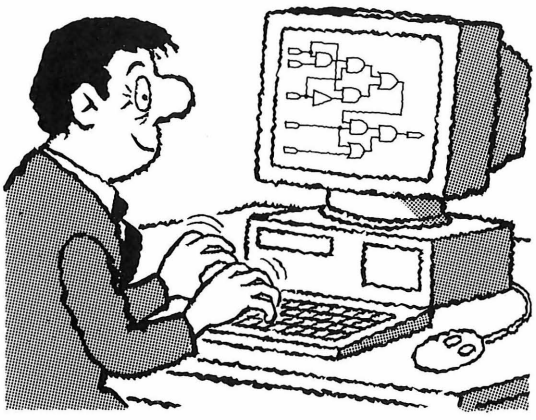


● ASIC デザイン・センタへ

やがてA君は，実験室で設計の完了した回路図をもって，D社のLSI開発デザイン・センタを訪れました．ここでは，ユーザ(LSIを設計するメーカ)ごとに独立したデザイン・ルームが割り当てられ，さながら個室専用オフィス・ビルのように，それぞれお互いに関与しないユーザが，ずらりと並んだ部屋の中で黙々と仕事をするのです．開発期間中は，ICカード化されたデザイン・ルームの電子キーがA君に渡されました．この期間，他の利用者はもちろんLSIメーカの社員であっても，この部屋に勝手に入ることはできず，A君は設計に関する資料を持ち込んで広げておいても安心なのです．

A君はしばらくの間，自分の会社でなくこのデザイン・ルームに毎日通うことになりました．部屋の電話から会社に定期連絡をとり，ASICの設計が完了するまでは会社に帰らない覚悟です．デザイン・センタは完全な24時間体制で，寝袋を持参する利用者もいるそうです．入場・退場は玄関のカード・リーダに電子キーを入れることで自動的に記録・管理されていますし，食堂の食券も電子キーで購入できます．コーヒー党のA君は，ロビーの自動販売機にだいぶお世話になりました．毎日食堂とかで顔を合わせるために，まったく別の業界のASIC開発エンジニアにも，何人か友人ができてしまいました．

新しいデザイン・ルームは会社の実験室よりもずっときれいで，A君はすっかりリラックスできました．EWSとパソコンが並んだ大きなデスクと，さらに回路図を広げられる別のデスク，それにプリンタのラックと，ASIC設計ツールのマニュアル(マル秘・禁帯出のハンコが生々しい)が十数冊置かれた書棚．疲れない椅子にふかふかのカーペット(会社の冷たい床とは大違い)．わからないことは電話でサポート担当の技術者に問い合わせできるし，会社からの電話やFAXもすぐに届きます．A君はASICメーカの設計技術者が，ちょっとうらやましくなりました．



● CADによる回路設計

A君はまず，デザイン・ルームの設計CAD用パソコンに向かいました．D社では，LSIの回路設計の部分は，自社製の32ビット・パソコン上のCADツールを推奨しているためか，この作業だけはEWSではなかったのです．10,000ゲート規模の回路というのは，もちろん1枚の回路図にはなりません．LSIメーカのデザイン・サポート担当技術者のアドバイスを受けて，A君は全体を3層の階層構造としました．トップダウン式に設計すると，最上層の図面は，チップの入出力ピンにつながる入出力マクロセルと，内部の各機能ブロックの「箱」がならぶだけになりました．そして，つぎの階層の図面では，各ブロック内の実際の回路が，それぞれ別々の図面として記載されていきます．そして，その中の一部の回路ブロック(あちこちで共通に使われる，たとえば16ビット・ラッチ回路とか，8ビット4入力データ・セレクタ回路など)をまた「箱」と記述して，さらに下の階層の回路図として展開する，という作戦です．

過去の経験から，TTLやCPUの周辺LSIを使った回路設計に慣れていたA君にとって，CADの使い方(コマンドやマウスの操作法)さえ覚えてしまうと，このCAD回路設計はとても楽しい作業となりました．これまでの方眼紙の回路設計では，修正のたびに消しゴムのカスや線の引き直しに苦労しました．ところが，CADであれば消去や移動もカンタン，そして任意のTTLが，番号を指定するだけで引っ張ってこれるのです．内部バス・ラインや各セルの入出力ラインは，ちゃんとマウス・ドラッグによる箱の移動についてくるし，セルの入出力ピン番号も自動生成されます．演算処理回路に特有のワンパターン回路についても，似たようなブロックは，範囲指定とコピー指定によって，簡単に複製できます．A君は，これまでの手書きの2

倍から3倍のスピードで，TTLで数十個のブロック図を1日に1枚のペースで入力していきました．

CAD回路設計の作業の途中からは，回路図をプリンタから打ち出してチェックします．正式な全体回路図のプリント・アウトは，EWS経由でデザイン・センタ内の専用大型プロッタからも得られるのですが，設計途中の各ブロック単位の回路図は，デザイン・ルーム内のパソコンのプリンタからも出力できるのです．設計の途中では，別のブロックとの対応をしばしば調べる必要があり，さらにマーカ・ペンで書き込みながら，何度も何度もプリント・アウトするほうが便利なことも多いようです．一つのブロック回路図がA3のサイズで，これをセロテープでデザイン・ルームの壁につぎつぎと貼っていきます．ミスを発見したり，図面と図面の信号名などの対応が食い違ったときには，すぐにCADの図面を呼び出して修正をかけ，最新版をプリント・アウトして差し替えます．2週間ほどすると，10畳ほどのデザイン・ルームの壁は十数枚の図面で完全に埋まってしまいました．これで，まずは回路図の完成です．

●ハードウェアの論理検証

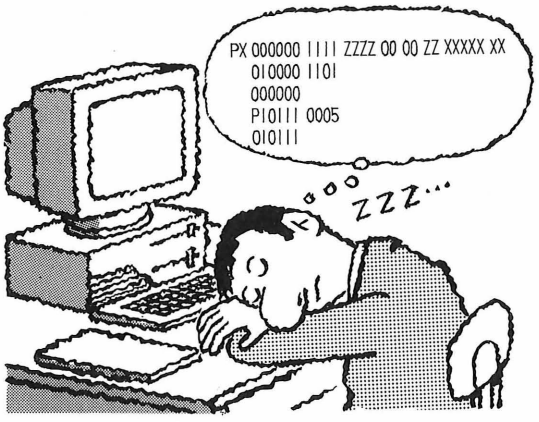
すべての回路図が揃うと，パソコンCADの図面情報ファイルを，同じデザイン・ルーム内のEWSにイーサネットを経由して転送します．そして，今度はEWS上でこのデータをコンパイルし，最初の段階である「論理チェック」を行います．これはシミュレーションの前の段階のチェックで，未定義のピンとか，未処理(オープン状態)の信号入力ラインなどの矛盾を検出してくれます．慎重に回路設計したつもりのA君でしたが，やはり数ヵ所の抜けがありました．この場合の追加・修正・再コンパイルも，CADの環境であればまったく苦になりません．

このハードウェアの論理チェック・ツールは，物理的な接続の問題だけでなく，各ゲートの入力負荷(ファン・イン)と出力ドライブ能力(ファン・アウト)の対応のチェック「ちょっとドライブがしんどいのでは？」とか，使わないので放っておいた出力ピンに対して「本当にいいの？」とか(実際には英語のメッセージで)聞いてくるなど，なかなか頼りになるものです．プリント基板の回路の場合には，最悪の場合には「切った貼った」の対策ができますが，ASICとしてチップ化された回路には修正がききませんから，過剰とも思えるほどのチェックがあるものなのだ，とA君は納得しました．

●テスト設計

そしてつぎの作業は，テスト・プログラムの設計です．ここでは基本的には，システム・クロックにしたがって，ある入力パターンの場合に，回路からどのような出力パターンが期待されるか，という「期待値」をプログラムしていきます．つまり，テスト・プログラムというのは，LSIを検証するためのテスト・パターンを必要にして十分なだけ集めたもの，ということなのです．ここで気をつけなければならないのが，「回路図に合わせてしまう」ことで，ついつい，回路図を見て結果を求めてしまいますが，実際には回路設計の際に「こう動作するはずだ」と想定された期待値を記述することになります．

テスト・プログラムは，最終的なシミュレーションの際には所定のデータ形式に統一されるのですが，設計の段階ではいろいろな形式がありました．A君は，メーカの用意したパソコン上の，マウスによるグラフィック形式のテスト・パターン用CADでなく，パソコン上の汎用テキスト・エディタを使って，テキスト形式のテスト・パターン・データ・ファイルを編集することにしました．LSIのテスト・パターンというのは「同じようなパターンの繰り返し」が非常に多く，使い慣れたエディタによる，カット&ペースト機能のほうが効率がよさそうだったからです．(そしてあとから振



り返ってみると，シミュレーション段階での検証のときにも，このテキスト形式のプリント・アウトはステップごとの変化が並んでいるために，チェックがしやすくて助かりました.)

ASICのテスト・プログラムというのは，LSI上のすべてのゲートの状態を変化させて，完全にチップ上の動きを検証できるものでなければなりません．このため，回路をいくつかのブロックに分けて，ブロックごとに入力パターンを変化させ，出力パターンを期待値として記述していきます．ところが，各ブロック図の入出力信号名というのは，相互をCAD内で結合するために便宜的につけた「記号」であって，LSIのテスト・プログラムとして規定できるのは，実際に外部との接点となる，LSIの入出力ピンだけしかありません．そこで，いちばん外側の入力ピンからいろいろなデータを与えて，クロックのステップが進むとともに内部の回路状態が変化していき，ついに目的とするゲートの状態を変化させ，今度はこの変化を最終的な出力ピンから得なければならないことになります．テスト・プログラムの種類は五つ，六つと増えていきますが，なかなか全部のゲートの状態を検証することができません．A君はちょっとアセッてきました．

●論理シミュレーション

テスト・プログラムの作成と並行して，先に出来たパターンからEWSに転送して，所定のデータ形式に変換するコンバージョンをかけて，いよいよ論理シミュレーションも始まりました．これは，完成した回路図データに対して，テスト・プログラムのパターンを入力ピンに与えたときの出力を，EWS上のシミュレーション・ソフトによって検証するものです．出力ピンから得られるパターンが，テスト・プログラムで与えた期待値と一致すればOKなのですが，実際に走らせてみると「エラー」が出ます．そのたびに原因を検討して，回路図のミスがあれば回路を修正して再度コンパイルし，テスト・パターンのミスがあれば修正して再度コンバージョンし，そして再度，シミュレーションにかけます．シミュレーション結果に「合わせて」安易にテスト・パターンを変更すると，回路のミスがマスクされてしまって危険なので，この修正には細心の注意が必要です．

いくつかのテスト・プログラムが通って(OKになって)くる一方で，どうしても回路の奥深くの一部のゲートの状態を検証することができず，A君は困ってしまいました．演算回路の末端近くのために，入力ピンからのデータ変化が到達するまでに2,000ステップほどかかってしまって，思ったようにその部分のテスト・パターンが記述できないのです．メーカのサポート担当技術者に相談して，A君は回路図を大幅に変更することにしました．回路図のほぼ中間の部分にテスト用のバス・ラインをばっさりと入れて，テスト・モードでは上流の回路を経由せずに，直接に演算回路の入力部分にデータを設定するようにしたのです．これで，テスト・パターンのステップ数は大幅に減少しましたが，それまでに通っていたテストは，またすべてやり直しになりました．しかし，すでに作業の手順に慣れてきたので，まったく白紙からの再開というわけではなく，「急がば回れ」の格言どおり，これでようやく先が見えてきました．

デザイン・ルームでの設計がそろそろ1ヵ月になる頃，10種類ほどのテスト・プログラムと，壁に貼ってある十数枚の回路図も，だいぶ固まってきました．最初はとてつもない大規模な回路図と思えましたが，今ではA君にとって，10,000ゲートのどの部分でも，すぐにピンとくるまでに目の届くところとなりました．シミュレーションはすでにTypicalモードのテストを終えて，MaxとMinの条件(電源電圧・温度の両極端なケースのテスト)に入っています．これは，周囲条件によってゲート遅延時間がかなり変化するために，各ゲートの信号伝播が積み重ねられた状態をシミュレーションして，そのワースト・ケースでも期待値を満足させるための検証です．すべてのテスト・パターンのシミュレーションを走らせるためには，かの有名なEWSをフルに回転させても3時間ほどかかるために，A君はウォークマンを聴きながら，余裕でレポート書きをしています．

●「OK出し」以後の日々

やがてすべてのシミュレーションでOKが出ると，そこからはレイアウト，実配線シミュレーション，マスク，プロセス，そして試作のES(Evaluation Sample)チップまで，LSIメーカ側の作業となります．さす

がに，「これでテストはOKです」という承認の書類にハンコを押すときには緊張しましたが，もうやるべきことはやりましたから，運を天に任せて無事，完了となりました．

そして，A君は久しぶりに自分の会社に戻りました．しかし，ここから遊んで待っているわけではありません．まず，実際の製品のためのCPUボードの設計・試作がありましたが，これまでの経験から，比較的簡単にできました．CPUボード上に，やがて出来てくるはずのASICチップのリード・パターンを置いてしまえば，それでおしまいです．ただし，このチップはまだこの世にありませんから，この基板のテストはここでしばらくおあずけです．

つぎに，このASICを今後使っていくための資料として，ASICのデータ・シートを作成する作業も待っていました．社内汎用LSIとして使ってもらうためにも，「自分だけがわかる」資料では意味がありません．また，A君の作成したこの資料は，LSIメーカからA君の会社に公式に提出される，「LSI仕様書」の原稿ともなるのです．これまで，さんざん各種のLSIのデータ・シートを活用してきたA君ですが，いざ自分で書いてみると，なかなか大変なものであることを実感しました．

● ES評価用ボードの試作

そして，メーカから完成してくるESをテストするための「評価用ボード」を，急いで作らなければなりません．A君は以前に作った実験用CPUボードとユニバーサル基板とを組み合わせて，ESを入れるソケットを載せた，ES実験用ボードを作りました．ボード上のCPUソフトとして，いちいちICEを経由してテストするのは面倒なので，パソコンからテスト信号を転送するようにしました．つまり，基板上のCPUのソフトとしては，パソコンと通信して，パソコンからの信号をESに与えるだけの「モニタ」プログラムを組んだのです．

やがて約束の数十日が経過すると，ついにLSIメーカから十数個のESが届きました．開発費用をESの個数で割ると，この段階では，このLSIは1個が100万円以上のチップということになります．まずはアース線を握って静電気を逃がし，おそるおそるチップをES評価ボードにセットして，電源を入れて，慎重に機能チェックに入ります．ESの評価期間は契約により10日間．この間に考えられるすべてのチェックをして，「問題がない」ことを検証し，A君の責任でLSIの完成を宣言しなければなりません．A君はロジアナ，シンクロ，スペアナなど，実験室の機材を総動員して，じっくりとテスト項目に取り組みました．

● 「本承認」の儀式

そしてついに，「本承認」の期日になってしまいました．この承認書にハンコを押してしまえば，開発費用の残り半金が支払われ，LSIメーカは「量産」に入ってしまいます．しかし，ここまで来ればもう悔いはありませんから(ここで立ち止まっても仕方ない)，A君は**LSI承認手続き**の「儀式」を淡々と済ませました．といっても，LSIメーカの規格による「ES承認確認書」の担当者欄にハンコを押して，上司にもハンコをもらって，代理店の担当者にFAXするだけのことでした．

ここからは，いよいよ本番のCPUボードにESをハンダ付けして，システム本体のソフト開発からデバッグまで，手慣れた作業の開始です．あとは，量産の第1陣としてCS(Commercial Sample)が1ヵ月後に届けば，これまで1枚十数万円していたボードの回路が，開発費を別にすれば単体のコストとしては2ケタほど安い1チップになって，A君のシステム基板上で，そしてやがては別の製品でも活躍していくのです．

最初のシステム検討として開発を開始してから，ここまで約5ヵ月．A君は，入社以来のいろいろな経験と技術とが，世界に一つしかない自分のASICとして結実した手応えと，エンジニアとして一回り成長した自分とを，なんとなく感じていました．

＜エピローグ＞

エンジニア万歳！

●技術者のライフサイクル

本書では，フレッシュマン技術者の皆さんのために，おもにスタートラインから数年間のステップアップについて，いろいろな視点から考えてきました．ここでは，その先のエンジニアの姿，あるいはもっと長く「技術者のライフサイクル」について検討してみたいと思います．

新人時代の研修のときに「まずは10年間，なんでも吸収して一人前になってみろ」と教わった経験がありますが，この「10年サイクル」理論というのは，ちょっと面白いものだと思います．そこで，実際には個人差なり誤差があるとしても，約10年ごとに節目を区切った「フェーズ」の組み合わせとして，エンジニアの人生設計の考察を試みていくことにします．ただし，「人生70年」とかいいますが，残念ながら筆者はその半分にもなっていませんので，いろいろな機会(マスコミや学会など)で知った，多くの先輩エンジニアたちの姿を参考にした，たんなる「お話」だと思って読んでください．

まず，人生最初の**第1・第2フェーズ**というのは，日本人であれば誰でもほぼ共通の枠組みです．つまり，幼児教育・義務教育の期間，そして高校・大学・専門学校などの上級教育期間を経て「社会に出る」まで，を最初の2フェーズとしておきましょう．ここは，社会に出るための基礎的な勉強，などとケチなことはいわずに，ひたすら「人間形成」の期間だと思います．遺伝・環境・読書などによって形成される「性格」とか，学生生活・クラブ活動などで得る「親友」，「熱中」，「青春」，「趣味」などの経験が，その後の人生の原動力として最大の財産となります．

そしてつぎの**第3フェーズ**が，フレッシュマン技術者の皆さんが直面する「社会人になって最初の10年間」という期間に該当します．ここはとにかく，あらゆるものを吸収・経験・勉強して，エンジニアとして一人立ちするための大切な時期でしょう．さらに多くの人にとって，家庭や家族という生活基盤を築く期間だったり，一生モノとなる自分の「趣味」を追求する大切な期間でもあります．よく「仕事が趣味」という人がいますが，もともと趣味とは「稼ぎを貢いで追求するもの」ですから，仕事以外に頼るものがない人生ほど淋しいものはないと思います．

さて，この第3フェーズを経て一人前の技術者となった節目が，30歳から35歳あたりにやってきます．いよいよ，あらゆる意味で社会に貢献する，というスタートの時期です．ここからの展開については，雑誌に載った著名人の話とか，筆者の知っている実例が多くありますので，以下にいくつか「物語風」に紹介してみたいと思います．なお，この時点で「残りフェーズはあと3コマしかない」，あるいは「いよいよここからが本番の後半戦」と認識してみることは重要なことでしょう．節目の期間を区切って，その範囲ごとに目標をもって燃焼する，という姿勢は，自分の人生に対するスケジューリング技術といえるかもしれません．

開発・設計の現場でバリバリ頑張ったA氏は，主任，課長とポストが上がって，次第に「管理職」となりました．技術の現場から離れていくのがやや淋しいものの，若手と一緒に新しい技術についていくのが大変になってきたので，身についた技術で仕事ができるのは好都合という側面もあります．中間管理職としてのストレスや，新人類の扱いに苦労しながらも，このままなんとか今の会社と共に生きていけそうです．ただ，自分と同じ団塊の世代のライバルも多いので，将来のポストはちょっと心配です．

技術的な実績を十分に積んだB氏は，自分から人材

バンクに登録して，それまでとは異なる業界のメーカに転職しました．もとの業界での技術はほぼ獲得してしまって，それ以上続けても技術者としての成長が見込めない，という思いが強くなったからです．B氏はすでに身についた技術力をベースに，新しい業界でさらに技術フィールドを広げるとともに，収入も1.5倍ほどに増えました．この調子で節目ごとに「転業界」していくと，全部で四つの業界のプロになれそうです．

社会人として最初の第3フェーズでの経験から，技術よりもマネジメントの面白さに目覚めたC氏は，30代の節目に思い立って独立し，会社を設立しました．それまでの人脈から，何人かのエンジニアも確保できて，大企業でできなかったことを，どんどん実現しています．収入が不安定になることを危惧していたのですが，機動力のあるシステムハウスの仕事はたくさんあって，むしろ収入は大幅にアップしました．いずれは独自ブランドの製品も出したい，とC氏はさらに先を狙っています．

技術者としてほぼ一人前になったところで，本格的な「勉強」の必要性に目覚めた人たちも，たくさんいました．会社から「派遣研究員」の了解を取り付けて，単身赴任しながら技術研究財団に出向しているD氏．収入がダウンするのを承知で大学の教官に転職して，自由に研究できる環境で再スタートしたE氏．会社を辞め家族を連れて，アメリカの有名な研究機関に留学してしまったF氏．いずれ弁理士として独立するために特許事務所に転職し，毎日猛勉強しているというG氏．周囲の反対を押して，アルバイトで家族を養いながら再び大学に戻って，異なる学部の「大学院生」になったH氏．30歳になる前に自分の力をいちど試してみたい，と異なる言語環境でフリーのプログラマになったI氏．

いろいろ並べてみましたが，どうやら第4フェーズ以降の人生の組み立てというのは，ほとんど一人一人の技術者の個性そのもののようです．確実にいえることは，まずはエンジニア最初の第3フェーズをいかに充実しておくか，ということでしょう．基礎となる技術力，あるいはエンジニアとしての広い視点が成長していなければ，その後の人生の選択肢を自分から狭めてしまうことになるわけです．ライフサイクルのまさに真っ只中にいながら，自分のライフサイクル全体を眺めるというのは難しいことですが，ときどき思い出して考えてみたいものです．

●不良社員モードのすすめ

コンピュータ技術の世界には，「ハッカー」という言葉があります．もともとの意味から一人歩きして，最近はちょっとマイナスのイメージがありますが，アクティブな姿勢のエンジニアの称号として大切な呼称だと思います．与えられる開発システムと開発テーマに対して，ただ受動的に仕事をする技術者はハッカーにはなれません．仕事の中にオリジナリティを自発的に盛り込み，仕事のようでじつは趣味として楽しんで，既存のシステム(装置・ツール・ソフト)をどんどん改造して，いつのまにか自分の技術力をしたたかに高めてしまっている，というようなハッカーになりたいものです．

これと似たような意見として，「不良社員モード」の話というのがあります．ちょっと言葉が刺激的ですが，中身としてはなかなか賛成できる，フレッシュマンにも知ってもらいたい姿勢です．まず，大企業に限らず，会社組織に勤める人は，自分が「組織の歯車」であることを認識できるでしょうか．自分でないとできない重要な仕事をやっている，自分が今の仕事から突然に抜けたら会社は大打撃だ，などと内心思っていませんか．これは完全な間違いで，会社というのは自分が消えたところですこしも困らず，つぎの歯車を充当するだけなのです．そこで，ある意味できわめてクールに，自分と会社のギブ・アンド・テイクの関係を考えてみる必要があるようです．

たとえば，多くの若手技術者の自宅や会社に，「自己研修プログラムの案内」のDMや電話が届いているでしょう．「あなたは選ばれた××人のうちの一人です」とか「書店では販売しない，特別予約購読のみです」など，ほとんど悪徳商法スレスレの，きわめて高額なものばかりです．しかし，興味をもつ向上心は別として，筆者はこのような誘いにのることには反対です．本当にタメになるのであれば，会社にいってカネを出してもらいましょう，と提案したいのです．将来的にその技術者の能力向上として会社に貢献するものであれば，会社は支援してくれるはずです．書籍の購入にしても有料セミナへの参加にしても，なんでも「自費」で勉強するというのは，プロとしては甘すぎるのだ，という意見さえあるのです．

これをさらに一歩進めて，より積極的に会社を利用しているエンジニアも多くいます．会社から与えられた開発テーマの裏で，つねに自分なりの研究開発テーマをもって実験・試作している人，会社に会費を出してもらって学会に参加してどんどん勉強する人，会社のマシンから広域ネットワークの議論にアクセスして積極的に人脈を広げる人，会社の蔵書を資料として勉強して各種の資格を取得していく人，休憩時間に会社のコンピュータでオリジナル・ゲームを作ってアルバイトする人…．

しかし，これらの不良社員エンジニアというのは，「何も言わずに与えられた仕事だけをするサラリーマン社員」よりも，よっぽど素晴らしいのではないか，とさえ思います．なによりテクノロジーに敏感で，あちこち広範に興味をもち，ここぞというときには集中的に取り組みます．何かのきっかけで，次代を担う画期的なアイデアを発掘するのは，これらのアウトロー技術者の仕事ではないでしょうか．「求めない者には与えられない」という格言は，技術者の世界でも真実で，自分から伸びようとしない技術者を，会社は何もフォローしてはくれません．周囲の環境や会社の支援を積極的に利用して，たくましく自分を成長させてしまいましょう．

●真の「勉強」とは

アメリカの大学では，社会人になってから大学に通ったり，ある専門分野のプロとなってから，別の分野の学生として大学に戻ることが普通になっています．これは大変に素晴らしいシステムなのですが，残念ながら日本ではあまり一般的ではありません．しかし，いろいろな分野の多くの中堅技術者にとって，前述の第3フェーズ，つまり30歳を過ぎた頃から猛烈に勉強したくなることは多いようです．

筆者の場合には，エンジニアとしての経験を蓄積して，エレクトロニクスやコンピュータの世界が自分のものとしてそれなりに整理・理解されてくると，受動的な教育でなく，本当の意味で「勉強」したくなりました．テーマは大きく人工知能と認知心理学，そして感性情報処理といった分野で，いずれも大学時代の専攻や実際の仕事とは関係のないものでした．ところが気付いたときには遅いのが世の常で，家庭をもち家族を養う身では，おいそれと学生の身分に戻ることも，ましてや研究のメッカ：米国に留学することも夢のまた夢というわけなのです．

エンジニアとしての生活と並行してできる「勉強」といえば，学会に参加して地道に過去の論文を読んでいくとか，ネットワークでその分野のプロの研究者に質問するとか，といったスローペースのものでしかありません．しかし，専業の研究者・学生のように多くの時間は割けないにしても，ここでの勉強は「地に足のついた」ものですし，自分の専門分野という一つのバックボーンもあります．いずれは何らかの形でモノにしてやろうと，ある意味では学生よりも情熱的に取り組んでいる，というのが「技術者の勉強」の姿のようです．当たり前のことですが，勉強とは一生続くものなのです．

もし，本書の読者の皆さんの中に，まだ学生時代を多く残している人がいたとしたら，世の中の多くの技術者の言葉として，「勉強をやりたいだけできる，という幸せな時期に，悔いのないようにトコトンやっておきなさい」とアドバイスしておきましょう．

●人生ゲーム

コンピュータ・ゲームのジャンルの一種に「ロールプレイング・ゲーム」がありますが，技術者の生き方もこれに近いように思います．おりしも，ちょうど昭和-平成のバブル経済が崩壊し，東欧に続いてソ連の旧体制が壊滅するという激動の時代となっていますが，これら政治・経済の世界と違って，ある意味でエンジニアの世界は，次第に「経験値」が増加して成長することはあっても，投機に失敗して破産することも，革命によって失脚することもない，着実な世界なのです．

もちろん偶然とか運命といった要素はあって，たまたま担当した研究開発テーマが時代をとらえてヒットするエンジニアもいれば，長年の担当分野が時代の遺物となってしまう悲運の技術者もいます．たまたま所属した企業・業界の景気次第によっては，苦労する人も順風を満喫する人もいるでしょう．技術の世界というのは，かなり厳しい競争の世界でもありますから，(一時の)勝者もいれば敗者もいるわけです．それでも，固定した先入観に頼らずに何にでも興味をもって，新しい考え方に積極的にトライする，柔軟な発想と熱き心を忘れないエンジニアにとっては，人生はバラ色の「楽しいゲーム」を提供してくれるのではないでしょうか．

筆者は大学時代のクラブ三昧の生活で，一生の親友といえる(「会社」では得られそうもない)多くの友人を得ました．それぞれ就職が決まると，その中で「銀行・金融・保険業界」に就職する文系の友人をつかまえて，「他人のカネを右から左に動かすだけで儲けるとは，とんでもない世界だ」などと冗談半分にからかったものです．しかしある意味で，10年前のこの意見は，いいところを突いていたようにも思います．多額の研究・開発投資を必要とし，さらに材料コストに生産コスト，そして営業・販売コストが上積みされる「製造業」(多くの技術者のメイン・フィールド)というのは，有形コストの不要なスマートな業界から見ると，えらく「重たい」業界かもしれません．

しかし明らかに，メーカがモノを作らなかったら，現代文明は存在できません．そして，この「モノを作る(生み出す)」ということの社会的意義に加えて，さらにエンジニアである自分自身を自分の力で成長させることができる，という可能性が，何より素晴らしいことだと思います．これからの技術者というのは，もっと正当な待遇面での評価(報酬・環境・処遇など)を得ていくものだと思いますが，当面の安月給を忍んでも頑張る多くのエンジニアを支えているのは，この仕事の「手応え」と「誇り」なのだろうと思います．

●エンジニア万歳！　(あとがき，あるいは筆者のひとりごと)

さて，本書の最後の小見出しは「エンジニア万歳」です．ここで内幕を明かしてしまいますが，筆者は本書を完全な「トップダウン方式」で執筆してきました．つまり，巻頭の目次を最初に完成させて，その後に目次の小見出しをタイトルとして，各章の各節をワープロで執筆していったのです．ということは，本書の最後は「エンジニア万歳」でハッピーエンドに終わる作戦に，最初からなんとなく決めていたということです．

しかし実際には，最終「エピローグ」の章は難航しました．1回目の原稿はあまりに暗くなってしまい，自分から「没」としました．ちょっと油断すると，えらく暗いものに落ち込んでしまうのが「エンジニアの将来像」なのかな，と自分でも驚いたほどです．そして，最終的な本書のこのバージョンは何回めかのトライの結果なのです．この事実から読者の皆さんが何を感じるかは自由ですが，正直な舞台裏はそのようなものでした．

そして最後に，やはり筆者の本音としては「エンジニア万歳！」と素直に唱えたいと思います．もし読者の皆さんが社会人としてスタートの前後にいるとしたら，筆者は約10年間だけ先輩，皆さんが30歳代前半であるとしたら同年代，40歳代以上であるとしたら，筆者はたんなる生意気な若輩者でしかありません．それでも，いろいろな分野で多くの先輩に恵まれて，さらに多くのフィールドにトライして，何より自分なりに頑張ってやってきた，という実感だけはあります．そして「モノを生み出す」とともに，次代を担うフレッシュマンを育てる，という新しい手応えの世界も知りました．この願いが，本書のそもそもの原動力となっているのです．

ここまで書いてきておいて，「じつは筆者も悩めるエンジニアです」というのは通用しないかもしれませんが，日々いろいろ考えることは多くあります．人生に残された時間と勉強したい事柄との当惑しそうなアンバランス，自分の生活や人生設計と家族や趣味のこと，この1週間の間に届いた新しい技術ネタの応用可能性，プロダクトとマネジメントとコンサルティングとの関係，文化・文明と技術との関係(地球環境も製造物責任も知的所有権もみんな根は同じ)，AIと哲学と脳の問題，英語を勉強しないと(会話もできない論文も読めない)駄目だなぁという毎度毎度のプレッシャー…．

参考になったかどうか，これがあるエンジニアの素顔というわけです．しかしこのエンジニアは，とっても楽しんでいます．どうぞ皆さんも，できればこの世界の仲間として，大いに自分を試していってほしいと思います．求めただけ与えられる，素晴らしい世界が誰の前にも広がっているのです．

INDEX

【ア　行】

アイソレート……75
アクティブ……59
アクティブ回路……41
アクティブ・フィルタ……43
アクノリッジ信号……67
アセンブラ……69,75
熱き心……5
アドレス・デコード……58
アドレス・バス……58
アナログ……41,105
アナログ-アナログ変換……45
アナログ⟷ディジタル変換……44,105
アンド……32
イーサネット……152
位相累算方式……90
イミュニティ……113
インターフェース……11,67,108,140
インタプリタ……62
インテル HEX 形式……80
インバータ……32
インパルス・ノイズ……75
ウィンドウ方式……54
ウォッチドッグ・タイマ……111
エクスクルーシブ・オア……32
エラー・メッセージ……80
エレクトロニクス技術……3
演算回路……36
エンジニア万歳……6,174
エンジニア冥利……4
エンジニアリング・プロデューサ……128
オア……32
オブジェクト指向……63
オブジェクト効率……83
折り返しノイズ……107
オリジナル CPU……149
オリジナル・ボード……51,72,153
音声合成……55
音声入力……54

【カ　行】

階層化……16,137
開発環境……14,79
開発支援ツール……77
回路図……17
拡張スロット……48,159
拡張性……72
拡張ボード……48
加算器……36
カスタム CPU……149
仮想現実感……20,56
仮想端末……152
学会……120
可変クロック発生回路……90
カラー・コード……17
「緩衝」機能……87
キーボード……53
記憶回路……40
機械語ルーチン……62
企画……12,124
規格……13
技術者の創造性……21
技術情報……13,17,116
技術不安……3
キャリ信号……36
究極チップ……139
共有情報の分散保持……142
境界値分析……111
組み込み機器……98
クライアント-サーバ方式……152
クロス・アセンブラ……75
クロック……32,88,90,139
計測技術……109
計測ネットワーク……153
ゲート……32
ゲートアレイ……129
ゲートアレイ内蔵 CPU……143
ゲート遅延……103,139
ゲーム……5,174
結合度……84
検証……15,126

コイル……41
ゴースト……59
構造化プログラミング……77
コスト……17,127
誤動作……115
コピーライト……63
コンセプト……13,124
コンデンサ……41
コントロール・バス……58
コンパイラ……63
コンパイルド・セル……132
コンピュータ技術……3

【サ　行】

サーバ……152
サブルーチン・コール……82
サラリーマン・エンジニア……5
暫定情報……95
サンプリング定理……107
シー・オブ・ゲート……133
シーズ……13,125
シールド……114
視覚的インターフェース……54
時間……106
シグナル・コンディショニング……45
自己診断プログラム……111
仕事地図……3
師匠……18
システム・オン・チップ……143
システムの出力技術……54
システムの入力技術……53
システムハウス……5,12
実装技術……15
実配線シミュレーション……168
シフト・レジスタ……36
時分割多重化……109
シミュレーション……132,147
社内汎用 LSI……132
周波数帯域……108
周辺 LSI……88
出力ポート……59
受動回路……41
シュミット・トリガ……93
順序回路……32
仕様……17
乗算器……38,132
情報の外挿……123
情報源……116
情報の差分……123
情報の補間……123
シリアル-パラレル変換……88
資料請求……119
人工知能……21,53
信頼性……113
信頼性技術……15,109
信頼性設計……110
真理値表……32
数量効果……162
スケジュール……17,127
スタンダードセル……132
ステータス・ライン……68
ステータス・レジスタ……68
ストローブ信号……67
スペック……17
スレーブ……67,152
セカンド・ソース……97
設計・開発……12
セミカスタム CPU……143
セミカスタム IC……98
セル・コンパイラ……132,149
線形性……20
センサ……45,153
セントロニクス……68,151
ソフトウェア割り込み……69,84

【タ　行】

ダイナミック表示……75
第 2 グループ……147
タイマ割り込み……70
ダウンロード……78
多値信号……106
ダブル・バッファ……87,92
ダンプ・リスト……63
知的所有権……13,120
チップ分割……135
チャタリング防止……60,93
調査・研究……12
陳腐化……96
通信……120
抵抗……41
データ・グローブ……55
データ計測技術……14
データ・シート……169
データ・バス……58
データフロー方式……160
ディジタル……105

デザイン・センタ ……166
デジアナ混在……52
テスタ……20
テスト設計 ……167
テスト・バス ……137
テスト・パターン ……132,140,167
テスト・モード ……137,168
デバイス・ドライバ……71
デバッガ……70
デバッグ ……126
デバッグ環境……79
デバッグ技術……20
デバッグ・モニタ……20
デュアル・システム ……111
デュアルポート RAM ……132,151
等価回路……32
同期回路……32,139
ドキュメント……15,124
特許……15,120,125,128
トップダウン・プログラミング……75
トランジスタ……41
トランスピュータ ……159
トラブル……79
トレードオフ……16,46,68,108,126

【ナ　行】

ナンド……32
ニーズ……13,125
ニューラル・ネットワーク ……128,158
入力ポート……59
人間工学……39
ネットワーク ……151
ノア……32
ノイズ……72,109,113
ノイズの伝導経路 ……114
ノイマン方式 ……135
ノウハウ……19

【ハ　行】

バーチャル・リアリティ ……20,56
ハーバード・アーキテクチャ ……160
ハーフ・アダー……36
ハイビジョン ……136
パイプライン ……140
バイリンガル技術者……14
バグ……20
バケツ・リレー ……156
パスコン ……115
バス・ファイト……59
パズル ……21,72
パソコン ……48,60
パソコン ICE……78
ハッカー……71,172
バック・アップ……79
パッケージ……93,129,162
ハンドシェイク……67,164
汎用エディタ……78
微細化 ……13,96
ビットスライス CPU……134
非同期待ち合わせ ……140
非ノイマン方式 ……160
秘密保持……51,100,165
ヒューマン・インターフェース ……39,56
標準化 ……125
ファウンドリ ……126,129
ファームウェア……98
ファジィ……53,157
フェイルセーフ ……111
フォース・ディスプレイ……55
フォールト ……110
物理する心 ……4
物理量……45,106
プライオリティ ……121
ブラックボックス ……17,41,103
フリップフロップ……32
フル・アダー……36
ブレッド・ボード ……139
フローチャート……60
プログラマブル・カウンタ……88
プロジェクト管理……21,127
プロトコル ……152
プロトタイピング ……126
分散処理 ……150
並列処理 ……158
ベクタ・テーブル……70
ポインタ……85,122
妨害波 ……113
ボード・マイコン ……48,72,153
ポーリング……67
ボディ・ランゲージ……53

【マ　行】

マージン……32
マイクロコンピュータ……11
マイクロプログラム方式 ……134
マイクロプロセッサ……11

マイコン技術者の守備範囲……11
マイコン・システム……11
マイコン・システム技術者……5,11
マイコン・システムの3形態……46
マクロ・コール……82
マスタ……68,152
まとめ……15
マルチタスク……54
マルチプロセッサ……11
マン-マシン・インターフェース……39,54
ミクロン・ルール……95,149
三つのトレードオフ……16
メガセル……137
メガセル内蔵ASIC……136
メニュー方式……54
メモリ……95
メモリ空間……58
メモリ・マップ……72
モジュール分割……84,111
モデリング技術……20
モデル・チェンジ……15,128
モニタ……80

【ヤ　行】

柔らかな頭……5
ユーザーズ・マニュアル……127
ユーザ・プログラムIC……104

【ラ　行】

ラッチ……40
ラッチ・パルス……59
リアルタイム・システム……71
離散化……106
リセット……77,80,111,139,156
リファレンス・マニュアル……128
量子化……107
量子化誤差……107
リング・バッファ……85
ルックアヘッド・キャリ……36
論理回路……32
論理検証……167
論理シミュレーション……168

【ワ　行】

割り込み……69
ワンパターン処理……83

【欧　字】

【A】 A-D変換……44,105,153
AC ライン……75
ASIC……119,129,162
ASSP……88,132,143
【B】 Basic……60
Basic コンパイラ……62
BIOS 割り込み……69
【C】 C言語……63
CAD……166
CPU……11,134,138
CPU コア……138
CRT……54
CS……169
【D】 D ラッチ……59
D-A コンバータ……59
D-A 変換……105
DOS……80
DOS 割り込み……69
DSP……135
【E】 EMC……113
EMC 対策部品……115
EPROM……78,155
ES……168
EWS……166
【F】 FIFO……40
FIFO バッファ……85
【G】 G ラッチ……59
【H】 HMD……55
【I】 ID 番号……156
ImPP……160
I/O……11
I/O 空間……58
【L】 LCA……102
LCD……54
LED……54,75
【M】 MPU……11
MS-DOS……95
【O】 Occam……159
OJT……18
OP アンプ……43
【P】 PIO……93
PLD……98
PSD……100
【R】 RAM……40,96
ROM……40,90,96
ROM 化……98

ROM BIOS ……71
RS-232-C ……111,151,156
【S】 SE ……12,39
SYMDEB ……70
【T】 TTL ……32,132
【U】 UART ……92
USART ……88
【V】 VRAM ……71
VR ……56

【数　字】

1チップ・マイコン ……96,155
1の補数 ……38
2の補数 ……38
3ステート入力 ……59
3ステート・バッファ ……59
8251 ……88,92
8253 ……88
8255 ……88

◆著者のプロフィール

長嶋洋一（ながしまよういち）

1958年茨城県生まれ．京都大学理学部(原子核物理)卒業．河合楽器で電子関係の研究開発エンジニアとしてスキルアップし，1991年ASL(Art & Science Laboratory)長嶋技術士事務所を開設，1993年独立開業．コンサルティング・エンジニアとして，コンピュータ・エレクトロニクス分野での技術指導や人材育成事業に従事するとともに，音楽情報処理(Computer Music)の研究者・作曲家としても活躍中．イメージ情報科学研究所研究員，京都芸術短期大学・神戸山手女子短期大学・国立長野高専講師，情報処理学会・人工知能学会・IEEE・ICMA・日本技術士会等各会員，Nifty-Serve SubSysOp(FMIDI)，技術士(情報処理・電気電子)．

Email：nagasm@hamamatsu-pc.ac.jp

Nifty：NBD03033

著者：「マイコン技術者スキルアップ事典」(CQ出版)

「研究技術開発成果促進マニュアル」(共著・アーバンプロデュース)

「音楽情報処理の技術的基盤」(共著・文部省科学研究費総合研究調査報告書)

趣味：音楽全般(「広く深く」がモットー)

マイコン技術者スキルアップ事典

定価は表四に表示してあります

1992年8月15日　初版発行
1994年7月10日　第2版発行

著者　長嶋洋一
発行人　神戸一夫
発行所　CQ出版株式会社
〒170 東京都豊島区巣鴨1—14—2
☎03-5395-2122(出版部)
☎03-5395-2141(営業部)
振替　00100-7-10665

写植／みやこワードシステム　印刷・製本／図印刷　Printed in Japan

マイコン技術者に
必要な技術をハード中心に整理．エンジニアの仕事地図や理想像も示した，ユニークな事典です!!

CQ出版社　定価1,650円（本体1,602円）
ISBN4-7898-3491-3 C3055 P1650E