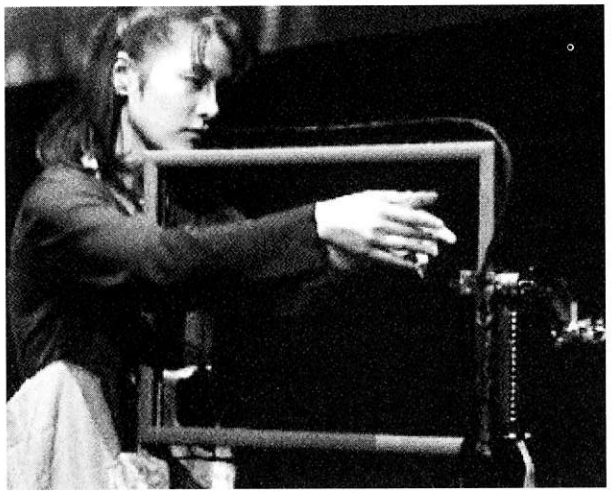
3 組み込みシステムの開発例 ❶一品料理

本章では、組み込みシステムの開発例として、もっとも小規模な「一品料理」の実例を紹介して、組み込み開発のステップに親しんでいきましょう。前章で紹介したように、組み込み業界のもっとも老舗である「システムハウス」と呼ばれる企業は、大企業や官公庁や大学・研究機関などからの「特注システム」として、量産でなく「一品料理」の特殊な装置や機器を受託開発しています。ハードウェア用に専用LSIを製造するような選択肢はありませんから、既存の電子部品を活用したシステム設計、そしてオリジナルCPUソフトウェアの開発により、システムを実現することになります。

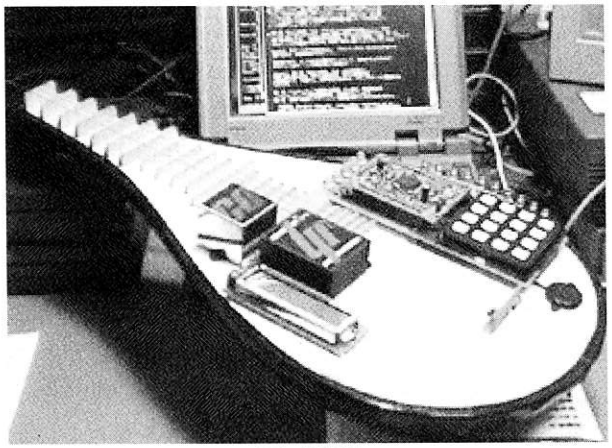
3-1
ターゲット例：「新しいインターフェース（楽器）」

筆者はコンサルタント（技術士）として、企業の技術・経営指導などを行い、時には依頼されて、一品料理の組み込みシステムを試作・開発しています。また研究者/作曲家として、メディアアート関係の特殊システムも開発しています。**図3-1**の（A）は「光の弦」、（B）は「超琵琶（Hyper-Pipa）」と名付けた、いずれも世界に1台、筆者オリジナルの「楽器」です。「光の弦」は縦横16本の光ビームセンサの「見えない絃」をハープの弦のように弾くことで、また「超琵琶」は3次元加速度センサとジャイロ（角加速度）センサが本体の運動状態を検出し、演奏情報としてコンピュータに送ることができます。

このような新しい組み込みシステムの開発例として、本章では、**図3-2**のような2種類のインターフェースシステムの開発事例を紹介します。**図3-2**（A）は「笙ブレスセンサ」、（B）は「筋電センサ」と呼んでいるもので、いずれも、人間の生体情報・身体情報をコンピュータシステムに取り込むための特製システムです。異なるテーマでの組み込みシステム開発において、手法として共通する部分をいかに活用しているか、という事例としても興味深いので、両者の相違点と共通点にも注目して下さい。



（A）光の弦



（B）超琵琶

図3-1　オリジナル楽器の例

(A) 笙ブレスセンサ　　(B) 筋電センサ

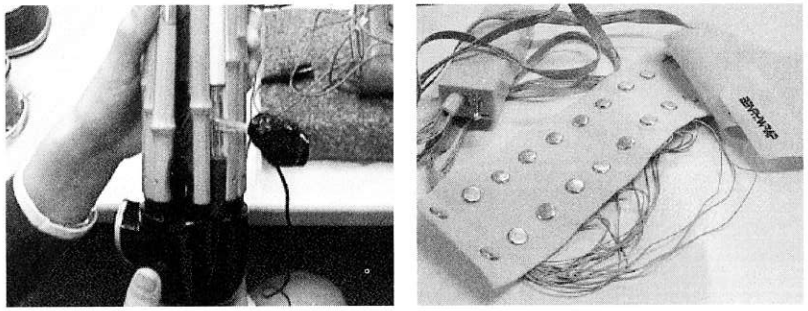


図3-2　紹介する２つのシステム

3-2
企画・設計フェーズ

図3-2（A）の「笙ブレスセンサ」は、雅楽の伝統楽器「笙」の演奏家・作曲家の東野珠実氏から開発を依頼されたプロジェクトです。一般に、小規模な組み込みシステム開発では、テーマや目的からシステム仕様を検討する企画の段階が、同時に設計作業となります。この例では、センサとして検出したいのは「笙を演奏する人間の呼気と吸気」の連続量です。工作機械などと違い、人間の呼気・吸気の範囲、それも「笙」という繊細な楽器に与えられる空気圧の場合には、過大な圧力による破壊の心配はありません。機能仕様としては、笙を演奏する呼吸情報のセンシング出力で映像や音響をライブ制御するために「MIDI規格」という情報で出力すること、センサのデータ分解能は７ビット精度（128段階）、時間的な分解能は「毎秒50-100回程度」と規定しました。このように、必要とされる機能仕様を明確に相談するなかで、使用するハードやソフトについても、既存の慣れたものの活用を検討していきます。

図3-2（B）の「筋電センサ」は、アナログ回路の専門家である照岡正樹氏との共同研究プロジェクトとして設計開発を進めました。ここでの機能仕様としては、両腕それぞれのベルトに８系統の筋電情報を検出するセンサを並べること、バッテリにより小型軽量で携帯できること、出力情報はMIDIとする、などの条件を設定しました。この例ではMIDIですが、一般にRS232C、USB、IEEE1394、GP-IB、イーサネット、等の国際汎用規格を利用することで、システムと接続する周辺機器との明確な「境界」を設定する意義は重要です。専用開発した機器と他の機器を接続して、もし何かトラブルがあった場合に、その原因がどちらにあるか切り分ける時に、汎用規格であれば「信号チェック」「情報の確認」なども容易になります。

図3-3は、「企画・設計フェーズ」を含む、このような小規模な組み込みシステムの開発プロジェクトの古典的なフローで、ウォーターフォール型と呼ばれています。特徴としては、

- ハードウェアには市販の汎用部品のみを使用する
- 回路規模、ソフトウェア規模がそれほど大きくない
- 外部インターフェースには汎用規格を使用する

というような典型例となります。それぞれのフェーズは比較的短期間であり、全体設計が終了するとハード開発に、ハードが出来たらソフト開発に、これが終了したら全体のデバッグとテスト（検証）、と順次進めて完成、となります。汎用部品を使用することで、部品の開発費用だけでなく「部品そのものの実験・開発」というプロセスも省略できるメリットがあります。

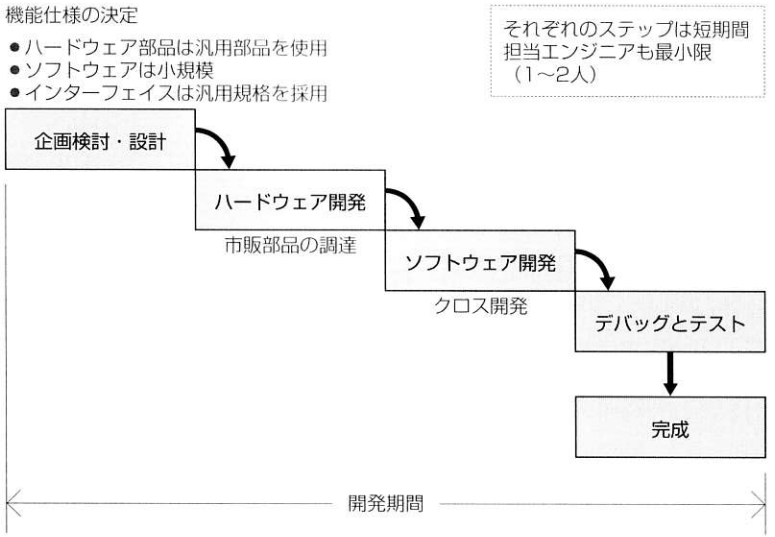


図3-3　ウォーターフォール型のシステム開発フロー

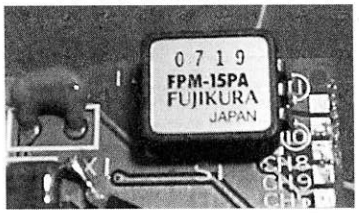
3-3
ハードウェア開発フェーズ

企画・設計フェーズを受けた「ハードウェア開発フェーズ」では、まずは具体的な回路設計からスタートします。**図3-2**（A）の「笙ブレスセンサ」システムの場合には、

- 笙の「空気溜まり」内部の圧力を検出する気体圧力センサ
- この気体圧力センサの出力を電圧に変換するアナログ回路
- このアナログ電圧をディジタルに変換してMIDI出力するマイコン回路

というような、大きく3つのハードウェア要素があります。センサそのものを開発するのは開発期間・開発コストの両方とも事実上不可能なので、専業メーカの量産製品のセンサを「部品」として利用する作戦になります。**図3-4**（A）は秋葉原の秋月電子で入手した「空気圧センサ」キットに使われていたフジクラ製の気圧センサで、プラスマイナス2気圧、という仕様も十分なものでした。このキットに付属していた回路図を参考に改造して、過去に実績のあるOPアンプによる増幅回路を組み合わせ、**図3-4**（B）のような回路を設計して実験し、予備的な試作で動作を確認しました。「笙」の17本の竹のうち、発音用リードのない2本の装飾用の竹の片方をパイプに取り替えて、**図3-5**のように取り付けます。

（A）秋葉原で入手した
フジクラの空気圧センサキット



（B）キットの付属回路を改造した
オリジナル信号処理回路（アナログ部分）

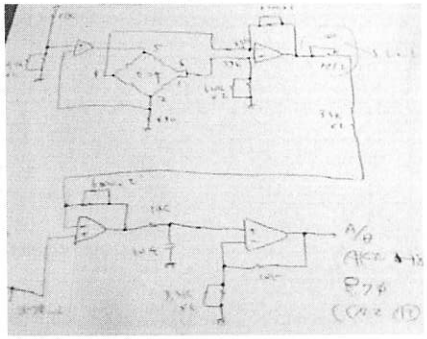


図3-4　気圧センサとその応用回路

図3-2（B）の「筋電センサ」の場合には、

- センサ電極を埋め込んだ、腕に巻く専用ベルト
- 環境ノイズと分離して電極の微弱信号を増幅するフロントエンド回路
- このアナログ電圧をディジタルに変換してMIDI出力するマイコン回路

という要素が必要です。ちなみに同等の性能の医療用機器は、数百万円もする専用装置であり、このセンサの手軽なキットというのは存在しません。**図3-6**は、照岡氏の設計したセンサ・フロントエンド回路で、これを片腕あたり8チャンネル、計16チャンネル分を製作しました。**図3-7**はその実験と製作の模様で、専用基板でなくユニバーサル基板（2.54mm間隔）に高密度空中配線しています。電極は純銀円盤を介護用の伸縮ベルトに取り付けました。

図3-5　笙ブレスセンサの使用状態

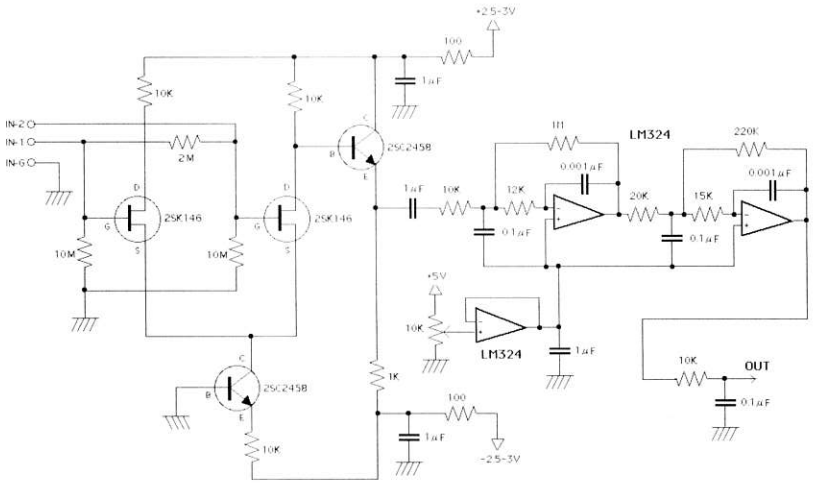
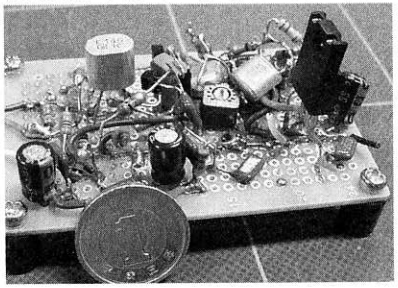
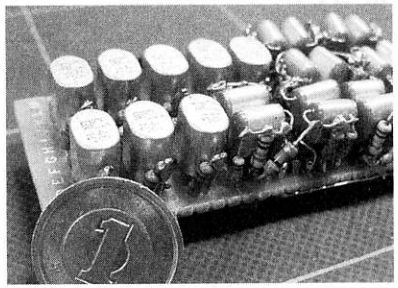


図3-6　筋電センサ回路のフロントエンド部分（1チャンネル分のみ）

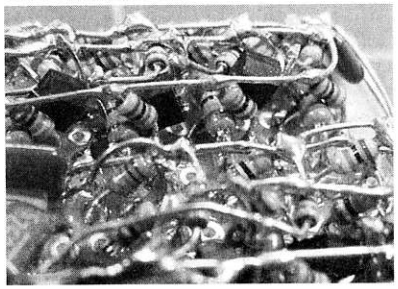
(A) 実験中の1チャンネル分のブラック回路



(B) 完成した8チャンネル分の回路基板



(C) 空中配線のアップ



(D) ケースに入れてバッテリも格納

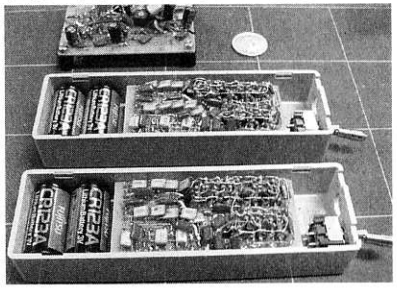
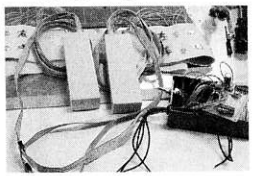
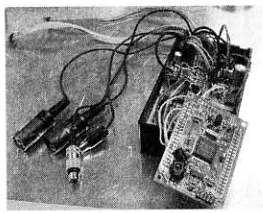


図3-7　筋電センサ部分の開発の様子

まったく異なった生体情報をセンシングするシステムですが、アナログ電圧として入力しMIDI出力のためにディジタル処理する1チップマイコン回路部分は、両方のシステムに共通します。**図3-8**は「筋電センサ」のフロントエンド回路の小型ボックスと接続するマイコン部分の開発例で、「笙ブレスセンサ」もほぼ同じように製作しました。このように、過去に開発した実績のある部分を活用することは、開発の手間を省

(A) センサ部分とケーブルで接続する小型ケース





(B) 秋月電子マイコンカード「AKI-H8」

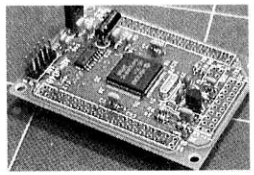


図3-8　マイコンを組み込んでハード完成

くだけでなく、信頼性や完成度の点で重要です。ここまでのハードウェアの簡単なチェックはテスタやオシロスコープなどの装置を用いて行い、本格的なテストは、ソフトウェアの完成後の「デバッグ」段階がメインとなります。

3-4
ソフトウェア開発フェーズ

ソフトウェア開発の段階で、大部分の組み込みシステムに共通する特徴は、「クロス環境による開発」の手法です。**図3-9**（A）のように、汎用パソコンのアプリケーションソフトやエンタープライズシステムの業務ソフトの場合には、「ソフトを開発する環境」＝「ソフトを実行する環境」という関係があります。ところが組み込みシステムの場合には、コスト制約により、システムに組み込まれるコンピュータは小型で安価なCPUとなり、そのCPUに開発環境を走らせた上で自分のプログラムを開発する、というのはほぼ不可能です。このため、**図3-9**（B）のように、

- パソコンなどの汎用開発環境でターゲットCPUのプログラムを試作
- この試作ソフトを種々の方法でターゲットシステム上のCPUに走らせる
- 完成したソフトウェアはROM化してターゲットシステムに搭載

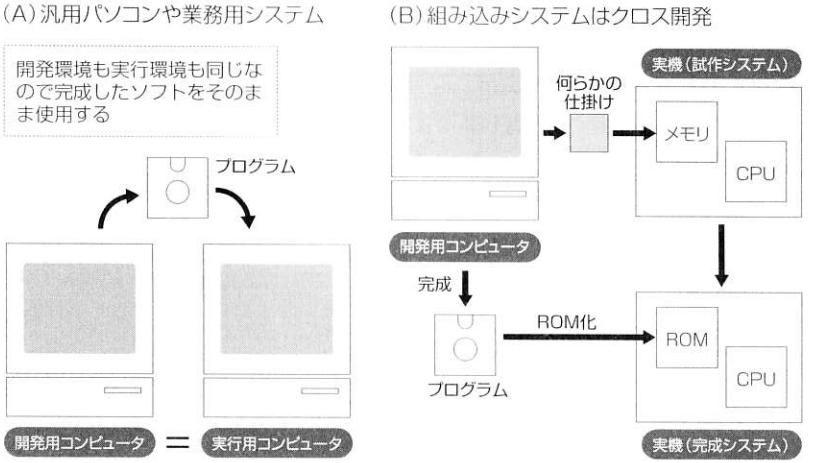


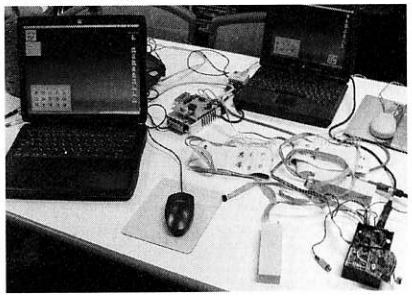
図3-9　クロスプロットフォーム開発環境

というような、ちょっと面倒な方法をとらないといけません。この「環境」をプラットフォームと呼ぶので、組み込みの多くは「クロスプラットフォーム開発」となります。

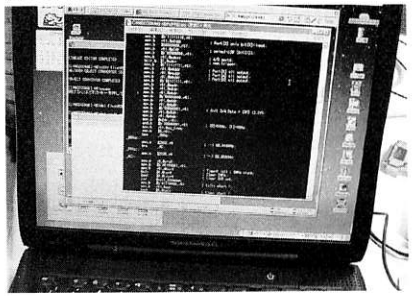
この様子をもう少し具体的に解説します。**図3-10**は、**図3-2**（B）「筋電センサ」のソフトウェア開発フェーズの風景です。**図3-10**（A）のように、開発デスクにはターゲットシステムのCPUボードと接続されたパソコンがあり、プログラムはこのパソコン上で（B）のように開発します。この例では「アセンブラ」という言語で開発していて、テスト用の出力MIDI情報は、もう1台のパソコンに検証用モニタのプログラムを実行させて確認しています。なお、開発用言語については、第4章で検討します。

クロスではない開発、つまり開発用コンピュータと実機とが同じ場合には、完成したプログラムを実機にインストールすれば開発完了ですが、組み込みシステムの場合には、**図3-9**（B）の「何らかの仕掛け」が必要になります。つまり、開発するターゲットシステムでは開発環境とはCPUが異なるだけでなく、開発したプログラムをターゲットシステム（開発中の実験機、あるいは最終的な実機）のCPUにどう読ませるのか、という仕組みが必要です。分類すると

（A）開発の全景



（B）アセンブラでプログラミング



（C）資料やメモ類

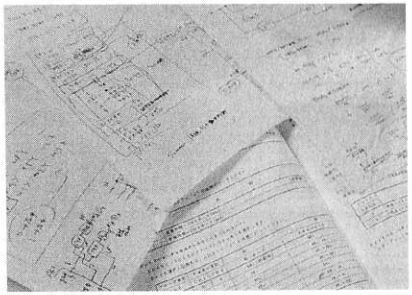


図3-10　ソフトウェア開発の風景

（1）実機にOS（Operationg System）を導入してロードする
（2）イン・サーキット・エミュレータ（ICE）という装置を使う
（3）ROMをソケットにしてROMライタで書き込んだEPROMを使う
（4）ROMソケットにROMエミュレータという装置を接続する
（5）「モニタ」という補助プログラムを使う
（6）フラッシュROMに書き込み回路を経由して書き込む

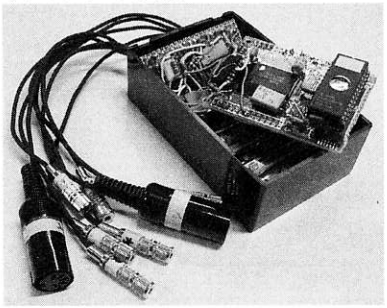
など、色々の方法があります。このうち分類（1）はパソコンなどクロスでない開発と同じことです。分類（2）は第5章で紹介しますので、残りの分類（3）（4）（5）（6）について、解説します。このあたりが組み込みに関するソフトウェア技術の「敷居の高い」部分ですが、ここを理解することは、組み込みの本質を理解するために重要です。

3-5
クロス環境による開発

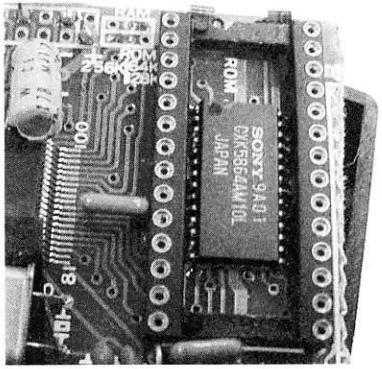
第1章の**図1-5**にあったように、組み込みシステムのプログラムというのは、基本的にシステム内のROMに格納されて、電源ONやリセットとともに、そのスタート部分から順に実行していきます。小規模な組み込みシステムでは、ハードウェアの心臓部に、それ自体で完結したコンピュータシステムである「ボードマイコン」を活用します。**図3-11**（A）はその一例で、カードサイズの基板上に、CPU、（B）のROMソケット、そのソケットの下には小型パッケージのRAMが搭載されています。このROMソケットに、（C）のようなメモリ（＝ROM）を差し込むことで、そのプログラムがCPUにより実行されます。もともとROMというのは「読み出し専用メモリ」なので書き込みできませんが、「ROMライタ」という専用装置を使って、電気的に書き込み（プログラム）できるので、PROM（Programmable ROM）と言います。

PROMの多くは（C）のようにパッケージ中央に「窓」があってチップが見え、**図3-12**（A）のような「ROMイレーサ」という専用装置で紫外線を照射すると、データを消去できます。つまり消去可能なPROM＝EPROM（Erasable PROM）と呼びます。紫外線で消去するタイプは特に「UV-EPROM」という名前です。

(A) システムの全景



(B) ROMソケット



(C) UV-EPROM

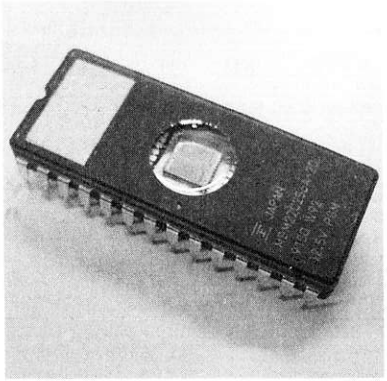
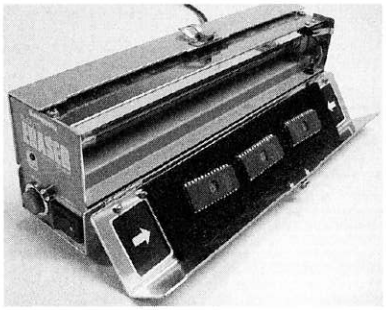
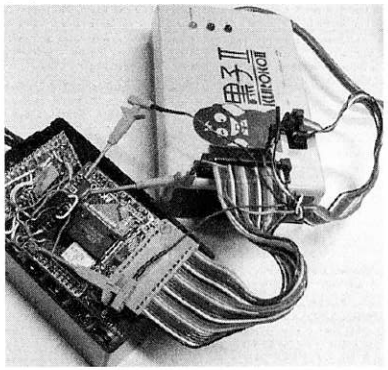


図3-11
EPROMを使ったシステム

(A) ROMイレーサ装置



(B) ROMエミュレータ装置



(C) プローブをROMソケットに接続

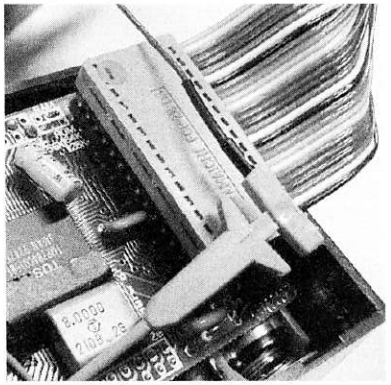
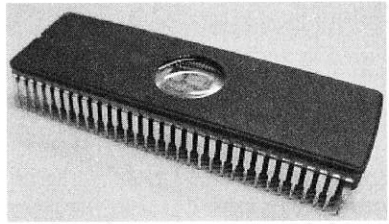


図3-12
ROMイレーサとROMエミュレータ

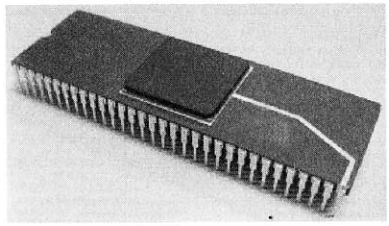
このEPROMを用いた組み込み開発は、小規模なシステムだけではなく、第5章で紹介するような「量産システム」でも利用されます。**図3-13**(A)は、1チップでCPU・ROM・RAM・I/Oまで搭載した「1チップマイコン」の、開発段階で使うEPROMタイプのもので、ちゃんと消去用の窓があります。これで開発完了すると、(B)のように半導体メーカが「マスクROM」という読み出し専用メモリ化したサンプルチップを高価なセラミックパッケージで試作し、テストがOKになると(C)のように安価なプラスチックパッケージ化して大量生産します。

分類(3)の、EPROMを使ったクロス開発の手順はもっとも基本となるので、ここでは**図3-14**で詳しく追いかけていきましょう。

(A)開発用EPROM版(セラミックパッケージ)



(B)評価用サンプル(セラミック)



(C)量産用(プラスチック)

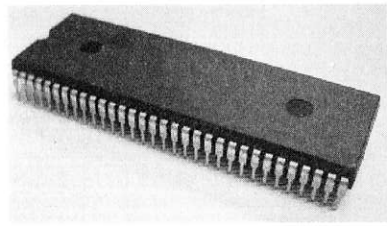


図3-13
1チップマイコンのパッケージのいろいろ

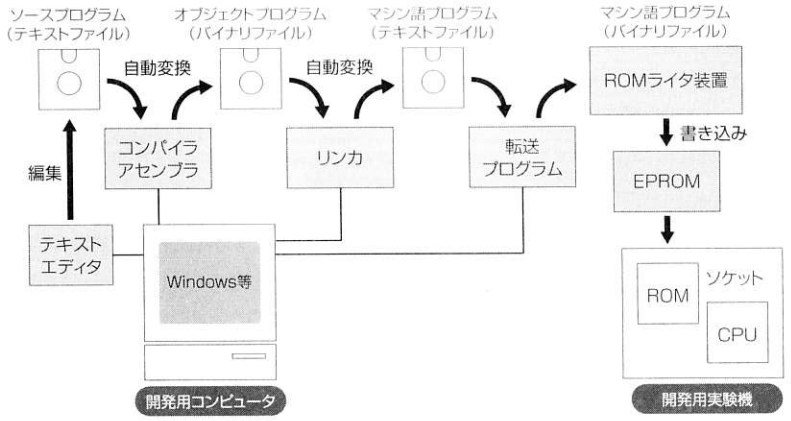
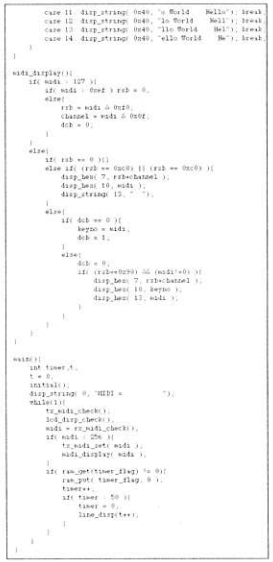


図3-14　EPROMを使ったクロス開発の流れ

プログラマは開発ツール（一般にはパソコン上、つまりWindowsなどの汎用の環境）でプログラミングを行います。ここで使うツールは「テキストエディタ」であり、小規模であればメモ帳でもワープロでも構いません。このソースプログラムは、人間が読める「C言語」や「アセンブラ」で記述された、テキストファイルです（**図3-15**）。

Cプログラムの例



アセンブラプログラムの例

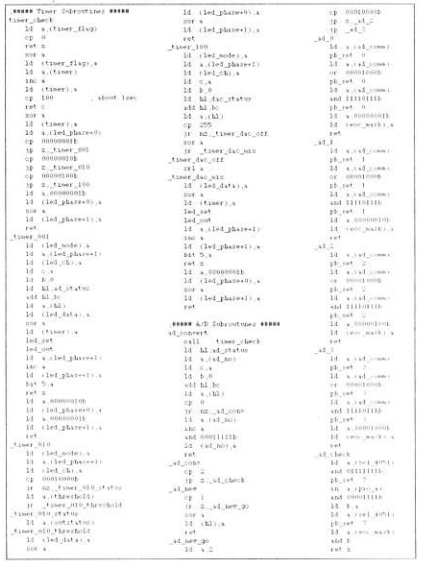


図3-15　ソースプログラムの例

これを、同じ開発用コンピュータの「コンパイラ」あるいは「アセンブラ」というツールにより、オブジェクトプログラムというバイナリファイルに自動変換します。ここでプログラムに形式的なエラー（バグ）があれば警告されます。このオブジェクトプログラムというのはいわば「ソフトウェア部品」であり、さらに「リンカ」というツールによって、最終的にターゲットシステムのCPUが動作する「マシン語（機械語）」プログラムになります。これは多くの場合、インテルHEX形式とかモトローラS形式、などと呼ばれるテキストファイルの形態をとり、転送ツールによって「ROMライタ」装置に転送します。

ROMライタでは、転送されたマシン語プログラムをEPROMに書き込み、これをターゲットシステム（開発用実験機）のROMソケットに

差し込んで、「せーの！」と電源を入れて、プログラムが正しければシステムは所望の動作をして完成、ということになります。ただし最初のプログラムから完成動作する事はほとんどありませんから、ハテ動かないとすればどこだろう…とチェックしていきます。これがいわゆる「デバッグ」という作業で、このループを完成するまで繰返す、ということになります。小規模な組み込みシステム、と言っても、CPUのソースプログラムは数百行から数千行にもなりますから、いきなり最終版までプログラミングするのではなくて、最初はハードウェアが正しく製作できていることを確認する基本的な処理だけを実行させて、本来の機能を実現する処理を少しずつ順に付加していく、というような流れです。

このようなデバッグループを回してソフトウェア開発を進める場合に、効率の点でもっとも問題となるのは、いちいちEPROMに書き込んでは消去する、という部分です。そこで分類（4）の方式として、**図3-12**（B）の「ROMエミュレータ」という専用装置も活用します。これは同図（C）のようにROMソケットに挿入するプローブを持ち、**図3-16**のフローのように開発用コンピュータと接続します。この装置は、内部的にRAMとして持っている領域にCPUプログラムをロードし、これをデバッグ時にはプローブ経由でターゲットシステムから「ROMのように見える」システムです。この結果、いちいちソケットにEPROMを抜き差ししたり消去する手間がなくなり、何度でもスムースにデバッグのループを回せることになります。

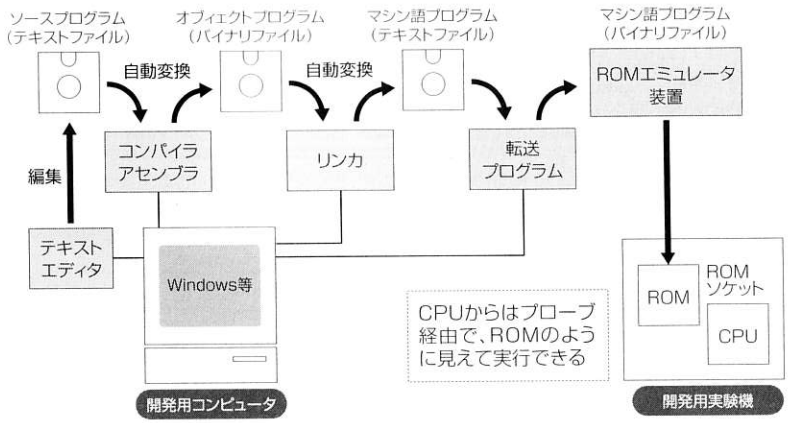


図3-16　ROMエミュレータを使った開発

3-6
モニタによる開発とデバッグ

プログラムを開発しては「せーの！」で試してみる、という場当たり的な手法には限界があり、一般的なデバッグのためには「動作を追跡する」（トレース）など、より高度な支援が必要です。第5章で紹介する「ICE」はこのための専用装置ですが、ここでは分類（5）の、「モニタ」を活用した開発とデバッグの手法について紹介します。モニタとは、

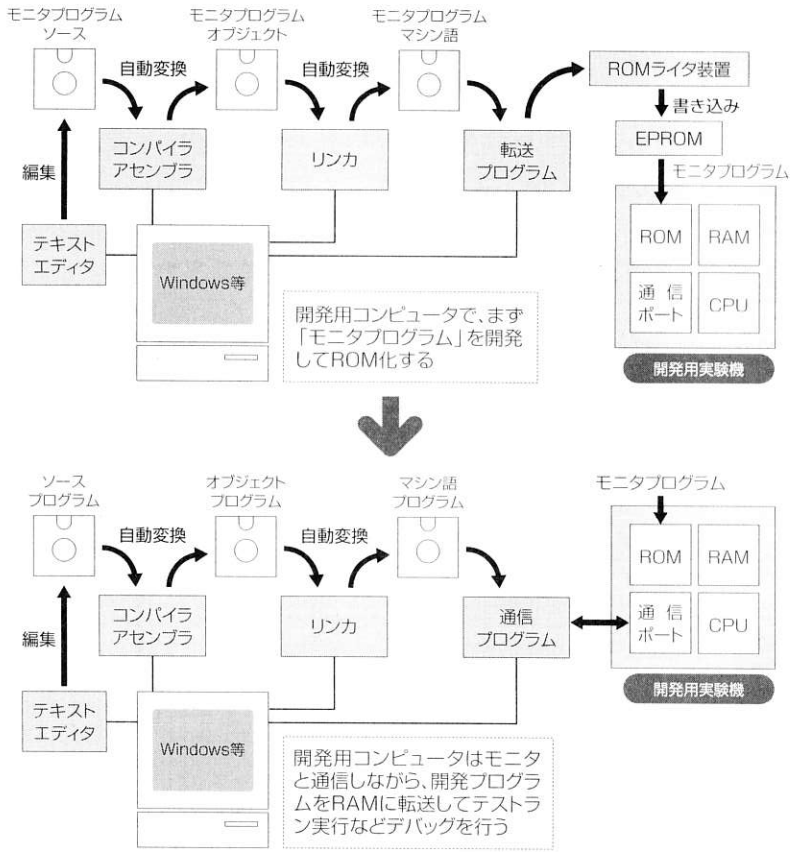


図3-17　モニタを使った開発

図3-17のように、ターゲットシステムの本来の機能とは別に、システム自体が「ある種の完結した動作」をするための簡易プログラムで、いわば「もっとも簡素化されたOS」というようなソフトです。

モニタプログラムは、組み込みシステムに求められる本来のCPU動作とは別に、開発やデバッグのための機能で、開発用ホストコンピュータと通信するためのインターフェースなども別途に必要となります。開発はまず、ターゲットの組み込みシステムに外部通信機能があればこれを活用し、使用する1チップCPUがシリアル通信ポートを内蔵していれば活用する、というように、最小限のコストでモニタとしての外部通信ルートを設計します。そしてモニタプログラム自身をまず開発してROM化します。ここまでは、分類（3）のEPROMライタや分類（4）のROMエミュレータの手法によります。CPUボードによっては、対応したモニタが付属していたり、別売製品として購入することができます。

モニタプログラムがターゲットマシン（開発対象の組み込みシステム）で走ったところで、図3-17下段のように、ターゲットマシンを単独で動作させながら、開発用ホストコンピュータの通信プログラムとの間でやりとりして、開発とデバッグを進めます。モニタプログラムの機能としては、

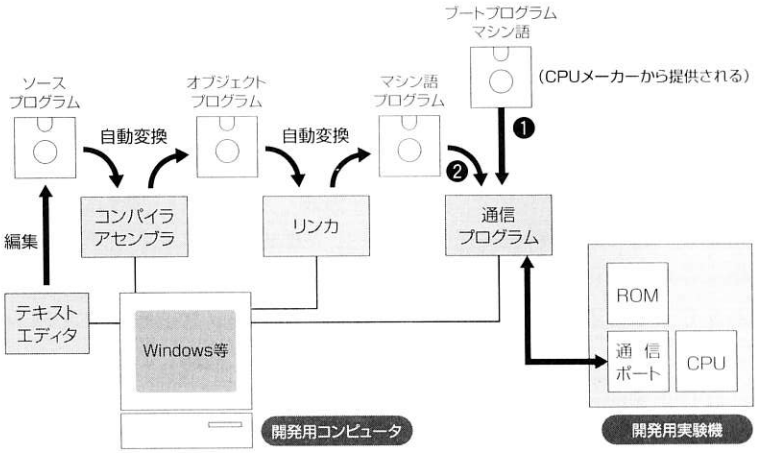
- ホストから指定されたRAMアドレスにデータを書き込む
- 指定されたRAMアドレスのデータを読み出してホストに返す
- 指定されたアドレスにジャンプする（テスト実行）
- CPU内の指定されたレジスタ（内部情報）をホストに返す
- 指定された一定領域のメモリデータをまとめて表示（ダンプ）
- 指定されたI/Oポートの状態を設定あるいはモニタする

などがあります。これをホスト側に用意する開発用ソフトウェアと組み合わせると、システムの動作についてかなり詳細に追跡したデバッグが可能となります。

開発対象システムの目的のCPUプログラムについては、開発段階ではモニタ経由でシステム内のRAMに転送します。このRAMアドレスにジャンプするようにモニタに指令すれば、実験プログラムをテスト実行できます。開発完了、となった場合、多くの場合にはモニタを外して、プログラムをROMエリアに移動してアドレスが変更になるために、この調整をする「ROM化」という作業が必要になりますが、本書ではこ

の部分の詳細は省略します。この分類（5）の「モニタ」による組み込み開発の手法は、これまで長い歴史があり、まだまだ現役です。通信機能を仕様として持つシステムであれば、開発で使ったモニタを「簡単なOS」のように捉えて、システムのプログラムの一部としてそのまま搭載・利用することもあります。

この一方で、最近では1チップCPUの内蔵ROMとして、フラッシュROMを採用することが多くなり、分類（6）の手法も登場してきました。これは**図3-18**のように、オンボードのフラッシュメモリやCPU内のオンチップフラッシュROMに対して、まず、特殊な「書き込みモード」に設定して、ホストからブートプログラムをロードします。これに続いて、ホストから本来の実機プログラムをロードした後に、フラッシュメモリを通常モードに戻します。すると、電源ONやリセットの時に、CPUはこのフラッシュROMの実機プログラム領域からスタートすることで、分類（3）と同様に開発することができます。モニタのようなデバッグ機能はありませんが、ホストと接続したままで開発できるメリットがあります。



❶まず、フラッシュ書き込みモードに設定して、ホストからブートプログラムをロードする。
❷引き続き、このブートプログラムを利用して、実機ソフトをフラッシュROMにロードする。
❸通常モードに戻すと、電源ONやリセットにより、CPUはロードされた実機プログラムからスタートする。

図3-18　フラッシュメモリを活用した開発

3-7
検証ツール：オシロスコープ

クロス環境によるソフトウェアの開発とデバッグについて紹介してきましたが、組み込みシステムではソフトウェアと同時に、ハードウェアもオリジナル開発することが多く、ハードウェアのデバッグ（テストや検証）のためのツールも必要です。ここでは小規模な組み込み開発で活躍する「オシロスコープ」を紹介しましょう。オシロスコープとは**図3-19**のような測定器で、基本的には、周期的に変化するアナログ電圧を計測表示する装置です。ブラウン管を使ったアナログタイプだけではなく、最近では液晶モニタを使ったディジタルタイプ、パソコンに外付けするプローブBOXとソフトがセットになったタイプもあります。

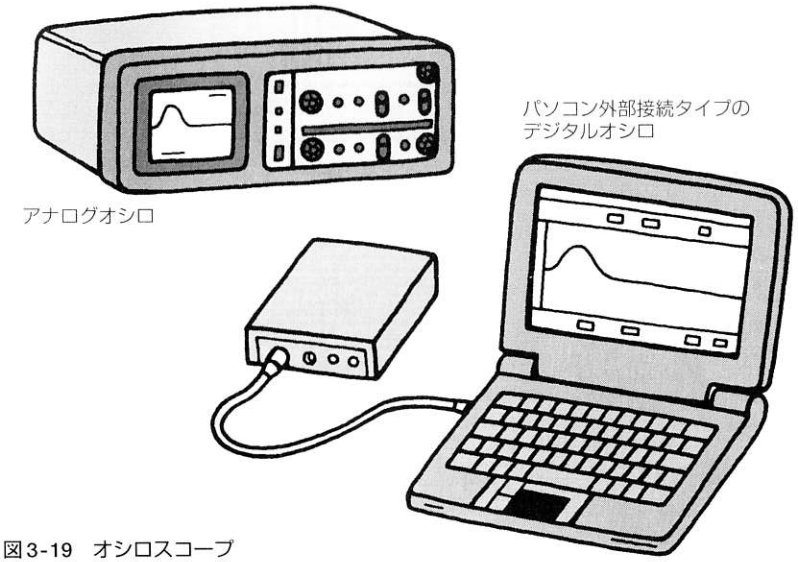


図3-19　オシロスコープ

図3-20は、本章で紹介した**図3-2**（B）「筋電センサ」のセンシング出力をオシロによって表示した例で、上段は筋肉をリラックスさせている状態、下段は緊張状態にしています。筋肉を制御する神経パルスによ

る電位変化を皮膚表面の電極で検出しているのですが、周囲のノイズをキャンセルして綺麗なデータが得られました。**図3-21**は、このセンサを「楽器」として用いた作品のドイツ公演の模様です。また**図3-22**は、本章で紹介した**図3-2**（A）「笙ブレスセンサ」のドイツ公演での演奏風景です。無事に完成した組み込みシステムが、このように現実の世界でお仕事をしてくれる、という「手応え」は、エンジニア冥利に尽きるものです。

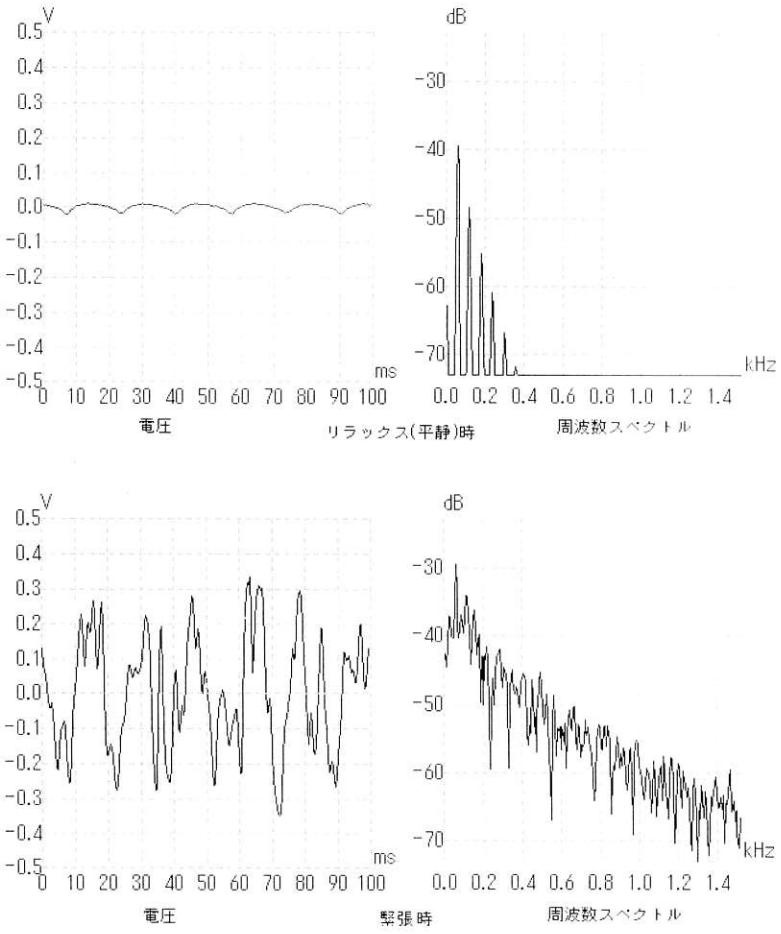


図3-20　筋電センサの出力信号の例

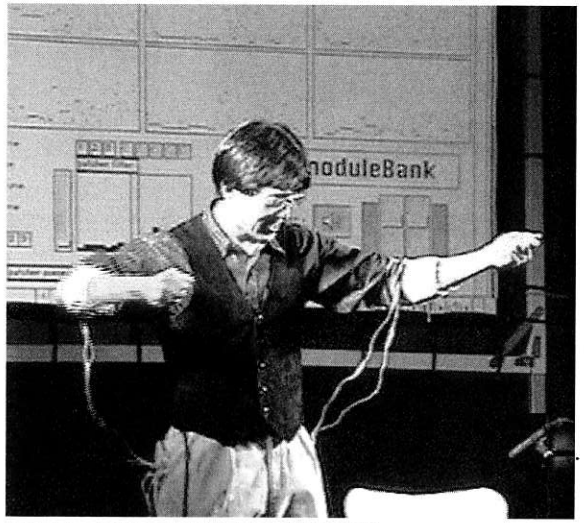


図3-21　筋電センサを用いた公演の模様

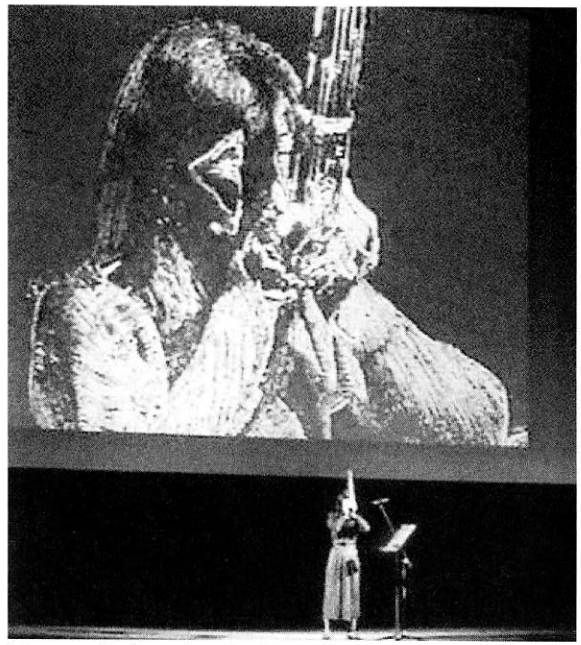


図3-22　笙ブレスセンサのドイツ公演での演奏風景

4 組み込みシステムの開発例 ❷ 専用装置

本章では、組み込みシステムの開発例として、１台きりの小規模な「一品料理」よりも高度な、計測用・産業用・製造用などの専用装置について紹介します。ここでも中心となるのは専業の「システムハウス」ですが、最近では大手の電機メーカー系列のエンジニアリング会社や、ソフトハウス業界の一部なども、参入してきています。システムの規模と複雑さが高度になっていく要請に対して、どのようなアプローチで組み込みを実現するのでしょう。

4-1 ロボットとカーエレクトロニクスの例

小規模な組み込みシステムでは、システム全体のアルゴリズム（ソフトウェア）を担当するコンピュータとして、単一の１チップマイコンを搭載したり、カードサイズの小型マイコンボードを利用する、という実

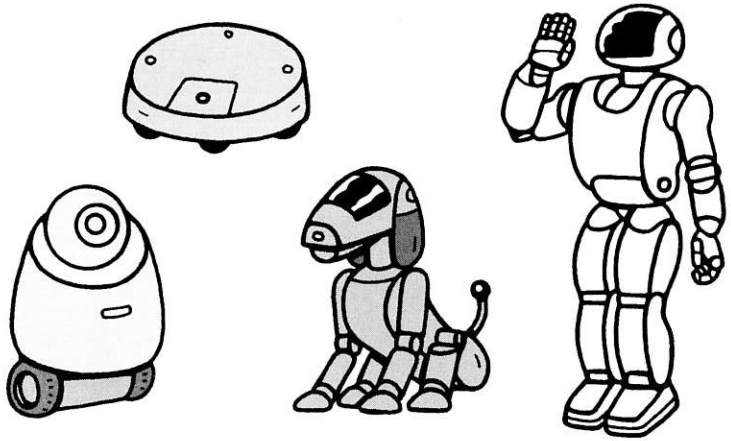


図4-1　いろいろなロボット

例を紹介してきました。より複雑な専用装置の場合、多くは複数のコンピュータが通信・連携する、というアプローチとなります。多数の分散コンピュータの統合システムの典型として最近注目を集めているのは、「ロボット」と「カーエレクトロニクス」の領域です。

図4-1のようなロボット、特に日本の人間型ロボットや動物型ロボットは世界の先端をリードしていますが、これは情報科学の立場から見ると、「高度な自律分散処理システム」の代表例です。ホンダのASIMOの場合を例にとると、26個の関節（運動の自由度）を持ち、**図4-2**のように、15年近い研究の成果として、スムースな２足歩行動作を実現してきたといいます。これに刺激されてか、最近では多くの研究所や大学でも２足歩行の自立ロボットを開発していますが、全てのロボットに共通しているのは、

- 多数のセンサとモータ群の情報をリアルタイム処理する・
- 個々の関節モジュール同志の連携によって自立歩行を実現する
- ロボット全体として10数個以上のコンピュータを使用する

という、まさに高度な組み込みシステムとなっていることです。バッテリまで背負って完全に自立歩行する軽量化のために、量産ベースの専用半導体などを開発するのは次の第５章のレベルとなりますが、生産工場の加工ロボットなどの場合は、発注する企業の現場に合わせて、システムハウスが１台から10台程度の規模で、専用装置として開発することも少なくありません。

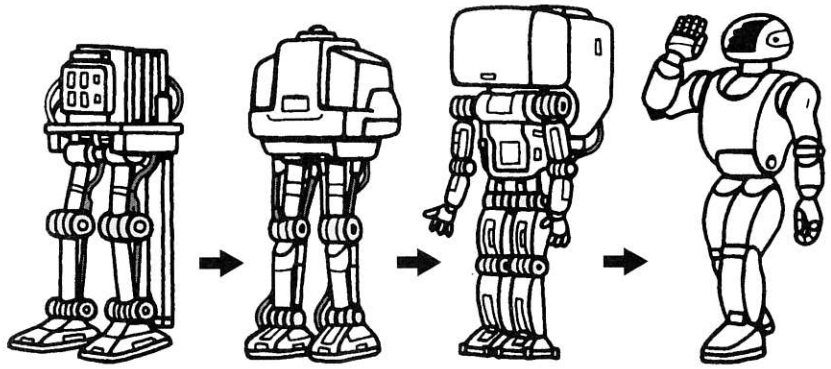


図4-2　２足歩行ロボットの進化

ロボットと並んで、現在の組み込み業界でホットな領域といえば、自動車のコンピュータ・ネットワーク化です（**図4-3**）。自動車部品のパーツは、自動車の走行・制御・管理・支援など、いくつもの重要な部分で活躍しますが、パーツ単体というわけではありません。センサがあれば、そのセンサ情報を検出して伝送するCPUとともに、ECU（電子制御ユニット、自動車ではこの用語が一般的）を構成します。つまり、自動車には、ブレーキ、エンジン、グリル、ドア、カーナビ、ETC、等々の機能ブロックごとに多数のコンピュータシステムが組み込まれていて、さらにそれらは高度に連携することで、1台の自動車としての全体システムを実現しています（この通信規格の標準化の話題は第8章で紹介します）。

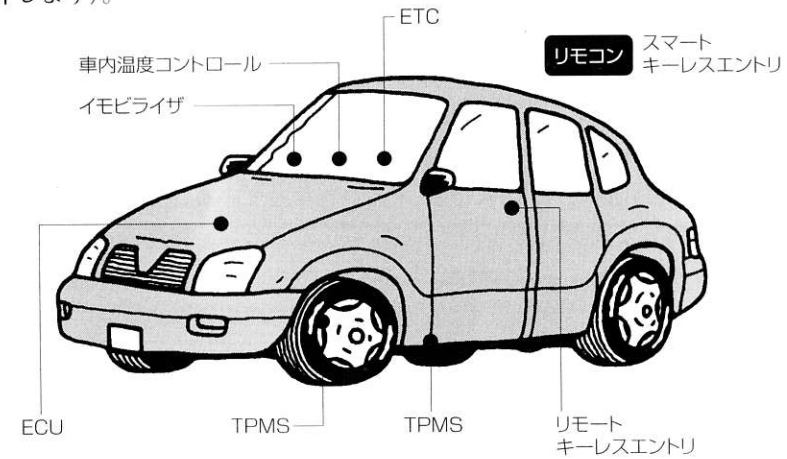


図4-3　カーエレクトロニクス部品の例

自動車に搭載されるCPUには、ノイズの環境が過酷であること、トラブルが人命に関わること、などの点で民生用とは違った信頼性が要求されるために、いくつかの専業メーカーが特別に車搭専用CPUを提供しています。しかし、これに加えて、カーナビやETCなどにより、自動車も情報化され、インターネットを含めた通信ネットワークに結び付こうとしています。この部分では家電や携帯を巻き込んだIT化の技術が活用できること、そして何より自動車に搭載されれば膨大な量産数量としてビジネス規模が大きいことから、組み込み業界は熱い視線でカーエレクトロニクスの行方を注目しています。

4-2
ターゲット例：「統合センシングシステム」

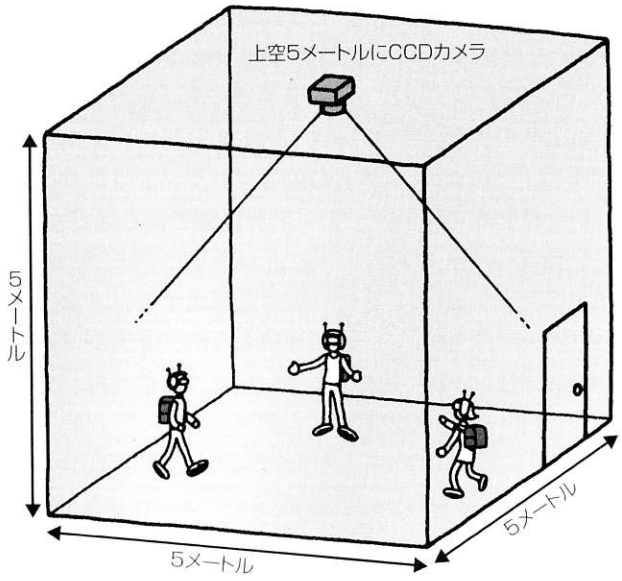
システムハウスが開発する専用システムでは、ロボットや自動車ほど多数のコンピュータを使わないまでも、複数のコンピュータの連携によってシステムを実現することは一般的な手法です。本章では、筆者が数年前にNTT研究所の依頼で開発した「統合センシングシステム」を例にして、実用的な組み込みシステムの開発プロジェクトについて紹介します。これは、NTTのメディア・アート美術館であるICC（Intercommunication Center）のビエンナーレ'97というイベントで、作家の前林明次氏の「Audible Distance」という作品を構成するシステムの一部として開発しました。この作品は、**図4-4**のように、タテヨコ高さ5メートルの真っ暗な空間に、3人の来場者が入って「うろうろ」する、という体験型のインスタレーション作品です。

ビエンナーレ'97の最終コンペに選ばれた10人の作家の中で唯一の日本人の前林氏のコンセプトは、立体音響ヘッドホンと3次元HMD（ヘッドマウント・ディスプレイ）によって聴覚と視覚を遮断された来場者が、バーチャル空間の中でお互いの「距離」を体験する、というものです。3次元音響生成、HMDでの3次元CG、というそれぞれの要素については、当時ようやく利用できるようになりましたが、このコンセプト全体を実現するための技術的な障壁は高く、ハリウッドで導入され始めたという数億円のシステムでも使わないとできないこと（ある審査員は「面白いが、実現するにはあと5年かかる」といったそうです）を、このプロジェクトでは2桁安く開発する、という挑戦になりました。

4-3
企画・検討フェーズ

図4-4のような「コンセプト」は企画段階のアイデアであり、プロジェクトは関係者のミーティングで、コスト制約の下で技術的にどのように実現するか、という企画検討のステップから始まります。作品のシス

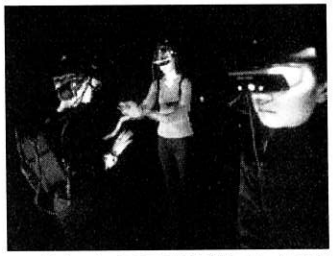
テム全体は、おおよそ**図4-5**のような機能に分割され、「統合センシングシステム」の部分（センサに関するワイヤレス装置３セットまで）を、



*タテヨコ5メートル四方の真っ暗な空間に3人の来場者が入ってうろうろする

来場者はそれぞれ、ヘッドホンをかけてヘッドマウントディスプレイ（HMD）をかけるのでお互いに何も見えず、何も聞こえない。

センサ帽子をかぶる。機材は背負ったバッグにより完全にワイヤレスとなって自由にうろうろできる



ヘッドホンからの空間音響には、他の2人の心臓の移動に同期したサウンドが聞こえてくる。HMDの3次元映像には他の2人の位置に対応した場所で心臓の鼓動に同期したCGが動くのが見える。これにより、3人の来場者は、バーチャルな「距離」を体験する。

図4-4　作品「Audible Distance」の概要

他と明確に分離しました。後でHMDの3D-CG部分（開発は日立製作所の担当）と統合した時のトラブルを避けるためにも、このような担当領域の明確な切り分けは重要です。

同時にこの作品を体験する３人の来場者のHMDには、それぞれ他の２人の位置に対応した3D-CGが表示され、ヘッドホンからは他の２人それぞれの位置に対応した３次元音響が鳴ります。従って、３人それぞれのHMDの映像（２チャンネル）とヘッドホンのサウンド（ステレオ）は、全て異なる計12チャンネルの無線で別個に送信する必要があります。センシングシステムとしては、タテヨコ５メートルの会場内での、３人の来場者の位置座標だけでなく、３人がそれぞれ、どの方向を向いているか、という「視線」の方向ベクトルも検出することで、頭を回転した時に、HMDの映像やサウンドを対応した方向に移動させる必要があります。これと同時に、CGとサウンドは心臓の鼓動と同期させるため、３人の心拍情報もそれぞれにセンシングします。

全体としての必須条件は、来場者がケーブルを引きずりながら歩くのは危険なので、完全にワイヤレスであることです。このため、全ての情報通信は無線で、さらに多数の機器とバッテリを背負うために、来場者のバッグに搭載する機器は「小型」「軽量」「低消費電力」という条件も加わります。無線で映像を受信してHMDに表示する機器と、無線でサウンドを受信してヘッドホンで聞く機器、の部分は市販製品をそのまま使うため、新規開発するセンシング装置に、この条件が強く要請されました。

組み込みシステムとして、コスト制約や機能仕様とともに、さらに必要となるのが時間的制約です。この例では、真っ暗な会場で、HMDで視線を遮られた来場者が「うろうろ」します。センシングシステムがうまく稼動すれば、他の人に近付いた時には、その方向でHMDに表示されるCGが大きく赤く変化し、心拍に同期するサウンドが大きくなって、衝突の可能性を警報できます。ところが、センシング速度が遅ければ、この警報が間に合わなくて、来場者が衝突して怪我をする危険性があります。つまり、自動車のシステムと同様に、このシステムには、時間的（反応速度）制約も厳しく加わっているのです。興味ある読者の皆さんは、ここで、どのような手法でこのシステムを実現できるか、クイズとして考えてみて下さい。

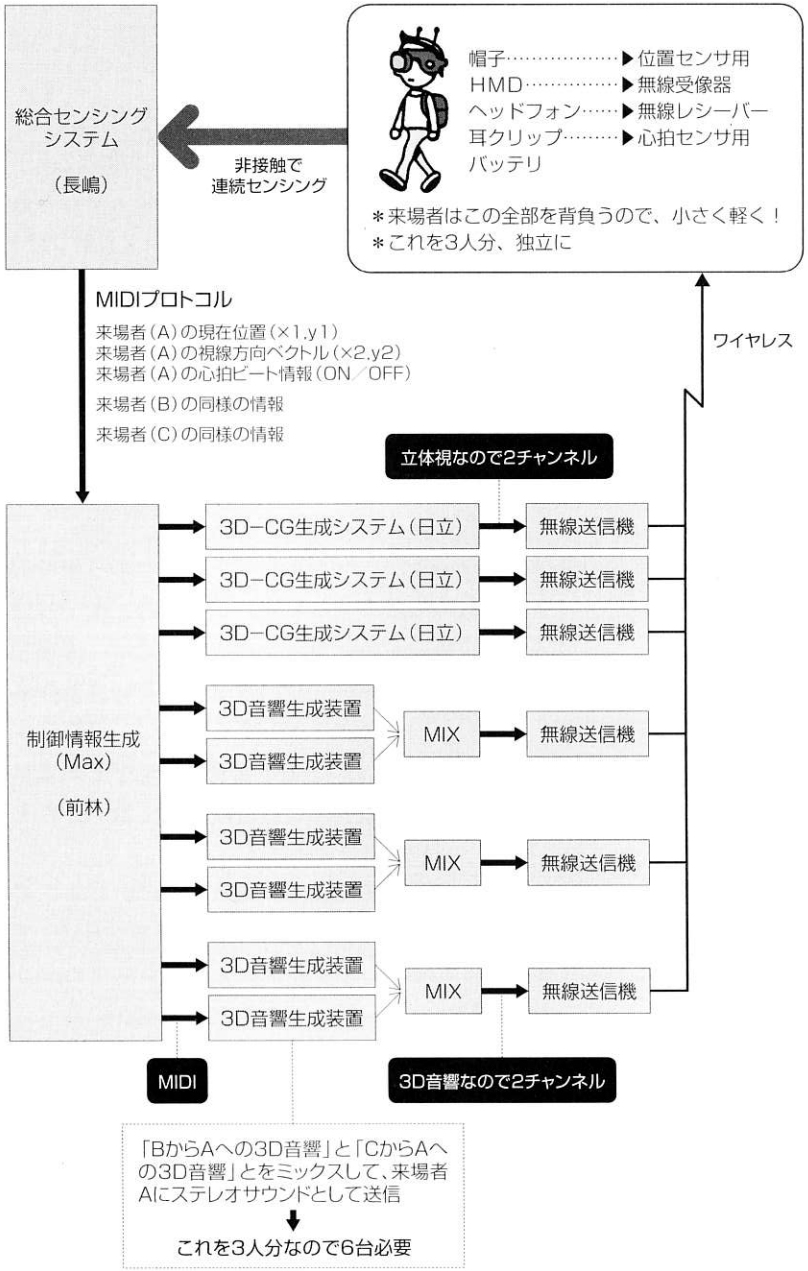


図4-5　「Audible Distance」のシステムブロック図

4-4
ハードウェア設計・開発フェーズ

コスト制約と機能仕様の条件だけであれば、基本的にはハードウェアを設計し、試作した後で、次にソフトウェアを設計・開発する、という流れとなります。ところがこの例では、時間的な条件もクリアする必要があるために、設計段階の中で同時に実験・試作を行い、機能だけでなく時間的制約を満たすことの検証も必要となります。「機能vsコストvsスケジュール」の「トレードオフ」なのですが、これは組み込みシステムの開発ではいつものことです。

いくつかの予備的な実験を経て、最終的にこの統合センシングシステムとして設計したのが、**図4-6**のシステムブロック構成です。これは、図で「システム最終出力」とあるMIDI規格の情報を出すまでの部分であり、3D-CGの生成、3D音響の生成、それらの情報をワイヤレスで飛ばすシステム、などは全て範囲外として省略しています。CCDカメラからの画像情報をライブ処理する部分は、パソコンソフトでは画像処理速度が不足するために、パソコンのシステムバスを外部に延長する「拡張バス」ラック、というFA用の道具を活用し、ここに画像処理専用メーカーの製品であるボードを2枚入れて、さらに「画像処理ライブラリ」、という専用モジュール群を購入してソフトウェアを開発します。

図4-6のシステムの中で、3人の来場者の心拍情報をセンシングする部分は、**図4-7**のように開発しました。これは耳たぶを挟むクリップのようなセンサで、耳たぶの毛細血管の血流量が心拍により変化することを、赤外線LEDとフォトトランジスタのペアにより検出するものです。この耳たぶクリップの出力から心拍情報を抽出する小型カードマイコンの部分は、第3章で紹介したセンサと同じような手法（詳細省略）で、これを小型無線送信ボードによってワイヤレス出力します。3人の情報を別個に扱うために、無線の周波数を異なるように特注する必要があり、普通より部品入手までの期間（リードタイム）が長くなるので、あらかじめ早めに連絡・発注を行います。

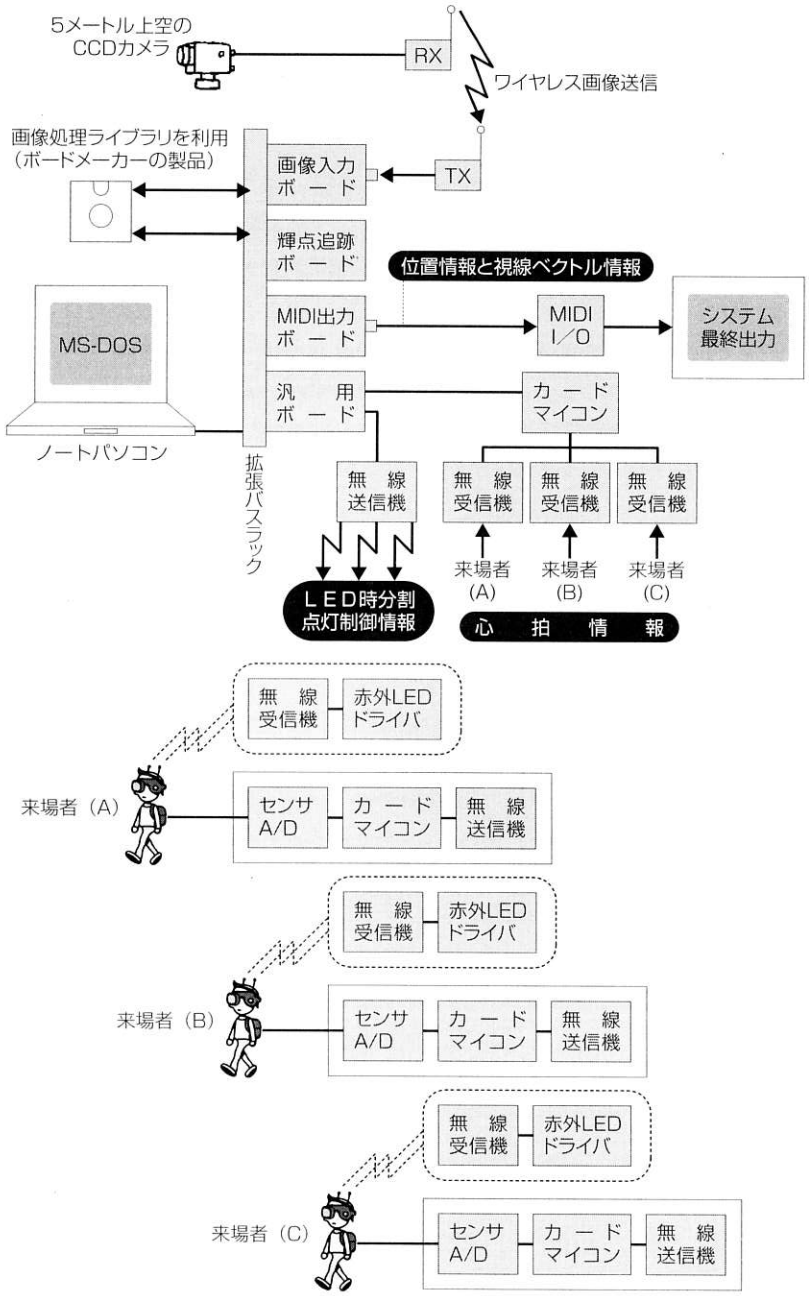


図4-6　「Audible Distance」のセンシングシステムのブロック図

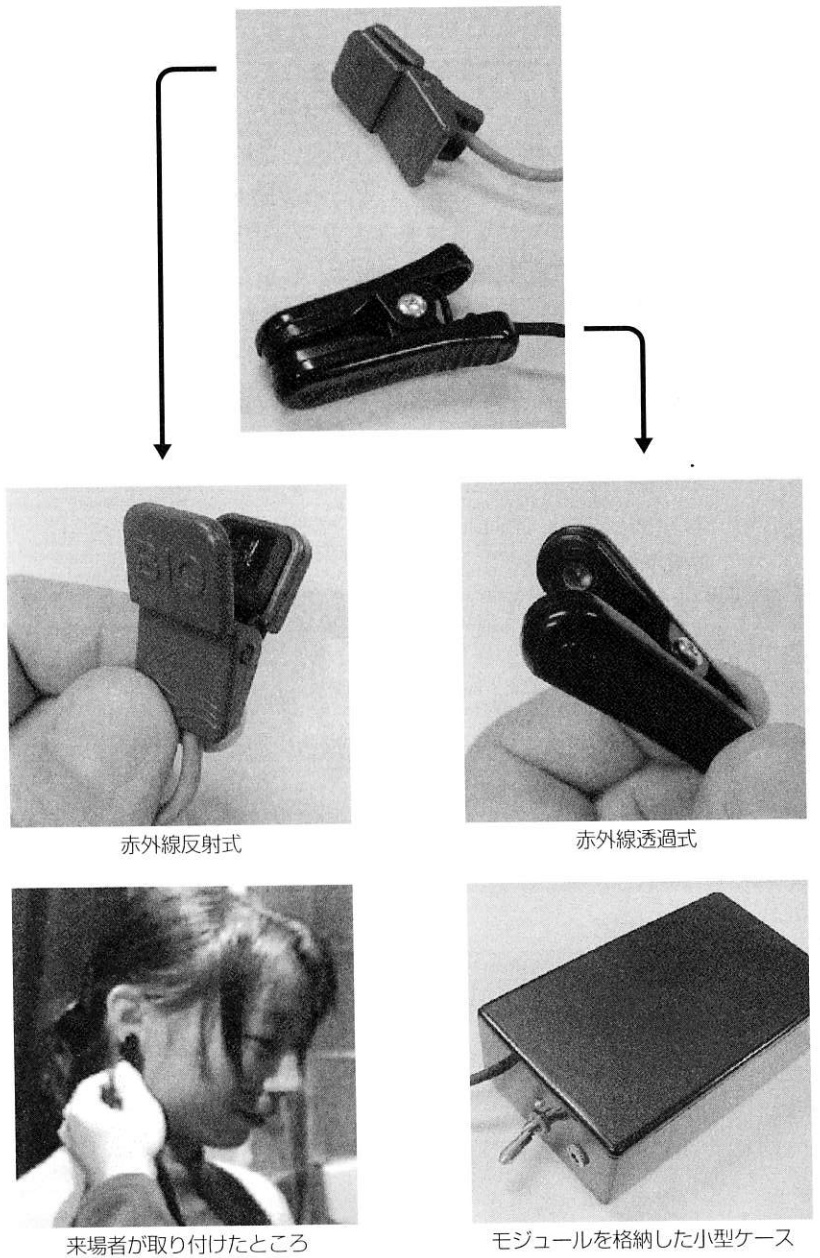


図4-7　心拍センサモジュール部分

図4-8は実験・試作の模様ですが、ハードの実験は、対応するソフトウェアの実験と同時進行になりました。これは、前章図3-3の単純な「ウォーターフォール」型ではなく、図4-9のように、ハードウェア開発とソフトウェア開発が、かなりの部分でオーバーラップしたもので、システムの規模と機能が高度化されていくと、このような「ハードとソフトの並行開発」が必須となります。ところで図4-5のように、3D-CG映像には立体視生成コンピュータを3人分で3台使い、3D音響の生成には全部で6台の音源装置を使っています。しかし、センシングシステムはコストの制約から、上空のCCDカメラを含めてシステムとしてまとめて1台（一式）、という条件があります。そこで「時分割多重化」という、ディジタル技術のもっとも基本的な手法を応用して、共有するハードを時間的に分割して複数の処理を並列に行いました。

システムは図4-6のように、来場者のかぶる帽子の前頭部と後頭部に

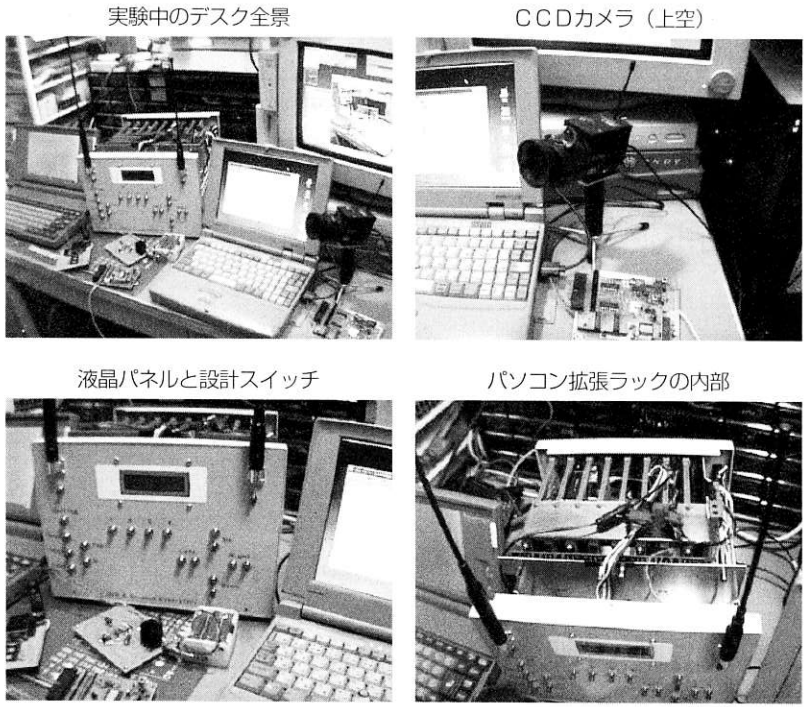


図4-8　センシングシステムの実験・試作の風景

それぞれ3個束ねた高輝度赤外LEDを、無線通信モジュールによって、3人の2箇所、計6系統を個別に瞬間的に光らせて、これを上空のCCDカメラから映像信号として出力します。このCCDカメラからのビデオ信号は、1.25GHzのワイヤレス画像通信モジュールで伝送して、「拡張バス」ラック内の「画像入力ボード」に供給します。図4-10（A）のように、CCDカメラの視界がボード上の512×512の画素データになるまでの処理所要時間は、毎秒30フレームということで、約30msecです。メーカから購入したライブラリ集の中の「2値化モジュール」を利用した全画面の2値化の所要時間は、（B）のように約1secです。その後、もう1枚の「エッジ検出ボード」に領域の座標を与えると、輝点の重心座標は（C）のように5msecほどで得られます。高価なライブラリを使っているにも拘わらず、6個の輝点のうちの1つの検出だけで（B）の部分に1secもかかり、このままでは使えません。

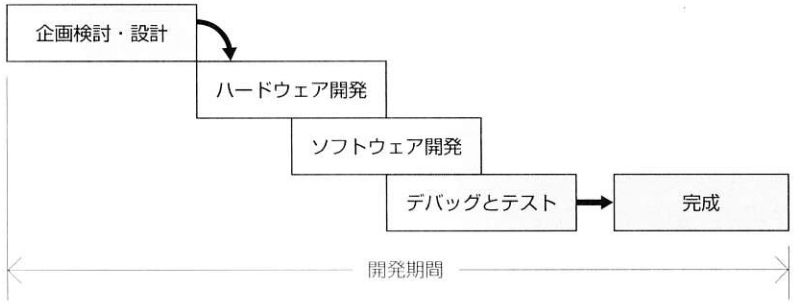


図4-9　より複雑なシステム開発フロー

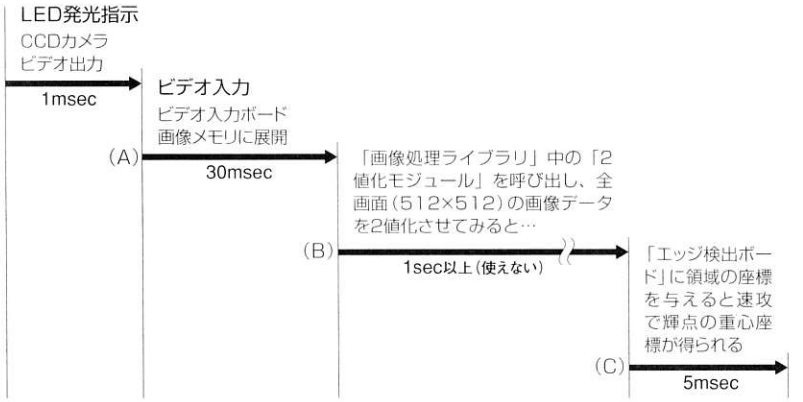


図4-10　画像センシングシステムのタイミング図

4-5
ソフトウェア設計・開発フェーズ

量産型の組み込みシステムであれば、ここで「画像処理専用のLSIを開発」というような、さらにハードウェア寄りのアプローチが続きますが、本件ではコストの制約もあり、ここから腰を据えて、ソフトウェアによる解決を模索します。解決方法は次の「デバッグ」の項で紹介しますが、**図4-11**は、時分割多重化により最終的に採用した、このセンシングシステムの画像処理に関するタイミングチャートです。ここでは、来場者（A）の前頭部の赤外LEDを「A1」、後頭部を「A2」、来場者（B）も同様に「B1、B2」などとしています。ポイントは、CCDカメラ、ビデオ入力ボード、2値化処理、エッジ検出ボード、のシステム要素がそれぞれ、並列的に処理を行うところで、最小限の処理時間で結果が得られます。参照のために赤外LEDを点灯させてからCCDカメラを経て画像入力ボードの取り込みまでの所要時間は、約35msecです。

プログラミング作業としては、センシングシステムの中核にノートパソコンを使用したので、このパソコン上でソフトウェアを開発します。つまり、前章で紹介した「クロス開発」でなくて、**図3-9**（A）のような、汎用システムと同じような開発環境です。ただし実際には、プログラミング言語として、**図4-12**の「BASIC」「C」「アセンブラ」という3種類を使いました。BASICは同図（A）のように、パソコンのOSの上で「BASICインタプリタ」というアプリケーションを走らせて、その中でBASIC言語によるプログラムを対話的に編集・実行できるもので、3種類の中ではもっとも実行速度が遅い「実験用環境」です。これは本番で使うわけではなく、ハードウェアの動作確認やデバッグの段階で、手軽にパソコンの命令を実行・検証するために活用します。

そして実際には、**図4-12**（B）の「C言語」、あるいは場合によっては（C）の「アセンブラ言語」を用いて、最終的なシステムを開発します。ほとんどの組み込みシステムのプログラムはCで開発するのですが、スピード重視の必要があるケースでは、CPUのもっとも高速な命令をストレートに記述できるアセンブラの方が数倍ほど早い場合もあり、組み合わせて使用します。画像処理ボードのメーカ製品として購入した「画像処理ライブラリ」の各ルーチンは、C言語でもアランブラでも呼び出して利用することが可能です。ところが、どうもこのライブラリにはバグがあるらしく、あまり性能が出ない（結果が出るとしても非常に遅く

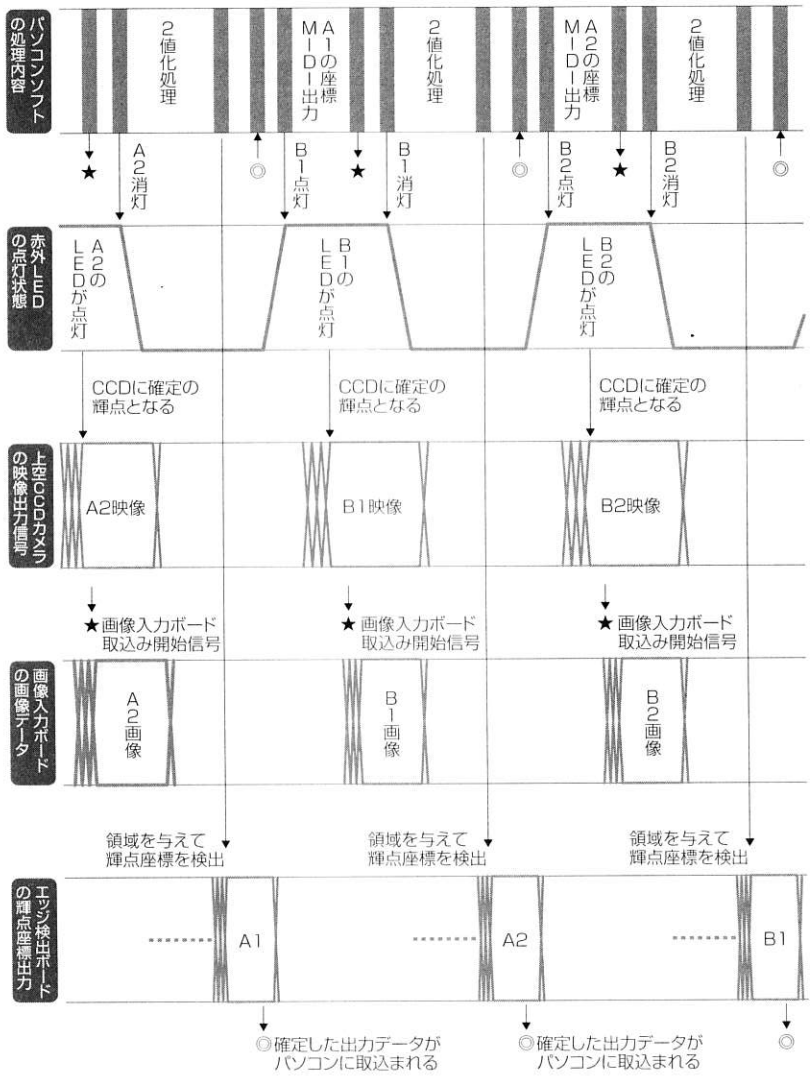


図4-11　実際のタイミングチャート

て使いものにならない）ため、開発プロジェクトの終盤は、この「2値化」処理をどのように高速化して現実的なセンシング性能を出すか、という点に絞られてきました。

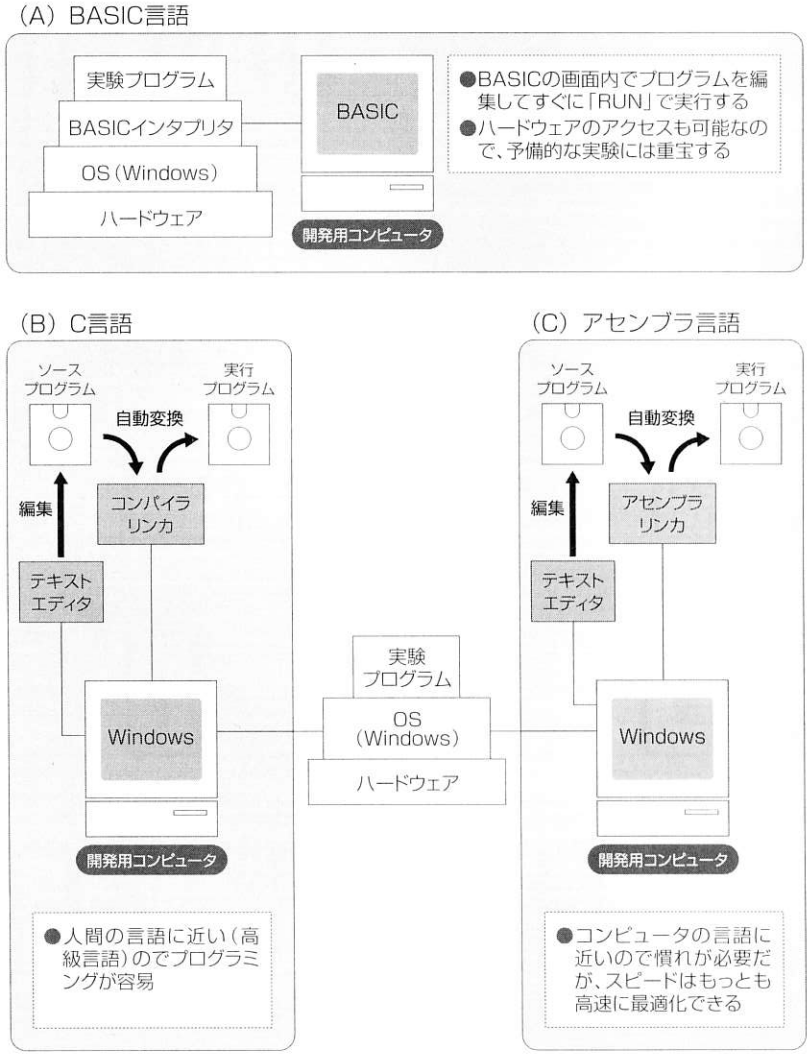


図4-12　3種類の開発言語による実験と開発

4-6
試作・デバッグのフェーズ

開発フェーズと重複して試作・デバッグのフェーズとなり、作業は佳境に入りました。「2値化」の部分が異常に長いことに目をつぶれば、**図4-11**のタイミングチャートとともに、まずは個々にこのやりとりを実現・確認していくことになります。ここで重要なのは、動作を基本的な単位要素に分けて、それぞれのステップを確認していくことと、「典型的データ」「境界値データ」などを駆使することです（**図4-13**）。これはソフトウェア工学にある「境界値分析」という考え方ですが、この例では、6点の赤外LEDの輝点座標は、上空5メートルのCCDカメラの画像の中で、512×512という領域内の座標として得られます。このあらゆる点の組み合わせを試す、というのは膨大な可能性ですが、試作・実験で確認する必要のある座標としては、領域の真ん中付近について確認できれば、あと重要なのは、領域の周囲にハミ出るかどうか、というあたりに絞られます。その途中については、内挿と外挿で補間します。ディジタルシステムで、理論的に「全てのデータの組み合わせ」の検証は原理的に困難なので、ある種の連続性や線型性の保証されているケースでは、このような手法も重要です。

C言語ベースで**図4-11**のタイミングチャートの動作を確認した段階で、1個のLEDについてのセンシング周期は7秒ほどありました。メーカーの技術担当者に問い合わせた結果、ライブラリのバグ（解決までの時間は期待できない）を確認できたので、現実的にこの処理を高速化する方法として、

- パソコンにもっと速いものを使う
- 画像処理ライブラリを解析し、アセンブラで自前のライブラリを作る
- 画像入力ボードを直接アクセスして、1ラインずつ飛び越しを行う

などの方法を検討し、一部を実験しました。パソコンはいくら速くても2倍、1ライン飛び越しの直接アクセスも全体としての効果は1.5倍程度、と困難を窮めた結果、エッジ検出ボードに与える「領域データ」を狭くする、というアイデアで、最終的に6点のセンシング全体を1秒以下に抑え込みました。

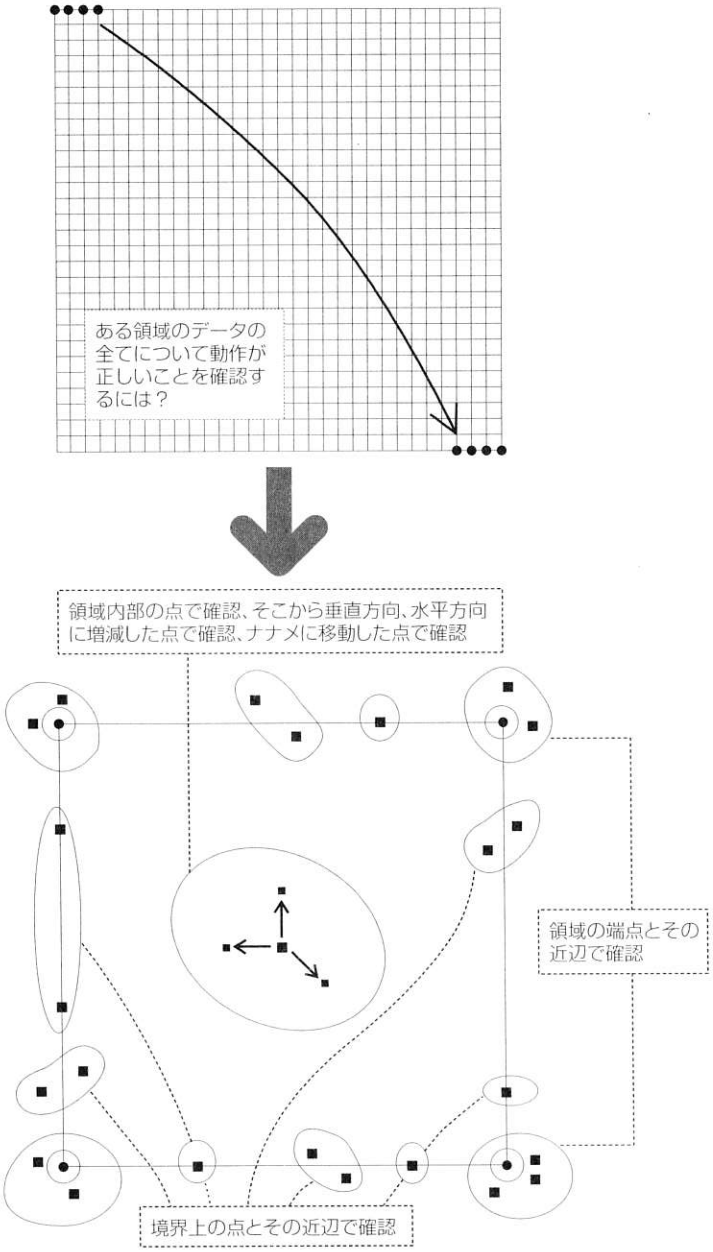


図4-13　境界値分析の手法

これは、「HMDをかけて真っ暗な空間をおそるおそる歩く来場者の位置は、瞬間的に1メートルなどの大きなジャンプをする事はない」という現実的な仮定のもとで採用した戦略で、システムは最初は反応時間を犠牲にして512×512の全領域をスキャンしますが、位置が判明した段階で、次回のセンシングでは「前回の位置の周辺の狭い領域を探索する」という方針としたのです。これにより、

- 探索範囲を狭くすればより高速に検出できる
- あまり狭くした範囲をはみ出ていると全域検索となって遅くなる

という、現場で調整できるパラメータを抽出して、これはオペレータが設定できるようなシステムとして完成しました。**図4-14**は、展示会場の現場でデバッグしている様子です。このように組み込みでは、最終的には稼動する現場での実機確認が重要です。

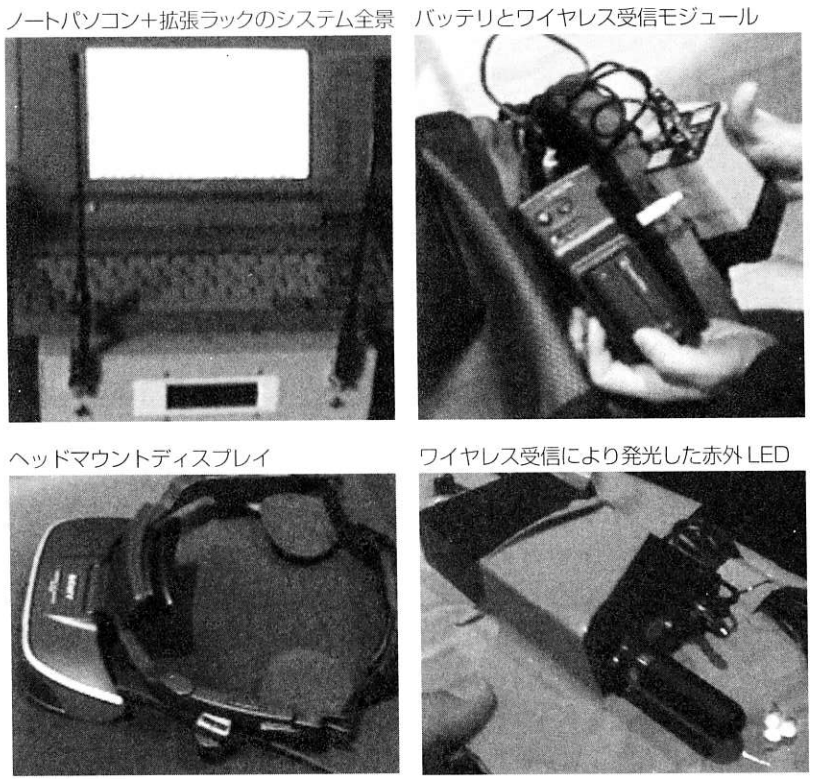


図4-14　デバッグ中の様子

4-7
製造、保守のフェーズ

本システムは全体としては1セット（一式）のものでしたが、来場者が入れ替わり身につけるワイヤレスモジュールや耳たぶセンサ等については、万一の破損や断線などのトラブルが予想されます。このために、最初から3セットでなく、周波数を4パターンに拡張した4セット分を開発して、修理の際には残り3セットを使用して展示を止めず、その間に交換・修理できるようにしました。また、バッテリ交換が容易なように、市販の実験用プラスチックケースを用いたり、耳たぶセンサとワイヤレスモジュールとのコネクタに汎用のプラグ・ジャックを使用して、メンテナンス性まで考慮します。このように、組み込みシステムの製造段階では、後々の「保守」までを守備範囲とします。

設計・開発フェーズの部分では省略しましたが、組み込みシステムでは、「ユーザー・オリエンテッド」という設計指針が重要で、これは、完成したシステム納品後の「運用・保守」というフェーズで効いてきます。例えば、工場の自動加工機などのような専用組み込みシステムでは、装置自体の寿命とは別に、連続使用のために磨耗・消耗して部品交換する、というケースを想定して、「交換部品が容易に手配・入手できる」「部品の交換が納入先の現場だけで可能」などの条件を設定します。遊戯場や科学館などの「体験マシン」などの専用組み込みシステムでは、利用するユーザーが初心者や子供だったりすると、設計者の想定外の「無茶な」使い方をする可能性もあります。これはPL（製造物責任）と通じる考え方ですが、事故や故障を避けるために、あらゆるユーザーを想定する、というのもエンジニアの仕事です。

本システムは、「ICCビエンナーレ'97」の2ヶ月間を無故障で乗り切りました。メディアアート美術館、という展示会場の特性から、このイベントの2ヶ月間は、基本的に作家の前林氏本人か、レクチャーを受けたスタッフが会場にいます。毎朝のシステム立ち上げについては、「ノートパソコン＋拡張ラック」という全体の電源を入れて、パソコンの自動立ち上げファイルに初期設定を記述しておけば、基本的には問題ありません。センサの感度調整が必要になった場合も、この設定値をデータファイルとして保存するようにして、常に最新のデータに対応することとしました。

そして、このシステムを組み込んだ作品「Audible Distance」は、コンペで「準グランプリ」を受賞しました。その後、改めてNTTからICCの常設展示作品として依頼され、大幅な改造が必要になりました。システムの一部にノートパソコンを使用するのは、より長時間の連続使用には向かない、期間限定運用システムのための方法です。この最大の弱点は、ハードディスクという回転機構に頼るところです。また「常設展示」となると、会場には一般の受付職員だけがいるので、マニュアル化されたバッテリの交換を除いて、毎日の立ち上げ設定調整とか、システムのトラブルの対応はできません。条件は「電源を入れたままで無停止の連続運転」ということです。

4-8
OSとソフトウェアのROM化

組み込みシステムでは、このような「メンテナンスフリーの連続運転」という条件は一般的です。もっとも過酷なのは人工衛星の観測機器などで、誤動作したといっても、リセットスイッチを押すこともできません。自動販売機なども、電源が入ってずっと無人で連続動作をするのが基本で、いちいち人間がメンテするのでは、ちっとも「自動」ではありません。このセンシングシステムの改造は、組み込みでよくある「OSとソフトウェアのROM化」という定番テクニックに進展することになりました。

図4-15は、このような「パソコンをシステムの一部として組み込む」という手法の組み込みシステムの「ROM化」です。通常は上図のように、パソコン自体のメインメモリ（RAM）と、OSやプログラムやデータをファイルとして格納しているハードディスクにより、システムは動的にメモリを管理しています。ここには一長一短があり、動的に管理することで、ソフトウェアを容易に変更できたり、データを更新してファイルとして保存することができます。その一方で、動的なデータ管理は時にはガベージという端数の「ゴミ」を生み出し、長時間の動作によっ

てはこの蓄積が暴走の原因となる場合もあります。

これに対して、**図4-15**の下図では、パソコンに接続した「拡張バス」ラックを活用します。このラックのバスは、パソコンの全ての信号が出張所として出ているので、ここにパソコン本体の代わりに、「パソコンに相当する全機能を搭載したCPUボード」を入れることも可能で、このボードがマスタとなってシステムを管理します。これと同時に、ハードディスクに相当する「ファイル読み出し装置」として、「ROMディスク」というボードを使用します。これはボード一面にEPROMソケットが敷き詰められた大容量メモリボードで、データ、プログラムだけでなく、OSそのものまで全てROMデータファイルとして格納しています。当然、何の書き換えもできないのですが、機械的な機構のない、長時間連続動作に最適のシステムへと変貌します。

ROM化システムのCPUボード上には、パソコンのマザーボード上のものと同じBIOS（基本入出力）ROMがあり、電源ON、あるいはリセットにより、BIOSはRAMに「ブートローダ」という初期設定プログラムをロードします。ハードディスクでなくROMディスクに格納されたブートローダが呼び出され、続いてハードディスク所定領域に格納されたOSに相当する、ROMディスク内のOSがメインRAMにロードされます。OSの初期自動実行プログラムもROM化されているので、これに従って、組み込みシステムとしてオリジナル開発したROM化プログラムがロードされ、実行されます。プログラムのバグの変更などは一切、出来ませんから、十分にデバッグしたプログラムをROM化する必要があります。

結局、ROM化されたこのセンシングシステムは、ICC常設展示作品として、まる2年間の連続運転（電源は365日、入れたまま）に耐えて、無停止無故障という記録を無事に達成できました。組み込みシステムとしては、それほど複雑なレベルではありませんが、それでもセンシングシステムだけでも6台のコンピュータの連携、作品全体のシステムとしては20台近いコンピュータの同時動作によってシステムが動いている、というのは壮観で、開発した甲斐がありました。

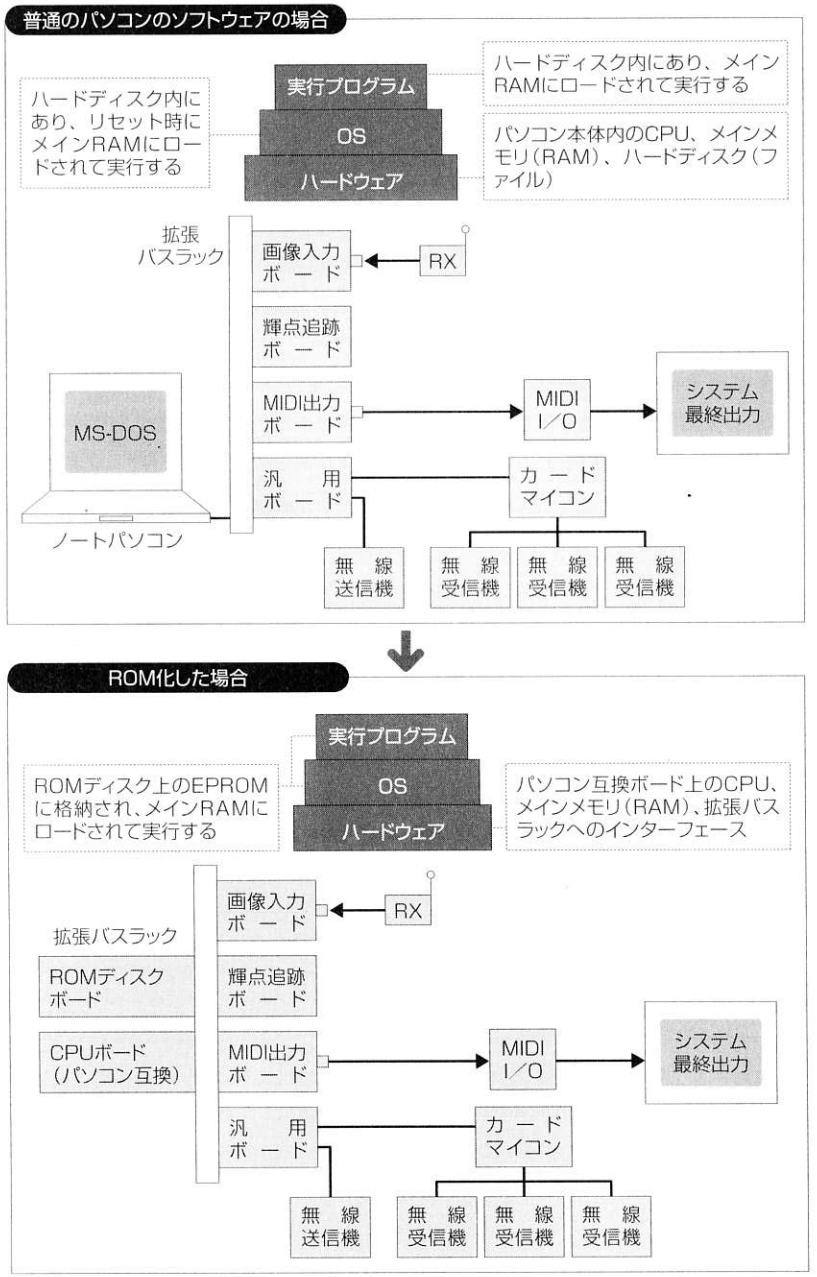


図4-15　OSとプログラムのROM化