第1章　作業を始める前に

パソコンでリアルタイム制御を始めるにあたって，組み込みプログラム開発技術者が，いま，どんな環境におかれているかについて述べます．

技術がめまぐるしく変化する状況において大事なことは，潮の流れ，変化の方向，将来の姿を読み取ることです．

何が変化しているのか，変化をもたらしている原動力は何か，変化のトレンドはどこを指しているのか，それらについて考察します．

具体的な技術が必要だという読者は，本章をスキップして，第2章へジャンプしてください．

■ 1.1　モデル・ベース開発の発展

大学の研究室や企業の開発部門において，機械の制御装置を開発する際に，モデル・ベース開発（Model Based Design: MDB）あるいはモデル駆動開発（Model Driven Development: MDD）の手法を採用するケースが多くなりました．

モデル・ベース開発を採用すると開発の過程はどのようになるか，その進行過程をスキット風に述べます．

仮にいま，化学プラントの制御装置を製作するとします（**図1.1**）．

プラントには，いくつかの反応器，触媒供給装置，反応器を結ぶパイプライン，調整弁，加熱バーナなどがあります．

まず，反応器内で起こる化学プロセスを分析します．

化学反応器には，いくつかの状態量があります．例えば，原材料の供給量，反応器内温度，圧力，出力物質の組成比などです．状態量の中には，観測可能な量とそうでないものがあります．測定可能な状態量を可観測量（observable）といいます．観測可能でない状態量に対して，必要ならば推定のための数式モデルを作ります．

化学反応器には，いくつかの制御量があります．例えば，器内温度，撹拌速度，触媒添加量，弁の開度などです．制御量を変えると，それにしたがって反応器の出力は変化します．

これらの変数を使って，反応器の数学的なモデルを作ります．これらは通常なら複雑な数式モデル

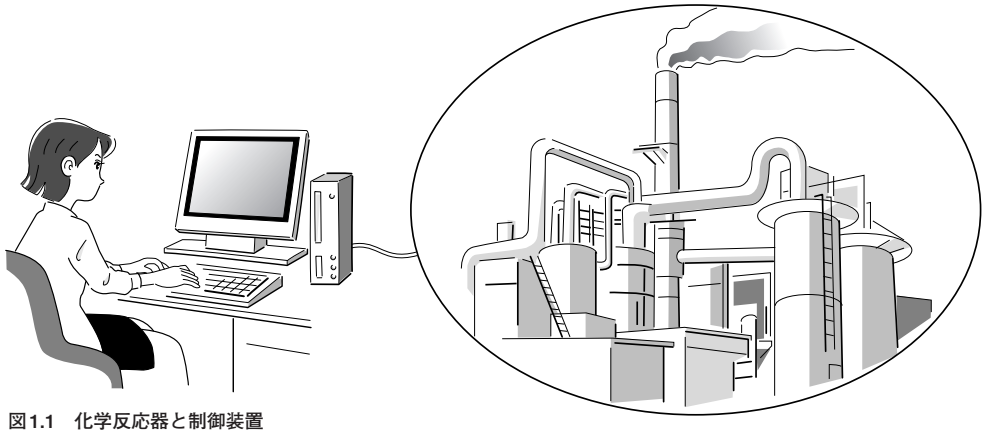

図1.1　化学反応器と制御装置

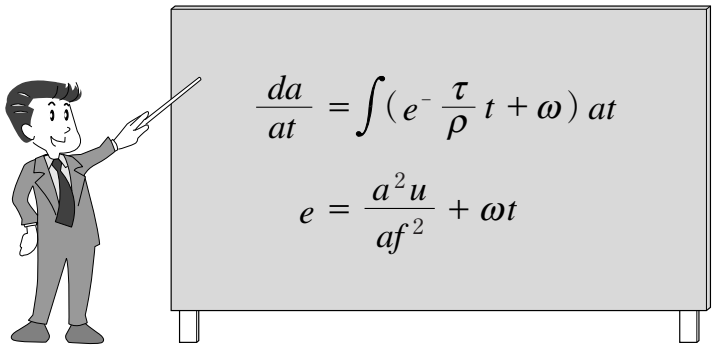

図1.2　数式モデルの例

になります（**図1.2**）．

反応器の数式モデルができあがれば，反応器を一つのブロックとして，コンピュータへインプットします．コンピュータのディスプレイ上にブロックをドラッグ・アンド・ドロップして，そのブロック内に数式モデルを書き込みます．必要ならば，ブロックのプロパティを変更します．

ブロックを書いたら，次はブロックを線で結びます．これらの線は，プラント内のパイプラインに対応します．コンピュータのディスプレイ上に，ブロックの線図ができます．このブロック線図を，本書ではスキーマ（schema）と呼びます（**図1.3**）．

スキーマは，ブロックとそれらのブロックを結ぶ線分から構成します．

ブロックは，入力ポイントあるいは出力ポイントのいずれか，あるいはその両者があります．入力ポイントなし，かつ出力ポイントもなし，というブロックは存在しません．

スキーマは，物理的なプラントをコンピュータ内に論理的に実現するものです．スキーマと言う代わりに，モデル（model），あるいはチャート図（diagram）と言うこともあります．

スキーマができあがったら，コンピュータを使ってシミュレーションを行います．

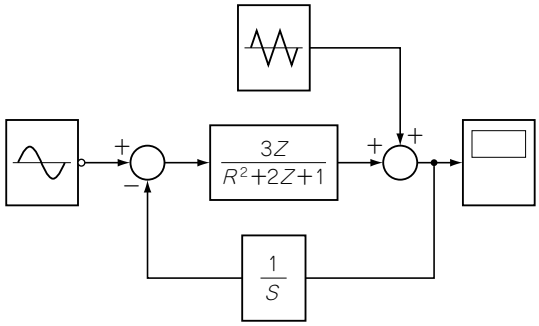

図1.3　化学プラントのスキーマ

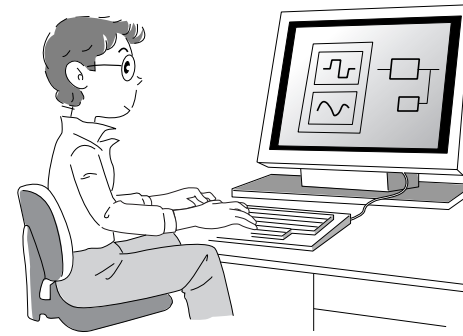

図1.4　シミュレーションのグラフ表示

ウインドウのツール・バー上のボタンをクリックして，プラントのシミュレーションを開始します．シミュレーションが進行すると，反応器内の温度，流量などの状態量がグラフ表示されます（**図1.4**）．

仮にシミュレーションの結果，反応器内の温度が設定値よりも低く，出力は設計値に至らなかったとします．

この場合，加熱ヒータのフィードバックの比例ゲインが不足していたと推測して，このゲインを10%上げて，再度シミュレーションを行います．

今度は，ゲインが大きすぎたために反応器は不安定になり，器内温度が激しく乱高下してしまいました．しかし，これはコンピュータ・シミュレーションなので，心配することはありません．反応器内温度が乱高下しても，反応器は爆発したりはしません．安心してください．

……

と，こんな風に，シミュレーションを進めてフィードバックのゲインのチューニングします．

以上で，制御に必要なフィードバック・ゲインの調整は完了しました．

続いて，実プラントにおける検証過程に入ります．

スキーマから制御ブロックを抽出して，そのブロックをCのプログラムへ変換し，それをコンパイラにかけて実行プログラムをビルドします．ビルドした実行プログラムを制御用のターゲットのパソコンへダウンロードします．

このパソコンは，アナログ入出力ボードなどを介して，プラントと直結します．制御プログラムを実装したパソコンの制御によって，プラントを稼動し，実世界のデータを採取します（**図1.5**）．

プラントの応答を調べて，シミュレーションの数学モデルを修正したり，制御のためのフィードバック・ゲインを再調整したり，あるいは制御アルゴリズムをPID制御から最適制御に変えたり，そういった変更を必要に応じて行います．

仮に，実機による実験データを分析した結果，反応器の寸法を変更する必要があるという判断に達したとします．そんなときでも，プログラムを自ら手直しする必要はありません．

コンピュータのディスプレイ上で，スキーマの数式モデルを変更します．変更したスキーマにビルドをかけると，新しい実行プログラムが自動的に生成されます．このプログラムを制御用のコンピュータ

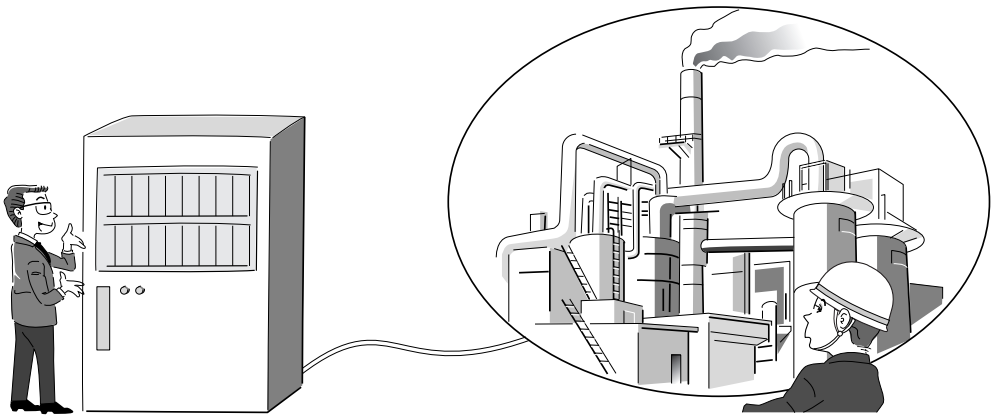

図1.5　実プラントにおける検証

へダウンロードすれば，直ちに実プラントにおけるデータ採取が可能になります．

モデル・ベース開発において，開発を行うホスト・コンピュータとプラント制御が直結しているので，上流のスキーマを変更することによって，すぐに物理プラントにおける実証実験が実施可能になります．

モデル・ベース開発に関して，もう一つのスキットを示します．

舞台は，同じく化学プラントの制御系設計問題のシーンとしましょう．

プロジェクトは順調に進行していたのだけれども，海外から大型プラントの受注が入り，試作部の全スケジュールが組みかえられて，化学反応器に関する試作機の製作が3ヵ月遅れることになったと仮定します．

社外秘の特命プロジェクトなので，試作を外注へ出すことはできません．あくまでも内製する必要があります．

制御装置はすでに完成しており，実機待ちの状態です．

皆さんなら，どうしますか．

これは良い機会だなどと言って，ちょいとハワイへ海外旅行に出かけたりしますか？ まさか，そんなことはないでしょう．若手技術者の提案で，シミュレーションに使用したモデルを制御部とプラント部に分割します（**図1.6**）．

プラント部をC言語に変換して，これをコンピュータ内に実現します．

制御装置の実機とプラントの動作をシミュレーションするコンピュータを接続して，仮想的な実験を実施し，制御プログラム内のバグ取りを行います．

制御装置の実機とプラントのシミュレータを接続して制御プログラムの検証を行う方法を，通常，HILS（Hardware In the Loop Simulation）といいます．HILSによって，制御プログラムの検証が完了するとは言いませんが，実機実験に必要な期間の短縮に貢献することは確実です．

一般に，プラントの規模があまりにも巨大で，試作プラントを作ることができない場合や，制御装置がプラントよりも早期に完成する場合などにHILSを採用します．

図1.6　HILSの構成

モデル・ベース開発は，HILSとして使うこともできるわけです．

■ 1.2　モデル・ベース開発の導入

モデル・ベース開発は，コンピュータのディスプレイ上にスキーマを描き，これを使ってシミュレーションを行い，その結果をダイレクトにプログラムに変換し，実機実験の実施を可能にします．

モデル・ベース開発は，コンピュータを用いて機械の制御装置の開発工程を上流から下流まで一元化し，開発期間を短縮し，かつ設計変更を容易にします．

しかし，モデル・ベース開発を導入するために，いくつかの条件があります．それらの諸条件について述べます．

モデル・ベース開発のスタート点は，コンピュータのディスプレイ上に描くスキーマです．スキーマは，２次元ディスプレイ上に展開したグラフです．

スキーマのブロックとしてUML（Unified Modeling Language）を使用すべきだ，という意見があります．UMLはグラフ表示を行うための標準言語である，というのです．これは明らかにまちがいです．

UMLのUは英単語のunifiedの頭文字であり，その意味は，複数の言語をまとめて一つの言語にしたという意味であり，具体的にいえば，いくつかのモデリング言語を組み合わせて一つの表記法を作ったという意味です．「標準（standard）」という意味は，どこにもありません．

モデル・ベース開発で使用するブロックは，直接シミュレーションを実行し，かつコンピュータ言語へ変換できるものでなければなりません．これが絶対の条件です．

例えば，制御工学で使用する伝達関数は，

$$G(s)=\frac{a_m s^m+\cdots+a_1 s+a_0}{b_n s^n+\cdots+b_1 s+b_0}$$

という形式になります．

伝達関数の内容は，分子と分母の多項式の係数を指定すれば，一意に決まります．係数の，

$a_m, a_{m-1}, \cdots, a_1, a_0 \quad b_n, b_{n-1}, \cdots, b_1, b_0$

をプロパティとして指定すれば，伝達関数の内容は厳密に決まります．曖昧性は一つもありません．

したがって，伝達関数は，モデル・ベース開発のブロックとして採用することができます．

UMLは，システム分析を行うような漠然とした状況において使うものであり，モデル・ベース開発などのように，数学的に厳密な構造を必要とする状況において使用するべきものではありません．

モデル・ベース開発において，ディスプレイ上にスキーマを描くと，シミュレーションが可能になり，プログラムの自動生成が可能になります．

ここにも注意点があります．ディスプレイ上のスキーマは，本質的に2次元平面上のグラフです．ところで，コンピュータのプログラムは，本質的に1次元上に配置する文字列です．2次元を1次元に変換する過程を，一般にシリアライズ(serialize)と呼びます．

モデル・ベース開発におけるシミュレーション，およびプログラムの自動生成において，スキーマのシリアライズが必要になります．2次元に展開されているグラフを1次元の文字列へ変換する一般的な方法はありません．それが可能という証明もありません．

数学的にいうと，任意の2次元グラフをシリアライズする一般的なアルゴリズムは存在しないので，通常は適当な選択基準を作り，それによって，便宜的に2次元のスキーマを1次元のプログラムに展開します．

すなわち，選択基準の選択によって，シリアライズの結果は異なるものになります．

例えば，**図1.7**に示す伝達関数のブロックがあったとします．

このブロックにおいて，入力を与えれば，出力は計算できます．**図1.8**に示すように，ブロックに対してフィードバック回路がついたとします．

ブロックの入力は，同じブロックの出力を含みます．このブロックの計算は，単純ではありません．ちょっと厄介です．入力の値が決まれば出力が計算できますが，その出力値が入力になるのです．つまり，蛇が自分の尻尾を飲み込むような形式になります．

また，ここでは深く追求しませんが，2次元のチャート上に，デッド・ロック(dead-lock)と呼ばれる図形を描くこともできます．このような図形に関して，シリアライズは正常に行われないので，シミュレーションの結果が信頼できないものになる恐れがあります(**図1.9**)．

このような場合には，人が介入してモデルの図形を別の図形に書き直したりします．

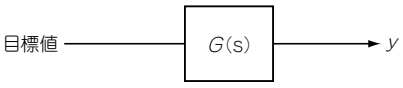

図1.7　ブロック線図

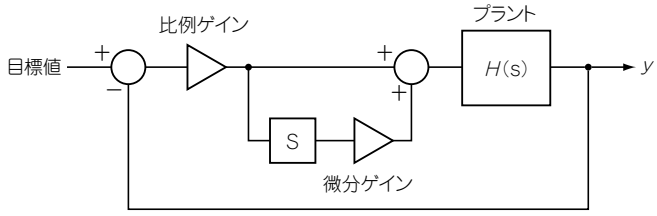

図1.8　フィードバックつきのブロック線図

モデル・ベース開発を導入する際に，人の性格が関係する場合があります．

例えば，A君は組み込みプログラムの技術者の中でも「何でも自分でやらないと気がすまない」というタイプです．本章の冒頭において，CPUとパソコンのBIOSの構造に関するスキットを述べました．このスキットに当てはめると，パソコンを使って制御実験を行う際に，パソコンのBIOSを完全に理解しないとプログラムの作成を始めることができないというタイプの人です．

A君は，自分の技術のレパートリを拡張することを過大に評価し，かつ自分の人件費を過小に評価します．USBを使う前にUSBの規格を徹底的に勉強し，LANを構築する際にはTCP/IPのプロトコル・スタックを作ったりします（**図1.10**）．

A君のようなタイプの技術者を職人気質と呼びます．職人気質の組み込みプログラマは，多くの場合，モデル・ベース開発の導入に抵抗します．「何でも自分でやらないと気がすまない」ので，開発システムのブラック・ボックスが容認できないのです．

また，このような気質の技術者は，開発する制御システムの規模が小さい間は問題を起こさないけれども，開発するシステムの規模が大きくなるにしたがって，システム開発の障害になる場合があります．細部にこだわり，全体を見ない傾向にあるからです．

モデル・ベース開発の導入によって，このようなタイプのプログラマはプロジェクトから排除されることになります．

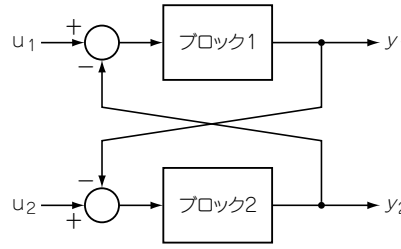

図1.9　デッドロックを含むブロック線図

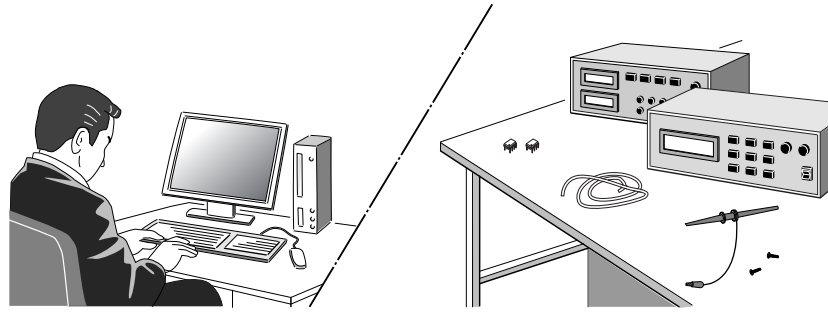

図1.10　職人気質のプログラマ

続いて，仮にモデル・ベース開発の導入を決定して，開発システムを購入したとします．

スキーマを描く段階において，開発システムの中に必要なブロックが用意されていないことを知って，びっくりすることがあります．こんなシステムは使いものにならない，などと言いたくなります．開発システムのメーカは，ブロックの開発に全力を挙げていることは確かだけれども，そうかといって，企業や研究室が必要とするブロックをすべて用意することはできません．そんなことは不可能です．

このような場合，ユーザは必要とするブロックを自力で制作して，それをシステムへ追加します．ユーザが自力で構築するブロックをカスタム・ブロック（custom block）といいます．

モデル・ベース開発を使いこなすために，カスタム・ブロックの制作技術を身につける必要があります．開発システムを購入すれば，それで「一丁あがり」ということにはなりません．注意してください．

■ 1.3　循環型の開発サイクル

モデル・ベース開発の導入が決まり，開発システムを購入し，システム分析を行い，スキーマができたとします．

実用的な問題において，スキーマは通常，数百あるいは数千のブロックを含むことがあります．循環型の開発では，大きなスキーマ全体を対象にするのではなくて，スキーマをいくつかの部分スキーマに分割し，階段的に検証を行います．

一つのスキーマを複数の部分スキーマに分割する過程は取り扱う対象に依存するので，ここで一般論として議論することはできません．

肝心なことは，各部分スキーマは必ず，

モデル　→　シミュレーション　→　プログラム生成　→　実機による検証

というサイクルを含むことです．

別の言葉で言えば，

プロジェクトを何段階かに分けて，階段的に実行する

ということになります．一つの段階ごとに必ず実機における検証を含むことが必須条件です．

いま，仮に第3段階の実機実験に失敗したとします．このとき，第3段階のスキーマを削除すれば，第2段階までを成功の状態にロール・バックすることができます．

このように，「石橋を叩いて渡る」を実現するのが循環型の開発システムです．

■ 1.4　変化に耐えるシステム

開発の過程において，客から仕様変更が通知されたとします．

なんていうことだ，そういうことは最初にじっくり考えておくべきことだ，などと激怒したくもなります．しかし，パニックに陥って右往左往することはありません．

モデル・ベース開発を採用する限り，コンピュータ内のスキーマを変更すれば，新しいモデルによる

シミュレーションも，プログラムの自動生成も，実機実験も自動的に行うことができます．

すなわち，それまでに書き上げたプログラムを棄てて新しいプログラムを一から作成する，というような状態へ追い込まれることはありません．

現在のように環境が激しく変化する時代において，一定の条件のもとで長期間，開発を進めることがまれであると考えるべきでしょう．

つまり，モデル・ベース開発を採用して，柔軟な開発環境を構築することが必要です．

■ 1.5　xPC Targetの紹介

MATLABを開発，販売するThe MathWorks,Inc.は米国の東海岸ボストン近郊に本拠を置く会社です．マトリックスの解法から出発して，現在，精力的にレパートリを拡大しています．フィルタ設計，制御系の設計，動特性を考慮した機械システムの設計などの分野において勢力を拡張しています．

MATLABの一つの分枝として，制御系の設計のモデル・ベース開発を可能にするプロダクト・ファミリがあります．

例えば，

MATLAB，Simulink，Real-Time Workshop

を購入すると，Simulinkによるモデルの構築，シミュレーションの実行，Real-Time WorkshopによるC言語プログラムのビルドなどが可能になります．この過程については，参考文献2において詳しく紹介しました．参考にしてください．

この3本のプロダクトに対して，

xPC Target

を追加すると，ホストにおいて自動生成したCのプログラムを実行形式にコンパイルして，それを通常のパソコンへ転送し，そこで制御実験を行うことが可能になります．

xPC Targetにおけるターゲットは，市販されている通常のPCです．

xPC Targetは小規模のリアルタイムOSを内蔵しているので，ターゲットにおける実行環境を別途に組み込む必要はありません．

パソコンBIOSの対応はxPC Targetに内蔵されているため，ユーザはそういった雑事に苦しむことはありません．

xPC Targetのアプリケーションはx86のリング0で走るので，ユーザのプログラムにおいてIO命令を使うことができます．とても便利です．

パソコンの価格が十分に安くなり，どんな開発環境においても手軽に使えるようになった現在，これらのプロダクト・ファミリを使ってモデル・ベース開発開発環境を構築することを薦めます．

これらのプロダクト・ファミリを使ってモデル・ベース開発を構築する過程を，次章以下において詳しく述べます．

第2章　パソコンの準備作業

xPC Taegetの最初の作業は，適当なパソコンを選び，それらのパソコンをxPC Taeget用に整備する作業です．

あたり前のことですが，用意するパソコンは，xPC Targetが要求する条件を満足するものでなければなりません．パソコンであれば何でもよい，というものではないので注意してください．

作業の工程を具体的に示すために，私は実際に4台のパソコンを選び，その中の1台をホストに，残りの3台をターゲットとして整備して実機によるテストを行いました．

その準備作業の過程をここに述べます．

私が作ったシステムは一つの製作事例であり，当然，ほかの形式のシステムは多く存在します．しかし，入門に最適な，すなわち最少の努力で構築できるシステムなので，読者が実際にxPC Targetを試みる際に採用することを薦めます．

■ 2.1　必要な機器と役割

作業を始めるにあたって，まずxPC Targetが必要とする機器，およびそれらの役割について述べます．

xPC Targetが必要とする機器とそれらの接続関係を**図2.1**に示します．

ここで考えるシステムの構成要素は，図示したように，

パソコン2台
制御対象の機械装置一式

です．

この制御対象の機械はユーザが抱えている問題によって変わるので，ここで詳細は述べません．

1台目のパソコンをホストと呼びます．ホストは通常のパソコンです．特殊な整備は必要ありません．

ここでは，ホストのOSをWindowsと仮定しますが，Linuxの場合もほぼ同様の手順で整備できます．

ホストに対して，プログラム開発用のシステムをインストールします．

私がインストールしたプログラムは，The MathWorks社が発売する次の四つのMATLABプロダクト・ファミリです．

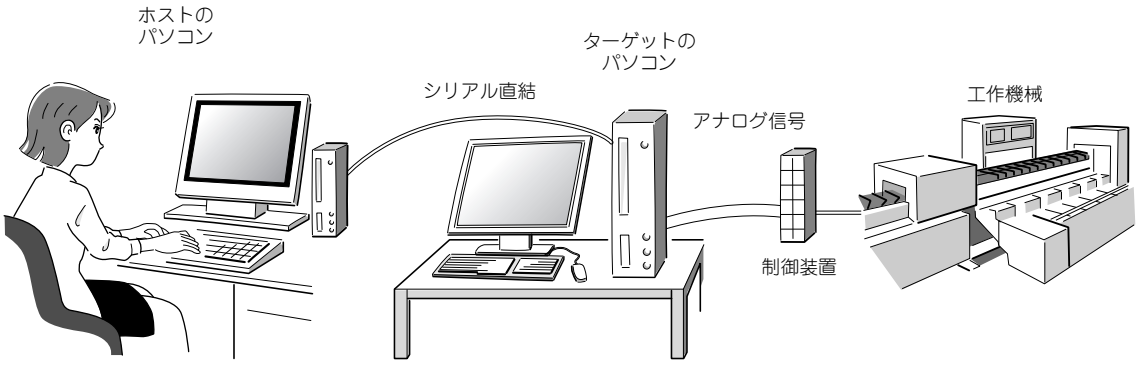

図2.1　パソコンの配置

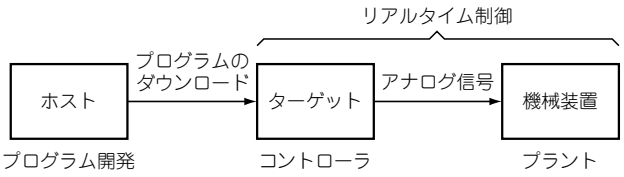

図2.2　システムのブロック図

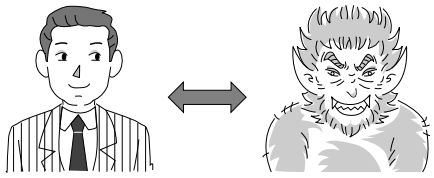

図2.3　ジキルとハイド

MATLAB ――――――プログラムの開発を行うフレームワーク
Simulink――――――ブロック線図を作成してシミュレーションを実行するツール
Real-Time Workshop ―ブロック線図からプログラムを生成するツール
xPC Target――――――アプリケーションをパソコンにおいて実行するツール

この四つのインストール過程の詳細は，2.3節において述べます．

ホストにおいて，制御アルゴリズムのブロック線図を組み上げて，シミュレーションを実行して妥当性を検証し，プログラムをビルドします．ビルドしたプログラムは，2台目のコンピュータへダウンロード，すなわちケーブルを介して転送します．

図2.1をブロック線図で示すと，**図2.2**となります．

2台目のパソコンをターゲットと呼びます．ターゲットというと，制御対象を意味するような感触を受けるかも知れませんが，そうではありません．

ターゲットは，制御用のコンピュータです．アナログ・ディジタル変換ボードなどを差し込み，制御対象の機械装置とケーブルで直結します．ターゲットは，機械のリアルタイム制御装置として使用しますが，実験に使用しないときは通常のパソコンとして使用できます．

小説の「ジキルとハイド」のように，あるときはパソコン，別のときは制御装置として使用します（**図2.3**）．

通常のパソコンとして使用する場合のOSは，Windows，Linux，DOS，そのほか何でもかまいませ

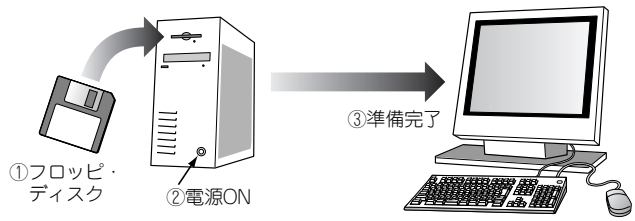

図2.4　フロッピ・ディスクによる起動

ん．このOSを，本書ではターゲットのパソコンOSと呼びます．パソコンOSなしにxPC Targetは成立しません．

ターゲットは，パソコンのハードウェアを裸の状態で使用します．裸の状態というのは，インストールされているパソコンOSとは隔離した状態で動作するという意味です．ターゲットはWindowsあるいはLinuxなどと無関係の状態で動作するので，それらの資源（例えば，OSのデバイス・ドライバなど）を使うことはできません．パソコンのハードウェアにダイレクトにアクセスします．

ターゲットのスタートアップは，3.5インチのフロッピ・ディスクを使用します．あらかじめプログラムを書き込んだ特別なフロッピ・ディスクを用意して（フロッピ・ディスクの作り方は2.9節において述べる），これをターゲットへセットして，電源をONにします（**図2.4**）．

もちろん，マザーボード上のBIOSの設定において，ブート・デバイスの順位を，

3.5インチのフロッピ・ディスク
Cドライブ

とします．

このように設定すると，フロッピ・ディスクをセットして電源をONにしたときは，ターゲットは制御装置となり，フロッピ・ディスクをセットしない状態で電源をONにしたときは，Windowsなど通常のパソコンになります．

ここで使用するフロッピ・ディスクは，パソコンをブート可能なフロッピ・ディスクでなければなりません．このタイプのフロッピ・ディスクを，とくに，レガシのフロッピ・ディスクと呼びます．IBMが最初にPCを発売したときに指定したIOアドレス，割り込みポート番号を使用する必要があります．

ノート・パソコンにおいて，USB接続のフロッピ・ディスクを使用することがあります．このタイプのフロッピ・ディスクはレガシのフロッピ・ディスクではないので，パソコンをブートすることができない場合があります．

また最近，フロッピ・ディスクを搭載していないパソコンが販売されています．これらのパソコンは，ターゲットとして使用できません．注意してください．

ターゲットは，フロッピ・ディスクからプログラムを読み込み，そのプログラムを実行します．このプログラムをスタート・プログラムと呼びます．

フロッピ・ディスクの記憶容量はおよそ1.5MB程度です．この中に，

リアルタイムOSのコアの部分
キーボードからコマンドを読み取るプログラム
ディスプレイへ文字などを表示するプログラム
ホストと通信するためのプログラム

などを詰め込む必要があります．すなわちスタート・プログラムの内容は厳しく制限されることになります．例えば，複数のドライバを書き込んでおいて起動時に選択することなどは，ファイル容量を考えると現実的ではありません．

ターゲットのメモリに，スタート・プログラムが読み込まれたとします．

スタート・プログラムは，まずディスプレイに表示を行い，ホストとの通信を開始します．

ホスト-ターゲット間の通信は，

シリアル通信　　通信速度は低速
LAN　　通信速度は高速

のどちらかを選択することになります．通信に必要なプログラムは，前もってフロッピ・ディスクへ書き込んでおきます．

シリアル通信，LANのどちらか一方のドライバがフロッピ・ディスクへ入ります．したがって，もし1台のターゲットをシリアル通信とLANを使い分けたいのであれば，2枚の起動用フロッピ・ディスクを用意する必要があります．

シリアル通信は，レガシのCOM1およびCOM2のどちらかを選択します．拡張の通信ポート（COM3，COM4など）やUSB接続のCOM1などは使用できません．

LANは，IBMがPCを発売開始したときにはまだ普及していなかったので，レガシと呼ぶ手続きは存在しません．

スタート・プログラムは，すべてのネットワーク・ボードに対応することはできないので，当然使用できるネットワーク・ボードは特定のボードに限られます．

ホスト-ターゲット間の通信において，xPC Targetが指定したネットワーク・ボード以外のボードを使用することはできません．これが，また問題を引き起こすことがあります．

現在，パソコン・ショップの店頭に並んでいるパソコン，組み立て用のマザーボード，いずれにおいても，LANの機能はあらかじめマザーボード内に組み込まれています．

ここにインストールされているネットワーク・ボードが，xPC Targetが指定するネットワーク・ボードと一致することは期待できません．

LANを採用したターゲット・システムの構築の詳細は2.6節において述べますが，まず最初にシリアル通信（2.5節または2.7節参照）を採用することを薦めます．

ターゲットのPCIバスには通常，A-D変換ボード，D-A変換ボード，カウンタ・ボード，ステッピング・モータ制御ボードなどのボードを差し込みます．

PCIバスに差し込むボードはプラグ・アンド・プレイを前提としているので，IOアドレス，割り込みレベル番号などはブランクになっています（参考文献1）．

すでに述べたように，xPC Targetのスタート・プログラムはフロッピ・ディスクへ書き込むので，容量に余裕はありません．ここに，プラグ・アンド・プレイの処理を書き込むことはできません．プラグ・アンド・プレイは，BIOSに任せる必要があります．このために，BIOSの設定において，

プラグ・アンド・プレイ　NO

と設定する必要があります．

BIOSのプラグ・アンド・プレイの設定をNOとすると，これはOS側においてプラグ・アンド・プレイの操作を行うことができないという意味なので，BIOSは，OSに代わってプラグ・アンド・プレイを実行します．さらに，PCIバス上のボードに対してIOアドレス，割り込みポート番号などを割り付けます．この設定を行うことによって，ターゲットのPCIバス上のボードが使用可能になります．

ターゲットのOSは，データを記録する際に当然パソコンのディスクへアクセスします．

このディスクをターゲットのファイル・システムと呼びます．ターゲットのファイル・システムは，IDEのディスクCまたはDのいずれかでなければなりません．これ以外のディスクは受け付けません．

また，ディスクのフォーマットはレガシのFATを必要とします．NTFSをフォーマットとしたディスクは使用できません．これも，ときに問題を引き起こします．

通常，Windows XPなどをインストールする際に，NTFSによってフォーマットします．

このような状況を考えると，ターゲットのファイル・システムはパソコンOSとは別のディスクへ設定することになります．

以上をまとめると，私が推奨するターゲットの選択条件は，

レガシのフロッピ・ディスク
シリアル通信COM1ポート
FATでフォーマットしたIDEのディスクD

となります．

以下の節において，必要な準備作業を具体的に述べます．

■ 2.2　開発システムのインストール

まず最初に，ホストに対して，開発システムをインストールする過程について述べます．

ホストは，Windowsパソコンとして使用するので，選択に際して考慮する条件は事実上ありません．適当なパソコンを選びます．私が用意したパソコンは，マイクロソフト社のWindowsをインストールしたデスクトップ型のパソコンです．

ノートブック型のパソコンは，いろいろな意味で使いにくい点があるのでここでは使用しません．

私が選択したパソコンは，マザーボード上に，

3.5インチのフロッピ・ディスク　1
シリアル通信ポート　2
ギガビットLAN　1

を有するパソコンです．

ホストのOSは，

Windows XP Professional Version 2002 SP2

です．OSのバージョンは，参考のために記載しました．バージョンを一致させる必要はありません．

ホストに対して開発システムをインストールします．

ここではホストに対して，MATLABプロダクト・ファミリ，

MATLAB	**(7.0.4)**
Simulink,	**(6.2)**
Real-Time Workshop,	**(6.2)**
xPC Target	**(2.7.2)**

をインストールしました．バージョンは参考のために記しました．

本書では，各プロダクトの標準的な機能を使うので，本書の内容を理解する目的のためにバージョンを合わせる必要はありません．

インストールの過程にとくに変った点はないので，その詳細は述べません．指示に従ってインストールします．上記の4本のプログラムをデフォルトの状態でインストールしたとします．

MATLABの作業は，通常，workフォルダにおいて行います（参考文献2）．

デフォルトのインストールを行うと，workフォルダはMATLABのシステム内に置かれます．

私の場合は，

C:¥Program Files¥MATLAB704¥work

となりました．

実際にチェックします．MATLABを立ち上げると，起動画面は**画面2.1**となります．

MATLABのコマンドラインから**画面2.2**に示すようにコマンドを入力して，起動直後のワークスペースの場所を確認します．

システム・ファイルとの混乱を避けるために，ワークスペースをMATLABシステム外に移動します．例えば，**画面2.3**に示すように，ローカル・ディスクCの直下にワーク・スペースを作ります．

MATLABのスタート時に新規に設定したワーク・スペースworkがデフォルトの設定になるように，MATLABの設定を変更します．

デフォルトのインストールを行うと，**画面2.4**に示すようにMATLABのアイコンがデスクトップ上に置かれます．

このアイコンを右クリックすると，**画面2.5**に示したようにコンテキスト・メニューが開くのでプロパティをクリックします．

画面2.6に示すように，プロパティのダイアログに作業ホルダの場所を書き込み，［OK］ボタンをクリックします．

デスクトップのアイコンのプロパティを使ってworkディレクトリを変更した場合は，かならず，そのアイコンからMATLABを起動します．