第5章

1. 詳細設計では各モジュールの内部を設計する

詳細設計工程では，前工程（基本設計）におけるモジュール分割の結果を受けて，各モジュールをプログラムとしてどのように実現するかを設計します．すなわち，各モジュールの機能を関数に分割したものに対して内部の論理構造を設計し，関数ごとにその詳細な仕様を作成します．各関数の果たすべき役割を明確にすると同時に，ほかの関数との関係もはっきりさせます．この工程の作業は，扱う粒度は違いますが，一般に認識されている「ソフトウェア設計」と同じ内容であると思ってもよいでしょう．

● 図表を使って関数の役割や周囲との関係を検討する

関数の果たすべき役割やほかの関数との関係を明確にするために，関数で行う処理を構造化チャートや状態遷移図，状態遷移表などで表現します．また，この工程で定義するそれぞれの関数が以下のようになっているかを確認することも極めて重要です．

- 果たすべき役割が明確か
- 内容が理解しやすいか
- ほかの関数との役割分担が明確か
- ほかの関数と比較して役割が大きすぎないか/小さすぎないか
- テストしやすい分割になっているか

前工程でモジュール分割を行ったときにはモジュールの設計が良さそうに見えていたとしても，実際にモジュールの内部を設計すると複雑になってしまうようなら考えものです．関数の内部処理が複雑になるとテストや保守が行いにくくなることが考えられます．必要に応じてモジュール分割の工程に立ち返り，モジュールの役割や関連を見直してみてください．

● ソフトウェアの品質を決める要因を押さえる

この工程は，主だったグローバル変数やその使い方，各関数の機能と入出力，その意味を決めるという，恐らく日ごろソフトウェアを開発しておられる皆さんがなじんでいる工程だと思います．この開発工程の良しあしはどこで決まるのでしょうか．

もちろん前工程の基本設計が悪ければ，この工程の成果物も悪いでしょう．だから基本設計がしっかりしていることが大前提です．また，この工程の作業はソフトウェア設計と同じ内容と思ってもよいでしょう．この，ソフトウェアを設計する際の指針をどう持つのかが，この工程の出来に大きく関係します．またそれ以外にも，設計の品質を決めるいろいろな要因があります．以下にいくつか具体的な要因を紹介します．

● 組み込みソフトウェアでは例外処理が重要

「アルゴリズム＋データ構造＝プログラム」[(1)]という名著がありました（インターネットで調べると廃刊になっていた）．詳細設計工程は，まさにグローバル変数を使ったデータ構造とその処理方法である関数のアルゴリズムを決めるところです[注1]．この良しあしが，成果物の良しあしを決める一つの要因です．

「アルゴリズム」という言葉で処理内容のどこまでを含むのかははっきりしませんが，筆者は，処理したい内容のメ

KEYWORD ── 詳細設計，グローバル変数，関数，アルゴリズム，データ構造，例外処理，割り込み，時間制約，関数仕様，レビュー

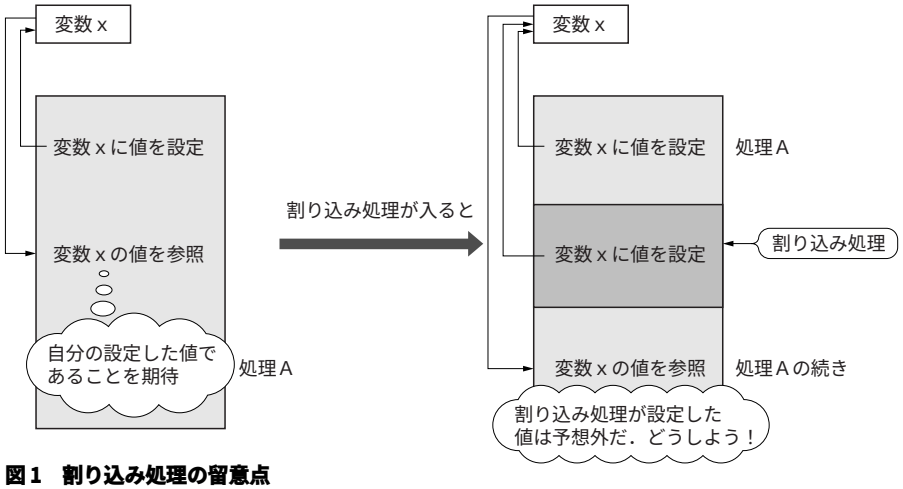

図1　割り込み処理の留意点

通常，プログラム内で変数 x に値を設定したら，その後 x を参照したとき自分の設定した値であることを期待する．しかし，割り込み処理が入る場合，処理 A にとっては思わぬ値を変数 x に設定されている可能性があり，このことに留意してプログラムを作る必要がある．

イン部分の手順を指しているように思います．組み込みソフトウェアではそういったメイン部分の手順もさることながら，エラー発生時の処理とシステム外部から非同期で入ってくる割り込み処理についての定義もそれと同等か，それ以上に重要です．そのため，「アルゴリズム＋データ構造＋例外処理＝プログラム」といってもよいと思います．

ここでは，「例外処理」にエラー処理と割り込み処理を含めています．割り込み処理は，メイン処理にとってはいきなり横から現れる「思わぬ処理」ですから，例外処理と見なしました．この例外処理をどこまできちんと設計に盛り込めるかが，成果物の良しあしを決めるもう一つの要因です．

● 例外処理を把握する

図1 に典型的な割り込み処理が絡んだときの問題を示します．割り込み処理でどの変数値が書き換わるかを把握した上で，書き換え禁止期間を設定するというような対策を盛り込む必要があります．また，エラー時の振る舞いがシステムとしてどうあるべきかという考察も重要です．基本設計段階で考えられていないエラーが詳細設計段階で出てくることも多くあります．モジュール内で決めてしまうのが適当ではないような例外処理は，もう一度基本設計工程のソフトウェア設計に戻って，モジュール分割と合わせて再考する必要があります．

● 時間制約を確認する

また組み込みシステムでは，時間に関する制約が存在する処理が結構あります．処理を行うにあたってのタイミングの制約をきちんと考察できるか，そしてそれを例えばテスト仕様を作成するメンバに伝えられるかといったことも，品質の良いプログラムを実現する上で重要になります．例えば，データのサンプリングは 10ms 以内に 1 度とか，あるレジスタを設定してから 10 サイクル以上後に結果の値を読み出さなければならないとか，いろいろな形の制約があります．例えば，処理 1 を実行した後の 20ms ～ 100ms の間に処理 2 を起動しなければならない，というタイミング制約があるとします（**図2**）．このことは，処理 2 を実現する関数に制約条件として引き継がれる必要があります．この制約については，ソフトウェア設計の段階で既に明記してあるかもしれませんが，詳細設計の段階でさらに詳細化した形で記載します．

● 時間を考察する

さて時間制約と前に書きましたが，制約の基準となる時

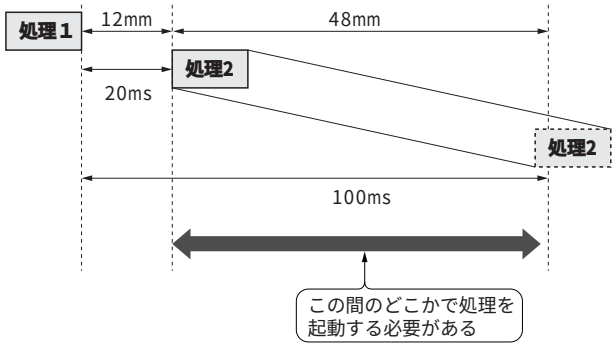

図2　タイミング制約のあるケース

「処理 1 を実行した後の 20ms から 100ms の間に処理 2 を起動しなければならない」というタイミング制約を図示した．

注1：少しインターネットで調べてみたら，「プログラム＝インターフェース＋アルゴリズム＋データ構造」という言葉を見つけた．これは，「アルゴリズム＋データ構造＝プログラム」はグローバル変数を中心としてプログラムを設計していた時代の考え方であり，現在の開発事情にはマッチしないとしている．そして，アルゴリズムやデータ構造がオブジェクトの中に隠ぺいしてしまっているため，オブジェクト間のインターフェースを中心に考えれば良いプログラムが書けるという考え方のようである．ただし組み込みソフトウェアでは，ナビゲーション・システムや携帯電話などの分野ならばいざ知らず，従来手法で開発されているシステムも依然として多いのではないかと思う．