第4章

基本的なディジタル回路を例にして

VHDLによる組み合わせ論理回路の記述

この章では，これまで紹介してきたVHDLの構文を使って，さまざまな組み合わせ論理回路を記述していきます．

構造記述と動作記述といった記述方法による違い，コンパイラの最適化によって生じる問題などについても触れておきます．

4-1 構造記述と動作記述を対比しながら 基本論理回路の記述

基本論理回路は，もちろんVHDLでは論理演算子として準備されているため，演算子を記述するだけで事足ります．

とはいえ，基本論理回路の記述法にも，従来の回路図から基本論理回路の組み合わせを考えて記述する「構造記述」と，真理値表や動作をもとに記述する「動作記述」があります．

いずれの記述法が優れているかは，ケースバイケースですが，構造記述は回路規模が大きくなっていくにつれ複雑度も増します．それに対して動作記述は，よりVHDLらしい抽象度の高い記述であると言えます．

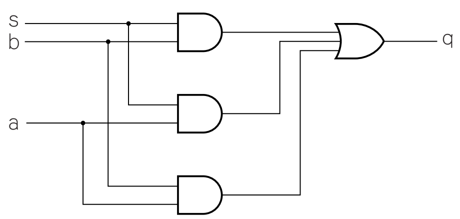
大きな回路の記述になると，動作記述で扱うほうがメリットが大きいのです．

ただ，同じ動作記述でも，真理値表を元にした記述は回路が大きくなりがちで，しかもあとで記述を見直すにも手間がかかるので，あまりお勧めしません．ある程度慣れが必要ですが，うまく整理して直感的でわかりやすい動作記述を心がけましょう．

さらに抽象度が高く，高レベルの動作記述を可能にする専用のツール（ソフトウェア）も登場しています．こうした最先端の動作記述ツールに比べれば，本書で扱う動作記述のレベルはずっと初歩的なものです．しかし，それでもゲート・レベルの構造記述に比べれば，十分に記述性は高いのです．

ここでは，上手な動作レベルの記述の習得を視

図1 AND/ORセレクタ



リスト1 回路図をもとにした構造記述の例
カルノー図などから簡略化した結果，Q＝A・B＋A・S＋B・Sという論理式が得られる．これをVHDLで記述したもの

```
library ieee;
use ieee.std_logic_1164.all;

entity ex4_1 is
    port( a, b, s : in std_logic;
          q : out std_logic );
end ex4_1;

architecture behavioral of ex4_1 is
begin
    q <= (a and b) or (a and s) or (b and s);
end behavioral;
```

Appendix-A デバイスやベンダによって相違がある VHDLによるメモリの記述方法

VHDLによって，ROM(Read Only Memory)やRAM(Random Access Menory)などのメモリを記述する方法について解説します．ターゲット・デバイスによっては制約があり，場合によっては思わぬところにメモリ回路が合成されていて，それが原因となって問題を引き起こす場合もあります．

そのような問題を解決するためには，VHDLでどのような記述によってROMやRAMが合成されるのかを知ることも大事でしょう．

● VHDLとメモリ

VHDLによって記述する回路は，一般にCPLDやFPGAなどのプログラマブル・デバイスのほか，ASICに実装されることをも想定しています．元来VHDLはターゲット・デバイスに依存しない回路記述言語ですが，それゆえ最終的に回路として実装する場合に，回路合成不可能な記述もできてしまうという問題があります．

その最もわかりやすい例がメモリです．

VHDLによってROMやRAMを記述することは簡単ですが，RAMはターゲット・デバイスによっては，物理的にきわめて不効率な実装になることがあります．

その理由は言わずもがなで，ターゲット・デバイスにRAMが存在しない場合，代替回路でRAMを作ろうとするからです．ROMも，場合によっては不効率になる場合があります．

具体的には，ここで紹介する方法は，ほとんどの場合，FPGA専用であり，CPLDやASIC向けではないということです．

そもそもメモリはどのような回路なのか，それをVHDLはどのように実装しようとするのか，そこのところがわかればメモリの記述に関する問題の半分以上は理解できたことになるでしょう．

▶ メモリの原理回路

最も基本的なメモリ回路は，フリップフロップを使った1ビット記憶回路です．大量にこれを配置し，同時に読み書きできるビット数(ワード構成)を決め，アドレスや読み書きコマンド制御信号を整備すれば，RAMとして利用できます．また，これに初期値を与えて読み出し専用にすればROMになります．

もちろん，このままでは必要となる回路規模に対して得られるメモリ容量が少なく，かなり効率の悪いメモリ回路となってしまいます．

実際にVHDLでこのような回路を記述すると，どのような回路が合成されるのか興味本位で設計してみました．FPGAに8ビット×256ワード，つまり256バイトのRAMをDフリップフロップで実装しようと試みたのです．論理合成してできた回路は図A-1のような，1753スライス，2048個のフリップフロップ，1675個のLUTを消費する巨大な回路でした．

16ビットCPUにテキスト・モードのビデオ表示回路，キーボード・インターフェース，タイマ/カウンタや割り込みコントローラ，DMA，ATAインターフェースを追加しても，ここまでのスライスは使いません．たった256バイトのRAMにしては，資源の無駄遣いです．

当然，コンパイラが変われば最適化によって状況は変わりますし，回路の記述をもう少しスマートにすれば消費スライス数を激減することも可能でしょうが，普通にフリップフロップを使ったのでは，やはりスライスを消費する以外に実装方法はありません．

ただし，これがメモリ回路の基本であることには違いありません．状態変数の記憶など，ごく小さな容量であれば，このような実装形態のほうがシンプルでわかりやすい場合もあるでしょう．実際，数バイト程度ならばRAMをもたないCPLDやASICなどのデバイスへも応用できます．

当然のことながら実際のメモリ・チップでは，もっと効率よくビット密度を上げられる独自の専用回路を使って実装されています．そのため，はるかに大容量なRAMやROMを得られるわけです．

▶ オンチップ・メモリ

一方，ターゲット・デバイスにRAMが搭載されているとすれば，どうでしょう？

FPGAは原理的にRAMの技

第6章
記述した回路を部品としてさまざまに使う

VHDLにおける階層設計

この章では，VHDLを使った階層設計について解説します．

階層設計によって，動作確認済みの回路をVHDLの「部品棚」へどんどん保管しておくことができます．実際の部品ではなくVHDLで記述した単なるテキスト・ファイルなので，場所を取ることもありませんし，検索や分類も楽です．また，何度も同じような回路を入力する手間も省けます．このように，動作確認済みの設計記述を組み合わせて，複雑な回路を組み上げる手法を階層設計と呼びます．

ある程度の規模の回路を設計する場合，全体をいくつかの機能別のブロックに分割し，ブロックごとの役割や仕様を決定したあと，ブロック間の接続や信号の仕様を決定する階層設計はよく使われます．それは回路図をベースとした時代から変わらず，HDLベースの回路設計でも有効な手法です．

6-1 設計を水平/垂直方向に展開できる VHDLによる階層設計とは

VHDLによる回路記述では，その記述ルール自体に機能別ブロックという考えかたが色濃く反映されており，エンティティ部で回路の外部仕様を記述し，アーキテクチャ部で内部の機能や動作を記述することが徹底されています．これらは小さな回路でも記述する必要があるため，やや煩雑感がありますが，階層設計のための重要な仕組みです．

エンティティ部に記述された外部端子は，そのアーキテクチャ部に記述した回路がICであるかのように扱うことができます．それを，ほかのVHDL記述の回路の一部から，端子に関する記述を通じて「接続」して利用するのです．

また，一つの機能ブロックの記述の中でも，特定の機能を一つの機能ブロックとして扱うための方法も提供されています．

一つのエンティティ内で，一部の回路記述を「サブルーチン」や「関数」として定義して，手続き化することで，繰り返し出現する回路の記述の簡略化や，パラメータ化した記述(パラメタライズ)を駆使した柔軟な機能の記述が容易になるわけです．

このような手法によってVHDLでは，図1のように設計を同レベルで展開したり，垂直方向に階層的に展開したりできます．もちろん，両方をミックスした設計も可能です．

図1 VHDLの同レベル展開と階層展開