実践的 VHDL設計 セミナ 3

ビギナーズ・スクエア

算術演算ライブラリの使い方

CD-ROM 設計データ EDAツール

竹本　悟

VHDLでは，よく使うデザイン・ユニットのタイプ宣言や関数の集まりを「パッケージ」として格納している．パッケージは各種用意される．また，複数のパッケージをまとめ，それらをコンパイルされたデザイン・データの集まりとしたものが「ライブラリ」である．ライブラリにも，VHDL言語で標準のもの，IEEEの標準化時に定めたもの，ASICベンダが用意したもの，など各種ある．さらに，ユーザが自ら定義して用意するものもある．初心者には難しいところだが，とくに算術演算を行いたいときに混乱し，トラブルの原因になりやすい．（編集部）

1　論理合成の一部として

◆算術演算ライブラリ std_logic_arith とは

算術演算ライブラリは，もともとVHDLの基本パッケージには含まれていない．しかし，回路設計者にとって算術演算ライブラリは，さまざまな演算を容易に記述するために必要な機能である．とくに加算や減算などは，カウンタの記述などで頻繁に使う．

VHDLでは，整数や実数やbit_vectorの演算を処理することはできるが，いま標準となっているstd_logic_vectorの演算の処理ができない．そこで別途パッケージを追加する必要があり，**std_logic_arithパッケージ**が用意された．

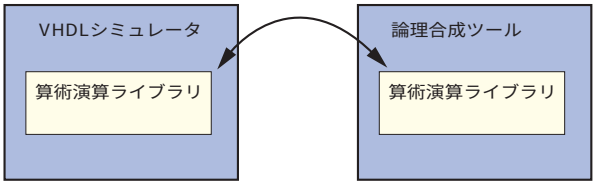
以前は，VHDLツール・ベンダごとに異なる算術演算ライブラリが存在した．それぞれは独立しており，その記述法も異なっていて，VHDLの移植性を損なう要因ともなっていた．そこで，統一した記述ができるように考えられたパッケージがstd_logic_arithなのである．

◆シミュレータとの整合性

算術演算ライブラリが問題になるのは，シミュレータと論理合成ツールとの整合性である．HDLによる開発環境では，複数メーカの各種ツールを組み合わせて使うことが一般的である．このとき，シミュレータの算術演算ライブラリと，論理合成で使う算術演算ライブラリを一致させておかなければ，合成時にエラーが多数発生する（**図1**）．

算術演算ライブラリが複数存在したとしても，少なくともプロジェクト内での算術演算ライブラリは統一しておく．この場合，論理合成で使う算術演算ライブラリが基準となる．なぜなら，合成ツールに付属している算術演算ライブラリには，その合成ツールに適したアトリビュート（属性）がいろいろと記述してあるからだ．そのため（動作の記述としては等価であっても），合成ツールでは自分自身の算術演算ライブラリを使わずにシミュレータ用のものを使うと正しく動作しないことがある．

最近ではIEEEのパッケージが標準となり，ほとんど意識する必要はなくなってきたが，それでも事前に確認は必要である．



〔図1〕算術演算ライブラリの一致

シミュレータと合成ツールは同じライブラリを使わなければならない．もし異なっているなら合成ツールのものをシミュレータに移植する．

2　ライブラリの位置づけ

◆ライブラリとパッケージ

VHDLでは，複数のタイプ宣言や関数などをまとめて記述する機能として**パッケージ**（package）がある．しかも，パッケージは個々にコンパイル可能である．個々のコンパイルしたパッケージをまとめて一つのディレクトリに収納したものが**ライブラリ**（library）で，これにより複数のentityから利用できるようになっている．

RTLでは，信号記述の標準データ・タイプとしてstd_logicを使うことを以前の回で述べた．しかし，std_logicは標準化されたVHDL言語として決められたタイプではない．そのため，いろいろな処理をする関数，たとえば加減算などが定義されていない．そこでパッケージを作って定義しなくてはならない．

標準的に各種宣言の集合体を用意したものが**IEEEライブラリ**である．IEEEライブラリは，いくつかのパッケージから成り立っている．すべてstd_logicまたはそのサブタイプであるsigned，unsignedの関数を記述している．

表1は，VHDLで回路を記述するときに使う主要なパッケージの一覧である．**STDライブラリ**は，VHDL言語のもともとの標準ライブラリで，VHDLの言語規約で定められたディレクティブを記述している．libraryでの宣言は必要のないパッケージである．一方IEEEライブラリは，std_logicを使って回路を記述するときに必要なライブラリである．論理合成を前提としたVHDLの記述を行うときには必須のライブラリである．ライブラリの中には演算の種類によっていくつかのパッケージが定義されている．

これらのパッケージは現在ほとんどのVHDLシミュレータおよび論理合成ツールにソース・コードと一緒に添付されている．多少ツールによって書き加えられているところはあるが，内容は同じである．

◆使用ライブラリの宣言

VHDLでライブラリ宣言をするとき，ライブラリの名前は，そのライブラリに所属するパッケージのデータが収まったディレクトリを意味する．どこを意味するかは，そのEDAシステムの構成に依存し，一定ではない．use文を使ってどのパッケージかを指定する．

普通，設計者はどのディレクトリにライブラリがあるかを意識する必要はない．ただし，パッケージがどんな機能をもつかは，よく意識していなければならない．気をつけないとまちがった結果を引き起こす場合もある．またuse文でいくつでも宣言してよいわけではなく，同時に宣言してはいけない場合も存在する．

3 設計例：16進カウンタ

実際の算術演算ライブラリの使い方を記述例を見ながら説明していこう．**リスト1**は16進カウンタの記述例である．この記述をみて

〔表1〕VHDLで使う標準的なパッケージ

ほとんどのVHDLシミュレータや論理合成ツールに標準で付属しているもの．

ライブラリ	パッケージ	用途
STD	standard textio	標準パッケージ テキスト入出力
IEEE	std_logic_1164 std_logic_1164_extension std_logic_arith std_logic_unsigned std_logic_signed ・ ・ ・	基本関数 拡張関数 算術演算 符号なし演算 符号付き演算

あれ！と思われる方もいるかもしれないが，これでもちゃんとした16進カウンタである．

◆演算するにはstd_logic_vectorタイプを変換する

カウンタの動作に使うフリップフロップ(以下FFと記述する)は，**Dout**という4ビットのstd_logic_vectorタイプで表現している．そして，クロックの立ち上がりエッジのたびにstd_logicであるDoutに整数“1”を足している．一見すると文法的にまちがっていそうだが，じつはこれで正しい．

コラムA ■サブプログラムのオーバロード

VHDLで使用するprocedureやfunctionなどの**サブプログラム**では，オーバロードが認められている．つまり，代入する引き数のタイプが異なっていれば，サブプログラムの名前が同じでも許されるということである(**図A**)．

たとえば，加算のファンクションを考えてみよう．“＋”の演算子は，加算の対象が整数であろうがbit_vectorであろうが同様に使われる．当たり前のように見えるが，じつは不思議なことである．加算の対象が違うなら，VHDLシミュレータの処理が当然違うはずなのに，なんの矛盾もなく受け入れられている．これを可能にしているのが**オーバロード**という仕掛けで，引き数のタイプによって呼ぶサブプログラムを区別している．これは，タイプの異なる変数に対して同じ演算を行うとき同じ名前を使えるので便利である．

std_logicを使った演算を行うとき，std_logicはあらかじめ決められていたタイプではないので，さまざまな演算(論理演算や算術演算など)をpackageで記述してやらなければならない．それを行うのがIEEEのライブラリである．**図B**の加算をstd_logicのタイプで行うときには頭にライブラリ宣言がなければならず，use文でのパッケージの指定も必要である．ただ，std_logicを使った演算パッケージは数種類あるので，あらかじめどれを使うか決めておかなければならない．複数のパッケージを一緒に宣言すると，このオーバロードが仇になり，コンパイル時にエラーになる．

この関数を使うときには引き数のタイプを明示的に示してやらなければならない．どちらの関数か判断できなければ，コンパイラは正常に処理できない．つまり，関数で使われている引き数のタイプをしっかり意識する必要がある．算術演算ライブラリには，このような関数が多く含まれる．引き数で使うタイプがはっきり確定するように書かないと，コンパイラはどちらの関数を呼んでいいか判断できなくなる．

```
function  add(a, b:std_logic_vector)return std_logic_vector;①
function  add(a, b:bit_vector) return bit_vector;②
:
  同じ名前の関数が存在するとき，コンパイラはその関数の引き数のタイプでど
  ちらを呼ぶかを決める．
《使用例》
     siganl  x, y:std_logic_vector(7 downto 0);
     signal  k, j:bit_vector(7 downto 0);

   X<=add(x, y);              --  ①の関数を呼ぶ
   K<=add(k, j);              --  ②の関数を呼ぶ

   B<=add(“00110011”, “11111111”);   --どちらを呼んでいいのか
                                      わからない
```

〔図A〕オーバロードの例

```
signal  A, B,C:integer;
siganal AV, BV, CV:bit_vector(7 downto 0);

    C<= A+B;                    --整数の加算
    CV<=AV+BV;                  --bit_vectorの加算
```

〔図B〕タイプの違う引き数で同じ演算子が使える