短期集中連載

デバイスの記事

SystemC TLM活用入門

第1回　トランザクションの考え方とFIFO

平尾智也，赤星博輝

最近，ハードウェア（LSI）の設計や検証の世界で，「トランザクション（Transaction）」という言葉を聞くことが増えてきた．例えば，検証の世界における教科書であるVMM（Verification Methodology Manual）[3]やAVM（Advanced Verification Methodology）[4]を見ると，トランザクションというキーワードがたくさん出てくる．本連載では，TLM（Transaction Level Modeling）の基本について説明する．第1回の今回は，トランザクションの意味について説明をした後，もともとSystemCにあったsc_fifoを改良したOSCI（Open SystemC Initiative）のTLM 1.0のtlm_fifoについて説明する．（筆者）

トランザクションは不可分なひとかたまりの処理のことで，いろいろな目的で利用されます．筆者の独断でトランザクションを使う理由を三つ挙げます．

- シミュレーションを高速にする
- モデリングを容易にする
- 再利用を容易にする

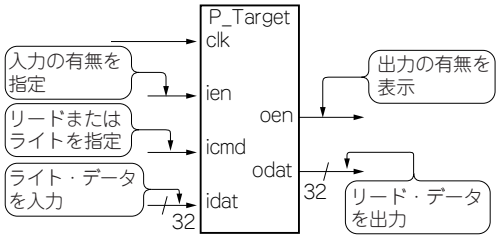

図1　RTLの設計モジュール

合成を意識したRTL（Register Transfer Level）の設計では，入出力がピン・レベルで設計される．

それではトランザクション・レベル・モデリングとは何か考えてみます．

1. トランザクション・レベルを理解する

図1のような入出力ピンを持つ回路を考えます．1ワードの記憶領域を持つモジュールです．外部からの読み出し（リード）と書き込み（ライト）を受け付けます．

ienが‘1’になったときのコマンド（icmd）が‘0’ならリードです．内部の記憶領域から値を読み出し，odatに出力するとともにoenを‘1’にします（**図2**）．ienが‘1’になったときのコマンド（icmd）が‘1’ならライトです．入力データの値を，記憶領域に対して書き込みます．

この回路の動作をSystemCで表現してみると，**リスト1**のようになります．入出力ピンの値によって，動作していることが分かります．

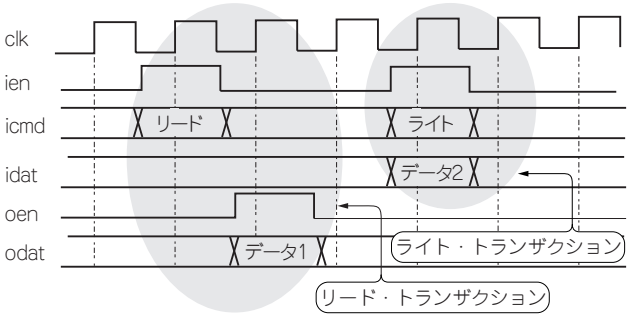

図2　ピン・レベルの入出力関係とトランザクション

ピン・レベルの設計では，入出力の波形を考える．不可分なひとかたまりの処理をトランザクションと呼ぶ．

Keyword　トランザクション，VMM，AVM，TLM，SystemC，OSCI，tlm_fifo，sc_fifo，ブロッキング，ノンブロッキング，FIFO

```
SC_MODULE(P_Target) {
  sc_in_clk   iclk;
  sc_in<bool> ien;
  sc_in<bool> icmd;
              // 0:read, 1:write
  sc_in<int>  idat;
  sc_out<int> odat;
  sc_out<bool> oen;
  int mem[MAX];

  void f() {
    odat =0;
    oen = 0;
    while(1) {
      while (ien.read()==0 ) {
                wait();
      }
      if ( icmd ==0 ) { // read
                  odat =      mem;
                  oen  = 1;
                  wait(1);
                  odat = 0;
                  oen = 0;
      } else { // write
                  mem = idat;
      }
      wait();
    }
  }
  SC_CTOR(P_Target) {
    SC_CTHREAD(f,iclk.pos());
  }
};
```

◀リスト1
ピン・レベルのSystemC記述
SC_CTHREADを用いて記述した例.

リスト2　トランザクションのためのインターフェース定義

関数readとwriteをインターフェースとして持つsimpleIFを定義した．この段階では実体は定義していない点に注意.

```
class simpleIF : public sc_interface {
public:
  virtual  int read  (  ) = 0;
  virtual  void write(int dat) = 0;
};
```

この回路の動作の本質を考えてみると，ien，icmd，idatを監視したり，oen，odatの値を出力することが目的ではありません．回路にデータを渡す(ライト動作)ことと，回路からデータを引き出す(リード動作)ことが，本質と考えることができます.

● 回路の本質がトランザクション

回路の本質である1回のリード/ライトの処理を「トランザクション」と呼びます．また，ポートではなくトランザクションを中心にモデルを作成することを，「トランザクション・レベル・モデリング」と呼びます.

このトランザクションという言葉は，データベースの世界などでよく利用されています．「分けることができない処理の単位」と考えればよいと思います．例えば，**図1**の回路でoenを'1'にするとか'0'にするということは，単体では意味を持ちません．**図2**の波形として"ひとかたまり"で意味があることが分かると思います.

● 言語仕様でTLMが取り入れられる

Verilog HDLやVHDLでは，入出力ピンを使ったモジュール接続が基本でした．このため，トランザクションといった単位のデータのやりとりは，あまり行われていませんでした．Verilog HDLやVHDLのトランザクション・レベル記述は，合成サブセットの範囲に限定しなければできないわけではありませんが(1)，現実にはあまり用いられていません.

新しい設計言語であるSystemCやSystemVerilogでは，言語仕様に「インターフェース」が取り入れられています．このインターフェースを利用することでトランザクション・レベルのモデルを作成することが容易になりました．インターフェースでは，入出力に信号を使用することも可能ですが，関数を使用することもできます.

● 関数による入出力のモデル化

関数を用いてどのように入出力のモデル化をすればよいかを考えてみます.

回路の本質は，データを書き込むトランザクションとデータを読み出すトランザクションです．そこで，データを書き込むために関数`write`とデータを読み出す関数`read`を用意します.

この二つの関数をインターフェースとして定義するためにSystemCでは，`sc_interface`を使って**リスト2**のように宣言します．このインターフェースでは通常使用する関数のひな形を定義しただけで，`read`/`write`の実体はありません．そのため，後でこの関数の実体を定義する必要があります(`virtual`と`=0`を使って定義された関数は，C++では純粋仮想関数と呼ばれる).

`read`と`write`の関数の実体は，モジュールで定義を行います[注1]．このインターフェースを使ったモジュールを設計したのが**リスト3**です．インターフェース`simpleIF`を

リスト3　インターフェースを持ったモジュール

sc_moduleに加えて，simpleIFを継承し，このモジュールで関数readとwriteの実体を定義し，必要な変数を用意している.

```
class Target : public simpleIF, public sc_module {
  int mem;
public:
  int  read( ) {return mem;}
  void write(int dat) { mem  = dat;   }
  Target(sc_module_name n) : sc_module(n) {  }
};
```

注1：インターフェースはチャネルで実装すると思った人もいるかもしれない．実は，`sc_module`と`sc_channel`は同じもので，SystemCのインクルード・ファイル中の`sc_module.h`の中に，"`typedef sc_module sc_channel;`"とある.