短期集中連載

デバイスの記事　ビギナーズ

SystemC TLM活用入門

第3回（最終回）　モデル作成の実際

平尾智也，赤星博輝

連載第1回（本誌2008年7月号，pp.103-112）と第2回（本誌2008年9月号，pp.139-148）では，OSCI（Open SystemC Initiative）のTLM 1.0について説明した．今回はTLM 1.0が実際にどのように使われているかを説明する．まず，実際にTLMが使われているSCML（SystemC Modeling Library）を見ていく．そして，TLM 2.0ではどうなってきているかについて，簡単に紹介する．（筆者）

ハードウェア，ソフトウェアに限らず，大規模な物を作成していくには，さまざまな決まり（独自ルールや標準ルールなど）を作って，再利用可能な部品を利用することがほとんどになると思います．再利用可能な部品ができると，それを使用するための道具も再利用できることになり，どんどん便利な環境が出来上がっていきます．

SystemCも同じことがいえます．今回紹介するSCML（SystemC Modeling Library）は，OSCI（Open SystemC Initiative）で標準化を行っていたTLM 1.0を利用して作成されています．さらに，OSCIから配布されているSCV（SystemC Verification）も利用しています．また，OCP-IP（OCP International Partnership Association, Inc.）のインターフェースなども利用しています．

SCMLは，2006年に米国CoWare社がオープン・ソースとしてリリースしたものです．ソース・コードやマニュアルなどは公開されています．CoWare社のツール上だけでなく，OSCIの配布しているリファレンス・シミュレータでも動作します．

SCMLがどのような点の機能を拡張したかについて知ることができる点も重要です．どのような点を考えて作られたかという視点でみれば，「トランザクション・レベルをどのように効果的に利用すべきか」や，「どういった点に注意すべきか」などの点が見えてくるかもしれません．

SystemCが利用される状況を考えてみると，

- 実際の機能が正しく動くかについて確認するFunctional View
- 良いアーキテクチャを探索するようなArchitect's View
- ソフトウェアを開発するProgrammer's View
- 実際のハードウェアとソフトウェアを組み合わせて検証を行うVerification View

が想定されています．そのために，SCMLの目的は，

- 高速なシミュレーション速度
- 効率的なモデル作成
- モデルの再利用性

が挙げられています．

個人的には，シミュレーション速度に関しては，このSCMLではトランザクション・レベルによるモデリングということが大きな影響を与えているといえます．今回は紹介しませんが，クロック・ベースのモデル用のライブラリなども用意されています．SystemCではRTL（Register Transfer Level）でモデリングを行うと，あまり効率良くシミュレーションができないため，シミュレーション速度を大きく落とすことになるのですが，そうしたものを改善するために`scml_clock_counter`，`scml_divded_clock`，`scml_clock_gate`などのライブラリが用意されています．

Keyword　OSCI，TLM，SCML，SystemC，トランザクション・レベル，スレーブ・モデル，イニシエータ，メモリ管理

本稿では，効率的なモデル作成という点では，スレーブ・モデルの作成およびマスタ・モデルの作成を解説し，SCMLを使うことでどのようにモデルが作成できるかについて見ていただきたいと思います．

1. スレーブ・モデルを作ってみよう！

スレーブ・モデルの基本構造を**図1**に示します．ターゲット・ポートPVtarget_portとレジスタやメモリを作成します．

● レジスタとメモリの作成

SCMLでモデル内にレジスタやメモリを作成するには，scml_memoryというライブラリを利用すると簡単に作成できます．詳細はマニュアル(2)を見てください．ここでは基本的なscml_memoryの使用方法を簡単に説明します．

リスト1のように，モジュール内ではscml_memoryの変数を宣言し，コンストラクタ(SC_CTOR)でサイズや使用するモード，初期値などを設定できます．内容にアクセスするときにも，通常の配列と同様にm_regs[i]でアクセスできます．これだけだと普通の配列を用いてもよいわけですが，後で述べるようにscml_memoryはSCMLを使ったモデルを簡単に作成できます．

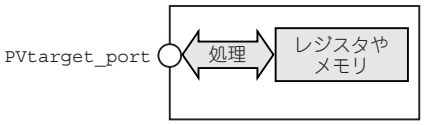

図1　SCMLのスレーブ・モデルの基本構造

PVtarget_portがtlm_transport_ifから継承して作成されている．そのtransportが呼ばれた際の処理を記述することで，レジスタやメモリの値を操作する．

リスト1　scml_memoryの使い方

高機能な配列として使用できる．

```
#include "systemc.h"
#include "scml.h"

SC_MODULE(target1){
  scml_memory <unsigned int> m_regs;

  void foo(void) {
    for (int i=0; i<10;i++) {
      m_regs[i]=i+100;
    }
    for (int i=0; i<10;i++) {
      cout << "m_regs[" << i << "]=" << m_regs[i] <<"¥n";
    }
    for (int i=0; i<10;i++) {
      m_regs.write( i+200,i);
    }
    for (int i=0; i<10;i++) {
      cout << "m_regs[" << i << "]=" << m_regs[i] <<"¥n";
    }
  }

  SC_CTOR(target1):m_regs("m_regs",  scml_memsize(10)){
    m_regs.set_addressing_mode(32);
    m_regs.initialize(0);
    SC_THREAD(foo);
  }
};

int sc_main(int argc, char**argv) {
  target1 u1("U1");

  cout << "sample program¥n";

  sc_start(1, SC_MS);
}
```

unsigned int型のscml_memoryを宣言

サイズを10で確保

アドレスを(バイトではなく)32ビットで定義する

内容を0で初期化

● ターゲット・ポートの作成

ターゲット・ポートは**リスト2**のように作られています．TLM 1.0であったtransportというインターフェースを持つインターフェースPV_ifを定義し，そのインターフェースをsc_exportで継承してターゲット・ポートPVTarget_portを作成しています．

ここでTLM 1.0から注意してほしいところがあります．SCMLではリクエストを扱うデータ型がPVReq<DT,AT>で，レスポンスを返すデータ型がPVResp<DT>になります．DTはデータ型で，ATはアドレス型です．例えば，データがintで，アドレスがunsigned intならPVReq<

リスト2　ターゲット・ポートPVTarget_portの定義

```
template <typename DT, typename AT = void>
class PV_if : virtual public tlm::tlm_transport_if<
PVReq<DT,AT>, PVResp<DT> >
{
  public :
    virtual PVResp<DT> transport(const PVReq< DT, AT
                                          >& arg_Req) = 0;
};
```

(**a**) PV_ifの定義

```
template <typename DT, typename AT>
class PVTarget_port: public sc_export< PV_if< DT, AT>
>
{
  private :
    PVTarget_process< DT, AT>* m_proc;

  public :
    PVTarget_port();
    explicit PVTarget_port( const char* name );

    ~PVTarget_port();
    template <class Module>
    void CB(Module* arg_module,
            PVResp<DT>( Module::*arg_cb)
                    (const PVReq< DT, AT >& arg_Req));
};
```

(**b**) PVTarget_portの定義