短期集中連載

デバイスの記事

SystemC TLM活用入門

第2回 インターフェースとチャネル

平尾智也，赤星博輝

前回（本誌2008年7月号，pp.103-112）は，TLM（Transaction Level Modeling）の中でもFIFOの部分を中心にご紹介した．今回はTLM 1.0のインターフェースと，あらかじめ用意されているチャネルを中心に解説する．また，TLM 1.0では，SystemCのsc_exportという観測・設定したいときだけ信号を接続できる機能を利用している．そこでsc_exportについても説明する．（筆者）

SystemCでモジュールの外部と通信をするとき，ポート接続の場合にはsc_in，sc_out，sc_inoutを，インターフェース接続の場合にはsc_portを利用しました．しかしこれらの接続には弱点があります．例えば，sc_in，sc_out，sc_inout，sc_portは未接続の状態ではシミュレーションを実行できず，エラーになります（この場合にはダミー配線を用意することで対応する）．

検証やデバッグでは，テストベンチ側から直接メモリやレジスタの値を観測・変更したり，値の変更があったときに新しい動作を起動したいことがあります．既存のポートでは，それらを効率良く行うことが難しい点がありました．こういった場合には，sc_exportを使うことで効果的に記述することが可能になる場合があります．

簡単な状態遷移を持った図1の回路を例にして，sc_exportで可能になることを見ていきます．

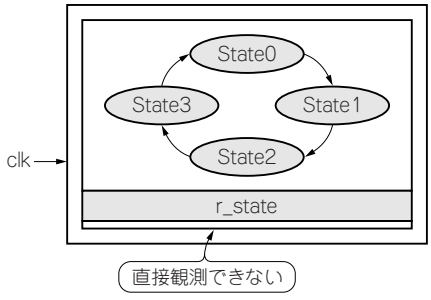

図1
状態遷移回路の例
四つの状態を順に遷移する回路を考える．

1. sc_exportの機能

リスト1は，図1の回路のVerilog HDLコードです．状態レジスタを出力ポートに接続していないので，外部モジュールから直接は観測できません．通常，入出力ポートは仕様として与えられており，勝手に追加できません．

内部の値によって入力する値を変えたり，トリガをかけたりする場合には，この状態変数r_stateをsc_exportの仕組みを利用して公開できます．このsc_exportの特徴は，観測・設定したいときだけ信号を接続できる点です．

リスト1
状態遷移回路のVerilog HDLコード

```
SC_MODULE(fsm) {
  sc_in<bool>    clk;
  sc_signal<char >   r_state;
  //
  void foo() {
    switch( r_state.read() ) {
    case 0:  r_state =1 ; break;
    case 1:  r_state =2 ; break;
    case 2:  r_state =3 ; break;
    case 3:  r_state =0 ; break;
    }
  }

  SC_CTOR(fsm):r_state(0) {
    SC_METHOD(foo);
    sensitive << clk.pos();
    //
  }
};
```

Keyword TLM，SystemC，ポート，インターフェース，sc_export，bind，チャネル，transport

● 観測・設定したい状態変数を外部に接続する

sc_exportによって状態変数を外部に接続可能にしたモジュールを**リスト2**に示します．

sc_expoortを使用する場合には，sc_exportのポートと内部の変数との対応付け（bind）を行う必要があります．ここでは，sc_exportのポートとしてsc_signal<char>の変数xp_stateとして定義します．このポートxp_stateと状態変数r_stateをbindで結び付けます．

リスト2　sc_exportとbindの記述例

sc_exportでポートを定義し，そのポートに対してbindで変数を結びつける．

```
SC_MODULE(fm) {
  sc_in<bool>    clk;
  sc_signal<char >   r_state;
  sc_export<sc_signal<char> >  xp_state;

  void foo() {
    switch( r_state.read() ) {
    case 0:  r_state =1 ; break;
    case 1:  r_state =2 ; break;
    case 2:  r_state =3 ; break;
    case 3:  r_state =0 ; break;
    }
  }

  SC_CTOR(fm):r_state(0) {
    SC_METHOD(foo);
    sensitive << clk.pos();
    xp_state.bind(r_state);
    dont_initialize();
  }
};
```

sc_exportによるポート定義

xp_stateと変数r_stateをbindで結びつける

● 上位モジュールからポートを利用する

今度は上位モジュールから，**図1**の状態遷移回路のsc_exportのポートを利用してみます．

リスト3のように，上位モジュール側ではsc_portの変数debug_fsmを定義します．そしてdebug_fsmとsc_exportのポートとを接続することで，上位モジュールでは状態変数を観測できます．今回は，debug_fsmの変化があると，SC_METHODであるprobeが実行されて，現在の状態を画面にダンプするようにしています．

● 接続しなくてもエラーにならない

これだけではsc_outを使った場合と同じです．次に接続をしない状況を作ってみましょう．**リスト3**の記述で接続に関する部分をコメント・アウトしたのが**リスト4**です．これをコンパイルして実行しても，モデルfsmのxp_stateの接続に関する実行時エラーは出ません．

sc_exportの換わりにsc_outを使った場合には，接続していなくてもコンパイルできるのですが，実行時にエラーになります．

● 4種類の接続が基本

これまでは信号に関するsc_exportを扱ってきました

リスト3　sc_exportへの接続例

fsmのsc_exportへの接続はsc_portの信号を用いて接続できる．debug_fsmのポートと，sc_exportしたu0->xp_stateを接続している．

```
SC_MODULE(top) {
  fsm *u0;
  sc_signal<bool> clk;
  sc_port<sc_signal<char> >  debug_fsm;

  void foo() {
    while(1) {
      clk = 0;
      wait(10, SC_NS);
      clk = 1;
      wait(10, SC_NS);
    }
  }
  void probe() {
    cout <<" FSM =" << int(debug_fsm->read()) << "  @"
<< sc_time_stamp() << "\n";
  }
  SC_CTOR(top){
    u0 = new fsm("FSM");
    u0->clk(clk);
    SC_THREAD(foo);
    SC_METHOD(probe);
    sensitive << debug_fsm;
    //
    debug_fsm(u0->xp_state);
  }
};
```

char型のsignalを接続するためのポート変数debug_fsmを定義

debug_fsmとモジュールfsmのxp_stateを接続

リスト4　sc_exportへの接続部をコメントアウト

これまでのsc_in，sc_outなどでは接続しないと実行時エラーになったが，sc_exportへの接続はなくても実行できる．

```
SC_MODULE(top) {
  fsm *u0;
  sc_signal<bool> clk;
  // sc_port<sc_signal<char> >  debug_fsm;

  void foo() {
    while(1) {
      clk = 0;
      wait(10, SC_NS);
      clk = 1;
      wait(10, SC_NS);
    }
  }
  void probe() {
    cout <<" FSM =" << int(debug_fsm->read()) << "  @"
<< sc_time_stamp() << "\n";
  }
  SC_CTOR(top){
    u0 = new fsm("FSM");
    u0->clk(clk);
    SC_THREAD(foo);
    // SC_METHOD(probe);
    // sensitive << debug_fsm;
    //
   // debug_fsm(u0->xp_state);
  }
};
```

sc_exportに接続する部分をコメント・アウトしてみる