FPGAでシリアルATAコントローラを設計する

第2回 シミュレーションおよび実機による シリアルATA物理層の動作確認

菅原 博英

連載第2回の今回は，前回（2008年7月号，pp.152-158）の解説で生成したシリアルATAのトランシーバ部分をシミュレーションさせる．そして，実機（Virtex-5 LXT）に実装して動作を確認する．（編集部）

1．シミュレーションでの確認

FPGA用に生成したトランシーバ・モデルの動作をシミュレーション上で確認します．通常の設計には，必ず他者（他社）に依存する部分のモデルと自分自身に帰する設計部分があります．まず他者依存部分を何らかの手段で再確認して，自分の理解が誤っていないかどうかを確認するプロセスが必要です．トランシーバ・モデルのシミュレーションや実機確認はこのプロセスに相当し，これを確認することにより，当該FPGAがシリアルATAに対応しているか否かを自分自身で再認識できます．

このテストにおけるFPGA_top配下の構成は**図1**のようになります．

PHYアナログ部（FPGAトランシーバ）とそれらを動作させる最低限のモジュール（PLLを含む），およびテスト用モジュール（PHY_ANA_TEST）を含みます．PLL（DCM_CLK）は通常の機能なので，各FPGAのWizardで生成します．これらはトランシーバと同じ理由で共通の外枠（wrapper）で囲みます．ポート名はベンダ標準の名称に合わせます．

phy_ana_test.vの主要部を**リスト1**に示します．テストの手順としては，

(1) TX_idle（リセット中，リセット解除後256サイクル）
→TXはハイ・インピーダンス（Hi-z）
(2) TX_start（次の1024サイクル）
→TXはD10.2送信
(3) TX_OOB（次の1024サイクル）
→TXはHi-zとD10.2を繰り返す
(4) RX_TRAIN（受信ロックまで）
→TXはD10.2，受信クロックの再ロックを行う
(5) RX_DONE0/1
（交互に繰り返す）
→TXは2パターン繰り返す

という構成になってます．

PHYアナログ部のポート数に対応して，phy_ana_test.vも二つ存在します．便宜上，SATA_HOSTとSATA_DEVICEと名付けていますが，全く同じ機能を対向させています．

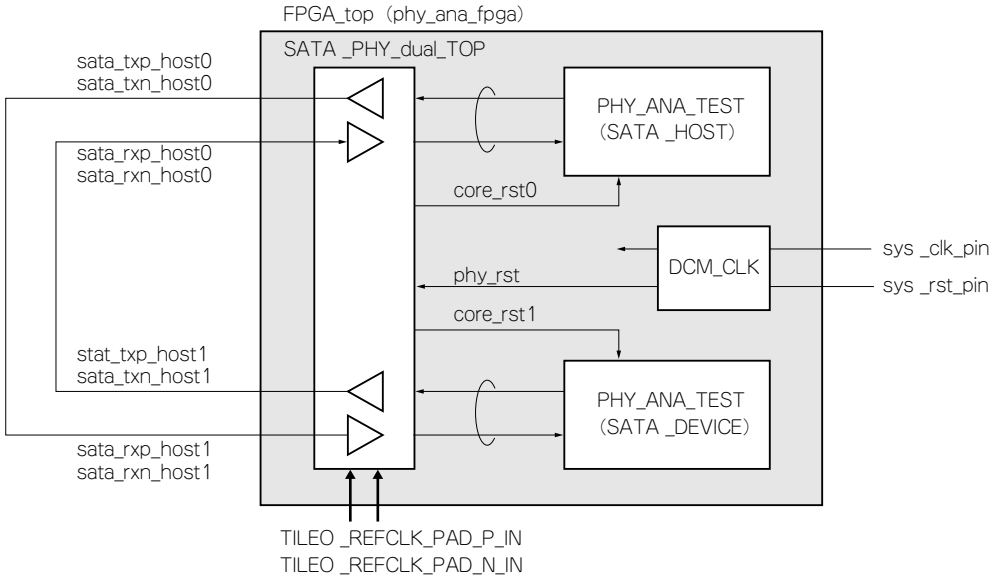

図1 物理層のテスト

リスト1　phy_ana_test.v の主要部

```
begin
//*-------------------------------------------------------*
   TDV=1' b0       ; TD=10' b0000000000    ;
   inc       = 0  ; clr     = 0       ;
//*-------------------------------------------------------*
    case(main_cs)

//-------------------------------------------------------
    TX_Idle:begin
         TDV=1' b0      ; TD=10' b0101010101    ;
         case({&count_q[7:0]})
         1' b0:begin main_ns=TX_Idle   ; inc=1' b1    ;end
         1' b1:begin main_ns=TX_Start  ; clr=1' b1    ;end
         endcase
         end

//-------------------------------------------------------
    TX_Start:begin
         TDV=1' b1      ; TD=10' b0101010101    ;
         case({&count_q[11:0]})
         1' b0:begin main_ns=TX_Start  ; inc=1' b1    ;end
         1' b1:begin main_ns=TX_OOB    ; clr=1' b1    ;end
         endcase
         end

//-------------------------------------------------------
    TX_OOB:begin
         TDV=|count_q[9:7] ; TD=10' b0101010101 ;
         case({&count_q[11:0]})
         1' b0:begin main_ns=TX_OOB    ; inc=1' b1    ;end
         1' b1:begin main_ns=RX_TRAIN  ; clr=1' b1    ;end
         endcase
         end

//-------------------------------------------------------------
    RX_TRAIN:begin
         TDV=1' b1      ; TD=10' b0101010101    ;
         case({I_PLL_LOCKED})
1' b0:begin main_ns=RX_TRAIN          ;end
         1' b1:begin main_ns=RX_DONE0          ;end
         endcase
         end
//-------------------------------------------------------------
    RX_DONE0:begin
         TDV=1' b1      ; TD=10' b1100110011    ;
                main_ns=RX_DONE1     ;
         end
//-------------------------------------------------------------
    RX_DONE1:begin
         TDV=1' b1      ; TD=10' b0001110001    ;
                main_ns=RX_DONE0     ;
         end
    //-----------------------------------------------------
    default:begin     main_ns = TX_Idle                   ; end
    //-----------------------------------------------------
    endcase
//*-------------------------------------------------------------*
    casex({clr,inc})
    3' b1x:begin count_d = 12' b0000_0000_0000           ;end
    3' b01:begin count_d = 12' b0000_0000_0001 + count_q  ;end
    3' b00:begin count_d = 12' b0000_0000_0000 + count_q  ;end
    endcase
//*-------------------------------------------------------------*
    O_PHY_TRAIN   =(main_cs==RX_TRAIN)         ;
    O_PHY_TDV= TDV                     ;
    O_PHY_TD  = TD                ;
//*-------------------------------------------------------------*
end
```

● Xilinx モデル

Xilinx 社のトランシーバでシミュレーションをする場合の問題は，swift モデルを使う必要があることです．例えば，プロセッサ用に比較的複雑なコアを混在させてシミュレーションする場合は，処理速度を稼ぐために別モデルを使用します．それが swift モデルです．Xilinx 社の場合はトランシーバのシミュレーションにもこのモデルを使用します．

一方，swift モデル（swift インターフェース）は比較的高価なシミュレータでしか動作させられません．ModelSim の場合はオプション付き PE バージョン，もしくは SE バージョンが必要となります．

今回は，SE バージョンを記事執筆用として借用できたので，このトランシーバの確認を行いました[注1]．

SE あるいは swift モデルを使用するには，ライブラリや環境ファイルの切り替えなどいろいろな手続きが必要です．Xilinx 社の Web サイト

1. % MODEL_TECH % ディレクトリの modelsim.ini ファイルを次のように修正する．

1. 1. 次の行を見つける．
　；Simulator resolution
　；Set to fs, ps, ns, us, ms, or sec with optional prefix of 1, 10, or 100.
　⇒ これらの行の後にある「Resolution = ns」を「Resolution = ps」に変更
1.2. 次の行を見つける．
　；Specify whether paths in simulator commands should be described
　；in VHDL or Verilog format. For VHDL, PathSeparator = /
　；for Verilog, PathSeparator = .
　⇒ PathSeparator = /」の行頭に「；」を追加してコメント文とする
1.3 次の行を見つける．
　；List of dynamically loaded objects for Verilog PLI applications
　⇒ この行の後に，次の文を追加する．
　Veriuser = $LMC_HOME/lib/pcnt.lib/swiftpli_mti.dll

1.4. 次の行を見つける．
　；Logic Modeling's SmartModel SWIFT software（Windows NT）
　⇒ この行の後に，次の文を追加する
　libsm = $MODEL_TECH/libsm.dll
　libswift = $LMC_HOME/lib/pcnt.lib/libswift.dll

必ず modelsim.ini に表示されるコマンドの順に変更すること．
順序が異なる場合シミュレーションが正常に機能しないことがある．

2. % LMC_HOME % ¥lib¥pcnt.lib が Path ユーザ変数にあることを確認し，なければ追加する．

図2　ModelSim（SE，PE）での SmartModel/SWIFT インターフェース使用法（抜粋）

注1：ModelSim SE