ST: Perl Package for Simulation and Test使用法

Kazutoshi Kobayashi (kobayasi at kit.ac.jp)

目 次

第1章　はじめに

ST は, シミュレーションならびにテスタのテストベクタを簡単に記述するための perl パッケージです. ST パッケージにはシミュレーション用の入力ベクタを作成するためのサブルーチンが定義されています. このサブルーチンを用いて, 汎用的なテストベンチを記述することができます. 記述したテストベンチは, SPICE や Verilog 等のテストベクタに変換することができます.

現在のところ, シミュレータとして, Spice 互換シミュレータ (spice, hspice, finesim, eldo), Verilog 互換シミュレータ, VHDL 互換シミュレータ, テスタとして, ヴェリジー社 (旧 HP, アジレント) の 83000, Hilevel 社の Griffin, アドバンテスト社の eva100 に対応しています. ベータ対応ではありますが, Synopsys 社の virsim(VHDL シミュレータ), SystemC にも対応しています. また, 論理合成, 配置配線ツールの合成制約を規程する sdc にも対応しています.

Perl のパッケージですので, シンタックスは Perl と全く同じです. Verilog, hspice 等のシミュレータでは, シミュレーションを実行することで, 期待値とシミュレーションの出力との比較を行うことができます.

第2章　ファイルの入手とインストール

2.1　配布先

配布先は, http://www-vlsi.es.kit.ac.jp/~kobayasi/ST/です.

2.2　実行環境

Perl が走る環境なら, UNIX, MacOS, Windows を問わず実行できます. 作者は Linux, MacOS Darwin, Windows10 上の CYGWIN 環境，Windows10 上の minGW にて動作を確認しております．Windows の Active Perl でも動くはずですが, インストールが自動化できません．Active Perl にて利用する場合には, 手動にてインストールをお願いします.

2.3　Cygwin/MinGW への make, perl のインストール

Windows の場合は, CYGWIN もしくは minGW をインストールします. その際, Cygwin は Devel 内の make と Perl，MinGW の場合も make と Perl がインストールされるようにしてください．UNIX 環境の場合は, まず間違いなく Perl と make がインストールされていますので, 何もしなくて結構です.

2.4　ST のインストール

使用中の perl のバージョンを確認します.

```
% perl -v
This is perl, version 5.005_02 XXXXXXXXXX
```

version が, 5.X 以上の場合は, ST を使用することができます．4.X の場合は, Perl 5.X をインストールします.

配布パッケージ ST_*.tgz (*はバージョンにより変わります) を下記のように解凍します.

```
# gnutar でない場合
% gzip -cd ST_*.tgz |tar xf -
#gnutar の場合
% tar xfz ST_*.tgz
```

ST_*(*はバージョン名) というディレクトリが作成されます. ディレクトリ ST に移ってインストール作業を行います.

2.4.1 UNIX の root でのインストール, CYGWIN でのインストール

UNIX/MacOS の場合は, su コマンドで, root ユーザーになります. CYGWIN の場合は不要です.

perl コマンドで Makefile.PL を実行します.

```
% cd ST_**
% perl Makefile.PL
# UNIX の場合，root になる．Windows の場合は管理者権限にて
% su root
# make install
```

ST のライブラリが使用する perl が標準で使用するライブラリパスの下にインストールされます. また, 既存の Verilog module からの ST テンプレートの生成節で説明する v2st.pl などが/usr/bin にインストールされます.

2.4.2 一般ユーザーでのインストール

一般ユーザーでインストールする場合は, アーカイブを展開した後, 次の通りにしてください.

1. 環境変数 PERL5LIB を”アーカイブを展開したディレクトリ/ST-バージョン/lib/”とする.
2. ”展開したディレクトリ/script”にパスを通す.
3. v2st.pl に, $INSTDIR を定義する.

```
$INSTDIR="/home/kobayashi/lang/perl/ST";
```

これにより, v2st.pl で作成したファイルは, 実行属性をつけるだけでそのまま実行できるようになる.

第3章　実行方法

ST 記述を格納したファイルを test.st とすると,

```
perl test.st
```

UNIX 環境の場合は, test.st の最初の行に,

```
#!/usr/bin/perl
```

と書きます. Perl のパスが/usr/bin/perl 以外の場合は, そのパスに合わせて先頭行を書き換えます. UNIX のコマンドラインで

```
chmod +x test.st
```

を実行した後に,

```
./test.st
```

を実行します. 通常は出力結果がそのままテストベクタとなります.

```
perl test.st > test.vec
```

もしくは

```
./test.st > test.vec
```

とすれば OK です.
-t オプションにより, 実行時に生成するターゲット名を変更することができます.

```
./test.st -t spice> test_sp.vec
```

を実行すると, test.st 中の target サブルーチンによる指定を spice に上書きすることができます.

```
./test.st: Command not found.
```

という表示がでた場合は, ファイルの先頭の

```
#!/usr/bin/perl
```

と, インストールされている Perl のパスが正しいことを確認してください.

第4章　実行時オプション

-a アーキテクチャ名. vhdl で architeture 名を指定する.

-b Hilevel テスタ用のベクタ生成で，バイナリ形式で出力する.

-d デバッグ出力を有効にする.

-D 複数チップ測定時に入力ピンを共有している場合に指定. 入力ピンが同じテスタチャネルに指定されていても無視する.

-f MMS の FPGA ボードの型番を指定

-m 複数チップを同時に測定するときのチップ数を指定する.

-n VHDL にて，ネットリストシミュレーションを実施．architecture 名の先頭に SYN_を付加する.

-p Verilog で電源ピン付きのシミュレーションのためのテストベンチを出力する.

-v Hilvel テスタ用のベクタ名を指定する．-v test とすると，test.trn が出力される．-v を指定しなければ，-m もしくは，$module で指定した名前.trn となる.

-w parsevcd で与える vcd ファイルを指定する.

-t ターゲット名. verilog, hspice, hilevel など

第5章　記述例

Perl で記述するので, Perl がサポートする任意の文を使ってテストベクタを書くことができます. 各サブルーチンの説明は, 9 章を見てください. また, チュートリアルを 6 章に用意しています.

リスト 1: 記述例

```
#!/usr/bin/perl
use ST; # ST パッケージ利用
level 0,5.0; # 入力レベルの指定
target "verilog"; # ターゲットの指定
module "calc"; # モジュール名の指定
channelfile "mot23-chip-dut.csv"; # テスタチャネル指定
verilog_logfile "alu_st.log"; # 期待値比較の結果の出力ファイル
shm "calc","AC"; # shm 波形出力の指定
pin "CE","input","defval=0","loc=5"; # ピンの指定
pin "CLK","clock","loc=7"; # CLK はクロック信号
pin "RST","input","defval=1","loc=6";
pin "decimal[9:0]","input","defval=0","loc=35:30 22 21 18 17";
pin "equal","input","defval=0","loc=8";
pin "minus","input","defval=0","loc=9";
pin "plus","input","defval=0","loc=16";
pin "out[6:0]","output","defval=0","loc=48:42";
pin "overflow","output","defval=0","loc=49";
pin "sign","output","defval=0","loc=50";
timing 1e-09,1e-08,10; # タイミングの指定．1ns の刻み幅で，10ns の周期．1 周期を 10 に分割
waveform "input","dnrz",".%........","CE","decimal","minus","plus","equal"; \
    # 波形の指定
waveform "input","r1",".%%%%%%%%.","RST";
waveform "output",".........%","out","overflow","sign";
clock "CLK","1111100000"; #クロック波形の指定
pinorder \
    "CE","RST","decimal","equal","minus","plus","out","overflow","sign";
#波形記述順の指定
beginvector;
vector  0,1,0,0,0,0,"X","X","X"; # pinorder で指定した順番に指定
svector "RST=0"; # 指定したピン以外はすべて defval で与えた値.
svector "decimal=3";
svector "out=3";
svector "decimal=2";
stimulus "plus=1"; #指定した値以外は前のサイクルの値を保存
stimulus "plus=0";
endvector;
```

第6章

第7章　シミュレーション/波形表示の方法

シミュレーションの方法は, ST より出力されたファイルとネットリスト, HDL 記述等をシミュレータに入力するだけである.

7.1　シミュレーションの方法

7.1.1　SPICE/HSPICE/finesim/ELDO の場合

ターゲット (target コマンドもしくは, -t) で, hspice, eldo を指定した場合, hspice, finesim, eldo の持つ論理値比較機能を用いて, 期待値とシミュレーション値の比較を行なうことができる. ターゲットで spice を指定すると期待値比較を行わない．finesim の入力ファイルは hspice と互換のため，-t で hspice を指定すれば良い.

ST からの出力とネットリストを.include コマンドによりメインのファイルから取り込めば良い. 例えば, ネットリストが net.cir で, ST からの出力が test.cir であるとすると,

```
.include 'net.cir'
.include 'test.cir'
```

を指定する.

SPICE の場合, 電圧入力はノードに電圧源を接続するので, 双方向ノードは実現不可能である. ST では, 電圧源とノードの間に, 電圧制御抵抗 (G element) を挿入することで, 双方向ピンを実現している.

7.1.2　Verilog の場合

ST からの出力とネットリストもしくは HDL ファイルを, Verilog シミュレータに与えるだけである. ただし, ST からの出力は 'timescale ダイレクティブにより, 時間の刻み幅が指定されているので, 必ず ST からの出力を最初の入力ファイルとして指定する. ネットリストが net.v で, ST からの出力が test.v である場合, xmverilog での実行は,

```
xmverilog test.v net.v
```

となる.

target を vcs もしくは modelsim とした場合，shm サブルーチンは無効になる．また, modelsim とした場合，シミュレーションの終了は$finish ではなく$stop となる．これは, modelsim の GUI 環境では，$finish を実行すると modelsim が終了してしまうからである.

7.1.3 VHDL の場合

testbench 名は，”testbench_モジュール名”となる．example/accumulator の fourbitaccum.vhd を vcs でシミュレーションする場合の手順は次のとおりである．

```
% perl fourbitaccum.st -t vhdl > fourbitaccum_test.vhd
% vhdlan -l fourbitaccum_test.vhdl.compile.log \
    fourbitaccum_test.vhd fourbitaccum.vhd
% vcs  -o fourbitaccum_test.exe testbench_fourbitaccum
% ./fourbitaccum_test.exe |& tee fourbitaccum_test.vhdl.log
```

7.2 波形表示の方法

現時点 (2021 年 11 月) においては, 次の波形表示ツールを使う.

cx (Custom Explorer) Synopsys 社の提供する hspice 用の波形表示ツール

gtkwave Verilog シミュレータから出力される vcd フォーマットのファイルに対応.

simvision Cadence 社の波形表示ツール．shm，vcd，tr0(ただし ASCII フォーマットのみ) に対応している．vcd を読み込むと shm 形式に変化するため読み込みに時間がかかる.

7.2.1 hspice

hspice では, 1

```
.options post
```

により, 波形ファイル (.tr0) が生成される．cx により, .tr0 形式の波形を表示することができる. ST では,

```
hspice_wave_format "ascii"
```

とすることで, simvision でも見ることが可能な ASCII 形式で出力することが可能となる.

7.2.2 verilog

Verilog の場合, vcd サブルーチンを使うと, vcd 形式で出力される. しかし, simvision で波形を見る場合, 一旦 shm 形式に変換されてしまうため, 時間がかかる．simvision で波形を見る場合, 変換の不要な shm 形式にて出力したほうが良い. したがって, vcd ではなく, shm サブルーチンを用いる.

なお, shm を用いる場合, シミュレーションのオプションとして, +access+r を付加しないと，波形が出力されないので注意する.

第8章　測定器，FPGAボード，LSIテスタへの対応

8.1　Advantest社EVA100での利用法

ターゲットをEVA100とする．moduleで指定した”モジュール名_EVA100”ディレクトリに下記のファイルが生成される．

DM64_FunctionalTest.ateliertext メインファイル

seq/DM64_FunctionalTest.seq Functional TestとDCPMUを含むテストシーケンスファイル

cond/DM64_FunctionalTest.cond 電圧やタイミングを指定するファイル

connection/DM64_FunctionalTest_connection.csv 信号名とピンの対応を取るファイル

pattern/MainPattern.csv 波形ファイル

8.1.1　複数チップ同時測定への対応

同じチップをDUT上に複数搭載して測定する場合には, -mオプションもしくは, multipleサブルーチンによりチップ数を指定する. その際には, channelfileにて指定するピンのテスタ側でのチャネル番号を与えるcsvファイルには, チップの数に対応したDUT番号を追加しておく. 詳細は, *unresolved reference*??に記載しているのでそちらを参照する．-mオプションや，multipleサブルーチンを使っての複数のLSIの一括測定にも対応している．-mオプションを使って複数のLSIを同時に測定する場合は，sourceで設定する電源もコンマ区切りで複数チャネルにアサインする．4個の同時測定の場合は，次のとおりとする．一つの電源を複数のチップに供給する場合は，channelに同じ番号を書く．

```
if($ST::_multiple==4){
    source "VCC1","channel=1,3,5,7","vh=$voltage","io";
    channelfile "../eva100_testx4.csv";
}else{
    source "VCC1","channel=1","vh=$voltage","io";
    channelfile "../eva100_test.csv";
}
```

8.1.2 制限事項

- loop コマンドで呼び出すサブルーチン内に記述できるベクタ数は 255 まで
- loop コマンドのループ回数は 2 の 16 乗 (16,777,216) 回まで

8.1.3 作成されるシーケンス

DCPMU と Functional Test (DM64_FunctionalTest) の 2 つのシーケンスが Flow Editor を開くと出てくる．DCPMU はテスタの IO ピンに DUT のピンが正常にコンタクトしているか確認するためのテストである．DM64_FunctionalTest は ST で作成したベクタを印加して正しい出力が出るかを確認するテストである．

8.2 MU300EM(em300) での利用法

ターゲットを mu300, em300 とすると, MMS 社の Power Medusa MU300EM の制御ソフトウエアである EVA III 用の各種設定ファイル, 入力ベクタ, 期待値を生成することができる．target を mu300, em300 とした場合の生成ファイルは次の通りである．

”モジュール名”.ini 一括設定用の設定ファイル

”モジュール名”_in.txt テストパターン (入力値)

”モジュール名”_io.txt 双方向制御パターン

”モジュール名”_header.txt 保存用ヘッダ

”モジュール名”_out.txt 入力値と出力の期待値

双方向ピンがない場合は，双方向制御パターン (io.txt) を MU300EM の制御ソフトウエア (EVAII) に読み込むとエラーとなる．双方向ピンがない場合は，io.txt は読み込んではならない．

8.3 Hilevel 社 Griffin での利用法

ターゲットを hilevel もしくは hilevelprg とすると, Hilevel 社 Griffin 用の各種設定ファイルを生成することができる．hilevelprg とすると, 同じベクタが多数続く場合に, program 機能を用いてベクタを短くするために, $LX が.vec ファイルに出力される. ただし, $LX が使えるのは, Compatible モードのみである．Native モード用には, うまく変換できない.

デフォルトの設定では, 次の二つのファイルを生成する．

”モジュール名”_pinlist.txt pinlist ファイル

”モジュール名”.vec ベクターファイル

.vec ファイルは ASCII 形式のベクターファイルであり, テスタに読み込む際には, バイナリデータに変換する必要がある. テスタが直接読み込めるバイナリ形式のデータも生成可能である. バイナリ形式でベクタを生成したい場合は, 次の通りベクタ生成の際に, -b オプションをつける.

```
% ./test.st -t hilevel -b
```

生成されるベクターファイルは, "モジュール名".trn となる. 生成されるベクターファイルをしてしたい場合は, -v オプションを付加する.

```
% ./test.st -t hilevel -b -v testvec
```

とすると, testvec.trn が生成される.

8.3.1 使用時に必須の設定

source コマンドにより最低一つの電源を指定しなければならない. また, IO レベルを与える電源に必ず"io"オプションを付加しなければならない.

```
source "VDD","channel=1","io";
```

上記の設定により, VDD がテスタの電源 1 となり, VDD が IO を与える電圧となる. VDD の電圧は何も指定しなければ, level コマンドで指定した値となる. 電圧を指定する場合は, "vh"オプションにより指定する.

```
source "VDD","channel=1","vh=3","io";
```

Symphony ソフトウエアの都合上, 外部電源を利用し, 外部電源から IO の電圧を与える場合でも, 内部電源の一つを IO 電圧と同じに定義しなければならない. この場合は, pinlist を読み込み後, Symphony 上で手動で電源電圧を切り替える.

8.3.2 波形ファイル名の指定

起動時に-v オプションによりベクターファイルのヘッダ名を指定することができる. -v test とすると, ベクタファイル名は test.vec となる.

8.3.3 複数チップ同時測定への対応

同じチップを DUT 上に複数搭載して測定する場合には, -m オプションもしくは, multiple サブルーチンによりチップ数を指定する. その際には, channelfile にて指定するピンのテスタ側でのチャネル番号を与える csv ファイルには, チップの数に対応した DUT 番号を追加しておく. 詳細は, *unresolved reference*??に記載しているのでそちらを参照する.

8.4 Altera FPGA への対応

Aletra FPGA のパッケージのピン名を”loc=”で記述することで，.qpf と.qsf ファイルに記述するピン配置を生成することができる．さらに，MMS(三菱電機マイコン機器ソフトウェア) の Power Medusa の一部のボードに搭載されている FPGA では，完全な.qsf を生成することができる．下記がその例である．mmsboardname により MMS の FPGA ボード名を指定する．現在対応しているのは MU500 と，MU200EC である．それ以外の FPGA も，ST_altera.pm を修正することで対応可能である．対応する FPGA の.qsf ファイルから該当部分をコピーして貼り付けても構わない．

```
mmsboardname("MU500");
module "counter";
verilog_logfile "counter_st.log";
pin "CLK","clock","loc=B12";
pin "RSTB","input","defval=0","loc=AB20";
pin "OUT[5:0]","output","defval=X","loc=C8,F8,A9,B9,E9,F9";
```

例を examples/counter_fpga に置いている．

```
% make counter.qpf
```

とすると，counter.qpf と counter.qsf を生成できる．Quarutus II が導入されたコンピュータでは

```
# make が導入されていない場合
% quartus_sh --flow compile counter.qpf
# make が導入されている場合
% make counter.sof
```

とすることで，SRAM Object File (SOF) ファイルを生成することができる．

第9章

第10章　期待値比較

ST では, Verilog, VHDL, hspice, finesim, eldo, SystemC のシミュレーション時に, 期待値比較を行なうことができる. Verilog, VHDL, SystemC では, 言語自体の持つ Syntax を利用して, 出力値と期待値の比較を行なうことにより, シミュレーションのみで期待値比較を行なうことができる. hspice, finesim, eldo は, 期待値比較の機能を内蔵しているためにその機能を利用している.

10.1　Verilog での期待値比較

Verilog では, 期待値との比較を行うことができる. この比較は Verilog の通常のシンタックスにより行われる. PLI による拡張ではないので, そのままの Verilog を使用することができる.

比較結果は st.log というファイルに, 期待値とシミュレーションとの比較結果が出力される.

期待値とシミュレーションが異なる場合, 以下のように出力される.

```
cycle(time): pin: exp. val != sim. result
         2(200): out:   2 !=1
         3(300): out:   5 !=6
```

2 が, サイクル数, 200 が, timing サブルーチンの最初の引数 (最小時間単位) を単位とした, 比較した時刻, out がピンの名前, 5 が期待値, 3 がシミュレーション結果となる.

期待値とシミュレーション結果が異なる場合,

```
Some simulation mismatches are detected!
```

が, また全く同じ場合は

```
No simulation mimatch is detected!
```

というメッセージが log ファイル (通常は verilog.log) に出力される.

なお, waveform サブルーチンで, window を指定しても, edge での比較となる.

```
waveform "output","window","........%%","out","overflow","sign";
```

は,

```
waveform "output","edge","........%.","out","overflow","sign";
```

と同じ意味になり, 1 サイクルが 100ns の場合, 80ns で値の比較が行われる.

10.2 HSPICE/finesim での期待値比較

HSPICE/finesim は, .dout 構文を用いることにより, 期待値比較が行なわれる. シミュレーション実行後の HSPICE の出力ファイルに下記の通りのメッセージが表示される. c0 はエラー無しであるが, それ以外の C1 から c4 は, 490ns に期待値との相違が起こっている

```
 *****Output vector error report*****
   output signal at node[(c0     ]:
     verified with no error.
   output signal at node[(c1     ]:
**warning**:incorrect logic state at node[(c1 ] time= 0.4900E-06, low \
    expected!
   output signal at node[(c2     ]:
**warning**:incorrect logic state at node[(c2 ] time= 0.4900E-06, low \
    expected!
   output signal at node[(c3     ]:
**warning**:incorrect logic state at node[(c3 ] time= 0.4900E-06, low \
    expected!
   output signal at node[(c4     ]:
     verified with no error.
```

10.3 eldo での期待値比較

eldo では，.setbus, .checkbus コマンドにより期待値比較が行える．"-t eldo"でベクタを作成したさい，下記の通りの期待値比較記述が出力される．比較される名前は，ピン名が OUT の場合，OUT_bus となる．

```
.setbus OUT_bus OUT[7] OUT[6] OUT[5] OUT[4] OUT[3] OUT[2] OUT[1] OUT[0]
.checkbus OUT_bus VTH=0.4 VTH2=0.6 BASE=HEXA
+1.950000e-08 0
+2.450000e-08 1
+2.950000e-08 2
+3.450000e-08 3
+3.950000e-08 4
```

シミュレーション後のログファイルに，下記の通り，出力されていれば期待値とシミュレーション値が一致している．

```
CHECKBUS PASS on OUT_BUS
```

一致しない場合は下記の通り，表示される．

```
0**** CHECKBUS INFORMATION  TEMPERATURE =   27.000 DEG C
0*********************************************************
at time 7.000000e+01 ns bus OUT_BUS is 0A and should be B

CHECKBUS FAIL on OUT_BUS

Number of checkbus tested       : 12
Number of checkbus passed       : 11
Number of checkbus warnings     : 0
total number of checkbus errors : 1
```

第11章　既存のテストベンチからSTへのフィードバック

すでに存在するVerilogテストベンチに, STのテストベクタを出力するコマンドを埋め込んで出力されたベクタをSTにかけることで, 既存のテストベンチからST側に情報をフィードバックすることができる.

11.1　利用方法

利用するには, vectorサブルーチンを除いたSTの記述 (beginvector, endvectorサブルーチンの間に何も書かなくて良い) を作成し, ターゲットとして, v2stを指定して, 一度STを実行する. すると, 既存のテストベンチに埋め込むためのVerilogファイルが出力される.

```
./test.st -t v2st > inc.v
```

ここで得られたinc.vを既存のテストベンチに下記のように挿入して, Verilogシミュレーションを実行する.

```
// module, endmodule 間に挿入する
‘include ’inc.v’
```

すると, st.outというファイルが得られる. これは, vectorサブルーチンが並んだファイルであるので, これを元のSTファイルのbeginvector, endvectorサブルーチン間に挿入する.

11.2　注意点

1サイクル中のwaveformで指定した点で, 入出力ピンの値を得る. したがって, 波形を確実に得られる点を指定すること. waveformでの指定は入力ピンは, dnrz, 出力ピンはedgeで行い. 1サイクル中にただひとつの%を指定すること. 得られたst.outを挿入する側のSTファイルのwaveformは, 正しい波形, ストローブ点を指定すること.

第12章　ASCII形式への出力

ST のターゲットとして, “dump” を指定して実行することにより, ASCII 形式にて出力する.

ダンプファイル名.in.dump 入力データのダンプ

ダンプファイル名.out.dump 出力データのダンプ

”ダンプファイル名”は, dumpfile サブルーチンで指定する. dumpfile サブルーチンが指定されていない場合, module サブルーチンの値となる.

下記のような形式にて, 出力される. name: 行は, ピンの名前, bit: 行は各ピンのビット幅を表している. その後に, 1 サイクル 1 行で各ピンの値が出力される.

リスト 2: ダンプファイル

```
name: CLK_CONF,RST_X,DIN,LINE,BLOCK,CTR,IO_W,IO_S,IO_E,IO_N
bit:  1,1,1,4,4,1,4,4,4,4
1,0,1,0,1,0,0,0,0,0
1,0,0,0,1,0,0,0,0,0
```

12.1　dumpfile サブルーチン

ASCII 形式の出力ファイルの名前を指定する.

```
dumpfile "ファイル名";
```

12.2　dump_pinorder サブルーチン

```
dump_pins "ピン名 1", "ピン名 2", "ピン名 3",.....;
```

ASCII 形式にて出力したいピン名を列挙する. 出力ファイル内のピンの並びは pinorder サブルーチンで指定した順番になる.

dump_pinorder で指定されていない場合は, 全ピンが出力される. すべてのピンの信号を出力したくない場合に dump_pins を使用する.

12.3　バイナリ形式への変換

dump ターゲットにより ASCII 形式で出力したファイルをバイナリ形式に変換することができる. この変換には, mem2bin.pl を利用する. 使用方法は下記の通りである.

```
% mem2bin.pl -B バイト幅 アスキー形式入力ファイル >バイナリ形式出力ファイル名
```

-B は, 1 サイクルで利用するデータのバイト幅である. 入力もしくは出力ピンの数が-B で指定した数の 8 倍以上になってはならない.

出力ファイルには, 入力ファイルに記載されている最初のピンが, LSB となる. リスト 2 に示すファイル (test.in.dump) を次のように変換する.

```
% mem2bin.pl -B 4 test.in.dump > test.in.dat
```

第13章　その他

13.1　内部変数を使った制御

target により, ST の動作を変えたい場合などには, 内部変数を使って, その動作を変更することができる. target コマンドで指定した値は, $ST::_target に格納されている. したがって, シミュレータにより動作を変更したい場合は,

```
if($ST::_target eq "verilog"){
  Verilog の時の動作
}else{
  それ以外の時の動作
}
```

とすれば良い. その他の変数は, ST.pm を参照すればすぐにわかる.

13.2　既存の Verilog module からの ST テンプレートの生成

既存の Verilog module から ST のテンプレートファイルを生成するための, v2st.pl を用意している.

```
v2st.pl [-c クロック信号名] [-m モジュール名] [-t ピン位置指定テキストファイル] verilog
ファイル > test.st
```

のように実行することにより, 既存の Verilog file から簡単に ST のテンプレートファイル test.stl を作成することができる. -c オプションで, クロックピンを指定する. 複数のクロックが存在するときは, コンマで区切る. -m オプションはモジュール名を指定するが, 省略すると, ファイル中で最初に見つかったモジュールの ST テンプレートが出力される.

-t オプションは, チップレベルの配置配線の際に用いるピン位置指定ファイルを指定する. ピン位置指定ファイルは, 各行番号が電源, グラウンドピンを除いたチップ上のピン番号に対応しており, このピン番号を”loc=”オプションとして出力する.

13.3　既存の VHDL 記述からの ST テンプレートの生成

vhdl2st.pl にて，VHDL 記述から，ST 記述を生成することができる．-c オプションでクロックピン，-e オプションで entity 名を指定する．

```
vhdl2st.pl [-c クロック信号名] [-e entity 名 (指定しなければ最初の entity)] [-p \
    perl のパス (デフォルトは/usr/bin/perl)] VHDL ファイル > test.st
```

13.4 sdc への対応

ST より, Synopsys 系ツールのタイミング制約を記述する sdc(Synopsys Design Constraint) ファイルを作成することができる. ターゲットとして sdc を指定することにより, sdc ファイルが作成可能である.

第14章　制限

14.1　想定している設計フロー

ST で想定しているのは, 大学で設計される LSI のように, 設計, シミュレーション, テストを自前ですべて行わなければならないような環境です. また, ちょっとした回路記述を気軽にシミュレーションして見たいというような場合に非常に便利です. ただし, プロセッサ等の自律的に動作するものには向いていません.

テスタによるテストは良品, 不良品の分別に使うようなテストではなく, あくまでも試作 LSI の動作確認用であると仮定しています.

Perl を用いてテストベンチを生成していますので, 動作はそれほど早くはありません. ループ等を用いて大量のテストベクタを生成する場合には, 非常に時間がかかります.

14.2　信号値

ST で扱える信号値は, ディジタル信号です. 0 は, ローレベル, 1 はハイレベル, X は不定値, Z はハイインピーダンスを表します. X,Z は大文字で指定します. 多ビットのバス信号に対しては, 任意の正の整数を使用することができます.

14.3　信号のビット幅

Perl の制限で, 32 ビット以上のバスは扱えません. 64 ビットバスを持つ回路をシミュレーションする場合は, 32 ビットずつに分ける等の処置を行ってください.