はじめてみよう！テストベンチ

概要

デザインの論理検証の重要性とテストベンチ

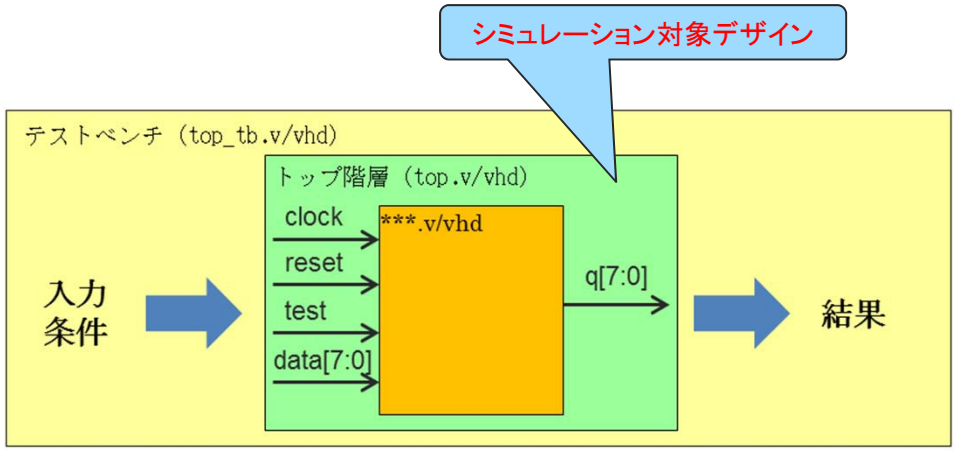
FPGA 内のデザイン（回路）は Verilog-HDL や VHDL といったハードウェア記述言語（HDL）で設計するのが主流になっていることは皆さんもご存じのことと思います。コンパイルや実機デバッグの前に、HDL で設計したデザインが正しく動作するかを確認するための検証作業が必要となります。デザインの論理的な検証はシミュレーションを実施することになりますが、シミュレーションをするにはデザインへの入力条件（テスト条件）を記述したテストベンチがデザインとは別に必要となります。そして、そのテストベンチも主に設計者がデザインと同様に HDL で記述することになります。

テストベンチを作る前には、どのような検証を行うかを考える必要があります。また、検証対象のデザインがどのような動作をするべきかを知っていなければなりません。それらをきちんと把握した上で、テストベンチを作成してください。

ここでは、はじめての方にもわかるようなテストベンチの作成に最低限必要となりそうな内容に絞って、記述例も交えて説明していきます。

テストベンチとデザイン（回路）の関係

FPGA 内のデザイン（回路）を設計するには、階層設計手法が主に用いられます。テストベンチは階層的にいうと、デザインのトップ階層よりも上位の階層に位置します。テストベンチ内でデザインのトップ階層に対する入力条件（クロック入力やリセット入力、その他の入力など）を HDL で記述して、下位階層に位置するデザインのトップ階層に入力させます。入力させた信号が各回路を経由して結果が出力され、その結果をシミュレータで確認することで動作が正しいかどうかを判断します。



ここからは、Verilog-HDL と VHDL のそれぞれにおける具体的なテストベンチの記述方法を解説します。なお、テストベンチのモジュール名（Verilog-HDL）やエンティティ名（VHDL）には _tb や _sim と付けているケースを多く見かけますが、これは決してルールではありません。自由に名前を決めてもらって大丈夫です。

Verilog-HDL 編 ➔ ページ6へ　　VHDL 編 ➔ ページ18へ

コーヒー・ブレイク：シミュレータ

設計したデザインをシミュレーションするには、シミュレータが必要です。アルテラ・ユーザ向けには ModelSim-Altera をオススメします。ModelSim-Altera の使い方については、公開中の資料をご覧ください。

- **ModelSim-Altera Edition - RTL シミュレーション**

  http://www.altima.jp/members/p1-literature/1-software/1-altera/2016004_msae_rtlsim.cfm

コーヒー・ブレイク：NativeLink

アルテラの FPGA/CPLD 開発ツールの Quartus Prime 開発ソフトウェアは、EDA シミュレータを Quartus Prime から実行させてシミュレーション結果を表示させるところまで自動実行させることができます。この NativeLink 機能を使用することで、シミュレータの起動やコンパイル、デザインのロードなどの作業をわざわざ手動で実行する手間が省けます。

但し、事前にテストベンチを作成しておく必要があるので、それをお忘れなく...

- **Quartus Prime はじめてガイド – EDA ツールの設定方法**

  http://www.altima.jp/members/p1-literature/1-software/1-altera/2016022_qp_bgnr_edasetting.cfm

※ *EDA：電子機器や半導体など電子系の設計作業を自動化し、支援するためのソフトウェア、ハードウェアおよび手法の総称。*

はじめてみよう！テストベンチ

Verilog-HDL 編

モジュールの書式

Verilog-HDL の場合、テストベンチを作成する時もデザインを作成する時と同じように module <モジュール名> から始めます。しかし、一般的にテストベンチには入出力ポートが存在しないため、モジュール名の後にポート・リストを記述する必要はありません。また、入出力ポートの宣言も必要ありません。従って、各種宣言や機能記述のところには検証対象となるデザインへ入力する信号の入力条件を HDL で記述していきます。

```
module top (clock, reset, test, data, q);

  各種宣言
  機能記述

endmodule
```

デザインの記述例

テストベンチでは入出力ポートがない！
（ ）省略可

```
module top_tb ();

  各種宣言
  機能記述

endmodule
```

テストベンチの記述例

モジュールの書式(詳細) & 信号の宣言

「各種宣言と機能記述」をもう少し細かくすると、「**信号の宣言**」と「**下位階層（デザインのトップ階層）の呼び出し**」、「**下位階層（デザインのトップ階層）への入力条件の記述**」に分けることができます。

「**信号の宣言**」のところは、先ほども触れたようにテストベンチには入出力ポートが存在しないので、input や output、inout を使用したポートの属性を宣言する必要はありません。テストベンチではテスト入力として使う信号のデータ型を **レジスタ宣言（reg 宣言）**し、反対に検証対象の出力ポートに接続するテストベンチで値を与えない信号のデータ型は **ワイヤー宣言（wire 宣言）**します。

```
reg          clock, reset, test;
reg   [7:0]  data;
wire  [7:0]  q;
```

```
module top_tb ();
  各種宣言
  機能記述
    信号の宣言

    下位階層（デザインのトップ階層）の
    呼び出し

    下位階層（デザインのトップ階層）への
    入力条件の記述
endmodule
```

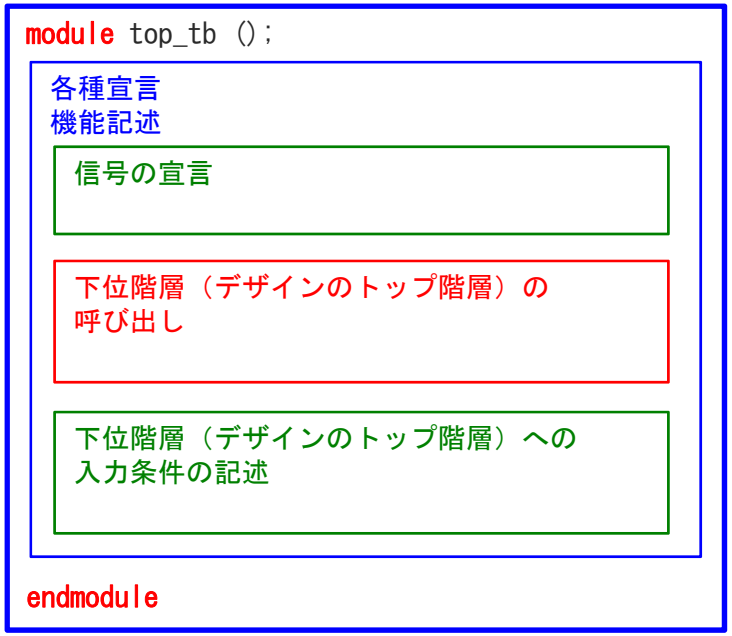
テストベンチの記述例

下位階層(デザインのトップ階層)の呼び出し

「下位階層（デザインのトップ階層）の呼び出し」のところは、デザインで一般的に使用する下位階層の呼び出しの記述とまったく同じです。

※ ここでは、デザインのトップ階層のインスタンス名を仮に “u1” としています。

```
top u1 (
  .clock (clock),
  .reset (reset),
  .test (test),
  .data (data),
  .q (q)
);
```



テストベンチの記述例

下位階層(デザインのトップ階層)への入力条件の記述

次に「**下位階層（デザインのトップ階層）への入力条件の記述**」について解説します。

ここでは、主に、

① 常にレベルが固定している信号の記述

② 一定間隔で値がインクリメントする信号の記述

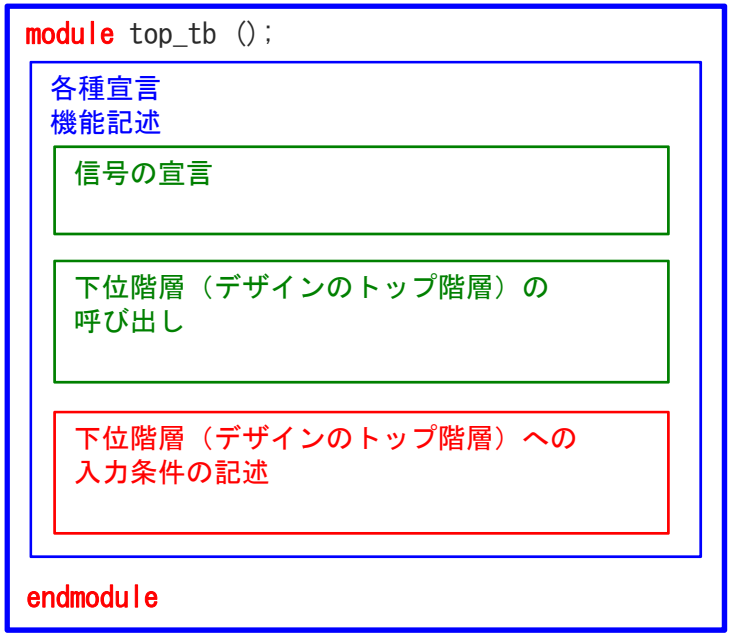
③ 非定期的に‘1’（H レベル）と‘0’（L レベル）を繰り返す信号の記述

④ 定期的に‘1’（H レベル）と‘0’（L レベル）を繰り返す信号の記述

について説明します。

これらをマスターすれば、テストベンチを自分で書けるようになります。

なお、ここで紹介した以外の記述方法もありますので、後々勉強して習得してください。



テストベンチの記述例

下位階層(デザインのトップ階層)への入力条件の記述

①常にレベルが固定している信号の記述

シミュレーションの開始には、initial 文を記述します。記載した式が1度だけ実行されます。シミュレーション実行中にレベルの変化がない信号は、initial 文に続いて信号名とレベルを記述すれば良いです。

```
initial test = 1;
```

下位階層(デザインのトップ階層)への入力条件の記述

②一定間隔で値がインクリメントする信号の記述

まず、initial begin の後に初期値を記述します。その後 always 文を使って一定間隔でインクリメンタルする式を記述します。

```
initial begin
  data <= 8'b00000000;
end
always #100000;
  data <= data + 1;
```

100000 タイムスケール後に 1 ずつインクリメント

下位階層(デザインのトップ階層)への入力条件の記述

③非定期的に '1'(H レベル)と '0'(L レベル)を繰り返す信号の記述

非定期的にレベルが変化する信号は、1つの initial 文に begin ～ end を使って、その間に複数の式を記述します。

```
initial begin
        reset = 0;
  #100  reset = 1;
#100000 reset = 0;
  #100  reset = 1;
end
```

100 タイムスケール後にレベルが 0 ⇒ 1 へ変化
100000 タイムスケール後にレベルが 1 ⇒ 0 へ変化
100 タイムスケール後にレベルが 0 ⇒ 1 へ変化

バスなどの複数ビットがまとまった信号は、以下のように記述します。8ビットの信号であれば、8'b の後に各ビットのレベルを記述します。b は 2進数の意味です。h を使って 16進数表示も可能です。

```
initial begin
        data = 8'b00000000;
 #10000 data = 8'b01010101;
 #50000 data = 8'b10101010;
 #20000 data = 8'b11111111;
end
```

10000 タイムスケール後に 00000000 ⇒ 01010101 へ変化
50000 タイムスケール後に 01010101 ⇒ 10101010 へ変化
20000 タイムスケール後に 10101010 ⇒ 11111111 へ変化

```
initial begin
        data = 8'h00;
 #10000 data = 8'h55;
 #50000 data = 8'hAA;
 #20000 data = 8'hFF;
end
```

10000 タイムスケール後に 00 ⇒ 55 へ変化
50000 タイムスケール後に 55 ⇒ AA へ変化
20000 タイムスケール後に AA ⇒ FF へ変化

下位階層(デザインのトップ階層)への入力条件の記述

④定期的に '1'(H レベル)と '0'(L レベル)を繰り返す信号の記述

クロックのように定期的に '1'(H レベル)と '0'(L レベル)を繰り返す信号を記述する時は、先ほどのような方法で永遠に記述する必要はなく、以下のように記述します。ここでは、2つの記述方法を紹介します。どちらも同じ入力条件になります。

- 方法1

```
always begin
        clock = 0;
  #5000 clock = 1;
  #5000;
end
```

5000 タイムスケール後にレベルが 0 ⇒ 1 へ変化
5000 タイムスケール後にレベルが 1 ⇒ 0 へ変化
5000 タイムスケール毎に繰り返す

- 方法2

```
initial begin
        clock = 0;
end
always #5000
        clock <= ~clock;
```

5000 タイムスケール毎にレベルが反転

コーヒー・ブレイク：`timescale

テストベンチの冒頭に `timescale の記述を見かけたことはありませんか？これはシミュレーション時刻の単位を指定するための記述です。

`timescale <1タイムスケールあたりの実時間> / <丸めの精度>

単位（fs, ps, ns, us, ms, s）を添えて記述します。`timescale の記述は、通常テストベンチにのみに記述します。

```
`timescale 1 ns / 100 ps

module top_tb();
  .
  .
```

#1（1 タイムスケール）は 1ns を表す

100ps 未満は切り捨てか切り上げになる（シミュレータ依存）

それから、式の中におけるシミュレーション時刻を記述する位置については、下記のどちらでも大丈夫です。

```
always begin
        clock = 0;
  #5000 clock = 1;
  #5000;
end
```

```
always begin
  clock =       0;
  clock = #5000 1;
          #5000;
end
```

コーヒー・ブレイク：$finish

テストベンチ中の式に $finish という記述を見かけます。これは、この行まで実行されるとシミュレーションが終了するというものです。この $finish がないと、シミュレーションが終了しません。時間指定してシミュレーションすれば $finish がなくても終わりますが、時間指定なしでシミュレーションを実行した場合は、この $finish を見付けてそこまで到達したら終了させます。どこか1か所で良いので、この $finish を入れましょう！

```
initial begin
        reset = 0;
   #100 reset = 1;
#100000 reset = 0;
   #100 reset = 1;
#500000 $finish;
end
```

500000 タイムスケール後にシミュレーションが終了

はじめてみよう！テストベンチ

VHDL 編

VHDL の書式

テストベンチを作成する時もデザインを作成する時と同じように**パッケージの呼び出し**と**エンティティ部**（entity ～）、**アーキテクチャ部**（architecture ～）を記述します。しかし、一般的にテストベンチには入出力ポートが存在しないため、エンティティ部にはポート宣言を記述する必要はありません。従って、各種宣言と回路記述部には検証対象となるデザインへ入力する信号の入力条件を HDL で記述していきます。

```
（パッケージの呼び出し）

entity エンティティ名 is
  ポート宣言など
end エンティティ名;

architecture アーキテクチャ名 of エンティティ名 is
  各種宣言
begin
  回路記述部
end アーキテクチャ名;
```

デザインの記述例

```
（パッケージの呼び出し）

entity top_tb is
end top_tb;

architecture sim of top_tb is
  各種宣言
begin
  回路記述部
end sim;
```

テストベンチでは
入出力ポートがない！

テストベンチの記述例

各種宣言(コンポーネント宣言 & 信号の宣言)

「各種宣言」をもう少し細かくすると、「コンポーネント宣言」と「信号の宣言」に分けることができます。

「回路記述部」で下位階層のデザインを呼び出す記述を書く時は、上位階層のアーキテクチャ部に下位階層のデザインのコンポーネント宣言をします。コンポーネント宣言のコンポーネント名は、下位階層のエンティティ名と同じにした方がわかりやすいです。下位階層のデザインのすべてのポート名を port の後の ( ) 内に記述してください。ポートの記述には入出力の方向やデータ型を記述しますが、これらは下位階層のデザインのエンティティ部と同じにしないと、正しくシミュレーションできません。

```
component top_tb
  port (clock, reset, test : in std_logic;
        data,              : in std_logic_vector(7 downto 0);
        q                  : out std_logic_vector(7 downto 0));
end component
```

```
(パッケージの呼び出し)

entity top_tb is
end top_tb;

architecture sim of top_tb is
  各種宣言
    コンポーネント宣言
    信号の宣言
begin
  回路記述部
    下位階層（デザインのトップ階層）の
    呼び出し
    下位階層（デザインのトップ階層）への
    入力条件の記述
end sim;
```

テストベンチの記述例

各種宣言(コンポーネント宣言 & 信号の宣言)

「信号の宣言」は、エンティティ部で使用する信号を宣言します。トップ階層への入力信号やトップ階層からの出力信号、テストベンチ内だけで使用する信号を signal 文を使用して信号名とデータ型を記述します。

```
signal clock, reset,test : std_logic;
signal data              : std_logic_vector(7 downto 0);
signal q                 : std_logic_vector(7 downto 0);
```

回路記述部（下位階層の呼び出し ＆ 入力条件の記述）

「下位階層（デザインのトップ階層）の呼び出し」は、通常の下位階層の呼び出しの記述と同様です。

インスタンス名を “u1” 、デザインの最上位階層のエンティティ名を “top” とした場合、以下のように記述します。

```
インスタンス名 : エンティティ名
port map（下位階層のポート名 => 現階層の信号名,
         下位階層のポート名 => 現階層の信号名,
         下位階層のポート名 => 現階層の信号名）;
```

```
u1 : top
port map (clock => clock,
          reset => reset,
          test => test,
          data => data,
          q => q);
```

（パッケージの呼び出し）

```
entity top_tb is
end top_tb;

architecture sim of top_tb is
```

各種宣言

- コンポーネント宣言
- 信号の宣言

```
begin
```

回路記述部

- 下位階層（デザインのトップ階層）の呼び出し
- 下位階層（デザインのトップ階層）への入力条件の記述

```
end sim;
```

テストベンチの記述例

下位階層(デザインのトップ階層)への入力条件の記述

ここでは、主に、

① 常にレベルが固定している信号の記述

② 一定間隔で値がインクリメントする信号の記述

③ 定期的または非定期的に‘1’(H レベル)と‘0’(L レベル)を繰り返す信号の記述

について説明します。

これらをマスターすれば、テストベンチを自分で書けるようになります。

なお、ここで紹介した以外の記述方法もありますので、後々勉強して習得してください。

テストベンチの記述例

下位階層(デザインのトップ階層)への入力条件の記述

①常にレベルが固定している信号の記述

1行で記述できます。なお、数値を記入する際、シングル・ビットの場合は‘ ’で囲み、複数ビットの場合は“ ”で囲みます。

```
test <= '1';
```

```
data <= "10101010";
```

下位階層(デザインのトップ階層)への入力条件の記述

②一定間隔で値がインクリメントする信号の記述

まず、process begin の後に初期値を記述します。その後、loop 文を使って一定間隔でインクリメンタルする式を記述します。なお、数値を記入する際、シングル・ビットの場合は‘ ’で囲み、複数ビットの場合は“ ”で囲みます。

```
process begin
  data <= "00000000";
    loop
      wait for 100 ns;
        data <= data + '1';
    end loop;
end process;
```

100 ns 後に 1 ずつインクリメント

下位階層(デザインのトップ階層)への入力条件の記述

③定期的または非定期的に '1'(H レベル)と '0'(L レベル)を繰り返す信号の記述

process begin の後に初期値を記述し、その後 wait for の後に次の値の変化までの時間を記述します。なお、数値を記入する際、シングル・ビットの場合は ' ' で囲み、複数ビットの場合は " " で囲みます。この記述では、一番下まで実行したら process 文を抜け、最初に戻って実行し、繰り返します。

```
process begin
    data <= "00000000";
  wait for 100 ns;
    data <= "01010101";
  wait for 200 ns;
    data <= "10101010";
  wait for 400 ns;
    data <= "11111111";
  wait for 800 ns;
end process;
```

100 ns 後に 00000000 ⇒ 01010101 へ変化

200 ns 後に 01010101 ⇒ 10101010 へ変化

400 ns 後に 10101010 ⇒ 11111111 へ変化

800 ns 後に 11111111 ⇒ 00000000 へ変化

クロックの例では、この記述で 5 ns 毎に '0' (L レベル) と '1' (H レベル) を永遠に繰り返します。つまり、1周期 が 10 ns、100 MHz のクロックとなります。

```
process begin
    clock <= '0';
  wait for 5 ns;
    clock <= '1';
  wait for 5 ns;
end process;
```

5 ns 後にレベルが 0 ⇒ 1 へ変化

5 ns 後にレベルが 0 ⇒ 1 へ変化

コーヒー・ブレイク：センシティビティ・リスト

センシティビティ・リストとは、VHDL の process 文の直後の（　）内に記述する信号で、この信号が変化した時に、begin ～ end process 内の処理が実行されます。

シミュレーションの場合は、記入しないケースがあります。センシティビティ・リストを記述しない時は（　）も不要です。シミュレーション開始と共に実行を始め、一番下まで行くと再び一番上から実行を始め、永久に繰り返します。

コーヒー・ブレイク：シミュレーションの停止

VHDL には、シミュレーションを停止するための文がありません。従って、先ほど紹介した process 文を使用した記述だけでは、シミュレータでシミュレーションを停止させるか、もしくはシミュレーション時間を指定して実行しないと永遠に終わりません。

そこで、テストベンチにシミュレーションを終わらせるための記述を書きたい時は、wait を使用します。

左の例では、2,700 (100 + 200 + 400 + 2000) ns 経ったら process 文を抜け、最初に戻って実行し、繰り返します。

右の例のように、end process の手前に wait を記述すれば、2,700 ns 後にシミュレーションが停止します。

```
process begin
    data <= "00000000";
  wait for 100 ns;
    data <= "01010101";
  wait for 200 ns;
    data <= "10101010";
  wait for 400 ns;
    data <= "11111111";
  wait for 2000 ns;
end process;
```

```
process begin
    data <= "00000000";
  wait for 100 ns;
    data <= "01010101";
  wait for 200 ns;
    data <= "10101010";
  wait for 400 ns;
    data <= "11111111";
  wait for 2000 ns;
  wait;
end process;
```

ここで
シミュレーションが
止まる

サンプル・デザイン

下記のページよりサンプルデータをダウンロードいただけます。

はじめてみよう！Verilog-HDL＜演習問題付き＞

http://www.altima.jp/members/p1-literature/1-software/1-altera/2016002_verilog_trial.cfm

はじめてみよう！VHDL＜演習問題付き＞

http://www.altima.jp/members/p1-literature/1-software/1-altera/2016001_vhdl_trial.cfm

※ こちらの資料は株式会社アルティマ / 株式会社エルセナで開催していた VHDL 入門編トライアル・コースのテキストと演習マニュアルおよび演習データです。これらのコースの定期開催は 2016年3月にて終了しております。