「FPGA ボードで学ぶ　Verilog HDL」
補助テキスト Vol.1

2008/04/16
5E　小田原弘樹

はじめに

私は昨年の工学研究の時間に、みなさんと同じく「FPGA ボードで学ぶ　VerilogHDL（以下教科書）」を用いて **VerilogHDL** を学びました。しかしこの教科書は少々分かりにくい表現があったり、説明が抜けている箇所が多々あります。そこでこのような教科書の穴を少しでも埋めるべく本テキストを作成するに至りました。

まずみなさんは VerilogHDL とは何ぞやと思われるでしょう。VerilogHDL は**ハードウェア記述言語**（回路設計のためのコンピュータ言語）の一つです。文法が C 言語に近いので他の言語よりは取っ掛かりやすいと思います。因みに、米国防総省が開発に携わった **VHDL** とは別モノです。

私がこの工学研究の授業に参加していた時は担当の教員が何人かいましたが、私が質問をしても誰も答えやヒントをくれません。何故ならどの教員もこの VerilogHDL に関してほとんど無知だったからです。知らないことは教えることなど出来ません。「教科書見れー」の一点張りです。教科書見て分からないから質問してるのに。

このように教科書や教員が不親切である以上、ちゃんとした知識が身につく筈がありません。本テキストでは、みなさんが思う素朴な疑問やエラーにぶち当たった時にどうすればいいのか等を、出来る限り分かりやすく説明したいと思います。ですが私もまだまだ未熟な学生ですので、カバーしきれていないところやみなさんにとって分かりにくい表現をしてしまっているかも知れません。ところどころで「このテキスト使えねーｗｗｗ」とか思うかも知れませんが、どうか大目に見てやって下さい。

本テキストの各章は教科書とリンクしています。本テキストを最初から頼らずに、教科書を読み進めていって疑問に思ったり、詰まったりしたときの補助として使うといった感じでも構いません。また、教科書を見れば分かるようなことは本テキストではあまり解説しません。

今回は教科書第 3 章までをカバーしていますが、今回の授業で第 3 章まで進む必要は全くありません。自分のペースで学習して下さい。

第 1 章　プログラマブル・デバイスで論理設計を学ぼう

SF 小説です。ハードウェア面の説明がほとんどですので、読み飛ばしてもらって構いません。

第２章　FPGA 開発を体験する

本章は非常に重要です。何故なら VerilogHDL を用いた回路プログラミングの全工程が書かれているからです。教科書に書かれている通りに進めていってください。

恐らく次の章に進んでも本章を何度も読み返すことになるでしょう。何度も読み返しているうちに自然と覚えていくと思います。

本章で使われるプログラムですが、これはただ打ち込むだけで、内容を理解する必要は全くありません。もしエラーが多発して、原因が分からずどうしても直らない場合は次の章に進んでも構いません。

第３章　VerilogHDL による論理回路設計の基本テクニック

3.1　スイッチで LED を点灯/消灯する

さて、いよいよ回路の設計に入ります。教科書の最初のリスト 3-1 のプログラムですが、非常に短いプログラムですのでそのまま打ち込めばエラーはあまり出ないでしょう。強いてエラーの要因を挙げるとすれば「**“；”や“,”が抜けている**」「**1（エル）と 1（イチ）を間違えている**」といった感じです。

assign 文のところで「**なんでスイッチに入力された信号を“！”で反転しなきゃいけねーの？**」と思った方がいると思います。これは教科書の図 3-2 を見れば分かります。“LED1”のダイオード記号の向きに注目して下さい。“FPGA ボードの LED 出力回路”側には、常に電圧が掛かっています。なので、“LED1_O”が“L”即ち 0 が入ると電位差が生じてビビビっと電流が流れ、“LED1”が点灯するという仕組みです。“LED1_O”に“H”が掛かっていると電位差が生じず、“LED1”は点灯しません。よって、この assign 文のところではスイッチに入力された信号を反転して“LED1_O”に伝えているのです。

因みに“assign”とは「割り当て」という意味です。

教科書には「SW1_I を 30 番ピン、LED1_O を 17 番ピンに割り当てます。」と書いてあるだけで、ピン配置の記述が載っておらず少し混乱している方がいるかもしれません。これは以下のように記述します。

```
NET　“SW1_I”　　LOC = “P30”　;
NET　“LED1_O”　　LOC = “P17”　;
```

第 2 章ではところどころ“**#**”の記述がありました。ピン配置の記述での“#”は**コメント文**を記すために用いるもので、教科書では**区切り**に使用されています。特に必要というわけではありません。

また、エラーが出たときの対処法を本テキストの最後に載せておきました。参考になるかは分かりませんが、ちらっと読んでおいて下さい。

【要点】 ①“;”や“,”による閉じ忘れに気をつける。
②assign 文では左辺に信号を出力する場所を、右辺に出力したい信号の値を記述する。LED を光らせる場合は、この信号の**値を反転**する必要がある。

3.2 スイッチで LED を点滅する

ここでは新しく“**レジスタ宣言**”と“**always 文**”が出てきます。教科書のリスト 3-2 ですが、always 文のところが少し変です。begin と end が二つずつありますが、一つでいいです（何故このような記述の仕方をしているのか分かりません）。

レジスタ宣言（reg ～）ですが、これは C 言語で言う int のようなもので、値を入れておく入れ物を宣言します。しかしリスト 3-2 の記述では 1 ビット（0 か 1）の情報しか入りません。もっと多くの情報を入れるための方法はもう少し後で出てきます。

また、reg 文で数字を頭にしたレジスタ宣言をすると怒られます。例えば“reg 1sec ;”といったものです。数字を使いたい場合は頭以外につけましょう。

always 文の **posedge** ですが、これは positive edge の略で、教科書にある通り、**立ち上がりエッジ**を示します。

“**begin**”と“**end**”は“(”と“)”のようなものです。**閉じ忘れに気をつけましょう。**

また、「led_status <= !led_status ;」の“<=”は**矢印**のようなものだと思って下さい。

【要点】 レジスタは、値を入れておく入れ物。数字を頭にして宣言することはできない。
“begin”と“end”による閉じ忘れに気をつける。
ピン配置は 3.1 のものと同じ。

3.3 四つの LED を順番に点灯する

ここでは新たに**バスの記述**が出てきます。“[4:1]”のように記述しますが、“4 対 1”ではありません。イメージ的には“4～1”といった感じです。

P41 では“led_status[4:1] <= { led_status[3:1] , led_status[4] } ;”という風に記述できるとありますが、これは“led_status[4:1] <= { led_status[3] , led_status[2] , led_status[1] , led_status[4] } ;”と同義です。

パラメータ宣言は、C 言語での **define 文**のようなものです。parameter と reg の違いは教科書に載っています。

さて、動作したでしょうか？この回路を動作させたとき、スイッチを一回しか押していないのに一度に 2，3 個シフトしてしまうという現象が起きた方がいると思います。これはプログラムのバグでも FPGA ボードの故障でもありません。これは**チャタリング**という現象で、教科書 P84 にも載っていますが簡単に説明すると、スイッチ部分に使われているバネの微小振動によってスイッチが数回押されてしまうというものです。これはプログラムによって防止することが出来ます。チャタリング除去回路の記

述法は次回のテキストに載せる予定です。
　因みにチャタリング（chattering）とは、「ガタガタいう」という意味です。

【要点】 スイッチを押すと、チャタリングという現象が起こることがある。どの FPGA ボードでも起こりえる現象であり、除去用の回路プログラムを組み込むことよって防止可能。

3.4　スイッチを押した回数を数えて四つの LED に表示する

　特に解説することはありませんが、教科書の表 3-5 は後でちょくちょく使うかもしれません。

3.5　スイッチを押した回数を数えて 7 セグメント LED に表示する

　ここでは新たに **case 文**が出てきます。C 言語では **switch 文**のようなものです。非常に便利ですので使い方を覚えましょう。

　教科書のリスト 3-5 の case 文がある always 文で、“always @*”とありますが、この“*”は「全ての状態」を表します。

　7 セグのビット配置は間違いやすいのでちゃんと修正しましょう。また、教科書のように表示ができたらビット配置を色々いじってみるのもいいかもしれません。
　また、“8’b10111111”のような記述の“b”のすぐ右の 1 ビットは、7 セグの右下にある**ドット**の ON OFF を表しています。数などを表示させる上では特に必要ないので、ここは“0”にして構いません。

【要点】　“always @*”の“*”は、「全ての状態」を表す。

3.6　1 秒ごとに LED 表示を変化する

　ここでは**クロック**の使い方を学びます。VerilogHDL を学習する上でクロックは非常に重要ですので、しっかり覚えましょう。

　さて、リスト 3-7 の parameter 文を見てください。「FF33M000_cnt = 32’h01F78A40」とあります。ちょっと分かりにくいですよね。16 進での記述は確かにソフトウェアやハードウェア的には効率の良い記述なのかもしれませんが、初心者にとっては**まどろっこしいだけ**です。分かりやすくするために、ここでは私たちが長年お付き合いしている 10 進数様を使いましょう。10 進数の記述は“h”を“d”に変えて、あとは 10 進の数字を打ち込むだけです。

```
parameter              cnt = 32’ d33000000   ;
```

　といった感じに記述すればとっても分かりやすいはずです。因みにビット数は変わらず 32 のままにしましょう。教科書 p.44 の表 3-5 を見ると、32 ビットは 4,294,967,296 まで表すことができます。3300 万くらいなら余裕で入れられますね。
【要点】　parameter 文を使って量的な値を宣言するときは、10 進数で宣言すると分かりやすい。

3.7　1秒ごとに7セグメントLEDの表示を変える

ここでは新たに“**下位モジュール呼び出し**”というテクニックが出てきます。これはC言語での“**サブルーチン呼び出し**”のようなものです。C言語のサブルーチンは一つのプログラム内に記述できますが、VerilogHDLの下位モジュールのプログラムは別々に作成します。

下位モジュール作成の手順が教科書に載っていないので以下に示します。

1. リスト3-8“top”を作成する。
2. 左上の“Sources”ウィンドウの“top (top.v)”を右クリック。
3. “New source”をクリック。
4. “File name”に下位モジュール名を入力（“pulse1sec”，“seg7enc”等）。
5. “Verilog Module”を選択して“Next”をクリック。
6. “Next”をクリック。
7. “Finish”をクリック。

あとはリスト3-9やリスト3-10を同じように打ち込んでいくだけです。

付録　エラーの対処法

エラーが出てしまったときは、下にある“Console”ウィンドウをスクロールしていくとエラーの内容を見ることができます。そこを見ることによってエラー改善のヒントを得ることができるでしょう。因みに、一番上のエラーを修正するとそれ以降のエラーも改善されることが多いです。

ここでは、よく見かけるエラーと改善のヒントをいくつか紹介したいと思います。

・“line XXX unexpected token: 'YYY'”
“unexpected”とは「予期せぬ」という意味なので「ラインXXXにYYYがあるのは予想外デス」という意味ですが、恐らくポート定義での“,”の閉じ忘れだと思われます。

・“line XXX expecting 'YYY', found 'ZZZ'”
“expecting”とは「予想する」という意味です。「ラインXXXにYYYがあると思うんだけどZZZがあるよ」という意味です。恐らく“;”や“begin”、“end”による閉じ忘れだと思われます。

・“line XXX 'YYY' has not been declared”
“declared”とは「宣言される」という意味で、「ラインXXXのYYYは宣言されてねーよ」という意味です。YYYのスペルが宣言されたものと違っていないかチェックしてください。

このように、エラー報告では“line XXX～”という風にどこがエラーの元凶なのかを表示してくれます。上記以外のエラーが出たとき、まずはその行を見て下さい。