第4章　シンプルな組み合わせ回路のHDLソースを紐解く

HDLでディジタル回路を書く！作る！

黒毛利 学
Manabu Kuromori

HDLを書くときの心構え

第3章では，開発ツール ISE WebPACKを使ってとりあえずCPLDを動作させてみました．本章では，記述したVerilog HDLがCPLDの中で実際にどのように動作するのかを見てみます．

「もう動いたんだから，わざわざCPLDの内部の回路を考える必要はないのでは？」と言う声が聞こえてきそうです．しかし将来，特集で扱った回路より，もっと規模が大きく実用的なディジタル回路を設計することになったとき，論理回路をイメージしながらHDLソースを書いてきた人とそうでない人では，できあがる回路の性能(動作速度や回路規模)に大きな差がつくでしょう．論理回路をイメージすることはとても重要なのです．

● HDLは回路を意識しながら書く

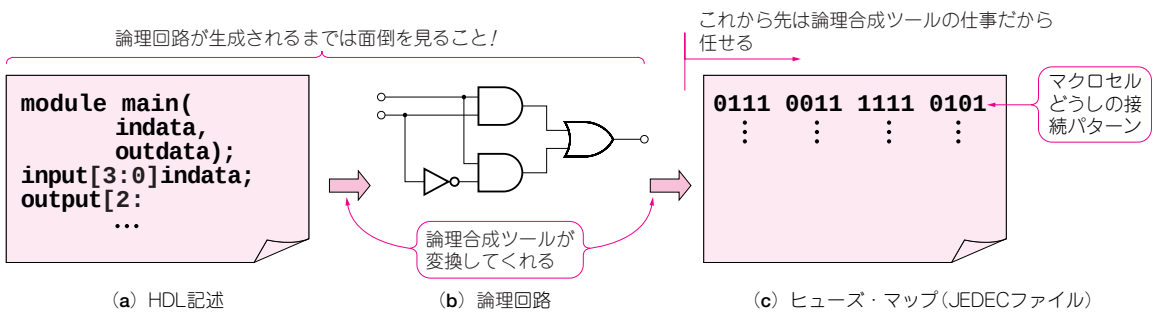
図1に示すように，論理合成ツールは，人間が理解できる言語であるVerilog-HDLやVHDLなどのHDLソースをPLDが理解する言葉に変換してくれます．

PLD語(ヒューズ・マップ)をいきなり手で書いてもいいんですが，100万ゲートもあるPLDだったらきっと破綻するでしょう．HDLで回路設計をする作業レベル(RTLと呼ぶ)でなら，論理合成ツールを利用することによって，たくさんのゲート数を組み合わせる複雑な回路でも，論理レベルを意識せずに容易に設計できます．

論理合成ツールは，HDLソースをPLDが理解できる言葉(ヒューズ・マップ)に変換してくれるだけでなく，PLD内で占有する回路面積が小さくなるように無駄をとってくれたり，動作速度を高速化してくれます．しかし，論理合成ツールは万能ではありません．気づかないところで無駄な回路を作り上げていることがあります．「無駄があっても動けばいいじゃないか」と思うかもしれませんが，動作周波数が上がり信号間のデータの受け渡しのタイミングがシビアになってくると，なかなか希望どおり動かなくなります．こうなると，論理合成ツールにいろいろ指示を出してもどうにもなりません．

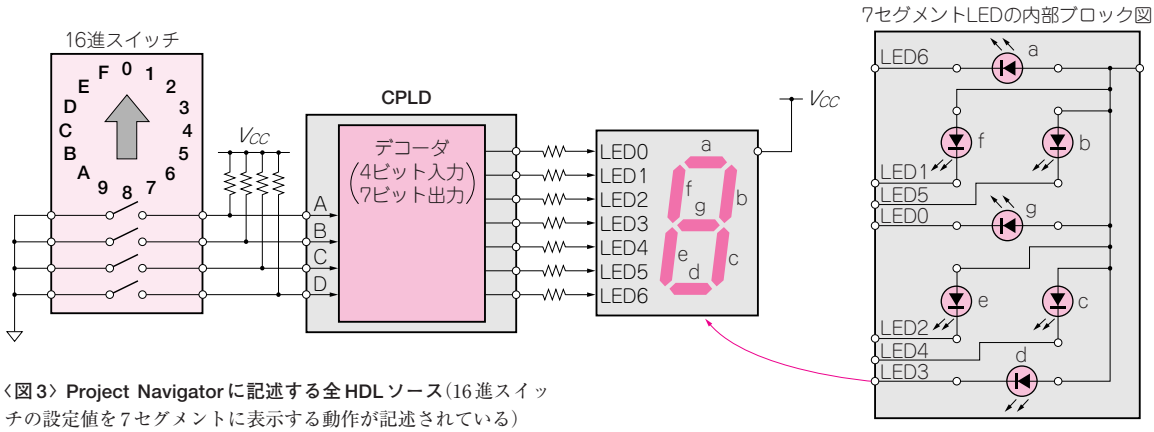
こんなトラブルに遭遇しないためには，HDLソースを書くときに，CPLDの最小機能単位であるマクロ

〈図1〉論理合成ツールに任せきりにすると後で痛〜い目に遭うかも



Keywords
case文，デコーダ，レジスタ宣言，UCFファイル，論理合成，真理値表，負論理，論理積，AND，論理和，OR，論理否定，NOT，論理ゲート，論理レベル加減算回路，assign文，キャリー・アップ，半加算器，全加算器，ヒューズ・マップ，JEDECファイル．

〈図2〉16進スイッチの設定値を7セグメントに表示する回路



〈図3〉Project Navigatorに記述する全HDLソース(16進スイッチの設定値を7セグメントに表示する動作が記述されている)

```
main.v
 1  module main(
 2      HexSW_ip,
 3      led_op);
 4
 5      input [3:0] HexSW_ip;
 6      output [6:0] led_op;
 7
 8      reg  [6:0] led_op;
 9
10      always@(HexSW_ip)begin
11         case (HexSW_ip)
12             4'b0000 : led_op <= 7'b0000001;//0
13             4'b0001 : led_op <= 7'b1001111;//1
14             4'b0010 : led_op <= 7'b0010010;//2
15             4'b0011 : led_op <= 7'b0000110;//3
16             4'b0100 : led_op <= 7'b1001100;//4
17             4'b0101 : led_op <= 7'b0100100;//5
18             4'b0110 : led_op <= 7'b0100000;//6
19             4'b0111 : led_op <= 7'b0001111;//7
20             4'b1000 : led_op <= 7'b0000000;//8
21             4'b1001 : led_op <= 7'b0001100;//9
22             4'b1010 : led_op <= 7'b0001000;//A
23             4'b1011 : led_op <= 7'b1100000;//b
24             4'b1100 : led_op <= 7'b1110010;//c
25             4'b1101 : led_op <= 7'b1000010;//d
26             4'b1110 : led_op <= 7'b0110000;//E
27             4'b1111 : led_op <= 7'b0111000;//F
28             default : led_op <= 7'b1111111;//
29         endcase
30     end
31
32 endmodule
33
```

HexSW_ipが0000のときled_opに0000001を代入せよ

入出力ピン(port)を定義

portの入出力方向を定義

レジスタを定義

HexSW_ipが上記以外のときは表示なし

HexSW_ipの状態を表す(表2参照).
一番右のビットがHexSW_ip[0].
4'bは4ビット信号という意味

〈表1〉16進スイッチによる7セグメントLED点灯回路のCPLDの端子とHDLモジュールの入出力端子の関連付け

信号名称	CPLDのピン番号
HexSW_ip[0]	P33
HexSW_ip[1]	P32
HexSW_ip[2]	P30
HexSW_ip[3]	P29
led_op[0]	P97
led_op[1]	P94
led_op[2]	P92
led_op[3]	P91
led_op[4]	P90
led_op[5]	P89
led_op[6]	P87

セルをイメージする必要があります．というのは，PLD内の一つのマクロセルを信号が通過するときに，入出力間で約2～3 nsの遅延が発生するからです．高速に動作させたかったら，信号が通過するマクロセルの数が少なくなるように，HDLソースを書く必要があります．HDLを書くとき，論理合成ツールがどのようにマクロセルを並べて回路を構成するかイメージできれば，どのくらいの遅延が起きるかを予測できます．

16進スイッチで7セグメントLEDを点灯させる回路を動かす

■ Verilog－HDLソースを書く

● case文を使う

開発ツールに慣れるために，もう少し高機能なHDLソースをCPLDに組み込んで動作させてみましょう．

HDLトレーナ上の16進スイッチで設定した値を7セグメントLEDに表示させてみます．回路を**図2**に，Project NavigatorにHDLソースを**図3**に示します．7セグメントLEDとはディジタル時計などに使われている表示素子で，数字を7個のLEDで表現したものです．

デコーダとは，n個の入力に対して，その値が特定