情報工学実験 II

柴田裕一郎

shibata@cis.nagasaki-u.ac.jp
情報工学コース

2018 年 11 月 29 日

重要な通知

3 年生 4Q の「情報工学実験 IV」は本実験を修得しないと履修できないため，万一，本実験が不合格になると 4 年生に進級できなくなってしまう可能性が生じます.

実験の進め方と評価方法

- 実験レポート（100%）
  内容の節目ごとに提出（未提出は不合格）
- 試験は行わない
- 講義資料
  `http://slab.cis.nagasaki-u.ac.jp/~shibata/lab2/wiki/`

FPGA（Field Programmable Gate Array）

- 書き換え可能なゲートアレイ
  - 回路情報（コンフィギュレーションデータ）を電気的に書き込むことで好きな回路を実現できる
  - 専用 LSI などに比べると動作速度は遅い
  - 何度でも書き換えられるので主にプロトタイピングに利用されていた
  - その後、テレビなど身近な商用製品にも使われるようになった
  - データセンターにおける大規模処理の高速化にも利用されている

ハードウェア記述言語（HDL）

- ハードウェア設計用の専用言語
  HDL 記述から実際の回路を生成することを論理合成という
- 利点
  - 記述の抽象度が高く他人が見ても分かりやすい
  - 論理の簡単化など詳細なレベルの設計は自動化可能
  - 設計時間が短縮できバグも減る
- 欠点
  - 合成可能な回路の形式が制限される（特に非同期回路など）
  - 最大性能は人手による設計に及ばないこともある（ほとんどない）

主な HDL

- Verilog HDL
  CADENCE 社のハードウェアシミュレーション言語用として出発．文法はゆるい．
- VHDL
  米国国防総省によりハードウェアの仕様記述言語として出発．
  文法が厳格．
- SFL
  NTT により開発．はじめから論理合成指向だが対象は単相同期回路に限定．
- SystemVerilog
  Verilog HDL の上位互換拡張．便利な Verilog HDL．

設計ツールの操作の流れ

1. デザインファイル（SystemVerilog）の記述
2. シミュレーションモジュール（SystemVerilog）の記述
3. シミュレーションの実行（make sim）
4. 波形の確認（vsim コマンド）
5. 論理合成・配置配線（make）
6. FPGA に回路を構成し動作確認（make config）

簡単な SystemVerilog ファイルの例（myand.sv）

```
1  `default_nettype none
2  /*
3   * Simple AND gate
4   */
5  module myand
6    (
7     input wire  in1,
8     input wire  in2,
9     output wire out1
10    );
11
12    assign out1 = in1 & in2;
13 endmodule
14 `default_nettype wire
```

よく使われる演算子

- 算術演算：
  +（加算），-（減算），*（乗算），/（除算），**（べき乗算）
- 比較演算：
  <（小なり），<=（以下），>（大なり），>=（以上）
- 等号演算：
  ==（等しい），!=（等しくない）
- 論理演算：
  &&（論理積），||（論理和），！（否定），
- ビット演算：
  ~（NOT），&（AND），|（OR），^（XOR），~^（XNOR）
- 条件演算：
  ?:（条件式？真の場合：偽の場合）
- シフト演算：
  >>（右論理シフト），<<（左論理シフト）

シミュレーションモジュール（sim_myand.sv）

```
1  `default_nettype none
2  `timescale 1ns/1ps
3  // Simulation module for myand
4  module sim_myand();
5      logic in1, in2;
6      wire  out1;
7
8      myand myand_inst(.in1(in1), .in2(in2), .out1(out1));
9
10     initial begin
11         in1 <= 1'b0;
12         in2 <= 1'b0;
13         #10
14           print();
15         in2 <= 1'b1;
16         #10
17           print();
```

シミュレーションモジュール（sim_myand.sv）

```
18        in1 <= 1'b1;
19        in2 <= 1'b0;
20        #10
21          print();
22        in2 <= 1'b1;
23        #10
24          print();
25        $finish;
26    end
27
28    task print();
29        $write("in1=%b in2=%b out1=%b\n", in1, in2, out1);
30    endtask
31 endmodule
32 `default_nettype wire
```

シミュレーションの実行

- 配布した Makefile を利用 [1]

```
% make sim
```

- 結果の確認（sim_myand.log にも保存されている）

```
...
# in1=0 in2=0 out1=0
# in1=0 in2=1 out1=0
# in1=1 in2=0 out1=0
# in1=1 in2=1 out1=1
```

4 通りの in1 と in2 の値の組合わせに対して，出力 out1 が正しく AND として機能しているかを確認.

[1] 先頭の「%」はシェルプロンプトを表すので打たないこと.

波形の確認

- 波形ビューアの起動

```
% vsim sim_myand.wlf &
```

- RTL シミュレーション
  - レジスタトランスファレベル（Register Transfer Level）
  - 入力が変化してから出力が変化するまでのゲート遅延や配線遅延は考慮されていない

設計に関する用語

- 論理合成
  言語による記述から実際の回路を作ること．この際，回路圧縮などの最適化も自動的に行われる．
- 配置配線
  回路を FPGA の内部にどう配置しどう配線するのかを決めること．入出力ピンの割当てなどをユーザが指定すればあとは自動．
- コンフィギュレーションデータ
  FPGA に読み込ませる回路情報のこと．本実験ではファイルの拡張子は「`.rbf`」．
- コンフィギュレーション（する）
  FPGA にコンフィギュレーションデータを読み込ませて回路を構成すること．

make コマンドによる論理合成・配置配線

- Altera の Quartus II を使った論理合成・配置配線
- 以下の 4 つのファイルを準備（中身は分からなくても良い！）
  - make コマンドを制御するための Makefile
  - 入出力ピンの配置などを指示する tcl ファイル（myand_top.tcl）
  - 回路の遅延制約を指示する sdc ファイル（myand_top.sdc）
  - 実験ボードのインタフェース回路記述（myand_top.v）
- make コマンドを実行

```
% make
```

- コンフィギュレーションデータ（myand_top.rbf）が生成されれば成功

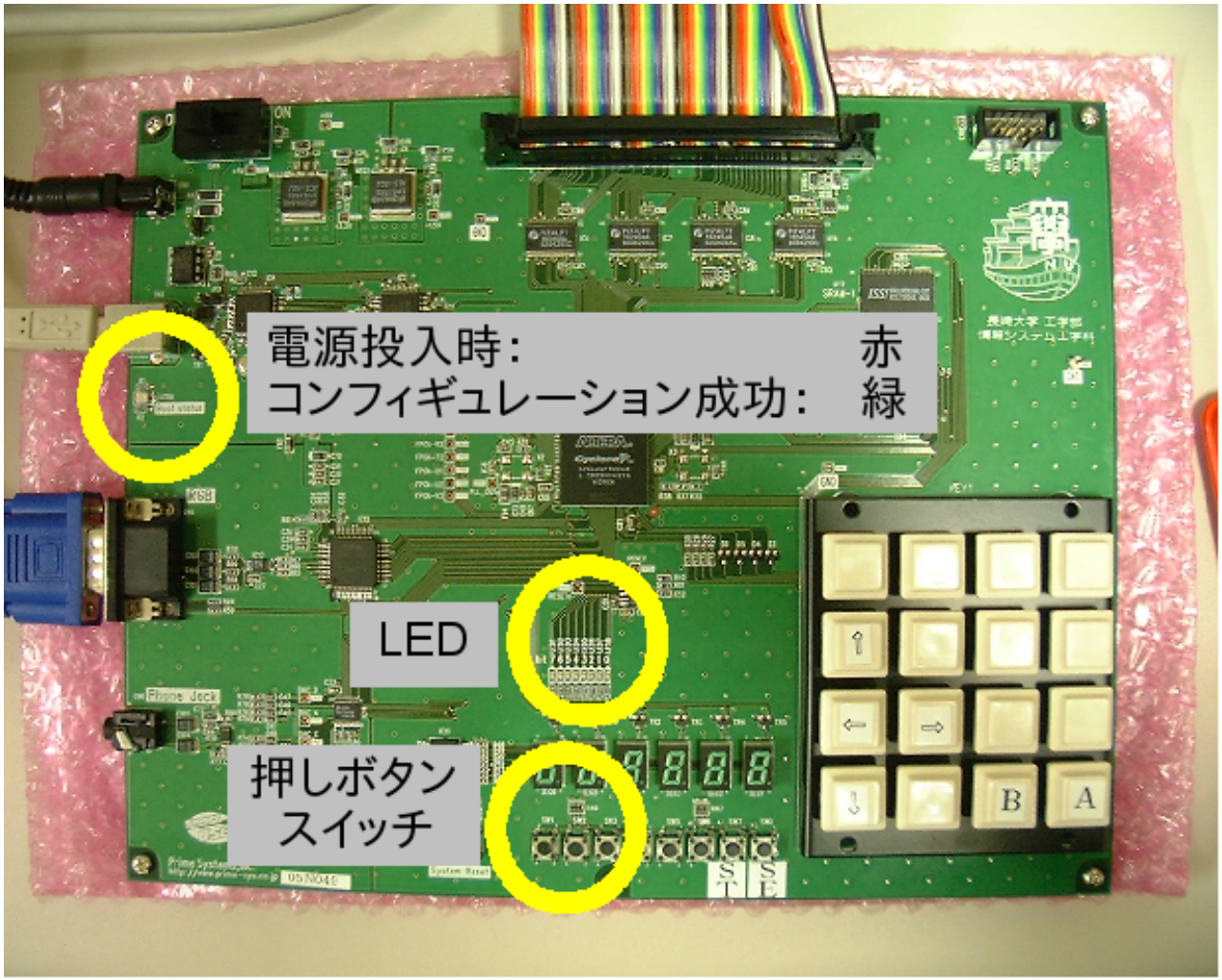
実験ボードでの動作確認

- 実験ボードの接続
  - USB ケーブルで自分の PC と接続
  - 電源投入
- FPGA のコンフィギュレーション

```
% make config
```

- 動作確認
  - 入力 `in1` は SW1
  - 入力 `in2` は SW2
  - 出力 `out1` は LED8 に接続されている

ボードの部品配置



複数の出力がある場合の例（two.sv）

```
1  `default_nettype none
2  /*
3   * Sample with two output ports
4   */
5  module two
6    (
7     input wire  in1,
8     input wire  in2,
9     output wire out1,
10    output wire out2
11    );
12
13    assign out1 = in1 & ~in2;
14    assign out2 = in1 |  in2;
15 endmodule
16 `default_nettype wire
```