再構成型アーキテクチャ特論 （5）

osana@eee.u-ryukyu.ac.jp

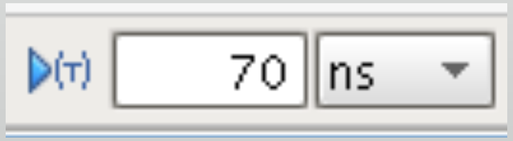
前回の復習: テストベンチ

- テストベンチ
  - `timescale, initial, always #
  - $ ではじまるシステムタスク
  - テスト対象モジュールにテストベンチで生成した信号を入力する
  - インスタンス名.信号名 で深い階層の信号も参照できる

前回の復習: Vivado

- FPGA の型番を指定してプロジェクトを作る
  - ソースファイルは別フォルダにしておいたほうがよい
  - RTL は “Design Source” で、
  - テストベンチは “Simulation Source”

前回の復習: Vivado Simulator

- Vivado Simulator の基本的な使い方
  - Simulation settings で runtime を 0 に
  - 設定した時間だけ進める 

  - 表示する信号を変更したら  ソースコードを変更したら 
- ソースコードが Web にあります:
  - http://mux.eee.u-ryukyu.ac.jp/lecture.html.ja

今日の予定

- ゴールはボードを動かすこと
  - LED くるくる → 7セグメント LED も動かしたい
  - RTL を書いて、シミュレーションして、論理合成と配置配線
- シミュレーションには難しいこともある: parameter の活用
- 課題が出ます

Implementation Flow

- 論理合成
- テクノロジマッピング
- 配置配線
- ビットストリーム生成
  - 詳しくは演習で

後半は実習

- 論理合成・配置配線して、ビットストリームを生成後 FPGA に書くまで
  - ソースファイルは Web で配布
  - 途中のスライドに出てくるソースコードはそれに含まれています

当面の目標

- ストップウォッチを作りましょう
  - プッシュスイッチと7セグメントLEDの制御回路
  - 10進カウンタ
- 今日使うソースコードをかなりの部分流用できるはず

スイッチ

- チャタリング除去
  - この間のでは押しっぱなしにすると定期的に信号が出ます
    - テスト用の回路は後ほど (配布に含まれているけど、完全ではない)
  - 解決方法は各自考えましょう

7セグメントLED

- アノードコモン
  - 点灯したい桁のアノードを L、
  - 点灯したい部分のカソードを L で点灯
  - アノードを切り替えながら残像で全桁を表示 (ダイナミック駆動)

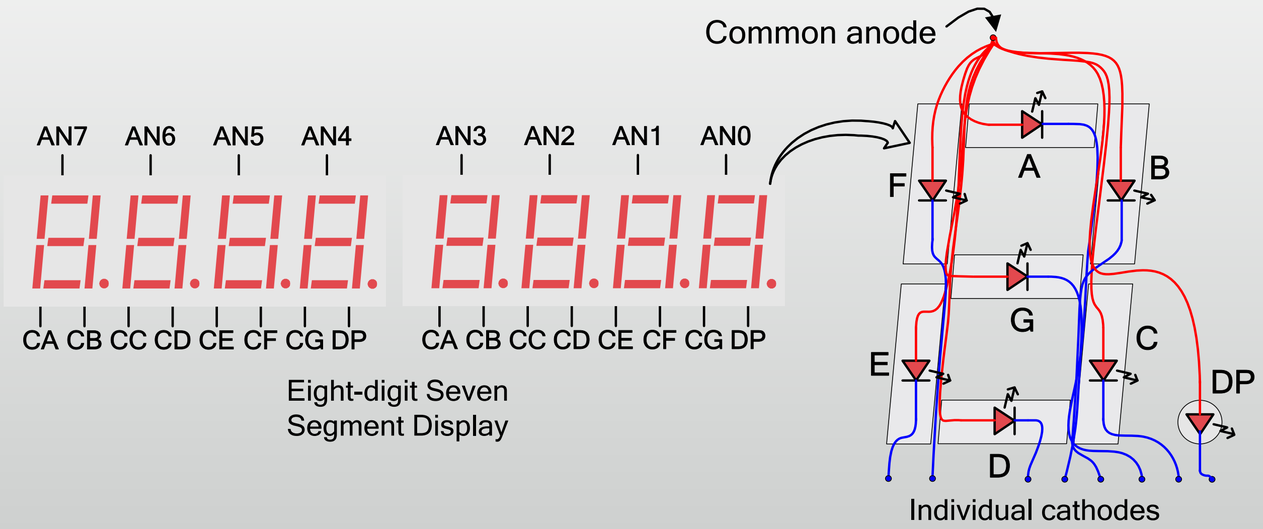


Figure 18. Common anode circuit node

人間と電子回路のタイムスケール

- 回路は 100MHz で動くけど人間はムリ
  - LED が 100MHz で動くと見えないので、カウンタを入れて遅くする
  - 6Hz なら 2^{24}=16M まで数えればよい
  - シミュレータの波形で 16M まで追いかけるのは…? (やっぱり無理)
    - シミュレーションと論理合成でカウンタのビット数を変更したい

LEDくるくる 再び

- 赤字のところを変更したい
  - たとえば2ビットなら
    波形見て追いかけられる

```
module led_kurukuru
    ( input  wire CLK, RST,
      output reg [15:0] LED );

  reg [23:0] CNT;
  wire STROBE = &CNT;

  always @ (posedge CLK) begin
     if (RST) begin
        CNT <= 0;
        LED <= 16'b1000_0000_0000_0000;
     end else begin
        CNT <= CNT+1;
        if (STROBE)
          LED <= {LED[0], LED[15:1]}
    end
  end
endmodule
```

Parameter を使った宣言

- やり方はふたつ
  - モジュールの先頭で宣言
  - ポート宣言の前に #() で宣言
- どちらも規定値が必要
- 後者はポート幅変更も可能

```
module led_kurukuru
    ( input wire CLK, RST,
      output reg [15:0] LED );

  parameter CounterBits = 24;
  reg [(CounterBits-1):0] CNT;
  (以下同文)
```

```
module led_kurukuru #
    ( parameter CounterBits = 24 )
    ( input CLK, RST,
      output reg [15:0] LED );

  reg [(CounterBits-1):0] CNT;
  (以下同文)
```

外部からパラメータを変更

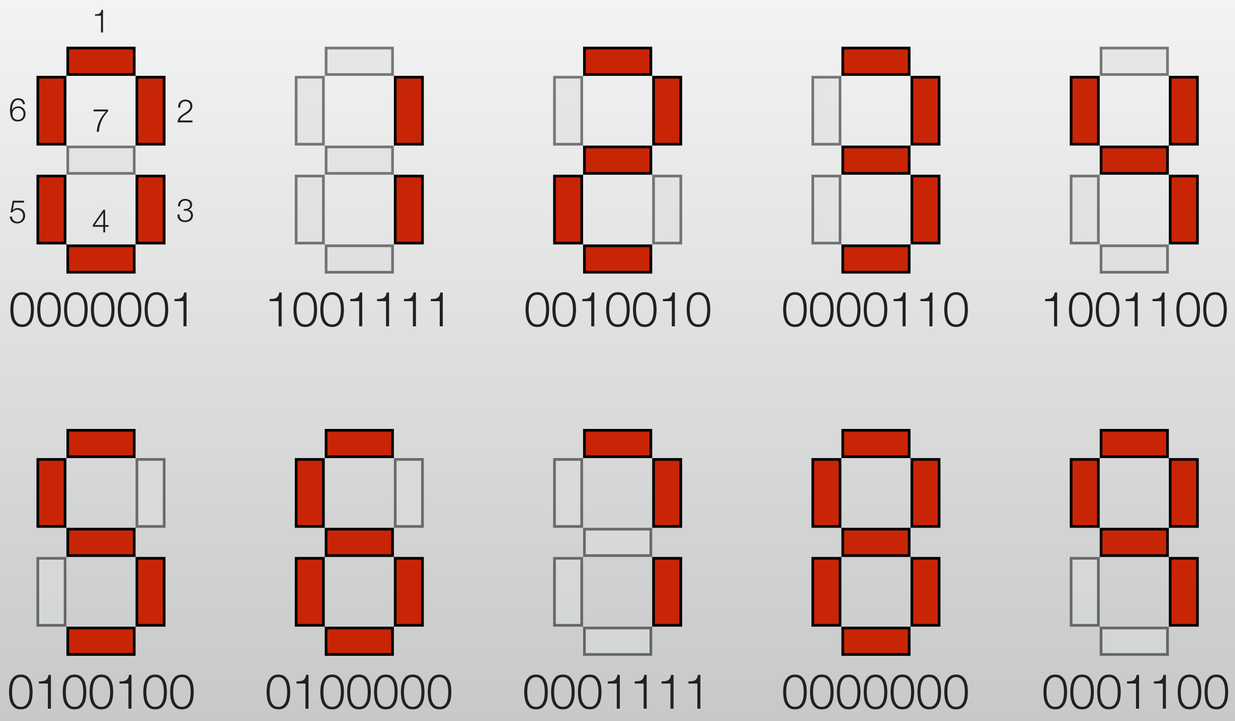
* 前の例は何もしないと 24
  * インスタンス宣言時に変更可
  * kuru2 では2ビットカウンタ
* テストベンチでは数字を変え、
  論理合成は規定値で、とか便利

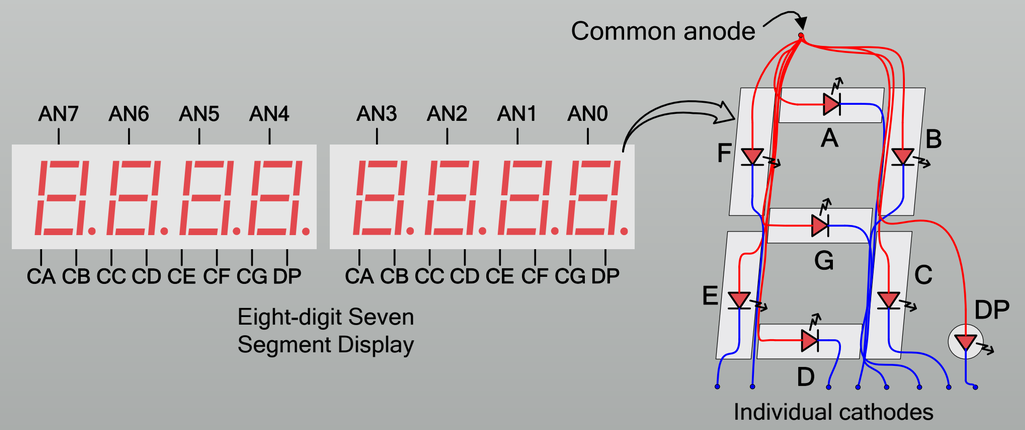
```
led_kurukuru
  kuru1(.CLK(CLK), .RST(RST), .LED());

led_kurukuru  # ( .CounterBits(2) )
  kuru2(.CLK(CLK), .RST(RST), .LED());
```

演習

7セグメントLED

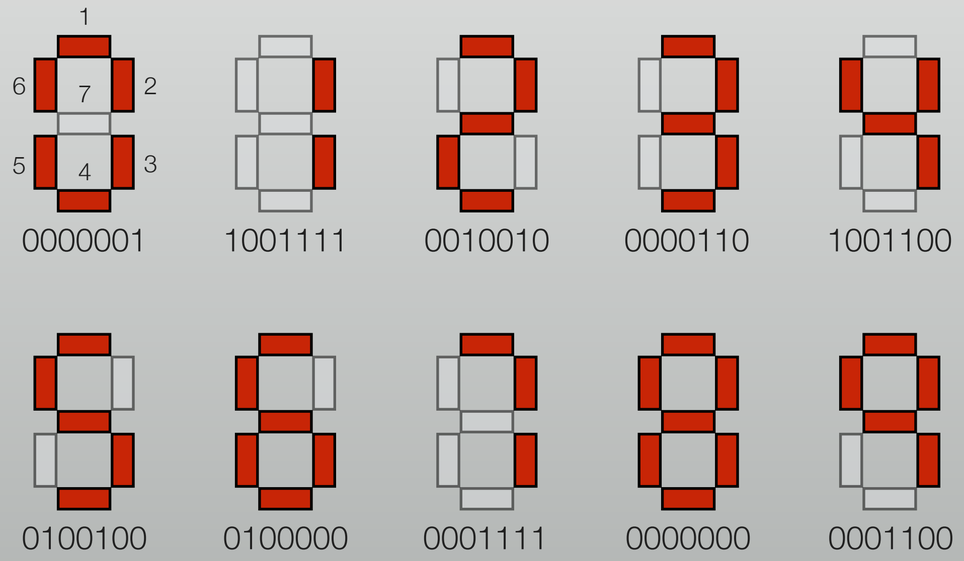
- 0~9の数字と点灯パターン
  - 小数点を別として7ビット
  - これをカソードに与える
  - 0で点灯
- アノード(桁)の駆動はまたあとで





１桁だけ動かすことを考える

* ４ビット入力→7ビット出力

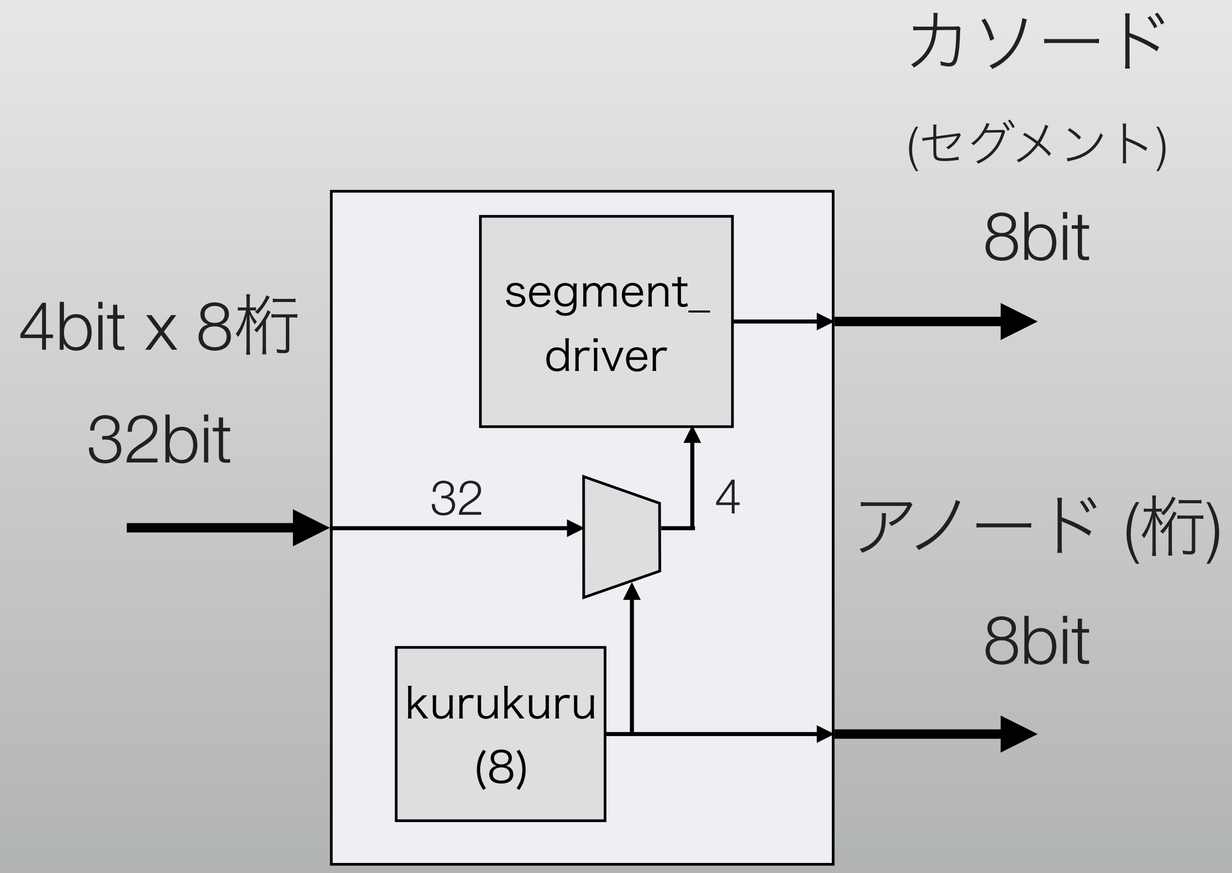



```
module segment_driver
  ( input  wire [3:0] VAL,
    output wire [6:0] CATHODE );

  assign CATHODE =
      (VAL==4'h0) ? 7'b000_0001 :
      (VAL==4'h1) ? 7'b100_1111 :
      (VAL==4'h2) ? 7'b001_0010 :
      (VAL==4'h3) ? 7'b000_0110 :
      (VAL==4'h4) ? 7'b100_1100 :
      (VAL==4'h5) ? 7'b010_0100 :
      (VAL==4'h6) ? 7'b010_0000 :
      (VAL==4'h7) ? 7'b000_1111 :
      (VAL==4'h8) ? 7'b000_0000 :
      (VAL==4'h9) ? 7'b000_1100 : 7'h0;
endmodule
```

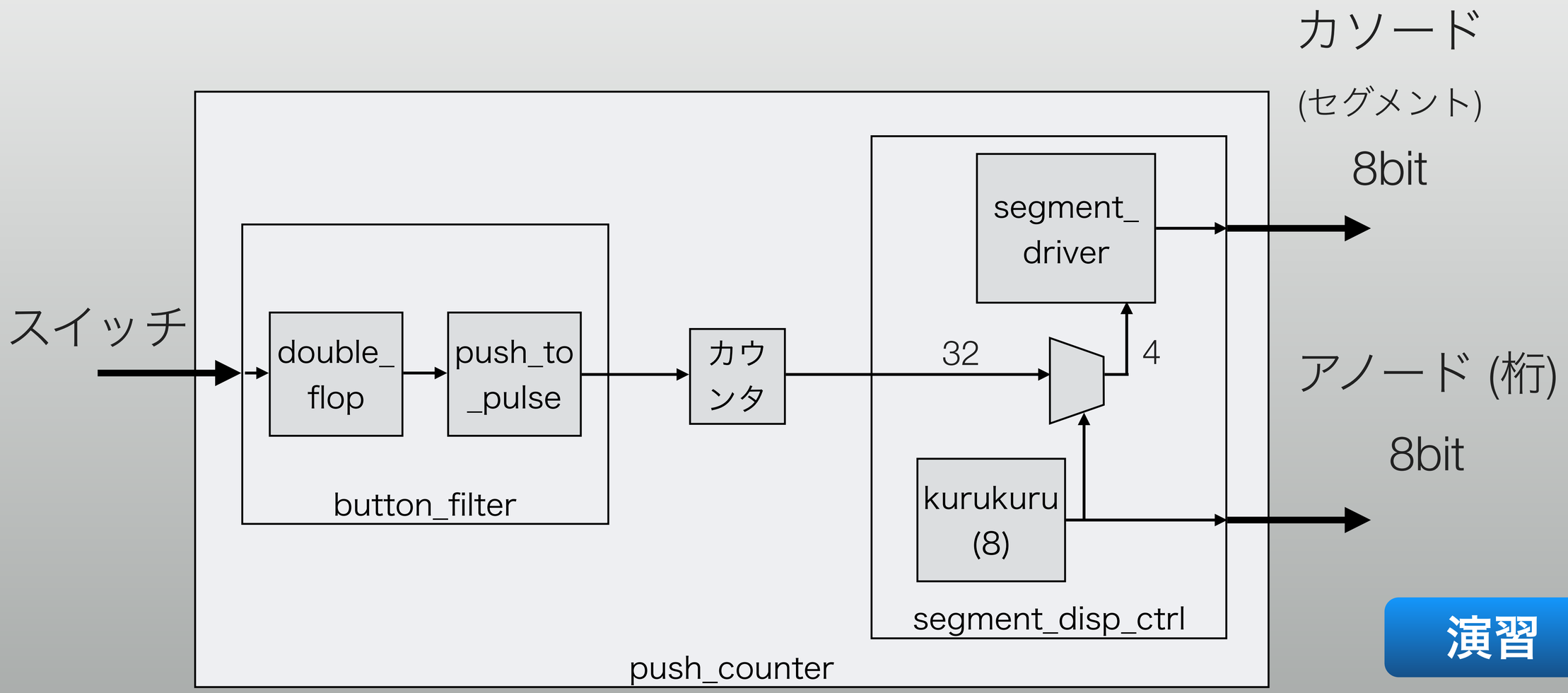
論理圧縮はツールがやってくれる！

最後はこうしたい→構成を考える

- 8桁分の値を32bitで入力
  - 各桁4bit
  - segment_driver の修正で16進表示も可能



ちゃんと使えるようにする



演習

課題

- ストップウォッチ作りましょう
  - 1/10000 秒まで測れる
  - 最大 9999.9999 秒
- 入力は RST と START-STOP のふたつ。後者はチャタリング除去する
- 100MHz のクロックは無限の精度だと思って信用してよい

演習1: LEDくるくる

演習の目的

- シミュレーションと実機で違う挙動にする
  - シミュレーション時はカウンタのビット数の parameter を変更
  - ソースファイルはすべて準備してあります
- Implementation flow での Vivado の使い方を理解する

ソースファイル

- ソースは src1/ に入っています
  - Constraint: led_kurukuru.xdc
  - Design Source: led_kurukuru.v
  - Simulation Source: led_kurukuru_test.v

プロジェクトを作る

- USB メモリから C:/Users/user01/ 以下にフォルダごとコピー
- lab1/vivado としてプロジェクトを作成
  - Device: XC7A100T-1CSG324
  - ソースファイルは先ほどの3つ

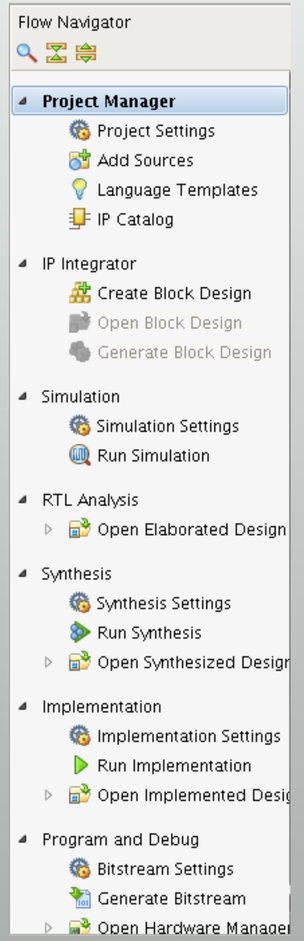
シミュレーションしてみる

Flow Navigator

- Project Manager: プロジェクト管理 (まとめ)
- Simulation: シミュレーション
- RTL Analysis: HDL の解析
- Synthesis: 論理合成
- Implementation: テクノロジマップと配置配線
- Program and Debug: ビットストリーム生成とFPGAへの書き込み

ここが実質的な
Implementation Flow
(上から下へ順番にやる)

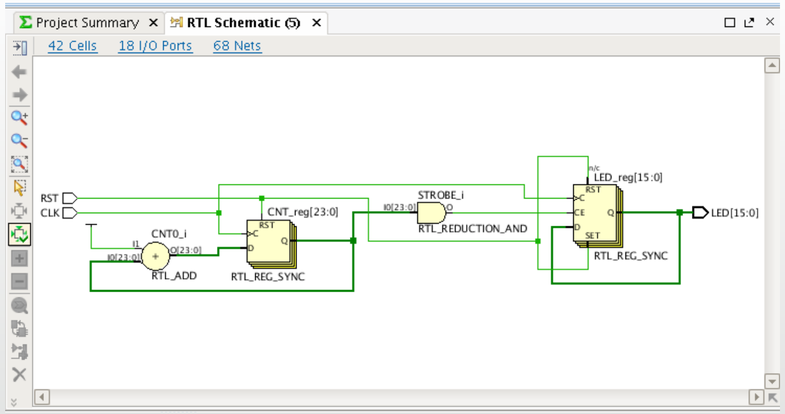
(IP Integrator は今日は使いません)



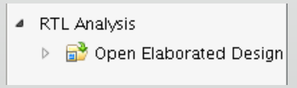
Project Manager

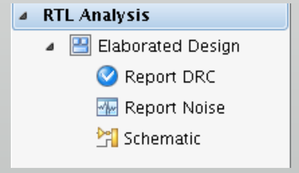
- 最初の画面
  - ターゲットデバイスとかそういう情報はここに
- 右下に表示されているもの:
  - Messages / Log: エラーが出たときなどに確認する
  - Design Runs: Implementation flow の進行状況を表示

RTL Analysis

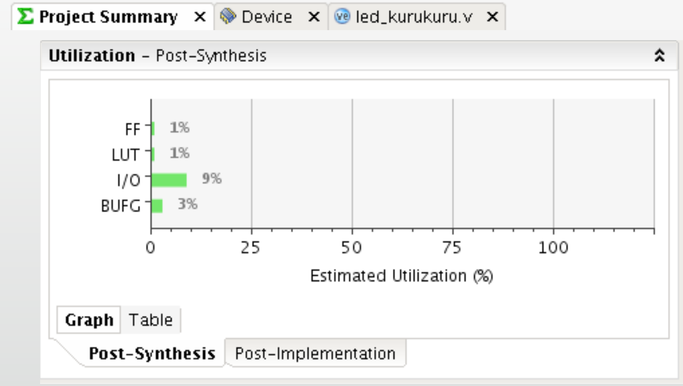


- “Open Elaborated Design” で開始
  - 合成から先のフローを起動しても、必要なら自動で走る
  - 完了すると Elaborated Design に関するメニューが出る
- Schematic は RTL と対応しており、右クリックで該当箇所へ
- 特に重要な意味はないですが、自分のRTL設計を概観したいときに

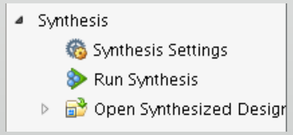


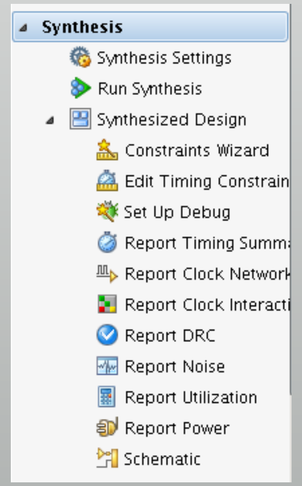


Synthesis



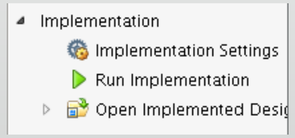
- Run Synthesis で起動
  - Open Synthesized Design すると各種レポートや Schematic
    - Device view はまだ入出力だけ (中身は配置されていない)
  - Project Summary にリソース使用量の予測が出ます
  - Implementation までいかないと確定しないけど、目安に

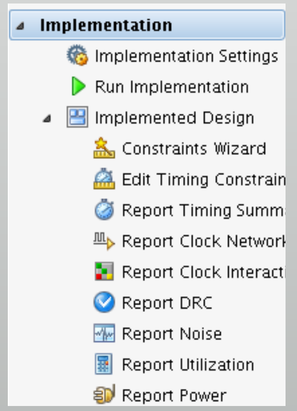




Implementation

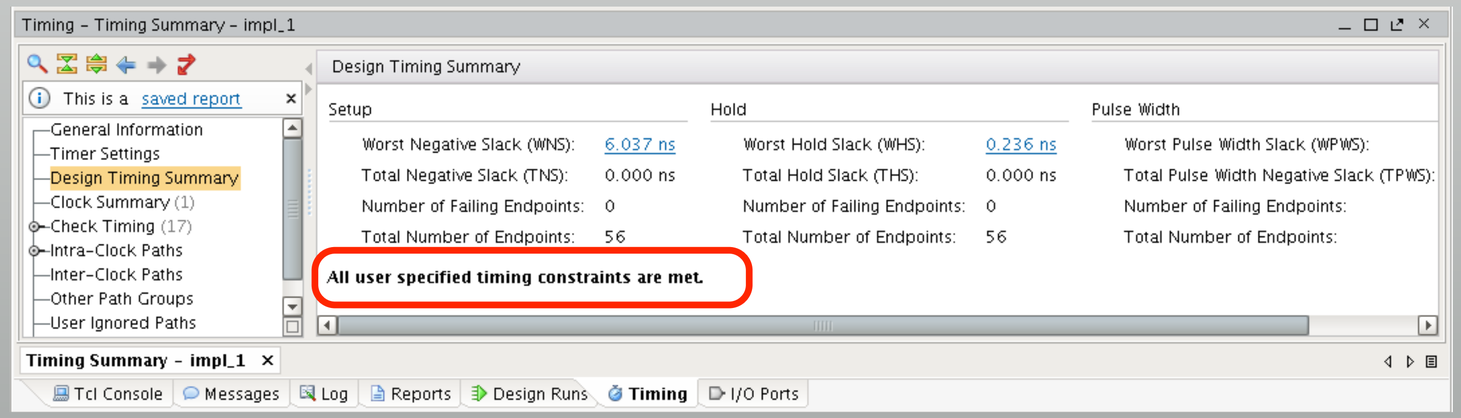
- Run Implementation で起動
  - I/O だけでなく回路がちゃんと配置配線された状態になる
  - Project Summary のリソース使用量が確定版に
  - タイミングと I/O の結果が非常に重要！





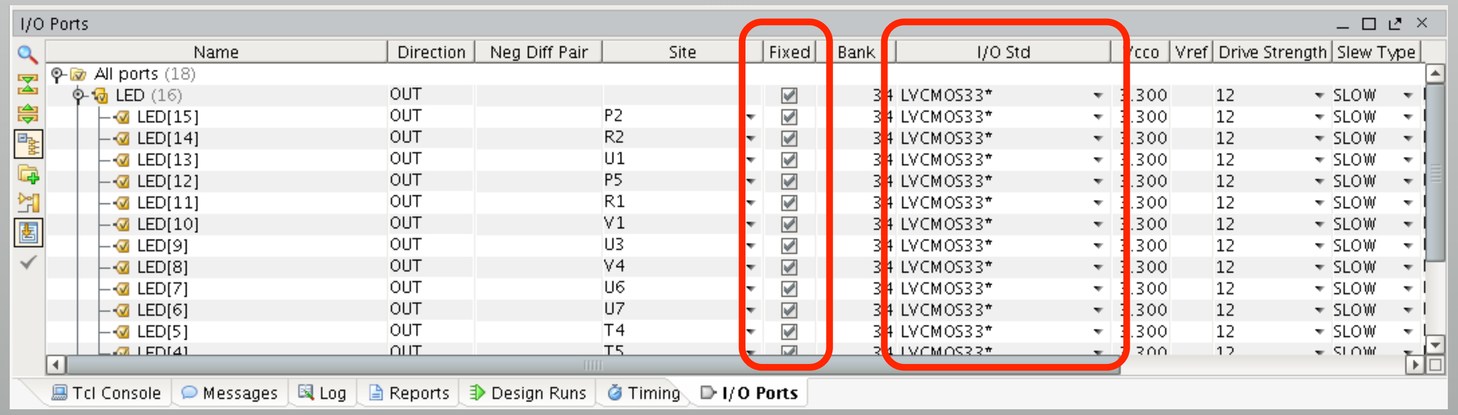
Implementation: Timing

- Implementation が完了すると出る
  - Implemented Design を開いて Window → Timing
  - “All user specified timing constraints are met” がないとダメ

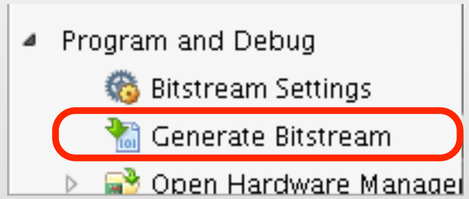


Implementation: I/O Ports

- Implemented Design を開いて Window → I/O Ports
  - 全部のポートが “Fixed” で、I/O Standard が正しいことを確認
  - 間違えると最悪の場合ボードが壊れます



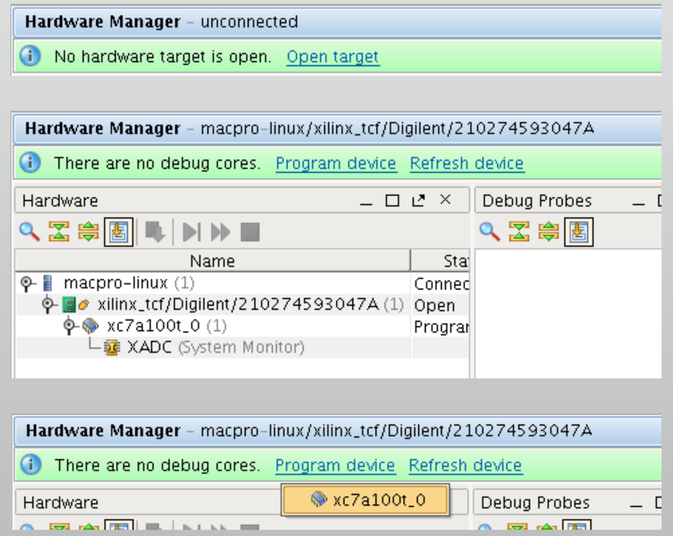
ビットストリーム生成



- 配置配線の結果をFPGAに書き込める形にする
  - ハードウェアマネージャでFPGAに書き込む前にやっておく
  - Synthesis → Implementation → Generate Bitstream と順番に起動しなくても、最初から Generate Bitstream すれば必要なステップは自動的に起動
    - これは Implementation とかでも同じなので、時間次第で使い分け

Hardware Manager

- ボードは事前に接続、電源 on しておく
- Open Target で FPGA へ接続
- ボード上のデバイスが認識されたことを確認
- Program device で書き込み
  - ファイル名とかは特に指定しなくてよい



チェックすること

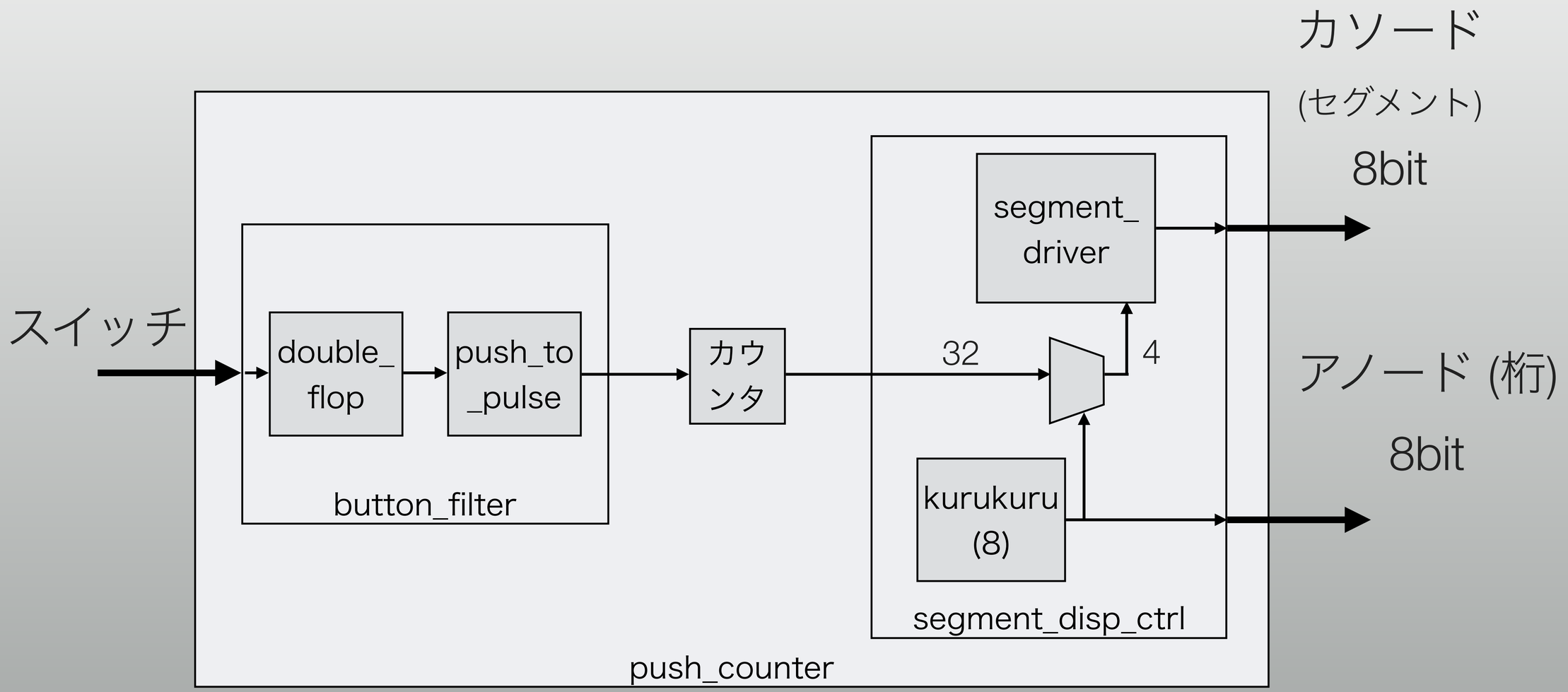
- 実機ではLED が順番に点灯すれば OK
- シミュレーションとは点灯の切り替え速度が違うことも確認しましょう
  - テストベンチから Parameter を変更してみるとなおよい

演習2: 16進LEDカウンタ

プロジェクトを作る

- そのまま File → New Project
- ソースは lab2/src
  - push_counter_test.v: テストベンチ, 7seg.xdc: 制約
  - その他はデザインソース

構成


カソード
(セグメント)
8bit
スイッチ
double_
flop
push_to
_pulse
button_filter
カウ
ンタ
32
4
segment_
driver
kurukuru
(8)
segment_disp_ctrl
アノード (桁)
8bit
push_counter

ボタンの割り当て

- 左 (BTNL) : ADV
  - カウンタをふやす
- 中央 (BTNC): RST
  - リセット

Parameter はふたつ

- LED の桁の切り替え (ColumnCounterBits)
  - 10bit カウンタ = 1/100 ms (ちょっと速すぎたかも…)
- チャタリング除去のカウンタ (ButtonFilterCount)
  - 20×10^6 = 1/5sec

試してみる

- 左ボタンを押しっぱなしにする
  - 5Hz でカウンタが増えてしまう
- push_counter.v で ColumnCounterBits = 24
  - 実習1 と同じ間隔で LED が切り替えになる

課題ふたたび

- ストップウォッチ
  - 今回の演習のモジュールを再利用して構いません
- 締切は 11/27
  - ソースファイル「だけ」、つまり *.v と *.xdc をメールで提出
  - どういう構成にしたかをA4で1, 2枚程度で記述してPDFで提出