2023年度版

Ver. 2023-05-17a

Course number: CSC.T341

コンピュータ論理設計 演習(6)
Computer Logic Design Exercise(6)

情報工学系　荒堀喜貴

Yoshitaka ARAHORI, Department of Computer Science

arahori_at_c.titech.ac.jp

Computer Logic Design support page https://www.arch.cs.titech.ac.jp/lecture/CLD/

アナウンス

- ACRiルームに，高速なコンピュータを導入しました．
- 次のマシンが高速です．空いていれば、ここから選んで使いましょう．
  - vs701 ～ vs710
  - vs801 ～ vs810
  - vs901 ～ vs910
  - vs101 ～ vs110（設定中）

コンピュータ論理設計 演習(Exercise)の注意点

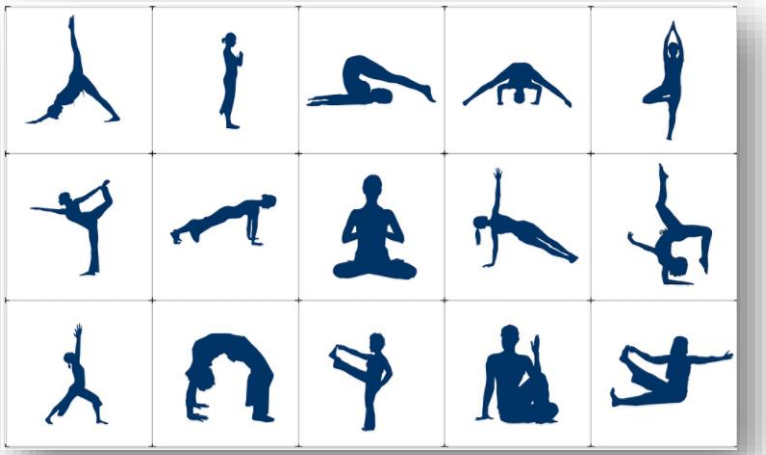
- 演習はACRiルームを利用します。
- 3～4人のグループを作成します．そのグループ内で情報を共有しながら演習を進めてください．
- 問題はグループ内で相談して解決する，あるいは，担当のTA(Teaching Assistant)や教員に質問してください．
- 演習には出席点があります．休まずにきちんと出席しましょう．
- 演習スライドにチェックポイントの図がある場所は，作業を確認してもらう場所です．すべてのチェックポイントをクリアしましょう．

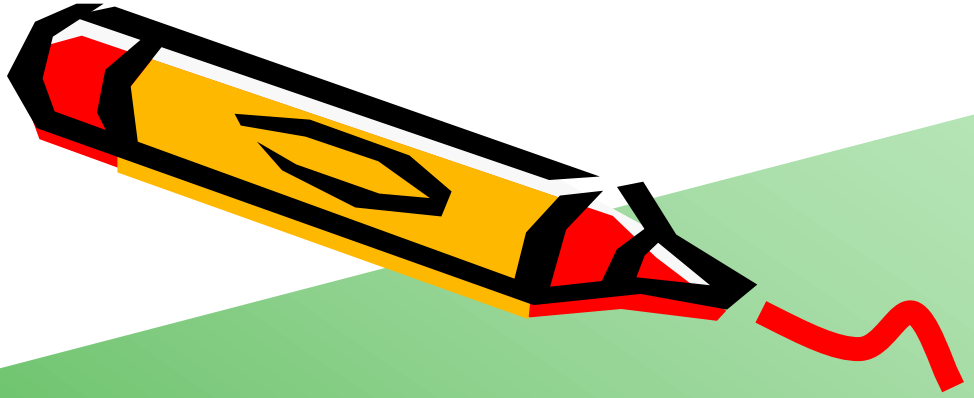
- 演習時間でなくてもACRiルームを利用できます．現在は，1日に4枠（3時間 x 4枠 = 12時間）を利用できます。独自のハードウェア設計などに挑戦しましょう．

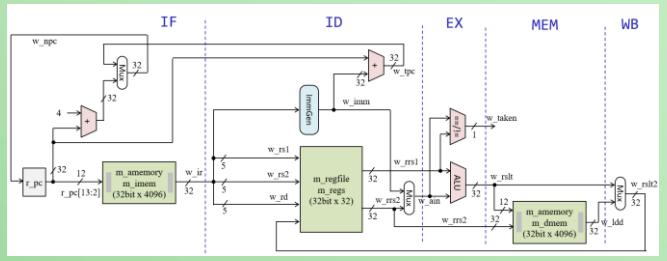
Exercise(6)

- Project_8
  - プロセッサデザインコンテストのルールなどを理解する.
  - プロセッサデザインコンテストの準備を進める.

- チェックポイント (CP) の確認は5月25日(木) 10:30 まで.
  これまでに，できるだけ多くのチェックポイントを通過すること.







Processor Design Contest

Design & Implementation

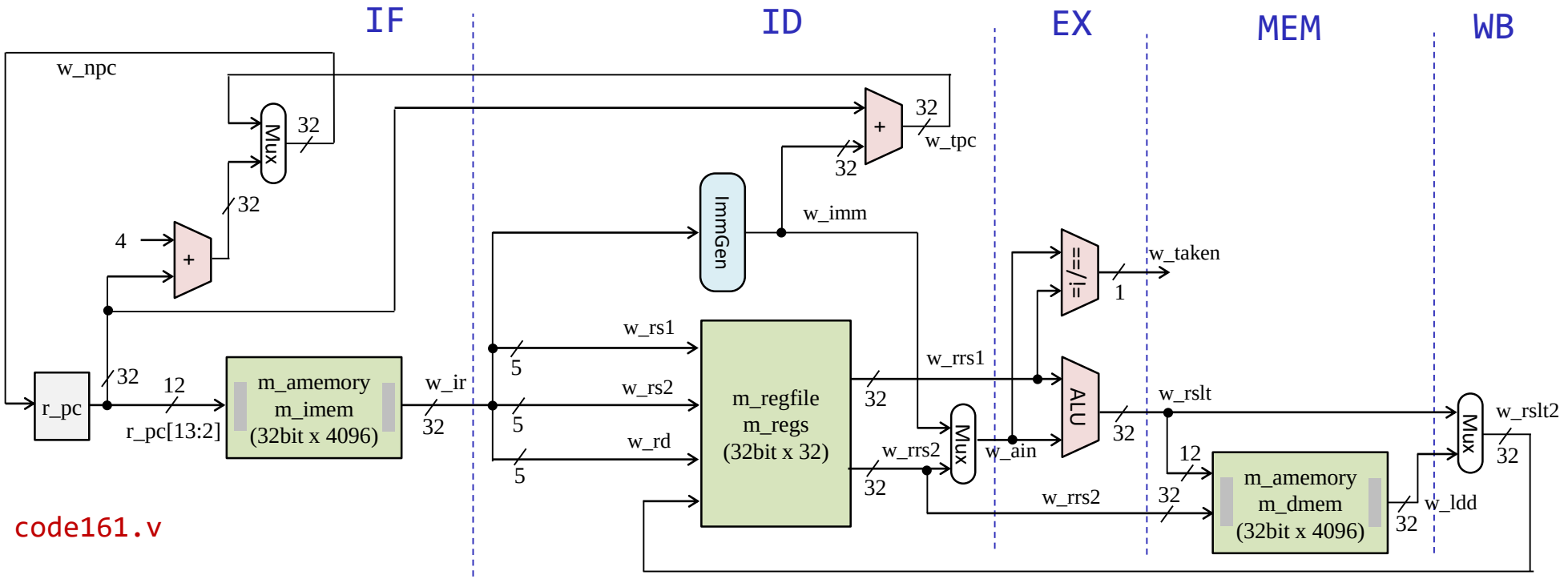
- コンテストに向けて，各グループで
  - 役割の分担をしましょう.
  - スケジュールの確認をしましょう.
  - 設計を進めましょう.

コンテストのルール (1/2)

- グループでプロセッサの高速化に取り組み，コンテスト形式で成果を競う．
- 高速なRISC-V（のいくつかの命令を処理できる）プロセッサ **m_proc** を設計する．
  - サポートする命令: **add**，**addi**，**sll**，**srl**，**lw**，**sw**，**beq**，**bne**
  - 与えられたプログラム (次のスライド) の実行時間が短いプロセッサを設計すること．
  - Vivado 2022.2 を使うこと．
  - Vivado のタイミング制約を満たす設計とすること．
  - 実行される全ての命令におけるレジスタファイルの入力データ w_rslt2 の値が正しいことをシミュレーションで確認すること．
  - FPGA で動作させて，VIO の値が 0x017fd000 となることを確認すること．
  - 命令メモリ，データメモリはそれぞれ4096ワードとすること．
  - VIO, Clocking Wizard で生成される IP を除いて，Verilog HDL で設計すること．
- 各グループはPowerPointのスライド（6枚程度）を用いて**6分**のプレゼンテーションをおこない，設計したプロセッサの性能と魅力を説明する．
- 利用するFPGAボード: Digilent Arty A7-35T
- 性能の高い数グループを表彰する．加えて，プロセッサアーキテクチャの独自性と，設計した方式の完成度の高さにより表彰する．

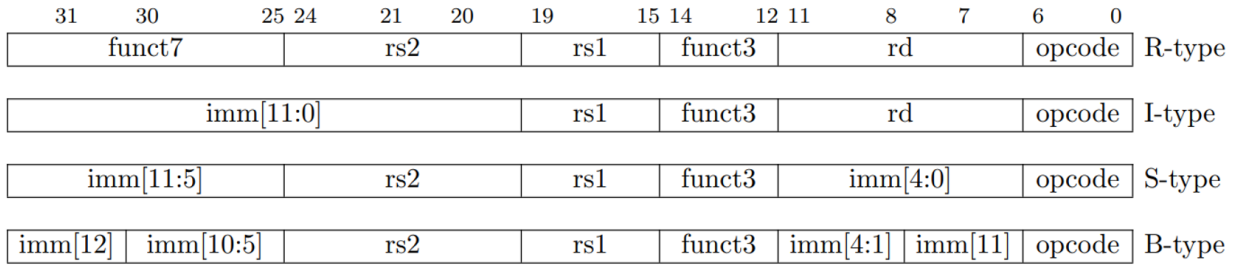
m_proc07 ベースラインのプロセッサ（シングルサイクル）

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, sll, srl, lw, sw, beq, bne命令に対応したプロセッサ



code161.v

f = 60MHz
IPC = 1.000
P = 60.00



RISC-V Program for the contest

- Result : 0x017fd000

```
#include <stdio.h>

main()
{
    int mem[2048];
    int i=0, j=0, sum = 0;

    for(j=0; j<3; j++){
        for(i=0; i<2048; i++) {
            mem[i] = i*4;
        }
        for(i=0; i<2048; i++) {
            sum += mem[i];
        }
    }
    printf("%d %x¥n", sum, sum);
}
```

```
/*************************************************************/
/** program for CLD design contest 2023 (Version 2023-05-14a) **/
/** do not modify this code                                  **/
/*************************************************************/
initial begin
cm_ram[ 0] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[ 1] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[ 2] = 32'h00000a13; //     addi x20, x0,  0
cm_ram[ 3] = 32'h00300a93; //     addi x21, x0,  3
cm_ram[ 4] = 32'h00000633; //     add  x12, x0,  x0
cm_ram[ 5] = 32'h00000593; // L03:addi x11, x0,  0
cm_ram[ 6] = 32'h40000413; //     addi x8,  x0,  1024
cm_ram[ 7] = 32'h40040413; //     addi x8,  x8,  1024
cm_ram[ 8] = 32'h00000493; //     addi x9,  x0,  0
cm_ram[ 9] = 32'h00000513; //     addi x10, x0,  0
cm_ram[10] = 32'h00b52023; // L01:sw   x11, 0(x10)
cm_ram[11] = 32'h00148493; //     addi x9,  x9,  1
cm_ram[12] = 32'h00458593; //     addi x11, x11, 4
cm_ram[13] = 32'h00959cb3; //     sll  x25, x11, x9
cm_ram[14] = 32'h009cdd33; //     srl  x26, x25, x9
cm_ram[15] = 32'h00058593; //     addi x11, x11, 0
cm_ram[16] = 32'h00450513; //     addi x10, x10, 4
cm_ram[17] = 32'h00940463; //     beq  x8,  x9,  L04
cm_ram[18] = 32'hfe0410e3; //     bne  x8,  x0,  L01
cm_ram[19] = 32'h40000413; // L04:addi x8,  x0,  1024
cm_ram[20] = 32'h40040413; //     addi x8,  x8,  1024
cm_ram[21] = 32'h00000493; //     addi x9,  x0,  0
cm_ram[22] = 32'h00000513; //     addi x10, x0,  0
cm_ram[23] = 32'h00052583; // L02:lw   x11, 0(x10)
cm_ram[24] = 32'h00148493; //     addi x9,  x9,  1
cm_ram[25] = 32'h00450513; //     addi x10, x10, 4
cm_ram[26] = 32'h00b60633; //     add  x12, x12, x11
cm_ram[27] = 32'h00160613; //     addi x12, x12, 1
cm_ram[28] = 32'hfff60613; //     addi x12, x12, -1
cm_ram[29] = 32'h00160613; //     addi x12, x12, 1
cm_ram[30] = 32'h00160613; //     addi x12, x12, 1
cm_ram[31] = 32'hfff60613; //     addi x12, x12, -1
cm_ram[32] = 32'h00160613; //     addi x12, x12, 1
cm_ram[33] = 32'hffe60613; //     addi x12, x12, -2
cm_ram[34] = 32'hfc941ae3; //     bne  x8,  x9,  L02
cm_ram[35] = 32'h015d5d33; //     srl  x26, x26, x21
cm_ram[36] = 32'h001a0a13; //     addi x20, x20, 1
cm_ram[37] = 32'h01140413; //     addi x8,  x8,  0x11
cm_ram[38] = 32'h01240413; //     addi x8,  x8,  0x12
cm_ram[39] = 32'h01340413; //     addi x8,  x8,  0x13
cm_ram[40] = 32'h01440413; //     addi x8,  x8,  0x14
cm_ram[41] = 32'hf75a18e3; //     bne  x20, x21, L03
cm_ram[42] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[43] = 32'h00060f33; //     add  x30, x12, x0
cm_ram[44] = 32'h000f0033; //     add  x0,  x30, x0
cm_ram[45] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[46] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[47] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[48] = 32'h00000033; //     add  x0,  x0,  x0
cm_ram[49] = 32'h00000033; //     add  x0,  x0,  x0
end
```

/home/tu_kise/cld/2023/baseline/program.txt

書き込まれる値が正しいことをシミュレーションで確認

- 下のモジュール m_top でシミュレーションする.
  - プロセッサが w_led でゼロでない値を出力したらシミュレーションが終了するように設定.
  - ある程度の時間(50000000nsec)が経過したらシミュレーションが終了するように設定.
- 実装したプロセッサ m_procXX のシミュレーションで, 出力を trace.txt に保存する.
- diff コマンドで, trace.txt と trace_good.txt (正しい出力)とが一致することを確認する. diff コマンドで何も表示されなければ, 出力は一致している.

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_led;

//  initial $dumpfile("main.vcd");
//  initial $dumpvars(0, m_top);

  reg [31:0] r_cnt = 1;
  always@(posedge r_clk) r_cnt <= r_cnt + 1;

  m_proc p (r_clk, 1'b1, w_led);

  always@(posedge r_clk) $write("%7d %08x¥n", r_cnt, p.w_rslt2);

//  initial $write("  clock: r_pc     w_ir     w_rrs1   w_ain    r_rslt2  r_led¥n");
//  always@(posedge r_clk)
//    $write("%7d: %08x %x %08x %08x %08x %08x¥n", r_cnt,
//            p.r_pc, p.w_ir, p.w_rrs1, p.w_ain, p.w_rslt2, w_led);
  always@(posedge r_clk) if(w_led!=0) $finish;
  initial #50000000 $finish;
endmodule
```

code161.v

```
$ cd
$ cd cld
$ cp /home/tu_kise/cld/2023/baseline/* .

$ iverilog code161.v
$ ./a.out > trace.txt
$ diff trace.txt trace_good.txt
```

シミュレーションで，実行のサイクル数を確認する

- 前のスライドを参考にシミュレーションし，出力を trace.txt に保存する．
- コマンド tail trace.txt で，ファイルの最後の10行を出力する．
- 最後の行の右の文字列が，017fd000 になっていることを確認する．
- 最後の行の左の数字が，実行に要したサイクル数になる．
- Baselineプロセッサの実行サイクル数は 129,077 であることがわかる．Baselineはシングルサイクルのプロセッサなので，129,077命令が実行されたことがわかる．
- プロセッサのアーキテクチャを工夫することで，動作周波数の向上を狙いながら，この実行サイクル数があまり増加しないように工夫する．

```
129068 00000400
129069 00000003
129070 00000811
129071 00000823
129072 00000836
129073 0000084a
129074 00030000
129075 00000000
129076 017fd000
129077 017fd000
```

```
$ iverilog code161.v
$ ./a.out > trace.txt
$ tail trace.txt
```

詳しいシミュレーションの結果を参考にする

- 下のモジュール m_top において，上の赤色の行をコメントアウトして，青色の4行を有効にしてシミュレーションすると，詳細なシミュレーション結果を出力する．
- このシミュレーション結果が /home/tu_kise/cld/2023/baseline/trace_detail.txt にあるので参考にすると良い．

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_led;

//  initial $dumpfile("main.vcd");
//  initial $dumpvars(0, m_top);

  reg [31:0] r_cnt = 1;
  always@(posedge r_clk) r_cnt <= r_cnt + 1;

  m_proc p (r_clk, 1'b1, w_led);

//  always@(posedge r_clk) $write("%7d %08x¥n", r_cnt, p.w_rslt2);

  initial $write("  clock: r_pc     w_ir     w_rrs1   w_ain    r_rslt2  r_led¥n");
  always@(posedge r_clk)
    $write("%7d: %08x %x %08x %08x %08x %08x¥n", r_cnt,
           p.r_pc, p.w_ir, p.w_rrs1, p.w_ain, p.w_rslt2, w_led);
  always@(posedge r_clk) if(w_led!=0) $finish;
  initial #50000000 $finish;
endmodule
```

code161.v

```
  clock: r_pc     w_ir     w_rrs1   w_ain    r_rslt2  r_led
      1: 00000000 00000033 00000000 00000000 00000000 00000000
      2: 00000004 00000033 00000000 00000000 00000000 00000000
      3: 00000008 00000a13 00000000 00000000 00000000 00000000
      4: 0000000c 00300a93 00000000 00000003 00000003 00000000
      5: 00000010 00000633 00000000 00000000 00000000 00000000
      6: 00000014 00000593 00000000 00000000 00000000 00000000
      7: 00000018 40000413 00000000 00000400 00000400 00000000
```

trace_detail.txt の最初の数行

コンテストのルール (2/2)

- プロセッサの動作確認について，性能改善の結果、中間のトレース内容は変わっても良いが以下に注意すること.
  - プロセッサによる計算結果の VIO 値が正しくないプロセッサはランキング対象外です.
  - 課題プログラムに特化した不正な最適化も対象外です.
    - VIOの値だけ合うようにして，その値に関係しない途中の命令群をスキップする最適化など.
    - 不正な最適化を防止するため，提出されたプロセッサで課題プログラムとは別のプログラムも正しく動作するか教員/TAが（成績報告までに）確認することがある.
- プロセッサのコードは Verilog HDL で，main.v という名前のファイルに全て記述すること.
- コンテスト当日時点での最新のプロセッサ性能およびソースコードを提出可能
  - 意図: 事前提出版より更に性能向上したプロセッサをランキング対象に含めたい.
  - 方法: コンテスト当日午前7:00までに Slack で提出すること.
    - 事前の提出版から性能が変わっていない場合には，再提出は不要です.

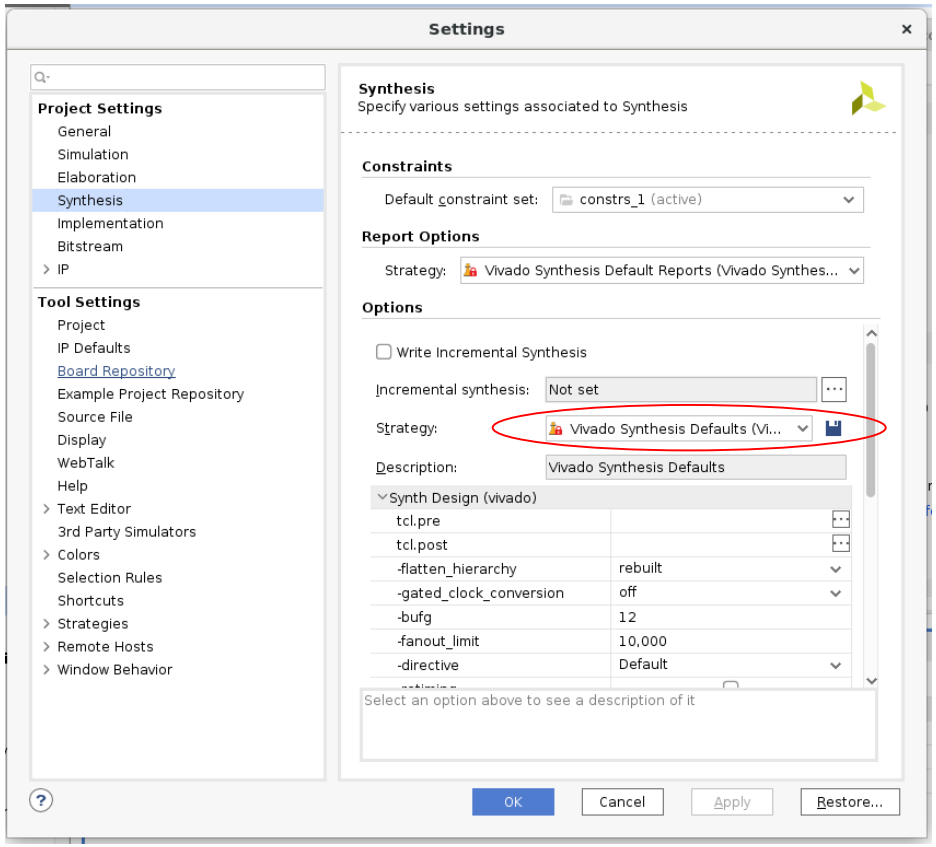
コンテストのスケジュール

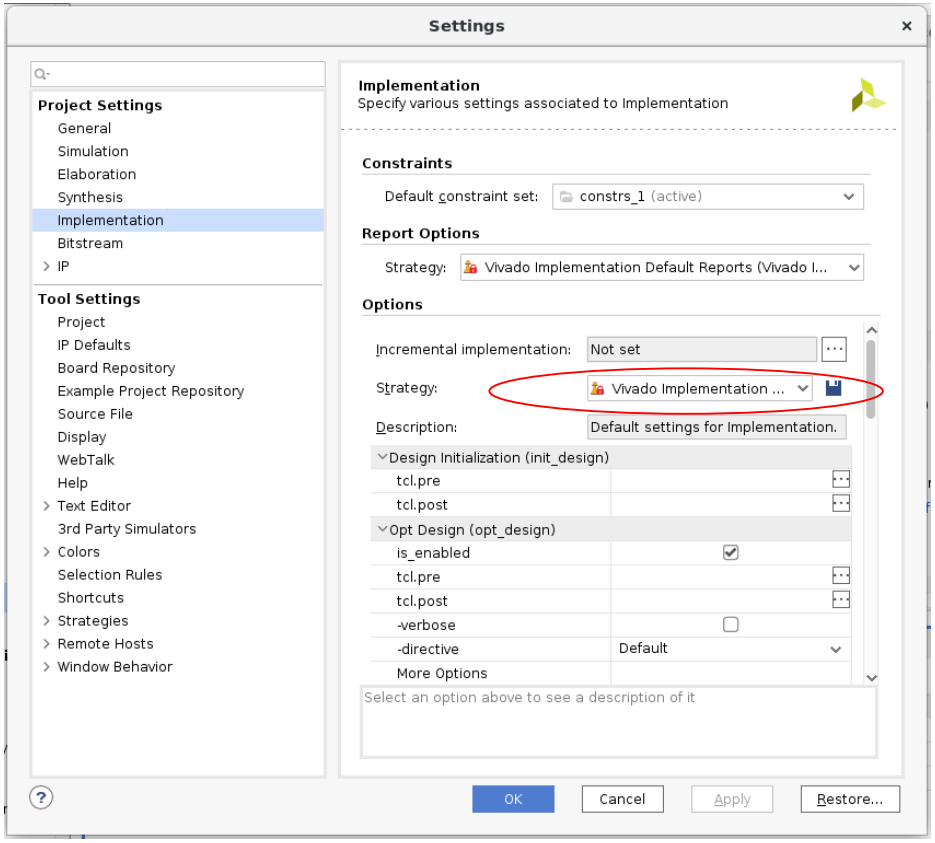
- 2023年5月30日(火) 23:00 までに，(1) 設計したプロセッサのソースコードと (2) 発表用の PowerPoint ファイルを提出
  - 発表用の PowerPoint ファイルの作成については，そのサンプルファイルの指示に従うこと.
  - PowerPoint ファイルの名前は，ContestSlideXX.pptx（XXはグループ番号）とすること.
  - 設計したプロセッサのコードを1つのファイルにまとめて，ファイル名は main.v とすること.
  - この講義用の Slack のダイレクトメッセージで，担当のTAあるいは教員に提出すること.

- 2023年6月1日(木) 8:50 - 12:25 コンテスト

補足：Vivado の最適化オプション

- Settings -> Synthesis -> Strategy : Vivado Synthesis Defaults
- Settings -> Implementation -> Strategy: Vivado Implementation Defaults



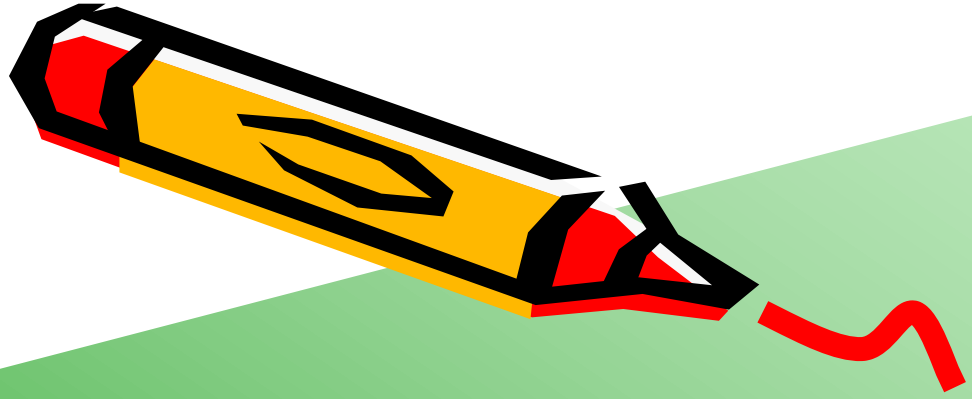


補足：Vivado の最適化オプション

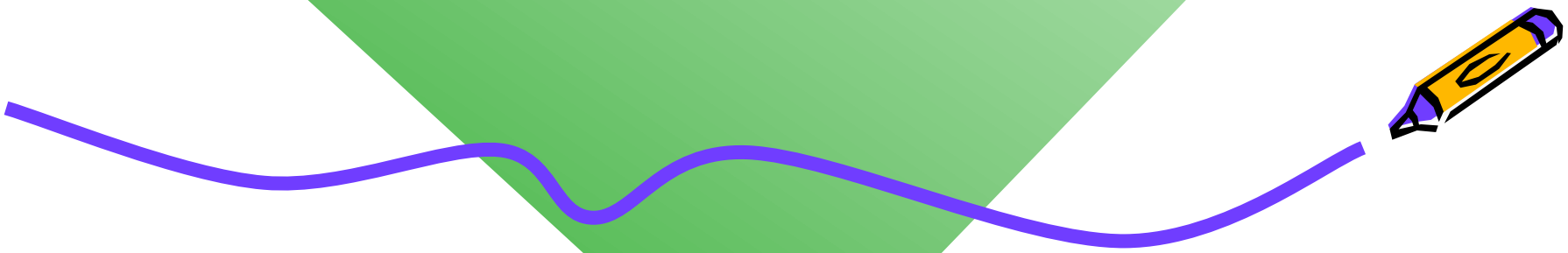
- プロセッサの高性能化のための推奨オプションは次の通り．デザインコンテストでは，これらのVivadoのオプションを適切なものに変更してもよい．

- Settings -> Synthesis -> Strategy :
  Flow_PerfOptimized_high
- Settings -> Implementation -> Strategy:
  Performance_ExplorePostRoutePhysOpt

- ただし，これらの最適化オプションを変更すると，論理合成と配置配線の時間が長くなるので，適切に使い分ける必要がある．



References



References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog