2023年度版

Ver. 2023-04-25b

Course number: CSC.T341

コンピュータ論理設計 演習(3)
Computer Logic Design Exercise(3)

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/

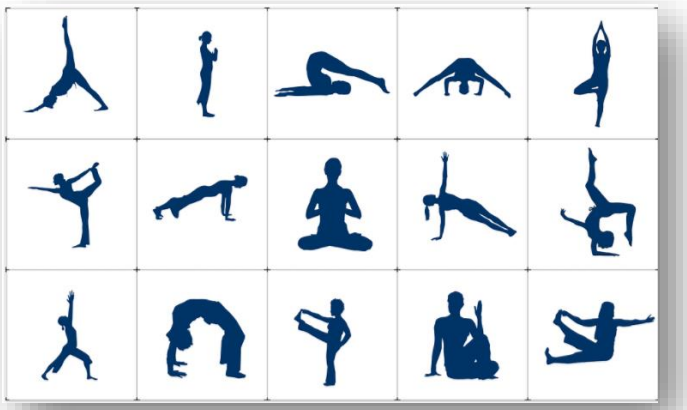
コンピュータ論理設計 演習(Exercise)の注意点

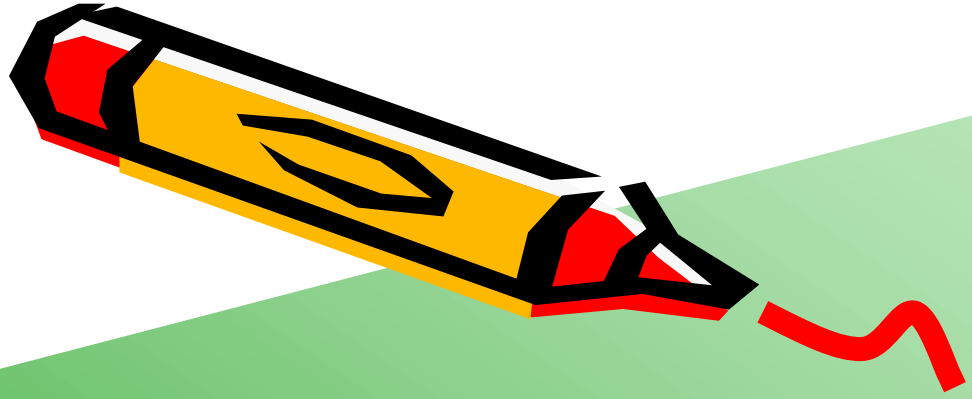
- 演習はACRiルームを利用します。
- 3～4人のグループを作成します．そのグループ内で情報を共有しながら演習を進めてください．
- 問題はグループ内で相談して解決する，あるいは，担当のTA(Teaching Assistant)や教員に質問してください．
- 演習には出席点があります．休まずにきちんと出席しましょう．
- 演習スライドにチェックポイントの図がある場所は，作業を確認してもらう場所です．すべてのチェックポイントをクリアしましょう．

- 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．

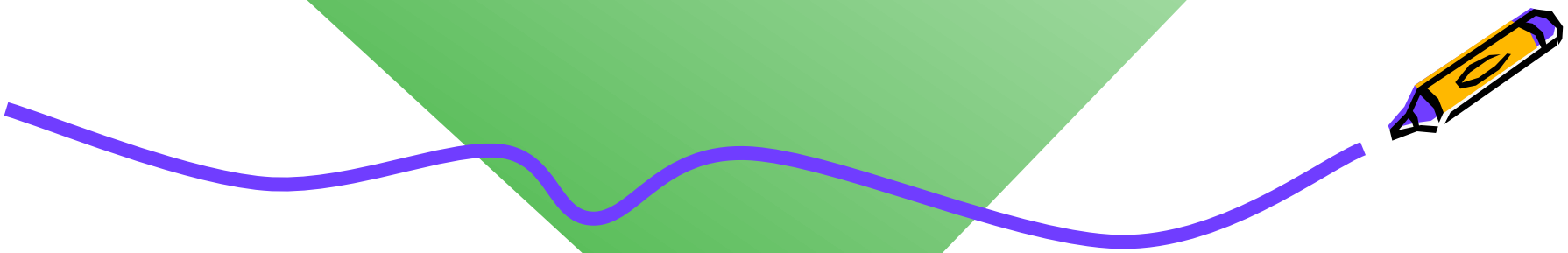
Exercise(3)

- Project_3
  - CMT (Clock Management Tile) を使って，FPGAの内部で 20MHz と 33MHz のクロック信号を生成する方法を学ぶ.
  - この方法を用いることで，設計したハードウェアの高性能化のために動作周波数を上げることができる.
- Project_4
  - 加算器，論理演算，シフターなどの主要な算術論理演算のハードウェア量を計測する．また，ALU全体のハードウェア量を計測する.
  - これにより，FPGA（xc7a35t）で利用できるハードウェア量を把握する.





Project_3



新しいプロジェクトの作成

- Project_1 と同様に，新しいプロジェクト project_3 を作成する．
  - 制約ファイルとして main11.xdc を登録する．
  - Verilog HDLファイルとして code057.v を登録する．vio_0 のコメントアウトを削除する．

code057.v

```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0]  w_led;

  reg        r_out = 0;
  reg [31:0] r_cnt = 0;
  always@(posedge w_clk) begin
    r_cnt <= (r_cnt==99999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? ~r_out : r_out;
  end
  assign w_led = {r_out, r_out, r_out, r_out};
  // vio_0 vio_00(w_clk, w_led[3], w_led[2], w_led[1], w_led[0]);
endmodule
```

新しい Vivado プロジェクトの作成とファイルの登録

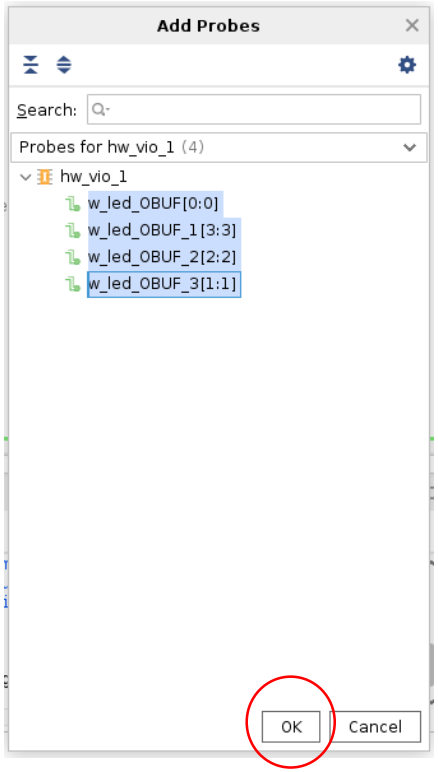
- 前回の演習を参考に，Vivado で新しいプロジェクト project_3 を作成する．
- Ubuntu で起動したターミナルで，次のコマンドを実行してファイルをコピーする．
  - /home/tu_kise は automount のディレクトリなので，アクセスしないとファイルが見えない．tabキーによる補完がうまく動作しないことがあるので注意する．
  - 最後の ls コマンドで，code057.v, code105.v, code106.v, main11.xdc が表示されることを確認．

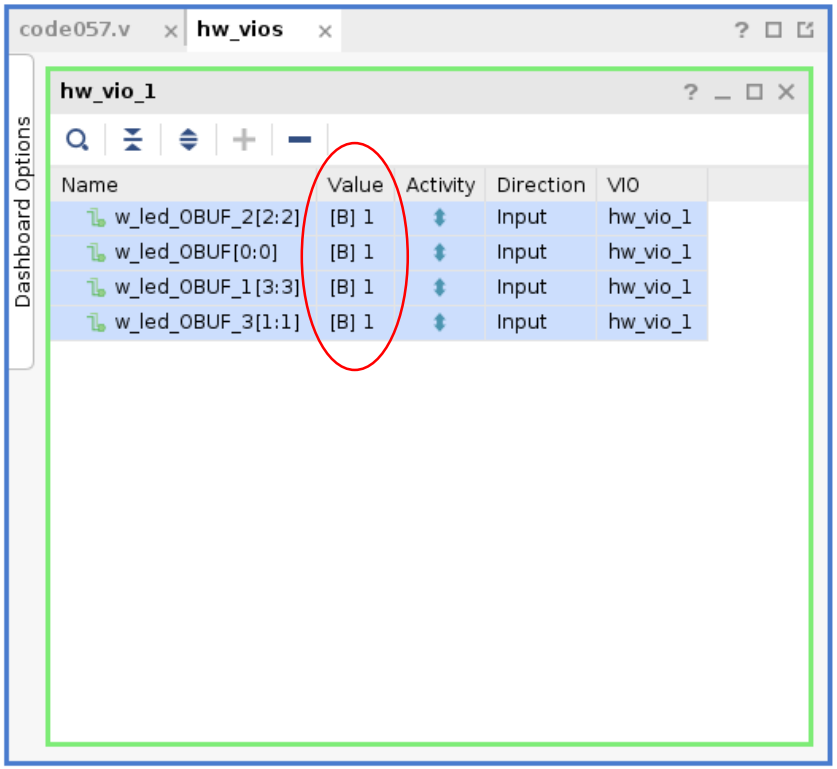
```
$ ls /home/tu_kise
$ cd ~/cld/project_3
$ cp /home/tu_kise/cld/2023/code057.v .
$ cp /home/tu_kise/cld/2023/code105.v .
$ cp /home/tu_kise/cld/2023/code106.v .
$ cp /home/tu_kise/cld/2023/main11.xdc .
$ ls
```

- Vivado で，project_3 の制約ファイルとして main11.xdc を登録する．
  - main11.xdc ファイルの内容を変更する必要はない．
- Vivado で，project_3 のVerilog HDLファイルとして code057.v を登録する．
- VIO の IP を生成して，次のスライドの通り，1秒毎に点滅することを確認する．

VIOを用いてFPGAの動作をリモートで確認

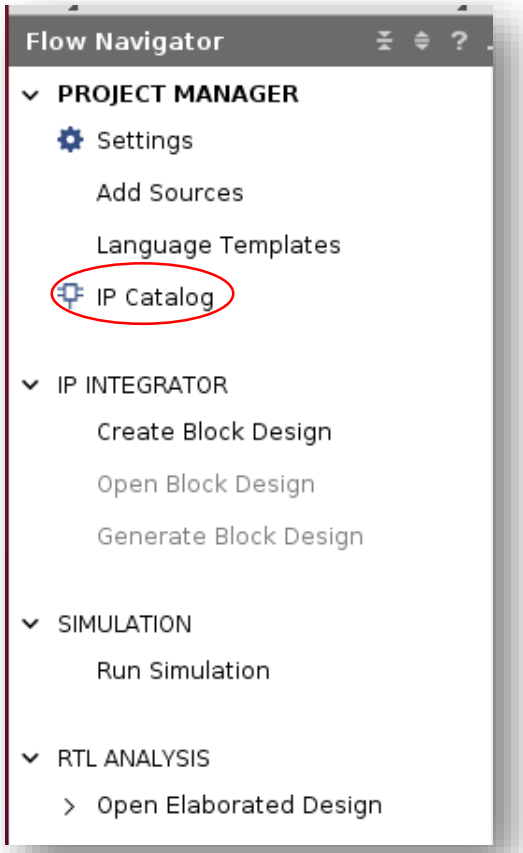
- Click + button in **hw_vio_1** window.
- Select w_led_OBUF[0:0], w_led_OBUF_1[3:3], w_led_OBUF_2[2:2], and w_led_OBUF_3[1:1] by clicking them pushing Shift button. Then click OK.
- You will see the values of four LEDs change every second!
- ここまでは Project_1 と同様の作業.

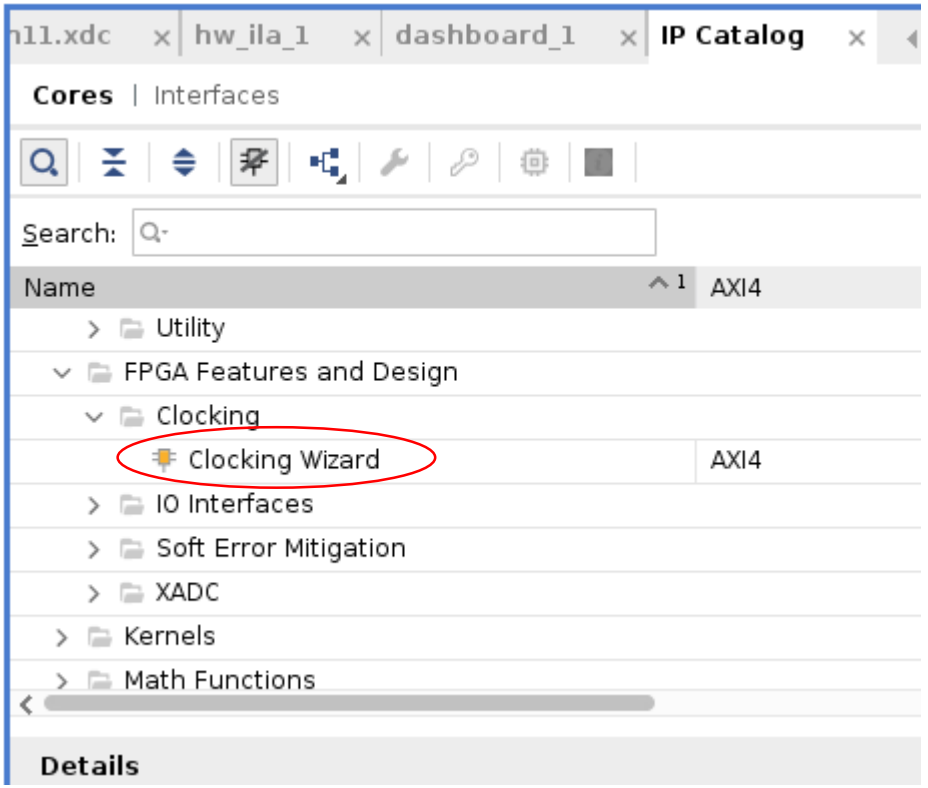




Clocking Wizard を起動する

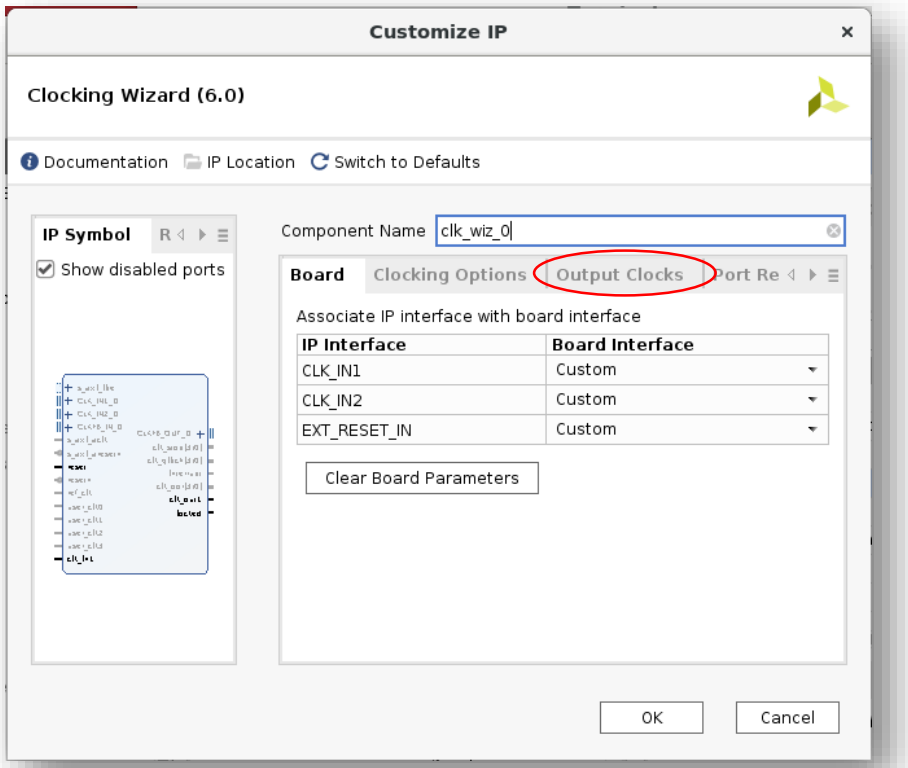
- Click **IP Catalog**
- Double click **Clocking Wizard** in IP Catalog window

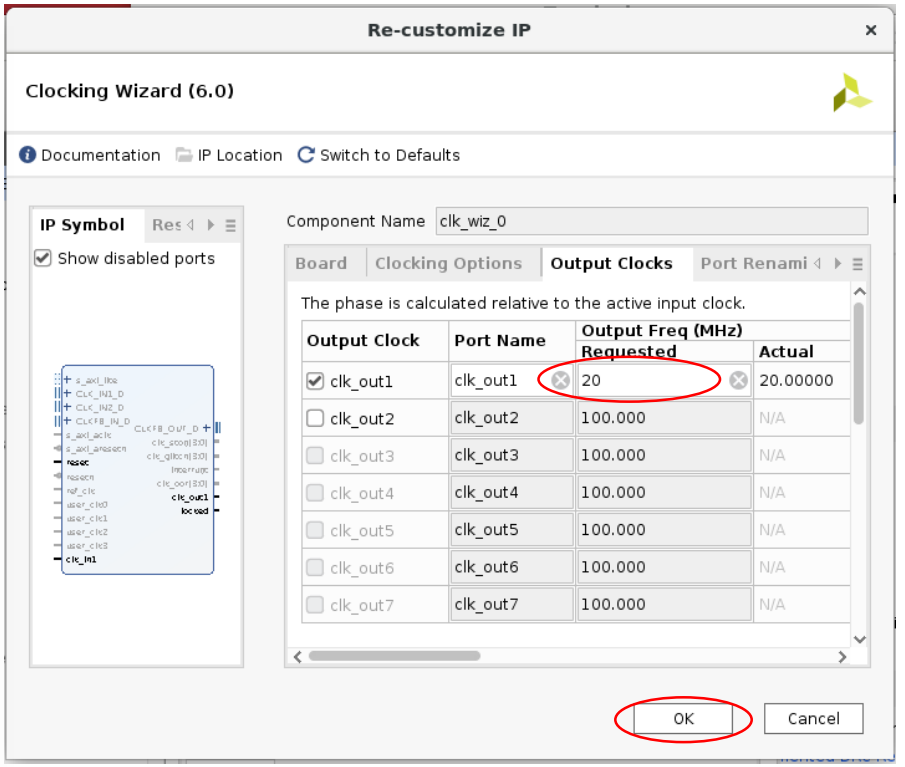




20MHzのクロックを出力する IP を生成する

- In Output Clocks, change the frequency from 100.000 to **20** for clk_out1 to generate 20MHz clock signal. Click **OK**.
- In Generate Output Products window, click **Generate**.



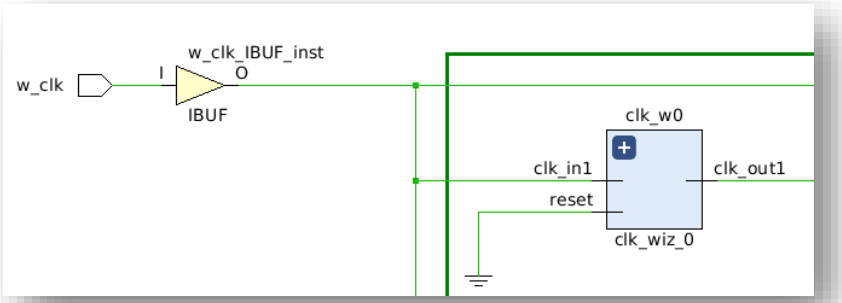


生成した IP を用いるようにコードを修正

- Verilog HDL のコードを，生成した IP を用いる様に code105.v の内容となるように変更する（赤色の文字の部分を入力あるいは変更する）.
- Synthesis, Implementation をおこない，Bitstreamファイルを作成，FPGAにコンフィギュレーションする.
- 20MHz で動作するので，レジスタの変化が遅くなる.

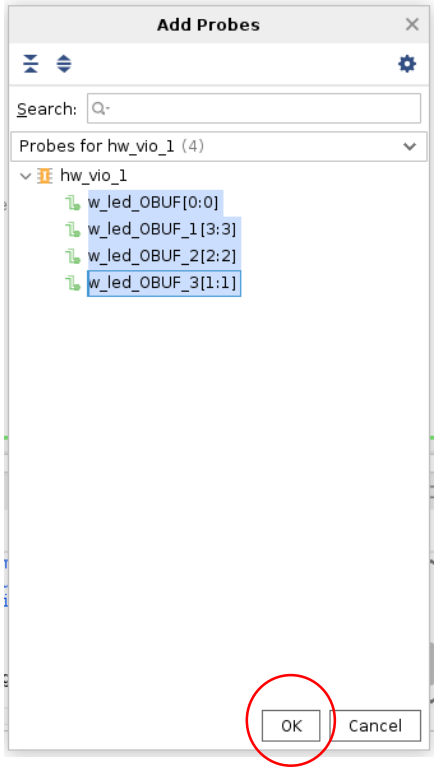
code105.v

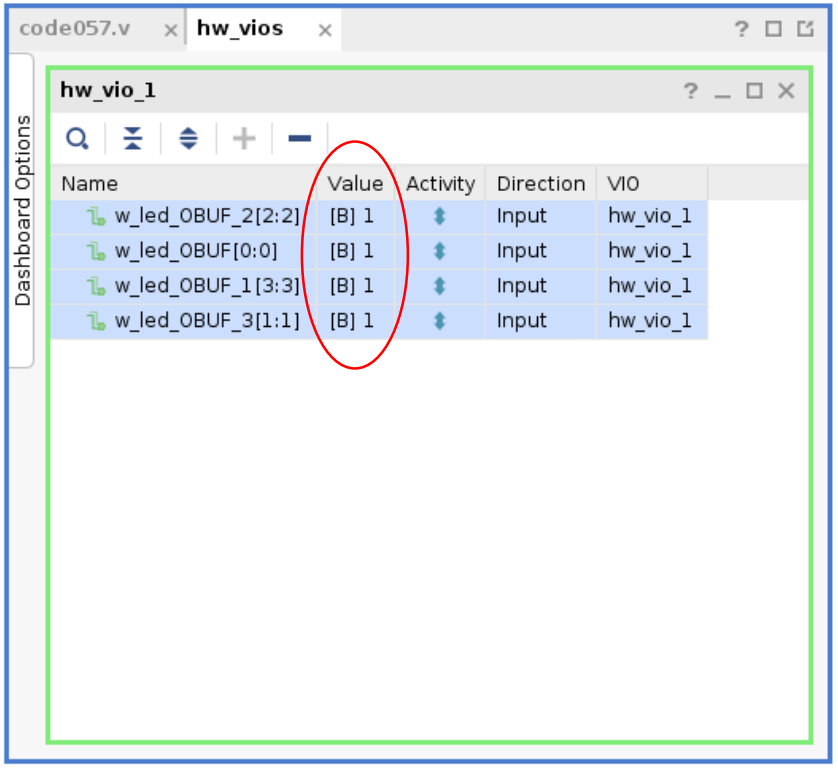
```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0]  w_led;

  wire w_clk2, w_locked;
  clk_wiz_0 clk_w0 (w_clk2, 0, w_locked, w_clk);

  reg        r_out = 0;
  reg [31:0] r_cnt = 0;
  always@(posedge w_clk2) begin
    r_cnt <= (r_cnt==99999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? ~r_out : r_out;
  end
  assign w_led = {r_out, r_out, r_out, r_out};
  vio_0 vio_00(w_clk2, w_led[3], w_led[2], w_led[1], w_led[0]);
endmodule
```



VIOを用いてFPGAの動作をリモートで確認

- Click + button in **hw_vio_1** window.
- Select w_led_OBUF[0:0], w_led_OBUF_1[3:3], w_led_OBUF_2[2:2], and w_led_OBUF_3[1:1] by clicking them pushing Shift button. Then click OK.
- You will see the values of four LEDs change every five seconds because the 20MHz clock signal w_clk2 is used !





演習（IPの構成方法，100MHzと33MHzクロックを使う回路）

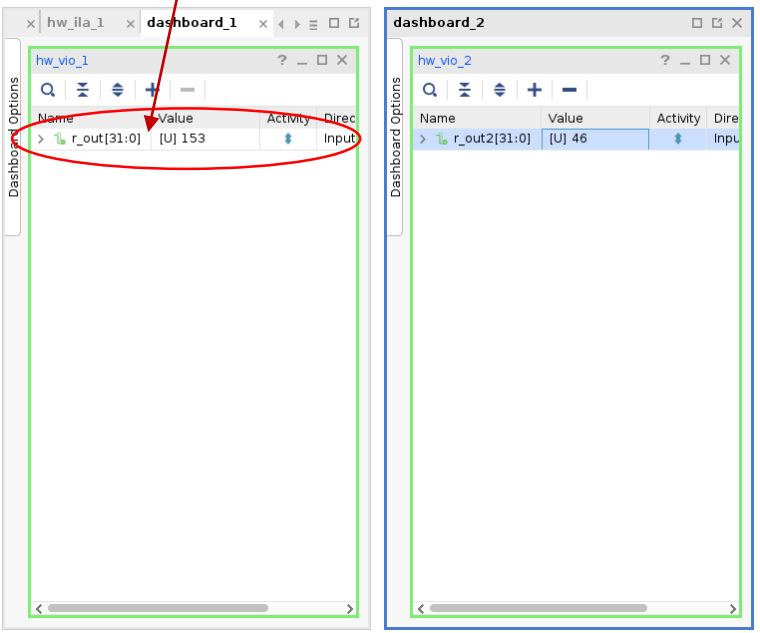
- 100MHzのクロック信号 w_clk と，33MHzのクロック信号 w_clk2 で異なるカウンタ r_cnt, r_cnt2 をインクリメントする次のコード code106.v をFPGAで動かす．
- プロジェクトに登録されている Verilog HDL のコードを code106.v の内容となるように変更すること．
- r_out2 と比べて r_out は3倍の動作周波数で動作しているので，例えば，VIO r_out2の値が10の時に，r_outの値が約30になっているはずである．

code106.v

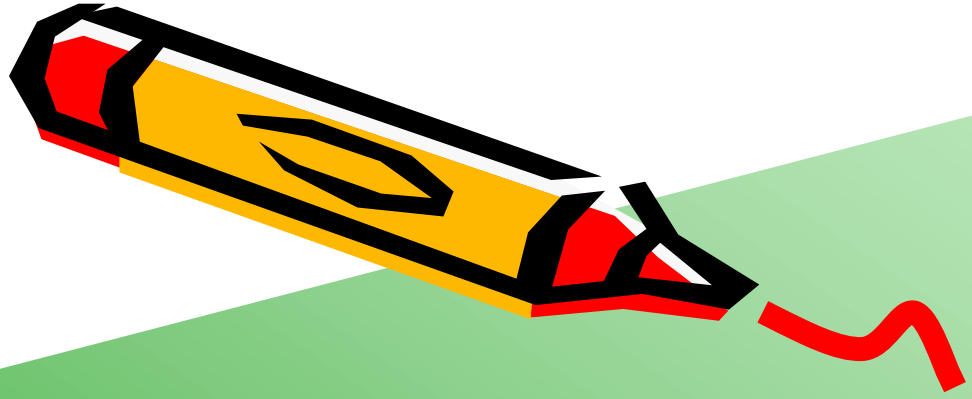
```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0]  w_led;

  wire w_clk2, w_locked; // 33MHz clock
  clk_wiz_0 clk_w0 (w_clk2, 0, w_locked, w_clk);

  reg [31:0] r_out = 0;
  reg [31:0] r_cnt = 0;
  always@(posedge w_clk) begin // 100MHz
    r_cnt <= (r_cnt==99999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? r_out+1 : r_out;
  end

  reg [31:0] r_out2 = 0;
  reg [31:0] r_cnt2 = 0;
  always@(posedge w_clk2) begin // 33MHz
    r_cnt2 <= (r_cnt2==99999999) ? 0 : r_cnt2 +1;
    r_out2 <= (r_cnt2==0) ? r_out2+1 : r_out2;
  end

  assign w_led = r_out[3:0];
  vio_0 vio_00(w_clk, r_out);
  vio_0 vio_01(w_clk2, r_out2);
endmodule
```

演習（IPの構成方法，100MHzと33MHzクロックを使う回路）

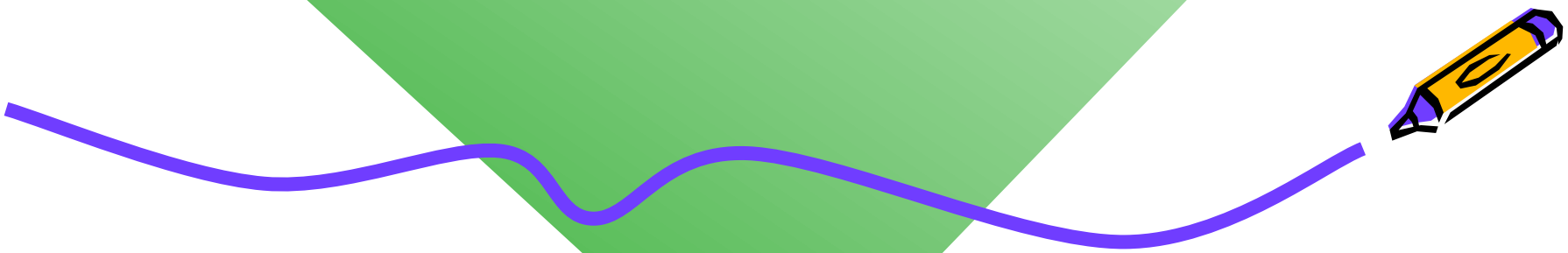
- Clocking Wizard でパラメータを適切に修正して，clk_wiz_0 を生成する．
- 32ビットのレジスタの値を VIO で表示できるように修正した vio_0 を生成する．
- 論理合成，配置配線，bitstreamファイルを生成し，FPGAをコンフィギュレーションする．
- **hw_vio_1** ウィンドウと **hw_vio_2** ウィンドウを開く．
- それぞれのウィンドウで，r_out の値と r_out2 の値を表示する．
- それぞれのウィンドウで表示する32ビットのレジスタの値を，符号なし10進数の表示 (Unsigned Decimal) に変更する．（**右クリック** -> Radix -> Unsigned Decimal）
- 2つのウィンドウを左右に表示して，その画面を「担当の教員あるいはTAに」示すこと．



Check Point 3



Project_4



新しい Vivado プロジェクトの作成とファイルの登録

- 前回の演習を参考に，Vivado で新しいプロジェクト project_4 を作成する．
- Ubuntu で起動したターミナルで，次のコマンドを実行してファイルをコピーする．
  - /home/tu_kise は automount のディレクトリなので，アクセスしないとファイルが見えない．tabキーによる補完がうまく動作しないことがあるので注意する．
  - 最後の ls コマンドで，code101.v, main11.xdc, cp4.txt が表示されることを確認．

```
$ ls /home/tu_kise
$ cd ~/cld/project_4
$ cp /home/tu_kise/cld/2023/code101.v .
$ cp /home/tu_kise/cld/2023/main11.xdc .
$ cp /home/tu_kise/cld/2023/cp4.txt .
$ ls
```

- Vivado で，project_4 の制約ファイルとして main11.xdc を登録する．
  - main11.xdc ファイルの内容を変更する必要はない．
- Vivado で，project_4 のVerilog HDLファイルとして code101.v を登録する．

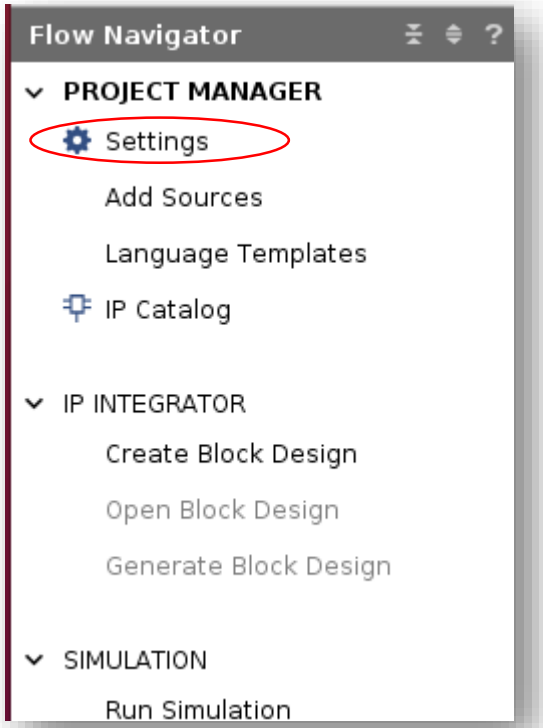
Code101.v ALU(Arithmetic and Logic Unit)の記述

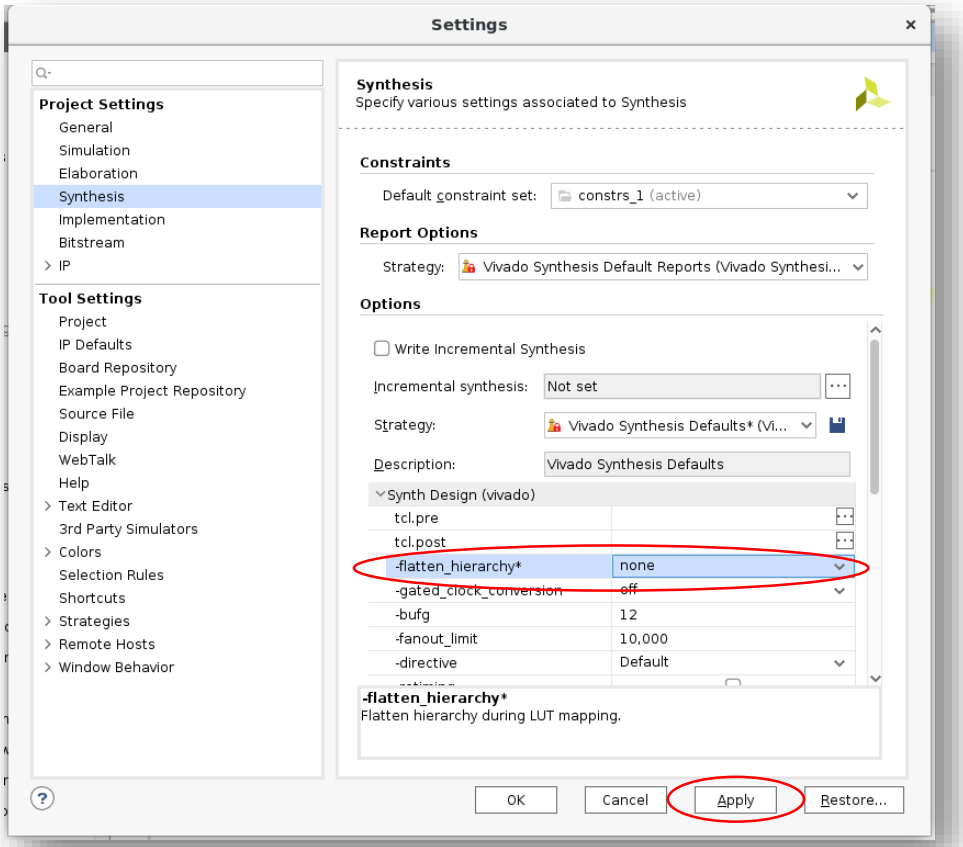
```
module m_main (w_clk, w_a, w_c, w_dout, w_zero);
    input  wire w_clk, w_a, w_c;
    output wire w_dout, w_zero;
    reg [31:0] r_a=0, r_b=0;
    reg [3:0] r_c=0;
    wire [31:0] w_out;
    assign w_dout = ^w_out;
    always@(posedge w_clk) begin
        r_a <= {w_a, r_a[31:1]};
        r_b <= w_out;
        r_c <= {w_c, r_c[3:1]};
    end
    m_ALU m_ALU0 (r_c, r_a, r_b, w_out, w_zero);
endmodule

module m_ALU (w_Ctl, w_A, w_B, r_Out, w_Zero);
    input wire  [3:0] w_Ctl;
    input wire [31:0] w_A, w_B;
    output reg [31:0] r_Out;
    output wire w_Zero;
    assign w_Zero = (r_Out==0);

    always@(*)
      case (w_Ctl)
          0:  r_Out <= w_A & w_B;
          1:  r_Out <= w_A | w_B;
          2:  r_Out <= w_A + w_B;
          6:  r_Out <= w_A - w_B;
          7:  r_Out <= (w_A < w_B) ? 1 : 0;
          12: r_Out <= ~(w_A | w_B);
          default: r_Out <= 0;
      endcase
endmodule
```

code101.v

設定の変更（モジュール間の最適化をしない，モジュールのハードウェア量を計測するため）

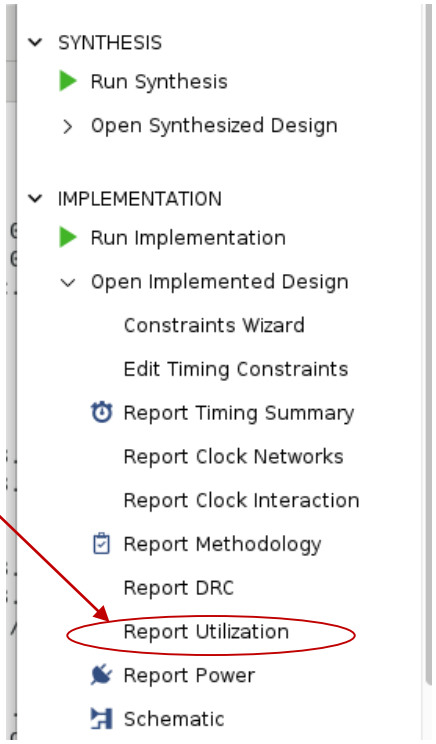
- Click Settings.
- Select **none** for **–flatten_hierarchy** and click **Apply**.
- If you have **Create New Run** window, select **No** in the window.
- Click **OK** in Setting window.





ハードウェア量の計測

- Implementation をおこない，Report Utilization をクリックする．
  - m_ALU0 が 168 個の LUT を用いることがわかる．



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (210)	BUFGCTRL (32)
N m_main	175	68	50	175	5	1
m_ALU0 (m_ALU)	168	0	49	168	0	0

AND と OR のそれぞれのハードウェア量を計測する場合

- Verilog HDL のコードを編集して AND (&) のみのALUとして，同様にハードウェア量を計測する.
- 次に，OR (|) のみのALUとして、同様にハードウェア量を計測する.
- 同様に，加算 (+)，減算 (-)，大小比較 (<)，NOR (~ |) のハードウェア量を計測する.
- また，全てのコメントアウトを削除して，全ての演算を含む ALU 全体のハードウェア量を計測する.

```
module m_ALU (w_Ctl, w_A, w_B, r_Out, w_Zero);
    input [3:0] w_Ctl;
    input [31:0] w_A, w_B;
    output reg [31:0] r_Out;
    output wire w_Zero;
    assign w_Zero = (r_Out==0);

    always@(*)
      case (w_Ctl)
            0:  r_Out <= w_A & w_B;
   //       1:  r_Out <= w_A | w_B;
   //       2:  r_Out <= w_A + w_B;
   //       6:  r_Out <= w_A - w_B;
   //       7:  r_Out <= (w_A < w_B) ? 1 : 0;
   //       12: r_Out <= ~(w_A | w_B);
          default: r_Out <= 0;
      endcase
endmodule
```

AND のハードウェア量を計測する場合

```
module m_ALU (w_Ctl, w_A, w_B, r_Out, w_Zero);
    input [3:0] w_Ctl;
    input [31:0] w_A, w_B;
    output reg [31:0] r_Out;
    output wire w_Zero;
    assign w_Zero = (r_Out==0);

    always@(*)
      case (w_Ctl)
   //       0:  r_Out <= w_A & w_B;
            1:  r_Out <= w_A | w_B;
   //       2:  r_Out <= w_A + w_B;
   //       6:  r_Out <= w_A - w_B;
   //       7:  r_Out <= (w_A < w_B) ? 1 : 0;
   //       12: r_Out <= ~(w_A | w_B);
          default: r_Out <= 0;
      endcase
endmodule
```

OR のハードウェア量を計測する場合

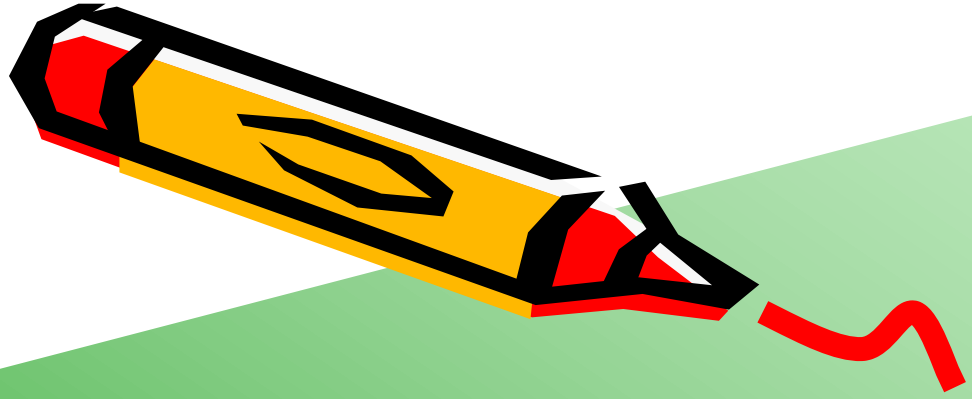
演習（計測したハードウェア量をまとめる）

- cp4.txt の xxxxx, yyyyy の部分を適切な数字で置き換えること.
- それぞれの演算を含む ALU で必要となる LUT の数 (# of LUTs), それを FPGA に搭載できる数 (# of modules on an FPGA) を記入する. FPGA に搭載されている LUT が 20,800個であることから, 20,800/(# of LUTs) によって, FPGA に搭載できるモジュールの数を計算すること.
- また, &, |, +, -, <, ~ | の6種類の演算を含む ALU で必要となる「LUTの数」の和を計算して, Sum の行に記入すること. このLUTの数から, FPGA に搭載できるモジュール数を計算して記入すること.
- すべての演算を含む ALU（code101.v の記述）のために必要となる LUT の数を ALU の行に記入する. 同様に, FPGA に搭載できるモジュール数を記入すること.
- 「Sum の行に記入した LUT の数と, ALU の行に記入した LUT の数が異なる理由」を考えて, 「担当の教員あるいはTA」に伝えること.

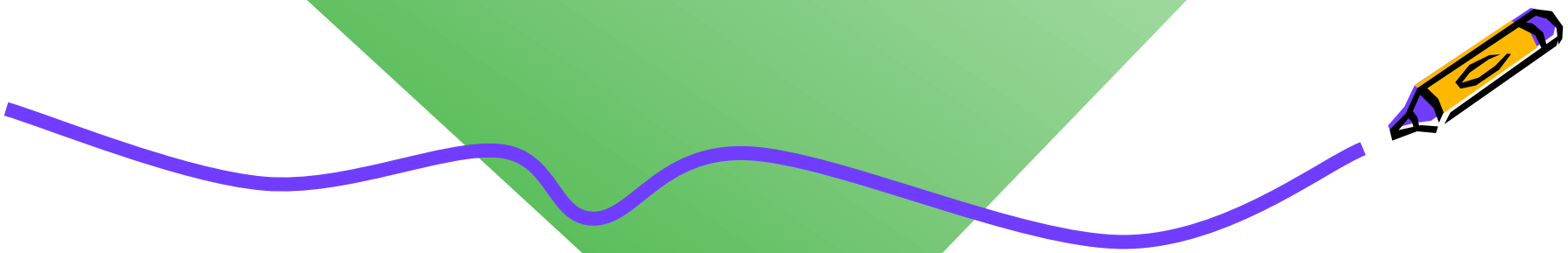
cp4.txt

operation	# of LUTs	# of modules on an FPGA
r_Out <= w_A & w_B;	xxxxx	yyyyy
r_Out <= w_A | w_B;	xxxxx	yyyyy
r_Out <= w_A + w_B;	xxxxx	yyyyy
r_Out <= w_A - w_B;	xxxxx	yyyyy
r_Out <= (w_A < w_B) ? 1 : 0;	xxxxx	yyyyy
r_Out <= ~(w_A | w_B);	xxxxx	yyyyy
Sum	xxxxx	yyyyy
ALU	xxxxx	yyyyy

Check Point 4



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