

# 実験課題10

## 「システムエレクトロニクス実験 プロジェクト」

# 実験の日程

項目	日程	
1	11/17	導入説明・FPGAマッピング・VerilogHDL記述の習得(Lチカ他)
2	11/21	以降の実装内容の発表(午後1時開始)
3	11/22	各自実験
4	11/24	
5	11/28	
6	11/29	
7	12/1	
8	12/5	
9	12/6	
12	12/8	最終発表会(午後4時開始)

# 実験の目的

- IoT時代にハードウェア実装に適した通信・情報処理の代表的なアルゴリズムを理解する
  - エラー訂正、FFTとデジタルフィルタ等
- ハードウェア記述によりハードウェア実装を行うことで、ソフトウェアとハードウェアの違いについて理解する

# 本日の実験課題

- 課題1: VerilogHDLを用いてFPGAを動作させる基礎の習得
  - Lチカの実装:
    - そのまま実装
    - 点灯周期の変更(…、— — —等)
    - 点灯パタンの変更(—・—等)
    - ボタンを押すことで、点灯パタンの変化: 状態の変化

# 本日の実験課題..2

- 課題2:FPGAボードの周辺チップの制御を通してハードウェア動作の理解
  1. DVIインターフェースでディスプレイに表示
  2. ボタンを使った画面操作
  3. シリアル通信の実現(同一ボード内)

# 11月21日発表会

- 本日の課題2の結果報告
  - 課題2－1、課題2－2のデモ
  - 課題2－3:シリアル通信におけるエラーレートの結果の説明
- 自由課題で何を実現するかの発表
  - 概要:動作仕様、性能
  - どのようなアルゴリズム
  - どのようなアーキテクチャ

# 12月8日作品内容発表

- ・一人5分、スライドを準備
- ・作成した作品の概要(仕様、機能、性能など)の目標を説明する
- ・作品で実装したアルゴリズムを簡単に説明する
- ・ハードウェア実装結果(Verilog行数、合成結果:最大動作周波数、使用ロジック数、使用演算器数、使用メモリ数)の説明
- ・FPGAにダウンロードし作品を紹介する
- ・4:00開始、期待値としては4:30頃終了

# 実験レポートに関する注意事項

- 導入部の課題：
  - 実験内容、実装内容に関して理解したことを明記する
  - VerilogHDLの記述量(コメントを除いた行数)、ハードウェア量・性能等(コンパイル結果)を確認する
  - 実行結果を明確にする
- 演習
  - 実装内容に関して詳細の説明：特にアルゴリズム、その実装アーキテクチャさらに実装結果に関して、記述量、実行に必要なクロック数、スループット(CPIに相当する内容)、ハードウェア量、性能等(コンパイル結果)を確認
  - 実行結果を明確にする

# レポート課題

1. IoTとよく言われるがそれに関して調べ、IoT機器に必要とされる、機能、仕様(スペック、性能等)をまとめよ  
とくにそれらのハードウェア実現の観点に着目してまとめ、マイコン+ソフトの実装との比較について論じること
2. IoT機器(センサーネットワーク)において実現されているセンサーについて、2点以上をとりあげ、機能、仕様をまとめよ
3. αGOのハードウェア実装について調べて簡潔にまとめよ

# いくつかよい例：実験結果

表1 ハードウェアによる記述量と合成結果

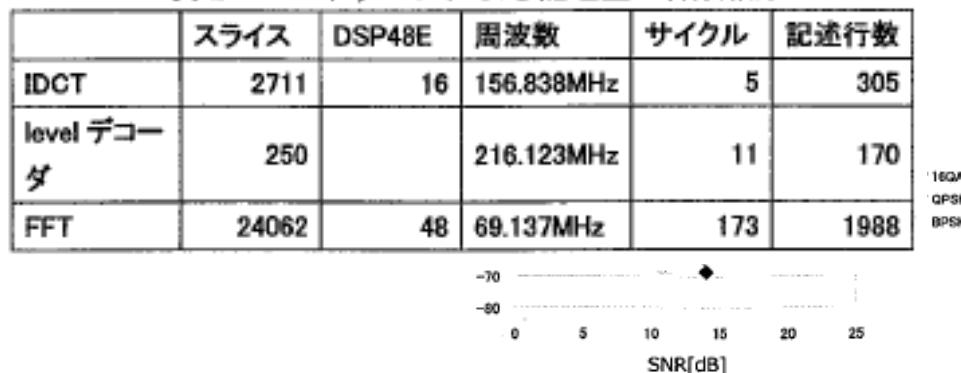


図6 BER-SNRのグラフ

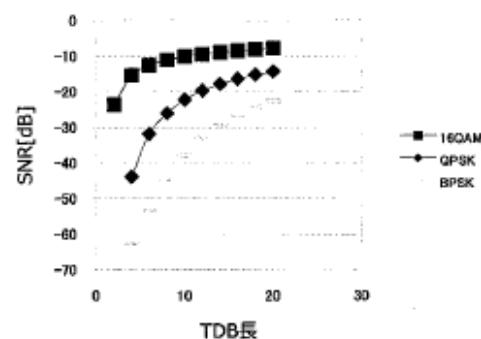


図7 BER-TDB長のグラフ

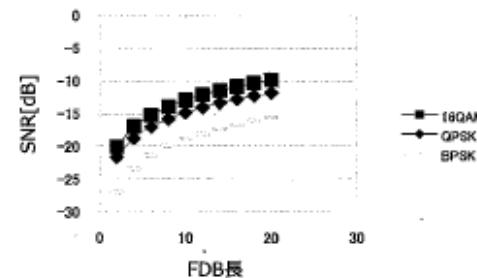


図8 BER-FDBのグラフ

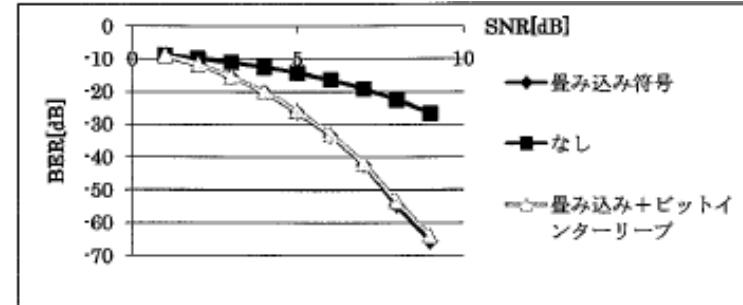


図9 SNRに対する誤り訂正符号とBERの関係

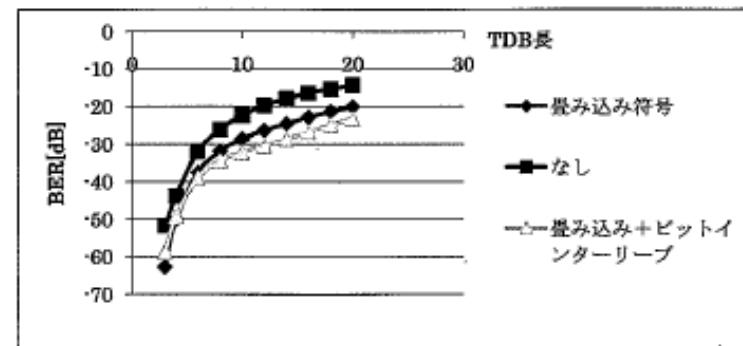


図10 TDBに対する誤り訂正符号とBERの関係

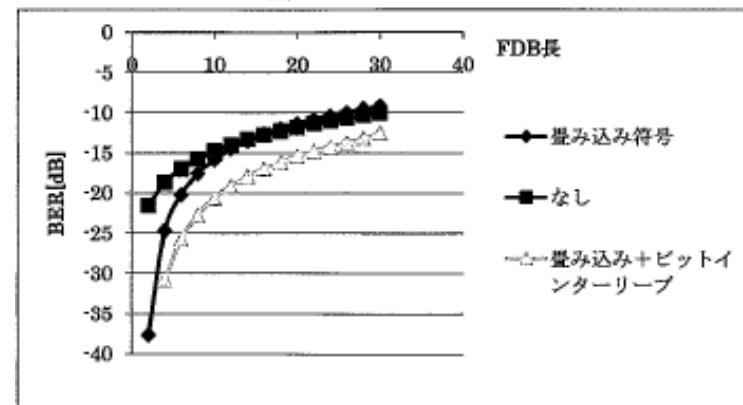


図11 FDBに対する誤り訂正符号とBERの関係

# いくつかよい例：考察

## 5.1 ソフトウェアとハードウェアの性能比較

FFT を PC 上にソフトウェアで行った時、1 秒間に変換できるサンプル数は 4M ぐらいだった。ハードウェアの場合、最大動作周波数とサイクルから計算すると

$$\frac{130MHz}{770} \cdot 64Sample = 10M Sample/s$$

確かにハードウェアの方が早いが、速度が 2 倍しか増えてない。FPGA の性能は PC よりはるかに低いにも関わらず早くなるという意味ですごいと思うが、さらにパイプラインなどを使って性能が高まることが大事だ。

ビタビ復号の場合

$$\frac{200MHz}{200} \cdot 62b = 62Mbps \gg 6.4Mbps$$

となり、ハードウェアの方が十倍も早かった。

## 5.2 符号による誤り率の向上

図 4-6 では符号かけた場合と無い場合の誤り率を比較した。ノイズの種類によって異なっている。例えば、White ノイズで

### (2)BER と変調方式、誤り訂正符号の関係

図 6~8 のグラフからわかるように、いずれのノイズに対しても、BER 率の低い順は、BPSK、QPSK、16QAM となっている。これは BPSK は 1 ビット、QPSK は 2bit、16QAM は 4 bit を送信するため、IQ 平面上においてそれぞれ 2 個、4 個、16 個の点が配置されるので、点が近いほど誤り率が大きくなるからである。

誤り訂正符号に関しては、図 9~11 のグラフから、疊み込み符号化を行うことによって、BER が低くなっていることがわかる。

それに加えてインターリープを行うと、TDB や FDB では BER が改善されるが、SN 比に関しては効果がないことがわかる。これは、インターリープは連続したノイズを分離するためのものであるため、ランダムノイズに関しては無意味であるからである。

同様にバースト長に関しても、長いノイズほどインターリープの有無による BER の差が大きくなる。

# 実験を通しての注意

- ・分からぬことがあつたら遠慮なくTAに聞いてください。(前で説明をしていない時はTAが巡回していなくてもどんどん呼んでください)
- ・パソコン、ディスプレイ、FPGAの電源は切って帰ってください。
- ・Verilogのコードは全て持ち帰って大丈夫です。
- ・家で続きをやりたい人はiVerilogを使うといいと思います。(テストベンチがないものは自分で書く必要があります。)

# 実験課題1:FPGA合成とXilinx ISE

- ターミナルを開き

```
cd ~/
```

```
cp -r /mnt/share/oneseg2016/day1/ikedajikken2016.tar.gz ~/
```

```
tar xvf ikedajikken2016.tar.gz
```

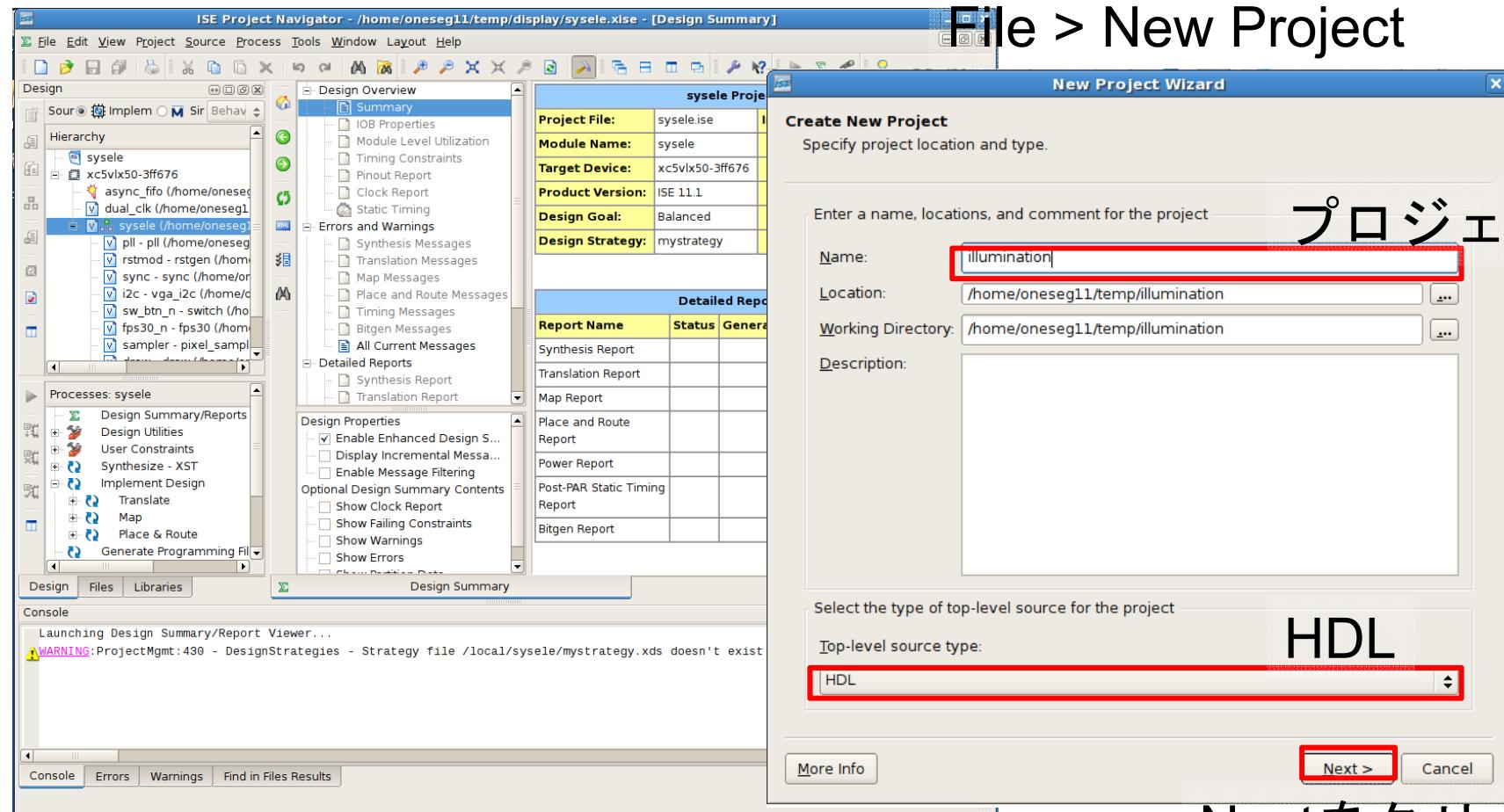
```
cp -r /mnt/share/oneseg2016/day1/LED.tar.gz ~/
```

```
tar xvf LED.tar.gz
```

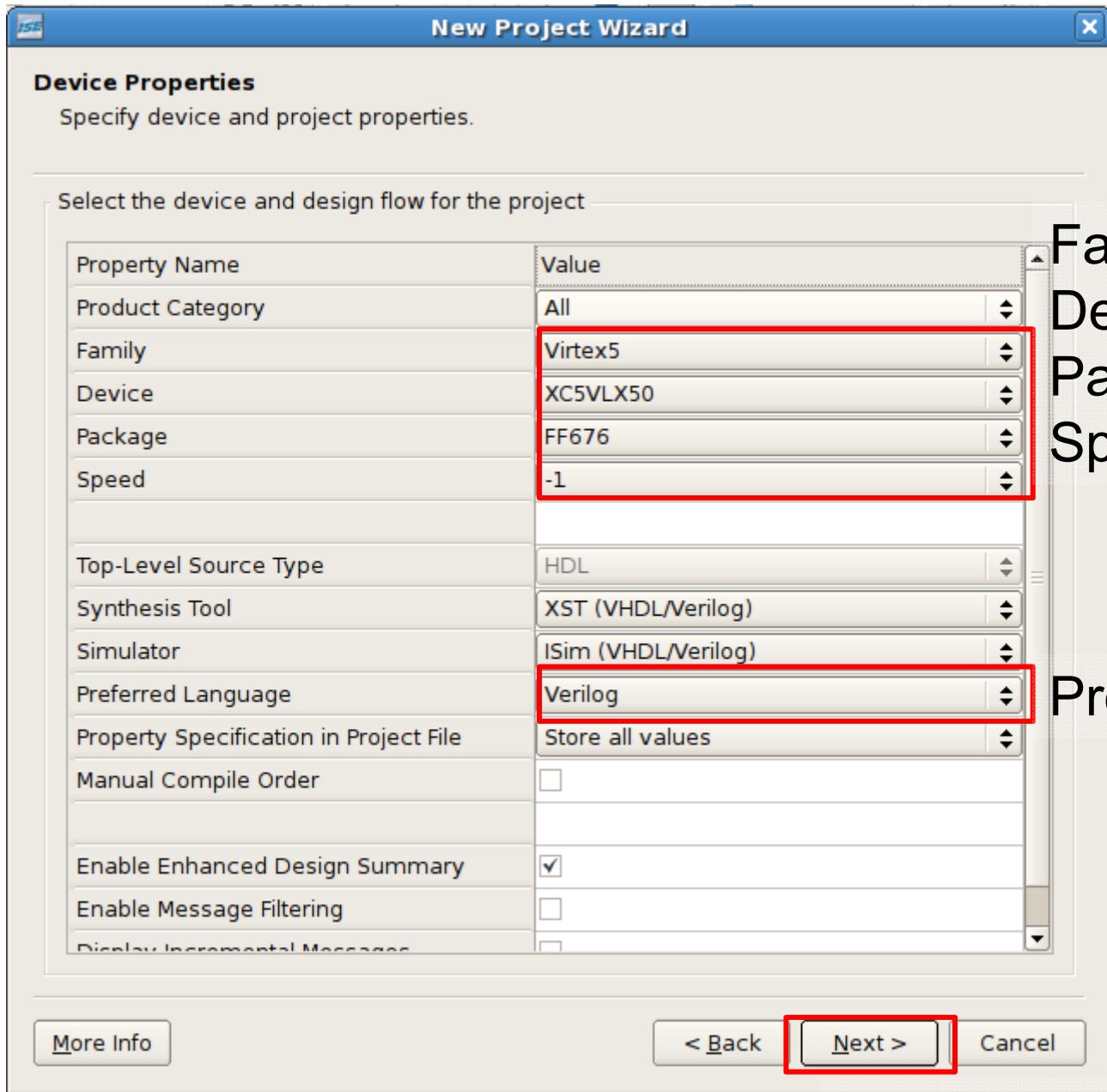
```
cd ~/LED
```

# 実験課題1 : FPGA合成とXilinx ISE

- ターミナルを開き ise で起動
  - 以後Yes, Noがいくつか出てくるが、特に指定がない場合Yesを選択



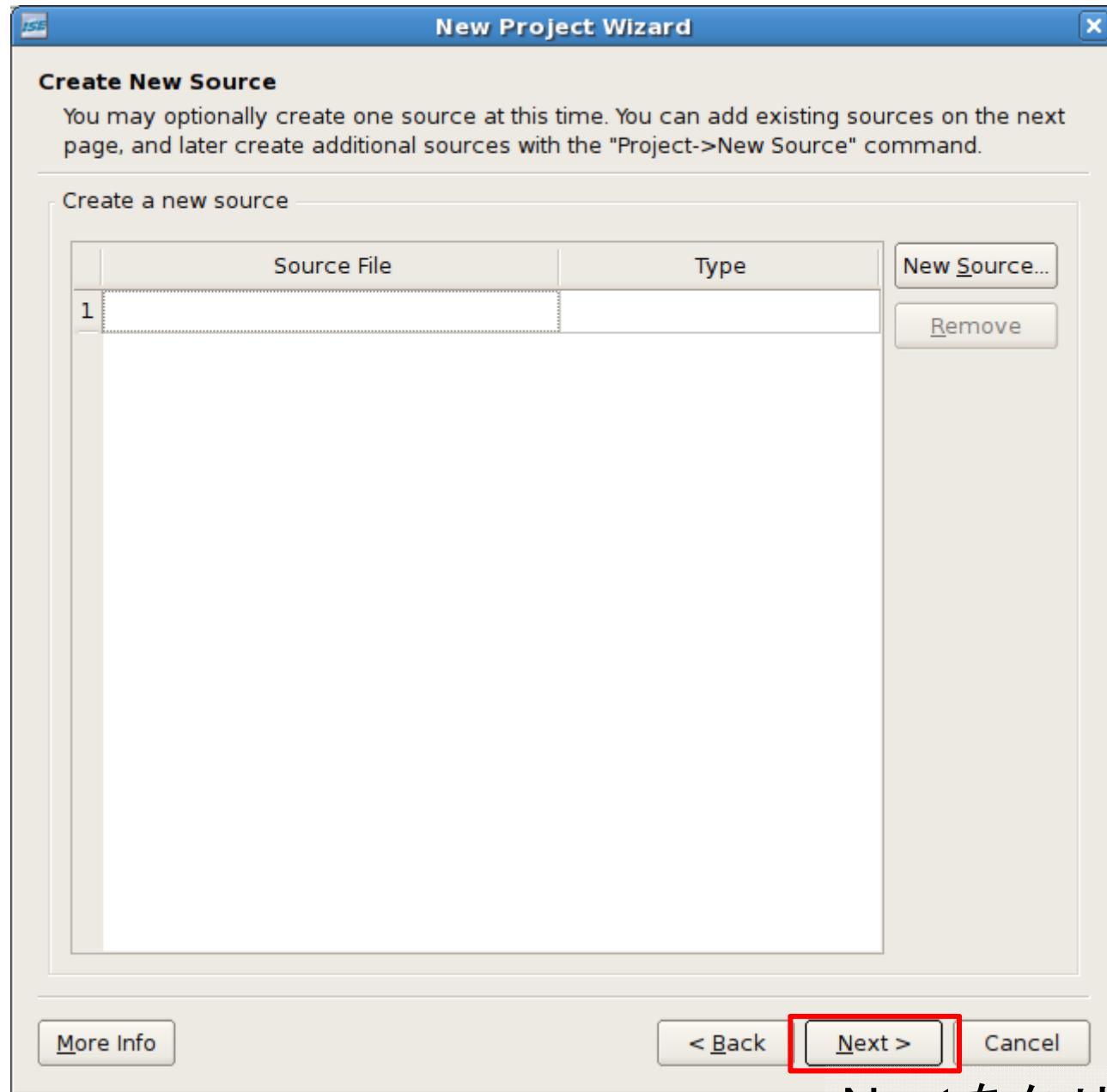
Nextをクリック



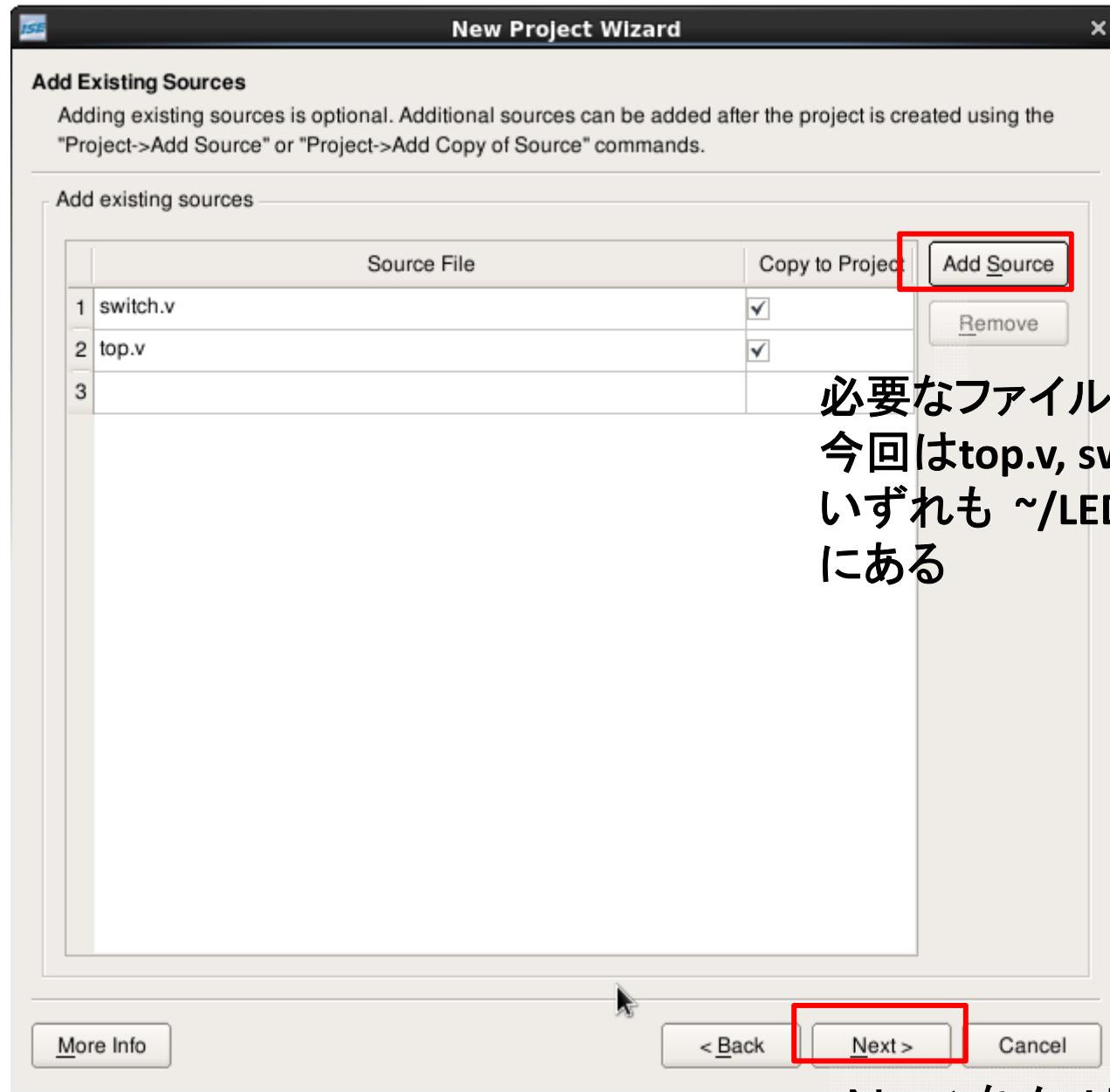
Family: Virtex5  
Device: XC5VLX50  
Package: FF676  
Speed: -1

Pref...: Verilog

Nextをクリック

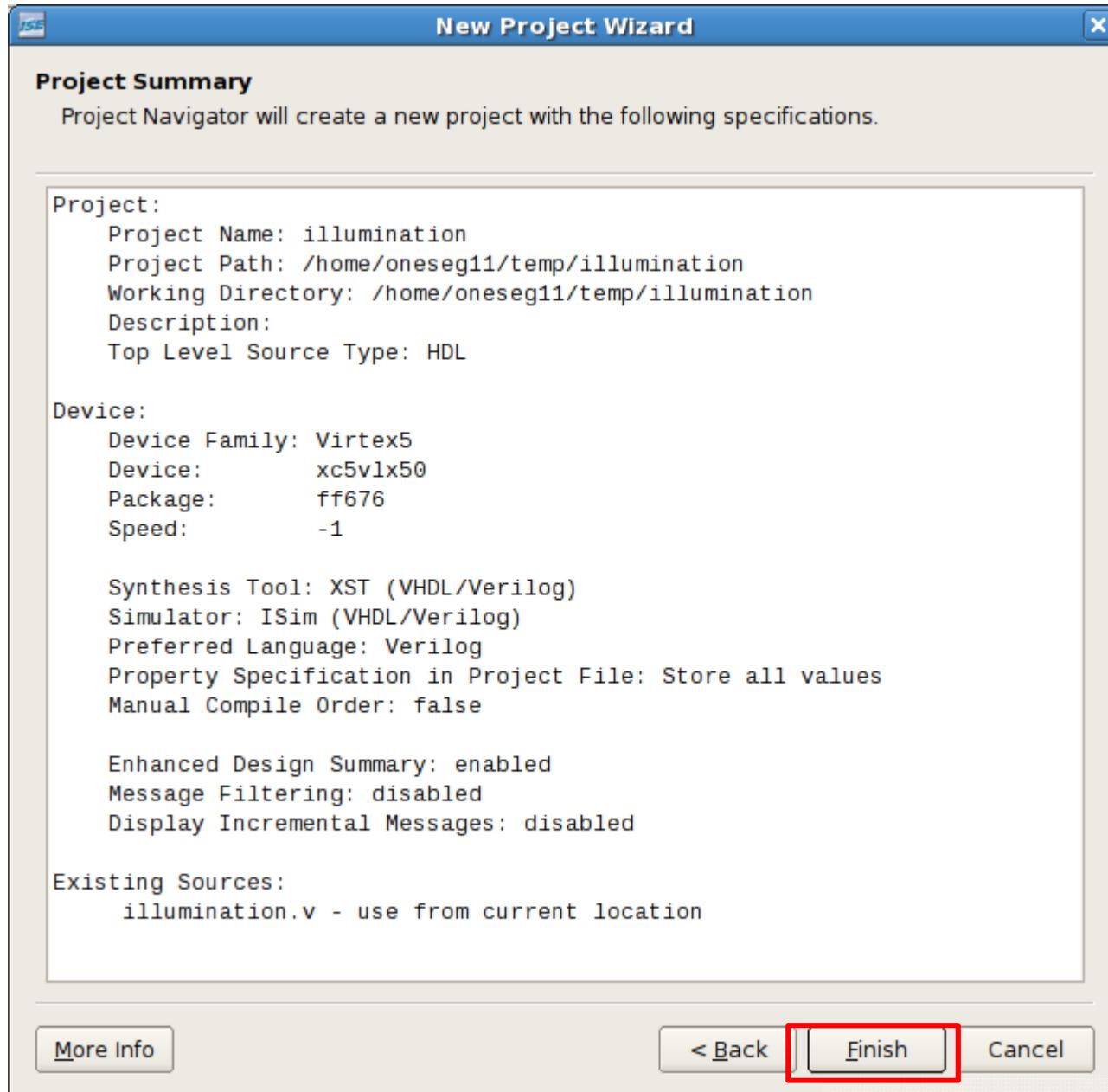


Nextをクリック

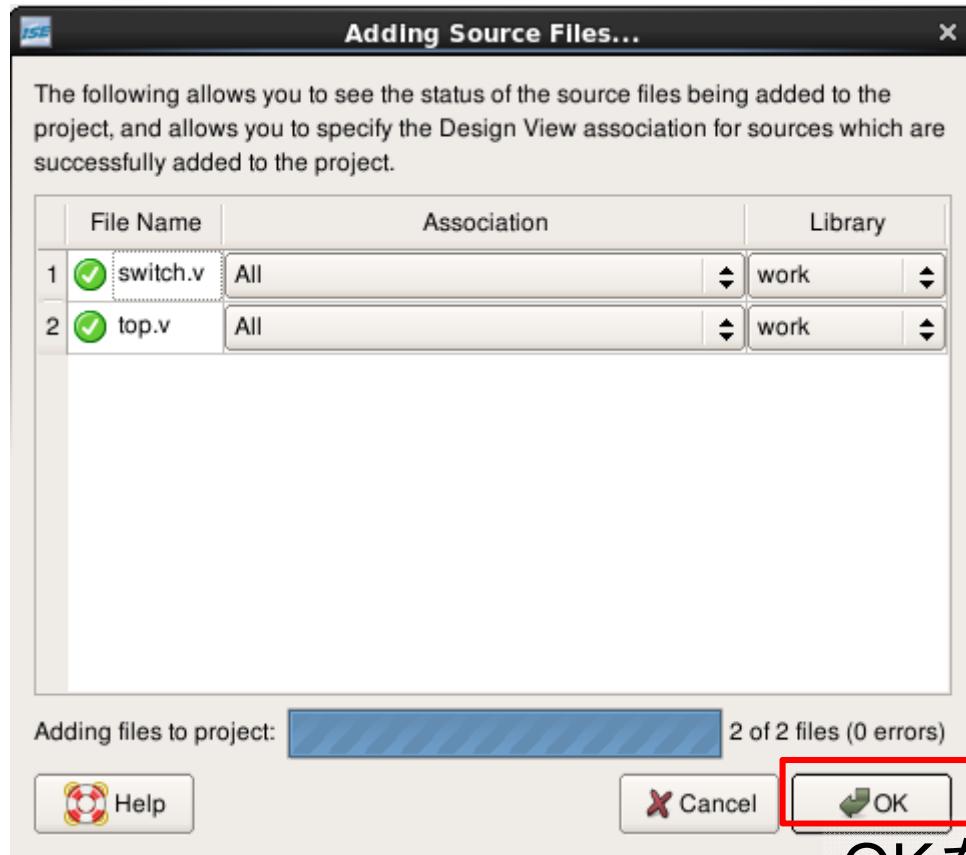


必要なファイルを全て選択  
今回はtop.v, switch.v  
いずれも ~/LED/rtl/  
にある

Nextをクリック

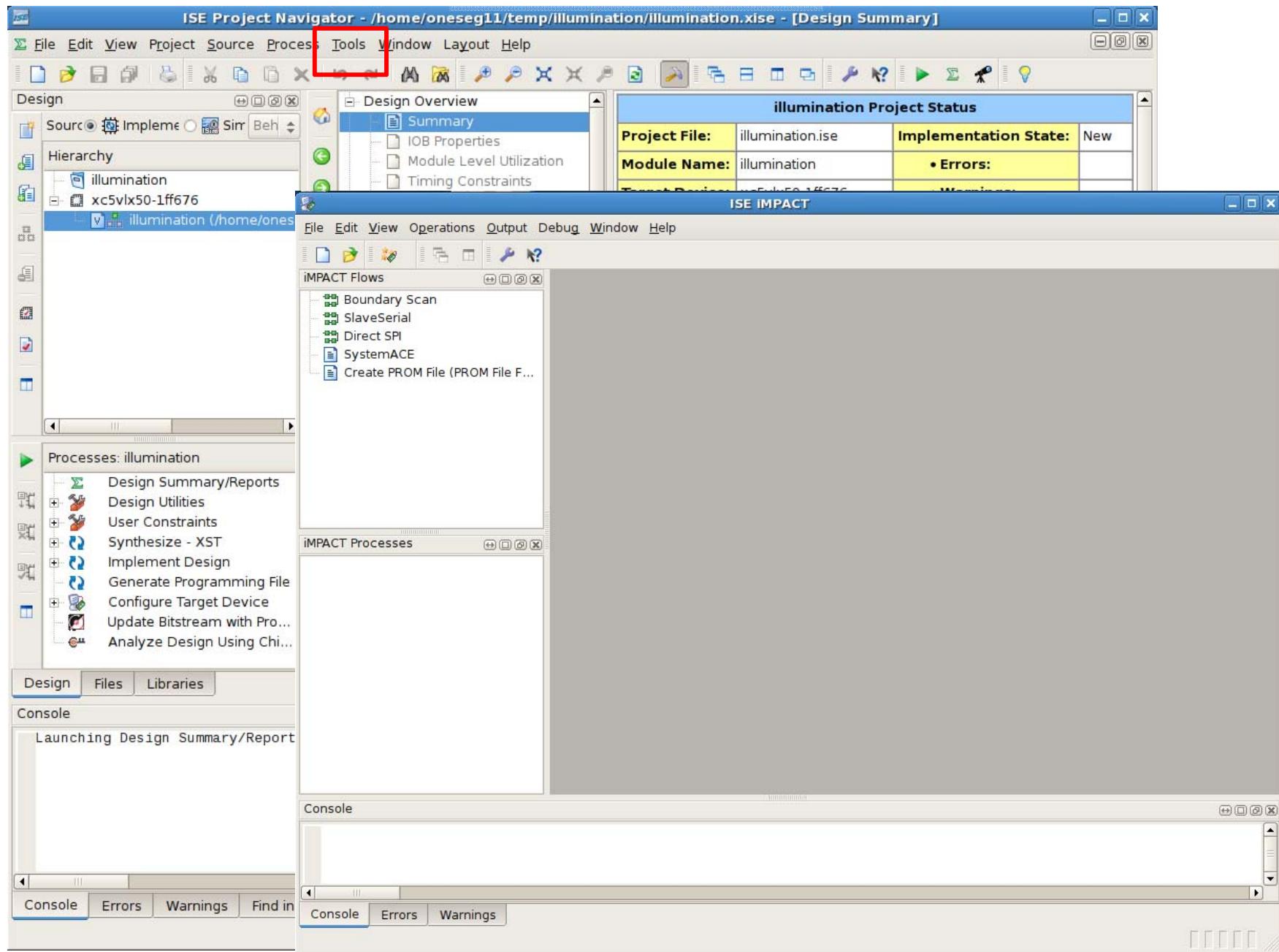


Finishをクリック

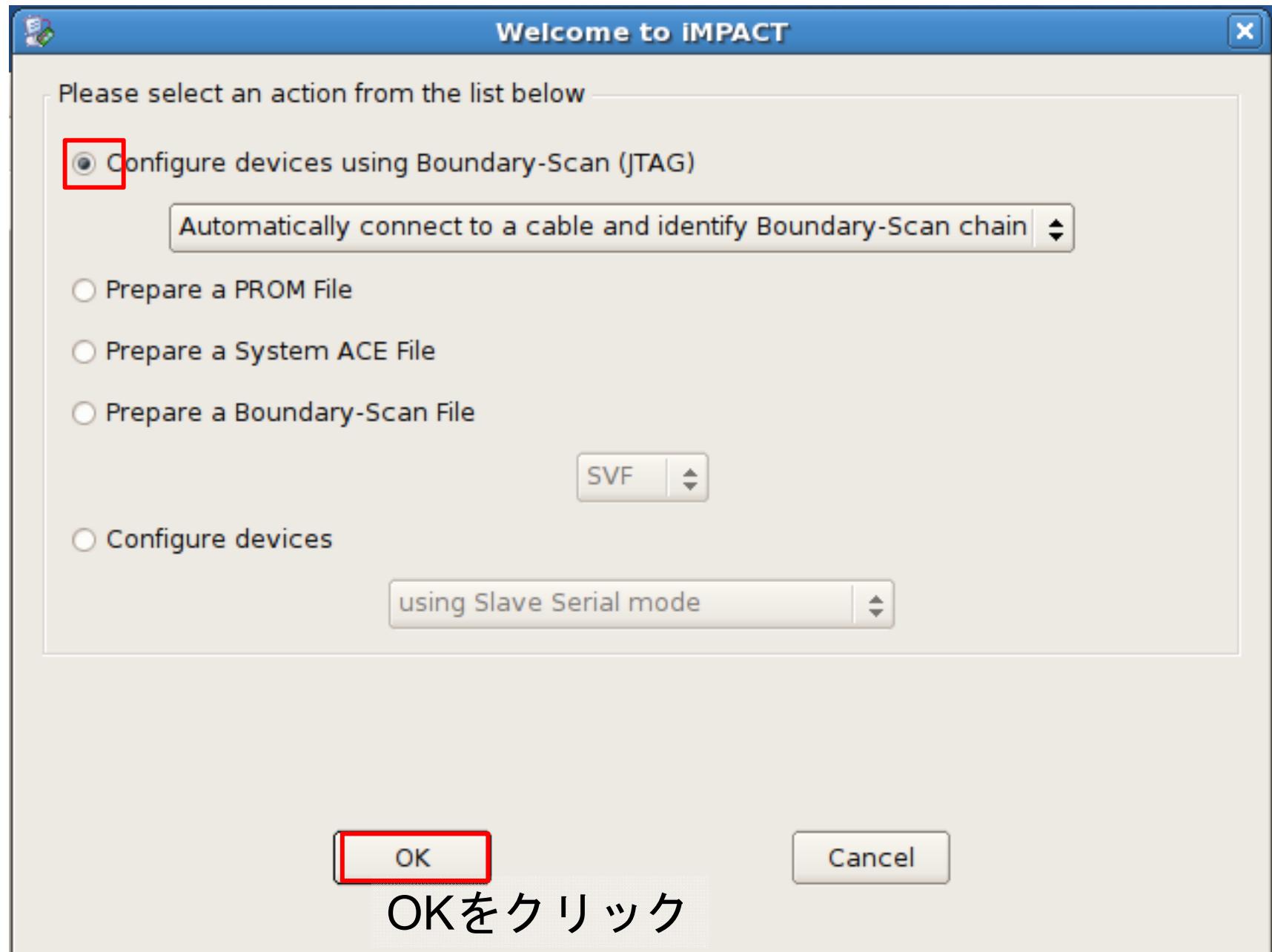


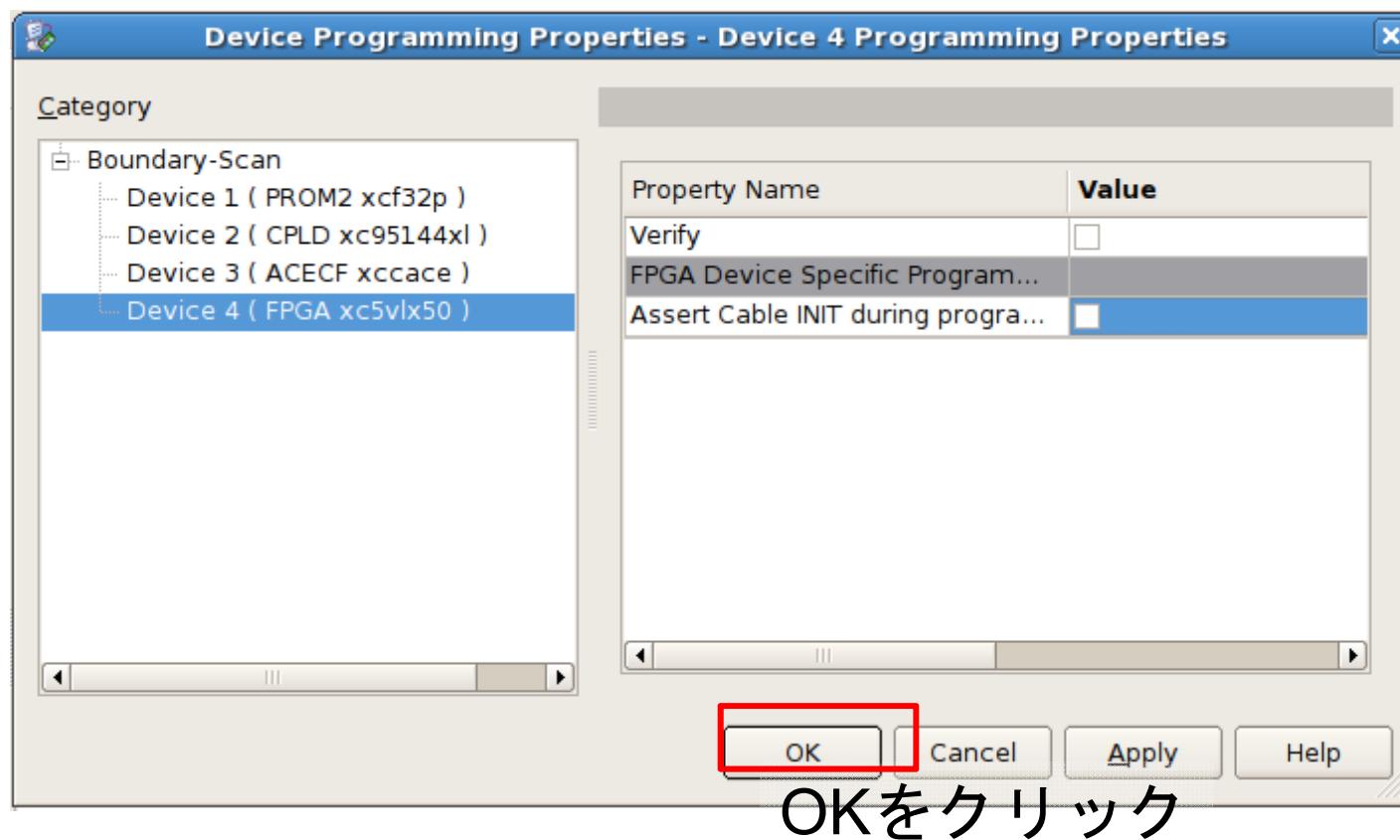
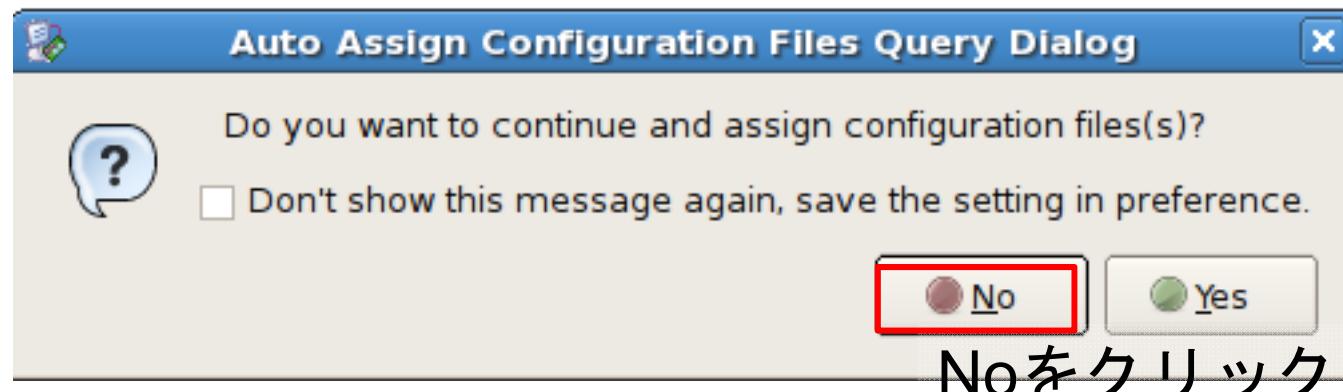
OKをクリック

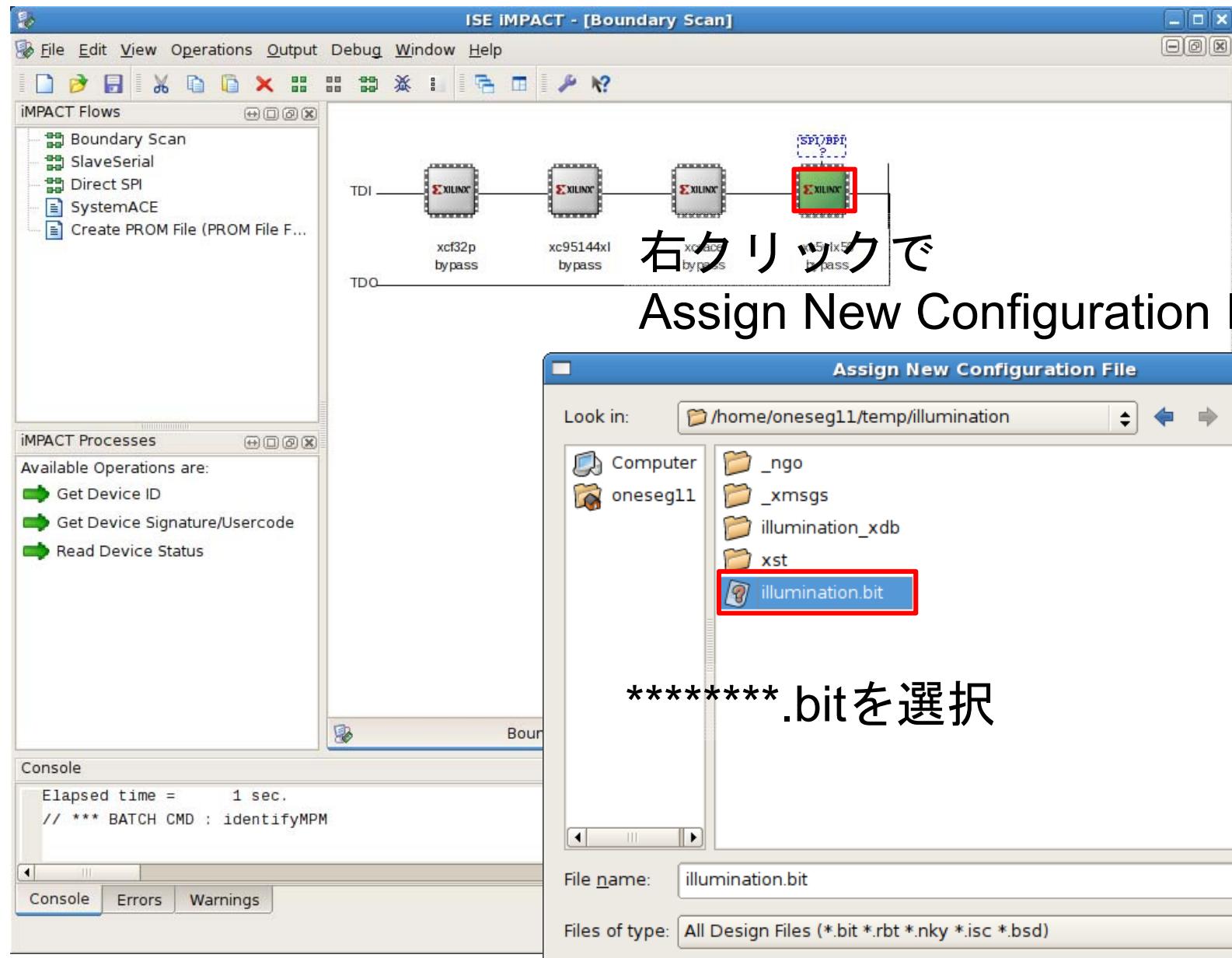
# Tools > iMPACT

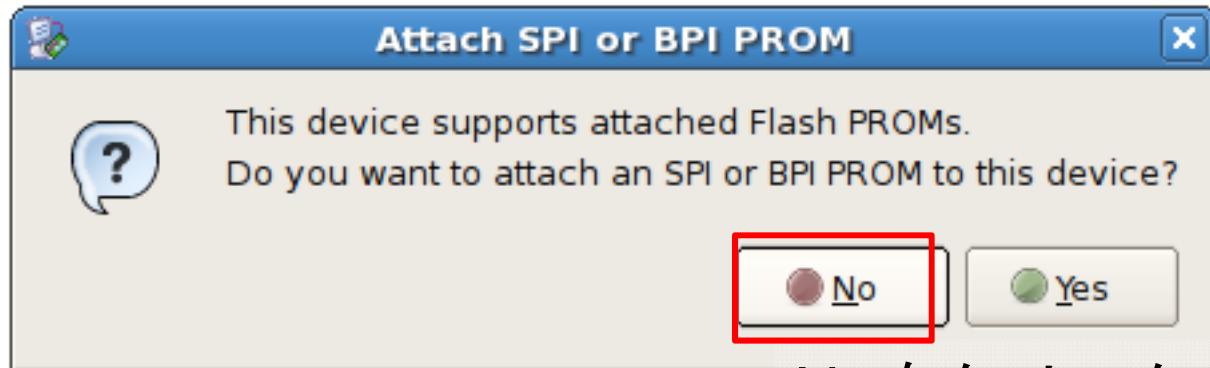


## File > New project





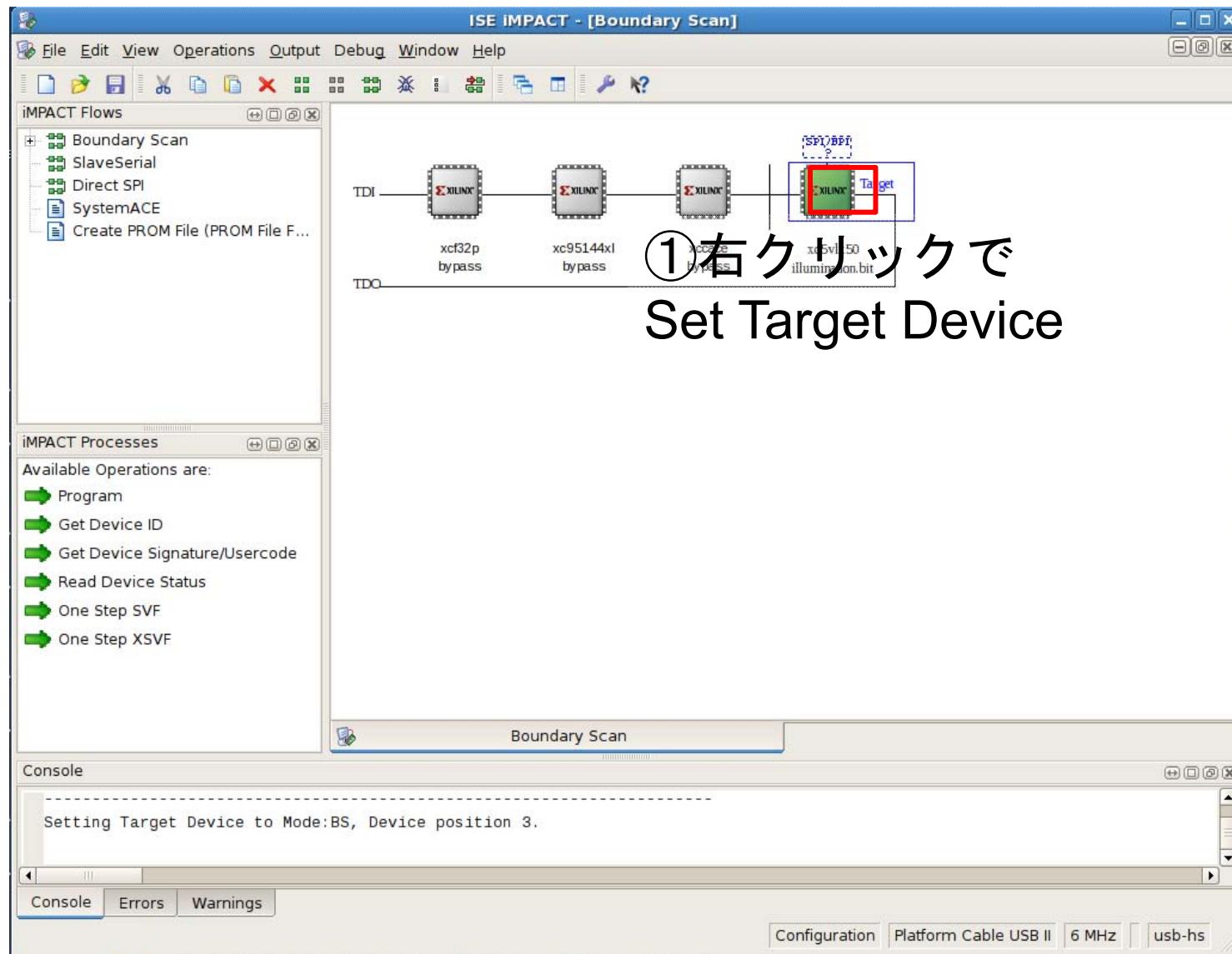




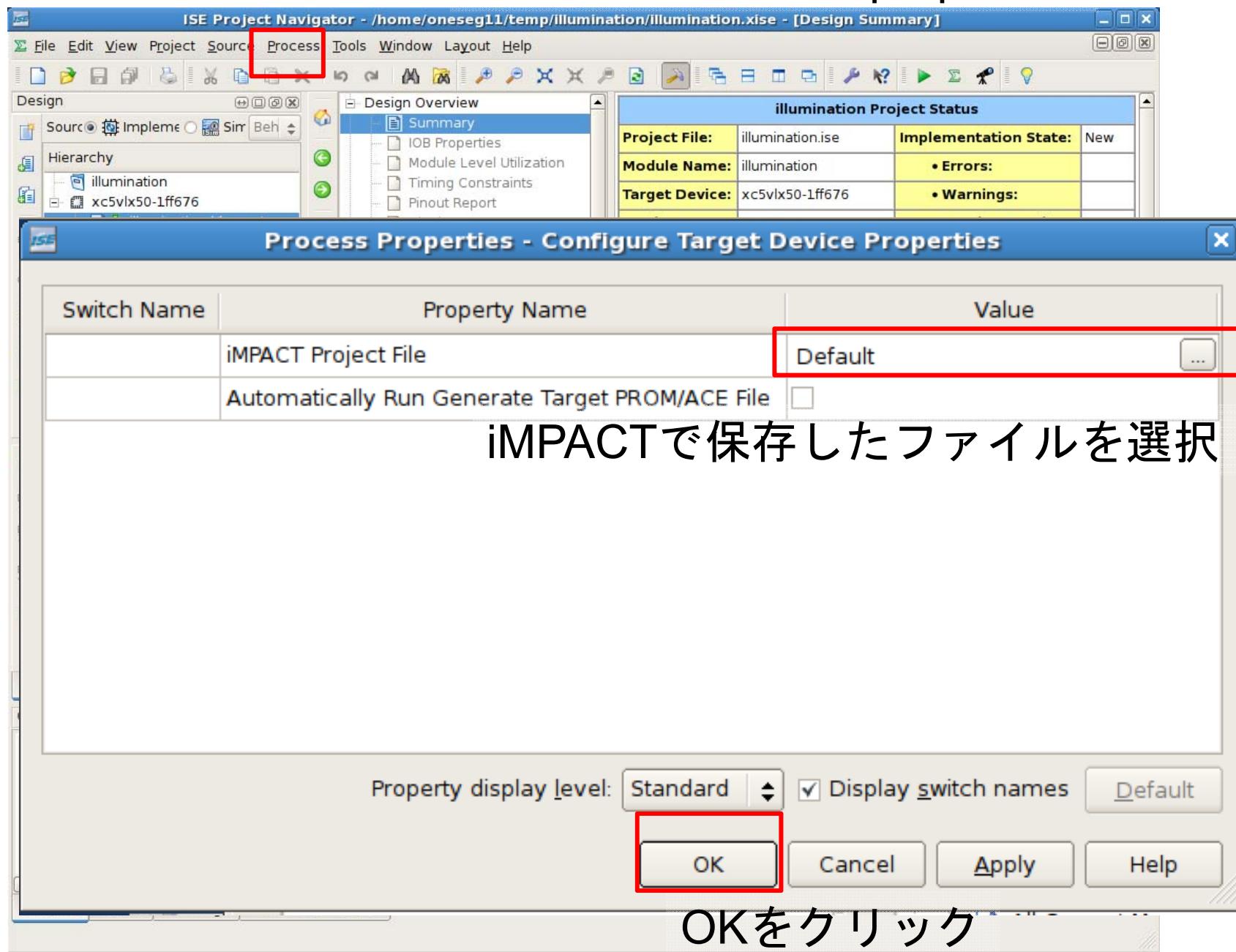
Noをクリック

②File → Save as

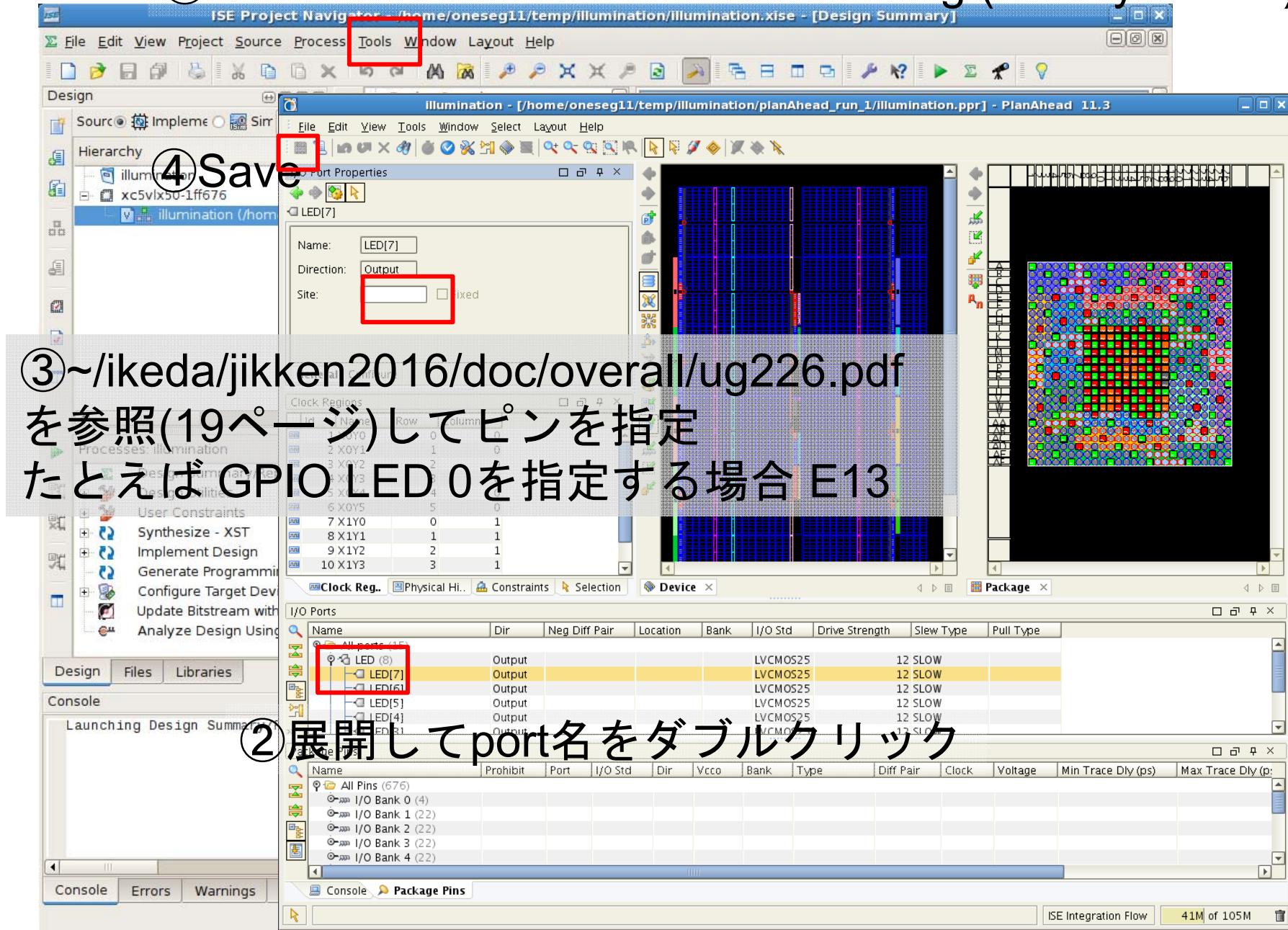
hogehoge.ipfで保存 hogehogeは何でもいいです。



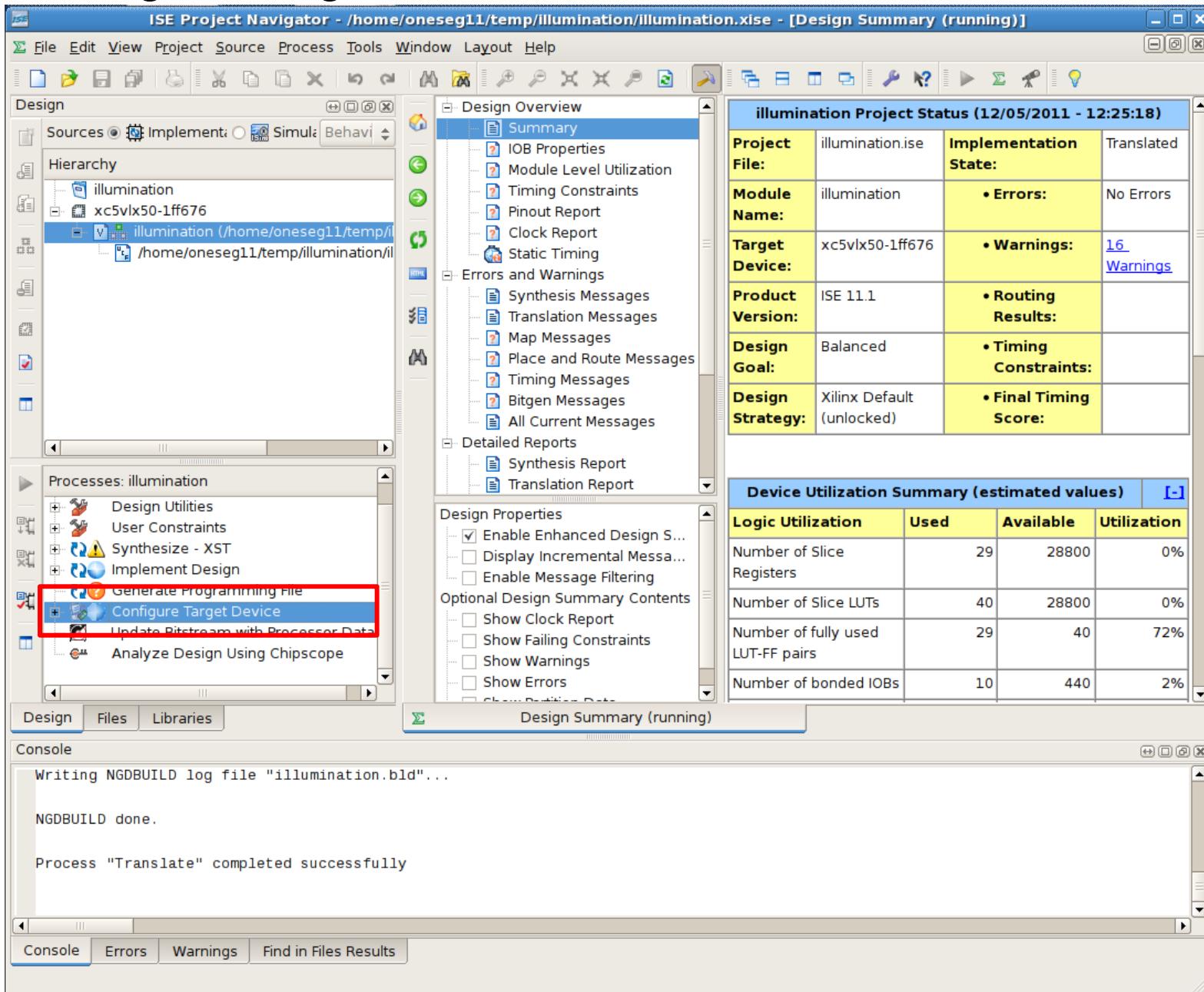
# Process > Process properties



# ①Tools > PlanAhead > IO Pin Planning (Pre-Synthesis)



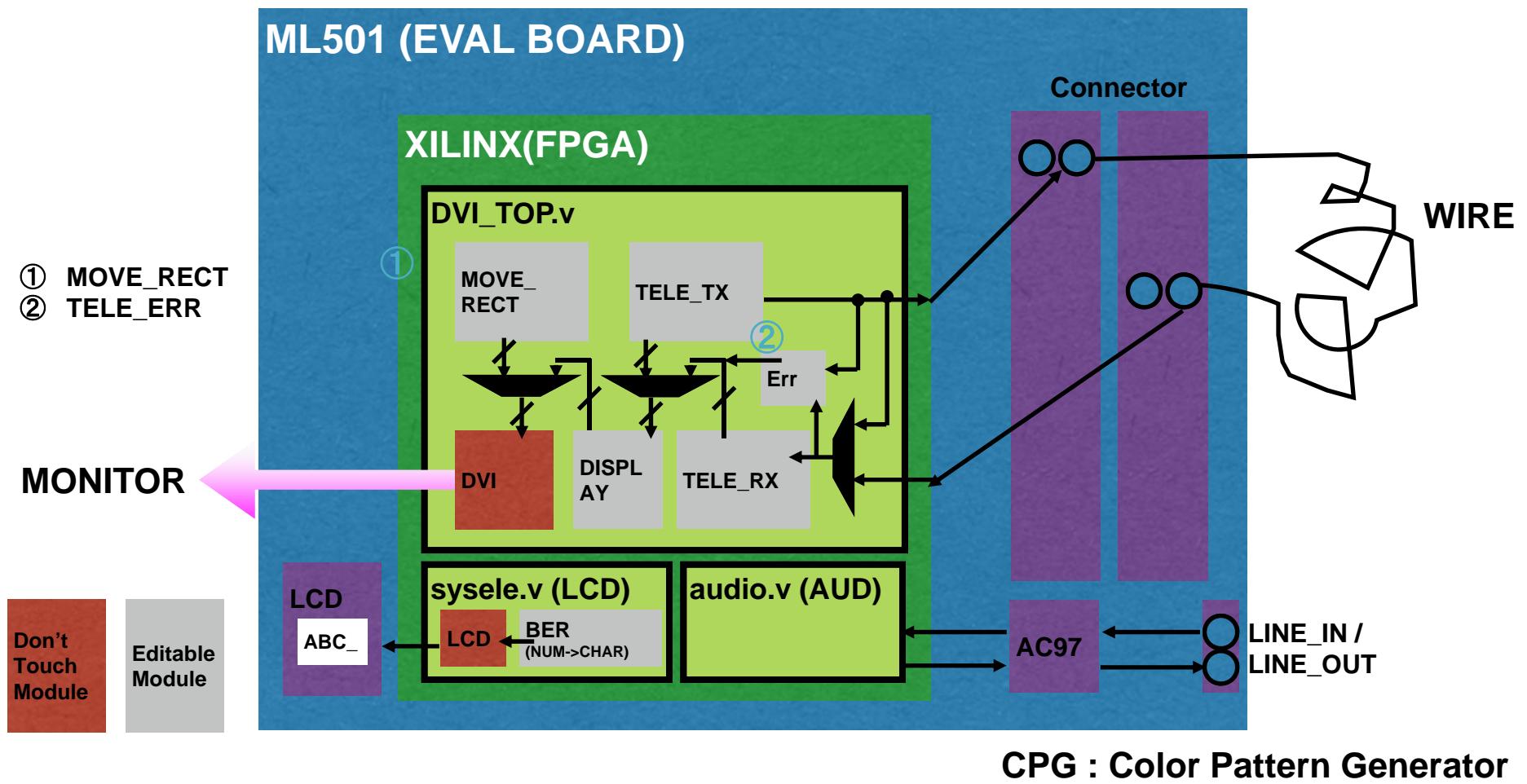
# Configure Target Deviceをダブルクリックで書き込み



# 実験課題2:FPGAボード



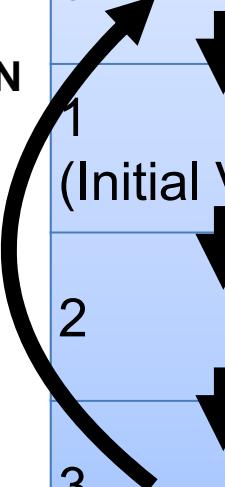
# FPGAボードと構成



# 表示モード

LED		DISPLAY MODE
PUSHBUTTON CENTOR	0	TEST_PATTERN
	1	MOVE_RECT
	2	OUTPUT OF TELE_TX
	3	OUTPUT OF TELE_RX

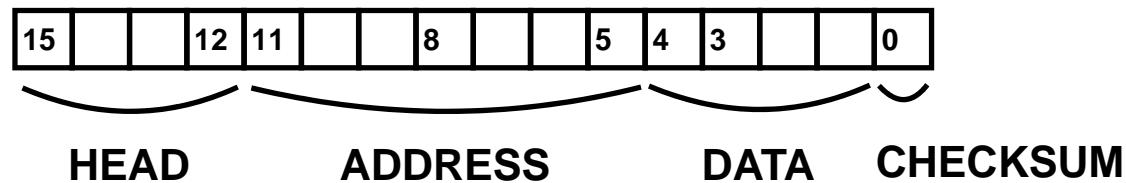
(Initial Value)



WHEN DISP\_MODE==3

PUSHBUTTON NORTH (LED)	Src. of TELE_RX
0	Internal Connection
1	From Outside
PUSHBUTTON SOUTH (LED)	Err. count
0	Normal
1	Err. count 5 digits(HEX)
PUSHBUTTON EAST (LED)	Err. Count update
0	Normal
1	UPDATE ONCE

# **TELE\_TX/RX**



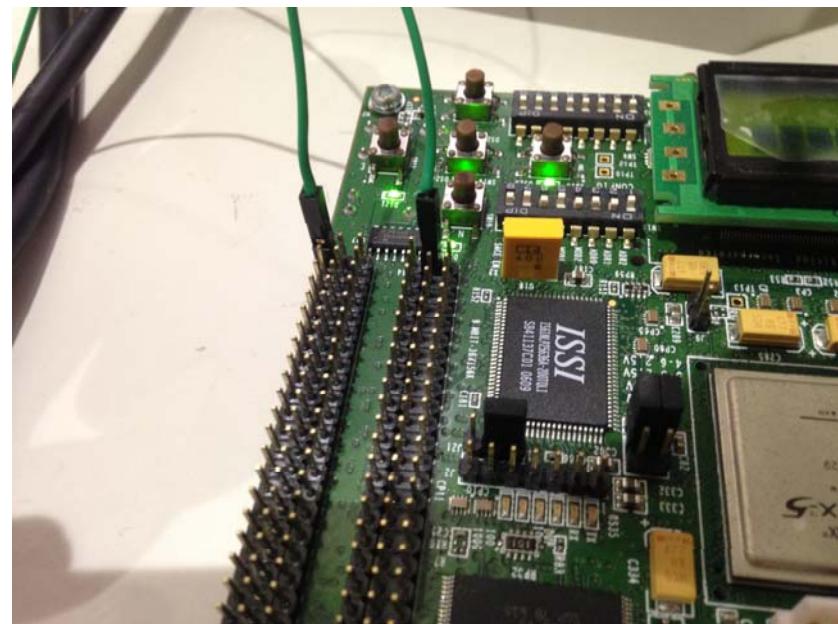
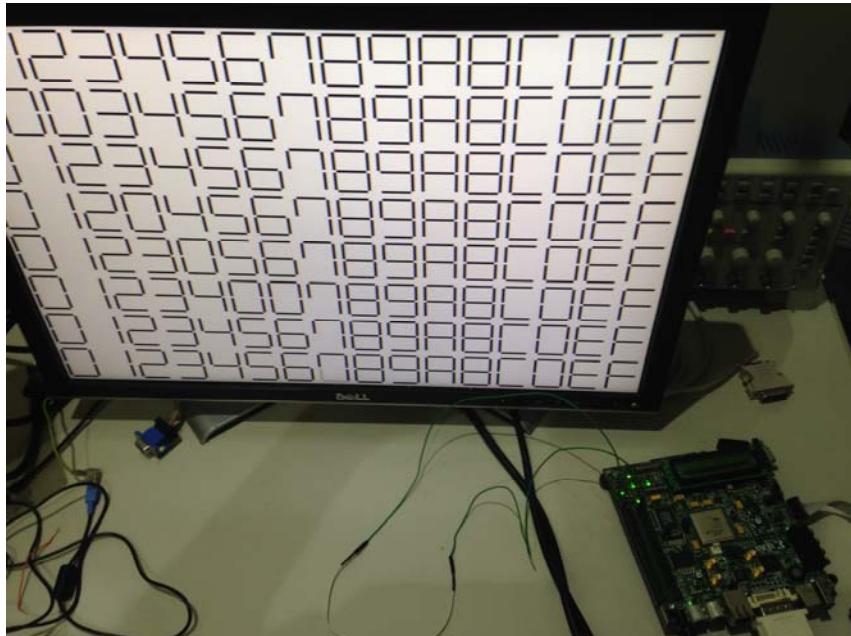
**HEAD : 4'b1010**

**ADDRESS : 7bits (DEC 0 ~ 127)**

**DATA : 4bits (DEC 0 ~ 15)**

**CHECKSUM : ^[15:1]**

# Screenshots



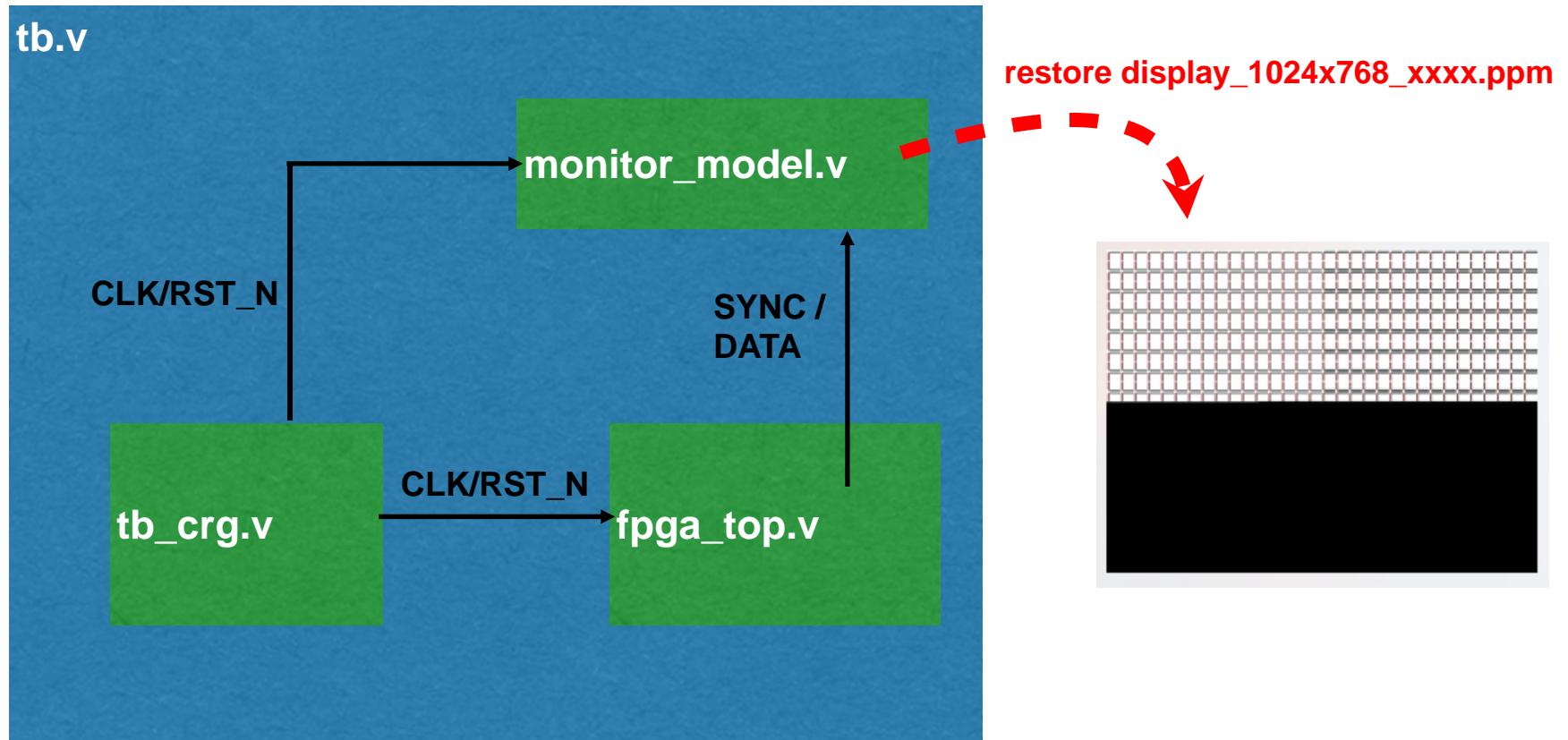
# 実験開始

- cd ~/ISE/
- run ISE -> Open Project -> ./ISE/ISE.xise

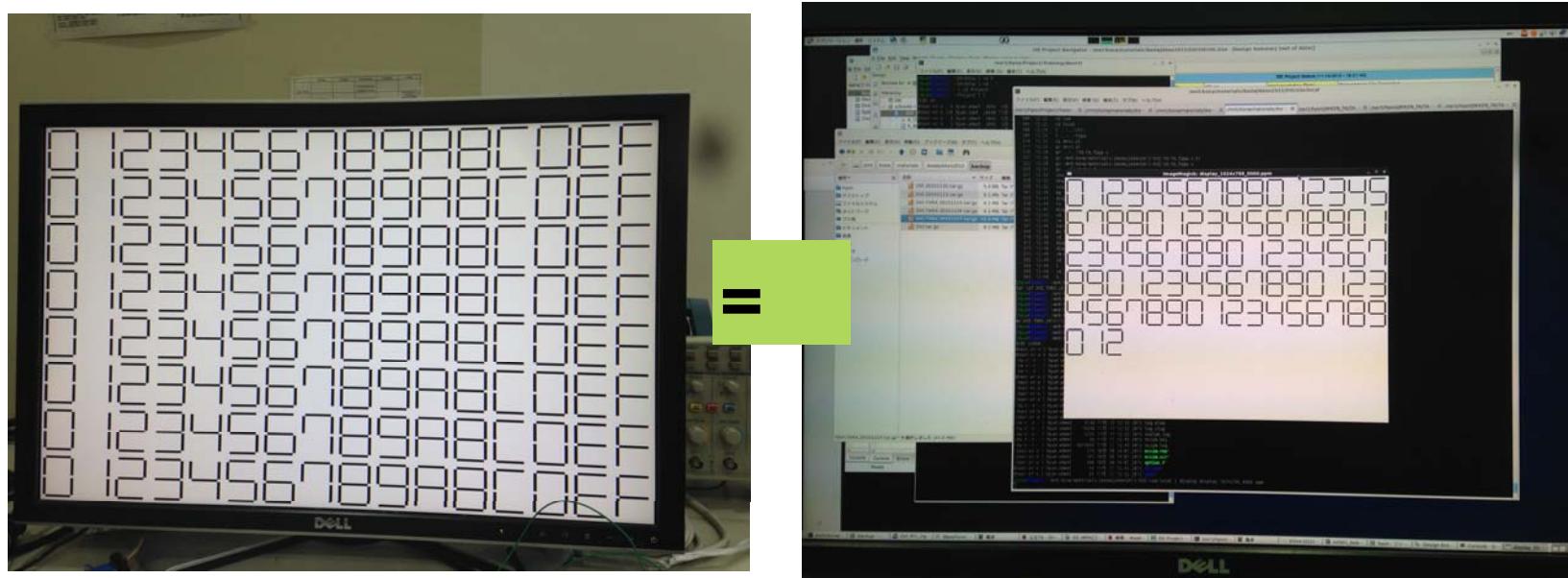
# ディレクトリ構成

- ikedajikken2016
  - doc : Documents
  - fpga : FPGA Top RTL, UCF(BALL MAP)
  - include : Include files
  - ISE : FPGA Work Directory
  - rtl : RTL Files
  - tb : testbench
  - sim : scripts for local simulation
  - fpga

# RTLの構成概要



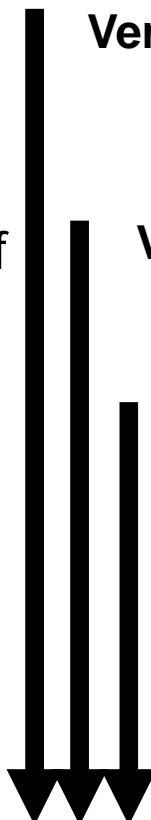
# シミュレーションでの表示の確認



◆ > **display display\_1024x768\_\*\*\*\*.ppm ( at ./sim/local )**

# シミュレーションの実行

- ◆ cd jikken2016
- ◆ FILE LIST : vi ./sim/list/top.f
- ◆ Local Sim. : cd ./sim/local/
- ◆ Compile : source compile dev top.f
- ◆ Elaborate : source elaborate tb
- ◆ simulate : source ncsim.scr



Verilogファイルを追加削除した場合

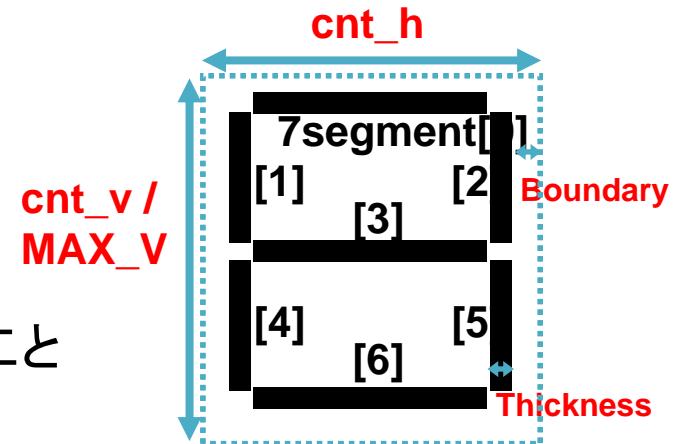
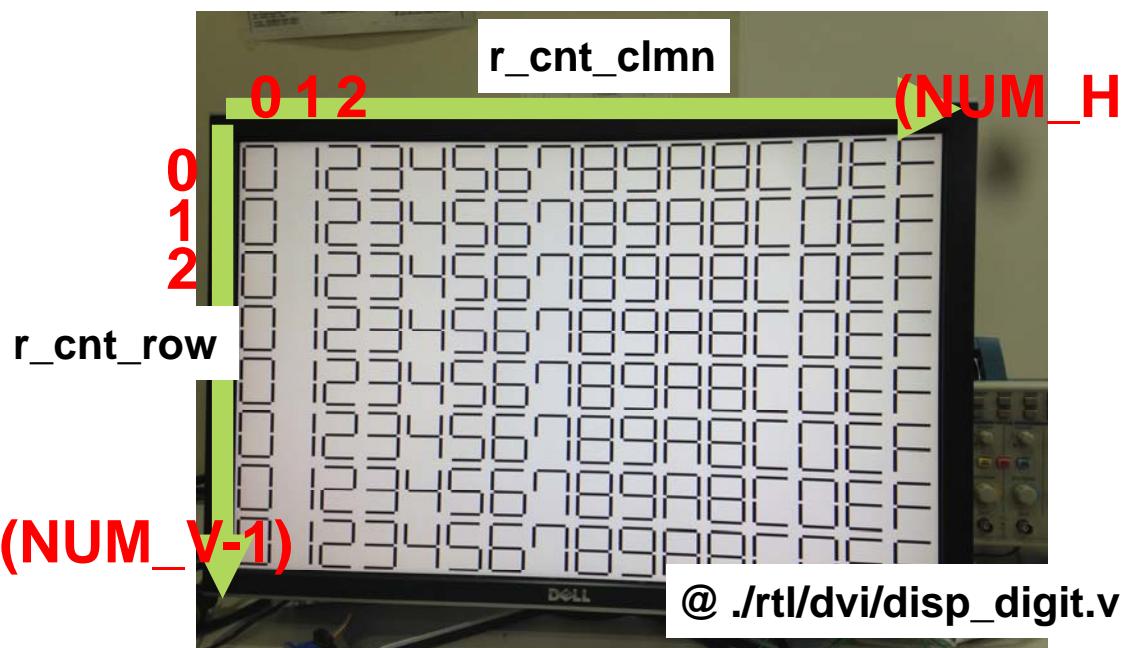
Verilogファイルを修正した場合 (全実行)

Verilogを微修正した場合 (simple)  
: SIMVISION -> SIMULATION -> REINVOKE  
ただし、この場合verilogプロセスが  
残ってしまうので適宜killするなり注  
意が必要

# 実験課題..2－1

- DVIインターフェースでディスプレイに表示
  1. サンプルをもとに画面表示を実現
  2. 数字(フォント)サイズを変更
  3. 画面に表示するキャラクタ数を変更
  4. フォントをきれいなものに変更

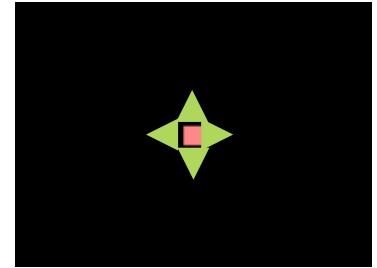
1を実施後、2－4のいずれかを選択して実施すること



@ ./rtl/dvi/disp\_digit\_seg.v

- 1) digit to 7seg
- 2) Seg Style : size , thickness, ..

# 実験課題..2－2



- 表示されたものをボタンを使って動かす
  - サンプルをもとに画面表示の移動を実現
  - button 操作による反応を変えてみる(以下は例)
    - 四角形の色を変えてみる。
    - 四角形を変更してみる(円形、三角形など)
    - 複数の図形を出力してみる。

1を実施後、2に関して最低一例実現

@ ./rtl/dvi/move\_rect.v  
@ ./rtl/dvi/draw\_rect.v

色指定

```
// Rectangle Color
parameter      RECT_COLOR_RED      = 8'd255;
parameter      RECT_COLOR_GRN     = 8'd128;
parameter      RECT_COLOR_BLU      = 8'd128;

o_sync_red  <= (area) ? RECT_COLOR_RED : BG_COLOR_RED ;
o_sync_grn  <= (area) ? RECT_COLOR_GRN : BG_COLOR_GRN ;
o_sync_blu  <= (area) ? RECT_COLOR_BLU : BG_COLOR_BLU ;
```

四画(描画領域判定)

```
always @ (*) begin
    if ( (r_cnt_y >= (r_pos_y-(RECT_H>>1))) &
        (r_cnt_y <= (r_pos_y+(RECT_H>>1))) ) begin
        if ( (r_cnt_x >= (r_pos_x-(RECT_W>>1))) &
            (r_cnt_x <= (r_pos_x+(RECT_W>>1))) ) begin
            area <= 1;
        end else begin
            area <= 0;
        end
    end else begin
        area <= 0;
    end
end
```

ボタンによる動作指定

```
always @ (posedge clk or negedge rst_n) begin
    end else begin
        if (i_pls_e) begin
            .....
        end else if (i_pls_w) begin
            .....
        end
    end
end
```

# 実験課題..2－3

- シリアル通信を実現し通信速度、通信路とエラーレートの関係を求める
  - 通信速度を変化させる
  - ケーブル長を変化させる
  - ケーブル中に抵抗などロスを挿入させる

1を実施後、2－4のいずれかを選択して実施すること

通信速度の変更

@ ./rtl/dvi/DVI\_TOP.v

通信 速度	エラーレート				
	ケーブル長			抵抗挿入	
	短(**cm)	中(**cm)	長(**cm)	1(**Ω)	2(**Ω)
Clk_1					
Clk_2					
Clk_4					
Clk_8					
Clk_16					

これをグラフ化する

上のtele\_clk\_1 – 16までを選択

```
tele_crg A_TELE_CRG (  
.src_clk      (Clock      ),  
.rst_src_n    (Reset      ),  
.src_clk_1    (tele_clk_1 ),  
.src_clk_2    (tele_clk_2 ),  
.src_clk_4    (tele_clk_4 ),  
.src_clk_8    (tele_clk_8 ),  
.src_clk_16   (tele_clk_16 )  
);
```

```
tele_tx A_TELE_TX(  
.clk          ( tele_clk_1 ),//i  
.....
```

```
tele_rx A_TELE_RX(  
.clk          ( tele_clk_1 ),//i  
.....
```