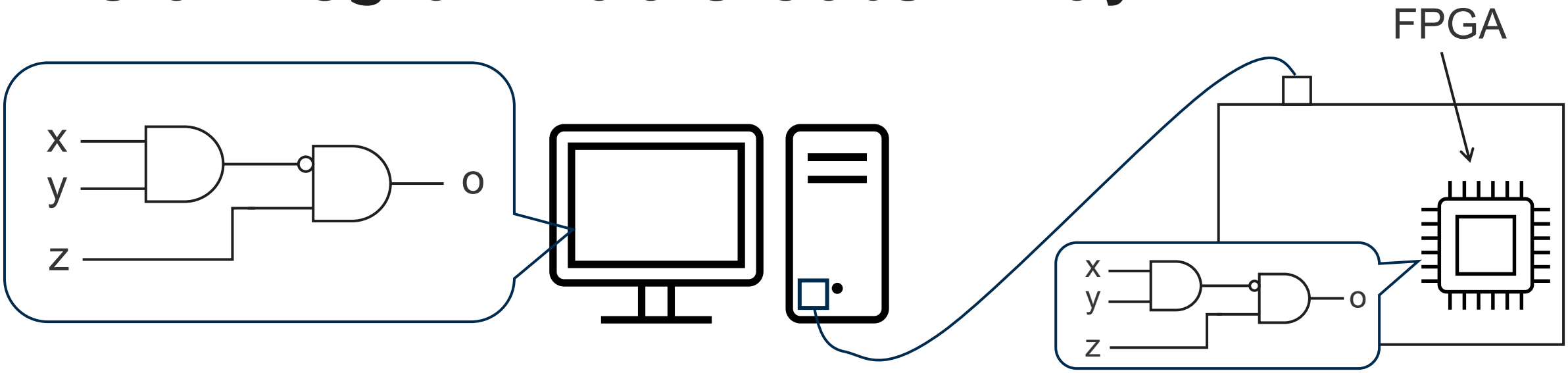
Verilogの基礎その1

ISHI会
フルデジタルで音声を操るためのロジック回路 1回目

目次

- FPGAについて
- ボードの準備
- スイッチとLEDを使った回路を通したEfinix FPGAの開発方法
- 基本ゲートのVerilog記述
- 今日の練習問題

Field Programmable Gate Array

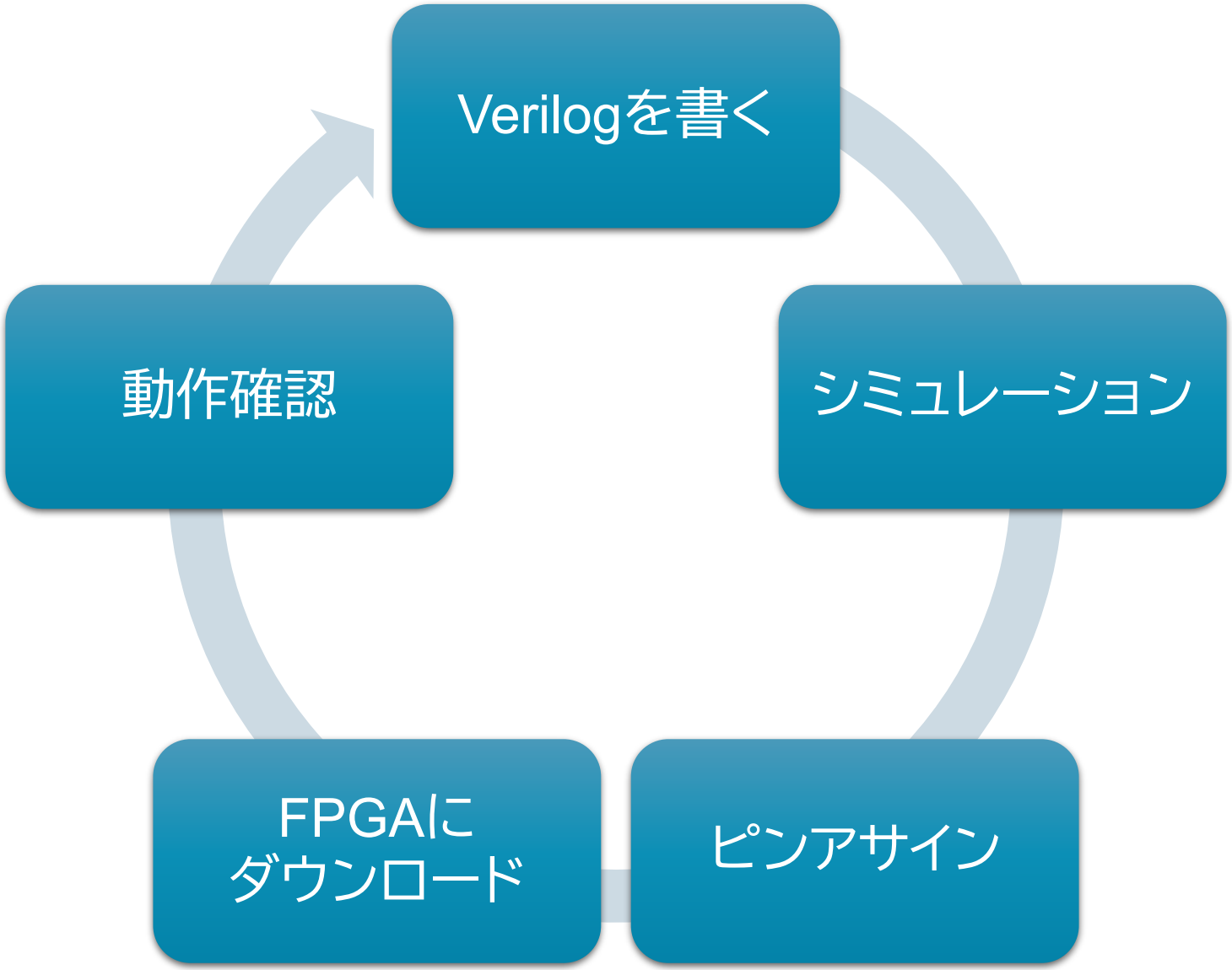


Field Programmable Gate Array(FPGA) はコンピュータでロジック回路を記述して、自由に回路を書き換えることができる**集積回路(IC)**。

Field (現場)でProgrammable(書き込める)特性を有している。

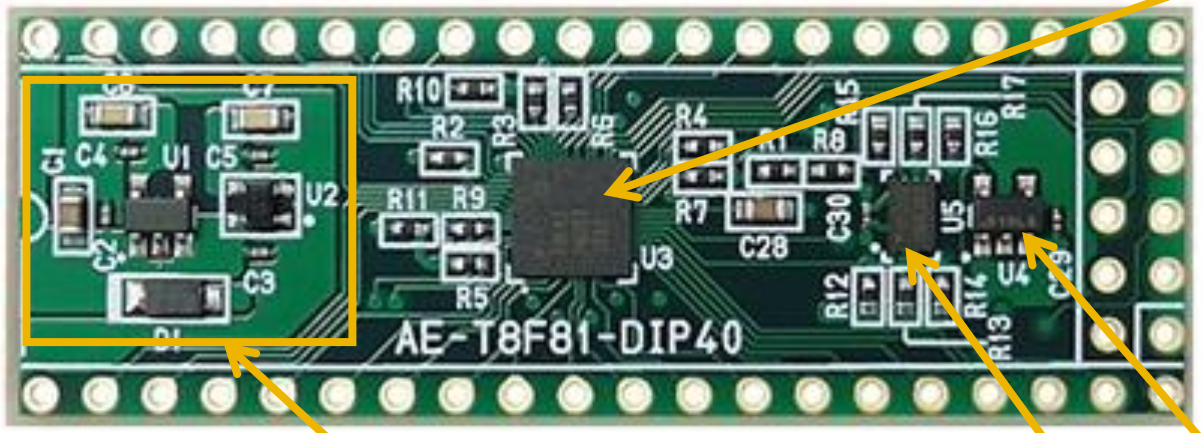
ICを作るより安価に自由なロジック回路を作り込めるだけでなく、何回でも回路を書き換えることができることから、頻繁に回路の変更が生じる試作や研究開発に広く使われている。

FPGAの開発手順



FPGAボードの準備

今回用いるFPGA



Efinix TRION T8F81C2X

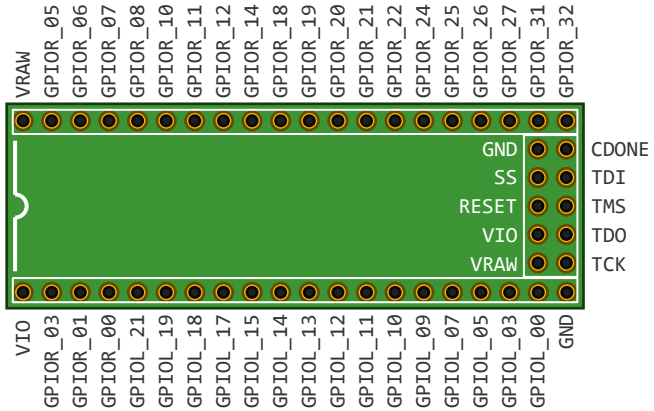
コア電圧：	1.1V
IO電圧：	3.3V
PLL数：	1
LE数：	7384
組み込みメモリ：	122.88kbits
GPIO：	55ピン

ローエンドFPGA

電源関係

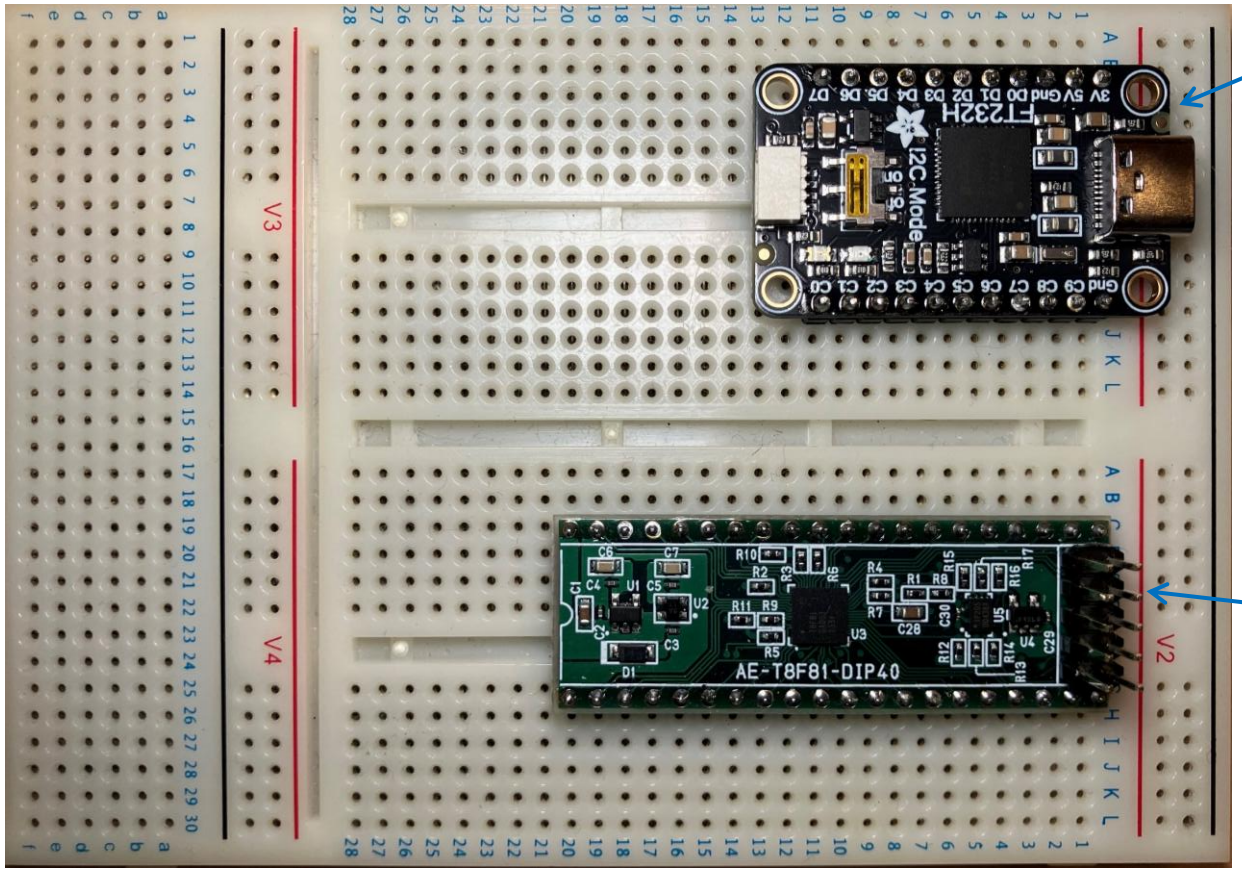
25MHzオンボードクロック

デザイン保存用ROM



FPGAボード開発ボード

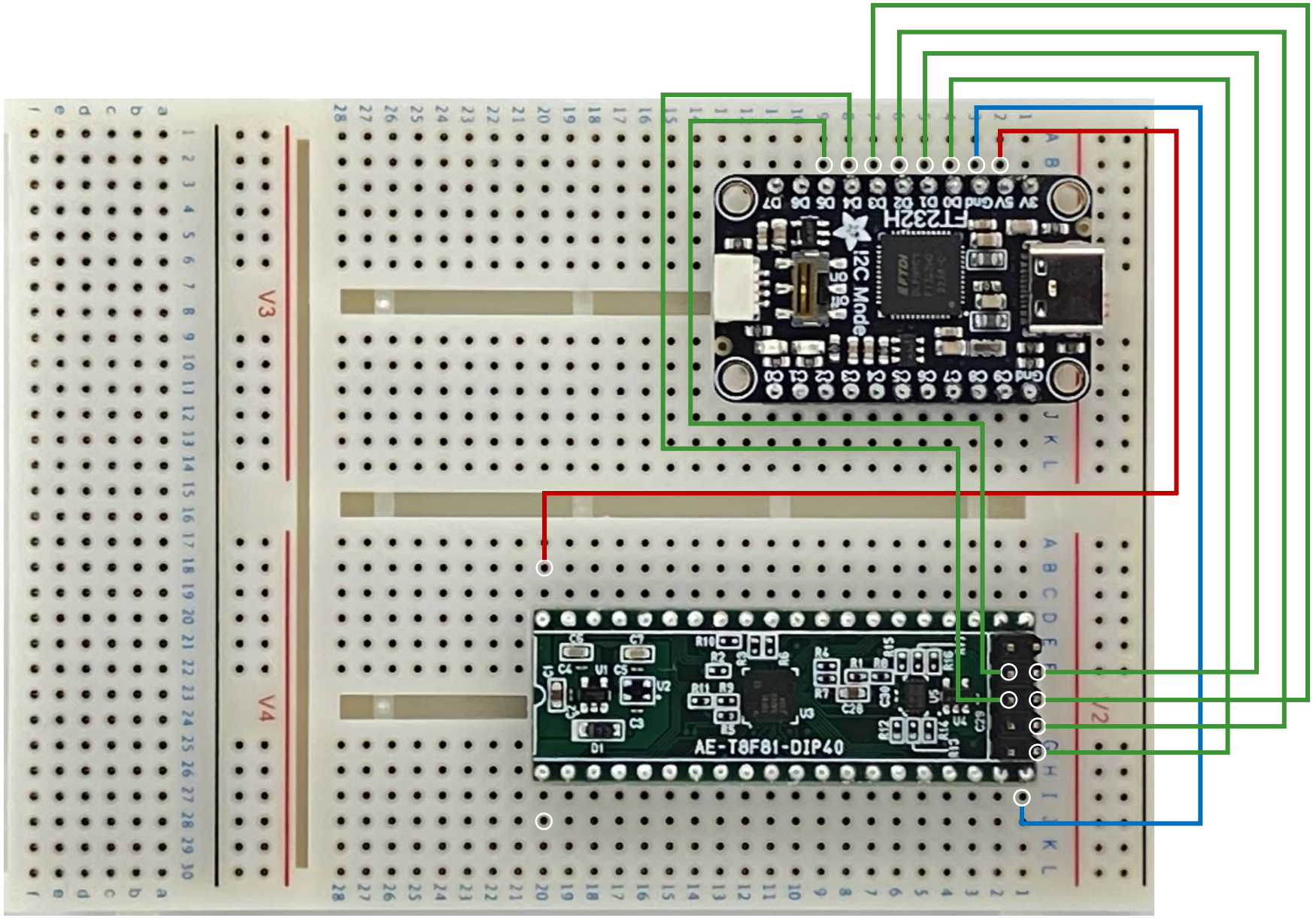
2つの基板が搭載されたブレッドボードをお渡しします。
一方がJTAG書き込み器で、他方がFPGAボードです。



書き込み器(JTAG)

FPGAボード

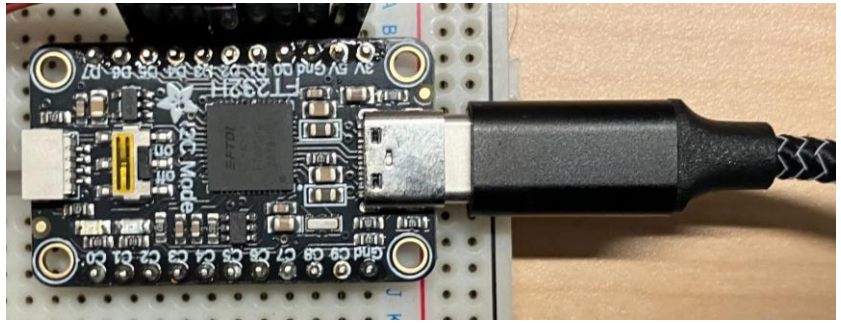
書き込み器とFPGAの接続



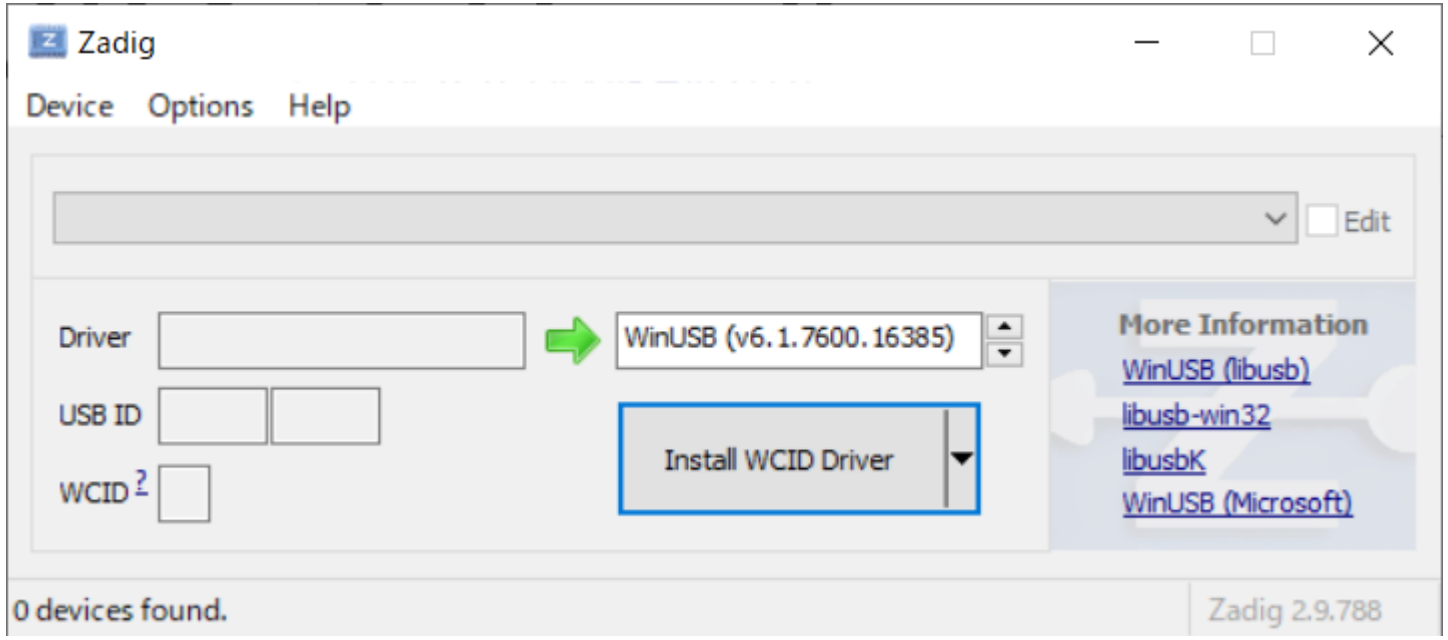
5V電源
GND
信号線(JTAG)

ドライバのインストール

書き込み器とPCをUSBケーブルで接続する

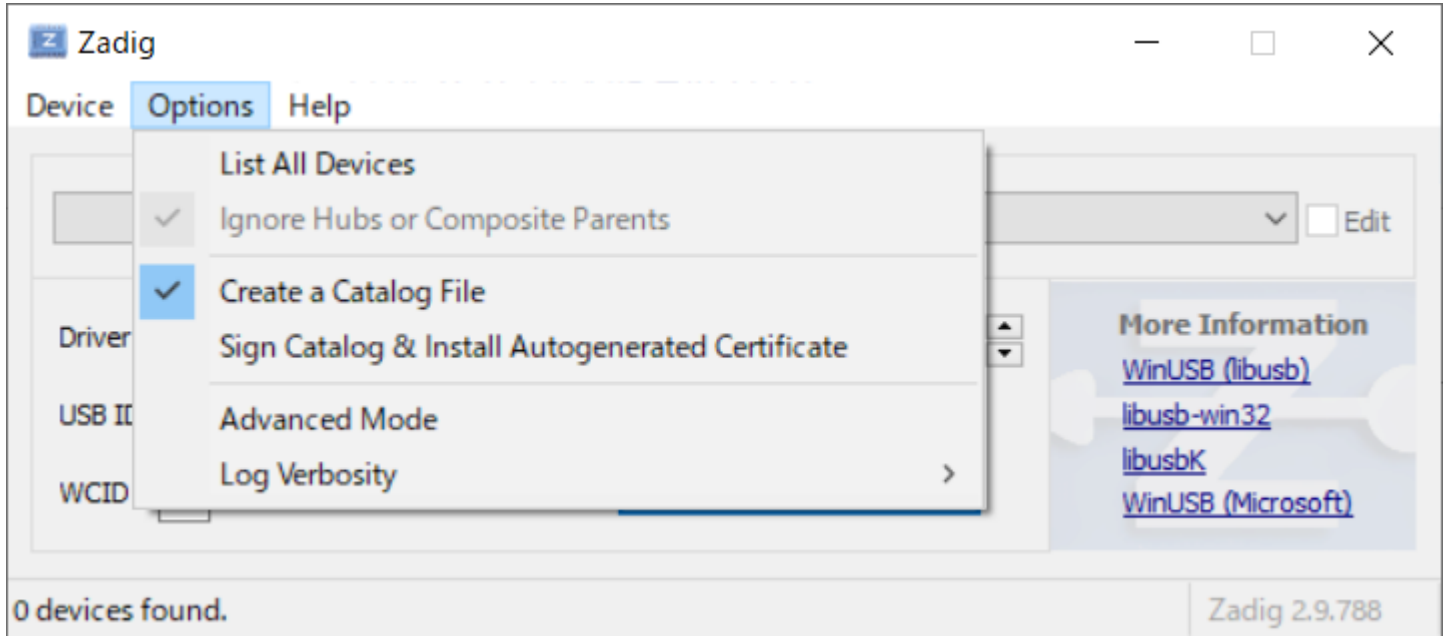


事前準備でダウンロードしたZadigを開く



Zadigの操作1

Options→List All Devicesをクリックする

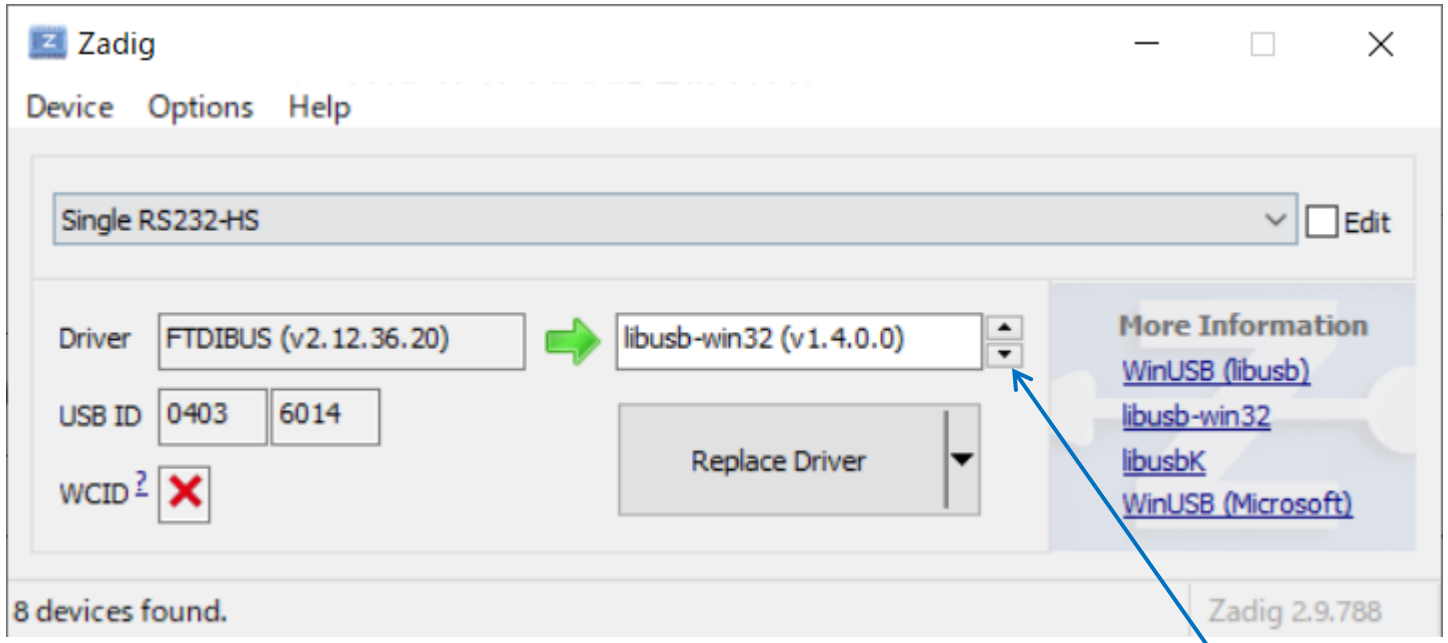


Zadigの操作2

リストから「Single RS232-HS」を選択する。

Zadigの操作3

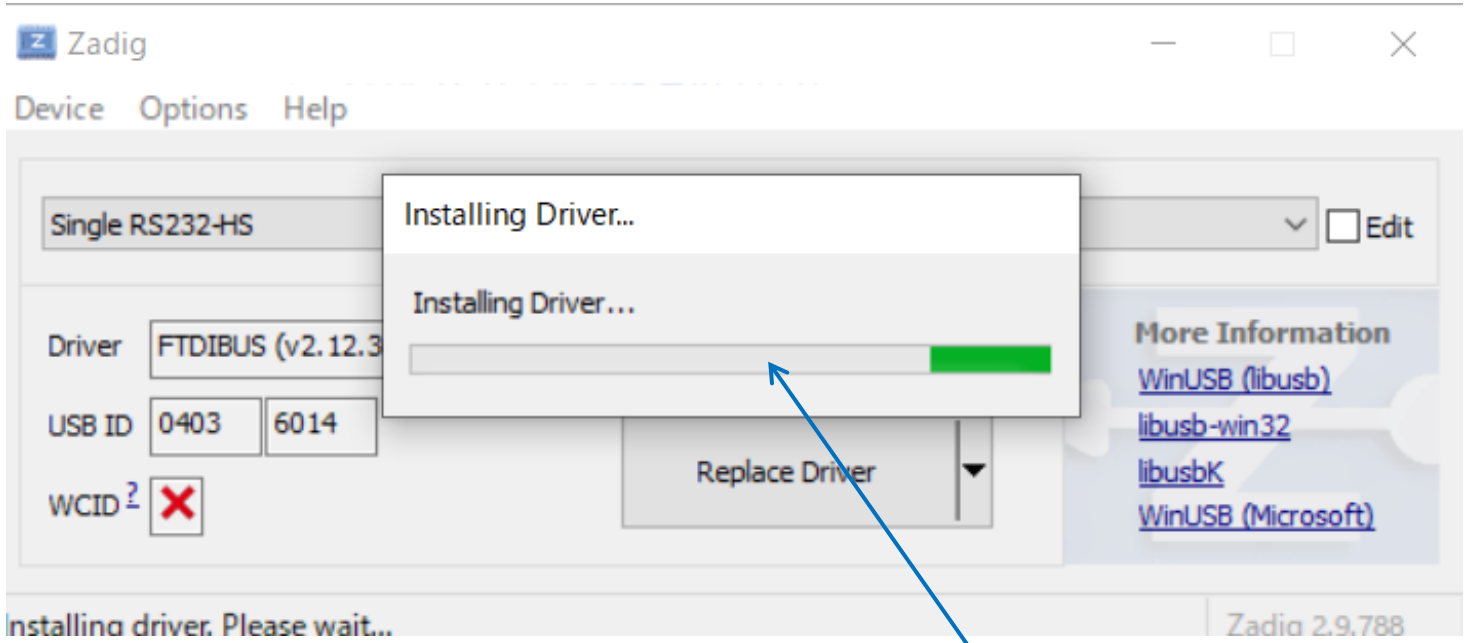
入れ替えるドライバーを「libusb-win32(v1.4.0.0)」を選択する。



矢印でドライバーを入れ替える。

Zadigの操作4

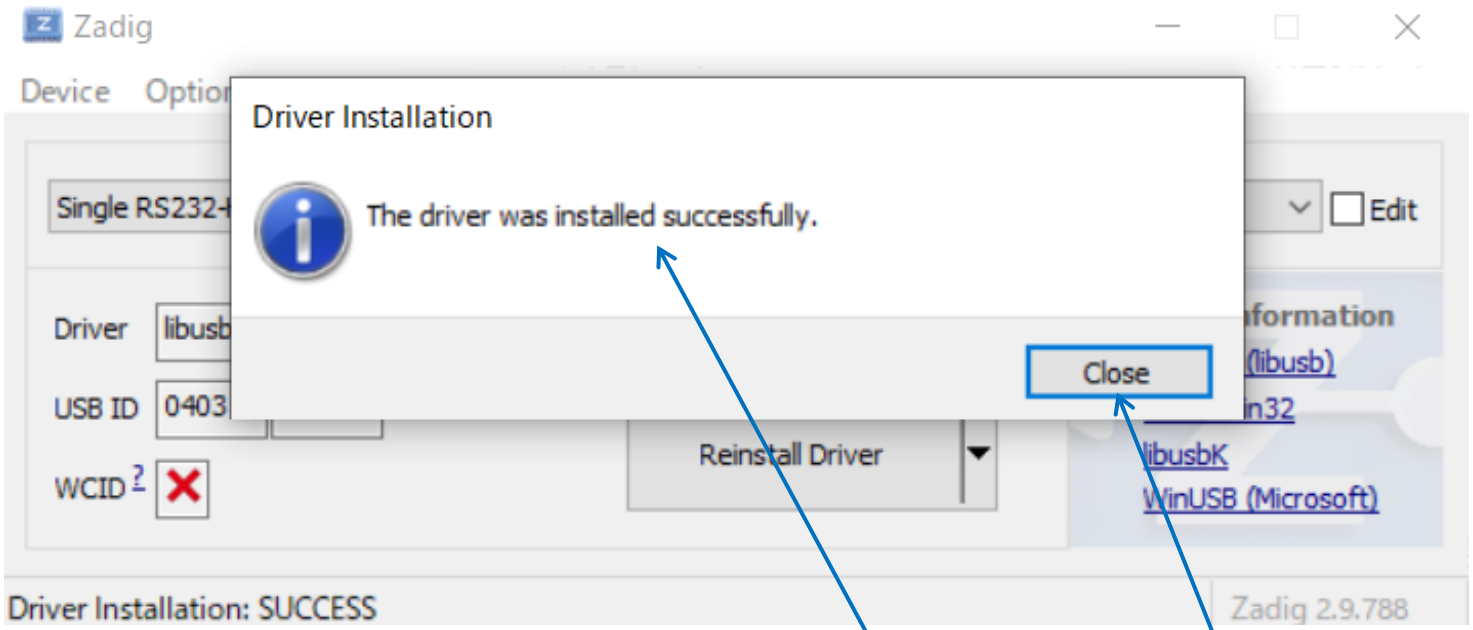
ドライバー入れ替え中



インストールが終わるまで待つ

Zadigの操作5

The driver was installed successfullyとなったらOK
右上Xボタンで終了する。

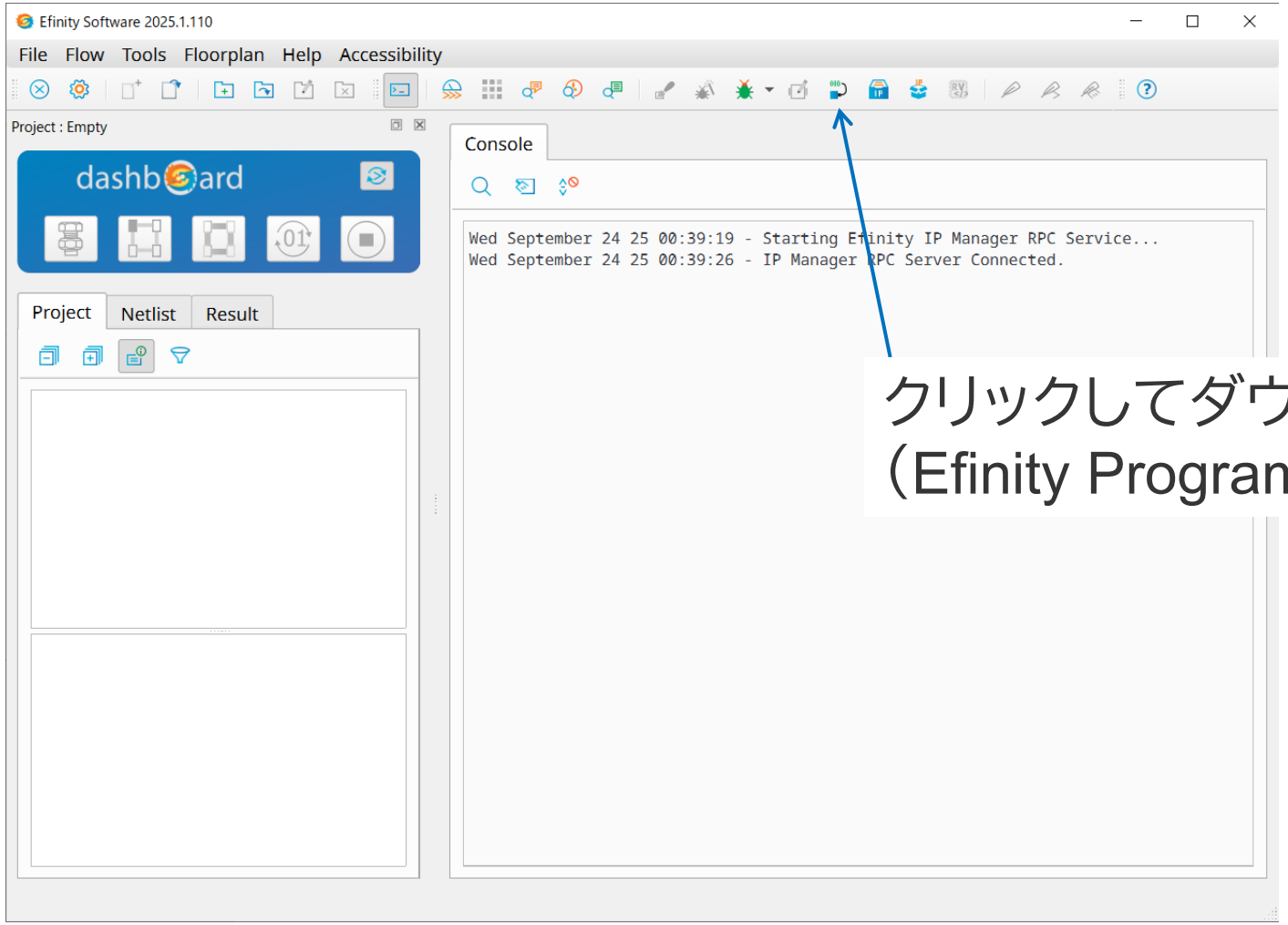


Closeをクリックして終了する

インストールが成功したときの表示

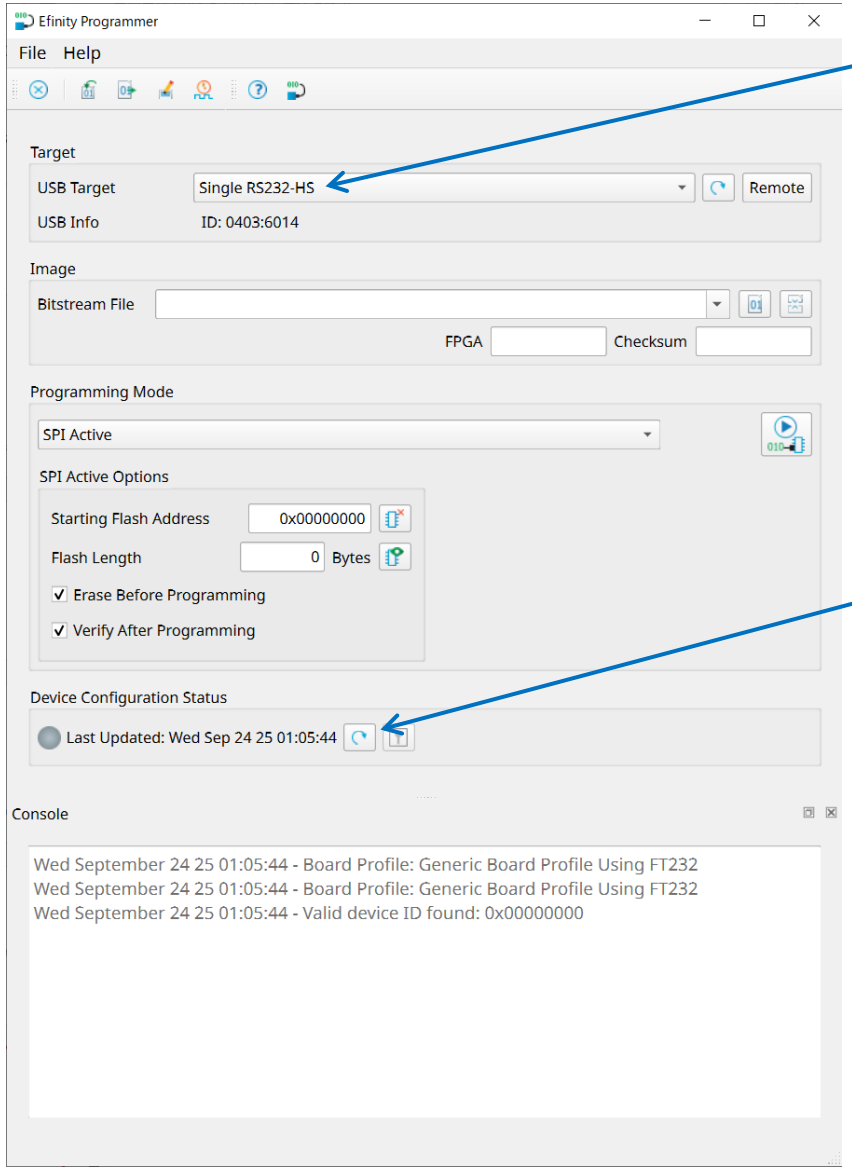
書き込み器とFPGAの接続確認

事前準備でインストールしたEfinityを起動する。



クリックしてダウンロードソフト
(Efinity Programmer)を起動

FPGAのチェック

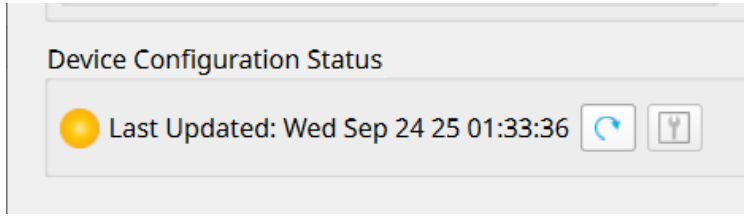


ドライバーが正しく割り当てられている場合、
Single RS232-HSが選択される。

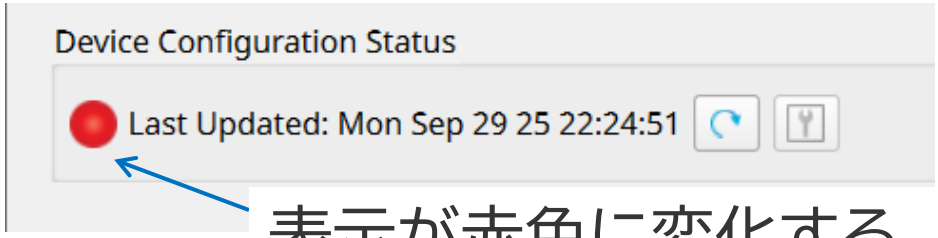
ここをクリックしてFPGAが認識するかチェックする

FPGAの認識

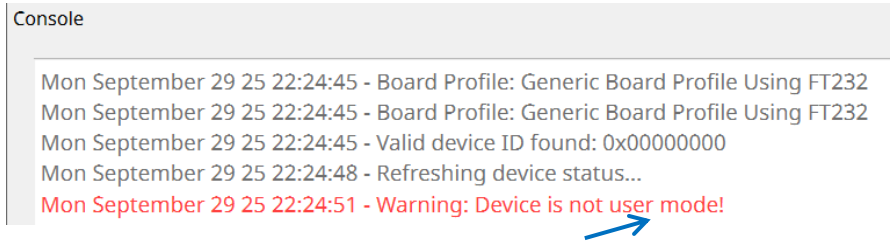
確認中・・・



認識できた場合

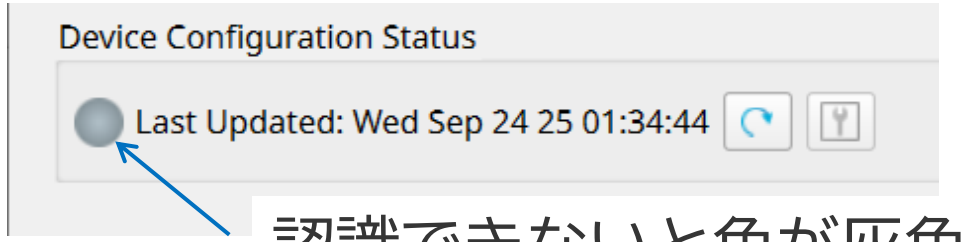


表示が赤色に変化する。



Warning: Device is not user mode!と
なっていればOK

認識できなかった場合

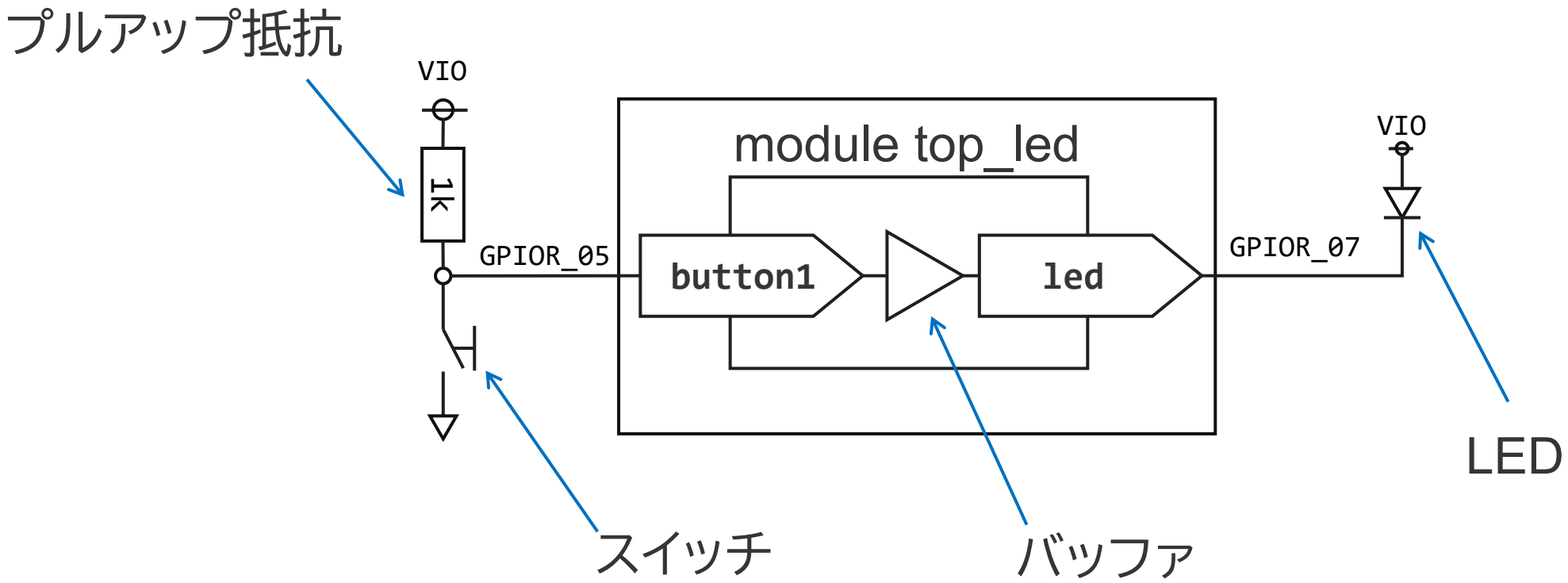


認識できないと色が灰色になる

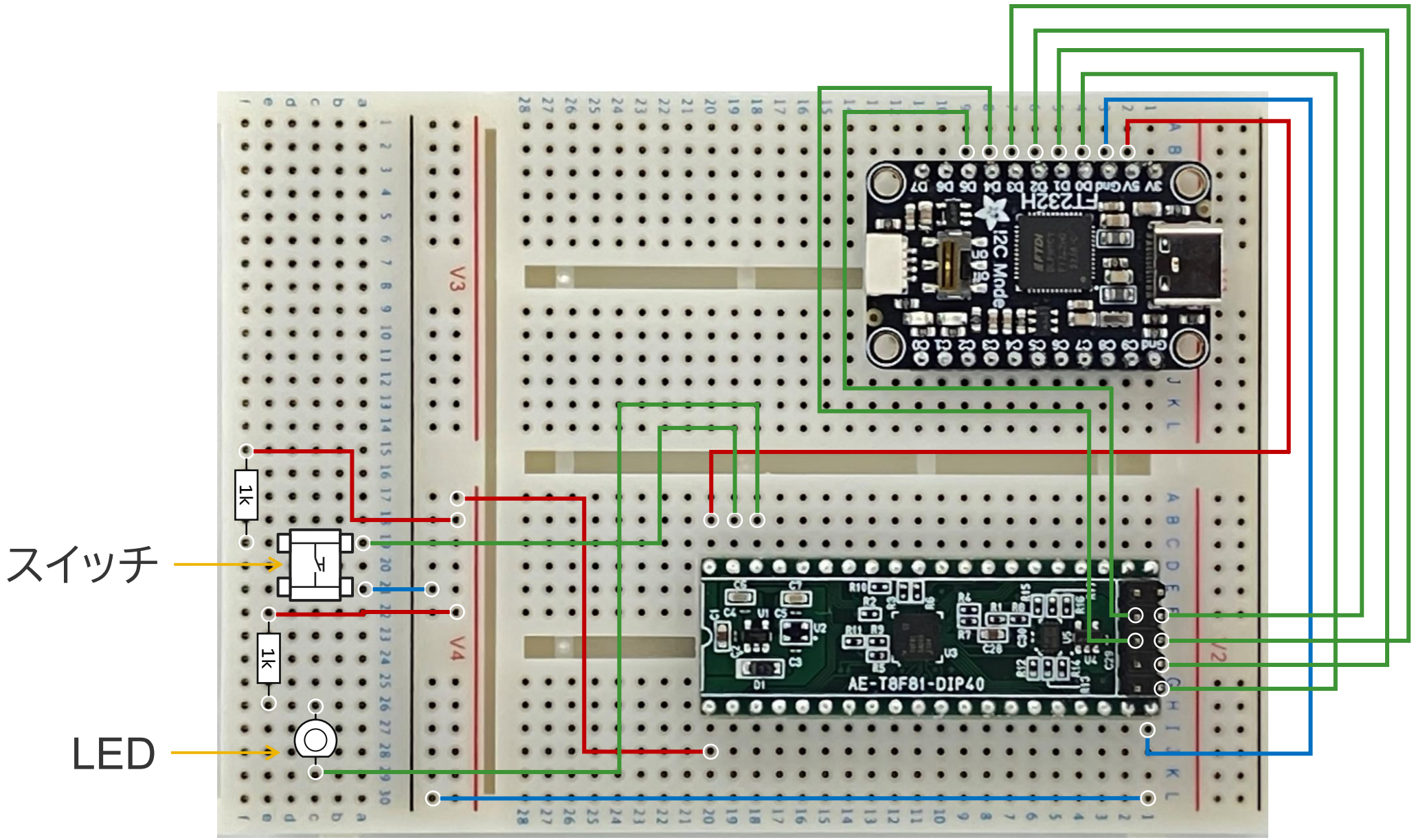
認識できない場合配線が間違っているので、よく確認する。

スイッチとLEDの回路と Efinityを用いたFPGA開発

LEDとスイッチ回路

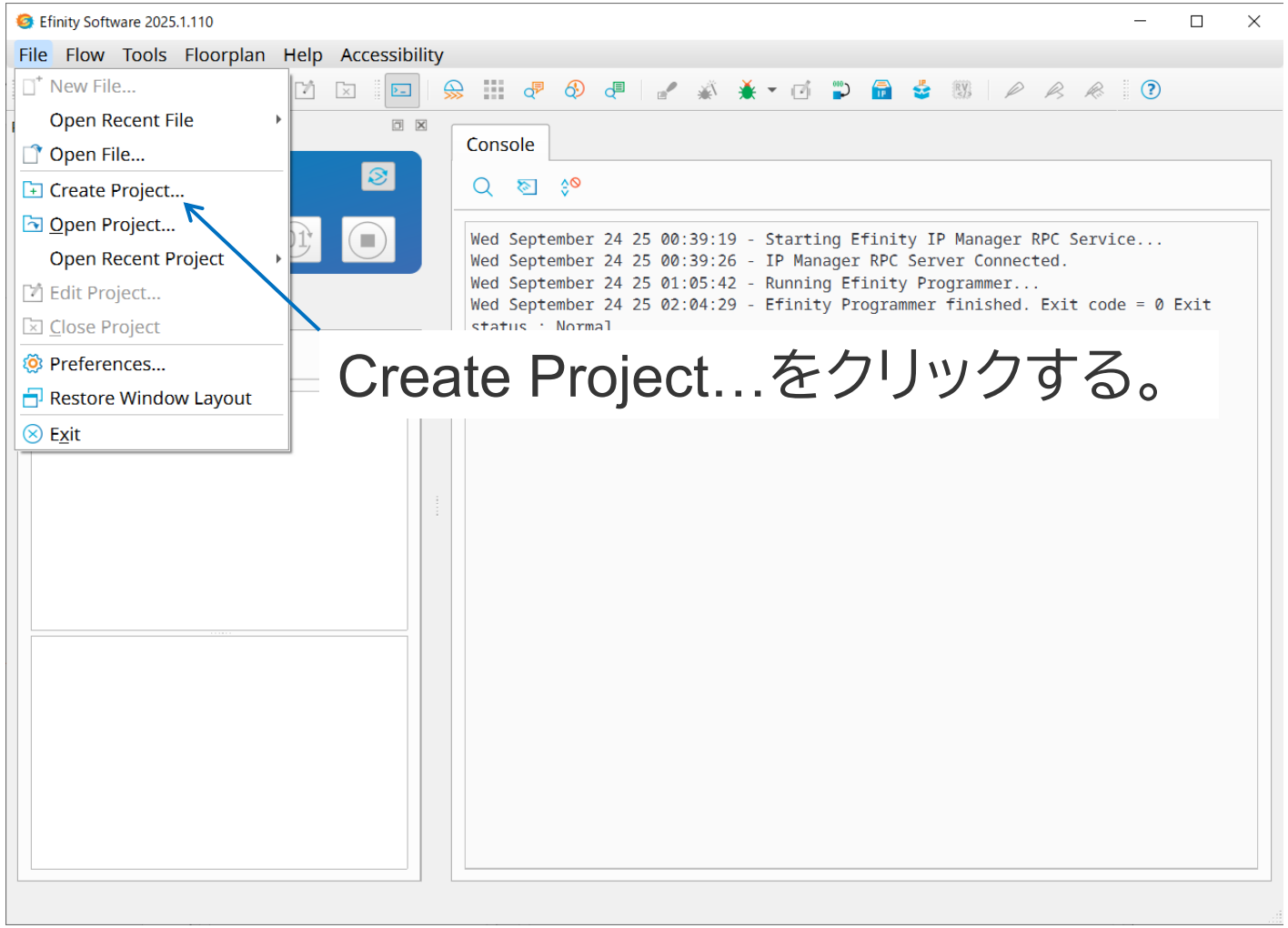

プルアップ抵抗
VIO
1k
GPIOR_05
module top_led
button1
led
GPIOR_07
VIO
LED
スイッチ
バッファ

LEDとスイッチ回路の配線図

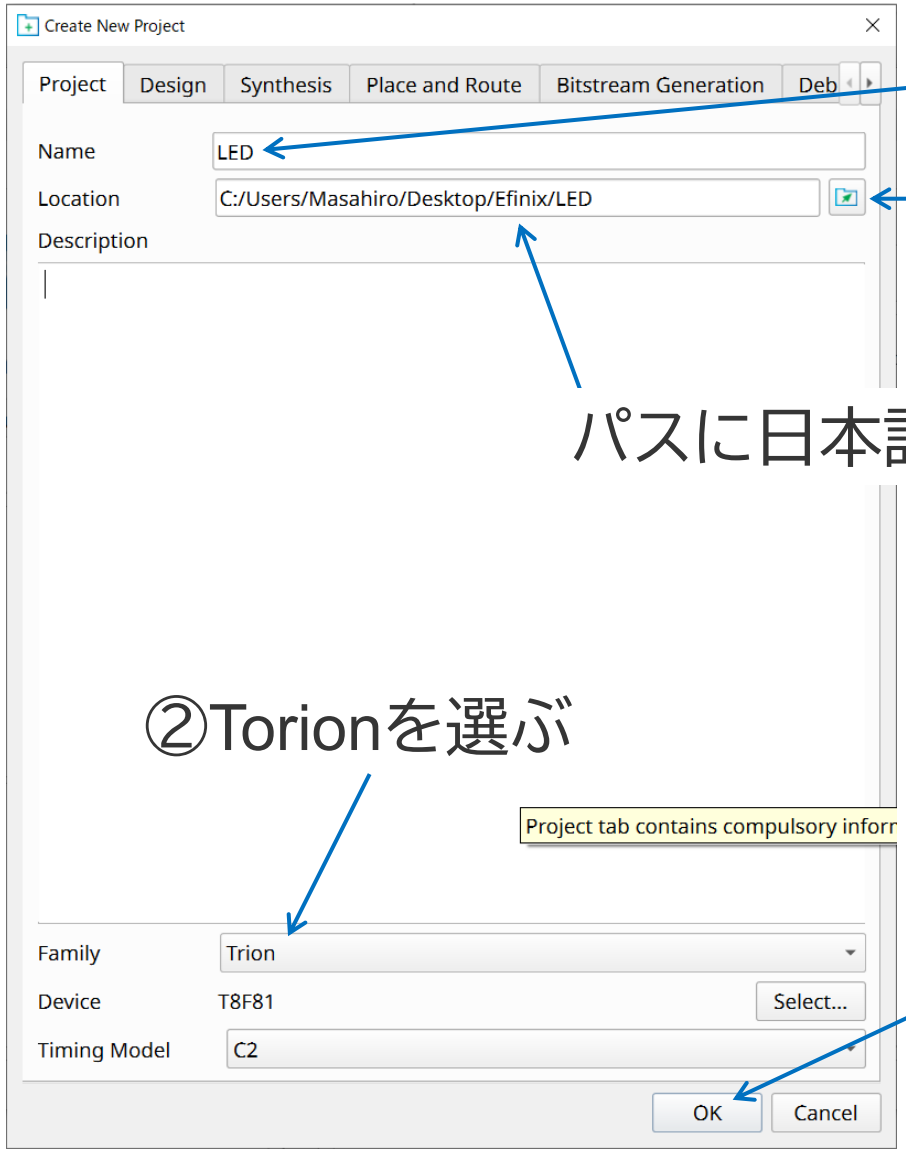


プロジェクト作成

File→Create Project…をクリックする。



プロジェクトウィザード



プロジェクト名はフォルダ名が自動でつく

①エクスプローラを開いてプロジェクトフォルダを作る

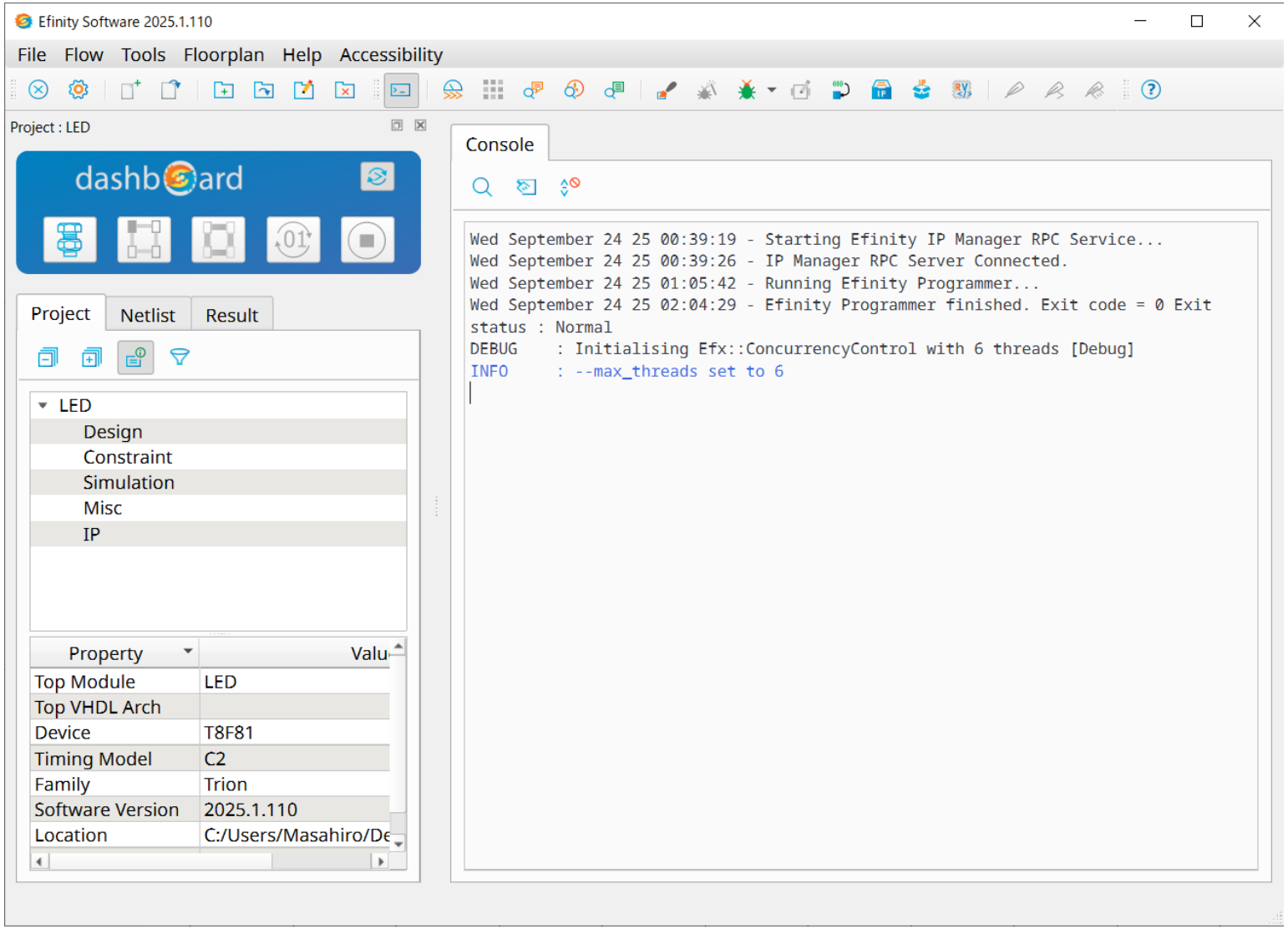
パスに日本語が入らないようにする

②Torionを選ぶ

③フォルダを作成したらOKをクリック

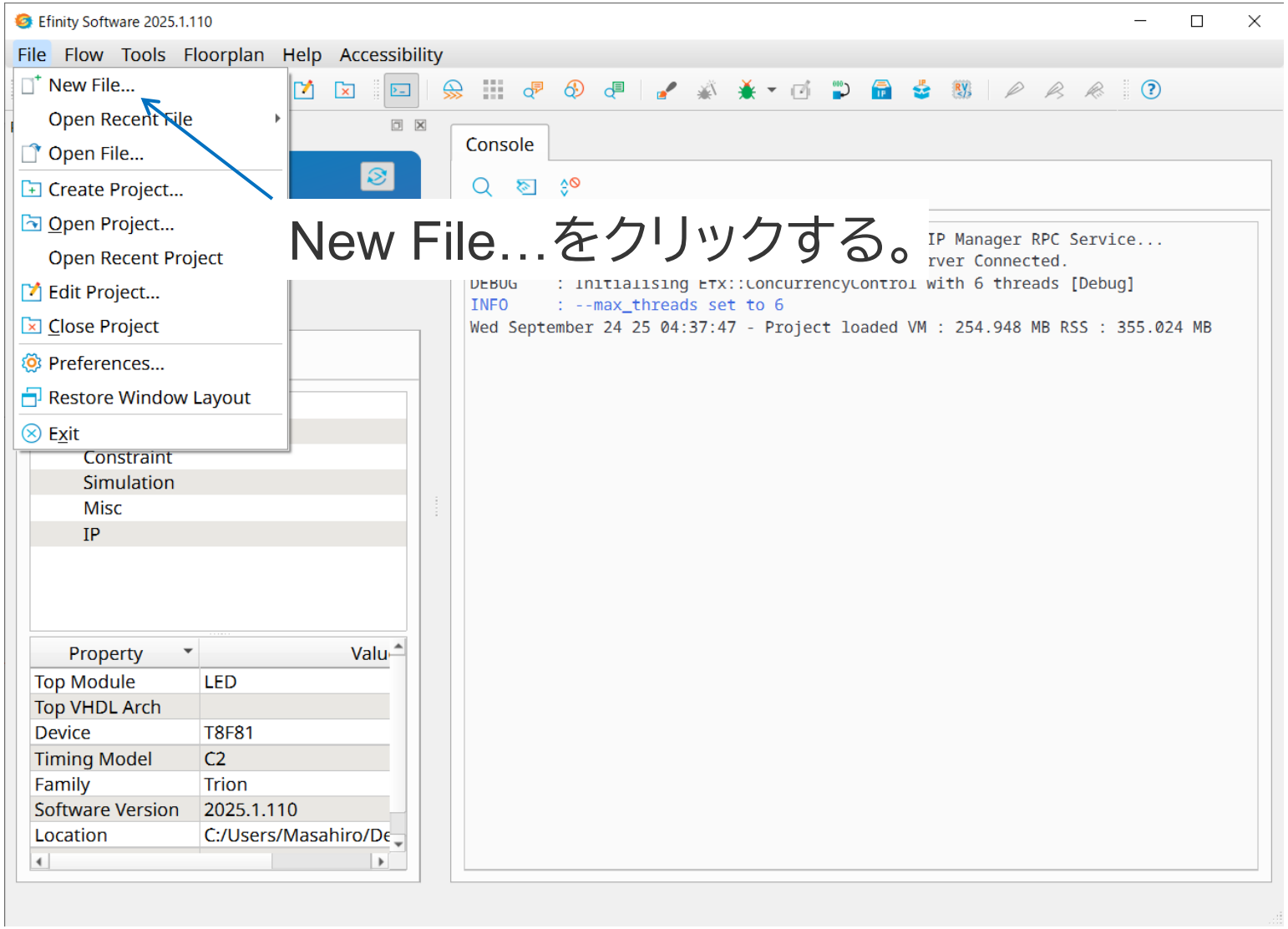
プロジェクトが開いた状態

この部分が色々表示が変わる

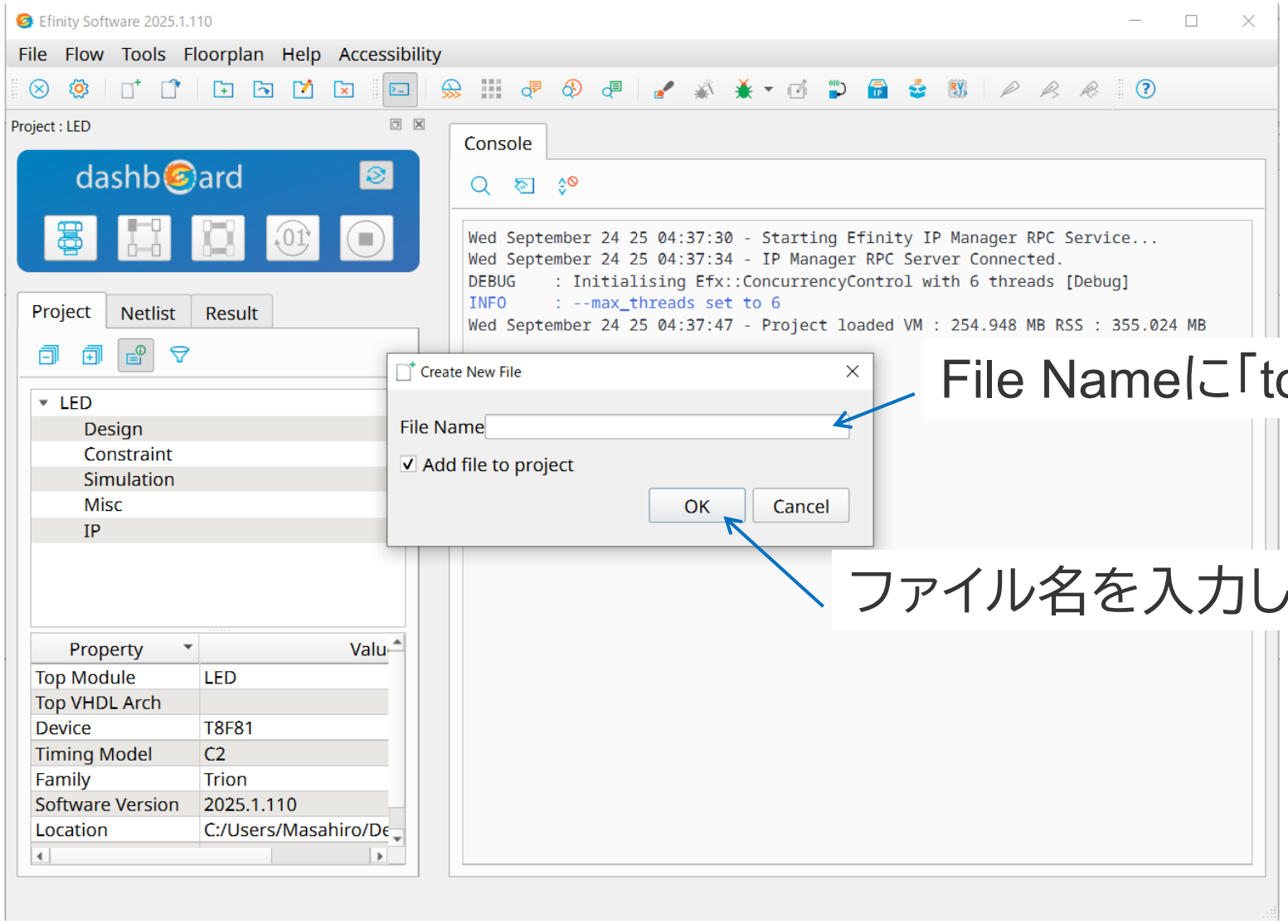


Verilogファイルの追加1

File→New File…をクリックする。



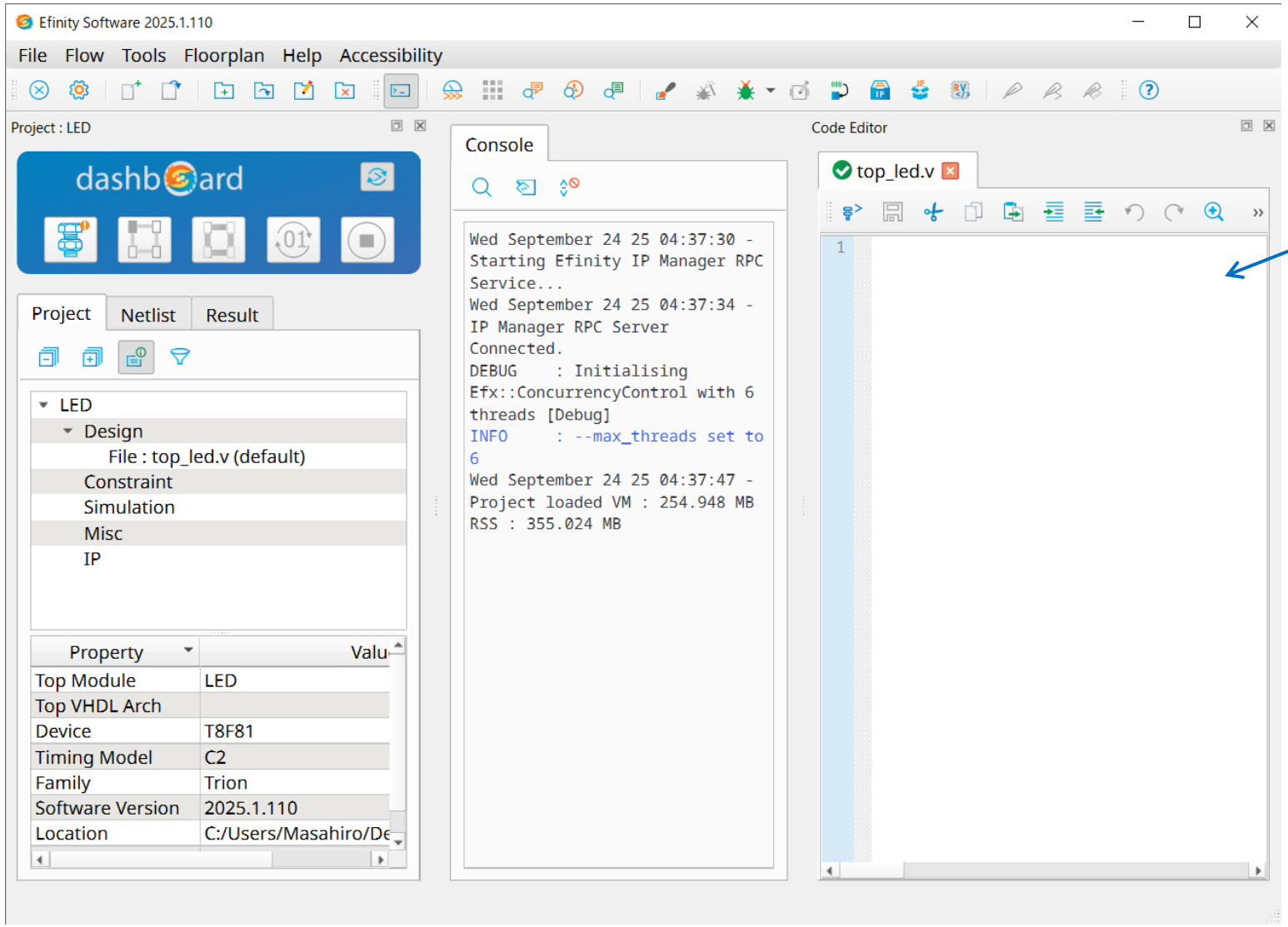
Verilogファイルの追加2



File Nameに「top_led.v」を入力

ファイル名を入力したらOKをクリック

コード入力画面



ここにコードを入力する。
真ん中のConsoleを縮めて、
入力画面を広げても良い。

入力するコード

top_led.v

```
1  module top_led(
2  input wire button,
3  output wire led);
4
5  assign led = button;
6
7  endmodule
8
```

モジュール名はファイル名から拡張子を除いたものと同じにする

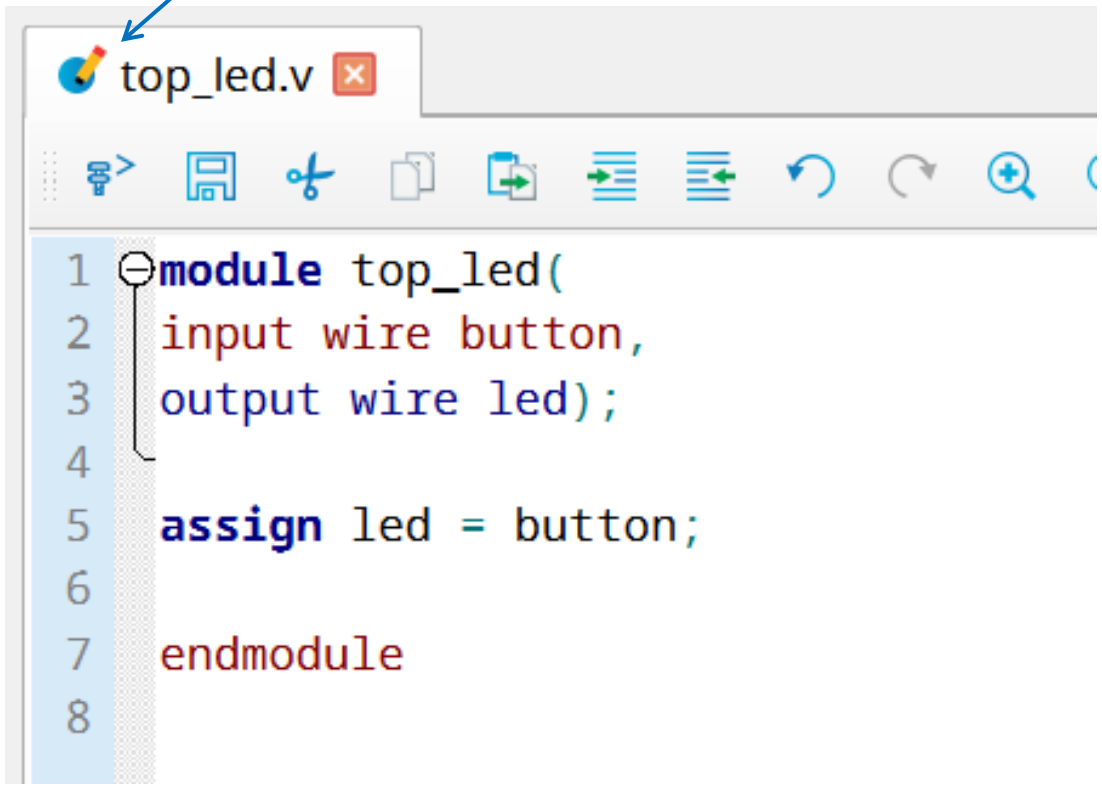
ボタンの信号を受け取るポート

LEDへ信号を出力するポート

ボタンの信号を反転(NOT)してLEDへ出力

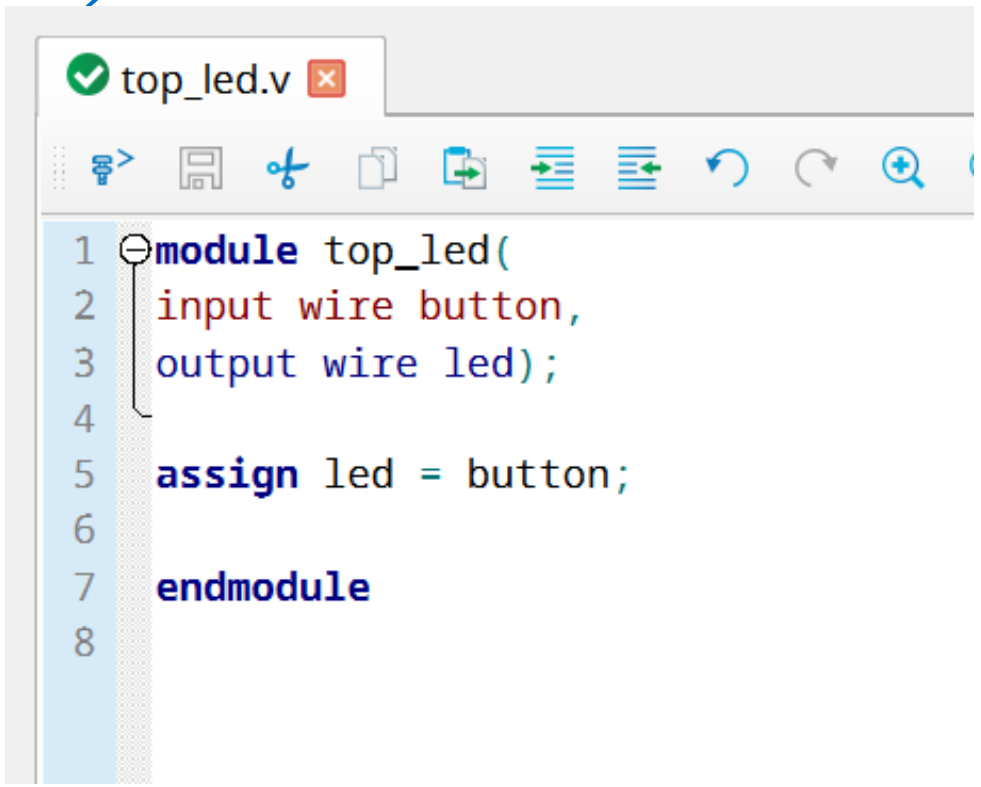
コードの保存

コード保存前は鉛筆アイコン



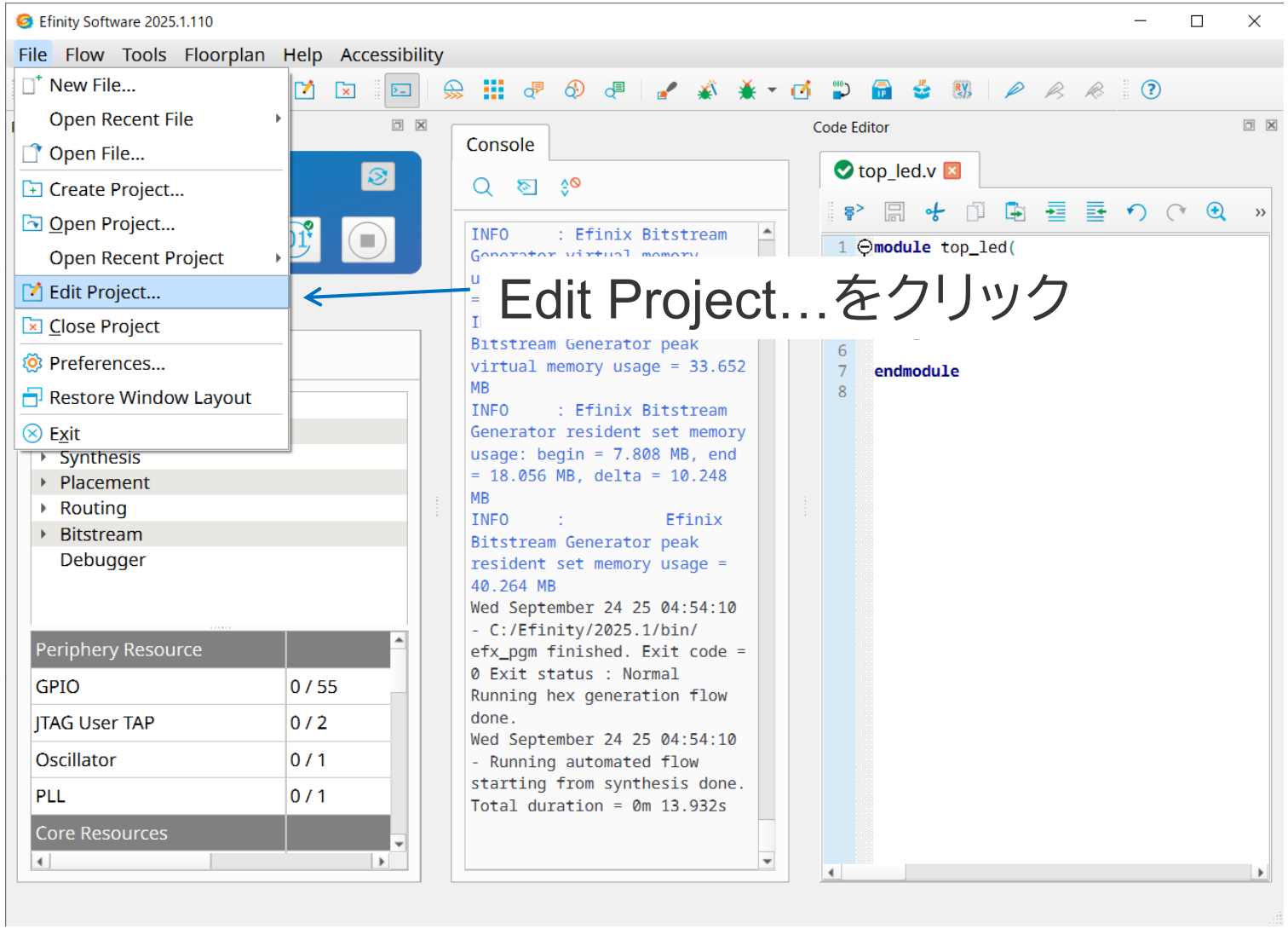
Ctrl + s

保存前の編集がなければ
チェックアイコンがつく



プロジェクト設定の変更(Top Level Moduleの設定)

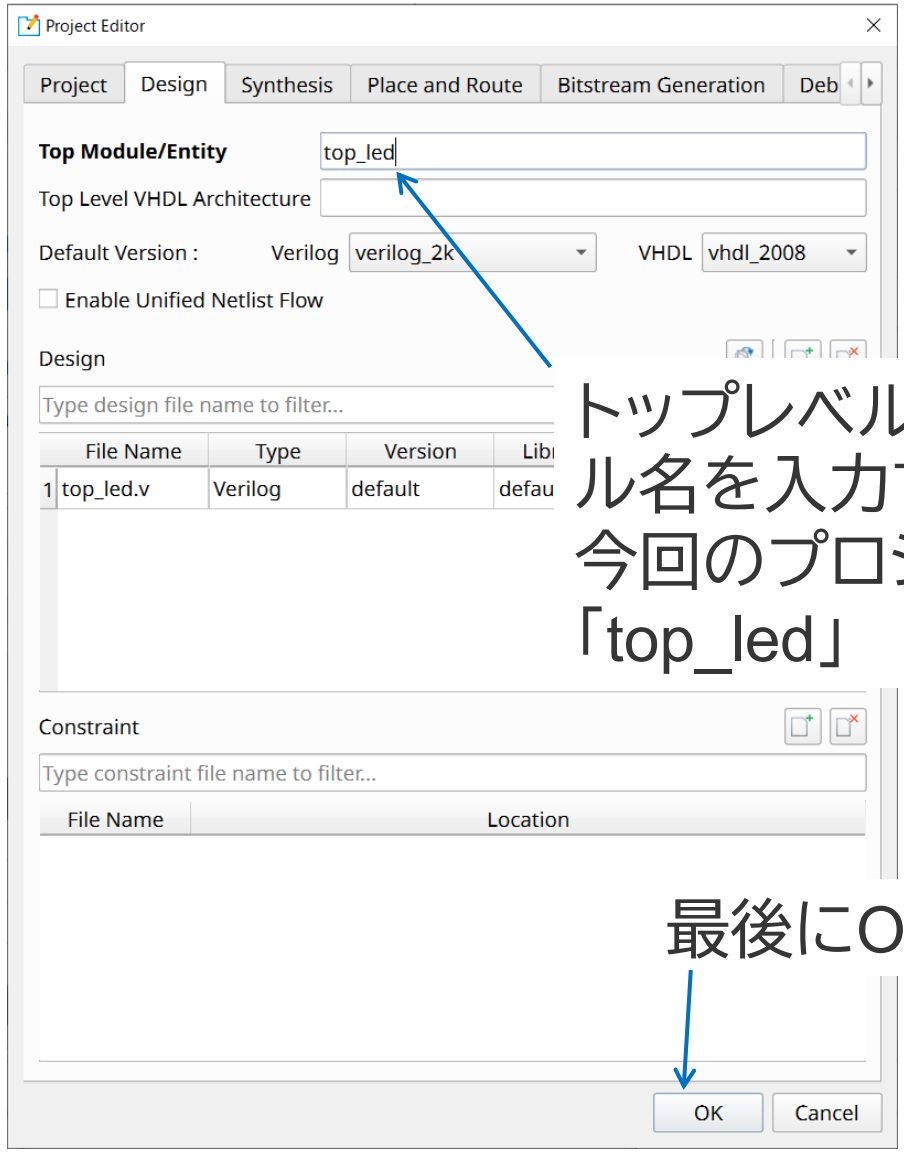
File→Edit Project…をクリックする。



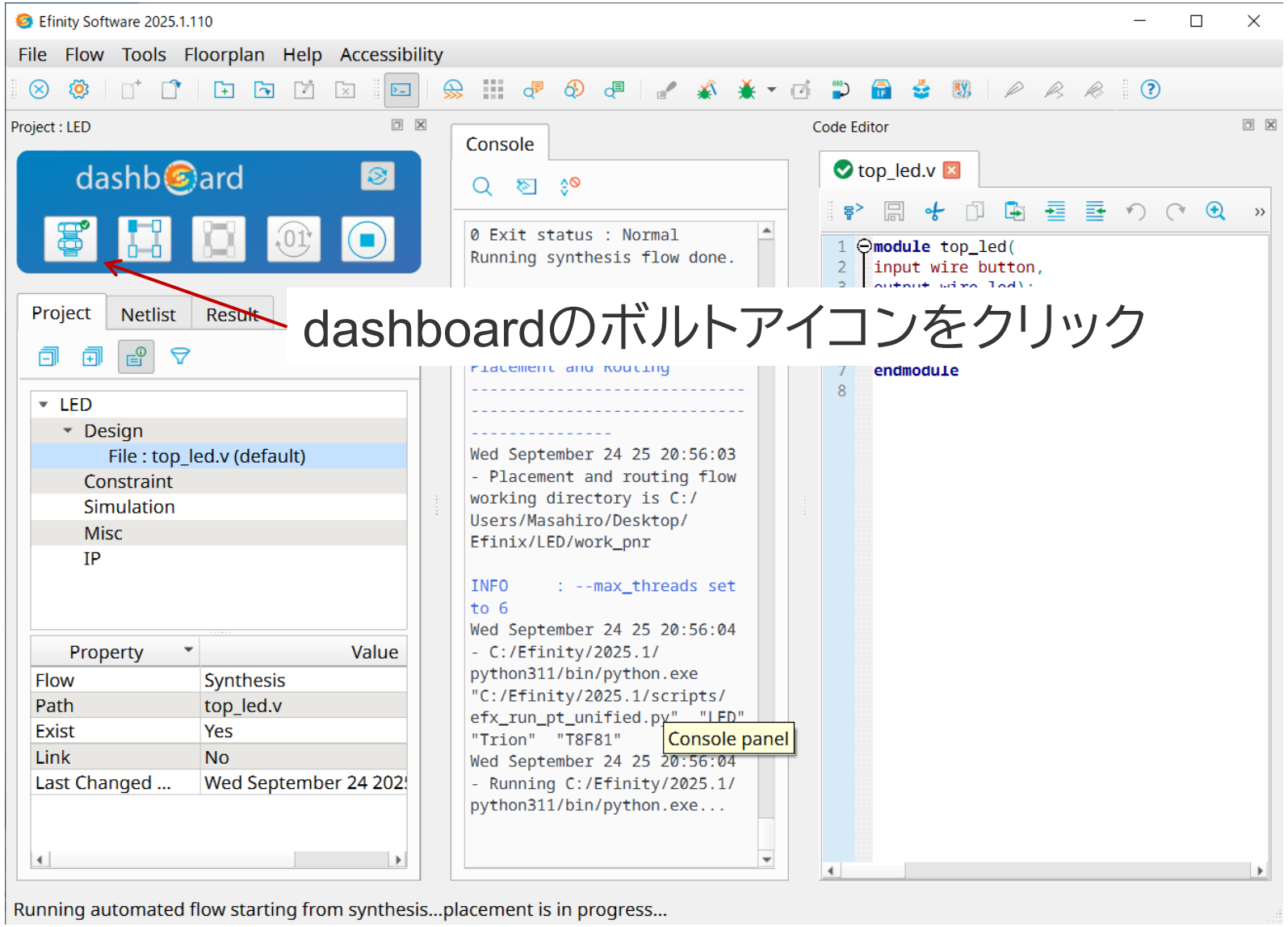
複数のVerilogコードがあるときに、
最上位になるVerilogコード
(Top level module)を指定する。
Verilogコードが1つであったとしても、
トップレベルモジュールは設定が必要。

Projectの設定


Project Editor
Project
Design
Synthesis
Place and Route
Bitstream Generation
Deb
Name
LED
Location
C:/Users/Masahiro/Desktop/Efinix/LED
Description
Designをクリック
Family
Trion
Device
T8F81
Select...
Timing Model
C2
OK
Cancel


Project Editor
Project
Design
Synthesis
Place and Route
Bitstream Generation
Deb
Top Module/Entity
top_led
Top Level VHDL Architecture
Default Version :
Verilog
verilog_2k
VHDL
vhdl_2008
Enable Unified Netlist Flow
Design
Type design file name to filter...
File Name
Type
Version
1
top_led.v
Verilog
default
トップレベルのモジュール名を入力する。
今回のプロジェクトでは「top_led」
Constraint
Type constraint file name to filter...
File Name
Location
最後にOKをクリック
OK
Cancel

Verilogコードの文法チェック

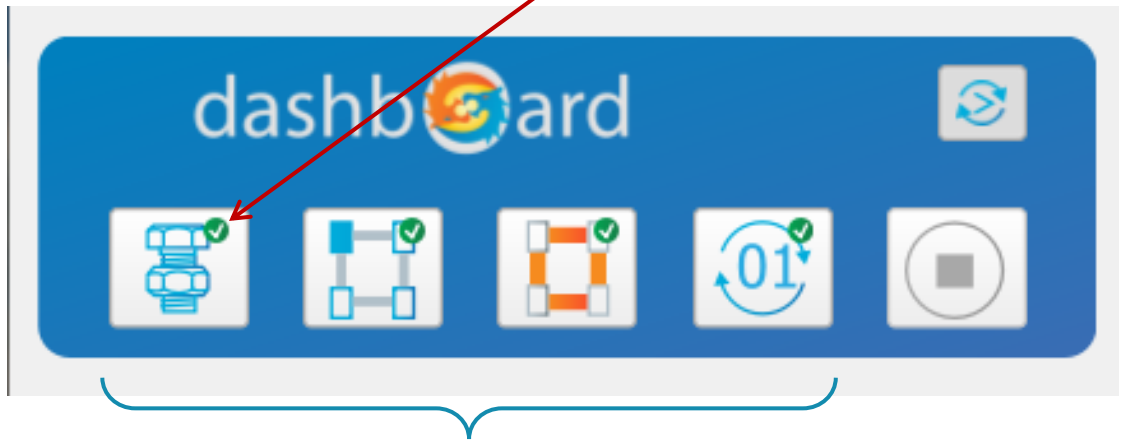


dashboardのボルトアイコンをクリック

実行後

エラーがない場合

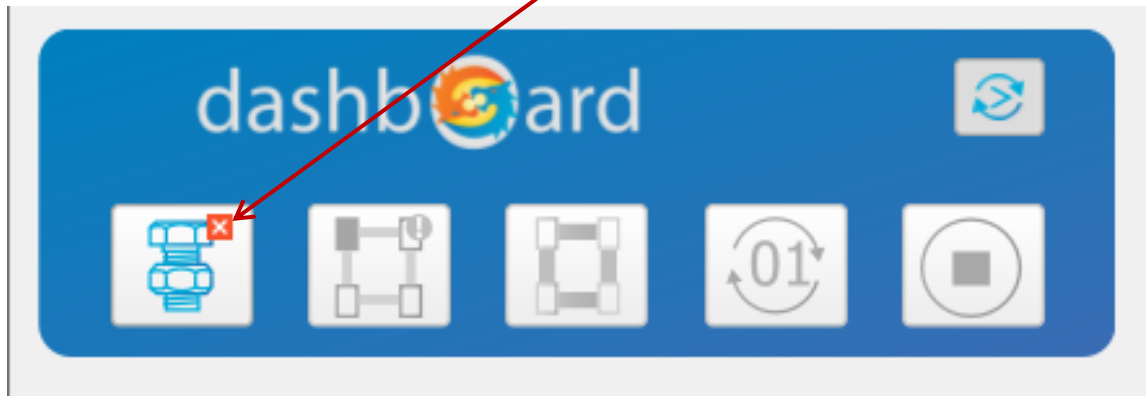
チェックマーク



すべてのアイコンの右上に緑で白抜きの
チェックが付けばOK

エラーがある場合

バツマーク



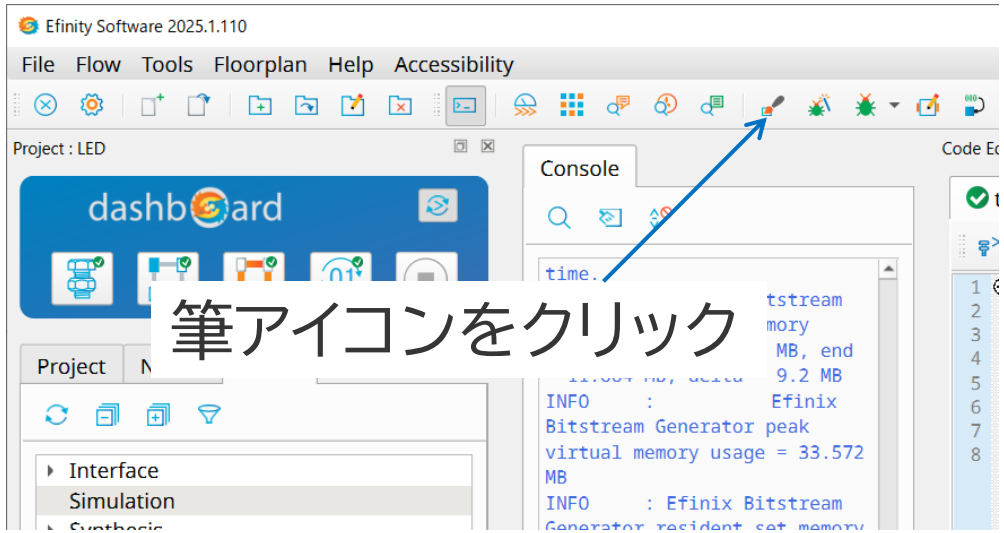

```
Wed September 24 25 21:01:28
- C:/Efinity/2025.1/bin/
efx_map finished. Exit code =
3 Exit status : Normal
Running synthesis flow fail.
See exit code and exit
status.
Wed September 24 25 21:01:28
- Running automated flow
starting from synthesis done.
Total duration = 0m 1.887s
```

```
ERROR    : C:
\Users\Masahiro\Desktop\Efini
x\LED\top_led.v(7): syntax
error near 'endmodule'
[VERI-1137]
ERROR    : C:
\Users\Masahiro\Desktop\Efini
x\LED\top_led.v(7): Verilog
2000 keyword 'endmodule' used
in incorrect context
[VERI-2344]
```

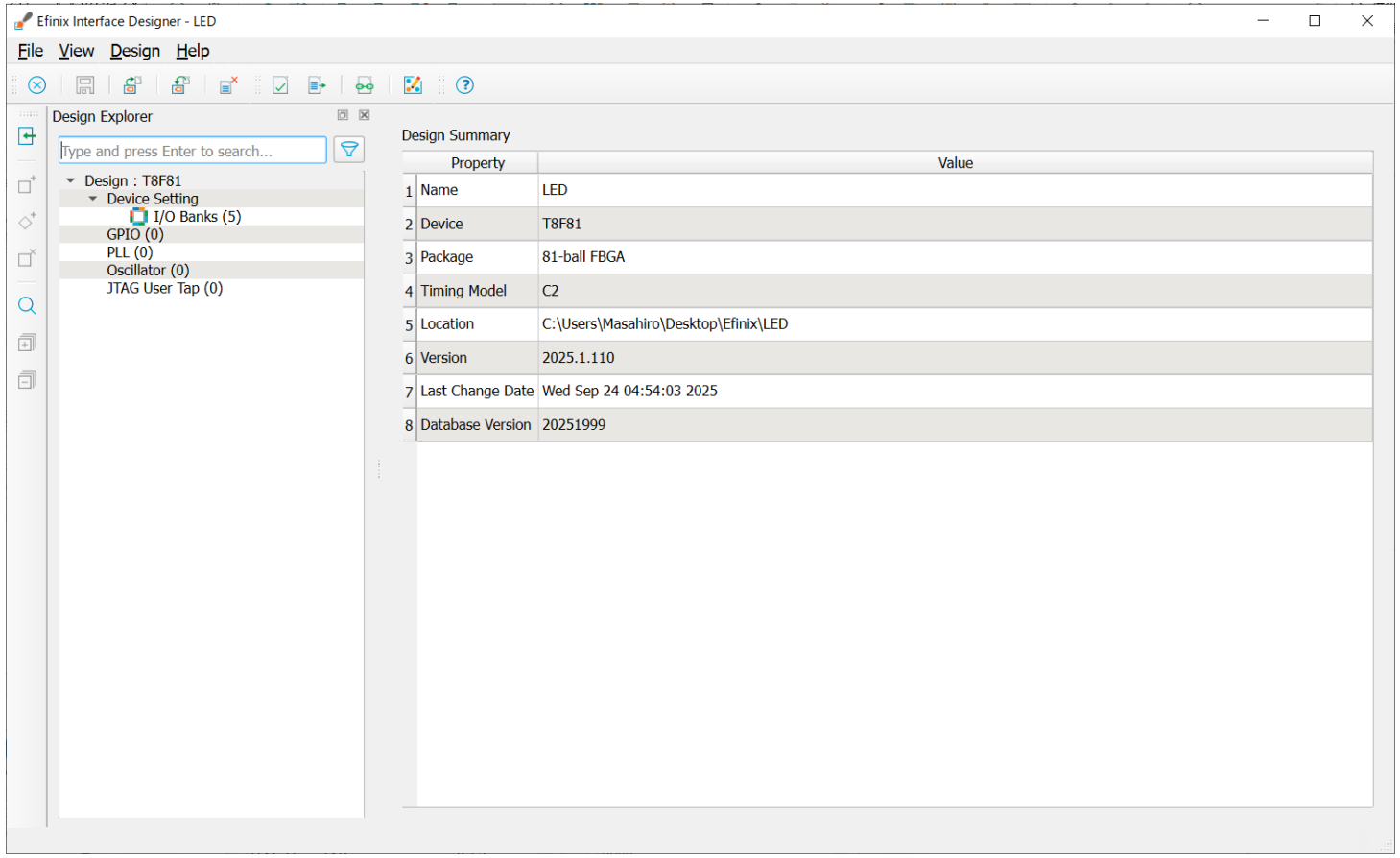
Console画面にエラーの詳細が出るので
よく読む

ピンアサインの手順

Verilogコードのネットリストに宣言したポートをFPGAのピンに割り当てる作業を行う。

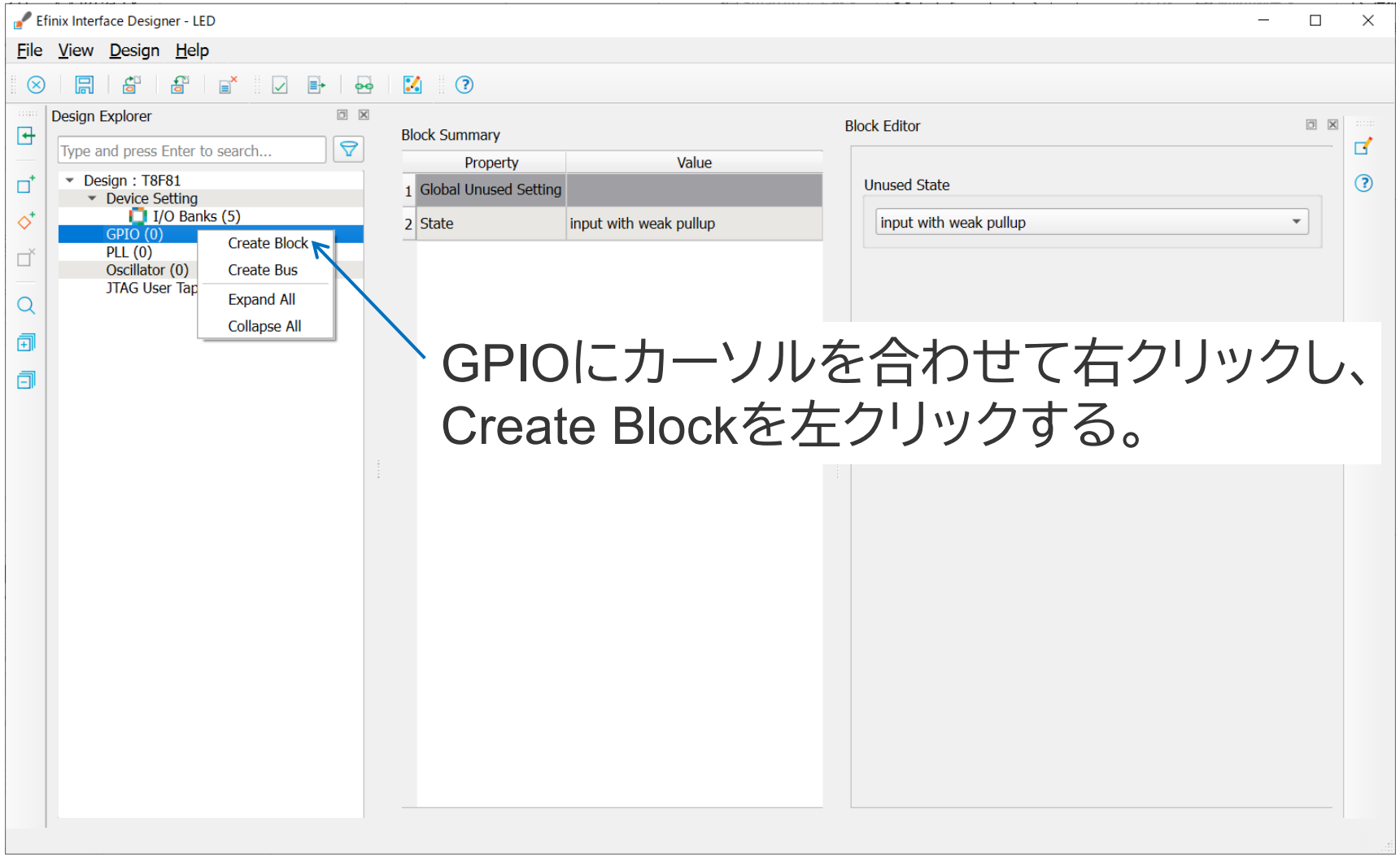


ピンアサインは
Efinix Interface Designer
より行う。



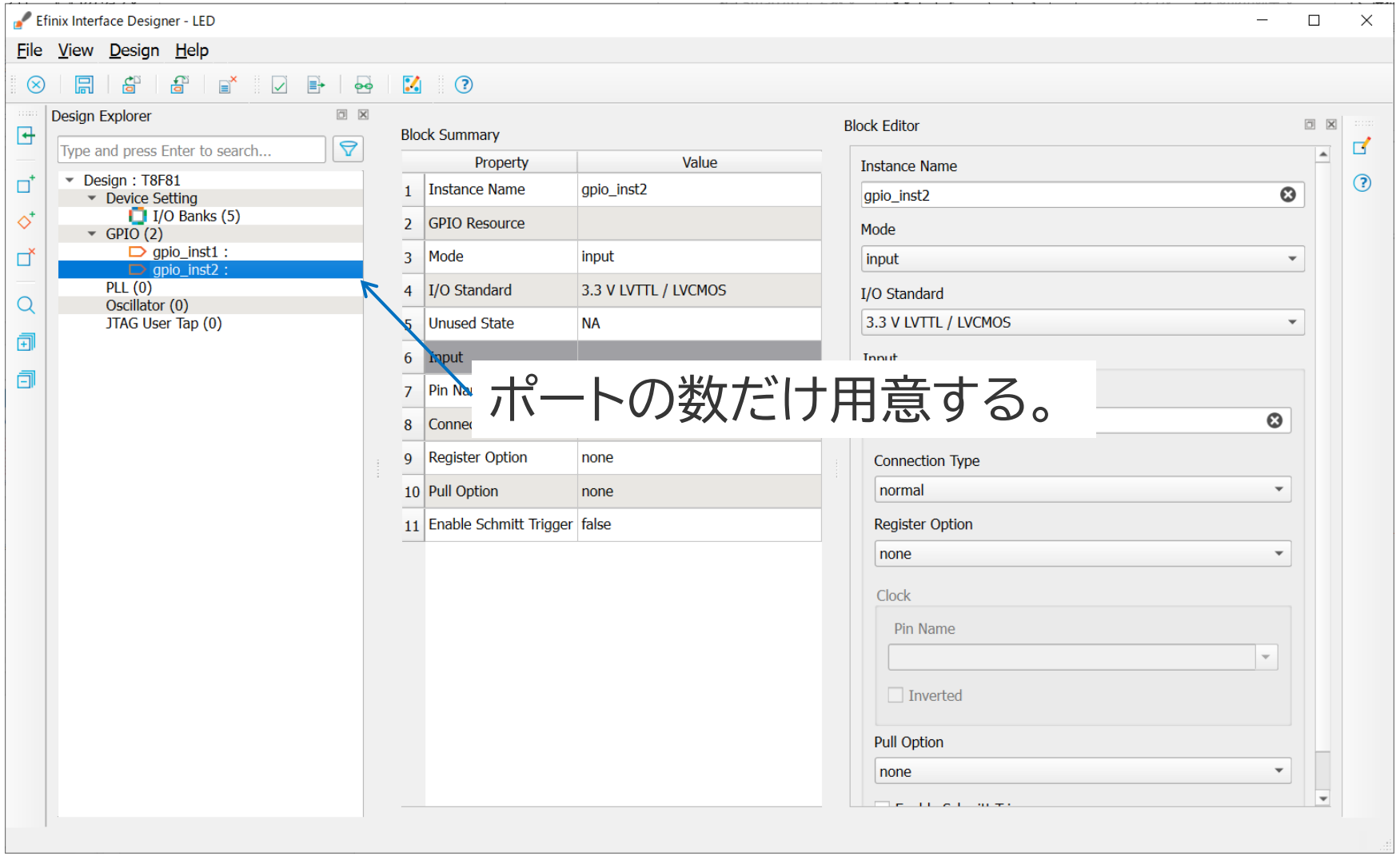
GPIOブロックの作成

GPIO → Create Blockを選択する。



必要な数だけCreate Blockをする

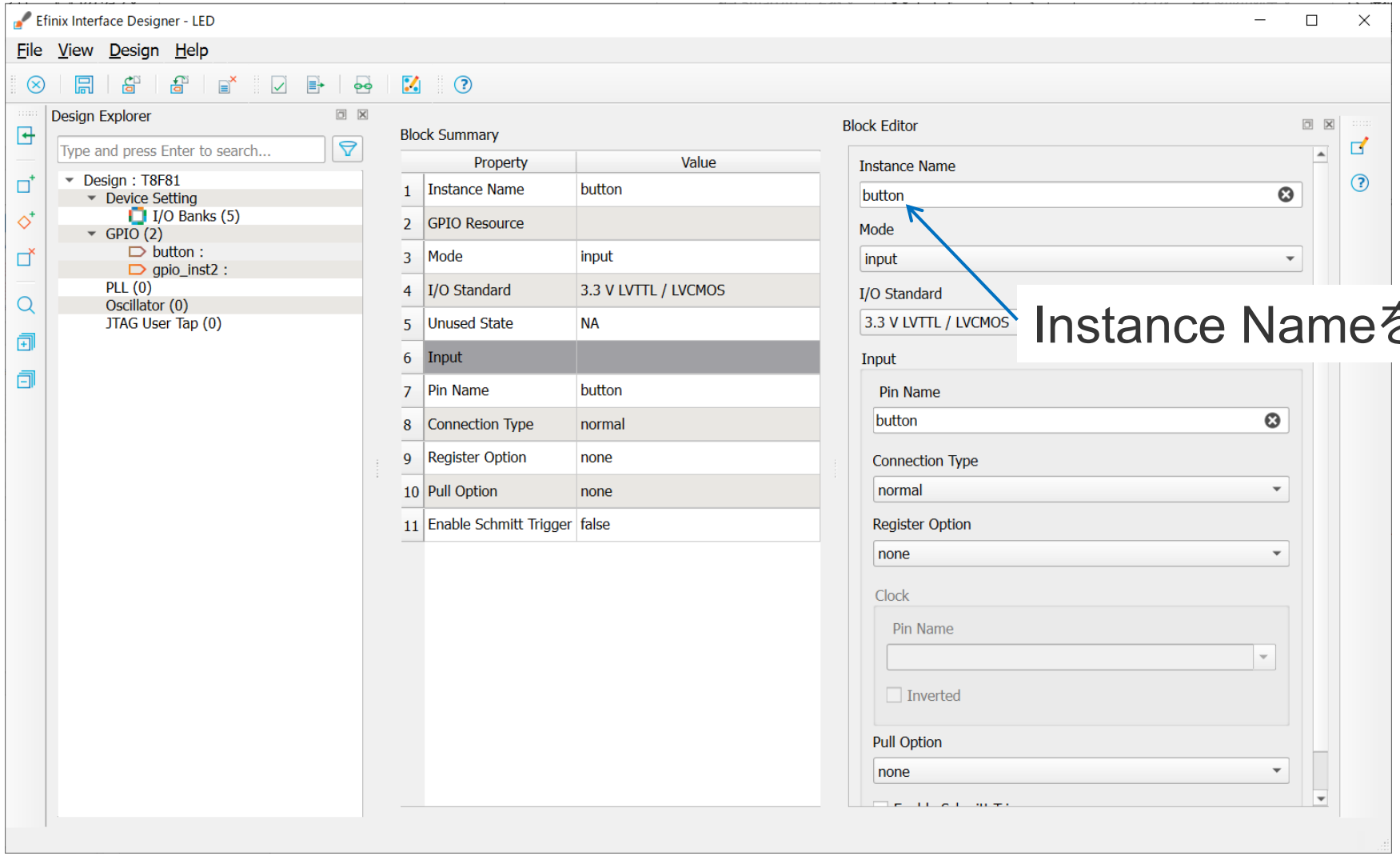
ポートはbuttonとledがあるので、2回Crate Blockをする。



他のFPGAと違って、Verilogからの自動生成はありません…

buttonポートの設定

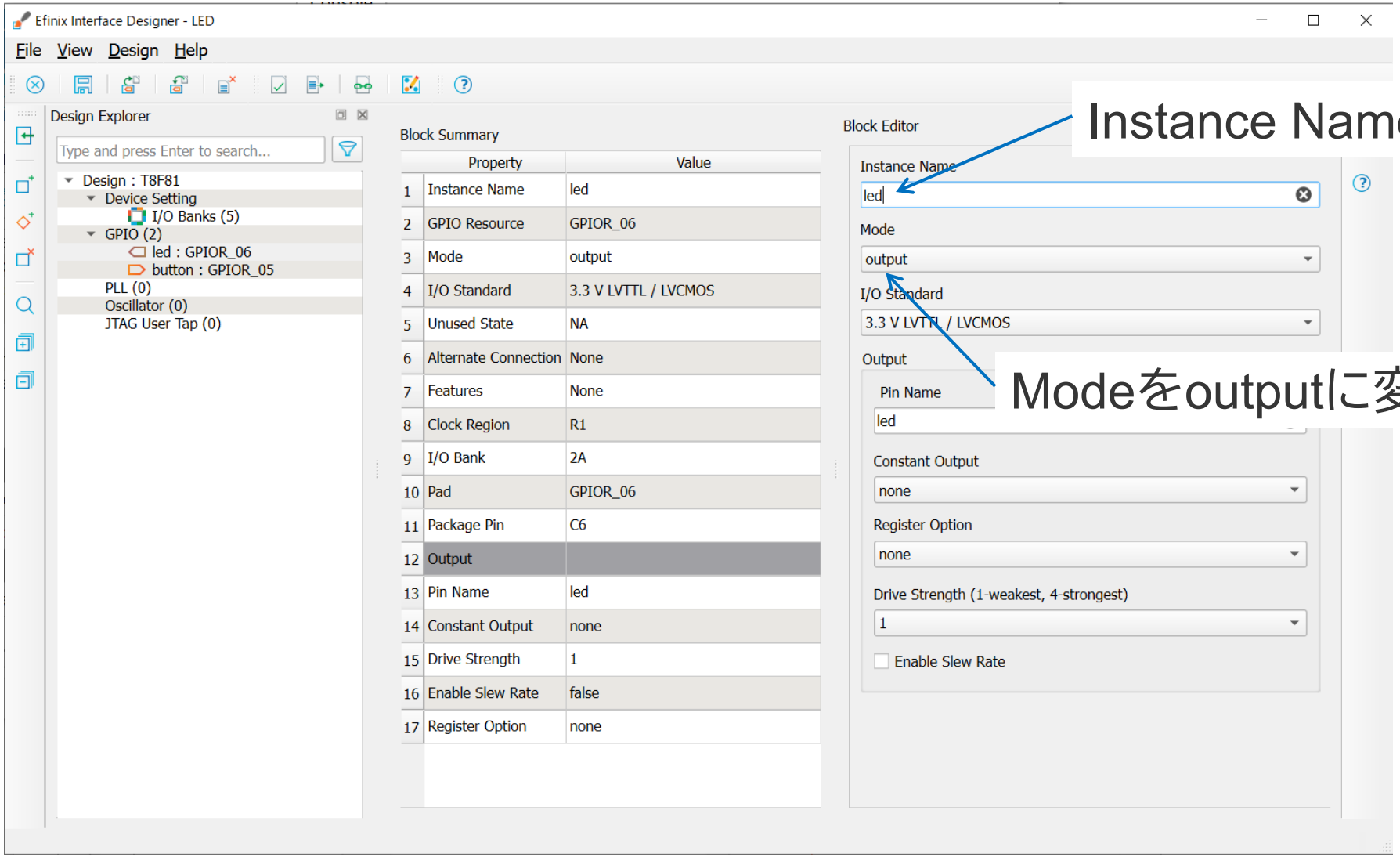
gpio_inst1のInstance Nameをbuttonに変更する。



Instance Nameをbuttonに変更

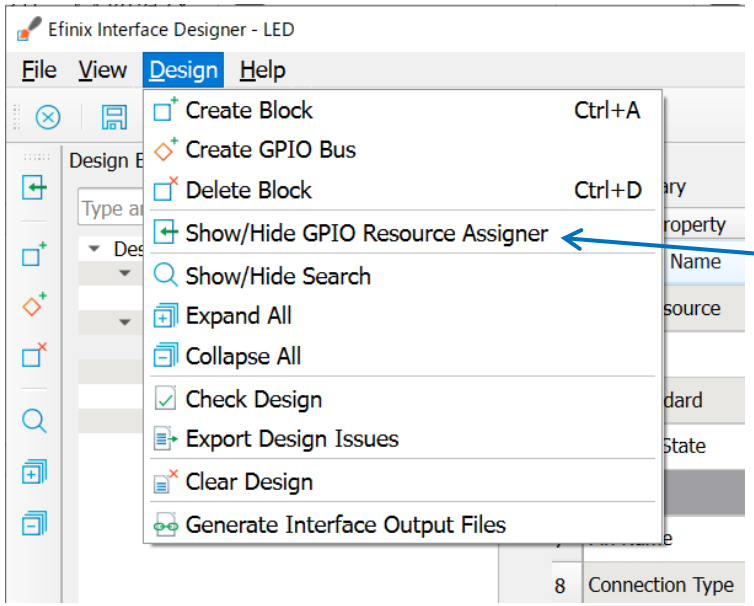
ledポートの設定

gpio_inst2のInstance NameをledにModeをoutputに変更する。

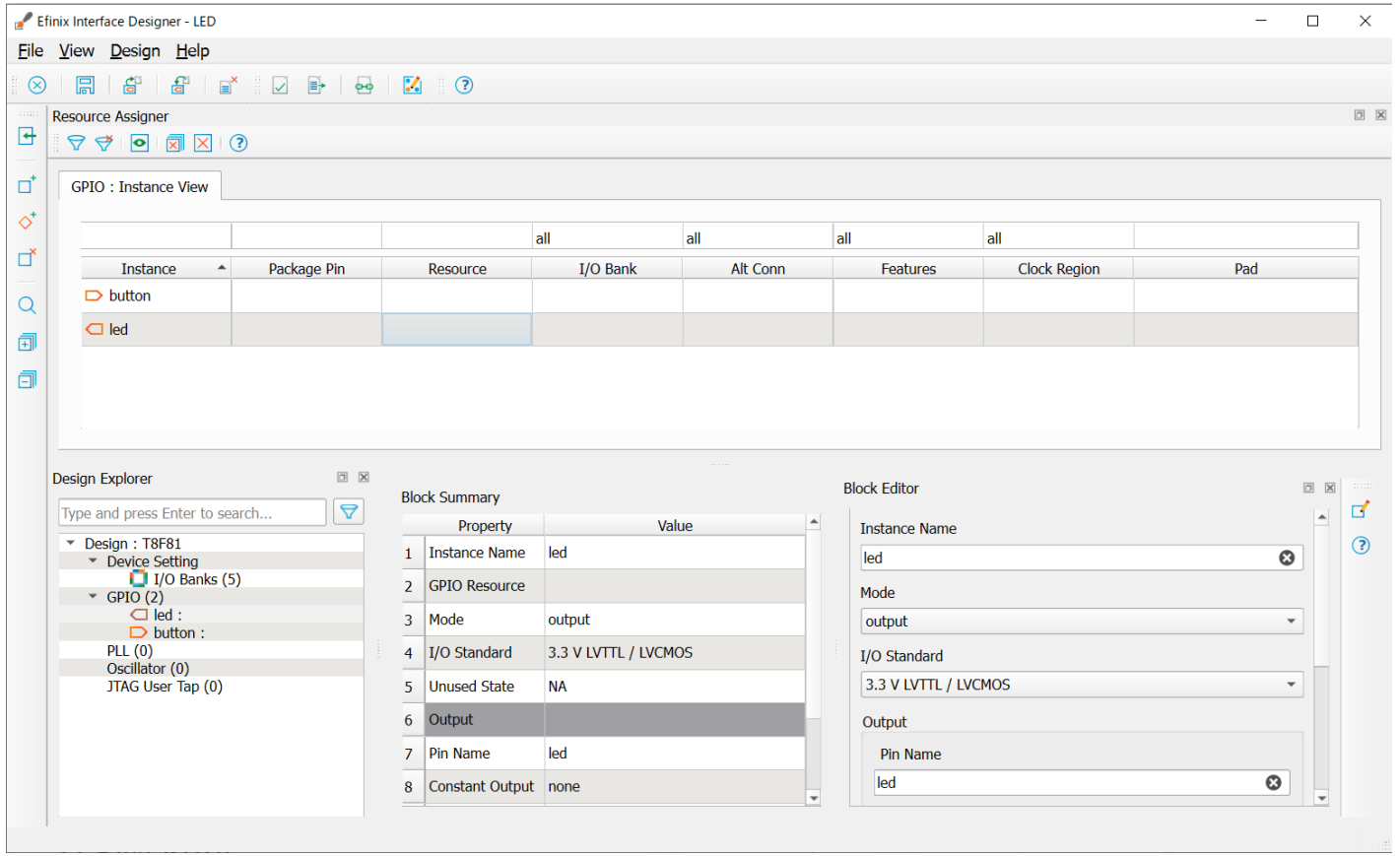


Instance NameをledにModeをoutputに変更

ポートのピンアサイン



Show/Hide GPIO Resource Assignerを左クリックする。



Resource Assignerウィンドウが追加され、GPIO: Instance Viewが現れる。

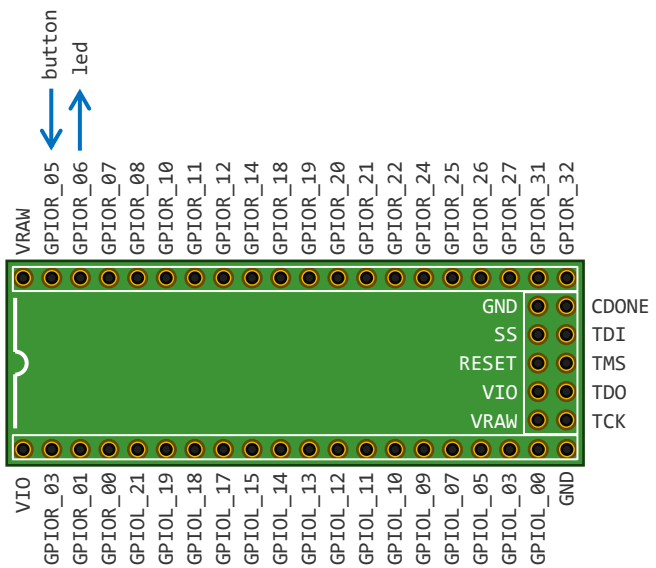
ピンアサイン

VerilogのポートとICパッケージのピンを結びつける。

GPIO : Instance View

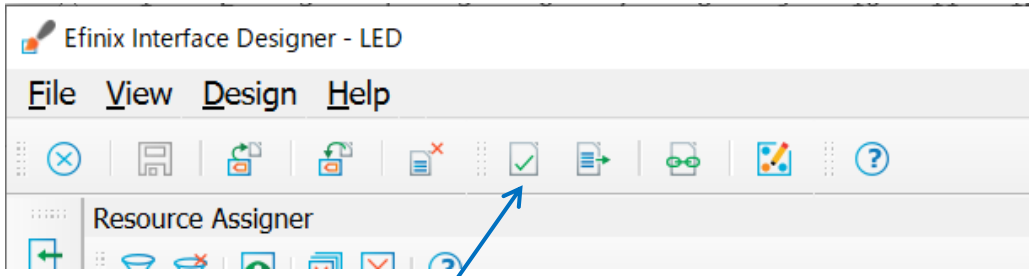
			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
led	C6	GPIOR_06	2A	None	None	R1	GPIOR_06

buttonに**GPIOR_05**、
ledに**GPIOR_06**を指定する。

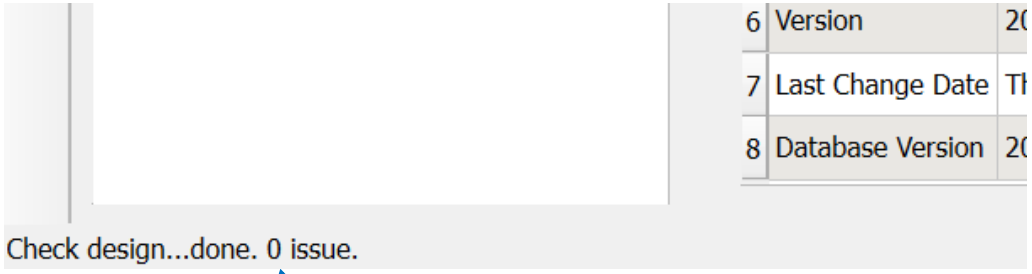


設定ファイルの作成

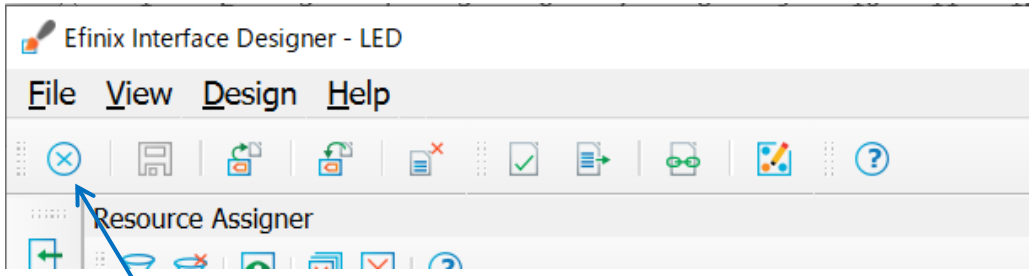
デザインに矛盾がないかチェックする。



チェックアイコンをクリックする。



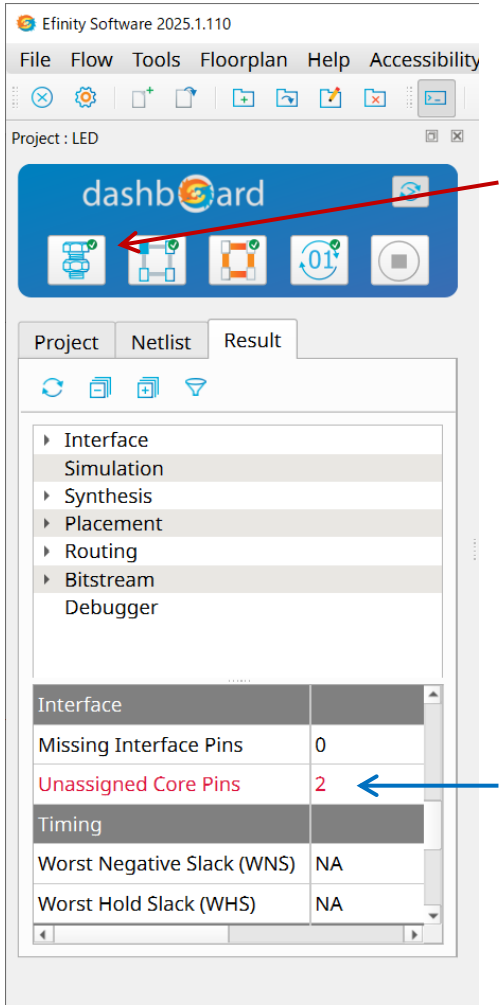
ウィンドウ左下部に0 issueと表示されればOK
これで自動的に設定ファイルが生成される。



バツアイコンをクリックして
Interface Designerを終了する。

合成のやり直し

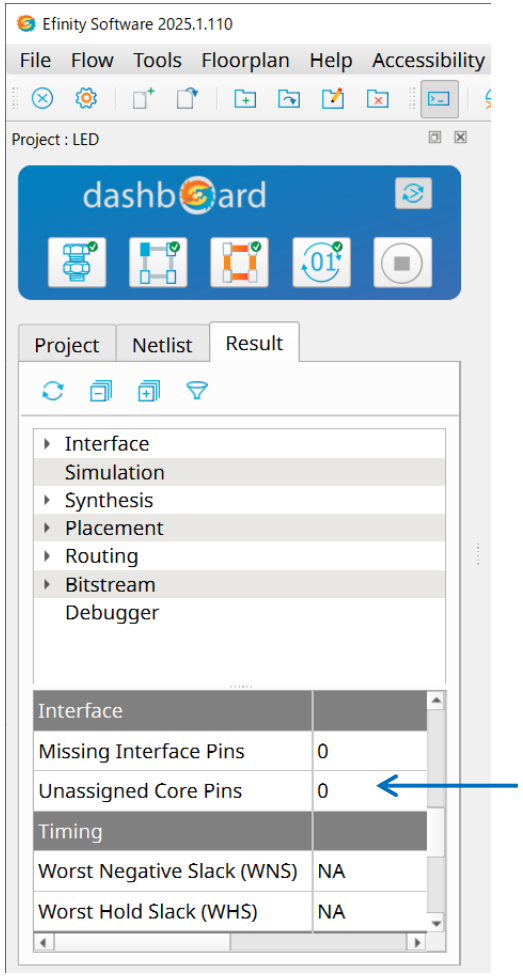
やり直し前



ボルトアイコンをクリックして合成をやり直す

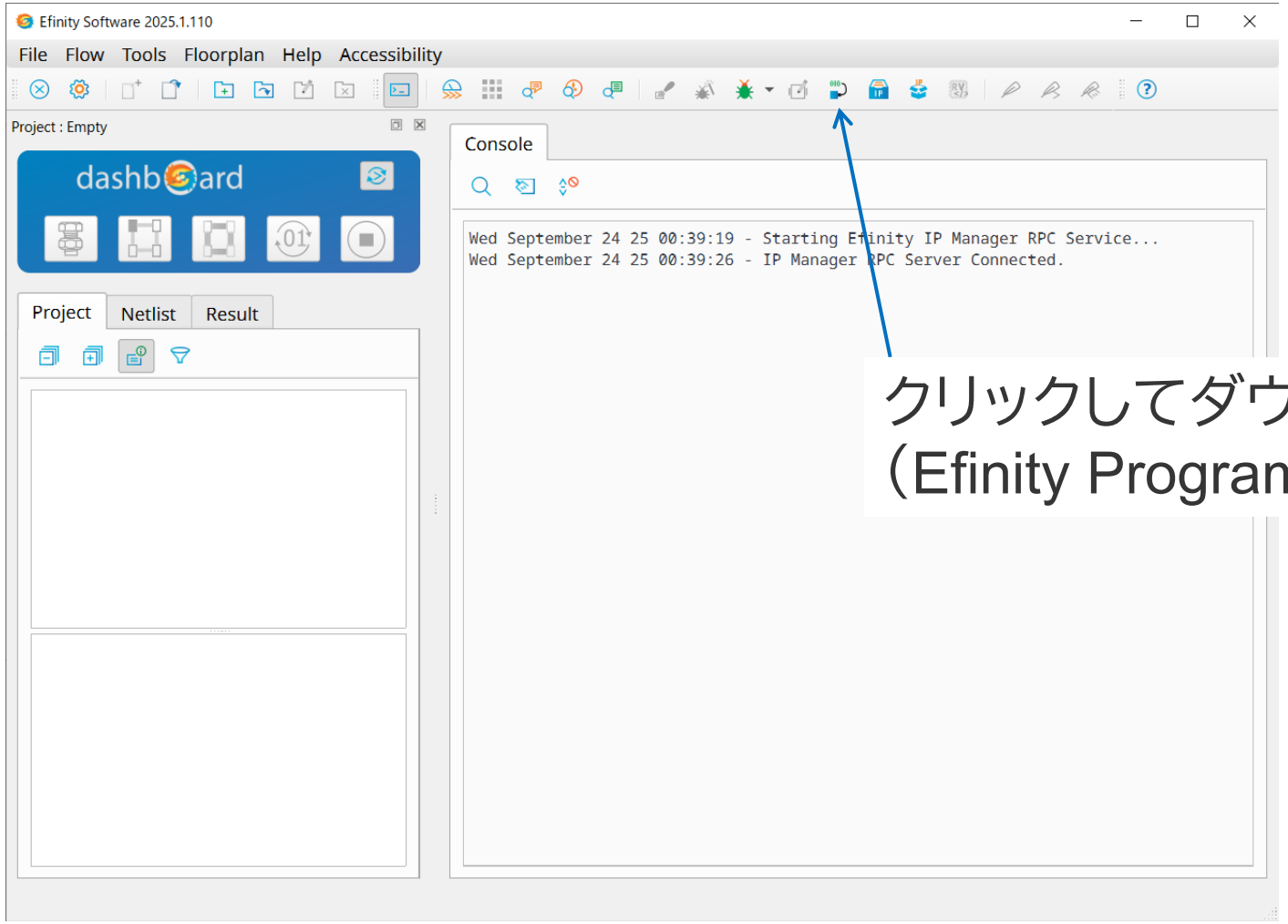
トップレベルからIC外部への接続がないことを意味するエラー

やり直し後



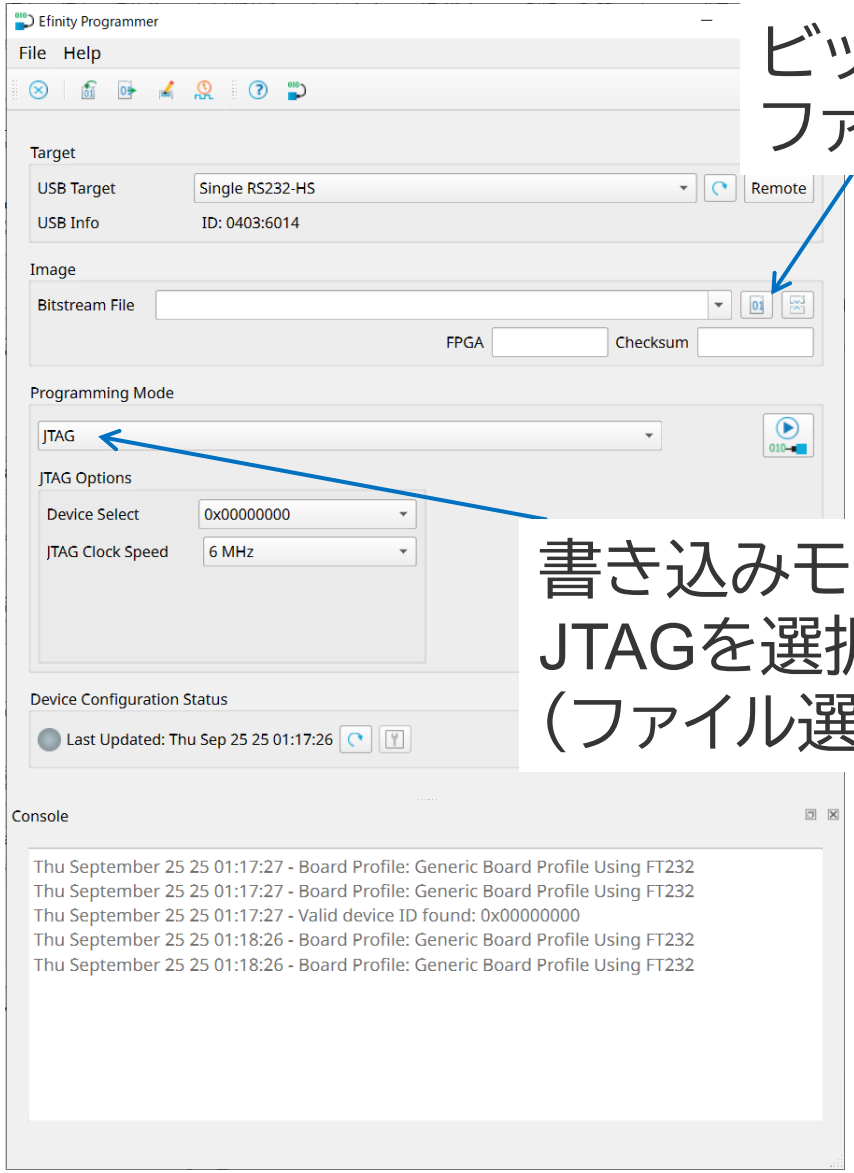
すべてのポートが外部と接続される

FPGAへの書き込みソフトを起動



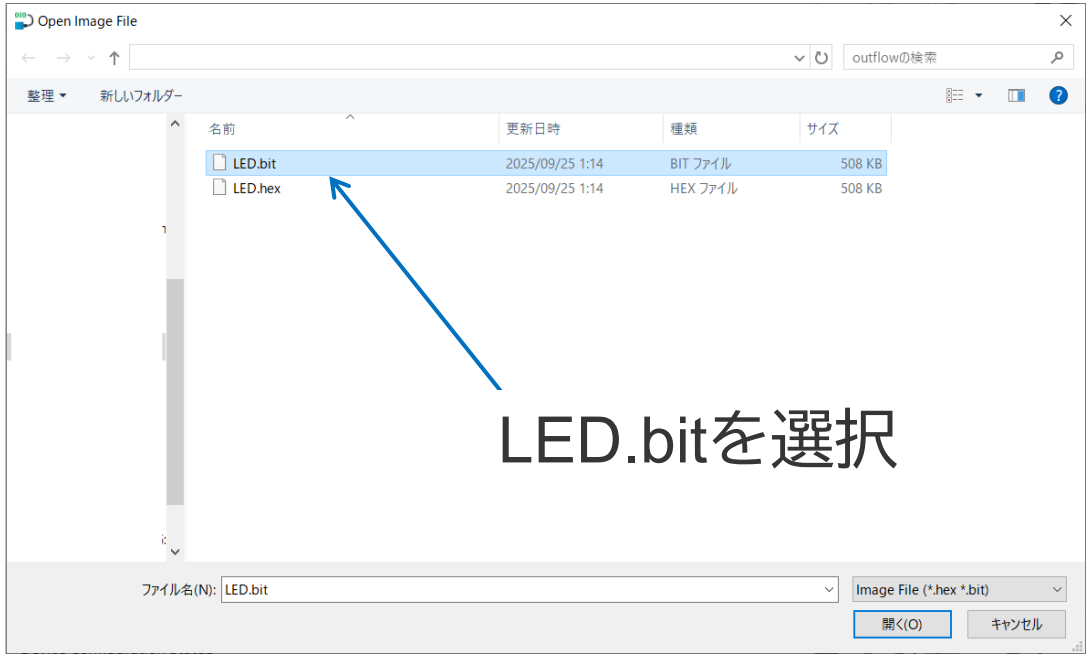
クリックしてダウンロードソフト
(Efinity Programmer)を起動

書き込みの設定



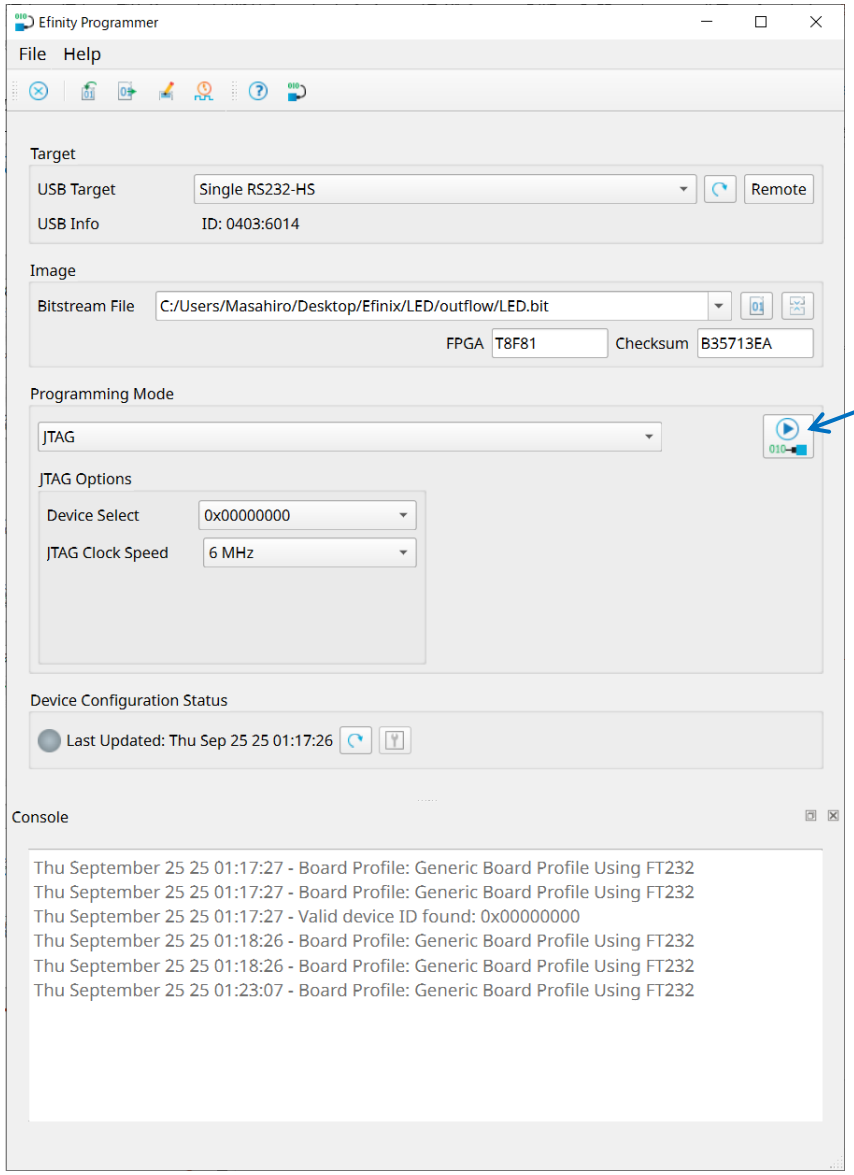
ビットストリーム
ファイルの選択

書き込みモードに
JTAGを選択
（ファイル選択前に）

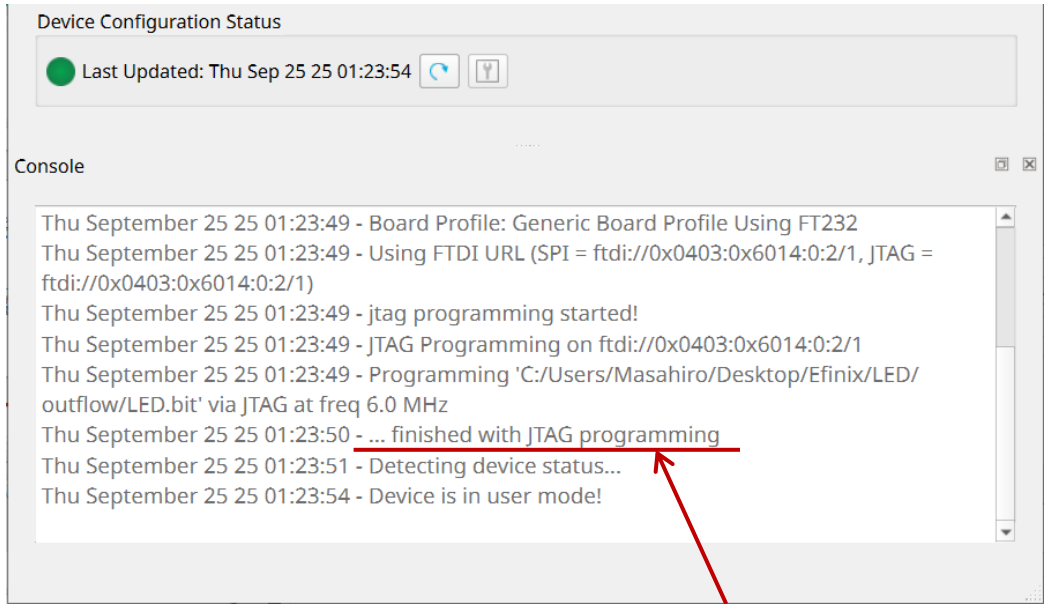


LED.bitを選択

書き込み



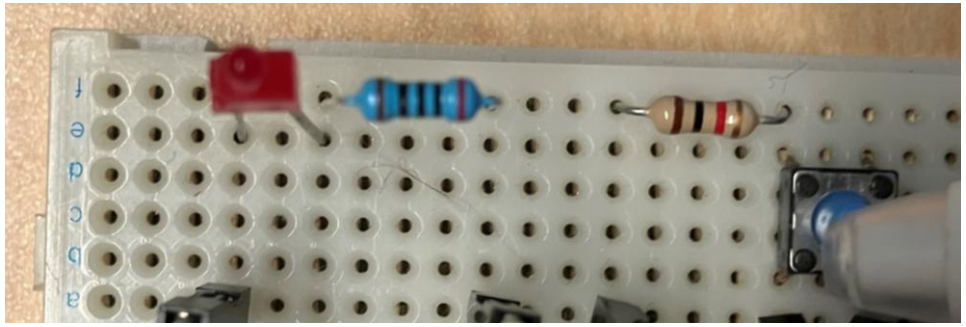
アイコンをクリックして書き込み

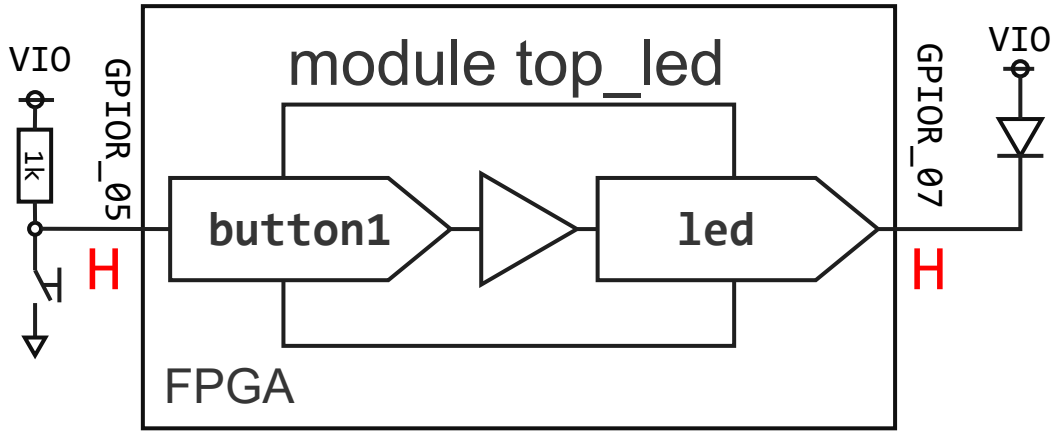


これが出ればOK

動作

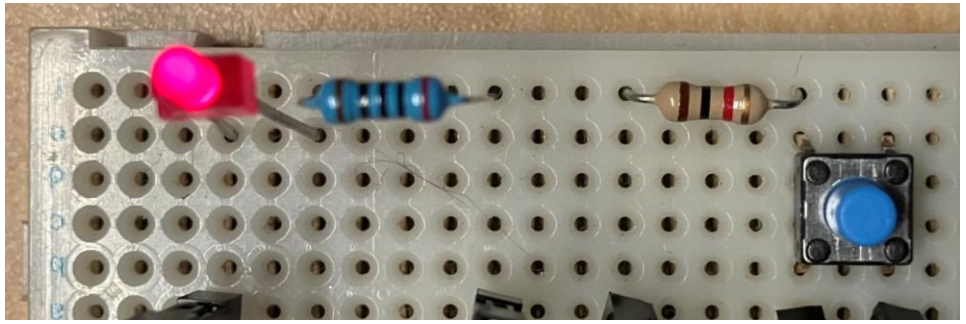
ボタンを押していない状態

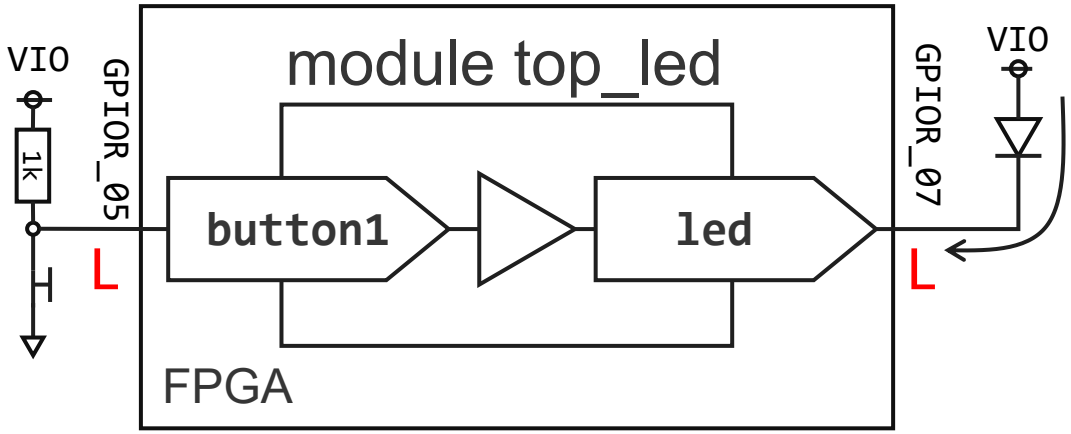




- 入力:プルアップ抵抗を介してH論理が入る
- 出力:HなのでLED両端に電位が生じず電流が流れないので消灯

ボタンを押した状態



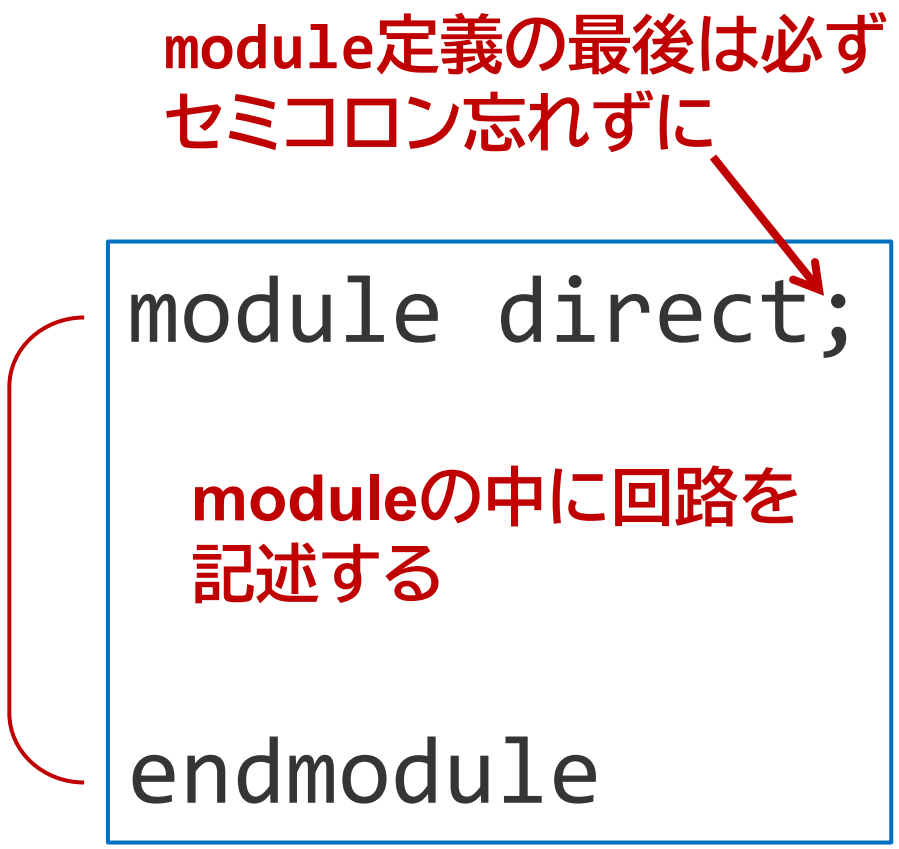


- 入力:スイッチを介してL論理が入る
- 出力:LなのでLED両端に電位が生じる。電流が流れることで点灯

入力・出力ともに負論理なので注意する

Verilog解説その1

Verilogコードの説明



- ファイルの中には**必ず1つ以上のモジュール**があります。
- モジュールは`module ~ endmodule`の範囲です。
- 回路は`module ~ endmodule`の範囲に書きます。
- `module`のあとにはモジュール名が入ります。モジュール名には算術演算子、スペース、その他予約語は入れられません。
- モジュール定義の終わりは**セミコロン(;)**が必要です。

Verilogコードの説明

```
module direct(
    input wire a,
    output wire b
    );

endmodule
```

ここにセミコロンを忘れない

- モジュール名のあとには回路の**入力と出力を定義**します。
- 入力は**input**、出力は**output**指示をつけます。
- **wire**は配線であることを意味します。
- 入出力にはそれぞれを区別する**ラベル**が必要です。
- ラベルを区切るために,(カンマ)が必要です。(改行は必須ではありません)
- モジュール定義のセミコロンはカッコの外に配置します。

Verilogコードの説明

```
module direct(
    input wire a,
    output wire b
    );
  ここのセミコロンを忘れない
assign b = a;


endmodule
```

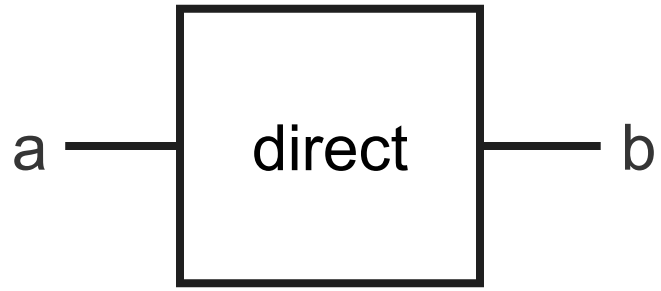
- ラベルaとラベルbの関係を結びつける方法の1つがassign文です。
  - 組み合わせ回路の記述方法の1つです
- assignのあとに
  結びつける対象 = 結びつけるルール
  の順に記述します。
- **配線aを配線bに接続するようなイメージ**です。
  代入ではありません。
- **文の最後には必ず「;」(セミコロン)が必要です。**

assignは配線をつなぐイメージ

初めてのVerilogの動作

```
module direct(
    input wire a,
    output wire b
    );

assign b = a;

endmodule
```

これまでの説明から左の回路Verilogは次の動作になる。



真理値表

入力a	出力b
0	0
1	1

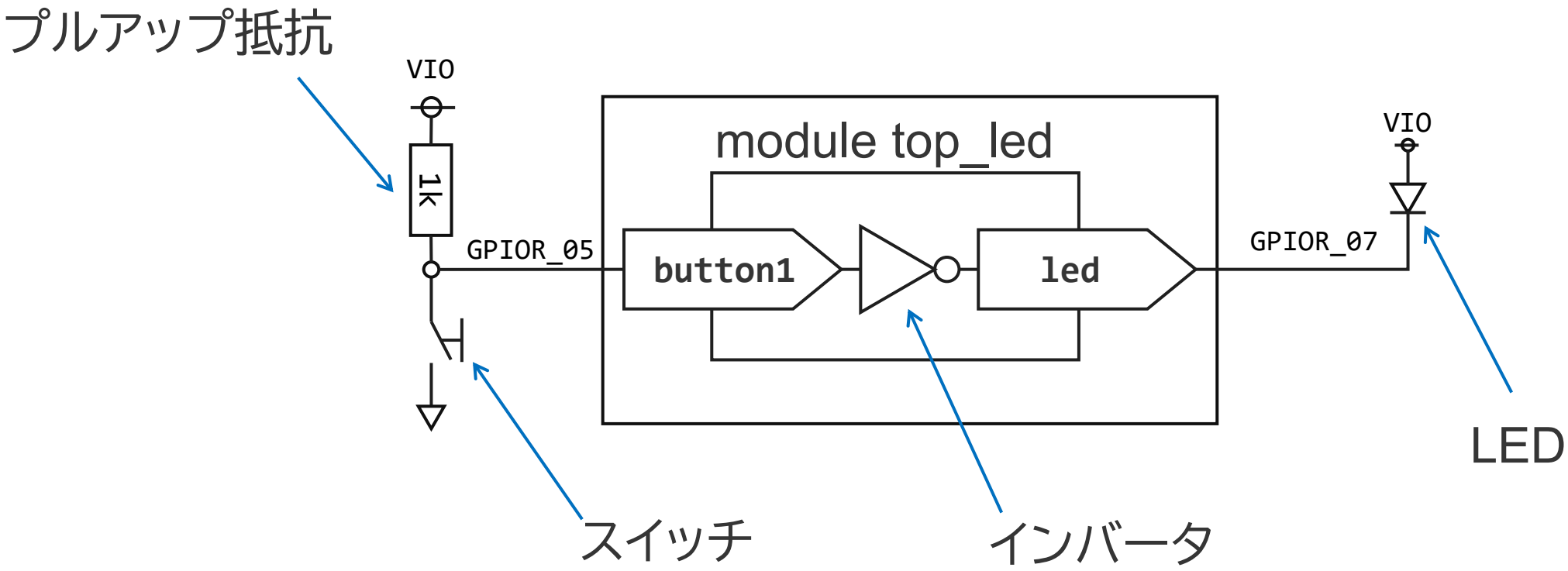
bはaをそのまま出力するので、

- **入力0**のときは**出力0**
- **入力1**のときは**出力1**

となる。

いろいろな演算子の回路

NOTゲートの動作確認



インバータのコード

top_ledのときと同様にFile -> New Fileからファイルを作成。ファイル名はmy_inv。

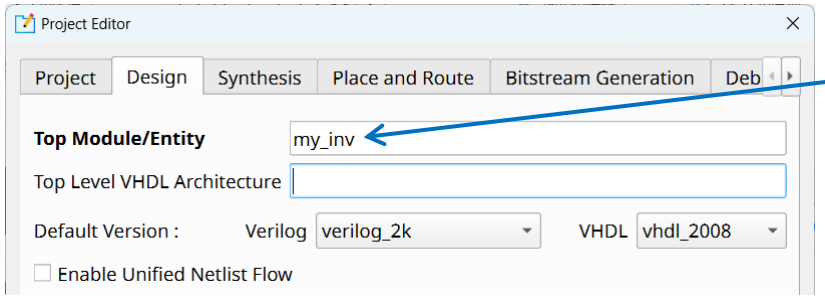
my_inv.v　top_led.v

```
1  module my_inv(
2  input wire button,
3  output wire led);
4
5  assign led = ~button;
6
7  endmodule
```

モジュール名はmy_invに

NOTゲートは~(チルダ)を用いて表す。

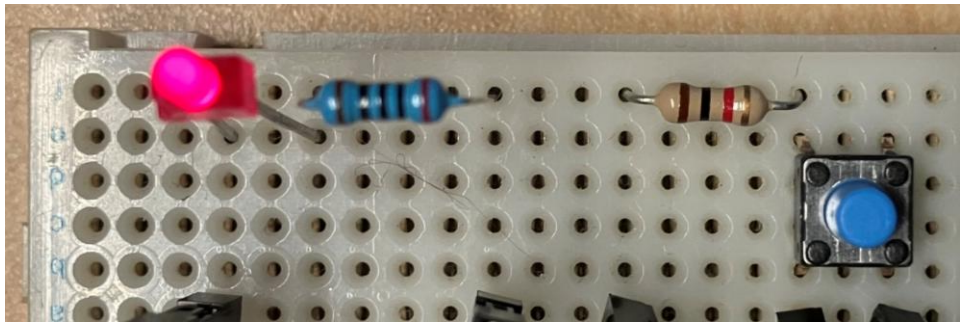
File -> Edit projectからProject Editorを開き、Designタブ内の
Top Module/Entityをmy_invに変更

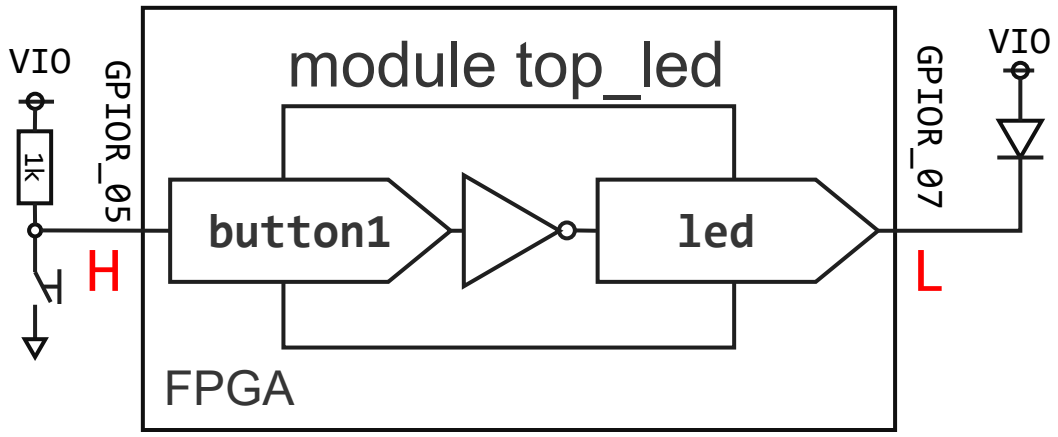


my_invに変更

インバータの動作

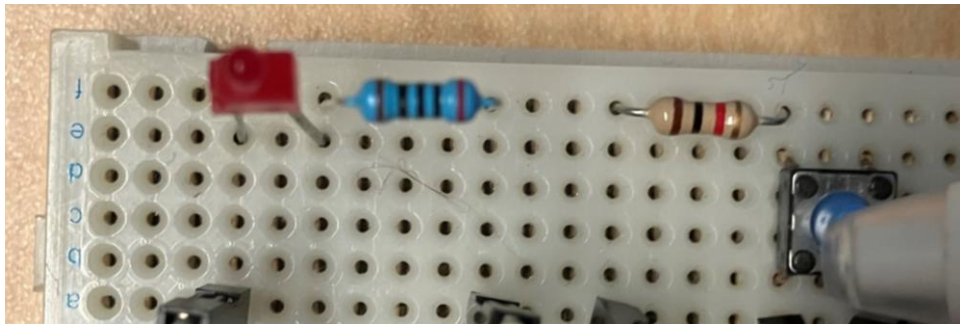
ボタンを押していない状態

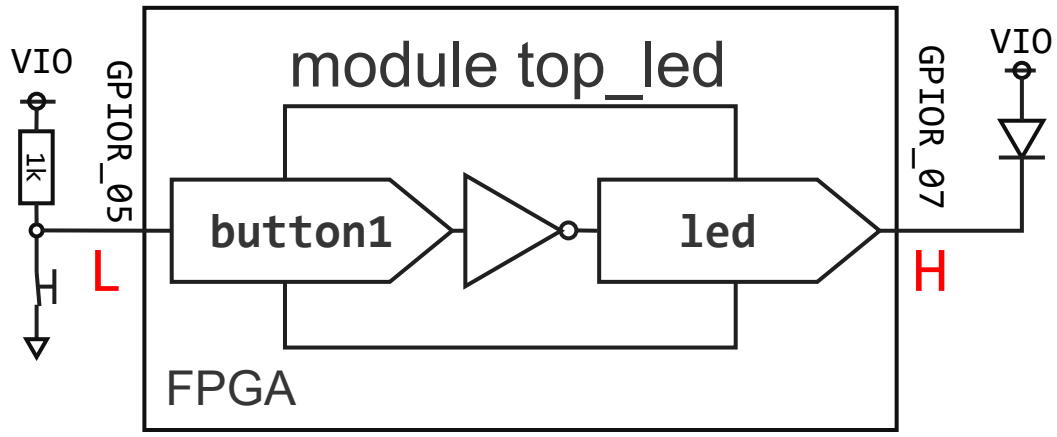




- 入力:プルアップ抵抗を介してH論理が入る
- 出力: LなのでLED両端に電位が生じる。電流が流れることで点灯

ボタンを押した状態

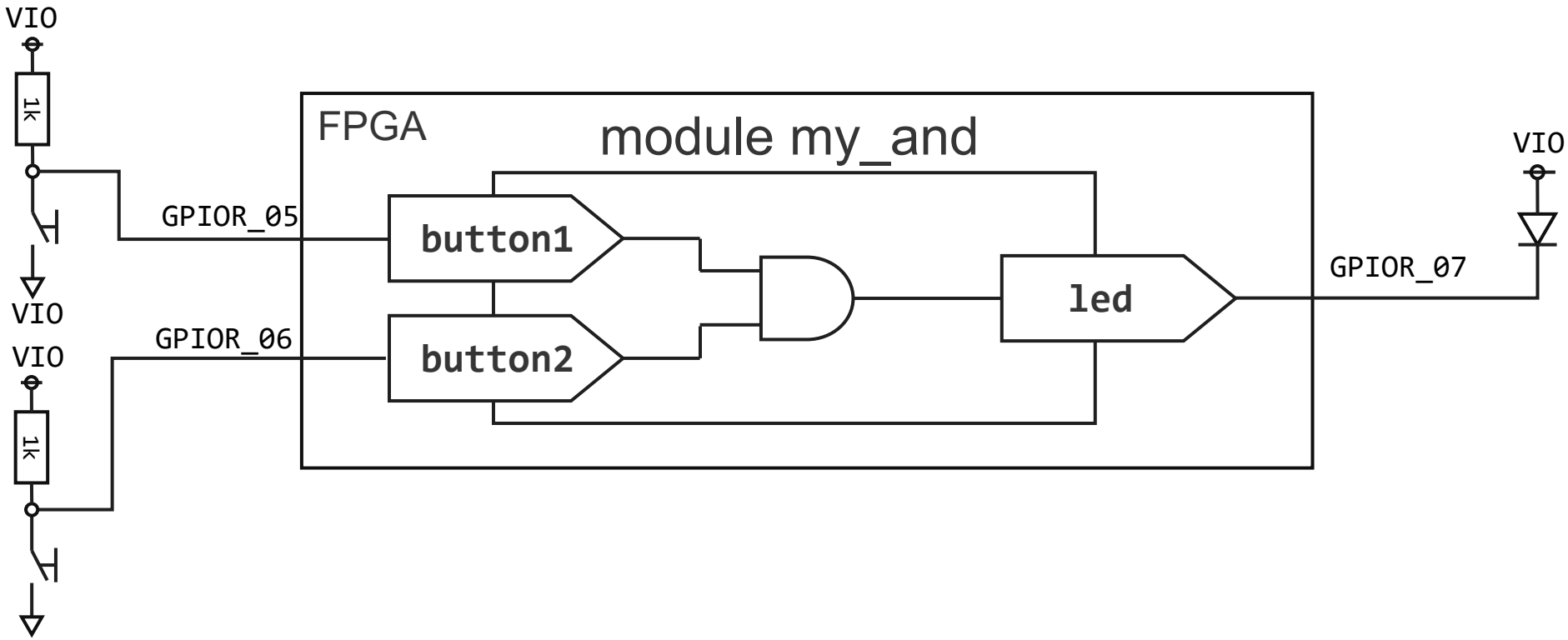




- 入力:スイッチを介してL論理が入る
- 出力:HなのでLED両端に電位が生じず電流が流れないので消灯

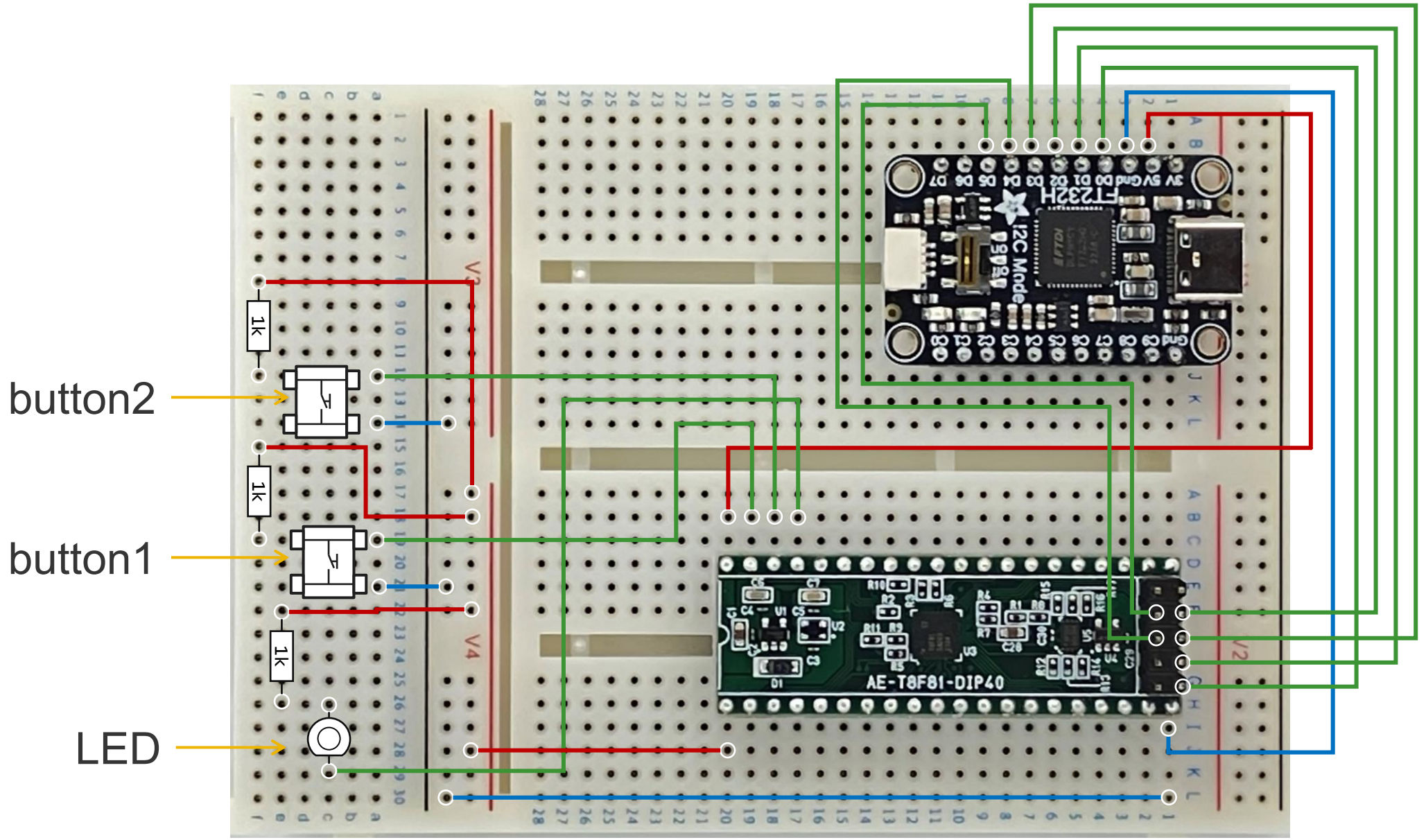
ANDゲートの動作確認

2入力ANDゲートとして動作する回路をFPGAに実装する。
入力として2つのボタンと1つのLEDを有する。



2個のボタンはGPIOR_05とGPIOR_06、LEDはGPIOR_07に接続する。

配線図



Verilogコードとピンアサイン

※Top Module/Entitiyをmy_andに変更するのを忘れずに

Code Editor

my_and.v

```
1  module my_and(
2  input wire button1,
3  input wire button2,
4  output wire led);
5
6  assign led = button1 & button2;
7
8  endmodule
9
```

モジュール名はmy_and

ANDゲートは&を用いて表す。

GPIO : Instance View

			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button1	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
button2	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
led	C7	GPIOR_07	2A	None	None	R1	GPIOR_07

ANDゲートの動作

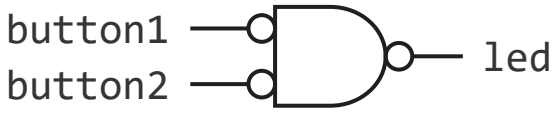
どちらのスイッチを押してもLEDが点灯するORのような動作をしていると思います。

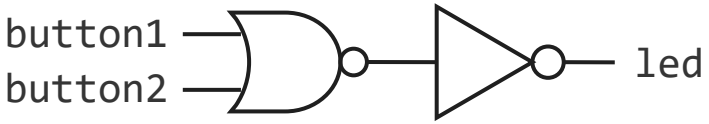
button1の状態	button2の状態	button1ピンの入力論理	button2ピンの入力論理	ledピンの出力論理	ledの状態
離す	離す	H	H	H	消灯
押す	離す	L	H	L	点灯
離す	押す	H	L	L	点灯
押す	押す	L	L	L	点灯

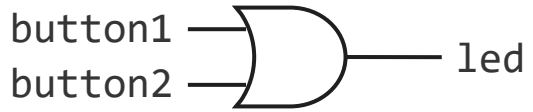
- ボタンを押したときにピンがL
- 出力がLのときにLEDが点灯

→ 入出力がLのときに動作
→ **負論理**という

なぜ入出力両方が負論理のときにANDはORになるのでしょうか？







$$led = \overline{\overline{button1}\ \&\ \overline{button2}}$$

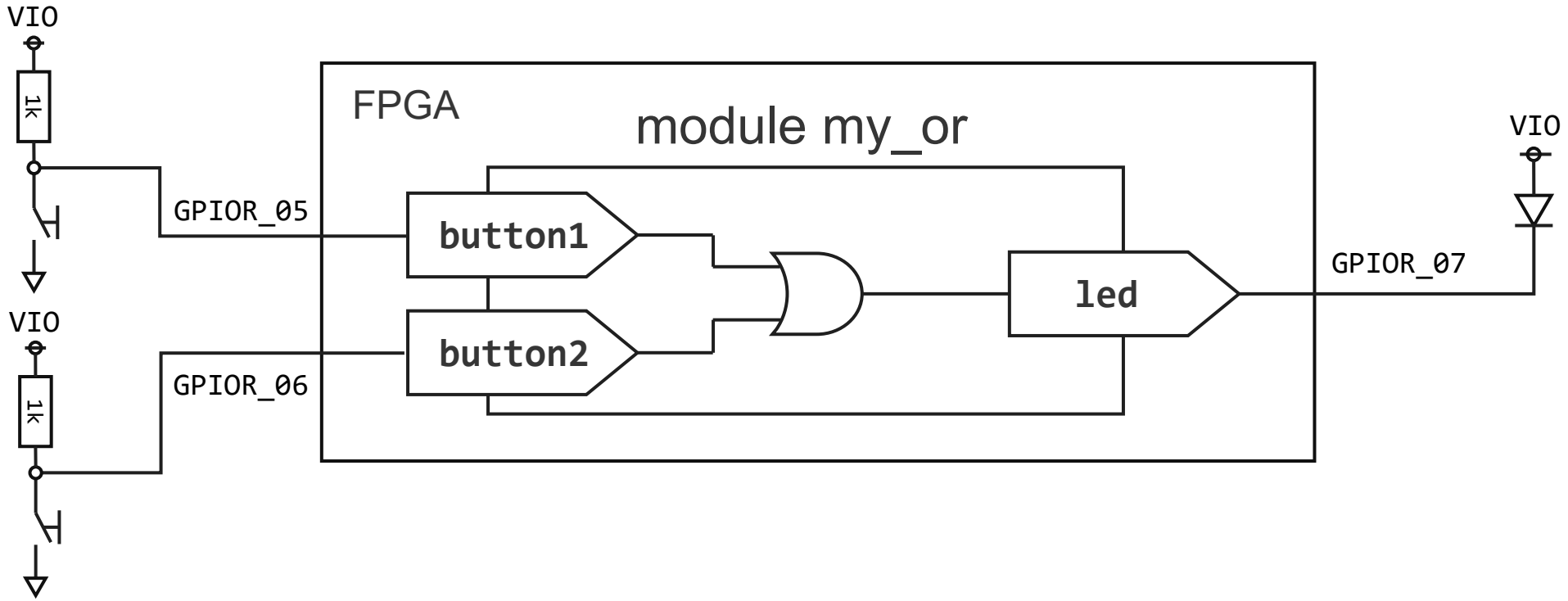
$$led = \overline{\overline{button1} \mid \overline{button2}}$$

$$led = button1 \mid button2$$

負論理のANDはこのように変形されるので、結果としてORのように振る舞います。

ORゲートの確認回路

2入力ORゲートとして動作する回路をFPGAに実装する。
入力として2つのボタンと1つのLEDを有する。



2個のボタンはGPIOR_05とGPIOR_06、LEDはGPIOR_07に接続する。
配線はANDゲートの実験と同じものを使用する。

Verilogコードとピンアサイン

※Top Module/Entitiyをmy_orに変更するのを忘れずに

Code Editor

my_or.v

```
1  module my_or(
2  input wire button1,
3  input wire button2,
4  output wire led);
5
6  assign led = button1 | button2;
7
8  endmodule
9
```

モジュール名はmy_orに

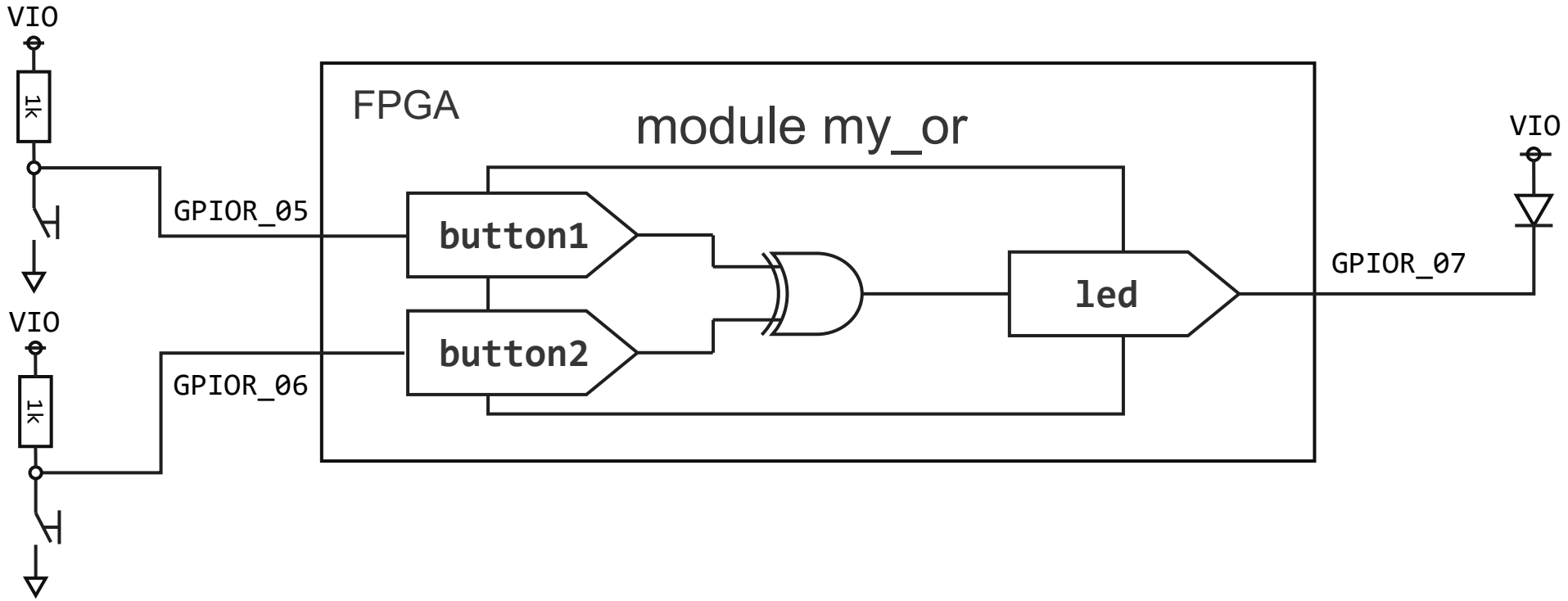
ORゲートは|を用いて表す。

GPIO : Instance View

			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button1	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
button2	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
led	C7	GPIOR_07	2A	None	None	R1	GPIOR_07

ExORゲートの確認回路

2入力ExORゲートとして動作する回路をFPGAに実装する。
入力として2つのボタンと1つのLEDを有する。



2個のボタンはGPIOR_05とGPIOR_06、LEDはGPIOR_07に接続する。
配線はANDゲートの実験と同じものを使用する。

Verilogコードとピンアサイン

※Top Module/Entitiyをmy_xorに変更するのを忘れずに

Code Editor

my_xor.v

```
1  module my_xor(
2  input wire button1,
3  input wire button2,
4  output wire led);
5
6  assign led = button1 ^ button2;
7
8  endmodule
9
```

モジュール名はmy_xorに

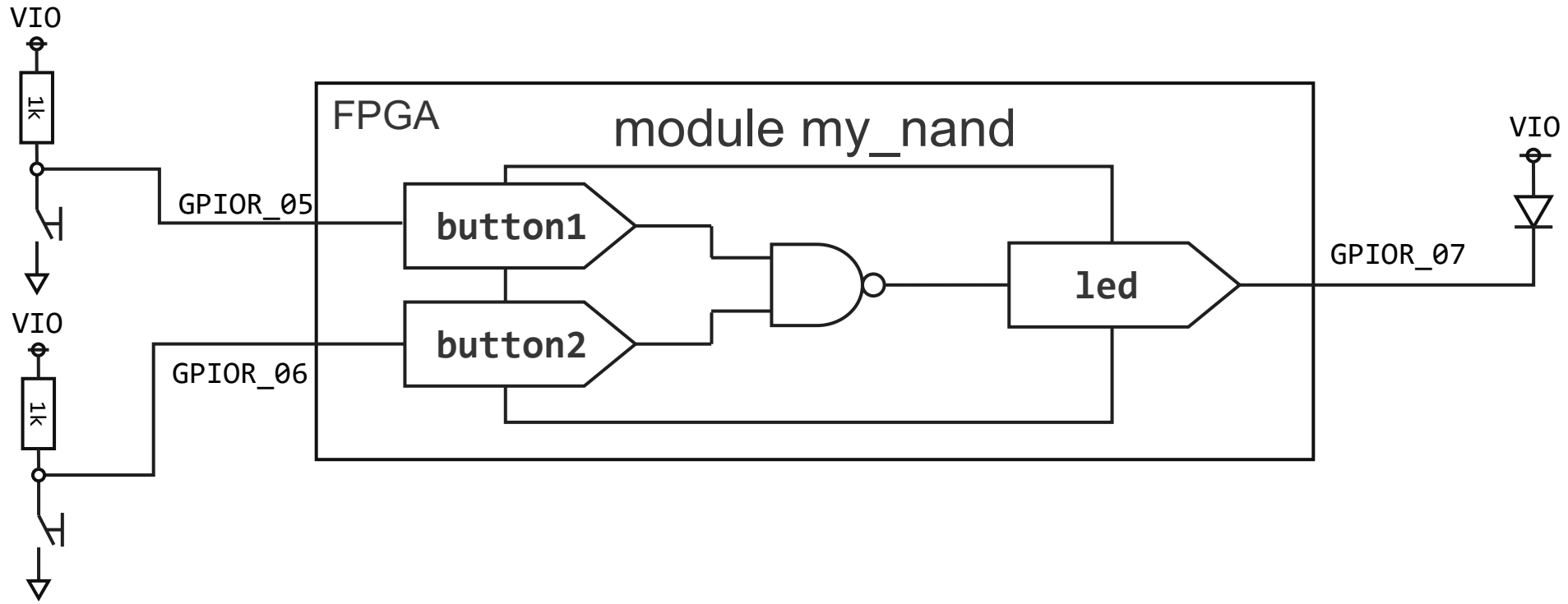
ORゲートは^(ハット)を用いて表す。

GPIO : Instance View

			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button1	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
button2	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
led	C7	GPIOR_07	2A	None	None	R1	GPIOR_07

NANDゲートの確認回路

2入力NANDゲートとして動作する回路をFPGAに実装する。
入力として2つのボタンと1つのLEDを有する。



2個のボタンはGPIOR_05とGPIOR_06、LEDはGPIOR_07に接続する。
配線はANDゲートの実験と同じものを使用する。

Verilogコードとピンアサイン

※Top Module/Entitiyをmy_nandに変更するのを忘れずに

Code Editor

my_or.v | my_nand.v

```
1  module my_nand(
2  input wire button1,
3  input wire button2,
4  output wire led);
5
6  assign led = ~(button1 & button2);
7
8  endmodule
9
```

モジュール名はmy_nandに

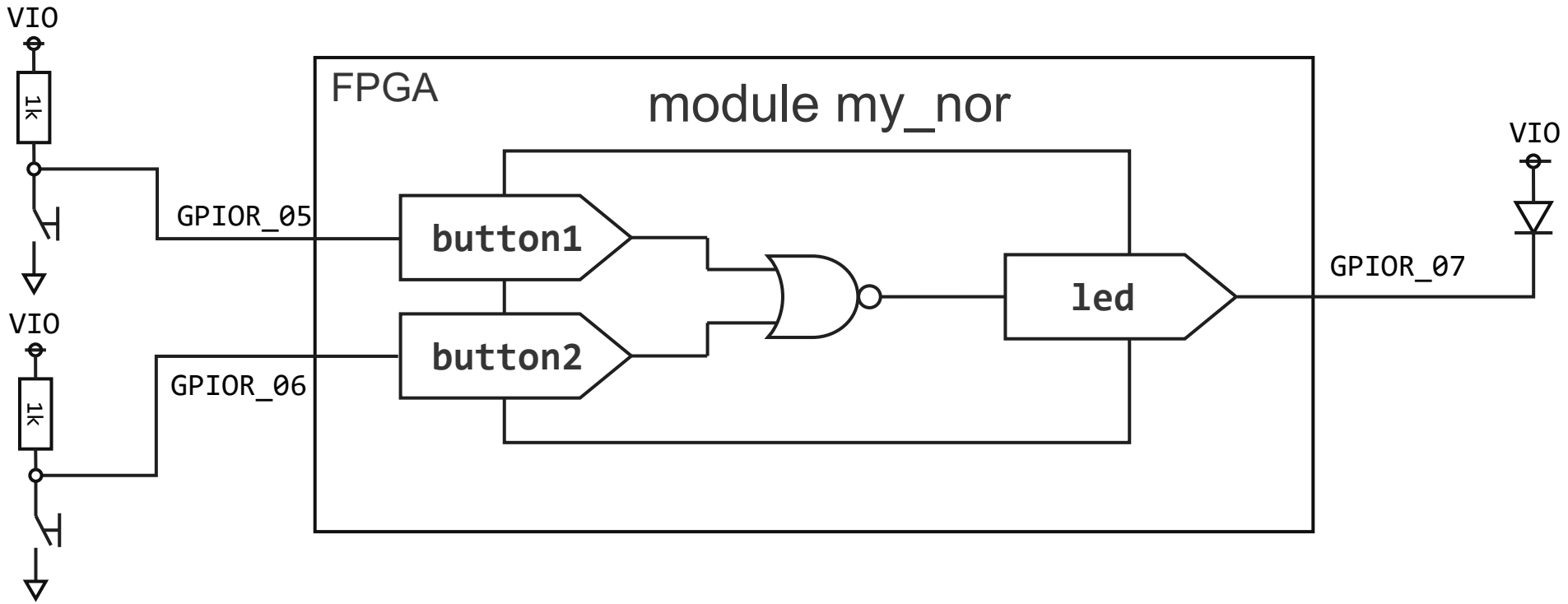
NANDゲートは()で囲った&を論理反転することで表す。

GPIO : Instance View

			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button1	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
button2	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
led	C7	GPIOR_07	2A	None	None	R1	GPIOR_07

NORゲートの確認回路

2入力ORゲートとして動作する回路をFPGAに実装する。
入力として2つのボタンと1つのLEDを有する。



2個のボタンはGPIOR_05とGPIOR_06、LEDはGPIOR_07に接続する。
配線はANDゲートの実験と同じものを使用する。

Verilogコードとピンアサイン

※Top Module/Entitiyをmy_norに変更するのを忘れずに

Code Editor

my_nor.v

```
1  module my_nor(
2  input wire button1,
3  input wire button2,
4  output wire led);
5
6  assign led = ~(button1 | button2);
7
8  endmodule
```

モジュール名はmy_norに

NORゲートは()で囲った|を論理反転することで表す。

GPIO : Instance View

			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
button1	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
button2	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
led	C7	GPIOR_07	2A	None	None	R1	GPIOR_07

Verilog解説その2

NOT

Verilogでは論理反転(NOT)は「~」(チルダ)を使用する。
論理反転したい信号ラベルの前に~を配置する。

NOT回路のVerilogコード

```
module my_not(
   input wire a,
   output wire b
);
   assign b = ~a;
endmodule
```

AND

Verilogでは論理積(AND)は「＆」を使用する。
論理積をとる2つの信号ラベルの間に＆を配置する。

AND回路のVerilogコード

```
module my_and(
  input wire a,
  input wire b,
  output wire c);

  assign c = a & b;
endmodule
```

入力を2つに増やすには、
inputを追加する。
ラベルはユニーク(一意)になるようにする

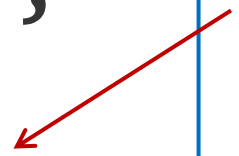
aとbを論理積を求める

OR

Verilogでは論理和(OR)は「|」を使用する。
論理和をとる2つの信号ラベルの間に|を配置する。

OR回路のVerilogコード

```
module my_or(
  input wire a,
  input wire b,
  output wire c);


  assign c = a | b;
endmodule
```



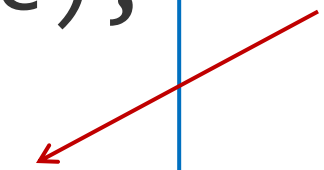
aとbを論理和を求める

ExOR

Verilogでは排他的論理和は「^」を使用する。
排他的論理和をとる2つの信号ラベルの間に^を配置する。

ExOR回路のVerilogコード

```
module my_exor(
  input wire a,
  input wire b,
  output wire c);


  assign c = a^b;
endmodule
```



aとbを排他的論理和を求める

NAND

Verilogでは**否定論理積(NAND) を直接計算する演算子は無い。**
そこで論理反転と論理積を組み合わせて実現する。

NAND回路のVerilogコード

```
module my_nand(
  input wire a,
  input wire b,
  output wire c);

  assign c = ~(a & b);
endmodule
```

aとbの論理積を求めてから、
その結果を反転する。
カッコ内は優先して演算される。

ここまでの演算子まとめ

- 単項演算子：オペランド（対象）が1つのみの演算子
  - ～：NOT
- 二項演算子：オペランド（対象）が2つの演算子
  - &　：AND
  - |　：OR
  - ^　：Exclusive OR

便利なVerilog文法

wire

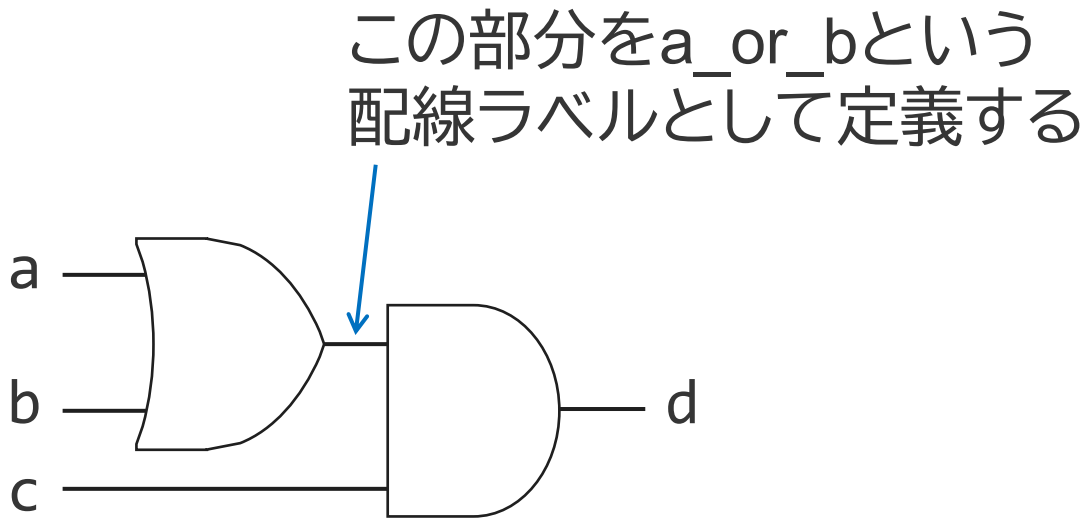
wire構文を用いることで新たに配線ラベルを定義できる。wire構文は、

```
wire wire_name;
```

のように記述する。wireの後ろには、配線名をつける。これで、wire_nameという配線が新たに定義された。

wireは複雑な組み合わせ回路で使用できる。

```
wire a_or_b;
assign a_or_b = a|b;
assign d = a_or_b & c;
```



組み合わせ回路

Code Editor

complex.v

```
1  module complex(
2  input wire a,
3  input wire b,
4  input wire c,
5  output wire d);
6
7  wire a_or_b;
8  assign a_or_b = a | b;
9  assign d = a_or_b & c;
10
11 endmodule
12
```

色がうまくつかないですが、
多分文法チェッカーがバグっています。

入力3個、出力1個となるように定義してください。
接続箇所はp.38を参照してください。

GPIO : Instance View

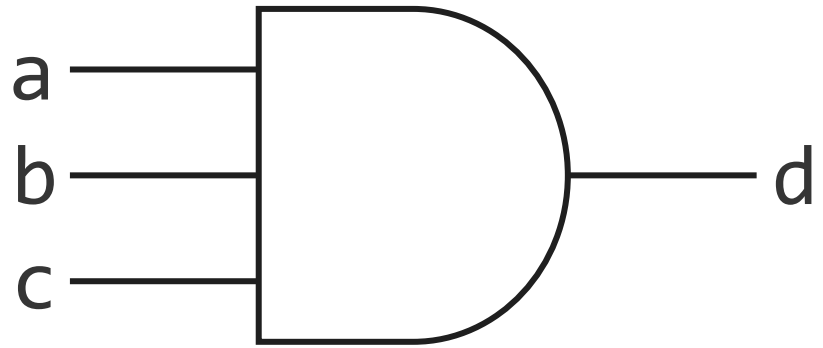
			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
a	B6	GPIOR_05	2A	None	None	R1	GPIOR_05
b	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
c	C7	GPIOR_07	2A	None	None	R1	GPIOR_07
d	A8	GPIOR_08	2A	None	None	R1	GPIOR_08

今日の練習課題

3入力の論理積回路

3入力AND回路のVerilogコード

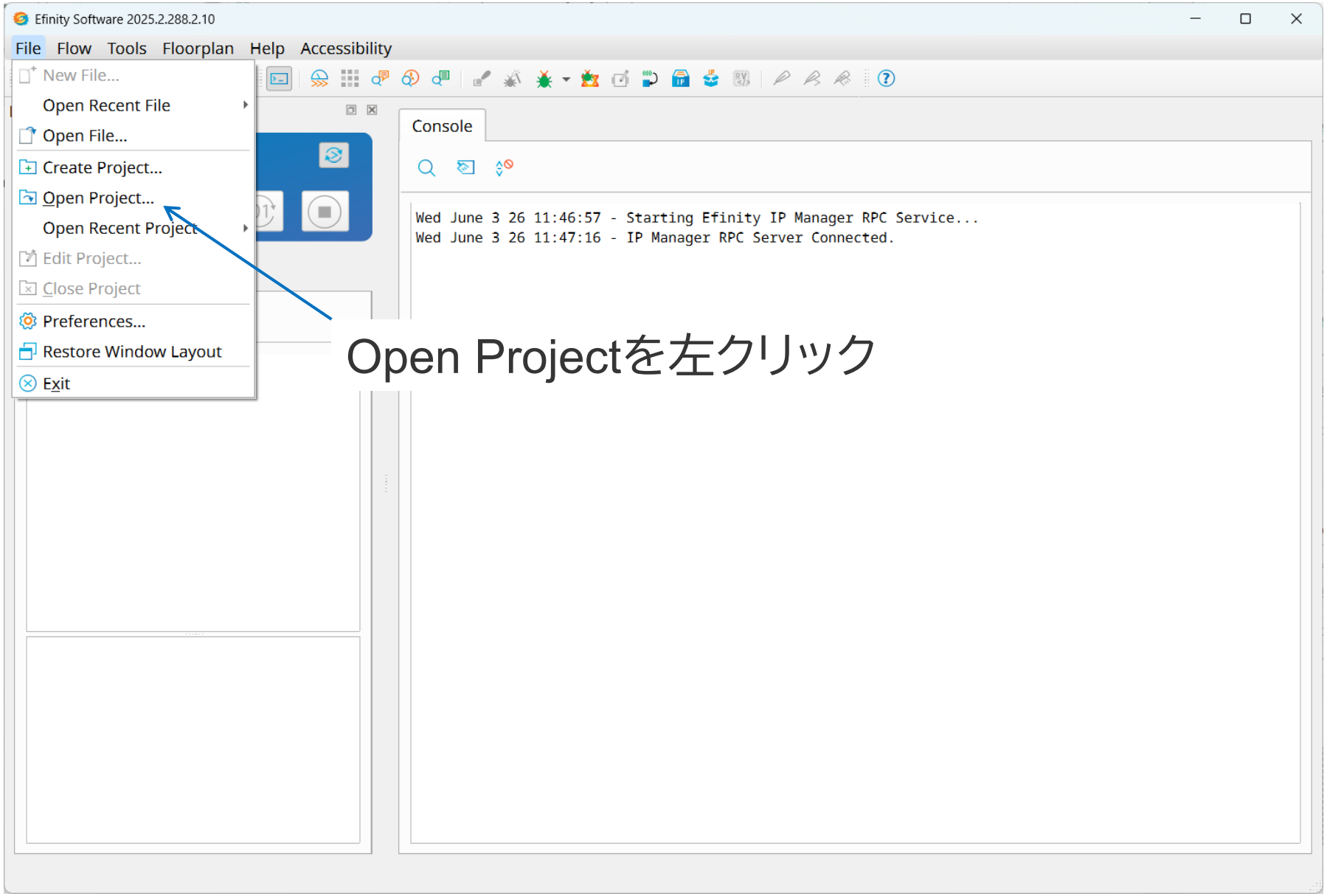
```
module three_and(
    ここに自力でコードを書く
    );

    assign d = ここに自力でコードを書く
endmodule
```



まとめ

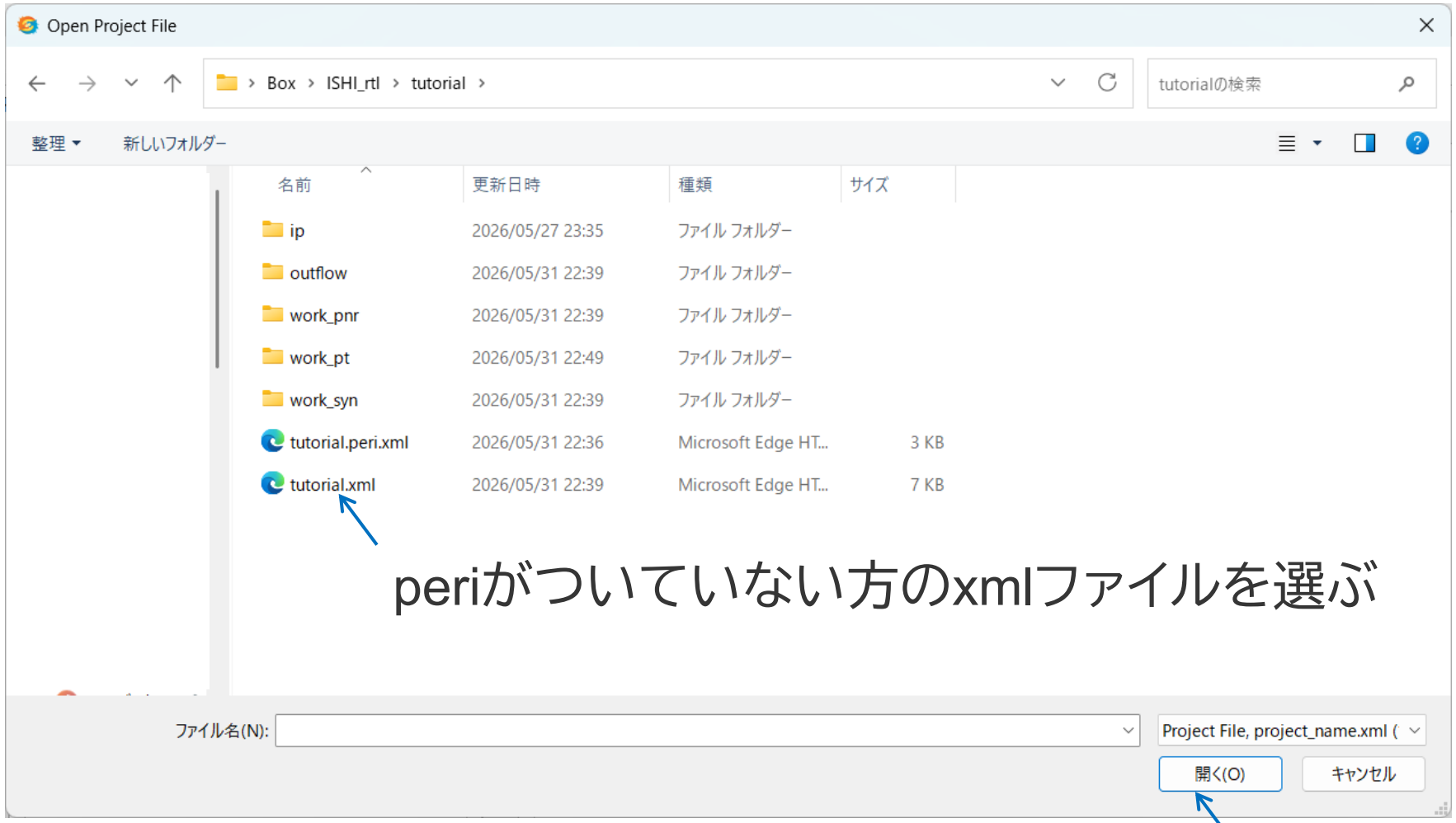
- EfinixのFPGA開発方法を説明しました。
- Verilogの基本的な文法を紹介しました。
  - 今日の文法
    - モジュール定義
    - 基本的な論理演算子
    - wire指定子の使い方

補足｜プロジェクトの開き方



補足｜プロジェクトの開き方

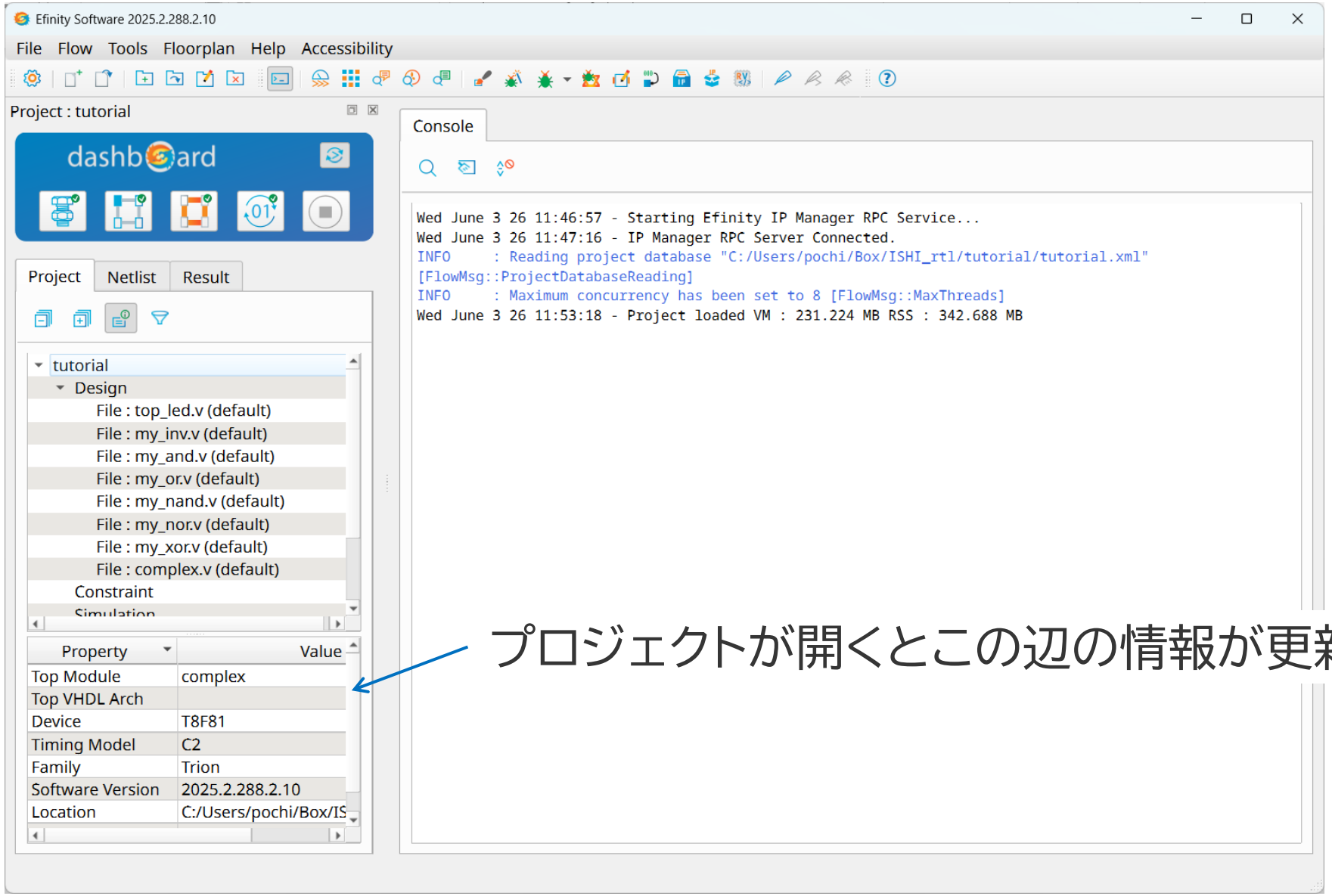
プロジェクトのフォルダに移動する。



periがついていない方のxmlファイルを選ぶ

選択したら開くをクリック

補足｜プロジェクトの開き方



プロジェクトが開くとこの辺の情報が更新される