FPGA実践的設計法ーツール編

2006年8月31日

内田智久 KEK

このセミナーについて

対象者：
これからFPGA設計を始めようとしている人

何故このセミナーを企画したのか

問題点：
参考書で学んでもFPGAが使えない

参考書は部分的に書かれている
HDLのみ、ツールのみ
　紹介が多い（初め使わない機能が多い）
　過度に簡単化した例題

理由：
設計から実装まで一貫した（ツールの）説明が無い
最小限何を知っていれば良いか分からない

目標

最小限の知識を設計フロー全体で説明する

参考書の内容をFPGAに実装できる

HDL記述方法は各人じっくり勉強してください

続編の開催希望がありましたら連絡下さい

受講者から実践的で具体的な質問や相談が来る事を願っています

セミナーの内容

- 組み合わせ回路
- 同期回路
- 論理シミュレーション
- FPGAの構造
- 設計時の注意

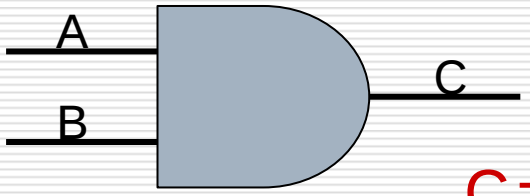
セミナーの内容

- 付録
  - MCSファイルの生成
  - デバイス依存ライブラリの使用方法
  - タイミング制約
  - XILINXライブラリの使い方

Field Programable Gate Array (FPGA)とは

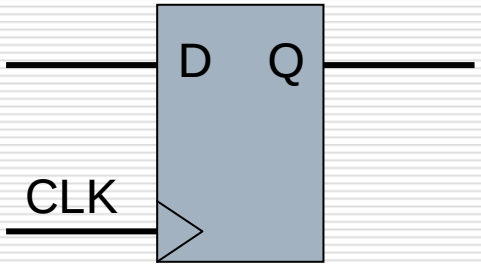
- 柔らかいハードウエア
  - 主としてデジタル（論理）回路
  - 最近はアナログ回路もできるようになっている
- 論理回路
  - ソフトウエアの様にダウンロードして使用する
  - 通常回路情報を格納するROMが必要
    - 電源投入時に自動的に動作開始させる為
    - 毎回手動でダウンロードする時は必要ない

論理回路　その4つの構成要素

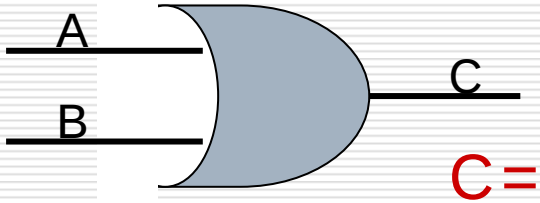


AND

C＝A&B

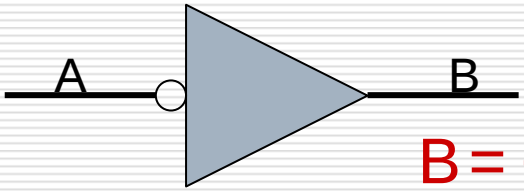


D-フリップ・フロップ (DFF)



OR

C＝A|B



NOT

B＝~A

全てのデジタル回路は
この4つの要素（回路）のみで構成されている

CPU，DSPなど一見複雑な回路もこの4つの組み合わせで設計可能

とりあえずやってみる

ツールを使ってみましょう！

SWがONの間LEDが光る回路

- 組み合わせ回路の記述方法
- ISEの使い方

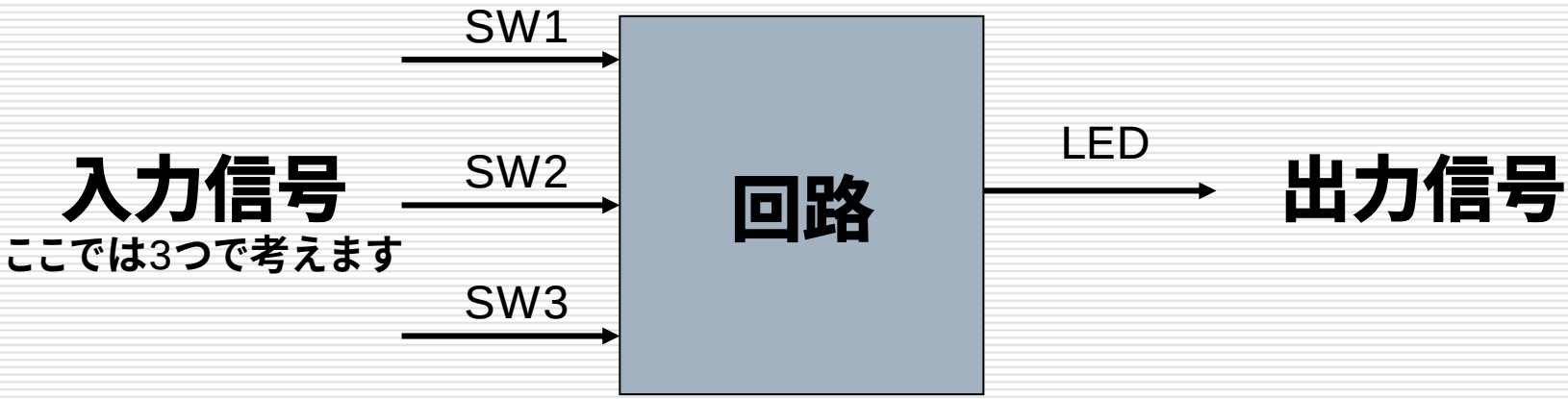
先ずはシミュレーション無しでやってみる

SWがONの間LEDが光る回路

- **使用部品**
  - SW3，SW2，SW1，SW0（**スライドスイッチ**）
  - LD0（LED）
- **仕様**
  - **一つのSWだけがONの時LEDを点灯**
  - **二つ以上のSWがONの時は消灯**

組み合わせ回路とは

AND，OR，NOTの組み合わせで構成



一つの検出器が検出した時のみ出力など

条件成立を抽出

する（したい）時に使用する

どのように考えるか？

- どんな時に出力したいか、と考えてみる
- 例えば
  - SW0のみON時、LEDをONしたい
  - LED＝（ONする時の入力状態を書く）
  - LEDがONするのは
    - SW0＝ONで、さらに
    - SW1＝OFFで、さらに
    - SW2＝OFFの時

ANDの気持ち

- Verilogで書くと
- LED=SW0 & ~SW1 & ~SW2;

- **一つの条件を作る時にANDを使う**

- **ここに、SW1のみON時も、LEDをONしたい**
- LED=~SW0 & SW1 & ~SW2;

条件を足す

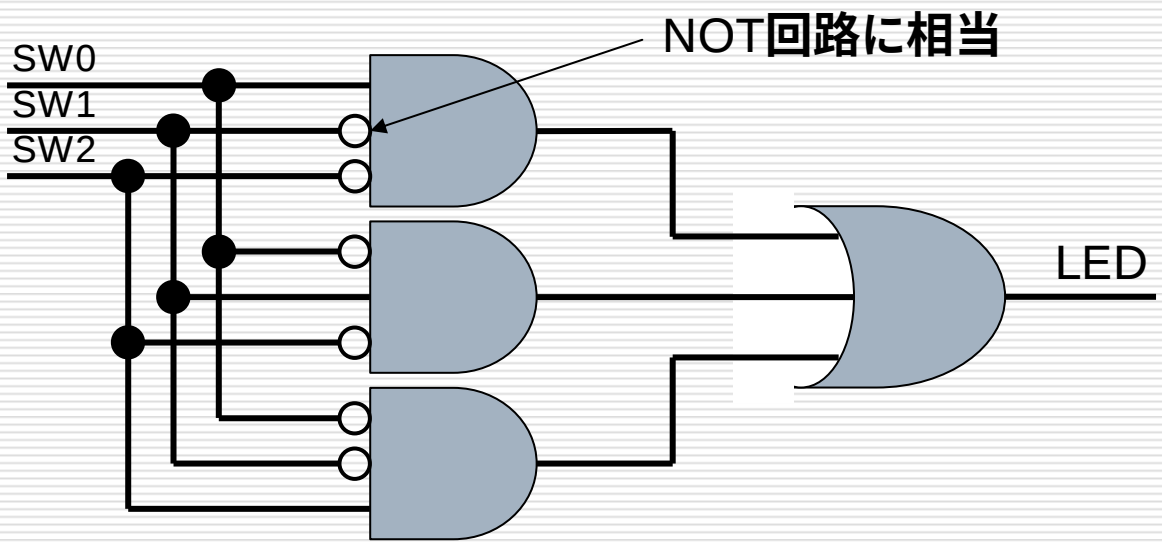
- SW1のみON時も、LEDをONするようにする
  - LED=~SW0 & SW1 & ~SW2;を足す
- LEDがONするのは
  - SW0のみONの時と
  - SW1のみONしている時
- 条件を足す時にORを使う
  - ～の時と～の時に出力をON

と言う事で、こうなります

```
LED =( SW0  & ~SW1 & ~SW2) |
     (~SW0 &  SW1  & ~SW2) |
     (~SW1 & ~SW1 &  SW2);
```

Verilogで書くと

簡単に書ける



回路図で書くと

大変！！

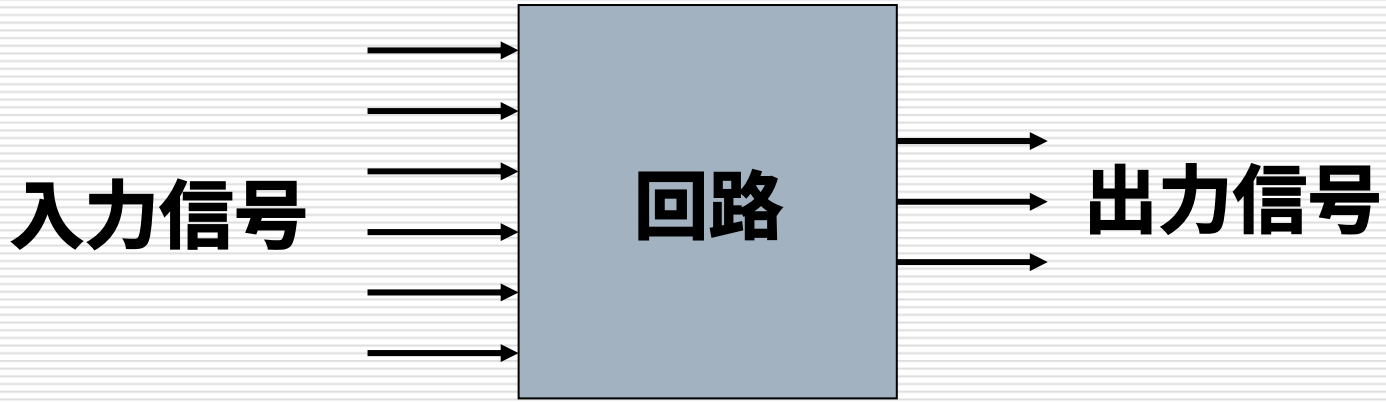
HDLは

- 設計作業を効率よく行うために誕生
- 効率よく入力できる事で
  - 入力間違いを少なく
  - 大規模回路が扱えるようになった
  - 読みやすい
- 他には
  - コードを仕様書に近づける

まとめ：組み合わせ回路

- 特定の入力パターンの時、出力をON
- ANDの気持ち
  - 特定の条件を作る：～時ON
- ORの気持ち
  - 条件を足す：～の時と～の時ON
- Verilog表記
  - AND：&
  - OR：|
  - NOT：~

結局、回路とは？



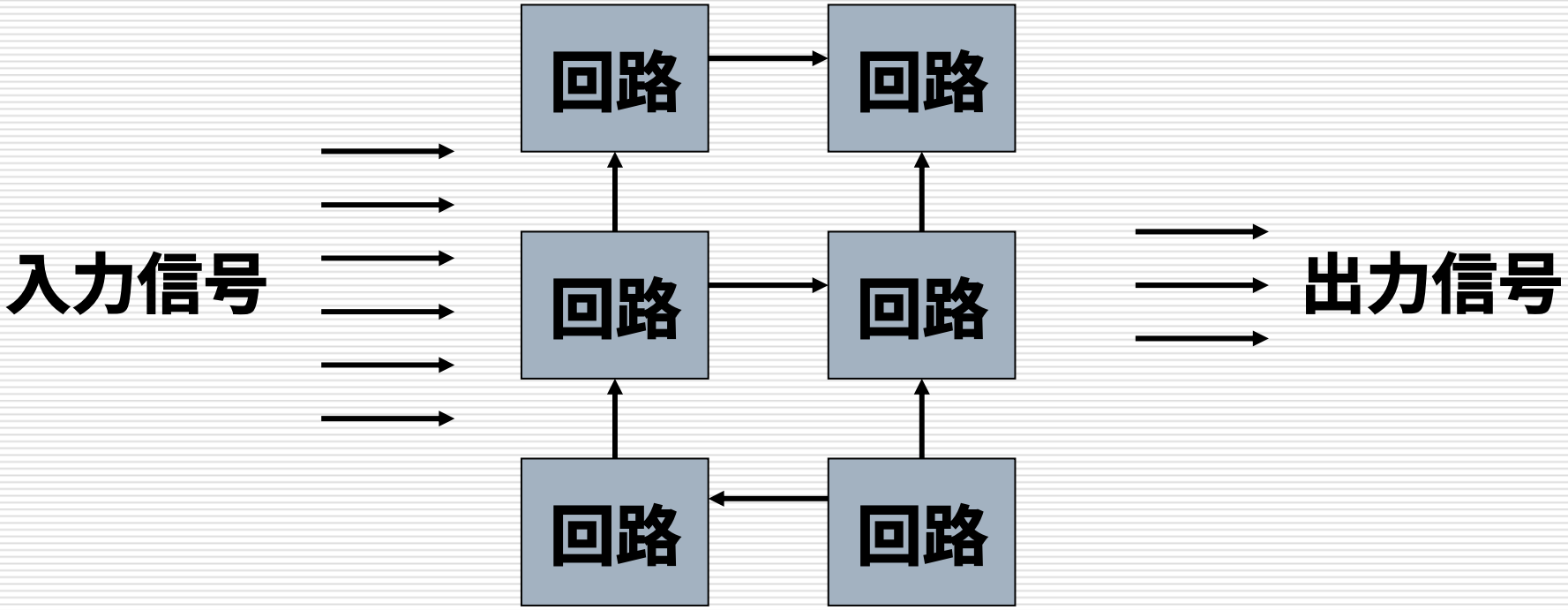
入力信号の組み合わせ＋現在の状態

決められた出力信号を出力

（これが設計）

回路設計とは

単純動作する箱に分割する事



単純動作する箱を上手く接続して
複雑な（希望の）動作をさせる

回路設計

やりたい事（仕様）を満たすように
単純動作する箱に分割する事

最終的には4つの要素で表現できるくらいに

- 機能（ブロック）分け
- その間のインタフェース　**非常に重要**
  - 何をやり取りするか

サンプル回路

- さっそくサンプル回路を見てみましょう

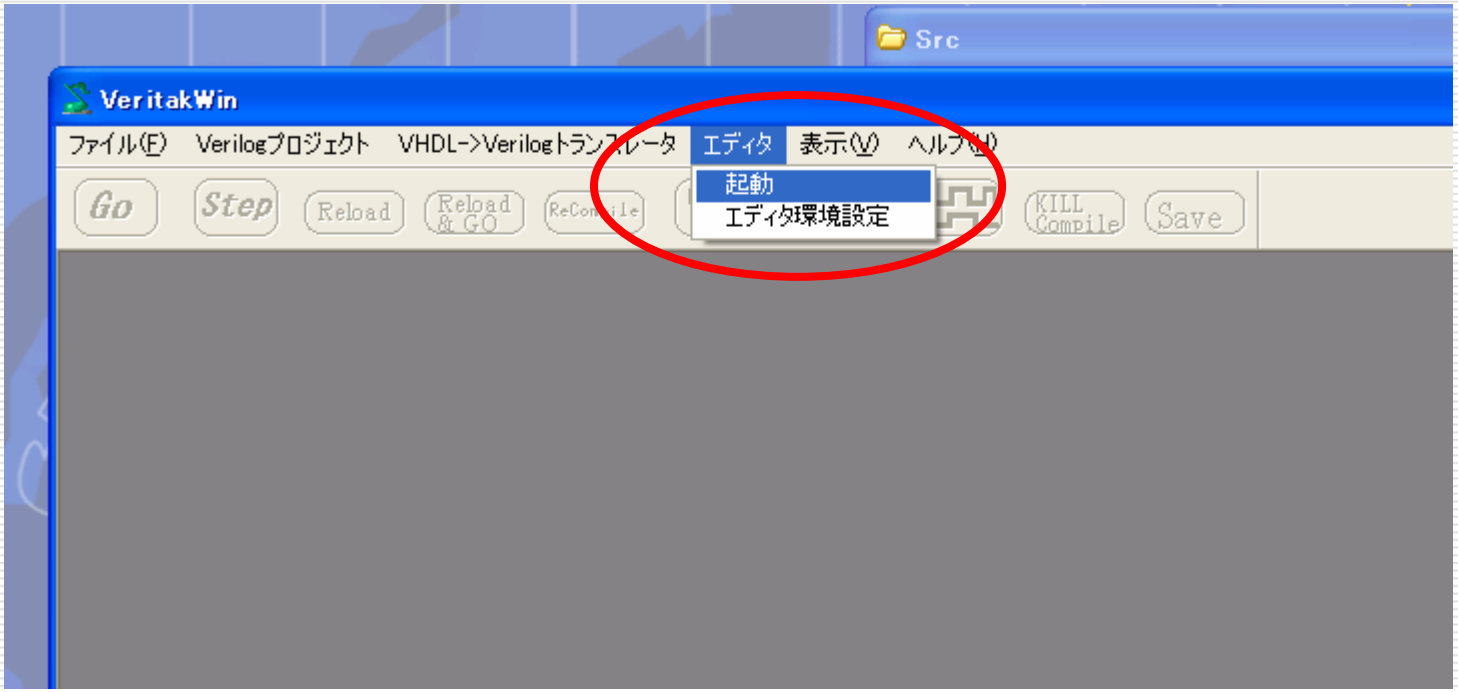
- 今後、作業ディレクトリをC:¥workとします。
- お渡ししたファイルをC:¥workとして展開してください

- Veritakのエディタを使ってみましょう
  - お気に入りのテキスト・エディタがある方はそれを使ってください

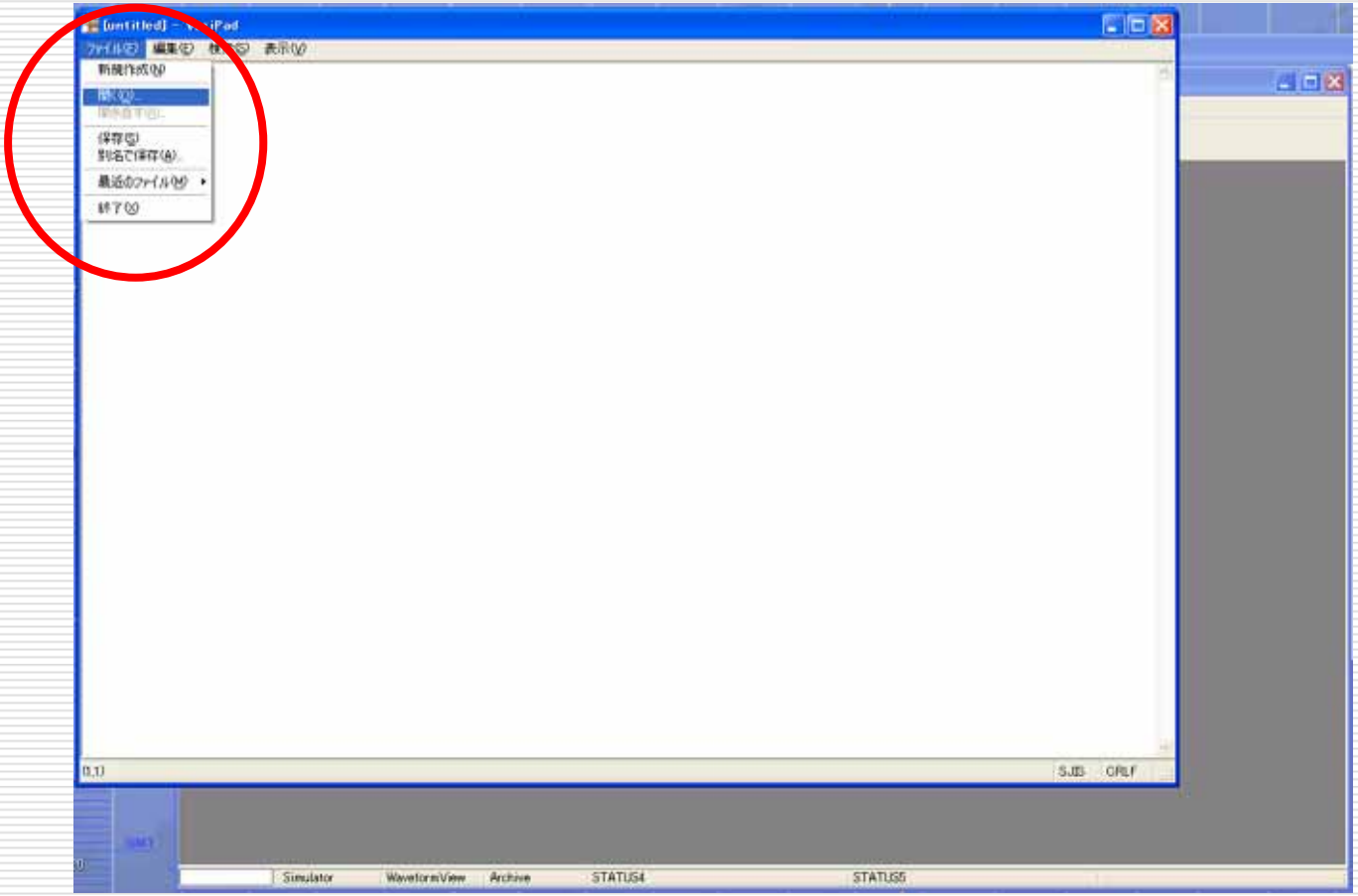
エディタの使い方

Veritakのエディタを使って見ましょう

Veritakを立ち上げてください

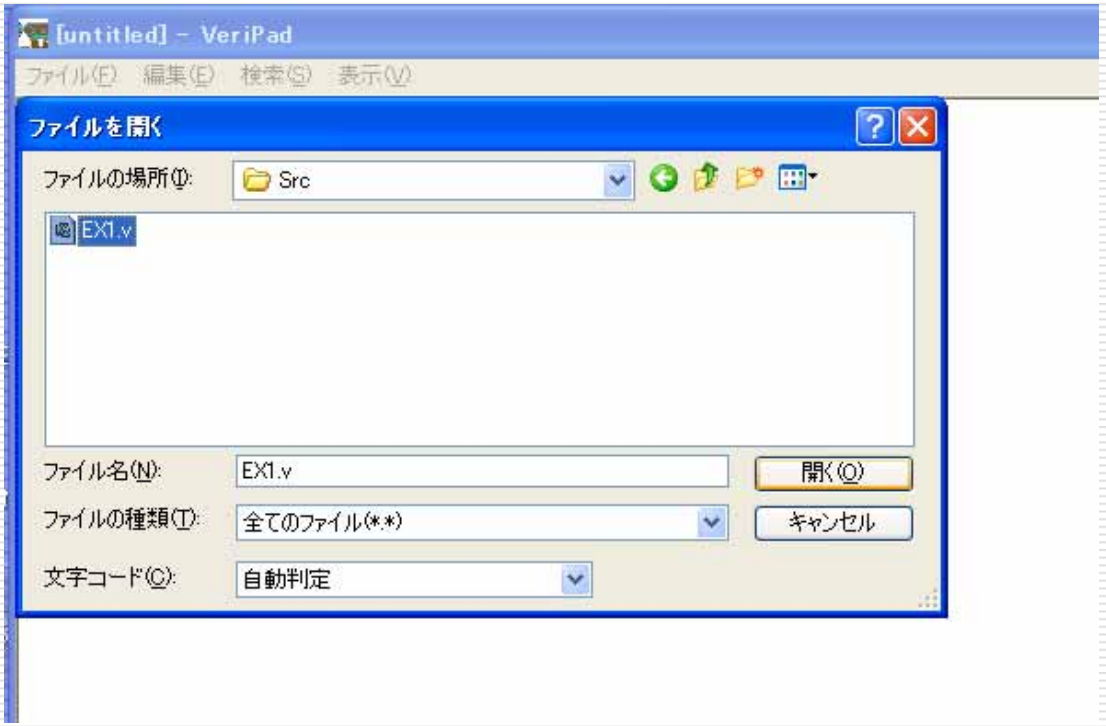


ファイルを開く



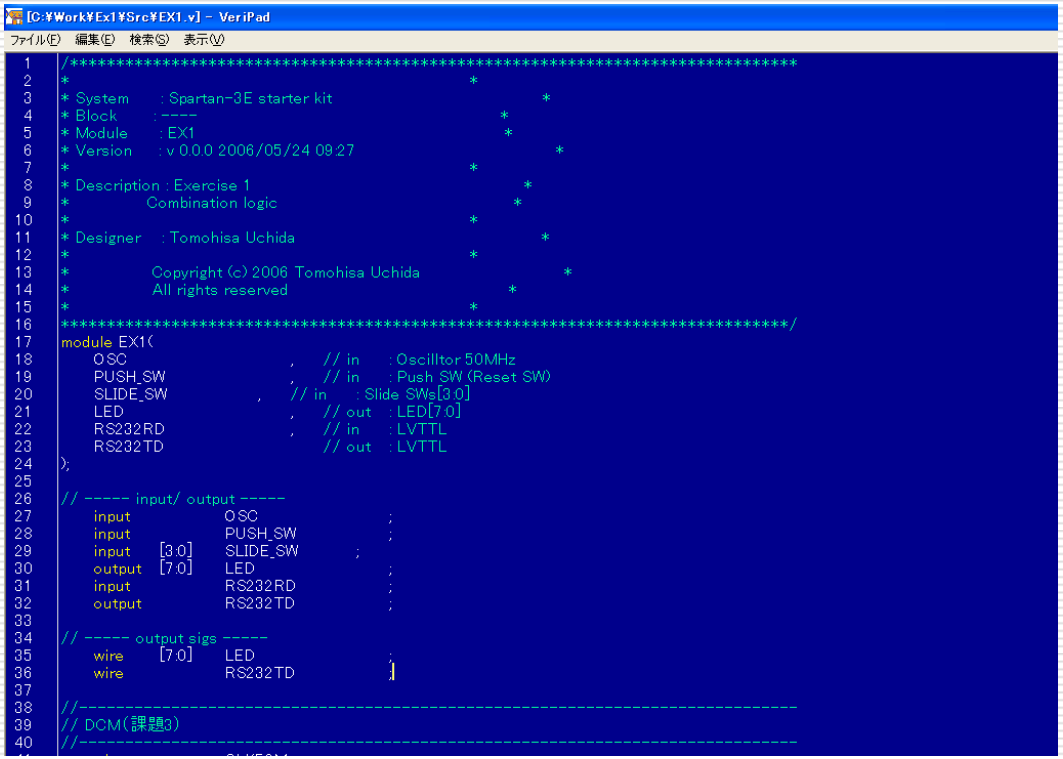
ファイルを開く

/work/Ex1/Src/Ex1.vを開いてください



Verilog-HDLソースファイル

これを使ってVerilog-HDLについて説明します。



Verilog-HDL概要

- **全体の話**
- **名前 (大文字、小文字)**
- **バス表記**
- Module
- WireとAssign

全体の話

- コメントはC言語と同じで
  - //から行末まで
  - /*から*/まで
  - 日本語は注意してください
- Verilogのコードは;で終わります。

注意！

C言語に似ていますが

ソフトウエア感覚で記述しない事！

不幸になります

大切な事は4つの構成要素でどう作れるかを常に考える事

名前

- 信号、モジュールなどは名前が必要です
- 大文字、小文字は区別されます
- アルファベット、数字、_などが使えます

バス表記

- 幅を持つデータ線などはバス表記が使えます
  - data[31:0]など
    - data[31],...data[0]をまとめて記述
  - data[31:0]をdataと略記できますが、しない事をお勧めします。
    - 記述エラーを見つけやすくする為
  - C[3:0]=A[3:0] & B[3:0];と書いたら？
  - C=(&A[3:0]);と書いたら？
  - C=(|B[3:0]);と書いたら？

module

- 回路の事です
  - 名前はEX1です
  - ここではFPGAの事です
  - ユーザーが勝手に定義する事ができます。
- 入力信号、出力信号定義
  - 使用するSW、LED
- 論理回路を記述

組み合わせ回路の書き方

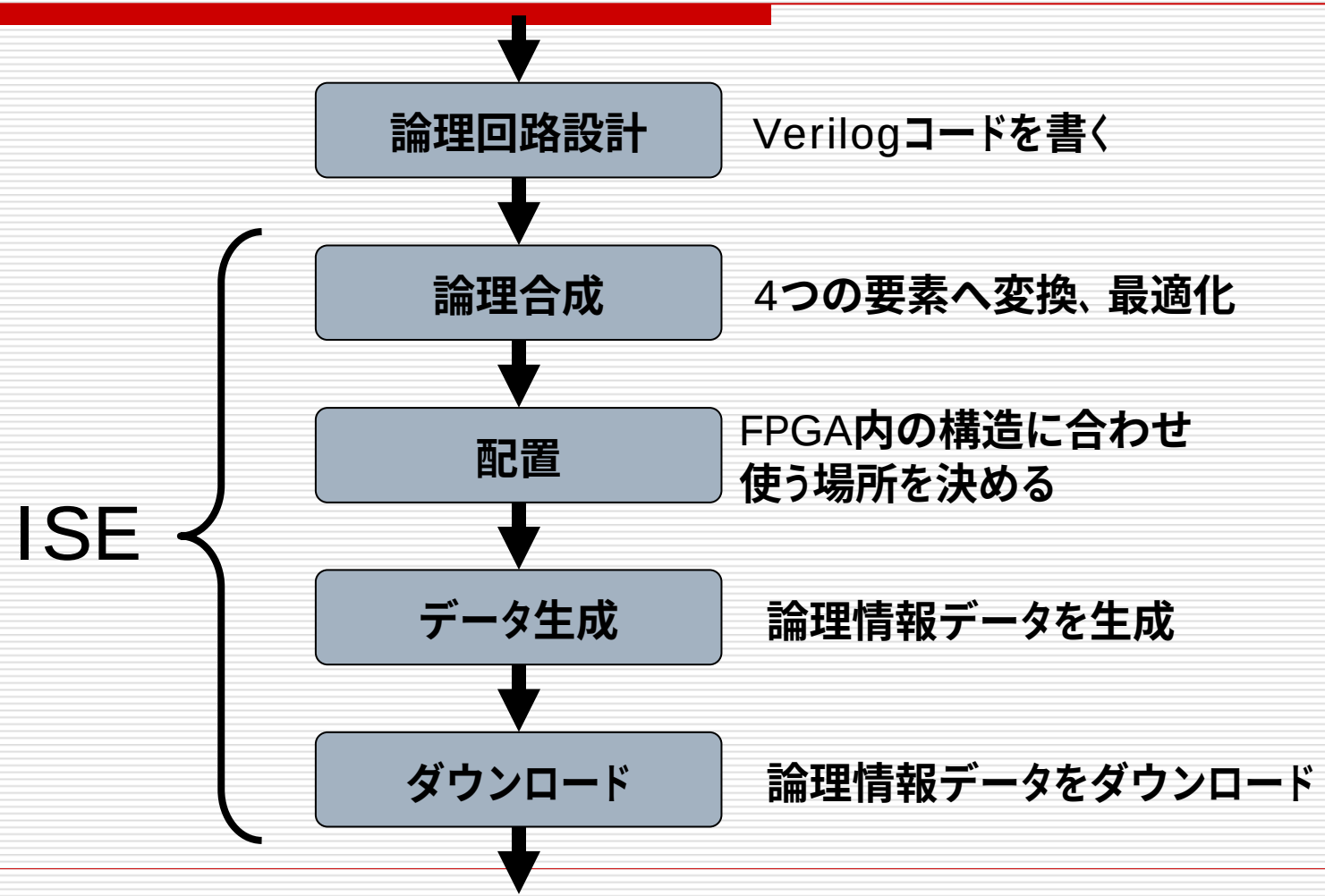
- 出力を定義
  - 組み合わせ回路の出力はwireで定義
  - DFF出力はregで定義
- 論理を書く
  - 組み合わせ回路はassignを使って書く
  - 右辺に左辺がONになる条件を書く
  - 他の書き方も出来るがお勧めしません
    - 慣れてきたら他の書き方にも挑戦してください
  - 実際、同期回路の場合はこれだけで十分！

正論理と負論理

- 正論理
  - High levelを'1' (ON) と定義
  - 通常は正論理を使う事をお勧めします
- 負論理
  - Low levelを'1' (ON) と定義
- 何故か正論理、負論理と呼びます
- FPGA外部との接続時に注意！！
  - 接続する仕様を良く確認してください
  - 動かないので悩んでいたら論理が逆だったと言う事が良くあります。

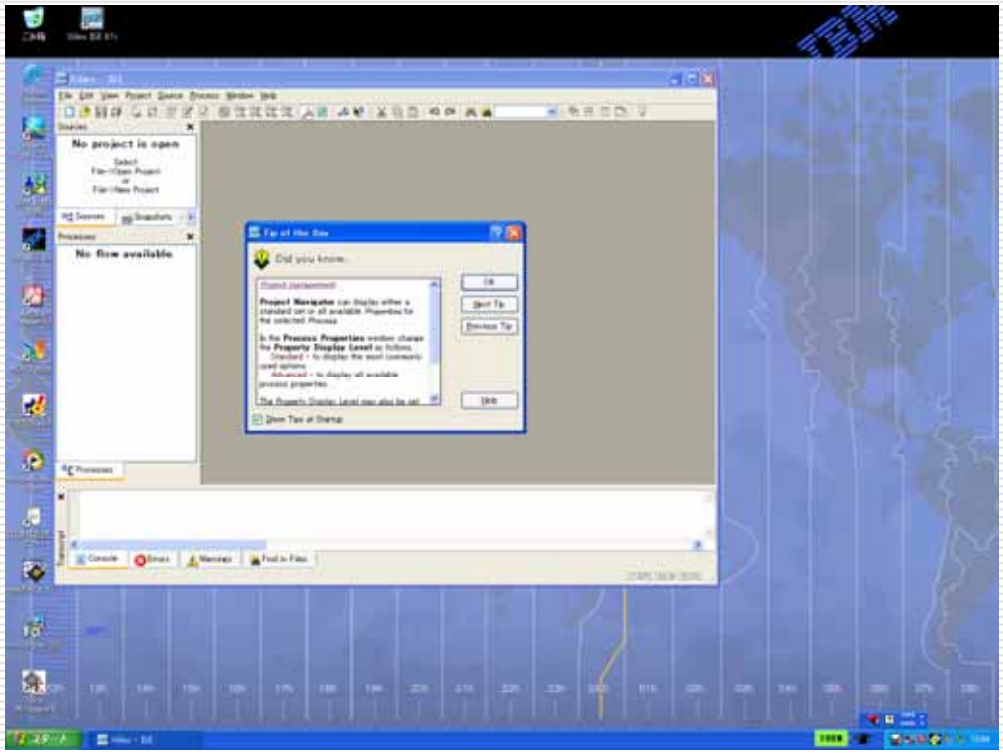
これをFPGAに実装しましょう！

作業の流れ


論理回路設計
Verilogコードを書く
論理合成
4つの要素へ変換、最適化
配置
FPGA内の構造に合わせ
使う場所を決める
データ生成
論理情報データを生成
ダウンロード
論理情報データをダウンロード
ISE

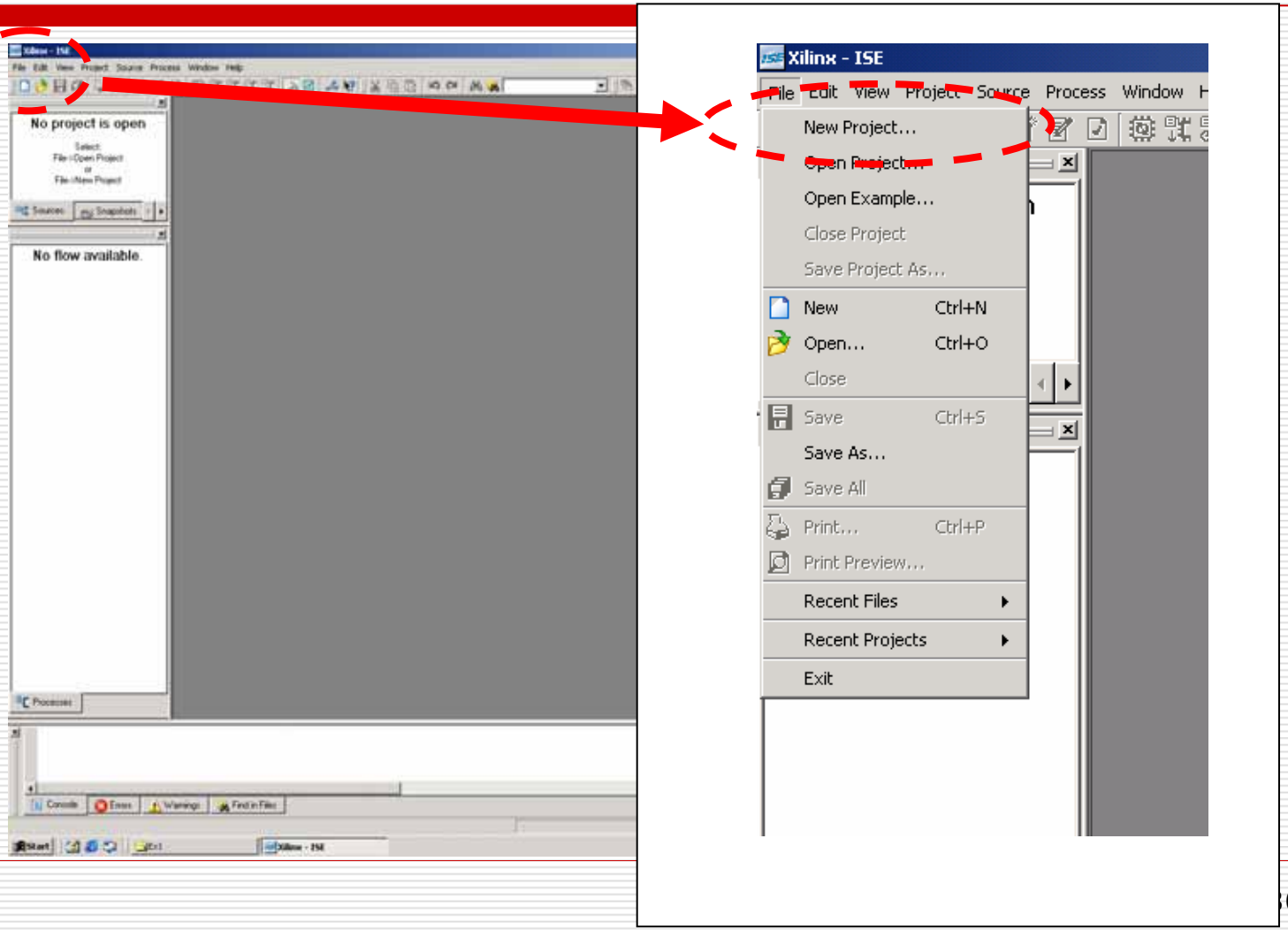
ISEを使ってみましょう！

ISE Project navigatorを立ち上げて下さい

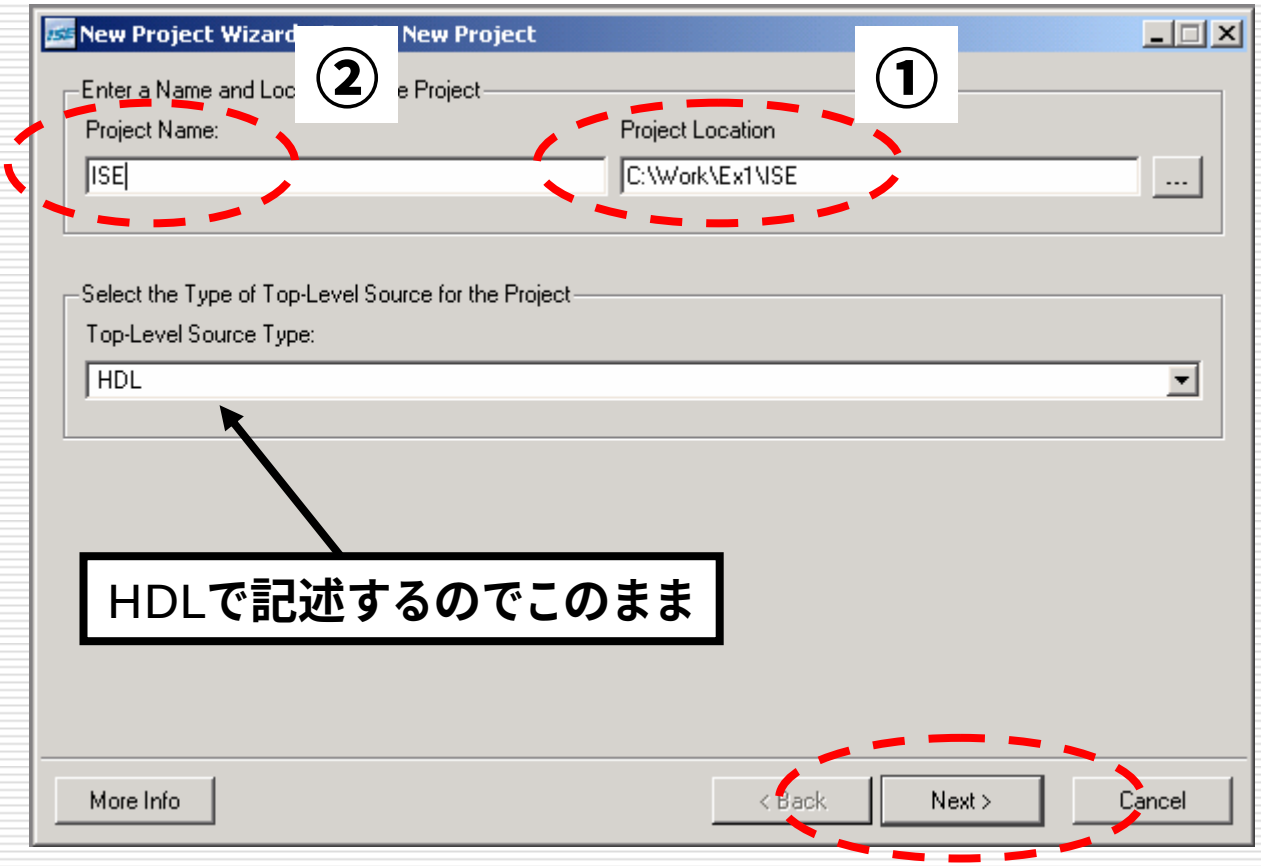


プロジェクトファイルの作成

File



プロジェクトファイルの作成
プロジェクト名と作業ホルダの指定

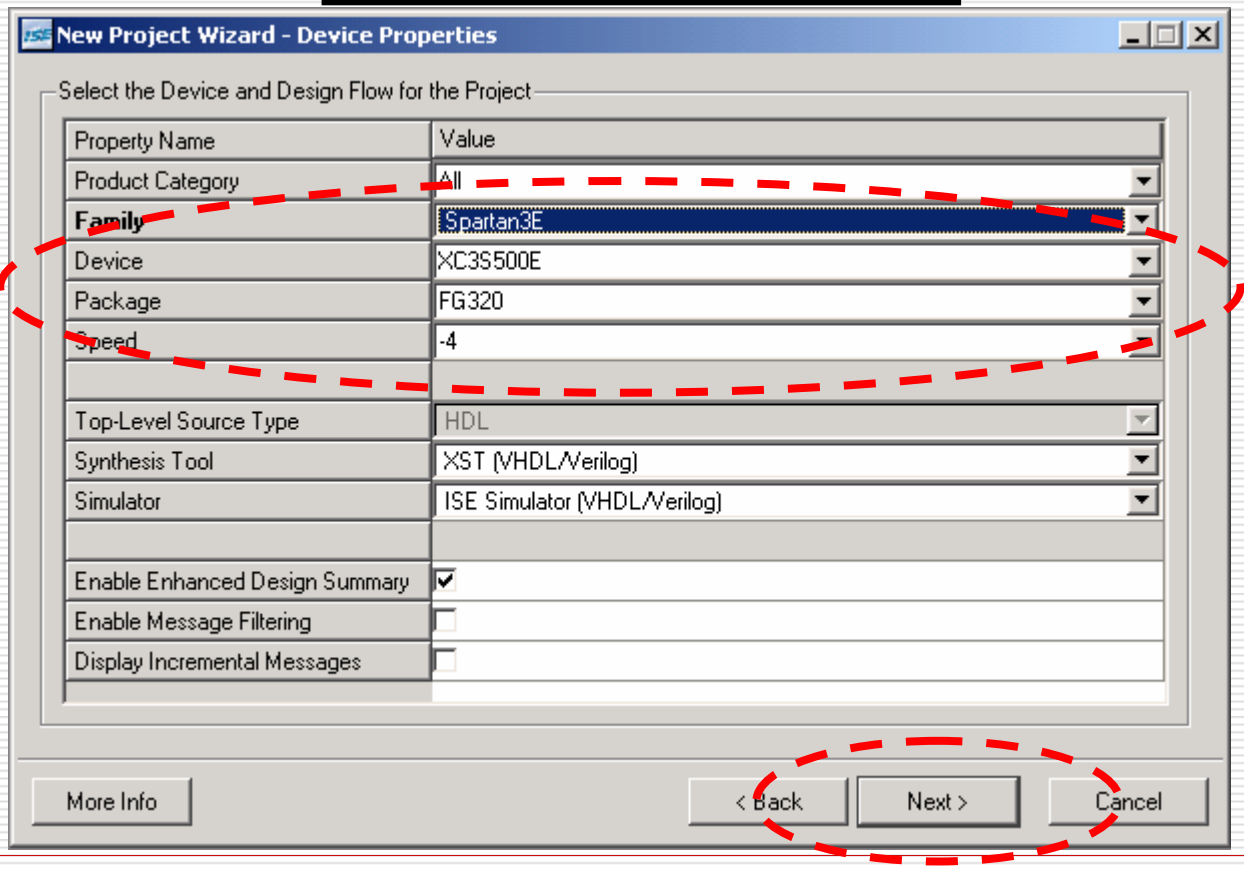


プロジェクトファイルの作成
デバイスの指定

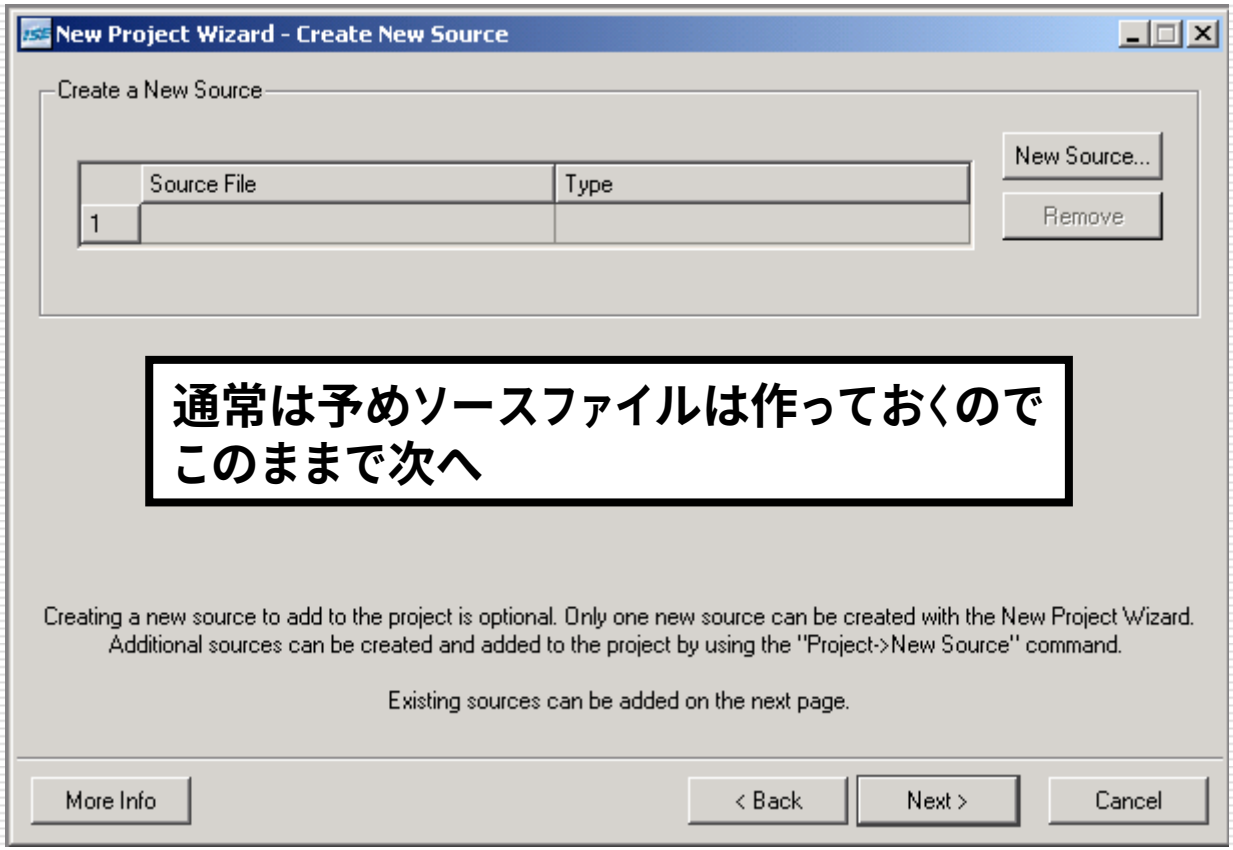
使用するデバイスを選択する

Spartan3
スターターキットを
使用している方は

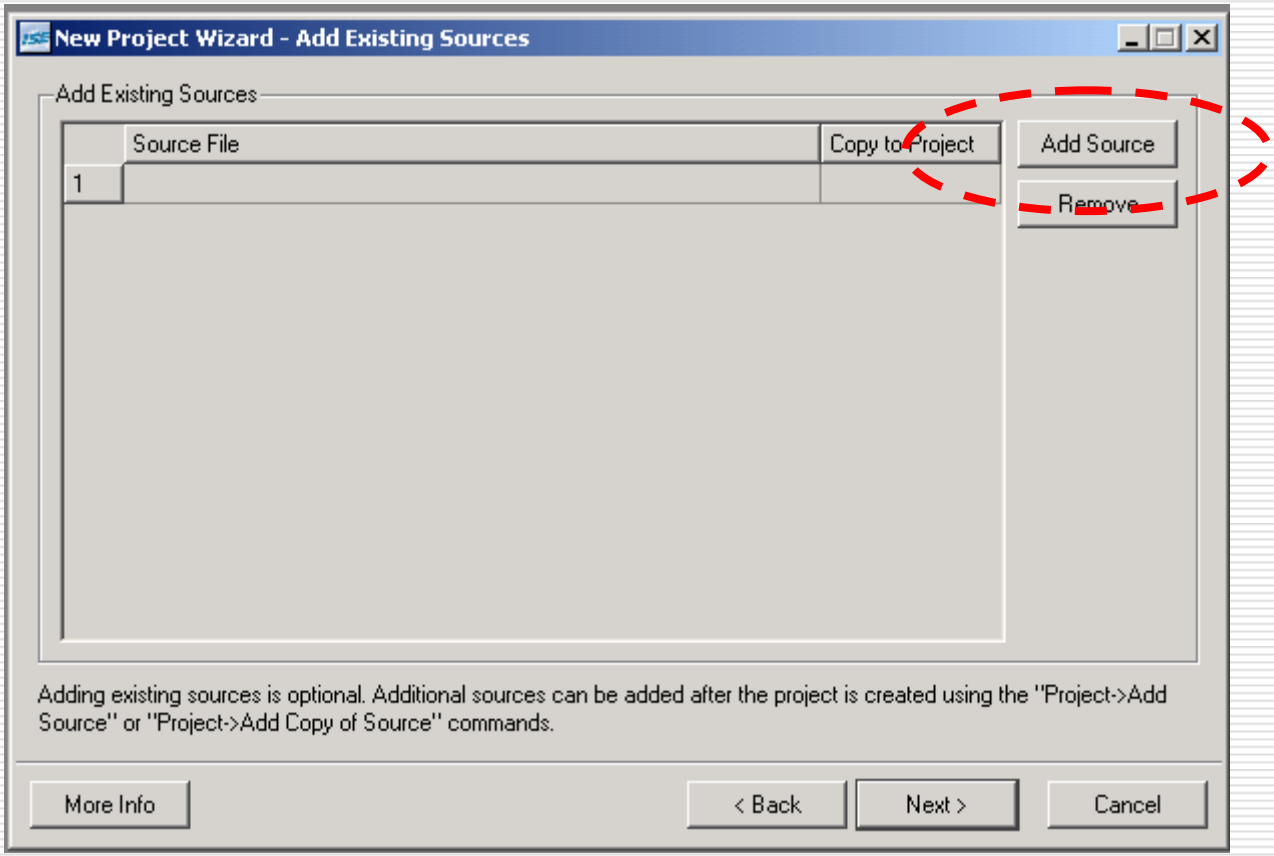
Spartan3
XC3S200
FT256



プロジェクトファイルの作成
新たしいソースファイル

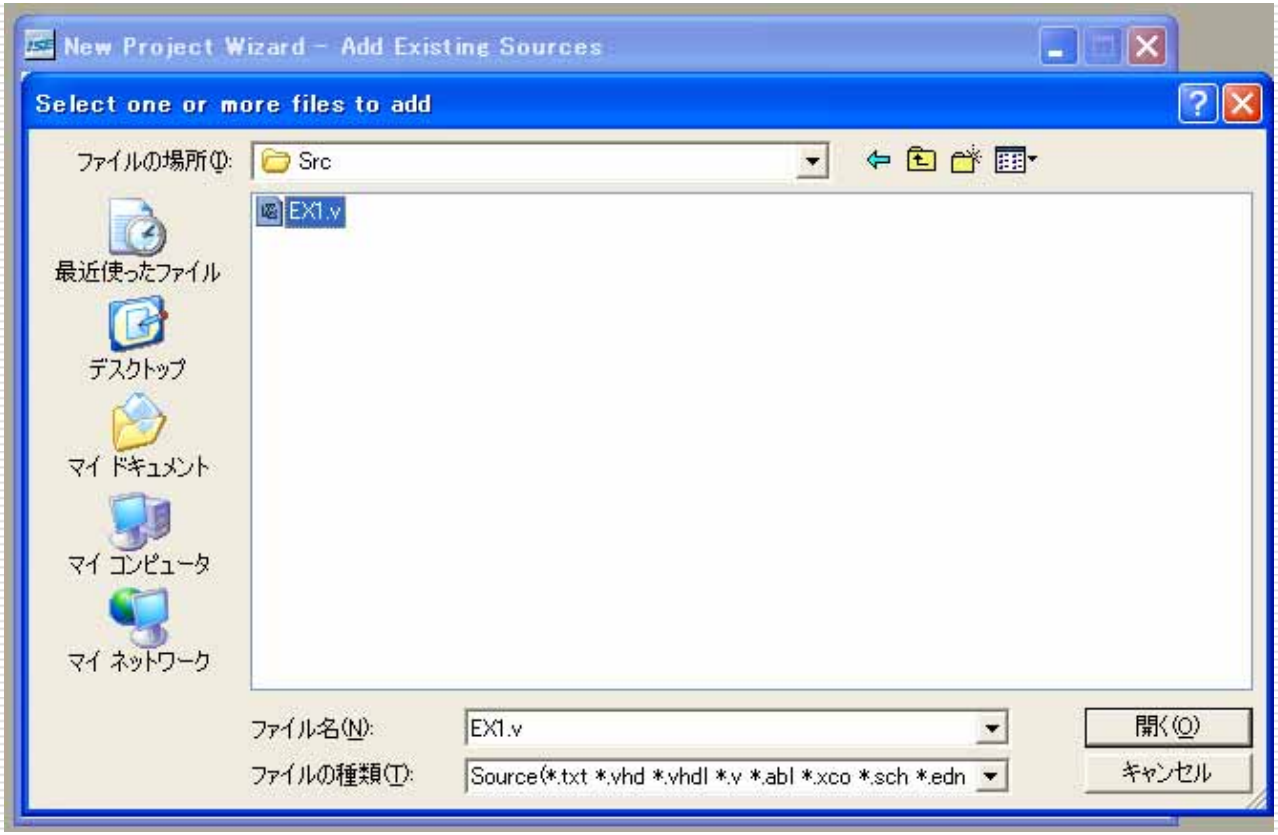


プロジェクトファイルの作成

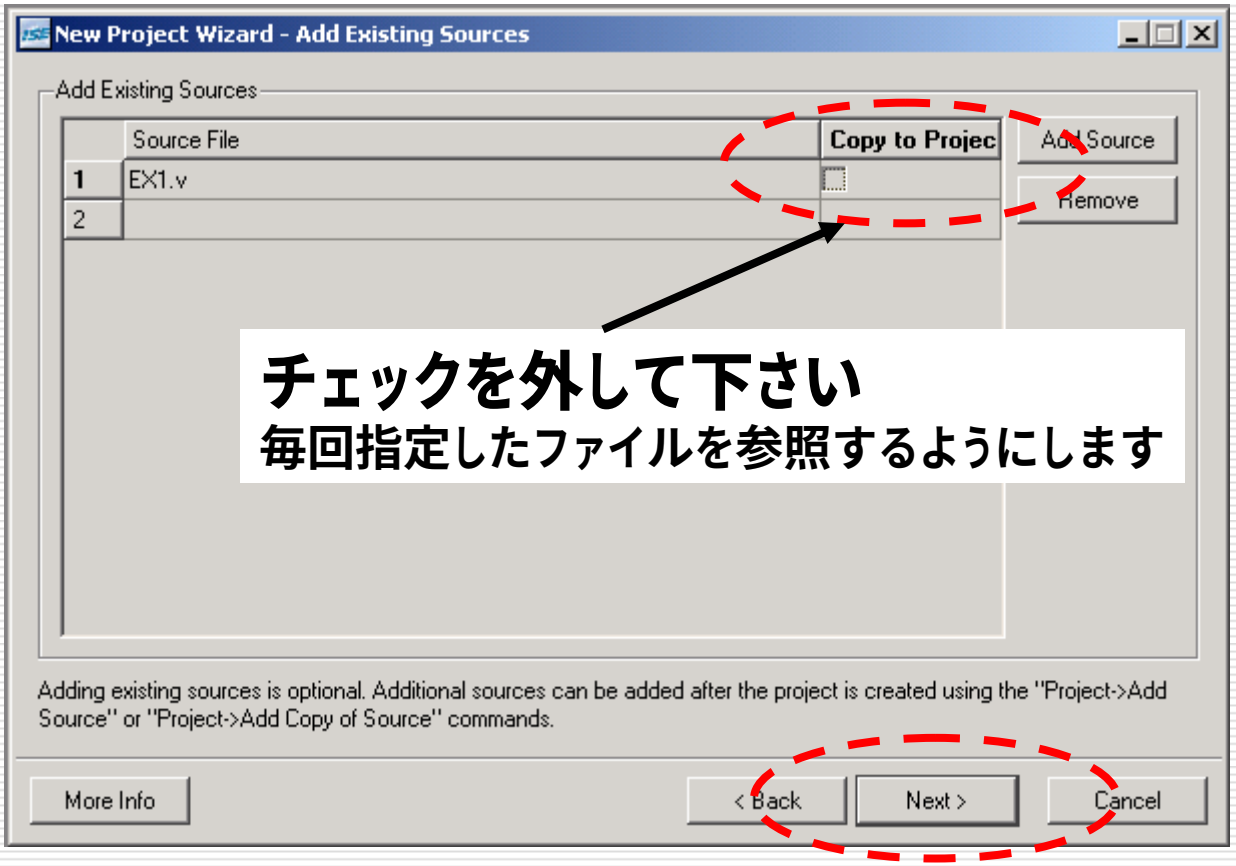


ソースファイルの指定

/work/Ex1/Src/Ex1.vを指定してください

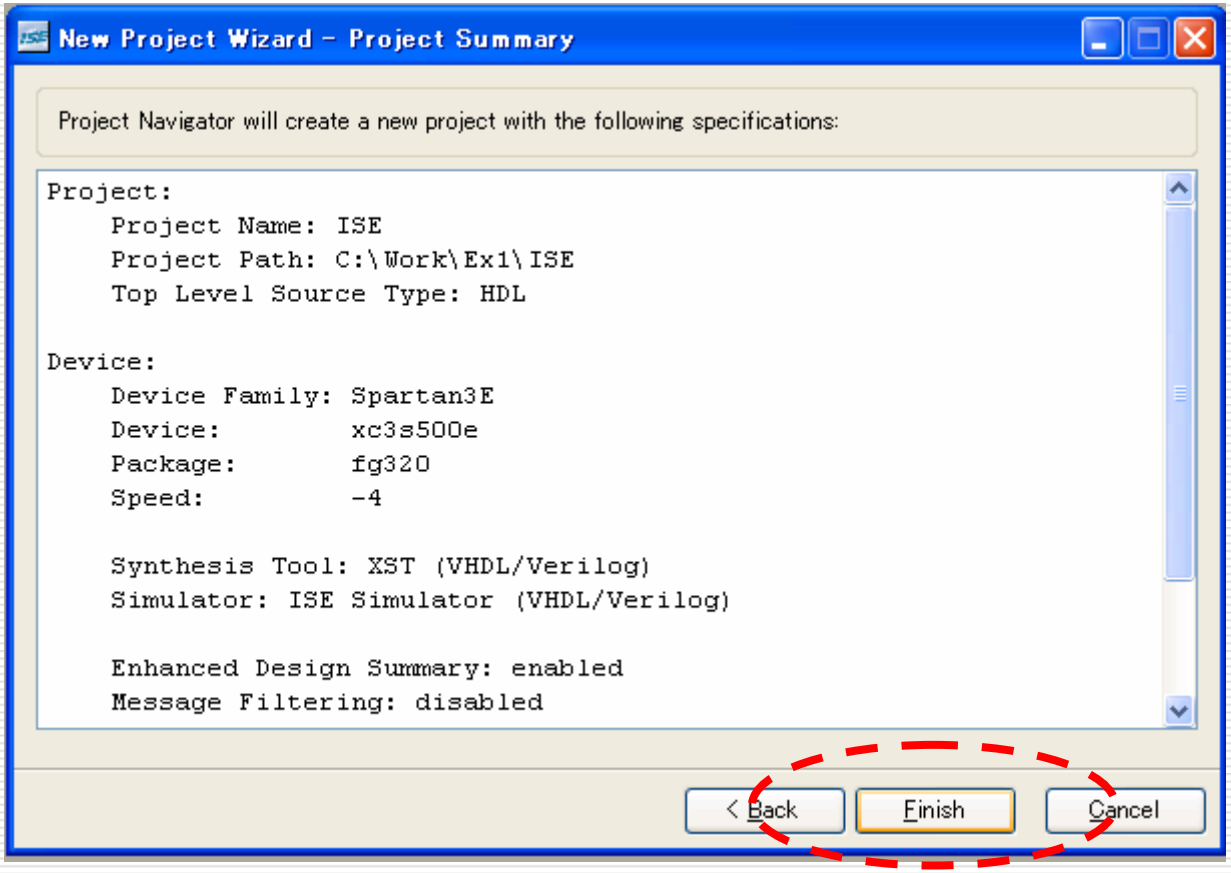


ソースファイルの指定

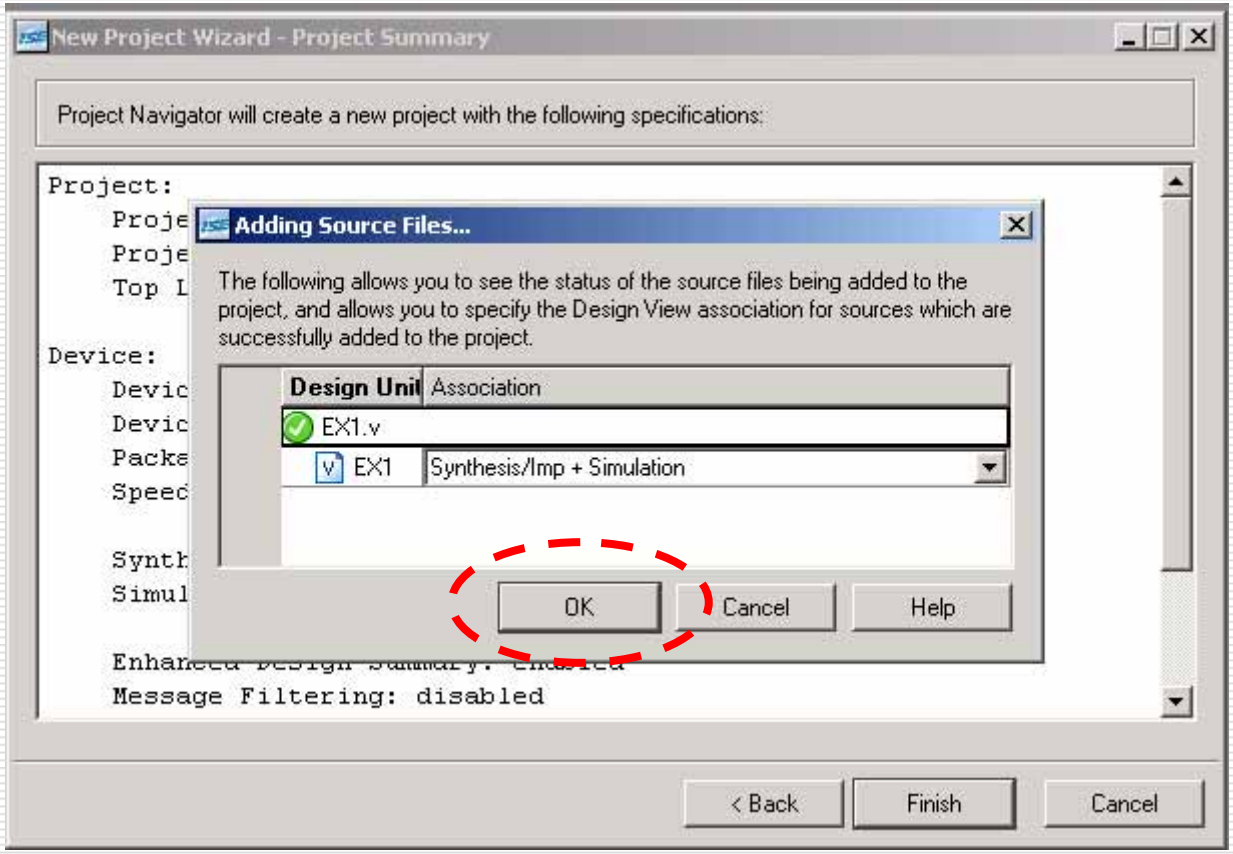


チェックをするとWorkディレクトリ(ISE)にソースファイルがコピーされ、そのコピーしたファイルを使用するようになります。
バージョン管理する場合などにはソースも保存する必要があるので、この様な機能があります。
ソースを書き換えたのに動きが何も変わらない！と、はまっている時はここを確認するのが良いです。

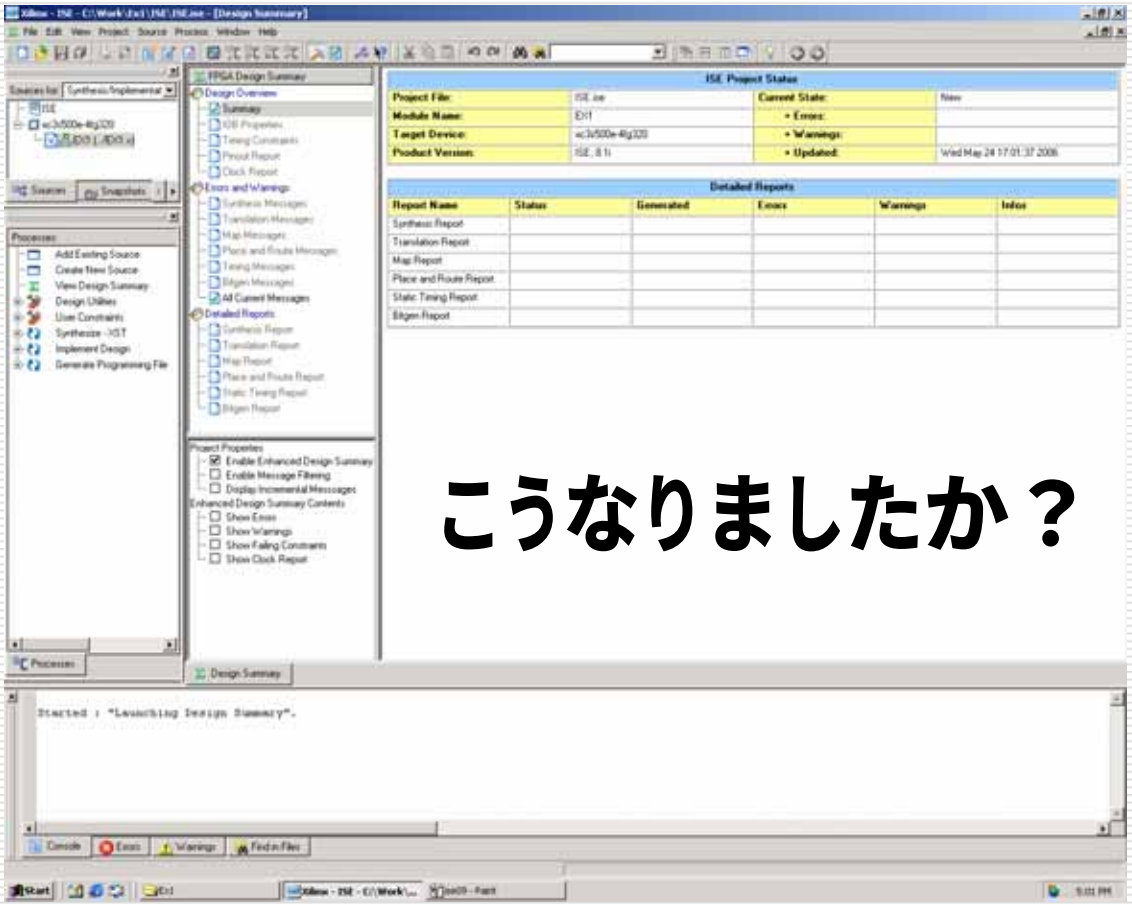
確認画面


New Project Wizard - Project Summary
Project Navigator will create a new project with the following specifications:
Project:
Project Name: ISE
Project Path: C:\Work\Ex1\ISE
Top Level Source Type: HDL
Device:
Device Family: Spartan3E
Device: xc3s500e
Package: fg320
Speed: -4
Synthesis Tool: XST (VHDL/Verilog)
Simulator: ISE Simulator (VHDL/Verilog)
Enhanced Design Summary: enabled
Message Filtering: disabled
< Back
Finish
Cancel

取り込んだファイルの確認

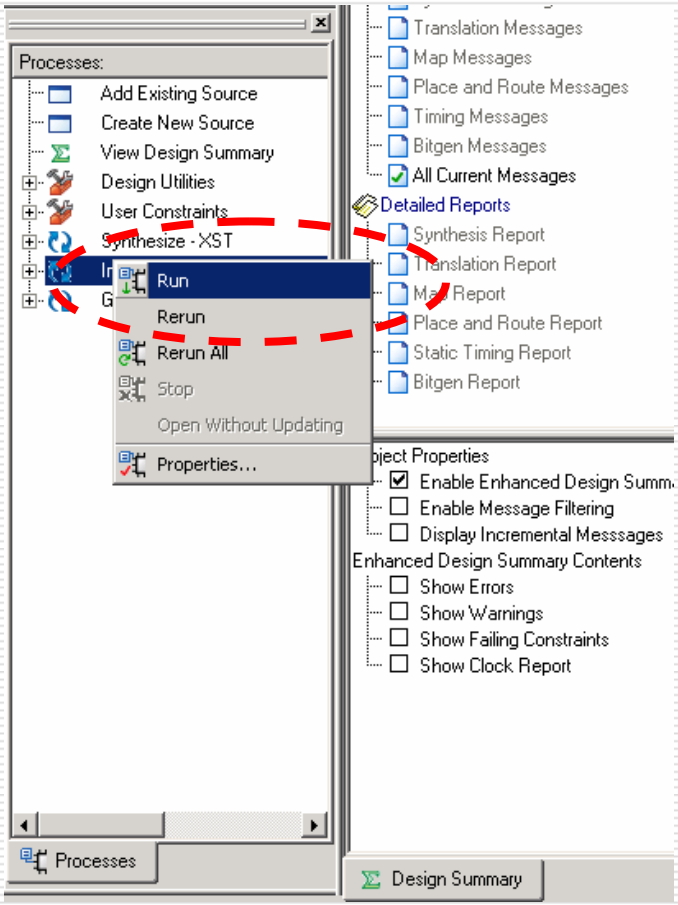


プロジェクトの初期画面

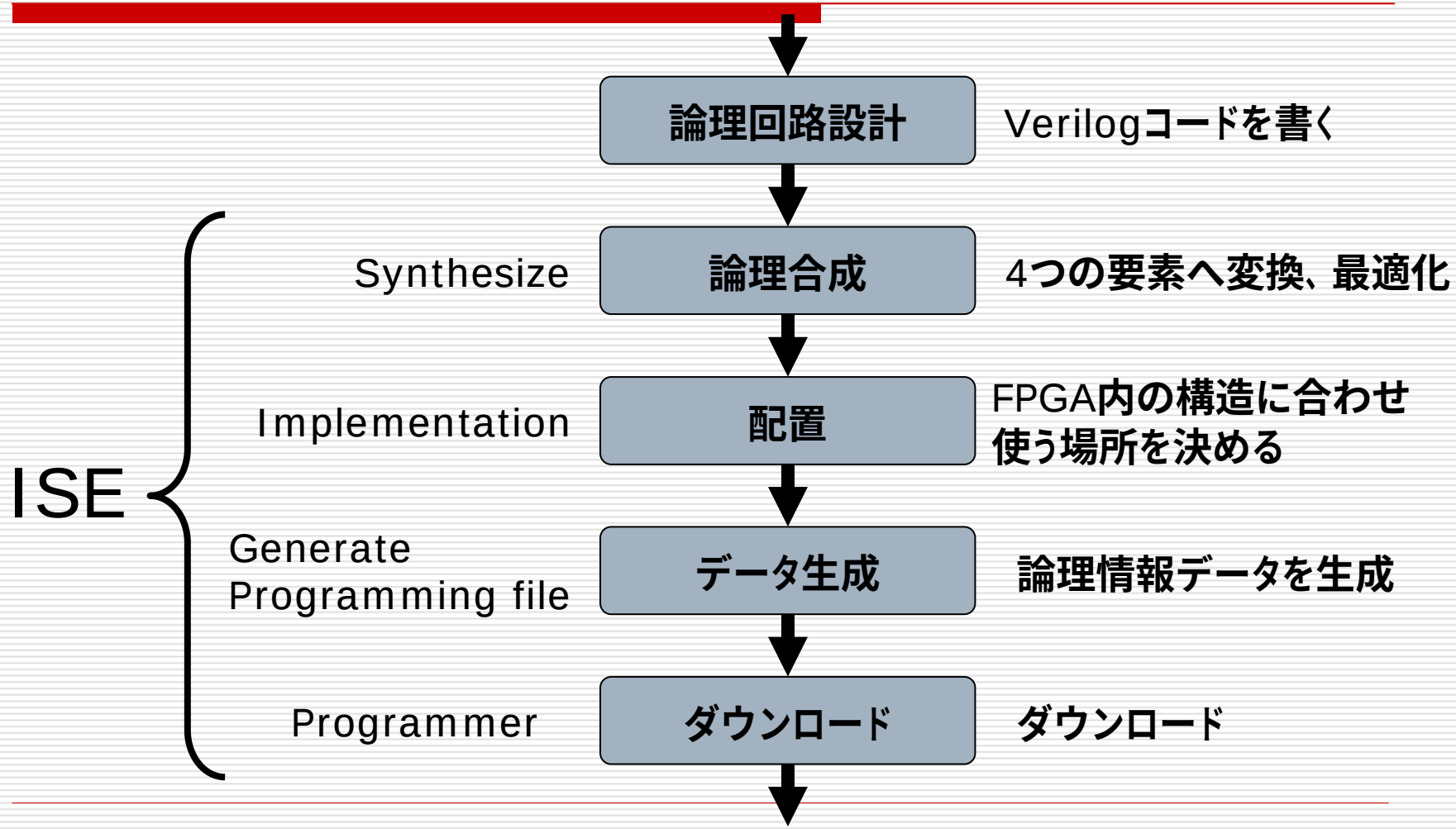


さっそくコンパイルしてみましょう！

Implementで
右クリック



作業の流れ


論理回路設計
Verilogコードを書く
ISE
Synthesize
論理合成
4つの要素へ変換、最適化
Implementation
配置
FPGA内の構造に合わせ
使う場所を決める
Generate
Programming file
データ生成
論理情報データを生成
Programmer
ダウンロード
ダウンロード

ピンの指定

ピン･アサイン

- ツールは信号名を知りません
  - ピンを割り当てる必要があります
- ピン情報入力ツールPACE
- 注意
  - 一度Implementプロセスが終わらないとピンリストが自動的に出てきません。
  - ピンを入力する時はコードのエラーチェックを兼ねてISEの全工程を実行しましょう

Spartan3E Starter Kit Board

Name	LOC	I/O Std.	Drive Str.	Termination	Slew
LED[0]	F12	LVTTL	8		SLOW
LED[1]	E12	LVTTL	8		SLOW
LED[2]	E11	LVTTL	8		SLOW
LED[3]	F11	LVTTL	8		SLOW
LED[4]	C11	LVTTL	8		SLOW
LED[5]	D11	LVTTL	8		SLOW
LED[6]	E9	LVTTL	8		SLOW
LED[7]	F9	LVTTL	8		SLOW
OSC	C9	LVCMOS33			
PUSH_SW	K17	LVTTL		PULLDOWN	
RS232RD	R7	LVTTL			
RS232TD	M14	LVTTL	8		SLOW
SLIDE_SW[0]	L13	LVTTL		PULLUP	
SLIDE_SW[1]	L14	LVTTL		PULLUP	
SLIDE_SW[2]	H18	LVTTL		PULLUP	
SLIDE_SW[3]	N17	LVTTL		PULLUP	

注) 表示されない信号は入力しないで下さい

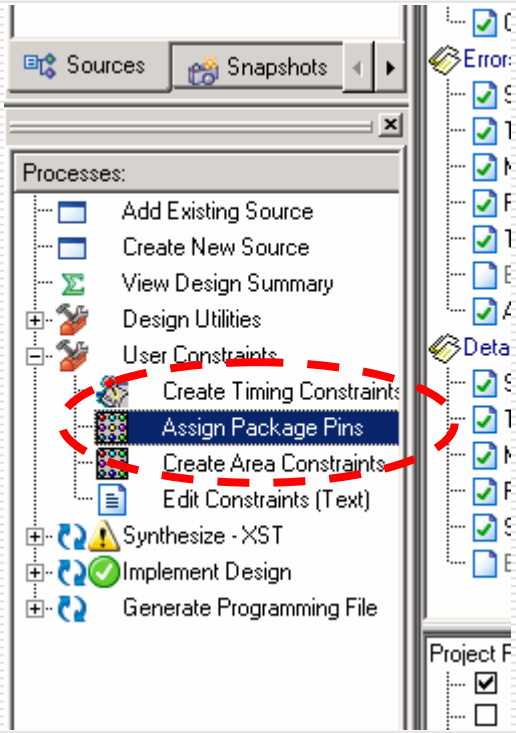
Spartan3 Starter Kit Board

Name	LOC	I/O Std.	Drive Str.	Termination	Slew
LED[0]	K12	LVTTL	8		SLOW
LED[1]	P14	LVTTL	8		SLOW
LED[2]	L12	LVTTL	8		SLOW
LED[3]	N14	LVTTL	8		SLOW
LED[4]	P13	LVTTL	8		SLOW
LED[5]	N12	LVTTL	8		SLOW
LED[6]	P12	LVTTL	8		SLOW
LED[7]	P11	LVTTL	8		SLOW
OSC	T9	LVCMOS33			
PUSH_SW	L14	LVTTL			
RS232RD	T13	LVTTL			
RS232TD	R13	LVTTL	8		SLOW
SLIDE_SW[0]	F12	LVTTL			
SLIDE_SW[1]	G12	LVTTL			
SLIDE_SW[2]	H14	LVTTL			
SLIDE_SW[3]	H13	LVTTL			

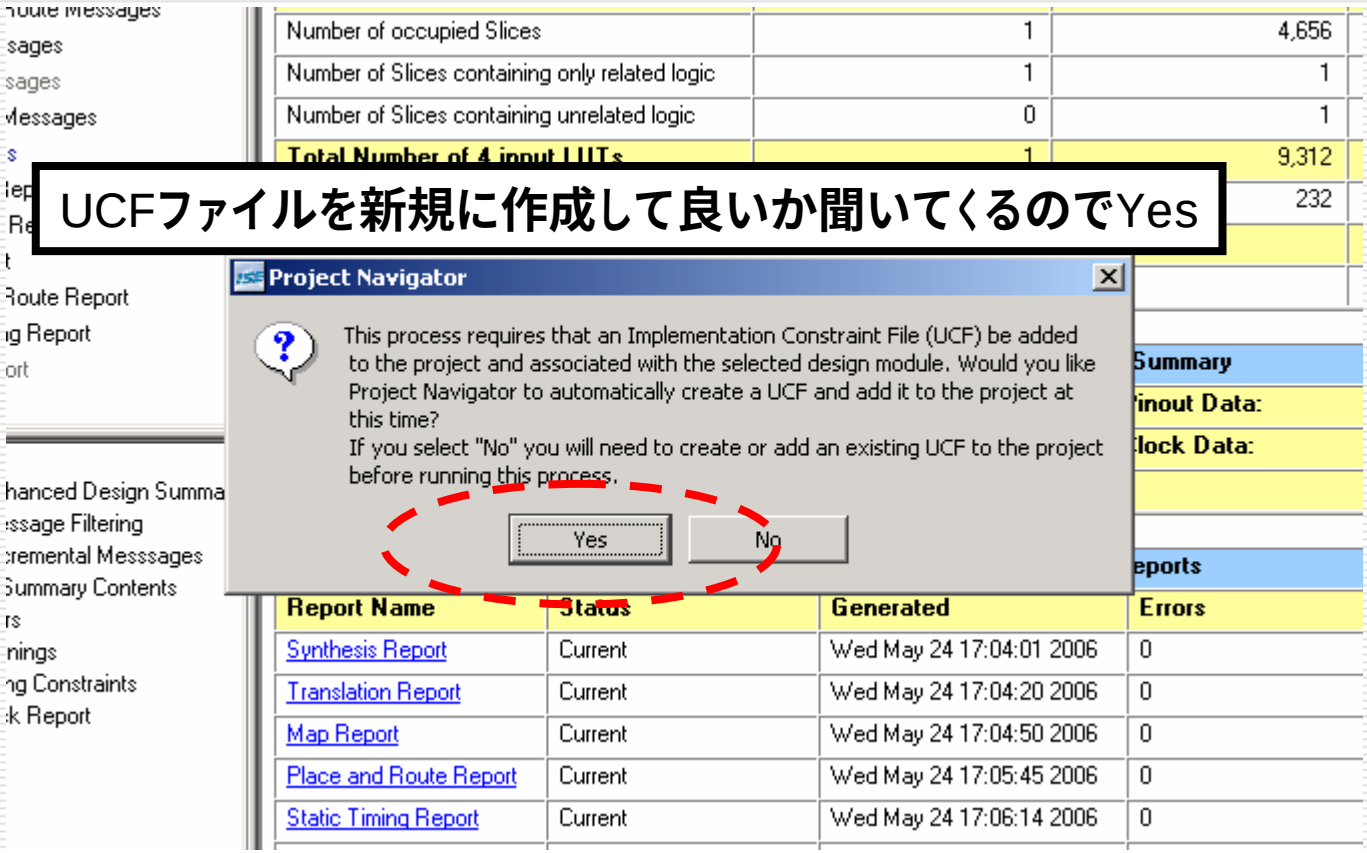
注) 表示されない信号は入力しないで下さい

PACEの起動

ダブルクリック



UCFの新規作成確認

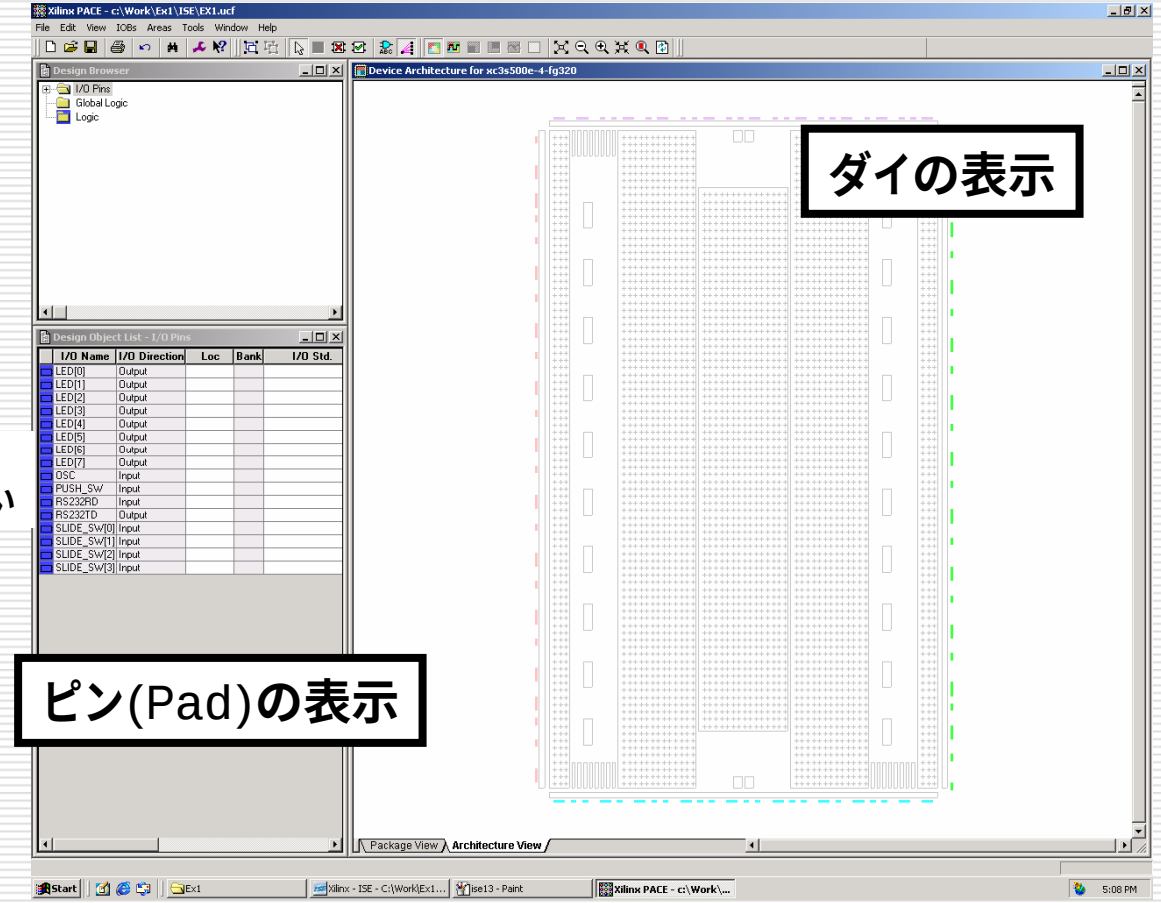

UCFファイルを新規に作成して良いか聞いてくるのでYes
Project Navigator
This process requires that an Implementation Constraint File (UCF) be added to the project and associated with the selected design module. Would you like Project Navigator to automatically create a UCF and add it to the project at this time?
If you select "No" you will need to create or add an existing UCF to the project before running this process.
Yes
No
Number of occupied Slices 1 4,656
Number of Slices containing only related logic 1 1
Number of Slices containing unrelated logic 0 1
9,312
232
Route Report
Report Name Status Generated Errors
Synthesis Report Current Wed May 24 17:04:01 2006 0
Translation Report Current Wed May 24 17:04:20 2006 0
Map Report Current Wed May 24 17:04:50 2006 0
Place and Route Report Current Wed May 24 17:05:45 2006 0
Static Timing Report Current Wed May 24 17:06:14 2006 0

UCFファイルとは？

- User Constraint File
  - ユーザーの要求を記述するテキストファイル
  - ここではEX1.ucf
  - 以前はテキストで直接書いていた
  - PACEは入力支援アプリ
- 制約例
  - ピン配置
  - タイミングなど
- ユーザーが指定することはUCFに書く

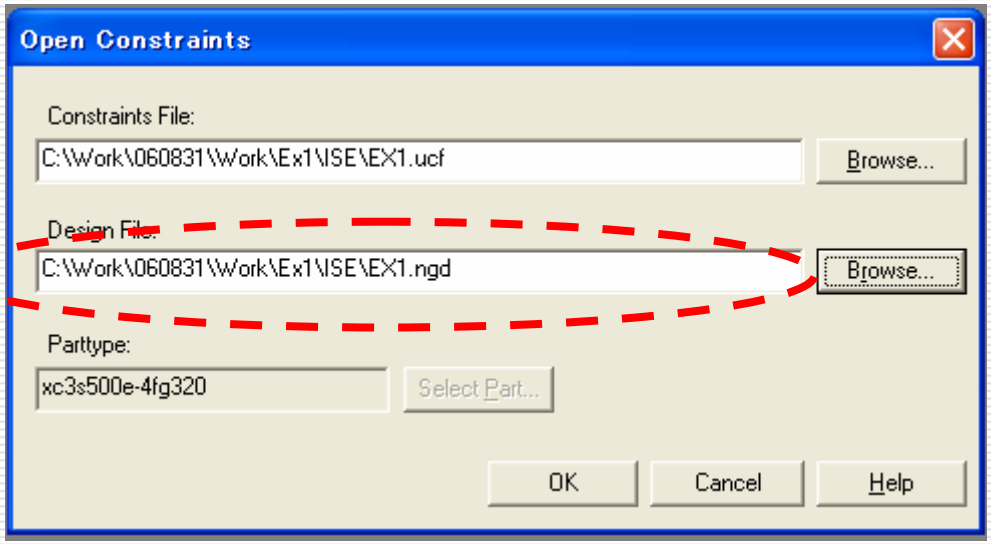
PACE起動画面

プロジェクト作成直後は
ピンリストが生成されない



画面が開かなかった人は

File -> Open



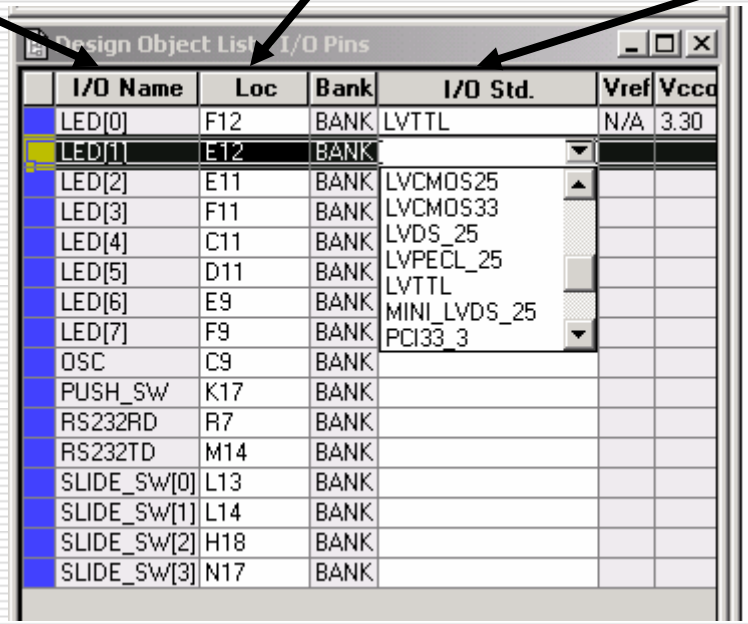
Ngdファイルが無い時はImplement Process**が**
終了していませんのでImplement Process**を実行してください**

ピン情報入力

入出力信号（ピン）名

ピン位置

信号の電気的仕様を入力



注意）QFPのピン位置はP１２の様に先頭にPが必要

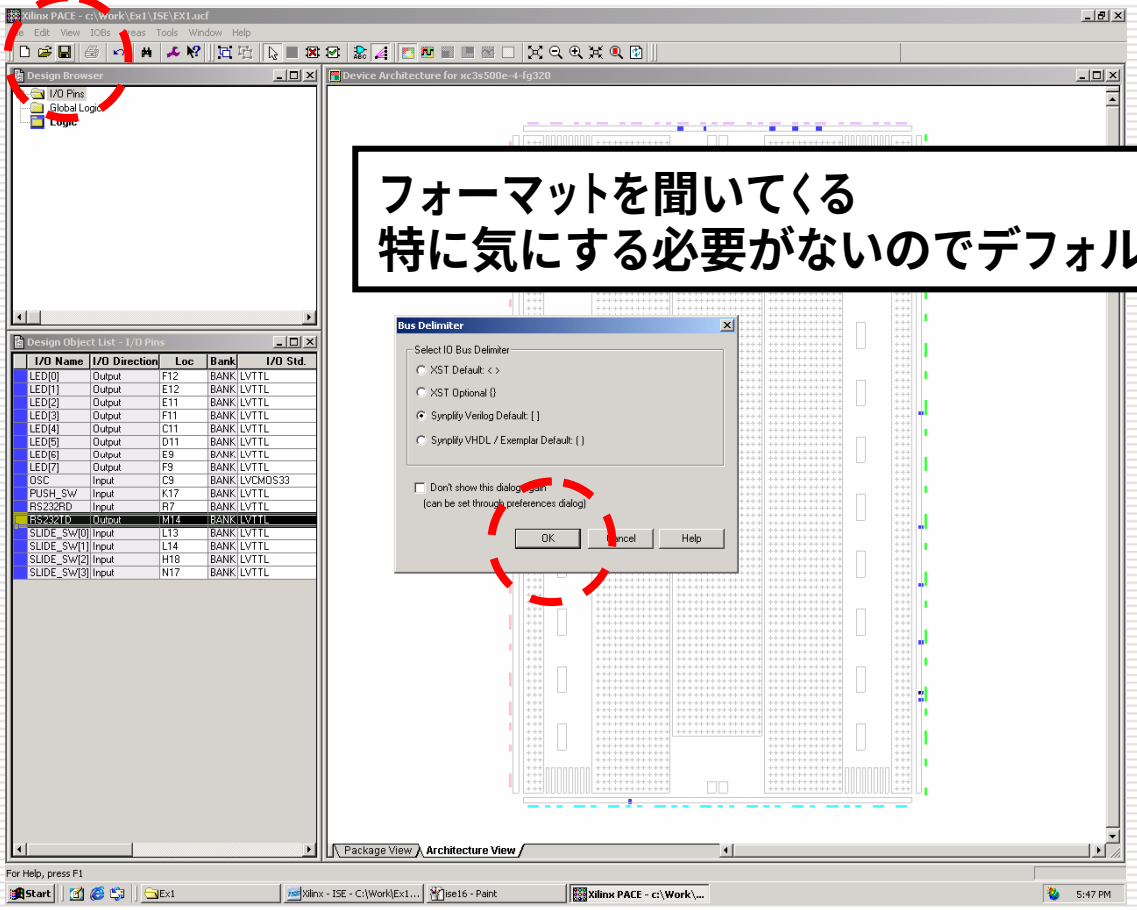
Driverの強さ、ターミネーション

PULLUP:抵抗を通して電源に接続

PULLDOWN:抵抗を通してGNDに接続

KEEPER:状態を保持

保存

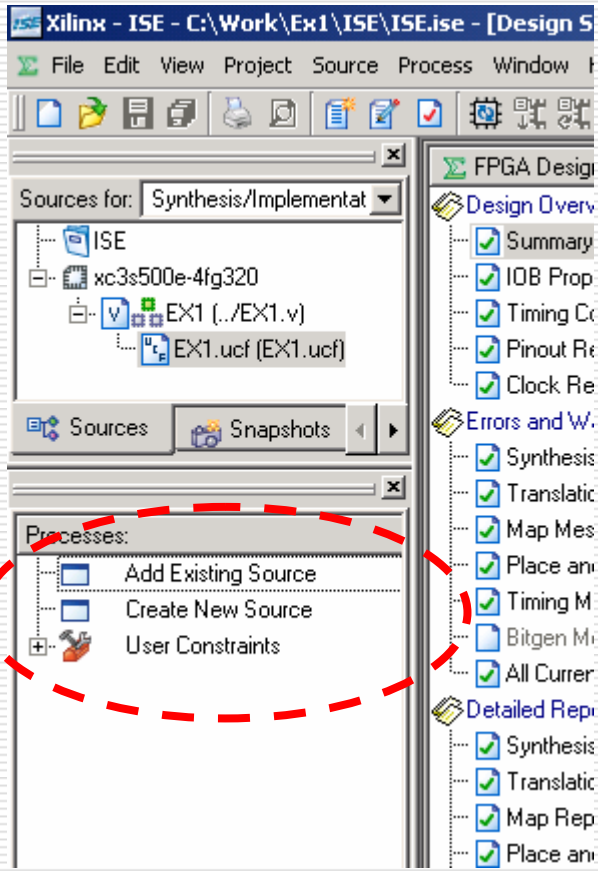

フォーマットを聞いてくる
特に気にする必要がないのでデフォルトでOK
Bus Delimiter
Select IO Bus Delimiter
XST Default: < >
XST Optional: { }
Synplify Verilog Default: [ ]
Synplify VHDL / Exemplar Default: ( )
Don't show this dialog again
(can be set through preferences dialog)
OK
Cancel
Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
I/O Name
I/O Direction
Loc
Bank
I/O Std.
Device Architecture for xc3s500e-4-fg320
Package View
Architecture View
For Help, press F1
Start
5:47 PM

もう一度実行

- ピン配置を考慮したファイルを生成します
- Process View画面が変わっていませんか？
  - Module Window内の選択したファイルにより表示が変わります
  - 選択したファイルのProcess表示と言う事です
  - EX1を選択して再実行してください

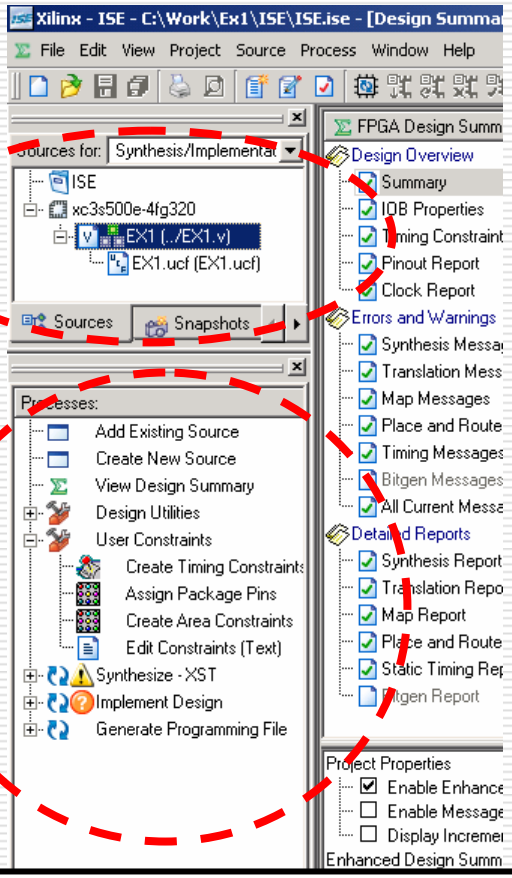
Process View

変わっている！



Sourceを変更する

Ex.vを選択



無事元の画面に

再度Implementを実行してください

データファイルの生成

Generate Programming File

- **よく使うFile形式は2つあります**
- BIT
  - **デバック時に良く使う**
  - **電源投入後、毎回手動でダウンロード**
  - FPGA Start-Up Clock＝JTAG clock
- MCS
  - ROM**に書き込むデータ**
  - FPGA Start-Up Clock＝CCLK
- **今回はBITを使います**

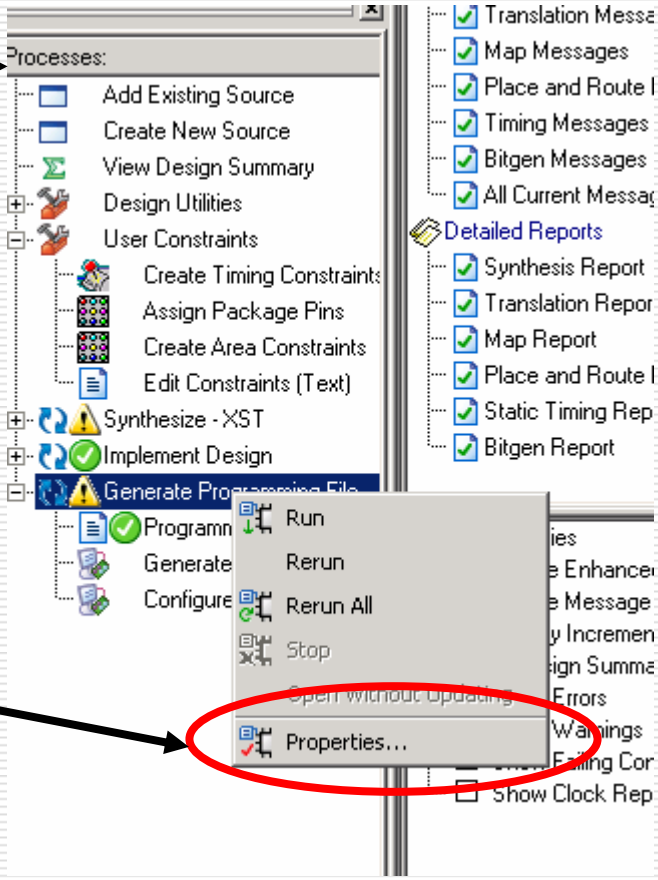
MCS**についてはユーザーガイドを参照してください**

Start Up-Clockの設定

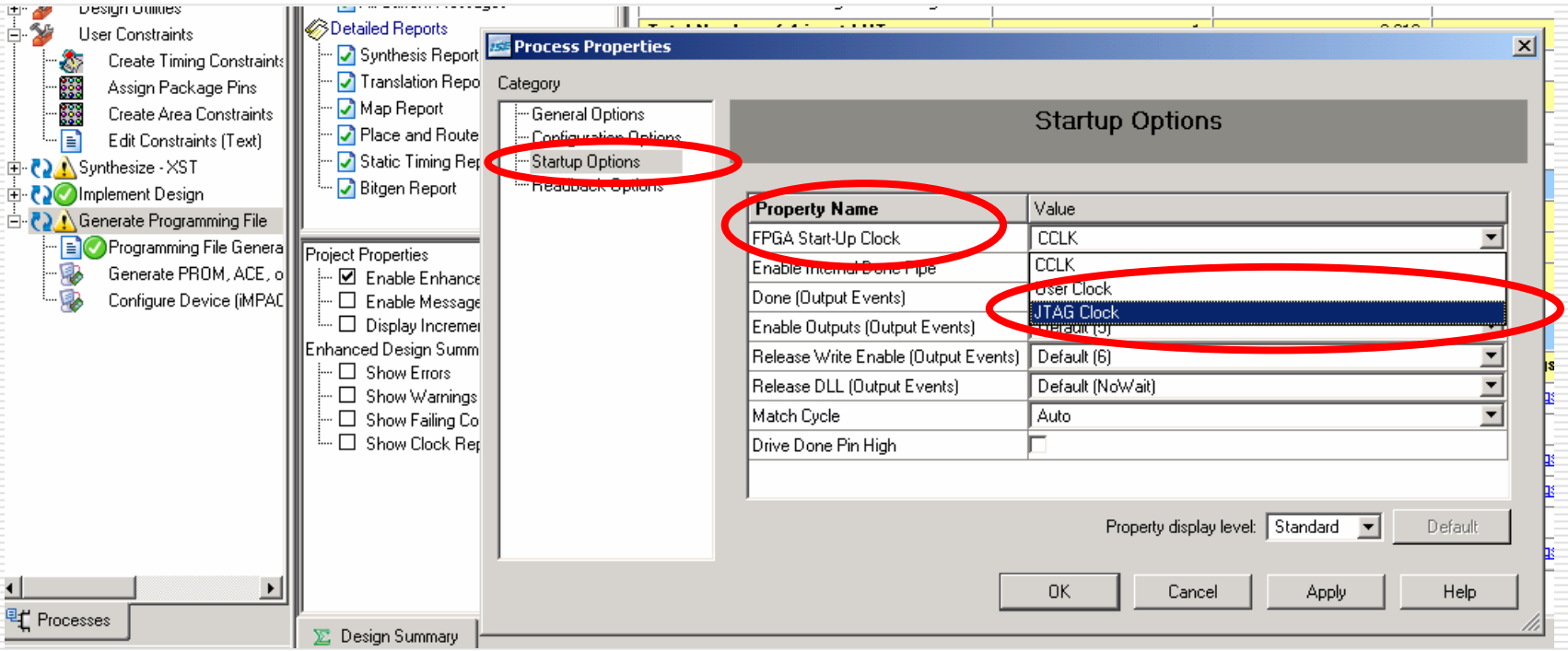
Process View

Generate Programming File

左クリック→Properties



Start Up-Clockの設定

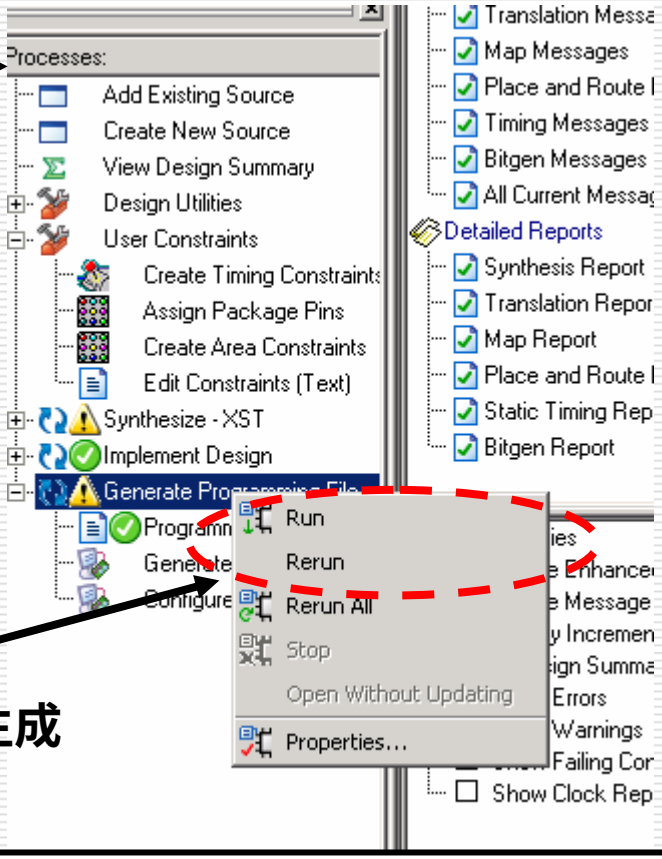


ファイルの生成

Process View

Generate Programming File

左クリック→Run, Rerunで生成



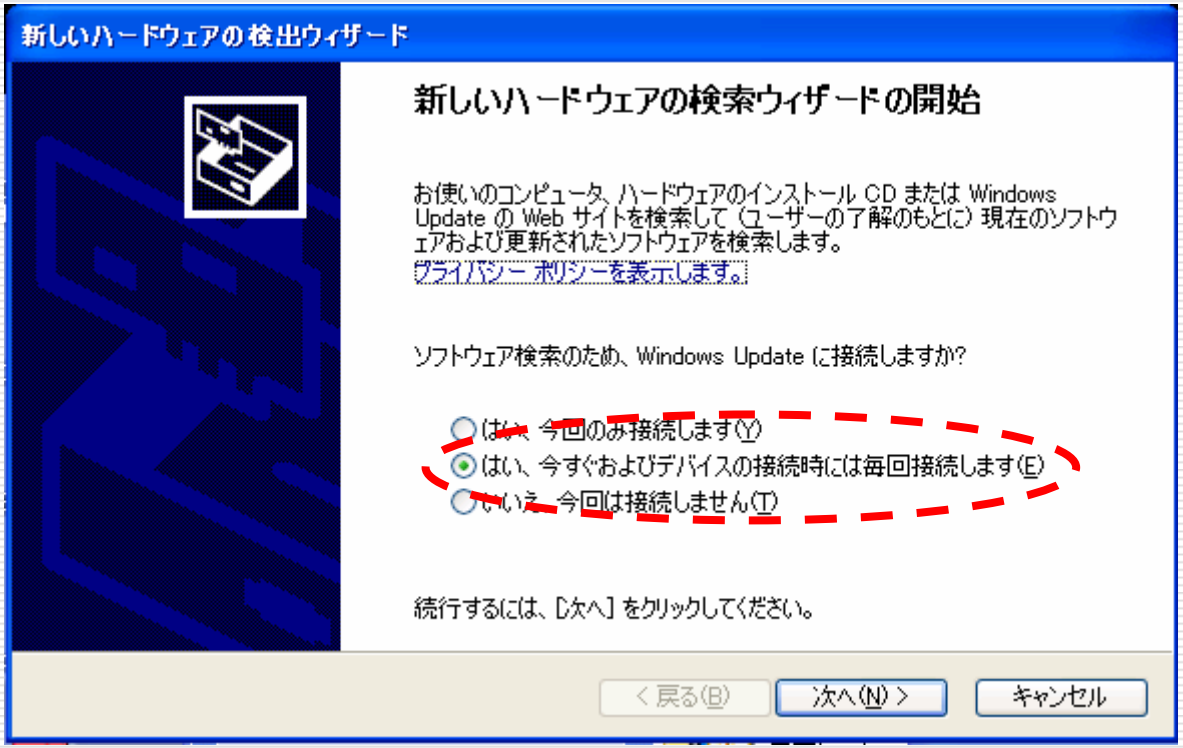
これでBITファイルが生成されました

FPGAへダウンロード

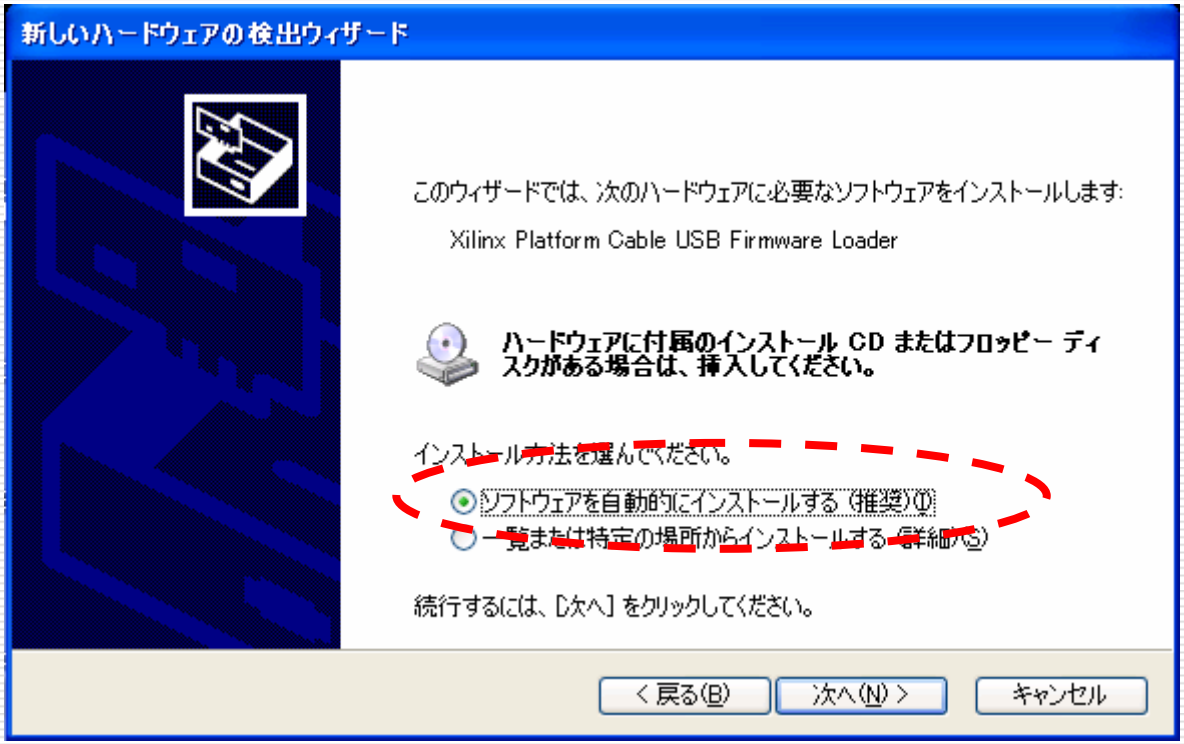
Configure device

- データをダウンロードする。
- ボードのダウンロードモードをJTAGにする
  - 3-starter kit: J8-M1のみ接続
  - 3E-starter kit: J30-M1のみ接続
- ボードの電源を入れる
- USBケーブルを接続、ドライバのインストール

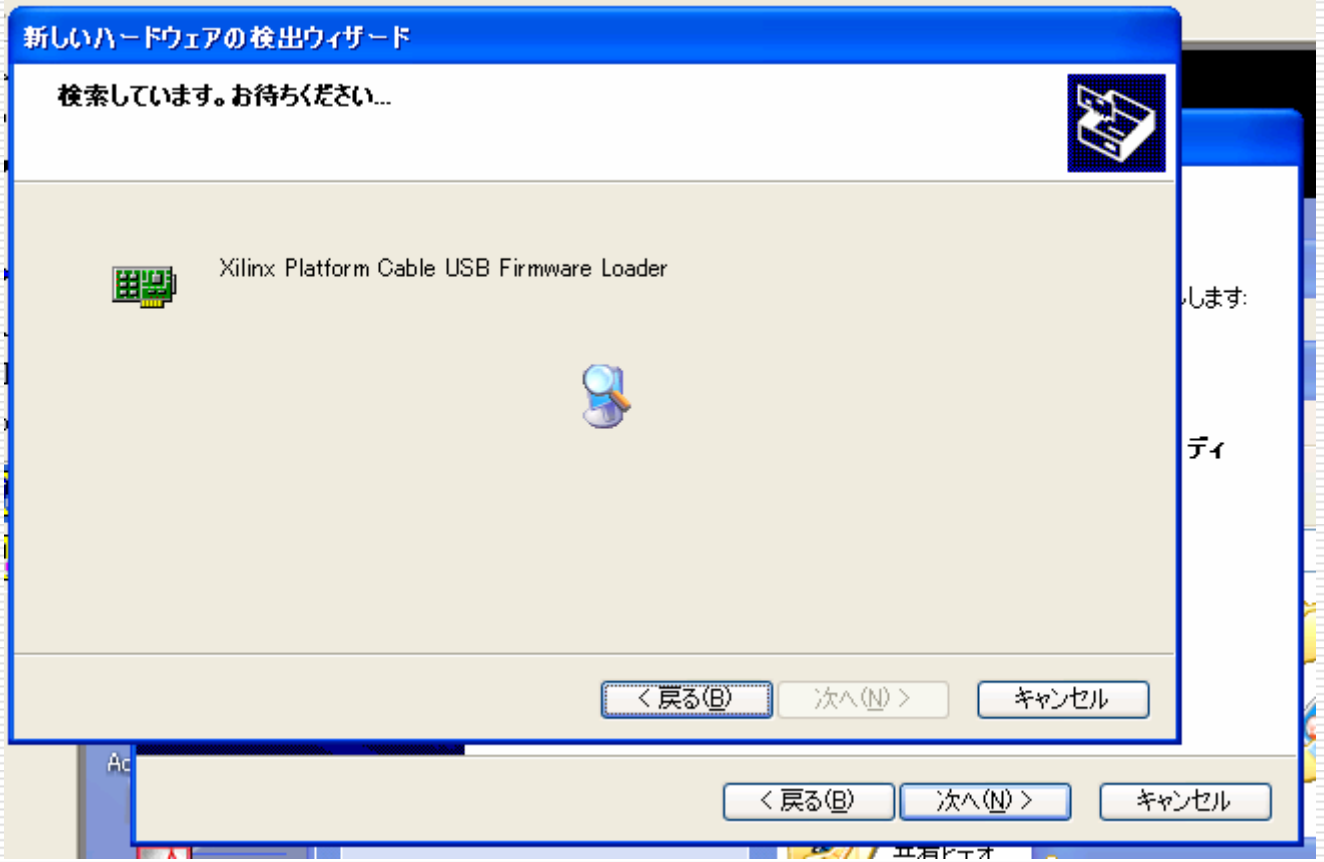
USBドライバのインストール



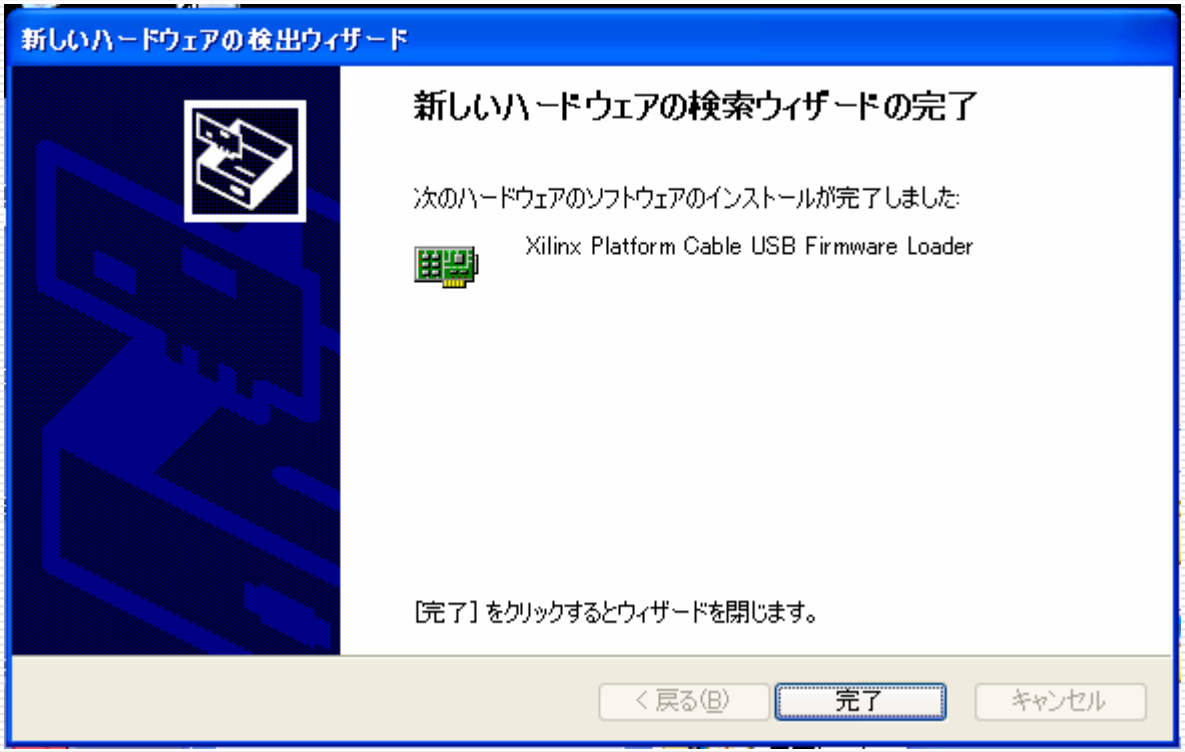
USBドライバのインストール



USBドライバのインストール

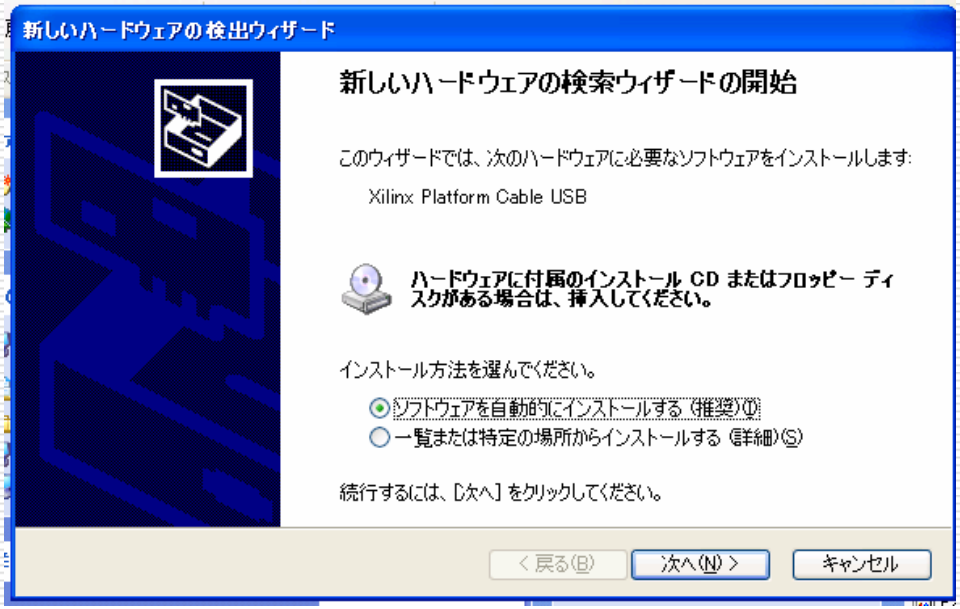


USBドライバのインストール



USBドライバのインストール

何度か同じ事を聞いてくるかもしれません

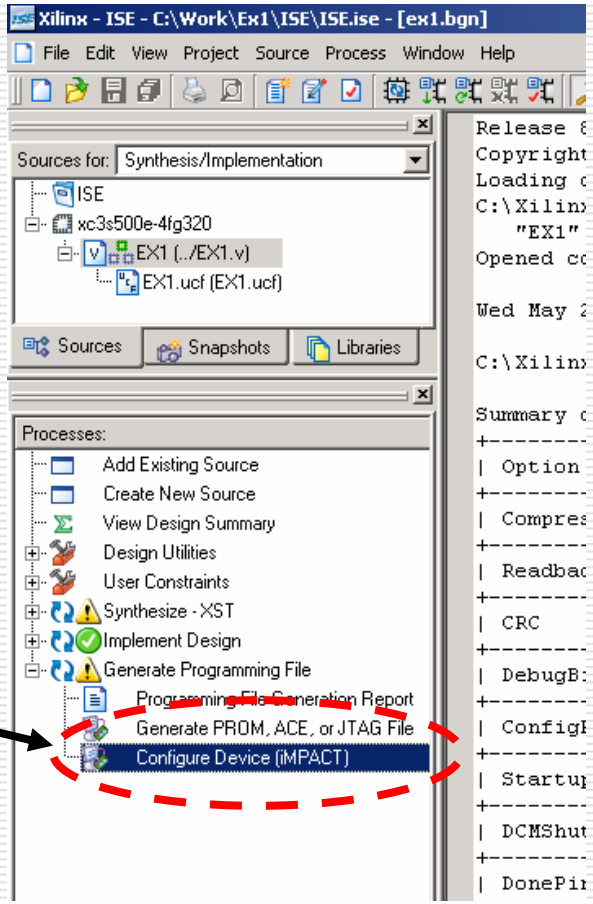


前回と同様にインストールしてください

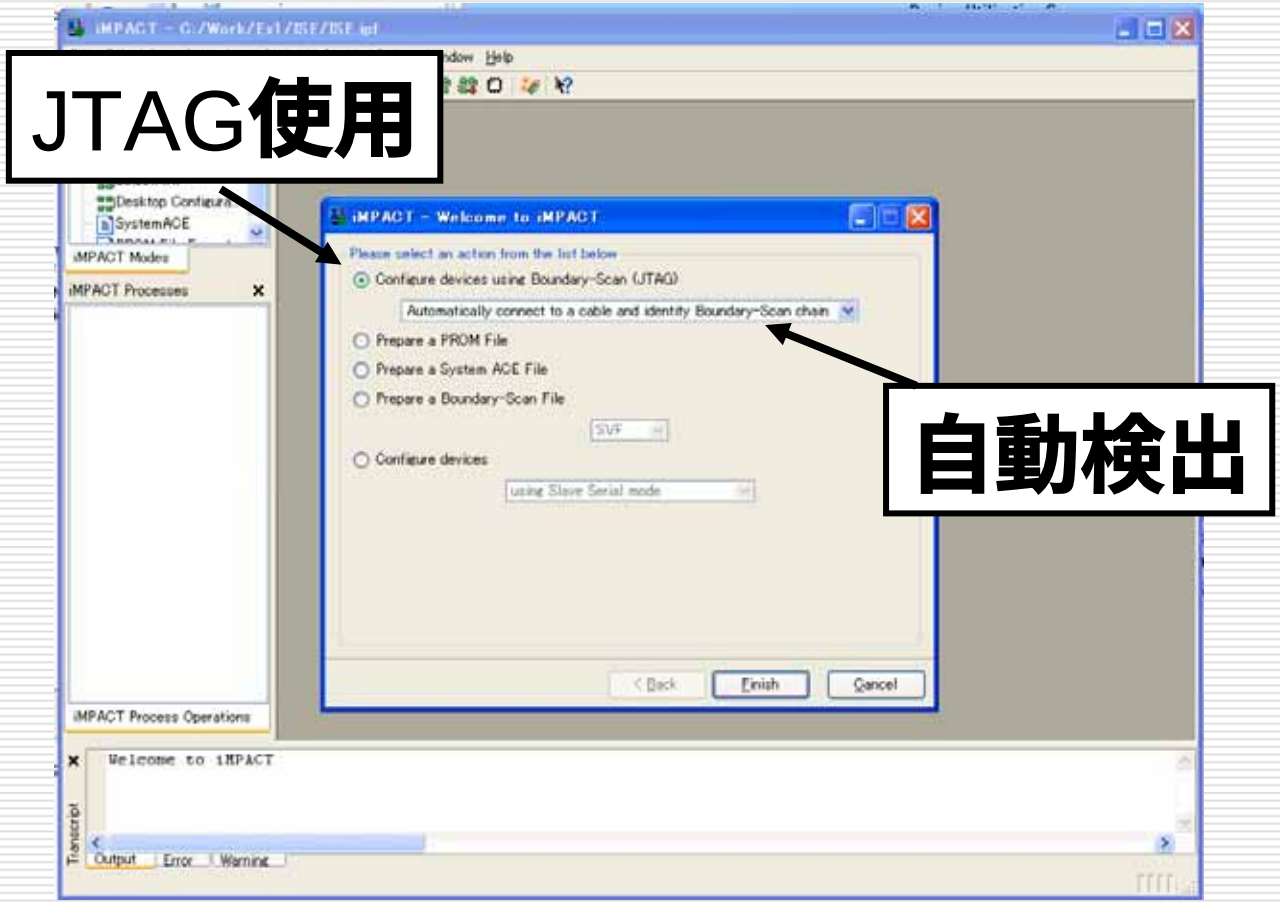
Configure device

ダウンロード･ツール
iMPACT

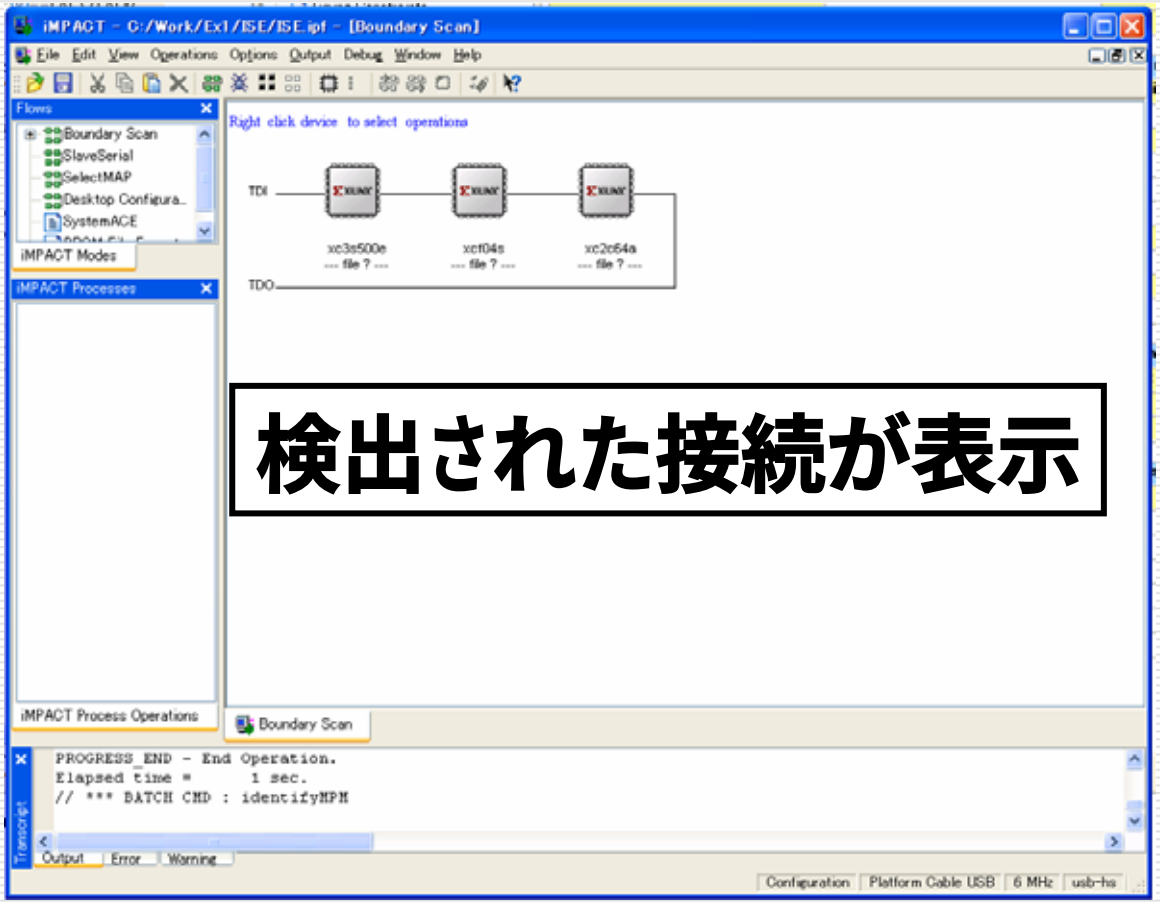
ダブルクリックして
iMPACTを立ち上げる



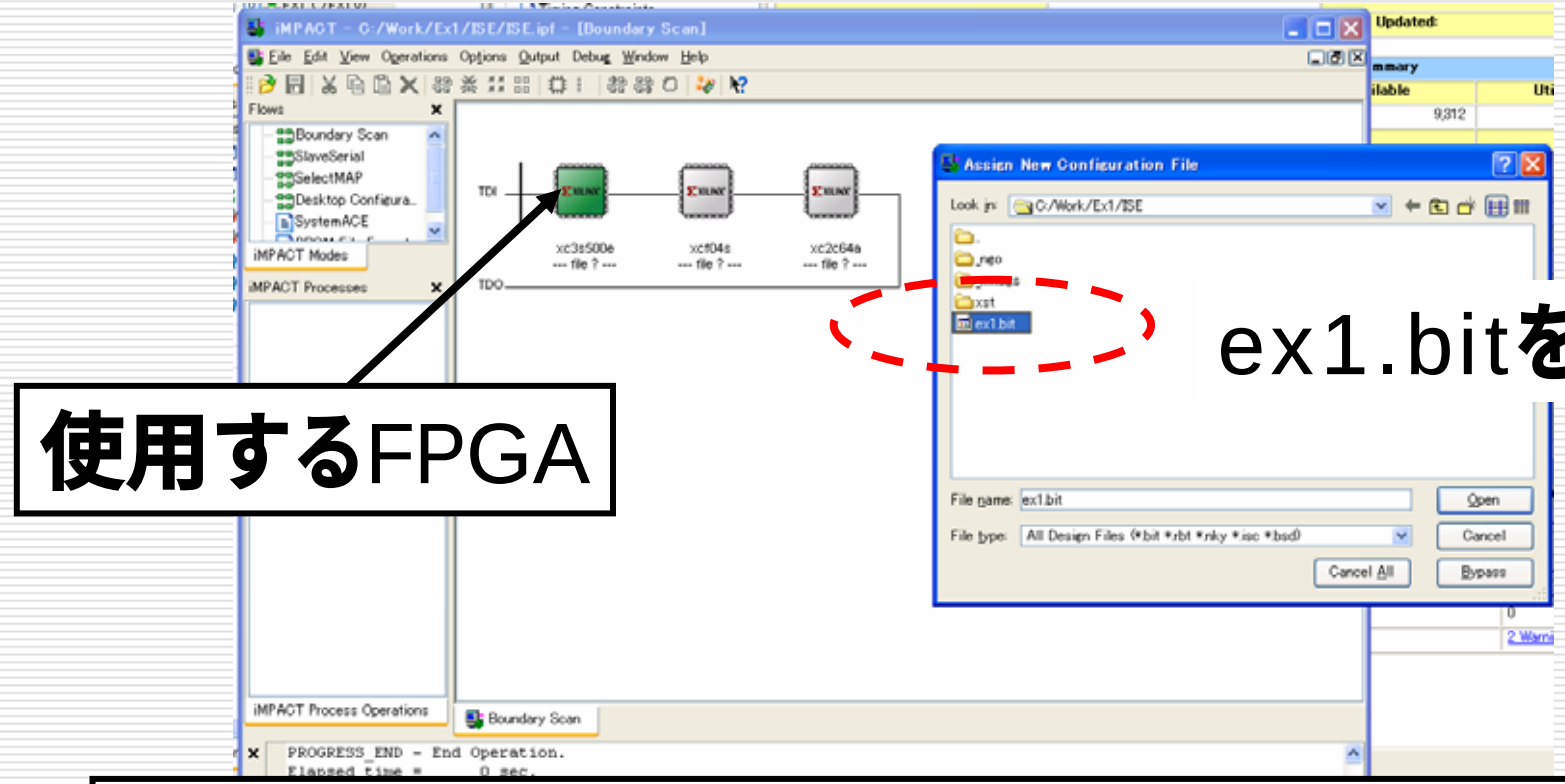
iMPACT起動画面


JTAG使用
自動検出
iMPACT - Welcome to iMPACT
Please select an action from the list below
Configure devices using Boundary-Scan (JTAG)
Automatically connect to a cable and identify Boundary-Scan chain
Prepare a PROM File
Prepare a System ACE File
Prepare a Boundary-Scan File
Configure devices
< Back
Finish
Cancel
iMPACT Modes
iMPACT Processes
iMPACT Process Operations
Welcome to iMPACT
Output
Error
Warning

Configure device

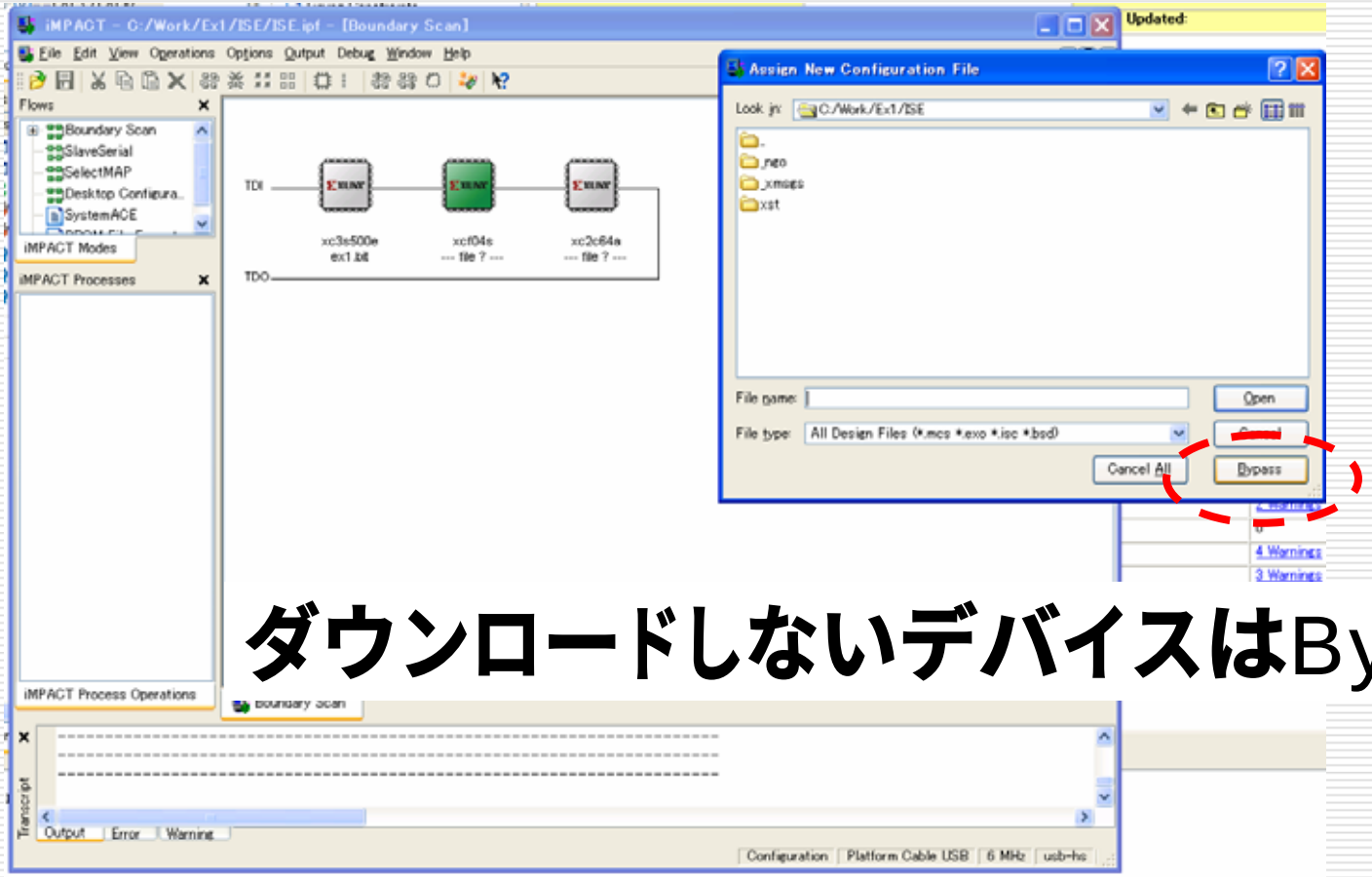


Configure device



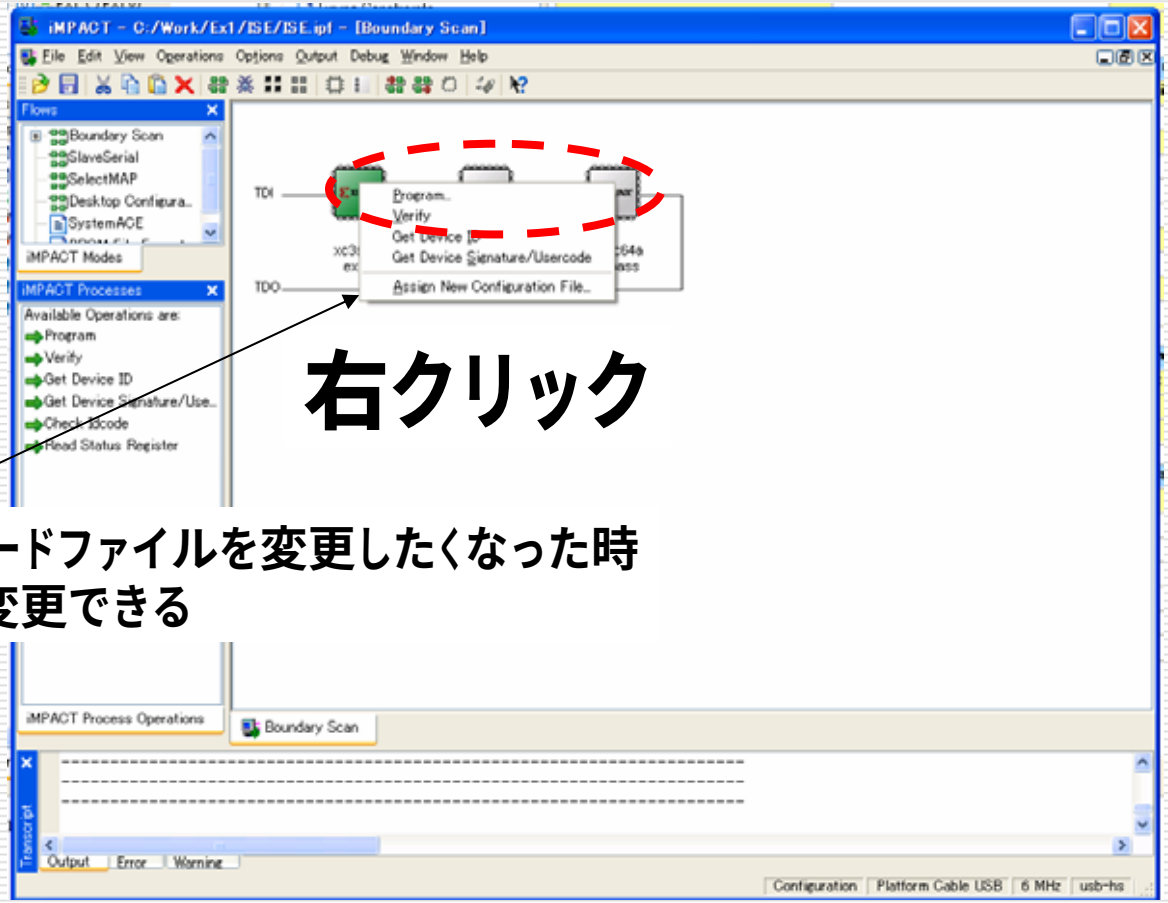
デバイス毎にダウンロードデータを指定する

Configure device



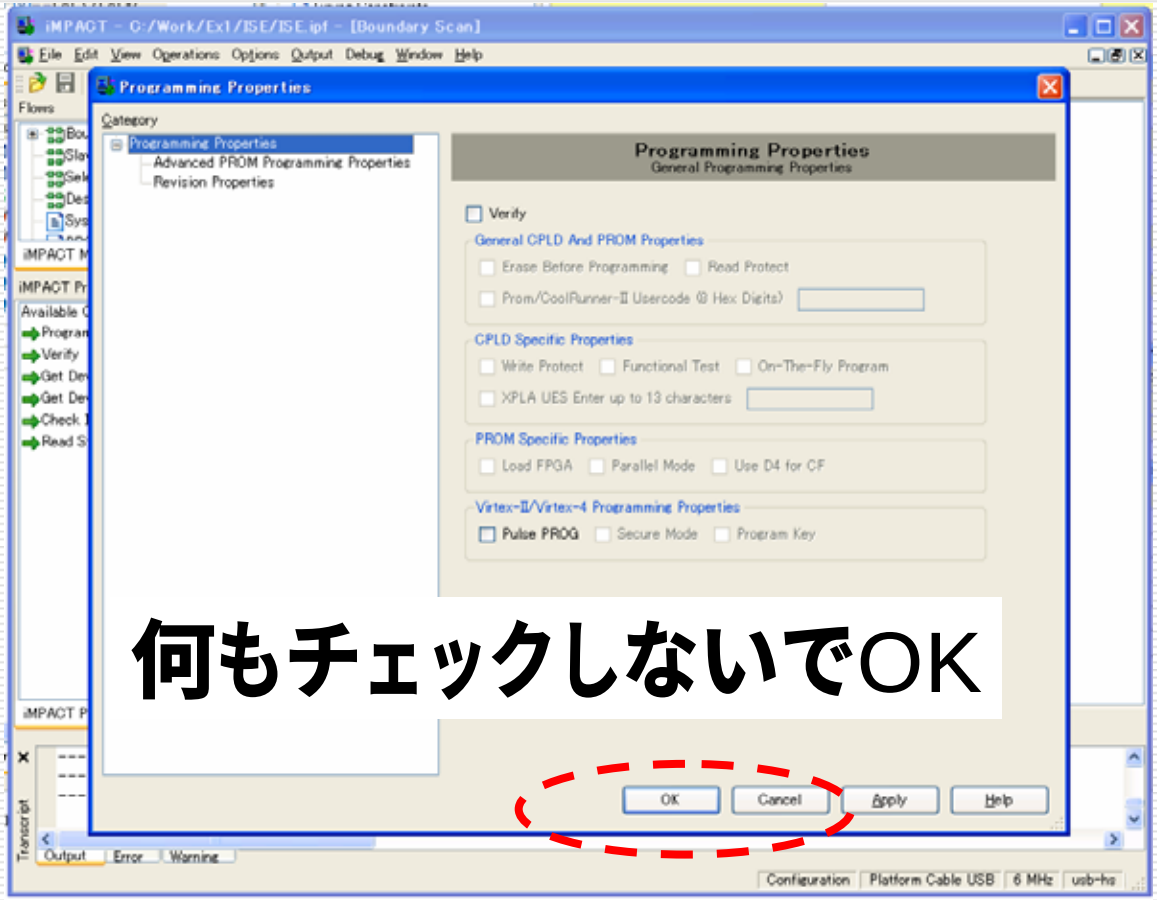
ダウンロードしないデバイスはBypass

Configure device

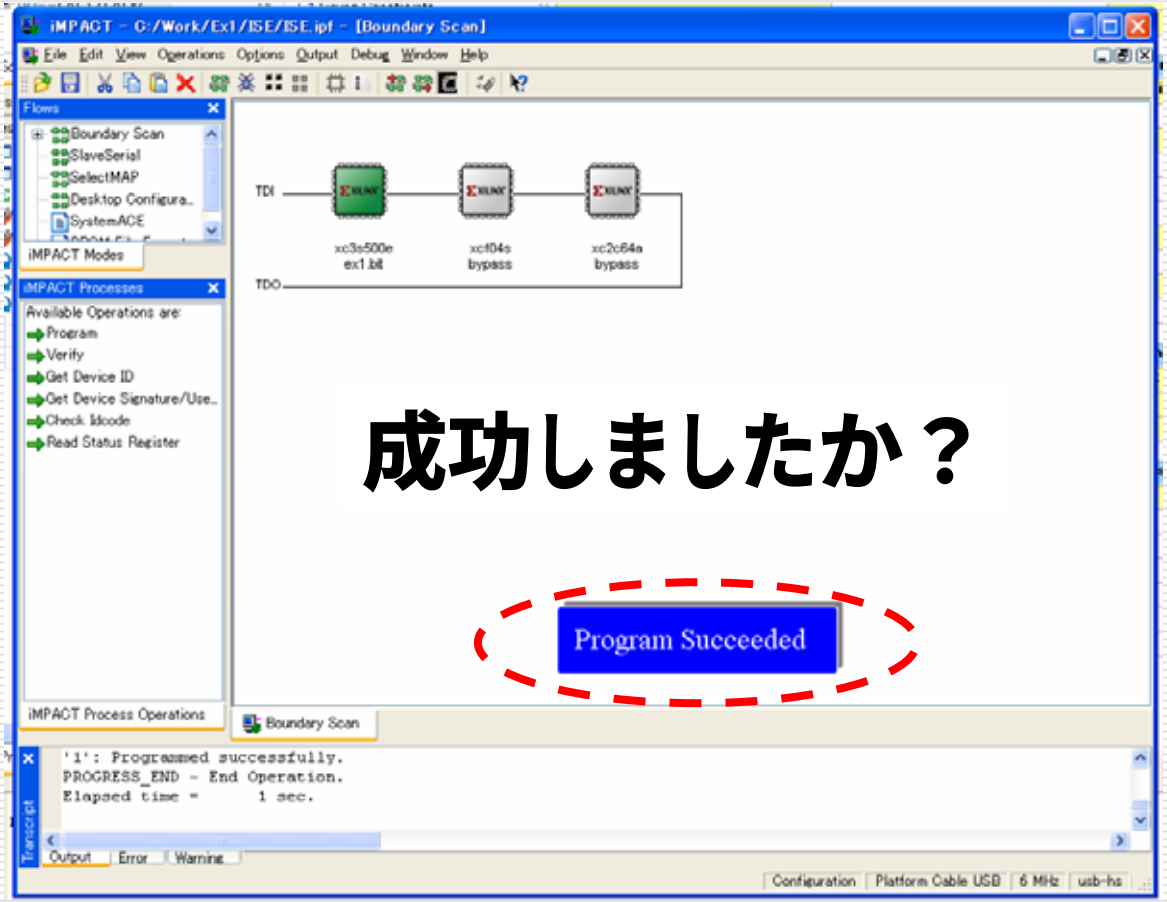


ダウンロードファイルを変更したくなった時
はここで変更できる

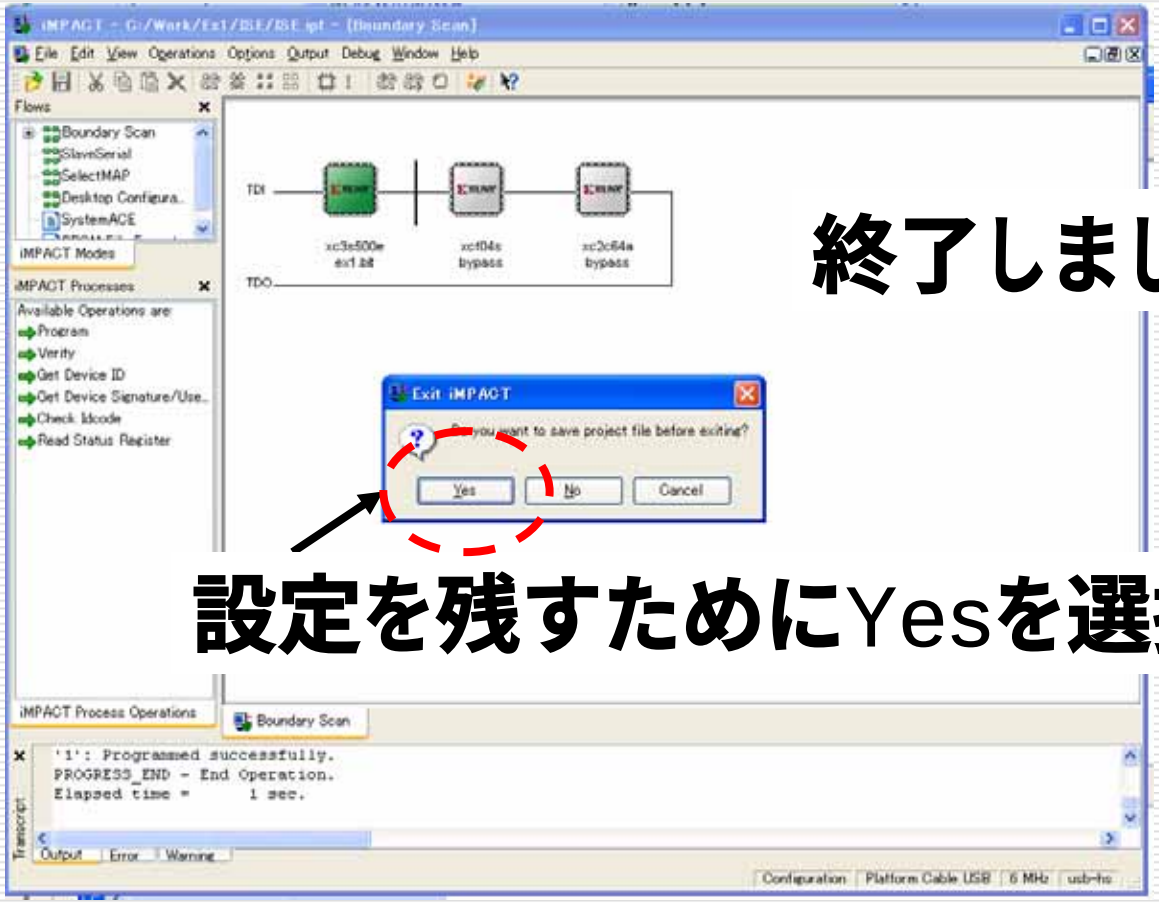
Configure device



Configure device



Configure device



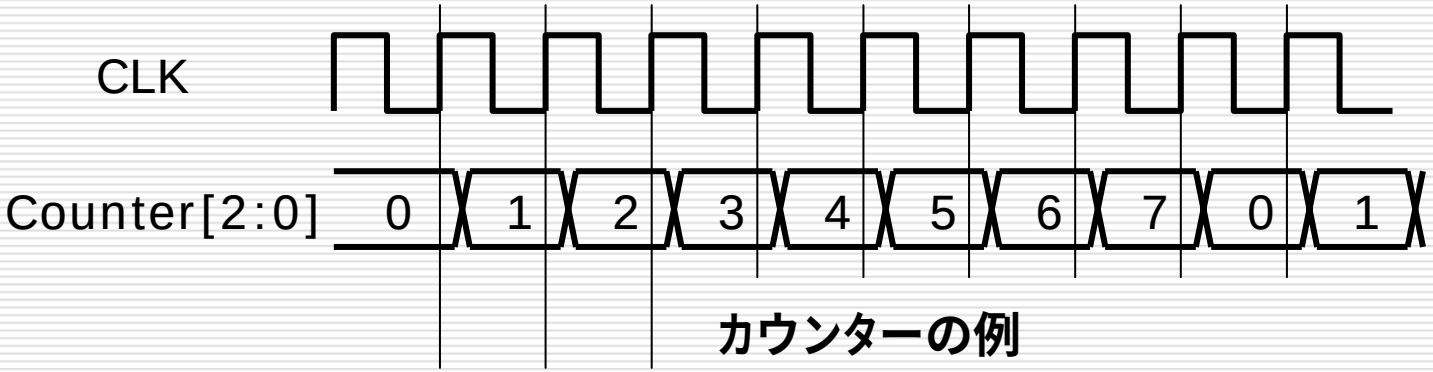
動作確認

- SWを色々動かして動作を確認してください
- 課題を実装して動作確認してください
  - 一つのSWのみがONの時、LED点灯
- 質問はありませんか？

同期回路

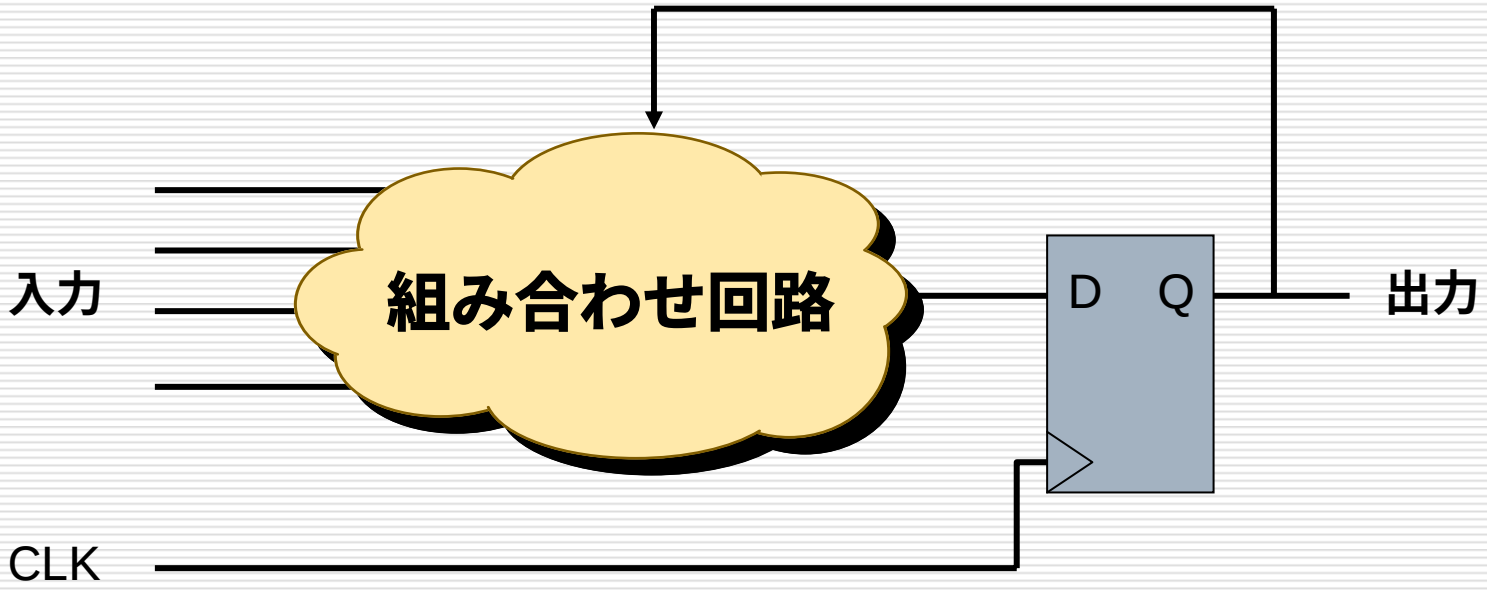
最初に

- FPGAは同期回路で使用する事を前提に設計されています。
  - FPGAで非同期回路を設計する時は注意
- 同期回路とは？
  - クロック（基準信号）で**のみ**状態が変化点



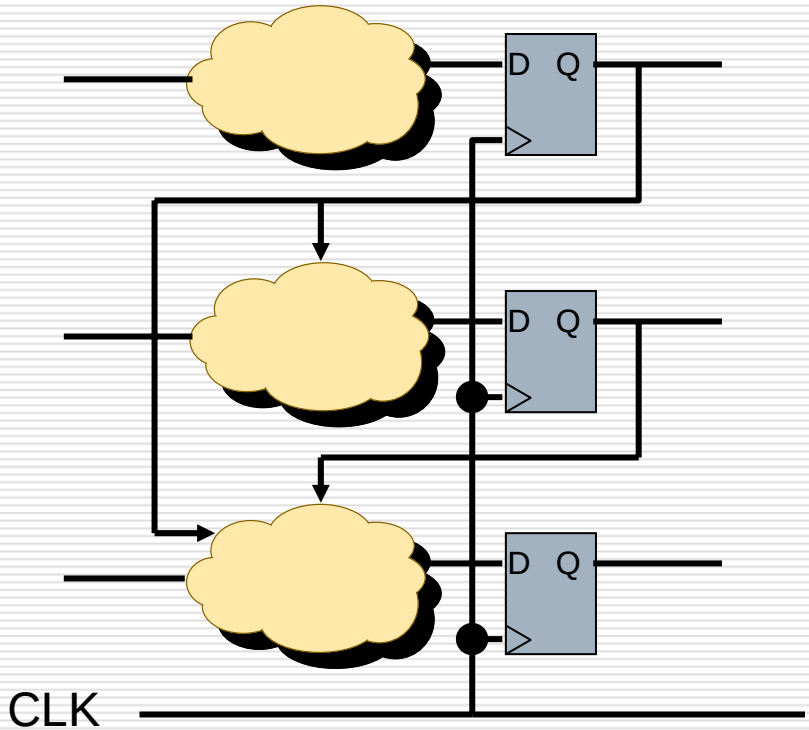
同期回路

CLKの周期毎に状態を考える事が出来る



一つ前の周期の出力を簡単に入力に使える
カウンタの場合：前出力の値に1を加える

同期回路



クロック単位で動作を考えれば良いので
タイミング設計が簡単

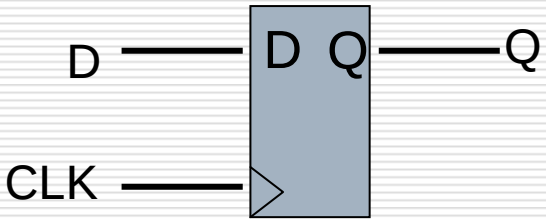
どんな風に設計しようが出力は
クロックで変化すると考えて設計できる

CLKが全てのDFFに入力されている

同期回路は

- 設計を簡単にする
  - タイミングはクロック周期で考えれば良い
  - 大規模回路の設計が出来る
- 順序回路（シーケンサー）が簡単に作れる
  - 前出力状態を入力として使用し動作する回路
- 広く使われています

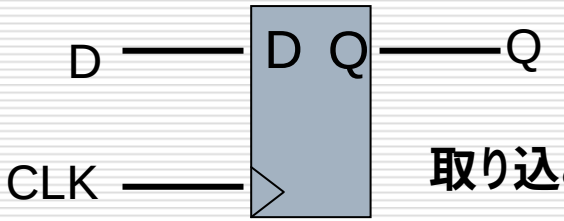
DFFのVerilogでの書き方




```
always @(posedge CLK) begin
    Q <= D;
end
```

CLKの立ち上がりの時は何時も～と書きます

DFFのVerilogでの書き方



立ち上がりエッジ　　クロックの名前

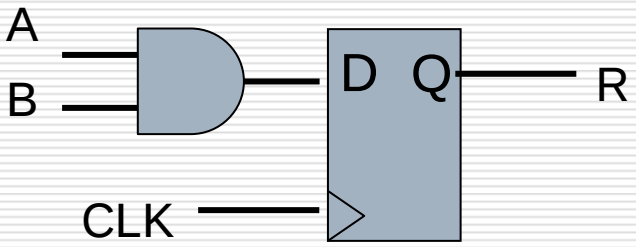
取り込み条件

```
always @(posedge CLK) begin
    Q <= D;
end
```

一般に立ち上がりを使います

“=”でなく”<=”な事に注意！

“=”でも動きますが、初めは”<=”のみ使用してください。
ちなみに私は”=”は使いません

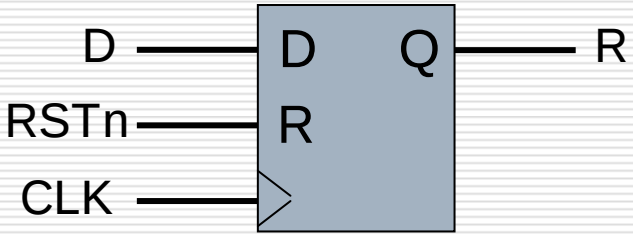



```
always @(posedge CLK) begin
    R <= A & B;
end
```

成立条件を書く
(組み合わせ回路の出力)

DFFの雛形、非同期リセット付き

リセット信号がLowになると、クロックに関係なく出力が0になる



リセット信号は負論理を使うことが多い

If文は()の中が真（“1”）の時に実行する部分

```
always @(posedge CLK or negedge RSTn) begin
    if(~RSTn)begin
        R <= 1'b0;
    end else begin
        R <= A & B;
    end
end
```

定数のみ書く！論理式は書かない！

初めはリセットで初期状態に戻る
このタイプのDFFを使用するようにしてください！

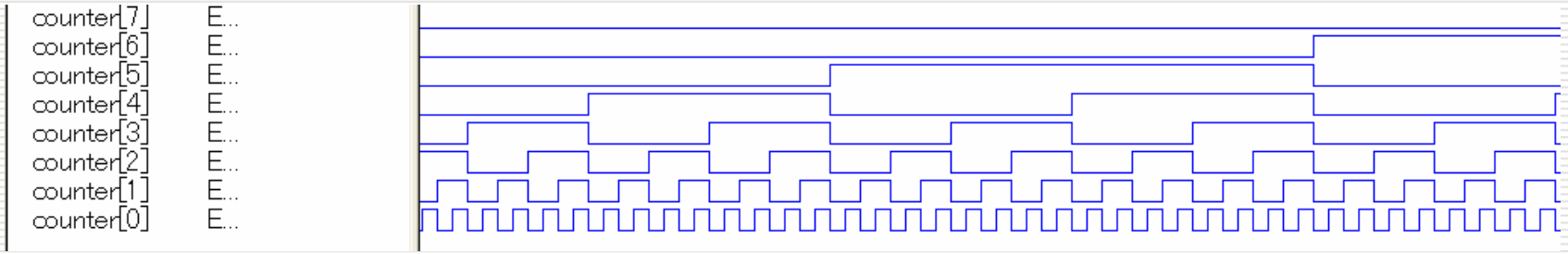
カウンターの書き方

```
always @(posedge CLK or negedge RSTn) begin
    if(~RSTn)begin
        counter[31:0] <= 32'd0;
    end else begin
        counter[31:0] <= counter[31:0] + 32'd1;
    end
end
```

再度Ex1.vを見てください

77行目のコメントアウトされている部分

カウンターの動作



後で実習するシミュレーション結果

上位ビットの周期は下位ビットの2倍になっている

分周されている

Heart beatを作りましょう

クロックが正常に供給されている時に LED[7]が点滅する回路

counter[23]をLED[7]に接続

counter[23]の周期=20ns x 2^{24}〜0.35s
(50MHzの周期＝20ns)

変更内容

- クロック
  - オシレータを使用：信号名OSC
  - 内部で信号名をCLKと変更して使用
    - Assign文を使う
- リセット
  - BTN Southを使用（押した時にリセット）
  - ピン信号名はPUSH_SW
  - 内部で信号名をRSTn（Lowでリセット）に変更して使用
  - 負論理に変換する事を忘れないで下さい
- PACEでピンアサインするのを忘れないで下さい
- ちゃんと点滅しますか？リセットはされますか？

論理シミュレーション

シミュレーションを何故やるのか？

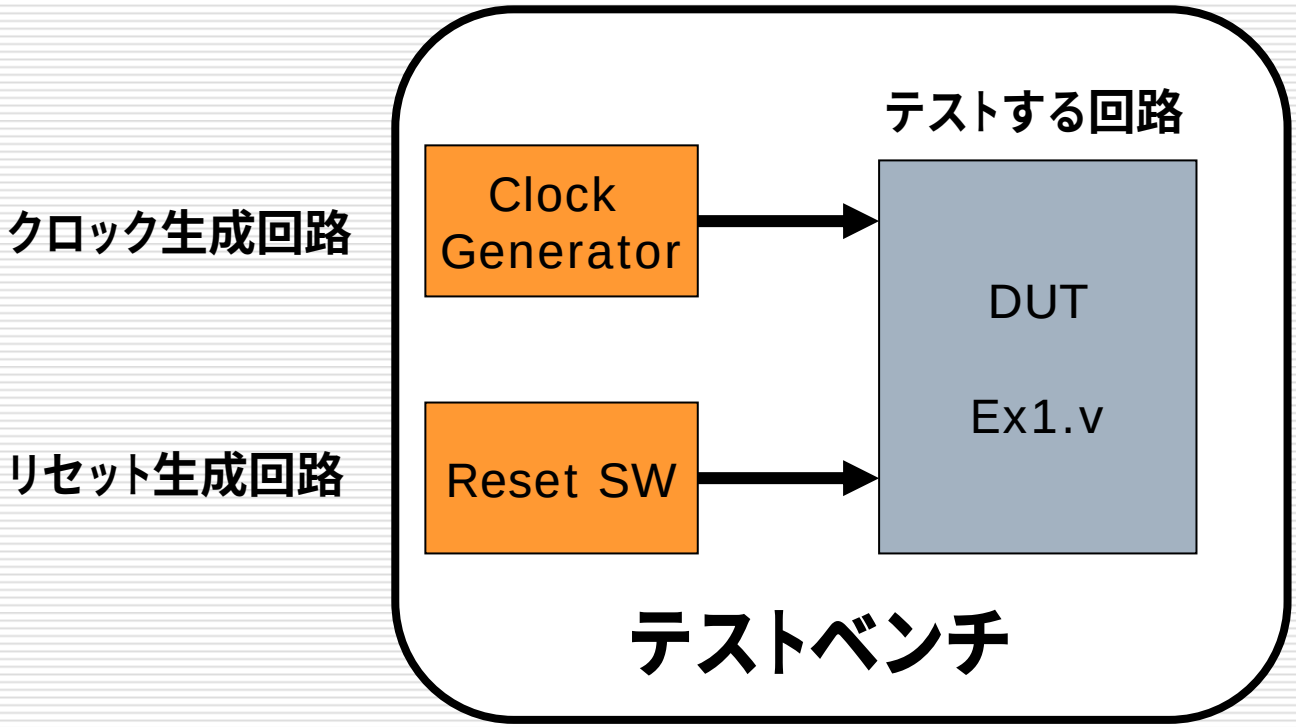
- 観測できる信号はピンのみ
  - 内部の信号は専用回路が必要
  - それでも全てを見ることはできない
- 実機を使わずに動作テストが出来る
  - 基板を作る前に仕様、動作検討ができる
  - 動かす前にバグを見つける事ができる
  - 内部の信号を全て観測できる
- 大規模回路の完成度を上げる
  - 動作検証は非常に大変
    - 膨大な数の信号線、ゲート

論理シミュレーションとは？

- 純粋に論理動作をシミュレーション
- 伝播遅延などアナログ量を無視
- 遅延を考慮するシミュレーションは遅延（ゲートレベル）シミュレーションと呼ばれる
  - 同期回路で設計する場合、ほとんどの部分は論理シミュレーションでよい
  - タイミングチェックはISEがやってくれる
    - 後で説明

今回のシミュレーション方法

クロックとリセットを供給しないと動作しない



シミュレーション用に回路などを作成する必要がある

一般的な方法

- テストベンチを使う
  - 実際の仕様環境をシミュレーションする
  - 周辺機器も同時にシミュレーション
    - シミュレーションモデル
- 実際の動作環境に近い動作で確認
- しかし、問題も・・・・

シミュレーションの問題

- 長時間テストが難しい
  - シミュレーション時間が長い、使用リソースが膨大
  - 例えば50MHz動作回路の10秒間テスト
    - 500M周期のテストが必要
  - と言ってもやらない訳にはいかないので工夫が必要
    - 後のRS232Cの時に説明
  - 重要なのは設計者が気になる状態をテスト
    - この状態の洗い出しで完成度、テスト時間が変わる
- テストベンチを間違えると全て間違える

テストベンチ

- 4つの構成要素で書く必要が無い
  - シミュレーションのみで使用
  - 実際のデバイスには実装されない
- ソフトウエアに近い記述方法で書ける
  - 簡単に動作を記述できる
    - Behavior model
- 注意
  - 複雑なテストベンチではモデル自身バグが出るので注意

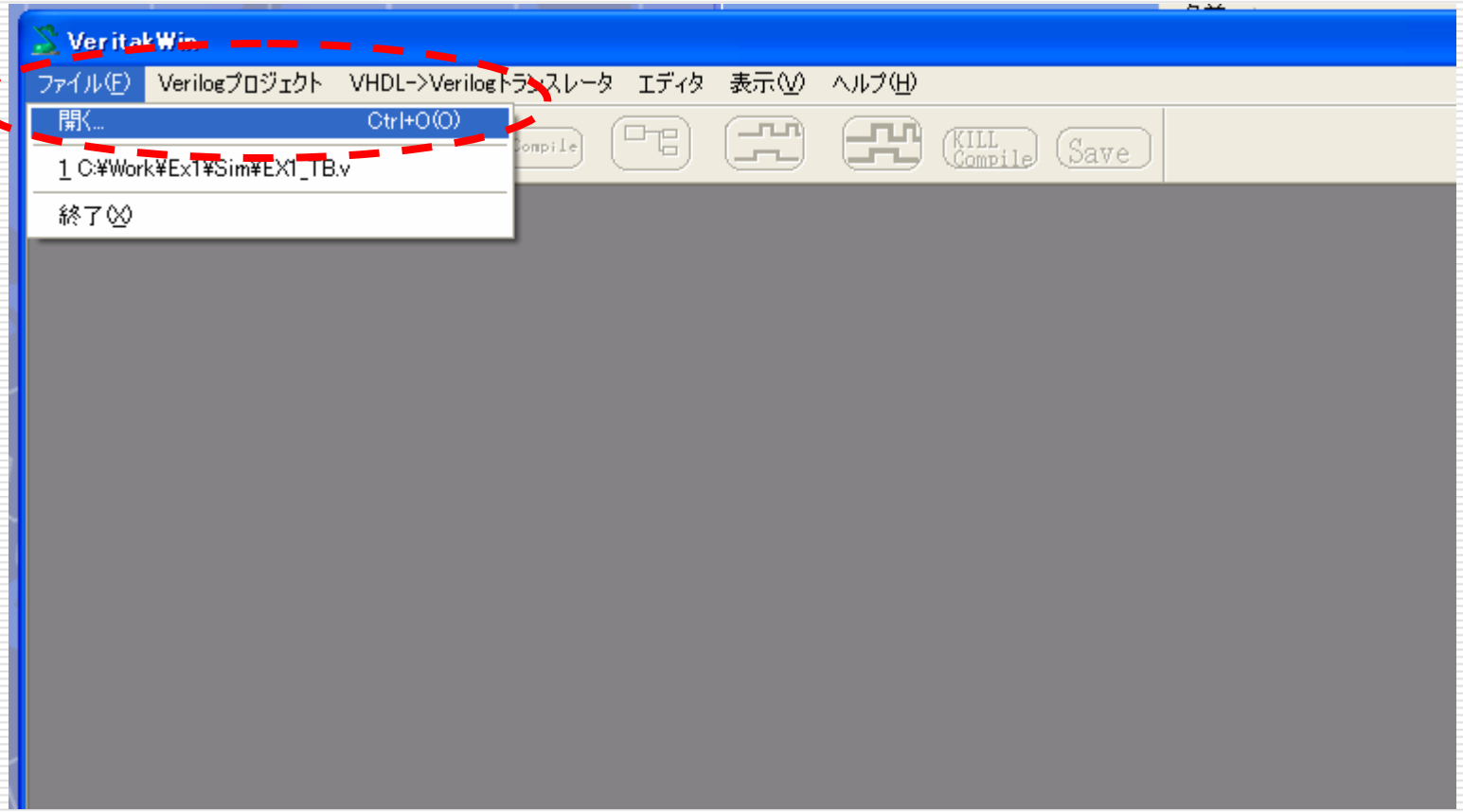
テストベンチについて

- 実際のファイルを見てみましょう
- EX1_TB.vを開いてください

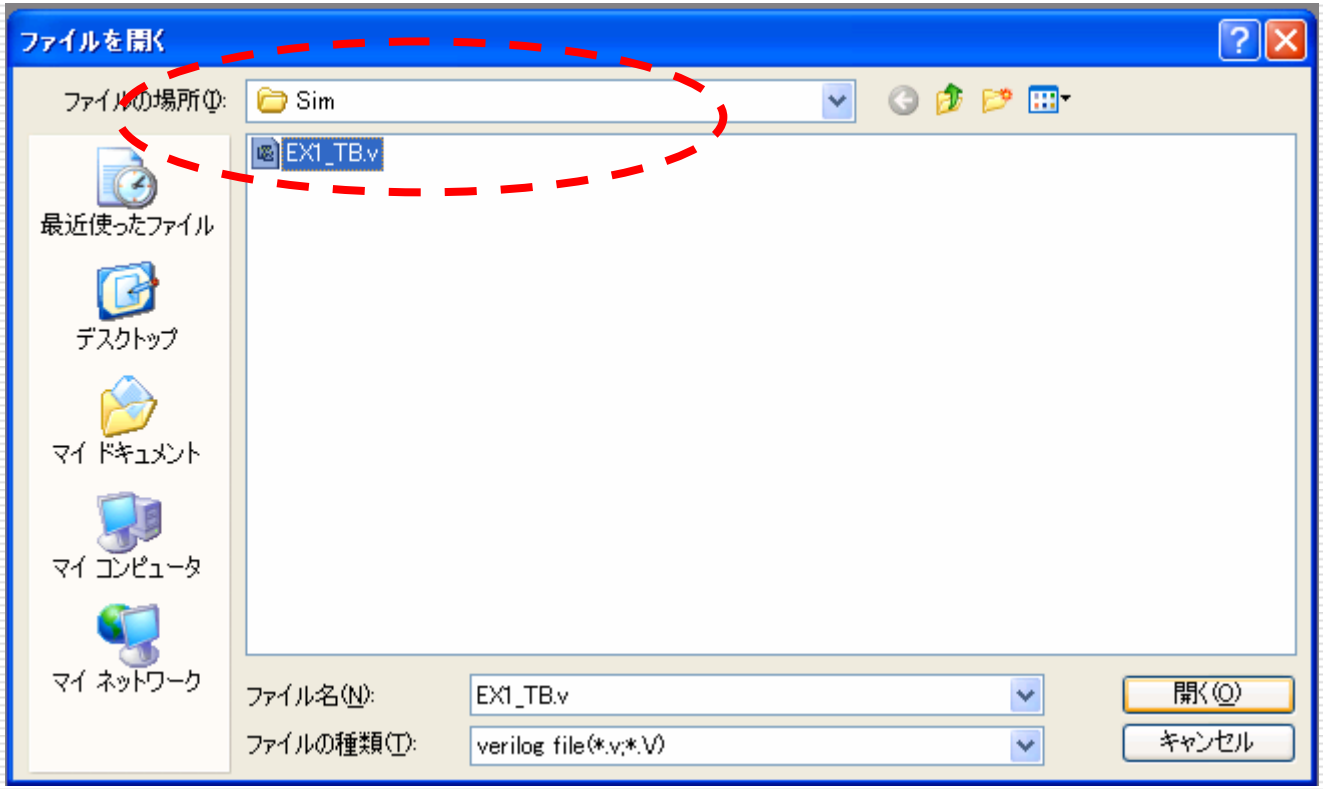
シミュレーターを動かしましょう

- 作ったカウンターをシミュレーションします
- veritakを立ち上げてください
  - ウインドウが現れましたか？

Veritak操作


VeritakWin
ファイル(F)
Verilogプロジェクト
VHDL->Verilogトランスレータ
エディタ
表示(V)
ヘルプ(H)
開く...
Ctrl+O(O)
1 C:¥Work¥Ex1¥Sim¥EX1_TB.v
終了(X)
KILL Compile
Save

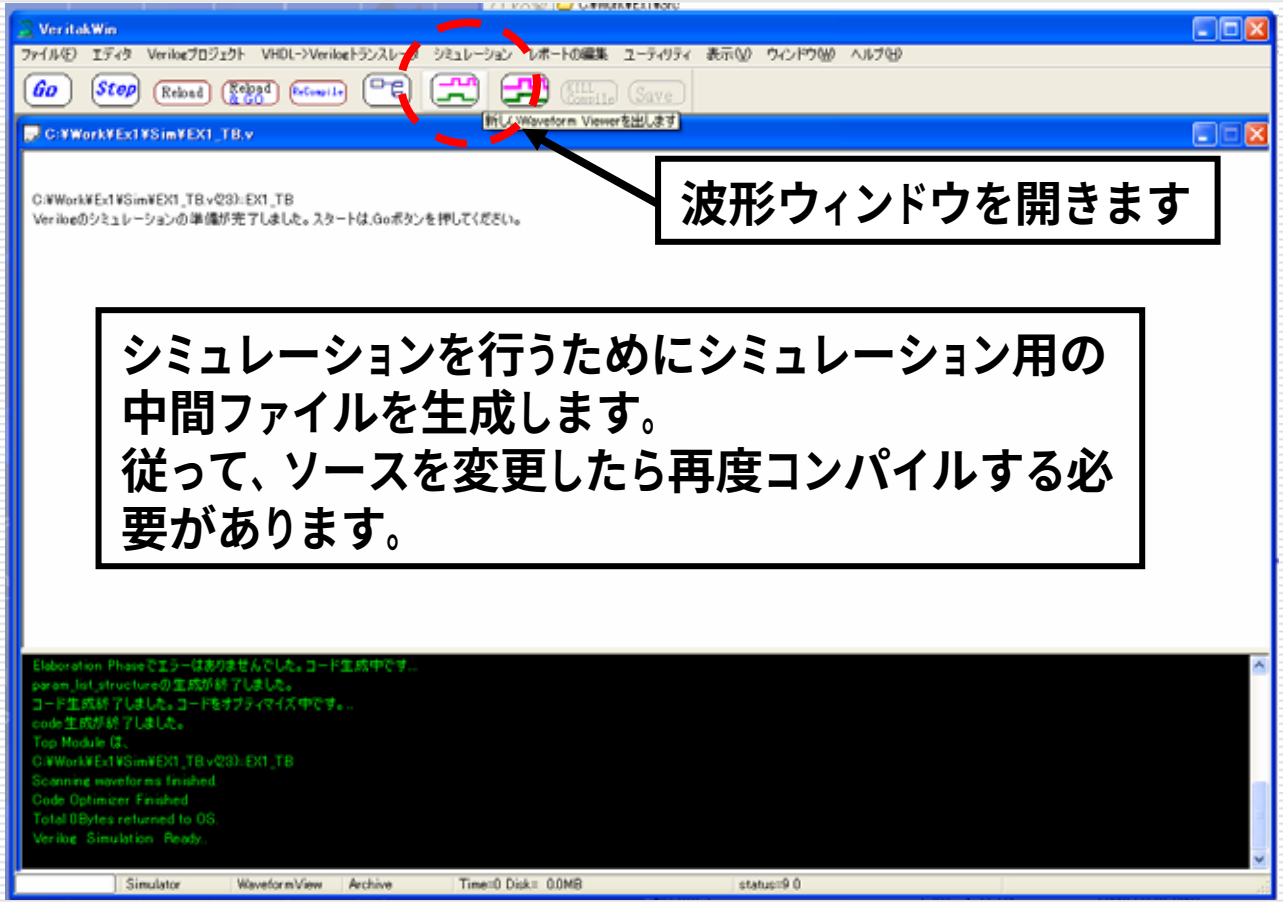
テストベンチ・ファイルの選択



テストベンチとは

- FPGAをシミュレーションするために必要な環境が記述されているファイル
- 必要な入力、出力が記述されている
  - OSC
  - PUSH_SW
  - など
- EX1.vはEX1_TB.vの中から呼び出します
- 後で改めて説明します

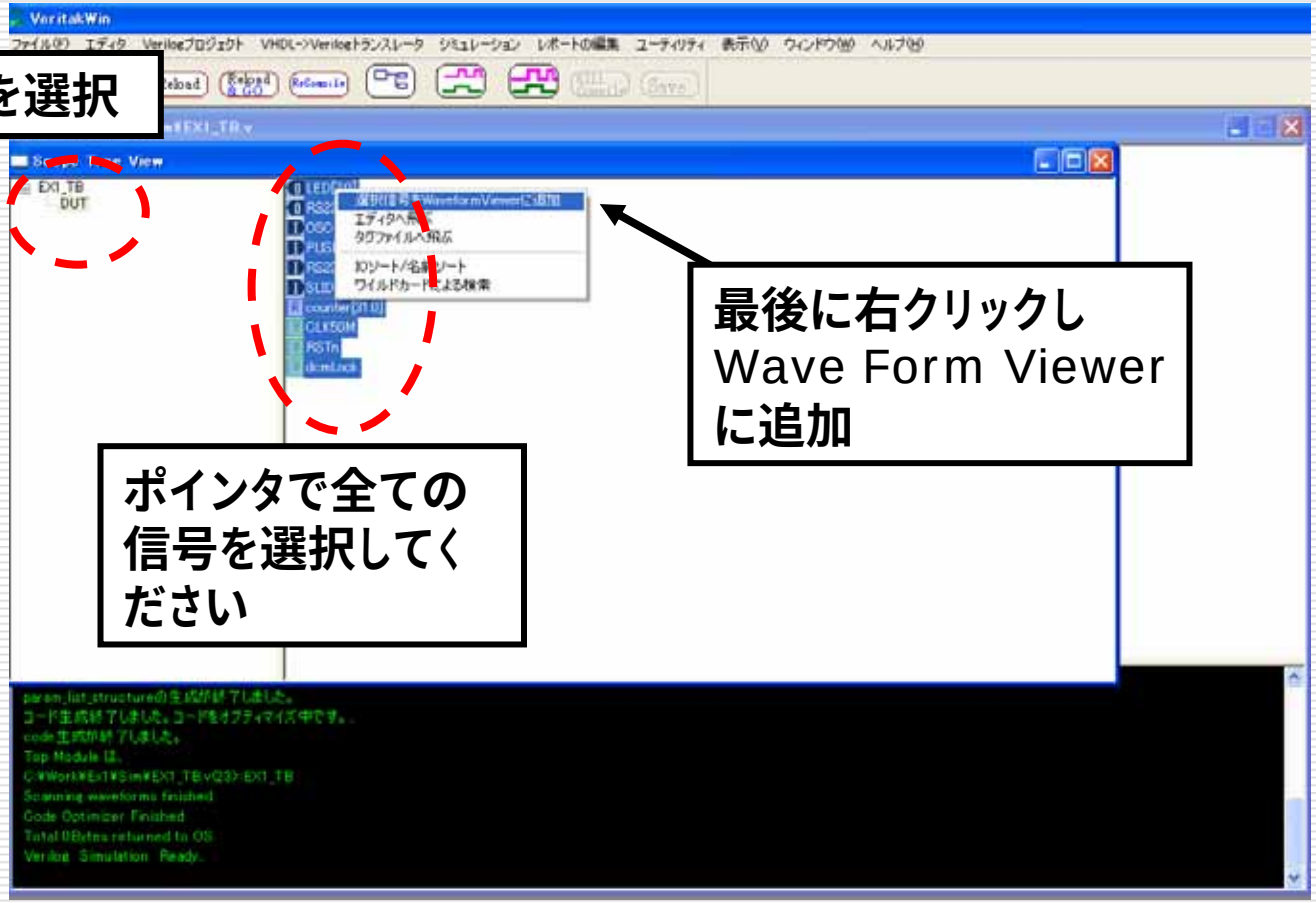
コンパイル終了



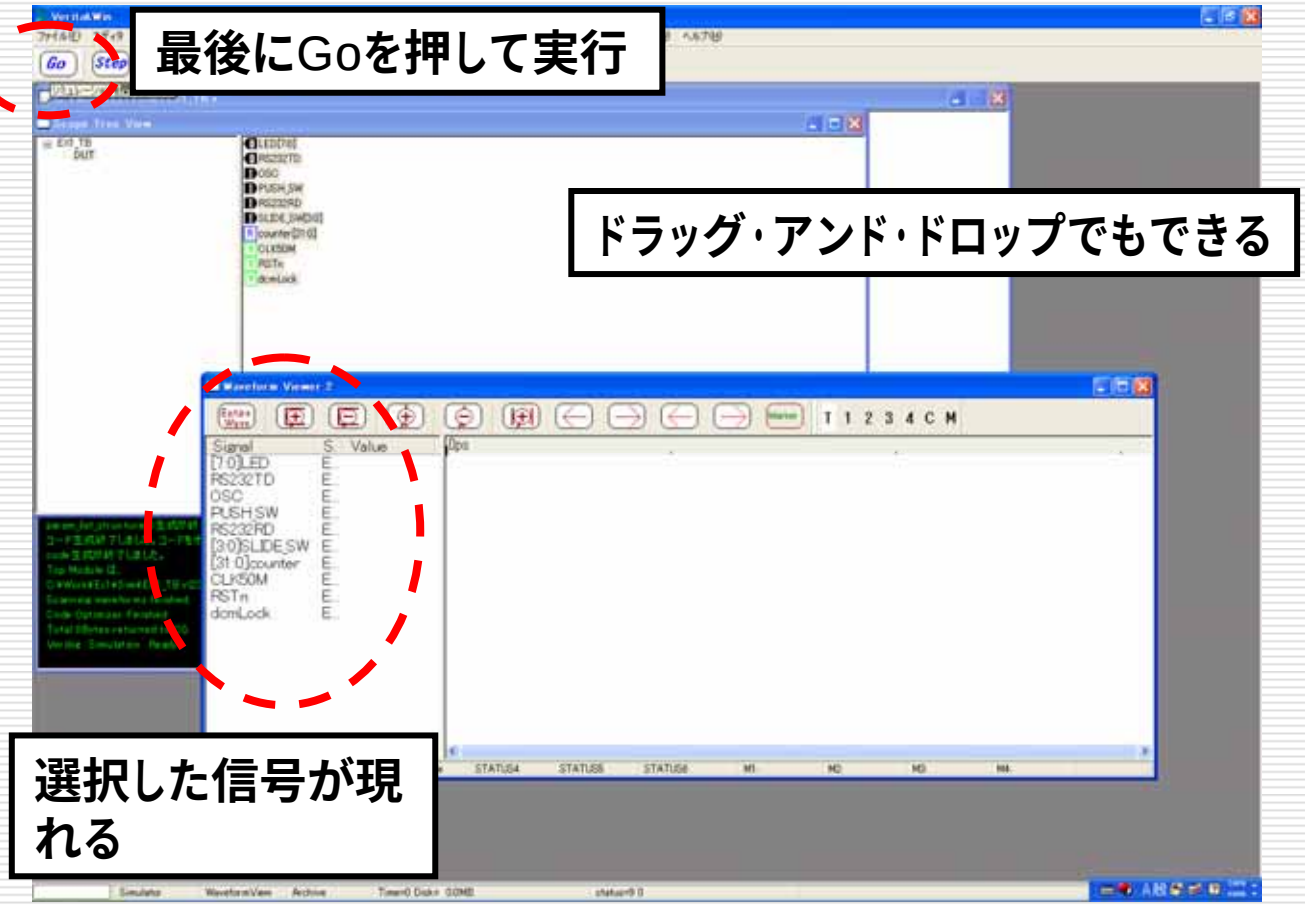
波形ウィンドウを開きます

シミュレーションを行うためにシミュレーション用の中間ファイルを生成します。
従って、ソースを変更したら再度コンパイルする必要があります。

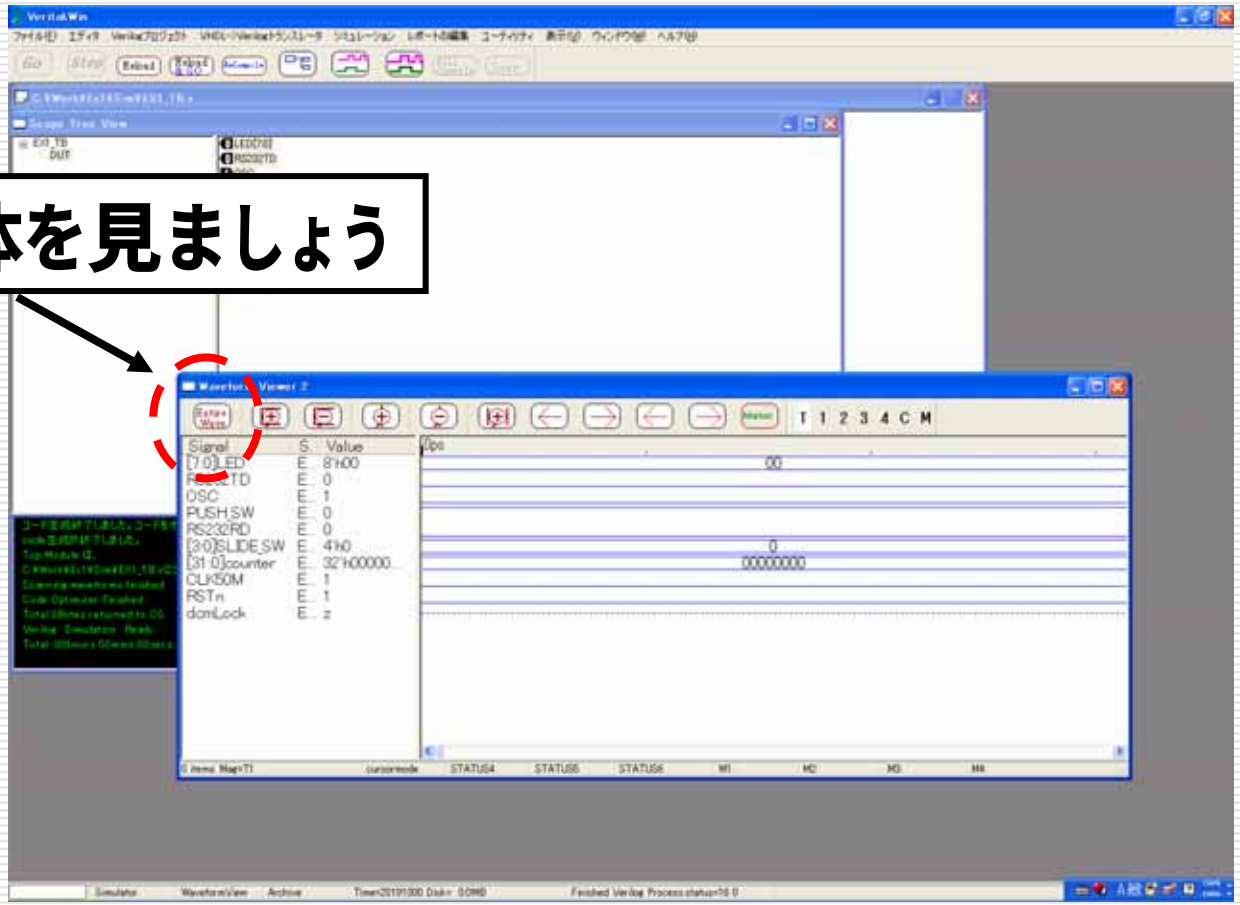
信号の選択


DUTを選択
ポインタで全ての信号を選択してください
最後に右クリックしWave Form Viewerに追加

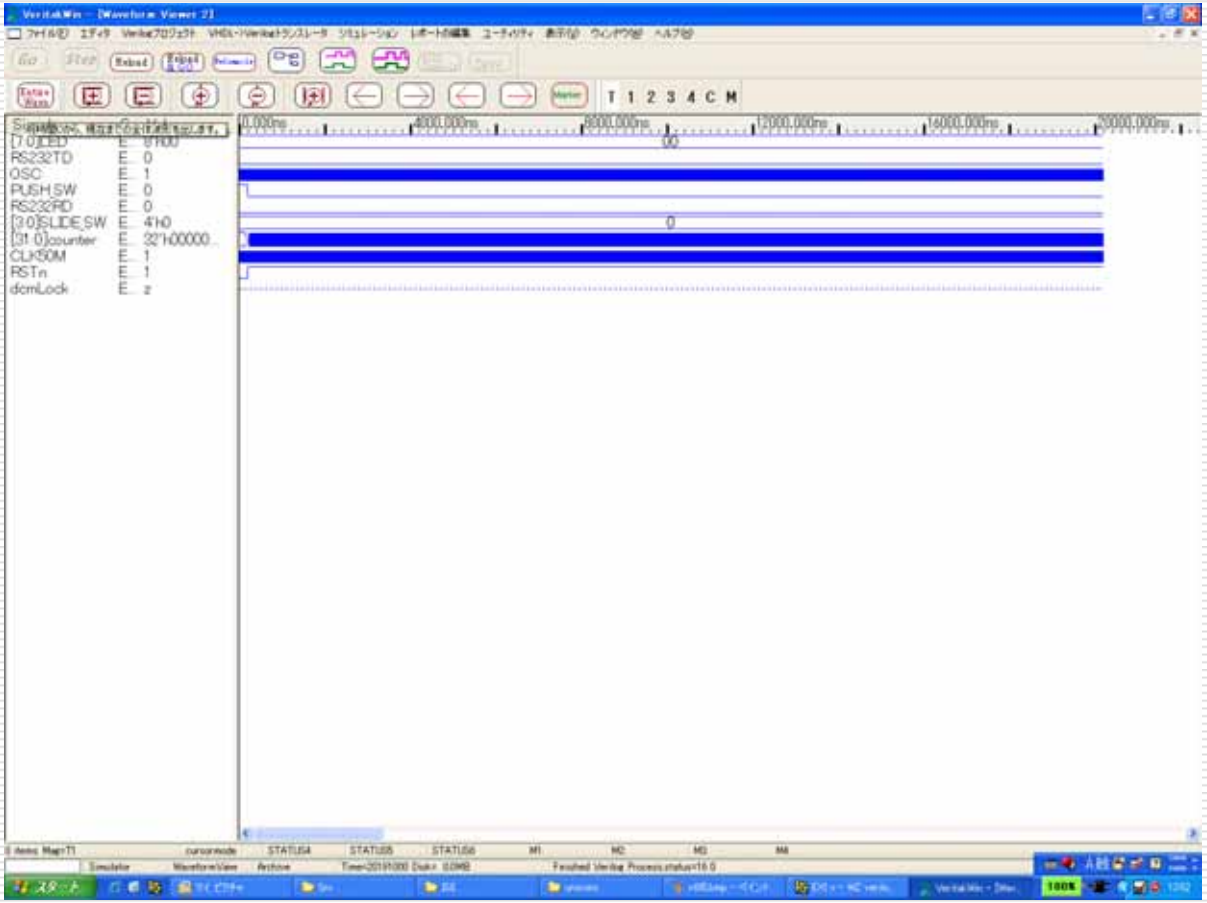
信号の選択



実行



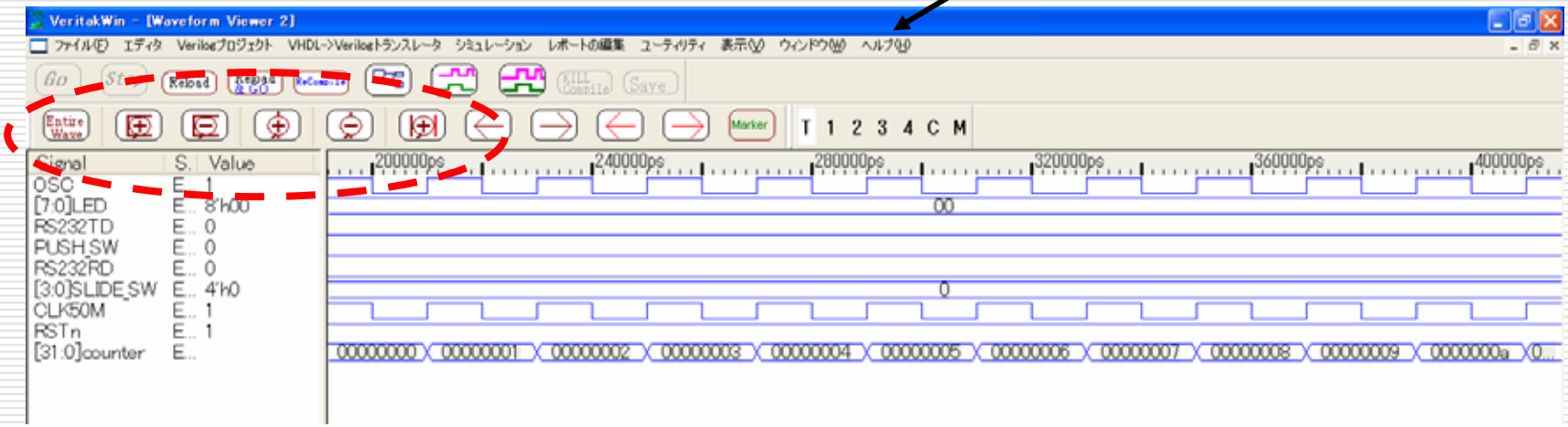
結果



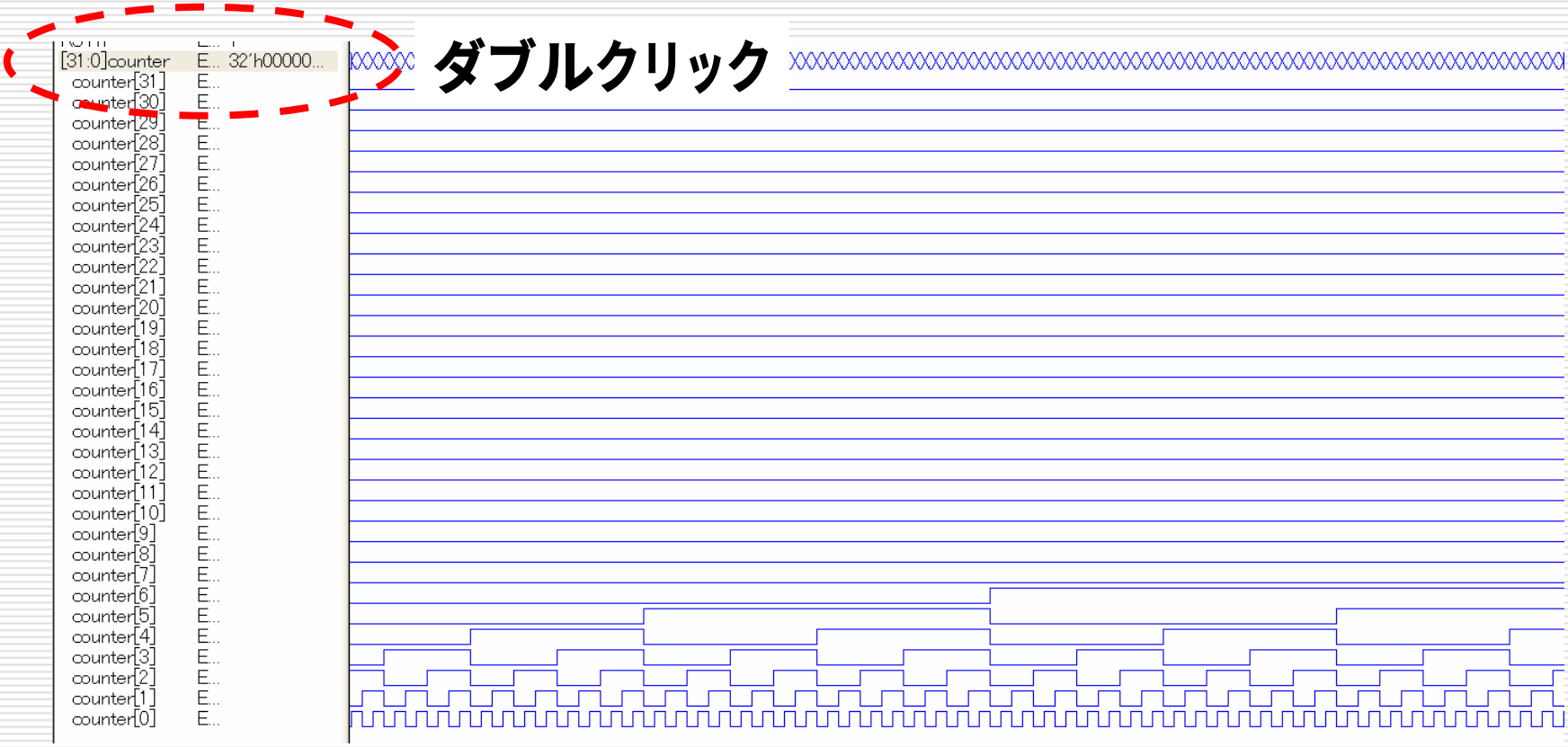
拡大、縮小

詳細はヘルプのチュートリアルを参照

いろいろ触ってみて下さい



バス信号の展開表示


ダブルクリック
[31:0]counter E... 32'h00000...
counter[31] E...
counter[30] E...
counter[29] E...
counter[28] E...
counter[27] E...
counter[26] E...
counter[25] E...
counter[24] E...
counter[23] E...
counter[22] E...
counter[21] E...
counter[20] E...
counter[19] E...
counter[18] E...
counter[17] E...
counter[16] E...
counter[15] E...
counter[14] E...
counter[13] E...
counter[12] E...
counter[11] E...
counter[10] E...
counter[9] E...
counter[8] E...
counter[7] E...
counter[6] E...
counter[5] E...
counter[4] E...
counter[3] E...
counter[2] E...
counter[1] E...
counter[0] E...

補足

- Veritakの制限
  - 一度に一定数以上シミュレーションできない
  - 「パターン数制限により停止しました」と表示される
  - 続きはもう一度Goでシミュレーション可能

補足

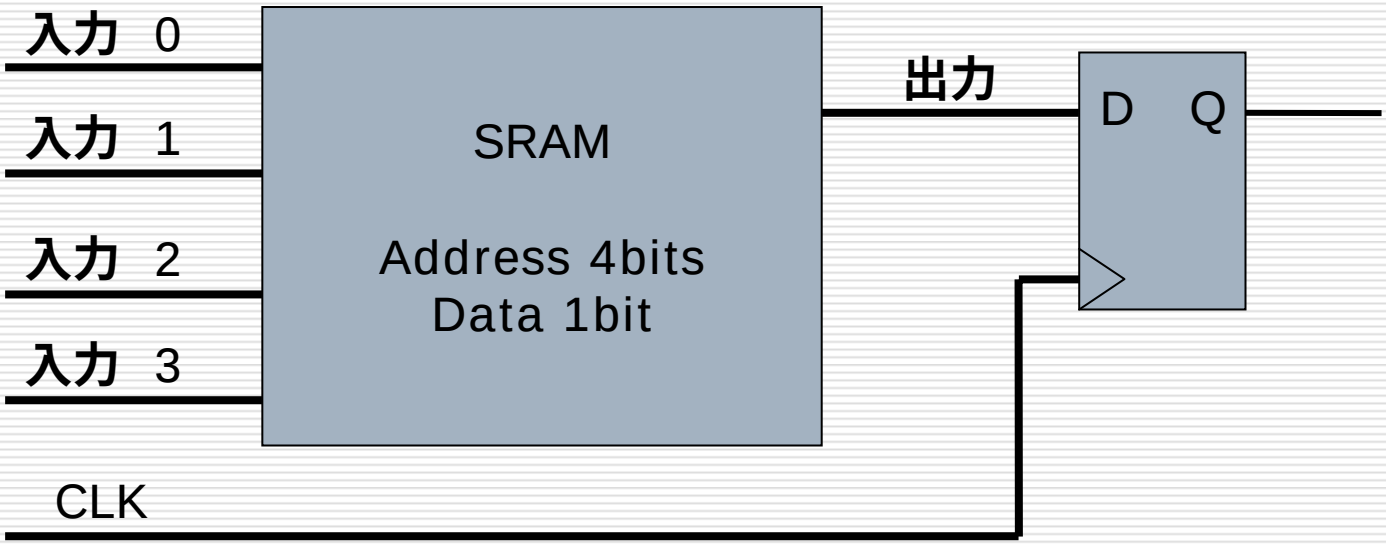
- 負論理の信号の書き方
  - 負論理、正論理とは？
  - RSTnの様にLowレベルでリセットの意味が有効になる信号を負論理の信号を言います。
  - 慣習で/RST, RSTn, RSTbなどと書きます
- RSTn　= ~PUSH_SW & dcmLock;
- RSTn　= ~(PUSH_SW | ~dcmLock)
  - 右辺全体に~を付けると左辺＝LOWの条件になる
  - PUSH_SW＝ONの時またはdcmLock＝OFFの時にLOWを出力

FPGAの構造

- **動作原理**
- SliceとCLB

動作原理

基本はSRAM＋DFF



RAMなので入力の全ての組み合わせの出力をテーブルを持たせる事ができる。

4入力ANDの時

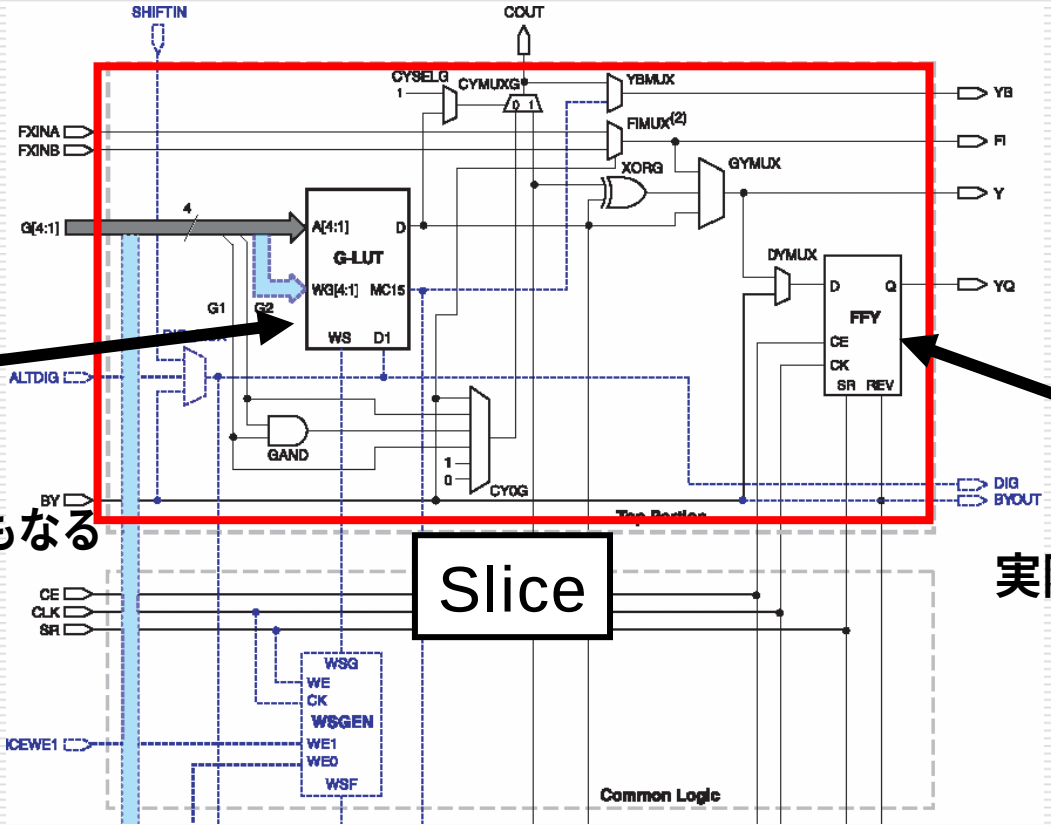
アドレス	データ	アドレス	データ
0	0	8	0
1	0	9	0
2	0	A	0
3	0	B	0
4	0	C	0
5	0	D	0
6	0	E	0
7	0	F	1

この様なデータをSRAMに書いておけばよい

Slice

- SRAMはLUT (Look Up Table)と言います
- この組み合わせ＋αをメーカー毎に名前を付けています。
  - XilinxではSliceと言います
  - Sliceが4つでCLBと言います
- データをダウンロードするとは
  - このLUTのデータを書き込むことです。

SLICE



LUT

実際はLUT以外にもなる

DFF

実際はD以外にもなる

CLB與SLICE

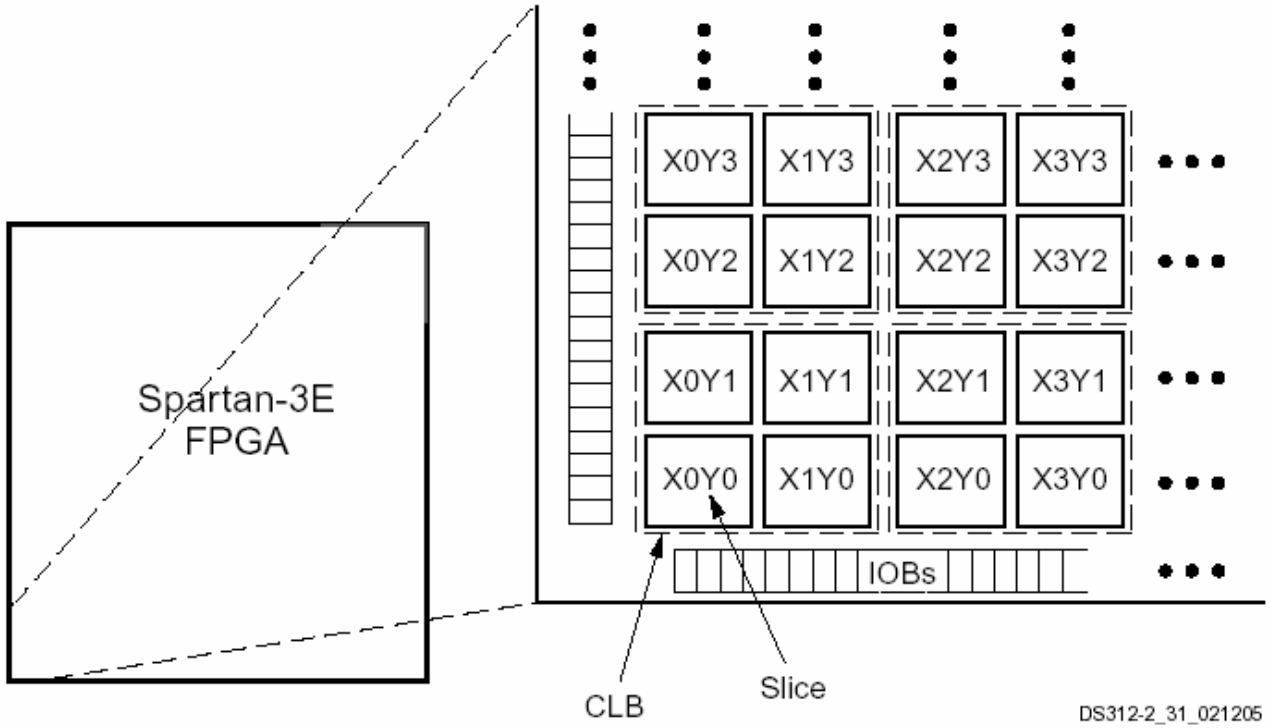
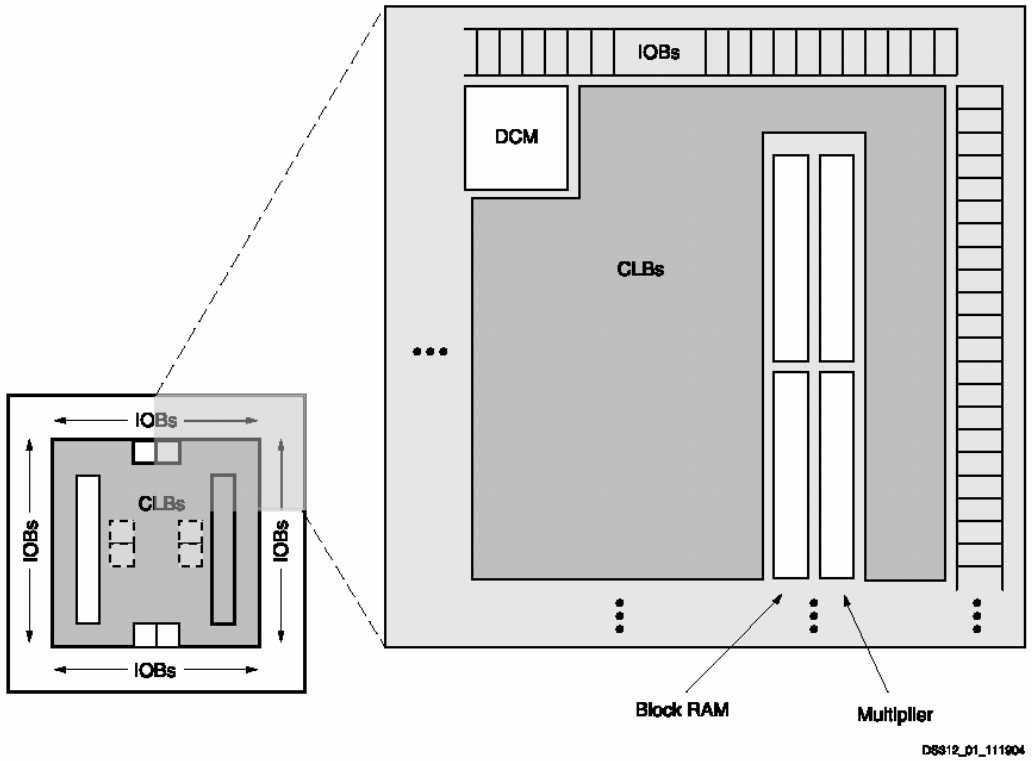


Figure 11: **CLB Locations**

FPGA全体の構成



Notes:

1. The XC3S1200E and XC3S1600E have two additional DCMs on both the left and right sides as indicated by the dashed lines. The XC3S100E has only one DCM at the top and one at the bottom.

Figure 1: **Spartan-3E Family Architecture**

FPGAの構成ブロック

- DCM
- CLB
- IOB
- Block RAM
- Multiplier

データシートを参照してください

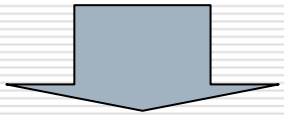
ここでは複雑になるので、これ以上説明しません。

設計時の注意

- お勧めルール
- してはいけない事
- 検出器信号の受け方（非同期入力）

なぜルール？

- 設計方法は沢山ある
- 好みもある
- HDLの書き方だけでも何種類もある



慣れるまでは設計ルールがある方が設計しやすい。

考慮する部分が少なくなるので

まずはここから(1)

- 私のお勧めは生成回路がツールに依存しにくい書き方です。
- 同期回路で設計する
- 基本4要素が分かる形で書き始めてください
- DFFにはリセット信号を入れる
- しかし、複雑な回路をすぐ設計する事はできないので
  - CORE Generatorを上手に使ってください

まずはここから (2)

- ステートマシンを使用するとき
  - 状態数をできるだけ少なく
  - 信号線数を少なく
- If分の条件に不等号を使わない
  - 使うときは一度DFF出力の信号を作る＋条件式
- 徐々にHDLの機能を使うようにしてください
- 対照的にテストベンチ用コードはHDLの機能を十分に使って書いてください
  - 回路にしないので

してはいけない事

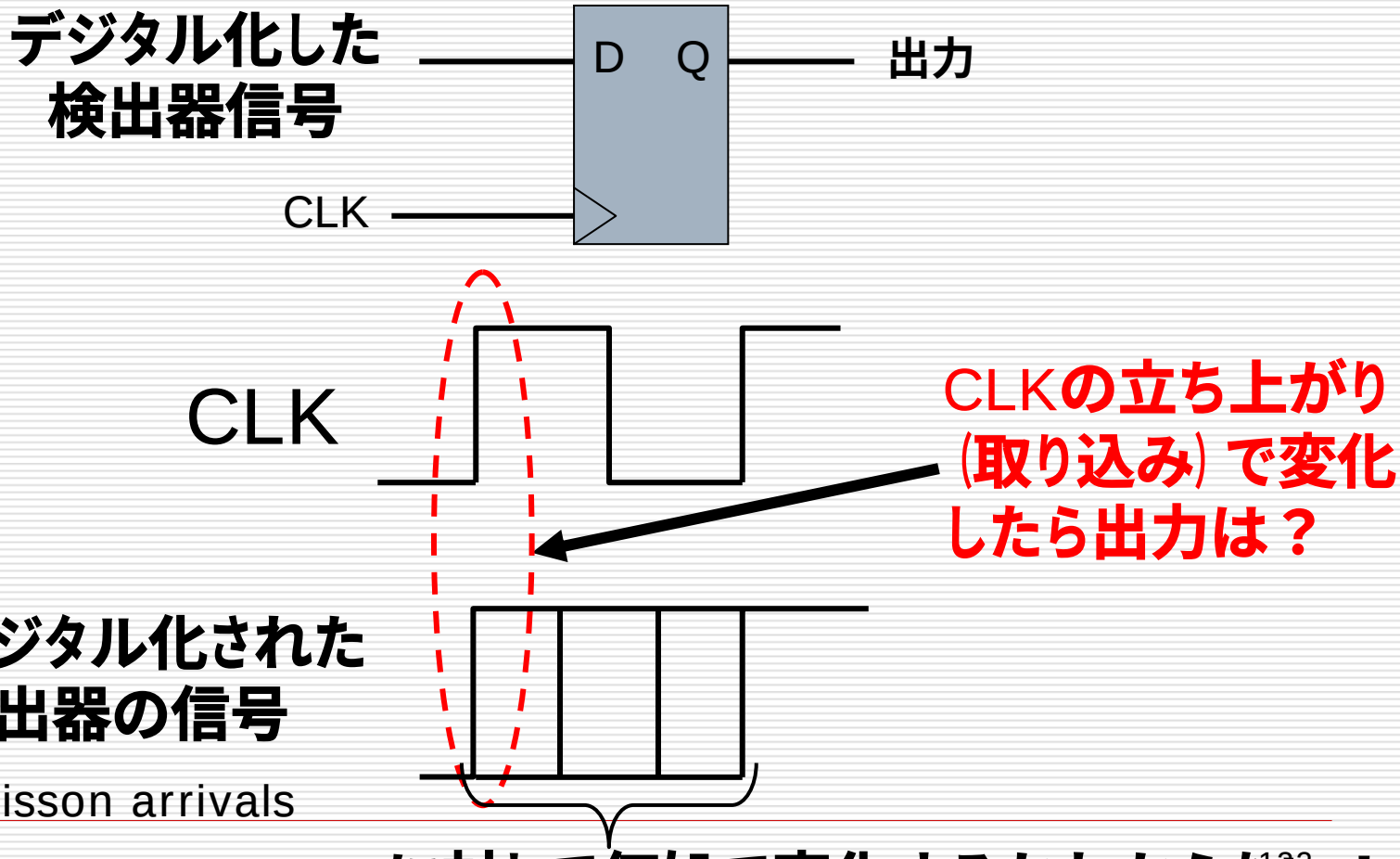
- DCMを使用しない
  - 使って悪い事が起こるときはFPGAの限界です
  - ただし5MHz以上のクロック入力が必要です
- 内部で3-state（双方向を含む）を使用する
  - 内部の3-state記述は合成ツールにより勝手に分解される
  - その回路はツールが知るのみ。
- 回路が分からないからHDL記述に頼る
  - その回路はツールの気分しだい
  - 動くか動かないかはツールと記述の相性しだい
  - HDLを理解すると言うよりツールを理解するに近い

検出器信号の受け方
非同期入力

検出器の信号

- 検出器の信号はアナログ信号
- FPGAに入力される時
  - 通常ディスクリ（デジタル化）出力を入力する
- 検出器の出力はランダム
  - Poisson arrival
- FPGAは同期回路
  - 非同期信号を入力しても問題は無いのか？

入力回路

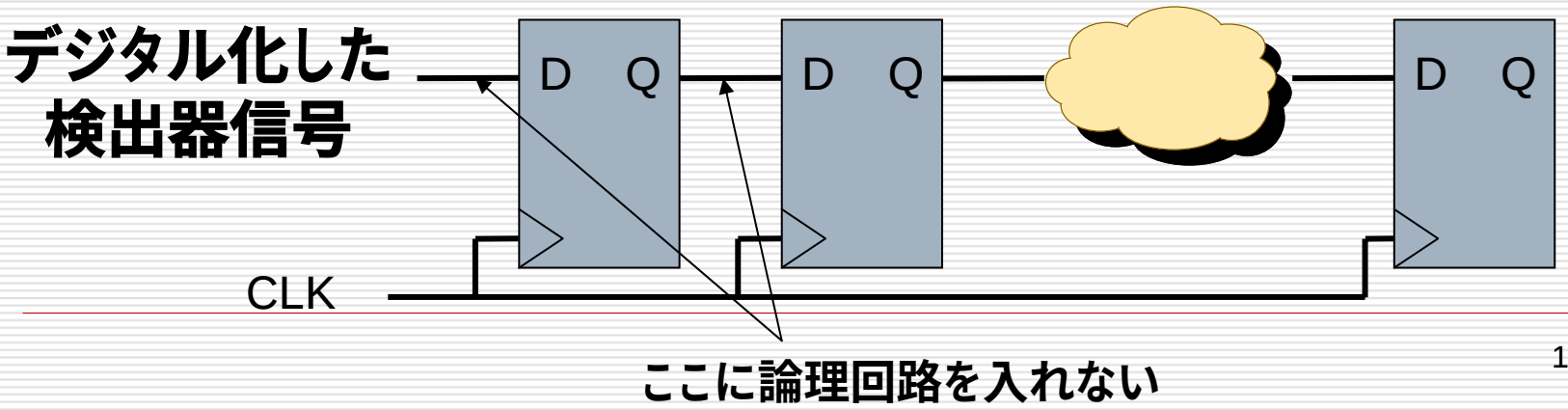

デジタル化した
検出器信号
D
Q
出力
CLK
CLK
CLKの立ち上がり
(取り込み) で変化
したら出力は？
デジタル化された
検出器の信号
Poisson arrivals
CLKに対して何処で変化するかわからない！

DFFのMeta-stable state

- CLKの立ち上がりで入力が変化した時
  - 出力がどうなるかはトランジスタレベルの振る舞いに依存します
- 例の場合
  - Lowのままかもしれませんし
    - サンプリング誤差になる
  - Highになるかもしれません
    - この時は問題ない
  - もしかしたら中間レベルになるかもしれません、時には振動する事もあります
    - これは誤動作する原因です！

対策

- 実は決定的な対策はありません！
- 誤動作発生確率を無視できるくらいまで下げることで対処します
  - 中間状態になる確率は低い事
  - 中間状態を取る条件がトランジスタレベルの個性で異なる事を利用する
- 通常、2段以上のDFFを直列に接続します



参考書

- Veritakのチュートリアル
  - 分かりやすく書いてあります
- Verilog-HDL
  - 入門Verilog HDL記述、小林優、CQ出版社
  - ISEソフトウエア・マニュアル
- FPGA
  - とにかくデータシート
  - アプリケーションノートを読むに限る

最後に

- これで参考書を見ながら頑張れるはずです
- 私のやり方を中心に話しました。これは必ずしも正しい方法ではありません。
- 設計方法は十人十色です。自分の設計方法を確立してください。
- 質問、疑問などあったらメールください

楽しく作りましょう！！

付録

付録

- MCSファイルの生成
- デバイス依存ライブラリの使用方法
- タイミング制約
- XILINXライブラリの使い方

MCSファイルの作り方

Configure device

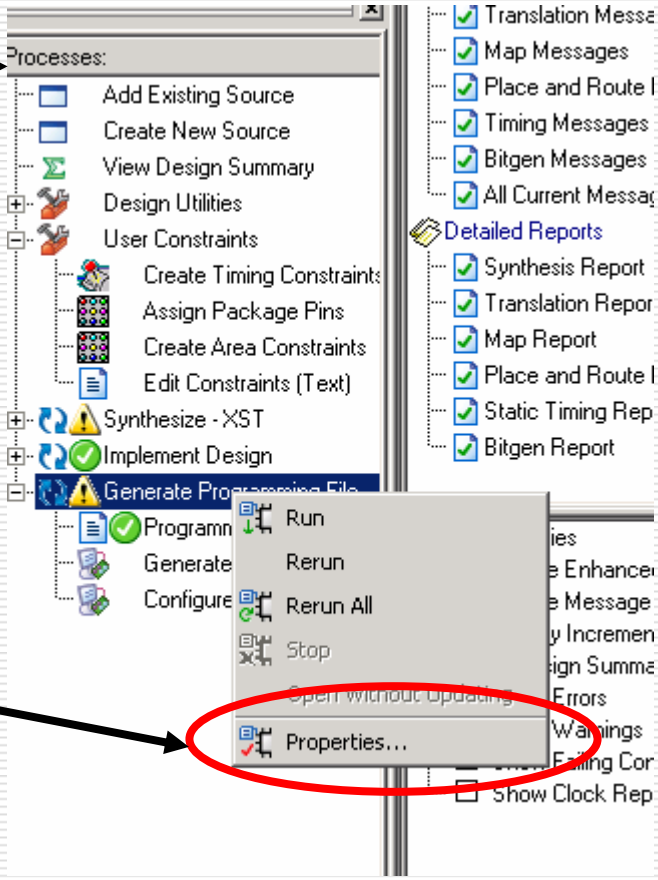
- ボードのダウンロードモードをMaster Serialにする
  - 3-starter kit: J8全て接続
  - 3E-starter kit: J30全て接続

Start Up-Clockの設定

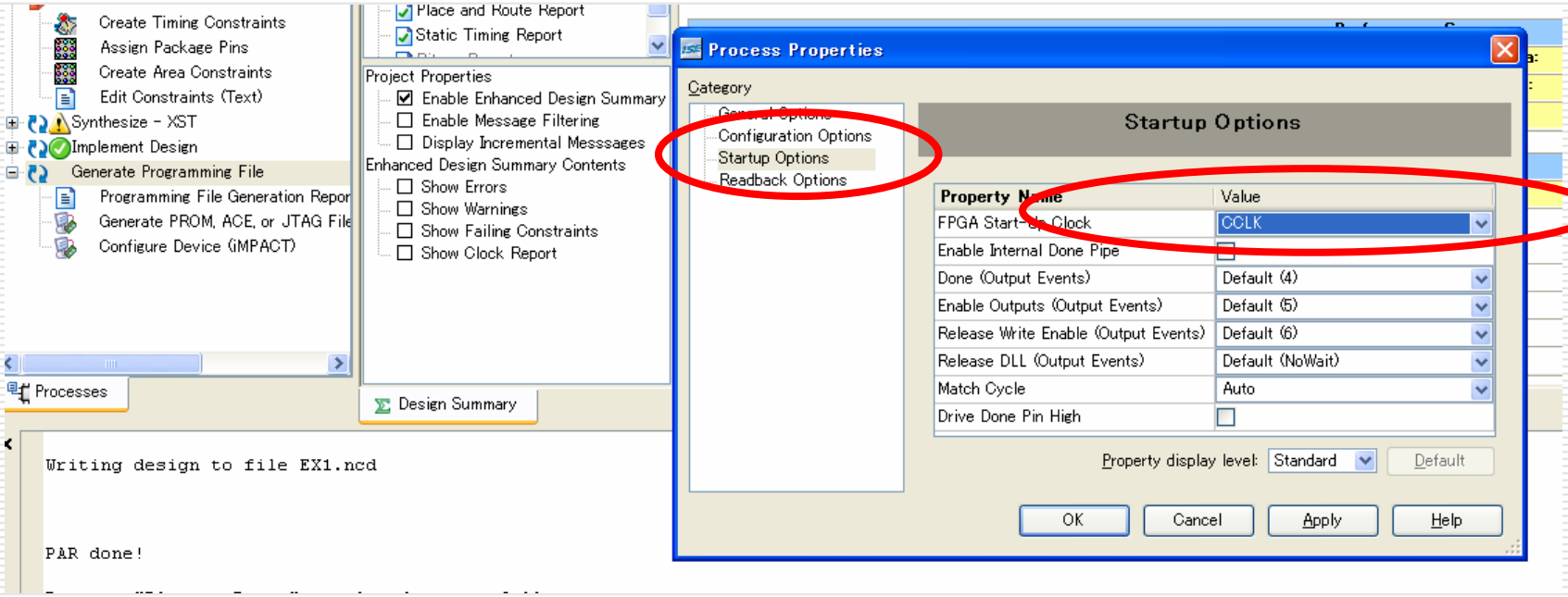
Process View

Generate Programming File

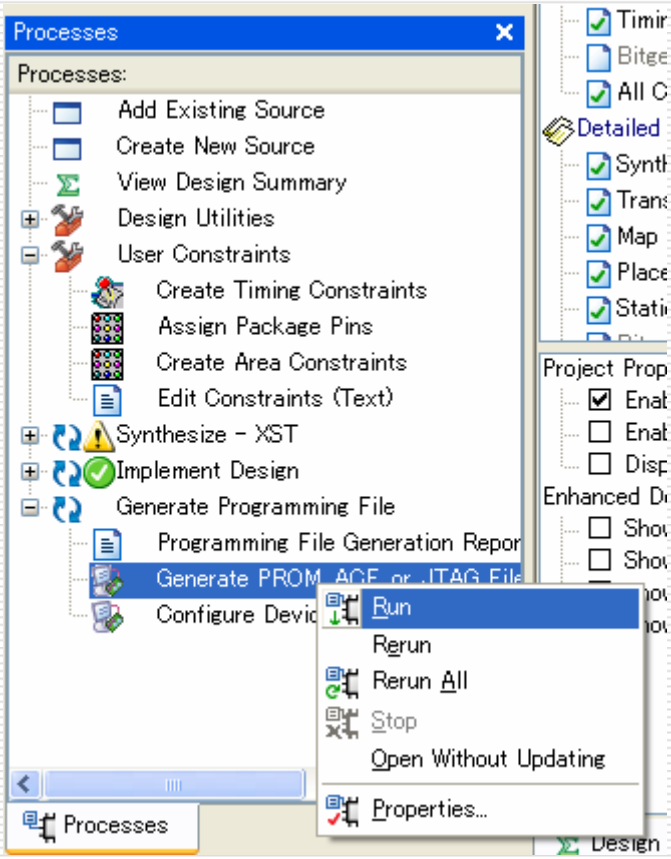
左クリック→Properties



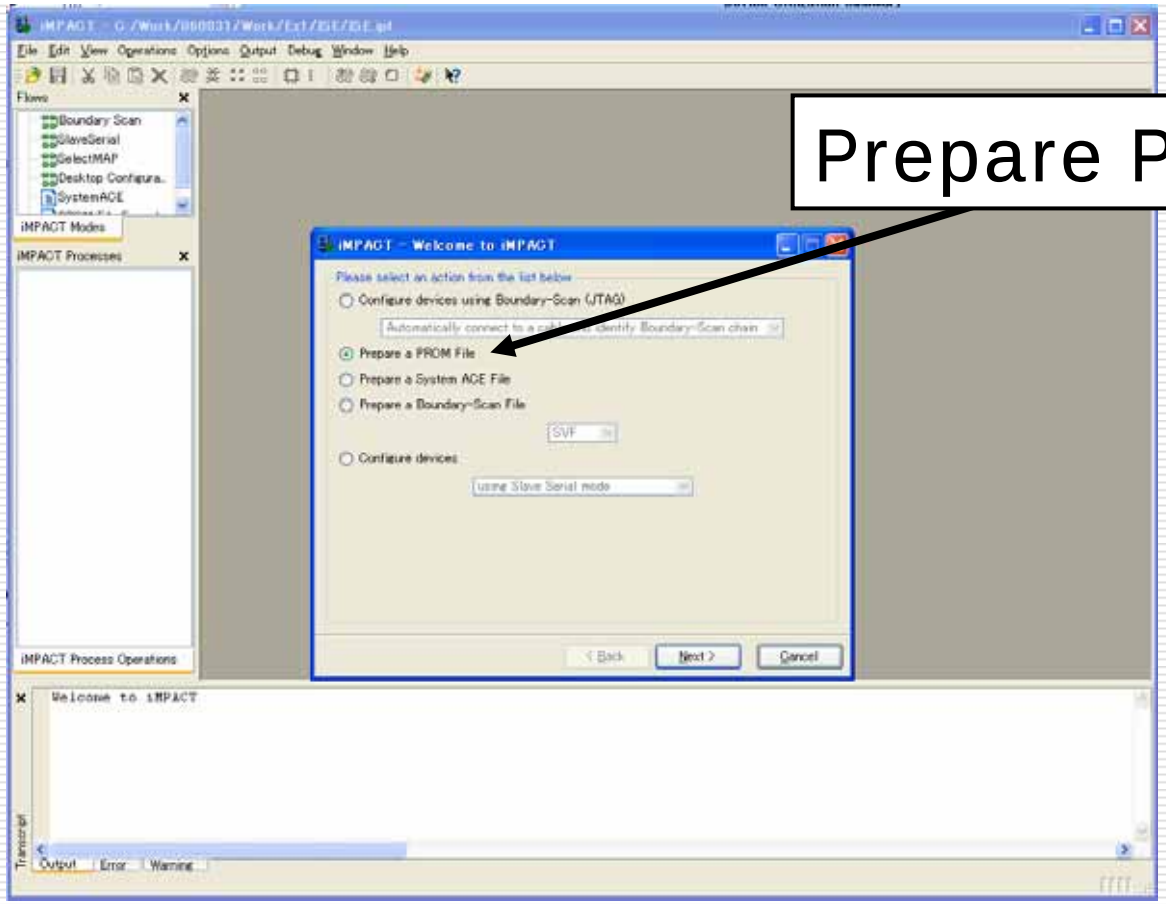
Start Up-Clockの設定



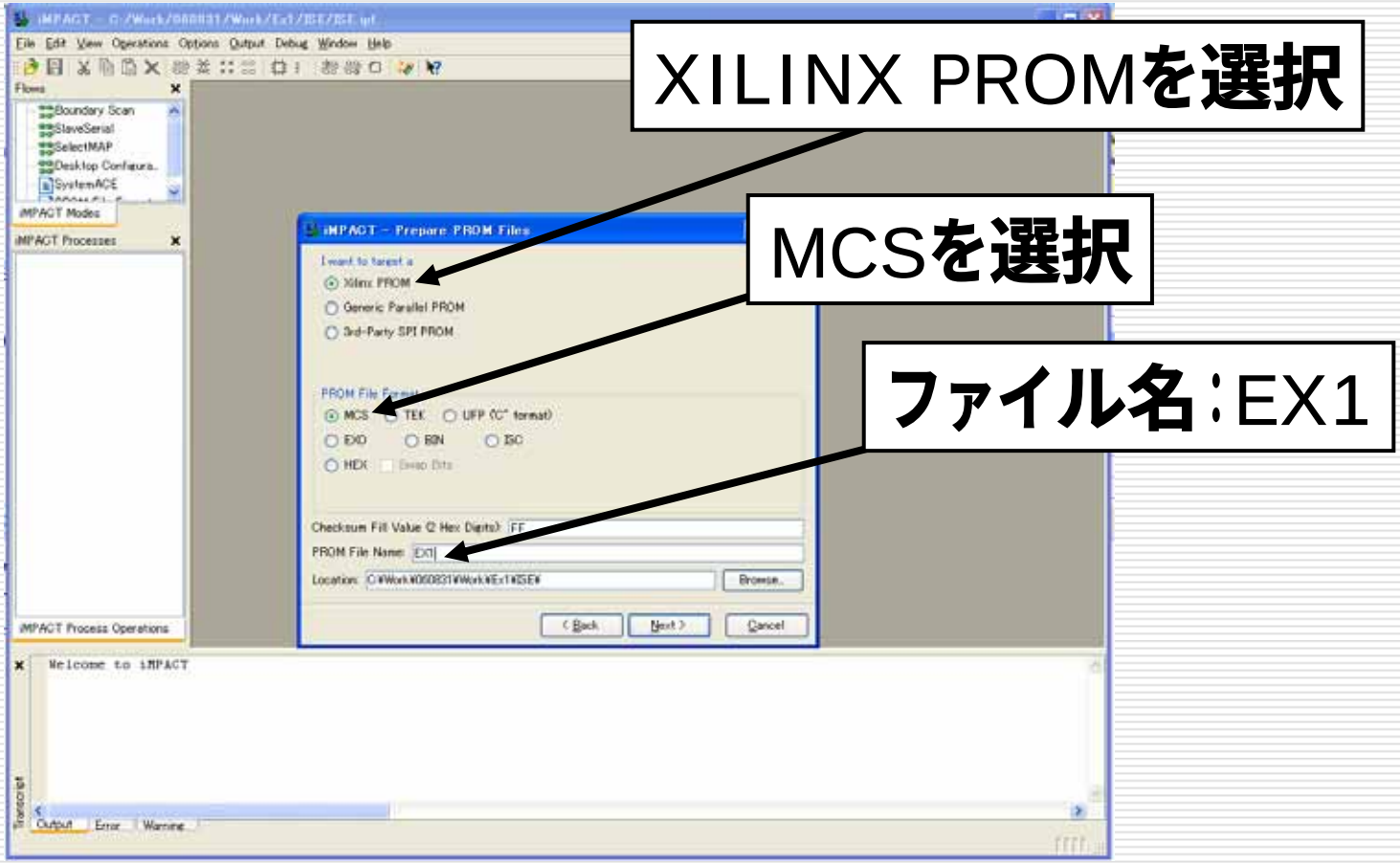
ROM file生成


Processes
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Create Timing Constraints
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
Implement Design
Generate Programming File
Programming File Generation Repor
Generate PROM, ACE, or JTAG File
Configure Devic
Run
Rerun
Rerun All
Stop
Open Without Updating
Properties...
Processes
Detailed
Project Prop
Enhanced D
Design

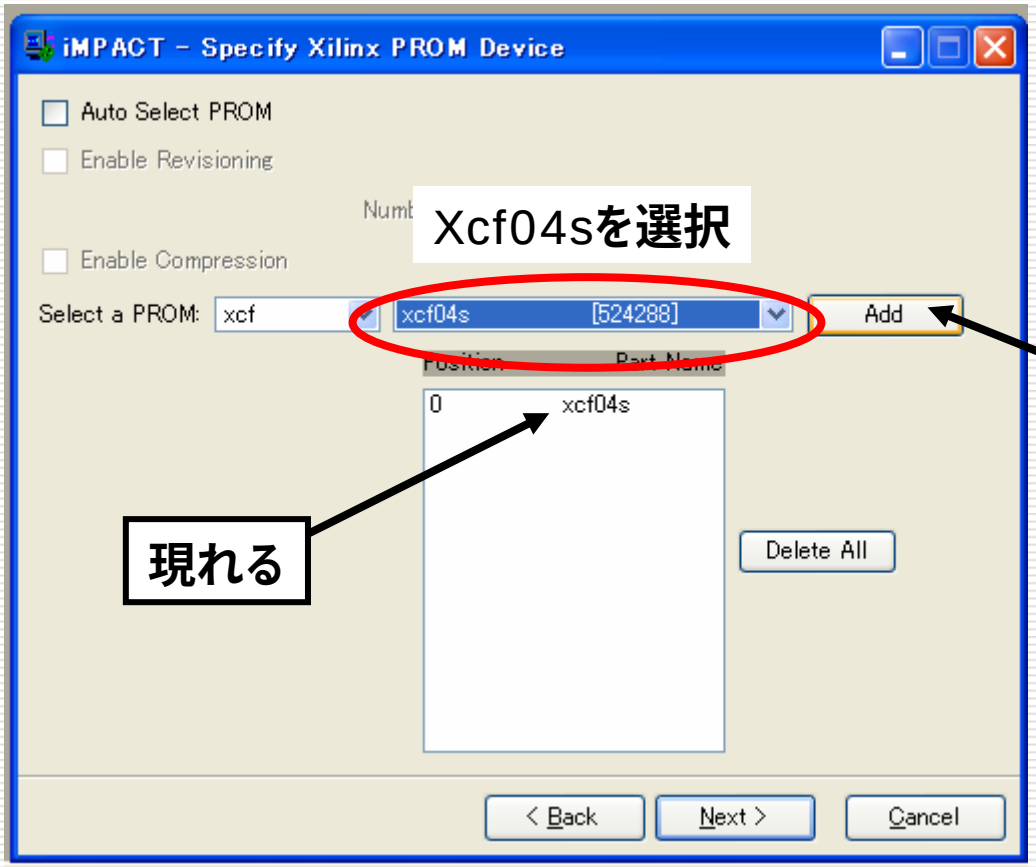
PROMファイルの設定



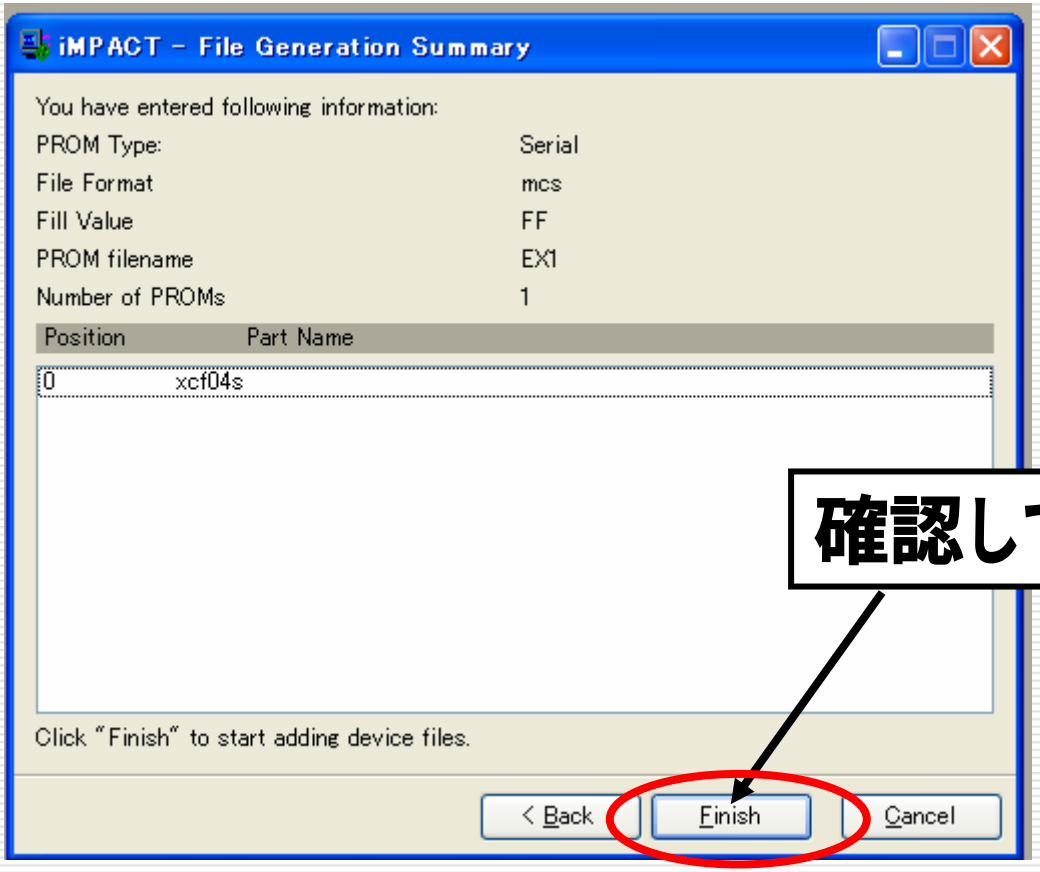
PROMファイルの設定



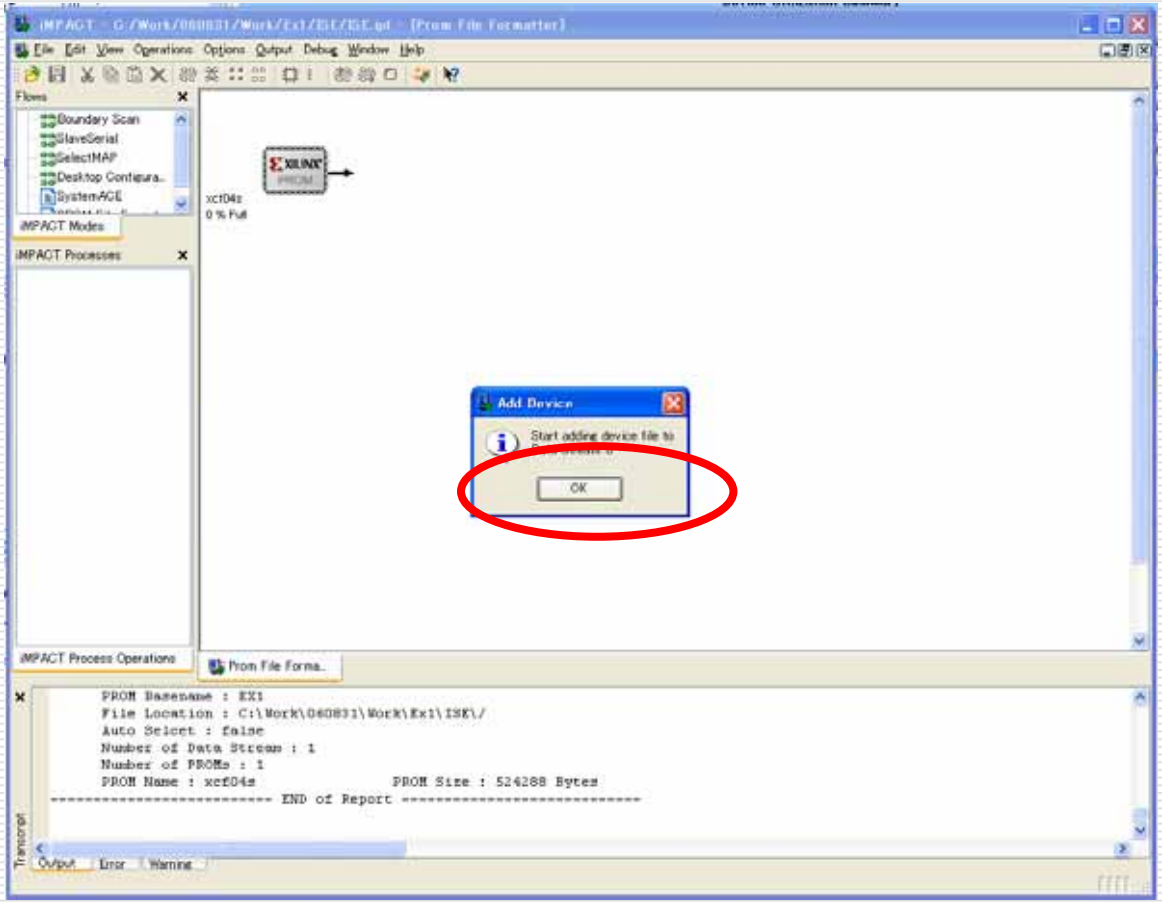
PROMファイルの設定



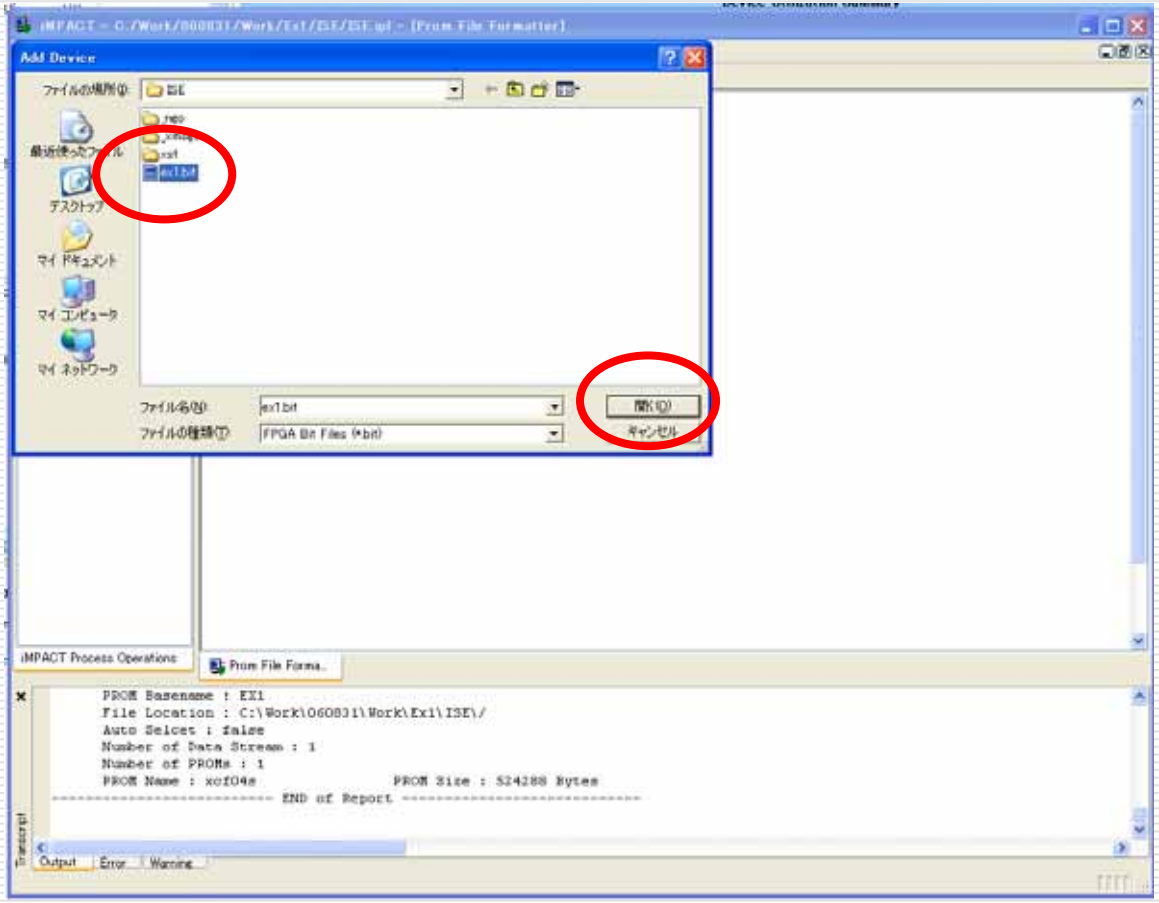
PROMファイルの設定



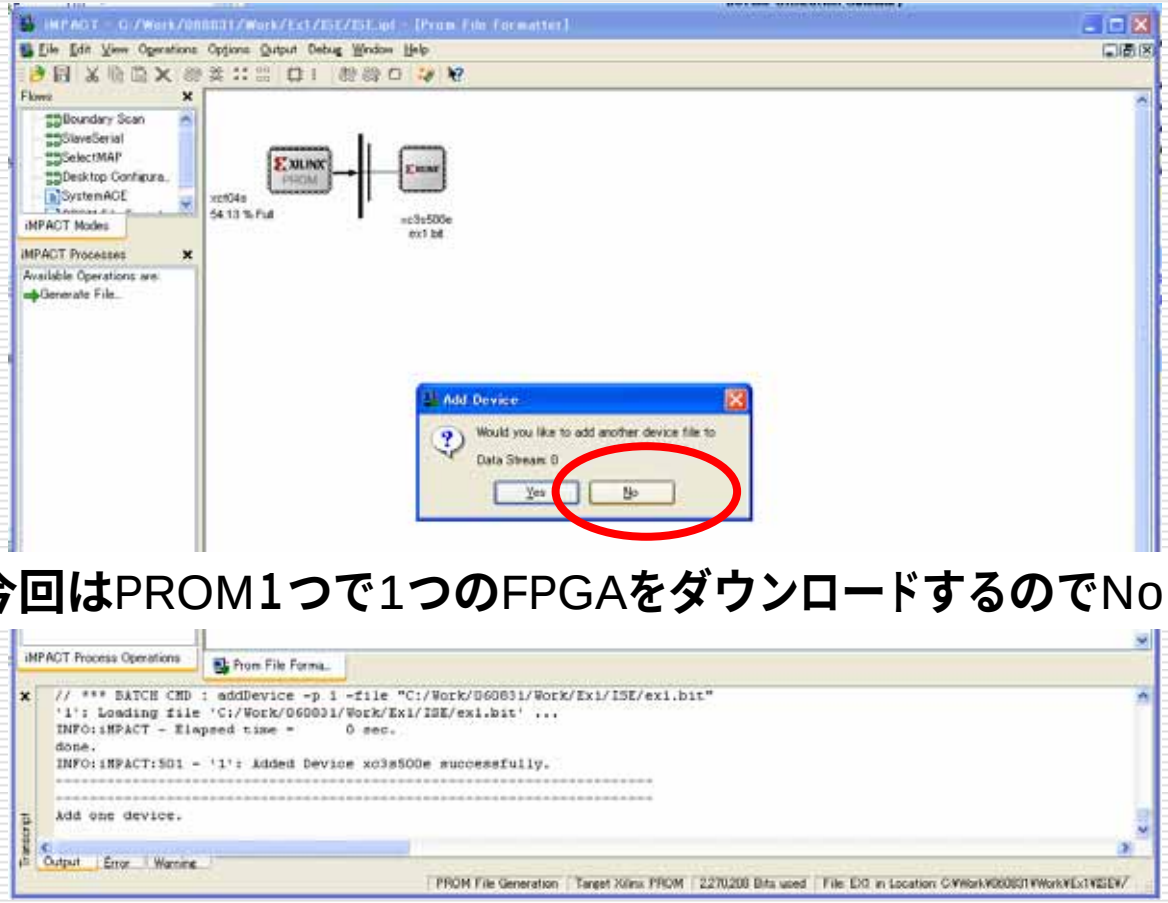
BITファイルの指定


File Edit View Operations Options Output Debug Window Help
Flows
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configura..
SystemACE
iMPACT Modes
iMPACT Processes
iMPACT Process Operations
xcf04s
0 % Full
Add Device
Start adding device file to
OK
Prom File Forma..
PROM Basename : EX1
File Location : C:\Work\060831\Work\Ex1\ISE\/
Auto Select : false
Number of Data Stream : 1
Number of PROMs : 1
PROM Name : xcf04s PROM Size : 524288 Bytes
---------- END of Report ----------
Transcript
Output Error Warning

BITファイルの選択


Add Device
ex1.bit
FPGA Bit Files (*.bit)
iMPACT Process Operations
PROM Basename : ex1
File Location : C:\Work\060831\Work\ex1\ISE\
Auto Select : false
Number of Data Stream : 1
Number of PROMs : 1
PROM Name : xcf04s
PROM Size : 524288 Bytes
END of Report
Output
Error
Warning

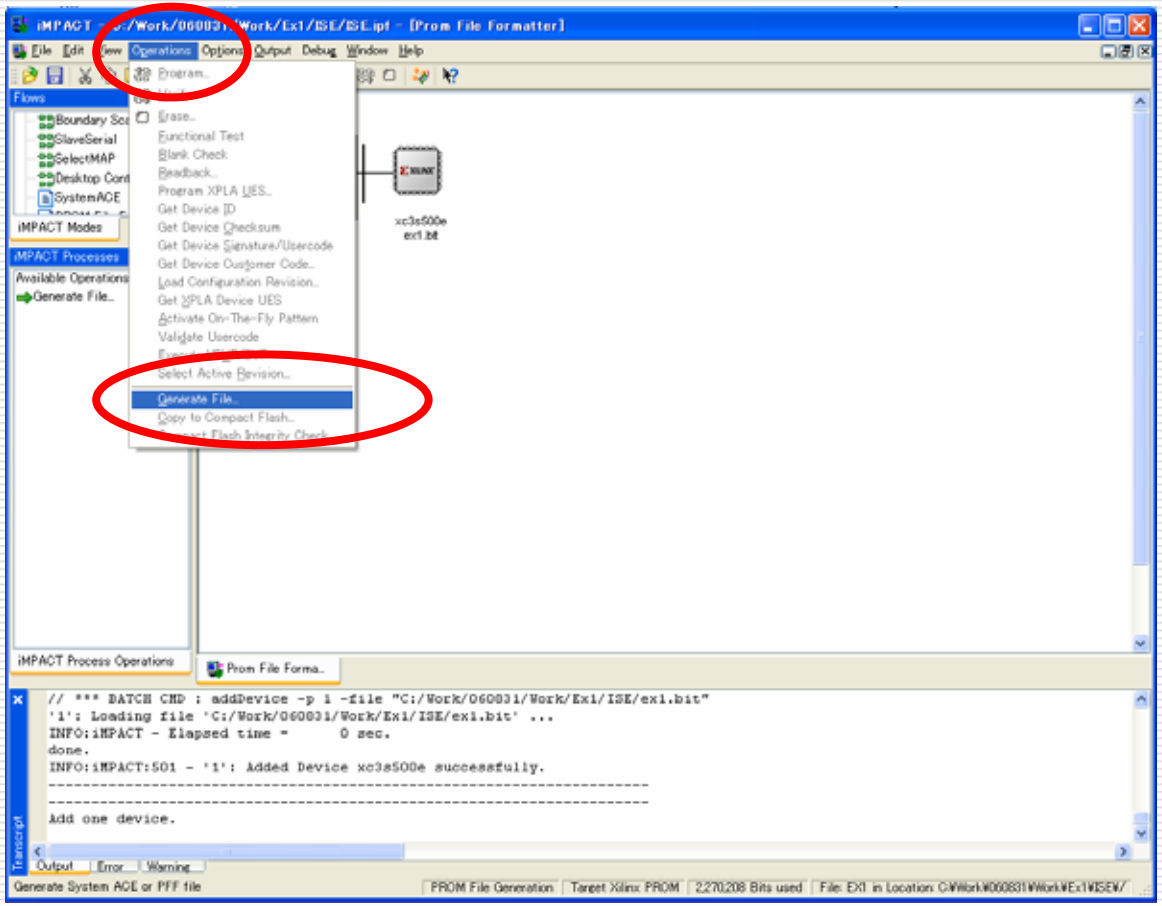
BITファイルの選択



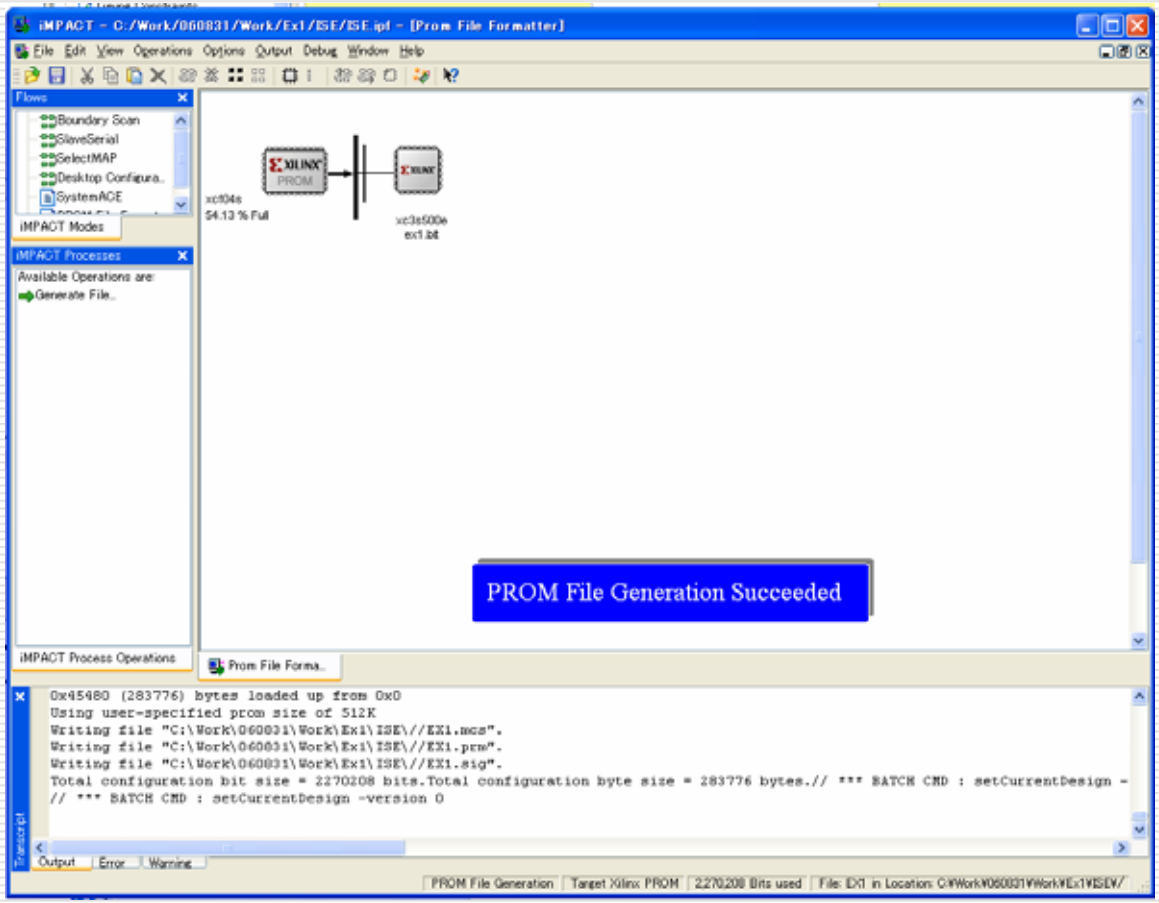
今回はPROM1つで1つのFPGAをダウンロードするのでNo

PROM1つで2以上のFPGAをダウンロードする時は接続されている順番にファイルをアサインする

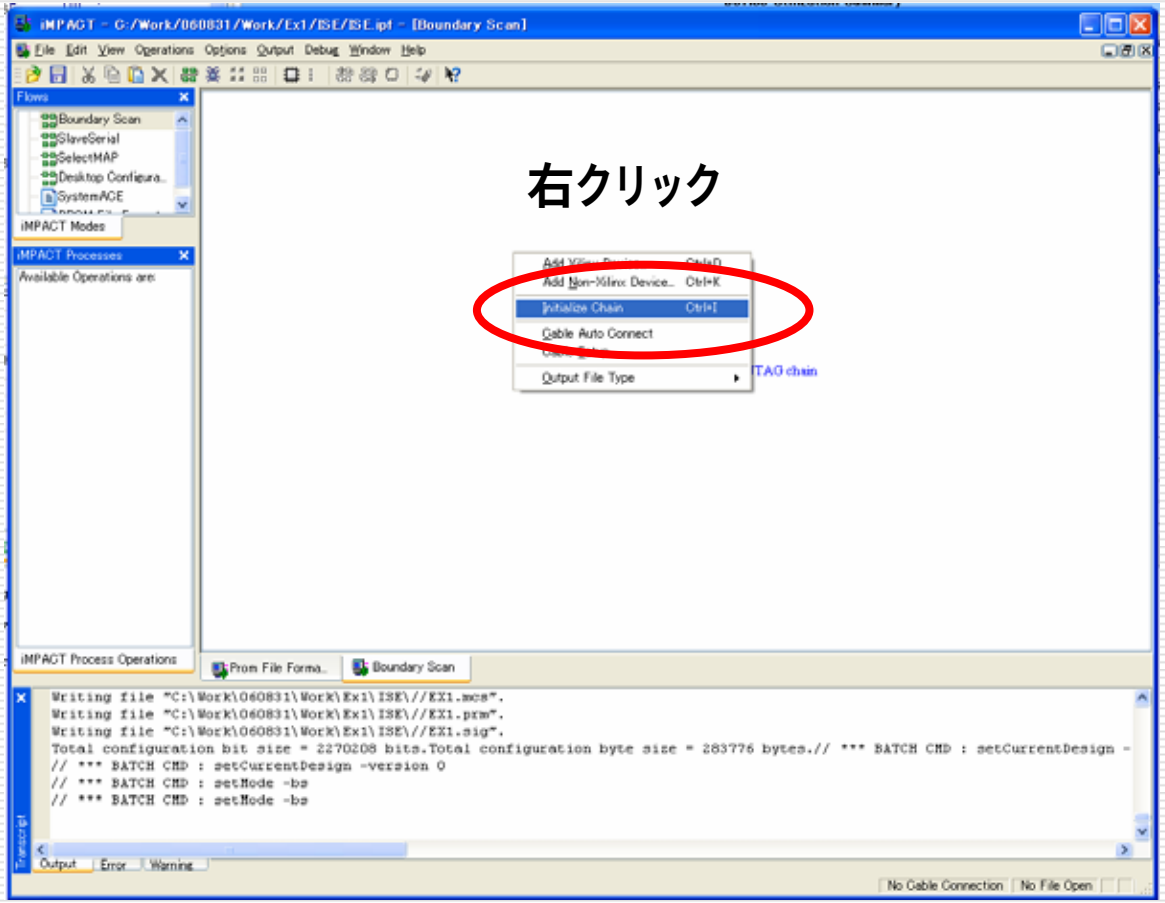
PROMファイルの生成



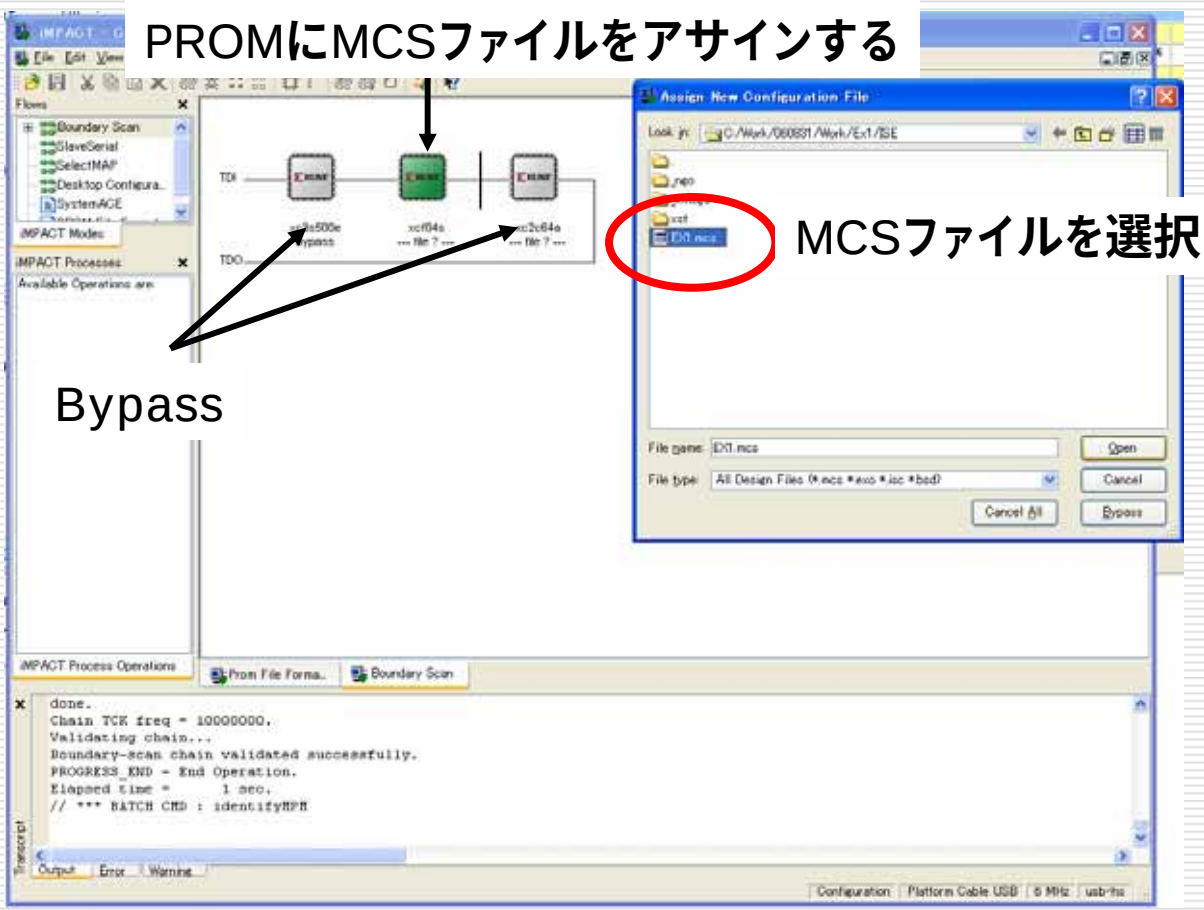
PROMデータ生成終了



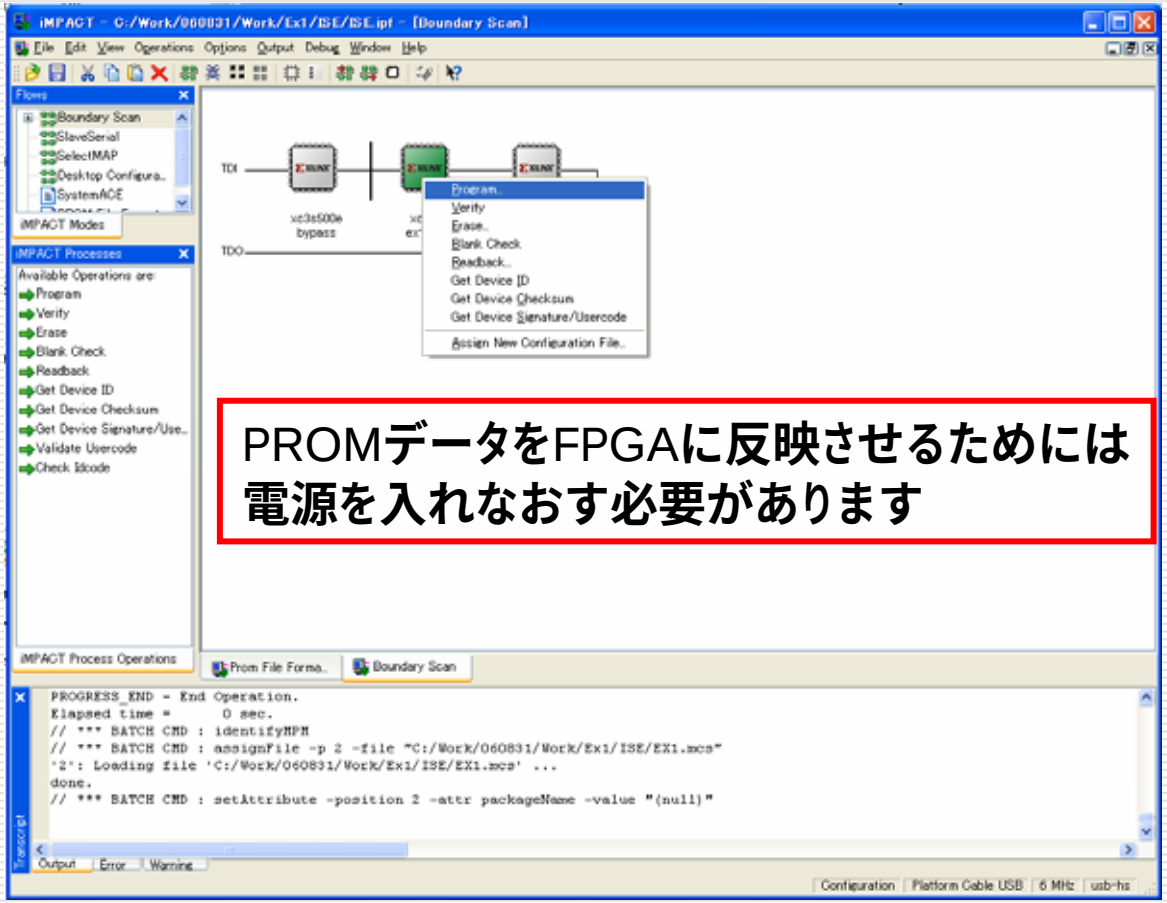
ファイルのダウンロード



MCSファイルのダウンロード



ダウンロード



PROMデータをFPGAに反映させるためには
電源を入れなおす必要があります

デバイス依存ライブラリ

デバイス依存のライブラリとは

- FPGA内には専用回路があります
  - Memory (Block RAM)
  - Clock manager (DCM)
  - DSP
- ライブラリとして必要な物用意されています
  - ブラックボックスとして扱えるように
  - シミュレーション･モデル
  - 配置情報など
- ここではDCMを例に使い方を説明します。

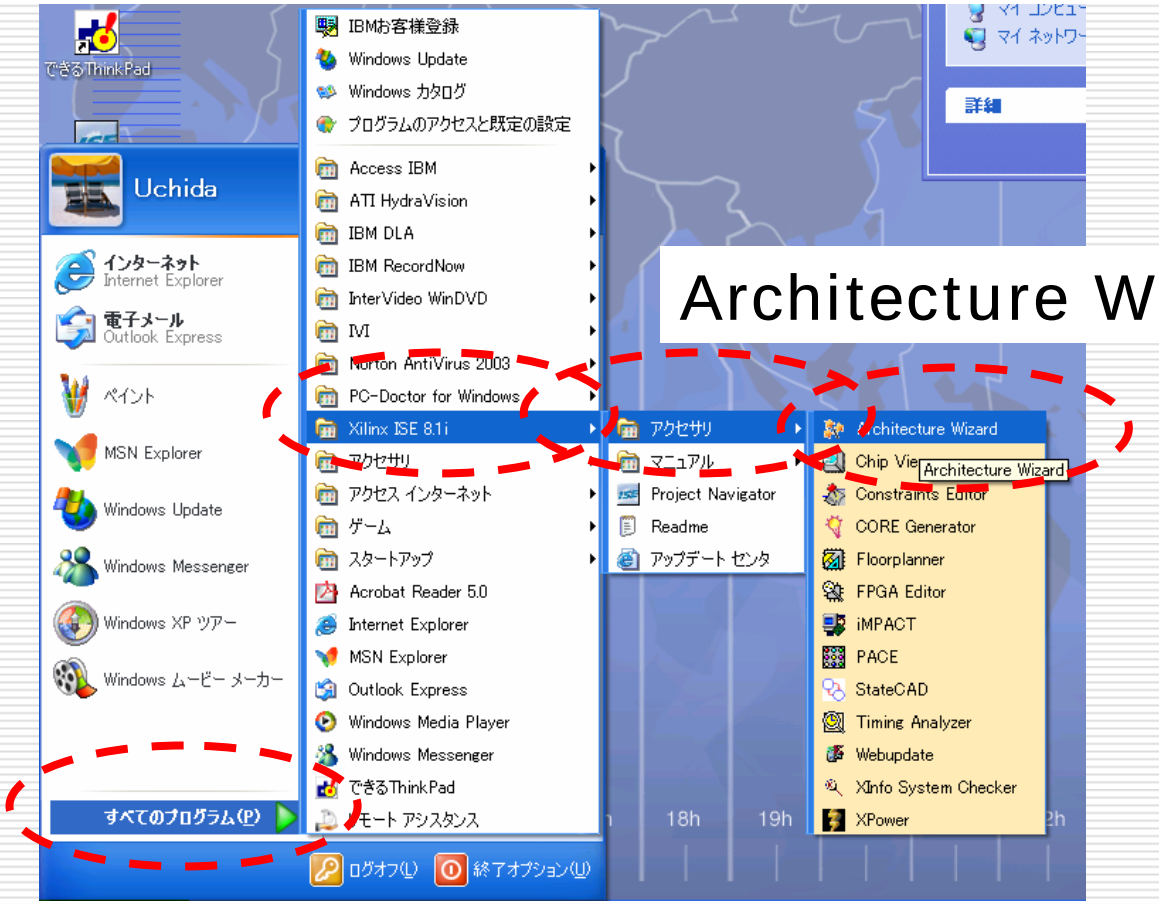
DCMとは

- クロック管理回路
  - 同期回路ではクロックは非常に大切な信号
    - 全DFFに同じクロックを分配する必要
    - 遅延時間のばらつきを最小に抑える（補償してくれる）
- 特徴
  - 入力クロックの1/2,2倍のクロックなど生成できる
    - 反転クロックなどもできる
  - 全ての位相は合っている
- 通常はDCMを使うようにしてください

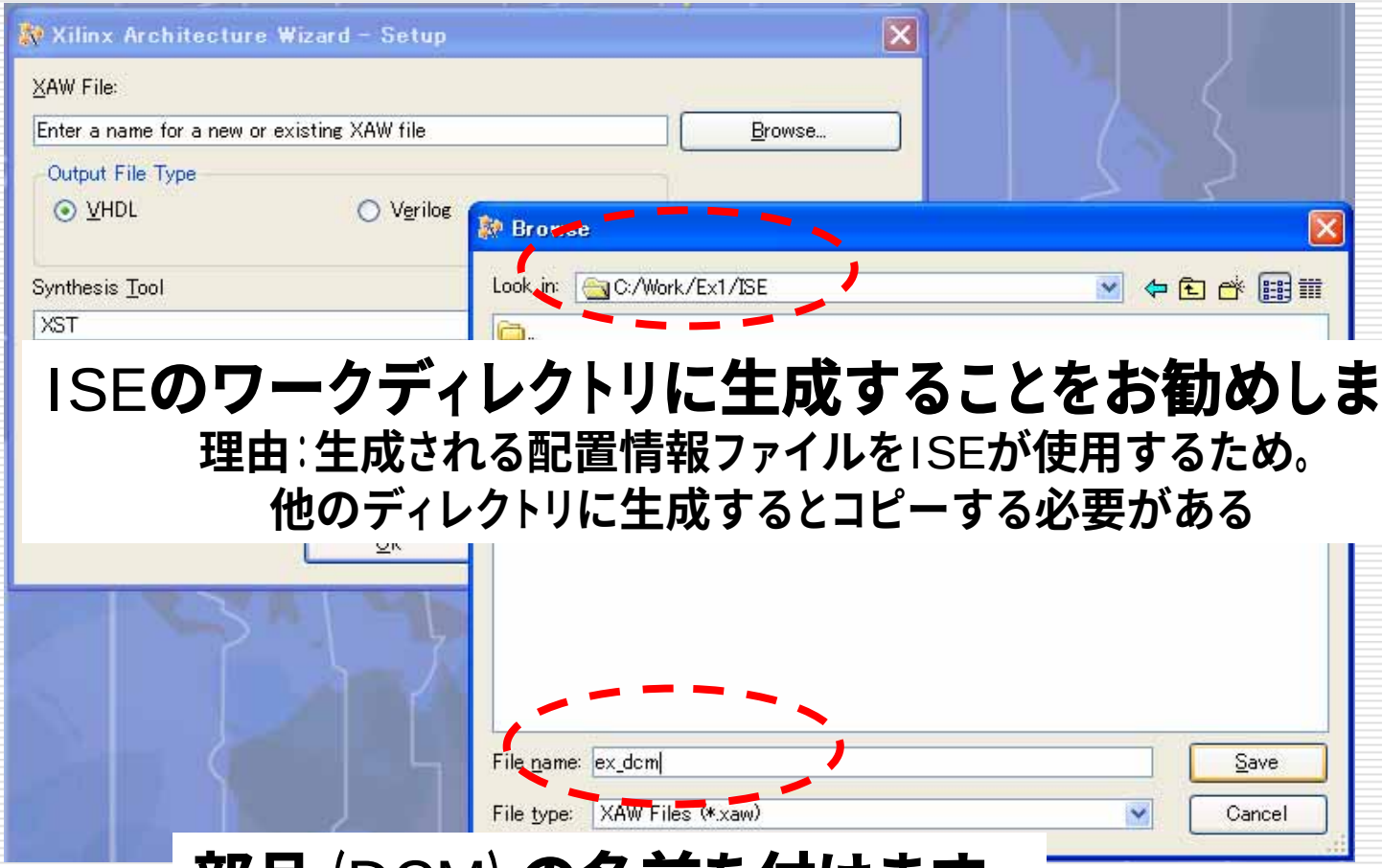
DCMの使い方

- DCMはArchitecture Wizardで生成します
- カウンター回路にDCMを入れましょう

Architecture Wizard



環境設定



ISEのワークディレクトリに生成することをお勧めします。

理由：生成される配置情報ファイルをISEが使用するため。

他のディレクトリに生成するとコピーする必要がある

部品 (DCM) の名前を付けます。

ここではEx_dcm

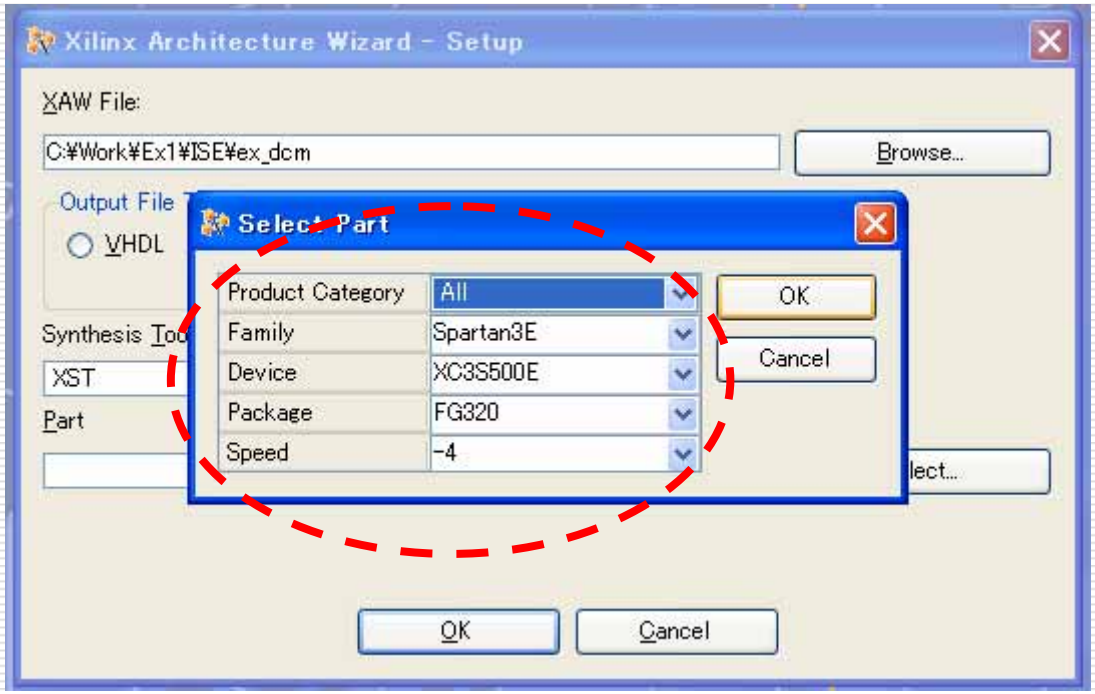
Output File Type

Verilogを選択

ISEの合成ツールを使用するのでXST

次にデバイスの選択

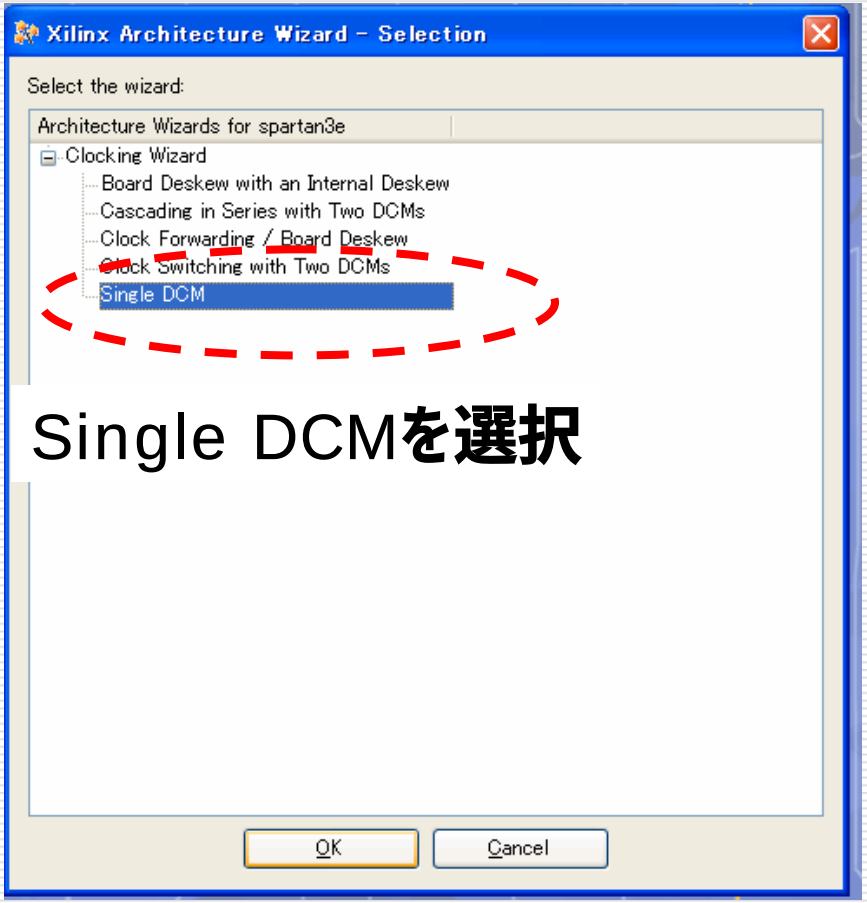
デバイスの選択


Xilinx Architecture Wizard - Setup
XAW File:
C:¥Work¥Ex1¥ISE¥ex_dcm
Browse...
Output File
VHDL
Synthesis Too
XST
Part
Select Part
Product Category
All
Family
Spartan3E
Device
XC3S500E
Package
FG320
Speed
-4
OK
Cancel
lect...
OK
Cancel

セットアップ終了


Xilinx Architecture Wizard - Setup
XAW File:
C:¥Work¥Ex1¥ISE¥ex_dcm
Browse...
Output File Type
VHDL
Verilog
Synthesis Tool
XST
Part
xc3s500e-fg320-4
Select...
OK
Cancel

DCMの種類を選択



Single DCMを選択

DCMの設定

使用するピンにチェックを入れる
CLKDVにチェックを入れる
(分周クロック使用)

入力クロックの周波数

位相制御

入力クロックの選択
FPGA**外か内か**
通常はExternal

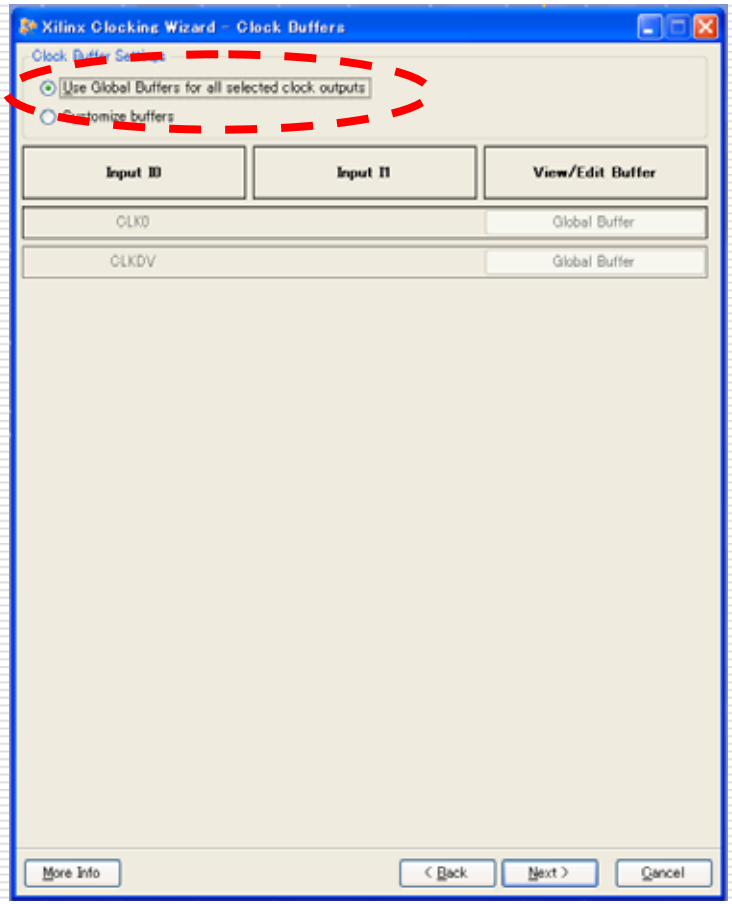
FeedbackがFPGA**内部か外部か**
通常はInternal

Feedback**に使用するクロックの選択**
通常はx1**を選択**

分周比
ここでは1/2**のクロックが**CLKDV**から出力**

DCM**の詳細はデータシートを参照**

Global bufferの選択



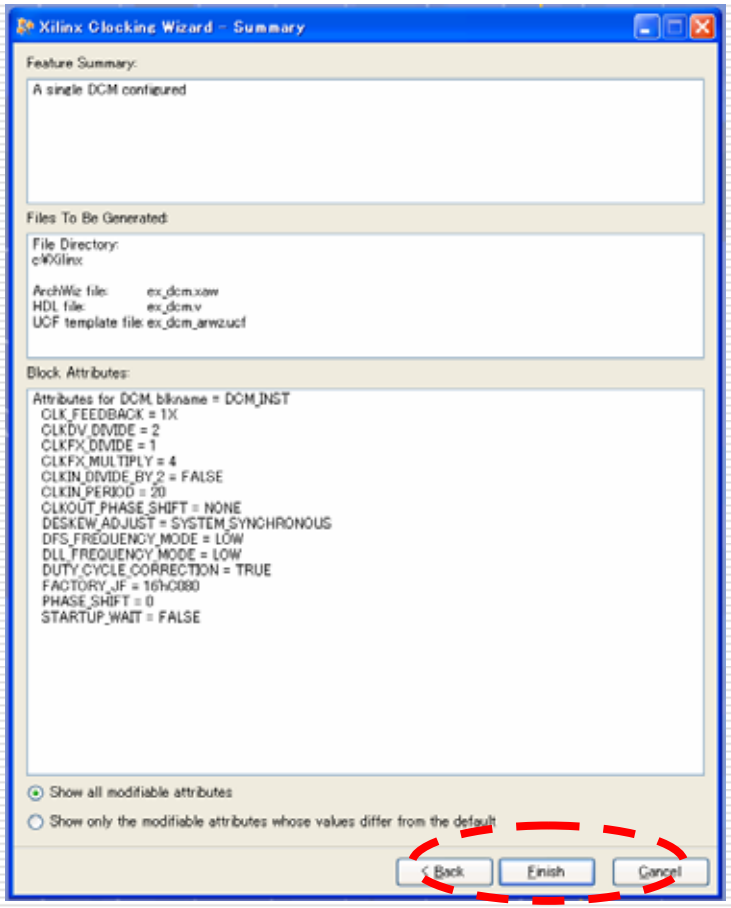
通常はGlobal bufferを使用

クロックは低遅延差で全てのDFFにクロックを供給する必要がある

FPGAではグローバルラインと言う特殊な専用信号線がある。

その専用信号線をドライブするためのバッファがGlobal bufferです

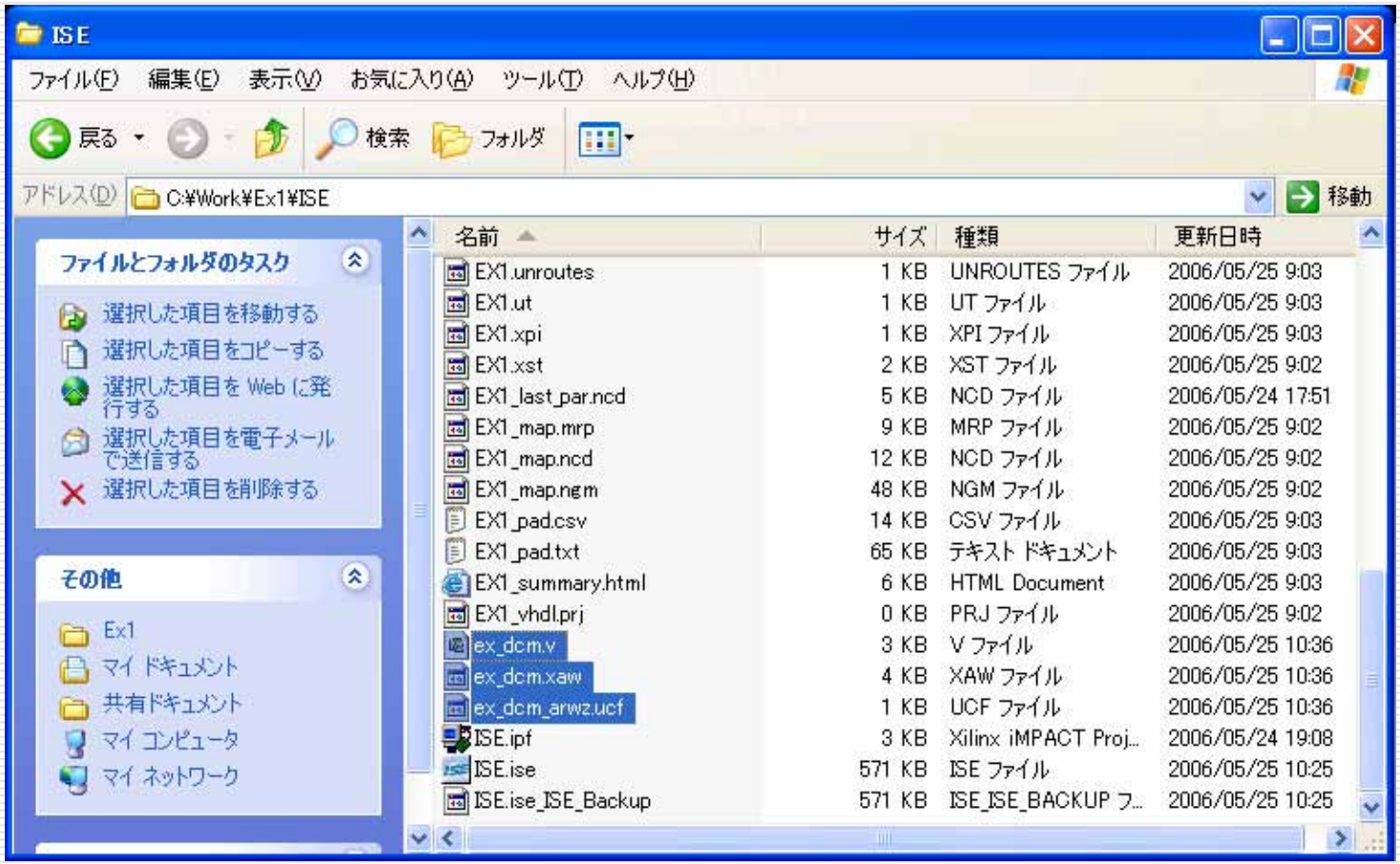
確認画面



DCMが生成されたはずです

Work/Ex1/ISEを見てください
Ex_dcm*というファイル
何種類かあるはずです。

生成されたファイル


ISE
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
戻る 検索 フォルダ
アドレス(D) C:¥Work¥Ex1¥ISE
移動
ファイルとフォルダのタスク
選択した項目を移動する
選択した項目をコピーする
選択した項目を Web に発行する
選択した項目を電子メールで送信する
選択した項目を削除する
その他
Ex1
マイ ドキュメント
共有ドキュメント
マイ コンピュータ
マイ ネットワーク
名前 サイズ 種類 更新日時
EX1.unroutes 1 KB UNROUTES ファイル 2006/05/25 9:03
EX1.ut 1 KB UT ファイル 2006/05/25 9:03
EX1.xpi 1 KB XPI ファイル 2006/05/25 9:03
EX1.xst 2 KB XST ファイル 2006/05/25 9:02
EX1_last_par.ncd 5 KB NCD ファイル 2006/05/24 17:51
EX1_map.mrp 9 KB MRP ファイル 2006/05/25 9:02
EX1_map.ncd 12 KB NCD ファイル 2006/05/25 9:02
EX1_map.ngm 48 KB NGM ファイル 2006/05/25 9:02
EX1_pad.csv 14 KB CSV ファイル 2006/05/25 9:03
EX1_pad.txt 65 KB テキスト ドキュメント 2006/05/25 9:03
EX1_summary.html 6 KB HTML Document 2006/05/25 9:03
EX1_vhdl.prj 0 KB PRJ ファイル 2006/05/25 9:02
ex_dcm.v 3 KB V ファイル 2006/05/25 10:36
ex_dcm.xaw 4 KB XAW ファイル 2006/05/25 10:36
ex_dcm_arwz.ucf 1 KB UCF ファイル 2006/05/25 10:36
ISE.ipf 3 KB Xilinx iMPACT Proj... 2006/05/24 19:08
ISE.ise 571 KB ISE ファイル 2006/05/25 10:25
ISE.ise_ISE_Backup 571 KB ISE_ISE_BACKUP フ... 2006/05/25 10:25

DCM回路ファイル
ex_dcm.v

- ex_dcmは回路なのでvファイルが生成されます。
- Work/Ex1/ISEにex_dcm.vが生成されているはずです。確認してください
- ex_dcm.vを開いてください

ex_dcm.v

```
P1 //////////////////////////////////////////////////////////////////////////////↓
 2 // Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.↓
 3 //////////////////////////////////////////////////////////////////////////////↓
 4 //   ____  ____ ↓
 5 //  /   /\/   / ↓
 6 // /___/  \  /    Vendor: Xilinx ↓
 7 // \   \   \/     Version : 8.1i↓
 8 //  \   \         Application : xaw2verilog↓
 9 //  /   /         Filename : ex_dcm.v↓
10 // /___/   /\     Timestamp : 05/25/2006 10:35:58↓
11 // \   \  /  \  / ↓
12 //  \___\/\___\/ ↓
13 //↓
14 //Command: xaw2verilog -st C:\Work\Ex1\ISE\ex_dcm.xaw C:\Work\Ex1\ISE\ex_dcm↓
15 //Design Name: ex_dcm↓
16 //Device: xc3s500e-fg320-4↓
17 //↓
```

これをEx1.vに組み入れましょう！

moduleが定義されている

```
21 `timescale 1ns / 1ps↓
22 ↓
23 module ex_dcm(CLKIN_IN, ↓
24                RST_IN, ↓
25                CLKDV_OUT, ↓
26                CLKIN_IBUFG_OUT, ↓
27                CLK0_OUT, ↓
28                LOCKED_OUT);↓
29 ↓
30     input CLKIN_IN;↓
31     input RST_IN;↓
32    output CLKDV_OUT;↓
33    output CLKIN_IBUFG_OUT;↓
34    output CLK0_OUT;↓
35    output LOCKED_OUT;↓
36    ↓
37    wire CLKDV_BUF;↓
```

- RST_IN:**リセット信号入力**
- CLKIN_IN:**クロック入力**
- CLKDV_OUT:**分周クロック出力**
- CLK0_OUT:**x1クロック出力**
- LOCK:**ロック出力**

DCMの組み込み

- この回路をEX1.vに組み込みましょう
  - この後、具体的に見ていきます。
- RST_IN:リセット信号入力
  - BTN_SWを反転して~入れてください
- CLKIN_IN:クロック入力
  - OSCを接続してください
- LOCK:ロック出力
  - クロック出力が安定になったら出力される
  - dcmLockと言う信号名で接続してください
  - ロックしていない時とリセットスイッチが押された時にRSTnがLowになるように書き換えてください

ex_dcm.v

```
P1 //////////////////////////////////////////////////////////////////////////////↓
 2 // Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.↓
 3 //////////////////////////////////////////////////////////////////////////////↓
 4 //   ____  ____ ↓
 5 //  /   /\/   / ↓
 6 // /___/  \  /    Vendor: Xilinx ↓
 7 // \   \   \/     Version : 8.1i↓
 8 //  \   \         Application : xaw2verilog↓
 9 //  /   /         Filename : ex_dcm.v↓
10 // /___/   /\     Timestamp : 05/25/2006 10:35:58↓
11 // \   \  /  \ ↓
12 //  \___\/\___\ ↓
13 //↓
14 //Command: xaw2verilog -st C:\Work\Ex1\ISE\ex_dcm.xaw C:\Work\Ex1\ISE\ex_dcm↓
15 //Design Name: ex_dcm↓
16 //Device: xc3s500e-fg320-4↓
17 //↓
18 // Module ex_dcm↓
19 // Generated by Xilinx Architecture Wizard↓
20 // Written for synthesis tool: XST↓
21 `timescale 1ns / 1ps↓
22 ↓
23 module ex_dcm(CLKIN_IN, ↓
24                RST_IN, ↓
25                CLKDV_OUT, ↓
26                CLKIN_IBUFG_OUT, ↓
27                CLK0_OUT, ↓
28                LOCKED_OUT);↓
29 ↓
30     input CLKIN_IN;↓
31     input RST_IN;↓
32    output CLKDV_OUT;↓
33    output CLKIN_IBUFG_OUT;↓
34    output CLK0_OUT;↓
35    output LOCKED_OUT;↓
36    ↓
37    wire CLKDV_BUF;↓
```

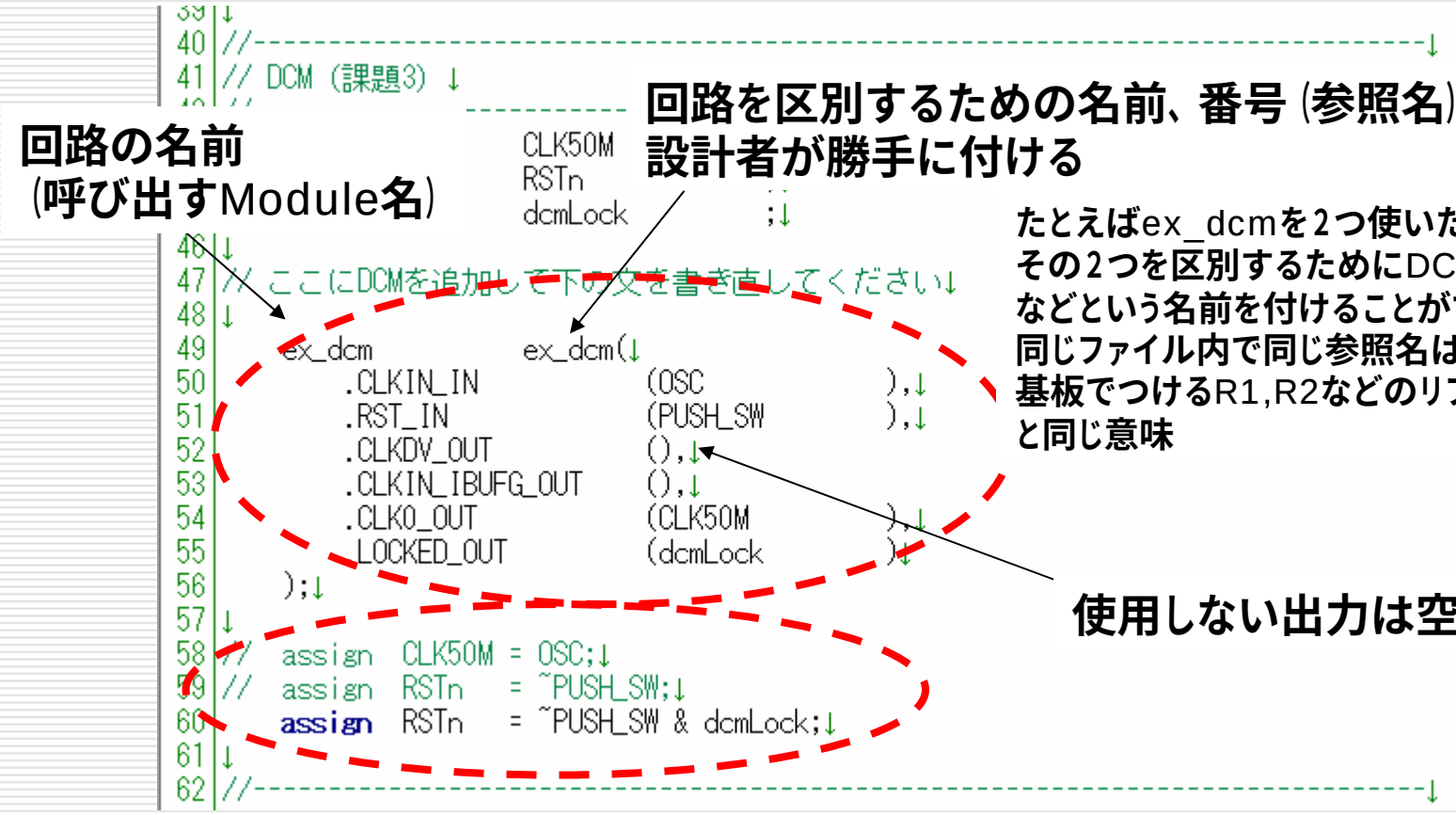
書き間違いを防ぐために
この部分をコピー

Ex1.v

```
39 // DCM (課題3) ↓
40 //----------------------------------------------------------------
41     wire            CLK50M          ;↓
42     wire            RSTn            ;↓
43 ↓
44 // ここにDCMを追加して下の文を書き直してください↓
45 ↓
46 ex_dcm(CLKIN_IN, ↓
47                 RST_IN, ↓
48                 CLKDV_OUT, ↓
49                 CLKIN_IBUFG_OUT, ↓
50                 CLK0_OUT, ↓
51                 LOCKED_OUT);↓
52 ↓
53     assign  CLK50M = OSC;↓
54     assign  RSTn   = ~PUSH_SW;↓
55 ↓
56 //----------------------------------------------------------------
```

ペースト

回路の組み込み


回路の名前
(呼び出すModule名)
回路を区別するための名前、番号 (参照名)
設計者が勝手に付ける
たとえばex_dcmを2つ使いたいとき
その2つを区別するためにDCM0,DCM1
などという名前を付けることができる。
同じファイル内で同じ参照名は使えない
基板でつけるR1,R2などのリファレンス番号
と同じ意味
使用しない出力は空けておく
// DCM (課題3)
CLK50M
RSTn
dcmLock
// ここにDCMを追加して下の文を書き直してください
ex_dcm ex_dcm(
.CLKIN_IN (OSC),
.RST_IN (PUSH_SW),
.CLKDV_OUT (),
.CLKIN_IBUFG_OUT (),
.CLK0_OUT (CLK50M),
.LOCKED_OUT (dcmLock)
);
// assign CLK50M = OSC;
// assign RSTn = ~PUSH_SW;
assign RSTn = ~PUSH_SW & dcmLock;

シミュレーションをする前に

- DCMの動作をSimulatorに教える必要あり
  - Simulation model
- もう一度ex_dcm.vを見てください

ex_dcm.v

実は動作は記述されていない

他のModule（回路）を使っている

```
41  wire GND1;↓
42  ↓
43  assign GND1 = 0;↓
44  assign CLKIN_IBUFG_OUT = CLKIN_IBUFG;↓
45  assign CLK0_OUT = CLKFB_IN;↓
46  BUFG CLKDV_BUFG_INST (.I(CLKDV_BUF), ↓
47                        .O(CLKDV_OUT));↓
48  IBUFG CLKIN_IBUFG_INST (.I(CLKIN_IN), ↓
49                          .O(CLKIN_IBUFG));↓
50  BUFG CLK0_BUFG_INST (.I(CLK0_BUF), ↓
51                       .O(CLKFB_IN));↓
52  DCM DCM_INST (.CLKFB(CLKFB_IN), ↓
53                .CLKIN(CLKIN_IBUFG), ↓
54                .DSSEN(GND1), ↓
```

他のModule（回路）はどこにあるのか？

Simulation Modelは何処にあるのか？

- /Xilinx/verilog/src/unisims/
  - このディレクトリの中にあります。
- このフォルダを開いてください
- さきほどのモデルがあることを確認してください
  - BUFG
  - IBUFG
  - DCM

と言う事でこれらを読み込む必要があります。

モデルを読み込む

- 何処で読み込むのか
  - EX1_TB.vの中
- なぜか？
  - シミュレーション・モデルだから
  - 実際はFPGA内の回路でトランジスタ・レベルの回路
  - 論理シミュレーションできるように
  - その動作と同じように振舞うだけで中身はまったく異なる。

モデルの読み込み

```
 8 *                                                                     *↓
 9 *                Copyright (c) 2006 Tomohisa Uchida                   *↓
10 *                All rights reserved                                  *↓
11 *
12 ************************************************************************
13 `timescale 1ps/1ps↓
14 ↓
15 `include "../../../Xilinx/verilog/src/glbl.v"↓
16 ↓
17 `include "../../../Xilinx/verilog/src/unisims/BUFG.v"↓
18 `include "../../../Xilinx/verilog/src/unisims/IBUFG.v"↓
19 `include "../../../Xilinx/verilog/src/unisims/DCM.v"↓
20 ↓
21 `include "../Src/EX1.v"↓
22 ↓
23 module EX1_TB;↓
24 ↓
25     reg             OSC             ;↓
26     reg             PUSH_SW         ;↓
27 ↓
```

Xilinxライブラリを使用するときのおまじない
初期化用ファイル

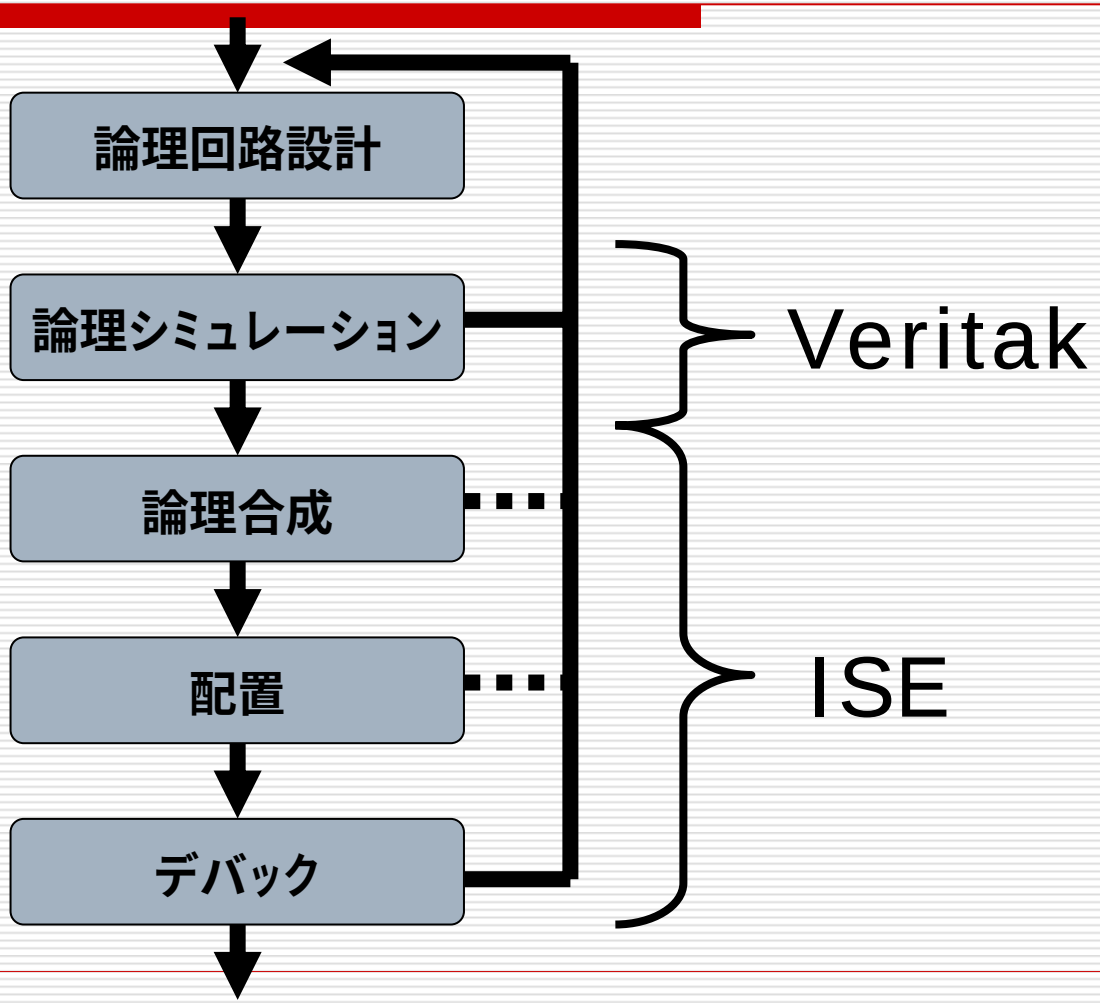
シミュレーション・モデルの読み込み

Simulatorの起動

- 先ほどと同じようにシミュレーションしてください
- 動きましたか？

- 最後に実装して確認しましょう！

作業の流れ

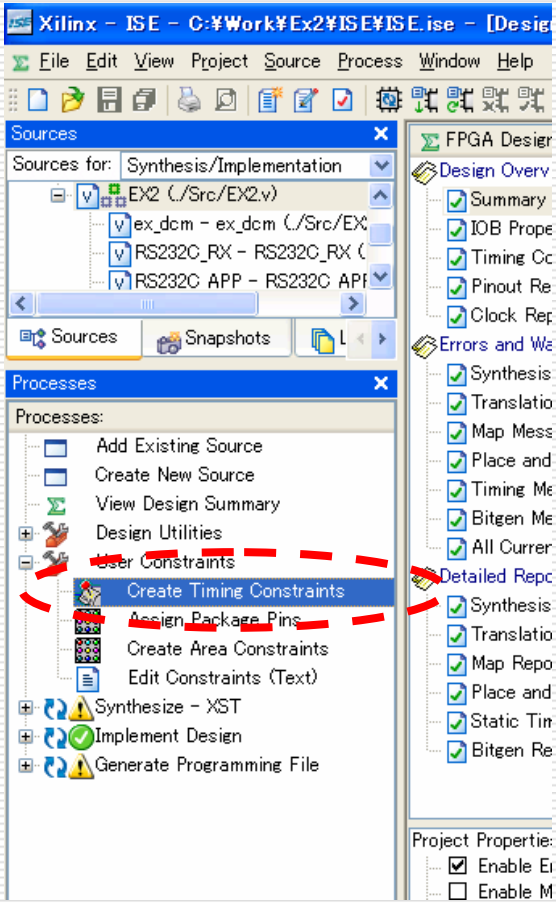


タイミング制約

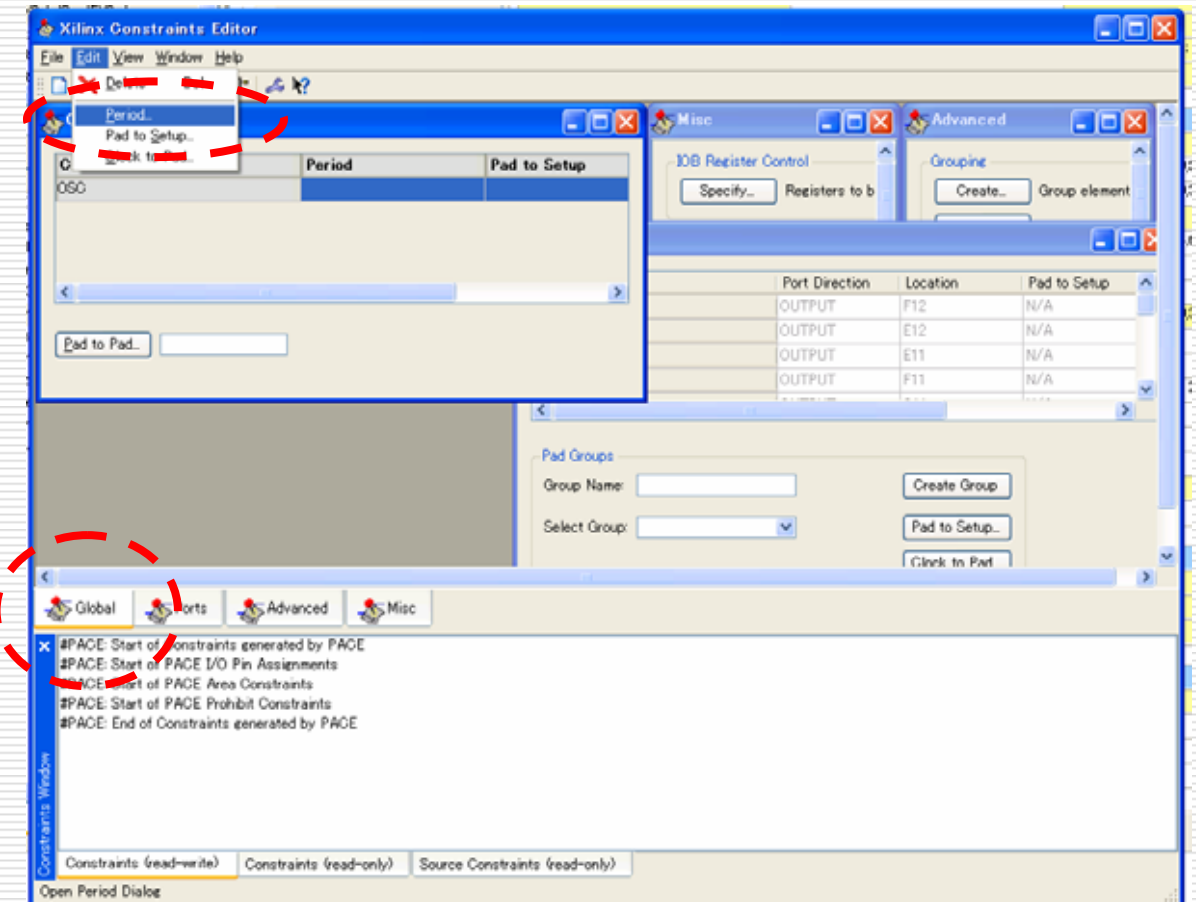
タイミング制約

- 動作周波数
  - 同期回路でよく使われる制約
  - 通常、設計時に動作させたい周波数がある
  - 満たされているかをツールが計算しレポートする

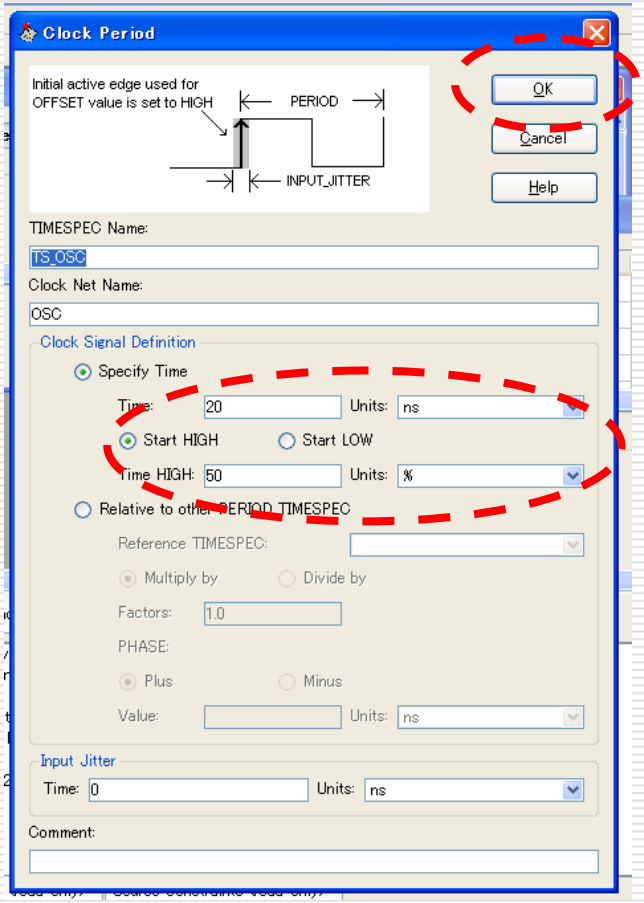
Create Timing Constraints



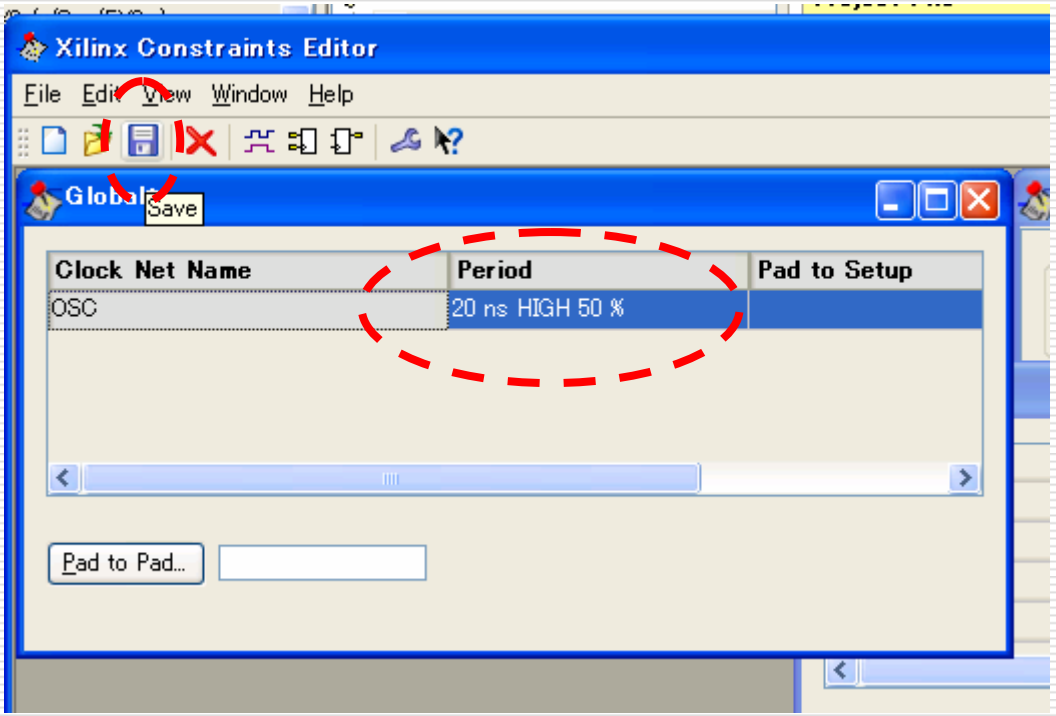
Create Timing Constraints



Create Timing Constraints



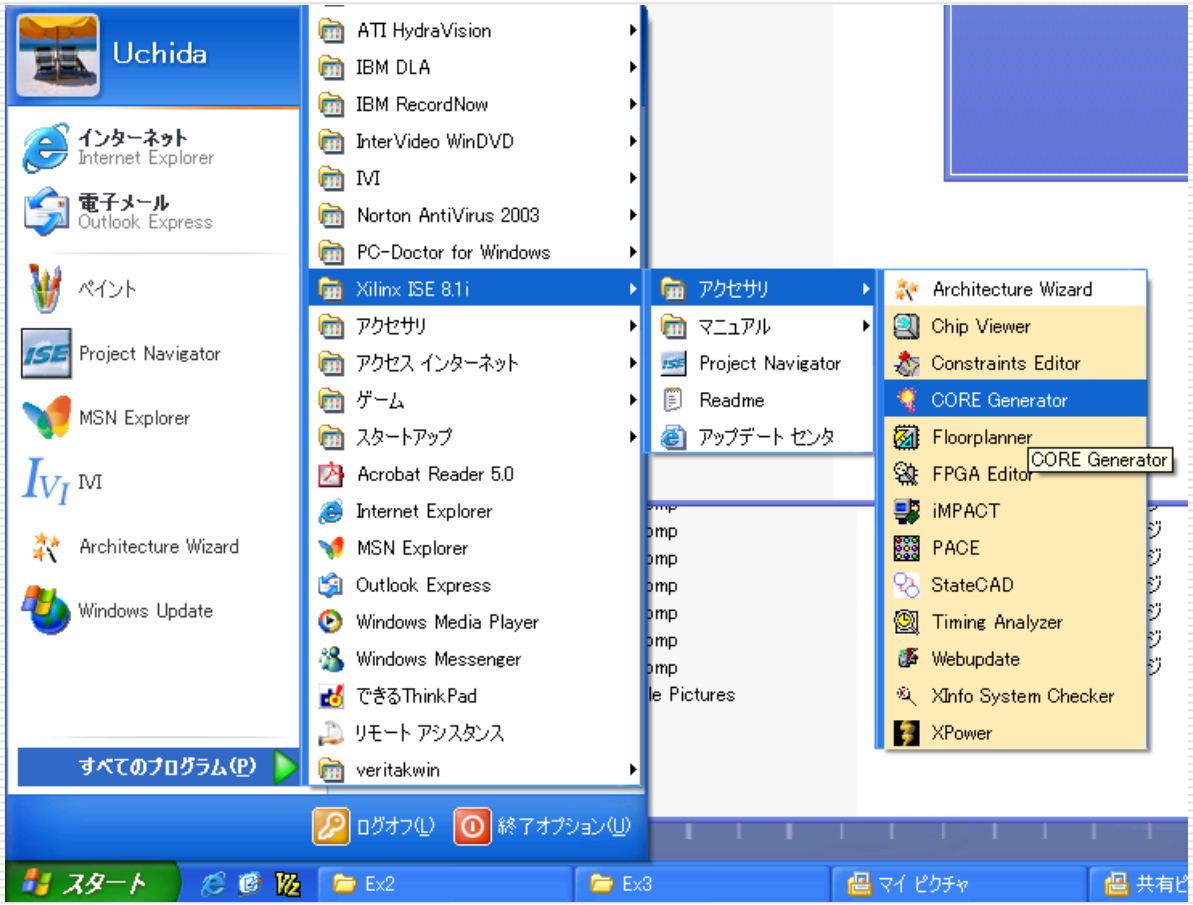
Create Timing Constraints



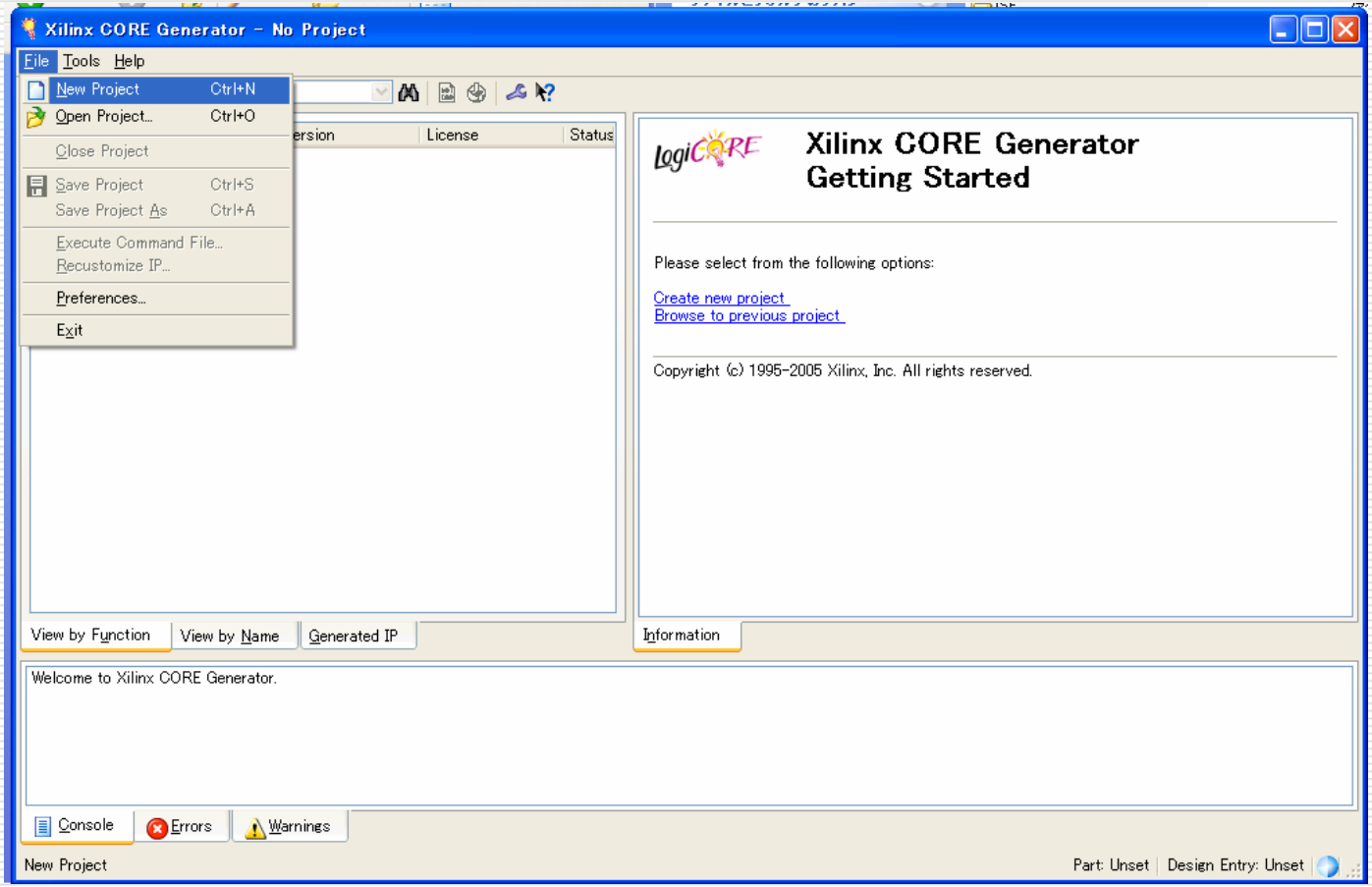
XILINXライブラリの使い方

- CORE generatorの使い方

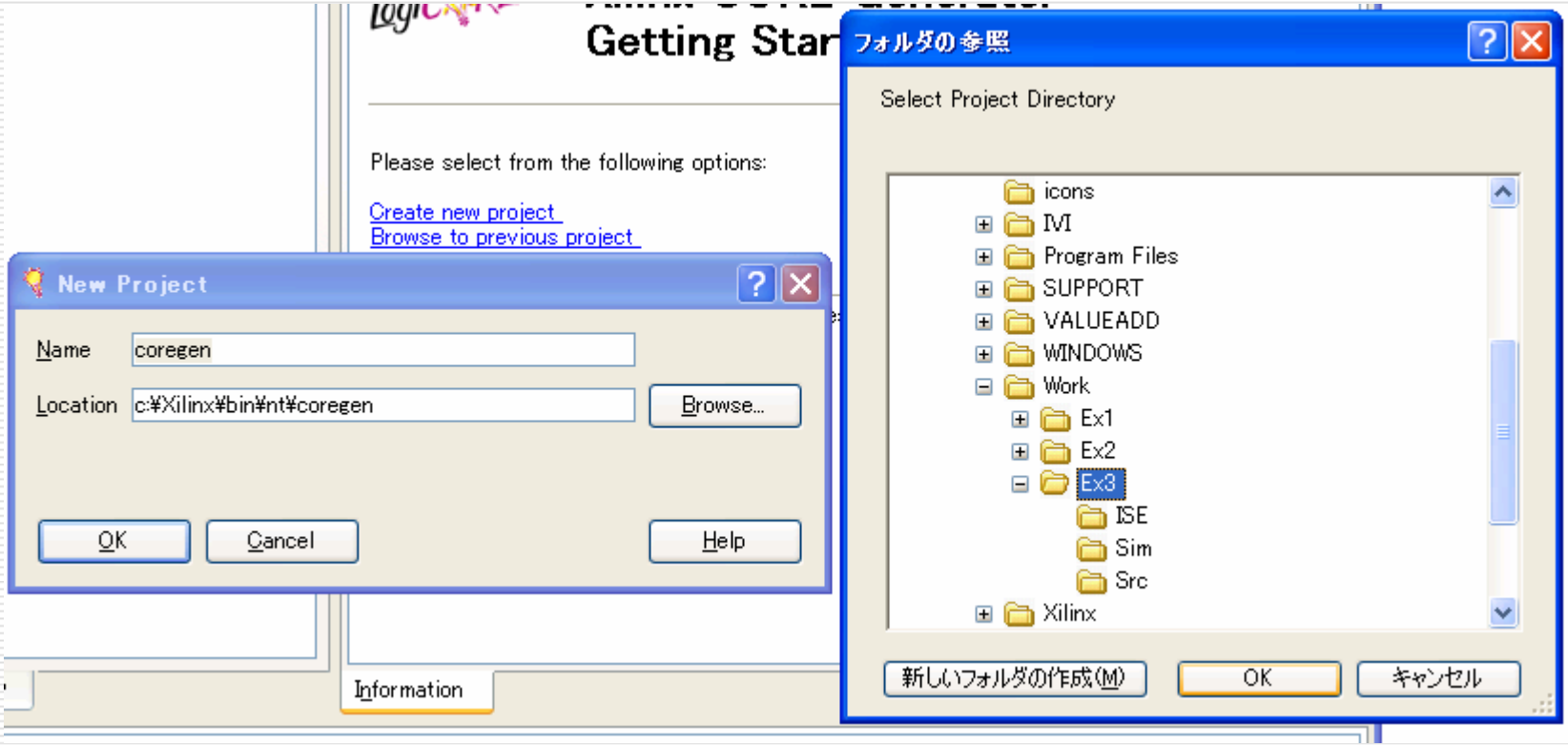
CORE Generatorの起動



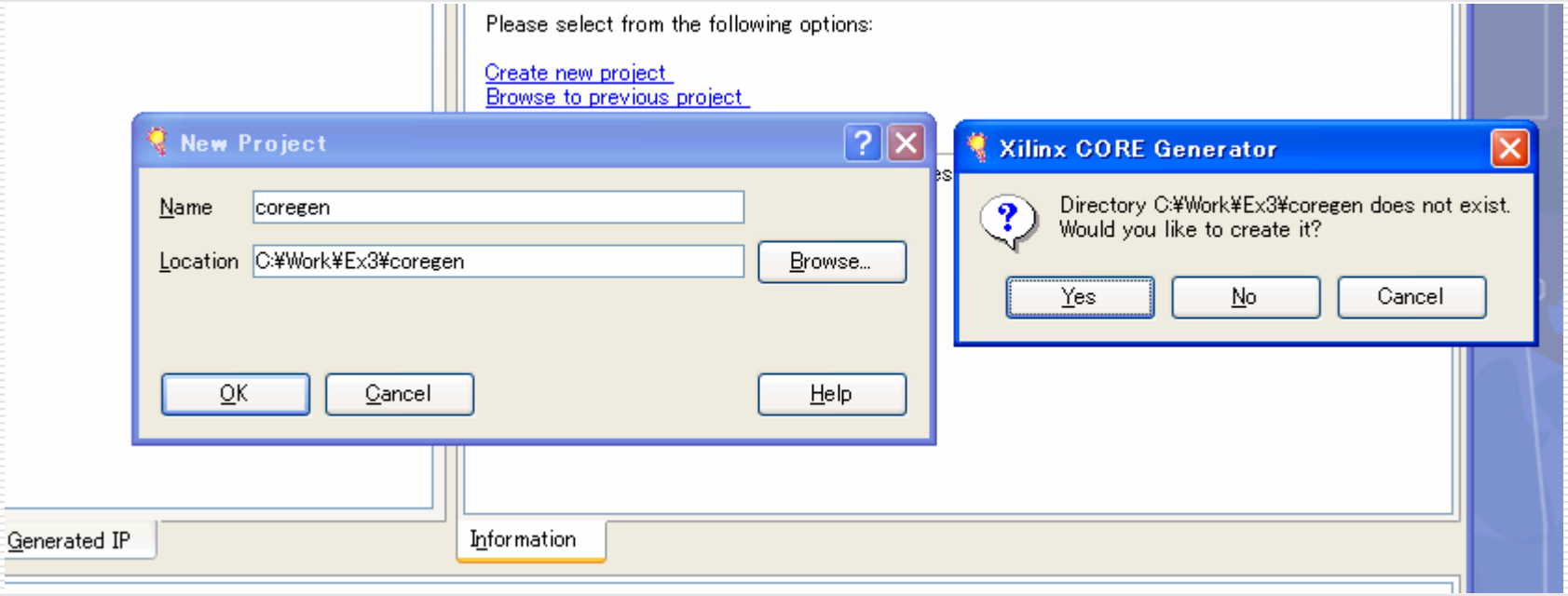
新プロジェクトの生成


Xilinx CORE Generator - No Project
File Tools Help
New Project Ctrl+N
Open Project... Ctrl+O
Close Project
Save Project Ctrl+S
Save Project As Ctrl+A
Execute Command File...
Recustomize IP...
Preferences...
Exit
License
Status
View by Function
View by Name
Generated IP
Xilinx CORE Generator
Getting Started
Please select from the following options:
Create new project
Browse to previous project
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
Information
Welcome to Xilinx CORE Generator.
Console
Errors
Warnings
New Project
Part: Unset
Design Entry: Unset

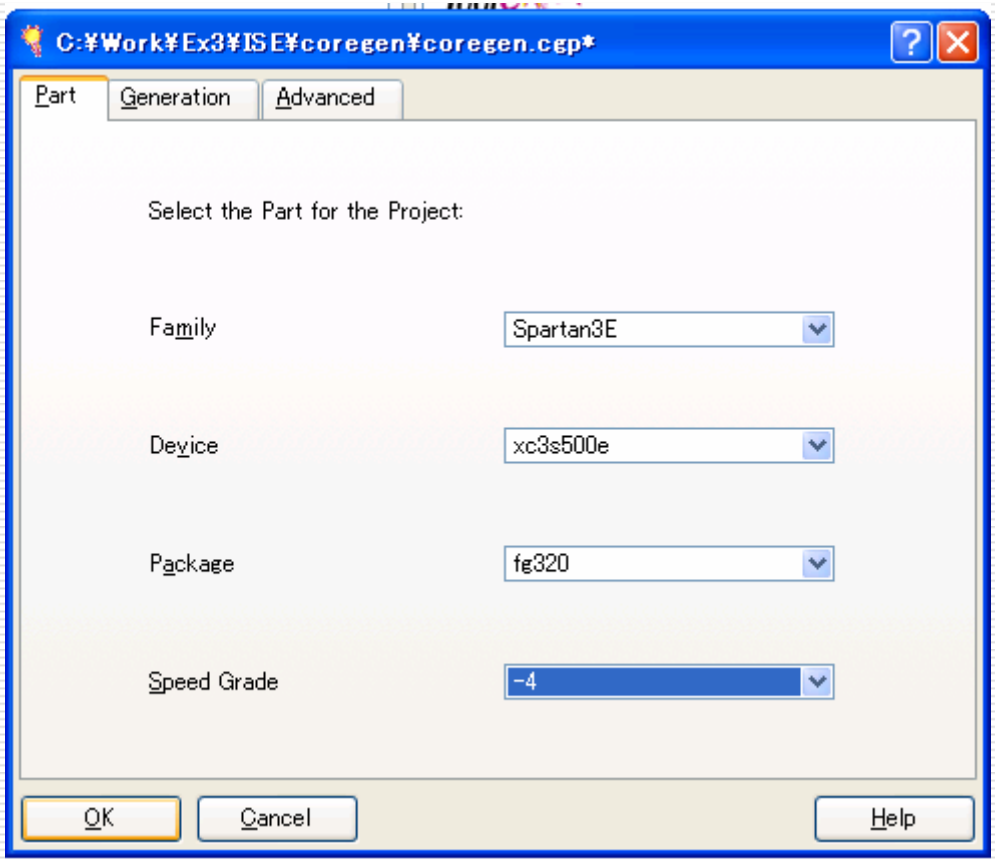
プロジェクトの新規作成



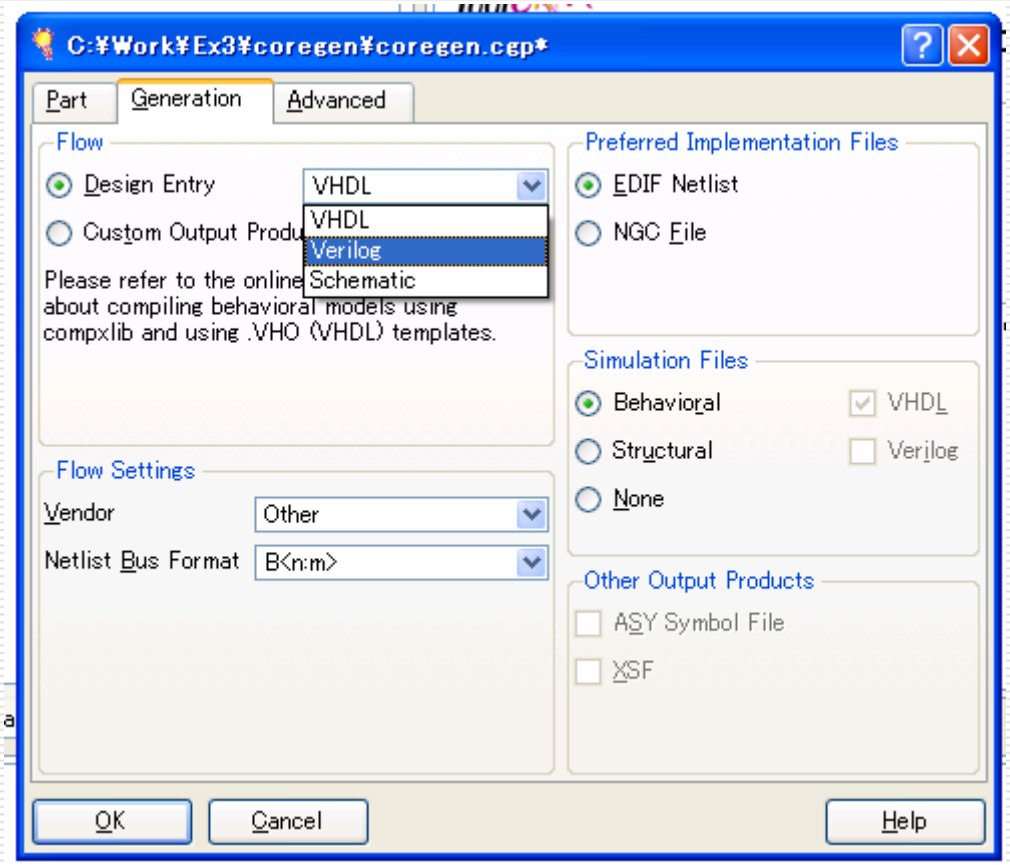
プロジェクトの新規作成



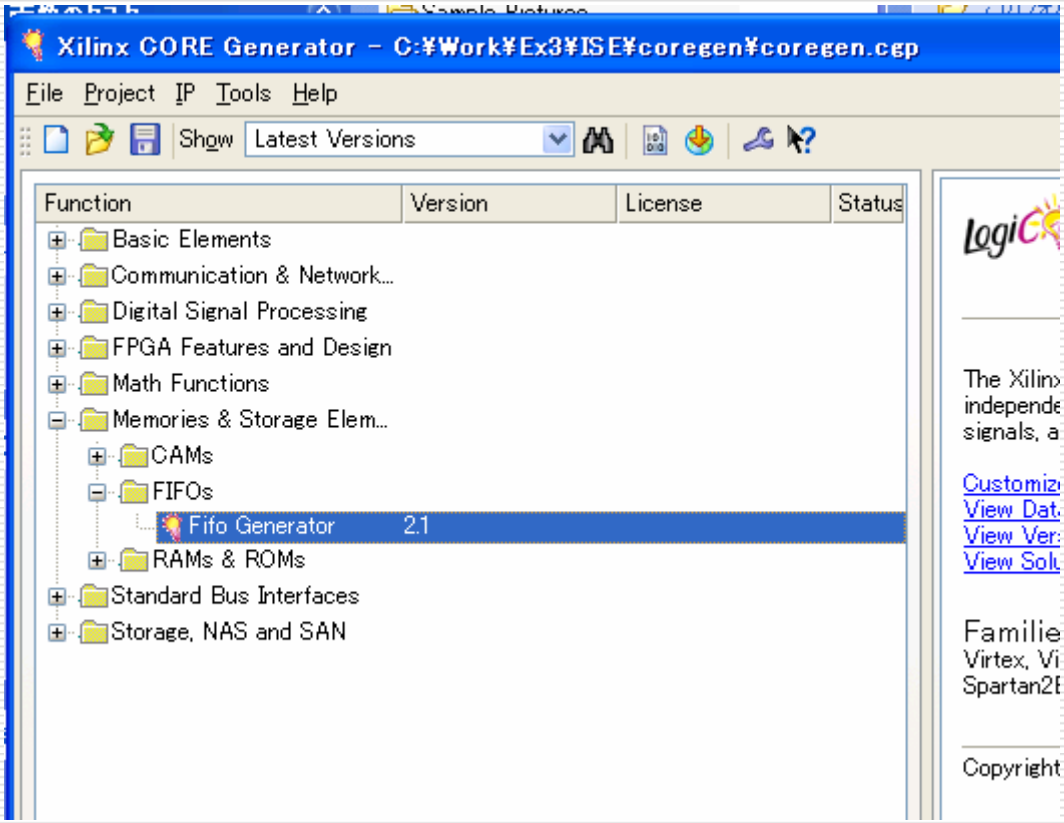
デバイスの選択


C:¥Work¥Ex3¥ISE¥coregen¥coregen.cgp*
Part
Generation
Advanced
Select the Part for the Project:
Family
Spartan3E
Device
xc3s500e
Package
fg320
Speed Grade
-4
OK
Cancel
Help

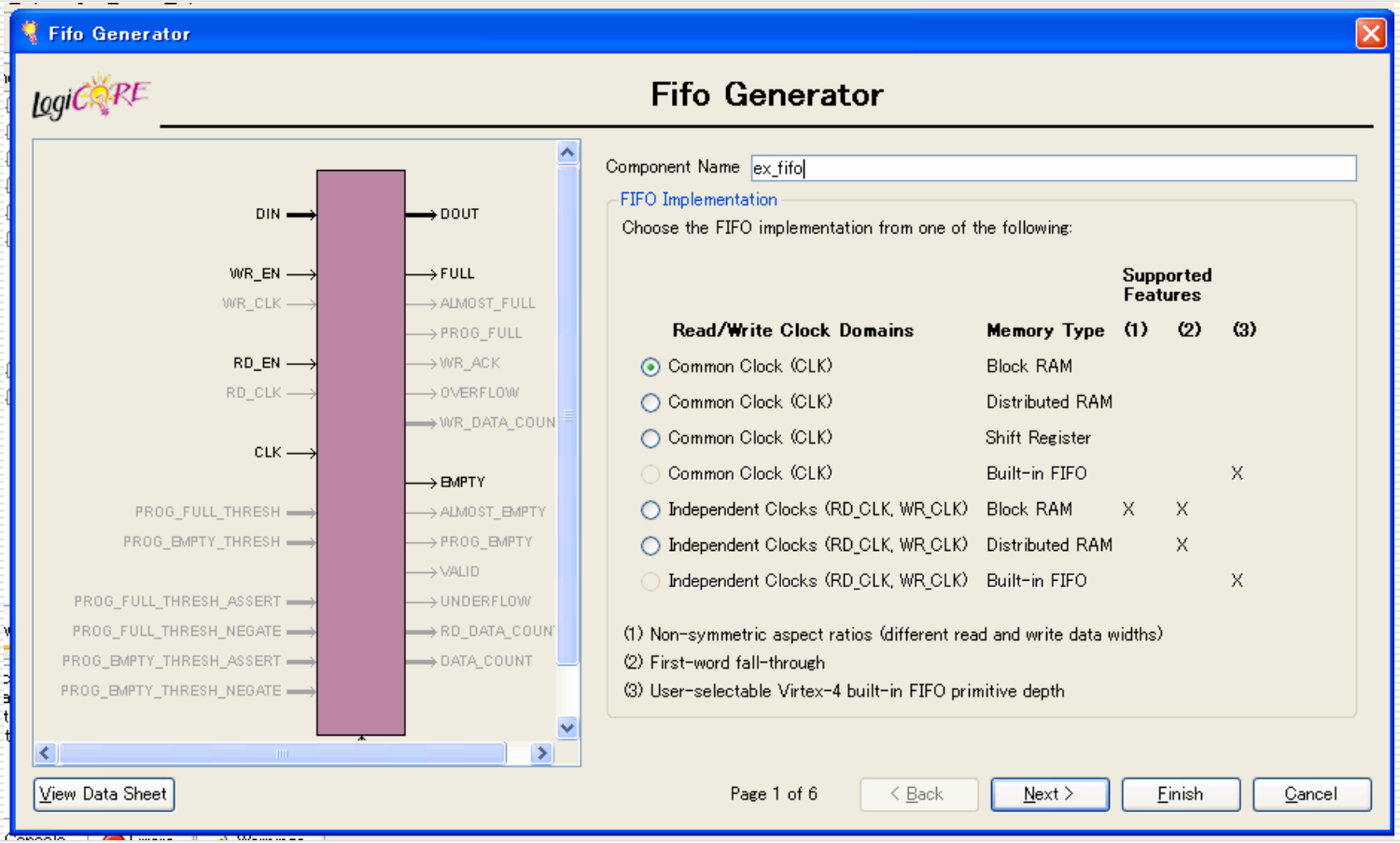
言語の選択


C:¥Work¥Ex3¥coregen¥coregen.cgp*
Part
Generation
Advanced
Flow
Design Entry
VHDL
VHDL
Verilog
Schematic
Custom Output Produ
Please refer to the online
about compiling behavioral models using
compxlib and using .VHO (VHDL) templates.
Preferred Implementation Files
EDIF Netlist
NGC File
Simulation Files
Behavioral
VHDL
Structural
Verilog
None
Flow Settings
Vendor
Other
Netlist Bus Format
B<n:m>
Other Output Products
ASY Symbol File
XSF
OK
Cancel
Help

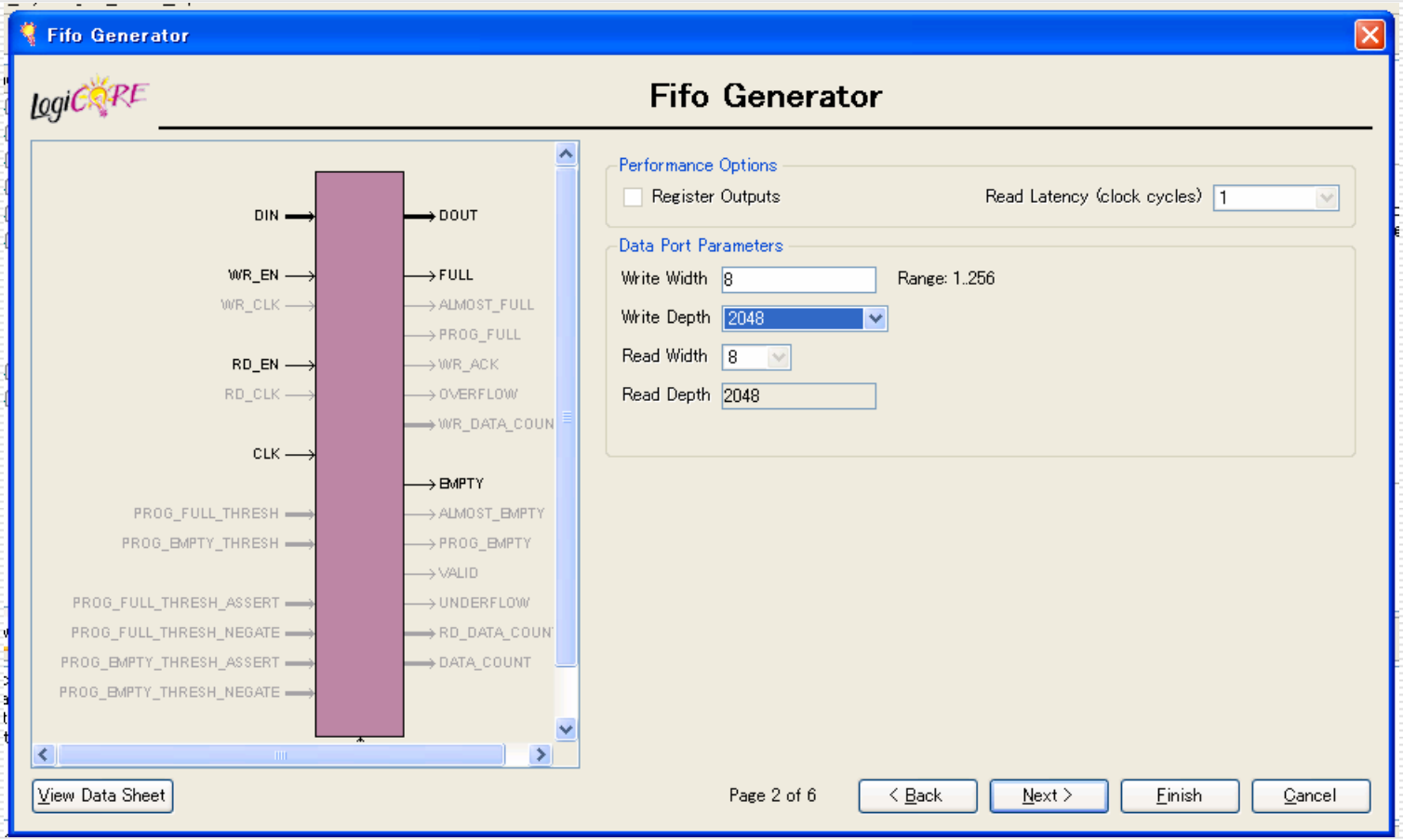
FIFOの生成


Xilinx CORE Generator - C:¥Work¥Ex3¥ISE¥coregen¥coregen.cgp
File Project IP Tools Help
Show Latest Versions
Function
Version
License
Status
Basic Elements
Communication & Network...
Digital Signal Processing
FPGA Features and Design
Math Functions
Memories & Storage Elem...
CAMs
FIFOs
Fifo Generator 2.1
RAMs & ROMs
Standard Bus Interfaces
Storage, NAS and SAN

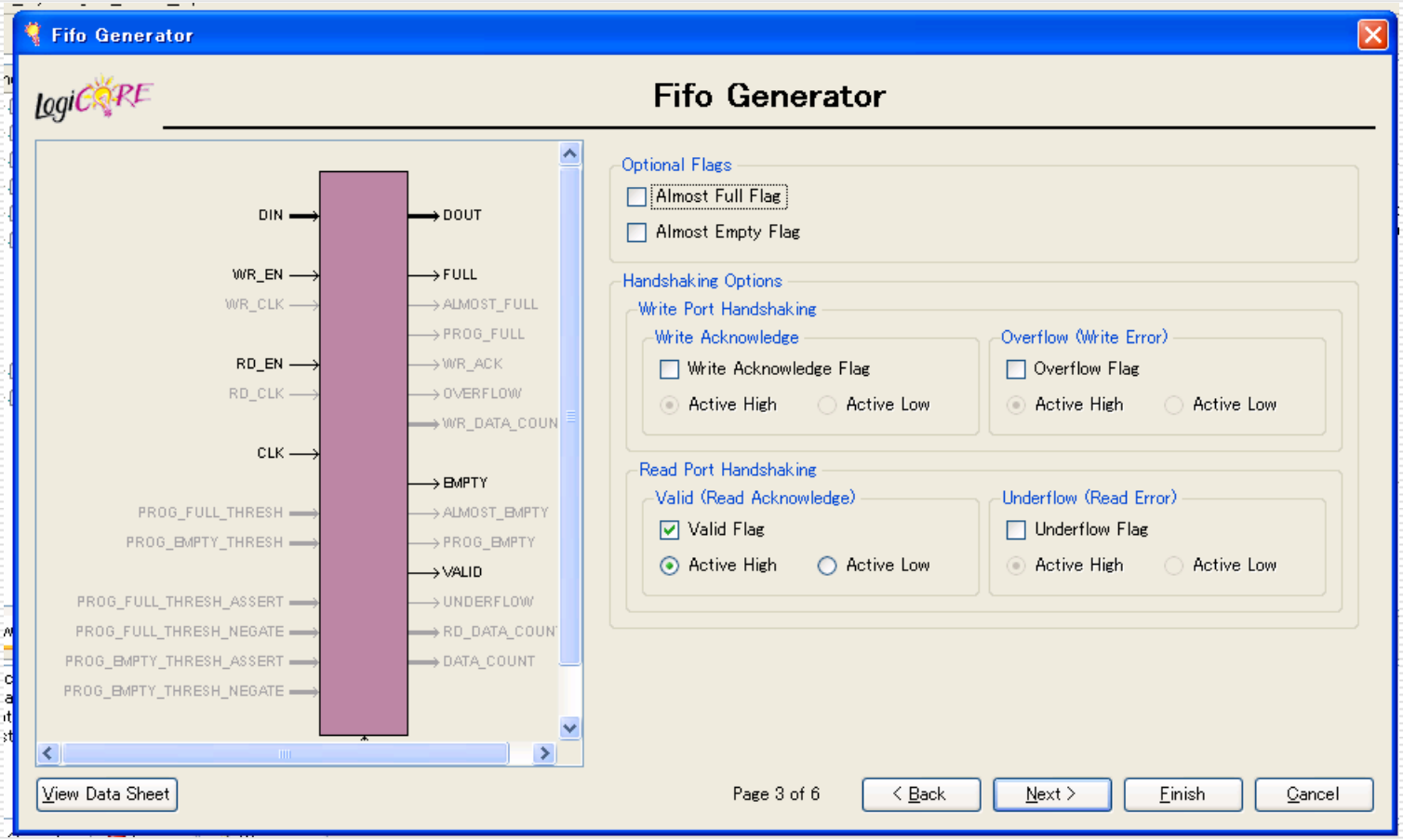
セットアップ


Fifo Generator
LogiCORE
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
Component Name ex_fifo
FIFO Implementation
Choose the FIFO implementation from one of the following:
Supported Features
Read/Write Clock Domains
Memory Type
(1) (2) (3)
Common Clock (CLK) Block RAM
Common Clock (CLK) Distributed RAM
Common Clock (CLK) Shift Register
Common Clock (CLK) Built-in FIFO X
Independent Clocks (RD_CLK, WR_CLK) Block RAM X X
Independent Clocks (RD_CLK, WR_CLK) Distributed RAM X
Independent Clocks (RD_CLK, WR_CLK) Built-in FIFO X
(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-word fall-through
(3) User-selectable Virtex-4 built-in FIFO primitive depth
View Data Sheet
Page 1 of 6
< Back
Next >
Finish
Cancel

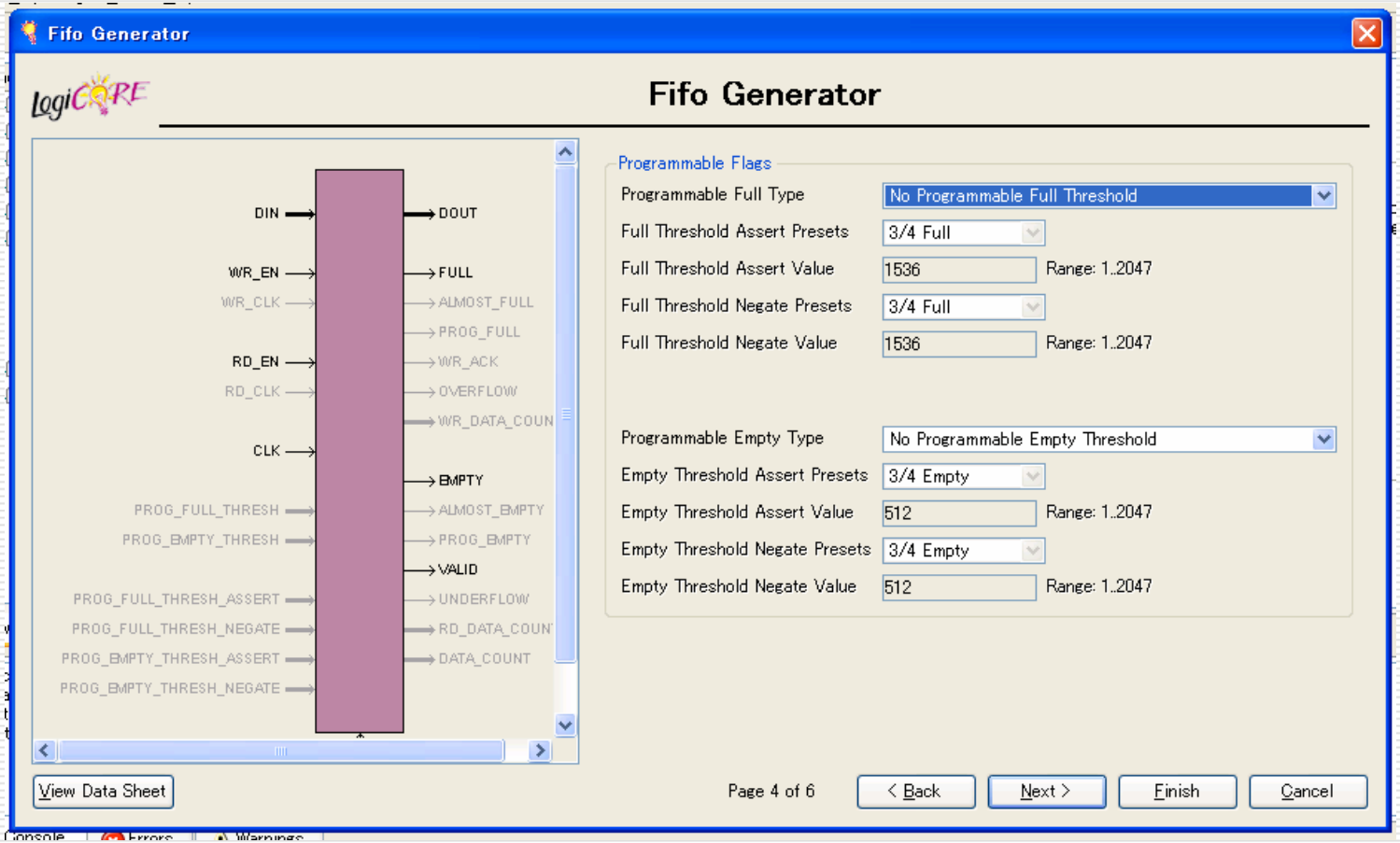
セットアップ


Fifo Generator
LogiCORE
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
Performance Options
Register Outputs
Read Latency (clock cycles) 1
Data Port Parameters
Write Width 8
Range: 1..256
Write Depth 2048
Read Width 8
Read Depth 2048
View Data Sheet
Page 2 of 6
< Back
Next >
Finish
Cancel

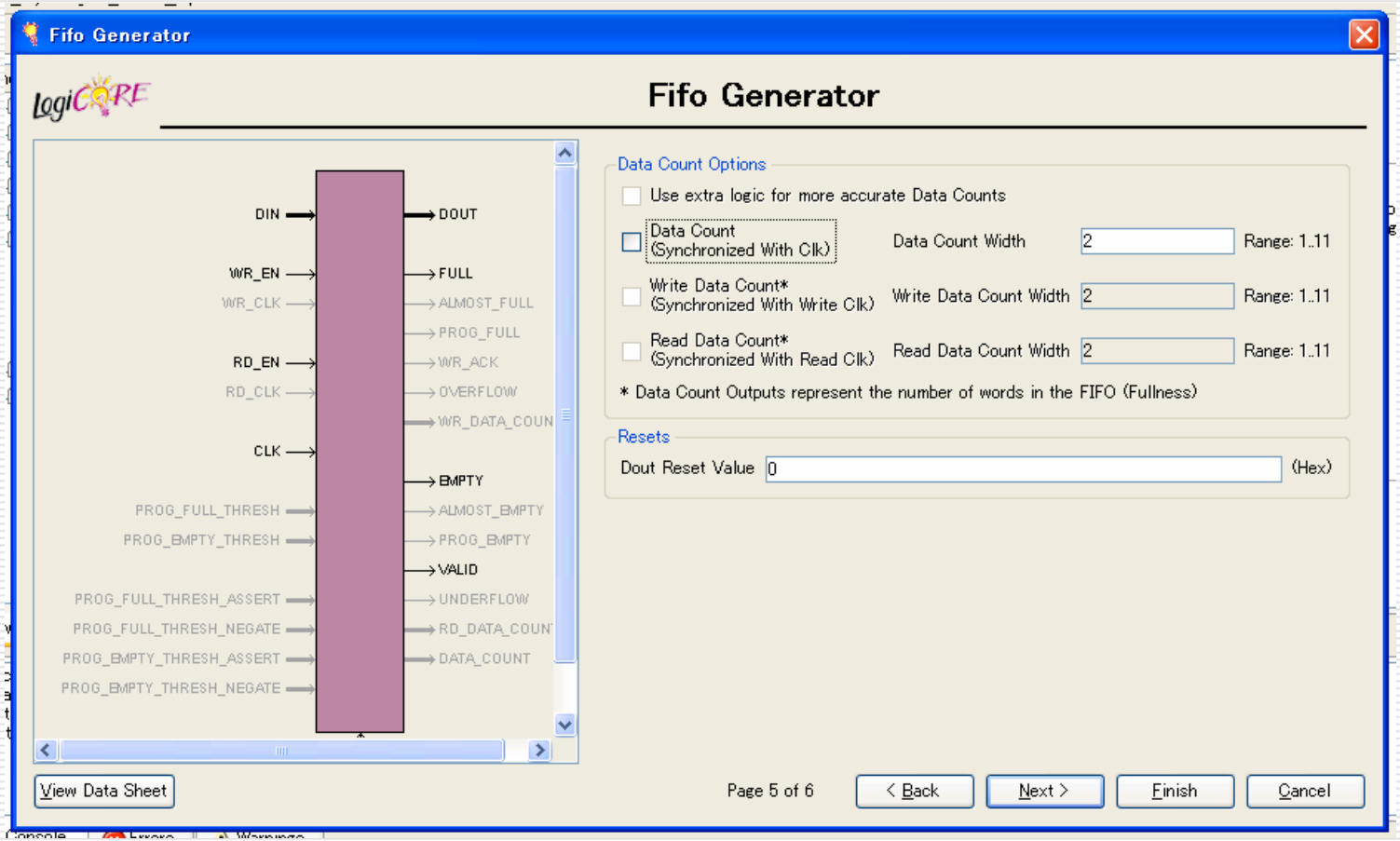
セットアップ


Fifo Generator
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
Optional Flags
Almost Full Flag
Almost Empty Flag
Handshaking Options
Write Port Handshaking
Write Acknowledge
Write Acknowledge Flag
Active High
Active Low
Overflow (Write Error)
Overflow Flag
Active High
Active Low
Read Port Handshaking
Valid (Read Acknowledge)
Valid Flag
Active High
Active Low
Underflow (Read Error)
Underflow Flag
Active High
Active Low
View Data Sheet
Page 3 of 6
< Back
Next >
Finish
Cancel

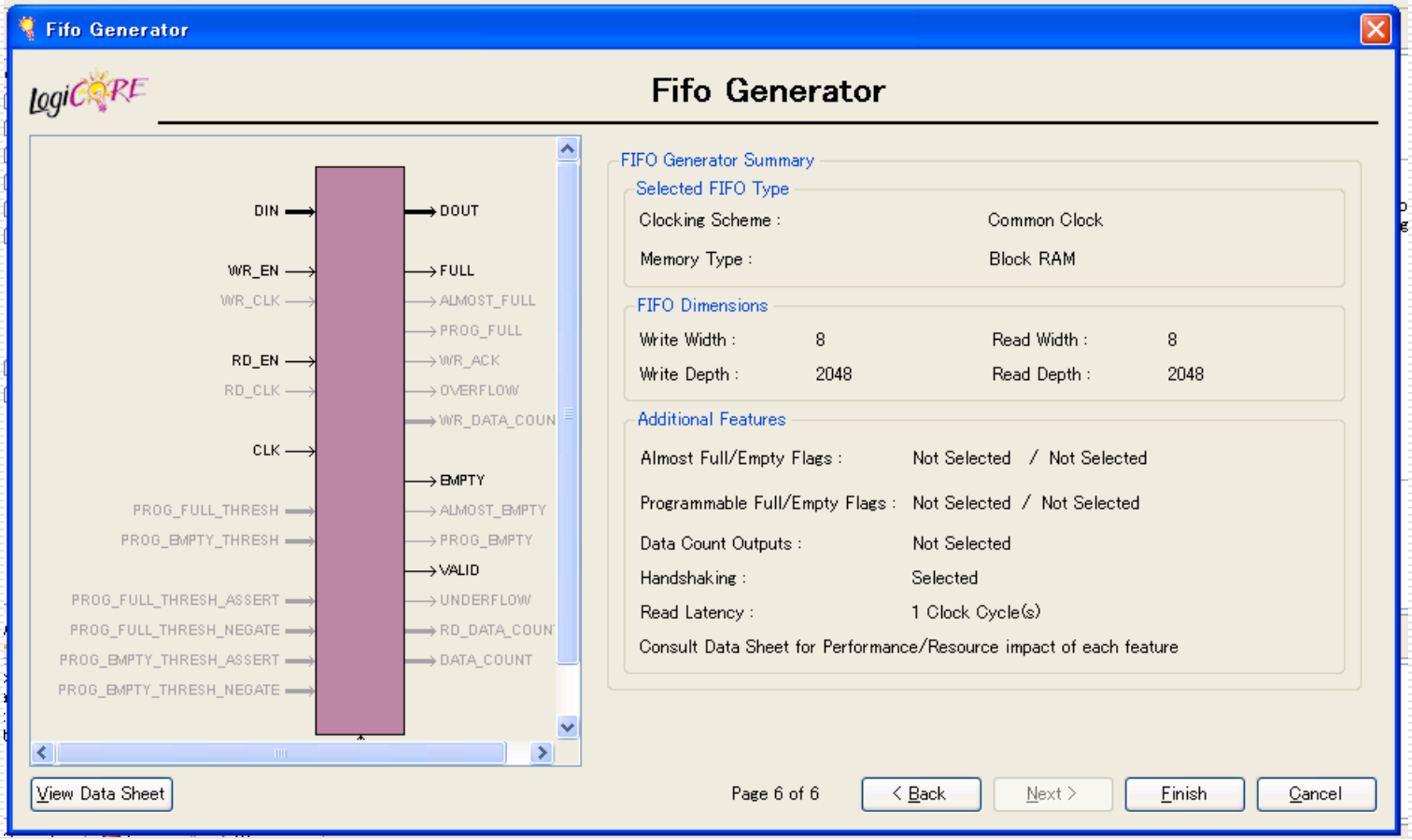
セットアップ


Fifo Generator
LogiCORE
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
Programmable Flags
Programmable Full Type
No Programmable Full Threshold
Full Threshold Assert Presets
3/4 Full
Full Threshold Assert Value
1536
Range: 1..2047
Full Threshold Negate Presets
3/4 Full
Full Threshold Negate Value
1536
Range: 1..2047
Programmable Empty Type
No Programmable Empty Threshold
Empty Threshold Assert Presets
3/4 Empty
Empty Threshold Assert Value
512
Range: 1..2047
Empty Threshold Negate Presets
3/4 Empty
Empty Threshold Negate Value
512
Range: 1..2047
View Data Sheet
Page 4 of 6
< Back
Next >
Finish
Cancel

セットアップ


Fifo Generator
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
Data Count Options
Use extra logic for more accurate Data Counts
Data Count (Synchronized With Clk)
Data Count Width
2
Range: 1..11
Write Data Count* (Synchronized With Write Clk)
Write Data Count Width
2
Range: 1..11
Read Data Count* (Synchronized With Read Clk)
Read Data Count Width
2
Range: 1..11
* Data Count Outputs represent the number of words in the FIFO (Fullness)
Resets
Dout Reset Value
0
(Hex)
View Data Sheet
Page 5 of 6
< Back
Next >
Finish
Cancel

セットアップ

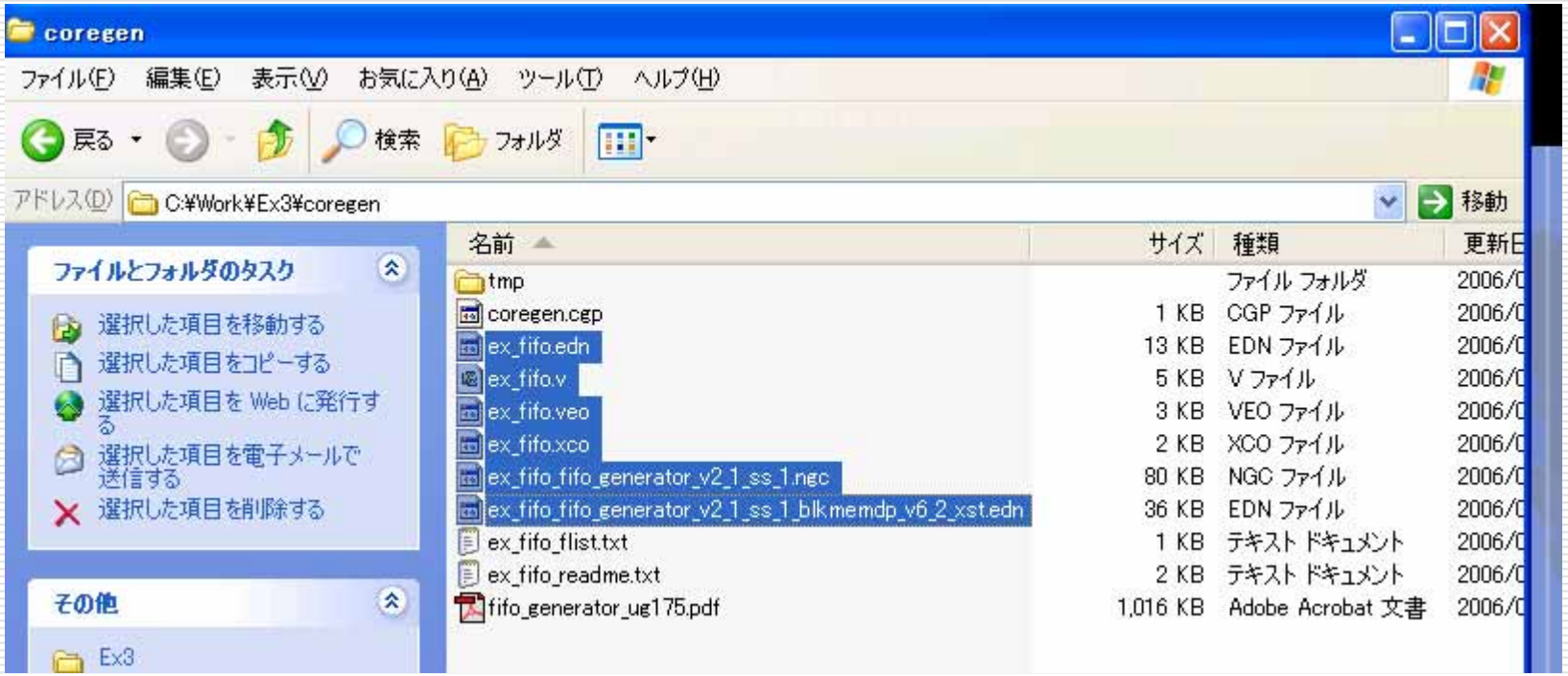

Fifo Generator
LogiCORE
Fifo Generator
DIN
WR_EN
WR_CLK
RD_EN
RD_CLK
CLK
PROG_FULL_THRESH
PROG_EMPTY_THRESH
PROG_FULL_THRESH_ASSERT
PROG_FULL_THRESH_NEGATE
PROG_EMPTY_THRESH_ASSERT
PROG_EMPTY_THRESH_NEGATE
DOUT
FULL
ALMOST_FULL
PROG_FULL
WR_ACK
OVERFLOW
WR_DATA_COUN
EMPTY
ALMOST_EMPTY
PROG_EMPTY
VALID
UNDERFLOW
RD_DATA_COUN
DATA_COUNT
FIFO Generator Summary
Selected FIFO Type
Clocking Scheme : Common Clock
Memory Type : Block RAM
FIFO Dimensions
Write Width : 8
Read Width : 8
Write Depth : 2048
Read Depth : 2048
Additional Features
Almost Full/Empty Flags : Not Selected / Not Selected
Programmable Full/Empty Flags : Not Selected / Not Selected
Data Count Outputs : Not Selected
Handshaking : Selected
Read Latency : 1 Clock Cycle(s)
Consult Data Sheet for Performance/Resource impact of each feature
View Data Sheet
Page 6 of 6
< Back
Next >
Finish
Cancel

ソースへの組み込み方

- DCMの時を思い出してください
- ただし、Simulation Modelがあるディレクトリが異なります
  - /Xilinx/verilog/src/XilinxCoreLib/
- 実際にコードを見てみましょう

生成されたファイル

ISEのワークフォルダへコピーする必要があります
(Work/Ex3/ISE)



再生成した時にコピーするのを忘れないようにしてください

回路ファイル ex_fifo.v

```
40 module ex_fifo(↓
41     clk,↓
42     din,↓
43     rd_en,↓
44     rst,↓
45     wr_en,↓
46     dout,↓
47     empty,↓
48     full,↓
49     valid);↓
50 ↓
51 ↓
52 input clk;↓
53 input [7 : 0] din;↓
54 input rd_en;↓
55 input rst;↓
56 input wr_en;↓
57 output [7 : 0] dout;↓
58 output empty;↓
59 output full;↓
60 output valid;↓
61 ↓
62 // synopsys translate_off↓
63 ↓
64       FIFO_GENERATOR_V2_1 #(↓
65         1,  // c_common_clock↓
66         0,  // c_count_type↓
67         2,  // c_data_count_width↓
68         "BlankString",  // c_default_value↓
69         8,  // c_din_width↓
```

Ex_fifoの読み込み

```
 5 * Module      : EX3                                                    *↓
 6 * Version     : v 0.0.0 2006/05/24 09:27                               *↓
 7 *                                                                      *↓
 8 * Description : Example 3                                             *↓
 9 *               Core-Generator FIFO                                    *↓
10 *                                                                      *↓
11 * Designer    : Tomohisa Uchida                                        *↓
12 *                                                                      *↓
13 *                Copyright (c) 2006 T                                  *↓
14 *                All rights reserved                                   *↓
15 *                                                                      *↓
16 ***********************************************************************/↓
17 ↓
18 `include "../ISE/ex_fifo.v"↓
19 ↓
20 module EX3(↓
21     CLK                 ,   // in   : Clock↓
22     RST                 ,   // in   : Reset↓
23     WE                  ,   // in   : Write enable↓
24     WD                  ,   // in   : Write data[7:0]↓
25     FULL                ,   // in   : Full flag↓
26     RD                  ,   // in   : Read data[7:0]↓
27     RE                  ,   // in   : Read enable↓
28     EMPTY               ,   // in   : Emoty flag↓
29     RV                      // out  : Read data valid↓
30 );↓
31 ↓
32 // ----- input/ output -----↓
33     input           CLK         ;↓
34     input           RST         ;↓
```

FIFOの読み込み

EX3_TB.vファイル

```
11 *                                                                              *↓
12 ********************************************************************************/↓
13 `define SIM↓
14 ↓
15 `timescale 1ps/1ps↓
16 ↓
17 `include "../../../Xilinx/verilog/src/glbl.v"↓
18 ↓
19 `include "../../../Xilinx/verilog/src/XilinxCoreLib/FIFO_GENERATOR_V2_1.v"↓
20 ↓
21 `include "../Src/EX3.v"↓
22 ↓
23 module EX3_TB;↓
24 ↓
25     reg             OSC             ;↓
26     reg             PUSH_SW         ;↓
27     reg             FIFO_WE         ;↓
28     reg     [7:0]   FIFO_WD         ;↓
29 ↓
30     wire            FIFO_FULL       ;↓
31     wire            FIFO_EMPTY      ;↓
32     wire            FIFO_RV         ;↓
33     wire    [7:0]   FIFO_RD         ;↓
34 ↓
35 //------------------------------------------------------------------------------↓
36 //  DUT↓
37 //------------------------------------------------------------------------------↓
38     EX3             DUT(↓
39         .CLK                (OSC            ),  // in   : Clock↓
40         .RST                (PUSH_SW        ),  // in   : Reset↓
41         .WE                 (FIFO_WE        ),  // in   : Write enable↓
42         .WD                 (FIFO_WD[7:0]   ),  // in   : Write data[7:0]↓
43         .FULL               (FIFO_FULL      ),  // in   : Full flag↓
44         .RD                 (FIFO_RD[7:0]   ),  // in   : Read data[7:0]↓
45         .RE                 (~FIFO_EMPTY    ),  // in   : Read enable↓
46         .EMPTY              ( FIFO_EMPTY    ),  // in   : Emoty flag↓
47         .RV                 (FIFO_RV        )   // out  : Read data valid↓
48     );↓
49 ↓
```

モデルの読み込み

ライブラリについて

- ここではFIFO_GENERATOR_V2_1.vひとつ読み込みました
- モデルによってはモデルの中で、さらに他のモジュールを読み込む必要がある場合もあります
  - ツールのエラーメッセージに注意して下さい
  - また、モデルのファイルを見てください
    - モジュールを読み込んでいる記述を見つける
  - そのモデルをunisims、XilinxCoreLibのどちらかにあるはずです。