計算機科学実験及演習2
ハードウェア
「CADツール上での論理設計」

高瀬，川原，小谷，加藤，
高木(一)，玉置
le2hw@lab3.kuis.kyoto-u.ac.jp

実験概要

- CADツールQuartus Primeの使用方法を習得し，論理回路の設計手法を学ぶ.
- マイクロプロセッサの主要な構成要素であるALUについて，複数の回路構成方式を検討し，その性能や回路規模を比較する.
- 任意の順序回路を考案し，CADツールを用いて設計する.

スケジュール

- 2019/12/24：講義・実習
  - 講義：FPGAことはじめ
  - 実習：Quartus Primeの使い方
    加算器・カウンタの設計
    ✓講義資料はPandAを参照のこと
- 2020/01/07： 演習(1) ALUの設計
- 2020/01/21： 演習(2) 任意の順序回路の設計

- 2020/01/28 17:00：レポート提出期限
  - **期限厳守！**

講義内容

- FPGAことはじめ
  - 基礎知識と構成
  - 活用のメリットと用途
  - CADツールと設計フロー
- 実習課題(1)：加算回路の設計
  - 半加算器を用いた全加算器
  - 全加算器を用いた4ビット並列加算回路
- 実習課題(2)：カウンタの設計
  - フリップフロップの利用
  - クロックとタイミング制約の設定

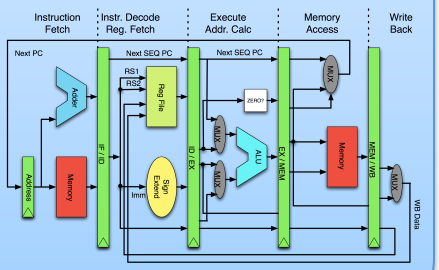
FPGAことはじめ

– 基礎知識と構成
– 活用のメリットと用途
– CADツールと設計フロー

計算資源

「言われたことはなんでもやる」
- すべてのケースをカバーする汎用データパス
- 固定されたデータ幅と演算

プロセッサ
&
ソフトウェア

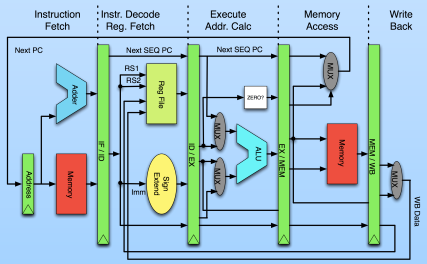


「決められたことをすぐにやる」
- 特定用途のみに特化した専用のアーキテクチャ構成
- 必要なだけのデータ幅と演算

ASIC
(専用回路)
ハードウェア

プロセッサ vs. ASIC

プロセッサ
&
ソフトウェア



性能・並列性

設計容易性・柔軟性

省電力性

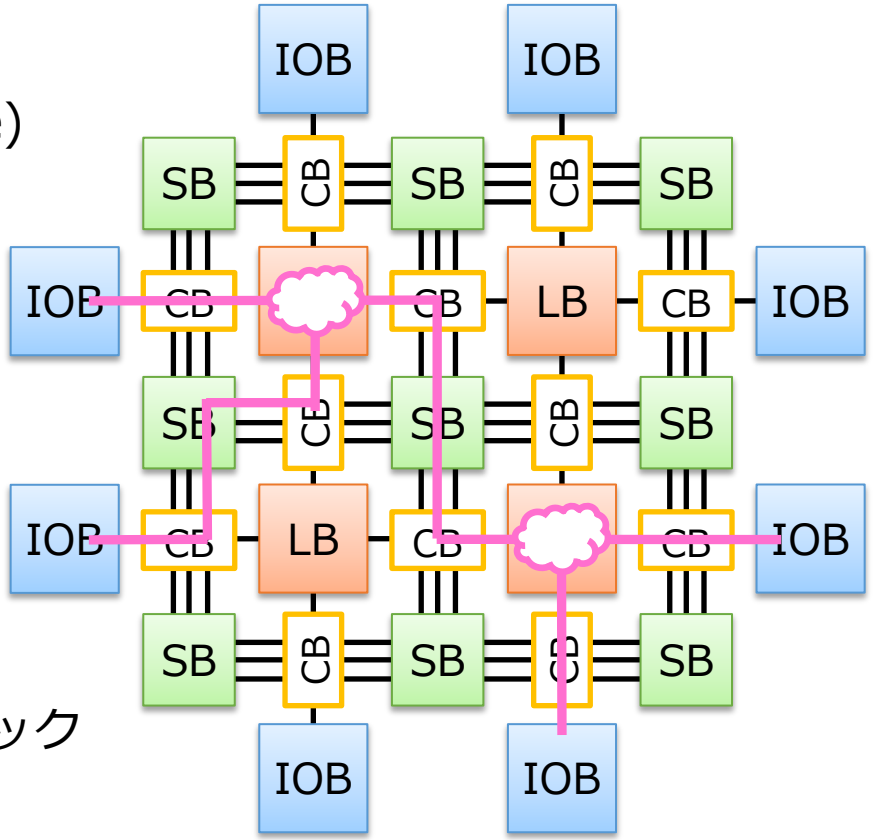
開発製造コスト

FPGA

ASIC
(専用回路)
ハードウェア

FPGAとは？

- Field Programmable Gate Array
  - 中身を改変可能なLSI
    (PLD: Programmable Logic Device)
  - ハードウェアそのものの
    振る舞いを変えられる
  - 独自のデジタル回路を
    自由に何回でも形成できる
  - FPGA二大ベンダ：
    Xilinx・Altera (powered by Intel)

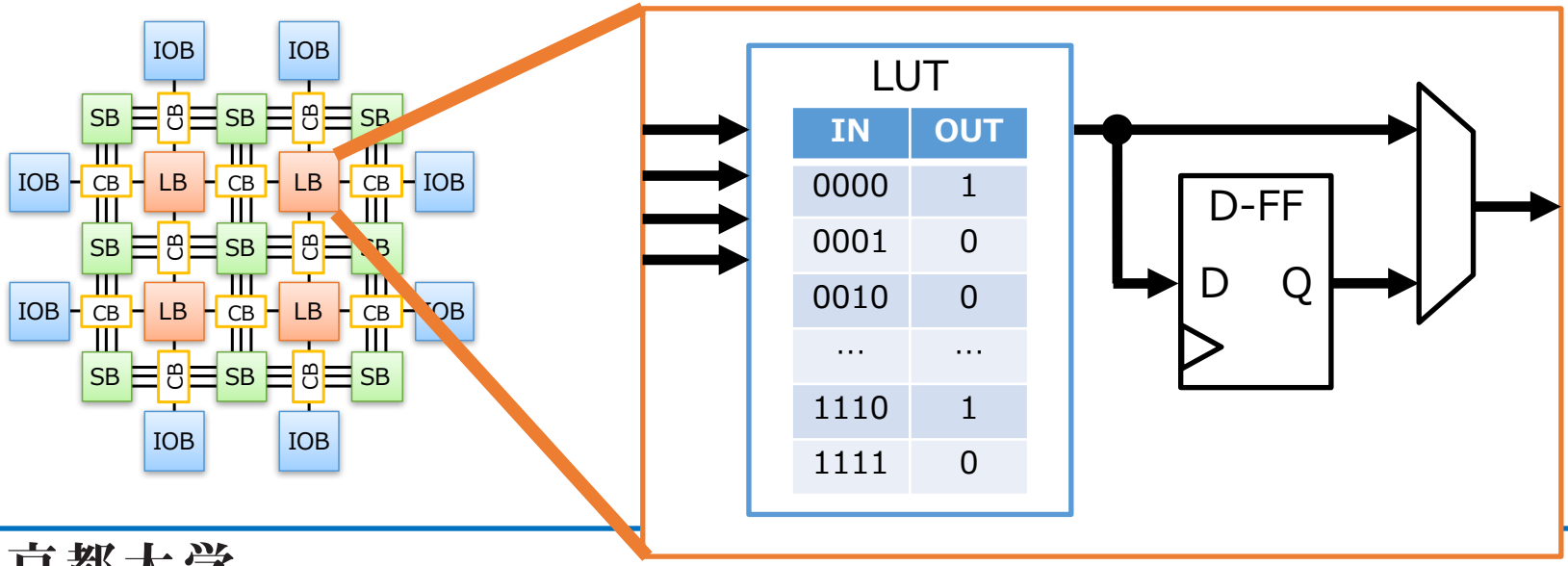


LB 論理ブロック　SB スイッチブロック

IOB 入出力ブロック　CB コネクションブロック

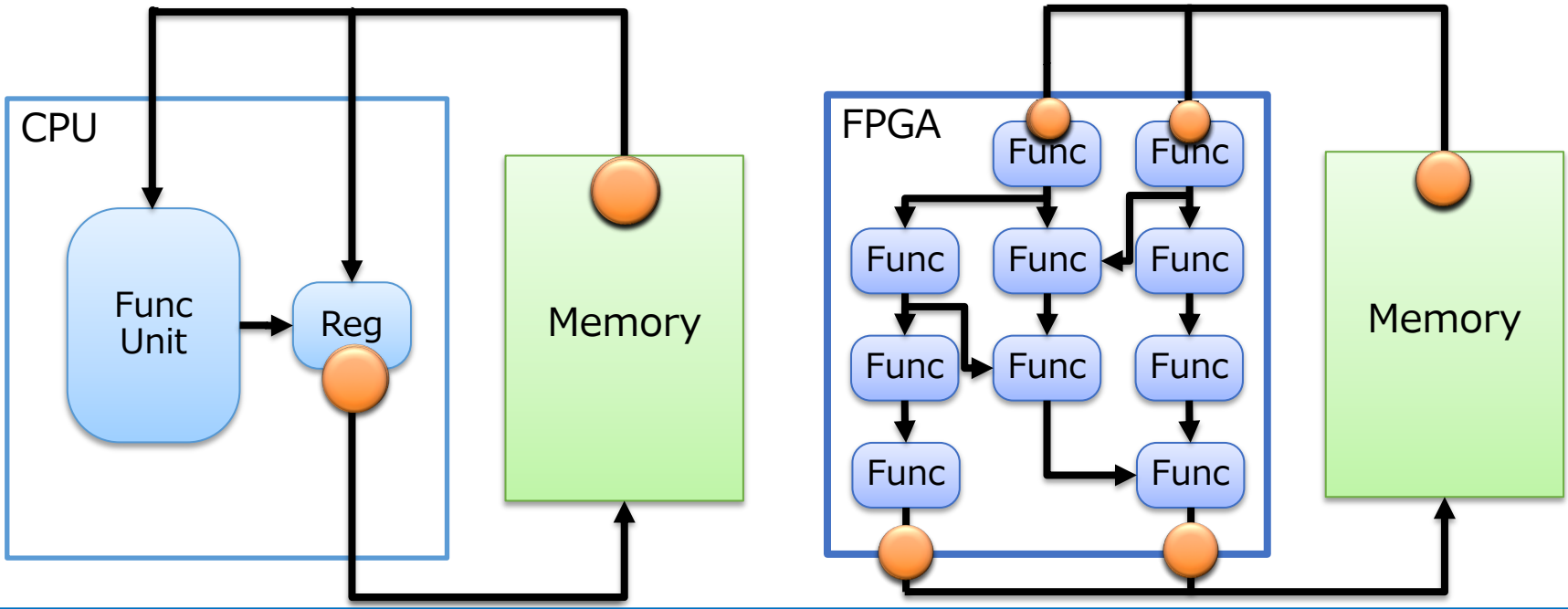
論理ブロックの論理表現

- プロダクトターム方式
  - 代表例：PLA (Programmable Logic Array)
  - ANDとORアレイのメッシュ接続による最小積和形表現
- ルックアップ・テーブル方式
  - フリップフロップと組み合わせて論理ブロックを実現



FPGAの強み

- 1つのLSIで様々なシステムを設計できる
  - プロセッサ処理より高性能かつ省電力を実現する
  - ASICよりも製造コストを抑えられる
- データストリーム処理の実現が可能になる
  - ノイマン・ボトルネックの解消に！



FPGAの用途

- ASICの設計・検証用LSIに活用
  - ASICの機能を製造前に検証できる
  - HWの完成前からSWの開発が始められる
- 最終製品に組み込むLSIに活用
  - 民生品での実用例はすでにキリが無いほど多い
  - 出荷後のデータ書き換え機能を持たせることも
  - 最近の実用例
    - ✓ 金融市場取引
    - ✓ ロボット
    - ✓ データセンター
    - ✓ 機械学習
    - ✓ オートモーティブ（ADAS）

金融市場取引

- 株式相場の値動きを解析して高速に売買取引
  - 高頻度取引（HFT）の自動売買では超高速応答が必須

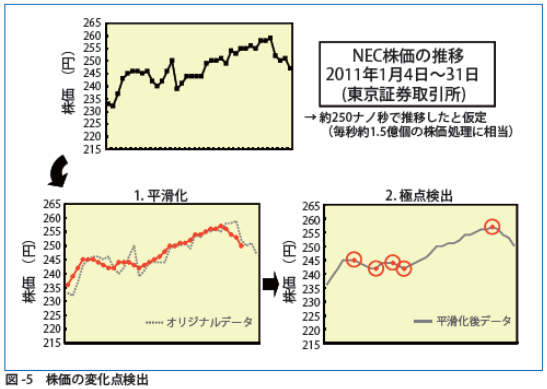


図 -5　株価の変化点検出

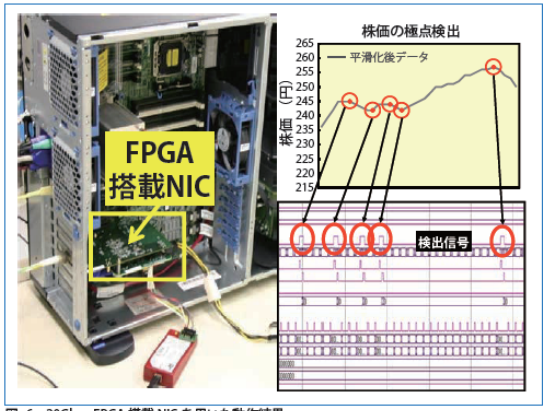
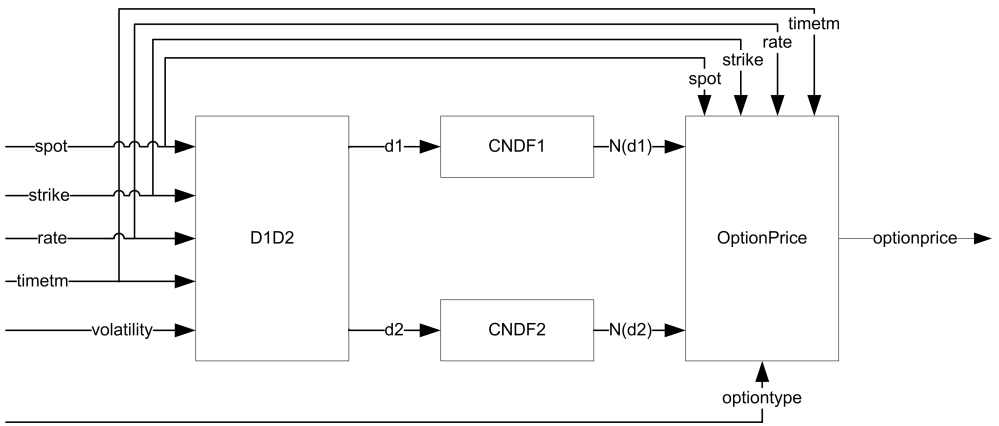


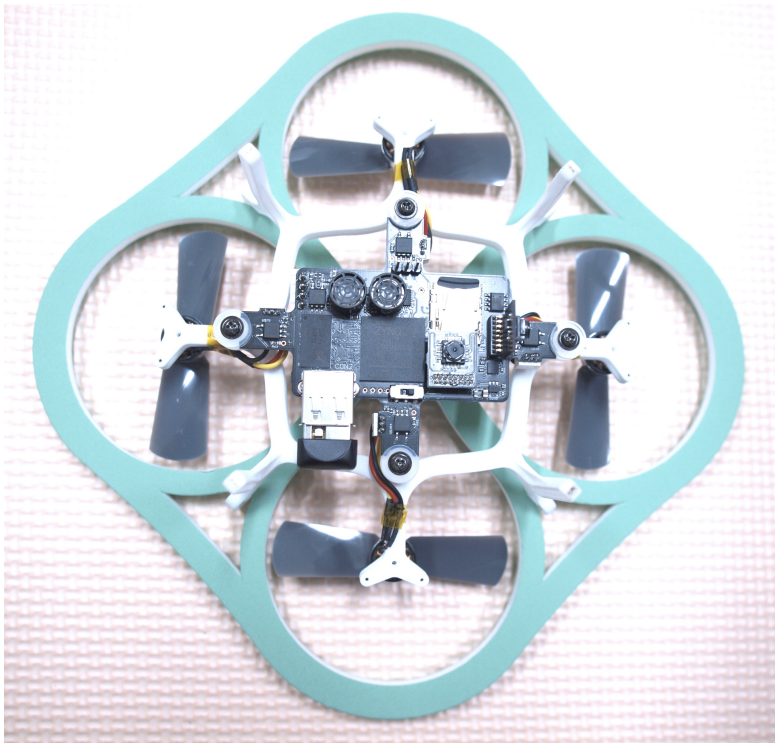
図 -6　20Gbps FPGA 搭載 NIC を用いた動作結果

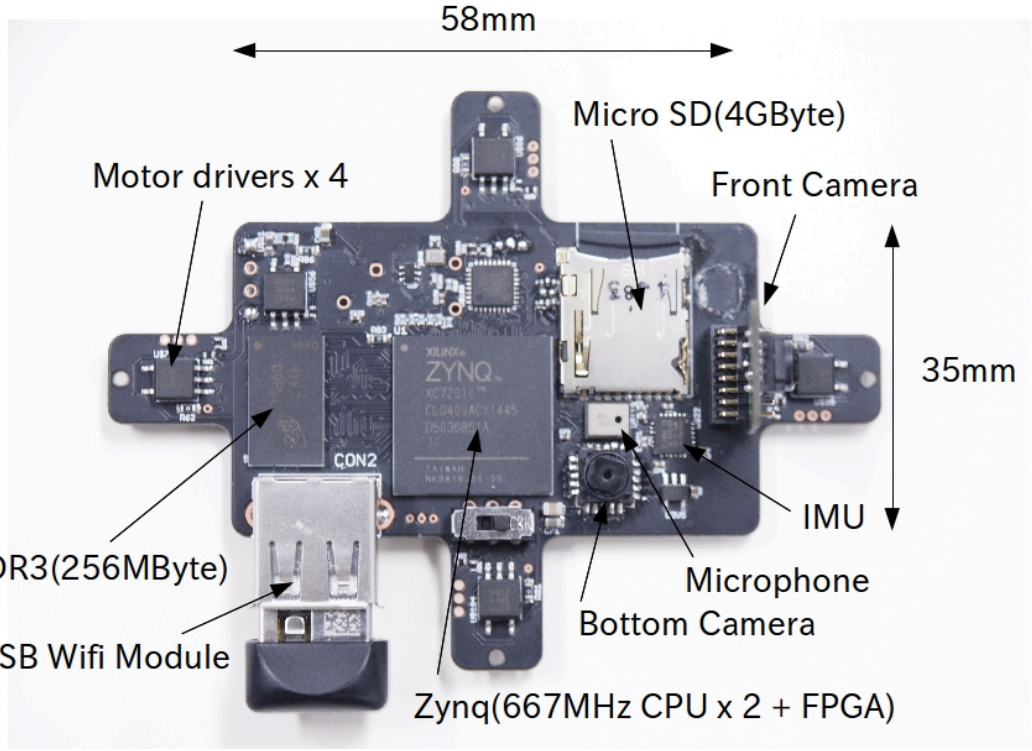


出典：井上, FPGAによる金融業務アクセラレーション–複合イベント処理を題材に–, 情報処理, 2012.
W. Wang, “Black-Scholes in hardware,” Overload Journal, 2012.

ロボット

- Phenox: FPGA搭載ドローン（Xilinx Zynq XC7Z010）
  - FPGA部分は主に信号処理を担っている





データセンター

- Microsoft Catapult

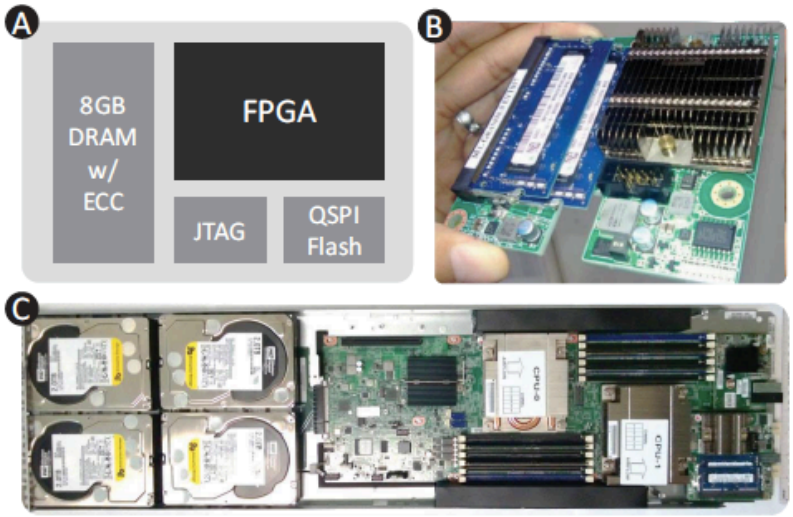
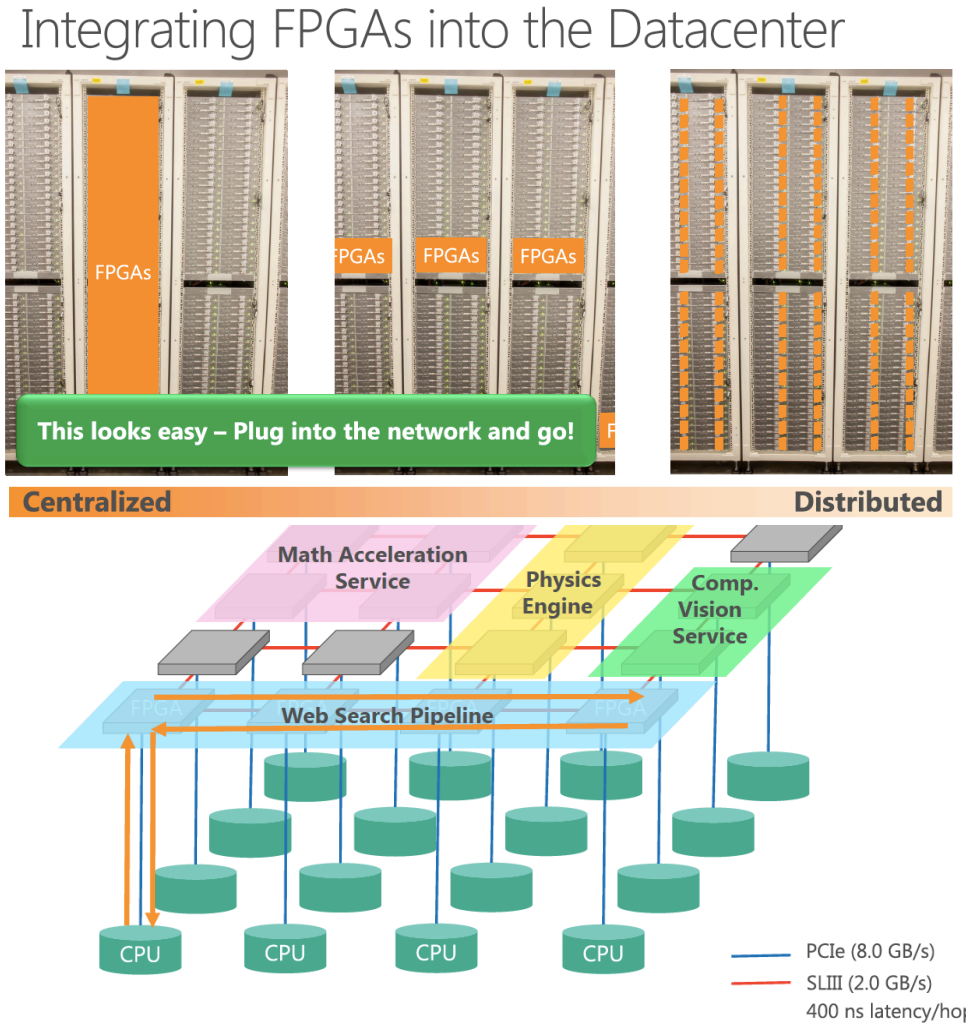


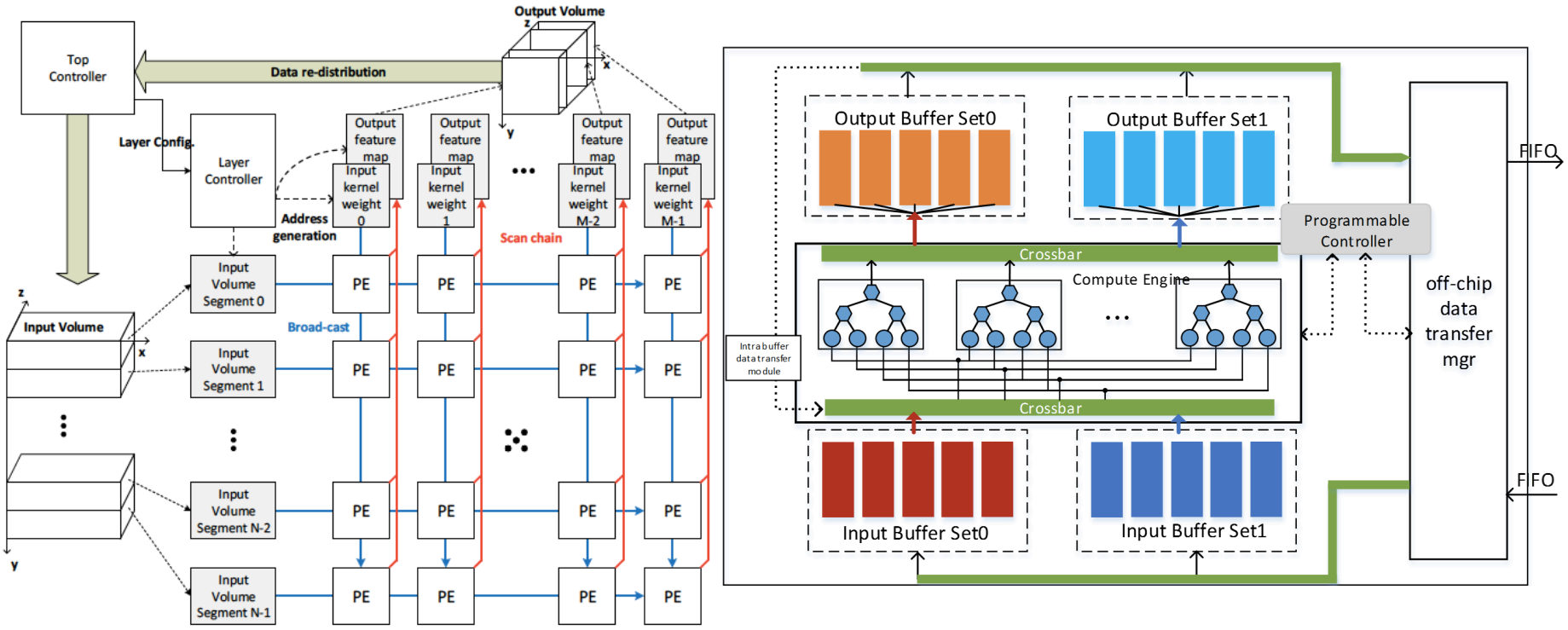
Figure 1: (a) A block diagram of the FPGA board. (b) A picture of the manufactured board. (c) A diagram of the 1 U, half-width server that hosts the FPGA board. The air flows from the left to the right, leaving the FPGA in the exhaust of both CPUs.



出典：A. Putnam, et al. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, ISCA, 2014.
A. Putnam, Large-Scale Reconfigurable Computing in a Microsoft Datacenter, HotChips26, 2014.

機械学習

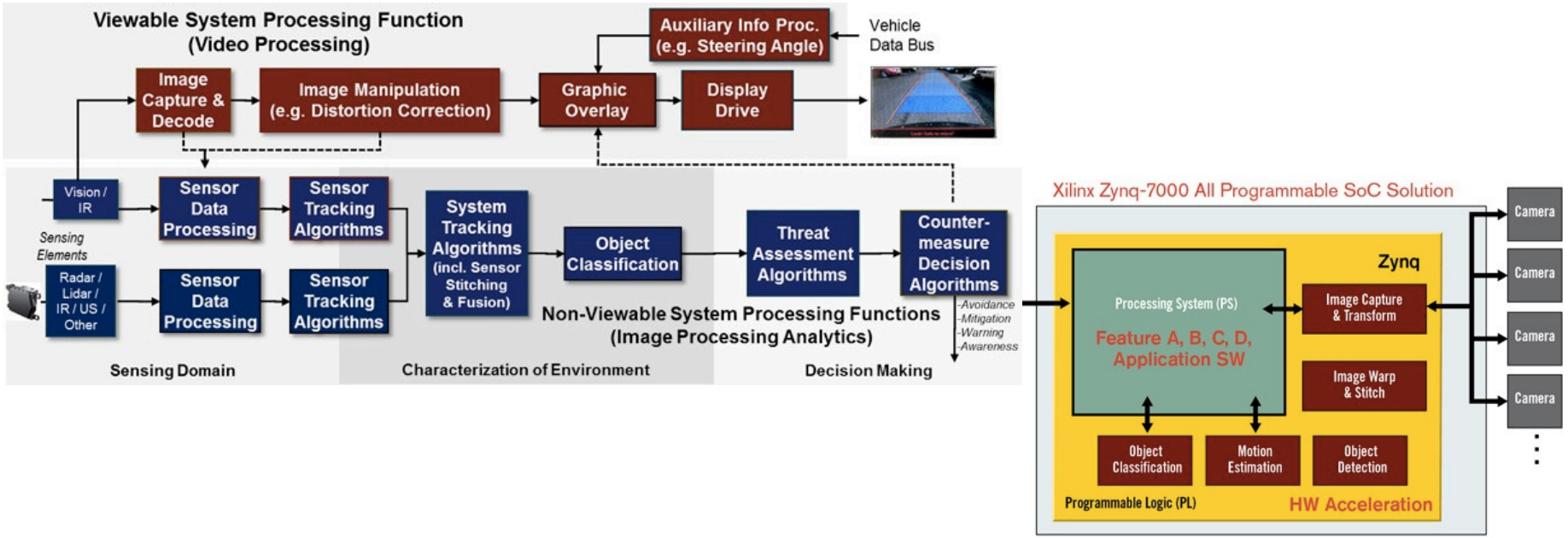
- CNN/DNNのアクセラレータ
  - パイプラインをニューロン・シナプスの値が流れる



出典：K. Ovtcharov, et al. Toward Accelerating Deep Learning at Scale Using Specialized Hardware in the Datacenter, HotChips27, 2015.
C. Zhang, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, FPGA 2014.

オートモーティブ

- ADASの実現支援
  - Xilinx：ADAS向け処理チップとして，
    2016年内で累計23社計85車種にFPGAが搭載予定
  - Altera：Audi社が自動運転システムzFASにCyclone Vを採用



出典：https://japan.xilinx.com/applications/automotive.html
http://techon.nikkeibp.co.jp/atcl/news/16/062702771/

最近の技術トレンド

- 回路規模・集積度の増大
  - 1つのLSIで高機能・多様なシステムを実現可能
  - マルチダイ化，3次元積層など今後もさらに増大
- プロセッサとFPGAの密結合化
  - 組込み：ハードマクロのArmプロセッサと混載（Zynq等）
  - 汎用・サーバ：Xeon等との統合（混載 or PCIe）
  - より高品質なHW/SW協調システムが実現可能になる
- 高位合成／協調設計
  - 抽象度の高い動作記述からRTLを生成する技術
  - 汎用プログラミング言語によって振る舞いを定義
    - ✓多くのツールはC/C++またはその拡張言語

FPGAの設計フロー

設計記述

↓

論理シミュレーション

↓

論理合成

↓

テクノロジ・マッピング

↓

配置配線

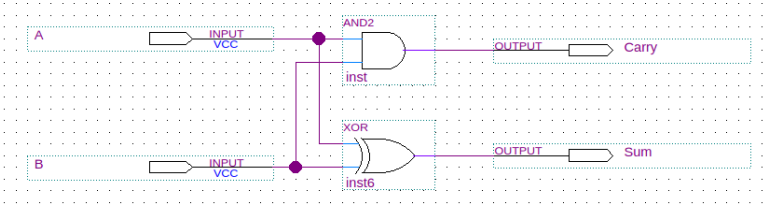
↓

ポストレイアウト
シミュレーション

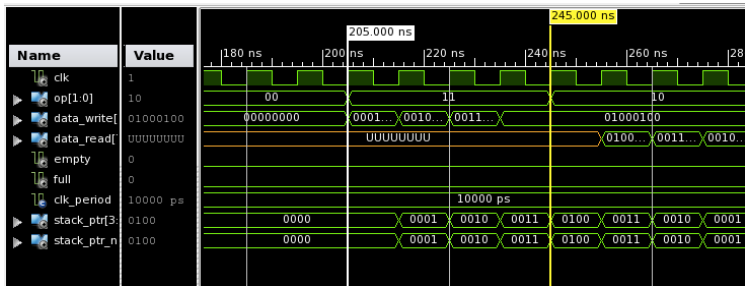
↓

ビットストリーム生成

ブロック図やIP(Intellectual Property)コア，ハードウェア記述言語（HDL，実験3にて）等による論理回路の設計



設計記述の論理動作が正しいかを検証



FPGAの設計フロー

設計記述

↓

論理シミュレーション

↓

論理合成

↓

テクノロジ・マッピング

↓

配置配線

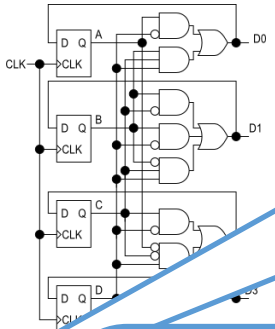
↓

ポストレイアウト
シミュレーション

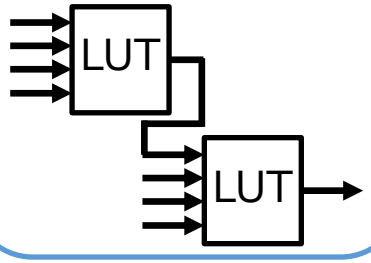
↓

ビットストリーム生成

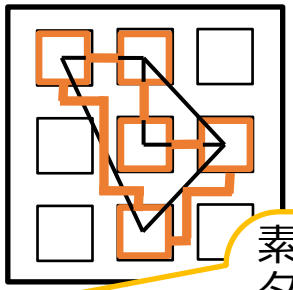
ゲートのネット
リストを合成



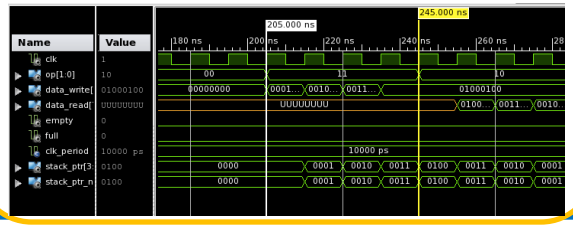
LUTのネット
リストを合成

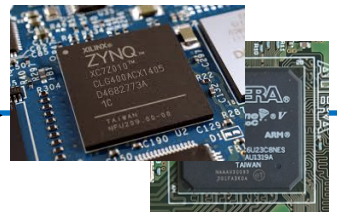


LBへのLUTの
割当てと接続を決定



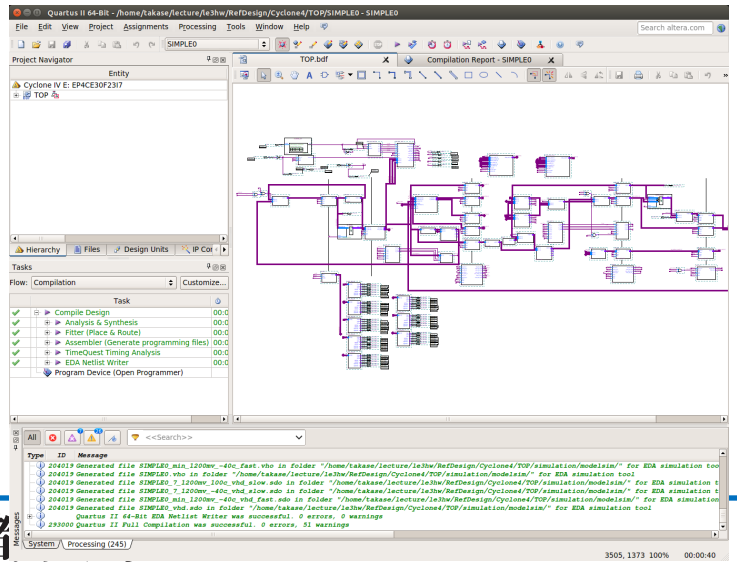
素子や配線の遅延も考慮した
タイミングが正しいかの検証

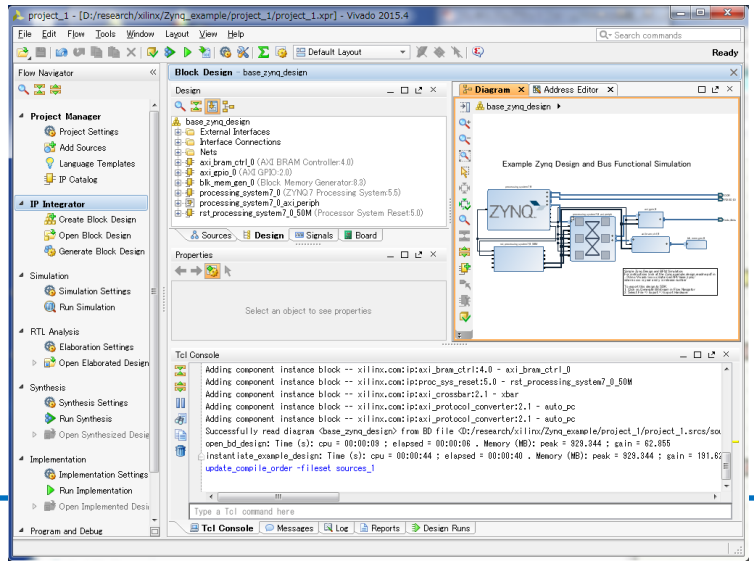




CADツール

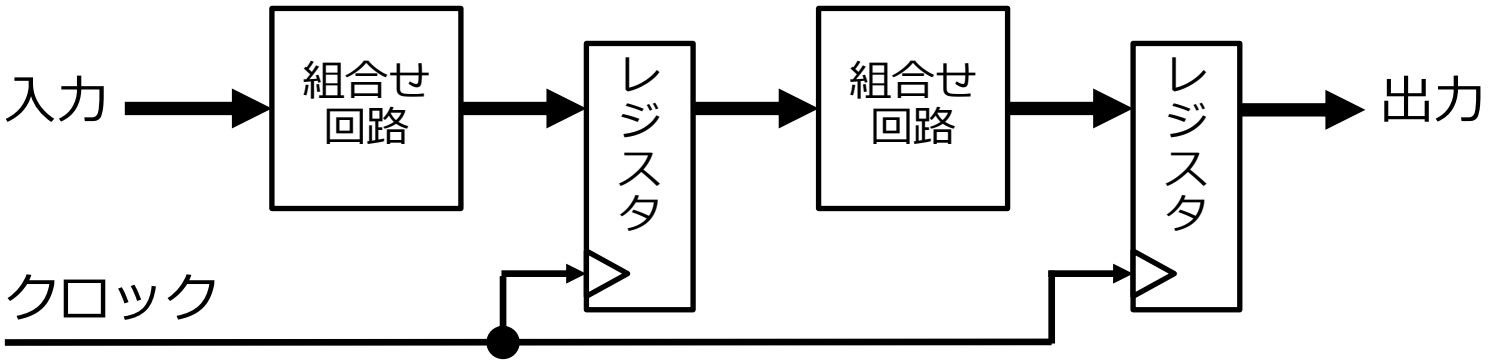
- Computer-Aided Design:
  論理回路を設計・合成するツール
  - 環境設定，回路の設計と記述，回路合成，
    ピンアサイン，シミュレーション，
    プログラミング（FPGAへの書き込み）までを行う
  - EDA(Electronic Design Automation)ツールとも呼ばれる





レジスタ転送レベル

- RTL: Register-Transfer Level
- デジタル回路を記述する手法の一種で，回路の動作を構成する表現
  - データ転送：レジスタ間のデジタル信号の流れ
  - それに対する論理演算の組み合わせ
    - ✓Timed設計：「なに」を「いつ」「どのように」を表現



実習課題(1)：加算回路の設計

– 半加算器を用いた全加算器
– 全加算器を用いた4ビット並列加算回路

実習課題(1)

- 加算回路の設計を通して，Quartus Primeの使用方法および論理回路の設計方法を学ぶ
  - A) 半加算器を用いた全加算器
  - B) 全加算器を用いた4ビット並列加算回路

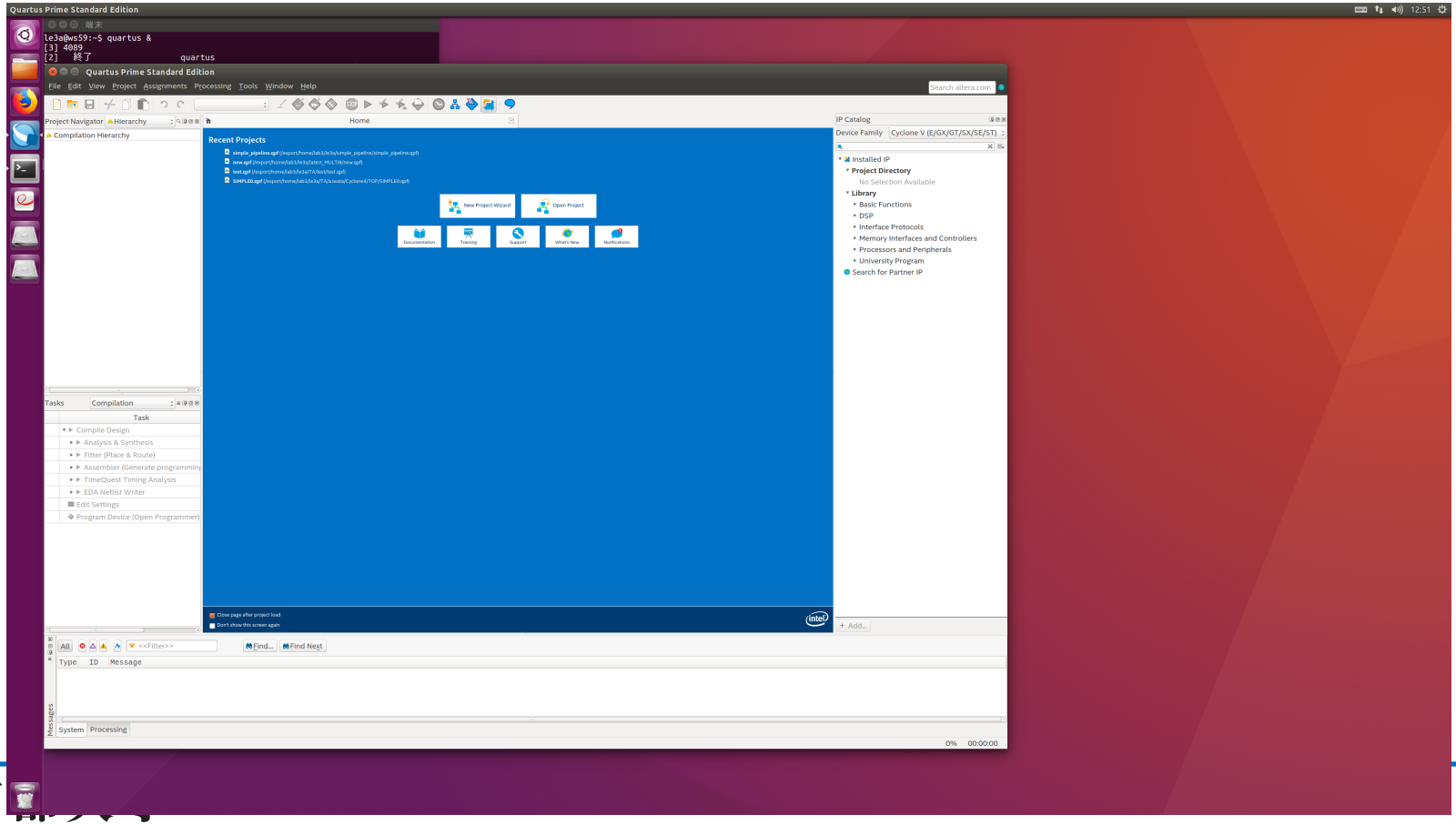
- 進め方
  - ハンズオン形式で行いますが，自力でできそうならどんどん実習課題・演習課題を進めていって構いません
  - 「実習課題で」詰まったら積極的にTAを捕まえましょう
  - 実習課題に関するレポートの作成・提出は不要です

実験環境：CADツール

- Intel Quartus Prime 17.1を用いる
  - 演習室PCには”Standard Edition”がインストール済み
  - Google先生に聞く時にはバージョン番号に注意！
    - ✓公式マニュアル・ドキュメントも別バージョンはアテにしない
- 各自のPCにもインストール可能
  - 機能制限有りの Lite Edition なら無償利用可能
    http://dl.altera.com/?edition=lite
    - ✓シミュレータはModelSim-Altera Starter Editionを入れる
    - ✓TIPSページの補足も参照のこと
  - Windows/Linuxのみ（Mac OS Xは不可）

Quartus Primeの起動

- ターミナル上で
  $ quartus &

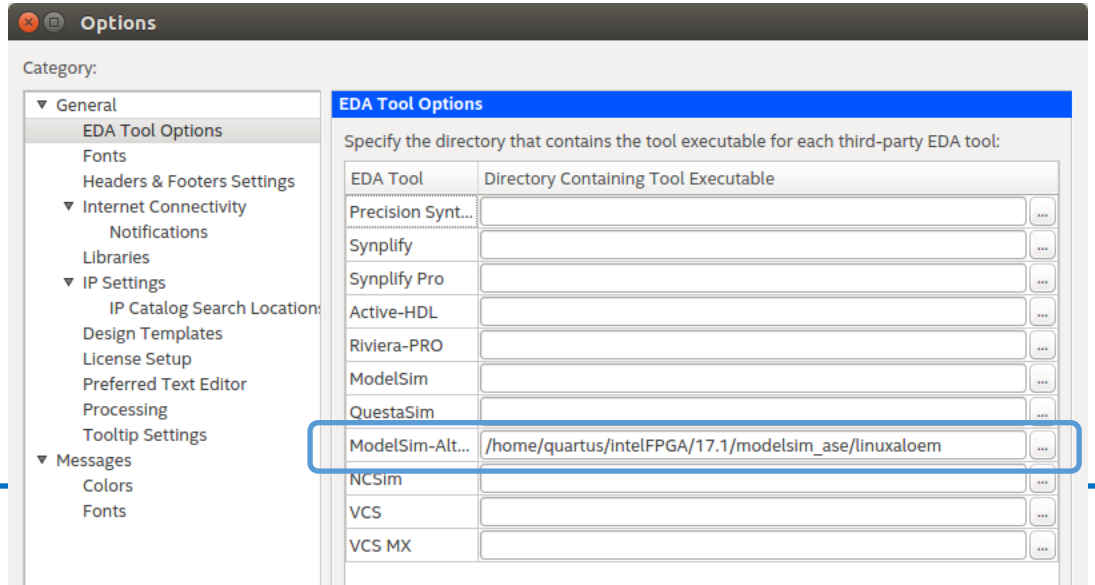


設定の確認

- ライセンス認証
  - Tools -> License Setup... で,
    “Not found” と言われていないことを確認
  - そう言われている場合や，起動時にLicense Setup Requiredとか出てくる場合は,
    “License file: “ に ”27000@is-fpg-00” を入力するか
    “LM_LICENSE_FILE variable:” にチェック・入力
- Webブラウザの設定
  - Tools -> Options... の Internet Connectivity で,
    “Web browser:” に好みのブラウザへのパスを設定
  - 何も好みがなければ ”/export/share/bin/firefox”
    - ✓エラー調査時に便利です

設定の確認

- Quartusからmodelsim-aseを呼び出すための設定をする
  - Tools -> Options... -> EDA Tool Options: ModelSim-Altera: /home/quartus/intelFPGA/17.1/modelsim_ase/linuxaloem
    - ✓‘s’ が抜けていることがあるので注意
    - ✓いったん空白にしてもよい



ツール利用と設計の流れ

a. 新規プロジェクトの作成と環境設定
b. 論理回路の設計
c. 回路図の作成
d. コンパイル
e. シミュレーションによる動作確認
f. トップレベル回路への入出力ピンの割当て
g. FPGAへの回路の書き込み

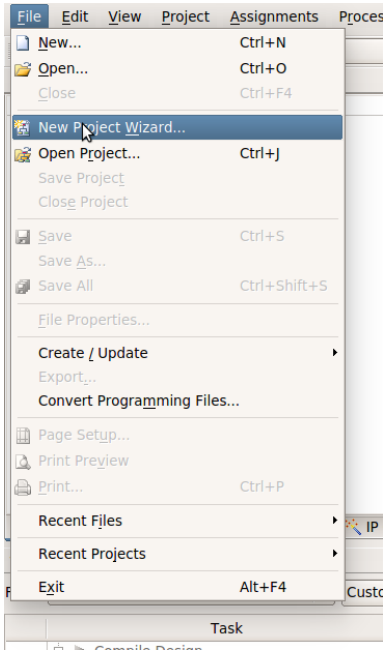
実験2ではf.以降は行いません
（実機で動かしたい余裕がある気になる方はTAまで）

A) 半加算器を用いた全加算器

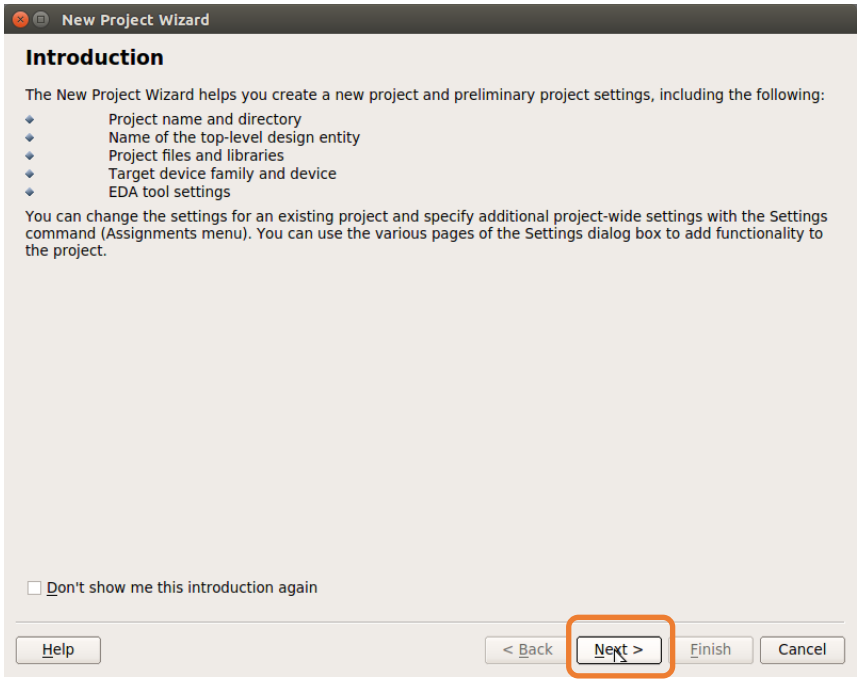
- 新規プロジェクトの作成と環境設定
- 論理回路の設計
- 回路図の作成とブロック化・流用
- コンパイル
- シミュレーションによる動作確認

a. 新規プロジェクトの作成と設定

- まずは半加算器を用いた1ビット全加算器を設計する
  - File -> New Project Wizard...

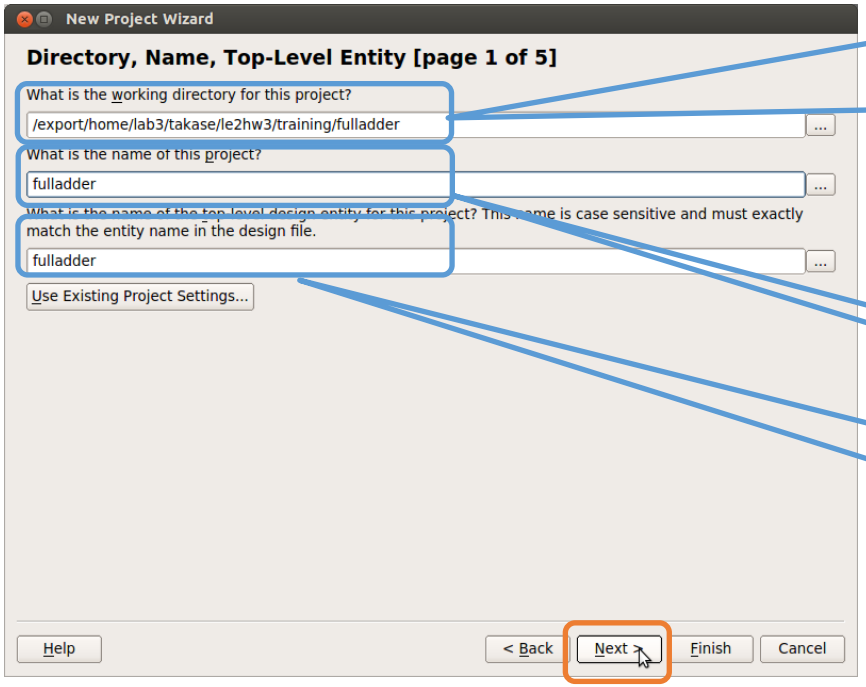


- Introduction “Next”



a. 新規プロジェクトの作成と設定

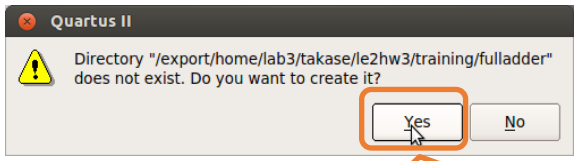
- Directory, Name, Top-Level Entity [page 1 of 6]



作業ディレクトリ：
ホームディレクトリそのものを指定せず，必ず専用のディレクトリを作ること．そうしないとツールが固まったり落ちたりする原因になる．
また，プロジェクト毎にディレクトリを作ったほうがよい（名前は自由）

プロジェクト名

トップレベルの回路名：
なにも気にしなければプロジェクト名と同一（自動入力される）



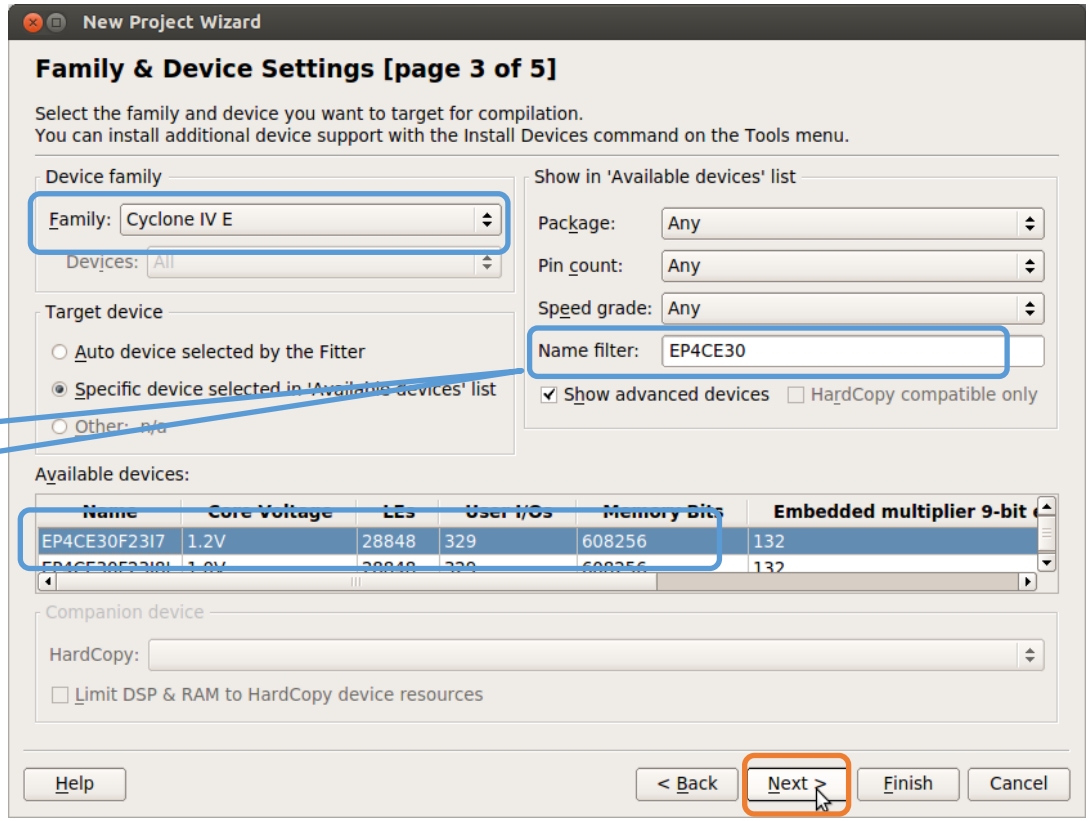
作業ディレクトリが存在しない場合は自動作成してくれる

a. 新規プロジェクトの作成と設定

- Project Type [page 2 of 6]
  - プロジェクトの種類を選択する
  - テンプレートからプロジェクトを作成することもできる
  - 今回は”Empty project”を選択
- Add Files [page 3 of 6]
  - 作成済みの回路をプロジェクトに追加する
  - 今回はまだ何も無いので空白のまま”Next”

a. 新規プロジェクトの作成と設定

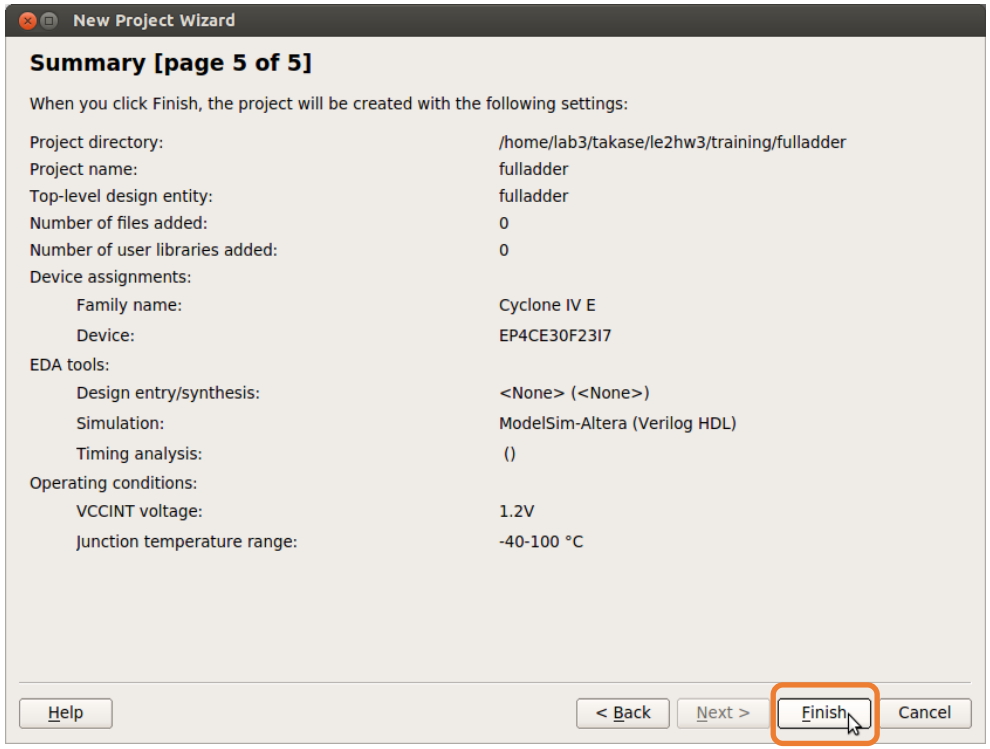
- Family & Device Settings [page 4 of 6]
  - Family:
    ”Cyclone IV E”
  - Available devices:
    ”EP4CE30F23I7”



選択が面倒な場合は
フィルタ検索も可能

a. 新規プロジェクトの作成と設定

- EDA Tool Settings [page 5 of 6]
  - 外部の設計ツールを使用／変更する場合に指定する
  - 学生実験では特に使用／変更しないのでこのまま”Next”
- Summary [page 6 of 6]
  - 設定を確認して“Finish”



b. 論理回路の設計

- 設計したい論理回路を考える
  - 外部仕様：入力・出力，機能
  - 内部仕様：回路構成
    - ✓ 真理値表，カルノー図，状態遷移図，，，　→ 論理式表現

  まずは設計してからツールを使い出すこと！

- 半加算器の外部仕様
  - 入力：1ビットの加算データA,B
  - 出力：1ビットの和Sum,
    1ビットの桁上げCarry

  論理式表現は分かりますよね，，，

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

b. 論理回路の設計

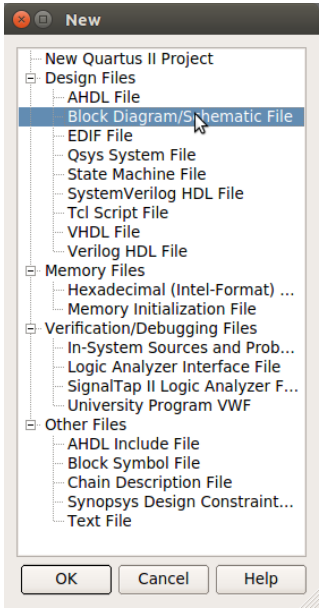
- 全加算器の外部仕様
  - 入力：1ビットの加算データA,B,
    1ビットの下位からの桁上げCin
  - 出力：1ビットの和Sum,
    1ビットの桁上げCout
  - 半加算器を利用すること

論理式表現は分かりますよね，，，

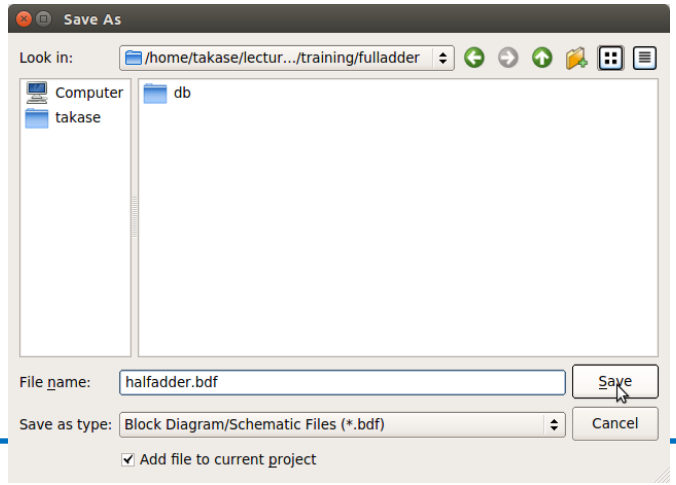
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

c. 回路図の作成

- 回路図エディタを使用する
  - File -> New -> Block Diagram/Schematic File
  - 実験2ではブロック図のみを入力とした設計とするハードウェア記述言語による設計は認めません！
    - ✓ 今年度は素子を組み合わせて回路を構成することを楽しんでください



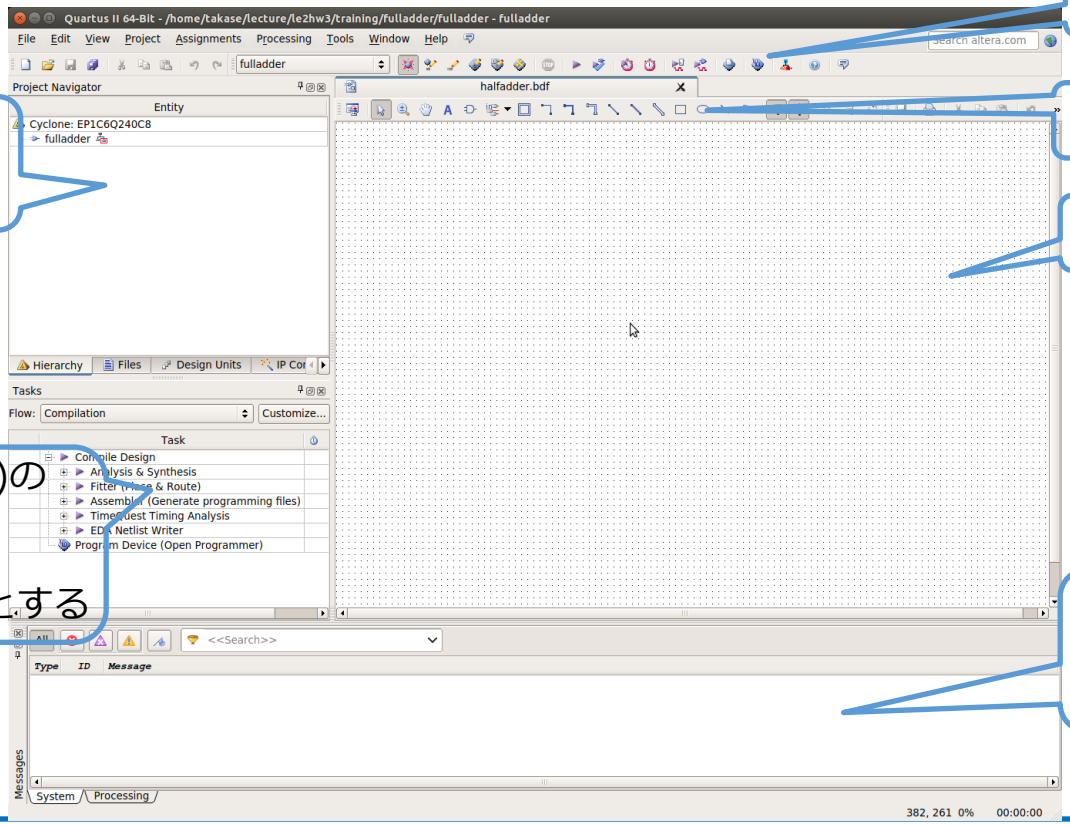
- まずは(泣く前に) 回路図ファイルを保存
  - File -> Save As...
  - “halfadder.bdf”



- Tasks窓のFlow:を“Compilation”にする

c. 回路図の作成

- (初期)ウインドウの構成
  - 使いやすいようにレイアウトやサイズ等を適宜調整してよい
    View -> Utility Windows など



ツール操作アイコン

回路図操作アイコン

回路図等のファイルの
一覧や呼び出し関係

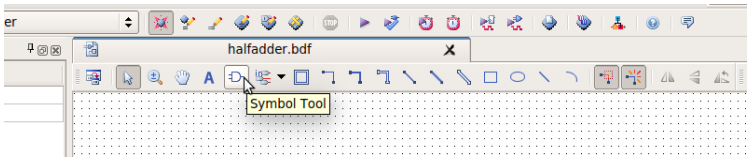
回路図面

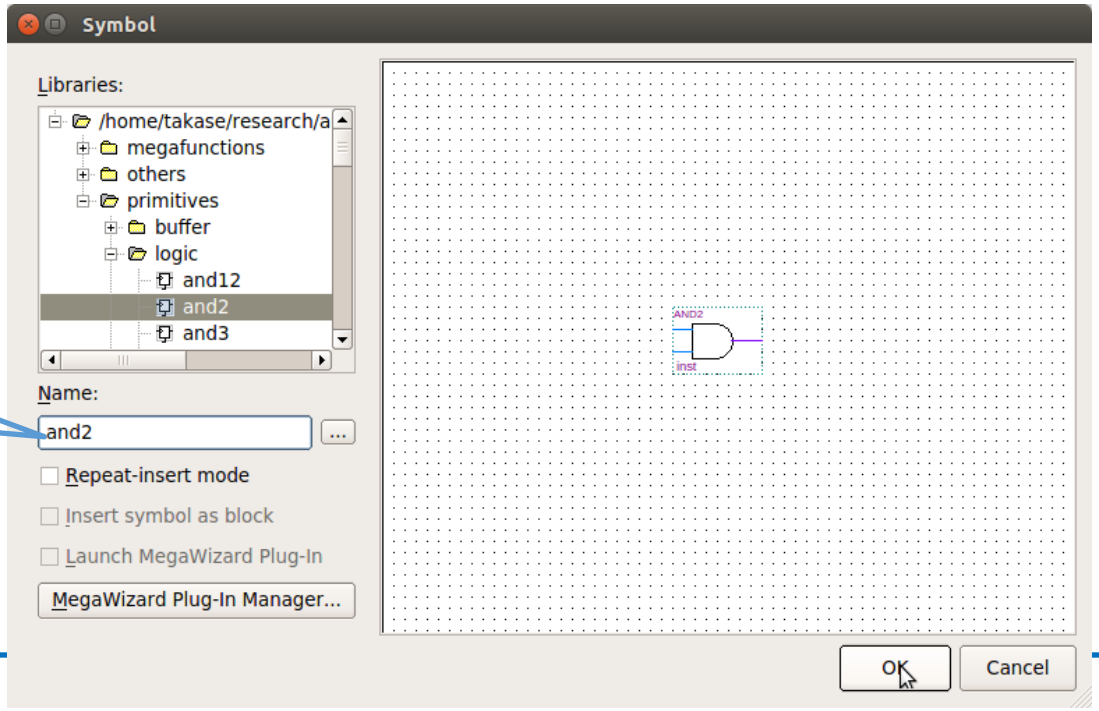
タスク(コンパイル等)の
進行状況の表示や
レポートの選択
Flow: “Comppilation”とする

メッセージウインドウ
処理メッセージや
エラー報告など

c. 回路図の作成

- 論理素子／回路ブロックの配置
  - 回路図面上のダブルクリック，または，回路図操作アイコンのSymbol Tool
  - 基本的に”Primitives”の中の素子だけを使いましょう
    - ✓ Megafunction等を使いたい場合は妥当な説明を示してください

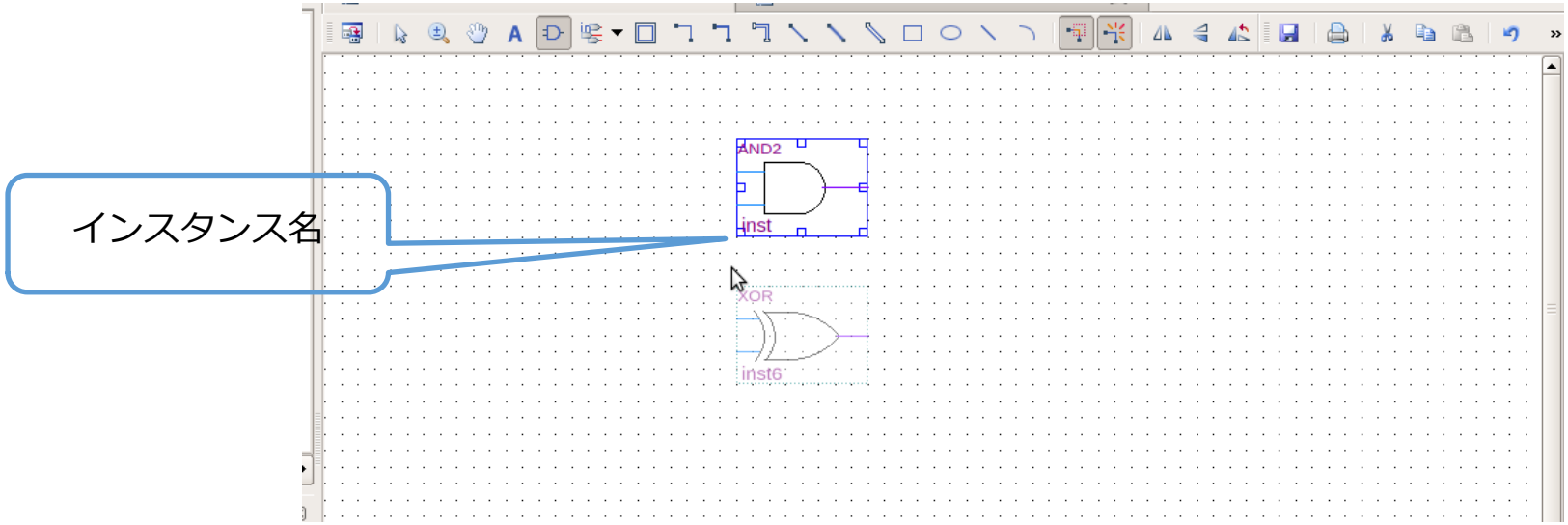




選択が面倒な場合は
フィルタ検索も可能

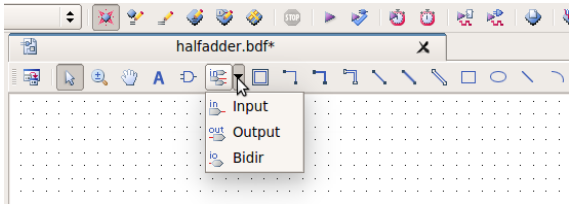
c. 回路図の作成

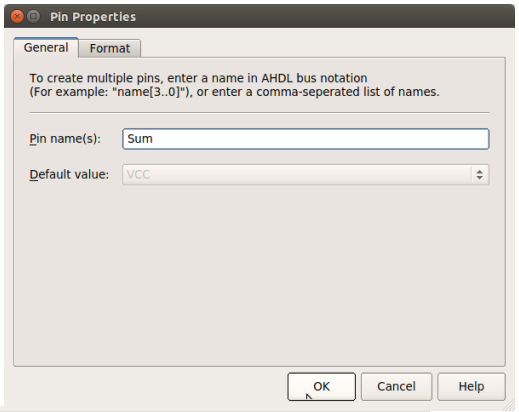
- 論理素子／回路ブロックの配置
  - 必要な論理素子を配置していく
  - 左上は素子名
  - 左下の“instX”の部分はインスタンス名
    - ✓ 名前を変更することができる（自動的に適当な番号を振ってくれる）
    - ✓ 要所要所に分かりやすい名前を付けておいたほうがデバッグが楽

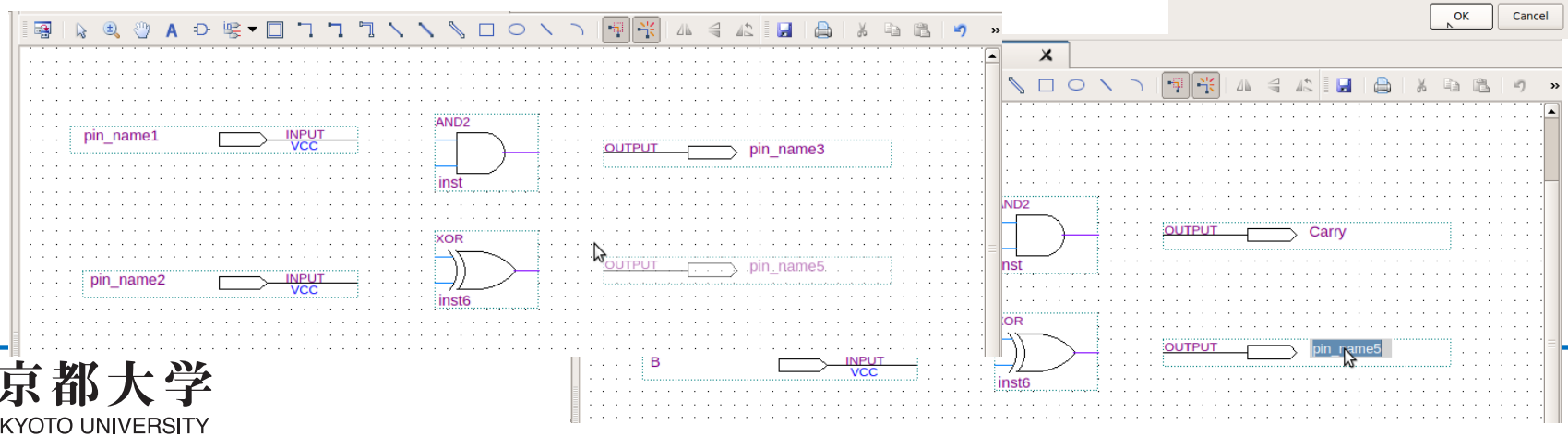


c. 回路図の作成

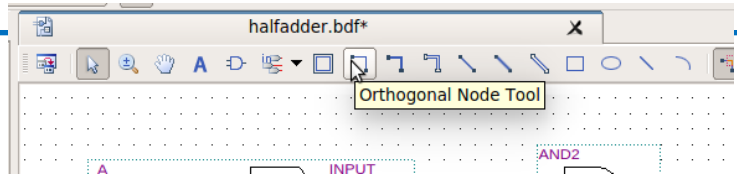
- 入出力端子の配置
  - Symbolウインドウの Primitives -> pin，または，回路図操作アイコンのPin Tool
  - 必要な入出力端子を配置していく
  - 左右の“pin_nameX”の部分はインスタンス名
    - ✓ こちらは必ず意味のある名前を付けるべき
    - ✓ ダブルクリックまたは右クリック “Properties” でも名前を変更できる
  - 入力端子右下のVCCは初期値（GNDに変更可）



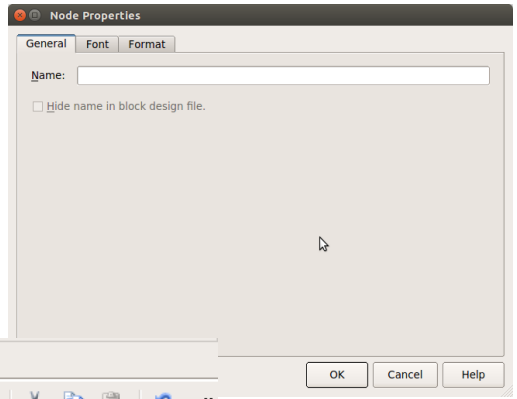


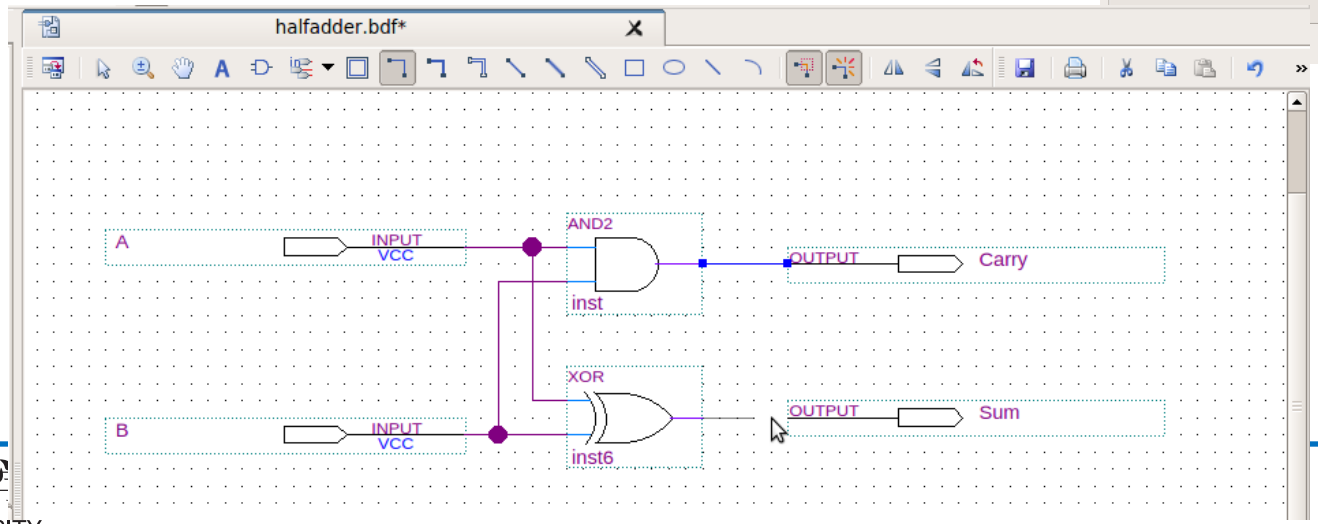


c. 回路図の作成



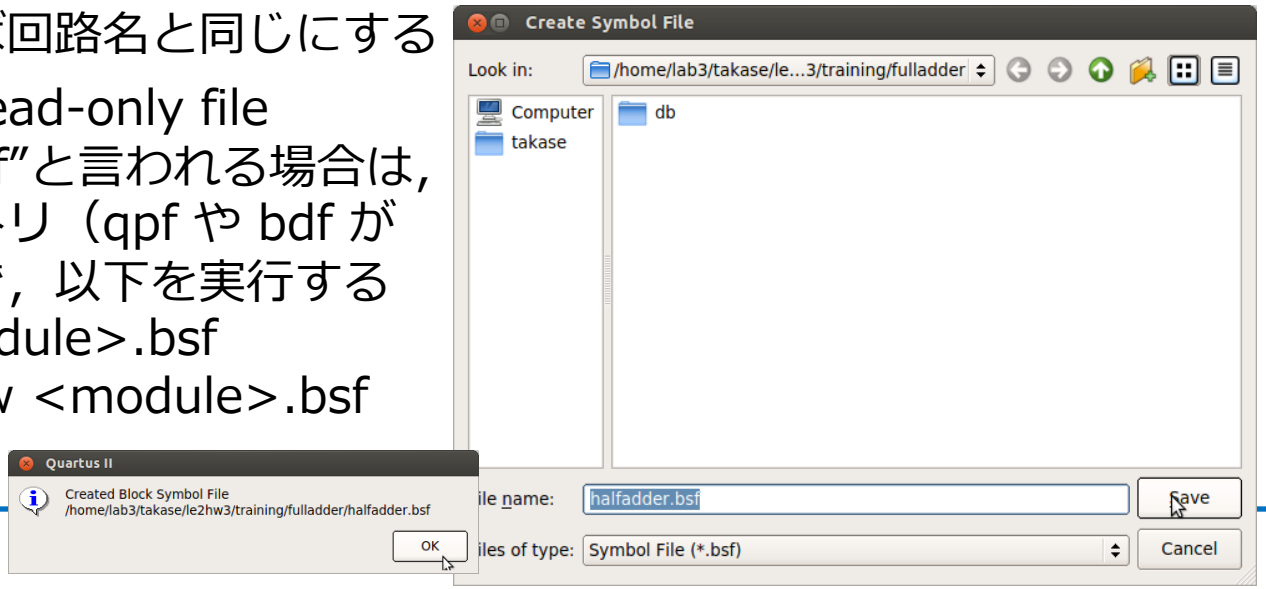
- 論理素子・端子などの間の配線
  - 回路図操作アイコンのOrthogonal Node Tool
    - ✓ ここでは1bit幅の配線のみでOK
    - ✓ 1回の配線につき1回までしか曲げられない
      もっと曲げたい場合は2回以上に分けて配線する
  - 必要に応じて配線にも名前を付けてよい
    - ✓ 右クリック “Properties”
    - ✓ 必ずしも端子名と一致させる必要は無い





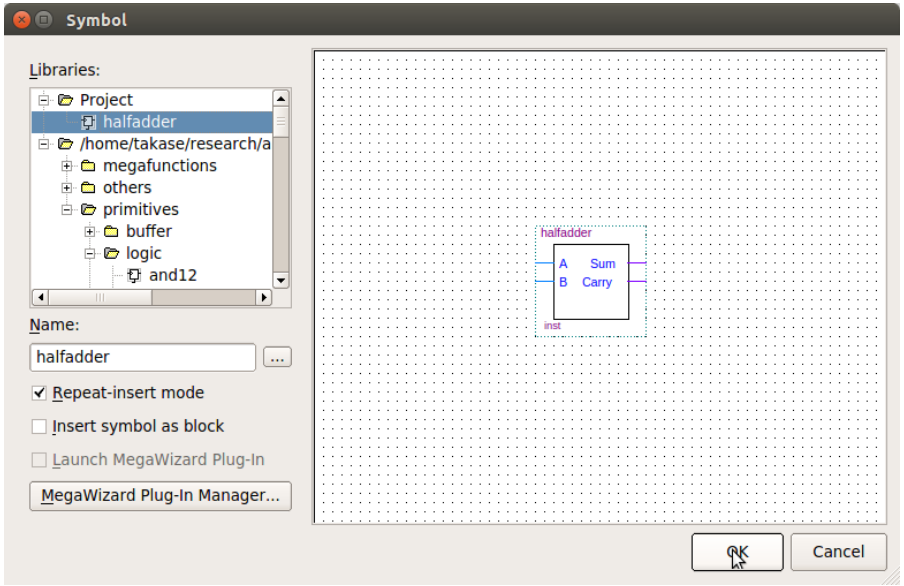
c. 回路図の作成

- 部品化（シンボル化）
  - 他の回路図ファイルから参照可能なシンボルファイルを生成する
    - ✓ プログラミングにおけるライブラリ化に相当するといってよい
    - ✓ 回路図ファイルを分割・階層化できる
  - File -> Create / Update ->
    Create Symbol Files for Current Files
  - “Symbol File (*.bsf)”として保存する
    - ✓ 理由が無ければ回路名と同じにする
    - ✓ “Can‘t open read-only file <module>.bsf”と言われる場合は，設計ディレクトリ（qpf や bdf があるところ）で，以下を実行する
      $ touch <module>.bsf
      $ chmod o+w <module>.bsf



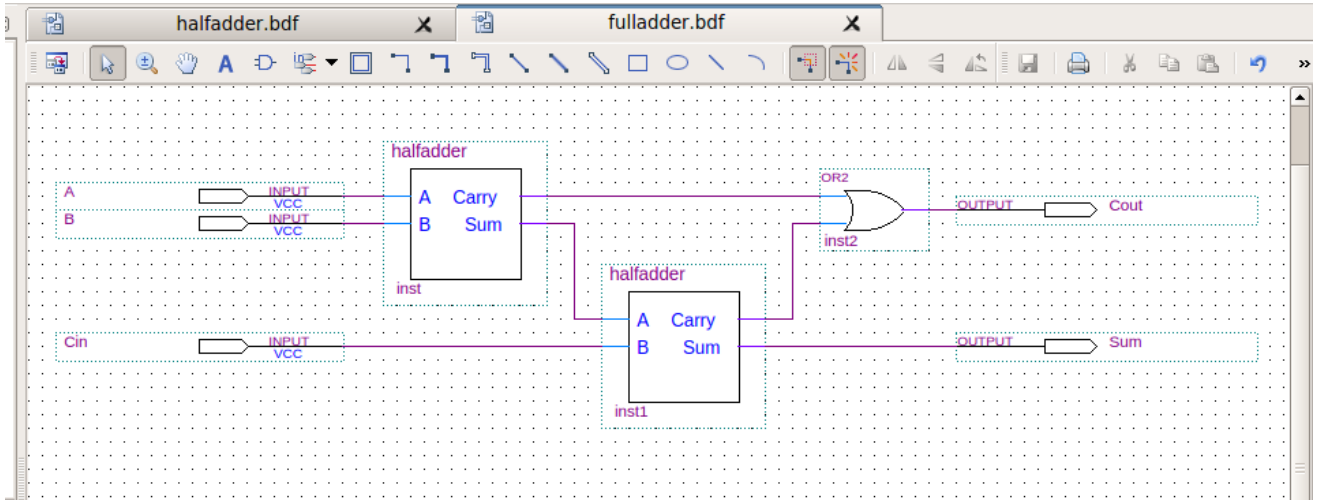
c. 回路図の作成

- 他の部品（シンボル）を利用した回路図の作成
  - 新しくブロック図ファイル（”fulladder.bdf”）を作成する
  - Symbol Toolの”Project”以下に先ほど生成した”halfadder”が追加されている
  - 元の回路図を変更した場合は，.bsf を保存して，シンボルの右クリック “Update Symbol or Block...” すると変更が反映される



c. 回路図の作成

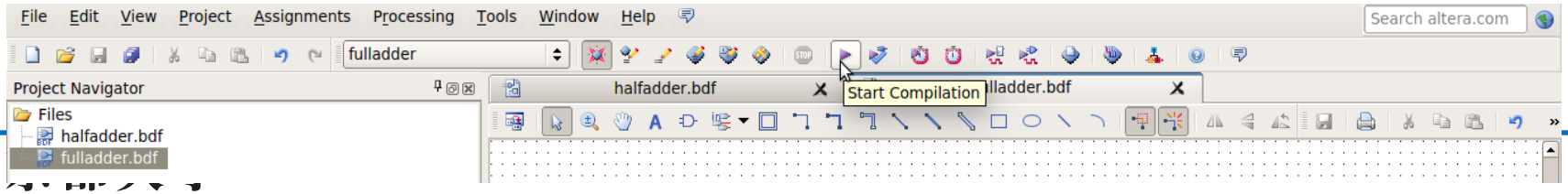
- 全加算器”fulladder.bdf”の作成
  - 今までの手順と同様



d. コンパイル

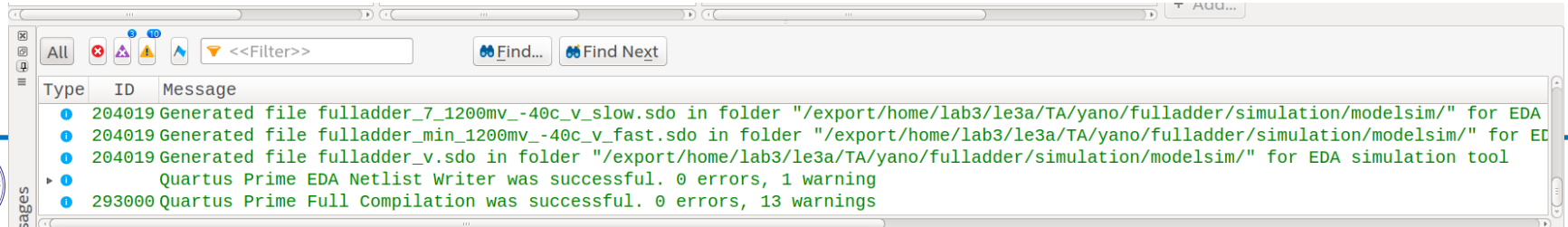
- FPGA内の論理ブロックの構成情報に変換する（Cのコンパイルに相当）
  - 回路図をレジスタ転送レベルに変換する
  - 論理合成・テクノロジマッピング・配置配線を行う
    - ✓ RTLで記述された回路をFPGA上の論理ブロックに割り付け，論理ブロックの配置や配線を行う
  - ビットストリームを生成する
    - ✓ 配置／配線の情報をバイナリ記述の構成情報ファイルに変換する

- コンパイル手順
  - Processing -> Start Compilation または ツール操作アイコンのStart Compilation または Ctrl+L
    - ✓ Start Analysis & Synthesis で入力チェックのみも可能



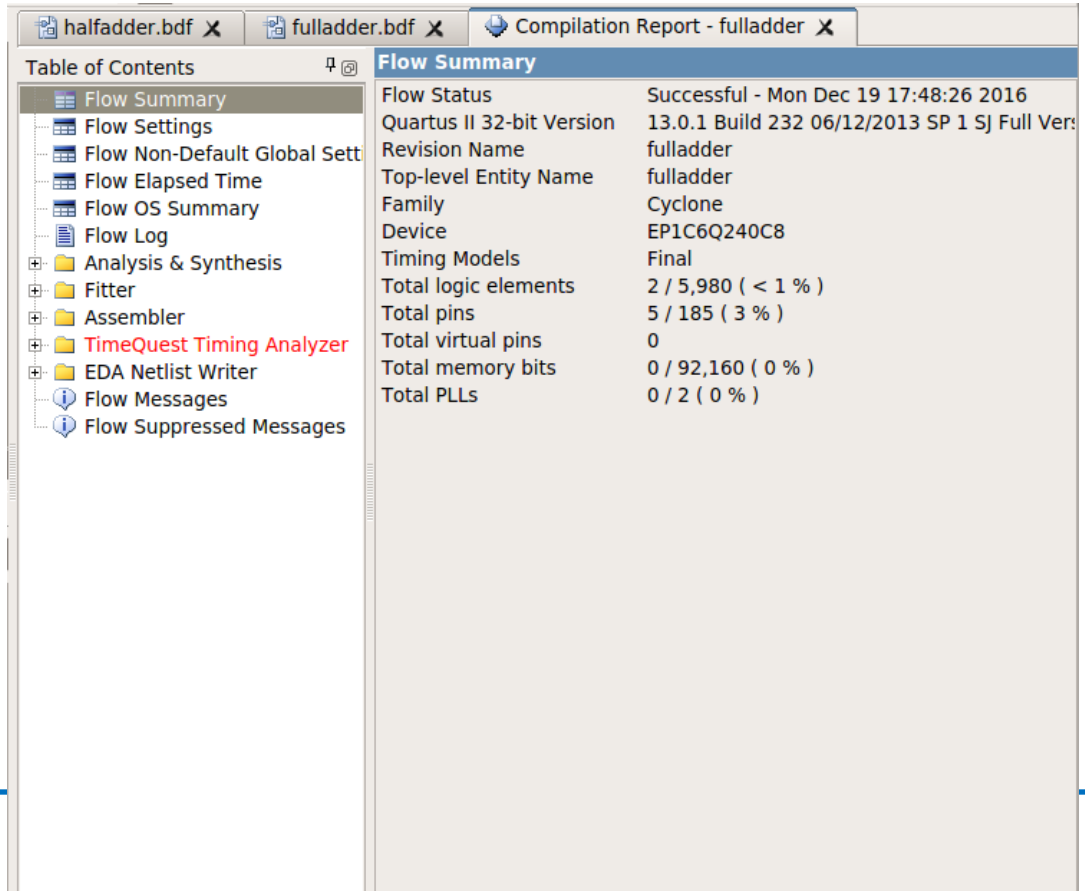
d. コンパイル

- メッセージウインドウに“Quartus Prime Full Compilation was successful”が表示されたら正常にコンパイル成功
  - タスク窓が全て緑字✓になっているはず
  - 黄字？はそのフローで再コンパイルが必要なことを表す
  - 赤字×はそのフローでエラーが発生していることを表す
- 正常終了しなかったら，下部のメッセージウインドウ または “Compilation Report -> Flow Messages” などを確認し，メッセージをもとにデバッグを行う
  - メッセージの意味は，右クリック -> Help でWeb検索可能
  - Critical Waning / Warning も要チェック
    - ✓ CADツールでは警告がシビアに出力されがちだが，警告の意味を理解した上で無視すること（設計ミスとかに気づくことも多い）



d. コンパイル

- Compilation Report で合成結果を確認する
  - Processing -> Compilation Report または Ctrl+R
  - 各項目の詳細は各自で調査すること

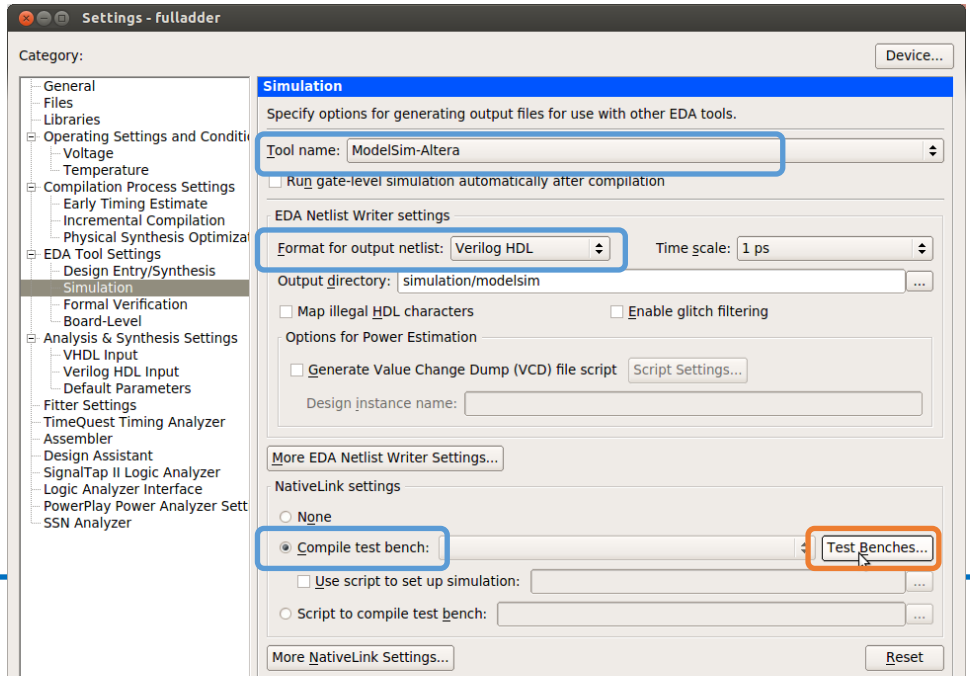


d. コンパイル

- 回路サイズ：Fitter -> Summary
  - FPGA上の各ブロックの使用量
  - 100%を超える項目があると該当のFPGAデバイスに収まらない
- 動作速度・遅延：TimeQuest Timing Analyzer
  - Datasheet Reportにそれぞれの入力端子から出力端子への伝搬遅延時間（単位はns）が表示される
  - 順序回路では，入力クロックを指定すれば，動作周波数なども表示できるようになる
    - ✓今は組合せ回路なので詳細は後ほど
  - 赤字の項目がある場合はタイミング制約を満たしていない
    - ✓ Unconstrained Pathsは実験2では無視してよい
      (FPGAデバイスとの入出力ピンを接続しないといけない，実験3にて)

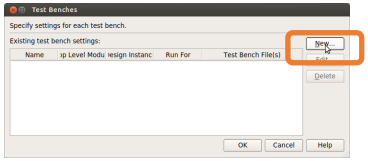
e. シミュレーション

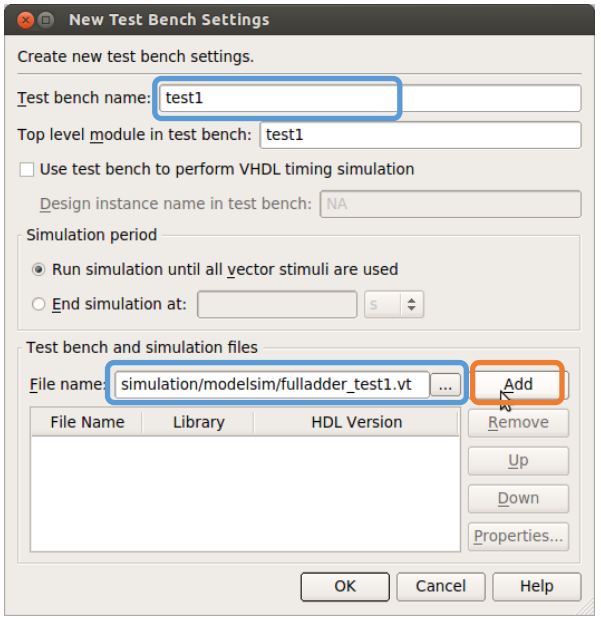
- 設計した論理回路が正しく動作するかを波形エディタで確認する
  - ModelSim-Altera Starter Edition (modelsim-ase) を用いる
    - ✓ modelsim-ae もあるので注意（ライセンスが切れていて動かない）
- まずはQuartusからmodelsim-aseを呼び出すための設定をする
  - Assignment -> Settings -> EDA Tool Settings -> Simulation
    - ✓ Tool name: ModelSim-Altera
    - ✓ Format for output netlist: Verilog HDL
      - VHDLでも良いが，後で必要な設定が少し増えるのでサポート外
    - ✓ NativeLink settings: Compile test bench
    - ✓ Test Benches... を選択

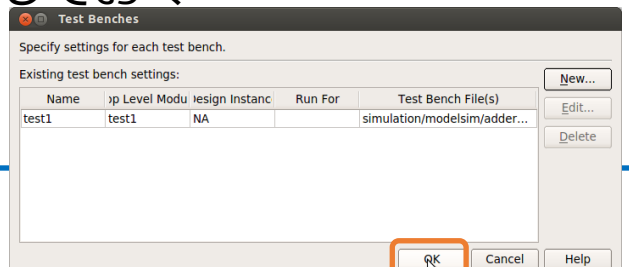


e. シミュレーション

- まずはQuartusからmodelsim-aseを呼び出すための設定をする
  - Test Benches
    - ✓ New... を選択
  - New Test Bench Settings
    - ✓ Test bench name: 名前を付けて作成
    - ✓ Top level module in test bench: を実際のテストベンチモジュール名 (Template Writer を使うなら <module>_vlg_tst) にしておくと，ModelSim起動の一発目で Error loading design が出なくなる
    - ✓ Test bench and simulation files: simulation/modelsim/<module>_test1.vt
      - ターミナルから空ファイルを作成しておく
    - ✓ Add を選択
  - Test Benchesに戻ってOKを選択







e. シミュレーション

- テストベンチファイルを作成する
  - Processing -> Start -> Start Test Bench Template Writer
  - ./simulation/modelsim/<module>.vt が作成される
  - ./simulation/modelsim/<module>_test1.vt などにコピー
    $ cp ./simulation/modelsim/fulladder.vt ¥
    ./simulation/modelsim/fulladder_test1.vt ¥
  - 所望の入力波形が得られるように <module>_test1.vt を編集
    - ✓ 時間単位を変更する（デフォルトは ps なので速すぎる）
      `timescale 1 ns / 1ps
    - ✓ 以下の記述の直下に初期設定を記述する
      // code that executes only once
      // insert code here --> begin
    - ✓ 以下の記述の直下に所望の値（波形）を記述する
      // code that executes for every event on sensitivity list
      // insert code here --> begin

e. シミュレーション

- テストベンチファイルの記法例
  - 代入：

```
// 1ビット
A <= 0;
B <= B + 1;
// ビット演算
Cin <= ~Cin;
// 4ビットの2進数
As <= 4'b10_01;
// 4ビットの2進数
As <= 4'b10_01;
```

    - ✓ (遅延が無ければ)並列に代入される
  - 遅延：

```
#1000
```

  - 繰り返し：

```
always begin
    #100
    clock <= ~clock;
end
```

    - ✓ クロック波形の生成に便利
    - ✓ initial begin - end や always begin - end の外側，module - endmodule の内側に記入すること
    - ✓ alwaysブロック同士は並列に実行される
    - ✓ @()で条件の記述も可能

e. シミュレーション

- テストベンチファイルの記述例
  - 時間単位も変更済み
    `timescale 1 ns / 1ps
  - 安直な真理値表の網羅なので，工夫してみてください
  - 単位や遅延はよく考える
    - ✓ あまりに小さいと？？

fulladder_test1.vt (/home/lab3/takase/le2hw3/training/fulla

開く　保存　元に戻す

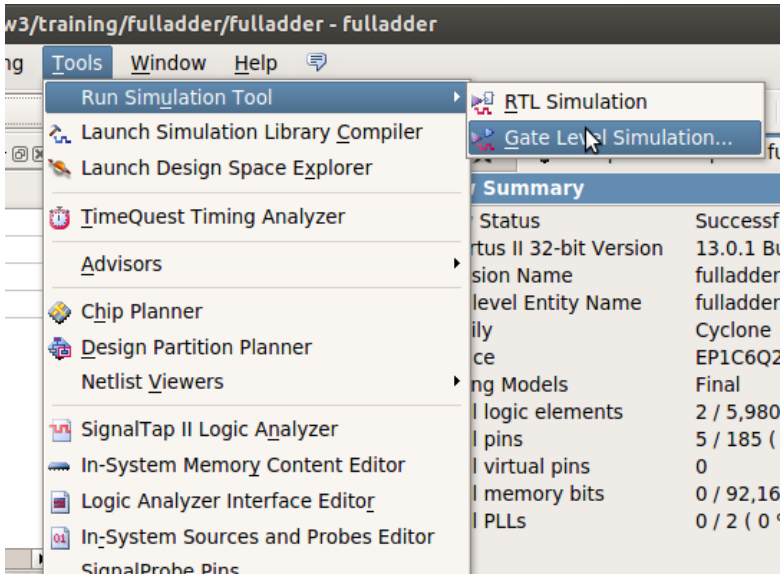
fulladder_test1.vt

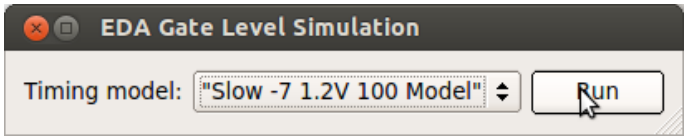
```
);
initial
begin
// code that executes only once
// insert code here --> begin
A <= 0;
B <= 0;
Cin <= 0;

// --> end
$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
#100
Cin <= 1;
#100
B <= 1;
Cin <= 0;
#100
Cin <= 1;
#100
A <= 1;
B <= 0;
Cin <= 0;
#100
Cin <= 1;
#100
B <= 1;
Cin <= 0;
#100
Cin <= 1;

@eachvec;
// --> end
end
endmodule
```

e. シミュレーション

- 設計を再コンパイルする（タスク窓が？になっているはず）

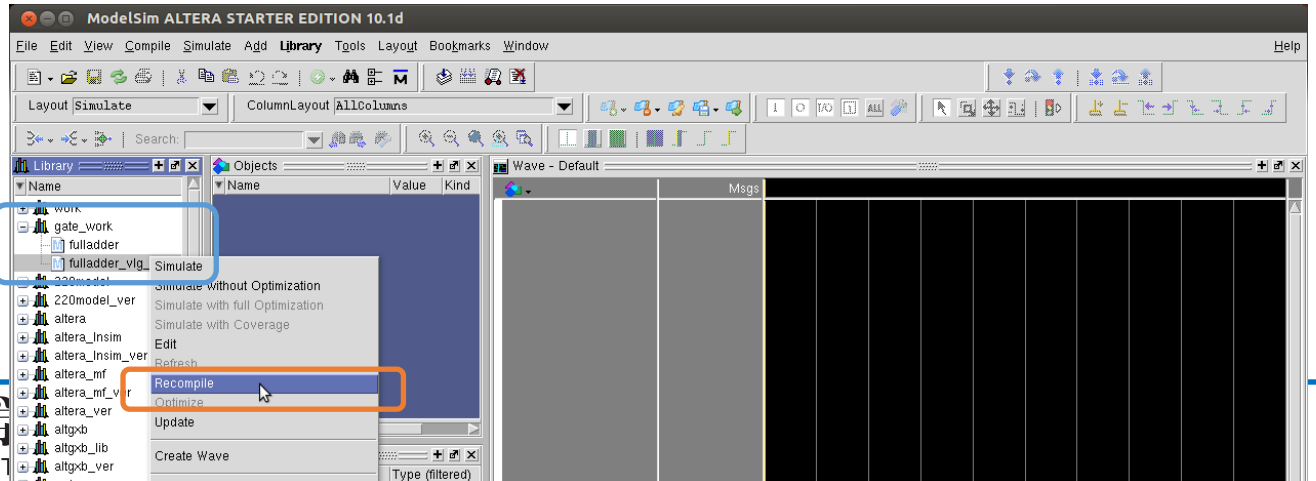
- シミュレータを起動する
  - Tools -> Run Simulation Tool -> Gate Level Simulation...
  - Timing model: “Slow -7 1.2V 100 Model” で Run





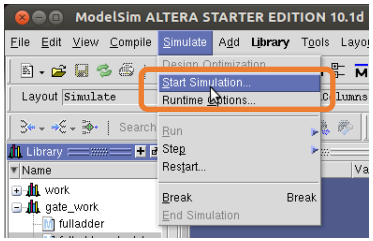
e. シミュレーション

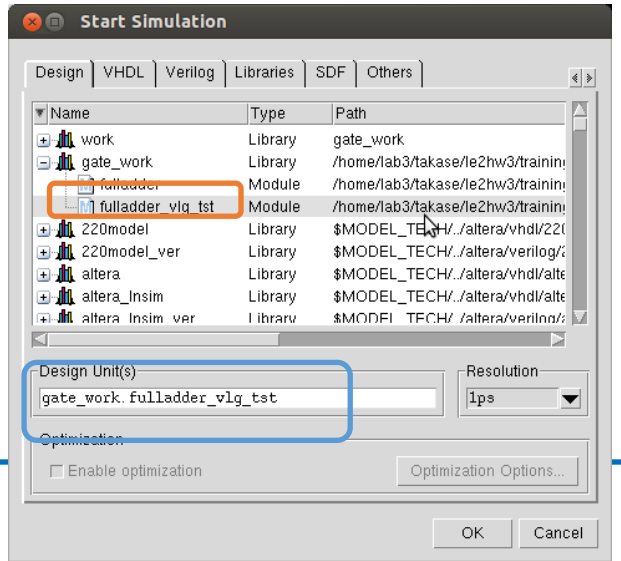
- シミュレータを起動する
  - 設計した回路は Library の gate_work 以下にある
    - ✓ work 内は gate_work へのリンク
- テストベンチファイルを念のため再コンパイル
  - “Library” ウインドウの gate_work -> <module>_vlg_tst を右クリックして “Recompile”
    - ✓ 起動直後は”# Error loading design”が出ることがあるが再コンパイルで解消されるので気にしなくてよい
    - ✓ テストベンチを書き換える都度で再コンパイルが必要となる

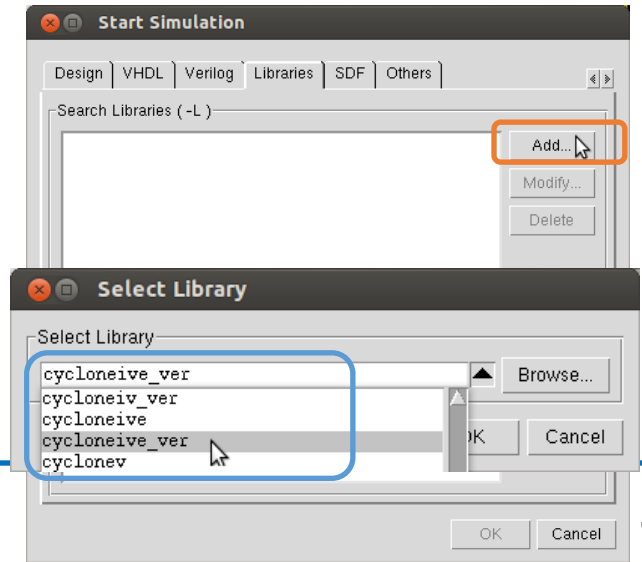


e. シミュレーション

- シミュレーションの開始
  - Simulate -> Start Simulation...
  - Design タブの ”Design Unit(s)” にて gate_work.<module>_vlg_tst を選択
  - Libraries タブの “Search Libraries” にて Add... して 必要なライブラリを追加する (”cycloneive_ver“のみ選択でよい)
    - ✓ ”altera_ver”, ”gate”, “gate_work” が自動的に入っていることがあるが，どちらでもよい



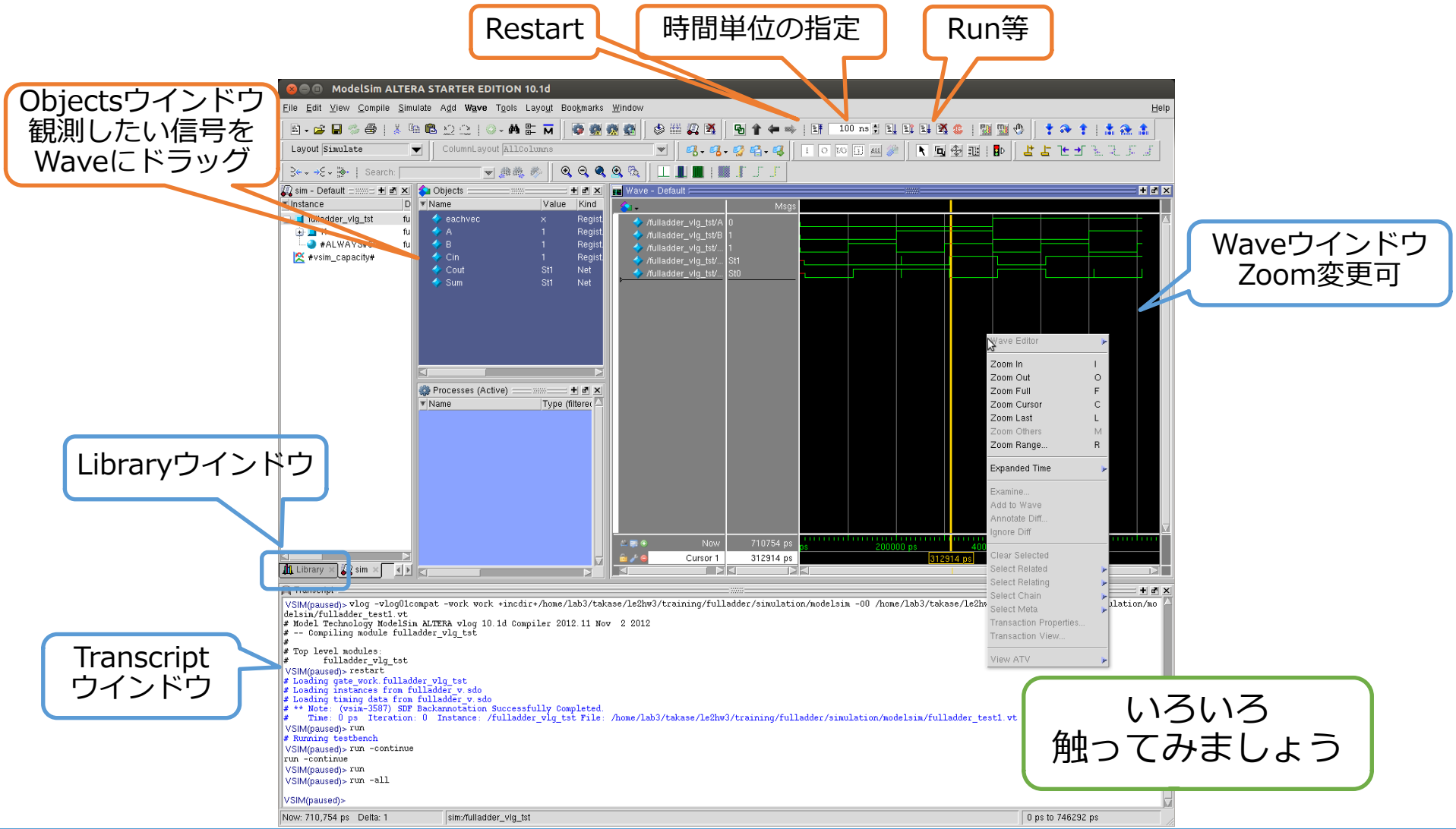




e. シミュレーション

- シミュレーションの実行
  - “Objects” ウインドウから観測したい信号線を”Wave” ウインドウにドラッグする
  - “Run Length” を所望の大きさにする（例：100ns）
    - ✓ 大きすぎるとシミュレーション時間が掛かる
  - “Run” や “Run - All” などをクリックする
    - ✓ “Transcript” ウインドウで直接入力も可能 (慣れてきたらこっちのほうが早いかも)
  - 波形の表示は右クリックまたはキー入力でZoom In/Out/Fullなどが変更可能
  - 波形はテストベンチで指定済みだが，直接指定も可能
    - ✓ 信号線の右クリックで”Force...”など
  - テストベンチを変更した場合は，“Library” ウインドウからRecompileして “Restart”

e. シミュレーション


Restart
時間単位の指定
Run等
Objectsウインドウ
観測したい信号を
Waveにドラッグ
Waveウインドウ
Zoom変更可
Libraryウインドウ
Transcript
ウインドウ
いろいろ
触ってみましょう
ModelSim ALTERA STARTER EDITION 10.1d
Wave - Default
Objects
Processes (Active)
Zoom In
Zoom Out
Zoom Full
Zoom Cursor
Zoom Last
Zoom Others
Zoom Range...
Expanded Time
Examine...
Add to Wave
Annotate Diff...
Ignore Diff
Clear Selected
Select Related
Select Relating
Select Chain
Select Meta
Transaction Properties...
Transaction View...
View ATV

B) 全加算器を用いた4ビット並列加算回路

– 他プロジェクトからの設計の流用
– バスの利用
– 並列加算回路の構成

4ビット並列加算回路

- 全加算器を接続し，4ビットの並列加算回路を設計する

- 解説項目
  a. 他プロジェクトからの設計の流用
     ✓これまでの設計回路を新規プロジェクトに追加する
  b. バスの利用
     ✓これまでは1ビット信号線のみ，
     複数ビット幅を指定できるバスを使う
  c. 並列加算回路の構成

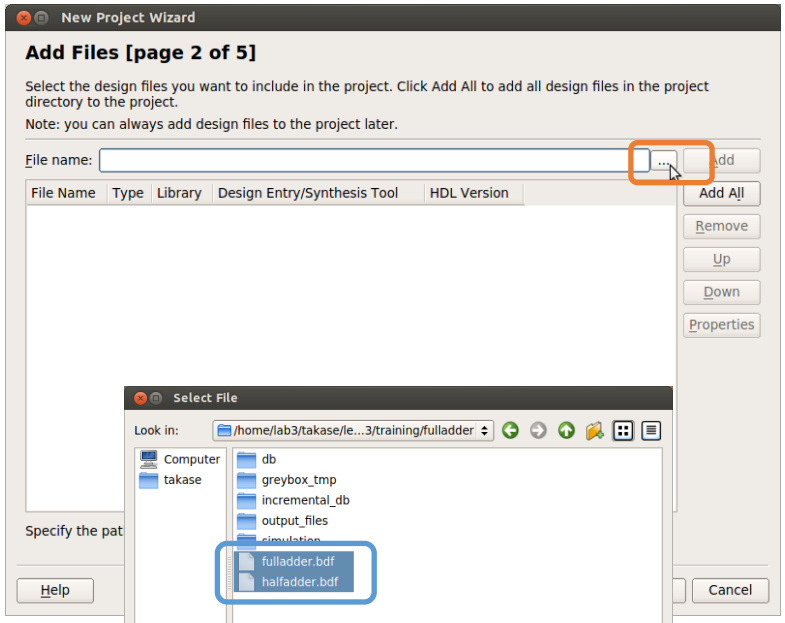
a. 設計の流用

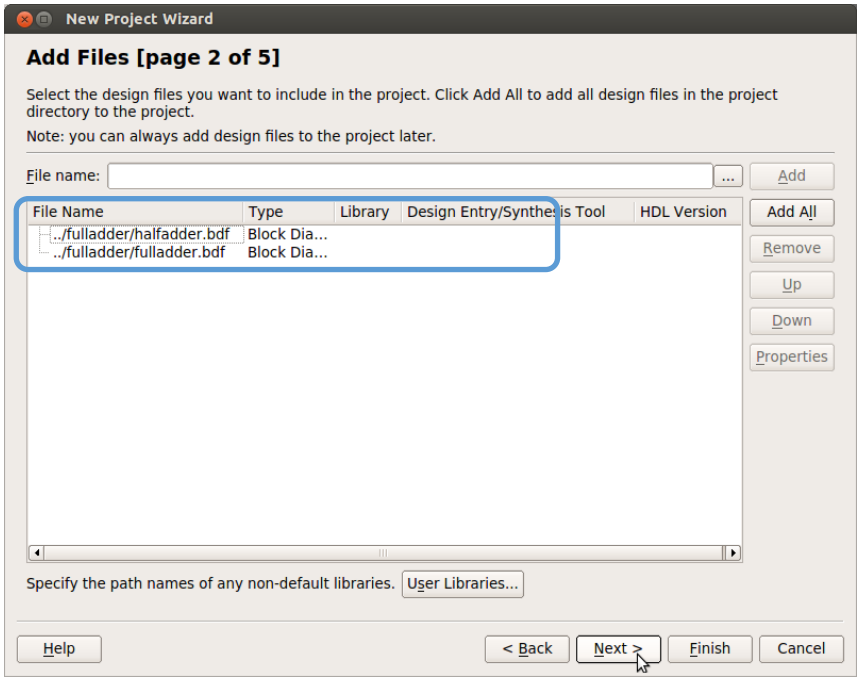
- 元プロジェクトからコピって追加する
  - 元ファイルとリンクしなくなる
  - それで良い時もあるが，元プロジェクトで設計を変更したら？？

- プロジェクト作成時の New Project Wizard で追加する
  - 元プロジェクトでシンボルファイル .bsf を作成しておく
  - “Add Files [page 2 of 5]” にて
    - ✓ 必要なファイルが決まっているなら楽
    - ✓ 必要なサブファイルも追加する
- Project Navigator ウインドウから追加する
  - Files タブ右クリックの “Add/Remove Files in Project...” にて
  - Project -> Add/Remove Files in Project... からでもOK
    - ✓ とりあえず新規プロジェクト作ってから追加できる

a. 設計の流用

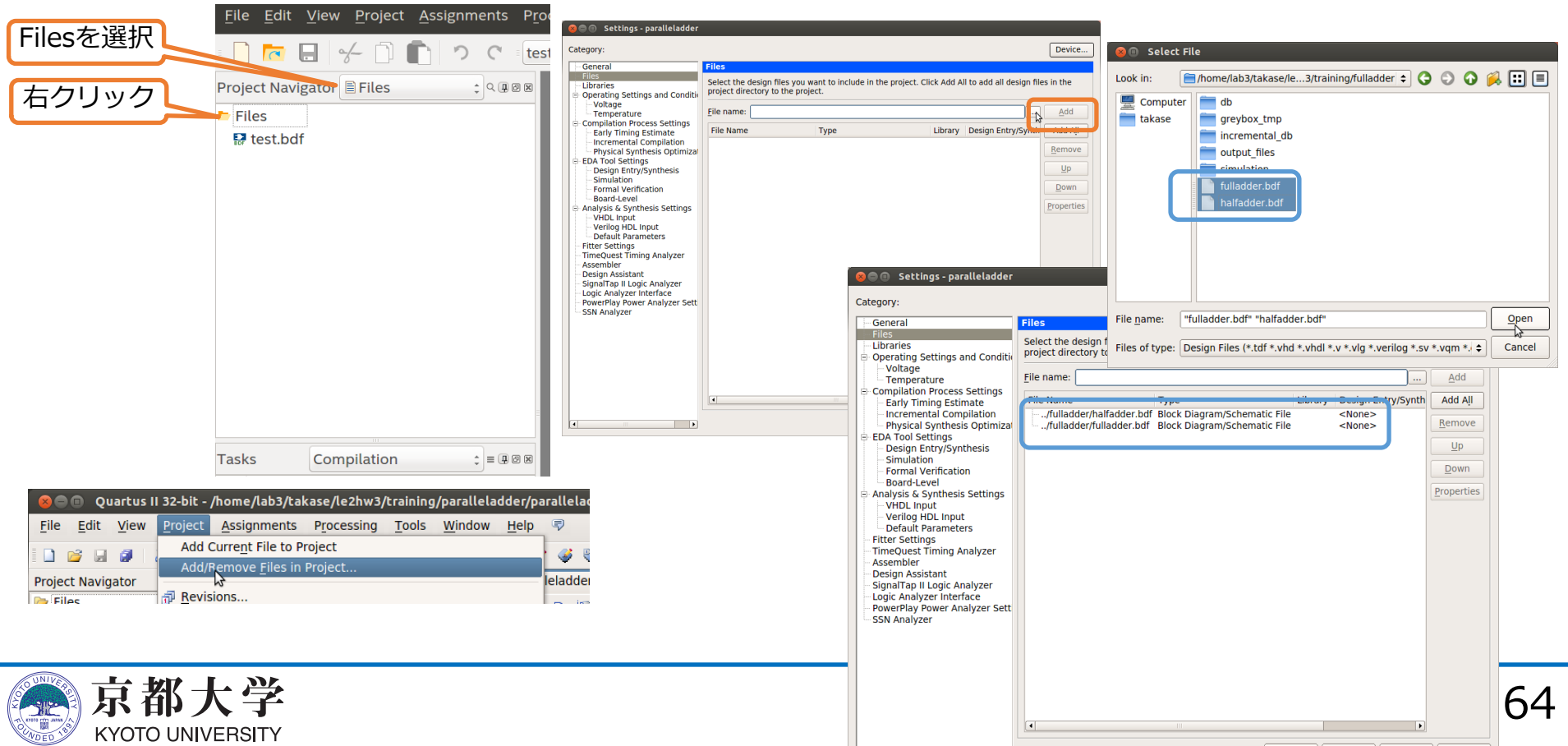
- プロジェクト作成時の New Project Wizard で追加する
  - “Add Files [page 2 of 5]” にて





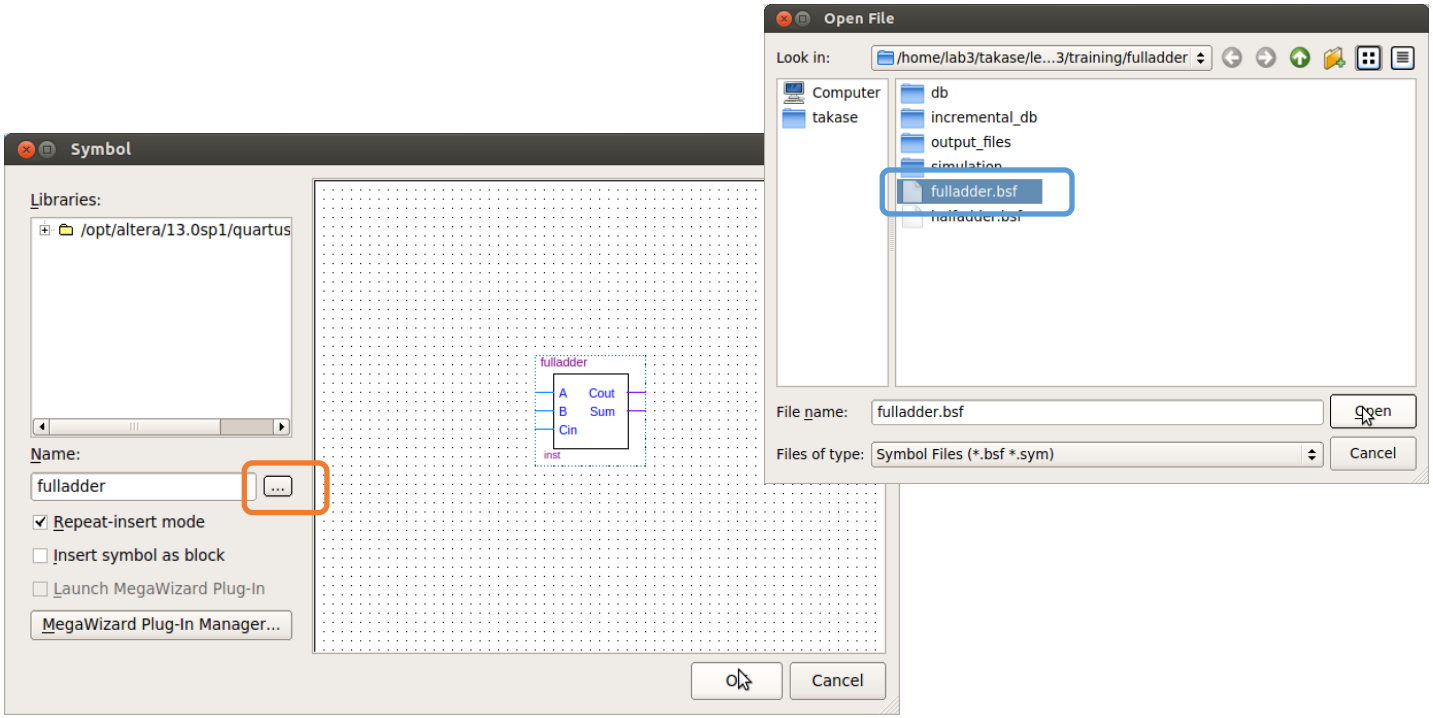
a. 設計の流用

- Project Navigator: Files ウインドウから追加する
  - Files 右クリックの “Add/Remove Files in Project...” にて
  - Project -> Add/Remove Files in Project... からでもOK

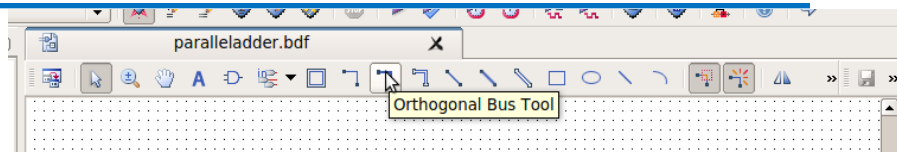


a. 設計の流用

- Symbol ウインドウから既存の .bsf を呼び出せる
  - 元プロジェクトのディレクトリから .bsf を開く

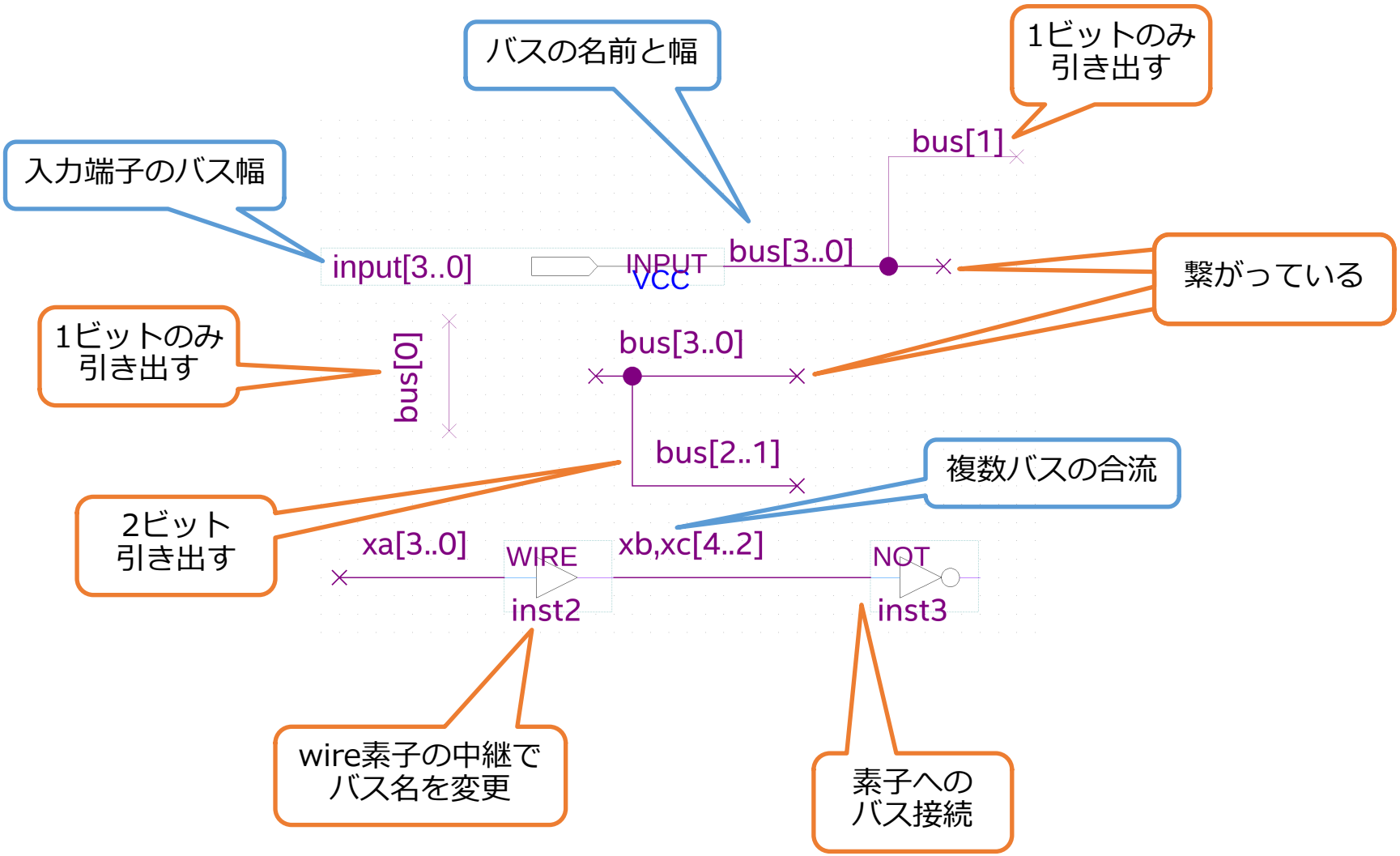


b. バスの利用



- 複数ビット幅を持つバスを引く
  - 回路図操作アイコンのOrthogonal Bus Tool
    - ✓ 太線がバス　細線は1ビットの信号線
  - 名前は右クリック ”Properties -> General -> Name” で付ける
    - ✓ バス幅は [x..y] で記述する　入出力端子にもバス幅を記述できる
      - [x..y]と[y..x]は異なる　特に理由無ければ降順で
      - 端子名とバス／信号名は必ずしも一致させる必要は無い
    - ✓ ビットの指定でバス内の一部の配線／バスを引き出すことができる
    - ✓ 離れた配線／バスに同じ名前を付けると繋がっているとみなされる
      - あまり多用するとデバッグで苦労するので，効果的に使うこと
    - ✓ 信号／バス間を合流させることができる（例：a[3..1],b[2]）
    - ✓ wire で配線名を途中から変更することができる
    - ✓ 論理素子にもバスを接続できるものがある
      - 入出力でバス幅を揃えることに注意

b. バスの利用

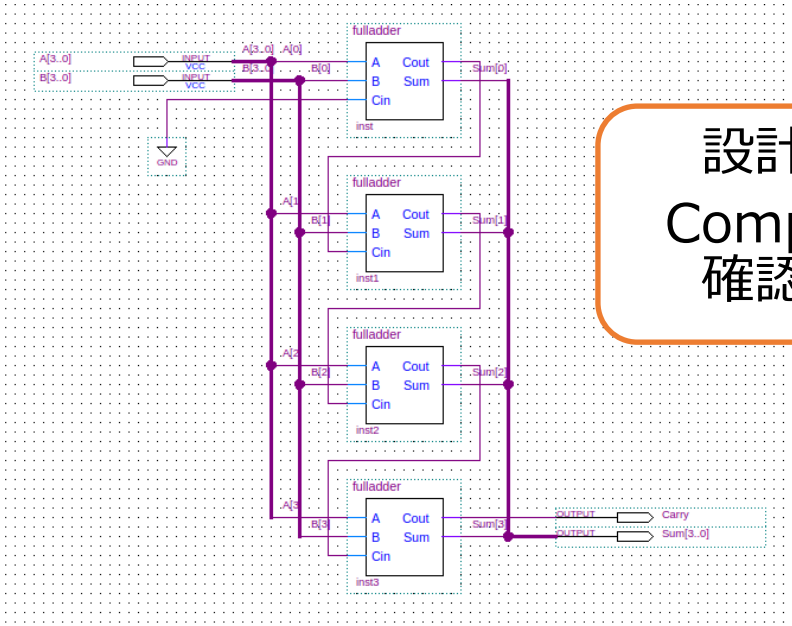

バスの名前と幅
1ビットのみ
引き出す
入力端子のバス幅
bus[1]
input[3..0]
INPUT
VCC
bus[3..0]
繋がっている
1ビットのみ
引き出す
bus[0]
bus[3..0]
bus[2..1]
複数バスの合流
2ビット
引き出す
xa[3..0]
WIRE
xb,xc[4..2]
NOT
inst2
inst3
wire素子の中継で
バス名を変更
素子への
バス接続

c. 並列加算回路の設計

- 回路構成
  - 全加算器を直列に接続する
  - 加算データは並列に接続される

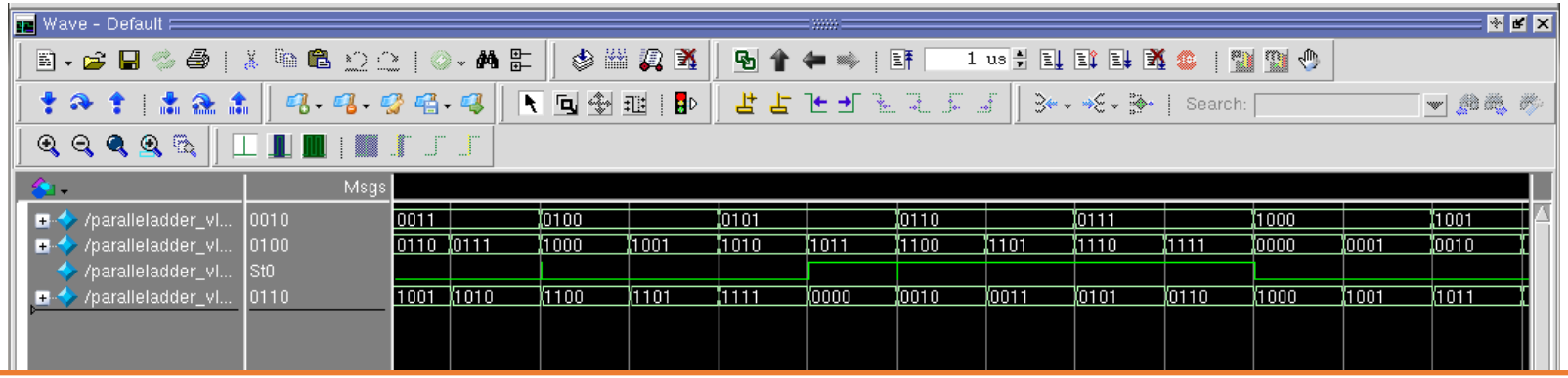
- 外部仕様
  - 入力：4ビットの加算データA,B
  - 出力：4ビットの和Sum，1ビットの桁上げCarry
  - (せっかくなので)設計制約：
    - ✓ 先に設計した全加算器と半加算器を利用すること
      - 全加算器のみでも構わない
    - ✓ 入力，出力端子ともにバス幅を指定すること

c. 並列加算回路の設計



設計してみましょう

Compilation Reportも確認してみましょう



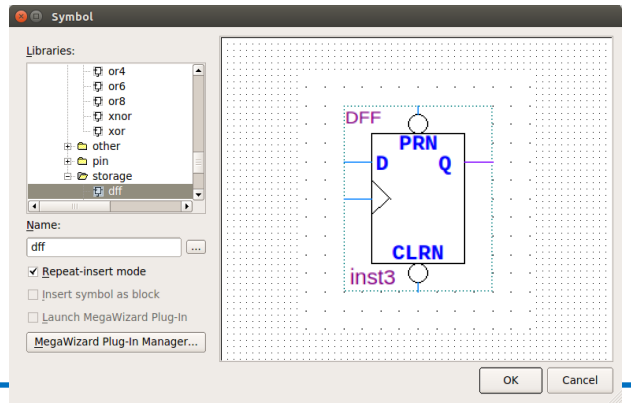
シミュレーション波形で正しく動いていることも確認しましょう

実習課題(2)：カウンタの設計

– フリップフロップの利用
– 3ビット5進カウンタの設計
– クロックとタイミング制約の設定

a. フリップフロップの利用

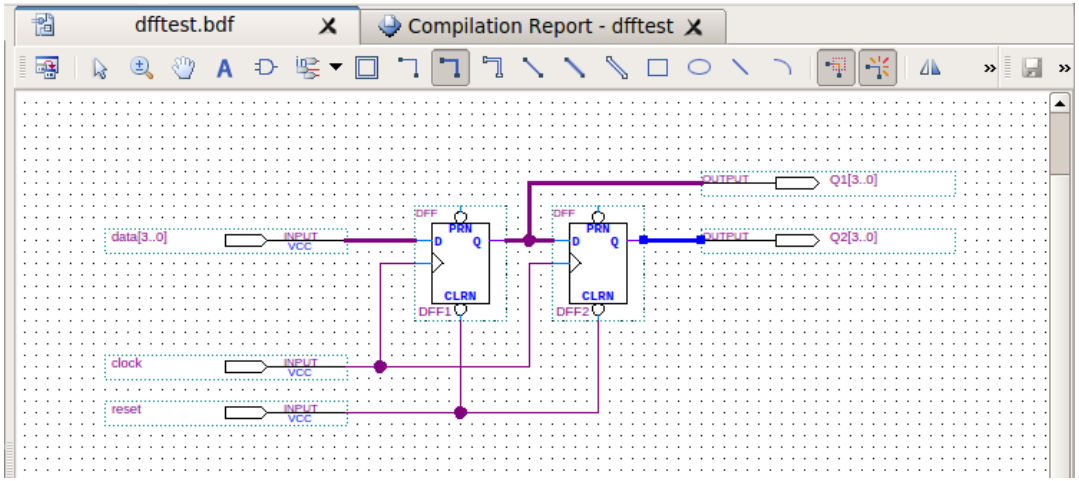
- データを一時的に保持できる記憶素子
  - 順序回路を構成する基本要素
  - 基本的にはD-FFを使用するのがよい
    - ✓ クロック・パルスが立ち上がった時に，入力値Dを取り込み，次のクロック・パルスで更新されるまでQより出力し続ける
    - ✓ 実際には，セットアップ時間とホールド時間の間でDの値を一定に保たないと正しい値を保持できない（要はタイミング制約）
    - ✓ Qは，CLRNに0を入力すると0に，PRNに0を入力すると1に固定される
      - CLRNもPRNも負論理

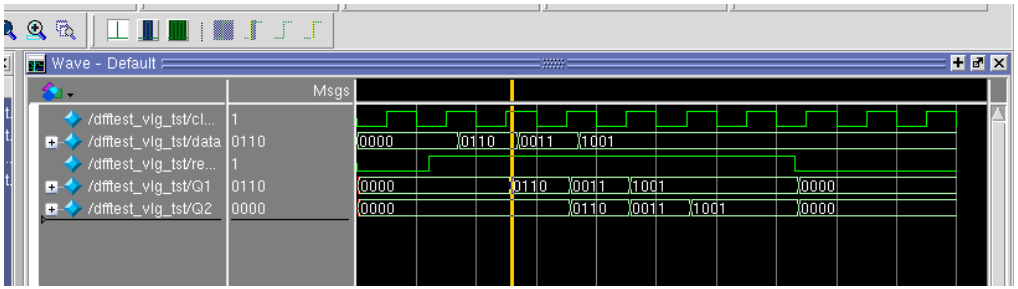


PRN	CLRN	CLK	D	Q
L	H	x	x	H
H	L	x	x	L
L	L	x	x	L
H	H	↑	L	L
H	H	↑	H	H
H	H	L	x	Qo
H	H	H	x	Qo

a. フリップフロップの利用

- よくわからない人は，入出力とD-FFのみの簡単な回路 “dfftest” を設計して，D-FFの動作をシミュレーションで確認してみましょう





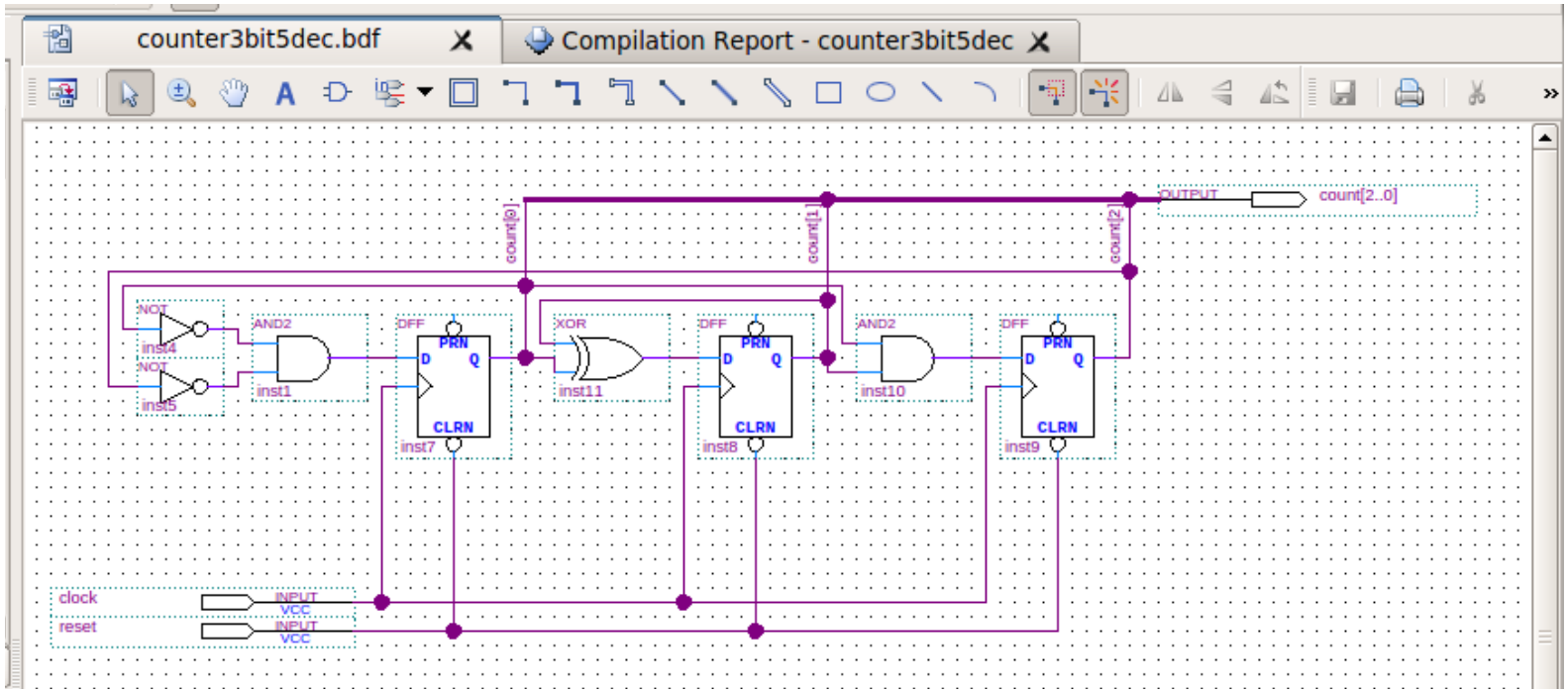

```
        .reset(reset)
);
initial
begin
// code that executes only once
// insert code here --> begin
clock <= 0;
reset <= 0;
data  <= 4'b0000;
// --> end
$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
#120
reset <= 1;
#50
data  <= 4'b0110;
#100
data  <= 4'b0011;
#100
data  <= 4'b1001;
#360
reset <= 0;

@eachvec;
// --> end
end

always begin
        #50 clock <= ~clock;
end

endmodule
```

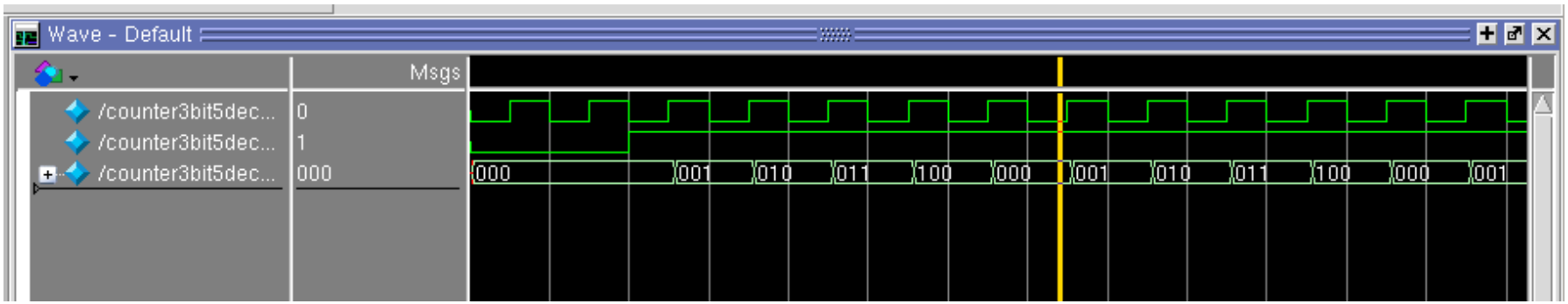
b. 3ビット5進カウンタの設計

- 回路構成
  - 同期式順序回路とする
    - ✓ 非同期式にするとCADツールの最適化との相性が悪くなることが多い
  - クロック信号の立ち上がり毎に値をインクリメントしていく
    - ✓ 0 -> 1 -> 2 -> ... -> 4 -> 0 -> 1 -> 2 -> ...
  - リセット信号が0の時は0を出力する
- 外部仕様
  - 入力：1ビットのクロック信号clock
1ビットのリセット信号reset
  - 出力：3ビットのカウンタ値count

Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	x	x	x
1	1	0	x	x	x
1	1	1	x	x	x

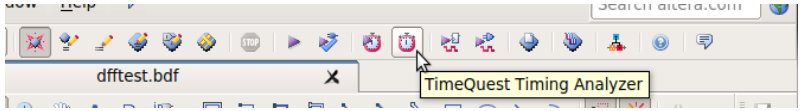
b. 3ビット5進カウンタの設計

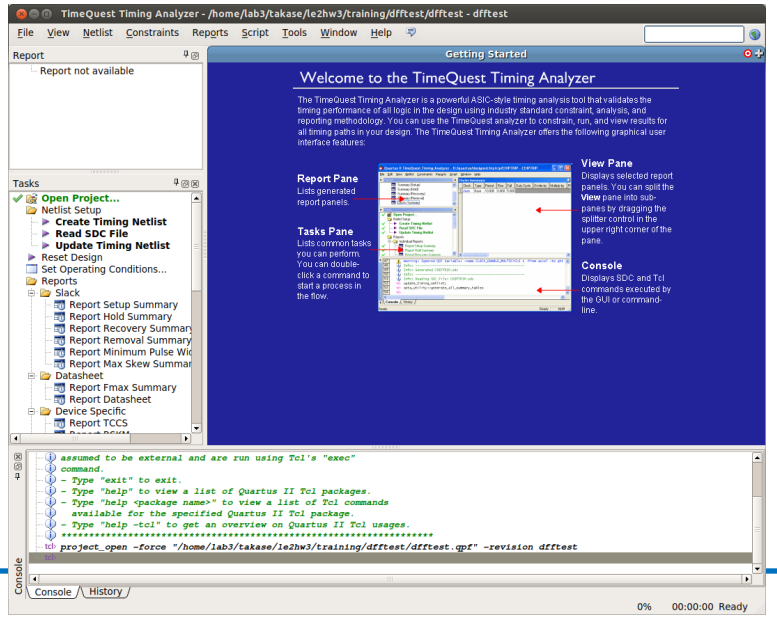




c. クロックとタイミング制約の設定

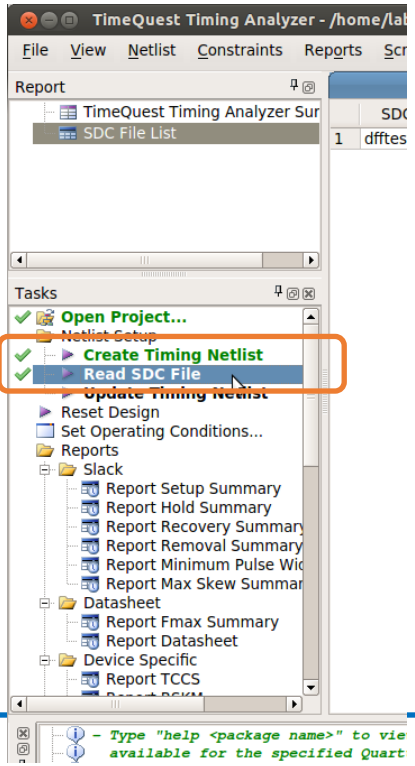
- クロックの指定やタイミング制約（レジスタ入力へのセットアップ時間やホールド時間，伝搬遅延など）の設定を行う
  - 基本的なクロックの指定のみ解説，詳細は各自で調査すること.
- TimeQuest Timing Analyzerを使用する
  - Tools -> TimeQuest Timing Analyzer または，
    ツール操作アイコンの “TimeQuest Timing Analyzer”

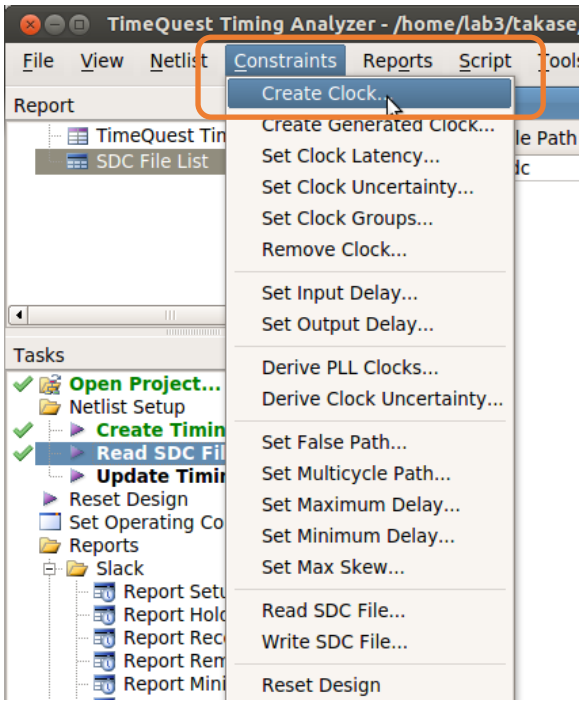




c. クロックとタイミング制約の設定

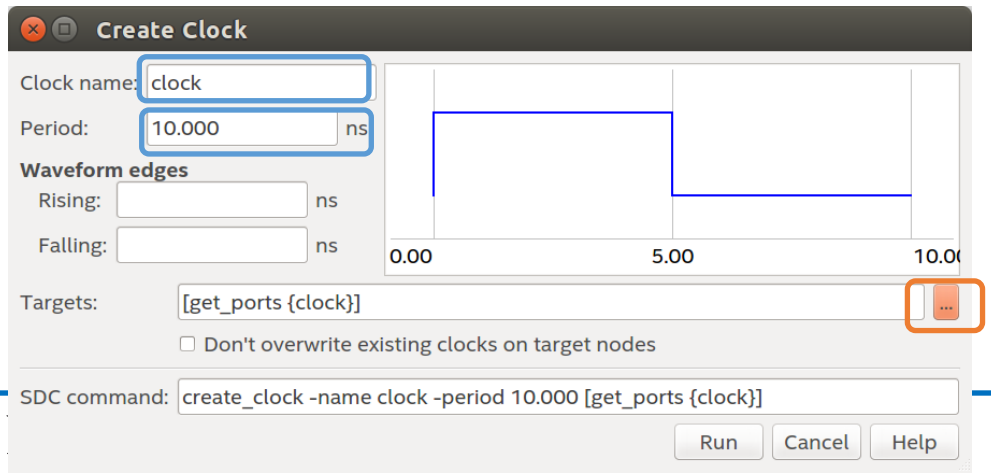
- TimeQuest Timing Analyzer
  - Tasks窓の “Netlist Setup -> Create Timing Netlist” を実行
  - Tasks窓の “Netlist Setup -> Read SDC File” を実行
  - Constraints -> Create Clock... からクロックを設定

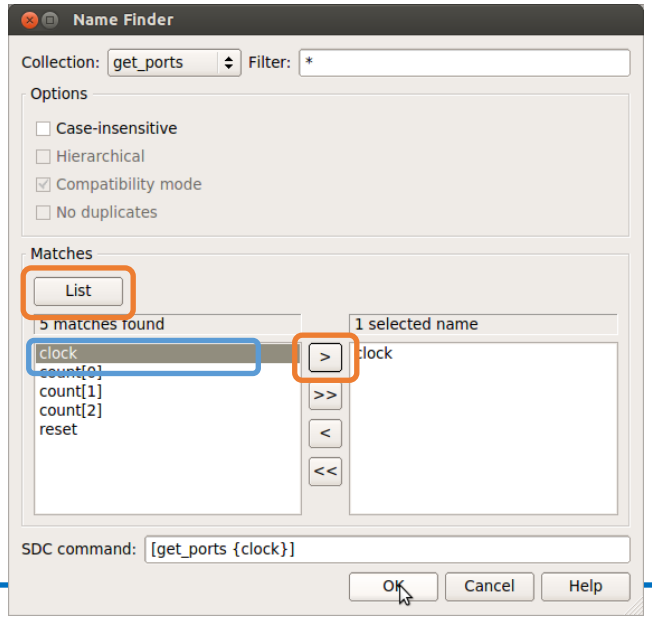




c. クロックとタイミング制約の設定

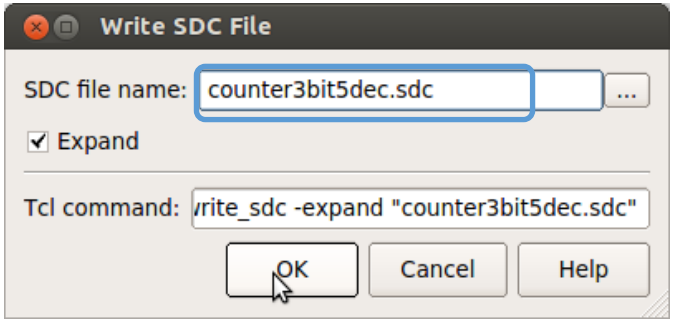
- TimeQuest Timing Analyzer
  - Constraints -> Create Clock... からクロックを設定
    - ✓ Clock name: クロック名の定義（例：clock，未記入でも可）
    - ✓ Period: クロック周期の設定
    - ✓ Waveform edges: 立ち上がり／立ち下がり時間の設定
    - ✓ Targets: 設計回路上のクロックを選択
      - “List”から表示して”>”で選択
    - ✓ SDC commandに生成される制約が表示される





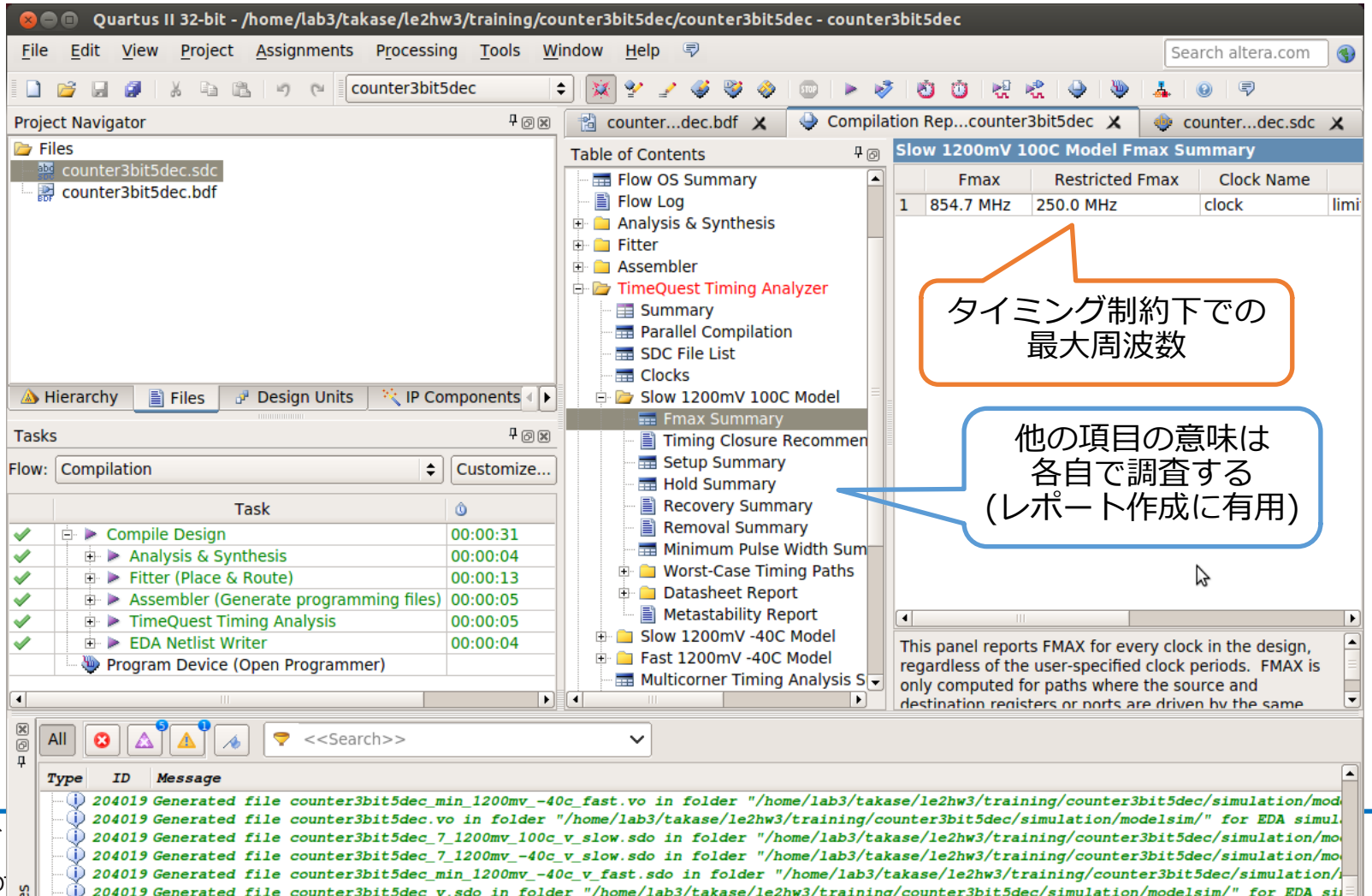
c. クロックとタイミング制約の設定

- TimeQuest Timing Analyzer
  - Reports/ 以下でタイミングに関する種々のレポートが報告されるようになる
    - ✓ 満足していないものがあれば制約の追加などで対処する
    - ✓ Compilation Report からも確認できるようになる
    - ✓ Fmax Summary: 動作可能な最大周波数（制約によって変わる）
    - ✓ Clocks: 現在設定値での設計回路のクロック周波数
  - Tasks窓の “Write SDC File...” で制約ファイルを保存
    - ✓ <module>.sdc とするとプロジェクトに自動認識される（.outを取ったほうがよい）
    - ✓ 手動で追加は Project Navigator ->Files から Add/Remove Files in Project...
    - ✓ 慣れてきたらテキストベースで制約ファイルを作成・編集してもよい



c. クロックとタイミング制約の設定

- 性能の確認（Compilation Reportから）



TIPS

- 回路図エディタについて
  - エディタ上のとにかく綺麗な素子配置・配線を心掛けましょう
    スパゲッティ回路図になると，あとで自分も(TAも)大変です．
  - 直線(斜線)の配線もできますが，見た目がすごく悪くなります．
- インスタンス／バス／端子の命名について
  - 記号はハイフン '-' とアンダースコア '_' くらいしか使えません．
    - ✓ハイフンはModelSimでマイナスと認識されてしまいます．
      つまり実質的に使えるのはアンダースコアだけです．
  - 大文字と小文字は区別しません．
  - ビット指定は[]を忘れずに．A[2]とA2は異なる信号線になります．
  - 数字で始まるモジュール名等はシミュレーションでハマります
  - 既存の論理素子など一般的すぎるもの（いわゆる予約語）は
    思わぬバグ混入のもとなので，避けたほうがよいでしょう．

TIPS

- 論理素子について
  - 基本的にはPrimitivesのみを使ってください.
  - 特にDFFのCLRNやPRNなど，負論理であるものがあります.
  - 決まった値を入力したい場合はVCCとGNDが使えます.
  - なにも接続していないところは0とみなされる（はずです）
  - 各論理素子の動作(真理値表)および説明を見たい場合は,
    Help -> Help Topics 内の検索窓または,
    Using HDL with the Quartus Prime Software
    -> List of Primitives の下から探してください.
    - ✓ Primitivesページのリンクは飛べないようです

TIPS

- モジュールについて
  - 回路全体ではなく一部だけを合成・シミュレーションしたい場合は，Project Navigator の該当ファイルを右クリックして ”Set as Top-Level Entity” を設定するとよいです.
  - シンボルの元の回路図を変更した場合は，シンボルの右クリック “Update Symbol or Block...” すると変更が反映されます.
- シミュレーション
  - modelsim-aseの Transcript サブウィンドウに直接コマンドを入力して実行できます.
    慣れてきたらメニューをクリックするより操作が早いですし，コマンドヒストリも使えます.
  - 実行履歴は “./simulation/modelsim/msim_transcript” に残るので，編集して別名で保存しておくのもよいです.
    Transcriptから do (ファイル名) で実行できます.

TIPS

- 各自PCへのQuartus Primeの導入
  - 機能制限有りの Lite Edition なら無償利用可能です
    http://dl.altera.com/?edition=lite
  - シミュレータはModelSim-Altera Starter Editionを入れてください
  - 演習室環境と同じStandard Editionだと30日間限定になります
    来年度実験3のことを考えると導入時期は要検討です.
  - Pro EditionはFPGAボード搭載のCyclone IVデバイスが対応しません

TIPS

- エディションの違いについて
  - 本演習の運用レベルでは大きな違いはありません.
    ただしLiteだとマルチプロセッササポートがありません.
    https://www.altera.co.jp/ja_JP/pdfs/literature/po/ss-quartus-comparison_j.pdf
- 合成性能・最適化結果がエディションで異なってくる可能性があります
  - レポート記載の際には Standard Edition を使用するか,
    実験環境として使用した Edition を明記してください
- バージョン
  - Google先生に聞く時はツールのバージョンに注意してください.
    Quartus Prime 17.1 Std
    ModelSim-Altera 10.5b
  - 古い／新しいために，実験環境のバージョンでは通用しない
    TIPSや解決策に当たることがあります.

演習課題

- 演習課題1：ALUの設計
  - 配布資料に示す外部仕様を満たす16ビットのALU（算術論理演算器）を設計しなさい.
  - 複数の構成方式を検討し，それらの比較を（少なくとも定性的に）示しなさい.
  - 必須機能と発展機能に分かれる.
    - ✓発展機能，複数方式の定量的な比較評価は，加点要素となる.
- 演習課題2：任意の順序回路の設計
  - 任意の同期式順序回路について仕様を検討し，CADツールを用いて論理設計しなさい.
  - ただし，演習課題1のALUを用いること.

演習課題

- レポート内容をまとめた報告書および設計データを提出すること.
  - 報告書：演習課題1と2の報告内容を1つのPDFファイルにまとめること
  - 設計データ：演習課題2のみ　プロジェクト全体をzipでまとめること
- 提出方法：PandAの課題提出ページ
- 提出期限：2020年1月28日(火) 17:00　**※期限厳守！**
  - レポートの品質によっては再提出を指示することがある.
    再提出に該当する場合は 2020年2月3日(月) 17:00 までに連絡する.
  - KULASISからの連絡を確認できるようにして，連絡があれば速やかに応じること.

- 詳細は，配布資料（PandAにも掲載）を参照のこと.
- 配布資料に示す注意事項をよく確認すること.