



# Zynq + Synthesizer 入門

わさらぼ 三好 健文

2015.3.16

# この資料について

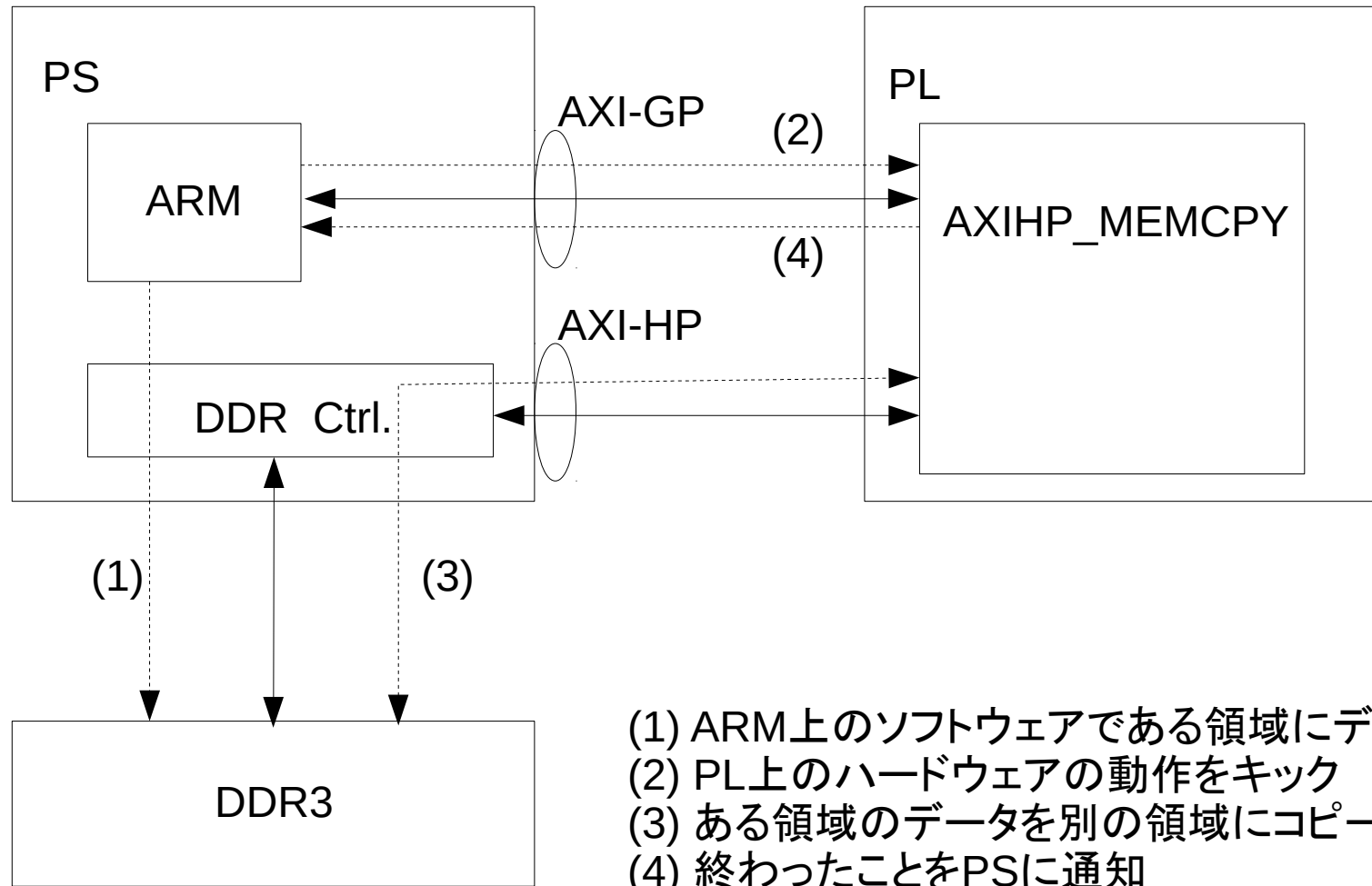
- ✓ この資料はZynqとSynthesizerを組み合わせたシステム設計の  
とっかかりにしてもらくことを目的にしています
- ✓ ターゲットボードは、ZedboardおよびZyboです
- ✓ Linuxでの実行を想定しています。

Windowsではコマンドを多少読み違える必要があります。

- ✓ Java8が必要です。
- ✓ 例題および流れは 杉本様 作の  
Zynq + Vivado HLS入門  
慶應義塾大学 天野研究室 杉本 成  
<http://www.slideshare.net/narusugimoto/zynq-vivado-hls>  
になっています

# この入門でのゴール

DDR3のある領域(src)を別の領域(dst)にコピーする



# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# Synthesijer関連リソースの準備

## 作業の概要

- ✓ Synthesijer用のディレクトリ(例 \$HOME/synthesijer)を作成
- ✓ Synthesijer<sup>\*1</sup>のページからjar, lib, applicationsをダウンロード
- ✓ lib, applicationsを展開
- ✓ jarにSYNTHESIJERという環境変数をセット
- ✓ libの展開ディレクトリにSYNTHESIJER\_LIBをセット
- ✓ applicationsの展開ディレクトリにSYNTEHSIJER\_APPをセット
- ✓ 作業用ディレクトリ(\$HOME/synthesijer/work)を作って移動

<sup>\*1</sup> <https://sourceforge.net/projects/synthesijer/files/synthesijer-2.0/>

# Synthesijer関連リソースの準備

## 作業の例

```
% mkdir $HOME/synthesijer
% cd $HOME/synthesijer
% wget https://sourceforge.net/projects/synthesijer/files/synthesijer-2.0/
synthesijer-20150315.jar
% wget https://sourceforge.net/projects/synthesijer/files/synthesijer-2.0/
synthesijer_lib_20150315.zip
% wget https://sourceforge.net/projects/synthesijer/files/synthesijer-2.0/
synthesijer-applications_20150315.zip
% export SYNTHESIJer=$HOME/synthesijer/synthesijer-20150315.jar
% unzip synthesijer_lib_20150315.zip
% export SYNTHESIJer_LIB=$HOME/synthesijer/synthesijer_lib_20150315
% unzip synthesijer-applications_20150315.zip
% export SYNTHESIJer_APP=$HOME/synthesijer/synthesijer-applications_20150315
% mkdir $HOME/synthesijer/work
% cd $HOME/synthesijer/work
```

# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行



# PL上のハードウェアの設計

## 作業の概要

- ✓ Javaコードを記述
- ✓ SynthesijerでJavaコードをコンパイル
- ✓ IPパッケージの作成(必要なソースをディレクトリにコピー)
  - \$SYNTHESIJER\_APP/hdl/vhdl/axi\_lite\_slave\_32.vhd
  - \$SYNTHESIJER\_LIB/vhdl/dualportram.vhd
  - \$SYNTHESIJER\_APP/hdl/vhdl/simple\_axi\_memiface\_32.vhd
  - synthesijer\_lib\_axi\_SimpleAXIMemIface32RTLTest.vhd
  - AXIHP\_MEMCPY.vhd

# Javaのコードを記述

## AXHP\_MEMCPY.java を書く

```
import synthesizer.lib.axi.*;
import synthesizer.rt.*;

class AXIHP_MEMCPY{

    private final AXILiteSlave32RTL s0 = new AXILiteSlave32RTL();
    private final SimpleAXIMemIface32RTLTest m0 =
        new SimpleAXIMemIface32RTLTest();

    private void run(){
        int src_addr = s0.data[1];
        int dest_addr = s0.data[2];
        for(int i = 0; i < 256; i++){
            int d = m0.read_data(src_addr + (i<<2));
            m0.write_data(dest_addr + (i<<2), d);
        }
    }

    @auto
    public void main(){
        s0.data[0] = 0x00000000;
        while(s0.data[0] == 0x00000000) ; // wait for kick from PS
        run();
        s0.data[0] = 0x00000000; // to notify DONE to PS
    }
}
```

# SynthesijerでJavaコードをコンパイル

## AXIHP\_MEMCPYをSynthsijerでコンパイル

```
% java -cp $SYNTHESIJER:$SYNTHESIJER_APP/bin:. synthesijer.Main \  
  --ip-exact=AXIHP_MEMCPY \  
  AXIHP_MEMCPY.java \  
  $SYNTHESIJER_APP/src/synthesijer/lib/axi/AXILiteSlave32RTL.java \  
  $SYNTHESIJER_APP/src/synthesijer/lib/axi/SimpleAXIMemIface32RTL.java \  
  $SYNTHESIJER_APP/src/synthesijer/lib/axi/SimpleAXIMemIface32RTLTest.java  
SchdulerBoard init: AXIHP_MEMCPY  
SchdulerBoard init: synthesijer.lib.axi.SimpleAXIMemIface32RTLTest  
Compile: AXIHP_MEMCPY  
Info: enters into >>>  
  
...  
Output VHDL: AXIHP_MEMCPY.vhd  
Output VHDL: synthesijer_lib_axi_SimpleAXIMemIface32RTLTest.vhd  
  
...  
% ls AXIHP_MEMCPY.vhd  
AXIHP_MEMCPY.vhd  
% ls AXIHP_MEMCPY_v1_0/  
component.xml  src  xgui
```

コンパイル  
メッセージ

← 作成されたHDLコードを確認

← IPパッケージ用のテンプレートディレクトリ

# IPパッケージの作成

## 必要なソースコードをIPパッケージ用ディレクトリにコピー

```
% grep src AXIHP_MEMCPY_v1_0/component.xml
<spirit:name>src/axi_lite_slave_32.vhd</spirit:name>
<spirit:name>src/dualportram.vhd</spirit:name>
<spirit:name>src/synthesijer_lib_axi_SimpleAXIMemIface32RTLTest.vhd</sp...
<spirit:name>src/simple_axi_memiface_32.vhd</spirit:name>
<spirit:name>src/AXIHP_MEMCPY.vhd</spirit:name>
<spirit:name>src/axi_lite_slave_32.vhd</spirit:name>
<spirit:name>src/dualportram.vhd</spirit:name>
<spirit:name>src/synthesijer_lib_axi_SimpleAXIMemIface32RTLTest.vhd</sp...
<spirit:name>src/simple_axi_memiface_32.vhd</spirit:name>
<spirit:name>src/AXIHP_MEMCPY.vhd</spirit:name>
%
% cp $SYNTHESIJER_APP/hdl/vhdl/axi_lite_slave_32.vhd AXIHP_MEMCPY_v1_0/src
% cp $SYNTHESIJER_LIB/vhdl/dualportram.vhd AXIHP_MEMCPY_v1_0/src
% cp synthesijer_lib_axi_SimpleAXIMemIface32RTLTest.vhd AXIHP_MEMCPY_v1_0/src
% cp $SYNTHESIJER_APP/hdl/vhdl/simple_axi_memiface_32.vhd AXIHP_MEMCPY_v1_0/src
% cp AXIHP_MEMCPY.vhd AXIHP_MEMCPY_v1_0/src/
% ls AXIHP_MEMCPY_v1_0/src
AXIHP_MEMCPY.vhd
dualportram.vhd
synthesijer_lib_axi_SimpleAXIMemIface32RTLTest.vhd
axi_lite_slave_32.vhd
simple_axi_memiface_32.vhd
```

必要なファイル

必要なファイルがコピーできた

# 作業の手順

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# Vivadoでの合成

---

## 作業の概要

- ✓ Vivadoのプロジェクト作成
- ✓ Processing System (PS) の追加とパラメタ設定
- ✓ AXIHP\_MEMCPYモジュールの追加
- ✓ HDLラッパーの生成と修正
- ✓ 合成

# プロジェクト作成～PSの追加・設定

- ✓ 基本的には

Zynq + Vivado HLS入門

慶應義塾大学 天野研究室 杉本 成

<http://www.slideshare.net/narusugimoto/zynq-vivado-hls>

の

p.58(VIVADO “Projectの作成 1/9”)～

p.83(VIVADO “PS入出力ポート生成3/3”)を参照

ただし、次のような手順で設定.

- ✓ プロジェクトは\$HOME/synthesijer/workの下にproject\_1として作成
- ✓ Zedboardの場合: プリセットを最初に読んでUART1以外を削除
- ✓ Zyboの場合: ZYBO\_zynq\_def.xmlをimortしてUART1以外を削除
- ✓ HP0の幅を32bitに変更(PSの設定5/7 相当)
- ✓ 割り込みはなし(PSの設定6/7 相当はスキップ)

# AXIHP\_MEMCPYモジュールの追加

## 作業の概要

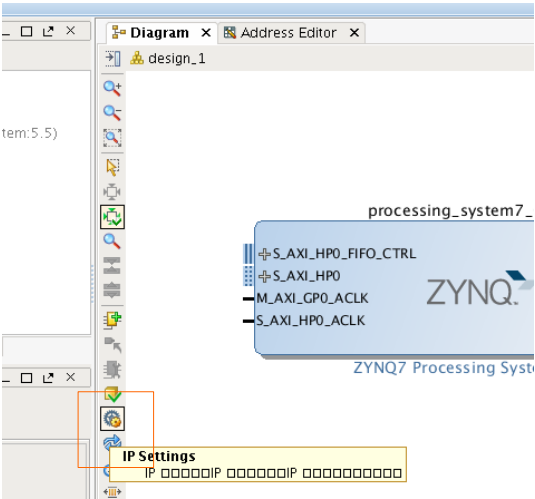
- ✓ IPコア参照リポジトリの追加
- ✓ コアのインスタンス化
- ✓ ポートの処理
  - ✓ AXIポートの接続(自動接続に任せる)
  - ✓ AXIHP\_MEMCPYの雑多なポートの処理
  - ✓ axi\_inter\_memconのACLK/ARESETNの処理
  - ✓ peripheral\_aresetnを外部に引き出す



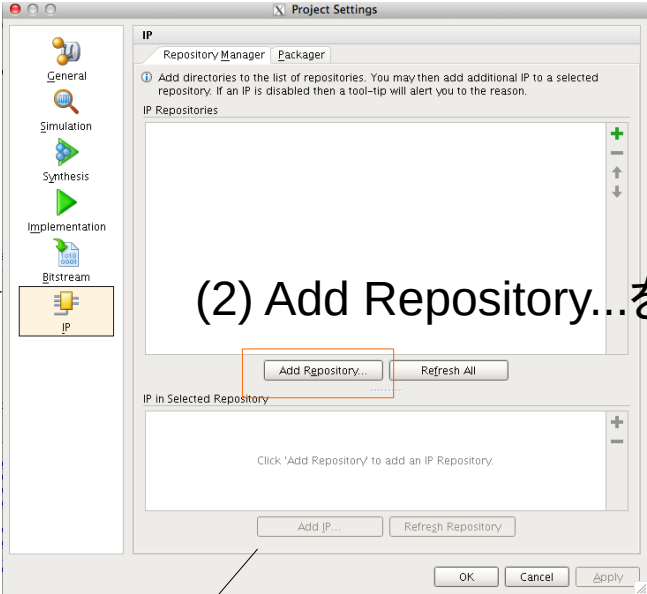
# AXIHP\_MEMCPYモジュールの追加(1)

## IPコア参照リポジトリの追加

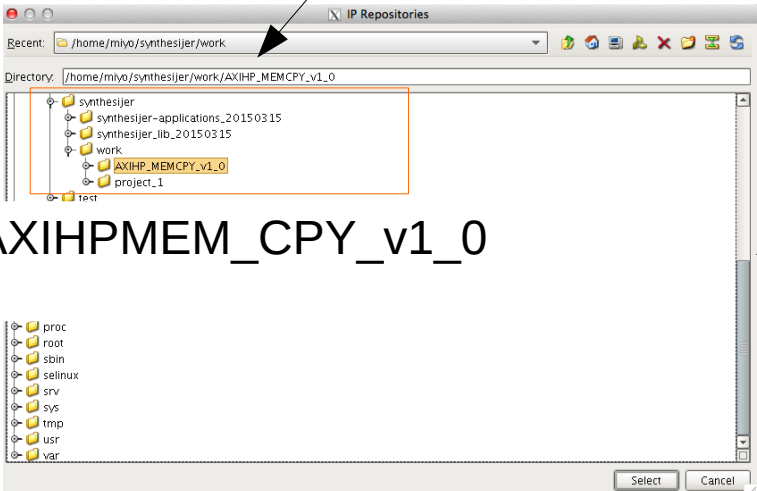
(1) 設定ダイアログを開く



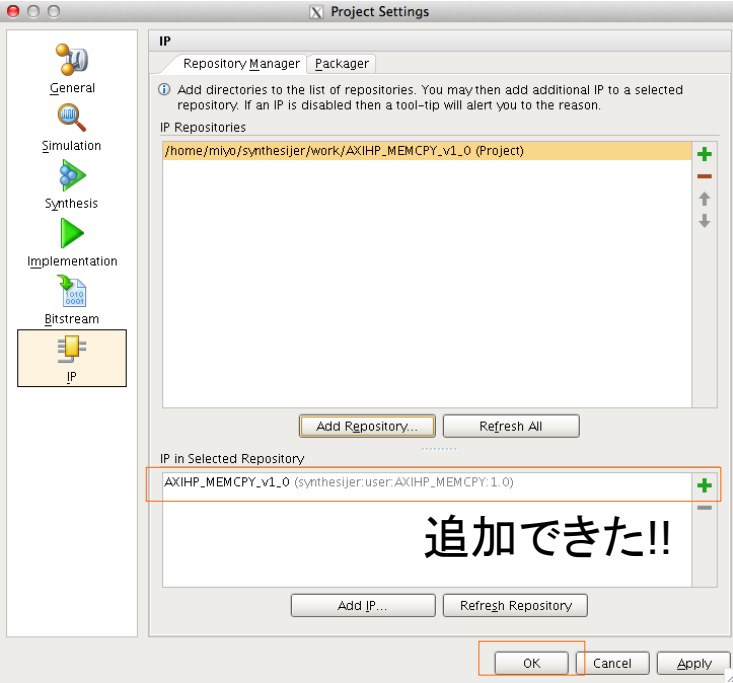
(2) Add Repository...をクリック



(3) 作成したAXIHPMEM\_CPY\_v1\_0を選択



(4) AXIHPMEM\_CPY\_v1\_0が見えたらOK. [OK]で終了

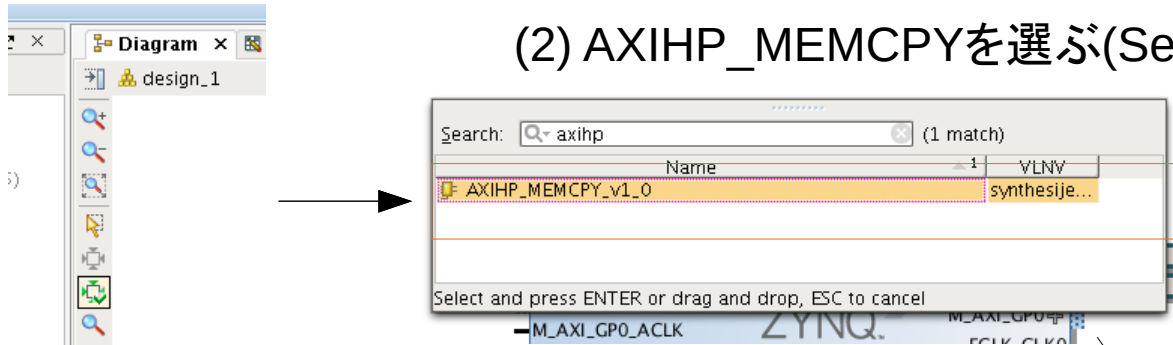


追加できた!!

# AXIHP\_MEMCPYモジュールの追加(2)

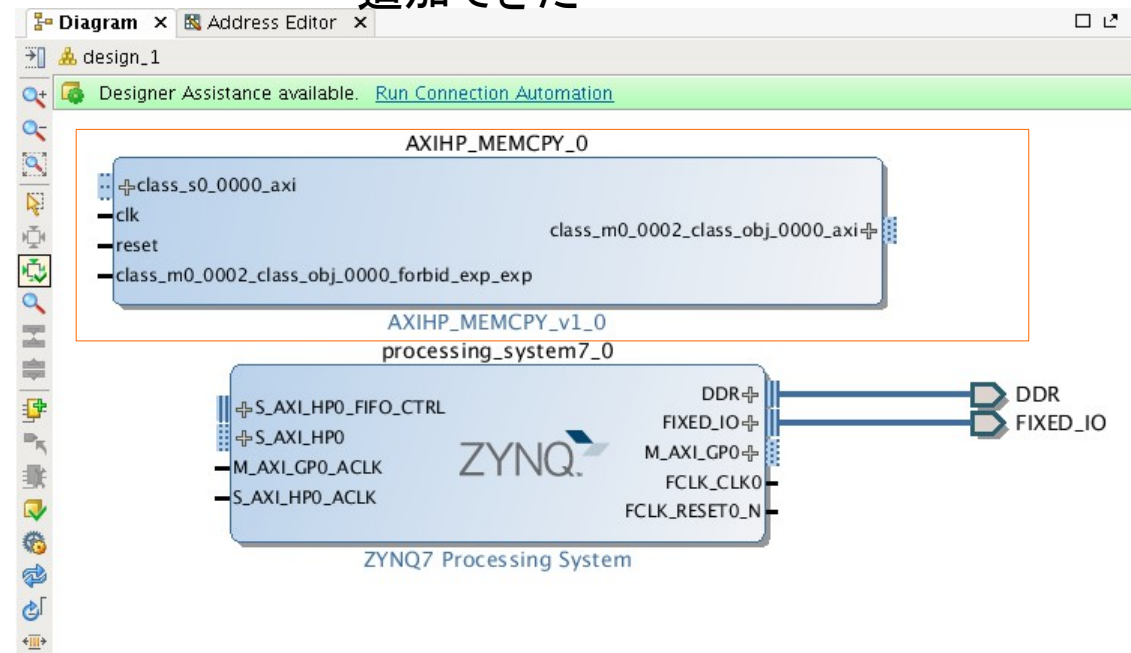
## コアのインスタンス化

(2) AXIHP\_MEMCPYを選ぶ(Search:を使うと楽に選択できる)



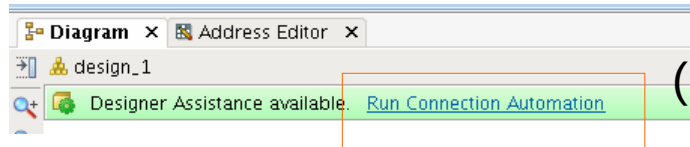
(1) コアの追加ダイアログをクリック

(3) AXIHP\_MEMCPYのインスタスを追加できた

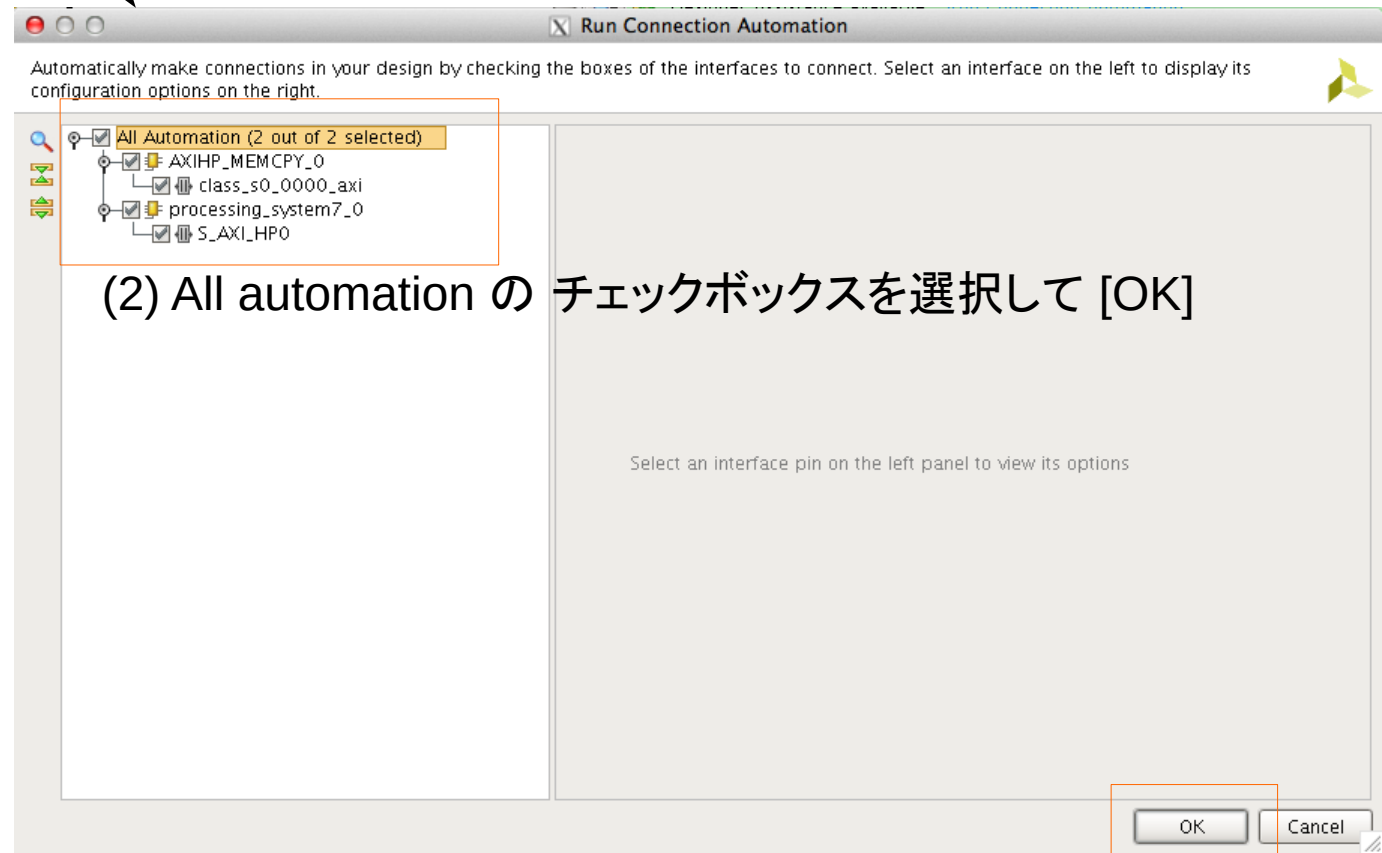


# AXIHP\_MEMCPYモジュールの追加(3.1)

## ポートの処理(AXIポートの自動接続)



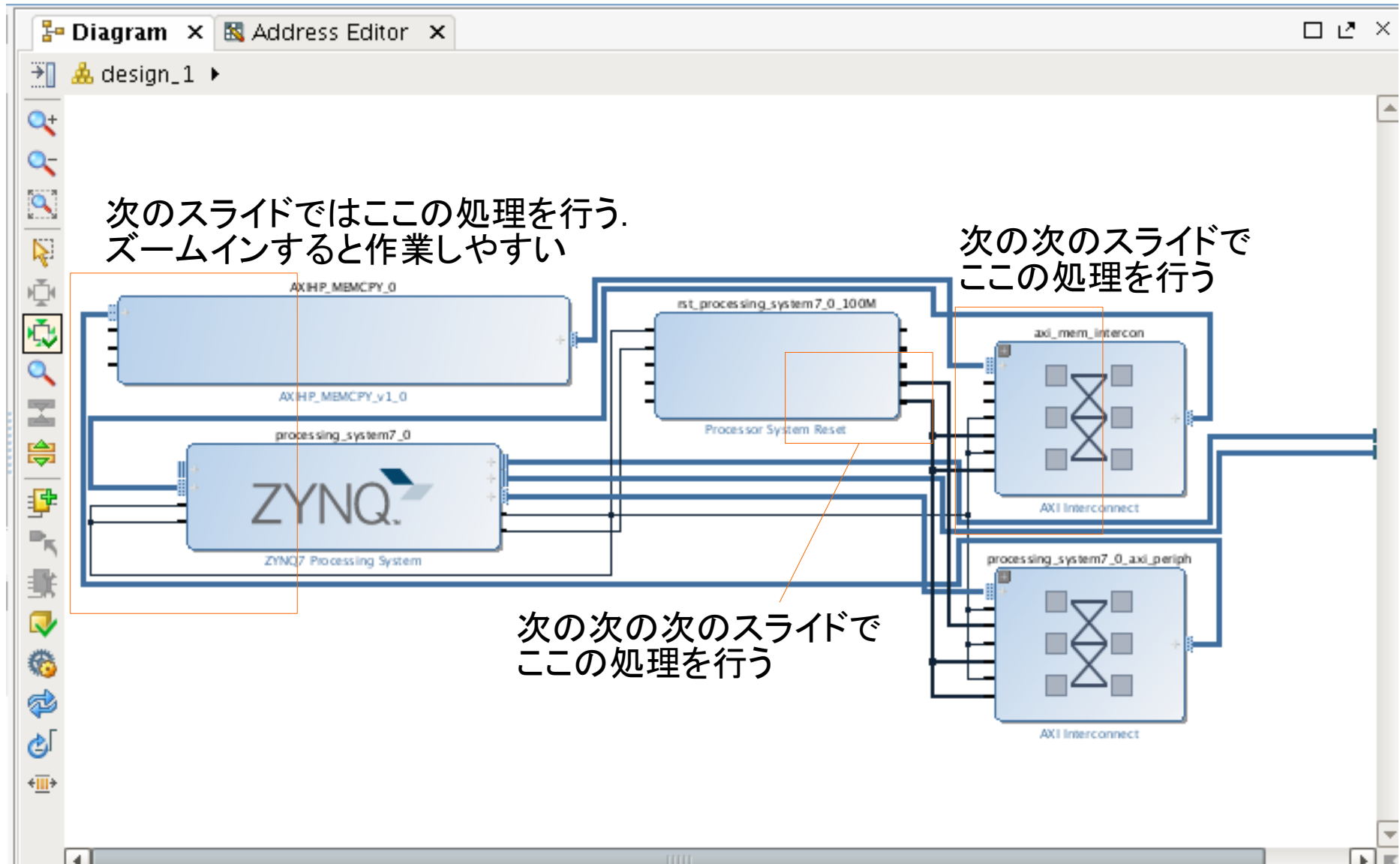
(1) Run Connection Automation を 選択



(2) All automation の チェックボックスを選択して [OK]

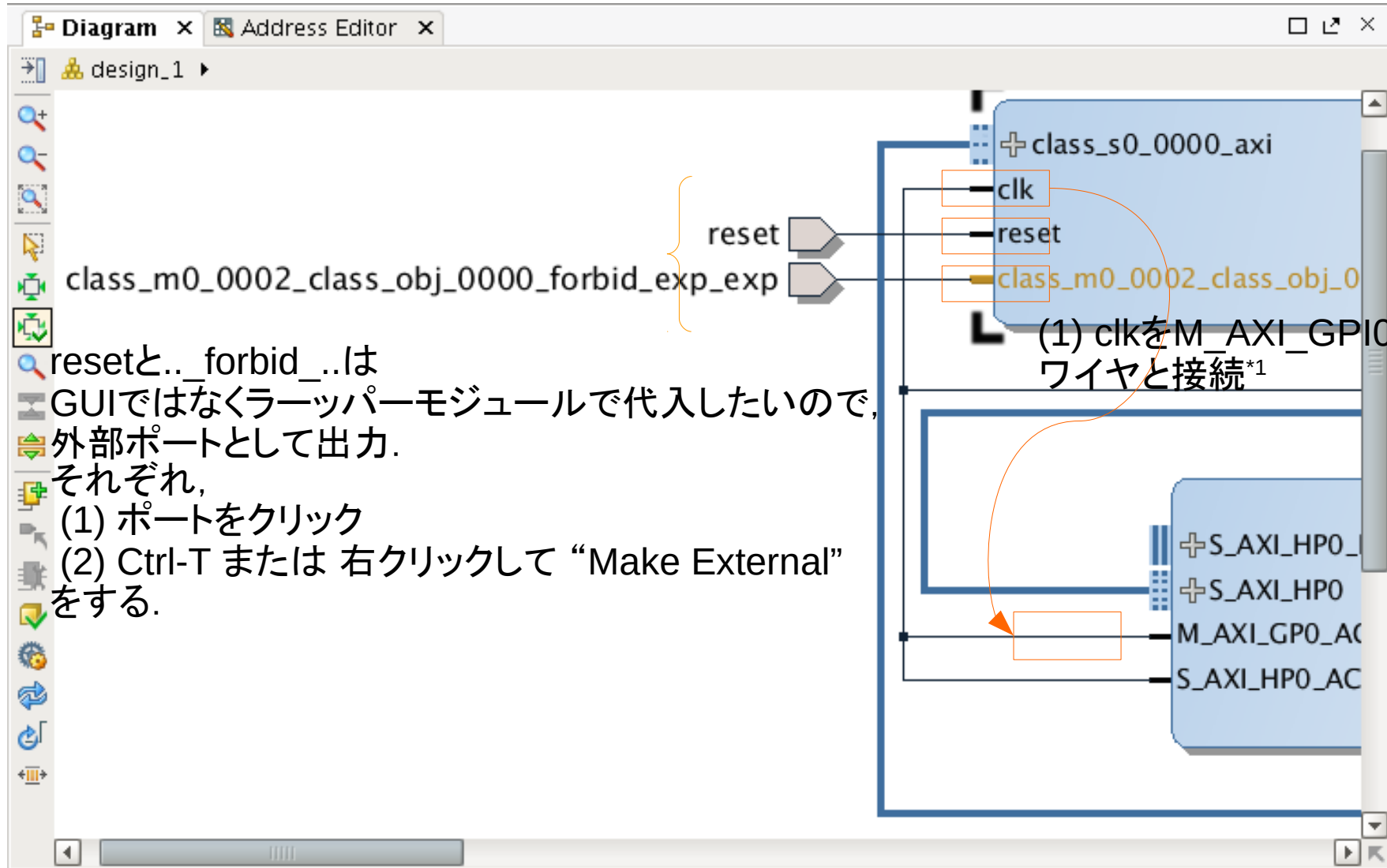
# AXIHP\_MEMCPYモジュールの追加(3.2)

## ポートの処理(AXIポートの処理が完了したところ)



## AXIHP\_MEMCPYモジュールの追加(3.3)

## ポートの処理(AXIHP\_MEMCPYのその他のポートの処理)

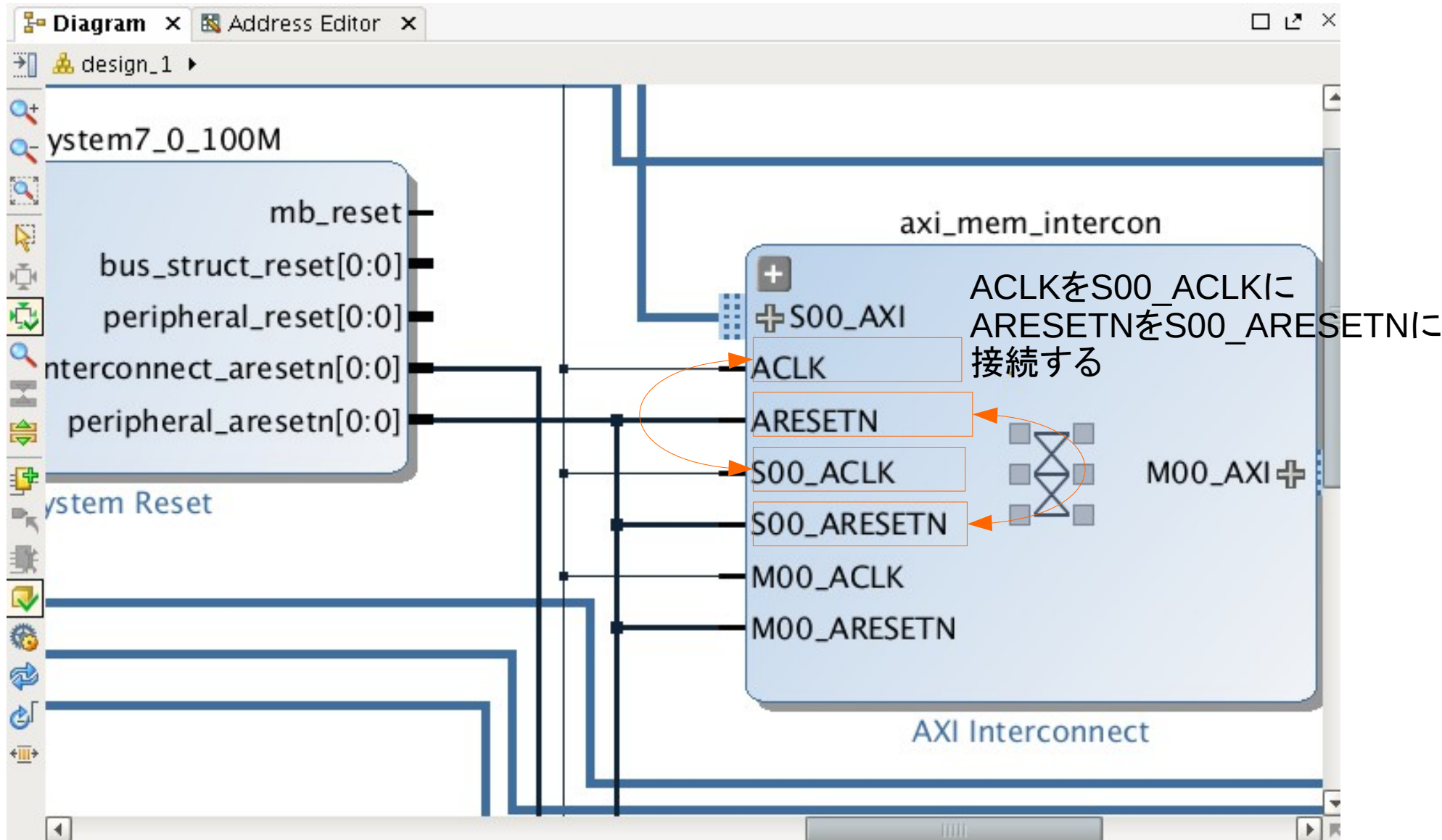


(1) clkをM\_AXI\_GPI0\_ACLKなどのワイヤと接続<sup>\*1</sup>

\*1 マウスでポートを選択. マウスカーソルが鉛筆状になった状態で接続元から接続先までドラッグ&ドロップするとよい

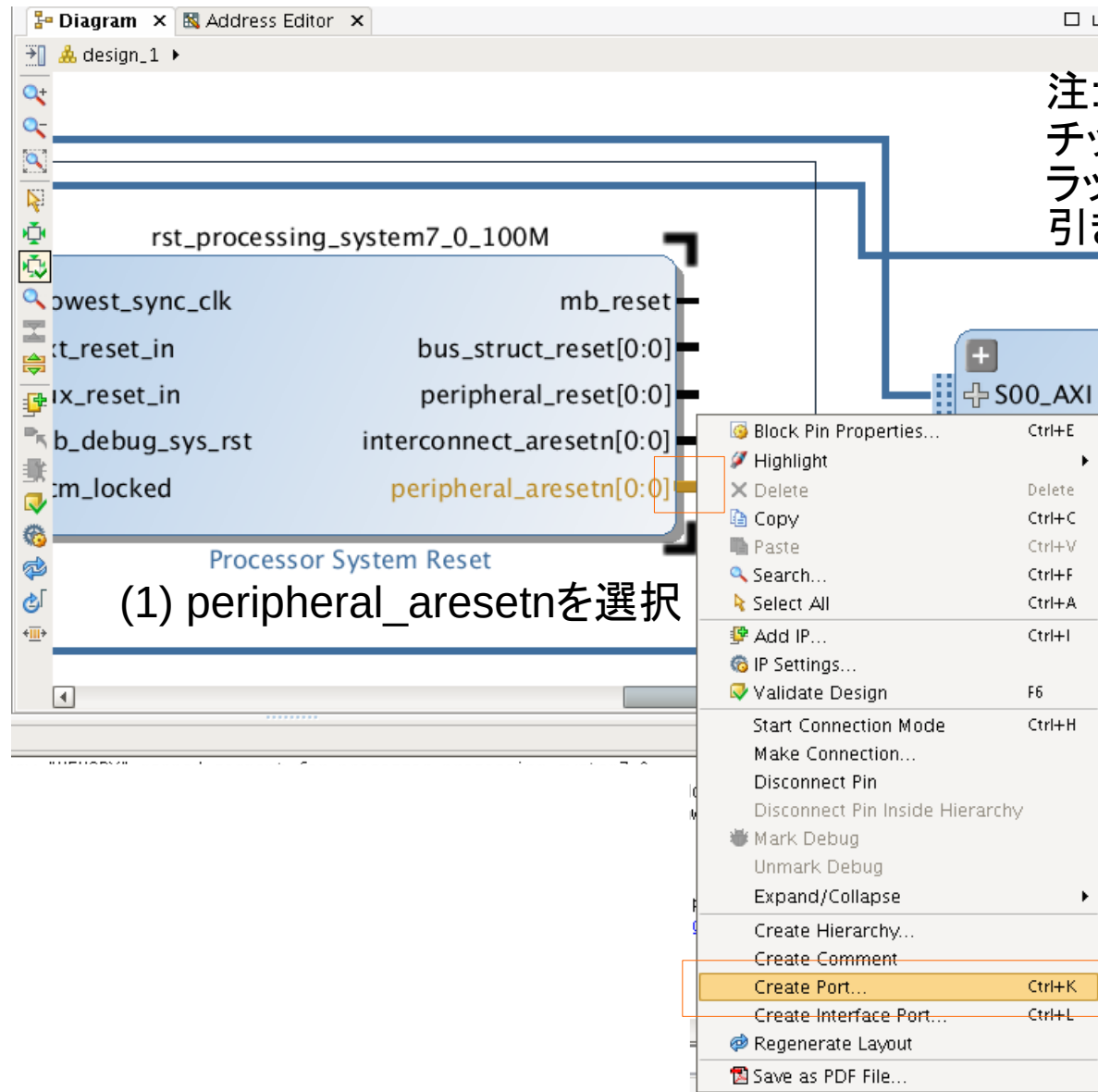
# AXIHP\_MEMCPYモジュールの追加(3.4)

## ポートの処理(axi\_inter\_memconのACLKとARESETNの処理)



# AXIHP\_MEMCPYモジュールの追加(3.5)

## ポートの処理(peripheral\_aresetnを外部に引き出す)



注: この信号は、本当にFPGAのチップ外に引き出したいわけではなくラッパーモジュールで扱うために引き出す。

(1) peripheral\_aresetnを選択

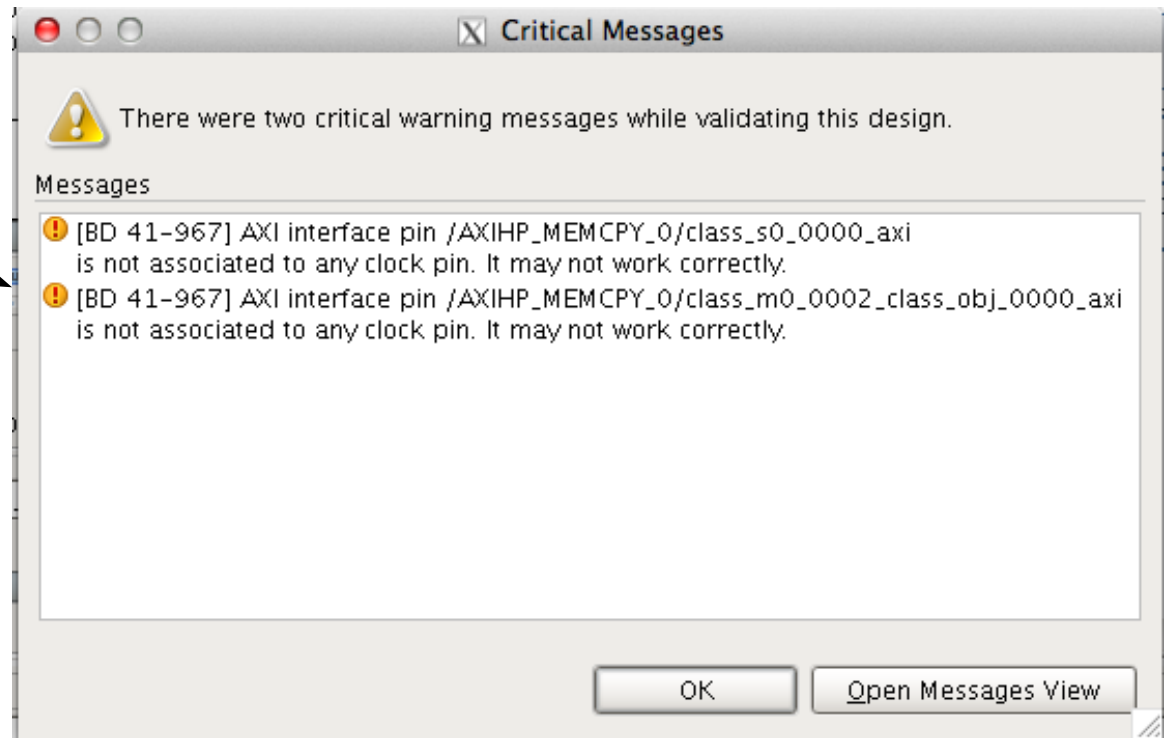
(2) 右クリックでメニューを開いて、Create Portを選択して生成(ダイアログはOKでよい)

# AXIHP\_MEMCPYモジュールの追加(3.6)

## ポートの処理(検証)



(1) Validate Designを選択



/AXIHP\_MEMCPY\_0/class\_s0\_0000\_axiと  
/AXIHP\_MEMCPY\_0/class\_m0\_0002\_class\_obj\_0000\_axiの  
クロックに関する警告がでる...のは現状想定範囲内なのでOK



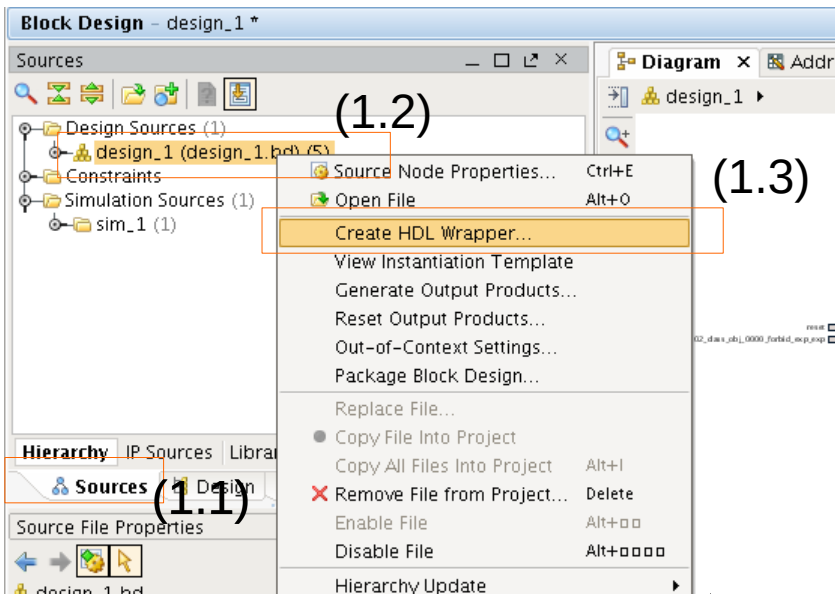
# HDLラッパーの生成と修正

## 作業の概要

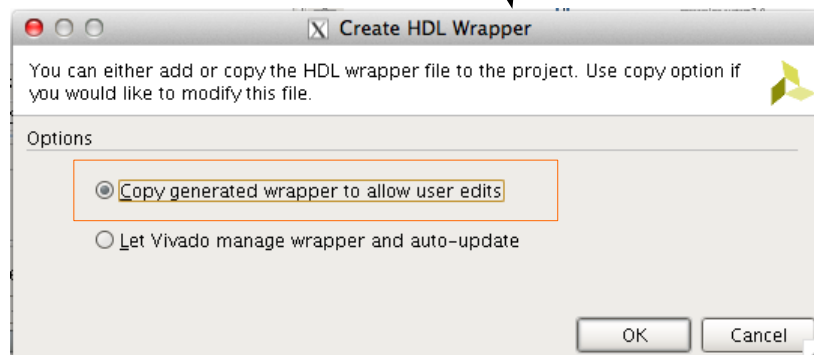
- ✓ Project SettingでTarget LanguageがVerilogなことを確認
  - ✓ VHDLの方が好きな人はVHDLでも良い. この資料ではVerilogで話を進める
- ✓ Sourcesタブのdesign\_1.bdからラッパーを生成
- ✓ resetとforbid信号の取り扱いを修正
  - ✓ ボードデザイン(GUIでの設計)では都合上ポートを作成したがチップ外部に引き出したいわけではない
  - ✓ resetには~peripheral\_aresetnを接続
  - ✓ ..\_forbid\_.. には 1'b0を与える
    - ✓ この信号1'b1を与えるとAXIアクセスを強制禁止できる
    - ✓ 今回は禁止\*しない\*ので即値で1'b0を指定

# HDLラッパーの生成と修正(1)

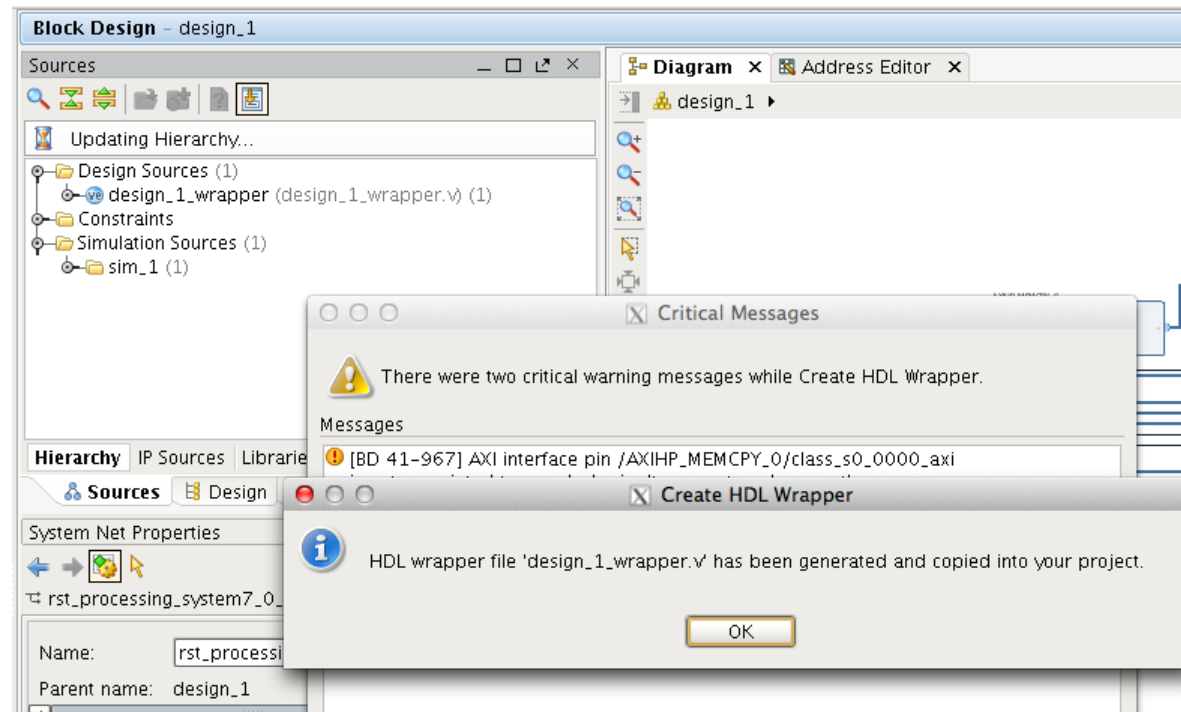
## HDLラッパーの生成



(1) design\_1.bdで右クリックして  
Create HDL Wrapper...を選択



(2) Copy generated... の方を選択する



(3) HDLラッパー  
(desgin\_1\_wrapper.v)が生成される

# HDLラッパーの生成と修正(2)

## HDLラッパーの修正

変更前

```
21 DDR_dq,  
22 DDR_dqs_n,  
23 DDR_dqs_p,  
24 DDR_odt,  
25 DDR_ras_n,  
26 DDR_reset_n,  
27 DDR_we_n,  
28 FIXED_IO_dds_vrn,  
29 FIXED_IO_dds_vrp,  
30 FIXED_IO_mio,  
31 FIXED_IO_ps_clk,  
32 FIXED_IO_ps_porb,  
33 FIXED_IO_ps_srstb,  
34 class_m0_0002_class_obj_0000_forbid_exp_exp,  
35 peripheral_aresetn,  
36 reset);  
37 inout [14:0]DDR_addr;  
38 inout [2:0]DDR_ba;  
39 inout DDR_cas_n;  
40 inout DDR_ck_n;  
41 inout DDR_ck_p;  
42 inout DDR_cke;  
43 inout DDR_cs_n;  
44 inout [3:0]DDR_dm;
```

変更後

```
21 DDR_dq,  
22 DDR_dqs_n,  
23 DDR_dqs_p,  
24 DDR_odt,  
25 DDR_ras_n,  
26 DDR_reset_n,  
27 DDR_we_n,  
28 FIXED_IO_dds_vrn,  
29 FIXED_IO_dds_vrp,  
30 FIXED_IO_mio,  
31 FIXED_IO_ps_clk,  
32 FIXED_IO_ps_porb,  
33 FIXED_IO_ps_srstb  
34 );  
35 inout [14:0]DDR_addr;  
36 inout [2:0]DDR_ba;  
37 inout DDR_cas_n;  
38 inout DDR_ck_n;  
39 inout DDR_ck_p;  
40 inout DDR_cke;  
41 inout DDR_cs_n;  
42 inout [3:0]DDR_dm;  
43 inout [31:0]DDR_dq;  
44 inout [3:0]DDR_dqs_n;
```

削除

メモ: GUIで生成した外部ポートはラッパーモジュールでは  
直接FPGA外部へ引き出されるポートになる。  
今回はデザイン内部で利用したいだけなので, FPGAの外には出さない。

特にコンマの扱いに注意。

# HDLラッパーの生成と修正(3)

## HDLラッパーの修正

### 変更前

```
Diagram x Address Editor x design_1_wrapper.v x
/home/miyo/synthesijer/work/project_1/project_1.srcs/sources_1,
35 inout [14:0]DDR_addr;
36 inout [2:0]DDR_ba;
37 inout DDR_cas_n;
38 inout DDR_ck_n;
39 inout DDR_ck_p;
40 inout DDR_cke;
41 inout DDR_cs_n;
42 inout [3:0]DDR_dm;
43 inout [31:0]DDR_dq;
44 inout [3:0]DDR_dqs_n;
45 inout [3:0]DDR_dqs_p;
46 inout DDR_odt;
47 inout DDR_ras_n;
48 inout DDR_reset_n;
49 inout DDR_we_n;
50 inout FIXED_IO_dds_vrn;
51 inout FIXED_IO_dds_vrp;
52 inout [53:0]FIXED_IO_mio;
53 inout FIXED_IO_ps_clk;
54 inout FIXED_IO_ps_porb;
55 inout FIXED_IO_ps_srstb;
56 input class_m0_0002_class_obj_0000_forbid_exp_exp;
57 output [0:0]peripheral_aresetn;
58 input reset;
59
```

削除

### 変更後

```
Diagram x Address Editor x design_1_wrapper.v * x
/home/miyo/synthesijer/work/project_1/project_1.srcs/sources_1,
35 inout [14:0]DDR_addr;
36 inout [2:0]DDR_ba;
37 inout DDR_cas_n;
38 inout DDR_ck_n;
39 inout DDR_ck_p;
40 inout DDR_cke;
41 inout DDR_cs_n;
42 inout [3:0]DDR_dm;
43 inout [31:0]DDR_dq;
44 inout [3:0]DDR_dqs_n;
45 inout [3:0]DDR_dqs_p;
46 inout DDR_odt;
47 inout DDR_ras_n;
48 inout DDR_reset_n;
49 inout DDR_we_n;
50 inout FIXED_IO_dds_vrn;
51 inout FIXED_IO_dds_vrp;
52 inout [53:0]FIXED_IO_mio;
53 inout FIXED_IO_ps_clk;
54 inout FIXED_IO_ps_porb;
55 inout FIXED_IO_ps_srstb;
56
57 wire [14:0]DDR_addr;
58 wire [2:0]DDR_ba;
59 wire DDR_cas_n;
```

メモ: GUIで生成した外部ポートはラッパーモジュールでは直接FPGA外部へ引き出されるポートになる。  
今回はデザイン内部で利用したいだけなので、FPGAの外には出さない。

# HDLラッパーの生成と修正(4)

## HDLラッパーの修正

- Javaで書いたモジュールにperipheral\_resetの極性を反転したものを与える
- forbid信号には1'b0(forbidしない, 常にAXIアクセスを有効にするの意)を設定。

### 変更前

```
Diagram x Address Editor x design_1_wrapper.v * x
/home/miyo/synthesijer/work/project_1/project_1.srcs/sources_1/
82 design_1 design_1_i
83     (.DDR_addr(DDR_addr),
84     .DDR_ba(DDR_ba),
85     .DDR_cas_n(DDR_cas_n),
86     .DDR_ck_n(DDR_ck_n),
87     .DDR_ck_p(DDR_ck_p),
88     .DDR_cke(DDR_cke),
89     .DDR_cs_n(DDR_cs_n),
90     .DDR_dm(DDR_dm),
91     .DDR_dq(DDR_dq),
92     .DDR_dqs_n(DDR_dqs_n),
93     .DDR_dqs_p(DDR_dqs_p),
94     .DDR_odt(DDR_odt),
95     .DDR_ras_n(DDR_ras_n),
96     .DDR_reset_n(DDR_reset_n),
97     .DDR_we_n(DDR_we_n),
98     .FIXED_IO_dds_vrn(FIXED_IO_dds_vrn),
99     .FIXED_IO_dds_vrp(FIXED_IO_dds_vrp),
100    .FIXED_IO_mio(FIXED_IO_mio),
101    .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
102    .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
103    .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb),
104    .class_m0_0002_class_obj_0000_forbid_exp_exp
105    .peripheral_aresetn(peripheral_aresetn),
106    .reset(reset));
107 endmodule
108
```

変更箇所

### 変更後

```
Diagram x Address Editor x design_1_wrapper.v * x
/home/miyo/synthesijer/work/project_1/project_1.srcs/sources_1/
82 design_1 design_1_i
83     (.DDR_addr(DDR_addr),
84     .DDR_ba(DDR_ba),
85     .DDR_cas_n(DDR_cas_n),
86     .DDR_ck_n(DDR_ck_n),
87     .DDR_ck_p(DDR_ck_p),
88     .DDR_cke(DDR_cke),
89     .DDR_cs_n(DDR_cs_n),
90     .DDR_dm(DDR_dm),
91     .DDR_dq(DDR_dq),
92     .DDR_dqs_n(DDR_dqs_n),
93     .DDR_dqs_p(DDR_dqs_p),
94     .DDR_odt(DDR_odt),
95     .DDR_ras_n(DDR_ras_n),
96     .DDR_reset_n(DDR_reset_n),
97     .DDR_we_n(DDR_we_n),
98     .FIXED_IO_dds_vrn(FIXED_IO_dds_vrn),
99     .FIXED_IO_dds_vrp(FIXED_IO_dds_vrp),
100    .FIXED_IO_mio(FIXED_IO_mio),
101    .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
102    .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
103    .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb),
104    .class_m0_0002_class_obj_0000_forbid_exp_exp
105    .peripheral_aresetn(peripheral_aresetn),
106    .reset(~peripheral_aresetn));
107 endmodule
108
```

# 合成

- ✓ Flow NavigatorのGenerate Bitstreamをクリックして合成
- ✓ 途中AXIHP\_MEMCPYのclkについて警告がでる
- ✓ 今回のケースではOKで続行

The screenshot displays the Vivado 2014.4 IDE interface during the synthesis process. On the left, the Flow Navigator shows the 'Generate Bitstream' step selected under the 'Implementation' section. The main workspace shows the 'Block Design' for 'design\_1', with a 'Sources' pane listing design files. A 'Project Settings' window is open, showing details like 'Project name: project\_1' and 'Product family: Zynq-7000'. A 'Bitstream Generation Completed' dialog box is centered, with 'View Reports' selected. The Log window at the bottom shows the synthesis progress, including warnings like 'WARNING: [Vivado\_Tcl 4-319] File design\_1\_wrapper.mmi does not exist'.

# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# ARM上のソフトウェア開発のための準備

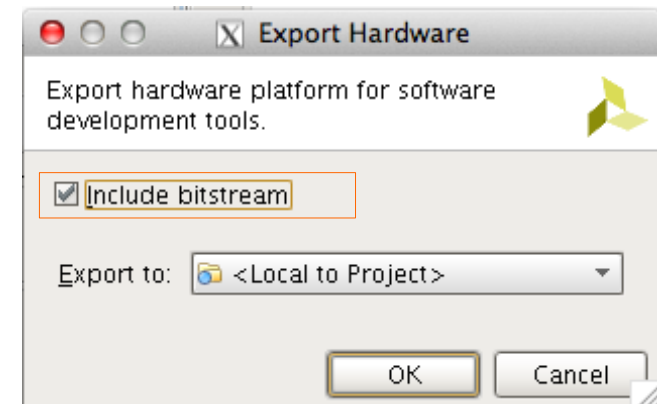
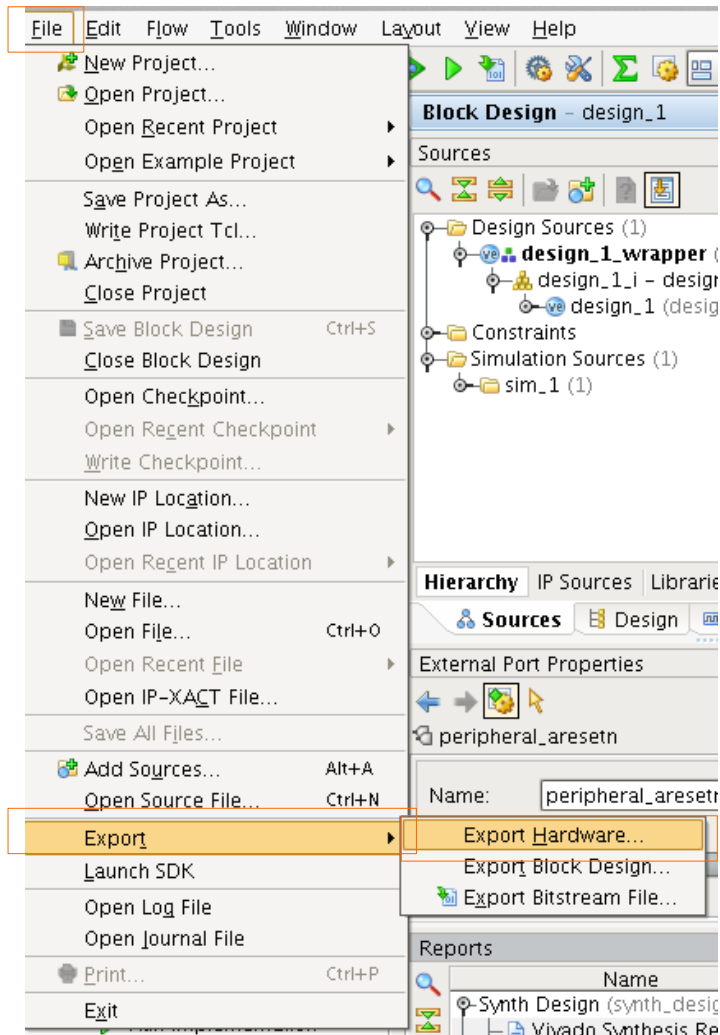
## 作業の概要

- ✓ ハードウェアプロジェクトのエクスポートとSDKの起動
- ✓ BSPの生成
- ✓ アプリケーションプロジェクトの生成
- ✓ Cソースファイルの生成



# エクスポートとSDKの起動(1)

## ハードウェアプロジェクトのエクスポート

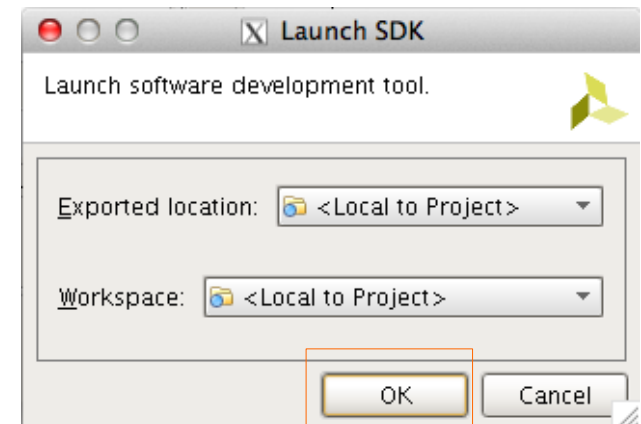
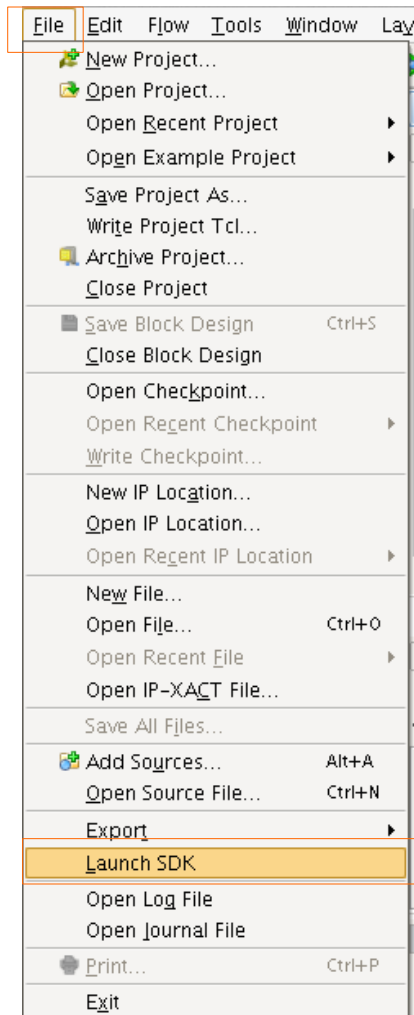


Include bitstreamのチェックボックスの  
チェックを入れて, [OK]

File → Export → Export Hardware... を選択

# エクスポートとSDKの起動(2)

## SDKの起動



そのまま[OK]

File → Launch SDK を選択

# BSPの生成～Cソースファイルの作成

✓ Zynq + Vivado HLS入門

慶應義塾大学 天野研究室 杉本 成

<http://www.slideshare.net/narusugimoto/zynq-vivado-hls>

の

p.117(SDK “Board Support Packageの生成1/4”)～

p.127(SDK “Fileの生成3/3”)を参照

# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# ARM上のソースコードの記述

- ✓ ソフトウェアの概要は

Zynq + Vivado HLS入門

慶應義塾大学 天野研究室 杉本 成

<http://www.slideshare.net/narusugimoto/zynq-vivado-hls>

の

p.128(SDK HLSコア制御アプリケーション雛形)～

p.136(SDK “axihs\_memcpyソフトウェア”)を参照

- ✓ レジスタ構成が若干違う
- ✓ 最終的なソースコードは次の通り

# ARM上のソースコードの記述

```
#include "xil_printf.h"

int main()
{
    Xil_DCacheDisable();
    int i, mismatch = 0;
    volatile unsigned int src_data[256], dst_data[256];
    for(i = 0; i < 256; i++) src_data[i] = i;
    unsigned int *baseaddr = (unsigned int*)0x43c00000;
    xil_printf("\r\n");
    baseaddr[1] = (unsigned int)src_data;
    baseaddr[2] = (unsigned int)dst_data;
    baseaddr[0] = 0xFFFFFFFF;
    xil_printf("memcpy start, src=%08x dest=%08x\n\r", src_data, dst_data);
    while(baseaddr[0] != 0) ;
    xil_printf("memcpy done\n\r");
    for(i = 0; i < 256; i++){
        xil_printf("src_data[%d] = %d, ", i, src_data[i]);
        xil_printf("dst_data[%d] = %d\n\r", i, dst_data[i]);
        if(src_data[i] != dst_data[i]) mismatch = 1;
    }
    (mismatch==0)? xil_printf("memcpy success!\n\r") :
                  xil_printf("memcpy fail\n\r");
    return 0;
}
```

Synthesijerで作ったコアへの  
パラメタ渡しと制御の開始

# 作業の手順

---

- ✓ Synthesijer関連 リソースの準備
- ✓ PL上のハードウェアの設計(Javaコードの記述とコンパイル)
- ✓ Vivadoでの合成
- ✓ ARM上のソフトウェア開発のための準備
- ✓ ARM上のソフトウェアの記述
- ✓ ソフトウェアのコンパイルと実行

# ソフトウェアのコンパイルと実行

- ✓ ソフトウェアのコンパイルと実行の概要は

Zynq + Vivado HLS入門

慶應義塾大学 天野研究室 杉本 成

<http://www.slideshare.net/narusugimoto/zynq-vivado-hls>

の

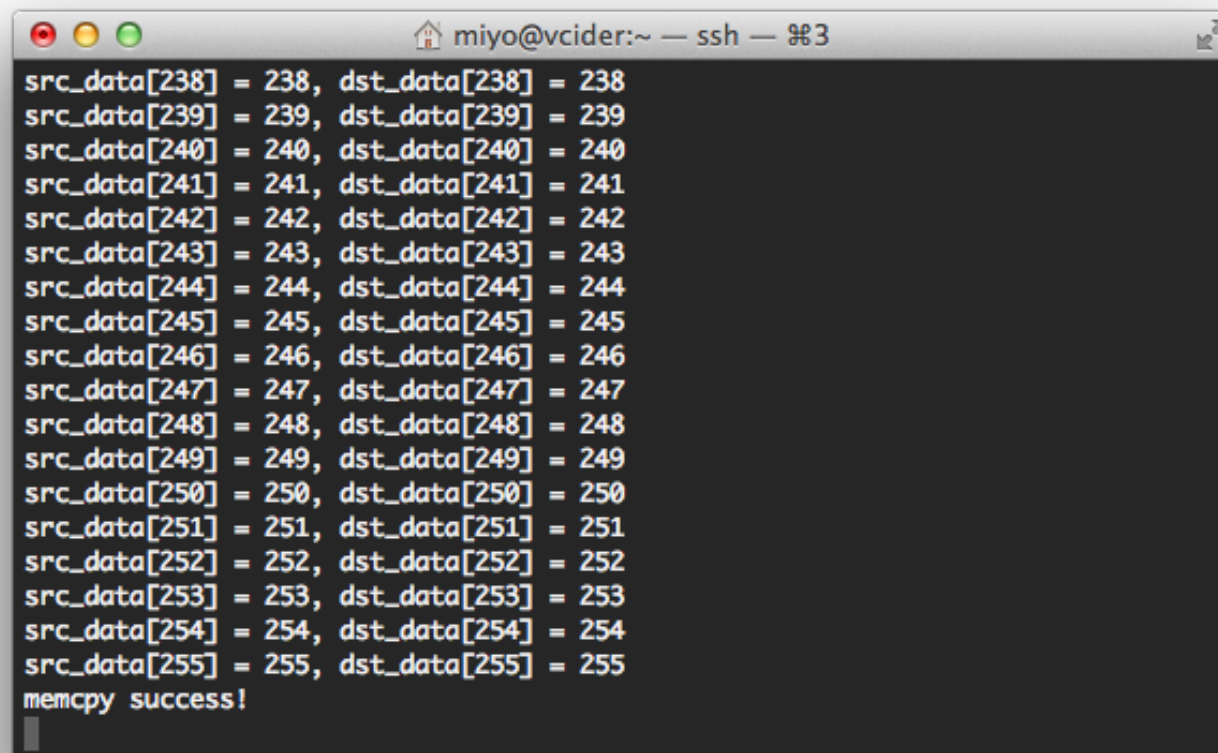
p.137(SDK “ソフトウェアのコンパイル”1/2)～

p.145(SDK “実行結果”)を参照

- ✓ bitstreamのパスはデフォルトで選択されるものを使用
- ✓ cuがなければscreenでもOK.
  - ✓ Zedboardなら: `screen /dev/ttyACM0 115200`
  - ✓ Zyboなら: `screen /dev/ttyUSB1 115200`



# 結果を確認



A terminal window titled "miyo@vcider:~ — ssh — ㊿3" displays the output of a memory copy verification process. The output consists of 18 lines, each showing a comparison between a source data element and a destination data element. The indices range from 238 to 255. All comparisons show identical values, indicating a successful copy. The final line of the output is "memcpy success!".

```
src_data[238] = 238, dst_data[238] = 238
src_data[239] = 239, dst_data[239] = 239
src_data[240] = 240, dst_data[240] = 240
src_data[241] = 241, dst_data[241] = 241
src_data[242] = 242, dst_data[242] = 242
src_data[243] = 243, dst_data[243] = 243
src_data[244] = 244, dst_data[244] = 244
src_data[245] = 245, dst_data[245] = 245
src_data[246] = 246, dst_data[246] = 246
src_data[247] = 247, dst_data[247] = 247
src_data[248] = 248, dst_data[248] = 248
src_data[249] = 249, dst_data[249] = 249
src_data[250] = 250, dst_data[250] = 250
src_data[251] = 251, dst_data[251] = 251
src_data[252] = 252, dst_data[252] = 252
src_data[253] = 253, dst_data[253] = 253
src_data[254] = 254, dst_data[254] = 254
src_data[255] = 255, dst_data[255] = 255
memcpy success!
```

# 今回のデザインへのエクスキューズ

- ✓ SimpleAXIMemIface32RTLTestは32bitのアクセスのたびに毎回AXIイベントを発行しています。より高速な転送のためには、バースト転送をする必要があるでしょう(次頁参照)。
- ✓ 割り込みについては、特に考えられていません。うまく扱えるようにしたいものです。
- ✓ BDに対する、CLK, RESETでのCritical Warningは気持ちが悪いです。なんとかしないといけません。

# Javaのコードを記述

## AXHP\_MEMCPY2.java を書く(バースト版)

```
import synthesijer.lib.axi.*;
import synthesijer.rt.*;

public class AXIHP_MEMCPY2{

    private final AXILiteSlave32RTL s0 = new AXILiteSlave32RTL();
    private final AXIMemIface32RTLTest m0 = new AXIMemIface32RTLTest();

    private void run(){
        int src_addr = s0.data[1];
        int dest_addr = s0.data[2];
        m0.fetch(src_addr, 256);
        m0.flush(dest_addr, 256);
    }

    @auto
    public void main(){
        s0.data[0] = 0x00000000;
        while(s0.data[0] == 0x00000000) ; // wait for kick from PS
        run();
        s0.data[0] = 0x00000000; // to notify DONE to PS
    }
}
```

Javaで1ワードずつコピーする  
わけではないので高速

# 補足1: CentOSのJava8のインストール

✓ たとえば

<http://qiita.com/hajimeni/items/67d9e9b0d169bf68d1c9>

を参考にするなどしてインストールしてください