

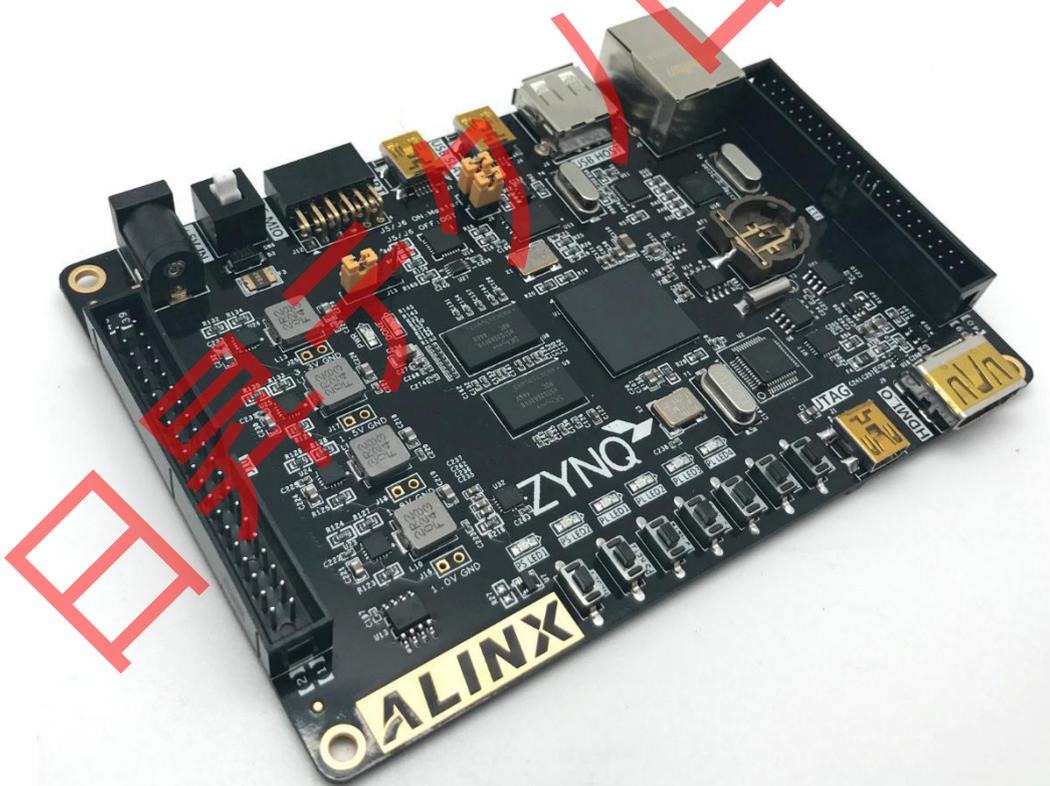
# ZYNQ XC7Z020 開発ボード マニュアル 基本編

株式会社日昇テクノロジー

<https://www.csun.co.jp>

[info@csun.co.jp](mailto:info@csun.co.jp)

作成日 2019/10/9



copyright@2019 - 2020

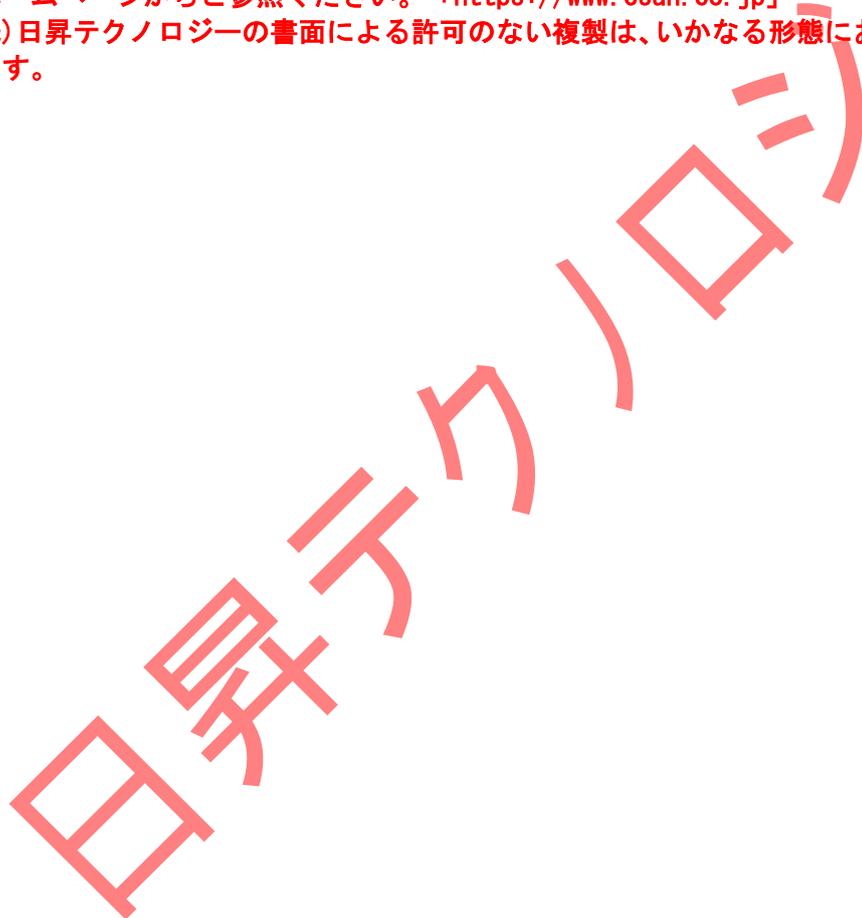


• 修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2019/10/9

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。「<https://www.csun.co.jp>」

※ (株)日昇テクノロジーの書面による許可のない複製は、いかなる形態においても厳重に禁じられています。



## 目録

第一章 パッケージソフトの説明と開発ボードのテスト .....	7
1.1 パッケージソフトの説明 .....	7
1.2 開発ボードテスト .....	7
1.1.2 テストに必要なもの .....	7
1.2.2 開発ボードのケーブル接続 .....	10
1.2.3 テストする .....	10
第二章 ZYNQ の紹介 .....	15
2.1 PS と PL の相互連結技術 .....	15
2.2 ZYNQ チップの開発流れの紹介 .....	22
2.3 ZYNQ の勉強はどんなスキルが必要 .....	24
第三章 Vivado 開発環境 .....	25
3.1 Vivado ソフトの紹介 .....	25
3.2 Vivado ソフトバージョン .....	25
3.3 Vivado ソフトの Windows でのインストール .....	25
第四章 PL の “Hello World” LED テスト .....	32
4.1 LED ハードウェアの紹介 .....	32
4.2 Vivado プロジェクトを作成 .....	33
4.3 Verilog HDL ファイルを作成し、LED を点灯する .....	38
4.4 ピン制約を追加 .....	42
4.5 シーケンス制約を追加 .....	46
4.6 BIT ファイルを生成する .....	50
4.7 ダウンロードとデバッグ .....	52
第五章 HDMI 輸出実験 .....	55
5.1 ハードウェアの紹介 .....	55
5.2 Vivado プロジェクトを作成する .....	55
5.2.1 HDMI コンパイラ IP コアを追加する .....	55
5.2.2 ピクセルクロックの PLL モジュールを追加する .....	60
5.2.3 カラーバー発生モジュールを追加する .....	62
5.2.4 トップモジュールを追加する .....	63
5.3 XDC 制約ファイルを追加する .....	64
5.4 ダウンロードとデバッグ .....	66
5.5 実験のまとめ .....	67
第六章 ARM を体験 .....	68
6.1 ハードウェアの紹介 .....	68
6.2 Vivado プロジェクトを作成 .....	68
6.2.1 UART 配置 .....	72
6.2.2 クロック配置 .....	72
6.2.3 DDR3 配置 .....	73
6.3 SDK デバッグ .....	80
6.4 実験のまとめ .....	93
6.5 よくある問題 .....	93
第七章 PS で PL の LED を点灯する .....	95
7.1 Vivado プロジェクトを作成する .....	95
7.1.1 UART 配置 .....	96

7.1.2	DDR3 配置	96
7.1.3	AXI GPIO を追加する	99
7.2	XDC ファイルで PL ピンを制約する	105
7.3	SDK プログラムをコンパイルする	106
7.4	ダウンロード及びデバッグ	110
7.5	実験のまとめ	112
第八章	PS タイマーインタラプト実験	113
8.1	Vivado プロジェクトを作成する	113
8.2	SDK をプログラミング	114
8.3	ダウンロードとデバッグ	118
8.4	実験纏め	119
第九章	PL キーインタラプト	120
9.1	Vivado プロジェクトを作成する	120
9.2	ダウンロードとデバッグ	124
9.3	実験まとめ	130
第十章	イーサネット実験 (LWIP)	131
10.1	Vivado プロジェクトを作成	131
10.1.1	PS 側のイーサネット配置	131
10.2	SDK プログラム	133
10.2.1	LWIP テンプレートに基づく APP を作成	133
10.3	ダウンロードとデバッグ	133
10.3.1	イーサネットテスト	133
10.4	実験のまとめ	136
第十一章	ユーザー定義 IP テスト	137
11.1	PWM 紹介	137
11.2	Vivado プロジェクトの作成	139
11.2.1	一つの vivado プロジェクトを作成	139
11.2.2	ユーザー定義 IP を作成する。	139
11.3	SDK ソフトのプログラミングとデバッグ	152
11.4	実験結果	161
11.5	よくある問題	161
11.5.1	AXI IP のベースアドレスを調べる	161
第十二章	VDMA を使用して HDMI ディスプレイを駆動する	162
12.1	Vivado プロジェクトの設立	162
12.1.1	UART のコンフィグ	165
12.1.2	I2C EMIO のコンフィグ	165
12.1.3	DDR3 のコンフィグ	166
12.1.4	コンフィグ割り込み	167
12.1.5	VDMA のコンフィグ	168
12.1.6	カスタム IP を追加する	171
12.1.7	HDMI エンコーダーを追加する	173
12.2	SDK ソフトウェアの作成とデバッグ	181
第十三章	プログラムの復帰	185
13.1	Vivado プロジェクトの設立	185
13.2	FSBL を生成する	187
13.3	BOOT ファイルを作成する	191
13.4	SD カードの起動テスト	195
13.5	QSPI テスト開始	196

13.6 Vivado のもとで QSPI を書き込む .....	197
13.7 バッチファイルを使用して QSPI をすばやく書き込む .....	200
第十四章 仮想マシンと Ubuntu システムをインストールする .....	202
14.1 仮想マシンソフトウェアのインストール .....	202
14.2 Ubuntu のインストール .....	202
14.2.1 システムのインストール .....	202
14.2.2 ソフトウェアソースサーバーを変更する .....	209
14.2.3 bash をデフォルトの sh に設定する .....	211
14.2.4 画面ロック時間を設定する .....	211
14.3 よくある問題 .....	212
14.3.1 仮想マシンには仮想化サポートが必要である。 .....	212
第十五章 Ubuntu で Linux バージョンの Vivado ソフトウェアをインストールする .....	213
15.1 Linux バージョンの Vivado をインストールする .....	213
15.2 許可設定 .....	218
15.3 ダウンローダードライバーをインストールする .....	218
15.4 Vivado をテストする .....	218
15.5 よくある問題 .....	220
15.5.1 Linux ダウンローダーのダウンロード時にプロンプトが表示される .....	220
15.5.2 ZYNQ に合うクロスコンパイラ .....	222
第十六章 Petalinux ツールのインストール .....	223
16.1 Petalinux の概要 .....	223
16.2 インストールに必要なライブラリ .....	223
16.3 Petalinux をインストールする .....	224
第十七章 NFS サービスソフトウェアのインストール .....	227
17.1 NFS サーバーをインストールする .....	227
17.2 NFS をテストする .....	228
17.3 よくある問題 .....	229
17.3.1 NFS マウントできない .....	229
第十八章 Petalinux で Linux システムをカスタマイズする .....	231
18.1 Vivado プロジェクト .....	231
18.2 Petalinux でプロジェクトを作成する .....	232
18.3 Linux カーネルをコンフィグする .....	238
18.4 ルートファイルシステムのコンフィグ .....	239
18.5 コンパイルする .....	240
18.6 BOOT ファイルが生成される .....	241
18.7 Linux をテストする .....	241
18.8 よくある問題 .....	243
18.8.1 Bad FIT kernel image format!が表示され、カーネルが起動できない。 ..	244
18.8.2 ファイルとコンフィグが保存できない .....	244
第十九章 SDK で Linux プログラムを開発する .....	245
19.1 SDK を使って Linux アプリケーションを作成する .....	245
19.2 NFS 共有を実行する .....	248
19.3 TCF-Agent を介してデバッグを実行する .....	250
19.4 TCF-Agent 問題 .....	253
第二十章 Linux 環境で GPIO 実験 .....	254
20.1 SHELL コントロールを使用する .....	254
20.2 C 言語を使ってコントロールする .....	255
20.2.1 GPIO のコードの確認 .....	257

---

20.2.2 物理 GPIO との関係の確認 .....	257
20.3 実験のまとめ .....	258
第二十一章 Petalinux での HDMI ディスプレイ .....	259
21.1 Petalinux のコンフィグ .....	259
21.2 Linux カーネルをコンフィグする .....	263
21.3 デバイスツリーを変更する .....	265
21.4 テスト petalinux プロジェクトのコンパイル.....	267
21.5 よくある問題 .....	269
21.5.1 システムのスリープを防ぐ方法.....	269
第二十二章 Debian デスクトップシステムの使用.....	270
22.1 Petalinux のコンフィグ .....	270
22.2 linux カーネルをコンフィグする .....	271
22.2.1 USB WIFI モジュールドライバをコンフィグする.....	272
22.2.2 USB カメラドライバをコンフィグする.....	273
22.3 Petalinux プロジェクトのコンパイルとテスト.....	273
22.4 SD カードファイルシステムを作成する.....	274
22.4.1 SD カードのパーティションを変更する。.....	274
22.4.2 ルートファイルシステムを SD カード EXT4 パーティションに同期する。..	279
第二十三章 QSPI Flash から起動の Linux の作成.....	283
23.1 Petalinux プロジェクトをコピーする.....	283
23.2 Petalinux のコンパイルとコンフィグ.....	284

日昇テクノロジー

## 第一章 パッケージソフトの説明と開発ボードのテスト

パッケージソフトウェアに主に含むものを紹介する。

### 1.1 パッケージソフトの説明

- 1) GP210x\_Windows\_Drivers.zip シリアルドライバー
- 2) **Xilinx\_Vivado\_SDK\_2017.4\_1216\_1.tar.gz** Vivado 2017.4 インストールパッケージ  
これは Windows と Linux が通用するバージョン。Windows を使う場合、解凍ソフトが必要になる
- 3) **petalinux-v2017.4-final-installer.run** petalinux インストールパッケージ
- 4) qt-opensource-windows-x86-mingw530-5.7.1.exe Windows バージョン Qt
- 5) qt-opensource-linux-x64-5.7.1.run Linux バージョン Qt
- 6) imageUSB.exe ミラーリカバリツール
- 7) 00\_resource 中に **Linux のソースコード、ルートファイルシステム**が含まれている
- 8) VMware-workstation-full-12.1.1-3770994.exe バーチャルマシンインストールパッケージ
- 9) ubuntu-16.04.3-desktop-amd64.iso **Ubuntu のインストールパッケージ、PC だけにインストールできる。開発ボードにはインストールできない**

### 1.2 開発ボードテスト

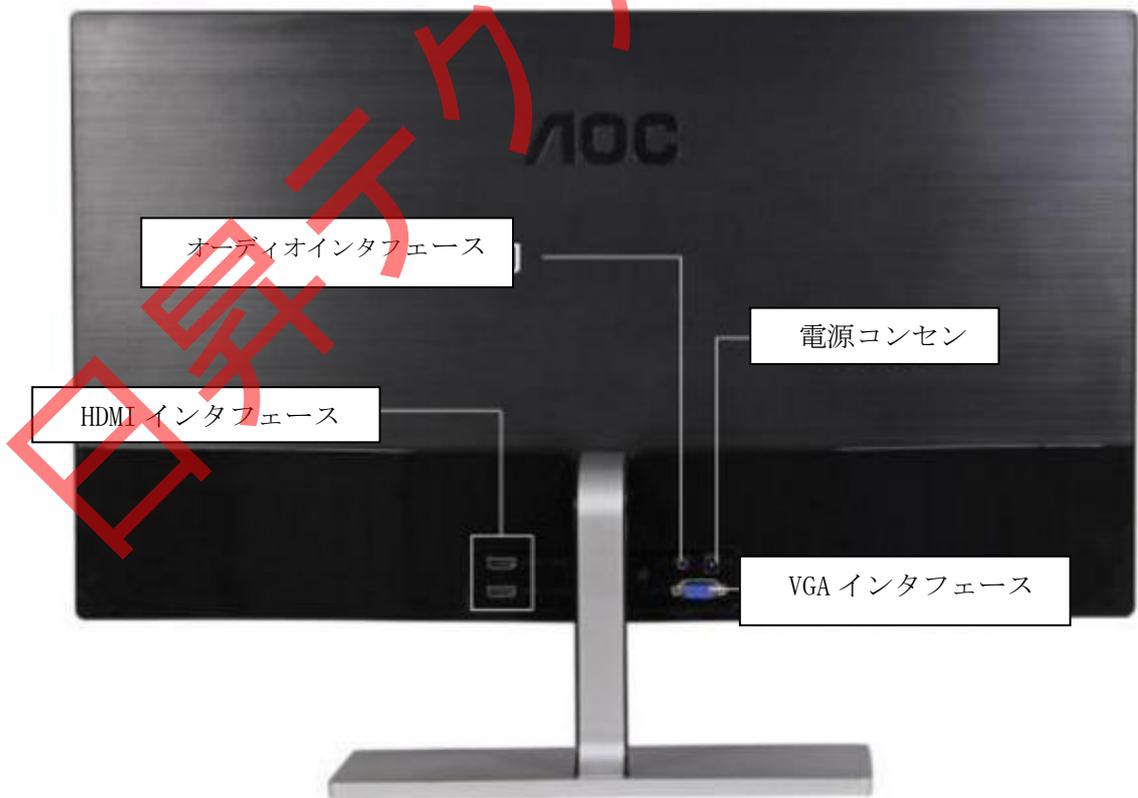
開発ボードを手に入れ、正常に働けるかどうかをテストする。これから、簡単に行えるテストを紹介する。

#### 1.1.2 テストに必要なもの

- 1) コンピューター



2) 解像度は 1920×1080 より高い、HDMI をサポートするモニターをもう一台を用意してください。電源なしで HDMI から VGA コンバーターを変換することは開発ボードがサポートできないので、給電が独立なコンバーター需要である。



3) HDMI ケーブル一つ

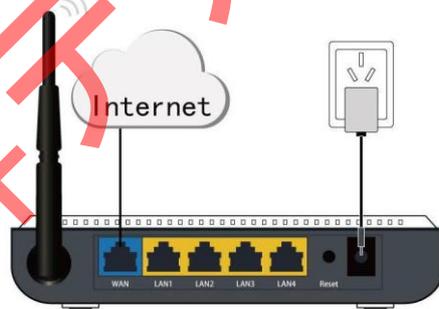


#### 4) USB ポートのマウスとキーボード



#### 5) ルーター

ネットワークをテストするため、インターネットを繋いだほうがいい。DHCP をサポートする。

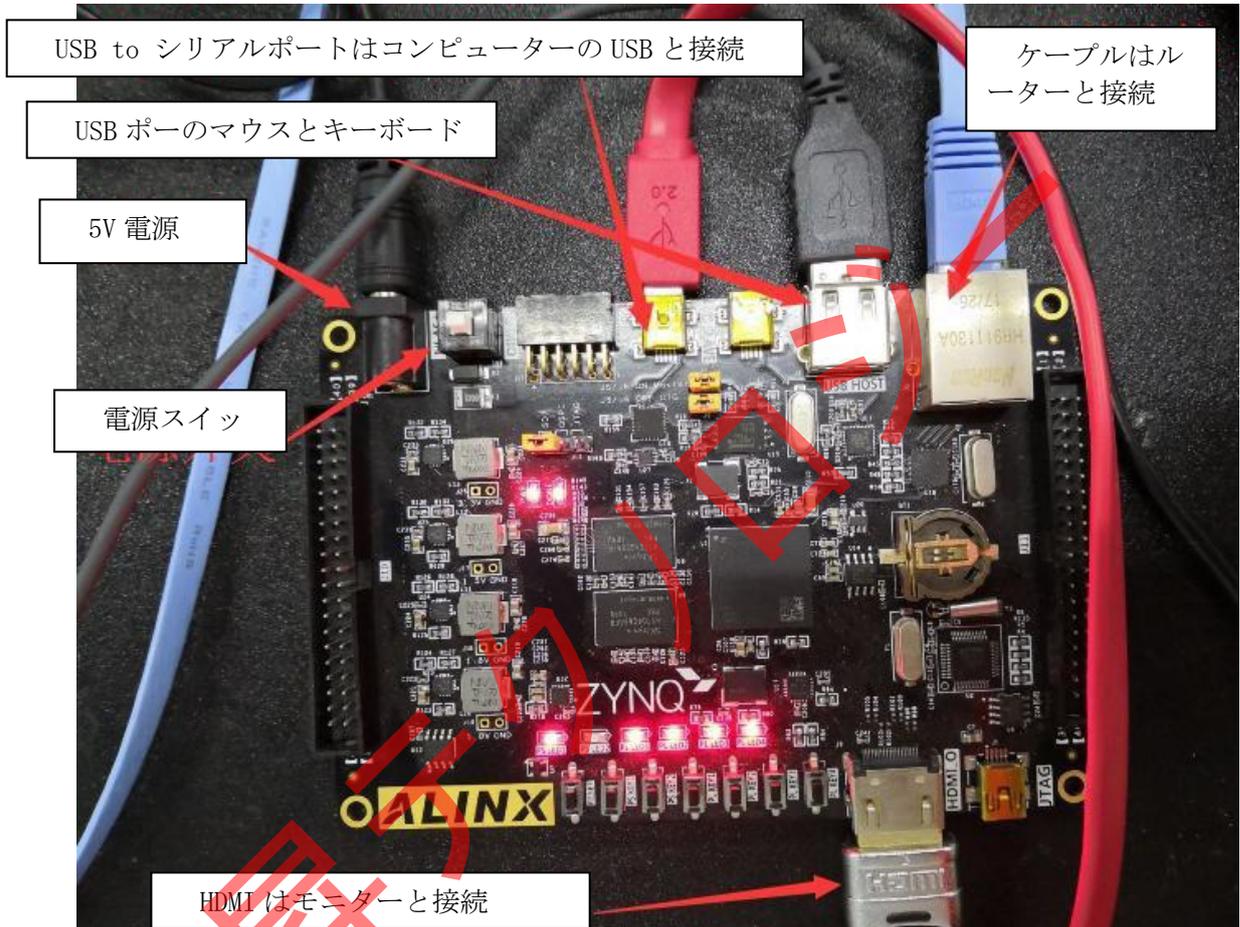


#### 6) LAN ケーブル



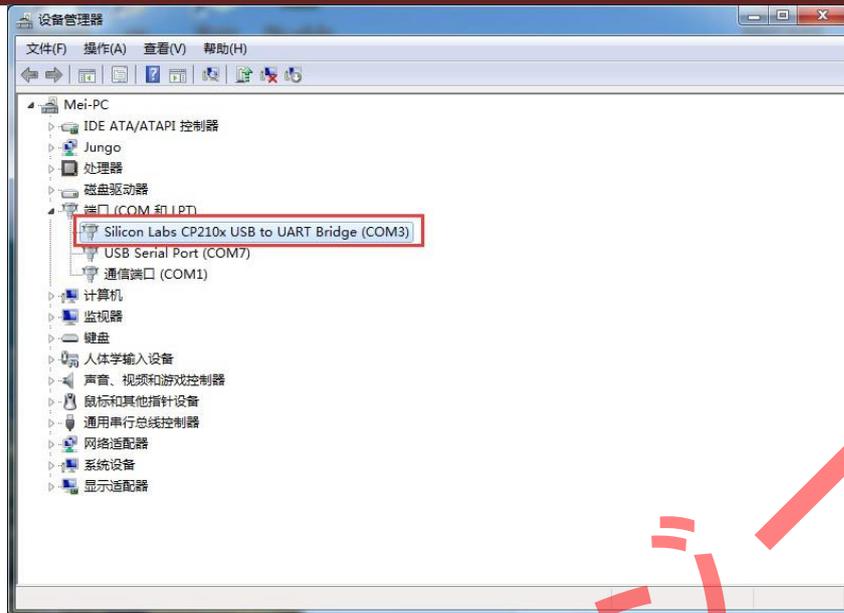
## 1.2.2 開発ボードのケーブル接続

- 1) HDMI モニターと接続
- 2) LAN ポートはルーターと接続する
- 3) 電源を接続



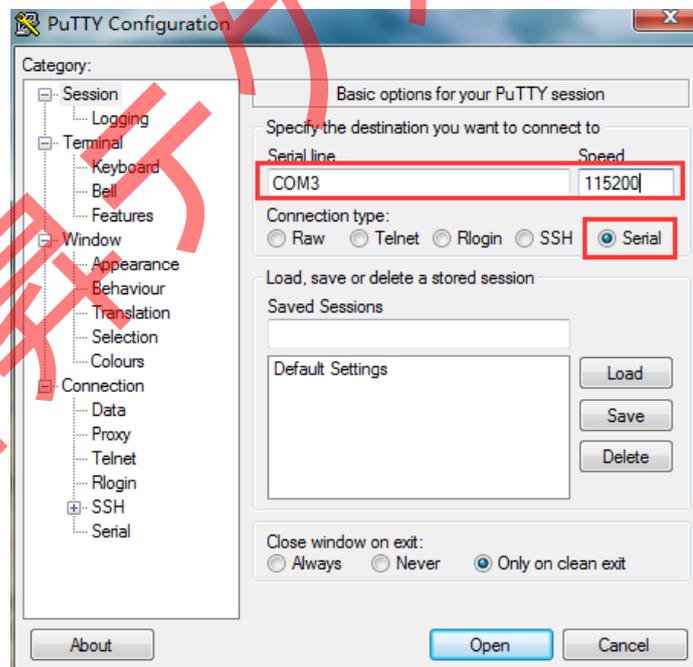
## 1.2.3 テストする

- 1) テスト前、USB からシリアルポートへの変換のドライバソフト(ソフト/CP210x\_Windows\_Driver\_zip) をインストールください。そうしなければ、シリアル通信テストができない。インストール完了後、USB ケーブルを使い、コンピューターの USB ポートと開発ボードの UART ポート (J1) をコネクしてください。これから、コンピューターのデバイスマネージャを開き。そこでシリアル設備 CP210x がみつけれられる。サンプルでは COM3。

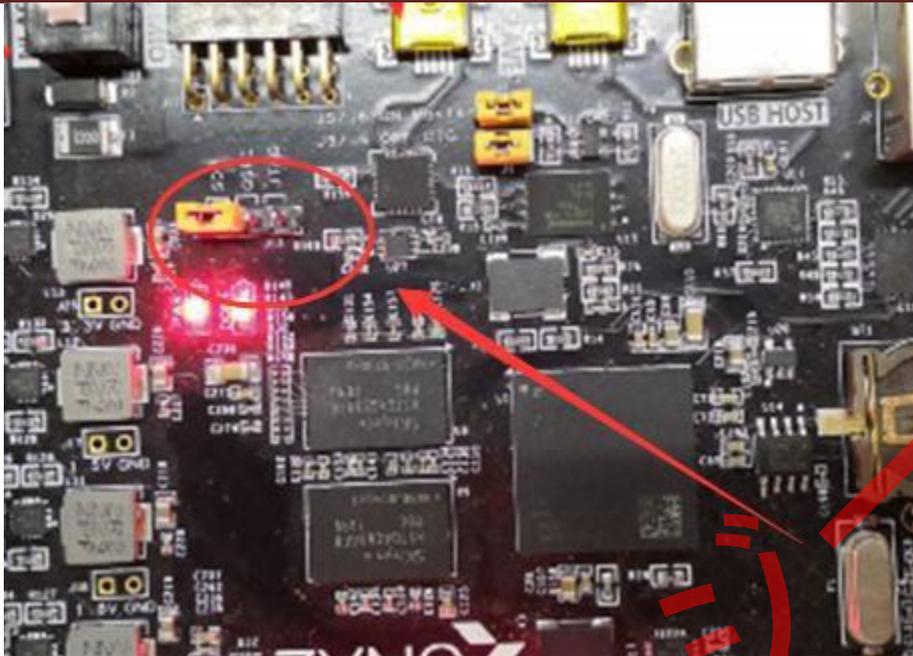


2) ターミナルツールはたくさんある。例えば、putty、tetaterm、Windows 内部のターミナルツールと SecureCRT などである。ここでは putty を利用する。

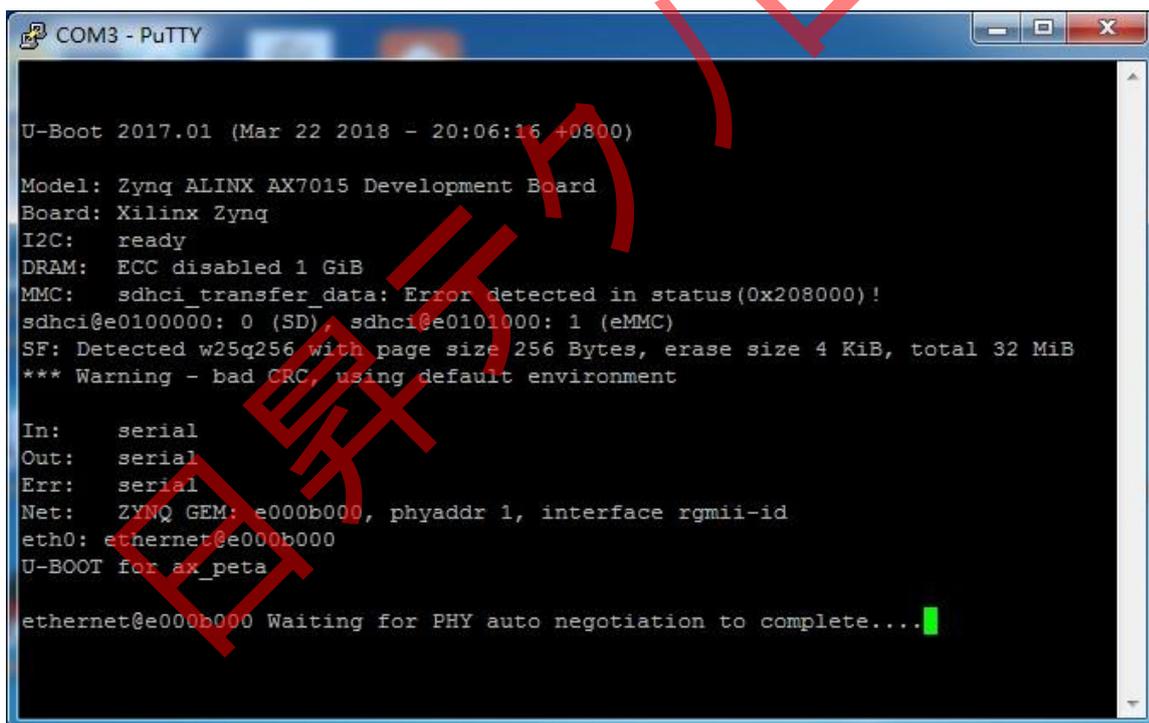
3) Serial と Serial line で COM3 を選んで、スピードは 115200 を書いてください。COM3 のシリアルナンバーはデバイスマネージャで表示されたものと同じく書いて、“Open”をクリックしてください。



4) 開発ボードの起動モードは SD モードかどうかを確認する（出荷時、デフォルトで開発ボードのカードスロットにカードがある、スタートモードも SD モードがデフォルト値になっている）、ジャンパーでスタートモードを変更できる。



5) 開発ボードの電源スイッチを入れて、putty の画面では u-boot と Linux システムのスタートメッセージが表示される。



```

COM3 - PuTTY
U-Boot 2017.01 (Mar 22 2018 - 20:06:16 +0800)

Model: Zynq ALINX AX7015 Development Board
Board: Xilinx Zynq
I2C:   ready
DRAM:  ECC disabled 1 GiB
MMC:   sdhci transfer data: Error detected in status(0x208000)!
sdhci@e0100000: 0 (SD), sdhci@e0101000: 1 (eMMC)
SF: Detected w25q256 with page size 256 Bytes, erase size 4 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   ZYNQ GEM: e000b000, phyaddr 1, interface rgmii-id
eth0: ethernet@e000b000
U-BOOT for ax_peta

ethernet@e000b000 Waiting for PHY auto negotiation to complete....
  
```

6) シリアルターミナルでシステムに登録できる。ユーザー : root、パスワード : root

大勢な人は putty を使用するのが初めて、あるいは、初めてシリアルを使うのである。説明しなければならぬのは、putty でコマンドの入力はホストキーボードで実現する。開発ボードと接続しているキーボードではない。

```
COM3 - PuTTY
[ OK ] Started Authenticate and Authorize Users to Run Privileged Tasks.
[ OK ] Started Network Manager.
[ OK ] Reached target Multi-User System.
net eth1: Promiscuous mode disabled.
xilinx_axienet 41000000.ethernet: of_phy_connect() failed
net eth1: Promiscuous mode disabled.
[ OK ] Reached target Graphical Interface.
Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.

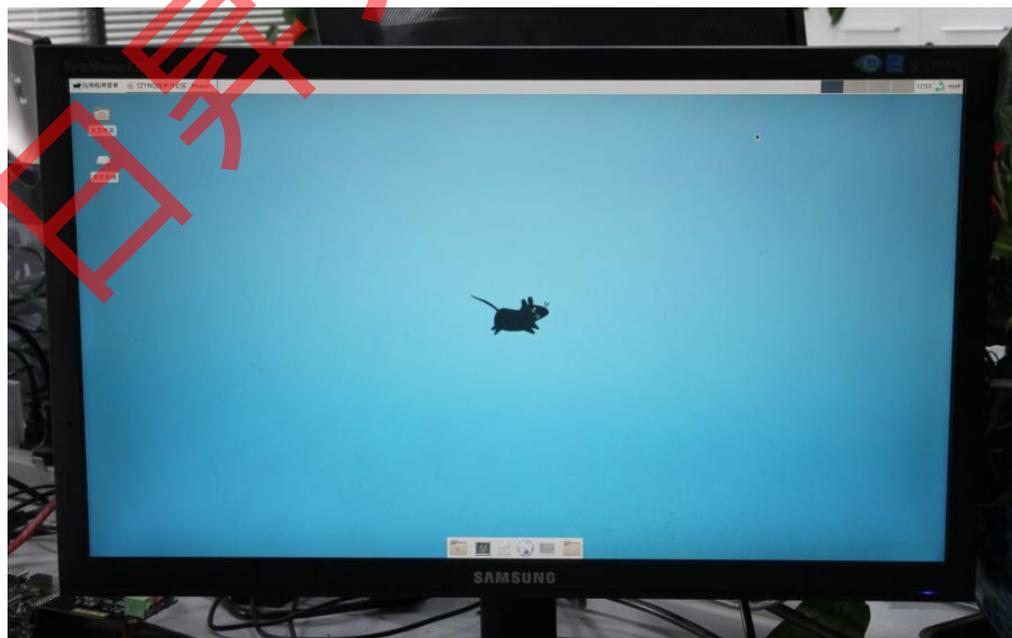
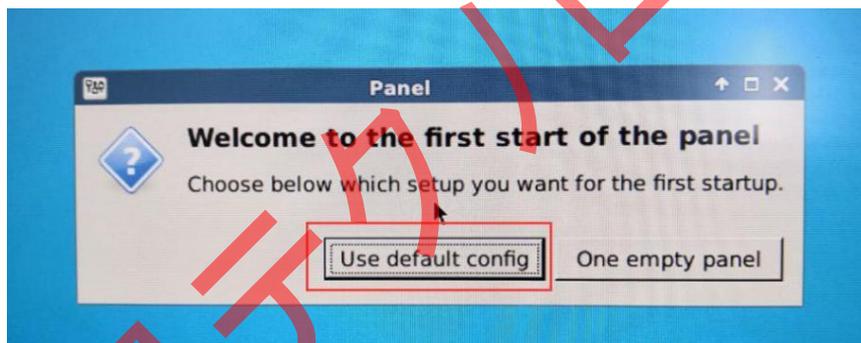
Debian GNU/Linux 8 zynq ttyPS0

zynq login: root
密码:
上一次登录: 四 1月 1 08:00:21 CST 1970ttyPS0 上
Linux zynq 4.9.0-xilinx #4 SMP PREEMPT Thu Mar 22 17:49:48 CST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@zynq:~#
```

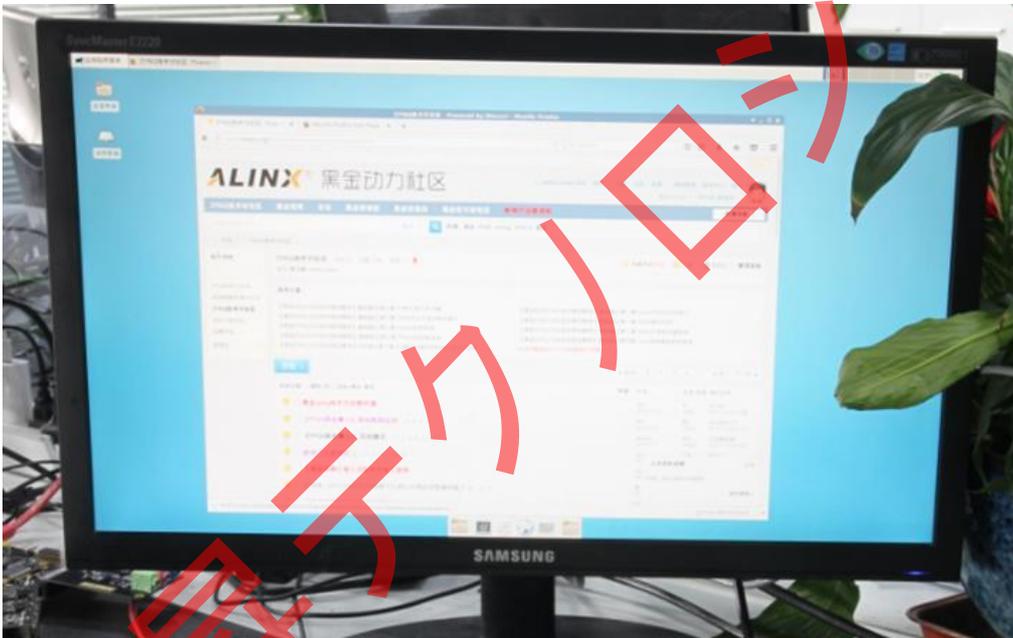
7) 起動した後、開発ボードと接続している HDMI モニターは Debian のデスクトップが表示される。パネルを選択するの画面が出る場合、デフォルトパネルを選んでいい。



- 8) ここに来て、もうマウスとキーボードを使って、操作出来る。マウスで Web ブラウザをダブルクリックしてください。ブラウザの起動は時間がかかりますので、しななくお待ちください。



- 9) アドレス欄で URL を入力してください。正常に開けば、開発ボードはもう普通に使える。

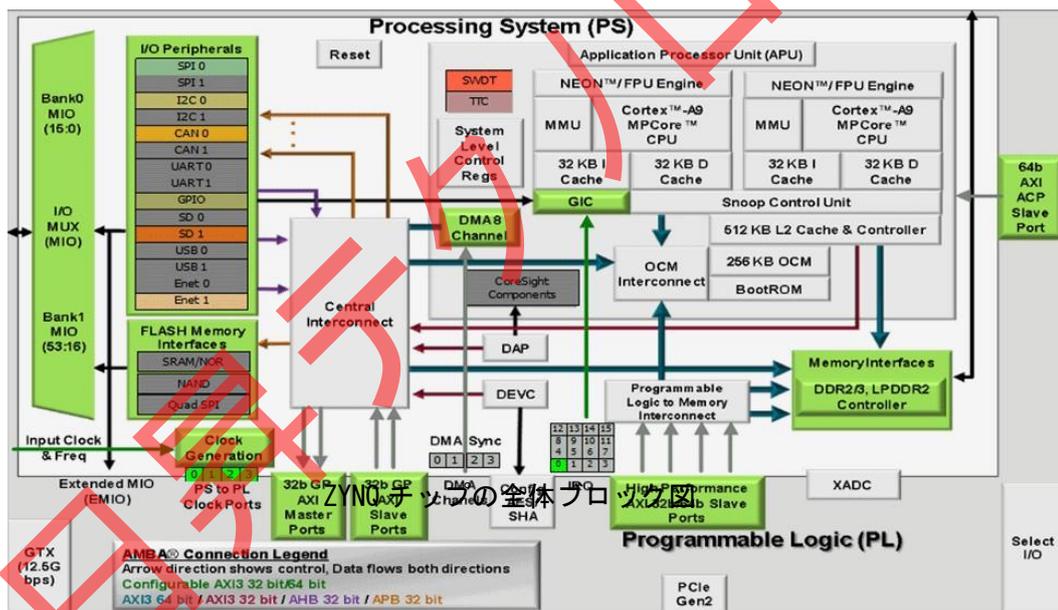


- 10) 開発ボードの簡易テストはここで終了する。

## 第二章 ZYNQ の紹介

Zynq シリーズの特長は FPGA にある完璧な ARM サブシステム (PS) である。各 Zynq システムのプロセッサは Cortex-A9 プロセッサが含まれていて、プロセッサの構造は全部これを中心としている。そして、サブシステムにメモリーコントローラと大量なペリフェラル統合されていて、Cortex-A9 のコアが Zynq-7000 内でプログラマブルロジックユニットから独立できるようになった。つまり、しばらくの間で、プログラマブルロジックユニット (PL) を使っていなければ、ARM プロセッサのサブシステムも独立で働ける。これは以前の FPGA と本質的な違いがあり、プロセッサを中心にしていきます。

Zynq は PS 部分と PL 部分、二つの機能ブロックである。はっきり言うと、ARM の SOC 部分と、FPGA 部分である。その中に、PS は二つの ARM Cortex™-A9 プロセッサ、AMBA®の相互連結、内部メモリ、外部メモリとペリフェラルを統合している。ペリフェラルの方は主に USB バスインターフェース、イーサネットインターフェース、SD/SDIO インターフェース、I2C バスインターフェース、CAN インターフェース、UART インターフェースと GPIO などを含んでいる。



PS : プロセッシングシステム (Processing System)、FPGA と無関連の ARM の SOC 部分。

PL : プログラマブルロジック、FPGA 部分。

### 2.1 PS と PL の相互連結技術

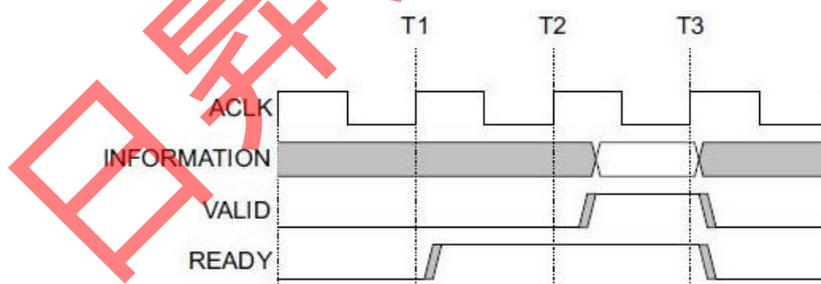
ZYNQ は高性能 ARM Cortex-A9 シリーズと高性能 FPGA をシングルチップで厳密に結びつける製品である。ARM プロセッサと FPGA との高速通信とデータ交換を実現し、両方の性能優位を発揮するため、効率が非常に高いチップ内高性能プロセッサと FPGA の連絡通路が必要である。そのため、効率が非常に高い PL と PS のデータ交換通路を設計することは ZYNQ チップの設計で最も重要なことと同時に、

製品の設計が成功できるかどうかに関わっている。この節、弊社が紹介する内容は主に PS と PL の連結で、ユーザーに連結技術を了解させる。

実は、具体的な設計には、連結での工夫はそんなに多くない。IP コアを加わったあと、システムは自動的に AXI インタフェースを使って、弊社の IP コアとプロセッサを連結する。あとは少し補充すればいい。

AXI のフルネームは Advanced eXtensible Interface で、Xilinx がシリーズ 6 の FPGA から導入するインタフェースプロトコルである。主にメインデバイスとサブデバイスのデータエントリー方法を説明した。ZYNQ で使っているバージョンは AXI4 であるから、AIX4.0 と ZYNQ の内部デバイスに AXI インタフェースがあることをよく見られる。実際に、AXI は ARM 会社が提出していた AMBA (Advanced Microcontroller Architecture) の一部で、高性能、高帯域幅とローディレイのチップインタナルバスである。これも前の AHB と APB バスの代わりになっている。初代バージョン AXI (AXI3) は 2003 でリリースされた AMBA3.0 に、二代の AXI (AXI4) も 2010 年リリースされた AMBA4.0 に含まれてる。

AXI プロトコルは主にメインデバイスとサブデバイスのデータエントリー方法を説明しました。両者はハンドシェーキングシグナルを利用して連結する。サブデバイスがデータの受け入れ準備が出来た時、READY 信号を出す。メインデバイスのデータが用意した時、VALID を維持する信号を出して、データ有効を示している。VALID と READY 信号両方も有効の時だけ、データは伝送を始まる。この二つも信号が有効のままだったら、メインデバイスは次のデータを伝送する。両方のどちらかが信号を取り消したら、伝送は停止になる。AXI のプロトコルは図に表したように、T2 のときは、サブデバイスの READY 信号が有効となり、同じく T3 のときはメインデバイスの VALID 信号が有効となる。データ伝送も始まる。



AXI ハンドシェーキングシグシエンス図

ZYNQ の中で、AXI-Lite、AXI4 と AXI-Stream 三つのバスをサポート出来る。表 5-1 から、この三つの AXI インタフェースの特性が見られる。

インタフェース プロトコル	特性	使用状況

AXI-Lite	アドレス/シングルデータを転送	低速ペリフェラルやコントロール
AXI4	アドレス/バーストデータを伝送	アドレスのロット伝送
AXI-Stream	データ伝送とバースト転送	データフローとメディア伝送

#### AXI4-Lite :

軽量級、構造簡単の特徴があって、小ロットデータや簡単なコントロールの状況に適切である。ロット伝送はサポートしない。読み込みと書き込みをするとき、一回で文字ひとつしかリードライトできない。主に一部の低速ペリフェラルを訪問とペリフェラルのコントロールに使う。

#### AXI4 :

インタフェースは AXI-Lite とだいたい同じである。ただ一つの機能、ロット伝送を増えた。この機能は連続でひとつのエリアのアドレスを一気にリードライトができる。つまり、データリードライトの burst 機能。

以上の二つはメモリマッピングでコントロールしてる。すなわち、ARM はユーザーデファイン IP をあるアドレスに編入して訪問する。リードライトの時もチップ内の RAM にしているように、プログラミングも便利で、開発も難しくない。それを代わりに、資源を取り過ぎで、他のアドレス読み出しコード、アドレス書き込みコード、データ読み出しコード、データ書き出しコード、リプライ書き込みコードなどの信号線が必要である。

#### AXI4-Stream :

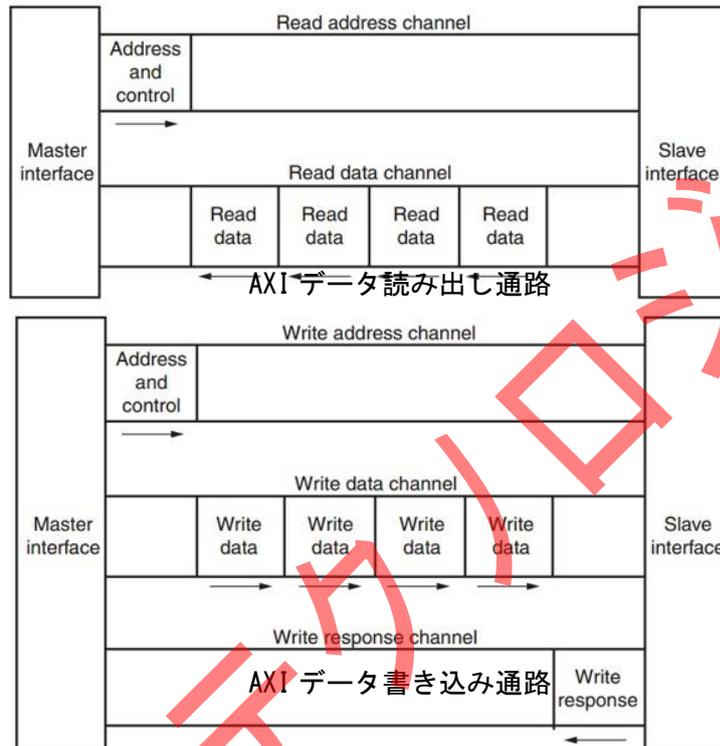
これはコンティニユアストリームインタフェースの一種で、アドレスコードが必要ない(FIFO に似ていて、ひたすらに読み出しあるいは書き込みすればいい)。このタイプの IP を対象にして、ARM は以上のようにメモリマッピングでコントロールできない(FIFO はアドレスのコンセプトがない)、コンバージョンデバイスが欠かせない。例えば、AXI-DMA モジュールでメモリマッピングからストリームインタフェースに変換を実現する。AXI-Stream が適用のケースは多い：ビデオストリームの処理；コミュニケーションプロトコルの変換；デジタル一眼レフカメラシグナルの処理；無線通信などで使います。本質ではニューメリカルフローにあって、構築されたデータ通路で、ソース(例えば ARM メモリ、DMA、無線レシーバフロントエンドなど)からシンク(例えば HDMI モニター、高速 AD サウンド出力など)まで連続データストリームをつくった。こんなインタフェースは実時間信号処理に向いている。

AXI4 と AXI4-Lite インタフェースは五つの通路を含んでいる :

- Read Address Channel
- Write Address Channel

- Read Data Channel
- Write Data Channel
- Write Response Channel

中に、各通路は独立の AXI ハンドシェーキングプロトコル。下の図はそれぞれ読み出しと書き込みのモデルを表している：



ZYNQ チップ内部はハードウェアで AXI バスプロトコルを実現した。9 つの物理インタフェースがあって、それぞれは AXI-GP0~AXI-GP3, AXI-HP0~AXI-HP3, AXI-ACP インタフェースである。

AXI\_ACP インタフェースは、ARM マルチコアアーキテクチャの一つのインタフェース、DMA のようなバッファが付けていない AXI ペリフェラルを管理する。PS ソケットは Slave インタフェースである。

AXI\_HP インタフェースは、高性能/帯域幅の AXI3.0 標準のインタフェースである。全部は四つあって、PL モジュールをメインデバイスに連結する。主に PL が PS (DDR と On-Chip RAM) のメモリをアクセスするとき使用する。

AXI\_GP インタフェースは、汎用な AXI インタフェースである。全部で四つあって、32 ビットのメインデバイスのインタフェースと 32 ビットサブデバイスのインタフェースが 2 つずつある。





次はよく使われる AXI インタフェース IP の機能紹介：

AXI-DMA：PS メモリから PL 高速伝送通路 AXI-HP<---->AXI-Stream の変換を実現する。

AXI-FIFO-MM25：PS メモリから PL 汎用伝送通路 AXI-GP<---->AXI-Stream の変換を実現する。

AXI-Datamover：PS メモリから PL 高速伝送エクスプレス通路 AXI-HP<---->AXI-Stream の変換を実現する。ただし、これは完全に PL にコントロールされて、PS はかんぜん被動的である。

AXI-VDMA：PS メモリから PL 高速伝送エクスプレス通路 AXI-HP<---->AXI-Stream の変換を実現する。ただし、ビデオ動画、画像などの二次元データにだけに応用できる。

AXI-CDMA：この仕事はデータをメモリのある場所から別の場所へ移動することである。PL で完成して、CPU が動き出す必要がない。

これらの IP をどう使うかについて、後の章で例を挙げ、説明する。時々ユーザー定義の IP を開発して PS と通信する必要がある。この場合はウィーザー生成を利用し、それ相応の IP をる。ユーザー定義の IP コアは AXI4-Lite、AXI4、AXI-Stream、PLB と FSL などのインタフェースを手に入れる。PLB と FSL は ARM がサポートできないため、使わない。

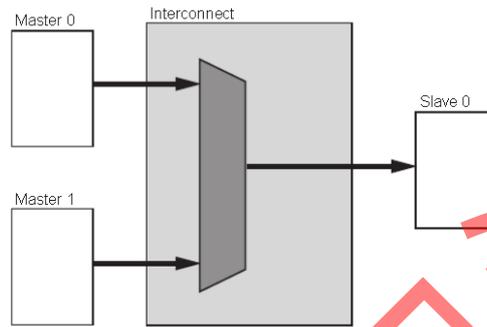
以上のオフィシャル IP とウィーザー生成のユーザー定義 IP があれば、ユーザーは AXI シーケンスに詳しく知る必要がある(問題が出る以外)。Xilinx は既に AXI シーケンスと関係あるディテールをパッケージングしたから、ユーザーは自分のロジックを実現できればいい。

厳しくいえば、AXI プロトコルは PTP のマスター・スレーブインタフェースプロトコルである。いくつかのペリフェラルが互いにデータ交換を必要する時、AXI Interconnect モジュールに加入する必要がある、いわゆる相互関連マトリクス。これの作用は一つあるいは多数の AXI メインデバイスと一つあるいは多数の AXI サブデバイスを連結する交換メカニズム(チェンジャーにある交換マトリクスに似ている)。

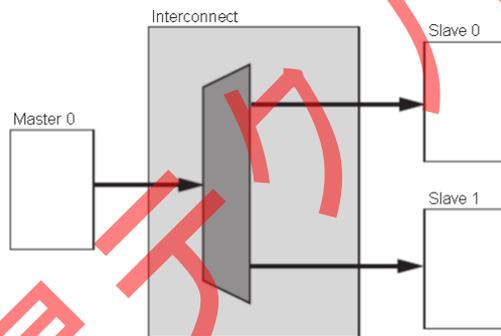
この AXI Interconnect コアは最大限でメインデバイス 16 個、サブデバイス 16 個をサポートできる。これ以上のインタフェースを必要する場合、IP コアを増えればいい。

AXI interconnect の基本的連結モードは以下の 4 種類ある :

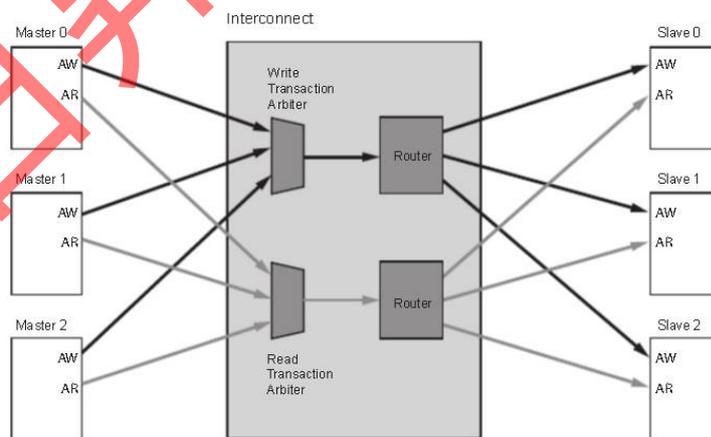
- N-to-1 Interconnect
- to-N Interconnect
- N-to-M Interconnect (Crossbar Mode)
- N-to-M Interconnect (Shared Access Mod)



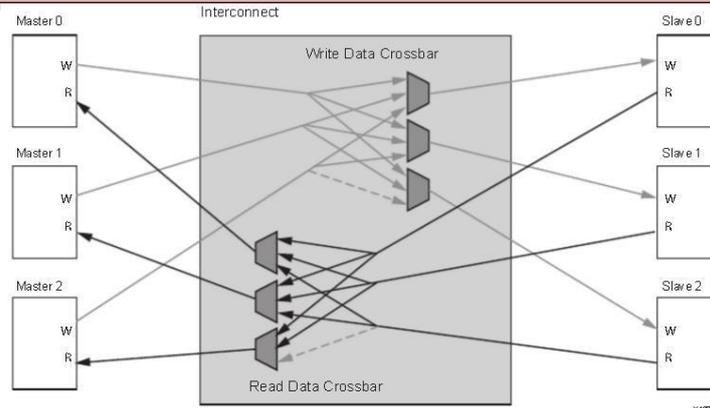
多対一の場合



一対多の場合

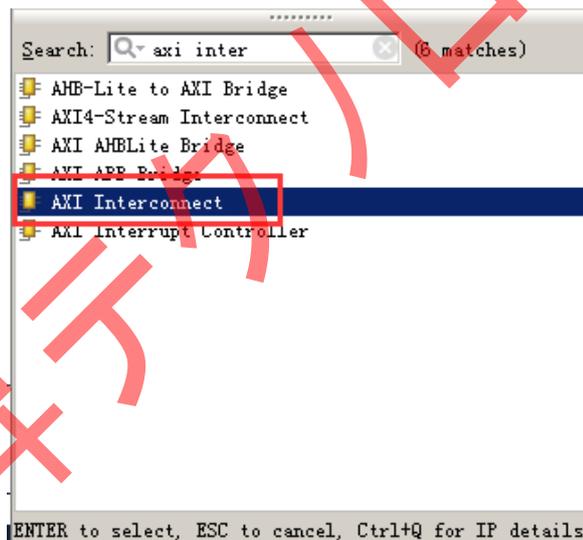


多対多リードライトアドレスチャンネル



多対多リードライトデータチャンネル

ZYNQ 内部の AXI インタフェースデバイスは相互関連マトリクスの方法で繋がっている。データ伝送の効率性を確保した同時に、連結の柔軟性も保証できる。Xilinx は Vivado でこの相互関連マトリクスを実現する IP コア axi\_interconnect を提供したので、必要の時はコールするだけでいい。



AXI Interconnect IP

## 2.2 ZYNQ チップの開発流れの紹介

ZYNQ は CPU と FPGA を統合したため、開発エンジニアは ARM の OS アプリケーションとデバイスドライバだけでなく、FPGA 部分のハードウェアロジックをプログラミングする。開発中は LinuxOS とシステムの構造を理解し、FPGA と ARM システムのハードウェアプログラムプラットフォームを構築するのも需要である。というわけで、ZYNQ の開発はソフトエンジニアがハードウェアエンジニアと協同して、プログラミングと開発することである。即ち、ZYNQ 開発中が言っていた“HW-SW Co-design”である。

ZYNQ システムのハードシステムとソフトシステムをプログラミング、開発するに必要な開発環

---

## 境とデバッグツール : Xilinx Vivado

Vivado デザインーツは FPGA 一部のプログラミングと開発、ピンとシーケンスの制限、コンパイルとシミュレーション、そして RTL からビットストリームのプログラミング流れを実現する。Vivado は ISE デザインーツを簡単にアップグレードしたものではなく、新たなデザインーツである。Vivado が ISE デザインーツにある全ての重要ツールにとって代わった。例えば、Project Navigator、Xilinx Synthesis Technology、Implementation、CORE Generator、Constraint、Simulator、Chipscope Analyzer、FPGA Editor などのデザインーツなどである。

Xilinx SDK (Software Development Kit)、SDK は Xilinx ソフトウェア開発ーツ (SDK) である。Vivado ハードウェアシステムのもとで、システムは自動的に一部の重要なパラメータを配置する。中にツールとベースパス、コンパイラオプション、JTAG とフラッシュメモリ設定、デバッガー連結とベアメタルボードサポートパッケージ (BSP) が含まれている。サポートしている全ての Xilinx IP ハードコアにも、SDK はドライバープログラムを提供している。SDK は IP ハードコア (FPGA) とプロセッサソフトウェアの協同デバッグをサポートしている。高級 C 又は C++ 言語を使って、AEM と FPGA システムを開発とデバッグすることができる。こうしてハードウェアシステムが順序に働いているかをテストする。SDK ソフトも Vivado 内部にあるもので、別々でインストールの必要がない。

ZYNQ に開発のハードウェア先で、ソフトウェアが後にする方法である。具体的な流れはこうなる :

- 1) Vivado に新規プロジェクトを作成し、組込み型のソースファイルを一つ増加する。
- 2) Vivado に PS と PL の一部基本的なペリフェラルを添加、配置しする。あるいは、ユーザー定義のペリフェラルを配置する。
- 3) Vivado にトトップファイル HDL を生成し、制約ファイルを加わる。そして、ビットストリームファイル (\*.bit) をコンパイルする。
- 4) ハードウェアメッセージを SDK ソフト開発環境に書き出する。SDK 環境にデバッグソフトを数個プログラミングして、これらのソフトでハードウェアとソフトウェアを検証したあと、ビットストリームファイルと合わせて ZYNQ システムを単独にデバッグする。
- 5) SDK に FSBL ファイルを生成する。
- 6) VMware バーチャルマシンに u-boot.elf、bootloader のカーネルミラーイメージを生成する。
- 7) SDK に FSBL ファイル、ビットストリームファイル system.bit と u-boot.elf ファイルで BOOT.bin ファイルを生成する。
- 8) VMware に Ubuntu のカーネルミラーファイル Zimage と Ubuntu のルートファイルシステムを生成する。他にも、FPGA のユーザー定義 IP を対象にドライバーをプログラミングする必要がある。

9) BOOT、カーネル、デバイスツリー、ルートファイルシステムを SD カードに書き込み、開発ボードの電源を入れたら、LinuxOS は SD カードから作動する。

以上は典型的な ZYNQ 開発ボード流れである。でも、ZYNQ も単純に ARM として使われる。こうすれば、PL 側の資源が必要なくなり、伝統的な ARM 開発とそんなに区別を付けていない。ZYNQ も PL 部分だけを使うが、PL の配置はやはり PS で完成する。つまり、PL だけ必要とするファームを固体化するには、伝統的な Flash の固体化方法で実現できない。

### 2.3 ZYNQ の勉強はどんなスキルが必要

ZYNQ の勉強は FPGA、MCU、ARM などの従来のツールよりレベルが高い、ZYNQ を使いこなすのもすぐ達成できることではない。

#### 2.3.1 ソフト開発エンジニア

- ✓ コンピューター基本知識
- ✓ C、C++言語
- ✓ コンピューターOS
- ✓ tcl スクリプト
- ✓ 優れた英語を閲覧する基礎

#### 2.3.2 ロジック開発エンジニア

- ✓ コンピューター基本知識
- ✓ C 言語
- ✓ デジタル回路基礎
- ✓ Verilog、VHDL 言語
- ✓ 優れた英語を閲覧する基礎

## 第三章 Vivado 開発環境

### 3.1 Vivado ソフトの紹介

Vivado は Xilinx 会社が 2012 年発表された新世代インテグレーションデザイン環境である。Vivado Design Suite User Guide の Getting Started (UG910) にこう書いていた：プログラムの効率を上げる為、Vivado を出すことにした。このソフトは大幅に Xilinx の 28nm 工芸のプログラマブルロジックデバイスのデザイン、総合と実現効率を上げる事ができる。予想できるのは、FPGA が 28nm 時代に入り、ISE ツールは時代から外されているように見える。ハードウェアがレベルアップしたのに、ソフトウェアがそのままだと、デザイン効率は必ず影響を受ける。

### 3.2 Vivado ソフトバージョン

ZYNQ 開発ボードの全てのサンプルと教程は Vivado2017.4 の開発環境で完成するので、ソフトバージョンで解明できない問題を避けるよう、勉強中はこちらと一致してください。使用する前に Vivado 2017.4 をインストールする必要がある。Xilinx の公式サイトからダウンロードできる。オフィシャルサイトのダウンロードはアカウント登録が必要である。

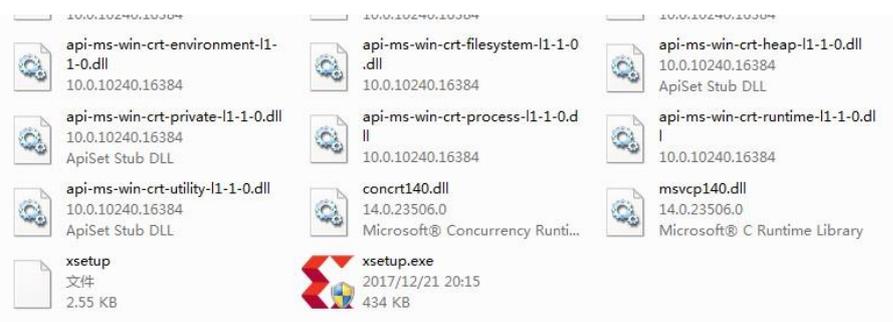
Vivado ソフトの Xilinx オフィシャルダウンロードサイト：

<https://japan.xilinx.com/support/download.html>

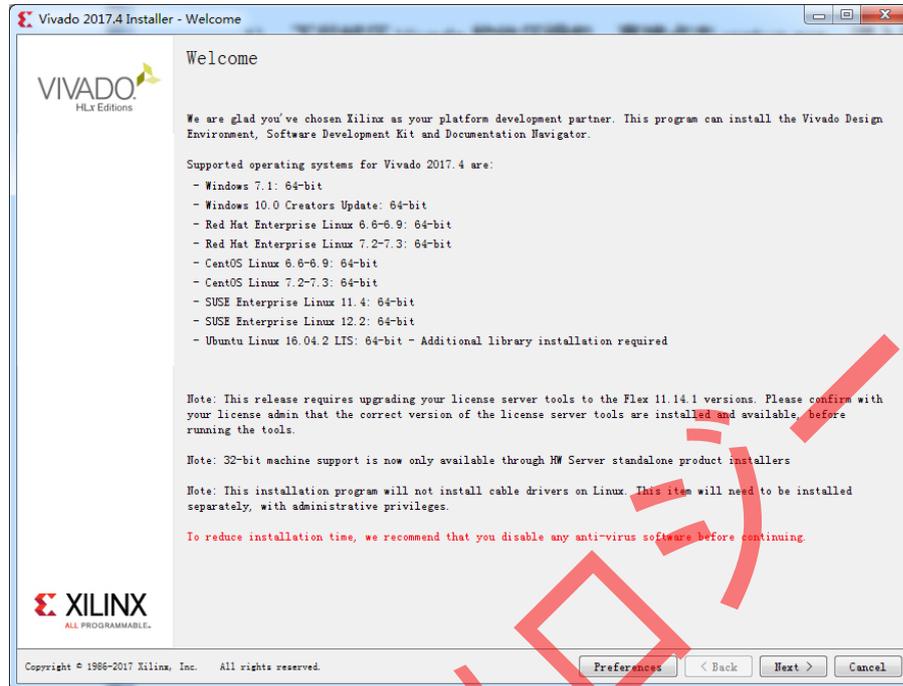
Vivado の Linux バージョンと Windows バージョン、それとツワンタイプのも提供している。ここで使うのはツワンタイプで、Windows と Linux 両方の開発を満足できる。Vivado を使用するには OS は 64 ビットになければならない。

### 3.3 Vivado ソフトの Windows でのインストール

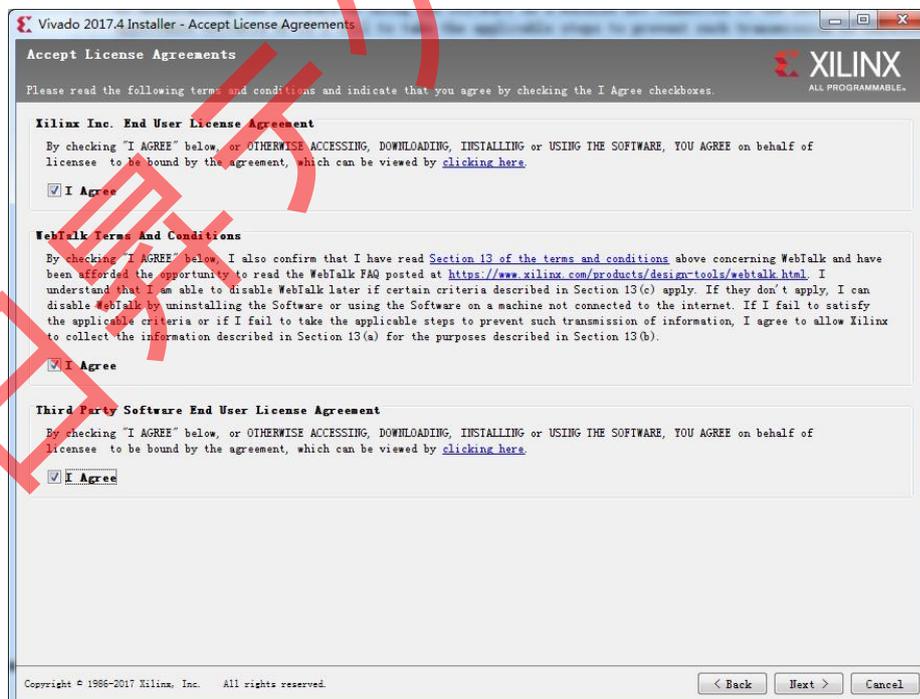
1) Vivado ソフトの圧縮ファイルダウンロードして、解凍する。xsetup.exe をクリックし、インストール開始。



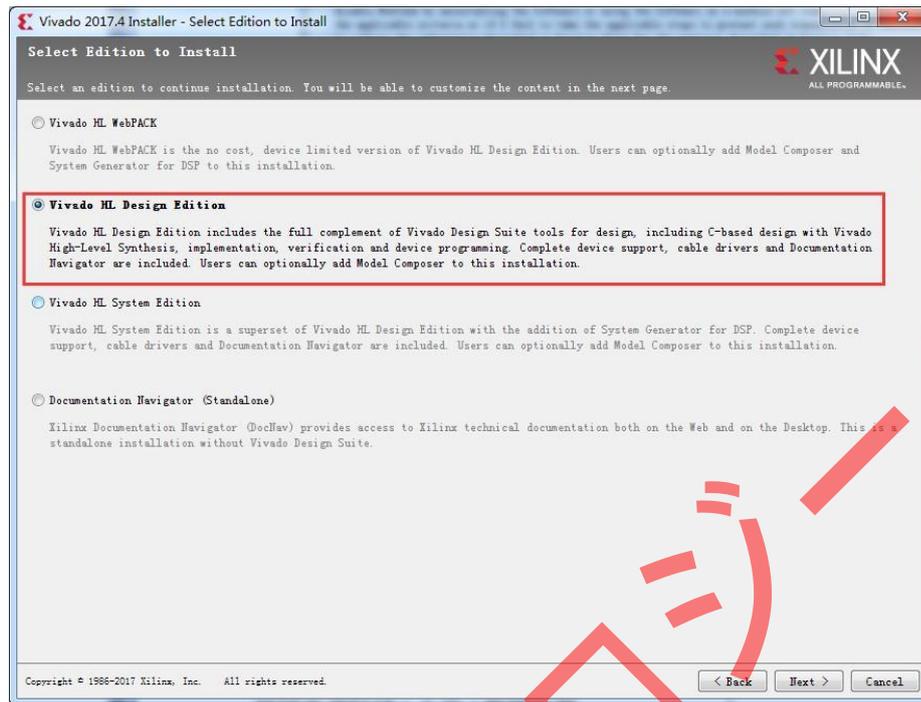
- 2) バージョンアップを提示されたら、それを無視して、“continue” をクリックする。
- 3) “next” をクリックする、Vivado からのシステムに対する要求が見える。



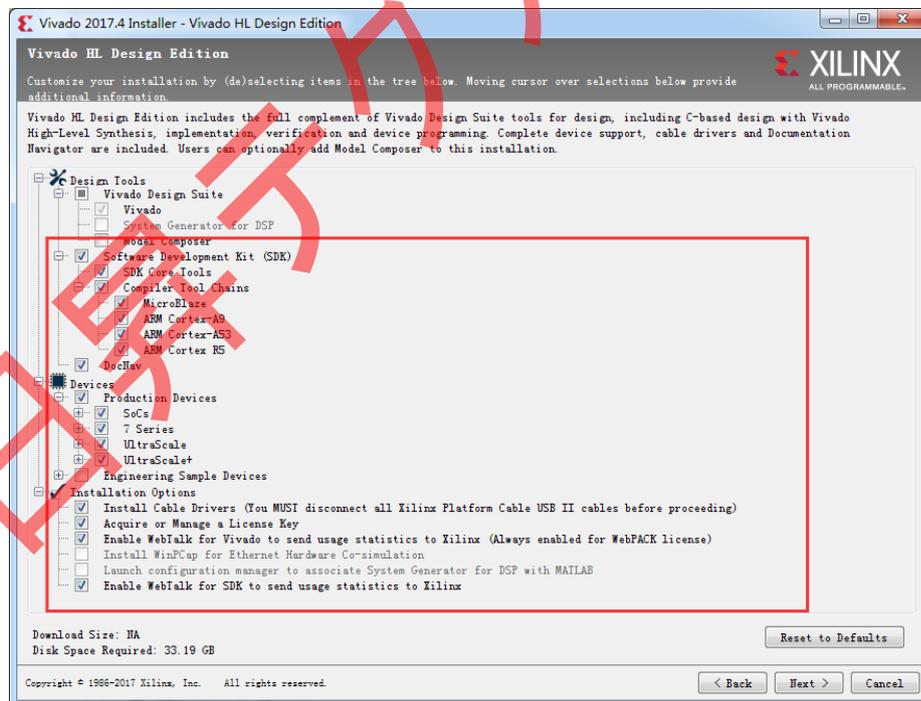
- 4) “I Agree” をクリックして、各条項を受ける。



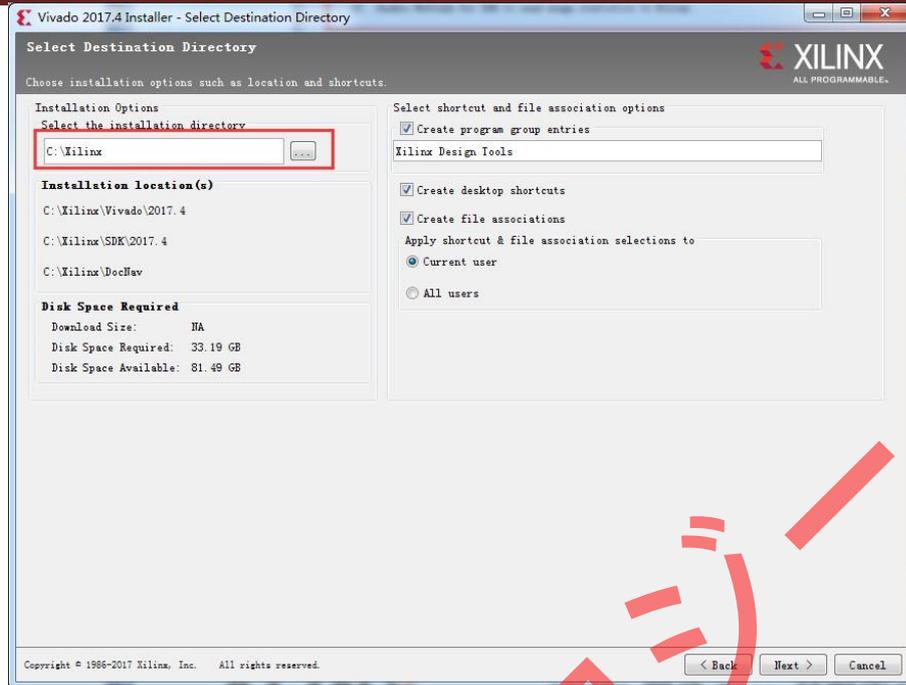
- 5) “Vivado HL Design Edition” を選択する。



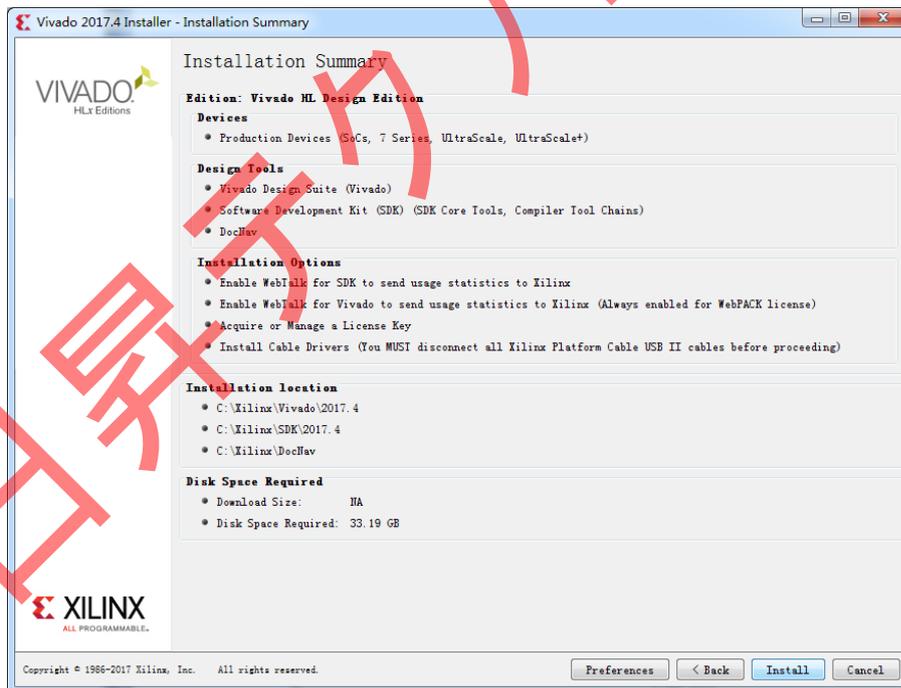
- 6) デフォルト配置を使用して、“next”をクリックする。



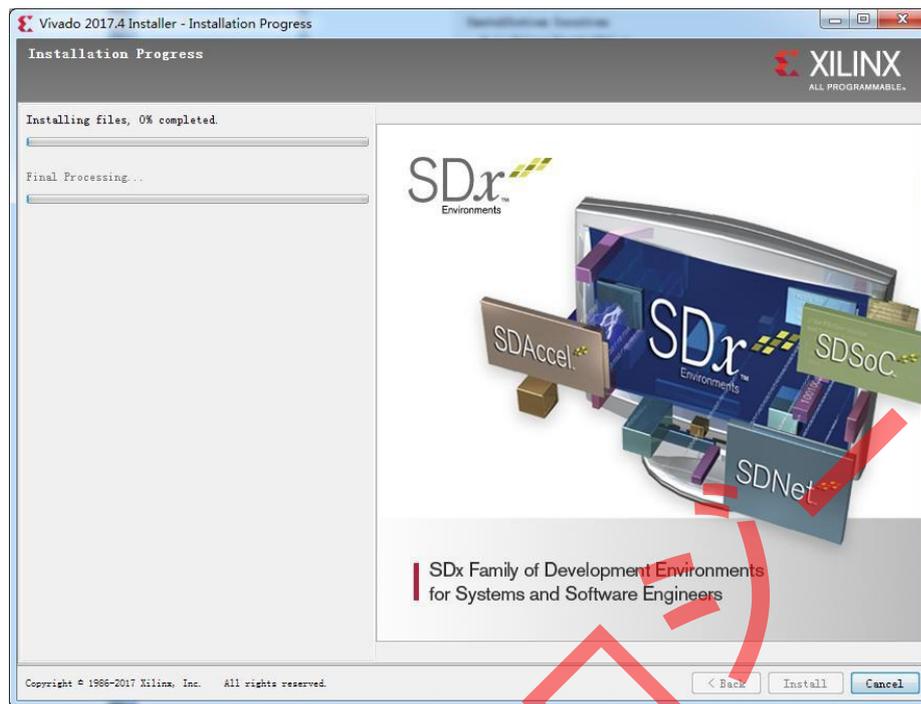
- 7) ここではインストールパスを変えていない。インストールパスには日本語、スペースなどの特殊文字を入れないようにする。コンピューターのユーザーネームも日本語、スペースを含めないようにする。Vivadoはハードディスクへの大きさ要求はおよそ33Gである。



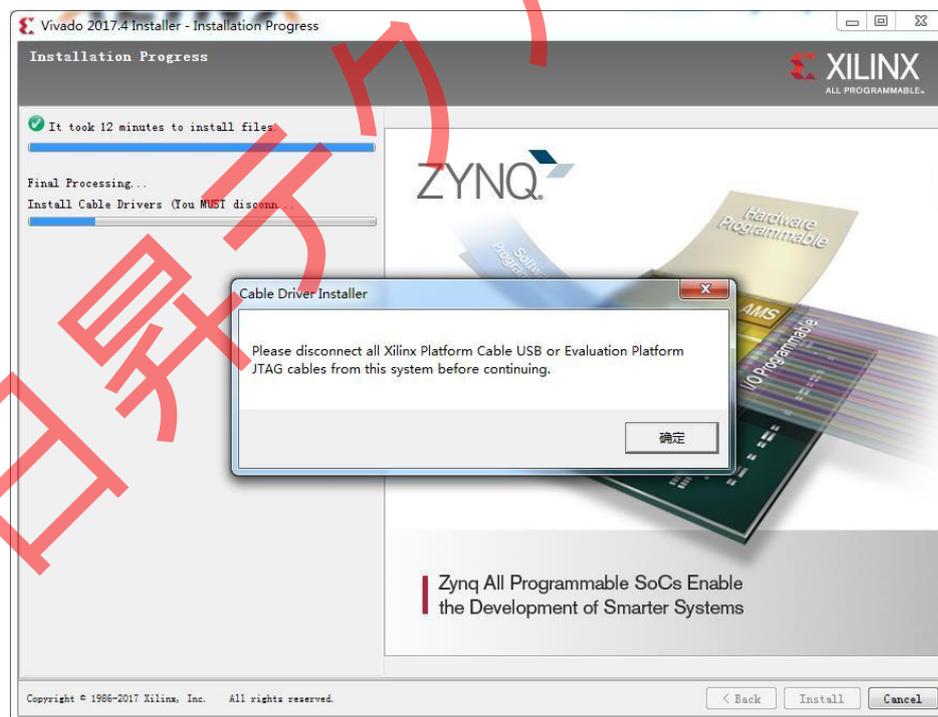
8) “Install” をクリックする。



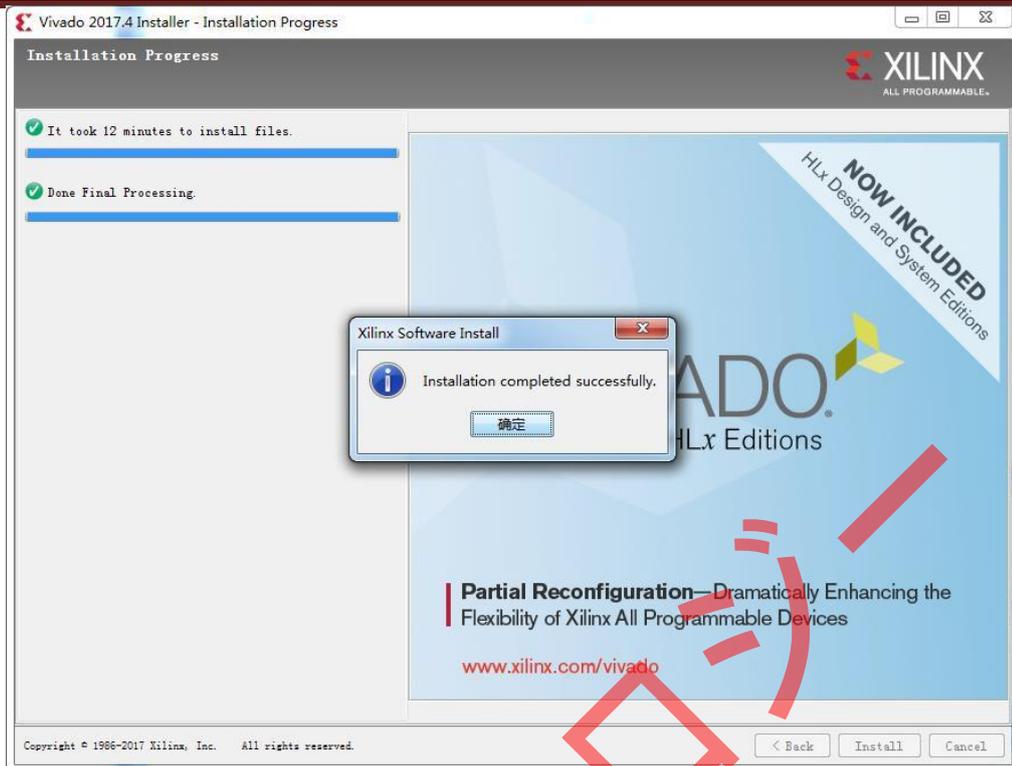
9) インストールを待つ時間がながく、ウイルス防止ソフトを止めなかったら、インストール中ブロックされ、インストール後使えない可能性がある。



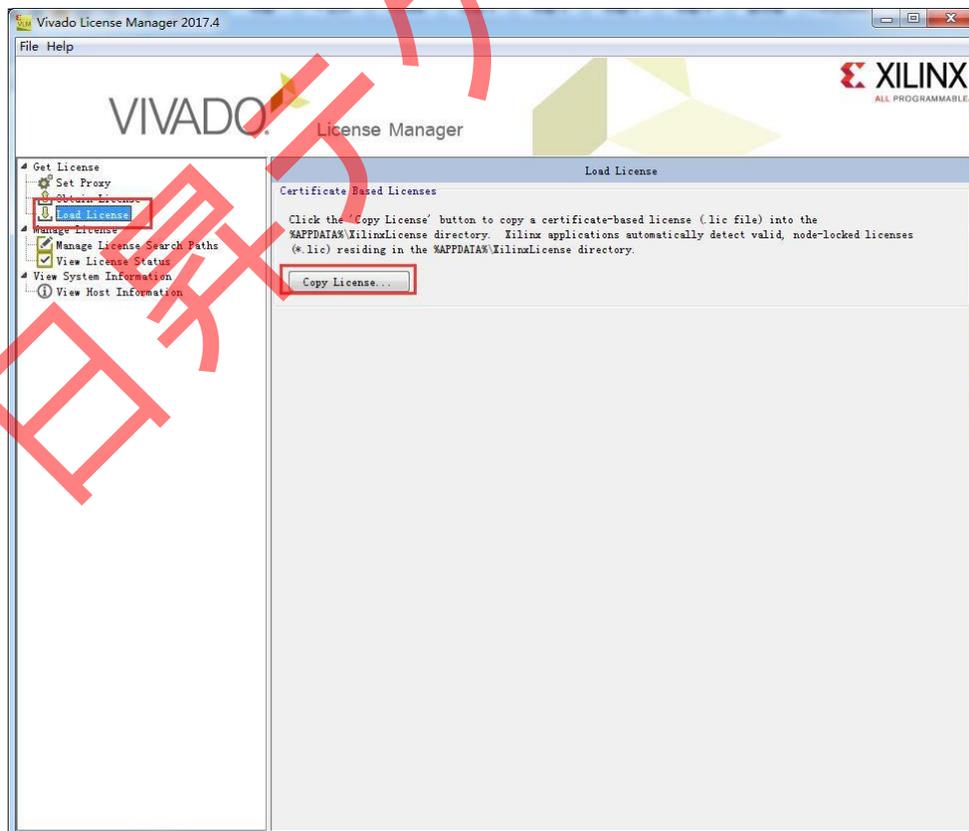
10) ダウンローダーや開発ボードの JTAG 線を切る画面が出て、“確定”をクリックする。



11) インストール完了の画面が出る。



- 12) License ファイルをインストールする。“Copy License” をクリックして、“xilinx\_ise\_vivado.lic” というファイルを選択する。



13) インストール成功のウィンドウが見える。



14) もう一度ダウンローダードライバーをインストールする必要があったら、Vivado のインストールパス “X:\Xilinx\Vivado\2017.4\data\xicom\cable\_drivers\nt64\digilent” に入って、“install\_digilent.exe” をクリックしてインストールする。その前に Vivado ソフトをしめる。Vivado がダウンローダーを認識できない場合、ファイアウォールとウィルス防止ソフトを閉めてください。同時にいくつかのバージョン違いの Vivado と ISE を開けることもいけない。

日昇テクノロジー

## 第四章 PL の “Hello World” LED テスト

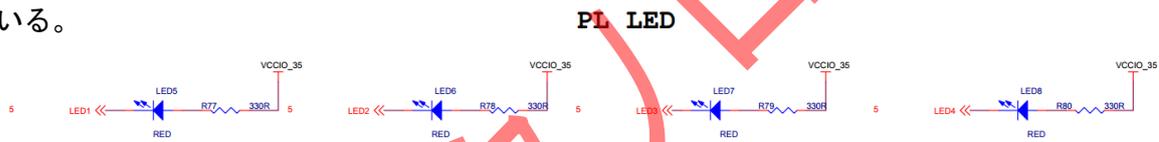
### Vivado をテストするプロジェクトは “led”

ZTNQ にとって、PL (FPGA) 開発はとても重要である。これも ZYNQ は他の ARM よりメリットあるところで、いろんな ARM 側のペリフェラルをカスタマイズできる。カスタマイズする前に、まずは LED サンプルで PL (FPGA) の開発流れと Vivado ソフトの基礎操作を慣れましょう。この開発流れは ARM なしの FGPA チップと完全一致している。

本サンプルにするのは LED ライトをコントロールする実験である。秒ことで開発ボード上の LED ライトをコントロールし、反転させる。ライトの点灯、消灯のコントロールを実現する。LED のコントロールができれば、ほかのペリフェラルも徐々に把握できる。

### 4.1 LED ハードウェアの紹介

1) 開発ボードの PL 部分は赤い LED ライトを 4 つ接続した。ライトは完全に PL でコントロールされている。



2) 回路図による接続関係で LED と PL ピンの連携関係を確認できる。

17P_T2_AD5P_35	H20	IO2_9P	15
17N_T2_AD5N_35	G19	IO2_9N	15
8P_T2_AD13P_35	G20	IO2_3P	15
3N_T2_AD13N_35	H15	IO2_3N	15
IO_L19P_T3_35	G15	IO2_16P	15
19N_T3_VREF_35	K14	IO2_16N	15
20P_T3_AD6P_35	J14	IO2_17P	15
20N_T3_AD6N_35	N15	IO2_17N	15
I_DQS_AD14P_35	N16	KEY1	12
I_DQS_AD14N_35	L14	KEY2	12
22P_T3_AD7P_35	L15	RTC_DATA	11
22N_T3_AD7N_35	M14	RTC_RESET	11
IO_L23P_T3_35	M15	LED1	12
IO_L23N_T3_35	K16	LED2	12
4P_T3_AD15P_35	J16	LED3	12
4N_T3_AD15N_35		LED4	12

対応のピンのインフォメーション

3) 回路図に PS\_MIO をはじめの IO は PS 側のもので、連携を必要ないし、連携できない。

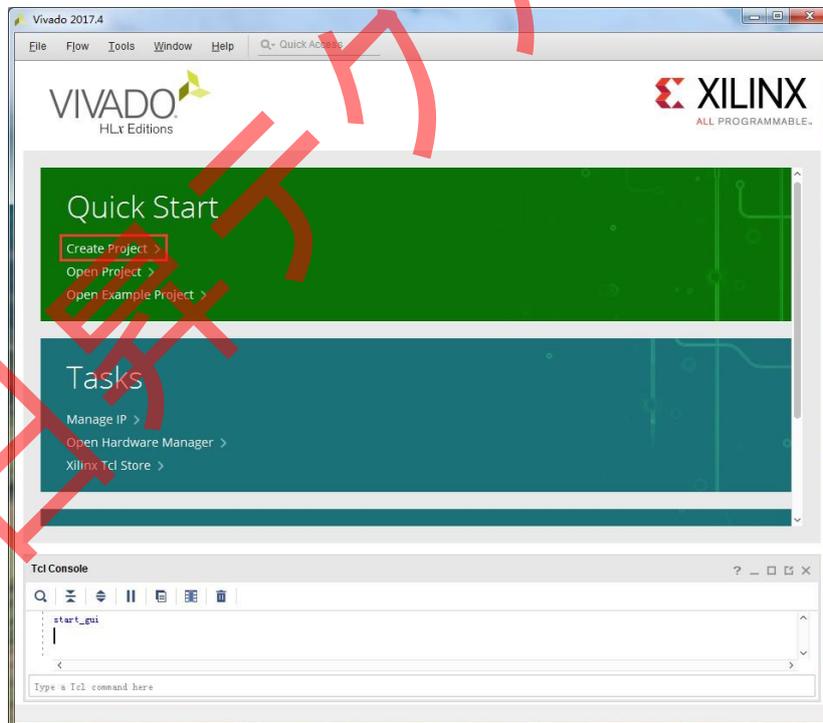
	E6
PS_MIO0_500	A7
PS_MIO1_500	B8
PS_MIO2_500	D6
PS_MIO3_500	B7
PS_MIO4_500	A6
PS_MIO5_500	A5 R1
PS_MIO6_500	D8
0 PS_MIO7_500	D5
PS_MIO8_500	B5
PS_MIO9_500	E9
PS_MIO10_500	C6
PS_MIO11_500	D9
PS_MIO12_500	E8
PS_MIO13_500	C5
PS_MIO14_500	C8
PS_MIO15_500	C7
PS_POR_B_500	E7
PS_CLK_500	

#### 4.2 Vivado プロジェクトを作成

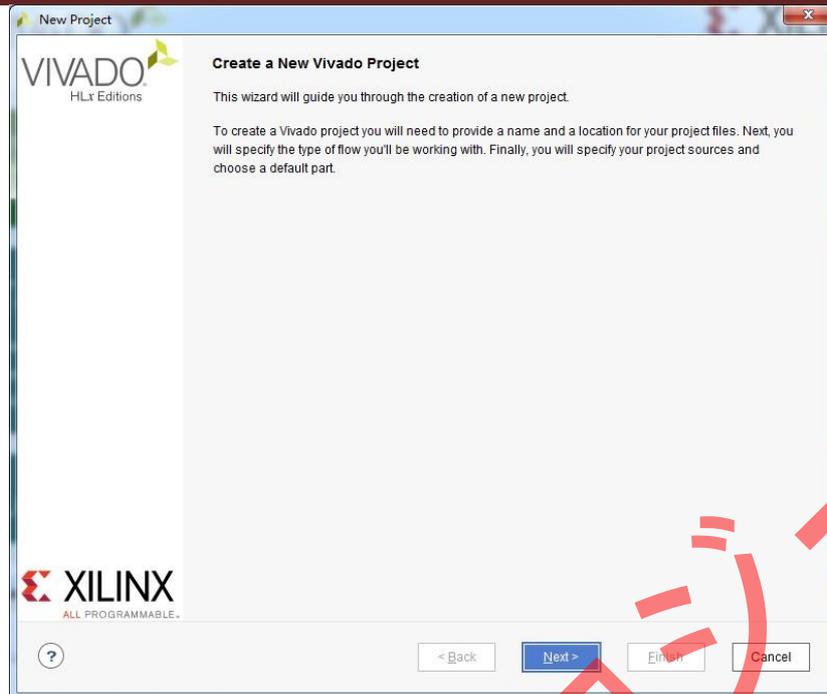
1) Vivado を起動する。Windows で Vivado のダブルクリックして起動する。



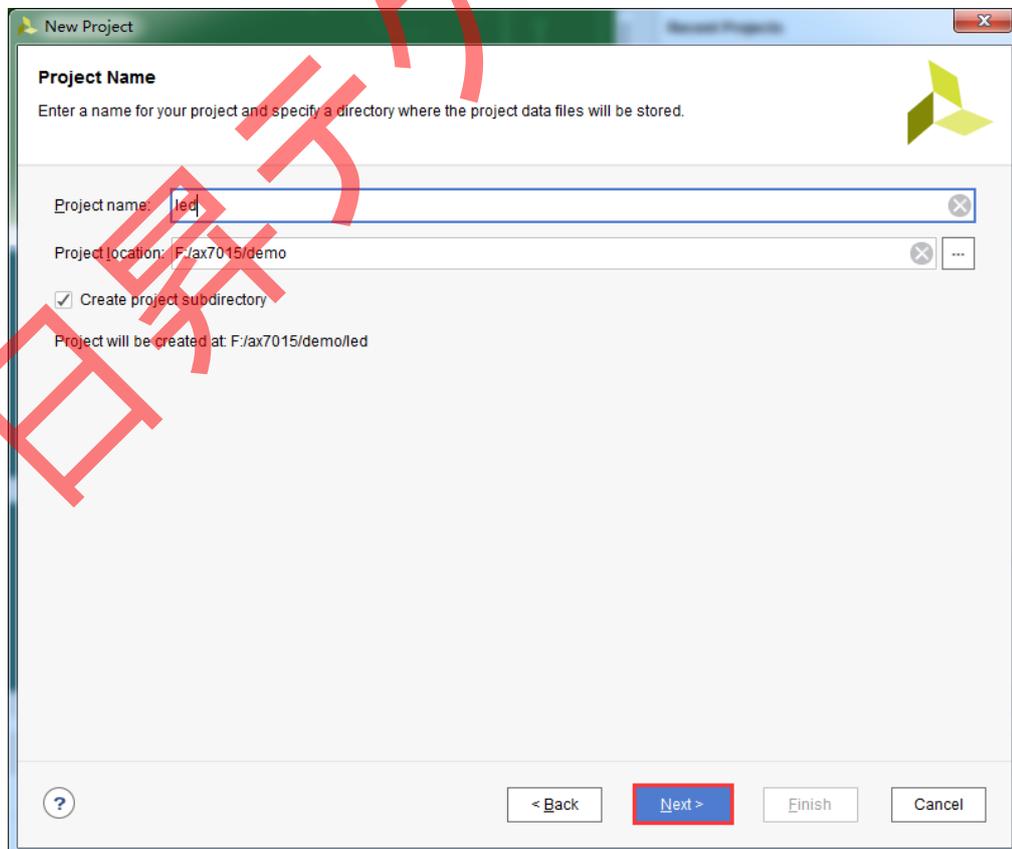
2) Vivado の開発環境に “Create New Project” をクリックして、新規プロジェクトを作成する。



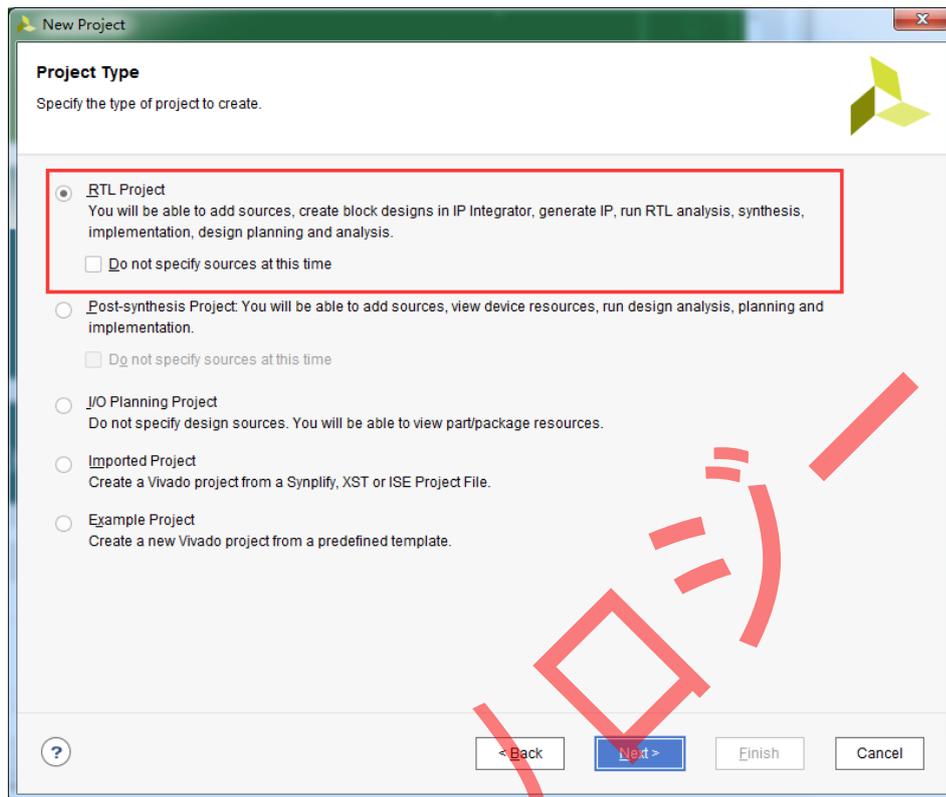
3) 画面に新規プロジェクトを作成するガイダンスが出て、“Next” をクリックする。



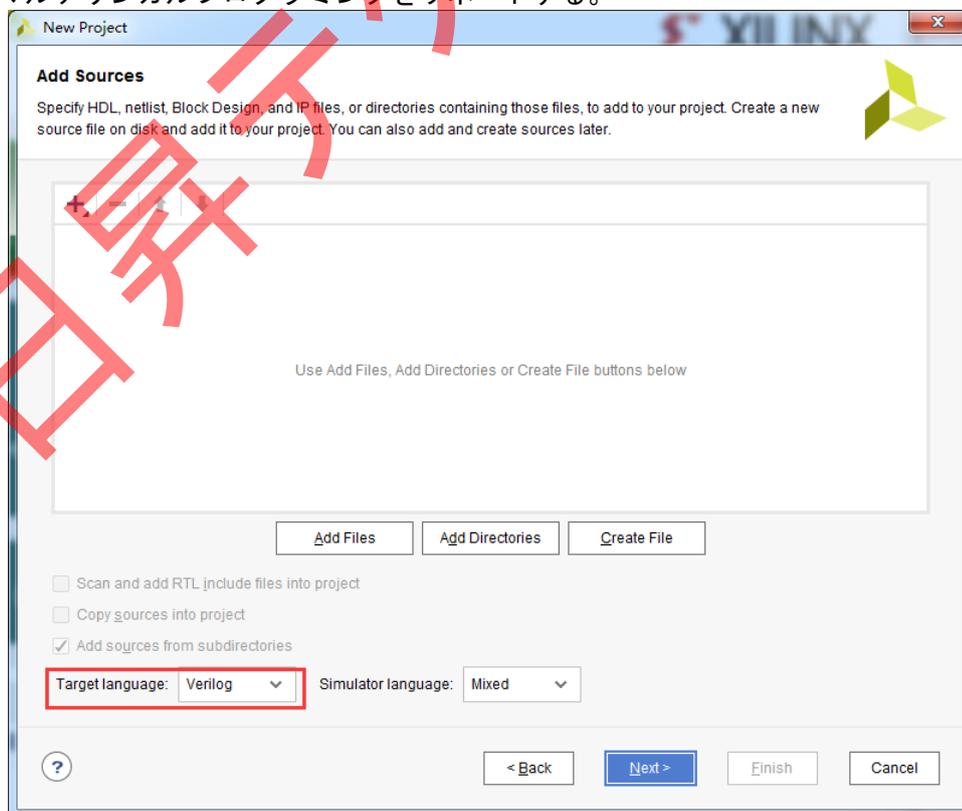
- 4) ポップアップしたダイアログにプロジェクトのネームと保存する場所を入力する。ここは led というプロジェクトネームを付ける。注意すべきことであるが、“Project location” に漢字とスペースを入れないように、パスの名称も長くつけてはいけない。



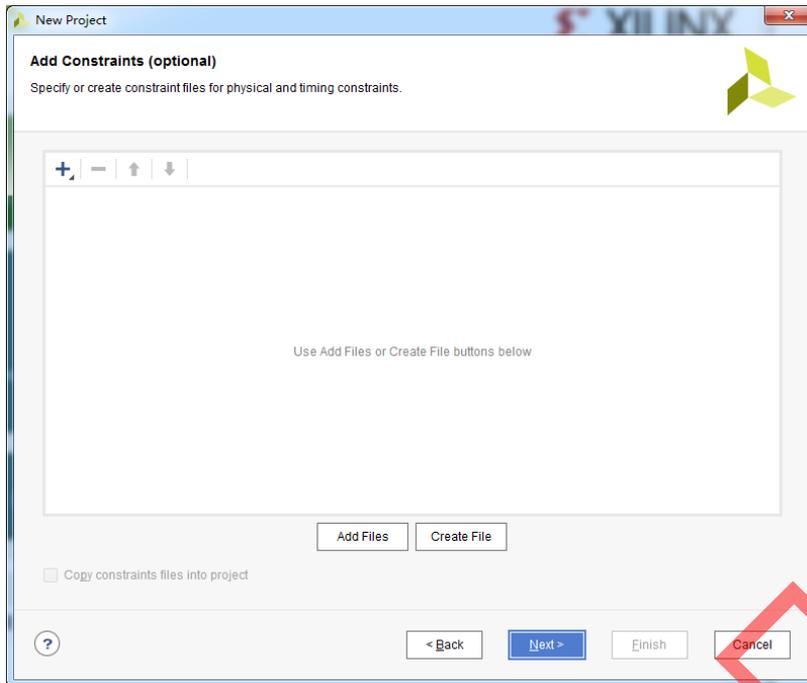
5) プロジェクトタイプで“RTL Project”を選択する。



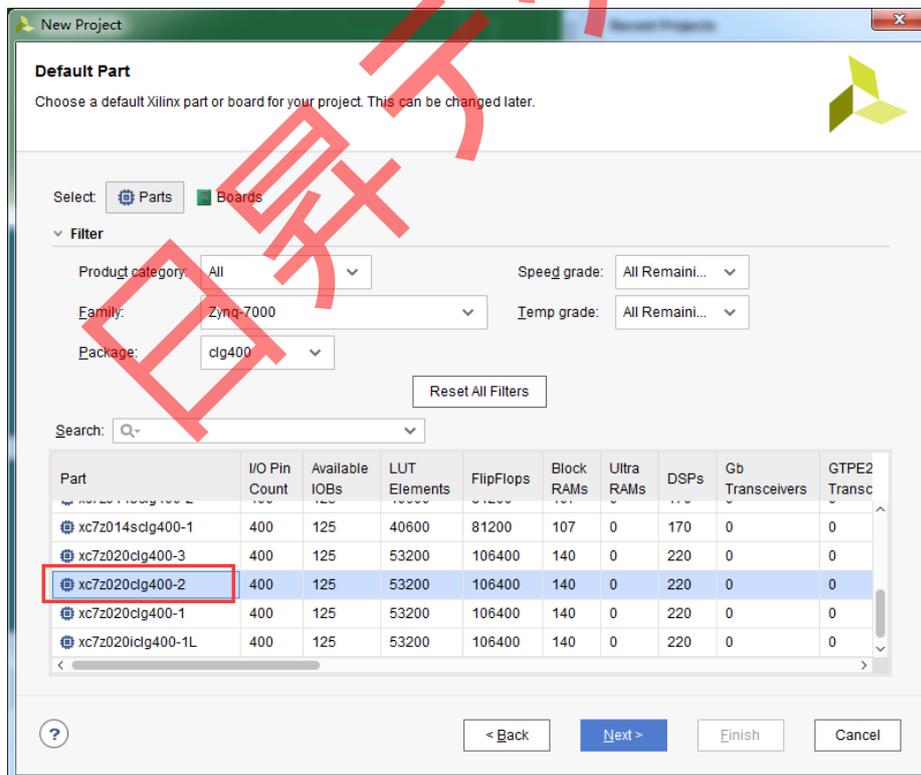
6) ターゲット言語“Target language”で“Verilog”を選ぶ。“Verilog”を選択したが、VHDLも使用できる。マルチリンガルプログラミングをサポートする。



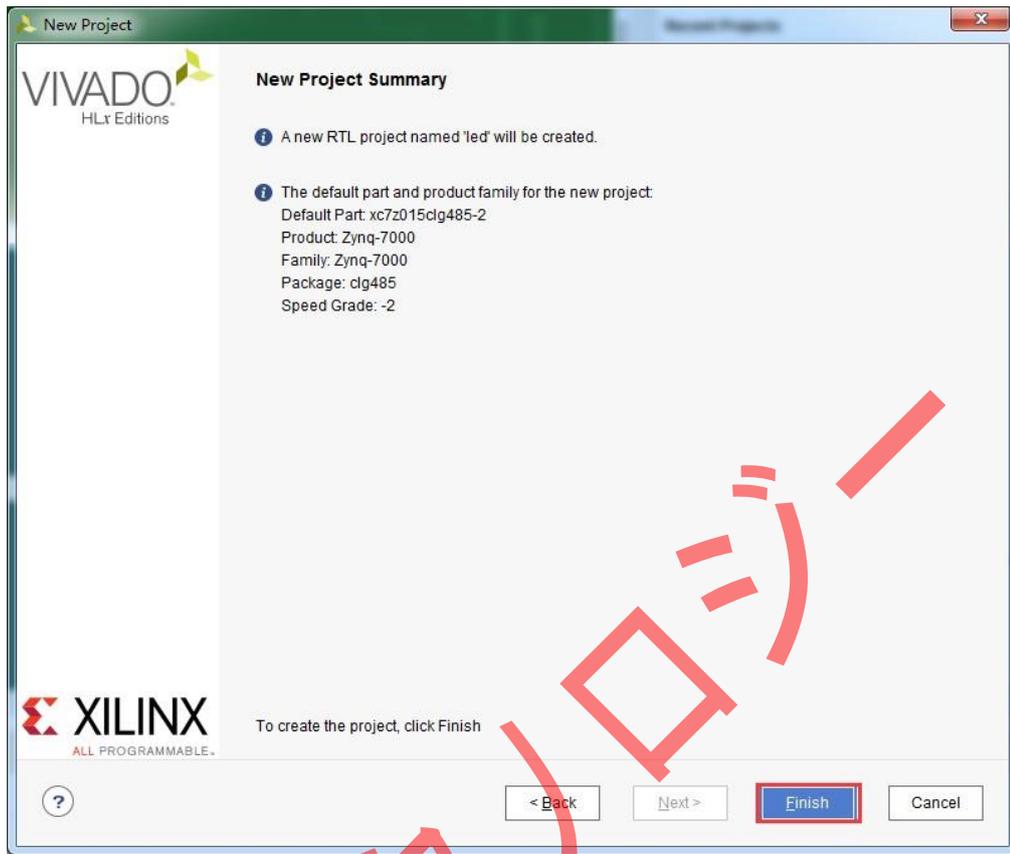
7) “Next” をクリックして、ファイルを追加しない。



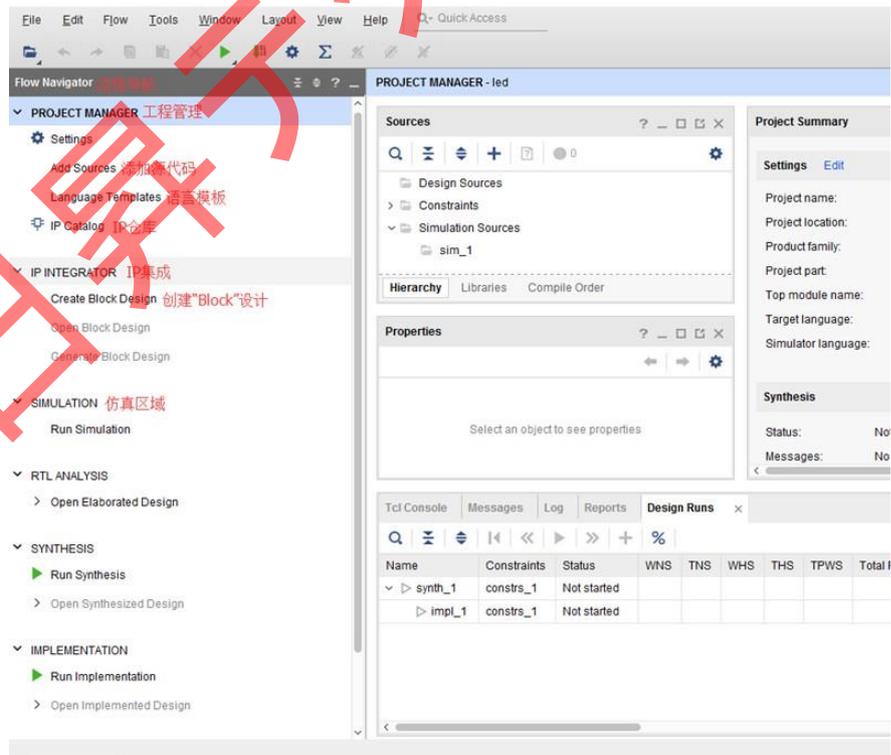
8) “Default Part” オプションを開き、Filter の “Family” で “Zynq-7000”、パッケージングタイプ “Package” で “clg400” を選択する。AX7020 の方はそれぞれプルダウンリストで “xc7z020clg400-2” を選ぶ。“-2” はスピードレベルを表している。数字が大きいければ大きいほど、性能はよくなる。スピード高いチップは下位互換性がある。



9) “Finish” をクリックして、“led” という名のプロジェクトは完成できる。

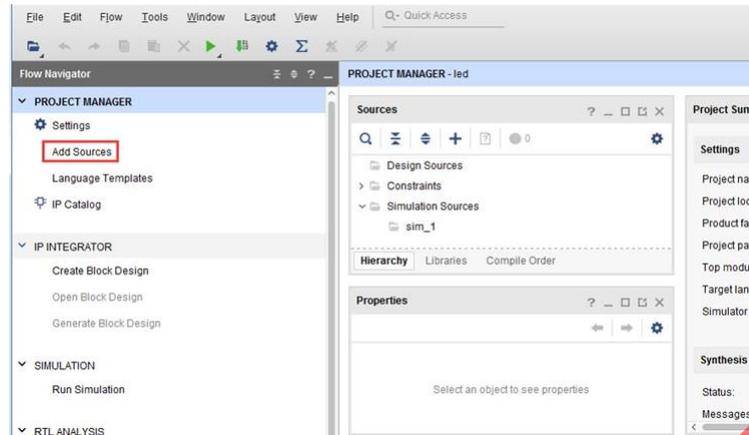


10) Vivado ソフトのインターフェイス

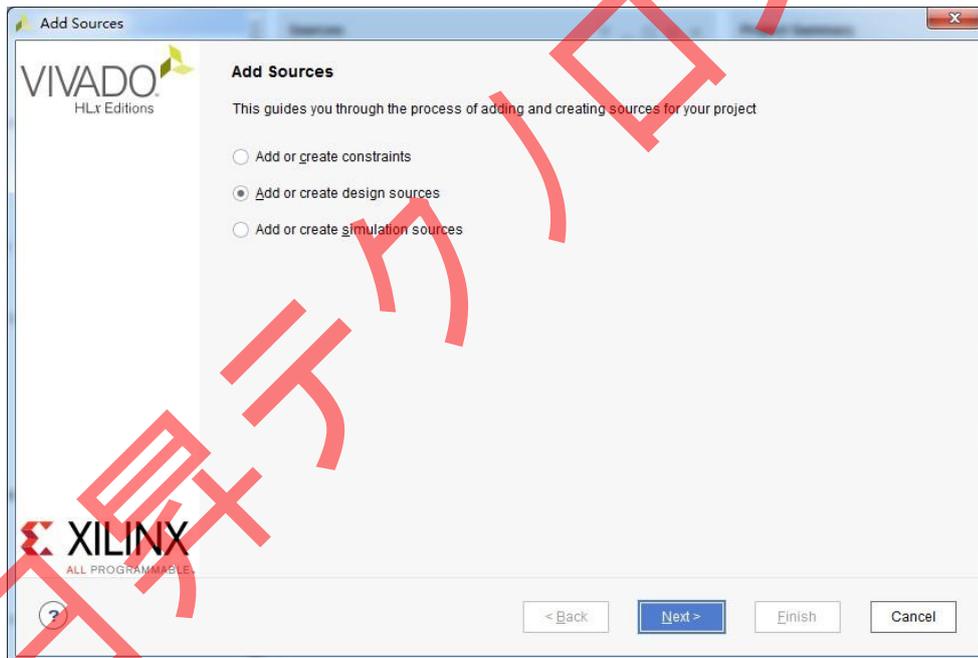


### 4.3 Verilog HDL ファイルを作成し、LED を点灯する

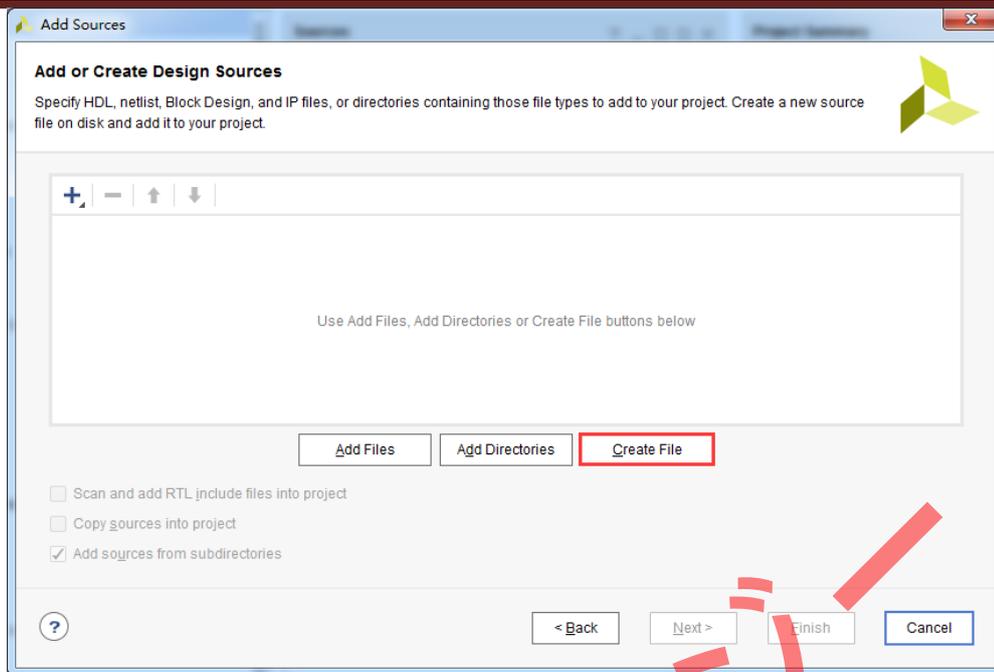
1) Project Manager の Add Sources をクリックする (あるいはショートキーAlt+A を使う)



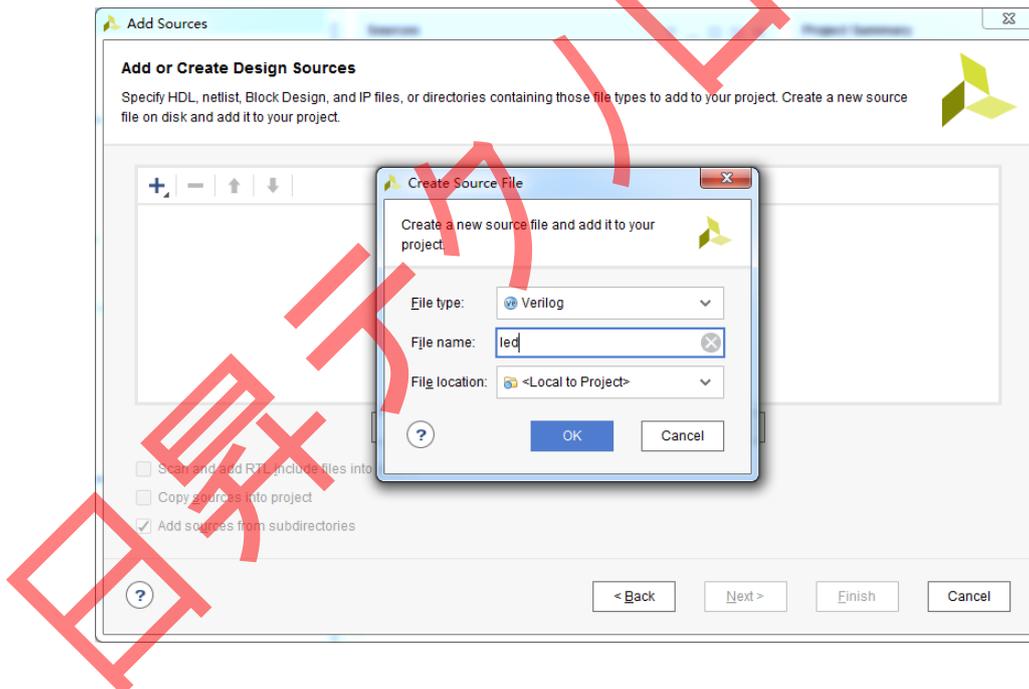
2) 設計ソースを追加また作成の “Add or create design sources” を選択して、“Next” をクリックする。



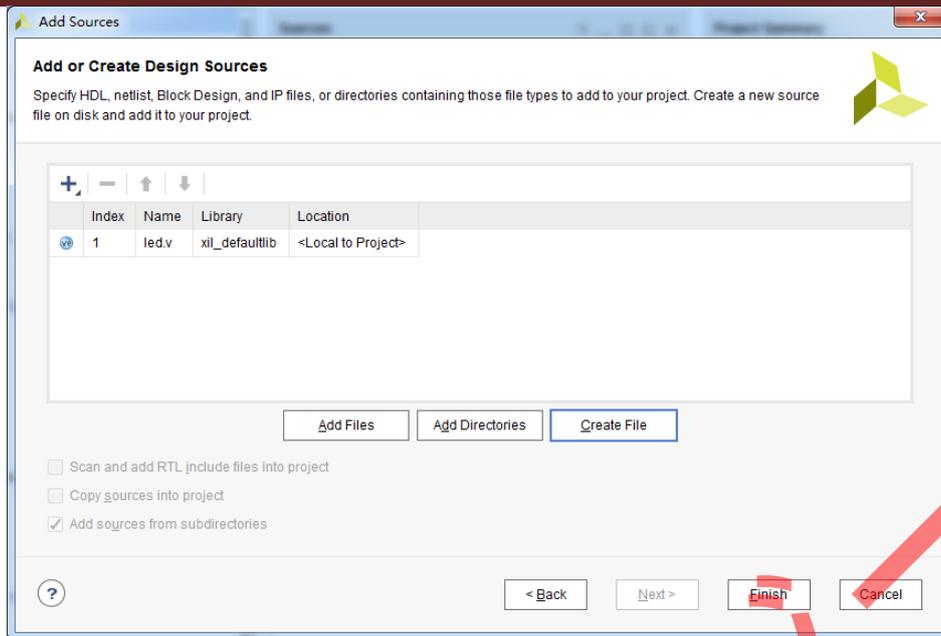
3) ファイル作成 “Create File” を選択する。



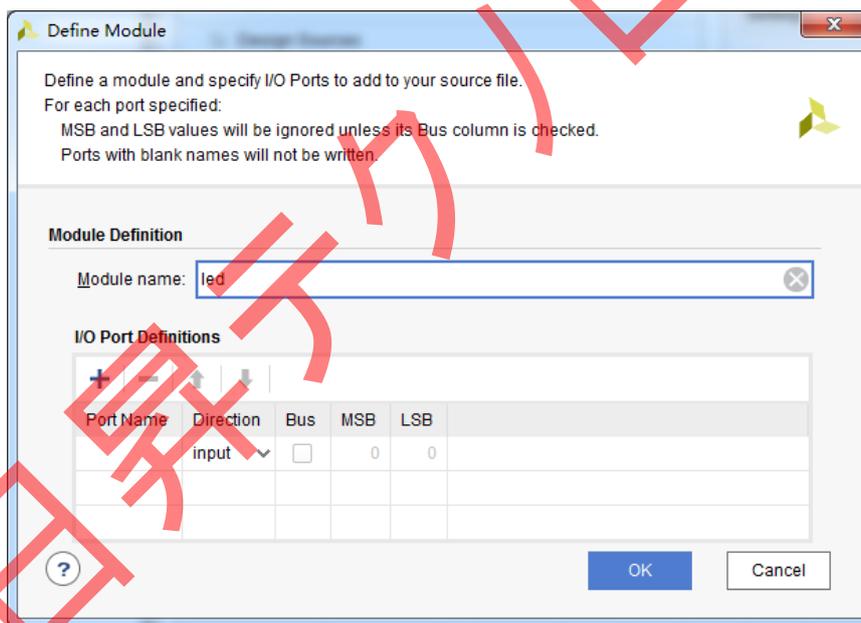
4) ファイル名“File name”を“led”にして、“OK”をクリックする。



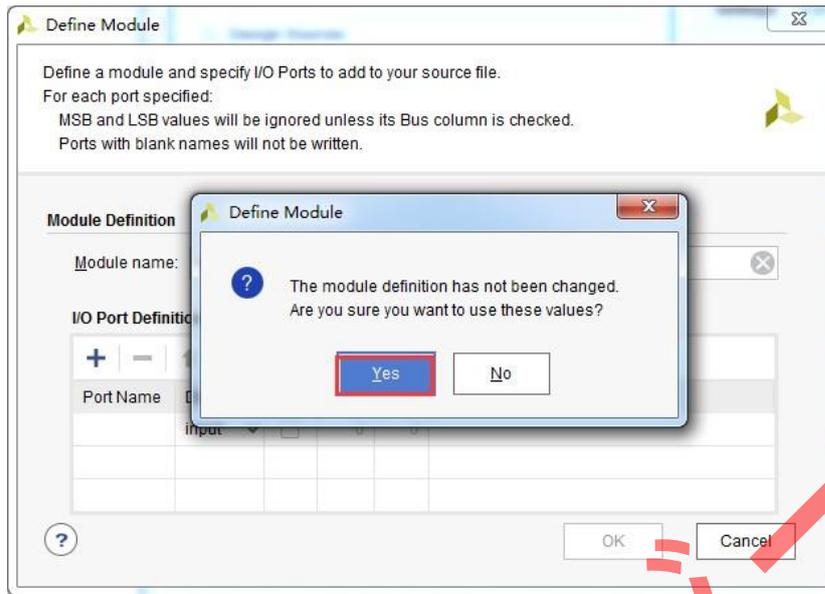
5) “Finish”をクリックして、“led.v”の追加は完成する。



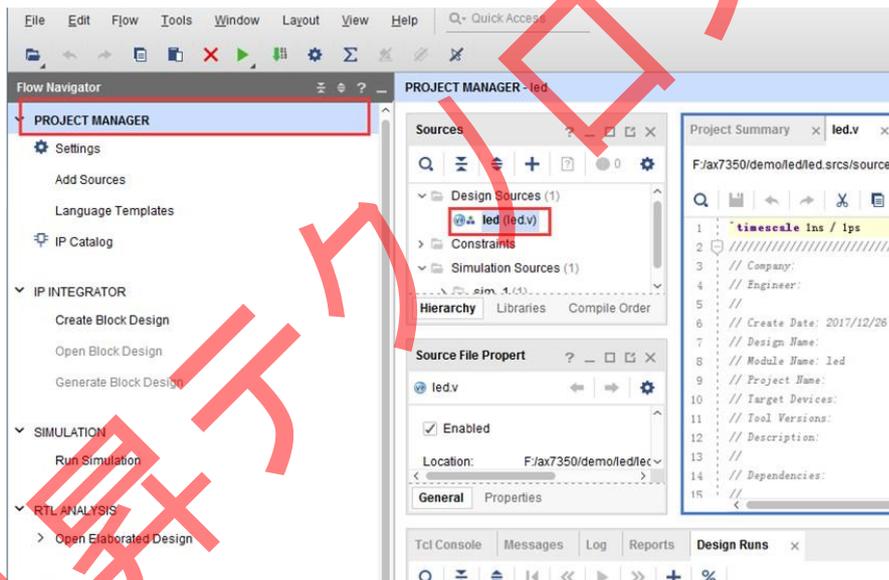
6) ポップアップしたモジュール定義 “Define Module” に、ファイル “led.v” のモジュール名を “Module name” に指定することができる。ここでは、“led”にする。一部のソケットも指定できるが、今は暫く指定しなく、“OK” をクリックする



7) ポップアップしたダイアログボックスに “Yes” を選択する。



- 8) “led.v” をダブルクリックして、ファイルが開ける。編集する。



- 9) “led.v” を編集する。ここで 32 ビットのレジスターtimer を定義した。0~49999999 (1秒) のループカウン트에使い、49999999 (1秒) に数えたら、レジスターtimer は0になり、四つのLED を反転する。こうして、消したLED は点灯される。逆に、元々点灯しているLED が消される。プログラミングしたコードは下のように :

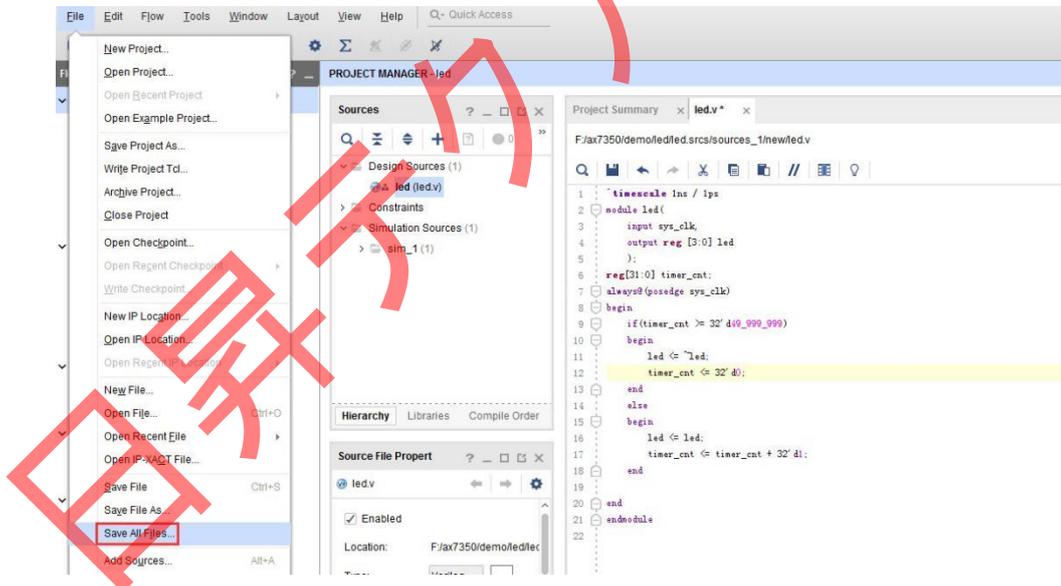
```
`timescale 1ns/1ps
module led(
    input sys_clk,
    output reg [3:0] led
);
```

```

reg[31:0] timer_cnt;
always@(posedge sys_clk)
begin
    if(timer_cnt >= 32'd49_999_999)
    begin
        led <= ~led;
        timer_cnt <= 32'd0;
    end
    else
    begin
        led <= led;
        timer_cnt <= timer_cnt + 32'd1;
    end
end endmodule

```

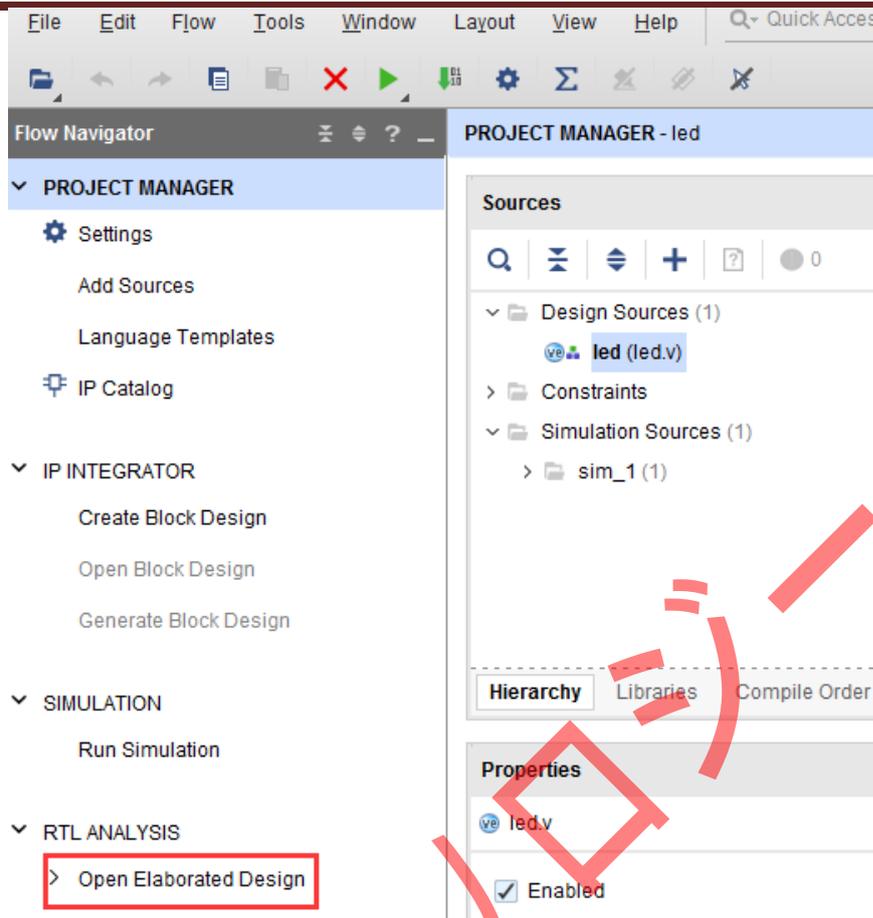
10) コードを編集したあと、メニュー “File→Save All Files” をクリックする。



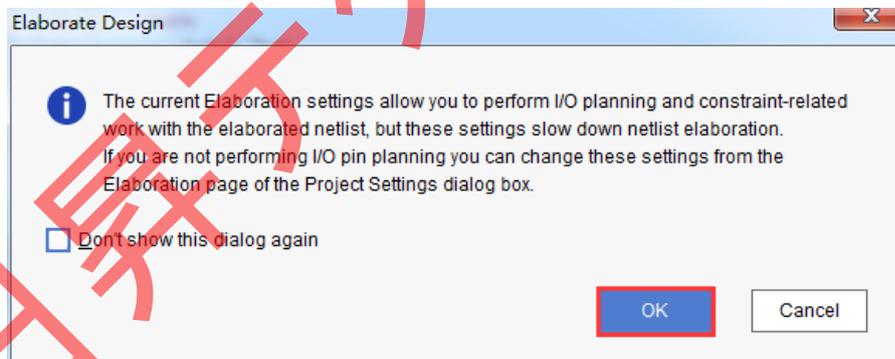
#### 4.4 ピン制約を追加

Vivada が使っている制約ファイルのフォーマットは xdc ファイルである。xdc ファイルは主にピンの制約、タイマーの制約、それとグループの約束である。ここでは led.v に入出力ポートを FPGA の実在なピンに分配することである。

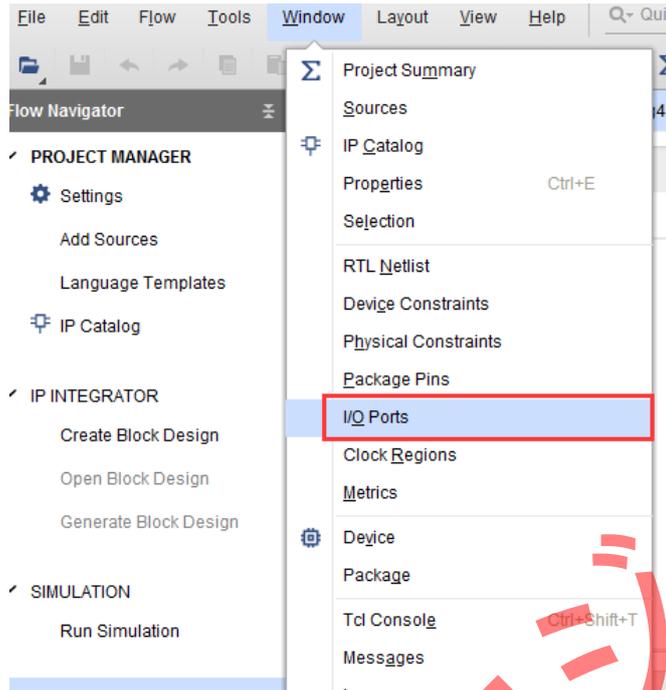
1) “Open Elaborated Design” をクリックする



2) ポップアップしたウィンドウに“OK”ボタンを押す



3) メニューで“Window -> I/O Ports”を選択する



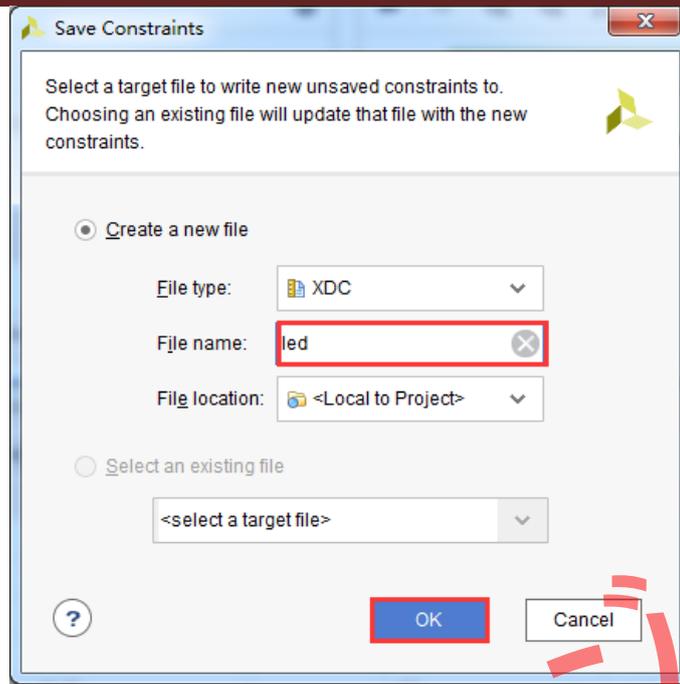
- 4) I/O Ports がポップアップされ、そこでピンの分配状況が見える

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
led (2)	OUT					(Multiple) LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[1]	OUT		A16			LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[0]	OUT		R7			LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Scalar ports (1)													
sys_clk	IN		Y9			LVC MOS33*	3.300				NONE	NONE	

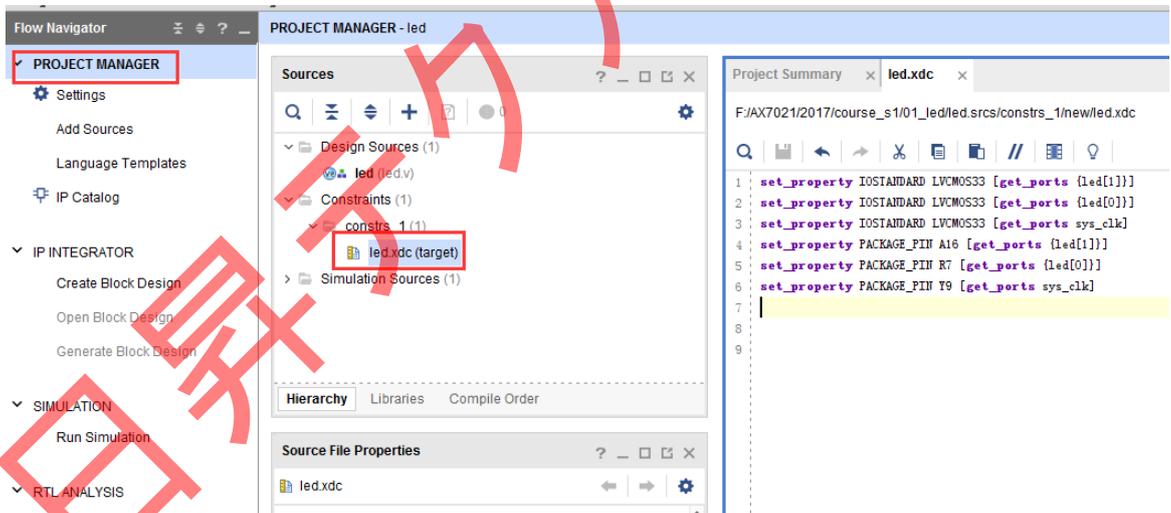
- 5) LED とタイマーにピン、レベル標準を分配する。完成したら、保存アイコンをクリックする。

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull
led (4)	OUT				35	LVC MOS33*	3.300		12	SLOW	NON
led[3]	OUT		J16		35	LVC MOS33*	3.300		12	SLOW	NON
led[2]	OUT		K16		35	LVC MOS33*	3.300		12	SLOW	NON
led[1]	OUT		M15		35	LVC MOS33*	3.300		12	SLOW	NON
led[0]	OUT		M14		35	LVC MOS33*	3.300		12	SLOW	NON
Scalar ports (1)											
sys_clk	IN		U18		34	LVC MOS33*	3.300				NON

- 6) ウィンドウがポップアップされ、制約の保存を要請する。ファイル名は“led”を書いて、ファイルタイプはデフォルトの“XDC”。“OK”をクリックする。



7) 先程生成したファイル“led.xdc”を開いて、TCL スクリプトが見える。これらの構文を分かれば、自分で led.xdc を編集することでピンを制約できる。



次は最も基礎的な XDC コーディングの文法を紹介する。普通な IO ポートはピン番号と電圧を制約すればいい。

ピンの制約は下のよう :

```
set_property PACKAGE_PIN "ピン番号" [get_ports "ポートネーム"]
```

レベル信号の制約は下のよう :

```
set_property IOSTANDARD "レベル標準" [get_ports "ポートネーム"]
```

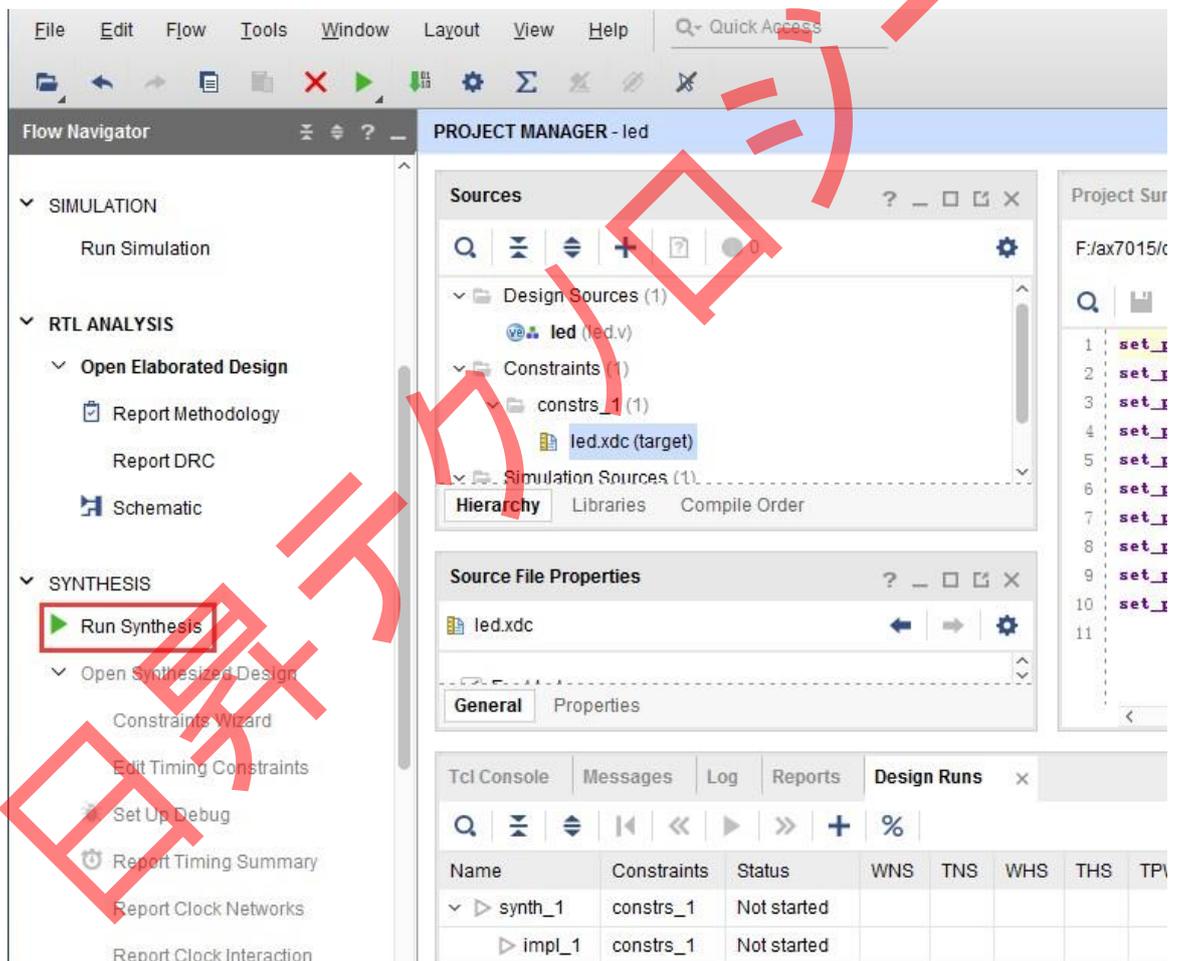
ここで注意するのは文字のフォントである。ポートネームはアレイだとしたら、 {} で括る。ポートネームは必ずソースコードの名前と一致して、キーワードと同じくしてはならない。

レベル標準で、“LVCMOS33”のあとの数字はFPGAのBANK電圧である。LEDがいるBANK電圧は3.3Vから、レベル標準は“LVCMOS”である。*Vivadoのデフォルトコマンドは全てのIOに正確なレベル標準とピン番号を分配することである。*

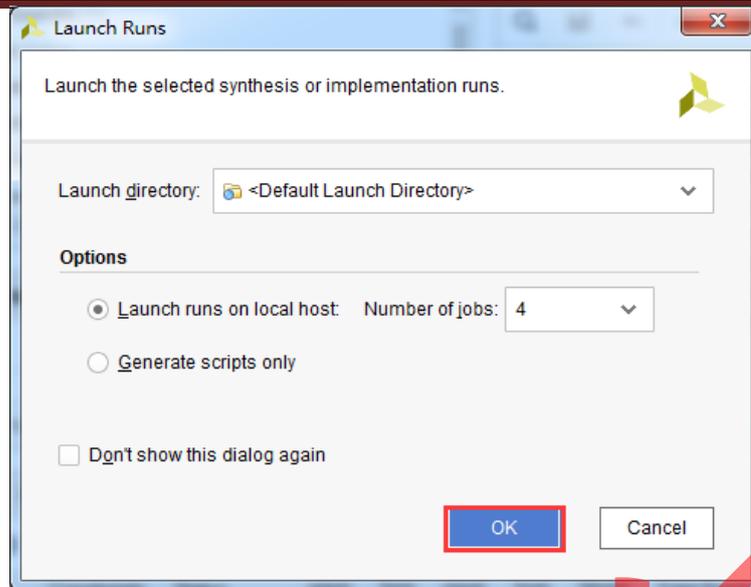
#### 4.5 シーケンス制約を追加

FPGAのデザインはピンの分配以外、もう一つ重要な制約ある。それはシーケンス制約である。ここはガイダンス方法でシーケンス制約を行う。

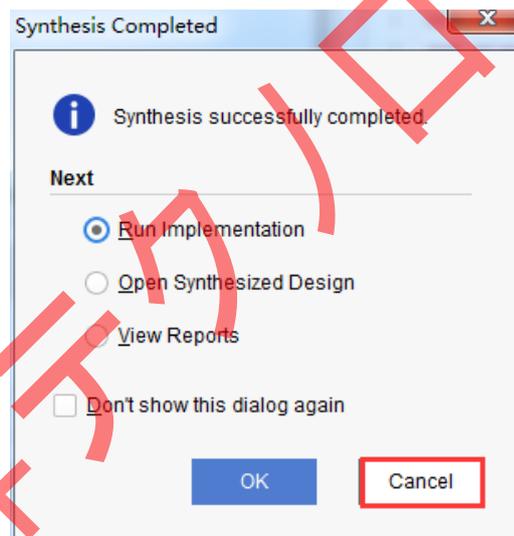
- 1) “Run Synthesis” をクリックして、統合を開始する。



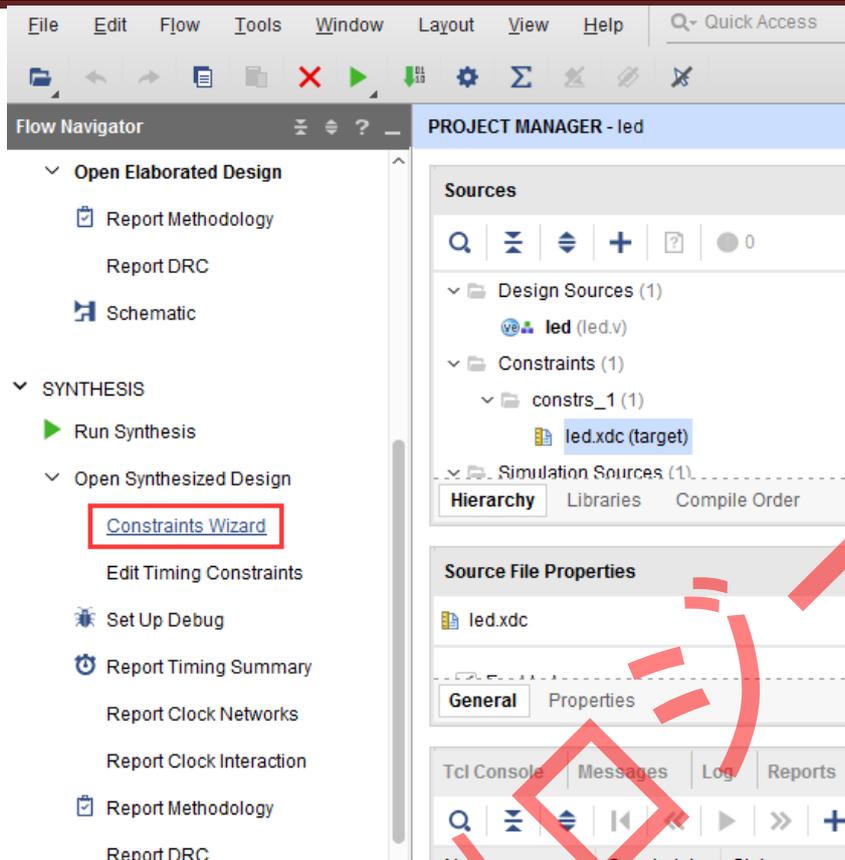
- 2) ダイアログがポップアップされて、“Ok” をクリックする



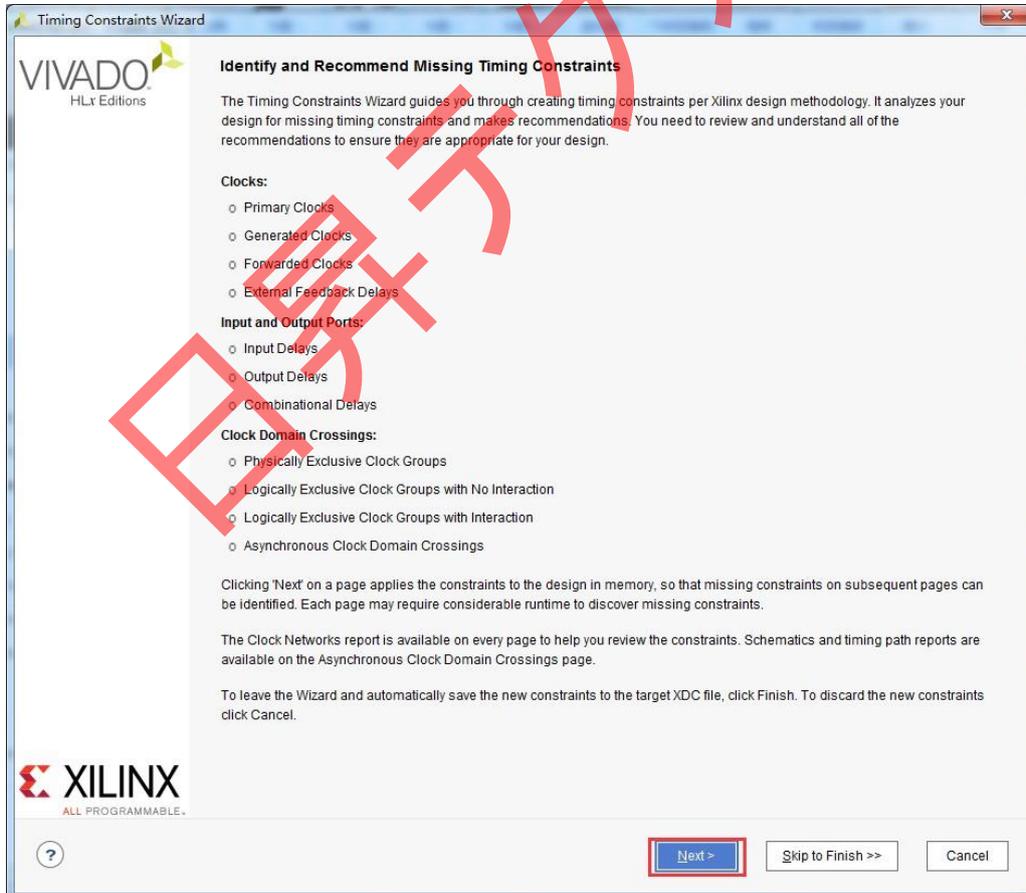
- 3) 統合を完成したら、“Cancel”をクリックする



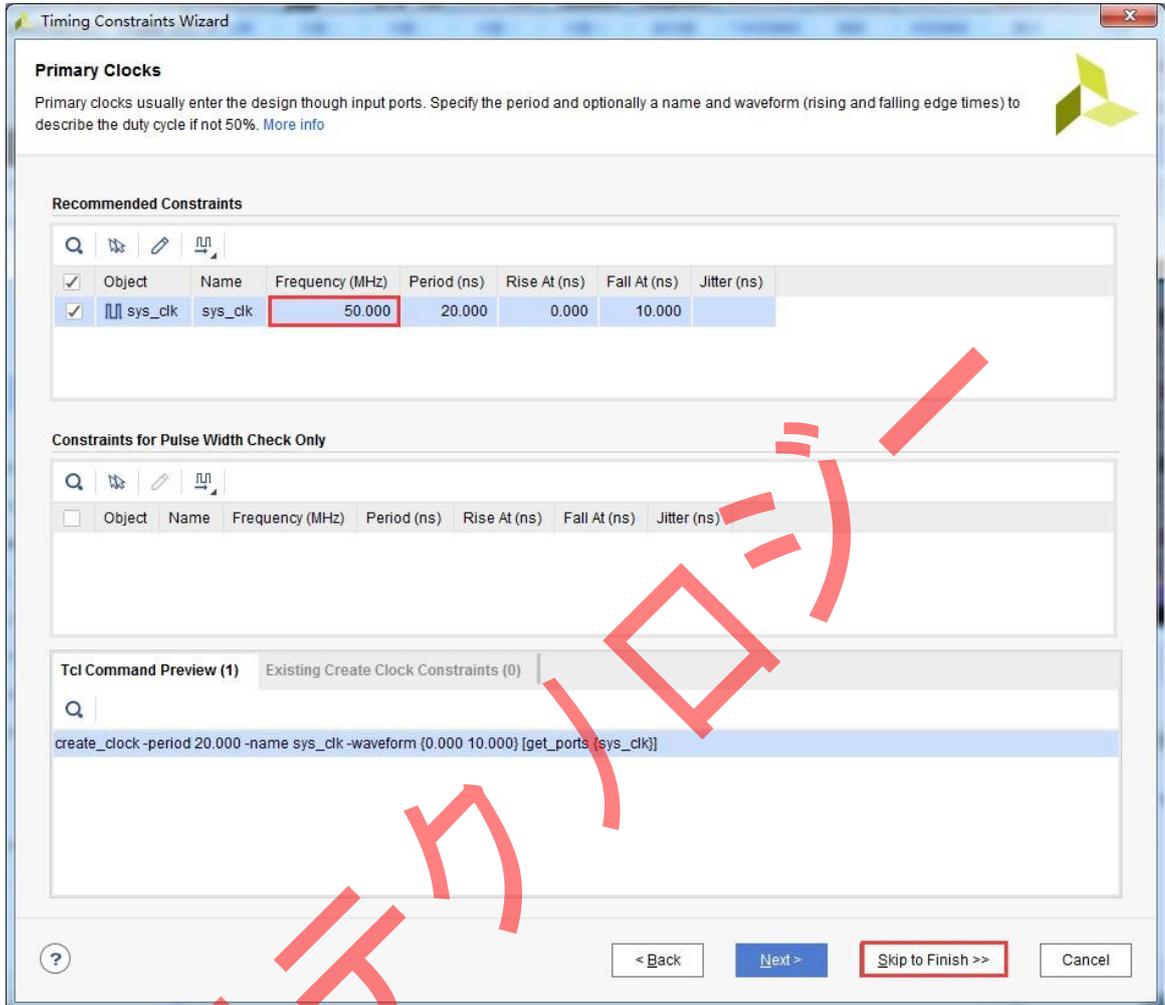
- 4) “Constraints Wizard”をクリックする



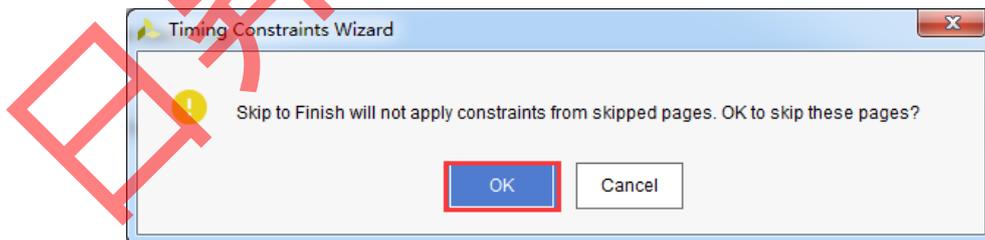
5) ポップアップされたウィンドウで“Next”をクリックする



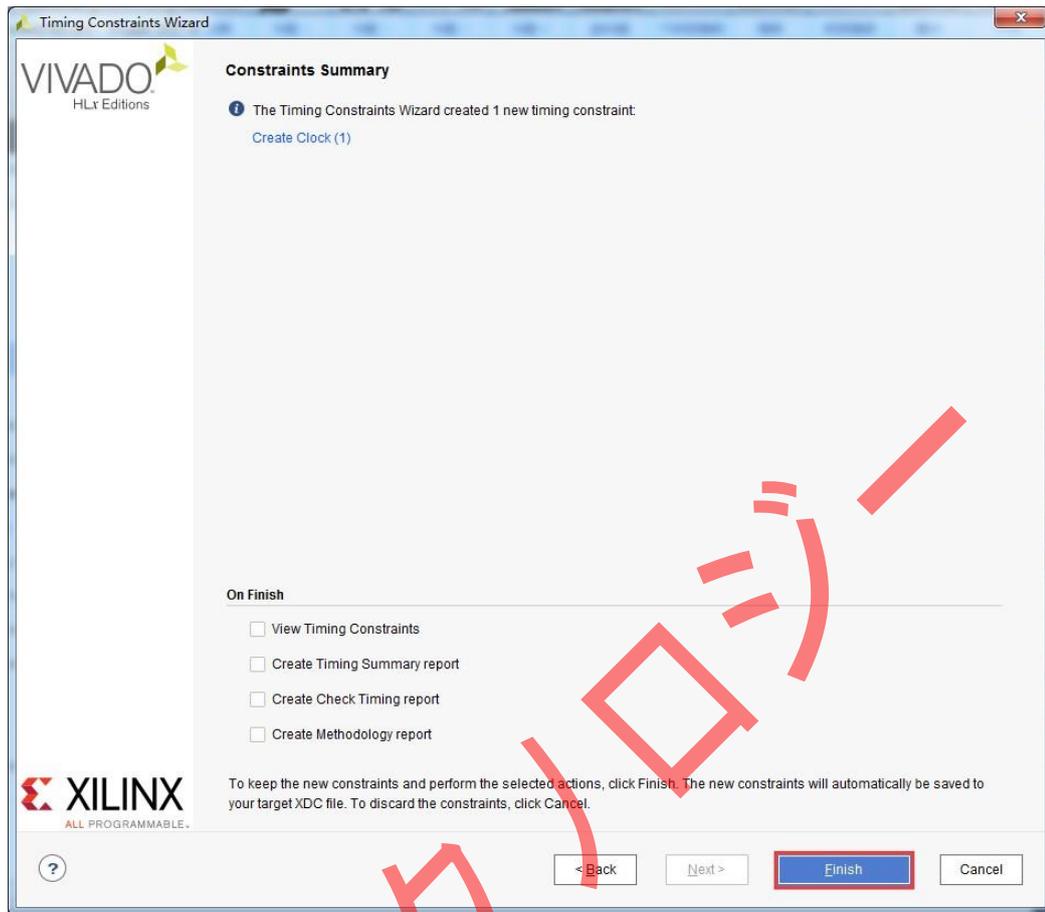
6) シーケンス制約ガイダンスを使ってデザイン中のタイマーを分析する。ここは“sys\_clk”頻度を 50MHz にして、“Skip to Finish” をクリックしシーケンス制約ガイダンスは終了。



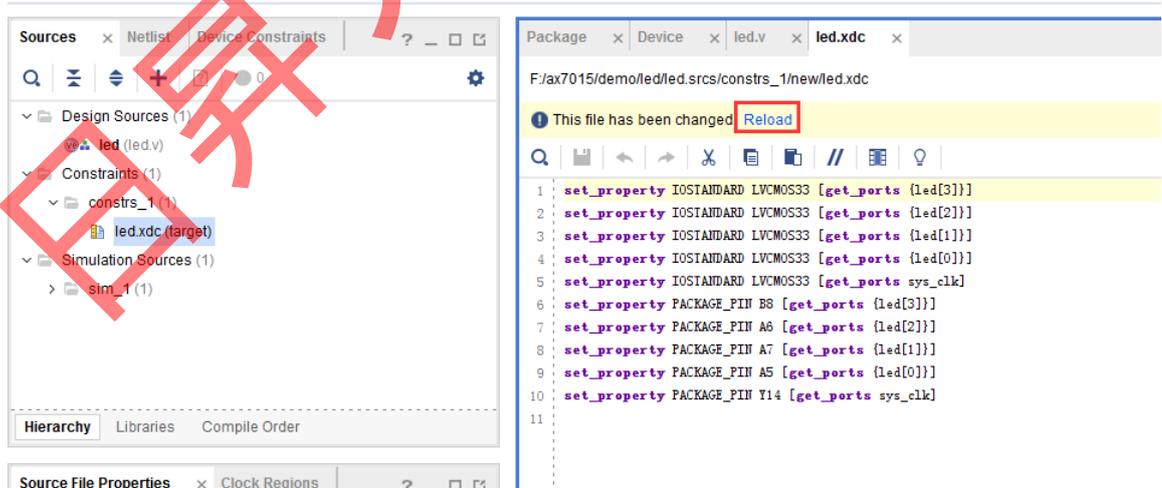
7) ダイアログがポップアップされて、“Ok” をクリックする



8) “Finish” をクリックする

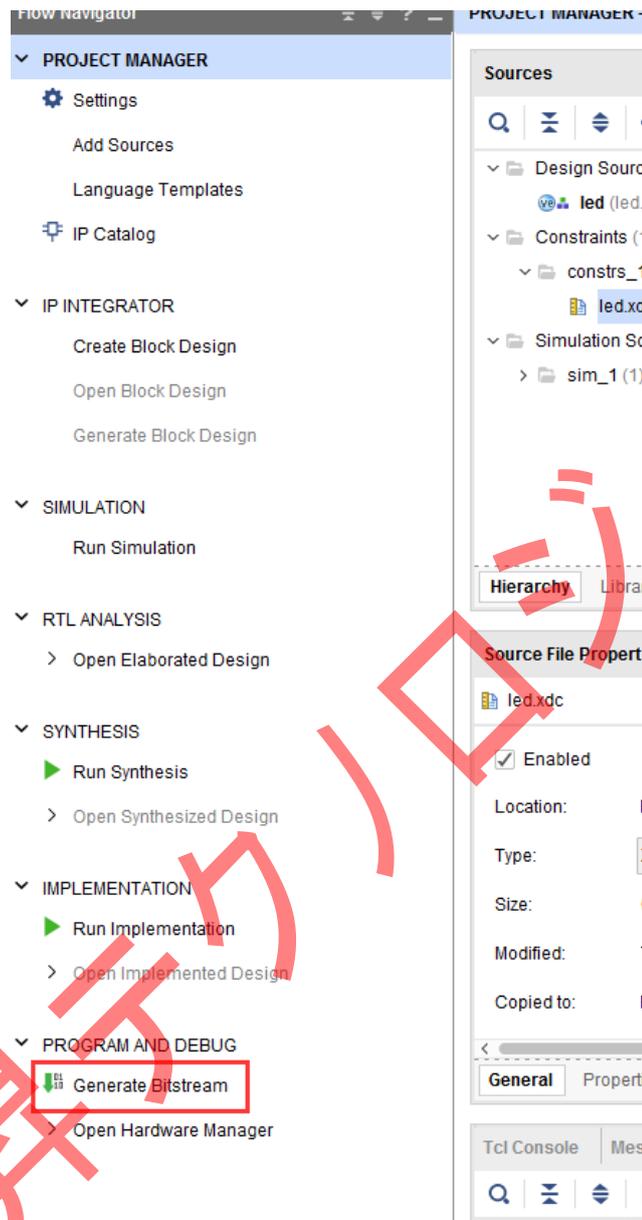


10) この時、led.xdc ファイルがアップデートされる。“Reload” をクリックして、改めてファイルをアップロードする

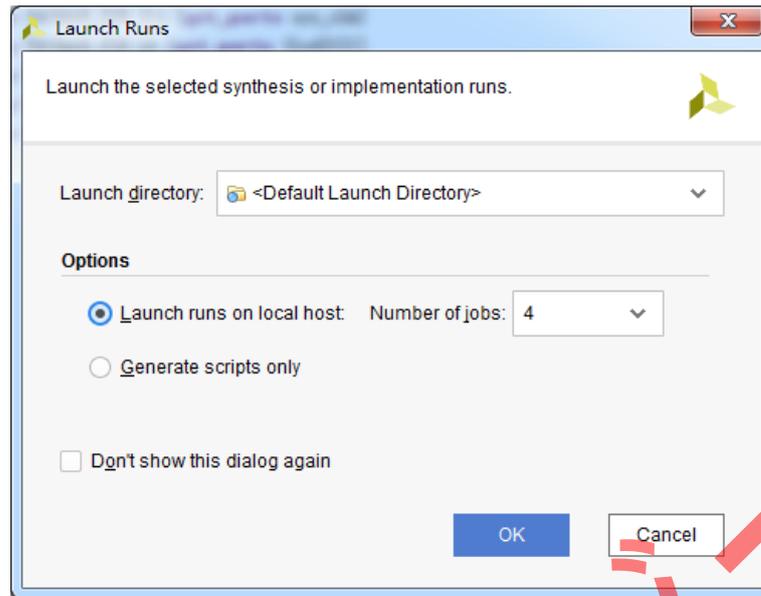


#### 4.6 BIT ファイルを生成する

1) プログラミング中は総合、レイアウト、BIT ファイルを生成することなどに細分化できる。ここでは、直接“Generate Bitstream”をクリックして、BIT ファイルを生成する。



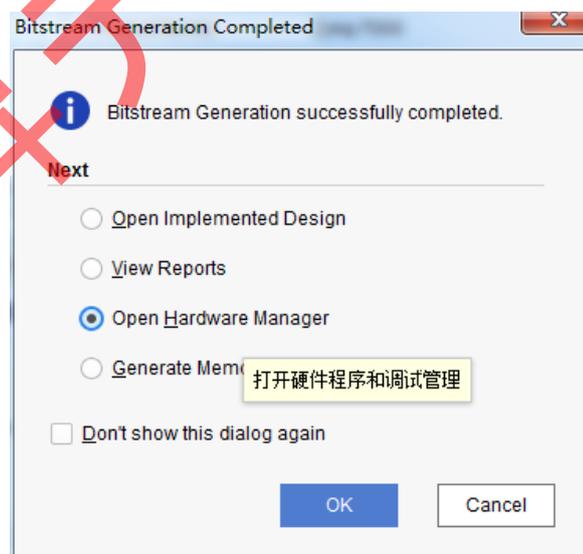
2) ポップアップされたダイアログに任務の数を選べる。ここと CPU コア数に関係している。一般的には、数字が大きければ大きいほど、プログラミングは早くなる。“OK” をクリックする。



3) コンパイルを開始して、右上のそこは状況を反映するメッセージ欄が見える。コンパイル中はウイルス防止ソフトにブロックされる可能性があって、コンパイルができない、又は長時間にコンパイルが成功できないことになる。

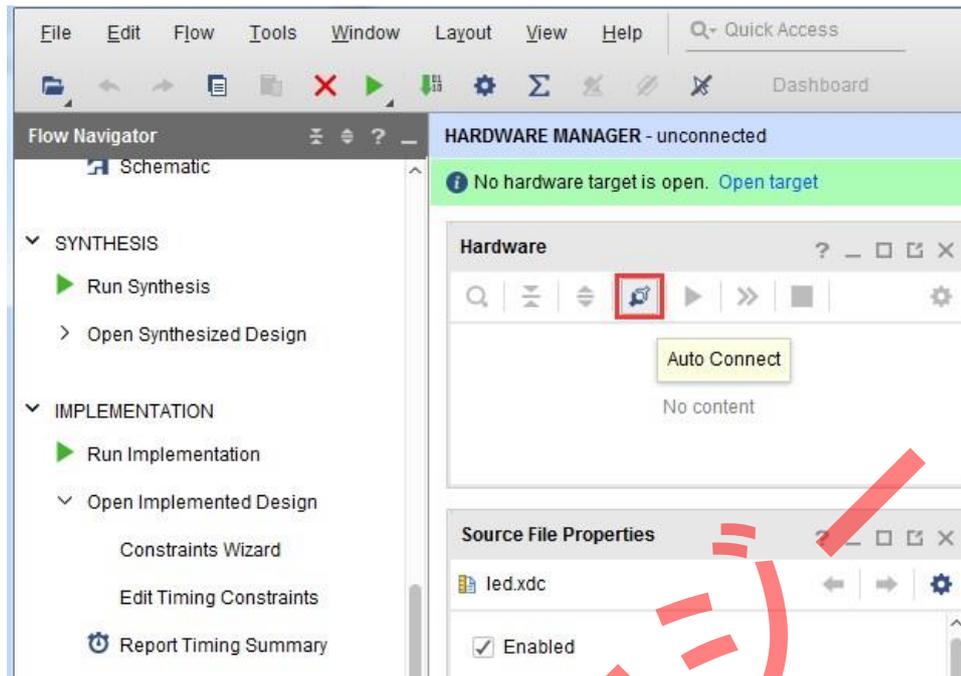


4) コンパイルはノーミスで完成したらダイアログがポップアップされて、引き続きの操作を選ぶ。“Open Hardware Manager”を選択し、“OK”をクリックする。もちろん“Cancel”も選べる。左のガイド欄で“Open Hardware Manager”をすればいい。

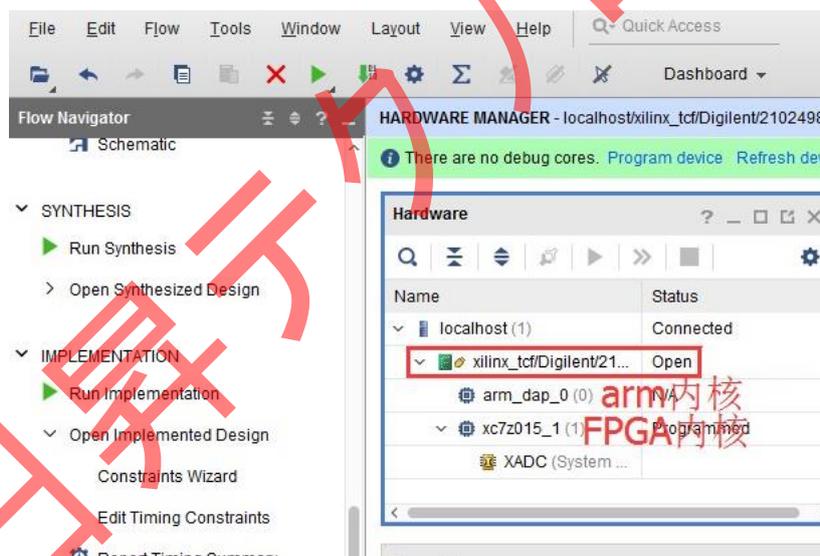


#### 4.7 ダウンロードとデバッグ

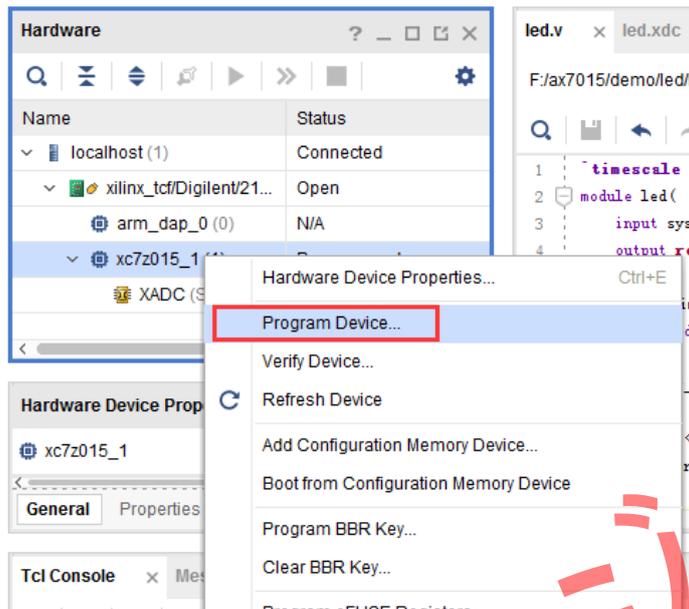
- 1) 開発ボードの JTAG インタフェースを接続して、電源を入れる。
- 2) “HARDWARE MANAGER” インタフェースで、“Auto Connect” をクリックして、自動的に設備と接続する。



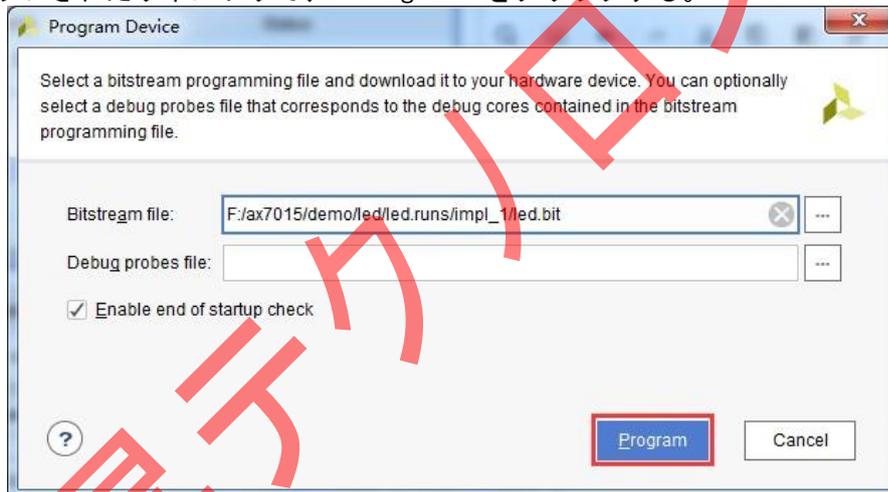
3) JTAG が ARM と FPGA のカーネルをスキャンする (図では xc7z015 が、AX7020 開発ボードの場合は xc7z020\_1 である)。また一つの XADC があって、システムの電圧と温度を検測できる。



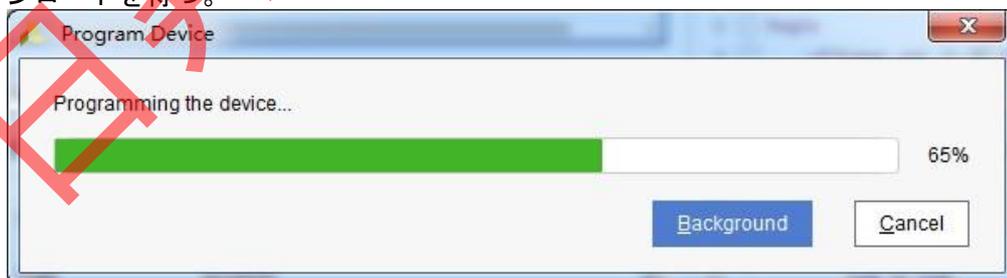
4) xc7z015\_1 (xc7z020\_1) を選択して、右クリックすると “Program Device” のオプションが出る。



- 5) ポップアップされたウィンドウで、“Program” をクリックする。



- 6) ダウンロードを待つ。



- 7) ダウンロード完成したあと、四つの LED は秒ことで変化し始める。ここまで、Vivado の簡単な流れ体験が終わる。後の章はこんなことを紹介する：プログラムを Flash に書き込む場合、PS システムの協力が必要になる。PL のみのプロジェクトは Flash に書き込めない。

## 第五章 HDMI 輸出実験

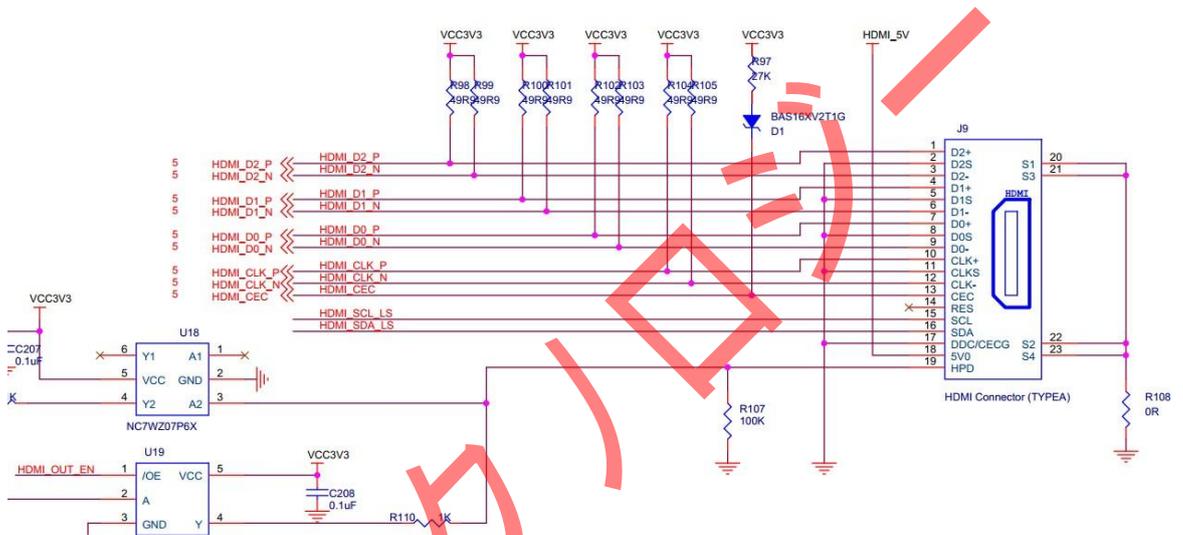
実験用 Vivado プロジェクトは “hdmi\_output\_test”。

前の章で、led の点滅実験を紹介した。これはただ Vivado の基本の開発流れを了解するため。本章の実験は LED 点滅事件よりすこし複雑で、内容は HDMI でのカラーバーの表示である。これも今後表示、ビデオ処理の基礎である。

### 5.1 ハードウェアの紹介

開発ボードは HDMI コーディングチップを使っていない。その代わりに、FPG の 3.3V 差分 IO を 直接 HDMI コネクタに接続する。

FPGA が 24 ビット RGB コードの出力で TMDS 差分信号を実現する。



### 5.2 Vivado プロジェクトを作成する

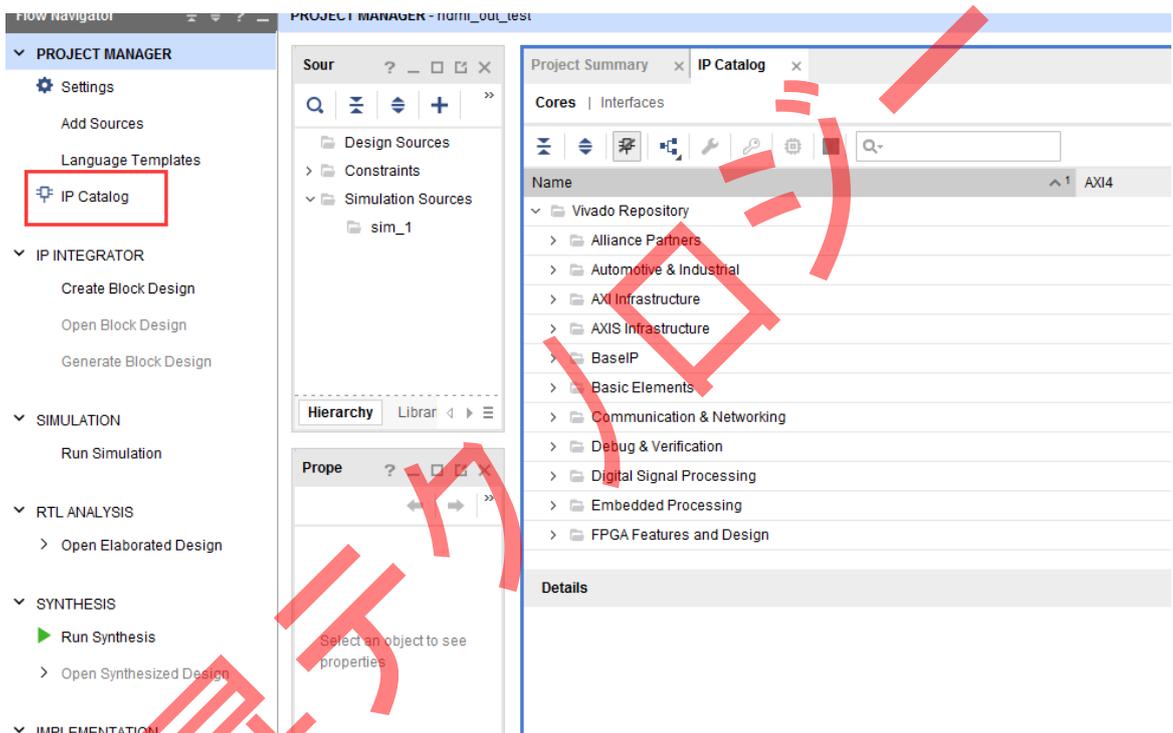
1) “hdmi\_output\_test” というプロジェクトを作成する。

#### 5.2.1 HDMI コンパイラーIP コアを追加する

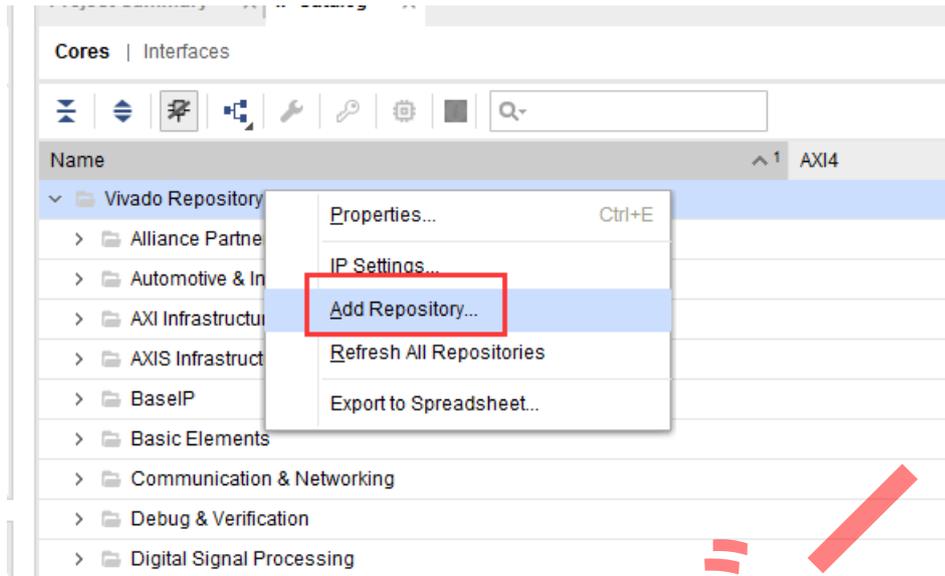
2) repo フォルダ（このフォルダは提供したサンプルプロジェクトにある）をプロジェクトメニューにコピーする。フォルダーに HDMI コンパイラーの IP がある

名称	修改日期	类型	大小
hdm_i_out_test.cache	2018/9/19 13:46	文件夹	
hdm_i_out_test.hw	2018/9/19 13:46	文件夹	
hdm_i_out_test.ip_user_files	2018/9/19 13:46	文件夹	
hdm_i_out_test.sim	2018/9/19 13:46	文件夹	
repo	2018/9/19 13:47	文件夹	
hdm_i_out_test.xpr	2018/9/19 13:46	Vivado Project Fi...	6 KB

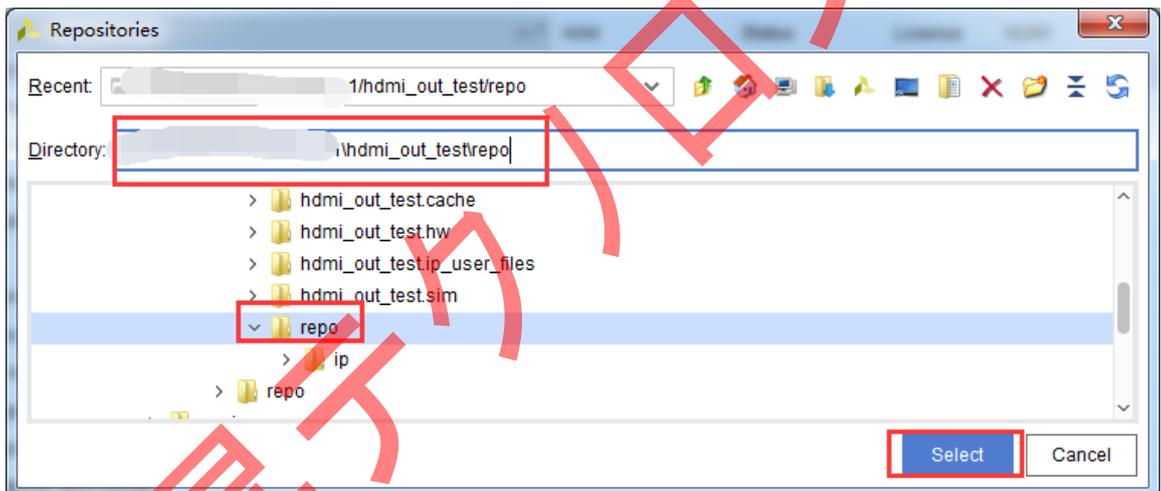
3) “IP Catalog” をクリックして、標準設置でこれらの IP はすべて Xilinx が提供されている。いまは第三者の IP、あるいは自分で作った IP を追加する。



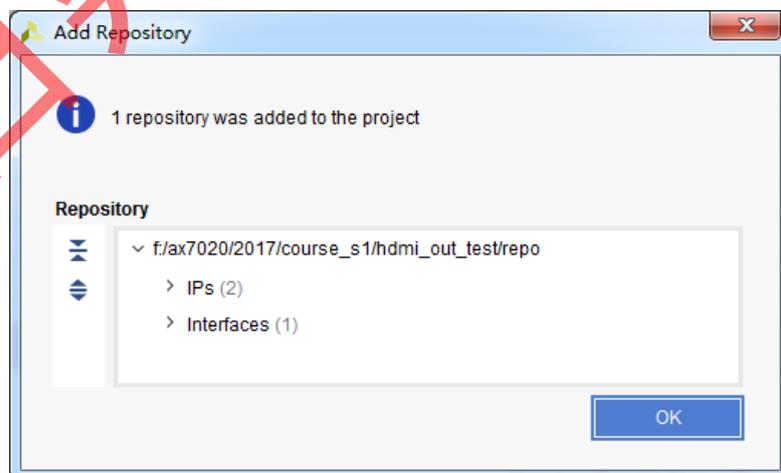
4) 右ボタンを押し、“Add Repository...” クリックする。



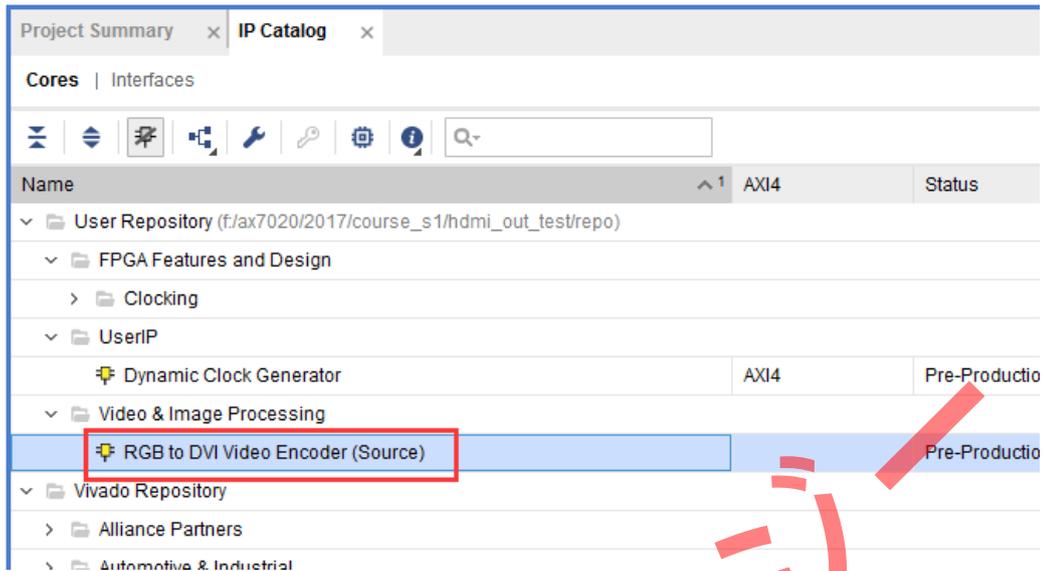
5) パスは先コピーした repo フォルターにする。



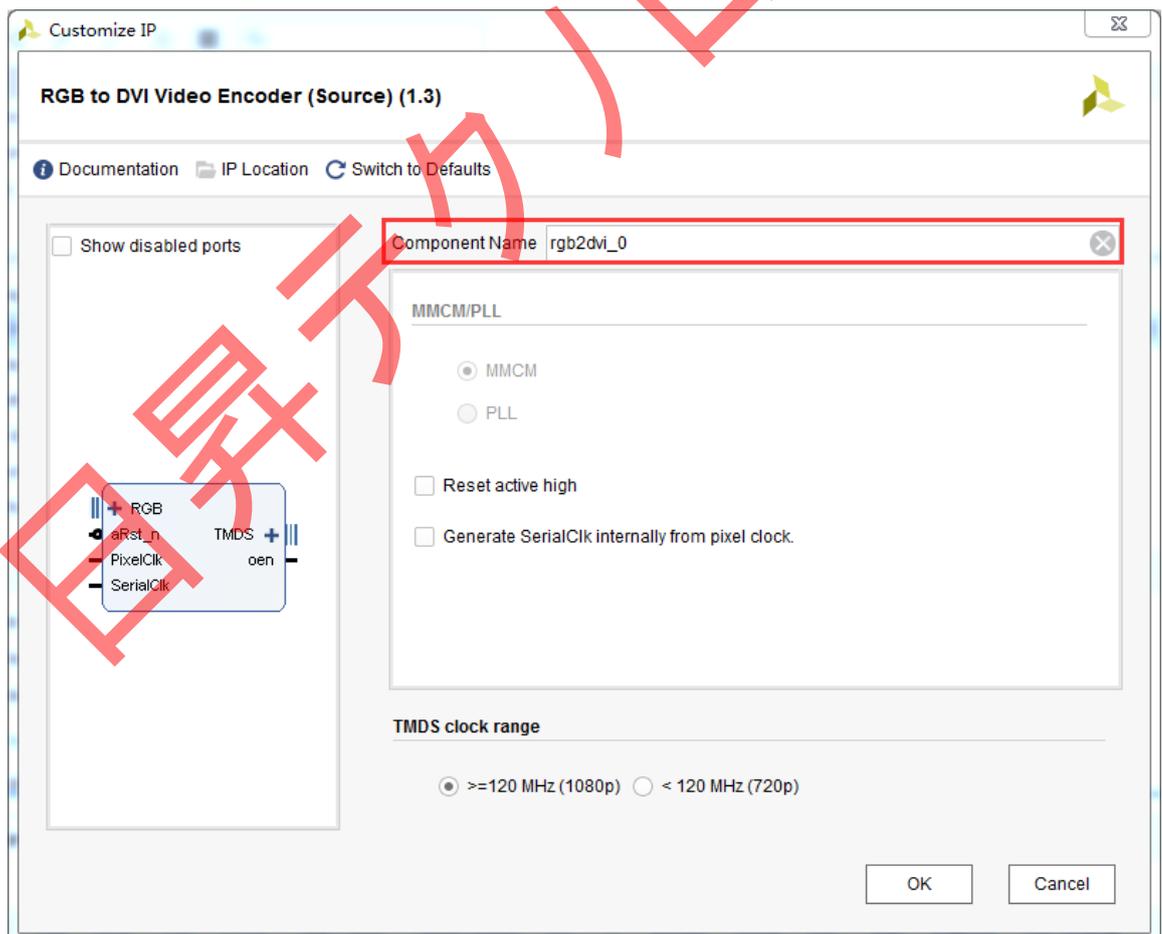
6) IP を追加して、成功に追加した IP の数を提示する。



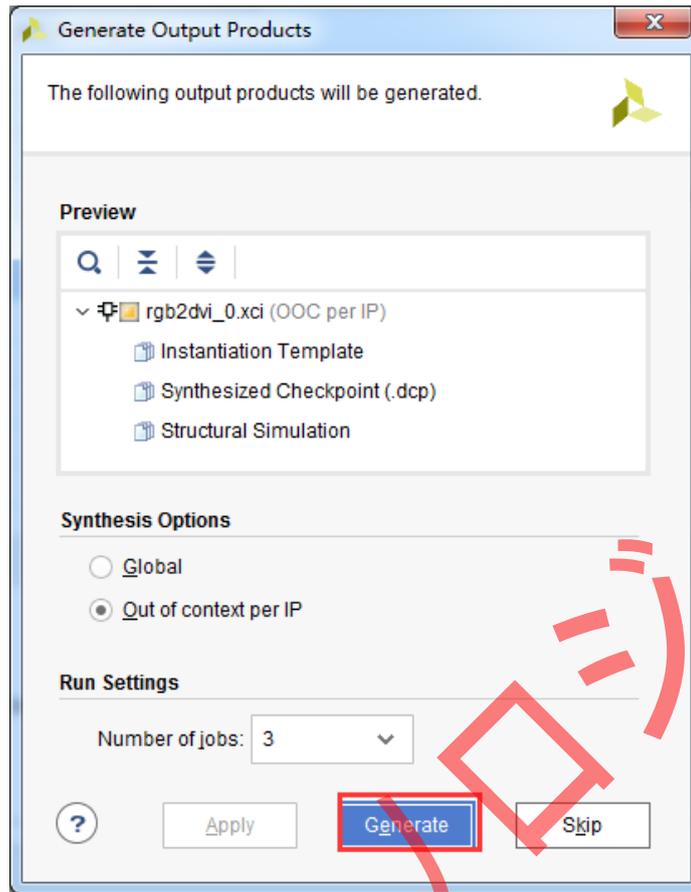
7) “RGB to DVI Video Encoder (Source)” を見つけ、ダブルクリックする。



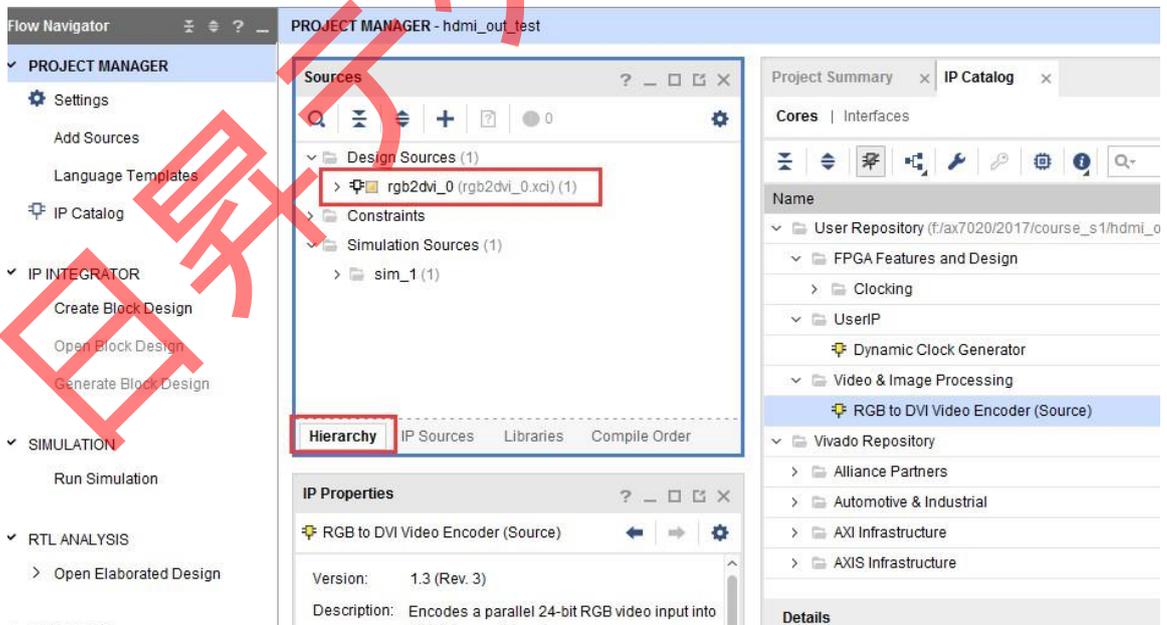
8) 下のウィンドウがポップアップされ、コンポーネント名 “Component Name” を変更せず、他のパラメータもそのまま。“OK” をクリックする。



9) “Generate Output Products” のウィンドウが出る。中でも “Number of jobs” はスレッド数で、数値が高ければスピードも早い。



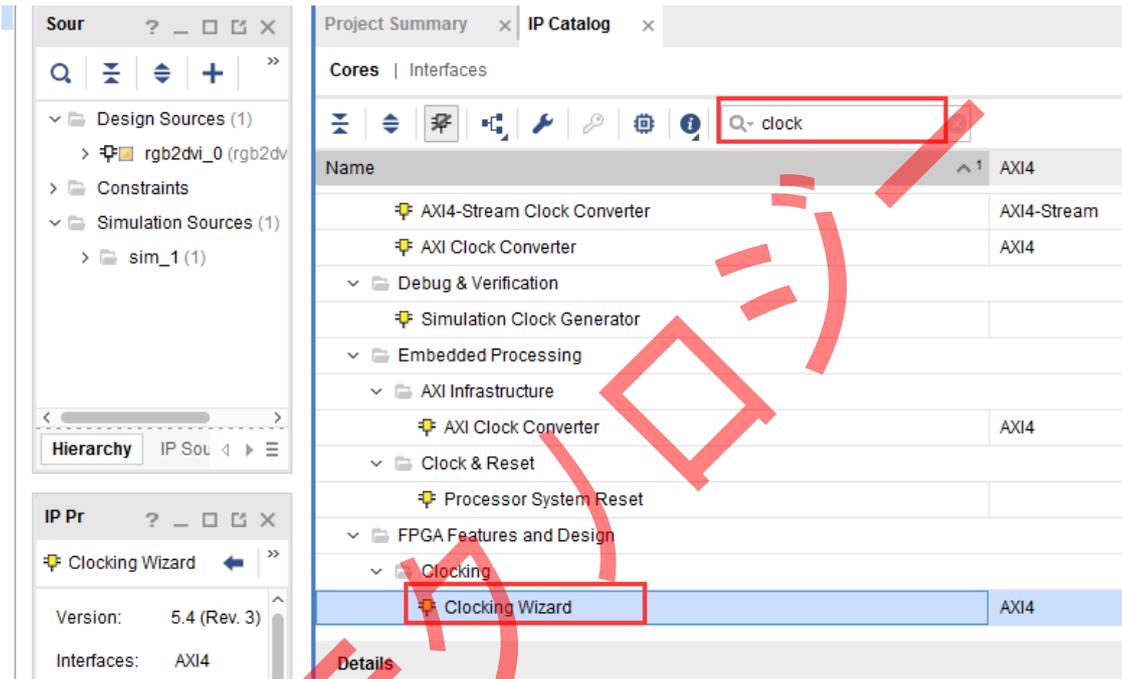
10) rgb2dvi\_0 というファイルが見える。



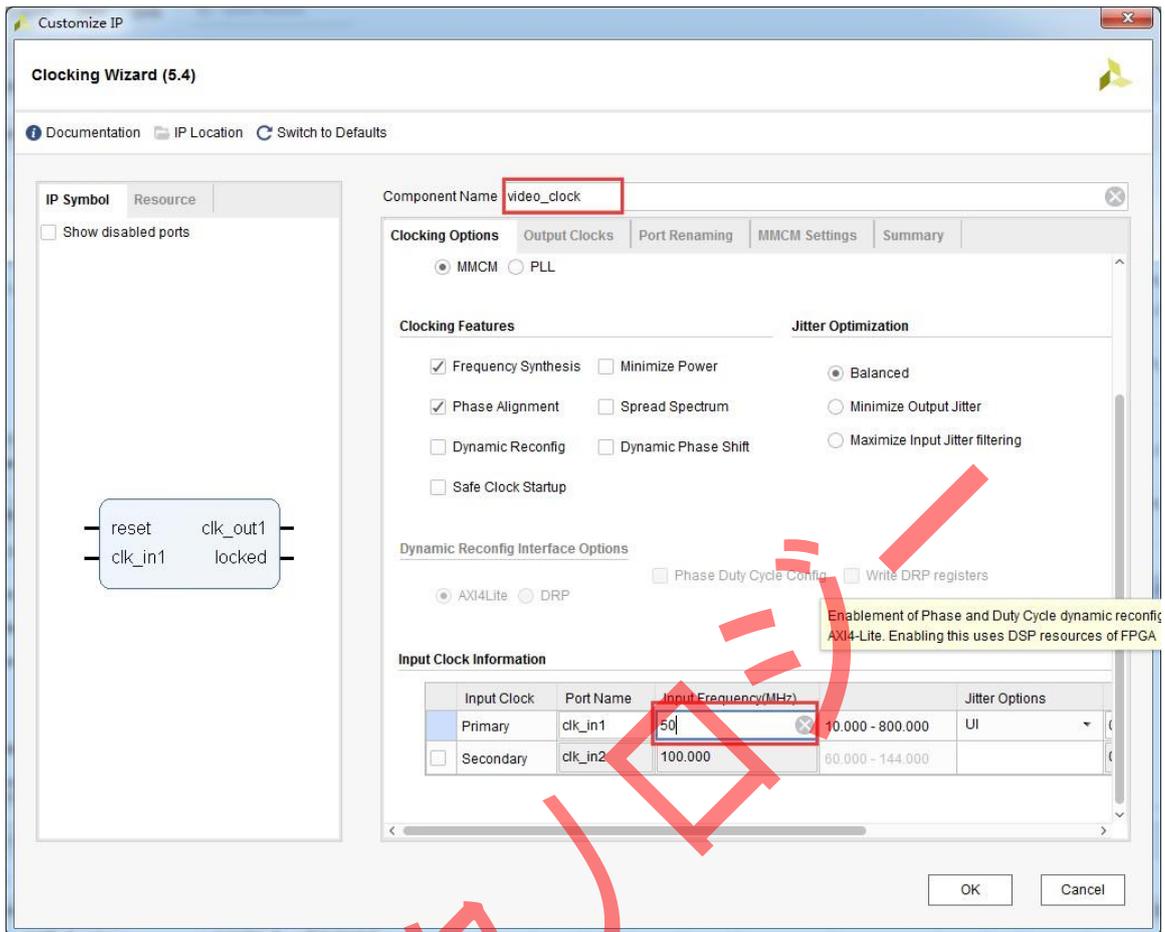
### 5.2.2 ピクセルクロックの PLL モジュールを追加する

HDMI コンパイラをドライブされる為、ピクセルクロックと 5 倍のピクセルクロックが必要になる。5 倍の方は 10:1 シリアル化に使用される。

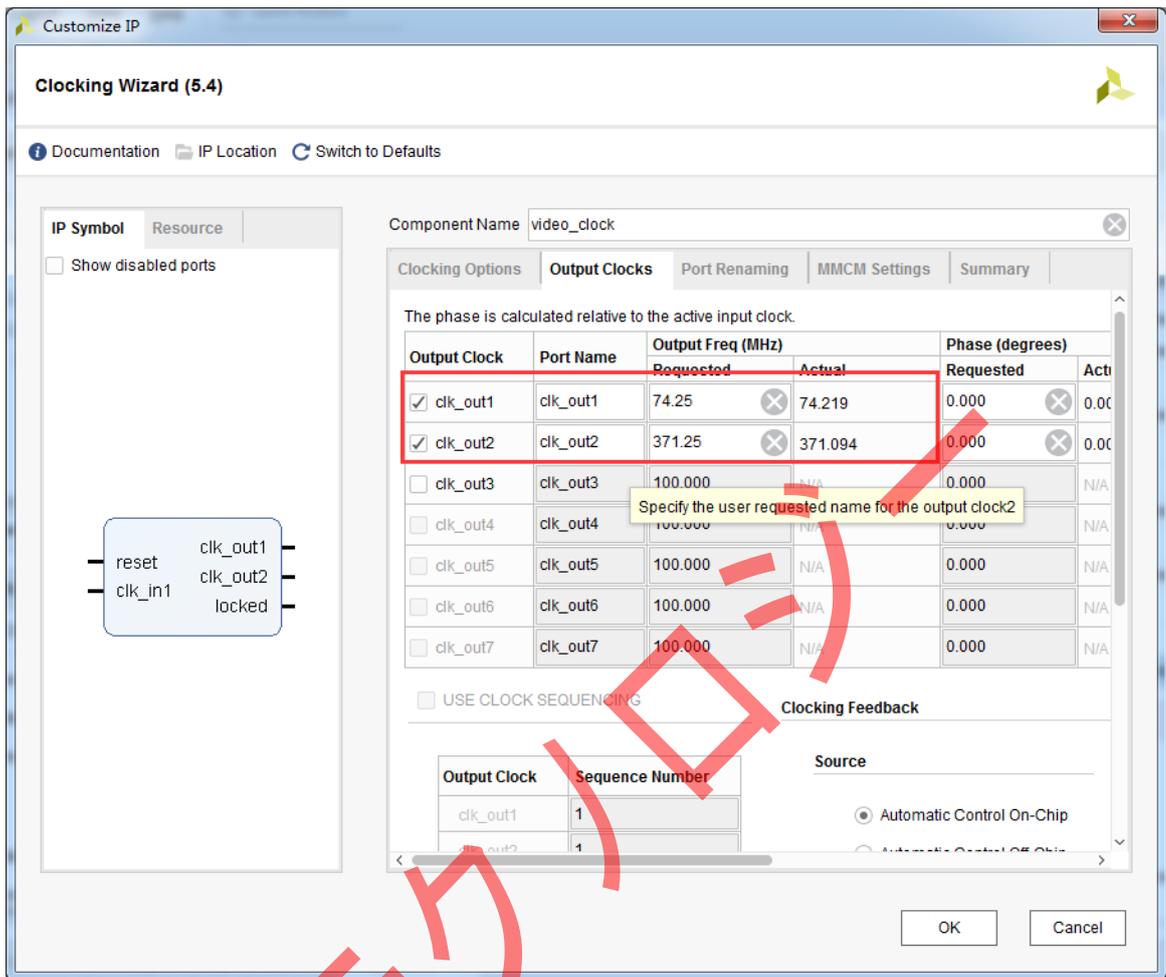
11) “IP Catlog” ウィンドウでキーワード “clock” をサーチして、“Clocking Wizard” をダブルクリックする。



12) 今回はコンポーネントに名前を付ける。这次给元件起个名字，“Component Name”欄で “video\_clock”、“clk\_in1”欄で50を入力する。この 50Mhz は開発ボード PL 側のクリスタル頻度と一致している。



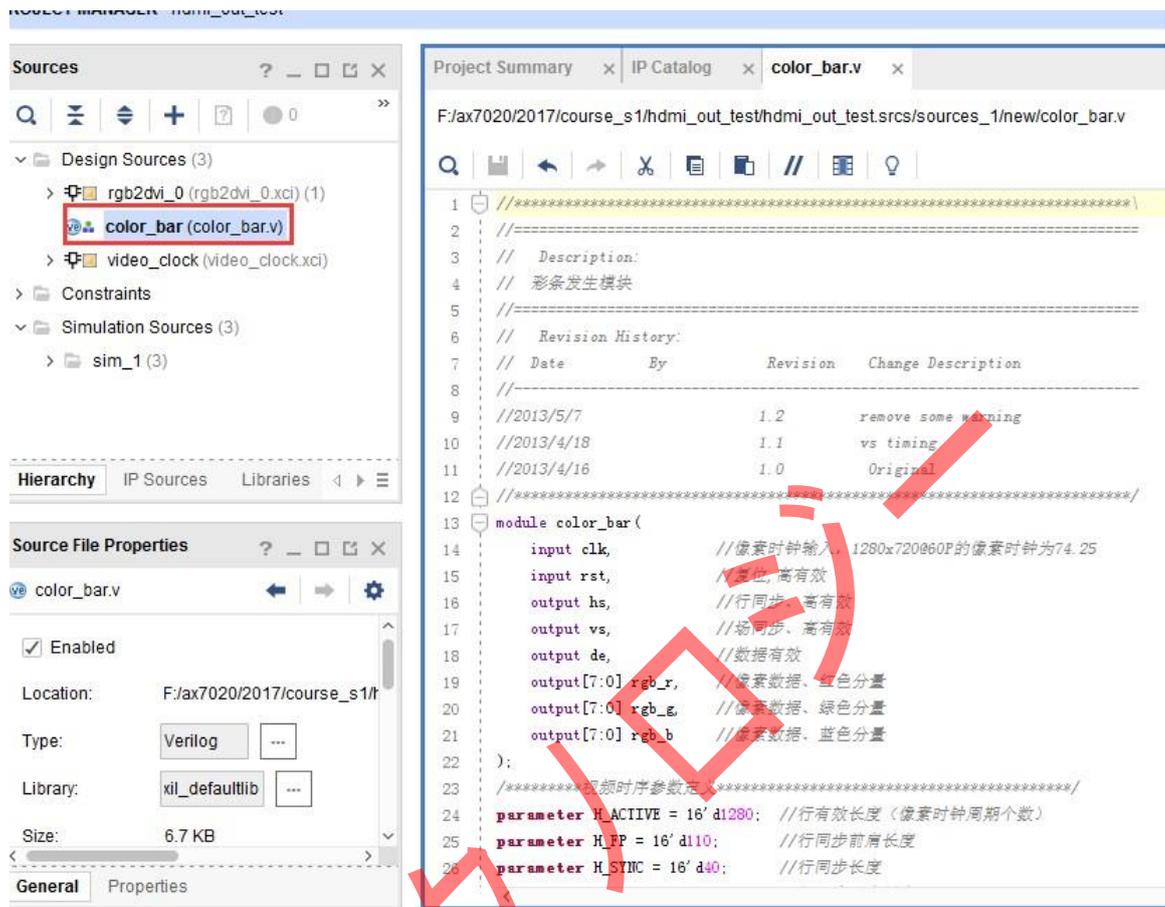
13) アウトプットクロック“clk\_out1”はビデオピクセルクロックに用いられる。ここで 74.25 を入力する。これは 1280x720@60 解析度のピクセルクロックである。各解析度のピクセルクロックが違って、ビデオ標準はかなり詳しいのだが、各ビデオ解析度のピクセルクロックを知っている。“clk\_out2”はコンパイラのシリアルかに使われている。ピクセルの 5 倍は 371.25 を入力する。次は“OK”をクリックして、IP を生成する。



### 5.2.3 カラーバー発生モジュールを追加する

14) カラーバー発生モジュールは一系列の Verilog コードで、ビデオシーケンスと用于产生视频时序和水平方向の八つのカラーバーを生成するに使われている。FPGA は本開発の

ポイントではないので、コードのことを詳しく説明しない。提供したサンプルにあるコードをコピーできる。



#### 5.2.4 トップモジュールを追加する

15) トップモジュールはカラーバー発生モジュール、HDMI コーディングモジュールとピクセルクロックをインスタンス化した。コードはサンプルにあるプロジェクトを参照する。

The screenshot displays the IDE interface for a Verilog project. On the left, the 'Sources' window shows a tree view with 'Design Sources (1)' containing 'top (top.v) (3)', which is highlighted with a red box. Below it, 'Constraints (1)' contains 'hdmi\_out\_test.xdc'. The 'Source File Properties' window for 'top.v' shows it is 'Enabled', located at 'F:/ax7020/2017/course\_s1/hdmi...', with a 'Verilog' type and 'xil\_defaultlib' library. The 'Project Summary' window on the right shows the Verilog code for 'module top', including input 'sys\_clk', output 'hdmi\_oen', and various IMDS data outputs, along with wire declarations and an instantiation of 'color\_bar hdmi\_color\_bar'.

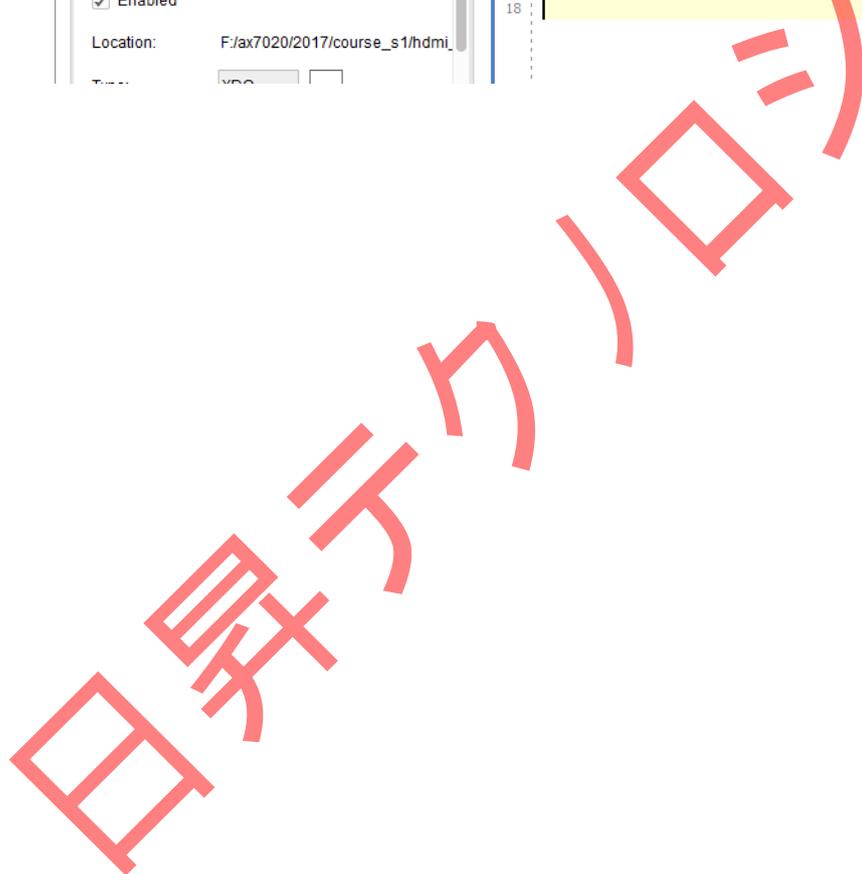
### 5.3 XDC 制約ファイルを追加する

下の xdc 制約ファイルをプロジェクトに追加する、ファイルにクロックと HDMI に関するピンも加わっている。



The screenshot shows the Vivado IDE interface. On the left, the 'Sources' pane displays a project hierarchy with 'hdmi\_out\_test.xdc' highlighted under 'constrs\_1'. Below it, the 'Source File Properties' pane shows 'hdmi\_out\_test.xdc' is 'Enabled' and its location is 'F:/ax7020/2017/course\_s1/hdmi...'. The main editor window shows the content of 'hdmi\_out\_test.xdc', which contains a list of constraints for an HDMI output test. The constraints include setting package pins, creating a clock, and configuring IOSTANDARD and IMDS\_33 properties for various pins and data lines.

```
1 set_property PACKAGE_PIN U18 [get_ports {sys_clk}]
2 set_property IOSTANDARD LVCMOS33 [get_ports {sys_clk}]
3 create_clock -period 20.000 -waveform {0.000 10.000} [get_ports sys_clk]
4 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_clk_n}]
5 set_property PACKAGE_PIN N18 [get_ports {IMDS_clk_p}]
6 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_clk_p}]
7 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_n[0]}]
8 set_property PACKAGE_PIN V20 [get_ports {IMDS_data_p[0]}]
9 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_p[0]}]
10 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_n[1]}]
11 set_property PACKAGE_PIN T20 [get_ports {IMDS_data_p[1]}]
12 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_p[1]}]
13 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_n[2]}]
14 set_property PACKAGE_PIN N20 [get_ports {IMDS_data_p[2]}]
15 set_property IOSTANDARD IMDS_33 [get_ports {IMDS_data_p[2]}]
16 set_property PACKAGE_PIN V16 [get_ports hdmi_oen]
17 set_property IOSTANDARD LVCMOS33 [get_ports hdmi_oen]
18
```



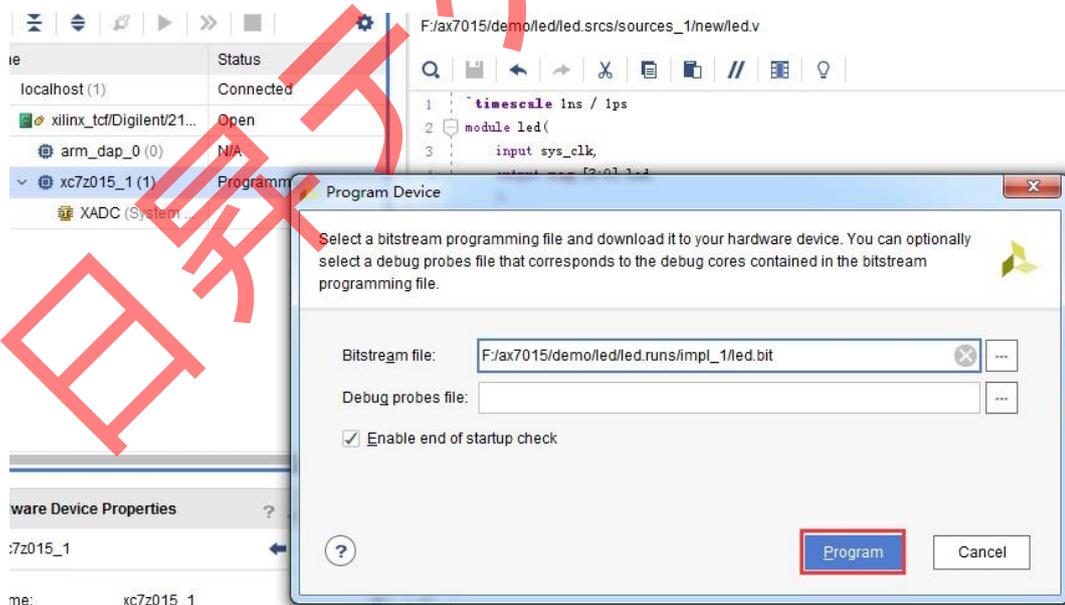
```

set_property PACKAGE_PIN U18 [get_ports {sys_clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {sys_clk}]
create_clock -period 20.000 -waveform {0.000 10.000} [get_ports
sys_clk] set_property IOSTANDARD TMDS_33 [get_ports TMDS_clk_n]
set_property PACKAGE_PIN N18 [get_ports TMDS_clk_p]
set_property IOSTANDARD TMDS_33 [get_ports TMDS_clk_p]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[0]}]
set_property PACKAGE_PIN V20 [get_ports {TMDS_data_p[0]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[0]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[1]}]
set_property PACKAGE_PIN T20 [get_ports {TMDS_data_p[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[2]}]
set_property PACKAGE_PIN N20 [get_ports {TMDS_data_p[2]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[2]}]
set_property PACKAGE_PIN V16 [get_ports hdmi_oen]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_oen]

```

## 5.4 ダウンロードとデバッグ

プロジェクトを保存し bit ファイルにコンパイルする、HDMI インタフェースを HDMI モニターに接続する。注意するのは、ここで 1280x720@60Hz を使うので、自分のモニターはこの解析度をサポートできることを確保してください。



ダウンロード後モニターは下の画面を表示する。



## 5.5 実験のまとめ

本実験は初歩にビデオ表示を接触した。ビデオ知識関わっているが、zynq を学ぶキープointではないので、詳しく紹介していない。しかし、zynq はビデオ紹介領域でよく使われているから、勉強者は良き基礎知識が必要である。実験中は PL だけ使用して HDMI チップをドライブして、第三者がカスタマイズする IP の使い方を初歩に学んだ。引き続きはカスタマイズ IP の作り方を学ぶ。

## 第六章 ARM を体験

実験用 Vivado プロジェクトは “ps\_hello”。

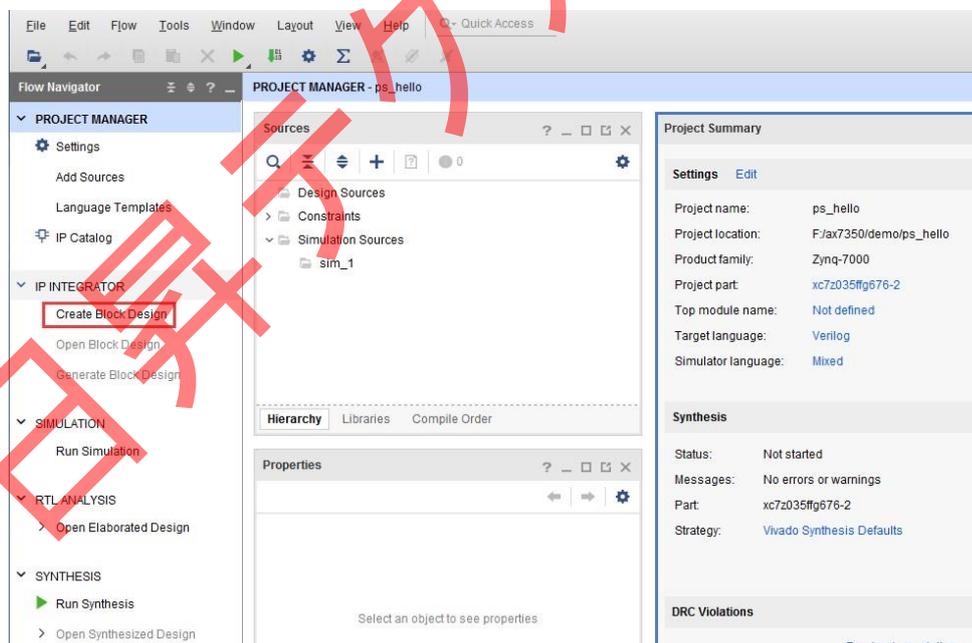
前の実験は PL 側で行われている。はっきり見えるのは、普通の FPGA 開発流れと何の区別がないことである。ZYNQ の優勢は FPGA と ARM を合理的に結ぶことなので、開発エンジニアにより高く要求している。本章から、ARM いわゆる PS の利用を始める。今回は簡単なシリアルプリントで Vivado SDK と PS の特性を体験する。

### 6.1 ハードウェアの紹介

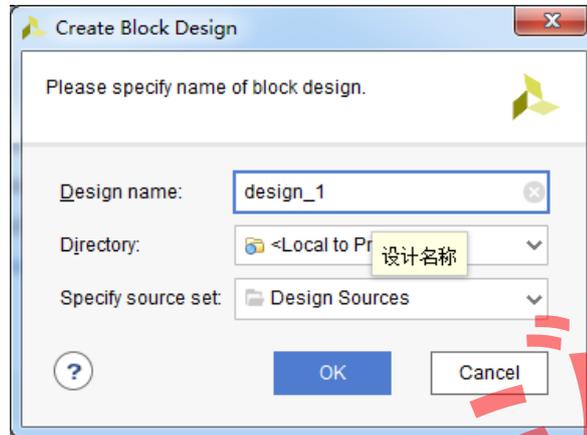
原理図から見えるのは、ZYNQ チップは PL と PS に分けられる。PS 側の IO 分配は相対的に固定で、任意で分配するにはいけない。Vivado ソフトでピンの配りも必要ない。本実験では PS しか使っていないが、Vivado プロジェクトの作成は需要で、PS ピンの配置に用いられる。

### 6.2 Vivado プロジェクトを作成

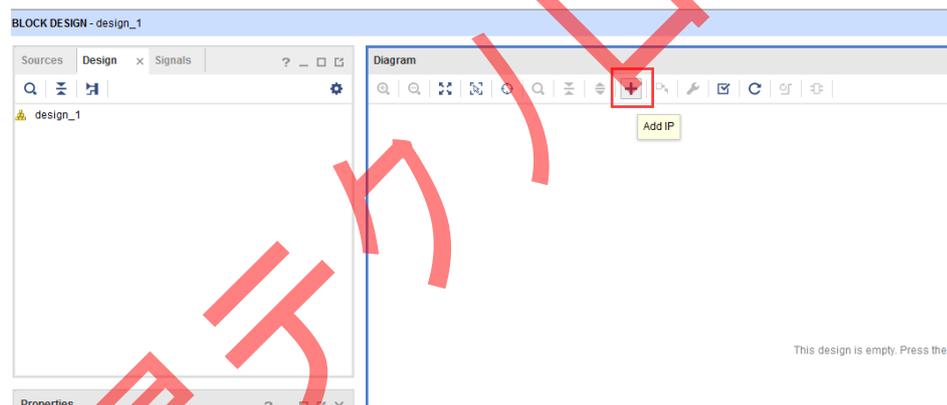
- 1) “ps\_hello” というプロジェクトを作成する。
- 2) “Create Block Design” をクリックし、Block デザインを作成する。



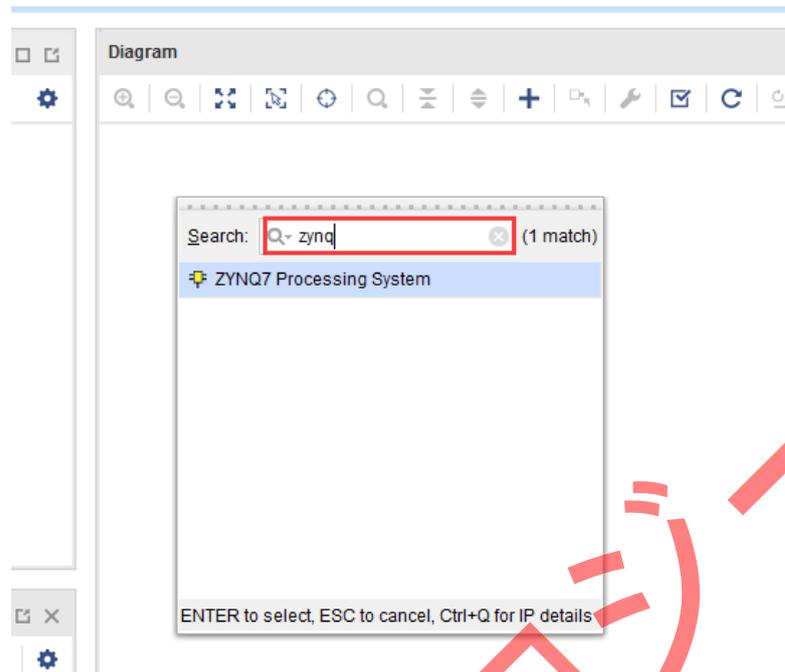
3) ここで“Design name”は変更しないで、デフォルトの“design\_1”にする。変更の需要があったら、名前はできるだけ短くする。そうしなければ、Windows でのコンパイルは問題が出る。



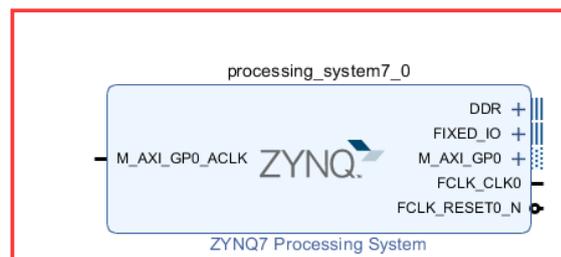
4) “Add IP” のショートアイコンをクリックする。



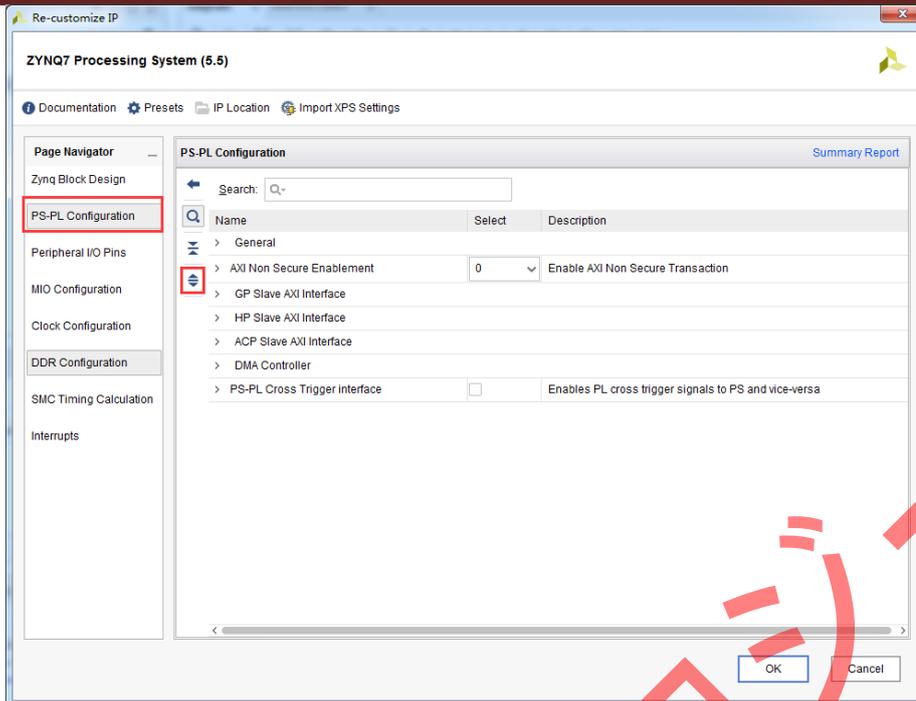
5) “zynq”を検索する。検索したリストで“ZYNQ7 Processing System”をダブルクリックする。



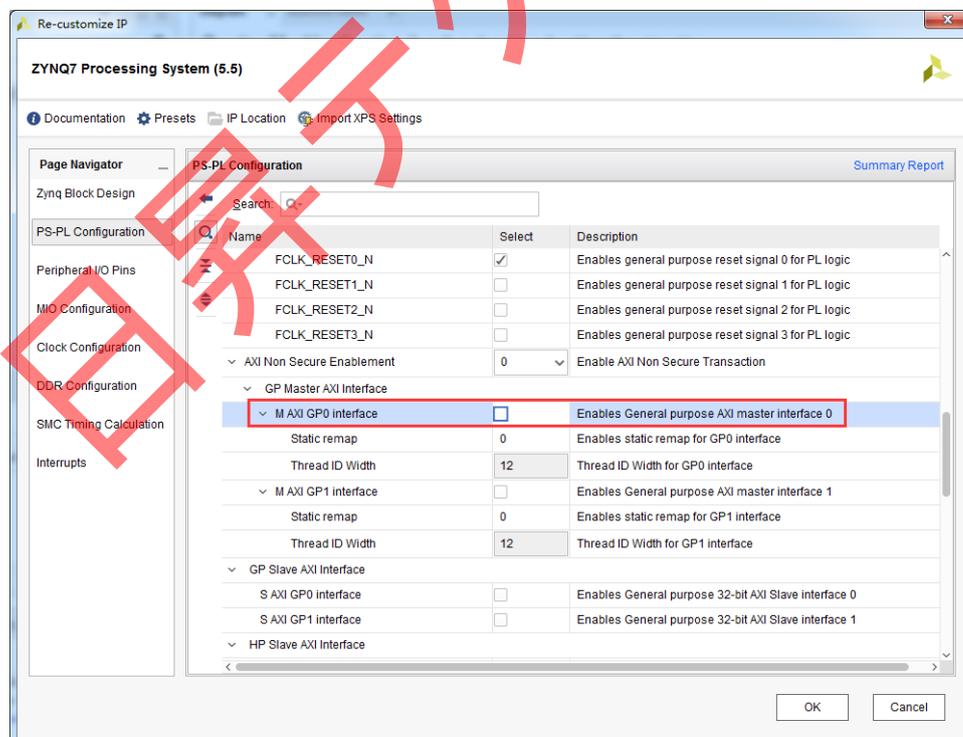
6) Block 図の “processing\_system7\_0” をダブルクリックして、関係するパラメータを追加する。



7) “PS-PL Configuration” で全ての項目を表にする。

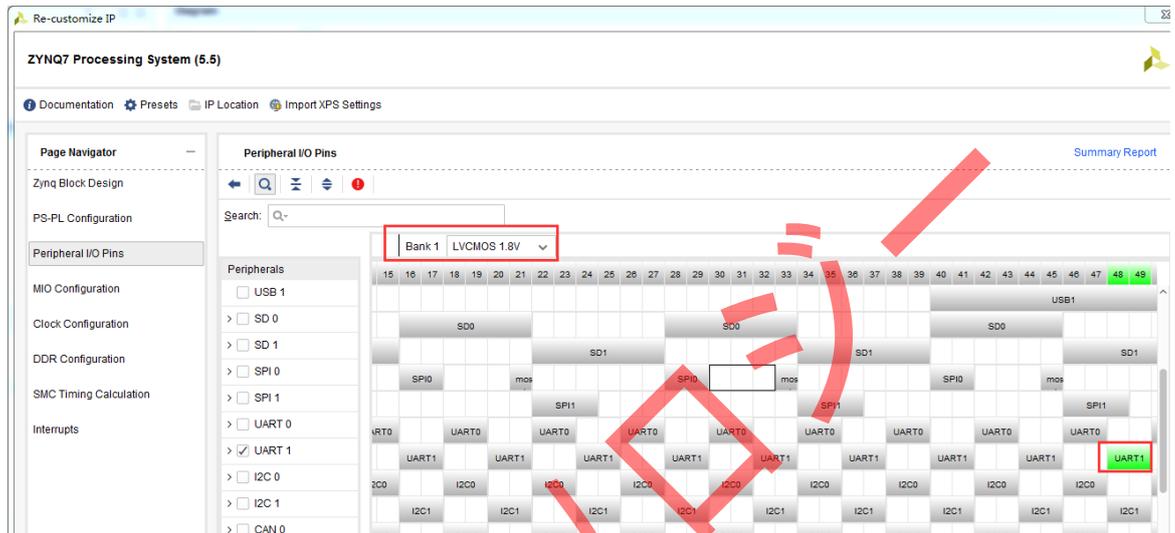


8) “M AXI GP0 interface” インタフェースを取り消す。このインタフェースはPL側のAXIインタフェースペリフェラルを拡張できる。この原因で、PLはPSとデータ交換をするなら、AXIバスインターフェースプロトコルに沿って交換を行う。Xilinxは大量なAXIインタフェースのIPコアを提供した。



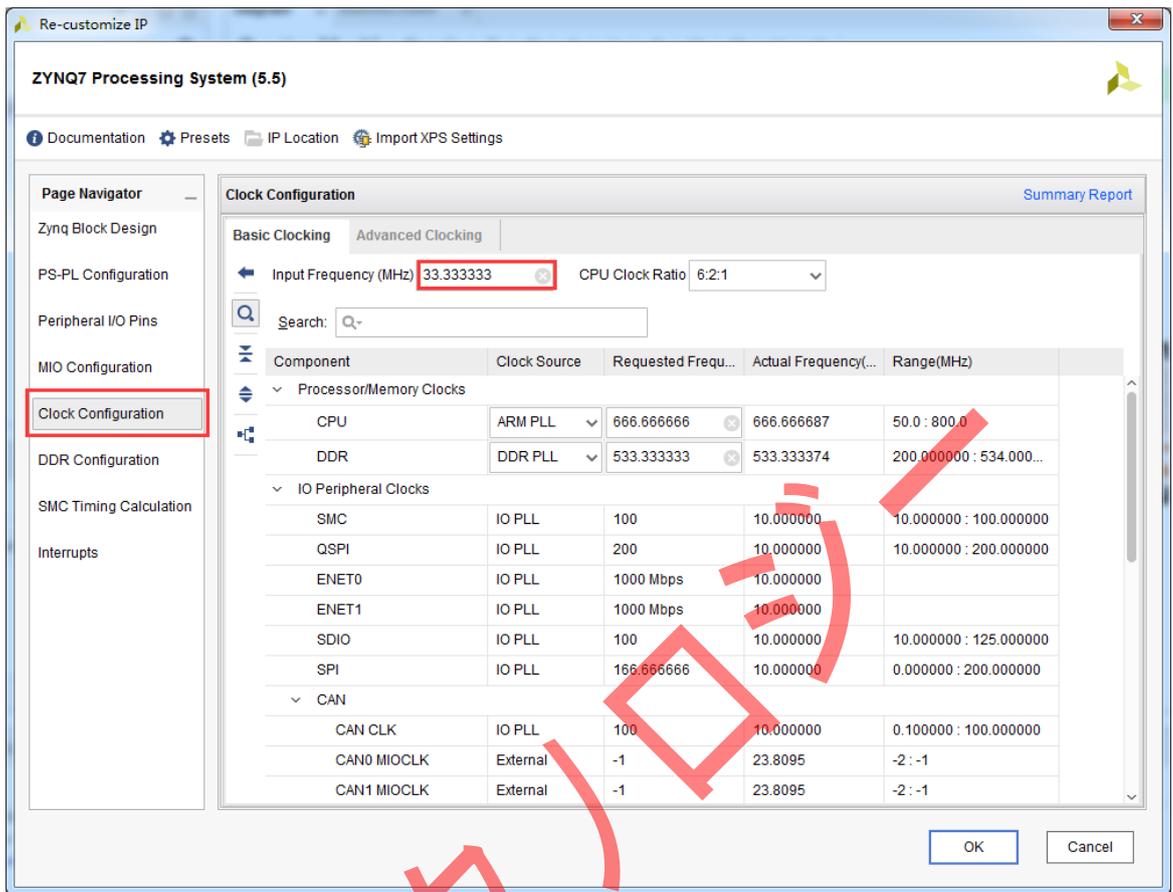
## 6.2.1 UART 配置

9) 原理図から、シリアルは PS の MI048-MI049 上に接続していることが見つかる。  
 “Peripheral I/O Pins” オプションで UART1 (MI048 MI049) を開く。Bank 0 電圧は “LVCMOS 3.3V”、Bank 1 電圧は “LVCOMS 1.8 V” を選択する。本実験はシリアル機能は一つだけ使っている。ここでは他の設備は使用しない。



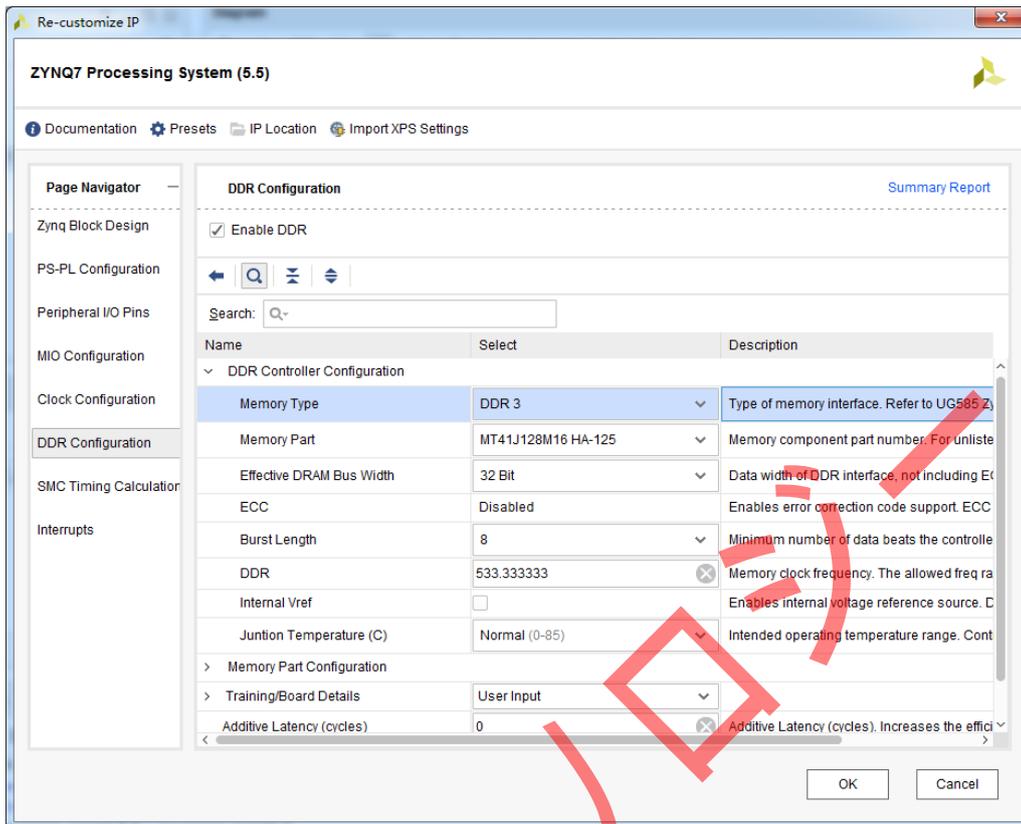
## 6.2.2 クロック配置

10) “Clock Configuration” のオプション欄で、PS クロックのインプット頻度を配置できる。デフォルト値はポートと同じく 33.333333 で、変更する必要ない。CPU 頻度のデフォルト値は 666.666666Mhz、ここも変更しない。同時に PS は PL 側にクロック四重も提供でき、頻度は配置できる。ここは必要ないから、デフォルト値にすればいい。

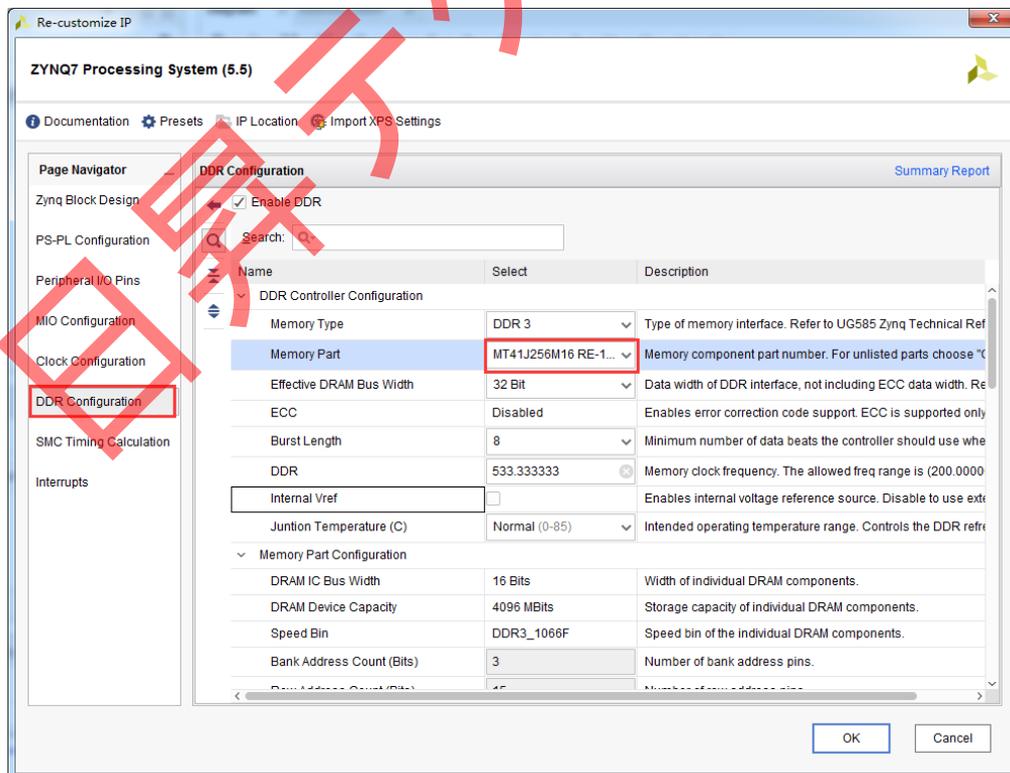


### 6.2.3 DDR3 配置

11) “DDR Configuration”オプションで PS 側の ddr パラメータを配置できる。AX7010 は DDR3 型番を “MT41J128M16 HA-125” に、AX7020 は DDR3 型番を “MT41J256M16 RE-125” にする。この ddr3 型番はポートでの ddr3 型番ではなく、パラメータが一番近いタイプである。“Effective DRAM Bus Width”で “32 Bit” を選択する。

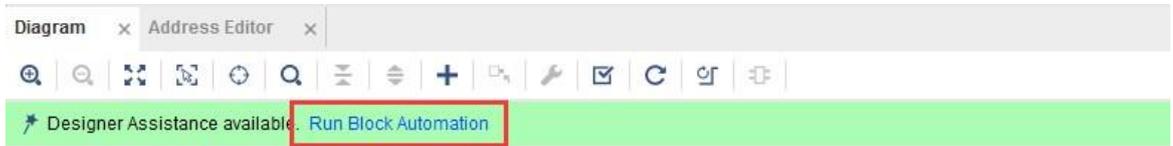


AX7010 DDR3 配置

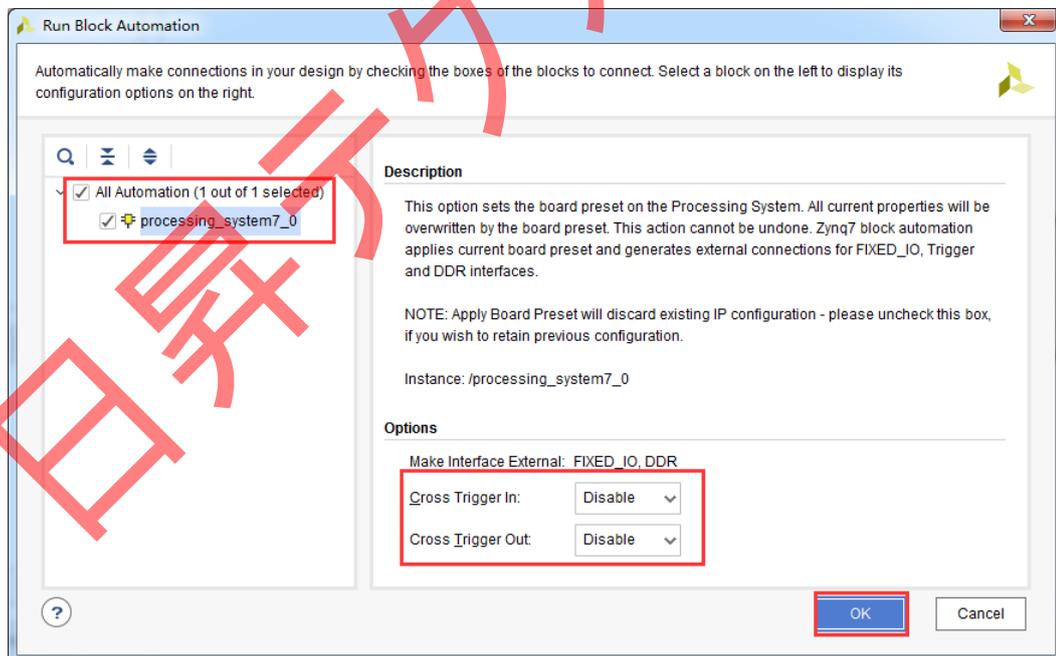


AX7020 ddr3 选择

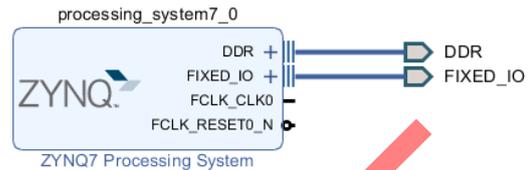
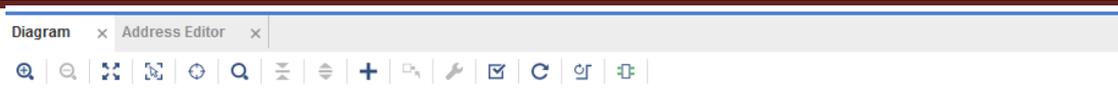
12) “Run Block Automation” をクリックして、vivado ソフトは自動的に一部のポート書き出す仕事を完成する。



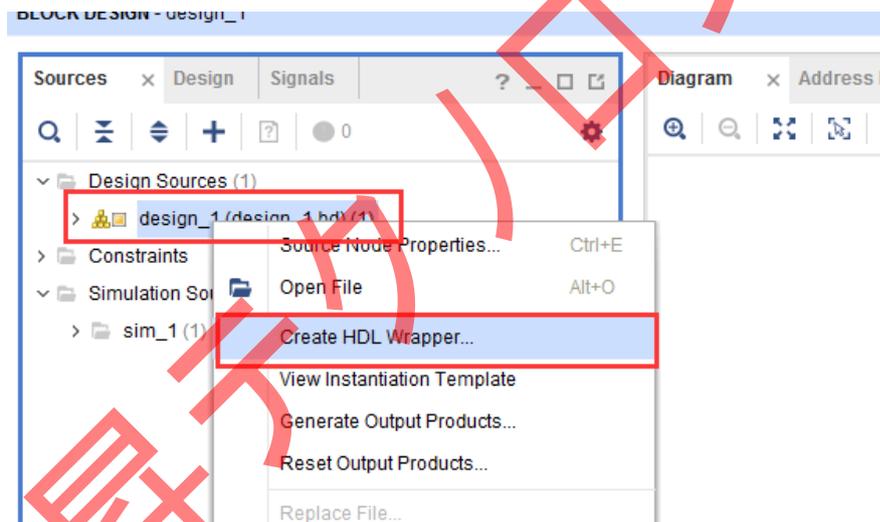
13) “OK” をクリックする。



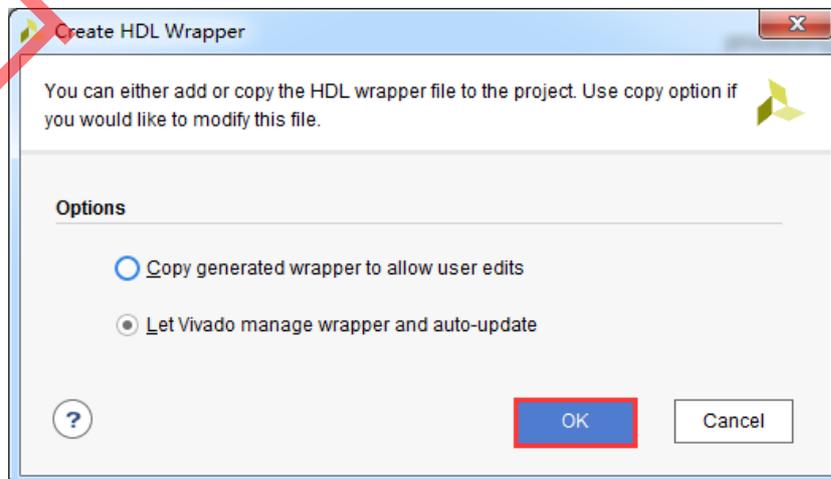
14) “OK” をクリックしたあとは、PS 側からエクスポートしたピンが見える。DDR と FIXED\_IO も含んでいる。“Ctrl + s” を押してデザインを保存する。



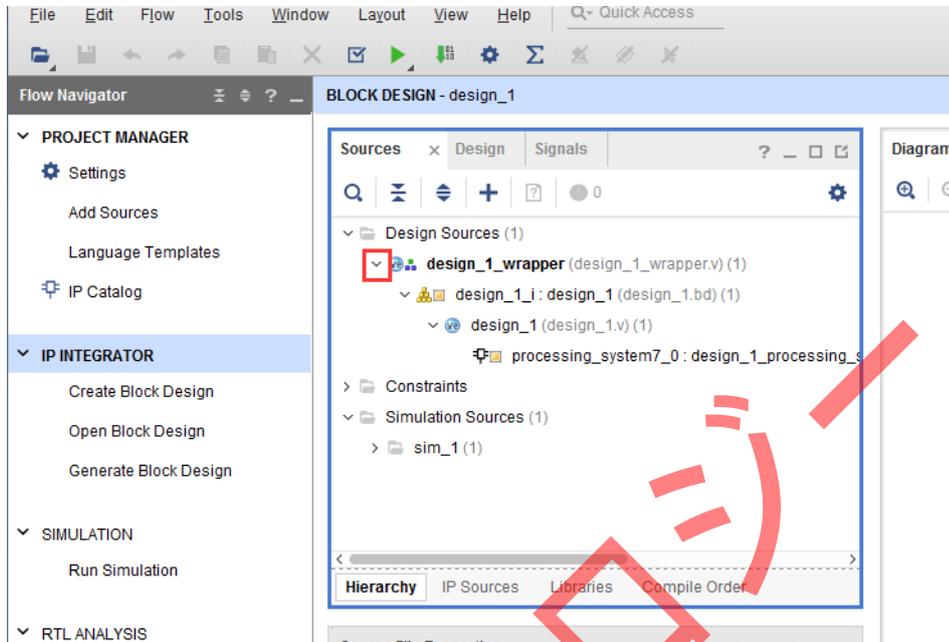
15) Block デザインを選択する。右ボタンを押し、“Create HDL Wrapper...”を選ぶ。Verilog または VHDL ファイルを作成する。



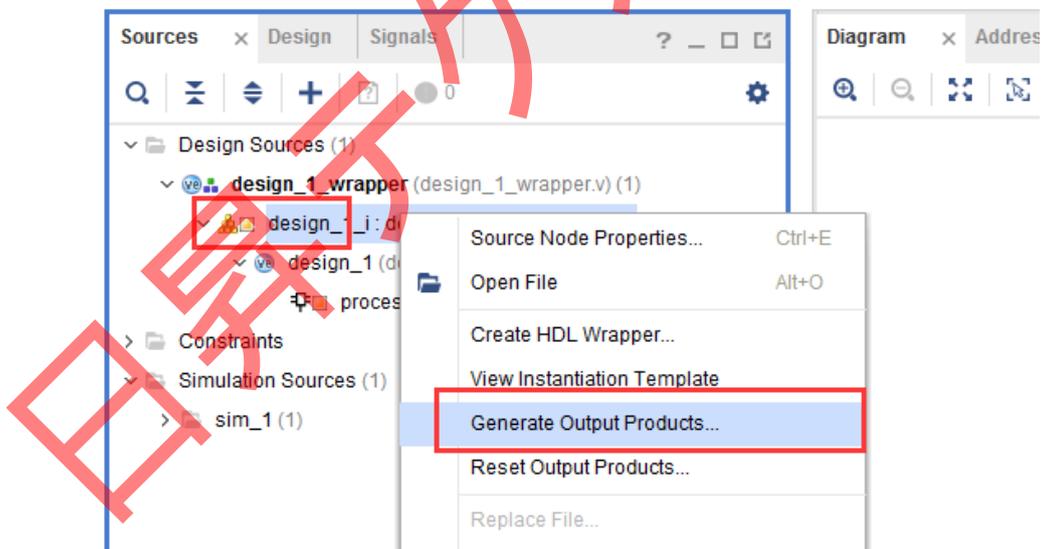
16) デフォルトオプションのままで、“OK”をクリックするため。



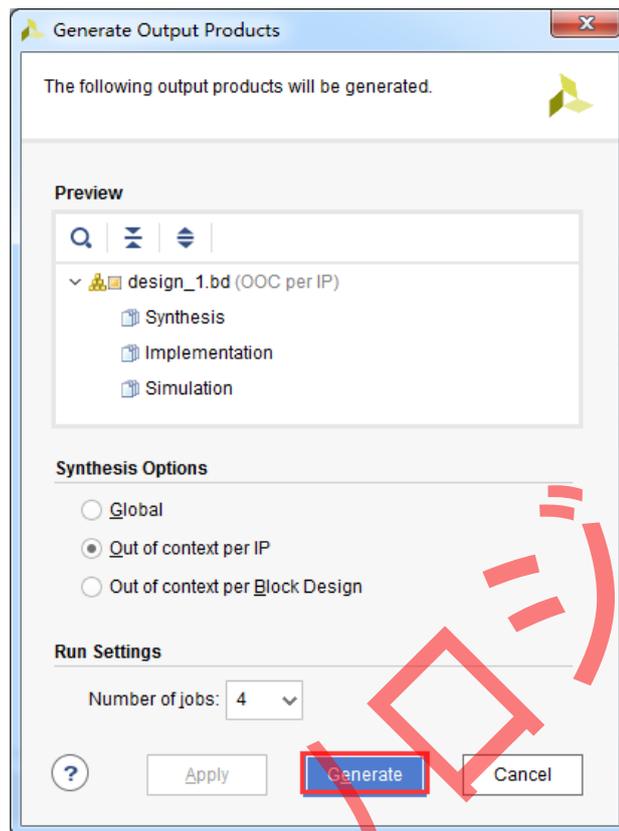
- 17) デザインを開け、PS は普通な IP で使われているの見える。



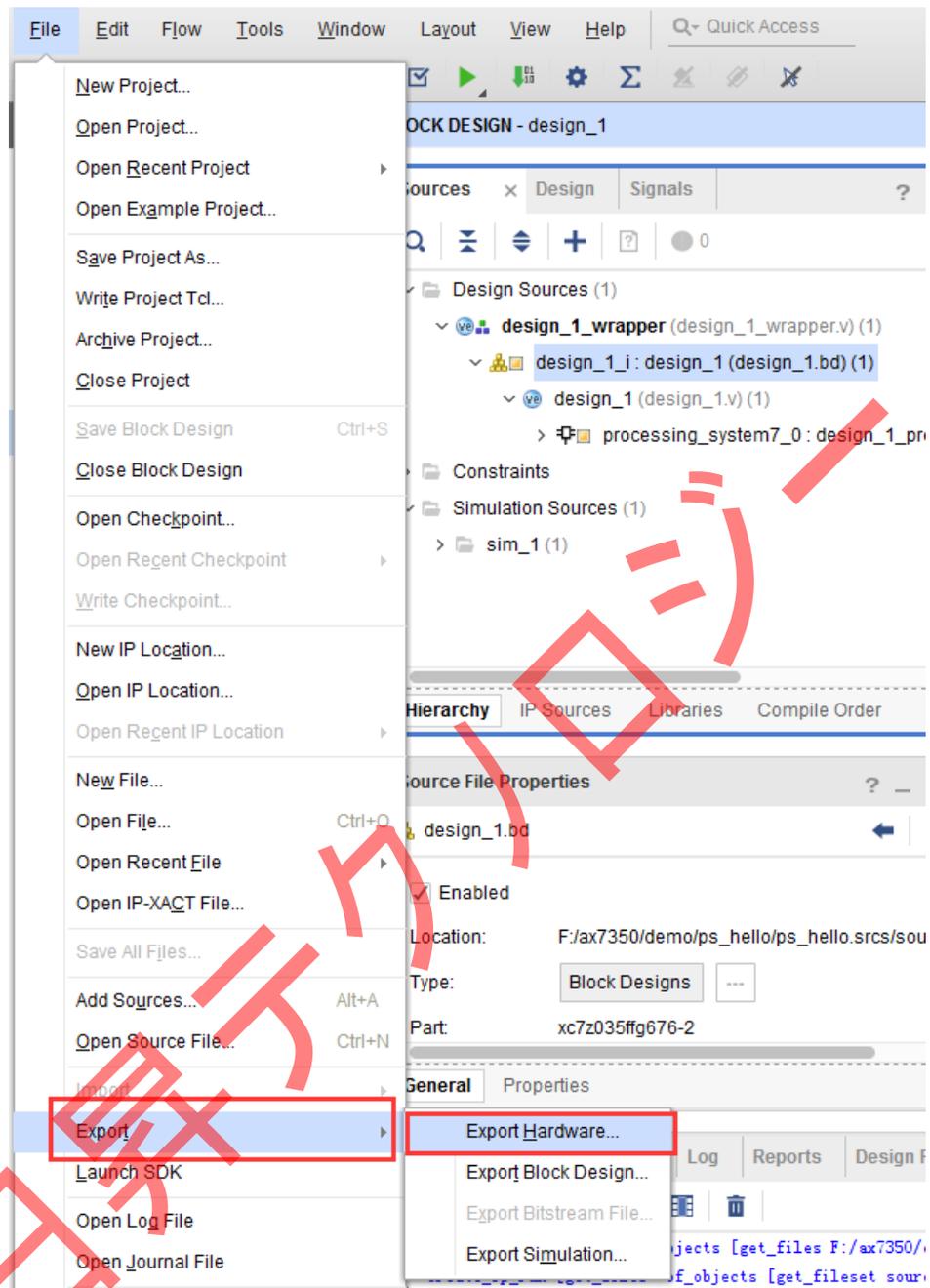
- 18) block デザインを選択する。右ボタンを押して、“Generate Output Products” を選ぶ。



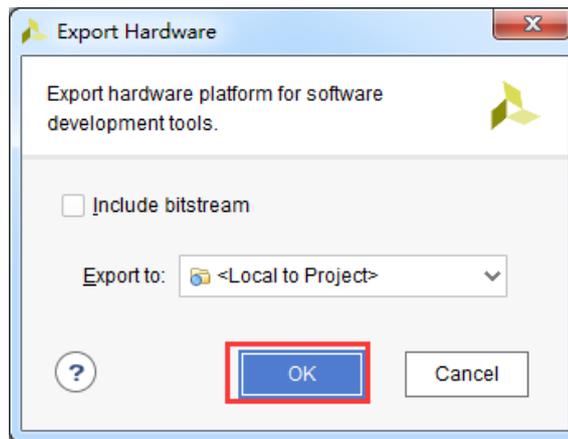
- 19) “Generate” をクリックする。



- 20) メニュー欄で“File -> Export -> Export Hardware...”の順番でハードウェアインフォメーションを書き出す。ここにPSの配置も含んでいる。

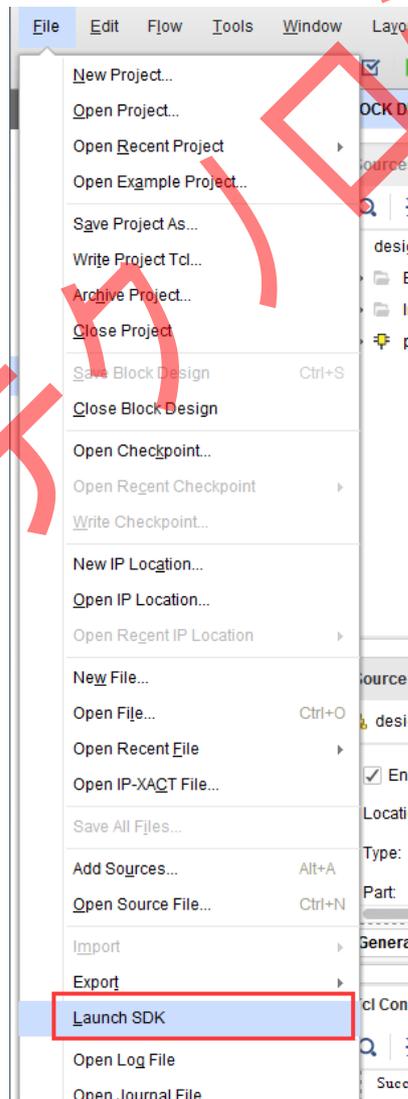


- 21) ポップアップしたダイアログに“OK”をクリックする。実験はただPSのシリアルを使って、PLが参加していないから、ここでは“Include bitstream”を作動させてない。

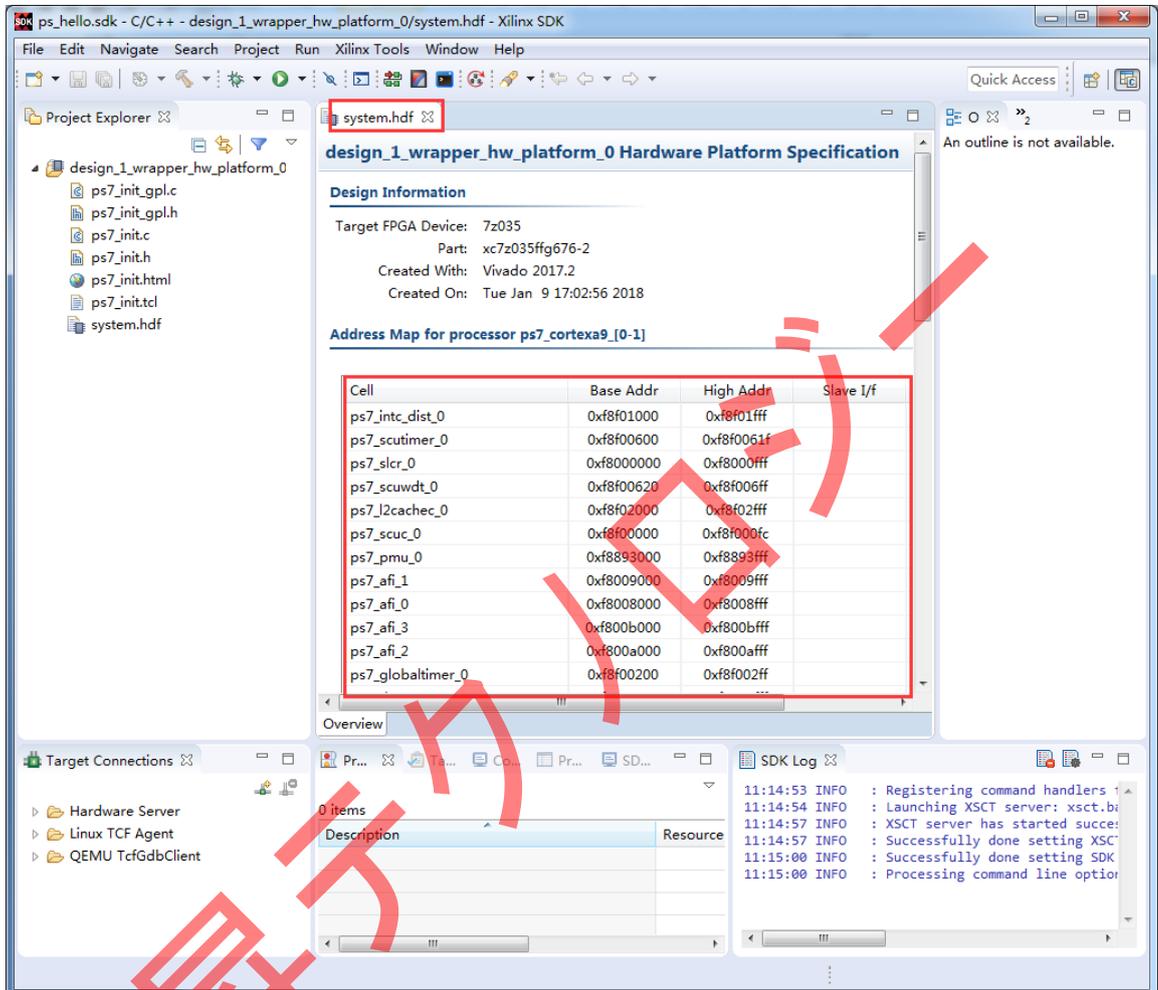


### 6.3 SDK デバッグ

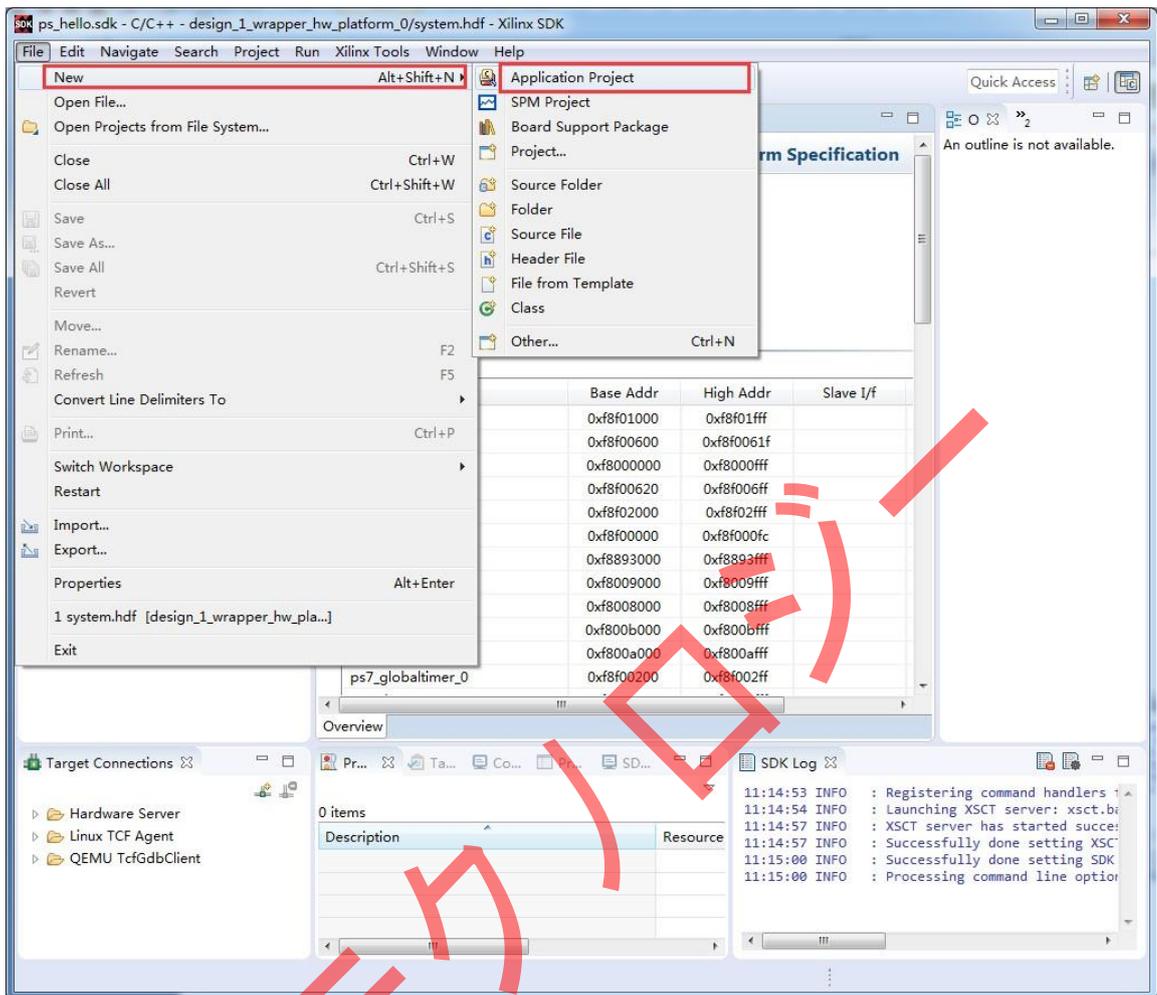
- 1) Vivado メニューで “File -> Launch SDK” をクリックして、SDK を起動する。



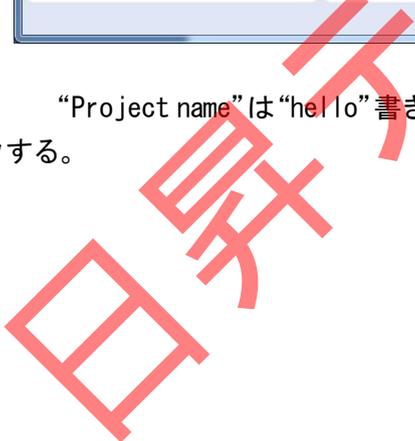
2) SDK 起動して、ファイルがわかる。"system.hdf" というファイルがあって、Vivado ハードウェアデザインのメッセージが含んでいる。ソフト開発にも使用できて、PS 側ペリフェラルのレジスタリストも見える。

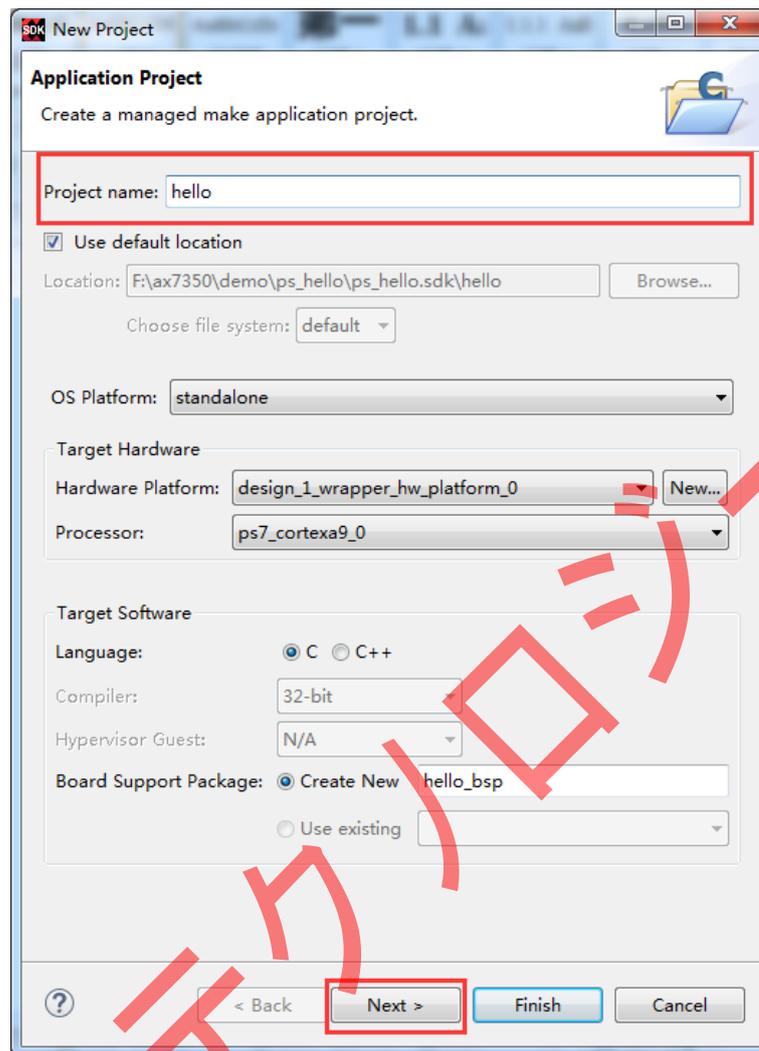


3) メニューで "New -> Application Project" をクリックして、APP プロジェクトを作成する。

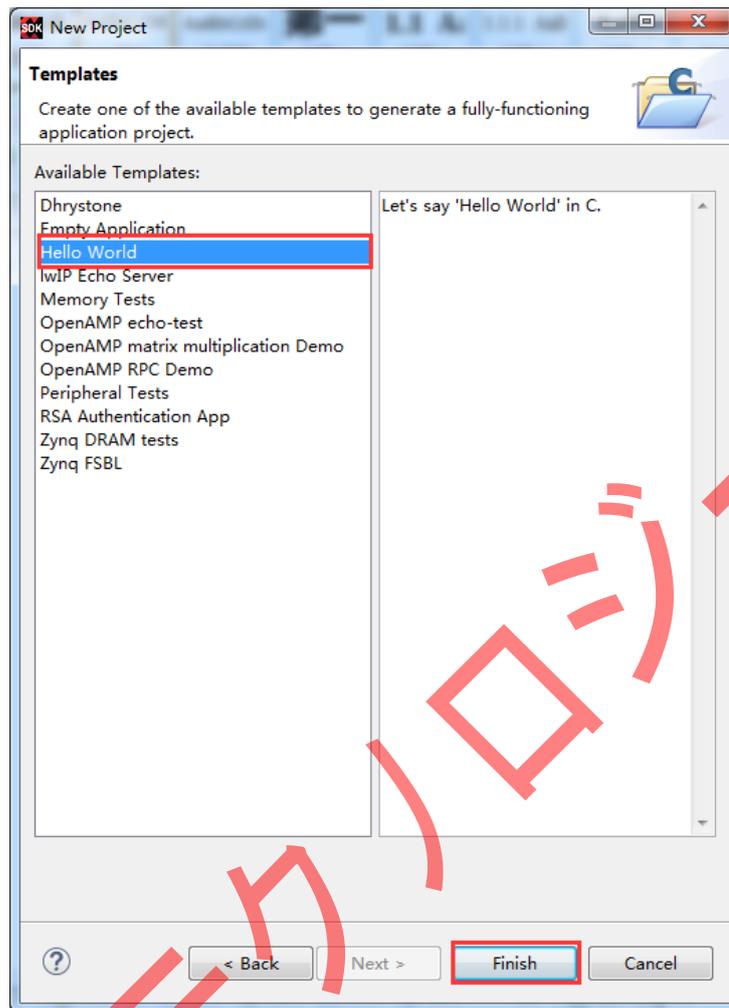


4) “Project name”は“hello”書き込んで、他のはデフォルト値にする。“Next”をクリックする。

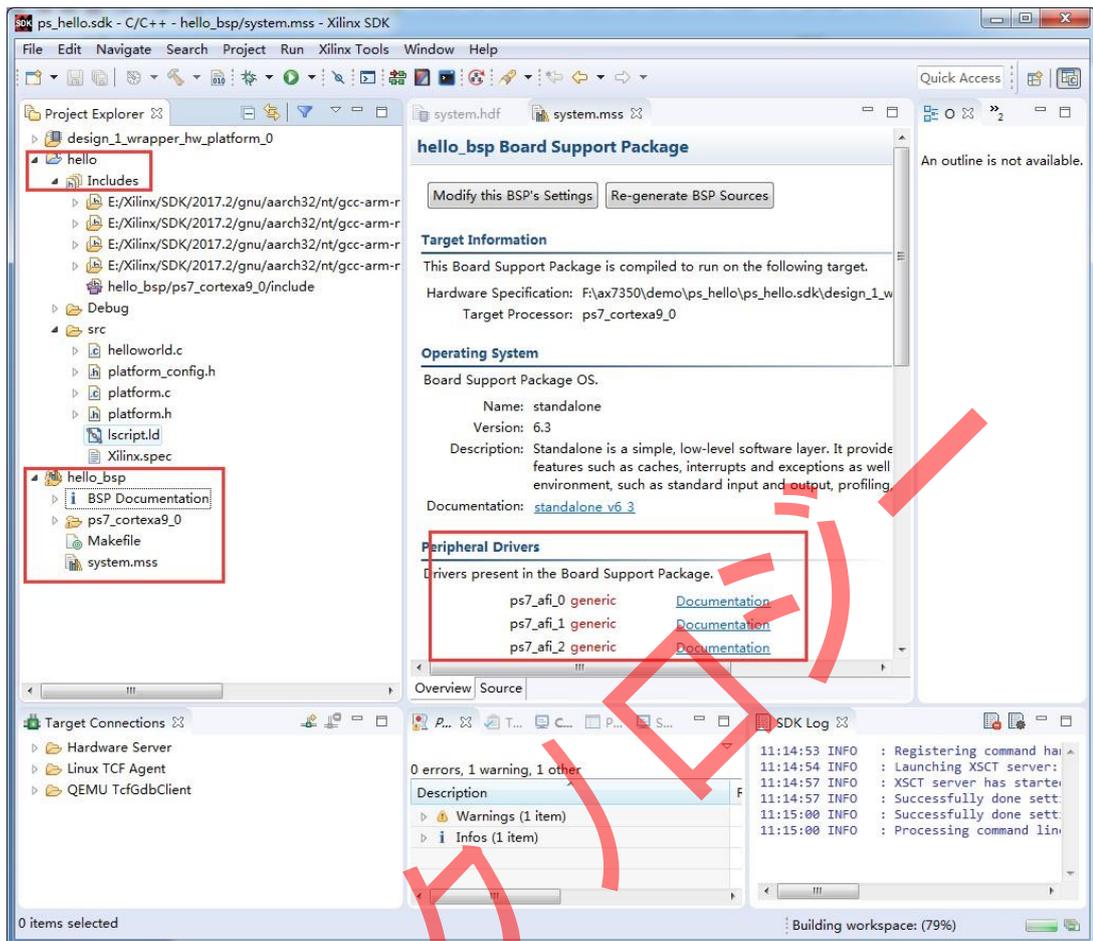




5) モジュールは“Hello World”を選択、“Finish”をクリックする。

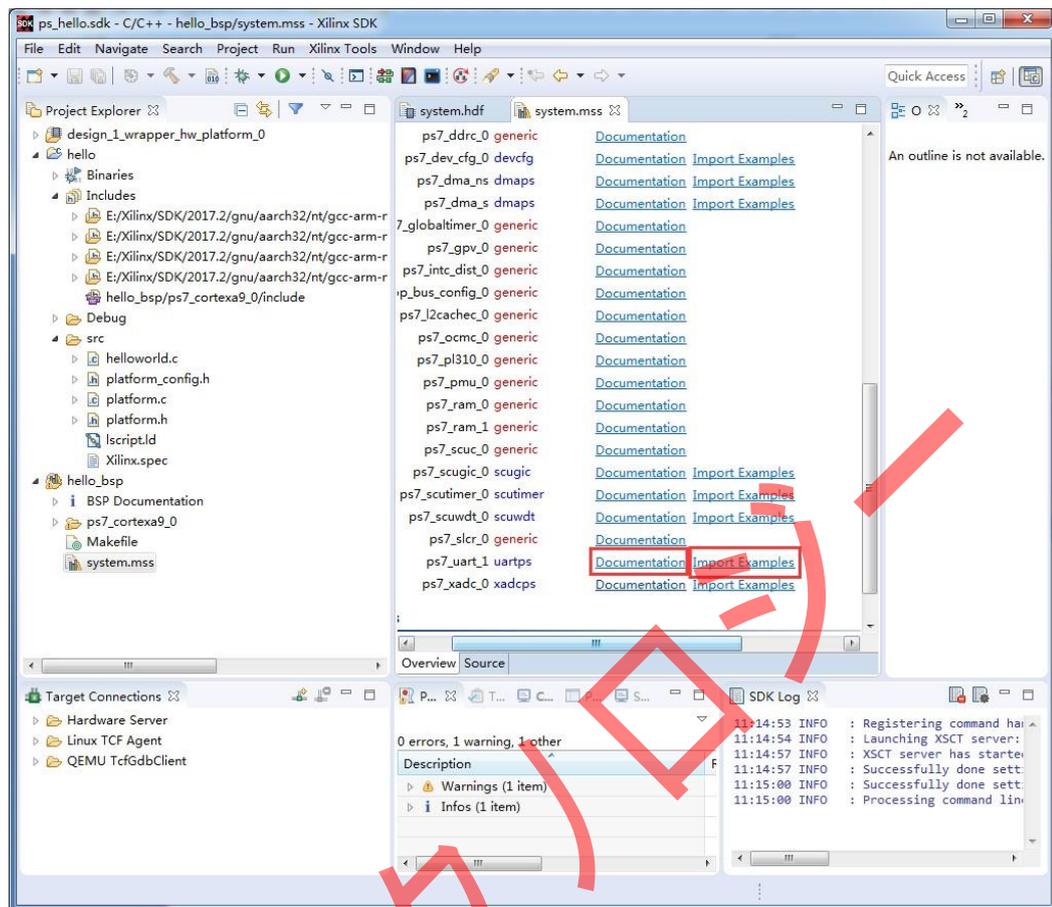


6) SDK は“hello”という目次ぐを作成した。もう一つの目次ぐは“hello\_bsp”である。  
 “hello\_bsp”の目次ぐで使えるもインフォメーション大量である。中の“BSP  
 Documentation”に一部のPS パリフェラルのAPI 説明が入っている。



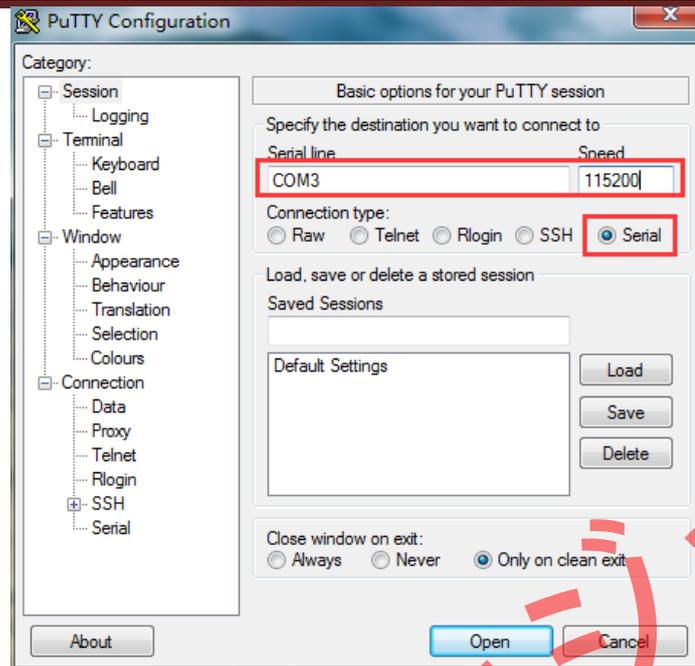
7) “system.mss” をダブルクリックする。一部の PS ペリフェラルはサンプルも提供している。これは対応のペリフェラルを学ぶ大切な資料である。

日昇

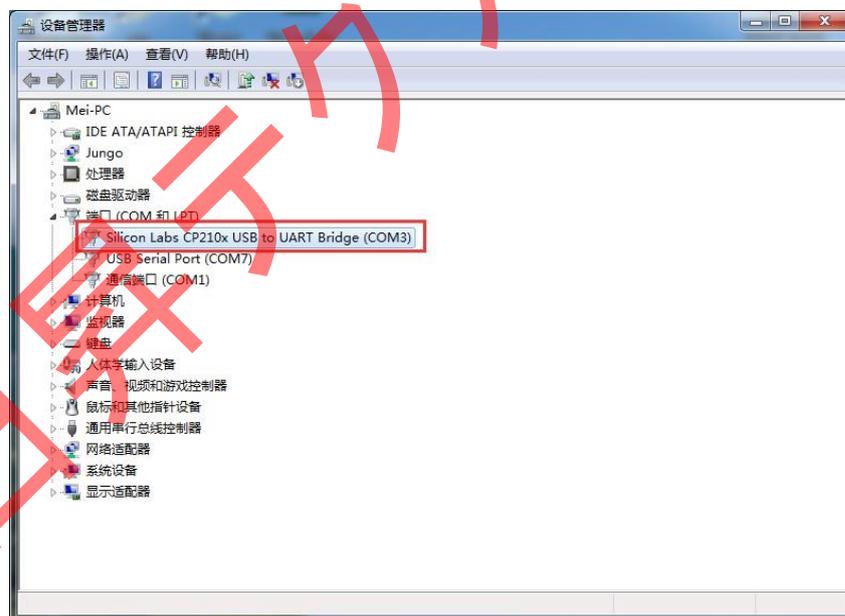


8) JTAG ケーブルを開発ボードに接続して、UART 的 USB ケーブルを PC に接続する。

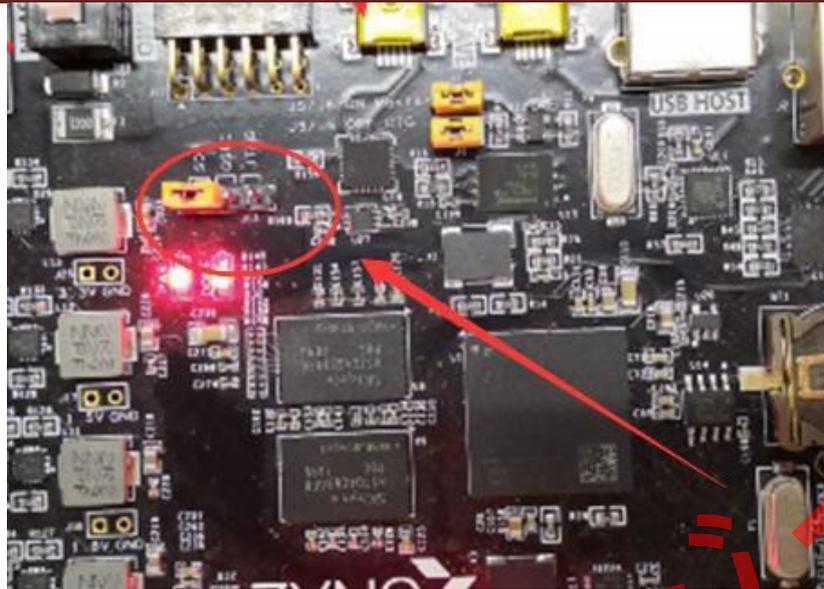
9) PuTTY ソフトをシリアルターミナルデバッグツールにする。PuTTY は無料ダウンロードのソフトである。



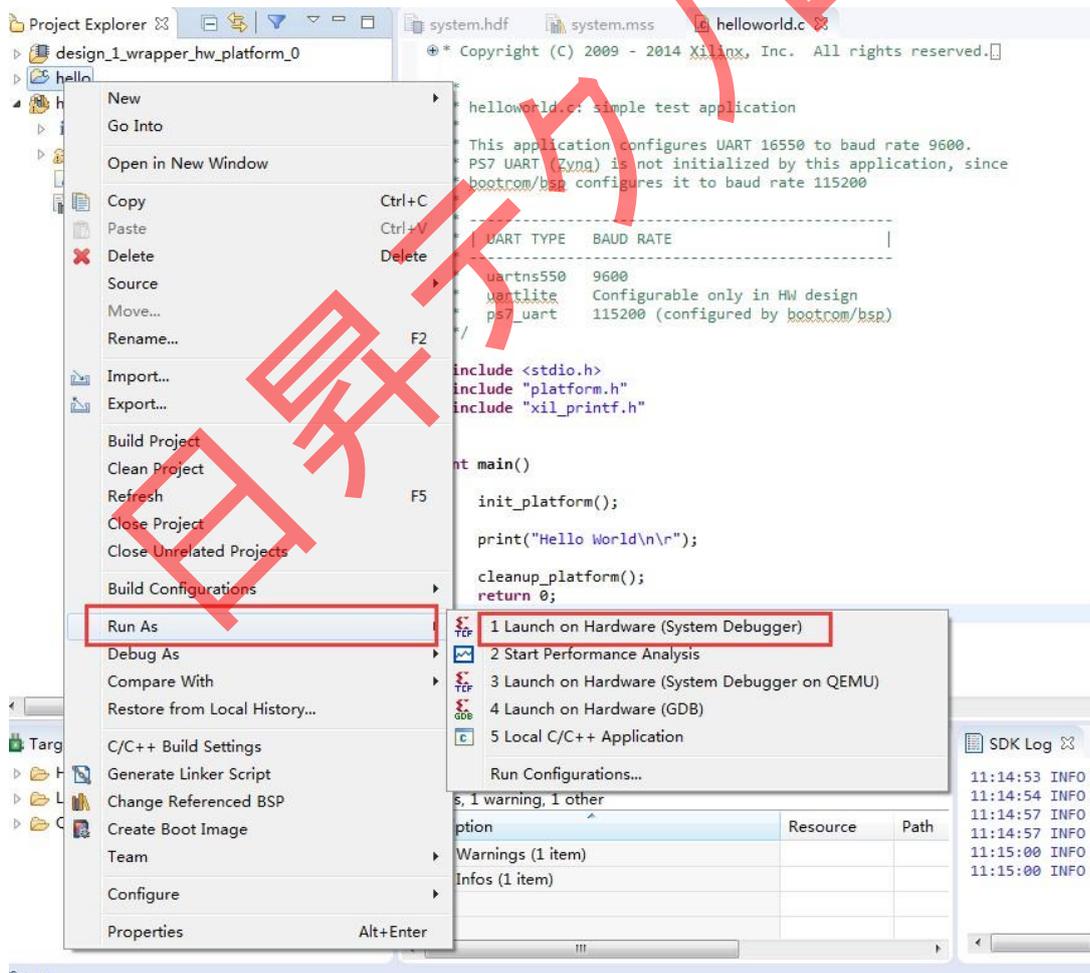
10) Serial, Serial line を COM3 欄にデータを入れる。Speed 欄に 115200 を書き込む。COM3 シリアル番号はデバイスマネージャに表示したように写す。“Open” をクリックする。



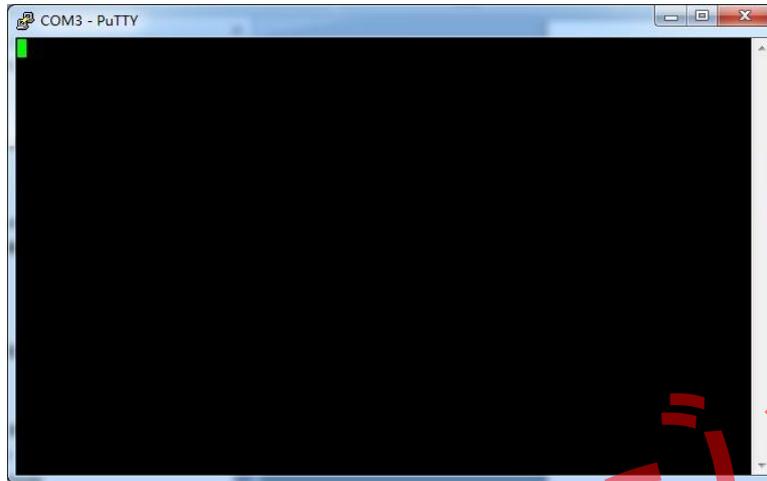
11) 電気を入れる前に、開発ボードの起動モードを JTAG モードに設置する。



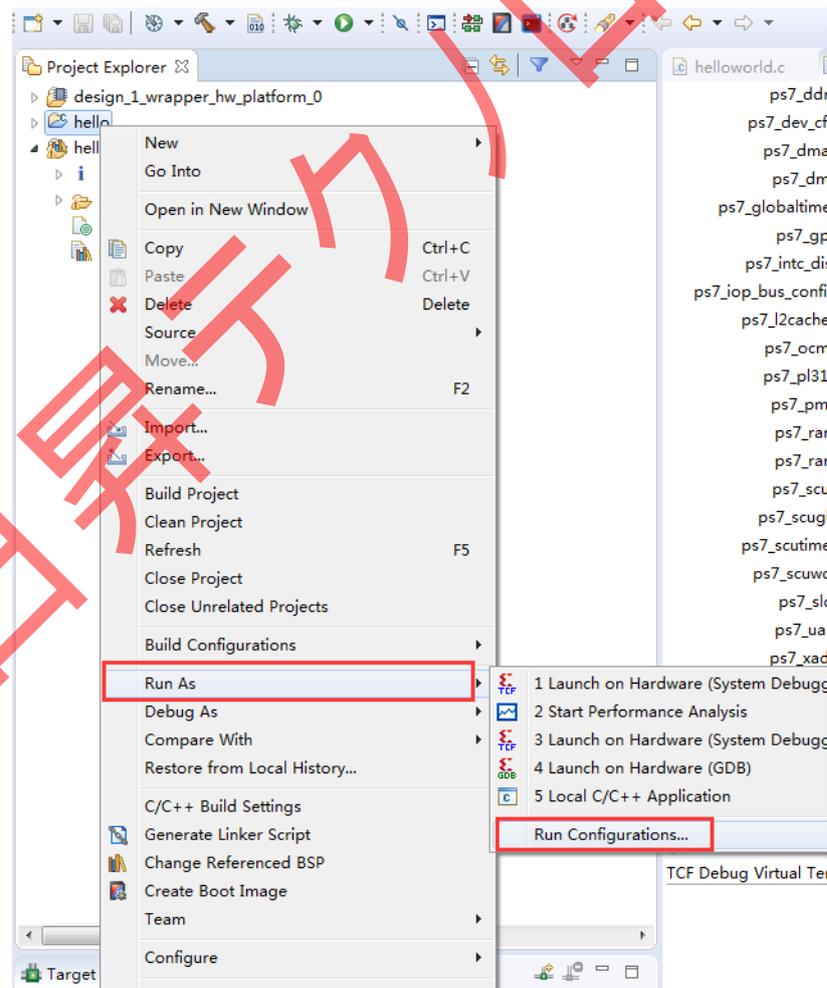
12) 開発ボードに電気をいれて、プログラムの作動に準備をする。出荷の時、開発ボードは既にプログラムがつけられている。ここで、作動モードを JTGA モードに選び、改めて電気いれをする。“hello” を選択し、右ボタンを押せば、オプションがたくさん見える。本実験に使うのは“Run as”で、プログラムを起動すること。“Run as”にもオプションたくさんあって、一番目“Launch on Hardware (System Debugger)”を選択する。システムでデバッグをして、直接プログラムを作動する。



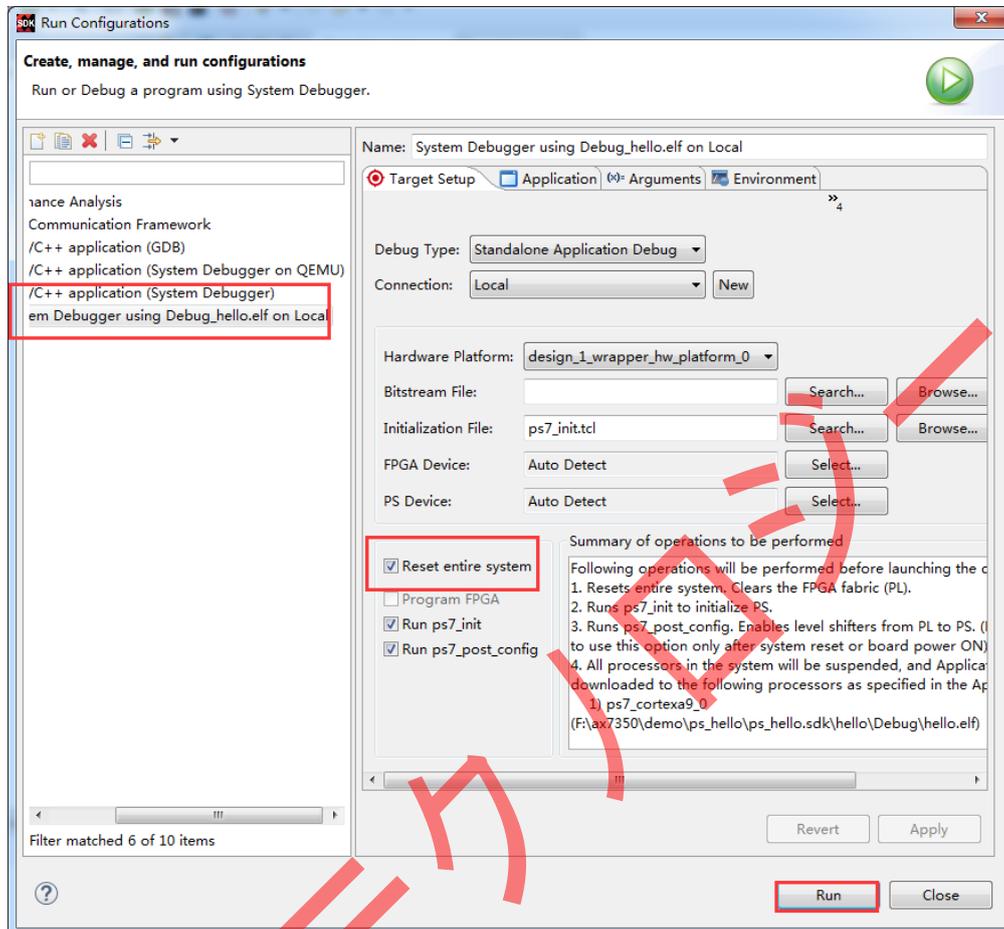
- 13) この時は PuTTY ソフトを観察する。アウトプットが表示されるかも、アウトプットがないかも知らない。



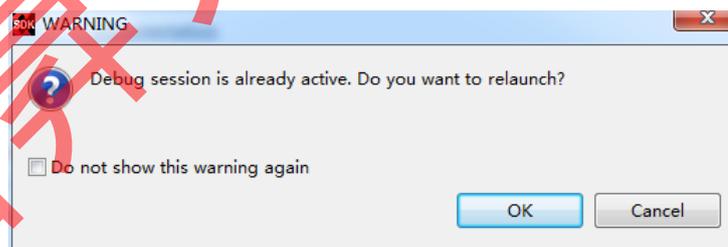
- 14) システムのデバッグを保証するため、配置一つ追加必要がある。右ボタンを押し、“Run As → Run Configuration...” の順番で操作する。



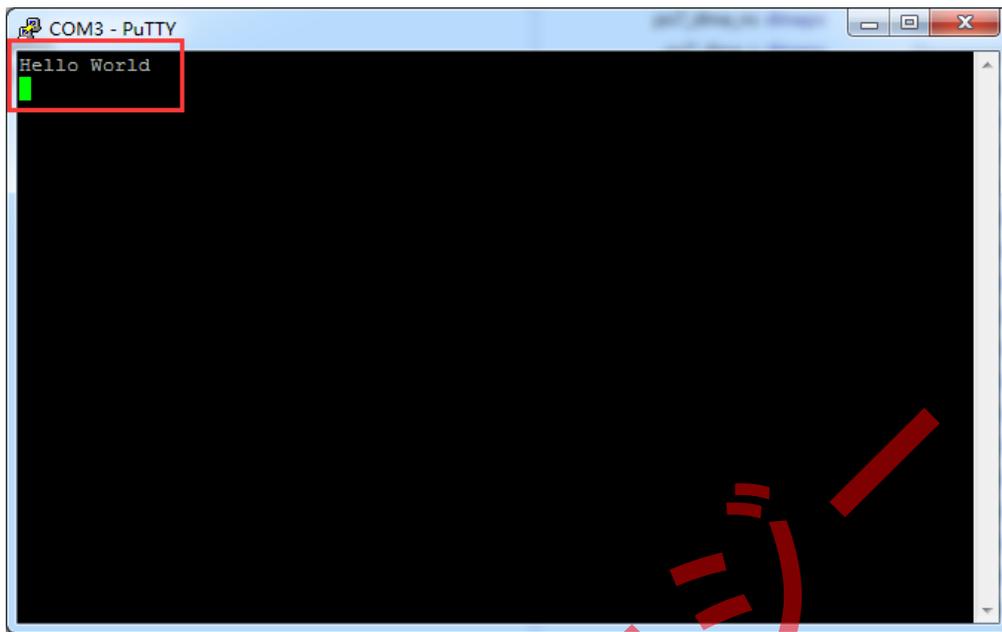
- 15) “Reset entire system” を選択して、システムをリセットする。システムにまだ PL デザインがあったら、“Program FPGA” を選択してから、再び “Run” をクリックする。



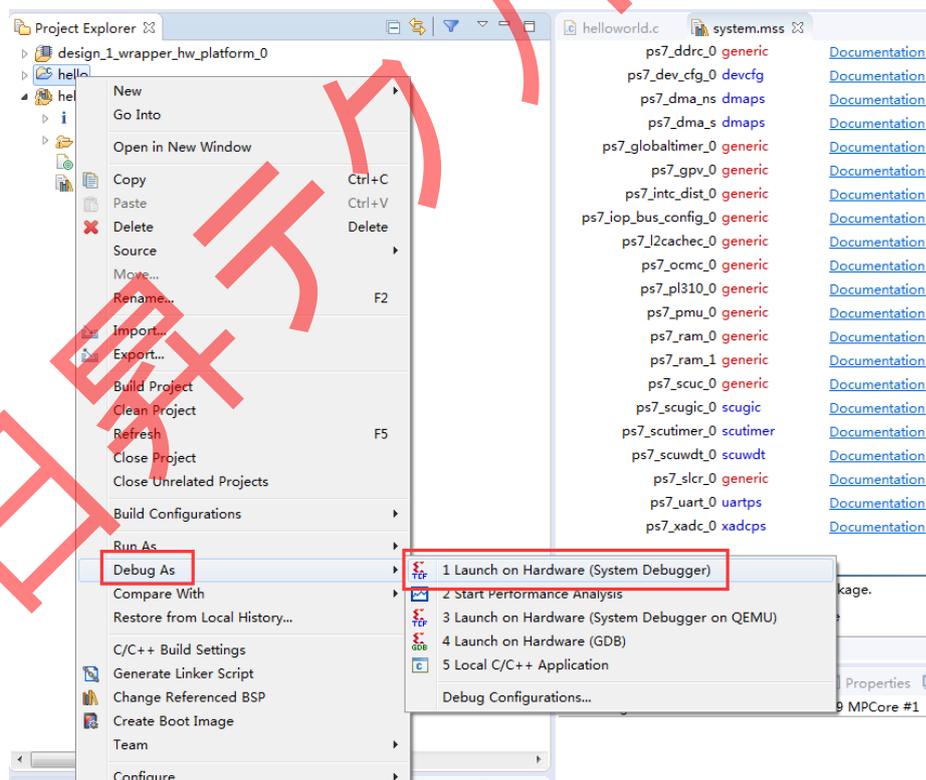
- 16) “OK” をクリックし、再起動を確定する。



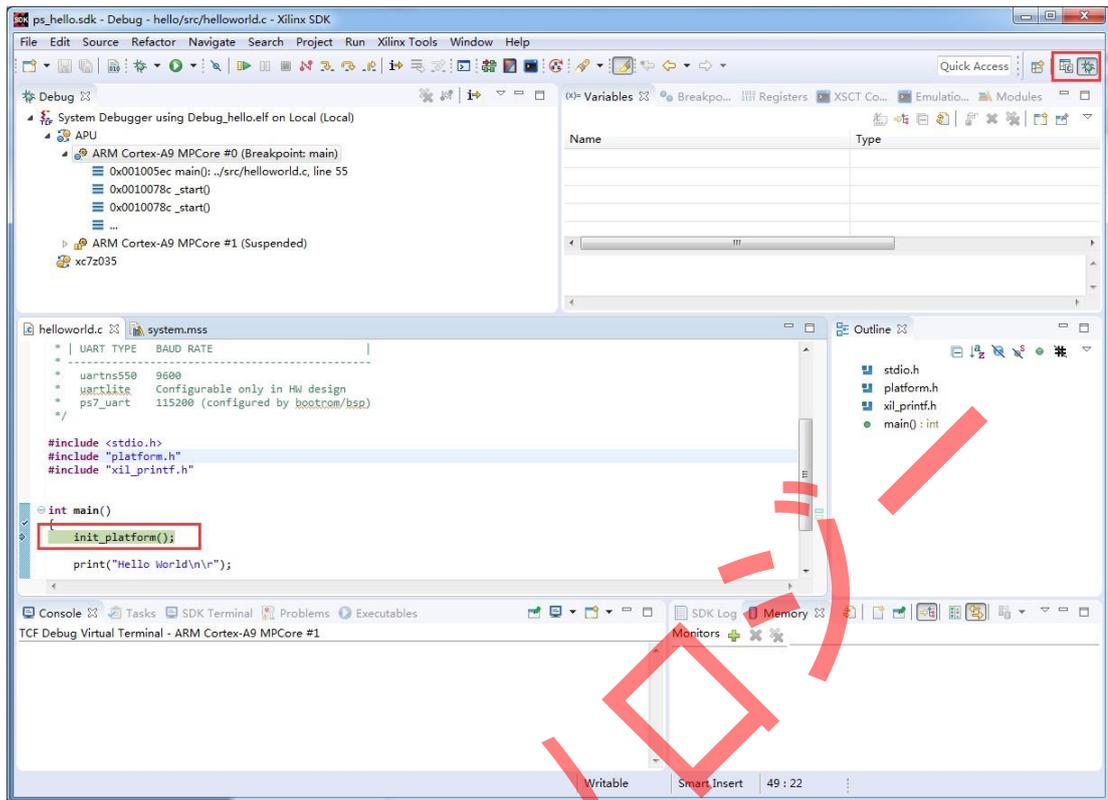
- 17) 今回見知りの “Hello World” が画面に表示された。



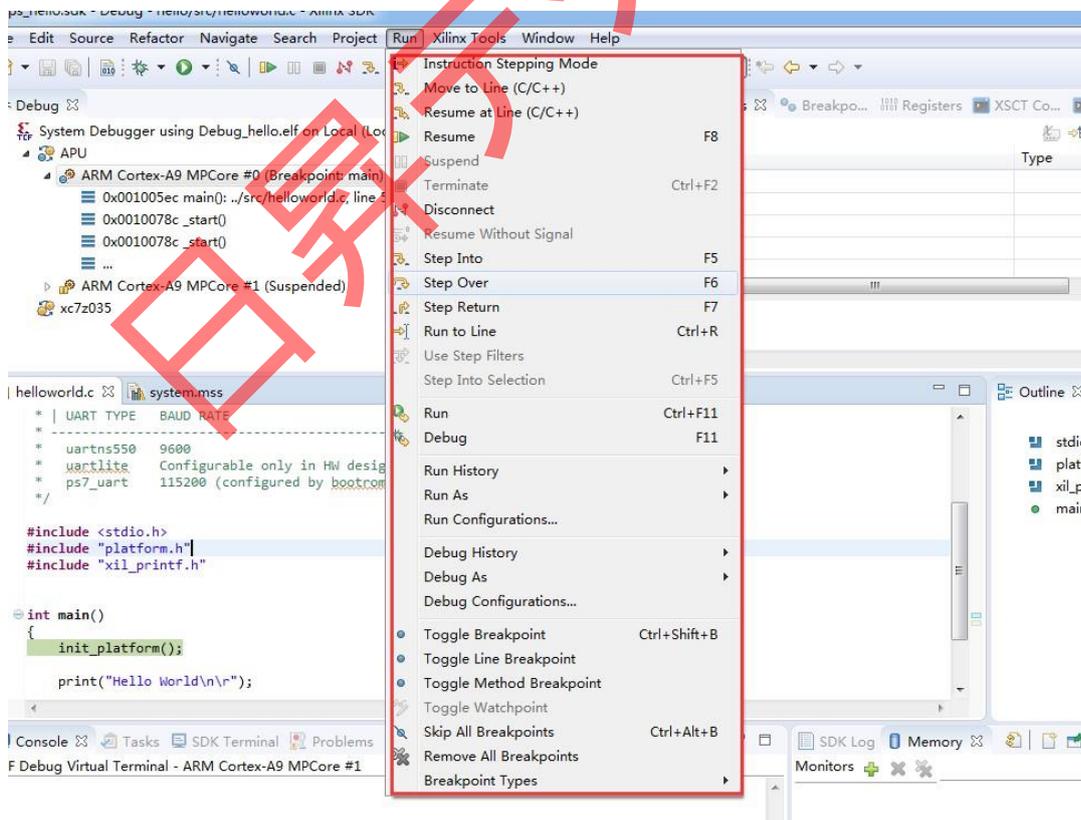
18) “Run As” 以外、“Debug As” もう使用できる。こうして、ブレイクポイントを設置でき、シングルステップで作動できる。



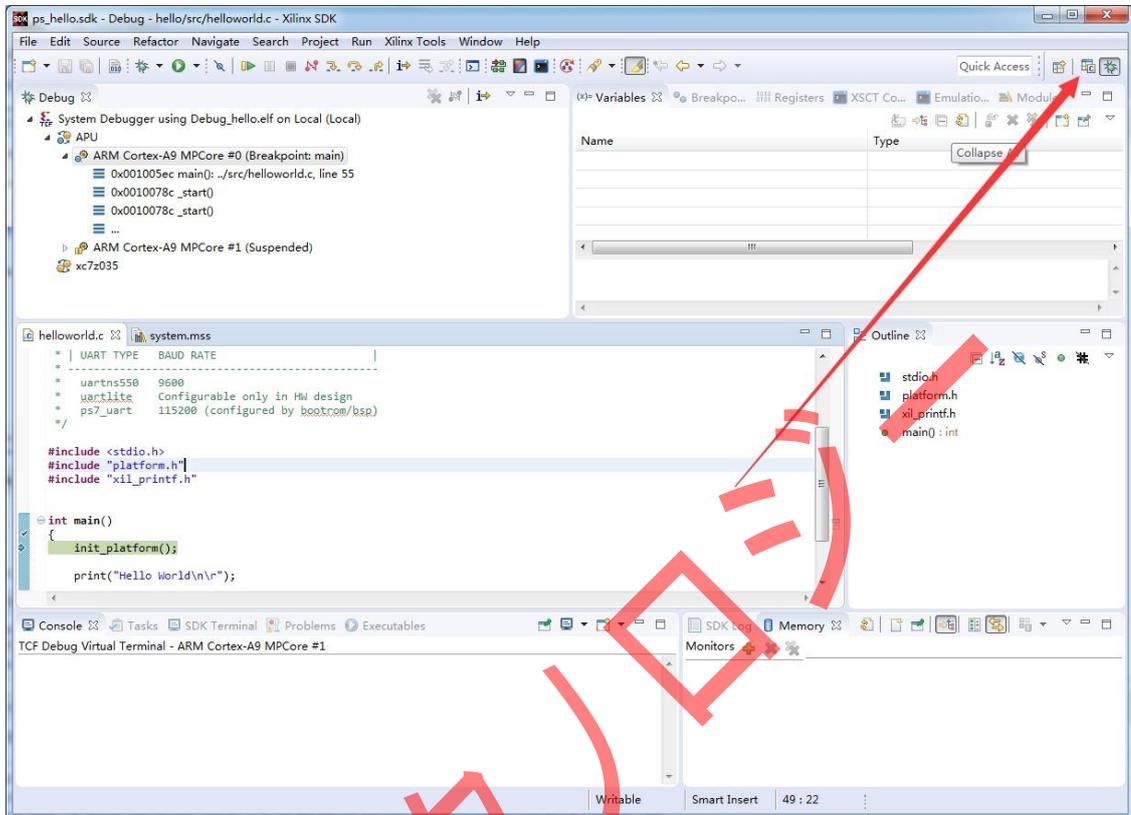
19) Debug モードに入る。



20) IDEを開発は他のC言語と同じ、ステップバイステップで作動するやブレイクポイントを設置できる。



- 21) 右上で IDE モードが変更できる。



## 6.4 実験のまとめ

本実験は簡単な Hello World で SDK の使い方を説明した。SDK は強い機能があって、一つ一つで紹介できないが、使用の繰り返しでだんだん把握できると思う。

## 6.5 よくある問題

### 6.5.1 vivadoに通じて SDK を起動後、ウィンドウがポップアップされていない

- 1) Vivado ソフトをインストールする時はかならず SDK もインストールする。
- 2) SDK を起動する前に sdk 目次ぐがあったら、SDK が起動できないことになる。この目次ぐを削除してから試す。



名前	修改日時	タイプ	サイズ
.metadata	2018/3/5 15:03	文件夹	
.Xil	2018/3/5 19:32	文件夹	
ps_hello.cache	2018/3/5 19:29	文件夹	
ps_hello.hw	2018/3/5 19:29	文件夹	
ps_hello.ip_user_files	2018/3/5 19:29	文件夹	
ps_hello.runs	2018/3/5 19:29	文件夹	
ps_hello.sdk	2018/3/5 19:32	文件夹	
ps_hello.sim	2018/3/5 19:29	文件夹	
ps_hello.srcs	2018/3/5 19:29	文件夹	
RemoteSystemsTempFiles	2018/3/5 15:03	文件夹	
ip_upgrade.log	2018/3/5 14:27	wrfile	5 KB
ps_hello.xpr	2018/3/5 16:20	Vivado Project Fi...	7 KB
SDK.log	2018/3/5 15:03	wrfile	1 KB
vivado.jou	2018/3/5 19:32	JOU 文件	1 KB
vivado.log	2018/3/5 19:44	wrfile	2 KB
vivado_8944.backup.jou	2018/3/5 16:22	JOU 文件	7 KB
vivado_8944.backup.log	2018/3/5 16:52	wrfile	15 KB

株式会社日昇テクノロジー

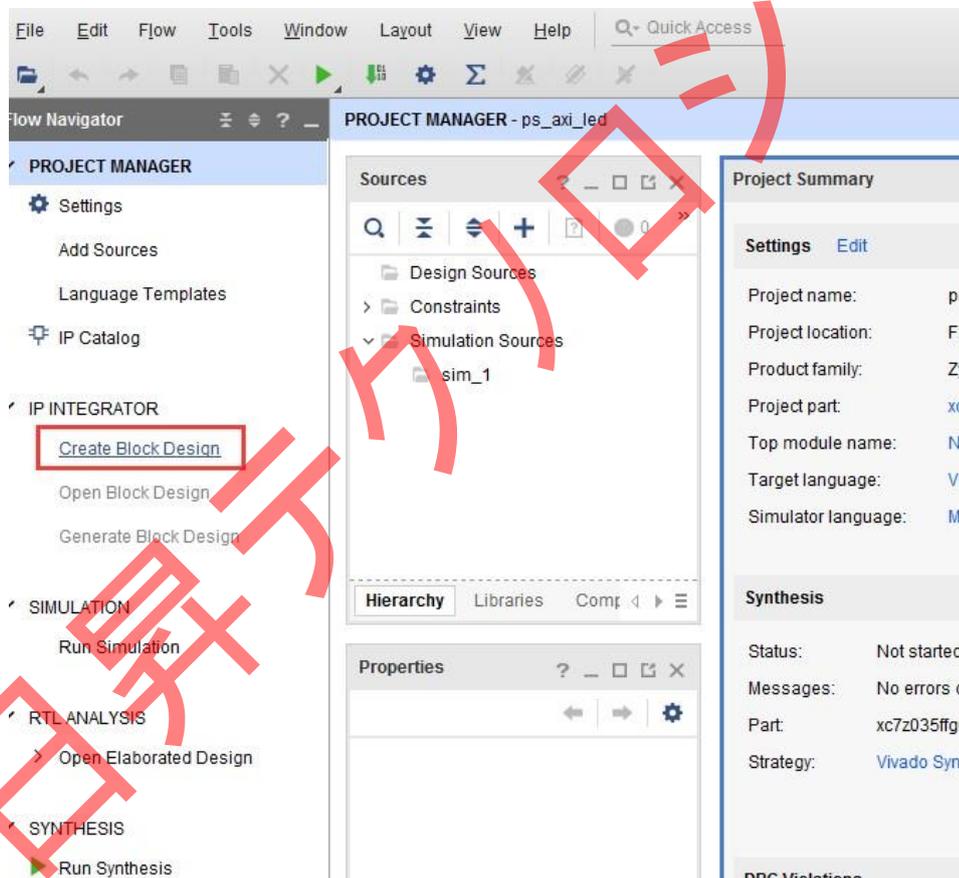
## 第七章 PS で PL の LED を点灯する

実験用 Vivado プロジェクトは “ps\_axi\_led”。

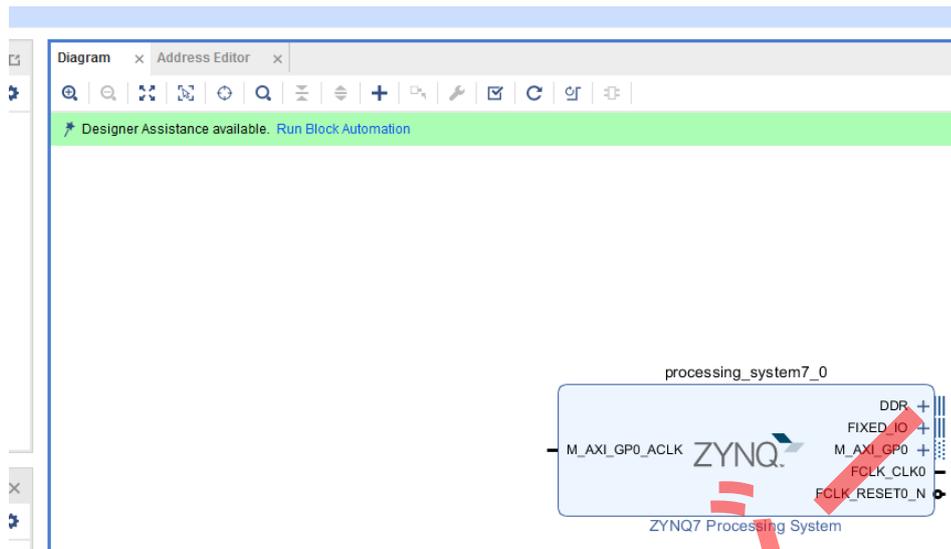
zynq の使用に一番大きい疑問はどうすれば PS と PL を結んで使用できることである。一般的に、ほかの SOC チップに GPIO がある。本実験は AXI GPIO の IP コアを一つ使って、PS 側を AXI バスで PL 側の LED ライトをコントロールする時は。実験は簡単が、PL と PS と結びつく方法を示す。

### 7.1 Vivado プロジェクトを作成する

- 1) “ps\_axi\_led” Vivado という名のプロジェクトを作成して、PS が AXI バスで LED ライトをコントロールすることを表示する。
- 2) Block デザインを作成する。

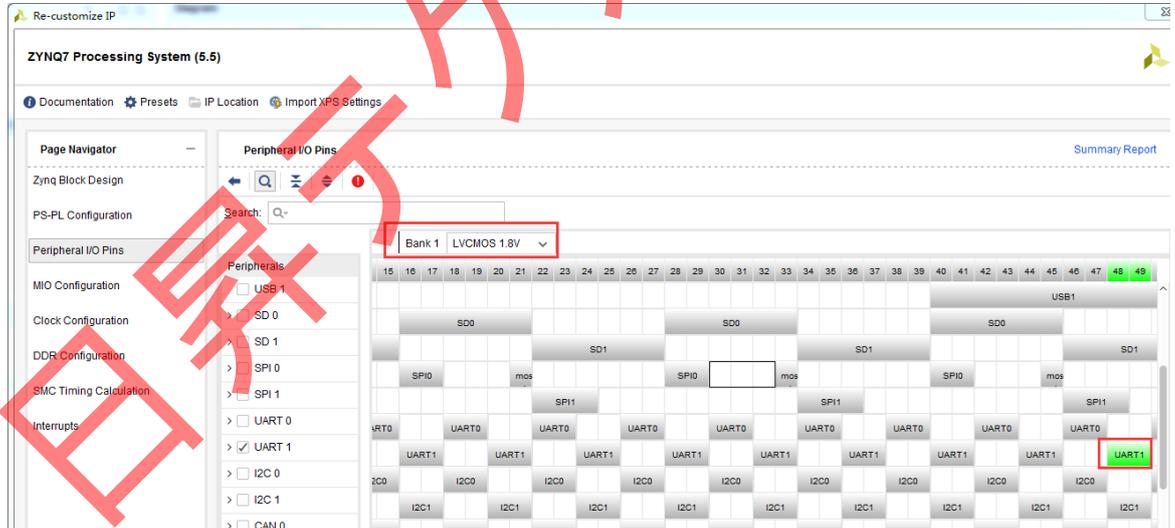


- 3) ZYNQ プロセッサを追加する。



### 7.1.1 UART 配置

1) Bank1 レベル標準をLVCMOS 1.8V に配置する。**Bank1 のレベル標準を配置しないと、シリアルポートが受け取れないかもしれない。**シリアルポートを作動させ、MI048 MI049 を使う。



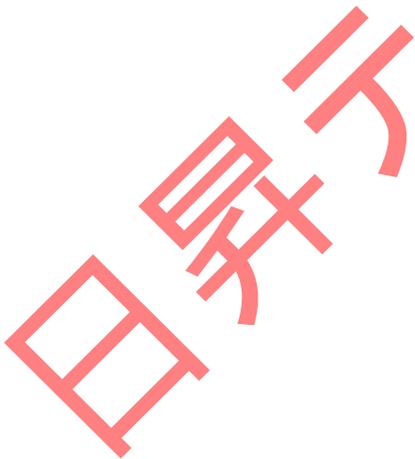
### 7.1.2 DDR3 配置

1) AX7010 は DDR3 の型番を“MT41J128M16 HA-125”に、AX7020 は DDR3 の型番を“MT41J256M16 RE-125”に配置する。**ここの ddr3 タイプはボードでの ddr3 タイプではなく、パラメータが一番近いタイプである。**



Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Type of memory interface. Refer to UG585 Z
Memory Part	MT41J128M16 HA-125	Memory component part number. For unliste
Effective DRAM Bus Width	32 Bit	Data width of DDR interface, not including E
ECC	Disabled	Enables error correction code support. ECC
Burst Length	8	Minimum number of data beats the controlle
DDR	533.333333	Memory clock frequency. The allowed freq ra
Internal Vref	<input type="checkbox"/>	Enables internal voltage reference source. D
Junction Temperature (C)	Normal (0-85)	Intended operating temperature range. Cont
Memory Part Configuration		
Training/Board Details	User Input	
Additive Latency (cycles)	0	Additive Latency (cycles). Increases the effici

AX7010 DDR3 配置



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration**
- SMC Timing Calculation
- Interrupts

DDR Configuration Summary Report

Enable DDR

Search:

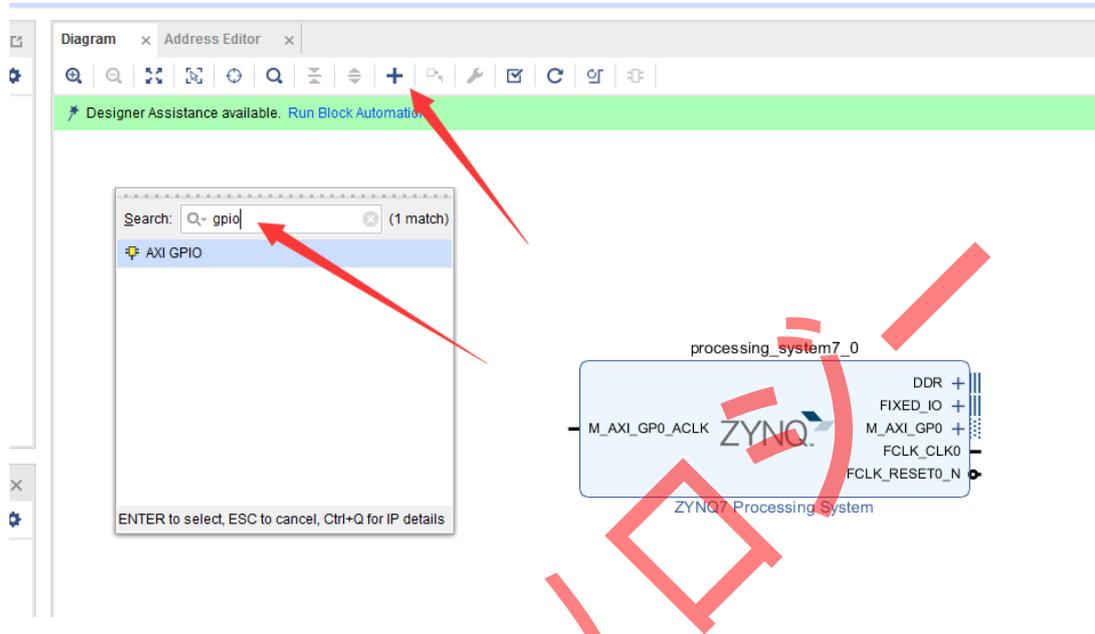
Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Type of memory interface. Refer to UG585 Zynq Technical Ref
Memory Part	MT41J256M16 RE-1...	Memory component part number. For unlisted parts choose "C
Effective DRAM Bus Width	32 Bit	Data width of DDR interface, not including ECC data width. Re
ECC	Disabled	Enables error correction code support. ECC is supported only
Burst Length	8	Minimum number of data beats the controller should use whe
DDR	533.333333	Memory clock frequency. The allowed freq range is (200.0000
Internal Vref	<input type="checkbox"/>	Enables internal voltage reference source. Disable to use ext
Junction Temperature (C)	Normal (0-85)	Intended operating temperature range. Controls the DDR refr
Memory Part Configuration		
DRAM IC Bus Width	16 Bits	Width of individual DRAM components.
DRAM Device Capacity	4096 MBits	Storage capacity of individual DRAM components.
Speed Bin	DDR3_1066F	Speed bin of the individual DRAM components.
Bank Address Count (Bits)	3	Number of bank address pins.
Row Address Count (Bits)	15	Number of row address pins.

OK Cancel

日昇テクノロジー

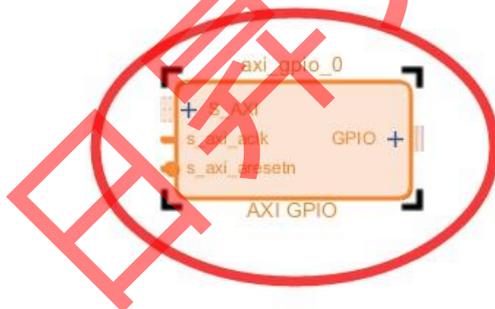
## AX7020 ddr3 选择

- AXI GPIO の IP コアを一つ追加する。

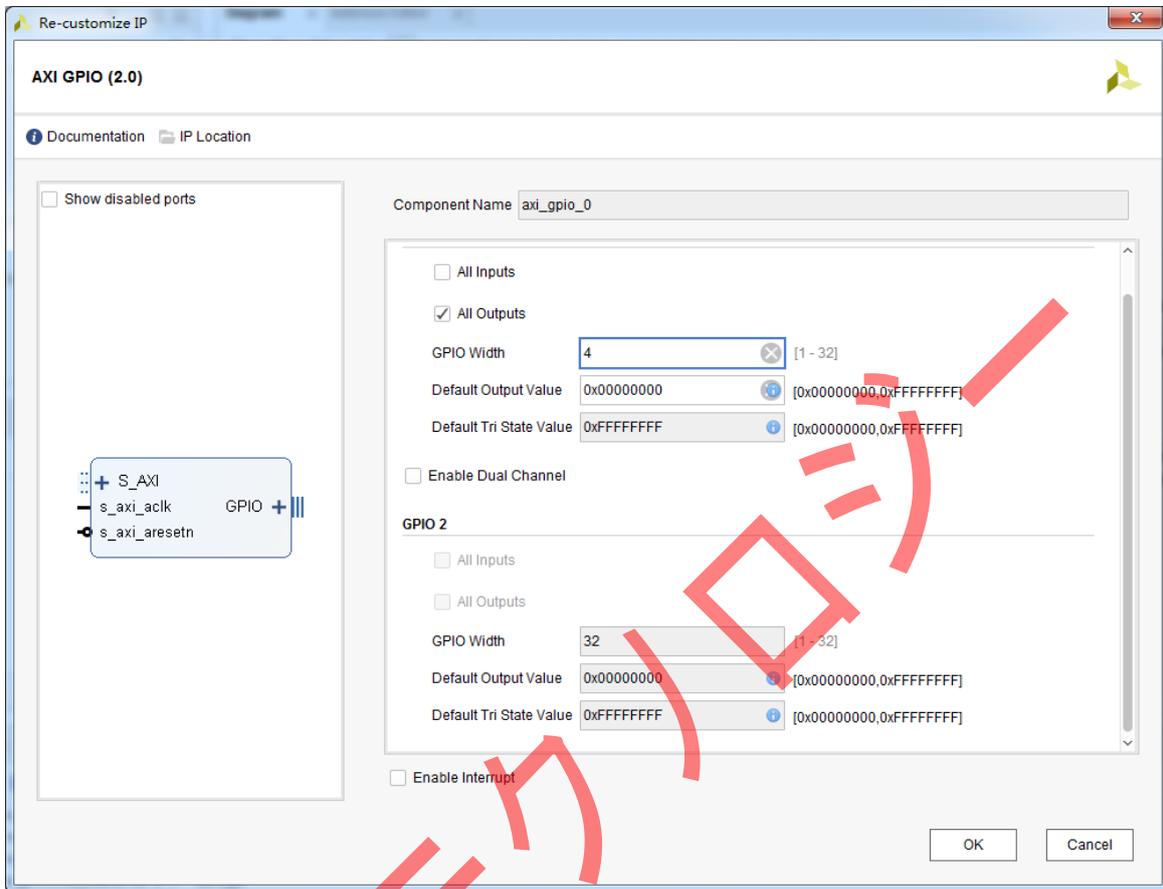


### 7.1.3 AXI GPIO を追加する

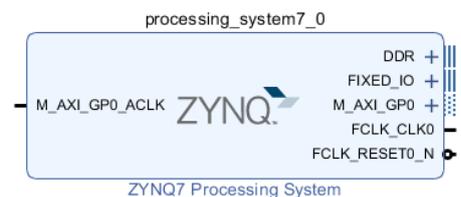
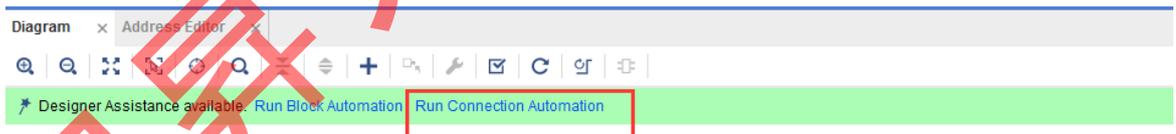
- 先程追加した “axi\_gpio\_0” の配置するパラメータをダブルクリックする。



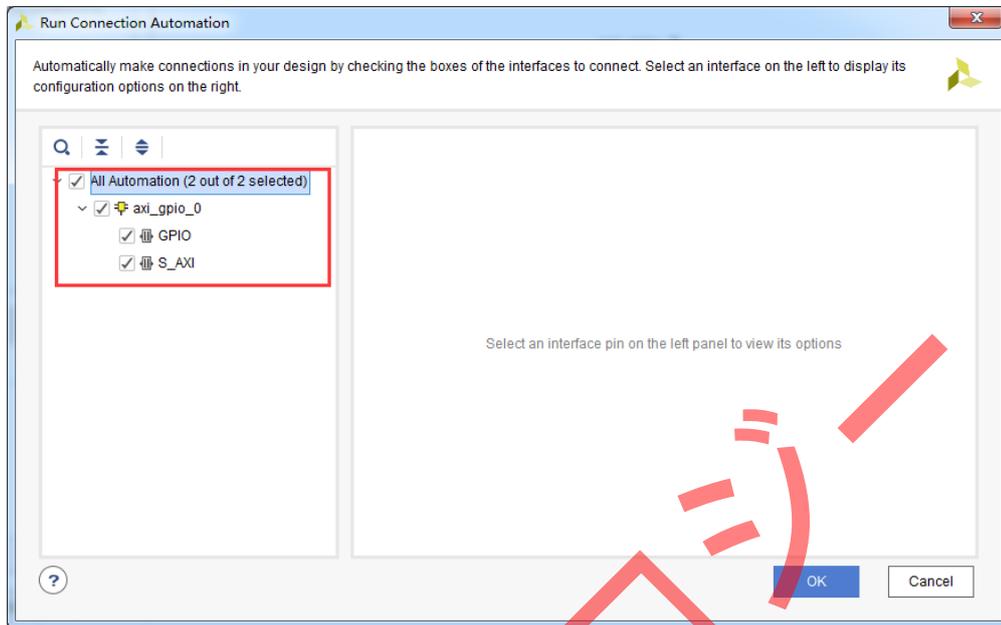
- 2) “All Outputs” を選択する。LED をコントロールしたから、アウトプットするだけでいい。“GPIO Width” 欄で 4 を入れて、4 つの LED をコントロールする。“OK” をクリックする。



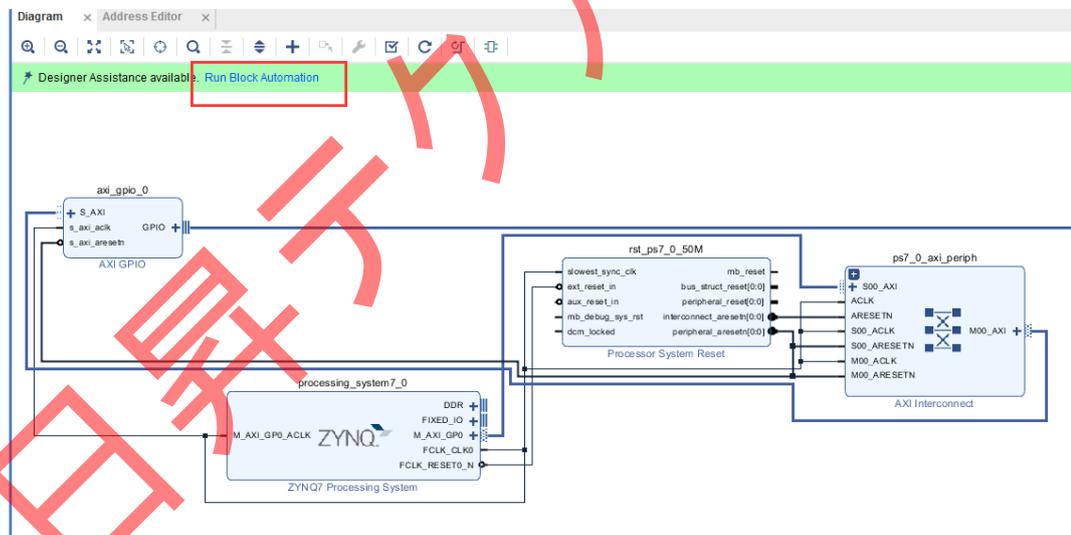
- 3) “Run Connection Automation” をクリックして、一部の自動連線ができる。



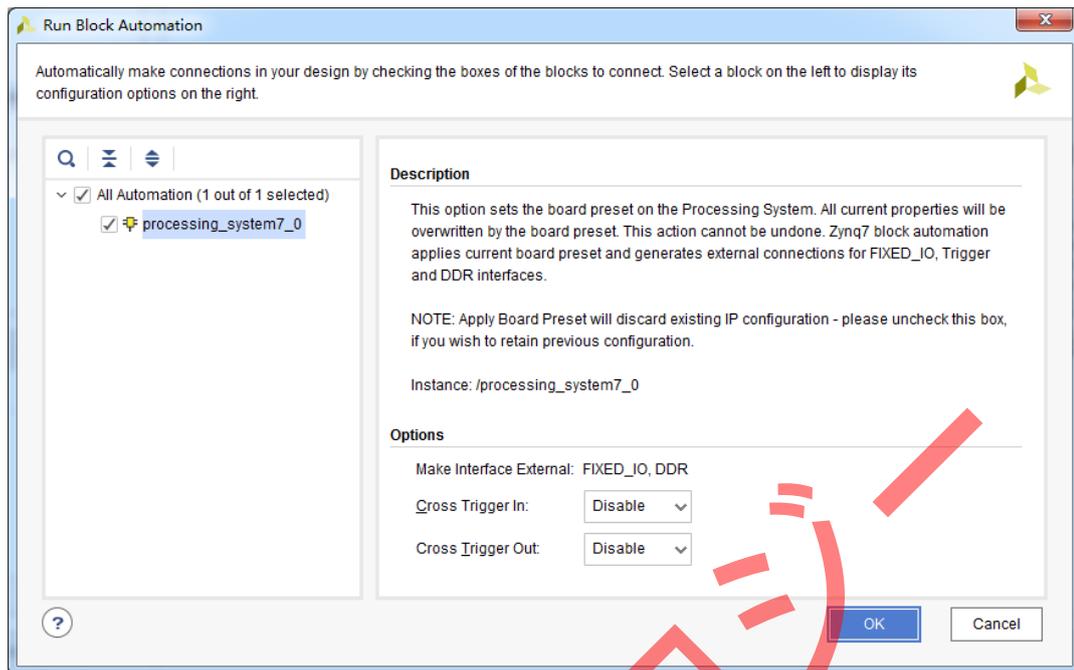
- 4) 自動的に接続するポートを選択する。ここはどこ全部選んで、“OK” をクリックする。



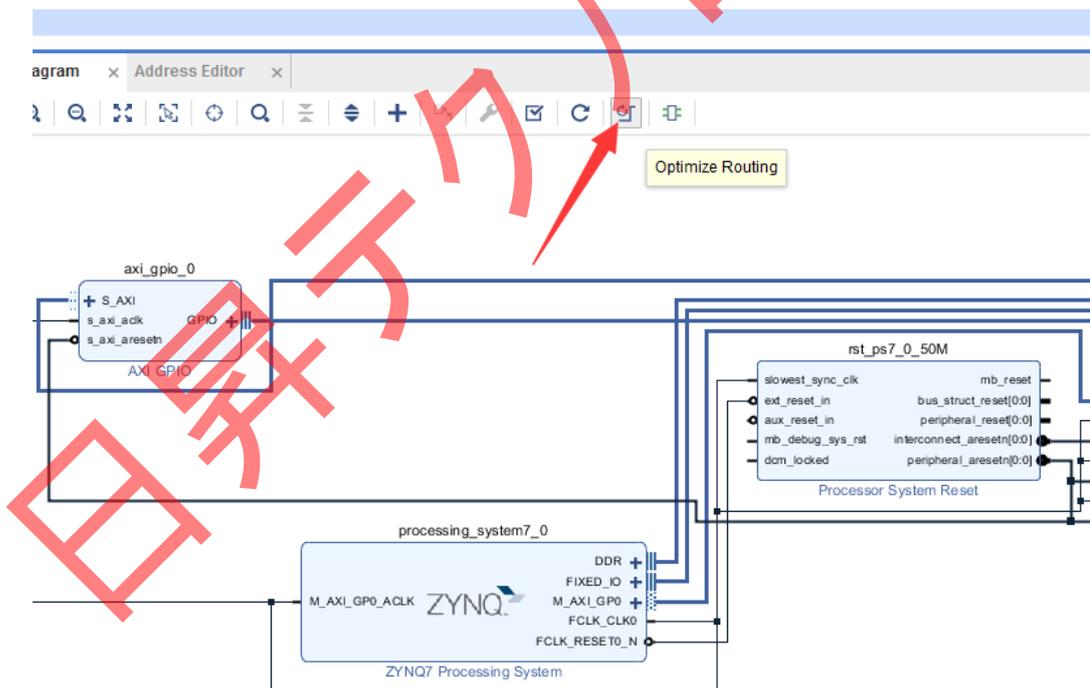
- 5) “Run Block Automation” をクリックする。



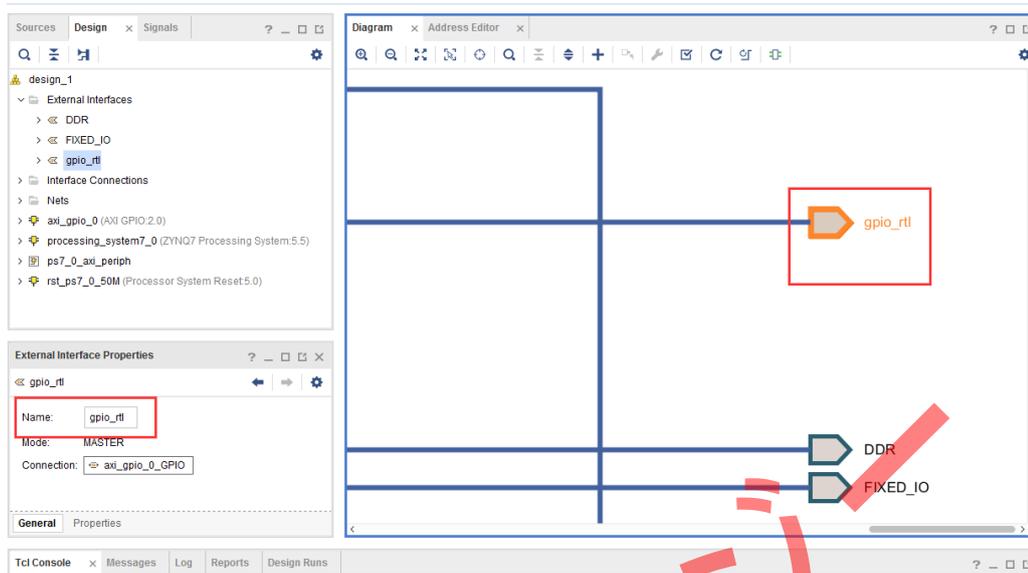
- 6) “OK” をクリックする



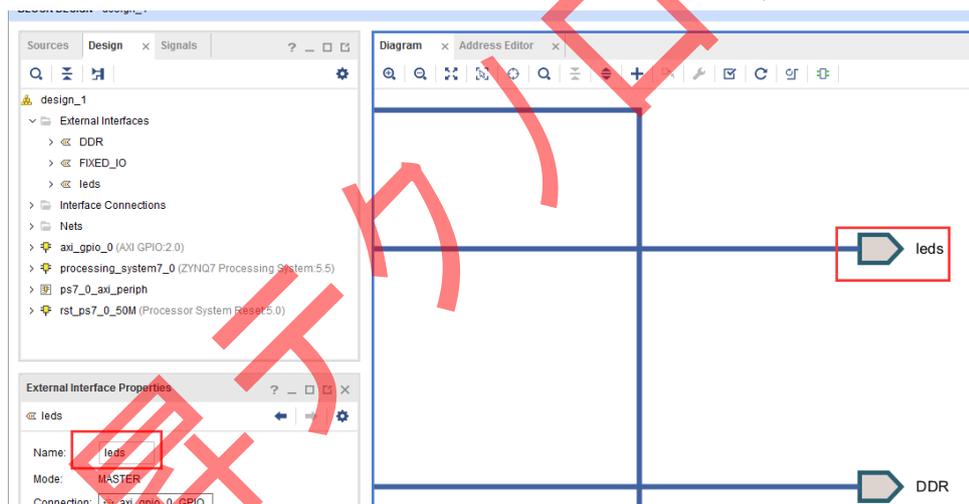
7) “Optimize Routing” をクリックして、レイアウトを最適化できる。



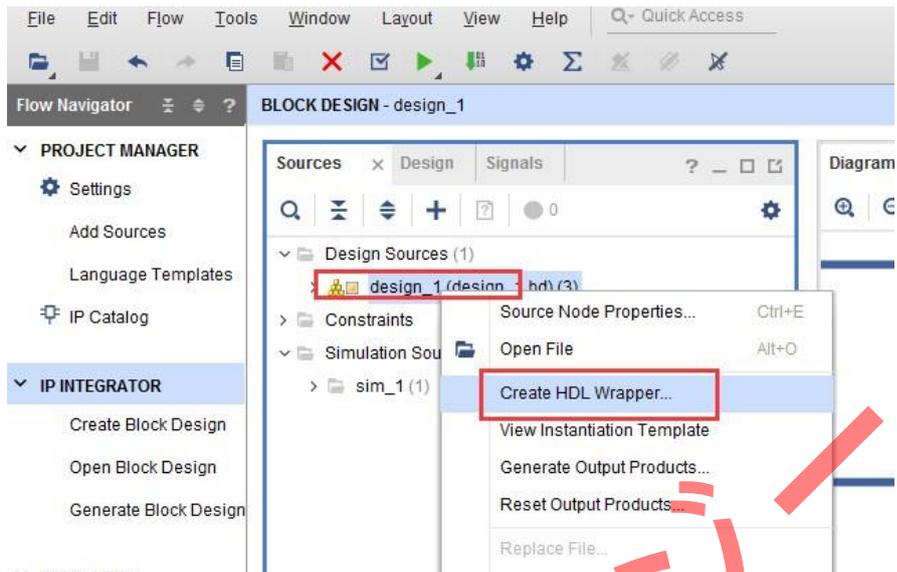
8) GPIO ポートの名前を変更する。



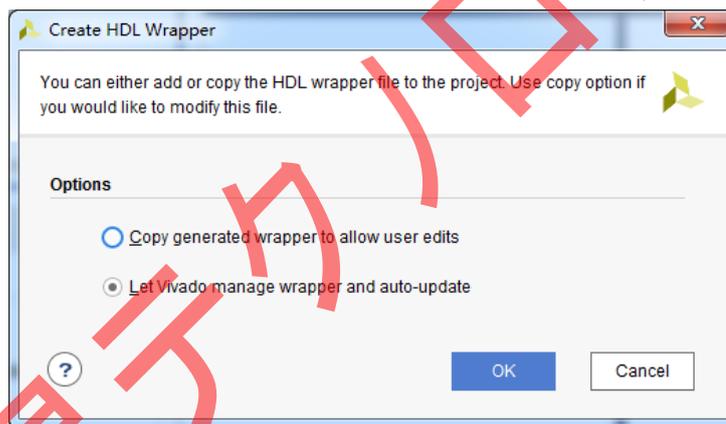
- 9) ネームを leds に変更する。



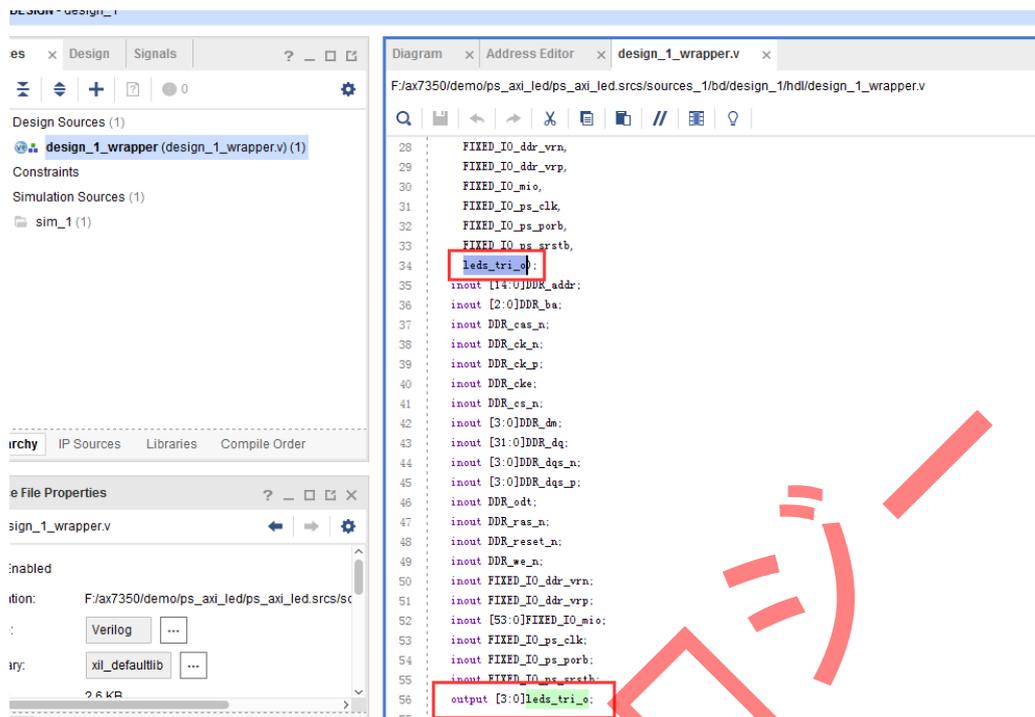
- 10) HDL ファイルを作成する。



- 11) “OK” クリックする。

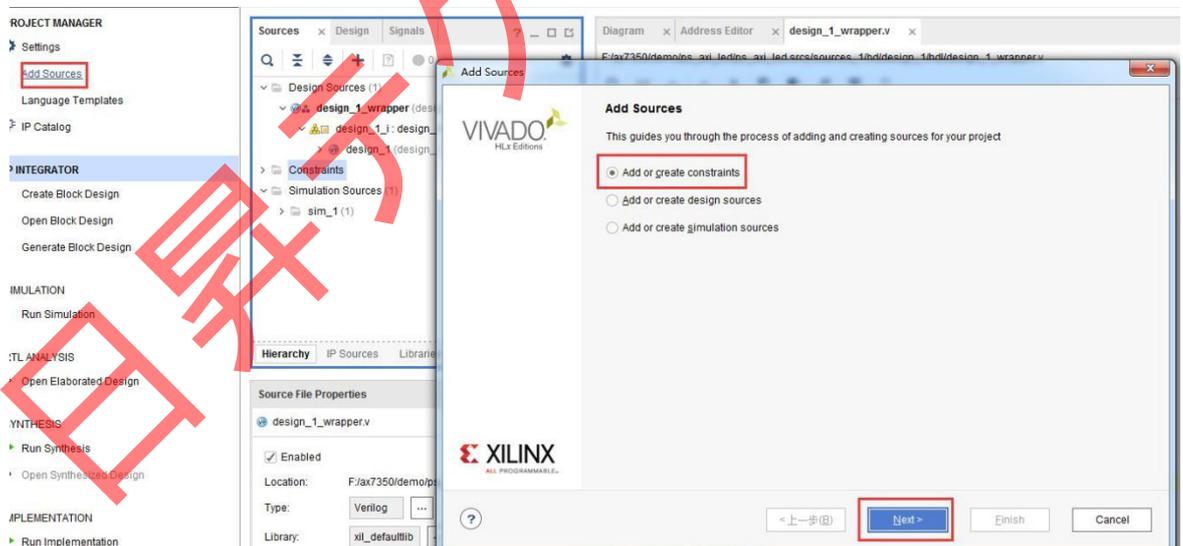


- 12) 生成した Verilog ファイルに、“leds\_tri\_o” のアウトプットポートが見える。これらのポートにピンを配る。

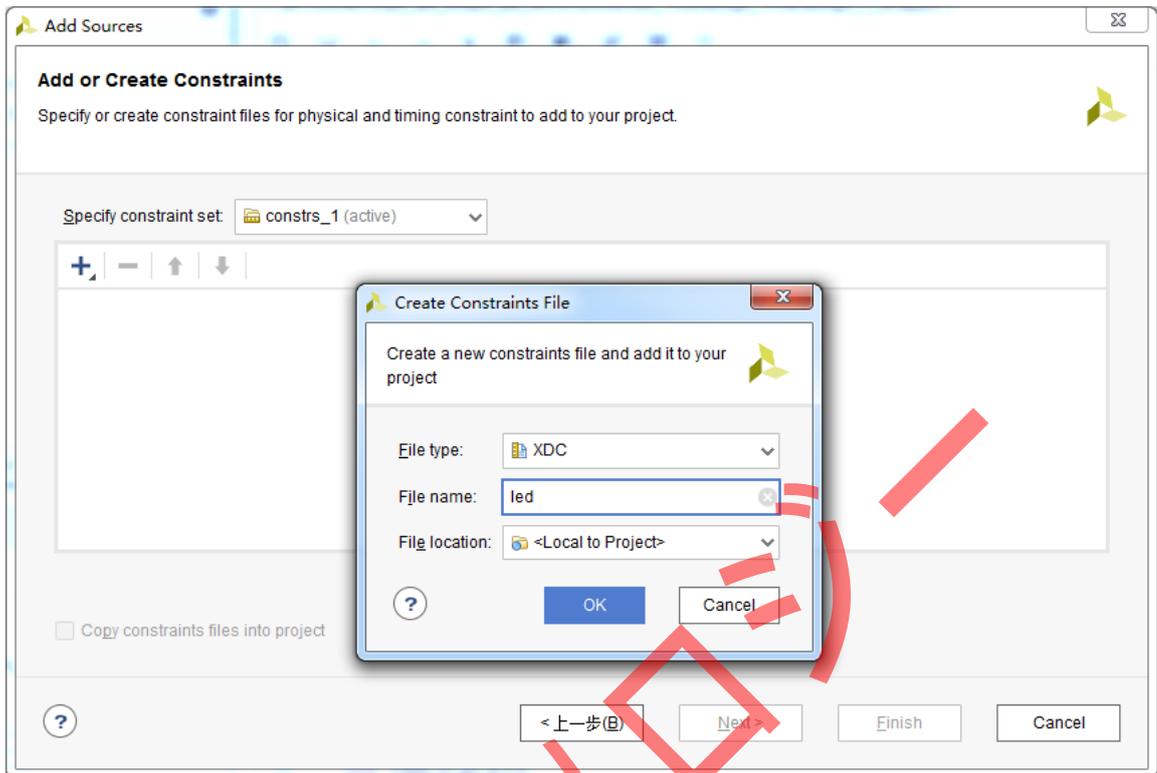


## 7.2 XDC ファイルで PL ピンを制約する

- 1) 新規 xdc 制約ファイルを作成する。



- 2) ファイルネームは led にする。



- 3) led.xdc に内容を追加する。ポート名はからなずトップファイルポートと一致する。

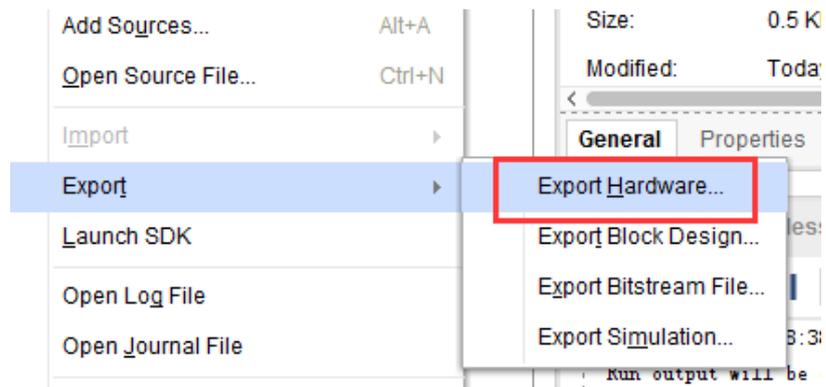
```
set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[3]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[2]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[1]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[0]}] set_property PACKAGE_PIN M14 [get_ports
{leds_tri_o[0]}] set_property PACKAGE_PIN M15 [get_ports
{leds_tri_o[1]}] set_property PACKAGE_PIN K16 [get_ports
{leds_tri_o[2]}] set_property PACKAGE_PIN J16 [get_ports
```

### 7.3 SDK プログラムをコンパイルする

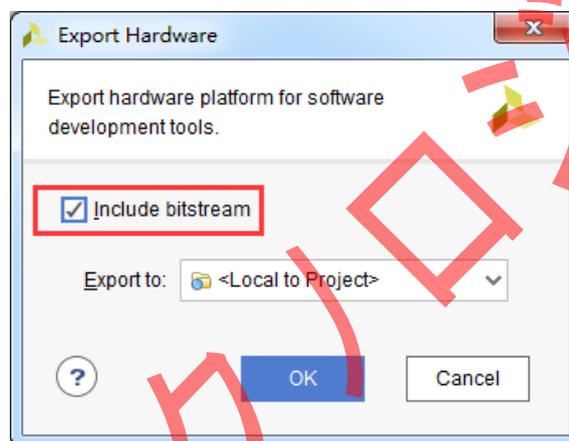
- 1) bit ファイルを生成する。



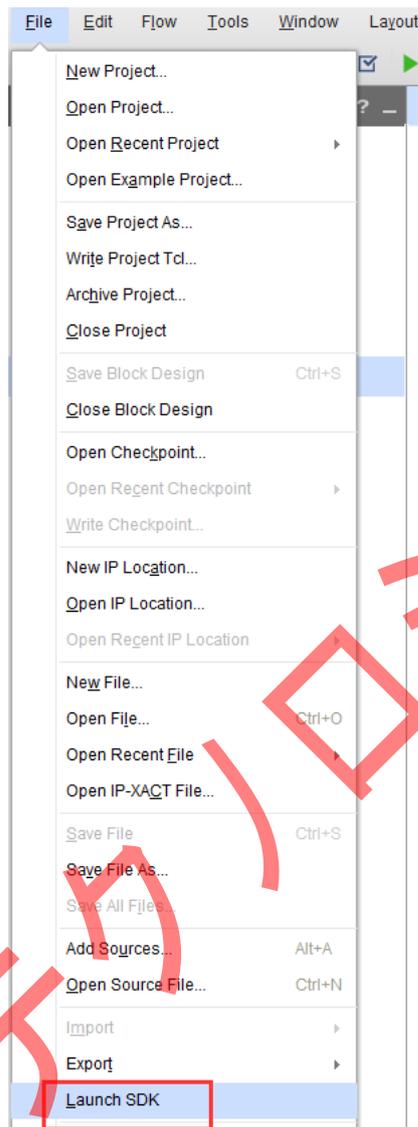
- 2) ハードウェアを書き出す。



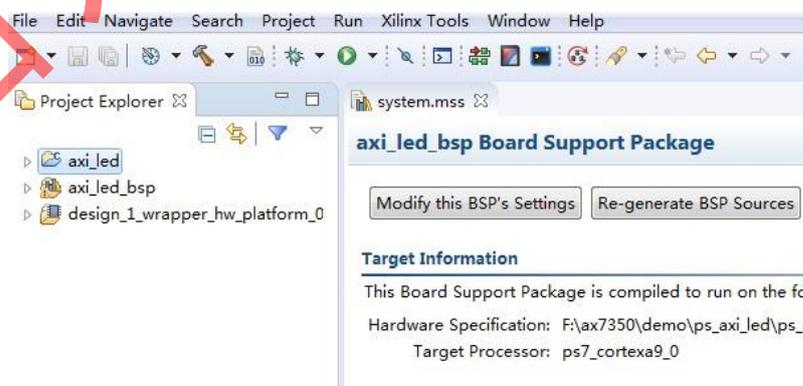
- 3) PL を使用するから、“Include bitstream” を選択し、“OK” をクリックする。



- 4) SDK を起動する。

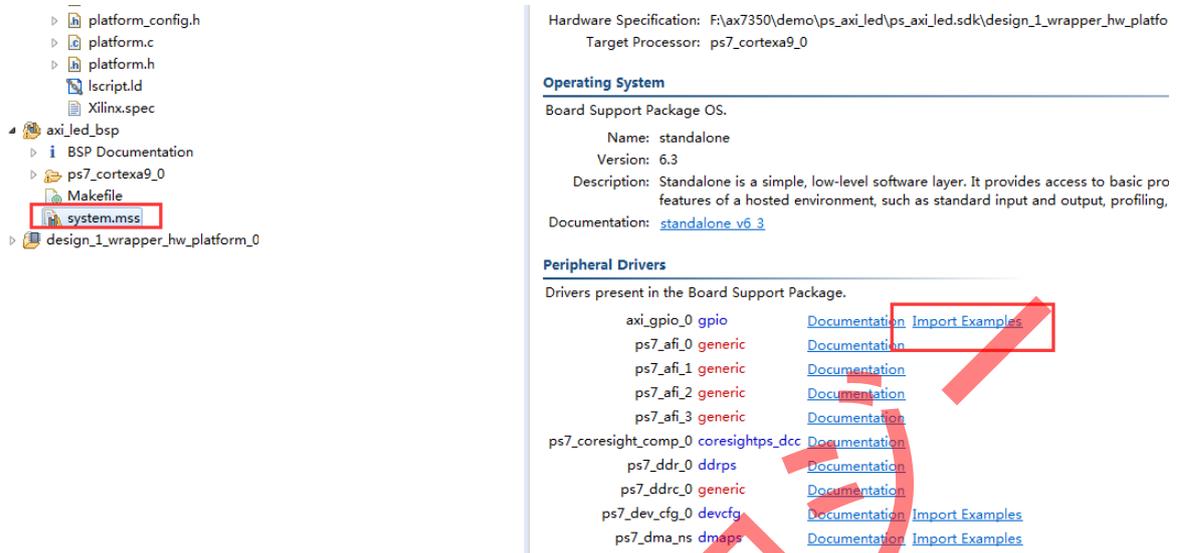


- 5) “axi\_led” 的という APP を作成して、プロジェクトテンプレートは Hello World に選ぶ。

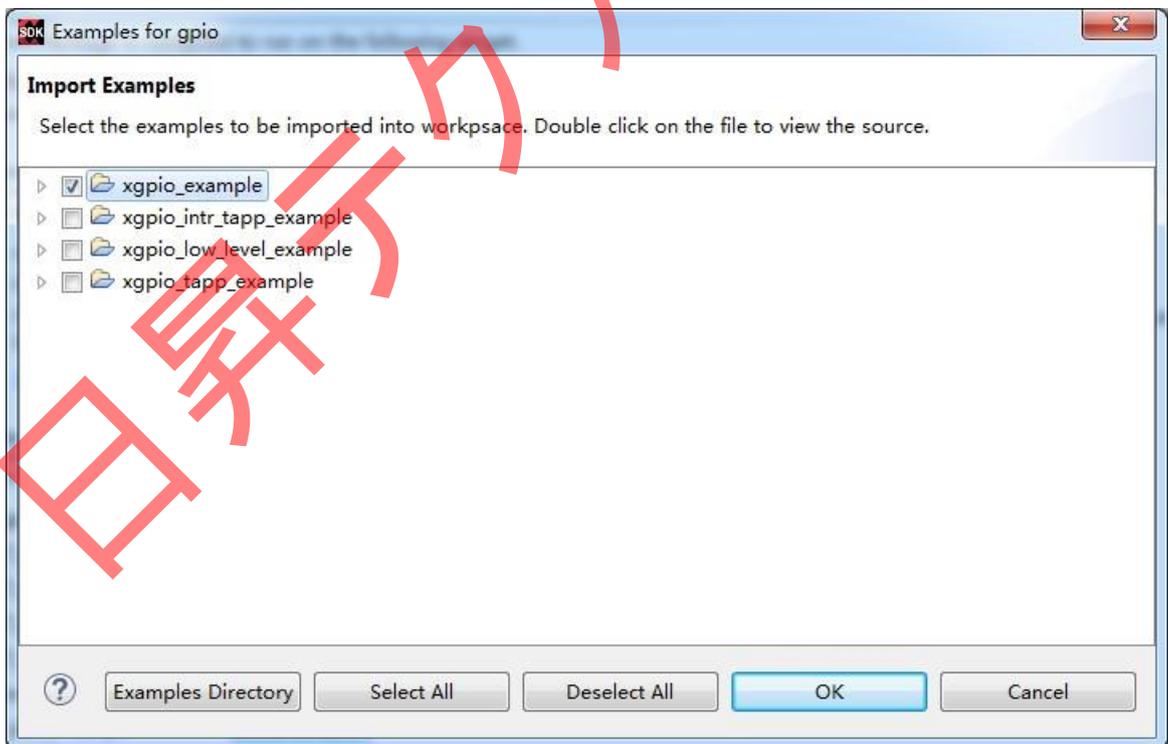


- 6) 慣れていない AXI GPIO をどうコントロールする？ SDK 内部のサンプルを試しておこう。  
7) “system.mss” をダブルクリック、“axi\_gpio\_0” を見つかる。“Documentation” をクリ

ックして、それに関するファイルを見る。ここを見れば、直接“Import Examples”をクリックする。



8) ポップアップされたダイアログはいくつかのサンプルがあり、ネームからどんな内容が見当がつける。ここは一番目の“xgpio\_example”を選ぶ。



9) サンプルは簡単で、少ないコードで、AXI GPIO の操作を完成した。

```

@pacem None
@return XST_FAILURE to indicate that the GPIO Initialization had failed.
@note This function will not return if the test is running.
.....
int main(void)
{
    int Status;
    volatile int Delay;

    /* Initialize the GPIO driver */
    Status = XGpio_Initialize(&Gpio, GPIO_EXAMPLE_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        xil_printf("Gpio Initialization Failed\r\n");
        return XST_FAILURE;
    }

    /* Set the direction for all signals as inputs except the LED output */
    XGpio_SetDataDirection(&Gpio, LED_CHANNEL, ~LED);

    /* Loop forever blinking the LED */
    while (1) {
        /* Set the LED to High */
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, LED);

        /* Wait a small amount of time so the LED is visible */
        for (Delay = 0; Delay < LED_DELAY; Delay++);

        /* Clear the LED bit */
        XGpio_DiscreteClear(&Gpio, LED_CHANNEL, LED);

        /* Wait a small amount of time so the LED is visible */
        for (Delay = 0; Delay < LED_DELAY; Delay++);

        xil_printf("Successfully ran Gpio Example\r\n");
        return XST_SUCCESS;
    }
}
    
```

中に GPIO に関する API 関数を使用している。ファイルで詳しいことを了解できる。この関数を選んで F3 を押せば具体的な定義も見える。これだけのインフォメーションがあつて、まだ AXI GPIO の使用を理解できないなら、C 言語基礎を補充すべきである。

### 7.4 ダウンロード及びデバッグ

1) SDK は数多くのサンプルを提供できるが、一部のサンプルは自分で変更するものである、この簡単な LED サンプルは変更せず、作動してみよう。予想の効果とかなり距離があつて、エラーの提示も出た。

```

.....
int main(void)
{
    int Status;
    volatile int Delay;

    /* Initialize the GPIO driver */
    Status = XGpio_Initialize(&Gpio, GPIO_EXAM
    if (Status != XST_SUCCESS) {
        xil_printf("Gpio Initialization Failure
        return XST_FAILURE;
    }

    /* Set the direction for all signals as is
    XGpio_SetDataDirection(&Gpio, LED_CHANNEL,

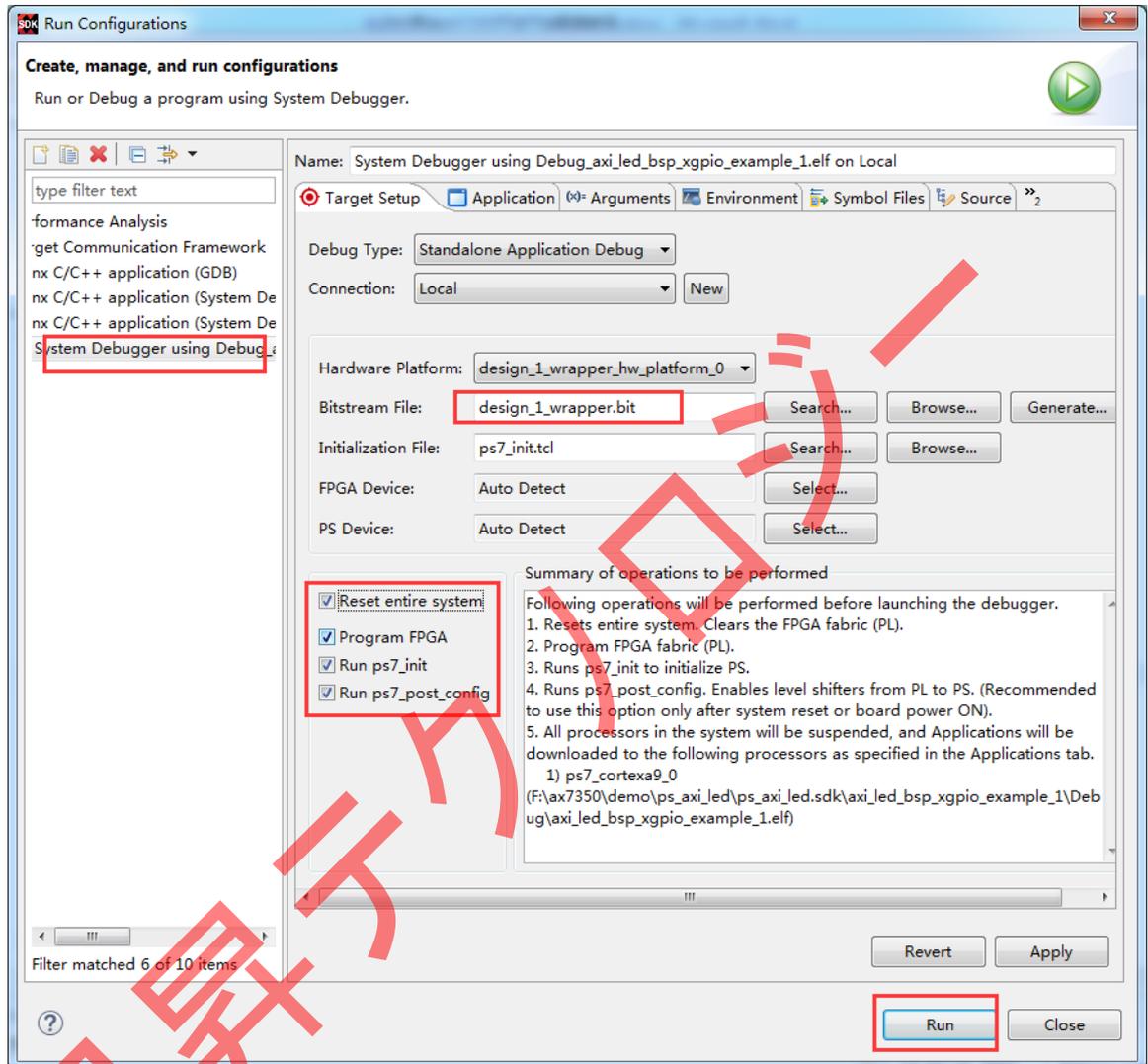
    /* Loop forever blinking the LED */
    while (1) {
        /* Set the LED to High */
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEI

        /* Wait a small amount of time so the
        for (Delay = 0; Delay < LED_DELAY; De

        /* Clear the LED bit */
        XGpio_DiscreteClear(&Gpio, LED_CHANNEI

        /* Wait a small amount of time so the
        for (Delay = 0; Delay < LED_DELAY; De
    }
}
    
```

2) 前の内容はもう“Run As” がシステムのリセットに効果抜群ということの説明した。PL のデザインは“Program FPGA”が必要である。PL が何度も更新されたら、改めてハードウェアを書き出すのが忘れないよう。下の画像のように配置して、再起動する。開発ボードの LED1 が素早くブリンクしている。



3) コードを変更して、2つの LED 両方もブリンクする。

```

/****************************************************************************
 *
 * ***** Include Files *****
 *
#include "xparameters.h"
#include "xgpio.h"
#include "xil_printf.h"
 *
 * ***** Constant Definitions *****
 *
#define LED 0x0F /* Assumes bit 0 of GPIO is connected to an LED */
 *
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 *
#define GPIO_EXAMPLE_DEVICE_ID XPAR_GPIO_0_DEVICE_ID
 *
 * The following constant is used to wait after an LED is turned on to make
 * sure that it is visible to the human eye. This constant must be at least

```

## 7.5 実験のまとめ

実験で PS は AXI パスに通じて PL をコントロールすることを了解した。しかし、ZYNQ の優位を表現されていない見たい。その原因は、LED ライトをコントロールすることなら、ARM も FGPA も楽に完成できる。でも LED をシリアルポートにしたらどうなるか？100 重シリアルポート通信や 8 重イーサネットなどのアプリを制御するようなことは、どこの SDC も完成できないと思う。これは ZYNQ しかできないこと。これも ZYNQ と普通の SDC との違いである。

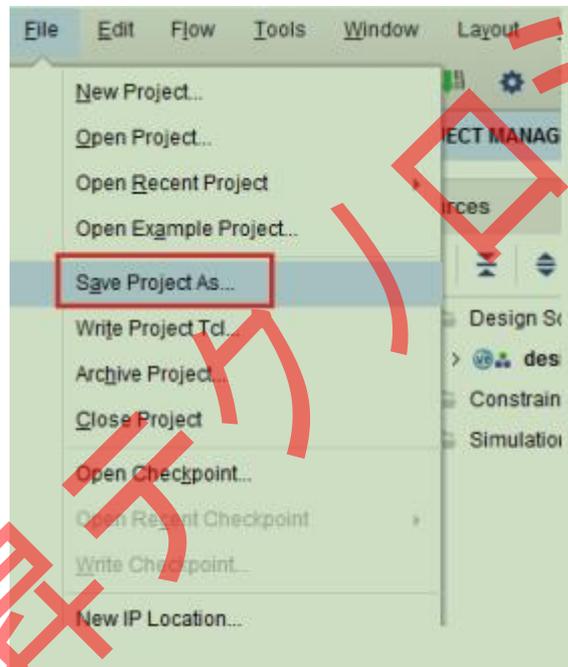
## 第八章 PS タイマーインタラプト実験

実験用 Vivado プロジェクトは “ps\_timer”。

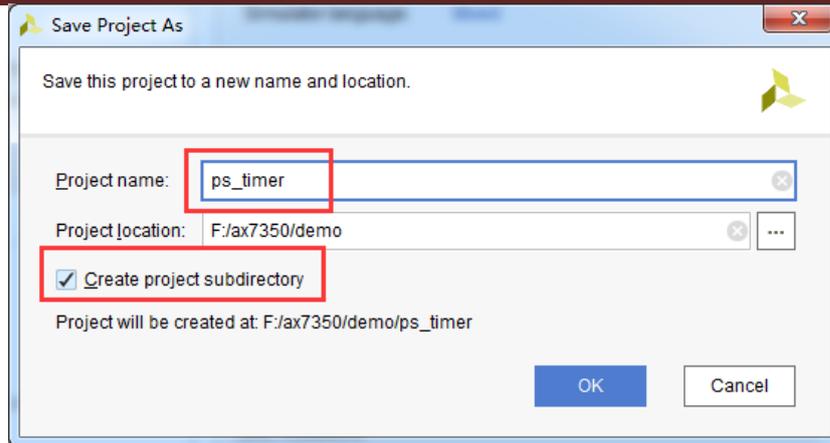
たくさんの SDC は内部にタイマーがある。ZYNQ の PS にもある。ZYNQ にはどんなペリフェラルがあるか、これらのペリフェラルはどんな特性あるか？それはエンジニアが了解すべくことなので、よく xilinx のマニュアル UG585 を読むのがお勧めです。

### 8.1 Vivado プロジェクトを作成する

- 1) 数個 Vivado プロジェクトを作成する。中にも同じ仕事は大量にあって、最適な解決法はプロジェクトをコピーして、変更する。“ps\_hello”を開く。
- 2) メニューで “File -> Save Project As...” をクリックする。

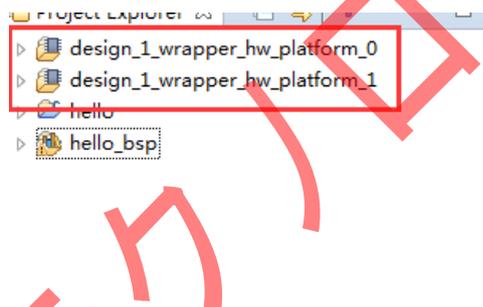


- 3) ポップアップされたダイアログに新規プロジェクト名 “ps\_timer” を入れる。サブディレクトリの作成を選択する。PS のタイマーはピンでアウトプットをするとき必要がないので、ピンを配置しない。

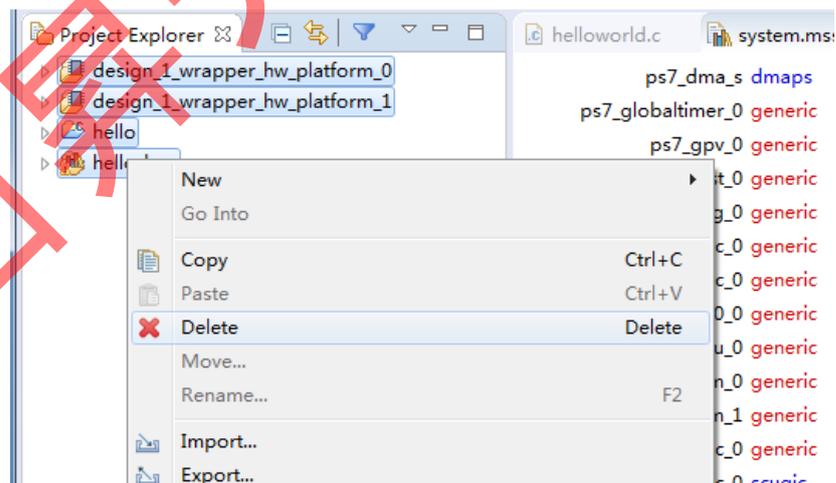


## 8.2 SDK をプログラミング

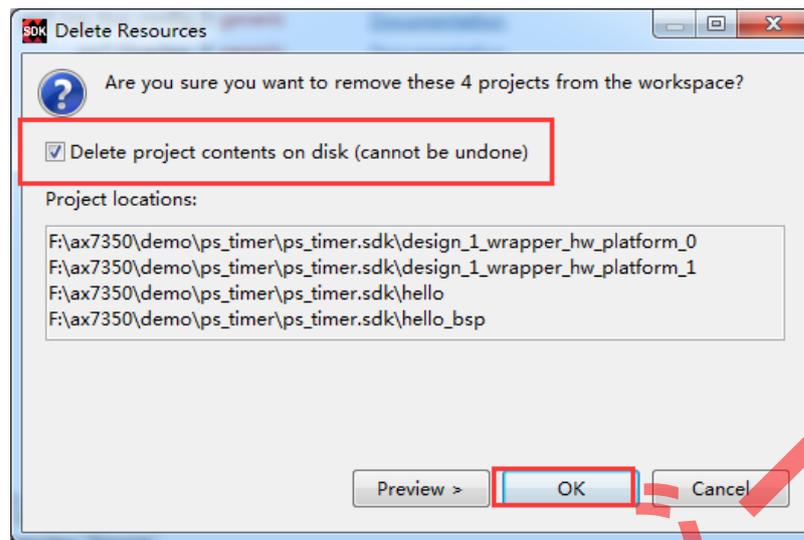
- 1) SDK を起動する。前のサンプルと違って、ここはハードウェアプラットフォームのファイルが一つ多くなった。



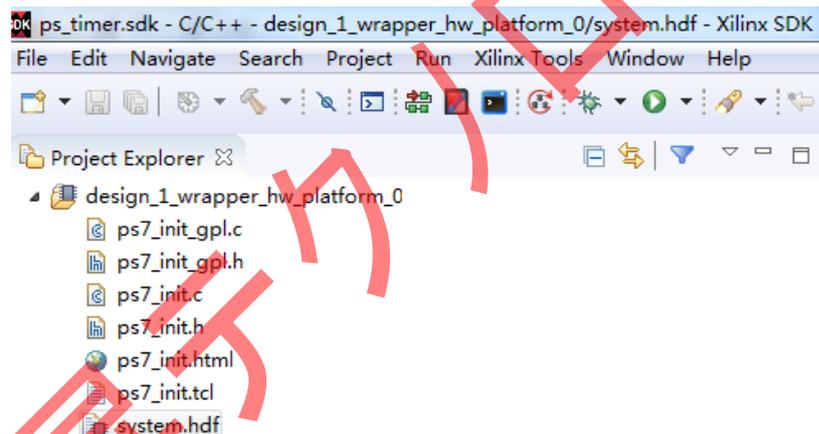
- 2) 他人の SDK プロジェクトを使うときも似たようなことがあって、全てを削除する。



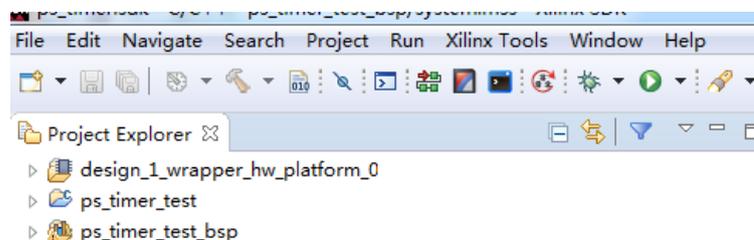
- 3) ファイルも削除する。



- 4) もう一度 Vivado に SDK を起動した、新しいハードウェアプラットフォームメッセージが見えてくる。



- 5) 新規プロジェクトを作成する。“ps\_timer\_test”という名で、テンプレートはHello Worldにする。



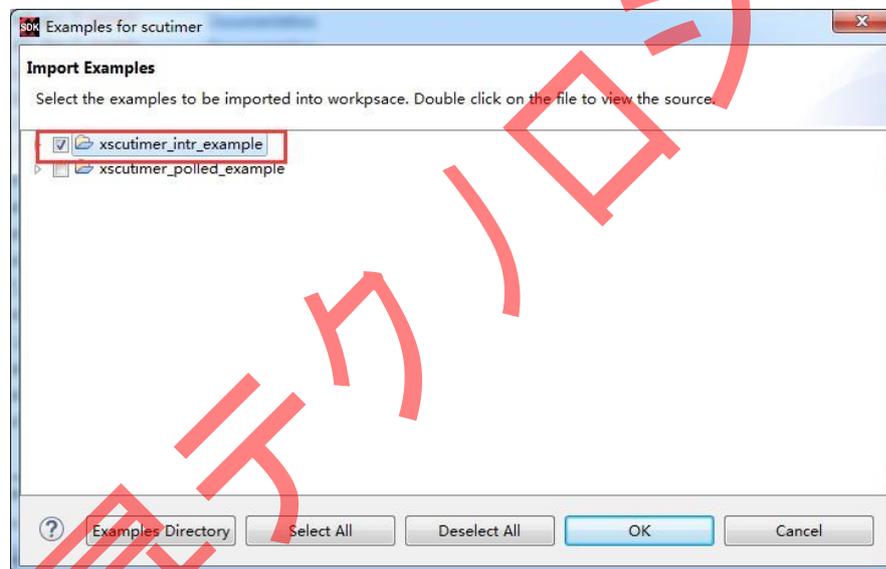
- 6) コーディングのときが来て、どこから始めるかは知らないでしょう。タイマーの使

い方も、インタラプトする方法も見当をつけない。いつもの通り、例をみよう。

ps7_ram_0 generic	<a href="#">Documentation</a>
ps7_ram_1 generic	<a href="#">Documentation</a>
ps7_scuc_0 generic	<a href="#">Documentation</a>
ps7_scugic_0 scugic	<a href="#">Documentation</a> <a href="#">Import Examples</a>
ps7_scutimer_0 scutimer	<a href="#">Documentation</a> <a href="#">Import Examples</a>
ps7_scuwdt_0 scuwdt	<a href="#">Documentation</a> <a href="#">Import Examples</a>
ps7_slcr_0 generic	<a href="#">Documentation</a>
ps7_uart_0 uartps	<a href="#">Documentation</a> <a href="#">Import Examples</a>
ps7_xadc_0 xadcps	<a href="#">Documentation</a> <a href="#">Import Examples</a>

Libraries

7) タイマーインタラプトの例一つあった。この例はインタラプトの例ってことはどう確認するか？“intr”から推測した。ということで、基礎がとても重要で、把握できないと、サンプルも探せない。



次はコードを読むと変更することである。すぐにコードを全部理解できないのも当然である。これから使用中で何度も練習するしかない。

8) 本実験はタイマーが一秒ごとにインタラプトして、メッセージをプリントアウトする。30秒後おわり。まずカウンターを変更するから。最大値はCPU頻度の半分にして、つまりカウンターのクロック頻度値になる。こうして、1秒ごとにインタラプトことになる。

```

/***** Include Files *****/
#include "xparameters.h"
#include "xscutimer.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"

/***** Constant Definitions *****/
/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are only defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#ifndef TESTAPP_GEN
#define TIMER_DEVICE_ID    XPAR_XSCUTIMER_0_DEVICE_ID
#define INTC_DEVICE_ID    XPAR_SCUGIC_SINGLE_DEVICE_ID
#define TIMER_IRPT_INTR   XPAR_SCUTIMER_INTR
#endif

#define TIMER_LOAD_VALUE   (XPAR_PS7_CORTEXA9_0_CPU_CLK_FREQ_HZ/2 - 1)

/***** Type Definitions *****/
/***** Macros (Inline Functions) Definitions *****/
/***** Function Prototypes *****/

```

9) カウンターの回数は3から30に変更する。

```

LastTimerExpired = TimerExpired;
/*
 * If it has expired a number of times, then stop the timer
 * counter and stop this example.
 */
if (TimerExpired == 30) {
    XScuTimer_Stop(TimerInstancePtr);
    break;
}
}
/*
 * Disable and disconnect the interrupt system.
 */
TimerDisableIntrSystem(IntcInstancePtr, TimerIntrId);

```

10) プリントのインフォメーションを追加する。

```

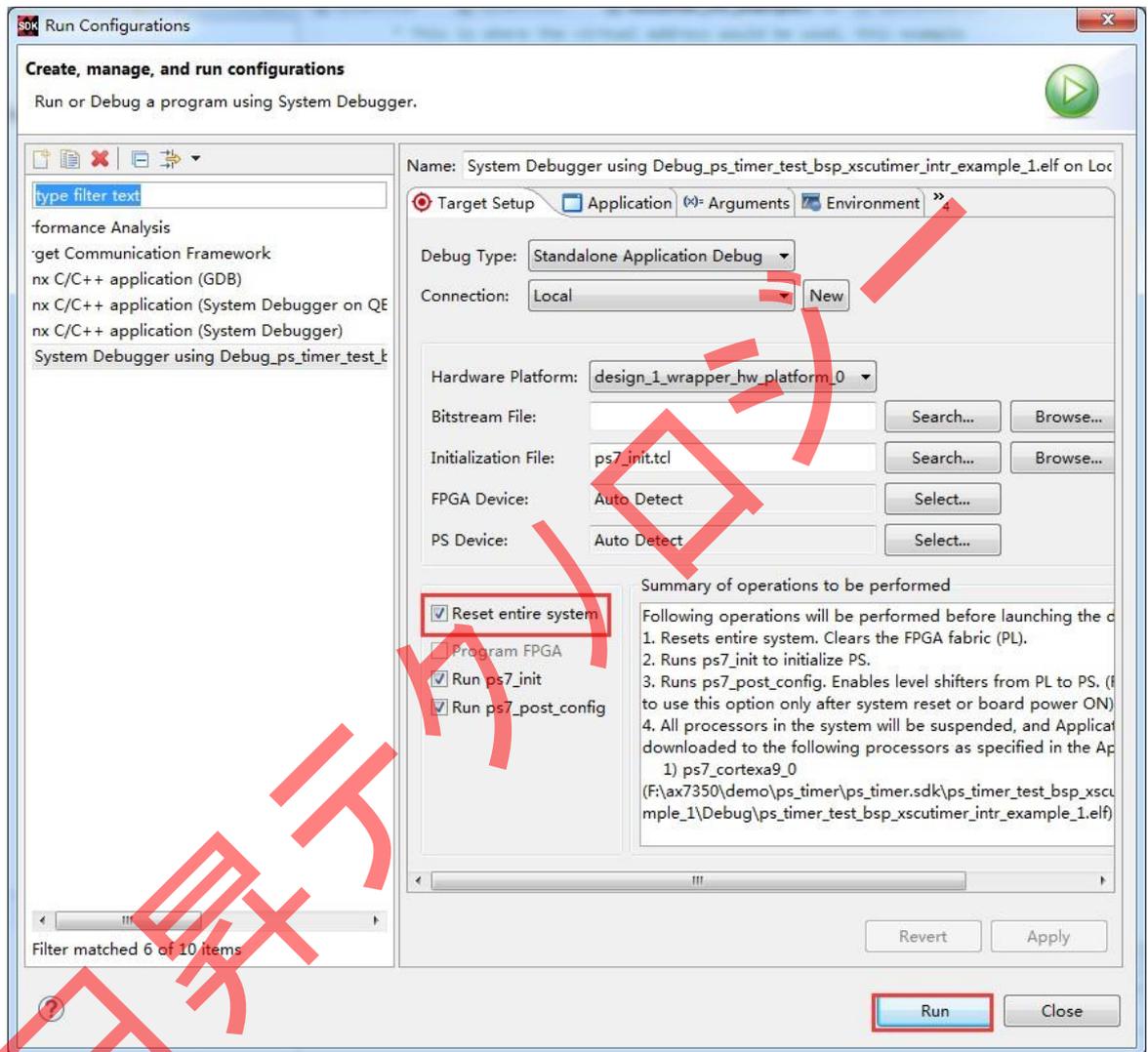
/*****
static void TimerIntrHandler(void *CallbackRef)
{
    XScuTimer *TimerInstancePtr = (XScuTimer *) CallbackRef;

    /*
     * Check if the timer counter has expired, checking is not necessary
     * since that's the reason this function is executed, this just shows
     * how the callback reference can be used as a pointer to the instance
     * of the timer counter that expired, increment a shared variable so
     * the main thread of execution can see the timer expired.
     */
    if (XScuTimer_IsExpired(TimerInstancePtr)) {
        XScuTimer_ClearInterruptStatus(TimerInstancePtr);
        printf("%d Second\n\r", TimerExpired);
        TimerExpired++;
        if (TimerExpired == 30) {
            XScuTimer_DisableAutoReload(TimerInstancePtr);
        }
    }
}

```

### 8.3 ダウンロードとデバッグ

- 1) PuTTY シリアルポートターミナルを開く。
- 2) デバッグプログラムのダウンロード前の章でもう説明した。



- 3) 予想通り、シリアルポートは秒ごとにメッセージをアウトプットする。

```
SCU Timer Interrupt Example Test
0 Second
1 Second
2 Second
3 Second
4 Second
5 Second
6 Second
7 Second
8 Second
9 Second
10 Second
11 Second
12 Second
13 Second
14 Second
15 Second
16 Second
17 Second
18 Second
19 Second
20 Second
21 Second
22 Second
23 Second
24 Second
25 Second
26 Second
27 Second
28 Second
29 Second
Successfully ran SCU Timer Interrupt Example Test
```

#### 8.4 実験纏め

実験中 SDK のサンプルを少し変更してタイマーとインタラプトの応用をかんせいした。操作簡単に見えるが、豊かな知識が含んでいる。タイマー、インタラプトの原理詳しく知る必要がある。これらの基本知識は ZYNQ を把握する必要条件である。

## 第九章 PL キーインタラプト

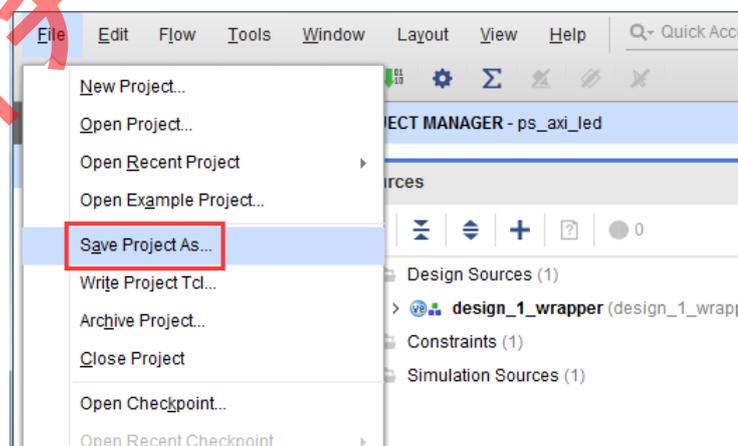
実験用 Vivado プロジェクトは “ps\_axi\_key”。

前の章にしたタイマーインタラプト実験は PS 内部のインタラプトで、本実験のインタラプトは PS から出てくる。PS は PL からのインタラプト信号は 16 も受けられて、立ち上がりエッジとハイレベルを通じてトグルする。本実験はボタンインタラプトを使用して LED をコントロールする。

Source	Interrupt Name	IRQ ID#	Status Bits (mpcore Registers)	Required Type	PS-PL Signal Name	I/O
PL	PL [2:0]	63:61	spi_status_0[31:29]	Rising edge/ High level	IRQF2P[2:0]	Input
	PL [7:3]	68:64	spi_status_1[4:0]	Rising edge/ High level	IRQF2P[7:3]	Input
Timer	TTC 1	71:69	spi_status_1[7:5]	High level	~	~
DMAC	DMAC[7:4]	75:72	spi_status_1[11:8]	High level	IRQP2F[27:24]	Output
IOP	USB 1	76	spi_status_1[12]	High level	IRQP2F[7]	Output
	Ethernet 1	77	spi_status_1[13]	High level	IRQP2F[6]	Output
	Ethernet 1 Wake-up	78	spi_status_1[14]	Rising edge	IRQP2F[5]	Output
	SDIO 1	79	spi_status_1[15]	High level	IRQP2F[4]	Output
	I2C 1	80	spi_status_1[16]	High level	IRQP2F[3]	Output
	SPI 1	81	spi_status_1[17]	High level	IRQP2F[2]	Output
	UART 1	82	spi_status_1[18]	High level	IRQP2F[1]	Output
	CAN 1	83	spi_status_1[19]	High level	IRQP2F[0]	Output
PL	PL [15:8]	91:84	spi_status_1[27:20]	Rising edge/ High level	IRQF2P[15:8]	Input
SCU	Parity	92	spi_status_1[28]	Rising edge	~	~
Reserved	~	95:93	spi_status_1[31:29]	~	~	~

### 9.1 Vivado プロジェクトを作成する

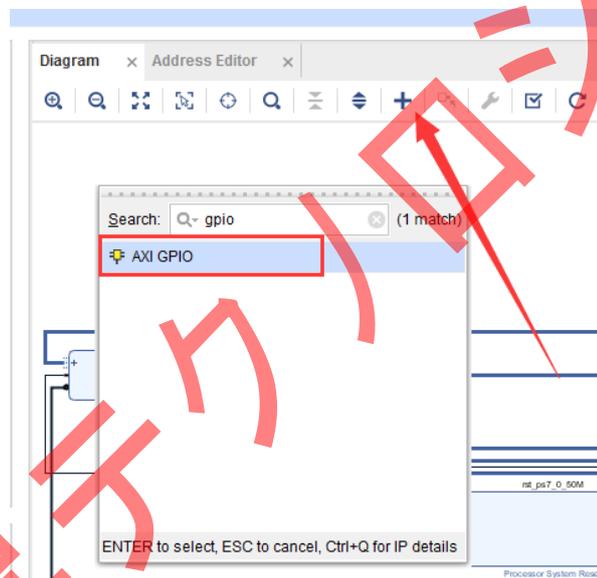
1) 本実験使用している Vivado プロジェクトは “ps\_axi\_led” だけを使う。このプロジェクトにボタン入力用の AXI GPIO を追加すればいい。メニューの “File -> Save Project As...” をクリックする。



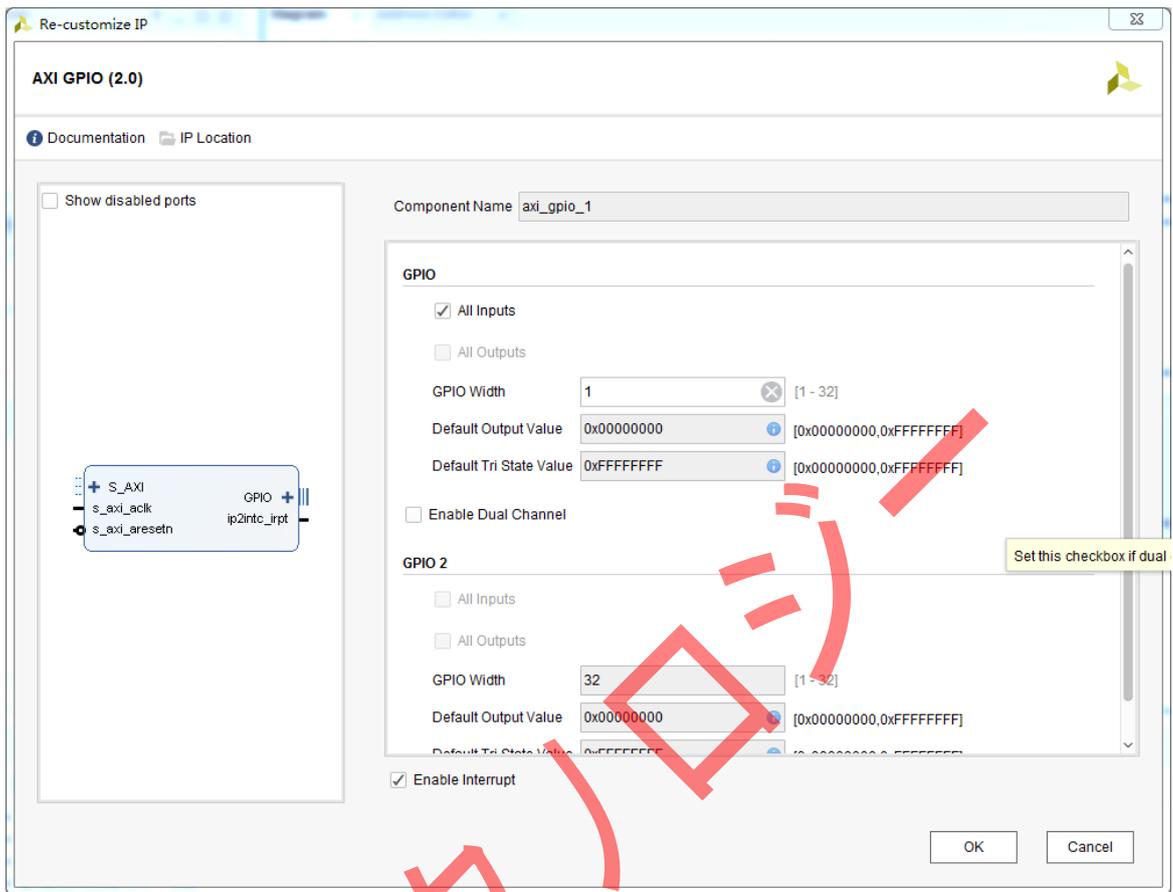
2) 新規プロジェクトは “ps\_axi\_key” にする。



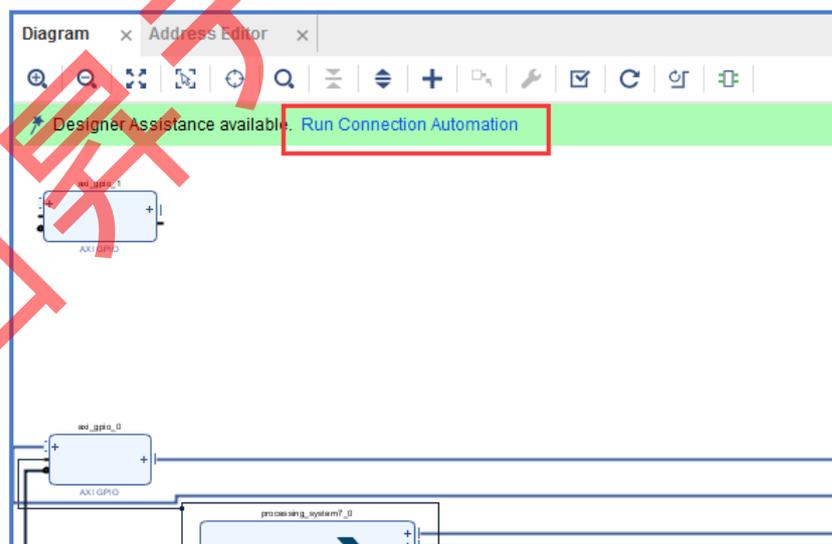
3) AXI GPIO を一つ追加する。



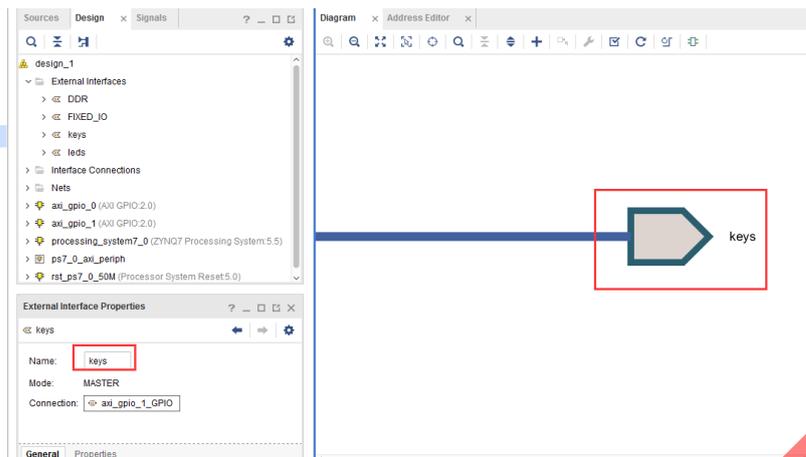
4) GPIO パラメータを配置して、全部をインプットにする。幅は1に設置し、インタラプトが発生できるように。



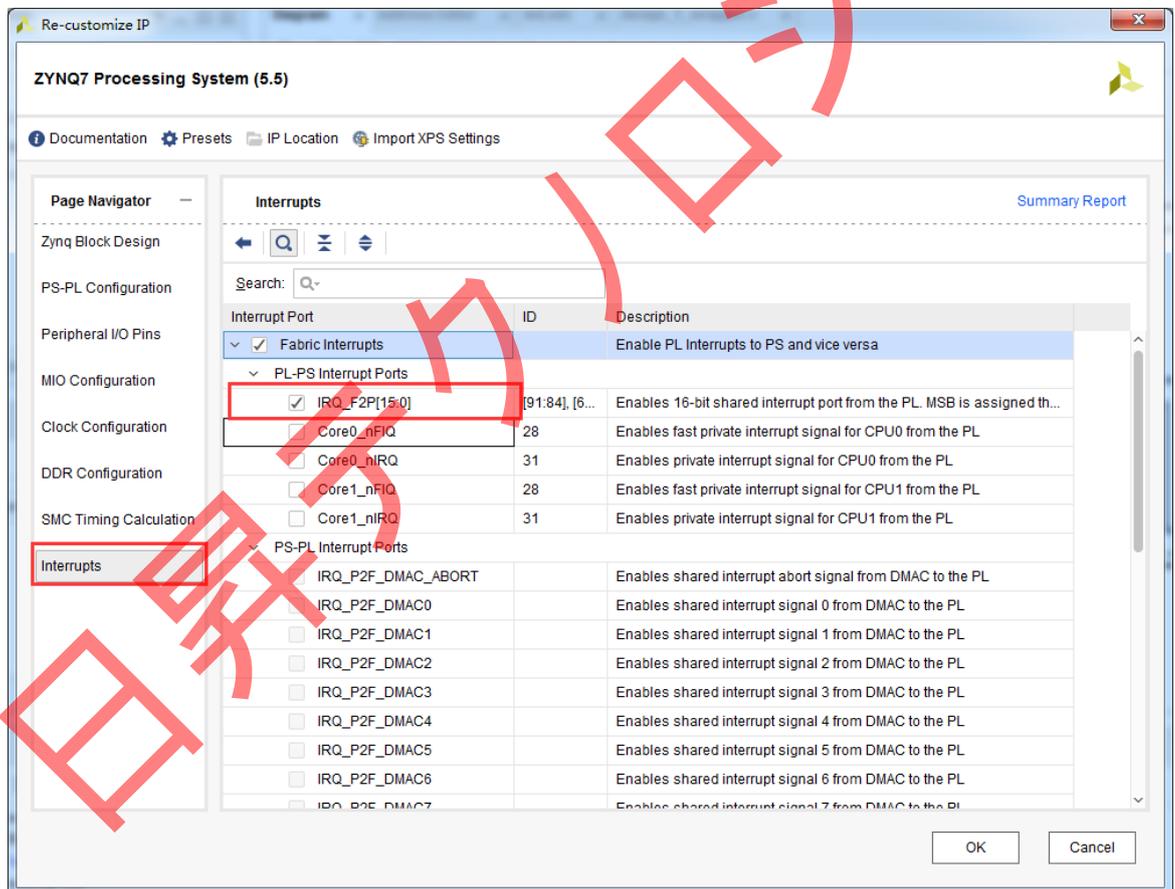
5) オートコネクションを使う。



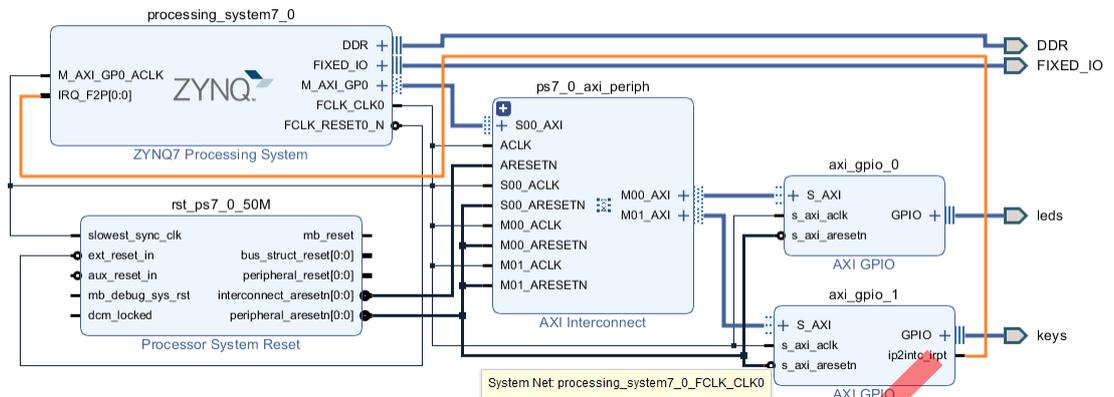
6) ポート名は keys に変更する。



7) ZYNQ プロセッサのインタラプトを配置して、IRQ\_F2P を選択する。



8) ip2intc\_irpt を IRQ\_F2Q に連結する。



9) xdc 制約ファイルを変更する。

```

set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[3]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[2]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[1]}] set_property IOSTANDARD LVCMOS33 [get_ports
{leds_tri_o[0]}] set_property PACKAGE_PIN M14 [get_ports
{leds_tri_o[0]}] set_property PACKAGE_PIN M15 [get_ports
{leds_tri_o[1]}] set_property PACKAGE_PIN K16 [get_ports
{leds_tri_o[2]}] set_property PACKAGE_PIN J16 [get_ports
{leds_tri_o[3]}]

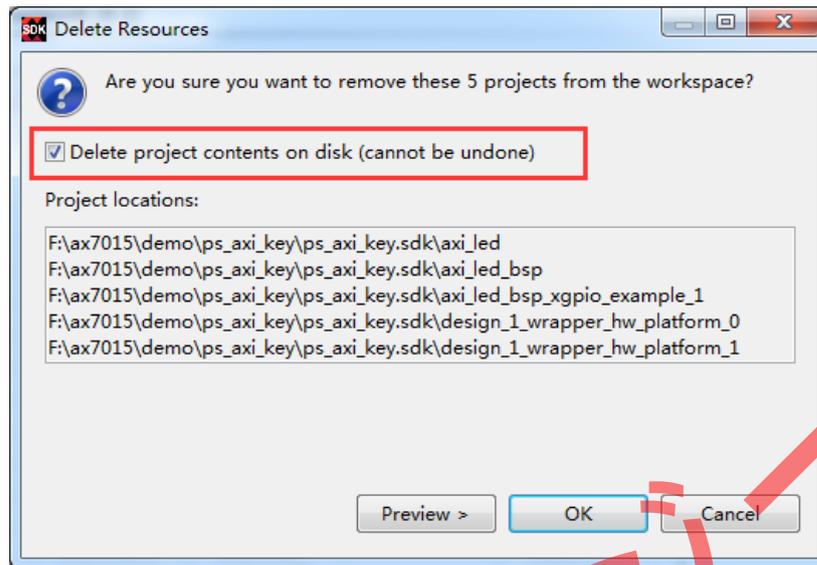
set_property IOSTANDARD LVCMOS33 [get_ports

```

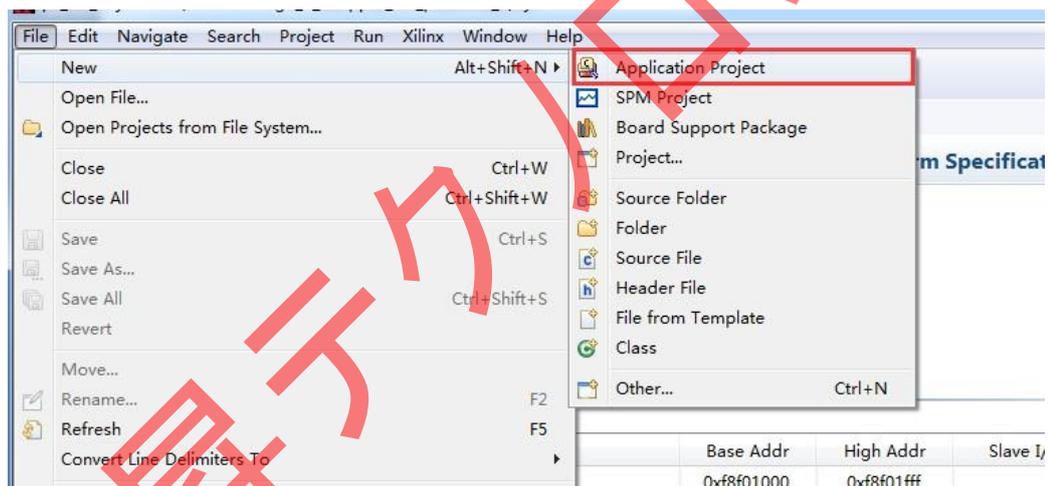
10) デザインを保存して、bit ファイルを生成する。

## 9.2 ダウンロードとデバッグ

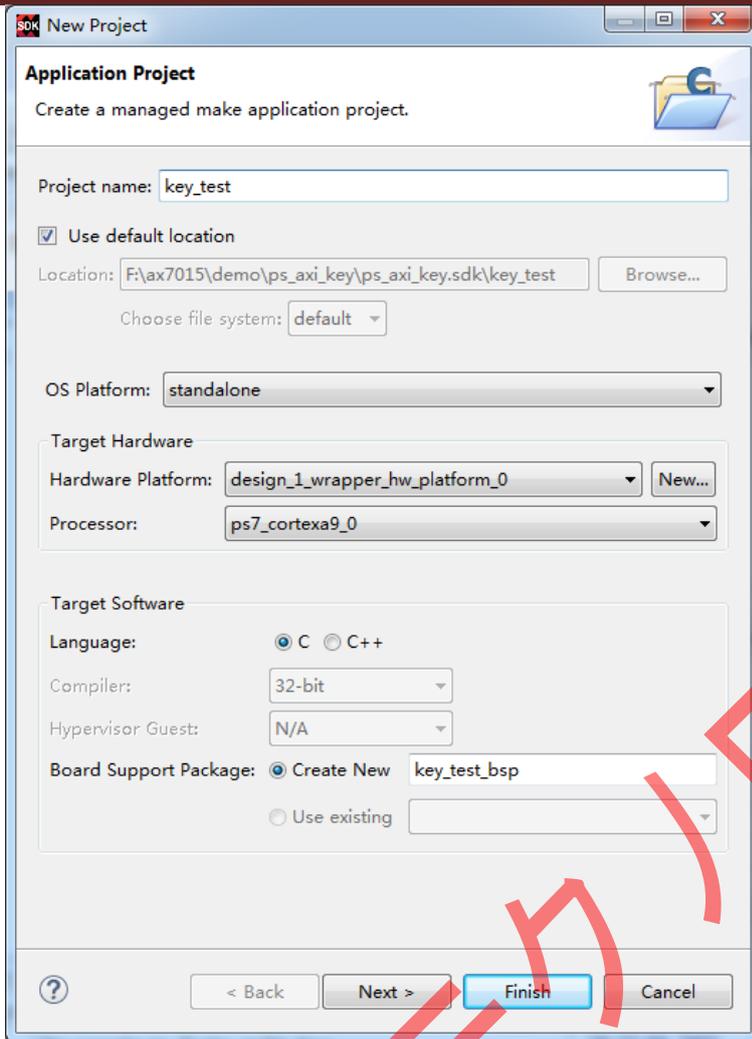
- 1) SDK を起動する。他のプロジェクトからコピーしたもので、SDK に必要ないファイルが入っている。それを全部削除してから、再起動する。



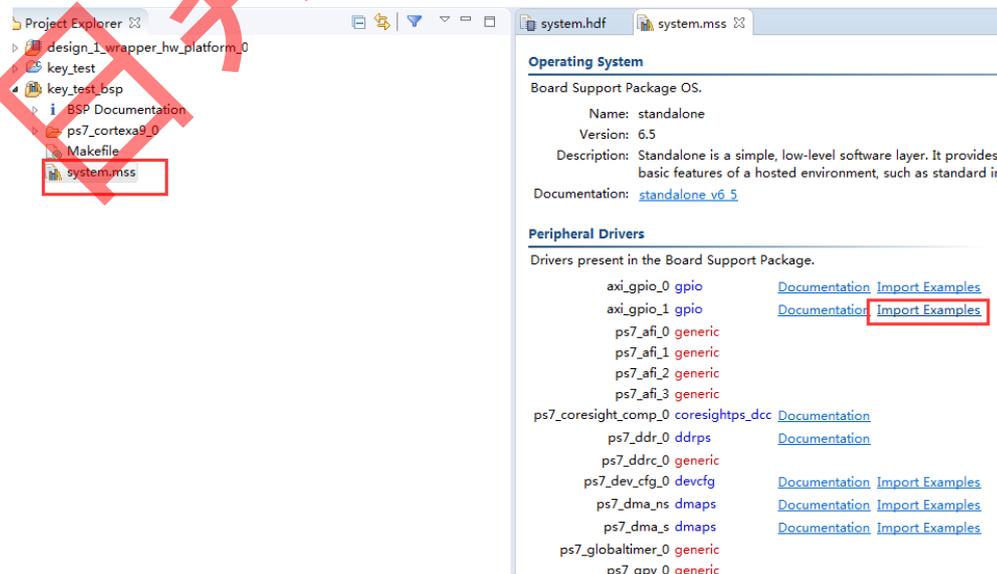
2) 新規 APP を作成。



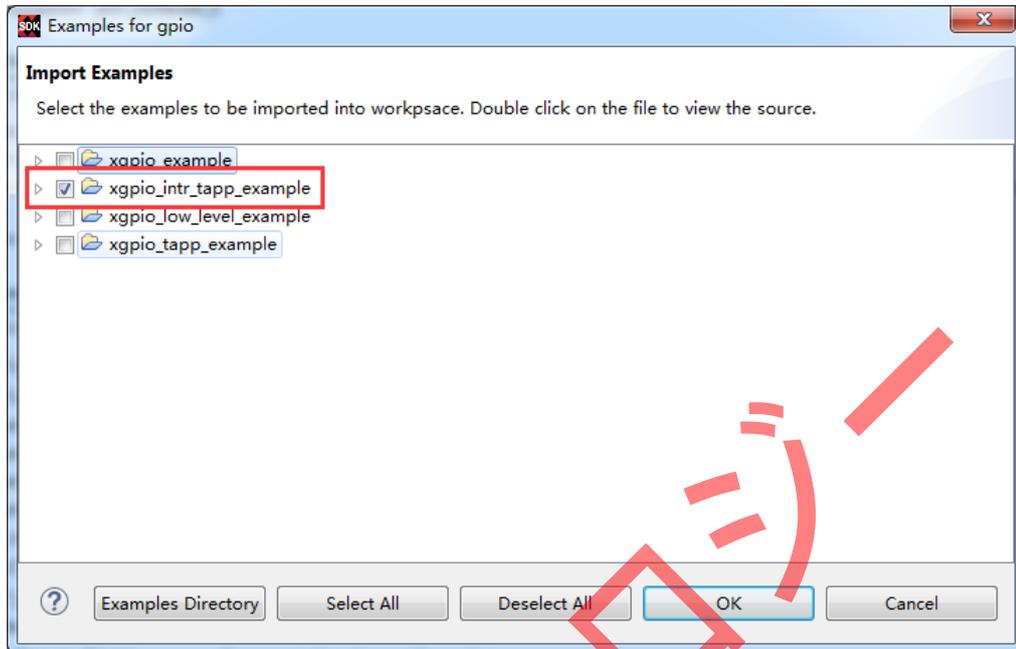
3) Project name を “key\_test” に設置する。



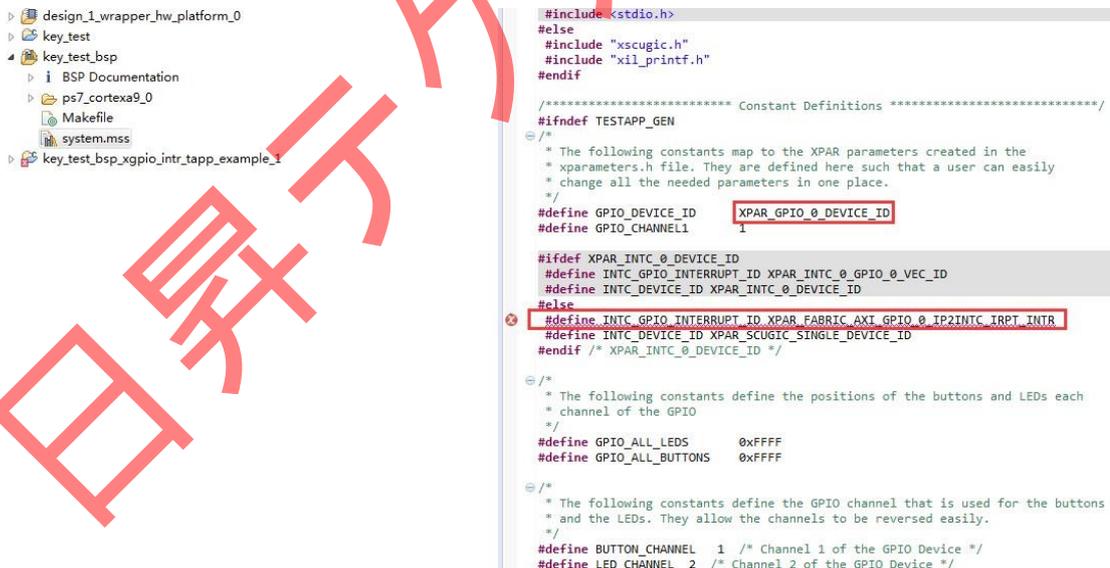
4) 前の言った通り、SDK プログラムに慣れていない状況で、できるだけ SDK 内部のサンプルを使って変更する。



5) “xgpio\_intr\_tapp\_example” を選択する。



6) サンプルをインポートしたあと、未知なエラーが出て、一部のコードを変更する必要がある。



7) 下の図に従って GPIO とインタラプト番号のマクロ定義をへんこうする。

```

system.mss  xgpio_intr_tapp_example.c
#include <stdio.h>
#else
#include "xscugic.h"
#include "xil_printf.h"
#endif

/***** Constant Definitions *****/
#ifndef TESTAPP_GEN
/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define GPIO_DEVICE_ID      XPAR_GPIO_1_DEVICE_ID
#define GPIO_CHANNEL1      1

#ifdef XPAR_INTC_0_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID XPAR_INTC_0_GPIO_0_VEC_ID
#define INTC_DEVICE_ID XPAR_INTC_0_DEVICE_ID
#else
#define INTC_GPIO_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_1_IP2INTC_IRPT_INTR
#define INTC_DEVICE_ID XPAR_SCUGIC_SINGLE_DEVICE_ID
#endif /* XPAR_INTC_0_DEVICE_ID */

/*
 * The following constants define the positions of the buttons and LEDs each
 * channel of the GPIO
 */
#define GPIO_ALL_LEDS      0xFFFF
#define GPIO_ALL_BUTTONS  0xFFFF

```

8) テストの遅延時間を変更して、ボタンを押す余裕があるように。

```

system.mss  xgpio_intr_tapp_example.c
 * The following constants define the GPIO channel that is used for the buttons
 * and the LEDs. They allow the channels to be reversed easily.
 */
#define BUTTON_CHANNEL 1 /* Channel 1 of the GPIO Device */
#define LED_CHANNEL 2 /* Channel 2 of the GPIO Device */
#define BUTTON_INTERRUPT XGPIO_IR_CH1_MASK /* Channel 1 Interrupt Mask */

/*
 * The following constant determines which buttons must be pressed at the same
 * time to cause interrupt processing to stop and start
 */
#define INTERRUPT_CONTROL_VALUE 0x7

/*
 * The following constant is used to wait after an LED is turned on to make
 * sure that it is visible to the human eye. This constant might need to be
 * tuned for faster or slower processor speeds.
 */
#define LED_DELAY 100000

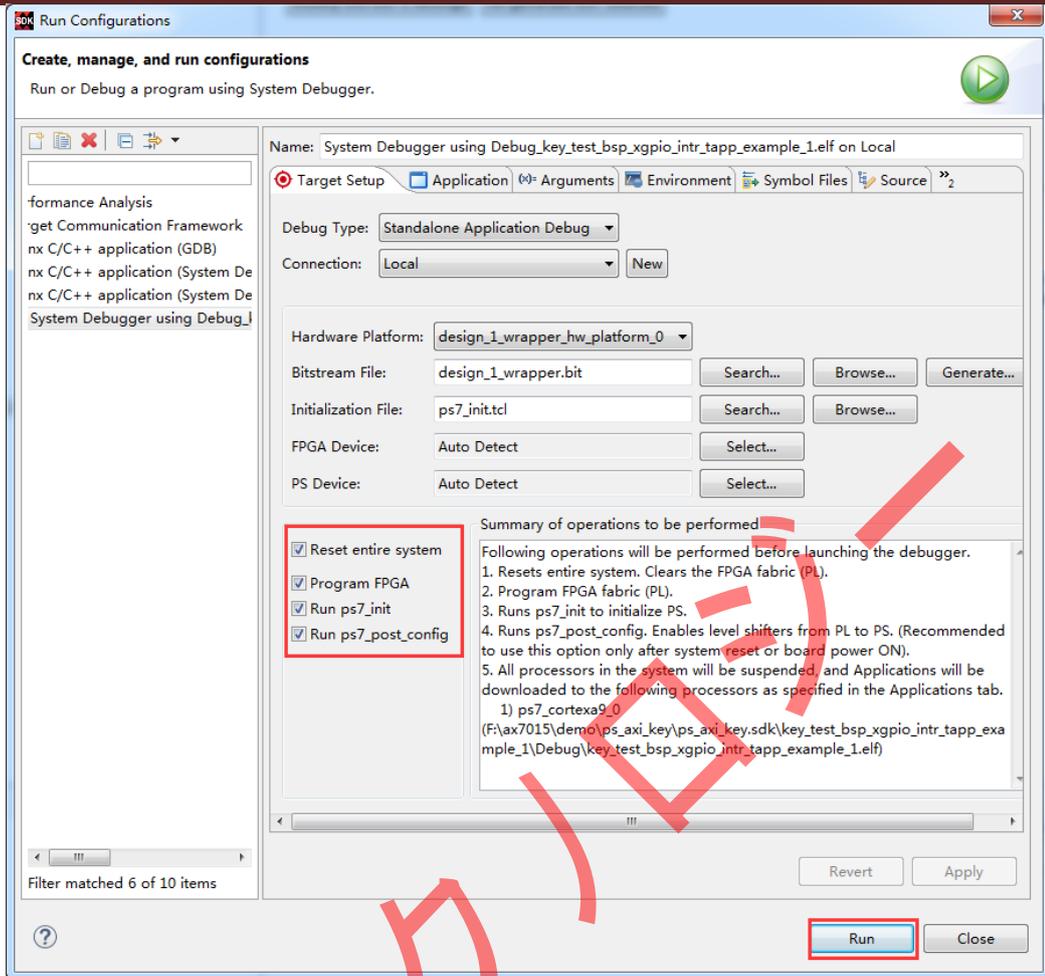
#endif /* TESTAPP_GEN */

#define INTR_DELAY 0x2FFFFFFF

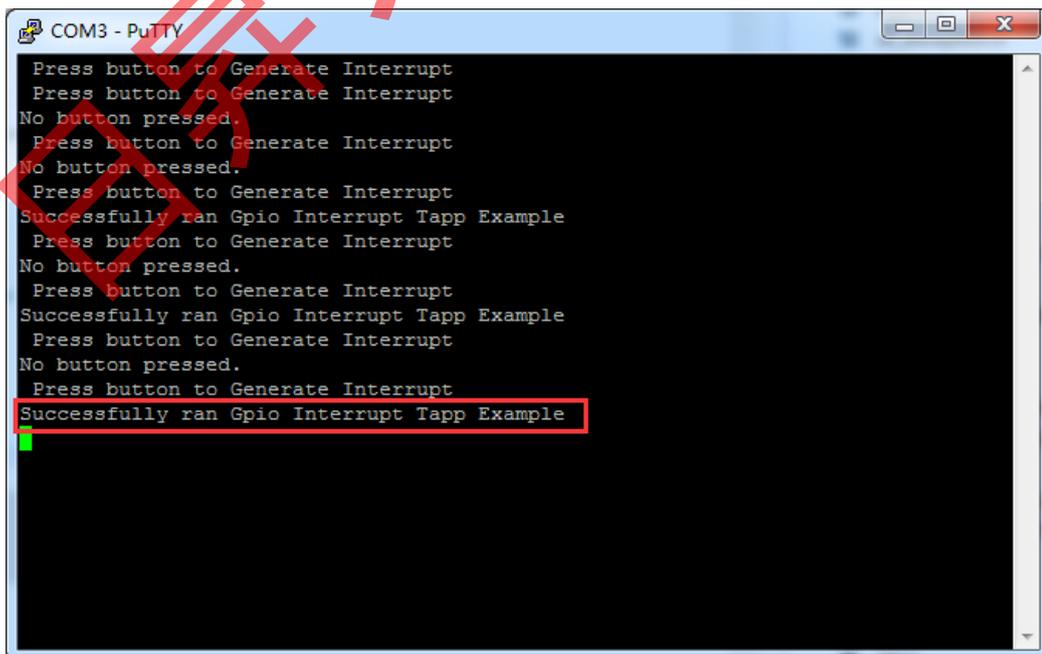
#ifdef XPAR_INTC_0_DEVICE_ID
#define INTC_DEVICE_ID XPAR_INTC_0_DEVICE_ID
#define INTC XIntc
#define INTC_HANDLER XIntc_InterruptHandler
#else
#define INTC_DEVICE_ID XPAR_SCUGIC_SINGLE_DEVICE_ID
#define INTC XScuGic
#define INTC_HANDLER XScuGic_InterruptHandler

```

9) シリアルターミナルを開き、プログラムを起動する。



10) 長時間ボタンを押さないと、シリアルポータルは “No button pressed.” を表示する。 “PL KEY1” を押すと、 “Successfully ran Gpio Interrupt Tapp Example” が表示される。



### 9.3 実験まとめ

PL 側は PS にインタラプト信号を送送でき、PL と PS のデータ交換 PL の効率を高めた。大数、低遅延の APP にインタラプト処理が必要である。

日昇テクノロジー

## 第十章 イーサネット実験 (LWIP)

実験用 Vivado プロジェクトは “net\_test”。

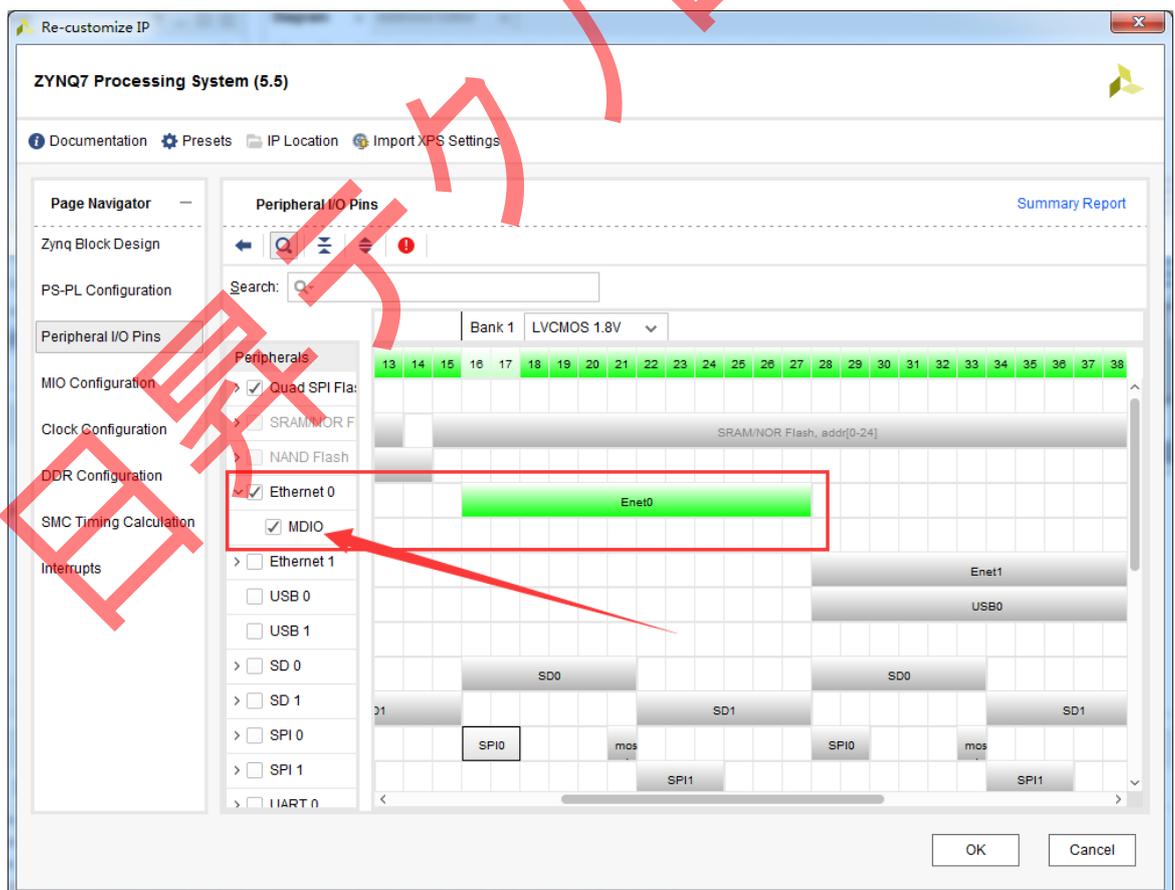
開発ボードに1重ギガイーサネットがあって、RGMII インタフェースで接続する。本実験は SDK 内部の LWIP テンプレートでギガイーサネット TCP 通信する方法を説明する。LWIP は軽量級プロトコルが、一度も使用したことがないなら使うときは少し難しい。こちらの意見は LWIP に関する知識を了解すべき。

### 10.1 Vivado プロジェクトを作成

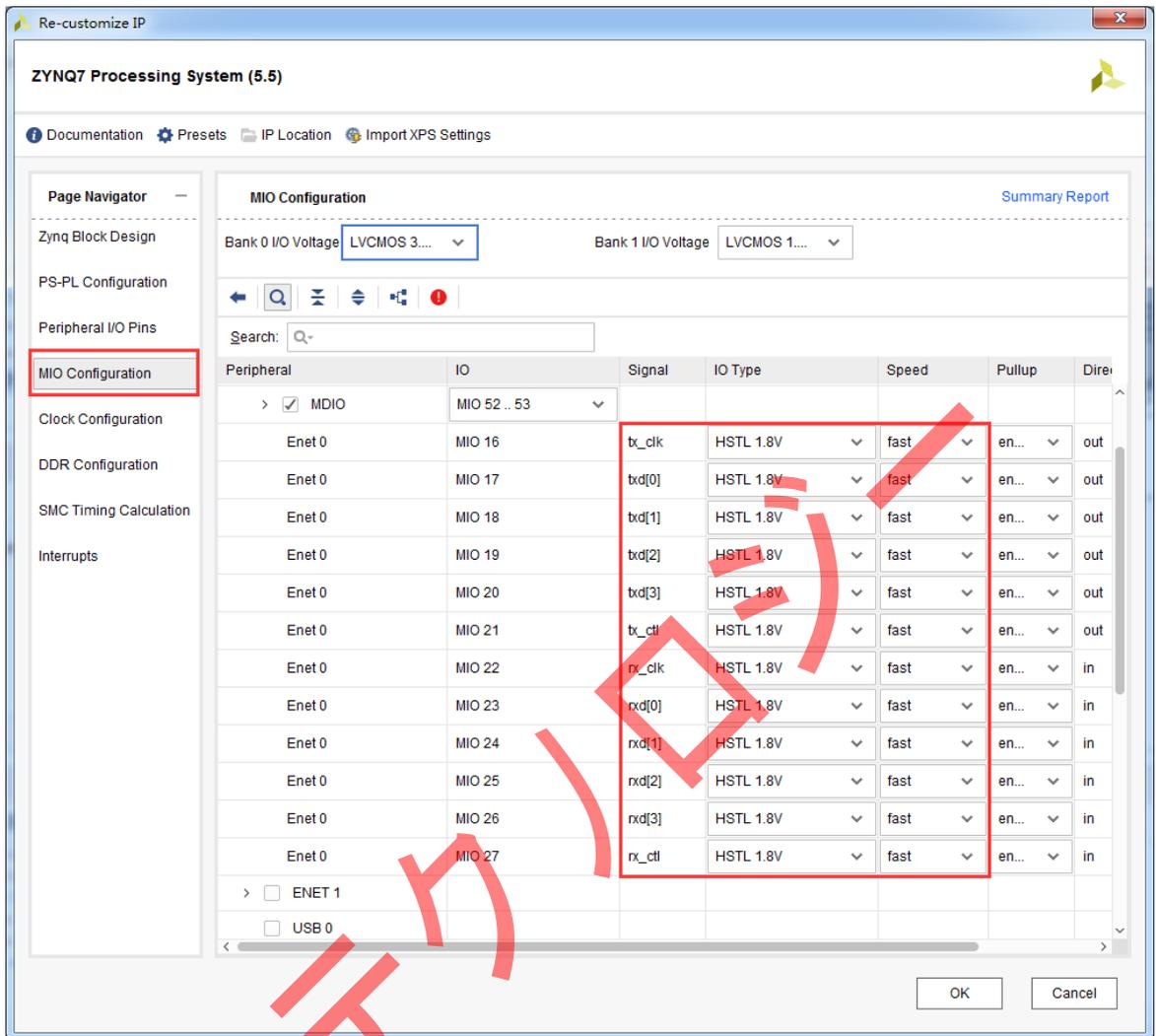
1) 新規 vivado プロジェクト “net\_test” を作成して、ZYNQ を追加する。前の教程にシリアルポールを配置する。詳しいパラメータはサンプル内部のプロジェクトを参考できる。

#### 10.1.1 PS 側のイーサネット配置

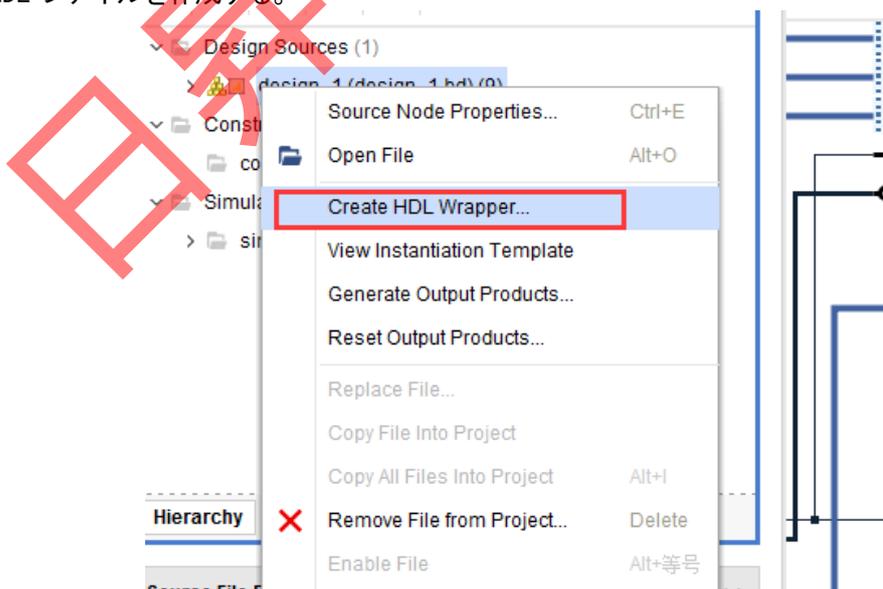
1) “Enet0” (MI016-MI027) と “MDIO” (MI052-MI053) を起動させる。



2) Enet0 のレベル標準を STL 1.8V にし、Speed を fast に変更する。これらのパラメータは非常に重要なので、変更しないと、ネットが繋がらない。



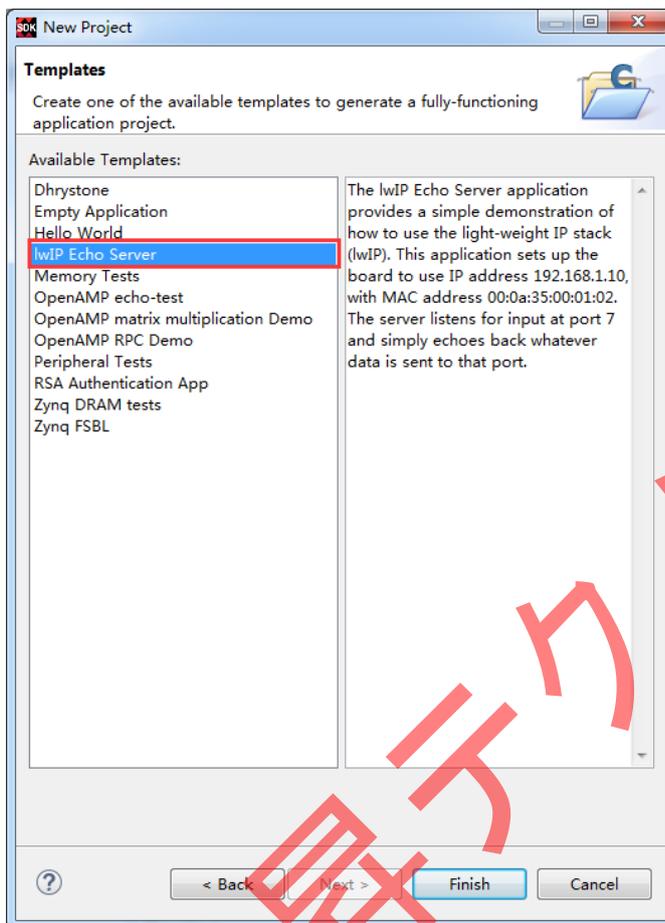
3) HDL ファイルを作成する。



- 2) bit ファイルを生成する。そして、ハードウェアインフォメーションをエクスポートする。SDK を起動する。

## 10.2 SDK プログラム

### 10.2.1 LWIP テンプレートに基づく APP を作成

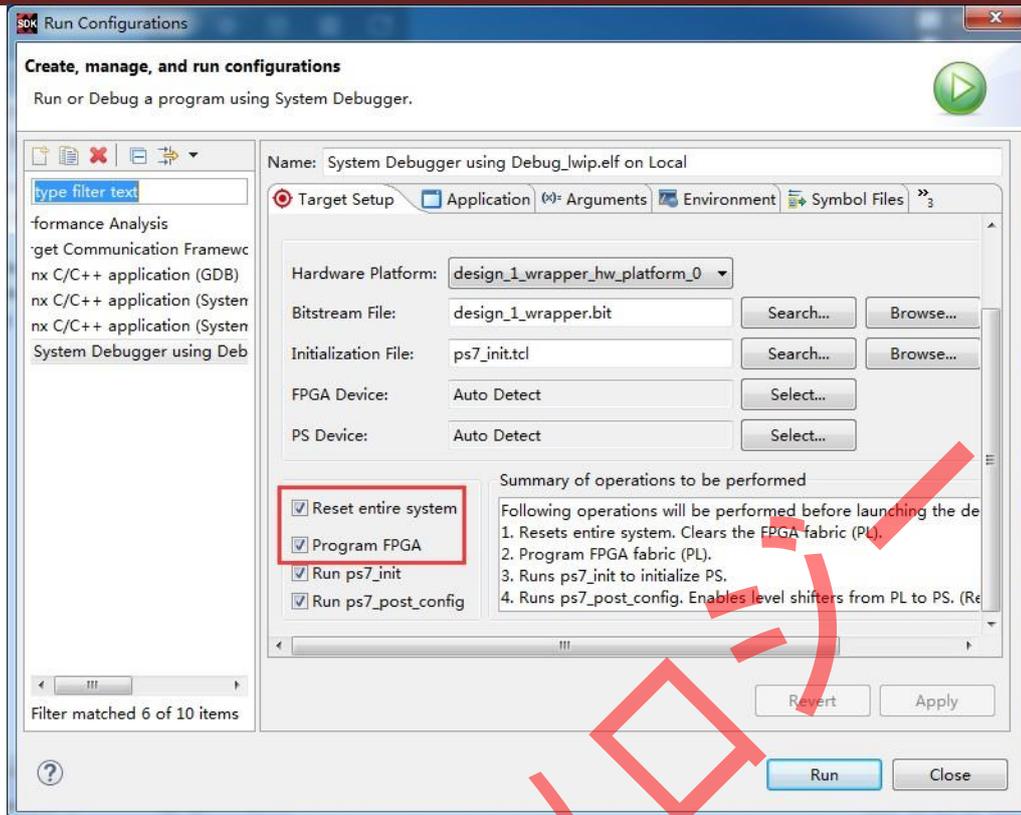


## 10.3 ダウンロードとデバッグ

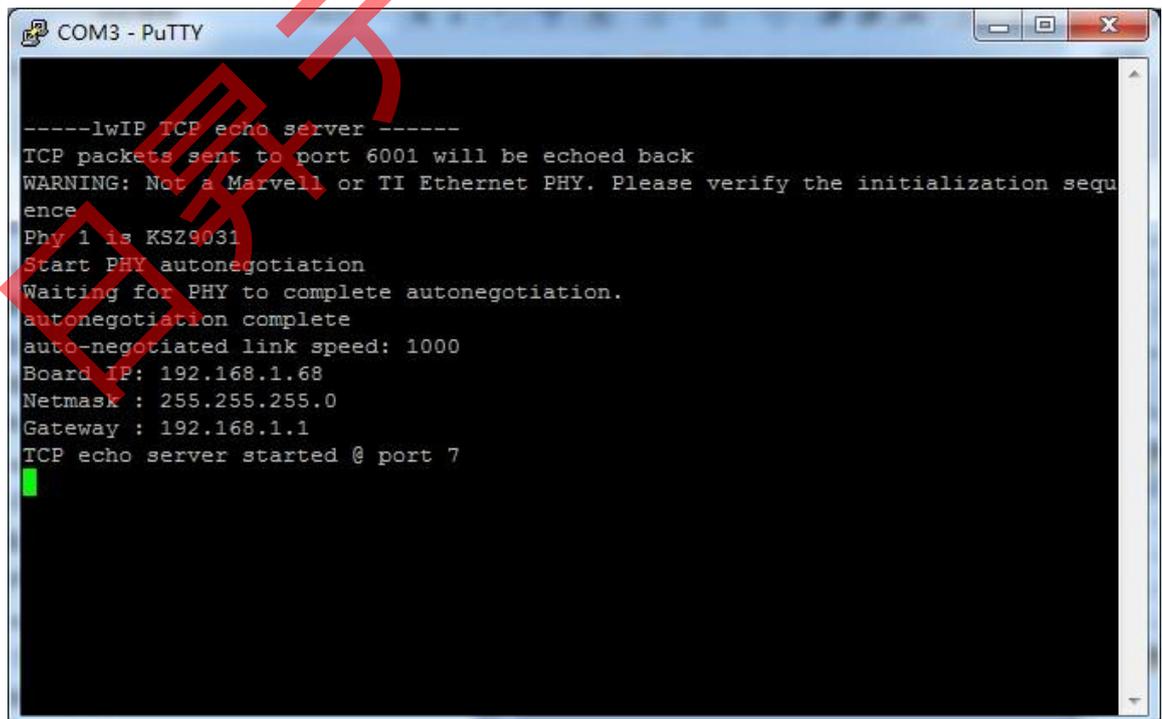
テスト中環境は dhcp をサポートするルーターが必要である。開発ボードはルーターと接続すると自動的に IP アドレスを獲得できる。実験中メインフレームと開発ボードは同じネットワークにいて、お互いに通信できる。

### 10.3.1 イーサネットテスト

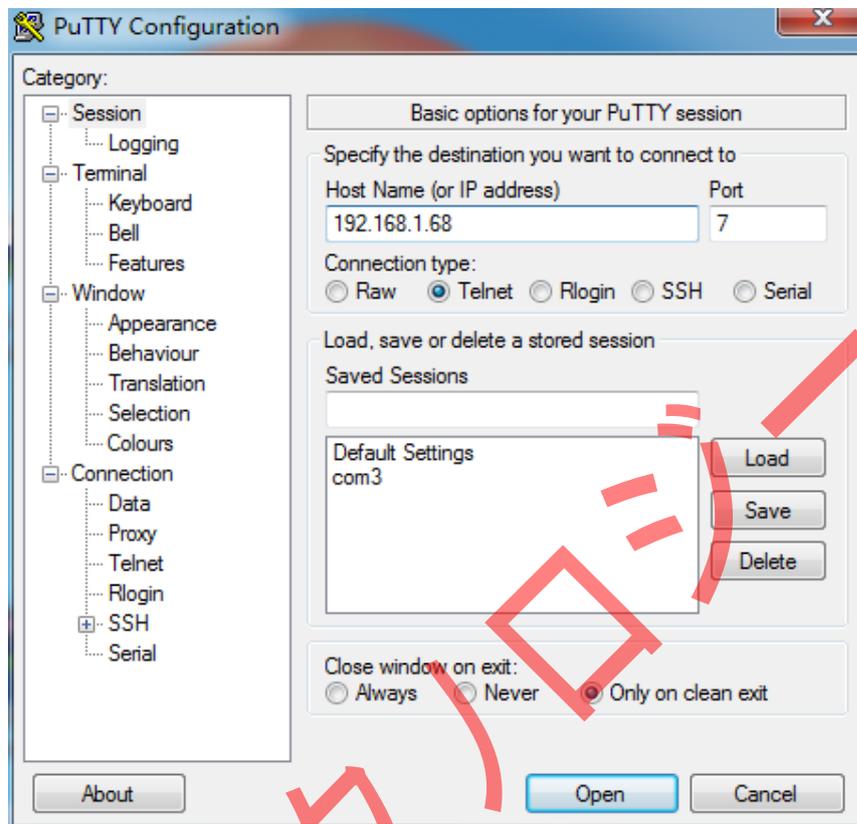
- 1) シリアルポールと接続して、シリアルでバグターミナルを開く。PS 側のイーサネットケーブルをルーターに接続して、SDK を起動する。



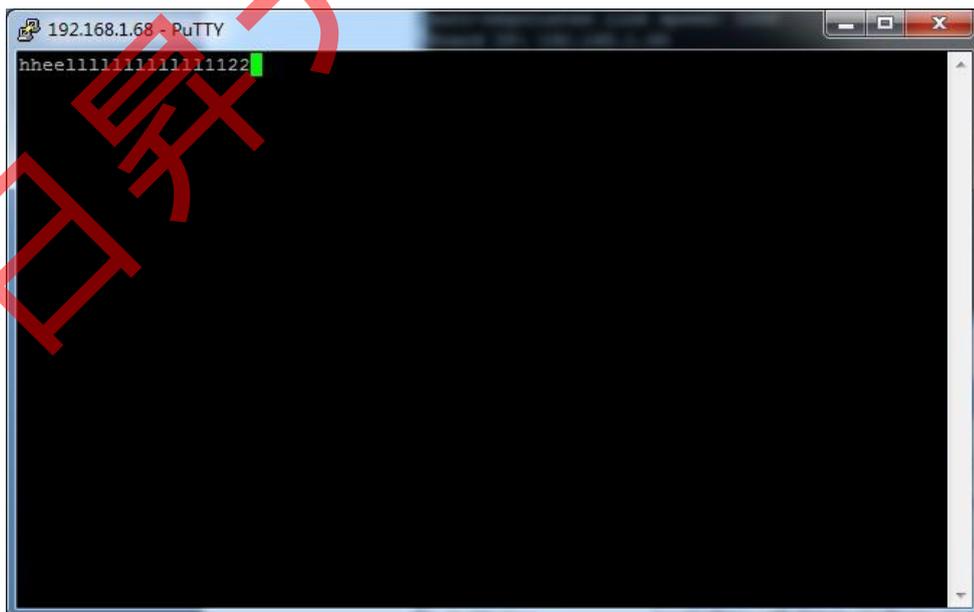
- 2) シリアルはインフォメーションをプリントアウトして、自動的に  
 “192.168.1.68” というアドレスを獲得した。接続速度は 1000Mbps で、tcp ポー  
 トは 7 である。



3) telnet を使って接続する。



4) 文字一つを書き込む時、開発ボードは同じ文字を表示する。



#### 10.4 実験のまとめ

実験をつうじて、SD プログラムの開発を深く理解できた。本実験はただ簡単に LWIP APP の作成を説明しただけである。LWIP は UDP、TCP 等のプロトコルを完成できる。続きの教程にこちらはイーサねに基づく具体的な APP を提供する。例えば、イーサネットを通じて、ADC が集めたデータを伝送することとか、カメラデータを上位機械に伝送することとか。

日昇テクノロジー

## 第十一章 ユーザー定義 IP テスト

実験用 Vivado プロジェクトは “custom\_pwm\_ip”。

オフィシャル Xilinx はたくさんの IP コアを提供した。Vivado の IP Catalog にこれらの IP コアを見ることができる。ユーザーは自分のシステムを構築する時、オフィシャル Xilinx のフリー IP コアだけを使うのができない。ユーザー自身の IP コアを作成するときが必要で、いい点もある。例えば、システムデザインをカスタマイズする；デザインを重複に使用できる。IP コアに license を追加して、有料で提供する；システムデザインをシミュレーションにして、デザイン時間も縮める。ZYNQ システムで IP コアをデザインするには、よく使われているのが AXI バスで PL と PL 部分の IP コアを接続することである。本実験は Vivado に AXI バスタイプの IP コアを構築する方法を紹介する。この IP コアは PWM を生成して、開発ボード上の LED をコントロールし、呼吸ライトの効果をつくる。

### 11.1 PWM 紹介

PWM はよく LED やブザーなどのコントロールに使われている。パルスのデューティ比を調節して LED の明るさを調節する。開発ボードに使われた pwm モジュールは下のよう：

```
////////////////////////////////////  
//  
//  
// Author: meisq  
// msq@qq.com  
// ALINX(shanghai) Technology Co.,Ltd  
// heijin  
// WEB: http://www.alinx.cn/  
// BBS: http://www.heijin.org/  
//  
////////////////////////////////////  
//  
// Copyright (c) 2017, ALINX(shanghai) Technology Co.,Ltd //  
// All rights reserved //  
// This source file may be used and distributed without restriction provided  
// that this copyright statement is not removed from the file and that any //  
// derivative work contains the original copyright notice and the associated  
// disclaimer. //  
//  
////////////////////////////////////  
//  
//=====  
// Description: pwm model  
// pwm out period = frequency(pwm_out) * (2 ** N) /  
frequency(clk);  
//=====  
// Revision History:  
// Date By Revision Change Description  
//-----  
// 2017/5/3 meisq 1.0Original  
//*****
```



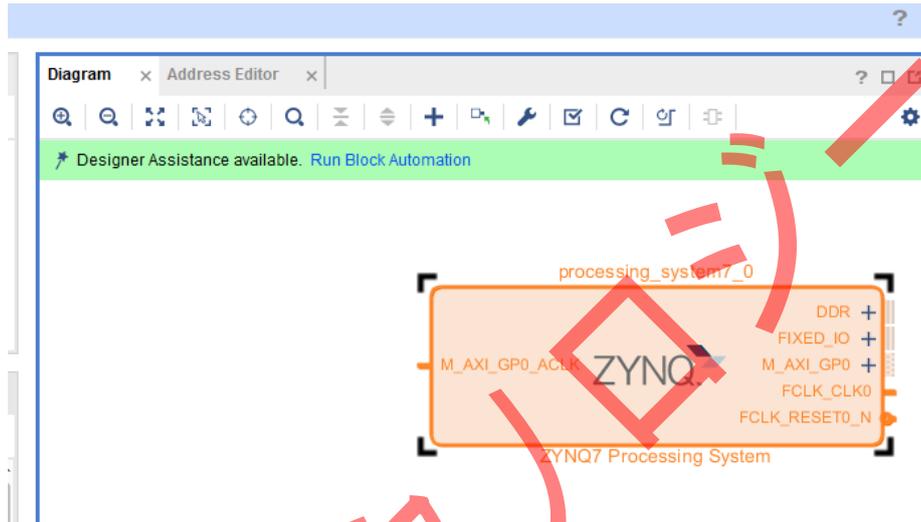
```
    **/  
timescale 1ns / 1ps  
module ax_pwm  
#(  
    parameter N = 32 //pwm bit width  
)  
(  
    input          clk,  
    input          rst,  
    input[N - 1:0]period,  
    input[N - 1:0]duty, output  
                    pwm_out  
);  
  
    reg[N - 1:0] period_r;  
    reg[N - 1:0] duty_r;  
    reg[N - 1:0] period_cnt;  
    reg pwm_r;  
    assign pwm_out = pwm_r;  
    always@(posedge clk or posedge rst)  
begin  
    if(rst==1)  
        begin  
            period_r <= { N {1'b0} };  
            duty_r <= { N {1'b0} };  
        end  
    else begin  
        period_r <= period;  
        duty_r <= duty;  
    end  
    end  
    always@(posedge clk or posedge  
rst) begin  
        if(rst==1)  
            period_cnt <= { N {1'b0} };  
        else  
            period_cnt <= period_cnt + period_r;  
        end  
    always@(posedge clk or posedge rst)  
begin  
        if(rst==1)  
            begin  
                pwm_r <= 1'b0;  
            end  
        else  
            begin  
                if(period_cnt >= duty_r)  
                    pwm_r <= 1'b1;  
                else  
                    pwm_r <= 1'b0;  
            end  
        end  
    end  
endmodule
```

この PWM モジュールは “period”、“duty” という二つのパラメータを利用して、頻度とデューティ比をコントロールする。レジューをデザインしてこれらのパラメータをコントロールする。ここは AXI バスを使用して、レジスターのリードライトをする。

## 11.2 Vivado プロジェクトの作成

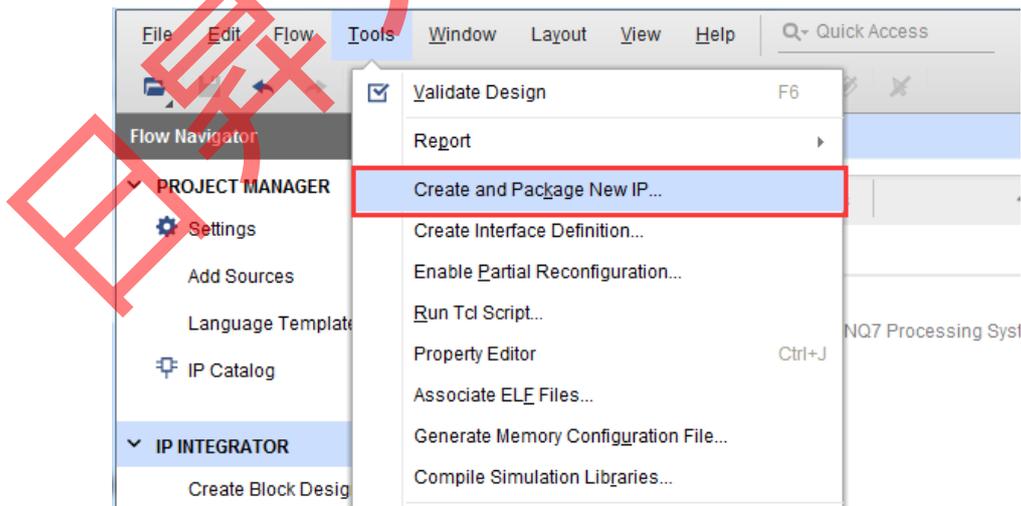
### 11.2.1 一つの vivado プロジェクトを作成

“custom\_pwm\_ip” という名のプロジェクトを作成する。zynqPS システムを追加して、パラメータを配置する。具体的な方法は前の章に参考できる。

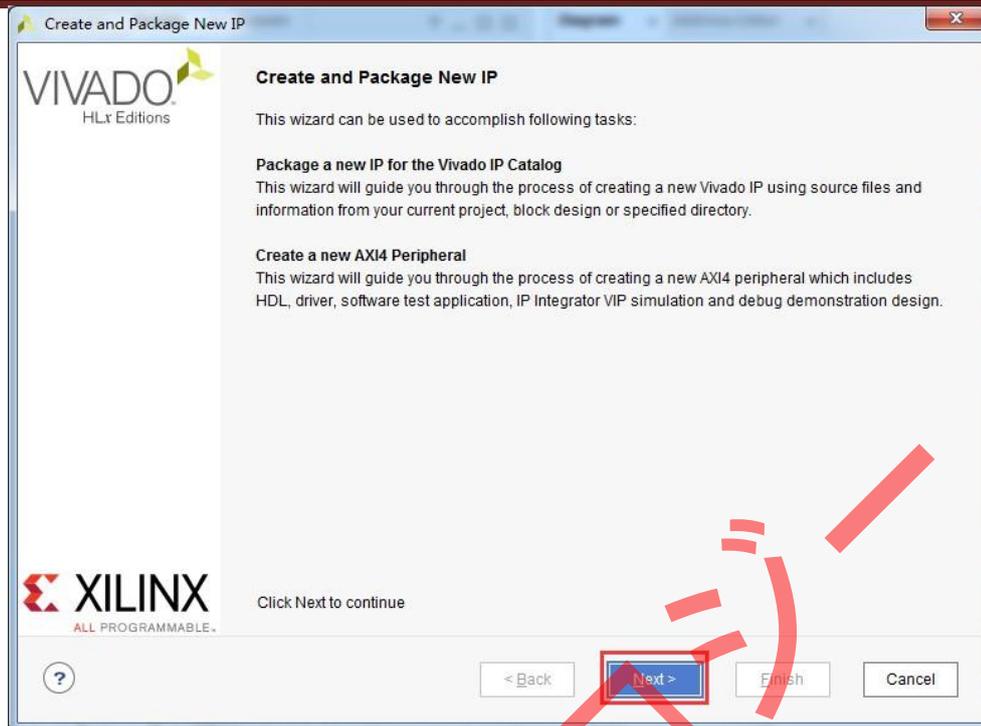


### 11.2.2 ユーザー定義 IP を作成する。

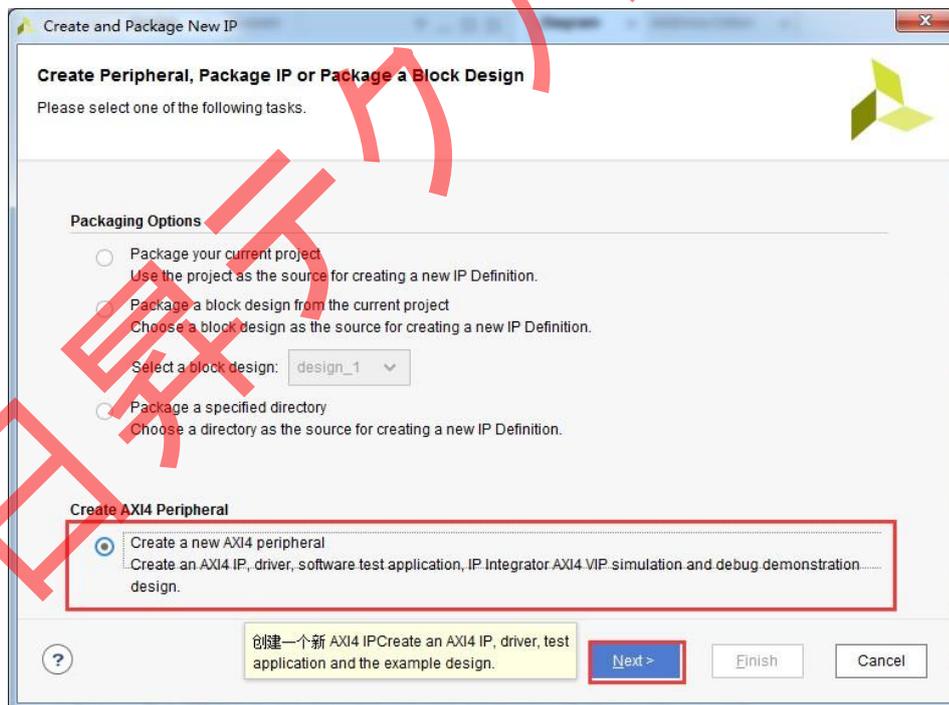
- 1) メニューの “Tools->Create and Package IP...” をクリックする。



- 2) “Next を” を選択する。



3) 新規 AXI4 デバイスを作成する。



4) ネームは“ax\_pwm”にして、デスクリプションは“alinx pwm”にする。そして、適切な位置を選んで IP を入れる。

**Create and Package New IP**

**Peripheral Details**  
Specify name, version and description for the new peripheral

Name:

Version:

Display name:

Description:

IP location:

Overwrite existing

< Back **Next >** Finish Cancel

5) 下のパラメータはインタフェースタイプやレジスタの数とかを指定できる。ここは変更せずに、AXI Lite Slave インタフェースとレジスタ4つを使う。

**Create and Package New IP**

**Add Interfaces**  
Add AXI4 interfaces supported by your peripheral

Enable Interrupt Support

Interfaces: S00\_AXI

ax\_pwm\_v1.0

Name: S00\_AXI

Interface Type:

Interface Mode:

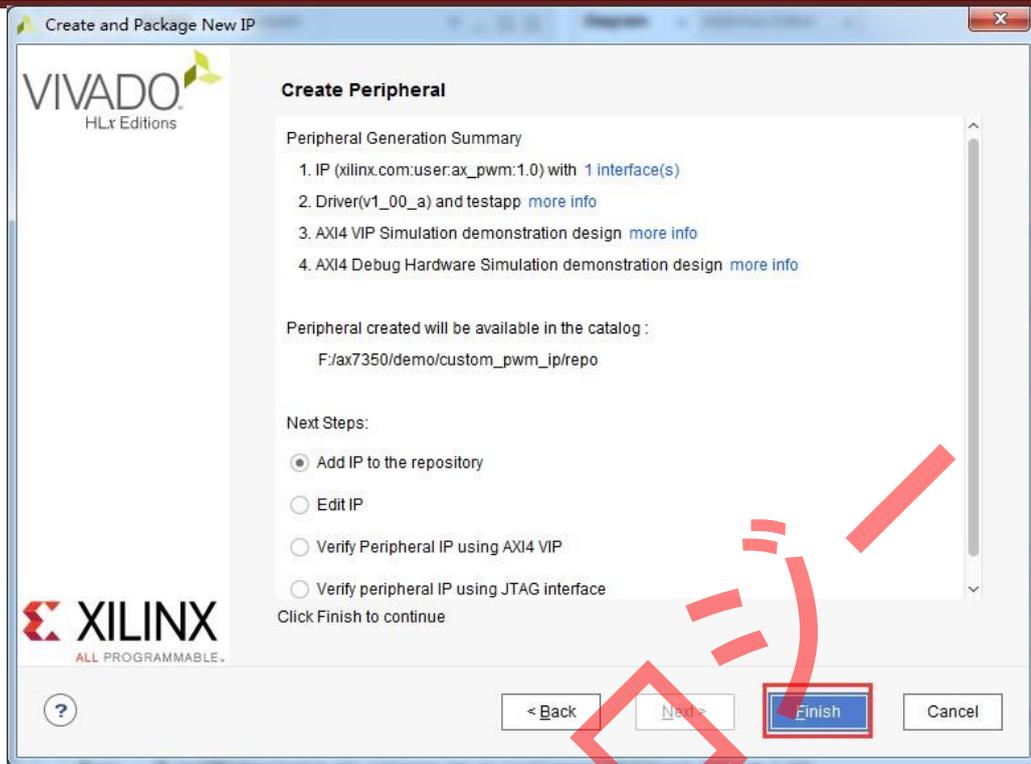
Data Width (Bits):

Memory Size (Bytes):

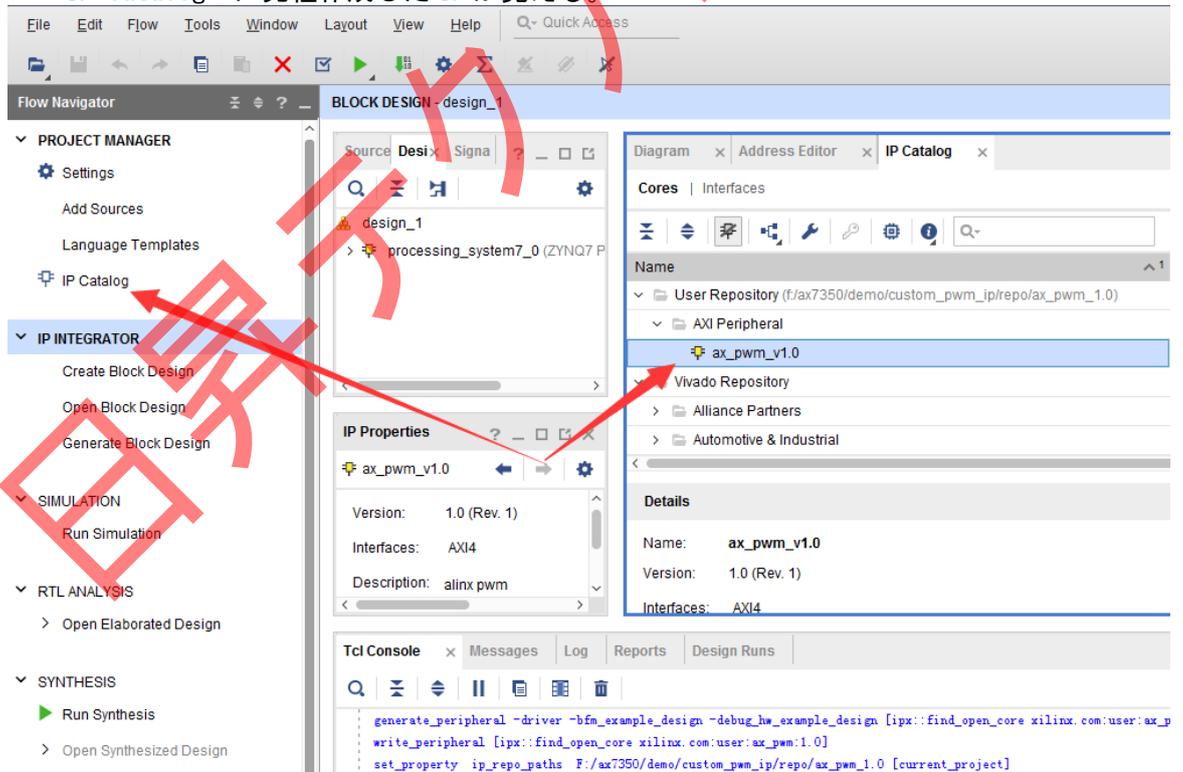
Number of Registers:  [4..512]

< Back **Next >** Finish Cancel

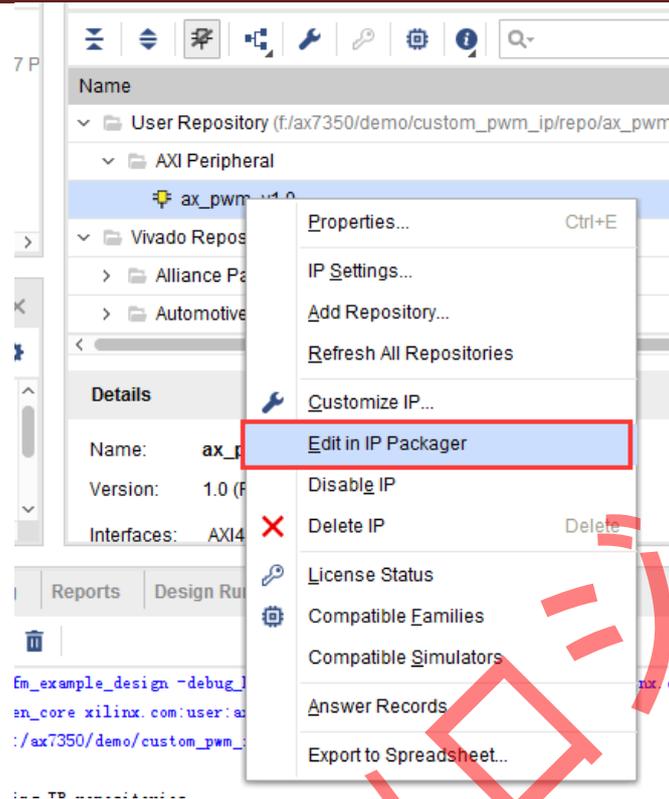
6) “Finish” をクリックして、IP の作成を完成する。



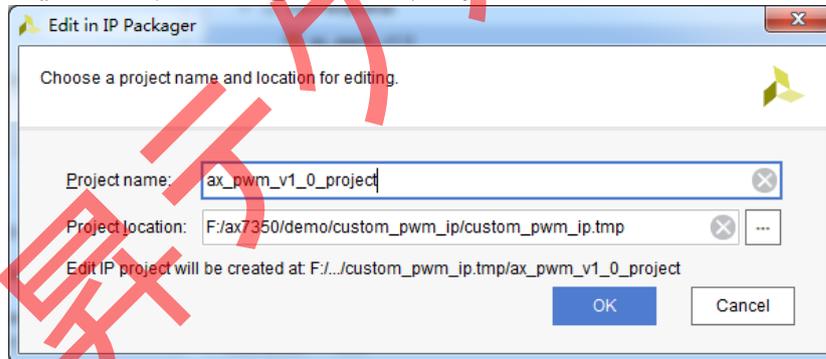
7) “IP Catalog” に先程作成した IP が見える。



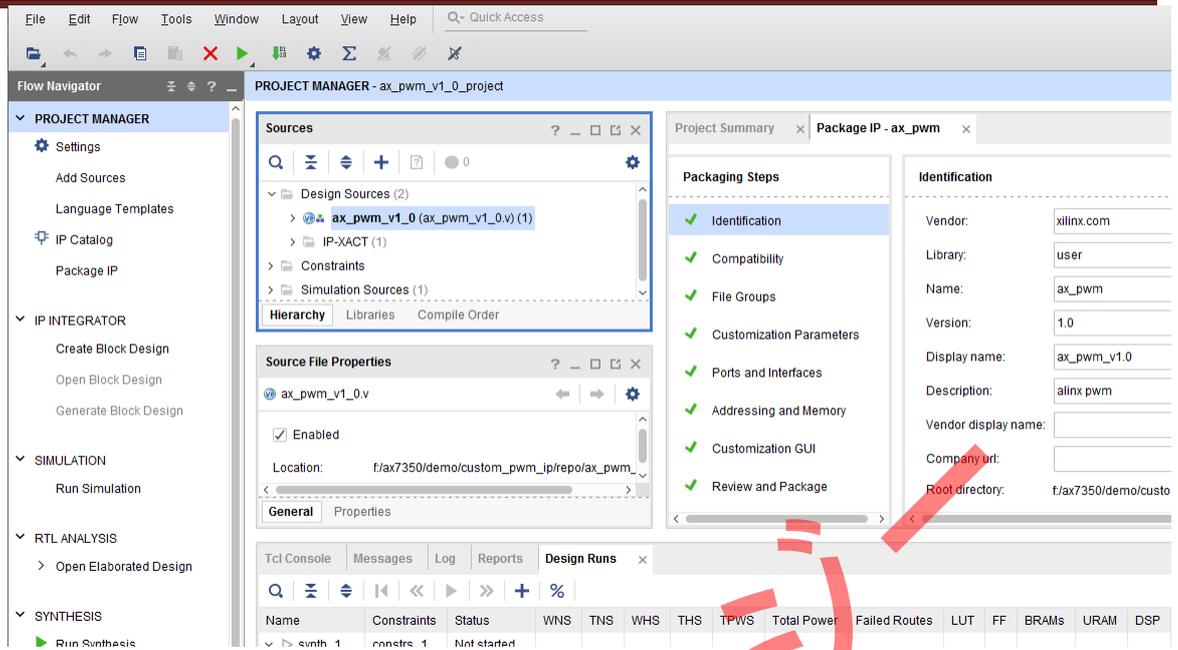
8) この時の IP は簡単なレジスターをリードライトする機能しかないので、IP を変更する必要がある。IP を選択して、右ボタンで “Edit in IP Packager” を探し出す。



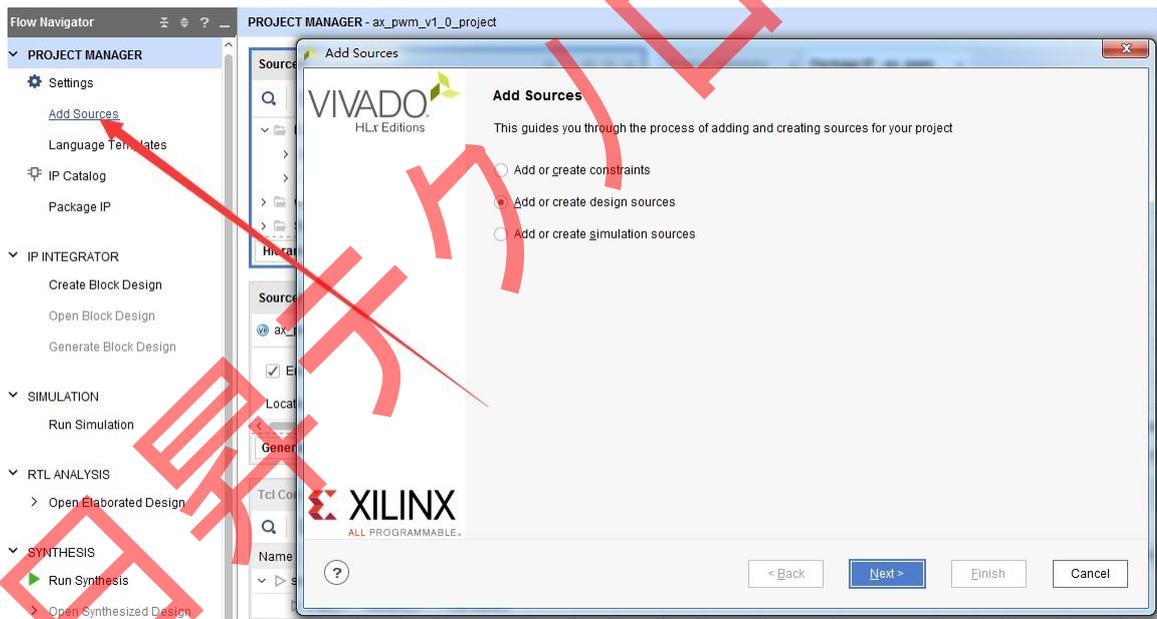
- 9) ダイアログがポップアップされて、プロジェクト名とパスを書き込める。ここはデフォルト値にして、“OK”をクリックする。



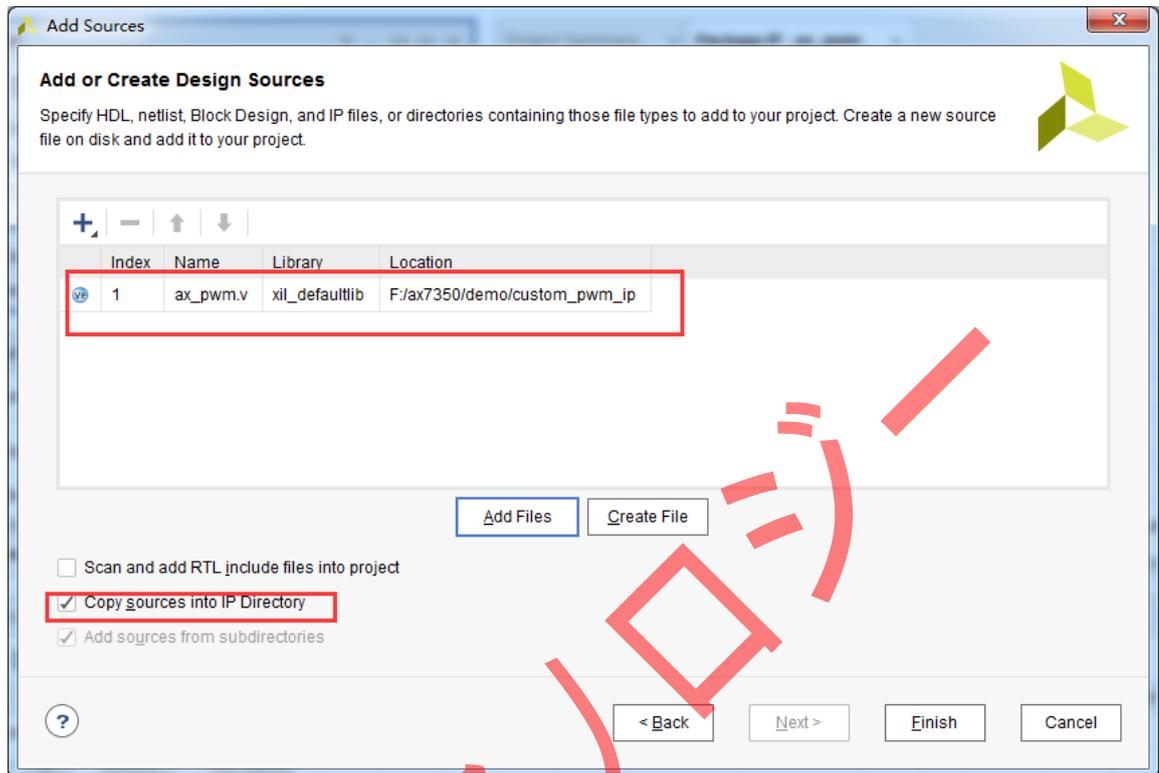
- 10) Vivadoは新しいプロジェクトを開いた。



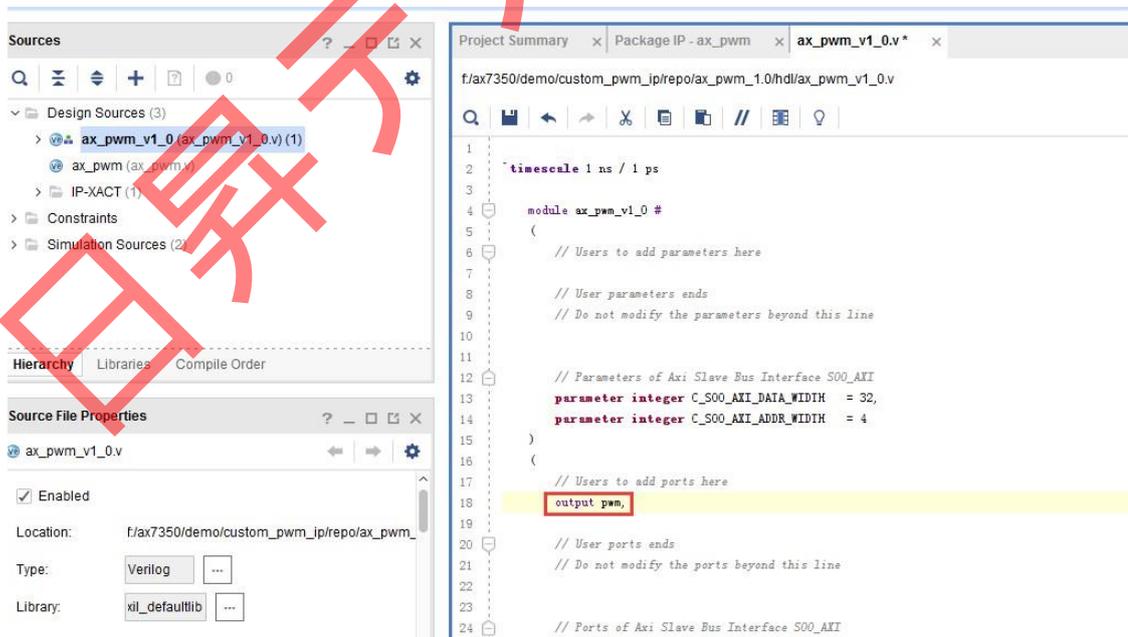
11) PWM 機能のコアコードを追加する。



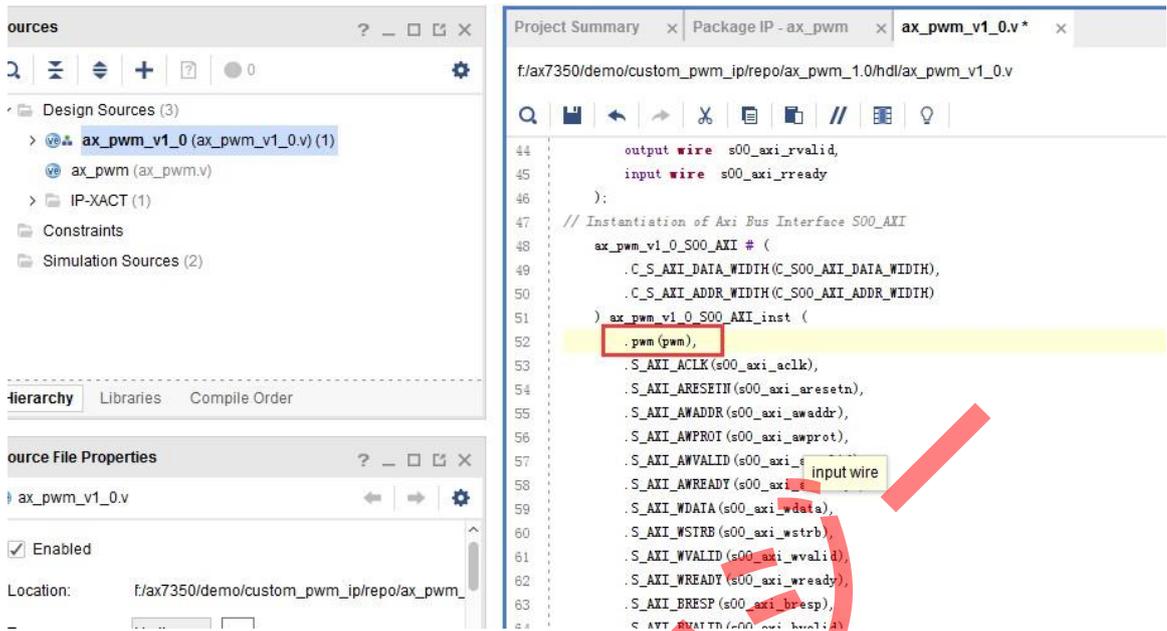
- 12) コード追加する時、コードを IP 目次ぐにコピーする。



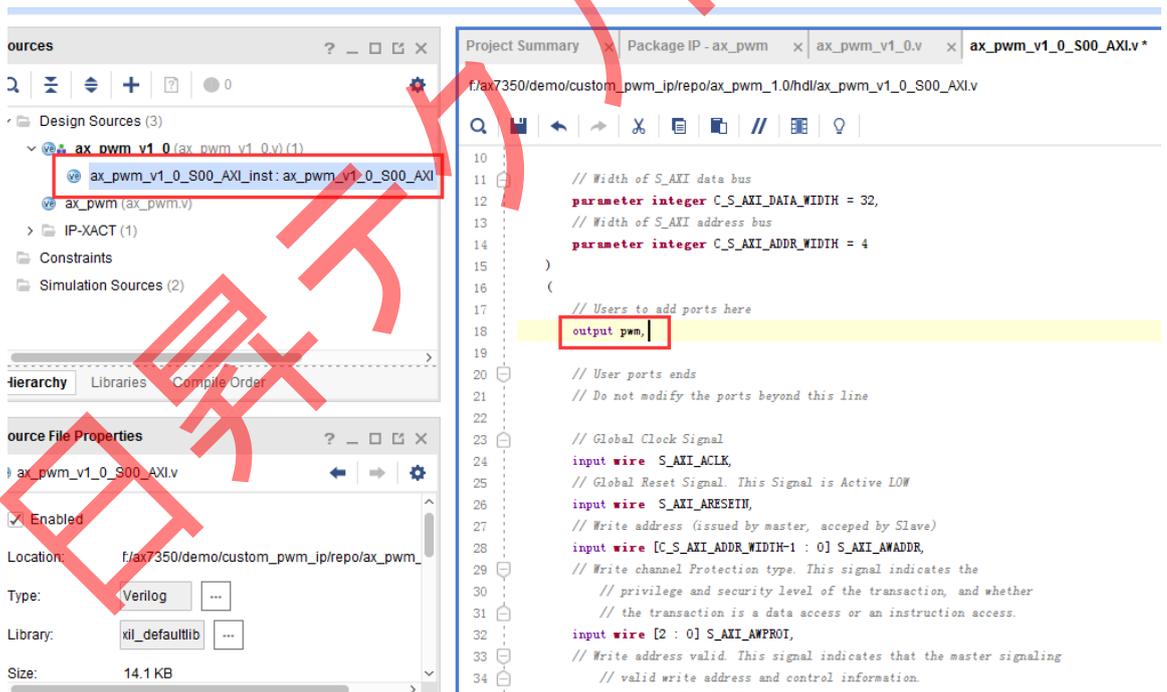
- 13) “ax\_pwm\_v1\_0.v” を変更し、pwm アウトポートを追加する。



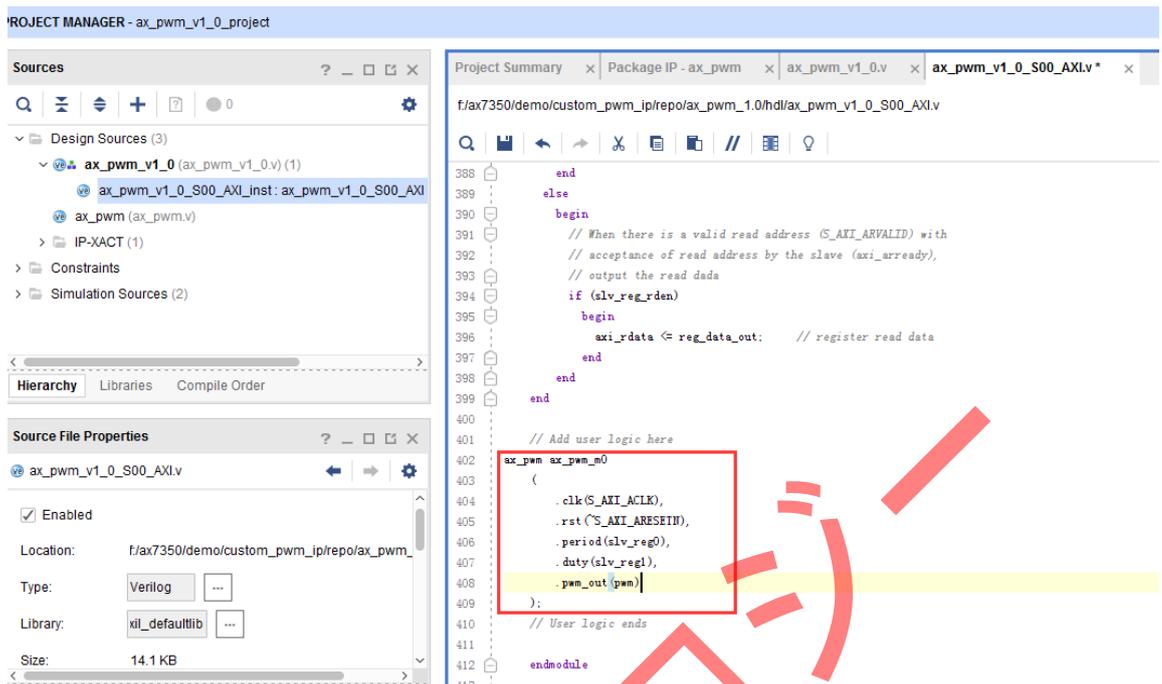
- 14) “ax\_pwm\_v1\_0.v” を変更し、インスタンス化 “ax\_pwm\_V1\_0\_S00\_AXI” に pwm ポートのインスタンス化を追加する。



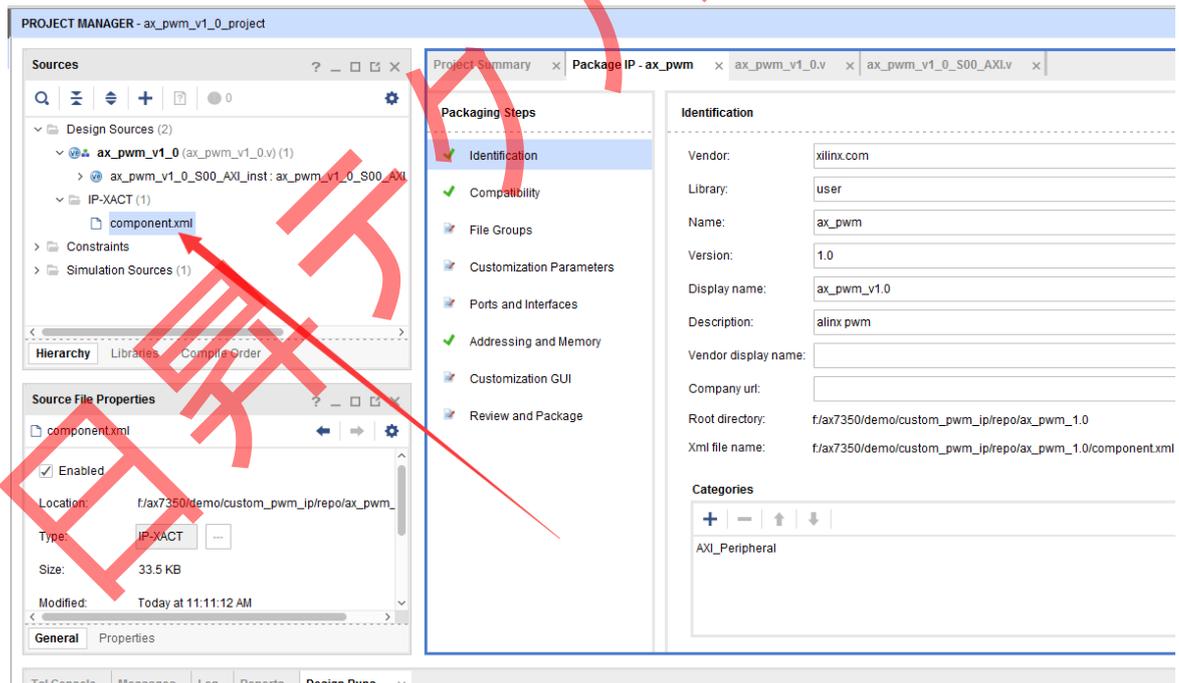
15) “ax\_pwm\_v1\_0\_s00\_AXI.v” ファイルを変更して、pwm ポートを追加する。このファイルは AXI4 Lite Slave を実現するコアコードである。



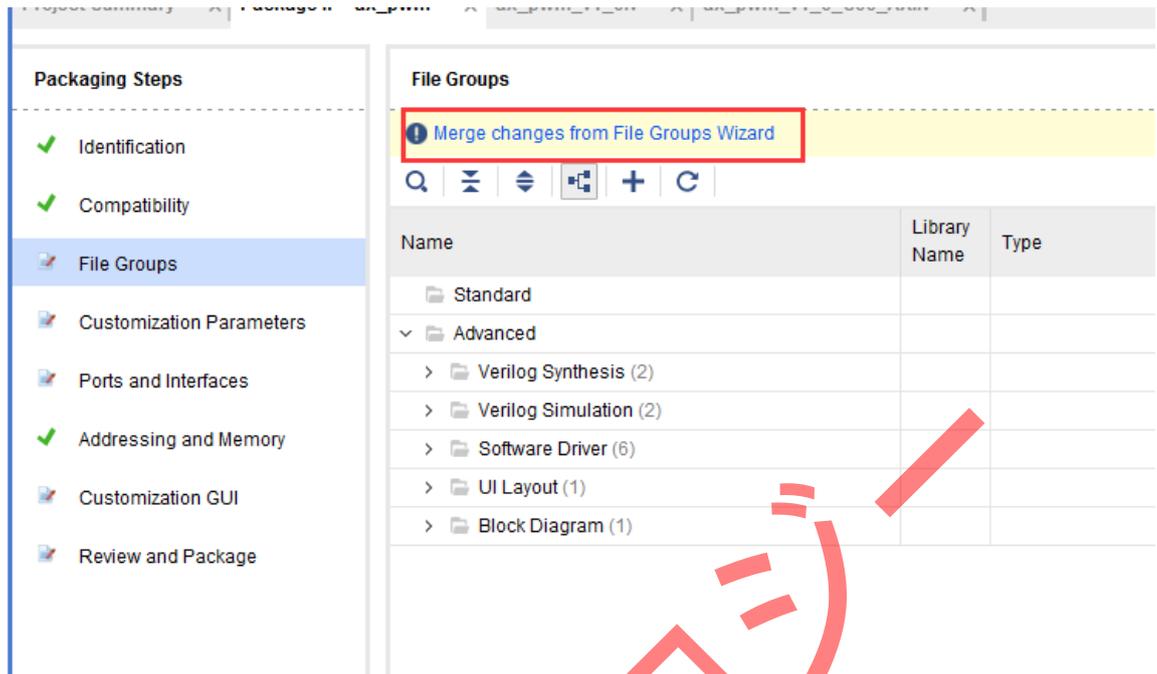
16) 修改 “ax\_pwm\_v1\_0\_s00\_AXI.v” ファイルを変更して、pwm 機能コアコードをインスタンス化する。レジスタ slv\_reg0 と slv\_reg1 を pwm モジュールパラメータのコントロールに用いる。



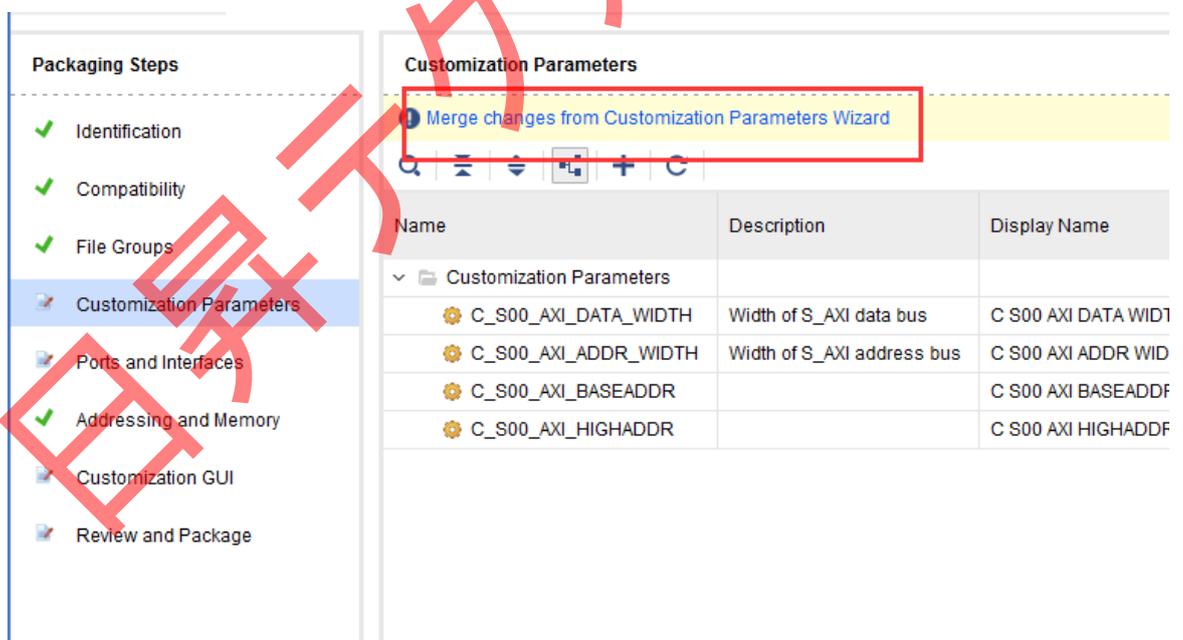
17) “component.xml” ファイルをダブルクリックする。



18) “File Groups” オプションに “Merge changers from File Groups Wizard” をクリックする。



19) “Customization Parameters” オプションに “Merge changes form Customization Parameters Wizard” をクリックする。

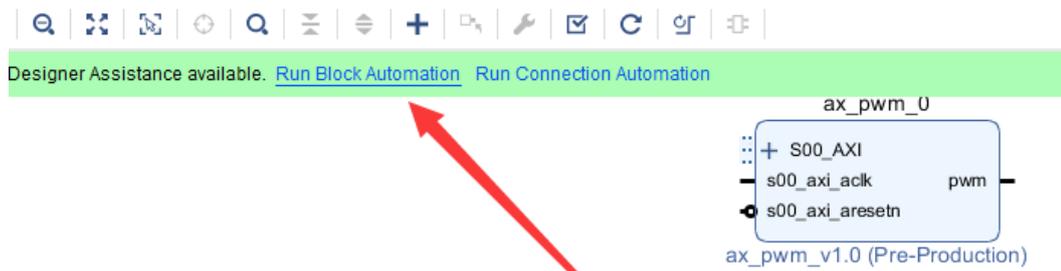


20) “Re-Package IP” をクリックして、IP の変更を完成する。

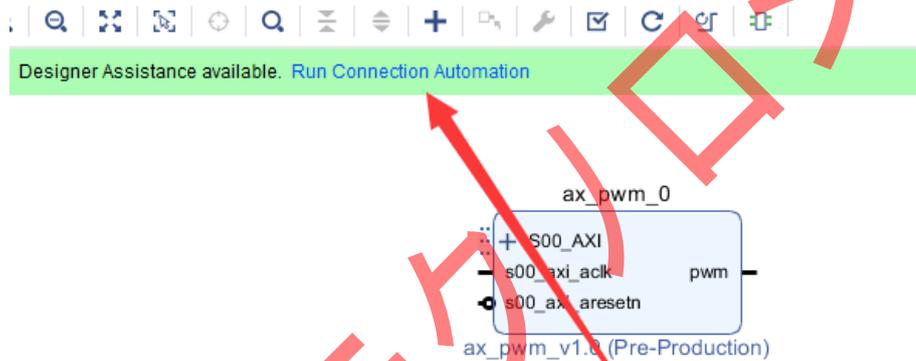
### 11.2.3 ユーザー定義 IP をプロジェクトに追加する

1) “pwm” をサーチし、“ax\_pwm\_v1.0” を追加する。

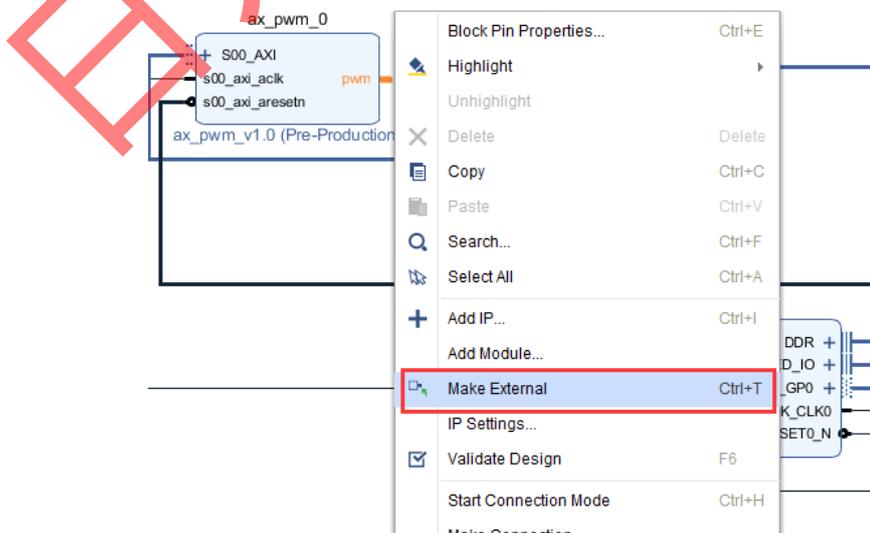
2) “Run Block Automation” をクリックする。

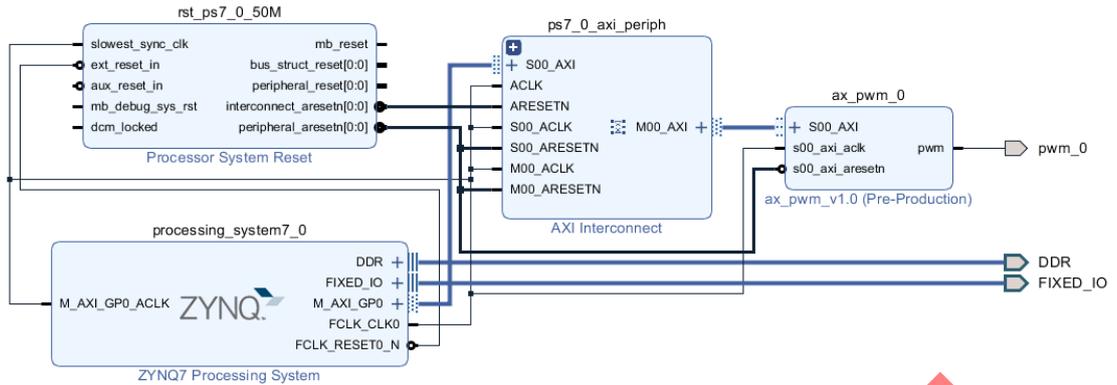


3) “Run Connection Automation” をクリックする。

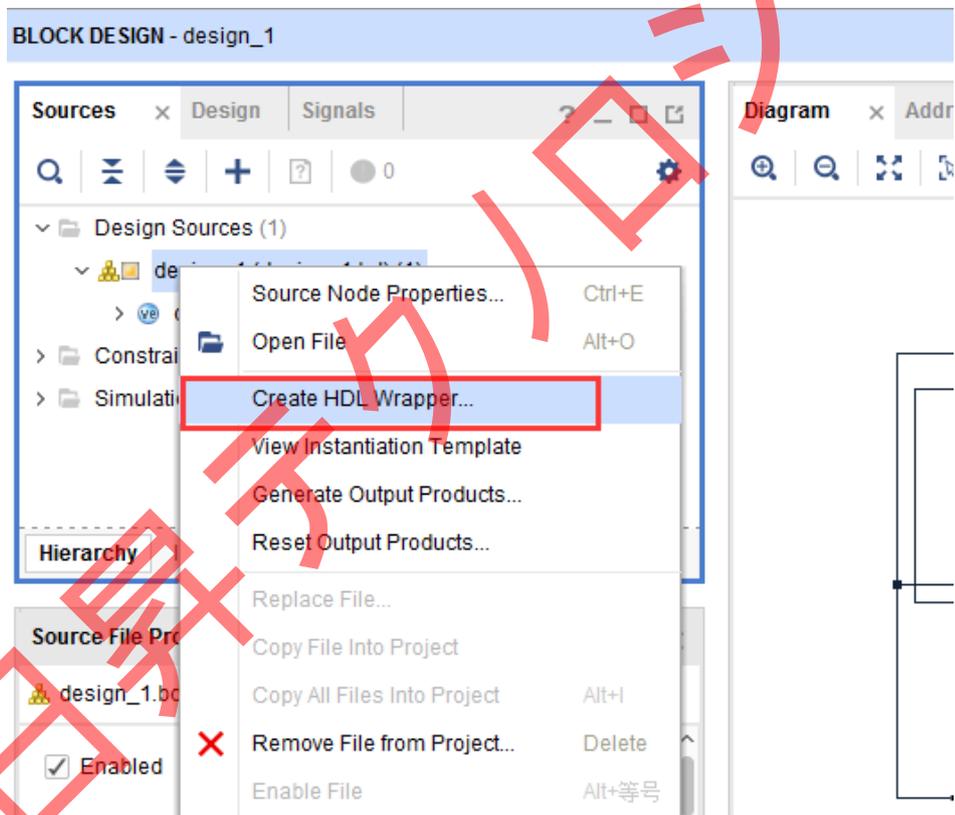


4) pwm ポートをエクスポート。



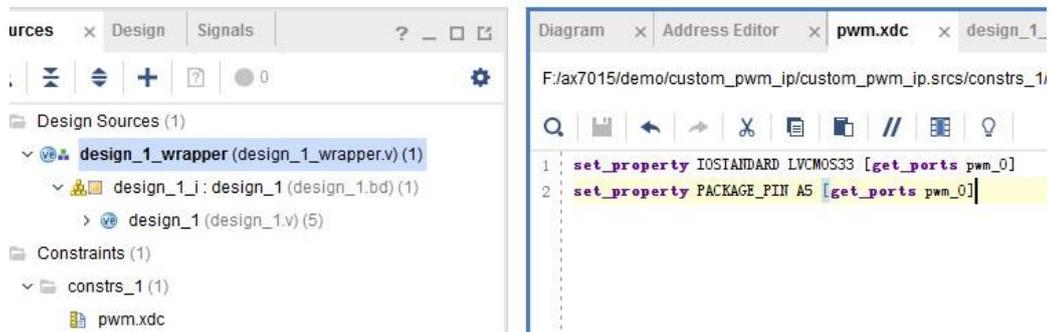


5) HDL ファイルを作成する。



6) xdc ファイルを追加して、P ピンを分配する。 pwm\_0 アウトポートポートを PLLED1 に配って、呼吸ライトを作る。

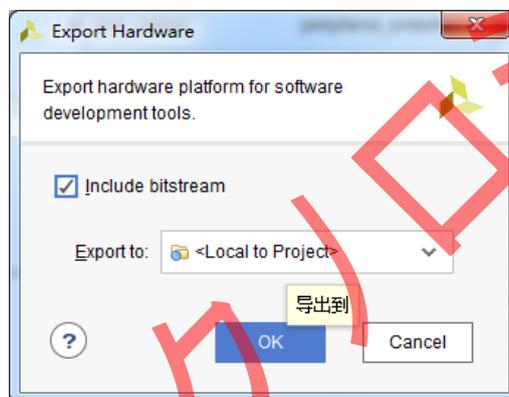
```
set_property IOSTANDARD LVCMOS33 [get_ports
pwm_0] set_property PACKAGE_PIN M14 [get_ports
```



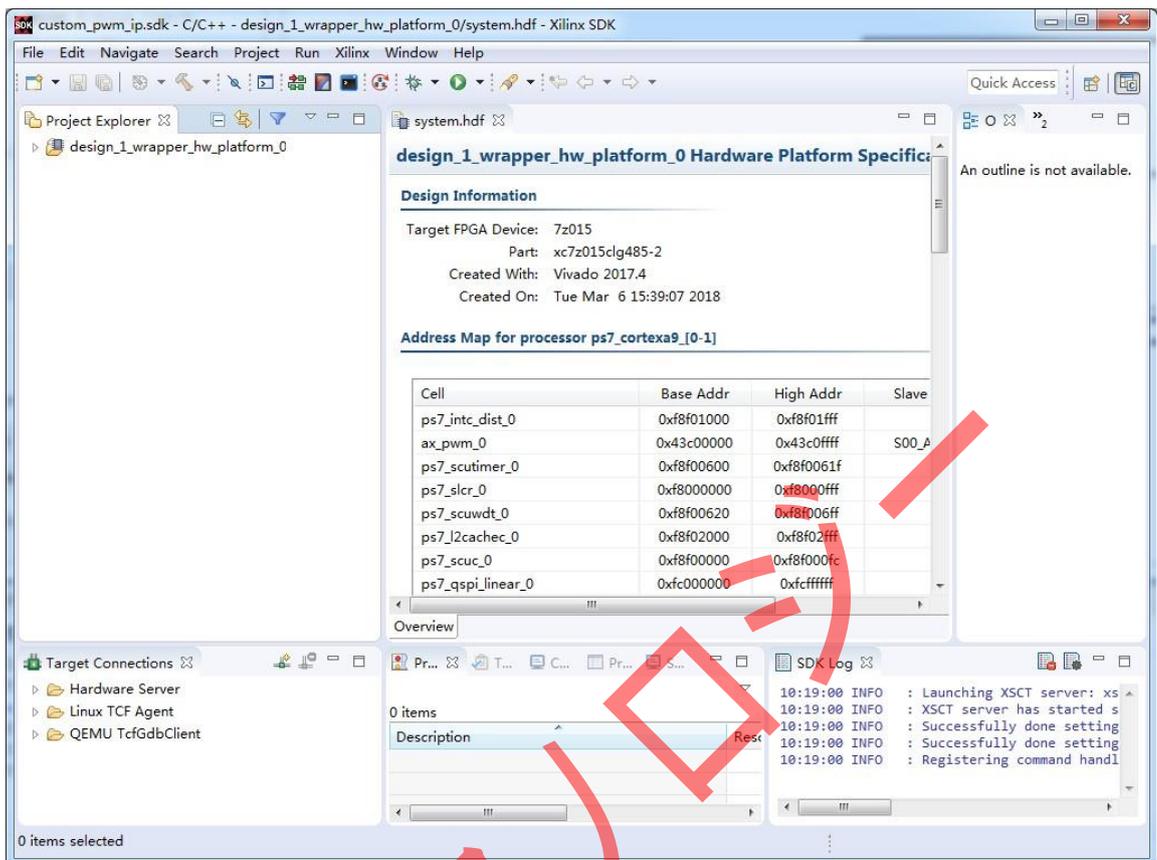
7) bit ファイルを生成する。

### 11.3 SDK ソフトのプログラミングとデバッグ

1) ハードウェアをエクスポートする。

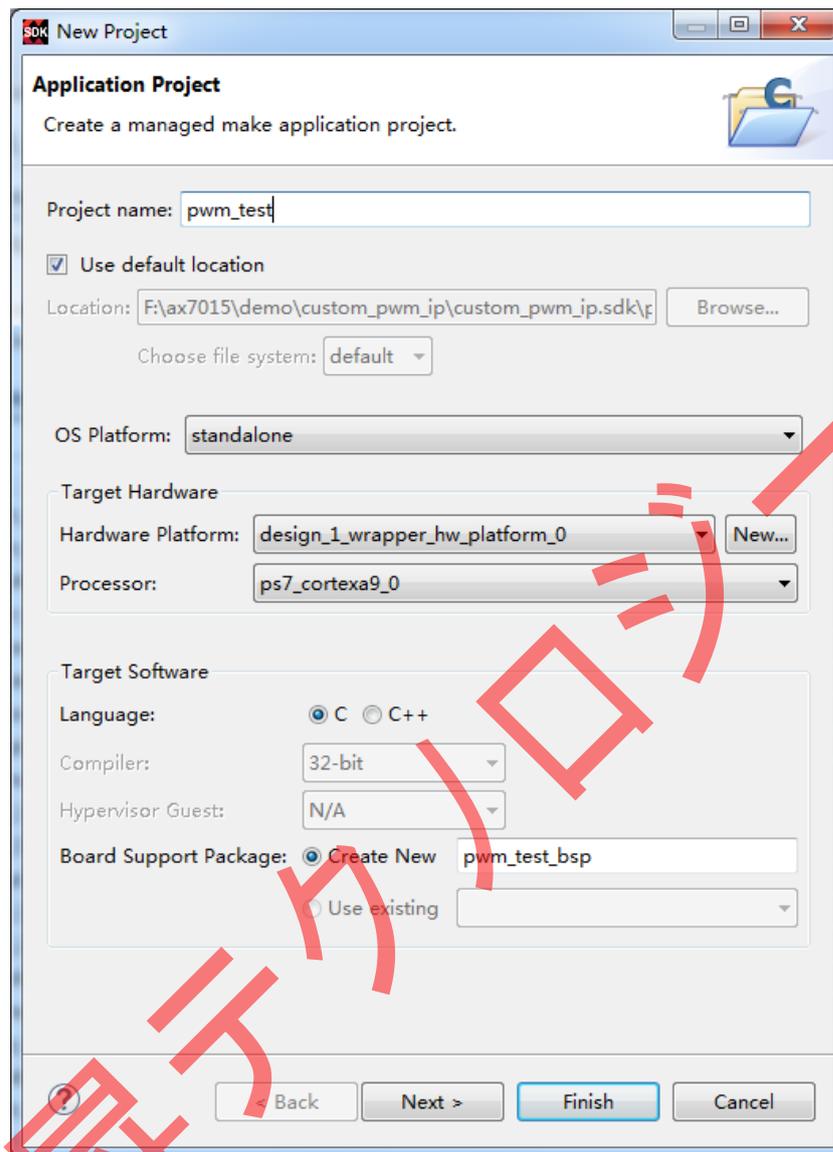


2) SDK 起動する。

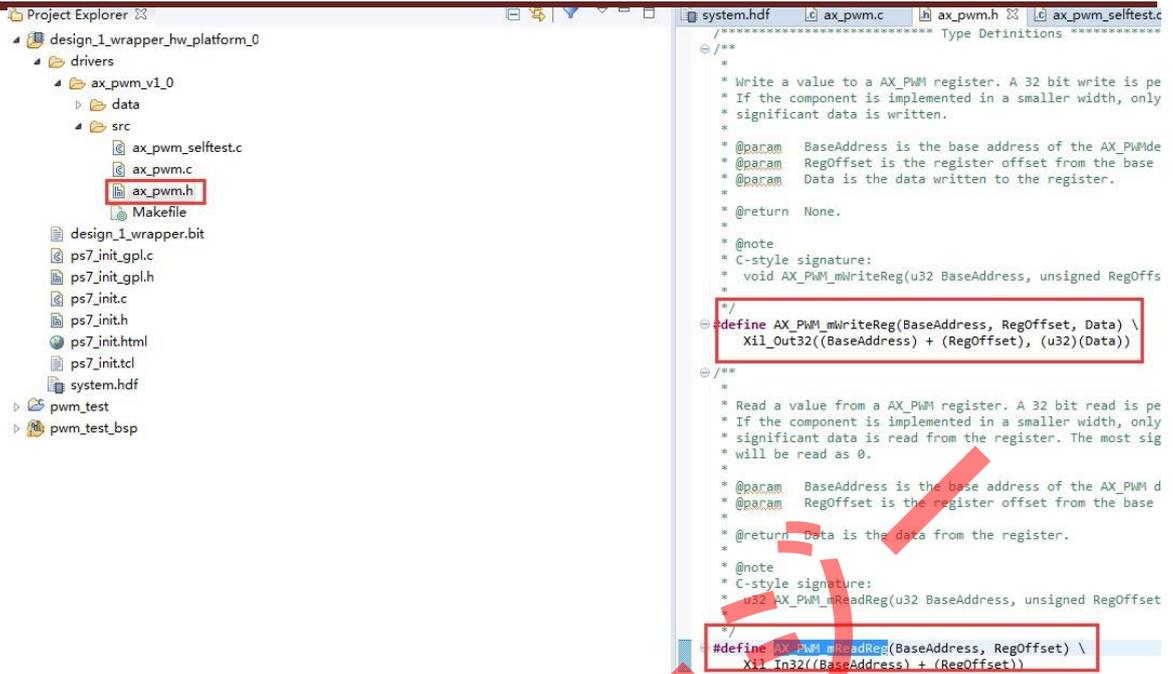


- 3) 新規 APP を作る。テンプレートは“Hello World”を選ぶ。

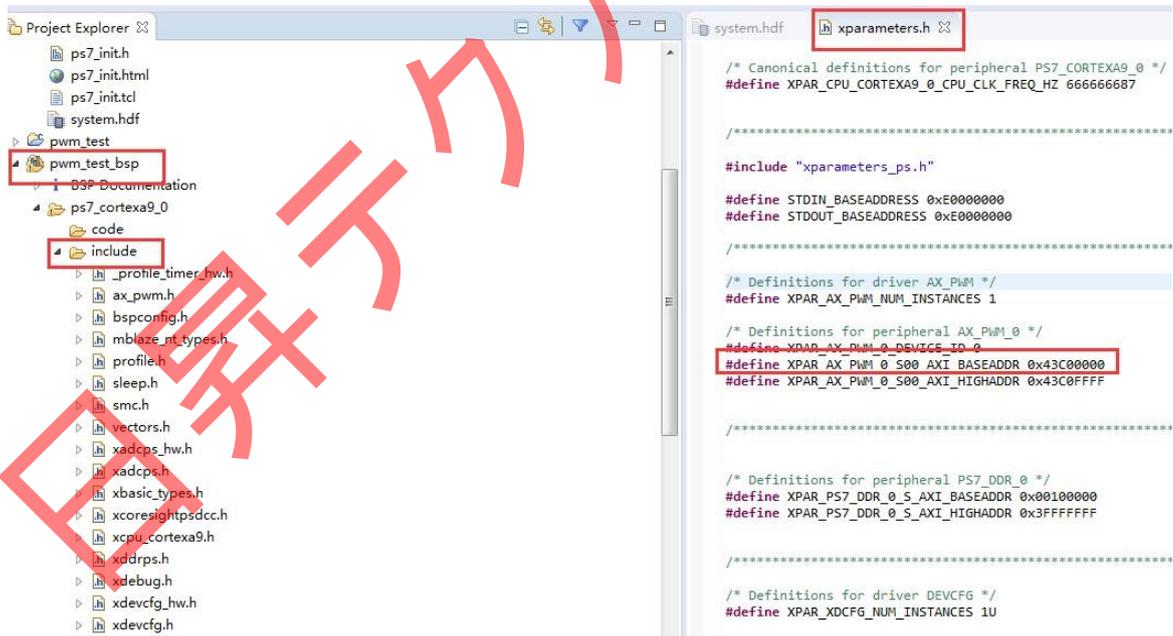
日昇テクノロジー



- 4) 前の例はすべて xilinx の IP を使っている。xilinx だいたい API を一セットで提供している、ユーザー定義 IP の場合、自分で開発するには必要がある。APP の目次ぐにあるいは資源を見て、ax\_pwm.h というファイルが見つかる。このファイルにユーザー定義U0レジスタのリードライトマクロ定義を含んでいる。



- 5) bsp に“xparameters.h”ファイルを見つける。これは非常に大事なファイルで、ユーザー定義 IP とこの IP にあるレジスタのベースアドレスが見つかる。



- 6) レジスタリードライトマクロとユーザー定義 IP のアドレスがあって、コーディングをはじめます。ユーザー定義 IP をテストするには、レジスタ AX\_PWM\_S00\_AXI\_SLV\_REG0\_OFFSET をライトすることで、PWM のアウトプット頻度をコントロールする。そして、レジスタ AX\_PWM\_S00\_AXI\_SLV\_REG1\_OFFSET をライトすることで、PWM アウトプットのデューティリ比をコントロールする。

```
unsigned int duty;

int main()
{
    init_platform();

    print("Hello

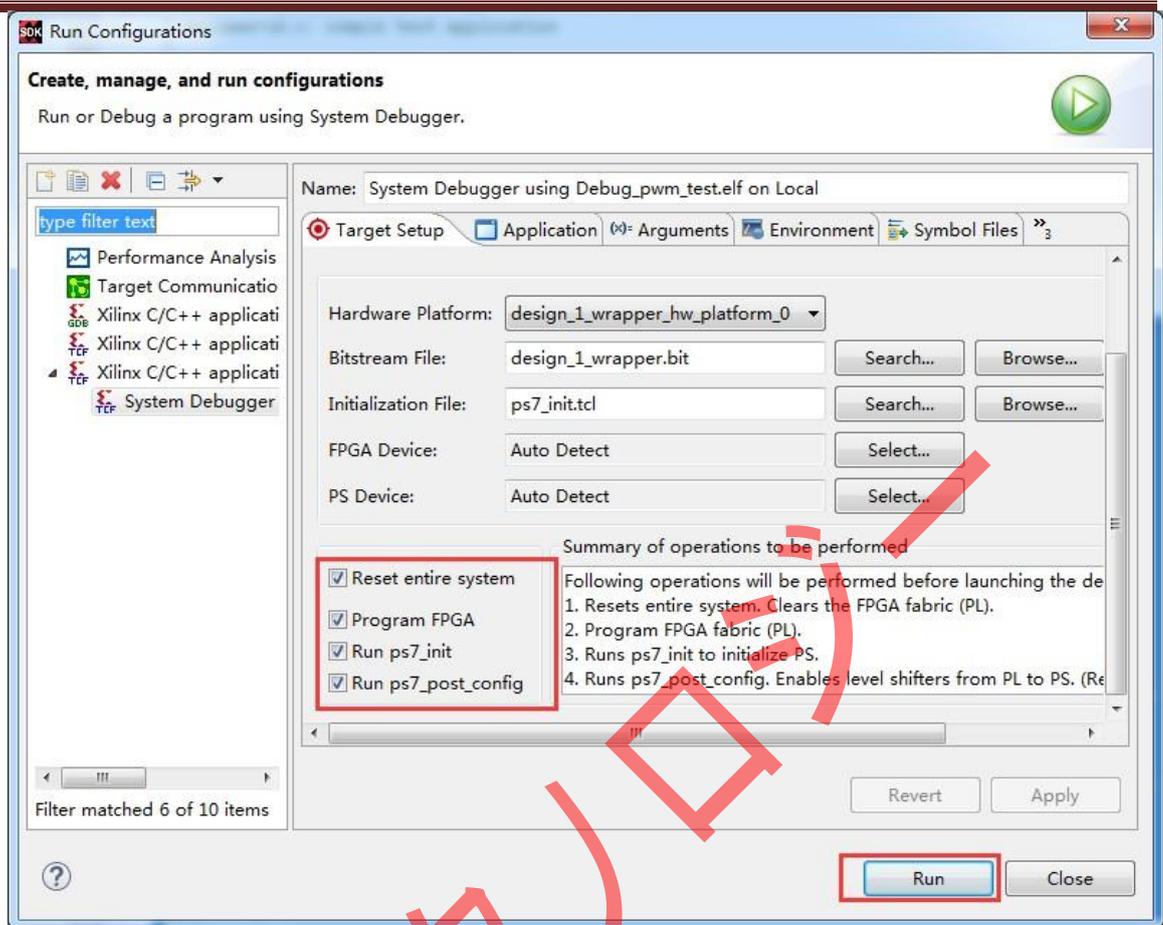
World¥n¥r");

    //pwm_out period = frequency(pwm_out) * (2 ** N) / frequency(clk);
    AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR,
    AX_PWM_S00_AXI_SLV_REG0_OFFSET,
    17179); //200hz

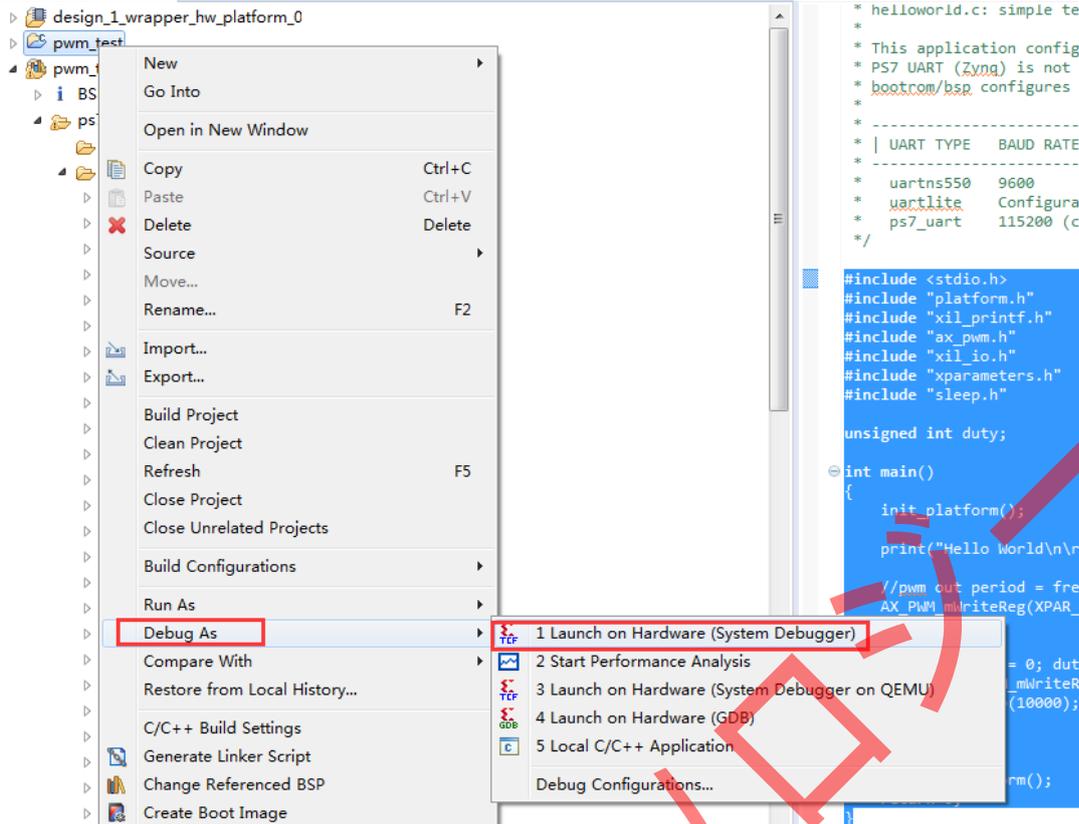
    while (1) {
        for (duty = 0x8fffffff; duty < 0xffffffff; duty = duty + 100000)
        { AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR,
        AX_PWM_S00_AXI_SLV_REG1_OFFSET,
        duty);
        }
        usleep(100);
    }

    cleanup_platform();
    return 0;
}
```

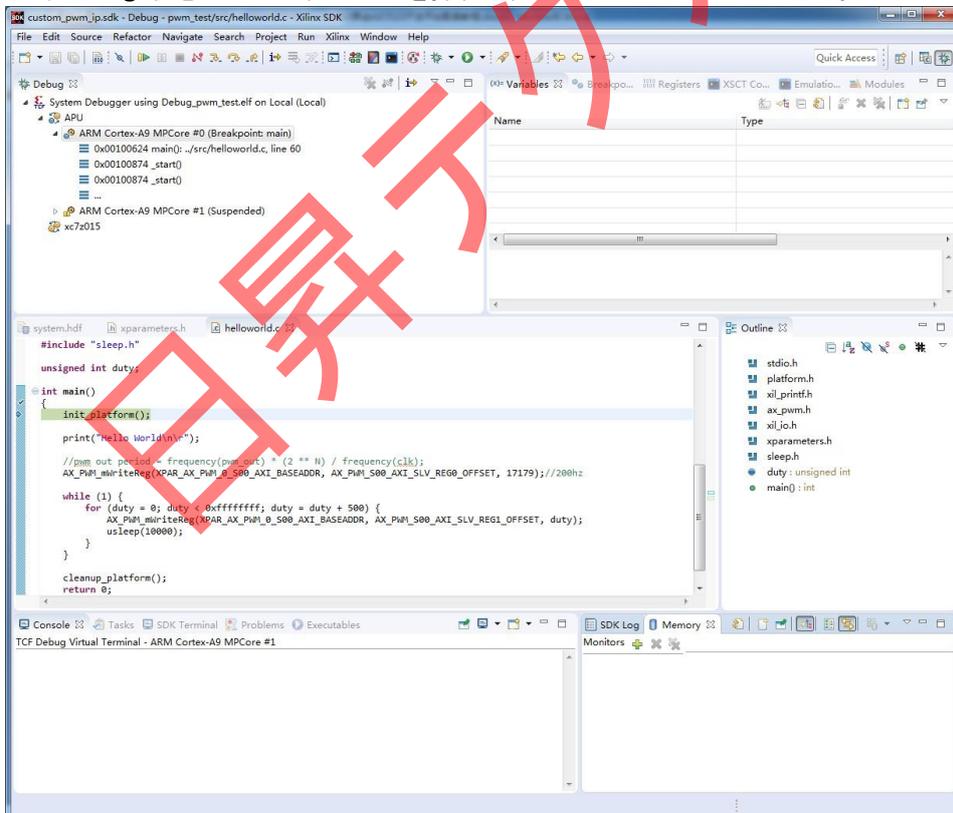
- 7) コードを作動することで、 PLLED1 が 呼吸ライトの効果を表現している。



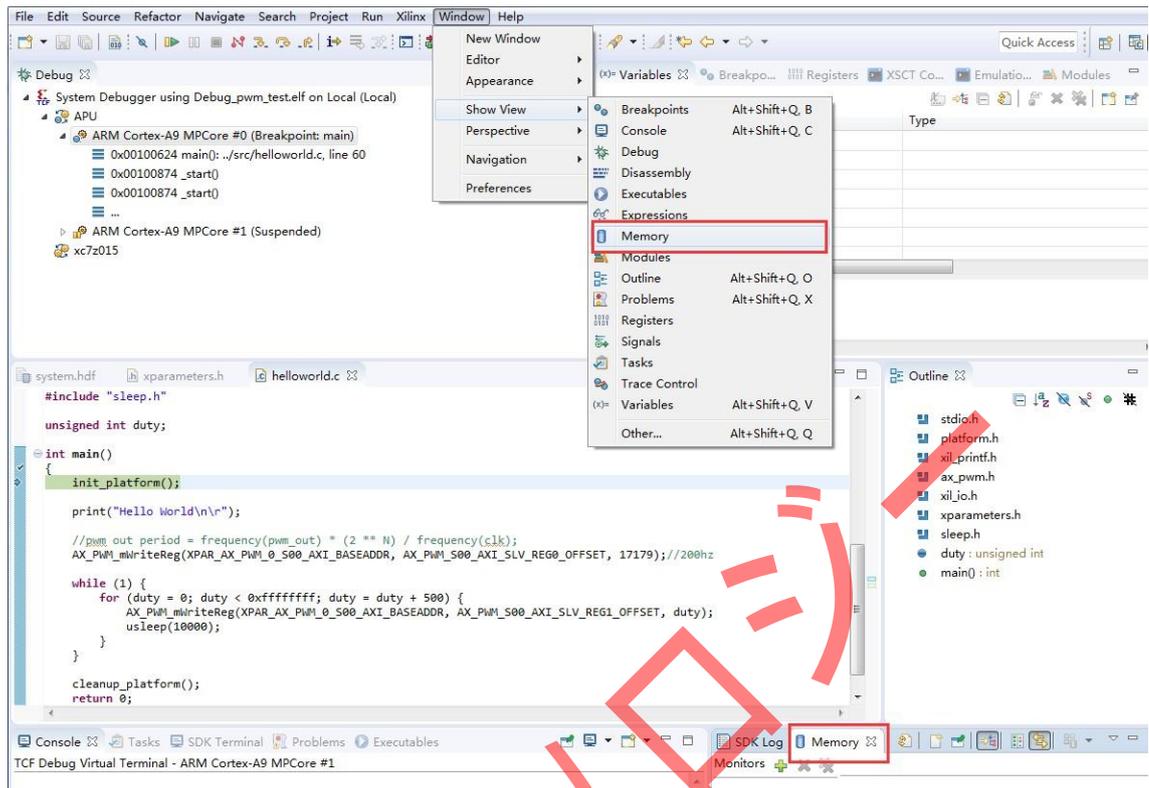
8) debug を通じて、レジスターの状況を見てみよう。



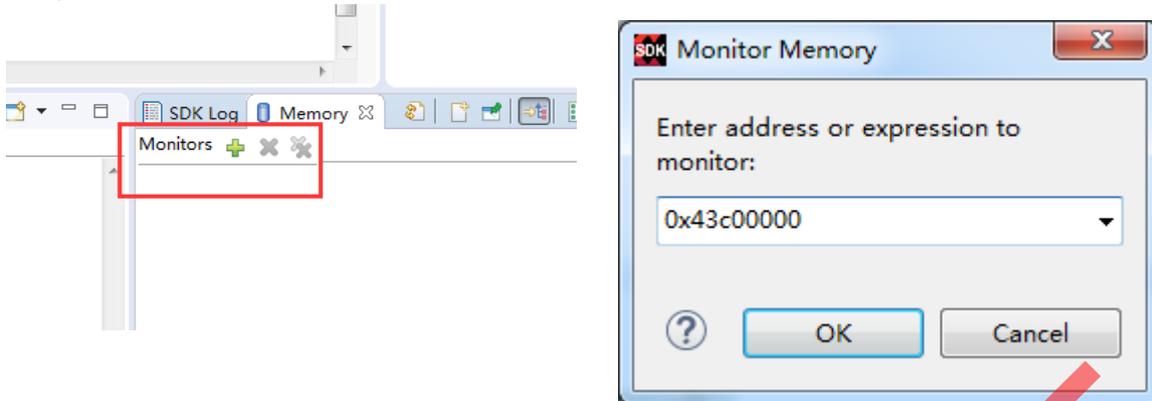
9) debug 状態に入って、“F6”を押すと、シングルステップができる。



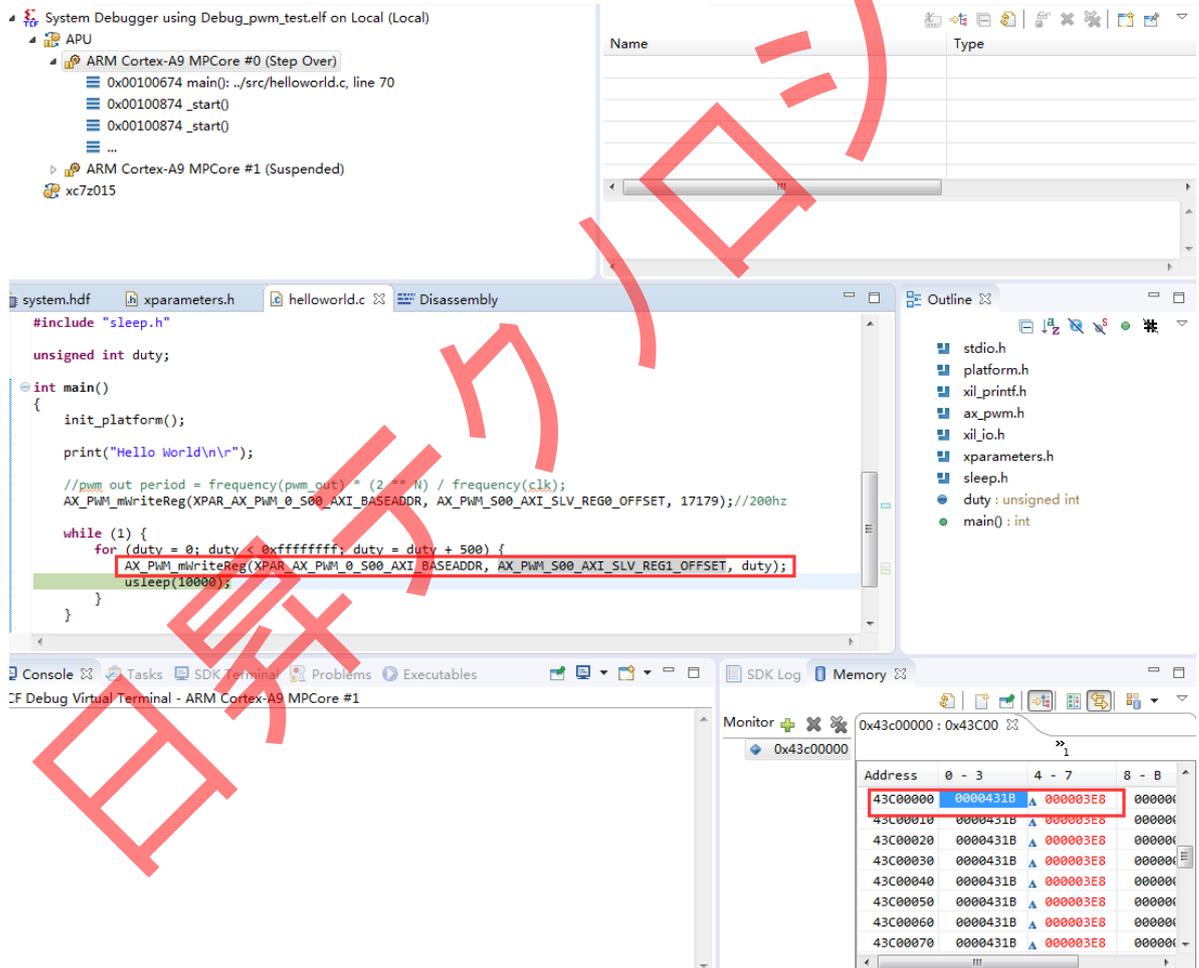
10) メニューから Window→Show View→Memory をクリックする。



## 11) 監視アドレス “0x43c00000” を追加する



## 12) ステップずつ実行して観察する。



System Debugger using Debug\_pwm\_test.elf on Local (Local)

- APU
  - ARM Cortex-A9 MPCore #0 (Step Over)
    - 0x00100674 main(): ../src/helloworld.c, line 70
    - 0x00100874 \_start()
    - 0x00100874 \_start()
    - ...
  - ARM Cortex-A9 MPCore #1 (Suspended)
    - xc7z015

```

#include "sleep.h"

unsigned int duty;

int main()
{
    init_platform();
    print("Hello World\n\r");

    //pwm out period = frequency(pwm_out) * (2 ** N) / frequency_clk;
    AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR, AX_PWM_S00_AXI_SLV_REG1_OFFSET, 17179);//200hz

    while (1) {
        for (duty = 0; duty < 0xffffffff; duty = duty + 500) {
            AX_PWM_mWriteReg(XPAR_AX_PWM_0_S00_AXI_BASEADDR, AX_PWM_S00_AXI_SLV_REG1_OFFSET, duty);
            usleep(10000);
        }
    }
}
    
```

Address	0 - 3	4 - 7	8 - B
43C00000	0000431B	A	00003E8
43C00010	0000431B	A	00003E8
43C00020	0000431B	A	00003E8
43C00030	0000431B	A	00003E8
43C00040	0000431B	A	00003E8
43C00050	0000431B	A	00003E8
43C00060	0000431B	A	00003E8
43C00070	0000431B	A	00003E8

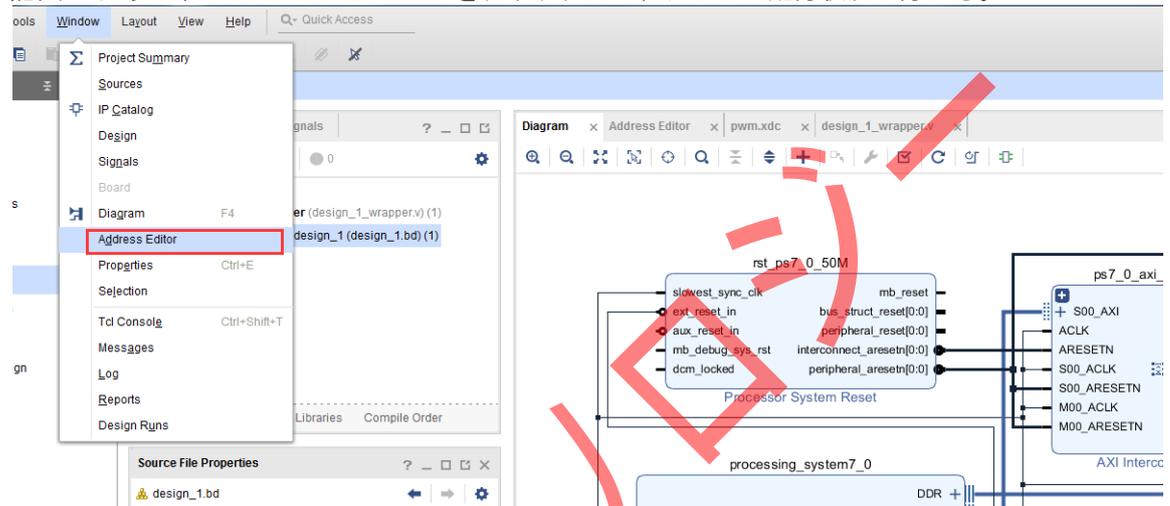
## 11.4 実験結果

本実験より SDK のデバッグ方法を勉強して、ARM と FPGA のデータ交換方法を把握する。

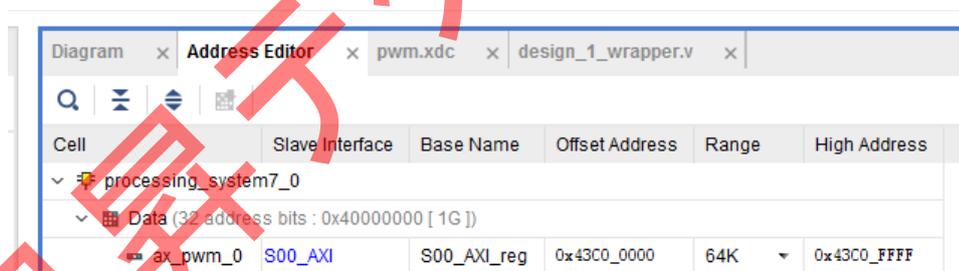
## 11.5 よくある問題

### 11.5.1 AXI IP のベースアドレスを調べる

- 1) 下記図のように、“Address Editor” をクリックして、アドレス配分状況が分かる。



- 2) 普通では Vivado が自動配分する。ユーザーもアドレスを修正できる。



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [ 1G ])					
ax_pwm_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

## 第十二章 VDMA を使用して HDMI ディスプレイを駆動する

実験Vivadoプロジェクトはvdma\_hdmi\_outである。

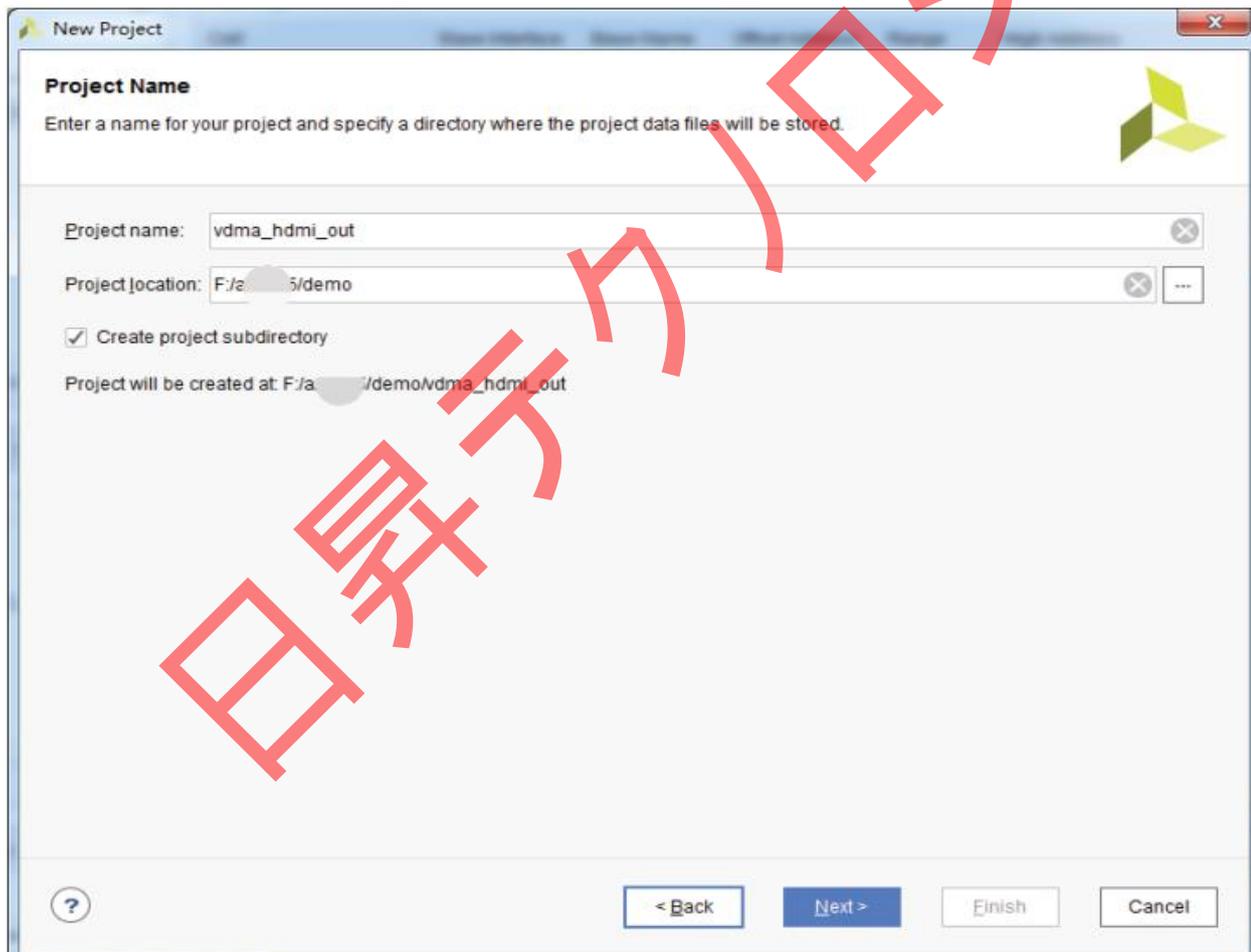
PSには統合されたディスプレイ制御システムがなく、PLを使用して実現する必要がある。実現の方法はいくつかあるが、どれもDMAシステムが必要である。DMAシステムは、ddr3からディスプレイへの表示を完了し、CPUオーバーヘッドを削減できる。VDMAは、ビデオ出力専用のxilinxが開発した特別なDMAであり、xilinxFPGAビデオ処理を学習する重要な部分である。

前のHDMI表示データはPLによって内部に生成されているが、この実験では、PSによって表示データが生成され、VDMAを介してPLがHDMIインターフェイスに送信される。

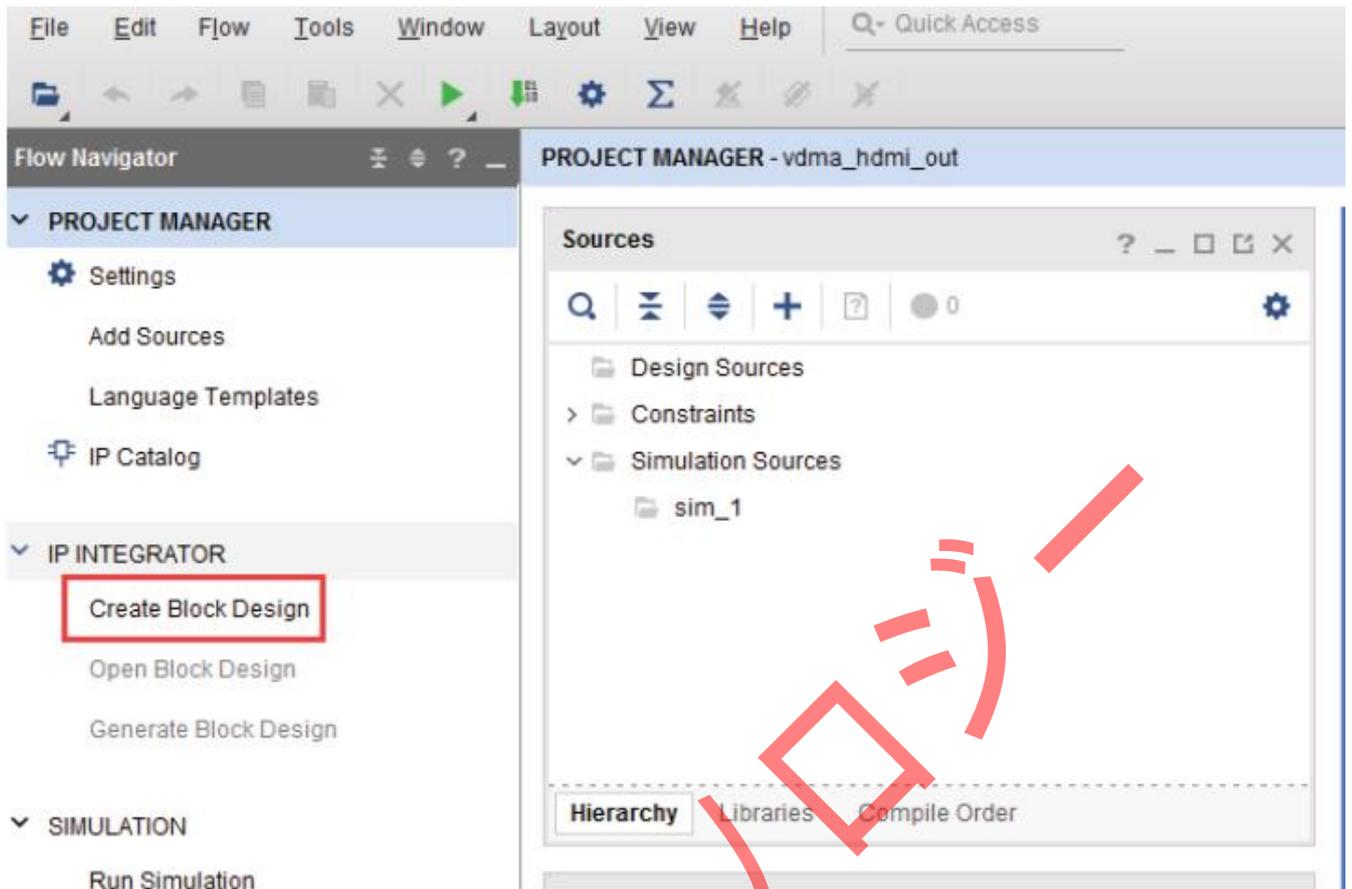
### 12.1 Vivado プロジェクトの設立

VDMAディスプレイは重要な部分であるため、この実験ではVivadoビルドプロセスの詳細を説明する。

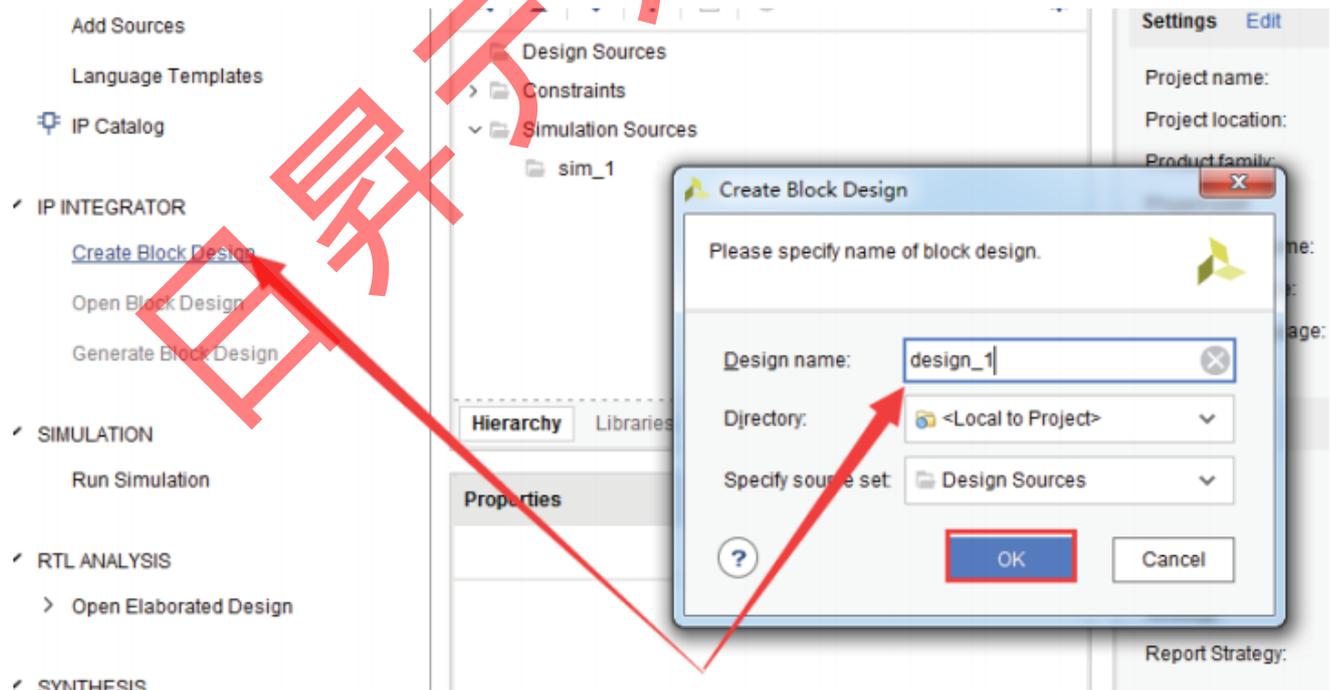
1) vdma\_hdmi\_outというプロジェクトを作成する。



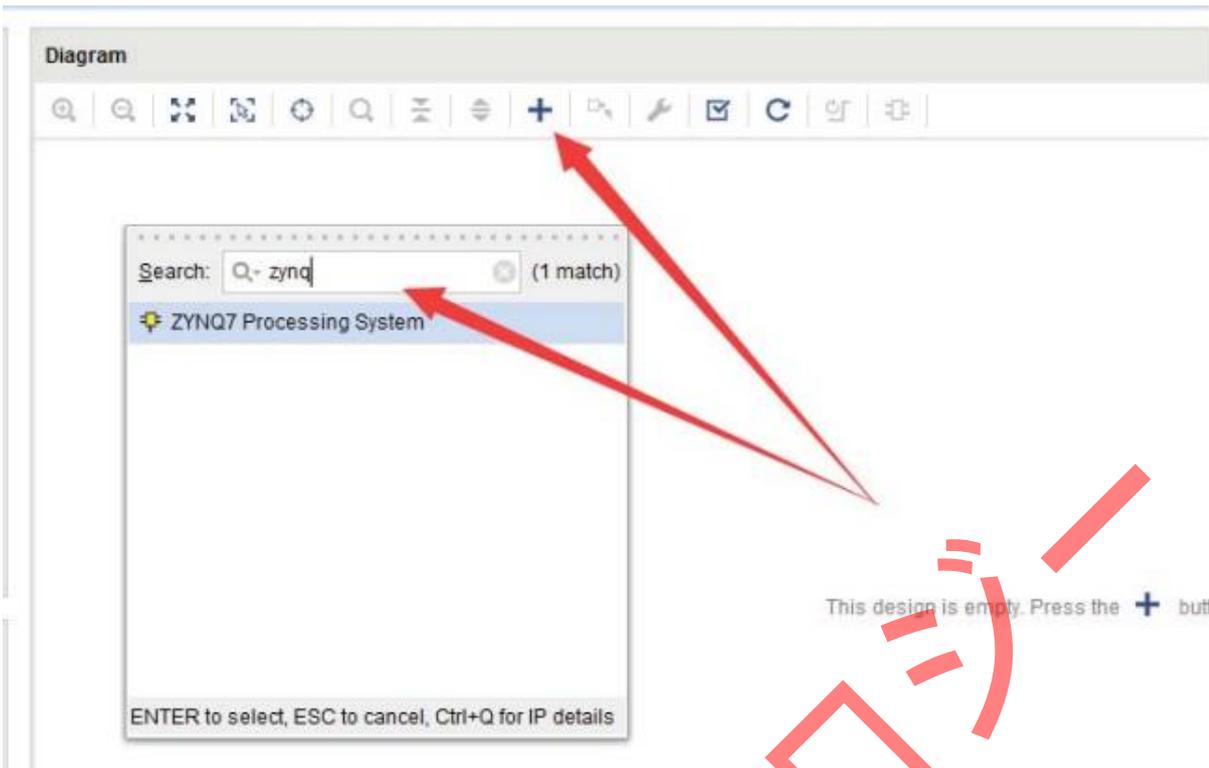
2) 新しいBlock デザインを作成する



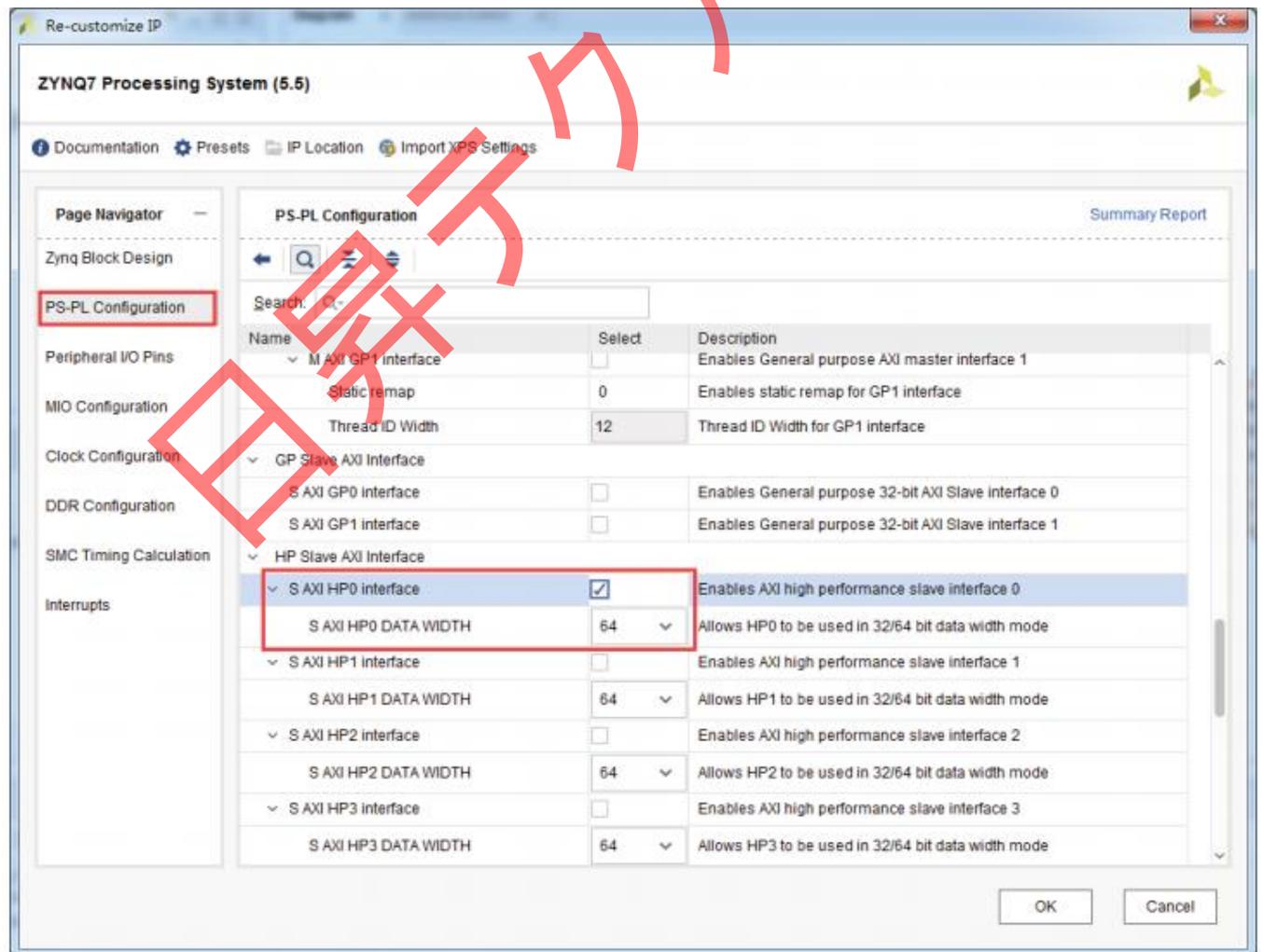
3) デザイン名はそのまま



4) ZYNQ プロセッサを追加する

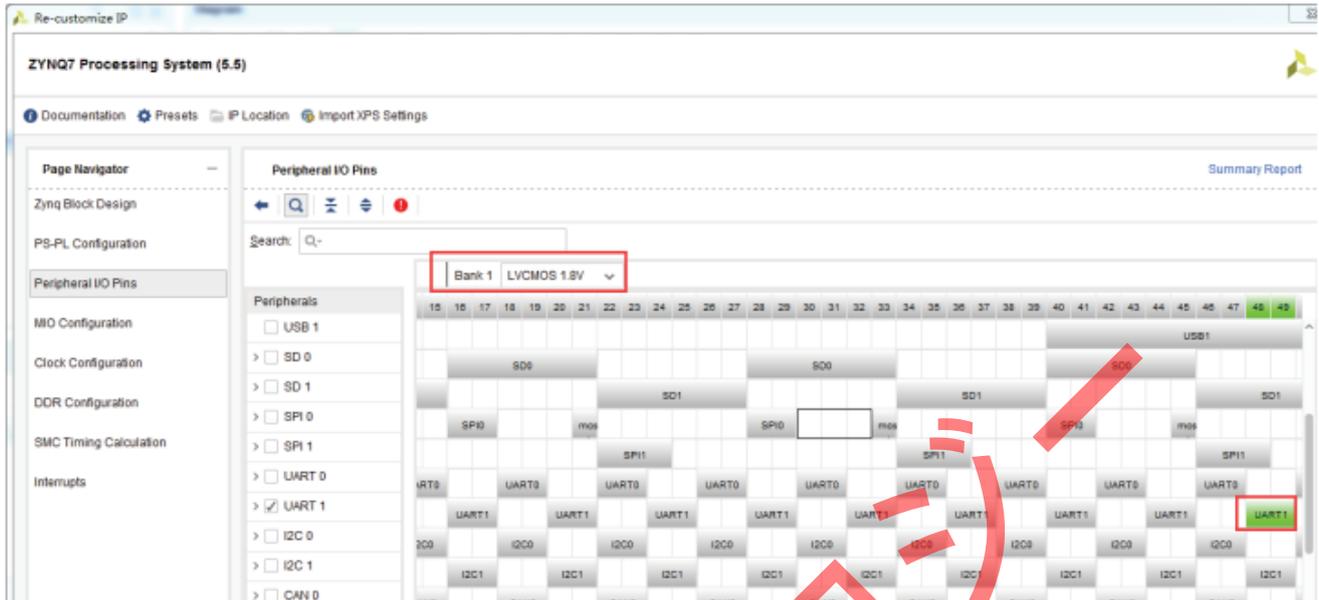


5) ZYNQ パラメーターをコンフィグし、VDMA 高速読み取り DDR3 の HPO インターフェイスを有効にする。



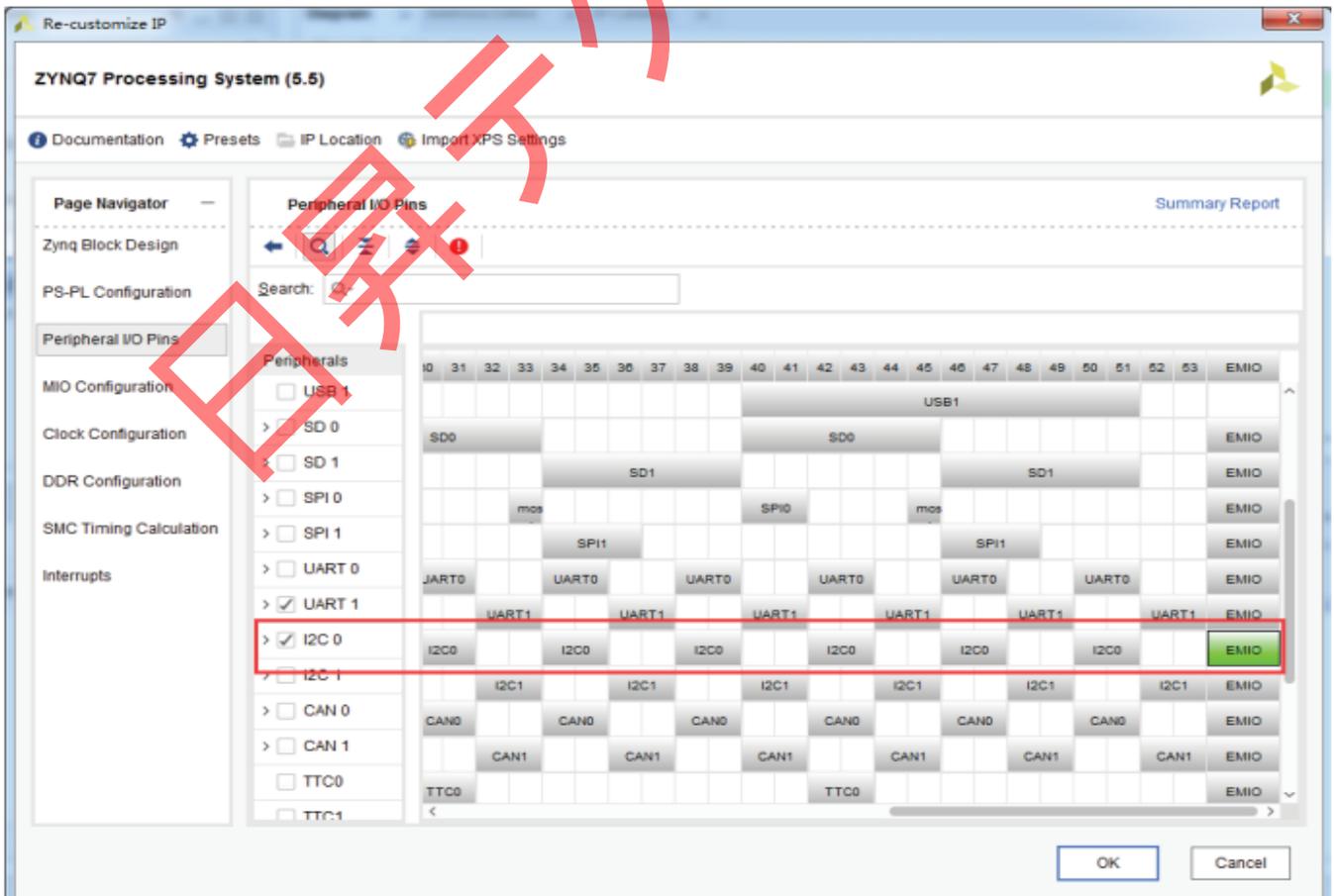
### 12.1.1 UART のコンフィグ

Bankレベル標準をコンフィグし、Bank0はLVCMOS 3.3V、Bank1はLVCMOS 1.8V、シリアルポートを有効にする

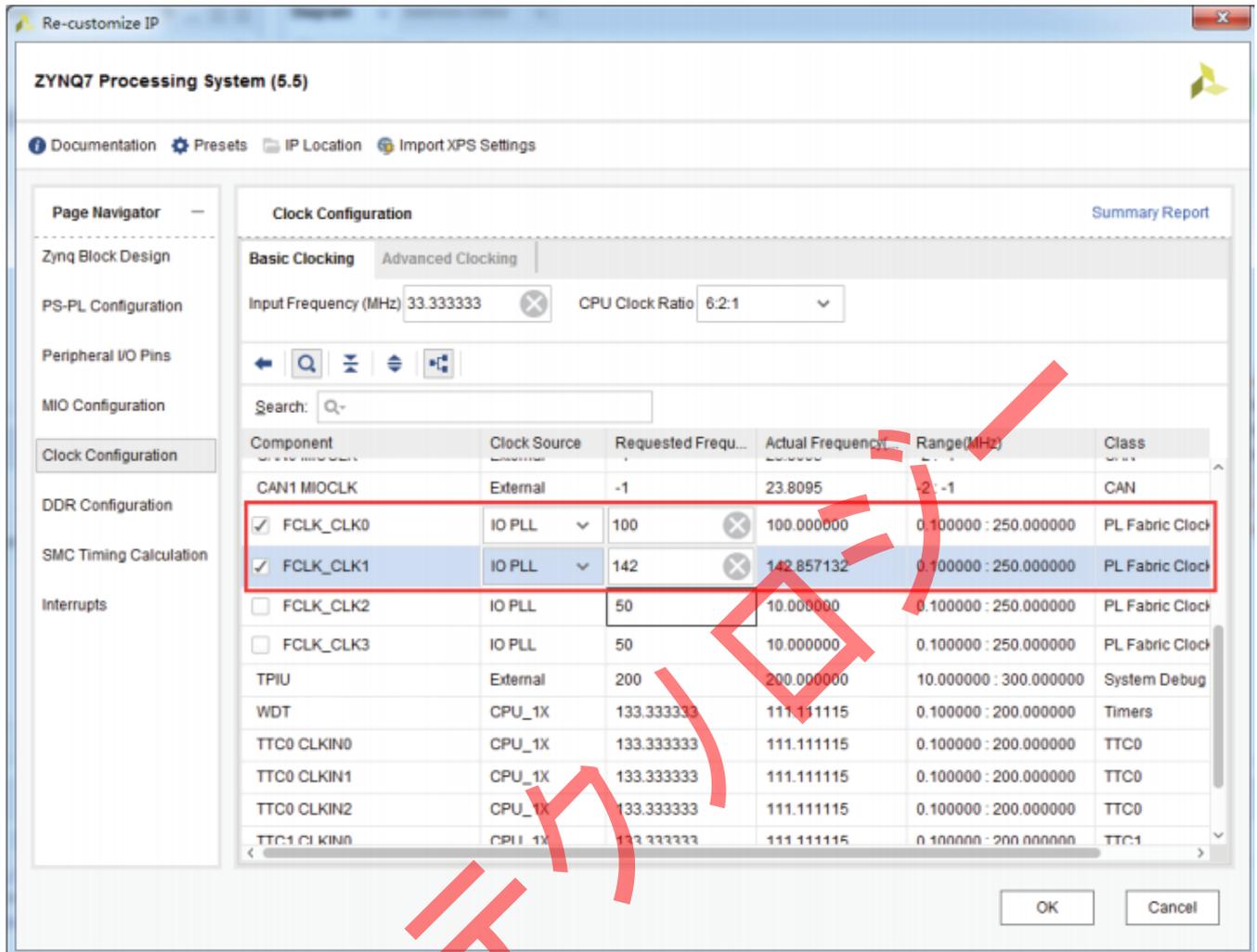


### 12.1.2 12C EMIO のコンフィグ

1) 12C0を有効にし、EMIOを選択して、12CをPL側に接続できるようにする。



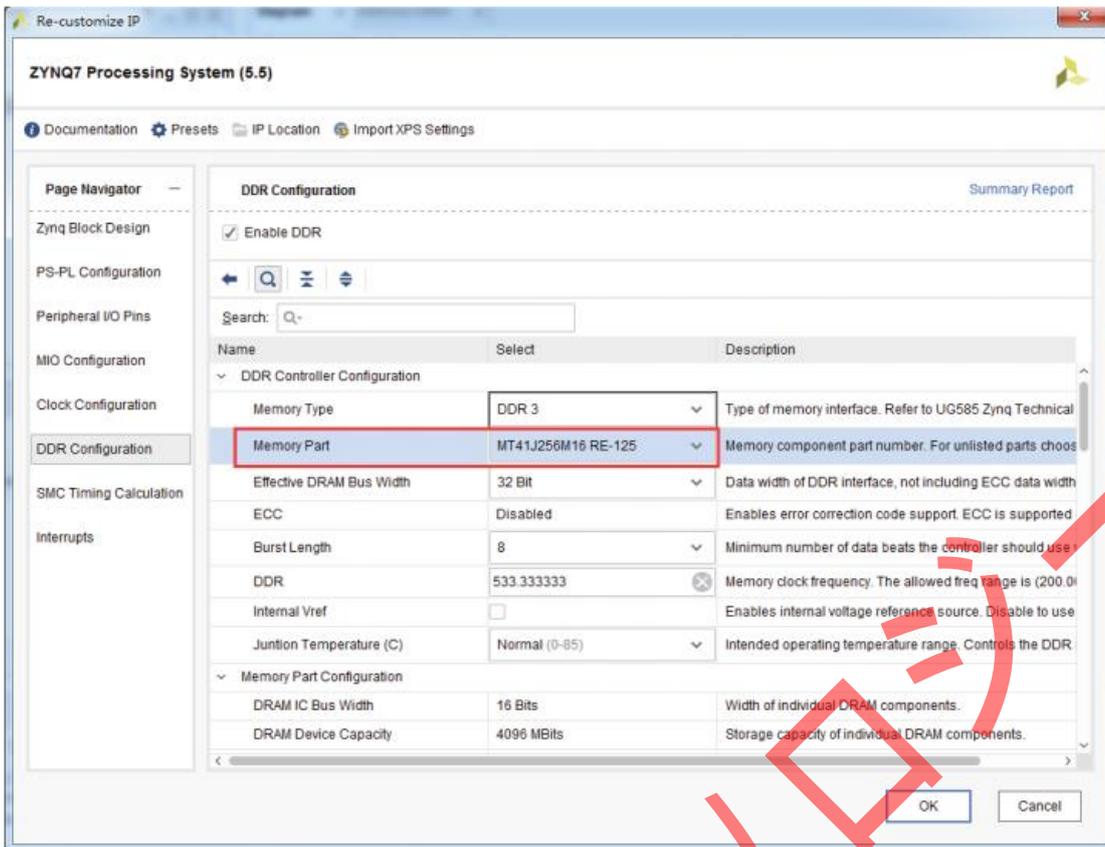
2) クロックを設定し、FCLK\_CLK0を100MHzに、FCLK\_CLK1を142MHzに設定し、このクロックの使用はVDMAデータを読み取るためである。



Component	Clock Source	Requested Frequency (MHz)	Actual Frequency (MHz)	Range (MHz)	Class
CAN1 MIOCLK	External	-1	23.8095	2 : -1	CAN
<input checked="" type="checkbox"/> FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 250.000000	PL Fabric Clock
<input checked="" type="checkbox"/> FCLK_CLK1	IO PLL	142	142.857132	0.100000 : 250.000000	PL Fabric Clock
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000	PL Fabric Clock
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000	PL Fabric Clock
TPIU	External	200	200.000000	10.000000 : 300.000000	System Debug
WDT	CPU_1X	133.333333	111.111115	0.100000 : 200.000000	Timers
TTC0 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000	TTC0
TTC0 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000	TTC0
TTC0 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000	TTC0
TTC1 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000	TTC1

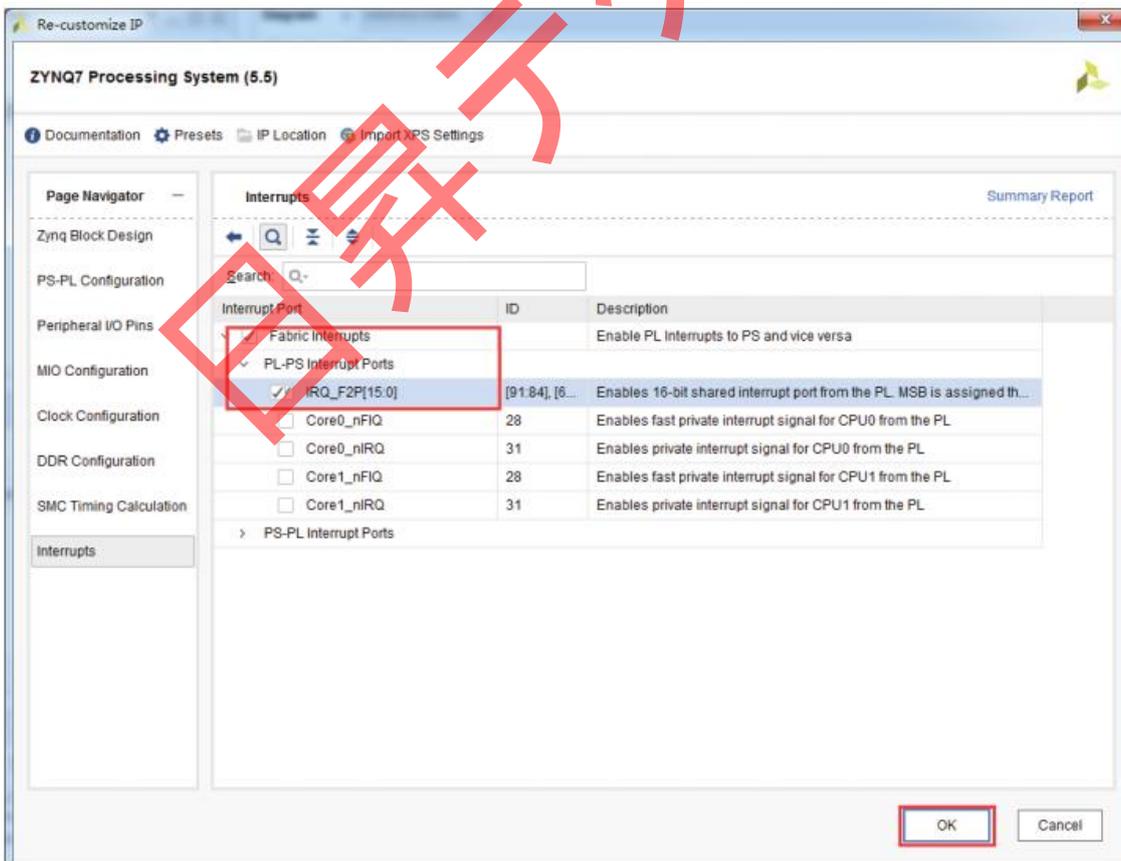
### 12.1.3 DDR3 のコンフィグ

DDR Configurationタブで、PS端子ddrのパラメーターをコンフィグできる。AX7010コンフィグDDR3モデルはMT41J128M16 HA-125、AX7020コンフィグDDR3モデルはMT41J256M16 RE-125である。ここでは、ddr3モデルはボード上のddr3モデルではない、パラメーターに最も近いモデルである。Effective DRAM Bus Width、32 Bitを選択する。



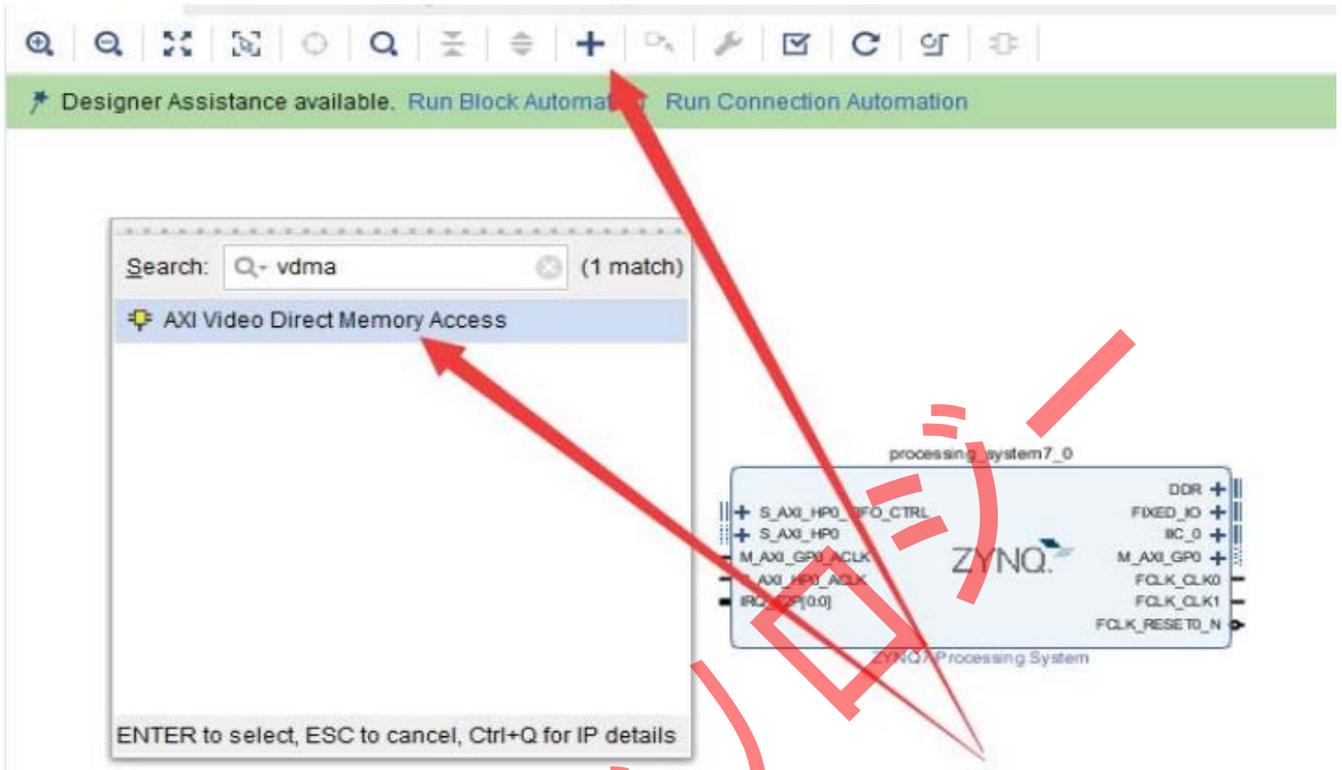
#### 12.1.4 コンフィグ割り込み

割り込みの設定、IRQ\_F2Pを有効にし、PL側で割り込みを受信

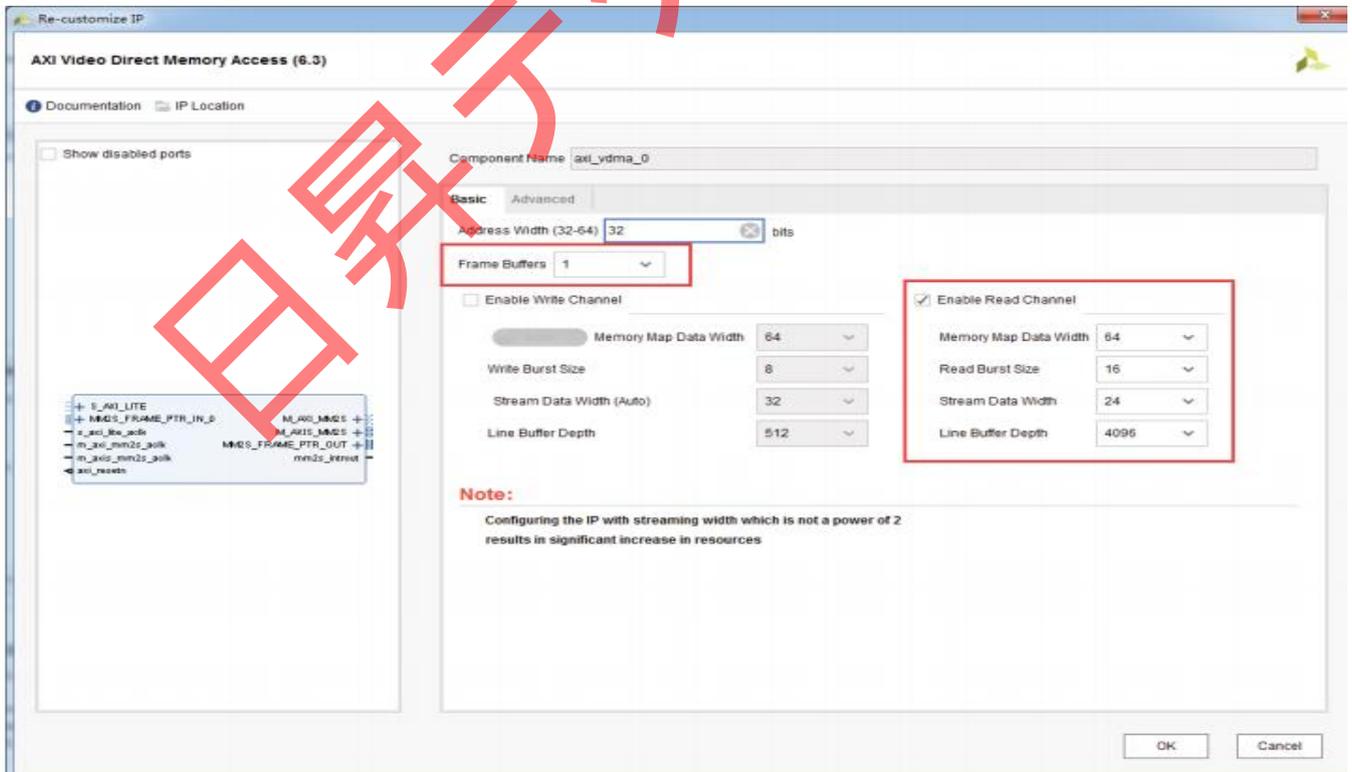


### 12.1.5 VDMA のコンフィグ

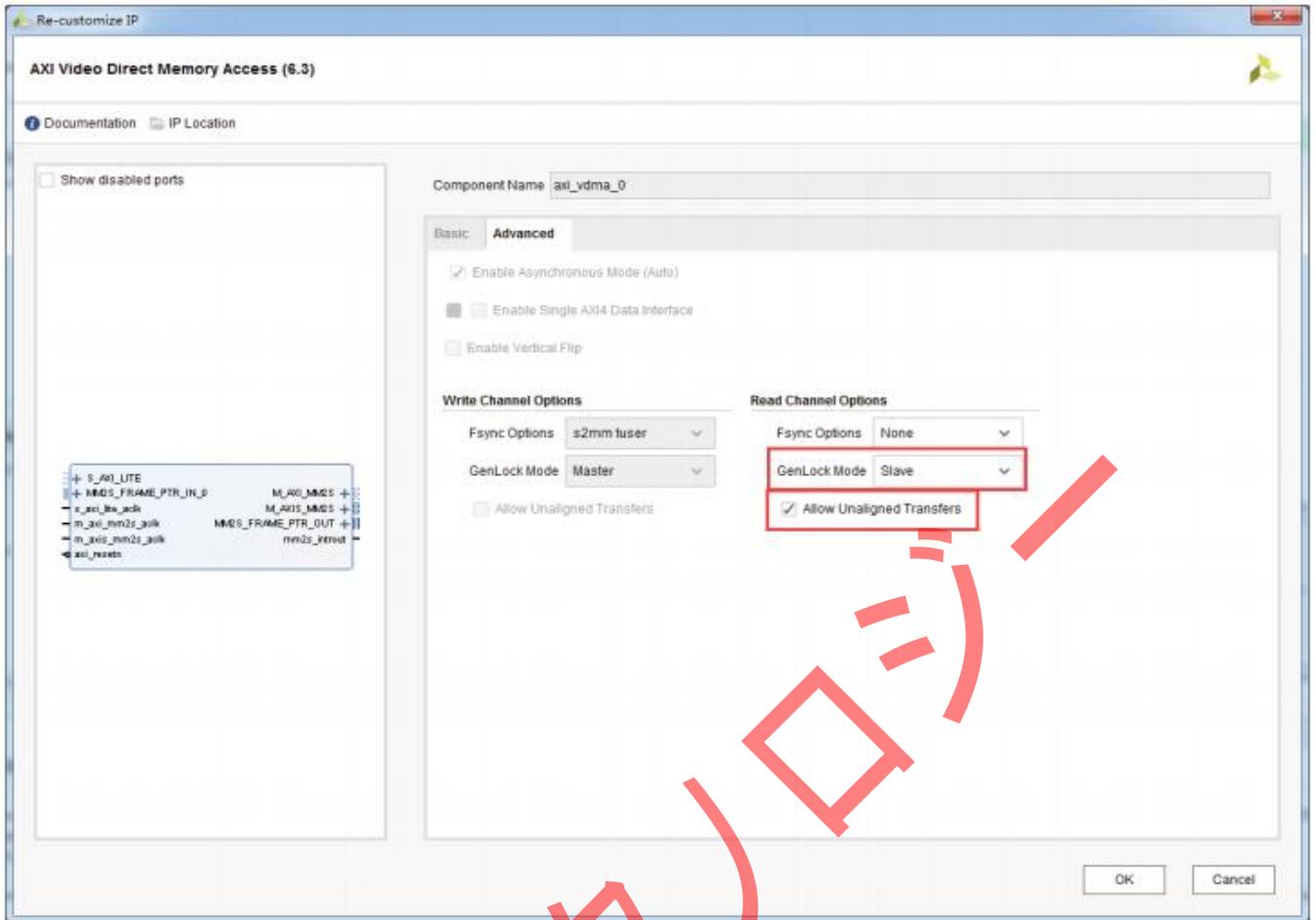
#### 1) VDMA IP を追加する



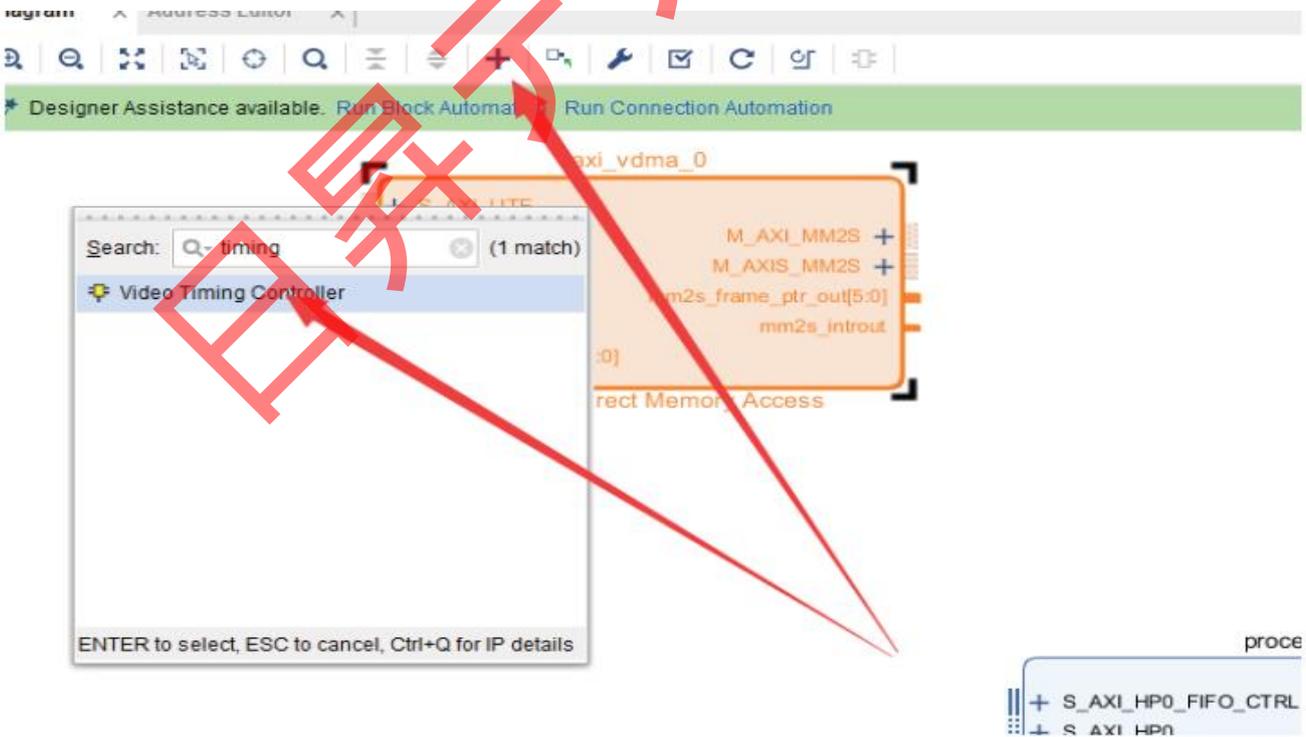
#### 2) 以下に示すように、VDMA基本パラメーターをコンフィグする。



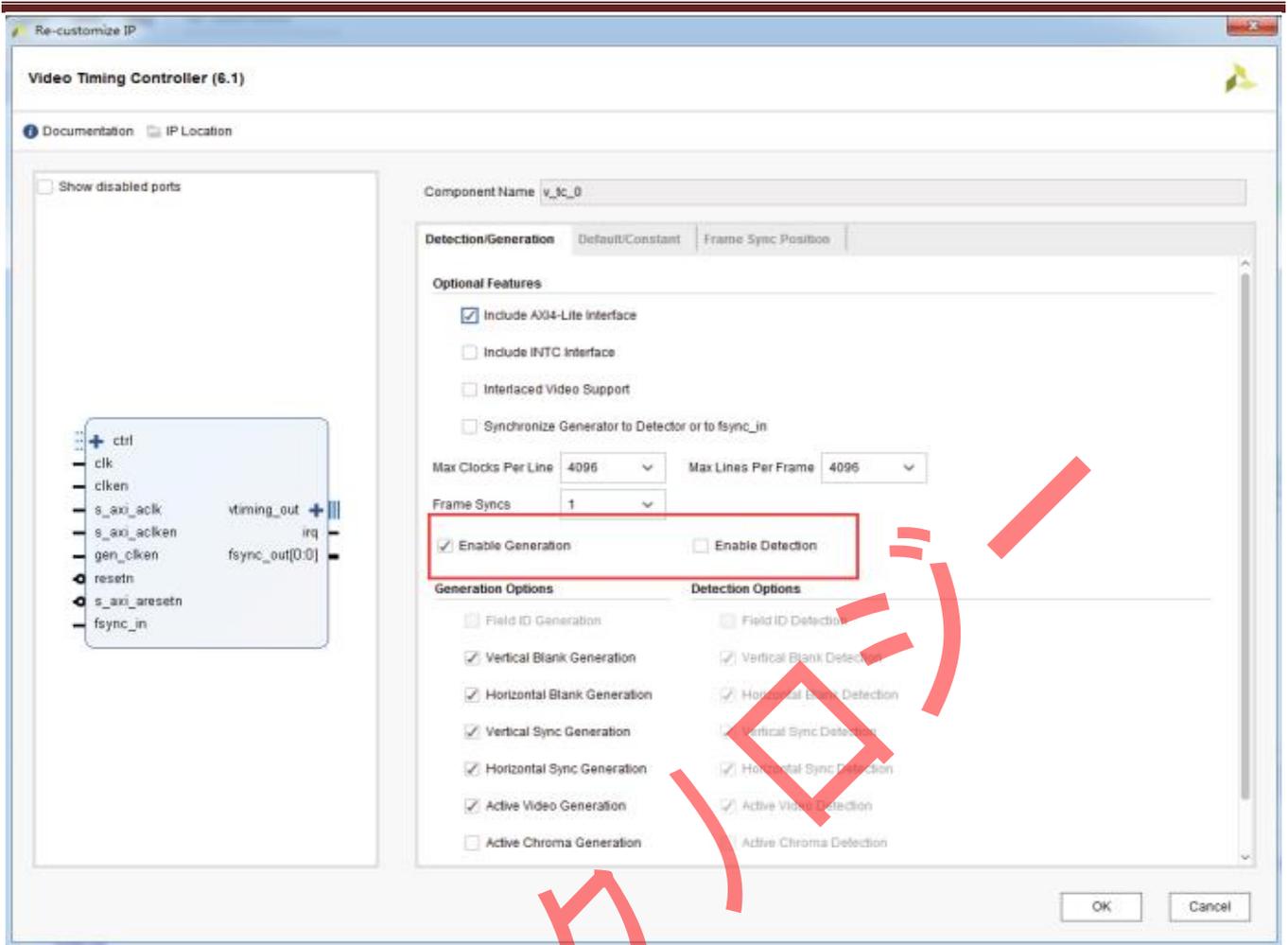
#### 3) VDMA高度なパラメーターのコンフィグ



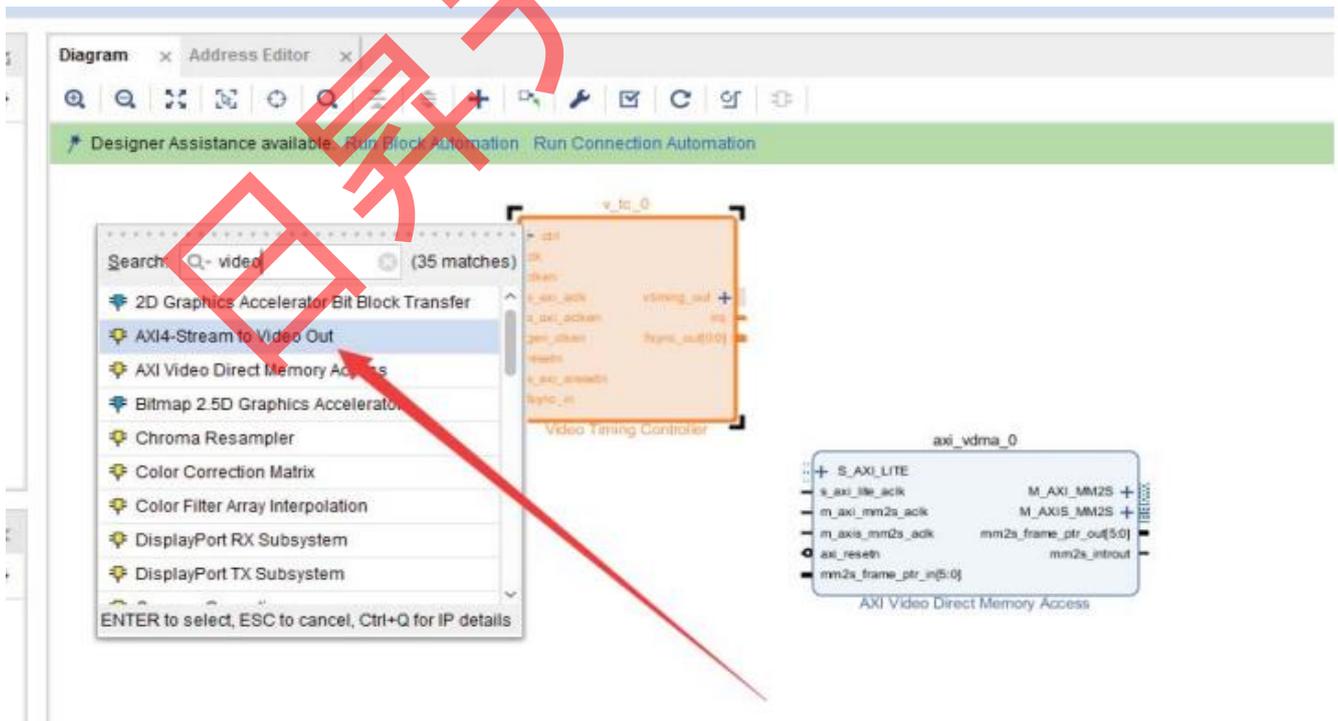
4) ビデオタイミングコントローラーを追加する



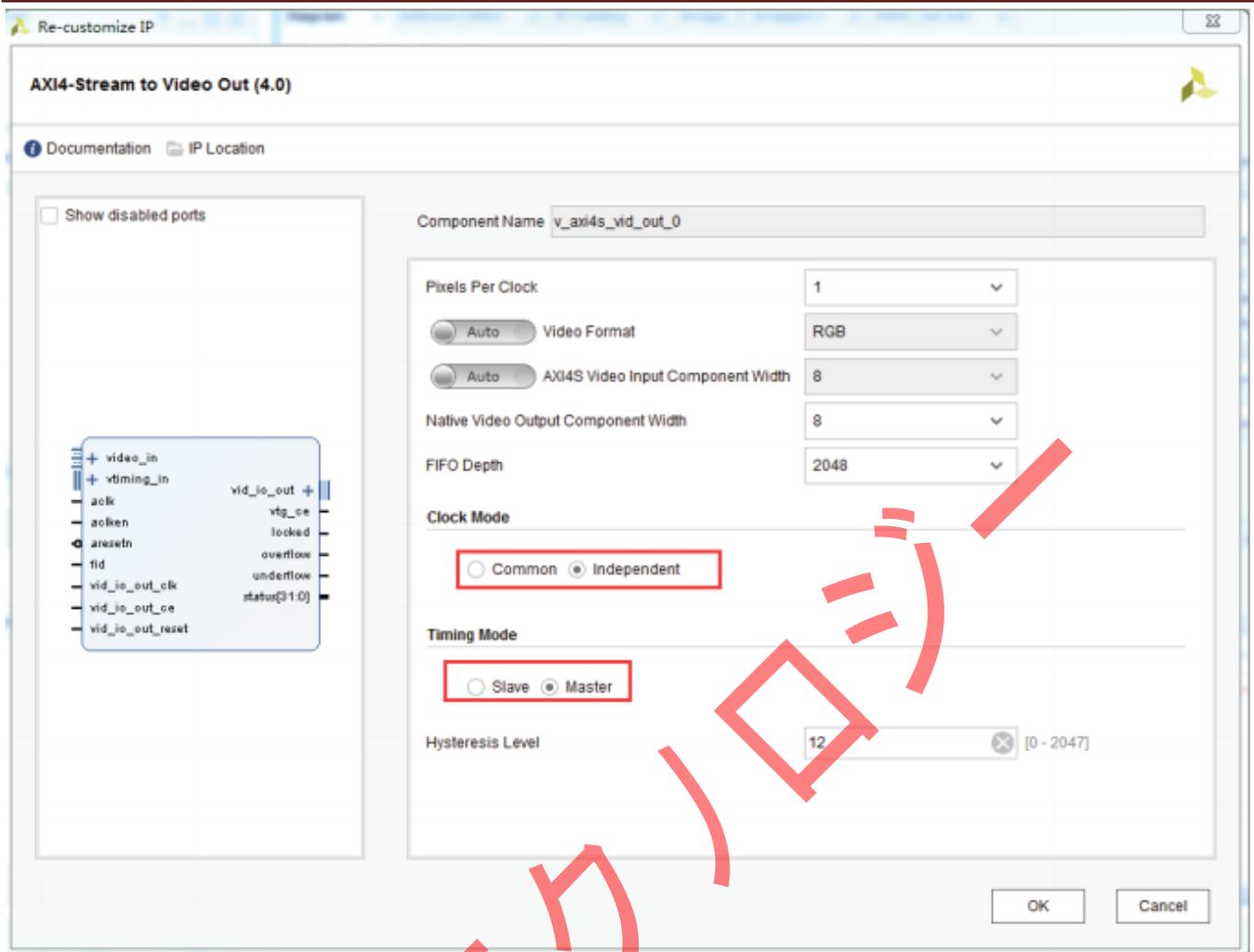
5) ビデオタイミングコントローラーパラメーターのコンフィグ



6) AXIストリーミングビデオ出力コントローラーを追加する



7) AXIストリーミングビデオ出力コントローラーパラメーターのコンフィグ

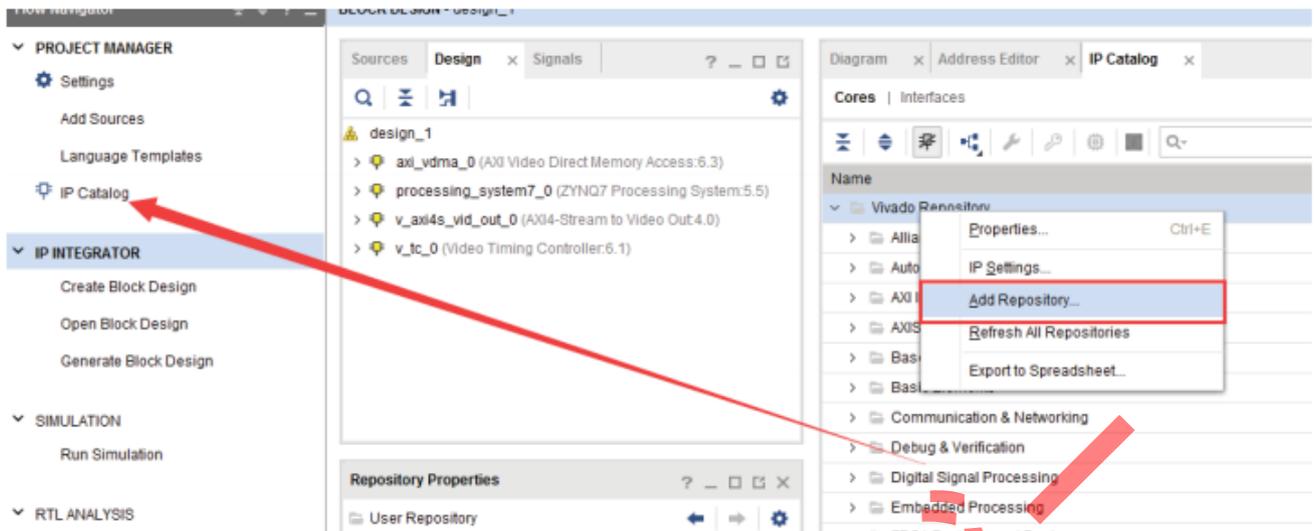


### 12.1.6 カスタム IP を追加する

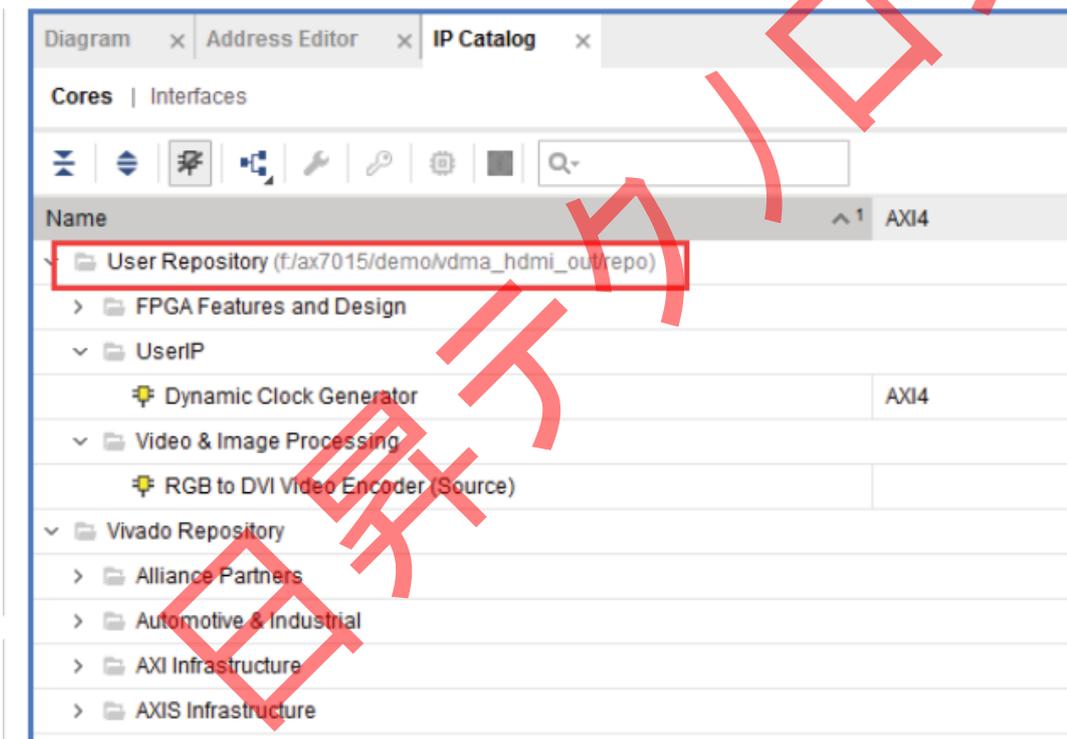
1) ビデオには多くの解像度があるため、さまざまなクロック周波数が異なるため、ダイナミッククロックコントローラーを使用する必要がある。このIPはオープンソースソフトウェアから提供されている。サンプル内のrepoディレクトリを見つけて自分のディレクトリにコピーする。



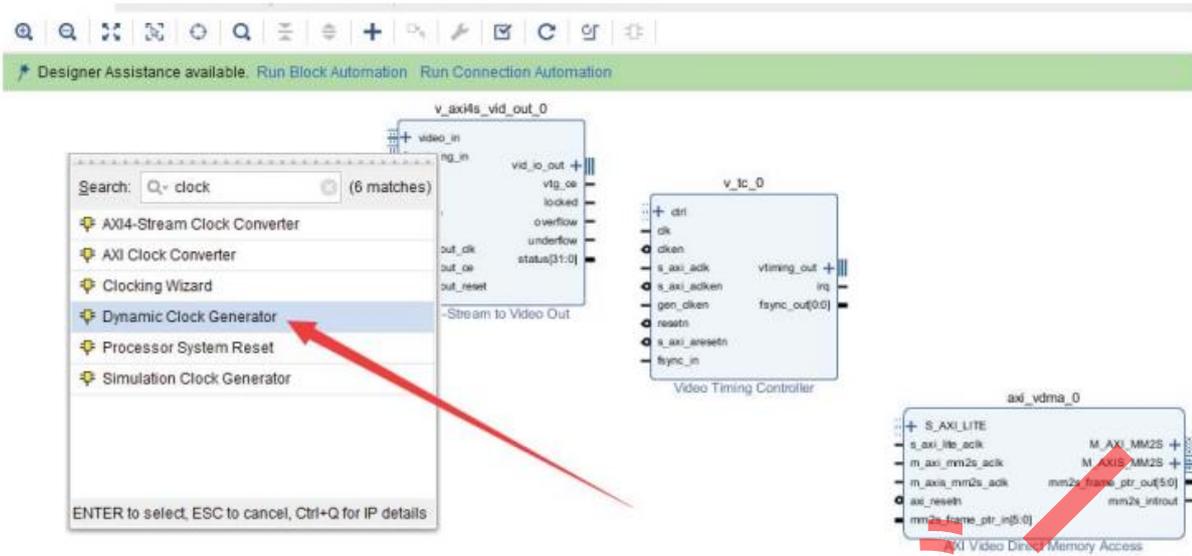
2) IPウェアハウスを追加する



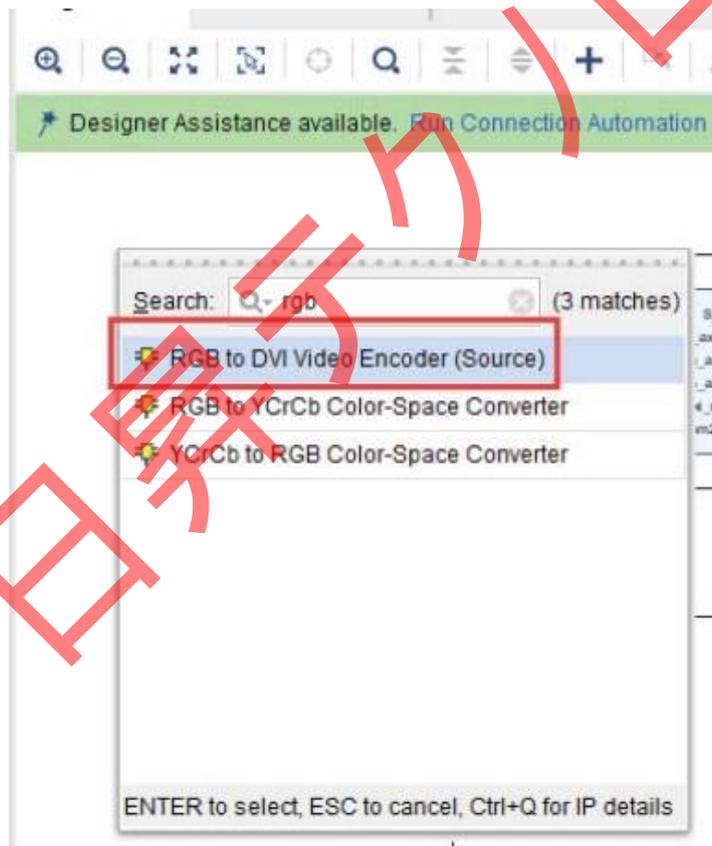
3) 追加が完了すると、多くのIPが表示される。

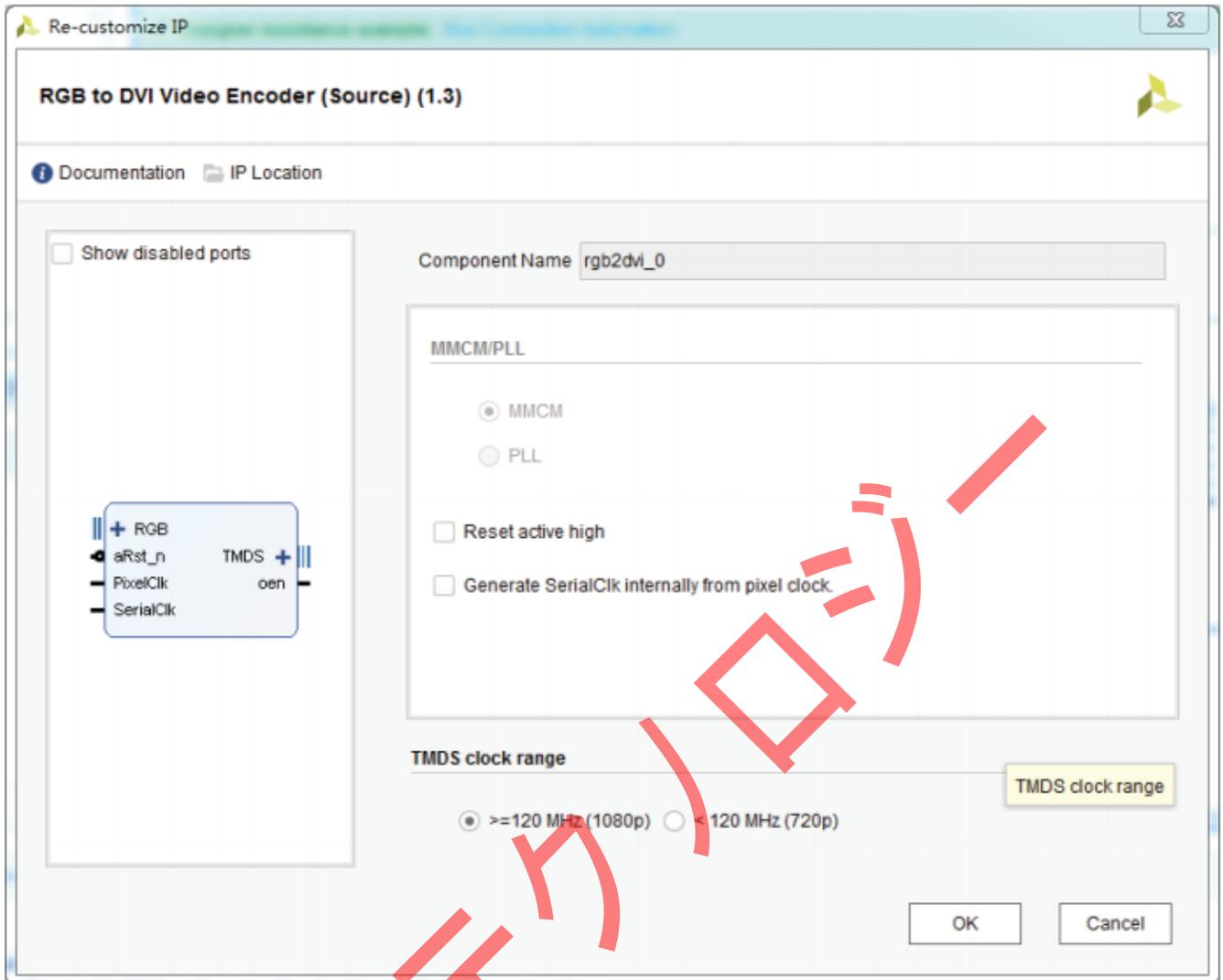


4) ダイナミッククロックコントローラーを追加する

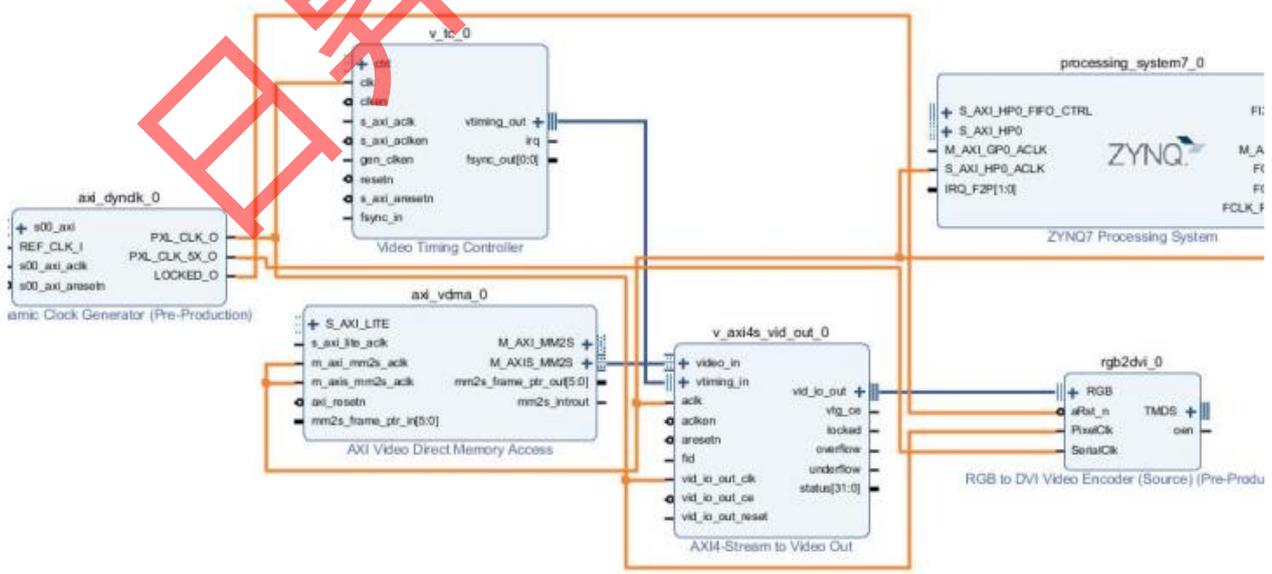


### 12.1.7 HDMI エンコーダーを追加する

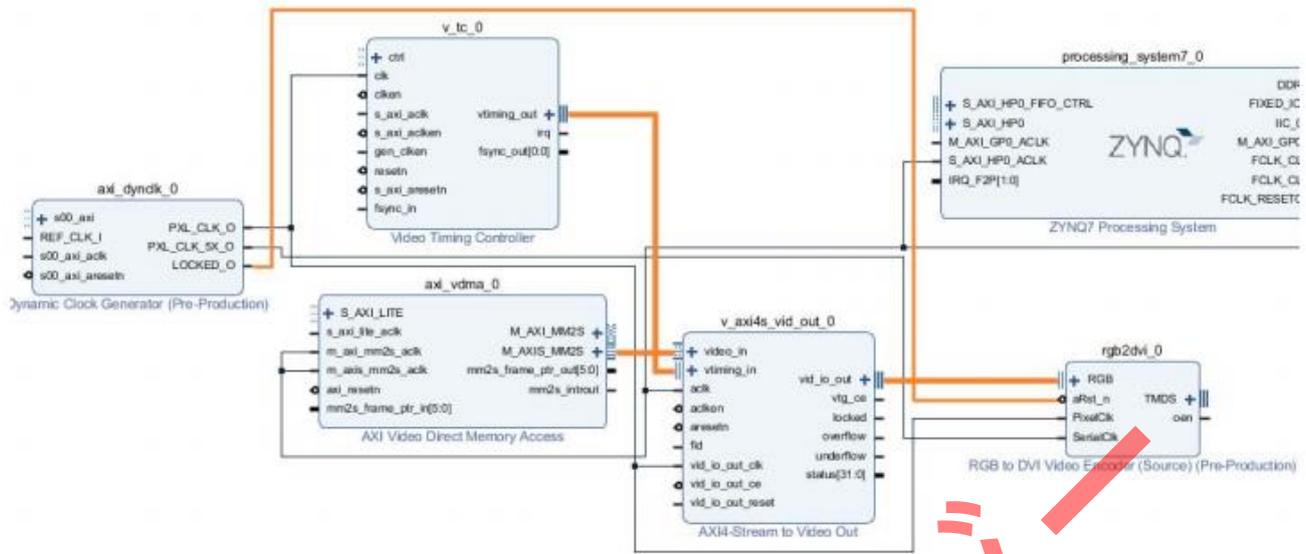




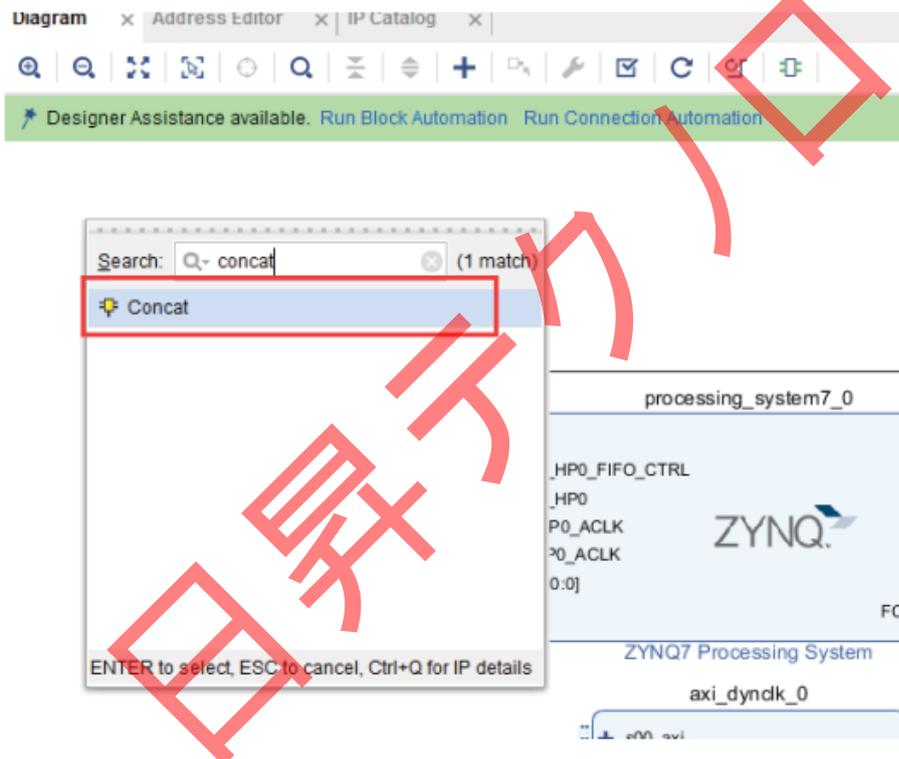
1) Vivadoへの接続がクロック信号に自動的に接続しない場合がある

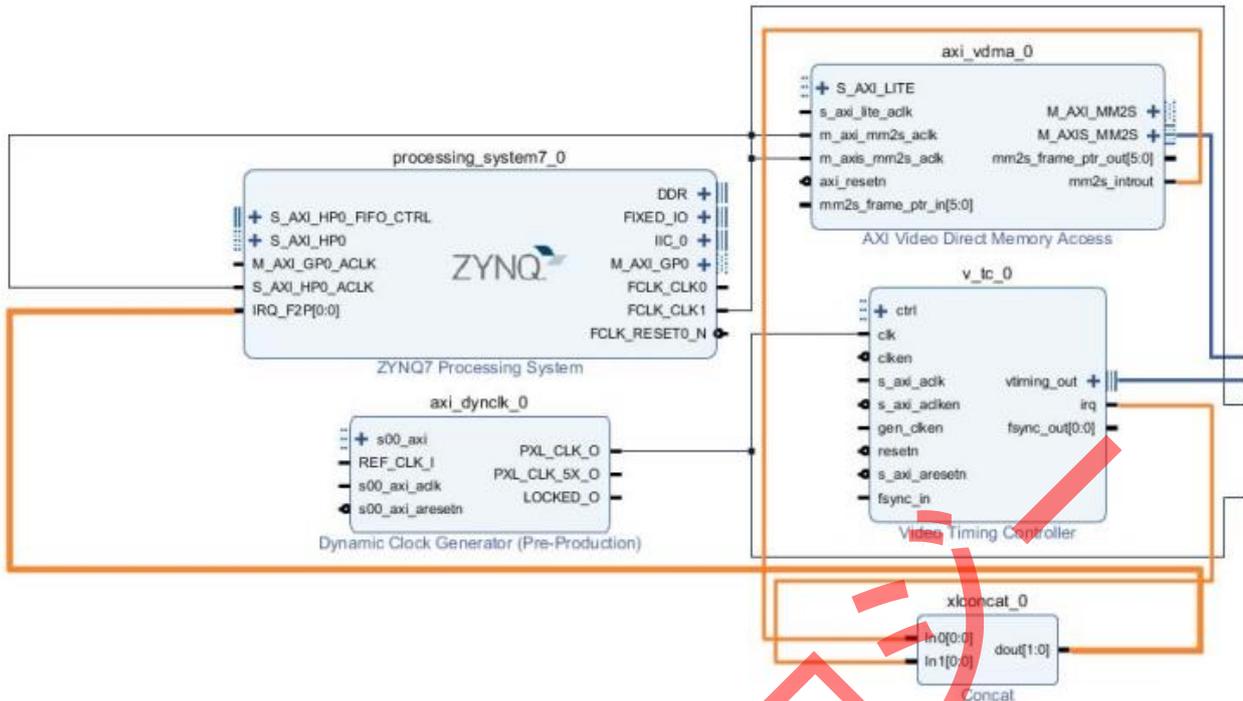


2) 他の重要な信号を接続する

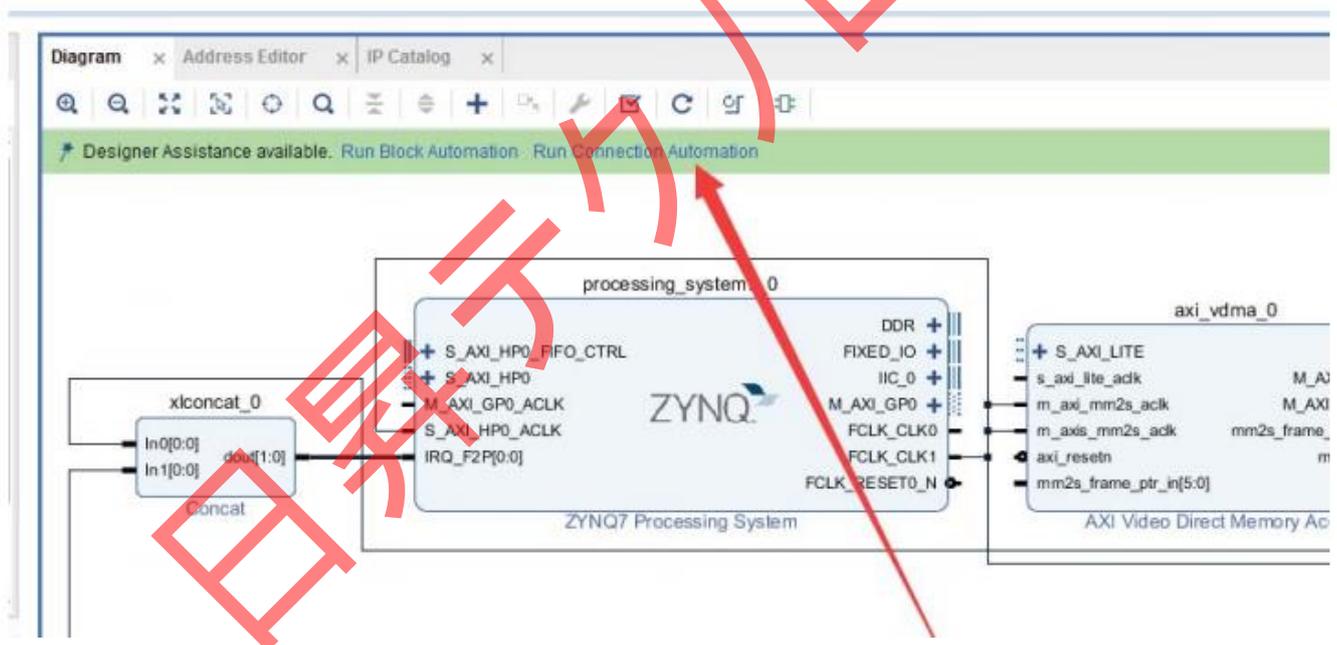


3) 割り込み信号に接続するには、最初に信号接続用のConcat IPを追加する必要がある。

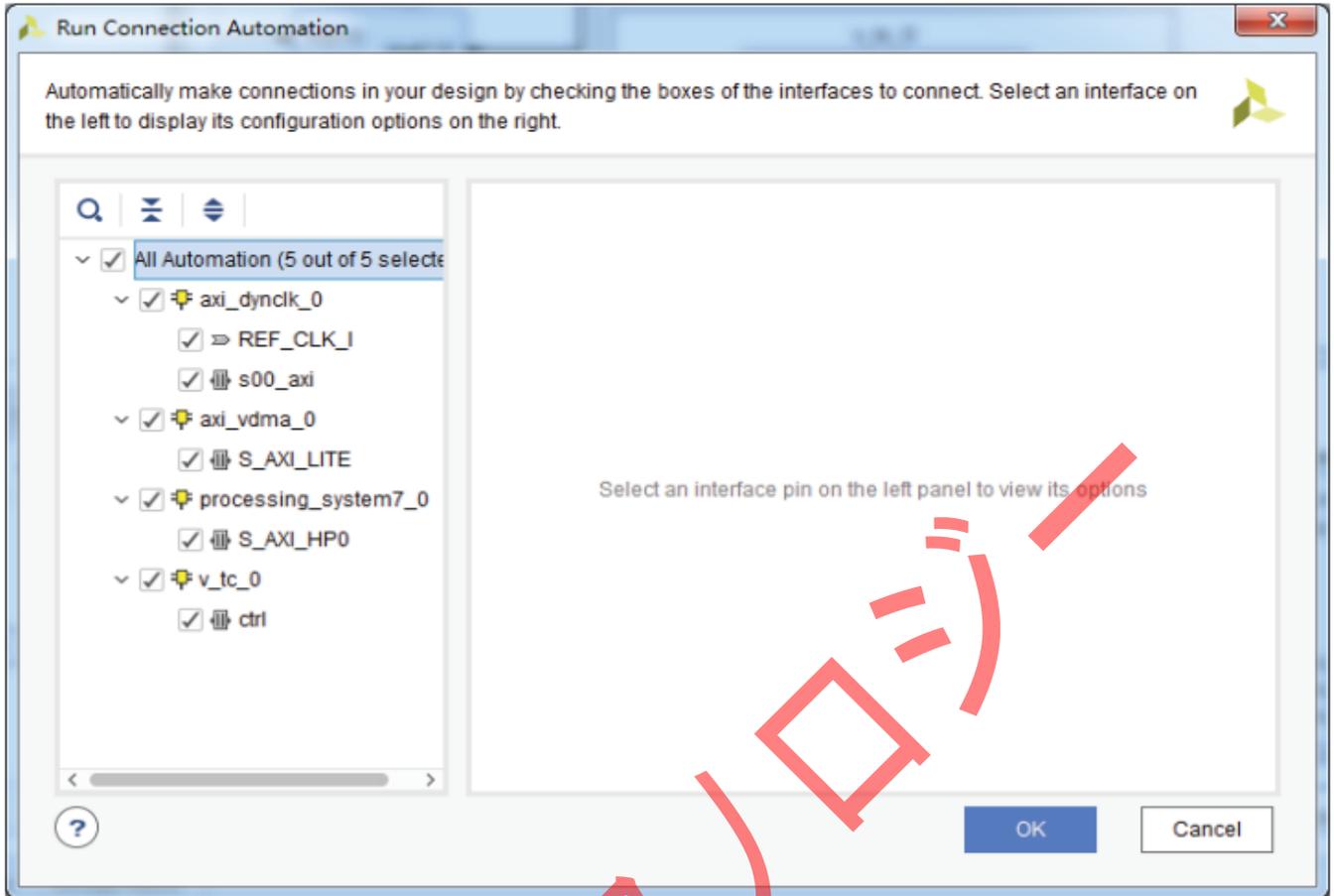




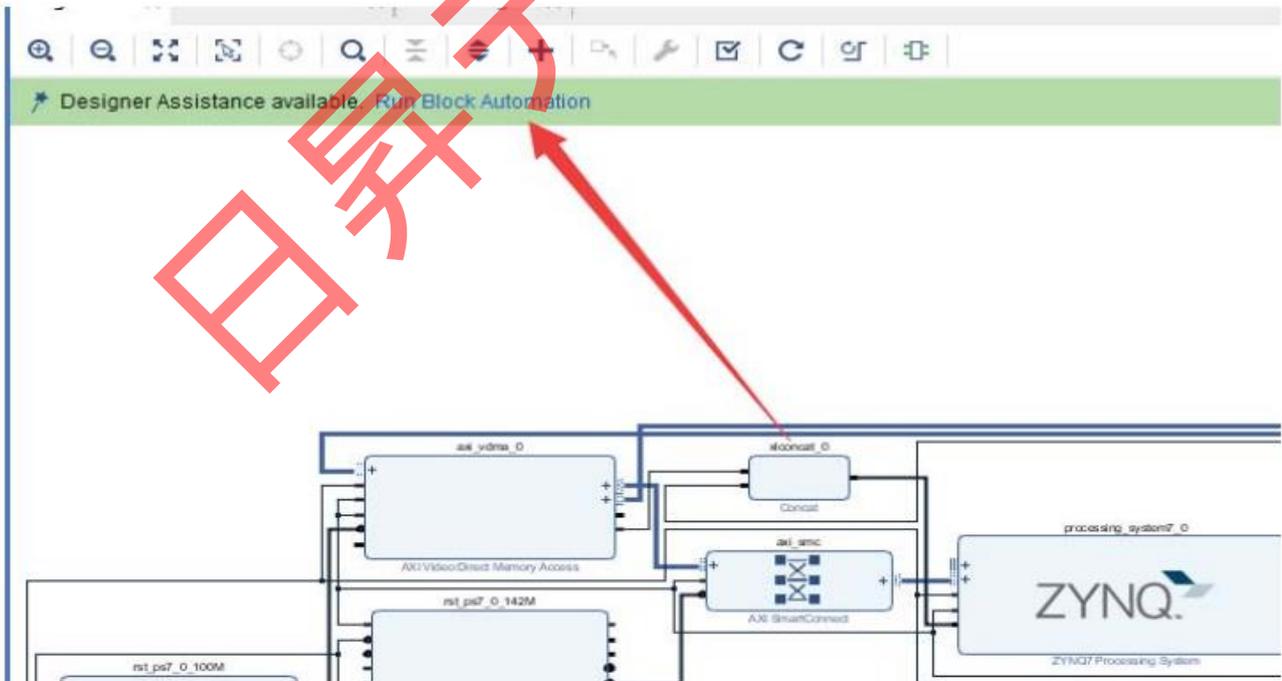
4) Vivadoの自動接続機能を使用して、残りのケーブル接続を完成させる。



5) すべてのモジュールを選択して自動的に接続する。

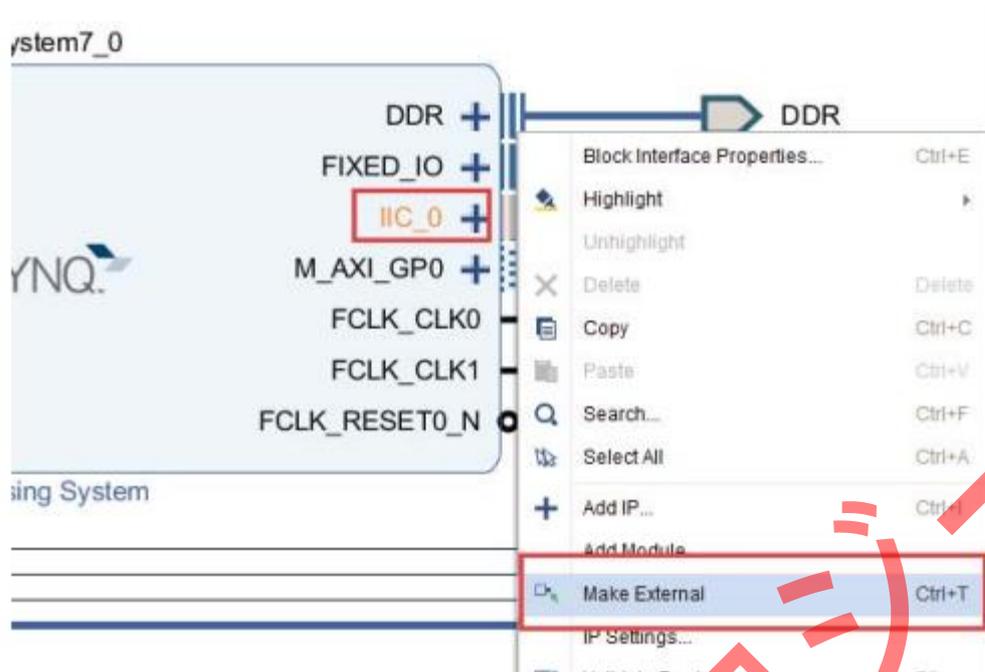


6) Run Block Automationを実行して、必要なポートエクスポートを完了する。

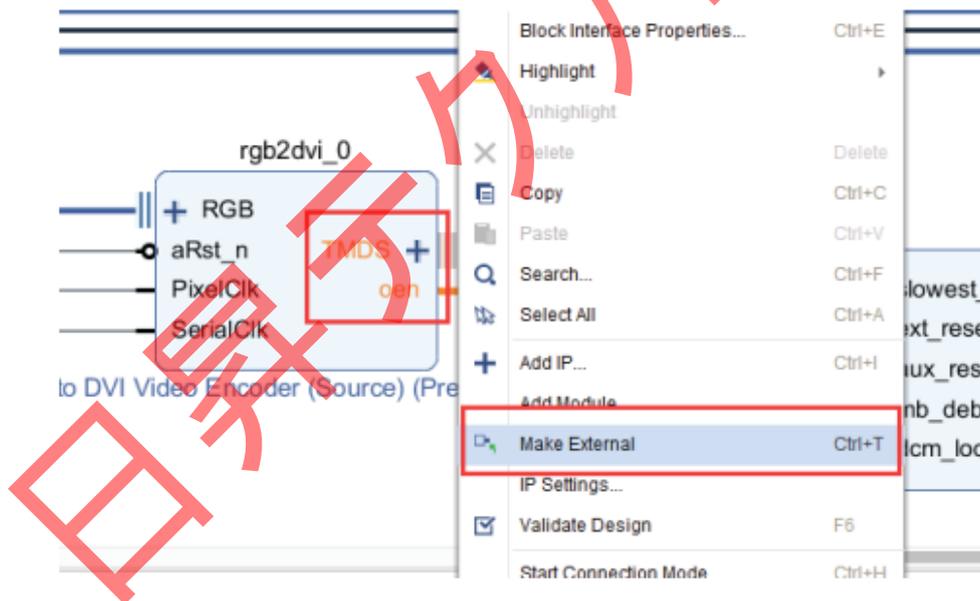


7) ポートを選択して、エクスポート。

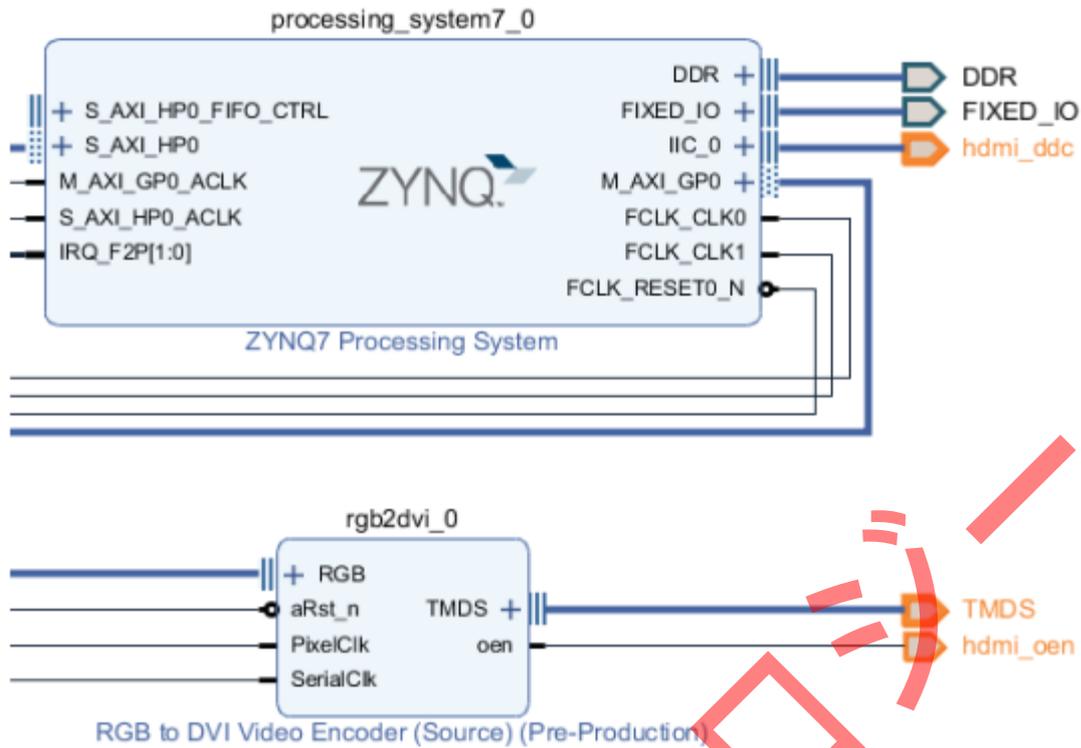
8) IIC\_0ポートをエクスポート



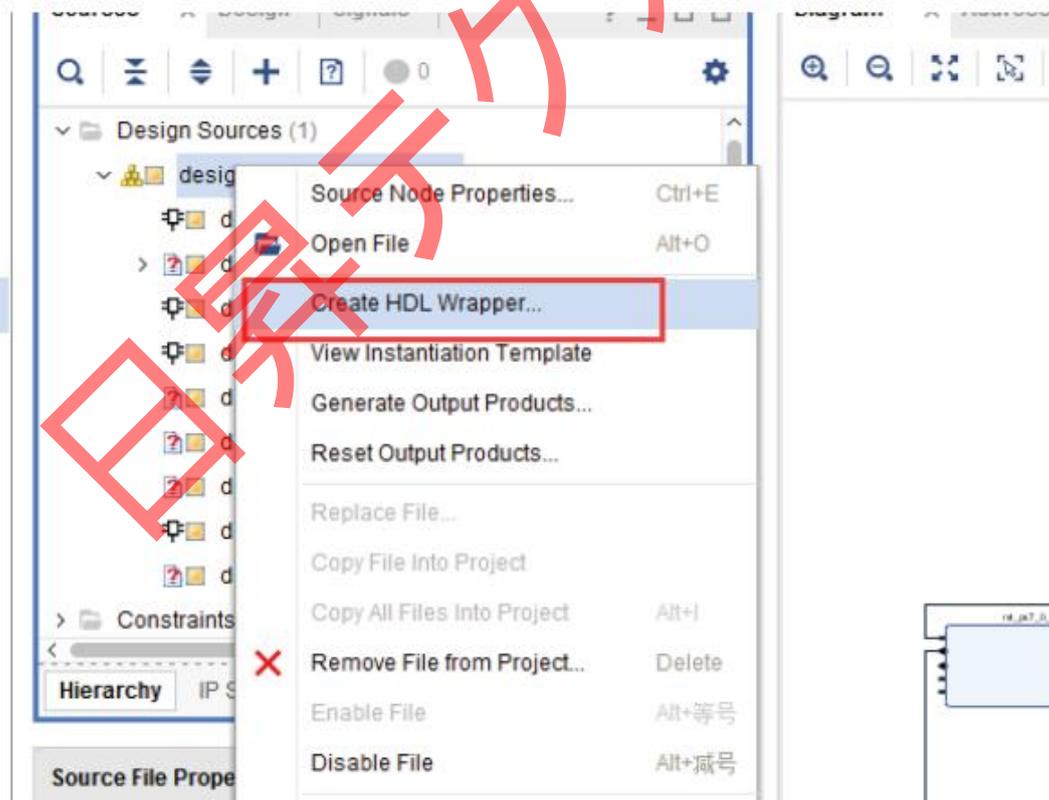
9) エンコーダーポートTMDSおよびOENのエクスポート



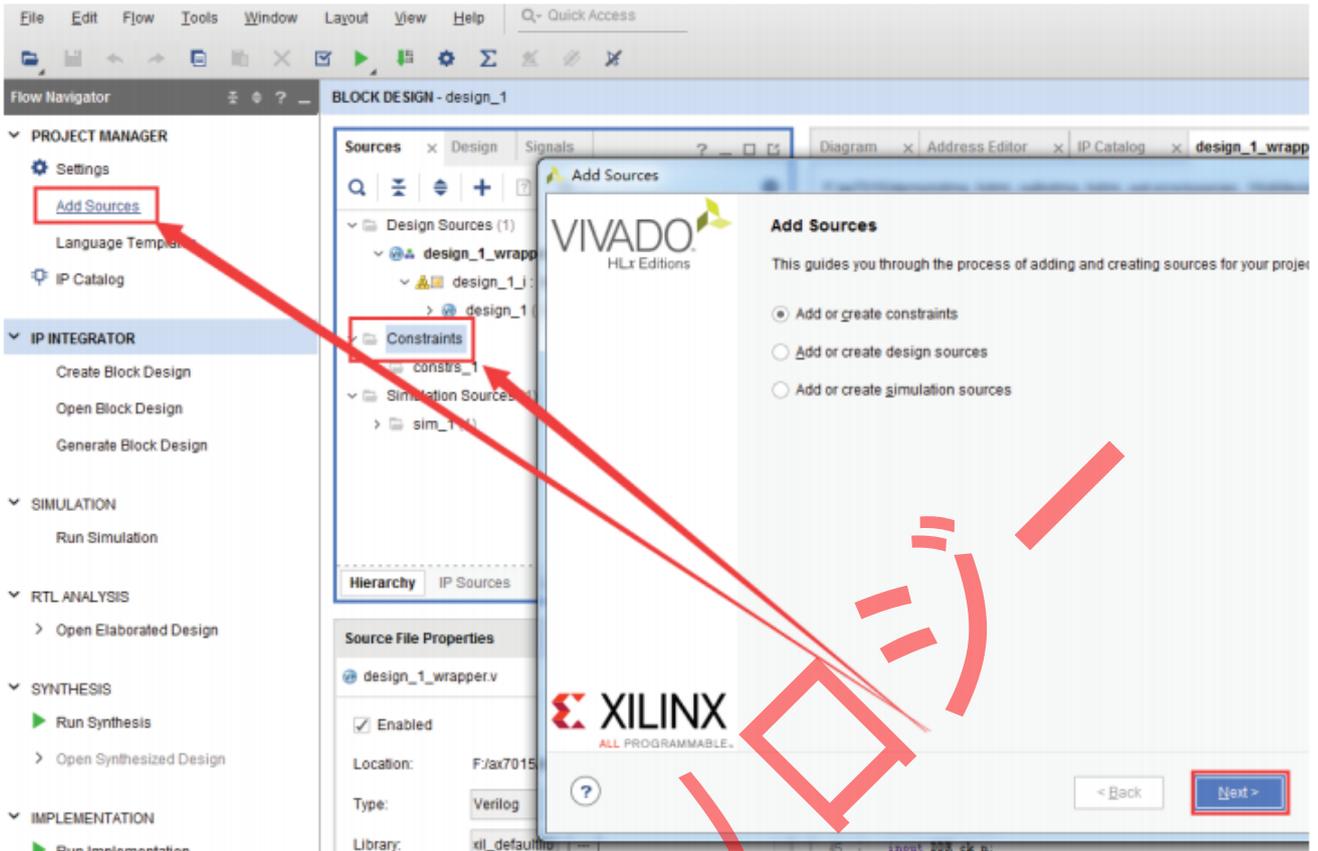
10) 別のポートの名前を変更する



11) 設計後、F6を押して設計を確認し、問題なかったらHDLファイルを作成する。



12) HDMI出力のxdcファイルを追加する。



13) xdcファイル内容は以下通りである。

```

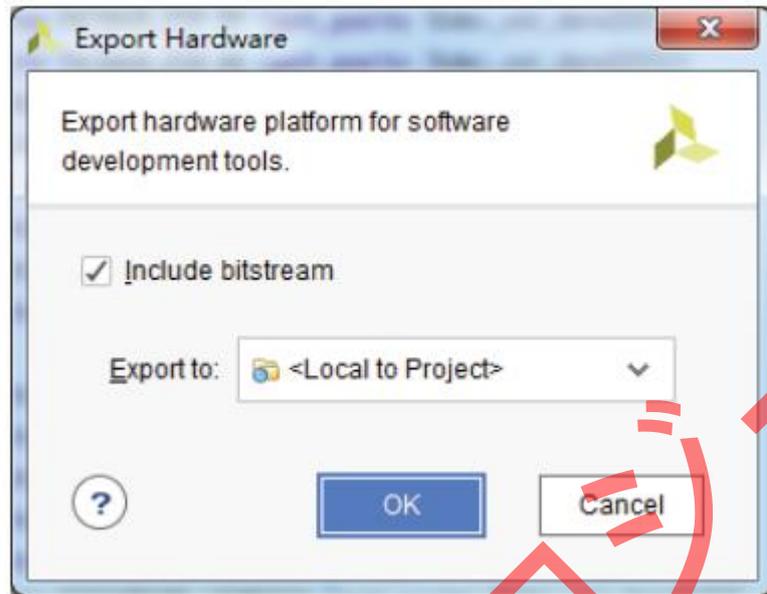
set_property IOSTANDARD TMDS_33 [get_ports TMDS_clk_n]
set_property PACKAGE_PIN N18 [get_ports TMDS_clk_p]
set_property IOSTANDARD TMDS_33 [get_ports TMDS_clk_p]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[0]}]
set_property PACKAGE_PIN V20 [get_ports {TMDS_data_p[0]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[0]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[1]}]
set_property PACKAGE_PIN T20 [get_ports {TMDS_data_p[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[1]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_n[2]}]
set_property PACKAGE_PIN N20 [get_ports {TMDS_data_p[2]}]
set_property IOSTANDARD TMDS_33 [get_ports {TMDS_data_p[2]}]
#set_property PACKAGE_PIN Y19 [get_ports {hdmi_hpd_tri_i[0]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {hdmi_hpd_tri_i[0]}]
set_property PACKAGE_PIN V16 [get_ports hdmi_oen]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_oen]
set_property PACKAGE_PIN R18 [get_ports hdmi_ddc_scl_io]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_ddc_scl_io]
set_property PACKAGE_PIN R16 [get_ports hdmi_ddc_sda_io]
set_property IOSTANDARD LVCMOS33 [get_ports hdmi_ddc_sda_io]

```

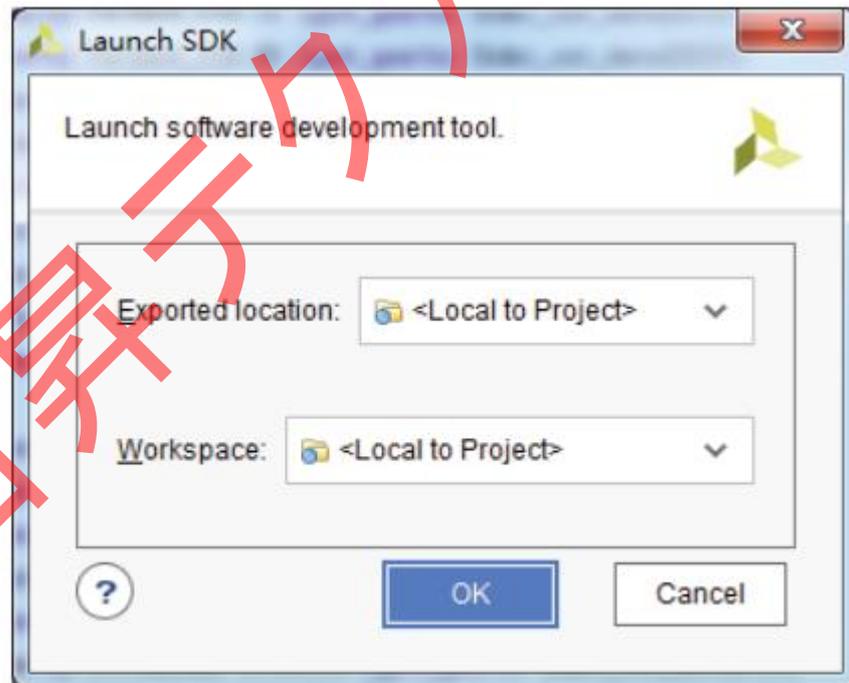
14) bitファイルのコンパイルと生成

## 12.2 SDK ソフトウェアの作成とデバッグ

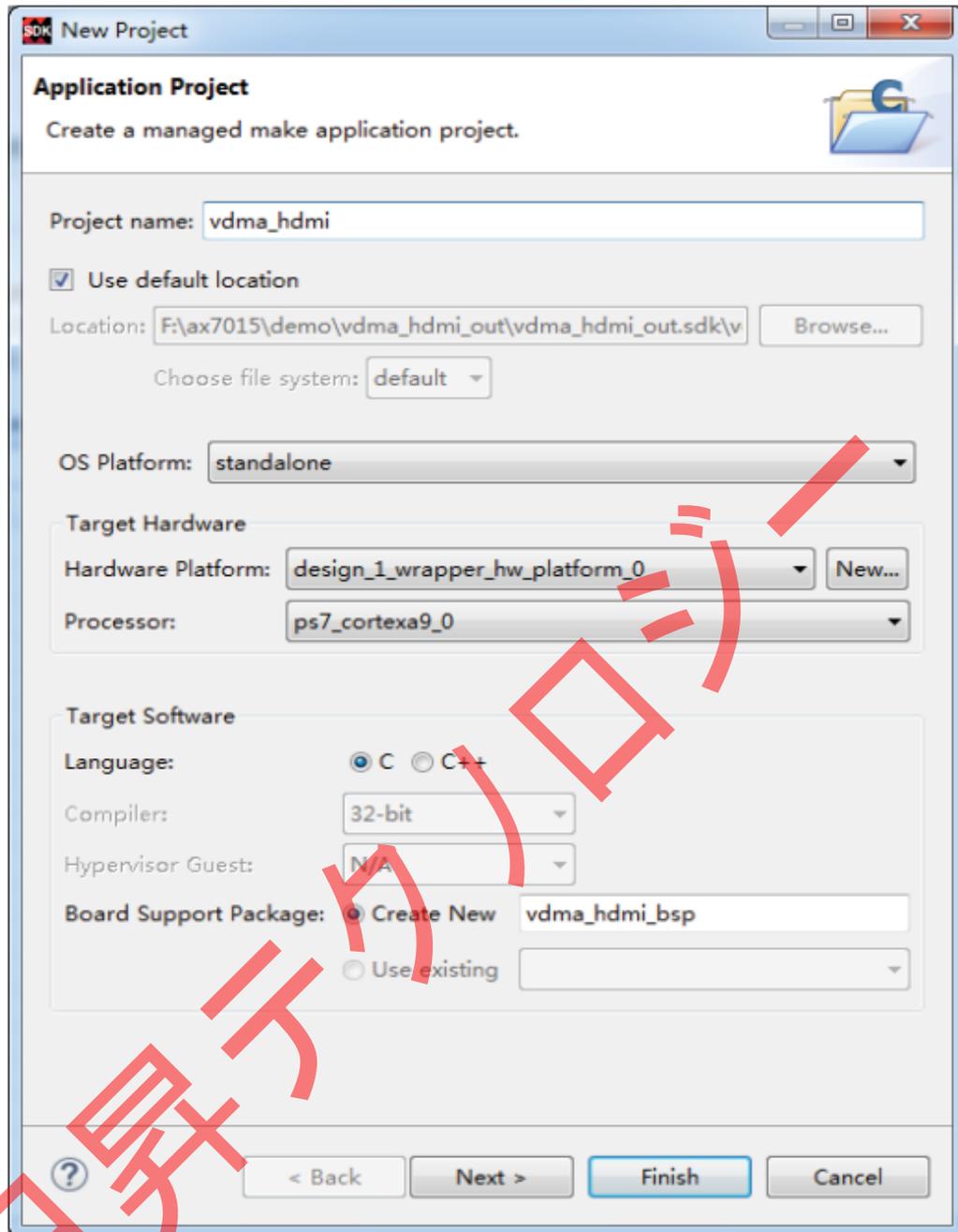
### 1) ハードウェアのエクスポート



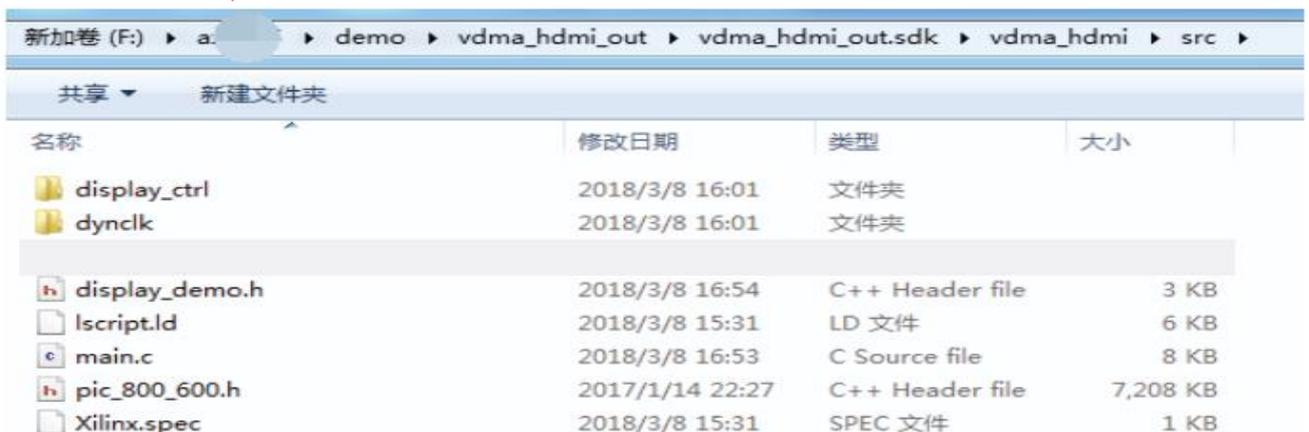
### 2) SDKを実行する



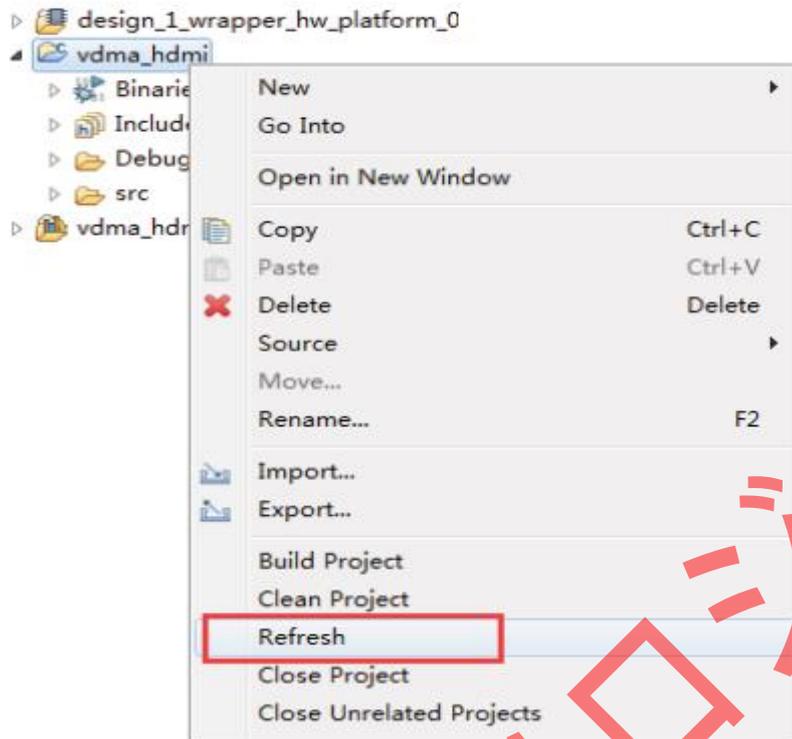
### 3) vdma\_hdmi というアプリを作成する



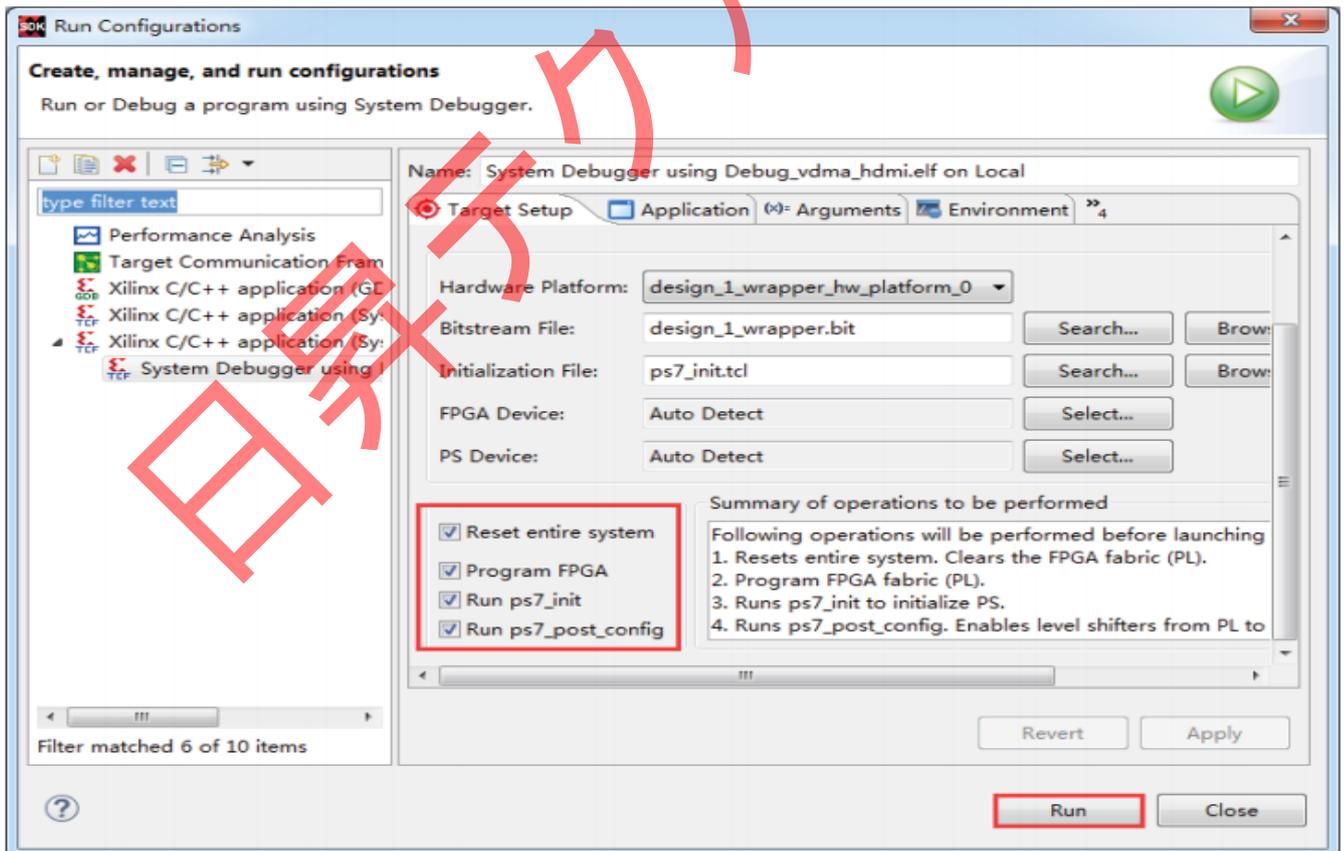
4) 多くのプログラムファイルがあるため、具体的に紹介しなく、直接にサンプルのソースコードをコピーする。srcディレクトリ内のファイルを削除し、代わりにサンプルのsrcディレクトリファイルを使用する。



5) SDKで更新する



6) HDMI出力ポートをディスプレイに接続し、コンパイルして実行する。



7) ディスプレイに画像が表示される



## 第十三章 プログラムの復帰

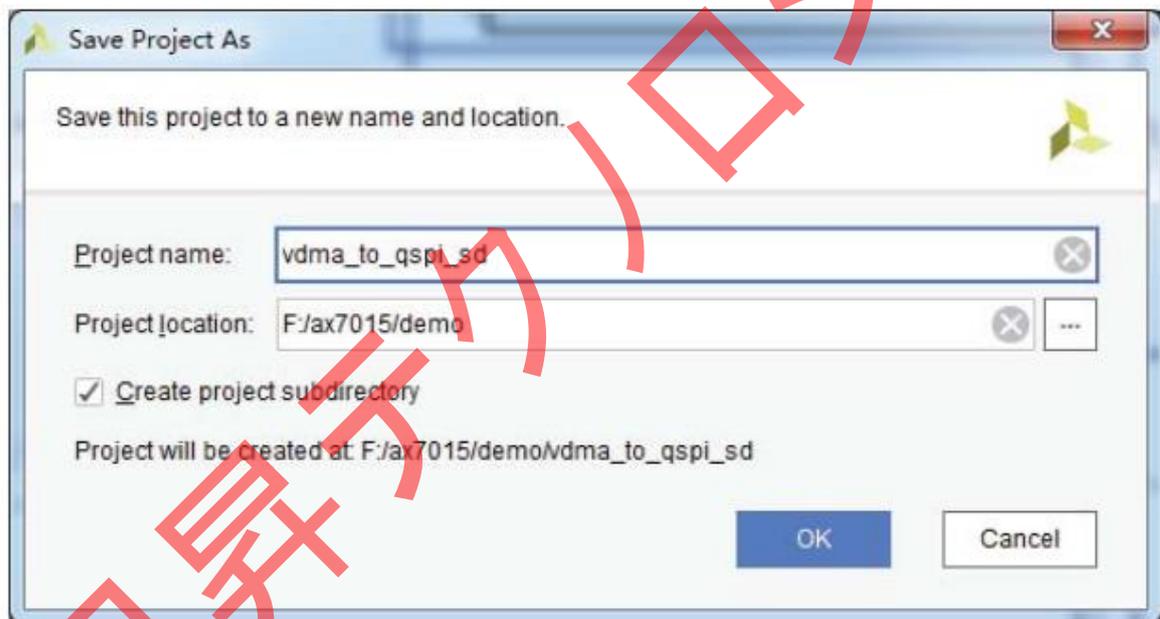
実験Vivadoプロジェクトはvdma\_to\_qspi\_sdである。

JTAGデバッグを通して、SDカードにプログラムを書き込む、またはQSPI FLASHに書き込む方法について、多くの実験を行った。まず、PLにはPSコンフィグが必要であるため、以前のFPDA書き込みのようにFlashに直接書き込むことはできない。この実験では、プログラムを復帰する方法について説明する。

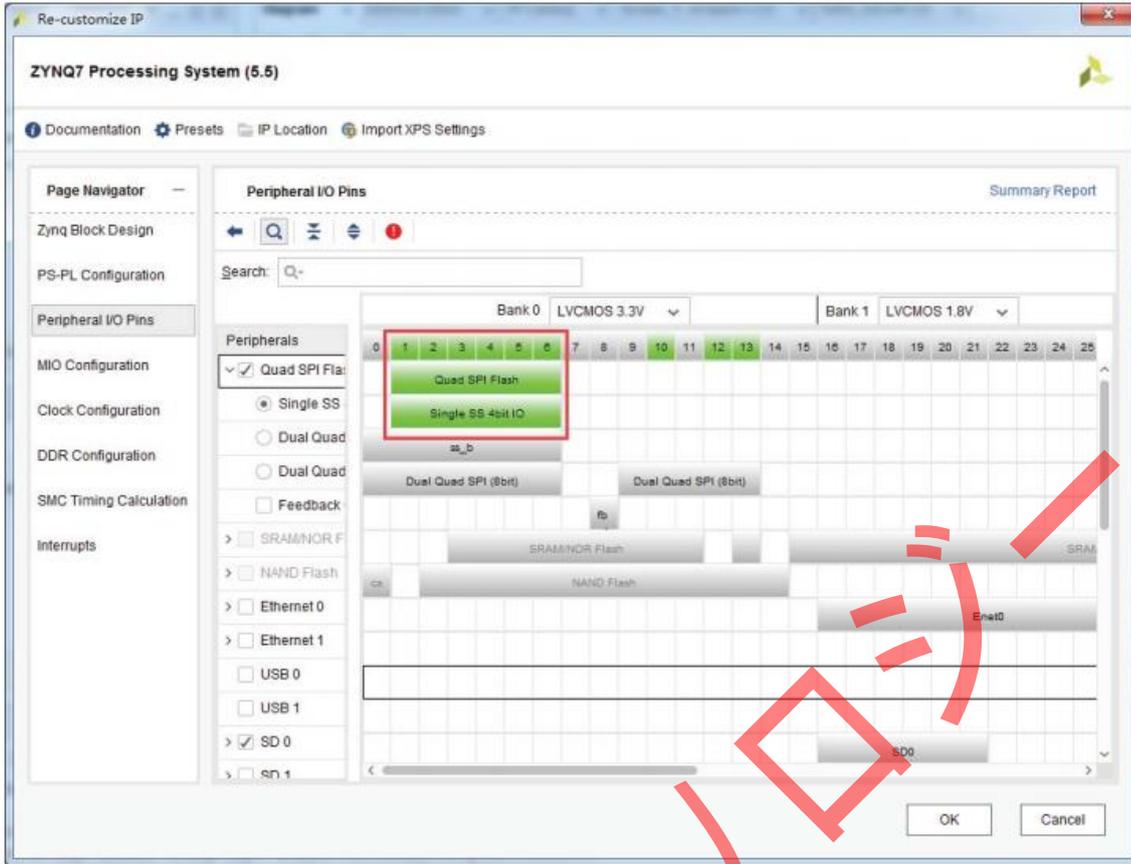
### 13.1 Vivado プロジェクトの設立

この実験では、VDMAテストエンジニアリングを選択して復帰したが、VDMAテストプロジェクトを設立する際、QSPIおよびSDカードを有効にできなかった。プログラムを復帰するには、QSPIまたはSDカードを有効にする必要がある。

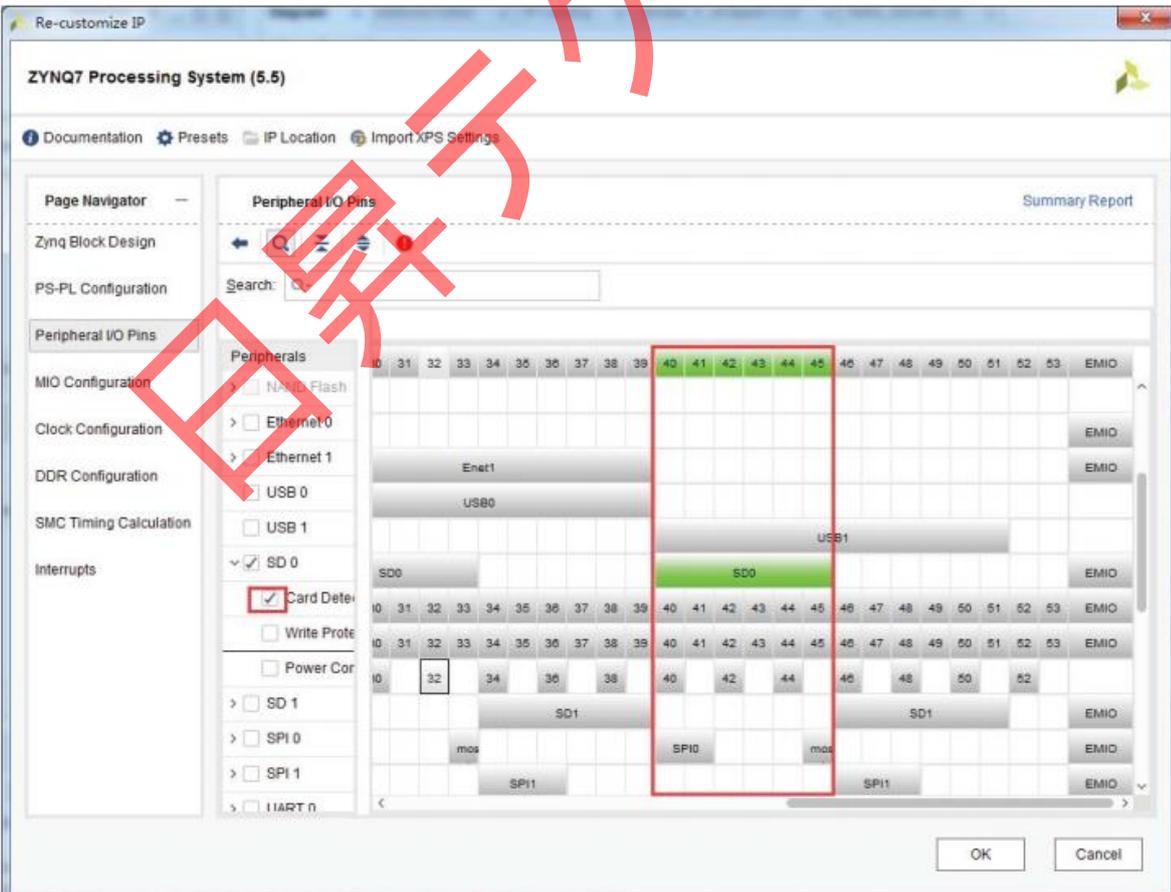
1) プログラムの復帰テストのため、VDMAテストプロジェクトのコピーを保存し、名前をvdma\_to\_qspi\_sdに変更する。



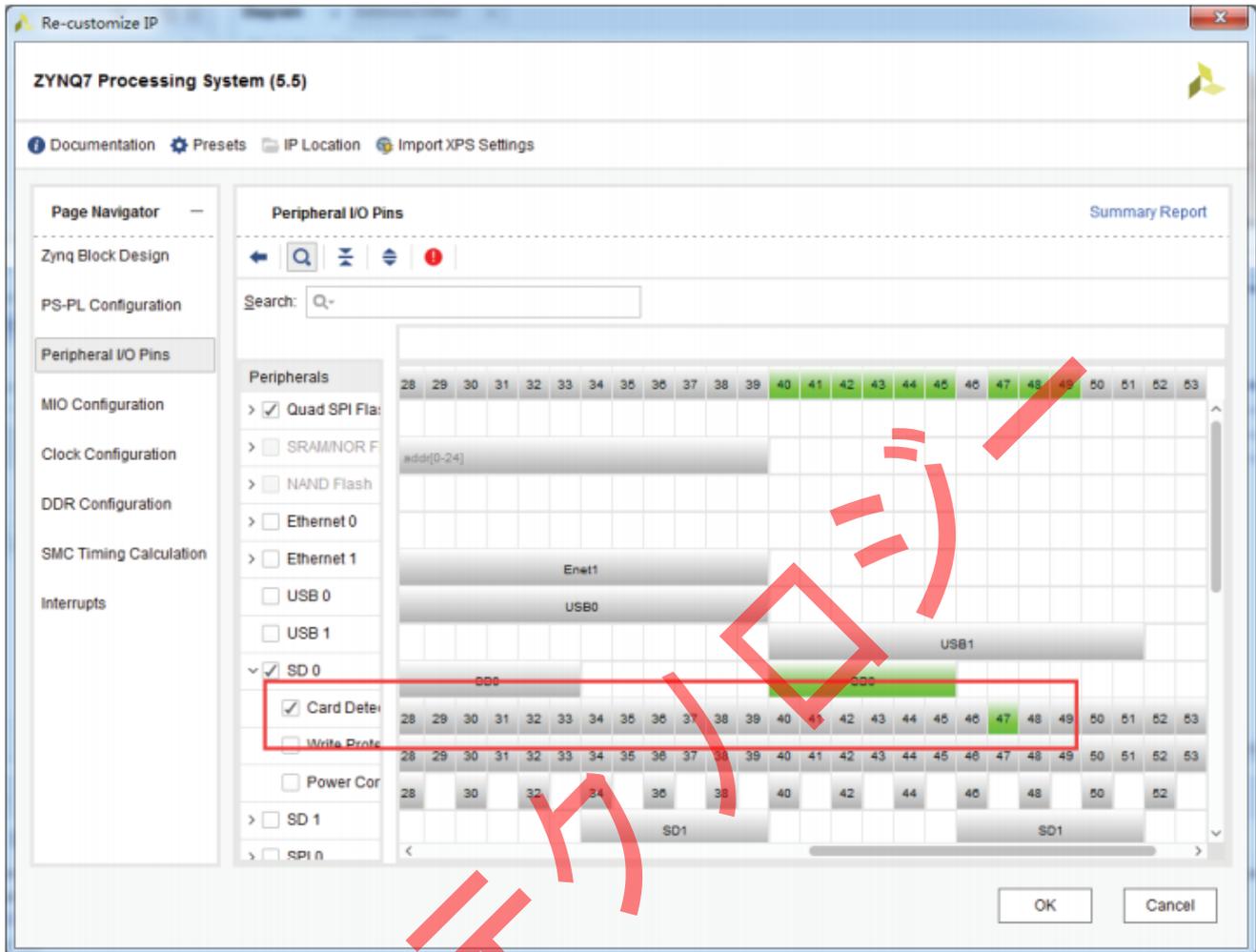
2) QSPIを追加し、MI01-6を使用する。



3) SD0コントローラーを追加、MI040-45及びTFカードインターフェイスを使用する



## 4) SDカードの検出ピンMI047を追加する

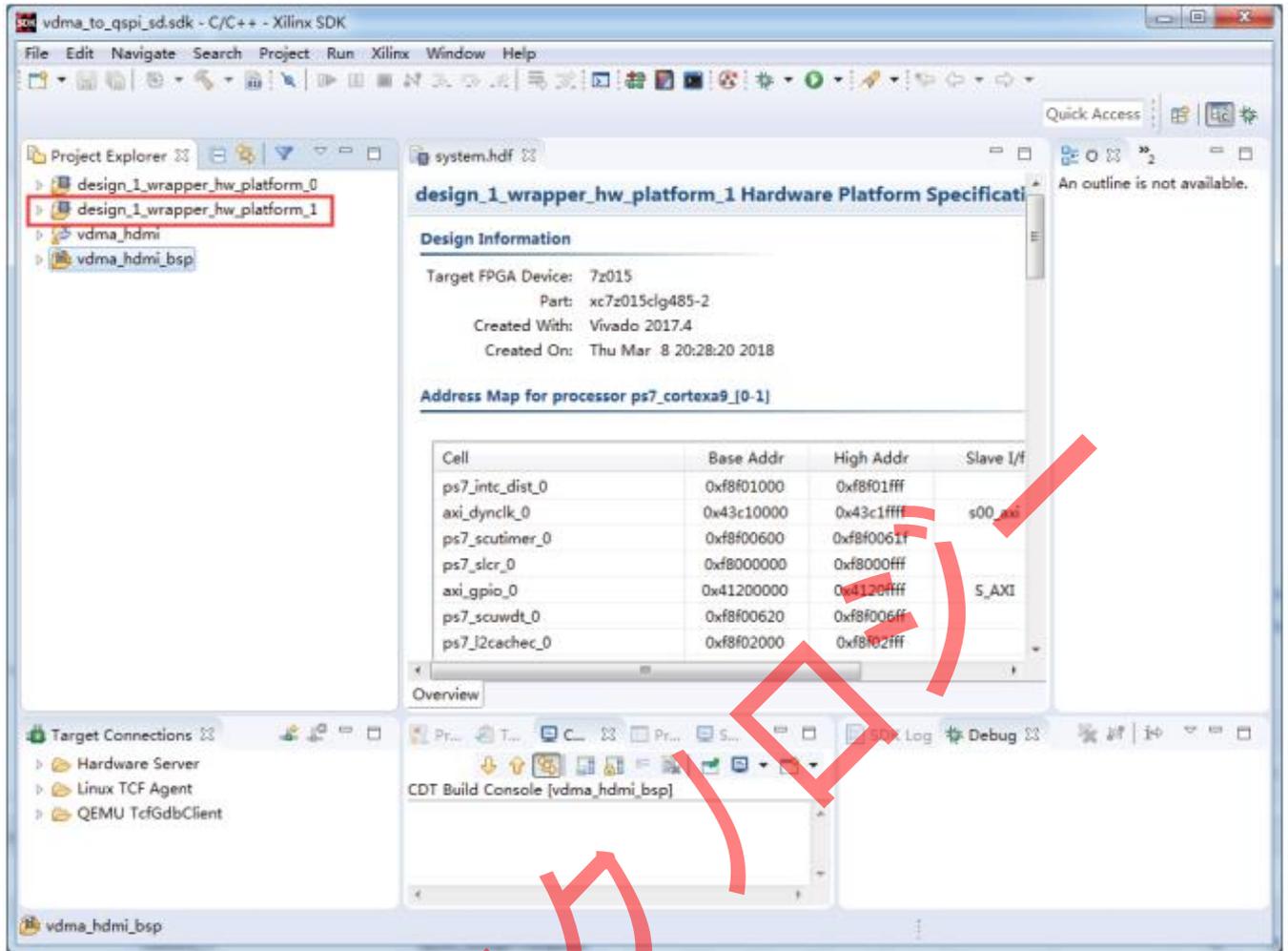


## 5) デザインを保存、bitファイルをコンパイルして生成し、ハードウェアを再度エクスポートする。

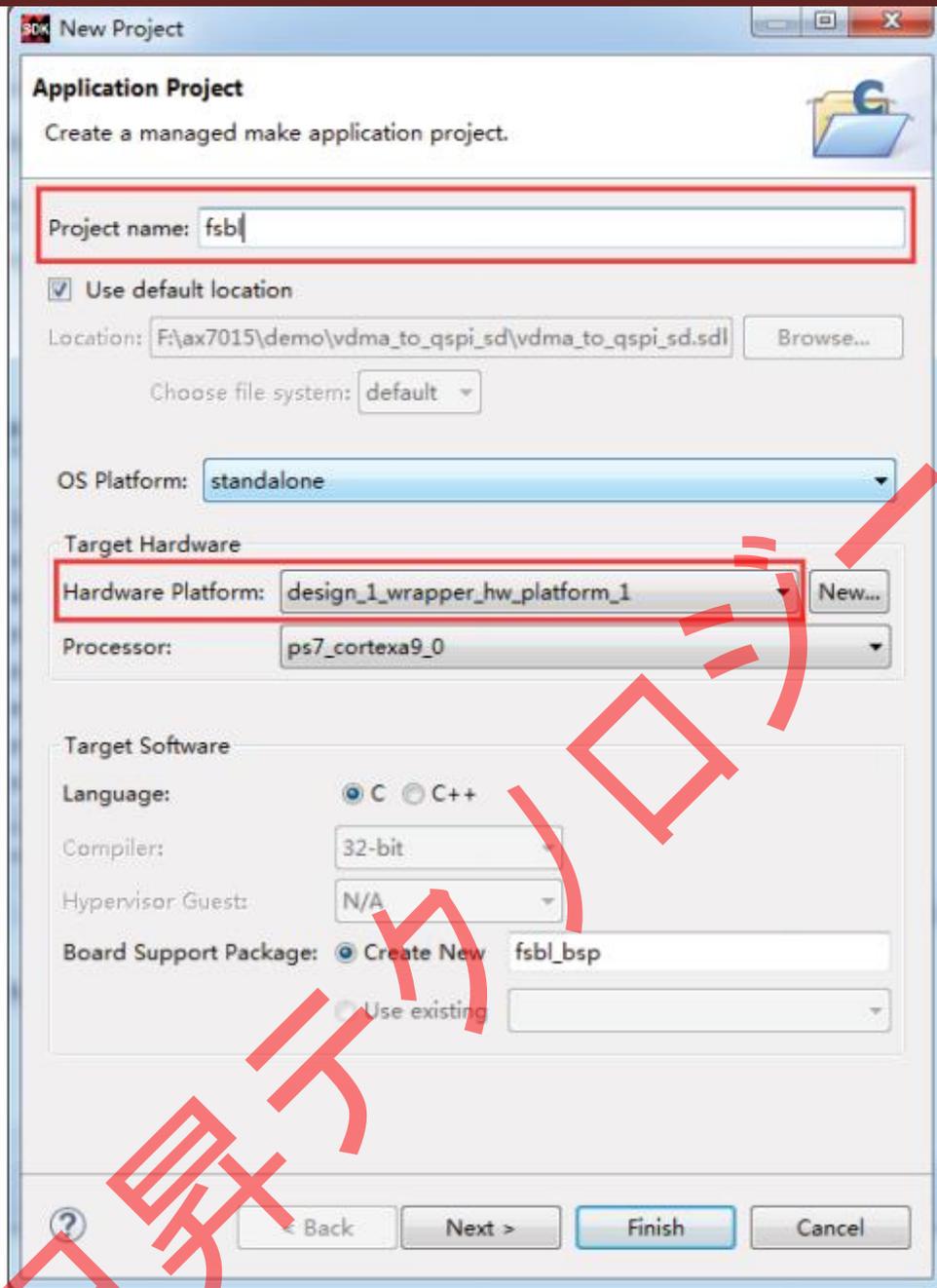
## 13.2 FSBL を生成する

FSBLは、MIOの割り当て、DDRコントローラーの初期化、SD、QSPIコントローラーの初期化を完了し、FPGAをコンフィグして、ユーザープログラムを読み込むセカンダリブートローダーである。

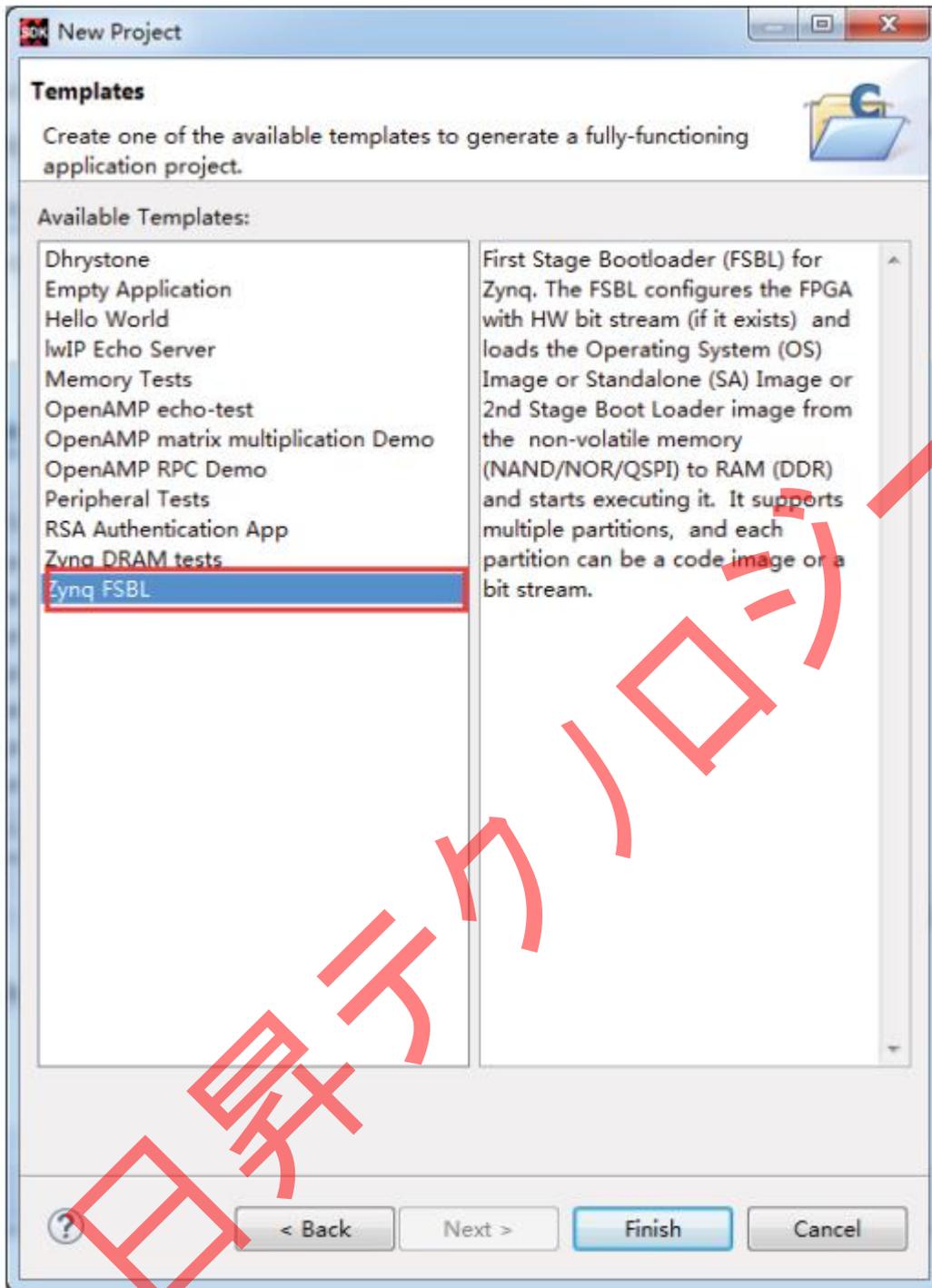
1) SDKソフトウェアを起動する。SDKソフトウェアは他のプロジェクトからコピーしてきたため、以前のSDKプロジェクトであり、パスの変更により、もう1つのhw\_platform\_1がある。



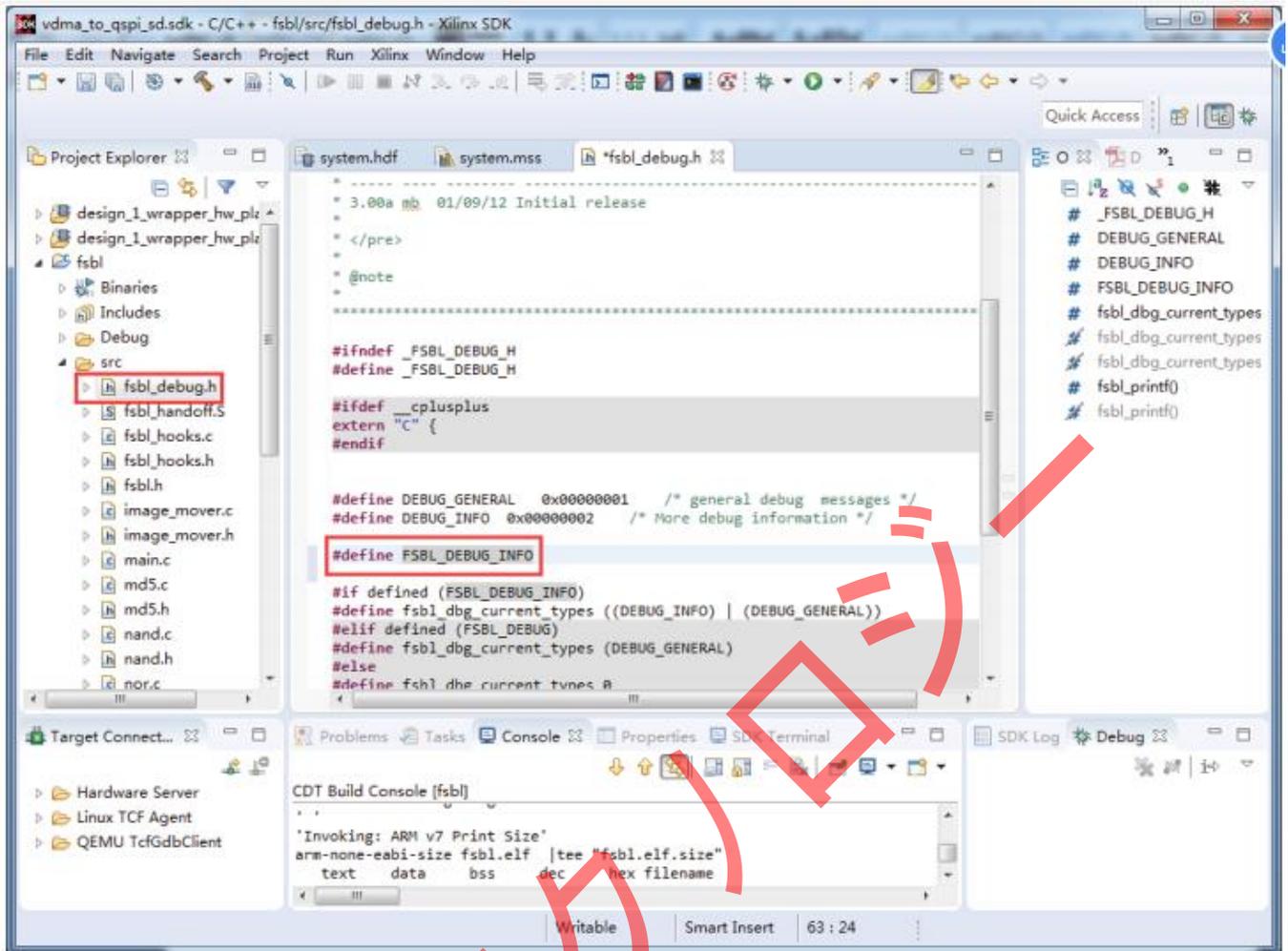
2) fsblという新しいアプリを作成し、ハードウェアプラットフォームに最新のプラットフォームを選択する。



3) テンプレートはZynq FSBLを選択する。



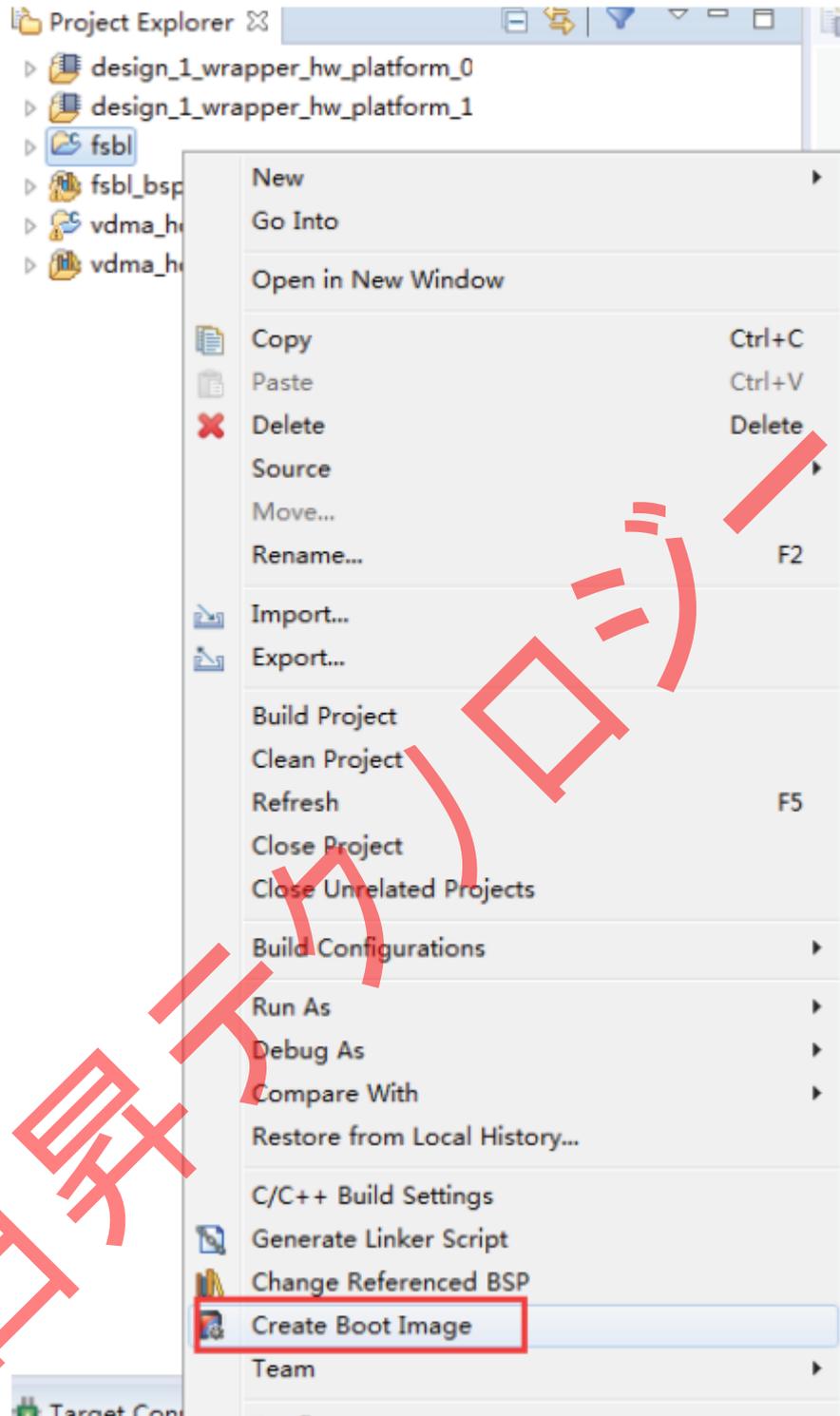
4) デバッグマクロ定義FSBL\_DEBUG\_INFOを追加すると、FSBLの出カステータス情報を開始できる。これはデバッグには役立つが、起動時間が長くなる。



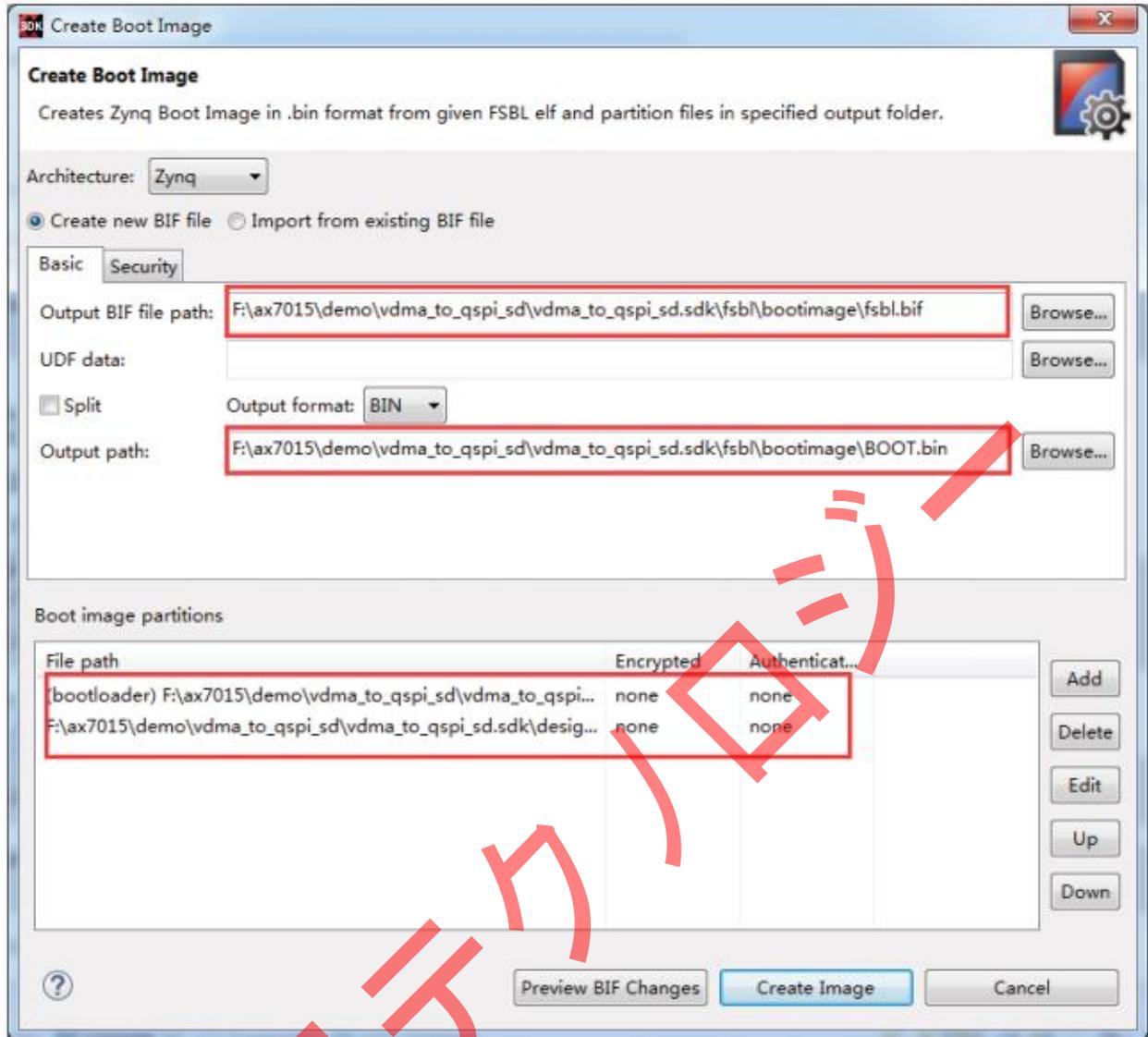
5) SDKはデフォルトで自動的にコンパイルされ、fsbl.elfファイルが生成される。

### 13.3 BOOT ファイルを作成する

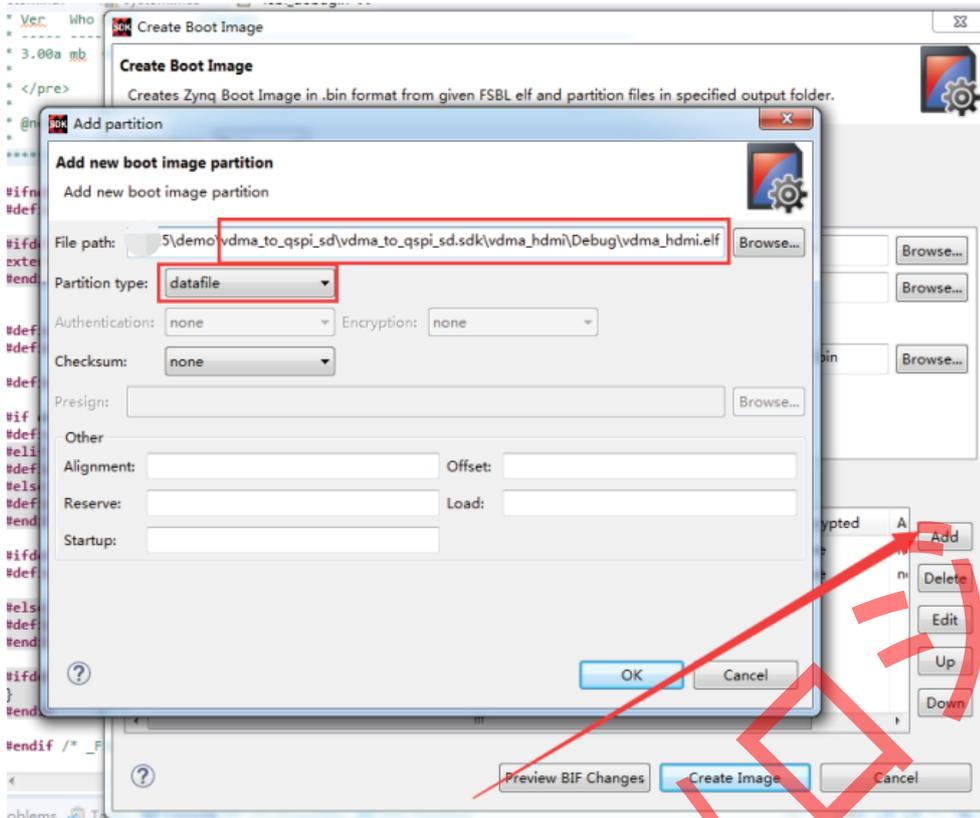
- 1) fsblプロジェクトを選択し、Create Boot Imageを選択する。



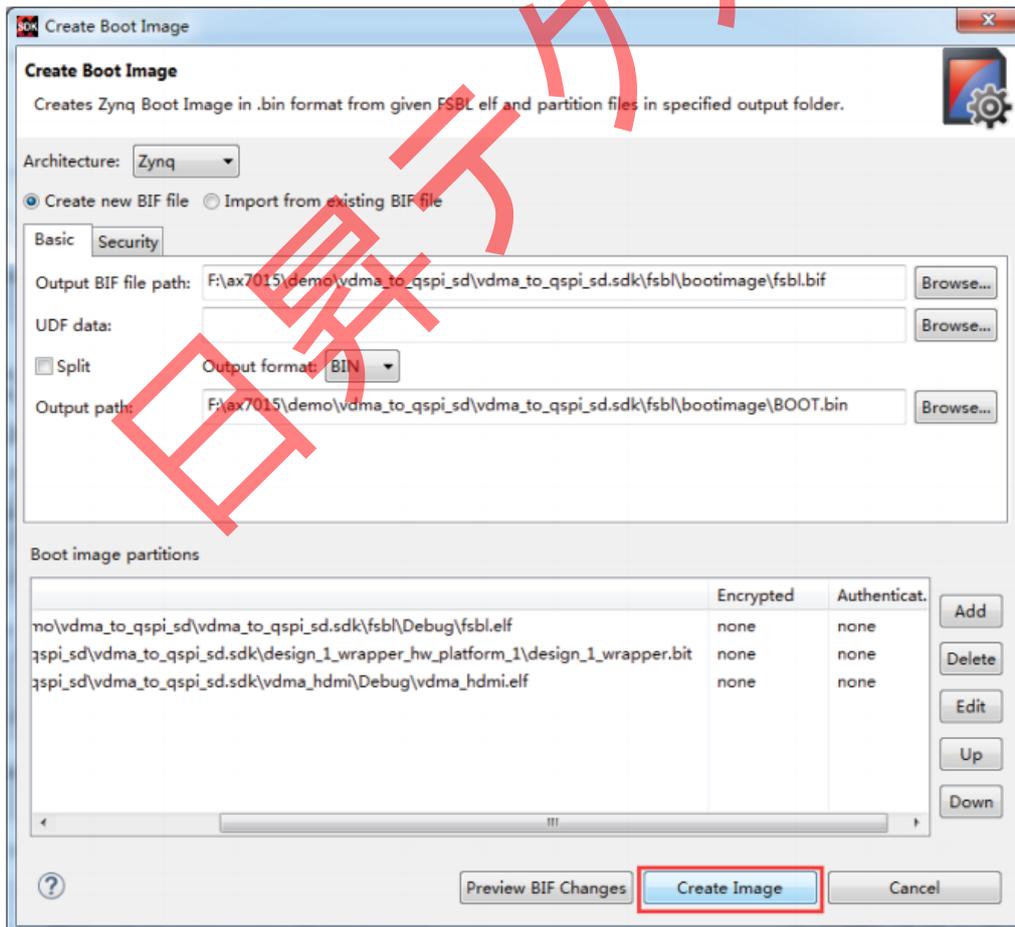
2) 生成されたBIFファイルパスは、ポップアップウィンドウに表示される。BIFファイルは、BOOTファイルを生成するためのコンフィグファイルであり、生成されたBOOT.binファイルパスである。BOOT.binファイルは、必要なスタートアップファイルであり、SDカードに入れることができ、QSPI Flashにも書き込み可能である。



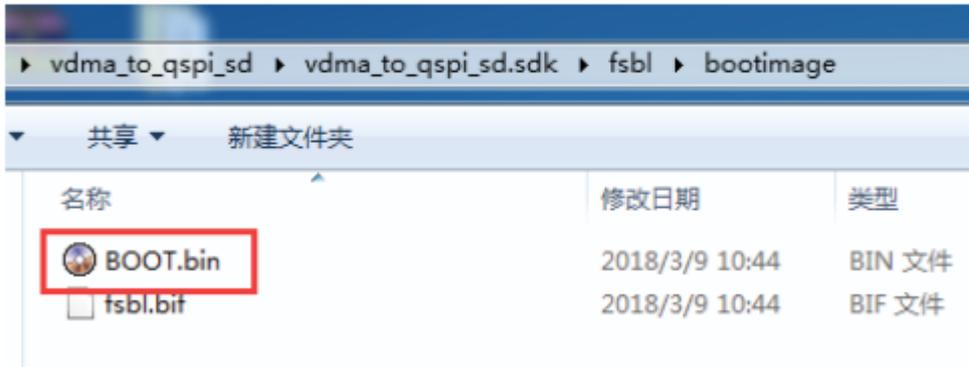
3) Boot image partitionsリストに合成するファイルがあ。最初のファイルはboot loaderファイルで、上記で生成されたfsbl.elfファイルである。2番目のファイルはFPGAコンフィギュレーションファイルである。[Add]をクリックして、VDMAテストプログラムvdma\_hdmi .elfを追加する。



4) [Create Image]をクリックして生成する。

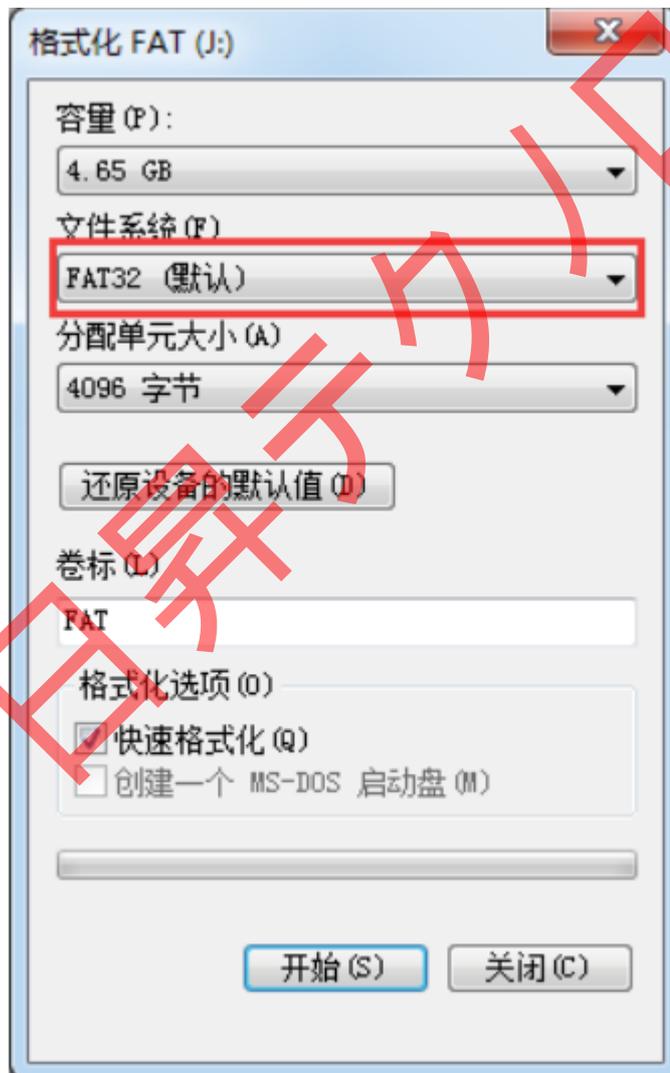


- 5) BOOT.binファイルは、生成されたディレクトリにある。



### 13.4 SD カードの起動テスト

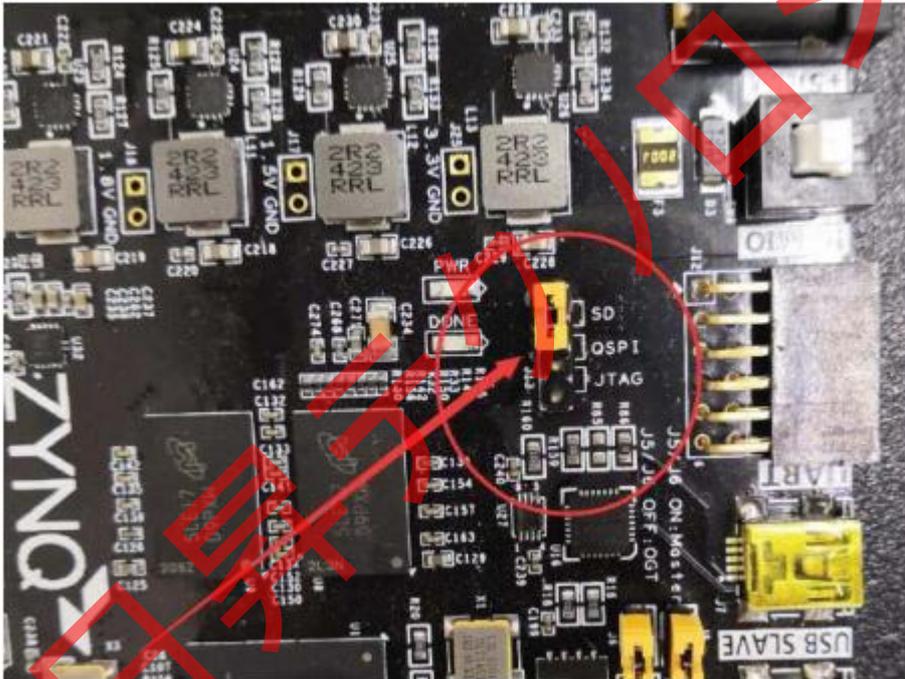
- 1) SDカードをフォーマットし、FAT32にフォーマットする。他のフォーマットは開始できない。



- 2) BOOT.binファイルをルートディレクトリに入れる。



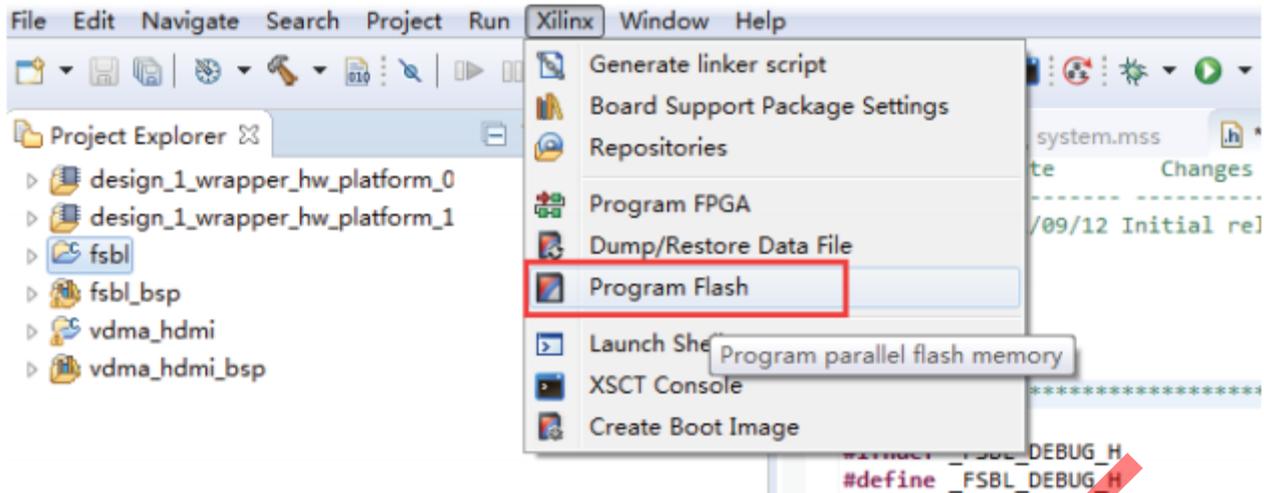
- 3) SDカードを開発ボードのSDカードスロットに挿入する。
- 4) 開始モードをSDカードブートに調整



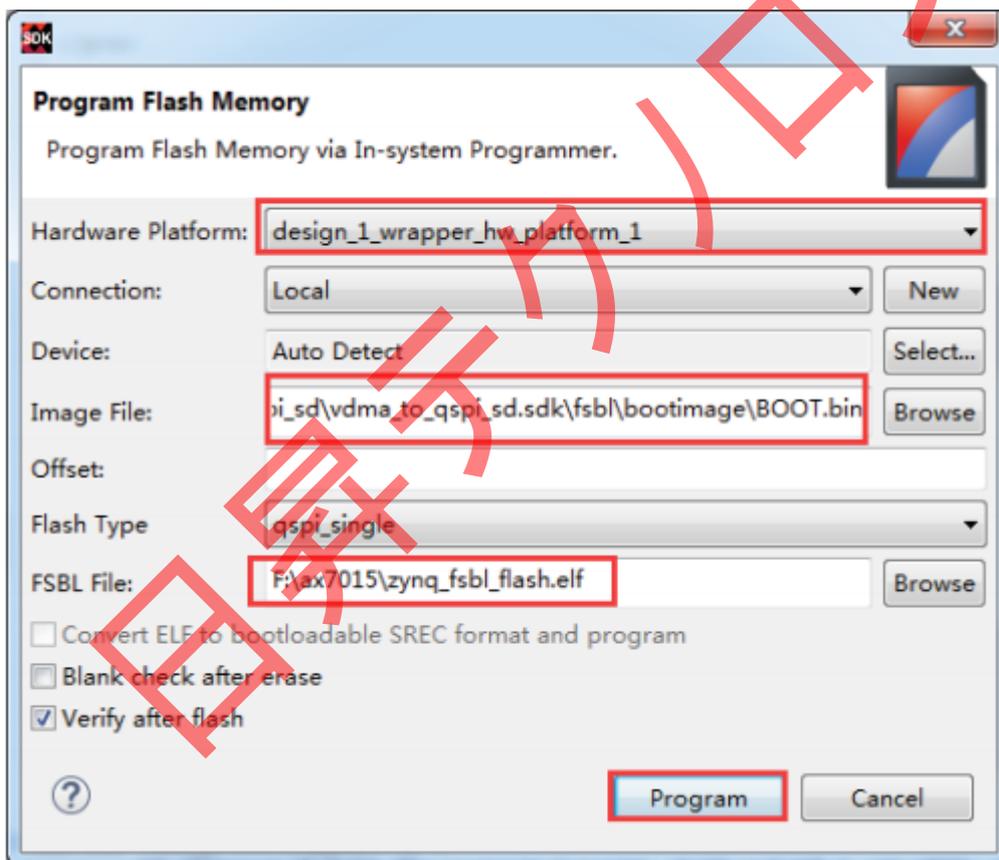
- 5) HDMIディスプレイを接続し、ボードの電源を入れると、ディスプレイに子猫の写真が表示される。

### 13.5 QSPI テスト開始

- 1) SDKメニューの[Xilinx]-> [Program Flash]



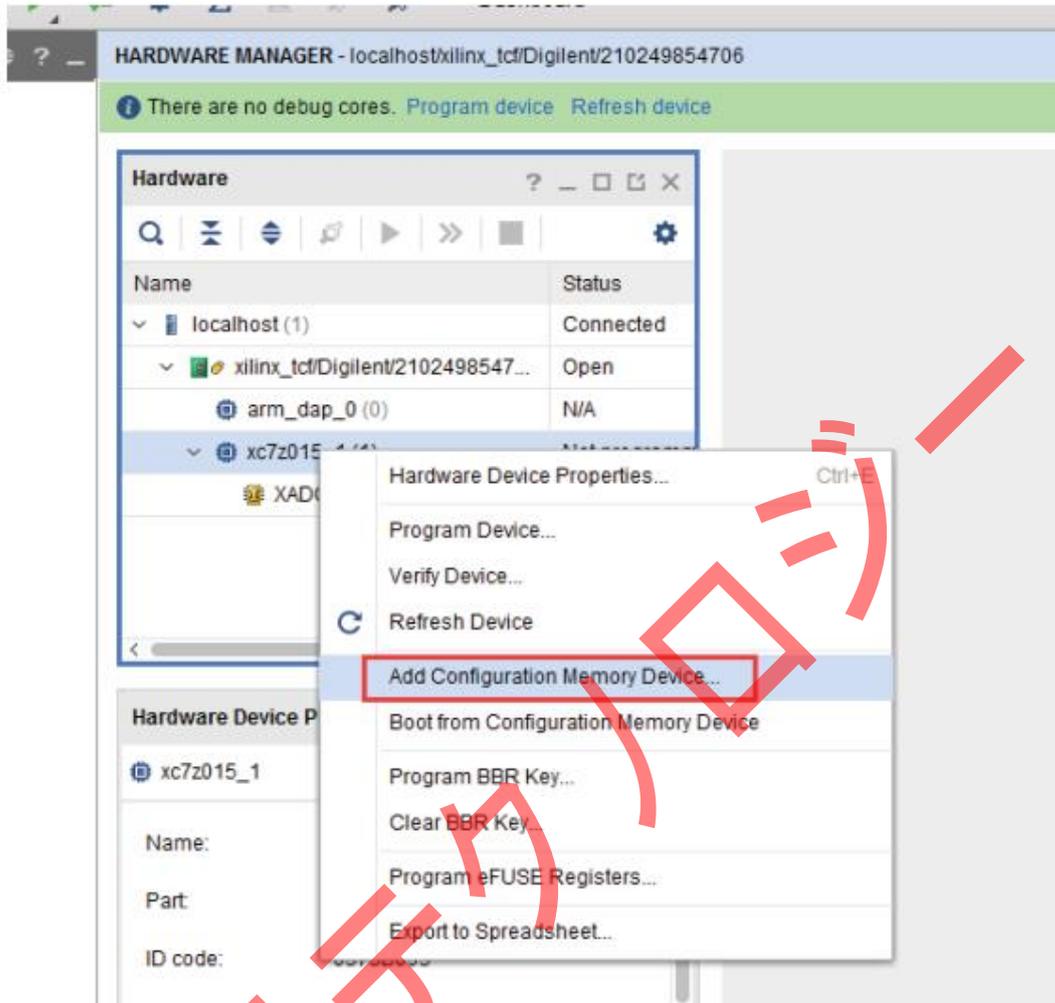
- 2) Hardware Platformは最新のを選択、Image Fileファイルは書き込んだBOOT.binを選択、FSBL fileはコア電子カスタムの特別バージョンfsbl.elfを選択し、このfsblしか書き込めない



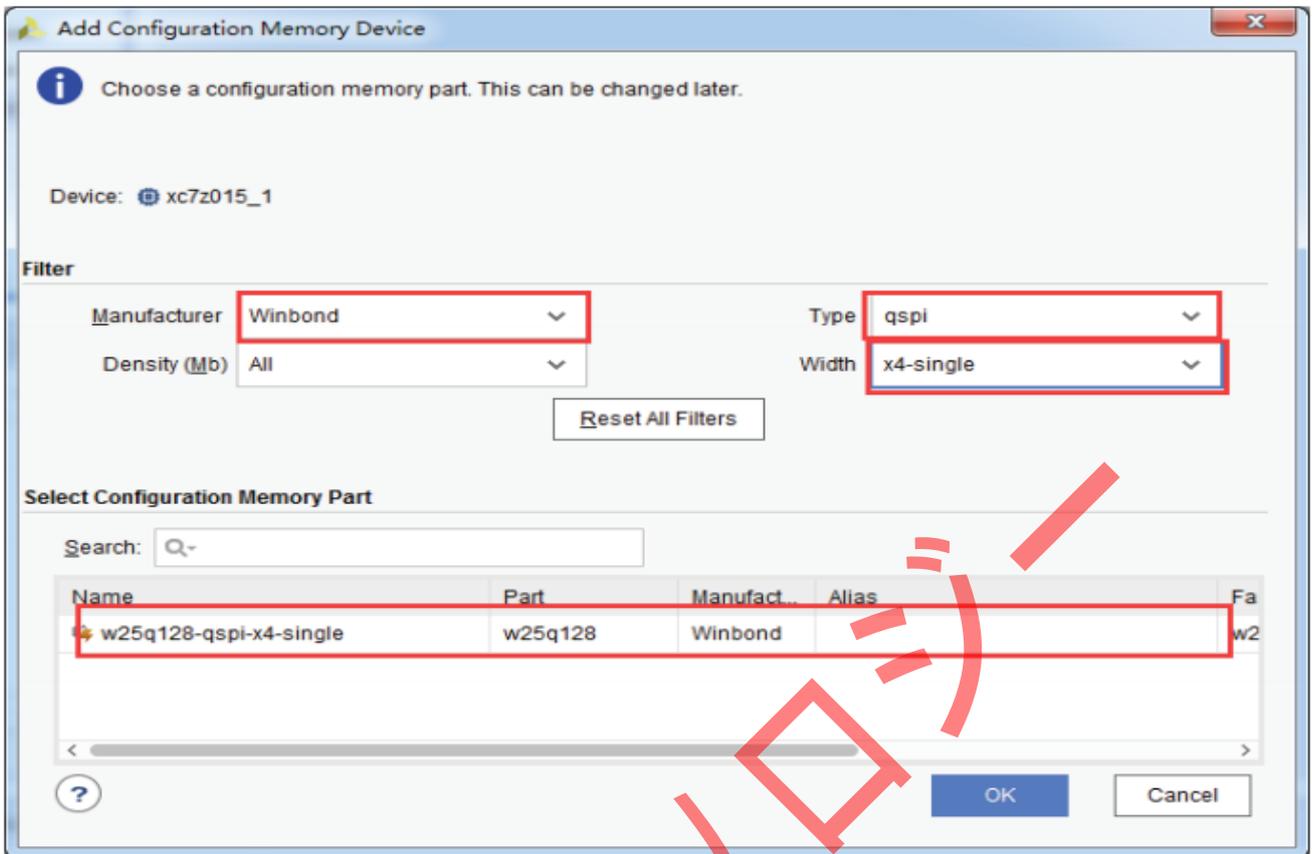
- 3) [Program]をクリックして、プログラミングが完了するのを待つ。
- 4) ブートモードをQSPIに設定して、もう一度起動すると、ディスプレイにディスプレイ出力が出てくる。

### 13.6 Vivado のもとで QSPI を書き込む

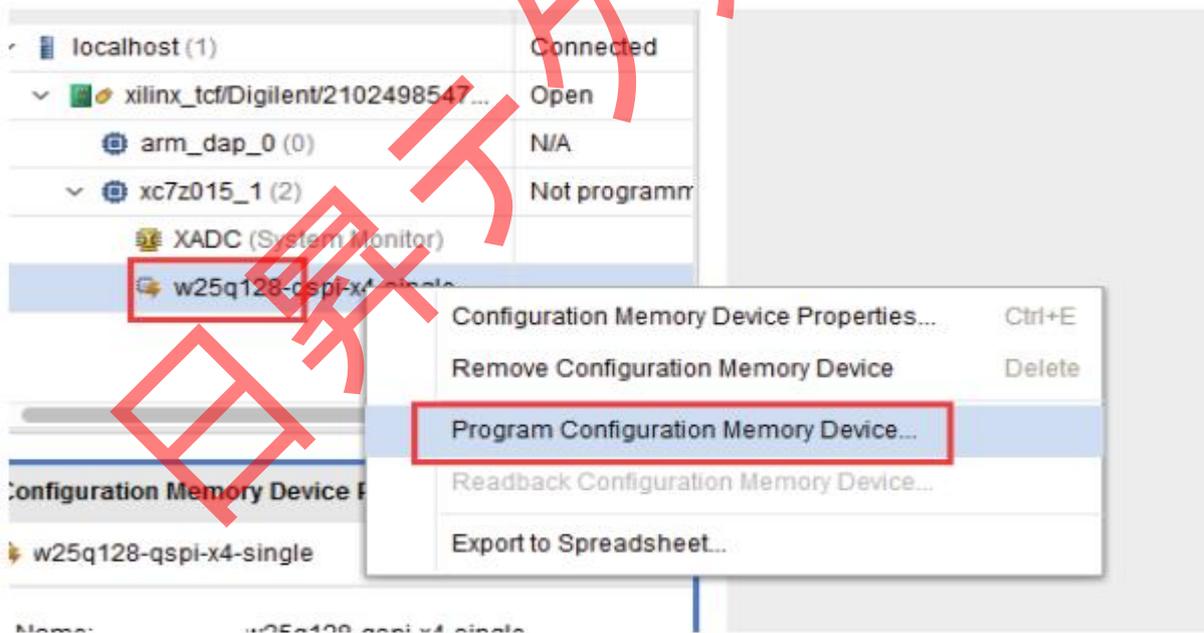
- 1) HARDWARE MANGERでデバイスを選択し、[Add Configuration Memory Device]をクリックする。



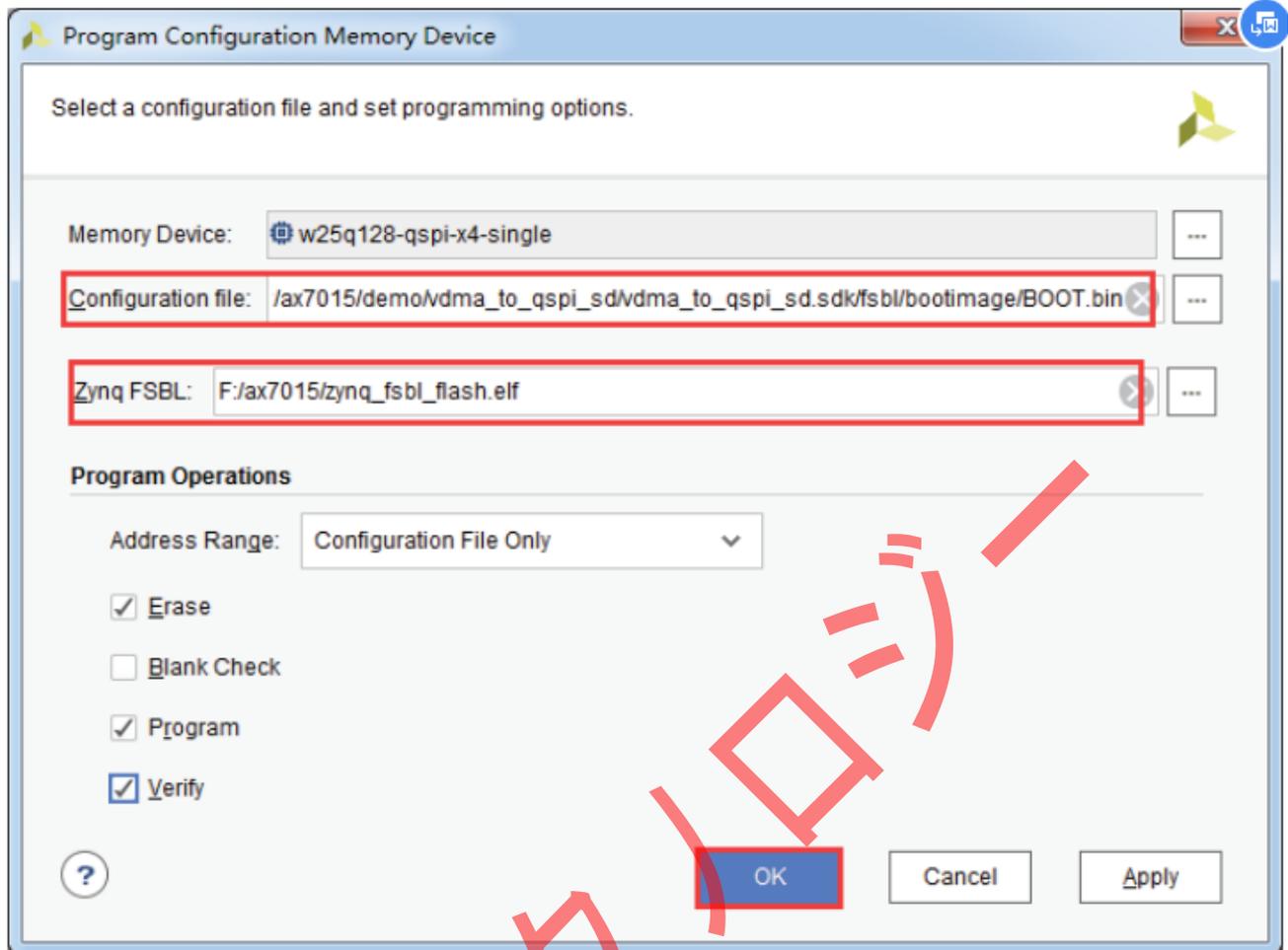
- 2) Winbondを選択し、タイプをqspi、幅をx4-single入力すると、w25q128が表示され、開発ボードはw25q256を使用、書き込みには影響がない。



3) 右クリックしてプログラミングファイルを選択する。



4) 書き込みたいファイルとコア電子がカスタムのfsblファイルを選択すると、書き込みができる。書き込む時、JTAGブートモードじゃないと、警告が出る。QSPIを書き込むときにJTAGブートモードを設定することをお勧めする。



### 13.7 バッチファイルを使用して QSPI をすばやく書き込む

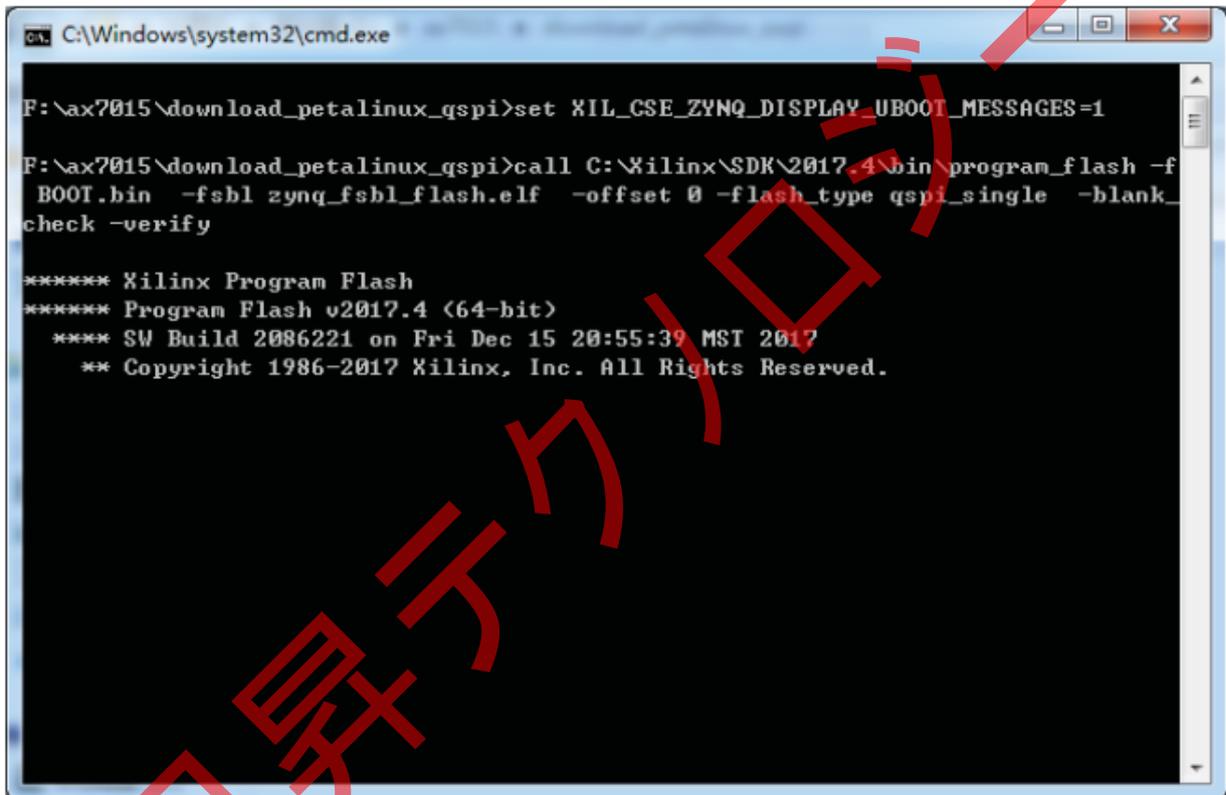
1) 新しいprogram\_qlspi.txtテキストファイルを作成し、拡張名はbatに変更、コンテンツは次のように入力する。Set XIL\_CSE\_ZYNQ\_DISPLAY\_UBOOT\_MESSAGES = 1はプログラミングプロセス中にuboot印刷情報を設定する。C:\Xilinx\SDK\2017.4\bin\program\_flashはツールパス、インストールパスに従って変更する。-fは書き込むファイル、-fsblは書き込まれるfsblweファイル（コア電子特定のファイル）、-blank\_check -verifyはチェックオプションである。

```
set XIL_CSE_ZYNQ_DISPLAY_UBOOT_MESSAGES=1
call C:\Xilinx\SDK\2017.4\bin\program_flash -f BOOT.bin -fsbl
zynq_fsbl_flash.elf -offset 0 -flash_type qlspi_single -
blank_check -verify
pause
```

2) 書き込むBOOT.bin、fsbl、batファイルをまとめる

名称	修改日期	类型	大小
 BOOT.BIN	2018/3/27 20:25	BIN 文件	14,879 KB
 program_qspi.bat	2018/3/22 11:04	Windows 批处理...	1 KB
 zynq_fsbl_flash.elf	2018/1/5 22:04	ELF 文件	181 KB

- 3) JTAGケーブルを差し込んで電源を入れ、batファイルをダブルクリックしてflashを書き込める。



```

C:\Windows\system32\cmd.exe
F:\ax7015\download_petalinux_qspi>set XIL_CSE_ZYNQ_DISPLAY_UBOOT_MESSAGES=1
F:\ax7015\download_petalinux_qspi>call C:\Xilinx\SDK\2017.4\bin\program_flash -f
BOOT.bin -fsbl zynq_fsbl_flash.elf -offset 0 -flash_type qspi_single -blank-
check -verify

***** Xilinx Program Flash
***** Program Flash v2017.4 (64-bit)
***** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
***** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
  
```

## 第十四章 仮想マシンと Ubuntu システムをインストールする

この後のマニュアルには、組み込みLinux開発が含まれ、通常、u-bootまたはLinux-kernelをコンパイルするにはLinuxオペレーティングシステムホストが必要である。Windowsオペレーティングシステムに仮想マシンをインストールしてから、仮想マシンにLinuxオペレーティングシステムをインストールするのは一番簡単な方法である。

### 14.1 仮想マシンソフトウェアのインストール

私たちが提供する仮想マシンのインストールソフトウェアバージョンはVMware-workstation-full12.1.1で、ユーザーは提供された情報でそれを見つける。VMware-workstation-full12.1.1-3770994.exeをダブルクリックすると、インストールが開始される。比較的簡単であるため、インストール手順は紹介せず、ユーザーは[Next]ボタンをクリックするだけでインストールできる。完了後のインターフェースに、VMware12のシリアル番号を入力するためにライセンスを選択する必要がある。インストール完了後、デスクトップにVMware Workstation Proのアイコンが表示される。



### 14.2 Ubuntu のインストール

#### 14.2.1 システムのインストール

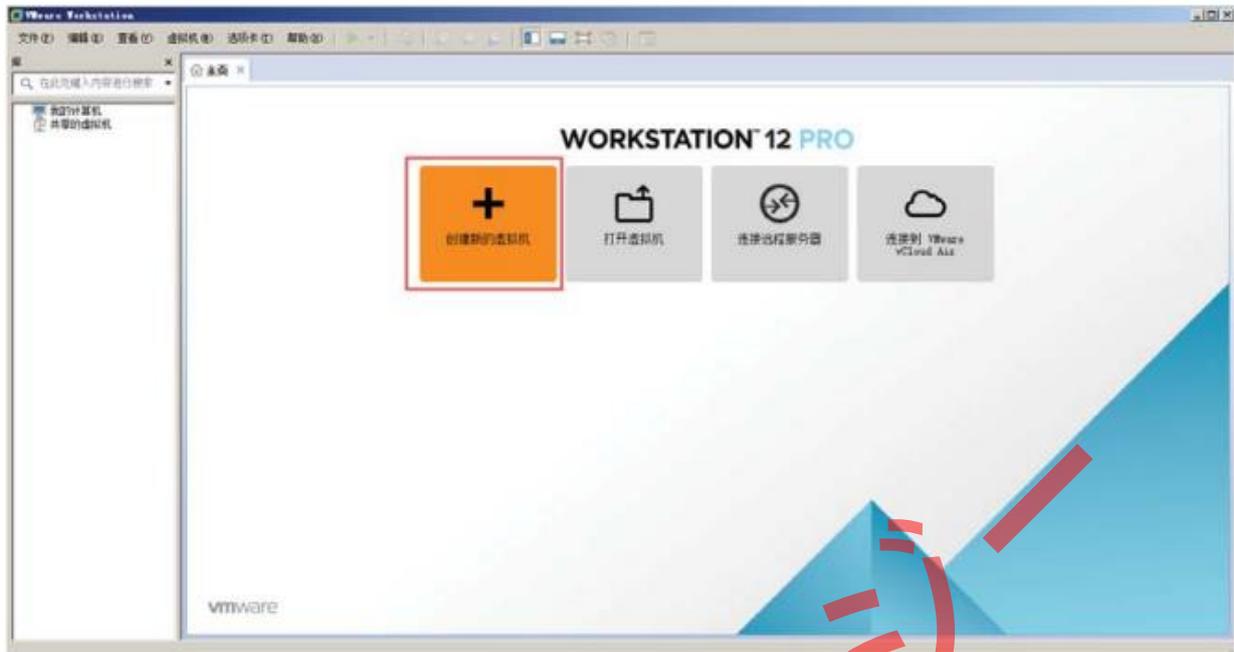
仮想マシンをインストールしたら、仮想マシンにLinuxオペレーティングシステムをインストールする。Ubuntuデスクトップオペレーティングシステムのインストールが簡単のため、ubuntuデスクトップオペレーティングシステムを選択した。

このマニュアルでは、Ubuntu 16.04.3 LTS 64ビットオペレーティングシステムを使用する。

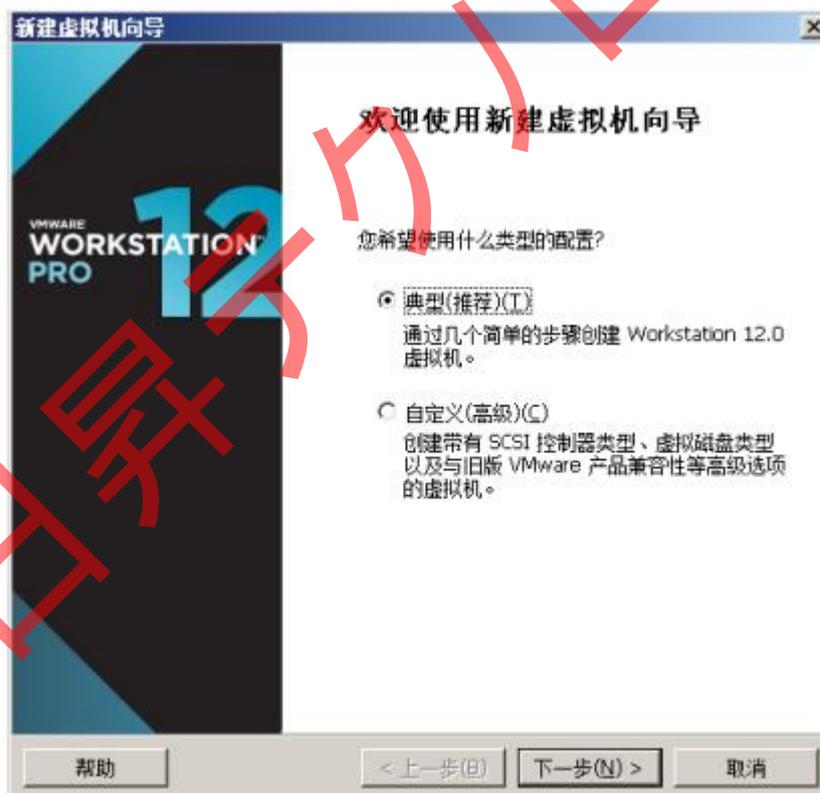
他のバージョンを使用している場合、予期しないエラーが発生する可能性があるから、バージョンの一貫性を維持してください。システムをアップグレードしないでください。

Ubuntuのインストール手順は次のとおりである。

- 1) デスクトップのVMware Workstation Proのアイコンをダブルクリックし、VMwareの作業インターフェイスで[新しい仮想マシンの作成]アイコンをクリックする。



2) 次のステップを選択してください。



3) 「Installer CD Image (iso)」アイテムを選択し、ダブルクリックしてubunt CDイメージファイルubuntu-16.04.3-desktop-amd64.isoを見つける。

4) 仮想マシンウィザードで、仮想マシンのフルネーム、ユーザー、およびパスワードを入力する。ここでのフルネーム、ユーザー名、パスワードはユーザーが設定できる。

新建虚拟机向导

简易安装信息  
这用于安装 Ubuntu 64 位

个性化 Linux

全名(F): alinx

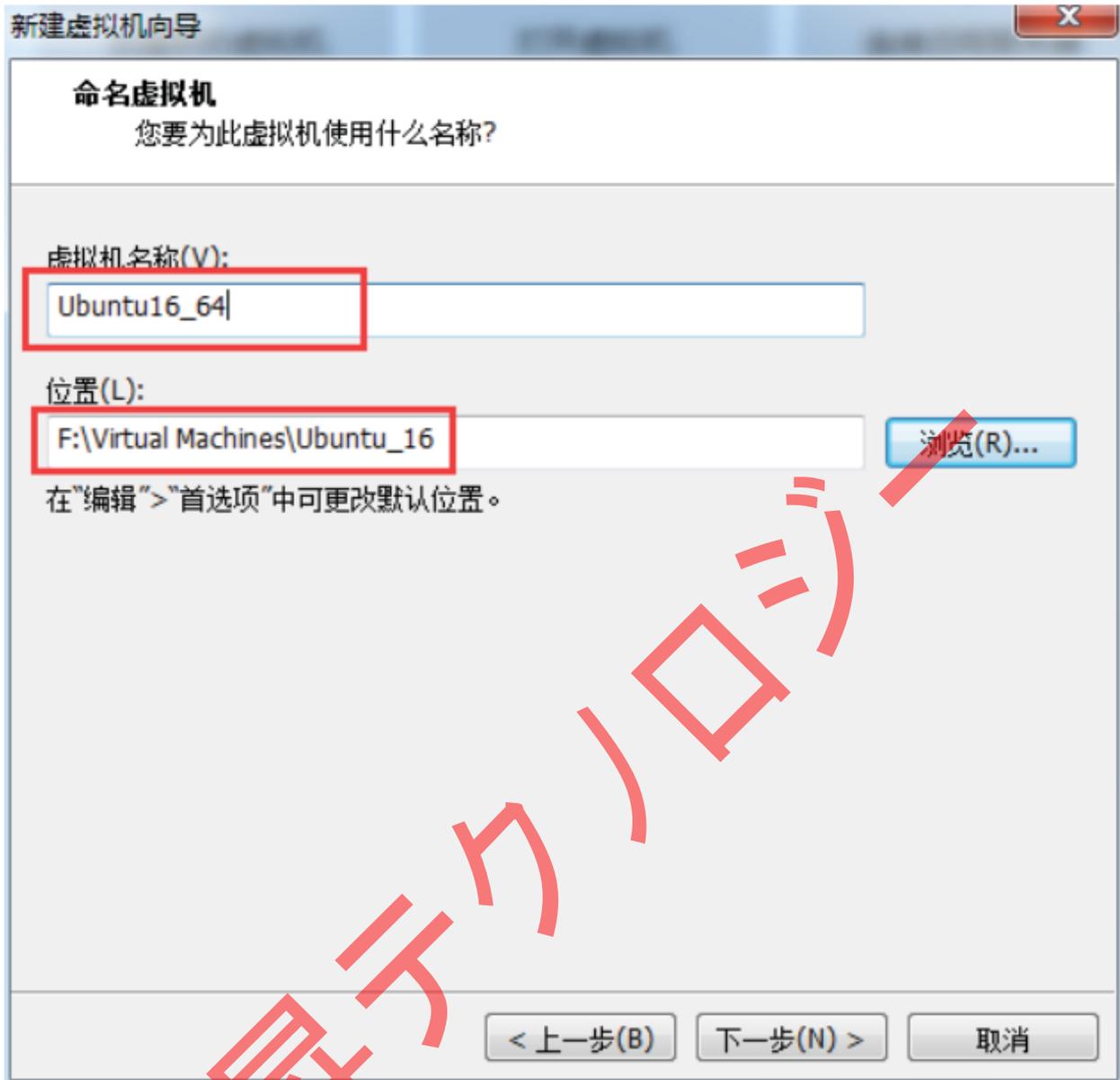
用户名(U): alinx

密码(P): \*\*\*\*\*

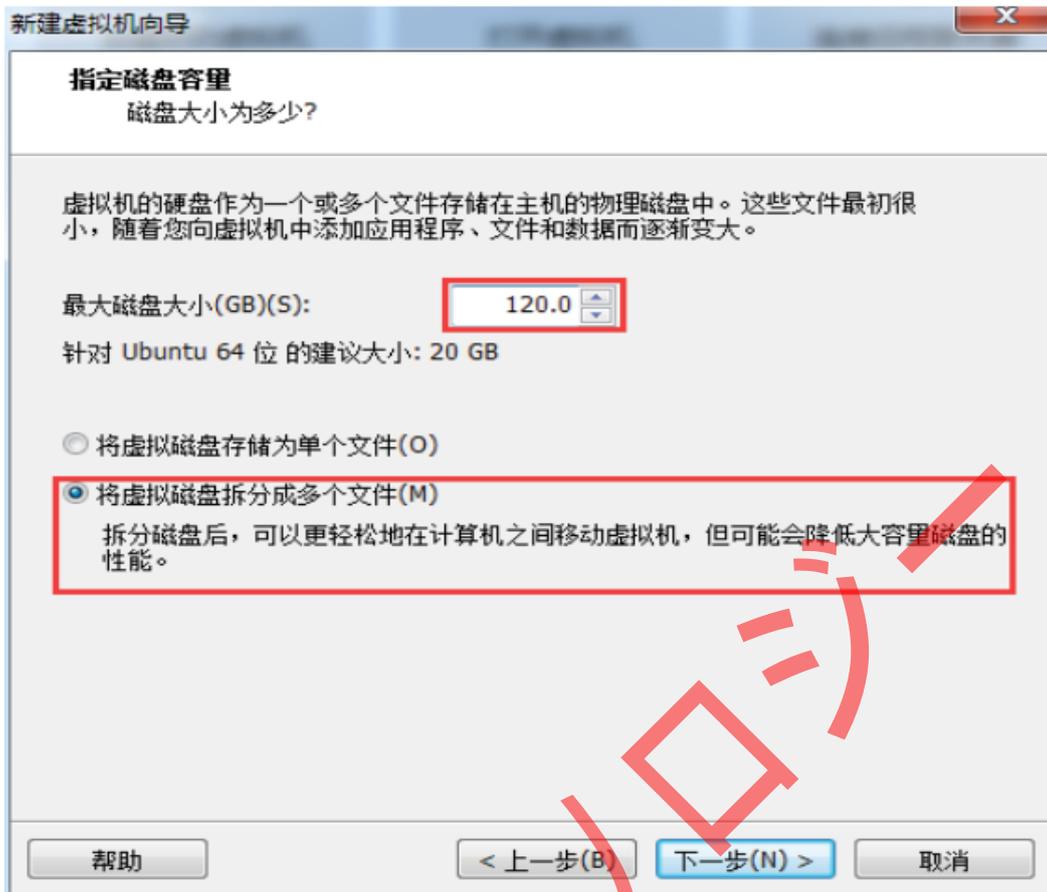
确认(C): \*\*\*\*\*

帮助 < 上一步(B) 下一步(N) > 取消

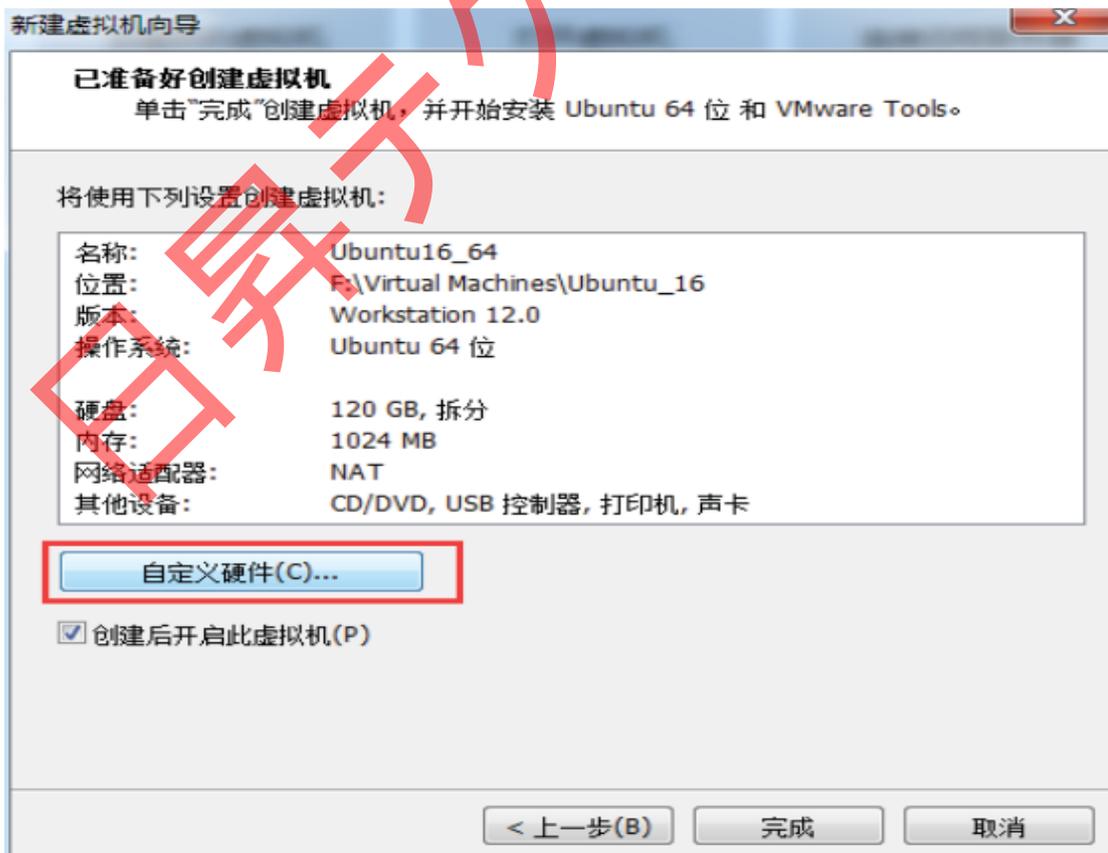
- 5) 仮想マシン名は変更できるが、十分な容量のあるディスクにインストールしてください。



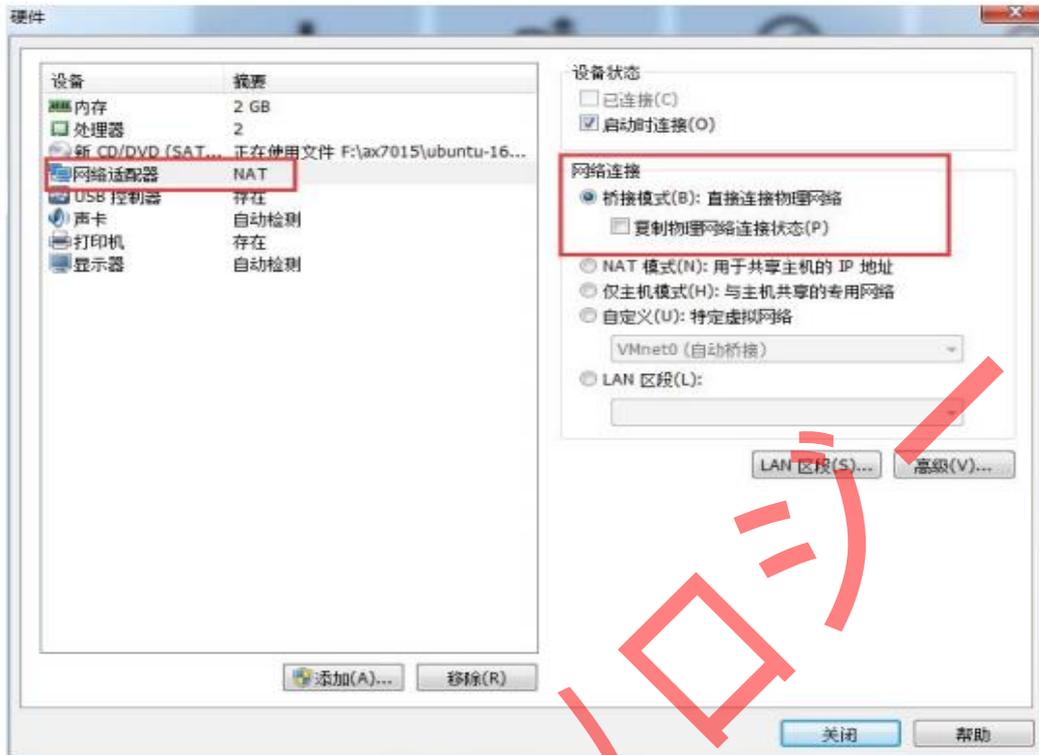
- 6) 最大ディスクサイズを300Gに設定する。システムをインストールするため、このスペースを大きくする必要がある。ユーザーは、ハードディスクの空き容量に応じて適切な容量を選択できるが、300G以上にするをお勧めする。



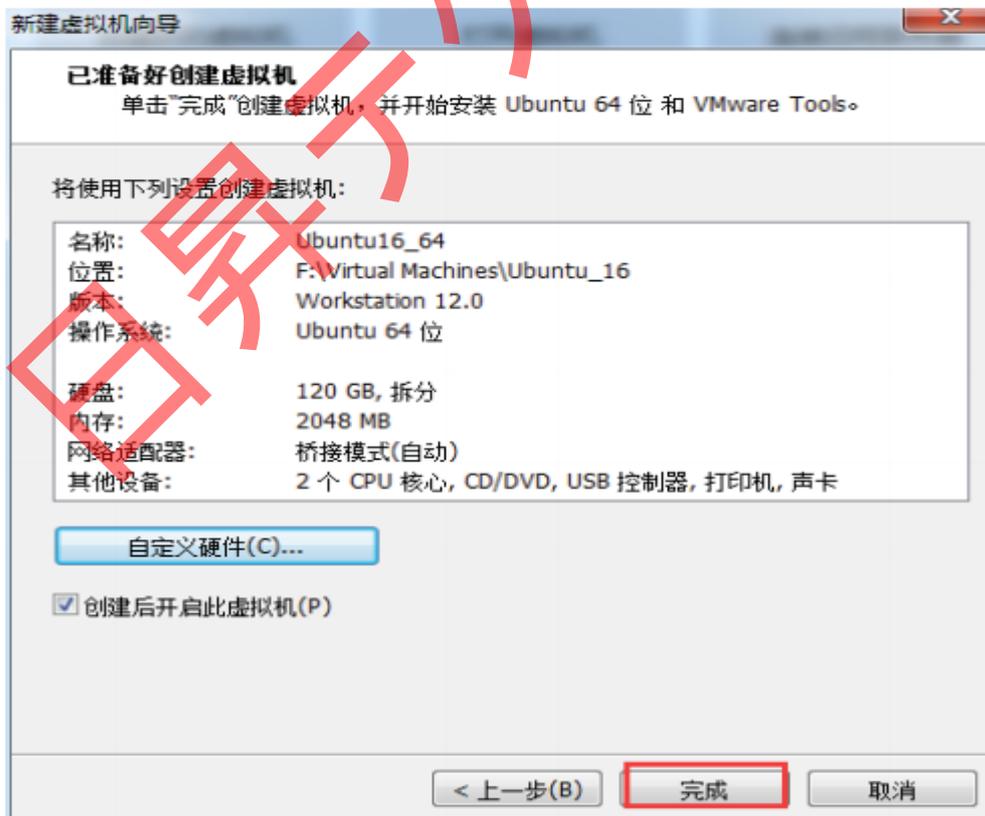
7) カスタムハードウェアを選択する



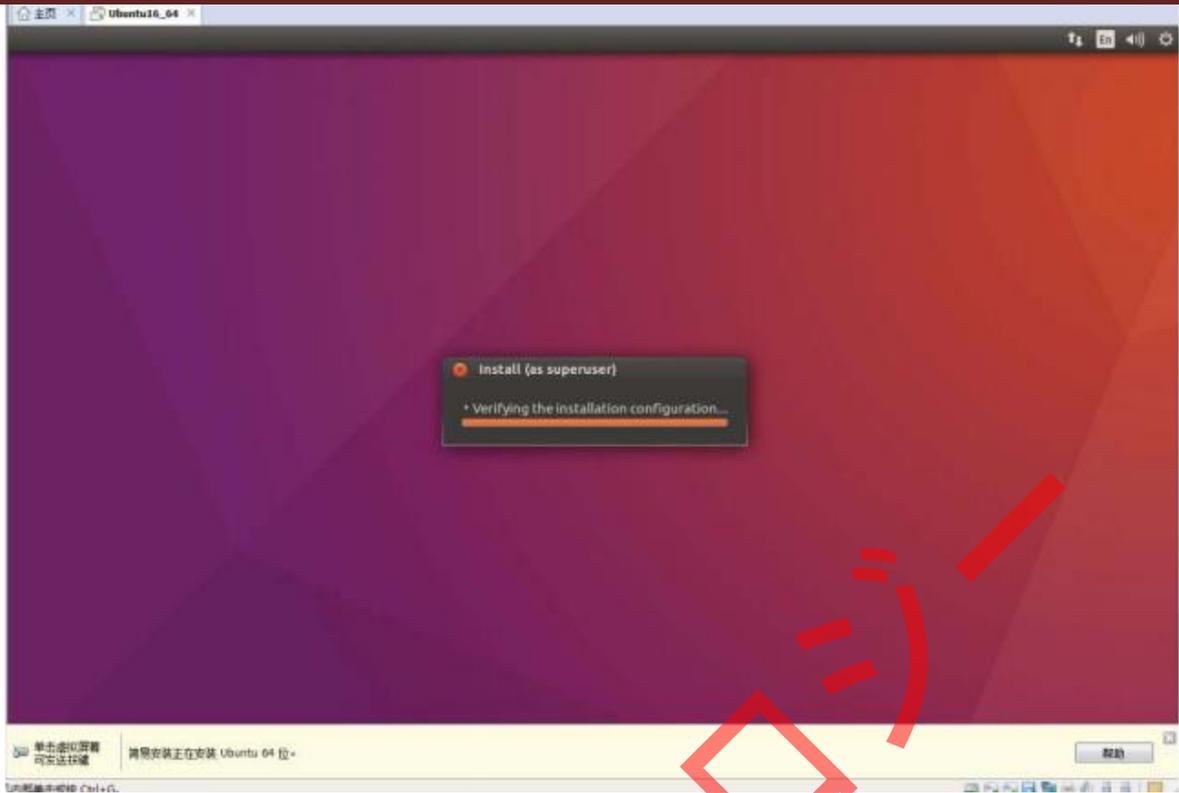
- 8) 変更されたメモリサイズとプロセッサコア、ネットワークアダプタオプション、ネットワーク接続に基づいてブリッジモードを選択できる。



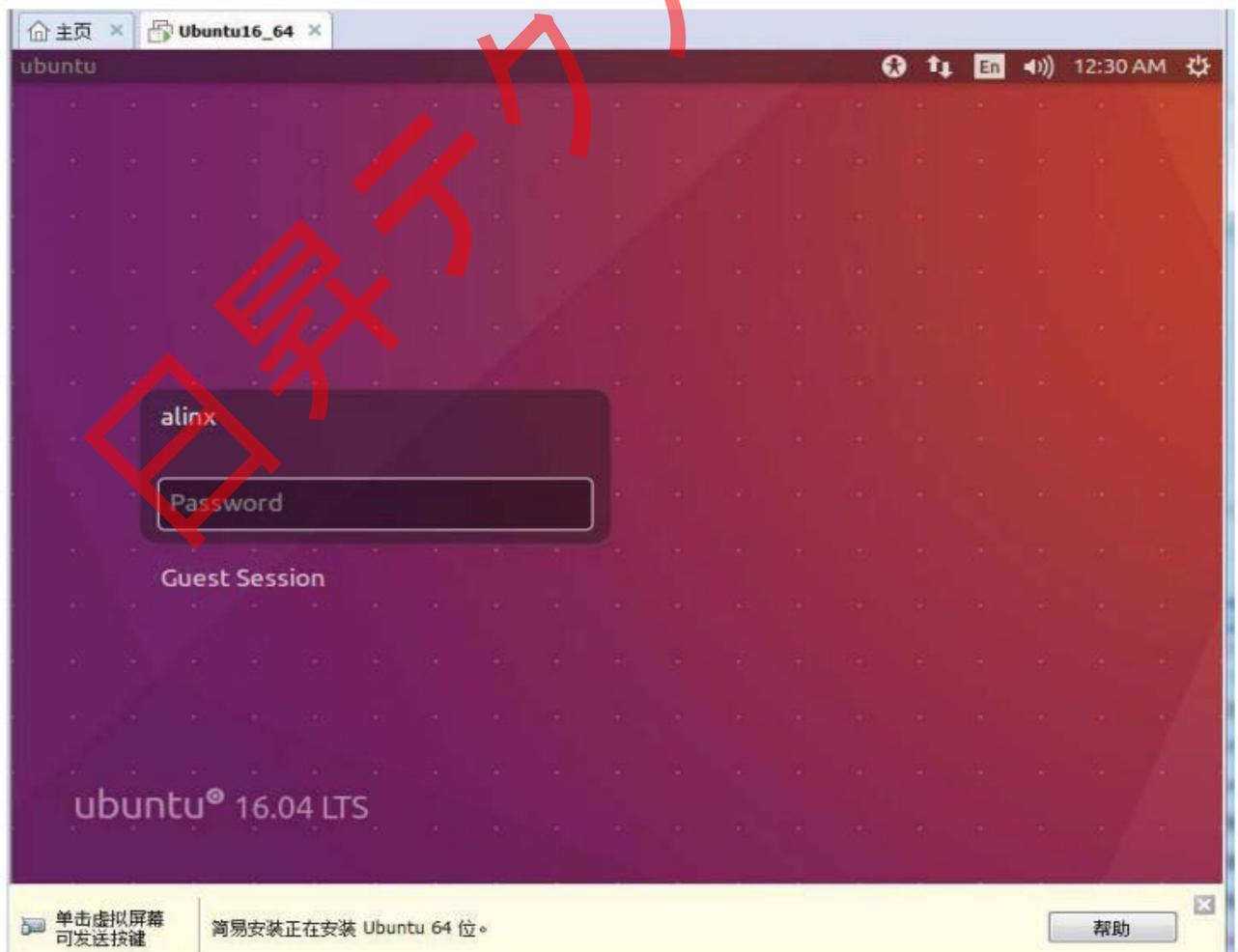
- 9) 「完了」をクリックして、インストールが始まる。



- 10) 時間かかるから、しばらく待ってください。

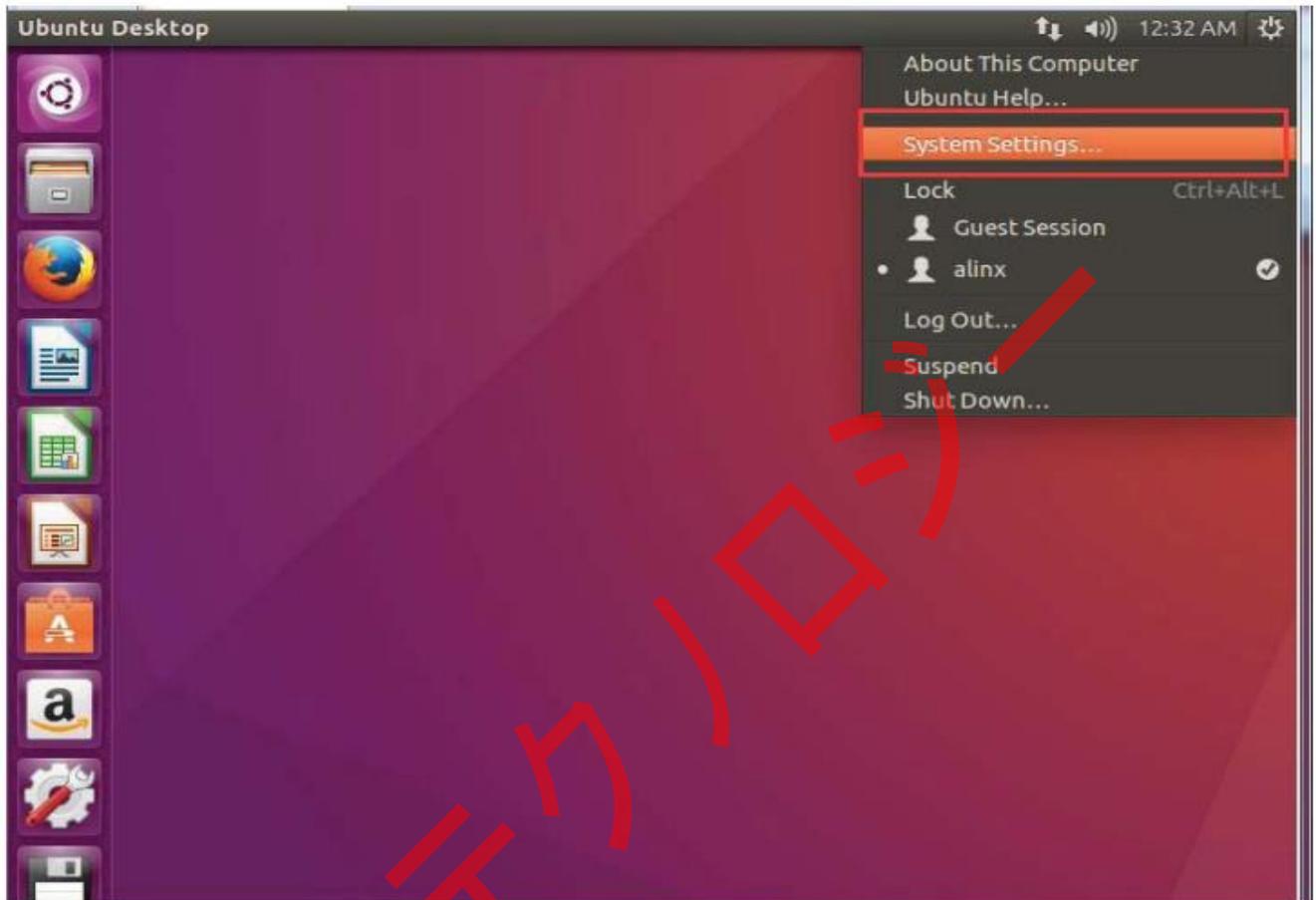


11) インストール完了後、システムに入る。

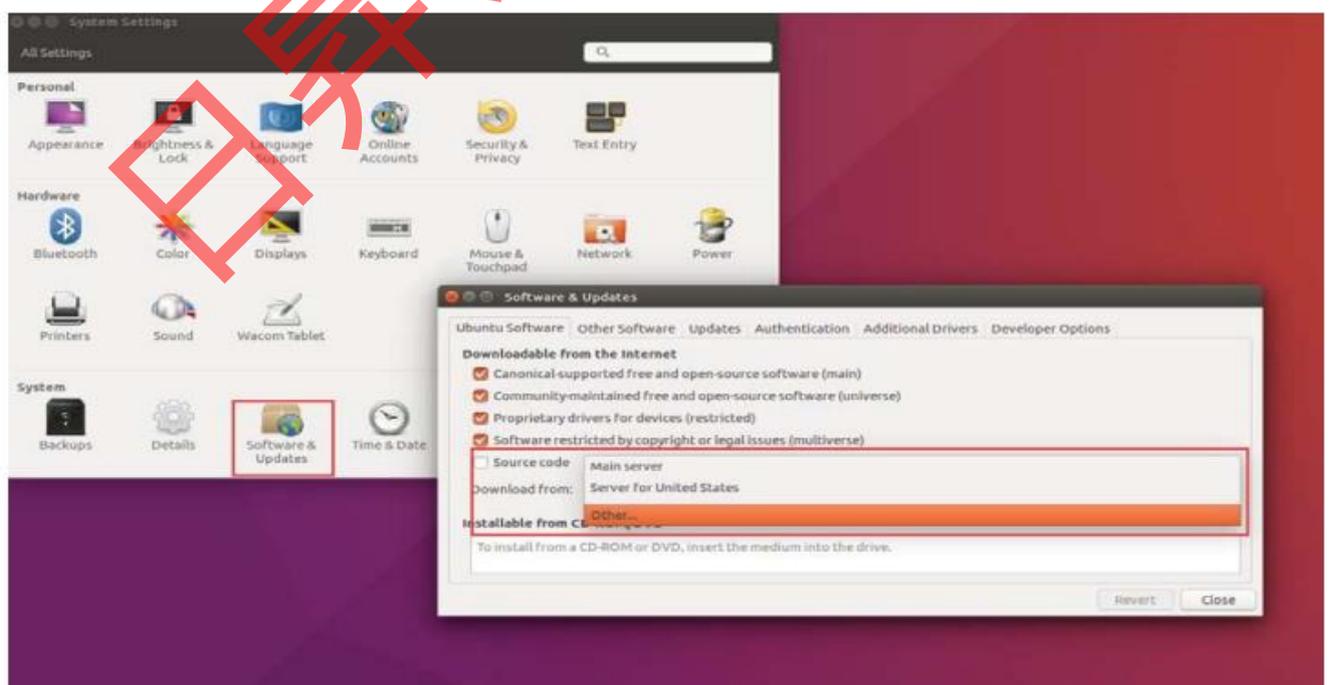


## 14.2.2 ソフトウェアソースサーバーを変更する

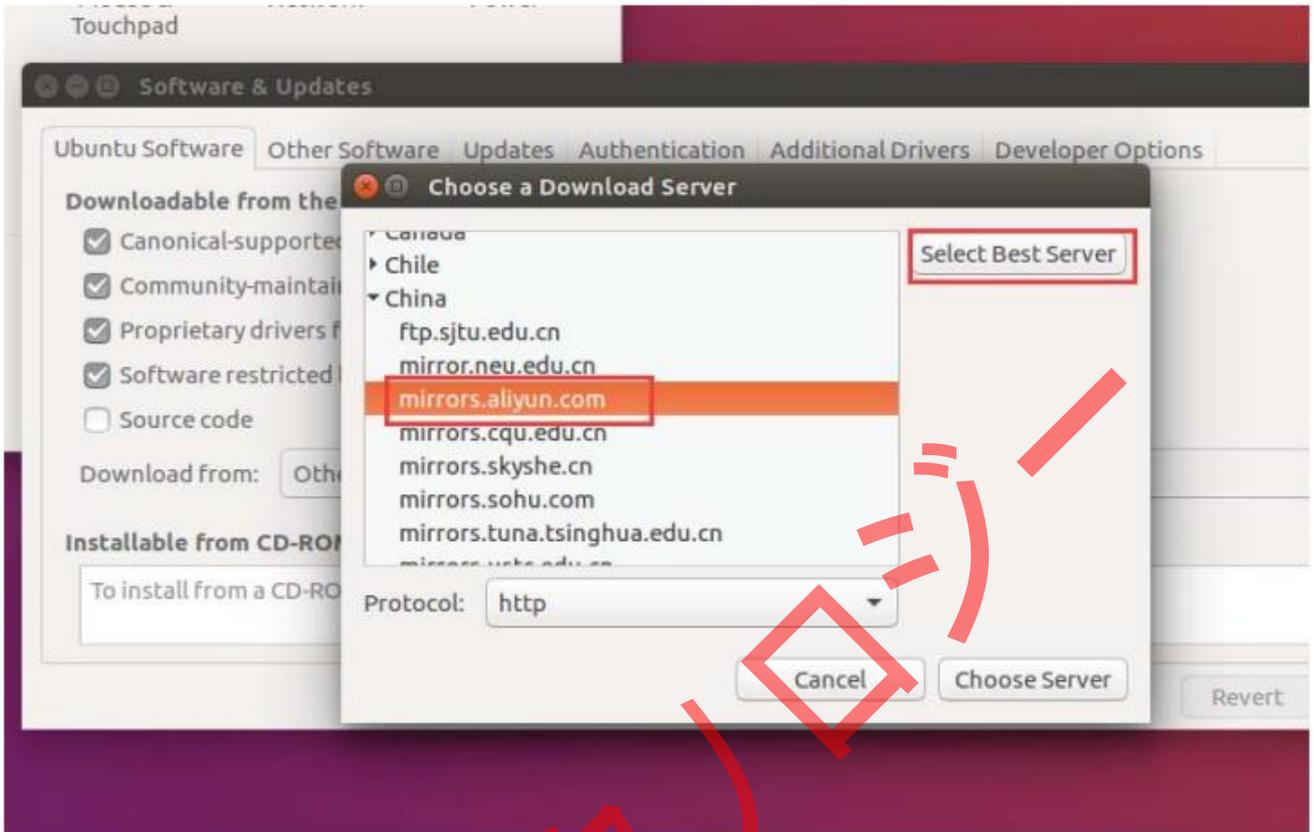
1) ソフトウェアを便利にインストールするには、ソフトウェアソースを設定し、システム設定をクリックする。



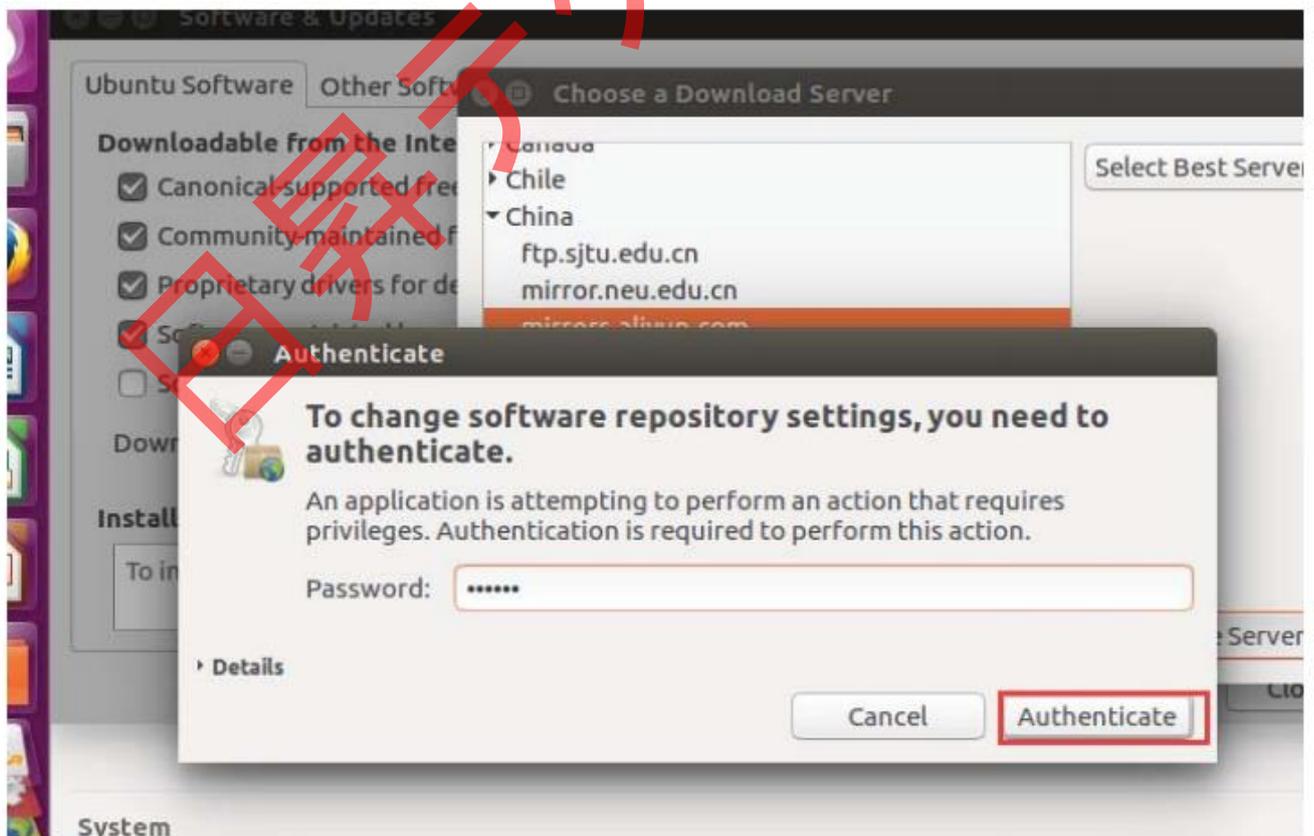
2) Software & UpdatesにOtherを選択する。



3) [select best sever]をクリックして、最速のサーバーが出て、次に、choose serverを選択する。これらの操作は、仮想マシンがインターネットに接続できるという事実に基づいている。

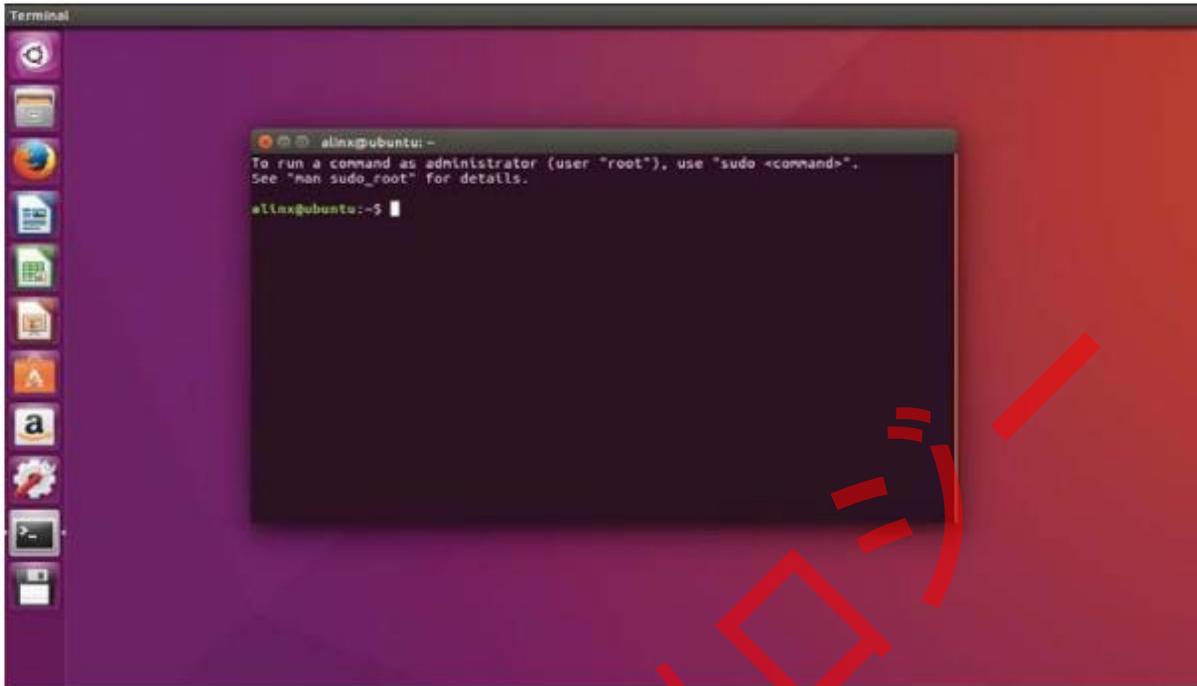


4) パスワードを入力して、ソフトウェアソースの変更を完了する



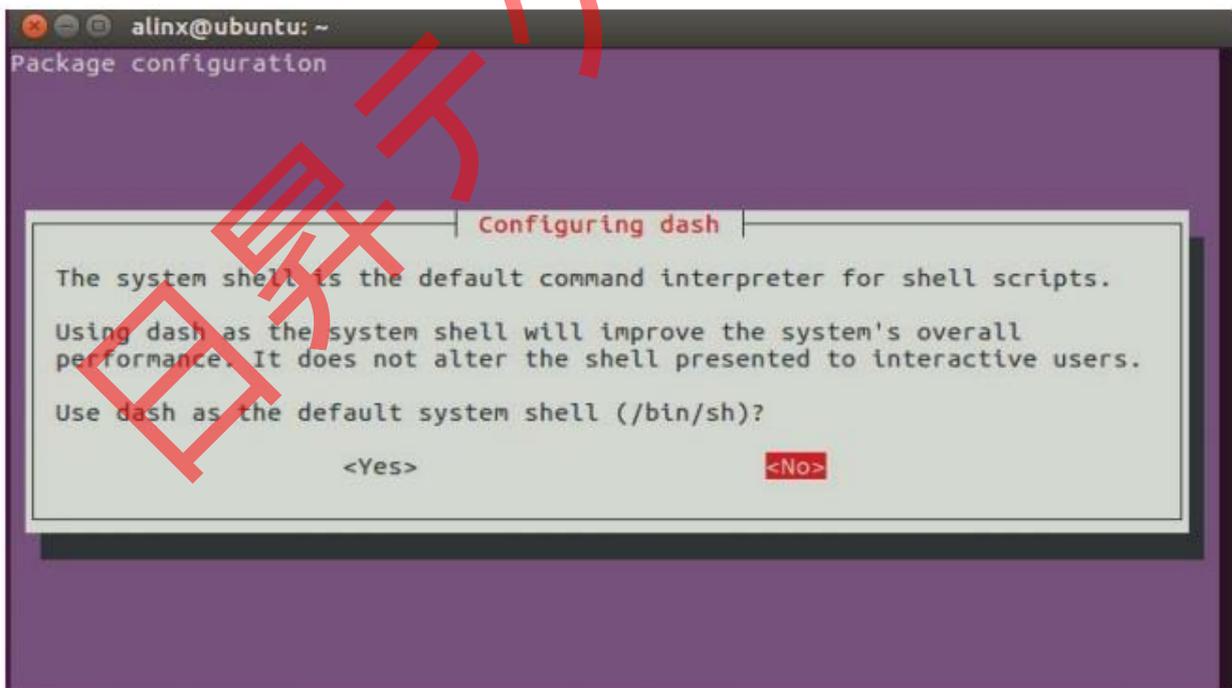
### 14.2.3 bash をデフォルトの sh に設定する

- 1) Ctrl + Alt + Tでターミナルを開く。



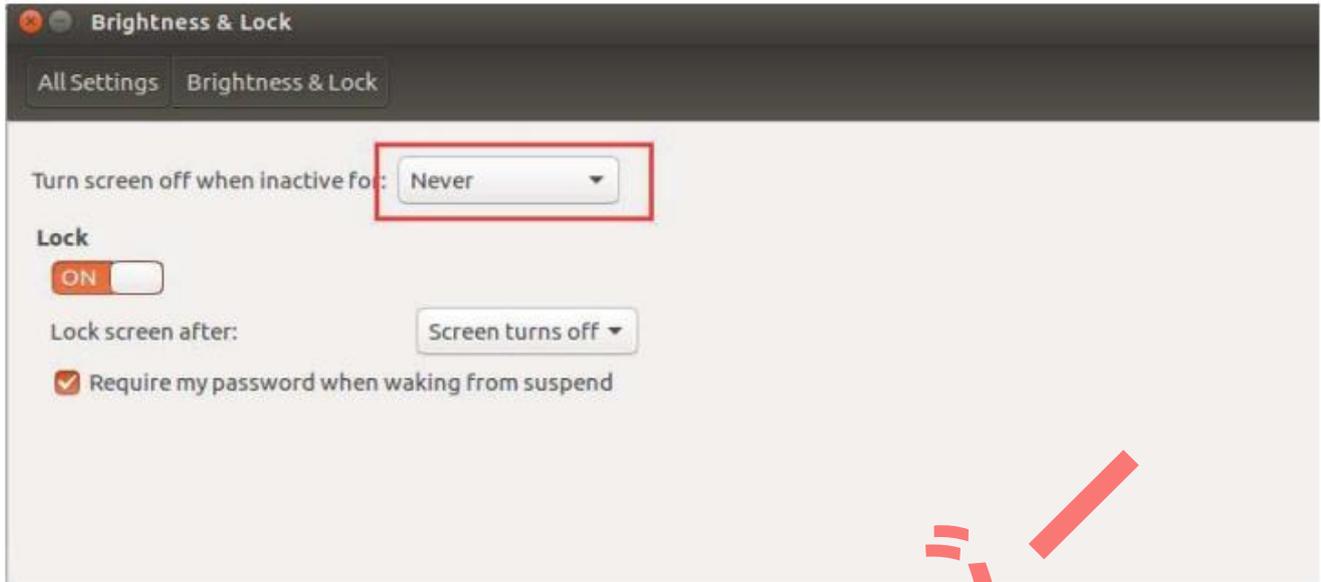
- 2) Configuring dashコマンドを入力し、NOを選択し、Enterを押して確認する

```
sudo dpkg-reconfigure dash
```



### 14.2.4 画面ロック時間を設定する

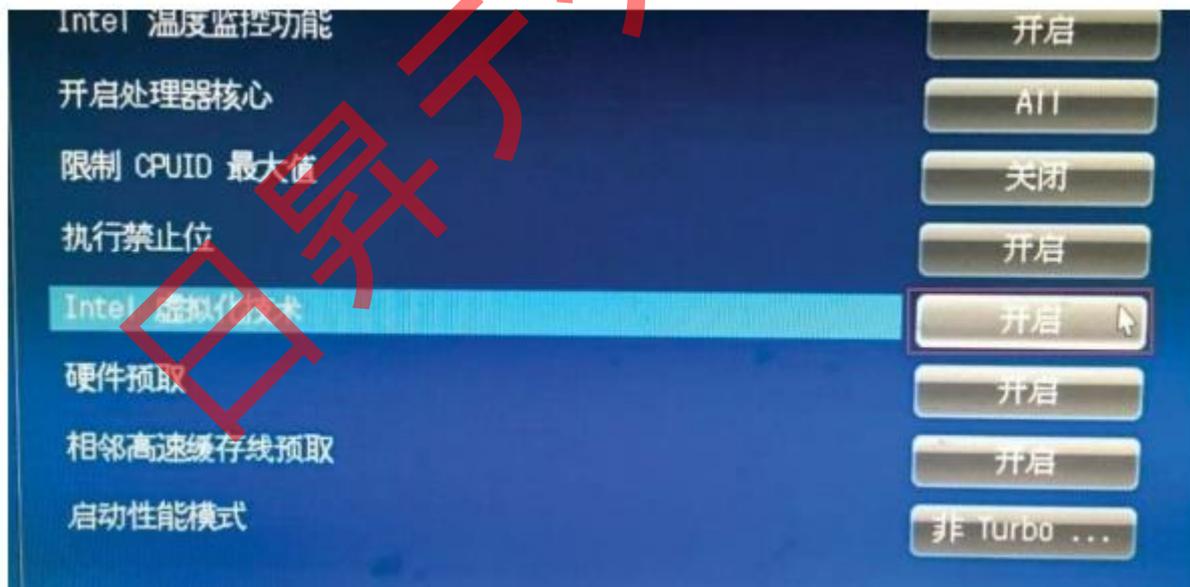
Ubuntuシステムに大きなファイルをコピーできるように、画面ロックをキャンセルする。



### 14.3 よくある問題

#### 14.3.1 仮想マシンには仮想化サポートが必要である。

- 1) Ubuntuをインストールし、次のエラーメッセージボックスが表示された場合、ユーザーはコンピューターを再起動し、BIOSに入ってセットアップする必要がある。  
 コンピューターを再起動した後、BIOSに入り、Intel仮想化を見つけて、[開く]をクリックする。マザーボードによって名前が異なる場合がある。

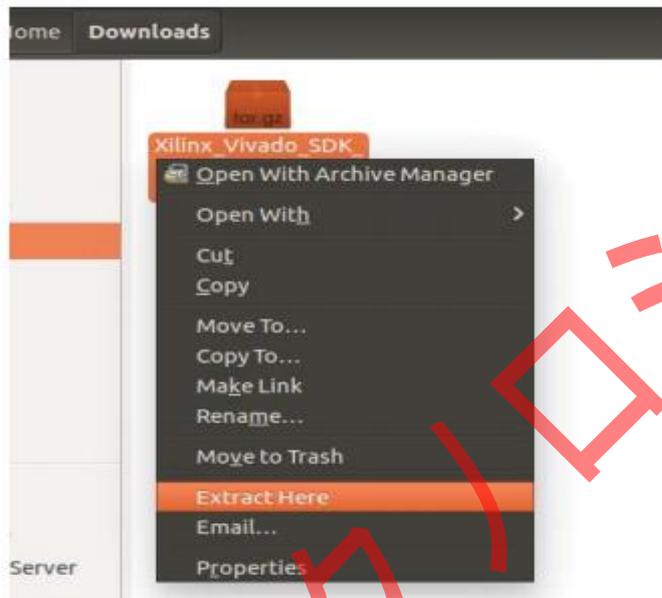


## 第十五章 Ubuntu で Linux バージョンの Vivado ソフトウェアをインストールする

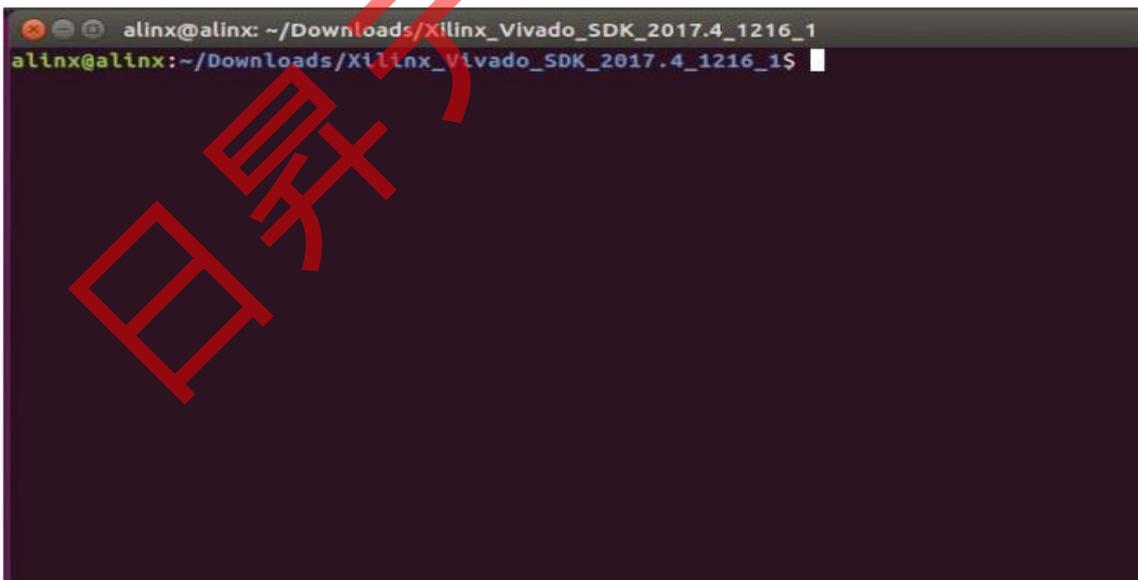
WindowsでのVivadoソフトウェアはほとんどの問題を解決できるが、場合によってはLinuxバージョンのvivado、特にSDKを使用する必要がある、多くのアプリケーションをクロスコンパイルできる。

### 15.1 Linux バージョンの Vivado をインストールする

- 1) インストールファイルを仮想マシンのubuntuにコピーし、ファイルを抽出する。



- 2) ターミナルを使用して、解凍されたファイルに入る



- 3) コマンドを実行する。

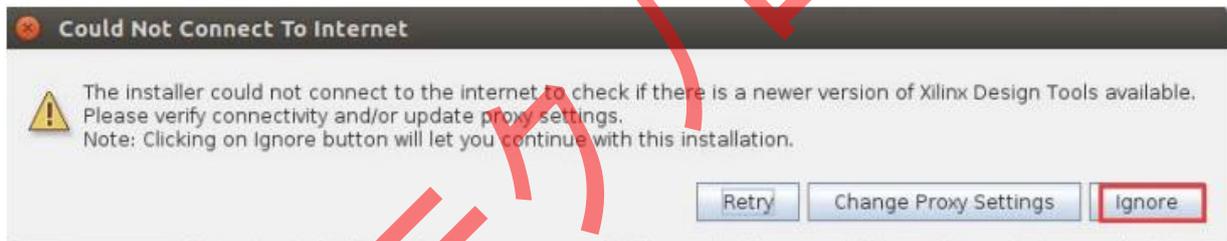
```
sudo chmod +x xsetup
```

```
alinx@alinx: ~/Downloads/Xilinx_Vivado_SDK_2017.4_1216_1
alinx@alinx:~/Downloads/Xilinx_Vivado_SDK_2017.4_1216_1$ sudo chmod +x xsetup
sudo: unable to resolve host alinx
[sudo] password for alinx:
alinx@alinx:~/Downloads/Xilinx_Vivado_SDK_2017.4_1216_1$
```

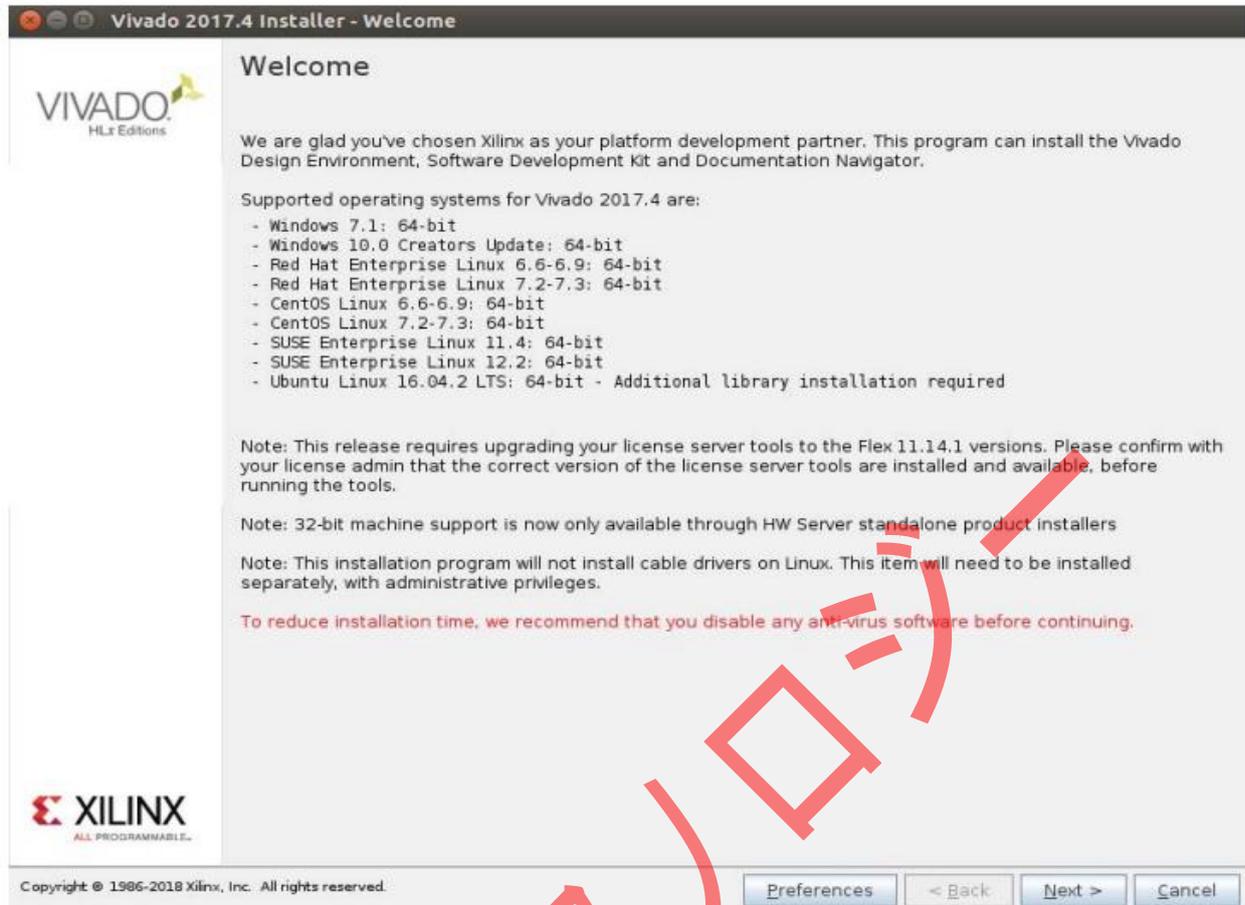
- 4) コマンドを実行し、インストールを開始する。**注意、ここはsudoインストールである。**

```
sudo ./xsetup
```

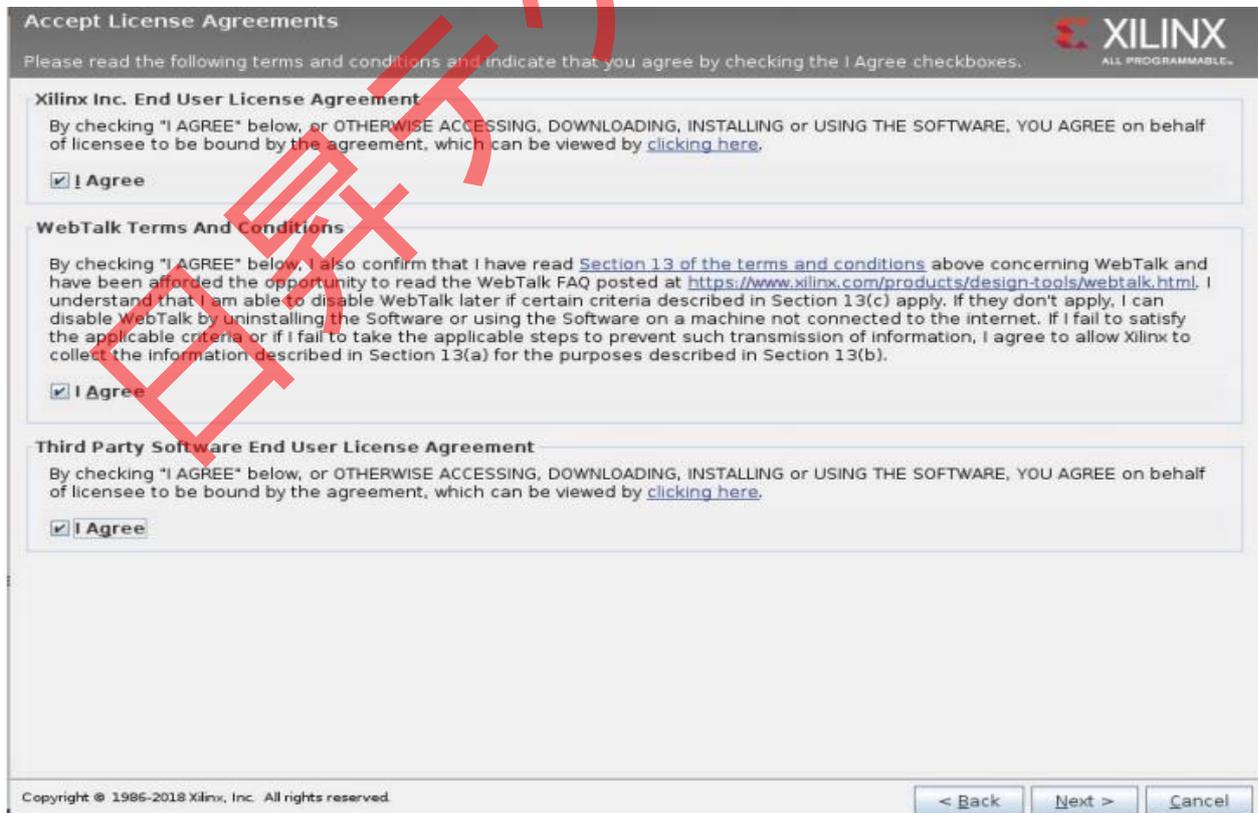
- 5) 以下のようなウィンドウが表示されたら、Ignoreをクリックする。



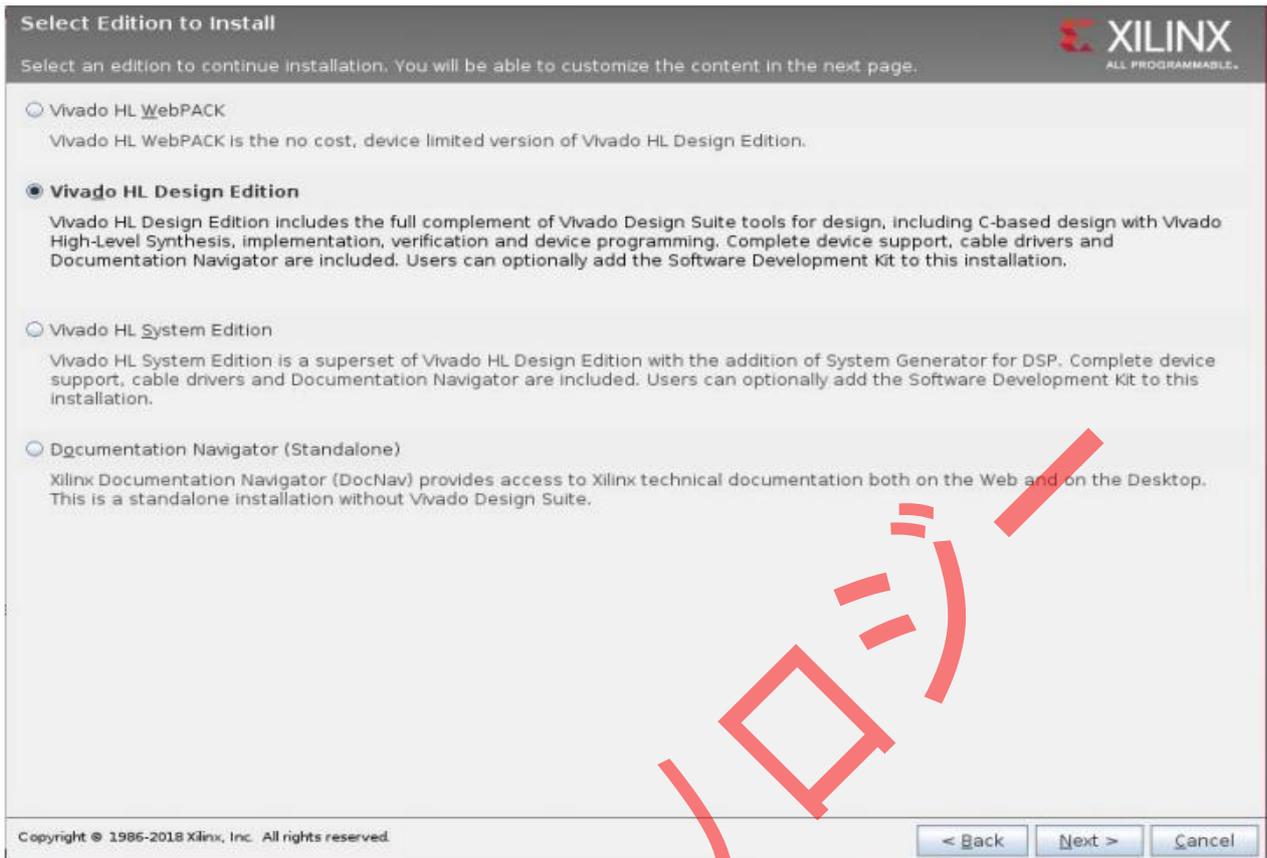
- 6) インストールプロセスでは、ウイルス対策ソフトウェアをオフにする必要がある



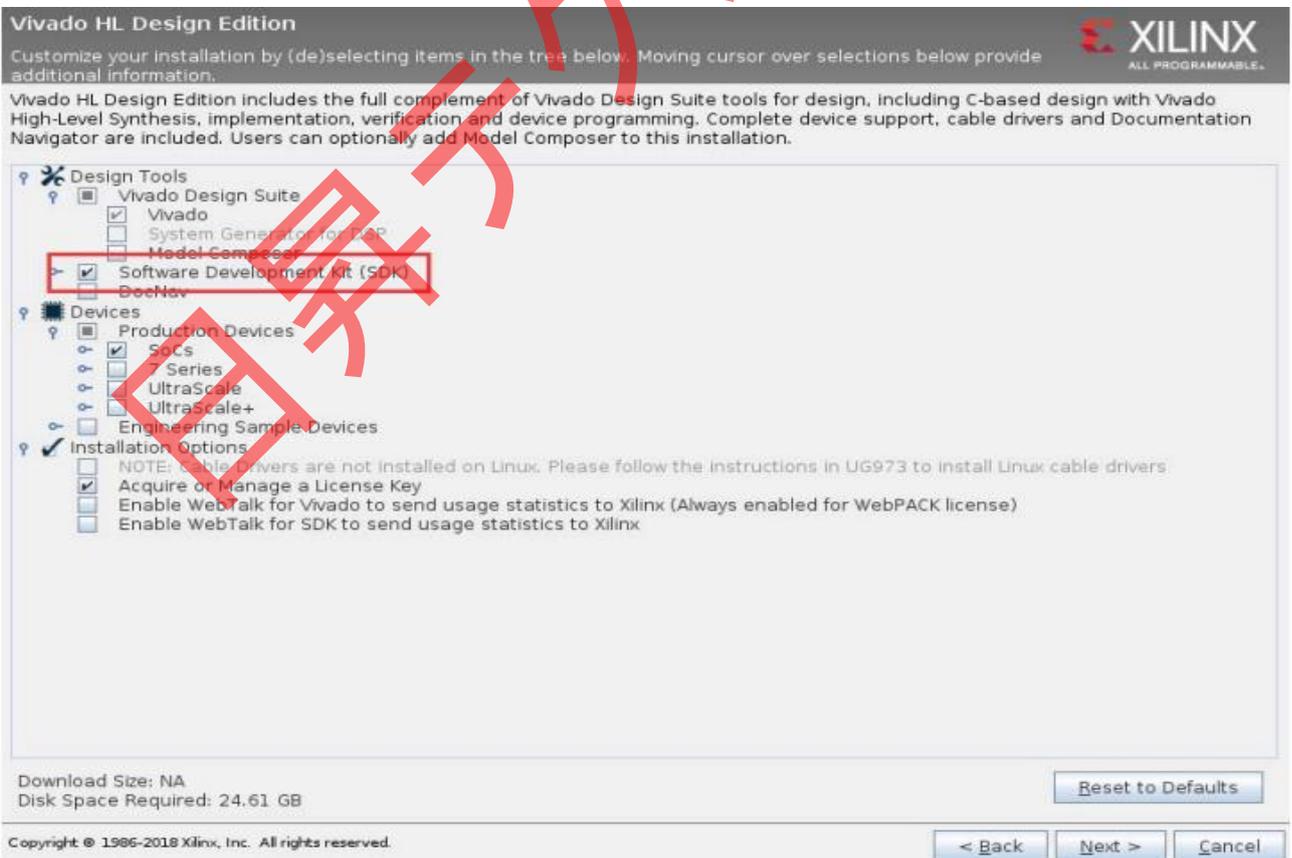
7) 利用規約に同意する



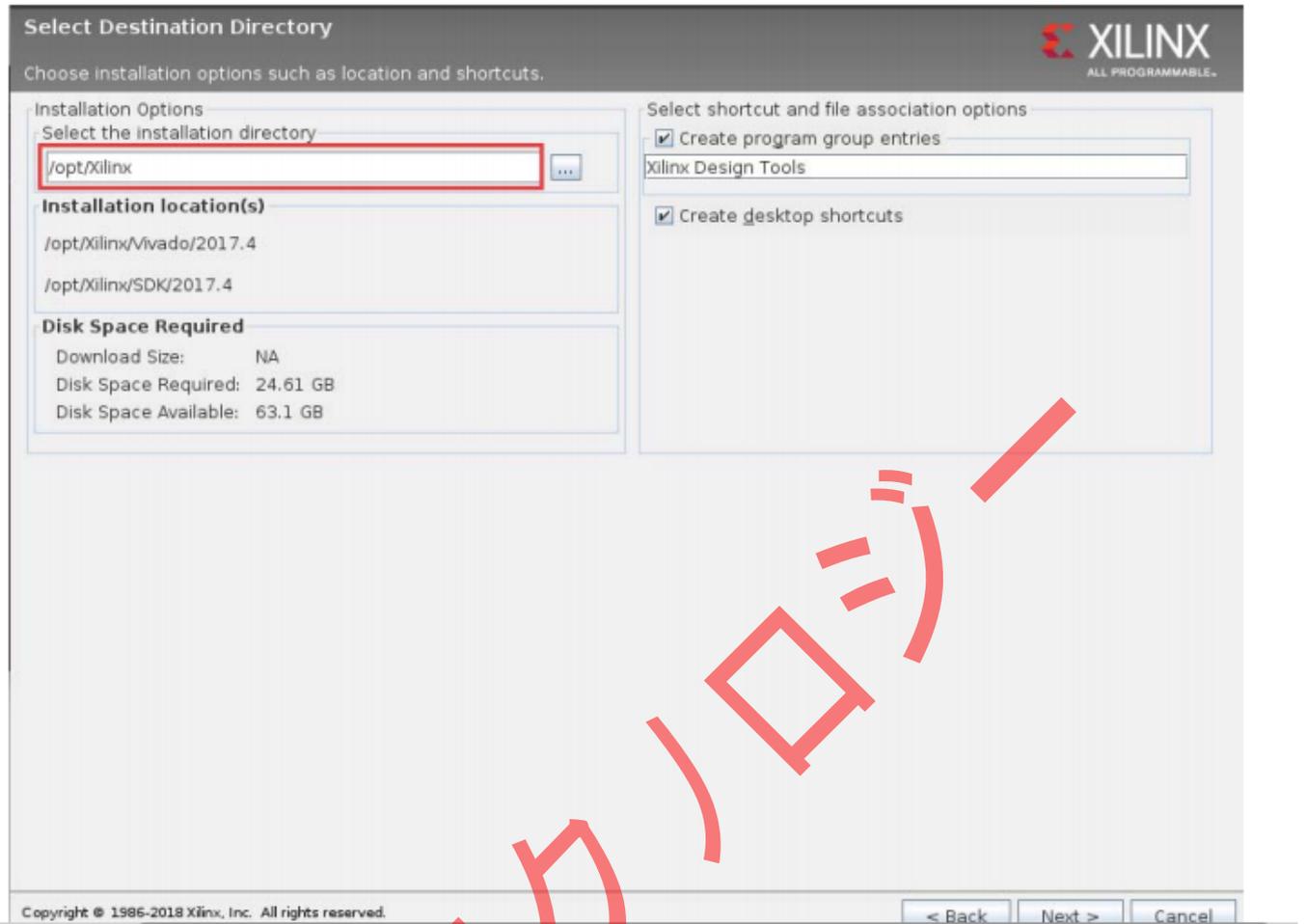
8) Vivado HL Design Editionを選択する



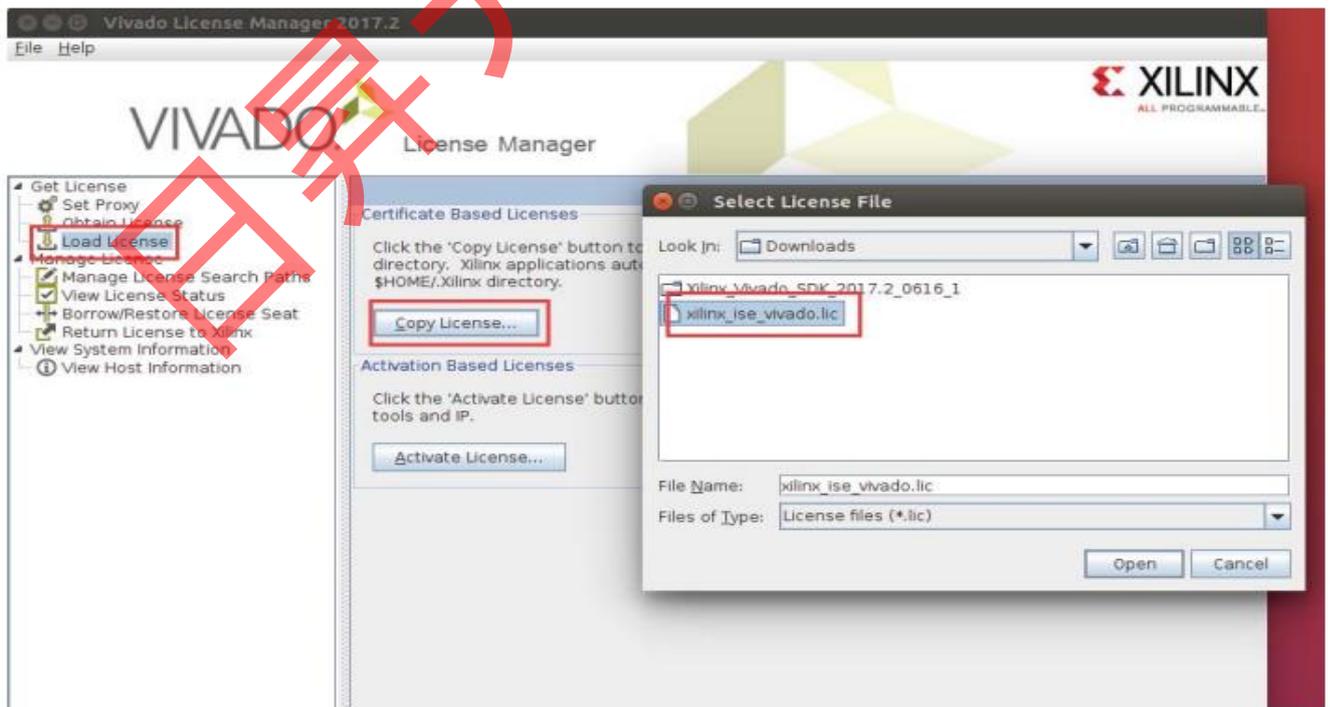
9) 必ずここでSoftware Development Kit (SDK) を選択してください。



- 10) インストールパスはデフォルトパスを使用する。



- 11) Copy licenseをクリックしてlicファイルをインストールする。



## 15.2 許可設定

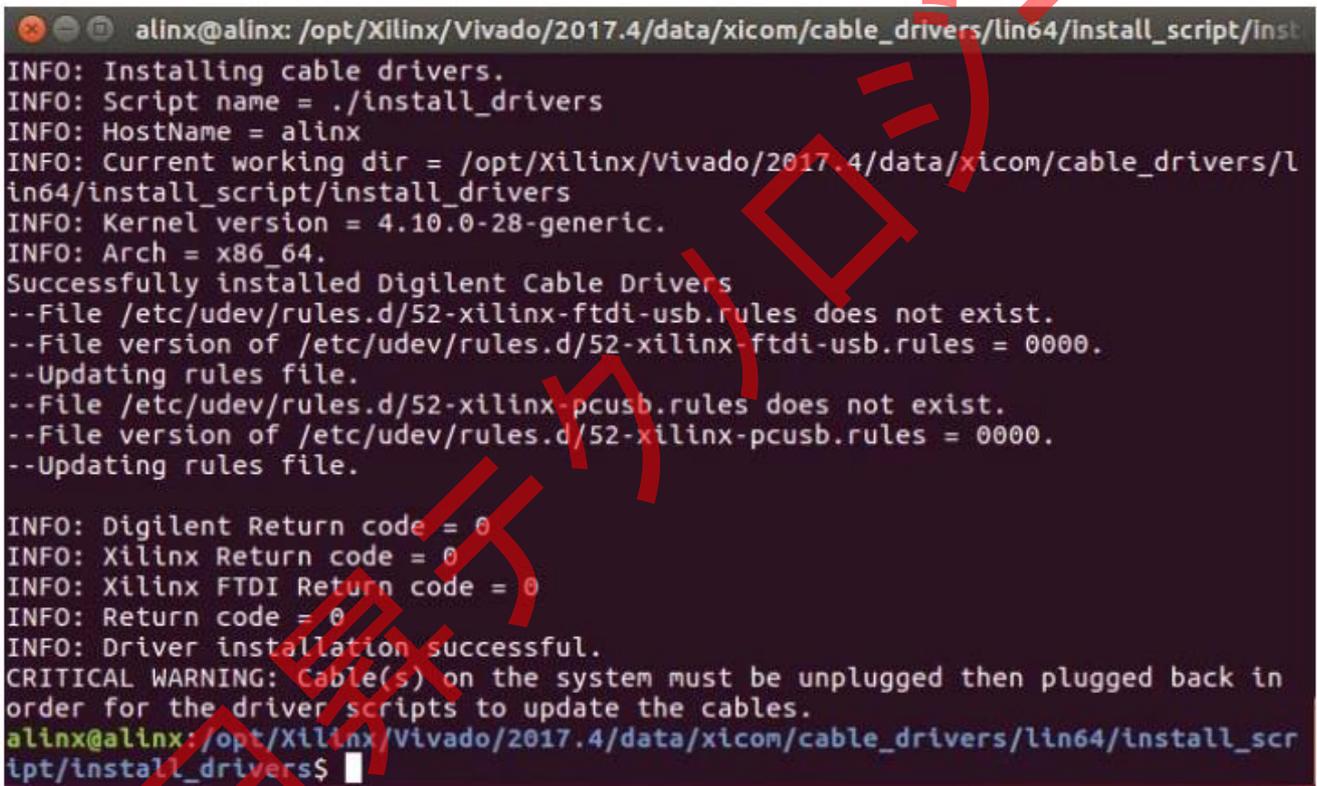
コマンドを実行して実行許可を追加する。

```
sudo chmod 777 -R /opt/Xilinx/  
sudo chmod 777 -R ~/.Xilinx/
```

## 15.3 ダウンローダードライバーをインストールする

次のコマンドを実行して、ダウンローダードライバーをインストールする

```
cd /opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_script/install_drivers/  
sudo ./install_drivers
```



```
alinx@alinx: /opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_script/inst  
INFO: Installing cable drivers.  
INFO: Script name = ./install_drivers  
INFO: HostName = alinx  
INFO: Current working dir = /opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/l  
in64/install_script/install_drivers  
INFO: Kernel version = 4.10.0-28-generic.  
INFO: Arch = x86_64.  
Successfully installed Digilent Cable Drivers  
--File /etc/udev/rules.d/52-xilinx-ftdi-usb.rules does not exist.  
--File version of /etc/udev/rules.d/52-xilinx-ftdi-usb.rules = 0000.  
--Updating rules file.  
--File /etc/udev/rules.d/52-xilinx-pcusb.rules does not exist.  
--File version of /etc/udev/rules.d/52-xilinx-pcusb.rules = 0000.  
--Updating rules file.  
  
INFO: Digilent Return code = 0  
INFO: Xilinx Return code = 0  
INFO: Xilinx FTDI Return code = 0  
INFO: Return code = 0  
INFO: Driver installation successful.  
CRITICAL WARNING: Cable(s) on the system must be unplugged then plugged back in  
order for the driver scripts to update the cables.  
alinx@alinx: /opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_scr  
ipt/install_drivers$
```

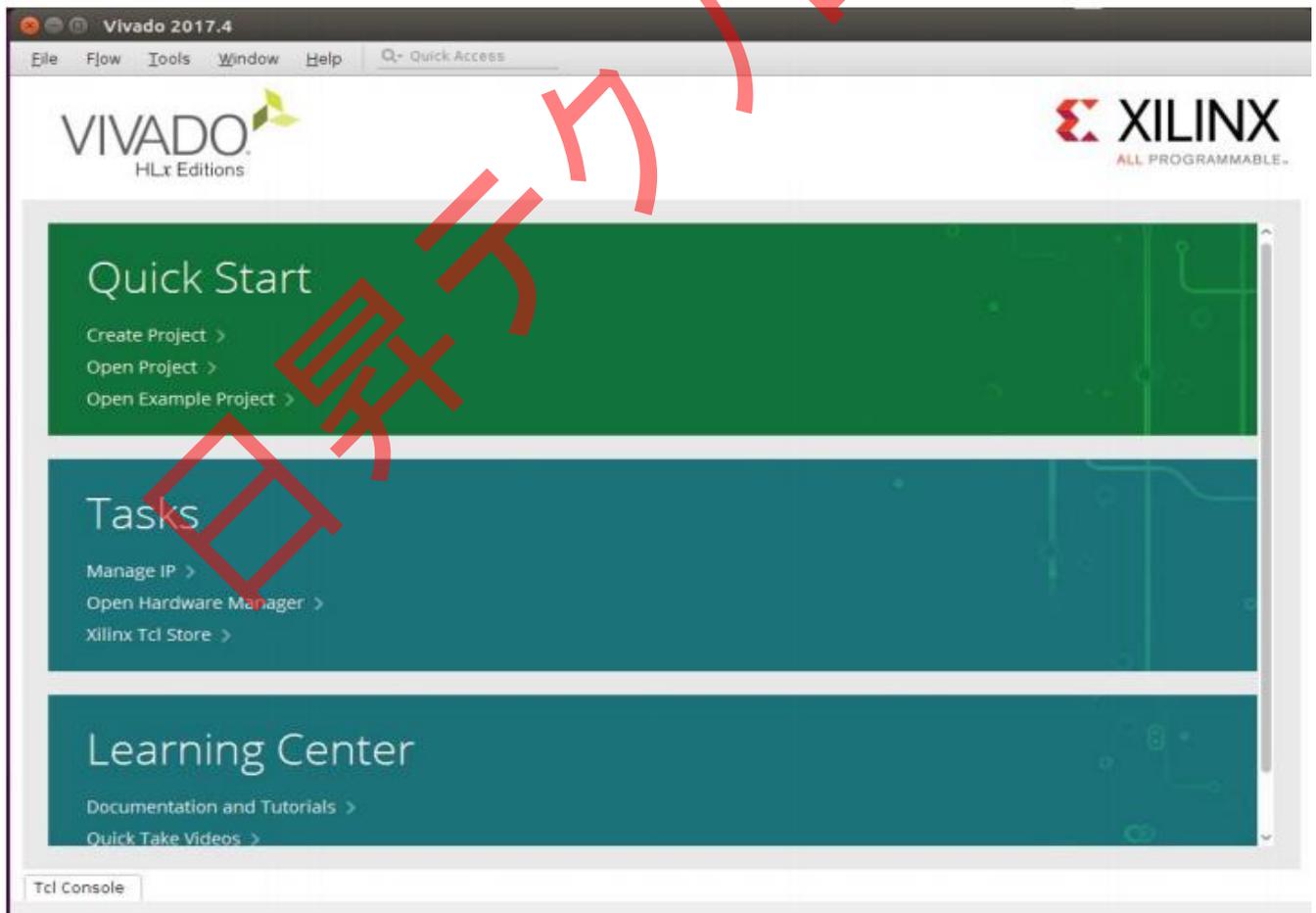
## 15.4 Vivado をテストする

1) 以下のコマンドを実行して、Vivadoを起動する

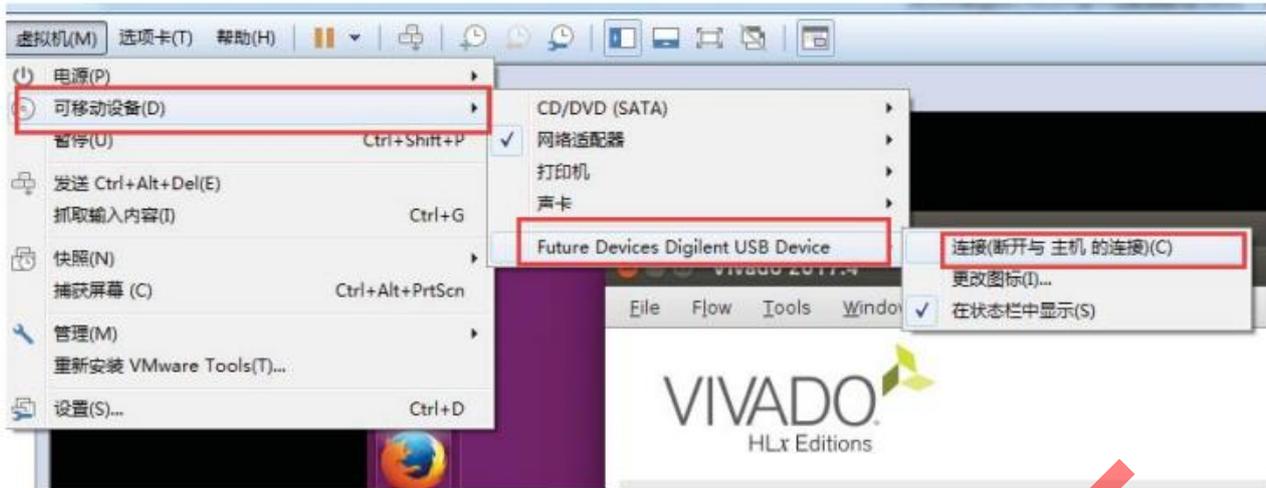
```
source /opt/Xilinx/Vivado/2017.4/settings64.sh  
vivado &
```

```
alinx@alinx: /opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_script/inst
--File /etc/udev/rules.d/52-xilinx-pcusb.rules does not exist.
--File version of /etc/udev/rules.d/52-xilinx-pcusb.rules = 0000.
--Updating rules file.

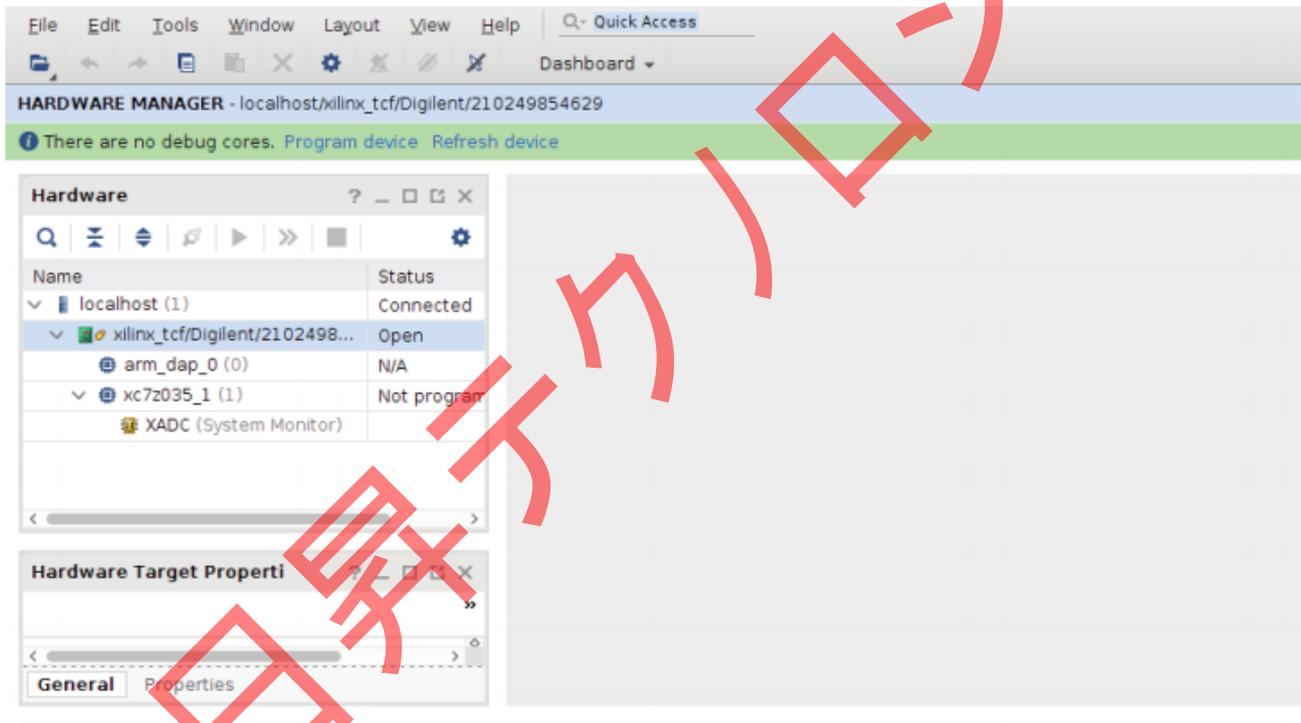
INFO: Digilent Return code = 0
INFO: Xilinx Return code = 0
INFO: Xilinx FTDI Return code = 0
INFO: Return code = 0
INFO: Driver installation successful.
CRITICAL WARNING: Cable(s) on the system must be unplugged then plugged back in
order for the driver scripts to update the cables.
alinx@alinx:/opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_scr
ipt/install_drivers$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@alinx:/opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_scr
ipt/install_drivers$ vivado &
[1] 7754
alinx@alinx:/opt/Xilinx/Vivado/2017.4/data/xicom/cable_drivers/lin64/install_scr
ipt/install_drivers$
***** Vivado v2017.4 (64-bit)
**** SW Build 2086221 on Fri Dec 15 20:54:30 MST 2017
**** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
```



2) ダウンローダーを仮想マシンに接続する



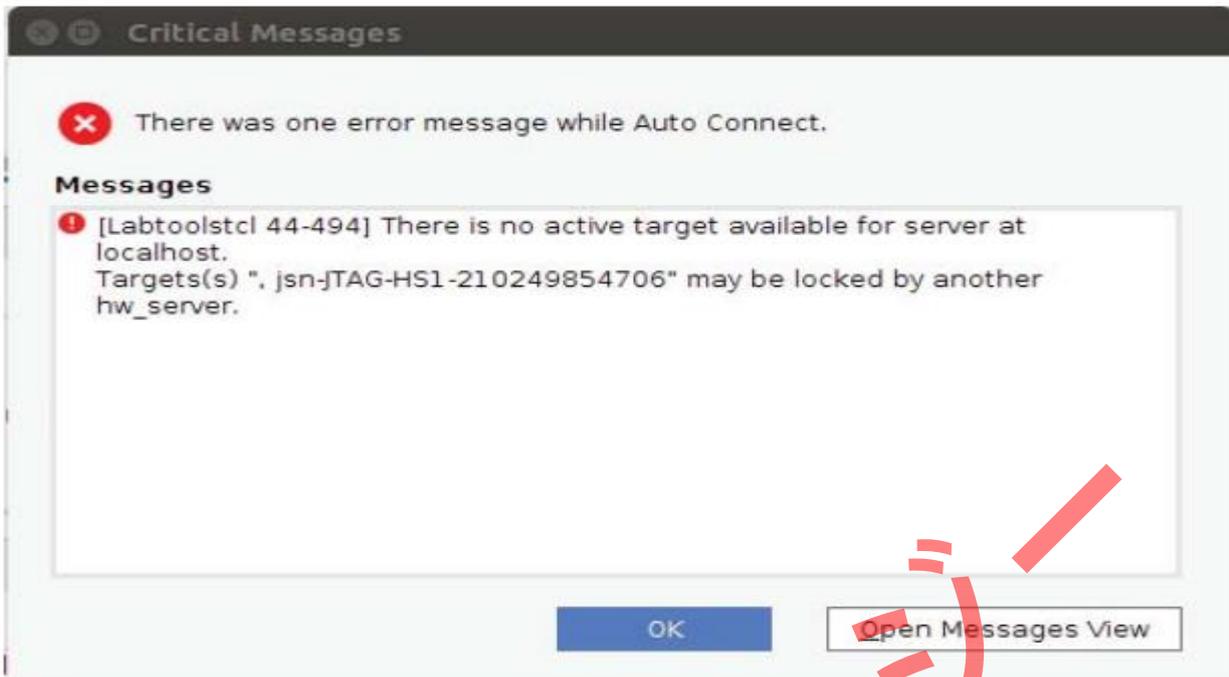
3) 開発ボードとダウンローダーを接続し、Open Hardware Managerでテストすると、通常の状態ではチップが見つかり、Vivadoおよびダウンローダードライバーが正常にインストールされていることがわかる。



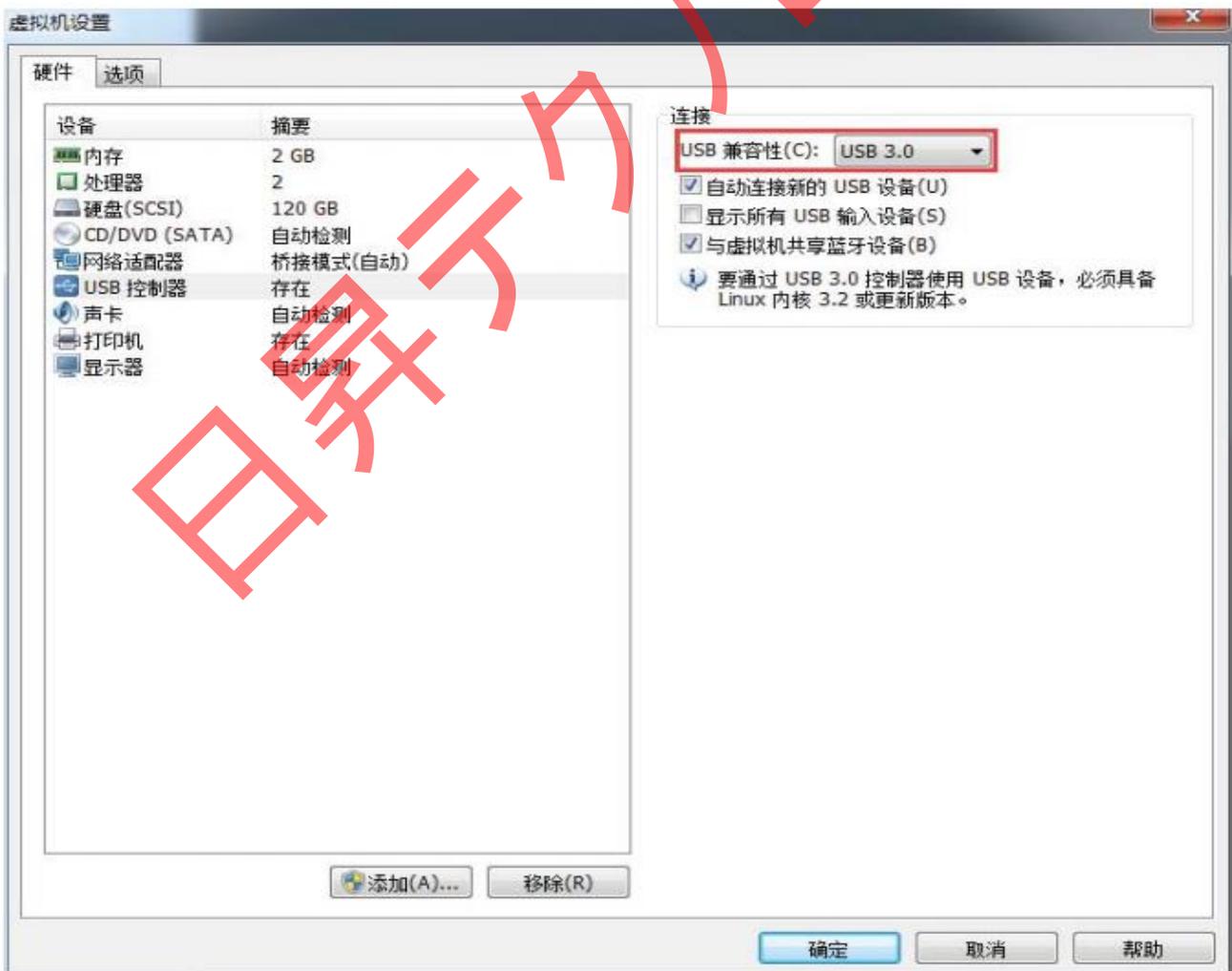
## 15.5 よくある問題

### 15.5.1 Linux ダウンローダーのダウンロード時にプロンプトが表示される

1) ハードウェアのテスト時に、ダウンローダーが見つかるが、エラーが表示される。



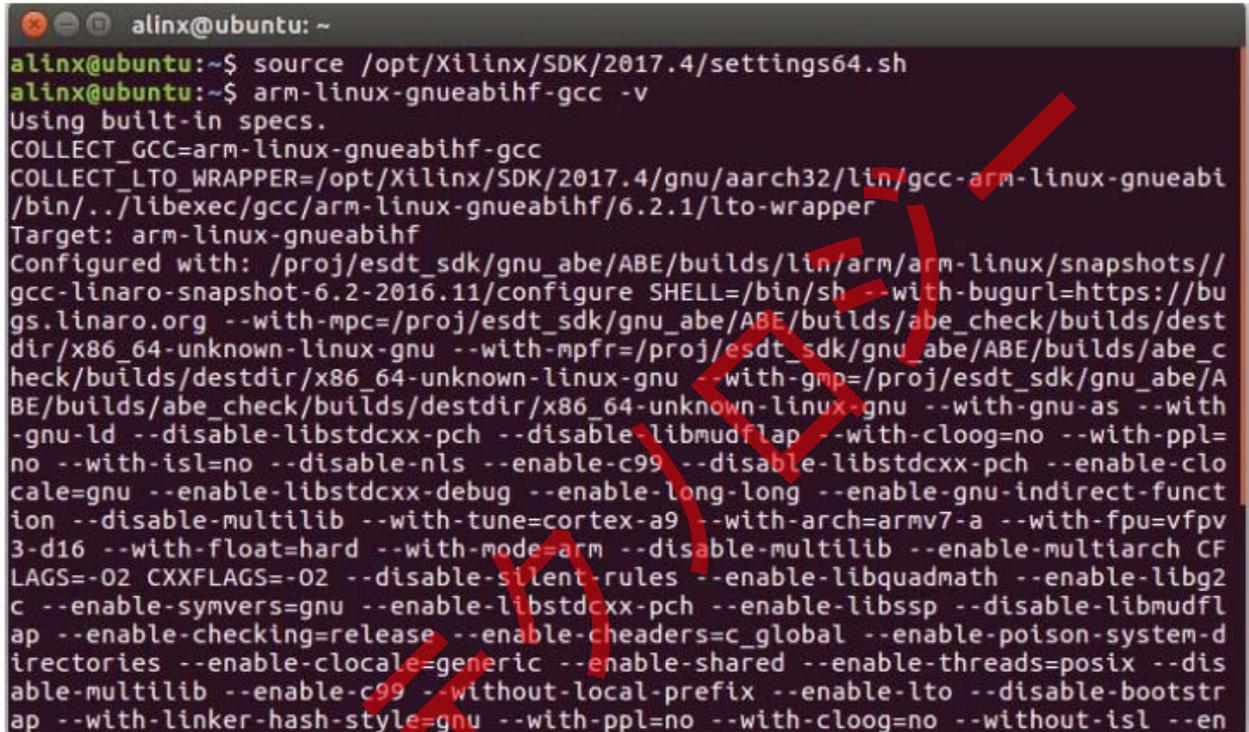
2) 一部のマザーボードでは、USB互換性を設定し、仮想マシンのUbuntuをオフにし、USB互換性をUSB3.0に設定して、もう一度試してください。ダウンローダー使用できない場合は、Windowsバージョンでしかダウンロードできない。



## 15.5.2 ZYNQ に合うクロスコンパイラ

Vivadoをインストール時にSDKをインストールしたため、SDKにはクロスコンパイラarm-linux-gnueabi-hf-gccが含まれている。

```
source /opt/Xilinx/SDK/2017.4/settings64.sh
arm-linux-gnueabi-hf-gcc -v
```



```
alinx@ubuntu: ~
alinx@ubuntu:~$ source /opt/Xilinx/SDK/2017.4/settings64.sh
alinx@ubuntu:~$ arm-linux-gnueabi-hf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabi-hf-gcc
COLLECT_LTO_WRAPPER=/opt/Xilinx/SDK/2017.4/gnu/aarch32/lin/gcc-arm-linux-gnueabi-
/bin/./libexec/gcc/arm-linux-gnueabi-hf/6.2.1/lto-wrapper
Target: arm-linux-gnueabi-hf
Configured with: /proj/esdt_sdk/gnu_abe/ABE/builds/lin/arm/arm-linux/snapshots//
gcc-linaro-snapshot-6.2-2016.11/configure SHELL=/bin/sh --with-bugurl=https://bu
gs.linaro.org --with-mpc=/proj/esdt_sdk/gnu_abe/ABE/builds/abe_check/builds/dest
dir/x86_64-unknown-linux-gnu --with-mpfr=/proj/esdt_sdk/gnu_abe/ABE/builds/abe_c
heck/builds/destdir/x86_64-unknown-linux-gnu --with-gmp=/proj/esdt_sdk/gnu_abe/A
BE/builds/abe_check/builds/destdir/x86_64-unknown-linux-gnu --with-gnu-as --with
-gnu-ld --disable-libstdcxx-pch --disable-libmudflap --with-cloog=no --with-ppl=
no --with-isl=no --disable-nls --enable-c99 --disable-libstdcxx-pch --enable-clo
cale=gnu --enable-libstdcxx-debug --enable-long-long --enable-gnu-indirect-funct
ion --disable-multilib --with-tune=cortex-a9 --with-arch=armv7-a --with-fpu=vfpv
3-d16 --with-float=hard --with-mode=arm --disable-multilib --enable-multiarch C
FLAGS=-O2 CXXFLAGS=-O2 --disable-silent-rules --enable-libquadmath --enable-libg2
c --enable-symvers=gnu --enable-libstdcxx-pch --enable-libssp --disable-libmudfl
ap --enable-checking=release --enable-headers=c_global --enable-poison-system-d
irectories --enable-locale=generic --enable-shared --enable-threads=posix --dis
able-multilib --enable-c99 --without-local-prefix --enable-lto --disable-bootstr
ap --with-linker-hash-style=gnu --with-ppl=no --with-cloog=no --without-isl --en
```

## 第十六章 Petalinux ツールのインストール

### 16.1 Petalinux の概要

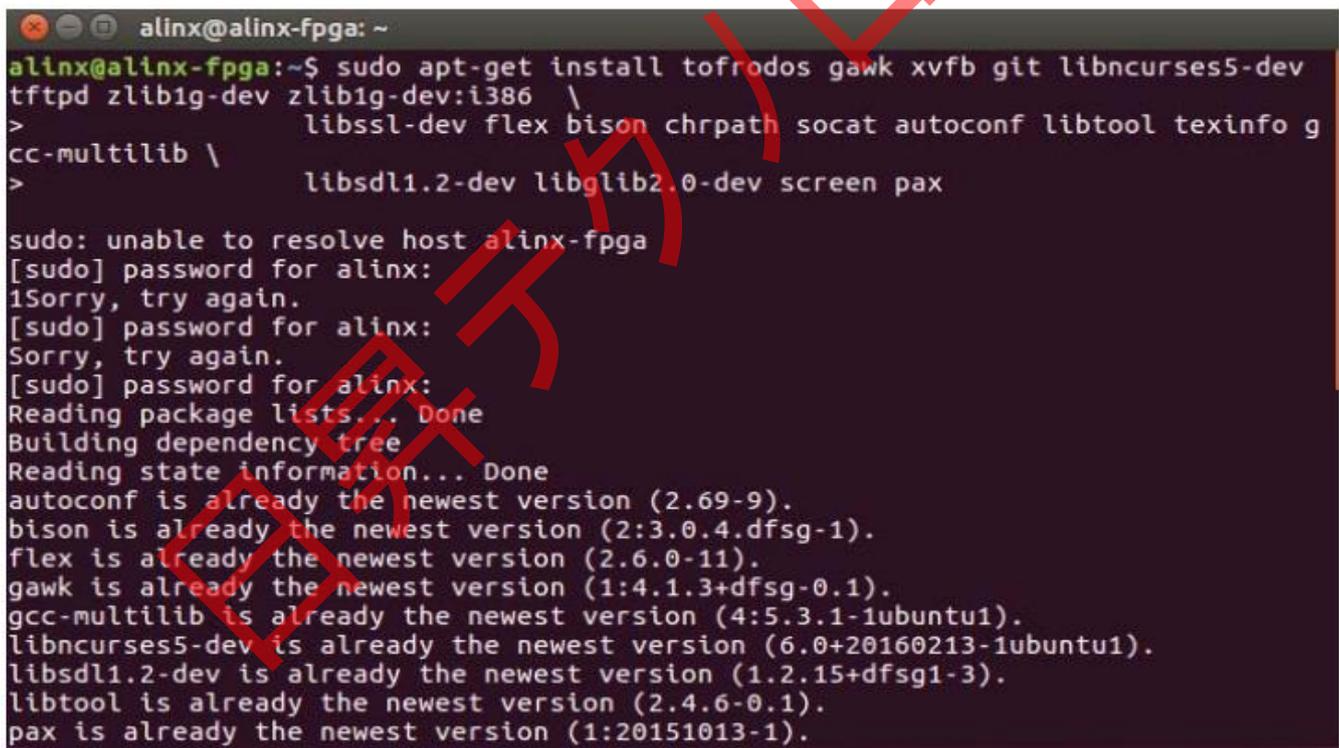
Petalinuxは特別なLinuxカーネルではなく、開発環境設定用のツールセットであり、uboot、カーネル、及びルートファイルシステムの設定作業負荷を軽減できる。Vivadoのエクスポートハードウェア情報から関連ソフトウェアを自動的に設定できる。

特に注意が必要なのは、Petalinuxがシステムバージョンと設定に厳しいこと。その他のバージョンでシステムを操作する場合、問題があったらご自身で解決してください。

### 16.2 インストールに必要なライブラリ

- 1) 以下のコマンドを実行して、ライブラリをインストールする

```
sudo apt-get install tofrodos gawk xvfb git libncurses5-dev tftpd zlib1g-dev zlib1g-dev:i386 \
libssl-dev flex bison chrpath socat autoconf libtool texinfo gcc-multilib \
libsdl1.2-dev libglib2.0-dev screen pax
```



```
alinx@alinx-fpga: ~
alinx@alinx-fpga:~$ sudo apt-get install tofrodos gawk xvfb git libncurses5-dev
tftpd zlib1g-dev zlib1g-dev:i386 \
> libssl-dev flex bison chrpath socat autoconf libtool texinfo g
cc-multilib \
> libsdl1.2-dev libglib2.0-dev screen pax

sudo: unable to resolve host alinx-fpga
[sudo] password for alinx:
!Sorry, try again.
[sudo] password for alinx:
Sorry, try again.
[sudo] password for alinx:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version (2.69-9).
bison is already the newest version (2:3.0.4.dfsg-1).
flex is already the newest version (2.6.0-11).
gawk is already the newest version (1:4.1.3+dfsg-0.1).
gcc-multilib is already the newest version (4:5.3.1-1ubuntu1).
libncurses5-dev is already the newest version (6.0+20160213-1ubuntu1).
libsdl1.2-dev is already the newest version (1.2.15+dfsg1-3).
libtool is already the newest version (2.4.6-0.1).
pax is already the newest version (1:20151013-1).
```

- 2) Tftp serverを設定する。TFTPから起動する必要がない場合、この手順は選択せきる。

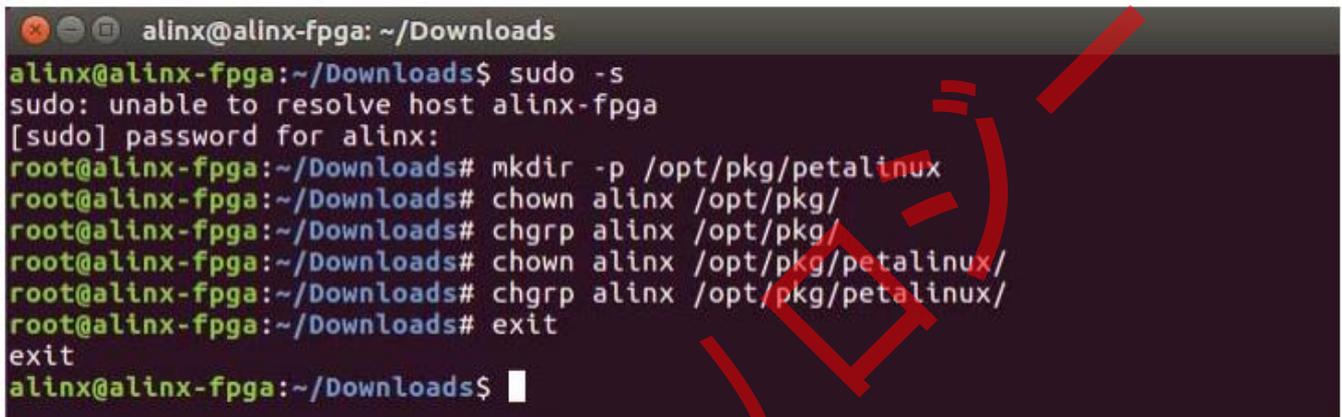
```
sudo -s
apt-get install tftpd-hpa
chmod a+w /var/lib/tftpboot/
```

```
reboot
```

### 16.3 Petalinux をインストールする

- 1) インストールのためにコマンドを実行、ユーザー名は「your\_user\_name」、たとえば、下の画像のalinx。

```
sudo -s
mkdir -p /opt/pkg/petalinux
chown <your_user_name> /opt/pkg/
chgrp <your_user_name> /opt/pkg/
chgrp <your_user_name> /opt/pkg/petalinux/
chown <your_user_name> /opt/pkg/petalinux/
exit
```



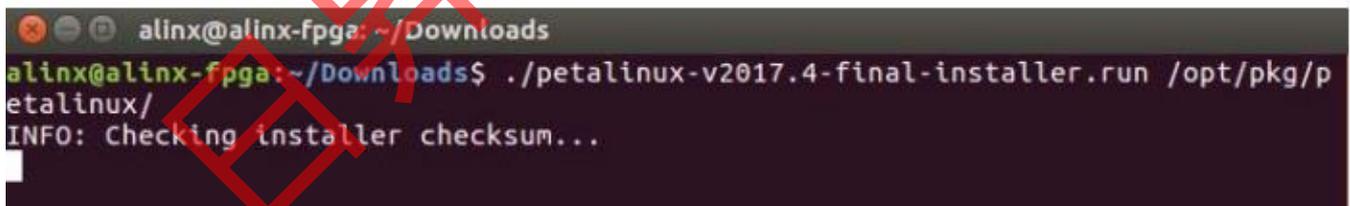
```
alinx@alinx-fpga: ~/Downloads
alinx@alinx-fpga:~/Downloads$ sudo -s
sudo: unable to resolve host alinx-fpga
[sudo] password for alinx:
root@alinx-fpga:~/Downloads# mkdir -p /opt/pkg/petalinux
root@alinx-fpga:~/Downloads# chown alinx /opt/pkg/
root@alinx-fpga:~/Downloads# chgrp alinx /opt/pkg/
root@alinx-fpga:~/Downloads# chown alinx /opt/pkg/petalinux/
root@alinx-fpga:~/Downloads# chgrp alinx /opt/pkg/petalinux/
root@alinx-fpga:~/Downloads# exit
exit
alinx@alinx-fpga:~/Downloads$
```

- 2) 実行許可をインストールファイルに追加する。もちろんpetalinux-v2017.4-final-installer.runこのファイルは最初にシステムにコピーする必要がある

```
sudo chmod +x petalinux-v2017.4-final-installer.run
```

- 3) インストールを開始する

```
./petalinux-v2017.4-final-installer.run /opt/pkg/petalinux/
```



```
alinx@alinx-fpga: ~/Downloads
alinx@alinx-fpga:~/Downloads$ ./petalinux-v2017.4-final-installer.run /opt/pkg/petalinux/
INFO: Checking installer checksum...
```

- 4) Enterキーを押してプロトコルコンテンツを確認できる

```
linux/  
INFO: Checking installer checksum..  
INFO: Extracting PetaLinux installer..  
  
LICENSE AGREEMENTS  
  
PetaLinux SDK contains software from a number of sources. Please review  
the following licenses and indicate your acceptance of each to continue.  
  
You do not have to accept the licenses, however if you do not then you may  
not use PetaLinux SDK.  
  
Use PgUp/PgDn to navigate the license viewer, and press 'q' to close  
Press Enter to display the license agreements
```

- 5) qを押してプロトコルコンテンツを終了する

```
XILINX, INC.  
END USER LICENSE AGREEMENT FOR PETALINUX TOOLS  
  
CAREFULLY READ THIS END USER LICENSE AGREEMENT FOR PETALINUX TOOLS ("AGREEMENT")  
. BY CLICKING THE "ACCEPT" OR "AGREE" BUTTON, ENTERING <93>YES<94> OR <93>Y<94>  
TO ACCEPT THIS AGREEMENT, OR OTHERWISE ACCESSING, DOWNLOADING, INSTALLING OR US  
ING THE SOFTWARE, YOU AGREE ON BEHALF OF LICENSEE TO BE BOUND BY THIS AGREEMENT.  
  
IF LICENSEE DOES NOT AGREE TO ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT,  
DO NOT CLICK THE "ACCEPT" OR "AGREE" BUTTON, ENTER <93>YES<94> OR <93>Y<94>, OR  
ACCESS, DOWNLOAD, INSTALL OR USE THE SOFTWARE.  
  
1. Definitions  
  
"Bitstream" means a machine-executable, binary form of a core used to program a  
Xilinx Device.  
  
"Licensee" means the individual, corporation or other legal entity who has downl  
oaded and installed the Software.  
  
"User" means a specific human being who is identified by Licensee as a person wh  
o is authorized to use the applicable Software on behalf of Licensee. In cases  
/tmp/tmp.Wt4jhy0kpJ/./etc/License/Petalinux_EULA.txt
```

- 6) yを押して同意する

```
linux/  
INFO: Checking installer checksum..  
INFO: Extracting PetaLinux installer..  
  
LICENSE AGREEMENTS  
  
PetaLinux SDK contains software from a number of sources. Please review  
the following licenses and indicate your acceptance of each to continue.  
  
You do not have to accept the licenses, however if you do not then you may  
not use PetaLinux SDK.  
  
Use PgUp/PgDn to navigate the license viewer, and press 'q' to close  
Press Enter to display the license agreements  
Do you accept Xilinx End User License Agreement? [y/N] >
```

- 7) License1はインストールプロセス中にポップアップ表示され、qを押して終了、yを押して同意する

## WebTalk Terms and Conditions

By indicating I accept this WebTalk notice, I also confirm that I have read Section 13 of the terms and conditions above concerning WebTalk and have been afforded the opportunity to read the WebTalk FAQ posted at <http://www.xilinx.com/webtalk>. I understand that I am able to disable WebTalk later if certain criteria described in Section 13(c) apply. If they don't apply, I can disable WebTalk by uninstalling the Software or using the Software on a machine not connected to the internet. If I fail to satisfy the applicable criteria or if I fail to take the applicable steps to prevent such transmission of information, I agree to allow Xilinx to collect the information described in Section 13(a) for the purposes described in Section 13(b).

```
/tmp/tmp.Wt4jhy0kpU/./etc/license/WebTalk_notice.txt (END)
```

```
INFO: Checking installer checksum...  
INFO: Extracting PetaLinux installer...
```

## LICENSE AGREEMENTS

PetaLinux SDK contains software from a number of sources. Please review the following licenses and indicate your acceptance of each to continue.

You do not have to accept the licenses, however if you do not then you may not use PetaLinux SDK.

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close

Press Enter to display the license agreements

```
Do you accept Xilinx End User License Agreement? [y/N] > y
```

```
Do you accept Webtalk Terms and Conditions? [y/N] > y
```

```
Do you accept Third Party End User License Agreement? [y/N] > y
```

```
INFO: Checking installation environment requirements...
```

```
INFO: Checking free disk space
```

```
INFO: Checking installed tools
```

```
INFO: Checking installed development libraries
```

```
INFO: Checking network and other services
```

```
INFO: Installing PetaLinux...
```

```
|
```

## 第十七章 NFS サービスソフトウェアのインストール

NFS (Network FileSystem、インターネットファイルシステム) はSUN会社が開発した技術であり、1984年に導入され、異なるマシンや異なるオペレーティングシステムでファイルを相互に共有する技術である。NFSはもともと異なるシステム間で使用するよう設計されていたため、その通信プロトコル設計はホストオペレーティングシステムとは関係ない。

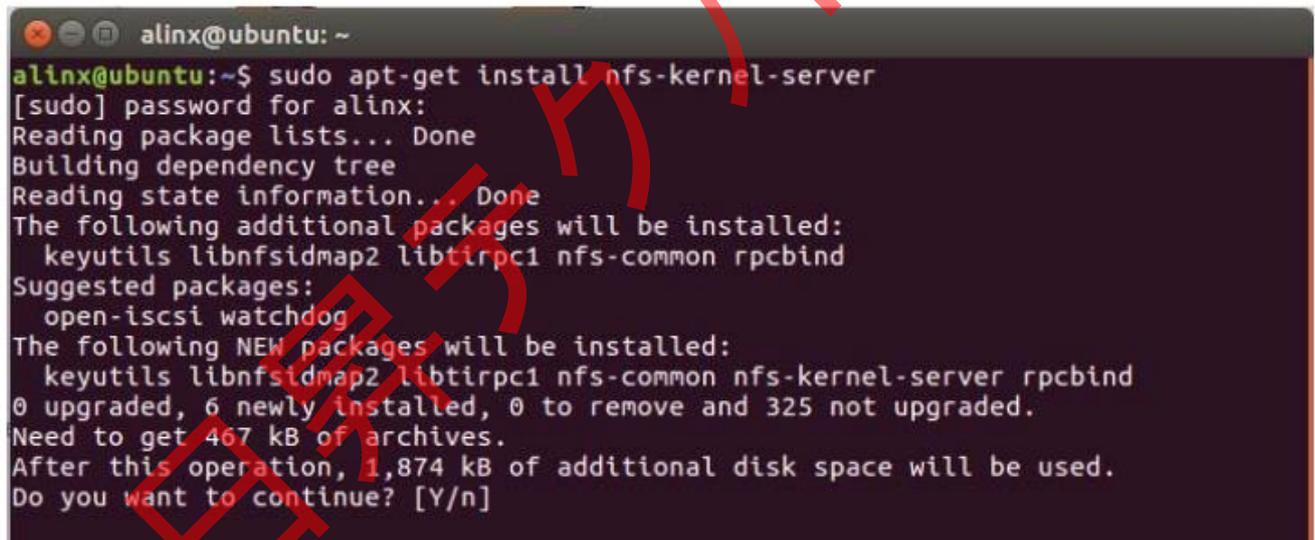
NFSサーバーとクライアントは、リモートファイルを使用する場合、mountコマンドを使用して、ローカルファイルシステムの下のリモートNFSサーバーにファイルシステムをマウントできる。NFSサーバーによって共有されるファイルまたはディレクトリは、/ etc / exportsファイルに記録されている。

組み込みLinux開発では、NFSがよく使用されるが、通常、ターゲットシステムはNFSクライアントとして使用され、LinuxホストはNFSサーバーとして使用されている。ターゲットシステムで、NFSを介してサーバーのNFS共有ディレクトリをローカルデバイスにマウントし、サーバー上のファイルを直接に実行できる。システムドライバーモジュールとアプリケーションのデバッグには、NFSが非常に必要である。また、Linuxは、リモートNFSルートからシステムを直接起動できるし、NFSルートファイルシステムもサポートできる。また、組み込みLinuxルートファイルシステムの調整および統合にも必要がある。

### 17.1 NFS サーバーをインストールする

- 1) 次のコマンドでNFS サーバーをインストールする

```
sudo apt-get install nfs-kernel-server
```



```
alinx@ubuntu: ~
alinx@ubuntu:~$ sudo apt-get install nfs-kernel-server
[sudo] password for alinx:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  keyutils libnfsidmap2 libtirpc1 nfs-common rpcbind
Suggested packages:
  open-iscsi watchdog
The following NEW packages will be installed:
  keyutils libnfsidmap2 libtirpc1 nfs-common nfs-kernel-server rpcbind
0 upgraded, 6 newly installed, 0 to remove and 325 not upgraded.
Need to get 467 kB of archives.
After this operation, 1,874 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

- 2) NFSの作業ディレクトリとして新しい作業ディレクトリを作成する。将来、このディレクトリにクロスコンパイルされたプログラムを配置できる。開発ボードはこのディレクトリ内のファイルを簡単に共有できる。



- 3) 次のコマンドを使用して / etc / exports ファイルを編集し、NFS サービスパスをコンフィグする。

```
sudo gedit /etc/exports
```

```
alinx@ubuntu: ~
alinx@ubuntu:~$ sudo gedit /etc/exports
[sudo] password for alinx:
(gedit:60516): IBUS-WARNING **: The owner of /home/alinx/.config/ibus/bus is not root!
```

- 4) 末尾に / home / alinx / work \* (rw, sync, no\_root\_squash, no\_subtree\_check) を追加し、NFS 作業ディレクトリに / home / alinx / work ディレクトリを作成する。

```
exports
/etc
# /etc/exports: the access control list for filesystems which may be exported
# to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/home/alinx/work *(rw,sync,no_root_squash,no_subtree_check)|
```

- 5) 次のコマンドを実行して、rpcbind サービスを再起動する。Nfs は RPC プログラムで、使用する前にポートをマップして rpcbind で設定する必要がある。

```
sudo /etc/init.d/rpcbind restart
```

- 6) 次のコマンドを実行して nfs サービスを再起動する。

```
sudo /etc/init.d/nfs-kernel-server restart
```

## 17.2 NFS をテストする

- 1) 次のコマンドでNFSをマウントし、NFS作業パスを/ mntディレクトリにマウントする。

```
mount -t nfs 127.0.0.1:/home/alinx/work /mnt
```

- 2) / mntと入力して、新しテストディレクトリtestを作成する。/home/alinx/workディレクトリにtestフォルダーが表示される。

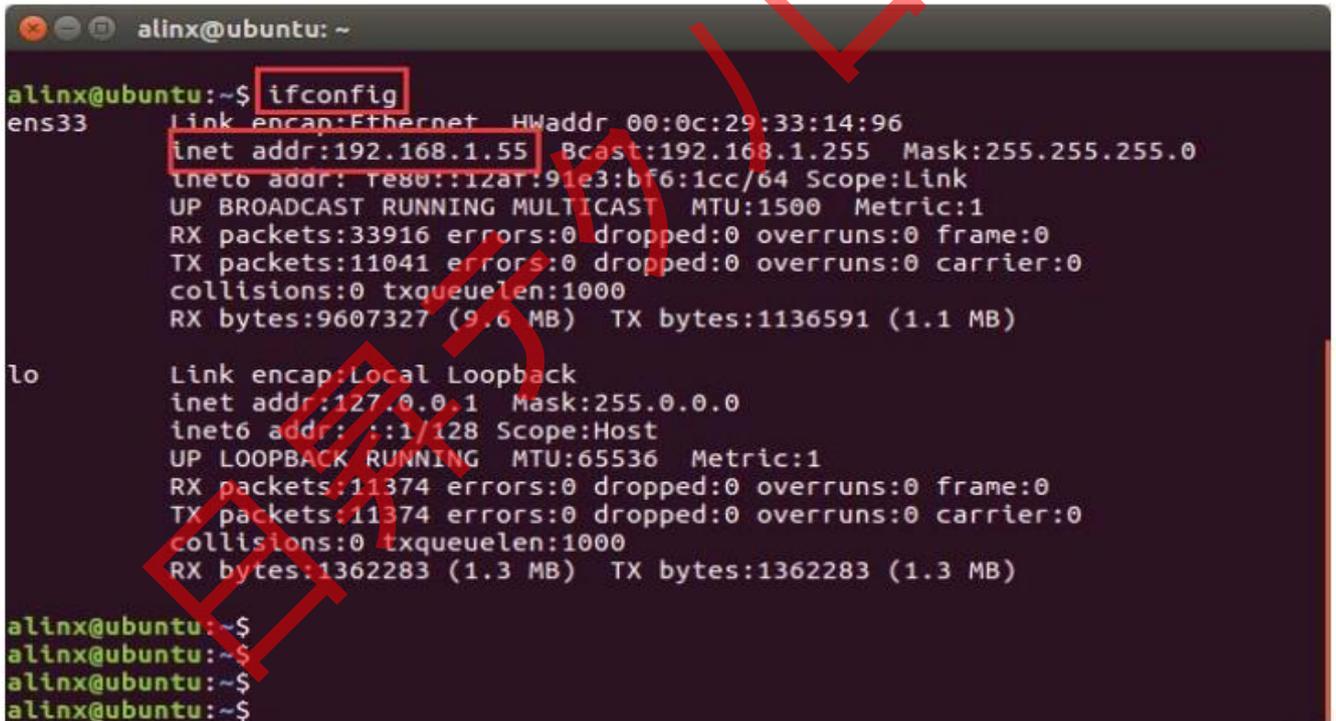
```
cd /mnt  
mkdir test
```

## 17.3 よくある問題

### 17.3.1 NFS マウントできない

最初に、仮想マシンと開発ボードがネットワークセグメントであるかどうかを確認する。

ifconfigコマンドを使用して、仮想マシンのIPアドレスを表示する。次の図の例は、192.168.1.55、192.168.1.55ネットワークセグメントである。開発環境にはDHCCPサーバーがあるため、仮想マシンのIPアドレスが自動的に割り当てられる。このテキストでは、ネットワーク環境が異なるため、ネットワークのコンフィグ方法については説明しない。ネットワークをコンフィグできない場合は、ネットワーク管理者に相談してください。



```
alinx@ubuntu: ~  
alinx@ubuntu:~$ ifconfig  
ens33  Link encap:Ethernet  HWaddr 00:0c:29:33:14:96  
       inet addr:192.168.1.55  Bcast:192.168.1.255  Mask:255.255.255.0  
       inet6 addr: fe80::12a7:91e3:bf6:1cc/64  Scope:Link  
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
       RX packets:33916  errors:0  dropped:0  overruns:0  frame:0  
       TX packets:11041  errors:0  dropped:0  overruns:0  carrier:0  
       collisions:0  txqueuelen:1000  
       RX bytes:9607327 (9.6 MB)  TX bytes:1136591 (1.1 MB)  
  
lo     Link encap:Local Loopback  
       inet addr:127.0.0.1  Mask:255.0.0.0  
       inet6 addr: ::1/128  Scope:Host  
       UP LOOPBACK RUNNING  MTU:65536  Metric:1  
       RX packets:11374  errors:0  dropped:0  overruns:0  frame:0  
       TX packets:11374  errors:0  dropped:0  overruns:0  carrier:0  
       collisions:0  txqueuelen:1000  
       RX bytes:1362283 (1.3 MB)  TX bytes:1362283 (1.3 MB)  
  
alinx@ubuntu:~$  
alinx@ubuntu:~$  
alinx@ubuntu:~$  
alinx@ubuntu:~$
```

シリアル端末でifconfigコマンドを使用して、開発ボードのIPアドレスが表示される。次の図の例は、192.168.1、ネットワークセグメントは192.168.1.46である。IPがない場合、または開発ボードIPに異なるネットワークセグメントがある場合は、ネットワーク管理者に連絡してください。

```
root@zynq: ~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0a:35:00:1e:53
          inet addr:192.168.1.46  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe00:1e53/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4597 (4.4 KiB)  TX bytes:3782 (3.6 KiB)
          Interrupt:29 Base address:0xb000

eth1      Link encap:Ethernet  HWaddr 00:0a:35:00:03:22
          UP BROADCAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1104 (1.0 KiB)  TX bytes:1104 (1.0 KiB)

root@zynq: ~#
```

シリアルターミナルで仮想マシンにpingし、サンプルでpingする。192.168.1.55 これは、仮想マシンのIPが192.168.1.155であり、これをpingしてNFSを通常にマウントできる。

```
root@zynq: ~# ping 192.168.1.55
PING 192.168.1.55 (192.168.1.55) 56(84) bytes of data.
64 bytes from 192.168.1.55: icmp_seq=1 ttl=64 time=0.862 ms
64 bytes from 192.168.1.55: icmp_seq=2 ttl=64 time=0.489 ms
64 bytes from 192.168.1.55: icmp_seq=3 ttl=64 time=0.779 ms
64 bytes from 192.168.1.55: icmp_seq=4 ttl=64 time=0.504 ms
64 bytes from 192.168.1.55: icmp_seq=5 ttl=64 time=0.574 ms
^C
--- 192.168.1.55 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4131ms
rtt min/avg/max/mdev = 0.489/0.641/0.862/0.153 ms
root@zynq: ~#
```

## 第十八章 Petalinux でLinux システムをカスタマイズする

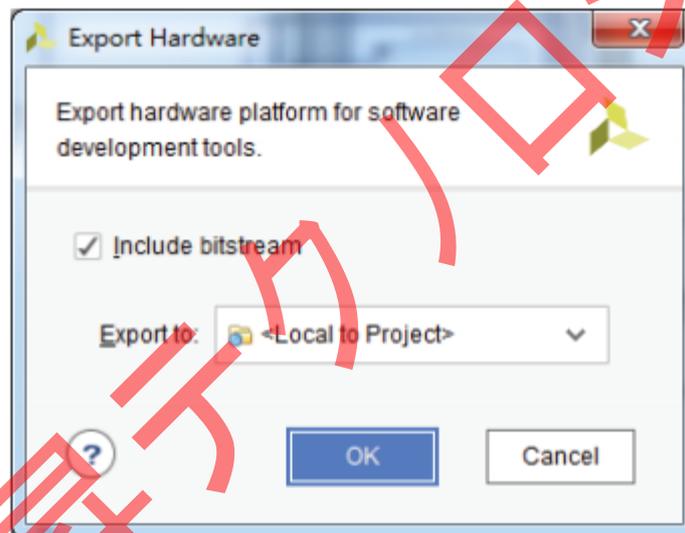
実験Vivadoプロジェクトはlinux\_baseである。

前のマニュアルでは、Petalinux環境を構築したが、このマニュアルでは主にPetalinuxの使用方法を示す。この実験は、Linuxホストがインターネットに接続できた場合のみ完了できる。

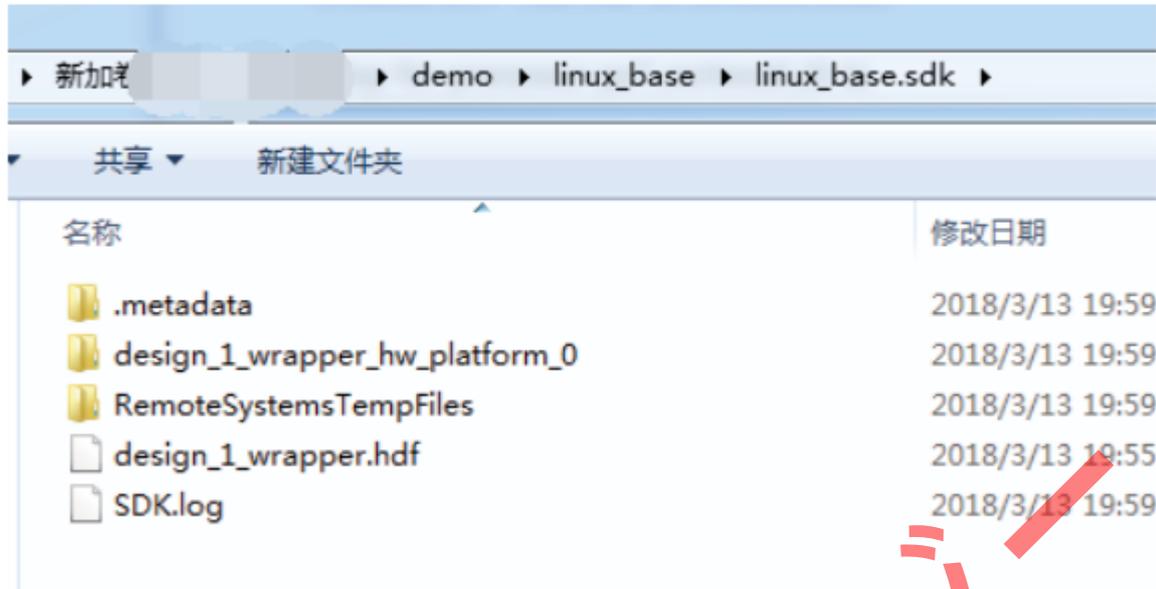
### 18.1 Vivado プロジェクト

Petalinuxを使用すると、組み込みLinuxシステムをカスタマイズするのに非常に便利である。Vivadoソフトウェアがハードウェア情報をエクスポートするだけで、その後、Petalinuxはこの情報に従ってuboot、カーネル、ファイルシステムなどをコンフィグする。vivadoプロジェクトの確立手順は、前の実験で説明したため、ここでは説明しない。

- 1) Bitファイルのコンパイルと生成
- 2) ハードウェア情報をエクスポートする



- 3) vivadoのプロジェクトディレクトリに\* .sdkディレクトリがあり、下に\* .hdfファイルがある。このファイルには、petalinuxが使用するファイルが含まれている。



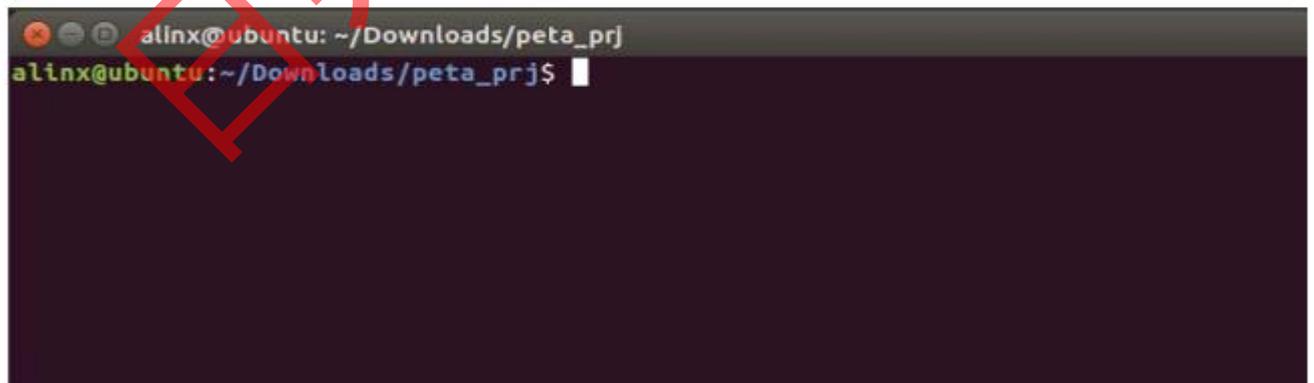
## 18.2 Petalinux でプロジェクトを作成する

- 1) linux\_base.sdkディレクトリをLinuxホストにコピーする



- 2) ターミナルを開き、ディレクトリに入る。

なぜここにpeta\_prjディレクトリがないのか、自分でディレクトリを作成する必要がある。次回マニュアルでディレクトリがあるのに、自分が場合、ご自身で作成してください。



- 3) petalinux環境変数を設定し、次のコマンドを実行する。

```
source /opt/pkg/petalinux/settings.sh
```

```

alinx@ubuntu: ~/Downloads/peta_prj
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and solution
alinx@ubuntu:~/Downloads/peta_prj$
  
```

4) 次のコマンドを実行して、vivado環境変数を設定する。

```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

```

alinx@ubuntu: ~/Downloads/peta_prj
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and solution
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/Downloads/peta_prj$
  
```

5) 次のコマンドで、ax\_petaという名前のpetalinuxプロジェクトを作成すると、petalinuxはax\_petaという名前のプロジェクトが自動的に作成される。 - は二つで、ご注意ください。

```
petalinux-create --type project --template zynq --name ax_peta
```



```

alinx@ubuntu: ~/Downloads/peta_prj
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/Downloads/peta_prj$ petalinux-create --type project --template zynq --name ax_peta
INFO: Create project: ax_peta
INFO: New project successfully created in /home/alinx/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj$
  
```

6) 次のコマンドを使用して, petalinuxディレクトリーに入る。

```
cd ax_peta
```

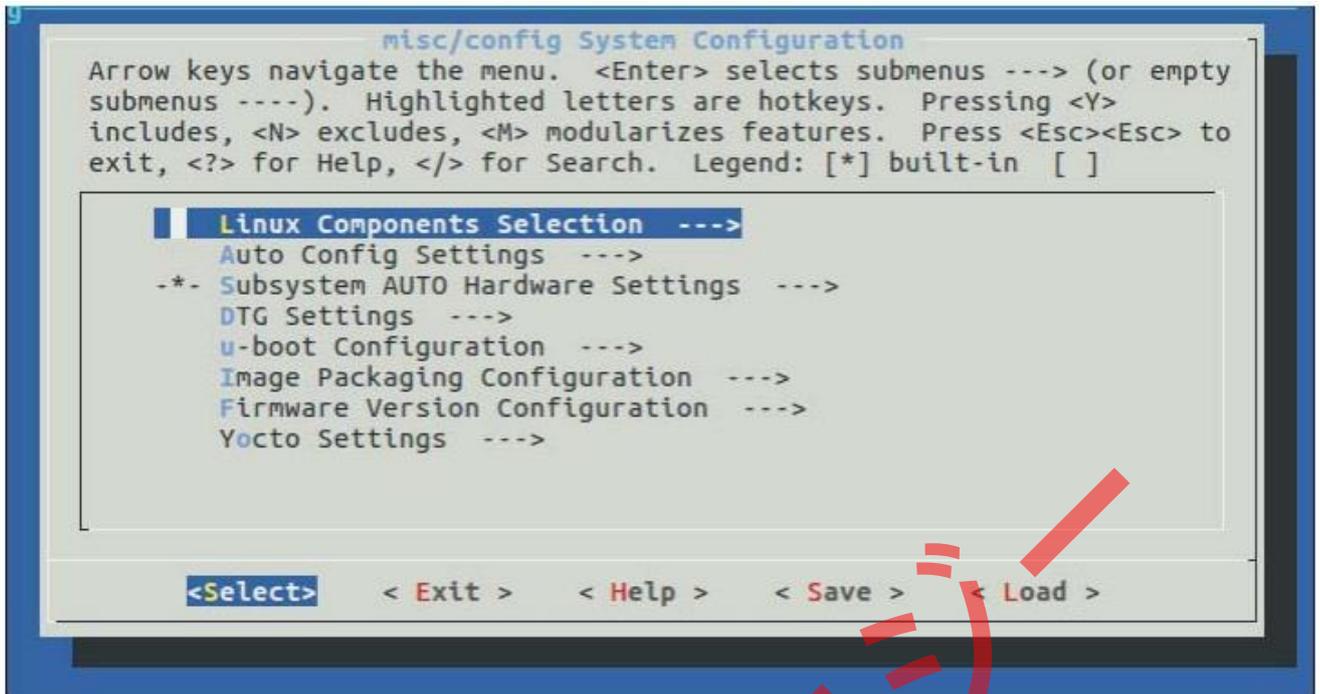
```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and solution
alinx@ubuntu:~/Downloads/peta_prj$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/Downloads/peta_prj$ petalinux-create --type project --template zyng --name ax_peta
INFO: Create project: ax_peta
INFO: New project successfully created in /home/alinx/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj$ cd ax_peta/
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
```

7) 次のコマンドで、Petalinuxプロジェクトのハードウェア情報をコンフィグする。../linux\_base.sdkディレクトリは、vivadoからエクスポートしたハードウェア情報である。

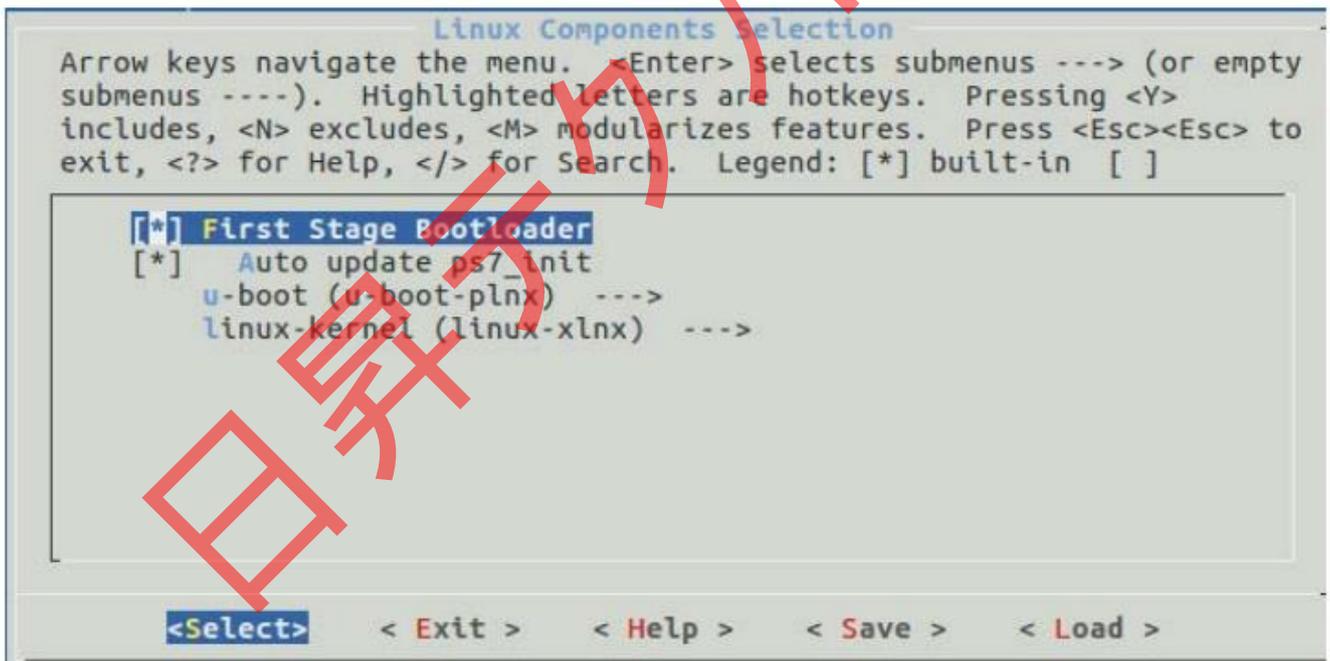
```
petalinux-config --get-hw-description ../linux_base.sdk
```

```
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-config --get-hw-description ../linux_base.sdk
INFO: Getting hardware description...
INFO: Rename design_1_wrapper.hdf to system.hdf
[INFO] generating Kconfig for project
```

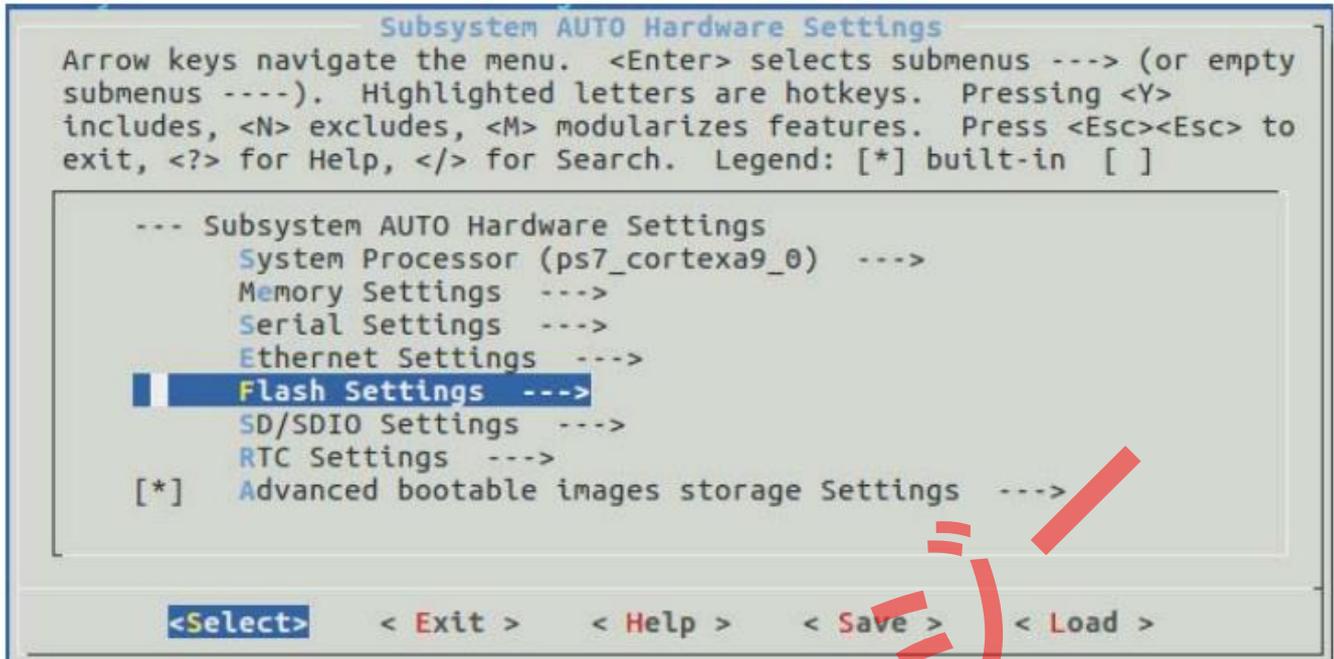
8) ポップアップウィンドウでpetalinuxプロジェクトをコンフィグできる。コンフィグ後に再度コンフィグしたい場合は、コマンドpetalinux-configを実行してコンフィグできる。



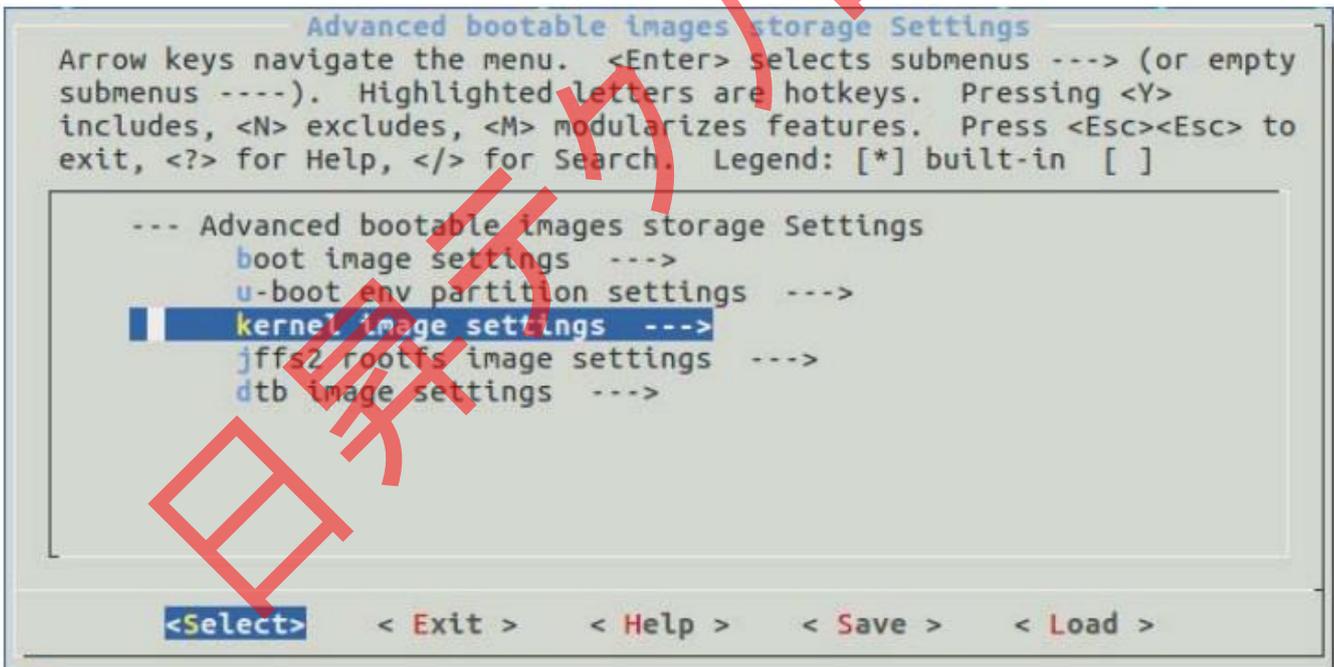
9) u-bootおよびLinuxカーネルのソースは、オプションLinux Components Selectionで設定できる。デフォルトでgithubにダウンロードされる。ダウンロードするにはインターネットとLinuxホストと接続する必要がある。このテストはデフォルト設定のまま。



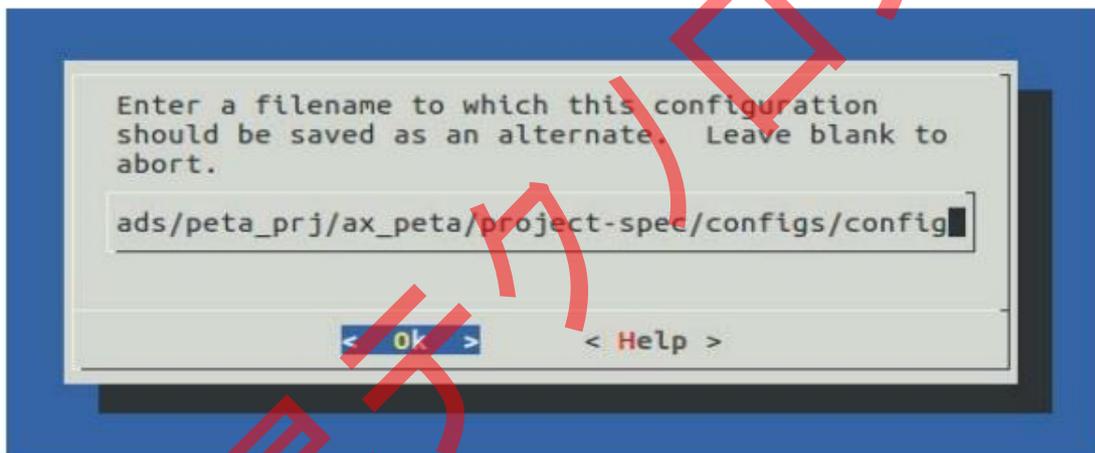
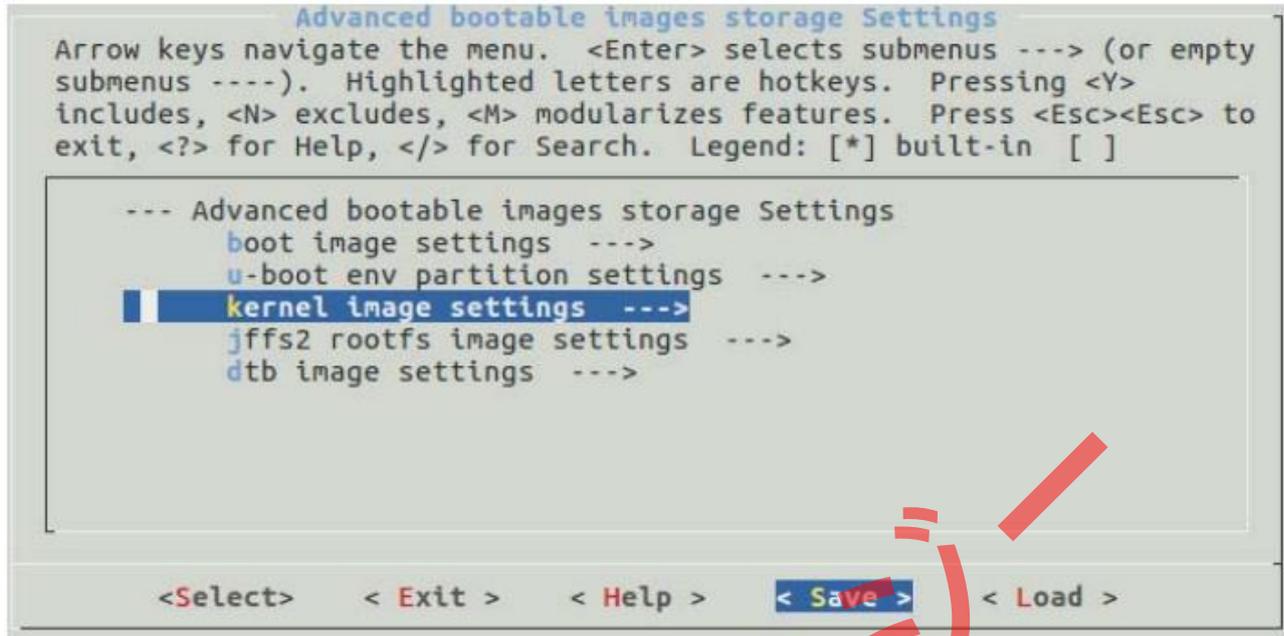
10) 周辺機器と起動モードは、オプションのSubsystem AUTO Hardware Settingsで設定できる。



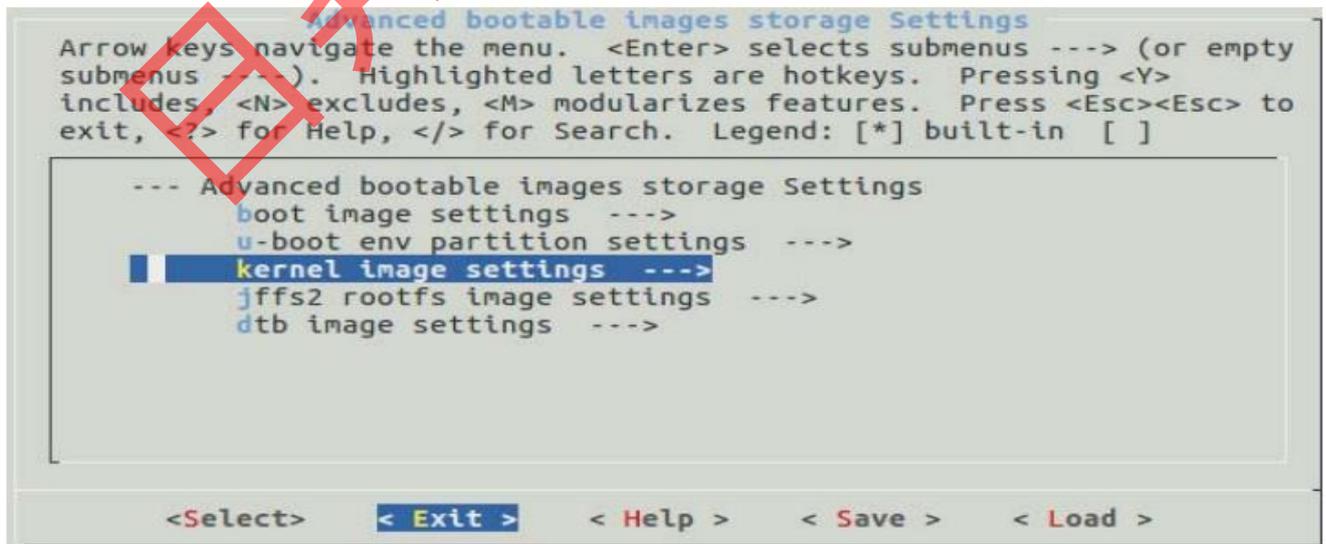
11) Advanced bootable images storage Settingsオプションで、起動モードをコンフィグする。デバッグのため、ここでは、デフォルトでSDカードから起動のまま、もしQSPI flashから起動する組み込みLinuxを作成する場合は、ここでコンフィグできる。



12) 完了した後も設定を維持する。この実験は基本的にデフォルト設定である。



13) コンフィグインターフェースを終了する



14) しばらくお待ちください。

```

alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
INFO: Getting hardware description...
[INFO] generating Kconfig for project

[INFO] menuconfig project
/home/alinx/Downloads/peta_prj/ax_peta/build/misc/config/Kconfig.syshw:30:warning:
 defaults for choice values not supported
/home/alinx/Downloads/peta_prj/ax_peta/build/misc/config/Kconfig:568:warning: co
nfig symbol defined without type

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_p
eta
~/Downloads/peta_prj/ax_peta
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_p
eta
  
```

### 18.3 Linux カーネルをコンフィグする

- 1) 次のコマンドでカーネルをコンフィグする。コマンドを実行したら、少し待ちください。

```
petalinux-config -c kernel
```

```

alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
eta
~/Downloads/peta_prj/ax_peta
[INFO] generating u-boot configuration files

[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta$ petalinux-config -c kernel
[INFO] generating Kconfig for project

[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_p
eta
~/Downloads/peta_prj/ax_peta
[INFO] generating machine configuration
[INFO] configuring: kernel
[INFO] generating kernel configuration files
[INFO] bitbake virtual/kernel -c menuconfig
Parsing recipes: 14% |#####
| ETA: 0:01:29
| ETA: 0:01:31
  
```

- 2) しばらく待ってから、カーネルをコンフィグするためのコンフィグインターフェースがポップアップする。

```

.config - Linux/arm 4.9.0 Kernel Configuration

Linux/arm 4.9.0 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

[*] Patch physical to virtual translations at runtime
  General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
  System Type --->
  Bus support --->
  Kernel Features --->
  Boot options --->
  CPU Power Management --->
  Floating point emulation --->

+ (+)

<Select>  < Exit >  < Help >  < Save >  < Load >
  
```

3) コンフィグを保存して閉じる。

```

.config - Linux/arm 4.9.0 Kernel Configuration
[...] rivers > Network device support > PHY Device support and infrastrucur
PHY Device support and infrastructure
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

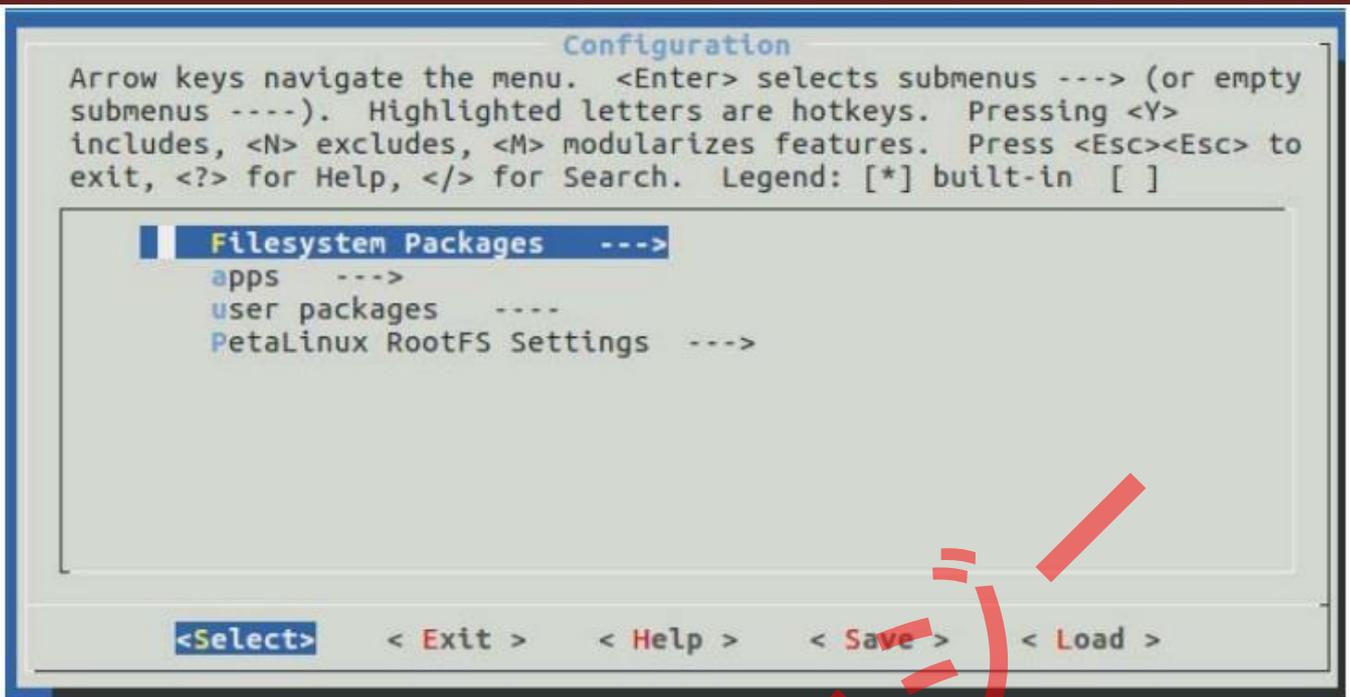
^ (-)
< > Microsemi PHYs
< > National Semiconductor PHYs
< > Quality Semiconductor PHYs
[*] Realtek PHYs
< > MSC PHYs
< > STMicroelectronics STe10Xp PHYs
< > Teranetics PHYs
[*] Vitesse PHYs
[*] Drivers for xilinx PHYs
< > Xilinx GMII2RGMII converter driver

<Select>  < Exit >  < Help >  < Save >  < Load >
  
```

#### 18.4 ルートファイルシステムのコンフィグ

次のコマンドを実行して、ルートファイルシステムがコンフィグできる。この実験ではデフォルトのコンフィグが維持される。

```
petalinux-config -c rootfs
```



## 18.5 コンパイルする

- 1) 次のコマンドで、uboot、カーネル、ルートファイルシステム、デバイスツリーなどをコンパイルする。

petalinux-build

```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
~/Downloads/peta_prj/ax_peta
[INFO] generating machine configuration
[INFO] configuring: rootfs
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] menuconfig rootfs

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] generating petalinux-user-image.bb
[INFO] successfully configured rootfs
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |#####| Time: 0:00:00
Loaded 3257 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2434 cached, 32 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
[initialising tasks: 44% |#####| ETA: 0:00:06
```

- 2) コンパイル完了

```

alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
[INFO] successfully configured rootfs
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |#####| Time: 0:00:00
Loaded 3257 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2434 cached, 32 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####| Time: 0:00:08
Checking sstate mirror object availability: 100% |#####| Time: 0:32:53
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
fsbl-2017.4+gitAUTOINC+77448ae629-r0 do_compile: NOTE: fsbl: compiling from exte
rnal source tree /opt/pkg/petalinux/tools/hsm/data/embeddedsw
NOTE: Tasks Summary: Attempted 2444 tasks of which 1884 didn't need to be rerun
and all succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
NOTE: Failed to copy built images to tftp dir: /tftpboot
[INFO] successfully built project
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
  
```

## 18.6 BOOT ファイルが生成される

次のコマンドを使用して BOOT ファイルが生成される。スペースと - 記号に注意してください。

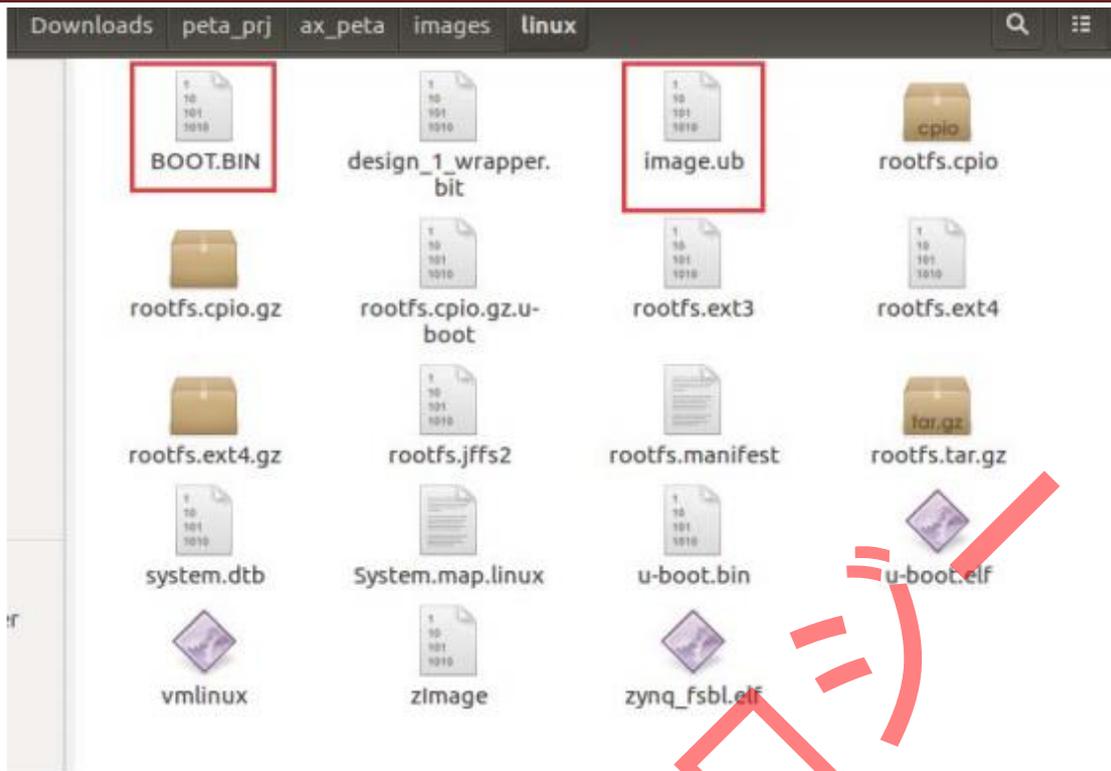
```
petalinux-package --boot --fsbl ./images/linux/zyng_fsbl.elf --fpga --u-boot --force
```

```

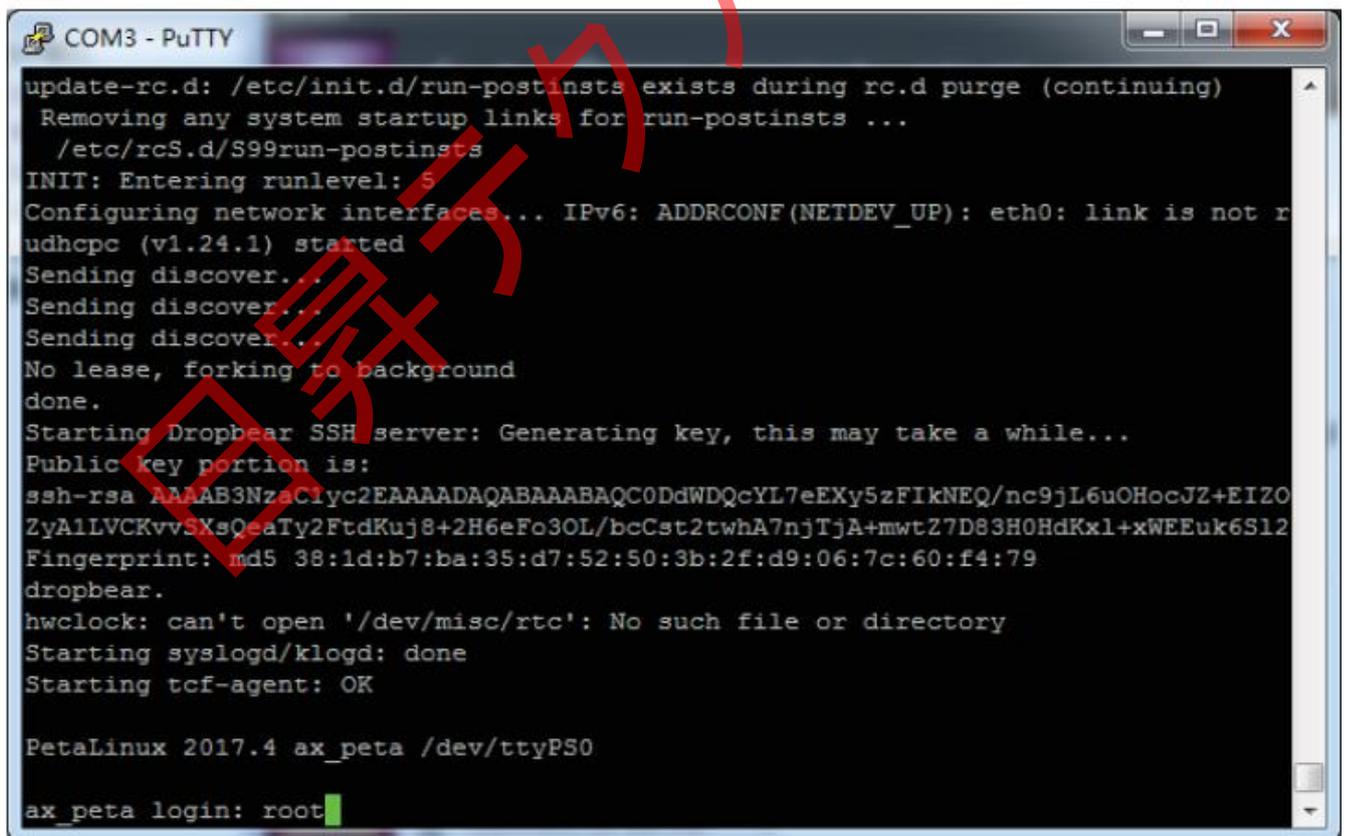
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-package --boot --fsbl ./ima
ges/linux/zyng_fsbl.elf --fpga --u-boot --force
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/zyn
q_fsbl.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/des
ign_1_wrapper.bit"
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/u-b
oot.elf"
INFO: Generating zynq binary package BOOT.BIN...
INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
webtalk failed:Invalid tool in the statistics file:petalinux-yocto!
webtalk failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
  
```

## 18.7 Linux をテストする

1) プロジェクト images → linux ディレクトリの BOOT.BIN と image.ub を SD カードにコピーする。コピーする前に SD カードをフォーマットして、開発ボードに挿入し、開発ボードを SD カードブートに設定する。



- 2) シリアルターミナルを開き、開発ボードを起動する。



- 3) root でログインし、デフォルトでパスワードは root、インターネットケーブルを接続した後（ルーターは自動 IP 取得をサポートする）、ifconfig コマンドを使用してネットワークステータスを確認できる。

```
COM3 - PuTTY
root@ax_peta:~# macb e000b000.ethernet eth0: link up (1000/Full)
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready

root@ax_peta:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:1E:53
          inet addr:192.168.1.46  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe00:1e53%lo/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4350 (4.2 KiB)  TX bytes:1902 (1.8 KiB)
          Interrupt:29 Base address:0xb000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1%1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ax_peta:~#
```

## 18.8 よくある問題

### 18.8.1 Bad FIT kernel image format!が表示され、カーネルが起動できない。

```
U-Boot 2017.01 (Sep 05 2018 - 15:31:52 +0800)

Board: Xilinx Zynq
I2C: ready
DRAM: ECC disabled 1 GiB
MMC: sdhci_transfer_data: Error detected in status(0x208000)!
sdhci@e0100000: 0 (SD), sdhci@e0101000: 1 (eMMC)
SF: Detected w25q256 with page size 256 Bytes, erase size 4 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: ZYNQ GEM: e000b000, phyaddr ffffffff, interface rgmii-id

Warning: ethernet@e000b000 (eth1) using random MAC address - 5a:51:56:3c:ab:16
eth1: ethernet@e000b000
U-BOOT for ax_peta

ethernet@e000b000 Waiting for PHY auto negotiation to complete..... TIMEOUT !
Hit any key to stop autoboot: 0
Device: sdhci@e0100000
Manufacturer ID: 41
OEM: 3432
Name: SD16G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 15 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
reading image.ub
Invalid FAT entry
4096 bytes read in 13 ms (307.6 KiB/s)
# Loading kernel from FIT Image at 10000000 ...
Bad FIT kernel image format!
ERROR: can't get kernel image!
```

解決方法：

SD カードの fat32 を再度フォーマットし、ブートファイルを再配置する

### 18.8.2 ファイルとコンフィグが保存できない

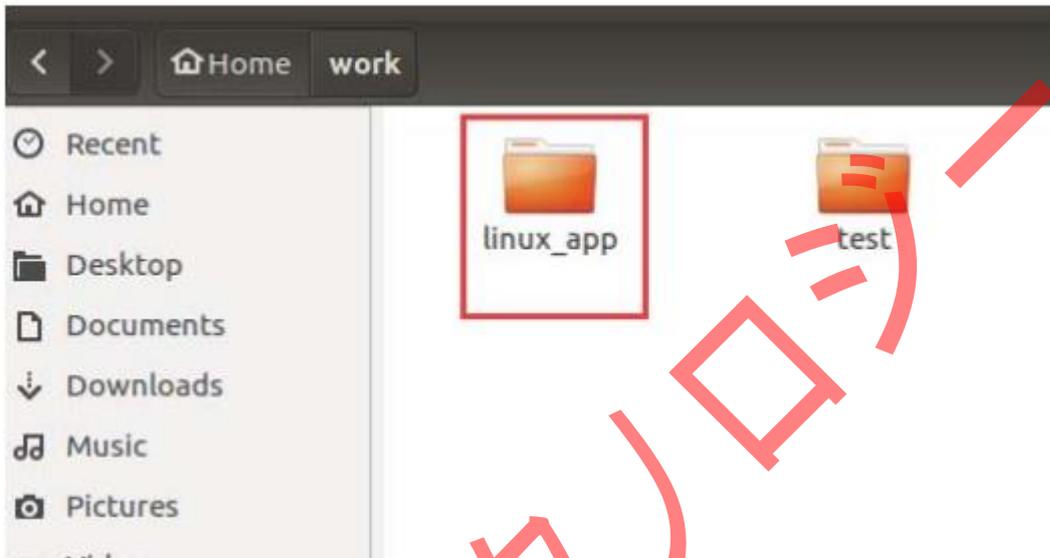
petalinux のデフォルトファイルシステムは RAM タイプであるため、保存できず、次のマニュアルで SD カードタイプに設定し、データは SD カードに保存できる。

## 第十九章 SDK で Linux プログラムを開発する

前のマニュアルでは、petalinux を使用して組み込み Linux システムを作成したが、この実験は Linux アプリケーションを作成し、開発ボード上で実行される。この実験では、上記の実験で Linux オペレーティング環境を使用する必要がある。

### 19.1 SDK を使って Linux アプリケーションを作成する

- 1) SDK ワークスペース用に / home / alinx / work にディレクトリ linux\_app を作成する



- 2) Vivado の環境変数を設定して、SDK を実行する

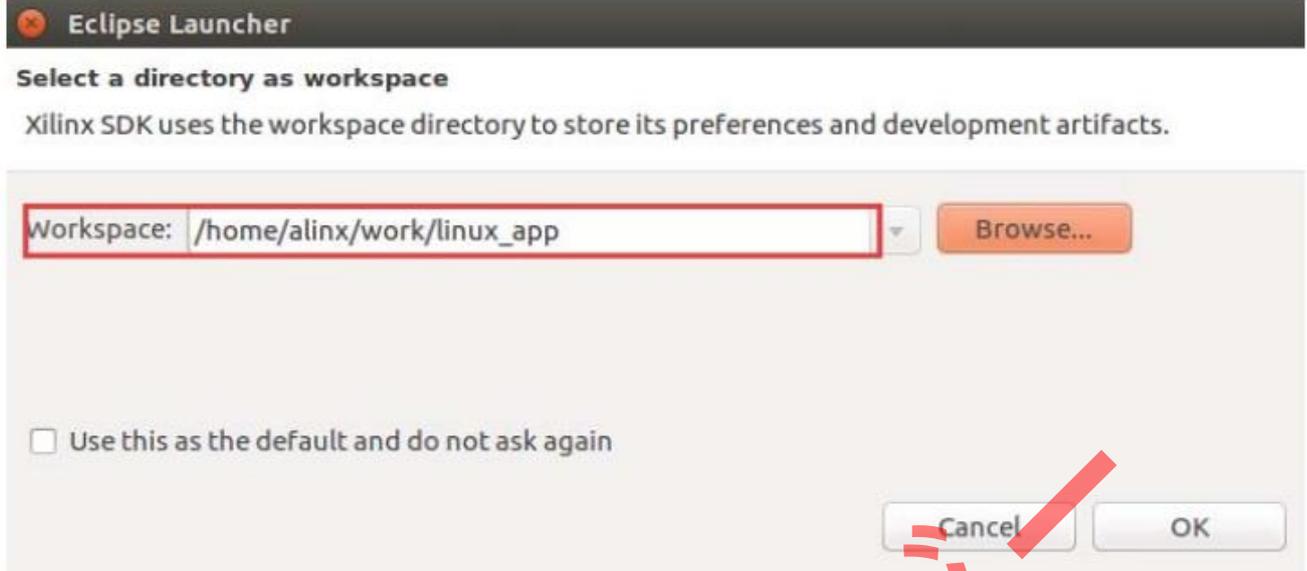
```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
xsdk
```

```
alinx@ubuntu: ~/work
alinx@ubuntu:~/work$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/work$ xsdk

***** Xilinx Software Development Kit
***** SDK v2017.4 (64-bit)
**** SW Build 2086221 on Fri Dec 15 20:54:30 MST 2017
** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

Launching SDK with command /opt/Xilinx/SDK/2017.4/eclipse/lrx64.o/eclipse -vmarg
s -Xms64m -Xmx4G -Dorg.eclipse.swt.internal.gtk.cairoGraphics=false
alinx@ubuntu:~/work$
```

- 3) ワークスペースは /home/alinx/work/linux\_app を選択する



4) Create Application Project を選択する



5) プロジェクト名は hello を入力、OS Platform は Linux を選択する。

**New Project**

**Application Project**  
Create a managed make application project.

Project name:

Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Processor Type:

Endianness:  Little-endian  Big-endian

Target Software

Language:  C  C++

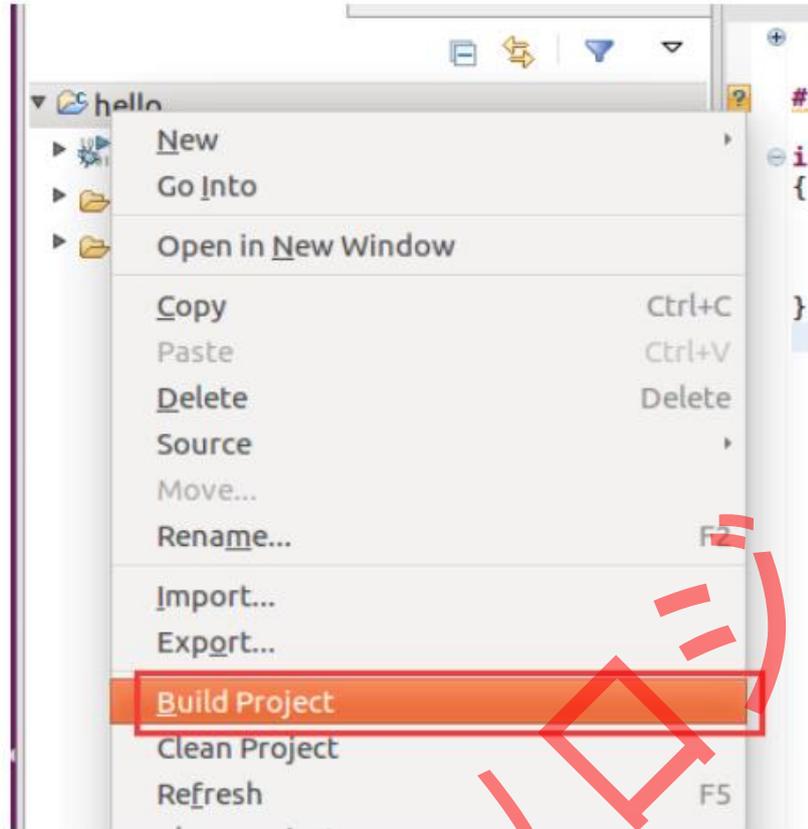
Compiler:

Hypervisor Guest:

Linux System Root:

Linux Toolchain:

6) Build Project



## 19.2 NFS 共有を実行する

1) 開発ポートにネットワークケーブルを挿入し（ルーターは自動的に IP を取得する必要がある）、電源を入れ、Linux ホスト NFS をマウントする。ホスト IP は 192.168.1.77、NFS ディレクトリは /home/alinx/work、/mnt は開発ポートのディレクトリである。ここでは、ホストと開発ポートが同じネットワークセグメント上にある必要がある。

```
mount -t nfs -o nolock 192.168.1.77:/home/alinx/work /mnt
```

```

COM3 - PuTTY
Sending select for 192.168.1.46...
Lease of 192.168.1.46 obtained, lease time 86400
/etc/udhcpd.d/50default: Adding DNS 192.168.1.1
done.
Starting Dropbear SSH server: Generating key, this may take a while...

Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC3fwPFZmbH0qhjkpV+J/zQsBk5nYuFDON9kbBkN7d1
B4MEEC/2aV7AmR0S7SFQ1JZedVCh2YBREBy1mBV2ld+Up125nGQnwwSOLj7T6kiFLU5fxRqDSHfxoOu5
nQq4ck7yS5kgyDbHE/vUD1yyE22JlAwcARDI91VRYCsv/aoumLWoEL3y85VLIwskNPUz8L3sXKICauv
IAshRYbkxc2zzsc4HkMGJZ/NXLZMC527taECp5y2DmXhEUTIDSvpXLH1cFf9PRFzjhaEoJpXgAu3thJ
AGTza2ZPDmghv5MTuAr5KVpgu2yTizcmoMDeZWImdpnh+OtgjU29FrEvvhxl root@ax_peta
Fingerprint: md5 f0:36:14:f7:ad:e9:48:56:87:71:80:b0:d7:f7:dd:6a
dropbear.
hwclock: can't open '/dev/misc/rtc': No such file or directory
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2017.4 ax_peta /dev/ttyPS0

ax_peta login: root
Password:
root@ax_peta:~# mount -t nfs -o nolock 192.168.1.77:/home/alinx/work /mnt
root@ax_peta:~#
  
```

2) ディレクトリ/mnt/linux\_app/hello/Debugに入り、hello.elfを実行する。Hello Worldがプリントアウトしたことが分かる。

```

cd /mnt/linux_app/hello/Debug
./hello.elf
  
```

```

COM3 - PuTTY
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC3fwPFZmbH0qhjkpV+J/zQsBk5nYuFDON9kbBkN7d1
B4MEEC/2aV7AmR0S7SFQ1JZedVCh2YBREBy1mBV2ld+Up125nGQnwwSOLj7T6kiFLU5fxRqDSHfxoOu5
nQq4ck7yS5kgyDbHE/vUD1yyE22JlAwcARDI91VRYCsv/aoumLWoEL3y85VLIwskNPUz8L3sXKICauv
IAshRYbkxc2zzsc4HkMGJZ/NXLZMC527taECp5y2DmXhEUTIDSvpXLH1cFf9PRFzjhaEoJpXgAu3thJ
AGTza2ZPDmghv5MTuAr5KVpgu2yTizcmoMDeZWImdpnh+OtgjU29FrEvvhxl root@ax_peta
Fingerprint: md5 f0:36:14:f7:ad:e9:48:56:87:71:80:b0:d7:f7:dd:6a
dropbear.
hwclock: can't open '/dev/misc/rtc': No such file or directory
Starting syslogd/klogd: done
Starting tcf-agent: OK

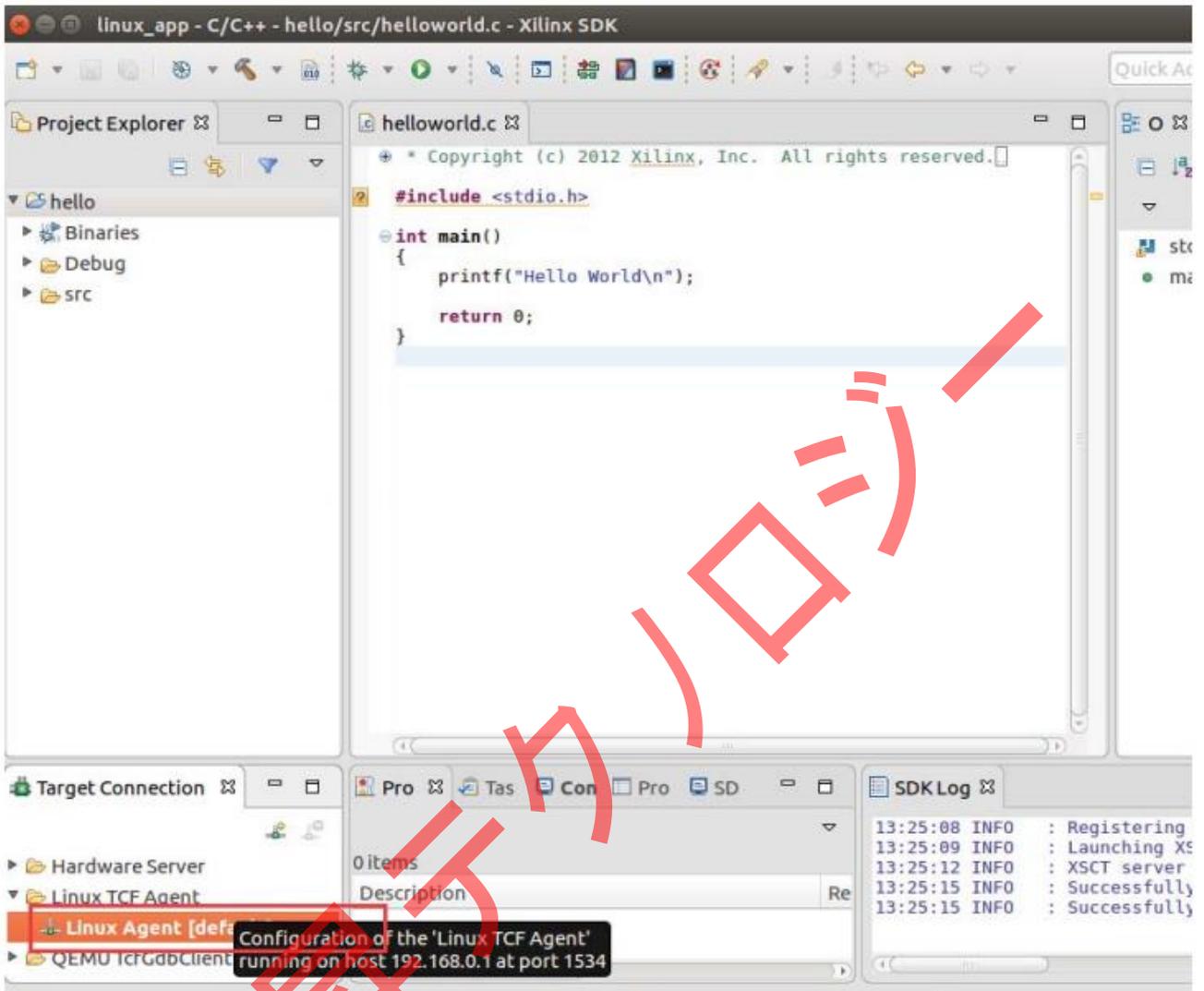
PetaLinux 2017.4 ax_peta /dev/ttyPS0

ax_peta login: root
Password:
root@ax_peta:~# mount -t nfs -o nolock 192.168.1.77:/home/alinx/work /mnt
root@ax_peta:~# cd /mnt
root@ax_peta:/mnt# cd linux_app/hello/Debug/
root@ax_peta:/mnt/linux_app/hello/Debug# random: crng init done

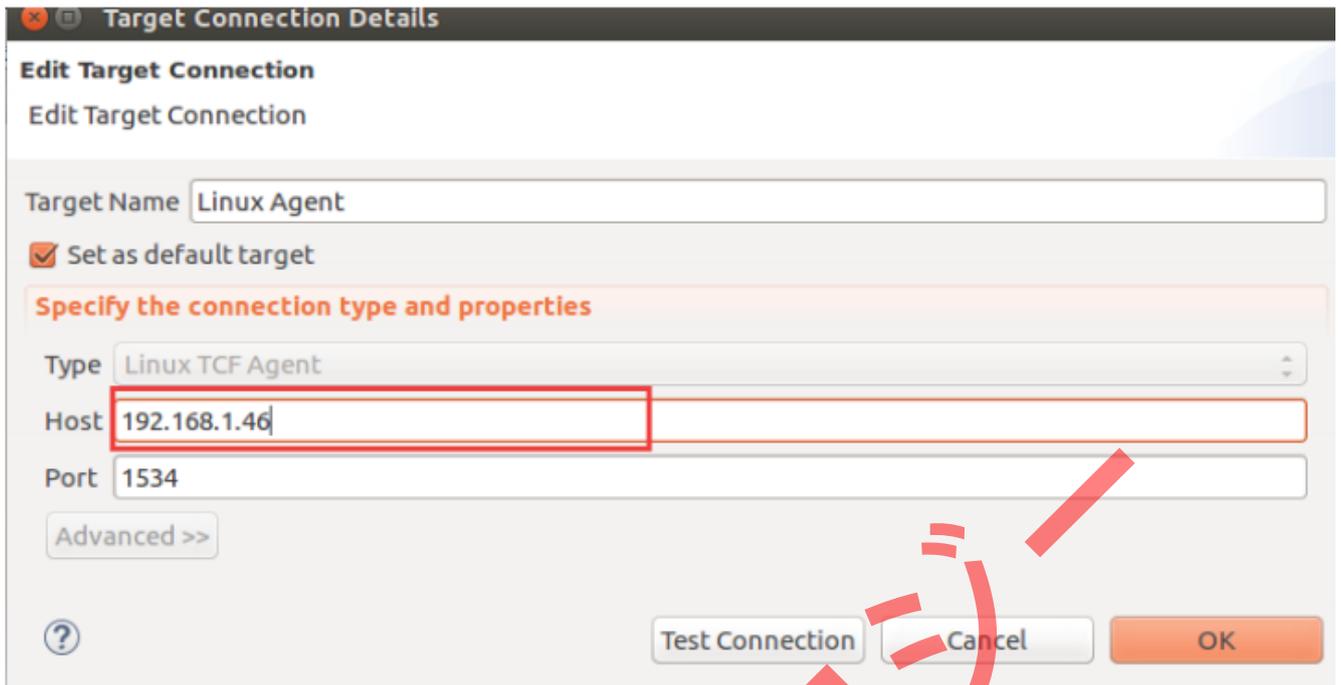
root@ax_peta:/mnt/linux_app/hello/Debug# ./hello.elf
Hello World
root@ax_peta:/mnt/linux_app/hello/Debug#
  
```

### 19.3 TCF-Agent を介してデバッグを実行する

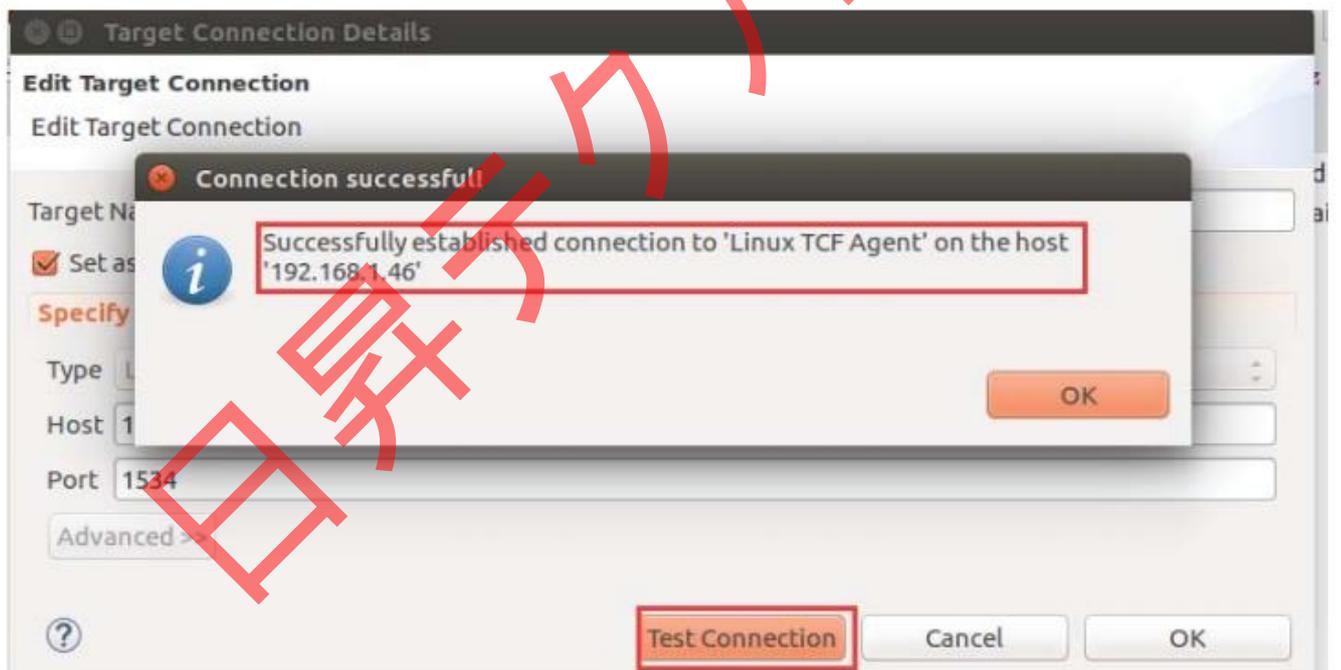
- 1) Linux Agent をダブルクリックして、接続パラメーターをコンフィグする。



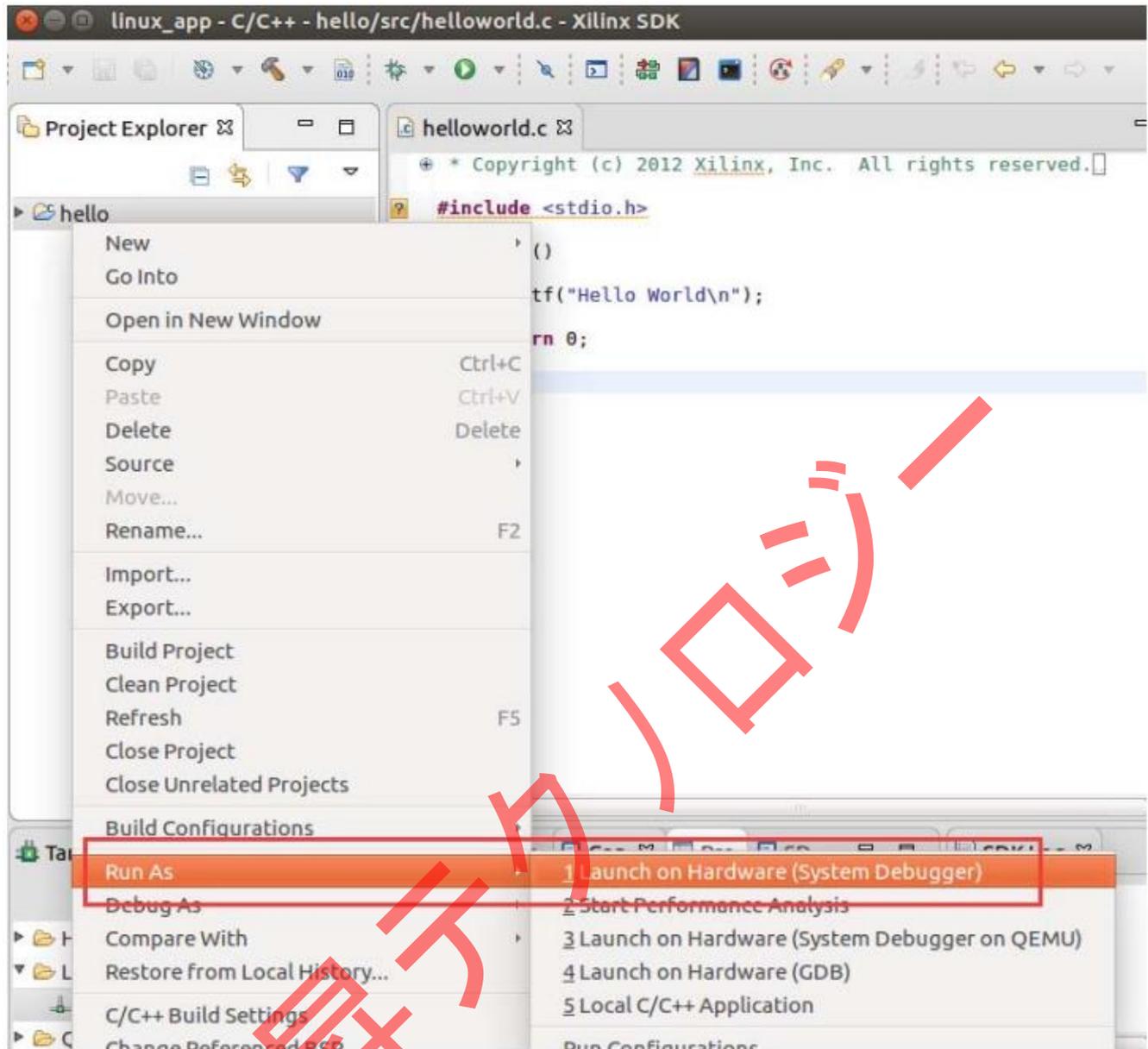
- 2) Host の IP アドレスを入力する。ここで入力するのは開発ポートの IP アドレスである。



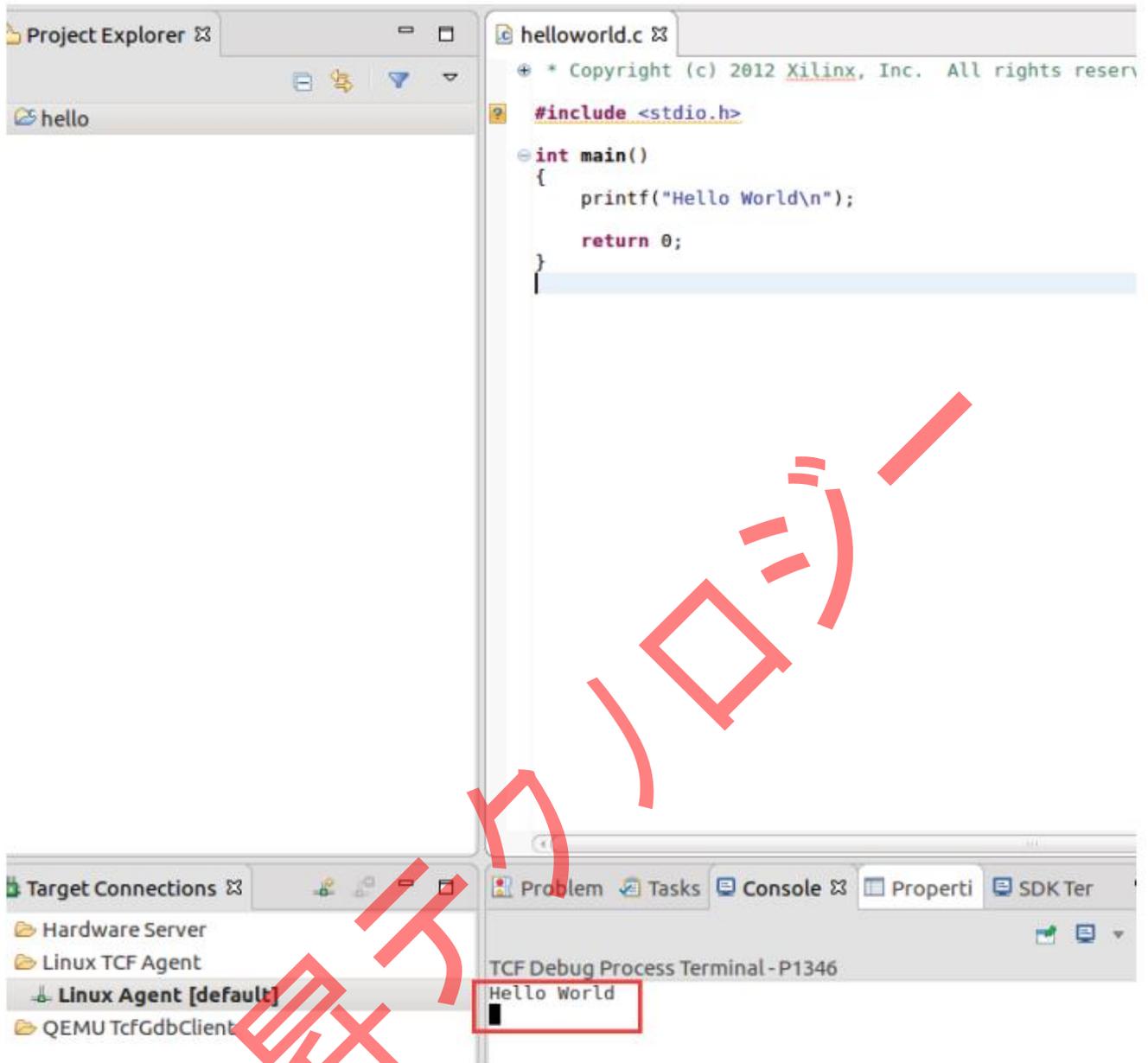
3) Test Connection をクリックして、テスト接続が成功した場合、Linux TCF Agent サービスが実行されており、デバッグを実行できる。



4) プロジェクトを選択して、右クリックして実行する。



- 5) Debug ターミナルで HELLO WORD の出力を確認できる。



#### 19.4 TCF-Agent 問題

TCF-Agent は Linux アプリケーションを簡単にデバッグできるが、NFS サポートも必要ない。しかし、複数行プログラムのデバッグにはあまり適していない。アプリケーションがクラッシュした場合、デバッグ環境の復元には役立たず、開発ボードを再起動する必要があるため、あまり使用されていない。

## 第二十章 Linux 環境で GPIO 実験

前のマニュアルでは、SDK を使用して zynq バージョンの helloWorld 実験をコーディングする方法について説明した。この実験では、zynq のペリフェラルをコントロールする方法を紹介する。実験では、GPIO を例として使用する。ZYNQ の GPIO は、二つのタイプがあって、一つは PS に付属する GPIO、一つは PL で実装された GPIO である。Vivado プロジェクトをビルドする時に、Xilinx の GPIO IP が追加される。Xilinx が提供する IP コアのほとんどはすでに Linux でドライバがあって、しかも、使える。例えば、AXI GPIO ドライバはカーネルで再構成せずに使用できる。

この <http://www.wiki.xilinx.com/Linux+Drivers> サイドに、Linux 下のすべての Xilinx ドライバを見つける。例えば、GPIO ドライバは以下のように、一部ドライバは詳しい使い方もある。

GPIO	Zynq and Zynq Ultrascale+ MPSoC	<a href="#">GPIO Driver</a>	Yes	<a href="#">drivers/gpio/gpio-zynq.c</a>
GPIO	axi_gpio	<a href="#">AXI GPIO Driver</a>	Yes	<a href="#">drivers/gpio/gpio-xilinx.c</a>
HDMI Clks	SI5324 Clock Multiplier/Jitter Attenuator	<a href="#">CCF SI5324 Driver</a>	No	<a href="#">hdmi-modules/clk/*</a>

GPIO ドライバの詳細ページ <http://www.wiki.xilinx.com/Linux%20GPIO%20Driver> に GPIO ドライバの使用範囲、デバイスツリーの例、及びプログラムの作成方法が紹介されている。

### 20.1 SHELL コントロールを使用する

Linux は強い SHELL 機能を提供しており、Linux を学習に必要な技能である。ZYNQ を勉強するには、しっかりと SHELL をマスターしないとイケない。このマニュアルでは、Linux と SHELL の使用方法を紹介しない。

ls /sys/class/gpio コマンドで GPIO 番号を確認できる。

```
root@zynq: ~# ls /sys/class/gpio
export  gpio900  gpio903  gpio906  gpio957  gpiochip898
gpio898  gpio901  gpio904  gpio919  gpiochip1016  unexport
gpio899  gpio902  gpio905  gpio956  gpiochip1020
```

gpio\_test.sh ファイル内容は以下通り、gpio\_test 関数は、パラメーターに従って GPIO をエクスポートし、次に for ループを 3 回行う。最初に 0 を書き込み、次に 1 を書き込み、gpio\_test を 5 回呼び出してから、5 つの LED を点灯する。その中で、898 は PS 側で、その他は PL 側である。(次のセクションでは、GPIO 番号を確認する方法について説明する)

```
#!/bin/sh
gpio_test() {
  gpio=$1
  echo $gpio > /sys/class/gpio/export
  echo out > /sys/class/gpio/gpio${gpio}/direction
  for i in $(seq 1 3)
  do
    echo 0 > /sys/class/gpio/gpio${gpio}/value
```

```

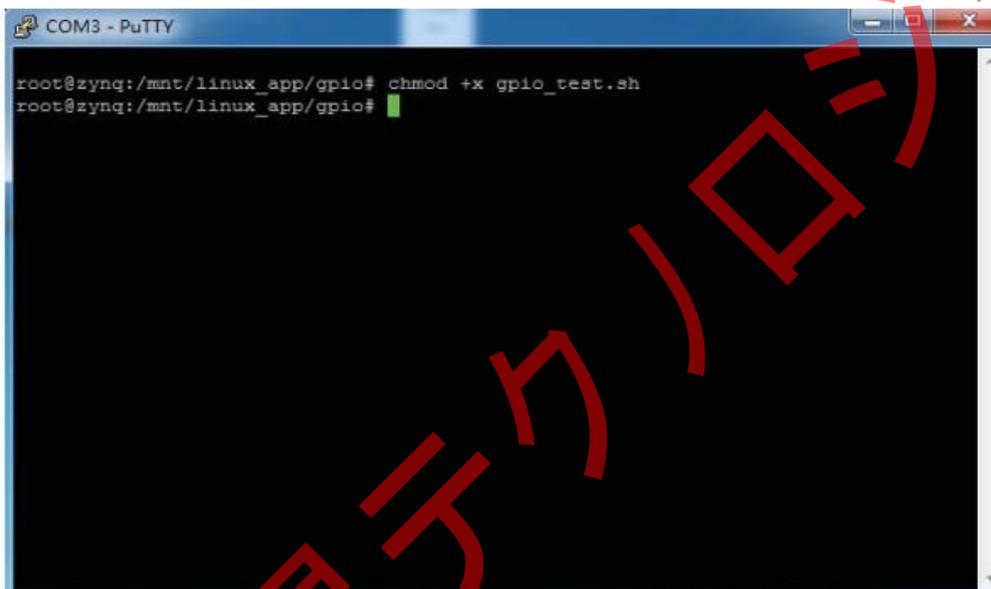
sleep 1
echo 1 >/sys/class/gpio/gpio${gpio}/value
sleep 1
done
echo $gpio > /sys/class/gpio/unexport
}
gpio_test 898
gpio_test 1016
gpio_test 1017
gpio_test 1018
gpio_test 1019

```

NFS をマウントすることでこの SHELL を実行できる

SHELL が実行できない場合、先に実行許可を追加しておく。コマンドは次のとおり。

```
chmod +x gpio_test.sh
```



## 20.2 C 言語を使ってコントロールする

ほとんどの場合、C 言語を使用して周辺機器をコントロールする必要がある。Xilinx の Wiki ページ <http://www.wiki.xilinx.com/GPIO%20User%20Space%20App> には、次の GPIO テストコードが見つかり、コード内容は以下通り。

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

// The specific GPIO being used must be setup and replaced thru
// this code. The GPIO of 898 is in the path of most the sys dirs
// and in the export write.
//
// Figuring out the exact GPIO was not totally obvious when there
// were multiple GPIOs in the system. One way to do is to go into
// the gpiochips in /sys/class/gpio and view the label as it should
// reflect the address of the GPIO in the system. The name of the
// the chip appears to be the 1st GPIO of the controller.
//
// The export causes the gpio898 dir to appear in /sys/class/gpio.
// Then the direction and value can be changed by writing to them.
//
// The performance of this is pretty good, using a nfs mount,

```

```
// running on open source linux,
// the GPIO can be toggled about every 1sec.
// The following commands from the console setup the GPIO to be
// exported, set the direction of it to an output and write a 1
// to the GPIO.
//
// bash> echo 898 > /sys/class/gpio/export
// bash> echo out > /sys/class/gpio/gpio898/direction
// bash> echo 1 > /sys/class/gpio/gpio898/value

// if sysfs is not mounted on your system, the you need to mount it
// bash> mount -t sysfs sysfs /sys

// the following bash script to toggle the gpio is also handy for
// testing
//
// while [ 1 ]; do
//   echo 1 > /sys/class/gpio/gpio898/value
//   echo 0 > /sys/class/gpio/gpio898/value
// done

// to compile this, use the following command
// gcc gpio.c -o gpio

// The kernel needs the following configuration to make this work.
//
// CONFIG_GPIO_SYSFS=y
// CONFIG_SYSFS=y
// CONFIG_EXPERIMENTAL=y
// CONFIG_GPIO_XILINX=y

int main()
{
    int valuefd, exportfd, directionfd;

    printf("GPIO test running...\n");

    // The GPIO has to be exported to be able to see it
    // in sysfs

    exportfd = open("/sys/class/gpio/export", O_WRONLY);
    if (exportfd < 0)
    {
        printf("Cannot open GPIO to export it\n");
        exit(1);
    }

    write(exportfd, "898", 4);
    close(exportfd);

    printf("GPIO exported successfully\n");

    // Update the direction of the GPIO to be an output

    directionfd = open("/sys/class/gpio/gpio898/direction", O_RDWR);
    if (directionfd < 0)
    {
        printf("Cannot open GPIO direction it\n");
        exit(1);
    }

    write(directionfd, "out", 4);
    close(directionfd);

    printf("GPIO direction set as output successfully\n");

    // Update the direction of the GPIO to be an output

    directionfd = open("/sys/class/gpio/gpio898/direction", O_RDWR);
    if (directionfd < 0)
    {
        printf("Cannot open GPIO direction it\n");
        exit(1);
    }

    write(directionfd, "out", 4);
    close(directionfd);

    printf("GPIO direction set as output successfully\n");

    // Get the GPIO value ready to be toggled

    valuefd = open("/sys/class/gpio/gpio898/value", O_RDWR);
    if (valuefd < 0)
    {
        printf("Cannot open GPIO value\n");
        exit(1);
    }

    printf("GPIO value opened, now toggling...\n");

    // toggle the GPIO as fast a possible forever, a control c is needed
    // to stop it

    while (1)
    {
        write(valuefd, "1", 2);
        sleep(1);
        write(valuefd, "0", 2);
        sleep(1);
    }
}
```

今回はコンパイルに SDK を使用しない。ソースコードの名前を `gpio.c` にし、次のコマンドを実行してコードをコンパイルする。

```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
arm-linux-gnueabi-gcc gpio.c -o gpio
```

コンパイルが完了すると、`gpio` ファイルが生成される。Windows や Linux とは異なり、拡張名は厳しくない。`gpio` ファイルは `elf` ファイルである。

`gpio` を実行すると、ps 端子 LED が点滅していることがわかる。これは、この 898 が PS 側の最初の LED であることを示している。

```
root@zynq:/mnt/linux_app/gpio# ./gpio
GPIO test running...
GPIO exported successfully
GPIO direction set as output successfully
GPIO value opened, now toggling...
█
```

### 20.2.1 GPIO のコードの確認

次のコマンドを実行すると、`gpiochip898` `gpiochip1016` `gpiochip1020` が表示される。これは、3つの GPIO コントローラーがあることを示し、数字はコントローラーGPIOの基数である。

```
ls /sys/class/gpio
```

```
root@zynq:/mnt/linux_app/gpio# ls /sys/class/gpio
export gpio898 gpiochip1016 gpiochip1020 gpiochip898 unexport
root@zynq:/mnt/linux_app/gpio# █
```

### 20.2.2 物理 GPIO との関係の確認

次のコマンドで、GPIO1016 と物理 GPIO の関係を確認する。デバイスツリーの `gpio` のノードは `/amba_pl/gpio @ 41210000` であることがわかる。デバイスツリーのノードでどの物理 GPIO を確認できる。

```
cat /sys/class/gpio/gpiochip1016/label
```

```
root@zynq:/sys/class/gpio/gpiochip1016# cat /sys/class/gpio/gpiochip1016/label
/amba_pl/gpio@41210000
root@zynq:/sys/class/gpio/gpiochip1016# █
```

### 20.3 実験のまとめ

この実験の焦点は、Xilinx から提供された情報を通して ZYNQ を学習すること。技術資料は迅速に更新され、チップメーカーから提供された最新の情報を入手して最新最高の技術を取得できる。 フォローアップマニュアルでは、PCIe ドライバと PL 側イーサネットドライバが Xilinx から提供されている、これらの資料は wiki で入手できる。

非 xilinx IP または独自の IP を使用する場合、独自のドライバを開発する必要がある。これは、Linux ドライバを実行していない開発者にとっての課題であるため、Xilinx の IP を使用してシステムを構築したほうがお勧めである。利点は、Linux ドライバを開発する必要がないこと、欠点は、十分な柔軟性がないこと。もし、IP に問題がある、または、ドライバに問題がある場合、問題をすばやく見つけることができない。

日昇テクノロジー

## 第二十一章 Petalinux での HDMI ディスプレイ

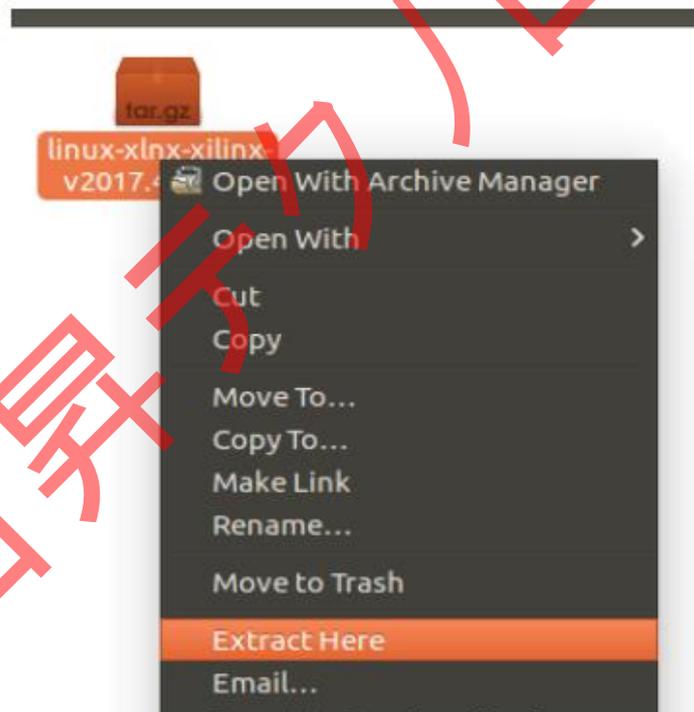
前のマニュアルで、Petalinux で組み込み Linux システムを開発するのを体験したが、使う機能は Petalinux のただ一部である。この実験では、独自のカーネルを使用して Linux を実行する方法を紹介する。これにより、HDMI ディスプレイなど、多くのドライバをカーネルに追加できる。

開発ボードは HDMI インターフェイスチップを使用せず、PFGA を使用してエンコードを完了し、Core Electronics は Xilinx が提供するカーネルに HDMI コード化 IP ドライバを追加した。その他のバージョンを使用するソフトウェア開発者に、このマニュアルでは変更された linux-xlnx-xilinx-v2017.4 カーネルのみを提供し、他のバージョンの変更されたバージョン、および変更手順が提供されていないことを注意してください。

### 21.1 Petalinux のコンフィグ

この実験は、前の Petalinux プロジェクトの実験で変更する。前の実験コンテンツを習得する必要がある。

- 1) カーネルソースファイルを Linux ホストにコピーして、次に解凍する。解凍後のカーネルディレクトリはこの実験 Petalinux に使うカーネルである。



- 2) ターミナルを開き、前の実験中の Petalinux プロジェクトディレクトリに入る。

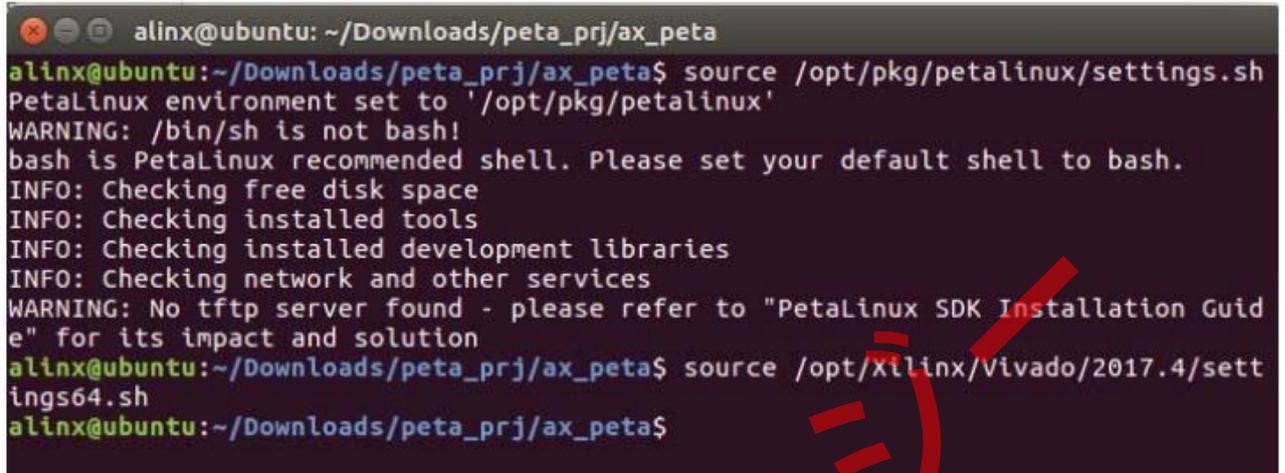
```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
```

- 3) Petalinux 環境変数を設定して、下のコマンドを実行する。

```
source /opt/pkg/petalinux/settings.sh
```

- 4) 下のコマンドを実行して、vivado 環境変数を設定する。

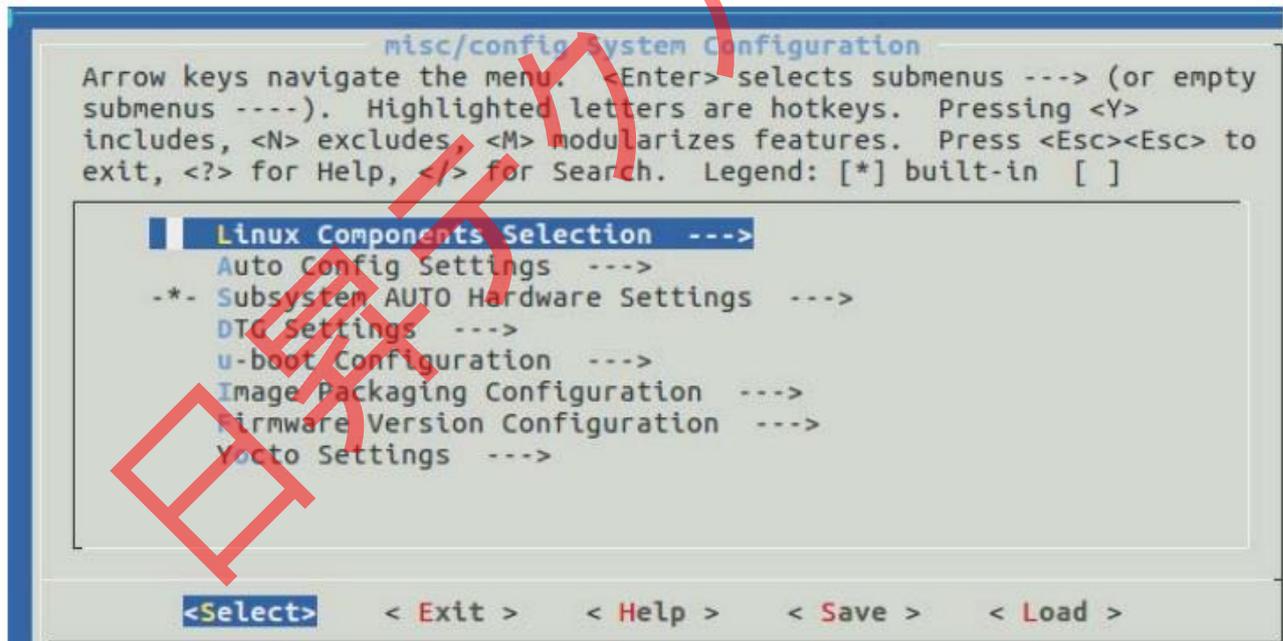
```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
```



```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and solution
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
```

- 5) 下のコマンドで PetaLinux をもう一回コンフィグする。

```
petalinux-config
```

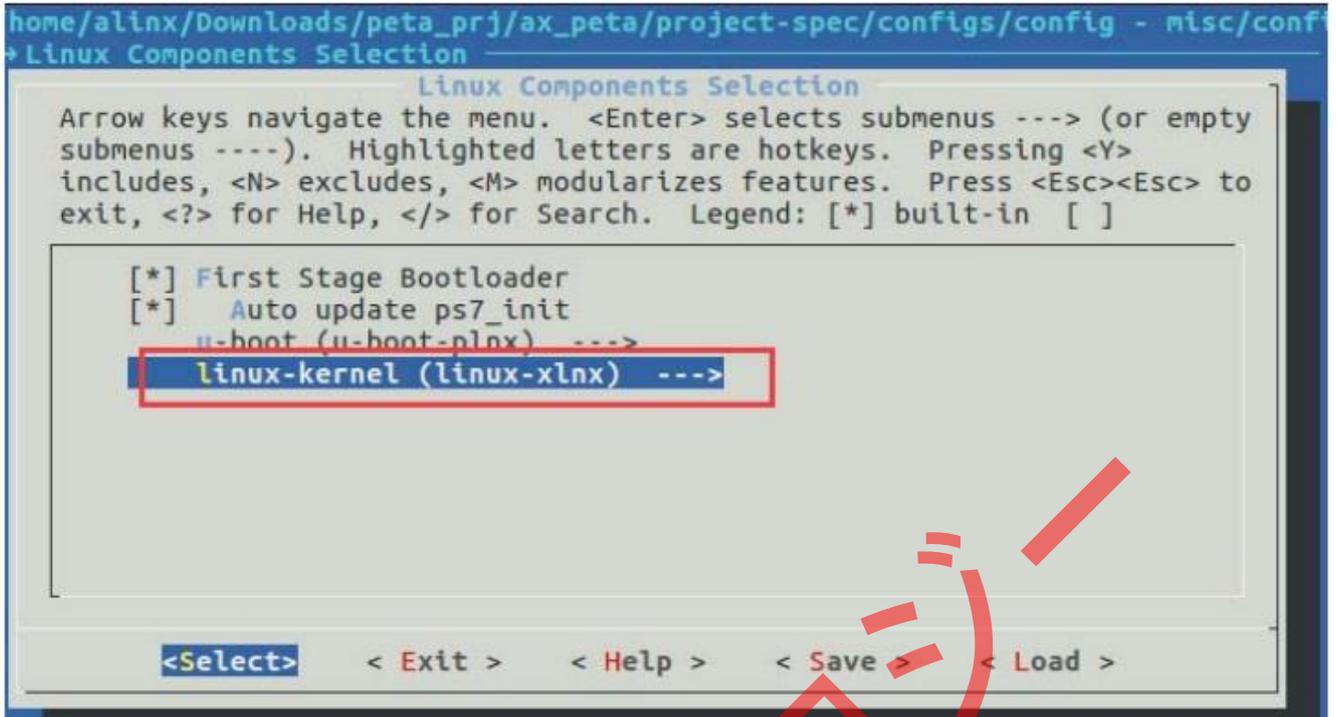


```
misc/config system Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

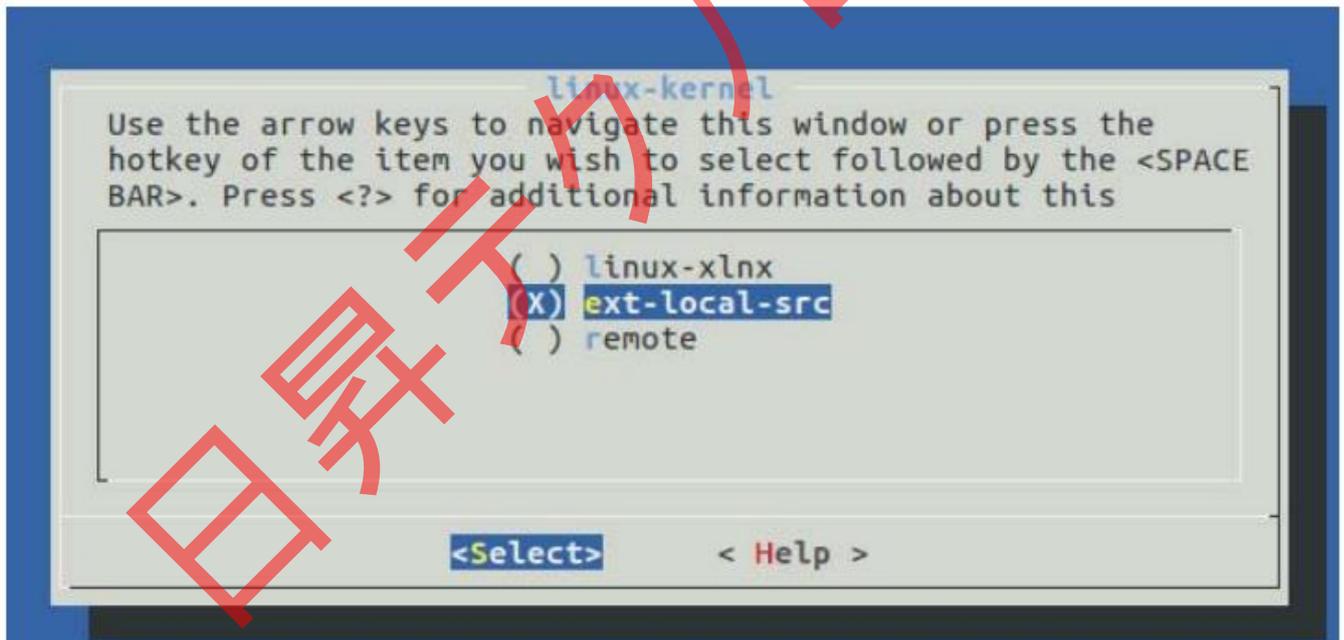
Linux Components Selection --->
  Auto Config Settings --->
  -* Subsystem AUTO Hardware Settings --->
  DTG Settings --->
  u-boot Configuration --->
  Image Packaging Configuration --->
  Firmware Version Configuration --->
  Yocto Settings --->

<Select> < Exit > < Help > < Save > < Load >
```

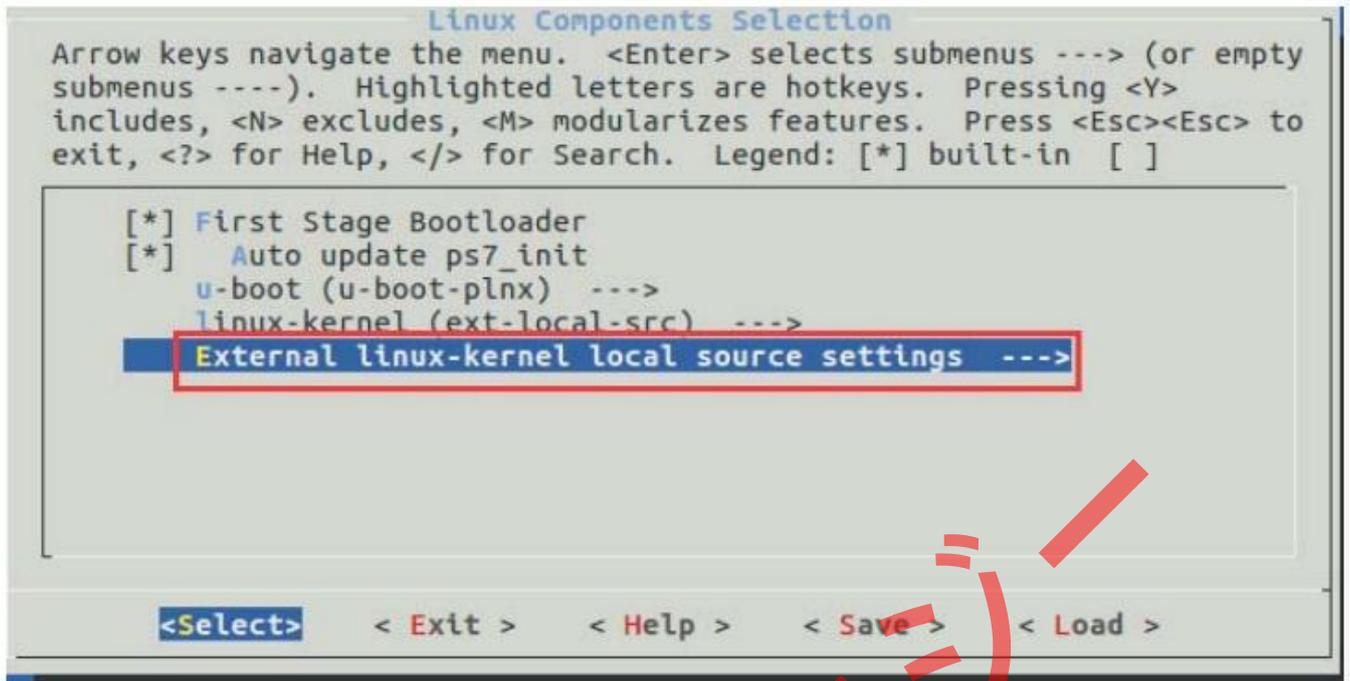
- 6) Linux Components Selection ---> linux-kernel (linux-xlnx) --->を選択する。



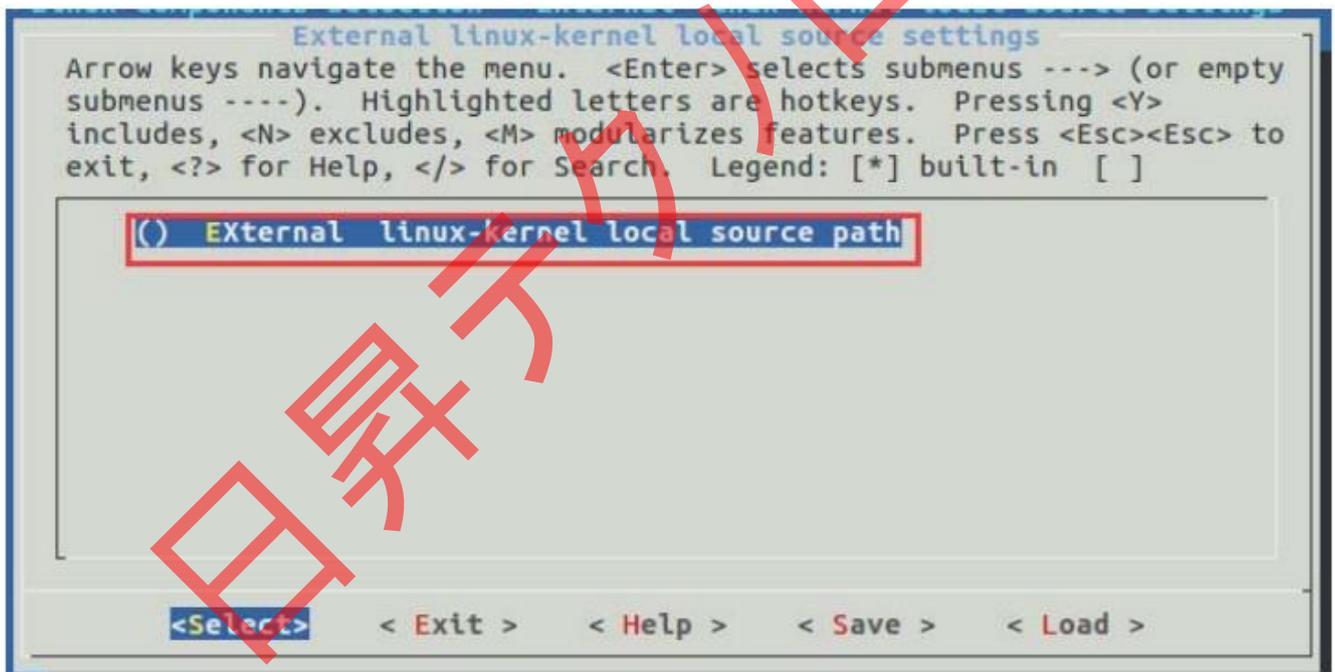
7) ext-local-src を選択してスペースキーを押す



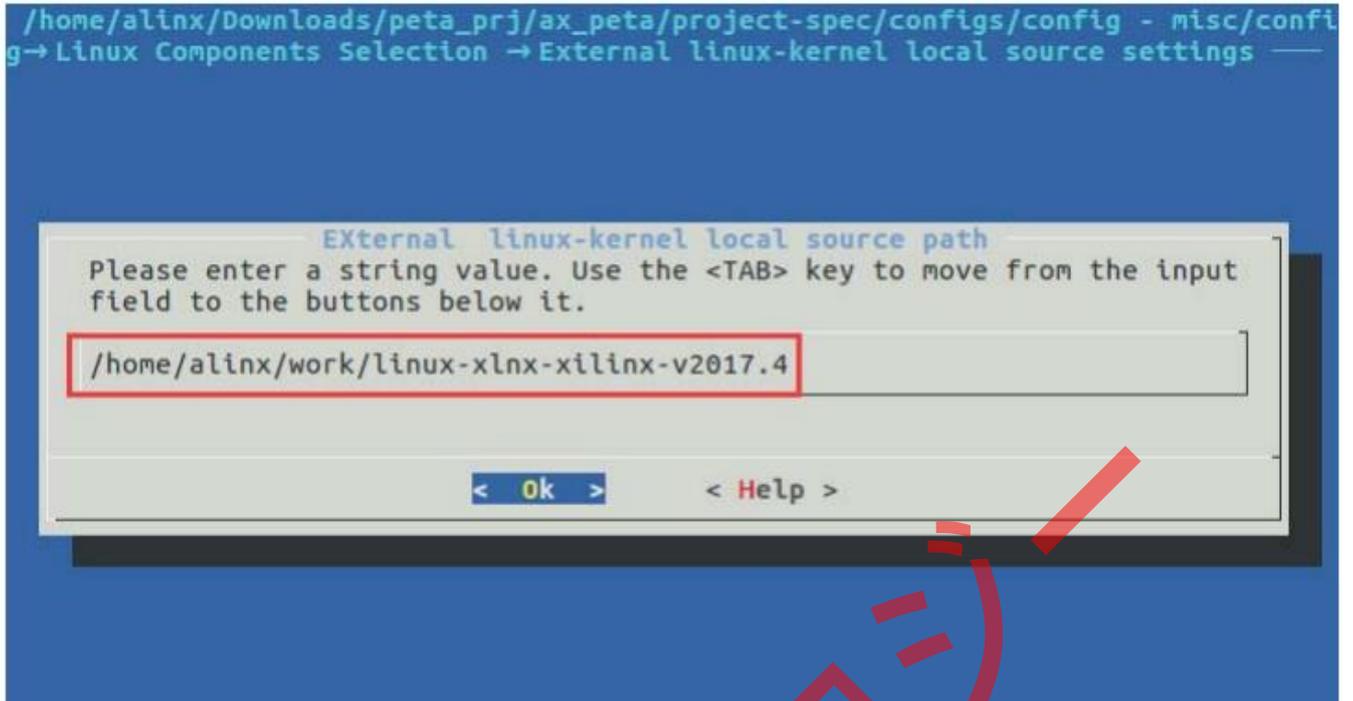
8) External linux-kernel local source settings --->を選択する。



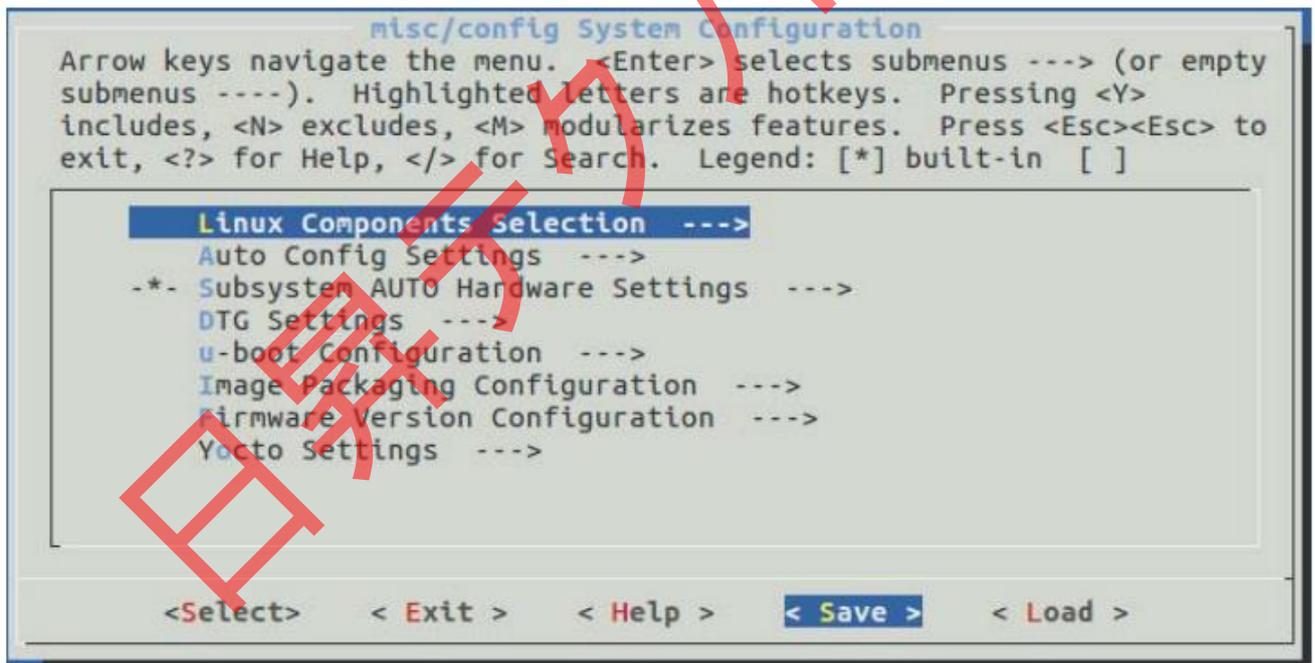
External linux-kernel local source path を選択する。



9) Linux カーネルソースコードのパスを/home/alinx/work/linux-xlnx-xilinx-v2017.4 入力する。実際のパスはカーネルの場所によって異なる。ここに例を示す。



10) 保存して閉じる。



## 21.2 Linux カーネルをコンフィグする

1) 下のコマンドを実行してカーネルをコンフィグする。

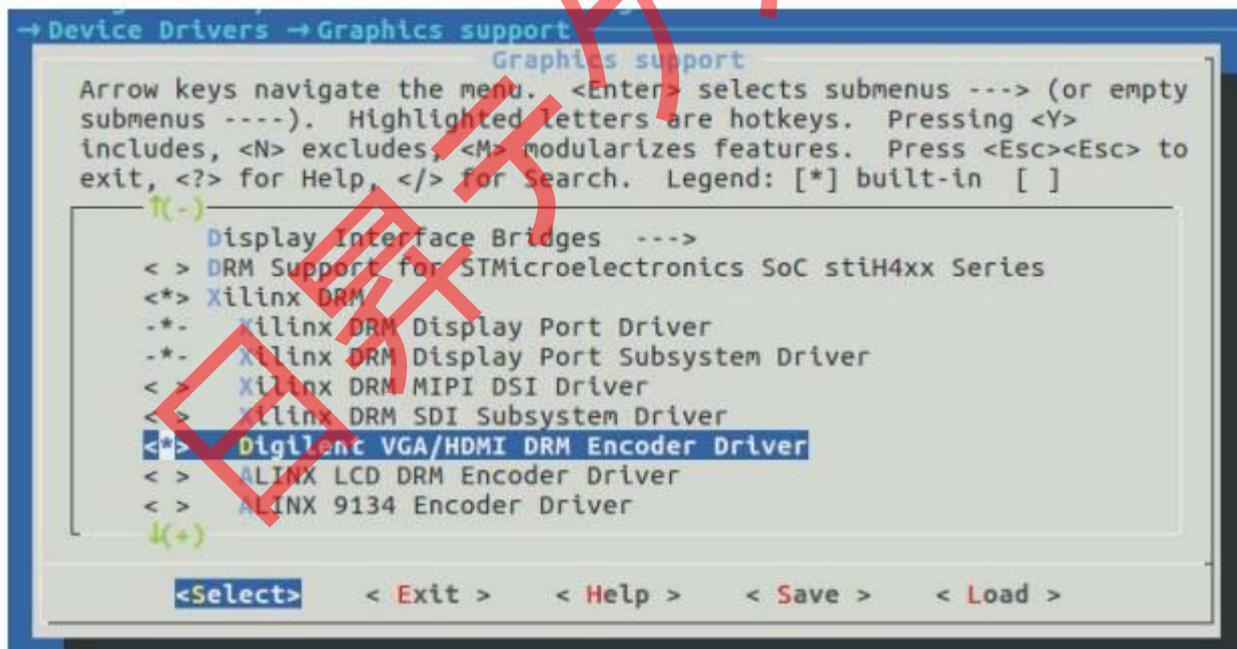
```
petalinux-config -c kernel
```

```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
*** Execute 'make' to start the build or try 'make help'.

[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_peta
~/Downloads/peta_prj/ax_peta
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_peta
~/Downloads/peta_prj/ax_peta
[INFO] generating u-boot configuration files

[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
[INFO] successfully configured project
webtalk failed:PetaLinux statistics:extra lines detected:notsent_nofile!
webtalk failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-config -c kernel
```

2) ポップアップウィンドウが表示され、Device Drivers→Graphics support に入り、Digilent VGA/HDMI DRM Encoder Driver を選択して y を s 押す。



3) Device Drivers の Common Clock Framework オプションから Digilent axi\_dyncclk Driver を選択して y を押す。

```
.config - Linux/arm 4.9.0 Kernel Configuration
-> Device Drivers -> Common Clock Framework
Common Clock Framework
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
π(-)
[*] Clock driver for Versatile Express OSC clock generators
< > Clock driver for SiLabs 5351A/B/C
< > Clock driver for SiLabs 514 devices
<*> Clock driver for SiLabs 570 and compatible devices
< > Clock driver for TI CDCE706 clock synthesizer
< > Clock driver for TI CDCE925 devices
< > Clock driver for CS2000 Fractional-N Clock Synthesizer & Cloc
< > AXI clkgen driver
<*> Digilent axi_dynclk Driver
[ ] Clock driver for Freescale QorIQ platforms

<Select> < Exit > < Help > < Save > < Load >
```

4) 保存して閉じる。

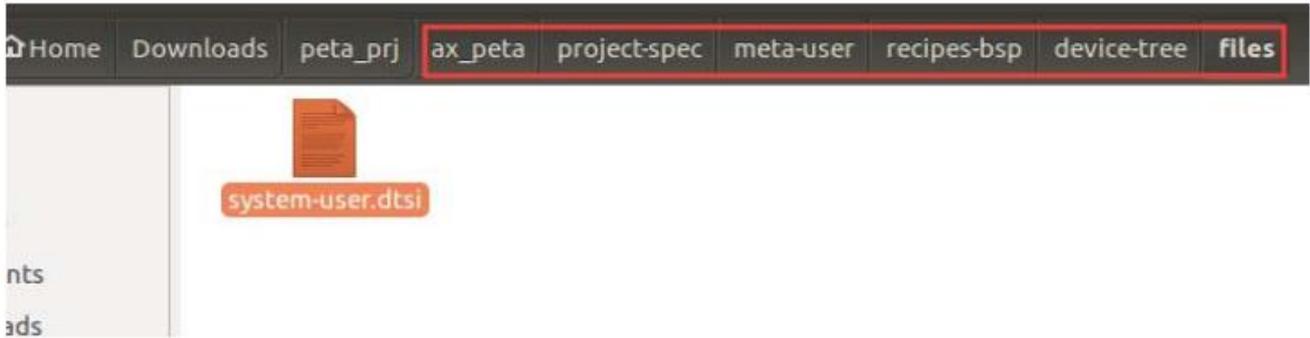
```
-> Device Drivers -> Common Clock Framework
Common Clock Framework
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
π(-)
[*] Clock driver for Versatile Express OSC clock generators
< > Clock driver for SiLabs 5351A/B/C
< > Clock driver for SiLabs 514 devices
<*> Clock driver for SiLabs 570 and compatible devices
< > Clock driver for TI CDCE706 clock synthesizer
< > Clock driver for TI CDCE925 devices
< > Clock driver for CS2000 Fractional-N Clock Synthesizer & Cloc
< > AXI clkgen driver
<*> Digilent axi_dynclk Driver
[ ] Clock driver for Freescale QorIQ platforms

<Select> < Exit > < Help > < Save > < Load >
```

### 21.3 デバイスツリーを変更する

デバイスツリーは、デバイス情報を記述するフォーマットされたテキストの一種で、このテキスト構造はXML やJSON に似ている。

1) petalinux プロジェクトファイルで system-user.dtsi という名前のファイルを開く。



2) デバイスツリーの変更内容は以下通りである。

```
/include/ "system-conf.dtsi"

/{
    model = "Zynq ALINX Development Board";
    compatible = "alinx,zynq", "xlnx,zynq-7000";
    aliases {
        ethernet0 = "&gem0";
        serial0 = "&uart1";
    };

    usb_phy0: usb_phy@0 {
        compatible = "ulpi-phy";
        #phy-cells = <0>;
        reg = <0xe0002000 0x1000>;
        view-port = <0x0170>;
        drv-vbus;
    };

};

&i2c0 {
    clock-frequency = <100000>;
};

&usb0 {
    dr_mode = "host";
    usb-phy = <&usb_phy0>;
};

&sdhci0 {
    u-boot,dm-pre-reloc;
};

&uart1 {
    u-boot,dm-pre-reloc;
};

&flash0 {
    compatible = "micron,m25p80", "w25q256", "spi-flash";
};
```

```

&gem0 {
    phy-handle = <&ethernet_phy>;
    ethernet_phy: ethernet-phy@1 {
        reg = <1>;
        device_type = "ethernet-phy";
    };
};

&amba_pl {

    hdmi_encoder_0:hdmi_encoder {
        compatible = "digilent,drm-encoder";
        digilent,edid-i2c = <&i2c0>;
    };

    xilinx_drm {
        compatible = "xlnx,drm";
        xlnx,vtc = <&v_tc_0>;
        xlnx,connector-type = "HDMI-A";
        xlnx,encoder-slave = <&hdmi_encoder_0>;
        clocks = <&axi_dynclk_0>;
        digilent,edid-i2c = <&i2c0>;
        planes {
            xlnx,pixel-format = "rgb888";
            plane0 {
                dmas = <&axi_vdma_0 0>;
                dma-names = "dma";
            };
        };
    };
};

&axi_dynclk_0 {
    compatible = "digilent,axi-dynclk";
    #clock-cells = <0>;
    clocks = <&clkc 15>;
};

&v_tc_0 {
    compatible = "xlnx,v-tc-5.01.a";
};

```

## 21.4 テスト petalinux プロジェクトのコンパイル

1) 下のコマンドで、uboot、カーネル、ルートファイルシステム、デバイスツリーなどをコンパイルする。

```
petalinux-build
```

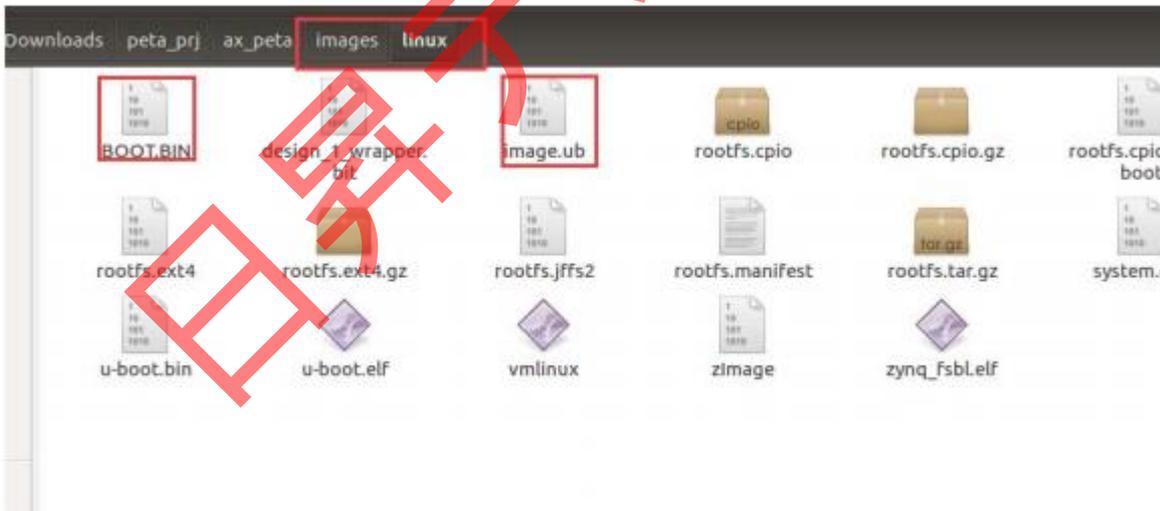
```

alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
Loading cache: 100% |#####| Time: 0:00:00
Loaded 3256 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2433 cached, 33 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####| Time: 0:00:04
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2 tasks of which 0 didn't need to be rerun and al
l succeeded.
[INFO] successfully configured kernel
weblink failed:PetaLinux statistics:extra lines detected:notsent_nofile!
weblink failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |#####| Time: 0:00:01
Loaded 3256 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2433 cached, 33 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 18% |#####| ETA: 0:00:08
  
```

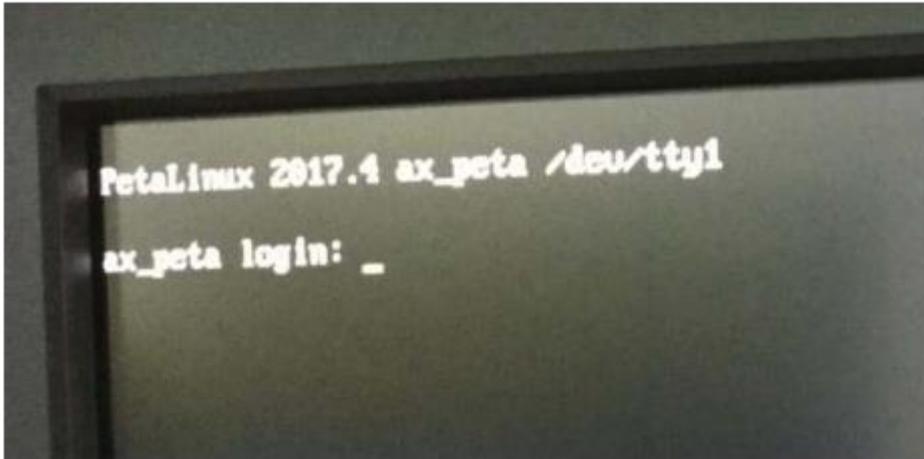
2) 下のコマンドを実行して BOOT ファイルを作成する。スペースとハイフンに注意してください。

```
petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga --u-boot --force
```

3) BOOT.bin と image.ub を SD カードにコピーして、開発ボードを SD モードで起動するように設定し、HDMI ディスプレイを接続して、開発ボードを起動する。



4) ディスプレイに以下の内容が表示される。



## 21.5 よくある問題

### 21.5.1 システムのスリープを防ぐ方法

スリープ前にコマンドを実行する。

```
echo -e "\033[9;0]\033[?33\033[?25h\033[?1c" > /dev/tty0  
echo -e "\033[9;0]\033[?33\033[?25h\033[?1c" > /dev/tty1  
echo -e "\033[9;0]\033[?33\033[?25h\033[?1c" > /dev/tty  
echo -e "\033[9;0]\033[?33\033[?25h\033[?1c" > /dev/console
```

## 第二十二章 Debian デスクトップシステムの使用

前のマニュアルで Petalinux で組み込む Linux システム及び HDMI ディスプレイを作成する方法を紹介した。この実験では、Debian に基づいてルートファイルを作成する。ルートファイルシステムの作成が複雑のため、この章では説明しない。生成された Debian ルートファイルシステムを直接使用する。

### 22.1 Petalinux のコンフィグ

Debian ルートファイルシステムが大きいいため、QSPI flash には保存できず、SD カードまたは emmc にしか保存できない、だから、Petalinux をコンフィグする必要がある。

この実験は、前の Petalinux プロジェクトの実験をまだ変更するため。以前のテストコンテンツをマスターする必要がある。

- 1) ターミナルを開き、前の実験の Petalinux プロジェクトディレクトリに入る

```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
```

- 2) Petalinux の環境変数を設定し、下のコマンドを実行する。

```
source /opt/pkg/petalinux/settings.sh
```

- 3) 下のコマンドを実行して vivado 環境変数を設定する

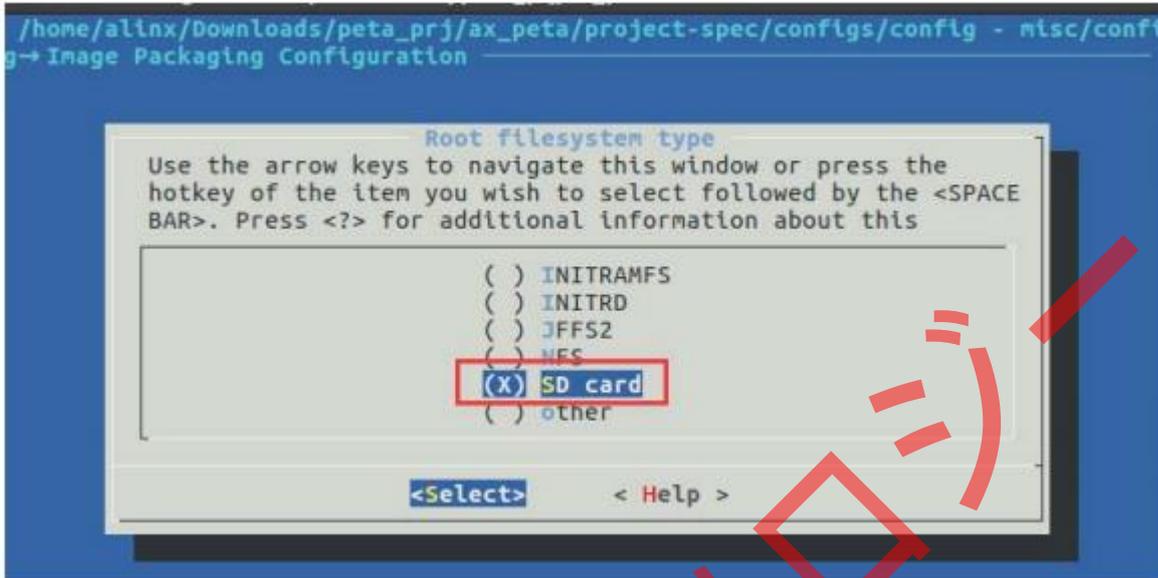
```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

```
alinx@ubuntu:~/Downloads/peta_prj/ax_peta
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to '/opt/pkg/petalinux'
WARNING: /bin/sh is not bash!
bash is Petalinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and solution
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$
```

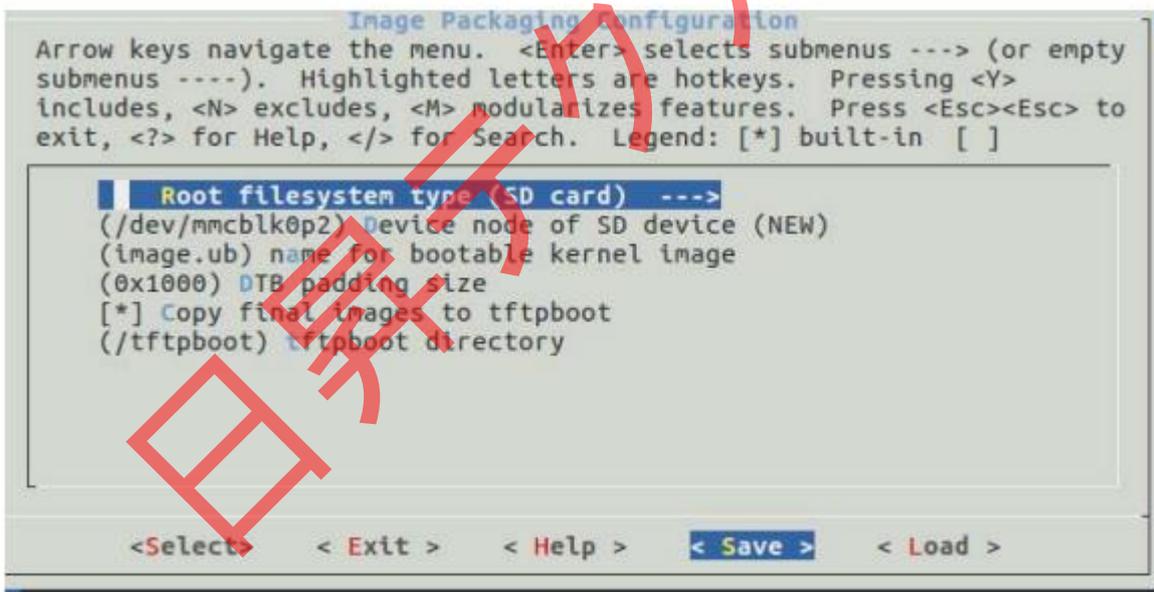
- 4) 下のコマンドで Petalinux をもう一回コンフィグする。

petalinux-config

- 5) Image Packaging Configuration --> Root filesystem type オプションから SD card を選択し、ルートファイルシステムを SD カードに置く。



- 6) 保存して閉じる。



## 22.2 linux カーネルをコンフィグする

- 1) 下のコマンドを実行してカーネルをコンフィグする

petalinux-config -c kernel

```

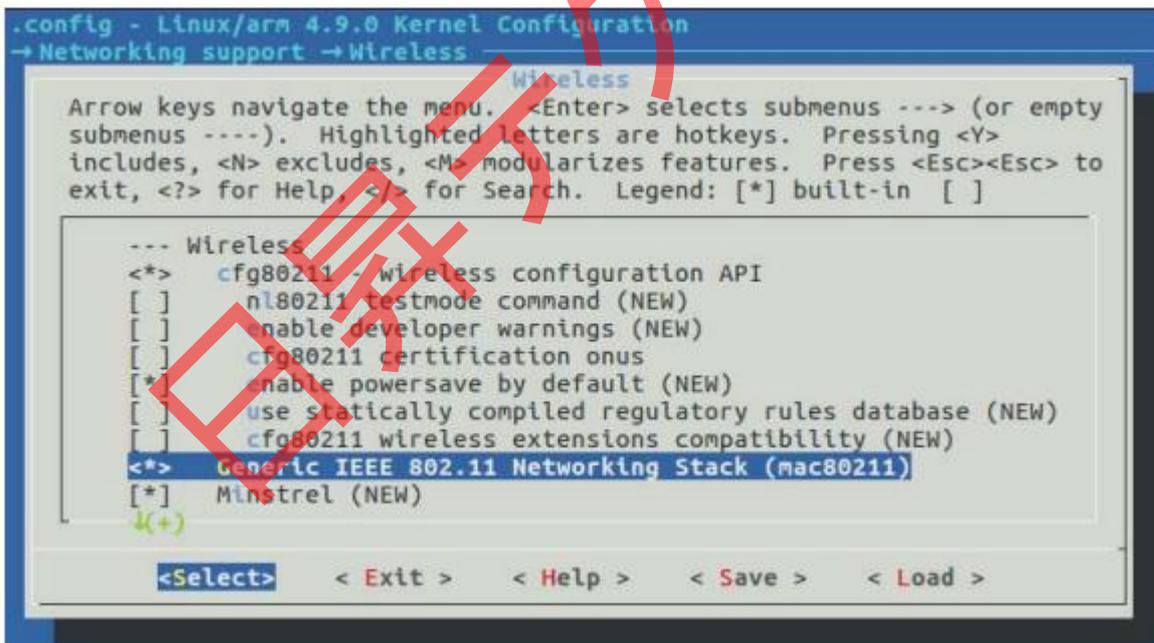
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
*** Execute 'make' to start the build or try 'make help'.

[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_peta
~/Downloads/peta_prj/ax_peta
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
~/Downloads/peta_prj/ax_peta/build/misc/plnx-generated ~/Downloads/peta_prj/ax_peta
~/Downloads/peta_prj/ax_peta
[INFO] generating u-boot configuration files

[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
[INFO] successfully configured project
webtalk failed:Petalinux statistics:extra lines detected:notsent_nofile!
webtalk failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-config -c kernel
  
```

### 22.2.1 USB WIFI モジュールドライバをコンフィグする

1) Networking Support → Wireless → オプションの中から cfg80211 - wireless configuration API を選択して、次に Generic IEEE 802.11 Networking Stack (mac80211) を選択する。



```

.config - Linux/arm 4.9.0 Kernel Configuration
→ Networking support → Wireless
  Wireless
  Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
  --- Wireless
  <*>  cfg80211 - wireless configuration API
  [ ]  nl80211 testmode command (NEW)
  [ ]  enable developer warnings (NEW)
  [ ]  cfg80211 certification onus
  [*]  enable powersave by default (NEW)
  [ ]  use statically compiled regulatory rules database (NEW)
  [ ]  cfg80211 wireless extensions compatibility (NEW)
  <*>  Generic IEEE 802.11 Networking Stack (mac80211)
  [*]  Minstrel (NEW)
  (4+)
  <Select>  < Exit >  < Help >  < Save >  < Load >
  
```

2) Device Drivers → Network device support → Wireless LAN → Realtek rtlwifi family of devices オプションから Realtek RTL8192CU/RTL8188CU USB Wireless Network Adapter を選択する。

```
.config - Linux/arm 4.9.0 Kernel Configuration
[...] twok device support → Wireless LAN → Realtek rtlwifi family of devices
      Realtek rtlwifi family of devices
      Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
      submenu ---). Highlighted letters are hotkeys. Pressing <Y>
      includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
      exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

      --- Realtek rtlwifi family of devices
      < > Realtek RTL8192CE/RTL8188CE Wireless Network Adapter (NEW)
      < > Realtek RTL8192SE/RTL8191SE PCIe Wireless Network Adapter (
      < > Realtek RTL8192DE/RTL8188DE PCIe Wireless Network Adapter (
      < > Realtek RTL8723AE PCIe Wireless Network Adapter (NEW)
      < > Realtek RTL8723BE PCIe Wireless Network Adapter (NEW)
      < > Realtek RTL8188EE Wireless Network Adapter (NEW)
      < > Realtek RTL8192EE Wireless Network Adapter (NEW)
      < > Realtek RTL8821AE/RTL8812AE Wireless Network Adapter (NEW)
      <*> Realtek RTL8192CU/RTL8188CU USB Wireless Network Adapter
      k(+)

      <Select> < Exit > < Help > < Save > < Load >
```

## 22.2.2 USB カメラドライバをコンフィグする

1) Device Drivers ----> Multimedia support ----> Media USB Adapters ---->オプションの USB Video Class (UVC) を有効にする。

```
.config - Linux/arm 4.9.0 Kernel Configuration
→ Device Drivers → Multimedia support → Media USB Adapters
      Media USB Adapters
      Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
      submenu ---). Highlighted letters are hotkeys. Pressing <Y>
      includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
      exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

      --- Media USB Adapters
      *** Webcam devices ***
      <*> USB Video Class (UVC)
      [*] UVC input events device support (NEW)
      <M> GSPCA based webcams (NEW) --->
      < > USB Philips Cameras (NEW)
      < > C-PIA2 Video For Linux (NEW)
      < > USB ZR364XX Camera support (NEW)
      < > USB Syntek DC1125 Camera support (NEW)
      < > USB Sensoray 2255 video capture device (NEW)
      k(+)

      <Select> < Exit > < Help > < Save > < Load >
```

2) 保存して閉じる。

## 22.3 Petalinux プロジェクトのコンパイルとテスト

- 1) 下のコマンドを使用して、uboot、カーネル、ルートファイルシステム、デバイスツリーなどをコンフィグする。

```
petalinux-build
```

```
alinx@ubuntu: ~/Downloads/peta_prj/ax_peta
Loading cache: 100% |#####| Time: 0:00:00
Loaded 3256 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2433 cached, 33 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####| Time: 0:00:04
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2 tasks of which 0 didn't need to be rerun and al
l succeeded.
[INFO] successfully configured kernel
webtalk failed:PetaLinux statistics:extra lines detected:notsent_nofile!
webtalk failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Loading cache: 100% |#####| Time: 0:00:01
Loaded 3256 entries from dependency cache.
Parsing recipes: 100% |#####| Time: 0:00:03
Parsing of 2466 .bb files complete (2433 cached, 33 parsed). 3259 targets, 226 s
kipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 18% |#####| ETA: 0:00:08
```

- 2) 下のコマンドを実行して BOOT.BIN ファイルを生成する。スペースとハイフンを注意してください。

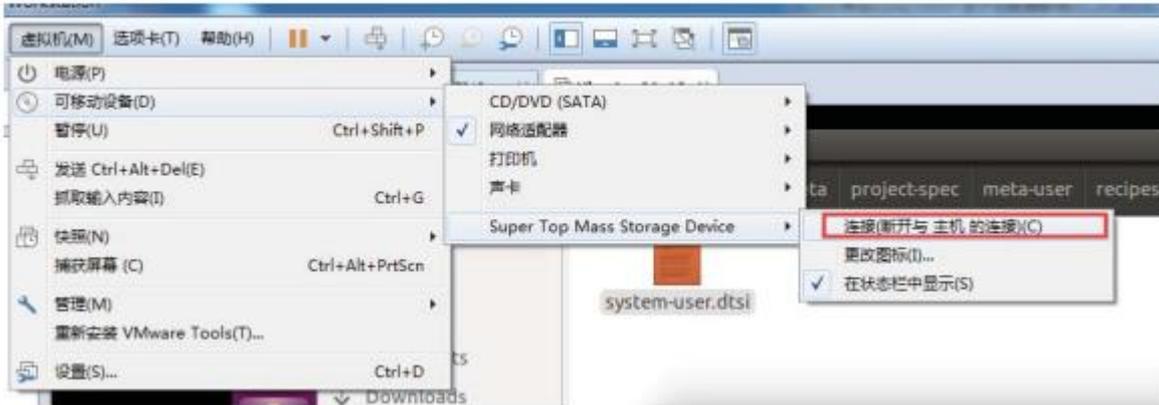
```
petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga --u-boot --force
```

## 22.4 SD カードファイルシステムを作成する

SD カードファイルシステムを作成するには SD カードの内容が失う可能性があるため、事前にバックアップしてください。

### 22.4.1 SD カードのパーティションを変更する。

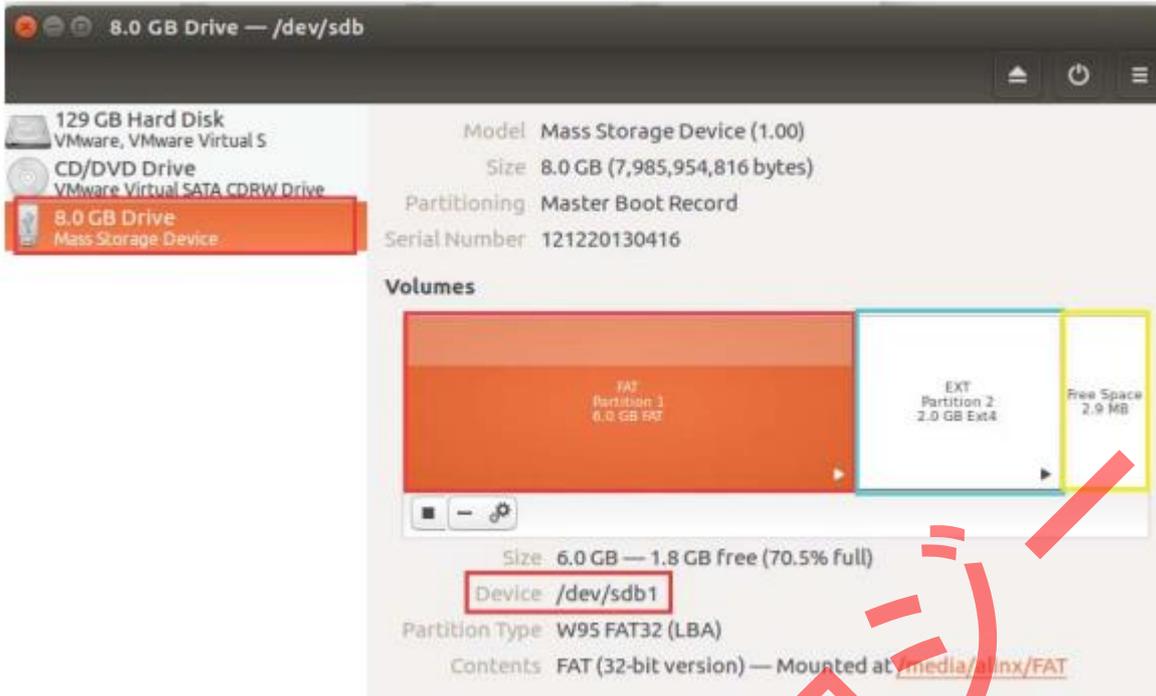
- 1) 開発ボードの SD カードをカードリーダーに挿入し、次にコンピューターの USB ポートに挿入する。
- 2) 仮想マシン Linux に接続する



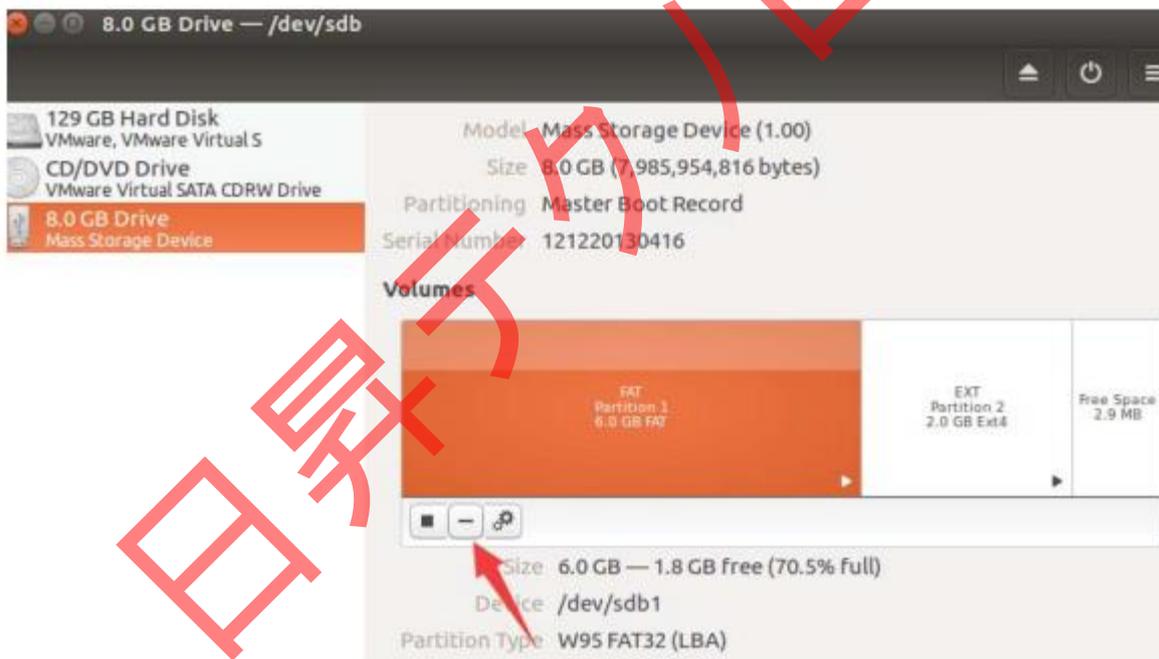
3) ubuntu の検索パスに disk と入力すると、Disks のアイコンが表示される。

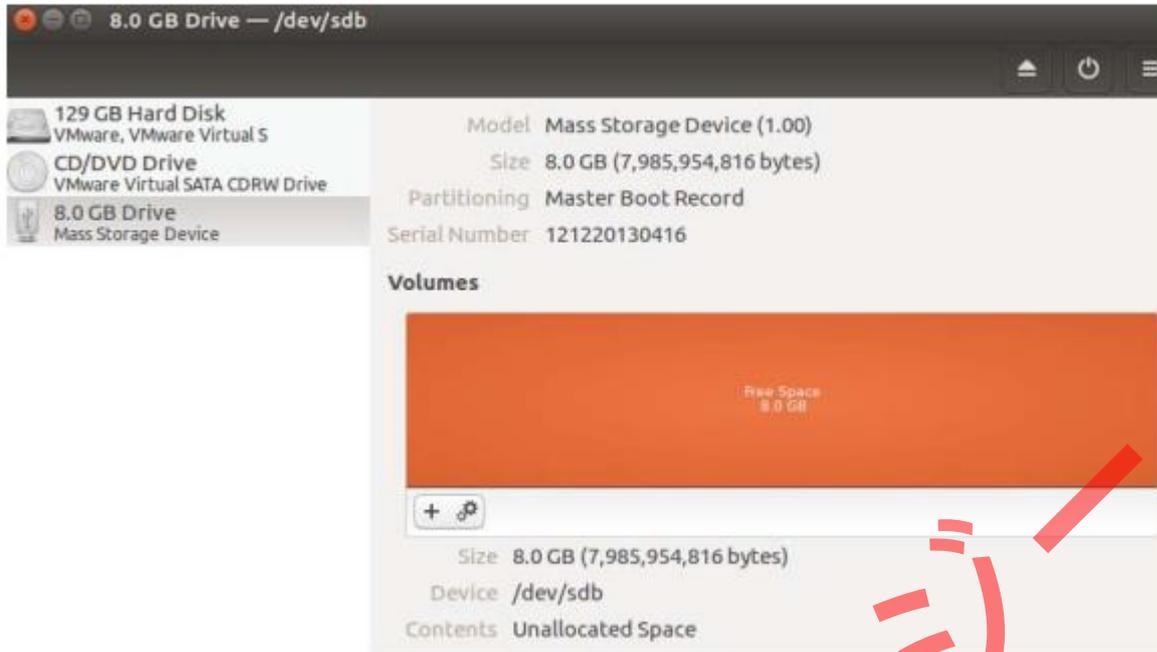


4) Disks アイコンをクリックすると、Disks のダイアログボックスが表示される。この実験で、SD カードは、FAT と EXT という2つのパーティションに分けているが、ここではもう一回分け直す。

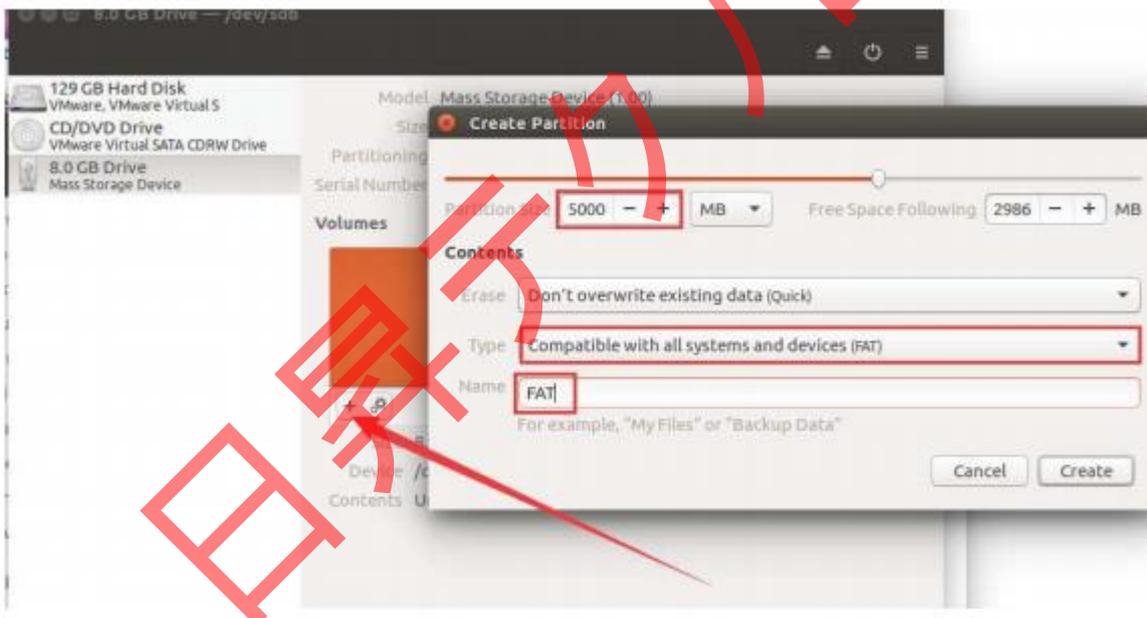


- 5) 各パーティションの下のパーティション削除アイコンを選択し、すべてのパーティションを削除する。

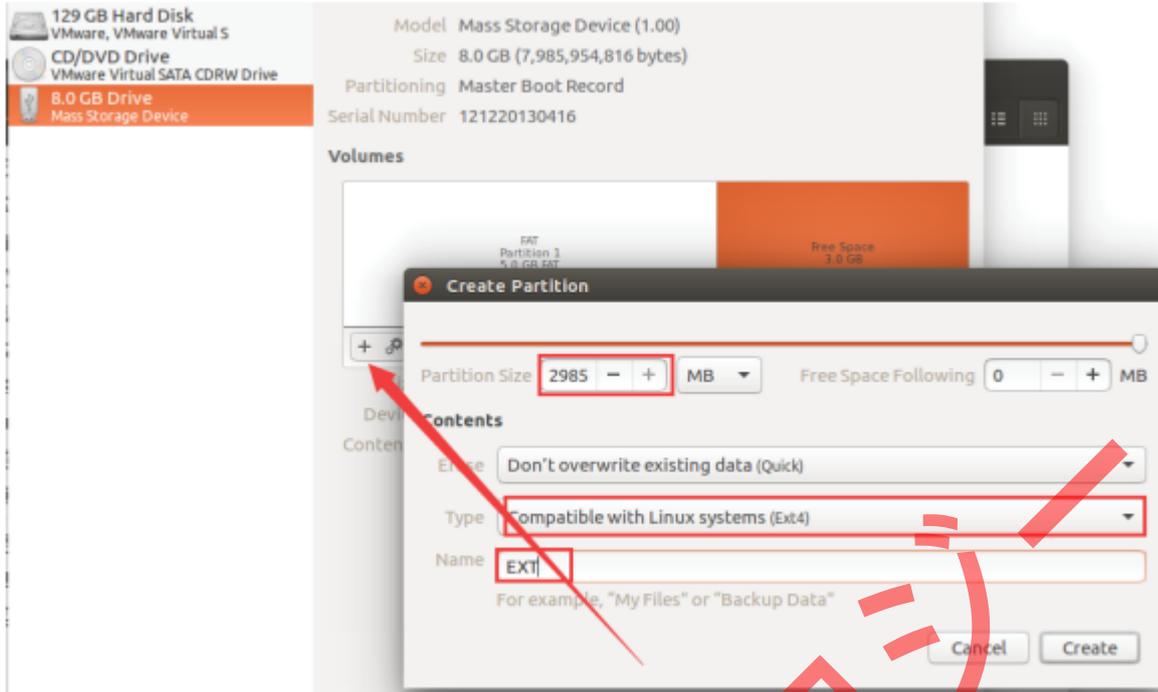




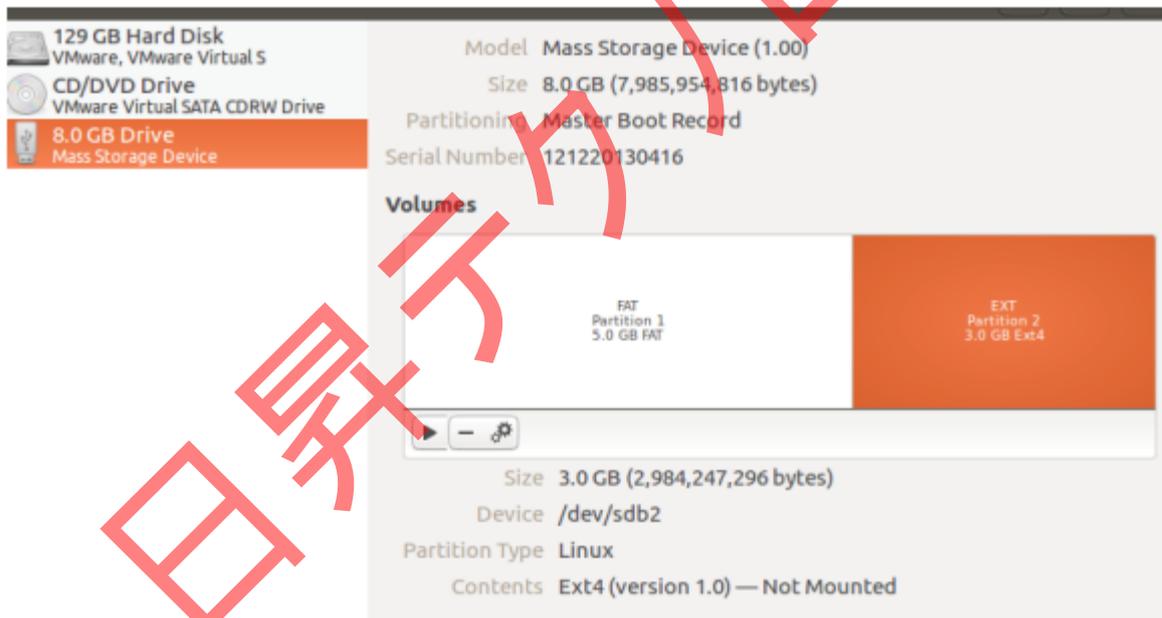
6) +アイコンをクリックして最初のパーティションを追加する。この実験は5000MBを入力し、フォーマットはFATで、ZYNQのブートファイルBOOT.binとカーネルファイル、及びデバイスツリーを保存するため、名前はFATにする。



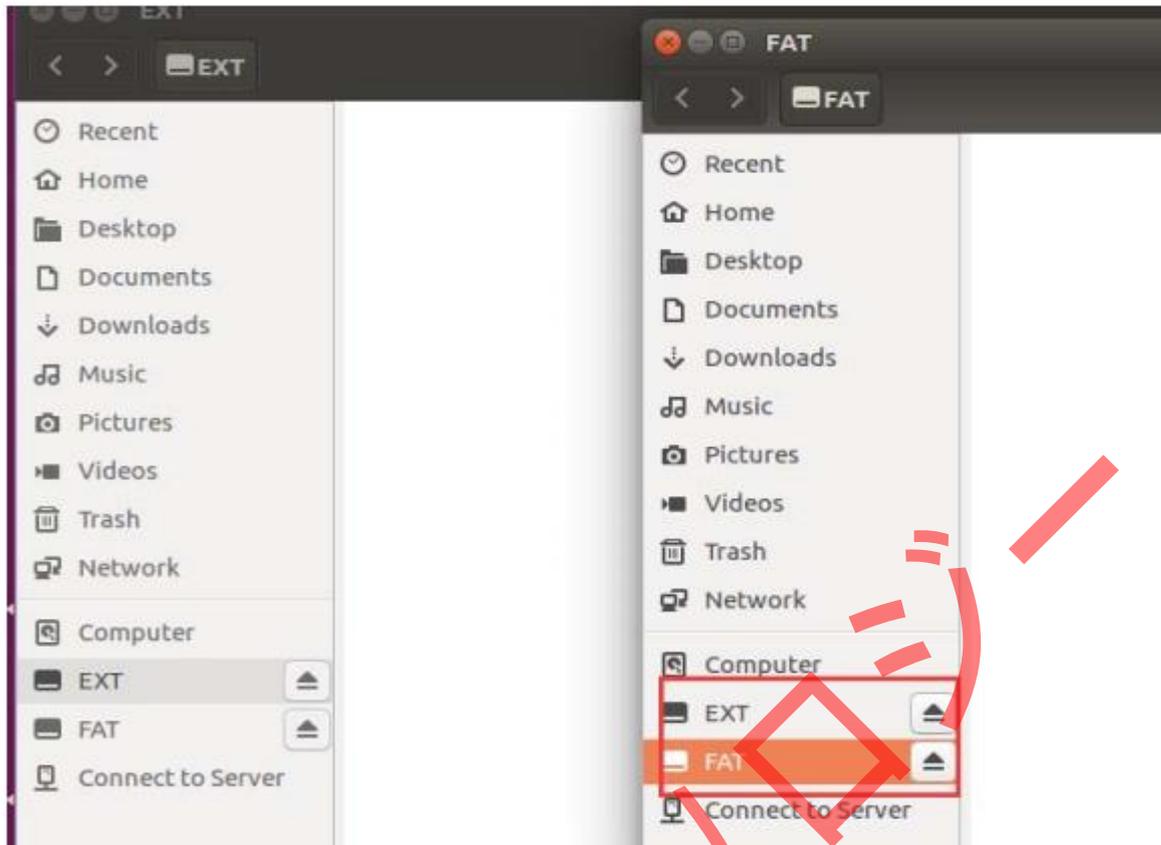
7) ルートファイルシステムを保存するため、第二のパーティションを作成し、フォーマットをEXT4にし、名前をEXTにする。



8) パーティションを作成した後、SD カードを再挿入する。

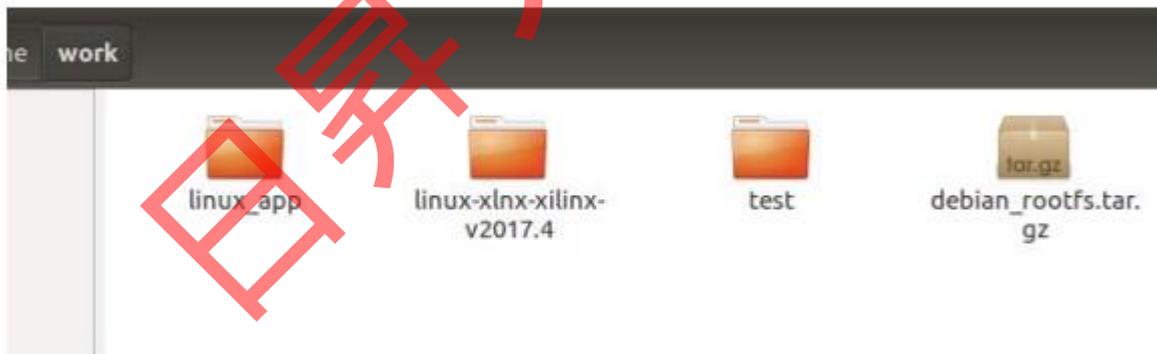


9) システムは自動的にパーティションをマウントし、ウィンドウをポップアップする

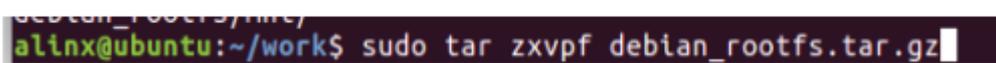


#### 22.4.2 ルートファイルシステムを SD カード EXT4 パーティションに同期する。

1) ルートファイルシステムの圧縮ファイルを Linux ホストにコピーする（実験を / home / alinx / work ディレクトリーにコピーする）



2) 解凍コマンド `sudo tar -zxvpf debian_rootfs.tar.gz` を入力して、ファイルシステムを抽出する。解凍には数分かかる場合がある。解凍方法はマニュアルと一致する必要があることに注意してください。



3) これらのファイルシステムのすべてのファイルを SD カードの EXT パーティションのルートディレクトリにコピーする必要がある。コマンドウィンドウでコマンド `cd debian_rootfs` を入力して、ルートファイルシステムのディレクトリに入る。

```

alinx@ubuntu: ~/work/debian_rootfs
debian_rootfs/usr/share/zoneinfo/Australia/Adelaide
debian_rootfs/usr/share/zoneinfo/Australia/Eucla
debian_rootfs/usr/share/zoneinfo/Australia/Darwin
debian_rootfs/usr/share/zoneinfo/Australia/Brisbane
debian_rootfs/usr/share/zoneinfo/Australia/North
debian_rootfs/usr/share/zoneinfo/Australia/NSW
debian_rootfs/usr/share/zoneinfo/Australia/Yancowinna
debian_rootfs/usr/share/zoneinfo/Australia/Canberra
debian_rootfs/usr/share/zoneinfo/Australia/West
debian_rootfs/usr/share/zoneinfo/Australia/Queensland
debian_rootfs/usr/share/zoneinfo/Australia/Perth
debian_rootfs/usr/share/zoneinfo/Australia/Sydney
debian_rootfs/usr/share/zoneinfo/Australia/Tasmania
debian_rootfs/usr/share/zoneinfo/Jamaica
debian_rootfs/usr/share/zoneinfo/GB-Eire
debian_rootfs/usr/share/zoneinfo/PST8PDT
debian_rootfs/usr/share/zoneinfo/EST
debian_rootfs/usr/share/zoneinfo/CST6CDT
debian_rootfs/usr/share/zoneinfo/Hongkong
debian_rootfs/usr/share/zoneinfo/localtime
debian_rootfs/usr/share/zoneinfo/Poland
debian_rootfs/mnt/
alinx@ubuntu:~/work$ cd debian_rootfs
alinx@ubuntu:~/work/debian_rootfs$
  
```

4) コマンドウィンドウで、`sudo rsync -av ./ /media/alinx/EXT` (`/media/alinx/EXT` を入力し、これはSDカードのEXT4パーティションのパスである。異なる場合があり、実際の状況に応じて変更してください。現在のディレクトリをSDカードのEXTパーティションのルートディレクトリに同期し始め、同期には数十分かかる場合がある。

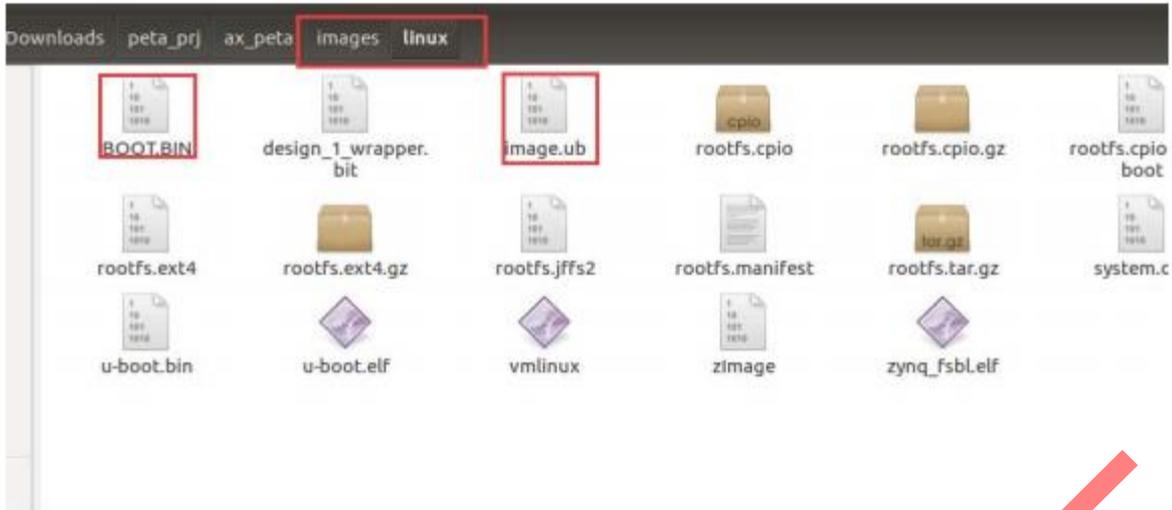
```

var/log/dpkg.log
var/log/faillog
var/log/fontconfig.log
var/log/lastlog
var/log/wtmp
var/log/apt/
var/log/apt/history.log
var/log/apt/term.log
var/log/fsck/
var/log/fsck/checkfs
var/log/fsck/checkroot
var/log/ntpstats/
var/mail/
var/opt/
var/spool/
var/spool/mail -> ../mail
var/spool/cron/
var/spool/cron/crontabs/
var/spool/rsyslog/
var/tmp/

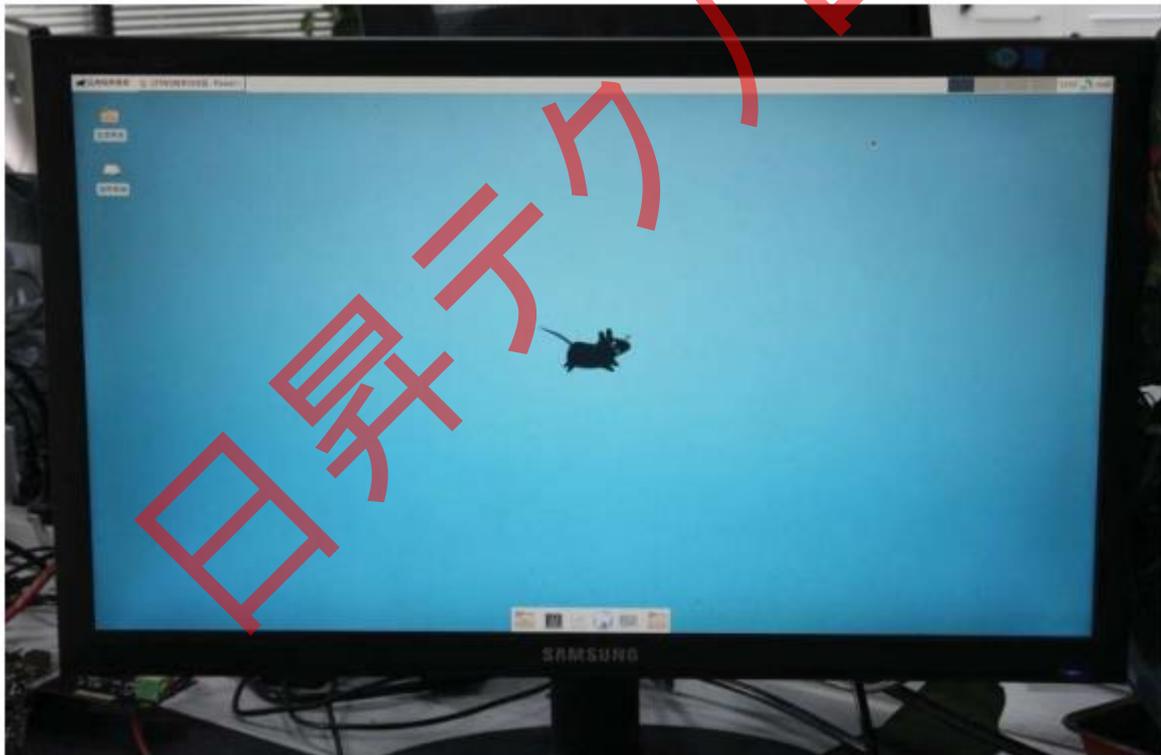
sent 1,117,338,469 bytes  received 809,451 bytes  5,414,759.90 bytes/sec
total size is 1,114,115,109  speedup is 1.00
alinx@ubuntu:~/work/debian_rootfs$ sudo rsync -av ./ /media/alinx/EXT
  
```

5) コマンドプロンプトがコマンドラインに再表示されると、同期が終了する意味である。

6) `BOOT.bin` と `image.ub` をSDのFAT32パーティション（最初のパーティション）にコピーし、開発ボードのsdモードを開始に設定し、HDMIディスプレイを差し込み、開発ボードを開始する。



7) SD カードを作成したら、できた SD カードを開発ボードの SD カードスロットに挿入する。USB シリアルケーブルを接続、HDMI ディスプレイを接続して、ボードの電源を入れた後、Debian オペレーティングシステムのインターフェースが HDMI ディスプレイに表示される。さらに、シリアルポートツールでは、オペレーティングシステムを起動するプロセスも確認できる。u-boot を実行後、Linux の実行を開始する。アカウント : root、パスワード : root。



8) システムに入ったら、ifconfig コマンドを使用してネットワーク接続を確認できる。

```
root@zynq:~# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:0a:35:00:1e:53
      inet addr:192.168.1.46  Bcast:192.168.1.255  Mask:255.255.255.0
      inet6 addr: fe80::20a:35ff:fe00:1e53/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:119 errors:0 dropped:0 overruns:0 frame:0
      TX packets:86 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:12231 (11.9 KiB)  TX bytes:7420 (7.2 KiB)
      Interrupt:29 Base address:0xb000

eth1  Link encap:Ethernet  HWaddr 00:0a:35:00:03:22
      inet addr:192.168.1.62  Bcast:192.168.1.255  Mask:255.255.255.0
      inet6 addr: fe80::20a:35ff:fe00:322/64 Scope:Link
      UP BROADCAST RUNNING  MTU:1500  Metric:1
      RX packets:26 errors:0 dropped:0 overruns:0 frame:0
      TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:2771 (2.7 KiB)  TX bytes:1184 (1.1 KiB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
      RX packets:236 errors:0 dropped:0 overruns:0 frame:0
      TX packets:236 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:18960 (18.5 KiB)  TX bytes:18960 (18.5 KiB)
```



## 第二十三章 QSPI Flash から起動の Linux の作成

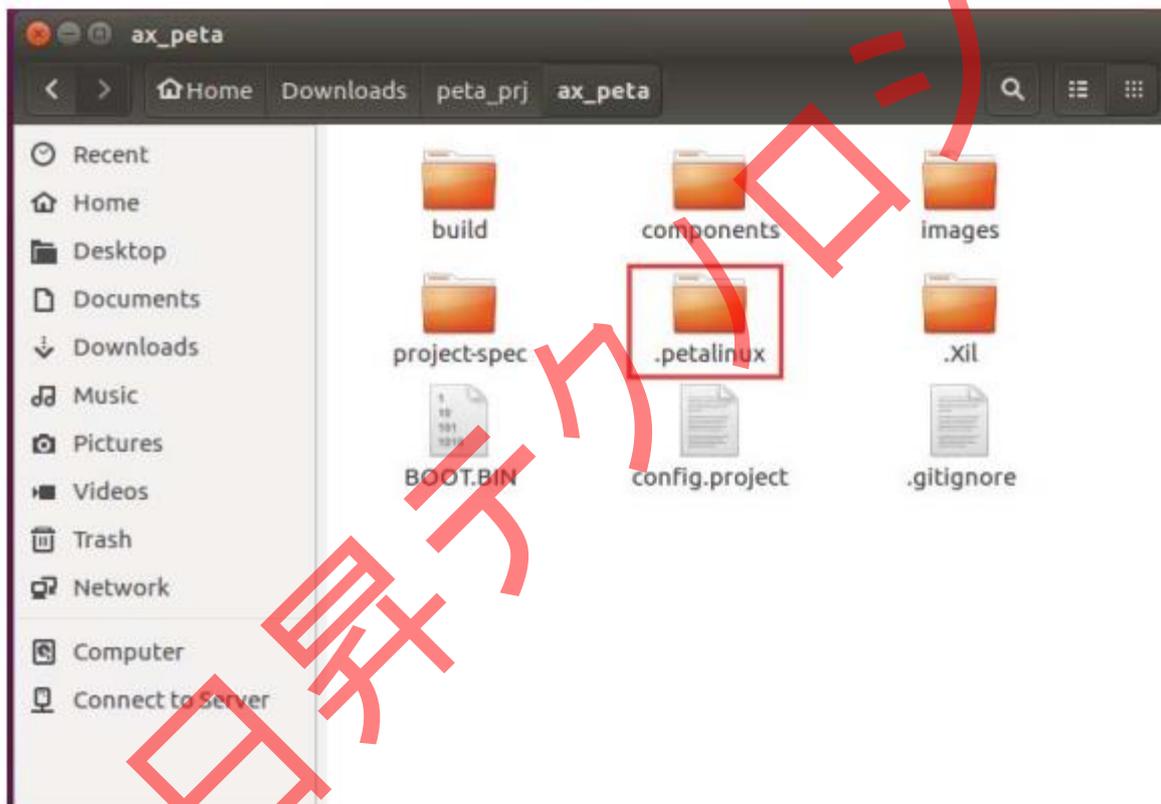
前のマニュアルで説明した Linux はすべて SD で開始するが、この実験では、Petalinux を使用して QSPI Flash から起動の Linux を作成する方法について説明する。

ここでは、ボード上の QSPI Flash サイズは 32M バイトだが、ZYNQ チップは 16MB しか使用できないため、ZYNQ 自体によって決定してください。Linux システムは 16MB を超えると使えなくなる。

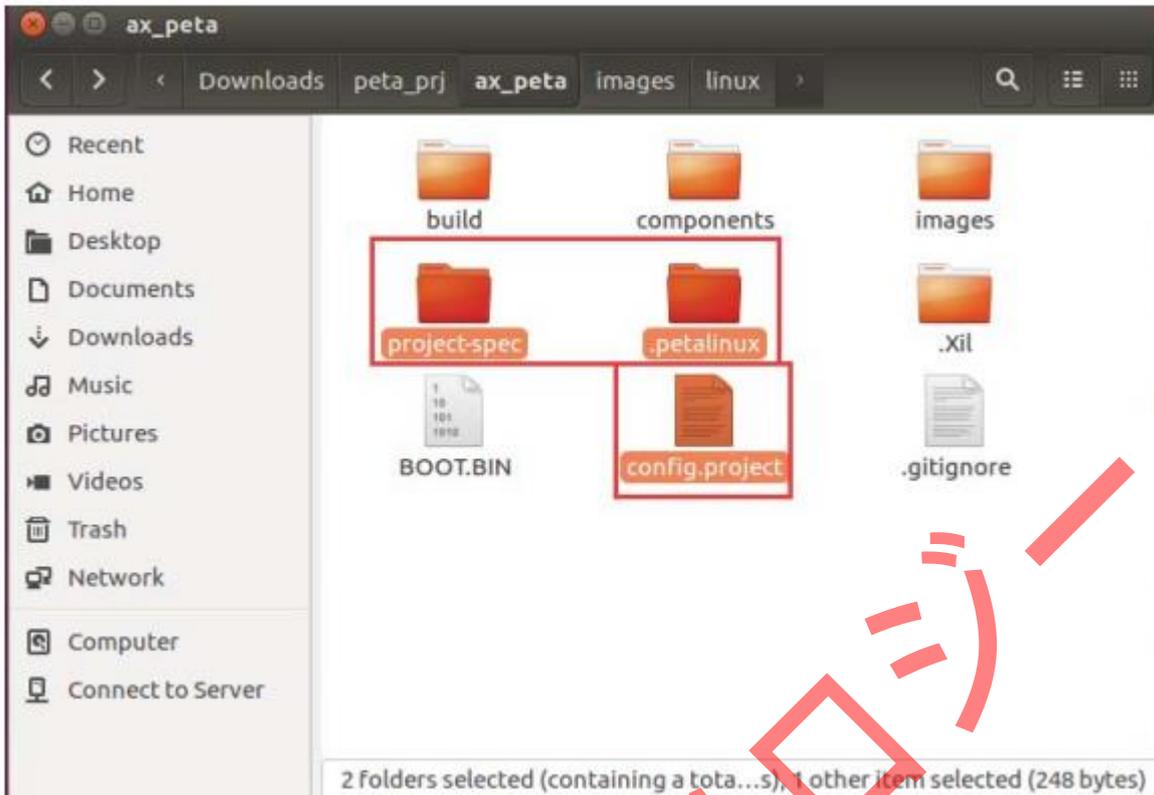
### 23.1 Petalinux プロジェクトをコピーする

前のマニュアルでは、Petalinux を使用して SD カードのブートのさまざまな実験を行った。SD ブートプロジェクトを保持したいが、新しいプロジェクトを作成したくないため、古いプロジェクトをコピーできる。

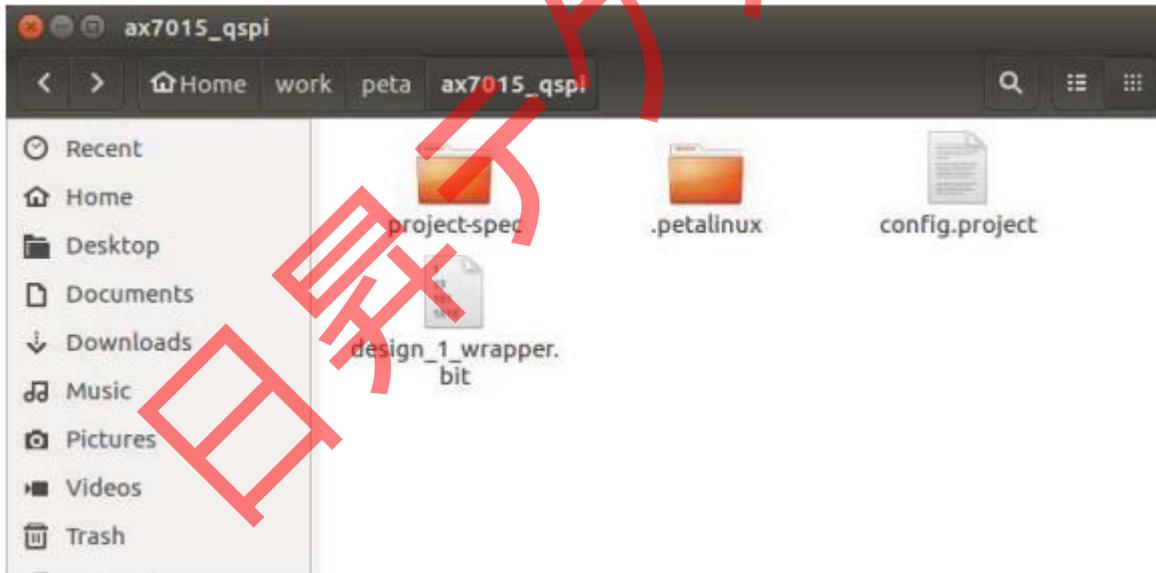
1) プロジェクトディレクトリで、Ctrl + H を同時に押して、シャドウファイルが表示される。



2) project-spec、.petalinux、config.project を新しいディレクトリにコピーして、新しい Petalinux プロジェクトにする。



3) PL コンフィギュレーションの BOOT を合成するために、images / linux ディレクトリの bit ファイルも新しいプロジェクトディレクトリにコピーする。



## 23.2 Petalinux のコンパイルとコンフィグ

1) 下のコマンドで環境変数を設定する

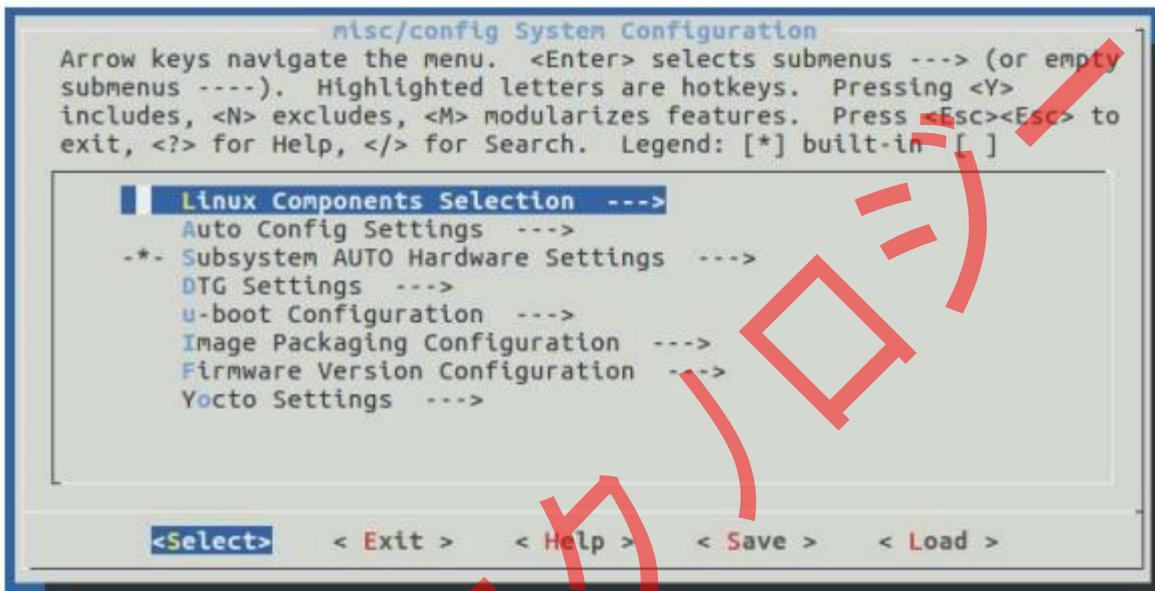
```
source /opt/pkg/petalinux/settings.sh
```

```
source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

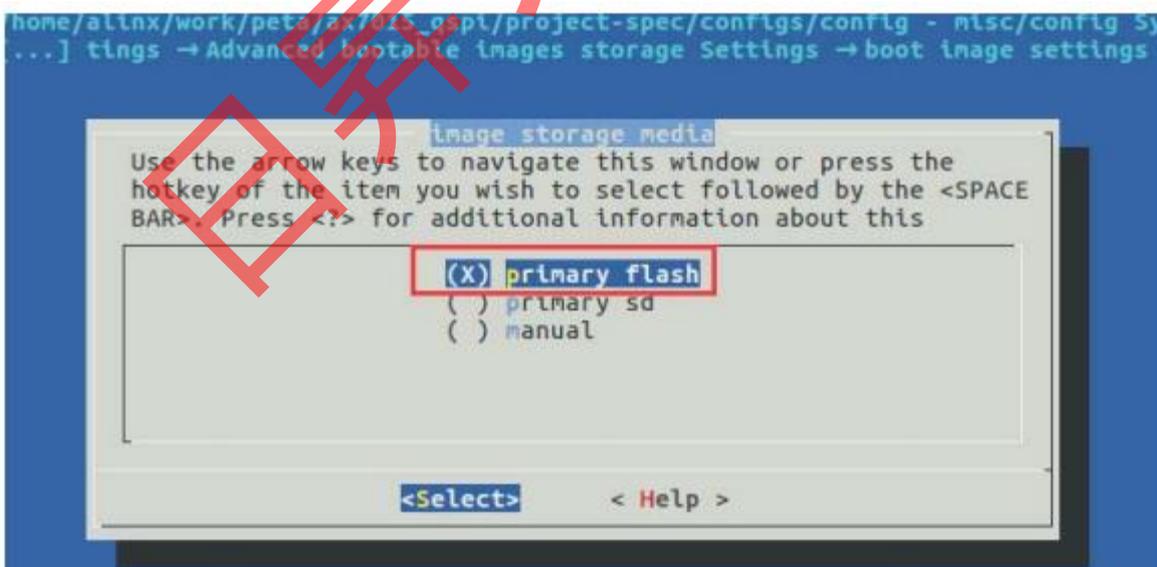
```
alinx@ubuntu:~/work/peta/ax7015_qspi$ source /opt/pkg/petalinux/settings.sh
```

```
alinx@ubuntu:~/work/peta/ax7015_qspi$ source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

- 2) petalinux-config コマンドを使用して petalinux をコンフィグする



- 3) Subsystem AUTO Hardware Settings → Advanced bootable images storage Settings → boot image settings → image storage media オプションから primary flash を選択する。



- 4) Subsystem AUTO Hardware Settings → Advanced bootable images storage Settings → kernel image settings → image storage media オプションから primary flash を選択する。

```

/home/alinx/work/peta/ax7015_qspi/project-spec/configs/config - misc/config Sys
[... ] ngs → Advanced bootable images storage Settings → kernel image settings
kernel image settings
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

image storage media (primary flash) --->
(kernel) flash partition name
(image.ub) image name

<Select> <Exit> <Help> <Save> <Load>
  
```

5) Subsystem AUTO Hardware Settings → Flash SettingsはQSPI Flashのパーティションを変更できる。デフォルトはほとんど使用可能だが、ファイルサイズがデフォルトのパーティションサイズを超える場合は、自分で調整する必要がある。

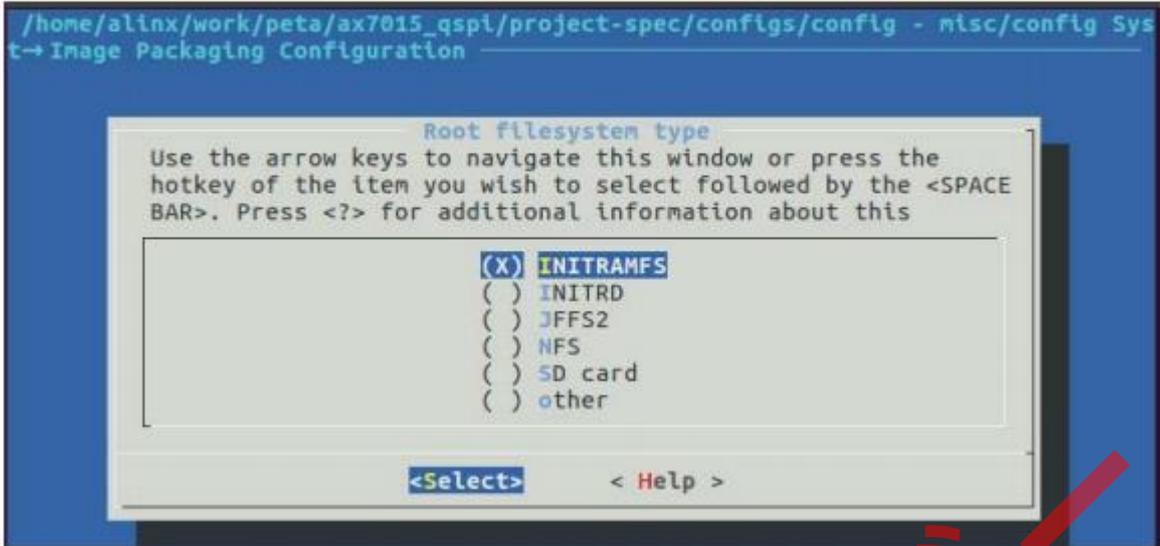
```

/home/alinx/work/peta/ax7015_qspi/project-spec/configs/config - misc/config Sys
t→ Subsystem AUTO Hardware Settings → Flash Settings
Flash Settings
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

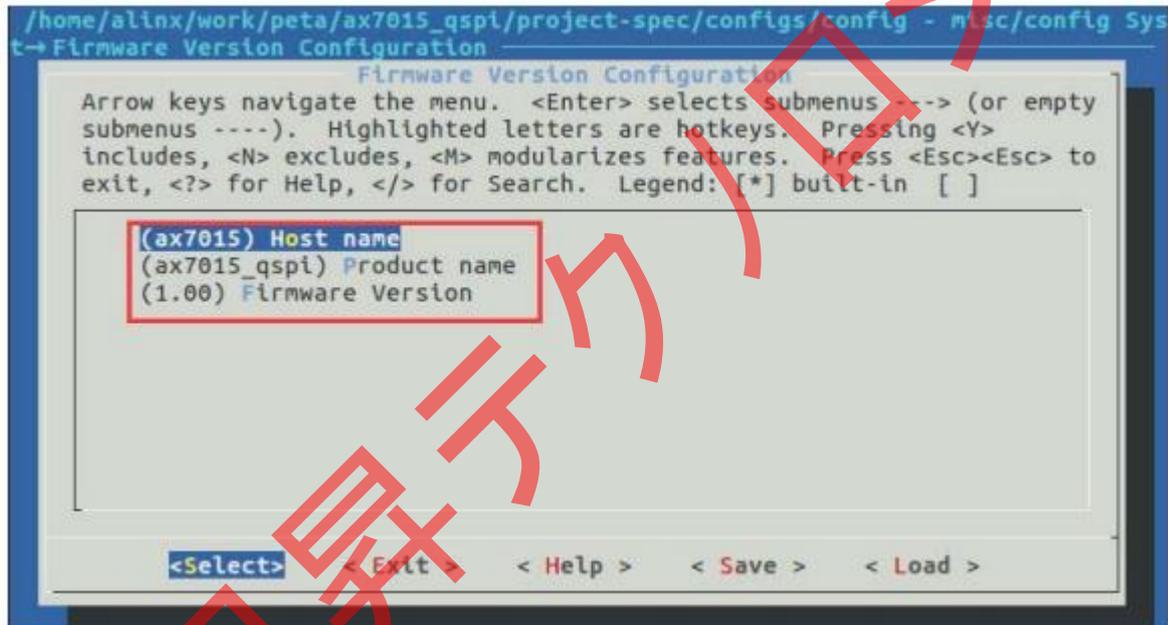
Primary Flash (ps7 qspi 0) --->
[ ] Advanced Flash Auto Configuration
*** partition 0 ***
(kernel) name
(0x500000) size
*** partition 1 ***
(kernel) name
(0x20000) size
*** partition 2 ***
(kernel) name

<Select> <Exit> <Help> <Save> <Load>
  
```

6) Image Packaging Configuration ---> Root filesystem typeにINITRAMFSを選択して、RAMタイプのルートファイルシステムを使用することにより、簡単にパッケージ化してQSPIflashに書き込むことができる。



7) Firmware Version Configuration --->に Host name 等の情報を修正できる。



8) コンフィグを保存する。

9) コンパイル

```

*** Execute 'make' to start the build or try 'make help'.
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/work/peta/ax7015_qspi/build/misc/plnx-generated ~/work/peta/ax7015_qspi
~/work/peta/ax7015_qspi
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
~/work/peta/ax7015_qspi/build/misc/plnx-generated ~/work/peta/ax7015_qspi
~/work/peta/ax7015_qspi
[INFO] generating u-boot configuration files

[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
[INFO] successfully configured project
webtalk failed:PetaLinux statistics:extra lines detected:notsent_nofile!
webtalk failed:Failed to get PetaLinux usage statistics!
alinx@ubuntu:~/work/peta/ax7015_qspi$ petalinux-build
  
```

10) 下のコマンドを使用してBOOTを合成する。以前のマニュアルとの違いは、-kernel オプションが追加され、カーネルをBOOT.bin ファイルにパッケージ化することである。

```

petalinux-package --boot--fsbl ./images/linux/zynq_fsbl.elf --fpga --u-boot --kernel --force
  
```

```

alinx@ubuntu:~/Downloads/peta_prj/ax_peta$ petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga --u-boot --kernel --force

INFO: Getting system flash information...
rlwrap: warning: your $TERM is 'xterm-256color' but rlwrap couldn't find it in the terminfo database. Expect some problems.: Inappropriate ioctl for device

INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/zynq_fsbl.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/design_1_wrapper.bit"
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/u-boot.elf"
INFO: File in BOOT BIN: "/home/alinx/Downloads/peta_prj/ax_peta/images/linux/image.ub"
INFO: Generating zynq binary package BOOT.BIN...
INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
webtalk failed:Invalid tool in the statistics file:petalinux-yocto!
webtalk failed:Failed to get PetaLinux usage statistics!
  
```

- 11) 第十三章を参照して、BOOT.bin を QSPIflash に書き込むことができる。
- 12) 起動モードを QSPI に調整する

以上。