Zynqberry×ZYNQ 活用セミナー

第2章　Zynqberryの使い方

2018年6月13日

特殊電子回路株式会社

目次

- ハードウェア
- MIO
- CPLD
- GPIO
- リファレンスデザイン
- サンプルプロジェクト
- Trenz製Debian Linuxの所在
- Trenzスクリプトの使い方
- 最初の書き込み
- Linuxの起動
- Linux上からのROM書き換え
- SDSoCについて
- 作成したFPGAとソフトのROM化

Zynqberryのハードウェアの特徴①

- XILINX ZYNQ XC7Z010-1CLG225C
  - 割と小規模。より小規模な7S版も最近はある。
- DDR3 SDRAM 512MB
  - ZYNQの最大メモリが1GBなのでMAXの半分
- LAN9514
  - ZYNQのGbEを使わない。USB HUBと兼用
- HDMI出力
  - 標準では720p。1080pもいけるはず。
- PWMによるオーディオ出力

Zynqberryのハードウェアの特徴②

- DSIコネクタ
- CSIコネクタ
  - CSIを受信するための回路が秀逸
  - CSIを利用するサンプルデザインが提供されている
- マルチプレクサでI2Cを拡張
- アナログ入力(V_P、V_N)はオーディオコネクタに接続
- ボードコントロール用にCPLDを搭載
- RTCは非搭載
- ユーザが使えるLEDはない

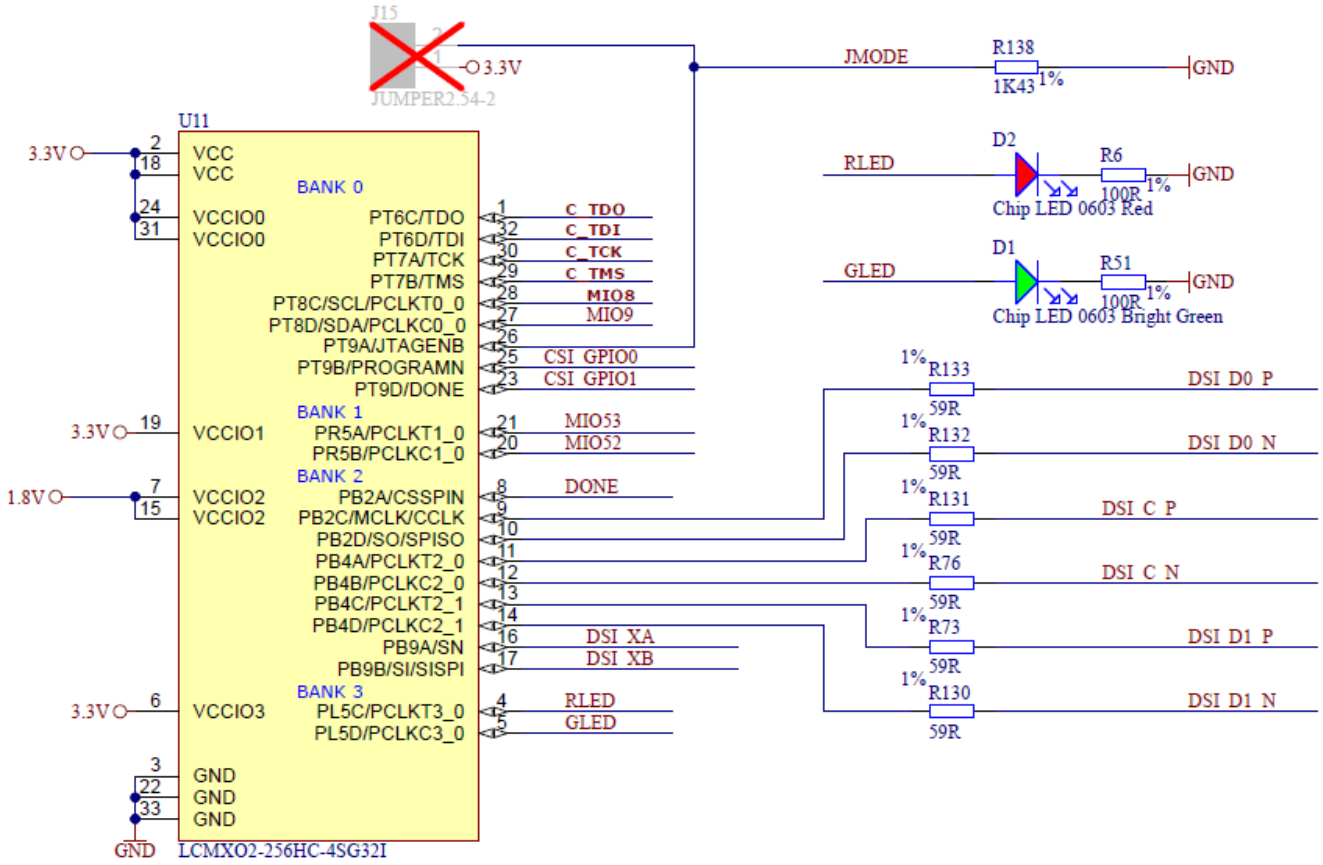
MIOの割り当て

MIO	機能	
0	I2C MUX	INT信号の受信?
1-6	QSPI FLASH ROM	S25FL127SABMFV10
7	USB RESET	Lでリセット
8-9	UART1	CPLD経由でUSBへ出力
10-15	SD1	
28-39	USB OTG	USB3320C
48-49	I2C1 MUX	TCA9544APWR
52-53	CPLD	
EMIO	I2C0	サンプルデザインでは未接続
EMIO	GPIO	24bit幅
EMIO	TTC	サンプルデザインでは未接続

※電源電圧は3.3V

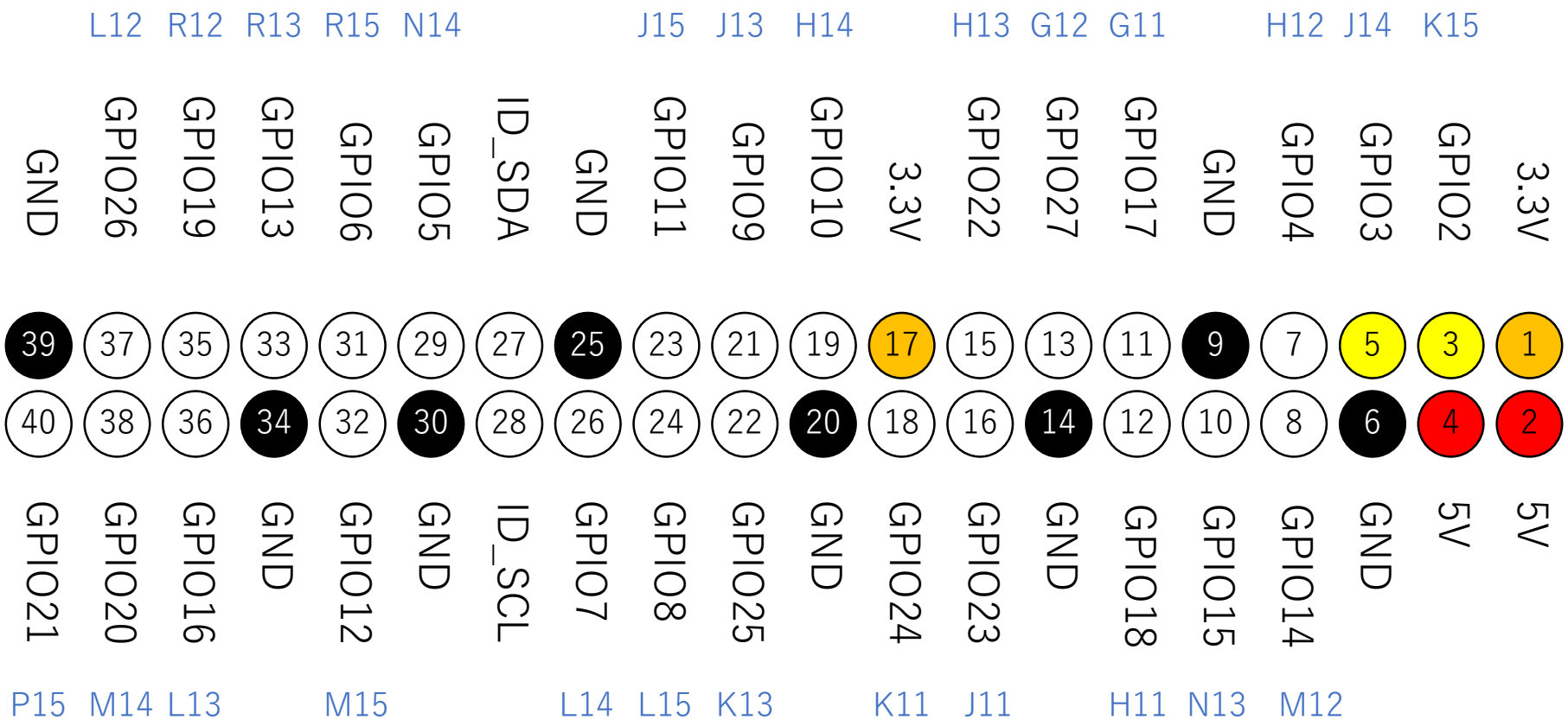
CPLDについて

- 詳細は不明
  - LEDの点灯
  - DSI(ディスプレイ)のコントロール
  - CSIのGPIO操作
  - MIO8,9をおそらくC_TDO、C_TDI経由でUSB-UARTに送っている



GPIOについて

GPIOのピン割り当て



※GPIO 14,15はEMIOからは操作できない
※GPIO2,3はプルアップされている

GPIOの端子割り当て一覧

index	FPGAピン	GPIO番号	ピンヘッダ
0	K15	2	3
1	J14	3	5
2	H12	4	7
3	N14	5	29
4	R15	6	31
5	L14	7	26
6	L15	8	24
7	J13	9	21
8	H14	10	19
9	J15	11	23
10	M15	12	32
11	R13	13	33
12	L13	16	36
13	G11	17	11
14	H11	18	12

index	FPGAピン	GPIO番号	ピンヘッダ
15	R12	19	35
16	M14	20	38
17	P15	21	40
18	H13	22	15
19	J11	23	16
20	K11	24	18
21	K13	25	22
22	L12	26	37
23	G12	27	13
24	F13		DSI_D0_N
25	F14		DSI_D0_P
26	F12		DSI_D1_N
27	E13		DSI_D1_P
28	E11		DSI_C_N
29	E12		DSI_C_P

index	FPGAピン	GPIO番号	ピンヘッダ
30	M11		CSI_D0_N
31	M10		CSI_D0_P
32	P14		CSI_D1_N
33	P13		CSI_D2_P
34	N12		CSI_C_N
35	N11		CSI_C_P
36	N8		PWM_R
37	N7		PWM_L

FPGAのGPIO[23:0]を操作すると
ピンヘッダに出ている
GPIO端子の2~27に反映される

GPIO 14,15はEMIOからは操作できない

FPGAのバスの
インデックス

RasPiでの番号

GPIOの操作方法

- GPIO汎用ドライバ(/sys/class/gpio)を使う
  - Linuxのシステムコールを使い、/sys/class/gpioにある仮想ファイルを操作します。
  - シェルでの操作をC言語から行うこともできます。

- /dev/memを使う方法
  - GPIOの物理アドレスに直接データを書き込む

GPIOドライバの説明

- /sys/class/gpio にPSからGPIOを操作するドライバあり
  - /sys/class/gpio/gpio906～gpio1023が1つ1つのbitに対応
- 906~959はMIO経由のGPIO(54本)
- 960~1023はEMIO経由のGPIO(64本)
  - RasPiピン番号が 2~13の場合："ピン番号+958"番の仮想ファイル
  - RasPiピン番号が16~26の場合："ピン番号+956"番の仮想ファイル

- gpio960 = FPGAのEMIO GPIO[0] = RasPiのGPIO2
  - EMIO GPIO0~GPIO11 → RasPiのGPIO 2~13 → gpio960~971
  - EMIO GPIO12~GPIO23 → RasPiのGPIO 16~27 → gpio972~983

GPIOドライバの操作方法①

/sys/class/gpioディレクトリの中身を確認してみます

```
root@zynqberry ~$ ls /sys/class/gpio
export  gpiochip906  unexport
```

※906はMIO0

このexportへ使いたいGPIOの番号を書き出すと、そのGPIOを使うための仮想ファイルが用意されます。
今回はGPIO2へのアクセスなので960番を書き出します。

```
root@zynqberry ~$ echo 960 > /sys/class/gpio/export
root@zynqberry ~$ ls /sys/class/gpio/
export  gpio960  gpiochip906  unexport
```

新しくできた/sys/class/gpio/gpio960の中身は

```
root@zynqberry ~$ ls /sys/class/gpio/gpio960
active_low  device  direction  edge  power  subsystem  uevent  value
```

GPIOドライバの操作方法②

仮想ファイルdirectionはGPIOの入出力の制御、
仮想ファイルvalueはGPIOの値(1ならHigh、0ならLow)の制御に使います。
まずGPIO 2を出力用にするため

```
root@zynqberry ~$ echo "out" >  /sys/class/gpio/gpio960/direction
root@zynqberry ~$ cat /sys/class/gpio/gpio906/direction
out
```

この状態でvalueに1や0を書き出すと、GPIOに接続したLEDを点滅させることができます。

```
root@zynqberry ~$ echo 1 >  /sys/class/gpio/gpio960/value
root@zynqberry ~$ echo 0 >  /sys/class/gpio/gpio906/value
```

/dev/memを使う方法

- GPIOのマップされているIOアドレス0xE000A000
- 1bitごとに1つのbitが対応している

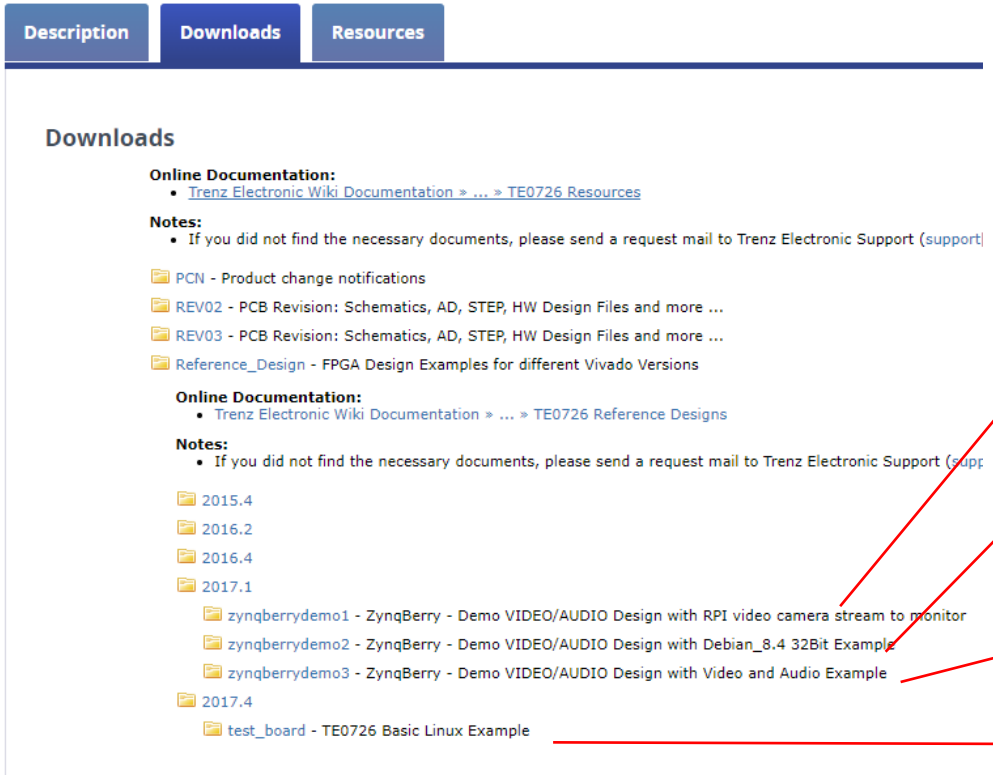
オフセット	レジスタ名	機能
0x00000284	DIRM_2	GPIO入出力。0で入力、1で出力
0x00000288	OEN_2	出力イネーブル
0x00000048	DATA_2	GPIOピンへの出力値

例:0xE000A284のbit0を1にするとEMIO(0)=RasPiGPIO(2)が出力になる

サンプルプロジェクト（リファレンスデザイン）

リファレンスデザイン

- 現在、4種類のリファレンスデザインが利用可能



zynqberrydemo1- ZynqBerry - Demo VIDEO/AUDIO Design with RPI video camera stream to monitor

zynqberrydemo2 - ZynqBerry - Demo VIDEO/AUDIO Design with Debian_8.4 32Bit Example

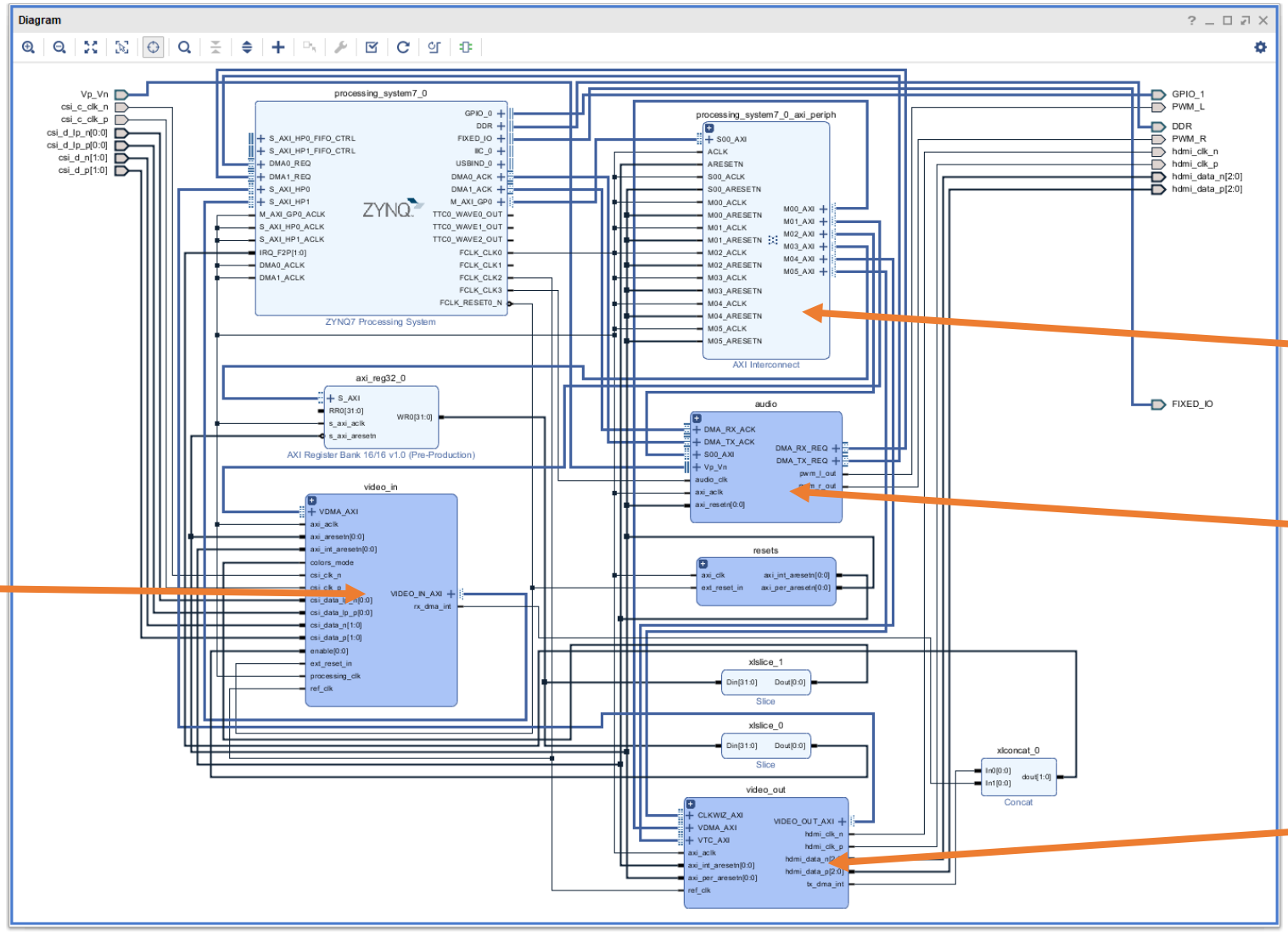
※Debianのイメージ入り

zynqberrydemo3 - ZynqBerry - Demo VIDEO/AUDIO Design with Video and Audio Example

本質的に同一

test_board - TE0726 Basic Linux Example ←シンプル

サンプルプロジェクト 2017.1



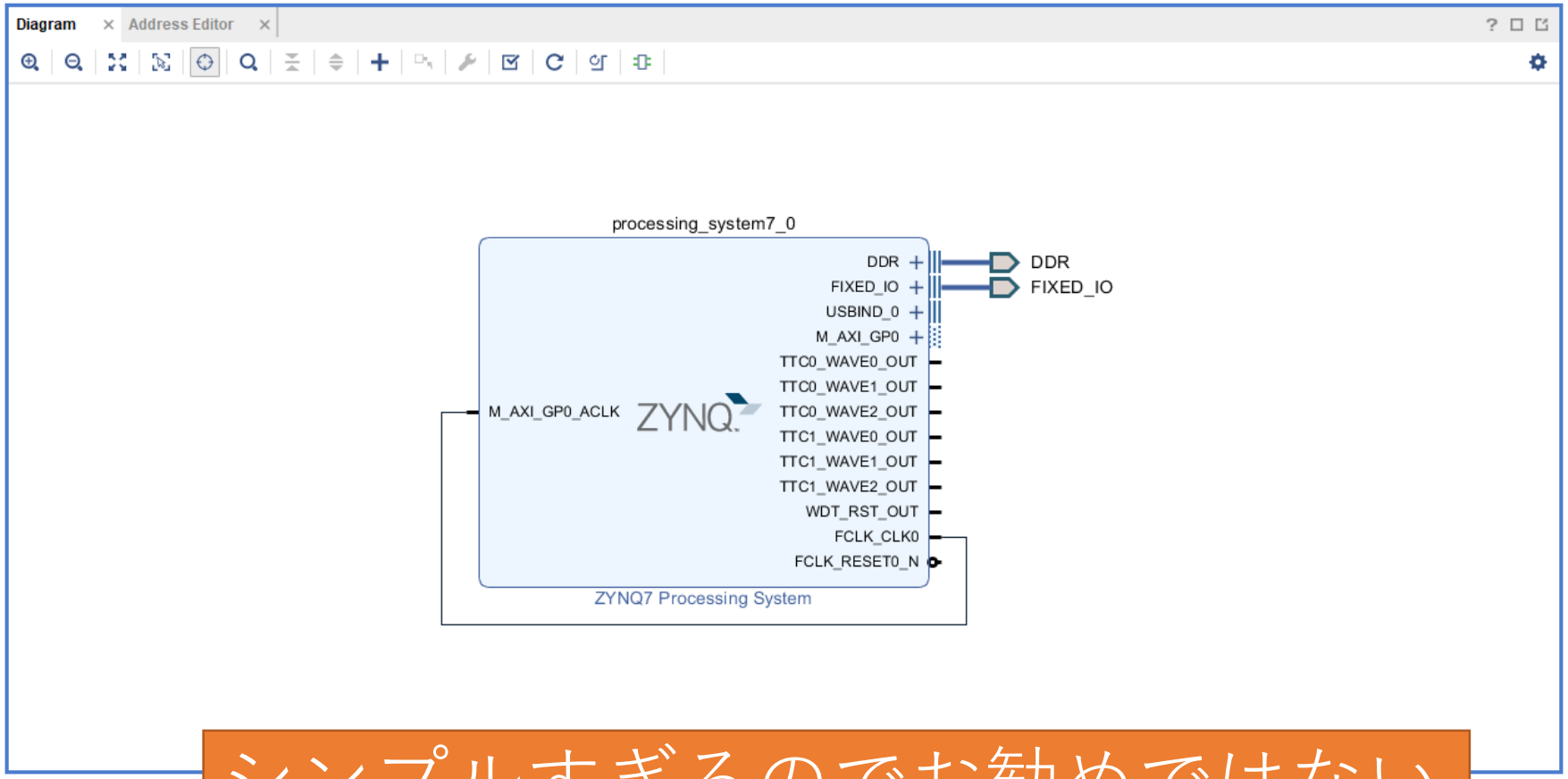
AXI Liteの分配

オーディオ

ビデオ入力

ビデオ出力

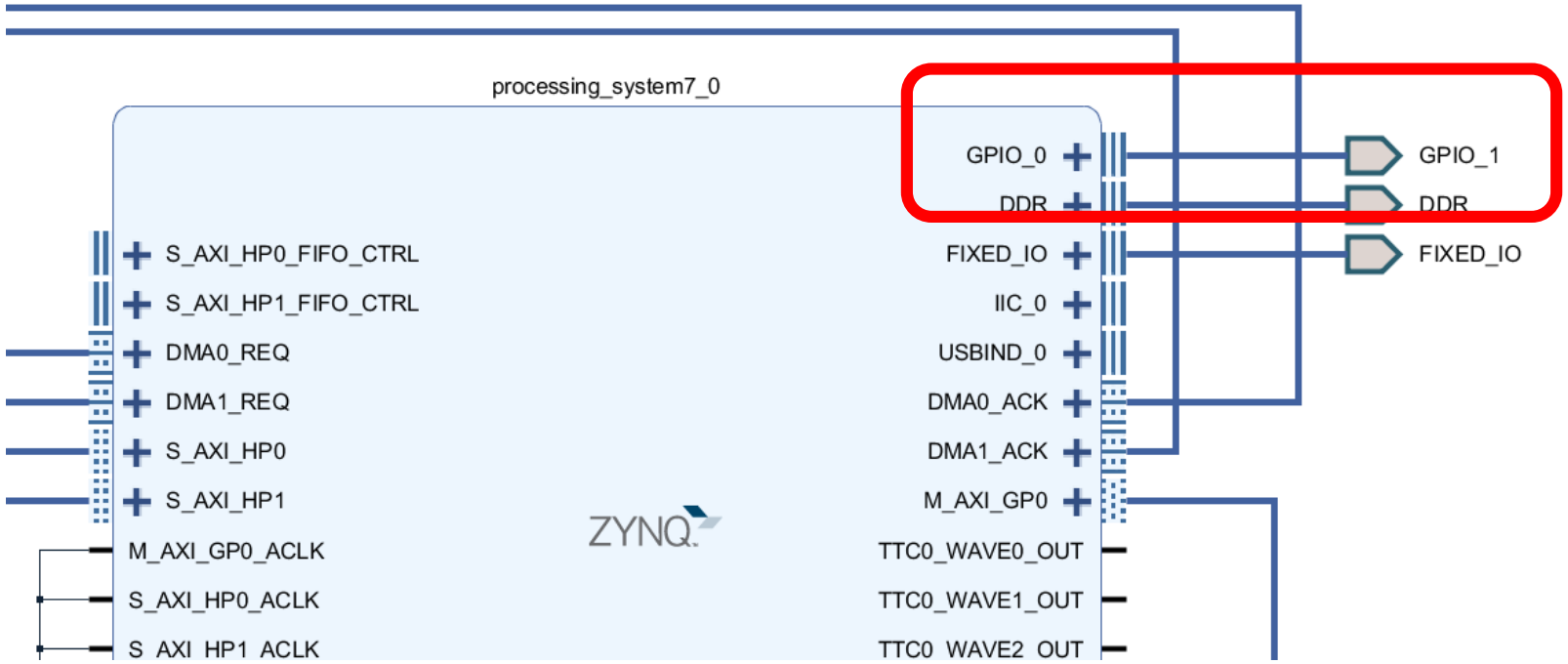
サンプルプロジェクト 2017.4



シンプルすぎるのでお勧めではない

バージョン間の相違(論理合成)

- Vivado2017.3以降では、MIOのEMIOから出る双方向ピンのImplementができない



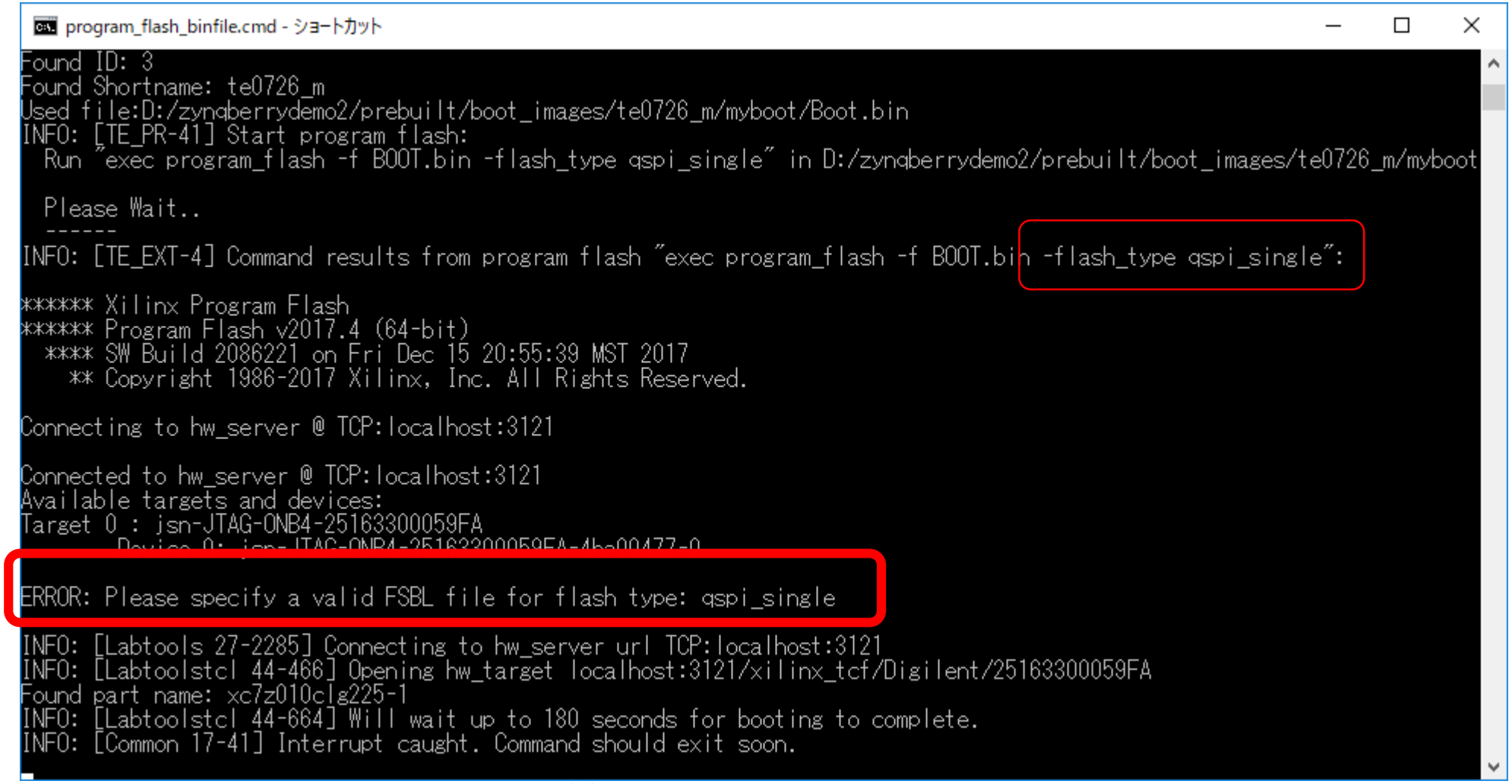
バージョン間の相違(書き込み)

- Vivado 2017.4を使用する
  - プロジェクト2017.1のスクリプトでboot.binの書き込みができない(qspiのFSBLかどうか確認せよ、と言われる)
  - プロジェクト2017.4のスクリプトでboot.binの書き込みはできる

- Vivado 2017.1または2を使用する
  - プロジェクト2017.1のスクリプトでboot.binの書き込みができる
  - プロジェクト2017.4のスクリプトでboot.binの書き込みはできる

書き込みに使うVivadoのバージョンは、Vivado 2017.1または2が良い

書き込み失敗のエラー画面

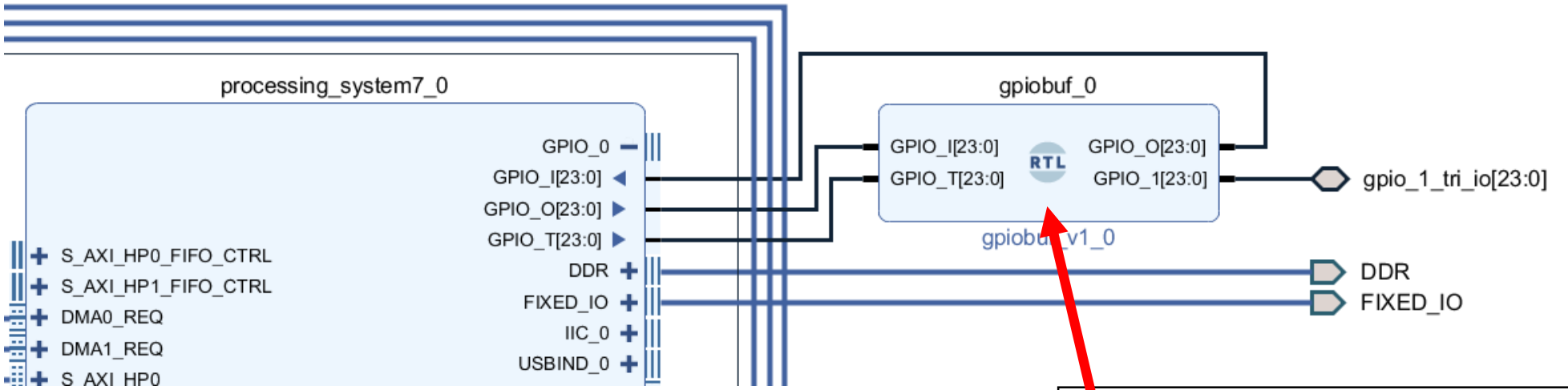


バージョン間の相違(まとめ)

サンプルプロジェクト のバージョン	機能	Vivado 2017.1~2	Vivado 2017.4	Debianイメージ
2017.1 zynqberrydemo2	リッチ ◎ (HDMI,PWM Audio, CSIカメラ,etc…)	論理合成〇 書き込み〇	論理合成× 書き込み×	あり
2017.4 test_board	シンプル × (何もない)	論理合成？ 書き込み〇	論理合成〇 書き込み〇	なし

Vivado 2017.3以降でzynqberrydemo2を使う方法

- 3ステートバッファを明示的に入れる

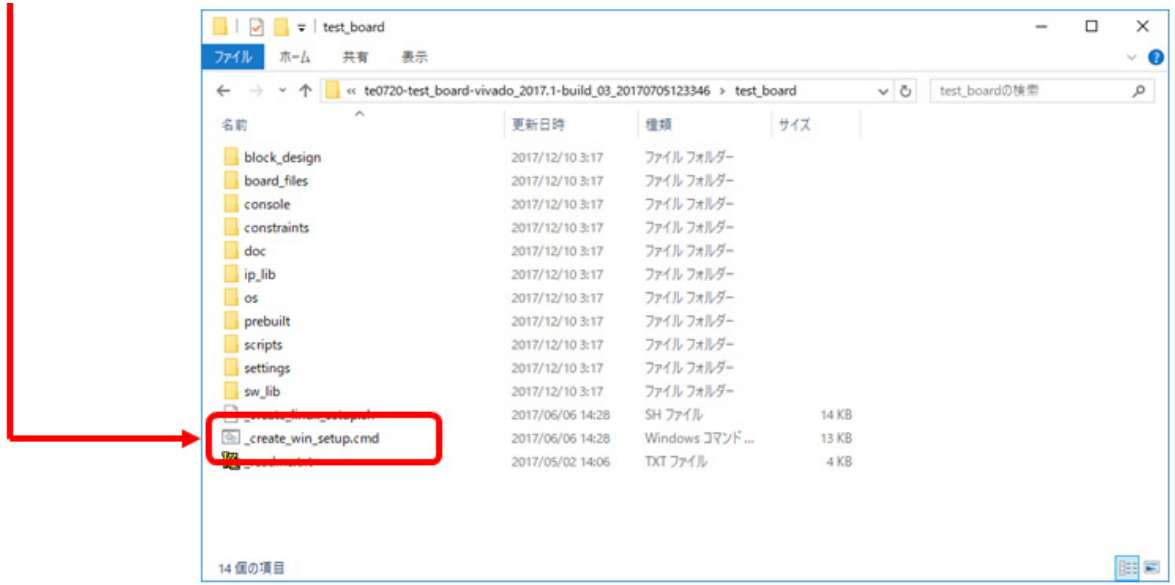



```
GPIO_LOOP : FOR i in 0 to 23 generate
    IOBUF_INST : IOBUF port map (
        IO => GPIO_1(i),
        O => GPIO_O(i),
        I => GPIO_I(i),
        T => GPIO_T(i)
    );
end generate;
```

Trenzスクリプト

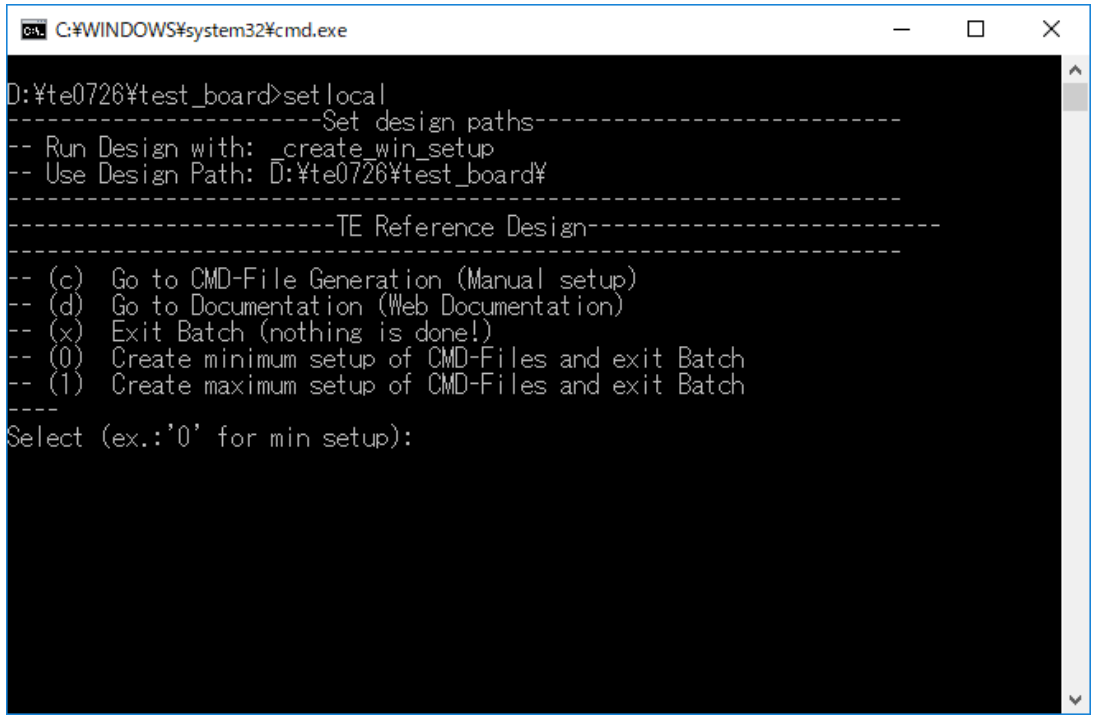
Trenzスクリプトの使い方

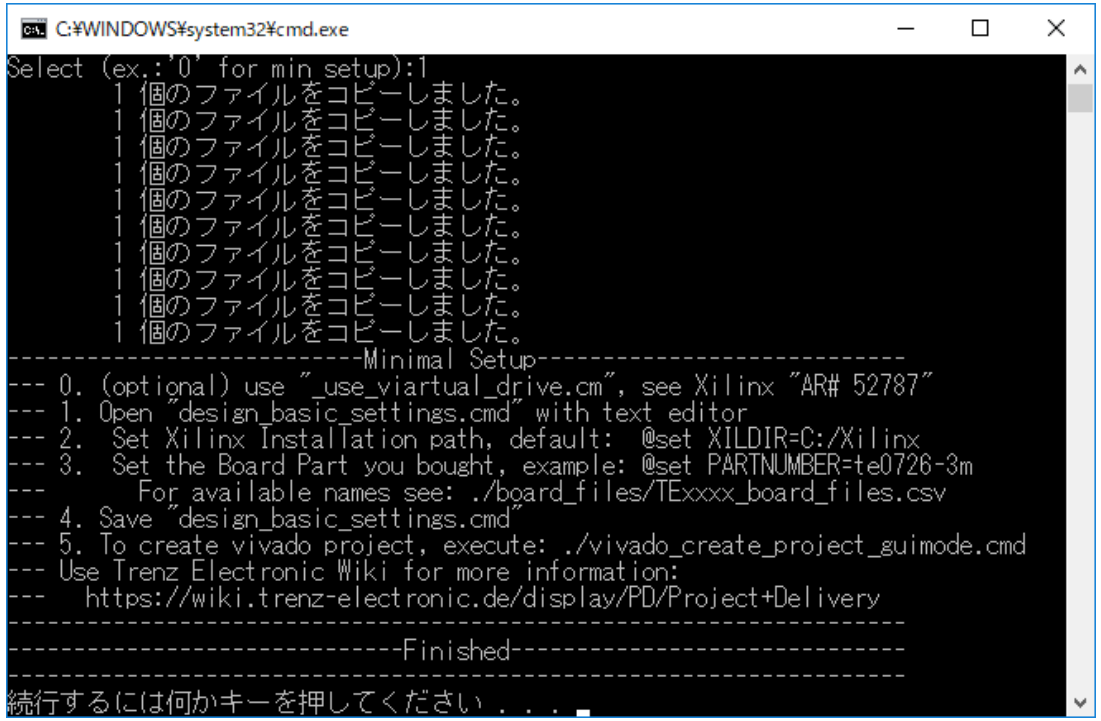
- Trenzスクリプトとは
  - Vivadoのプロジェクトの生成や、ビルドや書き込みを行うためのバッチファイルとTCLスクリプト
  - プロジェクトのフォルダの中にある_create_win_setup.cmdを実行



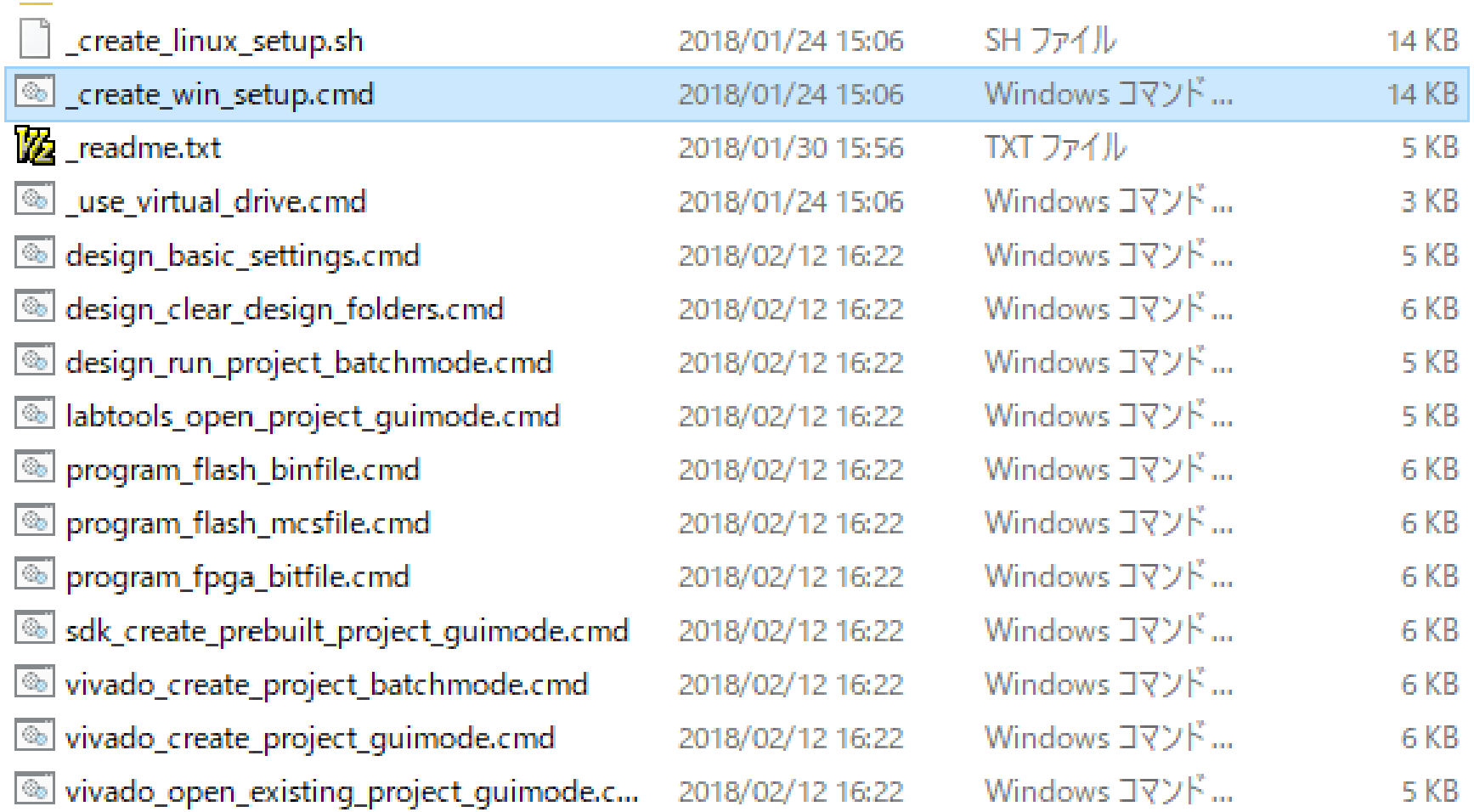
Trenzプロジェクトの生成

- DOSプロンプトが現れるので、「1」を押して、Enterを押します。





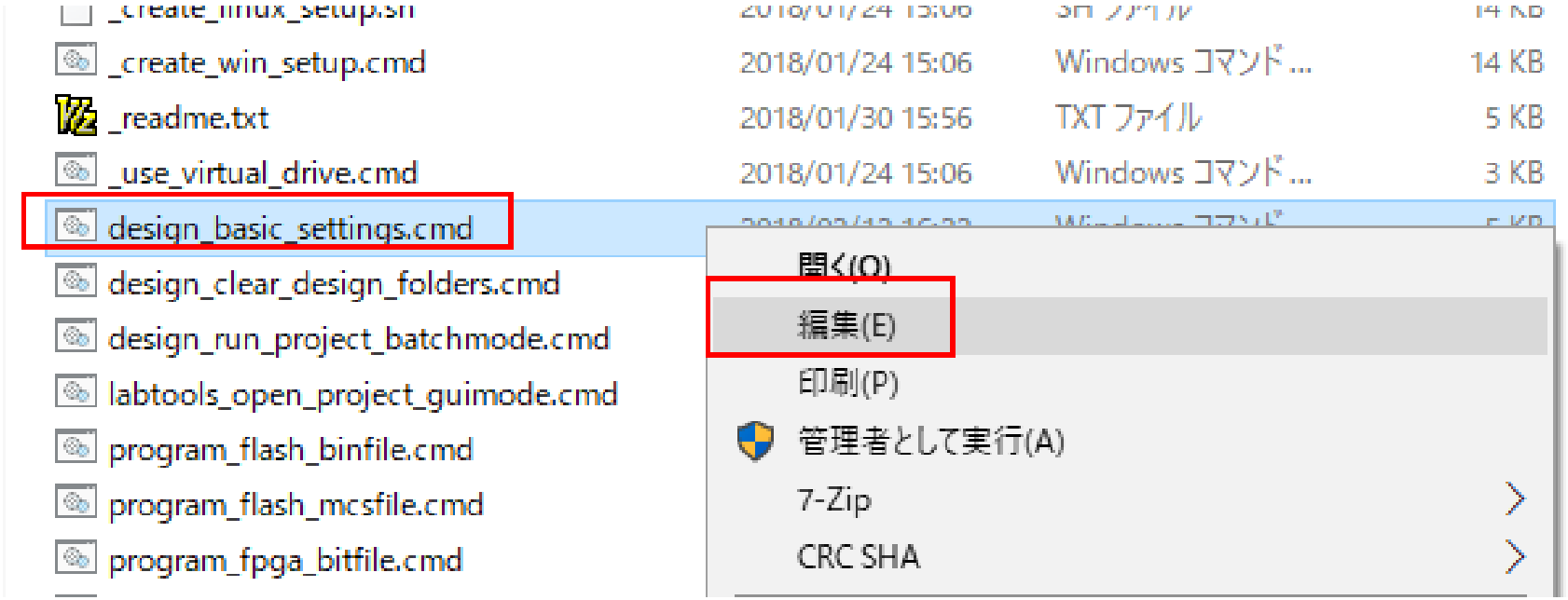
各種コマンドファイルが生成される



_create_linux_setup.sh	2018/01/24 15:06	SH ファイル	14 KB
_create_win_setup.cmd	2018/01/24 15:06	Windows コマンド...	14 KB
_readme.txt	2018/01/30 15:56	TXT ファイル	5 KB
_use_virtual_drive.cmd	2018/01/24 15:06	Windows コマンド...	3 KB
design_basic_settings.cmd	2018/02/12 16:22	Windows コマンド...	5 KB
design_clear_design_folders.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
design_run_project_batchmode.cmd	2018/02/12 16:22	Windows コマンド...	5 KB
labtools_open_project_guimode.cmd	2018/02/12 16:22	Windows コマンド...	5 KB
program_flash_binfile.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
program_flash_mcsfile.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
program_fpga_bitfile.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
sdk_create_prebuilt_project_guimode.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
vivado_create_project_batchmode.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
vivado_create_project_guimode.cmd	2018/02/12 16:22	Windows コマンド...	6 KB
vivado_open_existing_project_guimode.c...	2018/02/12 16:22	Windows コマンド...	5 KB

スクリプトのカスタマイズ

- design_basic_settings.cmdを右クリックし、編集



最低限編集すべき箇所

- XILINXディレクトリへのパス
- Vivadoのバージョン
- ボードの型番
- zsys_bd.tcl内のバージョン情報

```
@REM    -SUSoC (optional used for programming only): %XILDIR%¥SUSoC¥%VI
① @set XILDIR=C:/Xilinx
@REM    -Attention: These scripts support only the predefined Vivado Ver
② @set VIVADO_VERSION=2017.1
@REM --------------------
@REM Set Board part number of the project which should be created
@REM    -Available Numbers: (you can use ID,PRODID,BOARDNAME or SHORTNAM
@REM    -variable is used for project creation and programming
@REM    -Example TE0726 Module :
@REM      -USE ID              |USE PRODID                    |Use BOARDNAME
@REM       -@set PARTNUMBER=1  |@set PARTNUMBER=te0726-3m      |@set PARTNUMBER
③ @set PARTNUMBER=LAST_ID
@REM --------------------
```

使用するボード	PARTNUMBERに設定する値
TE0720-3-1CF(A) (Gigazee)	5
TE0720-3-2IF(A) (Gigazee)	1
TE0726-03M (Zynqberry)	3

block_designのバージョン

- 2017.1のプロジェクトファイルをVivado2017.1以外で使う場合は以下の変更が必要
  - block_designフォルダにzsys_bd.tclというファイルの25行目
    set scripts_vivado_version 2017.1

```
19 variable script_folder
20 set script_folder [_tcl::get_script_folder]
21
22 ################################################################
23 # Check if script is running in correct Vivado version.
24 ################################################################
25 set scripts_vivado_version 2017.1
26 set current_vivado_version [version -short]
27
28 if { [string first $scripts_vivado_version $current_vivado_version] == -1 } {
29    puts ""
```

よく使うスクリプト

スクリプト名	機能
design_run_project_batchmode.cmd	FPGAのビルドを行い、FSBLとU-Bootを結合してboot.binを生成する
design_clear_design_folders.cmd	v_logとVivadoプロジェクトを削除
vivado_create_project_guimode.cmd	Vivadoプロジェクトの生成
program_flash_binfile.cmd	生成されたboot.binを書き込み

基本的な流れ

1. vivado_create_project_guimode.cmd ・・・ プロジェクト作成
2. design_clear_design_folders.cmd ・・・プロジェクトクリーンアップ
3. design_run_project_batchmode.cmd ・・・ ビルド
4. program_flash_binfile.cmd ・・・ 書き込み

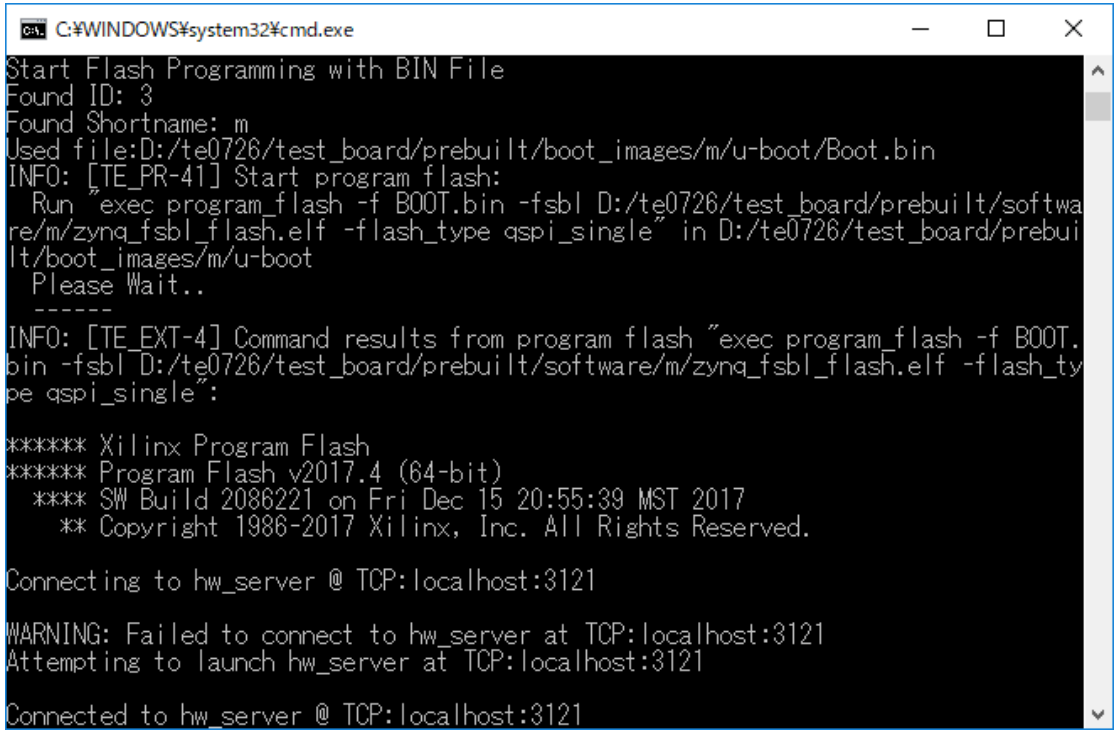
最初の書き込み

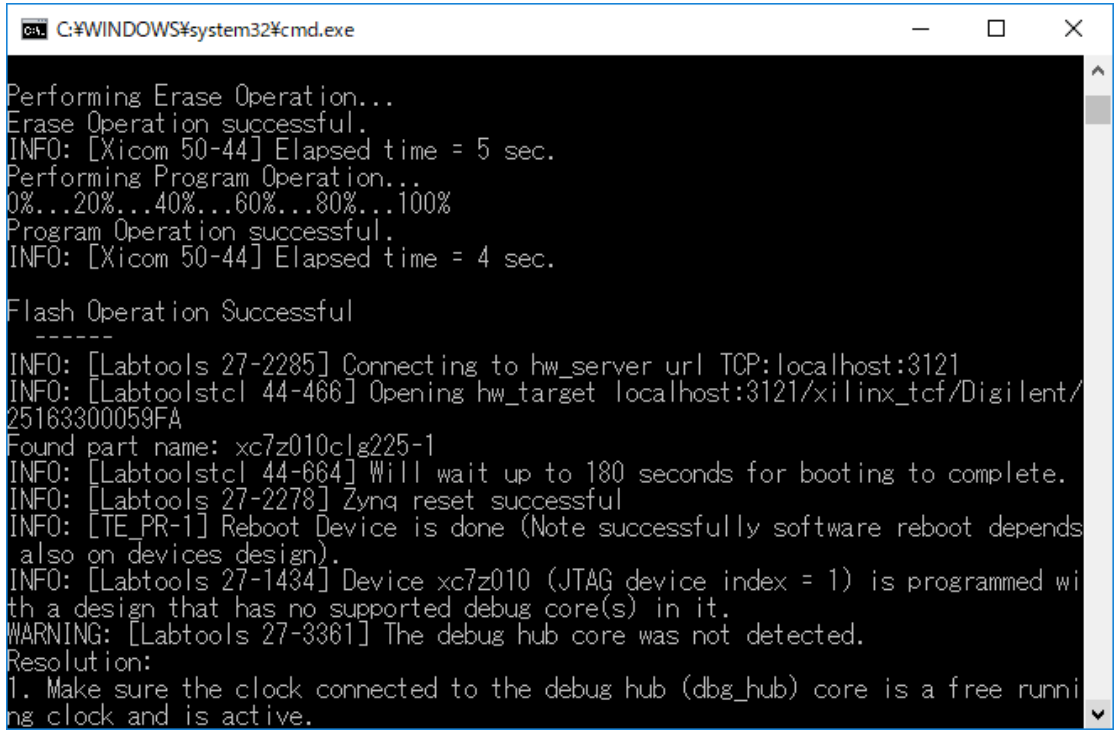
- ZynqberryはSPI ROMからしか起動できない(SDブート不可)
  - 書き込みにはTrenzスクリプトのprogram_flash_binfile.cmdを用いる
- 準備
  - design_basic_settings.cmdを開く
  - @set PARTNUMBER=3に設定する
  - @set SWAPP=NAを@set SWAPP=u-bootにする
- 書き込み
  - prebuilt/boot_images/**m**/**u-boot**/Boot.bin が書き込まれる

PARTNUMBERで決まる値　　SWAPPで決まる値

2017.4プロジェクトの場合

書き込み中のようす

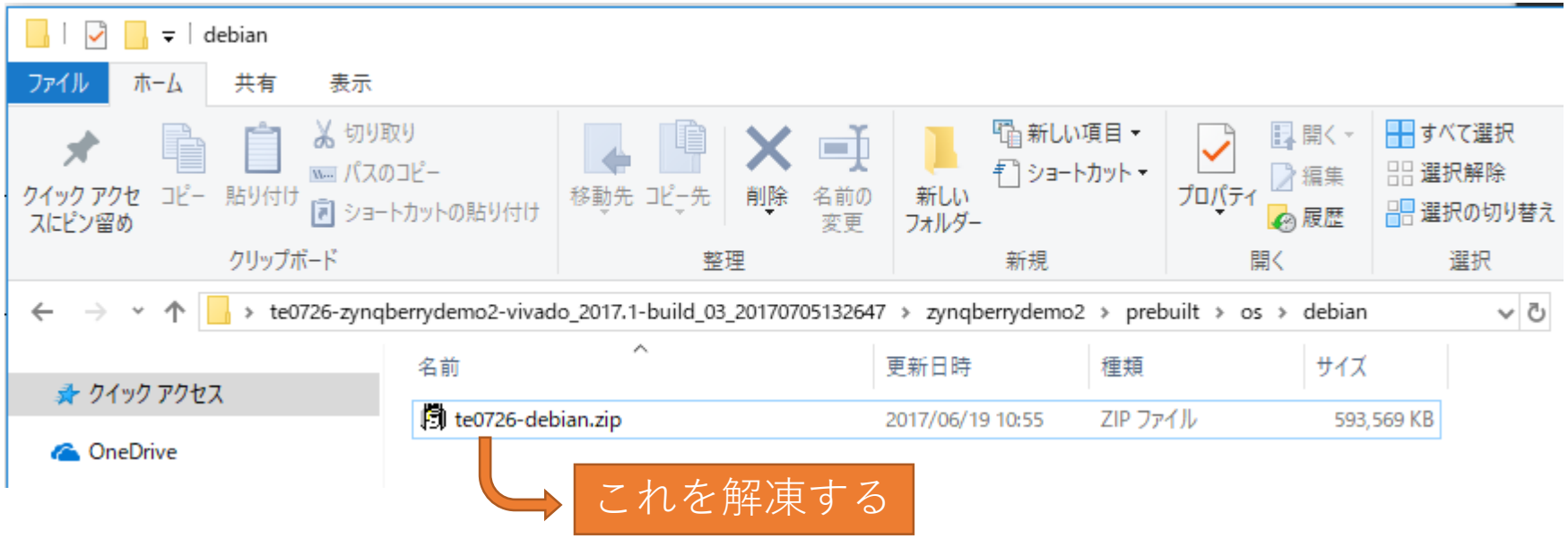




Trenz公式Debian Linux

Trenz製Debian Linuxの所在

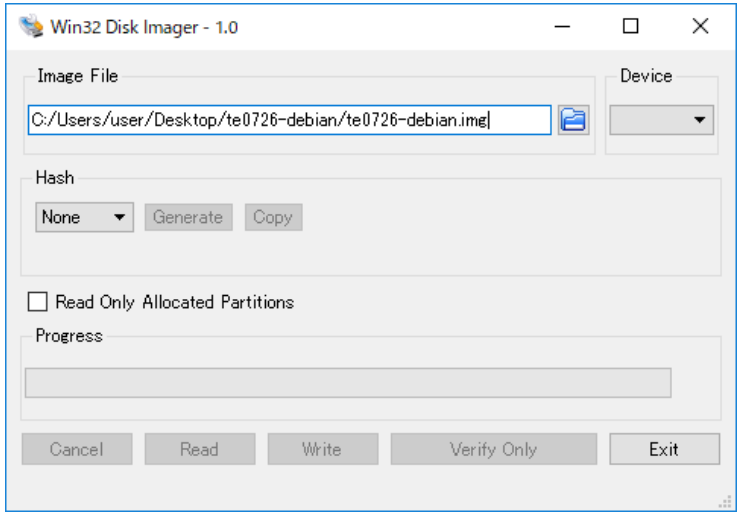
- リファレンスデザイン2017.1版のzynqberry_demo2のprebuildをダウンロード
- zynqberrydemo2¥prebuilt¥os¥debianにある

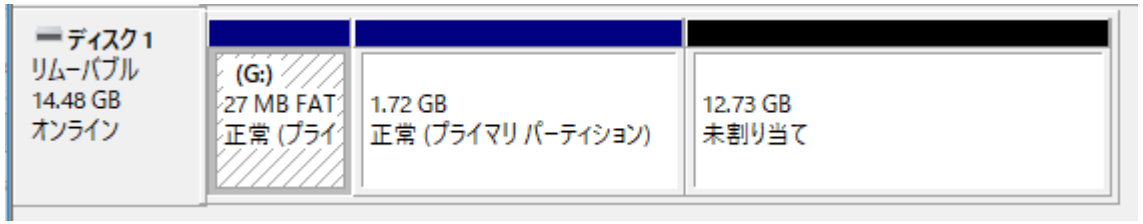


Debianのimgファイルを書き込む

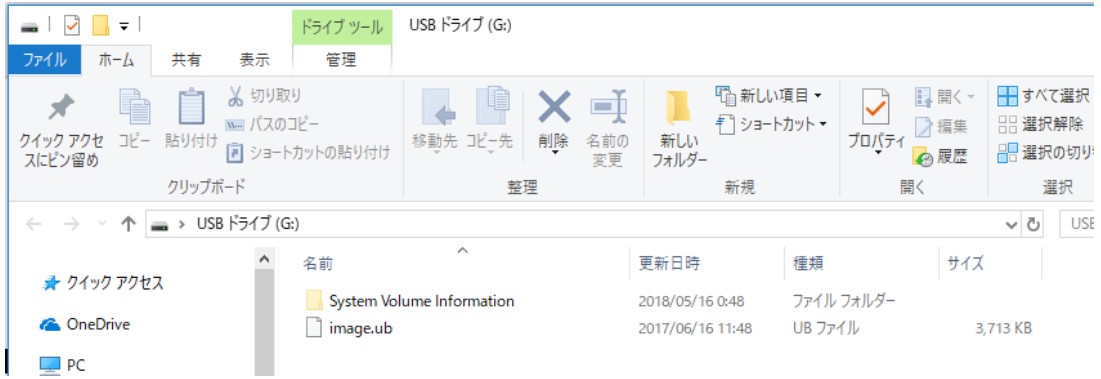
- Win32DiskImagerで書き込む

1.8GBある



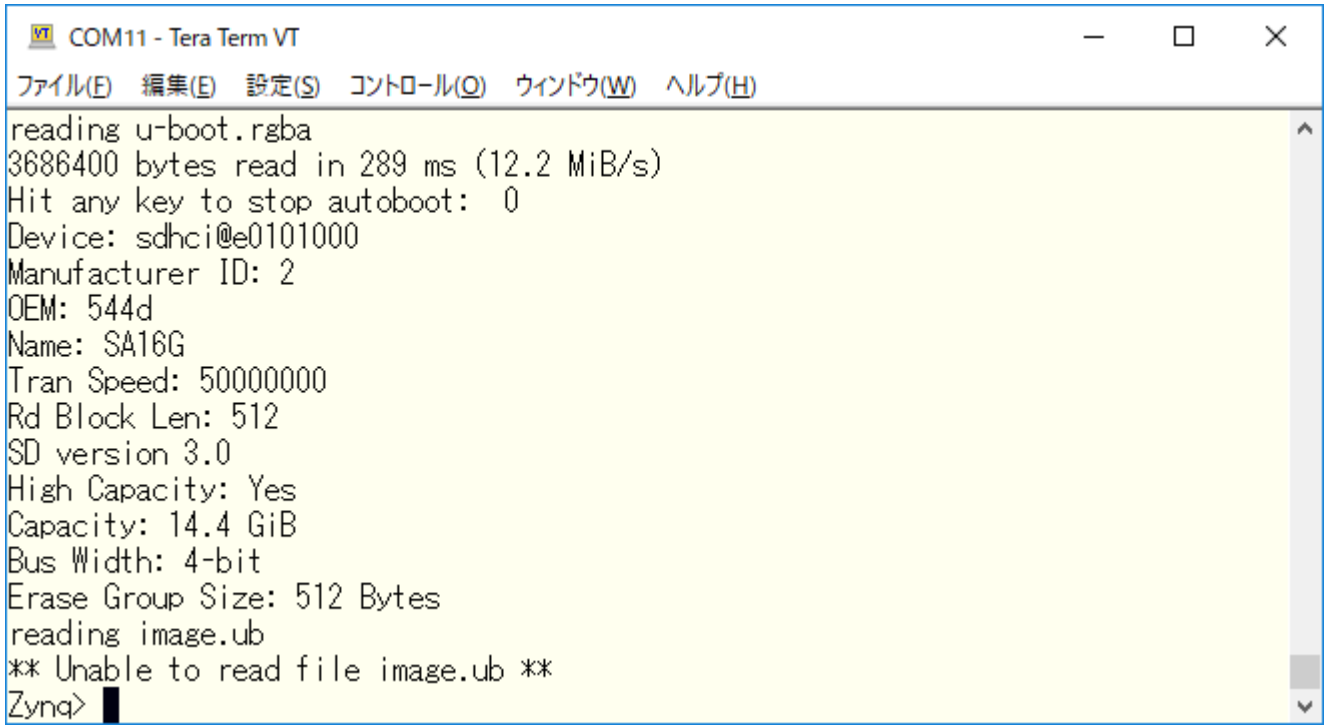


FATのパーティションには
uimage.ubのみ



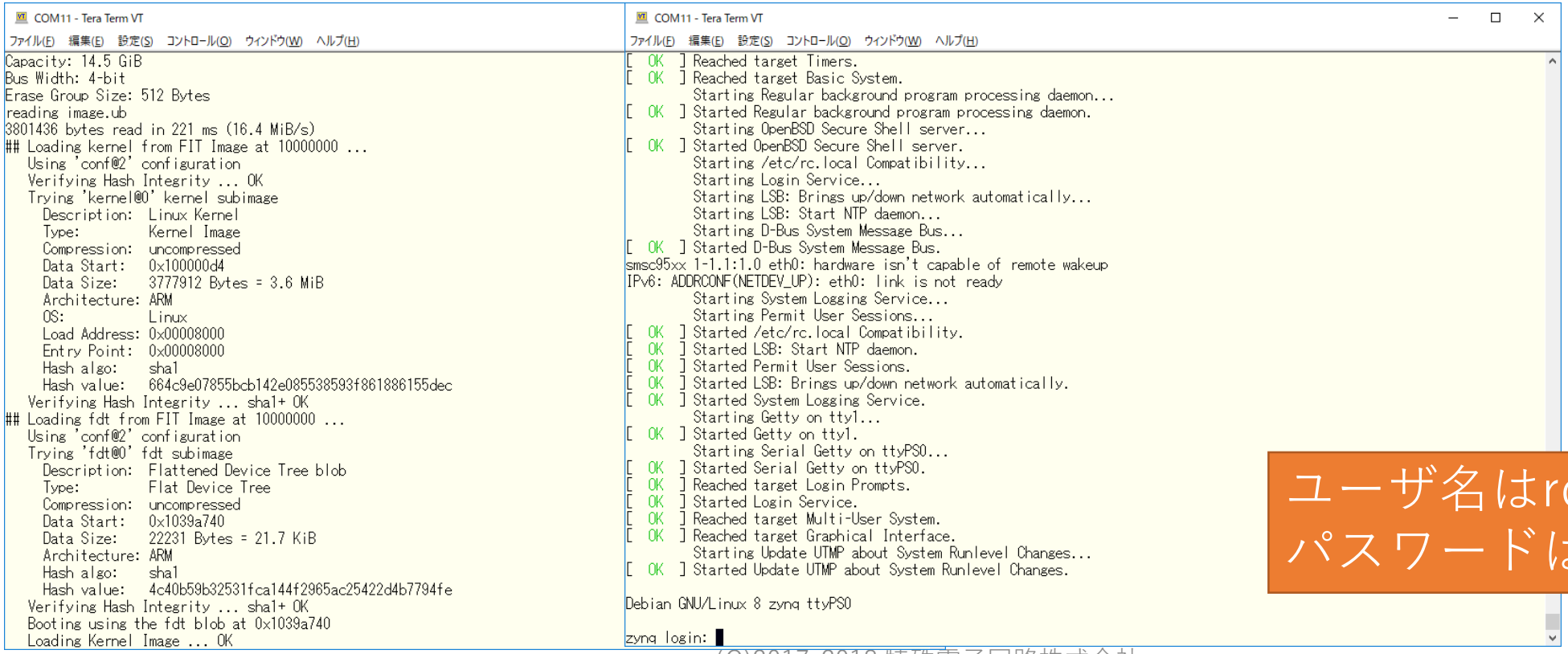
Linuxの起動1

- Trenzサンプルデザインのprebuildにあるboot.binを書き込むと、カーネルイメージ(image.ub)が要求される



Linuxの起動2

- zynqberrydemo2¥prebuilt¥os¥debianにあるdebianのイメージを書き込んだSDカードがあれば、起動する



ユーザ名はroot
パスワードはroot

デスクトップの起動

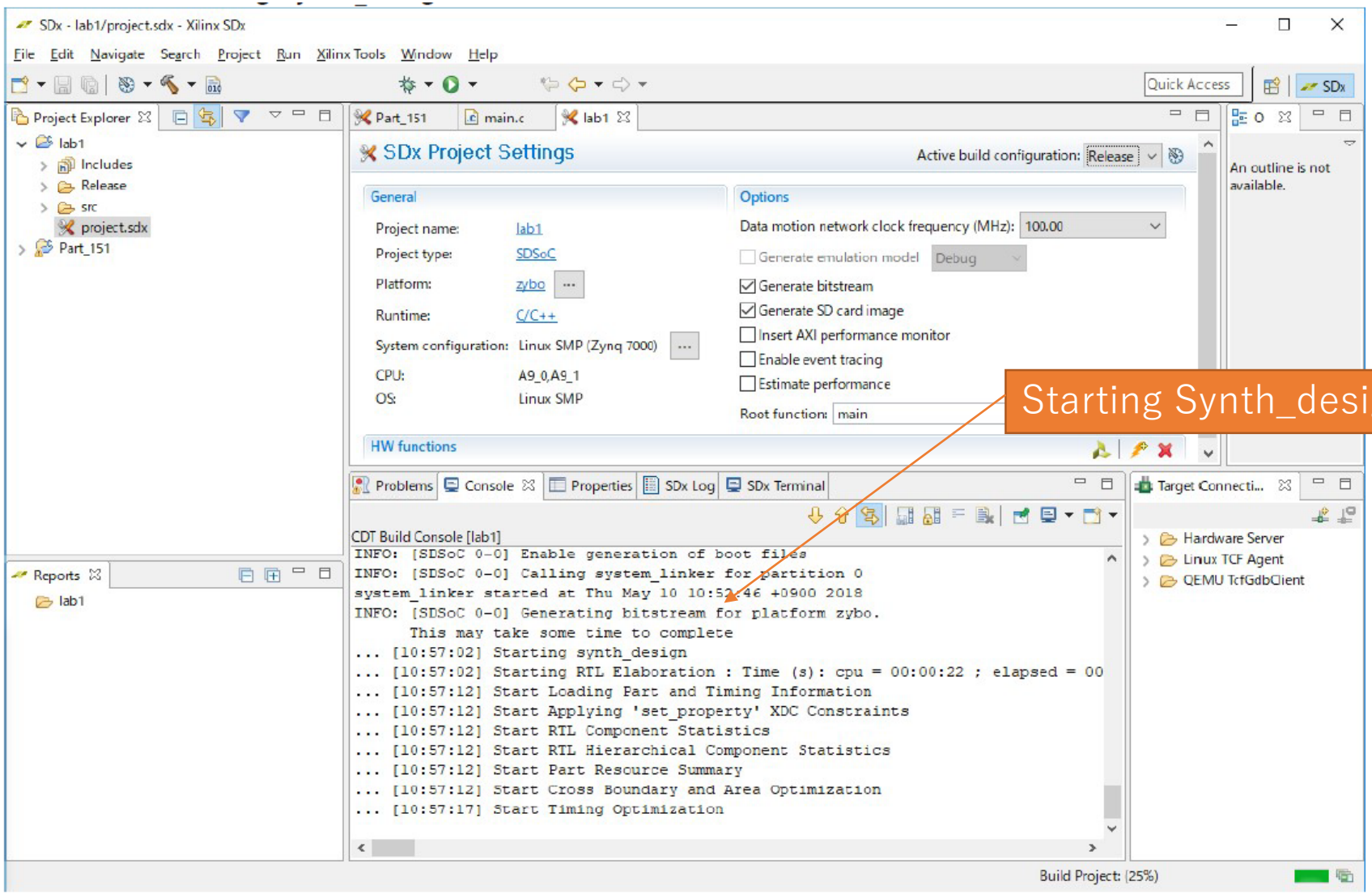
startxでデスクトップが起動

SDSoC

SDSoCについて

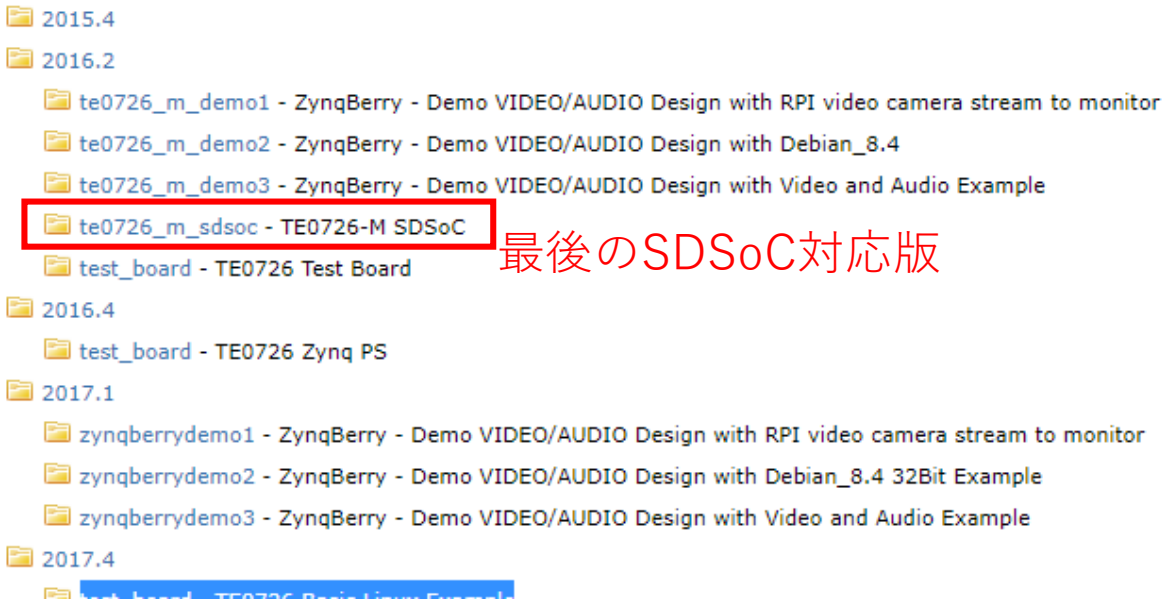
- VivadoとHLSとXSDKを統合したものがSDx (SDSoC)
- SDxをダウンロードすると両方インストールされる
- VivadoではRTLから書くが、SDxはソフトウェアを書いて、ハードウェア化したい部分を指定する
- 必ずBoard Platform ファイルが必要

SDSoCの動作画面


SDx - lab1/project.sdx - Xilinx SDx
File Edit Navigate Search Project Run Xilinx Tools Window Help
Quick Access
SDx
Project Explorer
lab1
Includes
Release
src
project.sdx
Part_151
Part_151
main.c
lab1
SDx Project Settings
Active build configuration: Release
General
Project name: lab1
Project type: SDSoC
Platform: zybo
Runtime: C/C++
System configuration: Linux SMP (Zynq 7000)
CPU: A9_0,A9_1
OS: Linux SMP
Options
Data motion network clock frequency (MHz): 100.00
Generate emulation model
Debug
Generate bitstream
Generate SD card image
Insert AXI performance monitor
Enable event tracing
Estimate performance
Root function: main
HW functions
An outline is not available.
Problems
Console
Properties
SDx Log
SDx Terminal
CDT Build Console [lab1]
INFO: [SDSoC 0-0] Enable generation of boot files
INFO: [SDSoC 0-0] Calling system_linker for partition 0
system_linker started at Thu May 10 10:52:46 +0900 2018
INFO: [SDSoC 0-0] Generating bitstream for platform zybo.
This may take some time to complete
... [10:57:02] Starting synth_design
... [10:57:02] Starting RTL Elaboration : Time (s): cpu = 00:00:22 ; elapsed = 00
... [10:57:12] Start Loading Part and Timing Information
... [10:57:12] Start Applying 'set_property' XDC Constraints
... [10:57:12] Start RTL Component Statistics
... [10:57:12] Start RTL Hierarchical Component Statistics
... [10:57:12] Start Part Resource Summary
... [10:57:12] Start Cross Boundary and Area Optimization
... [10:57:17] Start Timing Optimization
Target Connecti...
Hardware Server
Linux TCF Agent
QEMU TcfGdbClient
Reports
lab1
Build Project: (25%)
Starting Synth_designとある

ZynqberryとSDSoC

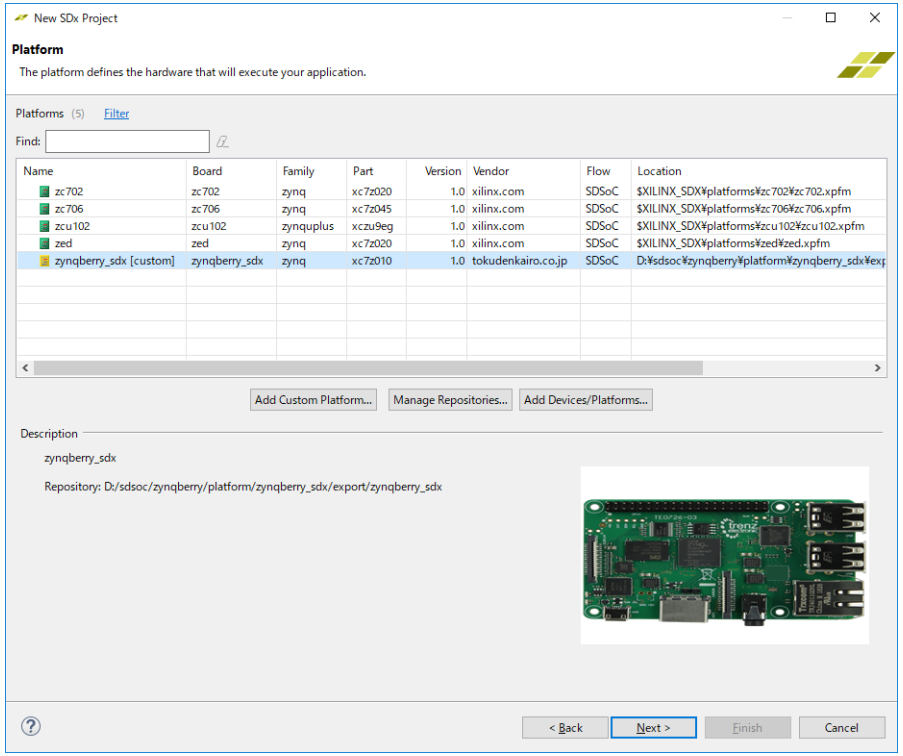
- ZynqberryのSDSoC対応は2016.2で止まっている
  - *2016.3からSDSocとSDAccelを統合したSDxに切り替わり、Xilinx的にはSDx一本でいく方針？のようなので2016.2からファイル構成や作法、必要ファイルなども大きく変わっていると思われます。*

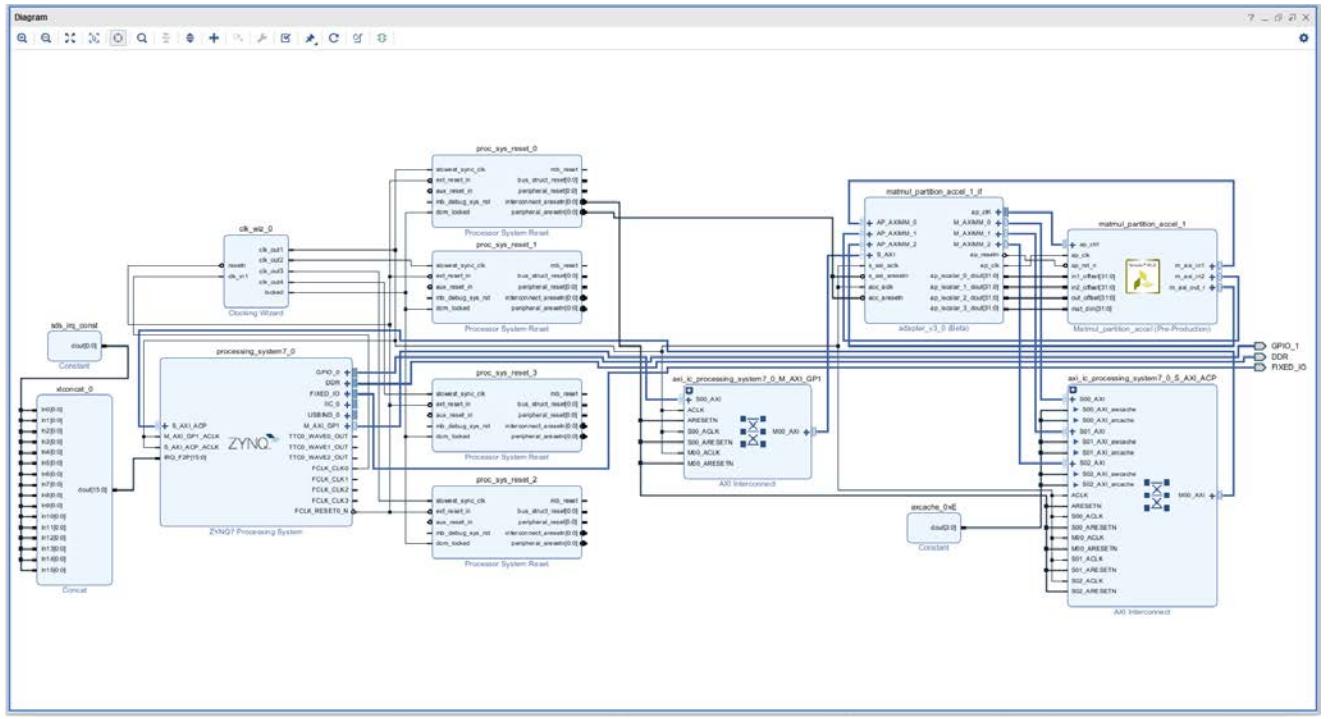


最後のSDSoC対応版

カスタムプラットフォームの作成

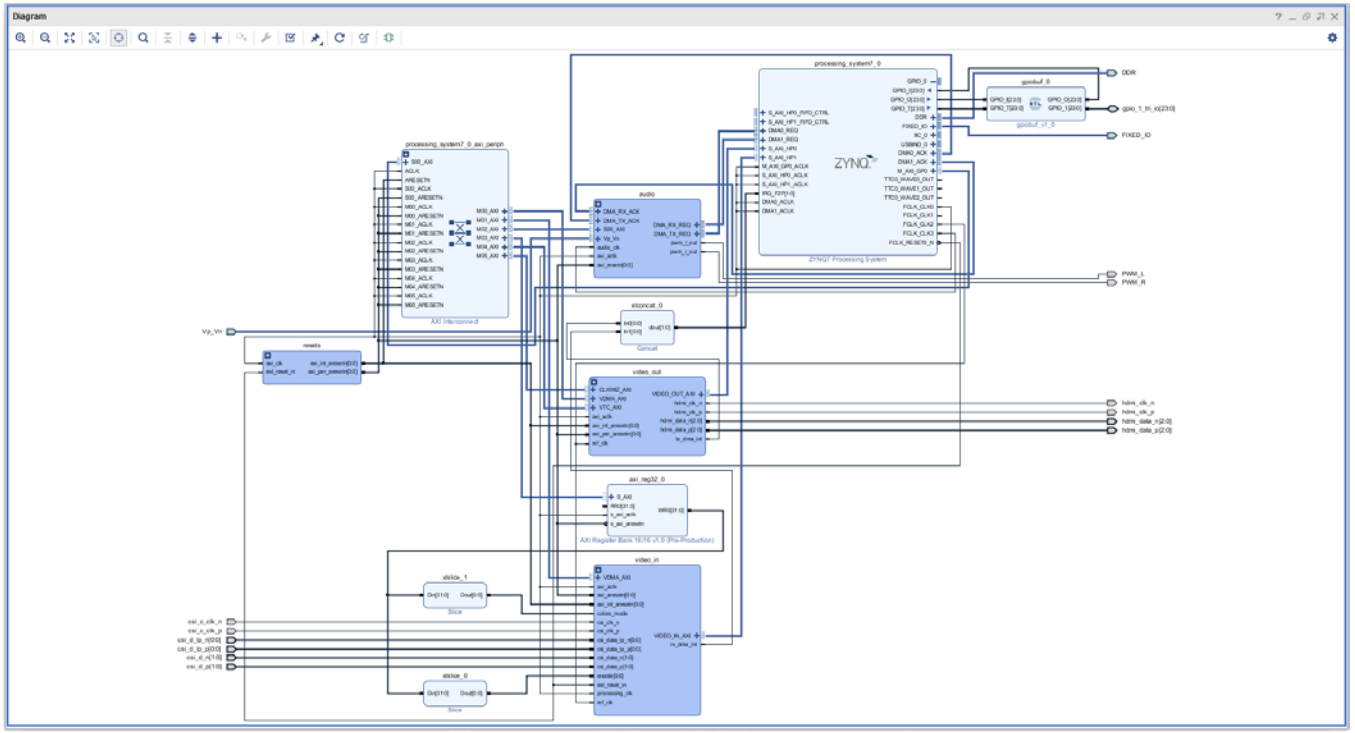
- 自分でZynqberry用のプラットフォームを作成
- 一応、C関数のハードウェア化までは確認



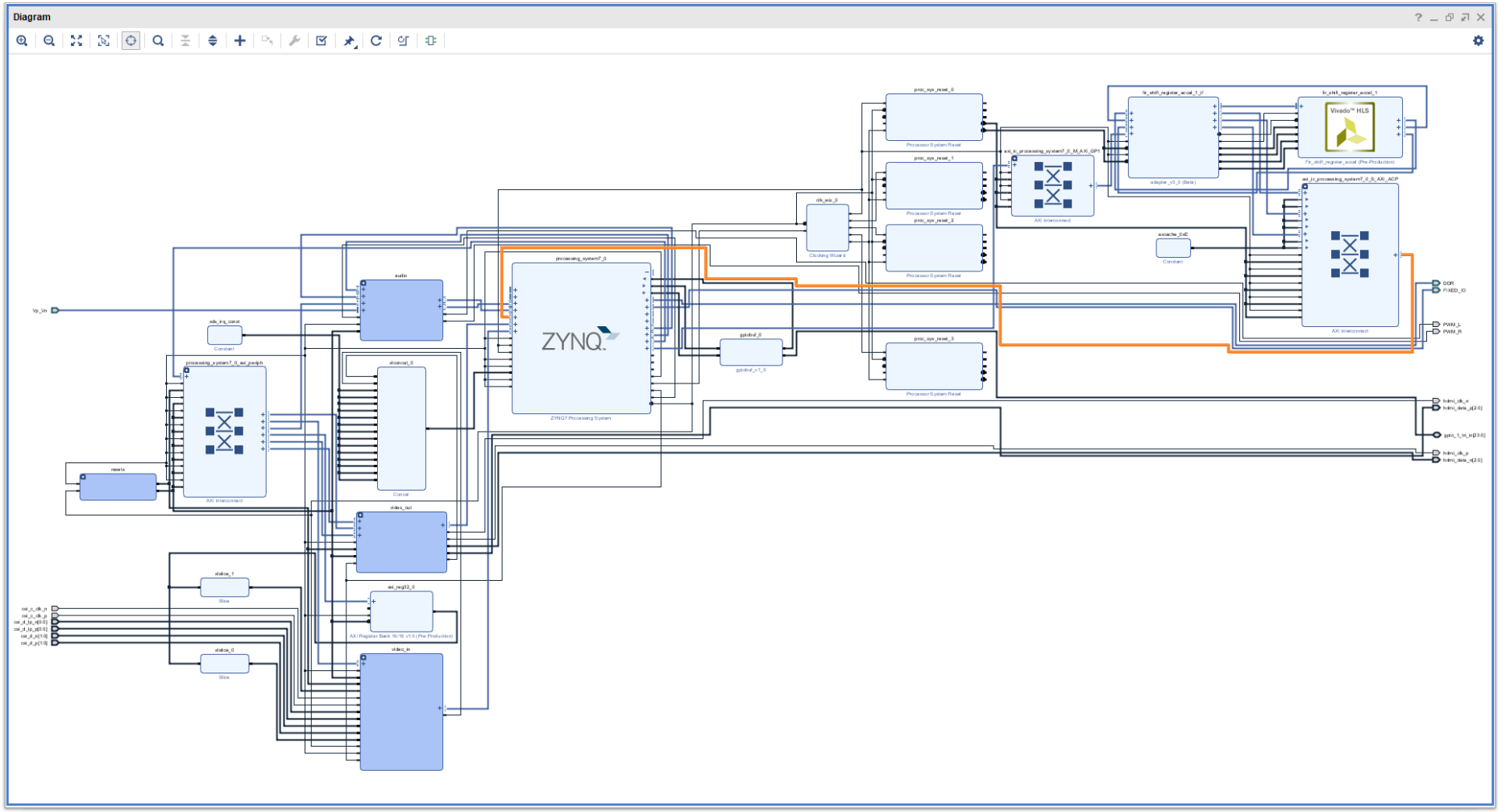


より、リッチなプラットフォームへ

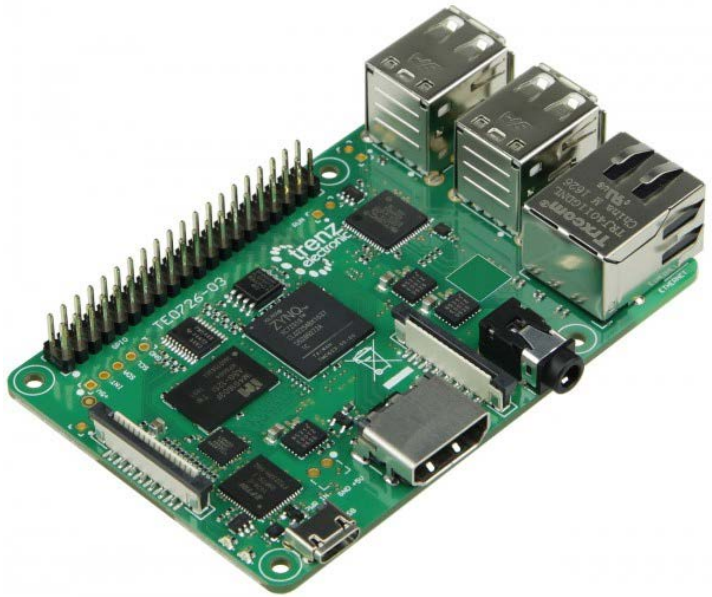
- zynqberrydemo2をベースにVivado 2018へ移行し、プラットフォームファイルを生成中



SDSoCの回路を組み込み



こういうことがやりたかったんじゃないかな？



root@zynqberry $ heavy_process

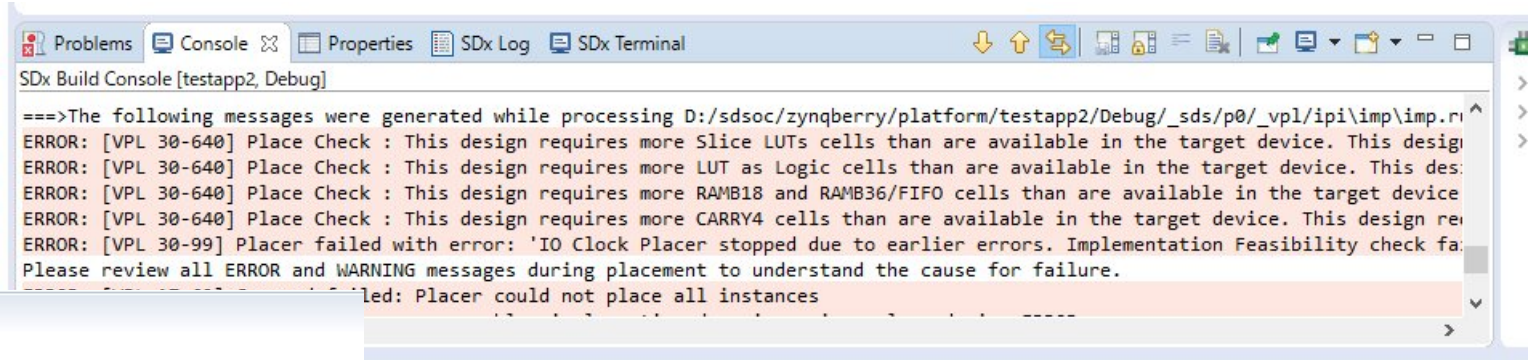
```
int main () {
        ・・・
          hw_func1();
          hw_func2();
        ・・・
}
```

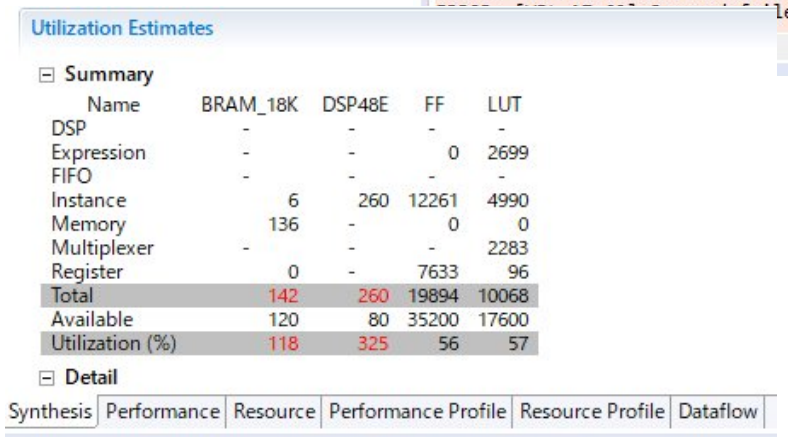
HW化された関数

FPGAをアクセラレータとして使えるRaspberry Pi互換のボード
動的にHWを再構成しながら重い処理をハードでこなす

行列計算プログラムをHW化

- SDSoCのサンプルにあるarray partition
- Zynqberry(XC7Z010)には入りきらなかった



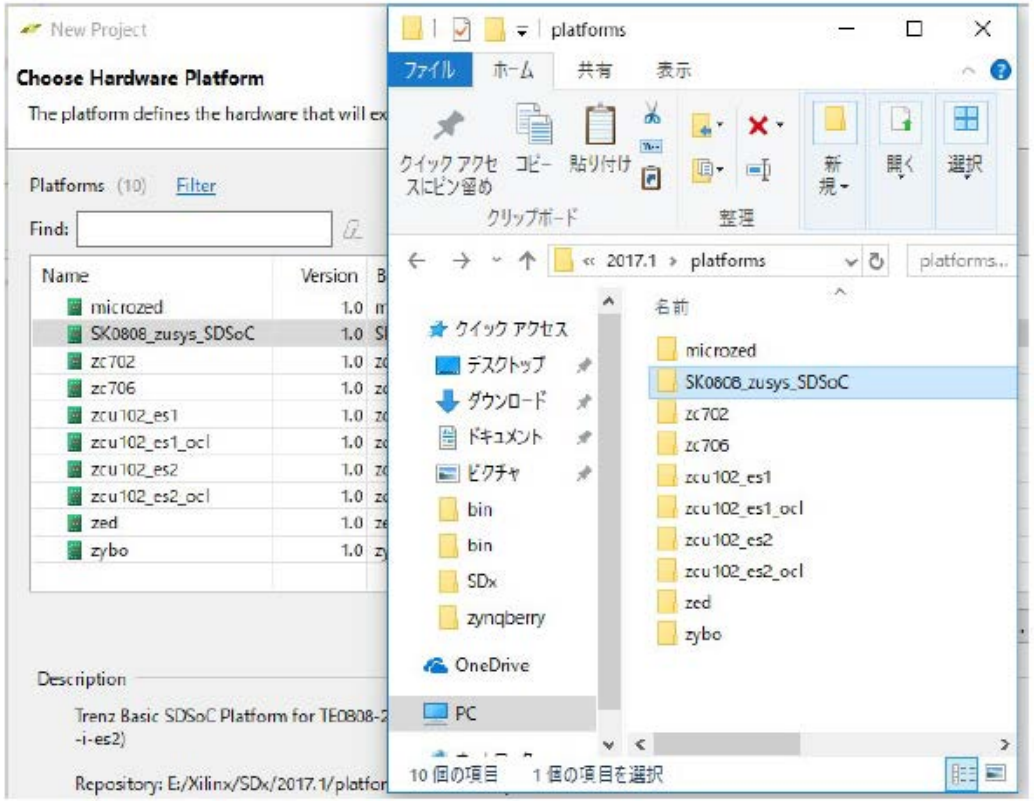


Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2699
FIFO	-	-	-	-
Instance	6	260	12261	4990
Memory	136	-	0	0
Multiplexer	-	-	-	2283
Register	0	-	7633	96
Total	142	260	19894	10068
Available	120	80	35200	17600
Utilization (%)	118	325	56	57

より大きなFPGAが必要

他のTrenzボードで試す

- TE0808 (UltraScale+ SOM)はSDx 2017.1のサンプルあり



・プロジェクトの生成がうまくいっていない？
・おそらく、plathomeの情報が足りない
・2016.3, 2016.4 , 2017.1, 2018.1で試した限りだと微妙に構成が変わりつつあってこのPlatformの記述や構成が互換がまったくなさそうある。

様々な書き込み方

① xdevcfgを使う方法

- FPGA(PL)の動的な書き換えが可能

  cat *bitstream.bit* > /dev/xdevcfg

- sambaとかでファイル共有しておいて、bitstreamを転送して、xdevcfgで書き換えると非常に楽
- /etc/rc.localの中で任意のbitstremをロードさせることが可能
- PSの設定（クロックなど）が変わる場合に対応できない

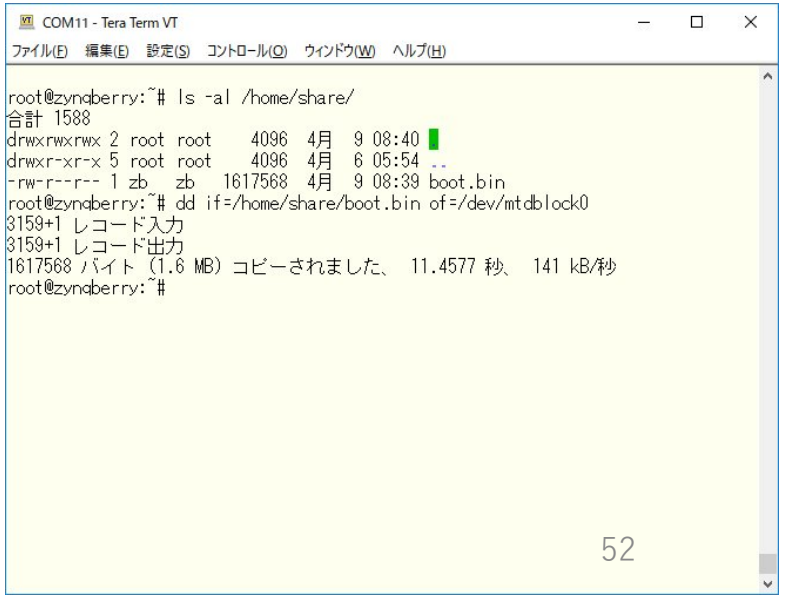
② Linux上からROMを書き換える方法

- LinuxがSPI ROMを認識していれば可能
- /dev/mtdblock0 デバイスをddで操作することで、SPI ROMの内容を生で読み書きする
- 書き込み

  dd if=/home/share/boot.bin of=/dev/mtdblock0
- 読み出し

  dd if=/dev/mtdblock0 of=spirom.bin

  hexdump -Cv spirom.bin -n 512



作成したFPGAとソフトのROM化

- PLの設定が変わるような場合はROM化しなければならない
- FSBL + bitstrem + u-boot → boot.binを生成する
- 特設サイトのファイルを使うのが一番楽

自分で作ったboot.binの書き込み方

- Vivado 2017.1~2とXSDK2017. 1~2を用意する
- 2017.1~2のTrenzプロジェクトを用意する
- design_basic_settings.cmdを以下のように設定する
  - VIVADO_VERSION=2017.2
  - PARTNUMBER=3
  - SWAPP=myboot
- 生成したboot.binをprebuilt¥boot_images¥te0726_m¥myboot に置く
- program_flash_binfile.cmdで書き込み開始

boot.binとLinuxの起動の組み合わせ

- Trenz製のboot.bin
  - petalinuxで作られている
  - prebuildに初めから入っているもの
    → Trenz製のdebian(uImage.ub)を起動する専用
    SPI ROMの扱いなどがないので、いろいろ不便

- 自主製作のboot.bin
  - これから述べる手順でオープンな手順で作ったもの
    → 普通のLinuxカーネル(uImage)を起動
    Ubuntu 14.10

まとめ

- MIOは不用意に動かしてはならない
- GPIOを動かすには /sys/class/gpio または /dev/mem
- リファレンスデザインは2017.1および2017.4。
  2017.1がリッチなデザインで、こちらを推奨
- 公式Debianは2017.1のzynqberry demo2の中にある
- スクリプトを使ってプロジェクトを生成する
- ROMの書き込みもスクリプトで行う
- SDSoCはこれから使えるようにする

課題

- zynqberry_demo2のboot.binを書き込めるようにしてください
- Linuxが起動するようにしてください
  （Trenz Debial Linuxでも、特電Ubuntuでも可）
- zynqberry_bootbin.zipの中のboot.binを書き込んでください
- GPIO2にLEDをつないで、チカチカさせてください
  - やり方は問いません