

PYNQ でお手軽に始める FPGA システム開発

2021-09-03 SWEST23

藤枝 直輝 (愛知工業大学)

ACRi

- ◆ Adaptive Computing Research Initiative
 - FPGA に代表されるアダプティブなハードウェアデバイスの活用方法を模索・研究する団体
 - 要するに、FPGA を盛り上げる会
 - 5大学, 17社が参加 (2021/7 現在)
- ◆ 本セッションは SWEST/ACRi の共同企画です



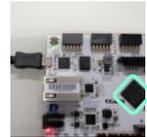
<https://www.acri.c.titech.ac.jp/>

アウトライン

- ◆ PYNQ とは何か? 何が嬉しいか?
- ◆ PYNQ-Z1 ボードのセットアップ
- ◆ LED 点滅回路を含む設計例
- ◆ グラフィックパターン送信回路を含む設計例

FPGA 使ってますか?

- ◆ FPGA = 論理が書換え可能なデジタルLSI
 - 開発コスト・期間の削減
 - 製造のインisialコスト低減
 - ハードのプロセッサコアと FPGA とがワンチップ化された製品も存在
 - ◆ Xilinx 社 Zynq シリーズなど
- ◆ FPGA のハード開発
 - ハードウェア記述言語 (HDL) がよく使われる
 - 最近では C/C++ で開発 (高位合成) することも



ありがちな? パターン

- ◆ FPGA の評価ボードを買ってみる
- ◆ LED をチカチカさせてみる

.....

◆ その先が長すぎて挫折する

- そもそも何をしたいかわからない
- 作った回路を周辺回路にパッケージするには.....
- プロセッサを含んだシステムを合成するには.....
- プロセッサ側で制御プログラムを動かすには.....
- プロセッサ側で OS を走らせるには.....

```

module led_disp (
  input logic clk,
  input logic rst,
  output logic led);
  logic [25:0] count;
  assign led = count[25];
  always_ff # (posedge clk) begin
    if (rst) begin
      count <= 26'h0;
    end else begin
      count <= count + 1'b1;
    end
  end
endmodule

```

PYNQ

- ◆ Python の生産性を Zynq に
 - Xilinx 社のオープンソースプロジェクト
 - 各種ライブラリ (ハードウェア・ソフトウェア), Python 開発環境, 事前に用意されたディスクイメージ等
- ◆ PYNQ のねらい
 - 「組込みアーキテクト・エンジニア・デザイナーが ASIC スタイルの設計ツールを使うことなく Zynq を利用できるようにする」*



* PYNQ Introduction, <https://pynq.readthedocs.io/en/v2.6.1/>, retrieved 2021-07-08.
画像出典: PYNQ: Python Productivity, <http://www.pynq.io>, retrieved 2021-07-08.

PYNQ の利点 (1)

◆ 論理の書換えの容易さ

■ オーバーレイ: FPGA 部分の情報一式

- ◆ .bit ファイル: 回路情報
- ◆ .hwh ファイル: ハードウェア構成



- Python で Overlay クラスのインスタンスを作成すると、対応する回路が FPGA 部分に自動的に書き込まれる
- ハードウェア構成ファイルを読み取り、対応するドライバクラスを自動的にインスタンス化



SWEST/ACRI 共同企画

2021/9/3

7

PYNQ の利点 (2)

◆ プログラミングの容易さ

■ Python でプログラムが記述できる

- ◆ 記述の抽象度の高さ
- ◆ 豊富なソフトウェアライブラリ
- ハードウェアへのアクセスも容易
 - ◆ ドライバクラスの read, write メソッドで MMIO にアクセス
 - ◆ カスタムのドライバクラスの作成も可能

```
from pynq import Overlay
import time

pl = Overlay("knight_pynq.bit")
gpio = pl.out_gpio_0
for i in range(3):
    gpio.write(0, 100000000); time.sleep(3)
    gpio.write(0, 200000000); time.sleep(3)
    gpio.write(0, 0xffffffff)
```

PYNQ における LED 点滅回路の制御プログラム例

SWEST/ACRI 共同企画

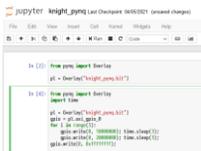
2021/9/3

8

PYNQ の利点 (3)

◆ ボードとの接続の容易さ

- Ethernet ケーブルでボードと接続
- Python のプログラミングは Jupyter Notebook で
 - ◆ Web ブラウザからアクセス
- ファイルのアップロードもお好みの方法で
 - ◆ SCP, SMB, Jupyter Notebook のアップロード機能...



SWEST/ACRI 共同企画

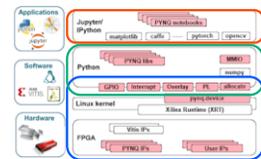
2021/9/3

9

つまり PYNQ では……

◆ 自作回路をプロセッサの周辺回路として含んだ FPGA システムを、お手軽に開発できる!

- ハードウェアオーバーレイと関連 API
- Python による開発環境・ライブラリ
- Web ベースの技術を活用した容易なボード利用



画像出典: PYNQ: Python Productivity, <http://www.pynq.io>, retrieved 2021-07-08.

SWEST/ACRI 共同企画

2021/9/3

10

PYNQ 対応ボード

◆ Zynq ベースの各種ボード、または Alveo

- ディスクイメージが用意されているボードを使うのが楽

◆ 本セッションの対象

- PYNQ-Z1 (Digilent社)
- PYNQ-Z2 (TUL社)でも OK
 - ◆ I/O の種類や配置に違いはあるが、スペックはほとんど同じ
 - ◆ PYNQ-Z2 の方が少し安い

Downloadable PYNQ images

If you have a Zynq board, you need a PYNQ 5 image from the table below. If an image is in **Orange** it is legacy.

Board	SD card image	E
PYNQ-Z2	v2.8	F
PYNQ-Z1	v2.8	F
ZCU104	v2.8	F
RFSoc 2x2	v2.8	F
ZCU111	v2.8	F
Ultra902	v2.8	F
Ultra96 (legacy)	v2.8	S
ty60M-3-ZUF6V	v2.5	C
ty60M-3A-ZU19EG	v2.5	C

 ローエンド(数万円程度)
 ミドルレンジ(数十万円程度)
 ハイエンド(それ以上)

画像出典: PYNQ: Python Productivity - Board, <http://www.pynq.io/board.html>, retrieved 2021-07-08.

SWEST/ACRI 共同企画

2021/9/3

11

アウトライン

◆ PYNQ とは何か? 何が嬉しいか?

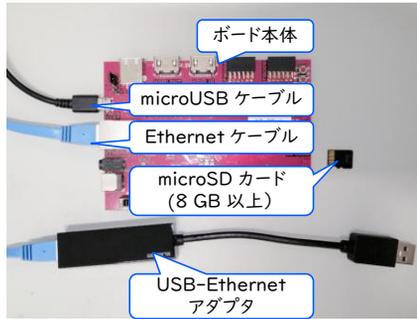
- ◆ PYNQ-Z1 ボードのセットアップ
- ◆ LED 点滅回路を含む設計例
- ◆ グラフィックパターン送信回路を含む設計例

SWEST/ACRI 共同企画

2021/9/3

12

機材を準備する



SWEST/ACRI 共同企画

2021/9/3

13

機材の準備 (補足)

- ◆ USB-Ethernet アダプタを使うと便利
 - ボードをホスト PC とだけ接続できる
- ◆ 単に既存の有線 LAN に接続しても OK
 - この場合はアダプタは不要
 - DHCP で IP アドレスが設定される
- ◆ セキュリティ、有線 LAN の物理的配線などを考慮して、お好みの方で

SWEST/ACRI 共同企画

2021/9/3

14

イメージファイルをダウンロード

- ◆ ボードに対応した SD カードイメージを <http://www.pynq.io/board.html> からダウンロード

- 圧縮時 2 GB 弱、展開後 6 GB 程度あるので注意

- 展開して pynq_z1_v2.6.0.img を取り出しておく

- ◆ v以降の数字は今後のバージョンアップで変わる可能性あり
- ◆ Windows 上で開こうとすると「イメージが壊れています」と言われるが、それで正常

Downloadable PYNQ images

If you have a Zynq board, you need a PYNQ C image from the table below. If an image is no

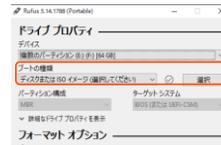
Board	SD card image	C
PYNQ-Z2	v2.6	F
PYNQ-Z1	v2.6	F
ZCU104	v2.6	F
RFSoc zyx	v2.6	F
ZCU111	v2.6	F
Ultra96V2	v2.6	F
Ultra96 (legacy)	v2.6	S
TySOM-3-ZUPEV	v2.5	C
TySOM-3A-ZU19EG	v2.5	C

画像出典: PYNQ: Python Productivity - Board, <http://www.pynq.io/board.html>, retrieved 2021-07-08.
SWEST/ACRI 共同企画 2021/9/3

15

イメージファイルを書き込み

- ◆ Windows なら Rufus などのツールを使って、イメージファイルを microSD に書き込む
 - ブートの種類 → 選択で先ほどのイメージを選択
 - ◆ **くれぐれも書き込み先デバイスを間違えないように!**
 - 書き込み後は2つのパーティションが作成される
 - ◆ 後者は Windows から認識できないが、それで正常



SWEST/ACRI 共同企画

2021/9/3

16

ボードの接続を確かめる



※ ボード左上側のジャンパは、PYNQ-Z2 ではボード右側、microSD カードの差込口は、PYNQ-Z2 ではボード下側。

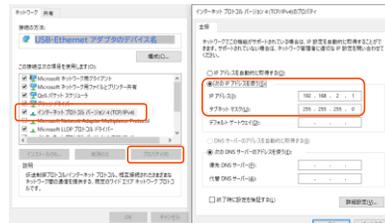
SWEST/ACRI 共同企画

2021/9/3

17

USB-Ethernet アダプタの設定

- ◆ 設定 → ネットワークとインターネット → イーサネット → アダプタのオプションを変更する
 - アダプタのプロパティを以下の通り設定



SWEST/ACRI 共同企画

2021/9/3

18

ボードの電源を入れる

- ◆ ボードの電源スイッチを ON 側に
 - 1分ほどすると LED が数回点滅したあと、緑色の4つの LED が点灯した状態になる
 - Tera Term などで仮想シリアルポートにアクセス (115,200 bps) すると、シェルの画面が表示される



SWEST/ACRI 共同企画

2021/9/3

19

Jupyter Notebook にログイン

- ◆ ブラウザで **192.168.2.99** にアクセスする
 - 既存の LAN に接続した場合は、ボードに割当てられた IP アドレスを確認して、そのアドレスにアクセスする
 - Jupyter Notebook のログイン画面が表示されるので、パスワードに **xilinx** と入力してログインする



SWEST/ACRI 共同企画

2021/9/3

20

アウトライン

- ◆ PYNQ とは何か? 何が嬉しいか?
- ◆ PYNQ-Z1 ボードのセットアップ
- ◆ LED 点滅回路を含む設計例
- ◆ グラフィックパターン送信回路を含む設計例

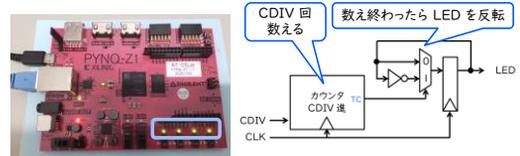
SWEST/ACRI 共同企画

2021/9/3

21

LED 点滅回路

- ◆ PYNQ の緑色 LED (4個) を、一定時間おきに点滅させる
 - 点滅周期 CDIV は入力として与える



SWEST/ACRI 共同企画

2021/9/3

22

LED 点滅回路のソースコード

```

module blink_pynq (
    input wire CLK, RESETN,
    input wire [31:0] CDIV,
    output reg [3:0] LED;
    reg [31:0] count, n_count;
    reg [3:0] n_led;

    always @ (CDIV, count, LED) begin
        if (count == CDIV) begin
            n_count = 1;
            n_led = ~LED;
        end else begin
            n_count = count + 1'b1;
            n_led = LED;
        end
    end

    always @ (posedge CLK) begin
        if (~RESETN) begin
            count <= 0;
            LED <= 4'b1001;
        end else begin
            count <= n_count;
            LED <= n_led;
        end
    end
endmodule
    
```

SWEST/ACRI 共同企画

2021/9/3

23

LED 点滅回路の制約ファイル

- ◆ 自作回路が外部入出力をもつ場合
 - 入出力信号の FPGA のピンへの割当ての情報が必要
 - 制約ファイル (拡張子 .xdc) を使う
- ◆ LED 点滅回路では
 - 回路の LED 出力を PYNQ-Z1 の LED 出カピン (右から R14, P14, N16, M14) へ割当ててる

```

blink.xdc
set_property -dict { PACKAGE_PIN R14 IOSTANDARD LVCMOS33 } [get_ports { LED[0] }];
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { LED[1] }];
set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVCMOS33 } [get_ports { LED[2] }];
set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { LED[3] }];
    
```

SWEST/ACRI 共同企画

2021/9/3

24

Vivado の準備

- ◆ PYNQ のバージョンに対応した Vivado を用意
 - 既存の周辺回路 (IP コア) のバージョンを合わせるため
 - PYNQ 2.6 に対応した Vivado: 2020.1
- ◆ ボードファイルを事前にインストール

https://pynq.readthedocs.io/en/latest/overlay_design_methodology/board_settings.html

Vivado board files

Vivado board files contain the configuration for a board that is required when creating a new project in Vivado. For most of the Xilinx boards (for example, ZCU104), the board files have already been included in Vivado; users can simply choose the corresponding board when they create a new project. For some other boards (for example, Pynq-Z1 and Pynq-Z2), the corresponding board files can be downloaded as shown below.

- Download the Pynq-Z1 board files
- Download the Pynq-Z2 board files

SWEST/ACRI 共同企画

2021/9/3

25

新規プロジェクトの作成 (1)

- ◆ File → Project → New
 - ウィザードが表示されるので、まず Next
 - 2つ目の画面でプロジェクト名と作業ディレクトリを指定
 - ◆ create project subdirectory にチェックすると、プロジェクト名と同じ名前のフォルダが自動的に作成される



SWEST/ACRI 共同企画

2021/9/3

26

新規プロジェクトの作成 (2)

- ◆ プロジェクト種別: RTL Project
 - その下の Do not specify~ にチェックしておく
- ◆ パーツ選択: Boards → PYNQ-Z1
 - 表示されない場合は以下をチェック
 - ◆ ボードファイルが所定の位置にコピーされているか
 - ◆ Vivado インストール時に Zynq-7000 をチェックしたか
- ◆ 最終確認画面で Finish



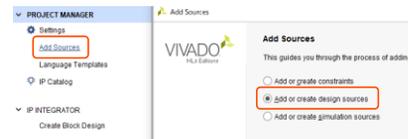
SWEST/ACRI 共同企画

2021/9/3

27

プロジェクトへのファイルの追加 (1)

- ◆ LED 点滅回路のソースをプロジェクトに追加
 - Project Manager → Add Sources でウィザードが表示される
 - Add or create design sources



SWEST/ACRI 共同企画

2021/9/3

28

プロジェクトへのファイルの追加 (2)

- ◆ LED 点滅回路のソースをプロジェクトに追加
 - Add Files ボタンでソースコードのファイルを指定する
 - ファイルがリストに追加されたことを確認して Finish
- ◆ 同じ要領で制約ファイルも追加
 - ただし、Add or create constraints を使う



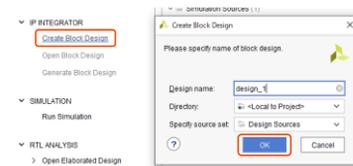
SWEST/ACRI 共同企画

2021/9/3

29

ブロック図の新規作成

- ◆ 点滅回路を含んだシステムのブロック図を作成
 - IP Integrator → Create Block Design
 - ダイアログが出るのでそのまま OK を押す



SWEST/ACRI 共同企画

2021/9/3

30

回路ブロックの追加 (1)

- ◆ プロセッサ部をブロック図に追加
 - ブロック図の + ボタンを押す
 - ZYNQ7 Processing System を探し、ダブルクリック
 - ◆ ログに警告が表示されるが、無視して構わない
 - Run Block Automation で初期設定



SWEST/ACRI 共同企画 2021/9/3 31

回路ブロックの追加 (2)

- ◆ 汎用 I/O (点滅周期用) をブロック図に追加
 - + ボタン → AXI GPIO を探してダブルクリック
 - 追加された AXI GPIO のブロックをダブルクリック
 - IP Configuration → All Outputs をチェック



SWEST/ACRI 共同企画 2021/9/3 32

回路ブロックの追加 (3)

- ◆ 汎用 I/O (点滅周期用) をプロセッサに接続
 - Run Connection Automation をクリック
 - axi_gpio_0 の S_AXI にだけチェックを入れて OK
- ◆ この段階でのブロック図の例: 右下



SWEST/ACRI 共同企画 2021/9/3 33

回路ブロックの追加 (4)

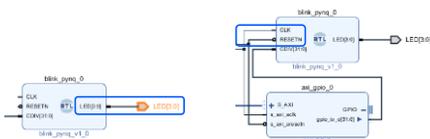
- ◆ 点滅回路をブロック図に追加・接続
 - ブロック図の何も無い場所を右クリックし、Add Module
 - blink_pynq を選択して、OK
 - AXI GPIO の gpio_io_0 出力を, blink_pynq の CDIV 入力へとドラッグして接続する
 - ◆ gpio_io_0 は GPIO の横の + をクリックすると表示される



SWEST/ACRI 共同企画 2021/9/3 34

回路ブロックの追加 (5)

- ◆ 点滅回路をブロック図に接続
 - blink_pynq の LED 出力を右クリックし、Create Port
 - ダイアログが表示されるので、デフォルトのまま OK
 - ◆ LED が出力ポートとして外に引き出される (図左側)
 - Run Connection Automation を実施
 - ◆ クロック、リセットが接続される (図右側)



SWEST/ACRI 共同企画 2021/9/3 35

ハードウェアの作成 (1)

- ◆ ブロック図の検証
 - ブロック図の何も無いところを右クリックし、Validate Design
 - Validation Successful と表示されることを確認



SWEST/ACRI 共同企画 2021/9/3 36

ハードウェアの作成 (2)

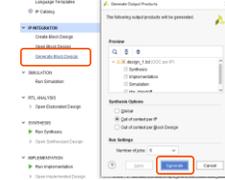
- ◆ ブロック図の HDL によるラップを作成
 - Sources → design_1 を右クリックし, Create HDL Wrapper
 - 「Let Vivado manage～」を選択して OK
 - design_1_wrapper というモジュールが作成され, 太字で表示されていることを確認



SWEST/ACRI 共同企画 2021/9/3 37

ハードウェアの作成 (3)

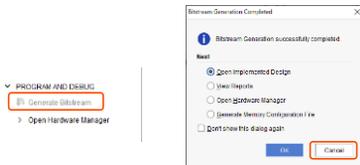
- ◆ ブロック図からの各種ファイルの生成
 - IP Integrator → Generate Block Design
 - ダイアログの設定は変更せず Generate ボタン
 - 1~2分待つとファイルの生成・合成が完了し, Vivado 画面右上に Ready と表示される



SWEST/ACRI 共同企画 2021/9/3 38

ハードウェアの作成 (4)

- ◆ ハードウェアの合成
 - Program and Debug → Generate Bitstream
 - 数分待つとハードウェアの合成に成功した旨が表示されるので, Cancel でダイアログを閉じる
 - ◆ 以降の作業は Vivado 上では行わないので



SWEST/ACRI 共同企画 2021/9/3 39

PYNQ にアップロード (1)

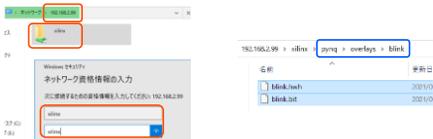
- ◆ オーバーレイに必要なファイルをコピー
 - プロジェクト名.runs¥impl_1¥design_1_wrapper.bit
 - プロジェクト名.srcs¥sources_1¥bd¥design_1_hw_handoff.design_1.hwh
 - それぞれ適当なディレクトリにコピーしてから, blink.bit, blink.hwh に名前を変更しておく



SWEST/ACRI 共同企画 2021/9/3 40

PYNQ にアップロード (2)

- ◆ オーバーレイを PYNQ にアップロード
 - エクスプローラで「¥¥192.168.2.99」にアクセス
 - xilinx をダブルクリック
 - ユーザ名・パスワードに xilinx と入力してログイン
 - pynq¥overlays ディレクトリに blink フォルダを作り, そこに先ほどの2つのファイルをコピー



SWEST/ACRI 共同企画 2021/9/3 41

PYNQ にアップロード: 参考

- ◆ Windows の設定やバージョンによっては, 事前にログイン情報の設定が必要
 - コントロールパネル → 資格情報マネージャ → Windows 資格情報の追加 から追加する



SWEST/ACRI 共同企画 2021/9/3 42

Python で動作確認 (1)

- ◆ Jupyter Notebook 上でスクリプトを作成
 - New → Python 3
 - ひとまず Overlay クラスのインスタンスを作成するだけのスクリプトを記述・実行してみる
 - LED がうっすら点灯する(しているように見える)
 - ◆ 実際には超高速で点滅している



SWEST/ACRi 共同企画

2021/9/3

43

Python で動作確認 (2)

- ◆ 点滅周期 CDIV を制御する
 - AXI GPIO に書き込みを行ってから一定時間待つ、という動作を行うスクリプトを記述・実行してみる



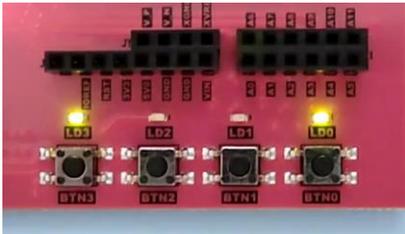
SWEST/ACRi 共同企画

2021/9/3

44

Python で動作確認 (3)

- ◆ 点滅周期 CDIV を制御する
 - 実行結果 (動画)



SWEST/ACRi 共同企画

2021/9/3

45

参考情報

- ◆ ここまでの内容 (PYNQ-Z1 のセットアップ, LED 点滅回路を含む設計例) と類似の例を, ACRi ブログの以下の記事で説明している
 - <https://www.acri.c.titech.ac.jp/wordpress/archives/11224>

実行回路を PYNQ につなぐ簡単な方法 (1)



実行回路を PYNQ につなぐ簡単な方法 (2)

実行回路を PYNQ につなぐ簡単な方法 (3)

- ◆ ソースコードは GitHub にて公開中

https://github.com/nfproc/connect_with_pynq/

SWEST/ACRi 共同企画

2021/9/3

46

アウトライン

- ◆ PYNQ とは何か? 何が嬉しいか?
- ◆ PYNQ-Z1 ボードのセットアップ
- ◆ LED 点滅回路を含む設計例
- ◆ グラフィックパターン送信回路を含む設計例

SWEST/ACRi 共同企画

2021/9/3

47

結局 HDL なの?

- ◆ HDL が書けるに越したことはない
 - クロックサイクル単位での細かい回路の動作の記述は, HDL でないと難しい
- ◆ C/C++ からの高位合成でも, 最近は結構なんとかなる
 - 開発期間を格段に短くできる
 - HDL での設計前のプロトタイピングに使う手も
 - 書き方に多少のノウハウは必要

SWEST/ACRi 共同企画

2021/9/3

48

開発の流れ

- ◆ 作成した関数を **Vitis HLS** でハードウェア化
- ◆ Vivado でブロック図を作成
- ◆ PYNQ にアップロードし、動作確認

この要領は点滅回路のときと同じ



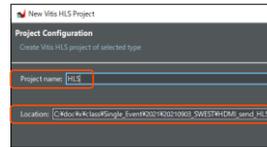
SWEST/ACRI 共同企画

2021/9/3

55

Vitis HLS のプロジェクト作成 (1)

- ◆ Vitis HLS を起動して、プロジェクト作成
 - File → New Project
 - プロジェクト名と作業ディレクトリを指定 (Vivado と同じ要領で) して、Next



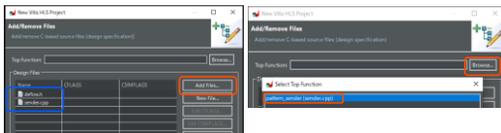
SWEST/ACRI 共同企画

2021/9/3

56

Vitis HLS のプロジェクト作成 (2)

- ◆ 回路記述のソースコードを指定する
 - Add Files で回路記述のソースコードを追加
 - Top Function 横の Browse をクリック
 - pattern_sender 関数を選択して、OK
 - Top Function が pattern_sender に変わったことを確認して、Next



SWEST/ACRI 共同企画

2021/9/3

57

Vitis HLS のプロジェクト作成 (3)

- ◆ テストベンチのソースコードを指定する
 - Add Files でテストベンチのソースコードを追加し、Next
- ◆ 対象ボードの選択
 - Part の横の ... ボタンを押す
 - Boards → PYNQ-Z1 を選択して、OK
 - 最後に Finish ボタン



SWEST/ACRI 共同企画

2021/9/3

58

C/C++ による動作チェック

- ◆ まずはソフトウェアとしてバグがないかの確認
 - Project → Run C Simulation
 - その後の設定はデフォルトのまま OK
 - しばらく待つと実行結果とともに「Finished C Simulation」と表示される



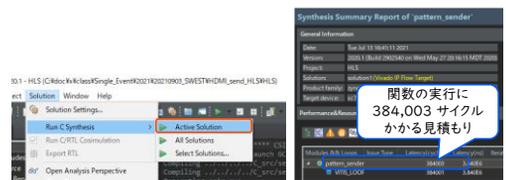
SWEST/ACRI 共同企画

2021/9/3

59

関数を高位合成

- ◆ 次に関数を高位合成でハードウェアに変換
 - Solution → Run C Synthesis → Active Solution
 - 1分程度待つと高位合成が完了し、概要レポートが表示される



SWEST/ACRI 共同企画

2021/9/3

60

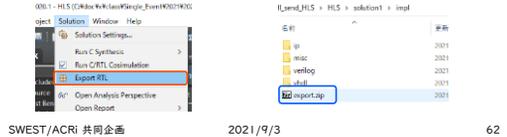
高位合成後の回路のチェック

- ◆ 関数が正しくハードウェア化されたか検証
 - Solution → Run C/RTL Cosimulation
 - テストベンチ部分は C/C++ で、ハードウェア化した関数は論理シミュレーション
 - うまく行けば、実行結果とともに PASS と表示される



高位合成後の回路のエクスポート

- ◆ 回路を Vivado で使える形式で出力
 - Solution → Export RTL
 - Format Selection が Vivado IP (.zip) であることを確認して、OK
 - プロジェクトのディレクトリの solution1/impl に export.zip が作成される



使用する回路の準備

- ◆ 適当な場所にフォルダを作成し、必要な回路のファイルを投入
 - rgb2dvi: Digilent 社のビデオエンコーダ*
 - tmds_vl_0: TMDS インタフェースの定義*
 - sender: 先ほど作成したパターン送信回路

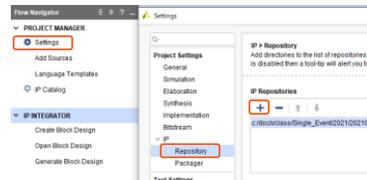


* <https://github.com/Digilent/vivado-library/> からダウンロードできる。

SWEST/ACRI 共同企画 2021/9/3 63

リポジトリの追加

- ◆ Vivado のプロジェクトを作成したら……
 - Project Manager → Settings
 - IP → Repository
 - + を押して、先に作成したフォルダを追加

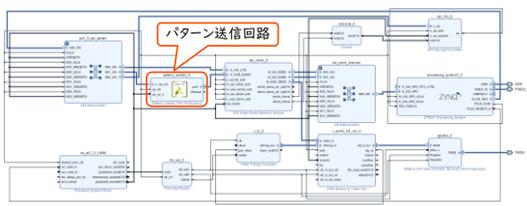


SWEST/ACRI 共同企画 2021/9/3 64

ブロック図の作成

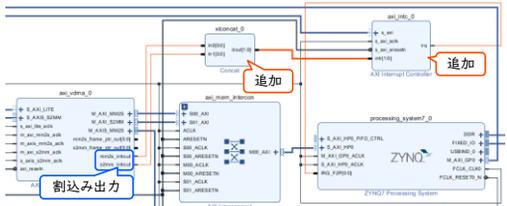
- ◆ ほぼ、以下の記事の「テストパターン表示システムの構築」の節と同じ

<https://www.acri.c.titech.ac.jp/wordpress/archives/8512>



ブロック図の作成: 相違点

- ◆ PYNQ の場合、割込みの信号線の接続が必要
 - 割込みコントローラなどを追加し、接続する



Python プログラム

- ◆ 一定時間パターンの送信を定期的に行う

```
In [2]: from pyng import Overlay
import time
import math

pl = Overlay("hdi_sender.bit")
sender = pl.pattern_sender_0
vdma = pl.axi_vdma_0

fbuf0, fbuf1, fbuf2 = video_initialize(vdma)
start_time = current_time + time.time()
current_frame = 0
frame_processed = 0

while current_time - start_time < 20:
    current_time = time.time()
    frame = math.floor((current_time - start_time) * 60)
    if current_frame == frame:
        current_frame = frame
        frame_processed += 1

    sender_register_map.frame = current_frame
    sender_register_map.CTRL_AP_START = 1
    while sender_register_map.CTRL_AP_DONE == 0:
        pass

    video_finalize(vdma, fbuf0, fbuf1, fbuf2)
    frame_processed += 1
```

フレーム番号 (経過時刻 × 60) が増加するまで待つ

パターン送信回路を起動し、終了するのを待つ

処理したフレーム数を出力

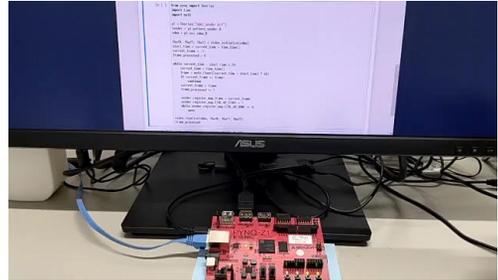
SWEST/ACRi 共同企画

2021/9/3

67

動作確認の結果

- ◆ 外部モニタへと出力させた結果 (動画)



SWEST/ACRi 共同企画

2021/9/3

68

参考情報

- ◆ PYNQ で高位合成により作成された回路を使用する別の例は、ACRi ブログの以下の記事で紹介している

<https://www.acri.c.titech.ac.jp/wordpress/archives/12514>

自作回路を PYNQ につなぐ簡単な方法 (5)



自作回路を PYNQ につなぐ簡単な方法 (5) (5)

- ◆ ソースコードは GitHub にて公開中

https://github.com/nfproc/connect_with_pynq/

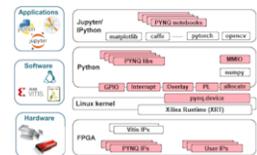
SWEST/ACRi 共同企画

2021/9/3

69

まとめ

- ◆ PYNQ で FPGA システム開発をお手軽に始めてみよう
 - ボードのセットアップ: ラズパイと同じ程度にはお手軽
 - HW オーバーレイ: 手順は多めだが慣れるとサクサク
 - 高位合成: C/C++ と Python で開発を始められる



画像出典: PYNQ: Python Productivity, <http://www.pynq.io>, retrieved 2021-07-08.

SWEST/ACRi 共同企画

2021/9/3

70

ACRi の各種サイト (1)

- ◆ ACRi ブログ
 - <https://www.acri.c.titech.ac.jp/wordpress/>
 - FPGA の使い方、応用などについて様々な記事が掲載されている (21年6月末時点で、34コース、175記事)
 - 「手を動かす系」の記事へのリンク集も <https://gw.acri.c.titech.ac.jp/wp/manual/links-to-acriblog>
- ◆ ACRi ルーム
 - <https://gw.acri.c.titech.ac.jp/wp/manual/welcome>
 - FPGA 開発環境・ボードを遠隔利用可能
 - ◆ PYNQ-Z1 への対応も検討中 (技術的課題多め)

SWEST/ACRi 共同企画

2021/9/3

71

ACRi の各種サイト (2)

- ◆ ACRi HLS Challenge (New!)
 - (正式オープンしたら URL 更新する)
 - お題に沿ったハードウェアを高位合成で作成・提出
 - 高位合成によるハードウェア設計技術の習得に

SWEST/ACRi 共同企画

2021/9/3

72