Python で FPGA プログラミング

PYNQ 祭り 2017 年 3 月 4 日

@ikwzm

自己紹介みたいなもの

- ハンドルネーム ikwzm
- 現在隠居中
- 52 才(けっこう年)
- 主に論理回路設計 (回路図～VHDL)
- たまにプログラム(アセンブラ～ C/Ruby)
- Python 歴 1 ヶ月

Python で FPGA プログラミング？

PYNQ (Python Productivity for Zynq) とは

- PYNQ = Python + Zynq
- Zynq = PS(Processing System) + PL(Programmable Logic)
  - PL は HDL or Vivado-HLS で記述。
  - PL を PS で制御。この制御を Python で記述。

- Python で PL をプログラミングするわけじゃない

本当に Python で FPGA プログラミング する話

目次

- Polyphony の紹介
- Polyphony での記述例(fibonacci)とインターフェース
- MessagePack-RPC Server on FPGA-SoC-Linux
- Design Flow
- PYNQ-Z1 での動作
- 課題

Polyphony (本日の主役) の紹介(1)

- Python ベースの高位合成コンパイラ
- 作っているのは　有限会社シンビー
  http://www.sinby.com/PolyPhony/index.html
- github でも公開中
  https://github.com/ktok07b6/polyphony
- 高位合成友の会 第３回(2015/12/08) でのスライド
  https://www.slideshare.net/ktok07b6/3-polyphony

Polyphony の紹介(2)

Polyphony はこんなツール(になる予定)

- 高位合成友の会 第 3 回(2015/12/08)のスライドより抜粋 -

・特徴
  - Python で書いたコードを Verilog-HDL に変換するための コンパイラ
  - コードは Python インタープリタで動く普通の Python のコード
  - 特殊なクラスや言語拡張は基本的になし
  - ただし、使える言語要素に制限有り
  - オープンソース

Polyphony の紹介(3)

Polyphony はこんなツール(になる予定)

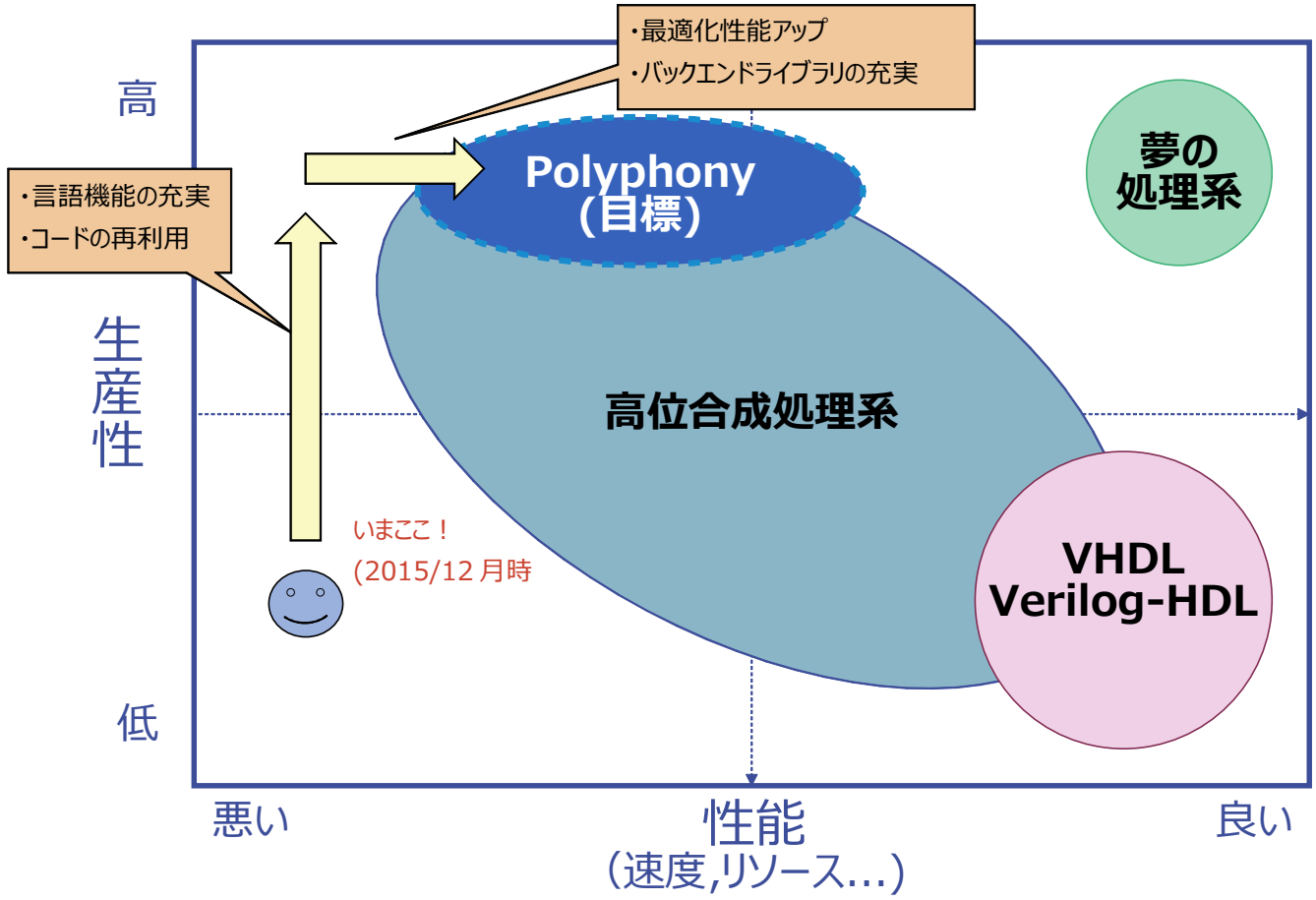
- 高位合成友の会 第３回(2015/12/08)のスライドより抜粋 -

- 得意なこと
  - 抽象度の高い処理記述
  - ソフトウェアのハードウェア化
  - 動くものを早く作る
- 出来ないこと、苦手なこと
  - クロック単位のタイミング制御
  - ハードウェアの制御
  - 性能の追求

Polyphony の紹介(4)

ハードウェア記述言語の生産性と性能

- 高位合成友の会 第３回(2015/12/08)のスライドより抜粋 -



Polyphony での記述例 - fib.py -

fib.py

```
from polyphony import testbench

def fib(n):
    if n <= 0: return 0
    if n == 1: return 1
    r0 = 0
    r1 = 1
    for i in range(n-1):
        prev_r1 = r1
        r1 = r0 + r1
        r0 = prev_r1
    return r1

@testbench
def test():
    expect = [0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610]
    for i in range(len(expect)):
        result = fib(i)
        assert expect[i] == result
        print(i, "=>", result)

test()
```

```
shell$ python3 fib.py
0 => 0
1 => 1
2 => 1
3 => 2
4 => 3
5 => 5
6 => 8
7 => 13
8 => 21
9 => 34
10 => 55
11 => 89
12 => 144
13 => 233
14 => 377
15 => 610
shell$
```

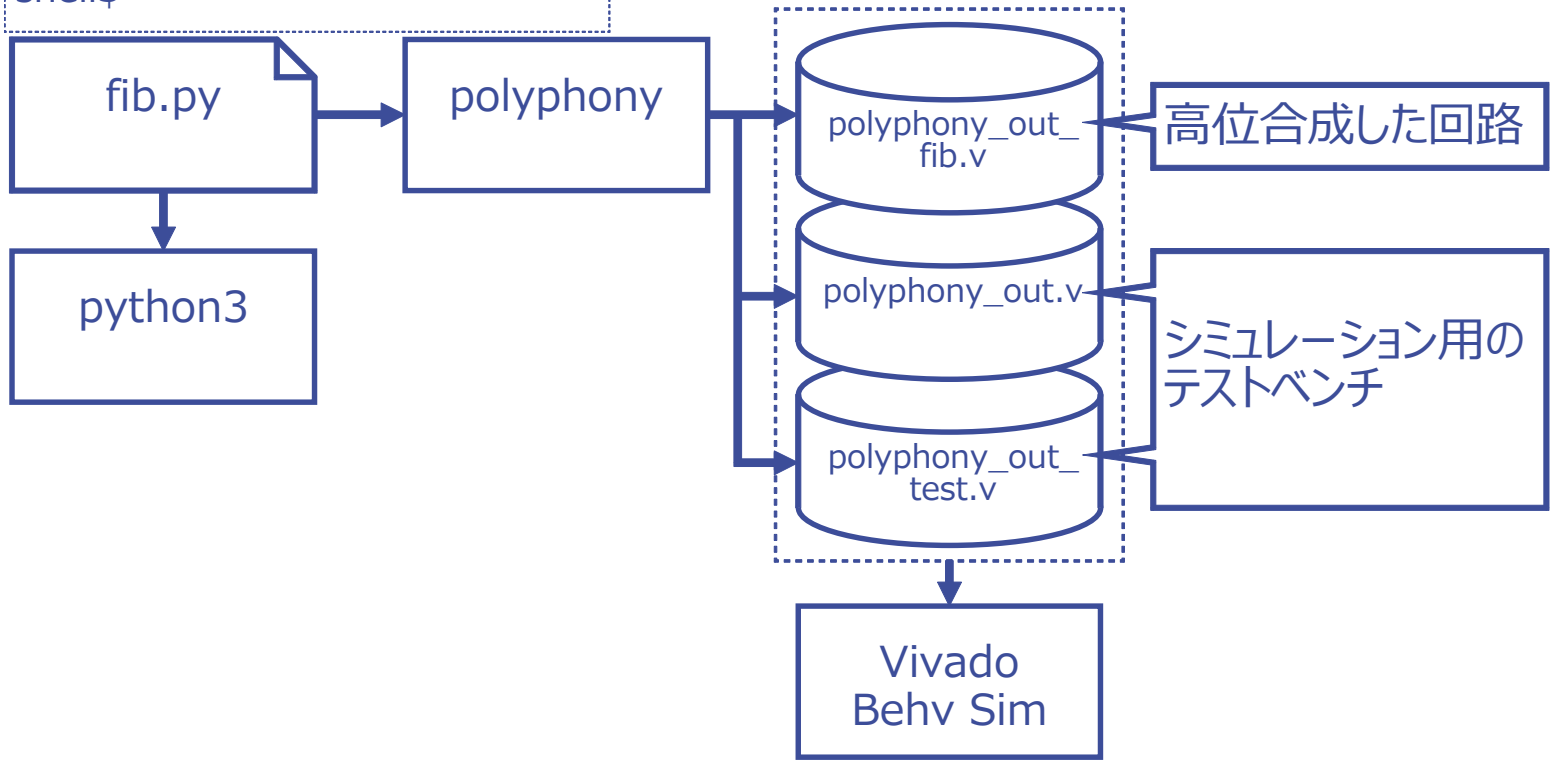
Polyphony で fib.py を高位合成

```
shell$ polyphony fib.py
shell$
```

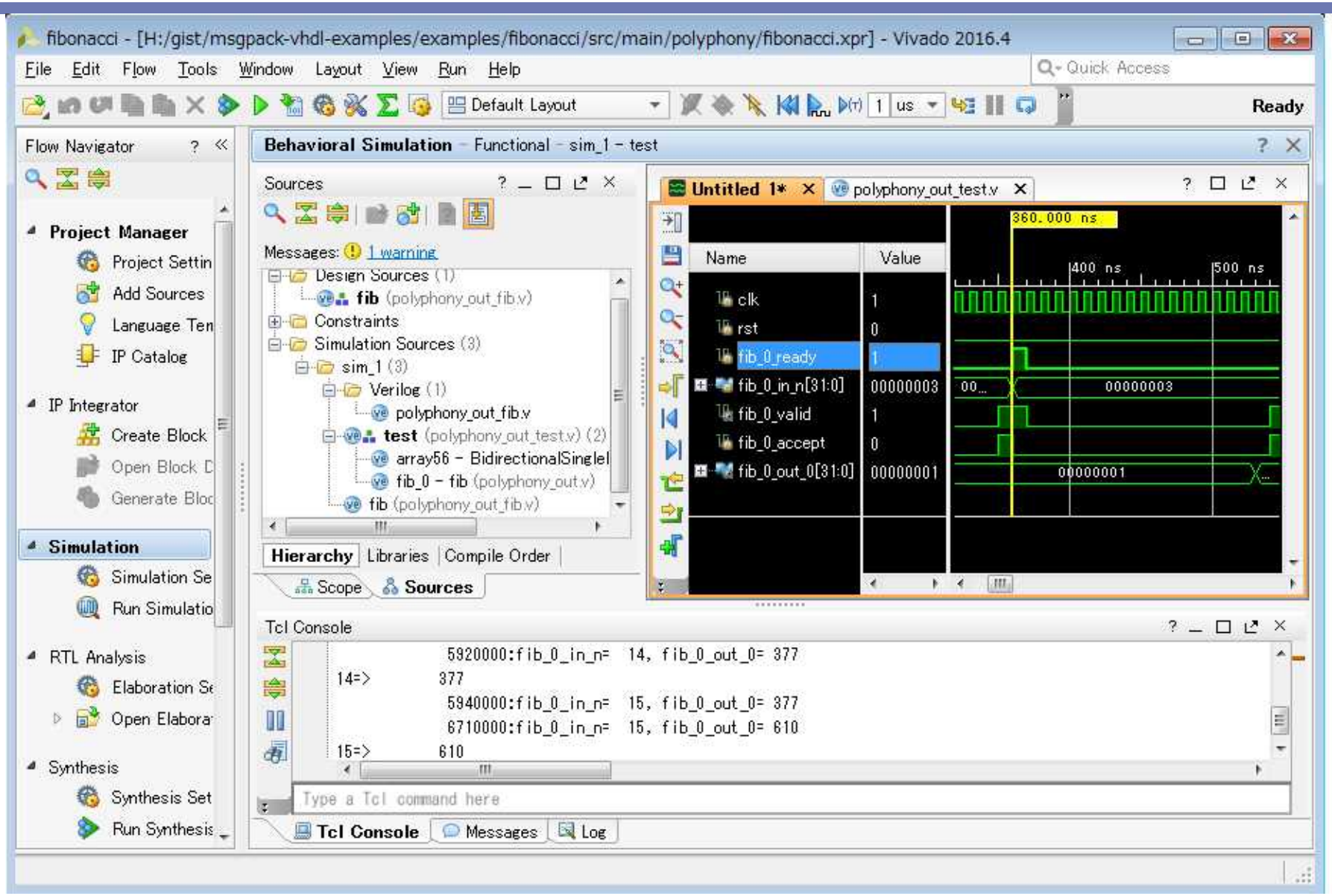
← 一見何も起きないようだが

```
shell$ ls -1
fib.py
polyphony_out.v
polyphony_out_fib.v
polyphony_out_test.v
shell$
```

← なんかファイルが出来てる



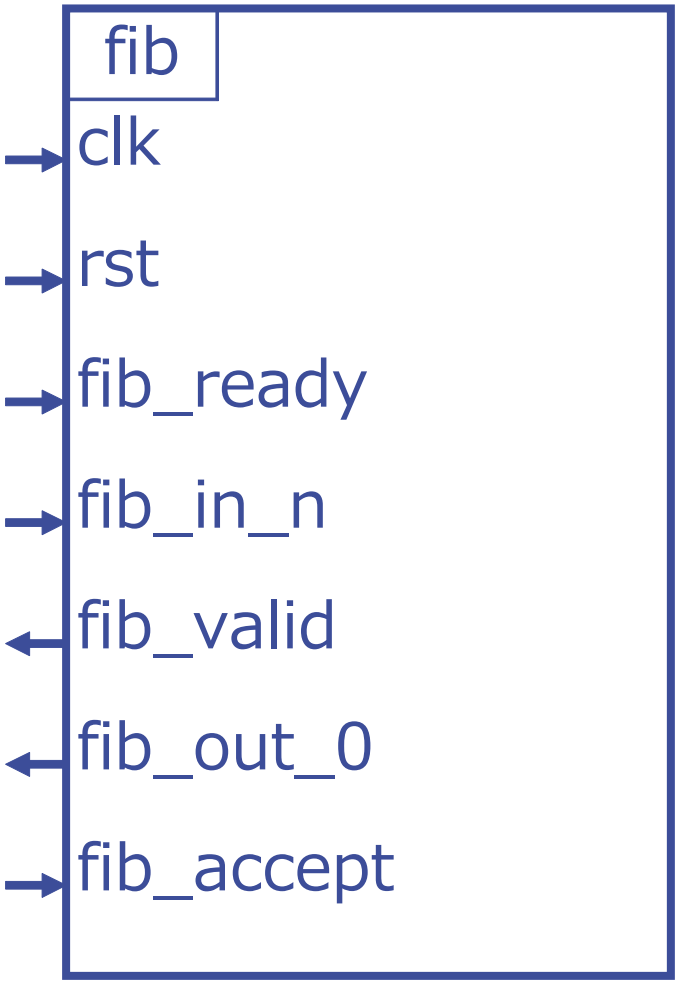
Vivado で Behavioral Simulation



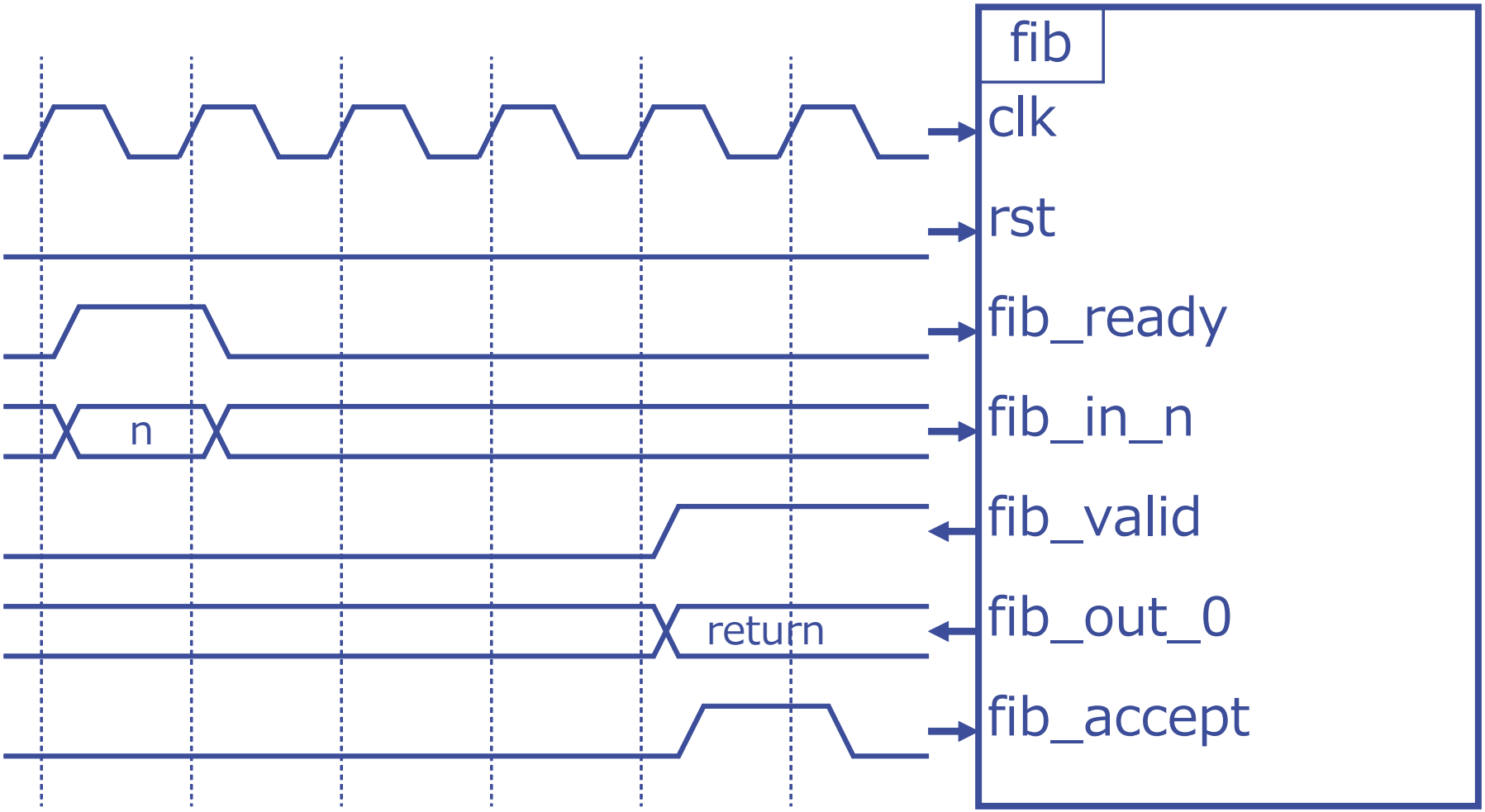
polyphony_out_fib.v の Interface(1)

polyphony_out_fib.v

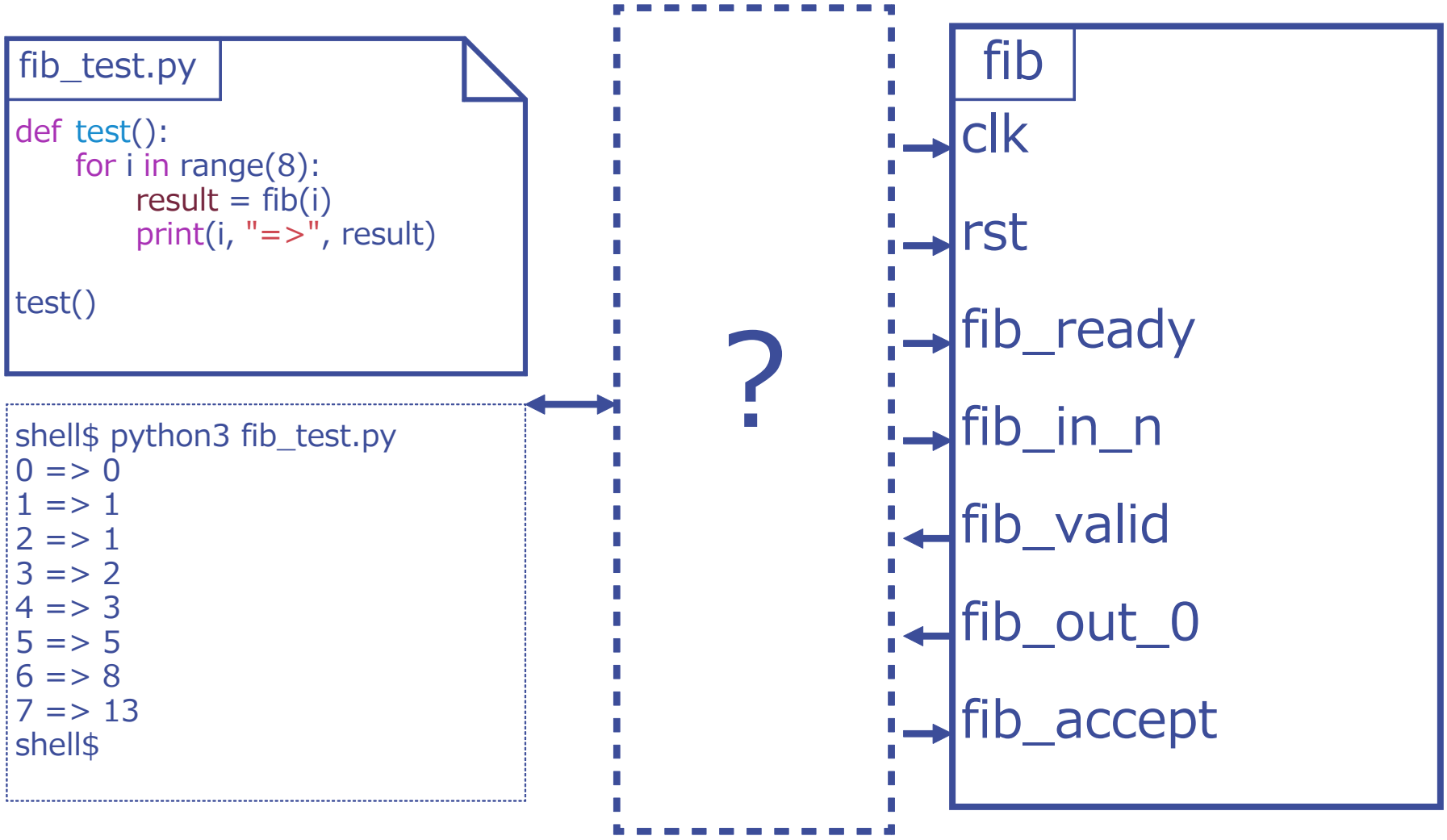
```
module fib
  (
      input wire clk,
      input wire rst,
      input wire fib_ready,
      input wire fib_accept,
      output reg fib_valid,
      input wire signed [31:0] fib_in_n,
      output reg signed [31:0] fib_out_0

  );

  //localparams
  localparam fib_b1_INIT = 0;
  localparam fib_b1_S1 = 1;
  localparam fib_ifthen3_S0 = 2;
  localparam fib_ifelse4_S1 = 3;
  localparam fib_ifthen6_S0 = 4;
  localparam fib_ifelse7_S0 = 5;
        :
     (中略)
        :
endmodule
```



polyphony_out_fib.v の Interface(2)



fib を PS(Processing System)から使うには？



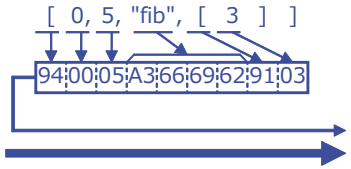
MessagePack-RPC (Remote Procedure Call)

fib_client.py

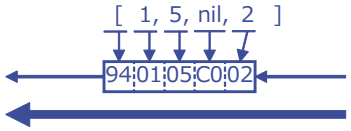
```
import msgpackrpc

cli = msgpackrpc.Client(
      msgpackrpc.Address(
            'localhost',18800))

def fib(n):
      return cli.call('fib', n)

def test():
      for i in range(5):
            result = fib(i)
            print(i, "=>", result)

test()
```

```
shell$ python3 fib_client.py
0 => 0
1 => 1
2 => 1
3 => 2
4 => 3
shell$
```

Request Message



Response Message

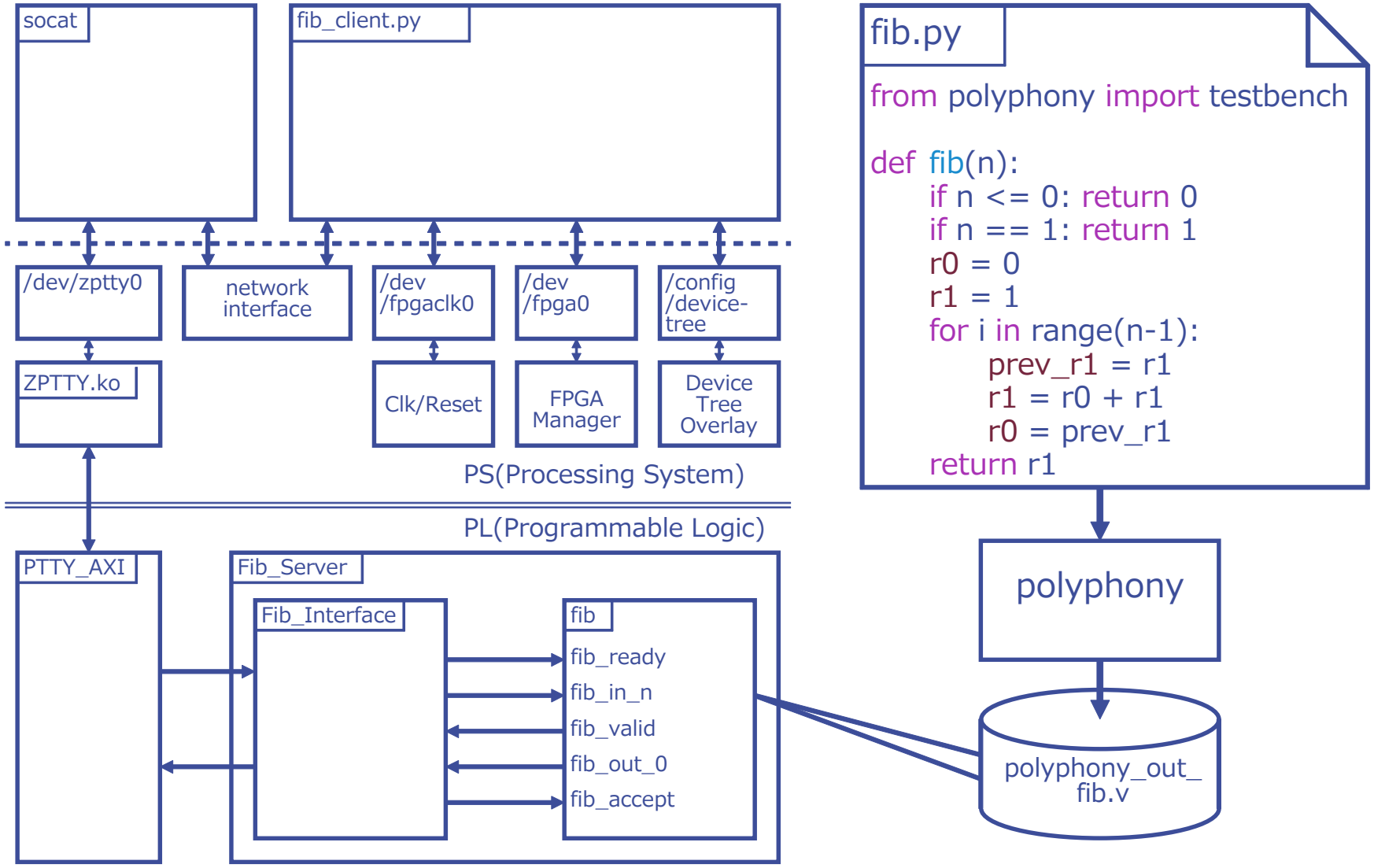


fib_server.py

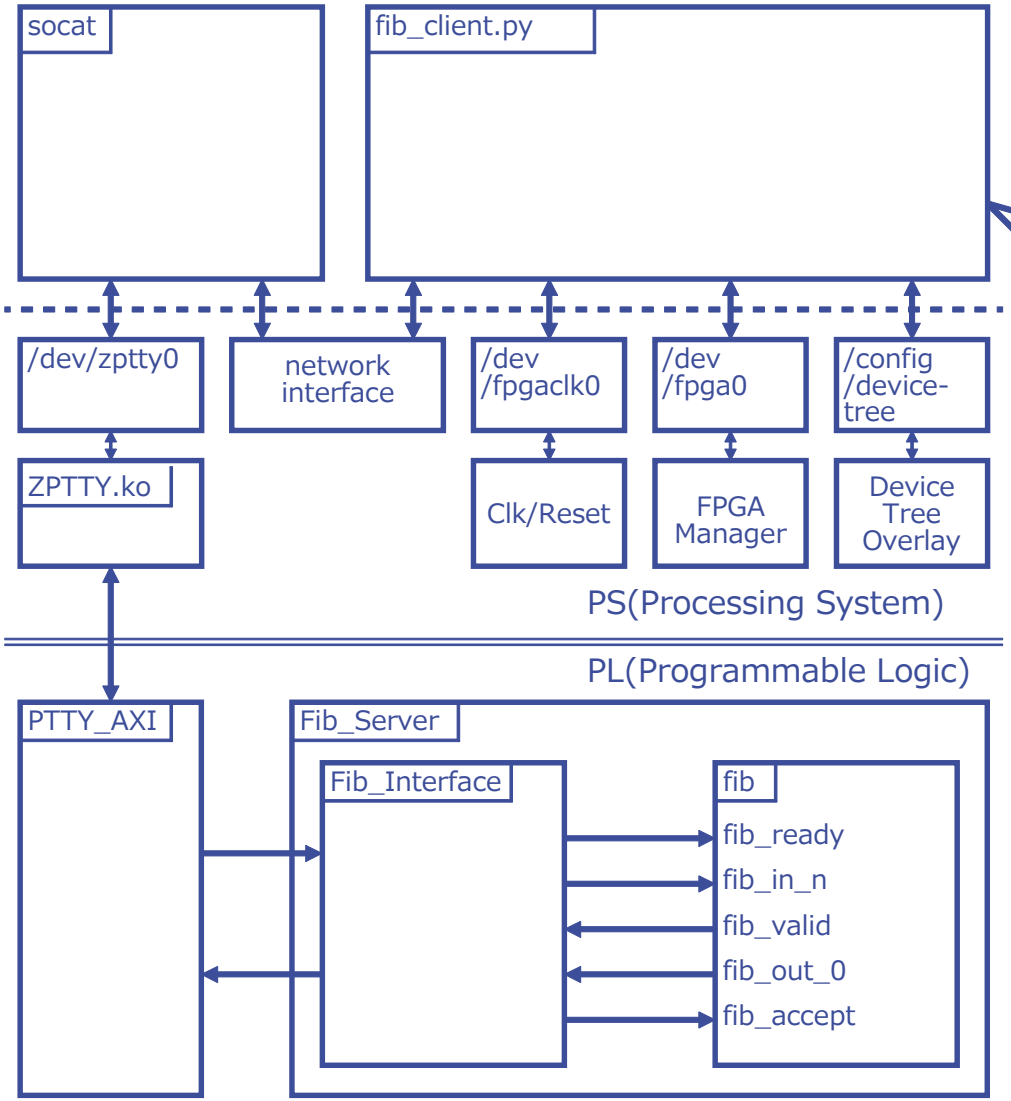
```
import msgpackrpc

class FibServer(object):
      def fib(self, n):
            if n <= 0: return 0
            if n == 1: return 1
            r0 = 0
            r1 = 1
            for i in range(n-1):
                  prev_r1 = r1
                  r1 = r0 + r1
                  r0 = prev_r1
            return r1

svr=msgpackrpc.Server(FibServer())
svr.listen(msgpackrpc.Address(
      'localhost',18800))
svr.start()
```

```
shell$ python3 fib_server.py &
shell$
```

MessagePack-RPC Server on FPGA-SoC-Linux(1)



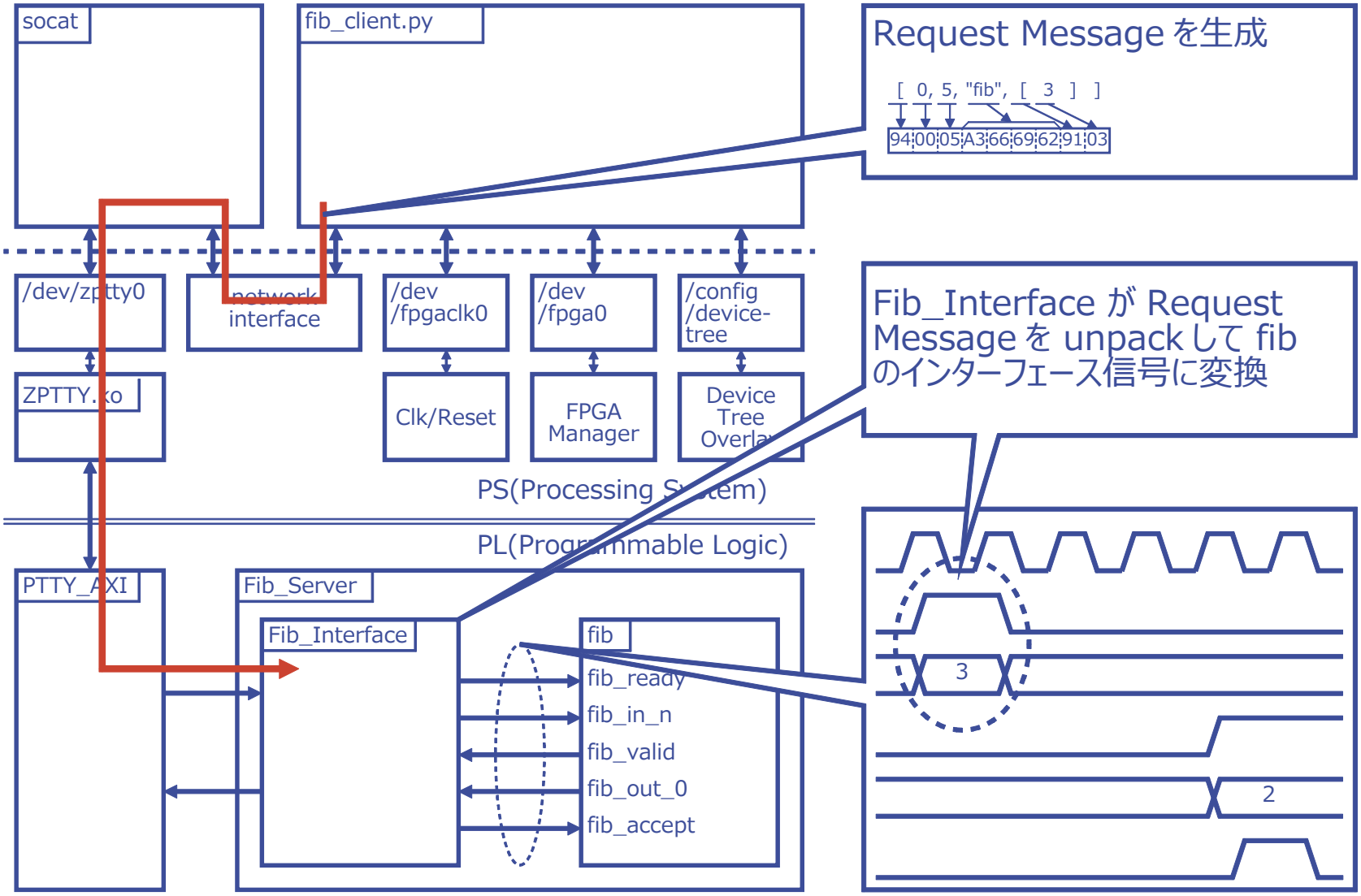
MessagePack-RPC Server on FPGA-SoC-Linux(2)



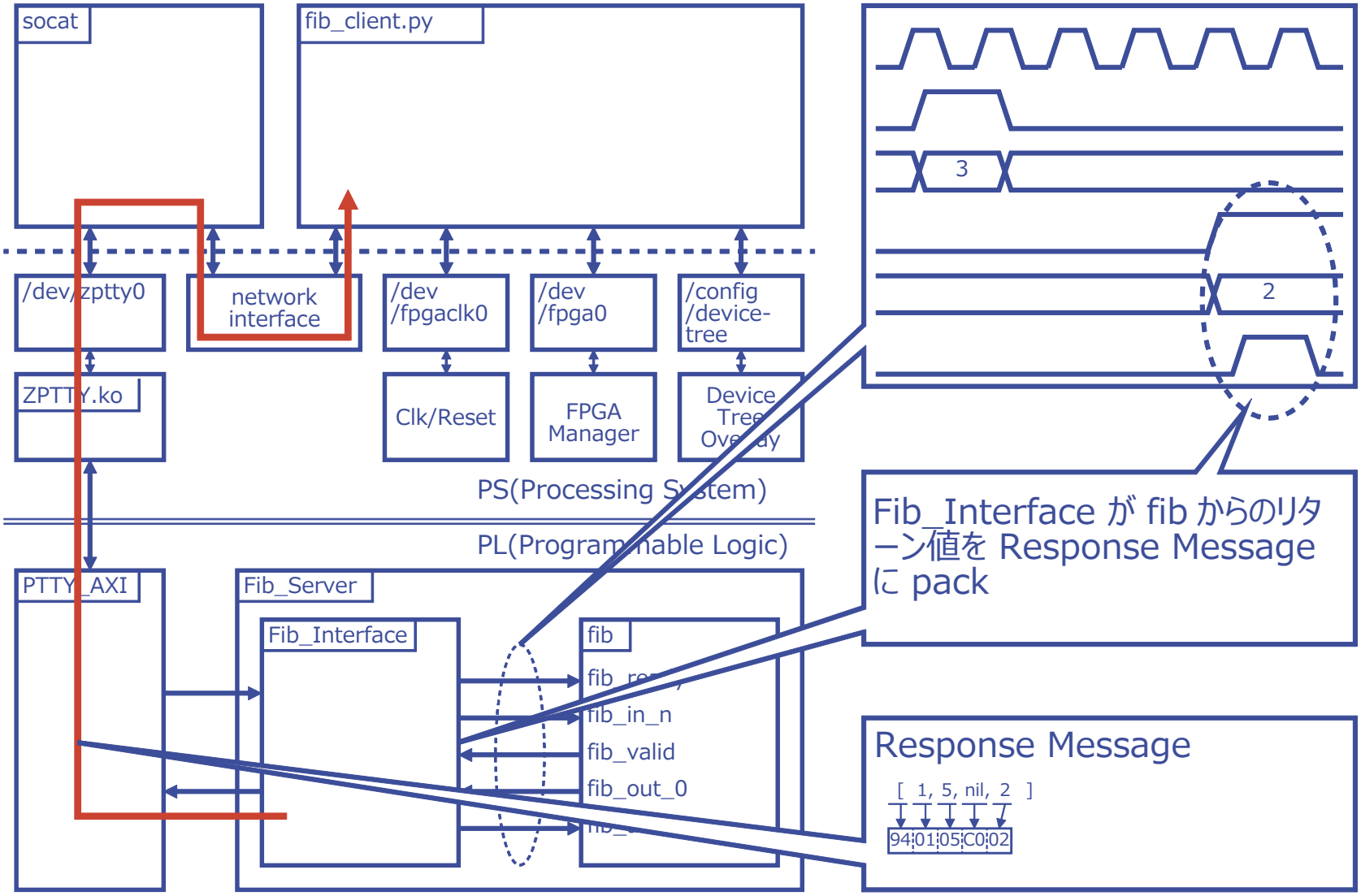
fib_client.py

```
import msgpackrpc

cli = msgpackrpc.Client(
      msgpackrpc.Address(
            'localhost',54321))

def fib(n):
      return cli.call('fib', n)

def test():
      for i in range(5):
            result = fib(i)
            print(i, "=>", result)

test()
```

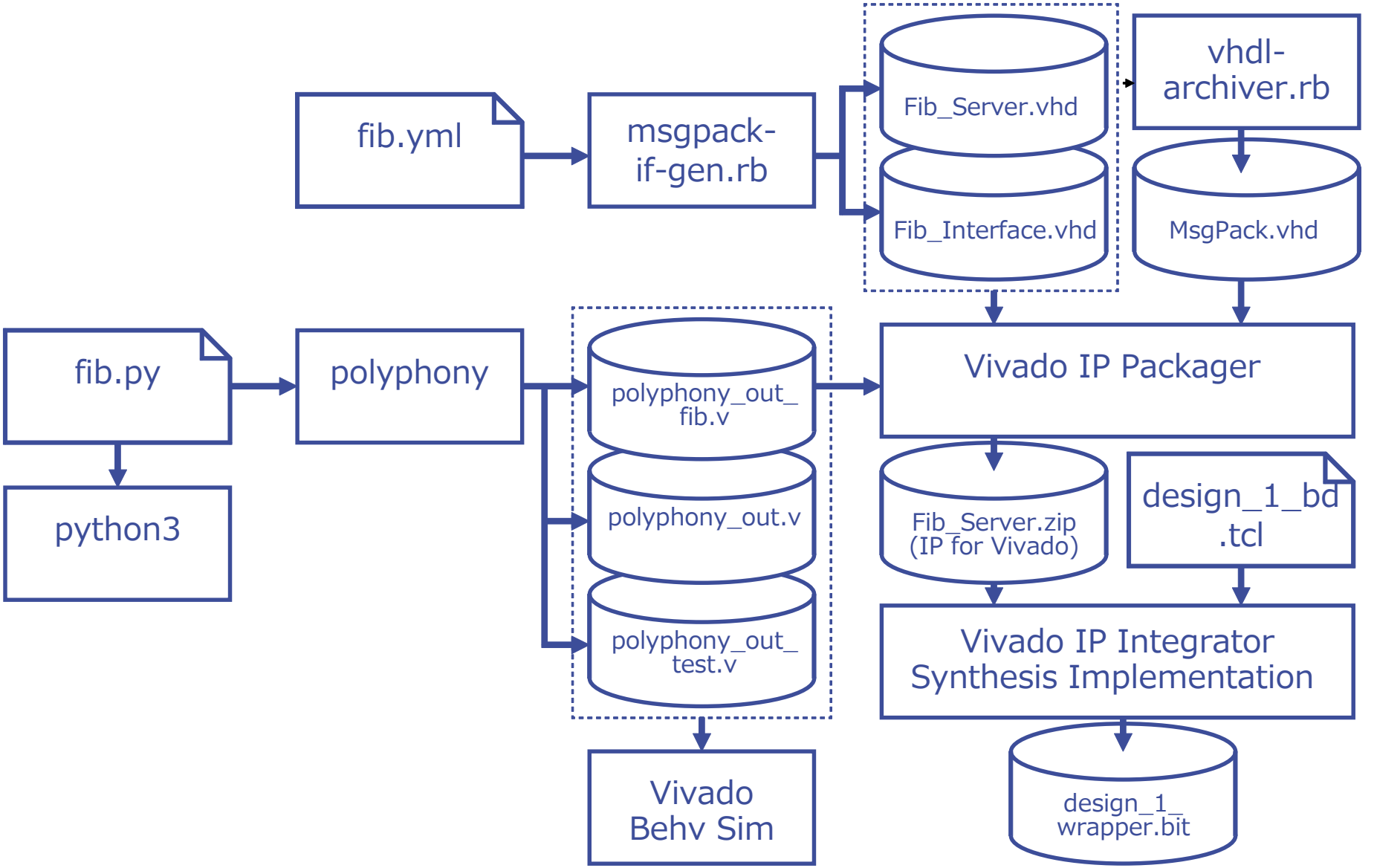
MessagePack-RPC Server on FPGA-SoC-Linux(3)



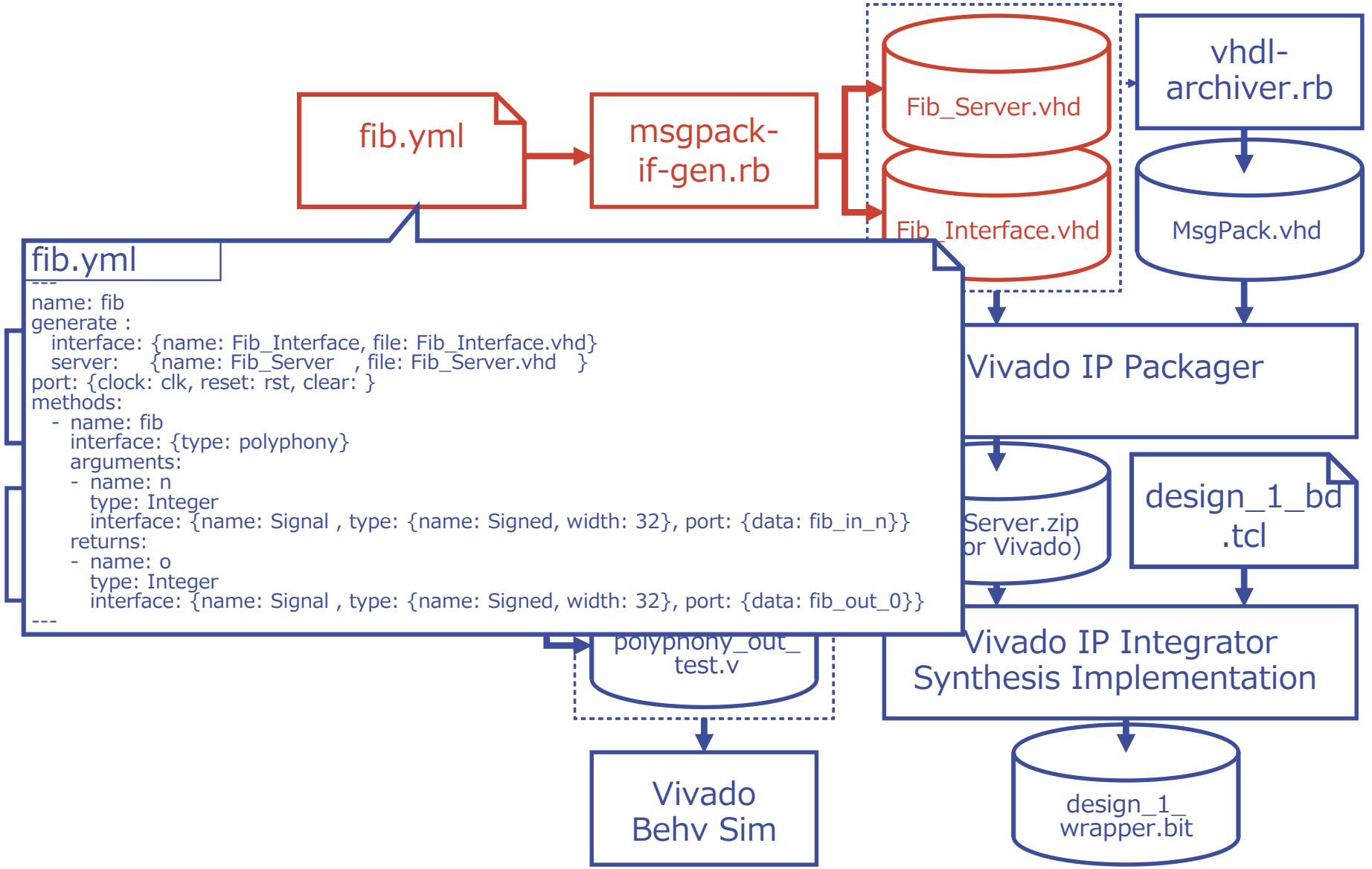
MessagePack-RPC Server on FPGA-SoC-Linux(4)



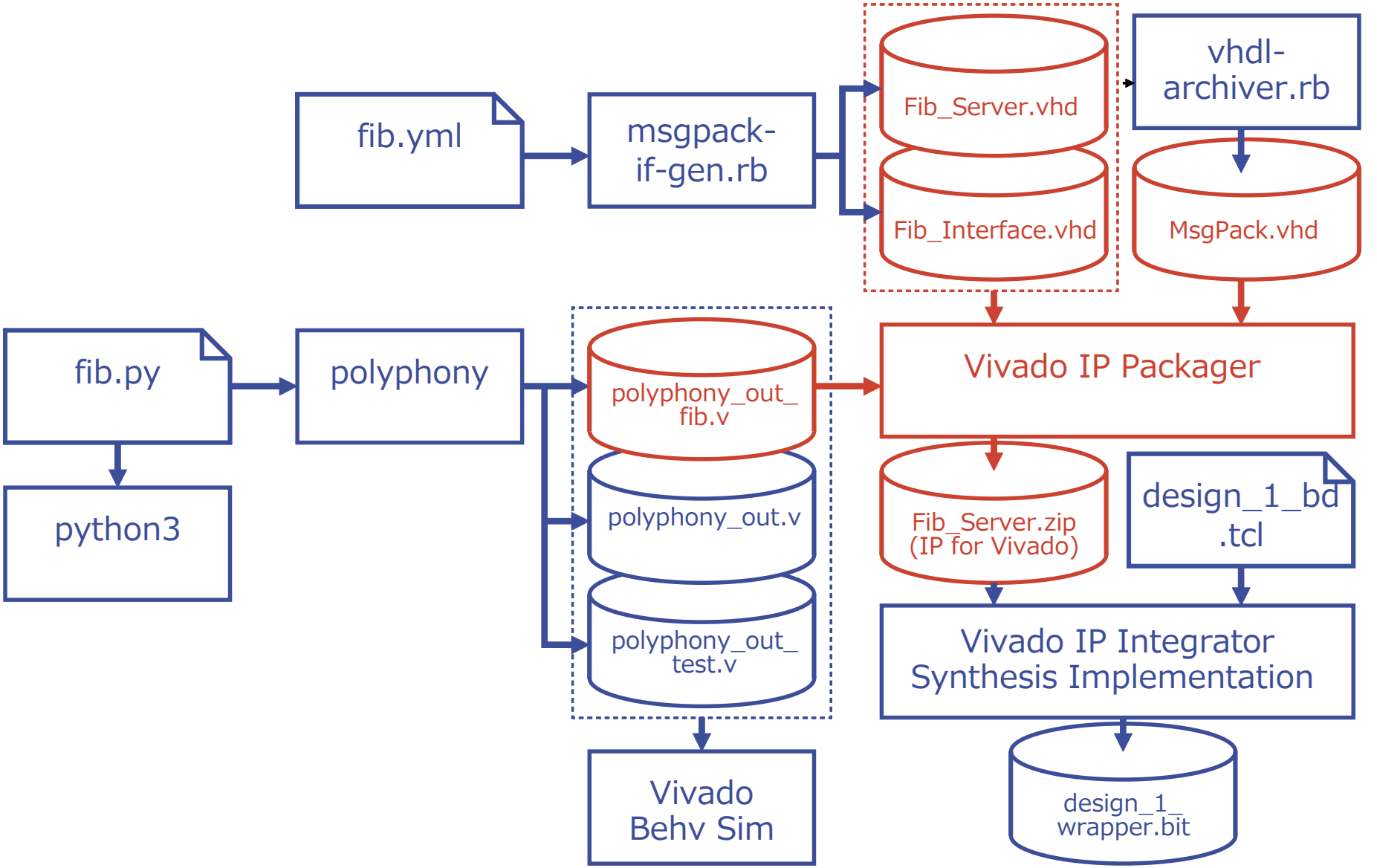
Design Flow


fib.yml
msgpack-
if-gen.rb
Fib_Server.vhd
Fib_Interface.vhd
vhdl-
archiver.rb
MsgPack.vhd
fib.py
python3
polyphony
polyphony_out_
fib.v
polyphony_out.v
polyphony_out_
test.v
Vivado
Behv Sim
Vivado IP Packager
Fib_Server.zip
(IP for Vivado)
design_1_bd
.tcl
Vivado IP Integrator
Synthesis Implementation
design_1_
wrapper.bit

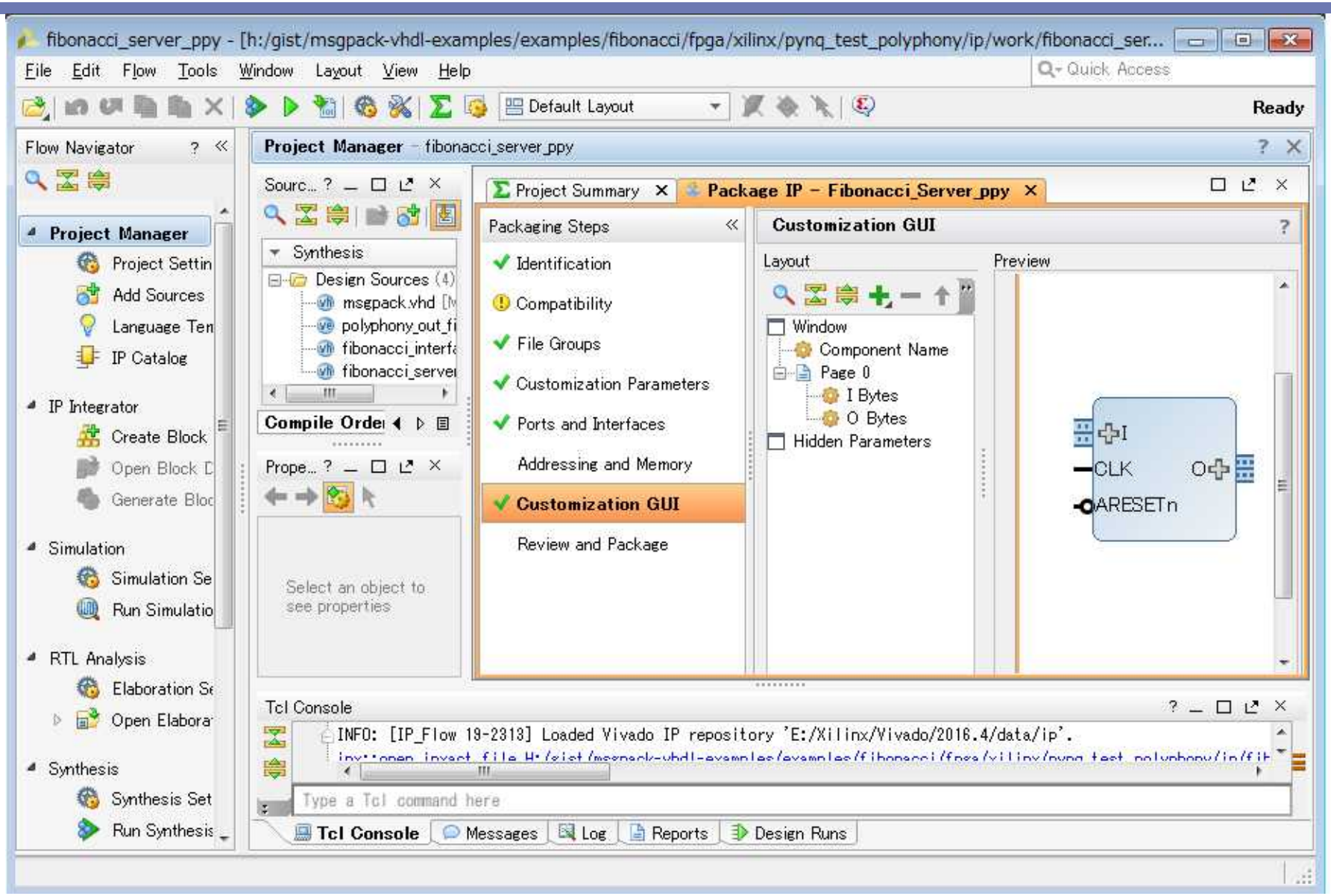
msgpack-if-gen



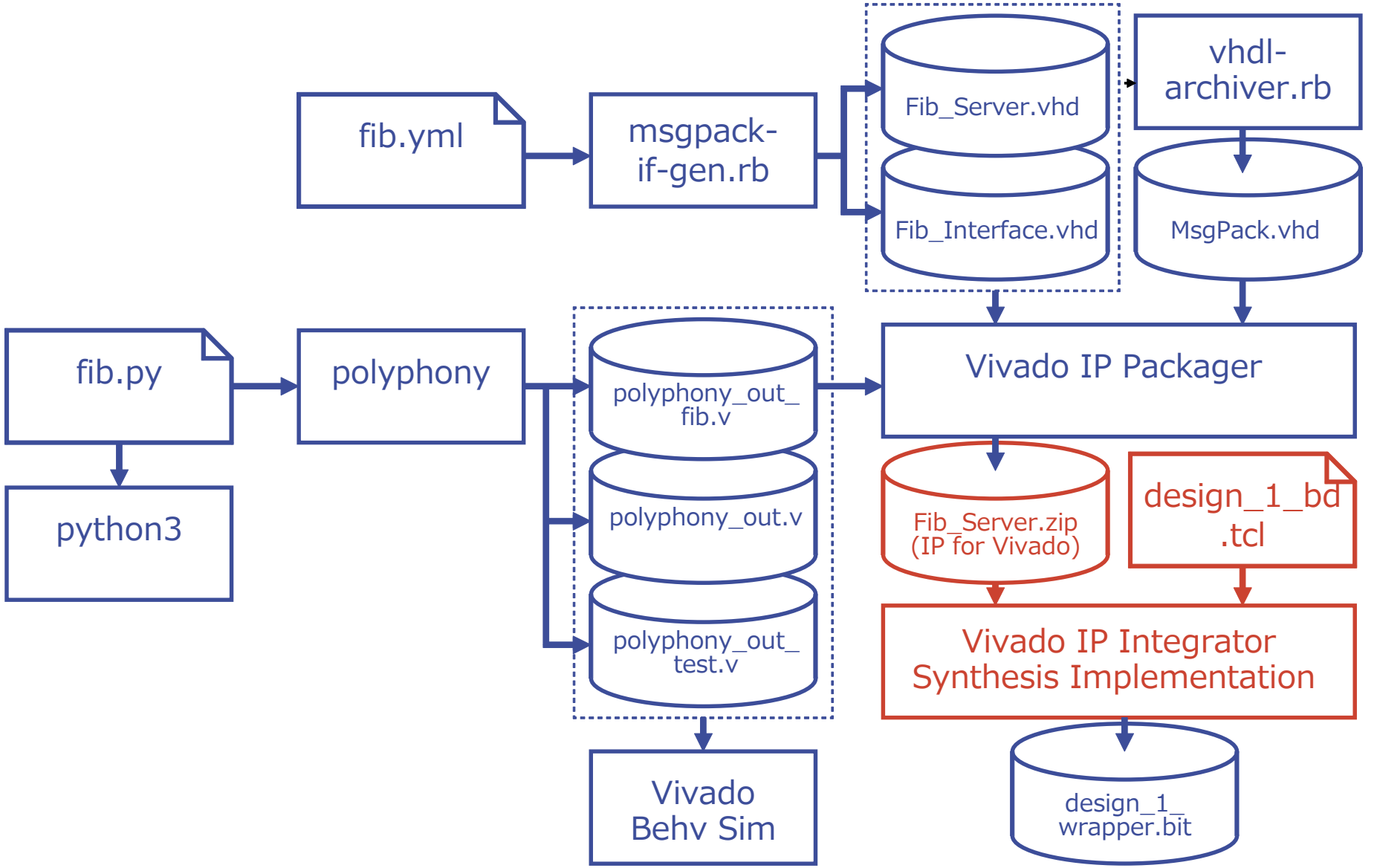
Vivado IP Packager (1)



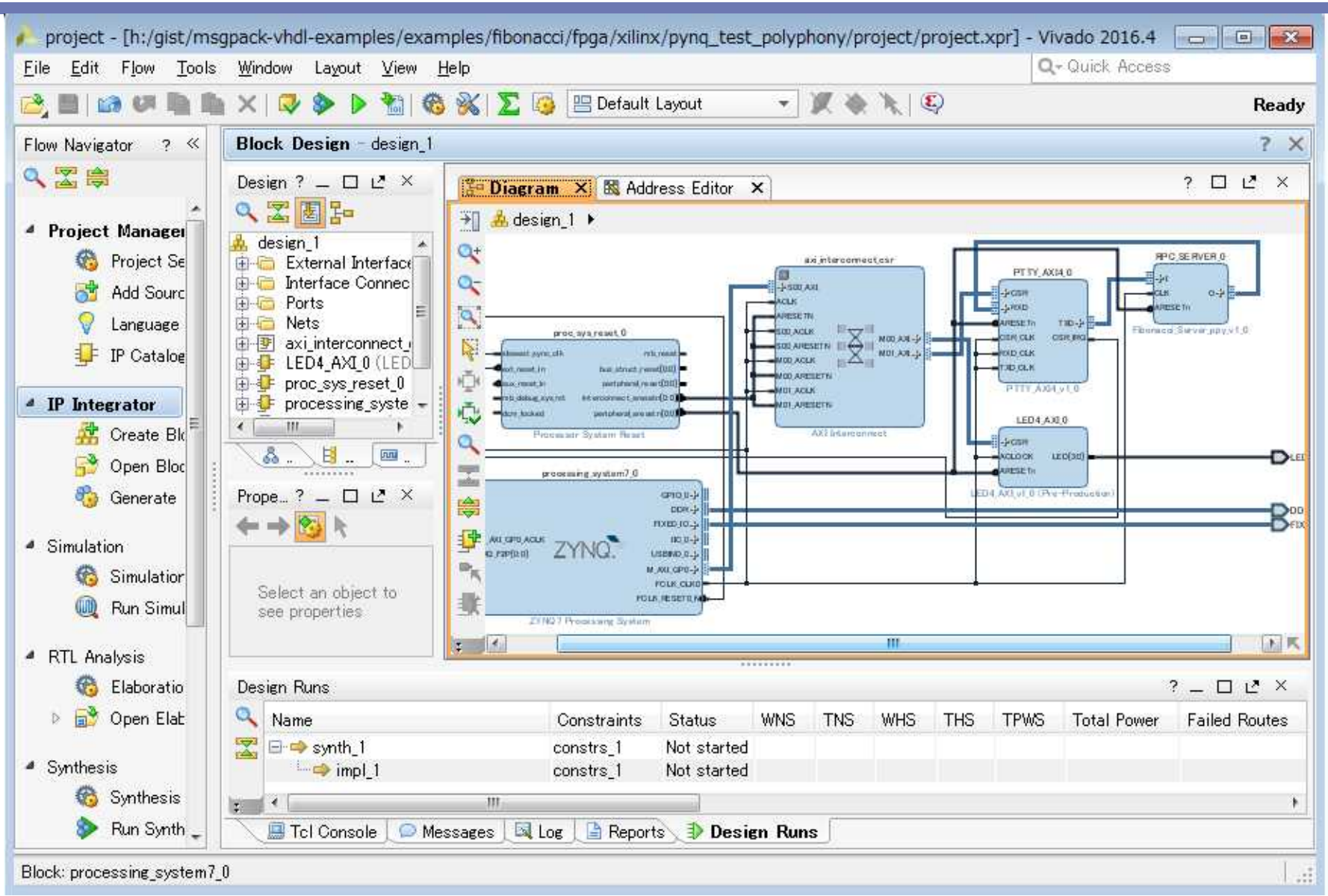
Vivado IP Packager (2)



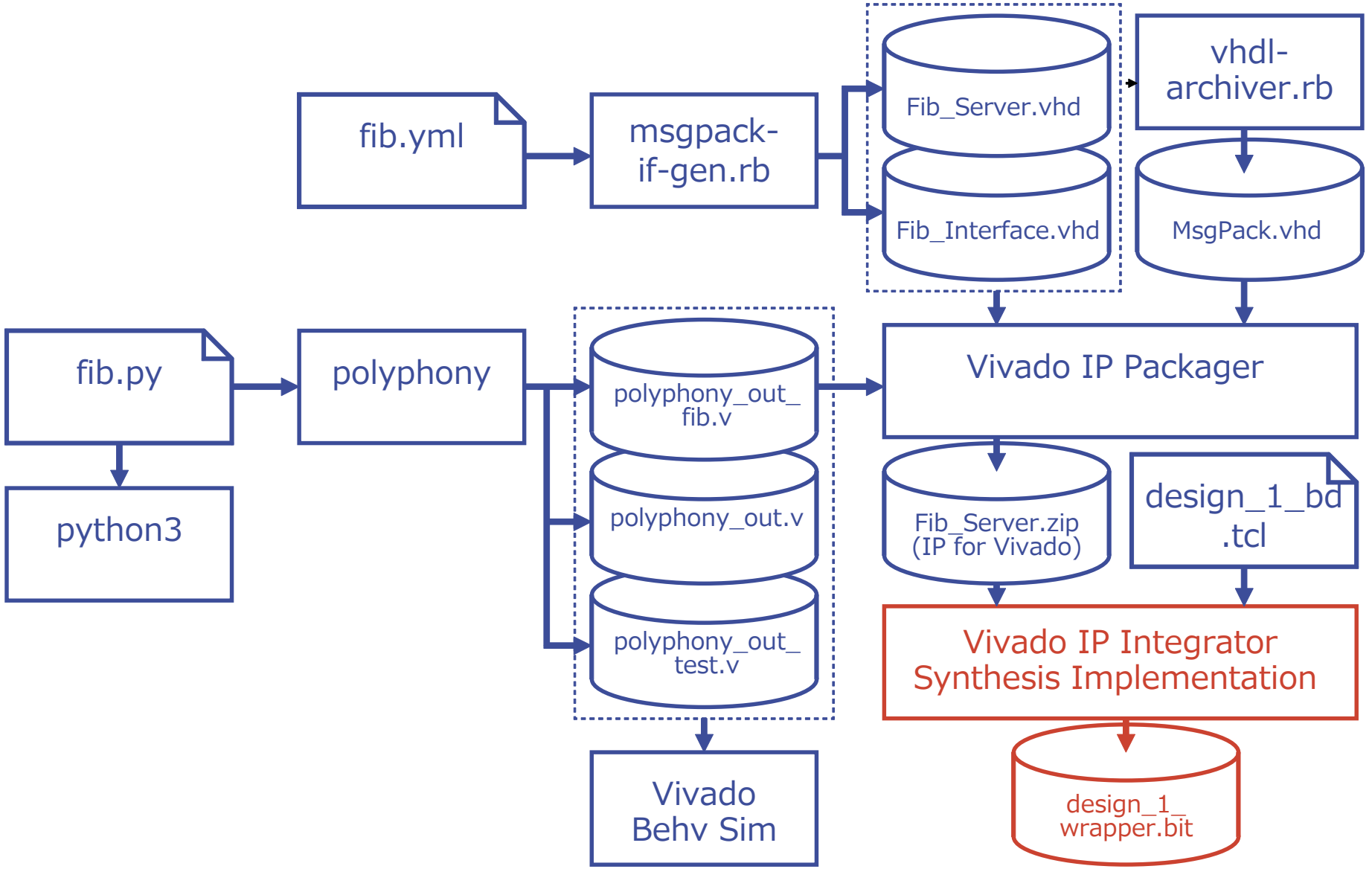
Vivado IP Integrator (1)



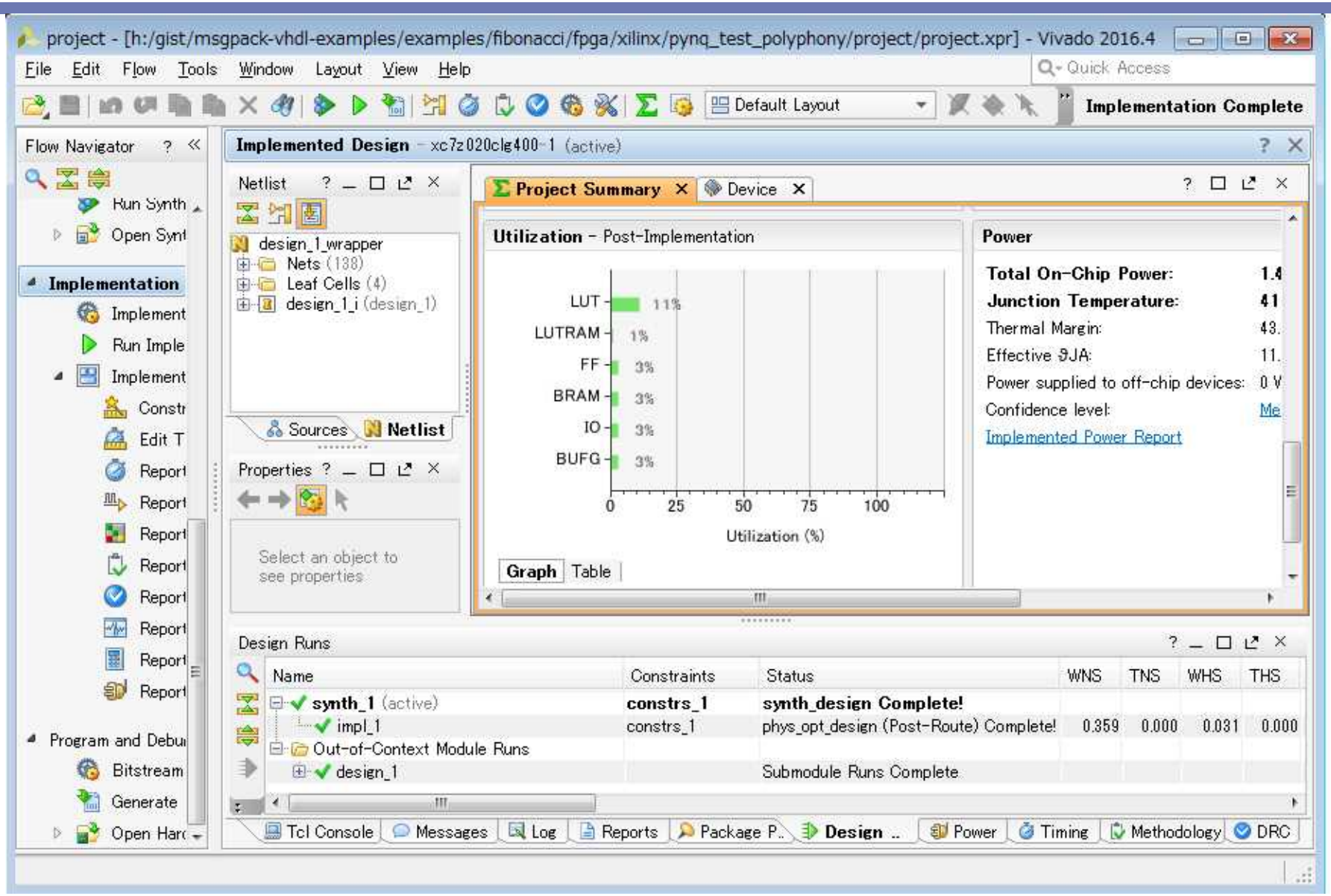
Vivado IP Integrator (2)



Vivado Synthesis & Implementation (1)



Vivado Synthesis & Implementation (2)



Load and start fib server on PL(Programmable Logic)

FPGA-SoC-Linux(https://github.com/ikwzm/FPGA-SoC-Linux)の場合

1. Download "design_1_wrapper.bit" to PL

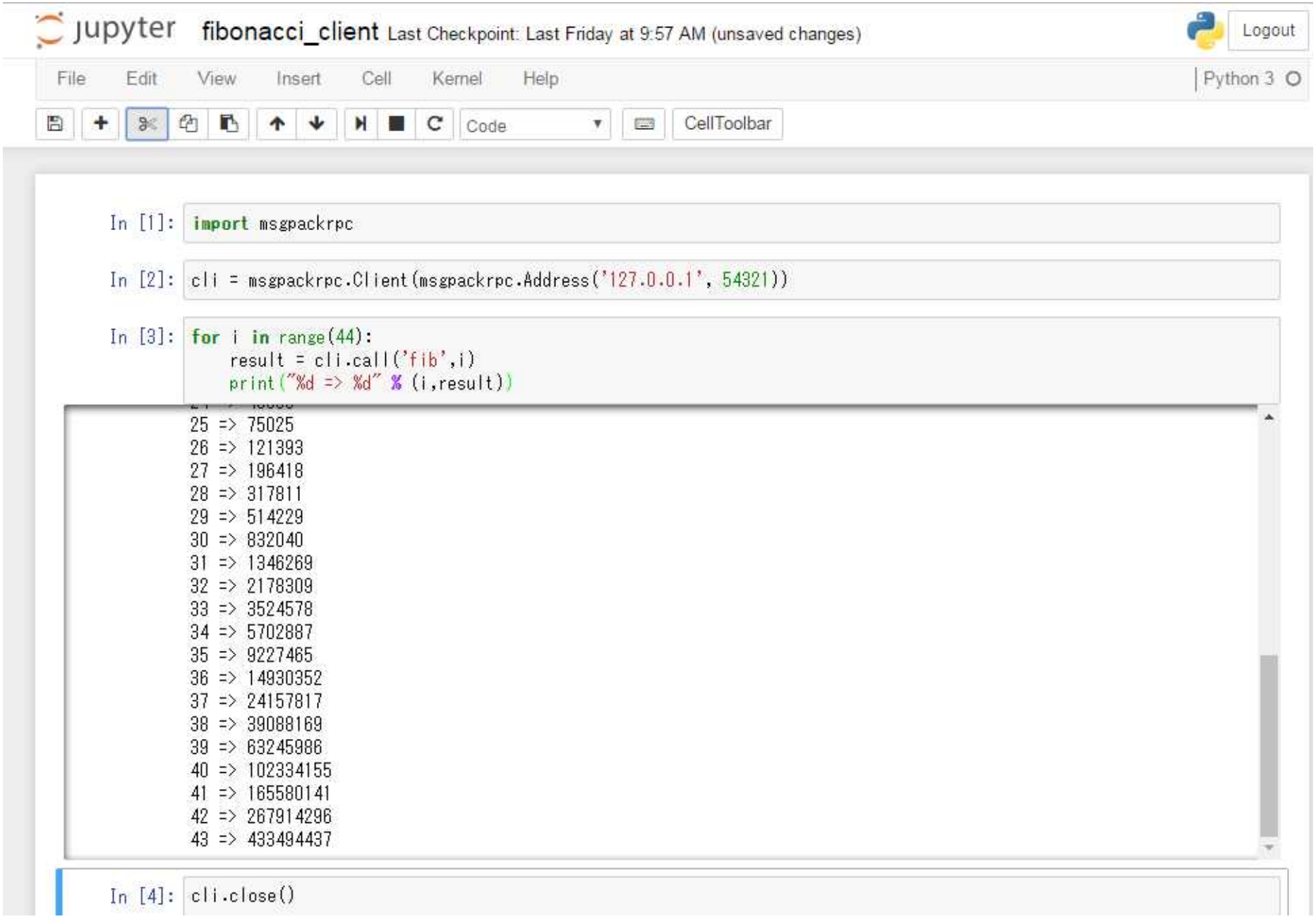
```
shell# echo 1 >/sys/class/fpgacfg/fpgacfg0/data_format
shell# echo 1 >/sys/class/fpgacfg/fpgacfg0/load_start
shell# dd if=design_1_wrapper.bit of=/dev/fpgacfg0 bs=1M
3+1 records in
3+1 records out
4045676 bytes (4.0 MB) copied, 0.296939 s, 13.6 MB/s
```

2. Load "zptty" device driver and overlay device-tree

```
shell# modprobe zptty
shell# dtbocfg.rb --install zptty0 --dts zptty0-zynq-zybo.dts
```

3. Start socat (background)

```
shell# socat -d -d tcp-listen:54321,fork /dev/zptty0,raw,nonblock &
[2] 2145
2017/03/01 09:34:13 socat[2145] N listening on AF=2 0.0.0.0:54321
```


jupyter fibonacci_client Last Checkpoint: Last Friday at 9:57 AM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Python 3
Code
CellToolbar
In [1]: import msgpackrpc
In [2]: cli = msgpackrpc.Client(msgpackrpc.Address('127.0.0.1', 54321))
In [3]: for i in range(44):
result = cli.call('fib',i)
print("%d => %d" % (i,result))
25 => 75025
26 => 121393
27 => 196418
28 => 317811
29 => 514229
30 => 832040
31 => 1346269
32 => 2178309
33 => 3524578
34 => 5702887
35 => 9227465
36 => 14930352
37 => 24157817
38 => 39088169
39 => 63245986
40 => 102334155
41 => 165580141
42 => 267914296
43 => 433494437
In [4]: cli.close()

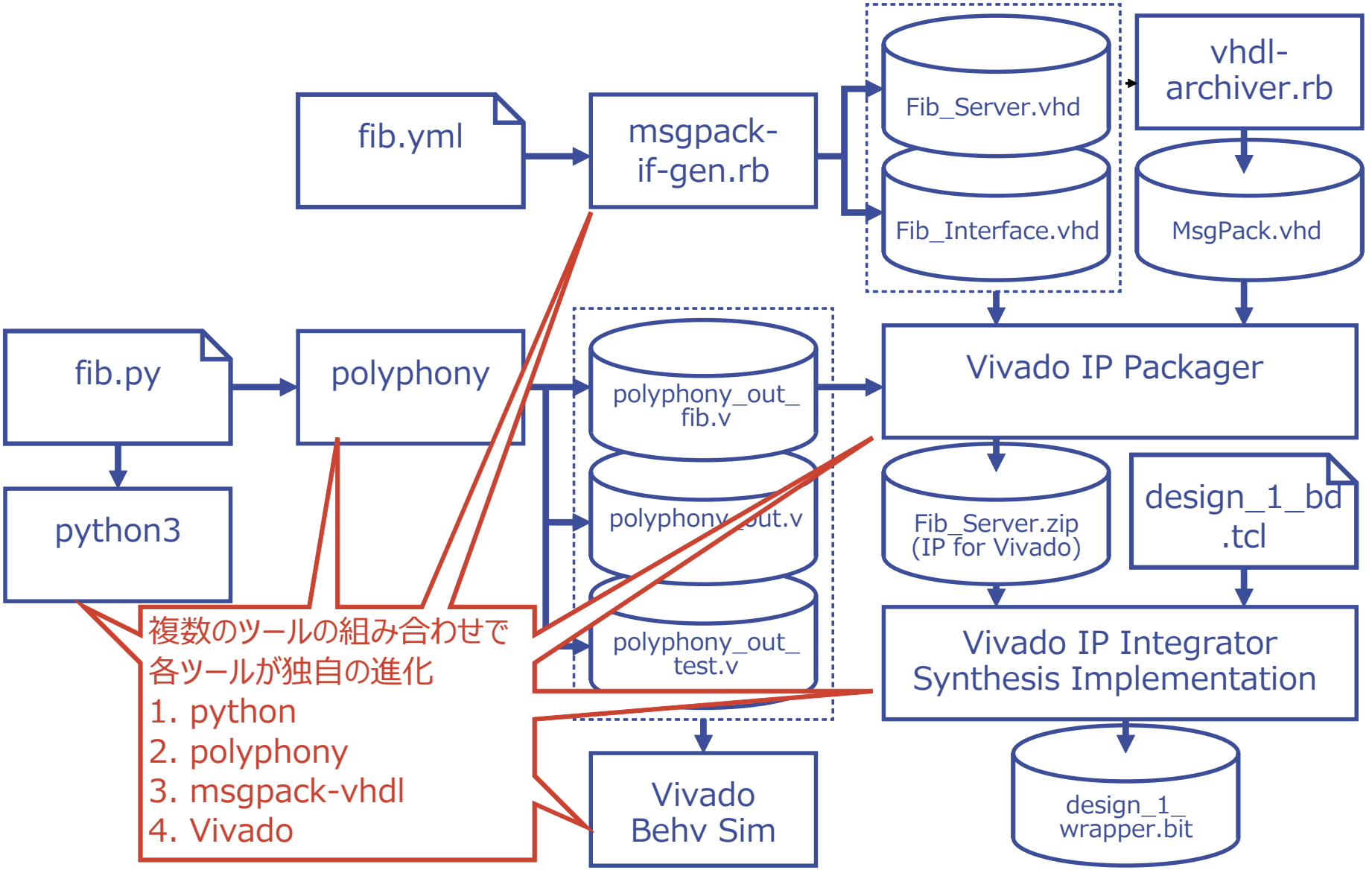
課題

1. Design Flow が複雑
    - すぐに切れそうな Tool Chain
    - msgpack-if-gen.rb 用の設定ファイル(fib.yml)が必要

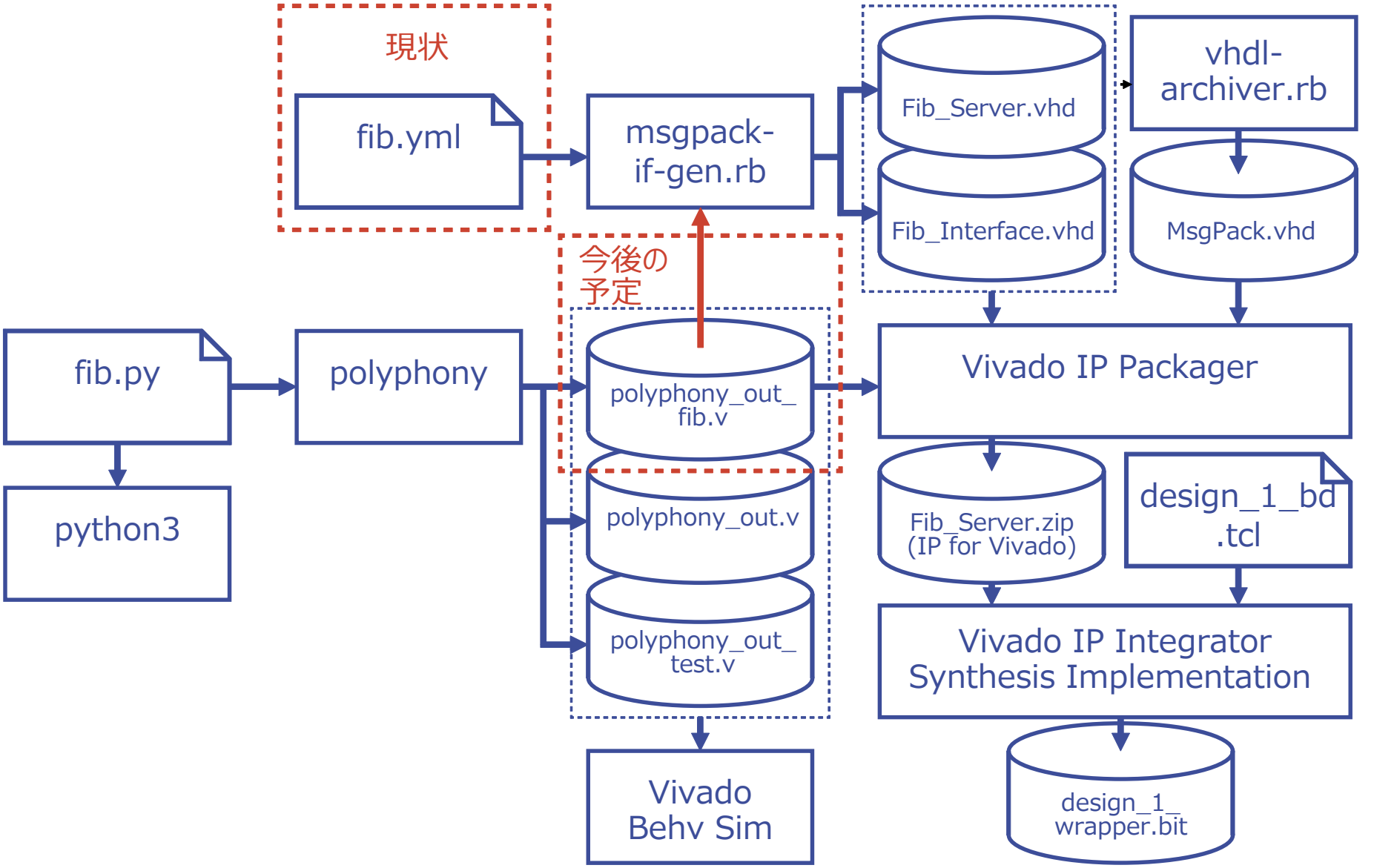
2. MessagePack-RPC のオーバーヘッド
    - レイテンシーの増大
    - スループットの悪化

3. Polyphony の言語的な制約

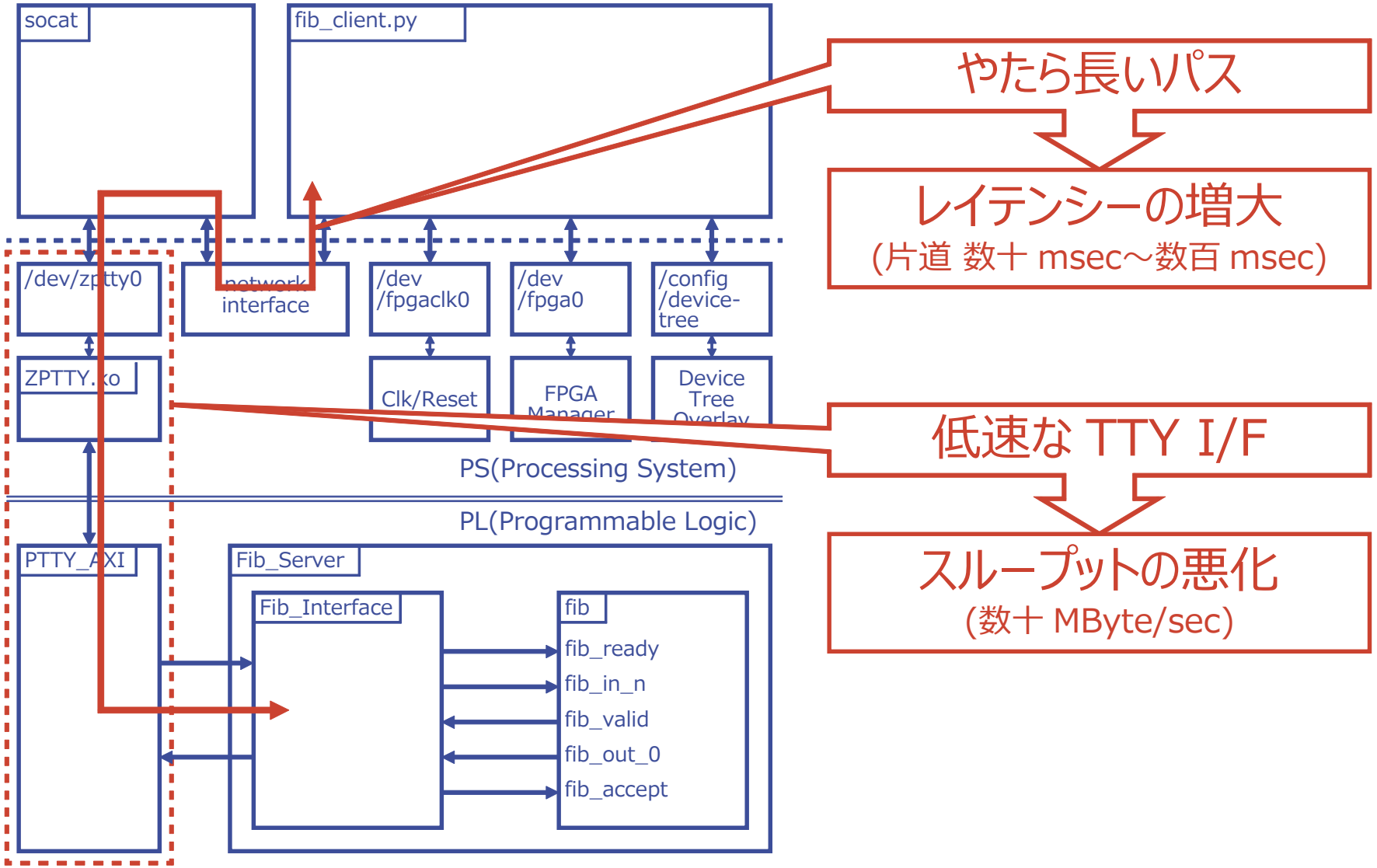
課題 1.1 Design Flow が複雑 - すぐに切れそうな Tool Chain



課題 1.2 Design Flow が複雑 - msgpack-if-gen.rb 用設定ファイル必要



課題 2. MessagePack-RPC のオーバーヘッド



課題 3. Polyphony の言語的な制約

- もともと論理回路は動的型付けが苦手(というかムチャブリです)
  ところが Python は動的型付け言語
  Polyphony では変数の型を限定することで対応
    使えるのは 32bit 整数、True、False だけ(2015/12 時点)

- 最近は List (1 次元かつ中身は整数のみ)やタプルも追加
- さらにクラスも使えるようになった(次頁参照)

- Polyphony は現在絶賛開発中
  今後いろいろと出来る様になるかも
  (多倍長整数、浮動小数点、文字列、辞書、多次元配列...)

Polyphony の進化 - mmult.py -

mmult.py

```
from polyphony import testbench
class StreamIn:
    def __init__(self, buf, size):
        self.size = size
        self.buf  = buf
    def read(self, addr):
        return self.buf[addr]
class StreamOut:
    def __init__(self, buf, size):
        self.size = size
        self.buf  = buf
    def write(self, addr, data):
        self.buf[addr] = data
def mmult(a_buf:list, b_buf:list, o_buf:list, size):
    a = StreamIn(a_buf, size)
    b = StreamIn(b_buf, size)
    o = StreamOut(o_buf, size)
    for i in range(size):
        o.write(i, a.read(i) * b.read(i))
@testbench
def test():
    a = [0,1,2,3,4,5,6,7]
    b = [0,1,2,3,4,5,6,7]
    o = [0,0,0,0,0,0,0,0]
    size  = 8
    mmult(a, b, o, size)
    for i in range(size):
        print(i, ":", a[i], "*", b[i], "=>", o[i])
        assert o[i] == a[i]*b[i]
test()
```

```
shell$ python3 mmult.py
0 : 0 * 0 => 0
1 : 1 * 1 => 1
2 : 2 * 2 => 4
3 : 3 * 3 => 9
4 : 4 * 4 => 16
5 : 5 * 5 => 25
6 : 6 * 6 => 36
7 : 7 * 7 => 49
shell$
```

- 2017 年 2 月現在
- List と Class が使える。
- テストベンチ(polyphony_out_test.v)は上手く動かなかった。
- List へのアクセスはウェイトが固定なので BlockRAM 限定。AXI マスター等は使えない。
- MessagePack-RPC は未対応。