

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)

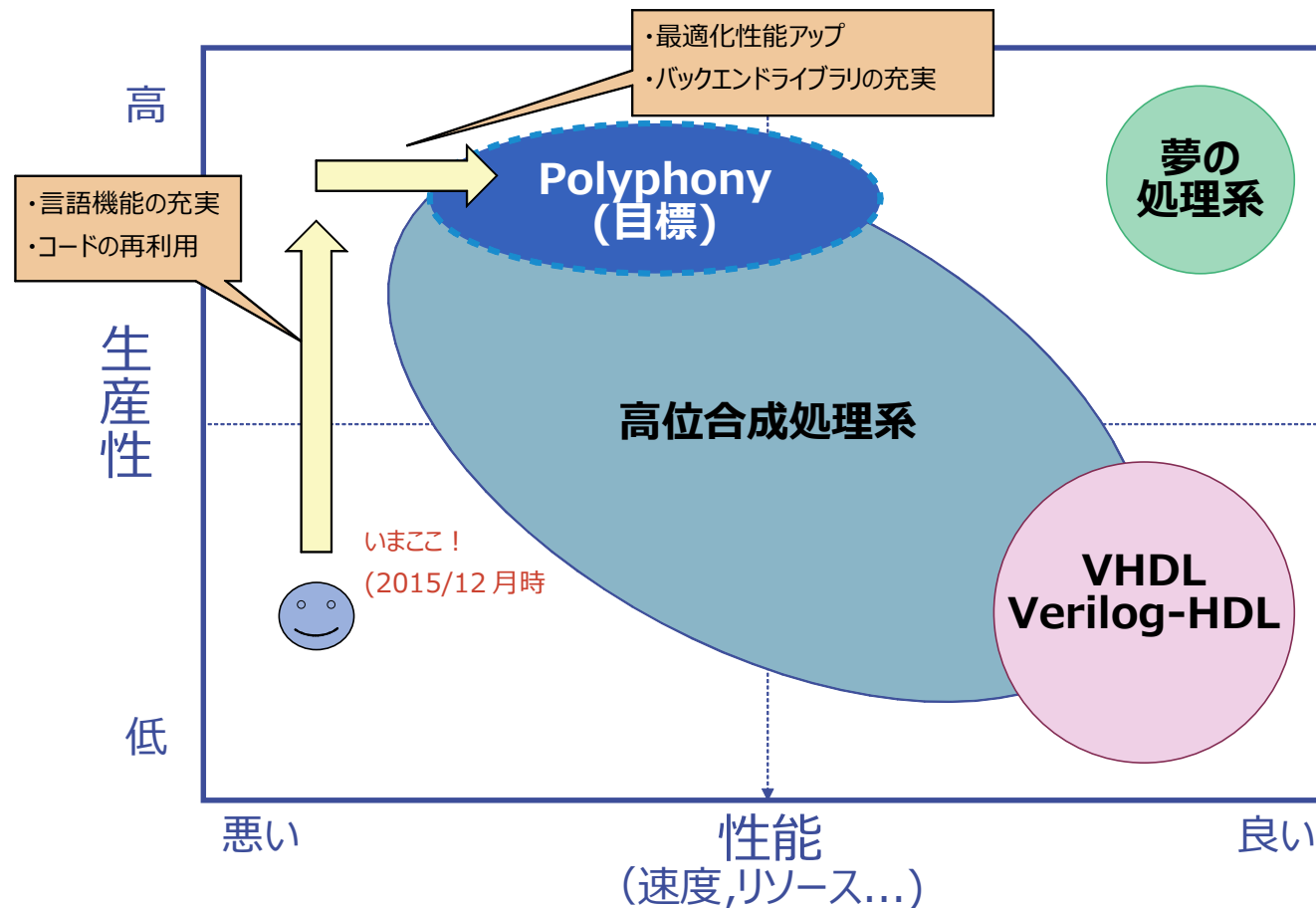
- 高位合成友の会 第3回(2015/12/08)のスライドより抜粋 -

- 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

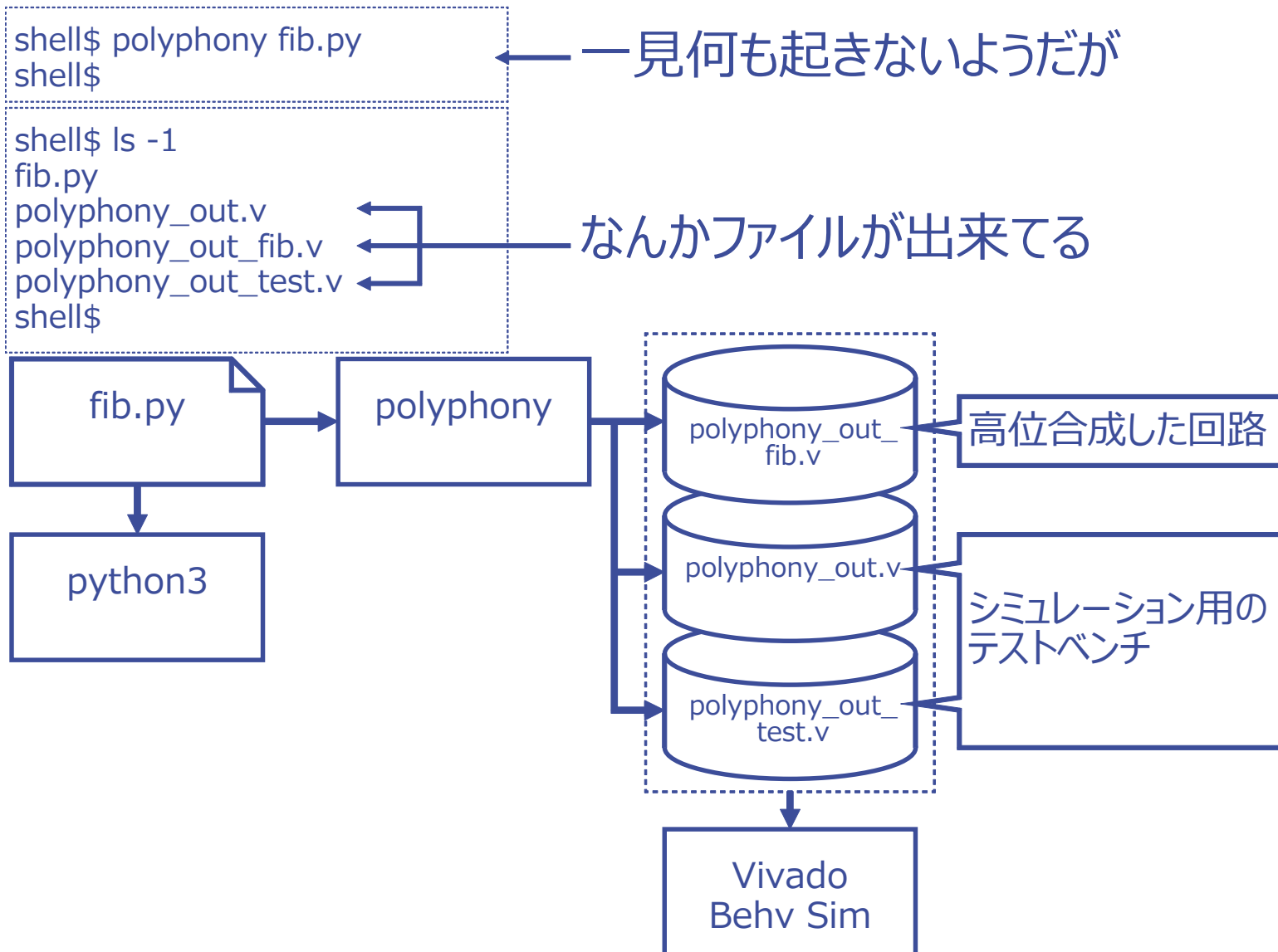
def fib(n):
    if n <= 0: return 0
    if n == 1: return 1
    r0 = 0
    r1 = 1
    for i in range(n-1):
        prev_r1 = r1
        r1 = r0 + r1
        r0 = prev_r1
    return r1

@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



Vivado で Behavioral Simulation

The screenshot displays the Vivado 2016.4 interface for a project named 'fibonacci'. The main window shows the 'Behavioral Simulation - Functional - sim_1 - test' results. The 'Sources' pane on the left lists the project files, including 'fib' (polyphony_out_fib.v) and 'test' (polyphony_out_test.v). The 'Messages' pane shows a warning. The 'Hierarchy' pane shows the simulation hierarchy. The 'Tcl Console' at the bottom displays the simulation results, showing the output of the 'fib' block for various input values.

Behavioral Simulation - Functional - sim_1 - test

Sources

- Design Sources (1)
 - fib (polyphony_out_fib.v)
- Constraints
- Simulation Sources (3)
 - sim_1 (3)
 - Verilog (1)
 - polyphony_out_fib.v
 - test (polyphony_out_test.v) (2)
 - array56 - BidirectionalSingle
 - fib_0 - fib (polyphony_out_fib.v)
 - fib (polyphony_out_fib.v)

Messages: 1 warning

Hierarchy | Libraries | Compile Order

Scope | **Sources**

Tcl Console

```
5920000: fib_0_in_n= 14, fib_0_out_0= 377
14=> 377
5940000: fib_0_in_n= 15, fib_0_out_0= 377
6710000: fib_0_in_n= 15, fib_0_out_0= 610
15=> 610
```

Type a Tcl command here

Untitled 1* | **polyphony_out_test.v**

Name	Value
clk	1
rst	0
fib_0_ready	1
fib_0_in_n[31:0]	00000003
fib_0_valid	1
fib_0_accept	0
fib_0_out_0[31:0]	00000001

Waveform

380.000 ns

400 ns

500 ns

polyphony_out_fib.v の Interface(1)

polyphony_out_fib.v

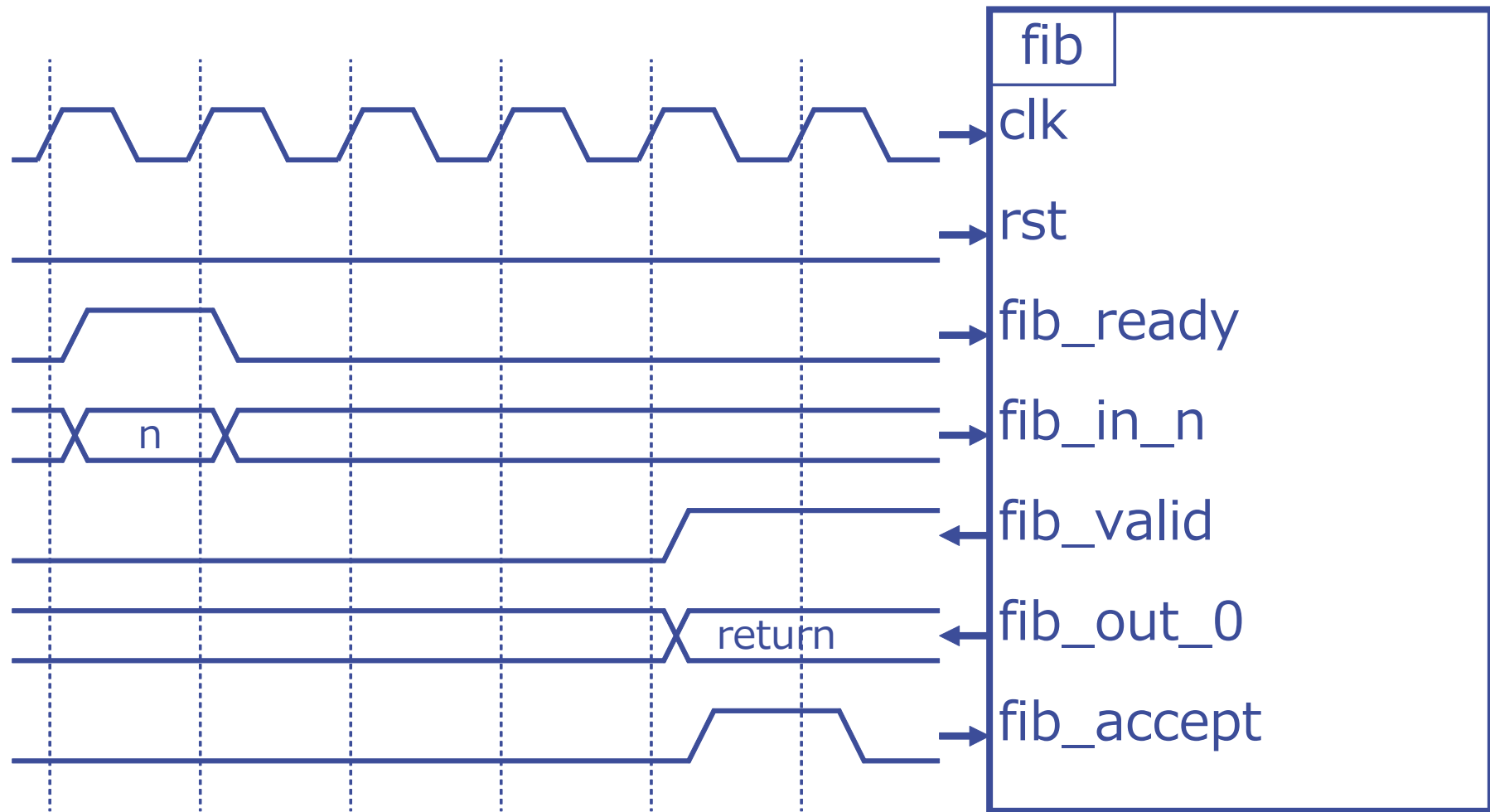
```
module fib
(
    input wire clk,
    input wire rst,
    input wire fib_ready,
    input wire fib_accept,
    output reg fib_valid,
    input wire signed [31:0] fib_in_n,
    output reg signed [31:0] fib_out_0
);

//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
```

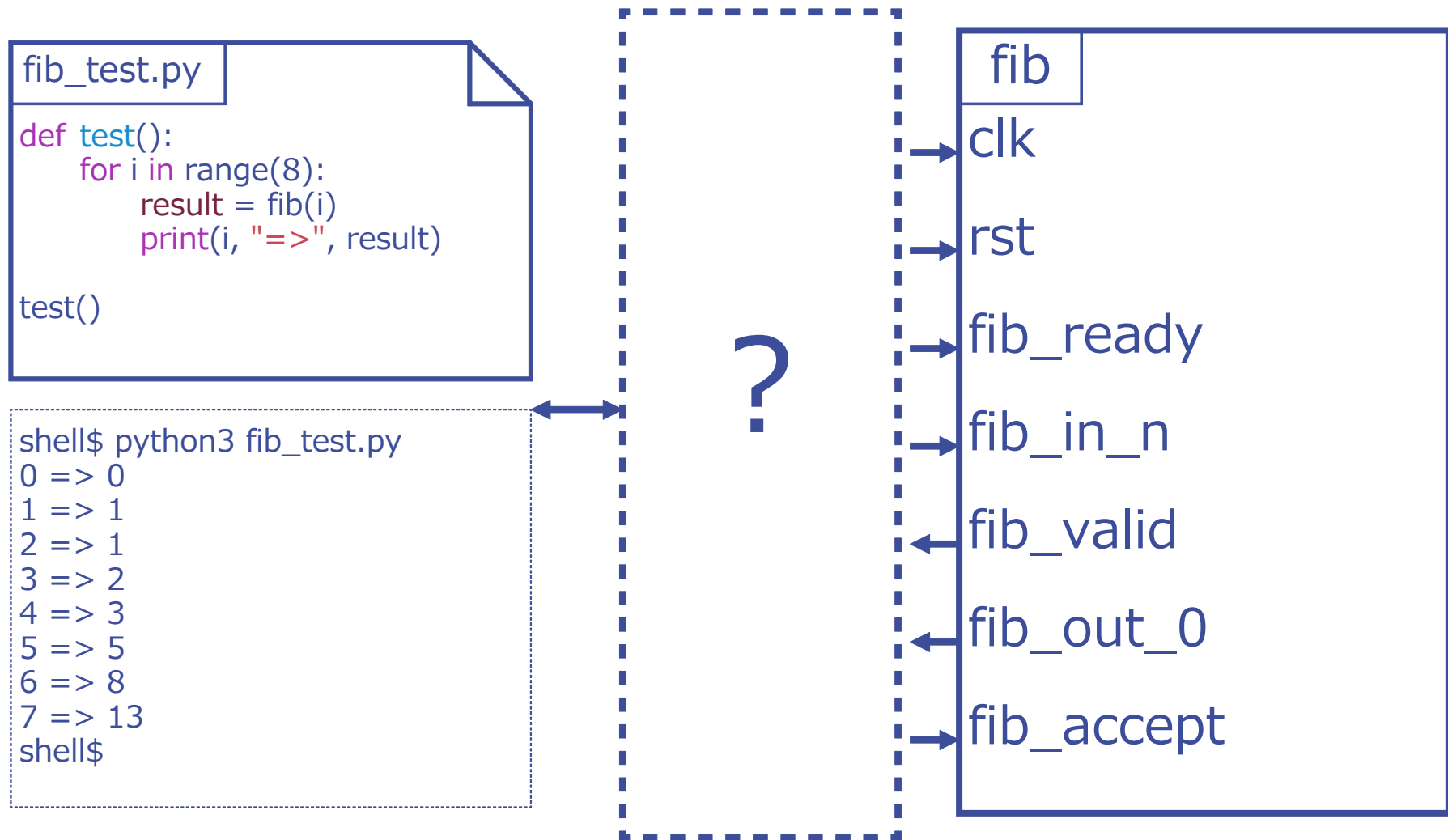
fib

→ clk
→ rst
→ fib_ready
→ fib_in_n
← fib_valid
← fib_out_0
→ fib_accept

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))

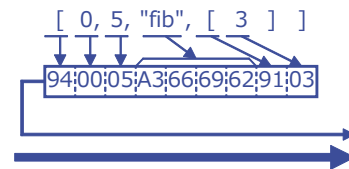
def fib(n):
    return cli.call('fib', n)

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

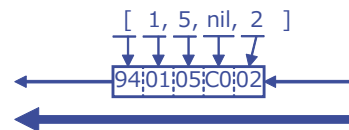
test()
```

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

Request Message



Response Message



fib_server.py

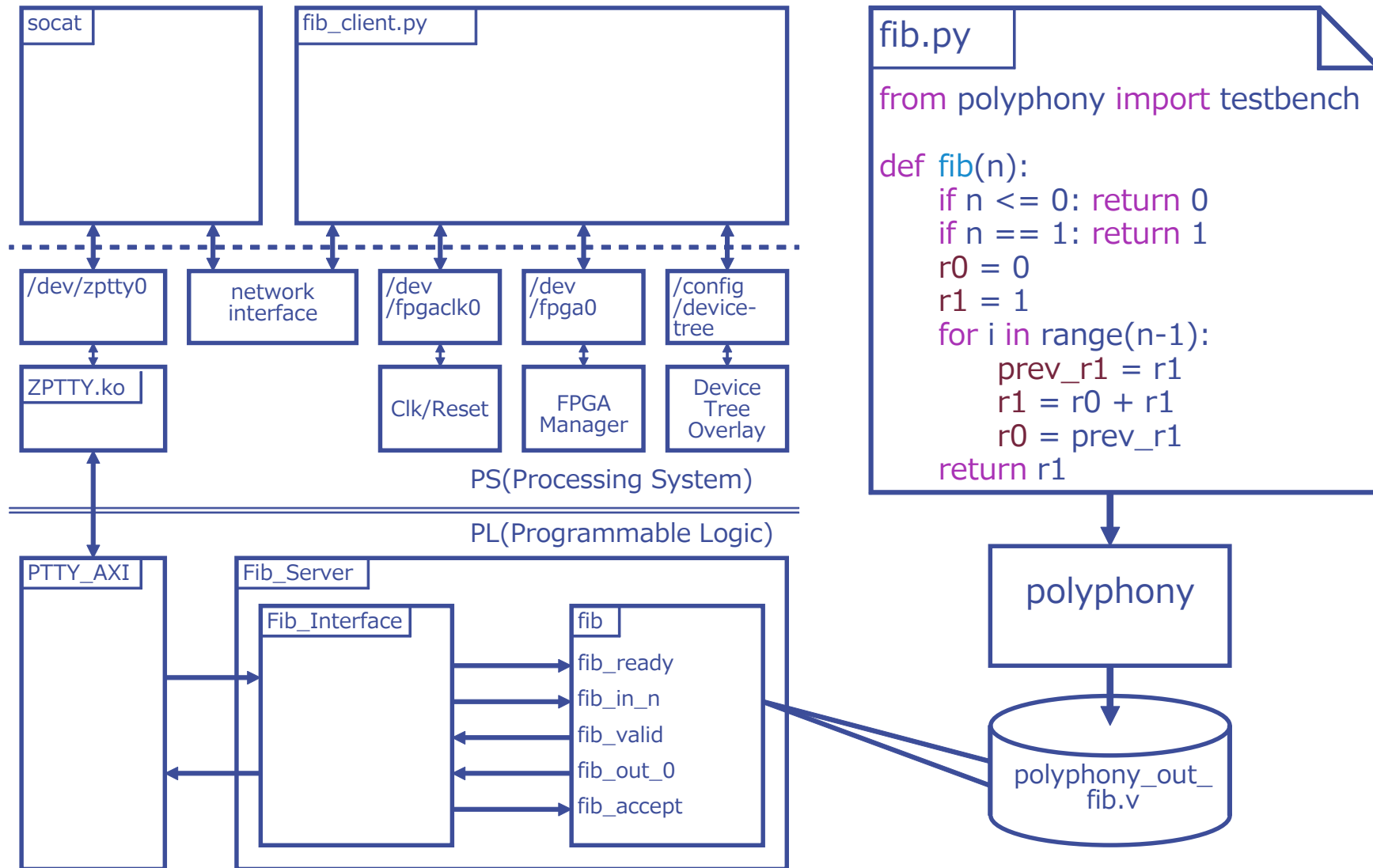
```
import msgpackrpc

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

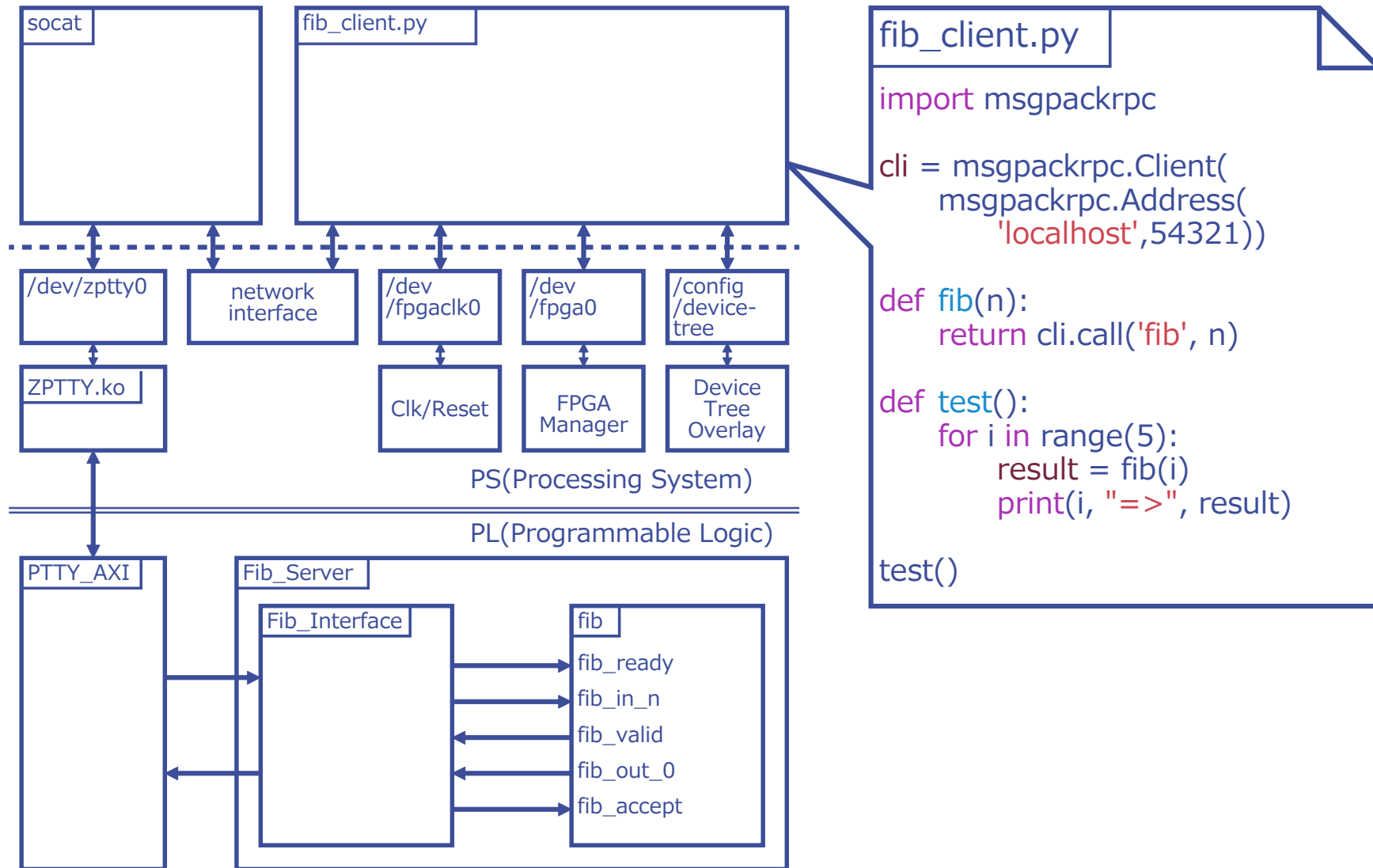
svr=msgpackrpc.Server(FibServer())
svr.listen(msgpackrpc.Address(
    'localhost', 18800))
svr.start()
```

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

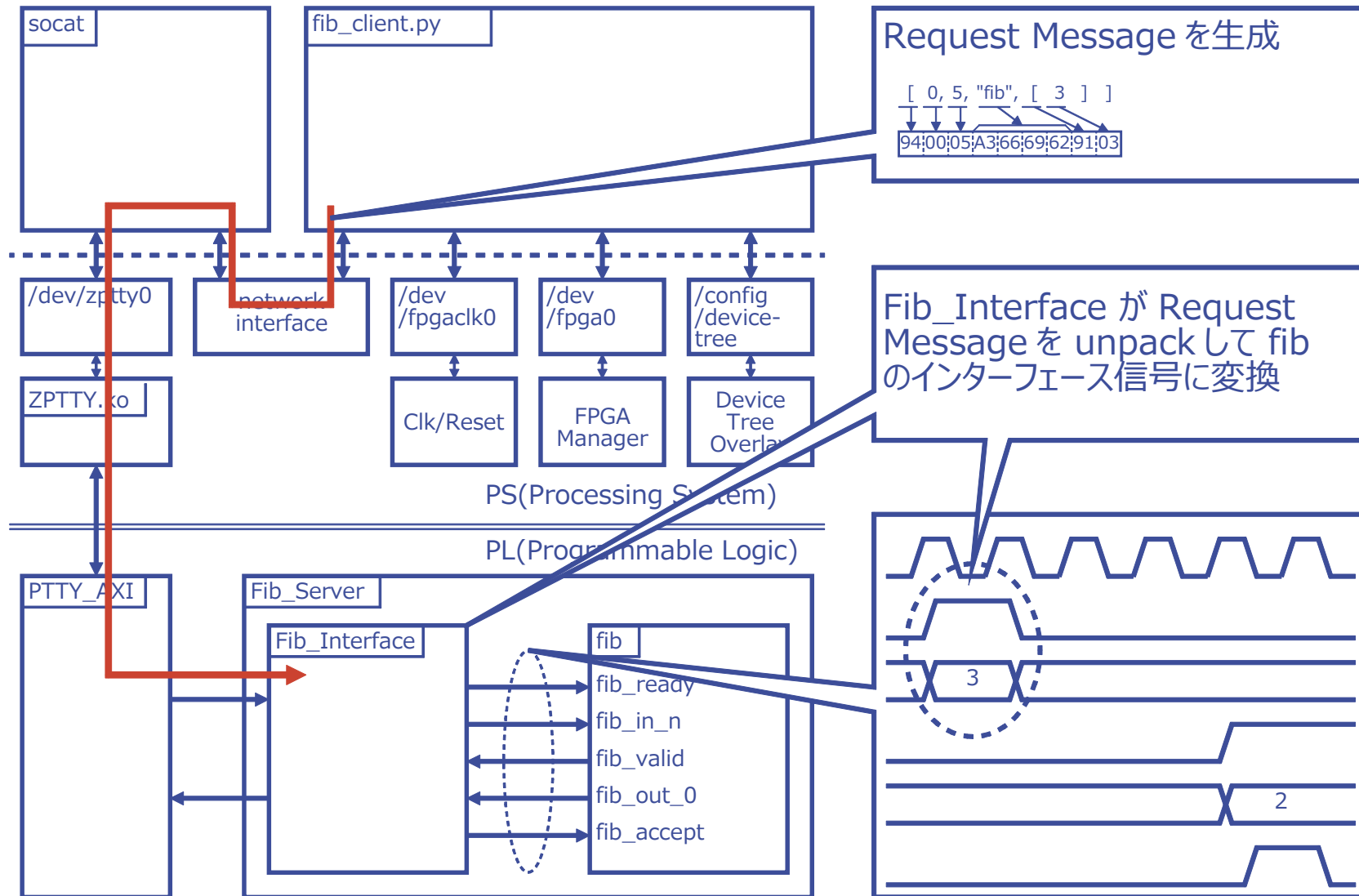
MessagePack-RPC Server on FPGA-SoC-Linux(1)



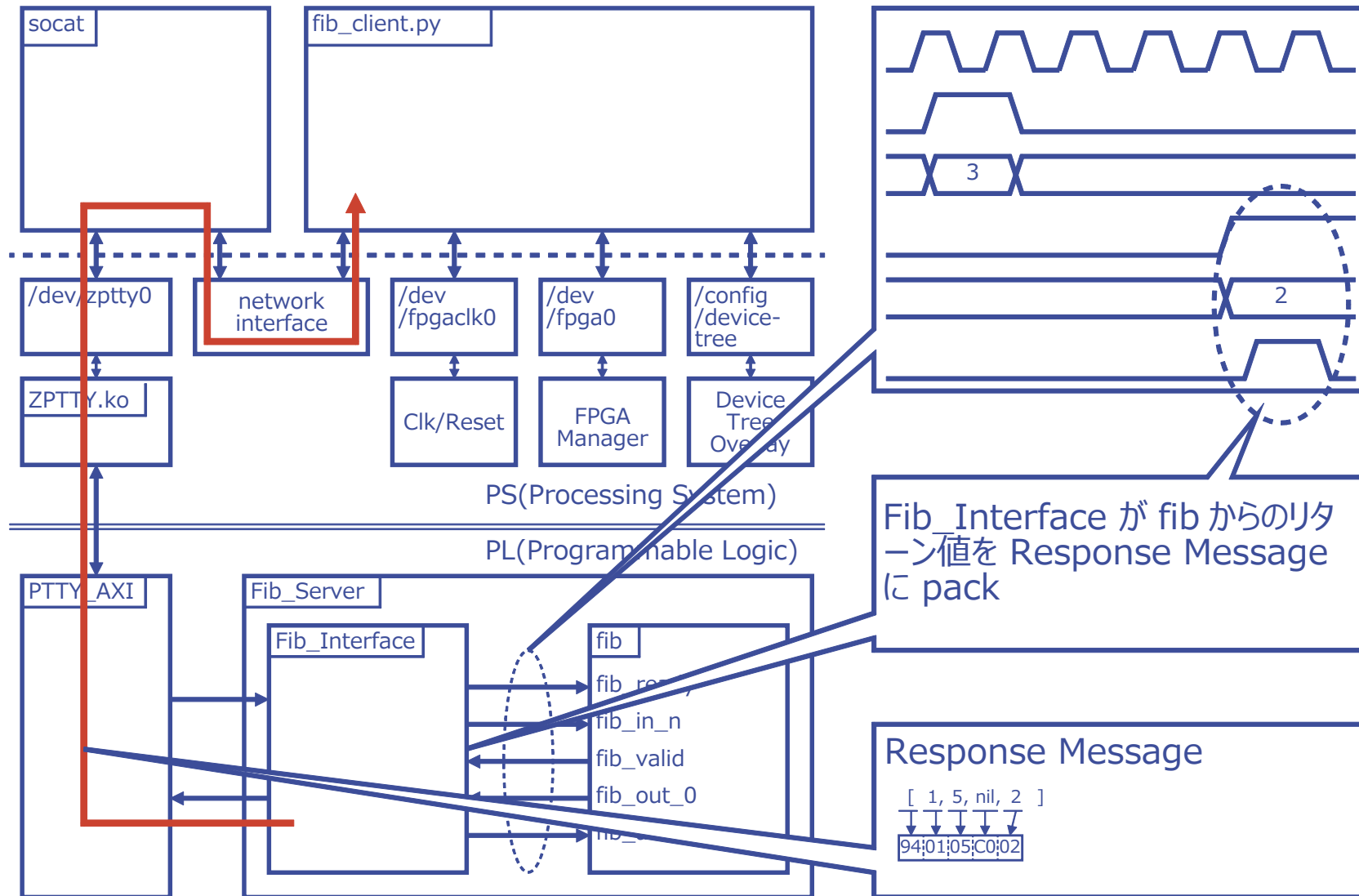
MessagePack-RPC Server on FPGA-SoC-Linux(2)



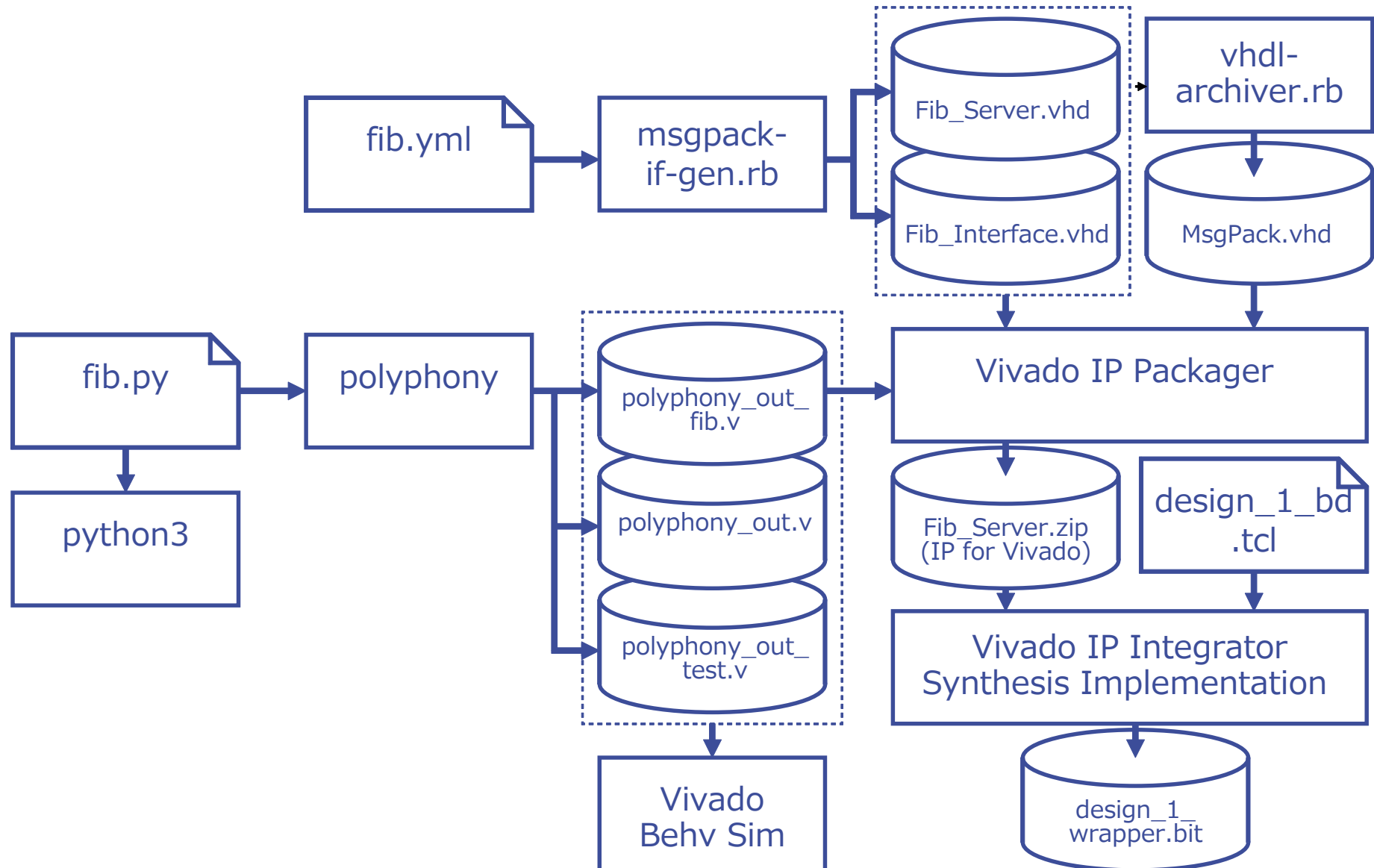
MessagePack-RPC Server on FPGA-SoC-Linux(3)



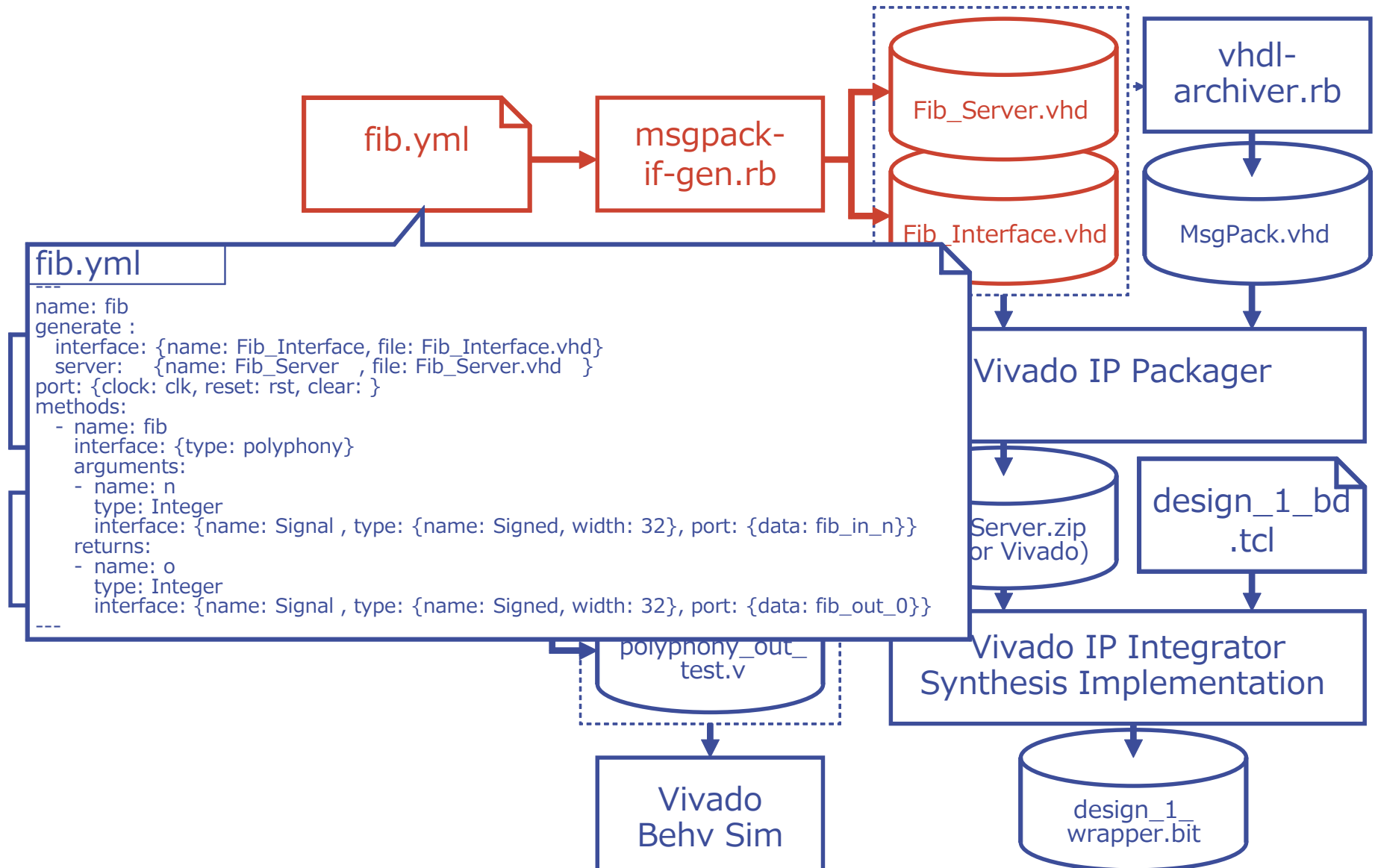
MessagePack-RPC Server on FPGA-SoC-Linux(4)



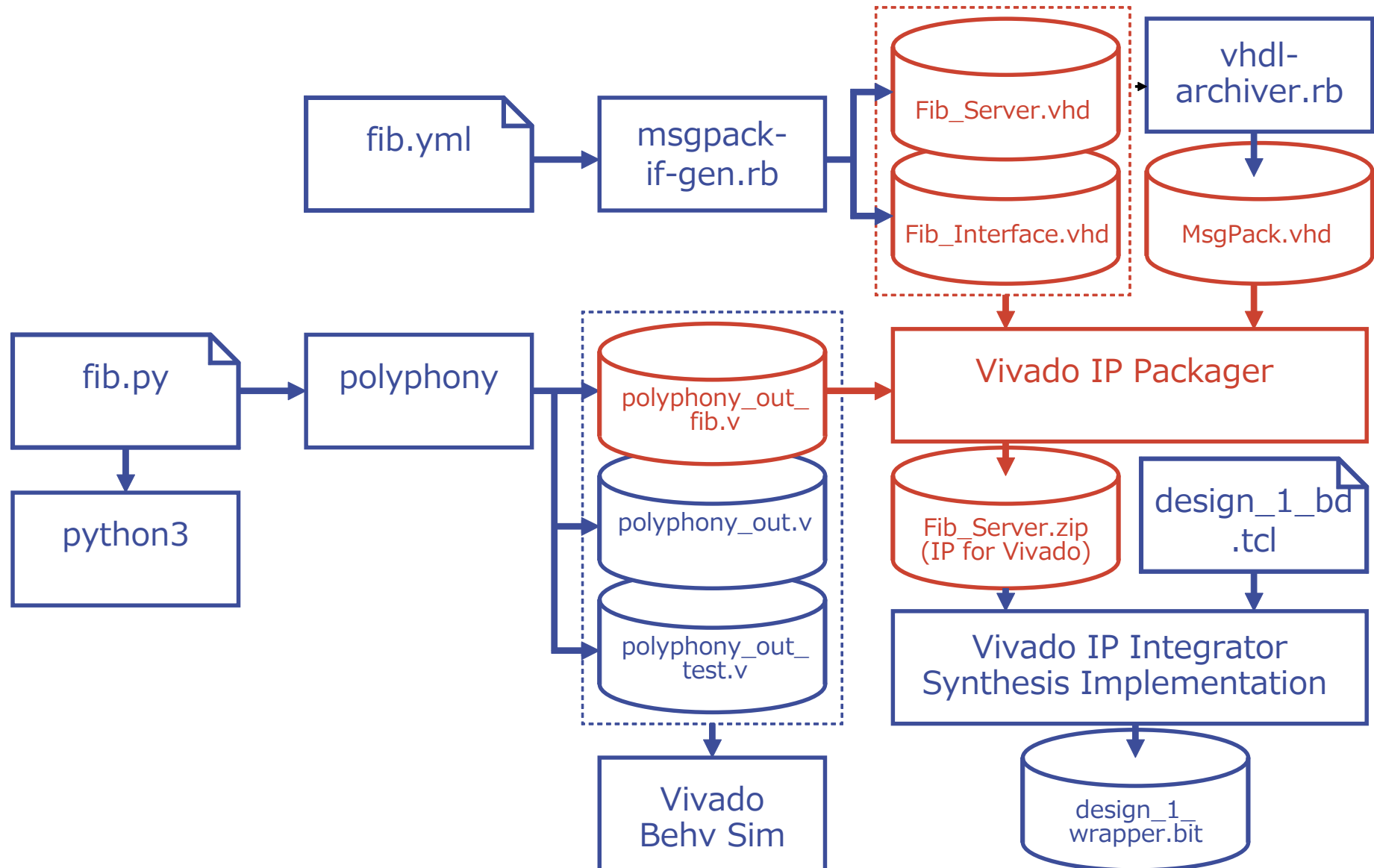
Design Flow



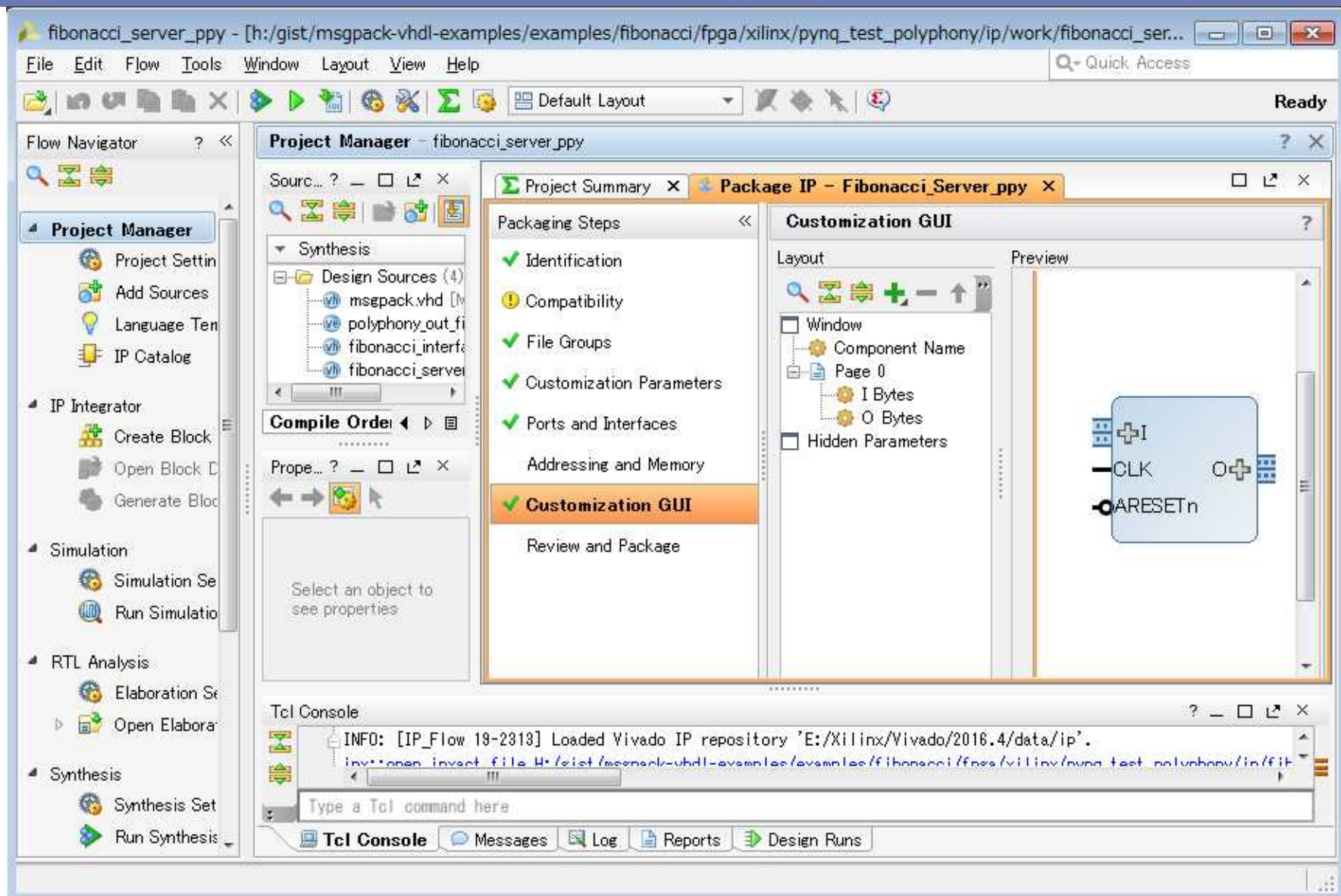
msgpack-if-gen



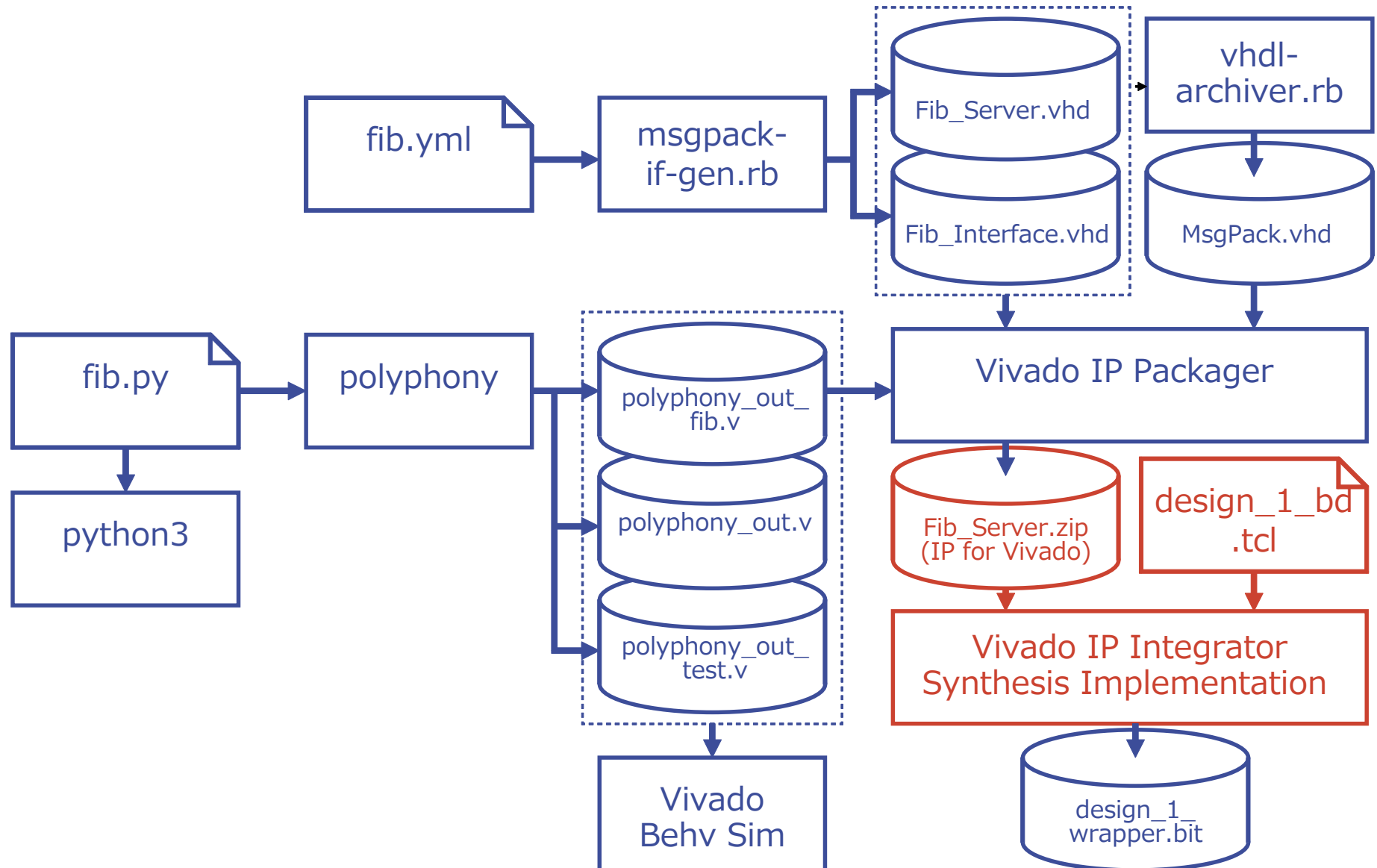
Vivado IP Packager (1)



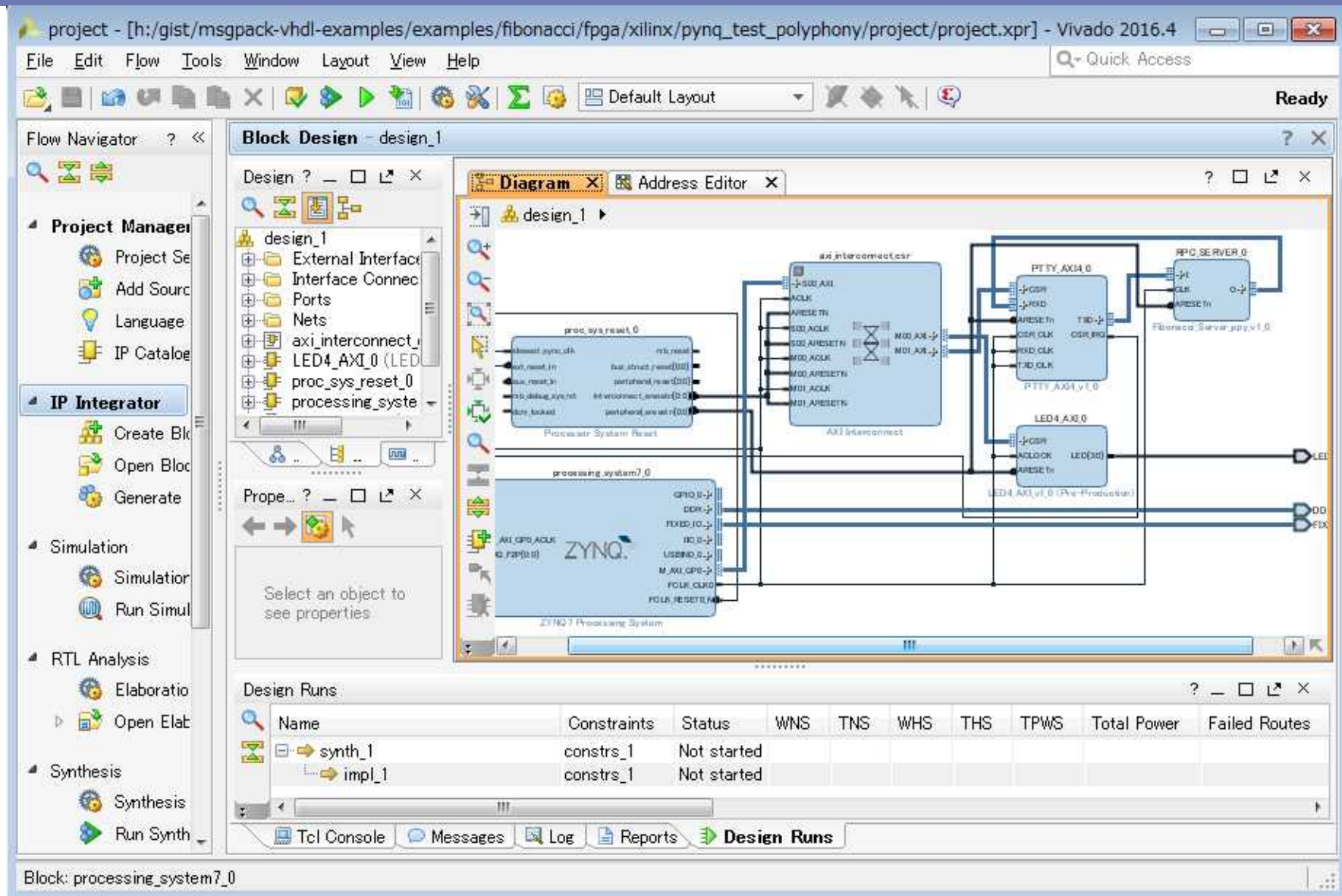
Vivado IP Packager (2)



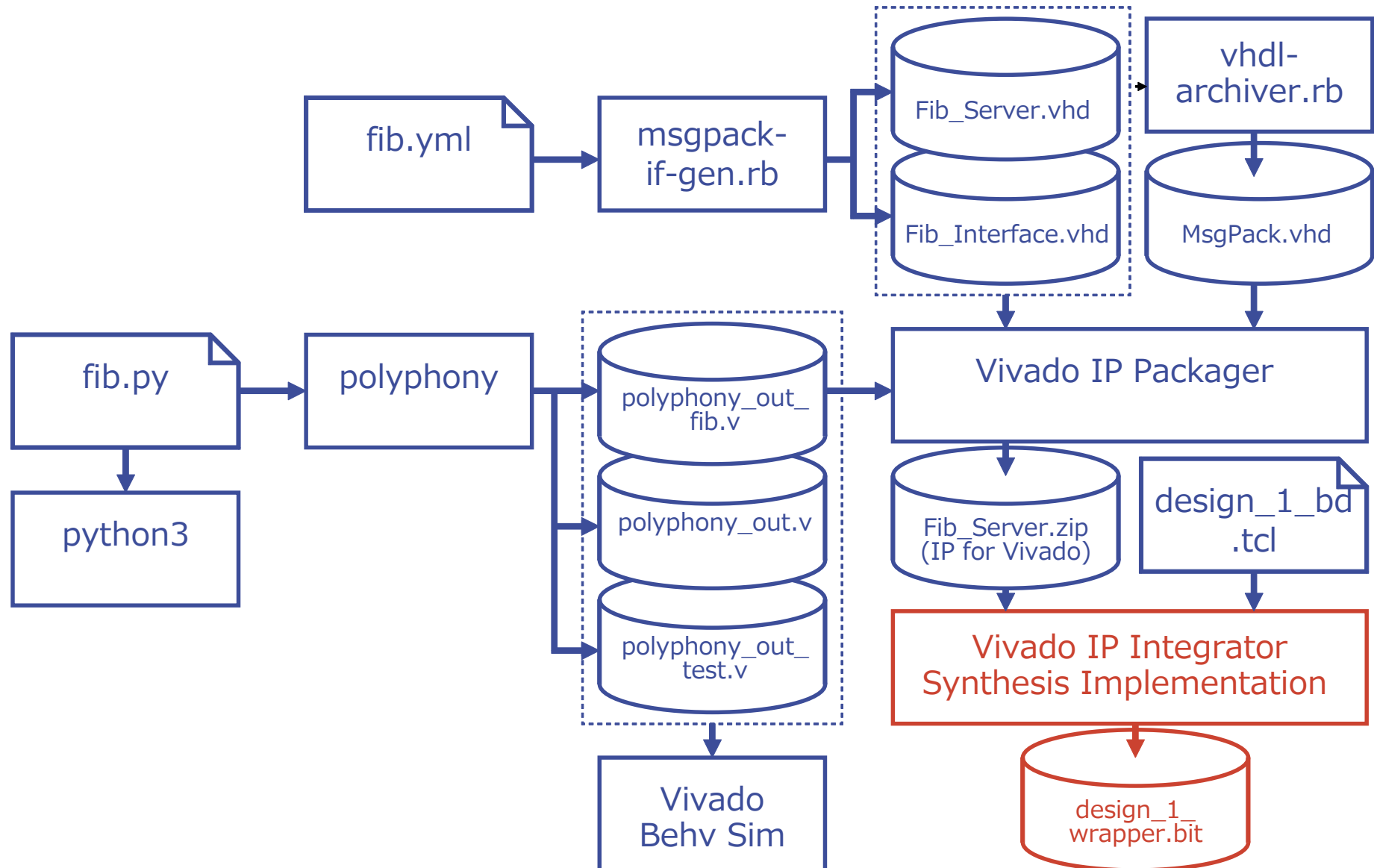
Vivado IP Integrator (1)



Vivado IP Integrator (2)



Vivado Synthesis & Implementation (1)



Vivado Synthesis & Implementation (2)

The screenshot displays the Vivado 2016.4 IDE interface. The top menu bar includes File, Edit, Flow, Tools, Window, Layout, View, and Help. The title bar shows the project path: project - [h:/gist/msgpack-vhdl-examples/examples/fibonacci/fpga/xilinx/pynq_test_polyphony/project/project.xpr] - Vivado 2016.4. The main workspace is divided into several panes:

- Flow Navigator:** Shows the project flow with options like Run Synth, Open Synt, and Implementation. The Implementation section is expanded, showing options like Implement, Run Imple, and Implement.
- Netlist:** Displays the netlist for the design, showing design_1_wrapper, Nets (138), Leaf Cells (4), and design_1_i (design_1).
- Project Summary:** Shows the utilization and power results for the implementation.
- Design Runs:** A table showing the status of various design runs.

Utilization - Post-Implementation

Resource	Utilization (%)
LUT	11%
LUTRAM	1%
FF	3%
BRAM	3%
IO	3%
BUFG	3%

Power

Parameter	Value
Total On-Chip Power:	1.4
Junction Temperature:	41
Thermal Margin:	43
Effective θ_{JA} :	11
Power supplied to off-chip devices:	0 V
Confidence level:	Me

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS
synth_1 (active)	constrs_1	synth_design Complete!				
impl_1	constrs_1	phys_opt_design (Post-Route) Complete!	0.359	0.000	0.031	0.000
Out-of-Context Module Runs						
design_1		Submodule Runs Complete				

The bottom status bar shows the current view is Design, with other views like Power, Timing, Methodology, and DRC available.

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,echo=0 &
[2] 2145
2017/03/01 09:34:13 socat[2145] N listening on AF=2 0.0.0.0:54321
```

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

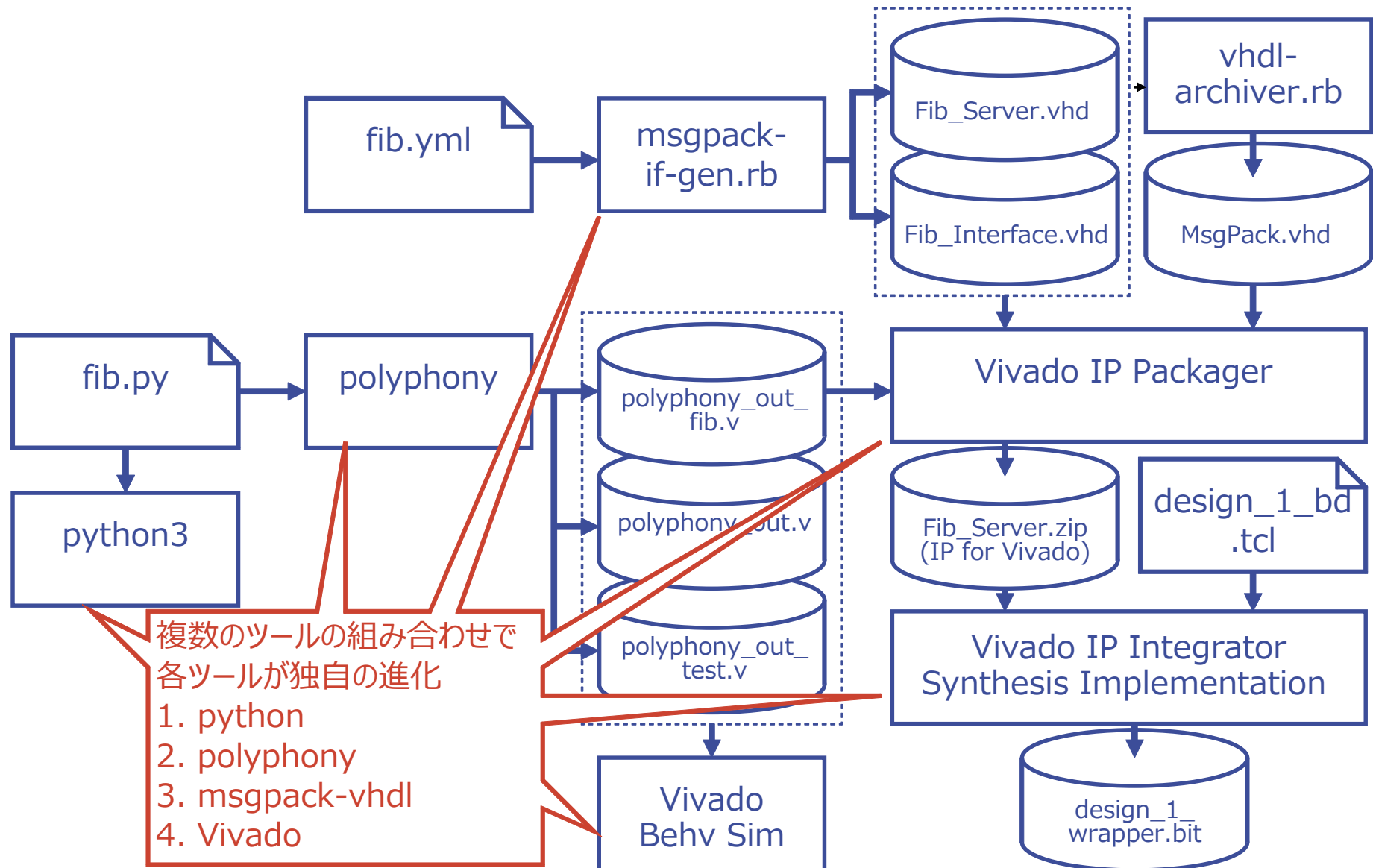
- ・ すぐに切れそうな Tool Chain
- ・ msgpack-if-gen.rb 用の設定ファイル(fib.yml)が必要

2. MessagePack-RPC のオーバーヘッド

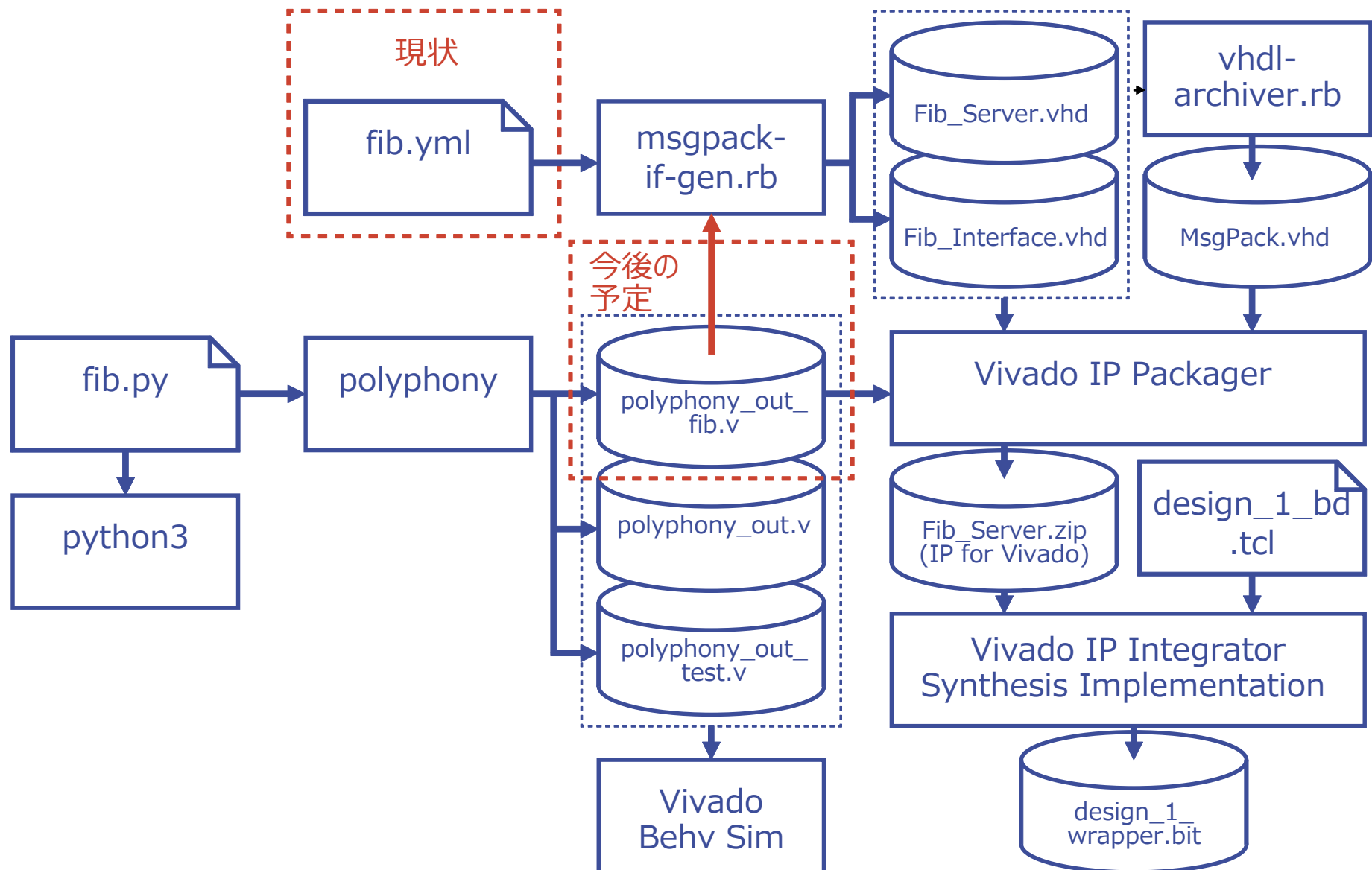
- ・ レイテンシーの増大
- ・ スループットの悪化

3. Polyphony の言語的な制約

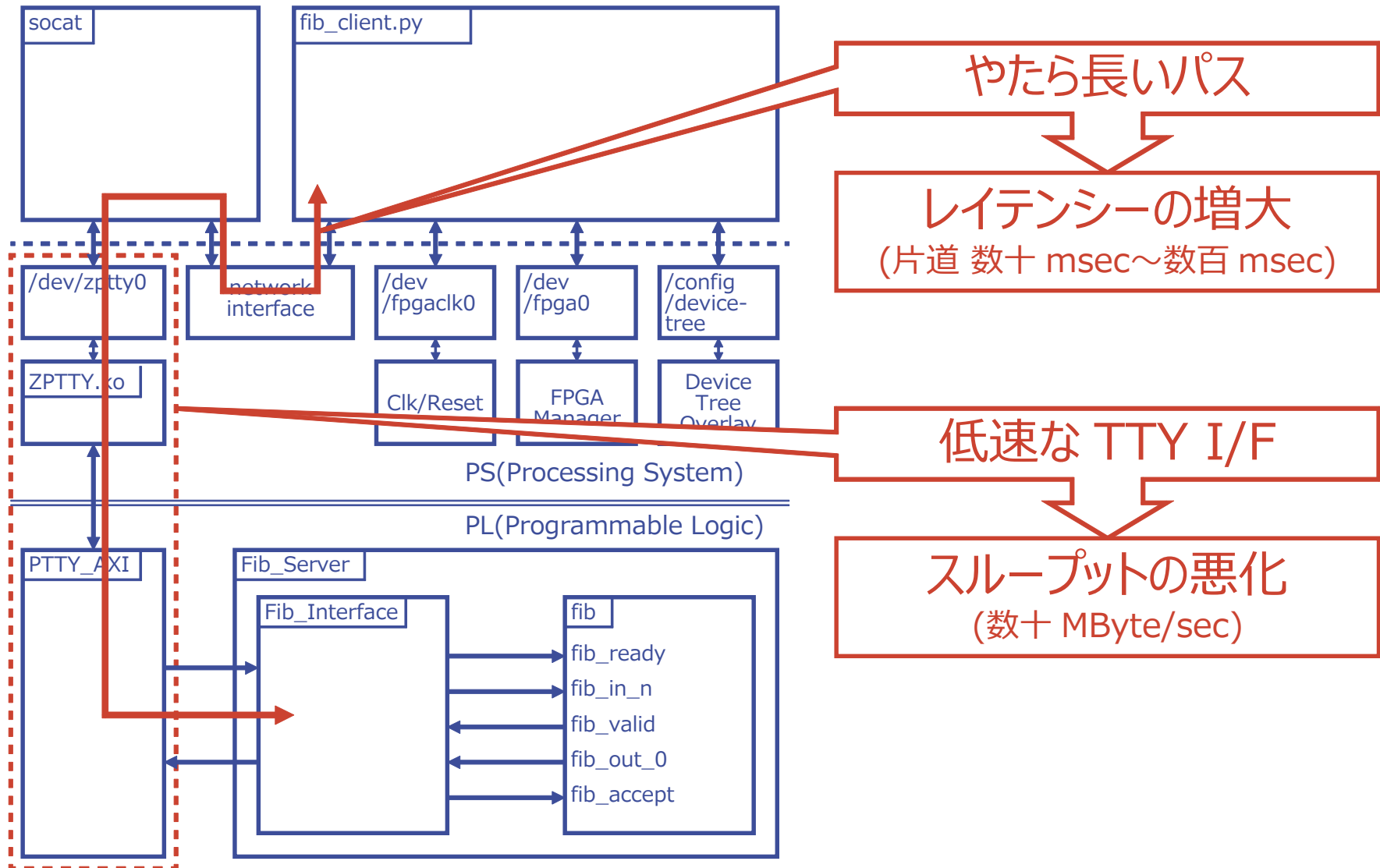
課題 1.1 Design Flow が複雑 - すぐに切れそうな Tool Chain



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)
- 最近では 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。

参考 - PYNQ-Festival

- 以下の資料やバイナリイメージ
 - 本日説明したスライド
 - u-boot
 - Linux Kernel 4.8.17 (FPGA manager+Device Tree Overlay)
 - Debian 8 Root File System
 - design_1_wrapper.bit (PYNQ-Z1 用 bitstream)
 - fibonacci_client.ipynb

はこちらにあります。

<https://github.com/ikwzm/PYNQ-Festival>

参考 - Polyphony

- 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>

参考 - MessagePack-VHDL

- MessagePack-RPC を使って FPGA を制御 @Qiita
<http://qiita.com/ikwzm/items/2644c6e50a7049c75d49>
- github で公開中
<https://github.com/ikwzm/msgpack-vhdl>
<https://github.com/ikwzm/msgpack-vhdl-examples>

参考 - FPGA-SoC-Linux

- FPGA+SoC+Linux+Device Tree Overlay+FPGA Manager(PYNQ-Z1 対応) @Qiita
<http://qiita.com/ikwzm/items/d6fdaa859906252fd8b0>
- github で公開中
<https://github.com/ikwzm/FPGA-SoC-Linux>