Caravel FPGA Introduction

Willy Chiang

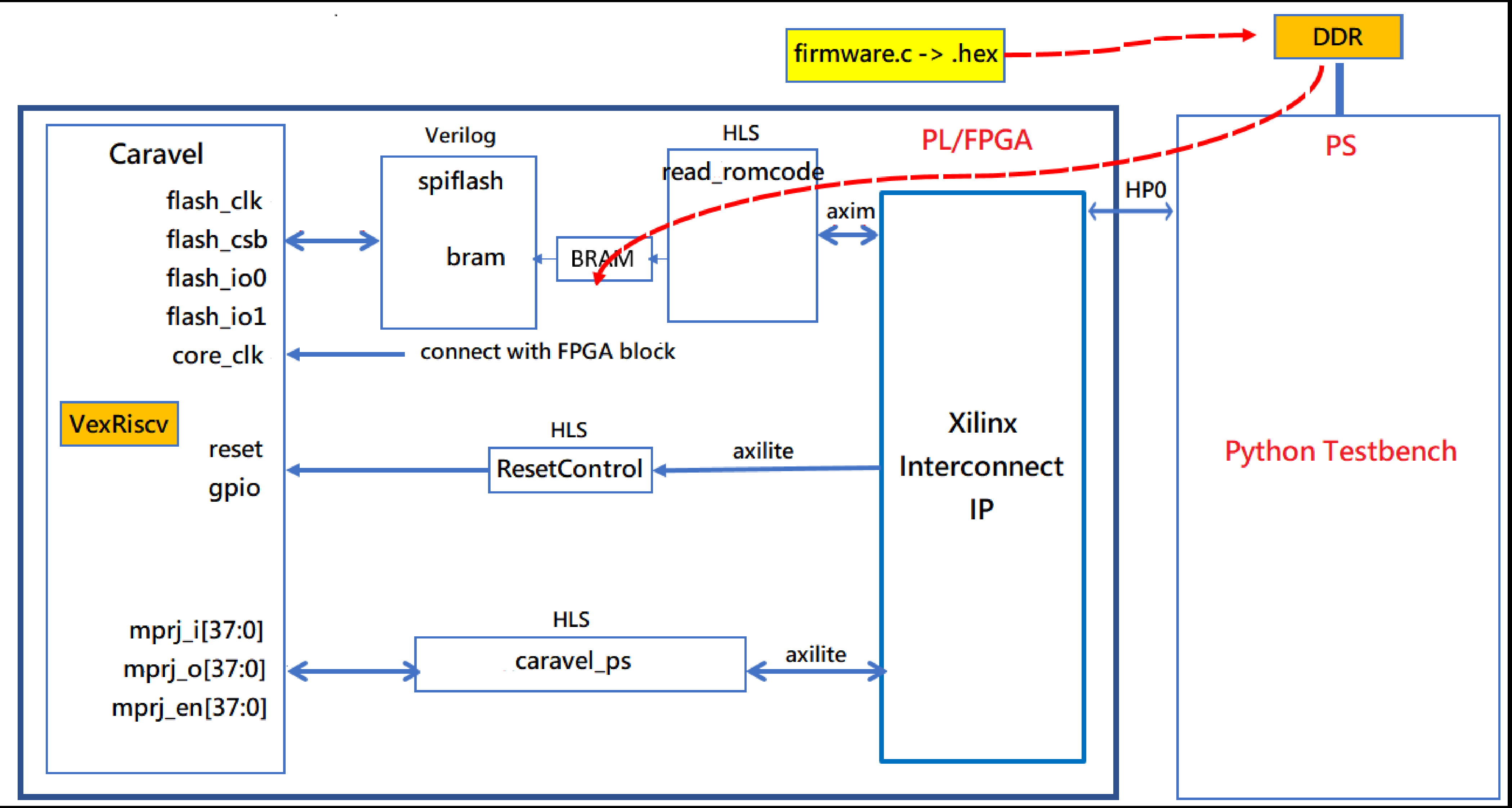
Outline

- Caravel FPGA Architecture Overview
- Jupyter Notebook Introduction
- Code Trace: caravel_fpga.ipynb

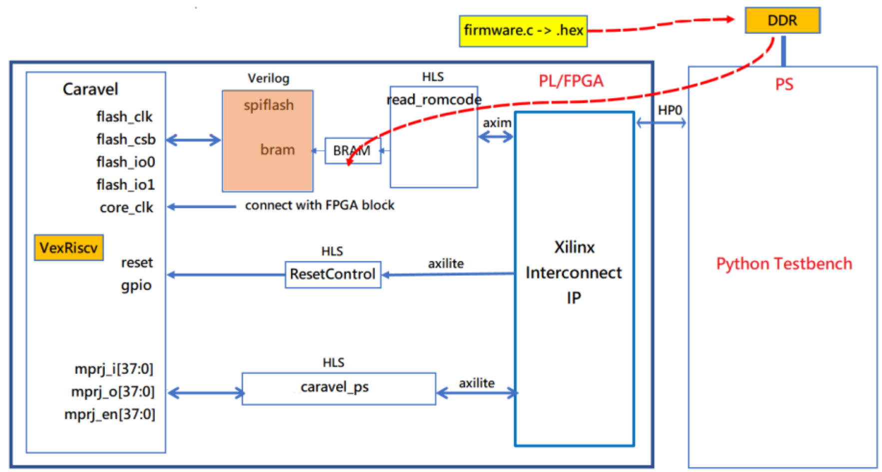
Outline

- Caravel FPGA Architecture Overview
- Jupyter Notebook Introduction
- Code Trace: caravel_fpga.ipynb

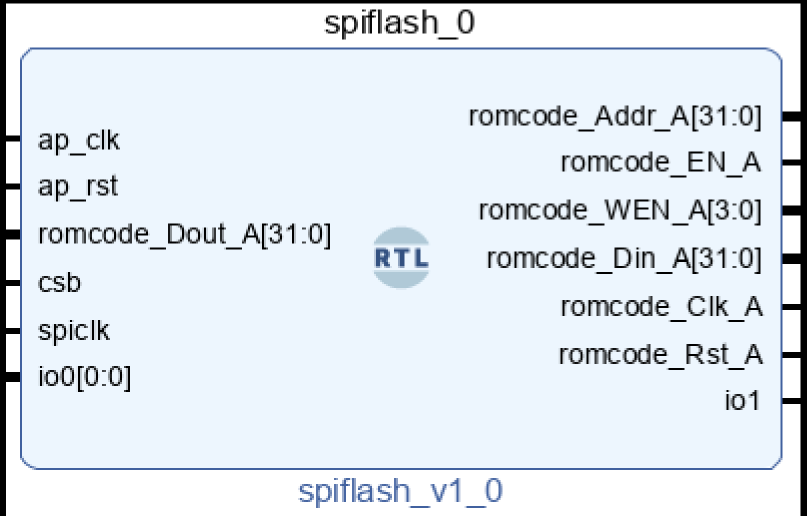
Caravel FPGA Block Diagram



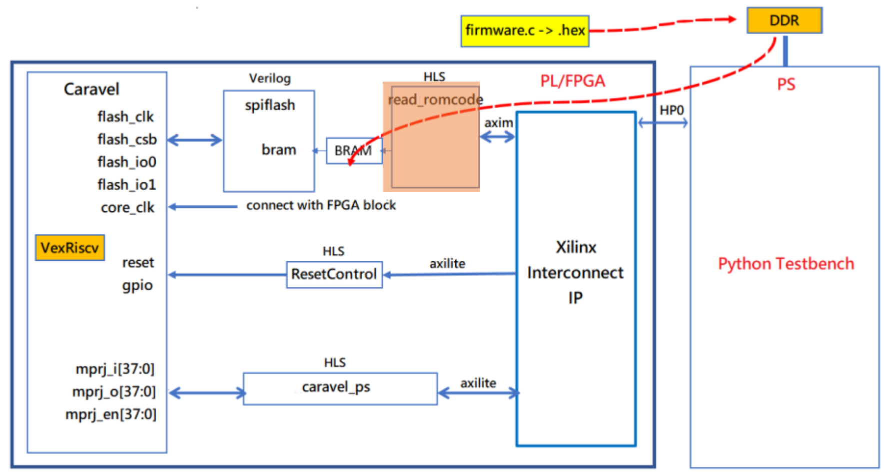
Spiflash



- Implement SPI slave device, only support read command (0x03)
- Return data from BRAM to Caravel

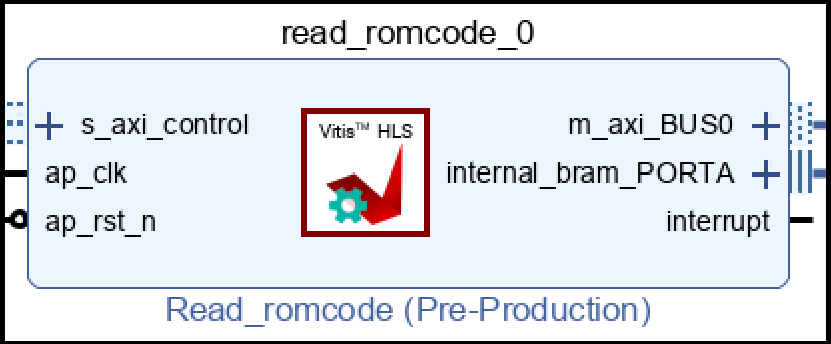


Read_romcode

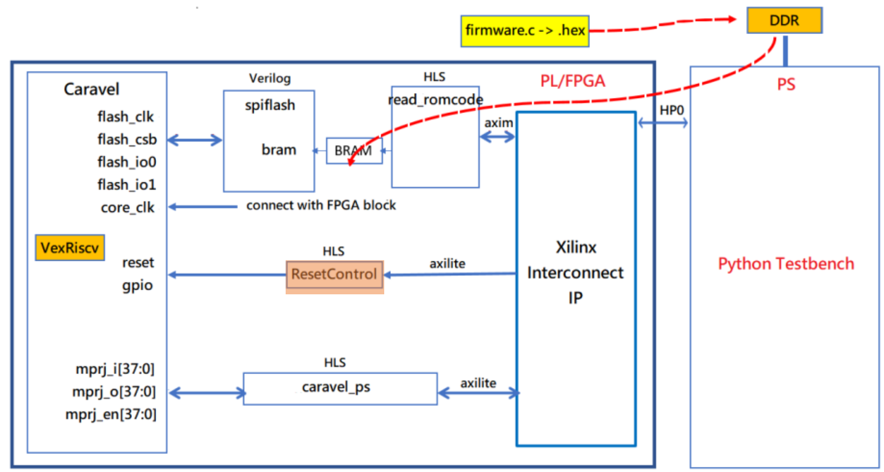


- Copy PS dram buffer to BRAM base on the size of binary file.
- Limit the BRAM size to 8K
- Implement by HLS and export IP for Vivado project usage.

```
# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)
```

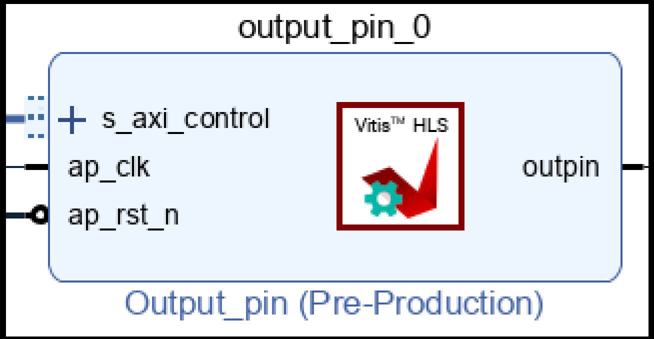


Output_pin (Reset Control)

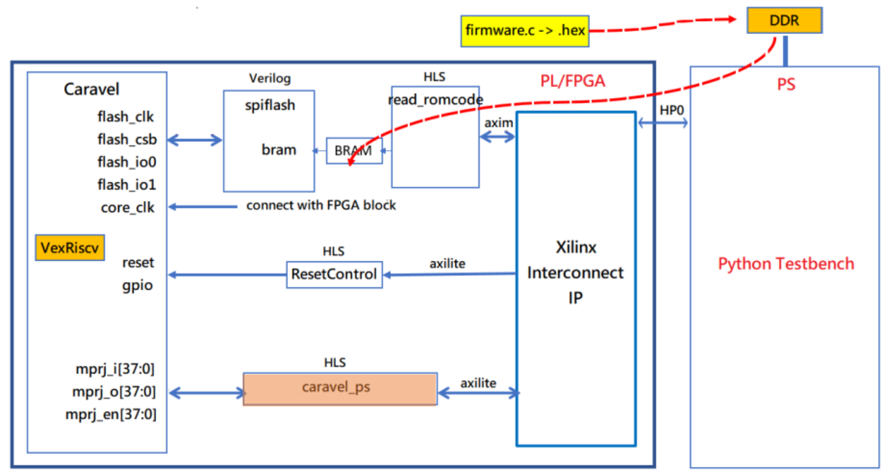


- Output 1 or 0 signal, which used to assert/de-assert Caravel reset pin
- Provide AXI-LITE interface for PS CPU to control the output.
- Implement by HLS and export IP for Vivado project usage.

```
# Release Caravel reset
# 0x10 : Data signal of outpin_ctrl
#        bit 0  - outpin_ctrl[0] (Read/Write)
#        others - reserved
```

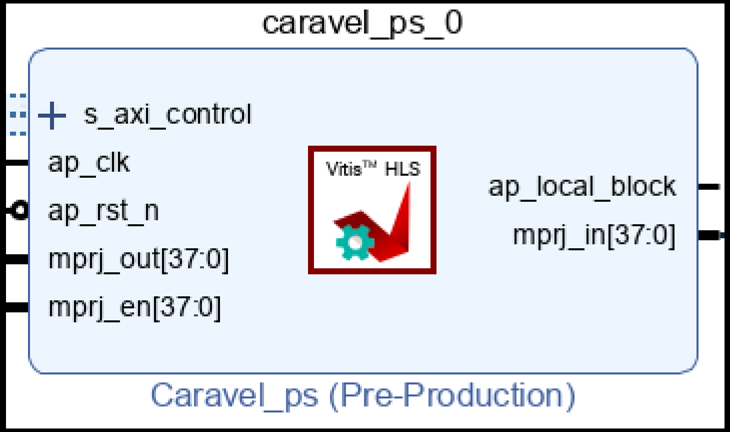


Caravel_ps



- Provide AXI-Lite interface for PS CPU to read the MPRJ_IO/OUT/EN bits
- Implement by HLS and export IP for Vivado project usage.

```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved
```

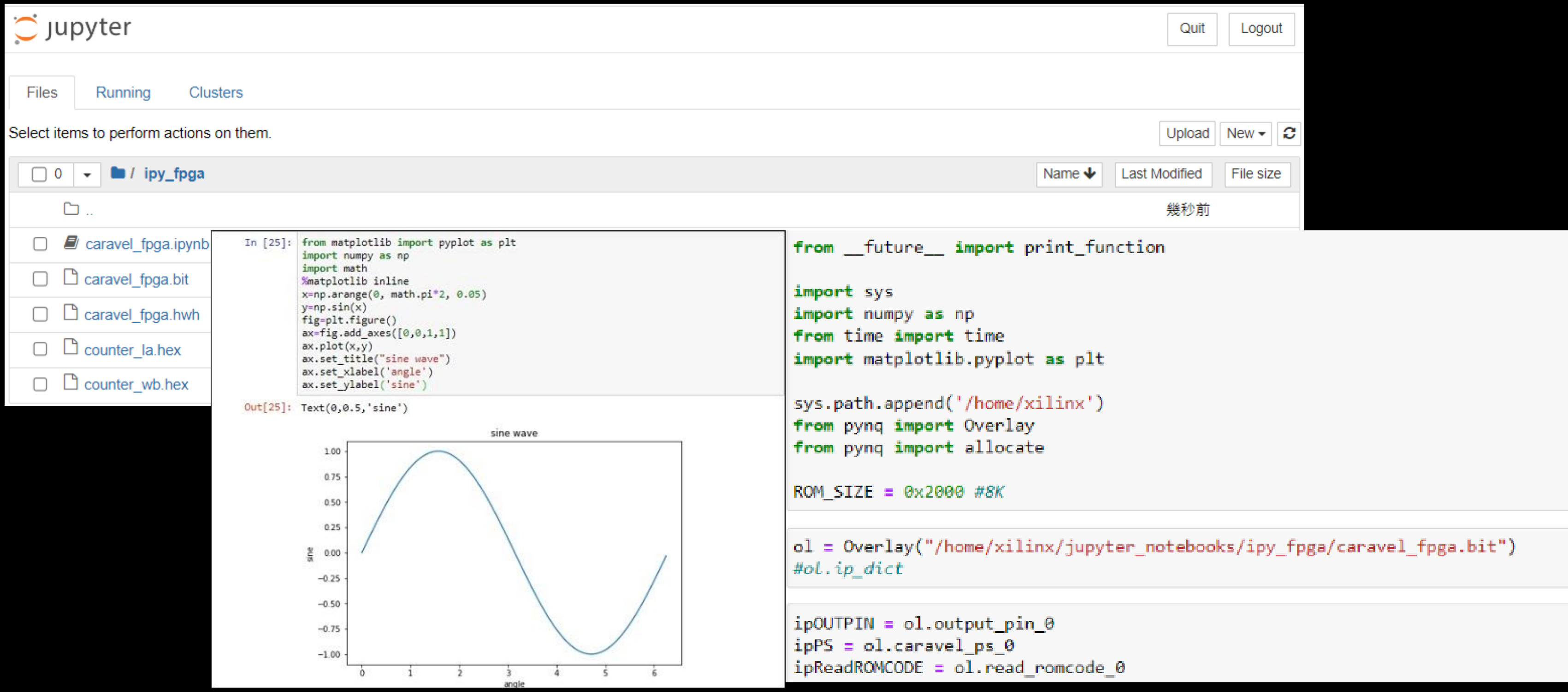


Outline

- Caravel FPGA Architecture Overview
- Jupyter Notebook Introduction
- Code Trace: caravel_fpga.ipynb

What’s the Jupyter Notebooks

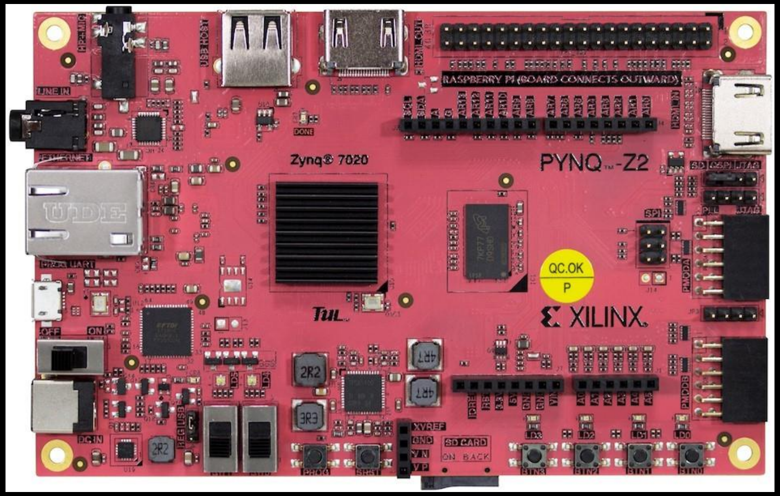
- https://pynq.readthedocs.io/en/v2.4/jupyter_notebooks.html
- The Jupyter Notebook is an **interactive computing environment** that enables users to author notebook documents that include:
  - ✓Live code
  - ✓Interactive widgets
  - ✓Plots
  - ✓Narrative text
  - ✓Equations
  - ✓Images
  - ✓Video



Notebook Kernels

- The Notebook supports a range of different programming languages.
- PYNQ is written in **Python**, which is the default kernel for Jupyter Notebook, and the only kernel installed for Jupyter Notebook in the PYNQ distribution.

XUP PYNQ-Z2



PYNQ Libraries

- https://pynq.readthedocs.io/en/v2.4/pynq_libraries.html#

- PYNQ Libraries
  - IP
    - Audio
    - AxiGPIO
    - AxiIIC
    - DMA
    - Logictools
    - Video
  - IOPs
    - Arduino
    - Grove
    - Pmod
    - RPi
  - PynqMicroBlaze
    - MicroBlaze Subsystem
    - Microblaze RPC
    - Microblaze Library
  - PS/PL interface
    - Interrupt
    - MMIO
    - PS GPIO
    - Xlnk
  - PS control
    - PMBus
  - PL control
    - Overlay
    - PL and Bitstream classes
    - Microblaze Library

Outline

- Caravel FPGA Architecture Overview
- Jupyter Notebook Introduction
- Code Trace: caravel_fpga.ipynb

caravel_fpga.ipynb

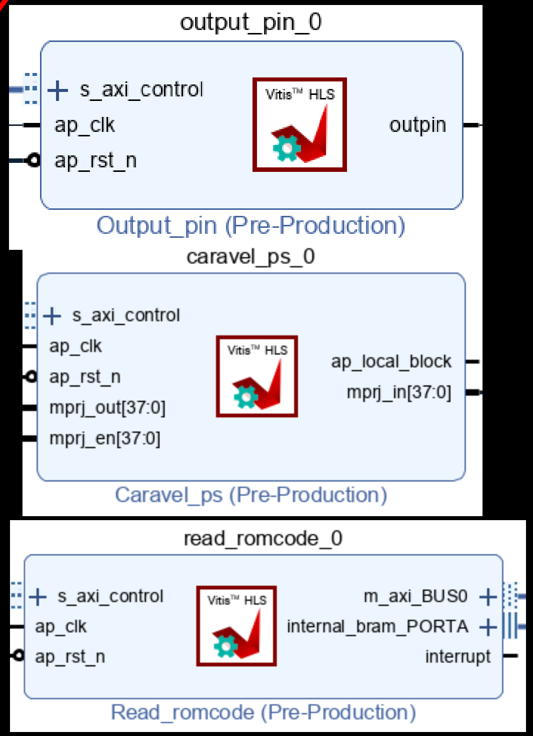
- https://github.com/bol-edu/caravel-soc_fpga/blob/main/vivado/jupyter_notebook/caravel_fpga.ipynb

```python
from __future__ import print_function

import sys
import numpy as np
from time import time
import matplotlib.pyplot as plt

sys.path.append('/home/xilinx')
from pynq import Overlay
from pynq import allocate

ROM_SIZE = 0x2000 #8K
```

```python
ol = Overlay("/home/xilinx/jupyter_notebooks/ipy_fpga/caravel_fpga.bit")
#ol.ip_dict
```

```python
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
```

1. Load new overlay (bitstream)
2. Instance IPs by navigating the overlay object



caravel_fpga.ipynb

```python
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

# Allocate dram buffer will assign physical address to ip ipReadROMCODE
npROM = allocate(shape=(ROM_SIZE >> 2,), dtype=np.uint32)

# Initial it by 0
for index in range (ROM_SIZE >> 2):
    npROM[index] = 0

npROM_index = 0
npROM_offset = 0
fiROM = open("counter_la.hex", "r+")
#fiROM = open("counter_wb.hex", "r+")
```

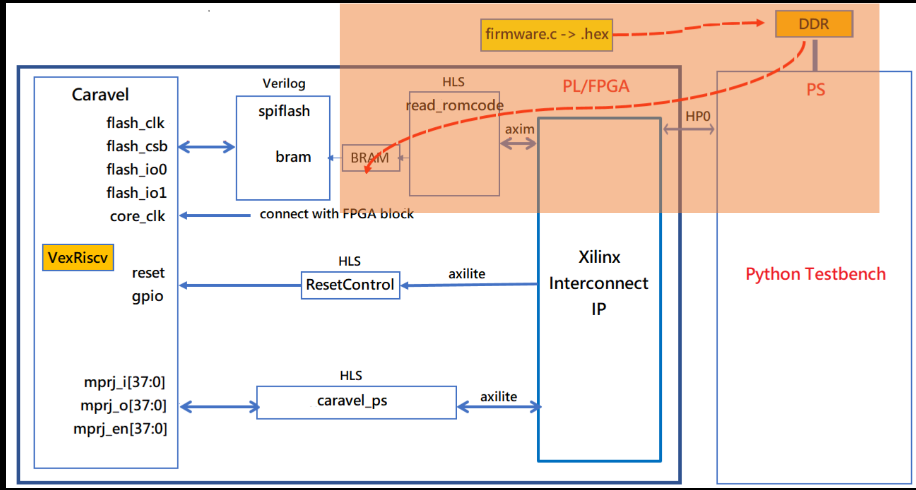
```python
for line in fiROM:
    # offset header
    if line.startswith('@'):
        # Ignore first char @
        npROM_offset = int(line[1:].strip(b'\x00'.decode()), base = 16)
        npROM_offset = npROM_offset >> 2 # 4byte per offset
        #print (npROM_offset)
        npROM_index = 0
        continue
    #print (line)

    # We suppose the data must be 32bit alignment
    buffer = 0
    bytecount = 0
    for line_byte in line.strip(b'\x00'.decode()).split():
        buffer += int(line_byte, base = 16) << (8 * bytecount)
        bytecount += 1
        # Collect 4 bytes, write to npROM
        if(bytecount == 4):
            npROM[npROM_offset + npROM_index] = buffer
            # Clear buffer and bytecount
            buffer = 0
            bytecount = 0
            npROM_index += 1
            #print (npROM_index)
            continue
    # Fill rest data if not alignment 4 bytes
    if (bytecount != 0):
        npROM[npROM_offset + npROM_index] = buffer
        npROM_index += 1
```

3. Allocate dram buffer and open the *.hex file.
4. Determine offset address by tracking @ flag
5. Parsing following data and write into buffer every 4 bytes
6. Pack remaining bytes if not 4 bytes alignments

counter_la.hex

```
@00000000
6F 00 00 0B 13 00 00 00 13 00 00 00 13 00 00 00
13 00 00 00 13 00 00 00 13 00 00 00 13 00 00 00
23 2E 11 FE 23 2C 51 FE 23 2A 61 FE 23 28 71 FE
23 26 A1 FE 23 24 B1 FE 23 22 C1 FE 23 20 D1 FE
23 2E E1 FC 23 2C F1 FC 23 2A 01 FD 23 28 11 FD
23 26 C1 FD 23 24 D1 FD 23 22 E1 FD 23 20 F1 FD
13 01 01 FC EF 00 00 11 83 20 C1 03 83 22 81 03
03 23 41 03 83 23 01 03 03 25 C1 02 83 25 81 02
03 26 41 02 83 26 01 02 03 27 C1 01 83 27 81 01
03 28 41 01 83 28 01 01 03 2E C1 00 83 2E 81 00
```

caravel_fpga.ipynb

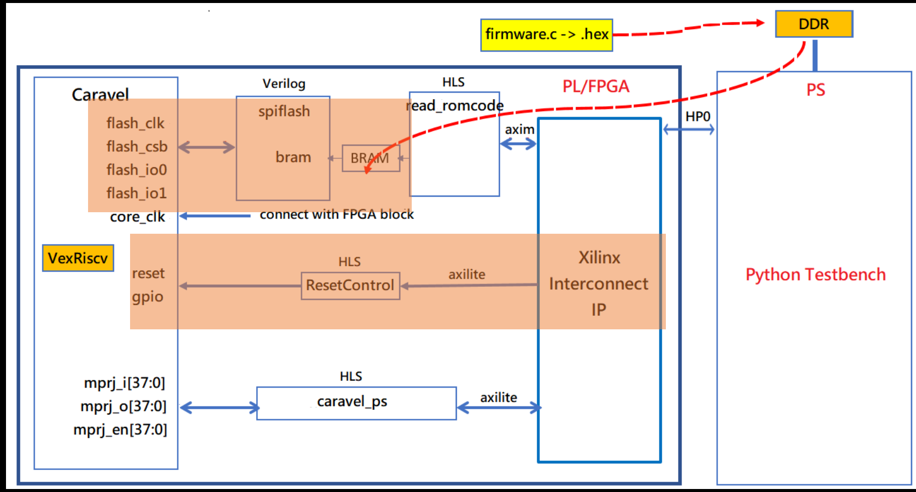



```python
# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)

# Program physical address for the romcode base address
ipReadROMCODE.write(0x10, npROM.device_address)
ipReadROMCODE.write(0x14, 0)
# Program length of moving data
ipReadROMCODE.write(0x1C, rom_size_final)


# ipReadROMCODE start to move the data from rom_buffer to bram
ipReadROMCODE.write(0x00, 1) # IP Start
while (ipReadROMCODE.read(0x00) & 0x04) == 0x00: # wait for done
    continue

print("Write to bram done")
```

7. Program readromcode IP for the source address and data length
8. Trigger IP start to move the data from dram to BRAM.

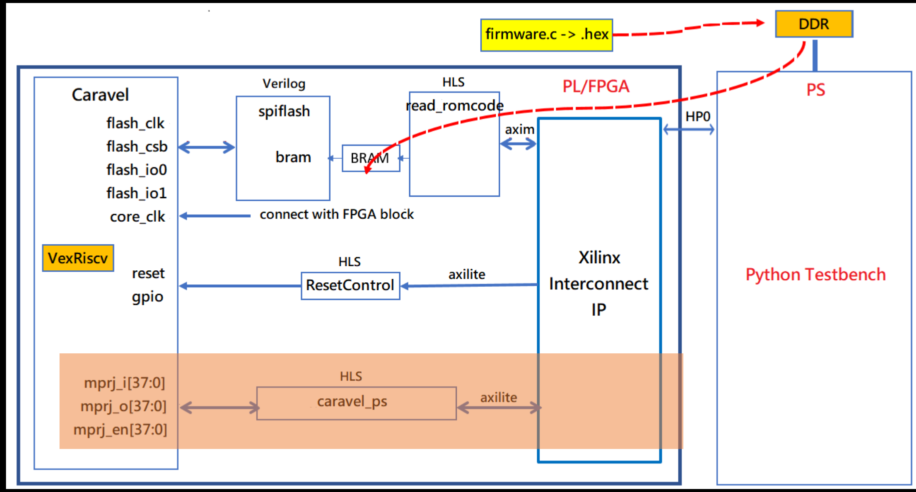
caravel_fpga.ipynb

```
# Release Caravel reset
# 0x10 : Data signal of outpin_ctrl
#        bit 0  - outpin_ctrl[0] (Read/Write)
#        others - reserved
print (ipOUTPIN.read(0x10))
ipOUTPIN.write(0x10, 1)
print (ipOUTPIN.read(0x10))
```



9. Program outputpin IP to de-assert Caravel reset pin (Caravel reset is low active)
10. Caravel CPU start fetch code via SPI interface and execute

caravel_fpga.ipynb




```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

11. Get mprj_i/o/en data by reading the Caravel_ps IP registers.
12. Compare the mprj_o value = 0xab51 (16-31bits) should sync to final result in the firmware code

counter_la.c

```
while (1) {
        if (reg_la0_data_in > 0x1F4) {
                reg_mprj_datal = 0xAB410000;
                break;
        }
}
//print("\n");
//print("Monitor: Test 1 Passed\n\n");  // Makes simulation very long!
reg_mprj_datal = 0xAB510000;
```

Thanks you for listening