SoC Design Lab

Lab5 - Caravel SoC FPGA Integration

Folder Structure

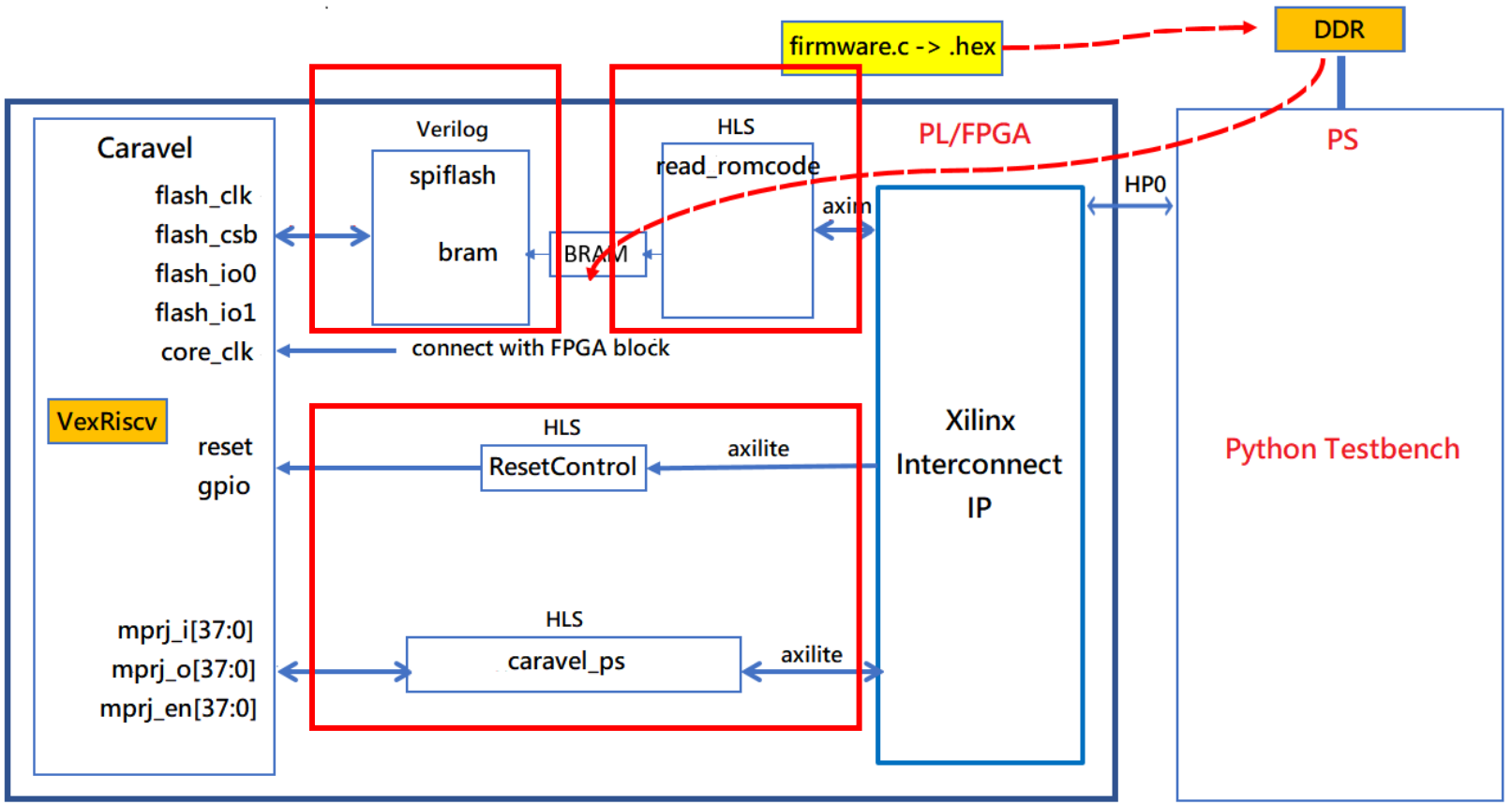
- Project
  - jupyer_notebook_project
    - Includes bitstream, firmware, and python code.
  - vitis_hls_project
    - All vitis source code (lab 1, lab 3 and outputpin IP) are included. Use run_vitis.sh to build the HLS project and export to IP separately.
  - vvd_srcs
    - Includes Caravel-SOC source code, testbench related code.
      Note: Some source codes are modified for the Caravel-FPGA project.
  - run_vitis.sh
  - run_vivado.sh
  - vvd_caravel_fpga.tcl

Outline

- Caravel FPGA Architecture Overview
- Jupyter Notebook Introduction
- Code Trace: caravel_fpga.ipynb
- Labi github
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/labi
- Lab
  - Video
    - https://www.youtube.com/watch?v=EF3vXdaVof0&t=63m46s
  - ppt
    - https://github.com/bol-edu/caravel-soc_fpga-lab/files/12035595/Caravel.FPGA.Introduction.pdf
- Build Environment
  - OS: Ubuntu 20.4.6 LTS 64bit
  - Vivado version: 2022.1

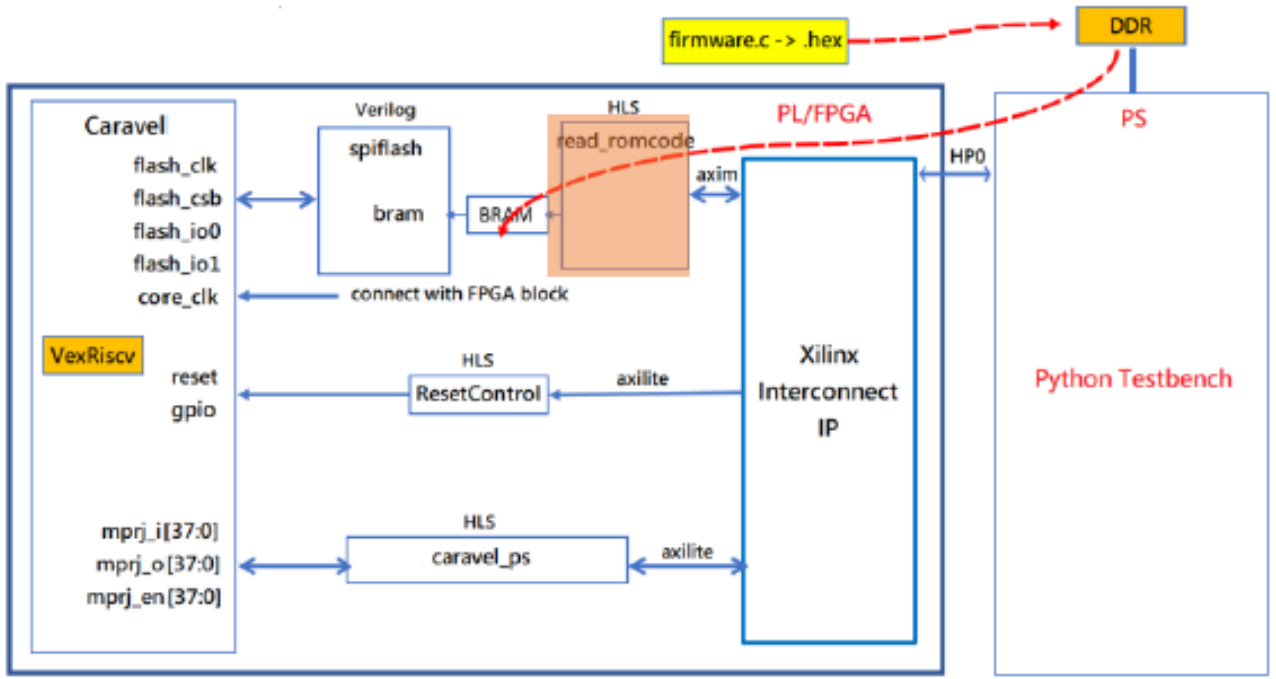
Architecture

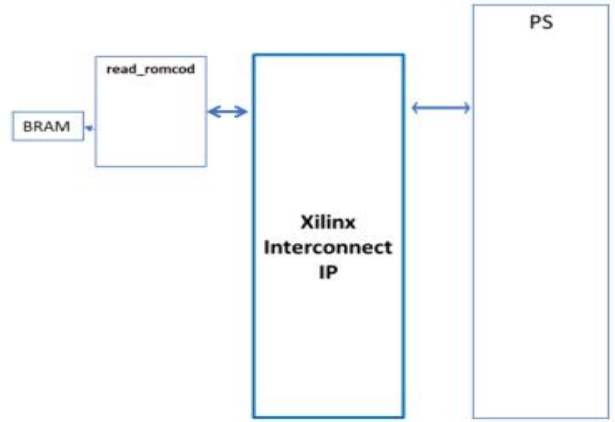
- Read_ROMcode
- Spiflash
- Caravel
  - PS
  - GPIO



Read_ROMcode

- Design source: read_romcode.cpp
- 1. Add another axi-master path to write to PS Memory
- 2. Load program.hex (RISCV code from any of the Caravel testbench) to PS memory buffer
- 3. Develop host code to load program.hex To BRAM, and read from BRAM.
- 4. Compare the input and output buffer content is the same

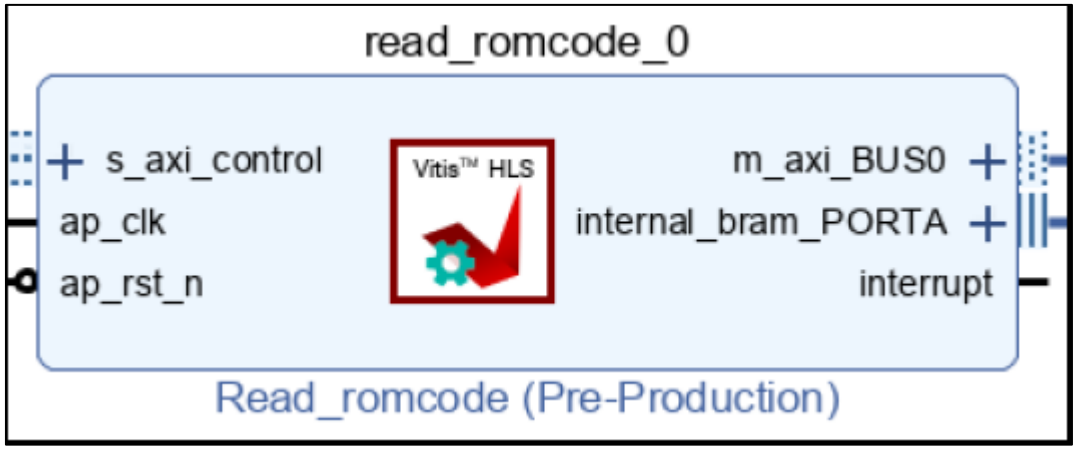




Read_romcode

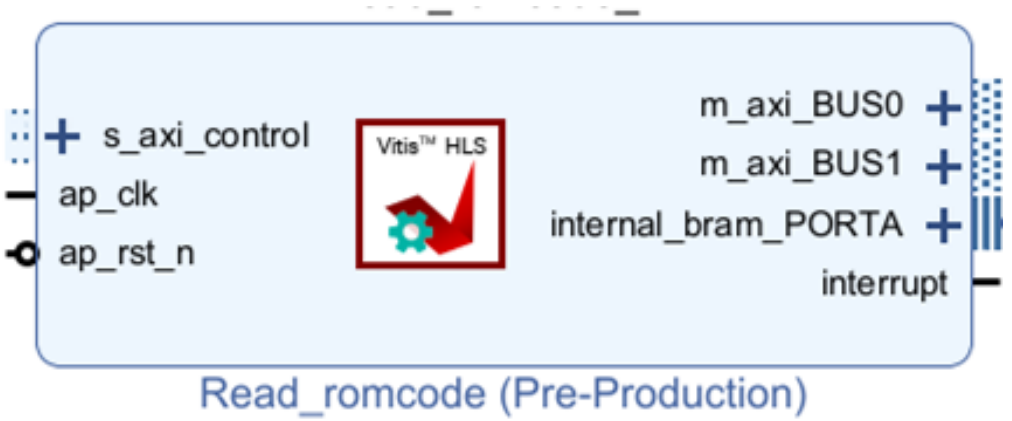
- Copy PS dram buffer to BRAM base on the size of binary file
- Limit the BRAM size to 8K
- Implement by HLS and export IP for Vivado project usage
- github reference
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab1

```
# 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)
```



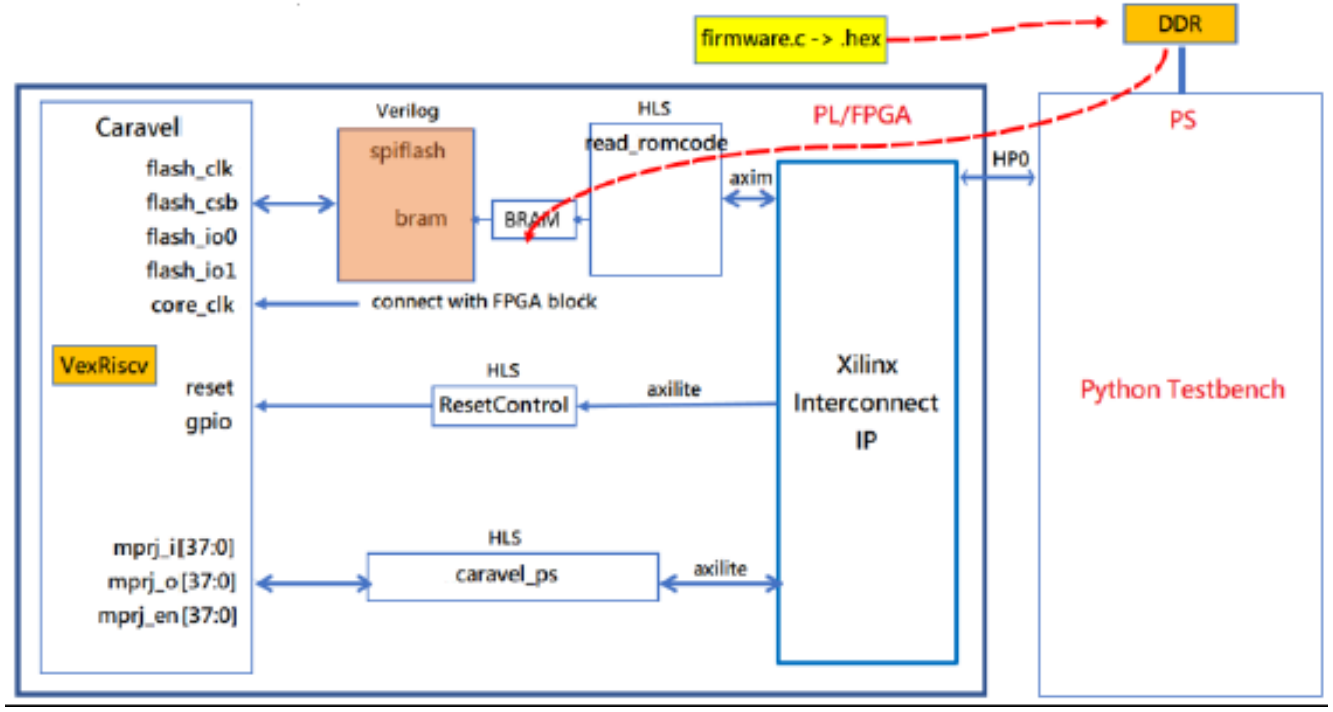
Read_ROMcode – control flow

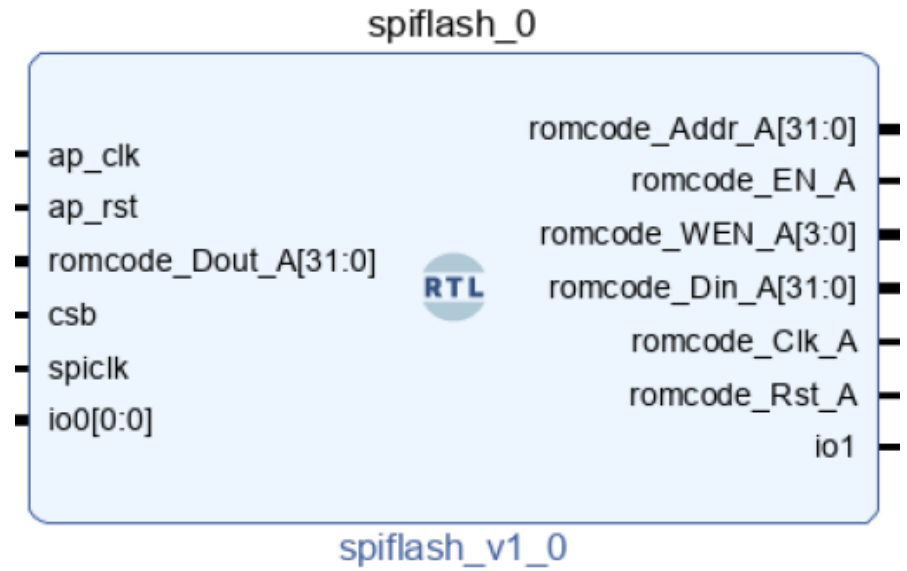
- IP read ROM code from DRAM
  - PS set m_axi_BUS0 base address
  - PS send read command
  - IP start read DRAM ROM code
  - PS wait for IP done
- IP write ROM code to DRAM
  - PS set m_axi_BUS1 base address
  - PS send write command
  - IP start write ROM code to DRAM
  - PS wait for IP done



Spiflash

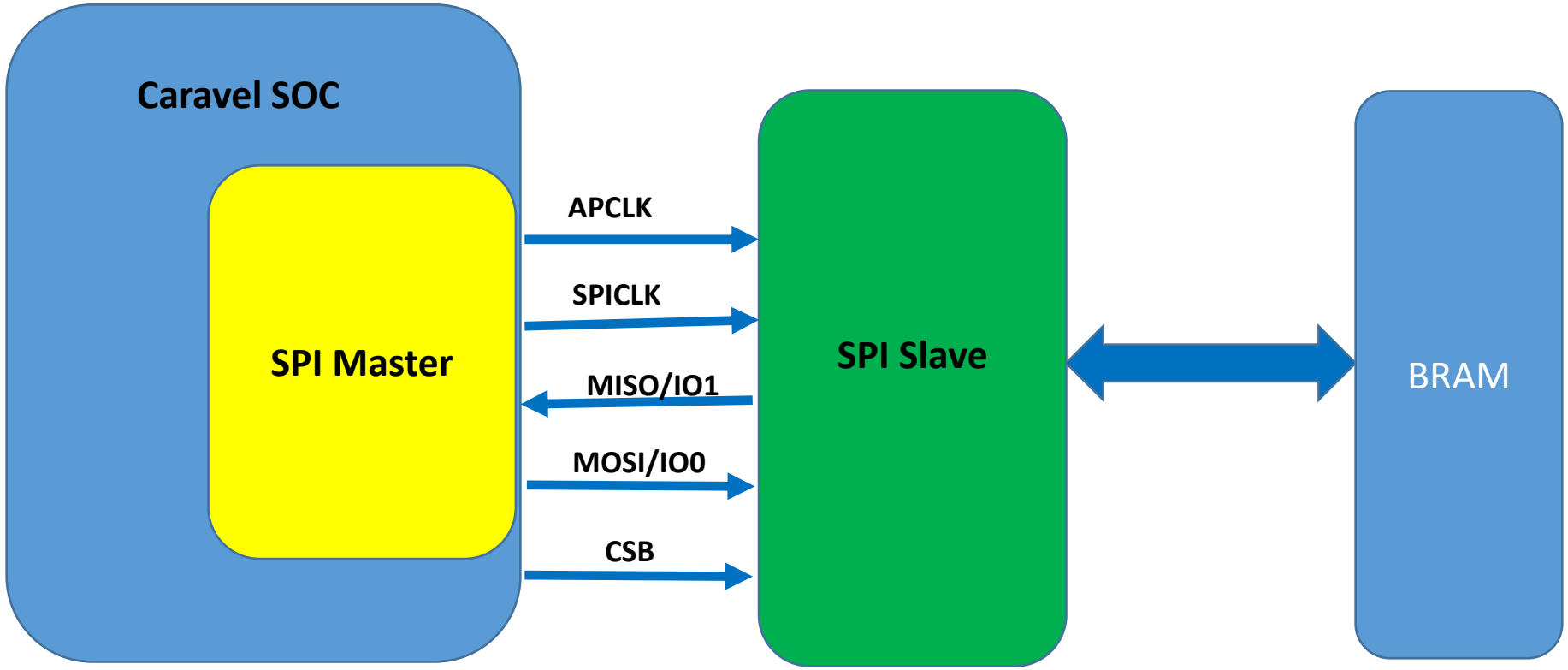
- Implement SPI slave device, only support read command (0x03)
- Return data from BRAM to Caravel





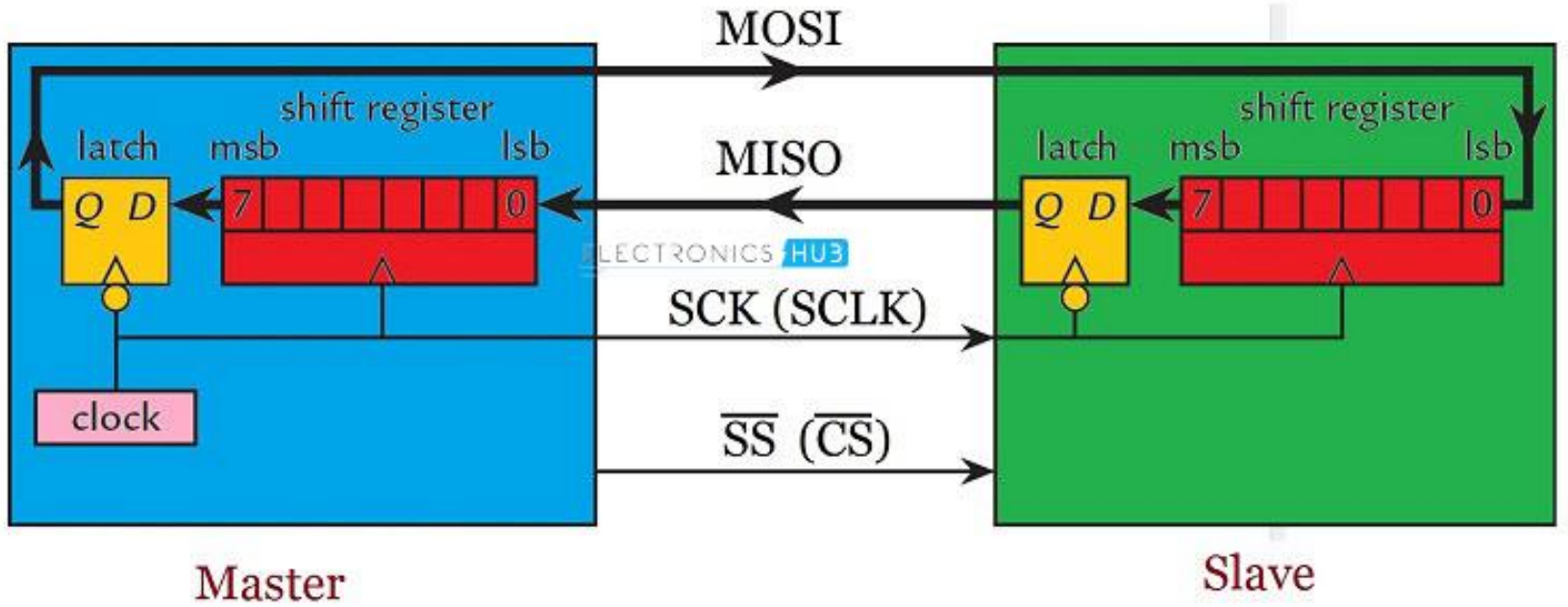
SPI spec introduction (1/2)

- SPI interface



SPI spec introduction (2/2)

- Master Output Slave Input (MOSI)
- Master Input Slave Output (MISO)
- Serial Clock (SCK)
- SS



Caravel

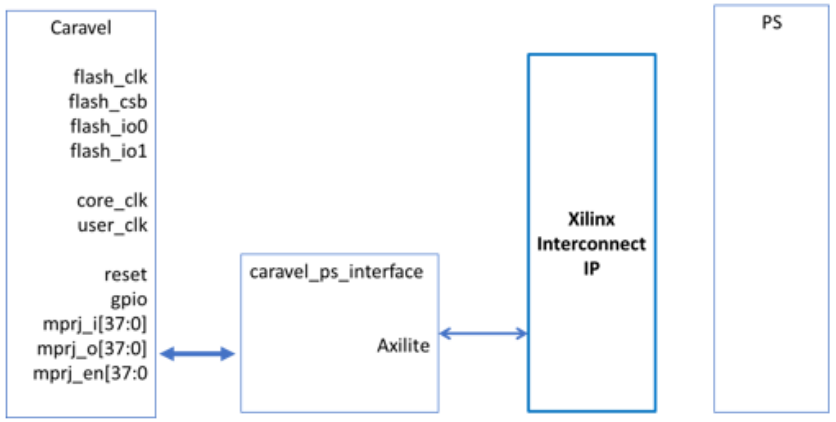
- Reference Design: caravel_ps.cpp
- Lab Content
  - Design a simple module mpri_control.v
    - Use one mprij_i pin (synchronize with host code) to stage through several steps, e.g.
    - Change mpri_o pins value
    - Some of mpri pins used for loop-back, e.g.
    - mprj_olx] = mprj_iIn]
    - Control mpri_en accordingly
  - Host use axilite to read mpri_.o, mpri_ .en values
- Integrate mprj_control.v & caravel_ps.v in

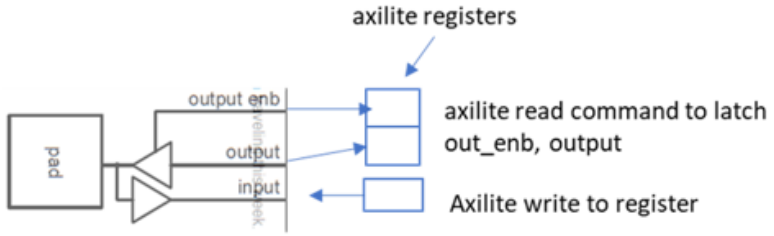
Block design - generate bitstream

- Develop Python host code to verify its

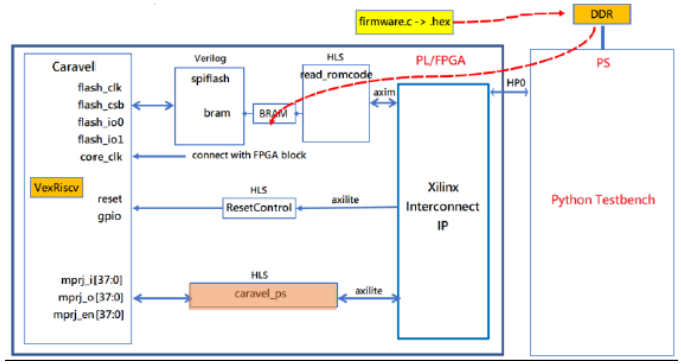
Behavior

- github reference
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab3



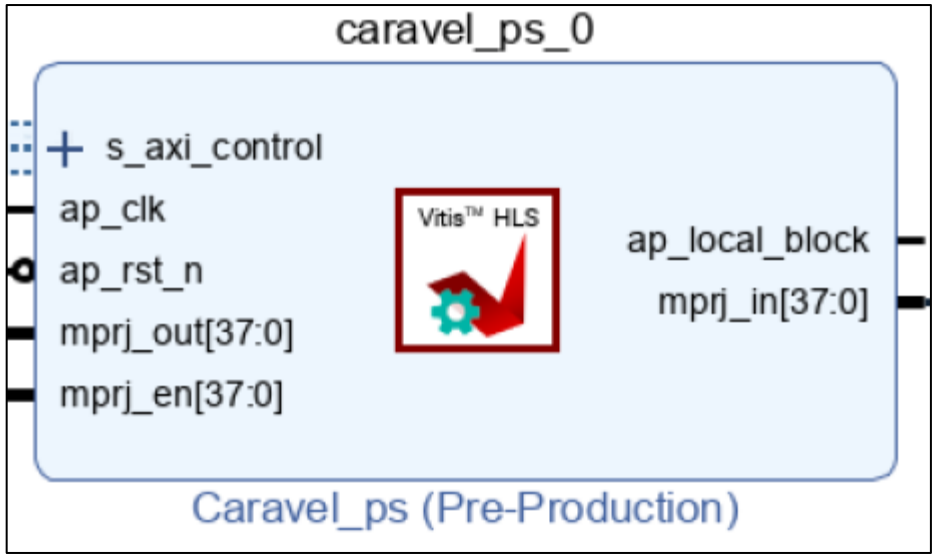


Caravel PS

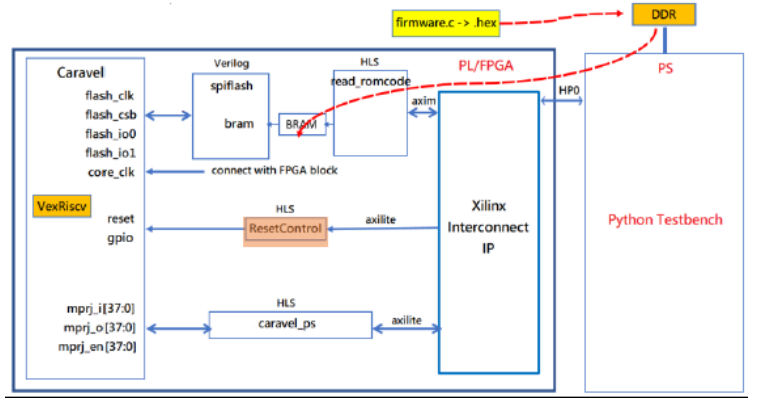


- Provide AXI Lite interface for PS CPU to read the MPRJ_IO/OUT/EN bits
- Implement by HLS and export IP for Vivado project usage.

```
# 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
```

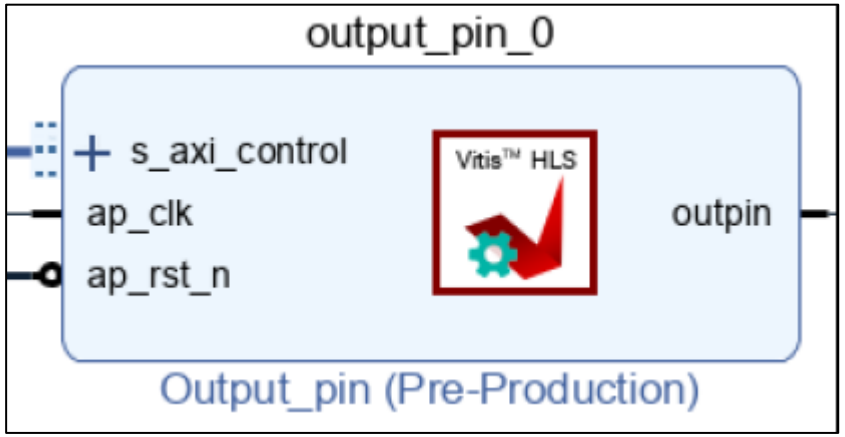


Caravel GPIO (Reset Control)



- 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
```

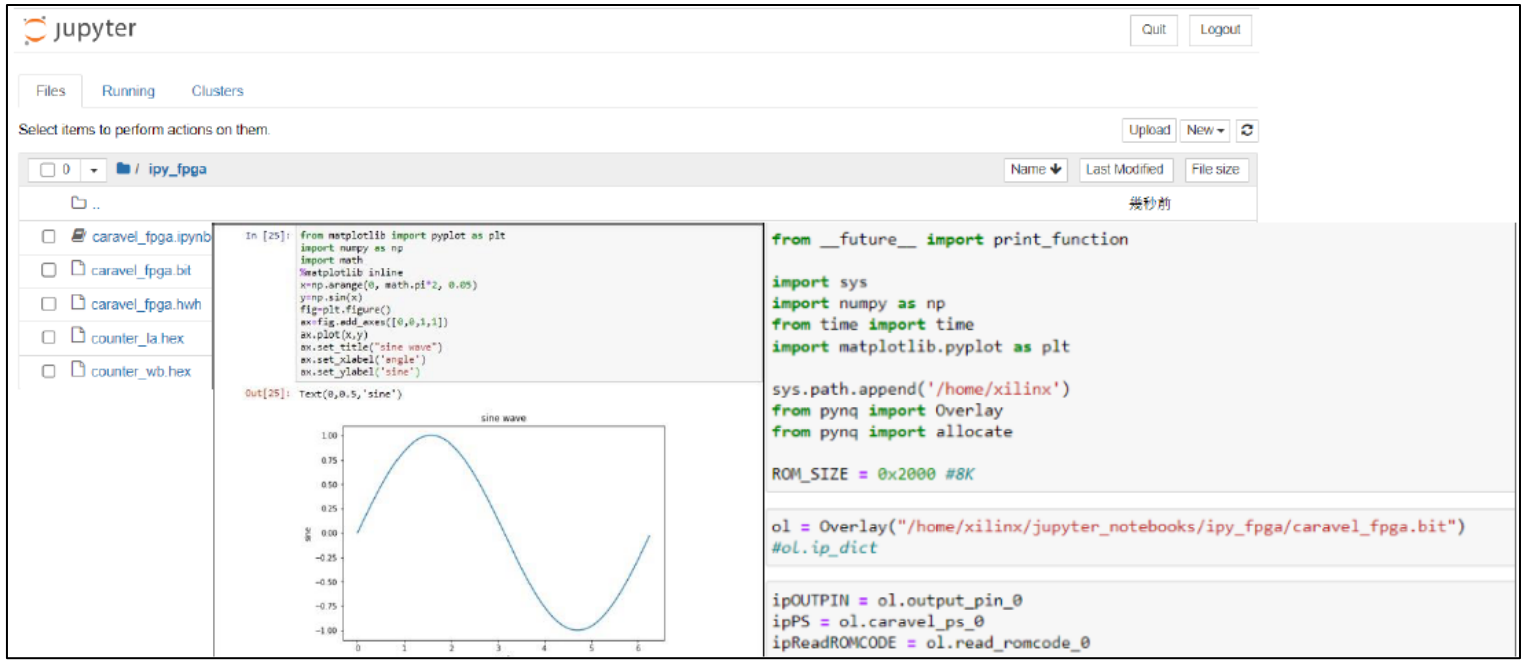


Built Steps

- Run bash run_vitis.sh the whole all hls projects will build up and export IP automatically.
  - Create **hls_caravel_ps.prj, hls_output_pin.prj, and hls_read_romcode.prj**
- Run bash run_vivado.sh to build up vivado project for Caravel-FPGA and execute “write_bitstream” step to generate bitstream.
  - Create **vvd_caravel_fpga**
  - Run_vivado.sh => User project counter with clk 50M
  - Run_vivado_gcd.sh => User project gcd with clk 10M
- The **caravel_fpga.bit** and **caravel_fpga.hwh** will be copy to folder: jupyter_notebooks_project
- Upload all files include in jupyter_notebooks_project to pynq-z2 board
- Labi github
  - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/labi

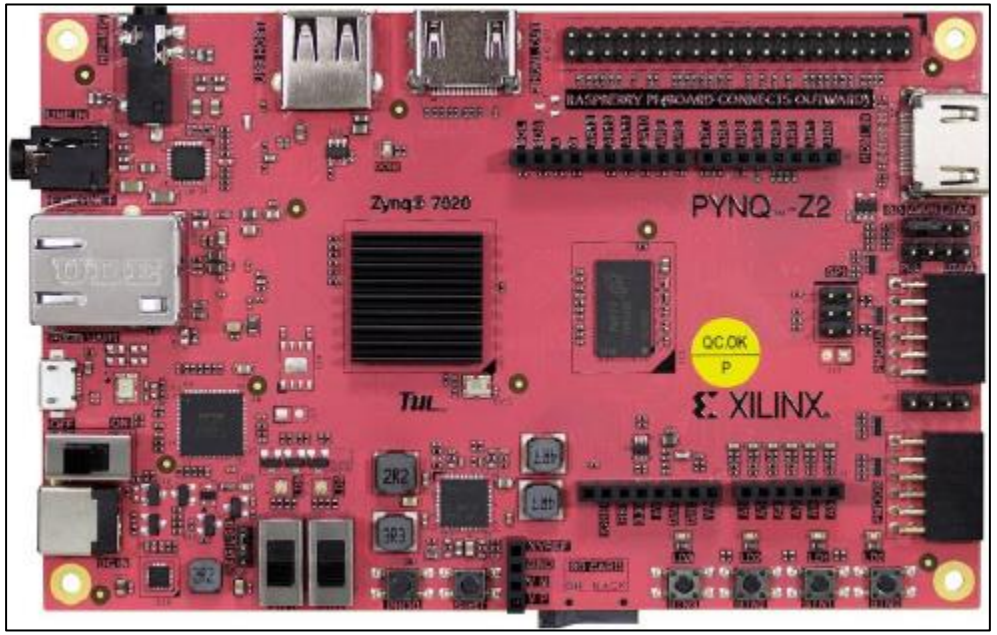
What's the Jupyter Notebooks

- 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



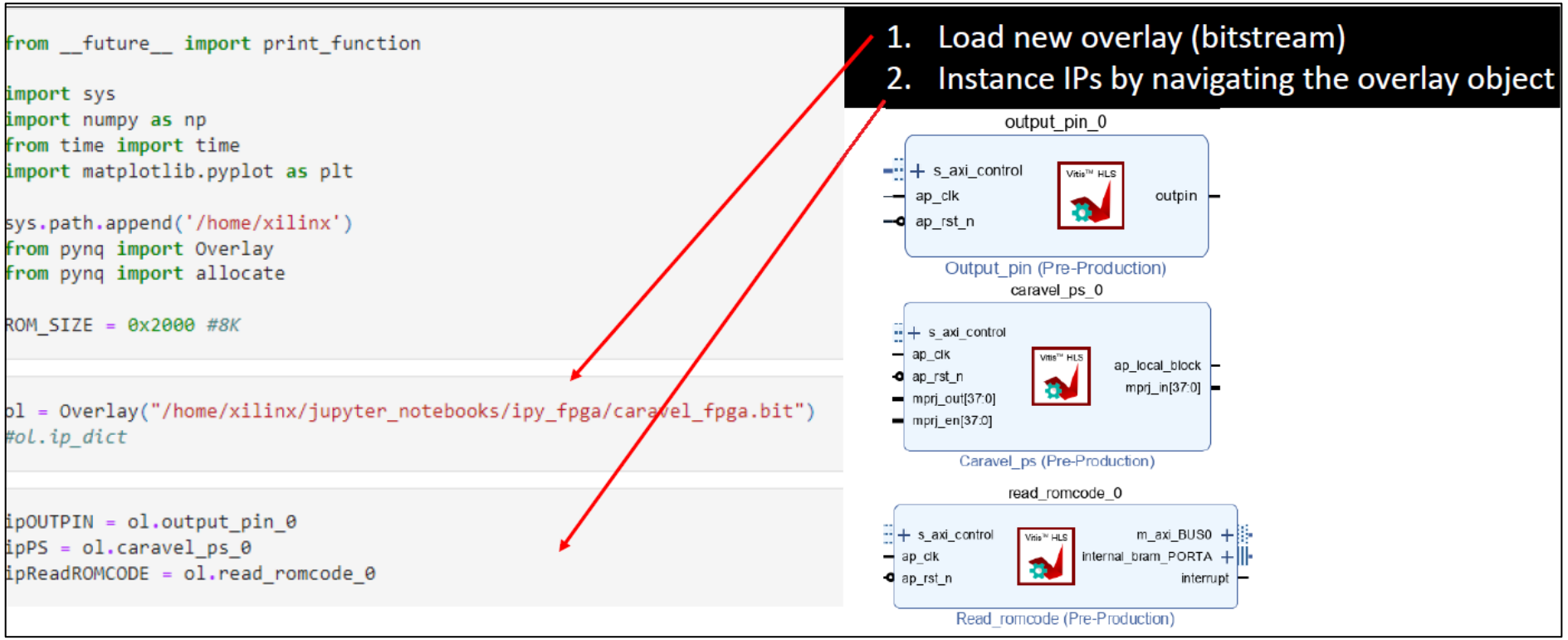
Notebook Kernels

- 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



caravel_fpga.ipynb

- https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/labi/jupyter_notebooks_project



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+")
```

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

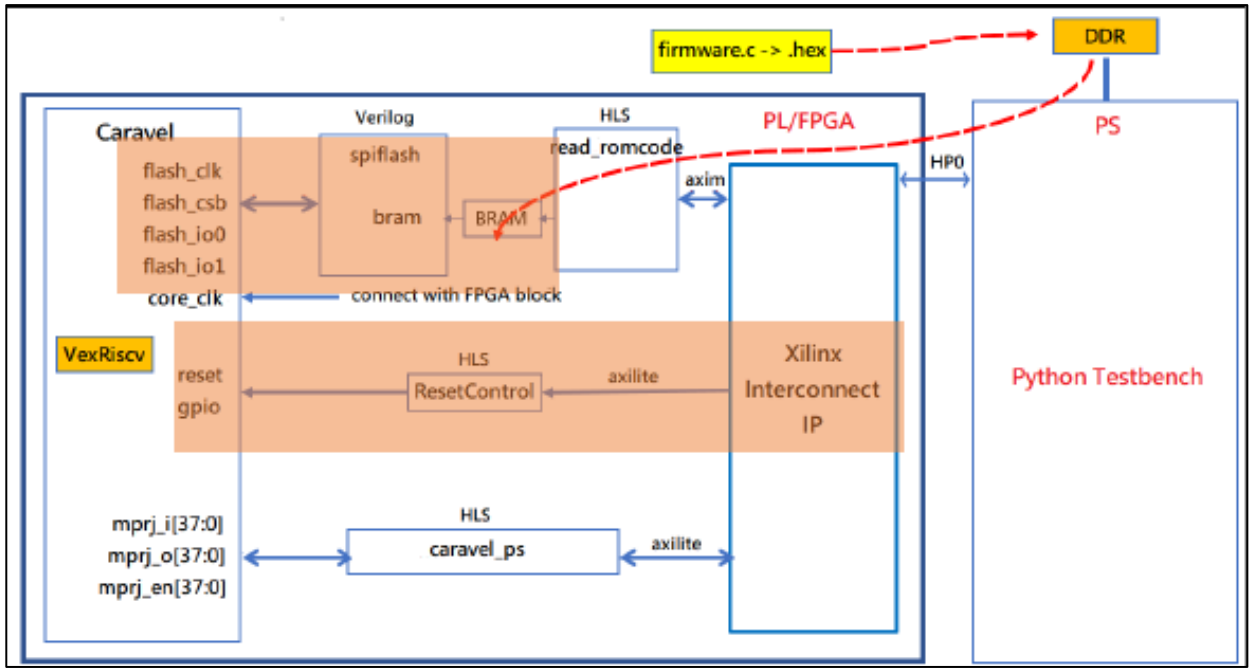
```python
for line in fiROM:
    # offset header
    if line.startswith('@'):
        # Ignore first char @
        npROM_offset = int(line[1:].strip(b'\x00'.decode()), base = 16)
        npROM_offset = npROM_offset >> 2 # 4byte per offset
        #print (npROM_offset)
        npROM_index = 0
      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

```
@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

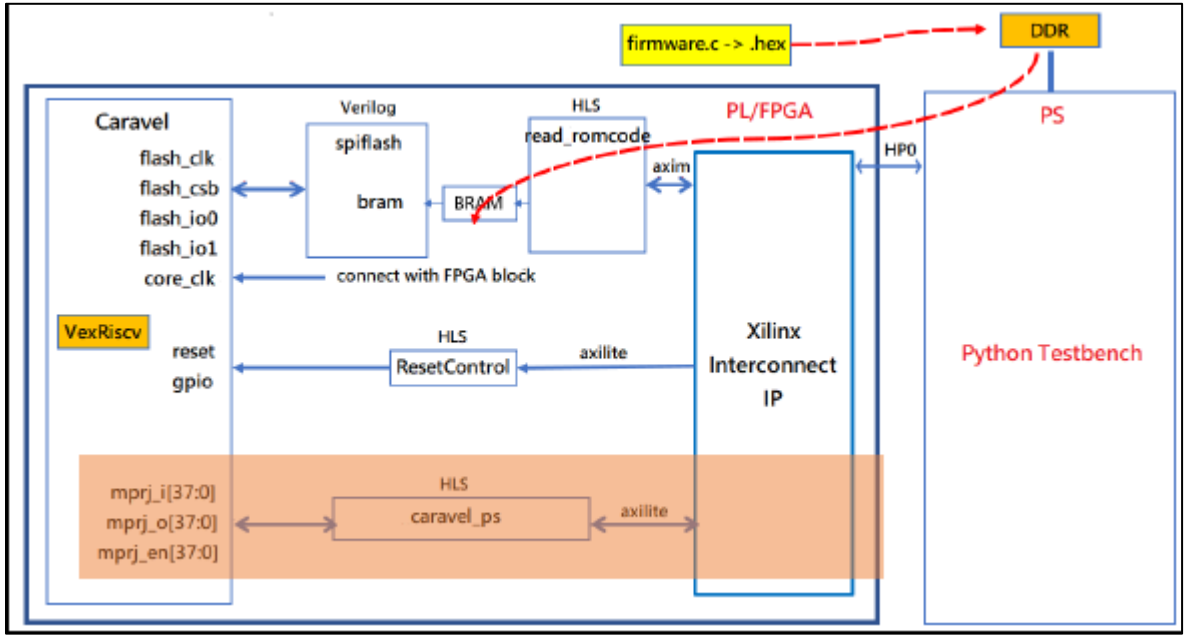
```
# Release Caravel reset
# 0x10 : Data signal of outpin_ctrl
#        bit 0  - outpin_ctrl[0] (Read/Write)
#        others - reserved
print (ipOUTPIN.read(0x10))
ipOUTPIN.write(0x10, 1)
print (ipOUTPIN.read(0x10))
```

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

```
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;
```

JuypterNotebook: caravel_fpga.ipynb

- Select one firmware binary to load into BRAM as SPIROM data
  - fiROM = open("counter_wb.hex", "r+")
  - fiROM = open("counter_la.hex", "r+")
  - fiROM = open("gcd_la.hex", "r+")
- Release reset to make Caravel-soc start execute firmware code
  - ipOUTPIN.write(0x10, 1)

  Note: If you want to load different firmware binary without restart FPGA, you need to assert & release reset signal for the caravel-soc to make CPU execute new firmware.

  1. Load counter_wb.hex
  2. Release reset by ipOUTPIN.write(0x10, 1), check mprj_i/o/en
  3. Load counter_la.hex
  4. Assert reset by ipOUTPIN.write(0x10, 0)
  5. Release reset by ipOUTPIN.write(0x10, 1), check mprj_i/o/en

Submission (1/3)

- Hierarchy:
  - StudentID_lab5/
    - Report.pdf

Submission (2/3)

- Report
  - Block diagram
  - FPGA utilization
  - Explain the function of IP in this design
    - HLS：read_romcode, ResetControl, caravel_ps
    - Verilog ：spiflash
  - Run these workload on caravel FPGA
    - counter_wb.hex
    - counter_la.hex
    - gcd_la.hex
  - Screenshot of Execution result on all workload
  - Study caravel_fpga.ipynb, and be familiar with caravel SoC control flow

Submission (3/3)

- Compress all above files in a single zip file named
  - StudentID_lab5.zip
- Submit to Submit to NTU COOL
- Deadline: 11/23 (Thu.) 23:59
  - 20% off for the late submission penalty within 3 days