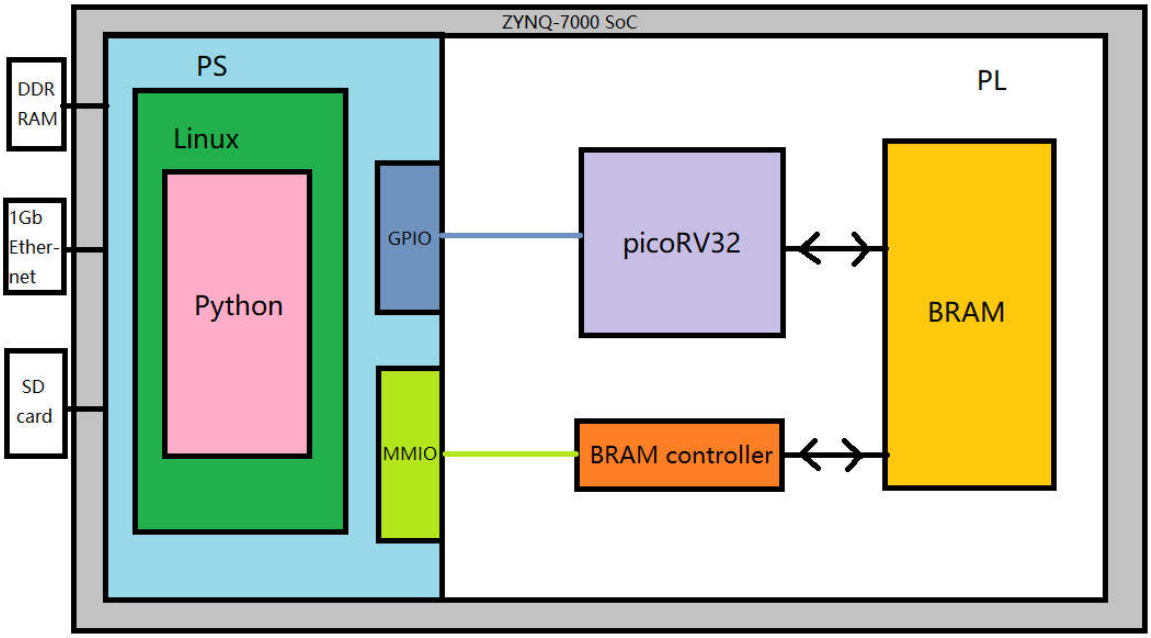
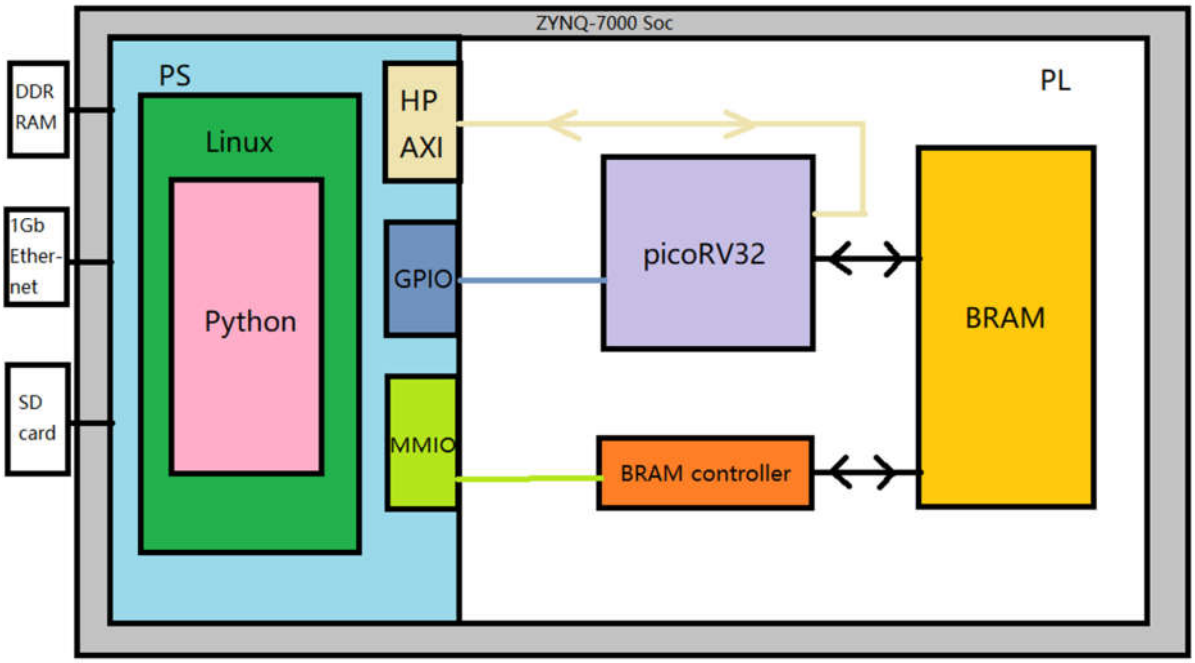
RISC-V 移植（RISC-V-On-PYNQ）

1. Overlay 简介

RISC-V-On-PYNQ 是一个将 RISC-V 工具链集成到 PYNQ 框架下，以实现对 FPGA 部分例化的 RISC-V 核心的编程的项目。



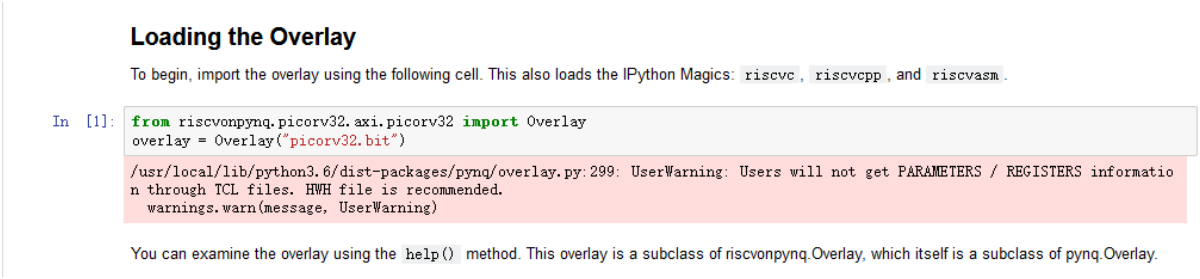


这两张图是工程项目示意图。其中 picoRV32 是一个开源的 RISC-V 核，它具有占用资源少的特点，并且它自带 AXI 接口，可以方便地使用 Xilinx 提供的各种基于 AXI 总线的 IP。本项目有两个工程，它们的区别主要是 picoRV32 接的 RAM：上图只使用 BRAM，而下图既使用 BRAM 也使用外部 DRAM（通过 PS 提供的 HP AXI 接口使用内存控制器）。

本项目的主要功能是在 jupyter notebook 上编写一段 C/C++/RISC-V 汇编程序，将编译后的二进制文件放到 picoRV32 上运行。

2. 示例 Notebook

安装好工程后，打开 RISC-V-Examples/PicoRV32 Processor Mixed-Memory Processor Demo.ipynb，这是使用 DRAM 和 BRAM 混合储存器的示例工程。实际上代码与使用只 BRAM 的工程类似，只不过使用了不同的 bit 文件。



Loading the Overlay

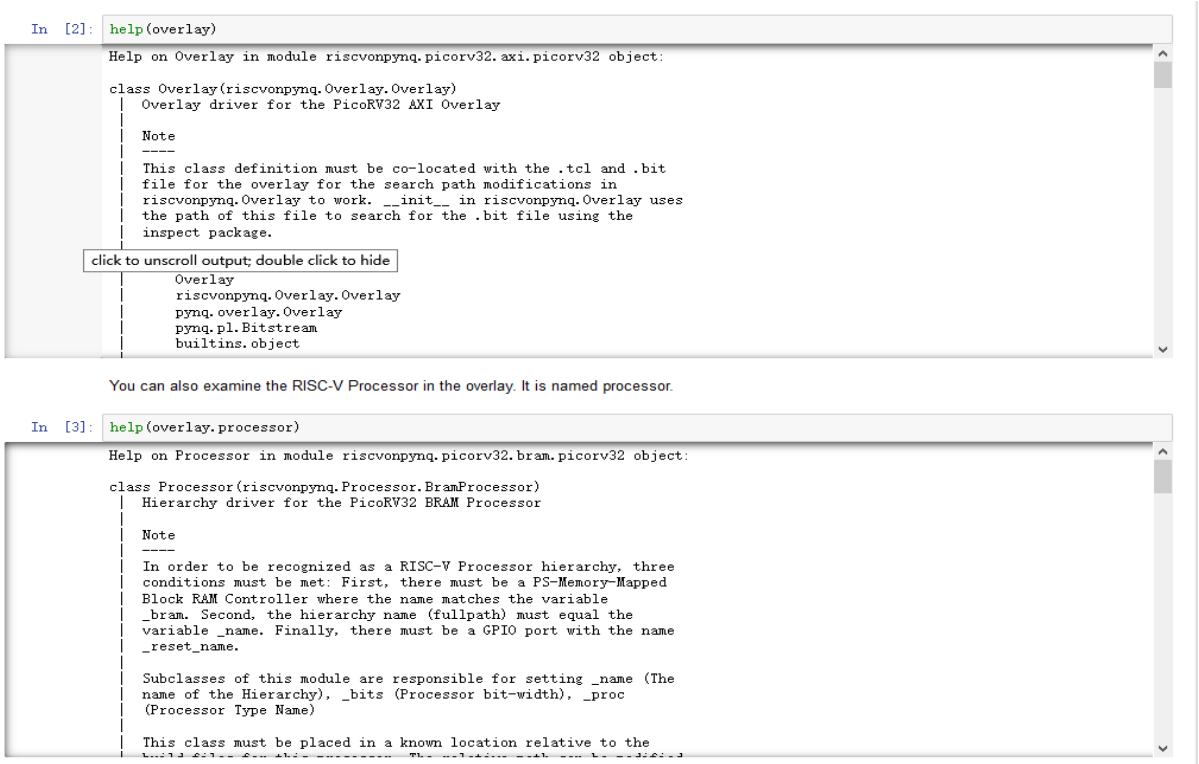
To begin, import the overlay using the following cell. This also loads the IPython Magics: `riscvc`, `riscvcpp`, and `riscvasm`.

```
In [1]: from riscvonpynq.picorv32.axi.picorv32 import Overlay
        overlay = Overlay("picorv32.bit")
```

```
/usr/local/lib/python3.6/dist-packages/pynq/overlay.py:299: UserWarning: Users will not get PARAMETERS / REGISTERS information through TCL files. HWH file is recommended.
  warnings.warn(message, UserWarning)
```

You can examine the overlay using the `help()` method. This overlay is a subclass of riscvonpynq.Overlay, which itself is a subclass of pynq.Overlay.

开始时下载 bit 文件，在这个过程中所有驱动都会注册完成。



```
In [2]: help(overlay)
```

```
Help on Overlay in module riscvonpynq.picorv32.axi.picorv32 object:

class Overlay(riscvonpynq.Overlay.Overlay)
 |  Overlay driver for the PicoRV32 AXI Overlay
 |
 |  Note
 |  ----
 |  This class definition must be co-located with the .tcl and .bit
 |  file for the overlay for the search path modifications in
 |  riscvonpynq.Overlay to work. __init__ in riscvonpynq.Overlay uses
 |  the path of this file to search for the .bit file using the
 |  inspect package.
 |
```

click to unscroll output; double click to hide

```
 |      Overlay
 |      riscvonpynq.Overlay.Overlay
 |      pynq.overlay.Overlay
 |      pynq.pl.Bitstream
 |      builtins.object
 |
```

You can also examine the RISC-V Processor in the overlay. It is named processor.

```
In [3]: help(overlay.processor)
```

```
Help on Processor in module riscvonpynq.picorv32.bram.picorv32 object:

class Processor(riscvonpynq.Processor.BramProcessor)
 |  Hierarchy driver for the PicoRV32 BRAM Processor
 |
 |  Note
 |  ----
 |  In order to be recognized as a RISC-V Processor hierarchy, three
 |  conditions must be met: First, there must be a PS-Memory-Mapped
 |  Block RAM Controller where the name matches the variable
 |  _bram. Second, the hierarchy name (fullpath) must equal the
 |  variable _name. Finally, there must be a GPIO port with the name
 |  _reset_name.
 |
 |  Subclasses of this module are responsible for setting _name (The
 |  name of the Hierarchy), _bits (Processor bit-width), _proc
 |  (Processor Type Name)
 |
 |  This class must be placed in a known location relative to the
```

可以用 help 函数查看 overlay 和 processor 的有关信息，可以看到，这里的 overlay 和 processor 使用了本项目设计的驱动，这说明驱动注册成功了。

RISC-V Magics

Our package provides three RISC-V Magics. The first is `riscvc`, which compiles C code.

In [4]:
```
%%riscvc test overlay.processor

int main(int argc, char ** argv){
    unsigned int * a = (unsigned int *)argv[1];
    return a[2];
}
```

Out[4]: Compilation of program test SUCCEEDED

You can run the test program above and pass it arguments. The arguments must be a Numpy type.

In [5]:
```
import numpy as np
arg1 = np.array(range(1, 10), np.uint32)
retval = overlay.processor.run(test, arg1)

if(retval != arg1[2]):
    print("Test Failed!")
else:
    print("Test Passed!")
```

Test Passed!

这里我们编写了一段 C 程序，作用是返回一个数组的第二个元素。可以看到我们使用了 python magics 来声明并编译一段 C 程序，这和 PYNQ 本身对 microblaze 核的编程方法类似。然后调用 processor 的 run 方法将程序装载进 RAM 中，令 picoRV32 执行。

You can also examine the processor's memory:

In [7]:
```
arr = overlay.processor.psBramController.mmio.array
for i in range(128):
    print(f'Memory Index {i:3}: {arr[i]:#0{10}x}')
```

```
Memory Index   0: 0x00000013
Memory Index   1: 0x00000093
Memory Index   2: 0x00008137
Memory Index   3: 0x00000193
Memory Index   4: 0x00000213
Memory Index   5: 0x00000293
Memory Index   6: 0x00000313
Memory Index   7: 0x00000393
Memory Index   8: 0x00000413
Memory Index   9: 0x00000493
Memory Index  10: 0xffc12503
Memory Index  11: 0xff812583
Memory Index  12: 0x00000613
Memory Index  13: 0x00000693
Memory Index  14: 0x00000713
Memory Index  15: 0x00000793
Memory Index  16: 0x00000813
Memory Index  17: 0x00000893
Memory Index  18: 0x00000913
```

我们也可以调用 BRAM controller 的 mmio 来看看内存的情况。

在这之后也有 C++和汇编程序的例子，这里不作赘述。

3. Overlay 详解（Vivado 需升级到 2018.3 版本）

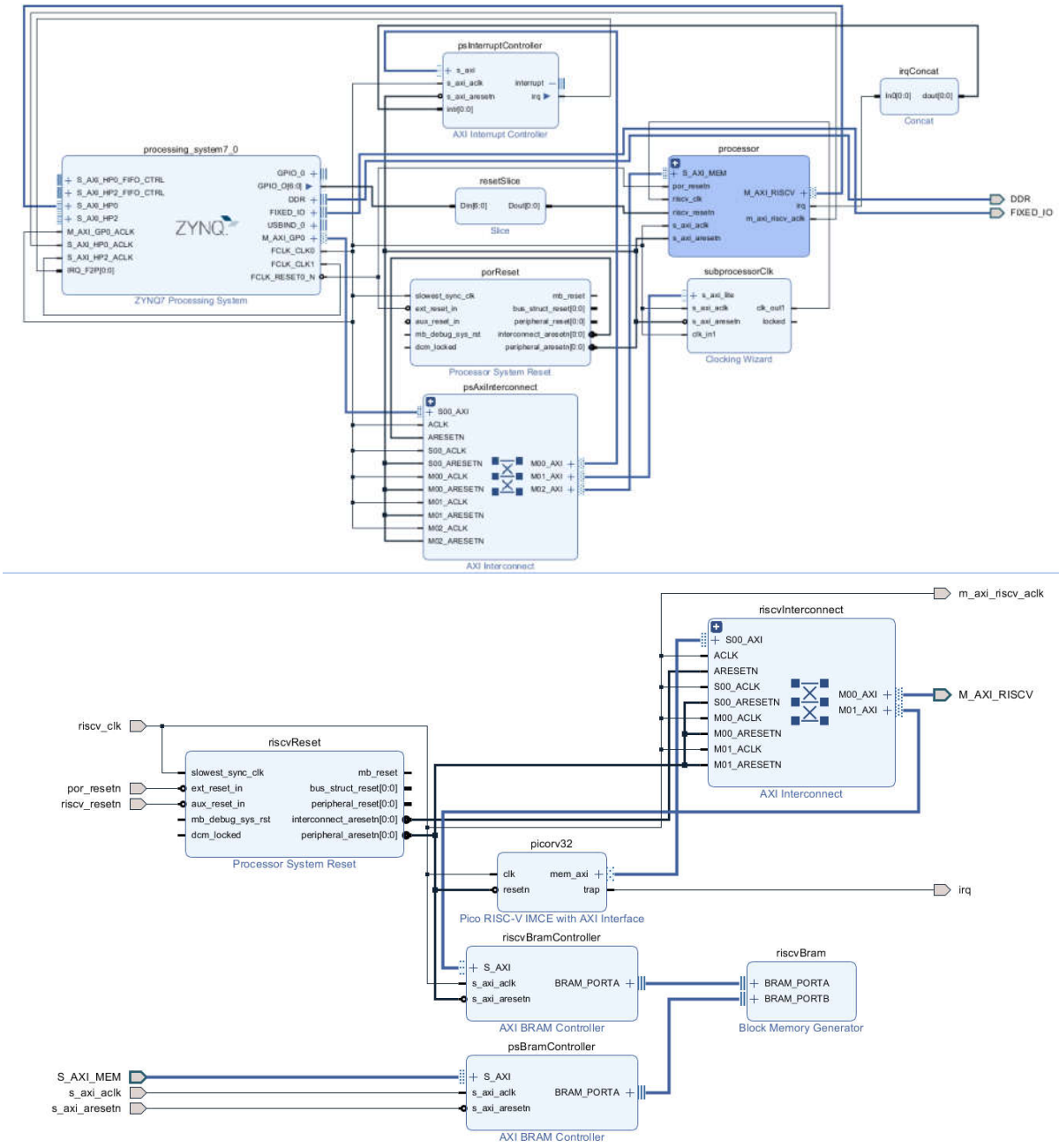
- PS 与 PL 功能划分

  PS 部分主要是用 Python 通过 PYNQ 框架控制 PL 中各 IP 的运行。

PL 部分则例化一个 pcioRV32 核和和 BRAM。

- Vivado 工程 block design 介绍

具体硬件设计是这样的：



上图是项目的总体布局，下图是 processor 展开后的内容。

其中 BRAM 是一个双口 RAM，它两端连接的是 PS 和 picoRV32。另外，可以看到 PS 的 GPIO 连接的是 processor 的复位端，当二进制程序装入 BRAM 后，复位 picoRV32，使它运行程序。picoRV32 运行结束后，会触发一个中断。利用这个设计，可以例化许多 RISC-V 核心，并让他们在访问图一片内存的情况下运行独立的程序，实现一个灵活可配置的众核处理器，事实上已经有用这种方法实现了例化超过一千个 RISC-V 核的项目。

在 processor 内部，picoRV32 通过 AXI 总线来访问其他 IP，可以使用 AXI 总线来给它增加各种各样不同的外设。在这里，本项目只是添加了另一个内存（通过 PS 的 HP AXI 0 接口访问内存控制器）。

另外，这里的 picoRV32 核心的运行频率可以通过利用 AXI 总线配置时钟资源来调整。

注意，这里 RISC-V 核的 hierarchy 名字必须为 processor，并且 processor 中的复位模块必须为 rscvReset。如果改动这些名字，必须修改对应的驱动源文件，否则不能正确加载驱动。

- PS 侧环境准备

在 PS 端，我们需要编译安装 RISC-V 工具链和我们的库。环境需求 PYNQ v2.4，Python 3.6.5。

首先运行如下代码：

```
git clone https://github.com/Siudya/RISC-V-On-PYNQ.git /home/xilinx/ RISC-V-On-PYNQ
```

然后在/home/xilinx/ RISC-V-On-PYNQ/notebooks/tutorial 中有 5 个 notebook，按顺序阅读并运行它们。这些 notebook 将完整介绍本项目的安装过程。

也可以使用做好的镜像文件，直接烧录到 SD 卡。链接：
https://pan.baidu.com/s/1MuG44EwHkZnmVfAEKO9CXg 提取码：1qaj

- Overlay API 介绍

在此工程中，通过 riscvc、riscvcpp 和 riscvasm 这三个 python magics 来编译一段程序。例如：%%riscvcpp test_cpp overlay.processor。其中 test_cpp 是程序的名字，overlay.processor 是想要写入的 RISCV 核 hierarchy 的名字。然后在本 cell 中编写代码，运行后将调用前面安装的 RISC-V 的工具链编译。

编译成功后，利用 processor 的 run 方法运行程序。例如：overlay.processor.run(test_cpp, test_cpp_arg)。其中 test_cpp 是程序名字，test_cpp_arg 是参数数组，它兼容 numpy 的类型。

对于更详细的解释，可以参看前面的 tutorial 中 5 个 notebook 和项目中的.py 源文件。