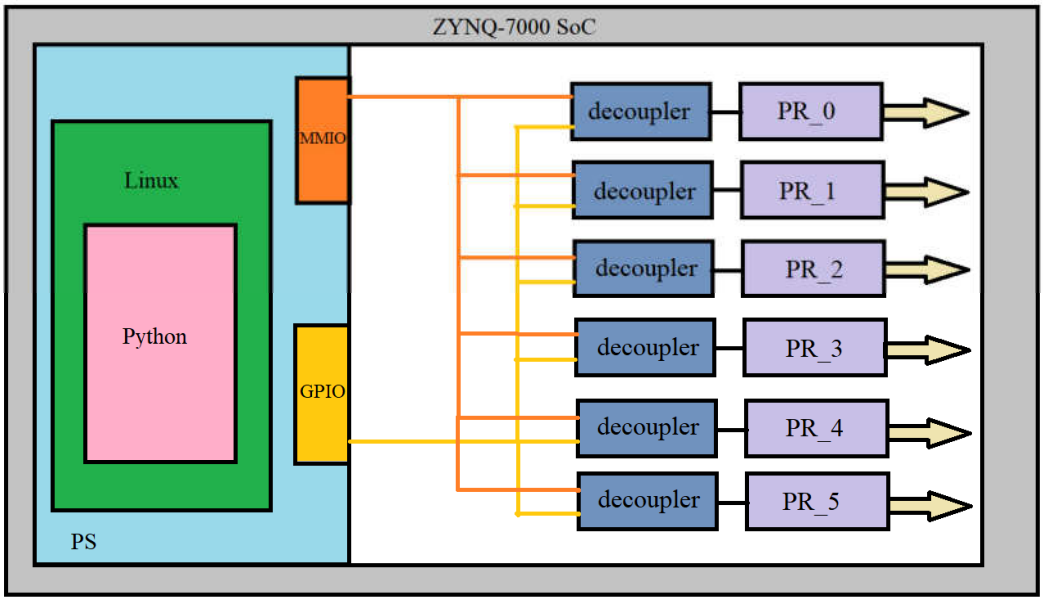
可重配置 IO（Partial Reconfiguration IO）

1. Overlay 简介

PYNQ-PRIO 是一个介绍如何利用 FPGA 部分可重配置特性和 PYNQ 框架提供的方便的 API，对 FPGA 分时复用，提高 FPGA 灵活性的项目。可重配置是指在一个 FPGA 工程中，划分了静态逻辑部分和动态逻辑部分，静态逻辑部分是指在运行过程中逻辑不变的部分，动态逻辑部分是指在运行过程中可以根据需要下载特定比特流文件实现不同逻辑的部分，在工作时，修改某一动态逻辑部分不会对静态逻辑部分和其他动态逻辑部分造成影响，实现了 FPGA 的功能上的分时复用。



上图为工程项目示意图。通过软硬件划分，在 PS 端中负责 Linux 和通信，PL 端例化了 6 个 PR（Partitial Reconfiguration）块。每个块中有 GPIO、uart 和 IIC 等几种不同的 RM（Reconfiguration Module），它们通过各自的 partitial bitstream 下载。动态逻辑部分和静态逻辑部分使用分离器保证它们互不影响，并在动态逻辑部分下载完成后对下载好的部分进行复位。

此项目中在 PYNQ 框架下有两种驱动这些 IP 的方式：一是直接使用 PYNQ 提供的 API 操作 overlay 里面的 IP，二是将这些 IP 通过 DTS（Device Tress Source）注册到 linux sysfs 中，然后调用 linux 提供的驱动。

2. 示例 Notebook

1　使用 pynq API 的方法：

打开~/prio/uart.ipynb，前面一段代码是 UART 的驱动程序和必要的功能代码，不是本文重点请读者自己研读。

Download the static bitstream

We first need to download the static or full bitstream before any partial bitstreams can be downloaded. Note that if the bitstream is not in the same directory as the notebook then the full path needs to be provided.

In [1]:
```
from pynq import Overlay

FULL_BITSTREAM_PATH = "/usr/local/lib/python3.6/dist-packages/prio/"
PARTIAL_BITSTREAM_PATH = "/usr/local/lib/python3.6/dist-packages/prio/partial/"

overlay = Overlay(FULL_BITSTREAM_PATH + "prio.bit")
```

Set up the reconfigurable region

Notice that as with the full bitstream, the full path to the partial bitstream must be provided when it is located outside of the current notebook's directory.

We will download partial bitstream and initialize each uart driver.

In [4]:
```
overlay.pr_download("pr_1", PARTIAL_BITSTREAM_PATH + "pr_1_uart.bit")
uart1 = UART(overlay.pr_1)

overlay.pr_download("pr_3", PARTIAL_BITSTREAM_PATH + "pr_3_uart.bit")
uart3 = UART(overlay.pr_3)
```

接下来就是下载 bit 文件。首先下载静态逻辑 bit 文件，静态 bit 文件中的 PR 部分是默认的 RM。然后下载 PR 的 bit 文件。注意，下载一个 PR 的 bit 文件后要立刻例化一个驱动实例，这是因为每个 PR 的 bit 文件所对应的 hwh 文件中其他 PR 部分都使用默认 RM，但是下载一个 PR 的 bit 文件并不会影响其他 PR 和静态逻辑。

Demo: Print UART Status

Prints the status of both of the UART modules

In [4]:
```
uart1.resetFIFOs()
uart3.resetFIFOs()
uart1.printStatus()
uart3.printStatus()
```

```
UART_pr_1 status:
        RX Available: False
        RX Full: False
        TX Empty: True
        TX Full: False
        Interrupts Enabled: False
UART_pr_3 status:
        RX Available: False
        RX Full: False
        TX Empty: True
        TX Full: False
        Interrupts Enabled: False
```

然后可以测试一下两个 UART 的状态，在这里我们并没有开启中断。

Demo: Bidirectional UART Messages

This cell will transmit a message back and forth between partial region 1 and partial region 3. After running the cell you will see output showing the message that was sent and the message that was recieved, going both directions.

Hardware setup: For this demo you should connect a wire between Arduino pin 8 (uart1 RX) and Arduino pin 35 (uart3 TX), and a wire between Arduino Pin 9 (uart1 TX) and Arduino pin 34 (uart3 RX). (uart3 RX). (the two wires should criss-cross)

In [5]:
```
import time


msg = [0xde, 0xad, 0xbe, 0xef]
print("***** Sending message: " + str(msg) + "*****")
uart1.write(msg)
time.sleep(1.0)
recvd = uart3.read(4)

if recvd == msg:
    print("Success: correct message received")
else:
    print("Failure: message received: (" + str(recvd) + ")")


msg = [0xaa, 0xbb, 0x55, 0x33]
print("\n***** Sending message: " + str(msg) + "*****")
uart3.write(msg)
time.sleep(2.0)
recvd = uart1.read(4)

if recvd == msg:
    print("Success: correct message received")
else:
    print("Failure: message received: (" + str(recvd) + ")")
```

```
***** Sending message: [222, 173, 190, 239]*****
Success: correct message received

***** Sending message: [170, 187, 85, 51]*****
Success: correct message received
```

下一步我们测试两个串口的收发。用杜邦线将 uart1 的 TX（Arduino 的 Pin 9）和 uart3 的 RX（Arduino 的 Pin 34），uart1 的 RX（Arduino 的 Pin 8）和 uart3 的 TX（Arduino 的 Pin 35）连接起来，运行这段代码，可以看到信息被正确地收发了。

Demo: Bidirectional UART Messages with Interrupts

This demo will repeat the demonstration from above, but this time it will utilize the interrupt functionality present in the PR regions. We will first redownload the partial bitstreams and reinitialize uart object, this time with interrupts enabled.

Hardware setup: *(Same as previous demo)* For this demo you should connect a wire between Arduino pin 8 (uart1 RX) and Arduino pin 35 (uart3 TX), and a wire between Arduino Pin 9 (uart1 TX) and Arduino pin 34 (uart3 RX). (uart3 RX). (the two wires should criss-cross)

In [7]:
```
overlay.pr_download("pr_1", PARTIAL_BITSTREAM_PATH + "pr_1_uart.bit")
interrupt = overlay.interrupt_pins['pr_1/axi_uartlite_0/interrupt']['fullpath']
uart1 = UART(overlay.pr_1, interrupt)

overlay.pr_download("pr_3", PARTIAL_BITSTREAM_PATH + "pr_3_uart.bit")
interrupt = overlay.interrupt_pins['pr_3/axi_uartlite_0/interrupt']['fullpath']
uart3 = UART(overlay.pr_3, interrupt)
```

接下来我们试试含中断的收发。首先仍然是例化实例，不过在这里我们要从 overlay.interrupts_pins 中读取每个 uart 的中断引脚，然后初始化 uart 实例。

In [8]:
```
import time
import asyncio

msg = [0xde, 0xad, 0xbe, 0xef]

uart1.resetFIFOs()
uart3.resetFIFOs()

# Send message from uart 1 to uart 3
print("***** Sending message: " + str(msg) + "*****")
uart3.enableInterrupts(True)
uart1.write(msg)

recvd = asyncio.get_event_loop().run_until_complete(uart3.isr_recv(len(msg)))
if recvd == msg:
    print("Success: correct message received")
else:
    print("Failure: message received: (" + str(recvd) + ")")

# Send message from uart 3 to uart 1
print("\n***** Sending message: " + str(msg) + "*****")
uart1.enableInterrupts(True)
uart3.write(msg)

recvd = asyncio.get_event_loop().run_until_complete(uart1.isr_recv(len(msg)))
if recvd == msg:
    print("Success: correct message received")
else:
    print("Failure: message received: (" + str(recvd) + ")")
```

```
***** Sending message: [222, 173, 190, 239]*****
UART_pr_3 isr received byte #1 of 4: 0xde
UART_pr_3 isr received byte #2 of 4: 0xad
UART_pr_3 isr received byte #3 of 4: 0xbe
UART_pr_3 isr received byte #4 of 4: 0xef
Success: correct message received

***** Sending message: [222, 173, 190, 239]*****
UART_pr_1 isr received byte #1 of 4: 0xde
UART_pr_1 isr received byte #2 of 4: 0xad
UART_pr_1 isr received byte #3 of 4: 0xbe
UART_pr_1 isr received byte #4 of 4: 0xef
Success: correct message received
```

利用 asyncio 库和中断，我们可以实现异步收发。

2 使用 linux sysfs API 的方法

打开~/prio_linux/uart_linux.pynb

PRIO Linux: UART Demo

This demo illustrates how to use device tree overlay's and partial reconfiguration to send messages over uart.

Step One:

Download the static bitstream

In [1]:
```
from prio_linux import PrIoOverlay

FULL_BITSTREAM_PATH = "/usr/local/lib/python3.6/dist-packages/prio_linux/"
PARTIAL_BITSTREAM_PATH = "/usr/local/lib/python3.6/dist-packages/prio_linux/partial/"
DTBO_PATH = "/usr/local/lib/python3.6/dist-packages/prio_linux/dtbo/"

overlay = PrIoOverlay(FULL_BITSTREAM_PATH + "prio_linux.bit")
```

Step Two:

Download the partial bitstream and insert the dtbo files

In [2]:
```
overlay.pr_download("pr_1", PARTIAL_BITSTREAM_PATH + "pr_1_uart.bit", DTBO_PATH + "pr_1_uart.dtbo")

overlay.pr_download("pr_3", PARTIAL_BITSTREAM_PATH + "pr_3_uart.bit", DTBO_PATH + "pr_3_uart.dtbo")
```

首先和上面使用 pynq 的 API 的方法一样，首先下载静态逻辑的 bit 文件，再下载 PR 的 bit 文件。但是不同的是同时注册了对应的 dtbo 到 linux 系统中。dtbo 由 dtso 编译而来，dtso 是描述设备树的文件，关于它的编写参阅以下链接：

https://www.raspberrypi.org/documentation/configuration/device-tree.md

In [3]:
```
import serial, time

# initilize the uart in PR region 1
uart1 = serial.Serial()
uart1.port = "/dev/ttyUL0"
uart1.timeout = 1
uart1.open()
uart1.flushInput()
uart1.flushOutput()

# initilize the uart in PR region 3
uart3 = serial.Serial()
uart3.port = "/dev/ttyUL1"
uart3.timeout = 1
uart3.open()
uart3.flushInput()
uart3.flushOutput()

# Send a message from uart1 to uart3
message = "Sending Data from uart1 to uart3"
uart1.write(message.encode())
recieved = uart3.read(10000)
recieved = recieved.decode()
if recieved == message:
    print("Success! Message Recieved: " + recieved)
else:
    print("Failure: Message Recieved: " + recieved)

# Send a message from uart3 to uart1
message = "Sending Data from uart3 to uart1"
uart3.write(message.encode())
recieved = uart1.read(10000)
recieved = recieved.decode()
if recieved == message:
    print("Success! Message Recieved: " + recieved)
else:
    print("Failure: Message Recieved: " + recieved)


uart1.close()
uart3.close()
```

```
Success! Message Recieved: Sending Data from uart1 to uart3
Success! Message Recieved: Sending Data from uart3 to uart1
```

注册完成后可以直接在/dev 下看到这两个 uart 模块。然后可以使用 linux 自带的 serial 库调用它们。

3 Overlay 详解

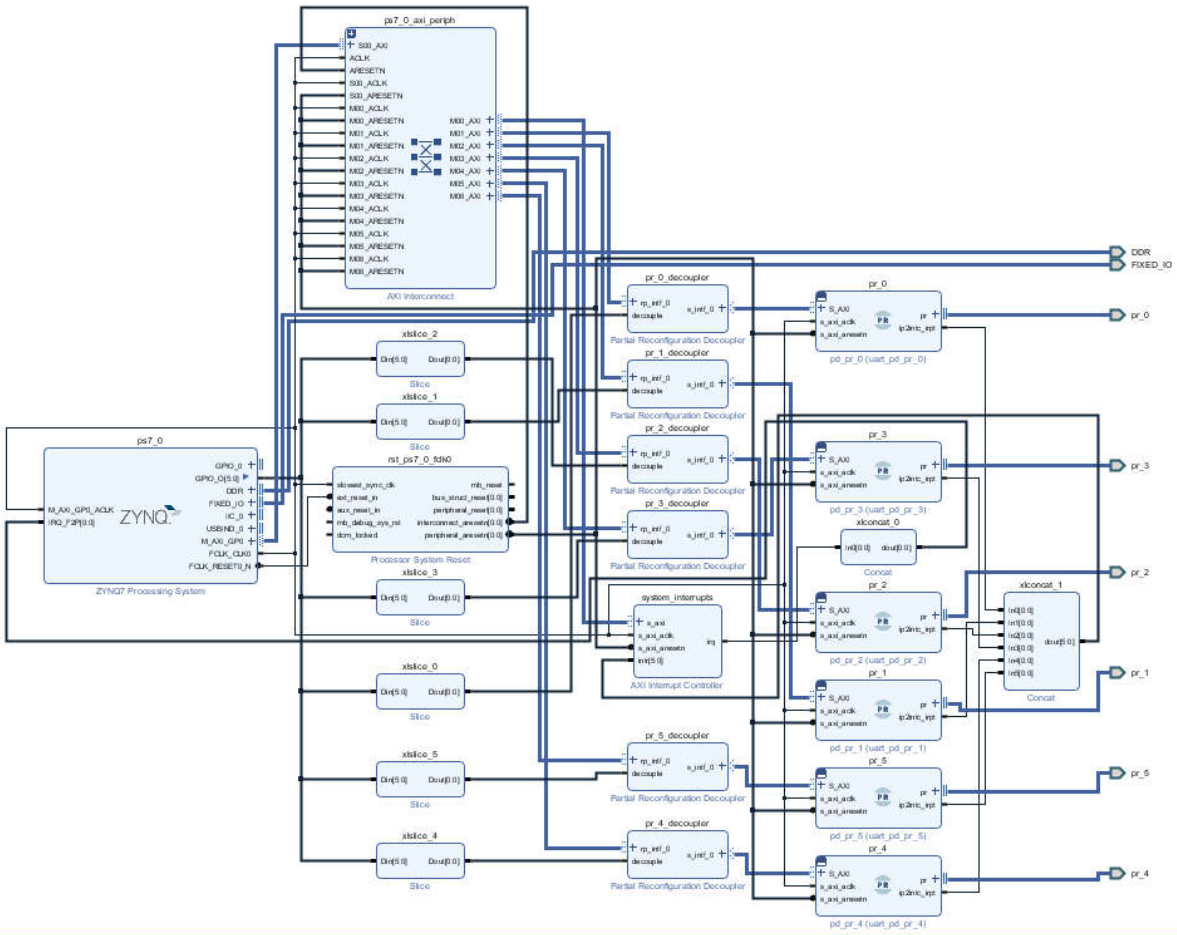
- PS 与 PL 功能划分

  PS 部分主要是运行 linux 系统和下载 bit 文件等。

  PL 部分则设计了六个 PR 块供使用。

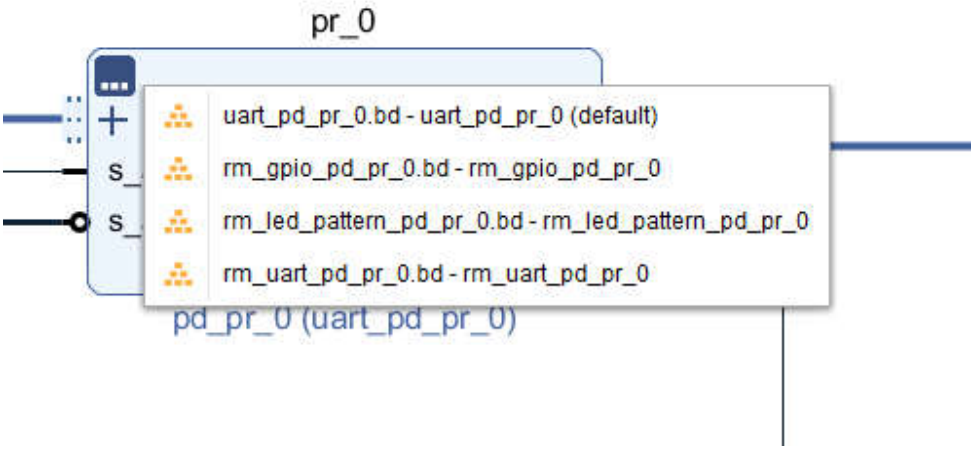
- Vivado 工程 block design 介绍

  具体硬件设计是这样的：（以 prio_linux 为例，prio 类似，只是 PR 中的 RM 略有不同）



本项目中用 decoupler 隔离六个 PR，并使用 PS 的 GPIO 控制 decoupler，使之在 PR 重配置时防止 PR 对其它部分造成影响。

例化了一个中断控制器来接收中断。



每个 PR 中包含四个 RM。

- PL 侧工程重建步骤

  重建工程只需要进入 boards/Pynq-Z1/prio(或 prio_linux)中，在 vivado 中按顺序 source 若干 tcl 文件（以 prio 为例）：

```
source prio.tcl
source create_design.tcl
source create_pr.tcl
source build_bitstream.tcl（直接生成 bit 文件）
source check_prio.tcl（检查时序）
```

  对于 linux 系统下，可以在上述目录下直接运行 make 命令。

  注意，如果希望自己手动创建工程，在建立工程后运行以下命令：

```
set_param project.enablePRFlowIPI 1
set_param bitstream.enablePR 2341
set_param hd.enablePR 1234
```

- PS 侧环境准备（如安装第三方库安装步骤）

```
PYNQ 版本 v2.5
Python 版本 3.6.5
sudo -H pip3 install git+https://github.com/Siudya/PYNQ-PRIO.git
```