UM–SJTU Joint Institute
System-on-Chip Design (ECE4810J)

Laboratory Report

Lab 3. PYNQ Overlays

Group 2

Name: Haochen Wu ID: 518021910558
Name: Siyuan Zhang ID: 518370910180
Name: Yihua Liu ID: 518021910998

Date: October 25, 2021

Contents

1 Overview

In this lab, we will learn about PYNQ overlays. The goals of this lab are to:

- Load overlay, use overlay and create overlay.

2 Loading an overlay

All the available overlays for the Arty Z7 boards:

- Base Overlay
- Logictools Overlay

Under `/usr/local/lib/python3.6/dist-packages/pynq/overlays`, there are `base`, `__init__.py`, `logictools`, and `__pycache__`, so all the available overlays for the Arty Z7 boards are base overlay and logictools overlay [2].

Once we instantiate the base overlay, we use `help(base_overlay)` command to know IPs and methods available, which is:

```
| iop_pmoda : IOP
|     IO processor connected to the PMODA interface
| iop_pmodb : IOP
|     IO processor connected to the PMODB interface
| iop_arduino : IOP
|     IO processor connected to the Arduino/ChipKit interface
| trace_pmoda : pynq.logictools.TraceAnalyzer
|    Trace analyzer block on PMODA interface, controlled by PS.
| trace_arduino : pynq.logictools.TraceAnalyzer
|    Trace analyzer block on Arduino interface, controlled by PS.
| leds : AxiGPIO
|     4-bit output GPIO for interacting with the green LEDs LD0-3
| buttons : AxiGPIO
|     4-bit input GPIO for interacting with the buttons BTN0-3
| switches : AxiGPIO
|     2-bit input GPIO for interacting with the switches SW0 and SW1
| rgbleds : [pynq.board.RGBLED]
|     Wrapper for GPIO for LD4 and LD5 multicolour LEDs
| video : pynq.lib.video.HDMIWrapper
|     HDMI input and output interfaces
| audio : pynq.lib.audio.Audio
|     Headphone jack and on-board microphone
```

Figure 1. Screenshot of available IPs in base overlay

3 Partial Reconfiguration

3.1 Preparing the Files

In this section, we use the demo files provided by the professor, as included in this example [1].

3.2 Loading Full Bitstream

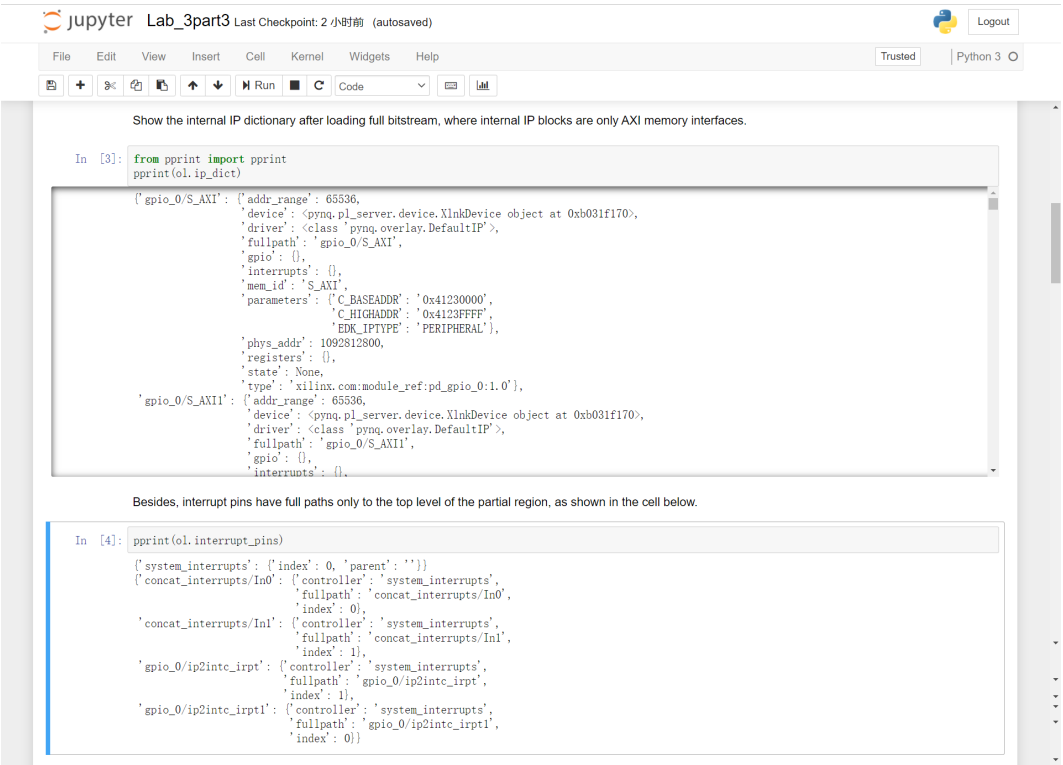


Figure 2. Screenshot that shows full bit stream is loaded.

3.3 Loading Partial Bitstream

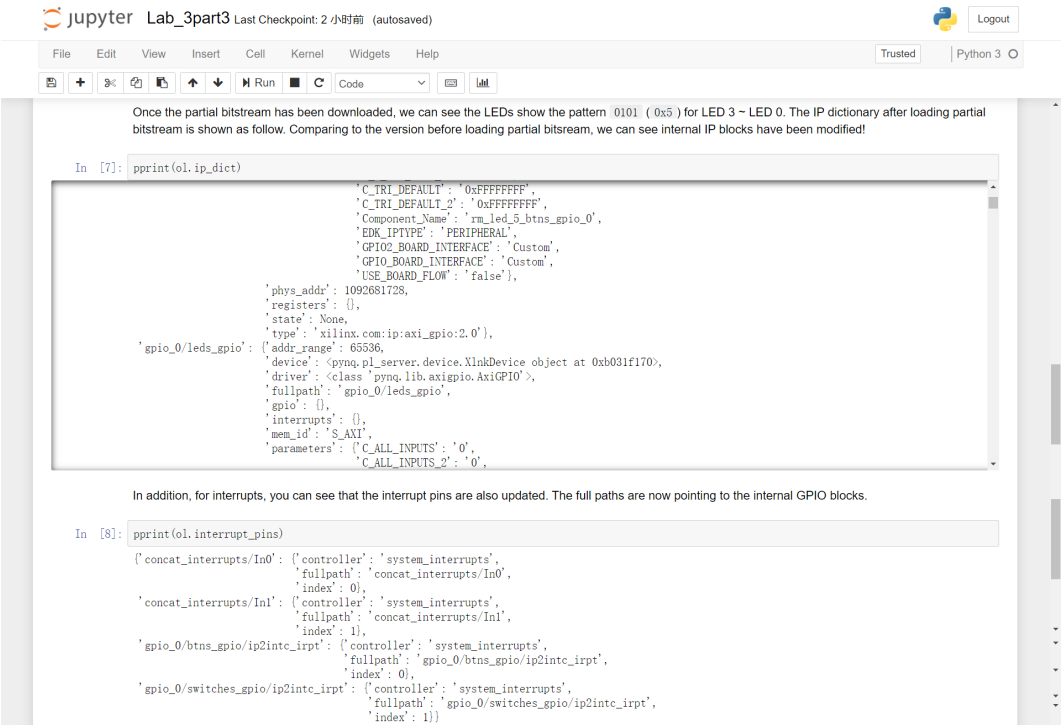


Figure 3. Screenshot that shows partial bit stream is loaded.

After loading partial bitstream, we find that both internal IP block hierarchy and interrupt pins have been modified. Besides, after partial reconfiguration, the LEDs on Zynq board shows pattern of 0101:

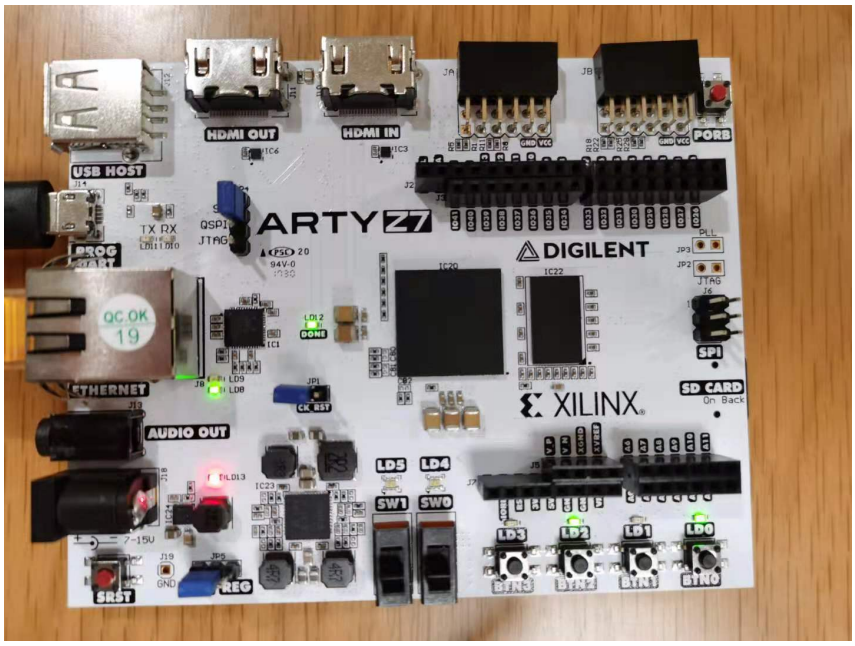
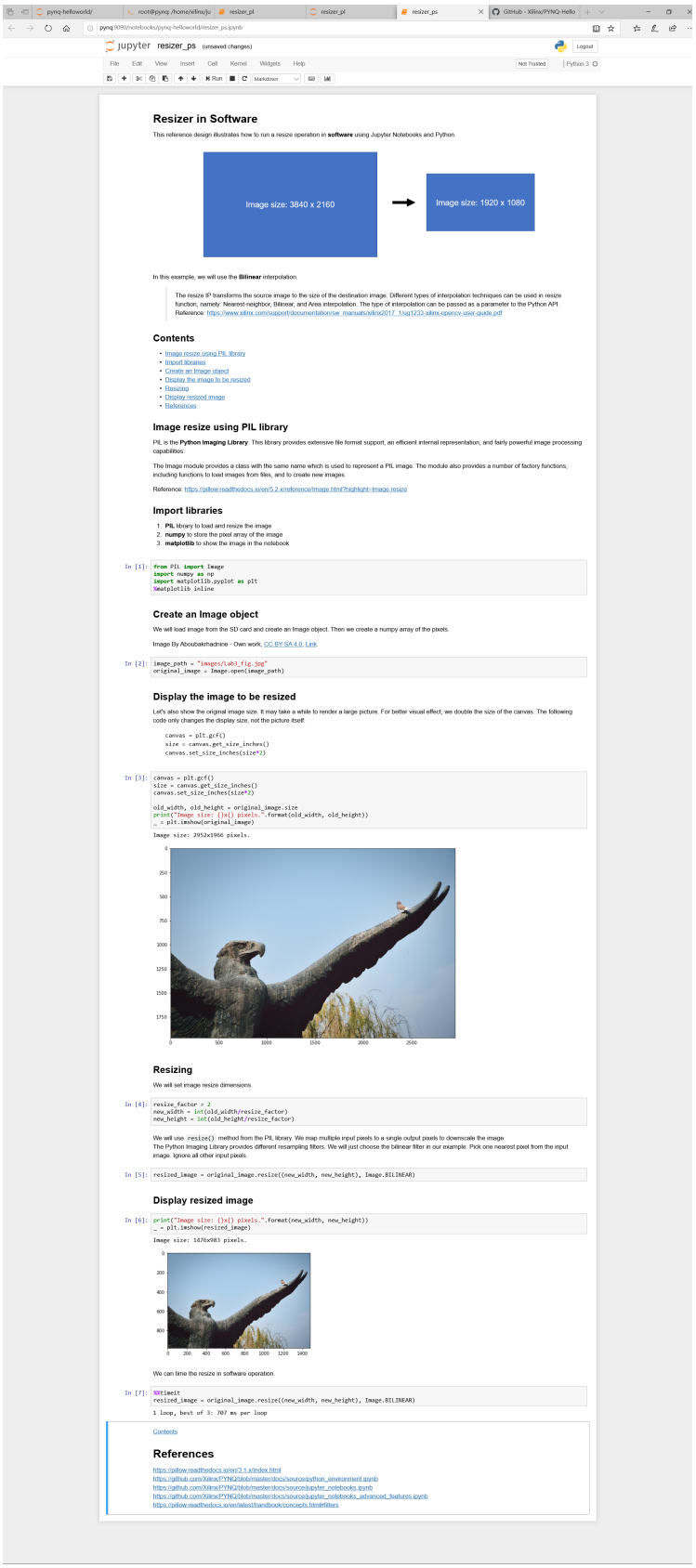


Figure 4. Photo of LED pattern on board after partial reconfiguration

4 PYNQ-Helloworld



Resizer in Software

This reference design illustrates how to run a resize operation in **software** using Jupyter Notebooks and Python.

Image size: 3840 x 2160 → Image size: 1920 x 1080

In this example, we will use the **Bilinear** interpolation.

> The resize IP transforms the source image to the size of the destination image. Different types of interpolation techniques can be used in resize function, namely: Nearest-neighbor, Bilinear, and Area interpolation. The type of interpolation can be passed as a parameter to the Python API.
> Reference: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1233-xilinx-opencv-user-guide.pdf

Contents

Image resize using PIL library

PIL is the **Python Imaging Library**. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The Image module provides a class with the same name which is used to represent a PIL image. The module also provides a number of factory functions, including functions to load images from files, and to create new images.

Reference: https://pillow.readthedocs.io/en/5.2.x/reference/Image.html?highlight=Image.resize

Import libraries

1. **PIL** library to load and resize the image
2. **numpy** to store the pixel array of the image
3. **matplotlib** to show the image in the notebook

In [1]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Create an Image object

We will load image from the SD card and create an Image object. Then we create a numpy array of the pixels.

Image By Aboubakrhadnine - Own work, CC BY-SA 4.0, Link.

In [2]:
```
image_path = "images/Lab3_fig.jpg"
original_image = Image.open(image_path)
```

Display the image to be resized

Let's also show the original image size. It may take a while to render a large picture. For better visual effect, we double the size of the canvas. The following code only changes the display size, not the picture itself:

```
canvas = plt.gcf()
size = canvas.get_size_inches()
canvas.set_size_inches(size*2)
```

In [3]:
```
canvas = plt.gcf()
size = canvas.get_size_inches()
canvas.set_size_inches(size*2)

old_width, old_height = original_image.size
print("Image size: {}x{} pixels.".format(old_width, old_height))
_ = plt.imshow(original_image)
```

Image size: 2952x1966 pixels.

Resizing

We will set image resize dimensions.

In [4]:
```
resize_factor = 2
new_width = int(old_width/resize_factor)
new_height = int(old_height/resize_factor)
```

We will use `resize()` method from the PIL library. We map multiple input pixels to a single output pixels to downscale the image
The Python Imaging Library provides different resampling filters. We will just choose the bilinear filter in our example. Pick one nearest pixel from the input image. Ignore all other input pixels.

In [5]:
```
resized_image = original_image.resize((new_width, new_height), Image.BILINEAR)
```

Display resized image

In [6]:
```
print("Image size: {}x{} pixels.".format(new_width, new_height))
_ = plt.imshow(resized_image)
```

Image size: 1476x983 pixels.

We can time the resize in software operation.

In [7]:
```
%%timeit
resized_image = original_image.resize((new_width, new_height), Image.BILINEAR)
```

1 loop, best of 3: 707 ms per loop

Contents

References

https://pillow.readthedocs.io/en/3.1.x/index.html
https://github.com/Xilinx/PYNQ/blob/master/docs/source/python_environment.ipynb
https://github.com/Xilinx/PYNQ/blob/master/docs/source/jupyter_notebooks.ipynb
https://github.com/Xilinx/PYNQ/blob/master/docs/source/jupyter_notebooks_advanced_features.ipynb
https://pillow.readthedocs.io/en/latest/handbook/concepts.html#filters

Figure 5. A screen shot of the software accelerated resizing.

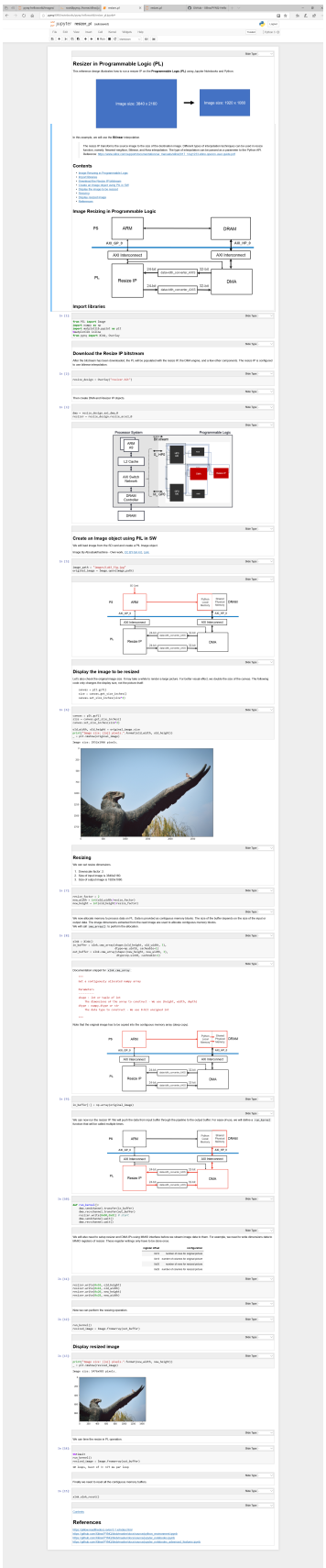


Figure 6. A screen shot of the hardware accelerated resizing.

As we can see in the figures above, the best loop in software is 3.707 ms while the beat loop in hardware acceleration is 3.177 ms.

5 Creating Overlays

The whole overlay package that we created: `https://github.com/yihuajack/PYNQ-Adder`. This Github repository includes our code for part 5.

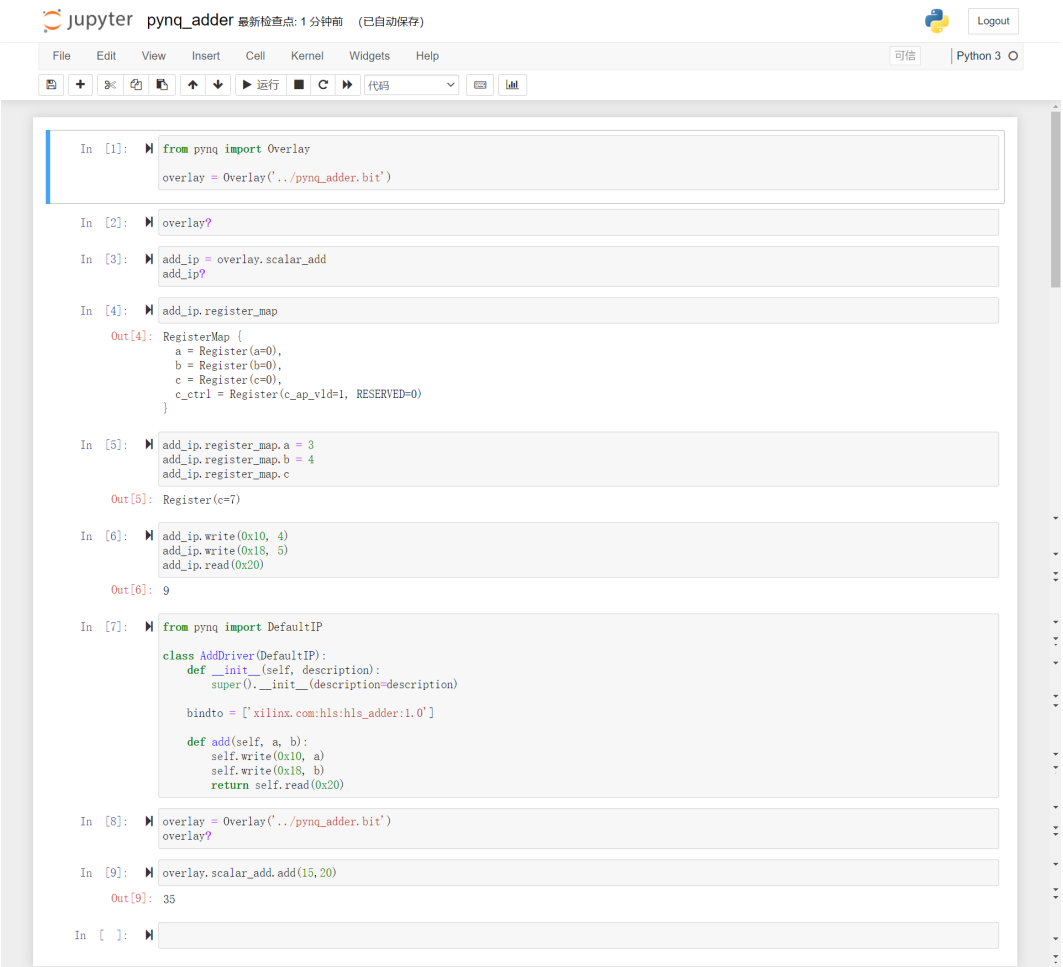


Figure 7. Screenshot that shows your overlay works.

6 Questions

1. Please list out any advantages of using overlay.
   Overlay can be regarded as hardware package. By using it, software engineers can simply use different overlays to handle known problems. It is easy to use without the knowledge of hardware designs.

2. In general, when you choose software approach vs. hardware acceleration approach, what are the considerations? Any tradeoffs?
   Software is a simpler one by just import the corresponding Python module or other software resources. Hardware acceleration needs a designed ip programmed in the programmable logic and use boards like Arty-7z to finish the task. Hardware acceleration is faster if there is a certain deigned overlay. However, it is not always the case. In

contrast, the software acceleration adapts more task requirements and it do not need any external device.

References

[1] PartialReconfig. *Partial reconfiguration example*. URL: https://github.com/yunqu/partial_reconfig_example/tree/master/boards/Pynq-Z1/gpio_pr/notebooks/partial_reconfig.

[2] Xilinx. *Python productivity for Zynq (Pynq)*. Version v2.5.1. Feb. 20, 2020. URL: https://pynq.readthedocs.io/en/v2.5.1/index.html.