WES 237A: Introduction to Embedded System Design (Winter 2026)
Due: 1/11/2026 11:59pm

In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- Upload your lab 1 report composed by a single PDF that includes your in-lab answers to the bolded questions in the Google Doc Lab and your Jupyter Notebook code.
- Answer two short essay-like questions on your Lab experience.

All responses should be submitted to Canvas. Please also be sure to push your code to your git repo as well.

Git Repo Setup

1. Edit your git repo public page to include all of your names, a short bio, and contact emails in the README.md public page. See markdown syntax if needed.

PYNQ Basics

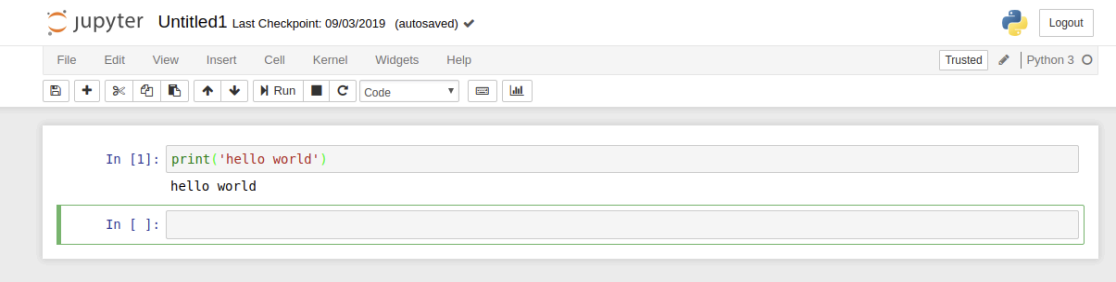
1. Go through the PYNQ Documentation and find the PYNQ Z2 Block Diagram for the Base Overlay
2. **What hardware controls the board peripherals (LEDs, buttons, PMOD headers, etc)?**

dual-core ARM® Cortex®-A9 processor (referred to as the Processing System or PS)

Hello World and LEDs

1. Boot the PYNQ board and connect to your wired private network on 192.168.2.99:9090
2. Select ‘New’ -> ‘Folder’

3. Rename the folder to ‘Lab1’
4. Go into the folder by double clicking and create a ‘New’ -> ‘Python 3’ notebook
5. In the first cell, write ‘print(“Hello World”)’
6. You can run code with the ‘Run’ button at the top, OR by hitting ‘Shift + Enter’ at the same time.



7. Now let's load the base overlay and access some of LEDs
    a. Import the base overlay and time package with

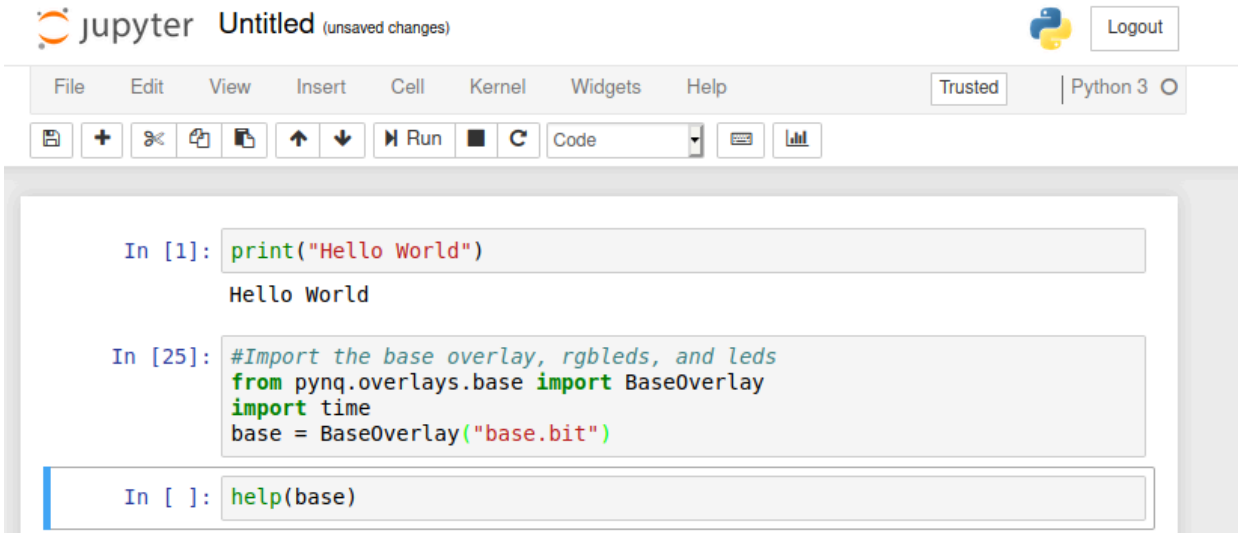
```
from pynq.overlays.base import BaseOverlay
import time
```

    b. Load the base overlay

```
base = BaseOverlay("base.bit")
```

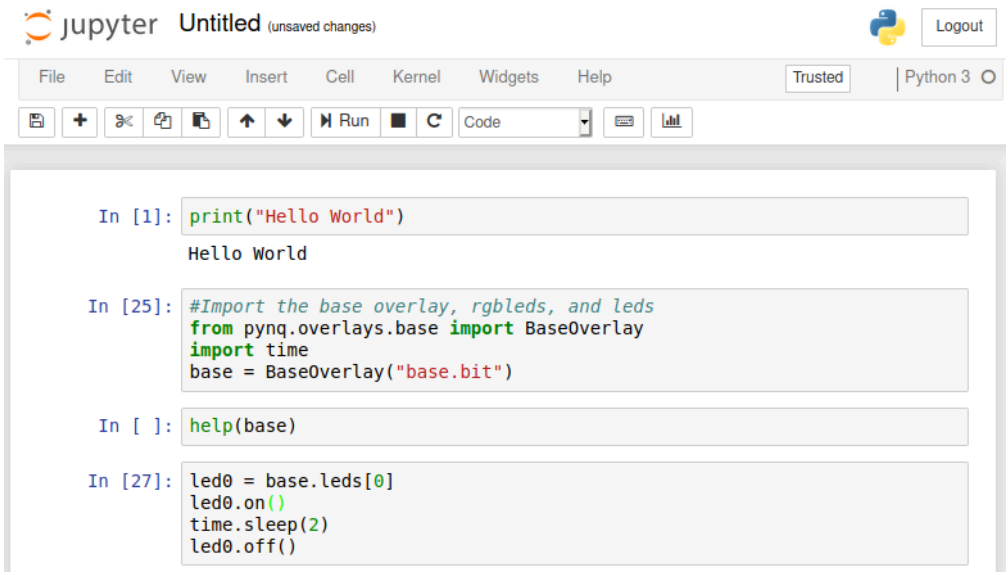
    c. Get the documentation of the base overlay

```
help(base)
```



8. Flash the LEDs with an interval of 2 seconds

```
led0 = base.leds[0]
led0.on()
time.sleep(2)
led0.off()
```



9. Now let's play with the rgb LEDs

```
In [1]: #Now let's deal with the two RGBLEDs
        from pynq.overlays.base import BaseOverlay
        import pynq.lib.rgbled as rgbled
        import time
        base = BaseOverlay("base.bit")

In [ ]: help(rgbled)

In [2]: led4 = rgbled.RGBLED(4)
        led5 = rgbled.RGBLED(5)

In [3]: #RGBLEDs take a hex value for color
        led4.write(0x7)
        led5.write(0x4)

In [4]: led4.write(0x0)
        led5.write(0x0)
```

10. Get a PDF of the jupyter notebook
    a. Go to File->Print Preview then print the print preview page as a PDF
    b. Or try File->Download As->PDF
    c. Only one of the two options needs to work.

ASYNC_IO

1. Download asyncio_example.ipynb from here
2. Upload the asyncio_example.ipynb file to the 'Lab1' folder
3. Open the asyncio_example.ipynb
4. Code is organized into 'cells'. To run the code in a 'cell', select the cell and hit 'Shift + Enter' at the same time. After running a 'cell', you will see [*] which means the code is still executing. Once you see a number in the brackets ([3]), the code has completed.
5. Go through the example code and be able to answer the following with a TA during lab
    a. ***What two lines of code load the FPGA bitstream onto the Programmable Logic (PL) of the PYNQ board?***

```
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
```

    b. ***Describe in your own words the difference between the 'looping' method and the 'async' method.***

The looping method always runs and there is no way to stop it unless you force end the program because the blinking is controlled by a while true loop. Then, periodically, it checks if the button is pressed in order to stop the blinking.
The async method defines the blinking inside a function and then calls the function in a loop, which allows the blinking to be kicked off in its own task and the button interrupt to be called in its own task.

6. Write code in the section 'Lab Work' to start the LED blinking when 'button 0' is pushed and stop when 'button 1' is pushed.

```
# write your code here.
import asyncio

cond = True

async def flash_leds():
    global cond, start
    while cond:
        if start:
            led4.write(0x1)
            led5.write(0x7)
            await asyncio.sleep(0.1)
            led4.write(0x0)
            led5.write(0x0)
            await asyncio.sleep(0.05)
            led4.write(0x1)
            led5.write(0x7)
            await asyncio.sleep(0.1)
            led4.write(0x0)
```

```
            led5.write(0x0)
            await asyncio.sleep(0.05)

            led4.write(0x7)
            led5.write(0x4)
            await asyncio.sleep(0.1)
            led4.write(0x0)
            led5.write(0x0)
            await asyncio.sleep(0.05)
            led4.write(0x7)
            led5.write(0x4)
            await asyncio.sleep(0.1)
            led4.write(0x0)
            led5.write(0x0)
        await asyncio.sleep(0.05)

async def get_btns(_loop):
    global cond, start
    while cond:
        await asyncio.sleep(0.01)
        if btns[0].read() != 0:
            start = False
        if btns[1].read() != 0:
            start = True
        if btns[2].read() != 0:
            _loop.stop()
            cond = False

loop = asyncio.new_event_loop()
loop.create_task(flash_leds())
loop.create_task(get_btns(loop))
loop.run_forever()
loop.close()
led4.write(0x0)
led5.write(0x0)
print("Done.")
```

GPIO

1. Download gpio_example.ipynb from here
2. Upload the gpio_example.ipynb file to the 'Lab1' folder
3. Open the gpio_example.ipynb
4. Go through the example code and be able to answer the following with a TA during lab
    a. ***What is the difference between cells that begin with %%microblaze base.PMODB and cells that don't?***

Microblaze signifies that you want to run on the microblaze subsystem, which is more bare metal and takes C programming instead of python

    b. ***Why do we reload the 'base' overlay in the second part of the notebook?***

We reload the base overlay in the second part of the notebook to reconfigure the FPGA, almost like a reset. That way any peripherals that got left in a weird state, such as writing a 2 to a GPIO pin, gets reset.

5. Write code in the section 'Lab Work' to use two pins (0 and 1) for send and two pins (2 and 3) for receive. You should be able to send 2 bits (0~3) over GPIO. You'll need to hardwire from the send pins to the receive pins.
    a. Start the code by copying 'cells' 1 and 2 from the beginning of the notebook into the 'Lab Work' section.
    b. Then begin editing the %%microblaze cell.

```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

```
%%microblaze base.PMODB

#include "gpio.h"
#include "pyprintf.h"

//Function to turn on/off a selected pin of PMODB
void write_gpio(unsigned int pin, unsigned int val){
    if (val > 1){
        pyprintf("pin value must be 0 or 1");
    }
    gpio pin_out = gpio_open(pin);
    gpio_set_direction(pin_out, GPIO_OUT);
    gpio_write(pin_out, val);
}

//Function to read the value of a selected pin of PMODB
unsigned int read_gpio(unsigned int pin){
    gpio pin_in = gpio_open(pin);
    gpio_set_direction(pin_in, GPIO_IN);
```

```
    return gpio_read(pin_in);
}

unsigned int send_val(unsigned int data_in){
    unsigned int a = data_in & 0x1;
    unsigned int b = (data_in >> 1) & 0x1;

    write_gpio(0, a);
    write_gpio(1, b);

    unsigned int c = read_gpio(2);
    unsigned int d = read_gpio(3);
    unsigned int received = c | (d << 1);

    return received;
}
```