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.

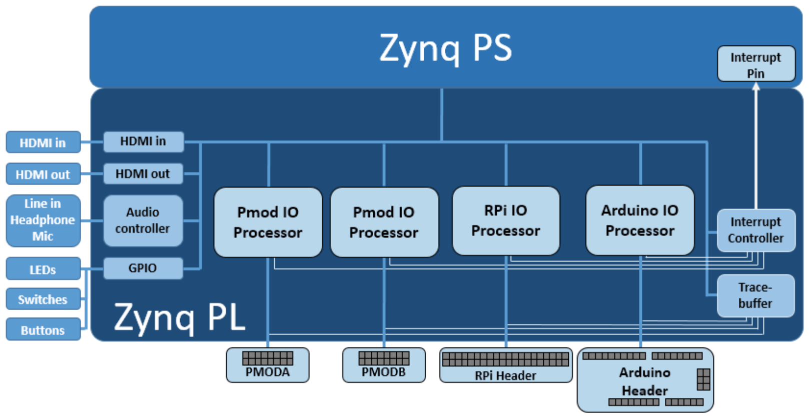
PYNQ Basics

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)?**

The Zync PL controls on-board peripherals like LEDs, buttons, PMOD headers, etc.

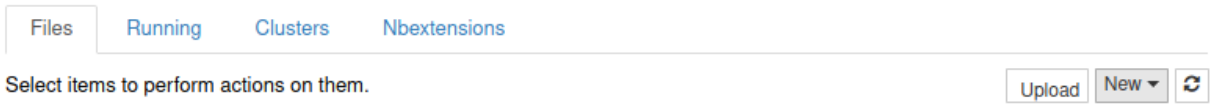
The PL provides the PS (and the end-user via the Jupyter notebooks) an interface to these peripherals.

Diagram from online documentation:



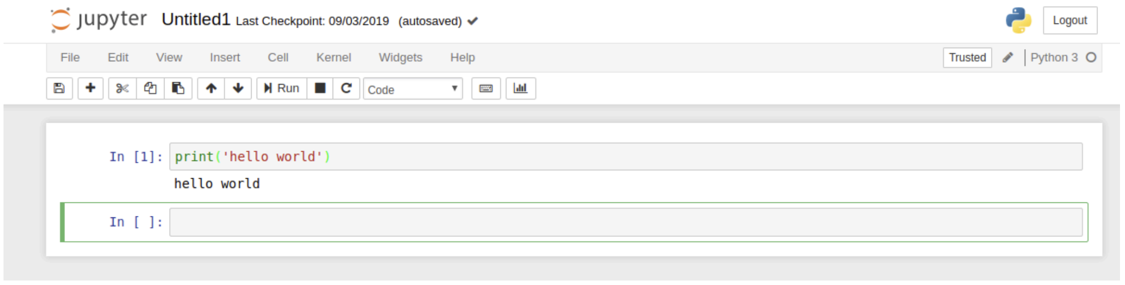
Hello World and LEDs

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



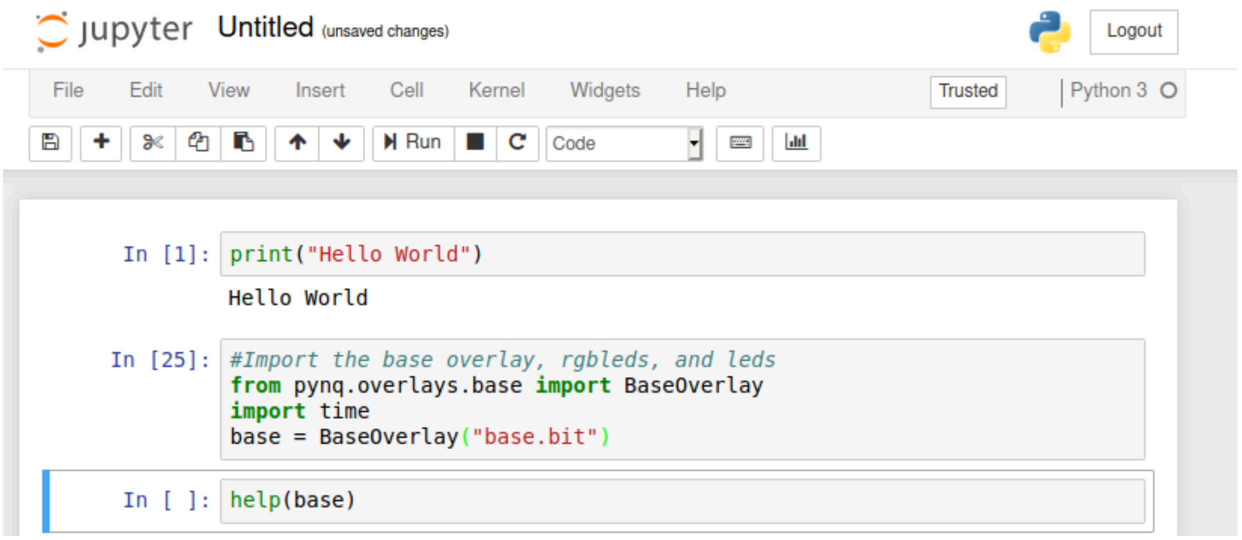
Select items to perform actions on them.

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.



```
In [1]: print('hello world')
        hello world
In [ ]:
```

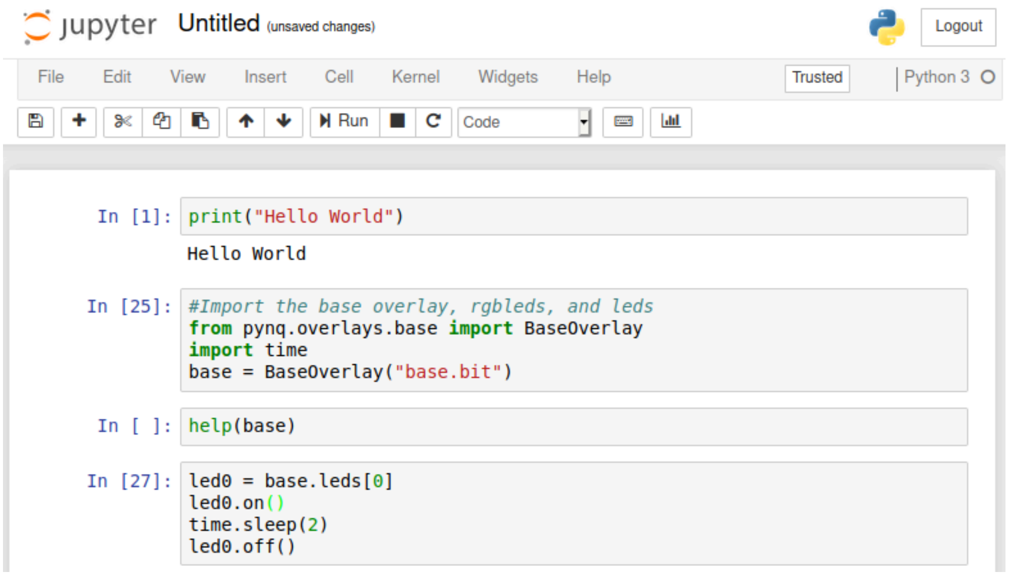
7. Now let's load the base overlay and access some of LEDs
    a. Import the base overlay and time package with
       ```
       from pynq.overlays.base import BaseOverlay
       import time
       ```
    b. Load the base overlay
       ```
       base = BaseOverlay("base.bit")
       ```
    c. Get the documentation of the base overlay
       ```
       help(base)
       ```



```
In [1]: print("Hello World")
        Hello World

In [25]: #Import the base overlay, rgbleds, and leds
         from pynq.overlays.base import BaseOverlay
         import time
         base = BaseOverlay("base.bit")

In [ ]: help(base)
```

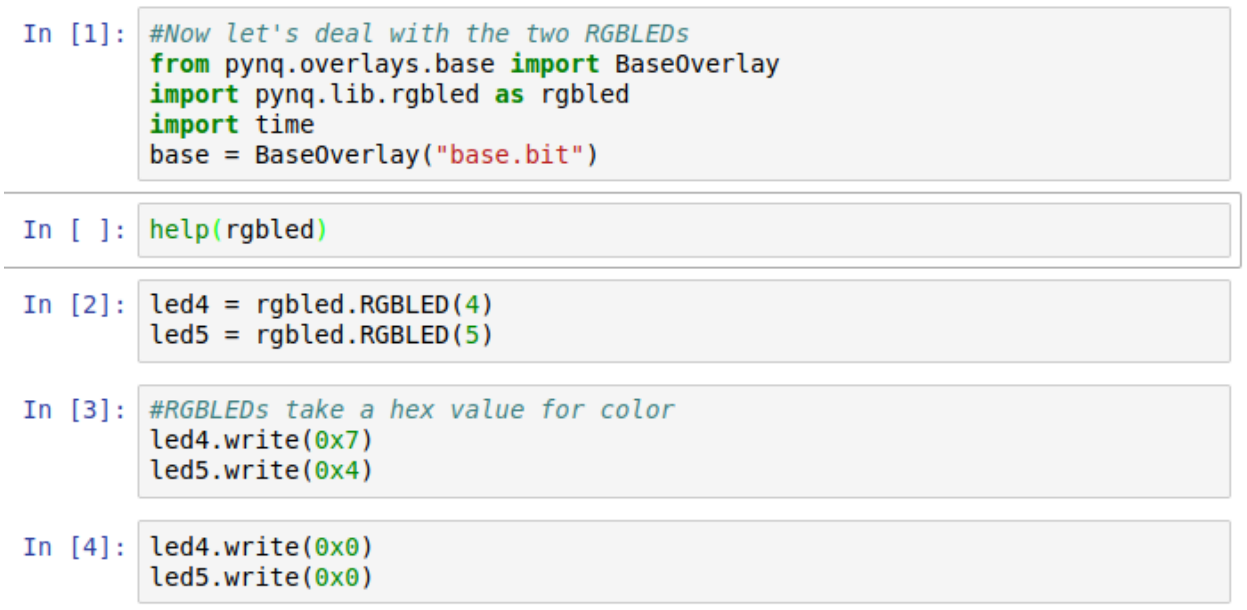
8. Flash the LEDs with an interval of 2 seconds
   ```
   led0 = base.leds[0]
   ```

```
led0.on()
time.sleep(2)
led0.off()
```



```
In [1]: print("Hello World")

Hello World

In [25]: #Import the base overlay, rgbleds, and leds
         from pynq.overlays.base import BaseOverlay
         import time
         base = BaseOverlay("base.bit")

In [ ]: help(base)

In [27]: 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 is a polling based mechanism - where the program sequentially checks for whether the push button was pressed after every pair of LED writes.

The “async” method, using Pynq’s asyncio module, is a parallelized one (akin to an interrupt based mechanism).

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.

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?***

The logic, in cells that begin with %%microblaze, runs on the MicroBlaze soft processor within the PL. The logic is written in C.

The logic, in cells that don't begin with %%microblaze, runs on the PS (ARM Cortex A9). The logic is written in Python.

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

We reload it because the MicroBlaze soft processor is part of the PL and we need to override the logic/bitstream already loaded in the first part.

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.

In [2]:
```
print("Hello World")
```

```
Hello World
```

In [7]:
```
from pynq.overlays.base import BaseOverlay
import time

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

```
Help on BaseOverlay in module pynq.overlays.base.base:

<pynq.overlays.base.base.BaseOverlay object>
    Default documentation for overlay base.bit. The following
    attributes are available on this overlay:

    IP Blocks
    ----------
    switches_gpio          : pynq.lib.axigpio.AxiGPIO
    btns_gpio              : pynq.lib.axigpio.AxiGPIO
    video/hdmi_in/frontend/axi_gpio_hdmiin : pynq.lib.axigpio.AxiGPIO
    video/hdmi_out/frontend/hdmi_out_hpd_video : pynq.lib.axigpio.AxiGPIO
    rgbleds_gpio           : pynq.lib.axigpio.AxiGPIO
    leds_gpio              : pynq.lib.axigpio.AxiGPIO
    system_interrupts      : pynq.overlay.DefaultIP
    video/axi_vdma         : pynq.lib.video.dma.AxiVDMA
    audio_codec_ctrl_0     : pynq.lib.audio.AudioADAU1761
    video/hdmi_out/frontend/axi_dynclk : pynq.overlay.DefaultIP
    video/hdmi_out/frontend/vtc_out : pynq.overlay.DefaultIP
    video/hdmi_in/frontend/vtc_in : pynq.overlay.DefaultIP
    video/hdmi_in/pixel_pack : pynq.lib.video.pipeline.PixelPacker
    video/hdmi_in/color_convert : pynq.lib.video.pipeline.ColorConverter
    video/hdmi_out/color_convert : pynq.lib.video.pipeline.ColorConverter
    video/hdmi_out/pixel_unpack : pynq.lib.video.pipeline.PixelPacker
    trace_analyzer_pmodb/axi_dma_0 : pynq.lib.dma.DMA
    trace_analyzer_pi/axi_dma_0 : pynq.lib.dma.DMA
    trace_analyzer_pi/trace_cntrl_64_0 : pynq.overlay.DefaultIP
    trace_analyzer_pmodb/trace_cntrl_32_0 : pynq.overlay.DefaultIP
    ps7_0                  : pynq.overlay.DefaultIP

    Hierarchies
    -----------
    iop_arduino            : pynq.lib.pynqmicroblaze.pynqmicroblaze.Microblaze
Hierarchy
    iop_pmoda              : pynq.lib.pynqmicroblaze.pynqmicroblaze.Microblaze
Hierarchy
    iop_pmodb              : pynq.lib.pynqmicroblaze.pynqmicroblaze.Microblaze
Hierarchy
    iop_rpi                : pynq.lib.pynqmicroblaze.pynqmicroblaze.Microblaze
Hierarchy
    trace_analyzer_pi      : pynq.overlay.DefaultHierarchy
    trace_analyzer_pmodb : pynq.overlay.DefaultHierarchy
    video                   : pynq.lib.video.hierarchies.HDMIWrapper
    video/hdmi_in           : pynq.lib.video.hierarchies.VideoIn
    video/hdmi_in/frontend  : pynq.lib.video.dvi.HDMIInFrontend
    video/hdmi_out          : pynq.lib.video.hierarchies.VideoOut
    video/hdmi_out/frontend : pynq.lib.video.dvi.HDMIOutFrontend

    Interrupts
    ----------
    None

    GPIO Outputs
    ------------
    None

    Memories
    ------------
    iop_pmodamb_bram_ctrl : Memory
    iop_pmodbmb_bram_ctrl : Memory
    iop_arduinomb_bram_ctrl : Memory
    iop_rpimb_bram_ctrl   : Memory
    PSDDR                 : Memory
```

In [10]:

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

In [11]:

```
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled
import time

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

```
Help on module pynq.lib.rgbled in pynq.lib:

NAME
    pynq.lib.rgbled

DESCRIPTION
    #   Copyright (c) 2016, Xilinx, Inc.
    #   SPDX-License-Identifier: BSD-3-Clause

CLASSES
    builtins.object
        RGBLED

    class RGBLED(builtins.object)
     |  RGBLED(index, ip_name='rgbleds_gpio', start_index=inf, device=None)
     |
     |  This class controls the onboard RGB LEDs.
     |
     |  Attributes
     |  ----------
     |  index : int
     |      The index of the RGB LED. Can be an arbitrary value.
     |  _mmio : MMIO
     |      Shared memory map for the RGBLED GPIO controller.
     |  _rgbleds_val : int
     |      Global value of the RGBLED GPIO pins.
     |  _rgbleds_start_index : int
     |      Global value representing the lowest index for RGB LEDs
     |
     |  Methods defined here:
     |
     |  __init__(self, index, ip_name='rgbleds_gpio', start_index=inf, devic
e=None)
     |      Create a new RGB LED object.
     |
     |      Parameters
     |      ----------
     |      index : int
     |          Index of the RGBLED, Can be an arbitrary value.
     |          The smallest index given will set the global value
     |          `_rgbleds_start_index`. This behavior can be overridden by d
efining
     |          `start_index`.
     |      ip_name : str
     |          Name of the IP in  the `ip_dict`. Defaults to "rgbleds_gpi
o".
     |      start_index : int
     |          If defined, will be used to update the global value
     |          `_rgbleds_start_index`.
     |
     |  off(self)
     |      Turn off a single RGBLED.
     |
     |      Returns
     |      -------
     |      None
     |
     |  on(self, color)
     |      Turn on a single RGB LED with a color value (see color constant
s).
     |
     |      Parameters
     |      ----------
     |      color : int
     |         Color of RGB specified by a 3-bit RGB integer value.
     |
     |      Returns
     |      -------
     |      None
     |
     |  read(self)
```

```
     |      Retrieve the RGBLED state.
     |      
     |      Returns
     |      -------
     |      int
     |          The color value stored in the RGBLED.
     |  
     |  write(self, color)
     |      Set the RGBLED state according to the input value.
     |      
     |      Parameters
     |      ----------
     |      color : int
     |          Color of RGB specified by a 3-bit RGB integer value.
     |      
     |      Returns
     |      -------
     |      None
     |  
     |  --------------------------------------------------------------------
--
     |  Data descriptors defined here:
     |  
     |  __dict__
     |      dictionary for instance variables (if defined)
     |  
     |  __weakref__
     |      list of weak references to the object (if defined)

DATA
    RGBLEDS_XGPIO_OFFSET = 0
    RGB_BLUE = 1
    RGB_CLEAR = 0
    RGB_CYAN = 3
    RGB_GREEN = 2
    RGB_MAGENTA = 5
    RGB_RED = 4
    RGB_WHITE = 7
    RGB_YELLOW = 6

FILE
    /usr/local/share/pynq-venv/lib/python3.10/site-packages/pynq/lib/rgbled.
py
```

In [15]:

```python
led4 = rgbled.RGBLED(4)
led5 = rgbled.RGBLED(5)

led4.write(0x7)
led5.write(0x4)
```

In [16]:
```
led4.write(0x0)
led5.write(0x0)
```

In [ ]:
```
```

Importing some libraries

In [2]:
```
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled
import time
```

Programming the PL

In [3]:
```
base = BaseOverlay("base.bit")
```

Defining buttons and LEDs

In [4]:
```
btns = base.btns_gpio
led4 = rgbled.RGBLED(4)
led5 = rgbled.RGBLED(5)
```

Using a loop to blink the LEDS and read from buttons

In [4]:
```
while True:
    led4.write(0x1)
    led5.write(0x7)
    if btns.read() != 0:
        break
    time.sleep(0.1)
    led4.write(0x0)
    led5.write(0x0)
    if btns.read() != 0:
        break
    time.sleep(0.05)
    led4.write(0x1)
    led5.write(0x7)
    if btns.read() != 0:
        break
    time.sleep(0.1)
    led4.write(0x0)
    led5.write(0x0)
    if btns.read() != 0:
        break
    time.sleep(0.05)

    led4.write(0x7)
    led5.write(0x4)
    if btns.read() != 0:
        break
    time.sleep(0.1)
    led4.write(0x0)
    led5.write(0x0)
    if btns.read() != 0:
        break
    time.sleep(0.05)
    led4.write(0x7)
    led5.write(0x4)
    if btns.read() != 0:
        break
    time.sleep(0.1)
    led4.write(0x0)
    led5.write(0x0)
    if btns.read() != 0:
        break
    time.sleep(0.05)

led4.write(0x0)
led5.write(0x0)
```

Using asyncio to blink the LEDS and read from buttons

In [5]:
```
import asyncio
cond = True

async def flash_leds():
    global cond, start
    while cond:
        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.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.")
```

```
Done.
```

Lab work

Using the code from previous cell as a template, write a code to start the blinking when button 0 is pushed and stop the blinking when button 1 is pushed.

In [7]:
```
# Using loop

def flash_leds():
    btn1 = btns[1]
    while True:
        led4.write(0x1)
        led5.write(0x7)
        if btn1.read() != 0:
            break
        time.sleep(0.1)
        led4.write(0x0)
        led5.write(0x0)
        if btn1.read() != 0:
            break
        time.sleep(0.05)
        led4.write(0x1)
        led5.write(0x7)
        if btn1.read() != 0:
            break
        time.sleep(0.1)
        led4.write(0x0)
        led5.write(0x0)
        if btn1.read() != 0:
            break
        time.sleep(0.05)

        led4.write(0x7)
        led5.write(0x4)
        if btn1.read() != 0:
            break
        time.sleep(0.1)
        led4.write(0x0)
        led5.write(0x0)
        if btn1.read() != 0:
            break
        time.sleep(0.05)
        led4.write(0x7)
        led5.write(0x4)
        if btn1.read() != 0:
            break
        time.sleep(0.1)
        led4.write(0x0)
        led5.write(0x0)
        if btn1.read() != 0:
                break
            time.sleep(0.05)

    led4.write(0x0)
    led5.write(0x0)

def poll_btn0():
    btn0 = btns[0]
    while 0 == btn0.read():
        time.sleep(0.05)

while True:
    poll_btn0();
    flash_leds();
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [7], in <cell line: 56>()
     54         time.sleep(0.05)
     56 while True:
---> 57     poll_btn0();
     58     flash_leds()

Input In [7], in poll_btn0()
     52 btn0 = btns[0]
     53 while 0 == btn0.read():
---> 54     time.sleep(0.05)

KeyboardInterrupt: 
```

In [6]:

```
# Using asyncio
import asyncio

btn0 = btns[0]
btn1 = btns[1]
cond = False

async def flash_leds():
    global cond, start
    while True:
        if cond:
            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)
        else:
            led4.write(0x0)
            led5.write(0x0)
            await asyncio.sleep(0.05)

async def get_btns():
    global cond, start
    while True:
        await asyncio.sleep(0.01)
        if btn0.read() != 0:
            cond = True
        if btn1.read() != 0:
            cond = False

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

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [6], in <cell line: 54>()
     52 loop.create_task(flash_leds())
     53 loop.create_task(get_btns())
---> 54 loop.run_forever()
     55 loop.close()
     56 led4.write(0x0)

File /usr/local/share/pynq-venv/lib/python3.10/site-packages/nest_asyncio.p
y:72, in _patch_loop.<locals>.run_forever(self)
     70 with manage_run(self), manage_asyncgens(self):
     71     while True:
---> 72         self._run_once()
     73         if self._stopping:
     74             break

File /usr/local/share/pynq-venv/lib/python3.10/site-packages/nest_asyncio.p
y:106, in _patch_loop.<locals>._run_once(self)
     99     heappop(scheduled)
    101 timeout = (
    102     0 if ready or self._stopping
    103     else min(max(
    104         scheduled[0]._when - self.time(), 0), 86400) if scheduled
    105     else None)
--> 106 event_list = self._selector.select(timeout)
    107 self._process_events(event_list)
    109 end_time = self.time() + self._clock_resolution

File /usr/lib/python3.10/selectors.py:469, in EpollSelector.select(self, tim
eout)
    467 ready = []
    468 try:
--> 469     fd_event_list = self._selector.poll(timeout, max_ev)
    470 except InterruptedError:
    471     return ready

KeyboardInterrupt: 
```

```
In [ ]:
```

Interacting with GPIO from MicroBlaze

In [1]:
```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [2]:
```
%%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);
}
```

In [3]:
```
write_gpio(0, 2)
read_gpio(1)
```

```
pin value must be 0 or 1
```

Out[3]:
```
1
```

In [4]:
```
# Testing
write_gpio(0, 1)
read_gpio(1)
```

Out[4]:
```
1
```

In [5]:
```
# Testing
write_gpio(0, 0)
read_gpio(1)
```

Out[5]:

```
1
```

Multi-tasking with MicroBlaze

In [6]:

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

In [7]:

```
%%microblaze base.PMODA

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

//Function to turn on/off a selected pin of PMODA
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 PMODA
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);
}

//Multitasking the microblaze for a simple function
int add(int a, int b){
    return a + b;
}
```

In [8]:

```
val = 1
write_gpio(0, val)
read_gpio(1)
```

Out[8]:

```
1
```

In [9]:

```
add(2, 30)
```

Out[9]:

```
32
```

Lab work

Use the code from the second cell as a template and write a code to use two pins (0 and 1) for send and two pins (2 and 3) for receive. You should be able to send 2bits (0~3) over GPIO. You'll need to hardwire from the send pins to the receive pins.

In [10]:

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

In [11]:

```
%%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);
}
```

In [12]:

```
# Going to be looping the 2 bits through 00, 01, 10 and 11

for i in range(0, 2):
    for j in range(0, 2):
        write_gpio(0, i)
        write_gpio(1, j)
        print(f"GPIO 0 and 1 set to: (%d, %d)" % (i, j));
        print(f"GPIO 2 and 3: (%d, %d)\n" % (int(read_gpio(2)), int(read_gpi
GPIO 0 and 1 set to: (0, 0)
GPIO 2 and 3: (0, 0)

GPIO 0 and 1 set to: (0, 1)
GPIO 2 and 3: (0, 1)

GPIO 0 and 1 set to: (1, 0)
GPIO 2 and 3: (1, 0)

GPIO 0 and 1 set to: (1, 1)
GPIO 2 and 3: (1, 1)
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```