PYNQ BOARD SETUP

Introduction:
In order to get ourselves slightly acquainted with the PYNQ board and its programming using the Jupyter notebook, we ran a few given examples on the board. For the initial setup, we loaded the latest PYNQ image (V2.4) onto the SD card and got the board working with the internet setup.

LED Demo:
To get a basic understanding we ran a simple LED blinking code on the PYNQ board. We used the given base overlay that contains the drivers for each of the modules present on the FPGA board. We configured the buttons on the board to drive different patterns on the LEDs and to control the colour of the RGB Leds. The behaviour modeled was as follows:

Button 0 pressed:	RGB LEDs change color.
Button 1 pressed:	LEDs shift from right to left (LD0 -> LD3).
Button 2 pressed:	LEDs shift from left to right (LD3 -> LD0).
Button 3 pressed:	Turns off all the LEDS and ends this demo.

The video demonstrating the working of this code on the PYNQ board:
https://drive.google.com/a/eng.ucsd.edu/file/d/1sa4ouarhqxSUnrwBy1JqdTcgipwRgju2/view?usp=sharing

Buffer Example:
We ran a simple example to understand how a numpy array on the ARM processor can be readily made accessible to the programmable logic. For this, the numpy array must be stored in memory that is accessible by both the processor and the programmable logic. The physical address of the buffer will have to be passed to the programmable logic. Using this address, the programmable logic can now communicate with the buffer and can manipulate its contents as needed.

Snippets of the Python notebook used to run this example have been given below:

In [1]:
```
import numpy as np
import pynq

def get_pynq_buffer(shape, dtype):
    """ Simple function to call PYNQ's memory allocator with numpy attributes

    """
    return pynq.Xlnk().cma_array(shape, dtype)
```

With the simple wrapper above, we can get access to memory that can be shared by both numpy methods and programmable logic.

In [2]:
```
buffer = get_pynq_buffer(shape=(4,4), dtype=np.uint32)
buffer
```

Out[2]:
```
ContiguousArray([[0, 0, 0, 0],
                 [0, 0, 0, 0],
                 [0, 0, 0, 0],
                 [0, 0, 0, 0]], dtype=uint32)
```

To double-check we show that the buffer is indeed a numpy array.

In [3]:
```
isinstance(buffer,np.ndarray)
```

Out[3]:
```
True
```

To send the buffer pointer to programmable logic, we use its physical address which is what programmable logic would need to communicate using this shared buffer.

In [4]:
```
pl_buffer_address = hex(buffer.physical_address)
pl_buffer_address
```

Out[4]:
```
'0x18048000'
```

Conclusion:

After demonstrating that the board works correctly, we decided to continue with PYNQ board after we have the baseline implementation working in C/C++.