In [4]:
```
print("Hello World")
```

```
Hello World
```

In [5]:
```
# Import the base overlay and time package with
from pynq.overlays.base import BaseOverlay
import time
base = BaseOverlay("base.bit")
```

In [6]:
```
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 [11]:
```
led0 = base.leds[0]
led0.on()
time.sleep(2)
led0.off()
```

In [12]:
```
# Dealing with two (2) RGB LEDs
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled
import time
base = BaseOverlay("base.bit")
```

In [13]:
```
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/rgble
d.py
```

In [14]:
```
led4 = rgbled.RGBLED(4)
led5 = rgbled.RGBLED(5)
```

In [17]:
```
# RGB LEDs take a hex value for color
led4.write(0x7)
led5.write(0x4)
```

In [19]:
```
led4.write(0x0)
led5.write(0x0)
```

Importing some libraries

In [1]:
```
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled
import time
```

Programming the PL

In [2]:
```
base = BaseOverlay("base.bit")
```

Defining buttons and LEDs

In [3]:
```
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 [10]:
```
# write your code here.
import asyncio
cond = True
flag = False

async def flash_leds():
    global cond, flag
    while cond:
        if flag:
            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(0.01)

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

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.
```

In [ ]:

In [ ]:

Interacting with GPIO from MicroBlaze

In [3]:
```
from pynq.overlays.base import BaseOverlay
import time
from datetime import datetime
base = BaseOverlay("base.bit")
```

In [4]:
```
%%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 [5]:
```
write_gpio(0, 2)
read_gpio(1)
```

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

Out[5]:
```
1
```

Multi-tasking with MicroBlaze

In [4]:
```
base = BaseOverlay("base.bit")
```

In [5]:
```
%%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 [6]:
```
val = 1
write_gpio(0, val)
read_gpio(1)
```

Out[6]: 1

In [7]:
```
add(2, 30)
```

Out[7]: 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 [7]:
```
%%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);
}

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

In [10]:
```
val0 = 1
val1 = 0
write_gpio(0, val0)
write_gpio(1, val1)
print(read_gpio(2))
print(read_gpio(3))
```

```
1
0
```

In [ ]:

Lab Questions:

1. The board's peripherals are all part of the Zynq PL (programmable logic), which is controlled by the ARM Dual Cortex-A9 MPCore System (CPU).
2. The two lines of code that load the FPGA bitstream onto the PL are:
   ```
   from pynq.overlays.base import BaseOverlay
   import pynq.lib.rgbled as rgbled
   ```
3. The main difference between these two methods is that to check for inputs, you will have to constantly hard-code a check for a button input every few lines, whereas the asyncio method lets you write out the general logic of the code, with a small section afterwards the code jumps to whenever an interrupt (button input in this case) happens. The asyncio method is much easier to read and program as a result.
4. The main difference between these cells is that the cells starting with "`%%microblaze base.PMODB`" are written in C, whereas the other cells are written in Python.
5. We need to reload the base overlay because the second part of the notebook deals with PmodA instead of PmodB.