asyncio_example

January 12, 2026

1 Importing some libraries

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

2 Programming the PL

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

3 Defining buttons and LEDs

[3]:
```
btns = base.btns_gpio
led4 = rgbled.RGBLED(4)
led5 = rgbled.RGBLED(5)
```

4 Using a loop to blink the LEDS and read from buttons

[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)
```

5 Using asyncio to blink the LEDS and read from buttons

[6]:
```
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.
```

6 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.

[7]:
```
help(btns)
```

```
Help on AxiGPIO in module pynq.lib.axigpio object:

class AxiGPIO(pynq.overlay.DefaultIP)
```

```
|  AxiGPIO(description)
|
|  Class for interacting with the AXI GPIO IP block.
|
|  This class exposes the two banks of GPIO as the `channel1` and
|  `channel2` attributes. Each channel can have the direction and
|  the number of wires specified.
|
|  The wires in the channel can be accessed from the channel using
|  slice notation - all slices must have a stride of 1. Input wires
|  can be `read` and output wires can be written to, toggled, or
|  turned off or on. InOut channels combine the functionality of
|  input and output channels. The tristate of the pin is determined
|  by whether the pin was last read or written.
|
|  Method resolution order:
|      AxiGPIO
|      pynq.overlay.DefaultIP
|      builtins.object
|
|  Methods defined here:
|
|  __getitem__(self, idx)
|
|  __init__(self, description)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  setdirection(self, direction, channel=1)
|      Sets the direction of a channel in the controller
|
|      Must be one of AxiGPIO.{Input, Output, InOut} or the string
|      'in', 'out' or 'inout'
|
|  setlength(self, length, channel=1)
|      Sets the length of a channel in the controller
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  Channel = <class 'pynq.lib.axigpio.AxiGPIO.Channel'>
|      Class representing a single channel of the GPIO controller.
|
|      Wires are and bundles of wires can be accessed using array notation
|      with the methods on the wires determined by the type of the channel::
|
|          input_channel[0].read()
|          output_channel[1:3].on()
|
```

```
|      This class instantiated not used directly, instead accessed through
|      the `AxiGPIO` classes attributes. This class exposes the wires
|      connected to the channel as an array or elements. Slices of the
|      array can be assigned simultaneously.
|
|
|  InOut = <class 'pynq.lib.axigpio.AxiGPIO.InOut'>
|      Class representing wires in an inout channel.
|
|      This class should be passed to `setdirection` to indicate the
|      channel should be used for both input and output. It should not
|      be used directly.
|
|
|  Input = <class 'pynq.lib.axigpio.AxiGPIO.Input'>
|      Class representing wires in an input channel.
|
|      This class should be passed to `setdirection` to indicate the
|      channel should be used for input only. It should not be used
|      directly.
|
|
|  Output = <class 'pynq.lib.axigpio.AxiGPIO.Output'>
|      Class representing wires in an output channel.
|
|      This class should be passed to `setdirection` to indicate the
|      channel should be used for output only. It should not be used
|      directly.
|
|
|  bindto = ['xilinx.com:ip:axi_gpio:2.0']
|
|  ----------------------------------------------------------------------
|  Methods inherited from pynq.overlay.DefaultIP:
|
|  read(self, offset=0)
|      Read from the MMIO device
|
|      Parameters
|      ----------
|      offset : int
|          Address to read
|
|  write(self, offset, value)
|      Write to the MMIO device
|
|      Parameters
|      ----------
```

```
|      offset : int
|          Address to write to
|      value : int or bytes
|          Data to write
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from pynq.overlay.DefaultIP:
|
|  register_map
|
|  signature
|      The signature of the `call` method
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from pynq.overlay.DefaultIP:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

[8]:
```
dir(btns)
```

[8]:
```
['Channel',
 'InOut',
 'Input',
 'Output',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
```

```
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_call',
 '_channels',
 '_fullpath',
 '_gpio',
 '_interrupts',
 '_register_name',
 '_registers',
 'bindto',
 'channel1',
 'channel2',
 'device',
 'has_interrupts',
 'ip2intc_irpt',
 'mmio',
 'read',
 'register_map',
 'setdirection',
 'setlength',
 'signature',
 'write']
```

[21]:
```
import asyncio
cond = True

async def flash_leds():
    global cond, start
    start=True
    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)
        else:
            led4.write(0x0)
            led5.write(0x0)
            await asyncio.sleep(0.01)

async def get_btns(_loop):
    global cond, start
    while cond:
        await asyncio.sleep(0.01)
        if btns[0].read():
            start=True
        if btns[1].read():
            start=False
        if btns[2].read() or btns[3].read(): #terminate
            _loop.stop()
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.
```

[ ]: