Using PYNQ library for PMOD_ADC

This just uses the built in Pmod_ADC library to read the value on the PMOD_AD2 peripheral.

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

In [2]:
```
adc = Pmod_ADC(base.PMODA)
```

Read the raw value and the 12 bit values from channel 1.

Refer to docs:
https://pynq.readthedocs.io/en/v2.1/pynq_package/pynq.lib/pynq.lib.pmod.html#pynq-lib-pmod

In [3]:
```
adc.read_raw(ch1=1, ch2=0, ch3=0)
```

Out[3]:
```
[768]
```

In [4]:
```
adc.read(ch1=1, ch2=0, ch3=0)
```

Out[4]:
```
[1.0029]
```

Using MicroblazeLibrary

Here we're going down a level and using the microblaze library to write I2C commands directly to the PMOD_AD2 peripheral

Use the documentation on the PMOD_AD2 to answer lab questions

In [5]:
```
from pynq.overlays.base import BaseOverlay
from pynq.lib import MicroblazeLibrary
base = BaseOverlay("base.bit")
```

In [6]:
```
liba = MicroblazeLibrary(base.PMODA, ['i2c'])
```

In [7]:
```
dir(liba) # list the available commands for the liba object
```

Out[7]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_build_constants',
 '_build_functions',
 '_mb',
 '_populate_typedefs',
 '_rpc_stream',
 'active_functions',
 'i2c_close',
 'i2c_get_num_devices',
 'i2c_open',
 'i2c_open_device',
 'i2c_read',
 'i2c_write',
 'release',
 'reset',
 'visitor']
```

In the cell below, open a new i2c device. Check the resources for the i2c_open parameters

In [8]:
```
device = liba.i2c_open(3, 0x28) # TODO open a device
```

In [9]:
```
dir(device) # list the commands for the device class
```

Out[9]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_call_func',
 '_file',
 '_val',
 'close',
 'read',
 'write']
```

Below we write a command to the I2C channel and then read from the I2C channel. Change the buf[0] value to select different channels. See the AD spec sheet Configuration Register. https://www.analog.com/media/en/technical-documentation/data-sheets/AD7991_7995_7999.pdf

Changing the number of channels to read from will require a 2 byte read for each channel!

In [ ]:
```
buf = bytearray(2)
buf[0] = int('00000000', 2)
device.write(0x28, buf, 1)
device.read(0x28, buf, 2)
print(format(int(((buf[0] << 8) | buf[1])), '#018b'))
```

Compare the binary output given by ((buf[0]<<8) | buf[1]) to the AD7991 spec sheet. You can select the data only using the following command

In [ ]:
```
result_12bit = (((buf[0] & 0x0F) << 8) | buf[1])
```

Using MicroBlaze

In [11]:
```
base = BaseOverlay("base.bit")
```

In [12]:
```
%%microblaze base.PMODA

#include "i2c.h"

int read_adc(){
    i2c i2c_device = i2c_open(3, 2);
    unsigned char buf[2];
    buf[0] = 0;
    i2c_write(i2c_device, 0x28, buf, 1);
    i2c_read(i2c_device, 0x28, buf, 2);
    return ((buf[0] & 0x0F) << 8) | buf[1];
}
```

In [13]:
```
read_adc()
```

Out[13]:
```
0
```