WES 237A: Introduction to Embedded System Design (Winter 2024)
Lab 5: Inter-Integrated Circuit (I2C) Communication
Due: 3/3/2024 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 5 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. You could either scan your written copy, or simply type your answer in this Google Doc. **However, please make sure your responses are readable.**
- 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.

- Connect the PMOD_AD2 peripheral to PMODA.
- Download the *iic_example.ipynb*
- Go through the notebook and answer the following questions. The following resources may be helpful
  - https://pynq.readthedocs.io/en/v2.6.1/pynq_libraries/pynqmb_reference.html
  - https://www.analog.com/media/en/technical-documentation/data-sheets/AD7991_7995_7999.pdf
  - https://pynq.readthedocs.io/en/v2.1/pynq_package/pynq.lib/pynq.lib.pmod.html#pynq-lib-pmod

- **What command opens a new i2c device in the MicroblazeLibrary? What are the two parameters to this command?**

```
<i2c_device>.i2c_open(sda, scl)
class MicroblazeFuncion_i2c_open(MicroblazeFunction)
 |  MicroblazeFuncion_i2c_open(stream, index, function, return_type)
 |
 |  Open an I2C device through an IO switch
 |
 |  Parameters
 |  ----------
 |  sda : int
 |      The data pin number on the IO switch
 |  scl : int
 |      The clock pin number on the IO switch
```

- **What does 0x28 refer to in the following line?**
  - *device.write(0x28, buf, 1)*

0x28 is the address offset. This refers to the device on the i2c channel as i2c supports multiple devices per channel.

- **Why do we *write* and then *read* when using the Microblaze Library compared to just *reading* in the PMOD Library?**

When using the pmod library, we read and write straight to the peripheral. Because we are using an i2c device we have to request a response from the device. This is why we write first, then read.

- **What does this code snippet mean?** *return ((buf[0] & 0x0F) << 8) | buf[1]*

Return buf[0] bitmasked so that we only see the 4 least significant bits, then shift those bits left by 8 bytes and bitwise OR buf[1]:
E.g
buf[0] = b'00000000 10001100
buf[1] = b'00000000 11000011

return b'00001100 11000011 = 0x0CC3

- **What is the difference between writing to the device when using the Microblaze Library and directly on the Microblaze?**

When using the microblaze library we are calling a preset library to write to a device on the i2c . When writing directly to the microblaze we are writing to the peripheral itself.

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]:
```
[3991]
```

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

Out[4]:
```
[1.9448]
```

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 [10]:
```
device = liba.i2c_open(3,2)# TODO open a device
```

In [11]:
```
dir(device) # list the commands for the device class
```

Out[11]:
```
['__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 [12]:
```
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'))
```

```
0b0000000100100000
```

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 [13]:
```
result_12bit = (((buf[0] & 0x0F) << 8) | buf[1])
```

Using MicroBlaze

In [14]:
```
base = BaseOverlay("base.bit")
```

In [18]:
```
%%microblaze base.PMODA

#include "i2c.h"
#I did not have the i2c.h file but I know what this is supposed to do :)
int read_adc(){
    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];
}
```

Out[18]:
```
Compile FAILED
cell_magic: In function 'int read_adc()':
cell_magic:9:15: error: 'i2c_device' was not declared in this scope; d
id you mean 'device'?
```

In [17]:
```
read_adc()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-da14b995cef8> in <module>
----> 1 read_adc()

NameError: name 'read_adc' is not defined
```

In [ ]: