Lab #5 Report and Reflections

ABDULLAH AJLAN **A69028719**

WES 237A: Introduction to Embedded System Design (Winter 2024)
Lab 5: Inter-Integrated Circuit (I2C) Communication

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

The command to open a new I2C device in the MicroblazeLibrary is XIic_Open. The two parameters are InstancePtr and ConfigPtr.

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

In the line device.write(0x28, buf, 1), 0x28 refers to the address of the I2C device on the bus. It's the address where the data will be written to.

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

When using the Microblaze Library on PYNQ-Z2, we write and then read to communicate with an I2C device because the Microblaze Library operates on a lower level, requiring separate write and read operations to perform a complete transaction. In contrast, the PMOD Library abstracts these operations, allowing for simpler read operations.

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

This code snippet returns a 12-bit value by combining the lower 8 bits from buf[0] (after masking with 0x0F) shifted left by 8 positions, and the value in buf[1].

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

When using the Microblaze Library on PYNQ-Z2, writing to the device involves interacting with peripheral devices connected to the Microblaze processor via specialized interfaces like I2C or SPI. This is done through library functions such as XIic_Write. On the other hand, writing directly on the Microblaze involves manipulating memory-mapped registers or accessing peripheral devices directly through low-level programming without using the Microblaze Library functions.

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 [83]:
```
from pynq.overlays.base import BaseOverlay
from pynq.lib import Pmod_ADC
base = BaseOverlay("base.bit")
```

In [84]:
```
adc = Pmod_ADC(base.PMODB)
```

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 [85]:
```
adc.read_raw(ch1=1, ch2=0, ch3=0)
```

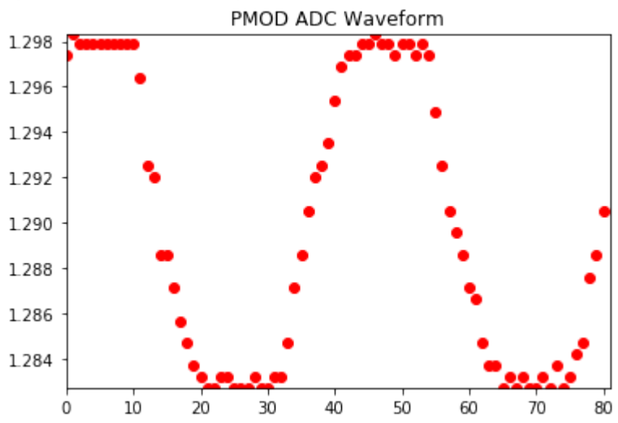
Out[85]:
```
[1895]
```

In [87]:
```
adc.read(ch1=1, ch2=0, ch3=0)
```

Out[87]:
```
[0.9448]
```

In [88]:
```
from time import sleep
from pynq.overlays.base import BaseOverlay
from pynq.lib import Pmod_ADC

base = BaseOverlay("base.bit")


if_id = input("Type in the interface ID used (PMODA or PMODB): ")
if if_id.upper()=='PMODA':
    adc = Pmod_ADC(base.PMODA)
else:
    adc = Pmod_ADC(base.PMODB)

freq = int(input("Type in the frequency/Hz of the waveform: "))
period = 1/freq
log_interval_us = 0

# Assume Channel 0 is connected to the waveform generator
adc.start_log(1,0,0,log_interval_us)
sleep(3*period)
log = adc.get_log()

# Draw the figure
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(range(len(log)), log, 'ro')
plt.title('PMOD ADC Waveform')
plt.axis([0, len(log), min(log), max(log)])
plt.show()

adc.reset()
del adc,base
```

```
Type in the interface ID used (PMODA or PMODB): PMODB
Type in the frequency/Hz of the waveform: 100
```



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

In [2]:
```
liba = MicroblazeLibrary(base.PMODB, ['i2c'])
```

In [3]:
```
dir(liba) # list the available commands for the liba object
```

Out[3]:
```
['__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 [ ]:
```
# Open a new I2C device
device = liba.i2c_open(1,2)
```

In [17]:
```
dir(device) # list the commands for the device class
```

Out[17]:
```
['__call__',
 '__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__',
 '_call_function',
 '_handle_stream',
 'call',
 'call_async',
 'function',
 'index',
 'return_type',
 'stream']
```

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 [ ]:
```

```

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 [19]:
```
result_12bit = (((buf[0] & 0x0F) << 8) | buf[1])
```

Using MicroBlaze

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

In [16]:
```
%%microblaze base.PMODB

#include "i2c.h"

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[16]:
```
Compile FAILED
cell_magic: In function 'int read_adc()':
cell_magic:9:15: error: 'i2c_device' was not declared in this scope; did you mean 'device'?
```

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

In [ ]:
```

```