WES 237A: Introduction to Embedded System Design (Winter 2024)
Lab 5: Inter-Integrated Circuit (I2C) Communication
Due: 3/3/2024 11:59pm

Ricardo Lizarraga, rlizarraga0@gmail.com 619.252.4157	PID: A69028483 https://github.com/RiLizarraga/WES237A_Lab5

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?**

liba = MicroblazeLibrary(base.PMODA, ['i2c'])
The 2 parameters:
1.- base.PMODA {PMODE to use}
2.- ['i2c'] {protocol}

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

0x28 is the HEX value of the address we are writing to, to configure the device to behave as AD7991-0

HEX	28
DEC	40
OCT	50
BIN	0010 1000

Table 8. I²C Address Selection

Part Number	I²C Address
AD7991-0	010 1000

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

We write I2C address (AD7991-0), then read that address from device

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

  This piece of code is to comply protocol, this will zero the unused bits (because only 12 bits are needed out of the 16 of the 2 bytes), then flip the order to comply the i2c endianness protocol.

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

When using the Microblaze library, this can be used straight from Python coding, but if we do it directly in Microblaze, we can do it on C code

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

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

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

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

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

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

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

In [ ]:
```
device = # TODO open a device
```

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

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

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

#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];
}
```

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

In [ ]:
```

```