

WES 237A: Introduction to Embedded System Design (Winter 2026)

Lab 5: Inter-Integrated Circuit (I2C) Communication

Due: 3/1/2026 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_open(int sda, int scl); the parameters refer to the corresponding pins on the PMOD that connect to the I2C device. In this case, pin 3 connects to sda, and 2 connects to scl.

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

'0x28' is the device ID, which is the address of this specific I2C device

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

We write a command to the slave in order for it to send back a signal for the Microblaze to read.

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

It returns the data that is coming back from the slave device.

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

Using the Library allows us to write all the code in Python, whereas writing directly on the Microblaze requires to run some C code.

Using PYNQ library for PMOD_ADC

This just uses the built in Pmod_ADC library to read the value on the PMOD_ADC2 peripheral.

```
In [1]: from pynq.overlay.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]: [4095]
```

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

```
Out[4]: [1.9995]
```

Using MicroblazeLibrary

Here we're going down a level and using the microblaze library to write I2C commands directly to the PMOD_ADC2 peripheral

Use the documentation on the PMOD_ADC to answer lab questions

```
In [5]: from pynq.overlay.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(3,2) # 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 [11]: import time  
while True:  
    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'))  
    time.sleep(1)
```



```
KeyboardInterrupt                                     Traceback (most recent call last)
Input In [11], in <cell line: 2>()
    6 device.read(0x28, buf, 2)
    7 print(format(int(((buf[0] & 0x0F) << 8) | buf[1]), '#018b'))
----> 8 time.sleep(0.1)
```

KeyboardInterrupt:

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

```
In [12]: result_12bit = ((buf[0] & 0x0F) << 8) | buf[1])
```

Using MicroBlaze

```
In [13]: base = BaseOverlay("base.bit")
```

```
In [14]: %%microblaze base.PMODA

#include "i2c.h"

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

```
In [15]: while True:
    print(read_adc())
    time.sleep(1)
```

304
311
308
311
304
311
283
251
284
282
251
286
287
250
254
256
254
288
297
307
308
309
309
308
311
307
307
309
309
309
308
309
308
311
308
309
309
310
309
308
310
305
265
253
250
251
252
288
292
264
246
280
275
240
241
277
271
273
271

242
277
240
275
240
277
241
241
242
277
240
242
242
242
242
242
241
242
242
242
242
242
243
242
243
248
293
309
308
310
308
310
309
309
309
308
310
308
310
308
310
308

```
KeyboardInterrupt
Input In [15], in <cell line: 1>()
  1 while True:
  2     print(read_adc())
----> 3     time.sleep(1)
```

Traceback (most recent call last):

KeyboardInterrupt:

In [1]: