03: C Interfaces

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Announcements

- Slack – See Website
- Office Hours – See Website / Syllabus
- P1: Due Friday
- P2: Out now! * -> Matteo
  - (Still working on AG a little)

Failed Login & Disk Space

- If you can't log into the Linux machines:
  - It might be you are out of disk drive space

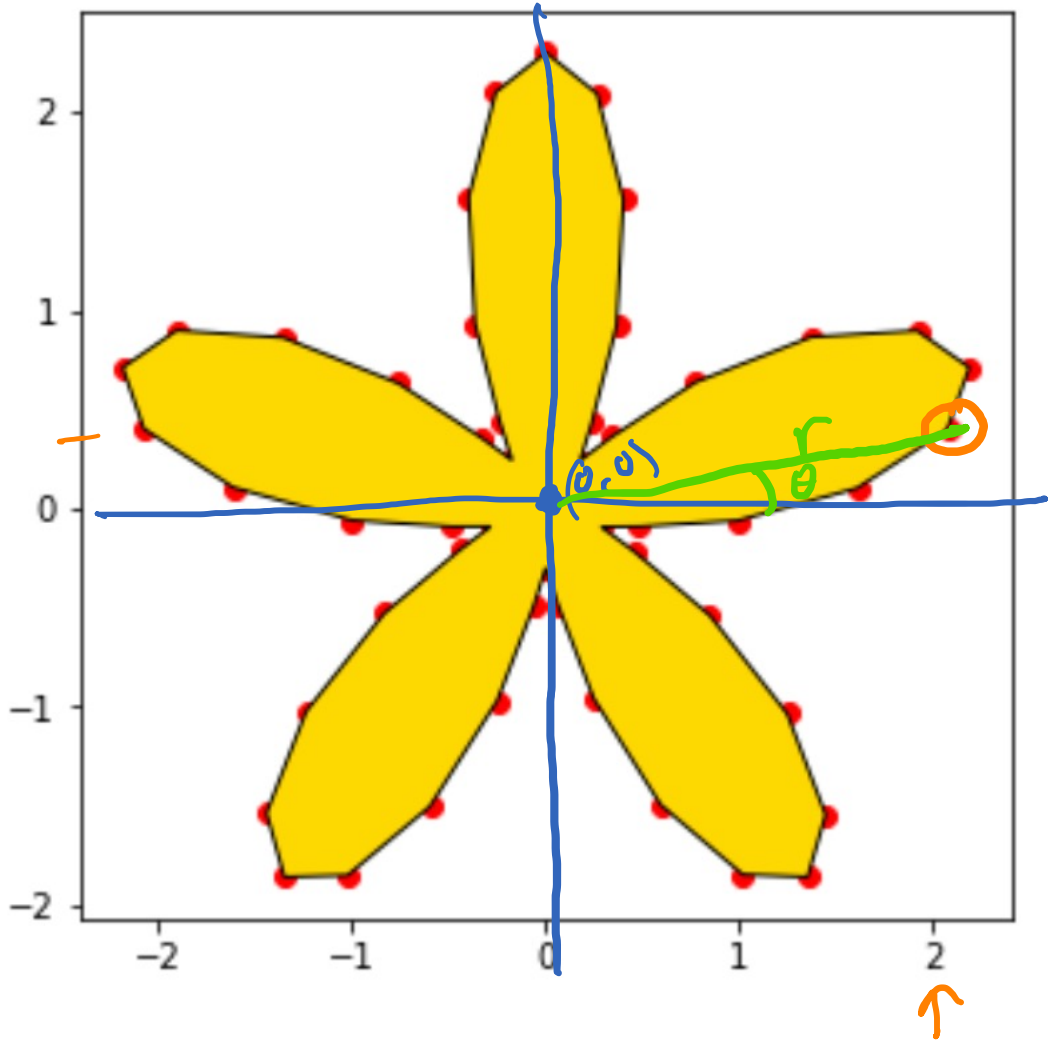
- ssh into silo/kj and do the following:

```
$ quota
```

If overfull, remove some things, then try to log in again.

setup /nobackup

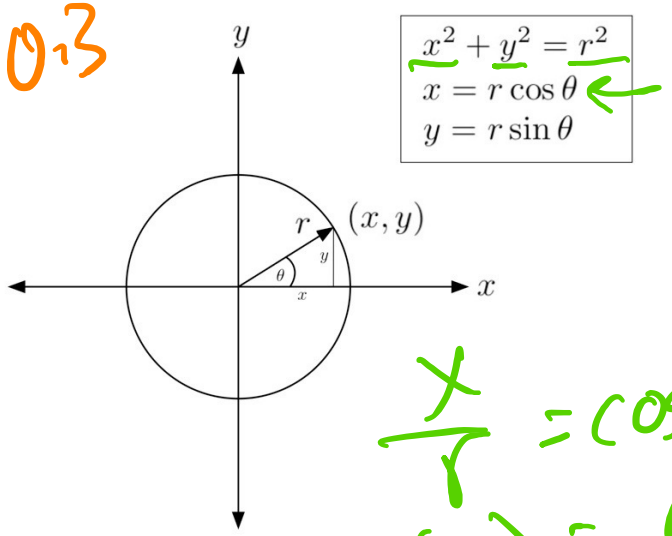
Project 1: Flowers



$(x,y) = 2, 0.3$

$r = 2.2$

$\theta = \sim 12°$

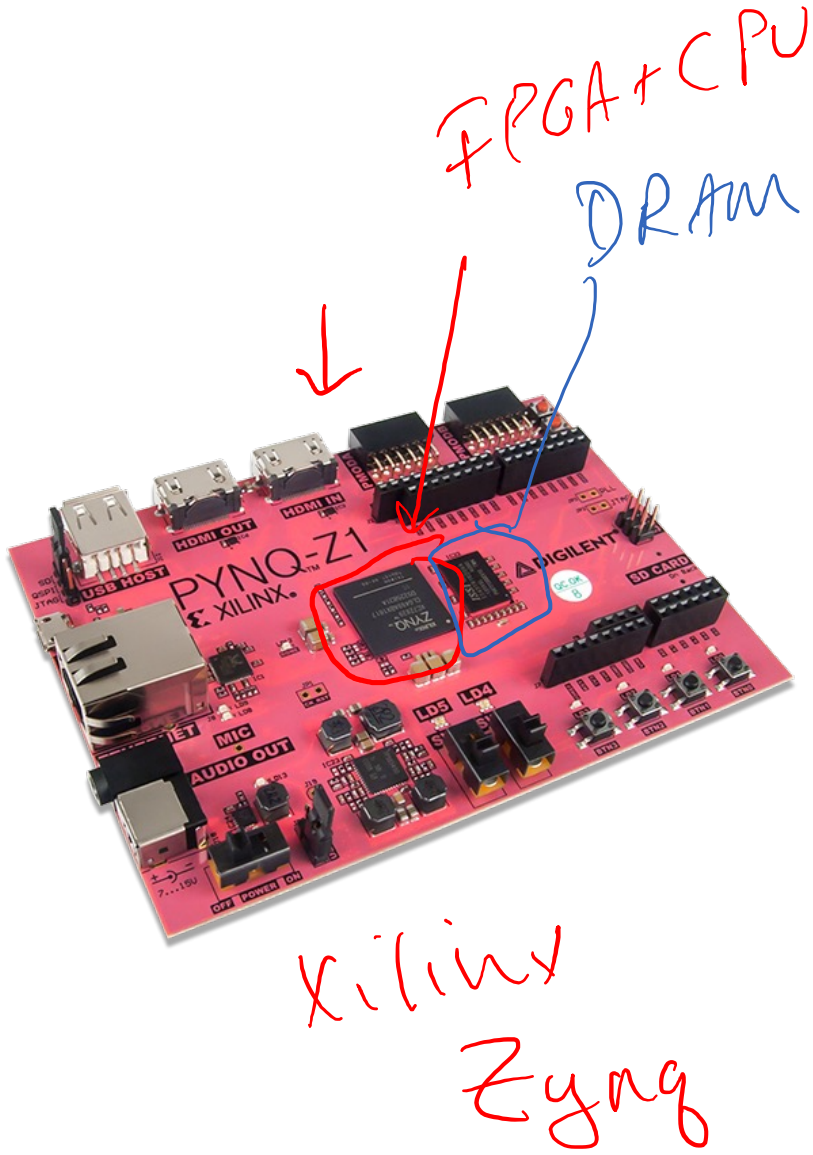


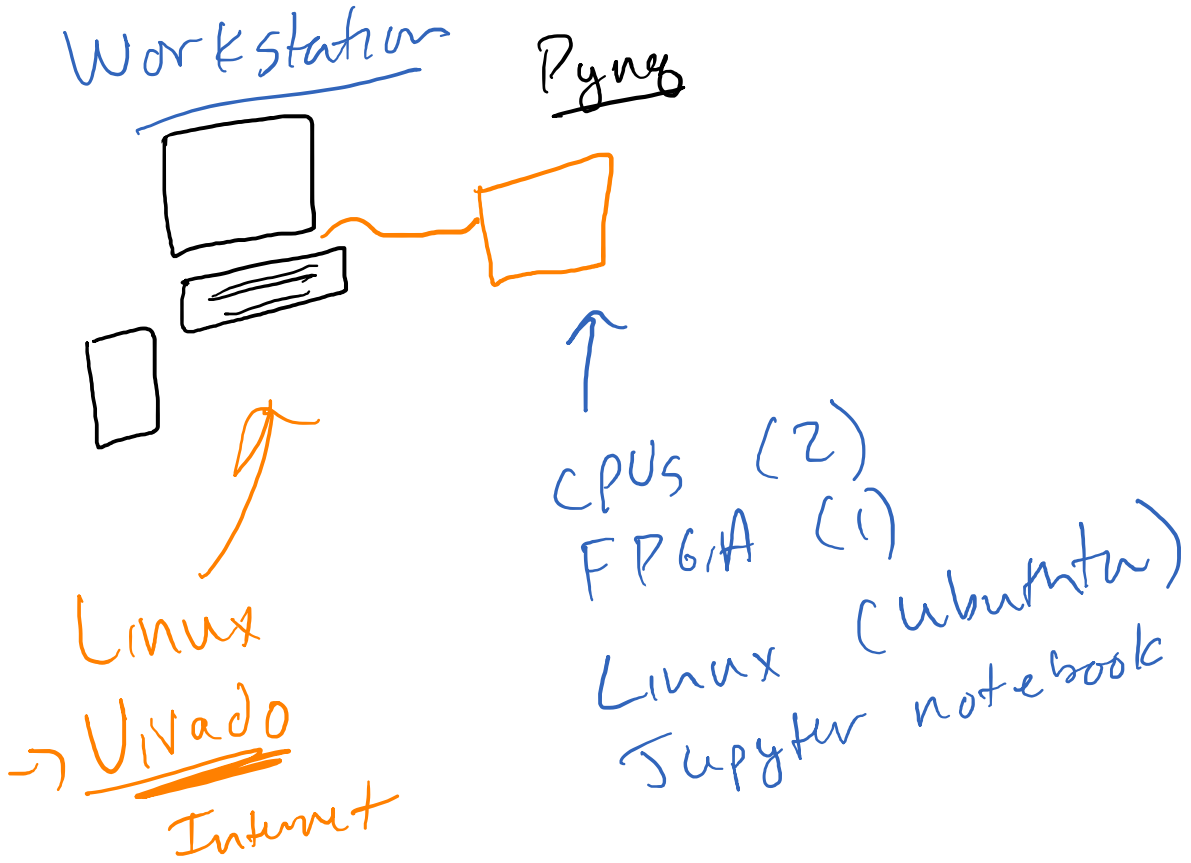
$\frac{x}{r} = \cos\theta$

$\cos^{-1}\left(\frac{x}{r}\right) = \theta$

The Pynq

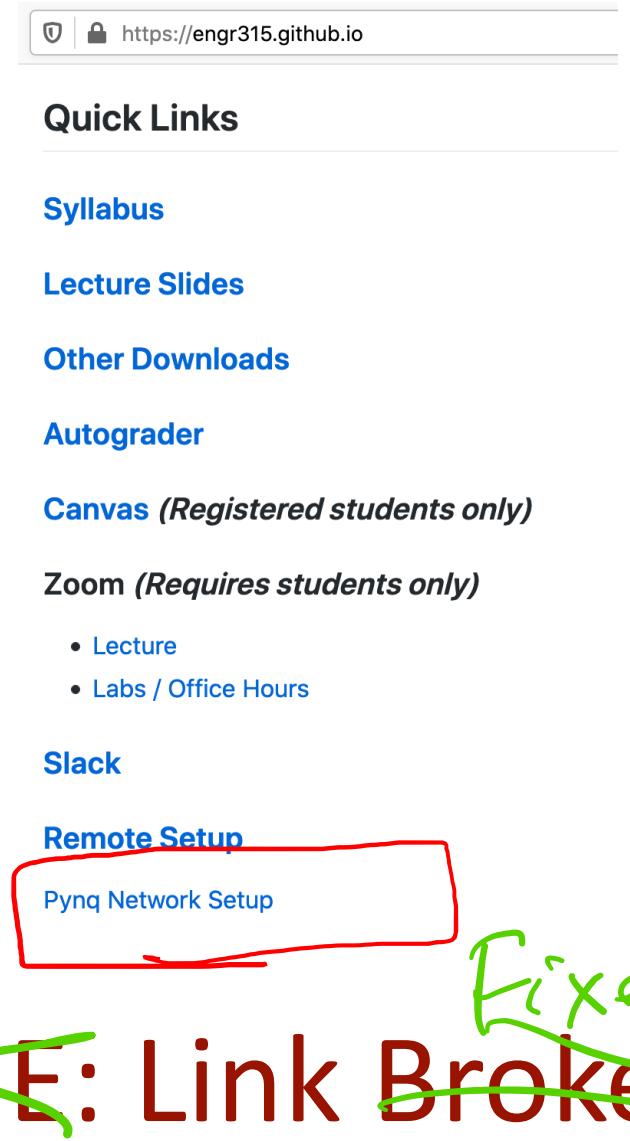
- Used for P2 onward
- System-on-Chip
  - SoC - “S-O-C” or “Sock”
- Contains both FPGA and CPU
- Runs Linux + Python




Workstation
Pynq
CPUs (2)
FPGA (1)
Linux (ubuntu)
Jupyter notebook
Linux
Vivado
Internet

Setup Notes

- 4111 is best.
  - Everything already set up
- Can work from home
  - need Pynq networked
  - “Some” effort support
- Pure-Remote students
  - Email me.

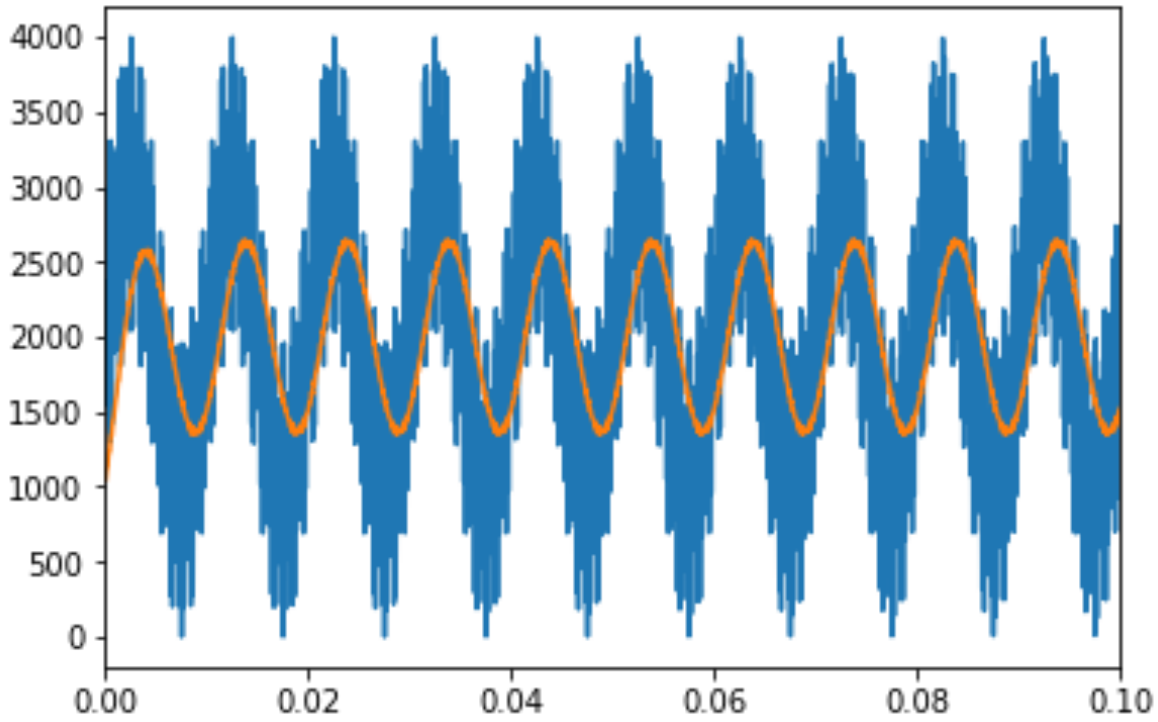


Let’s talk P2 (and P3)

- What is EMA?
- Pynq Setup
  - The password is always ‘iuxilinx’
- e315helper.py
- Vivado Setup

Wait for until P3...

Signal Filtering



EMA Example

$y[-1] = 0$

$y[n] = \alpha\, x[n] + (1-\alpha)\, y[n-1]$

$\alpha = 0.5$

$x = [\underbrace{0}_{x_0}, \underbrace{10}_{x_1}, \underbrace{0}_{x_2}, 0, 10]$

$y = [0, 5, 2.5, 1.25, 5.625]$

$y[0] = 0.5\, x[0] + 0.5\, y[-1]$

$= 0.5 \cdot 0 + 0.5 \cdot 0$

$y[0] = 0$

$y[1] = 0.5 \cdot 10 + 0.5\, y[0]$

$= 0.5 \cdot 10 + 0.5 \cdot 0$

$= 5$

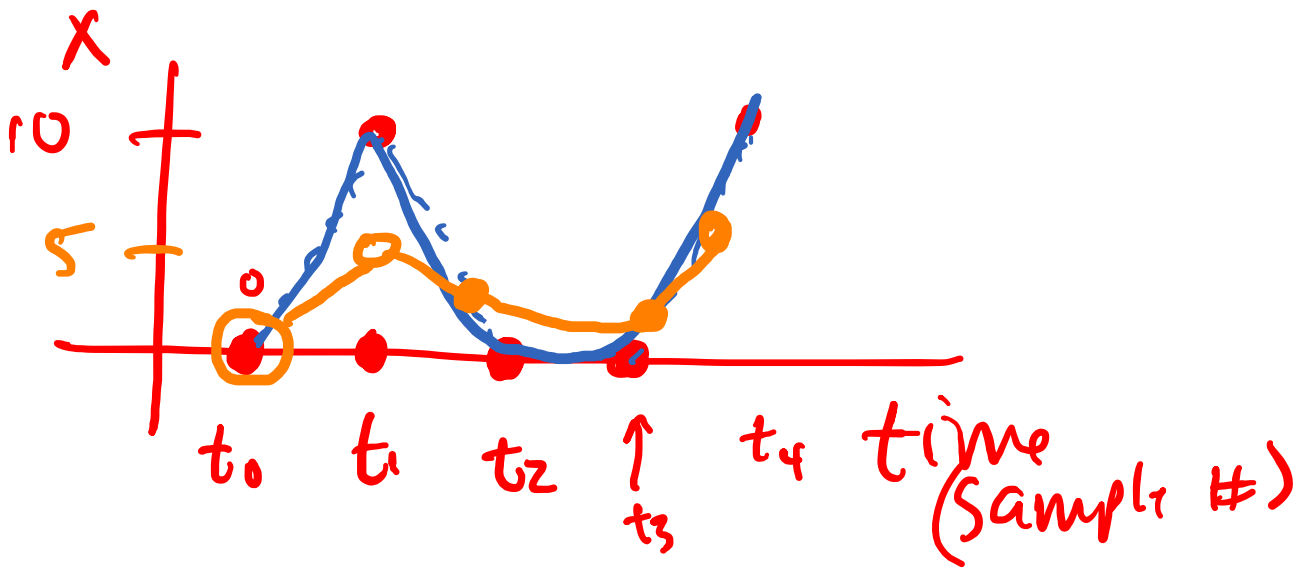
$y[2] = 0.5 \cdot 0 + 0.5 \cdot 5$

$= 2.5$

$y[3] = 0.5 \cdot 0 + 0.5 \cdot 2.5$

$= 1.25$

$y[4] = 0.5 \cdot 10 + 0.5 \cdot 1.25 = 5.625$



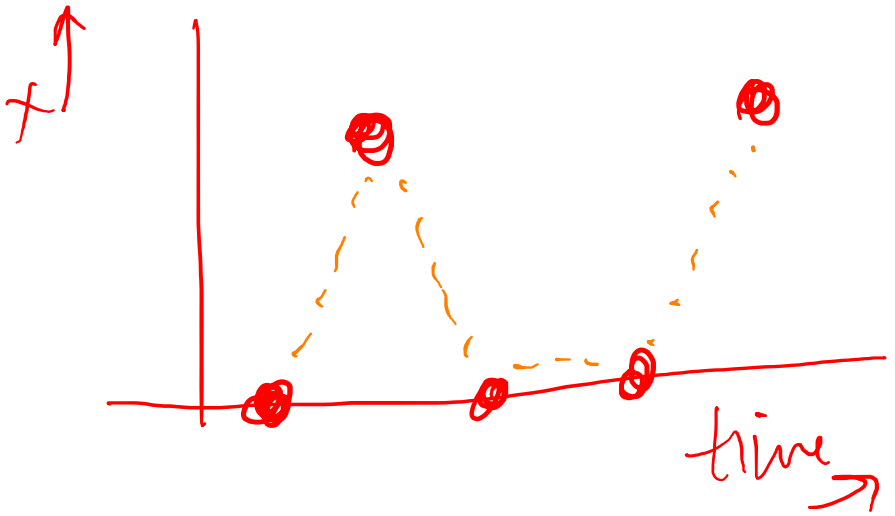
EMA Example

$y[-1] = 0$

$y[n] = \alpha x[n] + (1-\alpha) \cdot y[n-1]$

$\alpha = 0.5$

$x = [0, 10, 0, 0, 10]$



P2 Demo Time

Numpy is written in C/Assembly. It's faster.

```
import numpy
def find_ignore_case6(needle, haystack):
    return np.where(haystack==needle)
```

assembly ?

```
Find: 0.270210 seconds
Find2: 0.061821 seconds
Find3: 0.054265 seconds
Find4: 0.051191 seconds
Find5: 0.000265 seconds
Find6: 0.000052 seconds
```

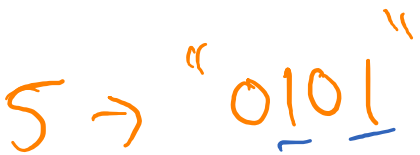
Popcount

- Count the number of binary 1's in a number
- 01000100101000010010000100000000
- 7 total 1's

Popcount

```
def popcount(num):
    return bin(num).count('1')
```

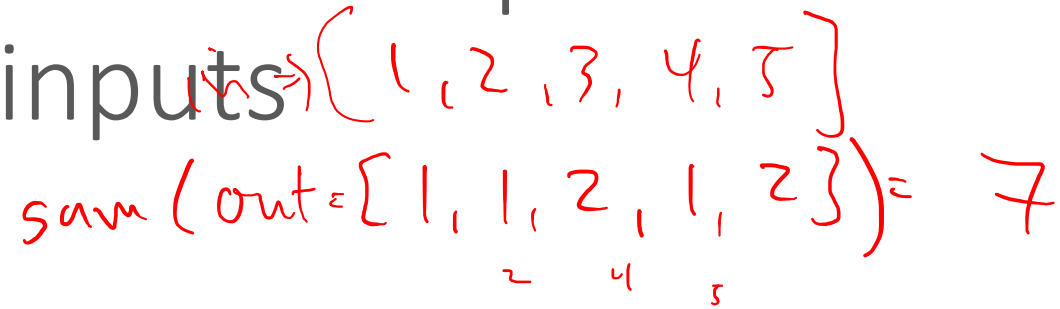
```
value: 0 bin: 0b0 popcount: 0
value: 1 bin: 0b1 popcount: 1
value: 2 bin: 0b10 popcount: 1
value: 3 bin: 0b11 popcount: 2
value: 4 bin: 0b100 popcount: 1
value: 5 bin: 0b101 popcount: 2
value: 6 bin: 0b110 popcount: 2
value: 7 bin: 0b111 popcount: 3
value: 8 bin: 0b1000 popcount: 1
value: 9 bin: 0b1001 popcount: 2
```



popcount (slower, but no external calls)

```
def popcount2(num):
    w = 0
    while (num):
        w += 1
        num &= num - 1
    return w
```

`Popcount_all` is a helper function to run larger blocks of inputs




```
def popcount_all(buf):
    return sum(map(popcount,buf))

def popcount2_all(buf):
    return sum(map(popcount2,buf))
```

Big Bitcount

```
np.random.seed(1)
buf = np.random.randint(0,1E9,int(1E6))

start_time = time.time()
sum_1s = popcount_all(buf)
end_time = time.time()
print("popcount: %f seconds (w/libs)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = popcount2_all(buf)
end_time = time.time()
print("popcount2: %f seconds (w/o libs)"
      % (end_time - start_time))
```

```
popcount: 0.307169 seconds (w/libs)
popcount2: 1.853192 seconds (w/o libs)
```

How did the library go so much faster?

- Python called C.
- The computations happened in C. It's faster.
- Can we do that?

Let's find out.

Popcount in Python vs. C

Python

```
def popcount2(num):

    w = 0
    while (num):
        w += 1
        num &= num - 1

    return w
```

C

```
int popcount(uint64_t num)
{
    int w=0;
    while (num) {
        w +=1;
        num &= (num -1);
    }
    return w;
}
```

```
np.random.seed(1)
buf = np.random.randint(0,1E9,int(1E6))
buf = buf.tolist()

start_time = time.time()
sum_1s = popcount_all(buf)
end_time = time.time()
print("popcount: %f seconds (w/calls)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = popcount2_all(buf)
end_time = time.time()
print("popcount2: %f seconds (w/o calls)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = sum(map(cPopcount.cPopcount,buf))
end_time = time.time()
print("c_popcount: %f seconds (64-bits in C)"
      % (end_time - start_time))
```

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
```

```c
static PyObject *
cPopcount_all(PyObject *self, PyObject *args)
{
    PyObject *obj;
    int64_t res = 0;

    //parse the list argument
    if (!PyArg_ParseTuple(args, "O", &obj)) {
        return NULL;
    }

    //hope it's iteratable
    PyObject *iter = PyObject_GetIter(obj);
    if (!iter) {
        return NULL;// error not iterator
    }

    //loop over all elements in list
    while (1) {
        PyObject *next = PyIter_Next(iter);

        if (!next) {
            // nothing left in the iterator
            break;
        }
```

```c
        // conver to int64_t
        int64_t num = 0;
        if (PyLong_Check(next)) {
            num = PyLong_AsLong(next);
        } else {
            printf ("unsupported type\n");
            return NULL;
        }

        //now do popcount!
        res += popcount(num);// do something with foo

        /* release reference when done */
        Py_DECREF(next);
    }
    Py_DECREF(iter);

    return PyLong_FromLong(res);
}
```

Two ways to handle lists:

- Iterators (previous slide)
- https://stackoverflow.com/questions/22458298/extending-python-with-c-pass-a-list-to-pyarg-parsetuple
- Array indices (not shown)
- https://stackoverflow.com/questions/39063112/passing-a-python-list-to-c-function-using-the-python-c-api

```
start_time = time.time()
sum_1s = sum(map(cPopcount.cPopcount,buf))
end_time = time.time()
print("c_popcount: %f seconds (64-bits in C)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = cPopcount.cPopcount_all(buf)
end_time = time.time()
print("c_popcount: %f seconds (List in C)"
      % (end_time - start_time))
```

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
c_popcount: 0.007329 seconds (List in C)
```

Same algorithm. C vs. Python.

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
c_popcount: 0.007329 seconds (List in C)
```

When performance matters, use C.
When it doesn't, use Python.

Did you know you can call assembly from C?

```
int popcount_asm(uint64_t num)
{
    uint64_t result;
    asm (
        "POPCNT %1, %0   \n"
        : "=r" (result)
        : "mr" (num)
        : "cc"
    );
    return result;
}
```

Python Demo failed

```
gcc -Wall -march=nehalem -o test.o test.c popcount.c
```

Let’s time assembly Popcount…

03: C Interfaces

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University