Assignment 3

In lab, we experimented with C++ code for initializing the PMU counters and retrieving the cyclecount. In this assignment, you'll be setting up your PMU counter to use in Python.

Part A3.0: New kernel_module code

- Download the new kernel_module.zip code, go to canvas -> files -> assignments.
- Make and insert this new module following the 'make' and 'insmod' instructions from lab and in the README.
- Check that it is inserted on both CPUs by checking the dmesg | tail output
-
- I just followed the same instructions as the lab. It was very straightforward and I did not have any issues with building.

Part A3.1: Access PMU from python

- Create a shared library object with two fuctions by wrapping the cycletime.h into a new shared opticed library (see Lab2)
  - One function to initialize the PMU counters
  - One function to get the cycle count
- Compile the shared library (see Lab2 if you've forgotten how to do this)
- Access the shared library functions using the ctypes module, **but don't wrap the function calls in a python function.**
-
- I redefined the cycletime.h to be just a header file, and then moved the definitions of the functions into a cycletime.c file.
- To compile, I copied the Makefile over from the Lab and just ran make.
  - At first it didn't work. I looked around online and edited the Makefile to make it work for the code and my own directory structure.
- I then created a jupyter notebook called test_pmu. In there, I ran the two functions and made some simple code to get the cycle difference after sleeping for some arbitrary time.

Part A3.2: Comparing and Gathering Data

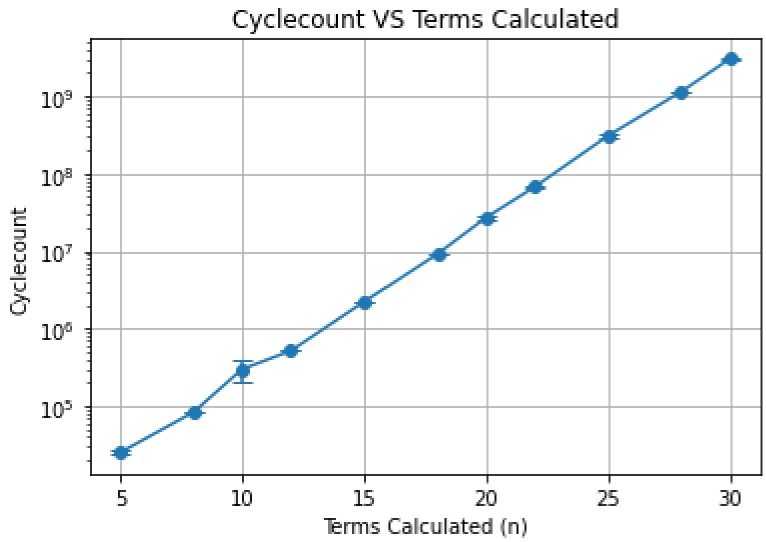
In this section, we are going to use psutil to monitor CPU usage in percent, and the time module and PMU counting to evaluate the recursive fibonacci sequence timing operations.

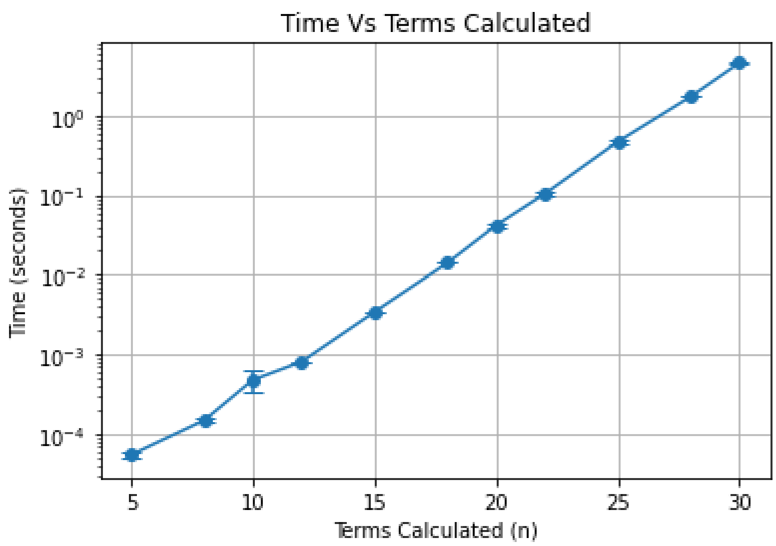
- Isolate CPU 1 by editing the bootargs (see lab work part 1)
- Insert the CPUcntr kernel object onto both cpus using the instructions from lab
- Write code to do the following using the recur_fibo function from lab
  - Initialize the cyclecounter
  - Get the 'before' time using the python time module
  - Get the 'before' cycle count
  - Run the recur_fibo function on a CPU 1
  - Get the 'after' cycle count
  - Get the 'after' time count using the python time module
  - Get the cycle count and the amount of time used
- Vary the number of terms from 1 to 30 **as you see fit** to compare the different execution times
  - Take multiple trials for each variation (i.e. get three cyclecounts for n=5, then get three cyclecounts for n=10, etc) and average the different tials.
  - The error for each 'n' will be the standard deviation from the mean which is the standard deviation of all the trials divided by the square root of the number of trials.
- Plot the average results for varying 'n' along with error bars of your measurments.

- In order to compare the timing module and PMU counting, we need them to be in the same units.
  - To get the CPU frequency, run cat /proc/cpuinfo in a new terminal or run lscpu
  - Use this frequency to convert the PMU output from clock counts to timing
  - Compare the timing of the PMU counter to the timing module
-
- Before I started writing any code, I restructured my folder so everything stays organized.

```
xilinx@pynq:~/jupyter_notebooks/Assignment3$ ls -R
.:
kernel_module  pmu_test  python

./kernel_module:
CPUcntr.c  CPUcntr.ko  CPUcntr.mod  CPUcntr.mod.c  CPUcntr.mod.o  CPUcntr.o  Makefile
modules.order  Module.symvers  README

./pmu_test:
cycletime.c  cycletime.h  libcycletime.so  Makefile

./python:
test_pmu.ipynb
```

- Using the recur_fibo function, I added lines corresponding to each of the tasks in the bulleted list.
- I wrapped this in a function so I can recursively test the cycle and time counts with a different n every time.
- I then made a list of the n values I wanted to test
  - I chose n_values = [5, 8, 10, 12, 15, 18, 20, 22, 25, 28, 30] for the following reasons:
    - Nothing less than 5 because the difference in computation is minimal and a little bit too fast for us to see anything
    - Equally spaced ish intervals so we can see the exponentially longer computation time.
- I created some arrays to store the average time/cycle for each iteration of the test, and then ran a for loop that collected all of the data points and calculated the mean and error of each n value in both time and cycles.
  - I was originally considering a dictionary with a key value pair, where the key is the n value and the value is whatever I was trying to calculate. However that was unnecessary and I just used normal arrays for everything.
  - I changed the arrays into np arrays so calculations are easier and I can use the built in functions
- An issue that I hit while gathering the samples was that the cycle time would be 0. I checked though and it turned out it was because I wasn't isolating the CPU properly.
- For the graph, I used matplotlib, which has an error bar graph plotting function. I put in the corresponding values and plotted the results.

- 

- 

YouTube link: https://youtu.be/torJvckuAA8

Github: https://github.com/cxu4426/WES237A-hw/tree/main/assignment_3

Deliverables

Each student must submit the following individually

- A PDF report detailing your work flow and relative Jupyter notebook cells relating to your progress.
  – Your report should detail your work flow throughout the assignment. Be sure to discuss any difficulties or troubles you encountered and your troubleshooting procedure. Also, detail your thought process which led to the design and implementation of the code. For example, describe your top-down design methodology (i.e. how did you split the large task into smaller, more incremental jobs? How were you able to test each of these smaller parts?)
  – Please also include a plot with errorbars where the x-axis is the number of terms calculated (n) and the y-axis is the cyclecount
  – Please also include a plot with errorbars where the x-axis is the number of terms calculated (n) and the y-axis is the time taken to complete
- Your complete Jupyter notebook, downloaded as a PDF attached at the very end of your report
  – You can do this by selecting 'File -> Print Preview' then printing to PDF from the browser.
  – Use a PDF stitching tool like pdfjoiner to join your Report and Jupyter Notebook into a single PDF file

Each team should submit the following one per team

- All relevant code (.ipynb, .py, .cpp, .c, etc files) pushed to your team's git repo.

```c
#include <stdint.h>

/* Initialize PMU counters */
void pmu_init(int32_t do_reset, int32_t enable_divider)
{
    int32_t value = 1;

    if (do_reset)
        value |= 6;     // reset all counters
    if (enable_divider)
        value |= 8;

    value |= 16;

    asm volatile ("MCR p15, 0, %0, c9, c12, 0\n\t" :: "r"(value));
    asm volatile ("MCR p15, 0, %0, c9, c12, 1\n\t" :: "r"(0x8000000f));
    asm volatile ("MCR p15, 0, %0, c9, c12, 3\n\t" :: "r"(0x8000000f));
}

/* Read cycle counter */
unsigned int pmu_get_cyclecount(void)
{
    unsigned int value;
    asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));
    return value;
}

```

In [1]:
```
from ctypes import CDLL, c_uint, c_int
import time
```

In [2]:
```
lib = CDLL("./libcycletime.so")

lib.pmu_init.argtypes = [c_int, c_int]
lib.pmu_get_cyclecount.restype = c_uint
```

In [19]:
```
lib.pmu_init(1, 0)

start = lib.pmu_get_cyclecount()
time.sleep(0.5)
end = lib.pmu_get_cyclecount()

print("Cycle difference:", end - start)
```

```
Cycle difference: 9252565
```

fib function

In [1]:
```
def recur_fibo(n):
   if n <= 1:
       return n
   else:
       return(recur_fibo(n-1) + recur_fibo(n-2))
```

In [2]:
```
from ctypes import CDLL, c_uint, c_int
import time
import matplotlib.pyplot as plt
import numpy as np
```

implementation

In [3]:
```
lib = CDLL("../pmu_test/libcycletime.so")

lib.pmu_init.argtypes = [c_int, c_int]
lib.pmu_get_cyclecount.restype = c_uint
```

In [4]:
```
lib.pmu_init(1, 0)
```

Out[4]: 1

In [5]:
```
def cycle_diff(c1, c0):
    MAX32 = 2**32
    if c1 >= c0:
        return c1 - c0
    else:
        return (MAX32 - c0) + c1
```

In [6]:
```
def fibo_duration(n):
    t0 = time.time()
    c0 = lib.pmu_get_cyclecount()
    recur_fibo(n)
    c1 = lib.pmu_get_cyclecount()
    t1 = time.time()

    time_elapsed = t1 - t0
    cycle_elapsed = cycle_diff(c1, c0)
    return time_elapsed, cycle_elapsed
```

gather cycles and times for different n for fibo function

In [7]:
```
n_values = [5, 8, 10, 12, 15, 18, 20, 22, 25, 28, 30]
num_trials = 5
```

In [13]:
```
time_avg = []
time_err = []
cycle_avg = []
cycle_err = []
```

```python
for n in n_values:
    print(f"\n--- n = {n} ---")
    times = []
    cycles = []

    for trial in range(num_trials):
        t, c = fibo_duration(n)
        print(f"  trial {trial}: time = {t:.6f} s, cycles = {c}")
        times.append(t)
        cycles.append(c)

    times = np.array(times)
    cycles = np.array(cycles)

    # mean
    mean_time = times.mean()
    mean_cycle = cycles.mean()
    print(f"  mean_time = {mean_time:.6f} s")
    print(f"  mean_cycle = {mean_cycle}")

    time_avg.append(mean_time)
    cycle_avg.append(mean_cycle)

    # std
    std_time = np.sqrt(np.sum((times - mean_time) ** 2) / (num_trials - 1))
    time_err.append(std_time / np.sqrt(num_trials))

    std_cycle = np.sqrt(np.sum((cycles - mean_cycle) ** 2) / (num_trials - 1))
    cycle_err.append(std_cycle / np.sqrt(num_trials))
    print(f"  std_time = {std_time:.6f}, err_time = {time_err[-1]:.6f}")
    print(f"  std_cycle = {std_cycle:.6f}, err_cycle = {cycle_err[-1]:.6f}")
```

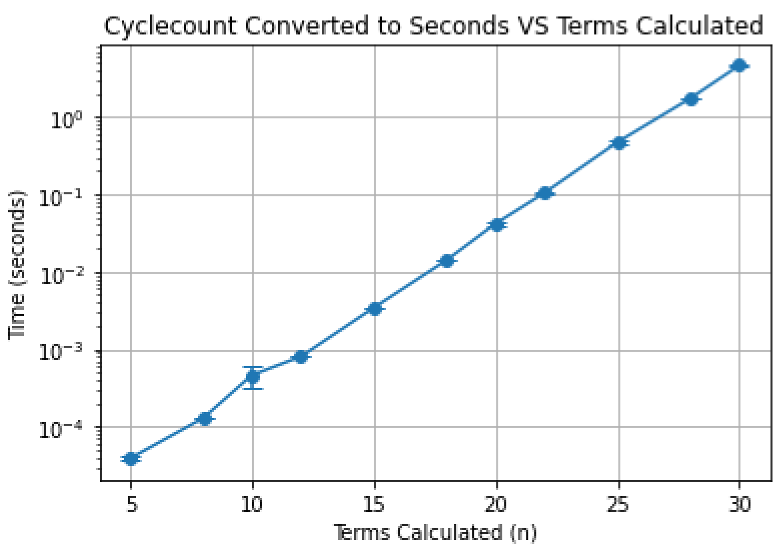
```
--- n = 5 ---
  trial 0: time = 0.000074 s, cycles = 29718
  trial 1: time = 0.000052 s, cycles = 25000
  trial 2: time = 0.000051 s, cycles = 24479
  trial 3: time = 0.000049 s, cycles = 23665
  trial 4: time = 0.000048 s, cycles = 23708
  mean_time = 0.000055 s
  mean_cycle = 25314.0
  std_time = 0.000011, err_time = 0.000005
  std_cycle = 2524.355066, err_cycle = 1128.925905

--- n = 8 ---
  trial 0: time = 0.000180 s, cycles = 88891
  trial 1: time = 0.000143 s, cycles = 83242
  trial 2: time = 0.000140 s, cycles = 82664
  trial 3: time = 0.000140 s, cycles = 82484
  trial 4: time = 0.000137 s, cycles = 80792
  mean_time = 0.000148 s
  mean_cycle = 83614.6
  std_time = 0.000018, err_time = 0.000008
  std_cycle = 3087.401626, err_cycle = 1380.727982

--- n = 10 ---
  trial 0: time = 0.000339 s, cycles = 207238
  trial 1: time = 0.001081 s, cycles = 687519
  trial 2: time = 0.000329 s, cycles = 203582
  trial 3: time = 0.000324 s, cycles = 201614
  trial 4: time = 0.000324 s, cycles = 202431
  mean_time = 0.000479 s
  mean_cycle = 300476.8
  std_time = 0.000336, err_time = 0.000150
  std_cycle = 216373.850543, err_cycle = 96765.327673

--- n = 12 ---
  trial 0: time = 0.000823 s, cycles = 522679
  trial 1: time = 0.000807 s, cycles = 516022
  trial 2: time = 0.000805 s, cycles = 514414
  trial 3: time = 0.000812 s, cycles = 519169
  trial 4: time = 0.000806 s, cycles = 515516
  mean_time = 0.000811 s
  mean_cycle = 517560.0
  std_time = 0.000007, err_time = 0.000003
  std_cycle = 3362.801585, err_cycle = 1503.890588

--- n = 15 ---
  trial 0: time = 0.003504 s, cycles = 2264527
  trial 1: time = 0.003372 s, cycles = 2182580
  trial 2: time = 0.003571 s, cycles = 2309944
  trial 3: time = 0.003363 s, cycles = 2176923
  trial 4: time = 0.003356 s, cycles = 2173125
  mean_time = 0.003433 s
  mean_cycle = 2221419.8
  std_time = 0.000098, err_time = 0.000044
  std_cycle = 62280.911319, err_cycle = 27852.870282

--- n = 18 ---
  trial 0: time = 0.014334 s, cycles = 9303733
  trial 1: time = 0.014102 s, cycles = 9155820
  trial 2: time = 0.014169 s, cycles = 9199638
  trial 3: time = 0.014101 s, cycles = 9155804
```

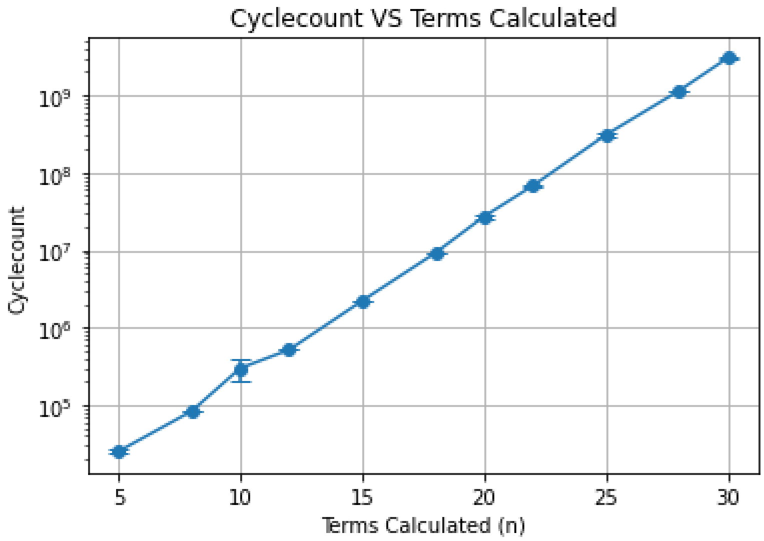
```
  trial 4: time = 0.014187 s, cycles = 9211677
  mean_time = 0.014179 s
  mean_cycle = 9205334.4
  std_time = 0.000095, err_time = 0.000043
  std_cycle = 60539.033609, err_cycle = 27073.878888

--- n = 20 ---
  trial 0: time = 0.036955 s, cycles = 24007753
  trial 1: time = 0.039717 s, cycles = 25801237
  trial 2: time = 0.053369 s, cycles = 34671425
  trial 3: time = 0.042405 s, cycles = 27547825
  trial 4: time = 0.036910 s, cycles = 23979806
  mean_time = 0.041871 s
  mean_cycle = 27201609.2
  std_time = 0.006817, err_time = 0.003049
  std_cycle = 4428876.523748, err_cycle = 1980653.794211

--- n = 22 ---
  trial 0: time = 0.096583 s, cycles = 62764395
  trial 1: time = 0.114845 s, cycles = 74630834
  trial 2: time = 0.098347 s, cycles = 63907237
  trial 3: time = 0.096155 s, cycles = 62489630
  trial 4: time = 0.115354 s, cycles = 74963700
  mean_time = 0.104257 s
  mean_cycle = 67751159.2
  std_time = 0.009934, err_time = 0.004442
  std_cycle = 6455190.057890, err_cycle = 2886848.755425

--- n = 25 ---
  trial 0: time = 0.427066 s, cycles = 277572020
  trial 1: time = 0.433800 s, cycles = 281949555
  trial 2: time = 0.511557 s, cycles = 332492205
  trial 3: time = 0.568255 s, cycles = 369340791
  trial 4: time = 0.426703 s, cycles = 277338097
  mean_time = 0.473476 s
  mean_cycle = 307738533.6
  std_time = 0.063931, err_time = 0.028591
  std_cycle = 41553756.960219, err_cycle = 18583405.056711

--- n = 28 ---
  trial 0: time = 1.749498 s, cycles = 1137152977
  trial 1: time = 1.753391 s, cycles = 1139682760
  trial 2: time = 1.749792 s, cycles = 1137345951
  trial 3: time = 1.746871 s, cycles = 1135446388
  trial 4: time = 1.761321 s, cycles = 1144837000
  mean_time = 1.752174 s
  mean_cycle = 1138893015.2
  std_time = 0.005615, err_time = 0.002511
  std_cycle = 3648802.261836, err_cycle = 1631793.978784

--- n = 30 ---
  trial 0: time = 4.782565 s, cycles = 3108644274
  trial 1: time = 4.549037 s, cycles = 2956855422
  trial 2: time = 4.606897 s, cycles = 2994460727
  trial 3: time = 4.790235 s, cycles = 3113632050
  trial 4: time = 4.556496 s, cycles = 2961702639
  mean_time = 4.657046 s
  mean_cycle = 3027059022.4
  std_time = 0.120193, err_time = 0.053752
  std_cycle = 78124409.773294, err_cycle = 34938298.191027
```

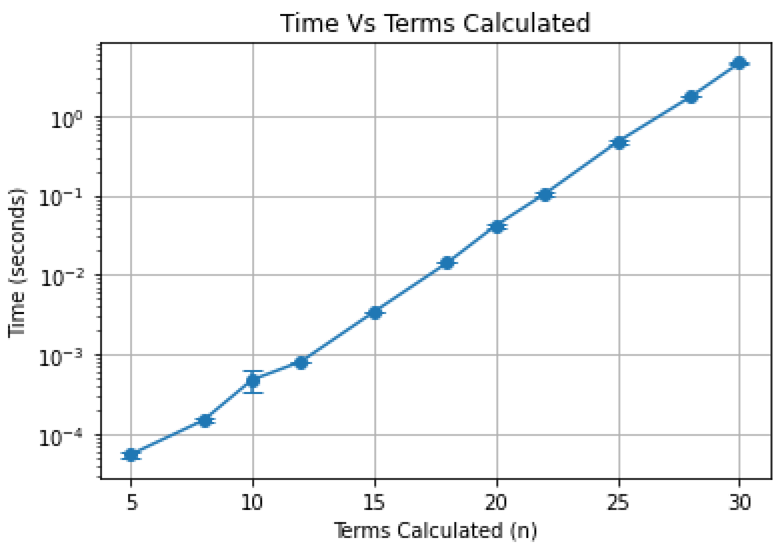
plot!

In [14]:
```
freq = 650e6

cycle_time_avg = np.array(cycle_avg) / freq
cycle_time_err = np.array(cycle_err) / freq
```

In [17]:
```
import matplotlib.pyplot as plt

plt.figure()
plt.errorbar(n_values, cycle_time_avg, yerr=cycle_time_err, fmt='o-', capsize=5
plt.yscale("log")
plt.xlabel("Terms Calculated (n)")
plt.ylabel("Time (seconds)")
plt.title("Cyclecount Converted to Seconds VS Terms Calculated")
plt.grid(True)
plt.show()

plt.figure()
plt.errorbar(n_values, cycle_avg, yerr=cycle_err, fmt='o-', capsize=5)
plt.yscale("log")
plt.xlabel("Terms Calculated (n)")
plt.ylabel("Cyclecount")
plt.title("Cyclecount VS Terms Calculated")
plt.grid(True)
plt.show()

plt.figure()
plt.errorbar(n_values, time_avg, yerr=time_err, fmt='o-', capsize=5)
plt.yscale("log")
plt.xlabel("Terms Calculated (n)")
plt.ylabel("Time (seconds)")
plt.title("Time Vs Terms Calculated")
plt.grid(True)
plt.show()
```




Cyclecount VS Terms Calculated
Cyclecount
Terms Calculated (n)


Time Vs Terms Calculated
Time (seconds)
Terms Calculated (n)

In [ ]: