In [2]:

```python
import ctypes

# Load the shared library
libPMU = ctypes.CDLL('./libMyLib.so')

# Initialize the PMU counters (arguments: do_reset, enable_divider)
libPMU.init_pmu_counters(ctypes.c_int(1), ctypes.c_int(0))

# Get the cycle count
cycle_count = libPMU.get_cycle_count()
print("Cycle count:", cycle_count)
```

```
Cycle count: 1143062
```

In [ ]:

```python

```

A3.2: recur_fibo source code

In [ ]:

```
import ctypes
import time
import numpy as np
import random

# Load the shared library
libPMU = ctypes.CDLL('./libPMU.so')

# Initialize PMU counters
def init_pmu_counters():
    libPMU.init_pmu_counters(1, 0)

# Get cycle count
def get_cycle_count():
    return libPMU.get_cycle_count()

# Recursive Fibonacci function
def recur_fibo(n):
    return n if n <= 1 else recur_fibo(n-1) + recur_fibo(n-2)

# Run timing test
def run_fibo_timing_test(terms):
    results = {'cycles': [], 'times': [], 'cycle_errors': [], 'time_errors': []}

    for n in terms:
        cycles, times = [], []
        for _ in range(3): # Three trials
            init_pmu_counters()

            # Get 'before' time and cycle count
            start_time = time.time()
            start_cycle = get_cycle_count()

            # Run the recur_fibo function
            recur_fibo(n)

            # Get 'after' cycle count and time
            end_cycle = get_cycle_count()
            end_time = time.time()

            # Calculate the differences
            cycle_diff = end_cycle - start_cycle
            time_diff = end_time - start_time

            cycles.append(cycle_diff)
            times.append(time_diff)

        # Calculate average
        results['cycles'].append(np.mean(cycles))
        results['times'].append(np.mean(times))

        # Calculate standard error
        results['cycle_errors'].append(np.std(cycles) / np.sqrt(3))
        results['time_errors'].append(np.std(times) / np.sqrt(3))
```

```python
    return results

# Randomly select 15 terms from 1 to 30
random_terms = random.sample(range(1, 31), 15)
random_terms.sort()

# Run the timing test
results = run_fibo_timing_test(random_terms)

# Convert cycle counts to time using CPU frequency (Hz)
cpu_freq = 325.00 * 1e6  # Convert MHz to Hz

cycle_times = [count / cpu_freq for count in results['cycles']]
cycle_errors = [error / cpu_freq for error in results['cycle_errors']]

# Print the results in a format suitable for plotting
print("Term, Avg Time (s), Time Error, Avg Cycle Time (s), Cycle Time Error")
for i, n in enumerate(random_terms):
    print(f"{n}, {results['times'][i]}, {results['time_errors'][i]}, {cycle_times[i]},
```