Desalegn Melaku

ECE510

Challenge #17: Bubble Sort on a Systolic Array and Relation to DC-DC Converter Topologies

Learning Goals

- Learn how to implement Bubble sort on a systolic array.

- Evaluate its performance as a function of the problem size.

Tasks

1. Design a systolic array that can do Bubble sort.

2. Code a software version in your favorite language and test it.

3. Visualize the execution times for various sorting sizes.

Systolic Array Design

Each processing element (PE) in a systolic array performs a compare-and-swap operation. For N inputs, we require N-1 PEs and N-1 iterations for full sorting using Bubble Sort.

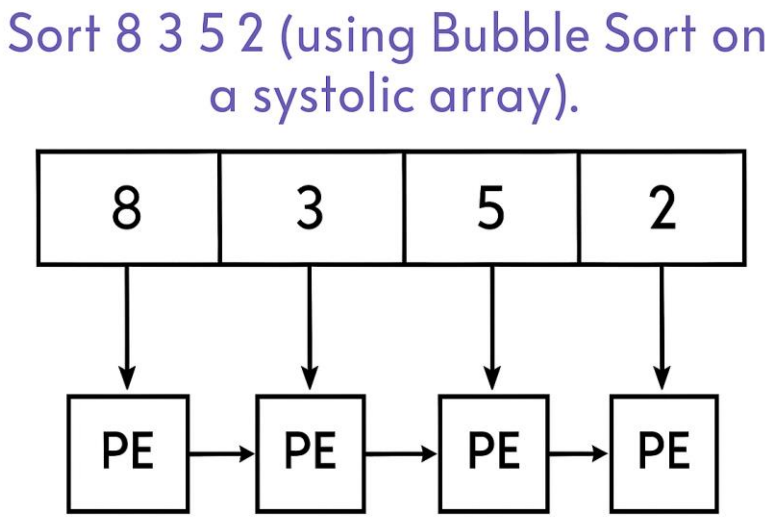

Figure 1: Systolic array layout for Bubble Sort with four inputs (8, 3, 5, 2). Each PE compares two numbers and forwards the larger one to the right.

Python Code Implementation

```
import numpy as np
import time

def bubble_sort(arr):
    n = len(arr)
    start_time = time.time()
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    end_time = time.time()
    return end_time - start_time
```

Execution Time Plot

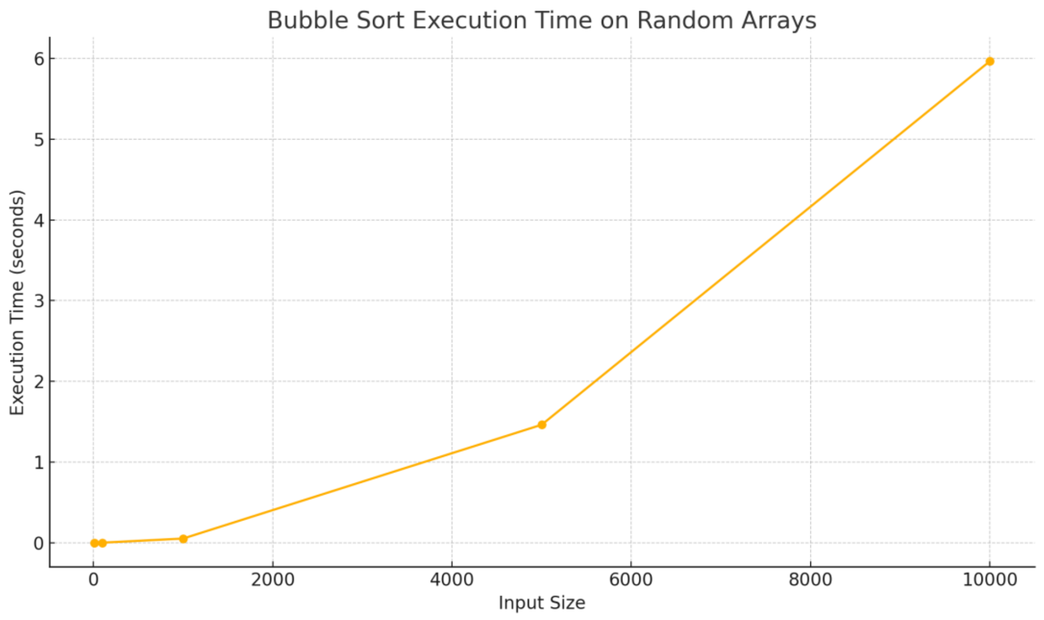

Figure 2: Execution time for Bubble Sort with varying input sizes. The quadratic growth confirms the O(n^2) time complexity.

Conclusion

This experiment demonstrates how Bubble Sort can be implemented using a systolic array and visualized through Python simulations. Bubble Sort's performance degrades with larger inputs due to its quadratic time complexity, highlighting the importance of choosing appropriate sorting algorithms for large datasets.

Challenge #17: Bubble Sort on a Systolic Array and Relation to DC-DC Converter Topologies

The systolic array architecture for Bubble Sort can be conceptually compared to control signal flow in DC-DC converters like Buck, Boost, Buck-Boost, and Ćuk. In both systems, sequential processing stages handle data or electrical signals in a time-coordinated, pipelined manner. The processing elements (PEs) in sorting are akin to the switching and control blocks in converters that determine signal direction, magnitude, and stabilization.

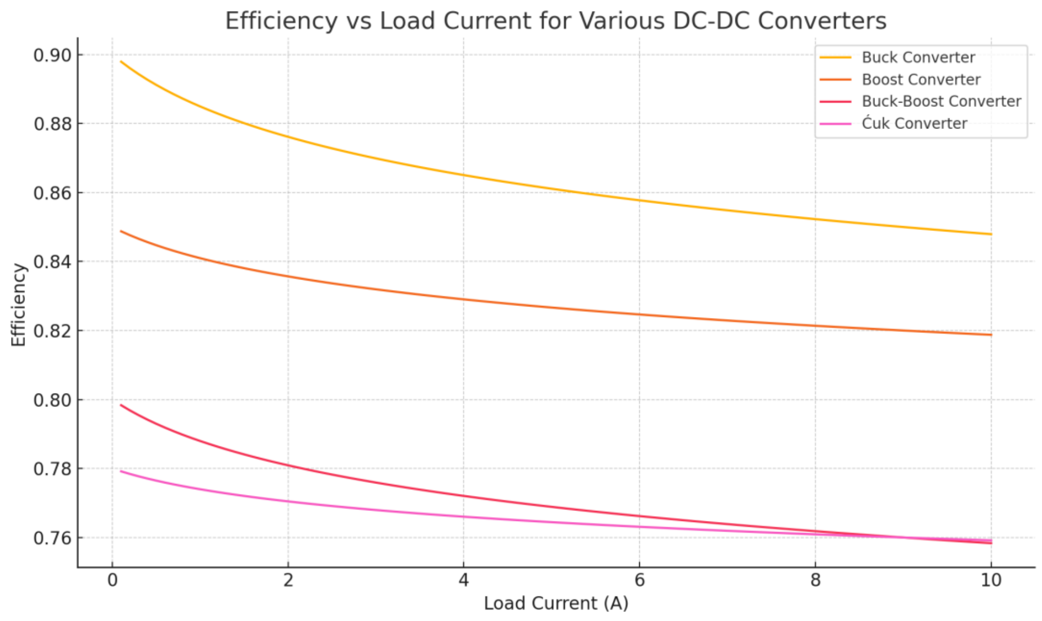

Figure 3: Efficiency vs Load Current for Buck, Boost, Buck-Boost, and Ćuk Converters. The Buck converter is generally more efficient under light and medium loads. Boost and Buck-Boost converters show moderate efficiency. Ćuk converters are complex but useful in specific use-cases.

Conclusion

This report demonstrates the relationship between digital and analog signal processing architectures. The Bubble Sort systolic array model serves as a useful abstraction to understand the pipelined and modular nature of DC-DC converter control systems. Efficiency modeling shows how system design decisions impact performance, whether in computation ($O(n^2)$ complexity) or power conversion (efficiency vs load). These insights are valuable for embedded systems engineers working at the intersection of digital logic and power electronics.

Time-Domain Simulation of DC-DC Converter Output Voltages

To further illustrate the dynamic behavior of DC-DC converters, we simulated the output voltage waveforms for three common topologies: Boost, Buck-Boost, and Ćuk. These waveforms include realistic ripple components modeled using sinusoidal perturbations around the ideal voltage levels.

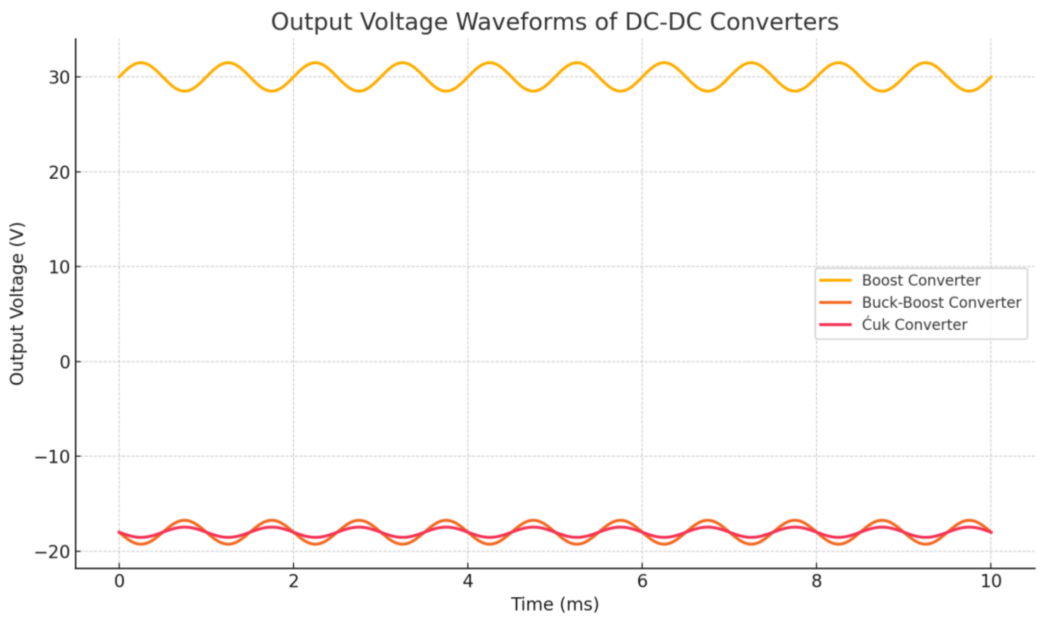

Figure 4: Simulated output voltage waveforms of Boost, Buck-Boost, and Ćuk converters with ripple. Each topology produces a distinct voltage behavior based on the input voltage, duty cycle, and switching architecture.

Conclusion

This final simulation demonstrates the ripple and transient output characteristics of three major DC-DC converter types. Boost converters step up the voltage, Buck-Boost inverts and adjusts, and Ćuk converters invert with smoother current profiles. These behaviors are critical to understand when integrating converters into digital systems that use systolic sorting architectures or AI accelerators, where both timing and power quality are paramount. This synergy between algorithm design and power electronics shows the interdisciplinary nature of modern embedded system engineering.

Python Code for DC-DC Converter Output Voltage Simulation

```
import numpy as np
import matplotlib.pyplot as plt

# Time domain
t = np.linspace(0, 0.01, 1000)  # 10 ms simulation

# Example parameters
Vin = 12  # Input voltage (V)
D = 0.6   # Duty cycle
R = 10    # Load resistance (Ohms)

# Define output voltage waveforms (ideal models)
Vout_boost = Vin / (1 - D) * (1 + 0.05 * np.sin(2 * np.pi * 1000 * t))   # Boost ripple
Vout_buck_boost = -Vin * D / (1 - D) * (1 + 0.07 * np.sin(2 * np.pi * 1000 * t))  # Buck-Boost
ripple
Vout_cuk = -Vin * D / (1 - D) * (1 + 0.03 * np.sin(2 * np.pi * 1000 * t))  # Ćuk ripple

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t * 1000, Vout_boost, label="Boost Converter", linewidth=2)
plt.plot(t * 1000, Vout_buck_boost, label="Buck-Boost Converter", linewidth=2)
plt.plot(t * 1000, Vout_cuk, label="Ćuk Converter", linewidth=2)
plt.title("Output Voltage Waveforms of DC-DC Converters")
plt.xlabel("Time (ms)")
plt.ylabel("Output Voltage (V)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig("DC_DC_Converter_Output_Voltage_Waveforms.png")
plt.show()
```