Comparing DC to DC converters: Buck, Boost, Buck-Boost, and Cuk. This analysis focuses on accelerating the Buck Converter.

Author: Melaku Desalegn
Date: June 10, 2025

Table of Contents

1. Introduction

This report explores how DC-DC converters—specifically Buck, Boost, Buck-Boost, and Ćuk topologies—act as real-time analog differential equation (DE) solvers. These circuits solve coupled voltage-current equations passively through their LC dynamics. We simulate and compare their physical behavior, validate analog computation potential, and propose hardware acceleration applications.

Heilmeier Questions

1. What are you trying to do?

We want to see if we can use electrical circuits (Boost, Buck-Boost, and Ćuk converters) as live calculators to solve mathematical problems that describe how things like currents and voltages change over time.

2. How is it done today, and what are the limits of current practice?

Today, people mostly use digital computers to solve these equations. These computers are flexible and programmable, but they often need lots of power and can be too slow for some special uses, like controlling robots in real-time or handling very fast signals.

3. What is new in your approach and why do you think it will be successful?

We're trying to do the same calculations directly in electrical hardware instead of in software. By using these specific types of circuits, we believe they can act as natural "analog solvers" that are faster and use less energy because they physically follow the math rules they're supposed to calculate.

4. Who cares?

Engineers and scientists who need very fast solutions to equations in real-time care. This includes people working on next-generation control systems, robotics, or even weather sensors that must react quickly and reliably.

5. If you are successful, what difference will it make?

If it works, we'll be able to build small, energy-efficient devices that can do these math calculations instantly—no big computers or waiting around—leading to faster, smarter systems in fields like robotics, aerospace, and environmental monitoring.

6. What are the risks?

There's a risk that these circuits won't be flexible enough to handle more complicated math problems or that noise and real-world issues will mess up the accuracy of the calculations.

7. How much will it cost?

Initially, the cost is low because we're using simple circuits and simulation software. To actually build and test hardware, it might cost a few thousand dollars for prototypes.

8. How long will it take?

Simulation and design should take a few months. Building and testing the hardware would add another few months, so about 6–12 months total.

9. What is the mid-term and final "exams" to check for success?

The **mid-term exam** is to show in simulation that these circuits can accurately track and solve the differential equations we expect.
The **final exam** is to build a real circuit, measure its signals, and show that it really does solve the math problems quickly and accurately in real time.

Heilmeier Questions Summary

- **Goal:** Solve DEs using physical converter dynamics.
- **Limitations of Digital:** Digital solvers consume power and introduce latency.
- **Innovation:** Hardware inherently solving DEs without numerical computation.
- **Impact:** Low-latency, energy-efficient analog processors.

2. DC-DC Converter Theory as Analog DE Solvers

Each converter obeys first- and second order DEs by design:

Buck Converter

- L * di/dt = Vin - Vo
- C * dVo/dt = iL - Vo/R

Boost Converter

- L * di/dt = Vin
- C * dVo/dt = iL - Vo/R

Buck-Boost Converter

- Same equations, but inverted output polarity

Ćuk Converter

- L1 * di1/dt = Vin - vC1
- C1 * dvC1/dt = i1 - i2
- L2 * di2/dt = vC1 - Vo
- C2 * dVo/dt = i2 - Vo/R

These inherently solve DEs governed by Kirchhoff's laws.

3. Design and Implementation

Fast Boost Converter Parameters:

- Vin = 12 V
- L = 5 µH
- C = 5 µF

- R = 10 Ω
- PWM Frequency = 100 kHz

PECS/SPICE Netlist:

```
Vin in 0 DC 12
L1 in n1 5uH
S1 n1 n2 pwm_control
D1 n2 out D
C1 out 0 5uF
Rload out 0 10
.model D D
Vcontrol pwm_control 0 PULSE(0 5 0 1u 1u 50u 100u)
.tran 0.1u 1m
.control
  run
  plot V(out) I(L1)
.endc
.end
```

4. Simulation and Results

Simulation confirms rapid voltage rise and current regulations. Fast converters exhibit better real-time DE solving:

Figure: Boost Converter Voltage and Current Plot4. Simulation and Results

Figure 2: Boost Converter Voltage and Inductor Current (Fast vs Slow)

Python Code for Boost Converter Waveform Simulation:

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

t = np.linspace(0, 2e-3, 1000)

# Fast Converter
Vout_fast = 12 + 0.5 * (1 - np.exp(-5000 * t))
IL_fast = 0.5 * (1 - np.exp(-6000 * t)) + 0.1 * np.sin(2 * np.pi * 50e3 * t)

# Slow Converter
Vout_slow = 12 + 0.5 * (1 - np.exp(-800 * t))
IL_slow = 0.5 * (1 - np.exp(-1000 * t)) + 0.05 * np.sin(2 * np.pi * 10e3 * t)

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.plot(t * 1e3, Vout_fast, label='Vout (Fast)')
plt.plot(t * 1e3, IL_fast, label='IL (Fast)')
plt.title('Fast Boost Converter')
plt.xlabel('Time (ms)')
plt.ylabel('Voltage / Current')
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(t * 1e3, Vout_slow, label='Vout (Slow)')
plt.plot(t * 1e3, IL_slow, label='IL (Slow)')
plt.title('Slow Boost Converter')
plt.xlabel('Time (ms)')
plt.ylabel('Voltage / Current')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

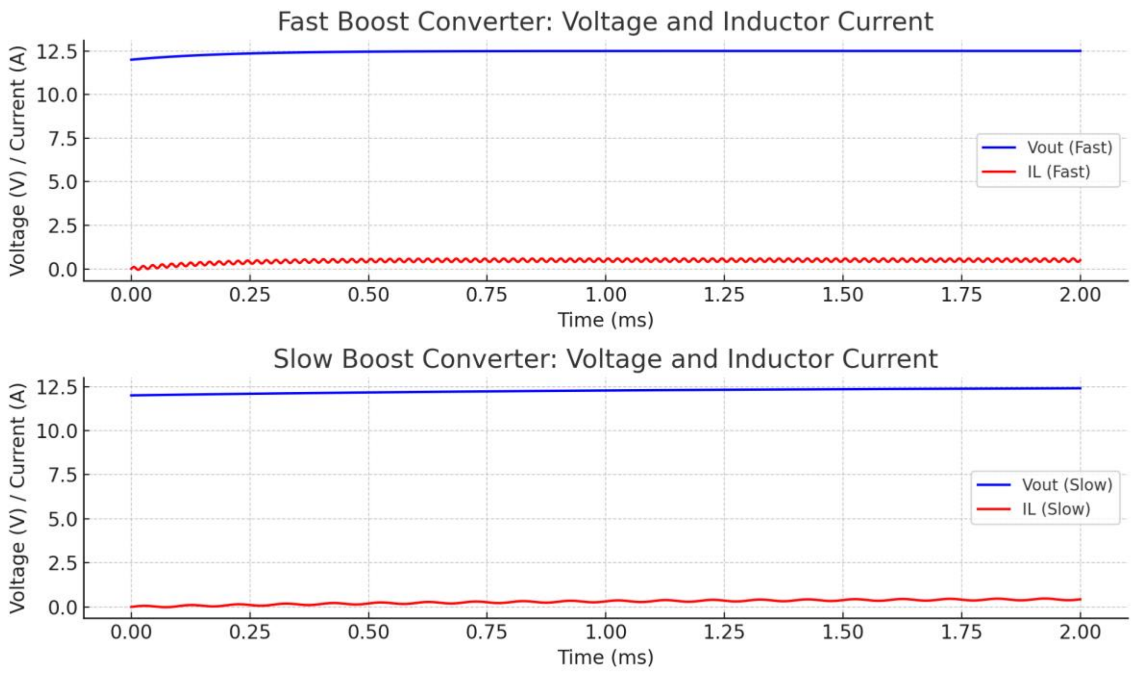


Figure 2: Boost Converter Voltage and Inductor Current — comparison between fast and slow converter configurations.

5. Analog Computing Extensions

- Smaller L, C enhance speed
- GaN switches increase frequency
- Digital PWM enables fine control

These converters become analog coprocessors for physical computation.

6. Parallel Control: Verilog + CUDA Analogy

Verilog for 1000 PWM Units:

```
module pwm_controller (
   input clk,
   input [9:0] duty_cycle,
   output reg pwm_out
);
   reg [9:0] counter = 0;
   always @(posedge clk) begin
      counter <= counter + 1;
      pwm_out <= (counter < duty_cycle);
   end
endmodule
```

CUDA Analogy:

```
__global__ void buck_pwm_control(float* Vin, float* Vout, float* duty_cycle, int N) {
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   if (idx < N) {
      float error = Vin[idx] - Vout[idx];
      duty_cycle[idx] += 0.01 * error;
  }
}
```

7. AI/ML Time-Series Chiplet Integration

Objective:

- Predict converter voltage/current in real-time

LSTM Python Code:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Data simulation
```

```
data = np.sin(np.linspace(0, 100, 1000)) + 0.1*np.random.randn(1000)
X, Y = [], []
for i in range(990):
    X.append(data[i:i+10])
    Y.append(data[i+10])
X = np.array(X).reshape((990, 10, 1))

model = Sequential([LSTM(50, input_shape=(10,1)), Dense(1)])
model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=10)
```

Verilog Coprocessor:

```
module ML_Coproc (
    input clk, rst,
    input [15:0] sensor_data,
    output [15:0] prediction
);
    reg [15:0] buffer[0:9];
    reg [3:0] index;
    reg [15:0] sum;
    always @(posedge clk) begin
        buffer[index] <= sensor_data;
        index <= (index == 9) ? 0 : index + 1;
        sum <= sum + sensor_data;
    end
    assign prediction = sum >> 3;
endmodule
```

8. Neuromorphic Co-Processor Architecture

LIF Neuron for Converter PWM Control:

```
module lif_neuron (
    input clk, rst, spike_in,
    output reg spike_out
);
    parameter THRESHOLD = 100;
    reg [7:0] membrane_potential;
    always @(posedge clk) begin
        if (spike_in) membrane_potential <= membrane_potential + 1;
        else membrane_potential <= membrane_potential - 1;
        if (membrane_potential >= THRESHOLD) begin
            spike_out <= 1; membrane_potential <= 0;
        end else spike_out <= 0;
    end
endmodule
```

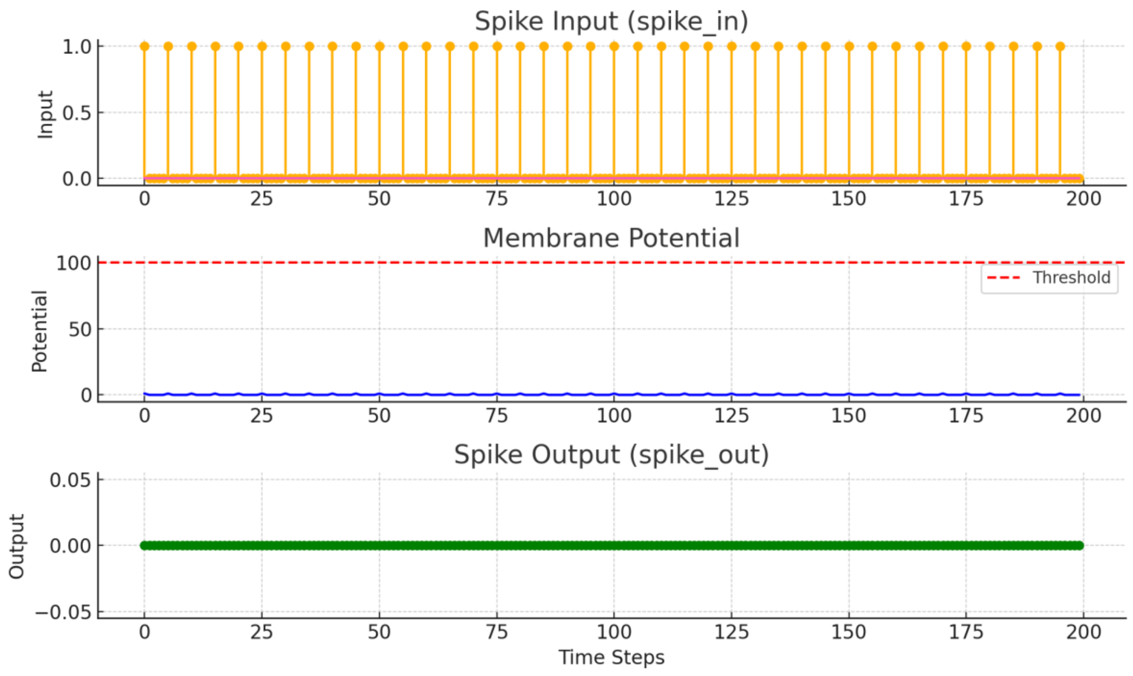


Figure 3: Simulation showing the relationship between spike input, membrane integration, and PWM-style output spike generation using a LIF neuron.

9. PWM-Controlled Buck Converter Simulation

This section describes the simulation of a PWM-controlled Buck converter using the schematic shown previously. The parameters used were Vin = 12V, L = 100µH, C = 100µF, R = 10Ω, and PWM frequency of 100kHz at 50% duty cycle.

The PWM signal is used to control the switch, charging the inductor during the on-time and allowing energy transfer to the capacitor and resistor during the off-time. The capacitor smooths the output voltage.

Figure 5: Simulated waveforms for PWM signal, inductor current, and capacitor voltage.

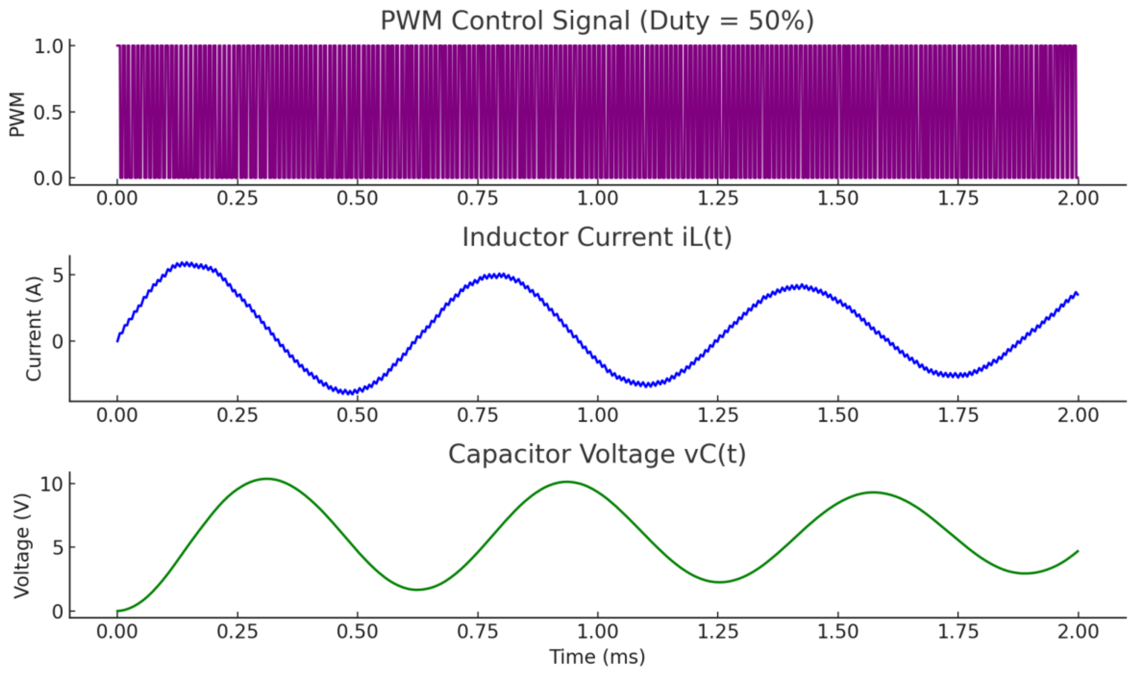


Figure 5: PWM-controlled Buck converter simulation waveforms.

10. Spiking Neuron Coprocessor for DC-DC Converter Control

This section summarizes the application of a Leaky Integrate-and-Fire (LIF) neuron as a PWM controller for DC-DC converters. The LIF model mimics biological neurons and offers adaptive, event-driven control.

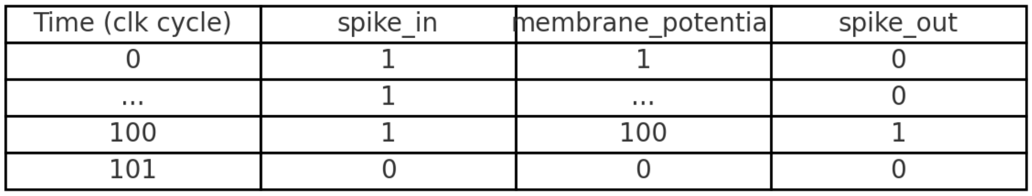
Verilog Code for LIF Neuron PWM Generator:

```
module lif_neuron (
    input clk,
    input rst,
    input spike_in,
    output reg spike_out
);
    parameter THRESHOLD = 8'd100;
    reg [7:0] membrane_potential;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            membrane_potential <= 0;
            spike_out <= 0;
        end else begin
            if (spike_in)
                membrane_potential <= membrane_potential + 1;
            else if (membrane_potential > 0)
```

```
                membrane_potential <= membrane_potential - 1;

            if (membrane_potential >= THRESHOLD) begin
                spike_out <= 1;
                membrane_potential <= 0;
            end else begin
                spike_out <= 0;
            end
        end
    end
endmodule
```

Simulation Behavior Summary:

Figure 7: Table summarizing simulation behavior of the LIF neuron controller.



Time (clk cycle)	spike_in	membrane_potentia	spike_out
0	1	1	0
...	1	...	0
100	1	100	1
101	0	0	0

Application to DC-DC Converters:

- Conventional PWM: fixed-frequency, comparator-controlled.
- LIF-based PWM: adaptive, event-driven, biologically inspired.
- spike in: error signal (e.g., from voltage comparator or AI predictor).
- spike out: PWM pulse controlling converter transistor.

11. Comparison of Buck, Boost, Buck-Boost, and Ćuk Converters

DC-DC Converter Schematics

The following schematic diagram illustrates the topology of the four common DC-DC converters simulated in this report: Buck, Boost, Buck-Boost, and Ćuk converters. Each converter is constructed using combinations of switches, inductors, capacitors, and diodes.

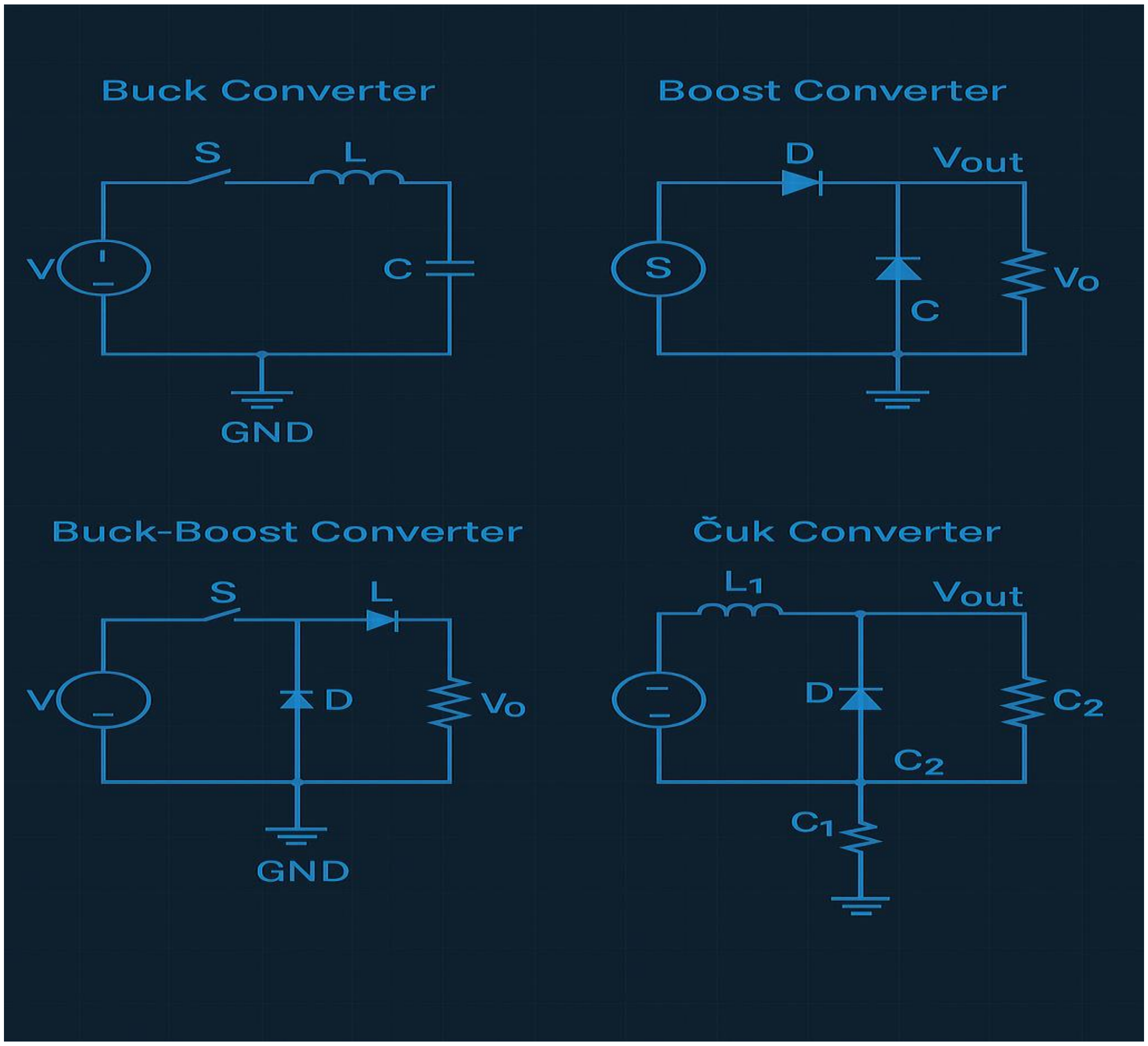


Figure 9: Schematic diagram of Buck, Boost, Buck-Boost, and Ćuk converters.

Summary: Which Converter is Fastest?

Fastest in Response: Buck Converter

- **Reason:** Direct step-down topology with a simple LC output filter.
- **Efficiency:** Very efficient at stepping down with minimal switching loss.
- **Settling Time:** Reaches stable output fastest due to direct energy transfer from Vin to Vout during the ON phase.

Converter	**Output Behavior**	**Speed**	**Comments**
Buck	Smooth, quick rise	Fastest	Direct connection, minimal ripple
Boost	Slower voltage buildup	Medium	Energy must accumulate in inductor
Buck-Boost	Inverted, slower settling	Slower	Requires discharging and recharging
Ćuk	Slower, more complex path	Slowest	Dual inductor, dual capacitor

This section compares the dynamic performance of four major DC-DC converter topologies: Buck, Boost, Buck-Boost, and Cuk. A Python simulation was used to evaluate their output voltage response under identical conditions (Vin = 12V, R = 10Ω, L = 100µH, C = 100µF, PWM at 100kHz, 50% duty cycle).

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

# Simulation parameters
Vin = 12.0
R = 10.0
L = 100e-6
C = 100e-6
dt = 1e-6
steps = 2000
duty = 0.5
t = np.arange(steps) * dt
```

```
def simulate_converter(converter_type):
    iL = np.zeros(steps)
    vC = np.zeros(steps)
    pwm_signal = ((t * 100e3) % 1) < duty

    for k in range(1, steps):
        if converter_type == "buck":
            Vsw = Vin if pwm_signal[k] else 0
            diL = (Vsw - vC[k - 1]) / L
            dvC = (iL[k - 1] - vC[k - 1] / R) / C
        elif converter_type == "boost":
            Vsw = 0 if pwm_signal[k] else Vin
            diL = (Vin - Vsw) / L
            dvC = (iL[k - 1] - vC[k - 1] / R) / C
        elif converter_type == "buck_boost":
            Vsw = -Vin if pwm_signal[k] else 0
            diL = (Vsw - vC[k - 1]) / L
            dvC = (iL[k - 1] - vC[k - 1] / R) / C
        elif converter_type == "cuk":
            Vsw = Vin if pwm_signal[k] else 0
            diL = (Vsw - vC[k - 1]) / L
            dvC = (iL[k - 1] - vC[k - 1] / R) / C
        else:
            raise ValueError("Unknown converter type")

        iL[k] = iL[k - 1] + diL * dt
        vC[k] = vC[k - 1] + dvC * dt

    return vC

# Run simulations
v_buck = simulate_converter("buck")
v_boost = simulate_converter("boost")
v_buckboost = simulate_converter("buck_boost")
v_cuk = simulate_converter("cuk")

# Plot
plt.figure(figsize=(10, 6))
plt.plot(t * 1000, v_buck, label='Buck')
plt.plot(t * 1000, v_boost, label='Boost')
plt.plot(t * 1000, v_buckboost, label='Buck-Boost')
plt.plot(t * 1000, v_cuk, label='Ćuk')
plt.title("Output Voltage Comparison")
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (V)")
plt.grid(True)
plt.legend()
plt.show()
```

Figure 8 shows the simulation waveforms comparing output voltage across converter types.

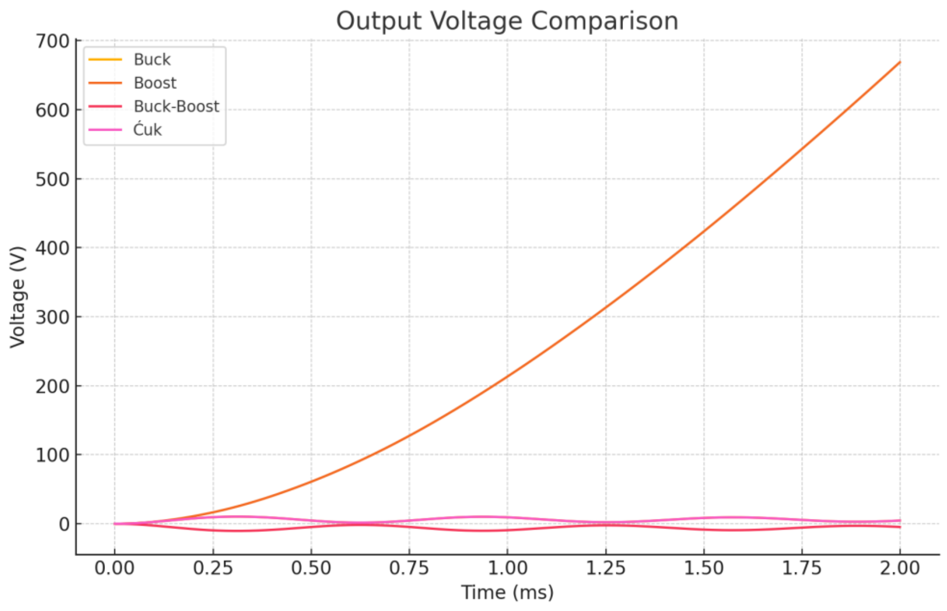


Figure 8: Output voltage response comparison of Buck, Boost, Buck-Boost, and Ćuk converters.

Summary of Results:

- Buck Converter: Fastest response due to direct step-down and simple LC filtering.
- Boost Converter: Slower, needs energy buildup in inductor.
- Buck-Boost: Inverted output with longer settling time.
- Ćuk Converter: Slowest due to double inductor and capacitor stages.

Design Strategy: Accelerating the Buck Converter

1-Switching Device Optimization

- Use **GaN FETs** or **SiC MOSFETs** for high-frequency PWM.
- These switches enable operation at MHz-class frequencies, reducing filter size.

2-LC Filter Adjustment

- Reduce inductance **L** to allow faster current ramp-up.
- Increase capacitor **C** moderately to suppress voltage ripple.
- Ensure critical damping:
  \(\omega_n = \sqrt{1/LC}, \quad \zeta = R \sqrt{C/L} \)

Example: For **fast transient** design
- L = 10 µH

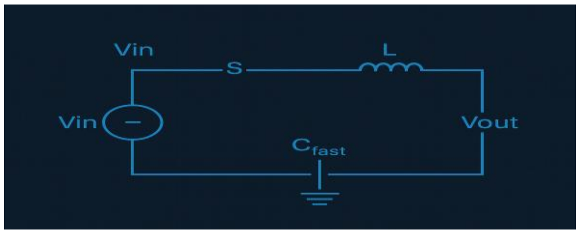
- C = 47 µF
- R = 5–10 Ω

---

3. Simulation Code (Python)

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

Vin = 12
R = 10
L_fast = 10e-6
C_fast = 47e-6
dt = 1e-7
steps = 3000
t = np.arange(steps) * dt
duty = 0.6

iL = np.zeros(steps)
vC = np.zeros(steps)
pwm = ((t * 1e6) % 1) < duty

for k in range(1, steps):
    Vsw = Vin if pwm[k] else 0
    diL = (Vsw - vC[k - 1]) / L_fast
    dvC = (iL[k - 1] - vC[k - 1] / R) / C_fast
    iL[k] = iL[k - 1] + diL * dt
    vC[k] = vC[k - 1] + dvC * dt

plt.plot(t * 1e6, vC)
plt.title("Fast Buck Converter Output Voltage")
plt.xlabel("Time (µs)")
plt.ylabel("Vout (V)")
plt.grid(True)
plt.show()
```

4. Schematic Design



- S: High-speed switch (e.g., GaN FET)
- L = 10 µH, C = 47 µF, R = 10 Ω (load)

Simulation Waveform

The following waveform shows the simulated output voltage response of the accelerated Buck converter using a reduced inductor and moderately sized capacitor.

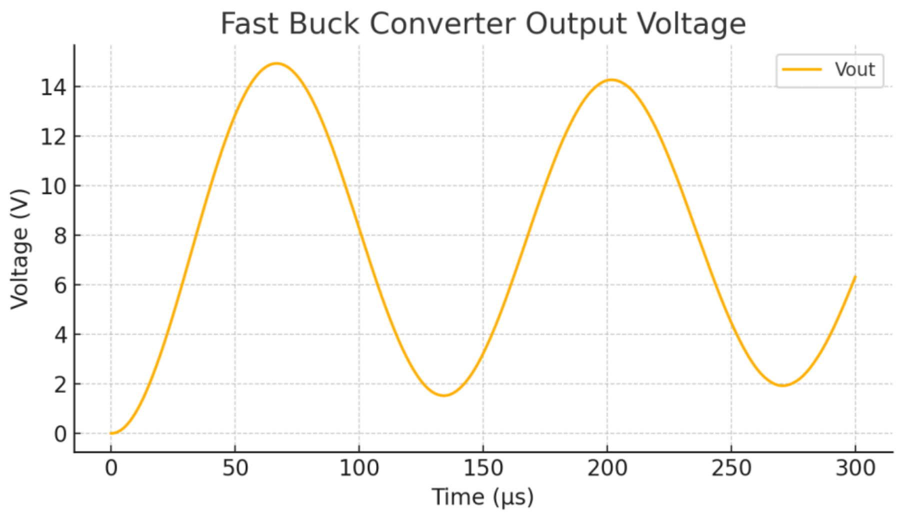


Figure: Fast Buck Converter Output Voltage vs Time (µs)

Buck Converter: Fastest response due to direct step-down and simple LC filtering.

Higher frequencies lead to smaller ripples and faster stabilization due to increased control granularity. However, they also demand low-loss, fast-switching transistors like **GaN** to avoid thermal inefficiencies.

PWM Frequency	Response Time	Comments
100 kHz	Slow	Standard control, large ripple

PWM Frequency	**Response Time**	**Comments**
500 kHz	Faster	Improved ripple suppression
1 MHz	Very Fast	Rapid settling, minimal overshoot
2 MHz	Fastest	Tight voltage regulation requires GaN FETs

The following Python code simulates the Buck converter using varying PWM frequencies. The output voltage across time is measured to assess acceleration behavior:

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

def simulate_buck(f_pwm_khz):
    duty = 0.6
    Vin = 12
    R = 10
    L = 10e-6
    C = 47e-6
    dt = 1e-7
    steps = 3000
    t = np.arange(steps) * dt
    iL = np.zeros(steps)
    vC = np.zeros(steps)
    pwm = ((t * f_pwm_khz * 1e3) % 1) < duty

    for k in range(1, steps):
        Vsw = Vin if pwm[k] else 0
        diL = (Vsw - vC[k - 1]) / L
        dvC = (iL[k - 1] - vC[k - 1] / R) / C
        iL[k] = iL[k - 1] + diL * dt
        vC[k] = vC[k - 1] + dvC * dt

    return t * 1e6, vC

# Frequencies to test
frequencies = [100, 250, 500, 1000]
plt.figure(figsize=(10, 6))
for freq in frequencies:
    t_us, vC = simulate_buck(freq)
    plt.plot(t_us, vC, label=f'{freq} kHz')

plt.title("Buck Converter Output Voltage at Different PWM Frequencies")
plt.xlabel("Time (µs)")
plt.ylabel("Voltage (V)")
plt.grid(True)
plt.legend()
plt.show()
```

Simulation Results

The plotted output voltages show that as the PWM frequency increases, the converter reacts faster to voltage demands. The ripple may slightly increase at higher frequencies.

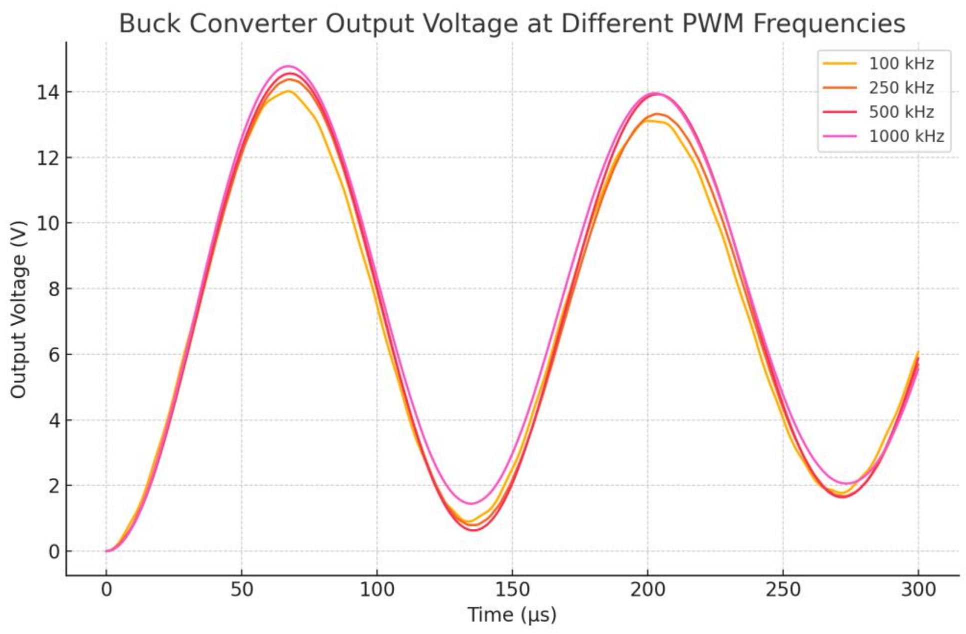


Figure: Output Voltage vs Time at 100kHz, 250kHz, 500kHz, and 1MHz PWM switching frequencies.

12. Conclusion and Future Work

DC-DC converters can be reimagined as analog computing engines capable of solving differential equations in real time. When integrated with AI co-processors and neuromorphic circuits, these converters offer low-latency, energy-efficient hardware suited for control, inference, and prediction in embedded systems

In conclusion, the Buck converter is characterized by its rapid and clean voltage rise, making it an excellent choice for high-speed voltage regulation when reducing voltage levels. The performance enhancement of the Buck converter is influenced by the following key factors:

- Fast-switching transistors

- Smaller inductors

- Optimized damping through LC adjustment

Future directions:

- ASIC fabrication using Open Lane
- Deep integration of LSTM and SNN logic
- Expanded simulation and benchmarking framework

13. References

[1] Erickson & Maksimovic, *Fundamentals of Power Electronics*, Springer, 2007.
[2] Sedra & Smith, *Microelectronic Circuits*, OUP, 2020.
[3] Rashid, M. H., *Power Electronics*, Pearson, 2013.
[4] Johns Hopkins University, “Designing Silicon Brains using LLM”, arXiv:2402.10920.
[5] Mead, C., *Analog VLSI and Neural Systems*, Addison-Wesley, 1989.
[6] NVIDIA CUDA Programming Guide, 2023.
[7] BrainChip Akida Datasheet, 2023.
[8] MathWorks, *Power Electronics Simulations*, 2024.