

## Análise do código do conversor AC-CC *Step-Down*

### 1. Importações e Configurações Iniciais

```
import tkinter as tk
from tkinter import ttk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
from matplotlib.figure import Figure
import matplotlib
```

As seguintes bibliotecas foram importadas:

***tkinter*** e ***ttk***: criação de interface gráfica

***numpy***: cálculos numéricos e simulação.

***matplotlib***: plotagem dos gráficos

***FigureCanvasTkAgg*** e ***NavigationToolbar2Tk***: integram os gráficos do *matplotlib* com auxílio da biblioteca *Tkintermatplotlib.use('TkAgg')*

***Tkintermatplotlib.use('TkAgg')***: Configura o *matplotlib* para usar o *backend TkAgg*, que permite a integração com *Tkinter*

Em seguida, cria-se uma classe principal a partir da qual todas as funções referentes ao código estarão sujeitas a ela (classe raiz – *root*).

### 2. BuckConverterApp

```
class BuckConverterApp:
    def __init__(self, root):
        # Configuração inicial da janela
        self.root = root
        self.root.title("Simulador de Conversor Buck CC-CC")
        self.root.geometry("1200x800")
        self.root.minsize(1000, 700)

        # Inicialização dos componentes
        self.setup_style()
        self.setup_variables()
        self.create_widgets()
        self.run_simulation()
```

Primeiro, estabelece-se uma *class* a qual todas as funções estarão sujeitas. A primeira função subsequente à definição da *class* está dedicada a gerar uma janela principal a partir das especificações passadas, como por exemplo tamanho, geometria e resolução da janela.

### 3. def setup\_style

```
def setup_style(self):
    self.style = ttk.Style()
    self.style.theme_use('clam')

    self.bg_color = '#f0f3f5'
    self.frame_color = '#ffffff'
    self.accent_color = '#4e73df'
    self.text_color = '#2e2e2e'

    self.style.configure('TFrame', background=self.bg_color)
    self.style.configure('TLabel', background=self.bg_color,
foreground=self.text_color)
    self.style.configure('TButton', font=('Segoe UI', 10),
padding=6)
    self.style.configure('Accent.TButton',
background=self.accent_color,
foreground='white', font=('Segoe UI', 10,
'bold'))
```

Define um esquema de cores e aplica estilos escrita a todos os *widgets* através do *tk.Style()* para escolha do tema *clam*, por exemplo.

### 4. def setup\_variables()

```
def setup_variables(self):
    # Parâmetros iniciais
    self.Vin = tk.DoubleVar(value=36.0) # Tensão de entrada
    self.Vout = tk.DoubleVar(value=12.0) # Tensão de saída
    desejada
    self.Iout = tk.DoubleVar(value=2.0) # Corrente de saída
    self.fsw = tk.DoubleVar(value=50000) # Frequência de
    chaveamento (Hz)

    # Resultados calculados
    self.results = {
        'Vavg': tk.StringVar(value='---'), # Tensão média na
carga
        'Vripple': tk.StringVar(value='---'), # Ripple de tensão
        'Iripple': tk.StringVar(value='---'), # Ripple de
corrente
```

```

        'Duty': tk.StringVar(value='---')      # Duty cycle (%)
    }

```

Cria variáveis Tkinter para armazenar tanto os parâmetros ajustáveis quanto os resultados calculados. Assim, pode-se realizar a atualização automática da interface *a posteriori*.

## 5. *def create\_widgets()*

```

def create_widgets(self):
    main_frame = ttk.Frame(self.root)
    main_frame.pack(fill=tk.BOTH, expand=True)

    # Painei esquerdo (controles)
    left_panel = ttk.Frame(main_frame, width=350)
    left_panel.pack(side=tk.LEFT, fill=tk.Y)

    # Painei direito (gráficos)
    right_panel = ttk.Frame(main_frame)
    right_panel.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    self.create_parameter_section(left_panel)
    self.create_results_section(left_panel)
    self.create_graph_section(right_panel)

```

Essa função consiste na criação de widgets para a visualização dos resultados bem como sua organização e disposição na tela para o usuário.

## 6. *def create\_parameter\_section()*

```

def create_parameter_section(self, parent):
    frame = ttk.LabelFrame(parent, text="PARÂMETROS DO
CIRCUITO")

    params = [
        ("Tensão de Entrada (V)", self.Vin),
        ("Tensão de Saída (V)", self.Vout),
        # ... outros parâmetros ...
    ]

    for text, var in params:
        row = ttk.Frame(frame)
        ttk.Label(row, text=text, width=20,
anchor=tk.W).pack(side=tk.LEFT)

```

```

        entry = ttk.Entry(row, textvariable=var, width=10,
justify=tk.RIGHT)
        entry.pack(side=tk.RIGHT)
        entry.bind('<KeyRelease>', self.validate_entry)

        ttk.Button(frame, text="SIMULAR",
command=self.run_simulation,
style='Accent.TButton').pack(fill=tk.X)

```

Cria um formulário com entradas para todos os parâmetros do circuito, vinculando cada entrada a uma variável *Tkinter* . O botão "SIMULAR" aciona a recálculo.

## 7. *def run\_simulation()*

```

def run_simulation(self):
    try:
        # Obter valores da interface
        Vin = self.Vin.get()
        Vout = self.Vout.get()
        fsw = self.fsw.get()
        L = self.L.get()
        C = self.C.get()

        # Verificar se Vin > Vout
        if Vin <= Vout:
            raise ValueError("A tensão de entrada deve ser maior
que a saída!")

        # Calcular duty cycle
        D = Vout / Vin

        # Configurar tempo de simulação
        t_sim = 5e-3 # 5 ms
        dt = 1 / (fsw * 200) # Passo de tempo
        t = np.arange(0, t_sim, dt)

        # Inicializar arrays
        Vout = np.zeros_like(t)
        V_L = np.zeros_like(t)
        V_C = np.zeros_like(t)
        I_L = np.zeros_like(t)

        # Simulação por iteração
        for i in range(1, len(t)):
            # Controle PWM
            if (t[i] * fsw) % 1.0 < D:
                V_L[i] = Vin - V_C[i-1] # Estado ligado
            else:
                V_L[i] = -V_C[i-1] # Estado desligado

```

```

        # Atualizar corrente no indutor
        I_L[i] = I_L[i-1] + (V_L[i] / L) * dt

        # Atualizar tensão no capacitor
        I_C = I_L[i] - (V_C[i-1] / R_load)
        V_C[i] = V_C[i-1] + (I_C / C) * dt

        # Tensão na carga com ESR
        Vout[i] = V_C[i] + (I_C * R_esr)

    # Processar resultados e atualizar interface
    self.process_results(t, Vout, I_L, D)

except Exception as e:
    messagebox.showerror("Erro", f"Falha na
simulação:\n{str(e)}")

```

Aqui há a implementação lógica do funcionamento do circuito, na qual está presente o chaveamento PWM simulando a operação de um MOSFET e a dinâmica da carga e descarga dos componentes ativos do Buck, indutor e capacitor. Além disso, também foi adicionado a resistência ôhmica referente ao capacitor, apenas para tornar a simulação mais próxima possível do real.

## 8. *def process\_results*

```

def process_results(self, t, Vout, I_L, D):
    # Ignorar transitório inicial (últimos 10%)
    start_idx = int(0.9 * len(t))

    # Calcular métricas
    Vavg = np.mean(Vout[start_idx:])
    Vripple = np.max(Vout[start_idx:]) -
np.min(Vout[start_idx:])
    Iripple = np.max(I_L) - np.min(I_L)

    # Atualizar interface
    self.results['Vavg'].set(f"{Vavg:.3f}")
    self.results['Vripple'].set(f"{Vripple:.3f}")
    self.results['Iripple'].set(f"{Iripple:.3f}")
    self.results['Duty'].set(f"{D*100:.1f}")

    # Atualizar gráficos
    self.update_plots(t, Vout, V_L, V_C, Vavg)

```

Apresenta os resultados ao usuário de modo já formatado através da formatação f""" presente de maneira nativa em Python. Os resultados são apresentados dentro da área delimitada nos *widgets*.

### 9. *def create\_graph\_section()* e *def update\_plots()*

```
def create_graph_section(self, parent):
    self.fig = Figure(figsize=(8, 6), facecolor=self.bg_color)
    self.canvas = FigureCanvasTkAgg(self.fig, master=parent)
    self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # Criar subplots
    self.ax1 = self.fig.add_subplot(311) # Tensão na carga
    self.ax2 = self.fig.add_subplot(312) # Tensão no indutor
    self.ax3 = self.fig.add_subplot(313) # Tensão no capacitor

    # Adicionar barra de ferramentas
    toolbar = NavigationToolbar2Tk(self.canvas, parent)
    toolbar.update()

def update_plots(self, t, Vout, V_L, V_C, Vavg):
    # Converter tempo para ms
    t_ms = t * 1000

    # Limpar e atualizar gráficos
    for ax in [self.ax1, self.ax2, self.ax3]:
        ax.clear()

    # Gráfico 1: Tensão na Carga
    self.ax1.plot(t_ms, Vout, 'b', label='Tensão na Carga')
    self.ax1.axhline(y=Vavg, color='r', linestyle='--',
label=f'Média: {Vavg:.2f}V')
    self.ax1.set_title('Tensão na Carga')
    self.ax1.set_ylabel('Tensão (V)')
    self.ax1.legend()
    self.ax1.grid(True)

    # Gráfico 2: Tensão no Indutor
    self.ax2.plot(t_ms, V_L, 'g')
    self.ax2.set_title('Tensão no Indutor')
    self.ax2.set_ylabel('Tensão (V)')
    self.ax2.grid(True)

    # Gráfico 3: Tensão no Capacitor
    self.ax3.plot(t_ms, V_C, 'm')
    self.ax3.set_title('Tensão no Capacitor')
    self.ax3.set_xlabel('Tempo (ms)')
    self.ax3.set_ylabel('Tensão (V)')
    self.ax3.grid(True)

    # Ajustar layout e redesenhar
```

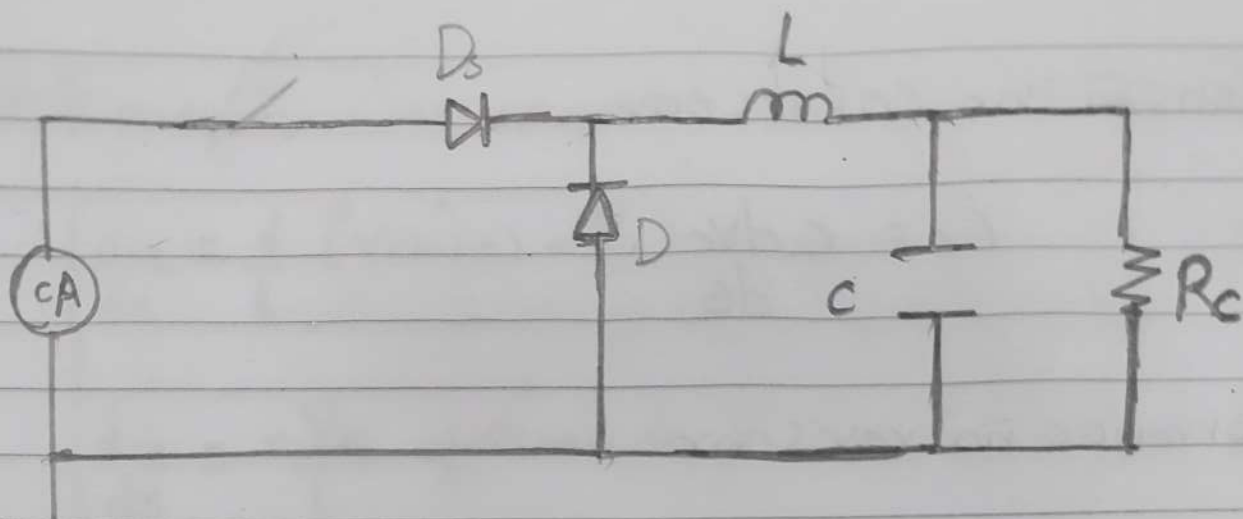
```
self.fig.tight_layout()  
self.canvas.draw()
```

Aqui, esses dois segmentos do código constituem a parte de apresentação gráfica, sendo que para a visualização dos três resultados distintos se é utilizado de *subplots* oriundos da biblioteca *matplotlib*, os quais são definidos, em tamanho e outras características, pela primeira função. Assim, para todas as novas entradas realizadas a função realiza o update dos gráficos presentes nos *subplots* plotando-os novamente.

## 10.Bloco Principal

```
if __name__ == "__main__":  
    root = tk.Tk()  
    app = BuckConverterApp(root)  
    root.mainloop()
```

Por fim, a aplicação é chamada e as funções entram em operação através do “*app*” *BuckConverterApp*. Assim, cria-se a janela principal através da *Tkinter* e instancia-se a aplicação desse modo o loop principal de eventos se inicia até que o usuário deseje encerrar a aplicação.



↳ ESTIPULA-SE A tensão de entrada e saída PARA SE ENCONTRAR, ATRAVÉS DO DUTY CYCLE, OS PARÂMETROS DO CONVERSOR

↳ tensão RMS: (Duty cycle:  $k_3$ )

input: 36V      output: 12V

↳ Diodo Schottky (0,3V - Germânico)

$$V_{rect}(t) = \begin{cases} V_{in}(t) - V_d & \text{Se } V_{in} > V_d \\ 0 & \text{Se } V_{in} = V_d \end{cases}$$

↳ Onda no indutor

$$V_{in} = L \frac{di}{dt} + V_c(t) = 0 =$$



↳ tensão no capacitor

$$i_c = C \frac{dv_c}{dt}$$

↳ Corrente no resistor

$$i_R(t) = \frac{V_d(t)}{R}$$

↳ Corrente no indutor

$$i_L(t) = i_c(t) + i_R(t)$$

Regime estacionário

$$L \frac{di}{dt} + \frac{1}{C} \int (i_L - \frac{V_c}{R}) dt = V_{ch}(t)$$

↳ Bloqueio do diodo Schottky

$$L \frac{di}{dt} + V_c(t) = 0$$

## ↳ Equações Características do Sistema

$$\begin{cases} \frac{di_L}{dt} = \frac{1}{L} (V_{in}(t) - V_C), \text{ quando } V_{in}(t) > V_C \end{cases}$$

$$\begin{cases} \frac{di_L}{dt} = -\frac{V_C}{L}, \text{ quando } V_{in}(t) \leq V_C \text{ (roda livre)} \end{cases}$$

$$\frac{dV_C}{dt} = \frac{1}{C} \cdot (i_L - \frac{V_C}{R})$$

## ↳ EDO do circuito Pós retificação (RLC "Paralelo")

$$\begin{cases} i_R = \frac{V_C}{R} \end{cases}$$

$$\begin{cases} i_L = \frac{1}{L} \cdot \int V_C dt \end{cases}$$

$$\begin{cases} i_C = C \cdot \frac{dV_C}{dt} \end{cases}$$

$$\frac{d^2 V_C}{dt^2} + \frac{1}{RC} \frac{dV_C}{dt} + \frac{1}{LC} V_C = \frac{1}{LC} V_{in}(t)$$

↳ Substituindo os valores dos parâmetros encontramos:

$$\frac{d^2 V}{dt^2} + 10^5 \frac{dV}{dt} + 10^6 = 0 =$$

↳ Assim, com uma EDO homogênea, com coeficientes constantes, de segunda ordem. Pode-se encontrar suas raízes

$$s^2 + 10^5 s + 10^6 = 0$$

$$\Delta = 9,996 \cdot 10^9$$

$$s_1 \approx -10$$

$$s_2 \approx -99990$$

↳ Assim, sua solução homogênea é:

$$V(t) = A e^{-10t} + B e^{-99990t} =$$

↳ Logo sua solução geral é:

$$V(t) = 36 + A \cdot e^{-10t} + B e^{-99990t}$$

↳ Seus coeficientes A e B, são:

$$A = -36,0036$$

$$B = 0,0036$$

↳ Obs.: Condições iniciais nulas

↳ Portanto, temos que:

$$V(t) = 36 - 36,0036 e^{-10t} + 0,0036 e^{-99990t}$$

↳ Assim, seu tempo de amortecimento é:

$$\tau = 40,5 \text{ ms}$$

↳ Prova real

↳ Para  $t = 40,5 \text{ ms}$ , tem-se que:

$$V(t) = 36 - 36,0036 e^{-10 \cdot 40,5 \cdot 10^{-3}} + 0,0036 e^{-99990 \cdot 40,5 \cdot 10^{-3}}$$

$$= V_{out}(t) = 36 - 24,002 + 0 =$$

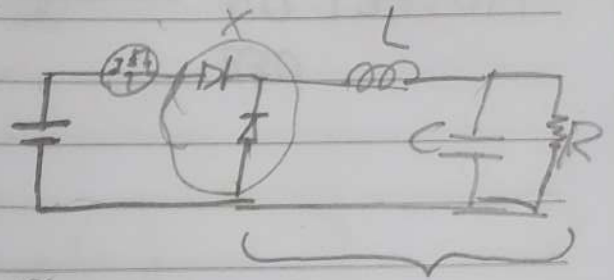
$$= V_{out} \approx 11,998 \text{ V}$$



## Conversor Buck CC-CC

↳  $V_{out} = 12V - 6A$

↳  $V_{in} = 36V - 2A$



↳ DUTY CYCLE - Parâmetro  $D$

↳  $33,333\% \Rightarrow \frac{1}{3}$

Resposta em  
Frequência  
(Diagrama de  
Bode)

↳ Indutor

$$L = \frac{(V_{in} - V_{out}) \cdot D}{\Delta I_L \cdot f} \Rightarrow L = \frac{(36 - 12) \cdot \frac{1}{3}}{(0,4 \cdot 2) \cdot 50 \cdot 10^3} = 220 \mu H$$

↳  $I_{out}$   
(descarga)

↳ CAPACITOR

$$C = \frac{\Delta I_L}{8 \cdot f \cdot \Delta V_{out}} \Rightarrow C = \frac{0,8}{8 \cdot 50 \cdot 10^3 \cdot 0,12} \approx 16,2 \mu F$$

↳ Caracter oscilatório da tensão no indutor  
se de ao chaveamento do MOSFET