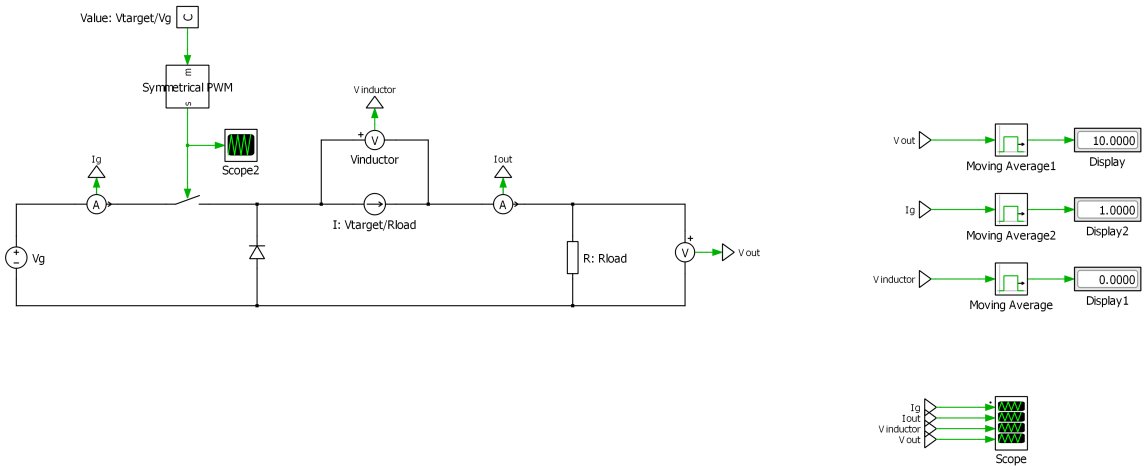
In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from UliEngineering.EngineerIO import format_value
%matplotlib
%matplotlib inline
```

```
Using matplotlib backend: TkAgg
```

Lab 1

Ian Eykamp



The ideal current source will always supply a constant V_{target}/R_{load} through the resistor (yielding a constant V_{target} voltage drop), no matter what the orientation of the switch or even the whole left hand side of the circuit. Interestingly, this is achieved through an alternating positive and negative voltage difference across the current source, depending on the state of the switch. When the switch is closed (connected), there is a $V_g - V_{target}$ voltage drop in the positive direction, dissipating (or storing) power from the voltage source V_g to lower the voltage for the load resistor. When the switch is open (not connected), the current source induces an increase in voltage from ground, drawing current through the diode; in this case, it is generating power. This alternating power source/sink behavior is reminiscent of an inductor.

Helper Functions

In [ ]:
```
# Helper functions

# helper function for formatting axis units
def axes_units(x_unit: str, y_unit: str, ax = None):
```

```
    def format_x(value, pos=None):
        return format_value(value, x_unit)
    def format_y(value, pos=None):
        return format_value(value, y_unit)

    # Set our formatters as X and Y axis formatters
    if ax:
        ax.xaxis.set_major_formatter(mtick.FuncFormatter(format_x))
        ax.yaxis.set_major_formatter(mtick.FuncFormatter(format_y))
    else:
        plt.gca().xaxis.set_major_formatter(mtick.FuncFormatter(format_x))
        plt.gca().yaxis.set_major_formatter(mtick.FuncFormatter(format_y))

# one-liner for simplifying axes code
def axes_labels(x_label: str, x_unit: str, y_label: str, y_unit: str, title = None,
    if ax:
        ax.set_xlabel(x_label)
        ax.set_ylabel(y_label)
        axes_units(x_unit, y_unit, ax)
        if title:
            ax.set_title(title)
    else:
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        axes_units(x_unit, y_unit)
        if title:
            plt.title(title)

# helper function for simplifying column names
def strip_labels(label: str):
    if label == "Time":
        return "t"
    idx = label.find(":")
    if idx < 0:
        return label
    return label[:idx]
```

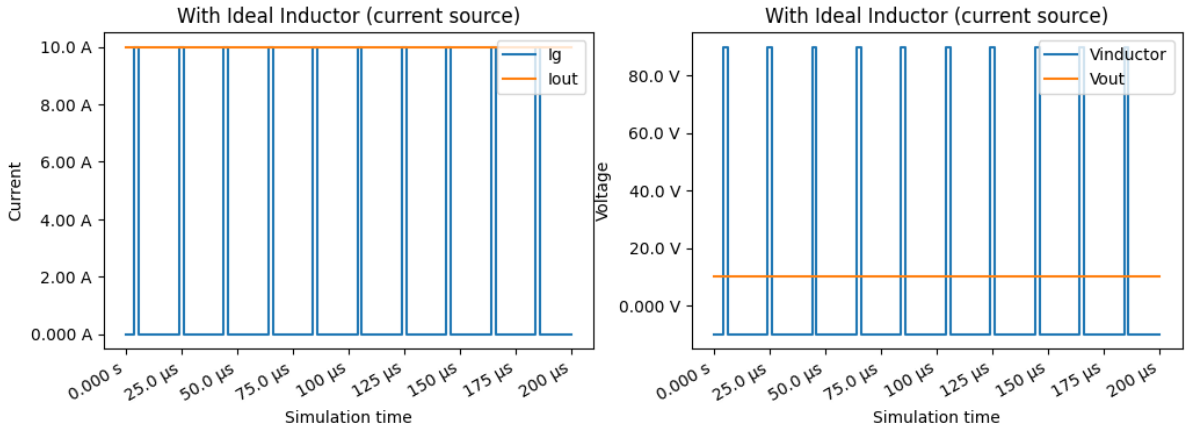
Analysis

In [ ]:

```
df = pd.read_csv("current_source_inductor.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df[df["t"] < 0.2e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()

# ax1.figure()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
axes_labels("Simulation time", "s", "Current", "A", title = "With Ideal Inductor (c
ax1.legend(loc = "upper right")

# plt.figure()
ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
```

```
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With Ideal Inductor (c
ax2.legend(loc = "upper right")
```

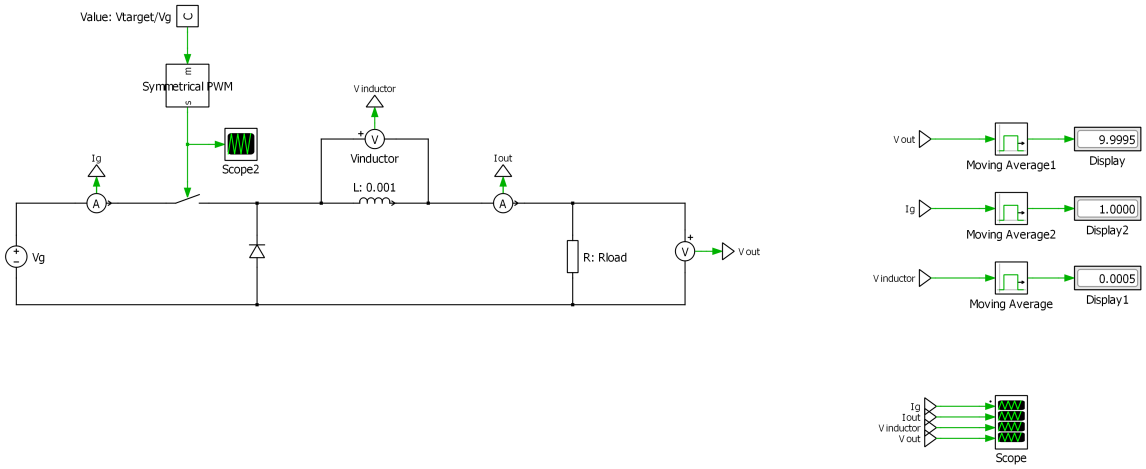
Out[ ]: <matplotlib.legend.Legend at 0x23e379c6e20>



	Peak	Average	Ripple
Ig	90% at 0A, 10% at 10A	1A	Large
Iout	10A	10A	Small
Vinductor	90% at-10V to 10% at 90V	0V	Large
Vout	10V	10V	Small

The voltage source `Vg` supplies power, and the load resistor dissipates power. The ideal inductor both stores and supplies power, with a net power of 0W. The diode dissipates power because current only flows when the voltage drop across it is positive, although it is an ideal diode in this case, and so its power dissipation is negligible.

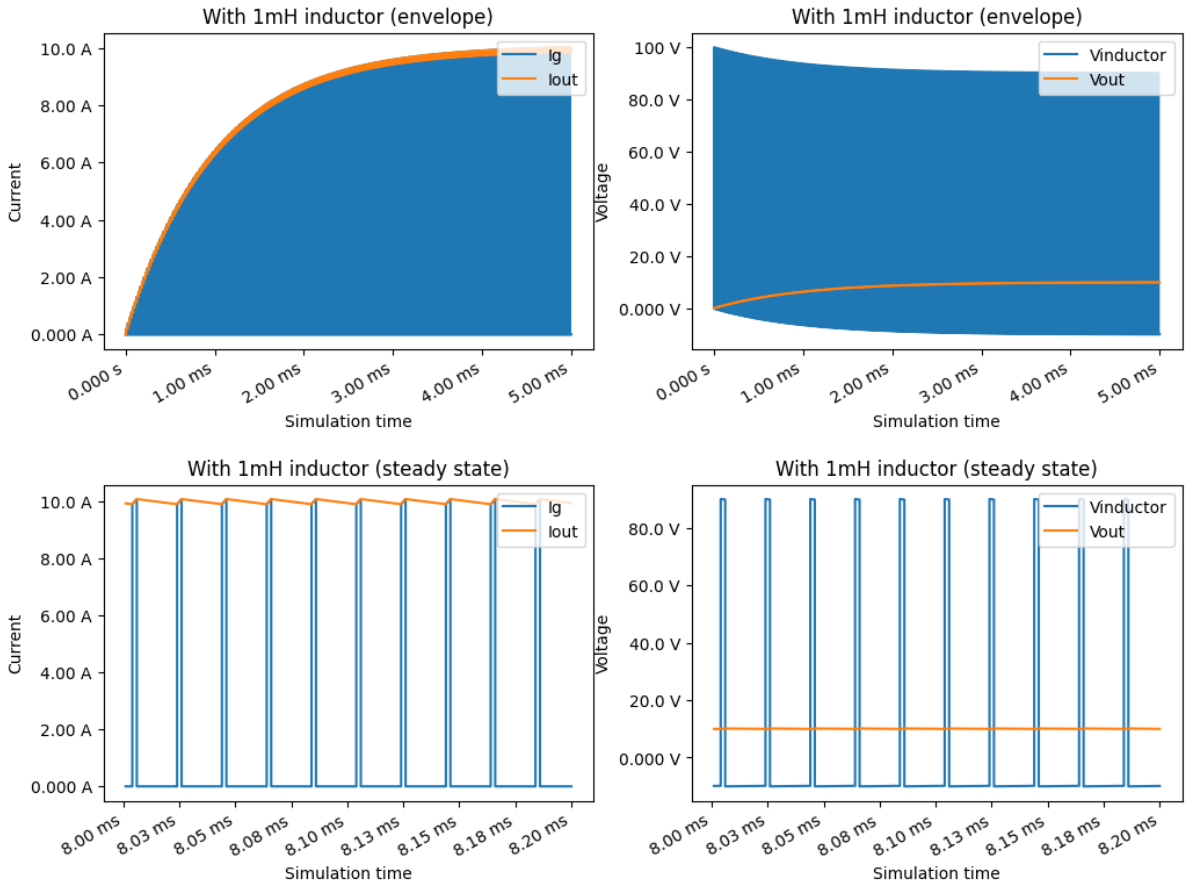
1mH Inductor



In [ ]:
```
df = pd.read_csv("1mH_inductor.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
```

```
df_envelope = df.loc[df["t"] < 5e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
axes_labels("Simulation time", "s", "Current", "A", title = "With 1mH inductor (env
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 1mH inductor (env
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
axes_labels("Simulation time", "s", "Current", "A", title = "With 1mH inductor (ste
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 1mH inductor (ste
ax2.legend(loc = "upper right")
```

Out[ ]: <matplotlib.legend.Legend at 0x23e37847760>



Upon startup, the current ramps up asymptotically towards the target current, reaching steady state after about 5ms. At steady state, the source current `Ig` still alternates between 0A and close to 10A with a 10% duty cycle, but the output current has a small amount of ripple due to the instantaneous discharging of the inductor. The output voltage is not noticeably different from before, although I would expect `Vout` to have the same ripple as `Iout`.

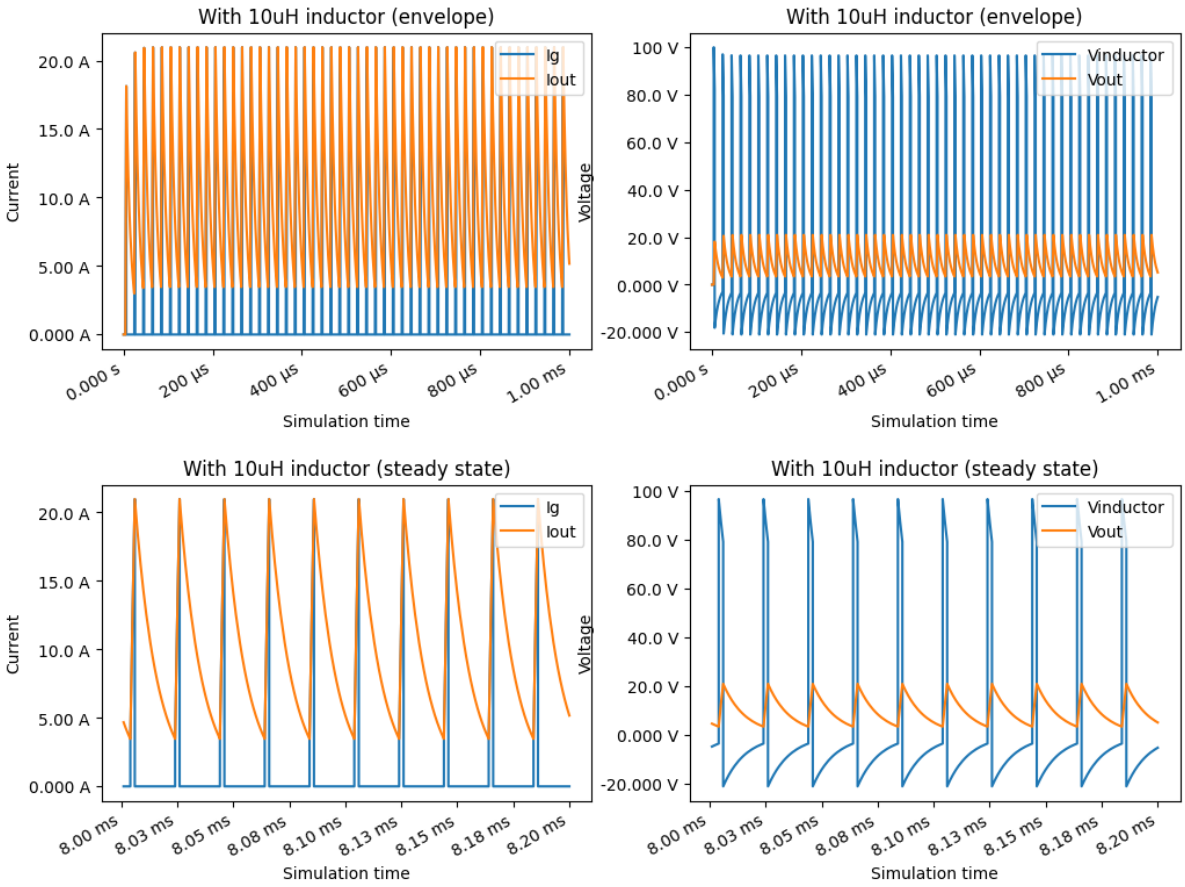
During steady state, the average voltage across the inductor is very close to 0V, although it is not identically zero due to its small amount of hysteresis. In fact it is 0.005V.

The voltage conversion ratio is nearly lossless for duty cycles of 10% and 50%, although the hysteresis-induced average voltage drop across the inductor does increase from 0.005V to 0.0023V.

10uH Inductor

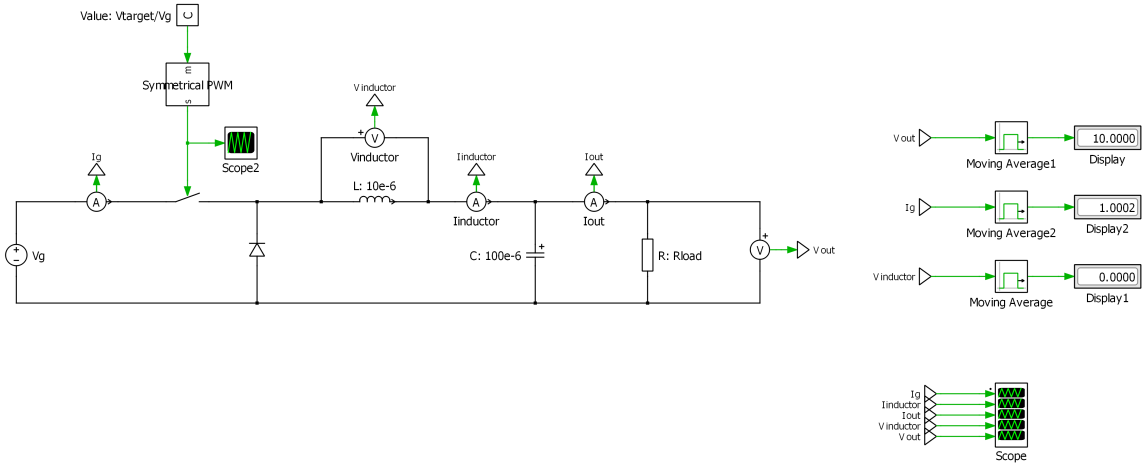
In [ ]:
```
df = pd.read_csv("1uH_inductor.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 1e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (en
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (en
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (st
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (st
ax2.legend(loc = "upper right")
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x23e371c6910>
```



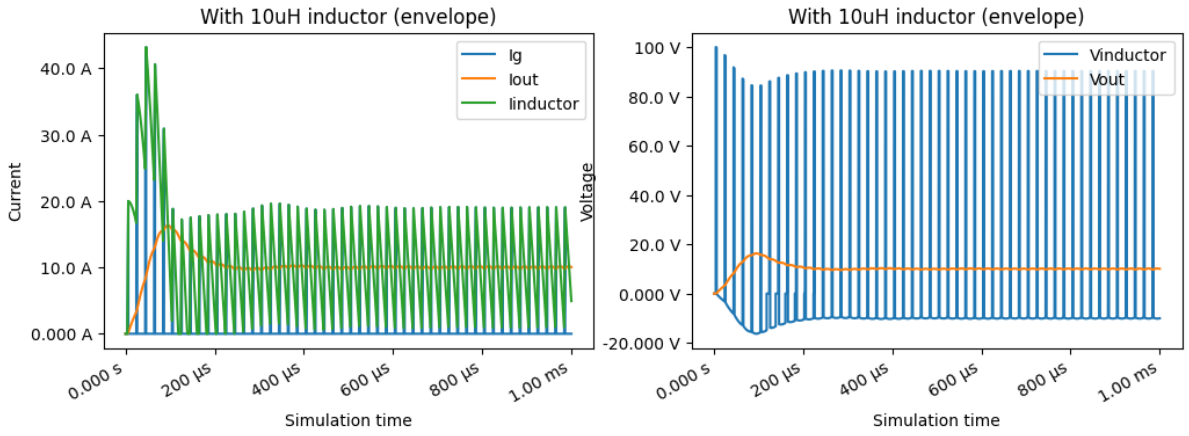
The inductor doesn't have enough capacity to maintain approximately steady state, so the ripple is greatly amplified as the inductor discharges and recharges between 5A and nearly 20A. The output voltage too flucuates between 5V and 20V, in order to maintain an average of 10V across the load. The inductor's behavior resembles a low-pass filter with too high a cutoff frequency, such that it rounds off the most extreme spikes in the input current signal but does not significantly dampen the underlying fluctuation.

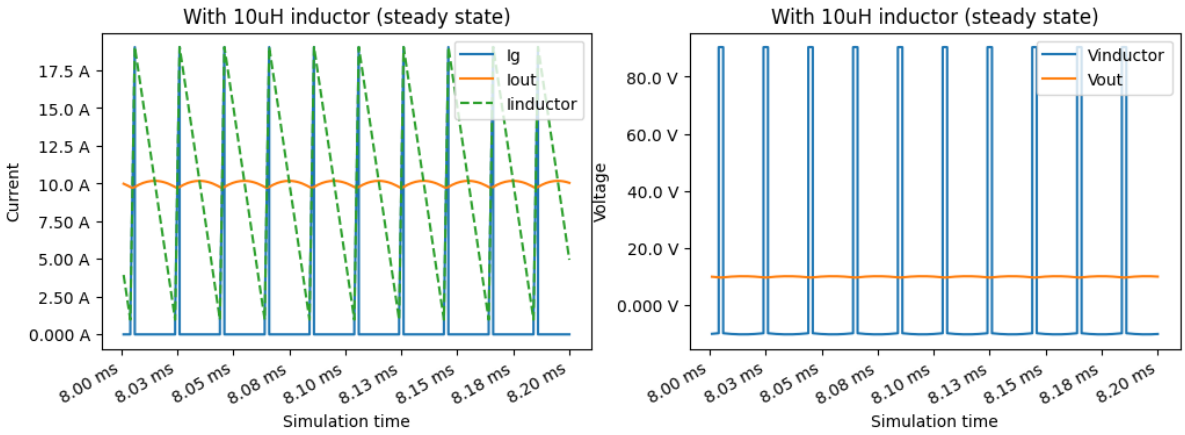
10uH Inductor, 100uF Capacitor



In [ ]:
```
df = pd.read_csv("1uH_inductor_100uF_capacitor.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 1e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
ax1.plot(df_envelope["t"], df_envelope["Iinductor"], label = "Iinductor")
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (en
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (en
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
ax1.plot(df_zoom["t"], df_zoom["Iinductor"], label = "Iinductor", linestyle = "dash
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (st
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (st
ax2.legend(loc = "upper right")
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x23e32dcf3d0>
```

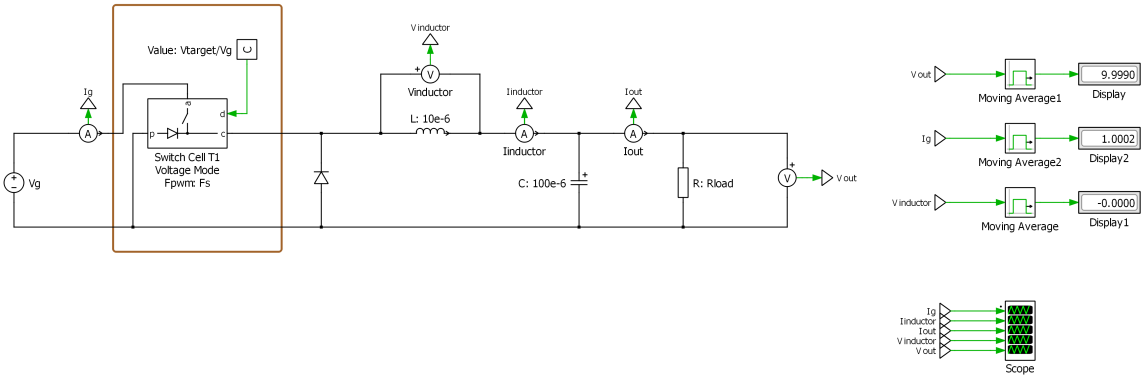




The addition of the capacitor forms a complete LC low-pass filter. The capacitor converts excess current from the inductor during the charging stage while the switch is closed into a small rise in the load voltage. During the discharging stage when the switch is open, the capacitor is able to source additional current in exchange for a slight decrease in the load voltage. This means that the load voltage becomes decoupled from the inductor current and so can remain more stable. The resulting output current and voltage are relatively constant at the desired value, with small, neatly rounded ripple, compared with the ripple with just the inductor.

The voltage conversion ratio is perfect for $D = \frac{V_{target}}{V_g} = 10\%$ and 50% and is proportional to the duty cycle.

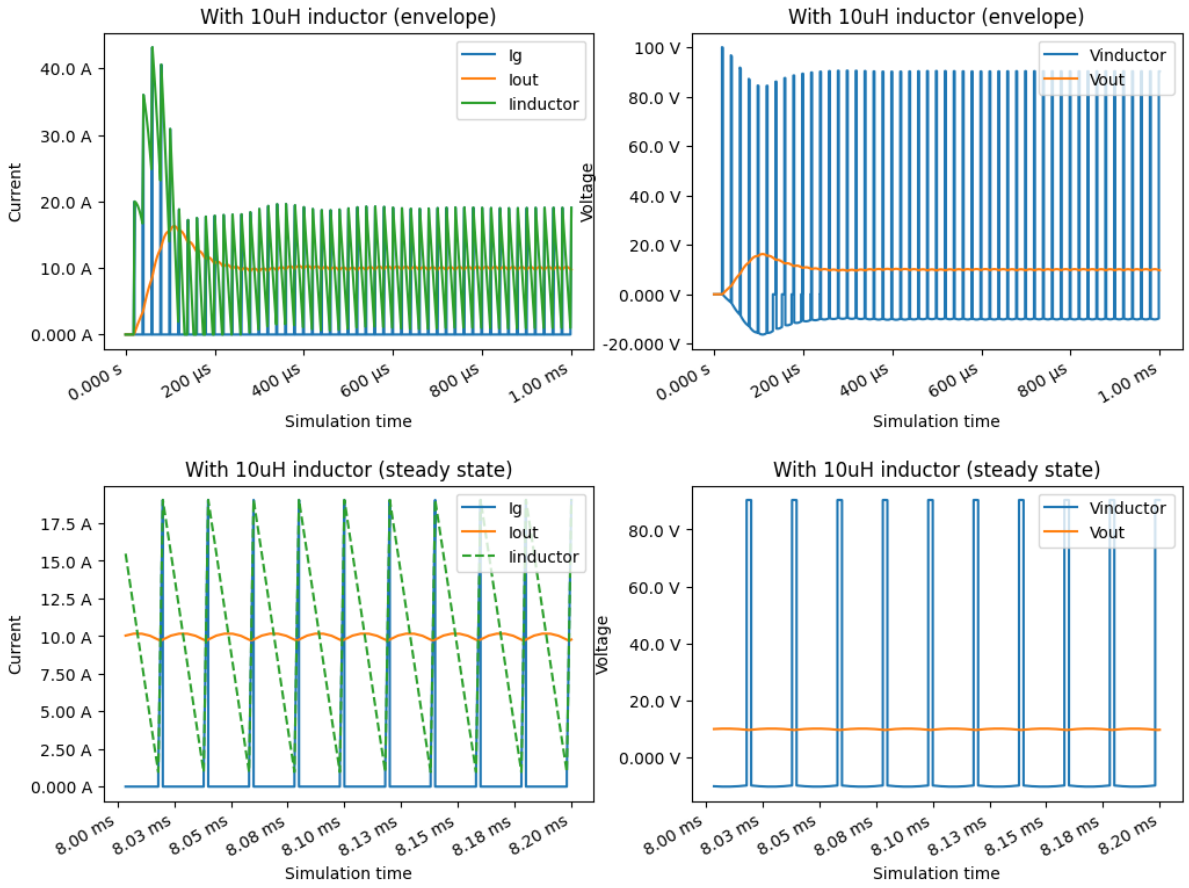
Type 1 Switch Cell



In [ ]:
```
df = pd.read_csv("type_1_switch_cell.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 1e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
ax1.plot(df_envelope["t"], df_envelope["Iinductor"], label = "Iinductor")
```

```
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (en
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (en
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
ax1.plot(df_zoom["t"], df_zoom["Iinductor"], label = "Iinductor", linestyle = "dash
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (st
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (st
ax2.legend(loc = "upper right")
```

Out[ ]: `<matplotlib.legend.Legend at 0x23e37569fd0>`



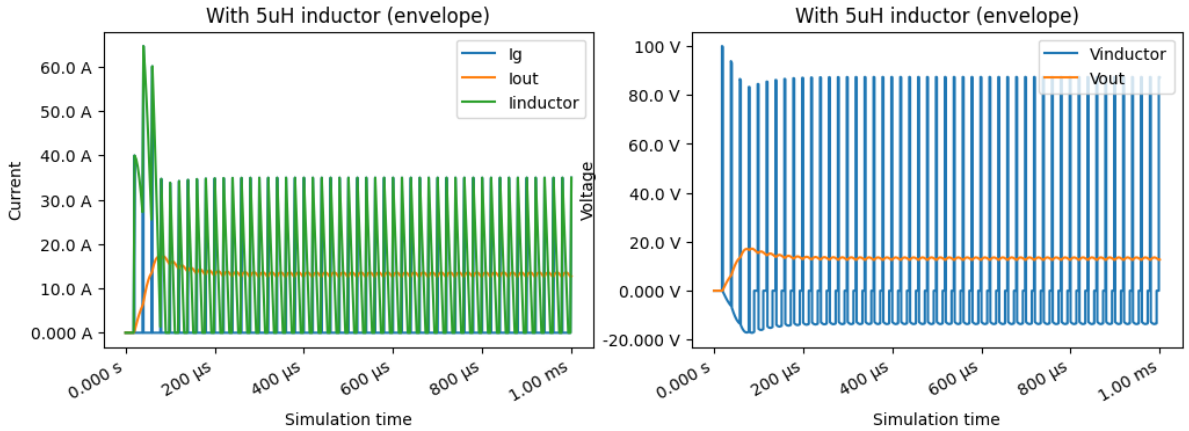
The type 1 switch cell is almost identical to an ideal switch with a symmetric PWM control.

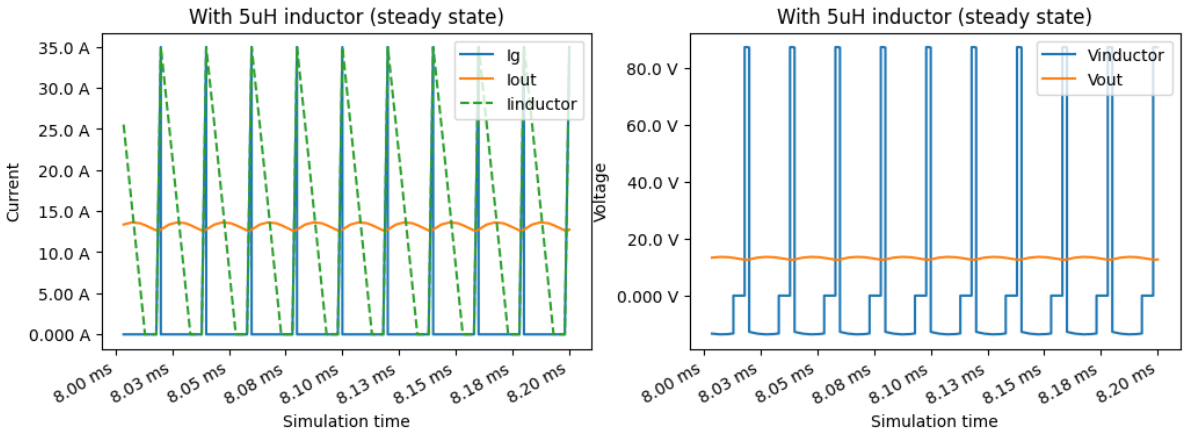
I propose the highlighted block should include the inductor and capacitor, as these form an integral part of the transformer-like behavior. In any case, the right hand side of the circuit can be approximated as an ideal transformer, because at steady state, the output and

current and voltage neatly follow the transformer relationship, $V_{in} \cdot I_{in} = V_{out} \cdot I_{out}$, where the ratio of V_{out} to V_{in} is the duty cycle of the switch cell, analogous to the ratio of windings in a transformer.

In [ ]:
```
df = pd.read_csv("type_1_switch_cell_5uH.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 1e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
ax1.plot(df_envelope["t"], df_envelope["Iinductor"], label = "Iinductor")
axes_labels("Simulation time", "s", "Current", "A", title = "With 5uH inductor (env
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 5uH inductor (env
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
ax1.plot(df_zoom["t"], df_zoom["Iinductor"], label = "Iinductor", linestyle = "dash
axes_labels("Simulation time", "s", "Current", "A", title = "With 5uH inductor (ste
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 5uH inductor (ste
ax2.legend(loc = "upper right")
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x23e37c53e20>
```





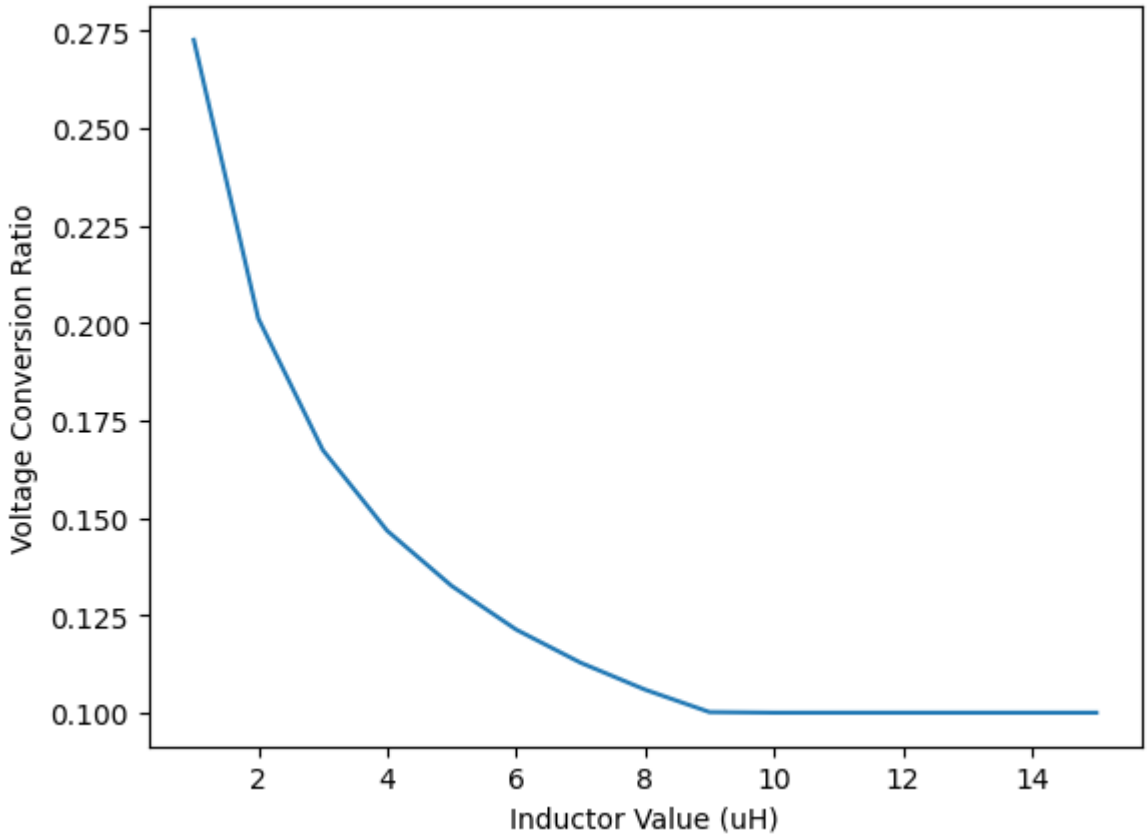
For the 5uH case, the inductor runs out of charge before the cycle finishes. This results in an output voltage that is higher than the target 10V. (This can be remedied by increasing the switching frequency or by increasing the inductor value). This causes the inductor voltage to stay at 0V for a certain fraction of the period. Curiously, due to the LC low-pass filter, this does not significantly affect the average output current and voltage.

Type 2 Switch Cell

In [ ]:
```
L = np.arange(15, 0, -1)
print(L)
Vout = np.array([9.9990, 9.9990, 9.9990, 9.9990, 9.9990, 9.9990, 10.0134, 10.5895,
plt.figure()
plt.plot(L, Vout / 100)
plt.xlabel("Inductor Value (uH)")
plt.ylabel("Voltage Conversion Ratio")
```

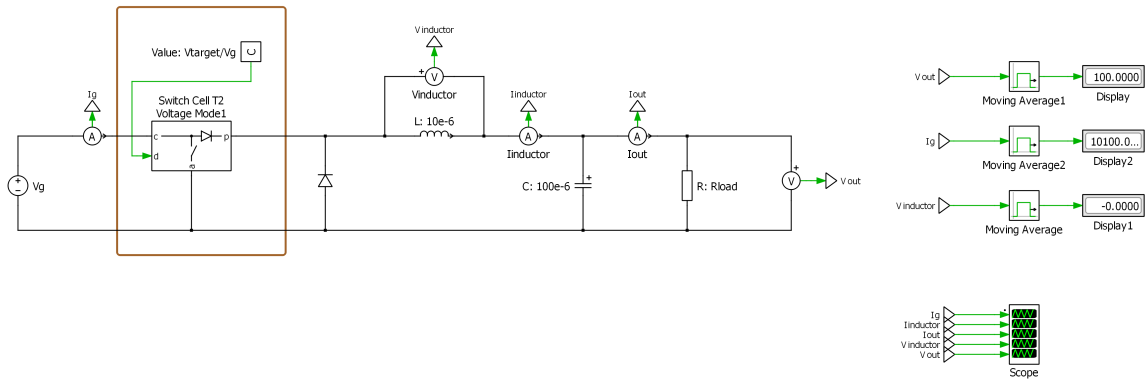
```
[15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
```

Out[ ]:
```
Text(0, 0.5, 'Voltage Conversion Ratio')
```



The voltage conversion ratio becomes less accurate at small inductor values, because the buck converter cannot fully reduce the input voltage to the desired level.

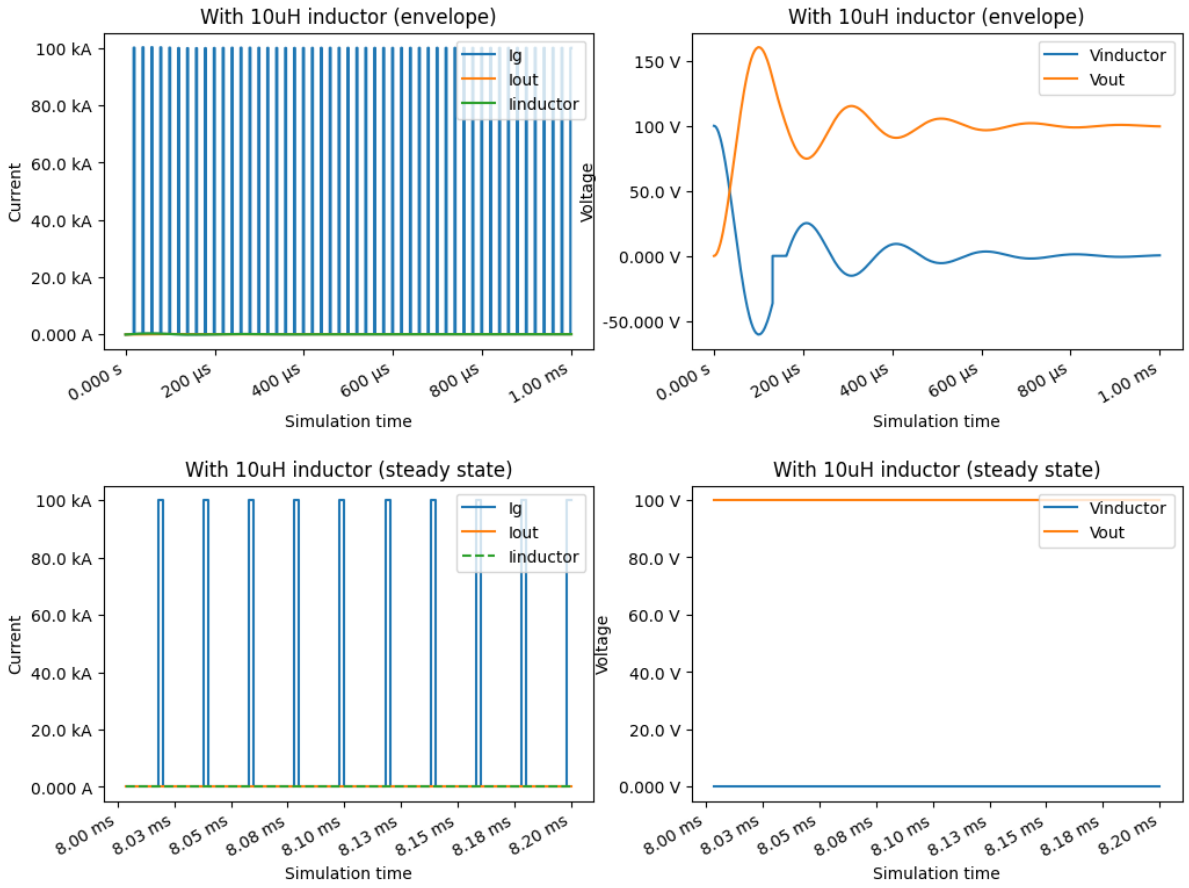
Type 2 Switch Cell



In [ ]:
```
df = pd.read_csv("type_2_switch_cell.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 1e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
ax1.plot(df_envelope["t"], df_envelope["Iinductor"], label = "Iinductor")
```

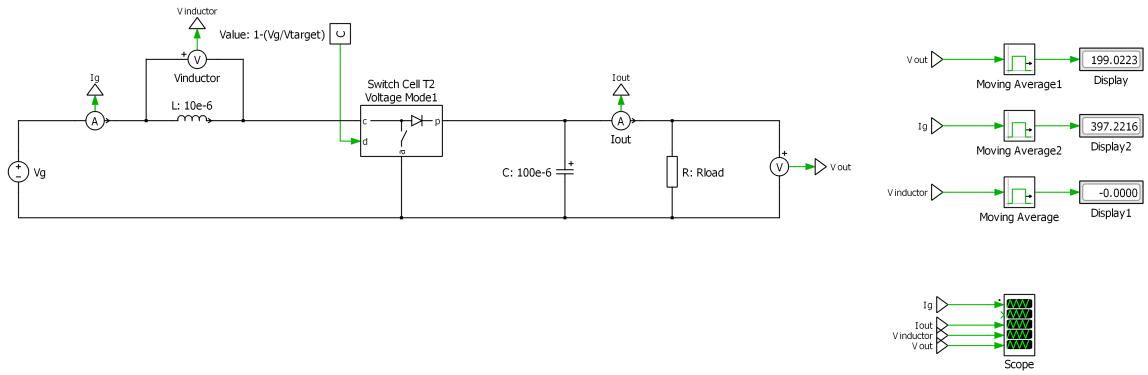
```
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (en
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (en
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
ax1.plot(df_zoom["t"], df_zoom["Iinductor"], label = "Iinductor", linestyle = "dash
axes_labels("Simulation time", "s", "Current", "A", title = "With 10uH inductor (st
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "With 10uH inductor (st
ax2.legend(loc = "upper right")
```

Out[ ]: `<matplotlib.legend.Legend at 0x23e37e47ca0>`



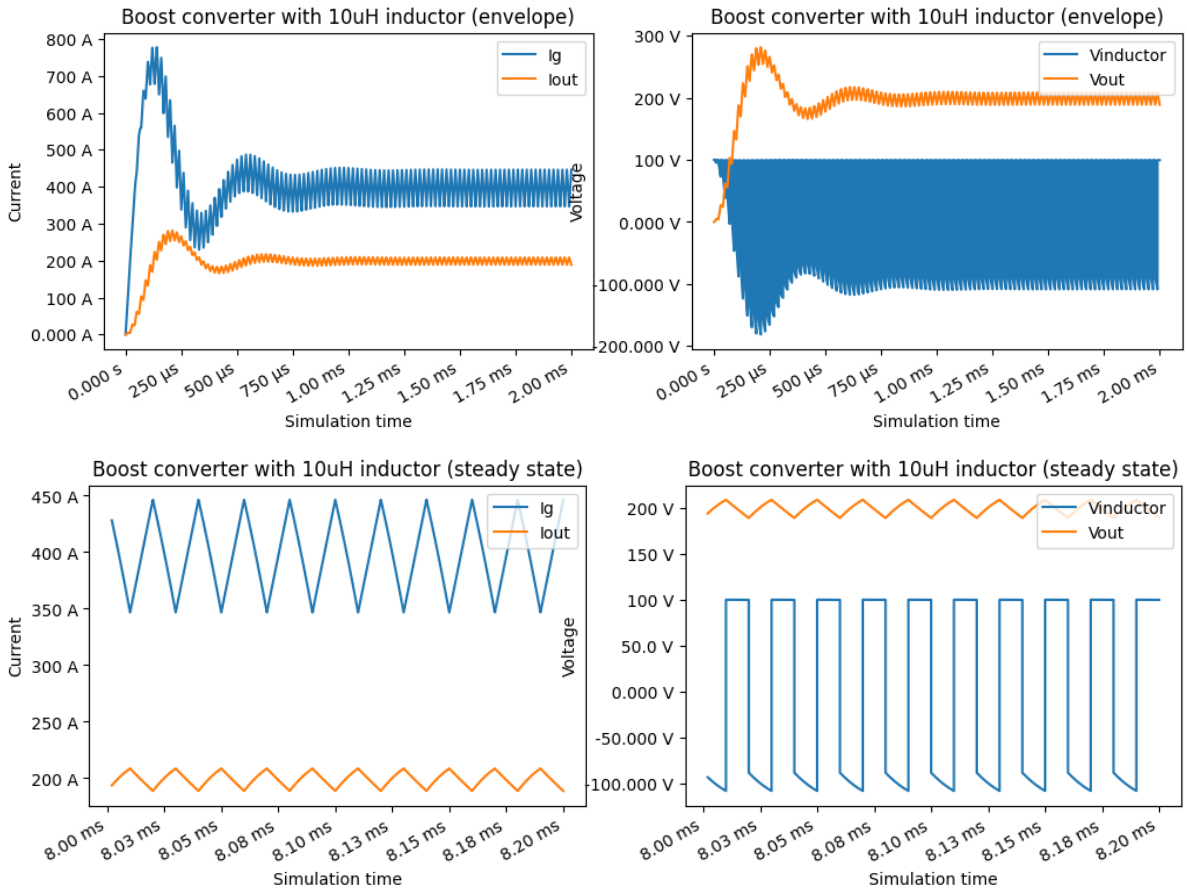
The Type 2 switching cell allows the output voltage to reach an average of 100V using brief periodic bursts of 100kA of input current.

Boost Converter



In [ ]:
```
df = pd.read_csv("boost.csv")
df.rename(mapper = strip_labels, axis = "columns", inplace = True)
df_zoom = df.loc[(df["t"] > 8e-3) & (df["t"] < 8.2e-3)]
df_envelope = df.loc[df["t"] < 2e-3]

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_envelope["t"], df_envelope["Ig"], label = "Ig")
ax1.plot(df_envelope["t"], df_envelope["Iout"], label = "Iout")
# ax1.plot(df_envelope["t"], df_envelope["Iinductor"], label = "Iinductor")
axes_labels("Simulation time", "s", "Current", "A", title = "Boost converter with 1
ax1.legend(loc = "upper right")

ax2.plot(df_envelope["t"], df_envelope["Vinductor"], label = "Vinductor")
ax2.plot(df_envelope["t"], df_envelope["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "Boost converter with 1
ax2.legend(loc = "upper right")

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, sharex = True, sharey = False,
fig.autofmt_xdate()
ax1.plot(df_zoom["t"], df_zoom["Ig"], label = "Ig")
ax1.plot(df_zoom["t"], df_zoom["Iout"], label = "Iout")
# ax1.plot(df_zoom["t"], df_zoom["Iinductor"], label = "Iinductor", linestyle = "da
axes_labels("Simulation time", "s", "Current", "A", title = "Boost converter with 1
ax1.legend(loc = "upper right")

ax2.plot(df_zoom["t"], df_zoom["Vinductor"], label = "Vinductor")
ax2.plot(df_zoom["t"], df_zoom["Vout"], label = "Vout")
axes_labels("Simulation time", "s", "Voltage", "V", title = "Boost converter with 1
ax2.legend(loc = "upper right")
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x23e3a02cb50>
```



`D = 50%` for this example.

Contrary to the buck converter, the boost converter increases the average output voltage using a moderate duty cycle and a signficant input current to charge and discharge the inductor. In this case, the target output voltage is 200V, inducing 200A across the 1Ohm load. This requires an average input current of 400A at the input voltage of 100V. The inductor voltage for the boost converter is driven to + and - 100V, the full range of the input voltage, again averaging 0V. Because of the configuration of the circuit, the inductor current is equal to the input current, `Ig`.

The input current transitions linearly in the range of 350A to 450A, following the nearly perfect square wave for the inductor voltage. (Remember, $V_{inductor} = L\frac{dI}{dt} = const$) at the top of the square wave). The output voltage and current also follow a sawtooth pattern due to weak filter damping.

One final note is that the envelope takes longer to reach steady state than for the buck converter analyzed earlier. In the above graphs, the time axis for the envelope spans 2ms, compared with 1ms for the buck converter graphs.

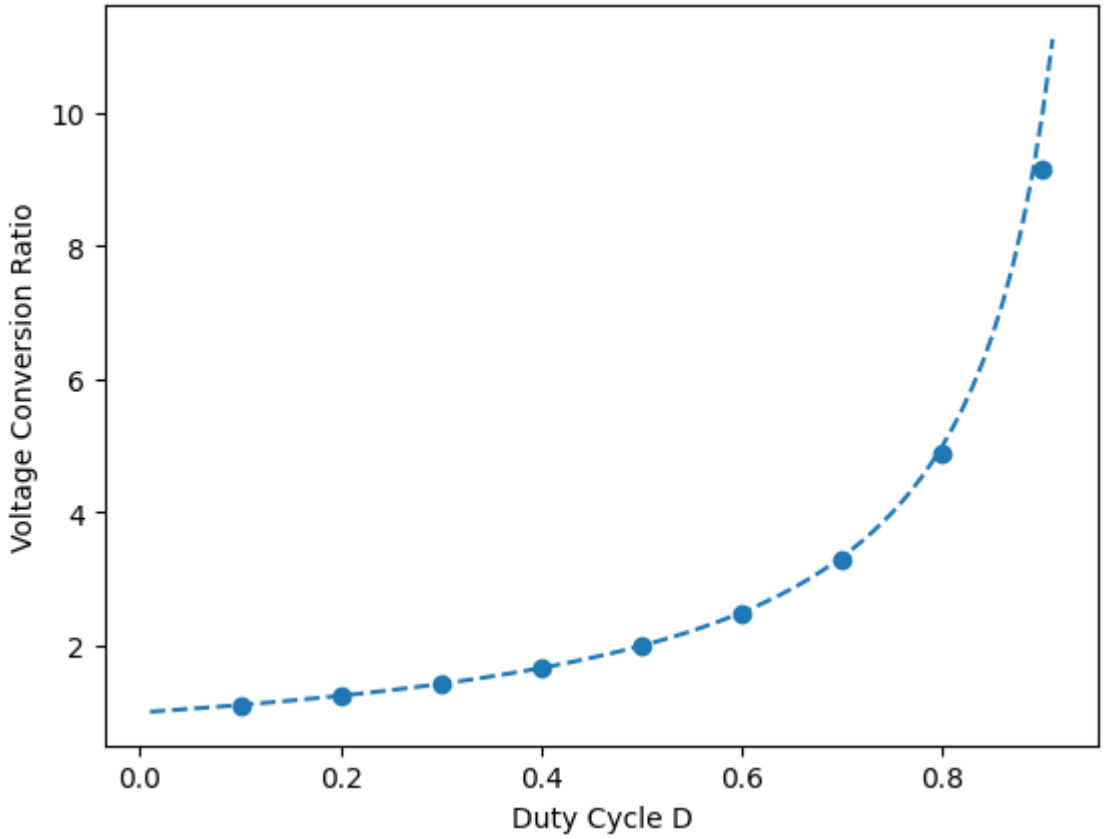
For `D = 50%`, the voltage conversion ratio is $1.99 \approx 2$.

In [ ]:
```
D = np.arange(0.1, 1, 0.1)
print(D)
Vout = np.array([111.0636, 124.8375, 142.5173, 166.0747, 199.0223, 248.2968, 329.75
```

```
D_eq = np.arange(0.01, 0.92, 0.01)
Vout_eq = 1 / (1 - D_eq)

plt.figure()
plt.plot(D_eq, Vout_eq, linestyle = "dashed")
plt.scatter(D, Vout / 100)
plt.xlabel("Duty Cycle D")
plt.ylabel("Voltage Conversion Ratio")
```

```
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

Out[ ]: `Text(0, 0.5, 'Voltage Conversion Ratio')`



The voltage conversion ratio matches nicely with the equation $\text{Voltage conversion ratio} = \frac{V_{out}}{V_{in}} = \frac{1}{(1-D)}$ (dashed line). To determine the appropriate duty cycle to use to reach a desired output voltage with a boost converter, set $D = 1 - \frac{V_{in}}{V_{out}}$. In reality, the voltage conversion is not perfect for high gains; when D = 0.9, for example, the output voltage is only 915V, compared with the expected 1000V. Due to the non-ideality of the components, the maximum voltage gain that can be achieved in the simulation is around 1500V at around `D = 95%`.