Pre-Lab 7 PLECS Model Calculations

Ian Eykamp

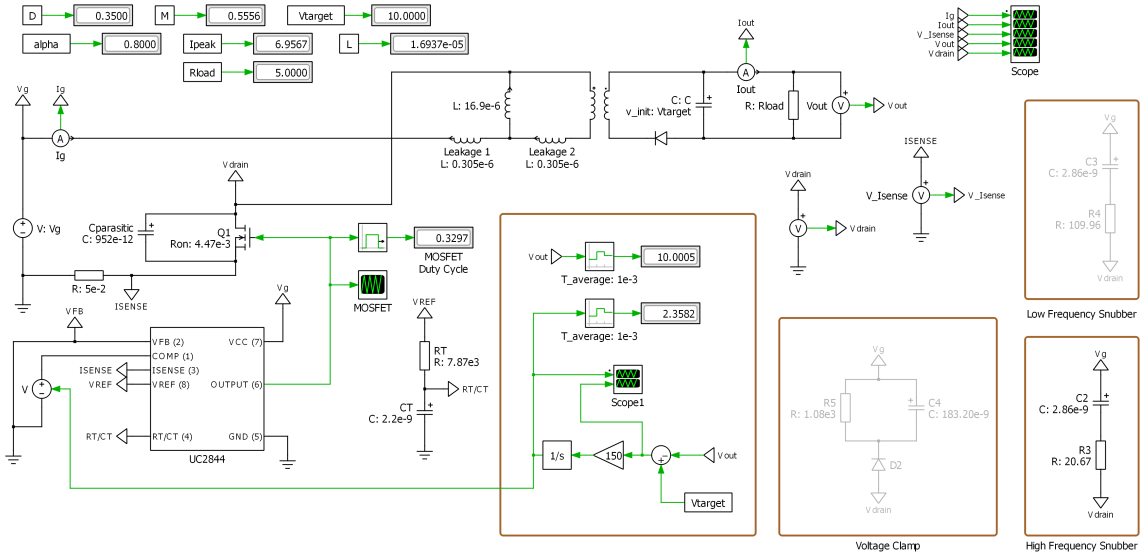
I was late for this assignment, so I calculated my snubber and clamp values separately from the rest of my group. My snubber and clamp values are printed at the bottom of this document.

I calculated the parasitic capacitance using the equation $\omega_0 = \frac{1}{\sqrt{L_{ring} C_{parasitic}}}$. The inductance is very different for the high-frequency ringing than for the low-freqency ringing due to the topology of the circuit. In the high-frequency case, L_{ring} is very nearly $L_{leakage}$, whereas in the low-frequency case, it is $L_{magnetizing} + \frac{L_{leakage}}{2}$ (assuming $L_{\sigma 1} = L_{\sigma 2}$). Solving for $C_{parasitic}$ using the respective inductances gave results that differed by a factor of four. This makes me suspicious of the leakage inductance values I measured, since I expected the parasitic capacitance to be similar in all topologies. I trust my inductance value for the low-frequency case better since it is dominated by the magnetizing inductance which is well known.

Another point of confusion is the treatment of radians in ω_0. I am not sure in the above equation if ω_0 should be measured in radians per second or in revolutions per second. This distinction is of great importance for the parasitic capacitance and snubber design. Choosing radians per second yields more reasonable values for the parasitic inductance (~1nF instead of 10pF) and snubber values more in line with my teammates' calculations.

PLECS Scematic

The final snubber values are also displayed in this PLECS schematic.



Import Libraries

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
from si_prefix import si_format
import plecs_helper as helper
%matplotlib
%matplotlib inline

# Imports and setup
from pint import UnitRegistry
from scipy.signal import find_peaks
from scipy.optimize import fsolve

# pandas display using scientific notation
# pd.set_option('display.float_format', lambda x: f'{x:.3e}')

# use pint
units = UnitRegistry()
units.default_format = "~P.2f"
```

```
Using matplotlib backend: TkAgg
```

Base Values

In [ ]:
```
Vg = 18 * units.volt
Vout = 10 * units.volt
M = Vout / Vg
Rload = 5 * units.ohm
Pout = 20 * units.watt
Iout = Pout / Vout
D = 0.35
alpha = 0.80
a = Vg / Vout * D / (np.sqrt(alpha) - D)
fs = 50 * units.kilohertz
Ts = 1 / fs
Lcrit = 1 / (M + 1) ** 2 * Rload * Ts / 2
L = alpha * Lcrit
Ipeak = 2 / np.sqrt(alpha) * (M + 1) * Vout / Rload

print(f"Winding ratio a = N1 / N2: {a}")
print(f"Inductor value L: {L.to('microhenry')}")
print(f"Peak current Ipeak: {Ipeak.to('amp')}")
```

```
Winding ratio a = N1 / N2: 1.16
Inductor value L: 16.53 µH
Peak current Ipeak: 6.96 A
```

Set Up Flyback Converter Calculations

In [ ]:

```
class FlybackParameters():
    def __init__(self, Lm, Ll, wd, time_constant, clamp_voltage, units_included = T
        self.Lm = Lm
        self.Ll = Ll
        self.wd = wd
        self.time_constant = time_constant
        self.Vclamp = clamp_voltage
        self.name = name
        if not units_included:
            self.Lm = self.Lm * units.Henry
            self.Ll = self.Ll * units.Henry
            self.time_constant = self.time_constant * units.second
            self.wd = self.wd * 2 * np.pi * units.radian / units.second
        self.calculate_parasitics()
        self.calculate_snubber()
        self.calculate_clamp()

    def __repr__(self) -> str:
        return f"[{self.name}] Flyback Converter with Lm = {self.Lm}, w0 = {self.w0

    def print_parasitics(self) -> None:
        print(f"Parasitic values for [{self.name}] Converter: L = {self.Ll.to_compa

    def print_snubber_values(self) -> None:
        print(f"Snubber values for [{self.name}] Converter: Csnubber = {self.Csnubb

    def print_clamp_values(self) -> None:
        print(f"Clamp values for [{self.name}] Converter: Cclamp = {self.Cclamp.to_

    def calculate_parasitics(self):
        self.beta = 1
        self.w0 = self.wd / self.beta
        # self.Cparasitic = 1 / self.Ll / (self.w0 / (2 * np.pi * units.radian)) **
        self.Cparasitic = 1 / self.Ll / (self.w0 / units.radian) ** 2
        self.alpha = np.abs(1 / self.time_constant)
        self.Rparasitic = 2 * self.alpha * self.Ll

    def calculate_snubber(self):
        self.Csnubber = 3 * self.Cparasitic
        self.zeta_target = 1 / np.sqrt(2)
        self.Rsnubber = self.zeta_target * 2 / np.sqrt(self.Csnubber / self.Ll)

    def calculate_clamp(self):
        self.Kcp = self.Vclamp / Vout
        self.energy_in_inductor = 1 / 2 * Ipeak ** 2 * self.Ll
        self.Wclamp = self.energy_in_inductor * self.Kcp / (self.Kcp - 1)
        self.Pclamp = self.Wclamp / Ts
        self.Rclamp = self.Vclamp ** 2 / self.Pclamp / 1.5
        self.delta_out = 0.1
        self.Cclamp = 1 / (self.Rclamp * self.delta_out) * (Ts - Ipeak * self.Ll /
```

Import Experimental Values and Print Calculations

In [ ]:

```
Lm = 16.96 * units.microhenry
Ll = 0.61 * units.microhenry

high_freq_wd = 87.3 * units.megaradian / units.second
low_freq_wd =  7.80 * units.megaradian / units.second
high_freq_tau = -273 * units.nanosecond
low_freq_tau = -5.64 * units.microsecond

clamp_voltage = 40 * units.volt

high_freq_parameters = FlybackParameters(Lm = Lm, Ll = Ll,          wd = high_freq_
low_freq_parameters =  FlybackParameters(Lm = Lm, Ll = Lm + Ll / 2, wd = low_freq_w

print(high_freq_parameters)
print(low_freq_parameters)
print()

print("Before adjustment:")
high_freq_parameters.print_parasitics()
low_freq_parameters.print_parasitics()
print()

# This calculation yields different values for the parasitic capacitance
# Set the parasitic capacitances to be equal
# I trust the low-freuency ringing more because it uses the magnetizing inductance,
high_freq_parameters.Cparasitic = low_freq_parameters.Cparasitic
# Recalculate snubber and clamp values given the new value for parasitic capacitanc
high_freq_parameters.calculate_snubber()
high_freq_parameters.calculate_clamp()

print("After adjustment:")
high_freq_parameters.print_parasitics()
low_freq_parameters.print_parasitics()

high_freq_parameters.print_snubber_values()
low_freq_parameters.print_snubber_values()
print()

high_freq_parameters.print_clamp_values()
print("Power Consumed by Clamp:", high_freq_parameters.Pclamp.to_compact(units.watt
```

[High Frequency] Flyback Converter with Lm = 16.96 µH, ω0 = 87.30 Mrad/s, τ = -27
3.00 ns
[Low Frequency] Flyback Converter with Lm = 16.96 µH, ω0 = 7.80 Mrad/s, τ = -5.64
µs

Before adjustment:
Parasitic values for [High Frequency] Converter: L = 610.00 nH, C = 215.10 pF, R =
4.47 Ω
Parasitic values for [Low Frequency] Converter: L = 17.27 µH, C = 952.02 pF, R =
6.12 Ω

After adjustment:
Parasitic values for [High Frequency] Converter: L = 610.00 nH, C = 952.02 pF, R =
4.47 Ω
Parasitic values for [Low Frequency] Converter: L = 17.27 µH, C = 952.02 pF, R =
6.12 Ω
Snubber values for [High Frequency] Converter: Csnubber = 2.86 nF, Rsnubber = 20.6
7 Ω
Snubber values for [Low Frequency] Converter: Csnubber = 2.86 nF, Rsnubber = 109.9
6 Ω

Clamp values for [High Frequency] Converter: Cclamp = 183.20 nF, Rclamp = 1.08 kΩ
Power Consumed by Clamp: 984.03 mW