5-2015

Semi-Automated Switching Regulator Modeling Method and Tool

Michael Heath Leonard
University of Arkansas, Fayetteville

Semi-Automated Switching Regulator Modeling Method and Tool

Semi-Automated Switching Regulator Modeling Method and Tool

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

by

Michael Leonard
University of Arkansas
Bachelor of Science in Electrical Engineering, 2013
Bachelor of Physics, 2013

May 2015
University of Arkansas

This thesis is approved for recommendation to the Graduate Council

Dr. H. Alan Mantooth
Thesis Director

Tom Vrotsos Committee Member	Dr. A. Matt Francis Committee Member

ABSTRACT

This thesis presents the results of research targeted at automating the behavioral modeling process for switching voltage regulators. These regulators are commonly used in many application areas including discrete use in larger systems, integrated in a System on a Chip (SoC), or as the primary use case for a design. When used in an integrated system these regulators can be a significant force in slowing down simulations. A common method for removing this slowdown is to use a behavioral model of the switching regulator. Creating behavioral models can be very time consuming and requires expertise.

The thesis discussion begins by developing a fundamental understanding of switching regulators, introduces common modeling methods used for switching regulators, and justifies the selection of the PWM switch modeling method. After discussing the fundamentals, the various methods of model generation and optimization are described and an examination of the software structure and development process is undertaken. The thesis concludes with a results presentation comparing automatically generated models with real-world measurement data.

ACKNOWLEDGEMENTS

I would like to first thank those who made this possible. Dr. Alan Mantooth for providing the opportunity to work project, facilities within which to work, guidance, and funding. Dr. Matt Francis for keeping the project on track, providing guidance and reviewing early versions of this work. Tom Vrotsos for his excellent mentorship and assistance throughout this project, as well as for his efforts in reviewing several pieces of my work. I also would like to thank the Semiconductor Research Corporation for project funding and support, Texas Instruments for directed funds and mentorship, and Freescale Semiconductor for mentorship and interest in usage.

Finally I would like to thank the people who make everything possible, my parents. First my Mom, for keeping me sane on the many late nights I have endured over the past six years and always providing encouragement and for caring for me more than any other person in the world. My Dad for his selfless efforts at helping me succeed in whatever I committed to. From the early years of T-Ball and Soccer, to the late nights of Lego League, to helping me with scholarships and then trusting me to handle my business, I can say with certainty that I wouldn’t be where I am today without their love. For that, I am forever grateful.

TABLE OF CONTENTS

List of Tables

List of Figures

NOMENCLATURE

Duty Cycle	Ratio of time a signal is high to the period of the signal.
Line Regulation	Ability of a system to maintain a constant output in relation to changes on the input.
Load Regulation	Ability of a system to maintain a constant output in relation to load variation.
Compensation Network	The combination of passive and active components used in a switching regulator controller to shape and extend its frequency response capability.
GUI	Graphical user interface.
Pareto Frontier	The set of parameterized data that are Pareto efficient.
Pareto Efficient	As applied to this work Pareto efficient is the state that exists when a parameterized dataset is optimal in at least one dimension and moving to a dataset that is less optimal in that dimension results in an improvement in the other dimension.
OTA	Operational transconductance amplifier
EVM	Evaluation module. A standard reference design that is sent to potential customers to evaluate part performance.
Test-driven development	Abbreviated as TDD. Test-driven development is a development philosophy that puts testing and code verification first. This allows the developer to confidently write code and stop as soon as all the tests pass, resulting in less time wasted developing unnecessarily. It also allows changes to be made in the code and know that the changes aren't causing breakage.
Functional programming	A programming paradigm in which the immutability of data is considered paramount. The paradigm itself is based off of the mathematics of lambda calculus and lends itself nicely to following a TDD philosophy. See "Test-driven development"
Object-oriented programming	A programming paradigm in which data structures are "objects" that contain *data* which is acted on by *methods*. This is often the first learned programming paradigm due to how intuitively it can be analogized to objects in the real-world.
ESR	Equivalent series resistance, a parasitic resistance associated with a capacitor or inductor
ESL	Equivalent series inductance, a parasitic inductance associated with a capacitor
Prototype	A common programming paradigm in which the parent class defines functions but leaves them unimplemented and requires any subclasses to implement the functions.
PWM	Pulse-width modulation

1. INTRODUCTION

The work presented within demonstrates methods and software tools for automatically generating behavioral models of switching regulators given a set of AC and transient data that adequately characterizes the device.

These models are useful on two fronts. On one side they can save engineers massive amounts of time in the design phase by reducing simulations times by a factor of more than 10000. These models can also be used to provide potential customers with accurate and fast models of devices that they may be interested in purchasing.

Automation is achieved through use of the iterative and analytical power of modern computers as well as the simplifying assumption that the most important characteristics of a typical switching regulator will be dominated by the selection of the compensation network. The validity of this approach and the assumptions made is described in the following sections.

1.1 Background of Switching Regulators

Voltage regulation has been required in some form or another since the dawn of electronics. One example of a common requirement for a voltage regulator is to take an input voltage and *step it down* to a lower voltage, then maintain that voltage even if the output load changes.

As an illustrative example consider a computer power supply. In the United States this might take a 120 volt AC supply voltage from the wall and rectify it to a steady 24 volts DC, AC to DC is a form of voltage regulation. After rectification there remains significant regulation that needs to take place. Modern computers require several different voltage rails such as ±12V, ±5V, and ±3.3V. The multiple voltage rail requirement implies that the power supply must be able to step down and invert the input to several different levels. Furthermore the voltage should remain nearly constant and independent of the output load. This requirement and how it can be achieved will be discribed further in following sections.

The computer power supply is only one example of voltage regulation. In practice nearly every electronic system requires some sort of regulation which makes this class of circuit very popular, thus ensuring that the field of power supply design remains an active field in electrical engineering.

This thesis does not attempt to provide a full reference on power supply design and modeling. Rather, the intent is to provide enough background information to properly motivate the research and ensure the reader maintains full comprehension throughout the document.

1.1.1 Linear Regulators

The simplest class of voltage regulator is the *linear regulator*. These regulators typically have the following characteristics:

- small footprint at low power, few to no external components required, often sold in a small package as well
- cheap, ~$0.10-$0.80 fully integrated as compared to ~$1-$2 plus components for switching regulators
- low noise, $\leq 10\ \mu V_{rms}$
- can have fixed or variable output voltage
- may have integrated protection circuitry

The absolute most basic "regulator" would be a simple voltage divider circuit, though this circuit provides little in the way of regulation. The theory of operation is illustrated through a brief example and analysis.

Voltage Divider

The most general circuit for a voltage divider is shown in Fig. 1.

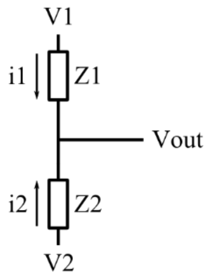


Fig. 1. Schematic of generic voltage divider.

In the ideal case where there is no output current the circuit can be described by the following three equations:

$$i_{Z1} + i_{Z2} = 0 \tag{1}$$

$$\frac{V_1 - V_{out}}{Z_1} = i_{Z1} \tag{2}$$

$$\frac{V_2 - V_{out}}{Z_2} = i_{Z2} \tag{3}$$

Substituting Eq. (2) and Eq. (3) into Eq. (1) gives

$$\frac{V_1 - V_{out}}{Z_1} + \frac{V_2 - V_{out}}{Z_2} = 0 \tag{4}$$

and solving this equation for V_{out} yields

$$V_{out} = \frac{V_1 Z_2 + V_2 Z_1}{Z_1 + Z_2} \tag{5}$$

or the more common form where $V_2 = 0$ gives

$$V_{out} = \frac{Z_2}{Z_1 + Z_2} V_1. \tag{6}$$

This equation is useful for developing an intuitive understanding of the circuit but in reality systems rarely operate under *no load* conditions. If the circuit shown in Fig. 1 were attached to another system then Eq. (1) would become

$$i_{Z1} + i_{Z2} = i_{out}. \tag{7}$$

The result of this change is the introduction of an i_{out} term in the system equation such as

$$V_{out} = \frac{V_1 Z_2 + V_2 Z_1 - Z_1 Z_2 i_{out}}{Z_1 + Z_2} \tag{8}$$

indicating that the output current has quite a large effect on the output voltage proportional to $Z_1 Z_2$! This effect is visualized in Fig. 2.

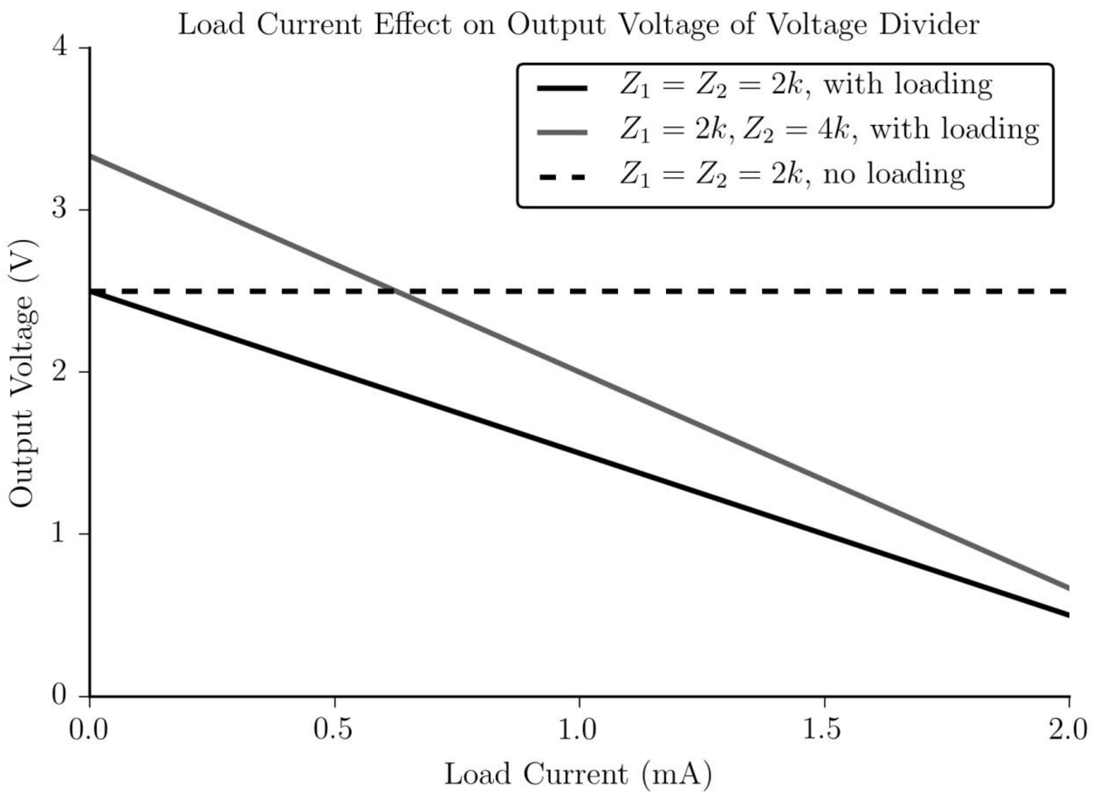


Fig. 2. Comparison of voltage divider with loading effects and without.

In Fig. 2 V_1=5V and V_2=0. The x-axis is an output load (i_{out}) sweep from 0 to 2 mA while the y-axis indicates the output voltage V_{out}. Solid lines indicate that the model considers output loading, while dashed lines show no response to the load change.

It is clear from this figure that any change in output load has a dramatic effect on the output voltage. Voltage dividers can easily be designed to provide a certain voltage at a certain load, but any deviation of load current will quickly cause the design to fail.

Simple Series Regulator

A more robust linear regulator design is the simple series regulator utilizing a bipolar transistor and a Zener diode. This design includes a bipolar transistor in an *emitter follower* configuration as well as a Zener diode attached to the transistor's base. An example of such a design is shown below in Fig. 3.

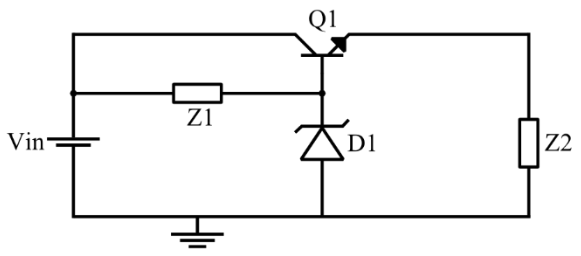


Fig. 3. A simple series regulator using a bipolar transistor and Zener diode.

In this circuit it is clear that the voltage across the load (Z_2) is being completely generated by the emitter current of Q1, therefore

$$V_{out} = i_E Z_2 \tag{9}$$

The behavior of this circuit changes non-linearly as the transistor moves from cutoff, through the active region, and into saturation. The response then changes again when the transistor base begins pulling too much current from the Zener diode and causes the circuit to lose its regulatory ability. The mathematics behind this behavior are more detailed than is required to motivate this thesis. As an alternative an intuition can be developed through visual means.

In Fig. 4 the three operating modes of a bipolar junction transistor (BJT) appear in the response. The first region when the input voltage is below about 1 V is the cutoff region. In this region the transistor is completely off so the output load isn't able to draw any power. Soon after that the transistor enters in to the active region where the output voltage is rising nearly linearly with the input voltage.

The behavior in this region looks quite similar to a BJT entering in to the saturation region but in fact in this simulation as the output approaches 4 V the Zener diode begins to become strongly reverse biased and starts conducting current to ground, causing the observed flattened response. This reverse biasing keeps the base of the transistor near the breakdown voltage of the Zener diode and thus keeps the transistor on and biased at around the same voltage regardless of output conditions.

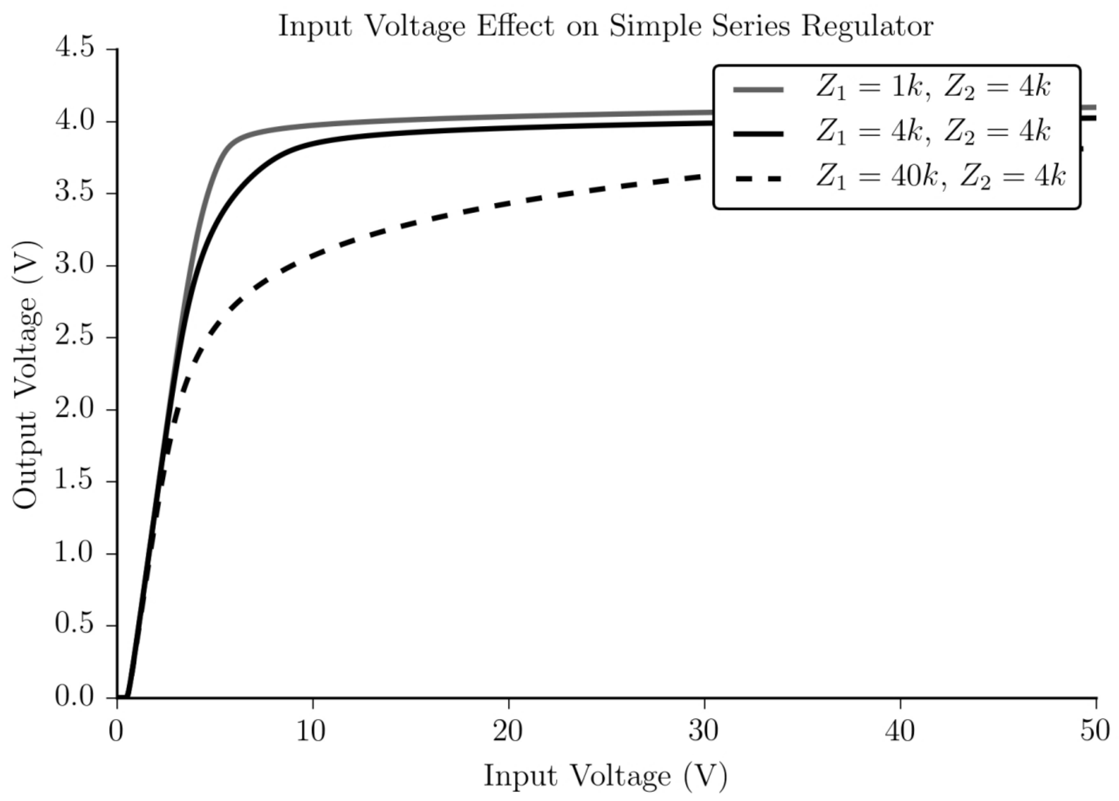


Fig. 4. Three different simulations of the simple series regulator.

Now consider this same circuit under load variation, shown in Fig. 5.

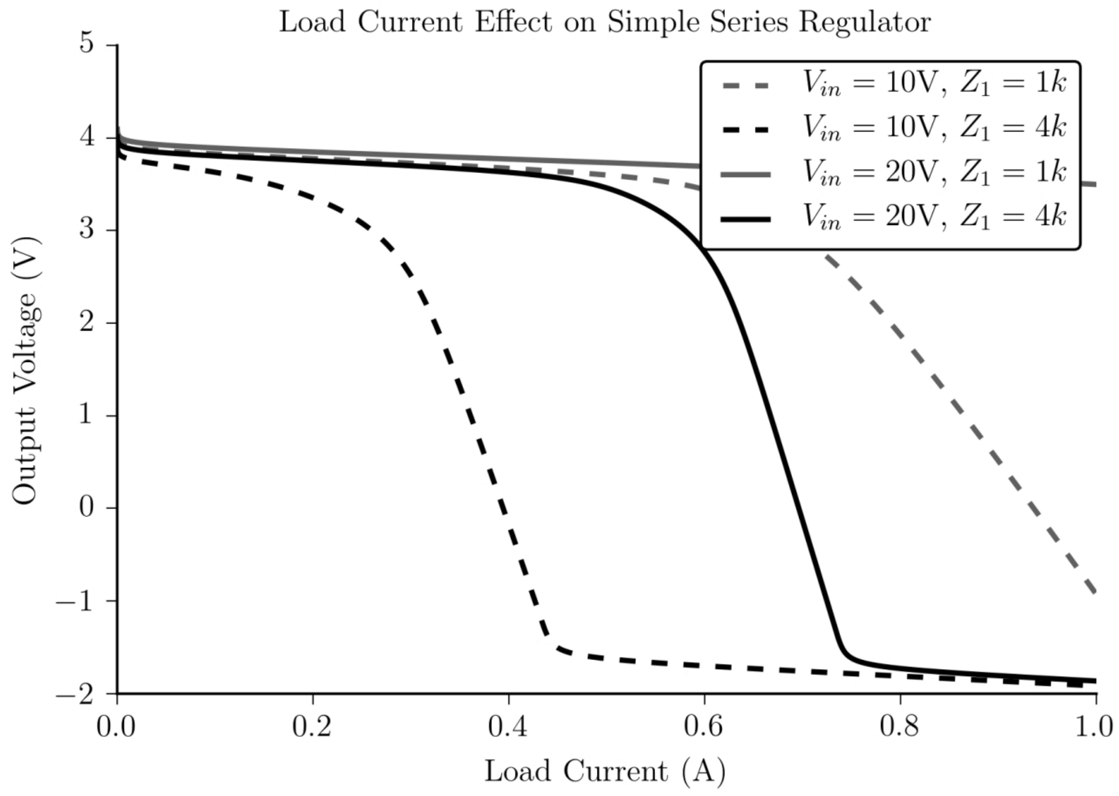


Fig. 5. This shows a comparison of four load current sweeps of the simple series regulator.

The tests in Fig. 5 swept the output current load from 0.1 mA up to 1 A. The other parameters varied include the Z1 value and the input voltage. Comparing this to Fig. 2 there is a noticeable improvement in the voltage regulation capability of the circuit. Changing the load current on the simple voltage divider caused a dramatic shift in the output voltage. With only a 1 mA load increase the output dropped more than a 1 V. In the case of the simple series regulator even the worst design is capable of operating correctly up to 500 mA load or higher.

The regulation capability of linear regulator circuits such as the simple series regulator can be further improved through the addition of feedback control, and functionality can be extended by adding more features such as overcurrent protection or thermal shutdown.

For all of the redeeming qualities of linear regulators they do come with some drawbacks, most notably linear regulators:

- **Are highly inefficient for any significant load** – implying that a significant amount of power is simply wasted as heat.
- **Only perform one function** – linear regulators are only capable of *lowering* the input voltage. If the system requires a negative voltage rail then another solution must be found, likewise if the system requires a higher voltage than is supplied externally.

Fortunately another class of circuit, the *switching regulator*, addresses these shortcomings.

1.1.2 Switching Regulators

Switching regulators assume a much more active role in voltage regulation. As the name implies this class of regulator relies on a switching element, usually a power MOSFET or BJT, to control the voltage at the output terminal.

Basics of Switching Circuitry

The simplest switching regulator circuit would just be a MOSFET with gate connected to a control signal and the drain connected to an output load. An example of such a circuit is shown in Fig. 6.

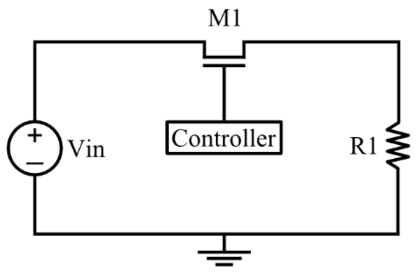


Fig. 6. A simplistic switching regulator circuit.

In this circuit if the MOSFET is conducting then the output node is essentially connected to the input voltage. If the MOSFET is off then the output node remains floating and the output voltage is ideally equal to 0 V.

For the purpose of developing some new concepts we will assume that the *controller* sends a signal to the MOSFET that is ON half of the time and OFF half of the time. This would give a duty cycle of 0.5 since duty cycle can be defined by

$$D = \frac{t_{on}}{T} = f_{sw}t_{on} \tag{10}$$

Where:

- t_{on} is the time in seconds that the input signal is high.
- T is the period of the control signal in seconds.
- f_{sw} is the switching frequency of the control signal.

In the simplest case $T = t_{on} + t_{off}$ but as the analysis becomes more complex an additional term will be introduced. With this knowledge, the circuit from Fig. 6 can be analyzed with varying duty cycle values. The results of this analysis are shown in Fig. 7.

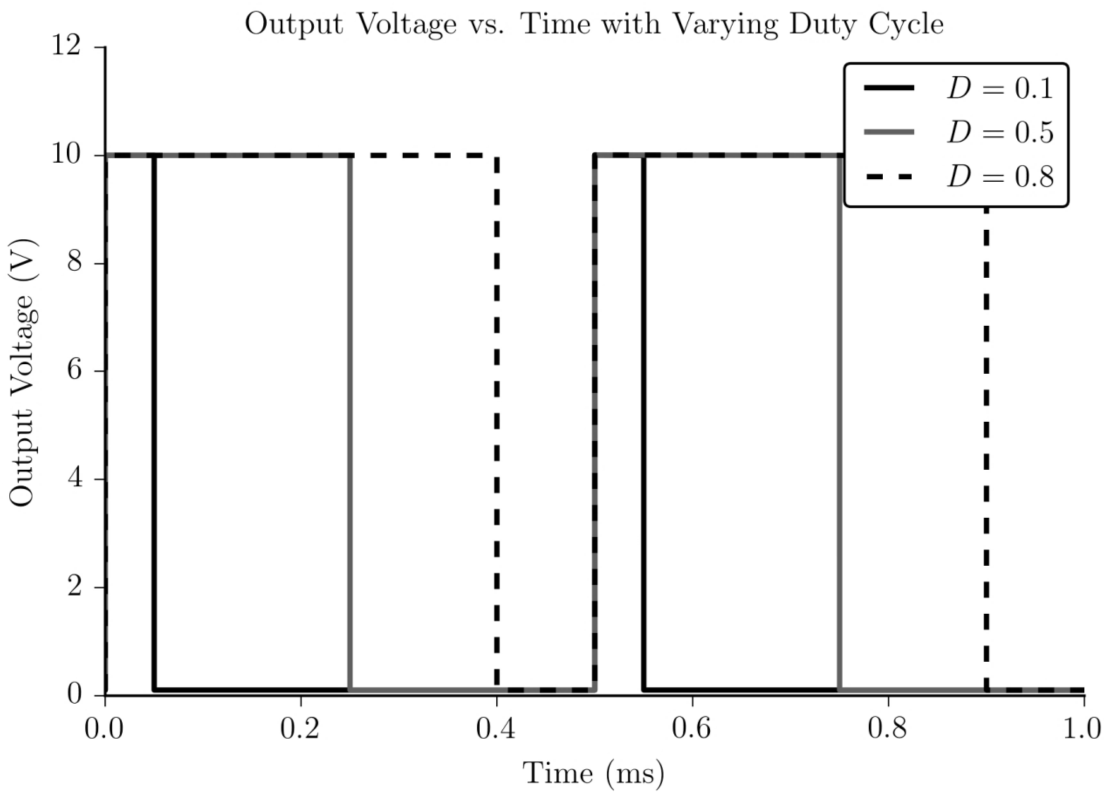


Fig. 7. Simulation results from analysis of the simple switch circuit..

The simulation shown above used an input voltage of 10 V, a load value of 10k and set the duty cycle to 0.1, 0.5, and 0.8 The output of this circuit is noticeably similar to the control input, being essentially the same square wave with a scaled up voltage. A square wave supply voltage is not of much use in the vast majority of electronics systems, but two important concepts can be extracted from Fig. 7.

The first concept is that of averaged voltage. Looking at a single period of the $D = 0.5$ waveform it is clear that over a single period the waveform has an average value of

$$V_{avg} = \frac{V_{in}t_{on} + 0 * t_{off}}{T}. \tag{11}$$

with a little bit of simplification Eq. (11) becomes

$$V_{avg} = V_{in}D \tag{12}$$

implying that the average output voltage is directly proportional to the input voltage by a factor of the duty cycle of the control signal; this duty cycle relation is the second important concept.

Knowing that the average value of the output voltage can be controlled simply by varying the duty cycle of the control signal, it would be useful to actually be able to average this signal so that it could be used to power other systems. This can essentially be accomplished through the addition of an LC output filter.

Adding an Output Filter

The output filter, in this case a series LC circuit, is composed of an inductor and a capacitor. The inductor can be thought of as storage energy to supply current while a capacitor stores charge to maintain voltage.

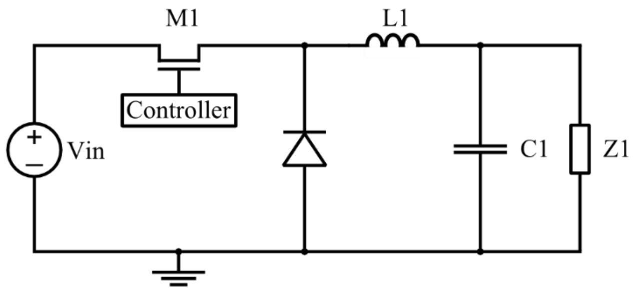


Fig. 8. Simple switch circuit with an LC output filter and blocking diode added. This is the basic circuit for a buck converter.

When the MOSFET is conducting it first must energize the inductor, and then current will flow through the inductor and charge the capacitor as well as provide an output voltage. When the MOSFET opens, the voltage stored on the capacitor discharges to supply the output voltage while the stored current in the inductor's magnetic field flows in to the capacitor to keep it near the same voltage. This can be thought of as two separate modes of operation and the equivalent circuits of these two modes are shown in Fig. 9.

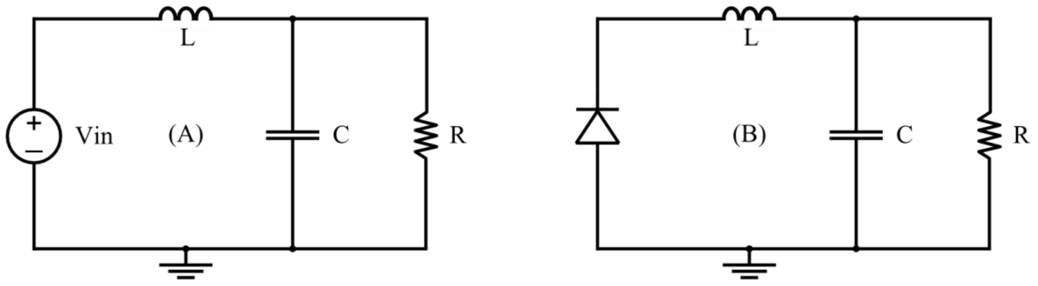


Fig. 9. A schematic comparison of the equivalent circuit formed when the transistor is ON or OFF.

In Fig. 9 above, (A) shows the equivalent circuit of a buck converter when the transistor is on while (B) shows the equivalent circuit when the transistor is off.

Switching Regulator Examples

With the basic switching regulator design concepts developed a more thorough analysis can be performed; for this purpose two types of switching regulators will be analyzed.

Buck Converter

The first example, a buck converter, is the switching regulator equivalent of a linear regulator. Buck converters are only capable of reducing the input voltage, but can do so with very high efficiency and can generally deliver much more current than a typical linear regulator. The circuit for a buck converter is shown above in Fig. 8.

In this topology the output voltage is directly proportional to the input voltage multiplied by the duty cycle. The three most important characteristic equations for the buck converter are

$$V_{out} \leq V_{in} \tag{13}$$

$$i_{in} \leq i_{out} \tag{14}$$

$$V_{out} = V_{in}D \tag{15}$$

Most switching regulators can be thought of as having two modes of operation, *continuous* and *discontinuous* mode. In continuous mode the current through the inductor (I_L) does not rest at 0 A. By contrast in discontinuous mode the current through the inductor may stay at 0 A for an indefinite time during a single switching period. The difference between the two modes is illustrated in Fig. 10.

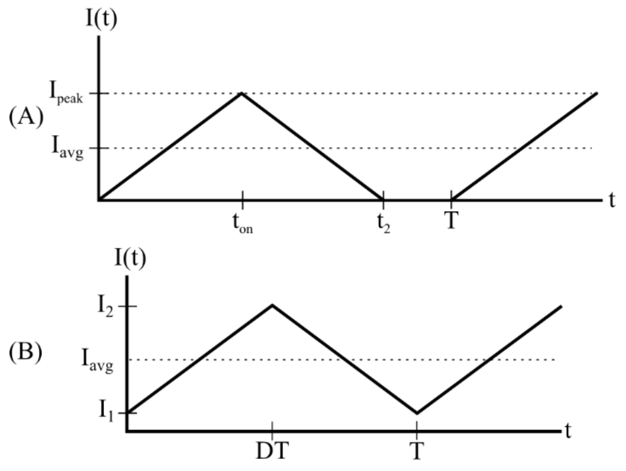


Fig. 10. A comparison of the two modes of operation for switching converters.

In the figure above (A) shows the inductor current for a regulator in the discontinuous mode while (B) shows continuous mode, this figure clearly shows the discontinuity observed in the discontinuous mode from t_2 to T while the continuous waveform is never broken. The significance of this difference is explored in the following two sections.

Continuous Mode

For a buck converter in continuous mode the circuit can be considered to be in one of two states; either the transistor is ON or the transistor is OFF. When the transistor is on the difference in voltage from V_{in} to V_{out} is given by the voltage drop across the inductor

$$V_{in} - V_{out} = V_L = L\frac{di}{dt}. \tag{16}$$

As long as the current through the inductor is continuous then

$$\frac{di}{dt} = \frac{I_2 - I_1}{t_{on}} \tag{17}$$

and substituting Eq. (17) into Eq. (16) and solving for t_{on} gives

$$t_{on} = L\frac{I_2 - I_1}{V_{in} - V_{out}} \tag{18}$$

When the transistor turns off an interesting phenomenon occurs. It is physically impossible to instantaneously change the current through an inductor, however when the voltage source is disconnected from the output then the current in the inductor needs to reverse direction. Rather than breaking the rules the inductor causes an effect known as *inductive kick* [1] which causes the voltage polarity of the inductor to immediately reverse. This implies that at this state transition $V_L = -V_{out}$ and therefore

$$-V_{out} = L\frac{I_1 - I_2}{t_{off}}. \tag{19}$$

Solving Eq. (19) for t_{off} gives

$$t_{off} = L\frac{I_2 - I_1}{V_{out}} \tag{20}$$

Since $I_2 - I_1$ is the same during both the on state and the off state then Eq. (18) and Eq. (20) can be rearranged and set equal giving

$$\frac{V_{in} - V_{out}}{L} t_{on} = \frac{V_{out}}{L} t_{off}. \tag{21}$$

Simplifying Eq. (21) then gives the expected result of

$$(V_{in} - V_{out}) t_{on} = V_{out} t_{off} \tag{22}$$

$$V_{in} t_{on} - V_{out} t_{on} = V_{out} t_{off} \tag{23}$$

$$V_{in} t_{on} = V_{out} t_{on} + V_{out} t_{off} \tag{24}$$

$$V_{in} t_{on} = V_{out} (t_{on} + t_{off}) \tag{25}$$

$$V_{out} = V_{in} \frac{t_{on}}{t_{on} + t_{off}} \tag{26}$$

therefore,

$$V_{out} = V_{in} D. \tag{27}$$

This implies that the buck regulator multiplies the input voltage by the duty cycle of the switching signal, since D is by definition less than one the output voltage will always be less than the input voltage.

Discontinuous Mode

The alternative to a regulator operating in the continuous mode would be operation in the discontinuous mode. This often happens when the regulator load is light, or in other words when the output current is low. The discontinuous mode can most simply be considered the mode of operation in which the inductor current falls to zero and the reason this occurs under light-load conditions can be intuitively explained. As defined below in Eq. 33 the energy stored in a conductor is proportional to the current change through the inductor over a certain period. Thus, during light load conditions a small current will be passing through the inductor and it will not store enough energy to sustain current all the way through the next cycle.

Much of the circuit behavior in the discontinuous mode is very much the same as in the continuous mode with one exception. In continuous mode the circuit can be thought of as operating in one two states:

- State 1 – Transistor ON, inductor current rising
- State 2 – Transistor OFF, inductor current falling

In the discontinuous mode an additional state must be considered where the transistor is off but no current is flowing through the inductor, this state will be defined as:

- State 3 – Transistor OFF, no inductor current

In this third state the inductor has discharged all of the stored magnetic energy and the output voltage is being supplied entirely by the output capacitor. The derivation of the output to input voltage relationship for the discontinuous mode is omitted here for brevity but can be found in [1], the result is given below as

$$V_{out} = \frac{2}{1+\sqrt{1+4L(1-D)/L_C}} V_{in} \tag{28}$$

where the critical inductance L_C is defined as

$$L_C = \frac{R(1-D)}{2f_{sw}}. \tag{29}$$

With the transfer functions of both continuous and discontinuous modes well defined, some further visualizations of these equations and other circuit behavior can be helpful. The first figure, Fig. 11, shows the relationship of the regulator transfer function V_{out}/V_{in} over varying duty cycle with the regulator in various states of inductor current continuity.

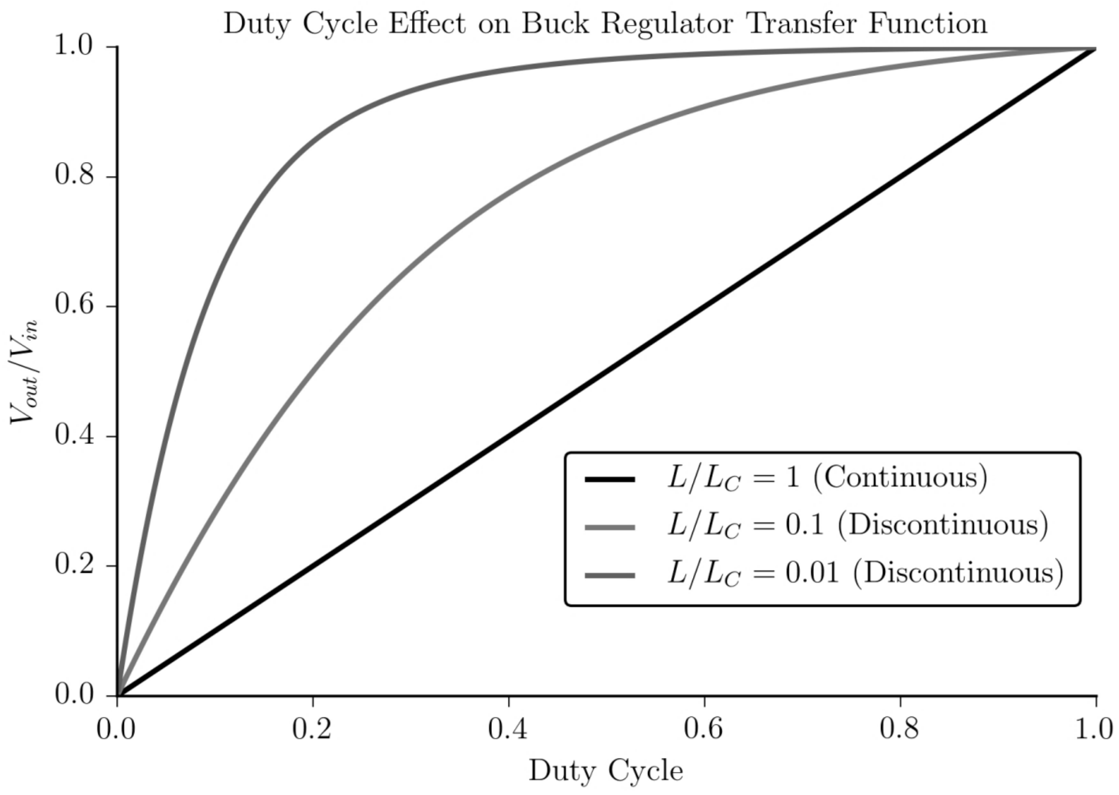


Fig. 11. Buck regulator open loop response. This figure visualizes how the transfer function of the regulator varies as the regulator enters or leaves discontinous mode.

As Fig. 11 shows, when the regulator is in continuous mode the voltage ratio is directly related to the duty cycle, as the regulator inductor current becomes increasingly discontinuous the voltage ratio becomes increasingly eccentric; this indicates that it is highly sensitive to duty cycle changes and thus more difficult to control.

Fig. 12 shows an example of a simple buck regulator operating at three different fixed duty cycle settings. The regulator was supplied with a 10 V input and simulated for 40 ms total.

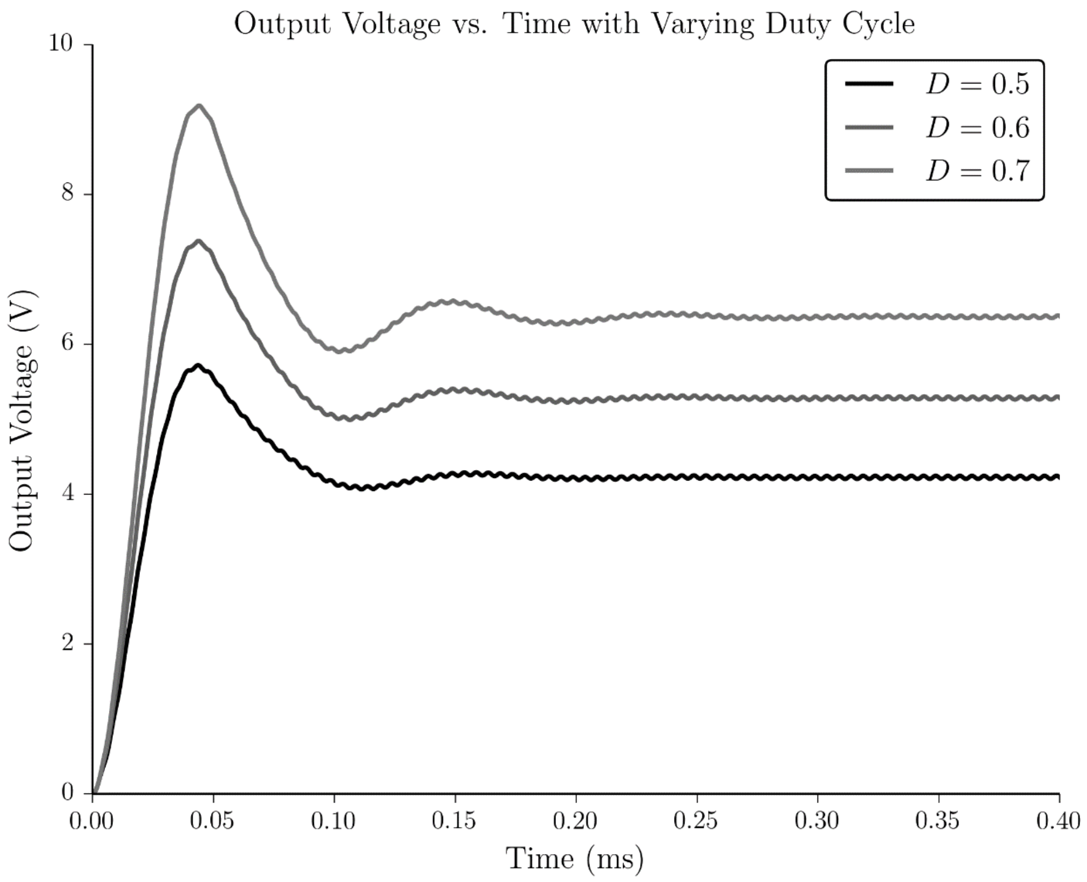


Fig. 12. *Buck converter output voltage over time with various duty cycle settings.*

Note the significant *overshoot* at the beginning of the simulation as well as the *output ripple* as the system settles in to steady state operation. The overshoot shown in Fig. 12 can be minimized through several methods. Since the duty cycle in this system is fixed this can effectively be considered the natural response of the inductor capacitor (LC) system, a true switching regulator would have some sort of dynamic control of the duty cycle that would significantly reduce the overshoot error. Another method for reducing overshoot is to use a *soft start* procedure which very slowly increases the output voltage to the desired level before entering into normal operation. Output ripple on the other hand is simply expected in any switching system, the effect of this ripple can be minimized through careful components selection in the output filter but it will always be present.

Boost Converter

The second important regulator type, a boost converter, has no equivalent in the linear regulator analogy. Boost converters are able to produce a higher output voltage than is supplied on the input. The mechanism for this will be discribed along with an analysis of various other factors that arise. Fig. 13 shows what a typical boost converter would look like.

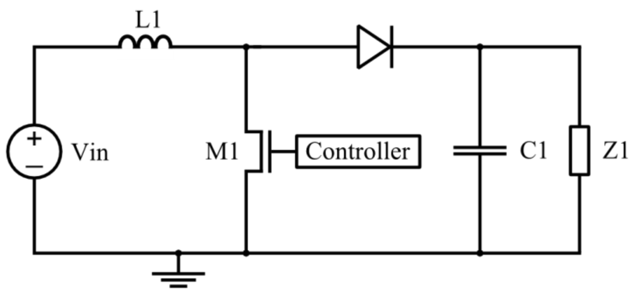


Fig. 13. A basic boost converter circuit where the switch is implemented with a MOSFET device.

Notice the switch, diode, and inductor have all switched places in comparison to the buck converter topology shown in Fig. 8.

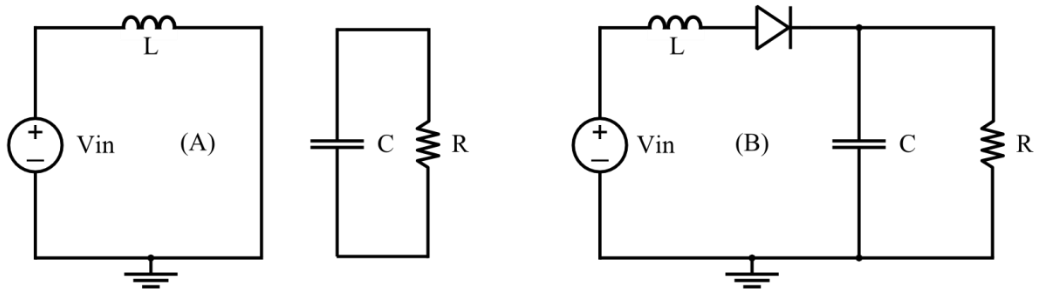


Fig. 14. A schematic comparison of the equivalent circuits of a boost converter formed when the transistor is ON or OFF.

In the figure above (A) shows the equivalent circuit of a buck converter when the transistor is on while (B) shows the equivalent circuit when the transistor is off. Much like the analysis in the *Buck Converter* section, the analysis of the boost converter will be split into continuous and discontinuous modes and the regulator will be considered to operate in the same three distinct states. As a reminder the states of operation are defined as:

- State 1 – Transistor ON, inductor current rising
- State 2 – Transistor OFF, inductor current falling
- State 3 – Transistor OFF, no inductor current

Continuous Mode

In state 1 the transistor-diode combination notably separates the input voltage source from the load. While in this state the inductor current is increasing as it stores energy from the input voltage source. If the output capacitor is charged then it is the sole supplier of current to the load. During the first few startup cycles the capacitor will not have enough stored charge to power the load for very long, if at all. However, in this steady state analysis it will be assumed that the capacitor is sufficiently charged to keep the boost regulator in the continuous mode. Therefore, to fully analyze state 1 it must be clear how much energy the inductor gains, and how much charge the capacitor loses. Once again referring to Fig. 10b we can see that during state 1 the inductor current is increasing from I_1 to I_2. As shown in Eq. 16 the voltage across an inductor is $V_L = L\frac{di}{dt}$. Furthermore from Fig. 14 it is clear that $V_{in} = V_L$ therefore:

$$V_{in} = L\frac{I_2 - I_1}{\Delta t} \tag{30}$$

During this state Δt is interchangeable with t_{on}, the change in energy stored in the inductor during this period can then be found by starting with the general form:

$$\Delta E = \frac{1}{2}L(I_2 - I_1)^2 \tag{31}$$

Rearranging Eq. 30 for $I_2 - I_1$ and substituting that into Eq. 31 gives the final form of

$$\Delta E = \frac{1}{2L}V_{in}^2 t_{on}^2 \tag{32}$$

Once the inductor current is approaching its peak value the transistor will switch to the off position and this analysis moves into state 2 where the inductor current is decreasing. The equivalent circuit for this state is shown in Fig. 14.

As before, in the analysis of the buck converter, the transistor switching action attempts to cause an instantaneous change in the current through the inductor but inductors do not allow such action and thus the phenomenon of inductive kick once again reverses the polarity of the voltage on the inductor. It is also clear from Fig. 14 that if the diode forward voltage is ignored the voltage across the inductor must be:

$$V_L = V_{in} - V_{out} \tag{33}$$

During this time, the current in the inductor is falling from I_2 down to I_1 so referring to Eq. 30 it can now be determined that

$$V_{in} - V_{out} = L\frac{I_1 - I_2}{\Delta t}. \tag{34}$$

During state 2 Δt is interchangeable with t_{off}, it is also known that the I_1 and I_2 values are the same during both states so therefore from states 1 and 2

$$I_2 - I_1 = \frac{V_{in} t_{on}}{L} = \frac{(V_{out} - V_{in}) t_{off}}{L}. \tag{35}$$

Substituting in $t_{on} = DT$ and $t_{off} = T(1 - D)$ and solving for V_{out}/V_{in} gives the following voltage ratio:

$$\frac{V_{out}}{V_{in}} = \frac{1}{1-D} \tag{36}$$

So it can be deduced from this equation that, like the buck converter, the boost converter output voltage is proportional to the input voltage and the duty cycle. However, in this configuration the output is inversely proportional to $1 - D$ rather than directly proportional to D. This relationship shows that as the duty cycle increases so does the output, from a minimum of V_{in} up to a value that is theoretically infinite, but practically limited by the size and performance of the components selected.

Discontinuous Mode

As in the buck converter section, the derivation of the boost regulator performance in discontinuous mode is omitted for brevity but can be found in [1]. The basic theory of operation is similar to continuous mode with the addition of an extra state as shown before. The voltage ratio for a boost converter in discontinuous mode is given by:

$$\frac{V_{out}}{V_{in}} = \frac{1}{2}\left[1 + \sqrt{1 + \frac{4DL_c}{L(1-D)^2}}\right] \tag{37}$$

A comparison of the ideal boost converter over varying duty cycle and in various modes of operation is shown below in Fig. 15.

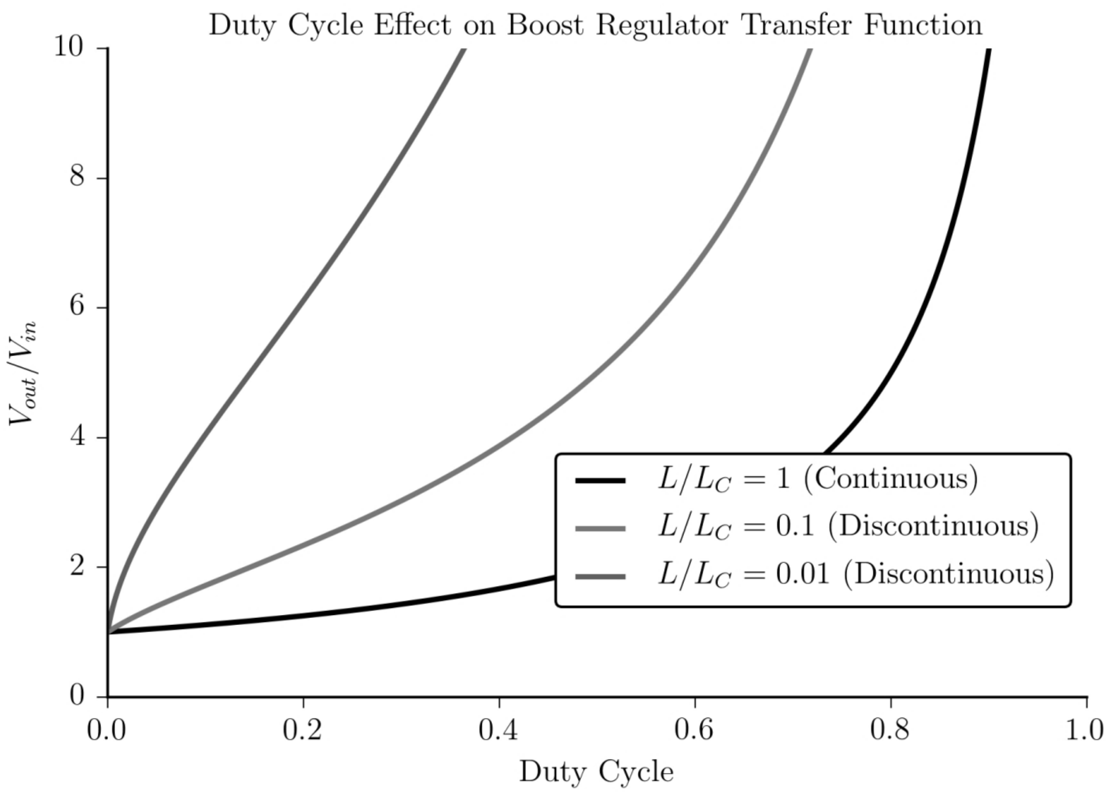


Fig. 15. Open-loop transfer function of boost converter in different modes of operation.

As Fig. 15 makes clear the boost converter output voltage increases with increasing duty cycle, one interesting difference from the buck converter open-loop response is that as the regulator enters further into discontinuous mode the eccentricity of the response actually decreases. In other words, rather than becoming more difficult to maintain a particular voltage, it actually becomes easier.

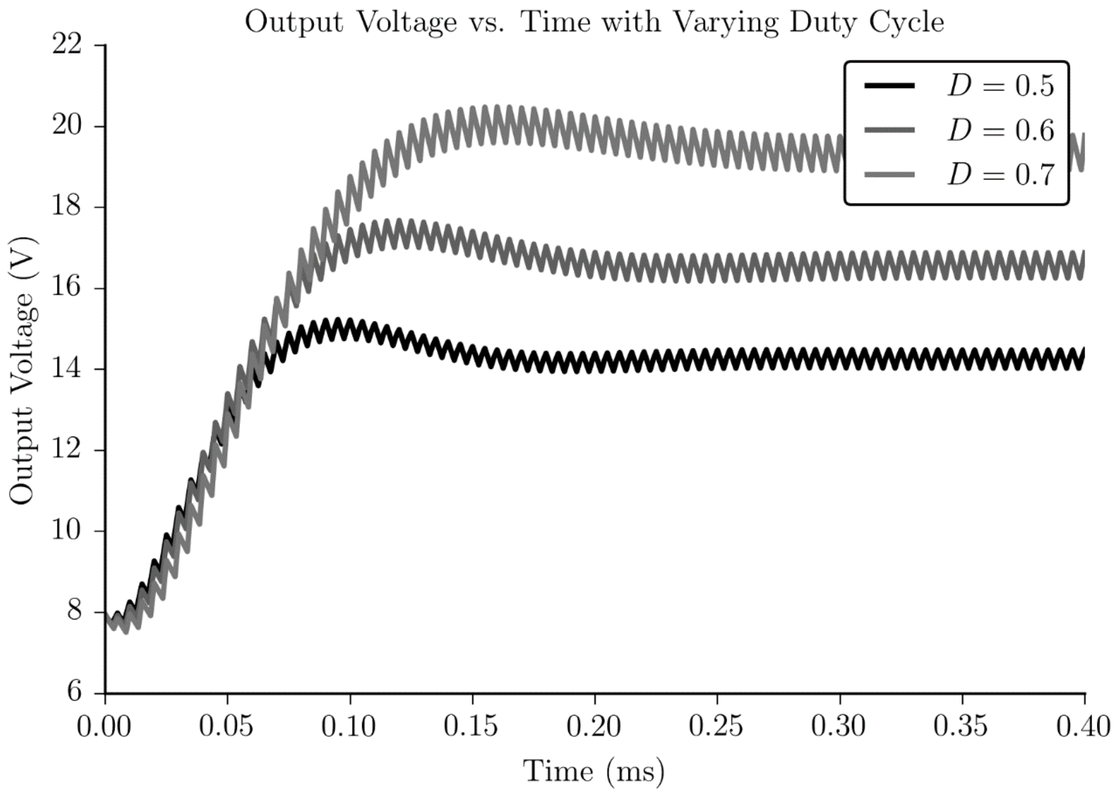


Fig. 16. *Output voltage of a boost converter over time.*

This system was also supplied with a 10 V input. The increased magnitude of the output ripple in comparison to the buck converter waveforms is a characteristic of boost converters but could be significantly reduced through a more intensive design process.

Switching Regulator Compensation Networks

Thus far the only switching regulator systems analyzed have been in the form of ideal open-loop regulators. While this type of analysis is instructive it is not representative of how switching regulators operate in regular use cases. In reality the duty cycle of a switching regulator is rarely, if ever, fixed.

Take for example a buck regulator with $V_{out} = DV_{in}$, if the input voltage is coming from a battery that is slowly draining then the input voltage will slowly drop. If the duty cycle is fixed then the output voltage will drop along with the input voltage. If however the duty cycle is variable and controlled then it can be raised as the input voltage drops in order to maintain a steady output voltage. The figure-of-merit in that example is *line regulation* which refers to the ability of the regulator to maintain a constant output voltage despite changes on the input.

Now consider an industrial DC motor that is powered by a switching regulator. If the motor is off, then the regulator should have no trouble holding a constant output voltage but if the motor is suddenly turned on then it will impart a large load on the switching regulator and if the control scheme is poorly designed then this load change may cause the output voltage to drop down far enough that the motor never even starts spinning. If the regulator were able to react more quickly to load changes then it would increase its *load regulation* figure-of-merit.

In both of these cases the primary factor in how well the regulator responds is how well the switch controller is designed, generally the dominating factor in designing this control scheme will be the components chosen for the compensation network. For a full reference on designing switching regulator control schemes the reader is referred to [1, 2]. In many cases the control scheme used will be a type of pulse-width modulation (PWM) where the duty cycle of the switching regulator is raised or lowered depending on the pulse width of a relatively high frequency signal. A typical voltage-mode PWM control scheme is shown in Fig. 17.

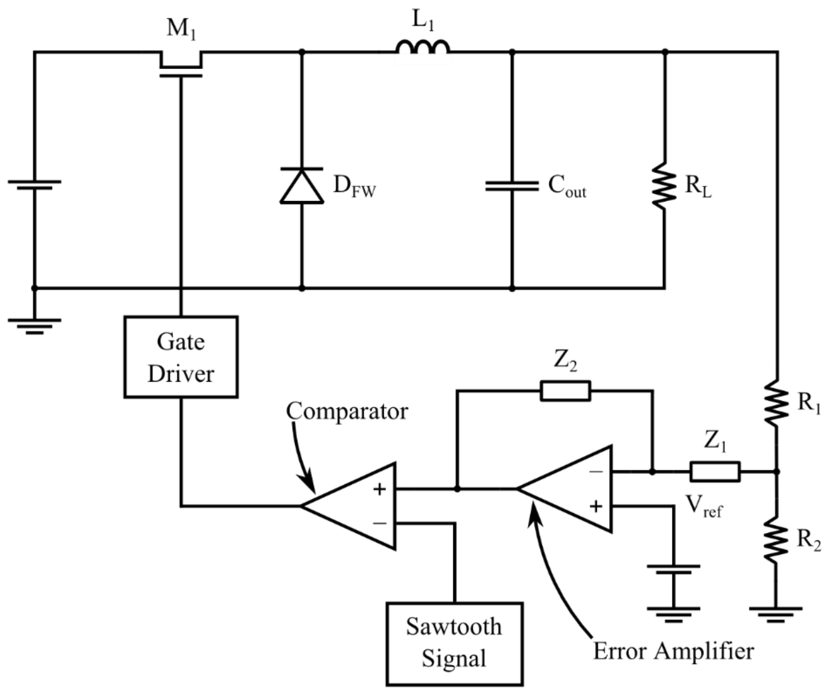
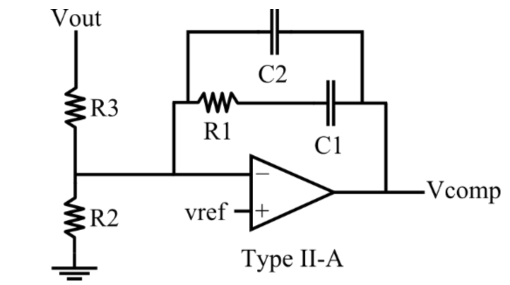


Fig. 17. Voltage-mode pulse width modulation control scheme on a buck regulator.

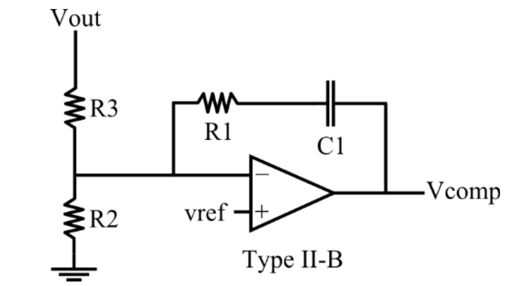
This figure shows Z_1 and Z_2, the two impedances which form the *error amplifier feedback loop*. The feedback loop combined with the error amplifier form what will be referred to in this work as well as within the tool as the *compensation network*. While this terminology is not universal it is commonly used in industry and will be consistently used throughout this work.

Table 1. Compensation network types as used within this thesis and modeling tool.

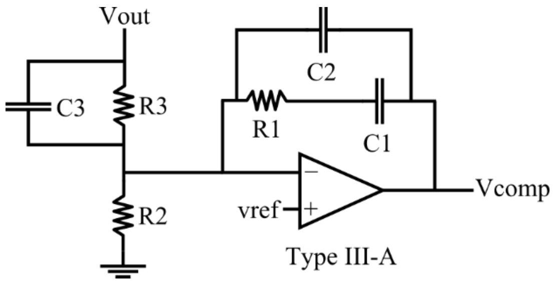
II-A



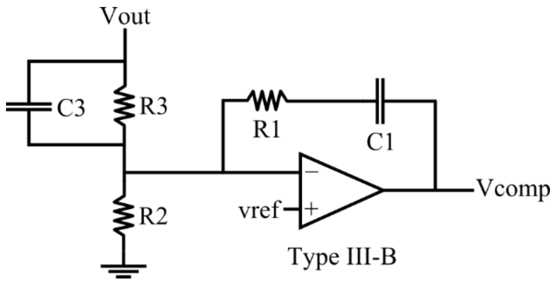
II-B



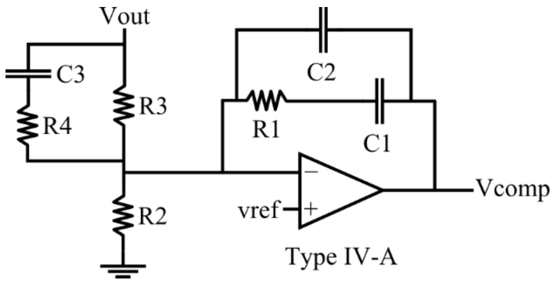
III-A



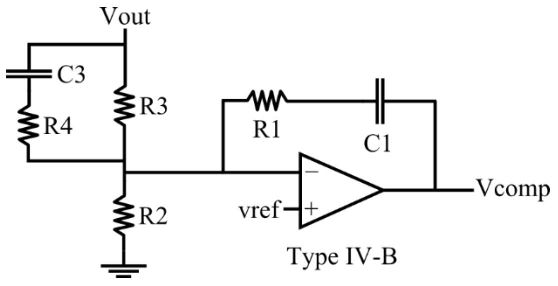
III-B



IV-A



IV-B



This table shows a fairly straightforward naming system where increasing the numeral increases the order of the input filtering function and changing from type A to type B removes C2 from the error amplifier feedback loop.

1.2 Regulator Behavioral Modeling Background

Simulation is an important part of any electronic product design and will be for the foreseeable future. Many designs incorporate a switching regulator in one form or another whether it is integrated as a system-on-chip (SOC) IC component or as a monolithic IC at the board level. Either way, the design engineer will want to simulate their system to ensure that all of the components work correctly together. This is generally good practice but switching regulators tend to complicate matters and can easily make simulations take hours where they would otherwise take minutes or even seconds. It is therefore common practice to use a simplified regulator model when it is convenient and useful to do so. As a result, there is significant interest in switching regulator modeling and there are many methods aimed at creating the best model that is both accurate and fast. An overview of the most common modeling methods is provided below in Table 2.

Table 2. Summary of common switching regulator modeling methods.

Method	**Description**
State-space averaging [3]	The state-space modeling method is the classic technique for creating many types of circuit models but proves specifically useful for switching regulator models since these circuits tend to have only 2-3 “states”.
PWM switch averaging [4, 5, 2]	Used in the current research, averages out behavior of switching element to provide a fast drop-in replacement.
Discrete time modeling [6]	Using state-space averaging in the discrete time domain. Produces highly accurate models but are not SPICE compatible.
Black-box modeling [7]	Parameterized models where parameters have no correlation to physical parameters.
Gray-box modeling	Similar to black-box models but parameters have a direct physical interpretation. Supplied as a built in set of functions in Matlab.

While there appear to be many different approaches to modeling switching regulators there are only two methods that have reached wide adoption. The first, state-space averaging is the oldest method and was proposed by Drs. Middlebrook and Ćuk in the 1976 conference paper “A general unified approach to modelling switching-converter power stages” [3]. The second method, and the approach used in this work, is the PWM switch average model. This method is also fairly mature though not as extensively employed as the state-space averaging method. The PWM switch average model was developed for continuous conduction mode (CCM) and discontinuous conduction mode (DCM) over a two paper series by Dr. Vorpèrian in the May 1990 issue of IEEE Transactions on Aerospace and Electronic Systems [4, 5]. The two methods were later unified to create a model that works in both operating modes [2]. The objective for both of these modeling methods is much the same, average out the switch but maintain the behavior that it imparts with the difference being how each method accomplishes that task.

In state-space averaging the modeler considers all of the states that the switching regulator can exist in and derives a duty-cycle based transfer function for each state. The transfer function is representative of all the components in the system from transistors down to diodes.

The PWM switch model is less flexible than the state-space averaging method in that it can only be used in regulators that are controlled using either the voltage or current controlled PWM control scheme. However, within this subset of switching regulators the PWM switch model is powerful and is incredibly easy to apply. Rather than requiring the modeler to derive a new set of state-space equations for each circuit this approach attempts to average the behavior of the switch itself while leaving the rest of the circuit untouched. Generally this means that the PWM switch model can serve as a direct drop-in to the original circuit. In addition to simplicity the most important feature of the PWM average switch model is that it is thousands of times faster to simulate than a cycle-by-cycle model using a simple switch element, and orders of magnitude faster to simulate than a transistor level cycle-by-cycle model.

Both methods have their advantages and disadvantages but due to the simplicity of implementing the PWM switch method, it appears to lend itself better to automation and thus is the chosen method applied in the rest of this work. This decision is revisited in the *Conclusions and Future Work* section.

1.3 Transient and AC Measurements of Switching Regulators

Performing measurements on switching regulators is essential to verifying that they are operating as expected, and it is useful for determining what the regulator is capable of. For the purposes of this research, high fidelity measurement data is essential in constructing an accurate model. While there are many different types of switching regulator measurements that produce valuable results, there are three specifically which are most relevant to creating behavioral models. These three measurement types are des in the sections below.

1.3.1 Transient Load Change Measurement

The transient load change measurement is accomplished by using the switching regulator in its normal mode of operation, observing the output voltage under a constant load, and then rapidly pulsing that load up and then back down. Ideally the output voltage of the regulator will not change. More realistically it will only show a small glitch before returning to the previous steady state value. In the worst case scenario the pulsed load will cause the regulator control loop to enter into an unstable condition and the output voltage will either rise uncontrollably until a critical component is damaged or it will fall as low as possible. The canonical testbench for a switching regulator under transient load change test is shown in Fig. 18.

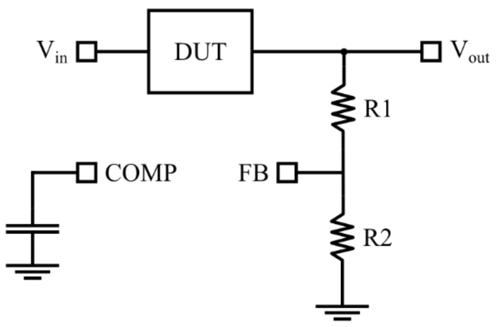


Fig. 18. Canonical testbench schematic for a switching regulator under transient load test.

In this schematic and for the remainder of this work the COMP pin is the output of the operational amplifier in the compensation network while the FB pin is the inverting input to the same operational amplifier.

The purpose of a transient load change test is twofold. On one hand the test provides quantitative results showing how a particular switching regulator will respond to load changes on the output, on the other hand the test can be useful for exploring or identifying the effects of load changes that put the system into an unstable state.

1.3.2 AC Loop Measurement

While the transient load change test can be useful for identifying instabilities, the AC loop measurement is purpose built to do so. The canonical testbench for the AC loop measurement is shown below in Fig. 19.

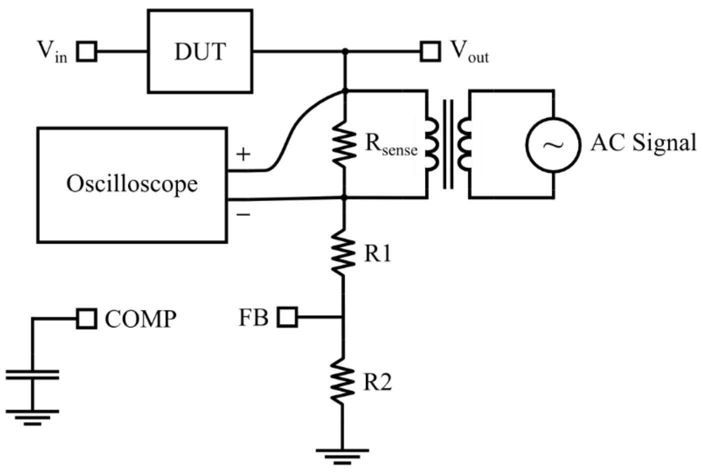


Fig. 19. Canonical testbench schematic for a switching regulator under AC loop test.

At a schematic level the essential difference between and AC loop measurement and a transient load change measurement is fairly clear. It is important that the compensation network be detached from the output node in order to *break the loop*. This terminology refers to the fact that disconnecting R1 from the Vout pin disconnects the compensation network from the output and thus breaks the feedback loop.

In order to perform an AC loop test the R_{Sense} resistor should be chosen as a small-resistance low-inductance resistor. A resistance of 50 Ohms or less is generally safe, though the exact value depends on the values of R1 and R2. With the R_{Sense} resistor in place a wide-bandwidth step-down signal transformer should also be introduced into the circuit; this is both to electrically separate the AC signal injection and AC signal measurement points and to ensure that the signal injected from the signal generator is truly as small-signal as possible. Finally, a small-signal AC voltage can be injected into the primary windings of the step-down transformer while the response is measured on the secondary side across the sense resistor. The parameters obtained from this measurement are then:

$$A_{loop} = dB\left(\frac{V(out)}{V(ac_{in})}\right)$$

and

$$\phi_{loop} = \Phi_{\mathrm{out}} - \phi_{ac}$$

These parameters represent the overall system response to forced voltage variations on the output node. The AC loop measurement is the most commonly produced AC measurement on commercial switching regulator devices and is thus the measurement that is used in comparisons in the *Results* section. For more information on the purpose of the AC loop measurement and how it is performed refer to [8].

1.3.3 AC Plant Measurement

Finally the AC plant measurement is similar to the AC loop measurement but rather than observing the response of the entire regulator and compensation network system this test bypasses the compensation network entirely and removes it from consideration. The purpose of this test is to determine how the switching regulator controller responds to forced voltage variations on the output with no compensation network to improve stability. These test results can be used to analyze the robustness of the switching regulator controller separately from the rest of the system. More commonly however, the AC plant measurement is used by the board level designer in order to select the optimal components for the compensation network. The testbench schematic for the AC plant measurement is show in

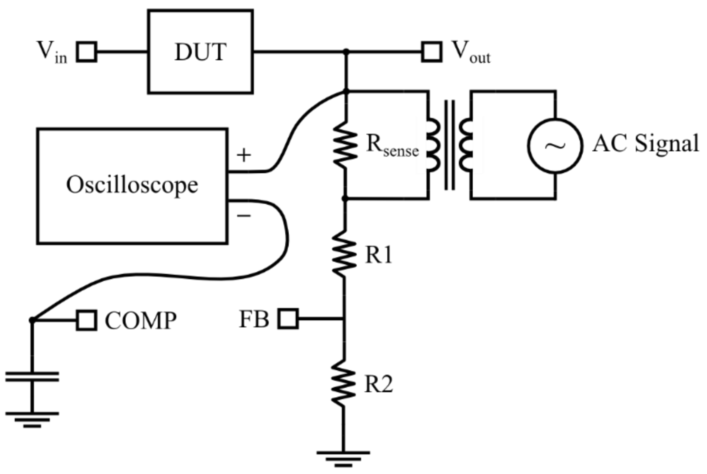


Fig. 20. Canonical testbench schematic for a switching regulator under AC plant test.

For further discussion on AC plant measurement and its utility refer to [8].

1.4 Thesis Statement

With a firm understanding of both what device this modeling method is aimed at and the PWM switch method as it is used in this work it is now possible to explore the topic of automating the modeling process. The work presented herein rests on the following **assumptions**:

- **A1.** The PWM switch modeling method is robust and accurate in a wide variety of applications under the condition that the regulator control scheme is PWM based.
- **A2.** The dominating factor in switching regulator behavior is the selection and design of the compensation network.[1]
- **A3.** Given a bode plot for a switching regulator and loose constraints on the compensation networks allowed, modern computers are capable of effectively exploring a design space to find a unique solution to what system produced the original plot.

With these assumptions in mind, this work attempts to **prove** that:

[1] This assumption proves true in general cases, however is less robust when faced with a system that is significantly affected by parasitics.

P1. It is both possible and feasible to automatically generate a behavioral model of a PWM switch controlled switching regulator from AC and transient data.

P2. The developed software effectively implements this method in a manner that is usable, useful, and extensible.

2. Model Generation Methods

This section introduces the modeling method and explains how the underlying software is designed. The operation of the tool will initially be described through an example of how a model would be created by hand. Finally automation of the model generation process is described.

There are currently three identified methods for creating models in the tool, *user configured*, *constrained optimization*, and *fully automated*. Only the user configured and constrained optimization methods have been implemented since further work is required on the fully automated method but all three are described in the following subsections.

2.1 User Configured Process

The first step in the regulator modeling process is to define which switching regulator topology is in use. This will usually be one of the more common topologies such as buck, boost, or buck-boost though there are many others. The tool currently only implements these three but due to the modular way in which this is implemented it would be simple to add others.

The next step is to identify the control type for the compensation network. As has been described in previous sections the compensation network samples the output node and determines how the switch should be modulated to maintain the output node's correct value. There are several methods that can be used to monitor and respond to the output. The three common methods are:

- **Voltage sampling** – Directly measure the voltage at the output using a voltage divider or other simple circuit.
- **Peak current sampling** – Measure the peak current through the output inductor.

- **Average current control** – This control scheme modulates the switch based on the average current through the inductor, this is of course similar to the peak current sampling method but can better compensation for operating mode transitions.

Next, the modeler identifies the parameters associated with the chosen compensation network. This includes component values, design parameters such as switching frequency of the PWM controller, and intrinsic device characteristics such as the open loop gain and gain-bandwidth product of the operational amplifier.

The switching regulator modeling process concludes with an analysis of system level non-idealities as well as major component level non-idealities. In this case, a non-ideality is considered anything that deviates from the canonical model. This includes common non-idealities such as inductor and capacitor equivalent series resistance, as well as less common non-idealities such as using a dual-switch configuration with both a low-side and high-side switch.

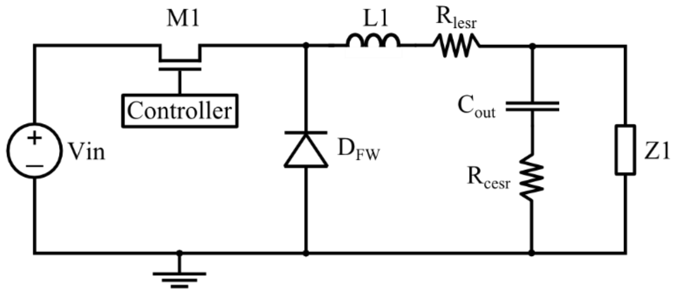


Fig. 21. Example of a buck regulator with non-ideal considerations for the output inductor and output capacitor.

The user configured modeling method presented here has been implemented in the software package in order to make the manual model creation process straightforward.

2.2 Constrained Optimization Model Generation Method

The constrained optimization model generation method takes the next logical step beyond the user configured process and allows the modeler to optimize the generated model based on a specified set of parameters. For the purposes of explanation it will be assumed that the modeler is optimizing the switching regulator model to a target AC response, but the method theoretically applies just as well to transient models.

The first few steps in the process are generally the same as in the user configured method; the simulation still must know what the regulator topology is and what the switching frequency is set at as well as similar basic information. After this information has been supplied the modeler will input a target frequency sweep dataset that includes gain in dB and phase in degrees. This dataset will be referred to as the *target set*, the target set is interpolated to produce a frequency axis with graduations at regular intervals and gain/phase values which occur at the same point. While data is often regularly spaced when obtained from a simulator or from a measurement this is almost never the case when working from digitized data.

Next the modeler must identify which parameters he believes will be dominant in optimizing the model; for each of these parameters an allowed range and a step size should be defined. Once all of the parameters have been identified and bounded, several parametric simulations will be initiated. For each of these parametric variations another AC dataset is created, these sets will be individually referred to as the *generated set* and collectively as the *parametric space*.

When optimizing an AC response both the phase response and the gain response must be considered when calculating error. This implies that for each comparison of a generated set to the target set there will be two errors to consider. The optimal agreement between target set and generated set occurs when the total error is minimized as compared to all other generated sets in the parametric space. This means that the only remaining task is to calculate the two values of error for each generated set in the parametric space when compared to the target set.

For this work a simple model of error calculation was assumed where the error of a single dataset can be calculated according to

$$ERR = \frac{\sum_{i=1}^{n} |x_{Ti} - x_{Gi}|}{N}$$

Where x_{Ti} and x_{Gi} refer to the quantity under consideration, such as phase, gain, or voltage, for the target and generated sets respectively and N is understood to be the number of interpolated data points in the set. In an AC optimization this leads to two well defined *coefficients of error*, the *coefficient of error of gain* and the *coefficient of error of phase* which can be defined by:

$$CoEG = \frac{\sum_{i=1}^{n} |A_{Ti} - A_{Gi}|}{N} \quad \text{and} \quad CoEP = \frac{\sum_{i=1}^{n} |\phi_{Ti} - \phi_{Gi}|}{N}$$

Finally, the optimal generated set can be found by finding the set within the parametric space which minimizes $CoEG + CoEP$. For verification of this process refer to the *Results* section in which this optimization routine is successfully applied to various switching regulator models.

2.3 Fully Automated Model Generation Method

Considering that the constrained optimization method successfully optimizes switching regulator models the next logical jump is to make the entire process as hands-off as possible. Such a process has been identified though not yet tested or implemented in the tool. The process requires a minimum viable data set of:

- regulator DC input voltage
- regulator steady state DC output voltage
- sawtooth wave switching frequency

- compensation network control type
- AC loop transfer function magnitude and phase response
- transient load test for model verification purposes

Given this information the modeling tool executes the following process to identify key regulator characteristics and build a model. Comparing the input voltage to the output voltage can lead to a unique solution for which type of regulator topology is in use. If the core types of buck, boost, and buck-boost are the only allowed types this comparison should map uniquely, however if other types of topologies are allowed the modeling tool may require further user input for this step. Once the topology is determined the next step is to determine the compensation network characteristics. The first part to this step is to identify what type of compensation network is in use in the regulator. This information can be determined by simply determining the number of poles and zeroes (roots) in the measured AC response. Each of the types of identified compensation networks has a specific number and location of roots and if data analysis functions were implemented which could identify these roots it would theoretically be possible to automatically determine what type of compensation network is in used solely from the AC response data. Furthermore, if the location of each pole and zero were able to be determined relatively precisely then they could be used to back-calculate the value of each component in the compensation network. With the compensation network parameters reasonably approximated the remainder of the process can be continued by proceeding forward with the previously described *Constrained Optimization Model Generation Method* where the limits and step sizes of parameters are programmatically determined.

3. Modeling Tool Architecture and Implementation

This section discusses the high level architecture of the modeling tool as well as implementation details. At the highest level there are four main components in this system, they are the plot digitizer, circuit simulator, Python core, and the graphical user interface (GUI). Throughout this section and for the remainder of the document code snippets or concepts which have a direct implementation in code will be `formatted in this manner`.

The plot digitizer component is forked from the original source code of the open-source *Plot Digitizer* project. This software is described by its creator as follows [9]:

> *Plot Digitizer is a Java program used to digitize scanned plots of functional data. Often data is found presented in reports and references as functional X-Y type scatter or line plots. In order to use this data, it must somehow be digitized. This program will allow you to take a scanned image of a plot (in GIF, JPEG, or PNG format) and quickly digitize values off the plot just by clicking the mouse on each data point. The numbers can then be saved to a text file and used where ever you need them. Plot Digitizer works with both linear and logarithmic axis scales. Besides digitizing points off of data plots, this program can be used to digitize other types of scanned data (such as scaled drawings or orthographic photos).*

The modifications required of this tool were minor. The only changes made were to remove unusable functionality, rename button labels to make them more intuitive, and add a mechanism whereby the plot digitizer tool automatically sends data back to the regulator modeling tool without requiring extra steps from the user.

The second component, the circuit simulator, is not a tool that was developed upon directly but tight integration was still required. Furthermore, rather than developing the modeling tool so that it could only interact with one simulator it was desired that the tool would be able to use any arbitrary simulator that meets the minimum requirements. This means that Interface-Adapters needed to be developed in order to support that level of modularity [10]. These adapters are described in the *Python Core* section.

The next component, the Python core, is the main collection of tools in the entire software package and is composed of six submodules. The details of this component will be described further in their own section. Development of this component comprised the majority of time spent in development and it is correspondingly the largest and most technically interesting component of the modeling tool. This portion of the tool is largely independent of the other components in the tool and is completely usable from the command line but has been developed in a manner such that it can easily cooperate with a simpler user interface.

The final component in the switching regulator modeling tool is the graphical user interface (GUI). The GUI can be considered the "glue" holding everything together and making the four distinct components operate as a cohesive unit. The interface design was carefully considered in order to provide a user experience that is intuitive without sacrificing technical capability. More on the development and usage of the GUI will be covered in the *Graphical User Interface* section below.

Fig. 22 below visualizes the interaction of these four separate components and their submodules in an effort to provide a comprehensive overview of the architecture of the tool. The two main components of the Python core and the graphical user interface are described more thoroughly in the following section.

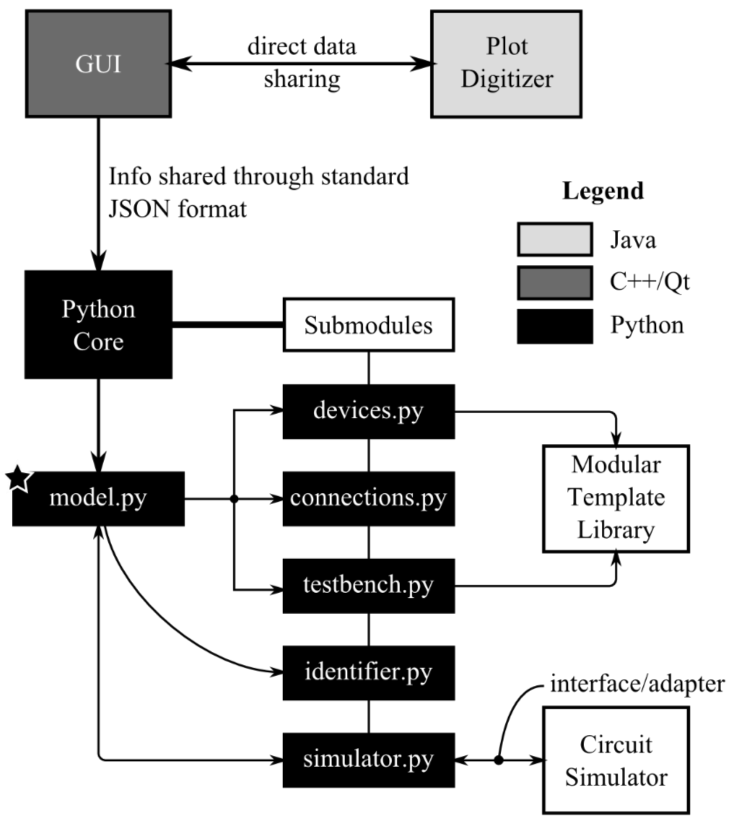


Fig. 22. Switching regulator modeling tool high level architecture.

Worth noting from Fig. 22 is the range of languages used in the project. The main core was written in Python, the GUI was developed in C++ against the Qt library, and modifications to Plot Digitizer were performed in Java. Additionally data sharing between components was performed using JSON syntax and the simulator specific templates were written using the third-party Python library Mako which has its own unique syntax. Data sharing between Plot Digitizer and the GUI was accomplished through a direct "pipe" between the two programs.

3.1 Python Core

As previously stated, the Python core is an umbrella term for the submodules that perform the primary functions of the switching regulator modeling tool. These submodules are `model.py`, `devices.py`, `connections.py`, `testbench.py`, `identifier.py`, and `simulator.py`. The purpose and implementation of each of these submodules is described in the following sub-sections.

3.1.1 model.py

`model.py` is the main entry point of the Python core and essentially serves as a command-line UI. The two primary purposes of this submodule are:

- Contain the majority of the error checking that is required through the program in order to minimize the likelihood that the modeler will enter into an unnecessarily long simulation.
- Provide a simple interface into the more low-level submodules described in the following sections.

The majority of what is implemented in the submodule is standard Python and not technically interesting so it does not need to be described in detail here, but it can be easily understood through the source code comments.

3.1.2 devices.py

This submodule provides a general interface to the devices required in a typical simulation model. Devices provided by this submodule are enumerated and described in Table 3 below.

Table 3. Summary of simulation devices provided by devices.py.

Device Name	**Parameters**	**Description**
`Device`	`device_name, nodes, hashed_nodes`	The parent device which provides methods and properties which are common to all simulation devices. These more specific devices are created by sub-classing this class.
`Resistor`	`value, device_name, type`	Implements the basic functionality of an ideal resistor. If a more complex resistor model is required then this class should be sub-classed with the additional parameters, and the corresponding simulator templates should be added. Simulator templates are described in section 3.1.7 below.
`Inductor`	`value, device_name, type, initial_condition`	An ideal inductor with an optional initial current condition. Sub-classing recommendations are the same as for a `Resistor`.
`Capacitor`	`value, device_name, type, initial_condition`	An ideal capacitor with an optional initial voltage condition. Sub-classing recommendations are the same as for a `Resistor`.
`Voltage`	Various, depends on `type`	A generic voltage device that can implement several types of voltage sources such as DC, AC, sine, pulse, and piecewise linear.
`Current`	Various, depends on `type`	A generic current device, internally implemented exactly the same as `Voltage`.
`PWM`	`device_name, control_type, inductor_value, switching_frequency, duty_cycle_*`, others depending on previous variables	This is the first of the devices which is not native to any simulator. It is implemented through a template and models the switching behavior in a switching regulator by using the PWM modeling method.

Device Name	**Parameters**	**Description**
`Compensation`	`device_name,` `device_type,` `control_type,` `**compensation_kwargs`	The second device that is non-native. Main determinant of the frequency response of the control system. It is implemented as three terminal subclass of `Device` with four required parameters.
`Ground`	None	Ground connection for the circuit. One weakness of this implementation is that there can only be one ground in the entire circuit, though this is okay for the vast majority of models.

In addition to providing an interface for these devices, this submodule contains a small collection of helper functions, classes, and exceptions. These helpers are not novel or technically interesting and are not described here, but are thoroughly documented in source code comments.

3.1.3 connections.py

This submodule is responsible for handling the interconnection between devices. It serves as a proxy for a formal netlist definition such as what is used in SPICE or Spectre. Instead of a netlist, the connections are described in an object-oriented paradigm where each device has a specified number of unique nodes which can then be connected to each other by resolving one of the node names to be the same as the other. The unique node identifiers are generated using a standard version 4 UUID generator that is then truncated down to 10 characters [11]. Multiple devices can be connected to each other by resolving all of the device nodes to the same unique node. Devices can be disconnected by changing the unique node name of one of the devices to a new UUID.

In addition to tracking device connections this submodule also handles part of the functionality of testbenches. As will be decribed in the following section the `testbench` module makes it simple to prepare a testbench that will work in multiple simulators. The `connections` module is responsible for tracking which nodes should be used for AC and Transient analysis. More specifically, the connections module tracks which nodes should receive the input signal and which node should be monitored as an output.

3.1.4 testbench.py

The `testbench` module is the third and final module that aims to make it simple to produce netlists and simulation commands for multiple types of simulators. In this case the module provides an abstraction for two types of testbenches, an `ACTestbench` and a `TransientTestbench`, both of which are a subclass of the prototypical `Testbench` class.

The `Testbench` class is a *prototype* that implements some of the basic features that are common to all of the testbench types, it also handles bookkeeping that is common to all types of testbenches such as:

- `input_node` and `output_node` – These nodes indicate where the test signal should be injected and where the response should be measured, respectively. They are specified by naming a node on a specified device in the connection map.
- `simulator, simulation type` – These parameters are used internally when looking up the templates to use when actually producing the netlist. Otherwise unused.
- `load_type` – Within the tool switching regulators can be loaded either with a specified current or with a specified resistance/capacitance. The allowed parameter values are `current` or `rc`.
- `input_voltage` and `output_load` – Two parameters which are an internal representation of the devices used for powering or loading the switching regulator circuit.

Subclassing `Testbench` is the first type of testbench, the `ACTestbench`. This type of testbench is, as the name implies, used for AC simulations. In addition to the base methods defined in `Testbench` the `ACTestbench` adds and implements the following parameters:

- `start_frequency` and `end_frequency` – These two parameters set the lower and upper limits on the AC sweep frequency, respectively.
- `sweep_type` – This can be one of three string values, “decade”, “octave”, or “linear”. The `sweep_type` parameter controls the spacing of the simulation data points.
- `points_value` – The `points_value` parameter is used to set the number of simulation points. In a logarithmically scaled simulation (“decade” or “octave”) this parameter is used to set the number of points in a single segment. In a “linear” simulation this parameter sets the total number of data points.

Finally the `TransientTestbench` class implements the methods for running a transient simulation. This class is simpler than the `ACTestbench` and thus requires fewer parameters. These parameters are described below:

- `print_step` – This is the time interval between successive data points. This is not necessarily related to the simulator. If the simulator is not capable of exporting data according to this specification then the data will be interpolated through the identifier.py submodule.
- `end_time` – The time in seconds at which this simulation should finish.

While most simulators offer other types of simulation such as noise analysis, Monte Carlo, or parametric sweeps the basic AC and Transient simulation are the core of any simulator and are most important in this work, thus they were implemented first. Others could be added if desired, though that may make compatibility difficult to maintain across various simulators.

3.1.5 identifier.py

This submodule is currently a collection of helper functions for working with generated simulation data. This submodule is intended to contain the functions used for automatically identifying poles and zeroes, as well as other data analysis features. However, as most of these features have not yet been implemented this module name is a bit of a misnomer.

The most important functions currently contained in this submodule are the interpolation functions and their helpers. This includes the `linear_interpolation()` function and the `spline_interpolation()` function, and their helpers `linear_axis()` and `logarithmic_axis()`.

The linear axis function expects two parameters, where the first, `x_data`, is a single set of data points and the second is an `increment` parameter which tells the function how far apart each x-axis point should be. The minimum and maximum data values in `x_data` are used as the lower and upper bounds of the new axis with each data point spaced one `increment` from the last.

The logarithmic axis function is similar to the linear axis function with the exception being that the values it returns are spaced logarithmically and the increment parameter refers to how many data points there are per decade or octave, it also requires an extra parameter to specify if the data is divided into octaves or decades.

This difference between the two functions is important for analyzing data that was captured over a logarithmic scale. Attempting to interpolate data from a frequency sweep that goes from 1 Hz to 1 MHz using a linear axis produces a prohibitively large data set.

Consider for example a linear axis from 1 to 1,000,000 at an increment of 1. From 1 to 10 there are 10 data points, this is manageable. From 10 to 100 there are 100 data points, this is manageable. However, from 100,000 to 1,000,000 there are 900,000 data points and the data becomes difficult to analyze. Analyzing the same data set using a logarithmic axis with an increment of 100 produces instead 100 data points from 1 to 10, 100 data points from 10 to 100, *etc.* The end result being that rather than analyzing 1,000,000 data points we only need to consider 6,000. Analyzing a smaller dataset means that more data can be analyzed at once which more quickly leads to results.

The other two functions, `linear_interpolation()` and `spline_interpolation()` are essentially just interfaces to the SciPy functions `interp1d()` and `InterpolatedUnivariateSpline()`. The interface pattern [10] is adopted here for two primary reasons:

- Providing a function locally means that the function name and parameters can be specified regardless of the implementation in SciPy. If the SciPy library were to change the name of one of these functions it would simply mean that I have to change the name in one place rather than many times throughout the code.
- Passing the data through another function before calling on SciPy allows the data to be validated first, this allows for more robust error prevention and error handling.

In addition to the interpolation functions this module is planned to be used to implement the other data analysis functions previously mentioned in the *Fully Automated Model Generation Method* section such as pole and zero identification, identification of compensation network type from frequency sweep, and Pareto frontier comparison of parametric sweeps.

3.1.6 simulator.py

As shown in Fig. 22 the `simulator.py` submodule operates as a two-way link to send commands to and receive data from simulators that are supported by the modeling tool. The most important function of this submodule is to act as an interface-adapter between the modeling tool and the selected simulator. This means the submodule is responsible for implementing a uniform API for running specific simulations regardless of the simulator, as well as implementing an API for reading data in from a specified simulation and translating that data into a common format which can be used throughout the rest of the modeling tool.

Understandably this submodule is critical to the reliable operation of the modeling tool. There are several helper functions contained in the `simulator.py` submodule but the two most important functions are `run_simulation()` and `get_simulation_data()`. Both of these functions are essentially wrapper functions that verify the supplied parameters and then pass the arguments on to more specific sub-functions.

The `run_simulation()` function expects at least three parameters but is capable of accepting an arbitrary number of parameters in order to flexibly support more advanced options that are available in one simulator but not in others. The required parameters are `simulator`, `id`, and `sim_file`.

- `simulator` will just be a string with the `internal_name` of the simulator to use. Currently supported simulators are `pspice` and `spectre`.
- `id` is a string or a raw Python UUID object which uniquely identifies this simulation run.
- `sim_file` is either the Python file-like object or the decoded string contents of such an object that will be sent to the simulator.
- Additionally a fourth `kwargs` parameter is allowed for, this is a standard Python declaration that allows an arbitrary number of parameters to be passed into the function as a single dictionary.

The `get_simulation_data()` function is similar to the `run_simulation()` function in the parameters that it expects though it is more simple. This function expects the `id` parameter in order to uniquely identify and locate the simulation file and allows for an optional `simulator` parameter in order to specify which simulator produced the dataset. If this is known ahead of time, as it usually is, then it is best to supply this to the function directly. If the `simulator` parameter is not known ahead of time then the function will attempt to identify the simulator automatically based off of the file syntax and storage format.

3.1.7 Template Library

The template library is the submodule that truly brings everything together. While the other submodules are used to provide general functionality internal to the tool, the template library is used to make the internal circuit and testbench description usable in a specific simulator. There are currently two types of templates that are used in the modeling process, the first that will be described are regulator templates which are used by `connections.py` to properly handle the specified regulator topology, the other type of template is simulator templates which are used by `devices.py` to know what syntax the simulator expects for a variety of device types as well as simulator commands.

Regulator Templates

The specification for regulator templates is as follows: The regulator template files should be proper Python files, meaning the filenames are expected to end with the .py extension and be fully Python 3.4 compatible. The templates should be placed in the "templates/regulators/" directory relative to the program installation directory. Regulator template files are then required to implement three functions.

The first required function, `program_usage_details()`, is straightforward and is only required to return a single dictionary with three keys of information. The dictionary should contain one key which is the `internal_name` the value associated with the `internal_name` key should be all lowercase with any spaces being replaced with underscores or hyphens and this value specifies how to regulator will be referred to in the code. The `external_name` value should be a human friendly name for the regulator topology, this is the name that is shown to the user. Finally, the `description` key should contain a brief description of what this regulator topology is used for, this is used in the GUI as a tooltip reminder to the user.

The other required functions are `connect_transient_model()` and `connect_ac_model()`, these functions handle all of the operations required to connect devices into the expected configuration for the specified test. The parameters required by these functions are uniform and are detailed in Table 4 below.

Table 4. A table describing the parameters required by the connect_*_model() functions.

Parameter	**Description**
`regvars`	A dictionary containing all of the switching regulator parameters that aren't directly related to the compensation network.
`compensation_information`	A dictionary containing all of the information related to the compensation network, this includes components values, network type, and similar parameters.
`cmap`	This should be the global `ConnectionMap` that is created at program initialization.
`testbench`	Testbench should be a subclass of the Testbench class which was previously described. This parameter is used to create the devices for input voltage and output loads.

These parameters are used to create and connect the required devices and implement a particular regulator topology. An alternative use for this system is described in the Conclusions and Future *Work* section.

Simulator Templates

The simulator templates are used primarily by the `devices.py` module to provide a straightforward method of using the same devices across different simulators. If the `devices.py` module can be thought of as providing the general interfaces for using these devices then the simulator templates library can be considered to be the module that brings the two together. These template files are written using the Mako template language from the Mako library for Python [12].

At the top level a simulator template package is simply a directory of several files where the directory name is the `internal_name` of the simulator it is implementing, and each file is either a .template file for a device model or device reference, or it is a .sim_directive indicating that it is used by simulator.py for properly running simulations. For each device defined in devices.py there are generally at least two .template files; one file with the filename format of [device]_reference.template and another with the format of [device]_model.template. The reference files are used when instantiating the device and can be called upon multiple times, while the model file is generally only called upon once per simulation and is used to inform the simulator how a non-native device should behave. For example given a SPICE netlist such as:

```
Line1: * A contrived sample circuit
Line2: R1 1 2 1k
Line3: R2 2 0 2k
Line4: X1 1 0 sample
Line5: .subckt sample in out
Line6: V1 in out 5
Line7: .ends
Line8: .end
```

Lines 2-3 are what would result from calling on the resistor_reference.template file while line 4 is the result of a call to sample_reference.template. Lines 5-7 are used to generate the subcircuit model and are thus the result of a call to sample _model.template. In addition to the device templates there are simulator-specific file templates that are expected as well, in the above example line 1 would be generated by the header.template file and line 8 would be generated by the footer.template file. Fig. 23 shown below details the process of converting device.py objects into SPICE and Spectre netlists.

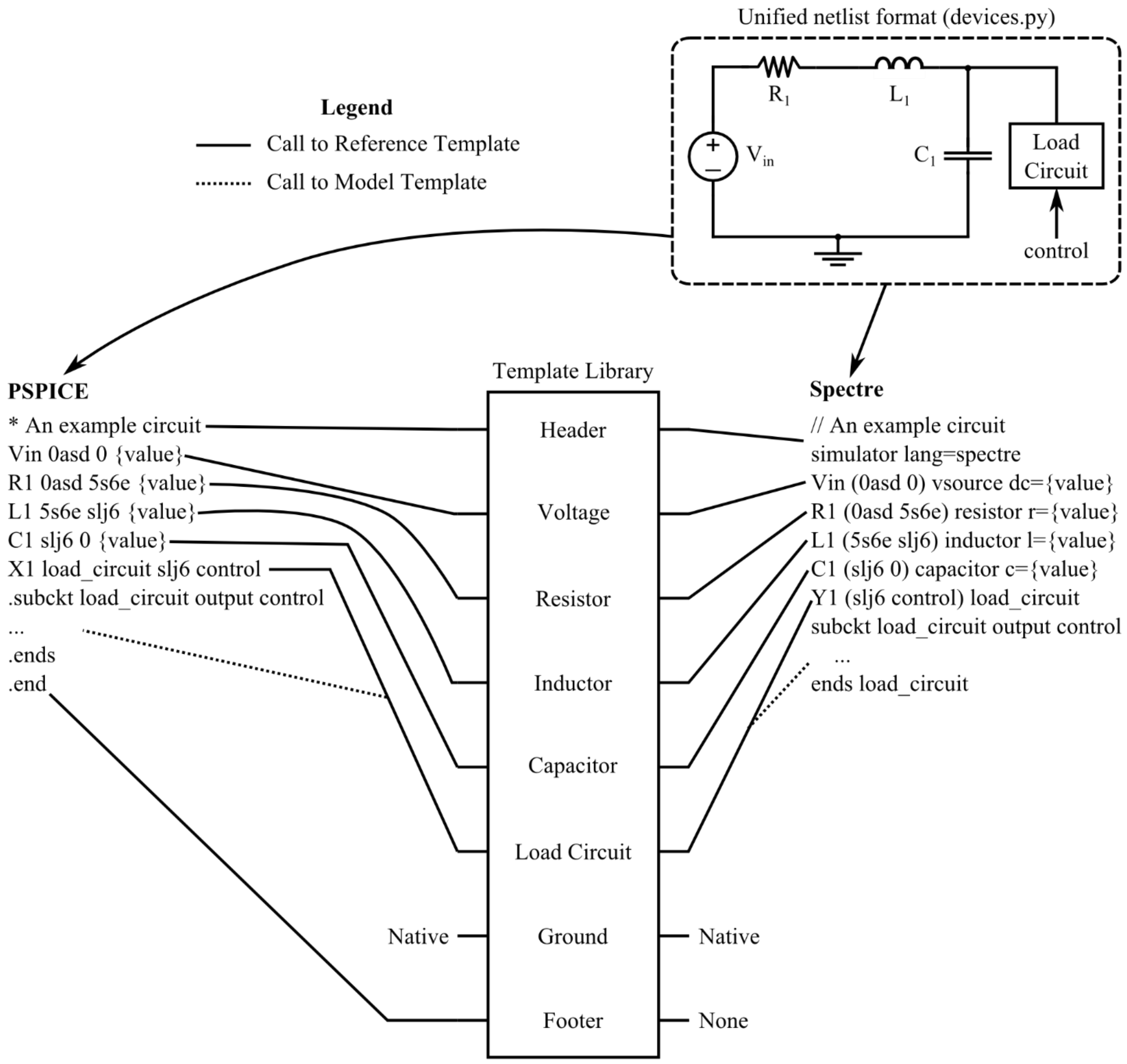
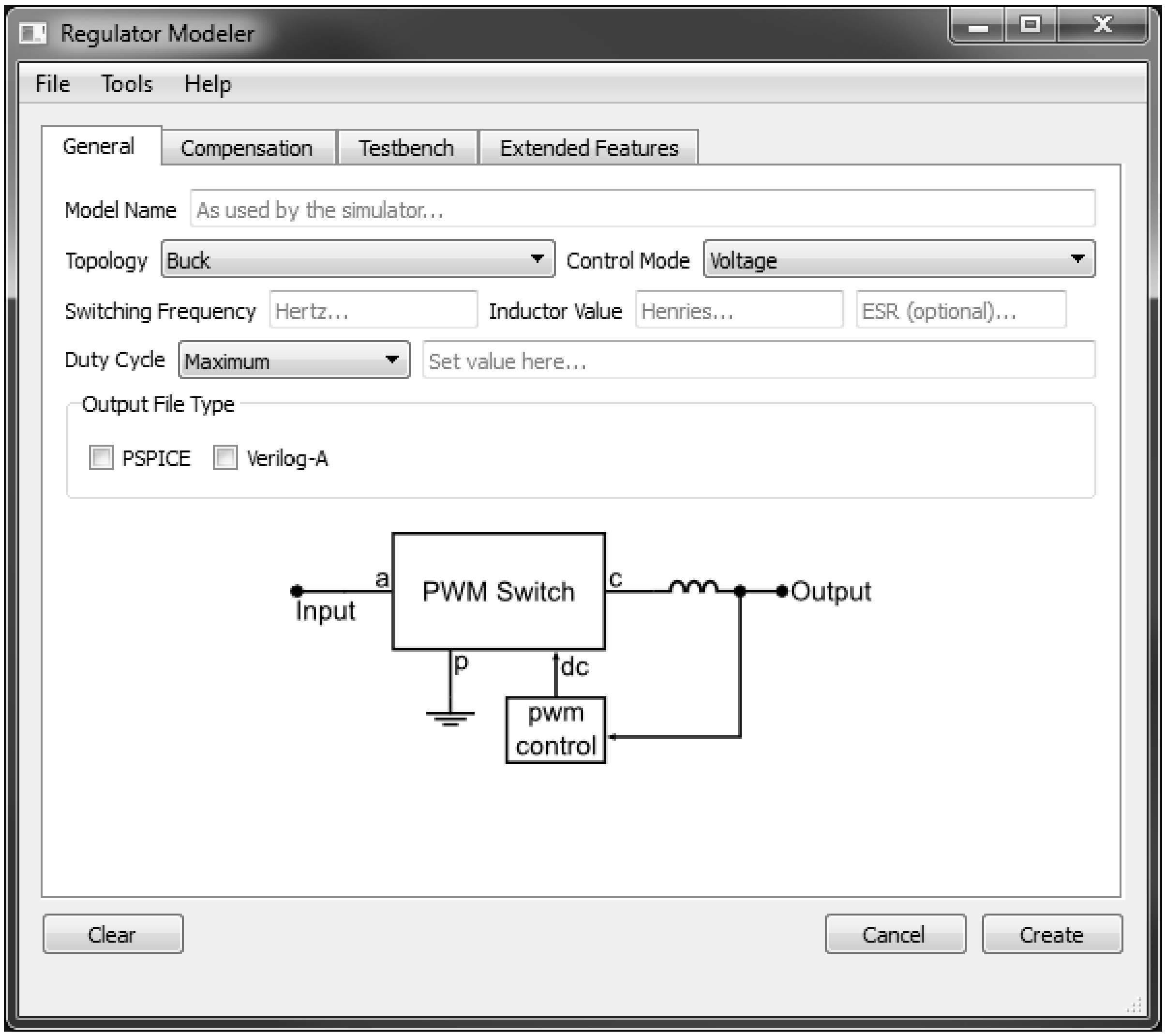


Fig. 23. Composite image showing the usage of the simulator template library.

Device types which can have multiple permutations may require more than just two template files; for example the compensation networks require a separate template file for each type of compensation network identified in Table 1. In addition to these device templates another class of file is used to inform the modeling tool how to direct the simulator to perform each type of simulation, these files have a `.sim_directive` file extension and use the same template method as `.template` files.

3.2 Graphical User Interface

The graphical user interface (GUI) has been developed in order to provide easier access to the functionality of the tool that would otherwise be difficult to perform using the command line alone. The remainder of this section discusses the main features provided by the GUI and how it is used when generating a model. The first interface the user sees upon opening the tool is shown in Fig. 24.



Model Name: As used by the simulator...

Topology: Buck

Control Mode: Voltage

Switching Frequency: Hertz...

Inductor Value: Henries... ESR (optional)...

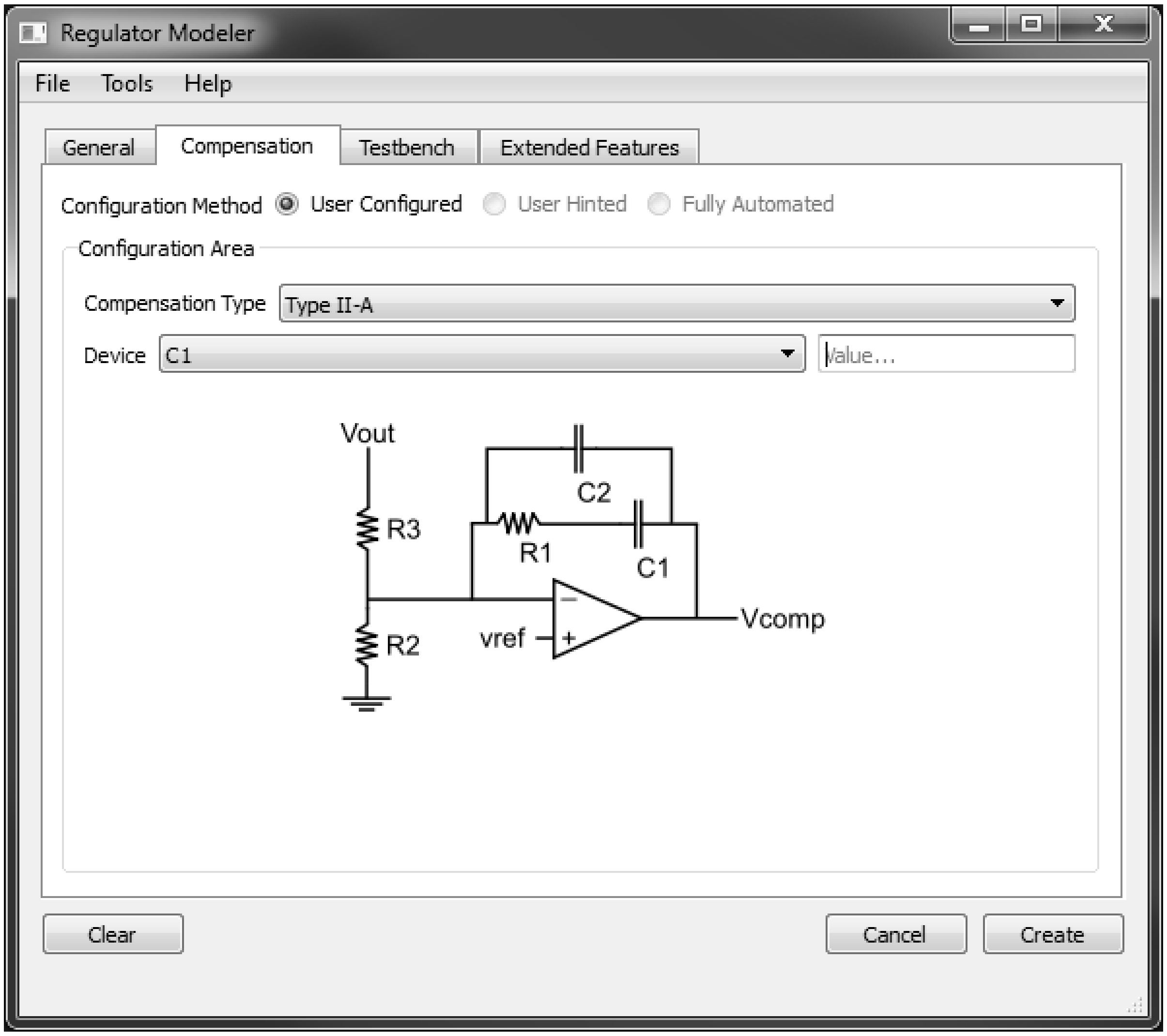
Duty Cycle: Maximum Set value here...

Output File Type

☐ PSPICE ☐ Verilog-A

Fig. 24. Main tab in the modeling tool GUI, collects general information about the regulator.

This is the first screen shown and is used to collect basic information about the regulator such as name of the model as it will be referred to in the simulator, topology (buck, boost, buck-boost), control mode type (voltage, current-peak, current-average), PWM controller switching frequency, and duty-cycle information. Below all of the data entry is an image viewer which provides a dynamic preview of the selected regulator topology. The next step is to collect information about the compensation network as shown in Fig. 25.



Configuration Method [✓ User Configured] User Hinted Fully Automated

Configuration Area

Compensation Type: Type II-A

Device: C1

Value...

Fig. 25. Compensation network tab for regulator modeling tool.

Starting from the top of this screenshot is the first visible feature of selecting a model optimization method. Three methods are presented and are implemented as follows:

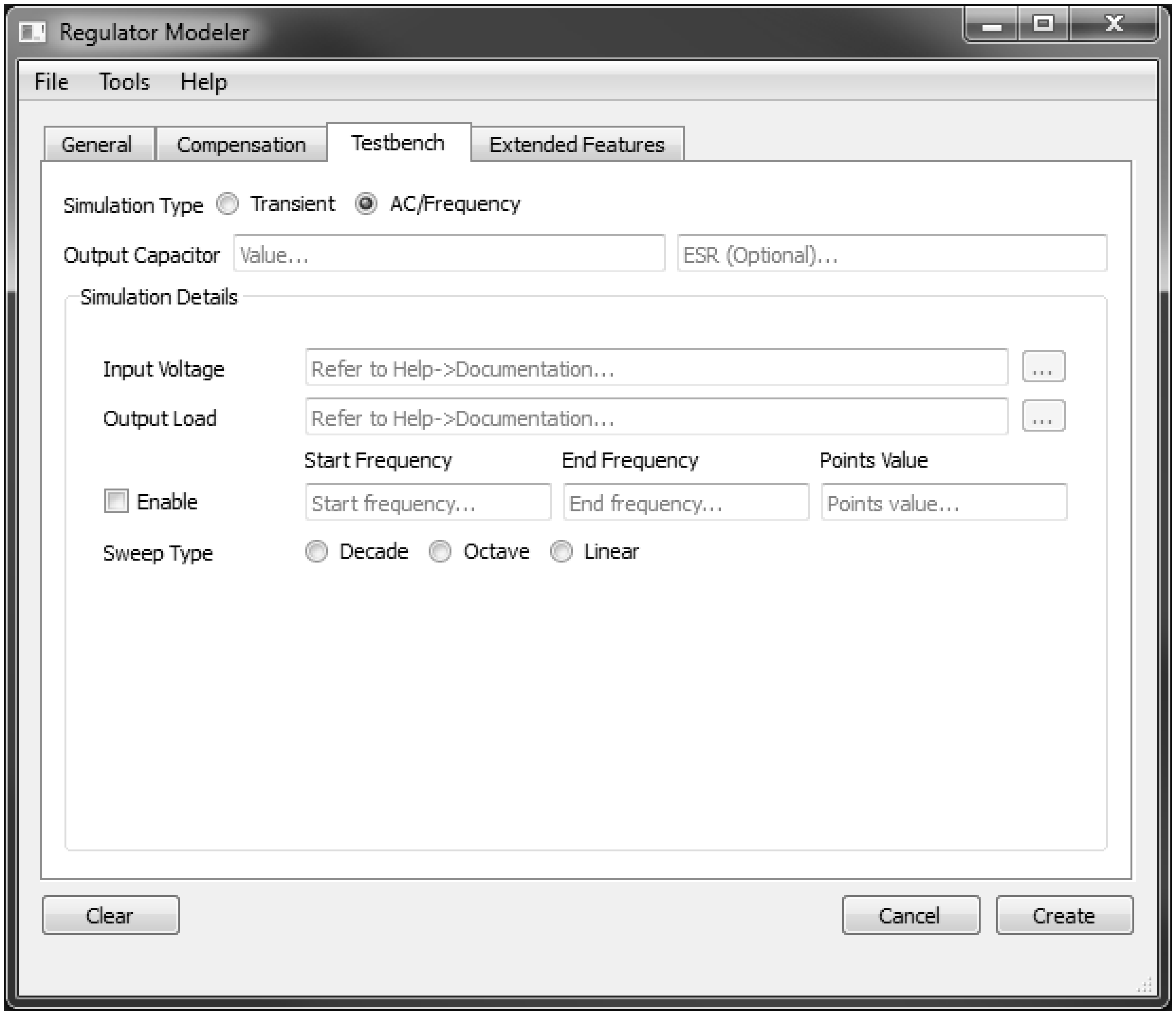
- **User Specified ("User Configured" in image)** – A fully manual model generation method. In this mode the tool is essentially a specialized template generation tool.
- **Constrained Optimization ("User Hinted" in image)** – Implements the interface for using the constrained optimization modeling approach previously described.

- **Fully Automated** – The method for just importing a dataset, entering a minimal set of information, and generating a simulation model. This is disabled since the necessary identification functions have not been implemented.

The area labeled *Configuration Area* contains a `QWidget`[2] that is independent from the rest of the interface and changes depending on which configuration method is selected. The user specified method is currently shown. In this mode the compensation network type is first selected and then the values of the devices are set below that. The image viewer shown below the form is updated when the compensation network type changes. In constrained optimization mode the interface is very similar with the addition of a few interface elements to allow the user to control which values are parameterized, what ranges are allowed, and how big of a step the parametric analysis uses for each value. Since the fully automated modeling method is not yet implemented there is no interface developed.

The final configuration tab is for setting up the testbench in order to perform measurements on the developed models. The different types of testbenches available have previously been described in the *Transient and AC Measurements of Switching Regulators* section. The default state for this tab is shown below in Fig. 26.

[2] A QWidget is the basic graphical unit in Qt, and is the base class for most graphical elements. It can be used independently or as a container for other elements. For example, a QPushButton is a subclass of QWidget but several QPushButtons can be grouped into a QGroupBox which is also a subclass of QWidget.



Simulation Type Transient [✓ AC/Frequency]

Output Capacitor Value... ESR (Optional)...

Simulation Details

Input Voltage Refer to Help->Documentation...

Output Load Refer to Help->Documentation...

	Start Frequency	End Frequency	Points Value
Enable	Start frequency...	End frequency...	Points value...

Sweep Type Decade Octave Linear

Fig. 26. Testbench definition tab showing options for an AC sweep testbench.

This screenshot presents the available options for a switching regulator AC simulation. The default AC test is the loop transfer test since this is much more common than the plant transfer test. The only identified drawback to this interface as it is currently implemented is that the input voltage and output load must be specified through a specific syntax that is defined in the source code comments. This makes it more difficult than necessary for the user to create simulation ready models, though the problem could be easily addressed with some UI improvements.

Finally, the simulator connection pane is presented below in Fig. 27.

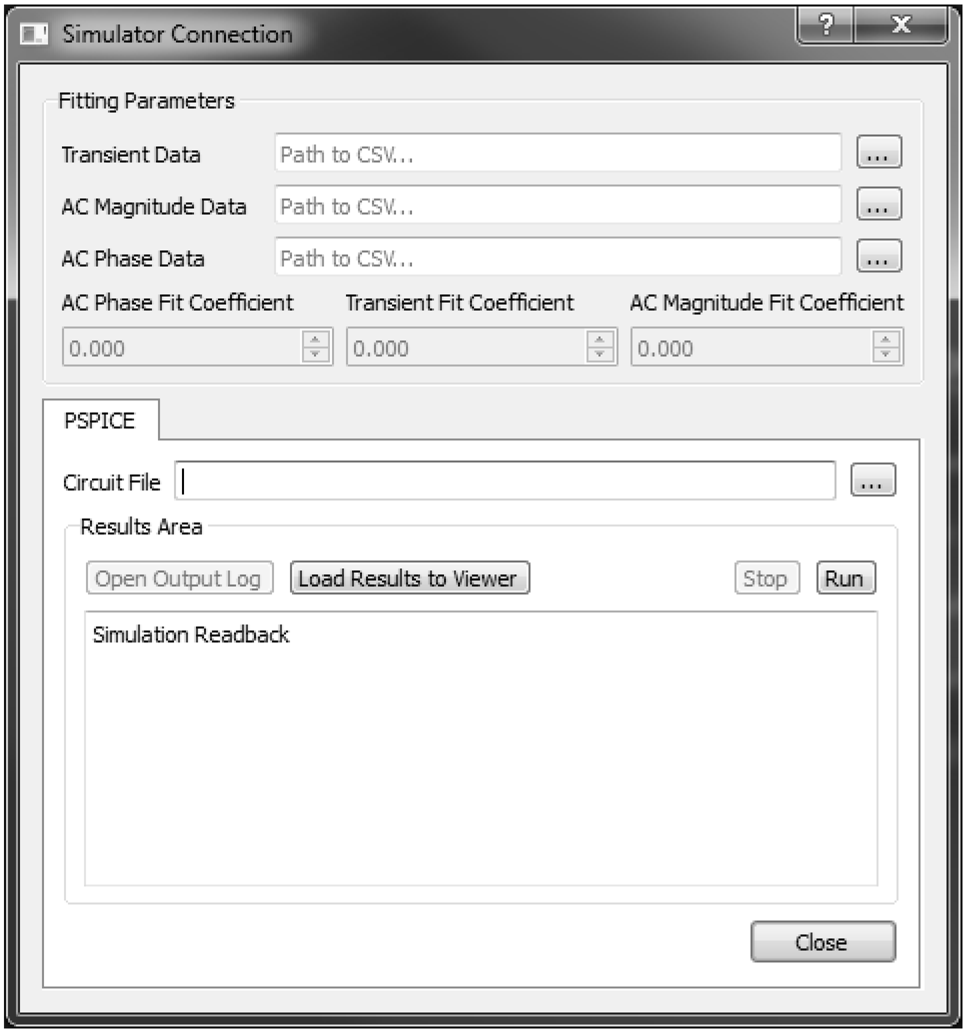


Fig. 27. Initial version of simulator connection pane. Provides an interface for fitting compensation network parameters to supplied data.

The simulator connection version presented here is an older concept designed for direct interaction with the simulator without leaving the modeling tool. A new module is currently being implemented which more fully abstracts the simulator and will make it easier to implement the constrained optimization and fully automated model generation methods.

4. RESULTS

Now that a full understanding has been developed towards learning what switching regulators do, how they are modeled, and how the model generation process and tool works, the results from this work are presented for further comment and analysis. This section presents the results of applying the constrained optimization model generation method to the datasheet measurement data of three switching regulators and compares these results. The parts analyzed include:

- **Texas Instruments TPS54320** – A current mode controlled buck converter with fully-integrated high-side FET.
- **Texas Instruments TPS54350** – A voltage mode controlled buck converter with integrated high-side FET and external low-side FET.
- **Freescale MC34713** – A voltage mode controlled buck converter with fully integrated high-side FET.

As will be described further in the following sections the data presented lends more validation to the PWM modeling method and validates the constrained optimization model generation method as applied to the PWM model.

4.1 Texas Instruments TPS54320

The TPS54320 is a current mode controlled buck converter with an integrated switching FET, the device is capable of handling an input voltage of 4.5 V to 17 V input and can supply an output current of up to 3 A continuously. The switching frequency on the device is adjustable from 200 kHz to 1.2 MHz and it includes a collection of extra features such as slow-start, overcurrent protection, and enable/disable capability.

For the measured evaluation module the switching frequency was set at 480 kHz, the input voltage was designed for 12 V regulated down to 3.3 V with the capability of maintaining a constant output load of 3 A. The compensation network selected for this part is a type 3-A network and the output filter components selected were a low ESR 6.8 µH inductor and a low ESR 47 µF capacitor. The schematic for this evaluation module is shown below in Fig. 28.

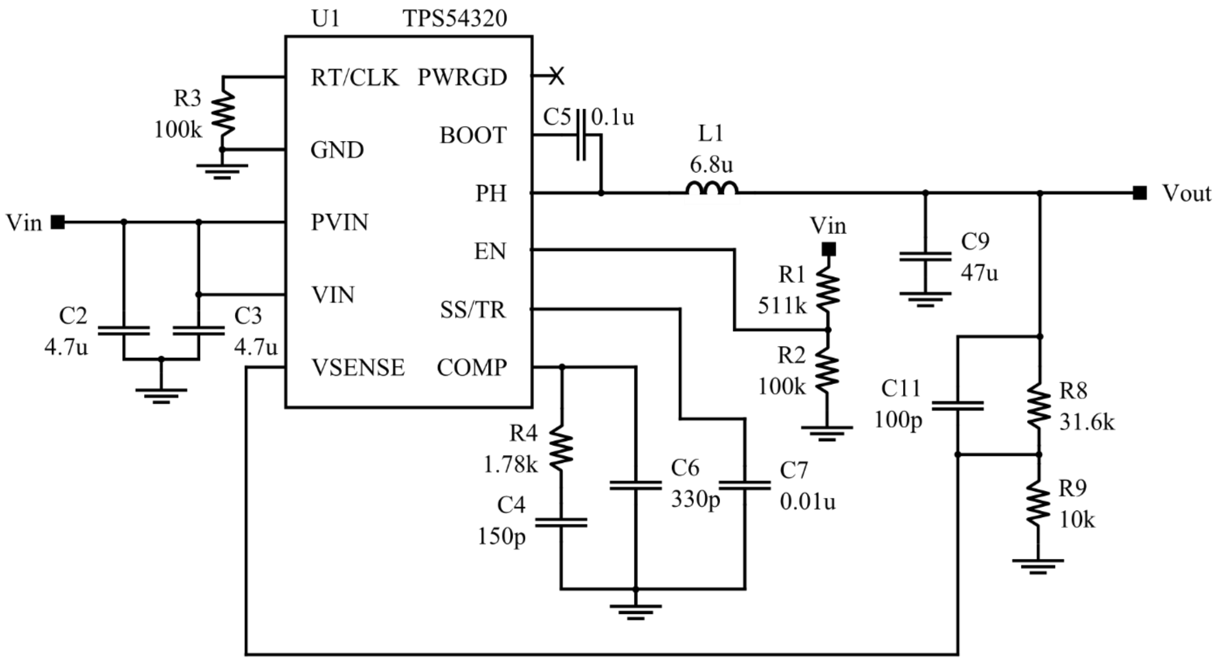


Fig. 28. Texas Instruments TPS54320 evaluation module schematic.

In order to verify the model generation method the AC and transient data from the TPS54320 datasheet was digitized and used by the tool in constrained optimization mode. This optimization was performed in two steps. Initially all of the compensation network component parameters were parameterized with relatively narrow ranges of variation and small step sizes. After selecting the best result from this sweep the compensation network component values were held steady while the output components and their respective ESRs were parameterized and swept.

The result from this optimization is shown in Fig. 29 below in comparison to the digitized datasheet information.

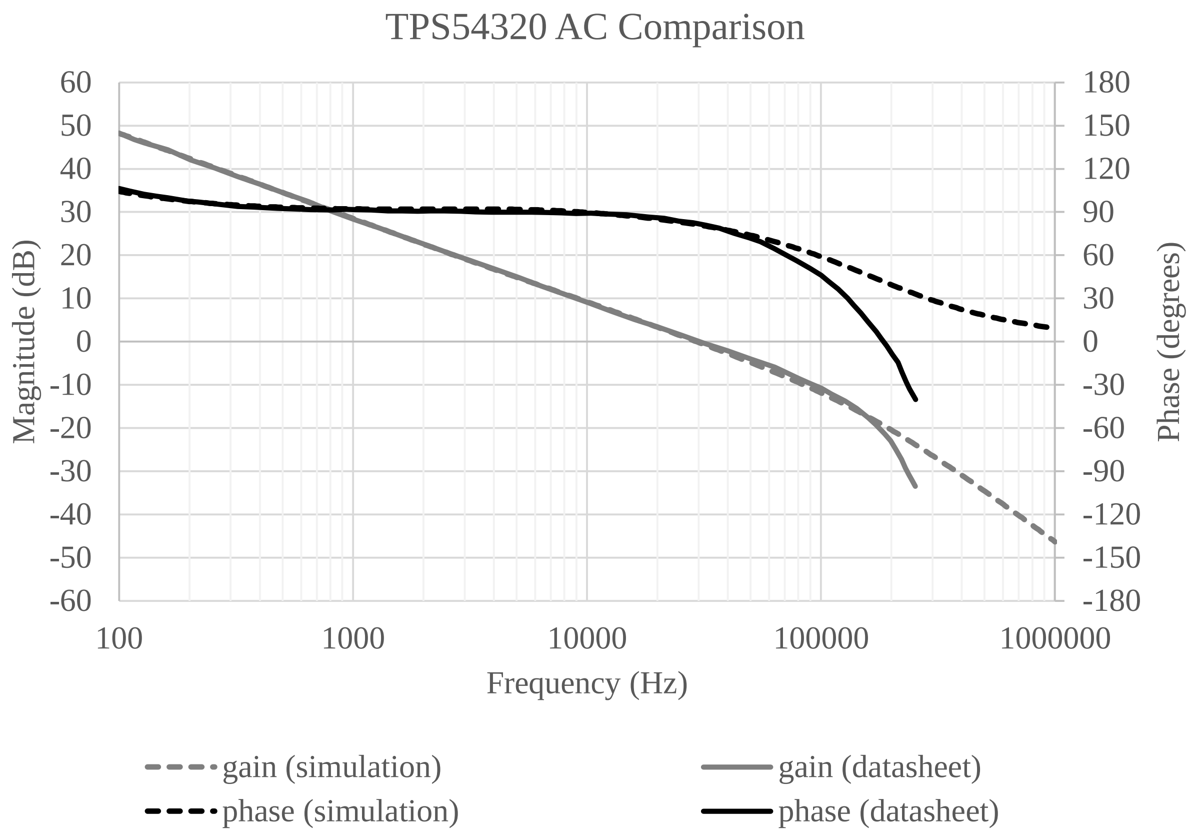


Fig. 29. AC gain and phase comparison for the Texas Instruments TPS54320.

This regulator shows a fairly uninteresting AC response with little variation over a 1MHz range. The simulation model shows close agreement with the measured data until the phase begins to diverge around 60 kHz and the gain begins to diverge around 200 kHz. It is common behavior in switching regulators that as the AC stimulus approaches $\frac{1}{2} f_{sw}$ the device performance will begin to decline from the optimal and it is clear that the model used does not capture this non-ideality. It is also possible that the divergence seen in the datasheet data is from the measurement equipment or test set-up but that explanation is less likely. Overall this is considered an excellent fit. Next, the AC optimized model is converted to a transient testbench and compared to the datasheet transient measurements, the results of this comparison are shown in Fig. 30.

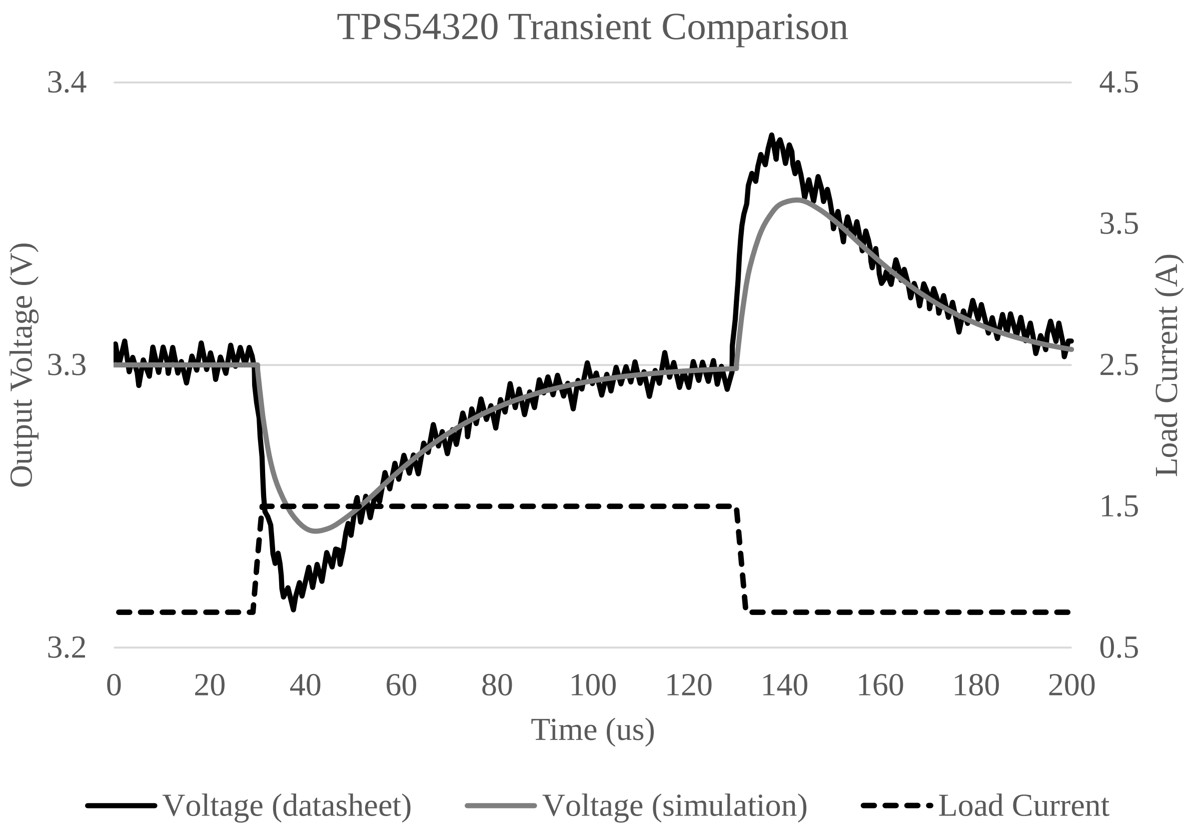


Fig. 30. Transient comparison for the Texas Instruments TPS54320.

The results shown above come from directly transferring the AC optimized model into a transient simulation. The test procedure was to subject the regulator to a 0.75 A initial load, pulse the load up to 1.5 A for 100 µs, then bring it back down to 0.75 A and observe the regulator response. The model shows excellent agreement through the relatively steady sections of the graph with a small non-optimal response in the pulse peaks. The model does not exhibit the same intensity of overshoot and undershoot, but the small non-ideality is considered acceptable in exchange for a practically negligible simulation time.

4.2 Texas Instruments TPS54350

The TPS54350 is a voltage mode controlled buck converter with one integrated and one external switching FET. The device is capable of handling an input voltage of 4.5 V to 20 V input and can supply an output current of up to 3 A continuously or 4.5 A briefly. The switching frequency on the device is adjustable from 250 kHz to 700 kHz and it includes extra features of slow-start, power-good signal, and enable/disable capability, among others.

The evaluation module design set the switching frequency at 500 kHz with an input voltage of 12 V regulated down to 3.3 V on the output and the capability of maintaining a constant output load of 3 A. The compensation network selected for this part is a type 4-A network and the output filter components selected were a low ESR 10 µH inductor and a low ESR 100 µF capacitor. The schematic for this evaluation module is shown below in Fig. 31

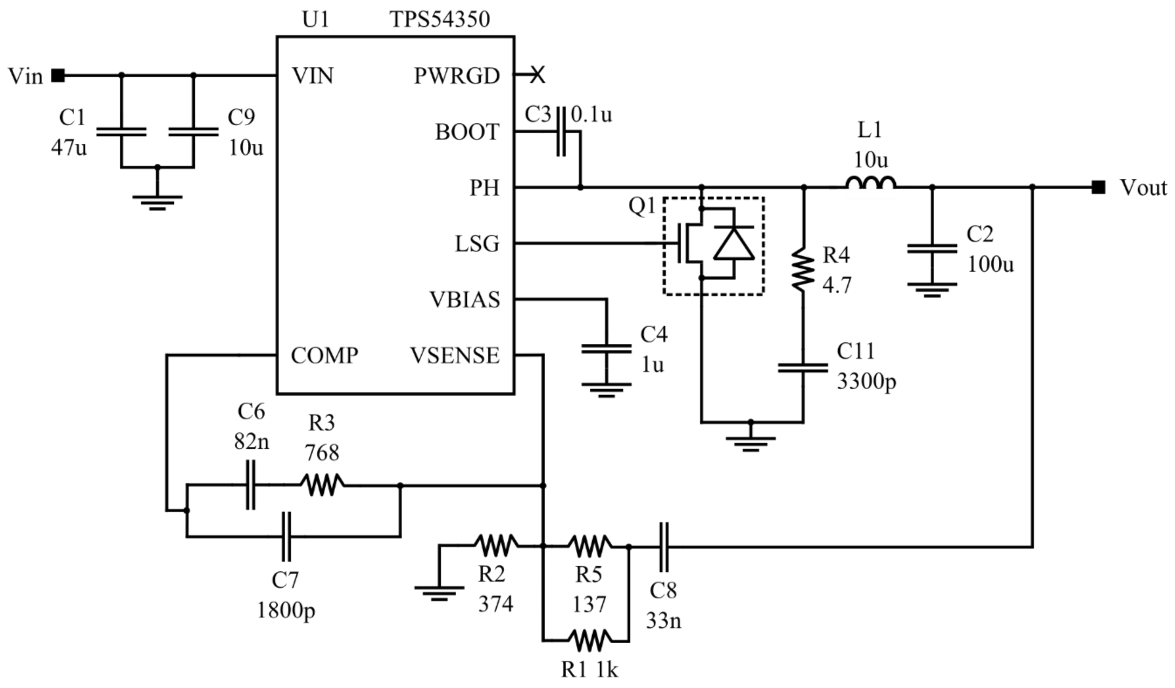


Fig. 31. Texas Instruments TPS54350 evaluation module schematic.

Similar to the TPS54320, the TPS54350 datasheet measurements were digitized and used by the tool in constrained optimization mode. For this model the optimization was performed in a single sweep, with only the R1, R5, C8, and R2 from the compensation network and C2 and L1 from the output filter being parameterized. Results from this optimization are shown in Fig. 32 below in comparison to the digitized datasheet information.

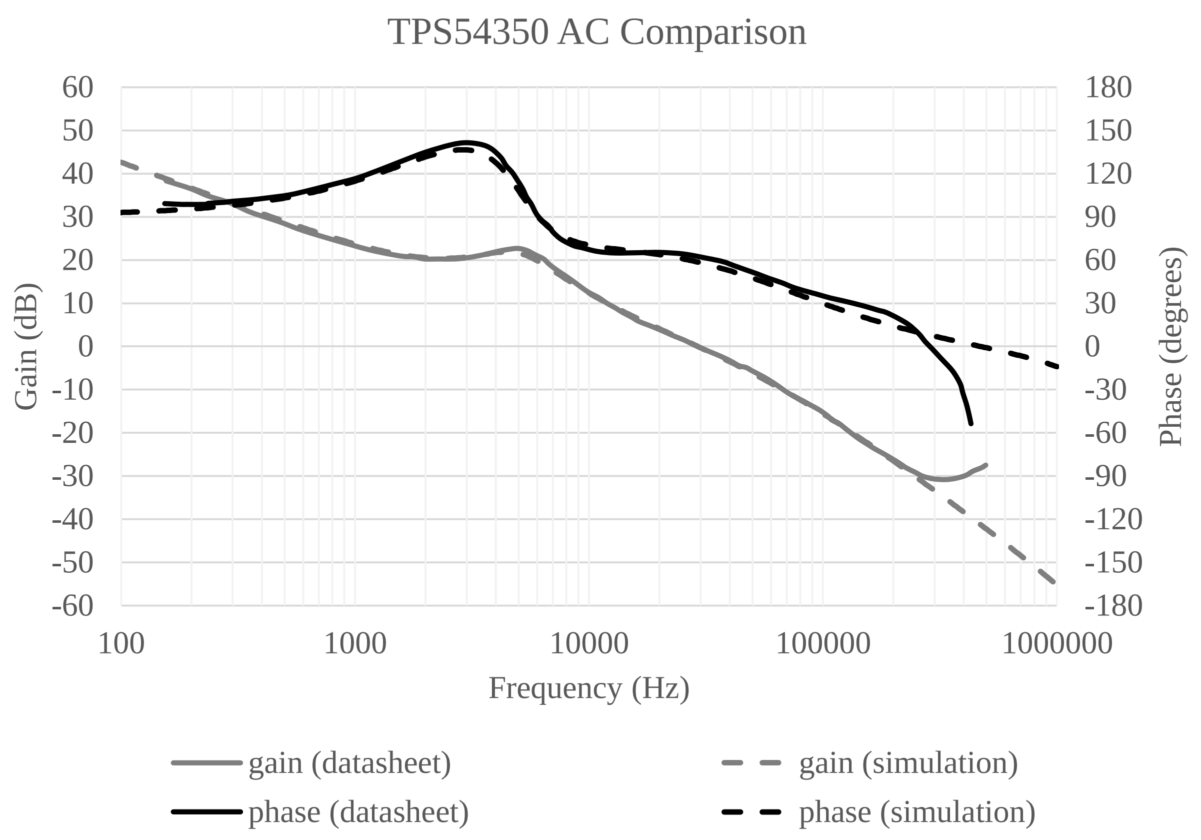


Fig. 32. AC gain and phase comparison for the Texas Instruments TPS54350.

The TPS54350 AC response proves more interesting than the TPS54320 AC response and the simulation model appears to be just as accurate in this case. The model and measured data begin to diverge in phase around the 300 kHz point while the gain begins to diverge around 200 kHz. As previously mentioned, this is expected behavior near the $\frac{1}{2}f_{sw}$ point. Interestingly the phase and gain diverge in opposing directions in this regulator suggesting that the OTA in the compensation network is the weak point in the system as it is likely entering into an inverting behavior. This simulation shows close agreement with the measured data.

Unlike the previous test results, for this modeling test an attempt was made to use a separate optimization process for the transient simulation model, as this process was not yet implemented into the tool it was done by hand and thus may not be a perfect implementation. The results of this optimization and simulation are shown in Fig. 33.

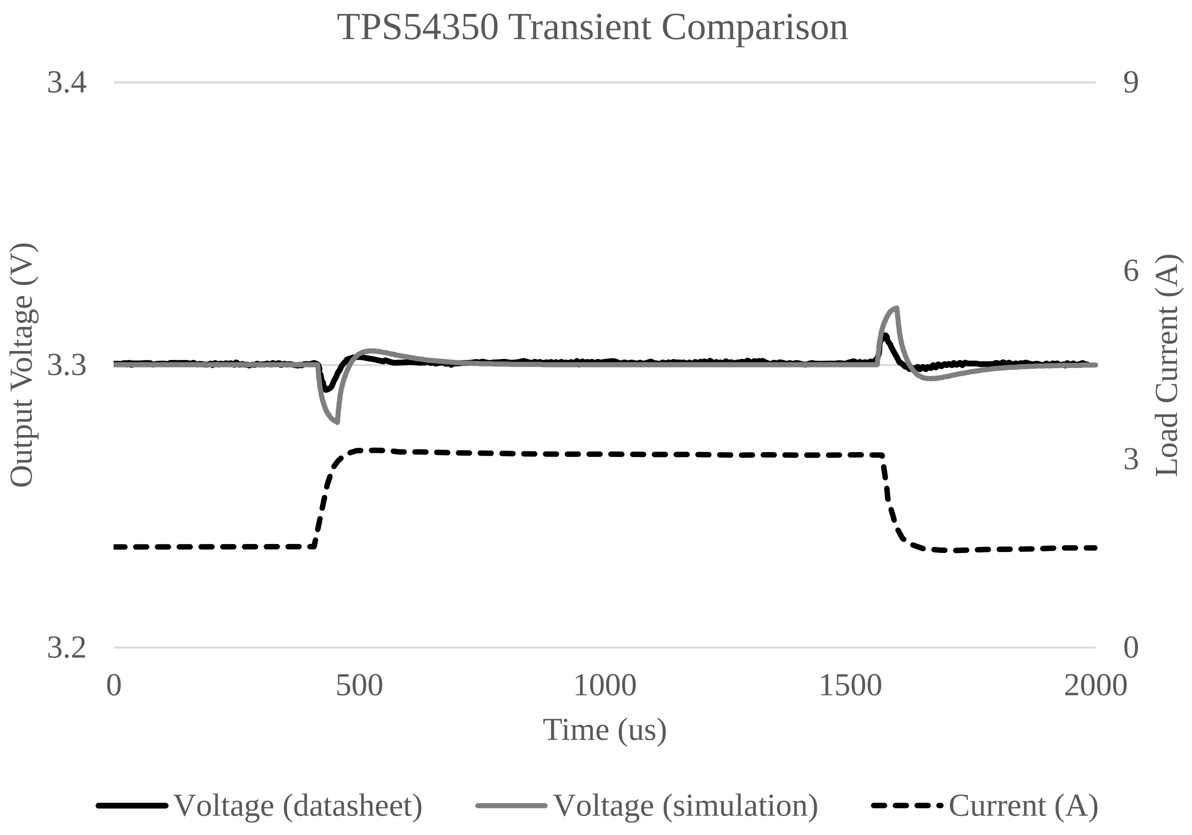


Fig. 33. Transient Comparison for the Texas Instruments TPS54350.

The first feature worth noting in this dataset is the significantly subdued response to a load change when compared with the TPS54320. Where the TPS54320 had a variation of 0.1 V from the mean, this regulator shows a variation closer to 0.025 V from the mean. Much like the previous simulation though, while the model agrees with the measured data in the steady state it is only a close approximation during the pulses. Considering the significant increase in simulation speed such a small mismatch in responses is likely not significant.

4.3 Freescale MC34713

The MC34713 is a voltage mode controlled buck converter with an integrated switching FET, the device is capable of handling an input voltage of 3 V to 6 V input and can supply an output current of up to 5 A continuously or 6.5 A briefly, it has an output voltage range of 0.7 V to 3.6 V. The switching frequency on the device is adjustable from 200 kHz to 1 MHz and it includes a collection of extra features such as slow-start, power-good, overcurrent and short-circuit protection, and a shutdown signal.

The evaluation module for this part established the switching frequency at 1 MHz, the input voltage of 5 V is regulated down to 3.3 V on the output with the capability of maintaining a constant load of 5 A. The compensation network selected for this part is a type 4-A network and the output filter components were chosen to be a low ESR 1.5 µH inductor and three low ESR 100 µF capacitors. The schematic for this device is shown below in Fig. 31

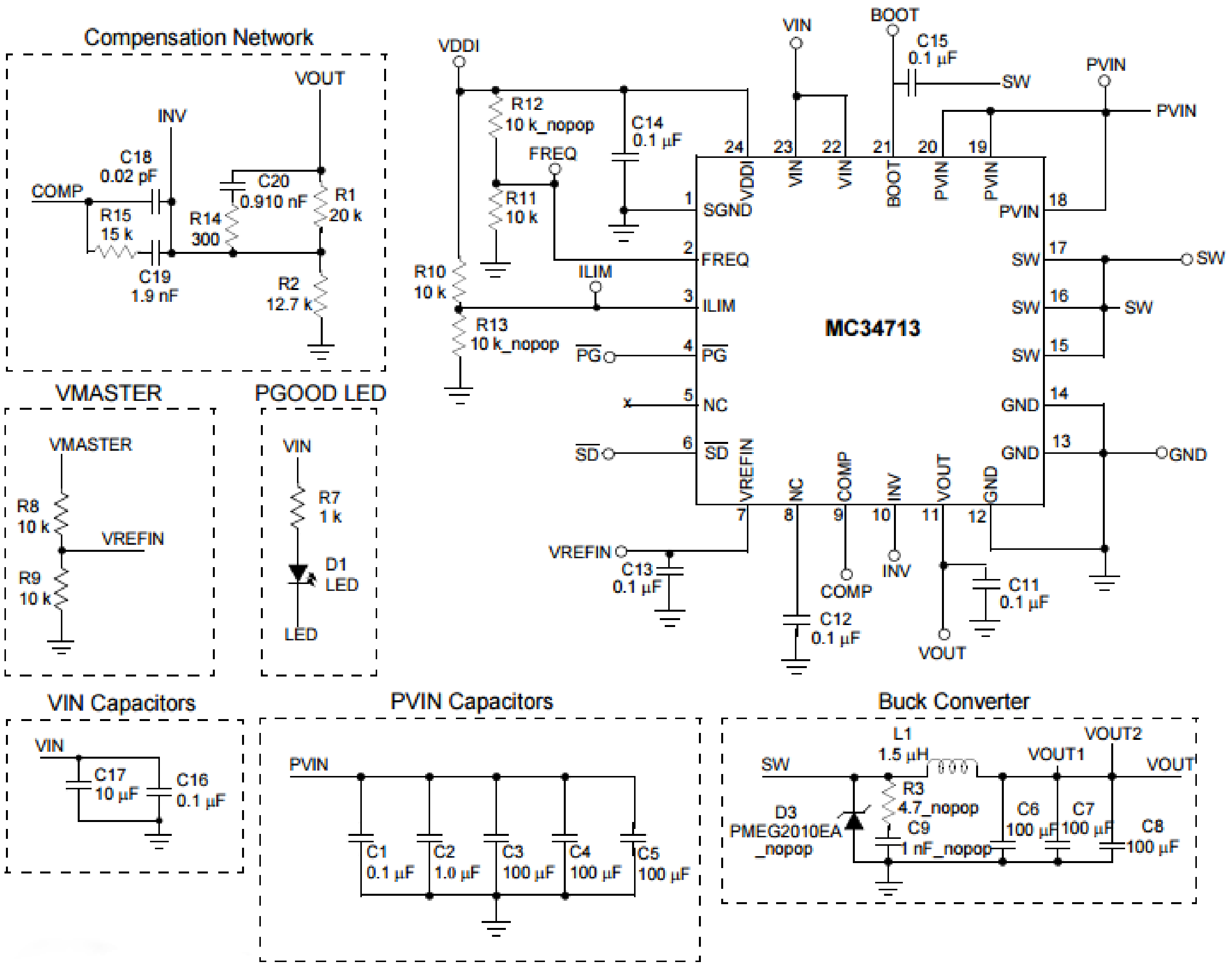


Fig. 34. Freescale MC34713 switching regulator evaluation module schematic.

For the MC34713 the optimization routine was also performed in a single sweep. Despite being shown as several separate devices the split PVIN capacitors were added together and simulated as a single device, as were the VOUT capacitors. This arrangement results in far fewer parameters to sweep through. The AC results from this optimization are shown below in comparison to the digitized datasheet information.

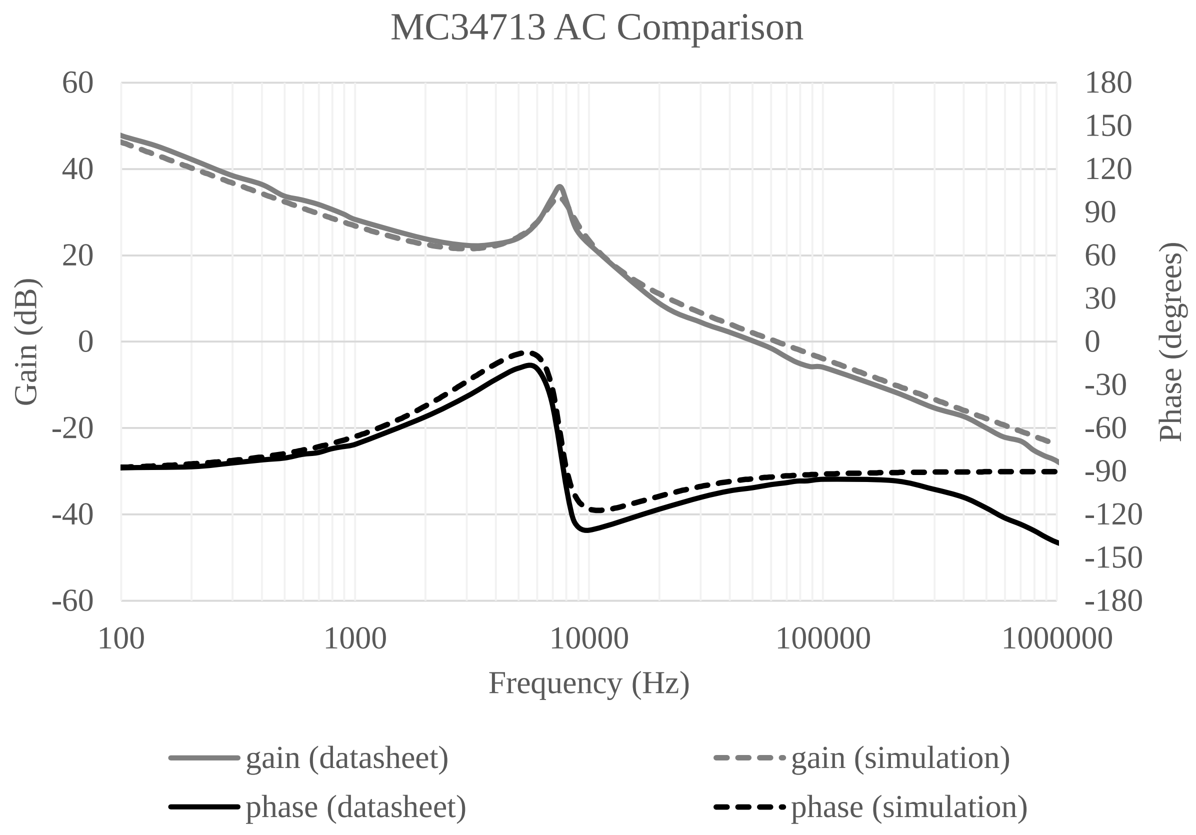


Fig. 35. AC phase and gain comparison results for the Freescale MC34713.

This simulation shows close agreement with the measured data. Most importantly this comparison does not show any significant divergence in the gain plot, and only begins to diverge on the phase plot around 250 kHz. This improved range of agreement between model and measurement lend further support to the suggestion that the earlier models diverged due to non-idealities that begin to occur near $\frac{1}{2} f_{sw}$ since the regulator is operating at ~2x f_{sw} of the previously examined regulators. The datasheet for this part did not provide transient data to compare against and therefore no transient comparison is shown.

5. CONCLUSIONS AND FUTURE WORK

This thesis has described the development and usage of a tool that makes the creation of switching regulator behavioral models easy and fast. The main contribution of this work is how the PWM average switch modeling method can be automated through the use of constrained optimization. In addition to the implemented constrained optimization method, a fully automated model generation method was developed and described.

The results from model-to-real-world comparisons proved that the proposed model generation method is highly effective at capturing the ideal and first-order non-idealities in a switching regulator for AC responses and that this AC optimized model transfers sufficiently well to the transient domain. In order to provide a roadmap for future work to build on, some suggestions are provided for what should be implemented differently in the future or what otherwise could be improved.

First, it is suggested that other modeling methods be investigated to be used in place of the PWM average switch method or as an alternative. Namely the state-space averaging method is suggested for further consideration. The PWM method is inherently attractive for a number of reasons; it is simple, easy to apply, and works well under ideal conditions. Unfortunately the PWM method makes it difficult to deal with non-idealities such as ESR and ESL on output filter components, dual switching component configurations, and other internal non-idealities as observed in the *Results* section. When using the PWM method each non-ideality must be addressed in the model separately, this can rapidly increase model complexity and complicates the automation procedure. The state-space modeling method on the other hand is designed such that it considers each state of the switching regulator as an independent mathematical system with the non-ideal behaviors inherently considered in the model. While this mathematical basis may make the method more intimidating or difficult to implement for the average modeler it would likely lend itself well to the already developed constrained optimization method for model generation. One drawback to this suggested improvement is that if another modeling method were selected to replace the PWM method it would render much of the existing tool useless since it is largely developed around the PWM method.

Next, it is suggested that the separation between the GUI and the Python core be removed and the two designed to work more closely together. The modularity of separating the GUI from the Python core may have been a wise decision in a multi-developer environment where one developer could proceed with advancing the GUI and the other proceeds with advancing the Python core. Unfortunately, this separation just increases overhead and makes it more difficult to move forward in a single-developer environment. In addition to the decrease in development complexity a more tightly integrated GUI would make it possible to have a more dynamic UI and perform advanced operations such as running multiple simulations simultaneously across different simulators or dynamically adding new topologies or devices without recompiling the GUI. This suggested improvement is currently in-progress as it is considered a necessity for properly implementing the constrained optimization and fully automated model generation methods.

For the final suggestion on tool improvements, it is concerning that the `simulator.py` module and `devices.py` module are not as modularized as the other parts of the program. If the user wanted to add more device types to the tool they would have to open up the source code and modify these files directly. Ideally the `simulator.py` and `devices.py` modules would be implemented in a Python *package* where each device type is separate module. The lack of modularity in these two modules is not a significant concern but this could be a simple point of improvement.

Finally, a potential spin-off project is proposed that could add value to the field of modeling and simulation in electrical engineering. The suggested project is to use portions of the Python core to create a generic netlist generator and simulation interface tool. This could feasibly be implemented in only a few weeks based off of the previously completed work on the switching regulator modeling tool. Such a project could prove indispensable to engineers that commonly work in multiple simulation environments or would like to verify their designs across multiple simulators. This spin-off project would additionally make a good addition to the open-source community and could be hosted by the University of Arkansas or personally by the author.

6. REFERENCES

[1] S. Ang and A. Oliva, Power-Switching Converters, Boca Raton, FL: CRC Press, 2011.

[2] C. Basso, Switch-Mode Power Supplies: Simulations and Practical Designs, New York: McGraw-Hill Professional, 2008.

[3] C. Slobadan and R. Middlebrook, "A general unified approach to modelling switching-converter power stages," in *Power Electronics Specialists Conference, IEEE Proceedings of*, Cleveland, OH, 1976.

[4] V. Vorpèrian, "Simplified analysis of PWM converters using model of PWM switch. Part I: Continuous conduction mode," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 26, no. 3, pp. 490-496, May 1990.

[5] V. Vorpèrian, "Simplified analysis of PWM converters using model of PWM switch. Part II: Discontinuous conduction mode," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 26, no. 3, pp. 497 - 505, 1990.

[6] D. Maksimovic and R. Zane, "Small-Signal Discrete-Time Modeling of Digitally Controlled PWM Converters," *Power Electronics, IEEE Trans. on,* vol. 22, no. 6, pp. 2552-2556, 2007.

[7] L. Arnedo, R. Burgos, F. Wang and D. Boroyevich, "Black-Box Terminal Characterization Modeling of DC-to-DC Converters," in *Applied Power Electronics Conference (APEC),* Anaheim, CA, 2007.

[8] Texas Instruments, "Texas Instruments, Inc.," April 2013. [Online]. Available: http://www.ti.com/lit/an/snva364a/snva364a.pdf. [Accessed 22 April 2015].

[9] J. Huwaldt, "Plot Digitizer Home Page," sourceforge.net, 18 April 2014. [Online]. Available: http://plotdigitizer.sourceforge.net/. [Accessed 14 March 2015].

[10] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Indianapolis, IN: Pearson Education, 1995.

[11] Internet Engineering Task Force: Network Working Group, "RFC 4122," July 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4122.txt. [Accessed 18 March 2015].

[12] M. Bayer, "Mako Templates," April 2012. [Online]. Available: http://www.makotemplates.org/. [Accessed 20 April 2015].

[13] K. Beck, Test Driven Development: By Example, Indianapolis, IN: Addisonn-Wesley Professional, 2002.

[14] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Computing Surveys*, vol. 21, no. 3, pp. 359-411, 1989.

[15] J. Morgan, "Don't Be Scared of Functional Programming," Smashing Magazine, 2 July

2014. [Online]. Available: http://www.smashingmagazine.com/2014/07/02/dont-be-scared-of-functional-programming/. [Accessed 3 March 2015].

[16] J. Williams, "Load Transient Response Testing for Voltage Regulators: Practical Considerations for Testing and Evaluating Results," October 2006. [Online]. Available: www.linear.com/docs/29876. [Accessed 12 February 2015].

7. APPENDIX A: BRIEF NOTE ON DEVELOPMENT PHILOSOPHY

In an effort to produce the best code possible in the shortest amount of time this software was developed with the following best-practices guidelines:

1. Test-driven development (TDD) leads to less unexpected bugs and makes identified bugs easier to track down. These two advantages combine to result in much less time spent debugging [13].
2. There is no such thing as too much documentation.
3. Functional programming is superior to object-oriented at minimizing the amount of *state complexity* in a program. The functional paradigm is especially useful when performing complex data processing or operating with concepts that do not rely on state.
4. Object-oriented programming is best suited when maintaining *state information* is a necessity [10].

The two former suggestions on this list are intended to help the developer confidently and quickly write code that is easily improved in the future. While TDD taxes the developer with a large initial investment in designing and implementing test suites it pays dividends when implementing new functionality or improving existing code, this is true for two primary reasons:

1. With a carefully crafted test suite, the developer can stop coding as soon as all the tests pass.
2. When improving existing code the developer can make sweeping changes without fear of breakage as long as all of the tests still pass when the changes are finalized.

The suggestion that more documentation is better is meant to ease the burden on incoming developers looking to improve upon code, though it can also be useful for other reasons. The latter two suggestions are targeted more at helping the developer decide on the right tool for the job.

In addition to these general suggestions, strict adherence to the Python PEP8 style guidelines was enforced. This adherence was assured through the use of JetBrains PyCharm editor which has an always-on PEP8 verification tool built in. Adherence to PEP8 ensures code is consistent throughout the program and as consistent as possible with commonly accepted conventions in the Python community.