OBJECT-ORIENTED POWER SYSTEM MODELING USING THE DYMOLA MODELING LANGUAGE

John S. Glaser, François E. Cellier, and Arthur F. Witulski
Department of Electrical and Computer Engineering
University of Arizona, Tucson AZ 85721
Email: glaser@hermes.ece.arizona.edu, cellier@ece.arizona.edu, witulski@ece.arizona.edu

ABSTRACT

The general purpose modeling language Dymola is used in conjunction with the ACSL simulator to provide an environment for the simulation of power electronic systems. The object-oriented nature of Dymola, along with its high-level handling of discrete events, makes it ideally suited to modeling switching converters and complete systems of several converters. Converter models can be constructed using a netlist format similar to SPICE. These models can then be used as components in more complex systems. This is demonstrated by example with a regulated dc-dc converter system.

I. INTRODUCTION

Simulation of switching power converters is a problematic task for several reasons. Such converters operate by using switches to change the configuration of an energy-storage (inductor-capacitor) network at frequencies up to several megahertz, and at least some of the network time constants are one or more orders of magnitude larger than the switching period. Each switching event is, in an ideal sense, a discontinuity. Typical power converter simulations can run for hundreds or thousands of switching cycles, which can create problems for simulation software that does not have explicit mechanism for handling these discontinuities (events). Among these problems are long simulation times, lack of convergence, and convergence to an erroneous solution.

The standard software for circuit simulation, SPICE (and its derivatives), is not well suited to power converter simulation due to its lack of true event handling [1]. This paper proposes the use of the general purpose modeling language Dymola [3] in conjunction with the standard simulation language ACSL [4] to simulate switching power converters. ACSL has true event handling, which makes it well suited for switching converter simulation; however, it is unsuitable for circuit designers since it requires an ODE (ordinary differential equation) model of the converter in question. Furthermore, ACSL is not object-oriented, rendering the development of even moderately complex power supply system models difficult. Dymola solves both problems: it can convert a circuit description into a set of ODEs, and it is object-oriented, i.e. models can be reused as components in larger systems.

Model development proceeds in the following manner. First, a set of electrical component models are developed. These components include the standard components such as resistors, inductors and capacitors. In addition, simple switch and diode component models are developed that contain the switching event descriptions. Second, the circuit designer can connect the components together to form a circuit such, and need not worry about simulation events or discontinuities. Next, such a circuit model may in turn be used as a component in a system, such as a multiple converter power supply. At each level, the modeling is completely object-oriented. Dymola generates an ACSL program from the model, which is compiled and run under ACSL.

In this paper, the object-oriented modeling of a complete power supply system, from the component to system level, will be demonstrated by modeling and simulation of a regulated power supply system.

II. SWITCH-MODE POWER CONVERTERS

Switch-mode power converters, due to their efficiency and high power density, .find use in nearly every system that requires electrical power. This section gives a brief overview of switching power converter operation, modeling, and simulation.

IIA. Switching Power Converter Operation

Switch-mode power converters operate by the periodic storage and release of electrical energy in capacitors and inductors. The flow of energy is controlled by means of switches, which are divided into two classes, time-event driven and state-event driven. Time-event driven switches are controlled by some periodic waveform external to the converter network, and are usually implemented with transistors. State-event driven switches are controlled by the state variables of the converter network, i.e., the inductor currents and capacitor voltages; such switches are usually implemented with diodes.

It is helpful to review the operation of a simple pulse-width-modulated (PWM) switching power converter. Figure 1 illustrates the common buck-boost converter (a) and its main operating mode (b), continuous conduction mode or CCM. The converter is controlled by modulating the duty cycle D of a control signal applied to the switch S_1. Switch D_1 is a diode, and is controlled by voltage and current waveforms internal to the converter network. The values of L and C are chosen such that the circuit's time constants are large compared to the period of the switching signal applied to S_1. This causes a dc output voltage $V = DV_g/(1 - D)$ which will have a small ripple superimposed due to the switching behavior of the circuit. The detailed behavior of switching power converters can be found in many texts and references, including [5].

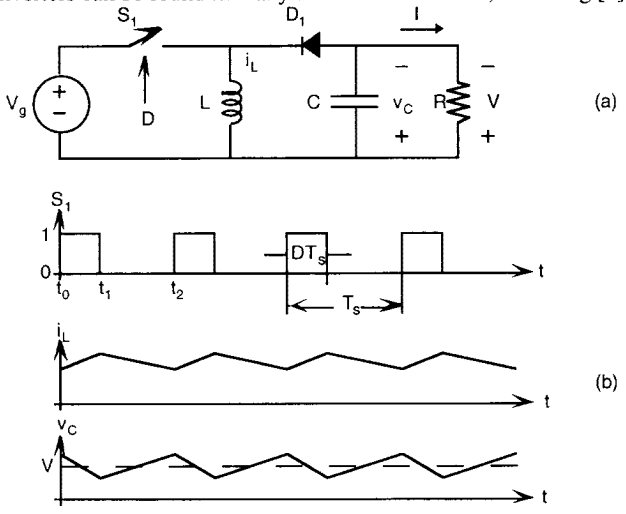

Figure 1. Buck-boost converter (a) and major waveforms (b).

0-7803-2730-6/95 $4.00 © 1995 IEEE

IIB. Switching Converter Modeling

Switch-mode converter models can be divided into two classes: instantaneous and averaged. Instantaneous models are simply the circuit model of the converter and as such are time-varying and discontinuous due to their periodically switched nature (Fig. 1 shows an instantaneous model). Such models accurately model component waveforms and large-signal behavior. They are useful for verification of operation, determination of operating points, transient phenomena, stress analysis, operating mode changes, etc. However, these models are of limited utility for the study of frequency response, stability analysis, etc. Furthermore, these models have the simulation difficulties mentioned in the introduction.

Averaged models are circuit or equation models in which the time-variations due to the switching behavior have been averaged out, resulting in a continuous time-invariant model. Averaged models are much more amenable to mathematical analysis, frequency response analysis, closed-loop stability studies, and so forth. Such models are usually limited to frequencies much lower than the switching frequency.

IIC. Utility Of Simulation Events in Converter Modeling

The use of events can improve both the speed and accuracy of simulation [6]. Suppose we are using a simulation program without true event handling, e.g. SPICE, to simulate an instantaneous model of a switching converter. At each discontinuity, the integration step size decreases markedly, since the simulator "perceives" the discontinuity as a sudden increase in the apparent magnitude of the system's eigenvalues. After the event is over, the step size usually increases slowly due to the conservative nature of typical integration algorithms. In the ideal system, the eigenvalues may change slightly or not at all during each event, allowing the same step size as before the event, but the integration algorithm doesn't know this. If events occur with sufficient frequency, the integration algorithm step size remains much smaller than the dynamic behavior of the system justifies, thus lengthening the simulation run unnecessarily. For example, in a typical switching converter, step size may be on the order of nanoseconds at the edge of a switching waveform, although a transient phenomenon may last milliseconds. Furthermore, it has been shown that lack of event handling can result in erroneous simulation results without any indication of error [6], an occurrence undoubtedly experienced by many who use SPICE for converter simulations. These errors are due to the failure of most integration error estimation methods at extremely small step sizes, and they can result in convergence to the wrong solution without warning.

Another disadvantage of programs without explicit event handling is that they may generate huge amounts of superfluous data, due to the very small step sizes in the region of each discontinuity. If the events are few and far between, this may not be a problem, but if they are numerous and closely spaced, as with switching converters, a program without event handling may generate a volume of data an order of magnitude larger than one that has explicit event handling. While one may attempt to alleviate this problem by specifying a larger output communication interval, by doing so one may miss event times, causing the output plots to be inaccurate (and in some cases look very strange).

These problems occur because software such as SPICE does not properly handle events. To explain this statement, consider what it means to possess true event recognition and handling. Most simulation programs, recent versions of SPICE included, can recognize that an event has occurred. For example, a statement of the form "if $v > V_{max}$ then transistor $Q1$ explodes" recognizes a certain event and spells out a consequence. What this statement does not do is guarantee that the consequence occurs at the time of the event. Recall that a simulator solves the system at discrete time points. Consider our example event above, and say that at time point t_n, $v < V_{max}$, but at the next time point t_{n+1}, $v > V_{max}$. Most software will just assume that the event occurred at time t_{n+1}, and continue from there. For the consequence "$Q1$ explodes", the exact event time t_{event} probably doesn't matter, but in many cases, it matters greatly. The problem is intensified if the system has slow dynamics between event times, e.g. switching power converters. In this case, a typical variable time-step integration routine will use a large time-step, and the error between the perceived event time t_{n+1} and the actual event time t_{event} can be substantial. A program that handles events properly will, upon occurrence of an event, employ a routine to search for the exact time t_{event}. It will then re-compute the system with t_{event} as the end-point of the simulation, store the appropriate variables, and restart the simulation at time t_{event}.

An instantaneous model makes good use of simulation events, since they use events to describe each switch opening and closing. Since event-handling simulation software recognizes discontinuities, it can store all state variables at the time of an event, and restart the integration with the same step size after the discontinuity is over. Furthermore, the simulation is no longer susceptible to erroneous convergence due to error estimation failure in the integration routine.

Although not covered in this paper, simulation events are also useful for constructing multi-mode averaged models. One of the limitations of averaged models is that different models are required for different operating modes. For simulation purposes, one may include more than one model description and switch between them according to the operating point.

III. Dymola Modeling of Electrical Networks

It has been shown in [7] that Dymola may be used for topological modeling of linear electrical networks (non-linear models are discussed in Section V). The topological description of the circuit can be given in a netlist format similar to that used by SPICE. Dymola solves the circuit description and computes a set of state equations, from which it can generate a model in a number of simulation languages. ACSL is preferred due to its event-handling capability.

The Dymola language uses an object-oriented (hierarchical) modeling method well suited to circuit modeling. Electrical components are developed into models, which can be used as part of larger models, similar to subcircuits in SPICE. In addition, model parameters and variables, even those at the bottom of the hierarchy, are always easily accessible; however, they need not be carried up the hierarchy, preventing unwieldy model descriptions. Of course, Dymola is a general purpose modeling language not limited to electrical circuits.

Dymola handles event descriptions in a high-level object-oriented manner [8]. This makes Dymola extremely useful for modeling switching power electronic circuits. The switching event descriptions are simply included in *switch models*, which can then be used like any other circuit components. The following sections describe models of electrical components.

IIIA. Linear Circuit Modeling

Many switching converters can be accurately modeled with linear components such as resistors, capacitors, inductors, transformers, etc. We first define a class that describes terminal behavior for various components. The component descriptions then call this description and inherit its terminal definitions. For example, the following listing describes terminal behavior of all two-terminal (one-port) devices:

```
model class OnePort
  cut WireA(Va/i), WireB(Vb/-i)
  main path AB <WireA - WireB>
  local v
    v = Va - Vb
    p = v*i
end
```

Dymola keywords appear in bold type. The **model class** command defines an object class of type *OnePort*. The **cut** command defines the terminal connections of an electrical component; note that each cut defines two variables, in this case electric potential and current. The **path** command creates a directed path from one cut to another, i.e. a port. The **local** variable v defines the voltage drop across the port.

Components are now easily described. For example, a capacitor is given by:

```
model class (OnePort) Capacitor
  parameter C=1.0
    C*der(v) = i
end
```

This model inherits the terminal description and equations given by model class *OnePort*. The capacitance is given by the **parameter** C, with a default value of 1.0. The capacitor behavior is described by the differential equation, where **der**(x) is the derivative of x. Other components (resistors, inductors, sources, ground, etc.) are similarly described. They are collected into a library file which can be called by another Dymola model.

IIIB. Event-Based Switch & Diode Models

An ideal switch can be considered to be a controlled resistor that has either zero or infinite resistance depending on whether the switch is on (closed) or off (open), respectively [8]. A Dymola model of an ideal switch is given by:

```
model class (OnePort) Switch0
  terminal On
    0 = if On then v else i
end
```

The **terminal** statement declares a variable *On* that may be connected to another model; this allows control of the switch. The **if** statement generates an appropriate event description in the target simulation language, thus allowing the simulation to properly handle the switching event.

A diode is a special kind of switch whose state is controlled by its terminal variables. An ideal diode has two possible states: (1) $v \le 0 \Rightarrow On = false$, or (2) $i > 0 \Rightarrow On = true$. Figure 2 describes the characteristic of an ideal diode. A Dymola model is given by:

```
model class (Switch0) Diode0
    new(On) = v > 0 or i > 0
end
```

This model inherits the *Switch0* behavior. The state of the switch is controlled by the switch's internal variables. On a zero-crossing of either i or v, an event is generated. The **new** statement resets the boolean variable *On* according to the boolean value of the right-hand side of the diode equation. The value of *On* changes only at the time of the event, so that between events, *On* remains constant.

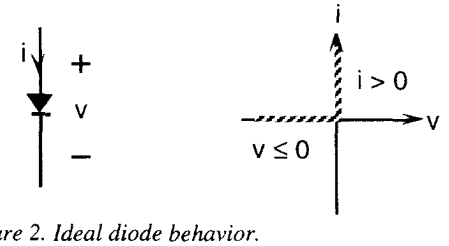

Figure 2. Ideal diode behavior.

When using ideal switches, one must deal with the issue of computational causality [7, 9]. In brief, computational causality deals with the fact that the description of an electrical (or other) component relates two or more variables by some defining equation. In an ODE formulation of a problem, one of these variables, the *effect*, must be given as a function of the other(s), the *cause(s)*. For some components, e.g. inductors and capacitors, this causality is fixed due to the fact that numerical integration is practical, but numerical differentiation is not. For example, to solve the inductor equation, we integrate the current to get the voltage, hence the current is the cause, and the voltage is the effect. A resistor, on the other hand, can take either possible causality, hence its causality is determined by surrounding components in the system. If the system is such that for some given component, either causality is possible, that component lies in what is called an *algebraic loop*. Note that causality is an artifact of the method of computer solution, and has little or nothing to do with physical cause and effect.

In the *Switch0* models, the computational causality of the switch differs according to the value of v and i, and is thus indicated by the state of the *On* variable, so that the switch can take on either possible causality. Hence, both causalities must be compatible with the circuit, i.e. the switch must lie in an algebraic loop. In practical terms, this means that an ideal switch cannot in general occur in a cutset whose only other elements are inductors and/or current sources, nor in a loop with only capacitors and/or voltage sources. This makes physical sense, as we cannot interrupt the current in an inductor nor instantaneously discharge a capacitor.

Unfortunately, this creates problems. Consider the operation of the buckboost converter of Fig. 1. For $t \in (t_0, t_1)$, S_1 is closed, and the inductor current ramps up with slope $(V_g - V)/L$. The voltage across the diode D_1 during this time is V_g, so D_1 is open. At $t = t_1$, S_1 is opened. The inductor current must go somewhere, so it flows through D_1, turning it on. However, the simulation program will not see it this way. It views D_1 as a switch, thus we have a cutset consisting of two switches (S_1 and D_1) and an inductor. The simulation runs until S_1 opens, at which point the simulation dies due to a division by zero.

There are a number of possible solutions to this problem. The first, and most obvious, is to make the diode into a non-ideal switch by giving it finite non-zero "on" and "off" resistances. The result of this is that the *topology* of the circuit never changes, only some component values. Below is a Dymola model for such a non-ideal switch, *Switch1*. We then create a *Diode1* model identical to *Diode0*, except that it calls *Switch1*.

```
model class (OnePort) Switch1
  terminal On
  parameter Ron=1.0E-4, Roff=1.0E4
    0 = if On then v-i*Ron else v-i*Roff
end
```

This solves the problem, but not without drawbacks. For instance, in the case of the buck-boost converter of Fig. 1, suppose the converter is operating in discontinuous conduction mode (DCM), so that during each switching cycle, the inductor current becomes zero while S_1 is off. This results in D_1 turning off as well. The large off resistance of D_1 in series with L results in a stiff system, i.e. one whose eigenvalues differ by orders of magnitude. This may create other problems, including long simulation times. This is seen in Section IV of this paper. Another solution has been proposed in [6], but this is not yet formulated in an object-oriented manner.

IIIC. Modulator Models

There are two common switch control waveforms for switching converters. The most common today are pulse-width control and

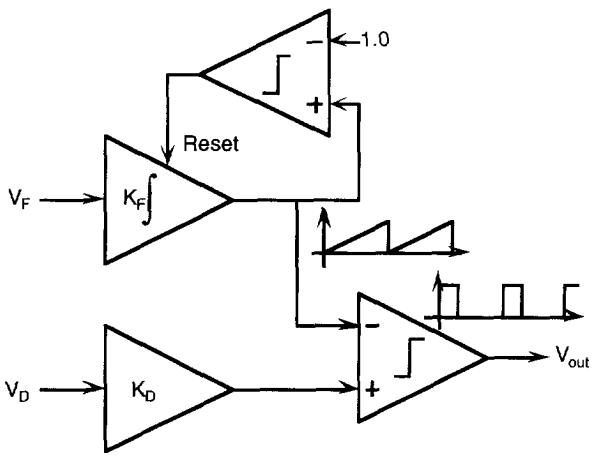

Figure 3. General purpose switching modulator. V_F *controls the switching frequency of* V_{out}, V_D *the duty cycle.*

frequency control, achieved by means of pulse-width modulators (PWMs) and voltage-controlled oscillators (VCOs). Both waveforms can be generated by the same system, shown in Fig. 3.

This is represented by the following Dymola model. The **terminal** variable *D* is a boolean variable used to turn a switch on or off. The **when-endwhen** construct senses when the integrator output exceeds unity and generates a corresponding event description in the target simulation language. The integrator state *x* is reset at the event time with the **init** statement. Note that this model has two voltage inputs, *v1* and *v2*, which modulate frequency and duty-cycle respectively. The terminal structure is inherited from the model class *TwoPort*, much like the model class *OnePort* previously defined. Note the simplicity of the model.

```
model class (TwoPort) Modulator
  parameter KFreq=1.0, KD=1.0, F0=0.0
  parameter RinFreq=1.0E6, RinD=1.0E6
  terminal D
  local x=0.0, xref, Frequency
    v1 = RinFreq*i1
    v2 = RinD*i2
    xref = KD*v2
    Frequency = F0 + KFreq*v1
    D = x < xref
    der(x) = Frequency
    when not x<1 then
      init(x) = 0
    endwhen
end
```

A pulse-width modulated signal can also be constructed directly, without requiring an integrator. The Dymola model below accomplishes the task. Note that this model includes a start time *Tstart* before which the output is zero. This variable can be used to offset the phase of converters in a multiple-converter system, a commonly used method to reduce switching noise.

```
model class (OnePort) PWM1
  parameter K=1.0, Ts, Tstart=0.0, Rin=1.0E6
  terminal D
  local x, xref, Nextime, Start=true
    v = i*Rin
    xref = K*v
    D = x < xref and not Time<Tstart
    x = if Time < Tstart then 0 ->
          else (Time - Nextime)/Ts + 1
    when (not Time < Nextime) or Start then
      new(Nextime) = if ->
        Start and (Tstart<0 or Tstart>0) ->
        then Tstart else Nextime+Ts
      new(Start) = false
    endwhen
end
```

IIID. Error Amplifier Model

A practical converter system requires a feedback loop to regulate the output. Part of this feedback loop is an error amplifier that compares the converter output voltage to some reference voltage. The following is a behavioral model given of a linear amplifier. It includes input and output resistance, a single pole response, and an output voltage limiter. The ability to use a behavioral model allows a very simple description.

```
model class (OpAmp) OpAmp1
  submodel (Resistor)   Rinput(R=Rin), Routput(R=Rout)
  submodel (Limiter)    Lim(LowLimit=Vlow,->
                        HighLimit=Vhigh)
  submodel (VSource)    Voutput
  submodel Common
  parameter Rin=1E6, Rout=1, Gain=1E6, PoleFreq=10
  parameter Vhigh=15, Vlow=-15
  local v
  node n0, n1
    connect Rinput from InPlus to InMinus
    connect Voutput from n1 to n0
    connect Routput from n1 to Out
    connect Common at n0
    der(v)  = PoleFreq*(Gain*Vin-v)
    Lim.x   = v
    Lim.y   = Voutput.V
end
```

IV. Dymola Models Of Converter Systems

The following section develops a converter model by connecting the various components defined in the previous section. This model is then used as part of larger systems. The first system is a simple open-loop system that allows discussion of some of the issues involved with simulation of power converters, and includes a comparison with SPICE. The second system is a closed-loop regulated system.

IVA. Flyback Converter Model

The example given is the flyback converter, shown in Fig. 4. The flyback converter is functionally equivalent to the buck-boost converter, but is more commonly used due to the isolation and turns ratio provided by the transformer.

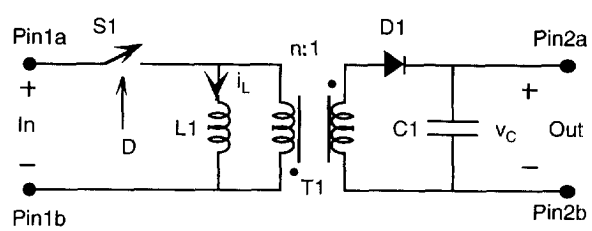

Figure 4. Flyback converter.

First, we define a terminal structure compatible with all single-input-single-output converters. This model class is similar to that for *TwoPort*, but reflects conventions used in power converter design:

```
model class Converter
  cut Pin1a(V1a/i1a), Pin1b(V1b/i1b)
  cut Pin2a(V2a/i2a), Pin2b(V2b/i2b)
  cut Port1[Pin1a, Pin1b], Port2[Pin2a, Pin2b]
  path In<Pin1a - Pin1b>, Out<Pin2a - Pin2b>
  main path P <Port1 - Port2>
    local v1, v2
      v1 = V1a - V1b
      v2 = V2a - V2b
end
```

Now, we construct the converter by connecting the electrical components together. Each **submodel** statement declares the components used and their parameters. In this example, the parameters for the components *L1*, *C1*, and transformer *T1* are also

parameters of the model class *Flyback0*, i.e. we can specify their values when we use the flyback model as part of a larger system. The terminal *D* is used to turn switch *Sw1* on or off. The **node** statement is used to connect the **cuts**, or wires, of the various components together; note that nodes and cuts are structural equivalents. The flyback converter model is:

```
model class (Converter) Flyback0
  submodel (Inductor)   L1(L=L)
  submodel (Capacitor)  C1(C=C)
  submodel (Transformer0)       T1(n=n)
  submodel (Switch0)  Sw1
  submodel (Diode1)     D1(Vd=Vd)
  parameter L, C, n, Vd
  terminal D
  node n1, n2
    connect L1 from n1 to Pin1b
    connect Sw1 from Pin1a to n1
    connect <In> T1 from n1 to Pin1b
    connect <Out> T1 from Pin2b to n2
    connect D1 from n2 to Pin2a
    connect C1 from Pin2a to Pin2b
    Sw1.On = D
end
```

IVB. Open Loop System

The *Flyback0* model can now be used in a larger system. The simplest example of this is to connect a voltage source to the input and a resistive load to the output, shown in Fig 5. A PWM is needed to supply the switching signal to the converter.

The following Dymola program represents the system of Fig. 5:

```
model circuit
  submodel (VSource)    Vg, Vcontrol
  submodel (Resistor)   RLoad(R=5)
  submodel (Flyback0)   Fly(L=200E-6, C=22E-6, ->
                        n=1, Vd=0.7)
  submodel (PWM1)   PWM(Ts=1.67E-5)
  submodel Common
    node n0, n1, n2, n3
    output  iLoad, vLoad
      connect Common at n0
      connect Vg from n1 to n0
      connect RLoad from n2 to n0
      connect <In> Fly from n1 to n0
      connect <Out> Fly from n2 to n0
      connect PWM from n3 to n0
      connect Vcontrol from n3 to n0
      Vg.V = 40
      Vcontrol.V = 1/3
      Fly.D = PWM.D
      vLoad = RLoad.v
      iLoad = RLoad.i
end
```

This model is compiled by Dymola into an ACSL model, which is then simulated. The results of this simulation and a SPICE3 simulation of this system are shown together in Fig. 6, where the load voltage and converter inductor current are plotted. The two results are virtually identical. The different simulators were not available on the same machine, so the results were normalized to the machine running ACSL for the fastest simulation (the actual CPU times are given in parentheses). ACSL was run on a DEC VAX/VMS system, and SPICE3 on a color NeXTStation. The simulation times were 1.0 (19.12 seconds) for ACSL and 5.0 (80.52 seconds) for SPICE3. In addition, the ACSL run generated 1280 data points, and SPICE generated over 20,000! Finally, note that in order to obtain an accurate SPICE simulation, the maximum allowed time-step was 0.1 microseconds. Larger time-steps resulted in convergence to erroneous results, but they did not cause a failure to converge, i.e. there was no indication (other than operator experience) that something had gone awry. This is a much more serious problem than slow simulation speed, for obvious reasons.

The above simulation runs very fast, and the use of the nonideal switches does not cause any problems due to stiffness. The reason for the latter is that for the given parameters, the converter runs in continuous conduction mode (CCM) meaning one of the two switches is always on. As a result, the system never becomes stiff. However, in a real system, one cannot count on always operating in CCM, so the converter was simulated to run in discontinuous conduction mode (DCM). In this mode, all switches are off for an interval during each switching cycle. If we reduce the value of *L* (the inductor) in the submodel *Fly* to 50μH and increase the load resistance *Rload* to 20Ω, the converter will operate in DCM (simulation results are shown in Fig. 7). The simulation time for a 2 millisecond run has increased to 4.9 (94.69) seconds for ACSL and 5.9 (92.26 seconds) for SPICE3. ACSL generated about 1500 data points, much less than the nearly 22,000 generated by SPICE. In this case, the system becomes stiff for a fraction of each switching period, and the ACSL simulation slows drastically. A solution to this would be to use an integration routine that handles stiff systems, e.g. the Gear algorithm; unfortunately, the ACSL implementation of the Gear algorithm is unsuitable for simulation of switching converters, as it cannot handle the discontinuities. Hence, we are stuck with using the 5th order variable-step Runge-Kutta algorithm. This is a poor algorithm for stiff systems, and contributes to the slow simulation speed.

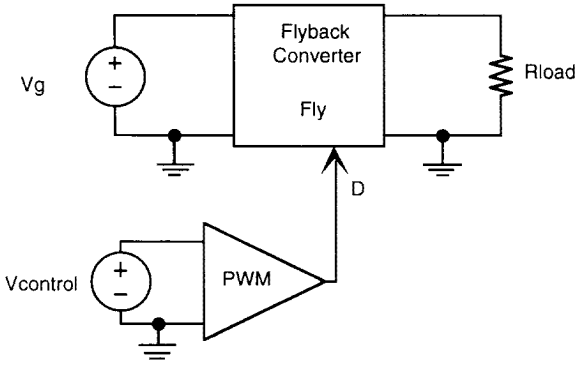

Figure 5. Flyback converter system.

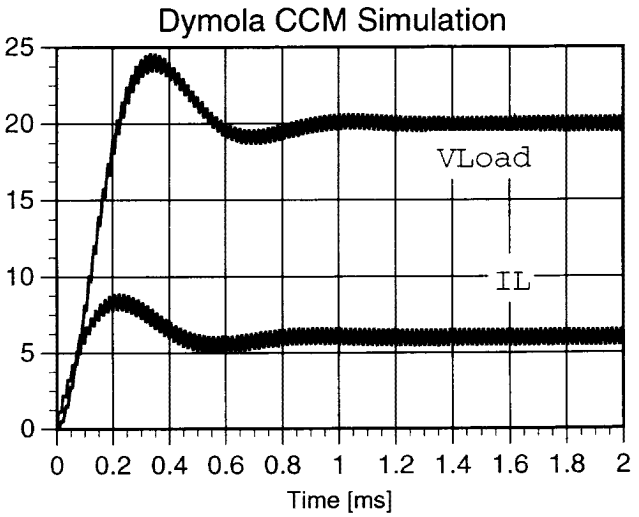

Figure 6. Dymola and SPICE3 simulation results for the system of Fig. 5, operating in CCM (continuous conduction mode).

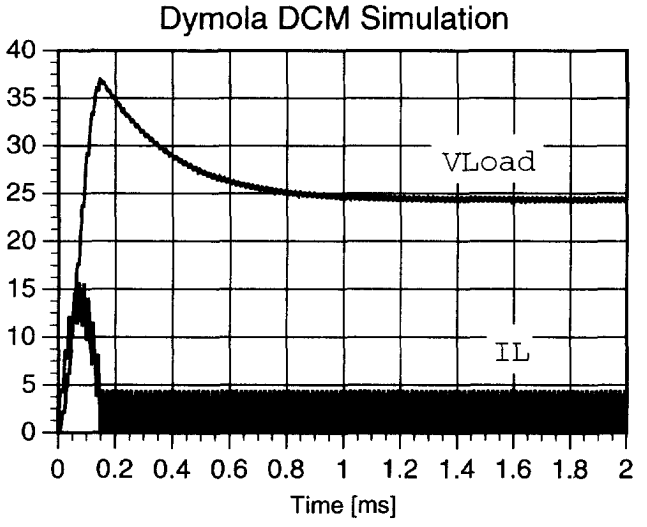

Figure 7. Dymola and SPICE3 simulation results for the system of Fig. 5, operating in DCM (discontinuous conduction mode).

The use of non-ideal diode models can cause another problem due to introduction of artificial stiffness. In the Dymola model for the system of Fig. 5, note that the ideal *Switch0* model was used for *Sw1*. We can get away with this since the non-ideal diode *D1* is actually modeled as a variable resistance, thus *Sw1* is always in an algebraic loop. If, however, we model *Sw1* with the non-ideal switch model *Switch1*, and run the converter in DCM, the inductor current exhibits extremely small, rapid oscillation about zero. This oscillation will cause the diode to turn on and off dozens of times each switching period, slowing the simulation to a crawl, although the results remain accurate.

There are at least two solutions to these problems. First, since Dymola can formulate models as a differential-algebraic equation (DAE), one may use a simulator that handles DAE systems. This does away with the switch causality problem, but one may pay a penalty in decreased simulation efficiency versus an ODE formulation, particularly if the system is not normally stiff. Second, one may use an extension of the Pantelides algorithm [6]. This allows one to use ideal switches and will result in an ODE formulation of the problem. However, Dymola does not yet implement this in an object-oriented manner.

IVC. Closed Loop System

In this section, an error amplifier and feedback loop is added to the system (Fig. 8), and the system simulated for a startup transient and a jump in input voltage from 50V to 40V. The simulation CPU time for an 8ms run was 98.61 seconds, and Fig. 9 shows the results (almost 480 switching cycles). The Dymola model is given below.

```
model circuit
  submodel (VSource)    Vg, Vref
  submodel (Resistor)   RL(R=0.5), Ri(R=1E4), Rf(R=2E4)
  submodel (Flyback0)   Fly(L=250E-6, C=470E-6, n=4, ->
                        Vd=0.7)
  submodel (PWM1)       PWM(Ts=1.67E-5, Dmax=0.5, ->
                        Dmin=0.02)
  submodel (OpAmp1)     Amp1(Rin=1E6, Gain=2, ->
                        PoleFreq=37.7)
  submodel Common
    parameter Tjump = 4E-3
    node n0, n1, n2, n3, n4, n5
    output  iL, iLoad, vLoad, Vvco, Vin
      connect Common at n0
      connect Vg from n1 to n0
      connect RL from n2 to n0
      connect <In> Fly from n1 to n0
      connect <Out> Fly from n2 to n0
      connect PWM from n3 to n0
      connect Amp1:Out at n3
      connect Vref from Amp1:InPlus to n0
      connect Ri from n2 to Amp1:InMinus
        Vg.V = Vin
        Vref.V= 5
        Fly.D = PWM.D
        iLoad = RL.i
        vLoad = RL.v
        iL = Fly::L1.i
        Vvco = PWM.v
        Vin = if Time < Tjump then 50 else 40
end
```

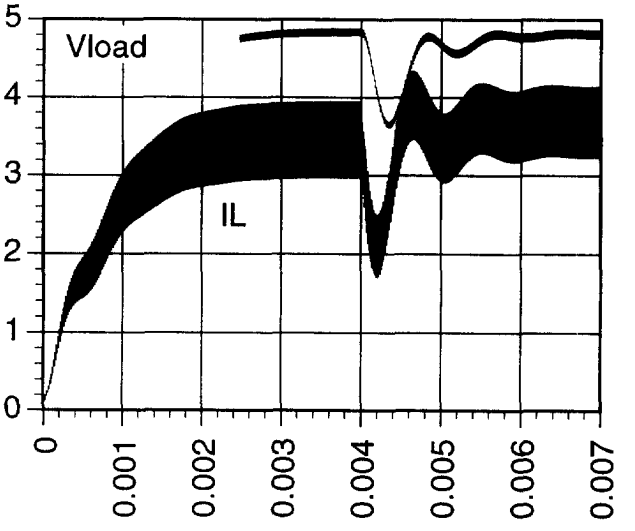

Figure 9. Simulation results for the regulated flyback system of Fig. 8.

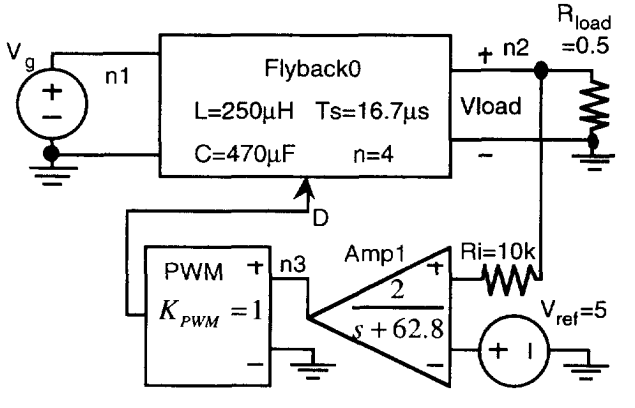

Figure 8. Regulated flyback converter system.

V. Nonlinear Systems

The reader has perhaps noticed that all the models used in this paper are piecewise linear. This is not a limitation of Dymola itself, but the fact that ACSL requires an ODE (ordinary differential equation) formulation of equations in order to simulate them. In this case, the task of Dymola is to convert the circuit descriptions into a set of ODEs and a system of simulataneous algebraic equations. The system of algebraic equations must be solved for the variables required by the ODEs. If linear, the system can be solved explicitly, but if nonlinear, it must be solved iteratively at each time-step. Recently, Dymola has extended in manner that, when it discovers systems of nonlinear simulataneous equations, automatically adds

code to the ACSL model to solve these equations via Newton iteration [10].

VI. Conclusion

As demonstrated in the previous sections, the Dymola language allows the modeling of power converter systems in a completely objected-oriented manner. One can develop electrical component models with which one can build power converter (and other) circuits. A topological circuit description similar to that used by SPICE can be used to develop such converter models, much as the *Flyback0* model was developed in this paper. Behavioral models of more complex components can also be developed, as demonstrated by the *Modulator, PWM1, and OpAmp* models given in Sections IIIC and IIID. All these models can be used together to model a complete system. Dymola will translate the models and their connections into a set of differential equations and generate from the latter a complete simulation model in a choice of several simulation languages.

The use of simulation events aids the simulation of switching power converters in several ways. It can help increase simulation speed and accuracy, and it can substantially reduce the size of the data set without loss of information. Unfortunately, most simulation languages that handle events, e.g. ACSL, require a formulation of the system in terms of ordinary differential equations; this formulation is unwieldy for all but the simplest of circuits. The problem is worsened by the fact that most simulation languages are not truly object oriented, making the modeling and simulation of practical systems difficult; this is especially true when one must include event scheduling in the models. Dymola solves both these problems at once: First, it allows event descriptions to be handled in a high-level manner by placing them inside a component model. Second, Dymola is inherently object-oriented, and the event-description-containing models are no exception. They can be employed in a system just like any other model.

Some of the problems encountered with the use of ideal switch models are discussed. In an ODE formulation of a problem, ideal switches can in general only be used when placed in an algebraic loop. Unfortunately, this creates problems in nearly all power electronic switching converters. The simplest solution, making the switches into variable resistances, can in some circumstances introduce artificial stiffness into the system, slowing the simulation speed. Other solutions include the use of a simulation language with a DAE solver, or perhaps an extension of the Pantelides algorithm to generate an ODE system where all switches end up in algebraic loops.

A practical example is given by means of the commonly used flyback converter. A converter model is constructed from common circuit components in the Dymola modeling language. This model is then used as a component in a larger system consisting of a PWM modulator model, along with a load and source. This model is compiled and simulated in ACSL. The results are consistent with those obtained by SPICE. Finally, a error amplifier and feedback loop are added to the converter to build a closed-loop regulated system, which is then simulated succesfully.

References

[1] Quarles, T.; A.R. Newton, D.O. Pederson, A. Sangiovanni-Vincentelli, *SPICE3 Version 3f3 User's Manual*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, 1993.

[3] Elmqvist, H., *Dymola - User's Manual*, DynaSim AB, Research Park Ideon, Lund, Sweden, 1993.

[4] Mitchell & Gauthier Associates, Inc., *Advanced Continuous Simulation Language (ACSL) Reference Manual*, Concord, MA, 1991.

[5] Cuk, S., *Advances in Switched-Mode Power Conversion, Vol. 2*. TESLAco, Pasadena, CA, 1983.

[6] Cellier, F., *Continuous System Simulation*. Springer-Verlag New York Inc., New York, NY, in press.

[7] Cellier, F., *Continuous System Modeling*. Springer-Verlag New York Inc., New York, NY, 1991.

[8] Elmqvist, H.; F. Cellier; and M. Otter, "Object-Oriented Modeling Of Hybrid Systems," Proceedings of the 1993 European Simulation Symposium (ESS'93), pp. xxxi-xli..

[9] F. Cellier; .M. Otter; and Elmqvist, H, "Bond Graph Modeling of Variable Structure Systems," Proceedings ICBGM'95, Second International Conference on Bond Graph Modeling and Simulation, Las Vegas, Nevada, January 15-18, pp.49-55.

[10] Elmqvist, H.; M. Otter; and F. Cellier ,"Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems," Proceedings of the 1995 European Simulation Multiconference, Prague, Czech Republic, June 5-7, 1995, in press..