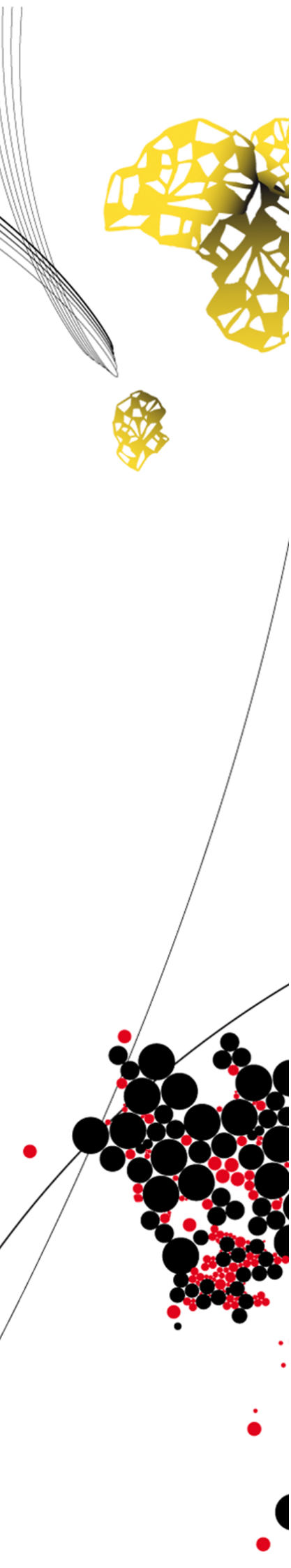


UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

FPGA Emulation of Analog Subsystems for Mixed-Signal Integrated Circuit Prototyping

Yoran D. Staal

B.Sc. Thesis
June 2021

Supervisor:
dr. ir. S. H. Gerez

Committee:
dr. ir. N. Alachiotis
dr. ir. R.A.R. van der Zee

Computer Architecture for
Embedded Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Mixed-signal integrated circuits are widely used in electronic designs. When developing mixed-signal integrated circuits, high-level models of analog systems are often used to validate the digital part of the design. This is mostly done using simulation software on a CPU-based machine. However, since digital system designs can be easily verified on FPGA, it might be beneficial to use the advantages of FPGA by emulating the analog part and make it synthesizable for this architecture. This way, the digital design can be verified on the same platform.

In this thesis, the concept of analog emulation on FPGA has been explored. This is done using a case-study of a DC-DC boost converter, that included a digital controller to keep the output voltage stable under varying input-voltage and output-load conditions.

An important part of this thesis is about the method: It takes multiple steps to go from an analog design to a digitally emulated circuit. These steps are explained and applied on a DC-DC boost converter. A selection of MATLAB/Simulink toolboxes that are used to simplify this process are explained as well.

The result of this thesis is a hardware design that approximates the dynamic behaviour of an analog circuit. The design was connected to a digital control system to validate its performance. For the case study, experiments showed that the emulated output voltage had an average relative error of 0.56% with a maximum error of 1.89%, compared to analog simulation in Simulink. This is considered sufficient for the purpose of digital-subsystem validation.

Contents

Chapter 1

Introduction

1.1 Motivation

While a lot of today's electronic processing is done in the digital domain, analog subsystems are still a fundamental part of integrated circuits (ICs). One big advantage of the digital domain is the fast prototyping. An example of a commonly used tool for digital hardware design verification is the field-programmable gate array (FPGA). On FPGA, digital ICs can be verified in a cost and time-efficient matter, since no test silicons have to be produced in the early stages of the development process.

Mixed-signal ICs (MSICs) contain not only digital, but also analog systems. For these systems, it becomes harder to test the digital hardware design since it will be connected to analog subsystems. A solution for this is analog simulation on CPU-based machines. Disadvantages of this are that interfaces might be required between the machine that runs that analog simulation and the FPGA. This can cause problems, for example when real-time simulation is needed. To overcome this problem, it might be interesting to emulate the analog part of mixed-signal chips on the same chip as the digital part.

In the context of this research, it is important to differentiate between the concepts of simulation and emulation. When a circuit is simulated, the dynamics are modeled to get insight in the behaviour. The goal of emulation is to realize a system on another system with identical behaviour. This is useful when the functional behaviour of a system is wanted in the other system. A simple example of emulation is a game emulators, for example a Nintendo 64 emulator. Using this, video games that only work on the Nintendo 64 game computer can be played on PC: you get the same behaviour, on a different system.

1.2 Goal of the assignment

The goal of this thesis it to find a method to simplify digital design validation on FPGA for mixed signal chips. This will be done by emulating the analog subsystem on FPGA.

In this thesis I will try to answer the following research question:

"How can FPGA emulation of the analog subsystems of mixed-signal ICs contribute to the design validation of the digital subsystems?"

To answer this question, multiple sub-questions have been defined:

- *"How can the continuous behaviour of an analog circuit be discretized"*
  The behaviour of analog circuits can be described by continuous ordinary differential equations. In order to emulate this on a digital system, the continuous definitions have to be numerically approximated.
- *"How can a discrete model be synthesized on FPGA"*
  If a discrete model is to be synthesized for FPGA, it should be described in a hardware description language (HDL). Data types also need to be optimized for the hardware.
- *"How does performance of analog simulation on FPGA compare to CPU-based simulation"*
  In order to validate the emulated circuit, performance will need to be compared to a traditional simulation on a CPU-based machine.

1.3 Report organization

In Chapter 2, the method for emulation of analog systems on digital hardware is presented. Chapter 3 shows the working principle of DC-DC converters. This chapter also presents the DC-DC converter design that will be used as case-study. After all the theory is given, Chapter 4 applies the emulation method that was given Chapter 2 on the circuit from Chapter 3. The result is a hardware design, that will be compared to a continuous DC-DC converter model. Results of this are shown in Chapter 5. Finally, in Chapter 6, the research questions that were given in Section 1.2 will be answered, followed by recommendations for improvements and future research.

Chapter 2

Analog hardware emulation

This chapter presents the different steps that are needed to convert a continuous analog system to a digitally emulated system with the same dynamic behaviour at the ports. The process can be divided into 4 different steps [1]:

1. First, the dynamics of the circuit are analysed, resulting in differential equations.
2. Discrete-time is required for an implementation on FPGA. Difference equations of the circuit are defined using a discretization method. A discrete block diagram can be made from this.
3. Next, the continuous model has to be converted to fixed-point.
4. Finally, the model is exported to HDL language and is synthesized for the FPGA platform.

Step 1 will be done in chapter 3. Step 2-4 will be analysed in this chapter and applied in chapter 4.

2.1 Discretization methods

The dynamic behaviour of analog circuits can be described by differential equations. A simple first-order differential equation is shown in Equation 2.1.

$$\frac{dy}{dt} = f(t, y) \tag{2.1}$$

Continuous functions have infinitely small time steps. However, digital systems have a system clock that has finite time steps. Therefore, in order to emulate a continuous system on a digital system, the differential equations need to be numerically approximated using a discretization method. The resulting difference equations

have finite time steps. Simple solutions have fixed time steps, however solutions that have variable time steps also exist [2]. In this work, only fixed time step methods are considered.

The dynamics of the system evaluated at the fixed time steps are called samples, where with the number of samples per second is called the sampling frequency. If the current time step is t_n, the next will be $t_{n+1} = t_n + T$ with T being the time between the two samples (sampling period).

Many different discretization methods exist, all with their own advantages and disadvantages. The simplest discretization method is the forward Euler method [3]. It is an explicit method, which means that it uses the previous values to calculate the current value. The forward Euler method is shown in Equation 2.2. Δt is the sampling period. The big advantage of this explicit method is that calculating the next value only takes one calculation cycle. This is a big advantage when a high simulating frequency is needed. For FPGA, clock frequencies are usually quite low, making the forward Euler method well suited. The approximation error is related to the sampling time Δt.

The implicit Euler method, also called the backward Euler method, uses all parameters at the same time level, as can be seen in Equation 2.3. This method is very stable and can be a good solution when simpler methods like the forward Euler method fail. The disadvantage of this method is the fact that it requires solving an algebraic equation for each time step, which is computationally more expensive.

$$y\left(t_{n+1}\right) = y\left(t_n\right) + \ \Delta t \cdot y'(t_n) \tag{2.2}$$

$$y\left(t_{n+1}\right) = y\left(t_n\right) + \ \Delta t \cdot y'\left(t_{n+1}\right) \tag{2.3}$$

The midpoint method is shown in Equation 2.4. It can be seen as a combination of the forward Euler method and backward Euler method: it takes the average of these two methods. This method gives more accurate results. However, it has the same disadvantage as the backward Euler method. The In [4], an optimization of the midpoint method is presented: ”By exploiting the time-scale separation properties of typical power electronics circuits and properly selecting the discretizing time step, the response of the signal-flow diagram can be made to converge to the original discretized circuit model response within a few iterations”.

$$y\left(t_{n+1}\right) = y\left(t_n\right) + \frac{\Delta t}{2}\left[y'\left(t_{n+1}\right) + y'\left(t_n\right)\right] \tag{2.4}$$

A discrete block diagram can be made in Simulink from the obtained difference equations. The equations can be represented in the Z-domain. The Z-domain representation of Equation 2.2 is shown in Equation 2.5. Using Unit Delay blocks, the sample delay z^{-1} can be realized.

$$y\left(z\right) = y(z)z^{-1} + \Delta t \cdot y'(z)z^{-1} \tag{2.5}$$

2.2 Fixed-point modeling

In the previous section, the problem of the infinitely small time steps of a continuous system was solved by converting to a finite time resolution. However, the values of the analog signals also have to converted from an infinite resolution and infinite range to digitally representable values.

Fixed-point numbers consist of two parts: an integer and a fractional part. Fixed-point numbers can represent fractional numbers, while algorithmic operations stay simple. The biggest challenge of using fixed-point numbers in a design is to select the length of the two parts in a way that no under- or overflow occurs. At the same time, the representable number has to have enough resolution to keep the quantization error low.

In Simulink, a unsigned fixed-point data type with a integer length of 6 and a fractional part of length 24 would be defined as following: fixdt(0,32,24). The arguments, respectively, are: signed, total word length, position of the point.

An analysis and discussion between fixed-point and floating-point numbers for FPGA is given in [5]. Fixed-point make the hardware design less complicated. The architecture is simpler and the overall design takes in less space on the chip. This is why the design in this thesis will have fixed-point numbers.

In Simulink, the double precision floating-point data type is used in models by default. The resulting block diagram model from Section 2.1 also uses this data type. In order to convert this to fixed-point values, the range and needed precision of all the signals in the model have to be analysed. Based on this, a minimal value for the word-length and the fraction length.

The Fixed-Point Designer Toolbox [6] plays an important roll in this process. The tool can automatically collect the ranges of all signals in the model, and can propose word and fraction lengths for fixed-point conversion.

When the tool is started, it will first prepare the model for conversion. Tolerances can be set for selected signals, both relative and absolute. After this, the ranges can

be collected by running a simulation. It is important here that this simulation gives a complete overview of the signal values of the system. The input signals should have at least the minimal and maximal values that they will get further on in the design process. The system should also have enough time to get into a steady state before the input changes to a different value.

When the ranges are collected, the signal ranges can be viewed in histograms. The Fixed-Point Designer cannot optimize word-length and fraction-length at the same time. This means that one should be selected manually beforehand. A way to approach this is by setting the word length to for example 32. The tool can then automatically propose the fraction length. Based on the result, the word-length can be adjusted until the model has the needed performance. It is possible to manually change data types in the tool after this. For example, a certain constant sources or gain values only use a few of the standard selected word-length.

The proposed and manually changed data types can then be applied to the model. After running another simulation with the new data types, the results can be compared to the double precision run.

2.3 Hardware implementation

In order to run the discrete model on FPGA, it should be implemented using a hardware description language (HDL). Verilog and VHDL are the two commonly used languages for digital hardware design. In this thesis, VHDL is used.

The HDL Coder [7] can convert Simulink models to VHDL or Verilog language. More than 300 so-called 'HDL-ready' Simulink blocks can be used to design models. Since the discretized model is already realized and converted to fixed-point in Simulink, using this tool is an efficient method for getting a VHDL implementation of the model. The model should be placed in a submodel, so that the correct ports can be configured. The HDL coder automatically detects the sampling frequencies that are used in the model. If all synchronous blocks in the Simulink model have the same sampling time, the exported code will only have one clock signal input. A reset, clock enable and clock enable out port will also be added with the default settings.

Block parameters that have an expression that consists of MATLAB work space variables will be exported to a constant value. If one of these parameters has to be configurable during run-time, the expression should be split up using adder, multiplier and division blocks before exporting the HDL.

The HDL coder will work without extensive setting up. The error messages that are generated when the tool cannot generate HDL are clear, making debugging

easy. The exported code is human readable. If needed, naming settings can be changed before exporting.

Simulating VHDL designs can be done using ModelSim. A testbench (TB) is required in order to efficiently evaluate the design. The TB provides input signals. Using this testbench, input data can be read from files, which is useful in the case where the input data is generated using other software, for example in MATLAB. During simulation the test bench can also write data to files. This way the data can be analysed in other software.

The final step to run the complete design on FPGA is to synthesize the VHDL design: the high level description is converted to a gate-level representation.

Chapter 3

DC-DC Boost Converter

3.1 Working principle

A DC-DC converter is a simple circuit that takes a voltage as input, and converts this to a lower (buck) or higher (boost) voltage at the output. The circuit can be designed to very specific input and output voltages, or it can be designed to operate on a wide range of voltages. If the voltage is only lower or only higher than the input signal, the circuit can be designed more specifically. In this thesis, a DC-DC boost converter is used. The analog part of the DC-DC boost converter can be seen in Figure 3.1. This can be referred to as the power plant [8]. The second part of the DC-DC boost converter is a control circuit. Both parts will be analysed and designed in this chapter.

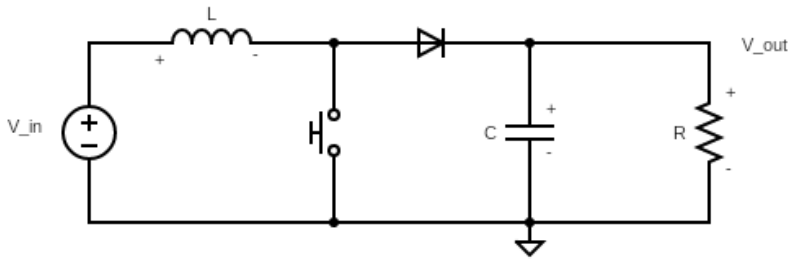


Figure 3.1: DC-DC boost converter

The switch changes the dynamics of the circuit. The analysis for this is given in Section 3.2. The switch is controlled using a PWM signal. The duty cycle of the signal can be adjusted, to control the output voltage [8]. A controller is needed to control the PWM generator block. There are many different control methods for DC-DC converters. A general setup is shown in the block diagram that is shown in Figure 3.2.

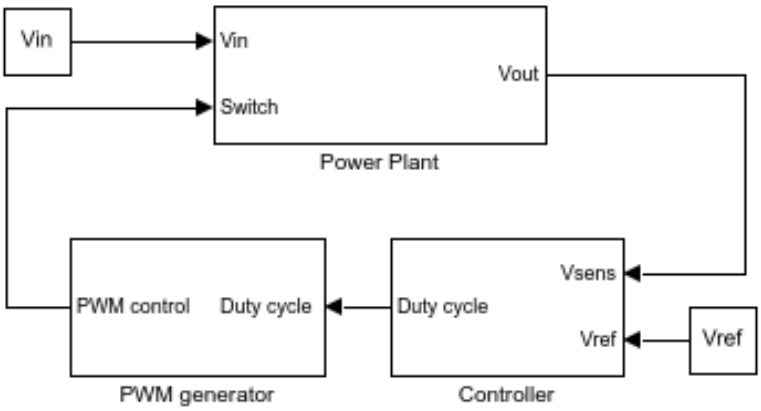


Figure 3.2: DC-DC boost converter

In this thesis, an ideal DC-DC converter circuit is used at all times. This means that the inductor, capacitor and resistor have no parasitic resistance and impedance. The semiconductor components are also ideal. Both the switch and the diode have two states. They behave either as an ideal wire or as an infinite resistance, dependent on the control input signal (switch) or voltage (diode).

With a non-ideal DC-DC converter, there would be a dead-time when the switch turns on or off. [8]. This is not the case for this ideal circuit, which makes analysis easier.

3.2 Power plant

3.2.1 Analysis

The DC-DC boost converter has components that have a state: The inductor and the capacitor. Normally, this would mean that the circuit can be described with two differential equations. However, since the circuit contains a switch that changes the dynamic behaviour of the circuit, it is easier to model it with four difference equation: two for the CLOSED state of the switch and two for the OPEN state. This method is also used in [4]. The effective circuit layouts for the two switch states are shown in Figure 3.3 (CLOSED state) and Figure 3.4 (OPEN state).

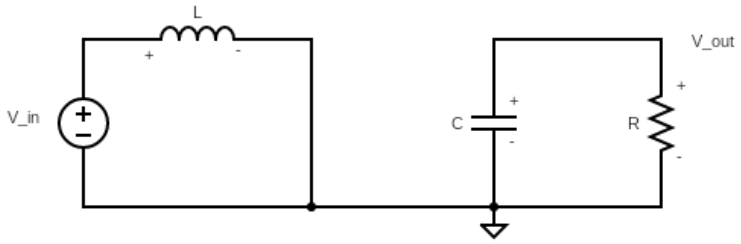


Figure 3.3: Effective circuit layout with switch closed (CLOSED state)

For the ON state, the inductor charges at a constant rate and the capacitor discharges at a constant rate:

$$\frac{di_L(t)}{dt} = \frac{V_{in}}{L} \tag{3.1}$$

$$\frac{dv_c(t)}{dt} = -\frac{v_C(t)}{RC} \tag{3.2}$$

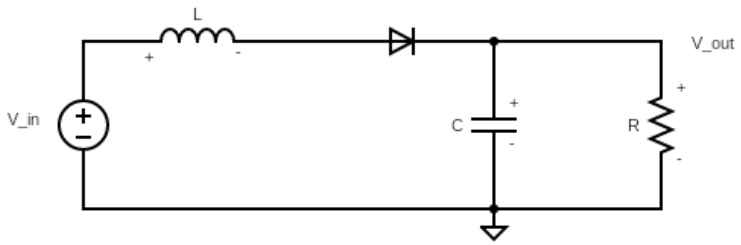


Figure 3.4: Effective circuit layout with switch open (OPEN state)

For the OPEN state, the differential equations are:

$$\frac{di_L(t)}{dt} = \frac{v_{in}(t) - v_c(t)}{L} \tag{3.3}$$

$$\frac{dv_c(t)}{dt} = \frac{i_L(t)}{C} - \frac{v_C(t)}{RC} \tag{3.4}$$

Since there is an ideal diode in series with the inductor, the inductor current will never be negative:

$$i_L(t) \geq 0 \tag{3.5}$$

3.2.2 Component selection

It has been mentioned before that a DC-DC boost converter will be used as a case study for this thesis. No specifications were given for the system. The starting point for the boost converter specifications is that it designed to be on an integrated circuit. The input and output voltage levels are standard logic level voltages. Component sizes will be as small as possible to minimize the size of the circuit.

The switching frequency is chosen to be 1MHZ. This relatively high switching frequency will also be a good validation factor for the emulated circuit, since the clock of the FPGA board that will be used for validation is 50MHz. For every switching period the analog emulation hardware only has 50 cycles.

Table 3.1: DC-DC converter specifications

Parameter	**Value**
Input voltage (V_{in})	2.5V ± 0.2V
Output voltage (V_{out})	3.3V
Output load ($R_{out,\ min}$)	330Ω
Switching frequency (f_{sw})	1MHz

Now that the input and output signal specifications have been defined, the size of the inductor and the capacitor can be determined. The same method that is used in [8] is used to find the minimal values for the components.

For the inductor, the value can be determined based on the average input current. This can be found by looking at the output power, with which we can determine the input power and thus the input current (the input voltage is known). The output power is shown in Equation 3.6.

$$P_{out} = V_{out} \cdot I_{out} = 3.3V \cdot 10mA = 33mW \tag{3.6}$$

For an ideal converter with a power efficiency of 100%, the input power would equal the output power. In this thesis, an ideal converter is used at all times. However, for the determination of component values, it is not bad to take power loss into account. The same design could for example be used in further research where parasitics would be added to the circuit. Therefore, I will assume this controller would have a 95% power efficiency. The input power is:

$$P_{in} = \frac{P_{out}}{\eta} = \frac{33mW}{0.95} = 34.74mW \tag{3.7}$$

$$i_{L,\ ave} = \frac{P_{in}}{V_{in}} = \frac{34.74mW}{2.5V} = 13.90mA \tag{3.8}$$

The waveform of the inductor current at the maximal output current is a triangle wave, as can be seen in Figure 3.5. Since the minimal value of the inductor current is 0 and the waveform is a triangle, the average current is exactly half the peak current, as can be seen in Equation 3.9. Rewriting the formula gives us Formula 3.10 for the peak current.

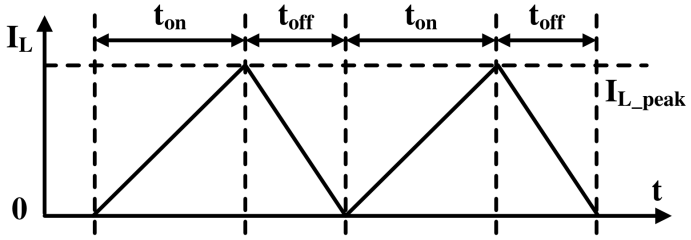


Figure 3.5: Waveform of the inductor current at the max load current [8]

$$i_{L,ave} = \int_0^T i_L(t)dt = \frac{1}{2} I_{L,peak} \tag{3.9}$$

$$I_{L,peak} = 2 \cdot \ i_{L,ave} = 27.8mA \tag{3.10}$$

Now that the maximum inductor current is known, the inductor size can be determined. According to [8], the inductor size is related to the switching frequency. The element equation for the inductor, as shown in Equation 3.11, can be used to get Equation 3.12, that is valid when the switch is closed (ON state).
t_{on} is the time that the switch is closed in one switching period when the input voltage and output load have the exact values that are defined in 3.1. This would mean that the switch can be controlled by a PWM source with a constant duty cycle.

$$V_L = L \cdot \frac{di_L}{dt} \tag{3.11}$$

$$V_{in} = L \cdot \frac{I_{L,peak}}{t_{on}} \tag{3.12}$$

Rewriting 3.12 gives the equation for the inductance. However, t_{on} is still unknown. For this we can use Equation 3.13 [8], which gives the ratio between t_{on} and

t_{off}. From this ratio the expressions of the timing values can be described in the switch period T, as is done in Equation 3.14.

$$\frac{t_{on}}{t_{off}} = \frac{V_{out} - V_{in}}{V_{in}} = \frac{3.3V - 2.5V}{2.5V} = 0.32 \tag{3.13}$$

$$t_{on} = 0.24T \qquad\qquad t_{off} = 0.76T \tag{3.14}$$

By combining Equation 3.12 and Equation 3.14, we can get the expression for the relation between the inductance and the switching frequency in Equation 3.15.

$$L \cdot \frac{1}{T} = L \cdot \ f_{sw} = 0.24 \cdot \frac{V_{in}}{I_{L,peak}} = 0.24 \cdot \frac{2.5V}{27.8mA} = 21.58\mu H/MHz \tag{3.15}$$

With Equation 3.15, the maximum inductance value for a certain switching frequency can be determined. As has been explained in the beginning of this section, $f_{sw} = 1MHz$. This means that the maximum inductance is 21.58μH. A rounded off value of 20 μH will be a good choice for the DC-DC boost converter design.

The choice of the capacitor value is based on the wanted maximum value output ripple [8]. Like has been mentioned before, good performance of the DC-DC boost converter is not the goal of this thesis: a ripple of $20mV$ will be a good value, since the ripple will be a good signal property for the validation of the hardware emulation. In Equation 3.16 the relation between the output ripple and capacitor value is shown. t_{dead} equals 0 since an ideal switch is used.

$$V_{o,ripple} = \frac{i_{load}\,(t_{on} + t_{dead})}{C_{load}} \tag{3.16}$$

By rewriting Equation 3.16, a expression for the minimal capacitor value can be found. Equation 3.17 shows that the minimal capacitor value is 222nF. Like the inductor value, this value is also rounded off to guarantee good performance for the DC-DC boost converter: a capacitor of 300nF is used.

$$C_{load} > \frac{t_{on} i_{load}}{V_{o,ripple}} = \frac{320ns \cdot .90mA}{20mV} = 222nF \tag{3.17}$$

The selected values for the DC-DC boost converter can be seen in Table 3.2.

Table 3.2: DC-DC converter component and timing values

Parameter	**Value**
Switching frequency (f_{sw})	$1MHz$
Duty cycle	24%
Inductor (L)	$25\mu H$
Load capacitor (C)	$300nF$

3.3 Control

In Section 3.2, it was assumed that the input voltage and the load of the DC-DC boost converter are constant values. When this is the case, a PWM source with constant duty cycle can be used to control the switch of the power plant. Many different control techniques exist. A good overview of all the different types of controllers is given in [9].

A discrete PI controller selected for the case-study design in this thesis, using the current output voltage as negative feedback. The controller can be designed using MATLAB Simulink toolboxes. This was done because of time restrictions. An analytical design approach for the controller would have cost more time, and the resulting design might still need tuning in Simulink as well.

The controller is designed using Control System Toolbox [10]. The power plant of the DC-DC boost converter first has to be linearized, including the PWM generator in the loop. This cannot be done directly by the Control System Toolbox: it cannot linearize the PWM generator, which has a duty cycle value with range 0-1 as input.

The System Identification Toolbox [11] can be used to get a linearization of the DC-DC boost converter. Using a step input, a linear step response curve can be automatically fitted to match the step response of the DC-DC boost converter as accurate as possible. The expected duty cycle that was calculated in Subsection 3.2.2 is used as step amplitude to get as close as possible to the operating point. A good approximation of a DC-DC converter can be made using a linear system with two poles and one zero [12]. The plot from the System Identification Toolbox can be seen in Figure 3.6

Now that the locations of the poles and the zero are defined in a linear system, the controller can be designed. This is easily done by tuning the response time and transient behaviour parameters in the in the Control System Toolbox. With a P gain of 0.04 and an I gain of 3275, the output voltage is stable and has good performance. The controller behaviour can be seen in Chapter 5.

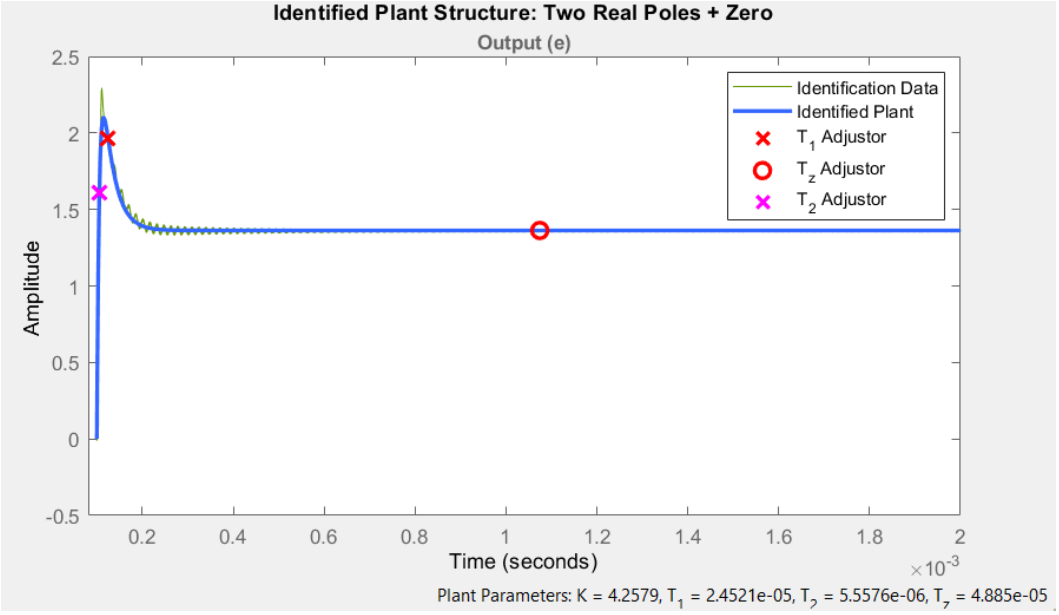


Figure 3.6: Linear approximation in the System Identification Toolbox

The controller has to be discretized, since it will be designed to run on an FPGA board. Parts from method that is used for the discretization of the analog plant can also be used for the controller. Therefore, the discretization and hardware implementation of the controller will be discussed in Section 4.2.

Chapter 4

Design

This chapter will apply the methods of Chapter 2 on the circuit that was presented in Chapter 3. The differential equations of the DC-DC boost converter that were found in Chapter 3 will be converted to difference equations. These difference equations are expressed in Z-domain, so that a block diagram can be made in Simulink. The block diagram will be converted to fixed-point, after which the model is ready to be exported to HDL.

4.1 Discretization of the power plant

The forward Euler method of discretization will be used for the design. The forward Euler method only takes one clock cycle per iteration. This is a great advantage since the switching frequency of the digital controller is only 50 times slower than the simulation clock frequency. Equation 4.1 and Equation 4.2 show the numerical approximation for the current of the inductor and the voltage of the capacitor of the DC-DC converter.

$$i_L\left(t_{n+1}\right) = i_L\left(t_n\right) + \ \Delta t \cdot i'_L(t_n) \tag{4.1}$$

$$v_C\left(t_{n+1}\right) = v_C\left(t_n\right) + \ \Delta t \cdot v'_C(t_n) \tag{4.2}$$

In Chapter 3, the solution for the analog plant was split up into an OPEN and an CLOSED part, each containing two differential equations. Filling in Equation 3.1 until Equation 3.4 in Equation 4.1 and Equation 4.2, Equation 4.3 until Equation 4.6 can be obtained. Note that the first two equations are the OPEN state equations and the latter two the CLOSED state equations. For V_{in}, there is no need to delay

the signal, since it is an input signal.

$$i_L\left(t_{n+1}\right) = i_L\left(t_n\right) + \frac{\Delta t}{L} \cdot v_{in}(t_{n+1}) \tag{4.3}$$

$$v_C(t_{n+1}) = (1 - \frac{\Delta t}{RC}) v_C(t_n) \tag{4.4}$$

$$i_L\left(t_{n+1}\right) = i_L\left(t_n\right) + \frac{\Delta t}{L} \cdot \ v_{in}(t_{n+1}) - \frac{\Delta t}{L} v_C\left(t_n\right) \tag{4.5}$$

$$v_C\left(t_{n+1}\right) = \left(1 - \frac{\Delta t}{RC}\right) v_C(t_n) + \frac{\Delta t}{C} i_L\left(t_n\right) \tag{4.6}$$

When comparing the two OPEN and CLOSED equations, it can be seen that the OPEN equations have all the terms on the right side that are also in the CLOSED equations, with one additional term. The whole system can be described in two difference equations by introducing the variable $S(t_n)$ that represents the switch state. $S(t_n)$ is defined in Equation 4.7.

$$S(t_n) = \begin{cases} 1, & \text{if the switch is closed at } t_n \\ 0, & \text{if the switch is open at } t_n \end{cases} \tag{4.7}$$

The two difference equations with $S(t_n)$ that describe the complete dynamic behaviour of the system are shown in Equation 4.8 and Equation 4.9. Like with V_{in}, $S(t_n)$ is also a system input. Thus, there is no need to delay the signal.

$$i_L\left(t_{n+1}\right) = i_L\left(t_n\right) + \frac{\Delta t}{L} \cdot \ v_{in}(t_{n+1}) - S(t_{n+1}) \frac{\Delta t}{L} v_C\left(t_n\right) \tag{4.8}$$

$$v_C\left(t_{n+1}\right) = \left(1 - \frac{\Delta t}{RC}\right) v_C(t_n) + S(t_{n+1}) \frac{\Delta t}{C} i_L\left(t_n\right) \tag{4.9}$$

The final step before the model can be realized is to transform the difference equations to Z-domain, so that the Simulink model can be realized. Equation 4.10 and Equation 4.11 show the result of the numerical approximation of the power plant.

$$i_L\left(z\right) = i_L\left(z\right) z^{-1} + \frac{\Delta t}{L} \cdot \ v_{in}\left(z\right) - S(z) \frac{\Delta t}{L} v_C\left(z\right) z^{-1} \tag{4.10}$$

$$v_C\left(z\right) = \left(1 - \frac{\Delta t}{RC}\right) v_C\left(z\right) z^{-1} + S(z) \frac{\Delta t}{C} i_L\left(z\right) z^{-1} \tag{4.11}$$

Block diagram

Using Equation 4.10 and Equation 4.11, a block diagram can be realized in Simulink. The complete model can be seen in Figure 4.1. Both the capacitor voltage and the inductor current signal are labeled, as V_C and I_L. The constant T is the sampling time that was defined as ΔT in the previous calculations. The output of the inductor adder is saturated at 0 lower bound, since it was shown in Equation 3.5 that the inductor current cannot be negative.

For most gain blocks, a constant gain expression that contains the constants T, C and L. However, the 'variload' subsystem (abbr. variable load) was designed so that the load value R can be changed during run-time. The subsystem can be seen in Figure 4.2, together with a single gain block that can only be used for a constant R for reference. The variload uses a division and an adder block to get the correct gain value, using the load value from the input port. A final multiplication block multiplies the input value on port 1 with the gain value.

$S(t_n)$ is realized using two switch blocks, with the OPEN state connected to a constant source of 0. The CLOSED state is connected to the signal that $S(t_n)$ is multiplied with in Equation 4.10 and Equation 4.11.

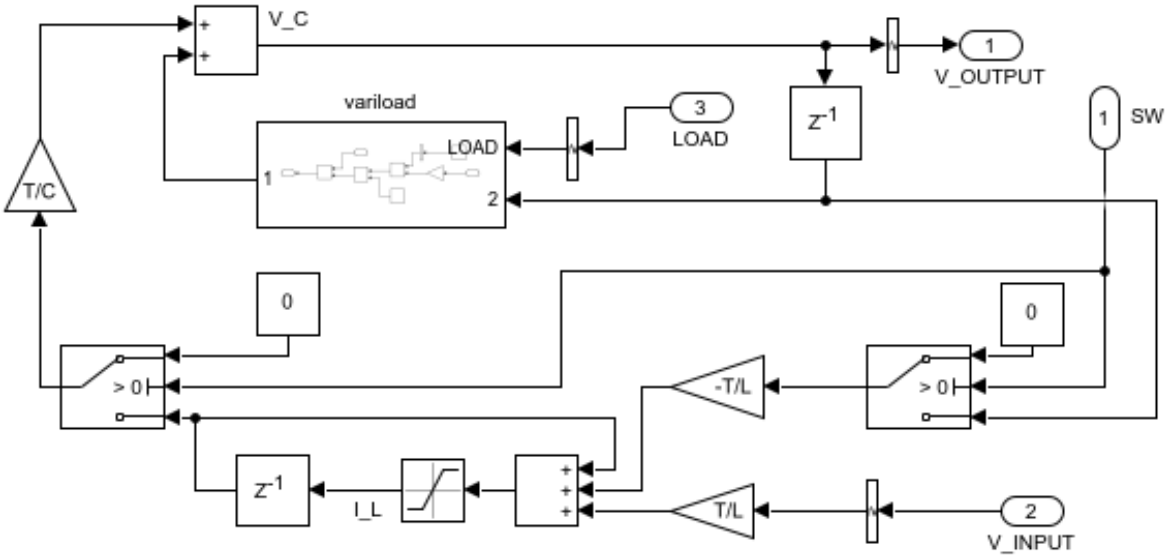


Figure 4.1: Discrete block diagram of the analog plant

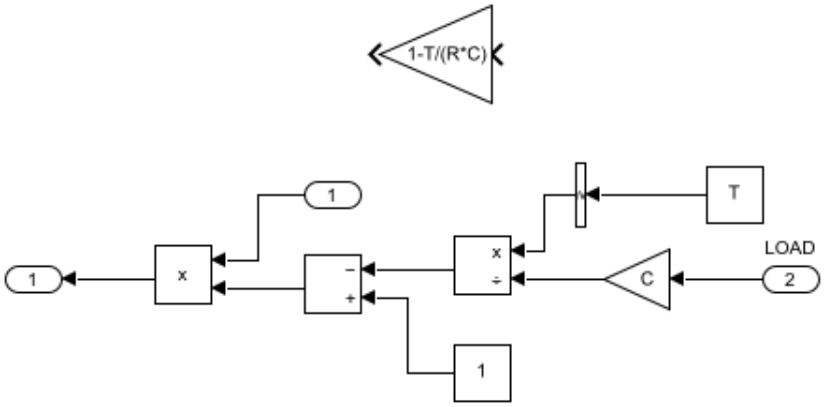


Figure 4.2: Block diagram of the variable load subsystem

4.2 Controller

In Section 3.3, a continuous PI controller was designed that takes a reference voltage and the current output voltage of the analog plant, and outputs a PWM control signal to the switch of the analog plant. This controller is the DUV, and will need to be implemented on FPGA using VHDL.

A PI controller can be realized by having a P-term and an I-term in parallel. For the P-term, nothing changes when the controller is discretized. However, the I-term contains an integral that has needs to be numerically approximated. Simulink has a Z-domain integrator that can be used for this. The gain has to be multiplied by the clock period, which happens in the block.

Since the switching period of the PWM generator is $1MHz$, a full switching period is 50 clock cycles. Therefore, it is straightforward to have a controller output range of [0,50]. The original designed controller in Section 3.3 was designed to drive a PWM generator that has a duty cycle input range of [0,1]. Therefore, the P and I gain value have to be multiplied by 50 to get the wanted output range. The discrete controller can be seen in Figure 4.3. Note that a saturation block has been added to limit the output to the correct range. The controller has been converted to 32 bit word-length fixed-point using the Fixed-Point Tool.

Using the HDL coder, the controller has been exported to VHDL language. However, the PWM generator block that was used in Simulink cannot be exported by the tool. Therefore, a simple PWM generator was implemented by hand-writing the code. The implementation is straightforward: a counter has to be implemented that counts to 50 and then resets. The output of the generator should be 1 if the duty cycle value from the controller is higher than the current counter value and 0 otherwise.

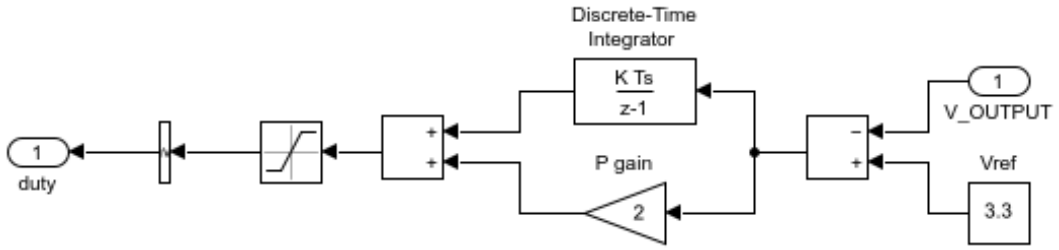


Figure 4.3: Block diagram of the digital controller

4.3 Hardware design

Fixed-point conversion

Now that the discrete model is realized in Simulink, it is time to optimize the model for hardware. When the model is placed in a subsystem, the Fixed-Point Designer can be started. The relative tolerance is set to 1% for the output voltage. Like has been explained in Section 2.2, it is important that the tool can collect the complete range that the internal signals will have during run-time. Using step source blocks in MATLAB that are added, and the sum is connected to the input ports of the model. This is done for both V_{in} (initial value 2.5V, range [2.3, 2.7]) and R (initial value 330Ω, range [280,430]). The load value exceeds the maximal rated value that was chosen in Subsection 3.2.2, which is done deliberately to be more certain that all ranges collecting the ranges.

After the tool collected information about all the ranges of the signals, the word-length was set to 32 bits for a first embedded run. However, under flows caused unstable behaviour. After multiple iterations it was found that a word-length of 38 is sufficient for the most demanding signals. No optimization was done to reduce word-length for signals that need less precision, for example constants. The comparison of the output voltage between the double data-type run (BaselineRun) and the fixed-point run (EmbeddedRun) can be seen in Figure 4.4, together with the 1% tolerance interval. The fixed-point model stays within the 1% tolerance. This means that this model it suitable for the continuation of the design method, which will be converting the model to VHDL and synthesizing for FPGA.

Figure 4.4: Output voltage of fixed-point model and floating-point model

VHDL implementation

The input and output ports of the analog plant were converted to 32 bit, which gives no noticeable difference on system performance. As has been explained before, the default 38 bit word-length is not necessary for all system parameters. The 32 bit ports makes connecting with the 32 bit controller possible without the need for additional conversion blocks outside the analog and digital blocks. The duty cycle value was converted to a 6 bit word-length, since the value is unsigned and never exceeds 50.

The HDL coder was used to export the VHDL code of the model that was shown in Figure 4.1. The variload subsystem is exported as a separate entity in a separate VHDL file, but this requires no special attention. The generated VHDL file is shown in Appendix A, and gives an impression of what code generated by the HDL coder looks like.

One block in the model required some attention: the model contains a division block. No extensive research was done on division on FPGA. The HDL coder gave an error for this block: ”Product block with divide input (/) supports fixed point division only when the fraction length of output is equal to the fraction length of the dividend minus the fraction length of the divisor.” This was solved by increasing the fraction length of the dividend, using a data conversion block. If the division would require multiple clock cycles per calculation, it would still be possible to have it in the design:

the gain parameter would not be able to change during these calculations, but the last calculated value can be used for multiplication with the current input value for the number of cycles that a division takes.

A block diagram of the structural design of the hardware in the VHDL test bench is shown in Figure 4.5. As has been explained in the previous section, the controller exists of a VHDL generated PI controller and a manually implemented PWM generator. The VHDL code of the PWM generator can be found in Appendix B. The test-vector controller (TVC) provides the clock and reset signals. Note that there is only one main clock of 50MHz.

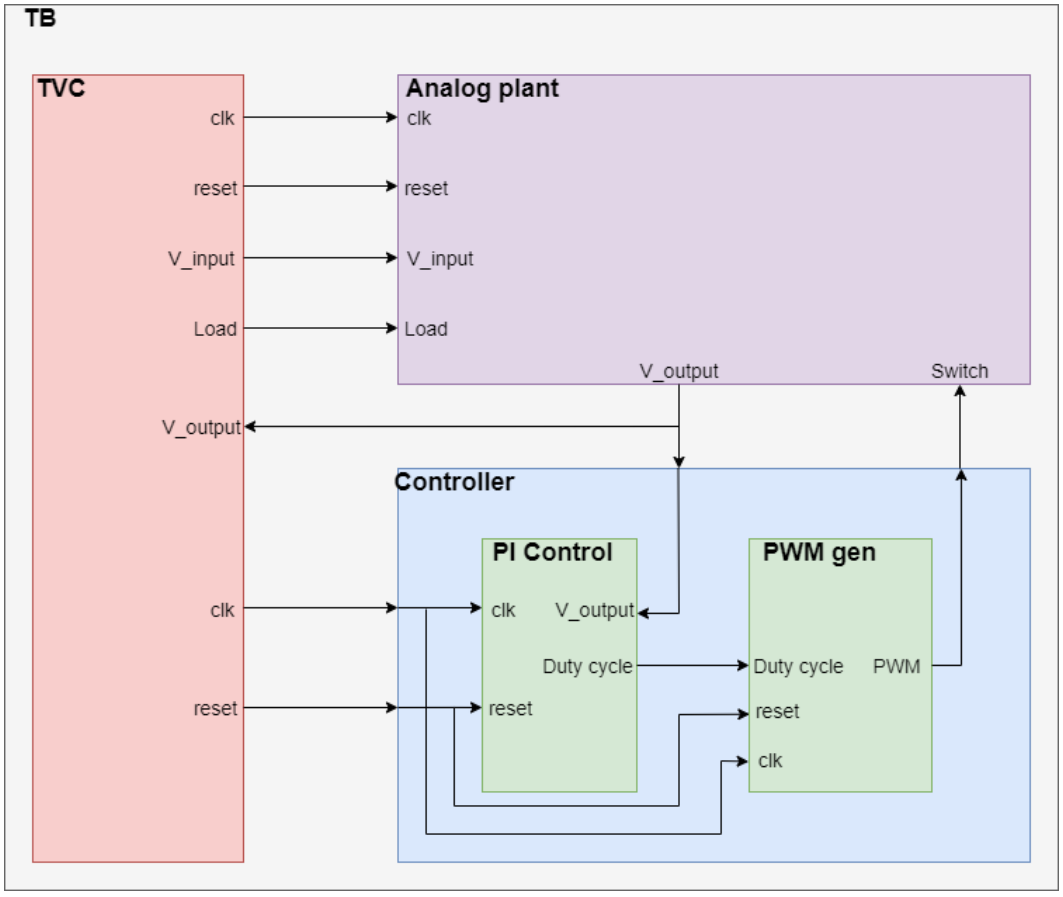


Figure 4.5: Structural hardware design of the DC-DC boost converter

Following the above steps in this chapter result in a VHDL implementation of the emulated DC-DC boost converter can be realized that can compile and run using ModelSim. Because of time restrictions, it was not possible to synthesize the design for FPGA.

The VHDL entity contains 46 internal signals. These signals are all 38 bits wide. This shows that even for a simple circuit with a few components, already quite a lot of resources are needed.

Chapter 5

Results

This chapter shows the performance of the hardware-emulated analog power plant of the DC-DC converter. Before the complete system will be tested, the behaviour of the emulated analog plant without feedback controller is compared to an analog simulation on a CPU-based machine. After this, the controller is connected to the analog plant and performance is compared to the reference simulation.

All results come from ModelSim simulations. The model that was used is shown in Appendix C. The reference data is generated using Simulink simulation. The Simscape Toolbox [13] is used for this. It provides physical system modeling and simulation, which includes electrical circuits.

Since the goal of the emulation is to get similar behaviour at the ports of the system, only the behaviour of the output voltage is evaluated.

5.1 Analog subsystem performance

This first section will validate the stand-alone performance of the emulated analog circuit. A control signal is still needed in order to simulate the circuit. This will be done with a constant duty cycle control signal. By not having a feedback controller in the simulation, the error between reference data and results of the hardware emulation has no external causes. A constant PWM source with a duty cycle of 24% is used to drive the analog plant. The output load is initially 330Ω, and is changed at $t = 0.5ms$ to 430Ω. Since no feedback control is present, the output voltage will change when the load changes.

The comparison of the voltage output of the ModelSim hardware simulation and Simulink simulation for the above described test can be seen in Figure 5.1. The RMSE is 0.025V (average relative error 0.69%) and the maximum error is 0.058V (relative error 1.39%).

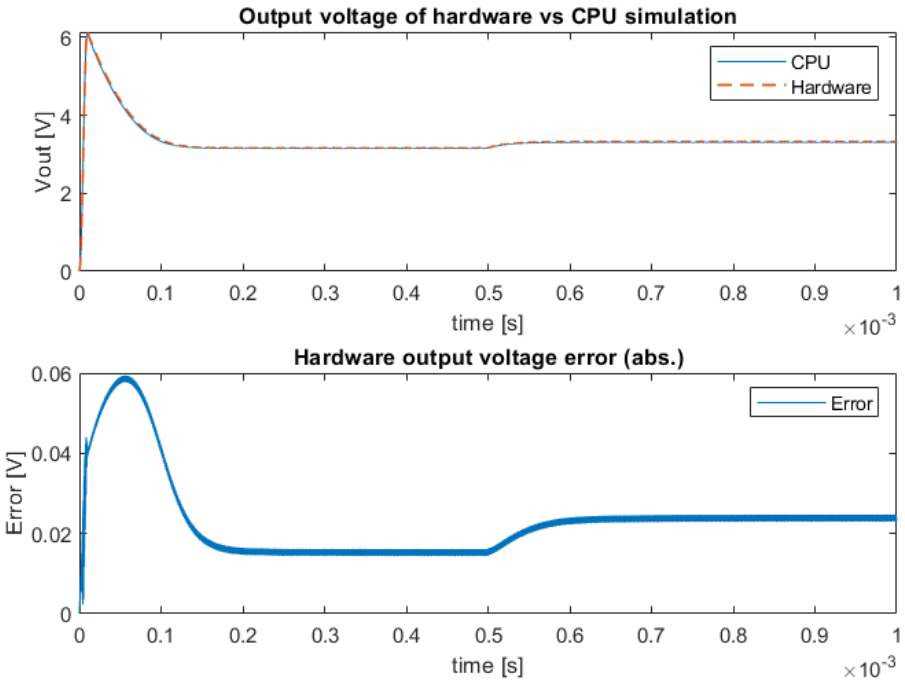


Figure 5.1: Constant duty cycle control signal, changing load at $t = 0.5ms$

5.2 Controller validation performance

In Section 5.1, it was shown that the digitally emulated analog plant of the DC-DC boost converter has similar dynamic behaviour compared to analog simulation. This means that it should be usable for validating a controller design. In the following simulation, performance of the emulated analog circuit that is controlled using the PI controller that was designed in Section 4.2 is evaluated. This is again done by looking at the output voltage of the DC-DC boost converter. In addition to this, the output of the controller is also compared to the controller output in the Simulink simulation. The output of the controller is a duty cycle value that goes to the PWM generator.

The input voltage will be the deviating parameter in this test, to see how the controller responds to changing circuit parameters. The initial value is $2.5V$ and is changed to $2.7V$, $2.3V$ and back to $2.5V$. The resulting output voltages with the absolute error can be seen in Figure 5.2. The controller duty cycle outputs can be seen in Figure 5.3.

In this test, the RMSE is $0.0128V$, (average relative error of 0.56%), and the maximum error is $0.056V$ (relative error 1.89%). The RMSE is lower than in the previous test. The likely reason behind this is that the controller also compensates

the error that was found in Section 5.1. For the controller, it can be seen that the output of the hardware controller has some differences compared to the simulation controller, especially for the steady state.

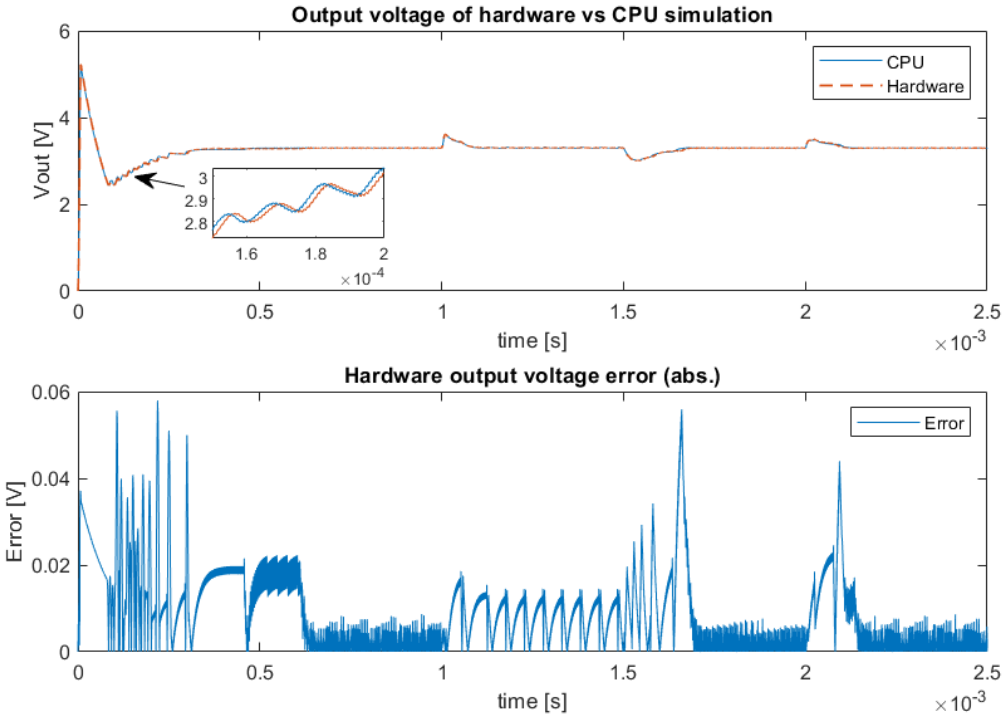


Figure 5.2: Performance with controller, changing V_{in} at $t = 1ms, 1.5ms, 2ms$

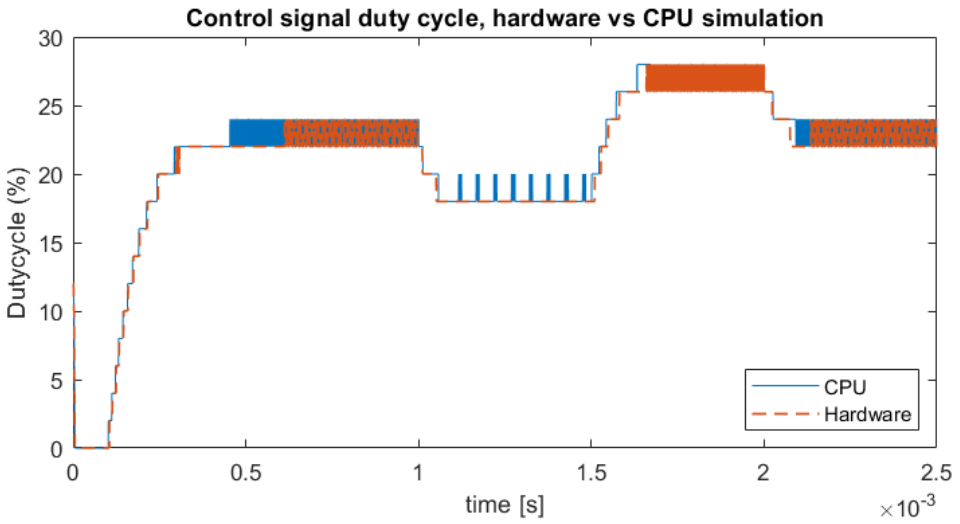


Figure 5.3: Controller duty cycle, changing V_{in} at $t = 1ms, 1.5ms, 2ms$

In Figure 5.4, the output voltage together with the switch state are plotted over an interval of 200 clock cycles. The data comes from ModelSim simulation. In Appendix D, the ModelSim simulation interface shown as well. The DC-DC converter is in the steady state. The individual time steps are visible. Every switching period, 50 values are calculated for every parameter in the analog plant.

The behaviour of the DC-DC boost converter can be seen nicely seen in Figure 5.4 as well. The capacitor is discharging at a constant rate. When the switch is closed, the current in the inductor is increasing (this cannot be observed in the plot). This current is then transferred into to capacitor when the switch opens again, resulting in a increasing voltage again.

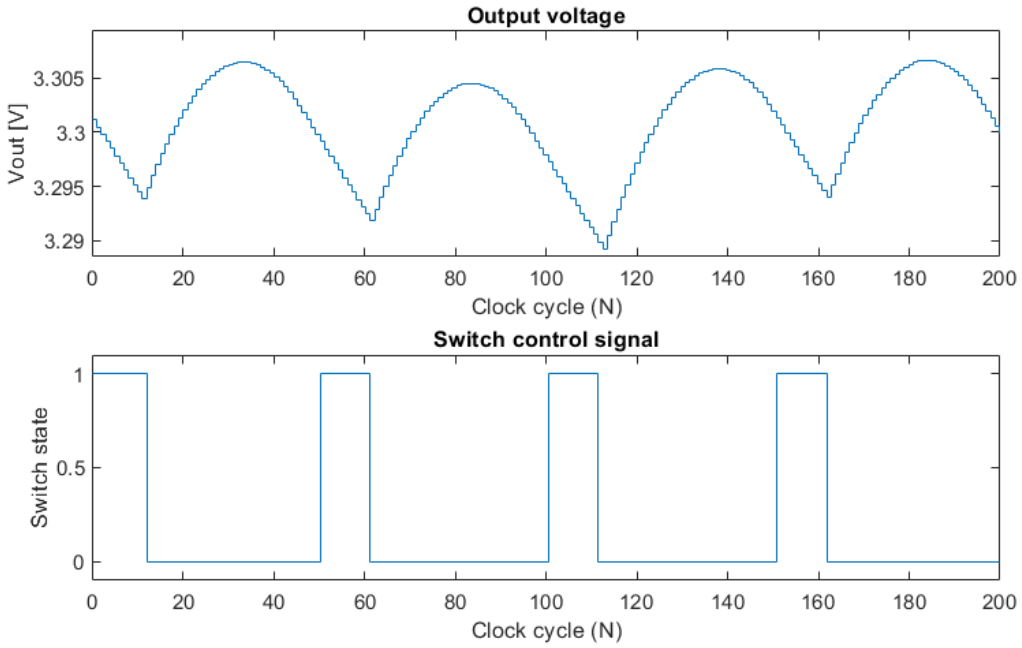


Figure 5.4: Output voltage and switch state during 4 switching periods

Chapter 6

Conclusions and recommendations

This chapter will evaluate the method and corresponding results of this thesis. The research questions will be answered. After this my view on the continuation of the research on the concept that has been presented in this thesis is given.

6.1 Conclusion

The first subquestion that was defined in this thesis was: ”How can the continuous behaviour of an analog circuit be discretized” These first steps were based on manual calculations. The dynamic behaviour of the circuit must be analysed to obtain the differential equations. These dynamics were converted to difference equations by numerically approximating them using the forward Euler method.

The second subquestion that was defined as following: ”How can a discrete model be synthesized on FPGA”. Once the discrete model was made in Simulink, the method changed its approach to mainly using software tools. The model was optimized for FPGA using the fixed-point and HDL coder toolboxes, together with other built-in MATLAB/Simulink features. However, no word-length optimization was done, resulting in a word-length of 38 bits for all signals. Once the HDL was exported, the last step was to synthesize the design for FPGA. This step was not applied on the case-study because of time restrictions. The real-time advantage of running the system on FPGA has thus not been tested yet. Also, the impact of word-length on the resources needed on FPGA could not be evaluated. For this simple circuit, already 46 internal signals were needed. This fact, in combination with the fact that the current word-length is 38 bits, shows that quite a lot of resources would be needed for emulation, even for a small circuit.

The third subquestion was defined for performance validation: ”How does performance of analog simulation on FPGA compare to CPU-based simulation”. To answer this question, the method has been applied on an ideal DC-DC boost con-

verter circuit. The resulting hardware design has been compared to analog Simulink simulations. An average relative error of 0.56% was achieved, which shows that the hardware design gives accurate results.

With the subquestions answered, this research can be concluded by answering the main research question: ”How can FPGA emulation of the analog subsystems of mixed-signal ICs contribute to the design validation of the digital subsystems?”. The simplest question one would want to answer when validating a digital subsystem is ’does it work’. For the DC-DC boost converter, the outcome of this question is similar for the Simulink simulation and hardware design: in both cases the controller worked as intended.

Of course, validating controller designs goes much further than that simple question. While no advanced validation of the digital controller of the DC-DC controller has been done in this thesis, it is proven that the used method results in hardware that accurately approximates the behaviour of the analog system. It is therefore expected that the resulting design can be used to do more advanced validation of digital subsystems, if FPGA synthesis is possible.

Parallel processing on FPGA can give time advantages over CPU-based simulation. The digital subsystem can be validated on FPGA without the need for interfacing with a CPU-based machine, making real-time validation possible. However, the word-length and number of the internal signals of the emulation hardware will be a determining factor. Optimization of fixed-point word length is important to keep the needed resources on FPGA achievable. However, large circuits and circuits that contain lots of parasitics will result in hardware designs that are not synthesizable because of the needed resources.

For high-frequency circuits (e.g. radio receivers that operate at frequencies higher than 1GHz), FPGA emulation could also be a great solution for digital subsystem validation. However, the emulated circuit will not be able to run real-time, since the clock speed of FPGA is limited. The time scale would need to be moved down, so that there will be enough iterations for the analog emulated hardware for every digital subsystem clock cycle.

6.2 Recommendations

The results of this thesis show that it is possible to emulate analog subsystems on hardware with the goal of validating digital subsystem performance. While the overall idea of the steps in the presented method will work in most cases, they have not been optimized to be used in real development applications yet.

There are many steps in the presented method. Since time was limited, there was no time to research the accuracy and optimization in these steps. The first

step that could be further researched is the discretization: using the parallelism of FPGA, more accurate discretization methods might be possible. Or, in the case of a discretization method that needs multiple clock cycles per calculation, the clock speed can be a factor that also can be researched further: Using phase-locked loops, higher clock speeds might be possible, possibly improving accuracy.

The fixed-point conversion has also not been optimized in this research. The word-length was chosen based on the highest-demanding parameters, and not optimized per individual parameter. Doing this can greatly reduce the resources needed on FPGA, making it possible to emulate larger circuits.

In addition to these steps, the current design has not been synthesised for a FPGA yet. It was originally planned to do this in this thesis, but this was not achieved because of time restrictions. Synthesis would give insight in the resources that are needed for emulating hardware.

In the conclusion it was emphasized that the first steps in the used method have to be done manually. While this is possible for simple circuits like the DC-DC boost converter that was used as a case-study, it becomes harder or even impossible to do for more complex circuits.

When all of the above steps are further researched, a method with more efficient and more accurate results can be achieved. This could even be the fundamental theory behind an all-in-one software solution that can automatically export HDL of analog circuit layouts. This would make the concept interesting for mixed-signal IC design in the industry.

Bibliography

[1] R. Bhattacharya, S. Biswas, and S. Mukhopadhyay, "FPGA based chip emulation system for test development of analog and mixed signal circuits: A case study of DC-DC buck converter," *Measurement: Journal of the International Measurement Confederation*, vol. 45, no. 8, 2012.

[2] W. Nzale, J. Mahseredjian, I. Kocar, X. Fu, and C. Dufour, "Two variable time-step algorithms for simulation of transients," in *2019 IEEE Milan PowerTech, PowerTech 2019*, 2019.

[3] M. A. Zidan, A. G. Radwan, and K. N. Salama, "The effect of numerical techniques on differential equation based chaotic generators," in *Proceedings of the International Conference on Microelectronics, ICM*, 2011.

[4] H. Zhang and J. Sun, "FPGA-based simulation of power electronics using iterative methods," in *2014 International Power Electronics Conference, IPEC-Hiroshima - ECCE Asia 2014*, 2014.

[5] J. Mart, "Digital Control Techniques for DC / DC Power Converters," *(Unpublished master's dissertation), Helsinki University of Technoligy*, 2009.

[6] "Fixed-point designer." [Online]. Available: https://nl.mathworks.com/products/fixed-point-designer.html

[7] "Hdl coder." [Online]. Available: https://nl.mathworks.com/products/hdl-coder.html

[8] J. Wang, "Design of a Boost DC-DC Converter for Energy Harvesting Applications in 40nm CMOS Process," *IEEE Transactions on Power Electronics*, vol. 33, no. November, 2014.

[9] N. Bajoria, P. Sahu, R. K. Nema, and S. Nema, "Overview of different control schemes used for controlling of DC-DC converters," in *International Conference on Electrical Power and Energy Systems, ICEPES 2016*, 2017.

[10] “Control system toolbox.” [Online]. Available: https://nl.mathworks.com/products/control.html

[11] “System identification toolbox.” [Online]. Available: https://nl.mathworks.com/products/sysid.html

[12] A. Platon, S. Oprea, A. Florescu, and S. G. Rosu, “Simple and Digital Implementation of PI Controller Used in Voltage-Mode Control,” in *Proceedings of the 10th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2018*, 2019.

[13] “Simscape.” [Online]. Available: https://nl.mathworks.com/products/simscape.html

Appendix A

VHDL code of the variload subsystem

```
1  -- -------------------------------------------------------------
2  --
3  -- File Name: hdlsrc\DCDC_discrete_fixed_variload_control_simpified\variload.vhd
4  -- Created: 2021-06-14 19:43:40
5  --
6  -- Generated by MATLAB 9.10 and HDL Coder 3.18
7  --
8  -- -------------------------------------------------------------
9
10
11 -- -------------------------------------------------------------
12 --
13 -- Module: variload
14 -- Source Path: DCDC_discrete_fixed_variload_control_simpified/dcdc_analog_variload/variload
15 -- Hierarchy Level: 1
16 --
17 -- -------------------------------------------------------------
18 LIBRARY IEEE;
19 USE IEEE.std_logic_1164.ALL;
20 USE IEEE.numeric_std.ALL;
21
22 ENTITY variload IS
23   PORT( LOAD                              :   IN    std_logic_vector(37 DOWNTO 0);  -- ufix38_En29
24         In2                               :   IN    std_logic_vector(37 DOWNTO 0);  -- ufix38_En35
25         Out1                              :   OUT   std_logic_vector(37 DOWNTO 0)  -- ufix38_En35
26         );
27 END variload;
28
29
30 ARCHITECTURE rtl OF variload IS
31
32   -- Signals
33   SIGNAL In2_unsigned                     : unsigned(37 DOWNTO 0);  -- ufix38_En35
34   SIGNAL Constant1_out1                   : unsigned(37 DOWNTO 0);  -- ufix38_En37
```

```
  SIGNAL Constant_out1                    : unsigned(37 DOWNTO 0);  -- ufix38_En63
  SIGNAL Data_Type_Conversion_out1        : unsigned(74 DOWNTO 0);  -- ufix75_En100
  SIGNAL LOAD_unsigned                    : unsigned(37 DOWNTO 0);  -- ufix38_En29
  SIGNAL Gain1_mul_temp                   : unsigned(75 DOWNTO 0);  -- ufix76_En88
  SIGNAL Gain1_out1                       : unsigned(37 DOWNTO 0);  -- ufix38_En50
  SIGNAL Divide1_out1                     : unsigned(37 DOWNTO 0);  -- ufix38_En50
  SIGNAL Add1_sub_cast                    : unsigned(37 DOWNTO 0);  -- ufix38_En37
  SIGNAL Add1_out1                        : unsigned(37 DOWNTO 0);  -- ufix38_En37
  SIGNAL Product_mul_temp                 : unsigned(75 DOWNTO 0);  -- ufix76_En72
  SIGNAL Product_out1                     : unsigned(37 DOWNTO 0);  -- ufix38_En35

BEGIN
  In2_unsigned <= unsigned(In2);

  Constant1_out1 <= unsigned'("10000000000000000000000000000000000000");

  Constant_out1 <= unsigned'("10101011110011000111011100010001100001");

  Data_Type_Conversion_out1 <= Constant_out1 & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
     & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0'
     & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0' & '0';

  LOAD_unsigned <= unsigned(LOAD);

  Gain1_mul_temp <= unsigned'("10100001000011111010111110100000011011") * LOAD_unsigned;
  Gain1_out1 <= Gain1_mul_temp(75 DOWNTO 38);

  Divide1_output : PROCESS (Data_Type_Conversion_out1, Gain1_out1)
    VARIABLE c : unsigned(74 DOWNTO 0);
    VARIABLE div_temp : unsigned(74 DOWNTO 0);
  BEGIN
    div_temp := to_unsigned(0, 75);
    IF Gain1_out1 = to_unsigned(0, 38) THEN
      c := unsigned'("11111111111111111111111111111111111111111111111111111111111111111111111111111111111111");
    ELSE
      div_temp := Data_Type_Conversion_out1 / Gain1_out1;
      c := div_temp;
    END IF;
    IF c(74 DOWNTO 38) /= "00000000000000000000000000000000000000" THEN
      Divide1_out1 <= "1111111111111111111111111111111111111";
    ELSE
      Divide1_out1 <= c(37 DOWNTO 0);
    END IF;
  END PROCESS Divide1_output;


  Add1_sub_cast <= resize(Divide1_out1(37 DOWNTO 13), 38);
```

```
82    Add1_out1 <= Constant1_out1 - Add1_sub_cast;
83
84    Product_mul_temp <= In2_unsigned * Add1_out1;
85    Product_out1 <= Product_mul_temp(74 DOWNTO 37);
86
87    Out1 <= std_logic_vector(Product_out1);
88
89  END rtl;
```

Appendix B

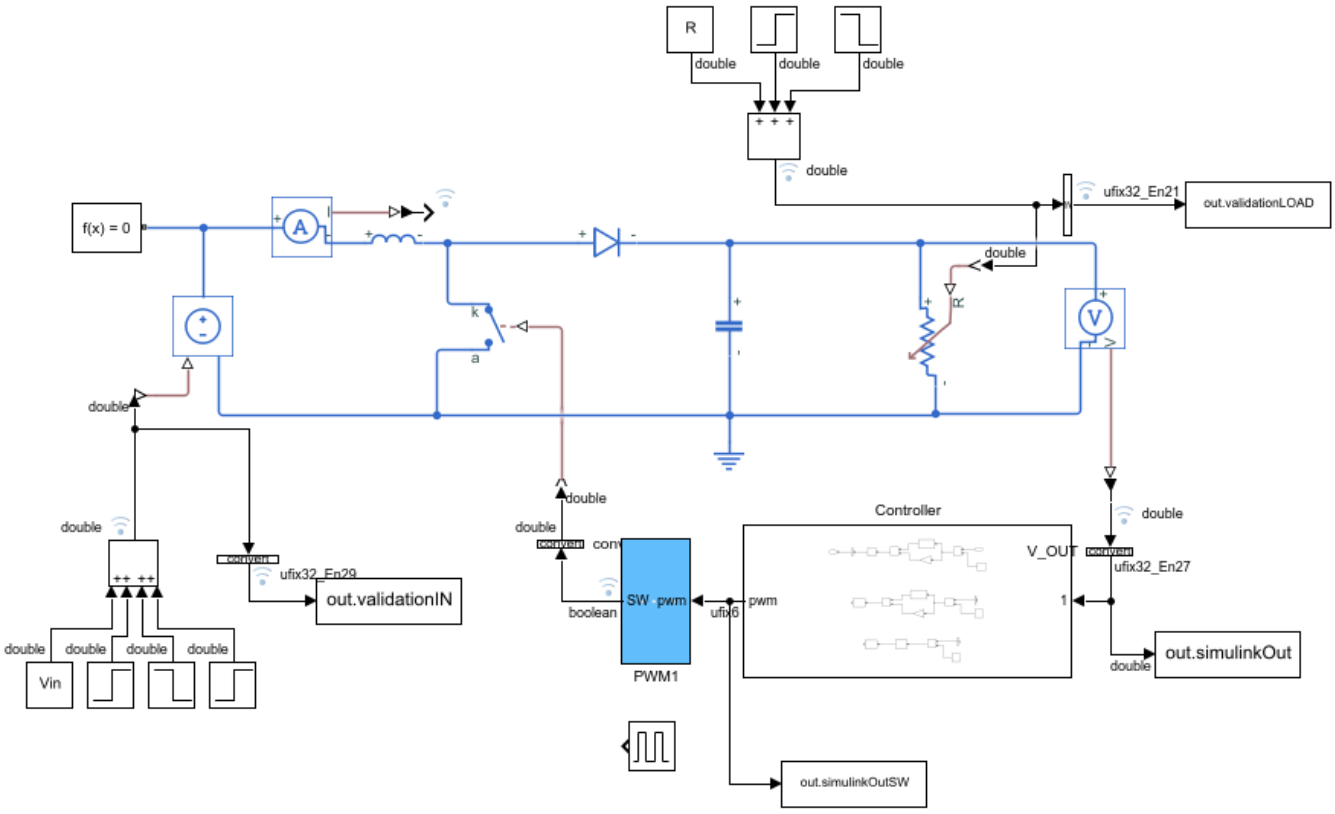
VHDL code of the PWM generator

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use work.all;
5
6  entity dcdc_pwmgen is
7      PORT( clk                          :   IN    std_logic;
8          clk_enable                     :   IN    std_logic;
9          reset                          :   IN    std_logic;
10         duty                           :   IN    std_logic_vector(5 DOWNTO 0);
11         switch_control                 :   OUT   std_logic
12     );
13 end dcdc_pwmgen;
14
15
16 architecture behaviour of dcdc_pwmgen is
17     constant sw_period : integer := 50;
18     signal counter : integer range 0 to sw_period-1;
19     signal duty_int : integer range 0 to sw_period-1;
20 begin
21
22     process(clk, reset)
23     begin
24         if reset = '1' then
25             counter <= 0;
26         elsif rising_edge(clk) then
27             if(clk_enable = '1') then
28                 if counter = 49 then
29                     counter <= 0;
30                 else
31                     counter <= counter + 1;
32                 end if;
33                 duty_int <= to_integer(unsigned(duty));      -- Use controller value
34                 -- duty_int <= 15;                             -- Use constant value
```

```
35              end if;
36          end if;
37      end process;
38
39      switch_control <= '1' when counter < duty_int else '0';
40
41  end behaviour;
42
```

Appendix C

Simulink analog model



Appendix D

ModelSim simulation