Formal Verification of Steady-State Errors in Unity-Feedback Control Systems

Muhammad Ahmad and Osman Hasan

School of Electrical Engineering and Computer Science (SEECS),
National University of Sciences and Technology (NUST),
Islamabad, Pakistan
{muhammad.ahmad,osman.hasan}@seecs.nust.edu.pk

Abstract. The meticulousness of steady-state error analysis of unity-feedback control systems has always been of vital significance as even a trifling glitch in this analysis may result in grievous penalties. To ensure a rigorous steady-state error analysis, this paper presents the formal verification of a generic relationship that is applicable to all kinds of inputs and types of unity-feedback control systems. This formalization builds upon the multivariate calculus theories of HOL-Light and our prior work on developing formal models of feedback control systems. To illustrate the usefulness of this result, the paper presents the formal steady-state error analysis of a Pulse Width Modulation (PWM) push-pull DC-DC converter, which is an extensively used component in various power-electronics and aerospace applications.

1 Introduction

Control systems [18] form an integral part of all automated systems used in a wide range of safety-critical applications, including industrial automation, surgical robots, automobiles and aerospace systems. These control systems work along with the given systems (plants) and are designed in such a way that they ensure the desired behavior of their corresponding systems while adhering to the stability constraints and allowable error margins.

Control systems can be configured in an open or a closed loop topology [18]. In open-loop systems, the controller generates the control signals based on a reference or input signal $R(s)$, as shown in Fig. 1.a . A disadvantage of this kind of configuration is that the controller has no information about the output of the plant (with open-loop transfer function $G(s)$) and thus cannot cater for unexpected disturbances. To overcome this limitation, control systems are often configured in a feedback or closed loop pattern where the output of the plant $C(s)$ is measured and compared with a reference or an input signal $R(s)$, as shown in Fig. 1.b. This error signal $E(s)$ is then used for decision making in the controller to compensate for disturbances. A unity-feedback is a frequently used closed-loop system where the output of the system is compared with the reference input signal as is, i.e., without any gain or loss in the feedback path.

The quality of the control system is judged based on its steady-state response [19], i.e., the response of the system when a large number of iterations in the closed-loop have taken place and the steady-state conditions have been attained. Steady-state error gives a parametric measure for the controllability of system and how well the system will respond to certain disturbances.

The steady-state analysis of unity-feedback control systems is performed in the Laplace domain because this choice allows us to model the main system in terms of the transfer functions of its sub-systems, as a block diagram. The overall transfer function of the plant $G(s)$ is then expressed as follows by manipulating the transfer functions of its subsystems using a set of predefined rules[18]:

$$G(s) = \frac{1}{s^b} \frac{Y(s)}{Z(s)} \tag{1}$$

where the integer variable $b : 0, 1, 2 \cdots$ categorizes the system type or the number of integrators in the forward path[18], and $Y(s)$ and $Z(s)$ represent the zeros and poles of $G(s)$ apart from $\frac{1}{s^b}$. Now, the net transfer function for unity-feedback error model is mathematically expressed as[18]:

$$E(s) = \frac{R(s)}{1 + G(s)} \tag{2}$$

where $R(s)$ models the input to our system, which in the case of steady-state error analysis is traditionally taken to be as the *unit step* ($\frac{1}{s}$), *ramp* ($\frac{1}{s^2}$) and *parabola* ($\frac{1}{s^3}$) functions. The steady-state error is measured at a very large time, i.e., when the time t tends to infinity. Thus, it can be defined in the Laplace domain by applying the Final Value Theorem to the error model:

$$e_\infty = \lim_{s \to 0} sE(s) \tag{3}$$

Traditional methods, like paper-and-pencil proof methods and computer simulations and numerical methods, cannot guarantee the accuracy of the above-mentioned steady-state error analysis. The paper-and-pencil based analysis methods are error prone due to the human involvement. Moreover, it is quite often the case that many key assumptions of the results obtained of sub-system using paper-and-pencil proof methods are not documented, which may lead to

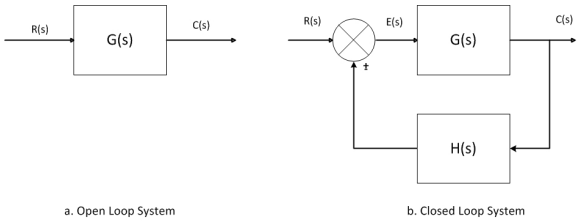

Fig. 1. Control System Configurations

to erroneous systems. Computer simulations and numerical methods, such as MathWorks Simulink [17], cannot guarantee accurate results while dealing with feedback-control systems mainly due to their inherent non-exhaustive nature coupled with the imprecision of computer arithmetics. The mathematical models of control systems can also be analyzed in computer algebra systems (CAS), such as Mathematica [15]. CAS are very efficient for computing mathematical solutions symbolically, but are also not completely reliable due to the presence of unverified huge symbolic manipulation algorithms.

In order to overcome the above-mentioned limitations, the usage of formal methods in the safety-critical domain of control system analysis is increasingly being investigated [20, 3]. However due to the continuous nature of the steady-state error analysis, automated theorem provers and model checking tools cannot ascertain absolute precision of analysis. Higher-order logic theorems provers have shown some promising results and a detailed review of the literature will be presented in the next section. One of the most interesting contributions, related to the formal steady-state analysis of control systems, is the higher-order-logic formalization of the basic building blocks[12], like forward transfer functions, summing junctions, feedback loops and pickoff points, of control systems using the multivariate analysis theories available in the HOL-Light theorem prover. These foundations can be built upon to formally specify a wide range of control systems in higher-order logic and reason about their steady-state errors within the sound core of a theorem prover. The process involves the verification of the error function for the given system based on its structure and the behavior its sub-blocks. This is followed by the verification of the limit of the error function of the given system, according to Equation (3), using the multivariate analysis libraries of HOL-Light. The approach was illustrated by verifying the steady-state error of a solar tracking control system. However, the reasoning process about the steady-state error model with closed loop transfer function T(s), shown in Fig. 2, is very cumbersome, the reason being the extensive user interaction requirement in verifying the limiting behavior, expressed in Equation (3), for the net expression for the error model of the given system. Moreover, this reasoning process has to be repeated all over again if the steady-state for a different type of input (unit step, ramp or parabola) is required for the same system, which is a very common occurrence in steady-state error analysis.

The main scope of this paper is to overcome the above mentioned issues. We build upon the formalization of the control system blocks of [12] to formalize the error model for the unity-feedback control systems. Moreover, we formally verify a generic expression for the steady-state error of unity-feedback control

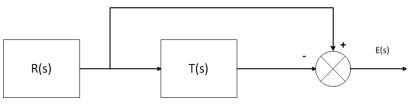

Fig. 2. Steady-State Error Model

systems using the multivariate analysis theories of HOL-Light. The unique feature of this expression is that it can be used to reason about the steady-state error of any system type and input. Moreover, it facilitates reusability when reasoning about the state-state error of the same system while considering different inputs. The quest for minimizing the user interaction in the higher-order-logic theorem-proving based analysis for steady-state errors led us to develop this useful relationship, which to the best of our knowledge has not been reported in the control systems literature before. In order to illustrate the utilization and practical effectiveness of our formalization for verifying real-world control systems, we use it to conduct the steady-state error analysis of the Pulse Width Modulation (PWM) push-pull DC-DC converters[8], which is a widely used component in power electronics and many safety-critical aerospace applications. In order to evaluate the usefulness of our work for control system engineers, we engaged a domain expert; trained her with basic theorem proving abilities in a couple of weeks and assigned her the task to use our formalization for analyzing the PWM push-pull DC-DC converter and her experiences are also shared in this paper.

2 Related Work

ClawZ [4] allows us to translate models of control systems developed in MathWorks Simulink into Z language specifications, which are then verified by proving the equivalence of the controller implementation using Ada in ProofProver. Another similar approach is presented in [1] in which the author translates the discrete-time Simulink model to Circus notations, which combines Z language and refinement calculus and then compares a parallel Ada implementation. An interesting methodology adopted in [7] calls for using the Timed Interval Calculus (TIC) library to capture the behavior of Simulink blocks, which could be verified in a theorem prover. A similar approach was adapted by Mahony, and modeling and analysis of feedback control systems was introduced using the DOVE environment [16]. Model checking has also been successfully used to analyze dynamic systems by abstracting the behavior of the system to a state-space model [22]. Herencia-Zapana [13] proposed to formally analyze control software properties by first expressing the stability proofs as C code annotations and then translating them to PVS proof obligations and automatically verifying them. All these pioneering frameworks are based on automatic formal verification tools and thus require some sort of abstraction mechanism to model the exact behavior of real-world control systems and their environments, which are always continuous in nature.

In order to formally model and analyze continuous models of control systems, Boulton et al. provided some reasoning support for verifying frequency response of continuous-time control systems using Hoare logic using the HOL98 theorem prover [6]. The main idea is to reason about the gain and phase relationships of a control system using the gain and phase relationships of its subsystems in the block diagram. This framework does not provide generic functions to model arbitrary block diagrams for control systems and also lacks reasoning support

for complex number analysis principles, such as limits and summation, which are essential to reason about many control system design related parameters, such as steady-state errors and stability. In order to overcome these shortcomings, Boulton et al [5] proposed to use automated symbolic methods to replace the classical graphical charts, such as Nichole and Bode plots along with their formal models. Based on this principle, the authors developed a prototype tool using Maple and the PVS system. Maple is used to compute the verification conditions for the given control system and PVS is used to discharge these conditions using theorem proving principles. Due to the usage of Maple, the accuracy of the analysis is again somewhat compromised as has been mentioned above.

The foremost foundation of analyzing the steady-state error of control systems is the formalization of complex number analysis theories. The multivariate calculus theories of HOL-Light theorem prover [11] fulfill this requirement. These theories have been recently used to formalize the basic building blocks of control systems [12] and the Laplace theory [21], which are the most relevant contributions to our work. We build upon and enhance the results reported in [12] to analyze steady-state errors of unity-feedback control systems and facilitate the formal reasoning process by verifying a generic expression for steady-state error in this paper. The recent formalization of Laplace theory[21] opens up many interesting research directions in the context of our work since now we can link our formalization to the time-domain as well.

3 Preliminaries

In this section, we give a brief introduction to the multivariate analysis theories in the HOL-Light theorem prover and the block diagram formalization of [12]. The intent is to provide some preliminaries to make the paper self contained and thus facilitate its understanding for a wider audience, including both formal methods and control communities.

3.1 Multivariate Calculus Theories in HOL-Light

A n-dimensional vector is represented as a $\mathbb{R}^n$ column matrix of real numbers in HOL-Light. All of the vector operations are then handled as matrix manipulations. This way, complex numbers can be represented by the data-type $\mathbb{R}^2$, i.e, a column matrix having two elements [9]. In this formalization of complex numbers, the first real number represents the real part and the second real number represents the imaginary part of the given complex number[10]. The main advantage of this choice is that all the topological and analytic formalization developed for vectors is inherited by the complex numbers.

Definition 1: *Complex Number*
`⊢ ∀ x y. complex (x,y) = vector [x; y]`

The following mappings allow us to obtain the real and imaginary components of a complex number:

Definition 2: *Real and Imaginary Components of a Complex Number*
⊢ ∀ z. Re z = z$1
⊢ ∀ z. Im z = z$2

Here the notation z$n represents the n^{th} component of a vector z. A real number a can be converted to an equivalent complex number as follows:

Definition 3: *Cx*
⊢ ∀ z. Cx(a) = complex(a,&0)

The normalization of a complex number is also a widely used phenomena and has been formalized in HOL-Light [10] as follows:

Definition 4: *Normalization of a Complex Number*
⊢ ∀ z. norm z = sqrt (Re z pow 2 + Im z pow 2)

where sqrt represents the HOL-Light square root function for real numbers.

The concept of limit of a function is used in our formalization to model the steady-state error and is formalized in HOL-Light as follows:

Definition 5: *Limit of a function*
⊢ ∀ f net. lim net f = (@l. (f → l) net)

The function lim is defined using the Hilbert choice operator @ in the functional form. It accepts a *net* with elements of arbitrary data-type A and a function f, of data-type $A \rightarrow \mathbb{R}^m$, and returns l:$\mathbb{R}^m$, i.e., the value to which the function f converges to at the given net.

Similarly, we also use the following theorem in our development:

Theorem 1: *Sum of a geometric Progression*
⊢ ∀ z. norm z < &1 ⇒
((λk.z pow k) sums z pow n / (Cx(&1) - z)) (from n)

Where the function f sums k (from n) ensures that the infinite summation of a multivariate sequence f is equal to k with n as the starting point.

3.2 Formalization of Block Diagrams in Control Systems

This section provides a set of formal definitions [12] of the basic building blocks of control systems, given in Fig. 3. These definitions can in turn be used to formalize a wide range of control systems in higher-order logic. The net transfer function of n subsystems connected in *cascade* is the product of their individual laplace transfer functions (Fig. 3.a).

Definition 6: *Cascaded Subsystems*
⊢ series [] = Cx (&1) ∧ (∀ h t. series (CONS h t) h * series t)

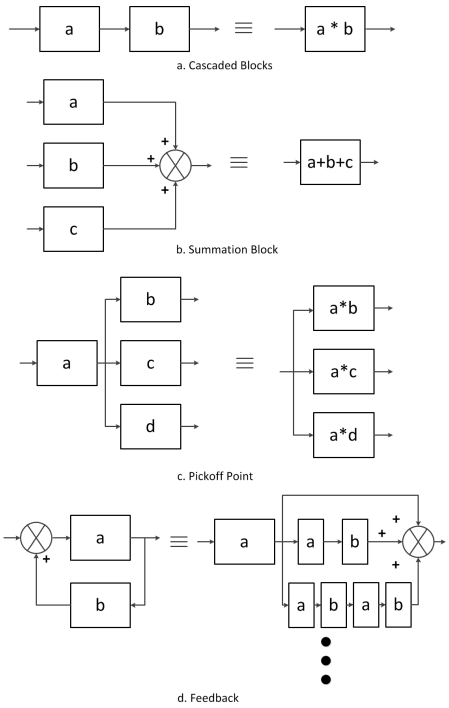

Fig. 3. Basic Building Blocks of a Control System

The function `series` accepts a list of complex numbers, corresponding to the transfer functions of all the given subsystems, and recursively returns their product. Here two type injections `&` and `Cx` are used to transform a positive integer to its corresponding real and complex number, respectively.

Fig. 3.b depicts a *summation junction* of transfer functions where the net transfer functions of a set of incoming branches is formed by adding their individual transfer functions. The formalization of this behavior accepts a list of complex numbers and returns their sum.

Definition 7: *Formalization of Summation Junction*

```
⊢ sum_junction [] = Cx (&0) ∧
  (∀ h t. sum_junction(CONS h t) h + sum_junction t)
```

The *pickoff point* represents a subsystem connected to a network of parallel branches of subsystems (Fig. 3.c):

Definition 8: *Formalization of Pickoff point*

```
⊢ ∀ A h t. pickoff A [] = [] ∧
           pickoff A (CONS h t) = CONS (h * A) (pickoff A t)
```

The function `pickoff`, accepts a complex number `a`, corresponding to the transfer function of the first subsystem, and a list of complex numbers, corresponding to the transfer functions of the subsystems in the branches, and returns a list of complex numbers corresponding to the equivalent block diagram.

The *feedback* block (Fig. 3.d), is the foremost element required to model closed-loop control systems. Due to the feedback signal, it primarily represents an infinite summation of branches that comprises of serially connected subsystems.

Definition 9: *Branch of a Feedback Loop*

```
⊢ ∀ a b n. feedback_branch a b 0 = Cx (&1) ∧
           feedback_branch a b (SUC n) = series [a; b] *
             (feedback_branch a b n)
```

The function `feedback_branch` accepts the forward path transfer function `a`, the feedback path transfer function `b` and the number of the branches `n`. It returns the net transfer function for n branches of a feedback loop as a single complex number. Now, the infinite summation of all branches of the feedback loop can be modeled as the following HOL-Light function:

Definition 10: *Feedback Loop*

```
⊢ ∀ a b. feedback_loop a b = (infsum (from 0)
                              (λk. feedback_branch a b k))
```

The HOL-Light function `infsum (from n) f` above provides the infinite summation a multivariate sequence f with n as the starting point. Now, we can model the behavior of the feedback loop in HOL-Light as follows:

Definition 11: *Feedback*

```
⊢ ∀ a b. feedback a b = series [a; (feedback_loop a b)]
```

The function `feedback` accepts the forward path transfer function `a` and the feedback path transfer function `b` and returns the net transfer function by forming the series network of the summation of all the possible infinite branches and the final forward path transfer function, since the output is taken after the forward path `a`.

A couple of simplification theorems used in this paper, are as follows:

Theorem 2: *Feedback loop simplification*

```
⊢ ∀ a b. norm (a * b) < &1 ⇒ feedback a b = a / (Cx(&1) - a * b)
```

The proof of Theorem 2 is primarily based on the infinite summation of a geometric series [10], given in Theorem 1.

Similarly, the equivalence relationship between the block diagrams, shown in Fig. 4, has been formally verified as follows:

Theorem 3: *Feedback loop simplification*

```
∀ a b c. (norm (a*b) + norm (a*c)) < &1⇒
    feedback a (sum_junction (pickoff Cx(&1) [b;c])) =
                 feedback (feedback a b) c
```

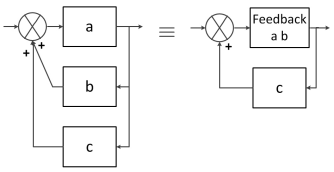

Fig. 4. Multiple Feedback Simplification Rule

The proof of Theorem 3 utilizes Theorem 2 along with some complex arithmetic reasoning. We use this theorem to convert any non-unity-feedback control system into a unity-feedback control system required for steady-state error analysis.

4 Steady-State Error Analysis

We now present the formal verification of a generic expression that can be used to reason about the steady-state error of any unity-feedback system (Fig. 5), irrespective of its type and input. We proceed in this direction by first formalizing a generalized representation of the transfer function according to Equation (1).

Definition 12: *General Transfer function*

```
⊢ ∀ Y Z a. general_tf Y Z b = (λs. Y s / (s pow b * Z s))
```

The function `general_tf` accepts two complex functions Y and Z of data type $\mathbb{R}^2 \to \mathbb{R}^2$ along with a complex number b and returns the transfer function using the lambda abstraction format.

Now, the error model of unity-feedback systems in terms of the generalized representation of $G(s)$, according to Equation (2), is as follows:

Definition 13: *Steady-state-error-model*

```
⊢ ∀ G a. uf_error_model G a =
  (λs. series [Cx (&1) /s pow a; feedback_loop (G s) (--Cx(&1))])
```

The function `uf_error_model` accepts a variable $G : \mathbb{R}^2 \to \mathbb{R}^2$, which represents the general transfer function, and a complex number $a : \mathbb{R}^2$, which generalizes the input type, i.e., if the input is a unit step then $a = 1$ and similarly $a = 2$ and $a = 3$ for the ramp and parabola inputs, respectively. The function uses the functions `series` and `feedback` to capture the structure of the error model of the unity-feedback system, depicted in Fig. 5, and returns its net transfer function with data type $\mathbb{R}^2 \to \mathbb{R}^2$.

Now, the steady-state error can be formally defined as the limit of the net transfer function of the error model, as given in Equation (3),

Definition 14: *steady-state-error*

```
⊢ ∀ E. steady_state_error E = lim (at (Cx(&0))) (λs. s (E s))
```

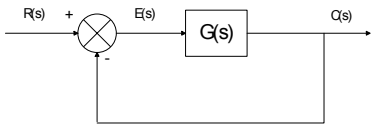

Fig. 5. Steady-State Error of Unity-Feedback Systems

where the function `lim(at(vec i))(λx.f x)`, represents the limit of a function f at point i, i.e., $\lim_{x \to i} f(x)$ in HOL-Light. The function `steady_state_error` accepts a variable $E : \mathbb{R}^2 \to \mathbb{R}^2$, which represents the net transfer function of the error, and returns its corresponding steady-state error as a complex value.

Now, based on the above definitions, we verified our generic expression as the following theorem

Theorem 4: *Unity-feedback steady-state error*

```
⊢ ∀ Y Z a b l m.
   (∀ s. ¬(Z s = Cx(&0))) ∧ ¬(l = Cx(&0)) ∧ ¬(m + l = Cx(&0)) ∧
   0 ≤ b ∧ 1 ≤ a ∧ a ≤ b+1 ∧ (Y → l) (at Cx(&0)) ∧
   (Z → m) (at Cx(&0)) ∧ ((∀ s.norm (Y s / (s pow b * Z s)) <&1)
   ⇒ steady_state_error (uf_error_model (tf_general Y Z b) a) =
      (if b = 0 then m / (m+l) else if a = b+1 then m / l
        else Cx(&0))
```

The first three assumptions are used to avoid singularities. The next two assumptions declare the allowable ranges of the system type and input characterization variables, respectively. The next assumption (a $\leq$ b+1) defines the upper bound of the input type based on the type of the system. The next two assumptions ensure that the variables, l and m, represent the limiting values of the functions Y and Z at point 0, respectively. The last assumption is required for the feedback simplification. To the best of our knowledge, this relationship between the type of the system and its allowable input, given in Theorem 4, is not mentioned in most of the control systems literature. To ascertain our finding, we consulted some control systems experts and they confirmed our results. Missing such corner cases is a common problem in paper-and-pencil based mathematical analysis and simulation and is one of the major causes for faulty system designs. The proof of Theorem 4 is based on various properties of limit of a complex function and complex arithmetic reasoning.

The formalization presented so far in this section consumed about 300 man-hours, which are mainly spent in the user guided verification due to the undecidable nature of the higher-order logic. Our proof script is available at [2]. The main benefit of this development, however, is that it greatly facilitates the formal reasoning about unity-feedback control system properties by reducing the human interaction in such proofs, as will be illustrated in the next section.

It is important to note that the universal quantification over the variables Y, Z, a and b in Theorem 4 allows us to use this result for reasoning about

steady-state error of any unity-feedback control system irrespective of its type, input and behavior. To the best of our knowledge, such a generic relationship for the steady-state error for unity-feedback systems has not been reported in the control systems literature.

Now, we outline the step-wise process for reasoning about the steady-state error of unity-feedback systems using Theorem 4. The first step is to use the formal definitions, given in Section 3, to develop a formal model of the given system using its structural description. Next, we verify the equivalence of this model and the expression `general_tf Y Z b`, by choosing appropriate assignments of the functions Y and Z and the variable b. Next, we express the theorem for the steady-state error of the given unity-feedback system: `steady_state_error (uf_error_model (<transfer function of the given system>) a) = <steady state error>`. Now, using Theorem 4 along with the fact that all of its assumptions hold for the given values of Y, Z, a and b, we can conclude the proof of steady-state error of the given unity-feedback system. In order to illustrate the effectiveness and practical utilization of Theorem 4 and the above mentioned process, we analyze a real-world control system in the next section.

5 Application: Pulse Width Modulation (PWM) push-pull dc-dc converter

The Pulse Width Modulation(PWM) push-pull dc-dc converters are widely used to step down dc voltages and thus have many applications in areas, like aerospace applications, where dc voltage is produced and consumed. The steady-state response of this electronic device is of utmost importance and thus has been extensively studied [14, 8]. A commonly used model [8] for steady-state error analysis of the PWM push-pull dc-dc converter is given in Fig. 6.

In this section, we share the experiences of a control system specialist in verifying the steady-state error relationship for the PWM push-pull dc-dc converter using our formalization. This person is a graduate student of Electrical Engineering and her research interests are in the area of mathematical analysis of control systems. The person had taken academic courses on discrete mathematics, programming languages and calculus but had no background about formal methods. We provided a two week HOL-Light extensive training to the person with major focus on formal reasoning about complex arithmetic and limits. During the course of the training as well as the case study, the person struggled in understanding the syntactical and type checking errors of HOL-Light and thus was significantly assisted in this regard.

The person initiated the exercise by developing the following higher-order-logic model for the block diagram, given in Fig. 6. Initially, she got confused in defining multiple feedback paths while using the function `pickoff` and mistakenly used the transfer function of `1/s` instead of 1 in the pre-fan-out block. She caught this mistake herself during the second step of our proposed approach, where the equivalence of the formal model is verified with the one obtained via the `general_tf` function and the correct definition is given below:

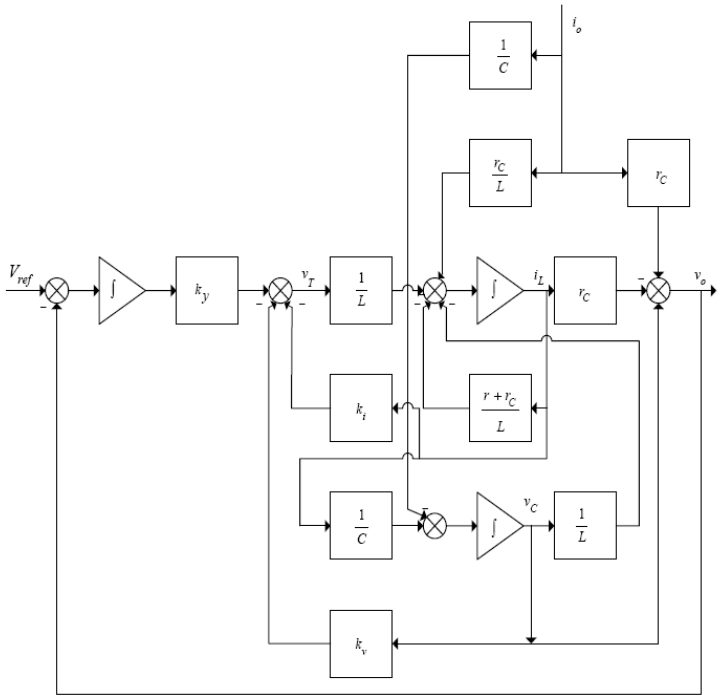

Fig. 6. Steady-State Error Model of PWM push-pull dc-dc converters

Definition 15: *PWM push pull dc-dc converter*

```
∀ L C r rc Ky Kv Ki s.
 dc_dc_converter L C r rc Ky Kv Ki s =
   series [ Cx(&1)/s ; Ky ; feedback ( series [Cx(&1)/L;
   feedback (Cx(&1)/s) (--sum_junction (pickoff (Cx(&1))
   [(r+rc)/L; series [Cx(&1)/s; Cx(&1)/C; Cx(&1)/L]]))])
   (--sum_junction (pickoff (Cx(&1))  [Ki;
   series [Kv; Cx(&1)/s; Cx(&1)/C]])); sum_junction [rc;
   series [Cx(&1)/s; Cx(&1)/C]]]
```

Where the `--` symbol represents the minus operator in HOL-Light. The variables L, C, r and rc donate inductor, capacitor, the equivalent resistance in series with an inductor and the equivalent resistance in series with a capacitor, respectively. While kv, ki and ky are the voltage, current and feedback loop gains, respectively. None of these variables can be zero for the correct operation of the dc-dc converter. A subgoal of the equivalence theorem is given below, where the structure of the PWN push pull dc-dc converter is simplified to obtain its corresponding characteristic equation given in [8]:

Theorem 5: *dc-dc converter Transfer function simplification*

```
⊢ ∀ L C s r rc Ky Kv Ki.
  ¬( C * L * r * rc * Ki * Kv * Ky = Cx (&0)) ∧ ¬(s = Cx (&0)) ∧
  (∀s. ¬(s pow 2 *C *L + s*C*(r + rc) +Cx(&1) = Cx(&0)))∧
  (∀s. norm(inv s * --((r+rc) * inv L + inv (s*C*L))) <&1) ∧
  (∀s. norm ((s * C) / ( s pow 2 * C * L + s * C * (r + rc) +
  Cx(&1)) * --(Ki + Kv * inv (s *C)))) < &1
  ⇒ dc_dc_converter L C r rc Ky K Ki =
      λs.(Ky * (s*C*rc +Cx(&1))) / (s pow 3 * C* L + s pow 2 *
        (r+rc+Ki) + s (Cx(&1)+Kv) )
```

Note that none of the physical values in the model can be zero and this is ensured by the first assumption. The next two assumptions are used to avoid singularities and the last two assumptions are required for solving the feedback paths. Our control engineer was not able to guess the right set of assumptions upfront and thus added the missing assumptions during the reasoning process based on the feedback she got from the generated subgoals. Thus, it was clearly observed in this exercise that interactive theorem provers do guide their users to find the right set of assumptions.

Since our given model is a Type 1 system, therefore its steady-state-error for the unit step input should be zero[14]. The result is verified as:

Theorem 6: *Steady-State Error for step input*

```
⊢ ∀ L C r rc Ky Kv Ki.
  ¬( C * L * r * rc * Ki * Kv * Ky = Cx (&0)) ∧
  ¬(Cx (&1) + Kv = --Ky) ∧
  (∀s. ¬(s = --((C * (r+rc+Ki + csqrt ((C * (r+rc+Ki)) pow 2 +
   --(Cx(&4)*C*L*(Cx(&1)+Kv)))) / (Cx(&2)*C*L))))) ∧
  (∀s. ¬(s = --((C * (r+rc+Ki + --csqrt ((C * (r+rc+Ki)) pow 2
   + --(Cx(&4) * C * L * (Cx(&1)+Kv)))) / (Cx(&2)*C*L))))) ∧
  (∀s. norm ((Ky * (s * C * rc + Cx (&1))) / (s pow 1 * (C * L *
   s pow 2 + s * C * (r+rc+Ki)+Cx(&1)+Kv))) < &1)
  ⇒ steady_state_error (uf_error_model general_tf
      (λs.Ky * (s*C*rc + Cx(&1))) (λs.C*L* s pow 2 +
        s*C*(r+rc+Ki) + Cx(&1) + Kv ) 1) 1) = Cx(&0)
```

The first assumption ensures that none of the component in the dc-dc converter has a zero value and the next three assumptions are uesd to avoid singularities. The last assumption is for the feedback simplification.

The verification of the above theorem involves the equivalence theorem, as described in the previous section, along with Theorem 4. Besides the above theorem, the control engineer also verified the relationship of the steady-state error for ramp input. The theorem is described below:

Theorem 7: *Steady-State Error for ramp input*

```
⊢ ∀ L C r rc Ky Kv Ki.
  ¬( C * L * r * rc * Ki * Kv * Ky = Cx (&0)) ∧
```

```
¬(Cx (&1) + Kv = --Ky) ∧
(∀s. ¬(s = --((C * (r+rc+Ki + csqrt ((C * (r+rc+Ki)) pow 2 +
 --(Cx(&4)*C*L* ( Cx(&1) + Kv)))) / (Cx(&2)*C*L))))) ∧
(∀s. ¬(s = --((C * (r+rc+Ki + --csqrt ((C*(r+rc+Ki)) pow 2
 + --(Cx(&4) * C * L * (Cx(&1)+Kv)))) / (Cx(&2)*C*L))))) ∧
(∀s. norm ((Ky * (s * C * rc + Cx (&1))) / (s pow 1 * (C * L *
 s pow 2 + s * C * (r+rc+Ki)+Cx(&1)+Kv))) < &1)
⇒ steady_state_error (uf_error_model general_tf
   (λs. Ky * (s*C*rc + Cx(&1))) (λs. C*L* s pow 2 +
    s*C*(r+rc+Ki) + Cx(&1) + Kv ) 1) 2) = (Cx(&1) + Kv) / Ky
```

The reasoning process was very similar to the one used for Theorem 6 since the same values for the functions Y and Z are used in these theorems. Further details about its verification can be found in our proof script[2].

The exercise of involving a control systems engineer for conducting these proofs was quite a learning experience for us as well. Some of the feedback that we got is shared here. The user faced many issues in interpreting the type checking and syntactical error messages generated by HOL-Light and this was the most frustrating issue for him. Thus, this is an area that can be improved. Moreover, the person was not too comfortable with the text-based interface of the theorem prover and suggested to bring in a more user-friendly graphical interface, specially for control system analysis. On the other hand, the user was quite amazed at the feedback she got from the theorem prover in understanding the behavior of the control system model and the requirement of an exhaustive set of assumptions to verify any theorem. She felt that this sort of rigorous analysis is a dire need in the case of safety-critical control system design. It was quite encouraging for us that the engineer was able to verify the goals with the basic understanding of limits and complex numbers in HOL-Light. This fact also demonstrates the effectiveness of our generic theorem.

6 Conclusions

This paper presents a formal framework to reason about the net transfer functions and steady-state errors of unity-feedback control systems within the sound core of a theorem prover HOL-Light. The main contribution of the paper is the formal verification of a generic theorem that facilitates in the formal reasoning about any kind of unity-feedback system. For illustration purposes, we presented a formal analysis of a PWM push pull dc-dc converter. Some of the interesting future directions of our work are to formally analyze the stability of control systems and establishing a link between the formalized Laplace transform theory[21] to be able to link time and Laplace domain models of a control system.

References

1. P. Clayton A. Cavalcanti and C. OHalloran. From Control Law Diagrams to Ada via Circus. *Formal Aspects of Computing*, 23(4):465–512, 2011.

2. M. Ahmad. Formal Verification of Steady State Errors in Unity-Feedback Control Systems. http://save.seecs.nust.edu.pk/students/ahmad/sseufcs.html, 2014.
3. R. Alur. Formal Verification of Hybrid Systems. In *Embedded Software*, pages 273–278, 2011.
4. R. Arthan, P. Caseley, C. O'Halloran, and A. Smith. ClawZ: Control Laws in Z. In *Formal Engineering Methods*, pages 169–176, 2000.
5. R.J. Boulton, H. Gottliebsen, R. Hardy, T. Kelsey, and U. Martin. Design Verification for Control Engineering. In *Integrated Formal Methods*, volume 2999 of *LNCS*, pages 21–35, 2004.
6. R.J. Boulton, R. Hardy, and U. Martin. A Hoare Logic for Single-Input Single-Output Continuous-Time Control Systems. In *Workshop on Hybrid Systems, Computation and Control*, volume 2623 of *LNCS*, pages 113–125, 2003.
7. J. S. Dong C. Chen and J. Sun. A Formal framework for Modeling and Validating Simulink diagrams. *Formal Aspects of Computing*, 21(5):451–483, 2009.
8. D. Czarkowski, L. R. Pujara, and M. K. Kazimierczuk. Robust Stability of State-Feedback Control of PWM DC-DC push-pull Converter. *IEEE Transaction on Industrial Electronics*, 42(1):108–111, 1995.
9. J. Harrison. A hol theory of Euclidean Space. In *Theorem Proving in Higher Order Logics*, volume 3603 of *LNCS*, pages 114–129, 2005.
10. J. Harrison. Formalizing Basic Complex Analysis. *Studies in Logic, Grammar and Rhetoric*, 10:151–165, 2007.
11. J. Harrison. The HOL Light Theory of Euclidean Space. *Journal of Automated Reasoning*, 50(2):173–190, 2013.
12. O. Hasan and M. Ahmad. Formal analysis of steady state errors in feedback control systems using hol-light. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1423–1426, 2013.
13. H. Herencia-Zapana, R. Jobredeaux, S. Owre, P. Garoche, E. Feron, G. Perez, and P. Ascariz. PVS Linear Algebra Libraries for Verification of Control Software Algorithms in C/ACSL. In *NASA Formal Methods*, volume 7226 of *LNCS*, pages 147–161. Springer, 2012.
14. Y. Hote. A New Approach to Time Domain Analysis of Perturbed PWM push-pull DC-DC Converter. *Journal of Control Theory and Applications*, 10(4):465–469, 2012.
15. M. D. Lutovac and D. V. Toic. Symbolic Analysis and Design of Control Systems using Mathematica. *International Journal of Control*, 79(11):1368–1381, 2006.
16. B. Mahony. The DOVE approach to the Design of Complex Dynamic Processes. In *Workshop on Formalising Continuous Mathematics*, pages 167–187. NASA conference publication, 2002.
17. MathWorks Simulink. www.mathworks.com/products/simulink, 2012.
18. N.S. Nise. *Control System Engineering*. Wiley and Sons, 2003.
19. K. Ogata. *Modern Control Engineering*. Prentice-Hall, 1997.
20. L. Pike. Pervasive Formal Verification in Control System. In *Formal Methods in Computer-Aided Design*. Panel Discussion, 2011.
21. S. H. Taqdees and O. Hasan. Formalization of Laplace Transform Using the Multivariable Calculus Theory of HOL-Light. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *LNCS*, pages 744–758. Springer, 2013.
22. A. Tiwari and G. Khanna. Series of Abstractions for Hybrid Automata. In *Workshop on Hybrid Systems: Computation and Control*, volume 2289 of *LNCS*, pages 465–478, 2002.