5-2014

Developing a Tool for Generation of Operational Amplifier Models from Datasheet Parameters

Benjamin Riley Sissons
University of Arkansas, Fayetteville

Follow this and additional works at: http://scholarworks.uark.edu/eleguht

Part of the VLSI and Circuits, Embedded and Hardware Systems Commons

Recommended Citation
http://scholarworks.uark.edu/eleguht/27

This Thesis is brought to you for free and open access by the Electrical Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Electrical Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.
This thesis is approved.

Thesis Advisor:

Dr. H. Alan Mantooth

Thesis Committee:
DEVELOPING A TOOL FOR GENERATION OF OPERATIONAL AMPLIFIER MODELS FROM DATASHEET PARAMETERS
DEVELOPING A TOOL FOR GENERATION OF OPERATIONAL AMPLIFIER MODELS FROM DATASHEET PARAMETERS

An Undergraduate Honors College Thesis in the

Department of Electrical Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

By

Benjamin Sissons

April 2014
ABSTRACT

Due to their flexibility and usefulness operational amplifiers are a very common circuit component that has been in use for over fifty years. Because of their widespread use in circuits there has always been a need for simulation models of the op amp. The purpose of this thesis is to demonstrate the design a tool for use with the modeling software ModLyng [1] that takes several parameters from the datasheet of an op amp and generates a model that can be exported to several popular hardware description languages including Verilog-A and Verilog-AMS. The tool was designed as a plugin for ModLyng written in the programming language python. Several models were generated from the op amp datasheets and simulated. The results of these simulations were then compared to the datasheet results.
ACKNOWLEDGEMENTS

I would first like to thank Dr. Alan Mantooth for giving me the opportunity to work at the mixed-signal CAD department where it has truly been an honor to work with some of the best and brightest at the University of Arkansas. It has been a wonderful learning experience and I am truly thankful for his guidance and support along the way.

I would also like to thank Dr. Matt Francis for helping guide me with my work in ModLyng. He always made sure to offer his valuable time to help solve any problem that arose and I am very appreciative of his help.

Finally, I would like to thank my family for all of their help getting to where I am now. My parents have always helped with any problem large or small and given me encouragement when I needed it. Thank you for all of your support.
# TABLE OF CONTENTS

I. Introduction .......................................................................................................................... 1

II. Background ......................................................................................................................... 3

   A. Background of the Operational Amplifier ........................................................................... 3
   B. The Need for Models .......................................................................................................... 3
   C. ModLyng Background ....................................................................................................... 4

III. Design and Implementation ................................................................................................. 7

   A. Researching the Operational Amplifier ............................................................................. 7
   B. Designing the Op Amp Model ............................................................................................ 8
   C. Researching the ModLyng Plugin Tool ............................................................................. 17
   D. Designing the Op Amp Plugin .......................................................................................... 18

IV. Simulation Results .............................................................................................................. 24

   A. UA741 Operational Amplifier .......................................................................................... 25
   B. 4.2 LMC660 CMOS Rail-to-Rail Operational Amplifier ............................................... 29
   C. 4.3 LM6720 Video Operational Amplifier ......................................................................... 30

V. Conclusions and Future Research ....................................................................................... 33

VI. References .......................................................................................................................... 34

VII. Appendix A ......................................................................................................................... 36

VIII. Appendix B ......................................................................................................................... 38
LIST OF FIGURES

Fig. 1. Operational amplifier symbol [7]........................................................................................................1
Fig. 2. Screenshot of ModLyng. .........................................................................................................................5
Fig. 3. Input stage of generic op amp model.......................................................................................................9
Fig. 4. Second stage of the op amp model: high gain, low pass filter. ............................................................10
Fig. 5. Output of y=tanh(x) function................................................................................................................12
Fig. 6. Output voltage cut-off effect. ................................................................................................................13
Fig. 7. Third stage of the op amp model: the output buffer. ............................................................................14
Fig. 8. Complete three stage op amp model.....................................................................................................15
Fig. 9. Two additional pole stages. ................................................................................................................16
Fig. 10. ModLyng dialog box log window. .........................................................................................................19
Fig. 11. The general design flow of generating a model. ..................................................................................21
Fig. 12. Creating new op amp model from blueprint model...........................................................................22
Fig. 13. Open-loop op amp testbench for frequency characteristics..............................................................24
Fig. 14. Testbench circuit for overshoot and slew rate ...................................................................................25
Fig. 15. Open-loop gain characteristic of UA741 op amp. ...........................................................................26
Fig. 16. Non-inverting amplifier step response of UA741 op amp. .................................................................27
Fig. 17. Closed-loop slew rate test UA741 op amp. .........................................................................................28
Fig. 18. Open-loop frequency response for LMC660 op amp. .....................................................................30
Fig. 19. Closed-loop frequency response of LMH6720. .................................................................................32
I. INTRODUCTION

The Operational Amplifier (Op Amp) is a very important analog circuit. Since its inception in the 1930s the op amp has left its mark as one of the greatest inventions of the 20th century. The uses for op amps are as diverse as performing mathematical functions (such as multiplication or integration [2]), signal filters for wireless communication [3], electrocardiogram (EKG) amplifiers [4], control loop feedback controllers [5], and countless other applications.

The flexibility of the op amp lies in its extremely high open loop gain. This extremely high gain allows the op amp (when used in a negative feedback configuration) to produce a very precise gain characteristic that only depends on the type of feedback network used [2].

Another reason the op amp has achieved widespread use is the fact that the actual model is typically very close to the ideal version of the op amp, i.e. infinite input impedance, infinite gain, no output impedance, etc. [6]. The ideal nature of the modern op amp allows circuit designers to know exactly what to expect from the op amps they use in their circuits. As shown in Fig. 1, there should ideally be very high input impedance, very high gain, and very low output resistance.

![Operational Amplifier Symbol](image)

Fig. 1. Operational amplifier symbol [7].
Because of the usefulness of the op amp, there will always be a need for models for design and planning purposes. The work in this thesis describes the method of analyzing an op amp model, finding the parameters that are important, modeling a generic op amp model, and implementing a method to allow users to generate a model quickly from a datasheet.

This thesis explores the process of designing and building the op amp generation tool from start to finish. This paper is broken up into five major sections. First is the introduction, followed by the background chapter, which gives information on the history of the op amp as well as the integrated modeling environment (IME) used as the platform for this op amp plugin. The third section describes the method employed to research the models for the op amp and how it was designed, as well as the process of programming the plugin tool. The fourth section provides a selection of op amps that were modeled and simulated using the op amp model generator described in this paper and the simulations were compared to the actual datasheet information. The final section is a conclusion, as well as a description of future work that could be done in this area. Additionally, Appendix A contains the datasheet graphs that were used to compare the simulated results and Appendix B contains a generated Op Amp Model in Verilog-A.
II. BACKGROUND

A. BACKGROUND OF THE OPERATIONAL AMPLIFIER

The op amp was first invented in the 1930s and utilized vacuum tubes for operation [8]. However, these types of circuits were very large and expensive. With the invention of the transistor in the 1950s, the invention of the monolithic operational amplifier was made possible. In 1963 Bob Widlar invented the first monolithic op amp, the μa702 operational amplifier [8].

The term operational amplifier was first coined in 1947 by Columbia University professor John Ragazzini because the circuit could perform mathematical operations [9]. In fact, the original purpose of the operational amplifier was to perform the mathematical operations in some of the first analog computers [3], including the one at the University of Arkansas circa 1959 [10]. However, that is only the tip of the iceberg when it comes to the usefulness of the op amp. Some other applications of the op amp include audio amplifiers, oscillators, comparators, filters, and many others [2]. Because of the inherent usefulness of the op amp, many different types of op amp topologies have been created to use in a variety of applications.

B. THE NEED FOR MODELS

An op amp modeling tool becomes extremely useful to analog circuit designers because it enables them to test a circuit’s capabilities at an early stage. For this reason manufacturers often include macromodels that describe the behavior of the op amps. However, the accuracy these models bring to the circuit designer comes at a cost. In general, the larger and more accurate the macromodel, the longer the simulations take.

This is where behavioral models come in. According to Dr. Ian Getreu, a macromodel is a model represented by an equivalent circuit using the components available to the simulator,
whereas a behavioral model is a model represented by nonlinear algebraic-differential equations in a hardware description language (HDL) [11]. Each modeling approach brings tradeoffs to the table in terms of speed of simulation and accuracy. The advantage of ModLyn is that it provides a graphical interface that allows the user to see exactly what it is that they are modeling. The same HDL output is generated as with a purely hand-written behavioral model, but ModLyn provides a clean, intuitive way of designing and explaining a complicated system to other engineers, or even non-engineers. This makes it an excellent tool for engineers to use to show other people their work.

C. MODLYNG BACKGROUND

ModLyn was a program originally conceived back in 2001 at the University of Arkansas under the guidance of Dr. Alan Mantooth [12]. At that time it was known as Paragon and developed exclusively by the University of Arkansas. The purpose of Paragon was to create an efficient development process for analog and mixed-signal modeling by utilizing an equation editor as well as a graphical interface. By meshing seamlessly with commercial HDL simulators, it was an excellent way to quickly test and verify a design before moving on to the next phase in the design process.

Beginning in 2004, Paragon was commercialized by Lynguent into the product now known as ModLyn.
Instead of simply modeling a system in a hardware description language (HDL) such as Verilog-A or VHDL, ModLyng allows the modeler to design a graphical model that can be exported to several HDLs including VHDL-AMS, Verilog, Verilog-A, and MAST. This is a very powerful method of modeling because it allows the designer to model on top of the languages [13] instead of around them. Although ModLyng is capable of implementing system level modeling, event based modeling, and electrical flow modeling, for the purposes of this model and paper the discussion will be limited to electrical models.

While keeping the same basic design flow as Paragon, ModLyng also enabled the program to achieve many of the goals that were previously unattainable due to not having a commercial budget or a full team of developers. Despite the benefits of the program, due to unforeseen circumstances development was stopped on ModLyng after several years.

The license for ModLyng came back to the University of Arkansas Mixed-Signal Computer Aided Design (MSCAD) department in 2013. Unfortunately, because of the interval of
time where the program was not developed, many of the tools that were used in the framework of
the program had compatibility issues due to newer versions of the tools. Some of these issues
could be resolved by simply updating the tools used. However, problems such as the outdated
licensing tool were much more complicated to solve.

In the summer of 2013, two students, including the author of this paper, worked to
resurrect ModLyng and restore it to its development potential. Once this task was accomplished,
the focus of the MSCAD department turned to new development efforts.

After spending the summer of 2013 restoring ModLyng to working development status,
the author decided to continue development of the program. This would serve two purposes:

- To keep ModLyng in active development in order to better service the program.
- To develop a plugin that would be useful for end users.

With these goals in mind it was decided to design and develop a plugin to generate
operational amplifier models.

ModLyng makes an excellent choice as the platform for the op amp generation tool
because of its flexibility i.e. its ability to export models into several different hardware
description languages including Verilog-A, Verilog-A/MS, and VHDL-AMS. ModLyng also
has a very easy plugin development flow that enables developers to add plugins written in either
Python or C++ to add to the usefulness of ModLyng. This simple but powerful plugin
development method ensures that useful scripts and plugins can be added to ModLyng by anyone
with a basic background in programming.
III. DESIGN AND IMPLEMENTATION

A. RESEARCHING THE OPERATIONAL AMPLIFIER

In order to begin work on the operational amplifier generation plugin, two major topics needed to be researched: how to model op amps and how the ModLyng plugin class interfaces with ModLyng.

The starting point for research into the op amp was reading through Ian Getreu and Ken Ruan’s discussion of the op3 generic behavioral model where a method was discussed to model op amps behaviorally. The main topic that was discussed for the Op3 model was how to have a model that is flexible with a large amount of parameters. In fact, the op3 model described by Dr. Getreu has over 40 possible input arguments [11]. Despite the large number of parameters described in that method, it was designed to be as lightweight as possible which was possible due to the model being written in the MAST hardware description language.

The model in this thesis was also designed to be lightweight enough that it would not add too much to the simulation time, but would still give accurate results. As per the method described in [14], a three stage design was implemented with the option for more stages based upon the number of additional poles required by the user. The three main stages of the op amp model are a differential input stage, low pass filter stage, and an output buffer stage. To correspond with the three stages of the model, Table 1 shows the list of functional parameters that will be accepted by this op amp model. These parameters are based off of the list of functional specifications described by Ian Getreu in [11]. However, in that method the parameters were separated based on which stage of the op amp they affected, while in the model described in this paper they are sorted based on the general type, i.e., technology, input, output. This allowed for easy sorting when the model was implemented in ModLyng.
Table 1. Op amp model parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Name in Model</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Loop Voltage Gain</td>
<td>avol</td>
<td>Technology</td>
</tr>
<tr>
<td>Unity Gain Bandwidth</td>
<td>fl</td>
<td>Technology</td>
</tr>
<tr>
<td>Slew Rate</td>
<td>sr</td>
<td>Technology</td>
</tr>
<tr>
<td>Temperature</td>
<td>temp</td>
<td>Technology</td>
</tr>
<tr>
<td>Nominal Power Supply</td>
<td>vnom</td>
<td>Technology</td>
</tr>
<tr>
<td>Input Resistance</td>
<td>rin</td>
<td>Input</td>
</tr>
<tr>
<td>Input Bias Current/Temperature Variation</td>
<td>ib/ibtc</td>
<td>Input</td>
</tr>
<tr>
<td>Input Bias Offset/Temperature Variation</td>
<td>ibos/ibostc</td>
<td>Input</td>
</tr>
<tr>
<td>Input Voltage Offset/Temperature Variation</td>
<td>vos/vostc</td>
<td>Input</td>
</tr>
<tr>
<td>Maximum Voltage Swing (how close the output can swing to rails)</td>
<td>vswing</td>
<td>Output</td>
</tr>
<tr>
<td>Output Resistance</td>
<td>rout</td>
<td>Output</td>
</tr>
<tr>
<td>Maximum Short Circuit Output Current</td>
<td>iout</td>
<td>Output</td>
</tr>
<tr>
<td>Additional Poles</td>
<td>N/A</td>
<td>Poles</td>
</tr>
</tbody>
</table>

Once the parameters were chosen for the model, it was time to investigate each parameter and how it affected the model.

B. DESIGNING THE OP AMP MODEL

The two main components of a model in ModLyng are the branches and ports. Branches model two basic electrical concepts, the through and across variables i.e. current and voltage. The through and across variables are related by a branch equation that ensures conservation of energy in the model [13]. However, the power of using an HDL over simply putting together a circuit in SPICE is that it enables the designer to quickly check their design rather than waiting
much longer for a circuit to simulate. The ports are simply the inputs and outputs of the model that connect it to the outside world.

The model for the input stage of the op amp is shown in Fig. 3 below.

![Fig. 3. Input stage of generic op amp model.](image)

The first stage of the model is based off of the method found in [14]. This stage takes the differential input voltage across the input resistor while adding in the offset voltage as an ideal voltage source in series with the positive input node. The bias currents are injected into the input branch with ideal current sources. The offset bias current is added in between the two to control the offset between biases.

The input resistance $r_{in}$ is the resistance across the input of the op amp. It is modeled as a resistor effect across the input terminals.
Next on the parameter list is the input offset characteristics, the input voltage offset $v_{os}$ and the input bias current offset $i_{bos}$. The input voltage offset is the amount that the input varies when the input differential voltage is otherwise zero. The input bias offset is the amount that the input bias currents vary through the positive and negative input terminals as shown in Eq. (3.1).

$$i_{bos} = i_{b}^+ - i_{b}^-$$ (3.1)

The second stage of the op amp is the low pass filter with a voltage controlled current source as the input. This is shown in Fig. 4 below.

---

**Fig. 4.** Second stage of the op amp model: high gain, low pass filter.
This is the stage that gives the model its high differential gain as well as modeling the first pole and the slew rate. The input branch is a voltage controlled current source. The current flowing through this branch is related to the voltage across the input resistance by Eq. (3.2).

\[ I_{tpf} = I_{out} \cdot \tanh(K \cdot v_{id}) \]  

(3.2)

The slew rate, \( sr \), is the limit of how fast the output of the circuit can change, or the maximum change in voltage with respect to the change in time. It is related to the maximum output current, \( i_{out} \), by the capacitance of the low pass filter stage as shown in Eqs. (3.3) – (3.5).

\[ sr = \frac{dv}{dt} \]  

(3.3)

\[ I_{out} = \frac{dv_c}{dt} \cdot C_{tpf} \]  

(3.4)

\[ \Rightarrow I_{out} = (sr)C_{tpf} \]  

(3.5)

In this model, the slew rate is modeled by the hyperbolic tangent function as described in [14].

\[ I_{tpf} = I_{out} \cdot \tanh(K \cdot v_{id}) \]  

(3.6)

The reason the hyperbolic tangent function can be used to model the slew rate is because it naturally models the output current saturation of a differential pair of transistors [15]. The output of the hyperbolic tangent, or tanh, function is shown in Fig. 5 below.
The way the output current is limited by the tanh function can be explained by looking at the full amplification expression of the second stage of the op amp expression when $v_{id}$ is sufficiently small as shown in Eq. (3.8).

$$I_{tpf} = I_{out} \cdot \tanh(K \cdot v_{id})$$  \hspace{1cm} (3.7)

(when $v_{id}$ is small) $\Rightarrow$ $I_{tpf} = I_{out} \cdot K \cdot v_{id}$  \hspace{1cm} (3.8)

Once $v_{id}$ becomes too large, the tanh function saturates to the $I_{out}$ value.

$$\Rightarrow I_{tpf} = I_{out}$$  \hspace{1cm} (3.9)

The gain of the second stage’s voltage controlled current source, $K$, comes from the concept of the Gain-Bandwidth-Product (GBP) of an op amp. The GBP is the product of the open loop gain and the frequency of that particular gain [6]. Using the principle that the GBP of
an op amp is a constant, the first pole frequency (the 3dB frequency) can then be calculated using values for the open-loop gain, $avol$, and the unity gain frequency, $fu$.

$$GBP = gain \times bandwidth$$  \hspace{1cm} (3.10)

$$\Rightarrow fu \times 1 = fp1 \times avol$$  \hspace{1cm} (3.11)

$$\Rightarrow fp1 = fu/avol$$  \hspace{1cm} (3.12)

Then, the resistive value for the low pass filter can be calculated based upon the results of Eqs. (3.5) and (3.12).

$$R_{lpf} = 1/(2 \times \pi \times fp1 \times C_{lpf})$$  \hspace{1cm} (3.13)

$$K = \frac{avol}{V_{out}+R}$$  \hspace{1cm} (3.14)

The parameter $vswing$ is how close the output of the op amp can swing to the rail voltages. This is important because it influences the voltage cut-off stage. This cut-off stage is implemented by a diode and ideal voltage source as shown in Fig. 6 below.

![Fig. 6. Output voltage cut-off effect.](image-url)
The way the voltage cut-off effect works is when the output voltage of the second stage is greater than \( Vcc - vswing \), the output simply becomes \( Vcc - vswing \). When the output voltage is less than \( Vee + vswing \), the output becomes \( Vee - vswing \).

The third and final stage of the op amp model is shown in Fig. 7 below.

![Diagram of the output buffer stage of the op amp model]

**Fig. 7. Third stage of the op amp model: the output buffer.**

The last two parameters \( iout \) and \( rout \) are used in this stage. The output resistance is modeled as a simple resistor in series with the output as shown in Fig. 7 above. The output current is a very important parameter as it influences both the output characteristic as well as the low pass filter stage as explained in Eq. (3.5) above.

The three main stages of the model that is generated by the plugin is shown in Fig. 8 below.
In order to increase the fidelity of the model, an additional specification was added to allow the user to manually add extra poles to the model. If the option for additional poles was selected, then after the regular three stage model is generated the additional pole stages are inserted between the initial low pass filter stage and the output buffer stage. These additional poles are modeled as RC filters with unity dc gains [14]. Two additional pole stages are shown in Fig. 9 below.
The unity gain, low pass filters are implemented in the model by choice of the resistor value and gain of the voltage controlled current source such that they cancel each other out as shown in Eq. (3.16) [14].

\[ I_{pole} = G_{pole} \cdot V_{in} \]  \hspace{1cm} (3.15)

\[ G_{pole} = \frac{1}{R_{pole}} \]  \hspace{1cm} (3.16)

Since the output voltage is simply the current times the resistor of the low pass filter, the output voltage is equal to the input voltage as shown in Eqs. (3.17) – (3.19).

\[ V_{out} = I_{pole} \cdot R_{pole} \]  \hspace{1cm} (3.17)

\[ \Rightarrow V_{out} = G_{pole} \cdot V_{in} \cdot R_{pole} \]  \hspace{1cm} (3.18)

\[ \Rightarrow V_{out} = V_{in} \]  \hspace{1cm} (3.19)

Thus, the extra pole stages will affect the frequency characteristics of the op amp model without affecting the gain.
C. RESEARCHING THE MODLYNG PLUGIN TOOL

According to the ModLyng documentation, “a ModLyng Plugin is a Python script or a C++ program that interacts with the ModLyng IME to provide certain functionality on demand.” It is a very powerful way to store parameter, create objects, and otherwise widely extend the usefulness of the program.

The plugin development flow for ModLyng begins with a dedicated plugin class that is included with ModLyng. This class includes a “ready to go” dialog box implemented with an application framework called QT for the graphical user interface (GUI) [16]. The dialog has several types of parameters, error message boxes, error handling, and most importantly a full API (application programming interface) that can access the inner most workings of the ModLyng IME.

This simple development flow ensures that anyone can be up and running very soon. The only required function that must be included in all plugins is an event handler function that is called by the ModLyng plugin API when an event is generated by ModLyng. These events are generated anytime an action occurs in ModLyng that would affect the plugin [17]. Some of these events include anytime a dialog parameter changes, a menu appears, or a button is pressed. The event handler function exists simply to check the reason for the event and call the appropriate functions. To summarize, the way that ModLyng handles a plugin is:

1) An event is generated by ModLyng.

2) The event is passed through to the plugin's event handler method.

3) The event handler executes any functions associated with the event's reason.

4) Any errors that occurred are reported back to ModLyng through the initial event that was called.
D. DESIGNING THE OP AMP PLUGIN

The plugin that is the topic of this paper was architected in a way to allow ease of access to the parameters to increase input flexibility. The overall architecture of the op amp generation plugin was based off of some of the current ModLyng plugins written by Dr. Matt Francis. The general flow of the plugin will be described in this section.

The first thing added to the plugin was a Global class that stores all of the variable names. Having this global class means that adding new parameters to the model is as simple as updating a list at the beginning of the code. Throughout the rest of the plugin the parameters are simply referenced from this class and are instantiated and updated using for-loops. For example, to load the second page of the dialog box the following code is executed.

```python
#Set the old page's inputs to be invisible
try:
    for name in oldNames:
        oldPar = plAct.par(name[0]).autoCast()
        oldPar.setVisible(False)
except:
    event.setErrorMessage("Couldn't change windows!")
    return

#Names for the new page
newNames = _listOfInputs.getFrameNames(newWin)
#go through the names in the dialog and make them visible
for name in newNames:
    newPar = plAct.par(name[0]).autoCast()
    newPar.setVisible(True)
```

In addition to the parameter handling, probably the most important function of the global class is its error checking capability. The way it works is when the plugin first appears, all of the mandatory plugin parameter names are added to a list (optional parameters may be left blank). Any time a dialog box is edited, the “check validity” event is handled by the plugin. After passing through an error checking function, if the value added to the dialog box is valid then its
parameter name is removed from the error list. If the value is not valid however, the name is kept on the list but with a “bad value” flag. Any empty or bad values are added to the message log at the bottom of the dialog box as shown in Fig. 10 below.

![Fig. 10. ModLyng dialog box log window.](image)

The next function created was a function that handles the registration of the plugin class within the ModLyng IME. This is also where all of the dialog parameters are instantiated in the dialog box. One shortcoming that the ModLyng plugin class had was the lack of tooltips (boxes of hints and other information that pop up when your mouse hovers over them), despite the fact that QT (the application framework for the GUI) allows this. In order to make the dialog
parameters a little easier to understand, a function was added to the ModLyng Plugin class to allow the instantiation of tooltips inside any plugin.

Once the plugin is registered within ModLyng, the user is free to open in from the menu and add parameter values. Once all of the required parameter values have been added to the dialog and the user press “OK”, the next step in the plugin is to generate the model.

In the process of designing the function to generate the model, there were two schools of thought. The first idea was to generate the model completely from scratch, adding each branch, port, and code fragment inside the python plugin through the ModLyng API. The second idea was to take an ideal model file and merge it into the current topology by using the python plugin to copy the entity and add the new parameters.

The first method was explored first. A plugin creation flow based upon the existing testbench creator plugin was designed to generate a model. By using the process described in Fig. 11 below, models were successfully generated completely from scratch.
Fig. 11. The general design flow of generating a model.
Despite the success of this method in creating a model, it did not seem to be as useful for the purpose of this plugin since the presentation of the generated model was complicated and non-intuitive for someone who might need to edit the model after generating it. Therefore, the second option was explored as well.

The second method required having one extra file to base the generated model on. However, this is not a problem and is an acceptable amount of overhead for the plugin. The design flow for this version of generating an op amp model is shown in Fig. 12 below.

**Fig. 12. Creating new op amp model from blueprint model.**
The last step of the plugin was to transfer all of the dialog parameters into the model. This was problematic at first because there was no easy way to inject a string into an existing model using the plugin API. Examples existed that showed how to copy existing ModLyng parameter values, but those would not work for this plugin. The method that was found to accomplish the task of adding the parameters to a ModLyng entity was to take the function that was used in the main ModLyng add parameter function and to add it into the python API for ModLyng. This function takes four inputs:

1) A string that contains the value to be converted to a parameter.
2) The entity that the parameter is going to be placed in.
3) The newly created parameter.
4) The type definition of the parameter.

Then, the function parses the string and returns a ModLyng compatible parameter. Adding this function to the plugin API not only serves its purpose for this plugin, but will enable future plugin developers to do more with their plugins in terms of adding new parameters.

There were several special parameters that called for extra handling while the parameters are being added. These include the low pass filter parameters, the temperature dependent parameters, and the extra poles. These special cases required extra calculations before they can be added to the model. Fortunately, with the power of Python it was simple to import an extra library that could handle performing calculations with strings.
IV. SIMULATION RESULTS

In order to test the accuracy of the generic op amp model generation tool three different op amps were chosen, each with very different characteristics. The three op amps chosen to test this plugin are the UA741, the LMC660 CMOS rail-to-rail op amp, and the LM6720 wideband video op amp. Each of these three op amps offer a very different type of performance and so provide a good way to test the accuracy and find any shortcomings of the model.

In order to accurately compare the models to the datasheet results, a digitizer was used. The digitizer converts the datasheet waveforms into a file of comma separated values. This allowed for an easy comparison to the modeled values. Appendix A has the exact datasheet waveforms from the datasheets.

For the testing of each op amp model, two testbenches were used. The first is a simple open loop configuration with an AC source to test the frequency characteristics of the op amp model. This testbench is shown in Fig. 13 below.

![Fig. 13. Open-loop op amp testbench for frequency characteristics](image)
The second testbench that was used was a non-inverting unity gain amplifier. This testbench was used to model the transient step response of the op amp model and is shown in Fig. 14 below.

![Testbench circuit for overshoot and slew rate](image)

**Fig. 14. Testbench circuit for overshoot and slew rate**

A. **UA741 OPERATIONAL AMPLIFIER**

The first op amp that was simulated was the ua741 op amp. Table 2 shows the datasheet parameters for the ua741 that were modeled here [18].
Table 2. UA741 datasheet parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Loop Voltage Gain</td>
<td>200k V/V</td>
</tr>
<tr>
<td>Unity Gain Bandwidth</td>
<td>1 MHz</td>
</tr>
<tr>
<td>Slew Rate</td>
<td>0.5 V/µs</td>
</tr>
<tr>
<td>Temperature</td>
<td>25°C</td>
</tr>
<tr>
<td>Input Resistance</td>
<td>2 MΩ</td>
</tr>
<tr>
<td>Input Bias Current</td>
<td>80 nA</td>
</tr>
<tr>
<td>Input Bias Offset</td>
<td>20 nA</td>
</tr>
<tr>
<td>Input Voltage Offset</td>
<td>1mV</td>
</tr>
<tr>
<td>Maximum Voltage Swing (how close the output can swing to rails)</td>
<td>±12 V</td>
</tr>
<tr>
<td>$V_{\text{supply}}$ Nominal</td>
<td>±15 V</td>
</tr>
<tr>
<td>Output Resistance</td>
<td>75 Ω</td>
</tr>
<tr>
<td>Maximum Output Current</td>
<td>40 mA</td>
</tr>
<tr>
<td>Extra Pole</td>
<td>4.6MHz</td>
</tr>
</tbody>
</table>

The comparison of the simulation and datasheet open loop frequency response is shown in Fig. 15 below.

![Open-loop gain characteristic of UA741 op amp.](image)

Fig. 15. Open-loop gain characteristic of UA741 op amp.
The frequency characteristic of the open loop gain of the model matched very closely with the datasheet result.

The next testbench that was used to test the ua741 was the non-inverting unity gain amplifier as shown in above Fig. 14 above. The stimulus used at the input was a 20 mV step input.

![Graph showing non-inverting amplifier step response of UA741 op amp.]

**Fig. 16. Non-inverting amplifier step response of UA741 op amp.**

The simulation gave a similar transient response compared to the datasheet waveform. The rise time to 90% of the final value was 0.3us from the datasheet and was 0.289us from the simulated model. The overshoot factor from the datasheet was 5% and the overshoot factor from the simulated model was 3.25%. The main way to tweak the transient response to give more similar results to the datasheet is to adjust the additional poles. The result shown in Fig. 16 above was established by quickly editing the extra pole until an appropriate value of 4.6MHz was found.
To demonstrate that the slew rate was modeled correctly, the closed loop non-inverting testbench was also run with a large signal step input as the stimulus. This result is shown in Fig. 17 below.

![Fig. 17. Closed-loop slew rate test UA741 op amp.](image)

The result for the closed-loop slew rate is has a slight offset from the output seen in the datasheet waveform. The cause of this could possibly be related to the testbench that was used when the ua741 was tested for the datasheet. Perhaps there was an initial offset voltage that caused the output to have a slightly higher starting voltage.
B. 4.2 LMC660 CMOS RAIL-TO-RAIL OPERATIONAL AMPLIFIER

The second op amp that was tested was the LMC660 CMOS rail-to-rail op amp. The parameters that were used for the model are shown in Table 3 below [19].

Table 3. LMC660 datasheet parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Loop Voltage Gain</td>
<td>440k V/V</td>
</tr>
<tr>
<td>Unity Gain Bandwidth</td>
<td>1.4 MHz</td>
</tr>
<tr>
<td>Slew Rate</td>
<td>1.1 V/μs</td>
</tr>
<tr>
<td>Temperature</td>
<td>25°C</td>
</tr>
<tr>
<td>Input Resistance</td>
<td>1 TΩ</td>
</tr>
<tr>
<td>Input Bias Current</td>
<td>0.0</td>
</tr>
<tr>
<td>Input Bias Offset</td>
<td>0.0</td>
</tr>
<tr>
<td>Input Voltage Offset</td>
<td>1mV</td>
</tr>
<tr>
<td>Maximum Voltage Swing (how close the output can swing to rails)</td>
<td>±14.63 V</td>
</tr>
<tr>
<td>$V_{\text{supply}}$ Nominal</td>
<td>±15 V</td>
</tr>
<tr>
<td>Output Resistance</td>
<td>10.0 Ω</td>
</tr>
<tr>
<td>Maximum Output Current</td>
<td>39 mA</td>
</tr>
</tbody>
</table>

The open loop response is shown in Fig. 18 below.
Unfortunately, the LMC660 datasheet did not include any form of transient analysis in the datasheet so no comparison is made in that regard.

C. 4.3 LM6720 VIDEO OPERATIONAL AMPLIFIER

The third op amp that was tested was the LMH6720 video op amp. This is a high speed, low gain, large bandwidth op amp. The parameters that were used for the model are shown in Table 4 below [20].
Table 4. LMH6720 datasheet parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-Loop Voltage Gain</td>
<td>2 V/V</td>
</tr>
<tr>
<td>Unity Gain Bandwidth</td>
<td>250 MHz</td>
</tr>
<tr>
<td>Slew Rate</td>
<td>1800 V/µs</td>
</tr>
<tr>
<td>Temperature</td>
<td>25 °C</td>
</tr>
<tr>
<td>Input Resistance</td>
<td>2 MΩ</td>
</tr>
<tr>
<td>Input Bias Current</td>
<td>1.0 uA</td>
</tr>
<tr>
<td>Input Bias Offset</td>
<td>0.0</td>
</tr>
<tr>
<td>Input Voltage Offset</td>
<td>0.2 mV</td>
</tr>
<tr>
<td>Maximum Voltage Swing</td>
<td>±3.9 V</td>
</tr>
<tr>
<td>Maximum Voltage Swing (how close the output can swing to rails)</td>
<td>±3.9 V</td>
</tr>
<tr>
<td>$V_{\text{supply Nominal}}$</td>
<td>±5 V</td>
</tr>
<tr>
<td>Output Resistance</td>
<td>0.06 Ω</td>
</tr>
<tr>
<td>Maximum Output Current</td>
<td>70 mA</td>
</tr>
<tr>
<td>Extra Pole</td>
<td>150 MHz</td>
</tr>
</tbody>
</table>

The datasheet for the LMH6720 op amp did not have an open loop frequency response graph, so a comparison was made for the closed loop non-inverting frequency response instead. This is shown in Fig. 19 below.
Fig. 19. Closed-loop frequency response of LMH6720.
V. CONCLUSIONS AND FUTURE RESEARCH

The objective of this thesis was to develop a method of automatically generating op amp models via a python based plugin. The overall method of generating models in ModLyng was successful. The generic model that was created is fairly accurate for several common transient and frequency characteristics of interest for circuit designers. However, there is still more work to be done. Future work for this plugin would be adding the ability to import a list of parameters via a comma separated value file. This would increase the reusability of each model and allow the user to quickly tweak different parameters. Another area that could be explored at a later time would be expanding the tool to model more op amp characteristics, thus increasing the usefulness for circuit designers.
VI. REFERENCES


VII. APPENDIX A

Data graphs that were digitized.

Figure A.1. UA741 Open Loop Frequency Response

Figure A.2. UA741 Closed Loop Step Response
Figure A.3. LMC660 Open Loop Frequency Response

Figure A.4. LMH6720 Closed Loop Frequency Response
VIII. APPENDIX B

Verilog-A model for op amp model based off of the ua741 datasheet parameters.

* pragma lynguent created by ModLyng-1.4.7-UARK */
/* pragma lynguent exported by Verilog-A Spectre 6.1 */

`include "disciplines.vams"

module OpAmp(outp, inp, inn, Vcc, Vss);
  //
  parameter real avol = 200e3;
  //
  parameter real f1 = 1.0e6;
  //
  parameter real sr = 0.5e6;
  //
  parameter real temp = 25.0;
  //
  parameter real rin = 1e6;
  //
  parameter real ib = 0.0;
  //
  parameter real ibtc = 0.0;
  //
  parameter real vos = 0.0;
  //
  parameter real vostc = 0.0;
  //
  parameter real ibos = 0.0;
  //
  parameter real ibostc = 0.0;
  //
  parameter real vswing = 12.0;
  //
  parameter real rout = 75.0;
  //
  parameter real iout = 40e-3;
  //Resistor value in LPF (calculated in plugin)
  parameter real RLPF_PAR = 398k;
  //The value for the LPF (calculated in plugin)
  parameter real CLPF_PAR = 80n;
  //What to saturate the output voltage as (calculated in plugin)
  parameter real VCLIP = 3.0;
  //
parameter real vccnom = 15.0;
  //
parameter real Extra_Pole1 = 1;
  //
parameter real Extra_Pole2 = 1;
  //
  //Output
  inout outp;
electrical outp;
  //Noninverting input
inout inp;
electrical inp;
  //Inverting input
inout inn;
electrical inn;
  //Positive power supply [5, 18]
inout Vcc;
electrical Vcc;
  //Negative power supply [-18, -5]
inout Vss;
electrical Vss;
electrical Ground_electrical;
ground Ground_electrical;

  //
electrical before_Rout;
  //
electrical inp_afterdc;
real vcc_thresh;
real gain_term;
  //
electrical limiter;
real K;
  //
electrical N7;
  //
electrical N14;
  //
electrical N12;
  //
electrical N22;
parameter real inlined_IC_CLPF = -1.0e38;
parameter real inlined_IS0_C3 = 1.0e-14;
parameter real inlined_Area_C3 = 1.0;
parameter real inlined_N_C3 = 1.0;
parameter real inlined_CJ0_C3 = 0.0;
parameter real inlined_VJ_C3 = 0.7;
parameter real inlined_M_C3 = 0.5;
parameter real inlined_TT_C3 = 1.0e-12;
parameter real inlined_IS0_C9 = 1.0e-14;
parameter real inlined_Area_C9 = 1.0;
parameter real inlined_N_C9 = 1.0;
parameter real inlined_CJ0_C9 = 0.0;
parameter real inlined_VJ_C9 = 0.7;
parameter real inlined_M_C9 = 0.5;
parameter real inlined_TT_C9 = 1.0e-12;
parameter real inlined_R_C4 = 100;
parameter real inlined_C_C6 = 374.5e-12;
parameter real inlined_IC_C6 = -1.0e38;
parameter real inlined_C_C8 = 374.5e-12;
parameter real inlined_IC_C8 = -1.0e38;
parameter real inlined_R_C10 = 100;
//
branch (limiter, Ground_electrical) b_filt;
//
branch (before_Rout, Ground_electrical) b_out;
//
branch (inp_afterdc, inn) b_in;
//
branch (limiter, Ground_electrical) b_lpf;
//Uses a resistance to draw current from the power supply.
branch (Vcc, Ground_electrical) b_vcc;
//Uses a resistance to draw current from the power supply.
branch (Vss, Ground_electrical) b_vss;
//
branch (N12, Ground_electrical) br_secPole;
//
branch (N12, Ground_electrical) br_secPoleOut;
//
branch (N22, Ground_electrical) br_thirdPole;
//
branch (N22, Ground_electrical) br_thirdPoleOut;
//
branch (inp_afterdc, inn) inlined_b_Rin_res;
//
branch (before_Rout, outp) inlined_b_Rout_res;
//
branch (limiter, Ground_electrical) inlined_b_R_LPFP;
//
branch (limiter, Ground_electrical) inlined_b_CLPF;
//
branch (inp, inp_afterdc) inlined_b_DC_OFFSET_SRC;
//
branch (limiter, N7) inlined_pn_C3;
//
branch (Vcc, N7) inlined_b_C5;
//
branch (N14, Vss) inlined_b_C7;
//
branch (N14, limiter) inlined_pn_C9;
//
branch (inn, Ground_electrical) inlined_b_C0;
//
branch (inp, Ground_electrical) inlined_b_C1;
//
branch (inp, inn) inlined_b_C2;
//
branch (N12, Ground_electrical) inlined_b_C4;
//
branch (N12, Ground_electrical) inlined_b_C6;
//
branch (N22, Ground_electrical) inlined_b_C8;
//
branch (N22, Ground_electrical) inlined_b_C10;

analog begin
   // pragma lynguent code fragment "eb_Diode_Ideal"
   @(initial_step)
   begin
      if (inlined_IS0_C3 <= 0.0)
         begin
            $strobe ("%M: Saturation current must be > 0.0");
            $finish (0);
         end
      if (inlined_N_C3 <= 0.0)
         begin
            $strobe ("%M: Emission coefficient N must be > 0.0");
            $finish (0);
         end
      if (inlined_CJ0_C3 > 0.0)
         $strobe ("%M: Selected architecture does not support junction capacitance");
   end
   // pragma lynguent code fragment "eb_Diode_Ideal"
@(initial_step)
begin
  if (inlined_ISO_C9 <= 0.0)
  begin
    $strobe("%M: Saturation current must be > 0.0");
    $finish (0);
  end
  if (inlined_N_C9 <= 0.0)
  begin
    $strobe("%M: Emission coefficient N must be > 0.0");
    $finish (0);
  end
  if (inlined_CJ0_C9 > 0.0)
    $strobe("%M: Selected architecture does not support junction capacitance");
end
// pragma lynguent code fragment "eb_Resistor"
V(inlined_b_Rin_res) <+ rin * I(inlined_b_Rin_res);
// pragma lynguent code fragment "eb_Resistor"
V(inlined_b_Rout_res) <+ rout * I(inlined_b_Rout_res);
// pragma lynguent code fragment "eb_Resistor"
V(inlined_b_R_LPF) <+ RLPF_PAR * I(inlined_b_R_LPF);
// pragma lynguent code fragment "eb_Capacitor"
if (analysis("ic") && inlined_IC_CLPF != -1.0e38)
  V(inlined_b_CLPF) <+ inlined_IC_CLPF;
else
  I(inlined_b_CLPF) <+ ddt((CLPF_PAR * V(inlined_b_CLPF)));
// pragma lynguent code fragment "eb_VsourceDC_PE"
V(inlined_b_DC_OFFSET_SRC) <+ vos;
// pragma lynguent code fragment "eb_VsourceDC_PE"
V(inlined_b_C5) <+ VCLIP;
// pragma lynguent code fragment "eb_VsourceDC_PE"
V(inlined_b_C7) <+ VCLIP;
// pragma lynguent code fragment "eb_IsourceDC_PE"
I(inlined_b_C0) <+ ib;
// pragma lynguent code fragment "eb_IsourceDC_PE"
I(inlined_b_C1) <+ ib;
// pragma lynguent code fragment "eb_IsourceDC_PE"
I(inlined_b_C2) <+ ibos;
// pragma lynguent code fragment "eb_Resistor"
V(inlined_b_C4) <+ inlined_R_C4 * I(inlined_b_C4);
// pragma lynguent code fragment "eb_Capacitor"
if (analysis("ic") && inlined_IC_C6 != -1.0e38)
  V(inlined_b_C6) <+ inlined_IC_C6;
else
I(inlined_b_C6) <+ ddt((inlined_C_C6 * V(inlined_b_C6)));  // pragma lynguent code fragment "eb_Capacitor"
if (analysis("ic") && inlined_IC_C8 != -1.0e38)
    V(inlined_b_C8) <+ inlined_IC_C8;
else
    I(inlined_b_C8) <+ ddt((inlined_C_C8 * V(inlined_b_C8)));  // pragma lynguent code fragment "eb_Resistor"
V(inlined_b_C10) <+ inlined_R_C10 * I(inlined_b_C10);  // pragma lynguent code fragment "PowerSupplyCode"
//Sequential Code Fragment
//Now calculate the K term for the maximum current
K = avol / (iout * RLPF_PAR);  // pragma lynguent code fragment "b_filt"
//Branch Equations b_filt
I(b_filt) <+ 0;
// pragma lynguent code fragment "b_out"
//Branch Equations b_out
V(b_out) <+ V(br_thirdPoleOut);
// pragma lynguent code fragment "b_in"
//Branch Equations b_in
I(b_in) <+ 0;
// pragma lynguent code fragment "b_lpf"
//Branch Equations b_lpf
//Voltage controlled current source. Uses K which is calculated in the sequential code
//The tanh term ensures a slew rate of 0.5 V/us
I(b_lpf) <+ -(iout * tanh(K * V(b_in))));  // pragma lynguent code fragment "br_secPole"
I(br_secPole) <+ -(0.01 * V(b_filt));  // pragma lynguent code fragment "br_secPoleOut"
I(br_secPoleOut) <+ 0;
// pragma lynguent code fragment "br_thirdPole"
I(br_thirdPole) <+ -(0.01 * V(br_thirdPoleOut));  // pragma lynguent code fragment "br_thirdPoleOut"
I(br_thirdPoleOut) <+ 0;
// pragma lynguent code fragment "eb_Diode_Ideal"
I(inlined_pn_C3) <+ inlined_Area_C3 * inlined_IS0_C3 *
    limexp(V(inlined_pn_C3) / (inlined_N_C3 * $vt($temperature))) - 1.0);  // pragma lynguent code fragment "eb_Diode_Ideal"
I(inlined_pn_C9) <+ inlined_Area_C9 * inlined_IS0_C9 *
    limexp(V(inlined_pn_C9) / (inlined_N_C9 * $vt($temperature))) - 1.0);  // pragma lynguent code fragment "eb_Diode_Ideal"
end
endmodule