SOFTWARE DESIGN AND IMPLEMENTATION
FOR SYMBOLIC MODELING OF SWITCHED NETWORKS

MINGYANG ZHOU
Spring 2014

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Electrical Engineering and Engineering Science
with honors in Electrical Engineering

Reviewed and approved* by the following:

Jeffrey Mayer
Associate Professor of Electrical Engineering
Thesis Supervisor

John Mitchell
Professor of Electrical Engineering
Honors Advisor

* Signatures are on file in the Schreyer Honors College.
ABSTRACT

Power converter is an important electronic component that has been applied to most of the electronic systems operated in our daily life. Design of high-quality power converter is extremely important to assure efficient performance of these systems. Owing to this demand, researches on improving the analysis method for the small scale power converters have been conducted to achieve a high accuracy of the analysis result. Currently, one of the most popular mathematical methods to be practiced to perform analysis on these electronic components is called state space model algorithm, which is also an easily programmable algorithm that can be conducted by a digital computer. A lot of advanced analysis method is developed based on the state space model analysis result of the power converter circuits. However, the analysis process performed by human is time consuming when the power converter circuit becomes extremely complex, even with some professional computer program, like MATLAB. In addition, study of state space model theory will also be a pain for design engineers which they don’t necessary need to know.

The idea of this research project comes from the motivation to save the time to generate a state space model for power converter circuits and improve the efficiency of the analysis process. With the help of a developed framework in C++ program, it is applicable to design a software program that implements the state space model algorithm on a power converter circuit controlled by a control system. Users will only need to enter an input file that describes the circuit system they want to perform analysis on, and the software will give the expected result within a short time. Additionally, this research project will provide a symbolic format solution instead of numerical result to give a more general and easy-understanding solution for users.
# TABLE OF CONTENTS

List of Figures ............................................................................................................. iv  
Recogngements ......................................................................................................... vi  
Chapter 1 Introduction ............................................................................................... 1  
  1.1 Motivation ........................................................................................................... 1  
  1.2 Summary of Contributions .................................................................................. 2  
  1.3 Organization ....................................................................................................... 3  
Chapter 2 Mathematical Theory ............................................................................... 4  
  2.1 Generalized State Space Model of a Switched Network .................................. 4  
  2.2 Power Converter Example .................................................................................. 6  
  2.3 Graph Theory for Electrical Network Analysis ................................................ 11  
  2.4 State Space Model of an Electrical Network .................................................... 13  
    2.4.1 Preliminaries .................................................................................................. 14  
    2.4.2 State Equation ............................................................................................... 15  
    2.4.3 Output Equation ............................................................................................ 16  
  2.5 Switching Surfaces ............................................................................................. 22  
Chapter 3 Software Design and Implementation .................................................... 25  
  3.1 Interconnected System Modeling Framework ................................................ 25  
    3.1.1 Component Models ....................................................................................... 25  
    3.1.2 Analysis Methods ......................................................................................... 27  
    3.1.3 Software Modules ......................................................................................... 27  
  3.2 Circuit Element Classes ..................................................................................... 28  
  3.3 Control Block Classes ....................................................................................... 34  
  3.4 Electrical Circuit Analysis Method .................................................................... 39  
  3.5 Control System Analysis Method ...................................................................... 54  
  3.6 Combined Analysis on PPLTI Electronic Network ........................................... 65  
Chapter 4 Test Cases .................................................................................................. 69  
  4.1 Electrical Circuit .................................................................................................. 69  
  4.2 Control System Diagram ................................................................................. 77  
  4.3 Combined System with Electrical Circuit and Control System ......................... 83  
Chapter 5 Conclusions and Future Work ................................................................... 97  
Appendix A Graph Definitions .................................................................................. 100  
Appendix B State-Space Modeling of RLCM Networks ........................................... 102  
Appendix C Netlist for Test Cases ........................................................................... 107
LIST OF FIGURES

Figure 2-1 Circuit of a dc-to-dc buck converter with direct user control of duty cycle...........7
Figure 2-2 Example of representing an electrical network as a graph.................................11
Figure 2-3 Circuit schematic of a dc-to-dc buck converter.............................................17
Figure 3-1 Electronic component class structure hierarchy of original software ....................28
Figure 3-2 Updated hierarchy for ElectricalElement......................................................32
Figure 3-3 Control system block class structure hierarchy of updated software ..................35
Figure 3-4 General flow diagram of electronic analysis algorithm ..................................40
Figure 3-5 Buck-converter circuit....................................................................................41
Figure 3-6 The signal flow diagram of the information retrieved process ..........................42
Figure 3-7 The process to set up graph in the software ....................................................44
Figure 3-8 The analysis function for the Original software .............................................45
Figure 3-9 Updated process to store the input information for electronic circuit .............48
Figure 3-10 Updated analysis for electronic circuit with switchable component ..........49
Figure 3-11 Analysis method 1 in updated analysis process ...........................................50
Figure 3-12 Analysis method 2 in updated analysis process ...........................................52
Figure 3-13 General flow diagram of control system analysis algorithm .......................55
Figure 3-14 Sample control system case.................................................................57
Figure 3-15 The information retrieving process for control system ...............................58
Figure 3-16 The process to set up graph in the software ..............................................59
Figure 3-17 The analysis function for the control system ..............................................61
Figure 4-1 Test case 4.1-1 for electronic circuit .........................................................70
Figure 4-2 Test case 4.1-2 for electronic circuit .............................................................72
Figure 4-3 Test case 4.2-1 for control system ...............................................................77
Figure 4-4 Test case 4.2-2 for control system ...............................................................78
Figure 4-5 Test case 4.2-3(part1) for control system.................................................................80
Figure 4-6 Test case 4.2-3(part2) for control system.................................................................80
Figure 4-7 Test case 4.2-4 for control system..............................................................................82
Figure 4-8 Test case 4.3-1 for a combined electronic network.....................................................84
ACKNOWLEDGEMENTS

The accomplishment of this thesis could not be achieved without the great support from people around me. First, I would like to give my greatest appreciation for my thesis advisor, Dr. Jeffrey Mayer, for his continuous inspiration and guidance to lead me through the challenges of this project. His critical suggestion for the project and thorough review of the thesis contribute to the success of the research work. I also want to give my special thanks to my honors advisor, Dr. John Mitchell, who has provided me with a lot of guidance to achieve success in my undergraduate study with Schreyer Honors College. Special thanks also go to Dr. Christine Masters and Melissa F. Fink, who have helped me to accomplish the academic study with Engineering Science department. I also owe my thanks to my academic advisor, Dr. Jian Xu, from Engineering Science department who has given me his thorough thinking on my research thesis.

Last but not least, I want to thank my parents for their constant encouragement, support and guidance, which I cannot achieve what I have without in my life.
Chapter 1
Introduction

1.1 Motivation

Power converters are critical components of most electronic systems, including information technology systems, industrial automation systems, and electric vehicle propulsion systems. Due to ever-present demand for these systems to be smaller, more efficient, and less costly, power converter design remains an active area of research. More specifically, conventional design methods rely on small signal models of the converter dynamics obtained through various averaging techniques [1][2], but the assumptions underlying these averaging techniques are becoming tenuous as the size of components becomes smaller.

Power converter design methods based on so-called generalized state-space models (GSSM) offer promising alternatives to conventional methods relying on small signal models, but these methods often require complex computer codes and significant effort to define the GSSM for a specific converter. For example, Kriventsov derived a method for stability assessment of power converters by applying Floquet theory to GSSMs with piecewise constant inputs [3]. Implementing that method involved writing approximately 2000 lines of fairly sophisticated MATLAB code. Applying the method to a new converter requires additional effort on the part of a design engineer or researcher to define a GSSM that may be several hundred lines of MATLAB code. That level of effort simply to define the GSSM of a converter presents a significant barrier to the acceptance of new GSSM methods by design engineers.
The main objective of this research is to design and implement software for an engineering design tool to facilitate generalized state-space modeling of power converters. The input to this software is a so-called netlist description of the power converter that includes information about the specifications and interconnection of circuit elements and control blocks for the converter. The output is a GSSM that is comprised of the following:

1. A set of continuous-time linear time-invariant state-space models (LTI models). Each LTI model corresponds to a particular circuit topology defined by the conduction states of the switches.

2. A set of switching surface functions and maps each of which defines the transition from one LTI model to another.

Details regarding the mathematical form of a GSSM and algorithms for generating one for a power converter are presented in Chapter 2.

An important and unusual feature of the software is that the GSSM that it produces is in symbolic, as opposed to numeric, form. This will allow the GSSM to be used with symbolic analysis methods. It also allows for simple parameterization and re-parameterization of the model when used with numerical analysis methods.

The implementation of the software is based on a previously developed C++ framework: Interconnected System Modeling Framework (ISMF). Details regarding the design and implementation of the software are present in Chapter 3.

1.2 Summary of Contributions

The main accomplishments of this research project are the design and implementation of software modules for generating symbolic matrices for the following:

- LTI models of a switched RLCM network.
- LTI model of a control system.
- Switching surface functions for a combined RLCM network and control system.

1.3 Organization

The remainder of this thesis is organized in four chapters. In Chapter 2, the mathematical form of a generalized state space model is presented along with the algorithms for generating such a model for a switched network. Chapter 3 introduces the software design process within the C++ framework. Chapter 4 provides a group of test cases that were developed during the course of the project. Chapter 5 concludes the thesis and provides some suggestions for future work.
Chapter 2
Mathematical Theory

The algorithm used to generate a generalized state-space model (GSSM) for a power converter is based on state space modeling of RLCM networks, which are networks comprised of resistors, inductors, capacitors, mutual inductances. State space modeling of RLCM networks is based, in turn, on graph theory. In this chapter, the form of a GSSM is presented along with aspects of graph theory and state space modeling of RLCM networks that are required to generate a GSSM.

2.1 Generalized State Space Model of a Switched Network

A GSSM is comprised of a set of linear time-invariant state space models (LTI models) along with a set of switching surface functions and maps for transitions between the respective LTI models [3][4]. Each LTI model and switching surface function corresponds to a particular circuit topology defined by the conduction states of the switches. Thus, the mathematical form of a GSSM is

\[
\dot{x}_m = A_m x_m + B_m u \\
y = C_m x_m + D_m u \\
\sigma_m = \sigma_{sm} x_m + \sigma_{um} u + \sigma_{dm} d + \sigma_{cm} \\
x_n = R_{nm} x_m
\]

where \( m \in [1, 2, \ldots, M] \) is the index of a particular topology, \( M \) is the total number of topologies, and \( n \in [1, 2, \ldots, M] \) is the index of the next topology[5,6,7].
Equation 2-1 is referred to as the state equation. It relates the rate of change of the state variable \( \dot{x} \) to the present value of the state variable \( x_m \) and the present value of the input variable \( u \). In general, \( x_m \) and \( u \) are vectors. Each element of \( x_m \) is a capacitor voltage, an inductor current, or an integrator state. The particular capacitor voltages and inductor currents to be included in \( x_m \) for a given topology \( m \) are determined using graph theory as described in Section 2.3. Each element of \( u \) corresponds to an independent voltage source, an independent current source, or an input signal. The matrices \( A_m \) and \( B_m \) for a given topology are determined through state space modeling of the RLCM network as described in Section 2.4 and state space modeling of the control system as described in Section 2.5.

Equation 2-2 is referred to as the output equation. It relates the output variable \( y \) to the state and input variables. In general, \( y \) is a vector. Each element of \( y \) represents a voltage, current, or signal of interest. Like \( A_m \) and \( B_m \), matrices \( C_m \) and \( D_m \) are determined through state space modeling of the RLCM network and control system.

Equation 2-3 is referred to as a switching surface function. When this function is evaluated, each element of the resulting \( \sigma_m \) can be thought of as a signed distance from a surface in the state space. This distance depends on the instantaneous duty cycle \( d \) or normalized time

\[
d = \frac{t}{T_s}
\]

(2-5)

where \( T_s \) is the switching period of the converter. A positive distance indicates that the system is to remain in the present topology \( m \). A zero or negative distance indicates that the system is to switch from present topology \( m \) to a new topology \( n \). The \( R \) map in (2-4) is used to transition the state variable from topology \( m \) to topology \( n \).

Generalized state space models provide a concise mathematical representation for switched RLCM networks that is can be used to determine the transient or periodic steady-state
response of the network. The process of deriving the GSSM for a particular converter is straightforward but tedious, making it a good candidate for automation in an engineering design tool. The process can be described by the following algorithm

1. Enumerate the topologies. There is one topology for each combination of switch conduction states, so the total number is \( M = 2^N \) where \( N \) is the number of switches.

2. For each topology \( m \in [1, 2, \ldots, M] \):
   a. Determine the LTI model matrices \( A_m, B_m, C_m \) and \( D_m \) for that particular RLCM network.
   b. Determine the switching surface matrices \( \sigma_x, \sigma_u, \sigma_d, \sigma_c \) by identifying each condition for which this topology is to be exited.
   c. Determine the state variable transition map associated with each switching surface.

### 2.2 Power Converter Example

A dc-to-dc buck converter with direct user control of duty cycle will be used to illustrate generalized state space modeling of a power converter. The same converter will be used subsequently as a test case for the software. A circuit schematic diagram for the converter is shown in Figure 2-1.
Figure 2-1 Circuit of a dc-to-dc buck converter with direct user control of duty cycle

The system as two inputs: the independent voltage source $v_1$ and the reference duty cycle $D^\ast$. Thus, the vector for the input variable is

$$\mathbf{u} = \begin{bmatrix} v_1 \\ D^\ast \end{bmatrix}$$

(2-6)

The capacitor voltage and inductor current are both of interest. Thus, the vector for the output variable is

$$\mathbf{y} = \begin{bmatrix} v_c \\ i_L \end{bmatrix}$$

(2-7)

As there is one switch and one diode in the network, there are four topologies. For Topology 1, the switch is off and the diode is off. Consequently, the inductor current is identically zero and not a state variable. This particular topology corresponds to the third and final part of a switching period when the converter is operating in the so-called discontinuous
conduction mode. As a result, there is a single switching surface that depends on $d$ but not on $x$ or $u$.

$$x_1 = \begin{bmatrix} v_c \end{bmatrix}$$

$$A_1 = \begin{bmatrix} -\frac{1}{RC} \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\sigma_{s1} = \begin{bmatrix} 0 \end{bmatrix}$$

$$\sigma_{u1} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$\sigma_{d1} = \begin{bmatrix} -1 \end{bmatrix}$$

$$\sigma_{c1} = \begin{bmatrix} 1 \end{bmatrix}$$

$$R_{31} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For Topology 2, the switch is off and the diode is on. This particular topology corresponds to second and final part of the switching period when the converter is operating in the continuous conduction mode and to the second part of the switching period when the converter is operating in the discontinuous conduction mode. Thus, there are two switching surfaces.

$$x_2 = \begin{bmatrix} v_c \\ i_L \end{bmatrix}$$
\[
A_2 = \begin{bmatrix}
-\frac{1}{RC} & \frac{1}{C} \\
-\frac{1}{L} & 0
\end{bmatrix}
\]

\[
B_2 = \begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

\[
C_2 = \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]

\[
D_2 = \begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

\[
\sigma_{x_2} = \begin{bmatrix}
0 & 0 \\
0 & 1
\end{bmatrix}
\]

\[
\sigma_{u_2} = \begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}
\]

\[
\sigma_{d_2} = \begin{bmatrix}
-1 \\
0
\end{bmatrix}
\]

\[
\sigma_{e_2} = \begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

\[
R_{32} = \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}, \quad R_{12} = \begin{bmatrix}
1 & 0
\end{bmatrix}
\]

For Topology 3, the switch is on and the diode is off. This particular topology corresponds to the first part of the switching period.

\[
x_3 = \begin{bmatrix}
v_C \\
i_L
\end{bmatrix}
\]
\[ A_3 = \begin{bmatrix} \frac{1}{RC} & \frac{1}{C} \\ \frac{1}{L} & 0 \end{bmatrix} \]

\[ B_3 = \begin{bmatrix} 0 & 0 \\ \frac{1}{L} & 0 \end{bmatrix} \]

\[ C_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]

\[ D_3 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \]

\[ \sigma_{x3} = \begin{bmatrix} 0 & 0 \end{bmatrix} \]

\[ \sigma_{u3} = \begin{bmatrix} 0 & 1 \end{bmatrix} \]

\[ \sigma_{d2} = [-1] \]

\[ \sigma_{c3} = [0] \]

\[ R_{23} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]

For Topology 4, the switch is on and the diode is on. This is a degenerate topology, as the independent voltage source is short circuited by the switch and diode. Consequently, the state vector and all matrices are regarded as null.
2.3 Graph Theory for Electrical Network Analysis

Graph theory provides a systematic approach to modeling the topology or structure of an electrical network. Fundamentally, a graph is a mathematical structure comprised of two related sets:

- A finite, nonempty node set $N$.
- A possibly empty edge set $E$ comprised of node pairs.

A graph can also be represented by a diagram and by various matrices. Diagrams facilitate visualization; matrices are useful for computer models. In a diagram, a node is represented as a point, and an edge is represented as a line connecting two different nodes. An example of the conversion from a circuit schematic to a graph is shown in Figure 2-2.

![Figure 2-2 Example of representing an electrical network as a graph](image)

A brief introduction to graph terminology is given in Appendix A. A few terms are especially important as they correspond to Kirchhoff's voltage law (KVL) and Kirchhoff's current law (KCL). Those terms are loop, cutset, and tree.

A loop is a subgraph $G_s$ of a connected graph $G_N$, if:

1. $G_s$ is connected.
2. Every node of $G_s$ has exactly two branches of $G_s$ incident at it.
A cutset is a set of branches of a connected graph $G_n$ if:

1. The removal of the set of branches results in a graph that is not connected.
2. After the removal of the set of branches, the restoration of any one branch from the set will result in a graph that is connected.

A tree is a subgraph $G_s$ of a connected graph $G_n$ if

1. $G_s$ is connected
2. Contains all nodes of $G_n$
3. $G_s$ contains no loops

The set of edges of $G_n$ that are not in a tree $G_s$ is called the co-tree. An edge that belong to a tree is called a branch; while an edge that belongs to the co-tree is called a link. A tree provides the means to systematically identify cutsets and loops within a graph and hence formulate independent KCL and KVL equations for an electrical network. [5,8,9,10,11,12]

The advantage of graph theory is obvious. A graph simplifies an electrical network visually, and it can be easily represented by matrices used to formulate state space model. Converting graph into matrices also allows it to be easily programmed on a computer.

A loop matrix $B_a$ represents loops with $a$ branches and $n$ oriented loops, which is defined as an $n*b$ matrix [5]:

$$
B_a = [b_{ij}] 
$$

(2-8)

where:

$$
b_{ij} = 1 \text{ if branch } j \text{ is in loop } i, \text{ and their directions agree}$$
$$
b_{ij} = -1 \text{ if branch } j \text{ is in loop } i, \text{ and their directions oppose}$$
$$
b_{ij} = 0 \text{ if branch } j \text{ is not in loop } i$$

With respect to KVL, if the branch voltage is represented by a voltage vector $v$, which has the same number of elements as the number of columns in $B_a$, then:
A cutset matrix is used to develop KCL equation. It is defined as a matrix $D_a$, which represents a cutset with $b$ branches and $n$ oriented cutsets. The matrix is therefore a $n*b$ matrix [5]:

$$D_a = [d_{ij}]$$  \hspace{1cm} (2-10)

where:

$$d_{ij} = 1 \text{ if branch } j \text{ is in cutset } i, \text{ and their directions agree}$$

$$d_{ij} = -1 \text{ if branch } j \text{ is in cutset } i, \text{ and their directions oppose}$$

$$d_{ij} = 0 \text{ if branch } j \text{ is not in cutset } i$$

If we have a vector of current that represent the branch current values for each cutset, then the KCL can be represented as:

$$D_a \mathbf{i} = \mathbf{0}$$  \hspace{1cm} (2-11)

As can be seen at the beginning of Appendix B, a matrix equation to show KCL and KVL is derived. While most of the variables included in that derivation is explained above, $F$ matrix represents the submatrix of a cutset matrix for different component branches and links in the electronic network.

2.4 State Space Model of an Electrical Network

As described in the introduction, the goal of this project is to design and implement software that can generate a symbolic generalized state space model for a power converter. Consequently, an important part of this project is state space modeling of electrical networks. The derivation of the state equation for an RLCM network is covered thoroughly in *Computer-sided analysis of Electronic Circuits*[5], so only the key results are given in this section (a
summary of the derivation itself is given in Appendix B). A new derivation of the output equation for RLCM networks is presented.

2.4.1 Preliminaries

Two assumptions are made in the state-space modeling of any RLCM network:

- There are no loops consisting of independent voltage sources only.
- There are no cutsets consisting of independent current sources only.

If either of these assumptions is violated, the network has no unique solution. Based on these assumptions, a normal tree or a proper tree can be constructed to contain the following:

- all independent voltage sources.
- no independent current sources.
- as many capacitors as possible.
- as few inductors as possible.

If the subscript $\mathfrak{Z}$ is used to denote elements of the normal tree and $\ell$ is used to denote links of the corresponding co-tree, then the network branch voltages and branch currents may be partitioned as follows:

$$
\mathbf{v} = \begin{bmatrix} v_{E,3} & v_{C,3} & v_{R,3} & v_{L,3} & v_{J,\ell} & v_{L,\ell} & v_{R,\ell} & v_{C,\ell} \end{bmatrix}^T
$$

$$
\mathbf{i} = \begin{bmatrix} i_{E,3} & i_{C,3} & i_{R,3} & i_{L,3} & i_{J,\ell} & i_{L,\ell} & i_{R,\ell} & i_{C,\ell} \end{bmatrix}^T
$$

The topology of the network can then be described using the fundamental cutset matrix $\mathbf{D}$, which corresponds to the generalized form of KCL, and the fundamental loop matrix $\mathbf{B}$, which corresponds to the generalized form of KVL:
The \( i-v \) characteristics for the RLCM elements of the network are described as:

\[
\begin{align*}
\text{R:} & \quad v_R = R_i \quad \text{or} \quad i_R = Gv_R \\
\text{C:} & \quad i_C = C \frac{dv}{dt} \\
\text{LM:} & \quad \begin{bmatrix} v_{L1} \\ v_{L2} \end{bmatrix} = \begin{bmatrix} L_1 & M \\ M & L_2 \end{bmatrix} \frac{di}{dt} \begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix}
\end{align*}
\]  

\( 2.4.2 \) State Equation

Equations 2-12 to 2-18 constitute the original network equations for the RLCM network as a system of differential algebraic equations (DAEs). A derivation showing the transformation of the DAEs to the state equation of a state space model is given in Chua and Lin’s book - *Computer-sided analysis of Electronic Circuits* [5].

The state variables selected for an RLCM network include the voltage across the tree capacitors and current through the co-tree inductors in co-tree:

\[
x = \begin{bmatrix} v_{C3} \\ i_{L_4} \end{bmatrix}
\]  

\( 2-19 \)
In the subscript, \( C \) or \( L \) indicates that the variable is associated with a capacitors or an inductor, and \( \mathcal{S} \) or \( \ell \) indicates that the element is in the tree or co-tree.

The input variables include all the independent sources:

\[
\mathbf{u} = \begin{bmatrix} \mathbf{V}_{E3} \\ \mathbf{I}_{j\ell} \end{bmatrix}
\]  
(2-20)

As described in Appendix B, the entire voltage source is pushed into tree and all the current source is pushed into co-tree. With the help of KCL, KVL and the V-I properties of different components, the procedures successfully set up the equation that relates the state variables and input variables. The original software includes this algorithm to conduct this process step by step to generate the \( \mathbf{A} \) and \( \mathbf{B} \) matrices.

### 2.4.3 Output Equation

In *Computer-sided analysis of Electronic Circuits*, there is an explicit derivation of the \( \mathbf{A} \) and \( \mathbf{B} \) matrices, but there is only a brief mention of the \( \mathbf{C} \) and \( \mathbf{D} \) matrices [5]. For generalized state space modeling of power converters, the output vector can be limited to include the voltages and currents associated with independent sources and capacitors and inductors – the voltages and currents associated with resistors are not needed to define the switching surfaces. As there will only be one full ultimate test case, the \( \mathbf{C} \) and \( \mathbf{D} \) matrices can be further simplified to match the interest of the unique circuit in this project. The value of interest in this project will be the voltage and current values across the diodes, which is significant to generate switching surface in the output file as explained in the next chapter. In order to show the derivation of \( \mathbf{C} \) and \( \mathbf{D} \) matrix clearly, an electronic circuit for the final test case that has been introduced in Section 2.2 marked with the voltage and current values for different elements is shown below:
Referring to Appendix B, it is clear that all the electronic components in the state space model would be categorized into different tree and co-tree structures. For this circuit, although the output current and voltage values that we are interested in just includes the V-I value across the diode, a more generalized output vector that includes all the V-I value for different components is demonstrated here. This output vector does not include any V-I values across the resistor, which will not affect the V-I value across the diode for this circuit. Therefore, the y vector should be:

\[
y = \begin{bmatrix}
V_{E3} \\
V_{C3} \\
V_{L3} \\
V_{\ell} \\
V_{L\ell} \\
V_{C\ell} \\
I_{E3} \\
I_{L3} \\
I_{\ell} \\
I_{L\ell}
\end{bmatrix}
\]  

(2-21)

\(V_{E3}\) and \(I_{E3}\) represent the voltage and current of independent voltage source in the tree structure. \(V_{C3}\) and \(V_{C\ell}\) includes all the information of the voltage across the capacitors in both tree and co-tree structures. \(V_{L3}, V_{L\ell}, I_{L3}\) and \(I_{L\ell}\) include all the voltage and current values through the
inductors both in tree and co-tree structures. $V_{fE}$ and $I_{fE}$ represent the voltage and current values of independent current source.

For output equation, we have:

$$ y = Cx + Du + D1\dot{u} + \cdots $$

Since all the input sources are dc source in the test circuit, the derivative term of input variables would be removed. $x$ and $u$ are the same as specified in the previous sub section. Thus, the matrix format of output equation is:

$$ \begin{bmatrix} V_{E3} \\ V_{C3} \\ V_{L3} \\ V_{fE} \\ V_{fE} \\ V_{E3} \\ V_{C3} \\ V_{L3} \\ I_{E3} \\ I_{L3} \\ I_{fE} \\ I_{fE} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ D_{1,1} & D_{1,2} \\ C_{2,1} & C_{2,2} \\ D_{2,1} & D_{2,2} \\ C_{3,1} & C_{3,2} \\ D_{3,1} & D_{3,2} \\ C_{4,1} & C_{4,2} \\ D_{4,1} & D_{4,2} \\ C_{5,1} & C_{5,2} \\ D_{5,1} & D_{5,2} \\ C_{6,1} & C_{6,2} \\ D_{6,1} & D_{6,2} \\ C_{7,1} & C_{7,2} \\ D_{7,1} & D_{7,2} \\ C_{8,1} & C_{8,2} \\ D_{8,1} & D_{8,2} \\ C_{9,1} & C_{9,2} \\ D_{9,1} & D_{9,2} \\ C_{10,1} & C_{10,2} \\ D_{10,1} & D_{10,2} \end{bmatrix} \begin{bmatrix} V_{C3} \\ I_{fE} \end{bmatrix} \quad (2-22) $$

With respect to Appendix B, elements on every row of the C and D matrices can be derived as a simplified version for the test circuit and the derivation process is demonstrated row by row as following:

$$ V_{E3} = V_{E3} \quad (2-23) $$

$$ C_{1,1} = 0; \ C_{1,2} = 0; \ D_{1,1} = 1; \ D_{1,2} = 0; $$

$$ V_{C3} = V_{C3} \quad (2-24) $$
C2,1 = I; C2,2 = 0; D2,1 = 0; D2,2 = 0;

According to equation B-13 in Appendix B,

\[
V_{L3} = -L_{33} \cdot F41 \cdot \frac{d(I_{f\ell})}{dt} + (L_{f\ell} - L_{33} \cdot F42) \cdot \frac{d(I_{L3})}{dt}
\]

As the entire input independent sources are DC source in the test circuit, all the derivative terms of input variables are removed. In addition, since there is only one inductor in the circuit, there will be no mutual inductance between two different inductors. Thus, \(L_{f\ell}\) is zero. The equation becomes:

\[
V_{L3} = (-L_{33} \cdot F42) \cdot \frac{d(I_{L3})}{dt}
\]

Since there is only one inductor in the test circuit, it can only be pushed into the inductor tree or co-tree. If it is pushed into tree, \(d(I_{L3})/dt\) will be equal to zero as there is not inductor in the co-tree. Vice versa, if the inductor is pushed into co-tree, \(L_{33}\) will be removed. Thus, \(V_{L3}\) will always be 0. Thus:

\[
V_{L3} = 0
\]

C3,1 = 0; C3,2 = 0; D3,1 = 0; D3,2 = 0;

According to initial KVL equation B-4 in Appendix B,

\[
V_{f\ell} = -F11 \cdot t \cdot V_{E3} - F21 \cdot t \cdot V_{C3} - F31 \cdot t \cdot V_{R3} - F41 \cdot t \cdot V_{L3}
\]
F11.t is the transpose matrix of F11. Since voltage across diode is not related to the voltage across the resistance, it can be ignore in the matrices equation. From the previous row equation, it is known that $V_{L3}$ is always zero in our test circuit, thus the $V_{J\ell}$ can be represented as:

$$V_{J\ell} = -F_{11}.t * V_{E3} - F_{21}.t * V_{C3}$$ (2-26)

$$C_{4,1} = F_{21}.t; \ C_{4,2} = 0; \ D_{4,1} = F_{11}.t; \ D_{4,2} = 0;$$

With respect to the equation 8-24 in the book *computer-aided analysis of electronic circuits* [5]:

$$V_{L\ell} = F_{12}.t * V_{E3} + F_{22}.t * V_{C3} + F_{32}.t * V_{R3} + F_{42}.t * V_{L3}$$

From previous section, it is clear that $V_{R\ell}$ and $V_{L\ell}$ can be removed from the equation, thus:

$$V_{L\ell} = F_{12}.t * V_{E3} + F_{22}.t * V_{C3}$$ (2-27)

$$C_{5,1} = F_{22}.t; \ C_{5,2} = 0; \ D_{5,1} = F_{12}.t; \ D_{5,2} = 0;$$

According to equation B-12 in Appendix B:

$$V_{C\ell} = F_{14}.t * V_{E3} + F_{24}.t * V_{C3}$$ (2-28)

$$C_{6,1} = F_{24}.t; \ C_{6,3} = 0; \ D_{6,1} = F_{14}.t; \ D_{6,2} = 0;$$

According to KCL at equation B-3 in Appendix B:

$$I_{E3} = -F_{11} * I_{J\ell} - F_{12} * I_{L\ell} - F_{13} * I_{R\ell} - F_{42} * I_{C\ell}$$

$I_{R\ell}$ can be removed from the equation, as known from the previous equation step. $I_{C\ell}$ needs to be re written as a combination of state variables and input variables. According to B-14 in Appendix B:

$$I_{C\ell} = C_{C\ell} * \left( F_{14}.t * \frac{d(V_{E3})}{dt} + F_{24}.t * \frac{d(V_{C3})}{dt} \right)$$
For the test circuit, as all the input independent source are constant DC source, \( d(V_{E3})/dt \) should equal to zero. In addition, there is only one capacitor in the circuit, which can only be pushed into either a capacitor tree or a capacitor co-tree. If it is pushed into a capacitor tree, the capacitor co-tree will be empty and \( C_1 \) will be zero. Vice versa, the capacitor tree will be empty, and \( d(V_{E3})/dt = 0 \). Taking all the simplification specified above for the test case, \( I_{C_\ell} \) will be:

\[
I_{C_\ell} = 0
\]

\[
I_{E3} = -F11 * I_{j_\ell} - F12 * I_{L_\ell}
\]

\( C7,1 = 0; C7,2 = -F12; D7,1 = 0; D7,2 = -F11; \)

According to equation B-11 in Appendix B:

\[
I_{L3} = -F41 * I_{j_\ell} - F42 * I_{L_\ell}
\]

\( C8,1 = 0; C8,2 = -F42; D8,1 = 0; D8,2 = -F41; \)

\[
I_{j_\ell} = I_{j_\ell}
\]

\( C9,1 = 0; C9,2 = 0; D9,1 = 0; D9,2 = I; \)

\[
I_{L_\ell} = I_{L_\ell}
\]

\( C10,1 = 0; C10,2 = I; D10,1 = 0; D10,2 = 0; \)

In Summary, combining all the equations for \( C \) and \( D \) matrix in every row, I get the simplified \( C \) and \( D \) matrix for my test circuit:

\[
C = \begin{bmatrix}
0 & 0 \\
I & 0 \\
0 & 0 \\
F21. t & 0 \\
F22. t & 0 \\
F24. t & 0 \\
0 & -F12 \\
0 & -F42 \\
0 & 0 \\
0 & I
\end{bmatrix}
\]

(2-33)
Switching Surfaces

As described in Section 2.1, a switching surface is a function that defines the mathematical conditions at the instant for the circuit to switch from one topology to another. The particular form of switching surface used in this project is defined in [3]

\[ \sigma_m = \sigma_{xm}x_m + \sigma_{um}u_m + \sigma_{dm}d + \sigma_{cm} \]

\( \sigma_m \) is the switching surface that is a vector of m, m represents the number of different possibilities when the electrical network switches its topology. Thus the number of the switchable components in the circuit will determines the number of possibilities for the circuit to switch its possibility. Switching of topology in the electronic network happens when the switchable components change their status, \( \sigma_{xm} \) and \( \sigma_{um} \) are constant vector of m in front of state variables and input variables, which defines the mathematical condition when the state variable and input variables affect the switching condition. Take the final test case as an example, as diode switches its on and off status when the current flows through it changes the direction, we treated the current through the diode as the output value and extract the C and D matrix part that represents the symbolic equation consisting of state variables and input variables. Thus, C and D matrix for appropriate switchable component in the circuit defines \( \sigma_{xm} \) and \( \sigma_{um} \). Some of the switchable components are

\[
D = \begin{bmatrix}
I & 0 \\
0 & 0 \\
0 & 0 \\
F11.t & 0 \\
F12.t & 0 \\
F14.t & 0 \\
0 & -F11 \\
0 & -F41 \\
0 & I \\
0 & 0
\end{bmatrix}
\]
controlled by the control system, and for a control system, there are two types of signal that will affect the status of switchable components. One is a clock signal and one is a constant input signal. $\sigma_{dm}$ and $\sigma_{cm}$ defines the moment when time signal and constant signal converts the status of the switchable element.

In the Section 2.4, a derivation to generate C and D matrix is performed for the dc-dc buck converter circuit as shown previously. Since V-I value of diodes can determine part of the switching surface for the electronic circuit, they are treated as the value of interest in the output vector. To find out the voltage and current values across the diode, it is important to understand the special strategy for the software to deal with diode. In Chapter 3, it will explain that diode is considered as a 0-V voltage source when it is on, and a 0-A current source when it is off. When the diode is treated as a 0-V voltage source, it converts its status as the current through it becomes negative. For the circuit shown in Figure 2-3, with KCL on node 1, it is easy to get the equation that:

$$I_{diode} = I_{switch} - I_L$$ \hspace{1cm} (2-34)

When it is treated as a 0-A current source, it converts its status as the voltage across it becomes positive. Using KVL in loop 1 and 2, we will get two voltage equations:

$$V_{diode} = V_L + V_C$$ \hspace{1cm} (2-35)

$$V_{diode} = V_{switch} - V_{source}$$ \hspace{1cm} (2-36)

From the three equations, it is clear to see that the current across the diode is related to $I_{switch}$ and $I_L$, and the voltage across the diode is related to $V_L$, $V_C$ and $V_{switch}$. As switch is also an electronic component that can convert the topology of the circuit, it is treated similarly as diode in the software. Therefore, $I_{switch}$ and $V_{switch}$ could be considered as voltage and current through independent voltage source or current source. With respect to the equation derived from
KCL and KVL, the output $y$ vector should include $V_l, V_c, V_{\text{switch}}, I_{\text{switch}}$ and $I_l$. This explains the $y$ vector that is generated in Section 2.4 is relevant for the generation of switching surface.
Chapter 3
Software Design and Implementation

The software for this project was designed and implemented using object oriented programming techniques in C++. It utilized an existing C++ framework, the Interconnected System Modeling Framework (ISMF). This framework supports the notion that a system can be modeled as a collection of interconnected components and the notion that each component can have multiple mathematical representations. Thus, it provides distinct interfaces for component model developers and analysis method developers. In this chapter, the various component models and analysis methods that were developed for this project are described.

3.1 Interconnected System Modeling Framework

The Interconnected System Modeling Framework (ISMF) supports software modules and application programs in which a physical system is modeled as a collection of interconnected component models each of which may have multiple mathematical representations. It provides interfaces for component model developers, analysis method developers, and software module developers.

3.1.1 Component Models

Any class can be registered with the ISMF as a component model class, if it has two required member functions: describe_type and describe_instance. Inheritance from any particular base class is not required, but inheritance is allowed.
In *describe_type*, the component model developer can declare type information to be associated with the class. More specifically, the component model developer can do the following:

- declare a base class (inheritance)
- declare a data member as a part (composition)
- declare a data member as a parameter or a variable
- declare a data member as a connection

A data member declared as a part may correspond to an object or a pointer, but it may not correspond to a reference. The class for the part should be registered with the ISMF. A part declaration is qualified by an identifier and an optional description. A data member declared as a parameter or variable may correspond to an object or a variable of built-in type. In the case of an object, the class must have stream operators `<<<` and `>>` defined. A parameter or variable declaration is qualified by an identifier and optional description and optional symbol. A data member declared as a connection may correspond to an object, a pointer, or a list of pointers (`std::list`). Connections are qualified in various ways, including nodes, signal input port and signal output port.

In *describe_instance*, the component model developer can declare instance-specific information to be associated with an instance of the component model class. Typically a function member is declared as a function to be associated with the component model instance. The function declaration is qualified with a modeling range and the data members that serve as inputs and outputs. The modeling range refers to one of the following: `instantiatel`, `instantiate2`, `interconnect1`, and `interconnect2`. 
3.1.2 Analysis Methods

Any class can be registered with the ISMF as an analysis method class, if it has two required member functions: `setup` and `analyze`. In the `setup` function, it will extract all the installed information of the electronic network to be analyzed in the objective created by ISMF and create a new objective of the analysis algorithm model to store all the extracted information. After that, these algorithm models will call the `analyze` to conduct different analysis algorithm with the stored circuit network information and give back an analysis result.

3.1.3 Software Modules

With the type and instance information provided by a component model developer, the ISMF is capable of reading an interconnected system model (.ism) file provided by an end user. An .ism file is a text file that contains one record for each component in a system. The first two fields of the record are the component model type and an instance identifier. The remainder of the record is comprised of initialization values for parts, parameters, and connections. A component model object is instantiated and initialized as the record is read; this includes a call to `describe_instance` for the object. After all component model objects have been instantiated and initialized from the .ism file (modeling range `instantiate1`), any functions declared with modeling range `instantiate2` are called to complete the initialization process.

Once all component model objects have been initialized, the ISMF interconnects the objects by resolving their respective connections. This is a two-pass process (modeling range `interconnect1` and `interconnect2`).

Once all component model objects have been interconnected,
3.2 Circuit Element Classes

Each type of circuit element has a corresponding component model class that is ultimately derived from an `ElectricalComponent` base class. A complete class hierarchy is shown Figure 3-1:

![Figure 3-1 Electronic component class structure hierarchy of original software](image)

Within `ElectricalComponent` there are two important parameters: `_identifier` and `_serialnumber`. The first of these `_identifier` is defined from the name of the electronic component derived from the user-defined input files. In the output MATLAB file, the `_identifier` for each electronic component is shown as a symbolic expression. The `_serialnumber` is an integer number used to index the electrical component in the state-space model building algorithm.

Below the `ElectricalComponent` classes there are two sub classes: `ElectricalElement` and `MutualInductance`. The `ElectricalElement` class serves as a
base class for the common circuit element classes. The ElectricalElement class includes significantly more information than the ElectricalComponent class. This includes variables for the voltage across and current through the component, and two ElectricalTerminal objects that serve as connectors. Each ElectricalTerminal is resolved to an ElectricalNode object, which includes a list of pointers to ElectricalTerminal objects.

The connection relationship between different electronic components is defined by users to form a closed electronic circuit that needed to be analyzed. During the mathematical analysis process, the connectivity of each object in the circuit could be easily called from its electrical terminals, so that the topologies of the electrical circuit can be identified for the analysis algorithm. Another important member parameter in the ElectricalElement class is _type, which defines the unique data type of different electronic components. Corresponding to the creation of all the new parameters in this class, the initializer function and describe_type function are also modified to work with the new parameters. An object of ElectricalElement type will be initialized with a data type, voltage value and current value across the object. In the describe_type function, the ElectricalElement class can retrieve the voltage and current values from the user-defined input file, as well as the connectivity for terminals. Additional member functions to call the member parameters of ElectricalElement type object are also developed to allow the algorithm class to easily track the important properties of different electronic components.

Some other member parameters are developed to identify the roles and functionality of the electronic components in the mathematical algorithm, including is_branch, was_branch and operator <. In the Electrical Circuit Analysis process, the functionality of these parameters will be discussed in more details.
All the third level of subclass data types for electronic components developed from the **ElectricalElement** class. There are four different categories, including **EnergyStorageElement**, **Resistor**, **CurrentSource** and **Independent Source**.

**Resistor** and **CurrentSource** are already the highest level subclasses that no longer break down to a more specific subclass. Compared to their parent class, more detailed functionality and characteristics are added with respect to their electronic property performance in the electronic circuit. For **Resistor** class, the initial voltage value and current value are set to zero for calculation. Additionally, a parameter called **_resistance** is included to represent the resistance value of the resistor component. This parameter is added to the initialize and **describe_type** function of **Resistor** class, which is defined by users in the input file. As the voltage and current across the resistor follows the resistance law, where resistance equals voltage divide current, corresponding functions in **Resistor** class are created to allow the software to make use of the V-I relationship on resistance in the mathematical algorithm. For **CurrentSource**, as it is an energy source in the circuit value, the initial value of its **_voltage** is always set to zero, while that of **_current** is always set to 1. The V-I relationship does not follow a proportional linear function like resistors, and the voltage value could be set to any values without considering the current the source can generate.

Different from **Resistor** and **CurrentSource, IndependentSource** and **EnergyStorageElement** classes can be further broken down to a higher level class. The reason why we want to develop these two classes is to represent the state variables and input variables, which are required for the state-space model to generate a matrix equation result with these two types of variable. While **IndependentSource** does not add any extra characteristics compared to **ElectricalElement, EnergyStorageElement** are featured with new properties. In **EnergyStorageElement**, there is a **mutualitylist** which represent the mutual inductance
that they generate with the electronic components connected to them. The member function
interconnect2 provide the functionality for the EnergyStorageElement type object to
generate MutualInstance. Another important reason to create EnergyStorageElement
class is to identify the state variables to generate state-space model matrix.

Below IndependentSource, there is VoltageSource which represent the voltage
source component in the electronic circuit. As voltage source is a power source that generates
voltages, the initial value of _current is set to be 0 and the _voltage value is set to be 3. The
linear V-I relationship doesn’t work with voltage source either and any current value across the
voltage source will not affect the voltage value of the voltage source. Capacitors and
Inductors are developed under EnergyStorageElement class to represent capacitors and
inductors in RLCM electronic circuit. In Capacitor class, there are unique features representing
the electronic property of capacitors. _capacitance is a member parameter created in this class
to store the capacitance value of the real capacitors, which will be defined by users and taken care
by the describe_type function in this class. With respect to the V-I relation across capacitors
in the circuit, where current equals the derivative of voltage over time times capacitance, there is
member function called compute_current defining the unique V-I relationship for
Capacitors. Inductors class are developed to involve all the electronic properties of
inductors, where there is _inductance member parameter to represent inductance and a
member function called compute_voltage to define the V-I relationship for Inductors.

Another second level class component is the MutualInductance class, which
represents mutual inductance between two Inductor objects. It includes two mutuality
parameters and a _inductance parameter added to the class type in addition to the parameters
took over from ElectricalComponent. The _inductance represent the inductance value of
the mutual inductance. The initialization of a MutualInductance type parameter includes the
initialization of the _inductance value. In the describe_type function, _inductance and the two mutuality are derived from user-defined input file. There are also member functions created to allow the MultuaInductance to interact with the mathematical algorithm. These functions will be discussed later.

With the exiting component classes, the software can work with the circuit that only has one topology, which means that the voltage and current across the electronic components will always stay in a steady status. When the topology of the circuit is changed, the analysis result for the circuit will also be different. Since most of the electronic circuits are able to change topologies during operation, analysis on multi-topology electronic circuit is expected to be done by the software. In order to realize this design objective, it is essential for the software to work with the electronic components that can change the topologies. As ideal diode and switch are two most common types of components that are include in the circuit flip the topologies, for this project, I created new component classes to include these two electronic components into the analysis system.

After creation of code for new classes, the class component structure hierarchy for electronic components is shown as figure 3-2:

Figure 3-2 Updated hierarchy for ElectricalElement
As can be seen in figure 3-2, a third-level class is developed under 

`ElectricalElement`, called `SwitchableElement`. `SwitchableElement` is a class type that represents all the electronic components that can change the topologies in the circuit. For this project, there are only diodes and switches categorized as `SwitchableElement`. The common electronic properties shared by all the components that can flip the topologies of the circuit are that they have two different statuses: on and off. In order to represent different status of these components, a member parameter called `_state` is developed in `SwitchableElement` class.

There are member functions in the class that can set up and call the `_state` value with respect to the status of the switchable electronic component.

The two different classes developed under `SwitchableElement` are `Switches` and `Diodes`, which represent switches and diodes in the electronic circuit. The diodes in the circuit are considered as ideal diodes, which has simple electronic property that is easy to be analyzed. As both switches and diodes are ideal when they are developed into component classes, there is not any unique feature added to these two components compared to their upper class except that they have an `initialize_px_computation` function to initialize the voltage and current across these components. In the RLCM electronic circuit, switches are controlled by control system circuit. Thus, `_output` and `_fanout` are included as two member parameters for switches to connect switch to control system. The function of `_output` and `_fanout` will be introduced in more details in the section of Control System Components Classes.

The first important milestone for this project is to complete the development of component classes for RLCM electronic circuit in order to allow the software to work with multi-topology circuit. The new component classes developed for the project properly fit into the developed structure hierarchy and successfully identify the components that can change the topologies of the circuit. In the latter section of Electrical Circuit Analysis Process, the design
work that involves the new component classes into the analysis algorithm will be introduced in
details.

3.3 Control Block Classes

The design of control system component classes starts from blank, as the original
software tool can only work with RLCM circuit that does not have a control system. Control
systems are applied to RLCM circuit to control the switchable elements with respect to time, so
that the behavior of the circuit will be changed when the topologies of the circuit are switched.
The design objective for the development of control system components classes is to create new
C++ data types that can represent different fundamental components in the control system, which
allows users to create a simple control system that can be taken care of by the software tool. As
the control system circuit will be integrated with the electronic circuits to control their behavior,
the analysis algorithm for an electronic network that includes both RLCM electronic circuit and
control system should be kept as the same, which is state-space-model analysis method. Control
system blocks that can be treated as state variables should also be included in the design work.
Thus, a simple control system circuit that can meet with the design requirement will include
inputs, outputs, basic operational blocks including gain blocks and sum blocks, and state-variable
block-integrator.

To create systemized component classes for control system blocks required by the design
objective, a similar criterion can be applied as when developing the electronic component classes:

1. All the control system blocks will be represented by a fundamental level new data
type in C++ framework, which is different from the fundamental data type for
   electronic components.
2. Second-level Subclasses will be developed under the fundamental data type to represent a group of control system blocks that have common functionality in the circuit or in the analysis algorithm.

3. Top-level subclasses will be developed to represent each specific control system block, where all the unique properties of the specific control system are included.

Following this criterion, a three-level control system block class hierarchy is developed and shown as the figure 3-3:

![Control System Block Class Hierarchy](image)

**Figure 3-3 Control system block class structure hierarchy of updated software**

Similar to the way to represent all the electronic components, I developed a class type called **ControlSystemBlock** to represent all the control blocks in the control system. This is a fundamental level class, where it has initializer and destructor as the essential member functions to create a new class. It also includes **describe_type** and **describe_instance** member functions, where **describe_type** defines how the control system block created from the user-defined inputfile and **describe_instance** identifies each object of **ControlSystemBlock** type in order to differentiate one from another. Important member parameters like **identifier** is also created, which represent a symbolic name of each control system block defined by users.
There is also a member function that allows object of `ControlSystemBlock` type to call the identifier. All the control system blocks have two types of connector to be connected to other control system blocks, which are inputs and outputs. Users will describe the connection relationship in the user-defined input ism file with a certain format. There are two different class type developed to represent the two different kinds of connectors—`SignalInputPort` and `SignalOutputPort`. The connection relation data of each control system block will be stored in the member parameters of the two connector class types. In `ControlSystemBlock`, there is a virtual function called `get_outputs` that allows the object of `ControlSystemBlock` type to call the connection data stored in output connectors so that it can find out the information of the connected control system block. Some other member functions are created to involve the control system blocks into the analysis algorithm, which will be introduced in details in latter section.

The second level sub class of controls system blocks include six different class types. Four of them can be further broken down to higher level sub class, and two of them are the top-level classes.

The two top-level sub classes are `SumBlock` and `GainBlock`, which represent the sum block and gain block in the control system. The function of sum block in the control system is to add all the values passed from the input control system blocks. It also allows minus operation, if a minor sign is assigned to the input control system block. Sum Block is a special block compared to other control system blocks, as it has multiple control system blocks connected to it as input while most of the other control system block only has one input control system block. Taking account of all the features of sum block, I introduced a new data type called `SignalInputPortList` by making use of an installed C++ template class type-list. `SignalInputPortList` is a list of the reference to different `SignalInputPorts`. Therefore, `SumBlock` has a member parameter of `SignalInputPortList` to store the connection information of all the control system block system. The `describe_type` and initializer member
function of SumBlock are also changed to take care of multiple different input control system block. Gain block can amplify the signal value sent from the input control system block, and pass the amplified value to the control system block connected to its output connector. Thus, the GainBlock class type has a member parameter called _gain, which is a string type in C++. The _gain parameter is a symbolic representation for the gain values of the gain block.

The four second-level sub classes that can be further broken down include InputTypeBlock, OutputTypeBlock, ClockTypeBlock and ControlSystemStateBlock. The main reason to develop these four sub classes is to identify the different roles of different control system blocks in the state-space-model analysis method. InputTypeBlock represents all the input blocks to a control system that provide a constant signal. The Input Blocks to a control system that provide a clock signal that will change along with time are represented by ClockTypeBlock class type. OutputTypeBlock represents all the output blocks of the control system which will provide an output signal that is processed from the original input signal. ControlSystemStateBlock represents all the control system blocks that are set as state variables in the state-space-model. In this project, integrators are categorized as ControlSystemStateBlock. With State-Space-Model Analysis algorithm, the software will generate a matrix equation for control system that consists of matrix list of all the four different types of control system block and a body matrix that represents the operation process through the control system. In order to provide assign all the control system blocks that belong to the four different types to an appropriate location in the matrix list, a member parameter called _serialnumber is created for all the four second-level sub classes. The _serialnumber is a unique integer assigned to each object defined as one of the four second-level sub class type, and a member function created for these four class type allows the object to call the serial number, so they can be located into an appropriate position in the matrix.
**InputBlock** is developed from **InputTypeBlock** to represent the constant-signal input block to the control system. As input blocks are the original resource of the signal to the control system, they only have an output connector to other control system blocks. Thus, in **InputBlock**, the **SignalInputPort** parameter is set as empty. **describe_type** and initializer function is fixed to only take care of the output connection data.

**SignalBlock** is derived from **ClockTypeBlock**, which represents the input block to the control system that has a varying signal with respect to time. The relationship of the signal intensity vs time for the clock signal block developed for this software is a right-angle triangle wave. This is a periodical wave, where the signal will increase proportionally along with time until it comes to the end of a period. At the end of period, it will jump back to 0 and starts again. Signal Block also does not have any control system block connected through its input connect, therefore the class is created in a way similar to **InputBlock**, except that the class name is different.

Under **OutputTypeBlock**, there is a derived sub class called **OutputBlock** to represent the output block of the control system. Output blocks are the end of the control system, and do not have any control systems blocks connected to its output connector. Thus, there is no **SignalOutputPort** type member parameter in **OutputBlock**. The **describe_type** and initializer member function are also fixed to only define the connection data for **SignalInputPort** type member parameter.

**Integrator** developed from **ControlSystemStateBlock** represents the integrator block in the control system, which will integrate the signal value passed into it and output the integrated value to the control system block connecting to its output. Unique member functions are developed for **Integrator** to involve its functionality into state-space-model analysis algorithm, and that will be discussed in details in the section of Control System Analysis Process.
The design of the component class for control system blocks successfully provides the software tool with the ability to conduct analysis on control system circuit and generate a state-space-model matrix. The structure of the class hierarchy clearly identifies each control system block with its unique functions in the control system and state-space-model analysis algorithm. The details of the design of the process to analyze the control system with state-space model will be introduced in the section of Control System Analysis Process.

3.4 Electrical Circuit Analysis Method

For electrical circuit analysis, the software tool takes in a user-defined circuit and operates it through a developed analysis algorithm class to generate a matrix result, as described in Chapter 2:

\[ M_0 \dot{x} = Ax + Bu \]

"\( \dot{x} \)" represents the matrix with all the derivative of state variable values in the electronic circuit. “\( x \)” is the matrix of state variables, and \( u \) is the matrix of input variables. “\( A \)” stands for the coefficient matrix of state variable values, while “\( B \)” stands for the coefficient matrix of input variable values. \( M_0 \) is the body coefficient of the derivative of state variables. The result matrix output by the software is stored in a MATLAB file, where \( M_0, x, A \) and \( B \) matrix are demonstrated in compatible with the matrix format required in MATLAB. From input circuit file to the output matrix in MATLAB, the software goes through a serious analysis steps shown as the following flow diagram:
At the beginning of the analysis process, users are required to define an input file to represent the circuit that they want to conduct analysis. The input file should be .ism and a strict format to define the circuit in the input file must be followed in order to fit into the software framework. The first step to define the circuit is to define all the nodes in a circuit. Figure 3-5 is a simple buck-converter circuit, which does not have a switch. There are four different nodes in this circuit that shared by all the electronic components, naming as a, b, c and gnd. In the ism file, the node will be defined as the following format:

Node  Name;

Users need to first specify the type id of electronic node defined in the software framework, which is Node. Then, users can give a symbolic name to the node, which will be set as an identifier of the electronic node in the software. For figure 3-5, the node will be defined as:

Node a;
Node b;
Node c;
Node gnd;
After defining the node, users are required to describe each electronic component in the following format:

```
Component Type ID  name  node1 ()  node2 ();
```

“Component Type ID” is required for the software to identify the type of the component in the frame work. This id must agree with the type id defined in the software for each type of component, in order to make the definition effective to work with the software frame. “Name” is a symbolic mark that is treated as the identifier for the electronic component, which is defined by users. Node 1 and 2 defines the two nodes on both sides of the component. Inside the parenthesis, the symbolic identifier of the nodes connected to the component will be specified by users, which is previously defined in the Node definition. The order of the node matters, where current direction should always flow from node1 to node2. Taking the voltage source in figure 3-5 as an example, it should be defined like:

```
VoltageSource  V1  node1 (a)  node2 (b);
```

The definition of resistors, capacitors and inductors includes one extra value that users need to specify, which are resistance, capacitance and inductance. In the input file, the three characters are represented by “r”, “c” and “l”. In the parenthesis followed by these three symbols,
numerical numbers will be specified by users, standing for the numerical value of the three characters. For the resistor in figure 3-5, the definition of it in the input file should look like:

Resistor R1 node1 (c) node2 (b) r (0.75);

Following the format introduced above, users can easily define an actual electronic circuit into the ism file, where the information of all the electronic components, nodes and schematic structure are specified. For figure 3-5, a full user-defined input file will be:

Node a;
Node b;
Node c;
Node gnd;

VoltageSource V1 node1(a) node2(gnd);
Resistor R1 node1(a) node2(b) r(1);
Inductor L1 node1(b) node2(c) l(0.5);
Capacitor C1 node1(c) node2(gnd) c(0.25);
Resistor R2 node1(c) node2(gnd) r(0.75);

After users define the input circuit file, the file will be processed into the software. Figure 3-6 demonstrate a flow diagram of the process in the second step:

![Figure 3-6 The signal flow diagram of the information retrieved process](image-url)
Once the software detected an input file is inserted, it will scan through the file to check whether the definition of the circuit is valid or not. If the format is not valid, the software will throw an exception to warn users to check whether the input file is organized in a correct format. Reasons that lead to the failure of format check lie on different aspects, including miss-spelling of component ID, missing important contents that are needed in the definition and connection error. If the format check succeeds, the software will create a new task object to store the information of the input file. Three types of important information are extracted from the input files into the object, including the electrical components in the circuit, mutual inductance generated in between the components and the nodes included in the circuit. For each electrical component defined in the input file, the task object will get its type id, identifier and information of connected node. If inductors and capacitors are included in the circuit, mutual inductance will be calculated for these two types of components and stored in the task object.

When the task object stored all the valid information from the input files, it is ready to be processed into the electronic circuit analysis algorithm in the software. The analysis algorithm of the software is called `ElectricalNetworkStateSpaceModelBuilder`, where it builds a state-space model of the input circuit and performs analysis over it to generate the result matrixes from the model. The state-space model in this software is an object of the data type called `ElectricalNetworkStateSpaceModel`, created in the analysis algorithm class. Inside the `ElectricalNetworkStateSpaceModel` class, there is a member object called `_graph`, which represents a graph model for circuit analysis. To create a new state-space model, a graph that represents the electronic circuit in a mathematical model needs to be created first. Figure 3-7 shows the process of creating the graph in the software:
ElectricalNetworkStateSpaceModelBuilder class takes the Task Object as an input argument. When this analysis algorithm is run by the software, a new and empty state space model of ElectricalNetworkStateSpaceModel data type is created. As described in Section 2.3 about the graph theory, the two important members that construct a graph are nodes and branches. In this project, the graph is defined to take the node in the electrical circuit as the node of the graph. The branch of the graph is defined to be the objects of ElectricalElement. In the analysis algorithm class, two member functions called Extract nodes and Extract branches extract the nodes and electronic components information from the task object. After extracting the nodes and electronic components information out, a member function installed in the state space model called add allows the state space model to insert the nodes and electronic component into graph as the graph nodes and branches. When graph takes in the electronic component as the branch, the identifier and connected node information of the electronic components are also inserted into the graph as part of branch. Connected node information determines the location of the branch in the graph, and identifiers allow the analysis
algorithm to output a symbolic matrix equation with the symbolic name of each electronic component. In addition to the creation of graph, mutual inductance information is also extracted from the task object to store in a member parameter called `multualinductances` in the state space model. After `ElectricalNetworkStateSpaceModelBuilder` operate the three member functions, the state space model with the graph installed is ready to be analyzed.

The class type of state space model includes a member function called `analyze`, in which it has a series step to build up the result matrix equation out of the circuit installed in the graph. The analysis process conducted by this function is shown in figure 3-8:

*Figure 3-8 The analysis function for the Original software*
The State Space Model, whose graph has been installed, calls the member function `analyze`. The `analyze` function in `ElectricalNetworkStateSpaceMordel` will then run through seven steps to generate an output MATLAB file of the expected matrix equations. The first step is to classify the elements, where the function creates a tree structure and identifies the branches and links in the tree. The tree structure is formed as a member object called `Tree`, which is created from the installed `_graph`. The software will then go through the `Tree` to identify all the branch elements and link elements with respect to their data type and store the branches and links of the same data type into different sets object.

After the “classify elements” step, it will start to extract all the f matrixes from the branch sets and link sets identified in the previous function. In the f matrices, there are 13 different f matrix elements that are defined by the combination of a branch set and link set, which is determined by the topology matrix and loop matrix described in Chapter 2. After the function extracts the f matrices, it starts to build the c matrix, which stands for the matrix of capacitance. As capacitors are both pushed into the branch sets and link sets, the `build_c_matrix` function extract the capacitors from both sets and build them into a tree structure and a cotree structure. There is an `if` statement to identify if there are only branch capacitors or only link capacitors or both. Correspondingly, different c matrix will be generated out of the capacitors tree and cotree structures. Next, the function builds up the l matrix with the member function `build_l_matrix`, where l matrix stands for the matrix of inductance. This build process is quite similar to that of `build a c matrix`, where an inductance tree and cotree structures are defined from the inductors branch sets and inductors link sets. After the evaluation of where the inductors come from, a l matrix is created from the inductance tree and inductance cotree. After both c matrix and l matrix are built in the state space model, it starts to build the r and g matrices, which stand for resistance matrix and conductance matrix. The building process is still quite similar to l and c matrix building process, where resistance tree and cotree are created from
Resistance branches and links. Resistance matrix is defined from the tree and cotree with respect to where the resistance comes from. The conductance matrix is defined by taking reciprocal of the elements in the resistance matrix.

Once all the essential matrixes are created, the last step is conducted with the member function called `build_state_matrices`. During this step, the expected result body matrix of the state variables and input variables are generated by applying state space analysis method on all the matrixes generated above. At the end of the `analyze` function, an output MATLAB file is generated with the state space matrix results.

As introduced in Section 3.2, electronic component classes for switchable components, including switch and diode, are created. In order to involve them into the analysis process, the original analysis algorithm is redesigned to output all the state space matrix results for different potential topologies, and demonstrate the next topology that the current circuit can switch to. In addition, as the output of C and D matrix is also an important part of the project, new member functions are included to generate the matrix for output equation as described in Chapter 2.

After redesign of the analysis algorithm with respect to the fields mentioned above, the over flow of the analysis process remains similar to the original algorithm as shown in figure 3-4, but two steps out of the whole process are changed to meet with all the requirements for the design goal of this project. The two steps are “set up graph” and “Analyze graph model”. In “set up graph”, the software add one more process to identify the switchable elements, like switches and diodes. The updated “Analyze graph model” successfully integrates the analysis method for multiple different potential topologies, defined by the number of switchable elements in the circuit, and includes the new member functions to build up C and D matrix.

The updated flow diagram of “set up graph” is shown in Figure 3-9:
As can be seen from the updated flow diagram, the difference occurs when the analysis algorithm extract the branches which are the electronic element in the circuit. A if statement is added to ask if the electronic element extracted out is a switchable element or not. If it is a switchable element, then it will be stored in a separate list called _Switchable_Elements, which is defined as a member parameter for algorithm class ElectricalNetworkStateSpaceModel. On the other hand, if the electronic element is not a switchable element, it will be directly pushed into the graph as a branch.

After all the graph is created without any switchable elements pushed in, it comes to the step of the redesigned “Analyze graph model”. The redesigned “Analyze graph model” is a multi-
optional analysis method in `ElectricalNetworkStateSpaceModel`, where there will be two different situations, as shown in Figure 3-10.

![Diagram showing flow of analysis processes](image)

**Figure 3-10 Updated analysis for electronic circuit with switchable component**

When there is no switchable element stored in `_Switchable_Elements`, the graph is completely stored, and the graph will be analyzed with method 1. Method 1 is similar to the “Analyze graph model” process, except that member functions to generate C and D matrix is included. The optimized “Analyze graph model” flow graph is shown in Figure 3-11.
It is clear to see from the flow graph that the first six steps of the analysis method 1 are completely the same as the original analysis method. After the software successfully identify all the state variables and input variables, it builds up the A and B matrix that defines the combination of state variables and input variables to form the state equation as introduced in Chapter 2. The added parts are four member functions included in \textit{ElectricalNetworkStateSpaceModel} algorithm class, which allowed the creating of C and D matrix. As the mathematical derivation is specified in Section 2.3, each of the four added function takes ¼ part of the C and D matrix to set up, following the matrix result in Chapter 2. In each function, it follows a row by row checking logic to identify whether there are expected output variables, state variables and input variables. Sometimes the circuit will not have all of these variables, for example: if the circuit only contains a capacitor, the inductor tree or co-tree
will be empty, then the second column of C matrix will not exist. Thus, according to the structure of the circuit, C and D matrix can be very different. The “CV” matrices includes the part of C and D matrix that defines all the voltage values in tree and co-tree structure, which are the equations between row 1 to row 6 in C matrices, and the “CI” matrices defines all the current values in the tree and co-tree structure. Similar for “DV” and “DI”, they specify the voltage and current in Dmatrices separately.

Another potential analysis method is conducted when there are switchable elements in the circuit, which means the graph is not complete. Since every switchable element has two possible different status, with n number of switchable elements in the circuit, there will be $2^n$ possible different combinations of these switchable elements. For each topology, every switchable element will be pushed in with one specific status. Once the graph is full filled, it will go through analysis method 1 for each topology. A flow process diagram is shown below:
The first step is to identify the number of switchable elements $s$ in the circuit, by calling the size of `Switchable_Elements`. Once $s$ is determined, the number of potential topologies $n$ will automatically equals $2^s$. In order to easily identify different topologies, a new structure called `Topology` is defined for the `ElectricalNetworkStateSpaceModel` analysis algorithm, which include an `int` type member parameter `index` and a `vector<bool>` type member parameter `_structure`. The status of the switchable element is represented by a `bool` value.

When the switchable element is on, a `bool` value equals `true` is pushed into the `Topology` data type; when the switchable element is off, a `bool` value equal `false` is pushed into `Topology` data type. In the analysis method 2, a list of `Topology` data type called `_topology_list` is defined with the size of $n$, which is the number of potential topologies. Setting up a `for` loop in
the function to run through all possible combinations of different switchable components with
different status, and define a Topology object and pushed the object into _topology_list.

For each object of Topology type, all switchable components with a specific status will be
stored as a vector of bool values in the member parameter _structure. For instance, if we have
2 different switchable elements s1 and s2, they will have four different possible combinations:

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>off</td>
<td>off</td>
</tr>
</tbody>
</table>

For each topology, the status of s1 and s2 is pushed into the topology as bool values
according to its on/off status. Once the vector<bool> is defined, the index of each topology is
defined by making use of binary number. Taking the s1 and s2 case as the example, for the four
different topologies, there are four different combinations of statues. If on status is represented by
integer 1 and off status is represented by integer 0, the topology _structure can be represented
as a binary number:

11;
01;
10;
00;

Converting the binary number to the normal decimal number, it will become:

3;
1;
2;
0;

These numbers are the index of each topology. When pushing the Topology object into
the _topology_list, they will be pushed into the location with respect to their _index value.
Once the \texttt{topology_list} is set up, the analysis method will go through each \texttt{Topology} in the \texttt{list} as shown in \texttt{for} loop 1, and check each element of the \texttt{structure} to pull out the status of each switchable element in the specific topology as shown in \texttt{for} loop 2. After that, it pushes the switchable element of a specific status into the graph. The design for pushing the switchable elements with different status into the graph creatively makes use of replacement of short and open circuit with 0-V voltage source and 0-A current source. In electronic circuit analysis, it is common to replace deactivated voltage source with a short circuit, and a deactivated current source with an open circuit. In this project, this replacement is performed reversely, as class components for voltage and current source are already installed in the software and they can be successfully integrated with the graph structure in the code. Therefore, when the switchable element is on, it will be pushed in as a 0-V voltage source; when the switchable element is off, it will be pushed in as a 0-A current source.

After all the switchable elements are pushed into the graph, the graph structure is completely set up. Then it will run through the same process as analysis method 1, to generate A, B, C and D matrix in the output file. As there are multiple different potential topologies, multiple combination of state space model result equation will be demonstrated in the output file.

\section*{3.5 Control System Analysis Method}

In the original software, the control system analysis algorithm is not included. However, in order to apply the software for analysis on a full RLCM electronic circuit controlled by a control system, the new analysis algorithm class is essential. The main purpose of control system analysis process is to generate a state space model output matrix for the control system, which can be utilized to set up switching surface matrix. As described in Section 2.1, switching surface
is an important part of the output of this project to define the conditions when the electronic circuit switches its topology.

The new analysis algorithm class defined for control system is called **ControlSystemStateSpaceModel**. With this new analysis algorithm class, the software can successfully take in an input file defined by users for a control system, and provide back with a state space model matrix for electronic circuit to define the switching surface matrix. A similar flow progress of analysis to the electronic circuit analysis process is performed on the control system analysis, shown in figure 3-13:

![Software Tool Diagram](image)

**Figure 3-13 General flow diagram of control system analysis algorithm**

At the beginning, users are required to define a ism file with specified format to describe the control system that they want to analyze. This format is quite different from the format for the electronic circuit definition, as all the control system blocks are connected through **SignalInputPort** and **SignalOutputPort** as described in Section 3.2. In the user defined
file, users need to describe all the blocks included in the control system, and define how they are connected. To describe the block, users will always start to specify the type of the control system blocks same as the type name defined for different blocks. Following that, they need to give an identification for the block. For example, to define an input block, the input file will look like:

```
InputBlock I1;
```

“InputBlock” specifies the block type, and “I1” serves as an identifier for this input block. To describe the connection relationship for control system block, a single direction method is applied. As for most of the blocks, they will only have one control system block connected to its input port, but they are able to output to multiple different blocks. Thus, it is easier to define the blocks connected to the input port of the blocks. Input block is a special type of block, as it serves as the signal source in the control system, there will be no control system block connected to its input port. Thus, the definition of input block will not include the part to specify which it is connected to. For other blocks, the definition in the ism file will look like:

```
ControlSystemBlockType Identifier in (Identifier2);
```

Inside the parenthesis after “in”, users put the identifier of the block that is connected to the defined block. For instance, if an input block “I1” is connected to an output block “O1”, the definition of the output block will look like:

```
OutputBlock O1 in(I1);
```

Sum block is another special block that needs to be defined uniquely, as a sum block will connect to multiple different blocks from the input port and a sign will be specified to define
whether it performs addition or subtraction. For example, if a sum block is connected to two input blocks, “I1” and “I2”, the definition in the ism file should look like:

**Addition:**

```
SumBlock S1 in_1(I1) in_2(I2);
```

**Subtraction:**

```
SumBlock S1 in_1(I1) in_2(I2) in_2.sign(-1);
```

A sum block can have over 2 control system blocks connected to its input port. Following this definition format, users can connect as many blocks as they need to sum block and specify the sign for each of them.

In order to give a clear explanation of the format for users to define the control system, a sample user-defined input file is generated for the control system below:

![Sample control system case](image)

The definition in the user-defined file will look like:

```plaintext
InputBlock I1;
InputBlock I2;
SumBlock S1 in_1(I1) in_2(I2) in_2.sign(-1);
GainBlock G1 in(S1);
GainBlock G2 in(S1);
IntegratorBlock In1 in(G1);
SumBlock S2 in_1(In1) in_2(G2);
```
After users define the input file, the input file will be taken in by the software. The next step is to create a new model object to store the information of the input file, so that the software can make use of it in the new step. The flow diagram for storing the input information is shown as figure 3-15:

![Flow Diagram](image)

**Figure 3-15 The information retrieving process for control system**

When the software takes in the user-defined input file, it will scan through to check if the format meets with all the criteria. Misspelling of control system block type or invalid definition of connection relationship will cause an exception warning for users to require them to redefine the control system. If the input file meets with all the requirements, a task object will be created by the software which includes all the information scanned from the input file. In the task object, there will be different `ControlSystemBlock` type objects to represent all the control system blocks in the input files, with the connection stored in its member parameter `SignalInputPort` and `SignalOutputPort`. 
The next step after storing the input file information in the software is to set up the graph for control system analysis algorithm. The graph type for control system is defined differently from the graph type for electronic circuit. In electronic circuit, the nodes in the electronic circuit consist of the graph nodes and the electronic element in between the electronic nodes consists of the branch for the graph. For Control system, there is no electronic node. Instead, a class type called `ControlSystemNode` is created to help definition of the graph nodes for control system. In this step, a full graph is not created. Instead, a partial graph with only the nodes pushed into the graph. The branch of the graph for control system is defined in the next step. The flow process of definition of the partial graph is shown in figure 3-16:

![Figure 3-16 The process to set up graph in the software](image)

As shown from the flow diagram, in `ElectricalNetworkStateSpaceModelBuilder`, a control system analysis algorithm model `ControlSystemStateSpaceModel` is created from the task object with all the input file information stored in. After that, the model will extract out all the control system blocks out from the task object. There is a member function called `add` in `ControlSystemStateSpaceModel`, which takes a `ControlSystemBlock` type object as an
input argument to be pushed into the graph. In `add` function, for each control system block, there will be a `ControlSystemNode` type object created accompany with it. The `ControlSystemNode` type object defines the type of the node for different types of the control system block. Generally, there are three special types of `ControlSystemNode`, which are “root”, “fork” and “leaf”. Identification of different control system blocks in terms of different types of nodes help to develop the analysis algorithm on the control system graph to generate the state space model matrix. Thus, in the `add` function, the control system block is pushed in with the `ControlSystemNode` created from it as a pair to define the node of the graph for control system analysis. After all the control system blocks are added into the graph, the partially defined graph is ready to be processed through the analysis function in `ControlSystemStateSpaceModel`.

The flow process of the analysis performed by the `analyze` member function in `ControlSystemStateSpaceModel` is demonstrated in figure 3-17:
Figure 3-17 The analysis function for the control system

With the partially defined graph from the previous step, the `ControlSystemStateSpaceModel` calls its analysis function `analyze` to finish setting up the full graph and generate a state space model matrix for the control system. The first step in the analysis function is to build the edges for the graph in order to completely set the graph up. A new class type called `ControlSystemEdge` is created to be pushed in as the edge of the graph. Unlike the branches in the graph for electronic circuits, the edge of the graph for the control system does not carry any physical value with it, but only defines the connection relationship between the nodes in the graph, which in this case are the different control system blocks.

Direction of the edge is also specified to define whether the blocks are connected with each other through input port or output port. If the block is connected to another block through input port,
the direction will be specified as 1; if the block is connected to another block through output port, the direction will be specified as -1.

After the build edge step, the graph is fully constructed. The next step is to classify the nodes in the graph. As mentioned earlier, the graph node is made up of a pair of a control system block and a `ControlSystemNode` type object. In the classify node step, the function will go through all the nodes in the graph and check the type of the control system block for each node: if it is a output block, the paired `ControlSystemNode` will be set as a “root” and the graph node will be pushed into a member parameter called `_roots`; if it is a input block, the paired `ControlSystemNode` will be set as a “leaf”; if it is a sum block, the paired `ControlSystemNode` will be set as a “fork”; if it is other type of block, the `ControlSystemNode` will remain with its default value. Integrator block is a special type of control system block, as it is the state variable in the control system. If an integrator block is identified in the classify step, while the paired `ControlSystemNode` will remain default, the graph node will be stored in a member parameter called `_state_variable` that will be used in the future step. The important purpose to classify nodes in the graph is to help identify the different type of control system blocks in order to perform different calculation method for them to generate the state space model matrix.

The next two steps are constructing the input matrix and constructing the output matrix, which defines the input variable matrix, output variable matrix and state variable matrix. The input variables \( u \) in the control system are the input blocks, and the output variables \( y \) in the control system are the output blocks. \( x \) for control system are the integrator blocks. The state space model matrix result should include two equations:

\[
\dot{x} = Ax + Bu \\
y = Cx + Du
\]
Since the control system is usually not too complex, the two equations for control system is output together as one big equation:

\[
\begin{bmatrix}
\dot{x} \\
y
\end{bmatrix} = E x + Fu
\]  
\[E = A + C\]  
\[F = C + D\]

The constructing input matrix list out all the identifiers of input blocks to demonstrate input variables \(u\). The constructing output matrix list out all the identifiers for output blocks and integrator blocks to demonstrate the combined output matrix:

\[
\begin{bmatrix}
\dot{x} \\
y
\end{bmatrix}
\]

The last step in the analysis process is to construct the E and F matrix for the control system. Before explanation of how this step works, it is important to understand how to generate a state space model matrix for control system manually. The analysis process will be introduced by using the control system in figure 3-14 as an example.

![Figure 3-14 Sample control system case](image)

To start the analysis process, it is important to find out what are the input variables, output variables and state variables. For this example, the input variables are I1 and I2, the output
variable is $O_1$, and the state variable is $I_{n1}$. Then the next step is to go through all the output variables and find out an equation that expresses each output variable with a combination of input variables and state variables. Since signal flow process for a control system without a loop is always in one direction, the relationship between the output blocks and the combination of input blocks and integral blocks could be easily tracked step by step backward from the output block. Taking this control system block as an example, the output block $O_1$ is the output of the sum block $S_2$, thus an equation can be easily set up as:

$$O_1 = S_2.\text{output} = I_{n1} + G_2.\text{output} \quad (3-4)$$

Since $I_{n1}$ is the state variable, so it is expected to be kept. The gain block $G_2$ is neither a state variable nor an input variable. Thus, it needs to be broken down further. Since $G_2$ is connected to the output port of the sum block $S_1$, the input of $G_2$ is equal to the output of $S_1$, where we have:

$$G_2.\text{input} = S_1.\text{output} = I_1 - I_2 \quad (3-5)$$

$$G_2.\text{output} = G_2.\text{input} \times \text{gain2} \quad (3-6)$$

Thus plug the two equations back to the equation $3-4$, the relationship between the output block and the combination of input variables and state variables is defined as:

$$O_1 = I_{n1} + (I_1 - I_2) \times \text{gain2} = I_{n1} + \text{gain2} \times I_1 - \text{gain2} \times I_2 \quad (3-7)$$

A similar process can also be performed to set up the state equation for each state variable. In this case:

$$I_{n1} = (I_1 - I_2) \times \text{gain1} = \text{gain1} \times I_1 - \text{gain1} \times I_2 \quad (3-8)$$

Combining equation $3-7$ and $3-8$ together, the state space model matrix result is demonstrated as following:

$$\begin{bmatrix} I_{n1} \\ O_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} I_{n1} + \begin{bmatrix} \text{gain2} & -\text{gain2} \\ \text{gain1} & -\text{gain1} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \quad (3-9)$$
\[ E = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \]  \quad (3-10)

\[ F = \begin{bmatrix} \text{gain2} & -\text{gain2} \\ \text{gain1} & -\text{gain1} \end{bmatrix} \]  \quad (3-11)

Compared to the generation of state space model matrix for electronic circuit, the process to generate state space model matrix for control system is more concise and simplified. The last step, constructing body matrix follows the process above to generate E and F matrix. The member function `construct_body_matrix` performs this step, by taking the list of graph nodes, `_roots` and the whole graph as input arguments. The graph nodes in the `_roots` list, as specified in classify node step, contains all the output blocks. The function will start to track back from each `_roots` node to ask for a string output that represents the symbolic format of the signal output from the connected previous node. If the graph node is not a “leaf” node or a state variable, it will ask for a string output from the previous block that connects to it. If the graph node is a “fork”, which means sum block is contained in the node, it will ask for all the string output from the nodes that it is connected to, and add all the string output from the previous nodes up as the output of the “fork” node. This process will be continued until a “leaf” node or a state variable is found. After going through the `_roots` graph node list, the function will go through the entire state variable node in `_state_variables`. Similar process will be performed until it reaches a “leaf” or a state variable. After these two processes, the function can generate two vectors of strings to represent E and F matrix.

### 3.6 Combined Analysis on RLCM Electronic Circuit Controlled by a Control System

In Section 3.4 and 3.5, a detailed introduction is given on the separate analysis process for electronic circuit and control system. The ultimate goal of this project is to make the software
eligible to perform an analysis on one simple comprehensive electronic network test case, as described in Section 2.2. Thus, the analysis process is optimized to allow software to conduct analysis for both of the network simultaneously, and generate the switching surface with respect to the connection between them. A list of specific parts that are optimized for the analysis process is shown as following:

1. Control system component class should include comparator in the control system, which is connected to the electronic circuit directly.
2. The user-defined netlist should include information of both the electronic circuit and control system, while it also defines the connection between the two systems.
3. Electronic circuit analysis process should add a process to identify the information of the control system connected to it.
4. Electronic circuit analysis process should include the process to generate switching surface.

In RLCM circuit, a control system is connected to an electronic circuit through a comparator block, which selects one of the two signals connected to it that goes higher than the other. The comparator is connected to the switch in an electronic circuit to control its status. This is significant to generate a complete switching surface for the combined network, as the signal selected by the comparator will determine $\sigma_{dm}$ and $\sigma_{cm}$ in the switching surface matrix as described in Section 2.5.

Defining comparator block in the software is similar to define a sum block, where both of them will have multiple input ports that can connect to more than one control system block. However, the output of a comparator block is switchable, which will be the selected signal connected to it. Therefore, a _status parameter is included in the Comparator to identify which signal is selected as the output.
To meet with requirement 2 in the list, a `SignalOutputPort` is added to the `Switch` class, which allows the users to define the connection between the switch in the electronic circuit to a control system block. In this case, the control system block will be the comparator block. Definition of the electronic circuit and control system to the netlist in the input file is the same as described in the previous sections, except the definition of the comparator block, which looks like:

```
ComparatorBlock Cp1 in_1(Signal 1) in_2(Signal 2) in_3(Switch);
```

The difference occurs at the last part of the definition, where “in_3” defines the switch that the comparator is connected to. When the input file that contains the information of the combined network is pushed into the software, while the software is still able to identify the component information of both systems, it is also eligible to find out how the two system is connected together.

Another member function called `get_output` is included in the `Switch` which can be called to find out the information of the comparator connected to it. The comparator will provide the state space model matrix result of the control system back to the switch. With respect to the state space matrix result from the control system and the status of the switch, $\sigma_{dm}$ and $\sigma_{cm}$ can be determined. However, in this project, as the control system in the ultimate test case only includes a comparator connected to a triangular clock signal block and a constant input block, instead of providing back a state matrix result, the comparator will directly provide, the comparator only provides a `bool` value called `Clock_Constant` to identify that the comparator is connected to signal block and a input block. The switch will be turned on when the clock signal block becomes higher than the constant input block, and will be turned off when the constant input block is higher than the clock signal. The design here is simplified to the final test case only, and it provides two different possible results for $\sigma_{dm}$ and $\sigma_{cm}$ when switch is on and off.

For the last requirement, a Change is made to the last step of the analysis for the electronic circuit, when the software is generating the output file. This step is accomplished by a
member function in `ElectricalNetworkStateSpaceModel`, which originally only generates vectors for state variables, input variables and output variables, as well as A, B, C and D matrix.

In the new member function, another function called `switch_surface` is created to generate the switching surface for the combined circuit. As described above, $\sigma_{dm}$ and $\sigma_{cm}$ is already developed by tracing back the information of the control system connected to the switch. In this project, except the switch controlled by the control system, another component that can determines the topology of the network is the diode. `switch_sufact` catches the specific rows of C and D matrix that represent the output equations of the voltage or current value of the diode to define $\sigma_{xm}$ and $\sigma_{um}$ in the switching surface matrix, which specifies the affection of the state variables and input variables in the electronic circuit to the diode.
Chapter 4
Test Cases

The design of the software tool follows an easy to complex path to improve the compatibility of the tool and different electronic network network. At each stage, appropriate number of careful designed cases has been applied to evaluate the functionality of the software tool. These cases cover all the variations and different situations at each stage to assure the reliability of the function of the software tool. In this chapter, there will be demonstration of typical test cases to evaluate different functionality of the software. All the demonstration of the test result is checked against the result from manual analysis to assure the accuracy of the performance. Test cases are generated separately for electrical circuit analysis and control system analysis, to show that the software could successfully perform analysis on both systems. At the end of the chapter, the output result generated by the software will be demonstrated to prove the accomplishment of the goal of this project.

4.1 Electrical Circuit

There are two different test cases shown in this section. One is a buck converter circuit that is formed without switchable elements, the other is the one include the switchable elements. The first test case evaluates the performance of the original software, which does not include switchable elements. The second test case evaluates the validity of the updated software, which is expanded to include the switchable electronic component classes and optimized analysis algorithm. As they are electronic circuit only test cases, switching surface matrix is not included in the output file.

Test Case 4.1-1
Electronic Circuit for Test Case 4.1-1:

![Electronic Circuit Diagram](image)

Figure 4-1 Test case 4.1-1 for electronic circuit

Output MATLAB File for Test Case 4.1-1:
As can be seen from the output file, the state variable matrix is demonstrated as \( x \). A and B matrix corresponding to the state equation is also performed. The symbolic result in the matrix is still not simplified. However, after simplification, it can be seen that the result meet with the result derived manually. C and D matrix is not demonstrated in the output file, as the C and D matrix derived in this project is only for the targeted test circuit. The C and D construction function in the software currently is unable to form a general format of C and D matrix for all different electronic circuits.
Test Case 4.1-2

Electronic Circuit for Test Case 4.1-2:

![Electronic Circuit Diagram](image)

Figure 4-2 Test case 4.1-2 for electronic circuit

Output MATLAB File for Test Case 4.1-2:
%%% Test Case 2
syms C1 L1 R1 V1 real

M = 4;

A_all = cell(M, 1);
B_all = cell(M, 1);

%%% Submodel 1
%
\{ 
  x = ... 
  \[
    v_{C1} 
  \] 
%

M0 = ...
[ 
  C1 
]

A0 = ...
[ 
  \(-1/R1) 
]

B0 = ...
[ 
  0 \(-1) \(-1) 
]

M0inv = simple(inv(M0))
A_all{1} = M0inv*A0
B_all{1} = M0inv*B0
Switch to:
Topology1
Topology2
%% Submodel 2

{% 
\text{x} = ... 
[ 
\text{v}_C1 \\
\text{i}_L1 
] 
%}

\text{M}_0 = ... 
[ 
\text{C}_1 \ 0 \\
0 \ \text{L}_1 
] 

\text{A}_0 = ... 0 
] 

\text{M}_0^{-1} = \text{simple}(\text{inv}(\text{M}_0)) 
\text{A}_{\text{all} \{2\}} = \text{M}_0^{-1}\text{A}_0 
\text{B}_{\text{all} \{2\}} = \text{M}_0^{-1}\text{B}_0 

\text{Switch to:} 
\text{Topology0} 
\text{Topology3}
%% Submodel 3
%
{x = ... 
[ 
v_C1 
i_L1 
] 
%
}

M0 = ...
[ 
C1 0
0 L1
]

A0 = ...
[ 
-(1/R1) (-1)
-1 0
]

B0 = ...
[ 
0 0 0
-1 1 0
]

M0inv = simple(inv(M0))
A_all{3} = M0inv*A0
B_all{3} = M0inv*B0
Switch to:
Topology3
Topology0
From the output file it can be seen that there are four different State Space Model Matrix result for four different topologies. When Switch and Diodes are at different on/off status, they will affect the state space model matrix result. At the bottom for each topology, there is a list of topologies that the current topology can switch to by converting the status of one of the switchable elements.
4.2 Control System Diagram

The test cases for analysis performance on control system diagram follow a easy to complex path, to prove that all the control system block component class type is developed with correct functionality. In addition, they also evaluate the accuracy and effectiveness of the new analysis algorithm for control system. As the state space model is not part of the output file that is required by users. The demonstration of the state space model equation for control system occurs on the C++ command window, and a screen shot of the corresponding command window for each test case is attached to show the analysis result.

Test Case 4.2-1

Test case 4.2-1 is a control system that only includes an input block and an output block. This test case is going to evaluate whether input block and output block component class perform appropriately.

Control System Diagram for Test Case 4.2-1:

![Control System Diagram](image)

Figure 4- 3 Test case 4.2-1 for control system

Output Command Window Screen Shot for Test Case 4.2-1:
As can be seen from the command window, correct input and output matrix is demonstrated. The 
F0 matrix shows that O1 = 1*I1, which is correct for the analysis result of this control system.

*Test Case 4.2-2*

The test case 4.2-2 is a control system block that includes an input block, a gain block 
and an output block. With the test result from test case 1, it is clear that input block and output 
block works appropriately with the software analysis algorithm and output an accurate state space 
output model result. Test case 4.2-2 is created to evaluate the functionality of gain block, and the 
accuracy of analysis result when gain block is pushed in.

Control System Diagram for Test Case 4.2-2:

![Control System Diagram](image)

*Figure 4-4 Test case 4.2-2 for control system*

Output Command Window Screen Shot for Test Case 4.2-2:

```plaintext
The input matrix is:
[  
 I1
 ]
The output matrix is:
[  
 O1
 ]
F0   = 
[  
 1
 ]
E0   = 
[  
 ]
```
As can be seen from the command window, the $F$ matrix is $[1 \cdot G1]$, which matches the result derived manually. Thus, it proves that the Gain Block class type is correctly developed for the software, and the analysis algorithm can accurately provide the correct result when gain blocks are included in the control system.

**Test Case 4.2-3**

In test case 4.2-3, there is a control system that includes input blocks, output blocks, gain blocks and a sum block. From the previous test cases, it is proved that input blocks, output blocks and gain blocks worked with the software appropriately. Test case 3 is designed to evaluate the functionality of sum block, and the accuracy of the analysis algorithm when sum block is included in the control system. There are two parts for test case 3. The first one evaluates the functionality of sum block performing addition, and the second one evaluates the functionality of sum block when performing subtraction. The two different parts assure the sign of the sum block can be appropriately taken care of by the software.

**Control System Diagram for Test Case 4.2-3:**

Part 1
Figure 4-5 Test case 4.2-3(part1) for control system

Part 2

Figure 4-6 Test case 4.2-3(part2) for control system

Output Command Window Screen Shot for Test Case 4.2-3:

Part 1
The input matrix is:
\[
\begin{bmatrix}
I1 \\
I2 \\
\end{bmatrix}
\]
The output matrix is:
\[
\begin{bmatrix}
O1 \\
\end{bmatrix}
\]
F0 =
\[
\begin{bmatrix}
1*G1 & -1*G1 \\
\end{bmatrix}
\]
E0 =
\[
\begin{bmatrix}
\end{bmatrix}
\]

Part 2

The input matrix is:
\[
\begin{bmatrix}
I1 \\
I2 \\
\end{bmatrix}
\]
The output matrix is:
\[
\begin{bmatrix}
O1 \\
\end{bmatrix}
\]
F0 =
\[
\begin{bmatrix}
1*G1 & 1*G1 \\
\end{bmatrix}
\]
E0 =
\[
\begin{bmatrix}
\end{bmatrix}
\]

In the command window for Part 1:

\[
F0 = \begin{bmatrix} 1 * G1 & -1 * G1 \end{bmatrix};
\]

In the command window for Part 2:

\[
F0 = \begin{bmatrix} 1 * G1 & 1 * G1 \end{bmatrix};
\]

This matrix results meet with the result derived manually for this control system. This proves the appropriateness of the development of sum block component class in the software. It
also shows that the analysis algorithm provide accurate result for control system including sumblock.

*Test 4.2-4*

In test case 4.2-4, a complete control system include input blocks, output blocks, sum blocks, integrator blocks and gain blocks. Test case 4 is designed to evaluate the functionality of the integrator blocks and whether the analysis algorithm can generate appropriate E matrix for state variables.

Control System Diagram for Test Case 4.2-4:

![Control System Diagram](image)

*Figure 4-7 Test case 4.2-4 for control system*

Output Command Window Screen Shot for Test Case 4.2-4:
As can be seen in the command window, the output matrix include the integrator block identifier as part of the output matrix, which is the same as specified in Section 3.4. The F0 and E0 matrix demonstrated in the command window as:

\[
F0 = \begin{bmatrix}
1*G1 & 1*G1 \\
0 & 0 \\
1 & 1
\end{bmatrix};
\]

\[
E0 = \begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix};
\]

Both E0 and F0 matrix meet with the manually analysis result. This test case proves the appropriate development of integrator block in the class and the effective functionality of the analysis algorithm to generate both E and F matrix for state space model.

### 4.3 Combined System with Electrical Circuit and Control System
A successful generation of an accurate result for this test case is the ultimate goal for this project as described in Section 2.2. A manually derived result is demonstrated in Section 2.2, which can be used to compare with the MATLAB result generated as following.

Electrical Circuit for Test Case 4.3-1:

![Electrical Circuit for Test Case 4.3-1](image)

*Figure 4-8 Test case 4.3-1 for a combined electronic network*

Output MATLAB File for Test Case 4.3-1:
syms C1 L1 R1 V1 real

M = 4;

A_all = cell(M, 1);
B_all = cell(M, 1);

%% Submodel 1

{% 
x = ... 
[ v_C1 ]
}%

{% 
u = ... 
[ v_V1 i_ID1 i_IS1 ]
}%

{% 
yv = ... 
[ v_V1 v_C1 v_L1 v_ID1 v_IS1 ]
}%

{% 
yi = ... 
[ i_V1 i_L1 i_ID1 i_IS1 ]
}%
\[ M_0 = \ldots \]
\[ \begin{bmatrix} C_1 \end{bmatrix} \]
\[ A_0 = \ldots \]
\[ \begin{bmatrix} -(1/R1) \end{bmatrix} \]
\[ B_0 = \ldots \]
\[ \begin{bmatrix} 0 & (--1) & (--1) \end{bmatrix} \]
\[ C_{0,V} = \ldots \]
\[ \begin{bmatrix} 0 & 1 & 0 & -1 & -1 \end{bmatrix} \]
\[ C_{0,I} = \ldots \]
\[ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]
\[ D_{0,V} = \ldots \]
\[ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]
\[ D_{0,I} = \ldots \]
\[ \begin{bmatrix} 0 & 0 & (-1) & 0 \end{bmatrix} \]
M0inv = simple(inv(M0))
A_all{1} = M0inv*A0
B_all{1} = M0inv*B0
Switch to:
Topology1
Topology2
si_x {1} = ...
  [ 0 ]

si_u {1} = ...
  [ 0 ]

si_d {1} = -1
si_c {1} = 1
%% Submodel 2

{% 
  x = ... 
  [ 
    v_C1 
    i_L1 
  ] 
%
}

{% 
  u = ... 
  [ 
    v_VD1 
    v_V1 
    i_IS1 
  ] 
%
}

{% 
  yv = ... 
  [ 
    v_VD1 
    v_V1 
    v_C1 
    v_IS1 
    v_L1 
  ] 
%
}

{% 
  yi = ... 
  [ 
    i_VD1 
    i_V1 
    i_IS1 
    i_L1 
  ] 
%
}

M0 = ... 
[ 
  C1  0 
  0   L1 
]
\[ A_0 = \ldots \\
\begin{bmatrix}
-(1/R1) & \cdot & \cdot \\
-1 & 0 & 0 \\
\end{bmatrix} \]

\[ B_0 = \ldots \\
\begin{bmatrix}
0 & 0 & 0 \\
-1 & 0 & 0 \\
\end{bmatrix} \]

\[ C_{0 \_V} = \ldots \\
\begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & -1 \\
\end{bmatrix} \]

\[ C_{0 \_I} = \ldots \\
\begin{bmatrix}
0 & \cdot & \cdot \\
0 & \cdot & \cdot \\
0 & 1 & \cdot \\
\end{bmatrix} \]

\[ D_{0 \_V} = \ldots \\
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix} \]

\[ D_{0 \_I} = \ldots \\
\begin{bmatrix}
0 & 0 & \cdot \\
0 & 0 & \cdot \\
0 & 1 & \cdot \\
0 & 0 & 0 \\
\end{bmatrix} \]
\[ M_{0inv} = \text{simple}(\text{inv}(M_0)) \]
\[ A_{all}(2) = M_{0inv} A_0 \]
\[ B_{all}(2) = M_{0inv} B_0 \]

Switch to:
Topology 0
Topology 3

\[
\begin{bmatrix}
s_{i_x}(2) &=& \ldots \\
0 & (--) & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
s_{i_u}(2) &=& \ldots \\
0 \\
\end{bmatrix}
\]

\[ s_{i_d}(2) = -1 \]
\[ s_{i_c}(2) = 1 \]
%% Submodel 3

{% 
x = ...
[ v_C1
  i_L1
] 
%
%
%
%
%
%
%

{% 
u = ...
[ v_VS1
  v_V1
  i_ID1
] 
%
%
%
%
%
%

{% 
yv = ...
[ v_VS1
  v_V1
  v_C1
  v_ID1
  v_L1
] 
%
%
%
%
%

{% 
yi = ...
[ i_VS1
  i_V1
  i_ID1
  i_ID1
  i_L1
] 
%
%
%
%

M0 = ...
[ C1  0
  0  L1
]
\[ A_0 = \ldots \]
\[
\begin{bmatrix}
-(1/R1) & --1 & -1 & 0 \\
\end{bmatrix}
\]

\[ B_0 = \ldots \]
\[
\begin{bmatrix}
0 & 0 & 0 & -1 & 1 & 0 \\
\end{bmatrix}
\]

\[ C_0 \_V = \ldots \]
\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\[ C_0 \_I= \ldots \]
\[
\begin{bmatrix}
0 & (--1) & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[ D_0 \_V = \ldots \]
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ D_0 \_I = \ldots \]
\[
\begin{bmatrix}
0 & 0 & (-1) & 0 & 0 & (-1) & 0 & 0 & 0 \\
\end{bmatrix}
\]
\[ M0inv = \text{simple}(\text{inv}(M0)) \]
\[ A_{all\{3\}} = M0inv \times A0 \]
\[ B_{all\{3\}} = M0inv \times B0 \]
Switch to:
Topology3
Topology0
si_x \{3\} = ...
\[[0 \ 0 \ ]

si_u \{3\} = ...
\[[0 \ ]

si_d \{3\} = -1
si_c \{3\} = D
%% Submodel 4

{%
x = ...
  [v_C1
   i_L1]
%
%
%
%
%
%}

{%
u = ...
  [v_VD1
   v_V1]
%
%
%
%
%
%}

{%
yv = ...
  [v_VD1
   v_V1
   v_C1
   v_L1]
%
%
%
%
%
%}

{%
yi = ...
  [i_VD1
   i_V1
   i_L1]
%
%
%
%
%
%}

M0 = ...
  [
  C1  0
  0  L1
  ]
\[ A_0 = \ldots \]
\[
\begin{bmatrix}
-(1/R1) & (\ldots 1)
-1 & 0
\end{bmatrix}
\]

\[ B_0 = \ldots \]
\[
\begin{bmatrix}
0 & 0
-1 & 0
\end{bmatrix}
\]

\[ C_{0\_V} = \ldots \]
\[
\begin{bmatrix}
0 & 0 & 0 & 0
1 & 0 & -1 & 0
\end{bmatrix}
\]

\[ C_{0\_I} = \ldots \]
\[
\begin{bmatrix}
0 & (\ldots 1)
0 & 0
0 & 1
\end{bmatrix}
\]

\[ D_{0\_V} = \ldots \]
\[
\begin{bmatrix}
1 & 0 & 0 & 0
0 & 1 & 0 & 0
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[ D_{0\_I} = \ldots \]
\[
\begin{bmatrix}
0 & 0 & 0 & 0
\end{bmatrix}
\]
Comparing the test result with the manually derived result in Section 2.2, it looks quite different due to two reasons:

1. The software does not simply the equation, where it kept all the symbols and signs with it.
2. For some of the topologies, there shouldn’t be a relevant GSSM result if derived manually. However, as software takes diode and switch in as 0-v voltage resource and 0-A current resource, it will generate a format of GSSM for all the topologies.

Taking a few steps further to simplify the generated result, it can be seen that the test case provide an accurate result same as the one derived in Section 2.2.
Chapter 5
Conclusions and Future Work

Conclusion

The research project successfully accomplishes the majority of design goals as specified in the original proposal to develop a symbolic expression for analysis on stability of a power converter circuit system consisting of an electronic circuit controlled by a control system. The test result on the designed test circuit is well-organized and accurate to meet with the requirements of a state space model matrix equation that can be applied to Stanislav G. Kriventsov’s non-linear analysis method on dynamic electronic system. Since Kriventsov’s method is also available to be developed into a computer program, the MATLAB output file generated by this project can directly applied to serve as an input file for the computer program conducting Krivetsov’s analysis algorithm. With this developed research project, analysis with GSSM on RLCM electronic network is no longer a time consuming process even when it comes to a large-scale and complex circuit case.

The accomplishment of this research is based on achievement of the project milestones, including optimization of electronic circuit analysis algorithm, design of control system analysis algorithm and combination of the two analysis process. Expansion of the electronic component class and optimization of circuit analysis algorithm allows the software to deal with more complex electronic circuit which includes switchable elements. Design of control system analysis from a different view allows the software to generate state space model matrix for control system with fundamental type of control system blocks included. The generated state space model matrix is applied to the switchable elements in electronic circuit to control their switching status. Last, the project successfully joins the effort on the analysis algorithm for the two different systems
together to provide a full result on the output MATLAB file, including a full state space model matrix and switching surface matrix for all potential topologies.

*Future Work*

While the current software successfully generate a full GSSM result for the designed test case, it is expected to generate a full GSSM results for all different kinds of electronic network that both includes an electronic circuit connected to a control flow diagram. The current software is well designed to deal with various electronic circuits and control system separately, and can generate state space model matrix for both of them. Improvement needs to be added to the combined analysis process, so that both of the state space model results can work together to generate a switching surface for a combined network. Some of the design of the steps in the analysis process in the software is simplified to deal with the final test case. However, in order to work with unknown circuit cases provide by users, those steps needs to be redesigned to be more flexible and accurate for all different kinds of situations. It is also good to make the software capable of simplifying the symbolic result to make it easier to be understood by users. Once all of the above changes are accomplished, future developers can include more component classes and analysis algorithm for both control system and electronic system, to improve the complexity of the network that the software can deal with. A list of specific tasks that can be done in the future to achieve all the improvement mentioned above is shown as following;

1. Derive an accurate C and D matrix based on the state space model method in *Computer-sided analysis of Electronic Circuits*, and redesign the function in the software to generate the accurate result of C and D matrix for all cases that can be developed from the included components.
2. Redesign the process to track the information of the control system connected to the switches, where a state space model matrix result for control system should be caught and kept in the switch.
3. With the help of 1 and 2, the function to generate switching surface should be optimized to generate accurate switching surface for all different cases that can be developed from the included components.

4. Add a function in the software to simplify the symbolic GSSM result to make it clean and precise, so that users can easily understand.

5. Improve the control system analysis algorithm, so that the state space model result can be generated for a control system that has a look

6. Improve the combined analysis algorithm, so that the electronic network that includes a loop feed-back between its electronic circuit and control system can be dealt with by the software.

7. Include more component classes to improve the capability of the software for the electronic network with higher complexity.
Appendix A
Graph Definitions

A graph $G$ is described mathematically as being comprised of two related sets:

- A finite nonempty set $V(G)$ of objects, each of which is called a vertex or node.
- A possibly empty set $E(G)$ of 2-element subsets of $V(G)$, each of which is called an edge, branch, or arc.

Every graph has a diagram associated with it:

- Each vertex is shown as a small circle.
- Each edge is shown as a line or arc that connects the two vertices with which it is associated.

Let $G$ be a graph and $e = \{u, v\}$ be an edge of $G$, then

- $e$ could just as easily have been written $e = \{v, u\}$
- $u$ and $v$ are each said to be an endpoint of $e$
- $u$ and $v$ are said to be adjacent in $G$
- $e$ is said to join $u$ and $v$
- $e$ is said to be incident to $u$ and incident to $v$

A multigraph may contain more than one edge that is adjacent to the same pair of vertices.

- Strictly speaking, a graph may not contain more than one edge that is adjacent to the same pair of vertices.
- Two or more edges that are adjacent to the same pair of vertices are called parallel edges.

The order of a graph $G$ is the number of vertices in $G$ or $|V(G)|$.
The size of a graph $G$ is the number of edges in $G$ or $|E(G)|$.
The degree of a vertex $v$ of a graph $G$ is the number vertices that are adjacent to $v$.

- For a true graph (as opposed to a multigraph), this is the same as the number of edges that are incident to $v$.
- The neighborhood $N(v)$ of vertex $v$ is defined as: $N(v) = \{u \in V(G) \mid \{u,v\} \in E(G)\}$
Path: An ordered set of edges \( \{e_1, e_2, \ldots, e_K\} \) in a graph \( G \) is called a path between two vertices \( u \) and \( v \), if all of the following are true:

- Consecutive edges \( e_k \) and \( e_{k+1} \) always have a common endpoint.
- No vertex in \( G \) is the endpoint of more than two edges in the set.
- \( u \) is the endpoint of exactly one edge in the set, and so is \( v \).

Connected graph: A graph \( G \) is said to be connected, if there exists a path between any two vertices of the graph.
Appendix B
State-Space Modeling of RLCM Networks

This appendix gives the derivation of state-space analysis method in [Chua1975].

Two assumptions are made in the state-space analysis of any RLCM network:
- There are no loops consisting of independent voltage sources only.
- There are no cutsets consisting of independent current sources only.

If either of these assumptions is violated, the network has no unique solution. Based on these assumptions, a normal tree or a proper tree can be constructed to contain the following:
- all independent voltage sources.
- no independent current sources.
- as many capacitors as possible.
- as few inductors as possible.

If the subscript $\mathcal{T}$ is used to denote elements of the normal tree and $\ell$ is used to denote links of the corresponding co-tree, then the network branch voltages and branch currents may be partitioned as follows:

$$
\begin{bmatrix}
    v_{E3} & v_{C3} & v_{R3} & v_{L3} & v_{J\ell} & v_{L\ell} & v_{R\ell} & v_{C\ell}
\end{bmatrix}^T
$$

(B-1)

$$
\begin{bmatrix}
    i_{E3} & i_{C3} & i_{R3} & i_{L3} & i_{J\ell} & i_{L\ell} & i_{R\ell} & i_{C\ell}
\end{bmatrix}^T
$$

(B-2)

The topology of the network can then be described using the fundamental cut set matrix $D$, which corresponds to the generalized form of KCL, and the fundamental loop matrix $B$, which corresponds to the generalized form of KVL:

$$
D_i = \begin{bmatrix}
    1_{E3} & 0 & 0 & 0 & F_{11} & F_{12} & F_{13} & F_{14} \\
    0 & 1_{C3} & 0 & 0 & F_{21} & F_{22} & F_{23} & F_{24} \\
    0 & 0 & 1_{R3} & 0 & F_{31} & F_{32} & F_{33} & 0 \\
    0 & 0 & 0 & 1_{L3} & F_{41} & F_{42} & 0 & 0
\end{bmatrix} i = 0
$$

(B-3)
The \( i-v \) characteristics for the RLCM elements of the network are described as:

\[
\begin{align*}
R : \quad & v_R = R i_R \quad \text{or} \quad i_R = G v_R \\
C : \quad & i_C = \frac{d}{dt} v_c \\
LM : \quad & \begin{bmatrix} v_{L1} \\ v_{L2} \end{bmatrix} = \begin{bmatrix} L_1 & M \\ M & L_2 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix}
\end{align*}
\]

All of these equations, (B-3) to (B-7), constitute the original network equations for the RLCM network as a system of DAEs. [Chua1975] proposed a 5 step state-space analysis method that can reduce the DAEs to a normal form state-space model:

\[
\dot{x} = Ax + Bu + B^{(1)} \ddot{u}
\] (B-8)

Step 1: From the topology equations:

\[
i_{C3} = -F_{21}^T i_{J} - F_{22}^T i_{L} - F_{23}^T i_{R} - F_{24}^T i_{C}
\] (B-9)

\[
v_{L3} = F_{12}^T v_{E3} + F_{13}^T v_{C3} + F_{12}^T v_{R3} + F_{14}^T v_{L3}
\] (B-10)

Step 2: Also from the topology equations:

\[
i_{L3} = -F_{41}^T i_{J} - F_{42}^T i_{L}
\] (B-11)

\[
v_{C3} = F_{12}^T v_{E3} + F_{24}^T v_{C3}
\] (B-12)

Step 3: Substitute the inductors’ characteristic equations into the topology equation:

\[
v_{L3} = L_{33} \frac{d}{dt} (-F_{41}^T i_{J} - F_{42}^T i_{L}) + L_{34} \frac{d}{dt} i_{L}
\]

\[
= -L_{33} F_{41} \frac{d}{dt} i_{J} + (L_{34} - L_{32} F_{42}) \frac{d}{dt} i_{L}
\] (B-13)
Similarly, substitute the capacitor’s characteristic equations into the topology equation:

\[
\mathbf{i}_{C} = C_t \frac{d}{dt} \mathbf{v}_{C} = C_t \mathbf{F}_{14} \frac{d}{dt} \mathbf{v}_{E3} + C_t \mathbf{F}_{24} \frac{d}{dt} \mathbf{v}_{C3}
\]  \hspace{1cm} (B-14)

Step 4: Express \(\mathbf{v}_{R3}\) in terms of \((\mathbf{v}_{E3}, \mathbf{v}_{C3}, \mathbf{i}_{Jr}, \mathbf{i}_{Lr})\), and do the same for \(\mathbf{i}_{Rr}\).

From the topology equations:

\[
\mathbf{i}_{R3} = -\mathbf{F}_{31} \mathbf{i}_{Jr} - \mathbf{F}_{32} \mathbf{i}_{Lr} - \mathbf{F}_{33} \mathbf{i}_{Rr}
\]  \hspace{1cm} (B-15)

\[
\mathbf{v}_{Rr} = \mathbf{F}_{13}^T \mathbf{v}_{E3} + \mathbf{F}_{23}^T \mathbf{v}_{C3} + \mathbf{F}_{33}^T \mathbf{v}_{R3}
\]  \hspace{1cm} (B-16)

Substitute the resistor’s characteristic equations:

\[
\mathbf{i}_{R3} = \mathbf{G} \cdot \mathbf{v}_{R3} = -\mathbf{F}_{31} \mathbf{i}_{Jr} - \mathbf{F}_{32} \mathbf{i}_{Lr} - \mathbf{F}_{33} \mathbf{G} \cdot (\mathbf{F}_{13}^T \mathbf{v}_{E3} + \mathbf{F}_{23}^T \mathbf{v}_{C3} + \mathbf{F}_{33}^T \mathbf{v}_{R3})
\]  \hspace{1cm} (B-17)

Define:

\[
\mathbf{G} = \mathbf{G} \cdot + \mathbf{F}_{33} \mathbf{G} \cdot \mathbf{F}_{33}^T
\]  \hspace{1cm} (B-18)

Then:

\[
\mathbf{v}_{R3} = \mathbf{G}^{-1} \cdot (-\mathbf{F}_{31} \mathbf{i}_{Jr} - \mathbf{F}_{32} \mathbf{i}_{Lr} - \mathbf{F}_{33} \mathbf{G} \cdot (\mathbf{F}_{13}^T \mathbf{v}_{E3} + \mathbf{F}_{23}^T \mathbf{v}_{C3} + \mathbf{F}_{33}^T \mathbf{v}_{R3}))
\]  \hspace{1cm} (B-19)

Similarly:

\[
\mathbf{v}_{Rr} = \mathbf{R} \cdot \mathbf{i}_{Rr} = \mathbf{F}_{13}^T \mathbf{v}_{E3} + \mathbf{F}_{23}^T \mathbf{v}_{C3} + \mathbf{F}_{33}^T \mathbf{R} \cdot (-\mathbf{F}_{31} \mathbf{i}_{Jr} - \mathbf{F}_{32} \mathbf{i}_{Lr} - \mathbf{F}_{33} \mathbf{i}_{Rr})
\]  \hspace{1cm} (B-20)

Define:

\[
\mathbf{R} = \mathbf{R} \cdot + \mathbf{F}_{33} \mathbf{R} \cdot \mathbf{F}_{33}
\]  \hspace{1cm} (B-21)

Then:

\[
\mathbf{i}_{Rr} = \mathbf{R}^{-1} \cdot (\mathbf{F}_{13}^T \mathbf{v}_{E3} + \mathbf{F}_{23}^T \mathbf{v}_{C3} - \mathbf{F}_{33} \mathbf{R} \cdot \mathbf{F}_{33} \mathbf{i}_{Jr} - \mathbf{F}_{33} \mathbf{R} \cdot \mathbf{F}_{33} \mathbf{i}_{Lr})
\]  \hspace{1cm} (B-22)

Step 5: Final substitution

According to the capacitor and inductor’s characteristic equations:
C_3 \frac{d}{dt} v_{C3} = -F_{21} i_{J_1} - F_{23} i_{J_3} - F_{23} i_{R_1} - F_{23} i_{C_1} \quad (B-23)

L_{i_1} \frac{d}{dt} i_{L_1} = F_{12}^T v_{E3} + F_{22}^T v_{C3} + F_{32}^T v_{R_3} + F_{42}^T v_{L_5} + L_{i_1} F_{41} \frac{d}{dt} i_{J_1} + L_{i_3} F_{42} \frac{d}{dt} i_{L_1} \quad (B-24)

Substitute the expressions for $v_{L_3}$ and $i_{C_1}$, i.e., equations (B-13) and (B-14), into the above two equations:

C_3 \frac{d}{dt} v_{C3} = -F_{21} i_{J_1} - F_{23} i_{L_1} - F_{23} i_{R_1} - F_{23} C_t F_{14}^T \frac{d}{dt} v_{E3} - F_{23} C_t F_{24}^T \frac{d}{dt} v_{C3} \quad (B-25)

L_{i_1} \frac{d}{dt} i_{L_1} = F_{12}^T v_{E3} + F_{22}^T v_{C3} + F_{32}^T v_{R_3} + (-F_{42}^T L_{23} + L_{i_3}) F_{41} \frac{d}{dt} i_{J_1} + (F_{42}^T (L_{i_3} - L_{33} F_{42}) + L_{i_3} F_{42}) \frac{d}{dt} i_{L_1} \quad (B-26)

After substitute the expression for $i_{R_1}$ and $v_{R_3}$ (equations (B-19) and (B-22)) into the above two equations, the state space form can be given symbolically:

$$M^{(0)} \frac{d}{dt} \begin{bmatrix} v_{C3} \\ i_{L_1} \end{bmatrix} = A^{(0)} \begin{bmatrix} v_{C3} \\ i_{L_1} \end{bmatrix} + B^{(0)} \begin{bmatrix} v_{E3} \\ i_{J_1} \end{bmatrix} + B_1^{(0)} \frac{d}{dt} \begin{bmatrix} v_{E3} \\ i_{J_1} \end{bmatrix} \quad (B-27)$$

$$M^{(0)} = \begin{bmatrix} C_3 + F_{24} C_t F_{24}^T & 0 \\ 0 & (L_{i_1} - F_{42}^T L_{33} - L_{i_3} F_{42} + F_{42}^T L_{33} F_{42}) \end{bmatrix} \quad (B-28)$$

$$A^{(0)} = \begin{bmatrix} -F_{23} R^{-1} F_{23}^T & (-F_{23} + F_{23} R^{-1} F_{33} R_{33} F_{32}) \\ (F_{22}^T - F_{32} G^{-1} F_{33} G_{33} F_{23}) & -F_{32} G^{-1} F_{32} \end{bmatrix} \quad (B-29)$$

$$B^{(0)} = \begin{bmatrix} -F_{23} R^{-1} F_{13}^T & (-F_{21} + F_{23} R^{-1} F_{33} R_{33} F_{31}) \\ (F_{12}^T - F_{32} G^{-1} F_{33} G_{13} F_{13}) & -F_{32} G^{-1} F_{31} \end{bmatrix} \quad (B-30)$$

$$B_1^{(0)} = \begin{bmatrix} -F_{24} C_t F_{14}^T & 0 \\ 0 & -F_{42}^T L_{33} F_{41} + L_{i_3} F_{41} \end{bmatrix} \quad (B-31)$$

where

$$R = R_t + F_{33} R_{33} F_{33} \quad (B-32)$$
\[ G = G_3 + F_{33} G_t F_{33}^T \]  \hspace{1cm} (B-33)

The \( M^{(0)} \) matrix is nonsingular as long as both of the following are true [Thottuvelil1991]

Both the \( C_3 \) and \( C_t \) matrices are positive definite

The matrix
\[
\begin{bmatrix}
L_{33} & L_{3t} \\
L_{t3} & L_{tt}
\end{bmatrix}
\]
is positive definite

Moreover, capacitors and inductors are decoupled in the \( M^{(0)} \) matrix.
Appendix C
Netlist for Test Cases

Test Case 4.1-1

Node a;
Node b;
Node c;
Node gnd;

VoltageSource V1 node1(a) node2(gnd);
Resistor R1 node1(a) node2(b);
Inductor L1 node1(b) node2(c) l(1);
Capacitor C1 node1(c) node2(gnd) c(0.25);
Resistor R1 node1(c) node2(gnd) r(1);

Test Case 4.1-2

Node a;
Node b;
Node c;
Node gnd;

VoltageSource V1 node1(a) node2(gnd);
Switch S1 node1(a) node2(b);
Diode D1 node1(gnd) node2(b);
Inductor L1 node1(b) node2(c) l(1);
Capacitor C1 node1(c) node2(gnd) c(0.25);
Resistor R1 node1(c) node2(gnd) r(1);
Test Case 4.2-1
   InputBlock I1;
   OutputBlock O1 in(I1);

Test Case 4.2-2
   InputBlock I1;
   GainBlock G1 in(I1);
   OutputBlock O1 in(G1);

Test Case 4.2-3 (part 1)
   InputBlock I1;
   InputBlock I2;
   SumBlock S1 in_1(I1) in_2(I2) in_2.sign(-1);
   GainBlock G1 in(S1);
   OutputBlock O1 in(G1);

Test Case 4.2-3 (part 2)
   InputBlock I1;
   InputBlock I2;
   SumBlock S1 in_1(I1) in_2(I2);
   GainBlock G1 in(S1);
   OutputBlock O1 in(G1);

Test Case 4.3-1
   Node a;
   Node b;
   Node c;
   Node gnd;

   VoltageSource V1 node1(a) node2(gnd);
   Switch S1 node1(a) node2(b);
   Diode D1 node1(gnd) node2(b);
   Inductor L1 node1(b) node2(c) l(1);
   Capacitor C1 node1(c) node2(gnd) c(0.25);
   Resistor R1 node1(c) node2(gnd) r(1);
InputBlock I1;
SignalBlock Sig1;
Comparator Cpl in_1(I1) in_2(Sig1) in_3(S1);
OutputBlock O1 in(Cp1);
BIBLIOGRAPHY

ACADEMIC VITA
Mingyang Zhou
miz5067@psu.edu

EDUCATION

Bachelor of Science in Electrical Engineering (with honors) Expected Graduation Time: May 2014
Bachelor of Science in Engineering Science, Engineering College’s Honors Curriculum
Minor in Engineering Leadership Development
Schreyer Honors College
The Pennsylvania State University, University Park, PA

THESIS

Topic: Software Design and Optimization for power converter stability analysis with symbolic expression
Thesis Supervisor: Dr. Jeffrey Mayer
Honors Advisor: Dr. John Mitchell

RESEARCH AND PROJECTS

Engineering Leadership Minor Capstone Design Project Sep 2013 – Present
- Worked with two multidisciplinary engineering teams from Penn State and Brigham Young University to accomplish a sponsored design project.
- Designed a phone app with Android java in order to remote control the product
- Designed a smart control system on the product with Raspberry pi in order to support wireless communication, information processing and app control.
Quad Copter Design Project Sponsored by Boeing Sep 2013 – Present
- Optimized the design of a quad copter with a multidisciplinary engineering team
- Designed the power system in order to make a 15-minute efficient flight.
- Designed the electrical circuit for pick-mechanism to pick up heavy tennis balls.

PROFESSIONAL EXPERIENCE

Teaching Intern for Electrical Engineering Department Spring 2014
Engineer Internship with CEC Eleventh Design & Research Institution Jul 2013 – Aug 2013
Software Engineer Internship with Chengdu Haicheng Technology Corporation Aug. 2013

ACADEMIC/OUTREACH EXPERIENCE

Engineering Ambassador Sep.2012 – Present
Member of Student Space Programs Laboratory Sep.2011-May.2012
Member of Chinese Connection Jan.2012 – May. 2013
Media and Technique Support for Penn State Speech Association Sep.2013 – Present
Member of 2013 Thon Rules and Regulations Committee Oct.2012 – Mar.2013

HONORS

Certificate of completion of lean-six-sigma process improvement Summer 2012
Dean’s List of Engineering College, All Semesters Fall 2010 – Spring 2013
Bronze Award for University Physics Competition Fall 2010
Tau Beta Pi Honor Society Fall 2013

SKILLS

Programming Languages: C++, MATLAB, CUPL, Able and Java