Behavioral Modeling and Simulation of MEMS Electrostatic and Thermomechanical Effects

by
Gilbert Wong

Master's Project Report

in

Electrical Engineering

at

Carnegie Mellon University

Advisors:

Professor Gary Fedder
Dr. Tamal Mukherjee

Spring 2004

Abstract

This thesis extends and updates the Nodal Design of Actuators and Sensors (NODAS) library previously developed at Carnegie Mellon University. NODAS is a library of atomic element lumped parameter behavioral models written in Verilog-A, an analog hardware description language (AHDL). It is used for the simulation of MEMS devices in a SPICE-like circuit simulator. Physics modeling, accuracy verification, and application examples are presented. The modeling of 2D beams and 2D electrostatic gaps for MEMS devices is presented in this thesis, however, the physics can be extended to 3D.

The NODAS beam model was updated and new physical effects were added. The inconsistent use of a user parameter which caused the length of the beam to be miscalculated was corrected. A new variable was created to address this problem and verification of the NODAS beam model compared with analytic equations. When compared to analytic equations for a fixed-fixed beam under a uniform distributed load and a fixed-guided beam with axial compressive stress under a uniform distributed load, the NODAS beam model was shown to be accurate to within $6.1\text{x}10^{-6}$ percent and $15\text{x}10^{-3}$ percent respectively when using multiple NODAS beam segments. The moment generated by the differences between the thermal coefficients of expansion in a multi-layer CMOS-MEMS beam was implemented. A multi-layer CMOS-MEMS beam under an applied temperature was simulated in NODAS and ANSYS. The NODAS simulation reported in-plane tip deflections to within 2% of the ANSYS simulation for different metal layer combinations and offsets. A thermal conduction model was added to account for the effects of electrothermal heating. The electromechanical gap model was updated and an electric-only gap model was created. A electrostatically actuated fixed-fixed beam simulation using electromechanical gap models was shown to produce beam deflections to within 2.2% of an ANSYS simulation when 128 NODAS beam/gap segments were used. The electric-only gap model was created to reduce the simulation time of large models such as an RF MEMS varactor. A thermal actuator, a varactor, and a mixer application example are given to demonstrate composability and hierarchical design. Finally, future areas of research are discussed.

Table of Contents

Abstract i

Table of Contents ii

List of Figures v

Code Listings vii

Chapter 1 Introduction 1

1.1 Prior Work 2

1.2 MEMS Processing 4

1.3 NODAS Background 5

Chapter 2 Electrothermal Beam Modeling and Verification 11

2.1 Mechanical Beam Model Update 11

2.2 Thermal Expansion 13

2.2.1 Neutral Axis Calculation 14

2.2.2 Moment Calculation 15

2.3 Thermal Conduction 17

2.4 Verification 19

2.4.1 Fixed-Fixed and Fixed-Guided beams 19

2.4.2 Multimorph Beam 21

2.4.3 Electrothermal heating 22

Chapter 3 Electrostatic Gap Modeling and Verification 24

3.1 Lumping 24

3.1.1 Lumping Distributed Forces 26

3.2 Gap Topology 29

3.3 Electrostatic Forces 32

3.3.1 Rotated Parallel Plate Approximation 33

3.3.2 Electrostatic Force Equation 34

3.4 Damping 35

3.5 Contact Model 36

3.6 Gap Model Lumping Equations 39

3.7 Lateral Electrostatic Force 40

3.8 Frames of Reference 41

3.9 Electrical Modeling 42

3.10 Special cases 44

3.11 Electric only gap model 45

3.12 Verification 47

3.12.1 Contact Model 47

3.12.2 Fixed-Fixed Beam 49

Chapter 4 Application Examples 53

4.1 Thermal Actuator 53

4.2 Varactor 56

4.3 Resonant Mixer 63

Chapter 5 Conclusions 70

5.1 Beam model 70

5.2 Gap model 71

5.3 Future work 72

Bibliography 73

Appendix A: Verilog-A Code 77

A.1 Electrostatic Gap Model Submodule 77

A.2 Electric Only Gap Model Submodule 89

A.3 Three Conductor Electrostatic Gap Model 97

A.4 Three Conductor Electric Only Gap Model 100

A.5 Thermal beam model 102

A.6 Neutral axis include file 108

Appendix B: Verilog-A Modeling 122

B.1 Division 122

B.2 Hidden States 123

B.3 Multiple discipline.h references 124

B.4 Tolerances 125

B.5 Nodal Conventions 126

B.6 Through and Across Variables 127

B.7 Analog Functions 127

B.9 NODAS Code Architecture 129

Appendix C: ANSYS Scripts 131

C.1 Fixed-Fixed Beam 131

C.2 Beam Lateral Bending Moment 134

C.3 Electrothermal Conduction 138

List of Figures

Figure 1-1. CMOS MEMS post-processing steps, cross section view. 5

Figure 1-2. Levels of abstraction 9

Figure 2-1. Beam Corner effects. 12

Figure 2-2. Thermal moment convention. 13

Figure 2-3. Neutral axis calculation model. M3, M2, M1, poly stack. 14

Figure 2-4. Electrothermal conduction models. 18

Figure 2-5. Beam verification problems. 20

Figure 2-6. Beam verification results. 21

Figure 2-7. Temperature profile of a poly resistor. 23

Figure 3-1. Distributed electrostatic force on a MEMS beam. 25

Figure 3-2. NODAS Electrostatic Gap Symbol. 25

Figure 3-3. Electrode configurations. 29

Figure 3-4. Rotated parallel plate approximation. 33

Figure 3-5. Squeeze film damping. 35

Figure 3-6. Canonical gap problem. 36

Figure 3-7. Electrostatic Force contact model: force vs. gap plots. 37

Figure 3-8. gcontact and gmin 38

Figure 3-9. Frames of reference. 41

Figure 3-10. Current flows. Iattach (dashed line), Itip (solid line). 43

Figure 3-11. Finger based varactor design, A.Oz. [30]. Area and gap tuning. 46

Figure 3-12. NODAS schematic for canonical gap problem. 48

Figure 3-13. Contact model simulation plots: gap versus voltage. 48

Figure 3-14. Fixed-Fixed beam with electrostatic actuation. 49

Figure 3-15. Fixed-Fixed beam mode shapes. 49

Figure 3-16. 4 NODAS beam and gap segments subcell. 50

Figure 4-1. Thermal Actuator 1A layout in Jazz 0.35 process. A. Oz [30]. 54

Figure 4-2. Cross section views of the offset metal layer stack in Actuator 1A. 55

Figure 4-3. Thermal Actuator 1A NODAS schematic. 55

Figure 4-4. Layout view of beam based varactor design, Jazz 0.35 process [30]. ..57

Figure 4-5. Capacitor finger subcell. 58

Figure 4-6. NODAS schematic hierarchy of varactor design in Jazz process. 59

Figure 4-7. Varactor quality factor. 61

Figure 4-8. Finger displacements. 61

Figure 4-9. Capacitance versus voltage plot of the Jazz 16 beam varactor. 64

Figure 4-10. Resonant Mixer Device 6 NODAS schematic. J. Stillman [29]. 64

Figure 4-11. Resonant Mixer Device 6 NODAS schematic. 65

Figure 4-12. Mixer “Device 6” NODAS AC simulation. 66

Figure 4-13. Parametric sweep of Young’s Modulus. 67

Figure 4-14. Parametric sweep of beam width. 68

Figure 4-15. Discrete Fourier transform (DFT) of transient simulation of “Device 6”.
68

Figure B-1. Tolerance examples. 126

Figure B-2. Nodal conventions. 126

Figure B-3. Through and across variables. 127

Code Listings

Listing 2-1. Revised beam model with corner effects. 12

Listing 2-2. Electrothermal conduction. 19

Listing 3-1. Static (layout) beam offset calculations. 30

Listing 3-2. Dynamic beam offset calculations. 31

Listing 3-3. Overlap region endpoint computation. 32

Listing 3-4. theta0 and gap calculation. 33

Listing 3-5. Electrostatic force per unit length. 34

Listing 3-6. Squeeze film damping force per unit length. 36

Listing 3-7. Contact model. 38

Listing 3-8. Beam shape function integral example. 39

Listing 3-9. Lumped forces and moments. 40

Listing 3-10. Lateral electrostatic force. 40

Listing 3-11. Frame of reference coordinate transforms. 42

Listing 3-12. Electrical model. 43

Listing 3-13. Special case: forces only exist when there is an overlap region. 44

Listing 3-14. Special case: forces in x direction only exist when there is an overlap. 45

Listing B-1. Division Example. 122

Listing B-2. Correct way to implement division. 122

Listing B-3. Incomplete if statement. 123

Listing B-4. Full case if statement. 123

Listing B-5. if statement with bar initialized to 0. 124

Listing B-6. Full case statement examples. 124

Listing B-7. Example of Verilog-A analog function definition. 127
Listing B-8. Example of Verilog-A analog function call. 128
Listing B-9. Redundant calculations: 128
Listing B-10. Initial step block. 129

1 Introduction

Microelectromechanical system (MEMS) design presents a unique engineering challenge. In addition to knowledge of both mechanical and electrical design, a MEMS engineer must have computer aided design (CAD) tools to design products. The design process begins with hand analysis of the mechanical structure under investigation. Analytic equations give the engineer an estimate of the performance of the structure. In the traditional MEMS design flow, the engineer will proceed with a finite element or boundary element simulation using software such as Coventor's ANALYZER [1], ANSYS [2], or FEMLAB [3]. The hand calculated design is verified and refined using these packages until the structure meets the engineer's specifications. A macromodel of the structure can then be created that will reproduce the behavior of the structure based on external stimuli. A macromodel is useful when a simple yet accurate model of the system is desired. Macromodels can be created for use in system level simulators such as Matlab [4] or in circuit simulators such as SPICE [5]. If used in a electrical circuit simulator such as SPICE, an electrical equivalent of the macromodel can be generated to model the mechanical behavior [6] [7] [8]. This macromodel can then be used with electrical devices to simulate the system behavior.

Although both structural and electronic halves of the MEMS design can be completed separately, it is often necessary to simulate the system as a whole in order to verify the functionality of the signal paths from the mechanical structures to the electrical circuits. Creating a macromodel of the mechanical system by extracting the behavior from a finite element or boundary element simulation and then using the macromodel in a circuit level or system level simulator is a valid design flow. However, if the mechanical design needs to be changed, the finite element or boundary element model must be recreated and the macromodel must be re-extracted. Finite element packages by themselves are not suitable to simulate the electrical circuit behavior. Likewise, off-the-shelf circuit simulators do not have the features necessary to simulate mechanical devices. The Nodal Design of Actuators and Sensors (NODAS) library [17] created at Carnegie Mellon University addresses the problem of co-simulation of mechanical and electrical devices by allowing the MEMS engineer to work in a single simulator. NODAS is written in Verilog-A [9] and can be used in any circuit simulator that supports the Verilog-A language standard. Cadence Spectre [10] [11] is an example of such a simulator.

1.1 Prior Work

Prior work in MEMS device modeling and simulation focused in the area of nodal analysis. Nodal analysis [9] solves the set of equations where the sum of flows into a node equals zero. In the electrical domain, Kirchhoff's current law (where current is the flow) is used. The mechanical equivalents are forces and moments. Prior work in the area of nodal simulation of MEMS devices includes using electrical equivalent circuits to represent mechanical structures and creating behavioral models of mechanical structures in system level simulators such as Matlab and hardware description language (HDL) enabled circuit simulators such as Cadence Spectre and Synopsys Saber [12].

H.A.C. Tilmans demonstrated that electrical equivalents of micromechanical structures could be created [6] [7] [8]. Since HDL-enabled circuit simulators were not available at the time, the goal was to allow engineers to use existing circuit simulators such as SPICE to quickly analyze mechanical structures. Several examples of electrical equivalents to mechanical quantities are: voltage for force, current for velocity, charge for displacement, inductance for mass, and capacitance for compliance (the inverse of the spring constant). Using these circuit equivalents, it was demonstrated that complex structures, such as comb-finger resonators, could be modeled and simulated.

SUGAR from UC Berkeley [13] [14] takes a different approach. Instead of using existing circuit simulators which forces use of electrical equivalents, the modified nodal analysis algorithm was implemented in Matlab. Behavioral models of elements such as beams, electrostatic gaps, and simple circuit components (resistors, voltage sources, etc.) were created as element stamps compatible with the Matlab implementation of nodal analysis. Though the mechanical and electrical domains were integrated under one simulator, only simple transistor models are available, limiting simulation accuracy of modern foundry CMOS processes.

Unlike UC Berkeley, G. Lorenz and R. Nuel at Bosch [15] stepped away from implementing a simulator by developing MEMS models in the MAST HDL language that could be simulated in Synopsys Saber. The MAST HDL language (as well as the Verilog-A HDL language) requires that the mechanical equivalents of current (flow) and voltage (potential) be chosen. Force and displacement are examples of one such choice for mechanical flow and potential. Appendix B.6 gives a brief introduction to mechanical equivalents of electrical flow and potential. A more detailed discussion of flows and potentials can be found in [9]. Lorenz later used these MEMS models to implement Coventor ARCHITECT [16]. ARCHITECT currently contains behavioral models from domains such as electromechanical, optical, fluidic, and RF. Not only does it provide a rich set of mechanical behavioral models, these models can be used in conjunction electrical circuit models from CMOS foundries.

Work done at Carnegie Mellon University focuses on developing a set of atomic element behavioral models (beams, plates, and electrostatic gaps) which could be used to build models of complex MEMS devices [17] [18] [19] [20] [21] [22]. The behavioral modeling language chosen was Verilog-A. The Cadence Spectre simulator was used for development and simulation. Spectre supports the electrical device models provided by the CMOS foundries Carnegie Mellon uses for its CMOS-MEMS process. Therefore, accurate electrical and mechanical simulation could be performed in a single simulator using NODAS. Furthermore, by using a standard IC design tool flow such as Cadence, automatic layout generation from a NODAS schematic and layout extraction into a NODAS schematic can be performed on MEMS devices [23] [24].

1.2 MEMS Processing

Several options exist for fabricating MEMS devices. A brief (and non-exhaustive) overview of these processes will be given. In general, MEMS processes can be split into two types: bulk and thin-film micromachining. In bulk micromachining, the silicon substrate is used to create mechanical structures. Typical thicknesses of these devices are in the hundreds of microns. Thin-film micromachining uses the deposited layers on top of the silicon wafer (polysilicon, oxide, and metal). The typical thicknesses are below ten microns. The MUMPS process [25] and post foundry CMOS micromachining processes [26] are typically used for thin-film micromachined MEMS devices. NODAS is able to model any rectangular geometries produced in any of these process. Currently, structural mechanics (static and dynamic), electrostatic forces, thermal expansion, thermal conduction, and electrothermal heating are modeled in NODAS. Though not currently implemented, other physical domains, such as kinematics, optics, and fluidics, and non-rectangular geometries, such as disks and spheres, can be added to the NODAS library.

Much MEMS research at Carnegie Mellon utilizes a CMOS-MEMS process. In an integrated complimentary MOSFET (CMOS) MEMS process, mechanical structures and electrical circuits are monolithically integrated. Typically, the electrical circuits are preamplifiers that amplify the small electrical signals and provide some common mode rejection (i.e. feed through rejection). Like the mechanical design process described above, the electrical design process is also iterative, starting with a hand design of the amplifier, followed by a simulation using a SPICE-like tool such as HSPICE [12] or Cadence Spectre [11]. CMOS-MEMS processing is discussed in [26] [27] [28]. Mechanical structures are built from metal, dielectric, and polysilicon beams. Figure 1-1a shows a cross section view of a CMOS-MEMS chip before post processing. A metal layer is used to protect

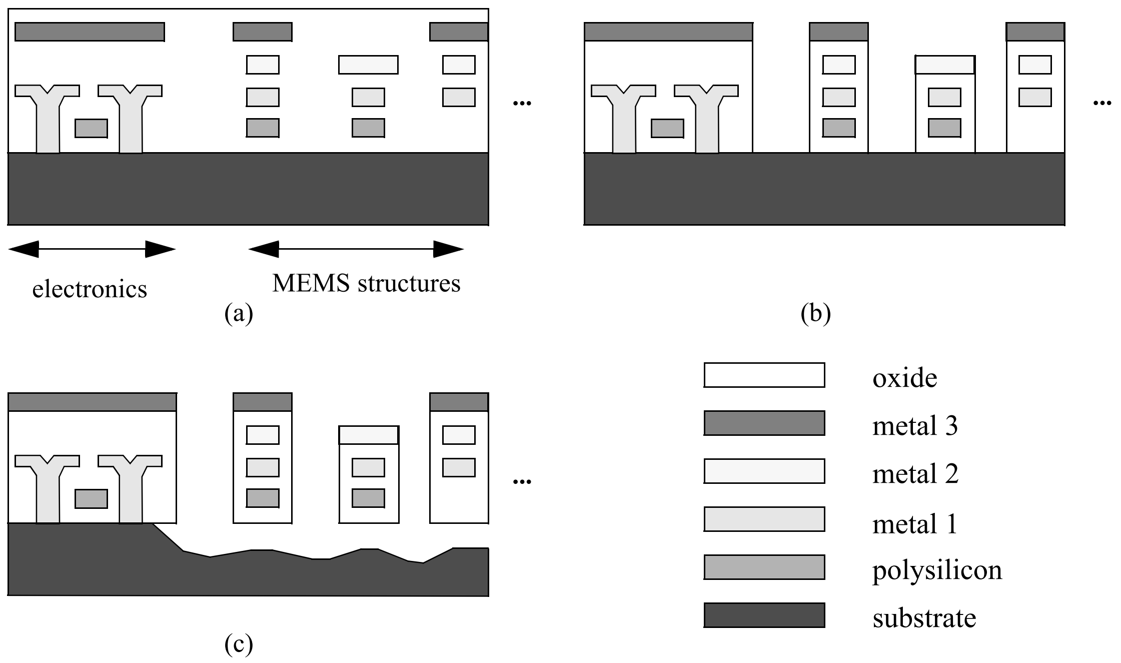


Figure 1-1. CMOS MEMS post-processing steps, cross section view.
(a) before postprocessing, (b) after anisotropic oxide etch, (c) after isotropic silicon etch.

electronic circuits as well as define the beams and other mechanical structures. In this example (a three metal layer process), metal 3 is used as a mask during the anisotropic oxide etch. Figure 1-1b shows the chip after the anisotropic oxide etch. The electronic devices have been protected from the etch by the metal 3 layer. The mechanical structure becomes apparent after the etch. Here, there are three beams which have been defined. The left beam has a polysilicon, metal 1, metal 2, and metal 3 stack. The center beam has a metal 2 mask and consists of a poly, metal 1 and metal 2 stack. Finally, the right beam consists of a metal 1, metal 2, and metal 3 stack. The final step in the CMOS MEMS process is to release the mechanical structures by using an isotropic silicon etch Figure 1-1c. Examples of devices fabricated using this process can be found in [18] [19] [27] [28] [29] [30].

1.3 NODAS Background

The NODAS library consists of atomic element models: anchors, beams, electrostatic gaps, and plates. The models are written in Verilog-A [9] and can be simulated in any Verilog-A enabled circuit simulator such as Cadence Spectre [10]. The Verilog-A enabled circuit simulator allows the

MEMS designer to simulate both the mechanical and electrical responses of the system together. Though this work focuses on the 2D beam and 2D electrostatic gap modeling for RF MEMS, a brief introduction on the other NODAS elements is useful. A detailed description of the models can be found in [17]. With the exception of the electrostatic gap model, all NODAS models have 2D and 3D versions. The 2D models capture the in-plane translational degrees of freedom (x, y) and a rotational degree of freedom (rotation about the z axis, ϕ_z) for each terminal of the model. Model terminals are used to communicate with the rest of the simulation schematic. For instance, a resistor has two terminals, positive and negative, and a beam has two terminals, one at each end of the beam's length. Each terminal can have multiple nodes. Each degree of freedom is a node in the simulation. For the 2D beam and electrostatic gap models, there are two terminals (each end of the beam or electrode), therefore a 2D model has six degrees of freedom (four translational and two rotational degrees of freedom). The 3D models capture the in-plane and out-of-plane translational degrees of freedom (x, y, z) and the rotational degrees of freedom about all three axes (ϕ_x, ϕ_y, ϕ_z). For the 3D beam and electrostatic gap models, there are twelve degrees of freedom (six translational and six rotational).

The NODAS library is modularized so that all of the physics is contained in a library of submodules called *NODAS_submodules*. This library contains the 2D and 3D mechanical beam elements (linear and non-linear), the electrical beam element, the 2D thermal beam element, the electrostatic gap elements (electromechanical and electric-only), the 2D comb elements, the 2D and 3D plate elements (rigid and elastic). By using these submodules, a library of one conductor models (for single structural layer polysilicon-based surfaced micromachined processes) and three conductor models (for CMOS-MEMS processes with embedded polysilicon, metal 1, metal 2, and metal 3 layers) can be easily maintained. For the three conductor models, poly is not counted as one of the conductors. The NODAS naming convention implies that poly is not used by itself in the CMOS-MEMS process, as a poly only beam cannot be fabricated (a metal mask layer is required). The submodules are written as if they were to be used for the three conductor case. To use the submodule as a single conductor model, only the appropriate parameters in the submodules are set. These libraries are called: *NODAS02_1C_2D* (one conductor, 2D), *NODAS02_1C_3D* (one conductor, 3D), *NODAS02_3C_2D* (three conductor, 2D), *NODAS02_3C_3D* (three conductor, 3D). The *02* indicates that this library architecture was developed in 2002. Finally, there are process-specific libraries that integrate these NODAS models (using soft links to these one of three conductor models) with process-specific parameters. The available processes are: AMS, HP, JAZZ, MUMPS, SIGE6, TSMC 035. For instance, the 2D AMS process uses the *NODAS02_3C_2D* models whereas the 3D MUMPS process uses the *NODAS02_1C_3D* models. See Appendix B.9 for more details on the NODAS code architecture.

The function of the atomic NODAS elements are now outlined. The NODAS anchor models set the angular displacement, cartesian displacements and temperature to zero. Two types of beam models are supported in NODAS: the linear and nonlinear models. The linear model is used when the beam bends a small distance or can be considered linear. The nonlinear model is used when the beam has a large geometric deflection or when the beam experiences a large axial stress. Two types of plate models are supported in NODAS: the elastic and rigid models. The elastic plate model is used when the plate experiences out-of-plane bending or in-plane stretching whereas the rigid plate model is used to transfer forces and moments without bending. Finally, two electrostatic gap models are supported in NODAS: electric-only and electromechanical. The electric-only gap model captures the capacitive effects of the gap, but does not generate any forces or moments. In addition to modeling the capacitive effects of the gap, the electromechanical gap model generates an electrostatic force and moment based on the applied voltages across the two beam electrodes.

Aside from the main benefit of being able to co-simulate mechanical and electrical devices together, the NODAS library allows the designer to create a composable and hierarchical design. The idea of composability is two fold. First, higher accuracy can be attained by breaking up a segment of the mechanical device into smaller pieces. For instance, a cantilever beam 100 μm long can be built using a single NODAS beam element model 100 μm long. However, more accuracy can be attained by breaking up the cantilever beam into two 50 μm beam elements, for instance. The discretization of the beam into smaller elements is similar to a mesh refinement in finite element packages. The second part of composability is that a single element can be connected, in different orientations, to other elements to form more complex elements. Figure 1-2 shows how a complex system such as a radio transceiver can be broken down into different levels of abstraction. The highest level of abstraction is the system level, followed by the component level (which includes the VCO, mixer, filter, LNA, etc.). Below the component level, there are functional elements like comb fingers. Finally, at the lowest level of abstraction are the atomic elements such as inductors, beams, and gaps. By connecting several atomic beam and gap elements together complex structures such as comb fingers can be created. Hierarchical design is used to abstract away and simplify large and complex structures. For instance, a varactor (Section 4.2) consisting of sixteen movable fingers and two thermal actuators would be tedious to create by using atomic elements only. By creating subcells which contain a basic repeatable structure and arraying these cells, the repetitive process of instantiating atomic elements can be eliminated. Multiple levels of subcells can be used to further simplify the design process. Furthermore, hierarchical design facilitates topology reuse. Subcells from previous designs can be reused as is or copied and modified to speed up the design process.

This thesis focuses on refinements to and verification of the 2D beam models and 2D electrostatic gap model. Verification can be split into three major areas (Table 1-1). Individual elements must be verified by themselves to ensure that the physics is modeled correctly when compared to analytic equations and finite element simulations. Next, the elements must be composable. Using many identical elements to discretize the problem should yield more accurate results when compared to analytic equations and finite element simulations. Finally, composing different atomic ele-

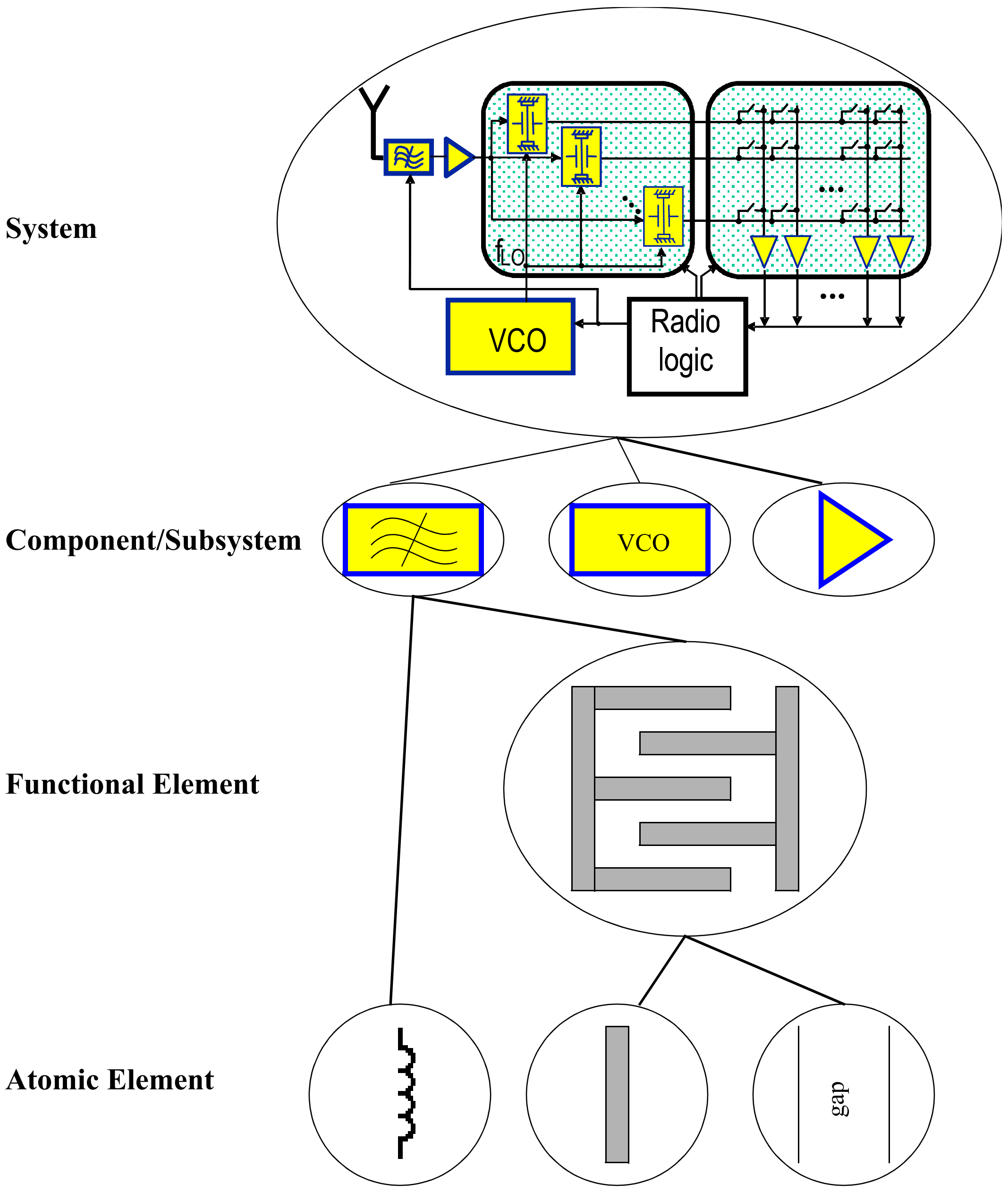


Figure 1-2. Levels of abstraction

Table 1-1. Types of Verification.

Type of Verification	Description
Individual element	Verifying that each element is accurate and models the physics intended. The model should also be invariant to rotations and flipping of the schematic symbols. Checking the accuracy against analytic and finite element analysis.

Table 1-1. Types of Verification.

Type of Verification	Description
Composability	Discretizing of a problem into many identical elements. For example, breaking up a 100 μm beam into ten 10 μm beam segments. Checking the accuracy against finite element analysis.
Applications	Using heterogeneous elements to build complex structures. For instance, using beam, gap and plate elements to build a crab leg resonator. Checking the results against finite element analysis and experimental results.

ments to make complex devices or systems should be compared to experimental or finite element simulations.

The updated 2D beam models are discussed in Chapter 2, followed by a discussion of the 2D electrostatic gap models in Chapter 3. Canonical problems are presented in both these chapters with comparisons to analytic and finite element simulations. Several applications of the electrostatic gap and beam models are given in Chapter 4. The source code of the electrothermal beam model and the electrostatic gap model are listed in Appendix A. General Verilog-A modeling issues are presented in Appendix B. Finally, ANSYS scripts used for the verification of the models are presented in Appendix C.

2 Electrothermal Beam Modeling and Verification

Electrostatic or electrothermal actuation can be used to move MEMS structures into different positions. For small stroke actuation, electrostatic actuators can be used efficiently. For longer stroke actuators, electrothermal actuators are needed. The thermal actuator works by exploiting the different temperature coefficient of expansions (TCEs) of materials. The TCE measures how a material expands when exposed to a temperature source. The larger the TCE, the greater the expansion. By heating an actuator built from a composite beam (metal, oxide, and poly), the different TCEs of each material will cause stress in the beam because each material will want to expand at a different rate. This stress causes a bending moment and forces the beam to bend in such a way as to relieve this stress. To generate the heat necessary to raise the temperature of the beam, a polysilicon resistor is typically used. Modeling the electrothermal actuator requires models for thermal expansion, residual stress, thermal conductance, and electrothermal power generation. Though this thesis deals only with 2D in-plane thermal effects, the derivations presented in this chapter can be extended to include out-of-plane thermal effects. Before continuing with the discussion of the electrothermal beam model, a mechanical beam model update is discussed.

2.1 Mechanical Beam Model Update

After running simulations to verify the accuracy of the beam model, an error in the beam model Verilog-A code was identified. To test the accuracy of the NODAS beam models, a beam was modeled using varying numbers of NODAS beam elements. An AC analysis was run to extract the resonant frequency of this beam. Adding more NODAS beam elements for a given length

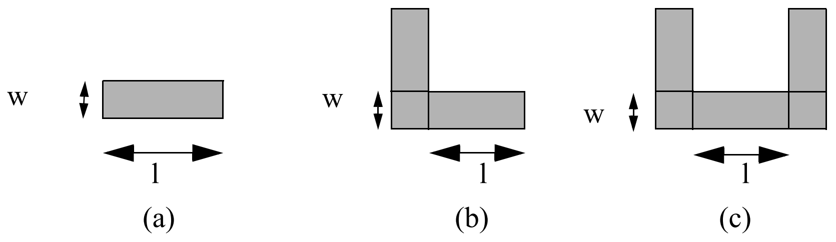


Figure 2-1. Beam Corner effects.
(a) joint_extension = 0 (no corners), (b) joint_extension = 1 (one corner), (c) joint_extension = 2 (two corners).

should lead to a more fine grained model of the beam, resulting in a more accurate model. However, as more NODAS beam segments were added to the NODAS schematic of this beam, the resonant frequency kept decreasing. The error was tracked down to the use of a variable that had inconsistent purposes.

In the initial version of the NODAS beam models, corner effects (where two beams meet at a corner) were modeled by fitting the NODAS simulation to a finite element simulation of a crab leg resonator [17]. The corner model was activated by using the *flag_shear* parameter, which is also used to turn on the shear beam model. The beam's length was increased by 30% of the beam's width when the sheer model was activated. However, there are cases when the shear model needs to be used without adding more length to the beam. The test simulation mentioned previously is one such case. The introduction of a new variable, *joint_extension*, was added to solve this problem. *joint_extension* can be set to 0 (default), 1 or 2. If it is set to 0, the length of the beam is not increased. If it is set to 1 or 2, 30% of the width (one corner) and 60% of the width (two corners) would be added to the length of the beam respectively (Figure 2-1). The Verilog-A implementation is shown in Listing 2-1. *beam_mech_lin* is the submodule name. The variables in the parentheses

```
beam_mech_lin # (.l(l+0.3*w*joint_extension), .w(w-overetch), .thick-
ness(thickness), .E(E), .density(density), .angle(angle),
.flag_shear(flag_shear),.z_air_gap(air_gap), .visc_air(visc_air),
.Poisson_ratio(Poisson_ratio), .Xc(Xc), .Yc(Yc))
    beam_mech_lin_1 (phia, phib, xa, xb);
```

Listing 2-1. Revised beam model with corner effects.

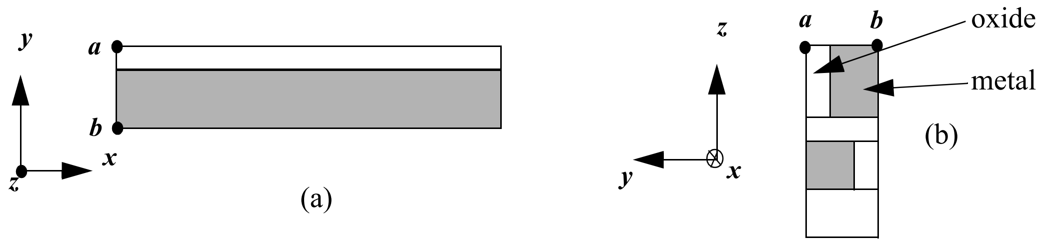


Figure 2-2. Thermal moment convention.
(a) top view, (b) cross section view

are used to overwrite the submodule's parameters. For instance, *.l(l+0.3*w*joint_extension)* means that the parameter *l* in the submodule *beam_mech_lin* should be overwritten with the value *l+0.3*w*joint_extension*. *beam_mech_lin_1* is the instance name and the variables in the parentheses are the signal wire names. These signal wires can be connected to other submodules. For instance, the beam module also instantiates the thermal beam model (*beam_thermal*) using the *phia* and *phib* signal wires.

2.2 Thermal Expansion

The difference between the thermal coefficient of expansion (TCE) of the metal and oxide layers in a CMOS beam will cause a beam to bend both laterally and out of plane. The 2D NODAS beam models capture the lateral in-plane bending produced by these thermal stresses. A convention for the calculation of the bending moments caused by the thermal expansion was chosen. Figure 2-2a shows the top view and Figure 2-2b shows the cross section view. The points *a* and *b* are the same points in both views. In order to derive the thermal bending moments, the location of the neutral axis must be calculated. The neutral axis is the axis in the cross section of the beam where there is no stress [31]. It is then used in the calculation of the moment generated by the thermal expansion of the beam layers. The moment arms are computed relative to the neutral axis.

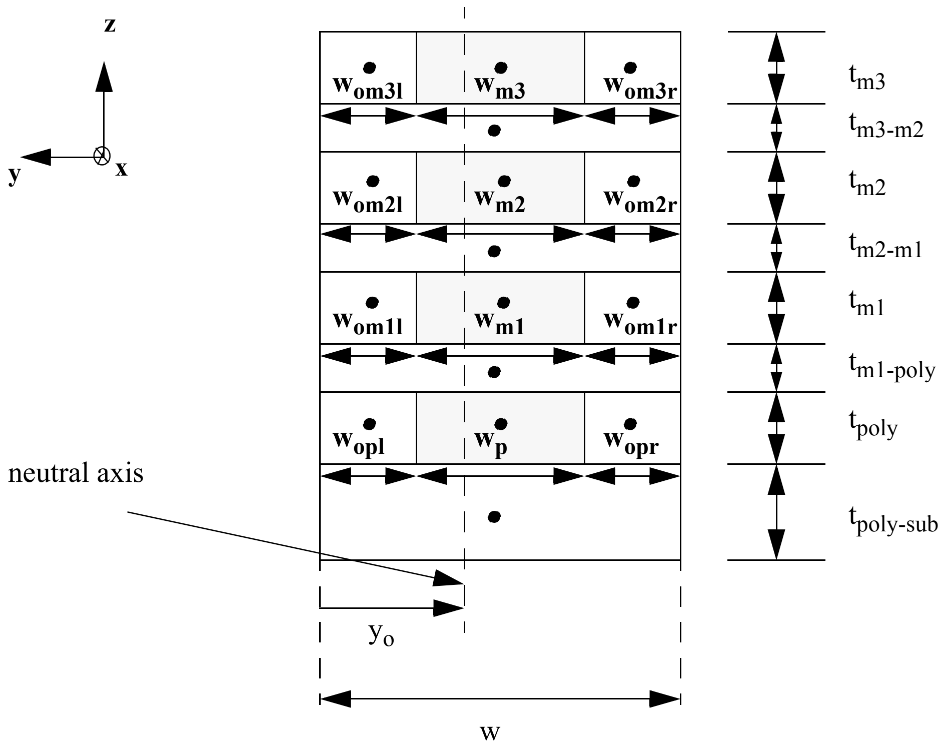


Figure 2-3. Neutral axis calculation model. M3, M2, M1, poly stack.
The black dots are the locations of the centroids for volume of material.

2.2.1 Neutral Axis Calculation

In order to determine the location of the neutral axis, a model of the CMOS beam must be created. Figure 2-3. shows the model for a metal 1, metal 2, metal 3, and poly stack. Sixteen stack combinations exist using these four layers. It is assumed that each metal layer could have oxide to the left, right, or both sides of the metal. If there is no oxide layer to the left and right of the metal, the widths w_{oil} and w_{oir} (the subscript i is the metal or poly layer) are set to 0 and the metal width is equal to w. Depending on which metal layers are chosen, the insulating oxide layer thickness between metal layers will vary. For instance, a metal 3 and poly stack will only have a single oxide layer between metal 3 and poly

The calculation for the location of the neutral axis is derived in [31]. (2.1) is the equation used to calculate the location of the neutral axis y_o. The summation is performed for each area in the cross section.

$$0 = \sum_i E_i \int_{A_i} y dA_i \tag{2.1}$$

Rearranging and solving for y_o, (2.1) can be rewritten as:

$$y_o = \frac{\sum_i E_i A_i y_{ci}}{\sum_i E_i A_i} \tag{2.2}$$

where y_{ci} is the location of the centroid of the i^{th} material section from the left edge of the cross section. E_i is the Young's modulus for the i^{th} material section (i.e. metal, poly, or oxide) and A_i is the area of the cross section for the i^{th} material section.

2.2.2 Moment Calculation

The moment due to the thermal coefficient of expansion for a multi-layer CMOS beam was derived in [18] [19]. For reference, the derivation presented in [19] is repeated here and modified for in-plane lateral bending.

M_{zi} (2.3) is the total moment about the z-axis produced by the interfacial forces of the i^{th} layer.

$$\sum_{i=1}^{n} P_i = 0; \left(\sum_{i=1}^{n} M_{zi} \right) = -(Y^T P) \tag{2.3}$$

where P is the force column vector and Y is the moment arm vector measured from the neutral axis. Since P_i is produced by action-reaction pairs, they sum to zero.

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_n \end{bmatrix}; \ \mathrm{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \tag{2.4}$$

where y_i is given by (2.5).

$$y_i = y_{ci} - y_o \tag{2.5}$$

If the thickness of the beam is assumed to be smaller than the radius of curvature (ρ), the radius of curvature can be assumed to be the same for all layers. Therefore, M_{zi} is:

$$M_{zi} = \frac{E_i I_{zi}}{\rho} \text{ where, } \mathrm{I}_{zi} = \frac{w_i h_i^3}{12} \tag{2.6}$$

I_{zi} is the moment of inertia of the ith layer. w_i is the width and h_i is the thickness of the ith layer. Let T_0 be the temperature at which the beam is flat (at the characteristic temperature).

Equating the strains at the interfaces between the layers due to a temperature change, $\Delta T = T - T_0$, the following relation results:

$$\frac{P_{i+1}}{E_{i+1}A_{i+1}} - \frac{P_i}{E_i A_i} + \Delta T(\alpha_{i+1} - \alpha_i) - \frac{y_i - y_{i+1}}{\rho} = 0 \tag{2.7}$$

By noting the uniformity of the subscript *i*, (2.7) can be rewritten as:

$$\frac{P_i}{E_i A_i} + \Delta T \alpha_i - \frac{y_i}{\rho} = C \tag{2.8}$$

where C is the uniform axial strain for all layers. α_i is the TCE for the *i*th layer. Multiplying by $E_i A_i y_i$, (2.8) becomes:

$$P_i y_i + E_i A_i y_i \Delta T \alpha_i - \frac{E_i A_i y_i^2}{\rho} = C(E_i A_i y_i) \tag{2.9}$$

Summing (2.9) over all layers and using (2.3) and (2.6), (2.10) is obtained.

$$-\sum_i \frac{E_i}{\rho}(I_{zi} + A_i y_i^2) + \sum_i E_i A_i y_i \Delta T \alpha_i = C \sum_i E_i A_i y_i \tag{2.10}$$

The first term of (2.10) contains the parallel axis theorem for computing moments and the right hand side reduces to zero. The total bending moment on a composite beam is obtained as (2.11).

$$M_z = \sum_i y_i w_i h_i E_i \Delta T \alpha_i \tag{2.11}$$

The model can be further extended by adding a residual stress term (σ_{ri}) for each layer (2.12).

$$M_z = \sum_i y_i w_i h_i (E_i \Delta T \alpha_i + \sigma_{ri}) \tag{2.12}$$

2.3 Thermal Conduction

To model electrothermal heating of a beam, a thermal conduction and an electrothermal model were developed. The thermal conductance of a material is given by (2.13)

$$G_T = \frac{Q}{\Delta T} = \kappa \frac{hw}{l} \tag{2.13}$$

where Q is the heat conduction rate (Watts/second), ΔT temperature difference between the two ends of the material, h is the thickness of the material, w is the width, l is the length, and κ is the thermal conductivity (with units of Watts/K-m). Thermal conductance is an analog of electrical conduction. Q and T are analogous to I and V. Similar to electrical conduction, thermal conductances in parallel add together. For a multi-layer CMOS beam, the total thermal conductance is the sum of the thermal conductances of each layer.

In a CMOS process, the high sheet resistance of polysilicon can be exploited to make a heating resistor. The power dissipated by a resistor is given by (2.14).

$$P_R = \frac{V^2}{R} \tag{2.14}$$

where V is the voltage difference between the two terminals of the resistor and R is the resistance. The power dissipated in the poly resistor generates heat and raises the temperature of the CMOS beam. By applying the conservation of power, the heat conduction rate flowing through the beam, Q, is equal to the power dissipated by the poly resistor, P_R. ΔT can be solved to find the change in

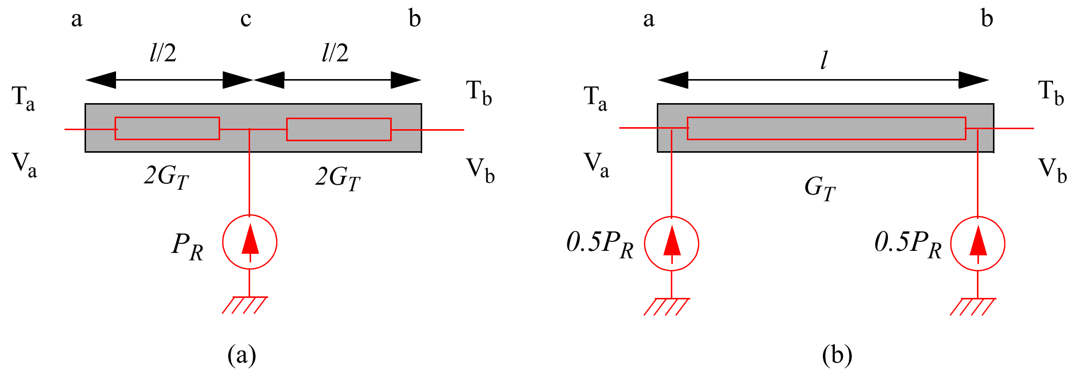


Figure 2-4. Electrothermal conduction models.
(a) T-model. The thermal power generated by the poly resistor is fed into node c, an internal node. The beam with length *l* is split into two equal segments of length *l*/2. The effective thermal conductance is twice the thermal conductance (G_T) of the single beam with length *l*. (b) π-model. Half of the thermal power generated by the poly resistor is fed into node a and the other half is fed into node b. The T-model is used in NODAS.

temperature of the beam. A T-model (Figure 2-4a) or a π-model (Figure 2-4b) can be chosen to lump the electrothermal heating for beam temperature computation. In the T-model, the beam is split into two equal lengths (*l*/2) and a new thermal node (node c) is created. The power generated by the resistor is fed into this node. The effective thermal conductance of the two beam segments is twice the thermal conductance ($2G_T$) of the original beam (G_T). In the π-model, the power generated by the poly resistor is split equally between nodes a and b. Though the T-model introduces an extra internal node, it was implemented in NODAS because it is more physically accurate for fewer beams. Consider the case when T_a and T_b are fixed to a common heat sink of temperature T_0. A voltage is also placed across the beam between V_a and V_b. For the T-model, T_c will be at a higher temperature than T_a and T_b and heat will flow from node c to nodes a and b. In contrast, the since $\Delta T = 0$ in the π-model, no heat flows through the beam. Both models produce the correct results, the T-model is more physically accurate.

The NODAS implementation of electrothermal conduction is shown in Listing 2-2. *flag_poly* is a parameter which is 1 when there is a poly layer and 0 where there is no poly layer. *sheet_resistance_poly* is the sheet resistance of the poly layer. *l* and *pw* are the length and width of the poly layer. *Rpoly* is the resistance of the poly layer. *Ppoly* is the electrical power flowing across

```
@ (initial_step) begin
   if (flag_poly==1) begin
      Rpoly = sheet_resistance_poly * l/pw;
   end
   else begin
      Rpoly = 0;
   end
end

if (flag_poly==1) begin
   Ppoly = pow(deltaV,2)/Rpoly;
end
else begin
   Ppoly = 0;
end

// assign the thermal power through the beam
Pwr(tc,ta) <+ 2*Geff*(Temp(tc) - Temp(ta));
Pwr(tc,tb) <+ 2*Geff*(Temp(tc) - Temp(tb));

// assign the thermal power due to the poly resistor
Pwr(tc) <+ -Ppoly;
```

Listing 2-2. Electrothermal conduction.

the poly resistor. It is only calculated when there is a poly layer. Next, the thermal power flowing through the beam resulting from a difference in temperature between the two ends of the beam (nodes *a* and *b*) is calculated. Since the T-model was used, the power is summed at the internal node *c*. Finally, the power generated by the poly resistor is added to node *c*. Since the two previous lines calculated the power leaving node *c*, *Ppoly* was made negative to make the power leave node *c* to ground (Kirchhoff's current law). *Pwr(tc, ta)* is the power flowing from *tc* to *ta*. *Pwr(tc)* is the power flowing into *tc*.

2.4 Verification

2.4.1 Fixed-Fixed and Fixed-Guided beams

With the corrections made to the joint model, the accuracy of the beam model could be compared to analytic equations. A fixed-fixed beam with a 50 nN uniform distributed load (Figure

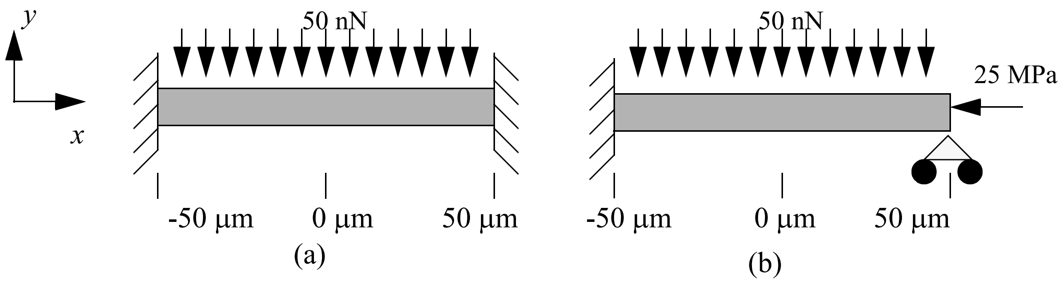


Figure 2-5. Beam verification problems.
(a) Fixed-fixed beam with 50 nN uniform distributed load. (b) Fixed-guided beam with 50 nN uniform distributed load and 20 MPa compressive axial stress. Both beams are 100µm long, by 2µm wide, by 2µm thick. The Young's modulus was set to 170 GPa and the density was set to 2330 kg/m^3.

2-5a) and a fixed-guided beam with a 50 nN uniform distributed load and 25 MPa compressive stress (Figure 2-5b) were chosen to test the accuracy of the model.

Figure 2-6 shows the results of the simulations. The *x* axis is the position along the beam and the *y* axis is the percent difference between the analytic equation and the NODAS simulation results (2.15). The 2, 4, 8, 16, and 32 NODAS 2D nonlinear beam segments were used to model the 100 µm long beam. For the fixed-fixed case (Figure 2-6a), the accuracy is greater than 6.1x10^{-6}% and for the fixed-guided case (Figure 2-6b), the accuracy is greater than 15x10^{-3}%.

$$\%\ difference = 100\left(\frac{y_{analytic} - y_{NODAS}}{y_{analytic}}\right) \tag{2.15}$$

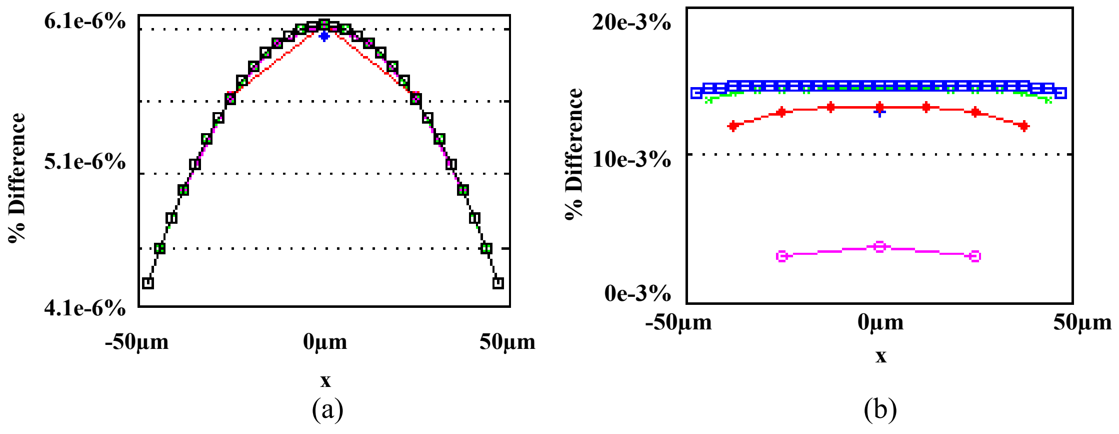


Figure 2-6. Beam verification results.
(a) y displacement of a fixed-fixed beam with a 50nN uniform distributed load. NODAS vs. Analytic. %Difference = 100*(1 - $y_{NODAS}/y_{ANALYTIC}$). 2, 4, 8, 16, and 32 beam segments plotted. (b) y displacement of a fixed-guided beam with 50nN uniform distributed load and 25MPa compressive stress. NODAS vs. Analytic. %Difference = 100*(1 - $y_{NODAS}/y_{ANALYTIC}$). 2, 4, 8, 16, and 32 beam segments plotted. Both beams are 100µm long, by 2µm wide, by 2µm thick. The Young's modulus was set to 170 GPa and the density was set to 2330 kg/m^3.

2.4.2 Multimorph Beam

The NODAS thermal expansion model was compared with an ANSYS simulation. The ANSYS model can be found in Appendix C.2. It uses the SOLID45 brick element. The NODAS schematic testbed uses a single NODAS linear beam. In both simulations, the beam is anchored on one end and free on the other end. The width of the beam was 10 µm and the length was 100 µm. The Young's Modulus for the oxide and metal were assumed to be both 62GPa for simulation purposes only. The thermal coefficient of expansion for metal and oxide were set to 23 µm/K and 8.1 µm/K for simulation purposes only. The reference temperature T_0 (T_{ref} in ANSYS) was set to 294K and the simulation temperature was set to 350K. Finally, Poisson's ratio was set to 0.3. The results of the simulation are presented in Table 2-1. The percent difference is calculated as *100*(ANSYS-NODAS)/ANSYS*, where NODAS and ANSYS are the tip deflections in the *y* direction reported by both simulations. As can be seen, the accuracy is within 2%. Adding more NODAS beam segments does not increase the accuracy of the simulations.

Table 2-1. Multimorph simulation comparison. NODAS versus ANSYS for different metal stack combinations. ANSYS simulations were run on a 2GHz Pentium 4 computer with 1GB of RAM. NODAS simulations were run on a dual 1GHz UltraSparc III Sun-Fire 280R server with 4GB of RAM.

Stack	% Difference	NODAS Sim Time (s)	ANSYS Sim time (s)
M3, M2, M1	1.08	0.02	120
M3, M2	1.02	0.01	69
M3, M1	0.60	0.01	69
M3	1.46	0	33
M2, M1	1.21	0.01	48
M2	0.89	0.01	27
M1	1.65	0	42

2.4.3 Electrothermal heating

A polysilicon resistor subjected to electrothermal heating was simulated in NODAS and in ANSYS. A 100 μm long, 5 μm wide, and 0.28 μm thick poly resistor was connected to a 300K temperature source on one end and a 500K temperature source on the other end. 5V was placed across the resistor. The thermal conductivity of poly was set to 30 W/K-m and the sheet resistance of poly was set to 100 ohms/square. The resulting temperature profile across the length of the poly resistor is shown in Figure 2-7. In this plot, 2, 4, 8, 16 and 32 NODAS beam elements were used. The ANSYS simulation uses the SOLID5 brick element. The NODAS results are within 1.4% of the ANSYS results for all points along the poly resistor's length. As can be seen, increasing the number of beam segments does not increase the accuracy of the results, but it does provide a more fine grained temperature distribution. The ANSYS model can be found in Appendix C.3.

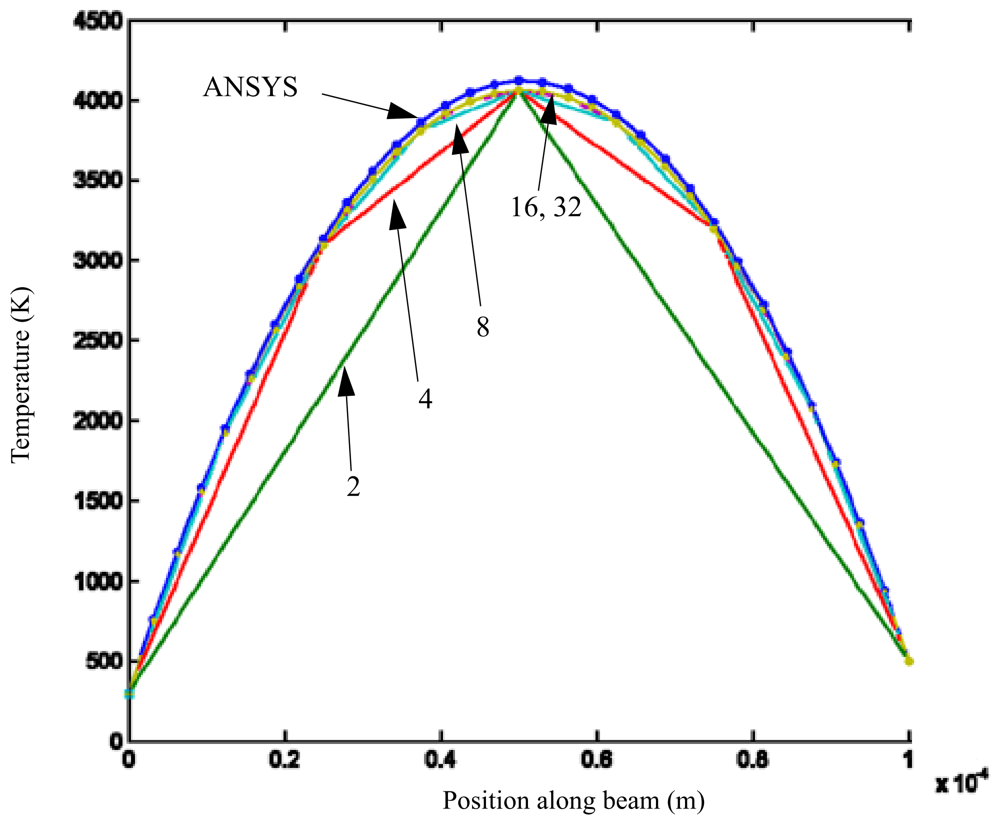


Figure 2-7. Temperature profile of a poly resistor.
Temperature versus position along a poly resistor. 2, 4, 8, 16, and 32 NODAS beam segments were used for this simulation. The left end of the poly resistor was connected to a 300K temperature source and the right end was connected to a 500K temperature source. 5V was placed across the poly resistor. The sheet resistance of poly was set to 100 ohms/square, the thermal conductivity of poly was set to 30 W/K-m. The length, width and thickness were set to 100 μm, 5 μm, and 0.28 μm respectively. The ANSYS temperature profile was a path through the center of the poly resistor running from y = 0 μm to 100 μm.

Note: Areal poly resistor would melt at these high temperatures. The data presented here is used to verify simulation accuracy only.

3 Electrostatic Gap Modeling and Verification

In MEMS devices such as mixers, filters, and resonators, electrical signals are used to generate mechanical motion. These electrical signals are coupled to the mechanical devices by electrostatic forces generated by the electric fields. The electrostatic gap model was created to model these forces. The modeling and verification of the NODAS electrostatic gap model is discussed in this chapter. Since the electrostatic force is a distributed phenomena, but the NODAS models are lumped parameter models, the distributed forces must be lumped to the nodes of the model. The first section of this chapter discusses the process of lumping a distributed forces (and moments) to the nodes of a model. The electrostatic force modeling is then discussed, followed by a discussion of the damping and contact models. A rotated parallel plate approximation is used to simplify the electrostatic force calculation. As demonstrated in Section 3.12, the error introduced in this approximation can be reduced by adding more beam and gap elements. Section 3.3.1 discusses the derivation of this approximation. Next, lateral electrostatic forces, electrical modeling, and the electric only gap model is discussed. Finally, the verification of the electrostatic gap model is presented.

3.1 Lumping

When analyzing physical systems, distributed forces (electrostatic forces, gravity, damping, etc.) are typically encountered. The distributed forces do not need to be treated differently if analytic equations are solved by hand, however, when using a technique such as nodal analysis, where the only points of communication between the outside world and the model is through the nodes on the model, the distributed forces must be mapped onto the nodes of the model. For

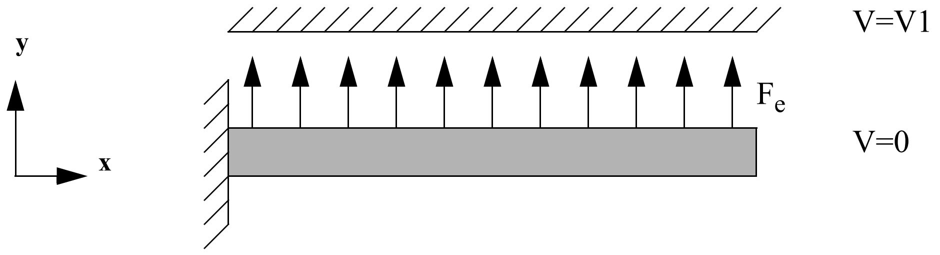


Figure 3-1. Distributed electrostatic force on a MEMS beam.

instance, in the electrostatic gap model, a distributed force arises due to an electric field generated by a potential difference between two beam electrodes (Figure 3-1).

Each degree of freedom in the NODAS gap model has two nodes: *a* and *b* (one for each end of the beam, Figure 3-2). For instance, *xa_t<0>* is the *x* displacement pin for the top beam electrode whereas *xb_b<1>* is the *y* displacement pin for the bottom beam electrode. *_t* and *_b* means top and bottom respectively. The effect of the distributed forces needs to be modeled at these nodes while maintaining accuracy. In the following derivation, the lumping of the forces is considered for one of the beams.

In Section 2.3, it was shown that the power generated by the poly resistor could be lumped to the center of the beam (with a new internal node) or to the ends of the beams. The lumping in this case was straightforward: either all of the power was lumped to the center of the beam or half the power was lumped to each end of the beam. The lumping discussed in this section has one major difference. Since beams can bend, in order to lump the forces and moments to the beam nodes, the

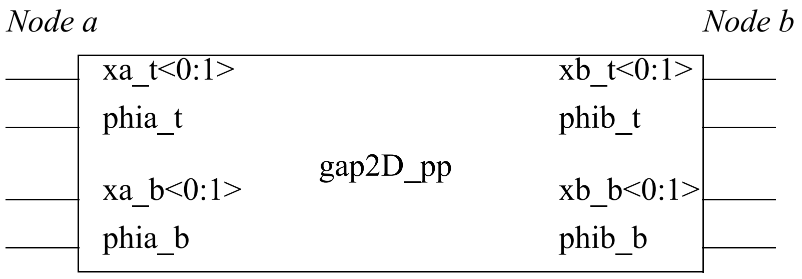


Figure 3-2. NODAS Electrostatic Gap Symbol.

beam's shape must be taken into account. The beam's shape function (also called a basis function) is used as a weighting function to the determine how much of the force or moment should be assigned to each node of the beam.

3.1.1 Lumping Distributed Forces

The derivation of the lumping of distributed forces can be found on pages 161 - 162 of [32]. Starting with the definition of virtual work, the work done by displacements caused by the distributed forces (δW in (3.1)) can be equated to the work done by the lumped forces (right hand side of (3.1)).

$$\delta W = \int_S \delta u^T \Phi dS = \delta U^T P_{equivalent} \tag{3.1}$$

where Φ is the matrix of surface forces. $P_{equivalent}$ is a vector of the lumped forces. δU is the virtual displacements in the direction of the forces, and δu is the distribution of virtual displacements.

$$u = aU \tag{3.2}$$

where a is the vector of basis functions. A complete list of basis functions can be found on page 293 of [32]. U is the vector of discrete displacements in the direction of the forces and u is the vector function of interior (distributed) displacements. In this case, if the electrostatic force acting on the beam electrodes in the y direction, u is the function $y(\xi)$ and U is a column vector given in (3.3), which represents the lumped displacements at the nodes of the beam.

$$\begin{bmatrix} y_a \\ \theta_a \\ y_b \\ \theta_b \end{bmatrix} \tag{3.3}$$

For example, for a 2-D beam of length L displaced in the y direction, a is given by (3.4) - (3.7), where $\xi = \frac{x}{L}$ (the normalized position along the beam's length).

$$a_1(\xi) = 1 - 3\xi^2 + 2\xi^3 \tag{3.4}$$

$$a_2(\xi) = (\xi - 2\xi^2 + \xi^3)L \tag{3.5}$$

$$a_3(\xi) = 3\xi^2 - 2\xi^3 \tag{3.6}$$

$$a_4(\xi) = (-\xi^2 + \xi^3)L \tag{3.7}$$

From (3.2), it follows that

$$\delta u = a\delta U \tag{3.8}$$

Substituting (3.8) into (3.1) we get

$$\delta U^T\left(\int_S a^T \Phi dS - P_{equivalent}\right) = 0 \tag{3.9}$$

Since the virtual displacements are arbitrary, it follows that

$$P_{equivalent} = \int_S a^T \Phi dS \tag{3.10}$$

Therefore, for a distributed force *q(x)* acting in the y direction, (3.11) is the equation for energy conservation:

$$\delta W = \int_S \delta u^T q(x) dx = \delta U^T P_{equivalent} \tag{3.11}$$

Converting to the normalized coordinates, the lumped forces and moments at the nodes are then given by (3.12) - (3.15). In this case, *q(x)* is the force per unit length.

$$F_a = L\int_0^1 a_1(\xi) q(\xi) d\xi \tag{3.12}$$

$$M_a = L\int_0^1 a_2(\xi) q(\xi) d\xi \tag{3.13}$$

$$F_b = L\int_0^1 a_3(\xi)q(\xi)d\xi \tag{3.14}$$

$$M_b = L\int_0^1 a_4(\xi)q(\xi)d\xi \tag{3.15}$$

A point force can be modeled as an infinite pressure acting over zero area such that product ΦdS is equal to the point force. For instance, consider (3.12), where $q(\xi)$ is a point force F_0 at some point ξ_0, the concentrated load F_a would be $a_1(\xi_0)*F_0$.

A further generalization for a combination of distributed force $q(x)$ acting in the y direction, distributed moment $r(x)$ acting around the z axis, N concentrated loads F_i acting at position ξ_i (i=1..N) and K concentrated moments M_i acting at position ξ_i (i=1..K) can be made. The corresponding lumped forces and moments at the nodes are given by (3.16) - (3.19).

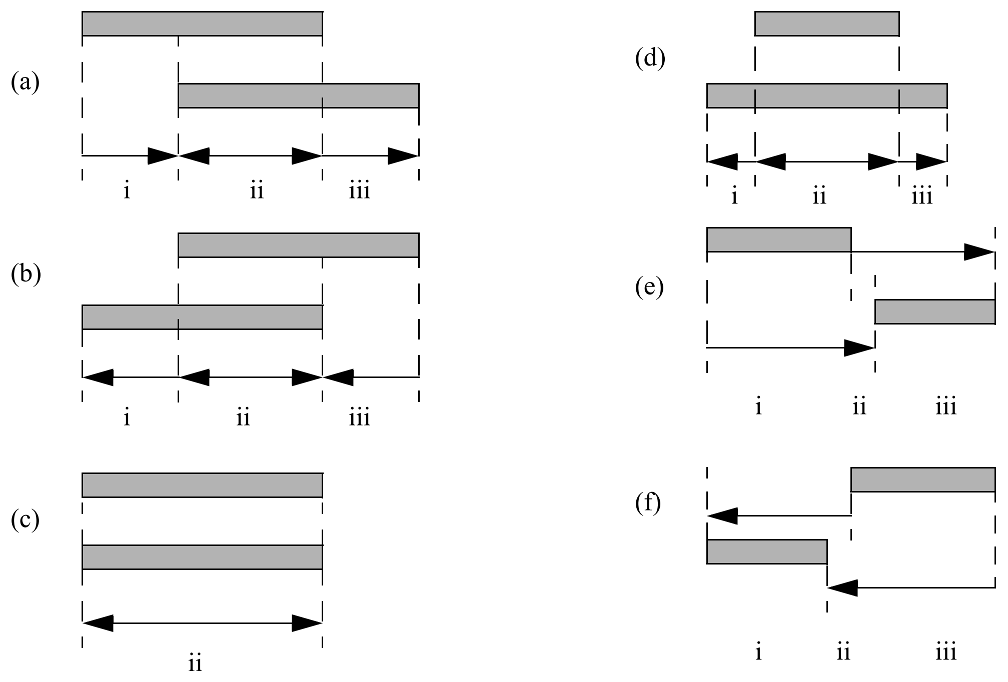
$$F_a = L\int_0^1 a_1(\xi)q(\xi)d\xi + \sum_{i=1}^{N} a_1(\xi_i)F_i + \int_0^1 a_1'(\xi)r(\xi)d\xi + \frac{1}{L}\sum_{i=1}^{K} a_1'(\xi_i)M_i \tag{3.16}$$

$$M_a = L\int_0^1 a_2(\xi)q(\xi)d\xi + \sum_{i=1}^{N} a_2(\xi_i)F_i + \int_0^1 a_2'(\xi)r(\xi)d\xi + \frac{1}{L}\sum_{i=1}^{K} a_2'(\xi_i)M_i \tag{3.17}$$

$$F_b = L\int_0^1 a_3(\xi)q(\xi)d\xi + \sum_{i=1}^{N} a_3(\xi_i)F_i + \int_0^1 a_3'(\xi)r(\xi)d\xi + \frac{1}{L}\sum_{i=1}^{K} a_3'(\xi_i)M_i \tag{3.18}$$

$$M_b = L\int_0^1 a_4(\xi)q(\xi)d\xi + \sum_{i=1}^{N} a_4(\xi_i)F_i + \int_0^1 a_4'(\xi)r(\xi)d\xi + \frac{1}{L}\sum_{i=1}^{K} a_4'(\xi_i)M_i \tag{3.19}$$

The notation $a_i' = da_i/d\xi$. The factors of L and $1/L$ in (3.16) to (3.19) come from the switch of the x coordinates to the normalized ξ coordinates.



i) bottom_electrode_offset
ii) overlap region
iii) offsetL2Right

Figure 3-3. Electrode configurations.

3.2 Gap Topology

In the original implementation of the gap model, a static variable called *topology* was used to describe the configuration of the beam electrodes. When *topology* = 1, the top beam was to the left of the bottom beam (Figure 3-3a) and when *topology* = 0, the bottom beam was to the left of the top beam (Figure 3-3b). This method of describing the beam electrode configuration was limited in that the beams must be overlapped and the cannot switch their topologies (i.e. *topology* = 1 becomes *topology* = 0). On the other hand, to support composable simulation, the gap may appear between any parallel plate configuration. Figure 3-3 shows a few possible gap configurations for two beam electrodes (top and bottom beams).

To solve this problem this thesis first determines gap topology from the layout configuration, and then updates the gap topology dynamically. This dynamic topology replaces the static

```
@ (initial_step) begin
    offsetL2Left = bottom_electrode_offset;
    magoffsetL2Right=finger_l_t+finger_l_b-overlap-
        (abs(offsetL2Left)+overlap);

    if (offsetL2Left >= 0) begin
        if (finger_l_t == overlap) begin
           offsetL2Right = magoffsetL2Right;
        end
        else if (finger_l_b == overlap) begin
           offsetL2Right = -magoffsetL2Right;
        end
        else if (overlap > 0) begin
           offsetL2Right = magoffsetL2Right;
        end
        else begin
           offsetL2Right = offsetL2Left+finger_l_b-finger_l_t;
        end
    end
    else begin
        if (finger_l_t == overlap) begin
           offsetL2Right = magoffsetL2Right;
        end
        else if (finger_l_b == overlap) begin
           offsetL2Right = -magoffsetL2Right;
        end
        else if (overlap > 0) begin
           offsetL2Right = -magoffsetL2Right;
        end
        else begin
          offsetL2Right = -(abs(offsetL2Left)+finger_l_t-finger_l_b);
        end
    end
end
```

Listing 3-1. Static (layout) beam offset calculations.

topology used in the earlier model. In the Verilog-A implementation, the top and bottom beams are renamed as beam 1 and beam 2. Four user parameters are sufficient to fully describe the relative positions of the beam electrodes (*bottom_electrode_offset*, *overlap*, *finger_l_t*, and *finger_l_b*). *bottom_electrode_offset* is the distance from the top beam's left edge to the bottom beam's left edge. It is denoted by **i** in Figure 3-3. *overlap* (**ii** in Figure 3-3) is the region where the two beams are overlapped. If there is no overlap (Figure 3-3e,f), *overlap* is set to 0. *finger_l_t* is the top electrode length, and *finger_l_b* is the bottom electrode length. The variable *offsetL2Right* can be calculated from these four parameters (Listing 3-1).

```
dynoffsetL2Left = offsetL2Left + (-xm1_l+xm2_l);
dynoffsetL2Right = offsetL2Right + (-xp1_l+xp2_l);

dynOverlap = (L1-abs(dynoffsetL2Left)+L2-abs(dynoffsetL2Right))/2.0;
if (dynOverlap > 0) begin
    ov = dynOverlap;
end
else begin
    // there's no overlap
    ov = 0;
end
```

Listing 3-2. Dynamic beam offset calculations.

Since the static beam configuration does not change, it is placed in an *@initial* block. Using the displacement applied at the nodes, the new beam configuration can be calculated (Listing 3-2).

L1 and *L2* are the top beam length and bottom beam lengths respectively. *xm1_l* is the x displacement on the left side of the top electrode. *xm2_l* is the x displacement on the left side of the bottom electrode. *xp1_l* is the x displacement on the right side of the top electrode. *xp2_l* is the x displacement on the right side of the bottom electrode. *m* and *p* are the left and right nodes of the beams respectively. *1* and *2* indicate whether the variable is used for the top or bottom beam respectively. *_l* indicates that the variables are local frame variables (Section 3.8). If the top beam moves left or the bottom beam moves right, the dynamic offsets *dynoffsetL2Left* and *dynoffsetL2Right* increase. Similarly, if the top beam moves right or the bottom beam moves left, the dynamic offsets *dynoffsetL2Left* and *dynoffsetL2Right* decrease. The first two lines of Listing 3-2 model this behavior. The dynamic overlap, *dynOverlap*, is calculated by subtracting the dynamic left and right offsets from the top and bottom beams respectively and taking the average. If the dynamic overlap is less than zero, the overlap variable, *ov*, is set to zero, otherwise it is set to *dynOverlap*.

Once the dynamic offsets and overlap have been calculated, the endpoints of the overlap region can be computed (Listing 3-3). These end points are needed for the lumping of the electrostatic forces. The variables *x1l*, *x2l*, *x1r*, and *x2r* are the x positions of the top beam left overlap

```
if (dynoffsetL2Left >=0) begin
        x1l = dynoffsetL2Left;
        x2l = 0;
        if (dynoffsetL2Right >= 0) begin
            x1r = L1;
            x2r = L2-dynoffsetL2Right;
        end
        else begin
            x1r = L1 + dynoffsetL2Right;
            x2r = L2;
        end
    end
    else begin
        x1l = 0;
        x2l = -dynoffsetL2Left;
        if (dynoffsetL2Right >= 0) begin
            x1r = L1;
            x2r = L2-dynoffsetL2Right;
        end
        else begin
            x1r = L1 + dynoffsetL2Right;
            x2r = L2;
        end
    end
```

Listing 3-3. Overlap region endpoint computation.

region endpoint, bottom beam left overlap region endpoint, top beam right overlap region endpoint and bottom beam right overlap region endpoint respectively.

The model must then calculate the electrostatic forces and lump the forces to the correct nodes. The lumping integrals, (3.12)-(3.15), are performed over the whole beam length, however, since the electrostatic force only exists in the overlap region (between *x1l* and *x1r*, *x2l* and *x2r*), the lumping integral only needs to be calculated between these points.

3.3 Electrostatic Forces

When a voltage is applied across the beam electrodes, the electrostatic force causes the beams to bend. The calculation of the electric field in the overlap region does not have an analytic solution. Thus the approximation that the beam electrodes are rigid rotated parallel plates is made. Though the approximation seems crude, it will be shown that accuracy of up to 2% can be achieved by discretizing the problem into more beam and gap elements.

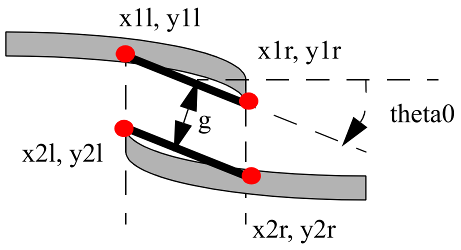


Figure 3-4. Rotated parallel plate approximation.

3.3.1 Rotated Parallel Plate Approximation

A rotated parallel plate approximation is made in the overlap region (Figure 3-4). In order to calculate the equivalent parallel plate representation, the gap, *g*, and the angle of rotation, *theta0*, must be calculated. To find the rotation angle *theta0*, the angle of the top and bottom beams in the center of the overlap region is averaged. The angle is found by taking the derivative of the beam basis function and evaluating it a a point along the *x* axis. Using the variables *x1l*, *x1r*, *x2l*, *x2r* and the beam shape functions, the y displacement of the endpoints of the overlap region (*y1l*, *y1r*, *y2l*, *y2r*) can be calculated. Listing 3-4 shows the Verilog-A implementation of these calculations. The function *topBeamY* calculates the *y* displacement of the beam at a point *x,* where *x* is the last parameter of the function. Similarly, the function *topBeamAngle* calculates the angle of the beam at a

```
y1l = topBeamY(ym1, yp1, am1, ap1, L1, x1l);
y1r = topBeamY(ym1, yp1, am1, ap1, L1, x1r);
y2l = topBeamY(ym2, yp2, am2, ap2, L2, x2l);
y2r = topBeamY(ym2, yp2, am2, ap2, L2, x2r);

// get the angle at the midpoint
ang1mid = (topBeamAngle(ym1,yp1,am1,ap1,L1,(x1l+x1r)/2.0));
ang2mid = (topBeamAngle(ym2,yp2,am2,ap2,L2,(x2l+x2r)/2.0));

// the average angle of the beams
theta0 = (ang1mid + ang2mid)/2.0;

// find the endpoint which has the smallest gap
dy_l = min((y1r-y2r+gap),(y1l-y2l+gap));
dy = dy_l*cos(theta0);
```

Listing 3-4. theta0 and gap calculation.

```
v = (V(va_t,va_b)+V(vb_t,vb_b))/2.0;
v_squared = pow(v,2);

Felec = -0.5*`eps0*thickness*v_squared/pow(dy,2);
```
Listing 3-5. Electrostatic force per unit length.

point *x*. The gap in the local frame (*g* in Figure 3-4, *dy_l* in the Verilog-A code) is chosen to be the minimum of the gap at the left and right side of the overlap region. Since the gap is calculated in the local frame, the rotated parallel plate gap (*dy*) is calculated by multiplying *dy_l* by *cos(theta0)*. See Section 3.8 for an explanation of the frames of reference.

3.3.2 Electrostatic Force Equation

By making a rotated parallel plate approximation, the electrostatic force between the two beam electrodes is simply the electric force generated by a parallel plate capacitor (3.20). *l* is the length of the overlapped region (Figure 3-3), *h* is the thickness of the beam electrode, *V* is the voltage across the beam electrodes, *g* is the distance between the two beam electrodes, and ε_o is the permittivity of free space.

$$F_{elec} = \frac{\varepsilon_o l h V^2}{2g^2} \tag{3.20}$$

The Verilog-A implementation for the parallel plate electrostatic force is shown in Listing 3-5. Since the force must be lumped to the nodes of the gap model, the electrical force per unit length is calculated here and will be used in the lumping integral (Section 3.6).

In Listing 3-5, the voltage between the two beams, *v*, is approximated as the average of the voltage at the left and right ends of the beam. *thickness* is the thickness of the beams (*w* in (3.20)) and *dy* is the gap between the two beams (*g* in (3.20)).

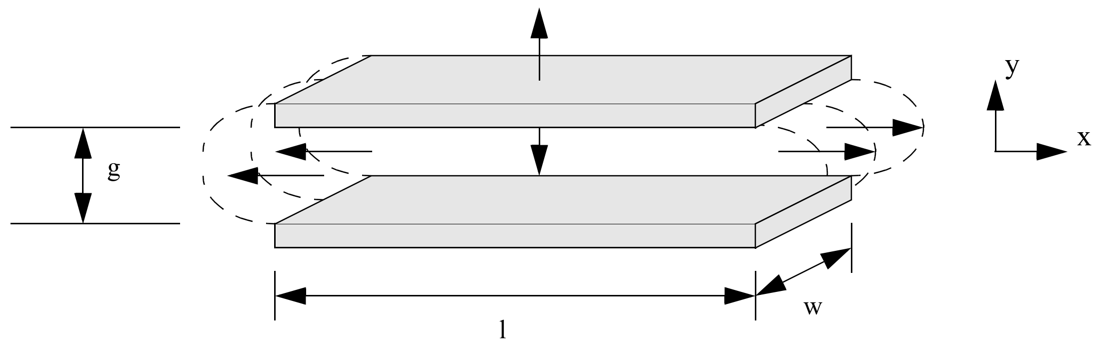


Figure 3-5. Squeeze film damping.

3.4 Damping

Two types of damping are modeled in NODAS: Couette and squeeze-film damping. The damping caused by the two beams sliding past each other while keeping the gap constant (in the *x* direction for Figure 3-5) is called Couette damping. Since the beam model has sufficient information to calculate the Couette damping force, it handled in the beam models. Squeeze film damping occurs when there is a changing gap between two beams (or two plates). When the gap decreases, the air between the two beams squeezes out and when the gap increases, air fills the new volume between the two beams. Since squeeze-film damping requires a gap, it is handled in the gap model. In the case of the gap model, a squeeze film damping force is generated when the beams move in the *y* direction (Figure 3-5).

The equation for squeeze film damping is given in (3.21).

$$F_{damp} = \frac{K\mu h l^3}{g^3}\frac{dy}{dt} \tag{3.21}$$

where *h* is the beam thickness and *l* is the beam length. *g* is the instantaneous gap between the two beam electrodes (*dy* in the gap model Verilog-A code). $\frac{dy}{dt}$ is the velocity of the beam. μ is the viscosity of air and *K* is a factor which depends on the geometry of the beam. For a beam with *w* = *l*, *K*=0.42. The Verilog-A implementation of (3.21) is shown in Listing 3-6. *Kdamp* is the factor *K*, *ov* is the length of the overlap (*l* in (3.21)), and *gap* is the gap (*g* in (3.21)). *thickness* is the thickness

```
Pos(V_dy) <+ ddt(dy);
Fy_damping = Kdamp*visc_air*ov*ov*thickness/pow(dy,3)*Pos(V_dy);
```

Listing 3-6. Squeeze film damping force per unit length.

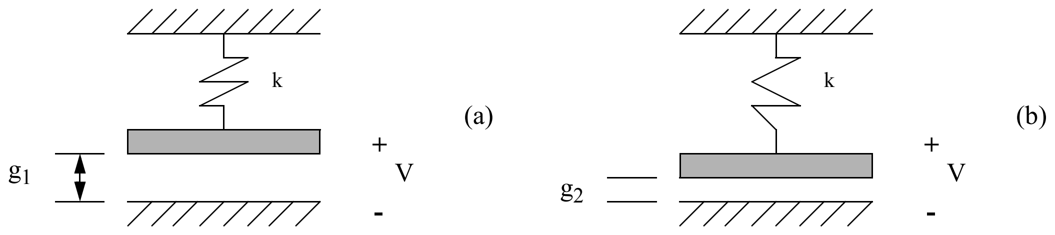


Figure 3-6. Canonical gap problem.
(a) V=0V, (b) Increased voltage.

of the beam (w in (3.21)). *ddt(dy)* is the time derivative of the gap (i.e. the velocity). Since the forces must be lumped to the ends of the beams, the force per unit length is used for *Fy_damping*. Therefore l^3 (ov^3) becomes l^2 (ov^2).

3.5 Contact Model

Figure 3-6 shows a canonical gap problem. The top beam electrode is connected to a spring whereas the bottom beam electrode is anchored. As the voltage between the beams increases, the gap between the two beams decreases. If sufficient voltage is applied, the beams may eventually come into contact with each other. When two-thirds of the original gap remains, the system becomes unstable and the top beam snaps in and contacts the bottom beam.

The electrostatic force is proportional to the gap squared (3.20). As the voltage increases, the force versus gap curve moves away from the origin (Figure 3-7a). Assuming a linear contact spring model [17], use of the $1/g^2$ relationship produces valid solutions in both the first and second quadrants, however, the only valid solutions are in the first quadrant (positive valued gaps). In order to prevent the simulator from finding the negative valued gap solution (second quadrant), a contact model was developed. The initial model is shown in Figure 3-7a [17]. It was a linear spring constant which was a function of the gap. The figure shows the load line plot of the contact model against the electrostatic force. As can be seen, valid stable first quadrant solutions exist in the lowest curves (solid line, far right most circles). Two other solutions exist in this case (middle circle and far left circle), however, the simulator will find the stable solution in the first quadrant. When the voltage is further increased, there are no solutions in the first quadrant (dashed curve, square), the simulator will only find the non-physical solution in the second quadrant.

In order to force the simulator to find a solution in the first quadrant, an improved contact model was developed (Figure 3-7b). First, the electrostatic force was capped at a maximum value, $F_{e,max}$, calculated from value of a maximum electric field, a user-defined process parameter. (3.22) shows the equation for the maximum electrostatic force. The quantity $\left(\frac{V}{g}\right)$ is the magnitude of the electric field across the gap (assuming a parallel plate configuration with no fringing electric fields). Therefore, $\left(\frac{V}{g}\right)_{max}$ can be considered the maximum electrical field, E_{max}. In the Verilog-A implementation, E_{max} is called *elec_breakdown*.

$$F_{e,\,max} = \frac{1}{2}\varepsilon_o hl\left(\frac{V}{g}\right)^2_{max} = \frac{1}{2}\varepsilon_o hlE^2_{max} \tag{3.22}$$

Next, minimum gap parameters, g_{min} and $g_{contact}$, were created. g_{min} is the absolute minimum gap possible (Figure 3-8c) and it is forced to intersect $F_{e,max}$. $g_{contact}$ is the point at which the two beams come into contact (Figure 3-8b). $g_{contact}$ can be thought of as an insulating layer surrounding the composite metal beam. It is this insulating layer which provides the restoring force

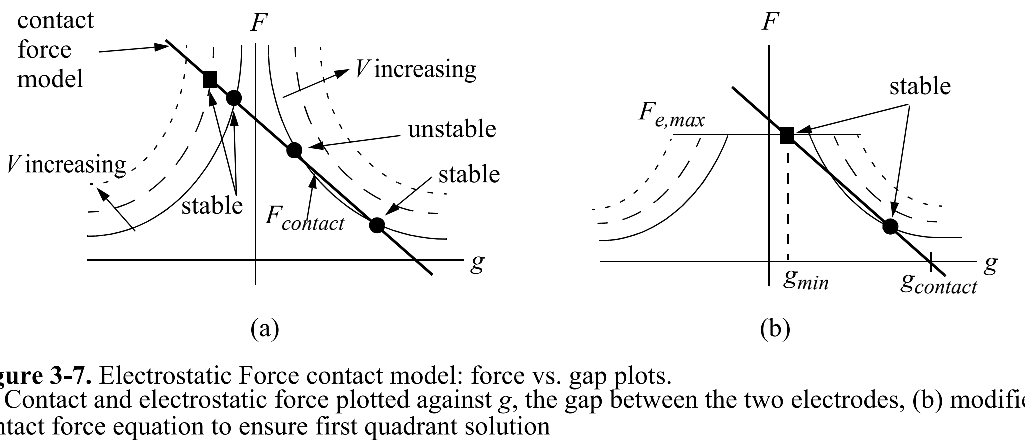


Figure 3-7. Electrostatic Force contact model: force vs. gap plots.
(a) Contact and electrostatic force plotted against g, the gap between the two electrodes, (b) modified contact force equation to ensure first quadrant solution

```
@ (initial_step) begin
      FeMax =  -0.5*`eps0*thickness*pow(elec_breakdown,2);
      // slope of the contact force (per unit length)
      slope = FeMax/(gmin - gcontact);
end

// check to see if there is a gap
if (dy<gcontact) begin
      // contact force
      Fy_contact = slope*(dy-gcontact);
end
else begin
      // there is a normal gap, no contact
      Fy_contact = 0;
end

// if the electric force per unit length
// is greater than the max electric force, set the force to FeMax
if (abs(Felec) > abs(FeMax)) begin
       Fl = FeMax;
end
else begin
       Fl = Felec;
end
```

Listing 3-7. Contact model.

when the two beams come into contact. The new contact model is a line defined by the two points: (g_{min}, $F_{e,max}$), ($g_{contact}$, 0). There is no longer a solution in the second quadrant because the $1/g^2$ curves are capped at $F_{e,max}$ and cannot intersect with the contact force load line.

The Verilog-A implementation of the contact model is shown in Listing 3-7. The variable *FeMax* is the maximum electrostatic force per unit length ($F_{e,max}/l$). Since *FeMax* is calculated from

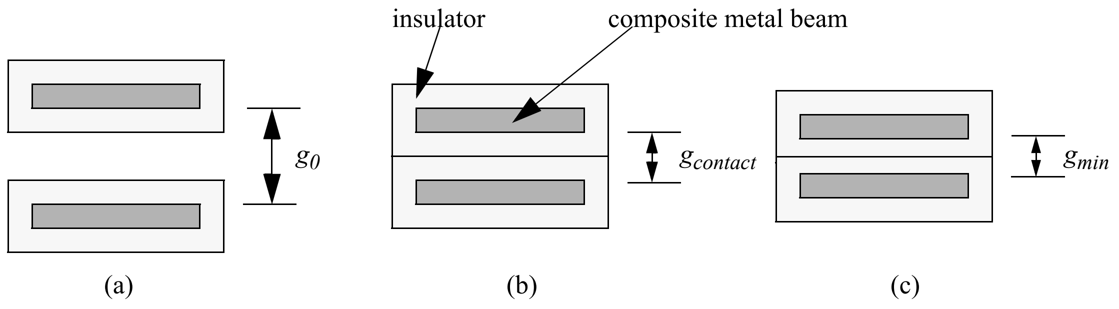


Figure 3-8. gcontact and gmin
An insulating layer was created for the contact model. This layer provides a reaction force when the beams come into contact. (a) normal gap, no contact. (b) gap when first in contact, $g_{contact}$. (c) minimum possible gap, g_{min}.

```
analog function real if11;
    input L,x;
    real L,x;
    if11 = x - pow(x,3)/pow(L,2) + pow(x,4)/(2.0*pow(L,3));
endfunction
```

Listing 3-8. Beam shape function integral example.

a user-defined process parameter, the breakdown electric field value, it does not have any time dependence and therefore can be calculated in the initial step. Similarly, the slope of the contact force line is constant since *gmin* and *gcontact* are user defined parameters that have no time dependence. The first if/else block checks to see if the instantaneous gap solution (*dy*) is less than *gcontact*. If it is, the beams are in contact and the contact model is activated, otherwise there is no contact force. The second if/else block imposes the maximum electrostatic force. If the electrostatic force per unit length exceeds *FeMax*, the force per unit length, *Fl*, is set to *FeMax*, otherwise it is set to *FeMax*.

3.6 Gap Model Lumping Equations

The electrostatic, damping and contact forces must be lumped to the nodes of the model. The integral of the force per unit length of these forces over the beam shape function accomplishes this task (Section 3.1). Listing 3-8 shows an analog function *if11* which is the definite integral of the beam basis function in (3.4). The other three beam shape functions can be found in the electrostatic gap Verilog-A code listed in Appendix A.

Listing 3-9 shows the equations for the lumped forces and moments in the rotated parallel plate frame of reference. See Section 3.2 for a description of *x1l*, *x2l*, *x1r*, *x2r*. In the rotated parallel plate approximation, the damping and contact force per unit lengths are not dependent on the position along the beam. Therefore, these forces can be moved outside of the lumping integrals, thus simplifying the calculation. *Fsum* is the force balance equation that sums the electrostatic, contact and damping forces per unit length.

```
Fsum = Fl-Fy_contact-Fy_damping;

Fa1_l = -Fsum*(if11(L1,x1r) - if11(L1,x1l));
Ma1_l = -Fsum*(if21(L1,x1r) - if21(L1,x1l));
Fb1_l = -Fsum*(if31(L1,x1r) - if31(L1,x1l));
Mb1_l = -Fsum*(if41(L1,x1r) - if41(L1,x1l));
Fa2_l = Fsum*(if11(L2,x2r) - if11(L2,x2l));
Ma2_l = Fsum*(if21(L2,x2r) - if21(L2,x2l));
Fb2_l = Fsum*(if31(L2,x2r) - if31(L2,x2l));
Mb2_l = Fsum*(if41(L2,x2r) - if41(L2,x2l));
```

Listing 3-9. Lumped forces and moments.

```
var1 = `eps0*thickness/(dy);
Fxd_l = 0.5*v_squared*var1;
```

Listing 3-10. Lateral electrostatic force.

3.7 Lateral Electrostatic Force

If an overlap region is non-zero and does not completely overlap either beam, an electrostatic force in the x direction exists. The force is:

$$F_{elec,x} = \frac{1}{2}\frac{d}{dx}(CV^2) = \frac{1}{2}V^2\frac{dC}{dx} \tag{3.23}$$

Since a parallel plate approximation is made, C is a function of the overlap length l (region **ii** in Figure 3-3). Therefore, the derivative is taken with respect to the change in the overlap region. The capacitance is given by (3.24).

$$C = \frac{\varepsilon_o h l}{g} \tag{3.24}$$

where h is the thickness, l is length of the overlap region, g is the gap and ε_o is the permittivity of free space. (3.23) can then be reduced to:

$$F_{elec,x} = \frac{1}{2}V^2\frac{\varepsilon_o h}{g} \tag{3.25}$$

The Verilog-A implementation of the lateral electrostatic force is shown in Listing 3-10. *Fxd_l* is the lateral electrostatic force per unit length. *var1* is a variable used to reduce redundant calculations, *eps0* is the permittivity of free space, *thickness* is the thickness of the beam, *dy* is the instantaneous rotated parallel plate gap (shown as *g* in (3.23) and (3.24)) and *v_squared* is the voltage squared.

3.8 Frames of Reference

The gap model uses three frames of reference: the chip (layout) frame, the local frame, and the rotated frame. The chip frame is the frame where the individual atomic elements (beams, plates, gaps, etc.) are drawn. The equations for all NODAS atomic elements operate in the local frame [17]. Finally, the rotated frame is the frame used to calculate the electrostatic forces for the rotated parallel plate approximation (Figure 3-4).

To transform the coordinates from the chip frame to local frame, (3.26) is used. To transform the forces from the local frame to chip frame (3.27) is used. Similarly, to transform the forces from the rotated frame to the local frame, (3.27) is used. (3.28) and (3.29) give the rotation matrix and its inverse for a 2D coordinate system. θ is the rotation angle.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.26}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = R^{-1} \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{3.27}$$

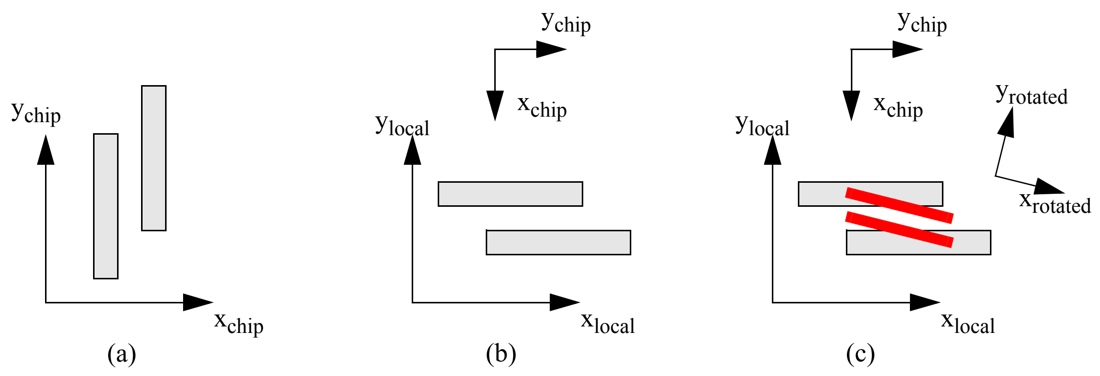


Figure 3-9. Frames of reference.
(a) chip frame, (b) local frame, (c) rotated frame.

```
// rotate positions from chip frame to local frame
xm1_l = cos_dc*Pos(xa_t[0]) + sin_dc*Pos(xa_t[1]);
ym1_l = cos_dc*Pos(xa_t[1]) - sin_dc*Pos(xa_t[0]);


// rotate forces from local frame to chip frame
F(xa_t[0]) <+  (cos_dc*Fa1x-sin_dc*Fy_1l)*finger_number;
F(xa_t[1]) <+  (sin_dc*Fa1x+cos_dc*Fy_1l)*finger_number;
```

Listing 3-11. Frame of reference coordinate transforms.

$$R = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \tag{3.28}$$

$$R^{-1} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \tag{3.29}$$

In Figure 3-9 and (3.26) (3.27), *x'* and *y'* correspond to y_{local} and x_{local}, whereas *x* and *y* correspond to x_{chip} and y_{chip}. An example of the rotation from the chip frame to the local frame is shown in first two lines of Listing 3-11. Similarly, the rotation from the local frame to the chip frame is shown in the last two lines of Listing 3-11. *cos_dc* and *sin_dc* are the calculated cosine and sine of the rotation angle between the chip and local frames. *finger_number* is a user parameter which multiplies the force and moments by that number. The default value is set to 1.

3.9 Electrical Modeling

The total current crossing the gap is given by (3.30).

$$I_{total} = \frac{dq}{dt} = \frac{d(CV)}{dt} \tag{3.30}$$

where *C* is the total parallel plate capacitance between the two beam electrodes (3.24). The total current must be lumped to both nodes of the gap model. A rigid plate approximation is used to lump these currents [17]. The lumped currents are given in (3.31) and (3.32).

$$I_{tip} = I_{total}\left(1 - \frac{ov}{2l}\right) \tag{3.31}$$

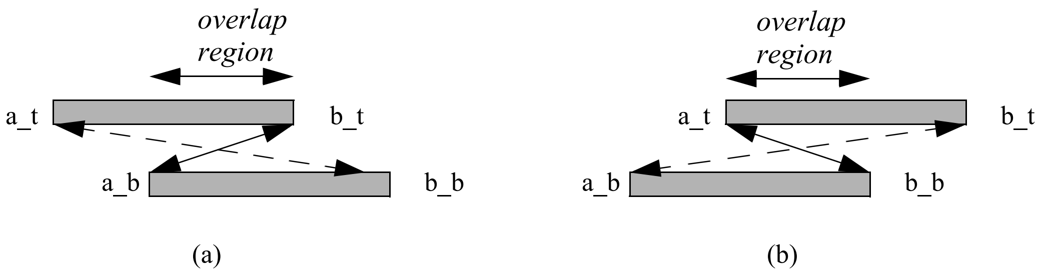


Figure 3-10. Current flows. I_{attach} (dashed line), I_{tip} (solid line).

```
var1 = `eps0*thickness/(dy);
cap = var1*ov;
q = v * cap;

var2 = ov/2.0/finger_l_t;
I_tip    = ddt(q)*(1-var2);
I_attach = ddt(q)*var2;

// the tip and the attached points depend on the dynoffset vars
if (dynoffsetL2Left >=0) begin
   I(vb_t,va_b) <+ I_tip;
   I(va_t,vb_b) <+ I_attach;
end
else begin
   I(vb_t,va_b) <+ I_attach;
   I(va_t,vb_b) <+ I_tip;
end
```

Listing 3-12. Electrical model.

$$I_{attach} = I_{total}\left(\frac{ov}{2l}\right) \tag{3.32}$$

where *l* is the length of the beam electrode, *ov* is the length of the overlap region. I_{tip} is the current that is flowing between the overlapped ends of the beam electrodes. I_{attach} is the current flowing between the other two ends of the beam electrodes. In Figure 3-10a, I_{tip} is the current flowing from node *b_t* to *a_b* and I_{attach} is the current flowing from *a_t* to *b_b*. If both beams are completely overlapped (i.e. *ov* = *l*), half of the total current flows through each node of a given beam.

The Verilog-A implementation of the electrical model is shown in Listing 3-12. *var1* and *var2* are variables used to reduce redundant calculations (Appendix B.8). *finger_l_t* is the electrode length of the top electrode. *ddt(q)* is the time derivative of the charge stored on the capacitor (i.e.

the current). The if/else block assigns the current based on the beam configuration. The if block corresponds to Figure 3-10a and the else block corresponds to Figure 3-10b.

3.10 Special cases

Electrostatic forces only exist when the overlap region is greater than zero. The gap model must capture this requirement. Listing 3-13 shows the Verilog-A code which calculates the electrostatic force per unit length, contact force per unit length, and gap. If there is no overlap, the force per unit lengths are set to 0 and the gap (*dy*) is set to the original gap. *dy* must be set to a value to prevent hidden states (Appendix B.2). When the overlap region exists, the calculations listed in Listing 3-4-Listing 3-5, and Listing 3-7 are performed. The code is not re-listed here to save space.

Similarly, when transforming the forces from the rotated frame to the local frame, care must be taken to ensure that the forces are correctly captured (Listing 3-14). Since lateral electrostatic forces in the x direction can exist, the code must be able to determine if there is a lateral electrostatic force. If the overlap region exists and is not equal to either beam's length, then the translation of forces proceeds as detailed in Section 3.8. If however, the overlap region does not exist, or the overlap region is equal to either beam's length, there is no lateral force. When there is no overlap region,

```
if (ov > 0) begin
   // calculate Felec, dy, Fy_contact, Fl
   ...

end
else begin
   // there is no parallel plate electric field
   Felec = 0;
   // set this to some reasonable number (such as the original gap)
   dy_l = gap;
   dy = gap;
   // there is no contact force
   Fy_contact = 0;
   // there is no force per unit length
   Fl = 0;
end
```

Listing 3-13. Special case: forces only exist when there is an overlap region.

```
if ((ov != L1) && (ov != L2) && (ov > 0)) begin
  // translate the forces from the rotated frame to the local frame
  ...
end
else begin
  // the overlap is equal to L1 or L2
  Fxd_l = 0;
  Fa1x = 0;
  Fb1x = 0;
  Fa2x = 0;
  Fb2x = 0;
  // no x component for the y forces
  Fy_1r = cos(theta0)*Fb1_l;
  Fy_1l = cos(theta0)*Fa1_l;
  Fy_2r = cos(theta0)*Fb2_l;
  Fy_2l = cos(theta0)*Fa2_l;
end
```

Listing 3-14. Special case: forces in x direction only exist when there is an overlap.

all of the forces in the x direction are set to zero and do not show up in the calculation of the forces in the y direction in the local frame.

3.11 Electric only gap model

An electric only gap model was created in order to address a simulation speed issue. While attempting to simulate a MEMS varactor (Section 4.2) using NODAS, it was determined that a full electrostatic and mechanical simulation of this type of structure was too computationally expensive. The varactor modeled is shown in Figure 3-11. The capacitor design utilizes both parallel-plate gap and area tuning. The capacitor yoke beams provide support for the capacitor fingers. In this design, there are 40 capacitor fingers on each yoke beam. There are 10 yoke beams in total, 5 beams are connected to a fixed frame and 5 are connected to a movable frame. When the thermal actuators are heated, the structure will move left or right, thus changing the overlap of the capacitor fingers (area tuning) as well as the gap between the yoke beams (gap tuning).

Table 3-1 shows the results of the full varactor simulation (40 fingers per frame by 5 frame sections, each section has a fixed and movable frame). The DC operation point simulation takes over 4 hours on a dual 1GHz UltraSparc III SunFire 280R with 4GB of RAM running SunOS 5.9.

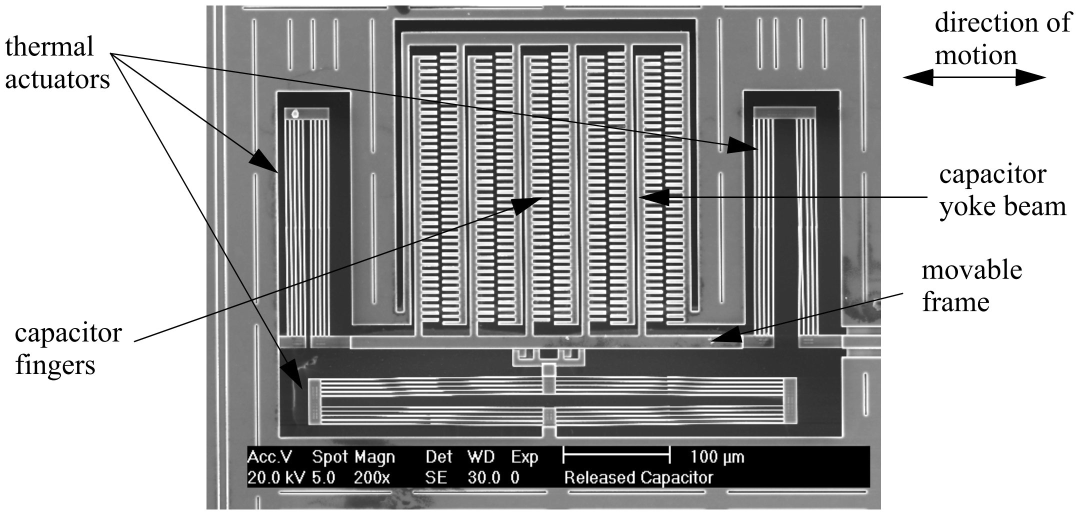


Figure 3-11. Finger based varactor design, A.Oz. [30]. Area and gap tuning.

As a simple test, the structure was reduced to a 5x5 (5 fingers per frame, 5 frame sections) and 5x1 (5 fingers per frame, 1 frame section) to see how the simulation time scaled. When increasing the number of nodes, N, the growth rate was determined to be $N^{3.5}$.

Table 3-1. Beam and electromechanical gap DC OP simulation time of finger-based MEMS varactor.

Size	Nodes	Equations	Iterations	Time (s)
Full Varactor	7051	61914	4	16602
5x5	1843	10851	5	106.6
5x1	439	2477	5	1.17

To simplify the problem, the NODAS gap elements were removed and a beam only simulation was run to get a baseline simulation time. Table 3-2 shows the results of that simulation. The electromechanical gap model was increasing the simulation time significantly. The growth rate with the beam-only simulation was determined to be $N^{1.8}$.

Table 3-2. Beam only DC OP simulation time of finger-based MEMS varactor.

Size	Nodes	Equations	Iterations	Time (s)
Full Varactor	10453	43708	2	43.17
5x5	1777	7354	2	1.29
5x1	379	1592	2	0.1

The simulation time growth rate was unacceptable and it was decided to create a gap model with the electrostatic forces and moments removed. This model only contained code to calculate the capacitive currents and the effects of externally applied displacements. The simulation time for this model is shown in Table 3-3. The growth rate was determined to be $N^{2.2}$, a significant improvement over the full electromechanical gap model.

Table 3-3. Beam and gap (electrostatic only) DC OP simulation time of finger-based MEMS varactor.

Size	Nodes	Equations	Iterations	Time (s)
Full Varactor	7051	60394	3	151.38
5x5	1213	8005	3	1.96
5x1	307	1829	2	0.16

The cause of the high growth rate in the full electromechanical model seems to stem from the fact that the finger based varactor has gap models which act in the *x* and *y* directions. It is possible that a sparse matrix is no longer maintained in this case, causing the simulation time to explode as the number of nodes increases.

The electric only gap model should be used in simulations with a large number of nodes and where the structure can be assumed to be stiff enough so that the beams do not have significant bending. In smaller simulations, the full electromechanical gap model can be used.

3.12 Verification

3.12.1 Contact Model

The canonical gap problem shown in Figure 3-6 is used to verify that the contact model functions properly. Figure 3-12 shows the NODAS schematic for the canonical gap problem. The original contact model allowed the simulator to find a solution where the gap was negative. Figure 3-13a is the DC sweep plot showing the gap (y axis) versus voltage (x axis) using the original contact model. The initial gap is 2 μm and the voltage is swept from 0V to 10000V. As the voltage is

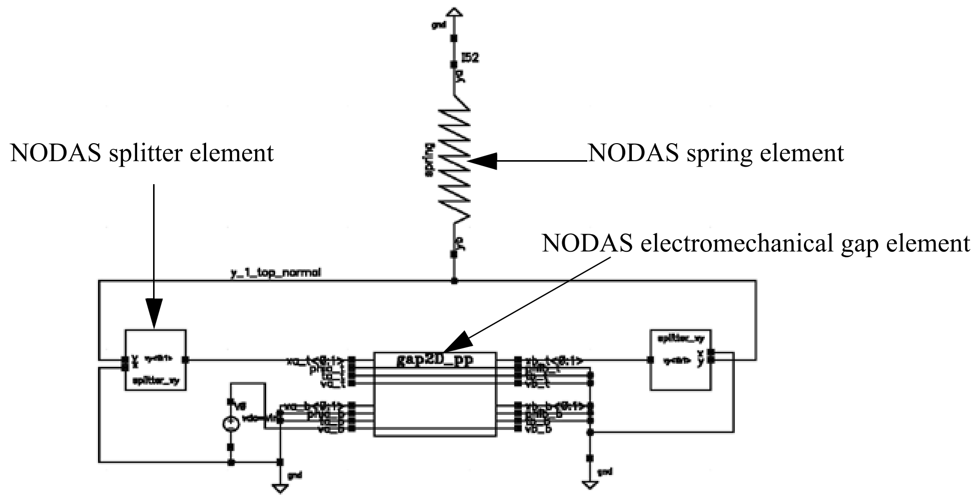
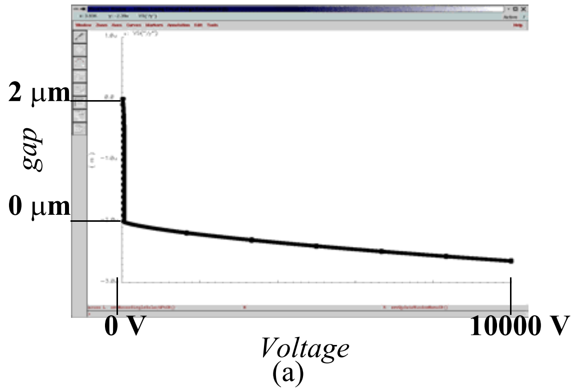
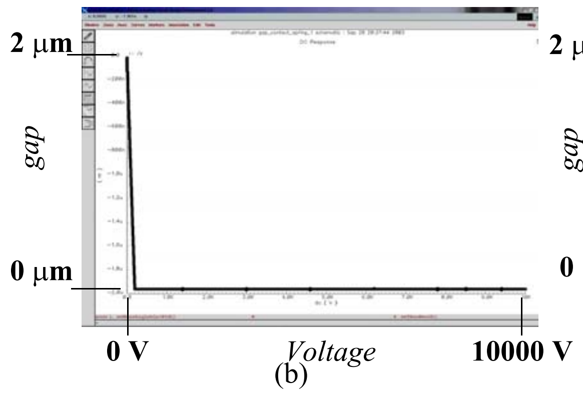


Figure 3-12. NODAS schematic for canonical gap problem.





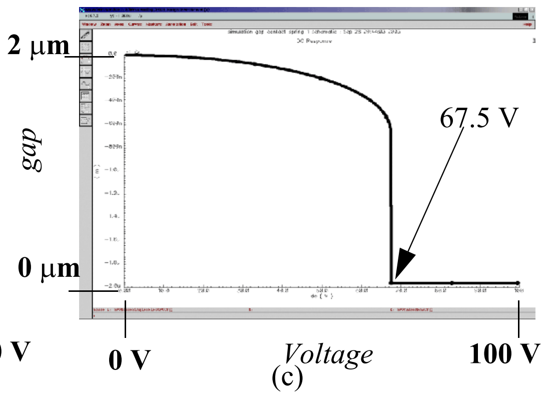


Figure 3-13. Contact model simulation plots: gap versus voltage.
(a) original contact model showing that a negative gap is found when sufficient voltage is applied. (b) revised contact model showing that the simulator does not find the negative gap solution. (c) 0V - 100V plot of (b) showing that the snap-in voltage occurs at 67.5V for this problem.

increased, the gap decreases, however, the gap becomes negative at high voltages. Figure 3-13b is the same DC sweep simulation but using the new contact model. As can be seen, the simulator only finds the positive gap solutions.

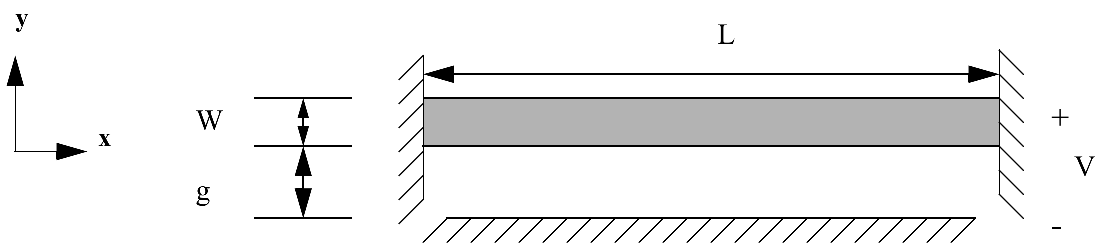


Figure 3-14. Fixed-Fixed beam with electrostatic actuation.

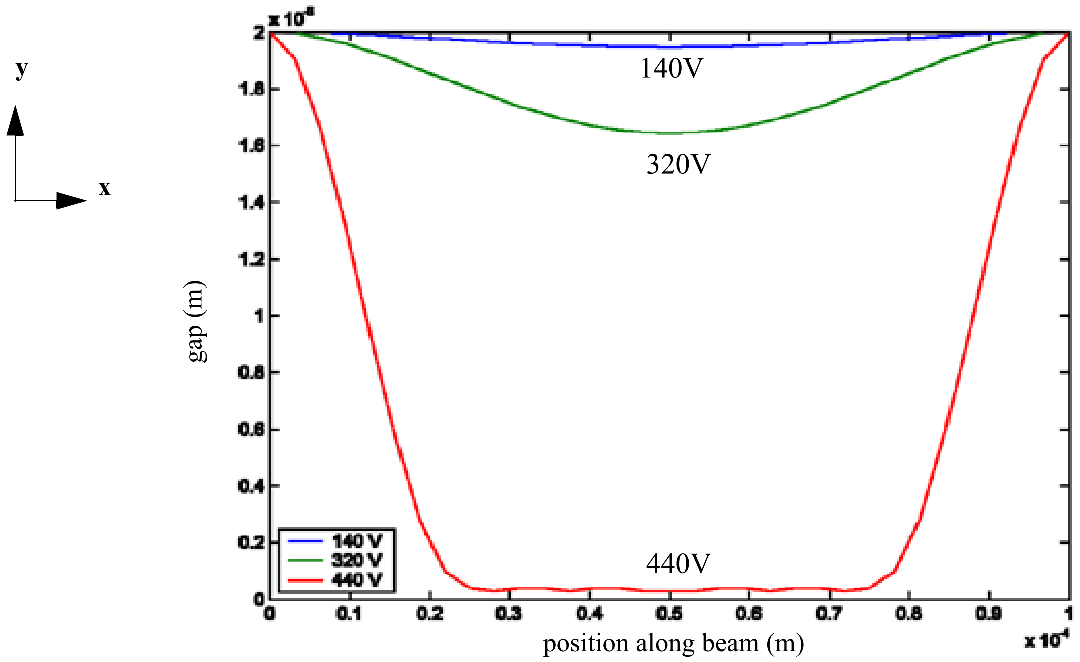


Figure 3-15. Fixed-Fixed beam mode shapes.
100 µm long beam, 2 µm wide, 2 µm thick, 2 µm gap, E = 170 GPa. Voltages: 140V, 320V, 440V. 128 NODAS beam/gap segments were used.

3.12.2 Fixed-Fixed Beam

To verify the gap model, a fixed-fixed beam (Figure 3-14) with electrostatic actuation was created in NODAS and ANSYS, a finite element package [2]. The fixed-fixed beam problem is a good test case because two important physical factors can be verified: large axial forces and non-parallel electric fields. As the voltage is increased between the fixed-fixed beam and the bottom electrode, the beam bends toward the electrode, eventually coming into contact with the electrode. The results of NODAS simulations of a beam that is 100 µm long, 2 µm wide, 2 µm thick, with a 2 µm gap, and with Young’s Modulus of 170 GPa is shown in Figure 3-15. The NODAS schematic

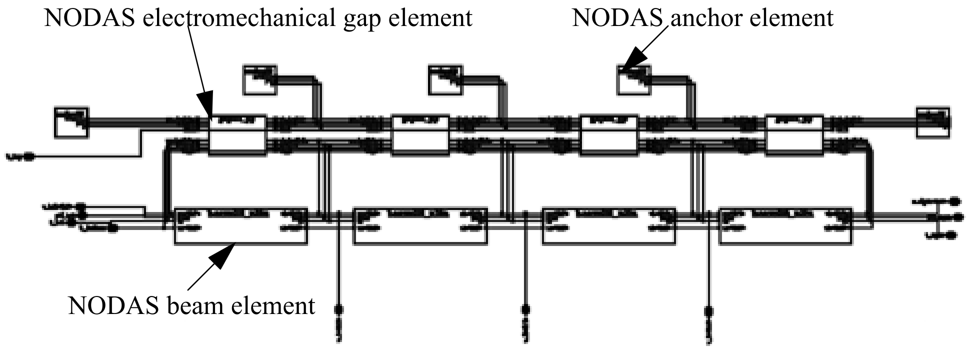


Figure 3-16. 4 NODAS beam and gap segments subcell.
This subcell is used for the 4, 8, 16, 32, 64, and 128 NODAS beam and gap element fixed-fixed beam simulations. The top nodes of the gap models are anchored to model the fixed electrode. Note that the schematic shown here models the fixed electrode at the top whereas in Figure 3-14, the fixed electrode is at the bottom. The displacements reported by the NODAS simulations were transformed in MATLAB by subtracting the displacements from the initial gap.

for this problem has 128 NODAS beam and 128 NODAS gap elements. These dimensions are typical for a polysilicon MEMS process such as MUMPS.

The ANSYS tutorial provides a sample electromechanical simulation (Appendix C.1), the results of which were used to compare to the NODAS simulation. Unlike the previous example shown in Figure 3-15, the beam in this example is thicker, so as to make the beam more compliant, and the gap is smaller. Both of these properties make this beam suitable to be used as a MEMS switch. Whereas the pull-in voltage in the previous example was between 320V and 440V, the pull-in voltage for this MEMS switch is around 16.5V and 17.5V. The beam's properties are: L = 80 μm, W = 0.5 μm, *thickness* = 10 μm, E = 169 GPa, ν = 0.25. In the ANSYS model, the Plane42 (with plane stress option), Trans109, and Contac12 elements were used. To model the system and to compare the accuracy of the NODAS model with the ANSYS model, the length of the beam was broken up into 2, 4, 8, 16, 32, 64, and 128 NODAS beam and gap segments. Figure 3-16 shows an example of 4 beam and gap segments.

Table 3-4 lists the percentage difference of the y displacement results between NODAS and ANSYS. The comparison was made at the center of the fixed-fixed beam, though the results are similar along the length of the beam. The percentage difference of the displacements is defined as:

$$\frac{y_{ANSYS} - y_{NODAS}}{y_{ANSYS}} \cdot 100 \tag{3.33}$$

Two voltage points are given. At 16.5V, the system is near the instability point and on the verge of snap-in. In the 2, 4, and 8 beam/gap segment cases, the NODAS model reported that the beam had snapped in, thus resulting in the high percentage difference between NODAS and ANSYS. As more segments were added, the NODAS model was able to model the electrostatic force and the beam bending more accurately. At 14.5V, the beam still exhibits a large deflection, but it not near the snap-in point. This case demonstrates the accuracy of NODAS when operating away from the snap-in voltage. Using 8 or more beam/gap segments results in accuracy of greater than 9%. Thus, if high accuracy is required, at least 8 beam/gap segments should be used.

Table 3-4. Fixed-Fixed beam: NODAS versus ANSYS. Comparison at the center of the beam.

Number of NODAS Beam/Gap Segments	481 ANSYS Elements % difference at 14.5V	481 ANSYS Elements % difference at 16.5V (near pull-in point)	6156 ANSYS Elements % difference at 14.5V	6156 ANSYS Elements % difference at 16.5V (near pull-in point)
2	29.54	167.97	30.85	172.58
4	15.35	141.76	16.51	145.92
8	8.82	142.44	9.92	146.61
16	4.96	15.32	6.01	17.33
32	2.87	6.81	3.91	8.65
64	1.82	3.57	2.84	5.35
128	1.33	2.16	2.34	3.91

The simulation times for the NODAS and ANSYS models are given in Table 3-5 and Table 3-6. The accuracy results in Table 3-4 were calculated using the 481 element ANSYS model. Accuracy comparisons between NODAS and ANSYS were also performed using the finest mesh possible (6156 elements) given the constraints of the computer used for the ANSYS simulations. The change in accuracy between NODAS and the ANSYS 481 and 6156 element meshes was approximately an additional 1%-5% difference. Even with the finest ANSYS mesh for this problem, NODAS achieves a accuracy of within 4% (with 128 NODAS beam/gap segments) with a 4x reduction of simulation time.

Table 3-5. NODAS simulation time versus # of NODAS elements for the fixed-fixed beam example. Simulations were run on a 2GHz Pentium IV machine with 1GB of RAM running Windows 2000 Professional.

# of NODAS beam/gap segments	# of Nodes	# of Equations	Simulation time (s)
2	25	117	0.03
4	53	259	0.07
8	101	506	0.14
16	197	1000	0.35
32	389	1988	1.23
64	773	3964	2.11
128	1541	7916	7.41

Table 3-6. ANSYS simulation time versus # of ANSYS elements for the fixed-fixed beam example. Simulation were run on a dual 1GHz UltraSparc III Sun-Fire 280R with 4GB RAM running SunOS 5.9.

# of mesh Elements	# of Nodes	Simulation time (s)
481	405	1
805	809	3
1441	1127	5
4005	3213	15
6156	4797	29

4 Application Examples

The goal of this chapter is to introduce several examples of MEMS devices which can be simulated using NODAS. Since the NODAS models are composable, complex devices can be created that yield accurate results. The types of devices that can be simulated are not limited by the examples shown in this chapter. As long as the device can be broken up into beam, plate and gap elements (Figure 1-2), it can be simulated in NODAS. Three examples will be discussed in this chapter: a thermal actuator, a varactor, and a resonant mixer. The thermal actuator example will demonstrate the use of the thermal beam model. The varactor and resonant mixer examples will demonstrate the use of the beam and gap models.

4.1 Thermal Actuator

The thermal extensions to the NODAS beam model described in Chapter 2 enable the simulation of electrothermal physics involved in a thermal actuator. A CMOS-MEMS thermal actuator is an example of how the NODAS beam model can be used to model thermal effects. The CMOS-MEMS thermal actuator exploits the differences in the thermal coefficient of expansions (TCEs) of the different materials in a CMOS beam stack. Figure 4-1 shows the layout view of a thermal actuator designed by A. Oz [30]. When a temperature is applied, the actuator tip will move either left or right depending on the thermal coefficient of expansions and the metal layer offsets. By offsetting the metal position in the beam cross section, the effects of the thermal bending moment can be tuned to get the desired tip displacements. This thermal actuator is composed of four quadrants connected by plates. Each quadrant has five parallel beams and each beam is divided into three sections. Half

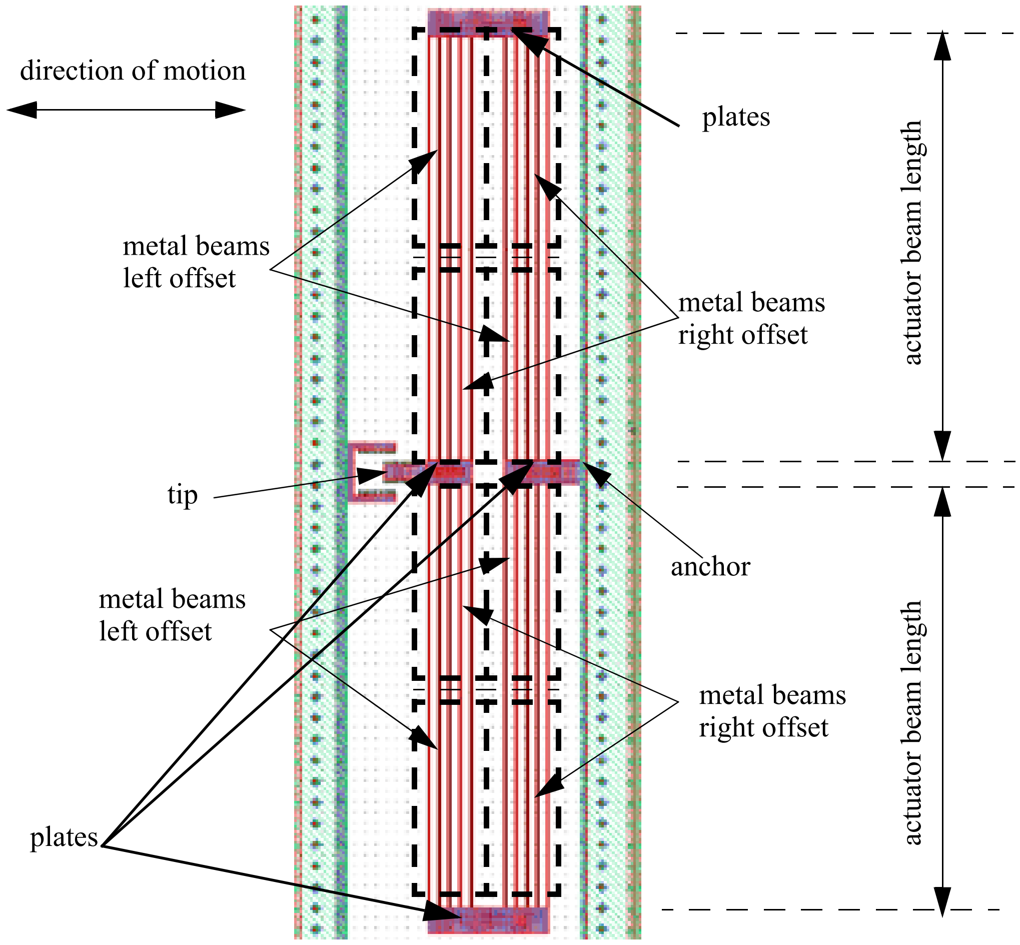


Figure 4-1. Thermal Actuator 1A layout in Jazz 0.35 process. A. Oz [30].

way through the beam length of a single actuator leg, the direction of the offset changes (see dotted boxes in Figure 4-1). Figure 4-2 shows the cross section view of a single actuator beam leg. Half of the beam's length has a left offset for metal 1 and metal 2 and the other half has a right offset. For this design, the metal 3 width is 1.2 μm and the offsets of the metal 1 and metal 2 layers are 0.6 μm. The widths of metal 1 and metal 2 are 0.6 μm. At the center of the beam's length, all three metal layers span the 1.2 μm width of the beam.

The NODAS schematic is of Actuator 1A is shown in Figure 4-3. As shown, each actuator beam length is split into three sections. The top and bottom sections of the length contain metal stacks where metal 2 and metal 1 are either offset to the left or to the right. The center beam is a small 2 μm region where the metal 3, metal 2, and metal 1 layers span the whole 1.2 μm width. A

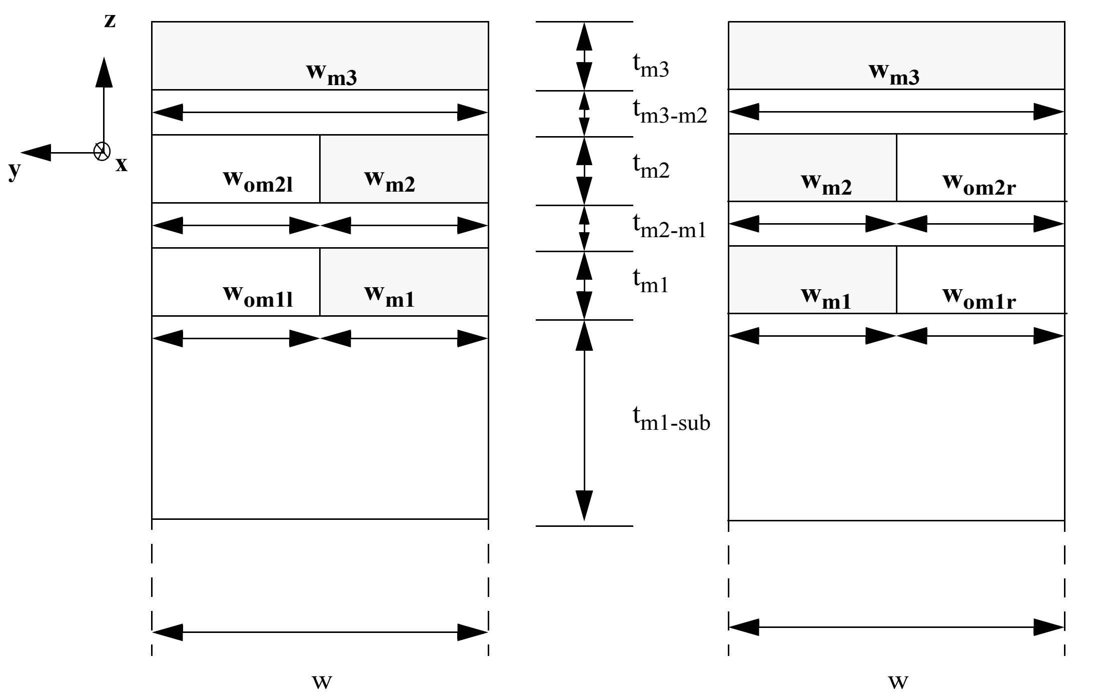


Figure 4-2. Cross section views of the offset metal layer stack in Actuator 1A.
Jazz 0.35 process [30].$w_{om2l} = w_{om1l} = w_{om2r} = w_{om1r} = 0.6$ µm. $w = w_{m3} = 1.2$ µm. $w_{m2} = w_{m1} = 0.6$ µm. This figure shows two different points on the beam's length. The metal 1 and 2 layers are either offset to the left or to the right.

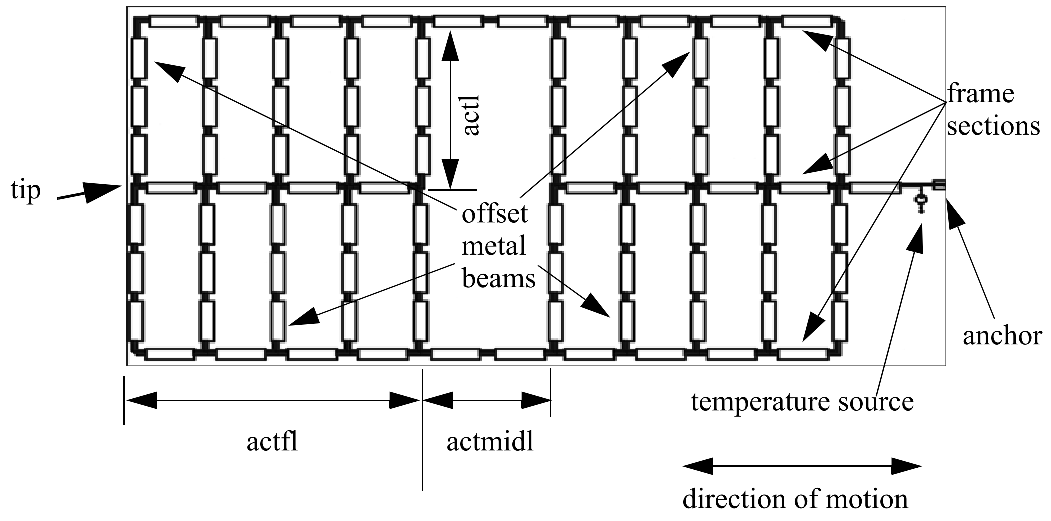


Figure 4-3. Thermal Actuator 1A NODAS schematic.
Uses NODAS 2D linear beams. *actl* = 200 µm, *midl* = 16 µm, *actfl* = 20 µm.

NODAS DC temperature source is used to apply a temperature to the structure. The three frames connecting the actuator beams all have the same widths and lengths.

The actuator's beam length, *actl*, is 200 µm, and the width, *actw* is 1.2 µm. The length of the frame connecting the five actuator sections, *actfl*, is 20 µm, and it's width is 12 µm. The frame section connecting the two halves (left and right groups of five), *actmidl*, has a length of 16 μm and a width, *actmidw*, of 12 μm. The thickness used for the simulations is 10 μm. The characteristic temperature, T_0, was experimentally determined to be 367K. For the simulations, the temperature coefficients of expansion, α, for metal and oxide are 28.3 μm/K and 0.4 μm/K respectively.

Table 4-1 summarizes the results of the measured actuator tip deflection, Coventor finite element simulation [1], and NODAS simulation displacements. The reported displacements are relative displacements from the self-assembly position, which is found at room temperature (294K). The original data and Coventor simulations can be found in [30]. The percentage difference in the relative displacements between Coventor and NODAS is less than 10%. The differences between NODAS and the experimental measurements can be accounted for by the uncertainty in the thermal coefficients of expansion, the exact value of the characteristic temperature as well as the actual fabricated dimensions.

Table 4-1. Thermal Actuator 1A NODAS versus measured. Displacements are relative to self assembly displacement at T=294K. For NODAS and Coventor simulations, α_{metal} = 28.3 μm/K, α_{ox} = 0.4 μm/K, and T_0 = 367K.

Temperature (K)	Measured Displacement (μm)	Coventor Displacement (μm)	NODAS Displacement (μm)	NODAS simulation time (s)
325K	5.5	4.3	4.2	0.11
350K	10.5	8.3	7.6	0.11
375K	16.0	12.3	11.0	0.11
400K	21.0	16.4	17.8	0.11
425K	25.5	20.5	21.2	0.11

4.2 Varactor

A varactor is a variable capacitor. A CMOS-MEMS implementation of a varactor was designed by A. Oz [30]. The NODAS schematic of this varactor uses both the electrostatic gap

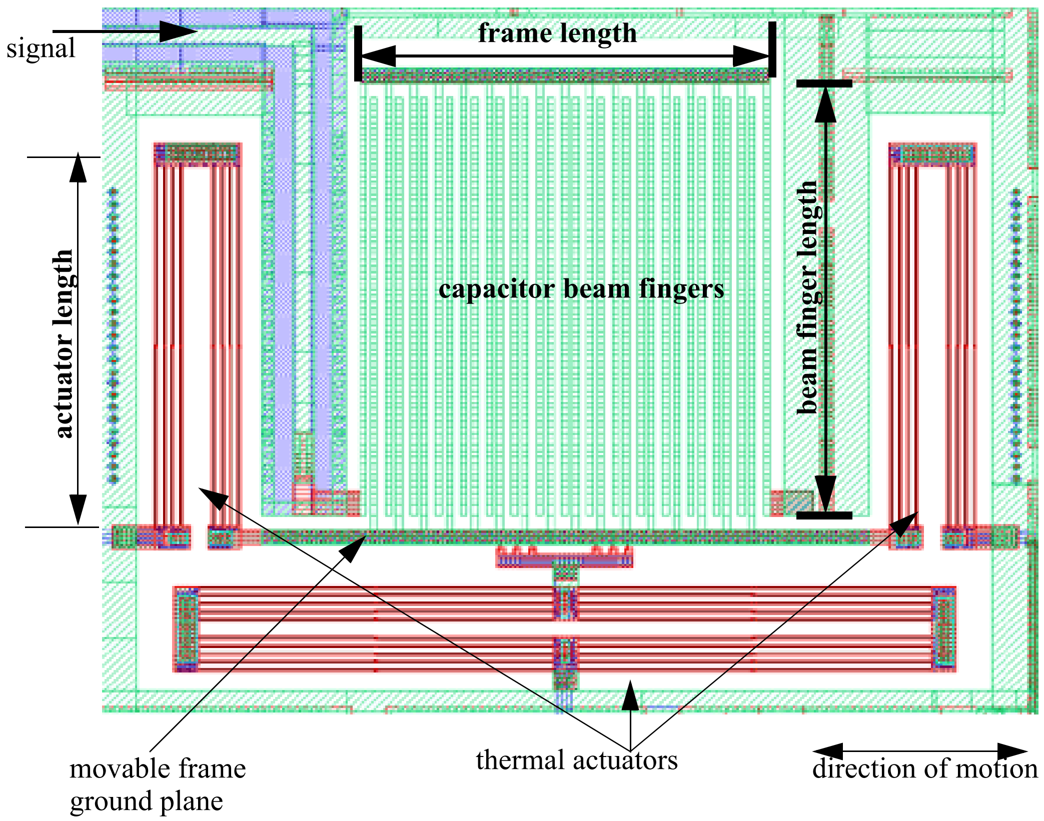


Figure 4-4. Layout view of beam based varactor design, Jazz 0.35 process [30]. The varactor is a gap tuning design.

model as well as the 2D linear beam model with thermal effects. Figure 4-4 shows the layout view of this varactor. It is a beam-based design. The gap between the capacitor beam fingers can be changed (gap tuning) by heating the thermal actuators, causing the movable frame to move left or right. The minimum capacitance occurs when the gap between the beams are equally spaced. Electrically, the signal enters from the upper left corner of the device. The movable frame is connected to electrical ground. The beam stack in the capacitor uses metal 1 through metal 4. The beam stack in the actuators use poly, metal 1, metal 2 and metal 3, however, only metal 2 and metal 3 are connected to ground. The poly in the actuators is used for thermal heating and metal 1 is used to supply the current to the poly resistors. The bottom thermal actuator is used to latch the capacitor into a high capacitance or low capacitance value. This latch mechanism is not modeled in this NODAS simulation. Metal 1 and metal 2 in the actuators are offset to the left or to the right edge of the beam cross section (see Section 4.1) in order to provide a thermal bending moment at room temperature.

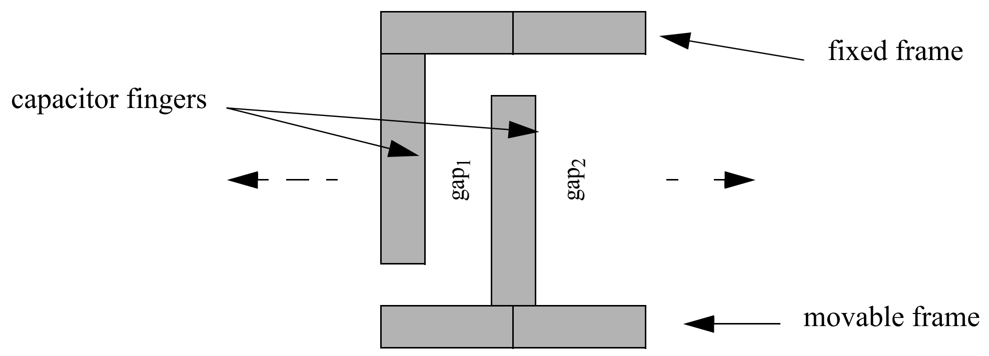


Figure 4-5. Capacitor finger subcell.

This moment causes the capacitor structure to move to it's self-assembled position after the structure is released.

A hierarchy of subcells with NODAS elements is used to model the varactor. The thermal actuator subcell models the offset metal beams used for thermal actuation. The capacitor subcell is broken up into two levels of hierarchy. The first level captures the number of capacitor beam fingers. It instantiates the common subcell for the capacitor section. Figure 4-5 shows the graphic representation of the capacitor beam finger subcell. It models one beam finger mounted on the fixed frame and one beam finger mounted on the movable frame. It uses two gap models to capture the electrostatic and capacitive effects between the capacitor beam fingers. Figure 4-6 shows the hierarchy of the varactor simulation.

A DC operating point and a S-parameter analysis were performed in Cadence Spectre. Table 4-2 shows the simulation time versus the number of nodes for each analysis type. The S-parameter simulation was swept from 100 MHz to 6GHz using the default conservative step size option in Spectre (20 total steps for this simulation).

Table 4-2. DC OP and S-parameter simulation time.The S-parameter simulation was run from 100 MHz to 6GHz. Simulations were run on a dual 1GHz UltraSparc III Sun-Fire 280R with 4GB of RAM running SunOS 5.9.

Simulation Type	Simulation Time (s)	# of Nodes	# of Equations
DC OP	3.76	819	16097
S-parameter	7.54	819	16097

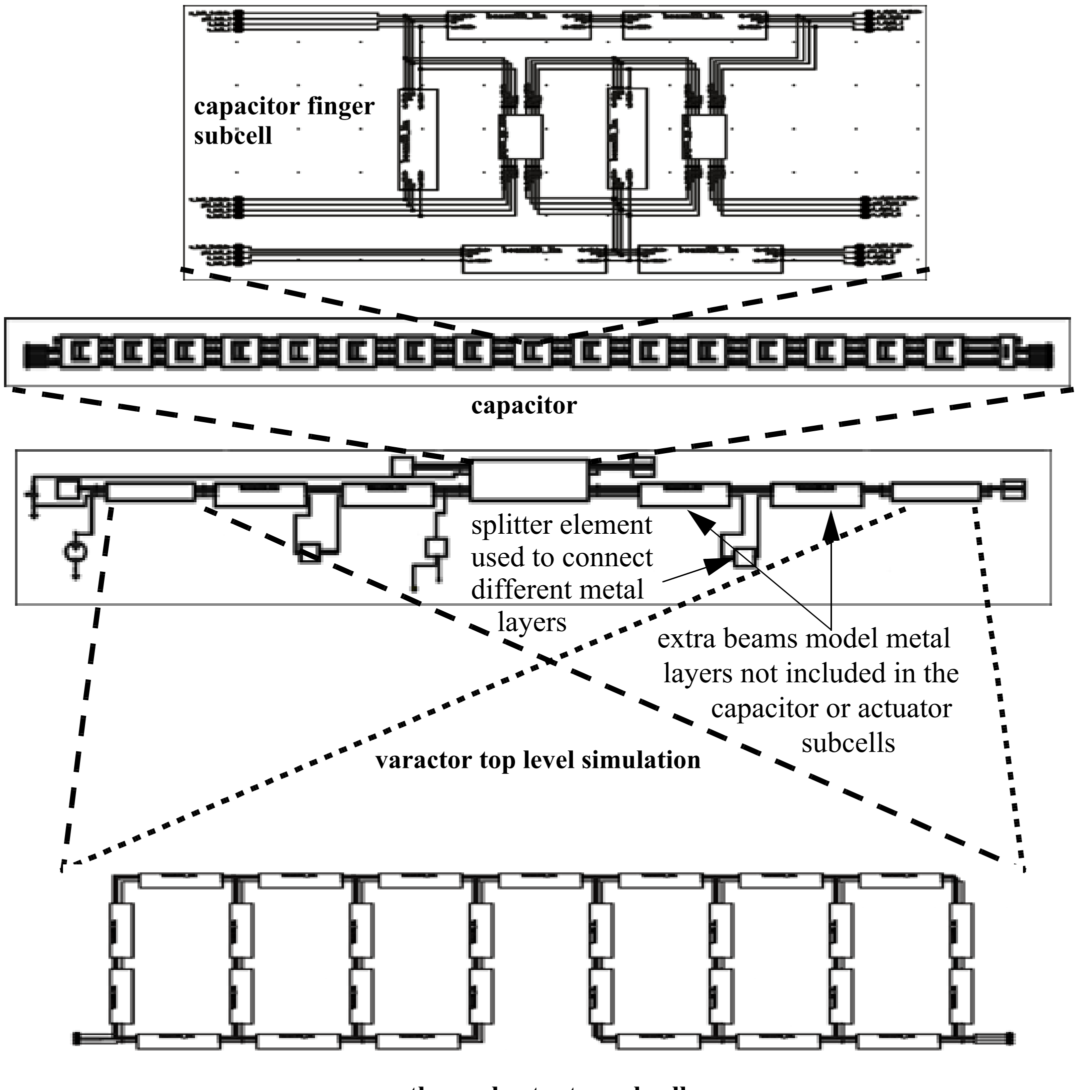

capacitor finger subcell
capacitor
splitter element used to connect different metal layers
extra beams model metal layers not included in the capacitor or actuator subcells
varactor top level simulation
thermal actuator subcell

Figure 4-6. NODAS schematic hierarchy of varactor design in Jazz process.

Both the experimental setup and the NODAS simulation used a single port S-parameter analysis. Therefore, the parameter of interest is S_{11}. From the S-parameter analysis, the quality factor Q, capacitance C, and resistance R can be extracted from their respective definitions in (4.1)-(4.3). The S-parameter analysis in Spectre can also calculate Z-parameters. The Z_{11} parameter was extracted from the S-parameter analysis and was used to calculate the quality factor.

$$Q = \frac{imag(Z_{11})}{real(Z_{11})} \tag{4.1}$$

$$C = \frac{1}{2\pi f(imag(Z_{11}))} \tag{4.2}$$

$$R = real(Z_{11}) \tag{4.3}$$

However, when Z-parameters are not readily available, S-parameters can be converted to Z-parameters. To convert from a single port S-parameter (S_{11}) to a single port Z-parameter (Z_{11}) the equation shown in (4.4) is used. R_s is the source resistance of the port used in the S-parameter analysis.

$$Z_{11} = R_s\frac{1 + S_{11}}{1 - S_{11}} \tag{4.4}$$

The NODAS simulation was set so that one side of the gap (gap_1 in Figure 4-5) was 1 μm and the other side (gap_2 in Figure 4-5) was 5 μm (a capacitance value of approximately 389fF). The experimentally measured quality factor [30] is compared to the quality factor calculated from the NODAS simulation and to a simple RC model developed by A. Oz in Figure 4-7. The RC model is a π-model, where half of the capacitance is lumped to each side of the beam. A single resistor is used to model the resistance of each beam.

The trend of the quality factor plot is similar between the measured and simulated values. The resistance and capacitance values affect the quality factor measurements. For instance, increasing the resistance in the NODAS model, the quality factor will decrease. A more accurate model can be achieved if the exact values of the sheet resistance of the metal layers and the exact gaps between the capacitor beam fingers were known. Also, the measured data does not de-embed the on-chip probe pads, which adds a loss in Q in the measured data. The NODAS and RC models match very closely.

By heating the thermal actuators, the gap between the capacitor beam fingers will change. The initial layout gaps are 2 μm and 4 μm (see Figure 4-8a). The maximum capacitance occurs

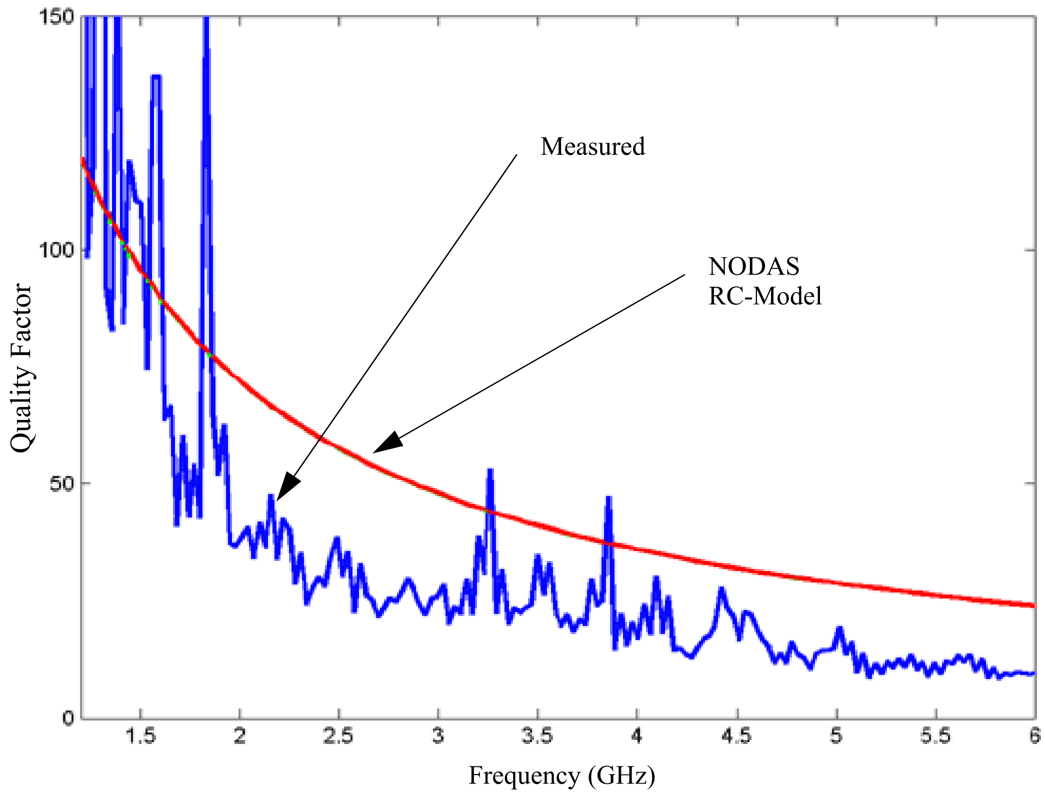


Figure 4-7. Varactor quality factor.
The NODAS gaps were set to 1 μm and 5 μm. Simple RC model and NODAS match.

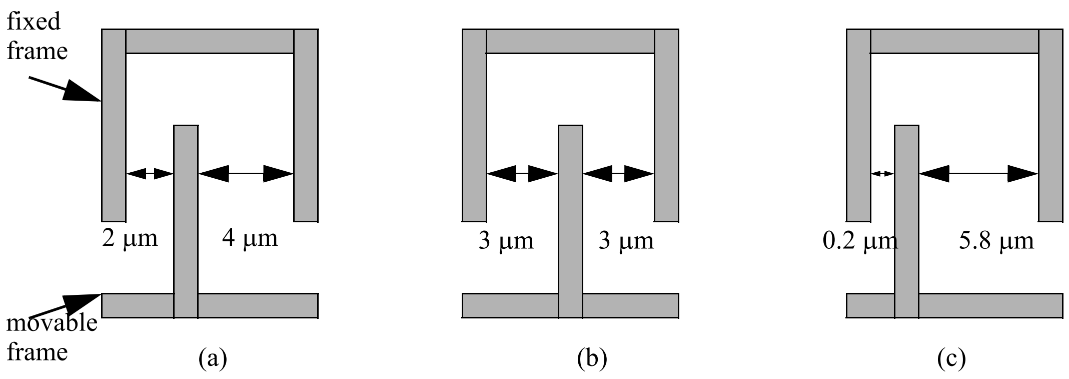


Figure 4-8. Finger displacements.
One beam pair shown. (a) layout, (b) C_{min}, (c) C_{max}.

when the minimum gap is reached on one side (0.2 μm and 5.8 μm, Figure 4-8c). The minimum capacitance occurs when the gaps are equal (3 μm each, Figure 4-8b).

When the structure is released, the beam fingers will move right to it's self-assembled position at room temperature (294K). Using a characteristic temperature (T_0) of 367K and the capacitor dimensions for the 16 beam Jazz process varactor (Figure 4-4), the temperature versus capacitance relationship shown in Table 4-3 can be extracted from a S-parameter analysis. The displacement reported is from the layout configuration (Figure 4-8a). Therefore, the maximum capacitance occurs when the displacement is -1.8 μm and the minimum capacitance occurs when the displacement is 1 μm.

Table 4-3. Capacitance versus temperature for the Jazz 16 beam varactor. T_0 was assumed to be 367K. Capacitance was calculated at 3GHz. The displacement is relative to the layout position. Positive displacements move right and negative displacements move left.

Temperature (K)	Displacement (μm)	Capacitance (fF)	Q	Resistance (Ohm)
294	1.345	230.0	87.8	2.63
300	1.235	228.2	88.5	2.63
312	1.014	226.9	89.1	2.62
325	0.774	228.1	88.6	2.62
350	0.313	237.9	84.2	2.65
375	-0.147	260.8	75.4	2.70
400	-0.608	306.8	62.3	2.77
425	-1.069	406.5	45.2	2.89
450	-1.530	714.0	24.2	3.04
465	-1.806	1611	10.4	3.16

Though the application of a DC temperature source in the simulation environment is one way to cause the actuator to move, the fabricated varactor uses electrothermal heating to raise the temperature of the actuator. The same simulation which produced the results of Table 4-3 was rerun but with a voltage source connected across the poly resistors embedded in the actuators. The poly resistors in this varactor was fabricated with a silicide block, which is used to create higher resistance unsilicided poly resistors. The sheet resistance of silicided poly is approximately 5 ohms/square, whereas unsilicided poly has a sheet resistance of approximately 100 ohms/square. The voltage across the poly resistors was swept from 0V to 4.55V. T_0 was assumed to be 367K, the sheet resistance of unsilicided poly was set to 100 ohms/square. A DC temperature source set to 294K was connected to the anchors of both actuators. This temperature source is a boundary condition for the simulation and acts as a heat sink. Table 4-4 shows the capacitance versus voltage data.

Table 4-4. Capacitance versus voltage for the Jazz 16 beam varactor. T_0 was assumed to be 367K. The sheet resistance of poly was set to 100 ohms/square. Capacitance was calculated at 3GHz. The displacement is relative to the layout position. Positive displacements move right and negative displacements move left. The voltage was swept from 0V to 4.55V.

Voltage (V)	Capacitance (fF)	Displacement (μm)	Q	Resistance (Ohm)
0.00	229.6	1.345	87.9	2.63
0.50	229.1	1.307	88.1	2.63
1.00	227.9	1.193	88.7	2.63
1.50	226.9	1.002	89.1	2.62
2.00	228.5	0.735	88.5	2.63
2.50	235.5	0.392	85.3	2.64
3.00	253.4	-0.027	78.1	2.68
3.50	296.0	-0.523	65.0	2.76
4.00	416.4	-1.095	44.0	2.90
4.50	1255	-1.743	13.5	3.13
4.55	1695	-1.812	9.9	3.17

The plot corresponding capacitance versus voltage plot is shown in Figure 4-9.

4.3 Resonant Mixer

MEMS devices can be used as resonators. Resonators can be found in system level devices such as mixers (Figure 1-2), filters, and accelerometers. In the case of mixers and filters, electrical signals (RF frequency, for example), are applied to the drive electrode of a MEMS device. The signal is coupled electrostatically across an air gap to the resonator which can filter out and mix signals at specific frequencies. The filtered and mixed signals can be read out at the sense electrode. Figure 4-10 shows an SEM of a resonant mixer ("Device 6") designed and tested by J. Brotz and J. Stillman [29].

A first order NODAS schematic of the resonant mixer is shown in Figure 4-11. Even though the drive and sense electrodes are on curl matched frames which can vibrate, this NODAS

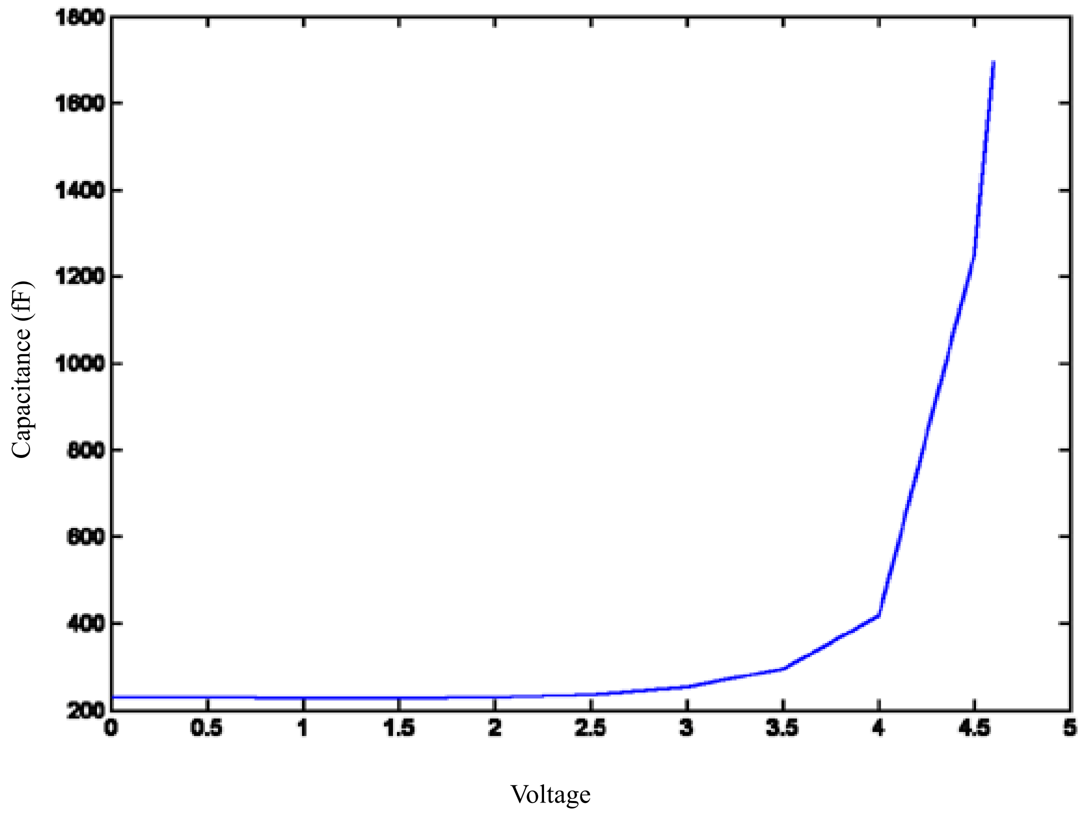
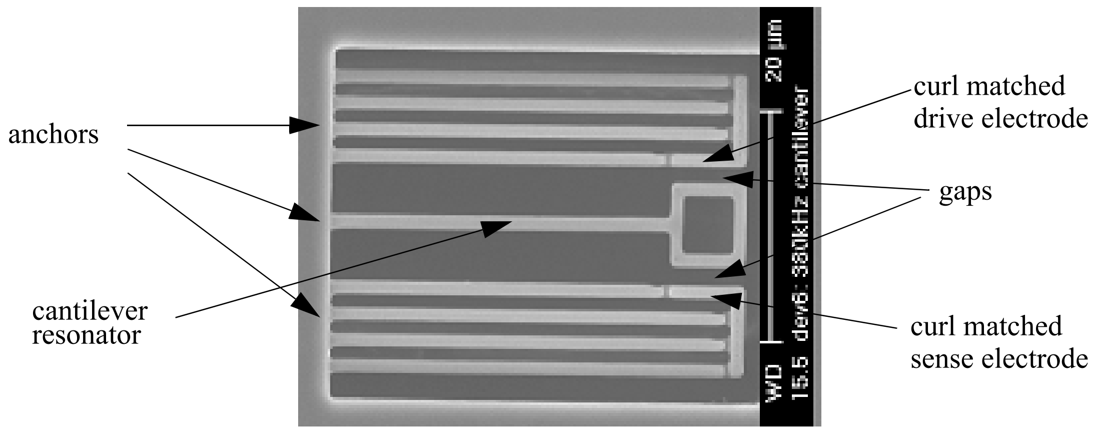


Figure 4-9. Capacitance versus voltage plot of the Jazz 16 beam varactor.



anchors
cantilever resonator
curl matched drive electrode
gaps
curl matched sense electrode
20 µm
WD 15.5 dev6: 380kHz cantilever

Figure 4-10. Resonant Mixer Device 6 NODAS schematic. J. Stillman [29].

model assumes that the drive and sense electrodes are immovable. Therefore, this NODAS schematic only captures the effect of the cantilever device. 2D NODAS linear beams and the full electromechanical gap model were used to model this system. For improved accuracy, 8 NODAS beam segments were used to model the cantilever beam section. In order to probe the displacements at the

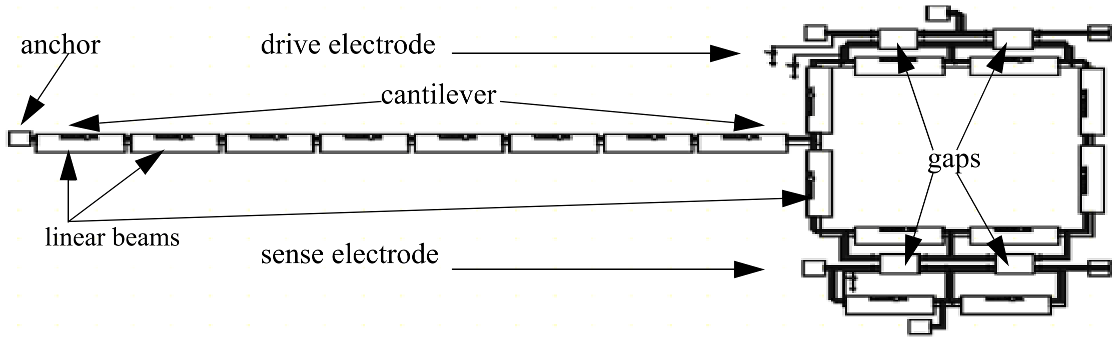


Figure 4-11. Resonant Mixer Device 6 NODAS schematic.

center or each side of the box section, 2 NODAS beam segments were used to model each side of the box.

To compare the results of the NODAS simulation with the experimental measurements, the measured dimensions shown in Table 4-5 were used. The only dimension not measured was the thickness and therefore a thickness of 3.41 μm was used, which is the same thickness used in the hand calculations found in [29].

Table 4-5. Resonant Mixer Device 6 measured quantities.

Parameter	Measured Value
thickness	not measured (3.41 μm used in simulation)
beam width	1.25 μm
cantilever length	32.5 μm
electrode length	7.28 μm
electrode width	1.25 μm
drive gap	1.25 μm
sense gap	1.25 μm
resonant frequency	412060 Hz at 23V DC Bias

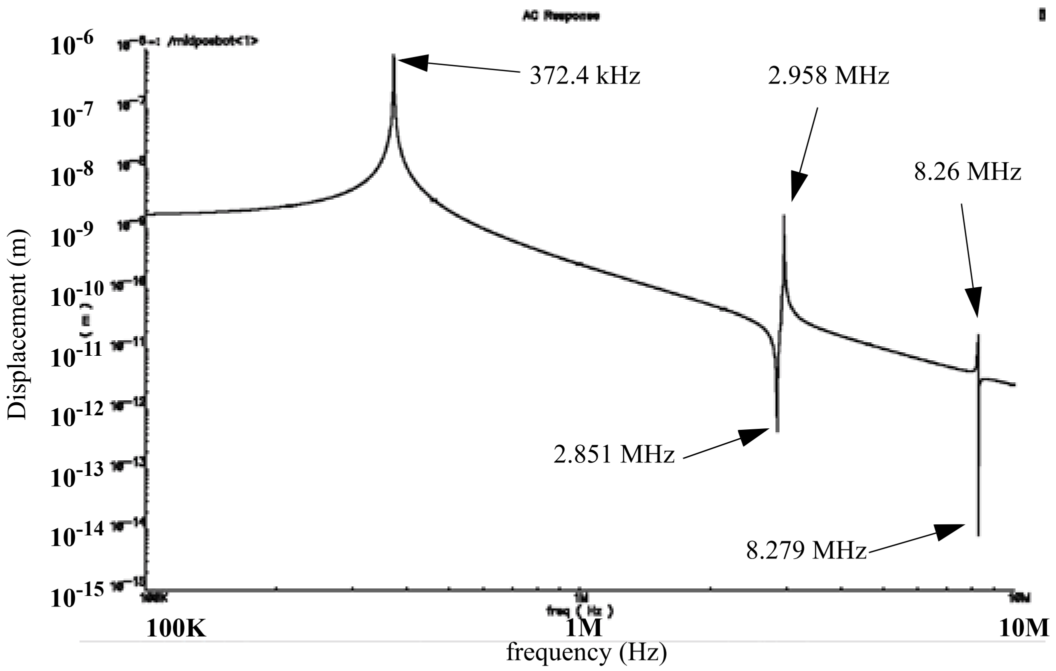


Figure 4-12. Mixer "Device 6" NODAS AC simulation.

Using the dimensions listed in Table 4-5 and the simulation parameters listed in Table 4-6,

Table 4-6. Resonant Mixer Device 6 NODAS simulation settings.

Parameter	Value
DC Bias	23 V
Young's Modulus	62 GPa (estimated)
Density	2496 kg/m^3 (estimated)

the NODAS simulation reports a resonant frequency of 372.4 kHz (Figure 4-12) compared to the measured resonant frequency of 412 kHz The CPU time for a 100 kHz to 1 MHz AC sweep with 500 points per decade is 1.1s on a dual 1GHz UltraSparc III SunFire 280R with 4GB of RAM running SunOS 5.9. The difference in the measured and simulated resonant frequency can be attributed to three unmeasured quantities: the thickness, Young's Modulus, and density. The benefit of using NODAS in a simulator such as Cadence Spectre is that the parameter sweep and the optimizations tools can be used to explore the design space. For instance, running a parameter sweep on the Young's Modulus shows how the resonant frequency changes Figure 4-13. Similarly, the width of

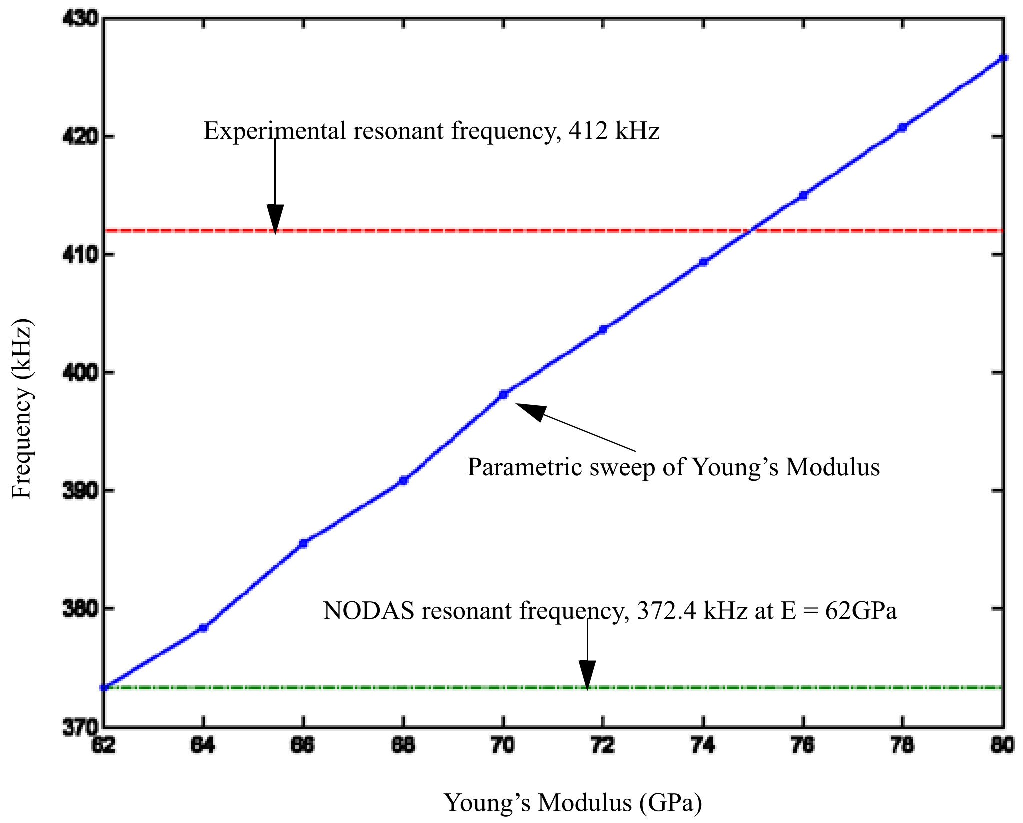


Figure 4-13. Parametric sweep of Young's Modulus.

the beam can be swept (Figure 4-14). A 0.125 μm difference in the beam width (1.25μm vs. 1.375μm) produces a 40 kHz change in resonant frequency. Thus, the inaccuracy of the measured beam width can also contribute to the difference between the NODAS simulation and the experimental results.

To demonstrate mixing, a transient analysis was performed. A 1V 50.373MHz sinusoidal signal was applied to the drive electrode. A 25V 50MHz sinusoidal signal with a 23V DC offset was applied to the cantilever beam. The resulting *y* displacement of the cantilever is shown in Figure 4-15. The right plot shows the transient analysis result and the left plot shows the discrete Fourier transform (DFT) of the transient analysis. The DFT transforms a waveform from the time domain into the frequency domain. The DFT of this simulation shows that there is a DC component as well as several peaks clustered in the 300 kHz to 400 kHz range. The resonant frequency of the mixer

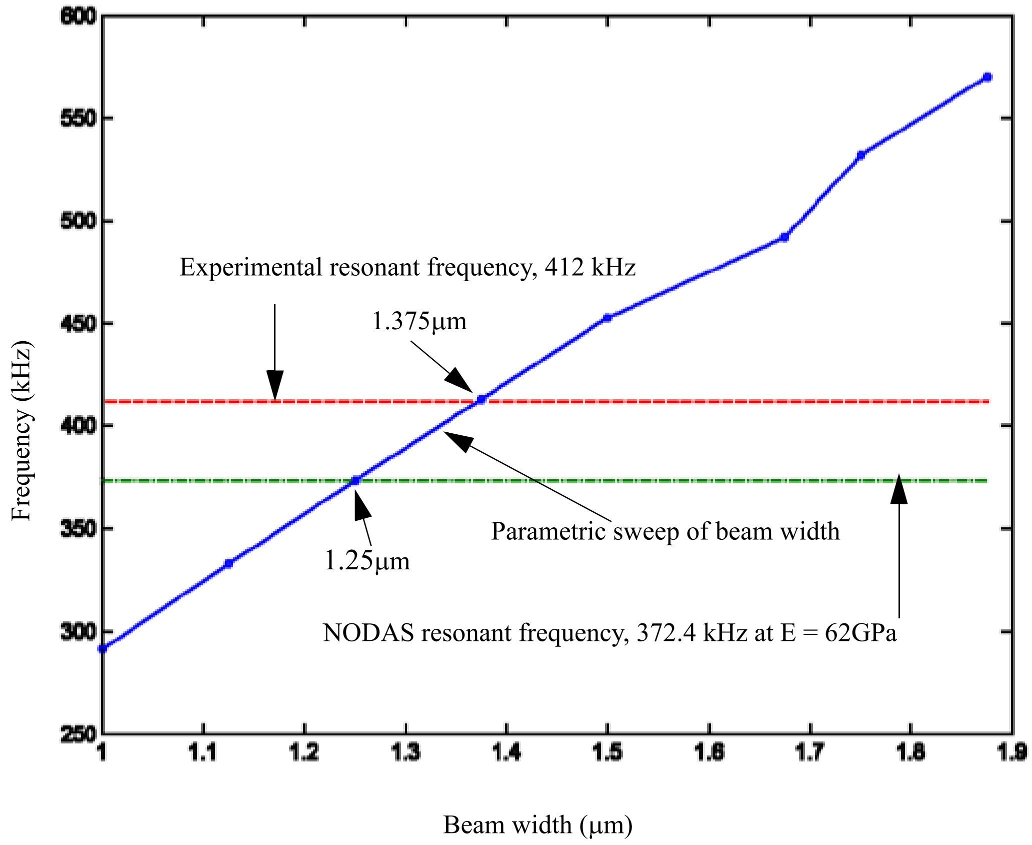


Figure 4-14. Parametric sweep of beam width.

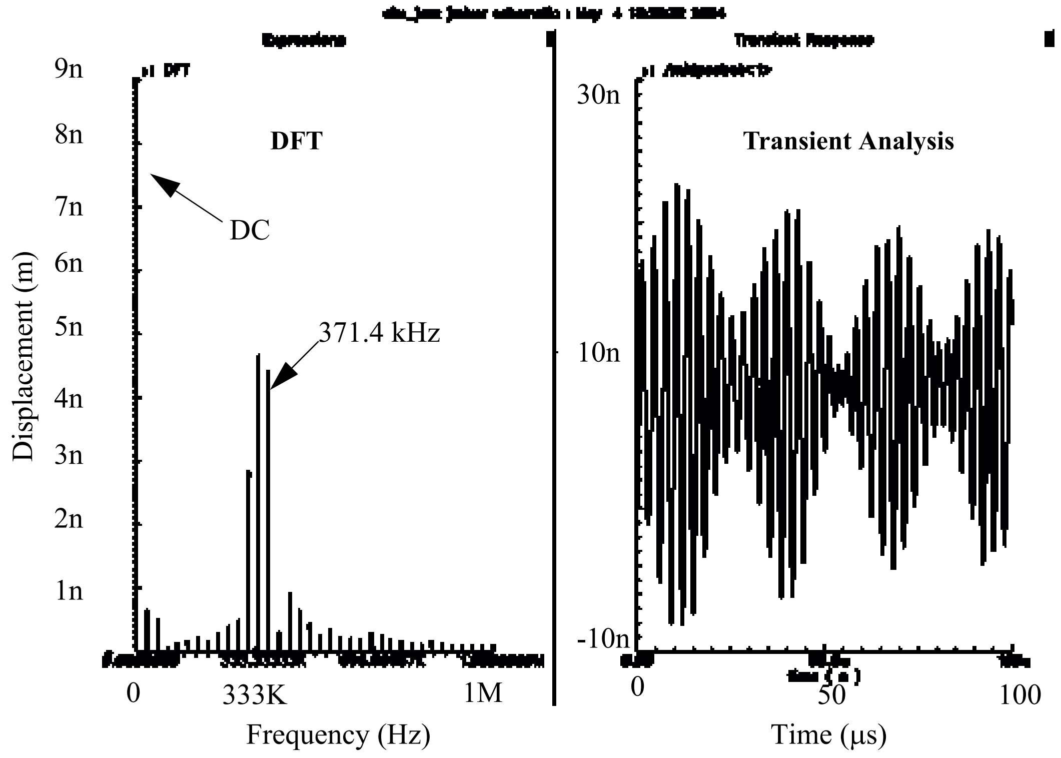


Figure 4-15. Discrete Fourier transform (DFT) of transient simulation of "Device 6".

(as reported by the AC analysis) is 372.4 kHz and the DFT of the transient analysis reports that there is a peak at 371.4 kHz.

The resolution of the DFT depends on two factors: the sampling rate and the size of the time steps in the transient analysis. The sampling rate must be high enough to capture the highest frequency component of the signal (Nyquist rate). Similarly, the maximum time step in the transient analysis should be smaller than the period of the largest frequency component of the signal.

5 Conclusions

5.1 Beam model

The NODAS beam model was updated and new physical effects were added. The inconsistent use of a user parameter in the Verilog-A code which produced a miscalculation in the beam's length was corrected. In-plane moments generated from the different coefficients of expansion in a CMOS-MEMS beam were modeled. A thermal conduction model was also implemented to capture the effects of electrothermal heating and heat conduction.

When compared to analytic equations for a fixed-fixed beam under a uniform distributed load and a fixed-guided beam with axial compressive stress under a uniform distributed load, the NODAS beam model was shown to be accurate to within 6.1×10^{-6} percent and 15×10^{-3} percent respectively when using multiple NODAS beam segments. A NODAS simulation of a multi-layer CMOS-MEMS beam under an applied temperature and with varying metal layer configurations was shown to produce tip deflections to within 2% when compared to the analogous ANSYS simulations.

The thermal actuator and the varactor examples were used to demonstrate the application of the NODAS beam models. The thermal actuator simulation showed that the NODAS thermal beam model was able to capture the thermal moments of an actuator built out of more than one beam. Extracting more accurate numbers for the material properties (specifically the characteristic temperature, T_0, and the thermal coefficients of expansion, α) as well as using measured device dimensions, a more accurate NODAS model could be created. Nevertheless, the goal of creating an accurate thermal beam model was achieved.

5.2 Gap model

Two gap models were developed: an electromechanical and electric-only model. The initial electrostatic gap model was revised to fix the contact model and to address composability issues stemming from the initial implementation of the code used to calculate the gap topology. The model was also renamed as the electromechanical gap model. The electromechanical gap model captures the physics involved when the voltage across the gap creates a force and moment. The electric-only gap model was created to address the large simulation time issue faced when trying to model the area and gap tuning varactor. When many gaps with rotation angles of 0 degrees and 90 degrees were present in a simulation, the large number of nodes, along with the fact that there was an interaction of forces in both the *x* and *y* directions, made the simulation time follow a greater than expected growth rate. The electric-only gap model should only be used when the effects of the electrostatic forces and moments are negligible (i.e. when the structure is very stiff). When fewer gap elements are needed, the electromechanical gap model should be used.

An electrostatically actuated fixed-fixed beam simulation demonstrated that the electrostatic gap model has high accuracy when compared to ANSYS. If 128 NODAS beam/gap segments are used, the displacement of an electrostatically actuated fixed-fixed beam is accurate to within 2.2% when compared with ANSYS. Using 8 NODAS beam/gap segments achieves an accuracy of within 9%. The trade-off of simulation speed versus accuracy was also demonstrated. In the fixed-fixed beam configuration, a 4x increase in speed was achieved using NODAS compared to using the finest mesh possible in ANSYS. The accuracy of the NODAS simulation when compared to the refined mesh ANSYS simulation was still within 4% of the ANSYS simulation result.

The varactor and mixer applications were introduced and it was demonstrated that NODAS was able to simulate the expected behavior of these MEMS devices. If more accurate material properties and dimensions could be obtained and if other physical effects such as fringing electric fields were added to the model, the simulation results would match better with the experimental data.

These two examples show that it is possible to build complex MEMS devices using the basic atomic NODAS beams and gaps.

5.3 Future work

This work focused on enhancements to the 2D beam model and the 2D electrostatic gap model. Near term work should concentrate on generating 3D models for the physics discussed in this thesis. A 3D thermal beam model is a logical and straightforward extension of the 2D thermal beam model. The derivations for the in-plane thermal bending moments can be easily modified to model out-of-plane curling. A 3D electrostatic gap model should be developed. Fringing electric field forces should also be investigated.

A verification suite should be created in order to test new models before being released to the general public. The canonical problems discussed here (fixed-fixed beam with electrostatic actuation and multimorph beam) should be included in such a suite. A table of NODAS simulation results and ANSYS (or any other finite element package) results for these test problems should be created so that the NODAS models can be quickly verified for accuracy. A list of suggested canonical problems follows: cantilever beam using a force source at the free end, cantilever beam using electrostatic actuation, fixed-fixed beam with a force source at the center of the beam, crab-leg resonator, resonant mixer, thermal actuator, and a simple varactor example.

Bibliography

[1] J.R. Gilbert, R. Letenberg, S.D. Senturia, “3D Coupled Electro-Mechanics for MEMS: Applications of CoSolve-EM”, *Micro Electro Mechanical Systems, 1995, Proceedings, IEEE*, pp. 122-127, Jan 29 - Feb. 2, 1995.

[2] ANSYS, http://www.ansys.com/.

[3] FEMLAB, http://www.femlab.com/.

[4] Matlab, http://www.mathworks.com/.

[5] SPICE, http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

[6] H.A.C. Tilmans, “Equivalent circuit representation of electromechanical transducers: I. Lumped-parameter systems”, *Journal of Micromechanics Microengineering (**6** No. 1)*, pp. 157-176, March 1996.

[7] H.A.C. Tilmans, “Equivalent circuit representation of electromechanical transducers: I. Lumped-parameter systems”, Errata, *Journal of Micromechanics Microengineering (**6** No. 3)*, p. 359, September 1996.

[8] H.A.C. Tilmans, “Equivalent circuit representation of electromechanical transducers: II. Distributed-parameter systems”, *Journal of Micromechanics Microengineering (7 No. 4)*, pp. 285-309, December 1997.

[9] *Affirma Verilog-A Language Reference*, Version 4.4.6, Cadence Design Systems, Inc., San Jose, CA, 2001.

[10] Cadence, http://www.cadence.com/

[11] *Affirma Spectre Circuit Simulator User Guide*, Version 4.4.6, Cadence Design Systems, Inc., San Jose, CA, 2000.

[12] Synopsys, http://www.synopsys.com/.

[13] J. V. Clark, N. Zhou, S. Brown and K.S.J. Pister, “Nodal Analysis for MEMS Simulation and Design”, *1998 Int. Conf. on Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuators (MSM ‘98)*, Santa Clara, CA, April 6-8, 1998.

[14] J. V. Clark, N. Zhou, S. Brown, and K.S.J. Pister, “MEMS Simulation Using SUGAR v0.5,” *1998 Solid-State Sensors and Actuators Workshop*, Hilton Head Is., SC, June 7-11, 1998, pp. 191-196.

[15] R. Neul, G. Lorenz and S. Dickmann, “Modeling and Simulation of Microelectromechanical Sensor Systems,” *Microtec 2000*.

[16] G. Lorenz and R. Neul, “Network-Type Modeling of Micromachined Sensor Systems,” *1998 Int. Conf. on Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuators (MSM ‘98)*, Santa Clara, CA, April 6-8, 1998.

[17] Q. Jing, *Modeling and Simulation for Design of Suspended MEMS*, Ph.D. Thesis, Carnegie Mellon University, 2003. (http://www.ece.cmu.edu/~mems/pubs/)

[18] H. Lakdawala, *Temperature Control of CMOS Micromachined Sensors*, Ph.D. Thesis, Carnegie Mellon University, 2002. (http://www.ece.cmu.edu/~mems/pubs/)

[19] S. Iyer, *Modeling and Simulation of Non-idealities in a Z-axis CMOS-MEMS Gyroscope*, Ph.D. thesis, Carnegie Mellon University, 2003. (http://www.ece.cmu.edu/~mems/pubs/)

[20] Q. Jing, T. Mukherjee and G. K. Fedder, “Schematic-Based Lumped Parameterized Behavioral Modeling for Suspended MEMS,” in *Proc. ACM/IEE International Conference on Computer Aided Design (ICCAD ‘02)*, San Jose, CA, November 10-14, 2002, pp. 367-373.

[21] J. Vandemeer, M.S. Kranz, and G. K. Fedder, “Nodal Simulation of Suspended MEMS with Multiple Degrees of Freedom,” *1997 Int. Mechanical Engineering Congress and Exposition: The Winter Annual Meeting of ASME in the 8th Sympostum, on Microelectromechanical Systems (DSC-Vol. 62)*, Dallas, TX, 16-21 November, 1997, pp. 113-118.

[22] G.C. Wong, G.K. Tse, Q. Jing, T. Mukherjee, G.K. Fedder, “Accuracy and Composability in NODAS,” *2003 IEEE International Workshop on Behavioral Modeling and Simulation (BMAS 2003)*, San Jose, CA, October 7-8, 2003.

[23] B. Baidya, *Layout Verification for Mixed-domain Integrated MEMS*, Ph.D. thesis, Carnegie Mellon University, 2003. (http://www.ece.cmu.edu/~mems/pubs/)

[24] B. Baidya, *MEMS Extraction*, MS thesis, Carnegie Mellon University, 1999. (http://www.ece.cmu.edu/~mems/pubs/)

[25] MUMPS, http://www.memsrus.com/.

[26] G.K. Fedder, S. Santhanam, M.L. Reed, S.C. Eagle, D.F.Guillou, M.S.-C. Lu, and L.R. Carley, “Laminated High-Aspect Ratio Microstructures In A Conventional CMOS Process,” *Sensors and Actuators*, vol. A57, no. 2, March 1998, pp. 103-110.

[27] H. Xie, *Gyroscope and Micromirror Design using Vertical-Axis CMOS-MEMS Actuation and Sensing*, Ph.D. Thesis, Carnegie Mellon University, 2002.

[28] X. Zhu, *Post-CMOS Micromachining of Surface and Bulk Structure*s, Ph.D. Thesis, Carnegie Mellon University, 2002.

[29] J. Stillman, CMOS MEMS Resonant Mixer-Filters, MS Thesis, Carnegie Mellon University, 2003. (http://www.ece.cmu.edu/~mems/pubs/)

[30] A. Oz, *CMOS/BICMOS Self-assembling and Electrothermal Microactuators for Tunable Capacitors*, MS Thesis, Carnegie Mellon University, 2003. (http://www.ece.cmu.edu/~mems/pubs/)

[31] S. Timoshenko, *Mechanics of Materials*, Van Nostrand Reinhold Co., New York, New York, 1972.

[32] J.S. Przemieniecki, *Theory of Matrix Structural Analysis*, McGraw-Hill, New York, New York, 1968.

[33] D. Teegarden, G. Lorenz and R. Nuel, "How to Model and Simulate Microgyroscope Systems," *IEEE Spectrum*, pp. 67-75, July 1998.

[34] S. Majumder, J. Lampen, R. Morrison and J. Maciel, "A Packaged, High-Lifetime Ohmic MEMS RF Switch," *Microwave Symposium Digest*, pp. 1935-1938, June 8-13, 2003.

[35] N.S. Barker, G.M. Rebeiz, "Distributed MEMS True-Time Delay Phase Shifters And Wide-Band Switches," *Microwave Theory and Techniques*, pp. 1881-1890, Nov. 1998.

[36] G.-L. Tan, G.M. Rebeiz, "A DC-Contact MEMS Shunt Switch," *Microwave and Wireless Components Letters*, pp. 212-214, June 2002.

[37] E. Fourn, C. Quendo, E. Rius, A. Pothier, P. Blondy, C. Champeaux, J.C. Orlianges, A. Catherinot, G. Tanne, C. Person, F. Huret, "Bandwidth And Central Frequency Control On Tunable Bandpass Filter By Using MEMS Cantilevers," *Microwave Symposium Digest*, pp 523-526, June 8-13, 2003.

[38] C.T.-C. Nguyen, "Transceiver Front-End Architectures Using Vibrating Micromechanical Signal Processors," *Silicon Monolithic Integrated Circuits in RF Systems*, pp. 23-32, Sept. 12-14 2001.

[39] S. Senturia, *Microsystem Design*, Kluwer Academic Press, Boston, 2001.

[40] N. Maluf, *An Introduction to Microelectromechanical Systems Engineering*, Artech House, Boston, 2000.

[41] S. Pamidighantam, R. Puers, K. Baert, and H.A.C. Tilmans, “Pull-in voltage analysis of electrostatically actuated beam structures with fixed-fixed and fixed-free end conditions”, *Journal of Micromechanics Microengineering (**12** No. 4)*, pp. 458-464, July 2002.

[42] J.G. Korvink, J. Funk, M. Roos, G. Wachutka, H. Baltes, “SESES: a comprehensive MEMS modeling system”, *Proceedings, IEEE Workshop on Micro Electro Mechanical Systems, 1994, MEMS ‘94*, pp. 22-27, January 25-28 1994.

[43] G.K.M. Wachutka, “Problem-oriented modeling of coupled physical effects in microtransducers and electronic devices”, *Proceedings, 20th International Conference on Microelectronics*, Volume 2, pp. 539-547, September 12-14 1995.

[44] G. Wachutka, “CAD tools for microdevices and microsystems: today’s demands, potentials and visions”, *2nd International Conference on Advanced Semiconductor Devices and Micrsystems*, pp. 299-309, October 5-7, 1998.

[45] F.P. Incropera, D.P. De Witt, *Fundamentals of Heat and Mass Transfer, Third Edition*, John Wiley & Sons, New York, 1990.

[46] G.K. Fedder, *Simulation of Microelectromechanical Systems*, Ph.D. Thesis, University of California at Berkeley, 1994.

Appendix A: Verilog-A Code

A.1 Electrostatic Gap Model Submodule

```
//(c) Copyright 1999-2003 Carnegie Mellon University
//    All Rights Reserved.
//
//VerilogA for NODAS02_3C_2D, NODAS02_1C_2D gap2D_pp, veriloga
//NODAS Version: 1.1 10/21/2003
//
//Synopsys:
// 2D, rigid, w/ initial angle and rotation, w/o fringing and ground plane.
// displacement and force/moments lumping are based on beam shape functions
// rotated straight electrical field
// 4th-order Taylor-expansion
//
//Revision History:
//Date             Who          What
//07-25-2002     qjing@ece      Created non-bus version
//08-06-2002       khe@ece      Modified to bus version
//01-06-2003    qjing@ece       fixed the asymmetry in x-direction
// What was changed is the last 5-12 lines of the verilogA code, that is, the
// definition of forces flowing though the connection terminals in the chip
// frame. In those forces, force terms for both topology 1 and 0
// should be included. But in the original code, for forces in
// x-direction, only terms for topology 1 were defined. The parts for
// y-direction was correct. That's why we saw asymmetric x-displacements in
// the simulation with the old model. The model is now symmetric.
//10-21-2003    gcwong@ece      new gap model based on rotated parallel plate
//02-10-2004gcwong@ecedivision by 180 needed a 180.0
//02-11-2004gcwong@ecefixed overlap force calculations
//02-17-2004gcwong@eceremoved metal layers from voltage calcs
//03-02-2004gcwong@ececreated electromechanical gap submodule


`include "../../constants.h"
`include "../../discipline.h"


module gap_pp_mech(phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b,
xa_t, xb_b, xb_t);

    inout phia_b;
    rotational phia_b;
    inout phia_t;
    rotational phia_t;
    inout phib_b;
    rotational phib_b;
    inout phib_t;
```

```
rotational phib_t;
inout va_b;
electrical va_b;
inout va_t;
electrical va_t;
inout vb_b;
electrical vb_b;
inout vb_t;
electrical vb_t;
inout [0:1] xa_b;
kinematic [0:1] xa_b;
inout [0:1] xa_t;
kinematic [0:1] xa_t;
inout [0:1] xb_b;
kinematic [0:1] xb_b;
inout [0:1] xb_t;
kinematic [0:1] xb_t;

parameter real bottom_electrode_offset = 0;

parameter real finger_l_t = 0;
parameter real finger_l_b = 0;

parameter real finger_number = 0;
parameter real gap = 0;
parameter real overlap = 0;

// thickness
parameter real thickness = 0;

parameter real angle = 0;

parameter real visc_air = 0;

// remaining gap at which the beams come into contact
parameter real gcontact = 0;

// minimum contact gap
parameter real gmin = 0;

// electic field breakdown (100V/um)
parameter real elec_breakdown = 0;

// squeeze film damping coeff
parameter real Kdamp = 0;

real Felec;// the calculated electrostatic force
real FeMax;// the maximum exectrostatic force
real Fl;// the constant electric force in the overlap region
              // which will be lumped into the beam ends
              // all three are force/length

real ov;  // overlap in rotated frame ~x
real dynOverlap;  // dynamic overlap variable
```

```
    real dy_l;      // change of gap in the local frame
    real dy;     // change of gap in the rotated frame
    real dx;     // change of gap
    real v;    // applied voltage
    real cap;    // capacitance

    real q1, q2;  // charge
    real q;  // charge
    real slope;

    real v_squared;  // v^2

    real Fy_damping;

    // some variables to reduce # of calculations
    real Fsum, var1, var2;

    kinematic V_dy;

    real I_attach, I_tip;
    real L1, L2, ym1, yp1, am1, ap1, ym2, yp2, am2, ap2;
    real Fa1, Fb1, Fa2, Fb2, Fy_1r, Fy_2l, Fy_1l, Fy_2r;
    real xm1, xm2, xp1, xp2, rad, cos_dc, sin_dc;
    real xm1_l, xm2_l, xp1_l, xp2_l, ym1_l, ym2_l, yp1_l, yp2_l;

    real Fxd_l, Fa1_l, Fb1_l, Fa2_l, Fb2_l, Fa1x, Fa1y, Fa2x, Fa2y, Fb1x, Fb1y,
Fb2x, Fb2y;

    real Fy_contact;

    real Ma1_l, Ma2_l, Mb1_l, Mb2_l;

    real ang1mid, ang2mid, theta0;

    //these are the static offsets of the beams, calculated in the initial block
    real offsetL2Right, offsetL2Left, magoffsetL2Right;
    // these are the dynamic offsets of the beams, calculated at each iteration
    real dynoffsetL2Left, dynoffsetL2Right;

    // the endpoints of the overlap region for each beam (beam 1, beam 2)
    real x1l, x1r, x2l, x2r;
    real y1l, y1r, y2l, y2r;

    // define the 8 beam basis functions
    // top beam function.  origin starts at left
    analog function real f11;
        input L,x;
        real L,x;
        f11 = 1.0 - 3.0*pow((x/L),2) + 2.0*pow((x/L),3);
    endfunction

    analog function real f21;
        input L,x;
        real L,x;
        f21 = x - 2.0*(pow(x,2)/L) + pow(x,3)/pow(L,2);
    endfunction

    analog function real f31;
        input L,x;
        real L,x;
        f31 = 3.0*pow((x/L),2) - 2.0*pow((x/L),3);
    endfunction

    analog function real f41;
        input L,x;
        real L,x;
        f41 = -pow(x,2)/L + pow(x,3)/pow(L,2);
    endfunction

    // bottom beam function.  origin starts at right and is flipped about y-axis
    analog function real f12;
        input L,x;
        real L,x;
        f12 = 1.0 - 3.0*pow(((L-x)/L),2) + 2.0*pow(((L-x)/L),3);
    endfunction

    analog function real f22;
        input L,x;
        real L,x;
        f22 = (L-x) - 2.0*(pow((L-x),2)/L) + pow((L-x),3)/pow(L,2);
    endfunction

    analog function real f32;
        input L,x;
        real L,x;
        f32 = 3.0*pow(((L-x)/L),2) - 2.0*pow(((L-x)/L),3);
    endfunction

    analog function real f42;
        input L,x;
        real L,x;
        f42 = -pow((L-x),2)/L + pow((L-x),3)/pow(L,2);
    endfunction

    // define the beam shape functions
    // the shape functions take 7 arguments, ya, yb, phia, phib, L, x
    // ya, yb, phia, phib are the constants multiplied with each of the four basis
    // functions (defined above)
    // L is the length of the beam
    // returns the y value at x

    // top beam function starts from left (i.e. origin at node a)
    analog function real topBeamY;
        input ya, yb, phia, phib, L, x;
        real ya, yb, phia, phib, L, x;
        topBeamY = f11(L,x)*ya + f21(L,x)*phia + f31(L,x)*yb + f41(L,x)*phib;
    endfunction
```

```
// bottom beam function starts from right (i.e. origin at node b)
// positive x is toward the left (i.e. the origin is mirrored about the y-axis)
analog function real bottomBeamY;
    input ya, yb, phia, phib, L, x;
    real ya, yb, phia, phib, L, x;
    bottomBeamY = f12(L,x)*yb + f22(L,x)*phib + f32(L,x)*ya + f42(L,x)*phia;
endfunction

// take the derivative of the basis functions and define the angle function
// as d/dx [y(x)]

// top beam
analog function real df11;
    input L,x;
    real L,x;
    df11 = -6.0*x/pow(L,2) + 6.0*pow(x,2)/pow(L,3);
endfunction

analog function real df21;
    input L,x;
    real L,x;
    df21 = 1.0 - 4.0*x/L + 3.0*pow((x/L),2);
endfunction

analog function real df31;
    input L,x;
    real L,x;
    df31 = 6.0*x/pow(L,2) - 6.0*pow(x,2)/pow(L,3);
endfunction

analog function real df41;
    input L,x;
    real L,x;
    df41 = -2.0*x/L + 3.0*pow((x/L),2);
endfunction

// bottom beam function.  origin starts at right and is flipped about y-axis
analog function real df12;
    input L,x;
    real L,x;
    df12 = 6.0*(L-x)/pow(L,2) - 6.0*pow((L-x),2)/pow(L,3);
endfunction

analog function real df22;
    input L,x;
    real L,x;
    df22 = -1.0 + 4.0*(L-x)/L - 3.0*pow(((L-x)/L),2);
endfunction

analog function real df32;
    input L,x;
    real L,x;
    df32 = -6.0*(L-x)/pow(L,2) + 6.0*pow((L-x),2)/pow(L,3);
```

```
endfunction

analog function real df42;
    input L,x;
    real L,x;
    df42 = 2.0*(L-x)/L - 3.0*pow(((L-x)/L),2);
endfunction

// top beam angle
analog function real topBeamAngle;
    input ya, yb, phia, phib, L, x;
    real ya, yb, phia, phib, L, x;
    topBeamAngle = df11(L,x)*ya + df21(L,x)*phia +
        df31(L,x)*yb + df41(L,x)*phib;
endfunction

// bottom beam angle
analog function real bottomBeamAngle;
    input ya, yb, phia, phib, L, x;
    real ya, yb, phia, phib, L, x;
    bottomBeamAngle = df12(L,x)*yb + df22(L,x)*phib +
        df32(L,x)*ya + df42(L,x)*phia;
endfunction

// take the indefinite integral of the basis functions, used for lumping

// top beam
analog function real if11;
    input L,x;
    real L,x;
    if11 = x - pow(x,3)/pow(L,2) + pow(x,4)/(2.0*pow(L,3));
endfunction

analog function real if21;
    input L,x;
    real L,x;
    if21 = pow(x,2)/2.0 - (2.0/3.0)*pow(x,3)/L + pow(x,4)/(4.0*pow(L,2));
endfunction

analog function real if31;
    input L,x;
    real L,x;
    if31 = pow(x,3)/pow(L,2) - pow(x,4)/(2.0*pow(L,3));
endfunction

analog function real if41;
    input L,x;
    real L,x;
    if41 = -pow(x,3)/(3.0*L) + pow(x,4)/(4.0*pow(L,2));
endfunction

// bottom beam function.  origin starts at right and is flipped about y-axis
analog function real if12;
    input L,x;
```

```
        real L,x;
        if12 = pow(x,3)/pow(L,2) - pow(x,4)/(2.0*pow(L,3));
    endfunction

    analog function real if22;
        input L,x;
        real L,x;
        if22 = (1.0/3.0)*pow(x,3)/L - pow(x,4)/(4.0*pow(L,2));
    endfunction

    analog function real if32;
        input L,x;
        real L,x;
        if32 = x - pow(x,3)/pow(L,2) + pow(x,4)/(2.0*pow(L,3));
    endfunction

    analog function real if42;
        input L,x;
        real L,x;
        if42 = -pow(x,2)/2.0 + (2.0/3.0)*pow(x,3)/L - pow(x,4)/(4.0*pow(L,2));
    endfunction


    analog begin

      @ (initial_step) begin
            // calculate the static offset of the bottom beam
            //
            // the positive direction is in the direction of the arrow
            //                      offsetL2right
            //   top electrode (L2) |+------->|
            //    ******************
            //
            //              bottom electrode (L2)
            //               ******************
            //    |+------->|
            //    bottom_electrode_offset = offsetL2Left
            //    (user parameter)          (internal variable)

            offsetL2Left = bottom_electrode_offset;
          magoffsetL2Right=finger_l_t+finger_l_b-overlap-(abs(offsetL2Left)+over-
lap);

            // rules for the calculation of the static offsetL2Right variable
            if (offsetL2Left >= 0) begin
                if (finger_l_t == overlap) begin
                    offsetL2Right = magoffsetL2Right;
                end
                else if (finger_l_b == overlap) begin
                    offsetL2Right = -magoffsetL2Right;
                end
                else if (overlap > 0) begin
                    offsetL2Right = magoffsetL2Right;
                end
                else begin
                    offsetL2Right = offsetL2Left+finger_l_b-finger_l_t;
                end
            end
            else begin
                if (finger_l_t == overlap) begin
                    offsetL2Right = magoffsetL2Right;
                end
                else if (finger_l_b == overlap) begin
                    offsetL2Right = -magoffsetL2Right;
                end
                else if (overlap > 0) begin
                    offsetL2Right = -magoffsetL2Right;
                end
                else begin
                    offsetL2Right = -(abs(offsetL2Left)+finger_l_t-finger_l_b);
                end
            end

          // convert the angle in degrees to radians
          rad = angle* `M_PI /180.0;
          cos_dc = cos(rad);
          sin_dc = sin(rad);

          // the length of the electrodes (beams)
          L1 = finger_l_t;
          L2 = finger_l_b;

          // Maximum electric field, based on the electric field breakdown
          // per unit length
          FeMax =  -0.5*`eps0*thickness*pow(elec_breakdown,2);

          // slope of the contact force (per unit length)
          slope = FeMax/(gmin - gcontact);

      end

      // rotate into local frame
      // m is the left node (node a), p is the right node (node b)
      // 1 is the top beam, 2 is the bottom beam
      xm1_l = cos_dc*Pos(xa_t[0]) + sin_dc*Pos(xa_t[1]);
      ym1_l = cos_dc*Pos(xa_t[1]) - sin_dc*Pos(xa_t[0]);

      xm2_l = cos_dc*Pos(xa_b[0]) + sin_dc*Pos(xa_b[1]);
      ym2_l = cos_dc*Pos(xa_b[1]) - sin_dc*Pos(xa_b[0]);

      xp1_l = cos_dc*Pos(xb_t[0]) + sin_dc*Pos(xb_t[1]);
      yp1_l = cos_dc*Pos(xb_t[1]) - sin_dc*Pos(xb_t[0]);

      xp2_l = cos_dc*Pos(xb_b[0]) + sin_dc*Pos(xb_b[1]);
      yp2_l = cos_dc*Pos(xb_b[1]) - sin_dc*Pos(xb_b[0]);

      // calculate the dynamic offset
      // recall that the offsets are relative to the top electrode
```

```
// if the top beam moves left (and the bottom beam moves right)
//    offsetL2Left and offsetL2Right increase
// if the top beam moves right (and the bottom beam moves left)
//    offsetL2Left and offsetL2Right decrease
dynoffsetL2Left = offsetL2Left + (-xm1_l+xm2_l);
dynoffsetL2Right = offsetL2Right + (-xp1_l+xp2_l);

// the position variables
xm1 = xm1_l;
ym1 = ym1_l;
xm2 = xm2_l;
ym2 = ym2_l;
xp1 = xp1_l;
yp1 = yp1_l;
xp2 = xp2_l;
yp2 = yp2_l;

// the angles
am1 = Theta(phia_t);
ap1 = Theta(phib_t);
am2 = Theta(phia_b);
ap2 = Theta(phib_b);

// calculate the overlap
dynOverlap = (L1 - abs(dynoffsetL2Left) + L2 - abs(dynoffsetL2Right))/2;
if (dynOverlap > 0) begin
    ov = dynOverlap;
end
else begin
    // there's no overlap
    ov = 0;
end

// find the endpoints of the overlap region (if it exists)
// these end points are relative to the beam's shape functions
// ** if there is no overlap, these end points mean nothing.  Instead of
// setting them to some default number, we just use whatever the logic
// tells us, realizing that in the end, the calculated forces will
// be zero
if (dynoffsetL2Left >=0) begin
    x1l = dynoffsetL2Left;
    x2l = 0;
    if (dynoffsetL2Right >= 0) begin
        x1r = L1;
        x2r = L2-dynoffsetL2Right;
    end
    else begin
        x1r = L1 + dynoffsetL2Right;
        x2r = L2;
    end
end
else begin
    x1l = 0;
    x2l = -dynoffsetL2Left;
    if (dynoffsetL2Right >= 0) begin
        x1r = L1;
        x2r = L2-dynoffsetL2Right;
    end
    else begin
        x1r = L1 + dynoffsetL2Right;
        x2r = L2;
    end
end

// calculate the voltage difference
v = (V(va_t,va_b)+V(vb_t,vb_b))/2.0;


// the square of the applied voltage
v_squared = pow(v,2);

// where the two beams overlap, find the gap that exists between the
// two beams. To do this calculation, we check 4 points on the two beams
// as depiced below. y1l and y1r are the left and right endpoints of the
// overlapping region for beam 1 respectively.  y2l and y2r are the
// left and right endpoints of the overlapping region on beam 2.
//                 |-- y1l
//                 v
// 1  ******************* <-- y1r
//                  (gap)
//        y2l --> *******************  2
//                       ^
//                       |-- y2r

// get y points for each x coordinate above
y1l = topBeamY(ym1, yp1, am1, ap1, L1, x1l);
y1r = topBeamY(ym1, yp1, am1, ap1, L1, x1r);
y2l = topBeamY(ym2, yp2, am2, ap2, L2, x2l);
y2r = topBeamY(ym2, yp2, am2, ap2, L2, x2r);

// get the angle at the midpoint
ang1mid = (topBeamAngle(ym1,yp1,am1,ap1,L1,(x1l+x1r)/2.0));
ang2mid = (topBeamAngle(ym2,yp2,am2,ap2,L2,(x2l+x2r)/2.0));

// the average angle of the beams
theta0 = (ang1mid + ang2mid)/2.0;

// the following calculations only make sense when the overlap
// is non-zero
if (ov > 0) begin
    // find the endpoint which has the smallest gap
    dy_l = min((y1r-y2r+gap),(y1l-y2l+gap));
    dy = dy_l*cos(theta0);

    // The parallel plate force in the overlap region
    // rotated by angle theta. The force is constant through this region
    // force per unit length
    Felec = -0.5*`eps0*thickness*v_squared/pow(dy,2);
```

```
        // the contact force
        if (dy<gcontact) begin
          // contact force
          Fy_contact = slope*(dy-gcontact);
        end
        else begin
          // there is a normal gap, no contact
          Fy_contact = 0;
        end

        // if the electric force per unit length
        // is greater than the max electric force
        // set the force to FeMax
        if (abs(Felec) > abs(FeMax)) begin
            Fl = FeMax;
        end
        else begin
            Fl = Felec;
        end
    end
    else begin
        // there is no parallel plate electric field
        Felec = 0;
        // set this to some reasonable number (such as the original gap)
        dy_l = gap;
        dy = gap;
        // there is no contact force
        Fy_contact = 0;
        // there is no force per unit length
        Fl = 0;
    end

    Pos(V_dy) <+ ddt(dy);
    // damping per unit length
    Fy_damping = Kdamp*visc_air*ov*ov*thickness/pow(dy,3)*Pos(V_dy);

    // variable for the sum of forces (avoids recalculation)
    Fsum = Fl-Fy_contact-Fy_damping;

    // The following 8 equations are lumped forces and moments
    // force balance in the local frame and lump in the local frame
    Fa1_l = -Fsum*(if11(L1,x1r) - if11(L1,x1l));
    Ma1_l = -Fsum*(if21(L1,x1r) - if21(L1,x1l));
    Fb1_l = -Fsum*(if31(L1,x1r) - if31(L1,x1l));
    Mb1_l = -Fsum*(if41(L1,x1r) - if41(L1,x1l));
    Fa2_l = Fsum*(if11(L2,x2r) - if11(L2,x2l));
    Ma2_l = Fsum*(if21(L2,x2r) - if21(L2,x2l));
    Fb2_l = Fsum*(if31(L2,x2r) - if31(L2,x2l));
    Mb2_l = Fsum*(if41(L2,x2r) - if41(L2,x2l));

    // variable to reduce recalculation
    var1 = `eps0*thickness/(dy);
    // total capacitance formed by the overlap of the electrodes
```

```
cap = var1*ov;

// Fex = 1/2*(d/dov)(cap)*V^2
if ((ov != L1) && (ov != L2) && (ov > 0)) begin
    // the overlap is not equal to L1 or L2
    Fxd_l = 0.5*v_squared*var1;

    // transform the forces in the displaced frame back to the local
    // frame we must apply the x force in the y direction
    // since we lump the x force to one end, we can do a simple mapping
    // for now:
    //  if neither beam is completely overlapped by the other beam then:
    //      - if the top beam is to the left of the bottom beam, then
    //      we assign the forces to the right of the top beam and to the
    //      left of the bottom beam
    //      - otherwise we do the opposite
    // same goes for x
    if ((dynoffsetL2Left > 0) && (dynoffsetL2Right > 0)) begin
        // x forces go to top beam right, bottom beam left
        Fa1x = -sin(theta0)*Fa1_l;
        Fb1x = cos(theta0)*Fxd_l-sin(theta0)*Fb1_l;
        Fa2x = cos(theta0)*(-Fxd_l)-sin(theta0)*Fa2_l;
        Fb2x = -sin(theta0)*Fb2_l;
        Fy_1l = cos(theta0)*Fa1_l;
        Fy_1r = sin(theta0)*Fxd_l + cos(theta0)*Fb1_l;
        Fy_2l = sin(theta0)*Fxd_l + cos(theta0)*Fa2_l;
        Fy_2r = cos(theta0)*Fb2_l;
    end
    else begin
        // x forces go to top beam left, bottom beam right
        Fa1x = cos(theta0)*(-Fxd_l)-sin(theta0)*Fa1_l;
        Fb1x = -sin(theta0)*Fb1_l;
        Fa2x = -sin(theta0)*Fa2_l;
        Fb2x = cos(theta0)*Fxd_l-sin(theta0)*Fb2_l;
        Fy_1l = sin(theta0)*(-Fxd_l) + cos(theta0)*Fa1_l;
        Fy_1r = cos(theta0)*Fb1_l;
        Fy_2l = cos(theta0)*Fa2_l;
        Fy_2r = sin(theta0)*Fxd_l + cos(theta0)*Fb2_l;
    end
end
else begin
    // the overlap is equal to L1 or L2
    Fxd_l = 0;
    Fa1x = 0;
    Fb1x = 0;
    Fa2x = 0;
    Fb2x = 0;
    // no x component for the y forces
    Fy_1r = cos(theta0)*Fb1_l;
    Fy_1l = cos(theta0)*Fa1_l;
    Fy_2r = cos(theta0)*Fb2_l;
    Fy_2l = cos(theta0)*Fa2_l;
end
```

```
      // determine the charge
      q = v * cap;

      var2 = ov/2.0/finger_l_t;
      I_tip    = ddt(q)*(1-var2);
      I_attach = ddt(q)*var2;

      // the tip and the attached points depend on the dynoffset vars
      if (dynoffsetL2Left >=0) begin
          I(vb_t,va_b) <+ I_tip;
          I(va_t,vb_b) <+ I_attach;
      end
      else begin
          I(vb_t,va_b) <+ I_attach;
          I(va_t,vb_b) <+ I_tip;
      end

      // rotate back from local to chip frame
      F(xa_t[0]) <+  (cos_dc*Fa1x-sin_dc*Fy_1l)*finger_number;
      F(xb_t[0]) <+  (cos_dc*Fb1x-sin_dc*Fy_1r)*finger_number;
      F(xa_b[0]) <+  (cos_dc*Fa2x-sin_dc*Fy_2l)*finger_number;
      F(xb_b[0]) <+  (cos_dc*Fb2x-sin_dc*Fy_2r)*finger_number;

      F(xa_t[1]) <+  (sin_dc*Fa1x+cos_dc*Fy_1l)*finger_number;
      F(xb_t[1]) <+  (sin_dc*Fb1x+cos_dc*Fy_1r)*finger_number;
      F(xa_b[1]) <+  (sin_dc*Fa2x+cos_dc*Fy_2l)*finger_number;
      F(xb_b[1]) <+  (sin_dc*Fb2x+cos_dc*Fy_2r)*finger_number;

      Tau(phia_t) <+  Ma1_l*finger_number;
      Tau(phib_t) <+  Mb1_l*finger_number;
      Tau(phia_b) <+  Ma2_l*finger_number;
      Tau(phib_b) <+  Mb2_l*finger_number;

    end
endmodule
```

A.2 Electric Only Gap Model Submodule

```
//(c) Copyright 1999-2003 Carnegie Mellon University
//    All Rights Reserved.
//
//VerilogA for NODAS02_3C_2D, gap2D_pp_elec, veriloga
//NODAS Version: 1.1 10/21/2003
//
//Synopsys:
// 2D, rigid, w/ initial angle and rotation, w/o fringing and ground plane.
// displacement and force/moments lumping are based on beam shape functions
// rotated straight electrical field
// 4th-order Taylor-expansion
//
//Revision History:
```

```
//Date             Who          What
//07-25-2002     qjing@ece      Created non-bus version
//08-06-2002       khe@ece      Modified to bus version
//01-06-2003    qjing@ece       fixed the asymmetry in x-direction
// What was changed is the last 5-12 lines of the verilogA code, that is, the
// definition of forces flowing though the connection terminals in the chip
// frame. In those forces, force terms for both topology 1 and 0
// should be included. But in the original code, for forces in
// x-direction, only terms for topology 1 were defined. The parts for
// y-direction was correct. That's why we saw asymmetric x-displacements in
// the simulation with the old model. The model is now symmetric.
//10-21-2003    gcwong@ece      new gap model based on rotated parallel plate
//02-10-2004    gcwong@ece      forked electrostatic and
//                      electrostatic/mechanical models
//02-17-2004    gcwong@ece      changed module name
//                      removed bottom beam functions
//                      removed metal layers from voltage calcs
//03-02-2004gcwong@ececreated electrical only submodule

`include "../../constants.h"
`include "../../discipline.h"


module gap_pp_elec(phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b,
xa_t, xb_b, xb_t);

    inout phia_b;
    rotational phia_b;
    inout phia_t;
    rotational phia_t;
    inout phib_b;
    rotational phib_b;
    inout phib_t;
    rotational phib_t;
    inout va_b;
    electrical va_b;
    inout va_t;
    electrical va_t;
    inout vb_b;
    electrical vb_b;
    inout vb_t;
    electrical vb_t;
    inout [0:1] xa_b;
    kinematic [0:1] xa_b;
    inout [0:1] xa_t;
    kinematic [0:1] xa_t;
    inout [0:1] xb_b;
    kinematic [0:1] xb_b;
    inout [0:1] xb_t;
    kinematic [0:1] xb_t;

    parameter real bottom_electrode_offset = 0;

    parameter real finger_l_t = 0;
    parameter real finger_l_b = 0;

    parameter real finger_number = 0;
    parameter real gap = 0;
    parameter real overlap = 0;

    // thickness
    parameter real thickness = 0;

    parameter real angle = 0;

    parameter real visc_air = 0;

    // remaining gap at which the beams come into contact
    parameter real gcontact = 0;

    real ov;  // overlap in rotated frame ~x
    real dynOverlap;  // dynamic overlap variable

    real dy_l;     // change of gap in the local frame
    real dy;     // change of gap in the rotated frame
    real dx;     // change of gap
    real v;    // applied voltage
    real cap;    // capacitance

    real q;  // charge

    real v_squared;  // v^2

    // some variables to reduce # of calculations
    real var1, var2;

    real I_attach, I_tip;
    real L1, L2, ym1, yp1, am1, ap1, ym2, yp2, am2, ap2;
    real xm1, xm2, xp1, xp2, rad, cos_dc, sin_dc;
    real xm1_l, xm2_l, xp1_l, xp2_l, ym1_l, ym2_l, yp1_l, yp2_l;

    real ang1mid, ang2mid, theta0;

    //these are the static offsets of the beams, calculated in the initial block
    real offsetL2Right, offsetL2Left, magoffsetL2Right;
    // these are the dynamic offsets of the beams, calculated at each iteration
    real dynoffsetL2Left, dynoffsetL2Right;

    // the endpoints of the overlap region for each beam (beam 1, beam 2)
    real x1l, x1r, x2l, x2r;
    real y1l, y1r, y2l, y2r;

    // define the 4 beam basis functions
    // top beam function.  origin starts at left
    analog function real f11;
        input L,x;
        real L,x;
        f11 = 1.0 - 3.0*pow((x/L),2) + 2.0*pow((x/L),3);
```

```
endfunction

analog function real f21;
    input L,x;
    real L,x;
    f21 = x - 2.0*(pow(x,2)/L) + pow(x,3)/pow(L,2);
endfunction

analog function real f31;
    input L,x;
    real L,x;
    f31 = 3.0*pow((x/L),2) - 2.0*pow((x/L),3);
endfunction

analog function real f41;
    input L,x;
    real L,x;
    f41 = -pow(x,2)/L + pow(x,3)/pow(L,2);
endfunction

// define the beam shape functions
// the shape functions take 7 arguments, ya, yb, phia, phib, L, x
//ya,yb, phia, phib are the constants multiplied with each of the four basis
// functions (defined above)
// L is the length of the beam
// returns the y value at x

// top beam function starts from left (i.e. origin at node a)
analog function real topBeamY;
    input ya, yb, phia, phib, L, x;
    real ya, yb, phia, phib, L, x;
    topBeamY = f11(L,x)*ya + f21(L,x)*phia + f31(L,x)*yb + f41(L,x)*phib;
endfunction

// take the derivative of the basis functions and define the angle function
// as d/dx [y(x)]

// top beam
analog function real df11;
    input L,x;
    real L,x;
    df11 = -6.0*x/pow(L,2) + 6.0*pow(x,2)/pow(L,3);
endfunction

analog function real df21;
    input L,x;
    real L,x;
    df21 = 1.0 - 4.0*x/L + 3.0*pow((x/L),2);
endfunction

analog function real df31;
    input L,x;
    real L,x;
    df31 = 6.0*x/pow(L,2) - 6.0*pow(x,2)/pow(L,3);
endfunction

analog function real df41;
    input L,x;
    real L,x;
    df41 = -2.0*x/L + 3.0*pow((x/L),2);
endfunction

// top beam angle
analog function real topBeamAngle;
    input ya, yb, phia, phib, L, x;
    real ya, yb, phia, phib, L, x;
    topBeamAngle = df11(L,x)*ya + df21(L,x)*phia +
        df31(L,x)*yb + df41(L,x)*phib;
endfunction

analog begin

  @ (initial_step) begin
      // calculate the static offset of the bottom beam
      //
      // the positive direction is in the direction of the arrow
      //                      offsetL2right
      //   top electrode (L2) |+------->|
      //    ******************
      //
      //                bottom electrode (L2)
      //                 ******************
      //    |+------->|
      //    bottom_electrode_offset = offsetL2Left
      //    (user parameter)          (internal variable)

      offsetL2Left = bottom_electrode_offset;
      magoffsetL2Right=finger_l_t+finger_l_b-overlap-(abs(offsetL2Left)+over-
lap);

      // rules for the calculation of the static offsetL2Right variable
      if (offsetL2Left >= 0) begin
          if (finger_l_t == overlap) begin
              offsetL2Right = magoffsetL2Right;
          end
          else if (finger_l_b == overlap) begin
              offsetL2Right = -magoffsetL2Right;
          end
          else if (overlap > 0) begin
              offsetL2Right = magoffsetL2Right;
          end
          else begin
              offsetL2Right = offsetL2Left+finger_l_b-finger_l_t;
          end
      end
      else begin
          if (finger_l_t == overlap) begin
              offsetL2Right = magoffsetL2Right;
          end
          else if (finger_l_b == overlap) begin
              offsetL2Right = -magoffsetL2Right;
          end
          else if (overlap > 0) begin
              offsetL2Right = -magoffsetL2Right;
          end
          else begin
              offsetL2Right = -(abs(offsetL2Left)+finger_l_t-finger_l_b);
          end
      end

      // convert the angle in degrees to radians
      rad = angle* `M_PI /180.0;
      cos_dc = cos(rad);
      sin_dc = sin(rad);

      // the length of the electrodes (beams)
      L1 = finger_l_t;
      L2 = finger_l_b;

  end

  // rotate into local frame
  // m is the left node (node a), p is the right node (node b)
  // 1 is the top beam, 2 is the bottom beam
  xm1_l = cos_dc*Pos(xa_t[0]) + sin_dc*Pos(xa_t[1]);
  ym1_l = cos_dc*Pos(xa_t[1]) - sin_dc*Pos(xa_t[0]);

  xm2_l = cos_dc*Pos(xa_b[0]) + sin_dc*Pos(xa_b[1]);
  ym2_l = cos_dc*Pos(xa_b[1]) - sin_dc*Pos(xa_b[0]);

  xp1_l = cos_dc*Pos(xb_t[0]) + sin_dc*Pos(xb_t[1]);
  yp1_l = cos_dc*Pos(xb_t[1]) - sin_dc*Pos(xb_t[0]);

  xp2_l = cos_dc*Pos(xb_b[0]) + sin_dc*Pos(xb_b[1]);
  yp2_l = cos_dc*Pos(xb_b[1]) - sin_dc*Pos(xb_b[0]);

  // calculate the dynamic offset
  // recall that the offsets are relative to the top electrode
  // if the top beam moves left (and the bottom beam moves right)
  //    offsetL2Left and offsetL2Right increase
  // if the top beam moves right (and the bottom beam moves left)
  //    offsetL2Left and offsetL2Right decrease
  dynoffsetL2Left = offsetL2Left + (-xm1_l+xm2_l);
  dynoffsetL2Right = offsetL2Right + (-xp1_l+xp2_l);

  // the position variables
  xm1 = xm1_l;
  ym1 = ym1_l;
  xm2 = xm2_l;
  ym2 = ym2_l;
  xp1 = xp1_l;
  yp1 = yp1_l;
```

```
xp2 = xp2_l;
yp2 = yp2_l;

// the angles
am1 = Theta(phia_t);
ap1 = Theta(phib_t);
am2 = Theta(phia_b);
ap2 = Theta(phib_b);

// calculate the overlap
dynOverlap = (L1 - abs(dynoffsetL2Left) + L2 - abs(dynoffsetL2Right))/2;
if (dynOverlap > 0) begin
    ov = dynOverlap;
end
else begin
    // there's no overlap
    ov = 0;
end

// find the endpoints of the overlap region (if it exists)
// these end points are relative to the beam's shape functions
if (dynoffsetL2Left >=0) begin
      x1l = dynoffsetL2Left;
      x2l = 0;
      if (dynoffsetL2Right >= 0) begin
          x1r = L1;
          x2r = L2-dynoffsetL2Right;
      end
      else begin
          x1r = L1 + dynoffsetL2Right;
          x2r = L2;
      end
  end
  else begin
      x1l = 0;
      x2l = -dynoffsetL2Left;
      if (dynoffsetL2Right >= 0) begin
          x1r = L1;
          x2r = L2-dynoffsetL2Right;
      end
      else begin
          x1r = L1 + dynoffsetL2Right;
          x2r = L2;
      end
  end

// where the two beams overlap, find the gap that exists between the
// two beams. To do this calculation, we check 4 points on the two beams
//as depicted below. y1l and y1r are the left and right endpoints of the
// overlapping region for beam 1 respectively.  y2l and y2r are the
// left and right endpoints of the overlapping region on beam 2.
//                |-- y1l
//                v
// 1  ****************** <-- y1r
//                 (gap)
//        y2l --> *******************  2
//                      ^
//                      |-- y2r

// get y points for each x coordinate above
y1l = topBeamY(ym1, yp1, am1, ap1, L1, x1l);
y1r = topBeamY(ym1, yp1, am1, ap1, L1, x1r);
y2l = topBeamY(ym2, yp2, am2, ap2, L2, x2l);
y2r = topBeamY(ym2, yp2, am2, ap2, L2, x2r);

// get the angle at the midpoint
ang1mid = (topBeamAngle(ym1,yp1,am1,ap1,L1,(x1l+x1r)/2.0));
ang2mid = (topBeamAngle(ym2,yp2,am2,ap2,L2,(x2l+x2r)/2.0));

// the average angle of the beams
theta0 = (ang1mid + ang2mid)/2.0;

// find the endpoint which has the smallest gap
dy_l = min((y1r-y2r+gap),(y1l-y2l+gap));
dy = dy_l*cos(theta0);

// calculate the voltage difference
v = (V(va_t,va_b)+V(vb_t,vb_b))/2.0;

// the square of the applied voltage
v_squared = pow(v,2);

// warn user if the gap disappears
if (dy<gcontact) begin
    $display("A gap no longer exists!");
end

// variable to reduce recalculation
var1 = `eps0*thickness/(dy);
// total capacitance formed by the overlap of the electrodes
cap = var1*ov;

// determine the charge
q = v * cap;

var2 = ov/2.0/finger_l_t;
I_tip    = ddt(q)*(1-var2);
I_attach = ddt(q)*var2;

  // the tip and the attached points depend on the dynoffset vars
  if (dynoffsetL2Left >=0) begin
      I(vb_t,va_b) <+ I_tip;
      I(va_t,vb_b) <+ I_attach;
  end
  else begin
      I(vb_t,va_b) <+ I_attach;
      I(va_t,vb_b) <+ I_tip;
  end
```

```
    end
endmodule
```

A.3 Three Conductor Electrostatic Gap Model

```
//(c) Copyright 1999-2003 Carnegie Mellon University
//    All Rights Reserved.
//
//VerilogA for NODAS02_3C_2D, gap2D_pp, veriloga
//NODAS Version: 1.1 10/21/2003
//
//Synopsys:
// 2D, rigid, w/ initial angle and rotation, w/o fringing and ground plane.
// displacement and force/moments lumping are based on beam shape functions
// rotated straight electrical field
// 4th-order Taylor-expansion
//
//Revision History:
//Date            Who         What
//07-25-2002     qjing@ece      Created non-bus version
//08-06-2002       khe@ece      Modified to bus version
//01-06-2003    qjing@ece      fixed the asymmetry in x-direction
// What was changed is the last 5-12 lines of the verilogA code, that is, the
// definition of forces flowing though the connection terminals in the chip
// frame. In those forces, force terms for both topology 1 and 0
// should be included. But in the original code, for forces in
// x-direction, only terms for topology 1 were defined. The parts for
// y-direction was correct. That's why we saw asymmetric x-displacements in
// the simulation with the old model. The model is now symmetric.
//10-21-2003    gcwong@ece      new gap model based on rotated parallel plate
//02-10-2004gcwong@ecedivision by 180 needed a 180.0
//02-11-2004gcwong@ecefixed overlap force calculations
//02-17-2004gcwong@eceremoved metal layers from voltage calcs
//03-02-2004gcwong@eceused submodule


`include "../../constants.h"
`include "../../discipline.h"
`include "../../process.h"
`include "../../design.h"


module gap2D_pp(phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b, xa_t,
xb_b, xb_t, ta_b, ta_t, tb_b, tb_t);

    inout phia_b;
    rotational phia_b;
    inout phia_t;
    rotational phia_t;
    inout phib_b;
    rotational phib_b;
```

```
inout phib_t;
rotational phib_t;
inout va_b;
electrical va_b;
inout va_t;
electrical va_t;
inout vb_b;
electrical vb_b;
inout vb_t;
electrical vb_t;
inout [0:1] xa_b;
kinematic [0:1] xa_b;
inout [0:1] xa_t;
kinematic [0:1] xa_t;
inout [0:1] xb_b;
kinematic [0:1] xb_b;
inout [0:1] xb_t;
kinematic [0:1] xb_t;
inout ta_b;
thermal ta_b;
inout ta_t;
thermal ta_t;
inout tb_b;
thermal tb_b;
inout tb_t;
thermal tb_t;

parameter real bottom_electrode_offset = `default_bottom_electrode_offset;

parameter real finger_l_t = `default_finger_l_t;
parameter real finger_l_b = `default_finger_l_b;

parameter real finger_number = `default_finger_number;
parameter real gap = `default_gap;
parameter real overlap = `default_gap_overlap;

// thicknesses used for calculation of total beam thickness and
// Young's modulus
parameter real t_m3 = `default_t_m3;
parameter real t_m2 = `default_t_m2;
parameter real t_m1 = `default_t_m1;
parameter real t_poly = `default_t_poly;

parameter real   t_m3_m2 = `default_t_m3_m2;
parameter real   t_m3_m1_overpoly = `default_t_m3_m1_overpoly;
parameter real   t_m3_m1_overfield = `default_t_m3_m1_overfield;
parameter real   t_m3_poly = `default_t_m3_poly;
parameter real   t_m3_sub = `default_t_m3_sub;
parameter real   t_m2_m1_overpoly = `default_t_m2_m1_overpoly;
parameter real   t_m2_m1_overfield = `default_t_m2_m1_overfield;
parameter real   t_m2_poly = `default_t_m2_poly;
parameter real   t_m2_sub = `default_t_m2_sub;
parameter real   t_m1_poly = `default_t_m1_poly;
parameter real   t_m1_sub = `default_t_m1_sub;
```

```
    parameter real   t_poly_sub = `default_t_poly_sub;

    parameter integer flag_m3 = 1;
    parameter integer flag_m2 = 1;
    parameter integer flag_m1 = 1;
    parameter integer flag_poly = 1;

    // thickness
    parameter real thickness = flag_m3*t_m3 + flag_m2*t_m2
      +flag_m1*t_m1 + flag_poly*t_poly
      +flag_m3*flag_m2*t_m3_m2
      +flag_m3*(1-flag_m2)*flag_m1*flag_poly*t_m3_m1_overpoly
      +flag_m3*(1-flag_m2)*flag_m1*(1-flag_poly)*t_m3_m1_overfield
      +flag_m3*(1-flag_m2)*(1-flag_m1)*flag_poly*t_m3_poly
      +flag_m3*(1-flag_m2)*(1-flag_m1)*(1-flag_poly)*t_m3_sub
      +flag_m2*flag_m1*flag_poly*t_m2_m1_overpoly
      +flag_m2*flag_m1*(1-flag_poly)*t_m2_m1_overfield
      +flag_m2*(1-flag_m1)*flag_poly*t_m2_poly
      +flag_m2*(1-flag_m1)*(1-flag_poly)*t_m2_sub
      +flag_m1*flag_poly*t_m1_poly
      +flag_m1*(1-flag_poly)*t_m1_sub
      +flag_poly*t_poly_sub;

    parameter real angle = `default_gap_angle;

    parameter real visc_air = `default_visc_air;

    // remaining gap at which the beams come into contact
    parameter real gcontact = `default_gcontact;

    // minimum contact gap
    parameter real gmin = `default_gmin;

    // electic field breakdown (100V/um)
    parameter real elec_breakdown = `default_elec_breakdown;

    // squeeze film damping coeff
    parameter real Kdamp = `default_Kdamp;

    gap_pp_mech # ( .bottom_electrode_offset(bottom_electrode_offset),
.finger_l_t(finger_l_t), .finger_l_b(finger_l_b), .finger_number(finger_number),
.gap(gap), .overlap(overlap), .thickness(thickness), .angle(angle),
.visc_air(visc_air), .gcontact(gcontact), .gmin(gmin),
.elec_breakdown(elec_breakdown), .Kdamp(Kdamp) )
      gap_pp_mech_1 (phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b,
xa_t, xb_b, xb_t);

endmodule
```

A.4 Three Conductor Electric Only Gap Model

```
//(c) Copyright 1999-2003 Carnegie Mellon University
//    All Rights Reserved.
//
//VerilogA for NODAS02_3C_2D, gap2D_pp_elec, veriloga
//NODAS Version: 1.1 10/21/2003
//
//Synopsys:
// 2D, rigid, w/ initial angle and rotation, w/o fringing and ground plane.
// displacement and force/moments lumping are based on beam shape functions
// rotated straight electrical field
// 4th-order Taylor-expansion
//
//Revision History:
//Date             Who          What
//07-25-2002     qjing@ece      Created non-bus version
//08-06-2002       khe@ece      Modified to bus version
//01-06-2003    qjing@ece       fixed the asymmetry in x-direction
// What was changed is the last 5-12 lines of the verilogA code, that is, the
// definition of forces flowing though the connection terminals in the chip
// frame. In those forces, force terms for both topology 1 and 0
// should be included. But in the original code, for forces in
// x-direction, only terms for topology 1 were defined. The parts for
// y-direction was correct. That's why we saw asymmetric x-displacements in
// the simulation with the old model. The model is now symmetric.
//10-21-2003    gcwong@ece      new gap model based on rotated parallel plate
//02-10-2004    gcwong@ece      forked electrostatic and
//                      electrostatic/mechanical models
//02-17-2004    gcwong@ece      changed module name
//                      removed bottom beam functions
//                      removed metal layers from voltage calcs
//03-02-2004gcwong@eceused submodule


`include "../../constants.h"
`include "../../discipline.h"
`include "../../process.h"
`include "../../design.h"


module gap2D_pp_elec(phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b,
xa_t, xb_b, xb_t, ta_b, ta_t, tb_b, tb_t);

    inout phia_b;
    rotational phia_b;
    inout phia_t;
    rotational phia_t;
    inout phib_b;
    rotational phib_b;
    inout phib_t;
    rotational phib_t;
    inout va_b;
    electrical va_b;
```

```
inout va_t;
electrical va_t;
inout vb_b;
electrical vb_b;
inout vb_t;
electrical vb_t;
inout [0:1] xa_b;
kinematic [0:1] xa_b;
inout [0:1] xa_t;
kinematic [0:1] xa_t;
inout [0:1] xb_b;
kinematic [0:1] xb_b;
inout [0:1] xb_t;
kinematic [0:1] xb_t;
inout ta_b;
thermal ta_b;
inout ta_t;
thermal ta_t;
inout tb_b;
thermal tb_b;
inout tb_t;
thermal tb_t;

parameter real bottom_electrode_offset = `default_bottom_electrode_offset;

parameter real finger_l_t = `default_finger_l_t;
parameter real finger_l_b = `default_finger_l_b;

parameter real finger_number = `default_finger_number;
parameter real gap = `default_gap;
parameter real overlap = `default_gap_overlap;

// thicknesses used for calculation of total beam thickness
parameter real t_m3 = `default_t_m3;
parameter real t_m2 = `default_t_m2;
parameter real t_m1 = `default_t_m1;
parameter real t_poly = `default_t_poly;

parameter real   t_m3_m2 = `default_t_m3_m2;
parameter real   t_m3_m1_overpoly = `default_t_m3_m1_overpoly;
parameter real   t_m3_m1_overfield = `default_t_m3_m1_overfield;
parameter real   t_m3_poly = `default_t_m3_poly;
parameter real   t_m3_sub = `default_t_m3_sub;
parameter real   t_m2_m1_overpoly = `default_t_m2_m1_overpoly;
parameter real   t_m2_m1_overfield = `default_t_m2_m1_overfield;
parameter real   t_m2_poly = `default_t_m2_poly;
parameter real   t_m2_sub = `default_t_m2_sub;
parameter real   t_m1_poly = `default_t_m1_poly;
parameter real   t_m1_sub = `default_t_m1_sub;
parameter real   t_poly_sub = `default_t_poly_sub;

parameter integer flag_m3 = 1;
parameter integer flag_m2 = 1;
parameter integer flag_m1 = 1;
```

```
    parameter integer flag_poly = 1;

    // thickness
    parameter real thickness = flag_m3*t_m3 + flag_m2*t_m2
      +flag_m1*t_m1 + flag_poly*t_poly
      +flag_m3*flag_m2*t_m3_m2
      +flag_m3*(1-flag_m2)*flag_m1*flag_poly*t_m3_m1_overpoly
      +flag_m3*(1-flag_m2)*flag_m1*(1-flag_poly)*t_m3_m1_overfield
      +flag_m3*(1-flag_m2)*(1-flag_m1)*flag_poly*t_m3_poly
      +flag_m3*(1-flag_m2)*(1-flag_m1)*(1-flag_poly)*t_m3_sub
      +flag_m2*flag_m1*flag_poly*t_m2_m1_overpoly
      +flag_m2*flag_m1*(1-flag_poly)*t_m2_m1_overfield
      +flag_m2*(1-flag_m1)*flag_poly*t_m2_poly
      +flag_m2*(1-flag_m1)*(1-flag_poly)*t_m2_sub
      +flag_m1*flag_poly*t_m1_poly
      +flag_m1*(1-flag_poly)*t_m1_sub
      +flag_poly*t_poly_sub;

    parameter real angle = `default_gap_angle;

    parameter real visc_air = `default_visc_air;
    //parameter real contact_oxide_t = `default_ntv_ox_t;
    // remaining gap at which the beams come into contact
    parameter real gcontact = `default_gcontact;

    gap_pp_elec # ( .bottom_electrode_offset(bottom_electrode_offset),
.finger_l_t(finger_l_t), .finger_l_b(finger_l_b), .finger_number(finger_number),
.gap(gap), .overlap(overlap), .thickness(thickness), .angle(angle),
.visc_air(visc_air), .gcontact(gcontact) )
      gap_pp_elec_1 (phia_b, phia_t, phib_b, phib_t, va_b, va_t, vb_b, vb_t, xa_b,
xa_t, xb_b, xb_t);

endmodule
```

A.5 Thermal beam model

```
//(c) Copyright 1999-2004 Carnegie Mellon University
//    All Rights Reserved.
//
//VerilogA for beam thermal effects, 2D, veriloga
//NODAS Version: 1.0 03-19-2004
//
//Synopsys:
//
//Revision History:
//Date             Who          What
//03-19-2004       gcwong@ece     Created

`include "../../constants.h"
`include "../../discipline.h"
```

```
// model name and bus pin definition, multiple disciplines

module beam_thermal(phia, phib, ta, tb, va, vb);

    inout phia, phib;
    rotational phia, phib;
    inout ta;
    thermal ta;
    inout tb;
    thermal tc;
    thermal tb;
    inout va, vb;
    electrical va, vb;

    // beam width and length
    parameter real w = 0;
    parameter real l = 0;
    // width of the oxide to the left and to the right of the metal in
    // the metal layer.  If set to zero, the metal width equals the beam width
    parameter real wom3l = 0;
    parameter real wom3r = 0;
    parameter real wom2l = 0;
    parameter real wom2r = 0;
    parameter real wom1l = 0;
    parameter real wom1r = 0;
    parameter real wopl = 0;
    parameter real wopr = 0;

    // metal,poly, oxide young's modulus
    parameter real Emetal = 0;
    parameter real Eoxide = 0;
    parameter real Epoly = 0;

    // which metal layers are used
    parameter integer flag_m3 = 0;
    parameter integer flag_m2 = 0;
    parameter integer flag_m1 = 0;
    parameter integer flag_poly = 0;

    // thickness of metal layers and oxide layers
    parameter real t_m3 = 0;
    parameter real t_m2 = 0;
    parameter real t_m1 = 0;
    parameter real t_poly = 0;

    parameter real t_m3_m2 = 0;
    parameter real t_m3_m1_overpoly = 0;
    parameter real t_m3_m1_overfield = 0;
    parameter real t_m3_poly = 0;
    parameter real t_m3_sub = 0;
    parameter real t_m2_m1_overpoly = 0;
    parameter real t_m2_m1_overfield = 0;
    parameter real t_m2_poly = 0;
    parameter real t_m2_sub = 0;
    parameter real t_m1_poly = 0;
    parameter real t_m1_sub = 0;
    parameter real t_poly_sub = 0;

    parameter real alpha_oxide = 0;
    parameter real alpha_metal = 0;
    parameter real alpha_poly = 0;

    // temperature when beam is not bending
    parameter real T0 = 0;

    // thermal conductivity
    parameter real koxide = 0;
    parameter real kpoly = 0;
    parameter real kmetal = 0;

    // residual stress parameters
    parameter real sigma_res_oxide = 0;
    parameter real sigma_res_poly = 0;
    parameter real sigma_res_metal = 0;

    // used for power calculation
    parameter real sheet_resistance_poly = 0;

    // the location of the neutral axis, temperature
    real w0, T, deltaV;
    // the calculated metal widths
    real m3w, m2w, m1w, pw;
    // variables for the moment calculation
    real Mz, Mz_temp, Mz_oxide_temp, Mz_metal_temp;
    real Mz_res, Mz_oxide_res, Mz_metal_res;
    // effective thermal conductance
    real Geff;
    // thermal resistor vars
    real Ppoly, Rpoly;

    // functions to simplify the calculation of the neutral axis
    // one is the numerator and the other is the denominator
    // for each metal or oxide layer, we add these functions to the
    // numerator or denominator to calculate the neutral axis

    // metal layer
    analog function real neutral_numerator_metal;
      input Emetal, Eoxide, h, w, wl, wr;
      real Emetal, Eoxide, h, w, wl, wr;
      // Emetal is the Young's Modulus for the metal/poly
      // Eoxide is the Young's Modulus for the oxide
      // h is the height of the metal layer
      // w is the width of the metal layer
      // wl is the width of the oxide section in the metal layer
      // wr is the width of the oxide section in the metal layer

      neutral_numerator_metal = h*(Eoxide*(pow(wl,2)-pow(wr,2)+2*w*wr) +
          Emetal*(pow(w,2)-pow(wl,2)+pow(wr,2)-2*w*wr));
```

```
endfunction

analog function real neutral_denominator_metal;
  input Emetal, Eoxide, h, w, wl, wr;
  real Emetal, Eoxide, h, w, wl, wr;
  // Emetal is the Young's Modulus for the metal/poly
  // Eoxide is the Young's Modulus for the oxide
  // h is the height of the metal layer
  // w is the width of the metal layer
  // wl is the width of the oxide section in the metal layer
  // wr is the width of the oxide section in the metal layer

  neutral_denominator_metal = 2*h*(Eoxide*(wl+wr) + Emetal*(w-wl-wr));
endfunction

// oxide layer
analog function real neutral_numerator_oxide;
  input Eoxide, h, w;
  real Eoxide, h, w;
  // Eoxide is the Young's Modulus for the oxide
  // h is the height of the oxide layer
  // w is the width of the oxide layer

  neutral_numerator_oxide = h*Eoxide*pow(w,2);
endfunction

analog function real neutral_denominator_oxide;
  input Eoxide, h, w;
  real Eoxide, h, w;
  // Eoxide is the Young's Modulus for the oxide
  // h is the height of the oxide layer
  // w is the width of the oxide layer

  neutral_denominator_oxide = 2*h*Eoxide*w;
endfunction


// functions to simplify the calculation of the moments in the metal/poly
// layer.  calculates the product of the ti, yi, and wi
// ti: thickness of the layer
// yi: distance from centroid of layer to neutral axis
// wi: width of layer
analog function real metal_t_y_w;
  input t, w0, wm, wl;
  real t, w0, wm, wl;
  // t is the thickness
  // w0 is the location of the neutral axis from the left edge of the
  //     cross section
  // wm is the width of the metal or poly in the layer
  // wl is the width of the oxide to the left of the metal in the layer

  metal_t_y_w = t*(wm/2.0+wl-w0)*wm;
endfunction
```

```
analog function real oxide_t_y_w;
  input t, w0, wl, wr;
  real t, w0, wl, wr;
  // t is the thickness
  // w0 is the location of the neutral axis from the left edge of the
  //     cross section
  // wl is the width of the oxide to the left of the metal in the layer
  // wr is the width of the oxide to the right of the metal in the layer

  oxide_t_y_w = t*((wl/2.0-w0)*wl + (w-wr/2.0-w0)*wr);
endfunction

analog function real M_metal_temp;
  input w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl, wopr,
      m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly,
      alpha_oxide, Eoxide, alpha_metal, Emetal, alpha_poly, Epoly,
      flag_m3, flag_m2, flag_m1, flag_poly;
  real w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl, wopr,
      m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly,
      alpha_oxide, Eoxide, alpha_metal, Emetal, alpha_poly, Epoly,
      flag_m3, flag_m2, flag_m1, flag_poly;
  // t_* is the thickness of a layer
  // w0 is the location of the neutral axis from the left edge of the
  //     cross section
  // w*l is the width of the oxide to the left of the metal in the layer
  // w*r is the width of the oxide to the right of the metal in the layer
  // alpha_* is the coefficient of expansion
  // E* is the young's modulus
  // flag_* indicates if the layer is present

  M_metal_temp = alpha_oxide*Eoxide*(
      flag_m3*oxide_t_y_w(t_m3,w0,wom3l,wom3r) +
      flag_m2*oxide_t_y_w(t_m2,w0,wom2l,wom2r) +
      flag_m1*oxide_t_y_w(t_m1,w0,wom1l,wom1r) +
      flag_poly*oxide_t_y_w(t_poly,w0,wopl,wopr)
    ) +
    alpha_metal*Emetal*(
      flag_m3*metal_t_y_w(t_m3,w0,m3w,wom3l) +
      flag_m2*metal_t_y_w(t_m2,w0,m2w,wom2l) +
      flag_m1*metal_t_y_w(t_m1,w0,m1w,wom1l)
    ) +
    alpha_poly*Epoly*flag_poly*metal_t_y_w(t_poly,w0,pw,wopl);

endfunction

analog function real M_metal_res;
  input w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl, wopr,
      m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly,
      sigma_oxide, sigma_metal, sigma_poly,
      flag_m3, flag_m2, flag_m1, flag_poly;
  real w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl, wopr,
      m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly,
      sigma_oxide, sigma_metal, sigma_poly,
      flag_m3, flag_m2, flag_m1, flag_poly;
  // t_* is the thickness of a layer
  // w0 is the location of the neutral axis from the left edge of the
  //     cross section
  // w*l is the width of the oxide to the left of the metal in the layer
  // w*r is the width of the oxide to the right of the metal in the layer
  // alpha_* is the coefficient of expansion
  // E* is the young's modulus
  // flag_* indicates if the layer is present

  M_metal_res = sigma_oxide*(
      flag_m3*oxide_t_y_w(t_m3,w0,wom3l,wom3r) +
      flag_m2*oxide_t_y_w(t_m2,w0,wom2l,wom2r) +
      flag_m1*oxide_t_y_w(t_m1,w0,wom1l,wom1r) +
      flag_poly*oxide_t_y_w(t_poly,w0,wopl,wopr)
    ) +
    sigma_metal*(
      flag_m3*metal_t_y_w(t_m3,w0,m3w,wom3l) +
      flag_m2*metal_t_y_w(t_m2,w0,m2w,wom2l) +
      flag_m1*metal_t_y_w(t_m1,w0,m1w,wom1l)
    ) +
    sigma_poly*flag_poly*metal_t_y_w(t_poly,w0,pw,wopl);

endfunction

analog begin

  @ (initial_step) begin

      // calculate the neutral axis for each metal layer combination
      // in separate file to reduce clutter
      // also calculates the moment terms except for the temperature
      // factor
      `include "neutral_axis.va"

      // calculate poly resistance if it exists
      if (flag_poly==1) begin
        Rpoly = sheet_resistance_poly * l/pw;
      end
      else begin
        Rpoly = 0;
      end
  end

  // for now, just average the temperature across the beam
  T = (Temp(ta) + Temp(tb))/2.0;

  // the total moment acting on the beam is the sum of the
  // moment due to the thermal expansion and the residual stress
  Mz = Mz_temp*(T-T0) + Mz_res;

  // calculate the electrical power dissipated in the poly resistor
  if (flag_poly==1) begin
      Ppoly = pow(deltaV,2)/Rpoly;
  end
```

```
      else begin
          Ppoly = 0;
      end

      // assign the thermal power through the beam
      Pwr(tc,ta) <+ 2*Geff*(Temp(tc) - Temp(ta));
      Pwr(tc,tb) <+ 2*Geff*(Temp(tc) - Temp(tb));

      // assign the thermal power due to the poly resistor
      Pwr(tc) <+ -Ppoly;

      // assign the thermal moment on the beam
      Tau(phia) <+ Mz;
      Tau(phib) <+ -Mz;

    end // end of analog block
endmodule// end of module
```

A.6 Neutral axis include file

neutral_axis.va is included in the beam_thermal Verilog-A file. This file calculates the location of the neutral axis with respect to the left edge of the beam's cross section. It also calculates the moment generated by the residual stress as well as the effective thermal conductance.

```
if (flag_m3==1 && flag_m2==1 && flag_m1==1 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m2,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_m1_overpoly,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m2,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_m1_overpoly,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = w-wom2l-wom2r;
    m1w = w-wom1l-wom1r;
    pw = w-wopl-wopr;
    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m3_m2+t_m2_m1_overpoly+t_m1_poly+t_poly_sub))/l;
end;

if (flag_m3==1 && flag_m2==1 && flag_m1==1 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m2,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_m1_overfield,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m2,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_m1_overfield,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = w-wom2l-wom2r;
    m1w = w-wom1l-wom1r;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      koxide*w*(t_m3_m2+t_m2_m1_overfield+t_m1_sub))/l;
end;

if (flag_m3==1 && flag_m2==1 && flag_m1==0 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m2,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m2,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = w-wom2l-wom2r;
    m1w = 0;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(
      t_m3_m2+t_m2_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m3_m2+t_m2_poly+t_poly_sub))/l;
end;

if (flag_m3==1 && flag_m2==1 && flag_m1==0 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m2,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m2,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = w-wom2l-wom2r;
    m1w = 0;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m3_m2+t_m2_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m3_m2+t_m2_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      koxide*w*(t_m3_m2+t_m2_sub))/l;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==1 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m1_overpoly,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m1_overpoly,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = 0;
    m1w = w-wom1l-wom1r;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(
      t_m3_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(
      t_m3_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);
```

```
    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m3_m1_overpoly+t_m1_poly+t_poly_sub))/l;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==1 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_m1_overfield,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_m1_overfield,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = 0;
    m1w = w-wom1l-wom1r;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m3_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m3_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      koxide*w*(t_m3_m1_overfield+t_m1_sub))/l;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==0 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = 0;
    m1w = 0;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m3_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m3_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m3_poly+t_poly_sub))/l;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==0 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_numerator_oxide(Eoxide,t_m3_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m3,w,wom3l,wom3r) +
       neutral_denominator_oxide(Eoxide,t_m3_sub,w));

    // metal width calculation
    m3w = w-wom3l-wom3r;
    m2w = 0;
    m1w = 0;
    pw = 0;
    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m3_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m3_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m3*(koxide*(wom3l+wom3r) + kmetal*m3w) +
      koxide*w*(t_m3_sub))/l;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==1 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_m1_overpoly,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_m1_overpoly,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = w-wom2l-wom2r;
    m1w = w-wom1l-wom1r;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(
      t_m2_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);
```

```
    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(
      t_m2_m1_overpoly+t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m2_m1_overpoly+t_m1_poly+t_poly_sub))/l;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==1 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_m1_overfield,w) +
       neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_m1_overfield,w) +
       neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = w-wom2l-wom2r;
    m1w = w-wom1l-wom1r;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m2_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m2_m1_overfield+t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      koxide*w*(t_m2_m1_overfield+t_m1_sub))/l;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==0 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = w-wom2l-wom2r;
    m1w = 0;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m2_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m2_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m2_poly+t_poly_sub))/l;
end;
```

```
if (flag_m3==0 && flag_m2==1 && flag_m1==0 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_numerator_oxide(Eoxide,t_m2_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m2,w,wom2l,wom2r) +
       neutral_denominator_oxide(Eoxide,t_m2_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = w-wom2l-wom2r;
    m1w = 0;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m2_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m2_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
    Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m2*(koxide*(wom2l+wom2r) + kmetal*m2w) +
      koxide*w*(t_m2_sub))/l;
end;

if (flag_m3==0 && flag_m2==0 && flag_m1==1 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_poly,w) +
       neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_poly,w) +
       neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = 0;
    m1w = w-wom1l-wom1r;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m1_poly+t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_m1_poly+t_poly_sub))/l;
end;

if (flag_m3==0 && flag_m2==0 && flag_m1==1 && flag_poly==0) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_numerator_oxide(Eoxide,t_m1_sub,w)) /
       (neutral_denominator_metal(Emetal,Eoxide,t_m1,w,wom1l,wom1r) +
       neutral_denominator_oxide(Eoxide,t_m1_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = 0;
    m1w = w-wom1l-wom1r;
    pw = 0;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_m1_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_m1*(koxide*(wom1l+wom1r) + kmetal*m1w) +
      koxide*w*(t_m1_sub))/l;
end;

if (flag_m3==0 && flag_m2==0 && flag_m1==0 && flag_poly==1) begin
    // neutral axis calculation
    w0 = (neutral_numerator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_numerator_oxide(Eoxide,t_poly_sub,w)) /
       (neutral_denominator_metal(Epoly,Eoxide,t_poly,w,wopl,wopr) +
       neutral_denominator_oxide(Eoxide,t_poly_sub,w));

    // metal width calculation
    m3w = 0;
    m2w = 0;
    m1w = 0;
    pw = w-wopl-wopr;

    // thermal expansion calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_temp = alpha_oxide*Eoxide*w*(w/2.0-w0)*(t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_temp = M_metal_temp(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, alpha_oxide, Eoxide,
alpha_metal, Emetal, alpha_poly, Epoly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_temp = Mz_oxide_temp + Mz_metal_temp;

    // residual stress calculations
    // partial calculation of the moment resulting from the oxide layers only
    Mz_oxide_res = sigma_res_oxide*w*(w/2.0-w0)*(t_poly_sub);
    // partial calculation of the moment resulting from the metal/oxide layer
   Mz_metal_res = M_metal_res(w0, wom3l, wom3r, wom2l, wom2r, wom1l, wom1r, wopl,
wopr, m3w, m2w, m1w, pw, t_m3, t_m2, t_m1, t_poly, sigma_res_oxide,
sigma_res_metal, sigma_res_poly, flag_m3, flag_m2, flag_m1, flag_poly);

    Mz_res = Mz_oxide_res + Mz_metal_res;

    // calculate effective thermal conductance
    Geff = (t_poly*(koxide*(wopl+wopr) + kpoly*pw) +
      koxide*w*(t_poly_sub))/l;
end;

// this case doesn't happen
if (flag_m3==0 && flag_m2==0 && flag_m1==0 && flag_poly==0) begin
    $display("ERROR: You selected no metal layers. Set the flag_* parameters.");
    $finish;
```

```
end;
```

Appendix B:Verilog-A Modeling

It is useful to discuss some of the issues and features in Verilog-A which may be encountered when creating models.

B.1 Division

Verilog-A supports many mathematical functions, including division. Dividing two numbers or variables is simple (Listing B-1):

However, the simulator will interpret the statement as integer division and report that *foo* equals 0 and *foobar* equals 1. To correct this problem, the integers must be written as 2.0, 3.0 and 4.0 (Listing B-2):

This code will result in *foo* = $0.\overline{6}$ and *foobar* = $1.\overline{3}$. Calculated values saved in variables will take the format of the variable declaration. For instance, if *bar* was declared as an integer, *bar* will be 4 instead of 4.0. Care must be taken when declaring variables and when using those variables as divisors to ensure that the simulator will not interpret the division as integer division.

```
foo = 2 / 3;
bar = 4;
foobar = bar / 3;
```

Listing B-1. Division Example.

```
foo = 2.0 / 3.0;
bar = 4.0;
foobar = bar / 3.0;
```

Listing B-2. Correct way to implement division.

```
if foo > 1 begin
     bar = 5;
end
else if foo == 1 begin
     bar = 1;
end
```

Listing B-3. Incomplete if statement.

```
if foo > 1 begin
     bar = 5;
end
else if foo == 1 begin
     bar = 1;
end
else
     bar = 0;
end
```

Listing B-4. Full case if statement.

B.2 Hidden States

Hidden states can be generated from incomplete *if* and *case* statements. For instance, the following if/else if block is incomplete (Listing B-3):

Assuming that *bar* was not previously set, *bar* will only be set when *foo* is greater than or equal to 1. The simulator does not know the value of *bar* when *foo* is less than 1. If a periodic steady state (PSS) simulation was executed on this code, the simulator would generate a hidden state error message. Two options exist to remedy this problem. First, an *else* statement block can be added to account for all remaining cases (Listing B-4) or *bar* can be initialized before the if/else if statement blocks (Listing B-5):

It is a matter of style as to which method is chosen, but the full case *if* statement in Listing B-6 is preferred for code readability. Similarly, when using the Verilog-A *case* statement, all cases should be accounted or the *default* statement should be set (Listing B-6).

```
bar = 0;
if foo > 1 begin
     bar = 5;
end
else if foo == 1 begin
     bar = 1;
end
```

Listing B-5. if statement with bar initialized to 0.

```
case 1
     (foo > 1) : bar = 5;
     (foo == 1) : bar = 1;
     (foo < 1) : bar = 0;
endcase

case 1
     (foo > 1) : bar = 5;
     (foo == 1) : bar = 1;
     default bar = 0;
endcase
```

Listing B-6. Full case statement examples.

B.3 Multiple discipline.h references

The discipline.h file contains definitions of the natures, disciplines and tolerances. In Cadence Spectre, a discipline.h file comes with the simulator. NODAS uses a customized version of the discipline.h file that all of the NODAS element models reference. A problem arises when analog hardware description language (AHDL) based libraries, such as transistor libraries, from third party vendors reference the default discipline.h file. Cadence Spectre will include the transistor libraries first and read the default discipline.h file. When it encounters a NODAS or any other customized discipline.h file, it will compare it with the discipline.h file(s) previously read into the simulator. If differences exist, Cadence Spectre will report an error stating that previous definitions of the natures have been defined differently. If the transistor libraries are not being used, the temporary solution is to remove the transistor library from the model library list. The permanent solution is to modify the Cadence search path so that it will find and load the customized discipline.h file first instead of the default discipline.h file.

B.4 Tolerances

The tolerances defined in the default discipline.h file included with Cadence are not suitable for MEMS simulations. The suggested tolerances can be found in Table 2-1. The tolerances were selected so that they are at least 10^{-3} times smaller than the typical value.

Table 2-1. Suggested Tolerance Values.

Nature	Value
Position	1e-12 m
Velocity	1e-9 m/s
Acceleration	1e-6 m/s^2
Force	1e-12 N
Angle	1e-6 rads
Angular Velocity	1e-6 rads/sec
Angular Acceleration	1e-6 rads/sec^2
Angular Force	1e-12 N-m

Figure B-1a shows an example of a NODAS simulation where the angular tolerance was set incorrectly. The test simulation was a cantilever beam with the free end displacement constrained. The angle pin was driven by a sinusoidal angular displacement source at t=0. The angular tolerance in Figure B-1a the was set to 1 radian whereas the angular tolerance in Figure B-1b was set to 1e-6 radians. As can be seen, Figure B-1a. does not display the ringing effect that Figure B-1b has. Since the tolerance was set to 1 radian and the angular displacement was on the order of 2 μradian, information about the ringing was lost

The tolerances must be set to values smaller than the typical expected values. Tolerance values of 1e3 to 1e6 times smaller than the typical value are suggested.

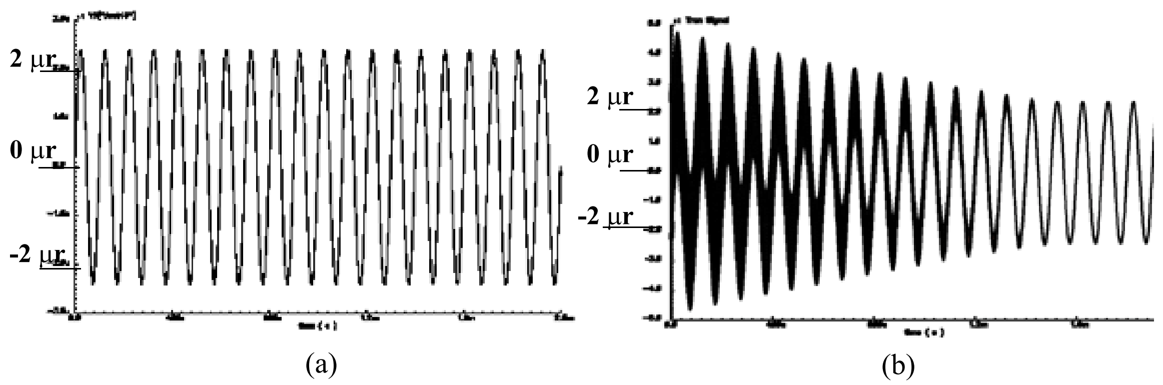


Figure B-1. Tolerance examples.
(a) Shows an angular tolerance of 1 radian. (b) Shows an angular tolerance of 1e-6 radians.

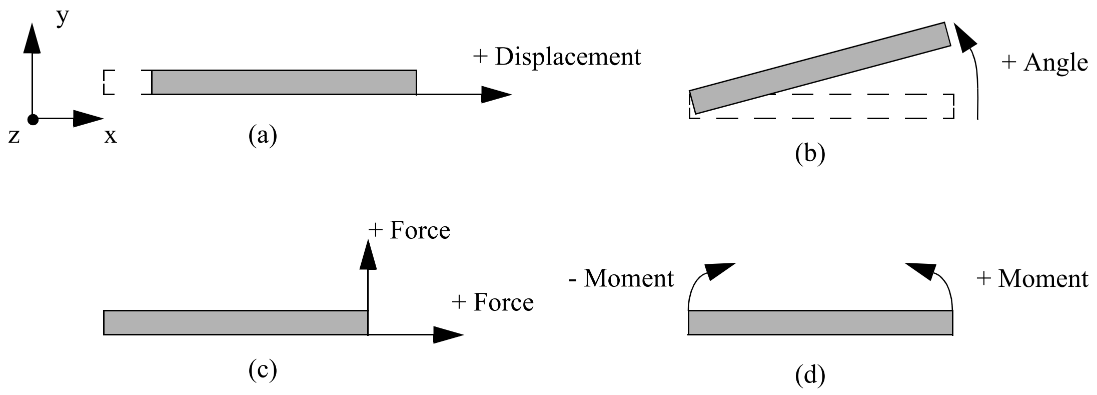


Figure B-2. Nodal conventions.
(a) positive displacement, (b) positive angles, (c) positive forces, (d) positive and negative moments.

B.5 Nodal Conventions

In order to ensure inter operability between the NODAS models, a nodal convention must be chosen. Table 2-2 lists the conventions used in NODAS.

Table 2-2. Nodal Conventions

Type	Meaning
Positive displacement	Positive axial displacement
Positive angles	Counterclockwise rotation around axis
Positive forces	Force acts in positive axial direction
Positive moments	Moment acts counterclockwise around axis

Figure B-2 shows the conventions graphically.

```
analog function real f11;
    input L,x;
    real L,x;
    f11 = 1.0 - 3.0*pow((x/L),2) + 2.0*pow((x/L),3);
endfunction
```

Listing B-7. Example of Verilog-A analog function definition.

B.6 Through and Across Variables

When creating models, the choice of through and across variables must be chosen. Through and across are also called the flow and potential between two nodes. Electrical engineers are familiar with the through variable current and the across variable voltage Figure B-3a. Other through and across variables are shown in Figure B-3b,c,d. The definition of these variables is set in the discipline.h file.

B.7 Analog Functions

Functions can be defined in Verilog-A to simplify the code as well as allowing for the reuse of user defined functions within a Verilog-A module [9]. The electrostatic gap model uses analog functions to define the beam shape functions, its derivatives and integrals. Listing B-7 is an example of a function defined in the electrostatic gap model.

The function *f11* returns a real value number which is set by the function variable *f11* in the function block. It takes two input arguments, *L* and *x* in that order. Listing B-8 shows a call to the

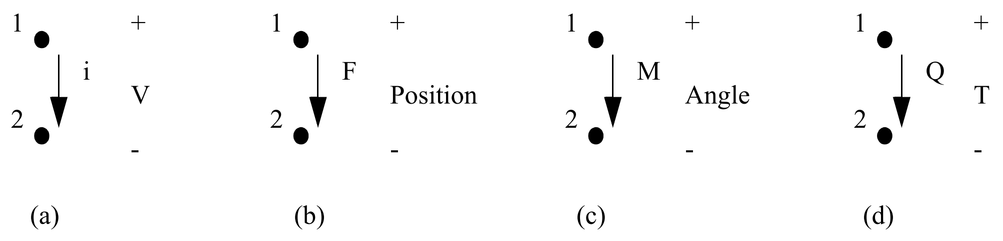


Figure B-3. Through and across variables.
(a) electrical: through = current, across = voltage, (b) kinematic: through = force, across = position, (c) rotational: through = moment, across = angle, (d) thermal: through = power (heat), across = temperature.

```
y1 = f11 (length, displacement);
y2 = f11 (1, 5);
```

Listing B-8. Example of Verilog-A analog function call.

function. The function can be called with variables (i.e. *length*, *displacement*) or by numbers (i.e 1, 5).

B.8 Code Performance

Two optimizations to the Verilog-A code can be used to decrease simulation time. The simplest optimization is to remove redundant calculations. Listing B-9 shows an example of removing redundant calculations.

Not all calculations need to be executed on every iteration. Variables that have values that are true for all time can be placed in an initial step block. The initial step block is executed in the first iteration. Listing B-10 is an excerpt of the initial step block from the electrostatic gap model.

Variables such as the spring constant, the mass, and rotation matrix calculations have been put into initial blocks in the NODAS library. For any given simulation, the values of these variables are constant. Similarly, in the electrostatic gap model, the calculation of the layout configuration of the electrodes is constant and therefore is put into an initial block.

```
foo = 2.5*cos(theta0)/(l*thickness*w*density);
bar = l*w*thickness*density*50/200.0;
x = w*thickness*l*density/50.0;
```

(a)

```
mass = l*w*thickness*density;
foo = 2.5*cos(theta0)/mass;
bar = mass*50/200.0;
x = mass/50.0;
```

(b)

Listing B-9. Redundant calculations:
(a) exist. (b) removed.

```
 analog begin

     @ (initial_step) begin

          offsetL2Left = bottom_electrode_offset;
          magoffsetL2Right=finger_l_t+finger_l_b-overlap-
(abs(offsetL2Left)+overlap);

     end

     ...
end
```

Listing B-10. Initial step block.

B.9 NODAS Code Architecture

As discussed in Section 1.3, the current NODAS architecture uses a library of behavioral model submodules called *NODAS_submodules*. The submodules are written as three conductor models. The submodules contain all of the physics of the element and facilitate easy code maintenance. Aside from the process parameters, the main differences between a one conductor and a three conductor model is that the thickness, the effective Young's modulus, and effective density are calculated differently. For a three conductor model, it is possible to have up to three metal layers, one poly layer, and the oxide layers between the conducting layers. The thickness, effective Young's modulus, and effective density are calculated using the geometric and material properties for these layers. Similarly, for a one conductor model, a poly layer and an oxide layer are the only possibilities, therefore the geometrical and material properties for these layers are used to calculate the thickness, effective Young's modulus, and effective density. To instantiate a submodule for a one conductor or three conductor module, the thickness, effective Young's modulus, and effective density (as well as the other parameters for the element) are calculated and are passed into the submodule using the syntax shown in Listing 2-1.

Each model in the *NODAS02_1C_2D* (one conductor, 2D), *NODAS02_1C_3D* (one conductor, 3D), *NODAS02_3C_2D* (three conductor, 2D), *NODAS02_3C_3D* (three conductor, 3D) libraries contains several sections in the Verilog-A code. The first part contains the module definition with the pin order. Next, each signal pin is assigned the proper nature (electrical, kinematic, rotational, etc.) and signal flow direction (input, output, inout). A list of user parameters follows. The thickness, effective Young's Modulus, and effective density is calculated in this parameter list. Variables and analog functions are defined in next and an analog block follows if any non-constant values need to be calculated. Finally, the submodules are called. Currently, all of the NODAS models follow this coding convention.

Appendix C:ANSYS Scripts

C.1 Fixed-Fixed Beam

From ANSYS 7.0 tutorial [2].

```
/batch,list
/title, Gilbert,static,hysteresis,weighted,sparse
/com, --------------------------------------------------------
/com,         2-D Beam under electrostatic load
/com,         --------------------------------
/com,   Compare with 3-D model from the paper:
/com,   J.R.Gilbert, G.K.Ananthasuresh, S.D.Senturia, (MIT)
/com,   "3-D Modelling of Contact Problems and Hysteresis in
/com,    Coupled Electro-Mechanics", MEMS'96, pp. 127-132.
/com,
/com,                      3-D Model:
/com,   Beam is clamped at either end, suspended 0.7 um over
/com,   a ground plane with contact stop at 0.1 um above the
/com,   ground plane.   Beam dimensions and material properties:
/com,   length bl=80um, width wb=10um, height bh=.5um, E=169GPa, mu=0.25
/com,   Initial Gap: gap=0.7um , finishing gap gfi=0.1um
/com,   Maximum displacement is 0.6um (gap-gfi)
/com,
/com, Value of the pull-in voltage: 18V
/com, Both pull-in and release behaviors are modeled (hysteresis loop).

!------------- Control parameters ---------------------

vltg1 = 11.0   ! Bias voltage 1
!vltg2 = 14.5   ! Bias voltage 2
!vltg2 = 16.5
vltg2 = 14.5
vltg  = 18.0      ! Pull-in voltage

esize=0.5        ! element mesh size

!---------------- Geometry parameters ---------------

bl=40    ! beam length
bh=.5    ! beam height
gap=.7    ! maximum gap
gap0=.6    ! air gap
eps0=8.854e-6
```

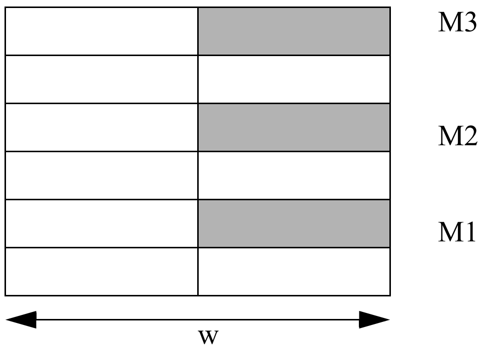
```
!---------------------- Model ---------------

/prep7

emunit,epzro,eps0

et,1,42,,,0
et,2,109,1,         ! weighted transducer
et,3,12,,,,1

mp,ex,1,169e3
mp,nuxy,1,0.25
mp,perx,2,1
mp,mu,3,0

r,1,c0,eps0
r,2,,1690

rect,,bl,gap,gap+bh
rect,,bl,,gap+bh
aovlap,all
nummrg,kp

ASEL,S,loc,y,gap+bh/2
AATT,1,,1
ASEL,INVERT
aatt,2,1,2
cm,area1,area

alls
esize,esize

asel,s,mat,,1
mshape,0,2
mshkey,2
amesh,all

asel,s,mat,,2
mshape,1,2
mshkey,1
amesh,all

type,3    ! gap element mesh
mat,3
real,2
*get,nomax,node,0,num,max
kn=bl/esize
k8=nomax+1
xl=0
*do,i8,1,kn+1
```

```
 n52=k8
 n53=n52+1
 n,n52,xl,gap
 n,n53,xl,gap-gap0
 e,n53,n52
 k8=k8+2
 xl=xl+esize
*enddo
nummrg,node
alls


!--------------- Boundary Conditions -----------------

esel,s,type,,2
nsle,s
nsel,r,loc,y,gap
cm,bnode,node

nsle,s
nsel,r,loc,y,0
d,all,volt,0   ! ground

alls
nsel,s,loc,x,0       !  fix left end
nsel,a,loc,y,0       !  fix bottom
d,all,ux,0
d,all,uy,0

alls
nsel,s,loc,x,bl      !  symmetry line
d,all,ux,0

esel,s,type,,3
nsle,s
nsel,r,loc,y,gap-gap0    ! fix gap elements
d,all,all

allsel,all
fini
save


/solu

!----------------- Loading (below pull-in) --------------------

/solu

eqslv,sparse
```

```
cnvtol,f,1,1.0e-4
AUTOTS,ON
NSUBST,1
outres,all,all
neqit,100
nlgeom,on

cmsel,s,bnode   ! Bias 1
d,all,volt,vltg1
alls

solve

cmsel,s,bnode   ! Bias 2
d,all,volt,vltg2
alls

solve
fini
```

C.2 Beam Lateral Bending Moment

This tutorial will demonstrate how to create a model of a multimorph beam in ANSYS for thermal simulation. It is a M1, M2, M3 stack where the metal layers are offset to the right. .



```
l = 100 µm
w = 10 µm

1.         Add the SOLID45 elements
           Preprocessor - Element Type - Add/Edit/Delete
           a.   Click the Add… button
           b.   Select Structural Mass - Solid - Brick 8node  45
           c.   Click OK
           d.   Click Close
```

2. Define the material properties to be used in this simulation
Preprocessor - Material Props - Material Models

a. Click on Material Model Number 1
b. Select Structural - Linear - Elastic - Isotropic
    i. EX (Young's Modulus) = 62e9
    ii. PRXY (Poisson's Ratio) = 0.3
    iii. Click OK
c. Select Structural - Thermal Expansion - Secant Coefficient - Isotropic
    i. ALPX = 8.1e-6
    ii. T = 294
    iii. Click OK
d. Copy the material property
    Select Edit - Copy from the Define Material Model Behavior window
    i. Set "from Material number" to 1
    ii. Set "to Material number" to 2
    iii. Click OK
e. Double Click on Material Model Number 2
f. Select Thermal Expansion (secant-iso)
    i. ALPX = 23 e-6
    ii. T = 294
    iii. Click OK
g. Exit the Define Material Model Behavior window by selecting the menu item Material - Exit.

3. Create the geometry
Preprocessor - Modeling - Create - Volume - Block - By Dimensions

a. Create each beam segment
    i. X1, X2 = 0, 5e-6
    ii. Y1, Y2 = 0, 100e-6
    iii. Z1, Z2 = 0, 1.3e-6
    iv. Click Apply
    v. X1, X2 = 0, 5e-6
    vi. Y1, Y2 = 0, 100e-6
    vii. Z1, Z2 = 1.3e-6, 1.665e-6
    viii. Click Apply
    ix. X1, X2 = 0, 5e-6
    x. Y1, Y2 = 0, 100e-6
    xi. Z1, Z2 = 1.665e-6, 2.615e-6
    xii. Click Apply
    xiii. X1, X2 = 0, 5e-6
    xiv. Y1, Y2 = 0, 100e-6
    xv. Z1, Z2 = 2.615e-6, 3.25e-6
    xvi. Click Apply
    xvii. X1, X2 = 0, 5e-6
    xviii. Y1, Y2 = 0, 100e-6
    xix. Z1, Z2 = 3.25e-6, 4.2e-6
    xx. Click Apply
    xxi. X1, X2 = 0, 5e-6

xxii. Y1, Y2 = 0, 100e-6
xxiii. Z1, Z2 = 4.2e-6, 4.835e-6
xxiv. Click OK

4. Copy the shapes and offset it by 5 mm
Preprocessor - Modeling - Copy - Volumes
a. Click the Pick All button
b. Set DX to 5e-6

5. Glue the volumes together
Preprocessor - Modeling - Operate - Booleans - Glue - Volumes
a. Click the Pick All button

6. Rotate the model to show the cross section view

7. Menu Item - PlotCtrls - Pan Zoom Rotate
a. Click Bot

8. Set the properties for each volume element
Preprocessor - Meshing - Mesh Attributes - Picked Volumes
a. Pick the oxide layers, click Apply
i. Select Material 1
ii. Click Apply
b. Pick the metal layers, click OK
i. Select Material 2
ii. Click OK

9. Set the mesh divisions of the cross section view
Preprocessor - Meshing - Size Cntrls - Lines - Picked Lines
a. Click the Box option
b. Pick all of the horizontal lines making sure to not pick any of the vertical lines
i. Do this by drawing a box with the mouse that overlaps the lines you want to pick
ii. Click OK
c. Set NDIV to 10
d. Click OK

10. Using the Pan-Zoom-Rotate window, click the Front button

11. Set the mesh divisions of the top view
Preprocessor - Meshing - Size Cntrls - Lines - Picked Lines
a. Click the Box option
b. Pick all of the vertical lines making sure to not pick any of the horizontal lines
i. Do this by drawing a box with the mouse that overlaps the lines you want to pick
ii. Click OK
c. Set NDIV to 30
d. Click OK

12. Mesh the model
Preprocessor - Meshing - Mesh - Volumes - Mapped - 4 to 6 sided
a. Click Pick All

13. Change the view type to Area and turn numbering on
a. Plot - Areas
b. PlotCtrls - Numbering
i. Click the box next to AREA Area Numbers

14. Apply displacement constraint

Preprocessor - Loads - Define Loads - Apply - Structural - Displacement - On Areas

a. Type in the area numbers into the box in the "Apply U,Rot on Areas"

i. For instance "93,115,89,111,85,107,81,103,73,99,3,79"

ii. Click OK

iii. Select All DOF and set VALUE to 0

iv. Click OK

15. Apply the temperature settings

Preprocessor - Loads - Define Loads - Settings - Uniform Temp

a. Set TUNIF to 350

b. Click OK

Preprocessor - Loads - Define Loads - Settings - Reference Temp

c. Set TREF to 294

d. Click OK

16. Save the model

a. File - Save as… and save the file as beam_thermal.db

17. Solve the problem

a. Setup the Analysis Options

Solution - Analysis Type - Sol'n Controls

i. Select Large displacement static for Analysis Options

ii. Click OK

b. Solve the problem

Solution - Solve - Current LS

i. If the solution does not converge, refine the mesh further and resolve the problem.

18. Plot the results

a. Select the last dataset

General Postproc - Read Results - Last Set

b. Plot the deformed Shape

i. General Postproc - Plot Results - Deformed Shape

ii. Select Def + undeformed

iii. Click OK

c. Create a 2D line plot of the displacement using paths

i. Define the path

General Postproc - Path Operations - Define Path - By Location

1. Set Name to dispx (or any other name)
2. You can also modify the nDiv to look at more internal points. The default is 20.
3. Click OK
4. Set NPT to 1 (point 1)
5. Set X,Y,Z to 10e-6, 0, 0
6. Click OK
7. Set NPT to 2 (point 2)
8. Set X,Y,Z to 10e-6, 100e-6, 0
9. Click OK

10. Click Cancel

ii. Map the path to a degree of freedom
General Postproc - Path Operations - Map onto Path
1. Set Lab to dispx (or any name)
2. Select DOF Solution, Translation UX
3. Click OK

iii. Plot the Path Item
General Postproc - Path Operations - Plot Path Item - On Graph
1. Select DISPY
2. Click OK
3. You can also get a text output by selecting List Path Items instead of On Graph
4. The tip deflection at (x,y,z) = (10e-6, 100e-6, 0) is -249nm

C.3 Electrothermal Conduction

This tutorial will demonstrate how to create a model the thermal conductance and electrothermal heating of a poly beam.

l = 100 µm
w = 5 µm
t = 0.28 µm

1. Add the SOLID5 elements
Preprocessor - Element Type - Add/Edit/Delete
a. Click the Add… button
b. Select Coupled Field - Scalar Brick 4
c. Click OK
d. Click the Options… button
e. Select Temp Volt Mag
f. Click OK
g. Click Close
2. Define the material properties to be used in this simulation
Preprocessor - Material Props - Material Models
a. Click on Material Model Number 1
b. Select Thermal - Conductivity - Isotropic
i. KXX (Themal Conductivity) = 30
ii. Click OK
c. Select Electromagnetics - Resistivity - Constant
i. RSVX = 0.28e-4
ii. Click OK
d. Exit the Define Material Model Behavior window by selecting the menu item Material - Exit.
3. Create the geometry
Preprocessor - Modeling - Create - Volume - Block - By Dimensions

```
      a.   Create each beam segment
           i.   X1, X2 = 0, 5e-6
           ii.  Y1, Y2 = 0, 100e-6
           iii. Z1, Z2 = 0, 0.28e-6
           iv.  Click OK
4.    Set the properties for each volume element
      Preprocessor - Meshing - Mesh Attributes - All Volumes
      a.   Pick the oxide layers, click Apply
           i.   Select Material 1
           ii.  Click OK
5.    Rotate the model to show the cross section view
      Menu Item - PlotCtrls - Pan Zoom Rotate
      a.   Click Bot
6.    Set the mesh divisions of the cross section view
Preprocessor - Meshing - Size Cntrls - Lines - Picked Lines
      a.   Click the Box option
      b.   Pick all of the horizontal lines making sure to not pick any
of the vertical lines
           i.   Do this by drawing a box with the mouse that overlaps
the lines you want to pick
           ii.  Click OK
      c.   Set NDIV to 10
      d.   Click OK
7.    Using the Pan-Zoom-Rotate window, click the Front button
8.    Set the mesh divisions of the top view
      Preprocessor - Meshing - Size Cntrls - Lines - Picked Lines
      a.   Click the Box option
      b.   Pick all of the vertical lines making sure to not pick any of
the horizontal lines
           i.   Do this by drawing a box with the mouse that overlaps
the lines you want to pick
           ii.  Click OK
      c.   Set NDIV to 30
      d.   Click OK
9.    Mesh the model
      Preprocessor - Meshing - Mesh - Volumes - Mapped - 4 to 6 sided
      a.   Click Pick All
10.   Change the view type to Area and turn numbering on
      a.   Plot - Areas
      b.   PlotCtrls - Numbering
           i.   Click the box next to AREA Area Numbers
11.   Apply temperature constraint to bottom and top areas (3 and 4 re-
spectively)
      Preprocessor - Loads - Define Loads - Apply - Thermal - Temperature
- On Areas
      a.   Type in the area numbers into the box in the "Apply TEMP on
Areas"
           i.   Area "3"
           ii.  Click OK
           iii. Select TEMP and set VALUE to 300
```

iv. Click Apply
v. Select VOLT and set VALUE to 5
vi. Area "4"
vii. Click OK
viii. Select TEMP and set VALUE to 500
ix. Click Apply
x. Select VOLT and set VALUE to 0

12. Save the model
    a. File - Save as… and save the file as beam_elec_thermal.db
13. Solve the problem
    a. Solve the problem
Solution - Solve - Current LS
        i. If the solution does not converge, refine the mesh further and resolve the problem.
14. Plot the results
    a. Select the last dataset
    General Postproc - Read Results - Last Set
    b. Create a 2D line plot of the displacement using paths
        i. Define the path (center of the poly beam)
        General Postproc - Path Operations - Define Path - By Location
            1. Set Name to ty (or any other name)
            2. You can also modify the nDiv to look at more internal points. The default is 20.
            3. Click OK
            4. Set NPT to 1 (point 1)
            5. Set X,Y,Z to 2.5e-6, 0, 0.14e-6
            6. Click OK
            7. Set NPT to 2 (point 2)
            8. Set X,Y,Z to 2.5e-6, 100e-6, 0.14e-6
            9. Click OK
            10. Click Cancel
        ii. Map the path to a degree of freedom
        General Postproc - Path Operations - Map onto Path
            1. Set Lab to ty (or any name)
            2. Select DOF Solution, Temperature TEMP
            3. Click OK
        iii. Plot the Path Item
        General Postproc - Path Operations - Plot Path Item - On Graph
            1. Select ty
            2. Click OK
            3. You can also get a text output by selecting List Path Items instead of On Graph
            4. The temperature at (x,y,z) = (2.5e-6, 50e-6, 0.14e-6) is 4120.2K