Modeling and Simulation for Design of Suspended MEMS

Qi Jing

**A dissertation submitted to the graduate school
in partial fulfillment of the requirements of the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering**

**Carnegie Mellon University
Pittsburgh, Pennsylvania 15213**

Committee:
 **Professor Gary K. Fedder (advisor)
 Dr. Tamal Mukherjee (co-advisor)
 Professor Pradeep Khosla
 Dr. Mary Ann Maher (MEMSCAP Inc.)**

May 21, 2003

Report Documentation Page

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **21 MAY 2003**	2. REPORT TYPE	3. DATES COVERED **00-00-2003 to 00-00-2003**

4. TITLE AND SUBTITLE **Modeling and Simulation for Design of Suspended MEMS**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Department of Electrical and Computer Engineering,Pittsburgh,PA,15213-3890**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

14. ABSTRACT **see report**

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**328**	

**To my parents and sister
and to my husband Hao.**

Abstract

This thesis presents a modeling and simulation methodology for the design of suspended MicroElectroMechanical Systems (MEMS), called "NODAS" (NOdal *D*esign of Actuators and Sensors). NODAS simulations are based on schematics composed of a small number of low-level "atomic" elements: anchors, beams, plates and electrostatic gaps. The lumped parameterized behavioral models are implemented in Verilog-A, which is an analog hardware description language. Key issues addressed include schematic representation, modeling physics, modeling accuracy, validation of the composibility and extensibility of the methodology.

Prior work on the NODAS model library is improved by adding in new features and new models. Model physics are expanded from 2D (in-plane) motion to 3D (in-space) motion. Beam and plate models include parameters for both single-conducting-layer processes and multiple-conducting-layer processes. Mechanical physics are enhanced to include beam geometric nonlinearity and shear effects as well as plate elasticity. The electrostatic gap element models allow distributed electrostatic and damping forces acting on electrodes formed from displaced beams. Detailed model derivations are given, followed by verification simulations and discussion on advantages and limitations. The atomic element models have been verified to give simulation accuracy to within 5% of finite element analysis.

This thesis also validates the composibility of suspended MEMS by simulating a set of validation cases using the same small set of atomic elements. The validation cases covers a variety of suspended MEMS devices. Strengths and weaknesses of the current model library are discussed, suggesting directions for future work.

The good simulation accuracy and speed of structured modeling with the library supports iterative design and evaluation. The methodology supports multi-physics analysis and co-simulation with transistor-level electronics, handles hierarchical design for large systems, and therefore acts as a foundation for future system-level MEMS design.

Table of Contents

Acknowledgments

My sincerest appreciation goes to my respectable thesis advisors, Prof. Gary Fedder and Dr. Tamal Mukherjee. Throughout the past five years, they have been continuously supportive of my study and research. Their broad and profound knowledge has been my constant source of help and inspiration and their encouragement has been a great comfort to me when I ran into difficulties. I'm full of gratitude to my advisors for patiently guiding me through my doctoral studies. Particularly, I want to thank Prof. Fedder for his innumerable valuable advice and guidance to the theoretical part of my thesis work. His extremely solid background in many different areas and sharp clear thinking in academic issues deeply impressed me. I heartily thank him for always willing to go through details with me and for being so considerate and nice to me through these years. Having him as my advisor is the best part of my life at CMU. My special appreciation also goes to my co-advisor, Dr. Tamal Mukherjee. His keen insight and knowledge in CAD areas and his amazingly good computer skills have been a great source of inspiration and help to me. I'm also thankful to him for his precious advice on research strategies and issues, for his concern about students and for willing to spare time for discussions with me. It's a great experience working under his guidance.

I would like to thank my inside thesis committee from ECE, Prof. Pradeep Khosla, and my outside committee from industry, Dr. Mary Ann Maher. I heartily thank them for their insightful advice to my thesis work and for spending their precious time reviewing my thesis draft. As an ECE student, I feel honored to have a well-known professor like Prof. Khosla on my committee and feel deeply grateful to him for the many good things he has done for us students as our department head. As a MEMS student, I'm greatly thankful to Dr. Maher for her cooperation with our

group and deeply appreciate her pioneering work and important contribution to the MEMS CAD industry. Her successful career development makes her my role model. It's my honor to have her on my committee.

I would like to thank my officemates and co-workers, Sita Iyer and Bikram Baidya. The academic discussions and personal chatting with them are good memories I will never forget. I learned a lot from them, full of appreciation to their hands-on help and feel very lucky to have them as the companion on our path to PhD degrees.

I would like to thank my peer students and good friends, Xu Zhu, Huikai Xie, Michael Lu, Jiangfeng Wu, Xiaochun Wu, and Hasnain Lakdawala. Discussion with them is inspiring and working with them is enjoyable. It was them who made the MEMS wing such an attractive place to go, and it is their friendships which added a lot of fun to my life.

I would like to extend my appreciation to all of my fellow students, faculties and staffs at MEMS group and ECE department. Special thanks goes to Xu Zhu and Suresh Santhanam for their helps with the device release and SEM, to Mary Moore for her elegance, kindness and consistent help on many many things, and to Prof. Kaigham Gabriel, Roxann Martin, Kai He, Drew Danielson, Debbie Scappatura, Nancy Burgy, Lynn Philibin, Elaine Lawrence and other ECE members who have helped me go through smoothly.

I would like to dedicate my thesis to my parents and thank for their constant unconditional support and encouragement. For me, their love is the strongest source of strength and their arms are the warmest place in the world. I owe them so much letting them living in China by themselves for so many years. I also want to thank my lovely sister. Her cheerful disposition has brought me so much fun and happiness. This thesis is also dedicated to my husband, Hao. I thank him with all my heart for his constant love, support and most considerate care of me.

Finally, I wish everyone who helped me and everyone I love a successful career, a happy family, and the best luck in their lives. And I thank Defense Advanced Research Projects Agency (Agreement F30602-97-2-0304, F30602-97-2-0323), National Science Foundation and the Pittsburgh Digital Greenhouse for supporting this research work.

Chapter 1   Introduction

1.1 Motivation

MicroElectroMechanical Systems (MEMS) are sensor and actuator systems made from microelectronic batch fabrication processes. Applications of arrayed MEMS, such as inertial measurement and navigation systems, micro mirror arrays and high-density data storage, provide tantalizing opportunities for commercialization. However, successful design and manufacture of these kinds of mixed-technology systems is currently very difficult. Single-chip and hybrid versions of these systems will require integration of digital and analog electronics with tens to thousands of mechanical structures, electromechanical actuators, and various sensing elements (*e.g.,* capacitive transducers for motion sensing). Computer-Aided Design (CAD) tools are thus needed to support rapid design of large-scale systems involving physical interactions between multiple physical domains (*e.g.,* mechanical, electrostatic, magnetic, thermal, fluidic, and optical domains).

MEMS design needs are similar to those driving advances in digital and analog circuit CAD. As is the case with pure circuit design, the existence of hierarchical design methodologies, layout synthesis tools and layout verification tools will enable MEMS engineers to build larger systems and enable them to achieve higher levels of integration.

In this thesis, we focus primarily on evaluating candidate designs to support an iterative design methodology. Similar to the computer-aided design for integrated circuits, A good performance evaluation method for MEMS in the design phase should have the characteristics of:

- Good accuracy;

This is the most fundamental requirement on any performance evaluation tool. The simulation results given by the tool should have good agreement to theoretical solutions, to results given by other well-acknowledged tools or to experimental data,

whichever are available.

- Fast speed;

Fast simulations greatly expedite the performance evaluation and the entire design flow. The simulation speed is affected by many factors such as the modeling and simulation methodology, the simulator performance and the hardware configuration of the computers. In this thesis, we focus on how the MEMS modeling and simulation methodology speeds up design iterations as compared to conventional performance evaluation methods.

- Ease of iterative evaluation;

One of the advantages of computer-aided design is that the design can be evaluated and modified prior to costly and time-consuming fabrication. If iterative evaluation is cumbersome, then this potential can not be exploited. Design modifications normally include changes in system structures, choice of models and perturbation of element parameters. A good performance evaluation tool should ease such modifications.

- Ability to handle large system complexity;

Another advantage of computer-aided design is the possibility to handle complicated systems which can not be analyzed by hand calculations. The scale of MEMS has grown rapidly, developing from single functional devices to integrated systems consisting of networks of large numbers of unique functional elements. Hierarchical design and simulation are commonly used to handle the design of large scale integrated circuits. Similarly, MEMS evaluation tools should be able to handle large microelectromechanical systems by supporting hierarchical design and simulation of MEMS.

- Ability to handle multi-physics simulation;

In contrast to electrical circuits, where the electrical behavior is the only concern, MEMS involve many other physical domains and the interaction between them. For example, electrostatic actuation and sensing are commonly used in MEMS design. In this case, the mechanical domain is closely coupled with the electrical domain requiring a self-consistent solution for the entire system. The fluidic domain needs to be considered to

model air damping. Thermal effects are widely used for design of thermal actuators and also need to be considered when the effects of the thermal stress on the mechanical behavior of the structure are non-negligible. Therefore, the performance evaluation tool for MEMS must be capable of modeling the interactions across the mixed physical disciplines.

- Ability to handle co-simulation with transistor-level electronics;

MEMS have evolved from basic devices such as micro resonators to complicated systems with multiple devices, such as integrated inertial measurement units. Interface and signal processing electronics are becoming important parts of the entire system. Accurate capture of the performance of electronics and the interactions with MEMS transducers is thus highly demanded. Simple electrical component models can no longer meet the design needs. To obtain accurate simulation results, the performance evaluation tool for MEMS is expected to support co-simulation of MEMS transducers with transistor-level electronics. Moreover, SPICE-compatible simulation is preferred, in order to take advantage of up-to-date transistor models.

- Coverage of physical effects;

The performance evaluation tool for MEMS is also expected to have a broad coverage of physical effects. The more physical effects are covered, the larger is the application range of the tool. MEMS involve multiple physical disciplines, the physical effects to be covered have much larger varieties than in the case of single-physical-discipline simulations and a single model library can not cover them all. Since not all of the involved physics have significant effect on the device performance and different types of designs have different sets of dominant physics, the evaluation tool is instead required to cover the primary physics involved in certain design space. Furthermore the modeling and simulation methodology should be open to other design spaces so that the application range can be increased as needed.

In this thesis, a performance evaluation tool for the design of suspended MEMS, called NODAS (NOdal Design of Actuators and Sensors), is presented, with focus on the

modeling and simulation methodology. This name was coined in 1997, along with the initial work on this methodology [1]. Detailed discussion and analyses are given through the following chapters, showing that the requirements on MEMS CAD tools as mentioned above are all met in NODAS.

1.2 Suspended MEMS Design Space

1.2.1 Definition

MicroElectroMechanical Systems (MEMS) include various process technologies with each spanning a large design space. This thesis focuses on the specific design space of suspended MEMS, in which all the movable parts of a device are suspended by springs attached to fixed anchor points.

Figure 1.1 gives the structure of a folded-flexure micro resonator, a typical suspended microelectromechanical device [2]. The shuttle mass in the center is suspended by folded springs (called a "folded-flexure") attached to the anchors. The electrostatic comb drives are used for actuation and sensing, with one side attached to the shuttle mass and the other side anchored. When a voltage is applied across the two sides of the comb drive, electrostatic forces are generated and the suspended part is moved in the x-direction.
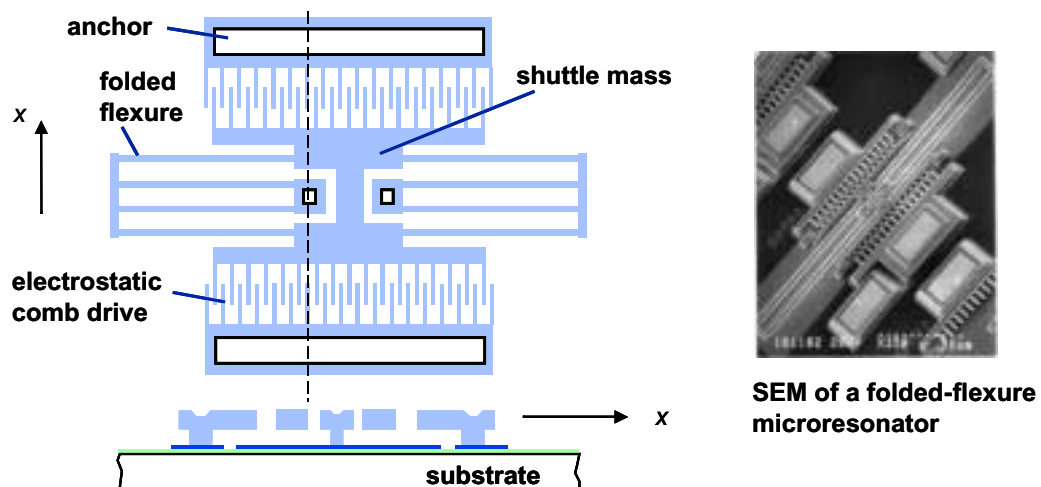
Figure 1.1: Structure of a folded-flexure micro resonator

1.2.2 Reasons to Choose Suspended MEMS

Suspended MEMS is chosen to be the focus of this thesis for two reasons. First, suspended MEMS devices cover a wide range of applications, such as accelerometers [3], micro mirrors [4], RF switches [5], resonator filters [6] and pressure sensors [7]. Many of these devices have been commercialized by industry and are already in large-volume fabrication. Secondly, no other MEMS design space has the near-term potential for integration with electronics in large systems-on-chip (SoC).

1.2.3 Coverage of Suspended MEMS

A survey of published MEMS designs has been undertaken to catalog the application types, physical effects and geometry types in suspended MEMS designs. Table 1-1 categorizes the suspended MEMS applications identified in the survey. Structures such as micro motors with hubs, 3D hinges, fluidic channels and grooves are not included as they are not suspended MEMS. Table 1-1 demonstrates the wide application range of suspended MEMS, and will be used to identify validation cases in Chapter 5.

Table 1-2 categorizes the physical effects and geometry types involved in the suspended MEMS applications listed in Table 1-1. More than 70% of these physical effects and geometry types, indicated by check marks, are covered by the current NODAS model

Table 1-1: Category of suspended MEMS applications

application types				
sensors	pressure sensors	actuators	linear actuators	
	accelerometers		torsional actuators	
	rate gyroscopes		rotary actuators	
	stress sensors		XY stages	
	tactile sensors	micromirrors	torsional horizontal mirrors	
filters	high-Q bandpass filters		piston vertical mirrors	
	variable capacitor filters	switches	RF switches	
			micromachined relays	

library described in this thesis. As can be seen, the covered physical effects focus on mechanical and electrostatic degrees of freedom (DOF). Piezoelectric, piezoresistive and thermal effects, as well as nonlinear elastic plate tension are not included. Curved geometry design styles are also excluded. The methodology of extending the existing model library to these yet uncovered physical effects and geometry types is discussed in Chapter 4.

Table 1-2: Physical effects and layout geometry in suspended MEMS

			covered in thesis
physical effects	beam mechanics	6 DOF (in-plane, beam axial tension and lateral bending)	√
		12 DOF (in-space, beam axial tension, lateral bending and torsion about longitudinal axis)	√
		nonlinear beam bending mechanics	√
	plate mechanics	3 DOF (in-plane, rigid plate)	√
		6 DOF (in-space, rigid plate)	√
		24 DOF (in-space, elastic plate, with in-plane stretching and out-of-plane bending at corners)	√
		nonlinear elastic plate mechanics	
	gap mechanics	2 DOF (in-plane, lateral or vertical motion of rigid electrodes)	√
		12 DOF (in-plane, motions of beam electrodes)	√
		24 DOF (in-space, motions of beam electrodes)	
		contact mechanism	√
	transduction principles	electrostatic (capacitive) transduction	√
		piezoelectric and piezoresistive transduction	
		thermal transduction	
layout geometry		Manhattan geometry	√
		all-angle geometry	√
		curved geometry	

* The numbers of DOF given in the table only include the mechanical degrees of freedom.

1.3 Low-Level Numerical Modeling and Simulation Methodologies

1.3.1 FEA/BEA

Finite-element analysis (FEA) and boundary-element analysis (BEA) are the conventional methods for numerical mechanical and electrostatic simulations. Commercial FEA/BEA tools commonly used by the MEMS design community include ANSYS [8], ABAQUS [9], Maxwell [10], CoventorWare [11], CFDRC [12], Intellisense [13], CAEMEMS [14], SESES [15] and SOLIDIS [16]. These methods first construct system matrices based on the meshing of the continuum mechanical structures and/or the electrostatic fields, and then solve the matrices with boundary conditions. Figure 1.2 (a) shows a mesh of Manhattan brick elements obtained in CoventorWare for a folded-flexure resonator. Figure 1.2 (b) shows a solid and dielectric mesh given by ANSYS for a cantilever beam with the surrounding air gaps [17].

With sufficient refinement of meshing, accurate simulation results can be obtained from FEA/BEA simulations. Moreover, various available types of meshing, such as the *Triangle* and the *Tetrahedra* mesh, allow the modeling of arbitrary shapes. To enable the multi-physics simulation of interactions between the mechanical and electrostatic domains, continuum simulation has been extended by doing separate mechanical and electrostatic
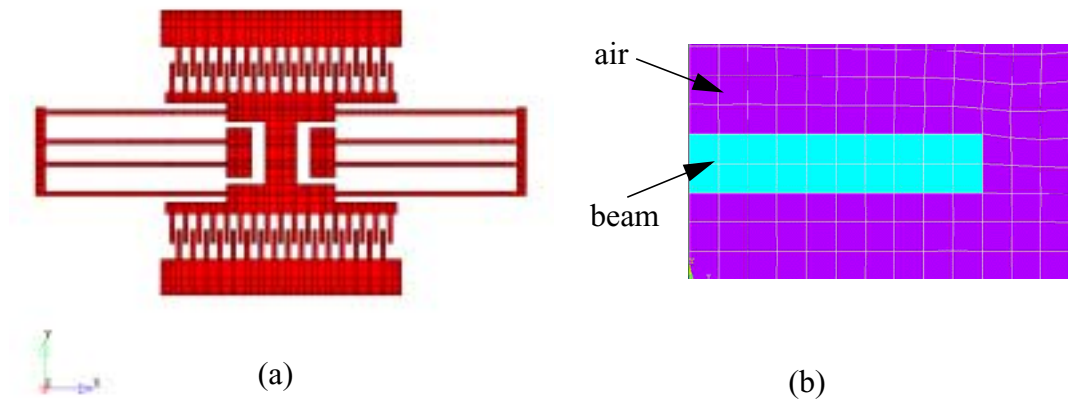
Figure 1.2: (a) A mesh of a folded-flexure resonator using brick elements (b) Solid and dielectric (air) mesh of a cantilever beam

analyses iteratively until a self-consistent convergence solution is obtained [11]. However, as can be seen from Figure 1.2, FEA/BEA methods are layout-based. Any change to the layout requires a new mesh, leading to inconvenient delay during design iteration. In addition, fine meshes lead to large system matrices, thus limiting the simulation speed.

1.3.2 Modeling from FEA/BEA

There are numerous modeling methodologies which aim to overcome the disadvantages in conventional FEA/BEA methods by reducing the degrees of freedom of the system matrices and create macro models for the devices, called reduced-order modeling (ROM).

The macro models can either be derived from FEA/BEA [18][19][20][21][22][23], or from analytical physical equations [1][24][25][26][27][28][29]. The macro model can then either be used in combination with finite element models to speed up the simulation, or be exported into circuit simulators as "black-box" models for system-level simulation. These methods can generate models for entire devices or systems, or models for the individual constituent parts.

Many approaches have been developed for the creation of macro models from FEA/ BEA, including models based on static analysis, on eigenmode analysis or on a combination of both.

The first kind of methods derive the macro models from static analyses. For example, in [20], a macro model for the mechanical spring is obtained by curve-fitting a polynomial to a set of static mechanical finite-element simulations. The displaced position and angular orientation of the mechanical tethers to be modeled along the desired degrees of freedom are varied over the desired ranges of operation. The reaction forces on the tether ends are then collected and fit into a polynomial equation to establish the relation between the forces and the displacements for a mechanical spring model of the device, called a "lumped finite-element model" [22]. Similarly, lumped models for electro-mechanical transducers are

obtained by first running a batch of boundary element electrostatic analyses and capacitance extractions at several displacements, then creating a mapping table or performing a curve fitting to define the capacitance-displacement relationship and self-consistent electrostatic forces are then obtained from these relations [22][30]. The lumped elements span multiple physical disciplines. They act as "black-box" models for the specific device. In this way, thousands of degrees of freedom in the finite element simulations are greatly reduced to a very small set of degrees of freedom of the "black-box".

Some other methods derive the macro models from eigenmode analyses [18][22]. For example, in [22], the effective mass of the device is obtained from eigenvalue analysis then used to form a lumped mass model for the device.

Macro models can be derived from a combination of both static and eigenmode analyses as well [19][22]. Instead of extracting lumped element parameters, which could be cumbersome for large systems, especially when general in-space motions are considered, this method, called the "substructuring method", reduces the degrees of freedom of the entire system matrix by keeping the "master" degrees of freedom and condensing out the "slave" degrees of freedom. The partition of "master" and "slave" degrees of freedom depends on the modes presented in the dynamic response of the device. Figure 1.3 gives an
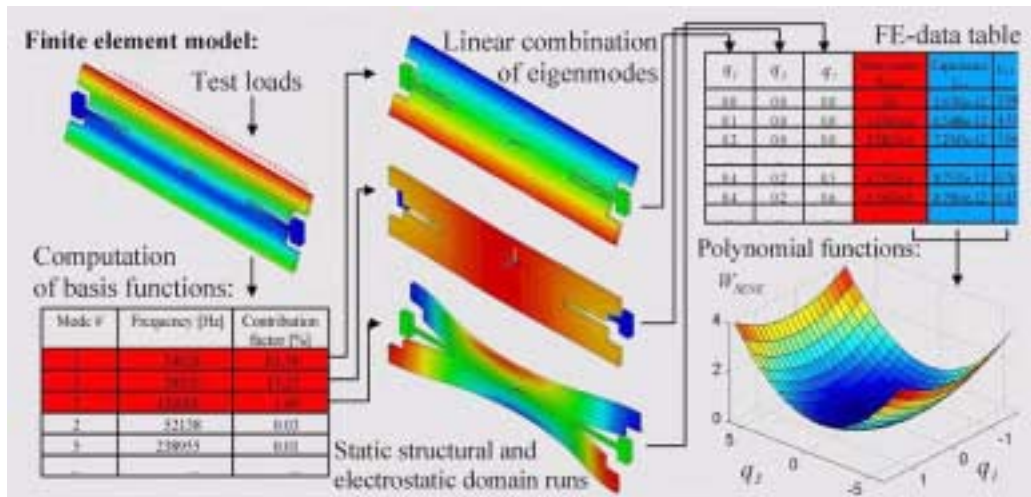
Figure 1.3: Generation of ROM for a micro mirror [31]

example of the ROM generation for a micro mirror [31]. First, a mesh is created for finite element analysis. Then a test load is applied to simulate the primary motion of the specific device and a modal analysis is performed to compute the mode shapes. The specific load deflection is then compared to the mode shapes to determine the contribution factor of each mode, giving a prioritized list indicating which basic shapes are dominant for the specific structure ("master" modes). Contribution factors of those dominant shapes are the generalized coordinates used in the reduced-order macro model. Finally, the macro models for each energy domain are assembled to form a macro model for the system [31].

The lumped finite element method is suitable for small systems for which the extractions of lumped parameters are relatively easy, and the substructuring method is more suitable for large systems for which the extractions are cumbersome. Both methods enable multi-physics simulation, including static as well as dynamic analyses. With parameterized macro models [20], the device geometric sizes can be modified conveniently, making the iterative design easier than using the macro models with both topologies and sizes fixed.

1.3.3 Mixed-Level Simulation

With careful attention to interoperability, the reduced-order models from FEA/BEA can be connected according to the device topology to form the reduced-order macro model for an entire device, as shown in Figure 1.4 [22]. As the across and through variables used
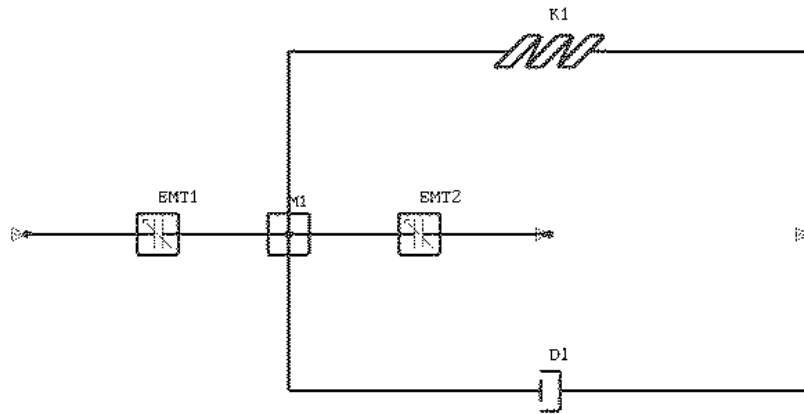
Figure 1.4: ROM for a folded-flexure resonator using lumped elements [22]
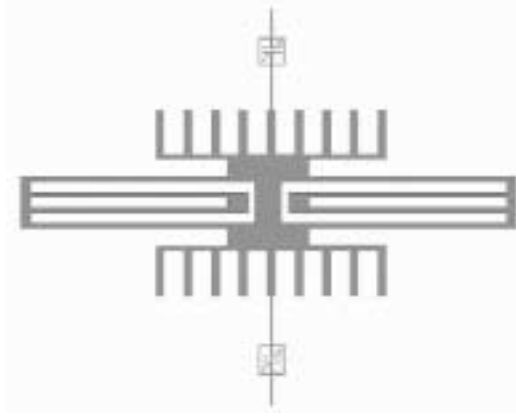
Figure 1.5: Finite element model for a folded-flexure resonator, with ROM transducers replacing the electrostatic comb drives [22]

in the lumped elements correspond to degrees of freedom of the standard finite elements, the lumped elements can also be used as concentrated loads and displacements in combination with distributed finite element models. Figure 1.5 shows the finite element model for a folded-flexure resonator with ROM transducers replacing the electrostatic comb drives [22]. In commercial finite-element tools, lumped models such as the Euler beam model, plate model and shell model are available in the library together with continuum models such the cubic element model [9]. These models are interoperable with each other at mixed level, providing more flexibility in simulations.

In addition, the macro models can be exported and transferred into an analog Hardware Description Language (HDL) and then inserted into an analog circuit simulator as a black-box model of the electromechanical system. As the currently available reduced-order modeling from FEA/BEA only supports a limited set of electrical lumped elements including resistors, capacitors, inductors, diodes, voltage sources and current sources [22], the capability of simulating systems with complicated electronics is limited. Exporting reduced-order models to aHDLs enables co-simulation with transistor-level circuits and allows use of transistor models available in modern circuit simulators [32][33][34].

1.4 Higher-Level Behavioral Modeling and Simulation Methodologies

The reduced-order macro models from FEA/BEA speed up the simulation by greatly reducing the degrees of freedom in the system matrices, however, such modeling is usually not fully coupled within design iteration loops for higher-level components. Layout topology changes during design require new macro models derived from a new batch of FEA/BEA simulations. A higher-level modeling and simulation methodology is needed to further relieve the designers from modeling so that they can better concentrate on the design. Hence, in recent years, circuit-based structural modeling for system-level simulation has been proposed.

1.4.1 Suspended MEMS Hierarchy

Suspended MEMS may be represented by a hierarchy of composable elements. For example, Figure 1.6 shows the hierarchical abstraction of a folded-flexure resonator system. The folded-flexure resonator system consists of the MEMS transducer part and the interface electrical circuit. In electrical circuit construction, systems may be represented at the lowest level by lumped circuit elements such as resistors, capacitors, inductors and transistors. As
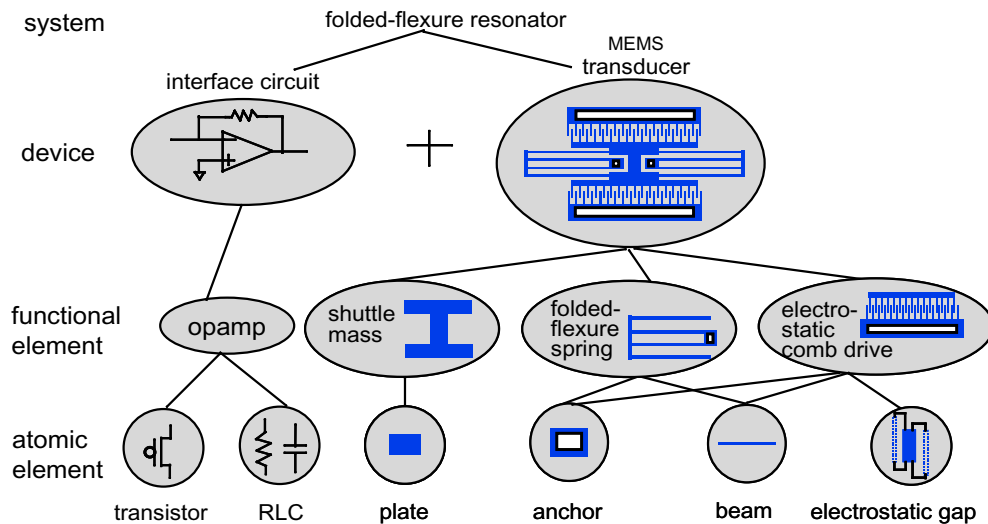
Figure 1.6: Hierarchical abstraction of a folded-flexure resonator

shown in the figure, the MEMS transducer has a functional hierarchy analogous to the electrical circuit. The MEMS transducer may be partitioned into *functional elements* including a shuttle mass, folded-flexure springs and electrostatic comb drives. The functional elements are then further decomposed into *atomic elements* including anchors, beams, plates and electrostatic gaps. Similar hierarchical abstraction is applicable to many other suspended MEMS systems, such as accelerometers [35], gyroscopes [36], and bandpass filters [37].

1.4.2 MEMS Modeling by Structure ("MEMS Circuit Building")

Suspended MEMS system may be structurally composed from behavioral models at any level in the hierarchy in any combination. The models are interoperable to support mixed-level hierarchical simulation. An example is given in [38], showing a nested-resonator system composed of functional-level spring models and atomic-level plate models.

Both behavioral modeling and structural modeling describe the physical behavior by formulating the relation between the variables at the interface connection terminals of the component. Here we define a "behavioral model" as a ROM of explicit ordinary differential equations, usually formed by the techniques described in Section 1.3.2. In contrast, a "structural" model is formed by interconnecting components in a circuit. Structural modeling enables the use of hierarchy in design. The elements in the model libraries may be low-level, reusable models, serving as building blocks for large systems. If suitable low-level, parameterized models are available, designers don't need to create new behavioral models as part of the iterative design loop. He or she just chooses the appropriate elements from the libraries based on the needs of his/her design, then interconnects these elements to form a model of the entire system. The circuit simulator then solves for a self-consistent solution for the system.

In an evolutionary step toward this design vision, the thesis presents a low-level

"atomic" behavioral modeling library to support structural modeling of suspended MEMS. The atomic elements are device-independent, lumped and geometrically parameterized. The initial models in this thesis are derived from analytical equations rather than curve-fits of FEA results. The distributed elastics, inertia and damping of the elements are lumped to a finite number of connection terminals, with the physical constitutive relation between the terminals modeled as matrices. The geometric parameterization greatly simplifies the iterative evaluation of the impact of size changes on the device performance. System matrices formed by the circuit simulator are much smaller than those from FEA, but larger than the lumped-element reduced-order modeling from FEA. From discussion in later chapters, we will see that as the atomic models are composable in much the same way as lumped finite elements. For example, using more atomic elements to compose beam springs gives better simulation accuracy.

In circuit-level behavioral simulation, the models are implemented in analog hardware description languages (aHDLs) or directly in element matrices, both inherently supporting simulations in mixed physical domains. Existing circuit-based representation or support for suspended MEMS include NODAS from Carnegie Mellon University [1], MEMS Pro [24], MEMS Xplorer [25] and MEMSMaster [29] from MEMSCAP, ARCHITECT from Coventor [26], SUGAR from UCBerkeley [27], and Chatoyant from Univ. of Pittsburgh [28]. The major distinctions among the existing tools are the representation styles, the design frameworks and the availability of specific model libraries and tool functions. For instance, in SUGAR, models are implemented directly in matrices, systems are represented in netlists and simulations are performed in Matlab; in ARCHITECT, models are written in MAST, systems are represented in the form of schematics and simulations are performed in Saber; in MEMS Pro and MEMS Xplorer, bi-directional links between 3D solid models in ANSYS to 2D mask layout in both Cadence and Mentor database formats are provided. For the NODAS approach described in this thesis, models are written in Verilog-A [39] for schematic-based simulation in Cadence Spectre [33][34]. These different features lead to

varied capabilities in handling co-simulations and other important design steps such as layout editing and verification.

In this thesis, a detailed description of the modeling and simulation methodology in NODAS will be given in Chapter 2. Chapter 3 gives the detailed model derivations and simulation verifications. Chapter 4 discusses the extensibility of NODAS methodology to new physical effects, processes, design styles and physical domains. Chapter 5 shows the simulations of a set of validation cases. Chapter 6 is the summary and future work.

Chapter 2   Schematic-Based Circuit-Level Behavioral Simulation in NODAS

This chapter focuses on the circuit-level behavioral simulation in NODAS. Several key issues about the modeling and simulation in NODAS are discussed, including the composition property of suspended MEMS, schematic representation, behavioral modeling, validation case design and extensibility.

NODAS uses analog HDLs, lumped parameterized behavioral modeling and SPICE-compatible simulators for circuit-level simulation of suspended MEMS. Design with NODAS starts from schematic entry, where MEMS transducers and electrical circuits can co-exist. The schematic is composed of elements from a cell library. The element instances are connected according to the topology of the device. A composite netlist for the entire system is then generated automatically and sent to the numerical solver for performance evaluation.

The NODAS cell library consists of a set of atomic elements including anchors, beams, plates and electrostatic gaps. Each element has a lumped behavioral model, a symbol, and a set of process-independent parameters for the geometric sizes of element instances. Additionally, all the elements are associated with a set of process-dependent parameters for material properties such as Young's modulus and mass density.

The behavioral models are written in an analog HDL. An early version of NODAS was implemented in the Analogy design framework with models written in MAST and simulation in Saber [1]. Current version of NODAS is implemented in the Cadence design framework with models written in Verilog-A [39] and simulation in Spectre [33][34]. The analog HDLs can encode physical effects co-existing in multiple physical disciplines in a behavioral model. They also allow each physical discipline to have its own individual tolerance option and blowup limit, which helps in the convergence for simulations involving

multiple physical disciplines with significantly different magnitudes of variable values. The transfer to Cadence design environment enables use of a widely used SPICE-compatible simulator, Spectre, which fully supports co-simulation with transistor-level electronics. The compact models for transistors can be easily included in the simulation to provide accurate evaluation of the electrical circuit performance. The implementation in Cadence also enables integration with the widely used commercial layout editing and verification tools thus leading to a complete integrated MEMS design flow. Similar tools supporting simulation with Verilog-A models include NanoSim from Synopsys [40] and ADVance MS (ADMS) from Mentor Graphics [41].

The modeling and simulation methodology in NODAS eases design iterations as well. Unlike FEA/BEA and reduced-order modeling, the design and modeling in NODAS are decoupled. The lumped behavioral models are derived from analytical equations rather than curve-fitting to FEA simulation results. They are reusable and not device-specific, capable of serving as building blocks for complicated systems. The models are geometrically parameterized, allowing the sizes of each element instance to be modified individually. Besides, the structural modeling nature of circuit-level schematic representation makes it easy to modify the device topology. Therefore, there is no need to regenerate the meshes or device macro models for design modifications in NODAS. The iterative design is greatly eased.

In addition to these characteristics, NODAS also supports hierarchical modeling and simulation, which helps with the handling of system complexity. In addition, the modeling and simulation methodology in NODAS is extensible to new design spaces, for full development into a comprehensive performance evaluation tool for the entire MEMS design community.

In the following sections, these above issues are detailed. This chapter also serves as a background chapter to help understand the atomic-level models to be discussed in Chapter 3, the extensibility issues to be discussed in Chapter 4 and the validation simulations to be

discussed in Chapter 5.

2.1 Composition Property of Suspended MEMS

The modeling and simulation methodology of NODAS is based on the hypothesis of the *composition property of suspended MEMS*: a small set of atomic elements, including anchors, beams, plates and electrostatic gaps, can compose a large variety of suspended MEMS designs.

As shown in Figure 1.6, MEMS transducers have a functional hierarchy analogous to the electrical circuits. They can be decomposed into atomic elements which are general elements reusable for multiple blocks of suspended microelectromechanical structures. The modeling of these atomic elements and the validation of the composition property of suspended MEMS are the major efforts of this thesis work.

The hierarchy existing in suspended MEMS makes it possible to do hierarchical modeling and simulation, which helps dealing with large system complexity as in IC designs. The schematic representation and the behavioral modeling are accordingly designed to enable the hierarchical representation.

2.2 Schematic Representation

In a schematic, connection terminals of element instances are represented by groups of pins. As physical effects in multiple or mixed physical domains are involved in MEMS elements, the connection terminals need to convey information in multiple physical disciplines. Therefore, the pins at connection terminals are a groups of individual pins, each having an associated discipline determining its physical nature.

2.2.1 Pin Definition

Pin definition directly affects the schematic composition. Figure 2.1 (a) and (b) explain
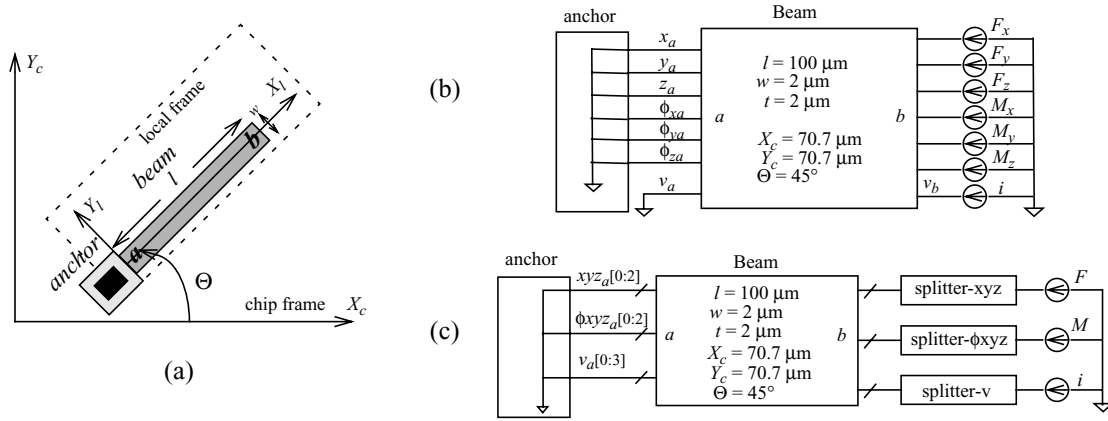
Figure 2.1: Pin definition in NODAS (a) a cantilever beam on a chip (b) schematic with individual pins (c) schematic with bus pins

the pin definition using a cantilever beam as an example. The beam behavior is lumped at terminal *a* and terminal *b*. Each terminal has three translational pins, named *x*, *y*, *z*, to represent the translational motions along the *X*, *Y* and *Z* axes, and three rotational pins, named ϕ_x, ϕ_y, and ϕ_z, to represent the rotational motions about the *X*, *Y* and *Z* axes. The beam is also an electrical conductor, which is crucial for electrostatic actuation and sensing. Thus, electrical pins v_a and v_b are also needed.

2.2.1.1 Bus Pins

Schematic assembly consumes much effort and is prone to error when the schematic has many elements. As indicated in Figure 2.1(b), at least seven wires are needed for each MEMS element terminal. In digital circuit schematics, buses are used for compaction of schematic representation. Similarly, analog buses are used in NODAS, as shown in Figure 2.1(c). Ideally, all the pins would be combined into one bus. However, due to the limitations in analog HDLs, only pins sharing the same discipline can be grouped as a bus. Thus three buses are needed for each terminal (translational, rotational, and electrical, respectively). The compact terminal representation reduces wiring effort as well as the probability of wiring errors. Behavioral blocks that convert scalar wires to bus wires ("splitters" in Figure 2.1(c)) are used to conveniently apply stimuli to individual degrees of freedom.

Elements with multiple connection terminals also benefit from the use of bus pins. In later sections, we will see that the plate element and the electrostatic gap element have many more connection terminals than the beam element. Using bus pins greatly eases schematic composition using these elements.

2.2.1.2 Global Nets

In addition to bus pins, global acceleration nets are used to reduce the clutter in schematic. When the chip is under an external acceleration or rotation, the same amount of excitation must be applied to every element on the chip. If wires have to be routed throughout the schematic to pass the acceleration and rotation information for each degree of freedom, the schematic will be very cumbersome. Hence, similar to the global ground in electrical circuit schematic composition, global nets for external acceleration and rotation are employed in NODAS, allowing the associated physical variables to be pass around through different schematics and different design hierarchies without the need for specific hierarchical pins and wires at the schematic interfaces.

There are three global translational acceleration nets: *ax_ext* for acceleration in *x*-direction, *ay_ext* for acceleration in *y*-direction, and *az_ext* for acceleration in *z*-direction. Rotations are represented by the rotation angular rates rather than the rotation angles, because the latter is normally the major concern especially for devices such as micro gyroscopes. The global rotation rate nets are: Ωx_ext for rotation angular rate about *x*-axis, Ωy_ext for rotation angular rate about *y*-axis, and Ωz_ext for rotation angular rate about *z*-axis.

As the physical variables associated with the acceleration and rotation rate nets need to be measured in the Verilog-A model and communicate with other variables, corresponding pins have to be defined in the Verilog-A view and thus in the symbol view as well. In electrical circuit schematic, global nets are usually defined by adding an exclamation mark "!" to the net names. Cadence's schematic entry tool (Composer) follows the same naming
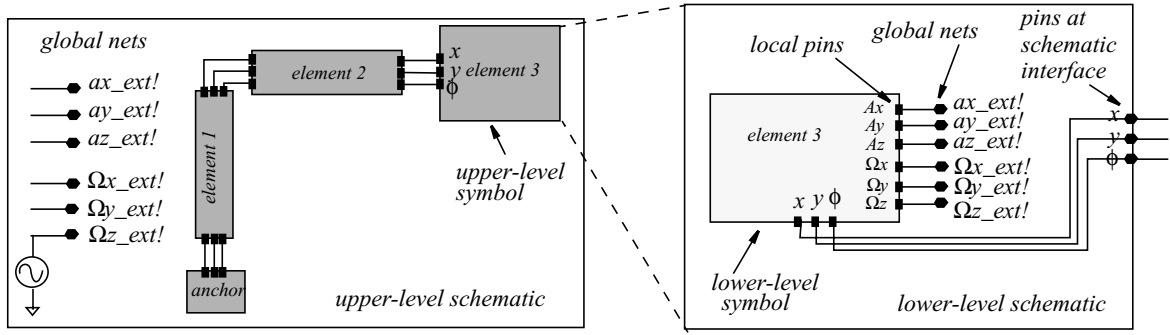
Figure 2.2: Definition of global external acceleration and rotation rate pins using hierarchical schematic

convention. Hence the most direct way to implement global nets in NODAS would be creating pins for the translation accelerations and rotation rates in the symbol view of each model, create nets for these pins in the schematic and then label the nets by names with an exclamation mark. As there are six global pins in each element instance, creating wires and naming the nets for all the instances in the schematic would be too cumbersome to meet the initial motivation of using global nets. Another method is to name the acceleration and rotation rate pins directly with an exclamation mark to make these pins global and to eliminate the need for creating wires. However, study shows that pin names such as "*ax_ext*!" are not allowed by Verilog-A syntax, thus this method is also not feasible. To solve these problems, hierarchical schematics are employed in NODAS, as shown in Figure 2.2.

In this implementation method, each element has an upper-level symbol view as well as an associated schematic view. There is no Verilog-A model attached and no external acceleration and rotation rate pins defined in the upper-level symbol view. Global translational acceleration and rotation rate nets (*ax_ext*!, Ωx_ext!, etc.) are defined in the associated lower-level schematic view, in which a lower-level symbol with local acceleration and rotations rate pins (*Ax*, Ωx, etc.) are also included. The local pins are connected to the global nets in the lower-level schematic to obtain the global acceleration

and rotation rate information, while they are not defined as hierarchical pins at the interface of the lower-level schematic and thus do not appear in the upper-level symbol view. The core Verilog-A model which describes the element behavior is associated with the lower-level symbol instead of the upper-level one. In this way, external acceleration and rotation rate sources can be instantiated in top level of the hierarchical schematics and passed down to every element on the chip through global nets, without conflict with the Verilog-A syntax and without clutter in the schematic. Parameters are specified by the user in the upper-level schematic and passed down to the lower-level symbol and Verilog-A views by using hierarchical parameter transfer syntax provided by Cadence design tools.

2.2.2 Across and Through Variables

Another representation issue is the definition of across and through variables. In electrical domain, the sum of current flowing into a node with n branches equals zero, specified as Kirchhoff's current law (KCL): $\sum_{k=1}^{n} i_k = 0$. The sum of voltages around a loop with m elements equals zero, specified as Kirchhoff's voltage law (KVL): $\sum_{k=1}^{m} v_k = 0$.

Currents flowing *through* the nodes of electrical elements are defined as *through* variables satisfying KCL, and voltages dropping *across* the electrical elements are defined as *across* variables satisfying KVL. Analogously, mechanical systems in translational motion must be in dynamic equilibrium: $\sum_{k} F_k = 0$. Using the analogy of force balance to KCL, forces are defined as through variables for translational discipline.

There are three choices for the definition of across variables for the translational discipline: positions, displacements or velocities. They all satisfy KVL. Displacements are preferred in suspended MEMS as they are usually of primary interest to be directly observed. Using displacements also enables the kinematics to be modeled explicitly. For

cases where velocities are the primary interest, using velocities as across variables may be a better choice. Using velocities is also convenient for cases where power is the major concern, since the product of force and velocity is the mechanical power which is analogous to the product of voltage and current as the electrical energy in electrical domain. In NODAS, displacements are defined as the across variables, similar to other MEMS CAD tools [24][26][27]. For rotational discipline, angular displacements are defined as across variables and angular moments are defined as through variables.

Multiple disciplinary across and through variables are very important for the support of multi-physics co-simulation. It allows the physical effects in multiple disciplines to be described in a uniform format: all physical effects are formulated into equations specifying the relations between the associated across and through variables at connection terminals. Communication between elements is thus available through these variables at the connection terminals of the same discipline, even between MEMS elements and electrical transistors.

2.2.3 Element Parameters

As mentioned in previous sections, NODAS models are geometric parameterized behavioral models, therefore, parameters are indispensable for the modeling and simulation.

There are two sets of parameters: material parameters and geometric parameters. Material parameters specify the mechanical material properties such as Young's modulus and mass density, the electrical material properties such as sheet resistance, and the properties of the ambient environment such as the viscosity of air, the distance to the substrate and the temperature of the microstructure. These parameters are generally process-dependent. Therefore, in the current version of NODAS, the default values for these parameters are encoded in a process header file. In order to enable simulations of manufacturing variations, these parameters are defined as user-specifiable parameters, so that the user can edit these parameters individually for each element instance.

Geometric parameters include size parameters and position parameters. Size parameters specify the geometric sizes of the elements, such as the length, width and thickness. Position parameters specify the orientation angles and the initial position of the element instance on the chip. For instance, as shown in Figure 2.1, the beam forms an angle of Θ with the coordinate system of the entire chip. The elements interact with each other in the common chip frame of reference, however, the beam mechanics is modeled based on the local coordinate system of the beam element as illustrated in the figure. Therefore, the beam model must know about the orientation angle of the beam instance, Θ, in order to correctly transfer information between the local frame and the chip frame. In addition, the initial layout position of the beam center relative to the layout origin of the chip frame, (X_c, Y_c), are also important for simulations especially for inertial sensors. As the initial layout positions are static values which do not change during dynamic simulation, they are represented as parameters. The values of X_c, Y_c can either be entered by the users or automatically calculated by a topological connectivity analysis algorithm based on the element connections and the element sizes in the schematic.

2.3 Behavioral Modeling

Geometrically parametrized lumped behavioral models are the core of the circuit-level simulation methodology of NODAS.

The models are derived from structural matrix analysis, which lumps the distributed element behavior to a limited number of terminal nodes and describes the physics of the element in the form of matrices. As an example, the linear beam model includes linear beam tension, bending and torsion, as well as inertial and damping effects. External forces and moments are restricted to concentrated loads at the beam ends. Using energy methods, beam tension, bending and torsion are modeled as a stiffness matrix $[k]$, derived from beam mechanics equations [42]. Beam inertia is modeled as a mass matrix $[m]$ and damping is

modeled as a damping matrix [*B*], both derived by assuming that the static mode shapes are effective for dynamic motion [43][44]. The relation between the force/moment vector and the displacement vector is established as:

$$\left[F_{beam}\right] = \left[m\right]\left[\ddot{x}\right] + \left[B\right]\left[\dot{x}\right] + \left[k\right]\left[x\right] \tag{1}$$

where for a 3D beam element, $[F_{beam}] = [F_{xa}\ F_{ya}\ F_{za}\ M_{xa}\ M_{ya}\ M_{za}\ F_{xb}\ F_{yb}\ F_{zb}\ M_{xb}\ M_{yb}\ M_{zb}]^T$, $[x] = [x_a\ y_a\ z_a\ \phi_{xa}\ \phi_{ya}\ \phi_{za}\ x_b\ y_b\ z_b\ \phi_{xb}\ \phi_{yb}\ \phi_{zb}]^T$, and node *a* and *b* are the terminals at beam ends.

Other atomic-level models follows the same definitions of force and displacement vectors. All the models describe the element physics in a uniform format by specifying the relation between the across and through variables in matrices, although the physical disciplines and the particular equations vary widely from model to model.

Modeling the atomic-level elements is a major effort of this thesis work. The model details will be given in next chapter with simulation verifications.

2.4 Coverage of NODAS Model Library

The atomic-level cell library in NODAS covers the physical effects and geometry types checked in Table 1-2 of Chapter 1, and includes:

- anchor model;
- layout_origin model;
- linear beam model with tension, bending and torsion;
- nonlinear beam model with geometric nonlinearity;
- rigid plate model;
- elastic plate model with in-plane stretching and out-of-plane bending;
- electrostatic gap model with beam electrodes and snap-in contact mechanism.

A major portion of the atomic elements in suspended MEMS designs is covered by these models. However, there still exist unchecked cases in Table 1-2. New technologies and

designs require extensions of the current NODAS model library. The issues about the extensibility are discussed in Chapter 4.

2.5 Validation

As mentioned at the beginning, the composition property of suspended MEMS is the basis of the NODAS modeling and simulation methodology and needs to be validated. The function and accuracy of the atomic models also need to be verified to be reliable models of practical use. The validation work is a very important portion of the thesis effort.

Validation is performed at the atomic level. All the validation cases are simulated using the specified set of atomic-level models, with focus on different physical effects. The validation case set given in Table 2-1 is chosen from a wide range of suspended MEMS applications, including bandpass filters [6][45], RF switches [5][46], micro mirrors [4][47], large-stroke actuators [48] and capacitive pressure sensors [7]. Table 2-1 shows that the checked physical effects and geometry types in Table 1-2 are covered by these validation cases. The function and accuracy of the atomic models are verified by comparing the simulation results from NODAS to the results from FEA/BEA or to the experimental data. The validation cases will be described and analyzed in detail in Chapter 5.

Table 2-1: Validation case set

physical effects and geometry types	bandpass filter	RF switch	micro mirror	large-stroke actuator	capacitive pressure sensor
beam bending	x	x		x	
beam torsion			x		
geometric beam nonlinearity		x		x	
elastic plate bending					x
capacitive transduction	x	x	x	x	x
contact mechanics	x	x	x	x	x
Manhattan topology	x	x	x		x
non-Manhattan topology				x	

2.6 Extensibility

A useful modeling and simulation methodology should be applicable to a wide range of applications, while a single library can not be expected to cover every desired feature. However, as long as the methodology is extensible to new physical effects, new processes, new design styles, and new physical domains, further improvement is possible and the application range can be increased as needed.

Any model is an abstraction and cannot include all the physical effects involved in the actual physical elements. Physical effects which are negligible in certain design spaces may become non-negligible in some other design spaces. Hence adding new physical effects is the most fundamental type of expansion to the application range of existing models. The addition of shear effect to the beam model will serve as an example to show the extensibility of NODAS methodology to new physical effects. An extension guideline will be given.

A large variety of processes exist for suspended MEMS fabrication. The extensibility of the NODAS methodology to new processes will be shown by the extension of beam model from single-conductor process to multiple-conductor CMOS-MEMS process and by simulations of the design cases given in Table 2-1, which include devices fabricated in a wide variety of processes. A list of general parameters needed for the extension to new processes will be given.

Design styles come in many varieties. For instance, while many devices have Manhattan topology and use straight beams with rectangular cross-sections, other design styles such as non-Manhattan topology, curved beam and trapezoidal cross-sections are employed or inevitable in some design cases. Therefore, the cell library should be extensible to these kinds of new design styles. A general routine will be formed to help create models for new-style elements based on prototype models.

In addition, as MEMS designers like to exploit all possible useful or interesting applications that occurs at micro scale, extensibility to new physical domain is important as

well. Physical domains involved in MEMS are not restricted to mechanical, electrical and electrostatic domains. In recent years, research on thermal, optical, piezoelectric, fluidic and even bioMEMS areas have attracted more and more interest. The modeling of curling effect caused by the thermal stress gradient in layer stacks of CMOS-MEMS beams will be briefly introduced to show the extensibility of NODAS methodology to new physical domains.

The extensibility issues mentioned above are discussed in detail in Chapter 4.

Chapter 3   NODAS Atomic-Level Model Library

Geometrically parameterized lumped behavioral models are the core of the NODAS-based simulation methodology, in which the distributed element behaviors are lumped to a limited number of terminal nodes. Atomic-level NODAS models for the design of suspended MEMS include beams, plates, electrostatic gaps, anchors and layout_origin. In this chapter, the model physics are given in detail, followed by verification simulations. The function and limitation of each model are clearly stated to guide the practical use of these models.

3.1 Anchor Model

Anchors are mechanical pillars supporting the suspended structures. They are the fixed points through which the movable suspended structures are attached to the chip substrate. Anchors have finite compliance, depending on the aspect ratios and material properties [49]. However, in most cases, anchors are much stiffer than the suspension springs, therefore, in NODAS, we model the anchor element as an ideal rigid element with infinite stiffness - any point attached to the anchor element has absolutely *zero* displacements in all directions. This abstraction is widely used in many MEMS CAD tools.

Figure 3.1 gives the symbol view of the 3D anchor element in NODAS. It has two bus pins, specifying *zero* translational displacements in *x*, *y* and *z* directions, and *zero* rotational displacements about *x*, *y* and *z* axes.

ANCHOR 3D

$w = 10\,\mu$ $x <0:2>$ ■
        $phi <0:2>$ ■
        $l = 10\,\mu$
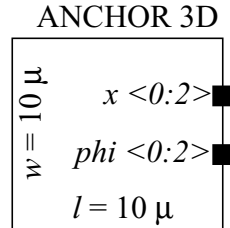
Figure 3.1: 3D anchor element - symbol view

The Verilog-A model is given as below:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
// module begins
module anchor3D(x, phi);
// pin definitions
inout [0:2] x; kinematic [0:2] x;
inout [0:2] phi; rotational [0:2] phi;
// parameter definitions
parameter real l = 0, w = 0;
// analog block begins
analog begin
// zero translational displacements
Pos(x[0]) <+ 0; Pos(x[1]) <+ 0; Pos(x[2]) <+ 0;
 // zero rotational displacements
Theta(phi[0]) <+ 0; Theta(phi[1]) <+ 0; Theta(phi[2]) <+ 0;
end  // of analog block
endmodule  // of module
```

Anchor elements are assumed to have a rectangular shape with two geometric parameters: *l* and *w*, as illustrated in the symbol view. These parameters are only used for automatic layout generation [50], and do not affect the simulation.

3.2 Layout_origin Model

Layout_origin is an element specifying the coordinate origin of the layout. It defines the reference point for the calculation of center layout positions, X_c and Y_c, for instances of all MEMS elements in the schematic. As mentioned in Chapter 2, X_c and Y_c are used for automatic layout generation (layout orientation angles are entered by the user as parameters) and for simulation of inertial sensors.

Since *layout_origin* specifies layout position, not displacement, theoretically, it can not be connected to other elements through translational pins whose across variables represent displacements. However, as the connectivity-based automatic layout generation algorithm uses layout_origin as the reference for calculation of X_c and Y_c of other elements, layout_origin has to be included in the schematic in such a way that the connection does not

affect simulation. Our solution is: connect the layout_origin to one of the anchors in the layout. In this way, the zero-layout-position specified by the layout_origin is compatible with the zero-displacement specified by the anchor element hence the simulation is not affected by the connection to the layout_origin element. Figure 3.2 gives the symbol view of the layout_origin element and the wiring connection to the anchor element and the beam element. Unlike the anchor element, layout_origin only has one translational bus pin.

The Verilog-A model of the layout_origin is given as below:

```
'include "../../constants.h"
'include "../../discipline.h"
module layout_origin(x);
inout [0:1] x;
kinematic [0:1] x;
parameter real X0 = 0;
parameter real Y0 = 0;
analog begin
            // empty analog block
end
endmodule
```

The analog block of the model is empty as the layout_origin doesn't specify any physical relations between the across (displacements) and through (forces) variables of the terminal pin. The element's two parameters, X_0 and Y_0, both have a default value of zero
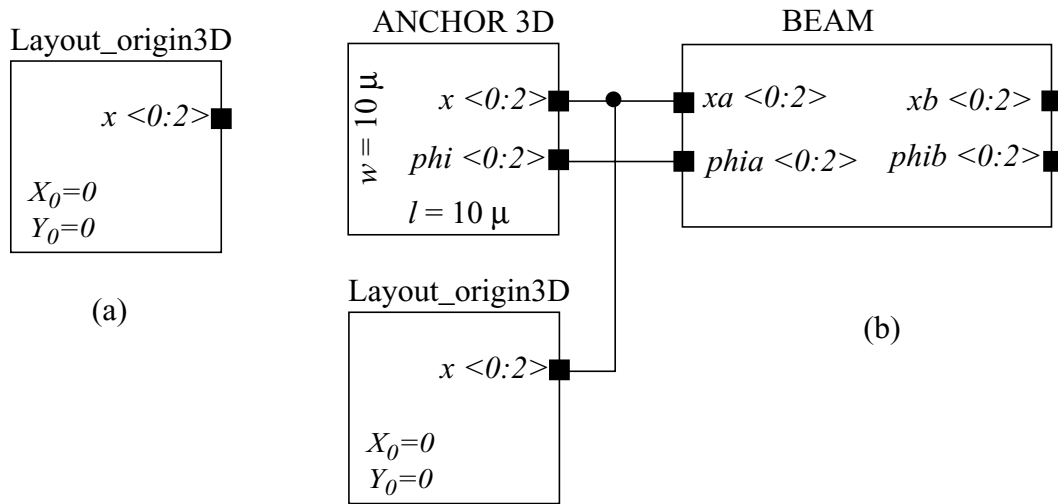
Figure 3.2: (a) Symbol view of Layout_origin3D (b) Connectivity of layout_origin to the anchor and beam element.

which is true for most cases where the rotation center is at the layout origin of the chip. For cases where the rotation center falls outside of the chip, non-zero values of X_0 and Y_0 must be entered by the user. The center coordinates relative to the chip layout_origin, X_c and Y_c, are then calculated by the automatic layout generation code based on the values X_0 and Y_0 and the connectivity of elements instances and used for layout generation and the calculation of centrifugal forces and coriolis forces.

3.3 Beam Model

Beams are one of the most fundamental constituent elements of suspended MEMS devices. They are building blocks for a large variety of suspension springs.

In this chapter, the cross-section of beam element is restricted to cuboid shape. The basic geometric parameters are length L, width w, and thickness t, as shown in Figure 3.3. Fundamental material properties are Young's Modulus E, shear modulus G, and mass density ρ. The behavior of the beam element is lumped to node a and b, which are physically located at the cross-section centers of the beam ends.

Beam mechanical behavior stems from a combination of multiple physical effects, including linear axial compression and tension, linear bending, geometric nonlinear bending, torsion and shear effects. Which physical effects are important in the behavior of a specific beam element is determined by the geometric aspect ratios, load conditions, and application ranges. Table 3-1 gives a list of different cases with corresponding dominant
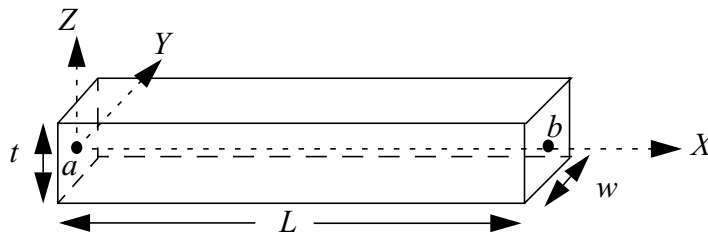
Figure 3.3: Geometric parameters of the beam element

physical effects. These are the major beam mechanical effects covered by the NODAS beam models. In the following sections, the analysis and modeling for each of these effects will be presented.

Table 3-1: List of Cases with different dominant beam mechanics

types		dominant physical effects
aspect ratios	$L \gg w, L \gg t$	beam bending without shear effect
	$L \cong w, L \cong t$	beam bending with shear effect
load conditions	axial	axial compression/tension
	lateral	lateral bending
	torsional	torsion about longitudinal axis
	lateral & axial	stress-induced nonlinear beam stiffening
application range	small displacement ($<=15\%$ L)	linear beam mechanics
	large displacement ($>15\%$ L)	geometric nonlinear beam mechanics

As nonlinear beam mechanical effects and shear effects are negligible in some design cases and the inclusion of these effects induces additional internal variables to the model and much longer coding thus slows down the simulation, the designer may want to have the choice to neglect some of these effects according to the design need. Therefore, in NODAS, the beam mechanical effects are grouped into four types of beam models: linear beam model with/without shear effect and nonlinear beam model with/without shear effects.

3.3.1 Linear Beam Model

In NODAS, the beam cross-section is restricted to rectangular shape and the forces and moments are restricted to concentrated loads at beam ends. The distributed beam behavior is lumped to the beam ends and implemented in matrix format using the *structural matrix analysis* method [43][44][44]. Linear beam mechanics include linear stiffness, inertia and damping effects. In this section, we will give a brief description of the modeling for these effects, as they are the fundamental features of the NODAS beam model and the basis of

other important features. Coordinate transformations between different frames of references are also presented to show how one set of physical equations and matrices are used to deal with beam elements of arbitrary layout orientation angles. The algorithm structure of the linear beam model is then given, followed by the implementation in Verilog-A and the verification simulations.

3.3.1.1 Stiffness Model

Linear beam stiffness mechanics include linear axial compression/tension, lateral bending and torsion about longitudinal axis. These basic effects have been widely studied [42][43][44] and modeled by a stiffness matrix $[k]$, derived from beam mechanics equations using energy methods [43][44].

In this section, derivation of the stiffness matrices for axial motion along x-axis and in-plane lateral bending motion about z-axis are given in detail. Stiffness matrices for out-of-plane bending motion about y-axis and torsional motion about x-axis are derived in a similar way. These matrices are then assembled to form the entire stiffness matrix for the 3D beam model.

Axial Compression/Tension

For a beam element under axial forces, F_{xa} and F_{xb}, as illustrated in Figure 3.4, the beam is simply modeled as a spring with linear stiffness. The axial displacement along x-direction, $u_x(x)$, is a linear function of x:

$$u_x(x) = a_{1x}x + a_{0x} \tag{3.1}$$
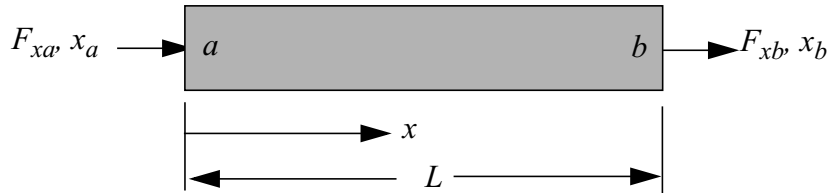
Figure 3.4: A beam element under axial forces

where a_{0x} and a_{1x} are two constants to be determined by the boundary conditions.

At $x = 0$,

$$u_x(x) = x_a = [a_{1x}x + a_{0x}]\big|_{x = 0} = a_{0x} \tag{3.2}$$

At $x = L$,

$$u_x(x) = x_b = [a_{1x}x + a_{0x}]\big|_{x = L} = a_{1x}L + a_{0x} = a_{1x}L + x_a \tag{3.3}$$

therefore

$$a_{1x} = \frac{x_b - x_a}{L} \tag{3.4}$$

where x_a and x_b are the node displacements in x-directions at beam ends a and b.

(3.1) is rewritten as:

$$u_x(x) = \frac{x_b - x_a}{L}x + x_a = f_{1x}(x)x_a + f_{2x}(x)x_b \tag{3.5}$$

where

$$f_{1x}(x) = 1 - \frac{x}{L} \tag{3.6}$$

and

$$f_{2x}(x) = \frac{x}{L} \tag{3.7}$$

$f_{1x}(x)$ and $f_{2x}(x)$ are called *shape functions*, describing the displacement along the beam associated with the node displacement x_a and x_b.

The axial strain is defined as [44]:

$$\varepsilon = \frac{\partial u}{\partial x} \tag{3.8}$$

and the axial force is

$$F = \sigma A = E\varepsilon A = EA\frac{\partial u_x}{\partial x} = EA(f_{1x}'(x)x_a + f_{2x}'(x)x_b) \tag{3.9}$$

where E is the Young's modulus and A is the beam cross section area. The axial strain energy is thus obtained as [44]:

$$U = \int_0^L \frac{F^2}{2EA} dx = \frac{EA}{2} \int_0^L (f_{1x}'(x)x_a + f_{2x}'(x)x_b)^2 dx \tag{3.10}$$

Applying Castigliano's theorem [44], we get

$$F_{ax} = \frac{\partial U}{\partial x_a} = \frac{2EA}{2} \int_0^L (f_{1x}'(x)x_a + f_{2x}'(x)x_b) f_{1x}'(x) dx$$

$$= \left(EA \int_0^L (f_{1x}'f_{1x}') dx \right) x_a + \left(EA \int_0^L (f_{2x}'f_{1x}') dx \right) x_b \tag{3.11}$$

and

$$F_{bx} = \frac{\partial U}{\partial x_b} = \frac{2EA}{2} \int_0^L (f_{1x}'(x)x_a + f_{2x}'(x)x_b) f_{2x}'(x) dx$$

$$= \left(EA \int_0^L (f_{1x}'f_{2x}') dx \right) x_a + \left(EA \int_0^L (f_{2x}'f_{2x}') dx \right) x_b \tag{3.12}$$

Rewrite (3.11) and (3.12) in matrix format, we get

$$\begin{bmatrix} F_{ax} \\ F_{bx} \end{bmatrix} = \begin{bmatrix} k_x \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} x_a \\ x_b \end{bmatrix} \tag{3.13}$$

where $[k_x]$ is called the *stiffness matrix* for the axial motion. The stiffness coefficients are

$$k_{ij,x} = EA \int_0^L f_{ix}'(x) f_{jx}'(x) dx, \; i=1,2 \text{ and } j=1,2. \tag{3.14}$$

Substitute the shape functions into (3.14), we get the stiffness matrix for axial motion:

$$\begin{bmatrix} k_x \end{bmatrix} = \begin{bmatrix} \dfrac{EA}{L} & -\dfrac{EA}{L} \\ -\dfrac{EA}{L} & \dfrac{EA}{L} \end{bmatrix} \tag{3.15}$$

Notices that $k_{ij,x} = k_{ji,x}$, thus the stiffness matrix $[k_x]$ is symmetric. The entire solving procedure for the stiffness matrix described above is an *energy method*, as the force is derived from the change of energy with displacement. Next, we will derive the stiffness matrix for the linear lateral beam bending following the same methodology.

In-Plane Lateral Bending

Unlike the linear relationship between axial forces and displacements, the in-plane lateral bending of a beam element with concentrated loads at beam ends is described by a 4th-order differential equation [42]:

$$\frac{d^4}{dx}u_y(x) = 0 \tag{3.16}$$

where $u_y(x)$ is the displacement in y-direction along the beam. Cubic polynomial beam bending shape functions are widely used in many structural matrix analysis methods [43][44][44][26][27] and also employed in NODAS [51]:

$$u_y(x) = a_{3y}x^3 + a_{2y}x^2 + a_{1y}x + a_{0y} \tag{3.17}$$

Applying boundary conditions to derive the expressions for the polynomial coefficients:

at $x = 0$,

$$u_y(0) = y_a = a_{0y} \tag{3.18}$$

and

$$\left.\frac{du_y}{dx}\right|_{x=0} = \phi_{za} = \left.[3a_{3y}x^2 + 2a_{2y}x + a_{1y}]\right|_{x=0} = a_{1y} \tag{3.19}$$

at $x = L$,

$$u_y(0) = y_b = a_{3y}L^3 + a_{2y}L^2 + a_{1y}L + a_{0y} \tag{3.20}$$

and

$$\left.\frac{du_y}{dx}\right|_{x=L} = \phi_{zb} = \left.[3a_{3y}x^2 + 2a_{2y}x + a_{1y}]\right|_{x=L} = 3a_{3y}L^2 + 2a_{2y}L + a_{1y} \tag{3.21}$$

y_a and y_b are the translational displacements at node a and b, and ϕ_{za} and ϕ_{zb} are the angular displacements at node a and b, as illustrated in Figure 3.5. Solving (3.18)-(3.21) simultaneously, we get
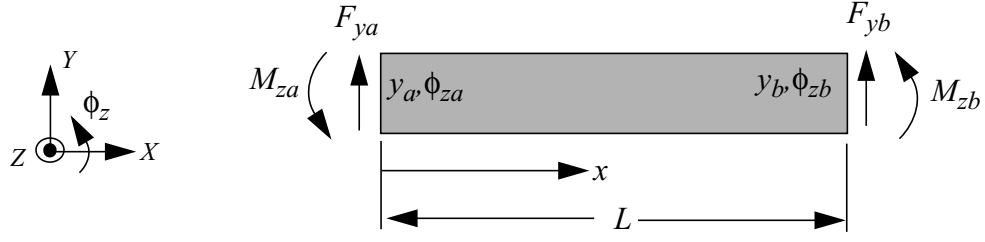
Figure 3.5: Definitions of lumped in-plane lateral forces, moments and displacements of a beam element

$$a_{0y} = y_a \tag{3.22}$$

$$a_{1y} = \phi_{za} \tag{3.23}$$

$$a_{2y} = -\frac{3}{L^2}y_a - \frac{2}{L}\phi_{za} + \frac{3}{L^2}y_b - \frac{1}{L}\phi_{zb} \tag{3.24}$$

and

$$a_{3y} = \frac{2}{L^3}y_a + \frac{1}{L^2}\phi_{za} - \frac{2}{L^3}y_b + \frac{1}{L^2}\phi_{zb} \tag{3.25}$$

Substituting (3.22)-(3.25) into (3.17) and rearranging the terms, we get

$$u_y(x) = f_{1y}(x)y_a + f_{2y}(x)\phi_{za} + f_{3y}(x)y_b + f_{4y}(x)\phi_{zb} \tag{3.26}$$

where

$$f_{1y}(x) = 1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3 \tag{3.27}$$

$$f_{2y}(x) = \left[\frac{x}{L} - 2\left(\frac{x}{L}\right)^2 + \left(\frac{x}{L}\right)^3\right]L \tag{3.28}$$

$$f_{3y}(x) = 3\left(\frac{x}{L}\right)^2 - 2\left(\frac{x}{L}\right)^3 \tag{3.29}$$

$$f_{4y}(x) = \left[-\left(\frac{x}{L}\right)^2 + \left(\frac{x}{L}\right)^3\right]L \tag{3.30}$$

$f_{iy}(x)$, i =1 to 4, are known as the lateral beam bending shape functions.

The bending strain energy is [44]:

$$U = \int_0^L \frac{M^2}{2EI_z} dx = \int_0^L \frac{\left(EI_z\left(\frac{d^2}{dx^2}u_y\right)\right)^2}{2EI_z} dx = \frac{EI_z}{2}\int_0^L \left(\frac{d^2}{dx^2}u_y\right)^2 dx \tag{3.31}$$

$$= \frac{EI_z}{2}\int_0^L (f_{1y}''(x)y_a + f_{2y}''(x)\phi_{za} + f_{3y}''(x)y_b + f_{4y}''(x)\phi_{zb})^2 dx$$

where I_z is the moment of inertia about z-axis.

Applying Castigliano's theorem, the bending forces in y-direction at node a is obtained by taking partial differentiation of the strain energy U with respect to y_a:

$$\begin{aligned} F_{ya} = \frac{\partial U}{\partial y_a} &= \frac{2EI_z}{2}\int_0^L (f_{1y}''(x)y_a + f_{2y}''(x)\phi_{za} + f_{3y}''(x)y_b + f_{4y}''(x)\phi_{zb})f_{1y}''(x)dx \\ &= \left(EI_z\int_0^L (f_{1y}''f_{1y}'')dx\right)y_a + \left(EI_z\int_0^L (f_{1y}''f_{2y}'')dx\right)\phi_{za} \\ &\quad + \left(EI_z\int_0^L (f_{1y}''f_{3y}'')dx\right)y_b + \left(EI_z\int_0^L (f_{1y}''f_{4y}'')dx\right)\phi_{zb} \\ &= k_{11}y_a + k_{12}\phi_{za} + k_{13}y_b + k_{14}\phi_{zb} \end{aligned} \tag{3.32}$$

where

$$k_{ij,y} = EI_z\int_0^L f_{iy}''(x)f_{jy}''(x)dx , \ i = 1,4 \text{ and } j = 1,4 \tag{3.33}$$

The bending force in y-direction at node b, F_{yb}, and the bending moments about z-axis at node a and b, M_{za} and M_{zb}, are derived similarly:

$$M_{za} = \frac{\partial U}{\partial \phi_{za}} = k_{21}y_a + k_{22}\phi_{za} + k_{23}y_b + k_{24}\phi_{zb} \tag{3.34}$$

$$F_{yb} = \frac{\partial U}{\partial y_b} = k_{31}y_a + k_{32}\phi_{za} + k_{33}y_b + k_{34}\phi_{zb} \tag{3.35}$$

$$M_{zb} = \frac{\partial U}{\partial \phi_{zb}} = k_{41}y_a + k_{42}\phi_{za} + k_{43}y_b + k_{44}\phi_{zb} \tag{3.36}$$

Substituting (3.27)-(3.30) into (3.32) and (3.34)-(3.36) leads to the stiffness matrix for in-plane lateral bending:

$$
\begin{bmatrix} F_{ya} \\ M_{za} \\ F_{yb} \\ M_{zb} \end{bmatrix} = \begin{bmatrix} k_y \end{bmatrix} \begin{bmatrix} y_a \\ \phi_{za} \\ y_b \\ \phi_{zb} \end{bmatrix} = \frac{EI_z}{L} \begin{bmatrix} \dfrac{12}{L^2} & \dfrac{6}{L} & -\dfrac{12}{L^2} & \dfrac{6}{L} \\[2mm] \dfrac{6}{L} & 4 & -\dfrac{6}{L} & 2 \\[2mm] -\dfrac{12}{L^2} & -\dfrac{6}{L} & \dfrac{12}{L^2} & -\dfrac{6}{L} \\[2mm] \dfrac{6}{L} & 2 & -\dfrac{6}{L} & 4 \end{bmatrix} \begin{bmatrix} y_a \\ \phi_{za} \\ y_b \\ \phi_{zb} \end{bmatrix} \tag{3.37}
$$

The stiffness matrix for lateral bending given in (3.37) is symmetric. Notice that the linear beam stiffness matrix as derived above has two assumptions:

1. The axial motion is independent of the lateral bending, thus the forces in *x*-direction have no effect on the bending in *y*-direction. They simply cause displacement along *x*-axis.

2. The lateral displacements are small compared to the beam length, *L*. Normally, a displacement less than 15% of the beam length is considered to be a small displacement. Figure 3.6 shows the static simulation of a cantilever beam. Results from NODAS simulation using the linear beam model are compared to results given by simulation in ABAQUS. *y/L* at the free end is plotted vs. the normalized force, $F_y L^2 EI$. We see that for this case, the error in y-displacement is within 5% compared to ABAQUS when the *y*-
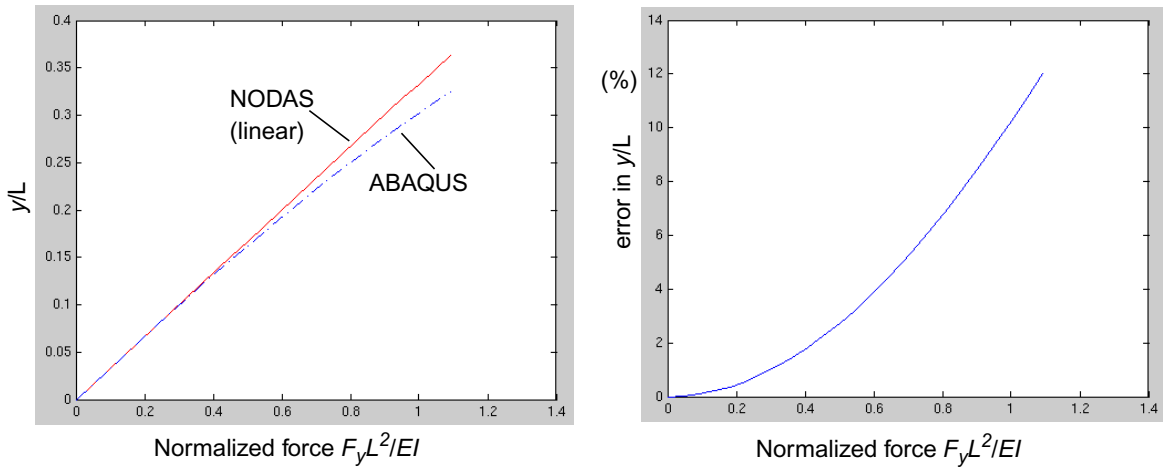
Figure 3.6: Simulation of a cantilever beam with F_y applied to the free end ($L = 100$ μm, $w = 2$ μm, $t = 2$ μm, $E = 165$ GPa) (a) NODAS (with linear beam model) vs. ABAQUS (b) percentage error in *y*-displacement

displacement is within 15%~20% of the beam length. Within this deflection range, the boundary conditions used in the derivation of cubic polynomial coefficients, $\phi \cong dy/dx$, as in (3.19) and (3.21), is valid to ensure that the consequent cubic shape functions are accurate. Later, we will see that when the deflection gets larger, the cubic shape functions are no longer able to capture the shape of the bending beam accurately. The beam starts to behave nonlinearly and therefore the stiffness matrix has to be modified to included nonlinear terms. Later sections about nonlinear beam model will also shown that for some cases such as a fixed-fixed beam, the beam starts to behave nonlinearly at a much smaller lateral displacement than 15% of the beam length, due to large axial stress stiffening. The linear beam model need to be extended to include the nonlinearity in both cases.

Out-of-Plane Lateral Bending

The out-of-plane lateral bending in *z*-direction and the rotation about *y*-axis due to bending follow the same physical principle as the in-plane deflection.

For in-plane deflection, the rotation is about *z*-axis, thus the moment of inertia about *z*-axis, I_z, is used. For out-of-plane deflection, the rotation is about *y*-axis, therefore the moment of inertia about *y*-axis, I_y, must be used instead. For a beam element along *x*-axis, with length of *L*, width of *w* and thickness of *t*, as shown in Figure 3.3,

$$I_z \ = \ \frac{1}{12} t w^3 \tag{3.38}$$

and

$$I_y \ = \ \frac{1}{12} t^3 w \tag{3.39}$$

Directions of the coupled displacements caused by the bending forces and moments are different for out-of-plane bending. Figure 3.7 shows the definitions of lumped forces, moments and displacements for the out-of-plane lateral beam bending. In this way, sign conventions consistent with the in-plane motion are formed. Notice that for in-plane
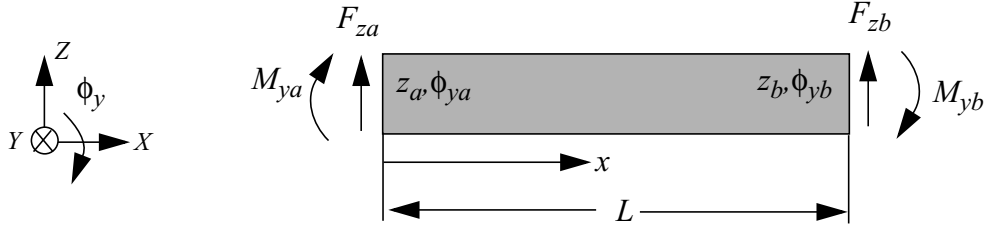
Figure 3.7: Definitions of lumped out-of-plane lateral forces, moments and displacements of a beam element

deflection, a positive bending moment M_{zb} causes positive displacement in y-direction, as shown in Figure 3.5, while for out-of-plane deflection, a positive bending moment M_{yb} causes negative displacement in z-direction, as shown in Figure 3.7. Similarly, for in-plane deflection, a positive bending force F_{zb} causes positive rotation about z-axis, while for out-of-plane deflection, a positive bending force F_{zb} causes negative rotation about y-axis. Therefore, the signs of certain stiffness coefficients are different in these two cases. The stiffness matrix for out-of-plane deflection is:

$$
\begin{bmatrix} F_{za} \\ M_{ya} \\ F_{zb} \\ M_{yb} \end{bmatrix} = \frac{EI_y}{L} \begin{bmatrix} \dfrac{12}{L^2} & -\dfrac{6}{L} & -\dfrac{12}{L^2} & -\dfrac{6}{L} \\ -\dfrac{6}{L} & 4 & \dfrac{6}{L} & 2 \\ -\dfrac{12}{L^2} & \dfrac{6}{L} & \dfrac{12}{L^2} & \dfrac{6}{L} \\ -\dfrac{6}{L} & 2 & \dfrac{6}{L} & 4 \end{bmatrix} \begin{bmatrix} z_a \\ \phi_{ya} \\ z_b \\ \phi_{yb} \end{bmatrix}
\tag{3.40}
$$

Torsion about the Longitudinal Direction

Torsion about the longitudinal direction, as shown in Figure 3.8, is an important physical effect which needs to be captured by the beam model. Similar to the linear relation between axial forces and axial displacements, the relation between the torsional moments and the angular displacements about x-axis is modeled as a linear torsional spring. The torsional spring constant is
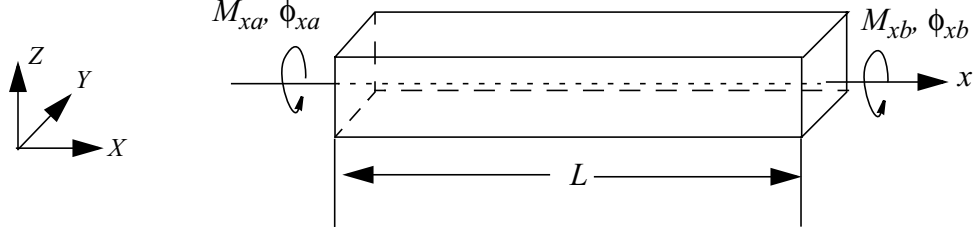
Figure 3.8: Beam torsion along the longitudinal direction

$$k_{\phi x} = \frac{GJ_t}{L} \tag{3.41}$$

G is the *shear modulus* defined as

$$G = \frac{E}{2(1+\nu)} \tag{3.42}$$

where ν is the Poisson's ratio [42]. J_t is the *torsion constant*. For a beam of rectangular cross-section, J_t is given by [52][83]

$$J_t = \frac{1}{3}t^3 w \left(1 - \frac{192}{\pi^5} \frac{t}{w} \sum_{i=1, iodd}^{\infty} \frac{1}{i^5} \tanh\left(\frac{i\pi w}{2t}\right) \right) \tag{3.43}$$

where $t < w$. If $t > w$, then the roles of t and w are switched in (3.43). Notice that in the implementation in Verilog-A, the summation in (3.43) has to be truncated to a finite number of i. In NODAS, the first 10 summation terms are taken, *i.e.*, $i=1$ to 19. Since the summand in (3.43) decreases as i^{-5}, for $i = 21$, the relative contribution is down to $21^{-5} = 2.45e{-}7$. The truncation error is small enough to be neglected.

Similar to the stiffness matrix for axial motion, the stiffness matrix for torsional motion is:

$$\begin{bmatrix} M_{xa} \\ M_{xb} \end{bmatrix} = \begin{bmatrix} k \end{bmatrix} \begin{bmatrix} \phi \end{bmatrix} = \begin{bmatrix} \dfrac{GJ_t}{L} & -\dfrac{GJ_t}{L} \\ -\dfrac{GJ_t}{L} & \dfrac{GJ_t}{L} \end{bmatrix} \begin{bmatrix} \phi_{xa} \\ \phi_{xb} \end{bmatrix} \tag{3.44}$$

It is again a symmetric matrix. The torsional motion is independent of the axial and lateral motions as well.

Stiffness Matrix for 12-DOF Beam Element

Combining the stiffness matrices for the axial, lateral and torsional motions, we get the entire stiffness matrix for the 12-DOF linear beam element [43]:

$$
\begin{bmatrix} F_{xa} \\ F_{ya} \\ F_{za} \\ M_{xa} \\ M_{ya} \\ M_{za} \\ F_{xb} \\ F_{yb} \\ F_{zb} \\ M_{xb} \\ M_{yb} \\ M_{zb} \end{bmatrix}_k = \begin{bmatrix} k \end{bmatrix} \begin{bmatrix} x \end{bmatrix} =
$$

$$
\begin{bmatrix}
\frac{Etw}{L} & & & & & & -\frac{Etw}{L} & & & & & \\
 & \frac{12EI_z}{L^3} & & & & \frac{6EI_z}{L^2} & & -\frac{12EI_z}{L^3} & & & & \frac{6EI_z}{L^2} \\
 & & \frac{12EI_y}{L^3} & & -\frac{6EI_y}{L^2} & & & & -\frac{12EI_y}{L^3} & & -\frac{6EI_y}{L^2} & \\
 & & & \frac{GJ_t}{L} & & & & & & -\frac{GJ_t}{L} & & \\
 & & -\frac{6EI_y}{L^2} & & \frac{4EI_y}{L} & & & & \frac{6EI_y}{L^2} & & \frac{2EI_y}{L} & \\
 & \frac{6EI_z}{L^2} & & & & \frac{4EI_z}{L} & & -\frac{6EI_z}{L^2} & & & & \frac{2EI_z}{L} \\
-\frac{Etw}{L} & & & & & & \frac{Etw}{L} & & & & & \\
 & -\frac{12EI_z}{L^3} & & & & -\frac{6EI_z}{L^2} & & \frac{12EI_z}{L^3} & & & & -\frac{6EI_z}{L^2} \\
 & & -\frac{12EI_y}{L^3} & & \frac{6EI_y}{L^2} & & & & \frac{12EI_y}{L^3} & & \frac{6EI_y}{L^2} & \\
 & & & -\frac{GJ_t}{L} & & & & & & \frac{GJ_t}{L} & & \\
 & & -\frac{6EI_y}{L^2} & & \frac{2EI_y}{L} & & & & \frac{6EI_y}{L^2} & & \frac{4EI_y}{L} & \\
 & \frac{6EI_z}{L^2} & & & & \frac{2EI_z}{L} & & -\frac{6EI_z}{L^2} & & & & \frac{4EI_z}{L}
\end{bmatrix}
\begin{bmatrix} x_a \\ y_a \\ z_a \\ \phi_{xa} \\ \phi_{ya} \\ \phi_{za} \\ x_b \\ y_b \\ z_b \\ \phi_{xb} \\ \phi_{yb} \\ \phi_{zb} \end{bmatrix}
\quad (3.45)
$$

3.3.1.2 Inertia Model

In addition to the static beam stiffness mechanics, the beam element is also subject to dynamic effects due to inertia and damping.

Beam inertia is modeled by a mass matrix $[m]$, derived by assuming the static mode shapes are effective for dynamic motion. The dynamic motion equation for the beam element is obtained from Lagrange's equation [53]:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_{nr} \tag{3.46}$$

where q_i is the generalized displacements, $L = T - V$ is the Lagrangian where T is the kinetic energy and V is the potential energy, and Q_{nr} is the non-conservative force. For the beam element, q_i represents the ith degree-of-freedom (the translational or rotational displacements at node a and node b). The potential energy V equals the strain energy U, which is the sum of strain energy for axial, lateral and torsional motions:

$$U = \int_0^L \left(\frac{F_x^2}{2EA} + \frac{M_z^2}{2EI_z} + \frac{M_y^2}{2EI_y} + \frac{M_x^2}{2GJ}\right)dx \tag{3.47}$$

The kinetic energy T is

$$T = \sum_i \frac{\rho A}{2}\int_0^L (\dot{u}_i(\xi))^2 d\xi \tag{3.48}$$

where ρ is the mass density, A is the cross-section area, and $u_i(\xi)$ are the axial, lateral and torsional displacements along the beam. For example, consider a simple mass of $m = \rho AL$ with a uniform velocity of $\dot{y}$, (3.48) reduces to the familiar kinetic energy equation:

$$T = \frac{\rho A}{2}\int_0^L (\dot{u}(\xi))^2 d\xi = \frac{\rho A}{2}L\dot{y}^2 = \frac{1}{2}\rho AL\dot{y}^2 = \frac{1}{2}m\dot{y}^2 \tag{3.49}$$

Q_{nr} is the sum of non-conservative forces

$$Q_{nr} = F_{ext} - F_B \tag{3.50}$$

where F_{ext} is the external force and F_B is the damping force.

We see that the strain energy U is a function of the displacements q_i rather than the velocity $\dot{q}_i$, and the kinetic energy T is a function of the velocity $\dot{q}_i$ rather than the displacement q_i, the Lagrange's equation in (3.46) hence reduces to:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) + \frac{\partial U}{\partial q_i} = F_{ext} - F_B \tag{3.51}$$

Defining

$$F_m = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right)$$

(3.52)

as the inertial force and

$$F_k = \frac{\partial U}{\partial q_i}$$

(3.53)

which is a restatement of Castigliano's theorem, as the spring force, the dynamic equation for the beam is then:

$$F_{ext} = F_k + F_m + F_B$$

(3.54)

Substituting the axial displacement (3.5) into the kinetic energy (3.48):

$$T = \frac{\rho A}{2}\int_0^L (\dot{u}(x))^2 dx = \frac{\rho A}{2}\int_0^L (f_1(x)\dot{x}_a + f_2(x)\dot{x}_b)^2 dx$$

(3.55)

then substituting (3.55) into (3.52):

$$
\begin{aligned}
F_{xa,m} &= \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{x}_a}\right) = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{x}_a}\left(\frac{\rho A}{2}\int_0^L (f_1(x)\dot{x}_a + f_2(x)\dot{x}_b)^2 dx\right)\right) \\
&= \rho A\left(\int_0^L (f_1(x)\ddot{x}_a + f_2(x)\ddot{x}_b)f_1(x)dx\right) \\
&= \rho A\left(\int_0^L f_1(x)f_1(x)dx\right)\ddot{x}_a + \rho A\left(\int_0^L f_1(x)f_2(x)dx\right)\ddot{x}_b \\
&= m_{11}\ddot{x}_a + m_{12}\ddot{x}_b
\end{aligned}
$$

(3.56)

Similarly,

$$
\begin{aligned}
F_{xb,m} &= \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{x}_b}\right) = \rho A\left(\int_0^L f_2(x)f_1(x)dx\right)\ddot{x}_a + \rho A\left(\int_0^L f_2(x)f_2(x)dx\right)\ddot{x}_b \\
&= m_{21}\ddot{x}_a + m_{22}\ddot{x}_b
\end{aligned}
$$

(3.57)

In matrix format,

$$\begin{bmatrix} F_{xa} \\ F_{xb} \end{bmatrix}_m = \begin{bmatrix} m \end{bmatrix} \begin{bmatrix} \ddot{x} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} \ddot{x}_a \\ \ddot{x}_b \end{bmatrix} \qquad (3.58)$$

$[m]$ is called the *mass matrix*. The matrix coefficients follow the general format:

$$m_{ij} = \rho A \int_0^L f_i(x) f_j(x) dx \qquad (3.59)$$

Effective mass matrix terms for the lateral bending and torsional motions are derived following the same methodology. The entire effective mass matrix for the 12-DOF linear beam element is given below [43]. Next, we will derive the damping model for the beam element.

$$\begin{bmatrix} F_{xa} \\ F_{ya} \\ F_{za} \\ M_{xa} \\ M_{ya} \\ M_{za} \\ F_{xb} \\ F_{yb} \\ F_{zb} \\ M_{xb} \\ M_{yb} \\ M_{zb} \end{bmatrix}_m = \begin{bmatrix} m \end{bmatrix} \begin{bmatrix} \ddot{x} \end{bmatrix} = \rho twl \begin{bmatrix} \frac{1}{3} & & & & & & \frac{1}{6} & & & & & \\ & \frac{13}{35} & & & & \frac{11l}{210} & & \frac{9}{70} & & & -\frac{13l}{420} & \\ & & \frac{13}{35} & & -\frac{11l}{210} & & & & \frac{9}{70} & & & \frac{13l}{420} \\ & & & \frac{I_y+I_z}{3tw} & & & & & & \frac{I_y+I_z}{6tw} & & \\ & & -\frac{11l}{210} & & \frac{l^2}{105} & & & & -\frac{13l}{420} & & & -\frac{l^2}{140} \\ & \frac{11l}{210} & & & & \frac{l^2}{105} & & \frac{13l}{420} & & & & -\frac{l^2}{140} \\ \frac{1}{6} & & & & & & \frac{1}{3} & & & & & \\ & \frac{9}{70} & & & & \frac{13l}{420} & & \frac{13}{35} & & & -\frac{11l}{210} & \\ & & \frac{9}{70} & & -\frac{13l}{420} & & & & \frac{13}{35} & & & \frac{11l}{210} \\ & & & \frac{I_y+I_z}{6tw} & & & & & & \frac{I_y+I_z}{3tw} & & \\ & & \frac{13l}{420} & & -\frac{l^2}{140} & & & & \frac{11l}{210} & & & \frac{l^2}{105} \\ & -\frac{13l}{420} & & & & -\frac{l^2}{140} & & -\frac{11l}{210} & & & & \frac{l^2}{105} \end{bmatrix} \begin{bmatrix} \ddot{x}_a \\ \ddot{y}_a \\ \ddot{z}_a \\ \ddot{\phi}_{xa} \\ \ddot{\phi}_{ya} \\ \ddot{\phi}_{za} \\ \ddot{x}_b \\ \ddot{y}_b \\ \ddot{z}_b \\ \ddot{\phi}_{xb} \\ \ddot{\phi}_{yb} \\ \ddot{\phi}_{zb} \end{bmatrix} \qquad (3.60)$$

3.3.1.3 Damping Model

Like the inertial forces, the damping forces are distributed along the beam rather than concentrated at the beam ends, as shown in Figure 3.9(a). Therefore, the damping forces are lumped to the beam ends based on the bending shape functions [54]. The lateral bending
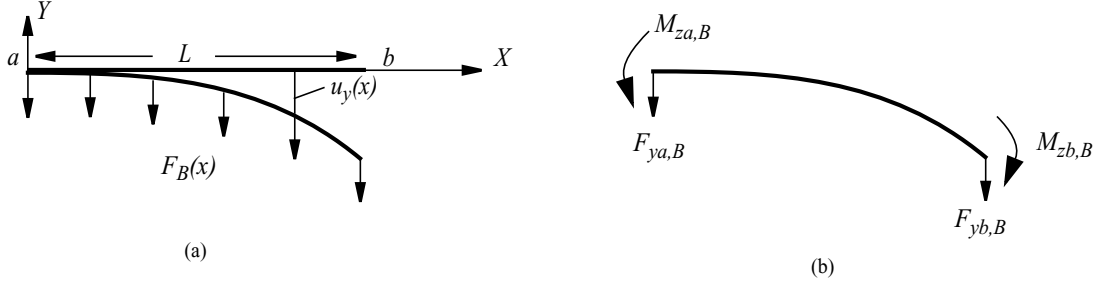
Figure 3.9: (a) Distributed damping forces for a beam element in-plane bending (b) lumped equivalent forces and bending moments

displacement is:

$$u_y(x) = f_{1y}(x)y_a + f_{2y}(x)\phi_{za} + f_{3y}(x)y_b + f_{4y}(x)\phi_{zb} \tag{3.61}$$

the distributed damping force is lumped into equivalent concentrated forces and bending moments at node a and b, as shown in Figure 3.9 (b):

$$F_{ya,\,B} = \int_0^L f_{1y}(x)F_B dx \tag{3.62}$$

$$M_{za,\,B} = \int_0^L f_{2y}(x)F_B dx \tag{3.63}$$

$$F_{yb,\,B} = \int_0^L f_{3y}(x)F_B dx \tag{3.64}$$

$$M_{zb,\,B} = \int_0^L f_{4y}(x)F_B dx \tag{3.65}$$

where F_B is the damping force per unit length along the beam.

Viscous air damping involved in MEMS devices include Couette flow [55], Stokes flow [55] and squeeze film damping [56][57]. In this thesis, only Couette air damping is included in the beam model. When the beam bends in y-direction, the Couette air damping force per unit length between the bottom surface of the beam and the substrate is:

$$F_B = \frac{\mu w}{\Delta}\dot{u}_y(x) \tag{3.66}$$

where μ is the viscosity of air, w is the width of the beam, and Δ is the air gap between the bottom surface of the beam and the substrate. Applying (3.66) to (3.62), we get:

$$F_{ya,\,B} = \int_0^L f_1(x)\tilde{F}_B dx = \int_0^L f_1(x)\frac{\mu w}{\Delta}\dot{u}_y(x)dx$$

$$= \frac{\mu w}{\Delta}\int_0^L f_1(x)(f_1(x)\dot{y}_a + f_2(x)\dot{\phi}_{za} + f_3(x)\dot{y}_b + f_4(x)\dot{\phi}_{zb})dx$$

$$= \left(\frac{\mu w}{\Delta}\int_0^L f_1(x)f_1(x)dx\right)\dot{y}_a + \left(\frac{\mu w}{\Delta}\int_0^L f_1(x)f_2(x)dx\right)\dot{\phi}_{za} \qquad (3.67)$$

$$+ \left(\frac{\mu w}{\Delta}\int_0^L f_1(x)f_3(x)dx\right)\dot{y}_b + \left(\frac{\mu w}{\Delta}\int_0^L f_1(x)f_4(x)dx\right)\dot{\phi}_{zb}$$

$$= B_{11}\dot{y}_a + B_{12}\dot{\phi}_{za} + B_{13}\dot{y}_b + B_{14}\dot{\phi}_{zb}$$

Following the same methodology, other lumped concentrated forces and moments are derived. Written in matrix format, we get:

$$\begin{bmatrix} F_{ya} \\ M_{za} \\ F_{yb} \\ M_{zb} \end{bmatrix}_B = [B][\dot{x}] = \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} \begin{bmatrix} \dot{y}_a \\ \dot{\phi}_{za} \\ \dot{y}_b \\ \dot{\phi}_{zb} \end{bmatrix} \qquad (3.68)$$

[B] is called the *damping matrix*. The matrix coefficients follow the general format:

$$B_{ij} = \frac{\mu w}{\Delta}\int_0^L f_i(x)f_j(x)dx \qquad (3.69)$$

The damping matrix is also symmetric like the stiffness and mass matrices.

Comparing (3.69) to (3.59), we notice that B_{ij} follows the same format as m_{ij} except that the mass factor ρA is replaced by the damping factor $\mu w/\Delta$. The damping matrix therefore looks very similar to the mass matrix. The entire damping matrix for the 12-DOF beam element is:

$$
\begin{bmatrix} F_{xa} \\ F_{ya} \\ F_{za} \\ M_{xa} \\ M_{ya} \\ M_{za} \\ F_{xb} \\ F_{yb} \\ F_{zb} \\ M_{xb} \\ M_{yb} \\ M_{zb} \end{bmatrix}_B = [B][\dot{x}] =
$$

$$
\left[\begin{smallmatrix}
B_x\frac{1}{3} & & & & & & B_x\frac{1}{6} & & & & & \\
& B_y\frac{13}{35} & & & & B_y\frac{11l}{210} & & B_y\frac{9}{70} & & & & -B_y\frac{13l}{420} \\
& & B_z\frac{13}{35} & & -B_z\frac{11l}{210} & & & & B_z\frac{9}{70} & & B_z\frac{13l}{420} & \\
& & & B_x\frac{I_y+I_z}{3tw} & & & & & & B_x\frac{I_y+I_z}{6tw} & & \\
& & -B_z\frac{11l}{210} & & B_z\frac{l^2}{105} & & & & -B_z\frac{13l}{420} & & -B_z\frac{l^2}{140} & \\
& B_y\frac{11l}{210} & & & & B_y\frac{l^2}{105} & & B_y\frac{13l}{420} & & & & -B_y\frac{l^2}{140} \\
B_x\frac{1}{6} & & & & & & B_x\frac{1}{3} & & & & & \\
& B_y\frac{9}{70} & & & & B_y\frac{13l}{420} & & B_y\frac{13}{35} & & & & -B_y\frac{11l}{210} \\
& & \frac{9}{70} & & -\frac{13l}{420} & & & & \frac{13}{35} & & \frac{11l}{210} & \\
& & & B_x\frac{I_y+I_z}{6tw} & & & & & & B_x\frac{I_y+I_z}{3tw} & & \\
& & B_z\frac{13l}{420} & & -B_z\frac{l^2}{140} & & & & B_z\frac{11l}{210} & & B_z\frac{l^2}{105} & \\
& -B_y\frac{13l}{420} & & & & -B_y\frac{l^2}{140} & & -B_y\frac{11l}{210} & & & & B_y\frac{l^2}{105}
\end{smallmatrix}\right]
\begin{bmatrix} \dot{x}_a \\ \dot{y}_a \\ \dot{z}_a \\ \dot{\phi}_{xa} \\ \dot{\phi}_{ya} \\ \dot{\phi}_{za} \\ \dot{x}_b \\ \dot{y}_b \\ \dot{z}_b \\ \dot{\phi}_{xb} \\ \dot{\phi}_{yb} \\ \dot{\phi}_{zb} \end{bmatrix}
\tag{3.70}
$$

where B_x is the damping coefficient for translation in *x*-direction and torsion about *x*-axis, B_y is the damping coefficient for translation in *y*-direction and rotation about z-axis, and B_z is the damping coefficient for translation in *z*-direction and rotation about *y*-axis. They are defined as:

$$
B_x = \frac{\mu L(w + \delta)}{\Delta} \tag{3.71}
$$

$$
B_y = \frac{\mu w(L + \delta)}{\Delta} \tag{3.72}
$$

$$
B_z = \frac{\mu t(L + \delta)}{\Delta} \tag{3.73}
$$

where δ is the edge factor used to take into account the edge and finite size effects, which is 4 μm for damping surfaces of rectangular shape [55][58]. In (3.71)-(3.73), a common air-gap, Δ, is used for damping in every direction, by assuming that the damping in every direction has the same damping depth. In fact, if the air damping happens between the bottom surface of the beam and the substrate, then the air gap between the beam and the substrate should be used. If the damping happens between side faces of the beam element and the adjacent element, then the distance between these elements should be used instead.

However, the distances between adjacent elements are not constants thus can not be hardcoded in the process file. They must either be specified manually by the user or automatically by the codes based on layout (for initial element distances on the layout), or be calculated by gap models based on displacements of both elements (for distances between displaced elements).

The similarity of the damping matrix to the mass matrix raises a question: since the inertial forces are also distributed forces along the beam, can the mass matrix be derived in the same methodology as the damping matrix? The answer is yes. The inertia force per unit length along the beam, F_m, is [54]:

$$F_m = \rho A \ddot{u}_y(x) \tag{3.74}$$

and the equivalent lumped inertia forces and moments are:

$$F_{ya,m} = \int_0^L f_{1y}(x) F_m dx \tag{3.75}$$

$$M_{za,m} = \int_0^L f_{2y}(x) F_m dx \tag{3.76}$$

$$F_{yb,m} = \int_0^L f_{3y}(x) F_m dx \tag{3.77}$$

$$M_{zb,m} = \int_0^L f_{4y}(x) F_m dx \tag{3.78}$$

Substituting the shape functions into (3.75)-(3.78), we get the mass matrix, which is exactly the same as the mass matrix derived from the energy method. This is not a coincidence. The results from these two methods are the same because both energy method and the lumping equations given by (3.75)-(3.78) are derived from the same physical principle - the virtual work principle [44] [54].

Summary of Linear Beam Mechanics

Based on the stiffness matrix [k], the mass matrix [m] and damping matrix [B], the relation between the force/moment vector and the displacement vector is established as:

$$\left[F_{beam}\right] = \left[m\right]\left[\ddot{x}\right] + \left[B\right]\left[\dot{x}\right] + \left[k\right]\left[x\right] \tag{3.79}$$

where $[F_{beam}] = [F_{xa}\,F_{ya}\,F_{za}\,M_{xa}\,M_{ya}\,M_{za}\,F_{xb}\,F_{yb}\,F_{zb}\,M_{xb}\,M_{yb}\,M_{zb}]^T$, $[x] = [x_a\,y_a\,z_a\,\phi_{xa}\,\phi_{ya}$ $\phi_{za}\,x_b\,y_b\,z_b\,\phi_{xb}\,\phi_{yb}\,\phi_{zb}]^T$, and node *a* and *b* are the nodes at beam ends.

These matrices cover the basic physical effects involved in linear beam mechanics. To include more physical effects, such as nonlinear beam bending, trapezoidal beam cross-section or squeeze-film damping, current matrices need to be modified or new matrices have to be created.

3.3.1.4 Coordinate Transformation

In previous sections, the beam mechanics are all derived with the assumption that the beam element has *x*-axis as the longitudinal axis, and the beam cross-section is perpendicular to *x*-axis, as shown in Figure 3.3. In fact, the beam elements can have various orientations. These elements must be able to communicate with each other. In electrical circuits, electrical components communicate through electrical wire connections, while in MEMS, the communication between mechanical elements not only needs the wire connection, but also needs the geometric orientations and positions of each element. Thus, frames of reference are defined for proper communication. As shown in Figure 3.10, the chip frame is where the element layout position is specified. The across and through
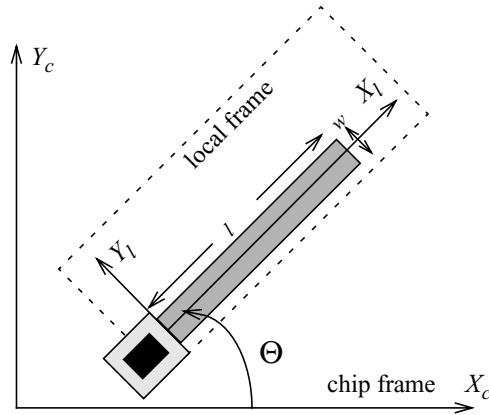
Figure 3.10: Frames of reference for the beam element

variables at the connection terminals represent the displacements, forces and moments of the elements in the chip frame. They are the variables used to form the system matrix. The system matrix is formed and solved in the chip frame, following the Kirchhoffian network laws specified by the topology. Local frames are attached to each element. They are indispensable for correct modeling of mechanics. The element communication is accomplished through the coordinate transformations between the frames of reference.

The coordinate transformation matrix is defined based on the *Euler angles* [59][60], as illustrated in Figure 3.11. The transformation from a 3D coordinate system *x-y-z* to another 3D coordinate system *x″*-y″-z″ can be implemented in various ways, with different sequences of rotation, such as the *x*-convention and the xyz-convention [61]. The *xyz*-convention, also called pitch-roll-yaw convention, is used in NODAS. The transformation starts from a rotation of angle γ about the original *z*-axis, leading to the first immediate system *x′*-y′-z, then followed by a rotation of angle β about the *y′*-axis, leading to the second immediate system *x″*-y′-z′, finally a rotation of angle α about the *x″*-axis, leading to the destination coordinate system *x″*-y″-z″. Angle α, β and γ are called Euler angles. In matrix format, the transformation is represented as a 3x3 matrix called the *Coordinate Transformation Matrix*, [*R*]:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \\ l_3 & m_3 & n_3 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} R \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \tag{3.80}
$$
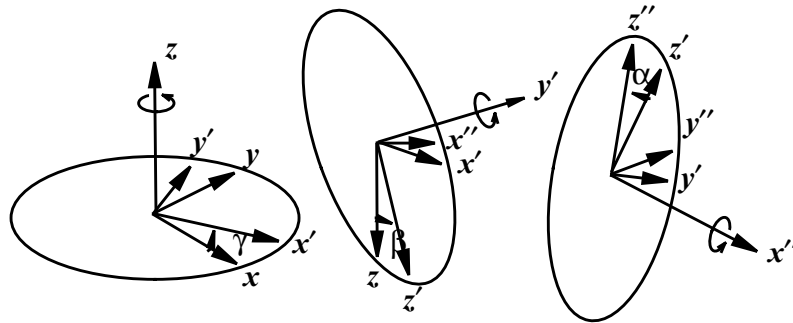
Figure 3.11: Coordinate transformation based on Euler angles

As shown in Figure 3.11, the coordinate transformation is fulfilled by three consecutive rotation steps, therefore, the coordinate transformation matrix $[R]$ is obtained by the multiplication of three transformation matrices each representing a specific rotation step:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & & \\ & \cos\alpha & \sin\alpha \\ & -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & & \sin\beta \\ & 1 & \\ -\sin\beta & & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & \sin\gamma & \\ -\sin\gamma & \cos\gamma & \\ & & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}
$$

$$
= \begin{bmatrix} \cos\beta\cos\gamma & \cos\beta\sin\gamma & -\sin\beta \\ \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\cos\beta \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\cos\beta \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}
$$

(3.81)

The transformation from coordinate system x''-y''-z'' to coordinate system x-y-z is accordingly represented by the inverse matrix of $[R]$:

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

(3.82)

Notice that the matrix $[R]$ given in (3.81) is dependent on the rotation sequence of *z-y-x*. If the order of rotation is changed, the transformation matrix must be rederived.

For an in-plane beam element on the chip with an angle of γ relative to the *x*-axis of the chip frame, β and α are all zeros, the transformation matrix reduces to:

$$
\begin{bmatrix} R \end{bmatrix} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

(3.83)

3.3.1.5 *Algorithmic Structure of the Linear Beam Model*

Figure 3.12 shows the algorithmic structure of the linear beam model. The model first measures the displacements in the chip frame, then transforms them into the local frame of the specific beam instance on the chip. The spring forces, the damping forces and the inertia forces are calculated by the beam model in the local frame and added up to give the total force in the local frame. The total force is then transformed back to the chip frame to form the system matrix. The system matrix is formed based on the through and across variables

$$[x]_{chip}$$

$$[R] \rightarrow [x]_{local} \rightarrow [F_k]_{local} = [k][x]_{local} \qquad\qquad \textit{Beam model}$$

$$[\dot{x}]_{local} \rightarrow [F_b]_{local} = [B][\dot{x}]_{local} \longrightarrow (\Sigma) \rightarrow [F]_{local}$$

$$[\ddot{x}]_{local} \rightarrow [F_m]_{local} = [m][\ddot{x}]_{local} \qquad\qquad [R^{-1}]$$

$$[F]_{chip}$$
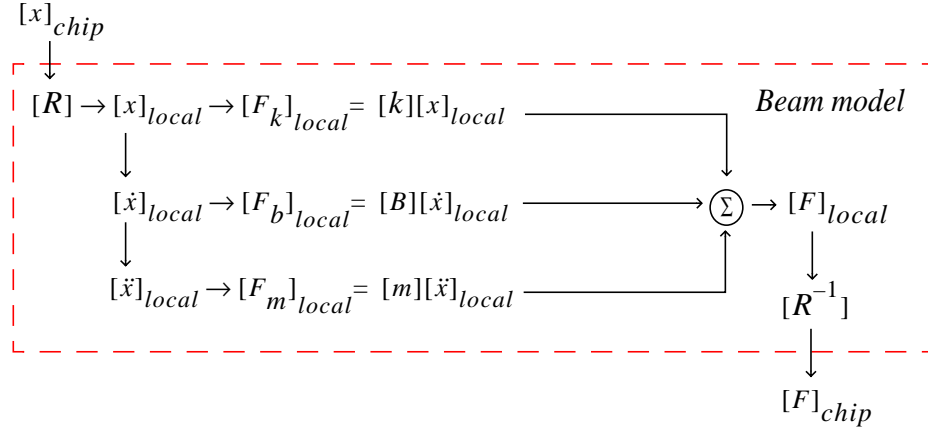
Figure 3.12: Algorithmic structure of the linear beam model

in the chip frame. The solving procedure in Figure 3.12 continues until a self-consistent solution is found for the entire system.

3.3.1.6 *Verilog-A Code Implementation*

Current NODAS models are written in Verilog-A [39]. The behavioral simulation is performed by Spectre [33][34], one of the built-in circuit network simulators in Cadence.

The Verilog-A sign convention is: *a variable flowing **out** of a node has a **positive** value.* The NODAS sign conventions are: 1. for across variables, ***positive** valued displacements are in **positive** axial direction* and ***positive** valued angles are **counterclockwise** about the axis (right-hand rule)*; 2. for through variables, *forces flowing **into** a node act in **positive** axial direction* and *moments flowing **into** a node act **counterclockwisely** about the axis.*

Segments of the linear beam model are given below to show the implementation of the algorithmic structure of the model:

```
// include header files
`include "../../constants.h"
//discipline.h is where the across/through variables for multiple physical disciplines are defined
`include "../../discipline.h"
`include "../../process.h"
`include "../../design.h"
module beam(xa, xb, phia, phib, va, vb);
// definition of bus pins
inout [0:2] xa;
```

```
kinematic [0:2] xa;      // discipline 1
inout [0:2] phia;
rotational [0:2] phia;  // discipline 2
inout [0:2] va;
electrical [0:2] va;      // discipline 3
...
// user-specified parameters
parameter real l = 100e-6;
...
// definition of internal variables
real Iz, Iy, G, It, k_1_1, m_1_1
...
// definition of internal nodes
kinematic Vya, Vyb;
rotational_omega  Vphiza, Vphizb;
...
// begin of analog block
analog begin
// calculation of internal variables
area = t*w;
Iz = 1/12.0*t*w*w*w;
...
// [k] matrix
k0_1_1 = ea/l;
k0_2_2 = 12.0*E*Iz/pow(l,3);
...
// [m] matrix
m_1_1 = 1.0/6.0*m0;
m_2_2 = m0*13.0/35.0;
...
// [B] matrix
B_1_1 = 1.0/6.0*Bx;
B_2_2 = 13.0/35.0*By;
...
// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
...
// measure displacements in chip frame
chip_xa = Pos(xa[0]);
chip_phixa = Theta(phia[0]);
...
// transform displacements from chip frame into local frame
l_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
l_phixa = l1*chip_phixa + m1*chip_phiya + n1*chip_phiza;
```

```
...
// [Fk]=[k][x], spring forces/moments in local frame
Fkxa = k_1_1*l_xa+k_7_1*l_xb;
Fkya = k_2_1*l_xa+k_2_2*l_ya+k_3_2*l_za
       +k_4_2*l_phixa+k_5_2*l_phiya+k_6_2*l_phiza
       +k_7_2*l_xb+k_8_2*l_yb+k_9_2*l_zb
       +k_10_2*l_phixb+k_11_2*l_phiyb+k_12_2*l_phizb;
...
// [F1]=[m][a]+[B][v], inertial and damping forces/moments
Pos(Vxa) <+  ddt(l_xa);
Omega(Vphixa) <+  ddt(l_phixa);
Fmxa = m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
Fbxa = B_1_1*ddt(l_xa)+B_7_1*ddt(l_xb);
...
// transform forces/moments from local frame back to chip frame
chip_Fxa=inv_l1*(Fkxa+Fmxa+Fbxa)+inv_m1*(Fkya+Fmya+Fbya)+inv_n1*(Fkza+Fmza+Fbza
);
...
// forces/moments applied to the ends of beam
F(xa[0]) <+ - chip_Fxa;
...
end          // end of analog block
endmodule  // end of module
```

The physical equations could be implemented in the aHDLs in multiple alternative ways. It has been noticed that the language implementation of the model affects the convergence property of the simulation [62]. The inclusion of internal states for the acceleration node representation adversely affect the convergence property of the simulation. These states are unnecessary and should not be included.

3.3.1.7 *Verification and Composibility of the Beam Elements*

A cantilever beam under a force or a moment at the free end, as shown in Figure 3.13, is simulated in NODAS with one and two beam elements. The results of DC and AC analysis are compared with the results from FEA simulation in ABAQUS with 50 beam element (B33H) and with 50 brick elements (C3D20), as given in Table 3-2.

The beam is of a length of 100 μm, a width of 2 μm, a thickness of 2 μm, a Young's
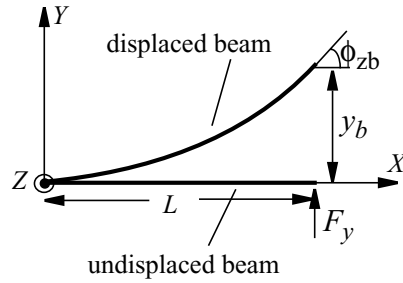
Figure 3.13: A cantilever beam under a force or moment at the free end

Table 3-2: Simulation Result of a Cantilever Beam (L = 100 µm, w = 2 µm, t = 2 µm, E = 165 GPa, F_y = 1 uN)

	NODAS	Abaqus (B33H)	Abaqus (C3D20)	error NODAS vs. B33H	error NODAS vs. C3D20
x_b (m)	1.5152e-10	1.5152e-10	1.5153e-10	0	0.01%
f_x (Hz)	21.53M (2 beam) 23.17M (1 beam)	21.04M	21.04M	2.3% 10.3%	2.3% 10.3%
y_b (m)	1.5152e-6	1.5149e-6	1.5097e-6	0.02%	0.36%
f_y (Hz)	273.17k	271.9k	272.5k	0.47%	0.25%
z_b (m)	1.5152e-6	1.5149e-6	1.5097e-6	0.02%	0.36%
f_z (Hz)	273.17k	271.9k	272.5k	0.47%	0.25%
ϕ_{xb} (rad)	6.993e-2	6.993e-2	6.4462e-2	0	8.5%
$f_{\phi x}$ (Hz)	12.3027M (2 beam) 13.1826M (1 beam)	12M	13.05M	2.5% 9.8%	5.7% 1%
ϕ_{yb} (rad)	4.545e-2	4.5477e-2	4.5106e-2	0.06%	0.8%
$f_{\phi y}$ (Hz)	273.17k	271.9k	272.5k	0.47%	0.25%
ϕ_{zb} (rad)	4.545e-2	4.5477e-2	4.5106e-2	0.06%	0.8%
$f_{\phi z}$ (Hz)	273.17k	271.9k	272.5k	0.47%	0.25%

Modulus of 165 GPa and a mass density of 2330 kg/m^3. It's clearly shown that with one NODAS beam element, the errors are within 1% for deflection modes but not for axial and torsional modes, while with two beam elements, the error for axial mode is reduced to within 3% and the error for torsional mode is reduced to within 10%. Using more beam

elements gives even better results.

To verify the composibility of the beam model, the physical cantilever beam is further simulated in NODAS with five, ten, twenty and fifty beam elements. Figure 3.14 gives the comparison to FEA simulations. Figure 3.14 (a) and (b) are for the case where a force of F_y is applied at the free end of the beam in *y*-direction. The *y*-displacements at the free end are compared to the results given by simulation in Abaqus using fifty B33H beam elements and fifty C3D20 brick elements, respectively. The force, Fy, is swept from 0 µN to 10 µN for each simulation. The percentage errors vs. the number of beam elements and the value of Fy are given in Figure 3.14 (a) and (b), showing that the linear beam model is a composable model. Using more beam elements to compose the actual physical beam gives better
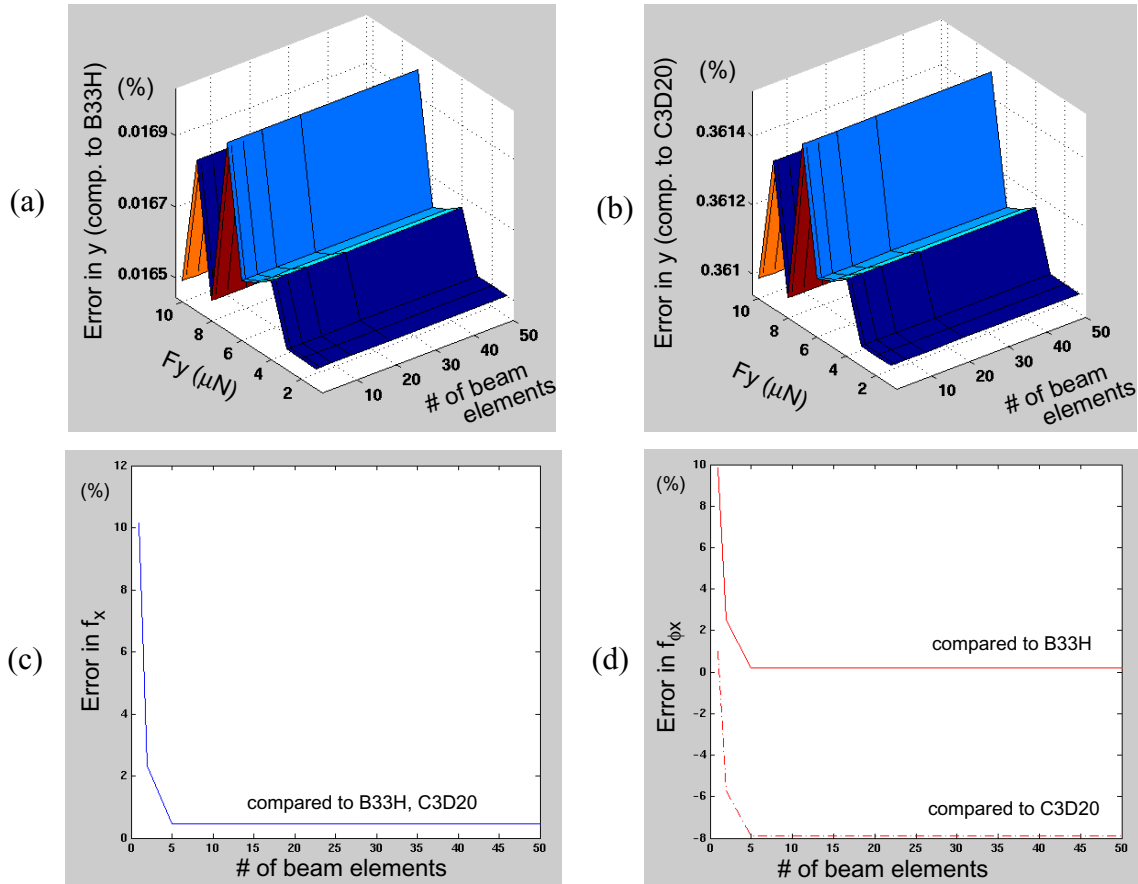
Figure 3.14: NODAS simulation of the cantilever beam with different numbers of linear beam elements (a) error in *y*-displacement compared to FEA simulation with B33H (b) error in *y*-displacement compared to FEA simulation with C3D20 (c) error in f_x compared to FEA simulation with B33H and C3D20 (d) error in $f_{\phi x}$ compared to FEA simulation with B33H and C3D20.

simulation accuracy. The simulation results converge along with the increase of the number of beam elements used in the simulation. For the lateral deflection mode, using one beam element already gives very good simulation accuracy to within 0.02% compared to B33H and within 0.36% compared to C3D20, thus one beam element is accurate enough for simulation of this kind of cases. Figure 3.14 (c) shows the percentage error in the resonant frequency of the axial mode, f_x, compared to Abaqus simulation with B33H and C3D20. The comparison shows that the results converge along with the increase of number of beam elements. For the axial mode, the error in resonant frequency reduces from 10% to within 3% when two elements are used, and the result converges to within 0.5% when five or more beam elements are used. Figure 3.14 (d) shows the percentage error in the resonant frequency of the torsional mode, $f_{\phi x}$. The percentage error compared to B33H reduces from 10% to within 3% when two beams are used, and converge to within 5% when five or more elements are used. The percentage error compared with C3D20 converges to about 8%, though, validating the composibility of the beam model, however, implying that the Euler-Bernoulli beam elements (NODAS beam and B33H) don't fully capture the torsional beam mechanics.

As mentioned, the linear beam model are for slender beams where shear effects are neglected. To explore the application range of the model, simulation for cantilever beams with varying w/L ratio are performed. The beam length, L, is fixed to 100 μm, while the width, w, varies from 2 μm to 100 μm. A force of 1 μN is applied to the free end in y-direction. DC simulation is performed and y-displacements are plotted. Figure 3.15 (a) gives the comparison of y-displacements given by NODAS and ABAQUS simulation with 50 Timoshenko (shear flexible) beam elements - B32H. Figure 3.15 (b) gives the percentage error in y-displacement. It's shown that shear effects starts to become non-negligible (error > 5%) when the w/L ratio reaches to 0.25. Thus the beam model without shear effects as given in this section is applicable to beams with the width less than 25% of the length. For
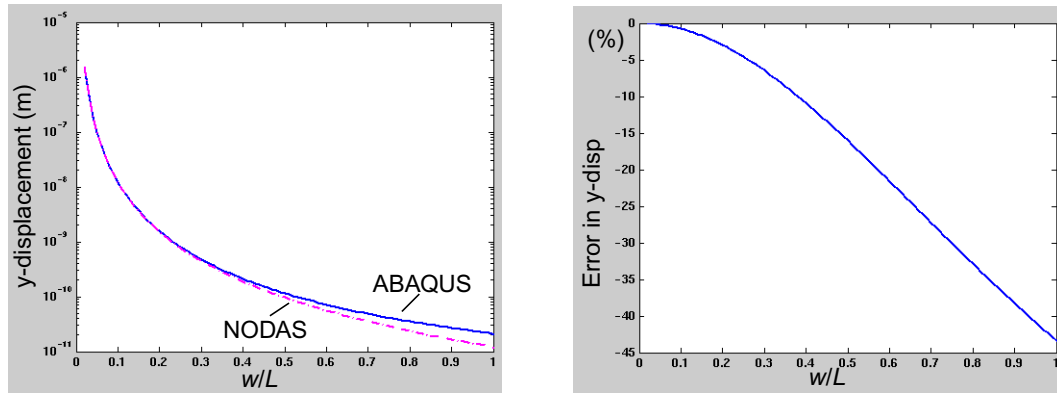
Figure 3.15: Simulation of cantilever beam with varying *w/L* (a) comparison of NODAS results to FEA results (b) percentage error in *y*-displacements

wider beams, shear effects must be included in the simulation. The modeling of shear effects in beams are given later in Chapter 4 as an example of adding new physical effects into existing model prototypes.

Test simulations are also performed with many other load conditions and beam orientations. Figure 3.16 shows several example cases with the graph of real structures and the corresponding NODAS schematics. Notice that in order to use the beam model properly to get correct simulation results, the orientation of the beam instance in the schematic, *i.e.*,
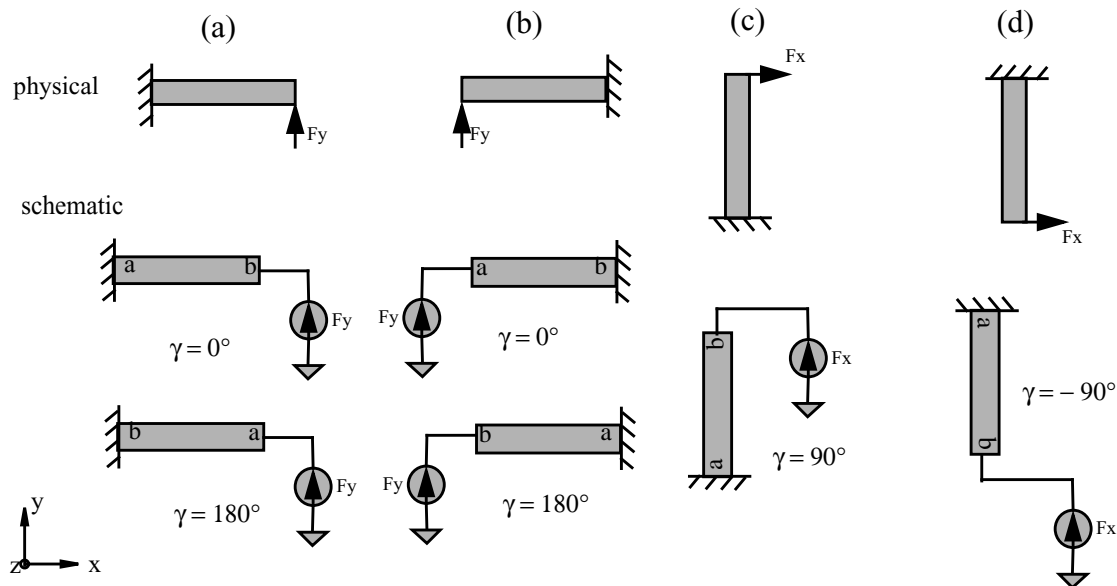
Figure 3.16: Consistent physical orientation, symbol orientation and angle parameter for a cantilever beam.

the relative position of node *a* and node *b* to the chip axes, must be consistent with the assigned value of the angle parameters. As shown in Figure 3.16, a horizontal beam instance with $\gamma = 0°$ is physically orientated with node *a* at the left and node *b* at the right, and a vertical beam instance with $\gamma = 90°$ is physically orientated with node *a* at the bottom and node *b* at the top. The symbol on the schematic can be placed in any arbitrary orientations. However, it is recommended that the symbol orientation matches the physical orientation and the value of angle parameter to avoid confusion.

3.3.2 Nonlinear Beam Model

The stiffness matrix in the linear beam model is only accurate for applications where the displacements are small. Although the linear model covers the basic beam mechanics existing in many devices such as low-frequency resonators and accelerometers, it is unsuitable for simulation of devices such as large-stroke actuators [48] and RF-MEMS switches [46] where the beam nonlinearity is non-negligible. Therefore, in this section, the NODAS linear beam model is extended to cover the beam nonlinearity.

Beam nonlinearity arises from two main effects: material nonlinearity, where the relation between the strain and the stress is nonlinear; and geometric nonlinearity, where the material is linear elastic but the relation between force and displacement of an element is nonlinear. The material nonlinearity is beyond the scope of this thesis. Our discussion will be restricted to the geometric nonlinearity. Typical applications involving geometric beam nonlinearity are large-stroke actuators, in which the beams are stiffened due to large deflection, and fixed-fixed beam devices, in which the beam starts to behave nonlinearly at very small displacements due to large axial stresses. These cases represent the main sources of geometric nonlinearity: large axial stress and large geometric deflection.

There are various methods to model geometric nonlinearity. Analytical solutions are available for several special cases of geometric nonlinearity such as a cantilever beam under single concentrated lateral load at the free end (the exact shape of the elastic curve is called

the "elastica") [52], a cantilever beam under both lateral force and moment [63], and a fixed-fixed beam with a single concentrated lateral load at the midpoint of the beam [66]. Nonlinear beams can also be analyzed using iterative finite element methods. In NODAS, the nonlinear beam model is aimed at handling both large axial stress and large geometric deflection, and being reusable and composable as well for use as a building block for simulation of beam networks in analog behavioral simulators.

3.3.2.1 Large Axial Stress

Beam stiffening due to large axial stress is one main source of geometric nonlinearity. For example, Figure 3.17(a) shows a fixed-fixed beam with a force, F_y, applied at the center in y-direction. When the beam bends up in the y-direction, the beam center point does not displace in x-direction because of the symmetric boundary conditions. The fixed-fixed beam is therefore forced to be in tension, with a large axial force, N, generated in the beam. Figure 3.17(b) shows the displacement of the center point in y-direction given by the static
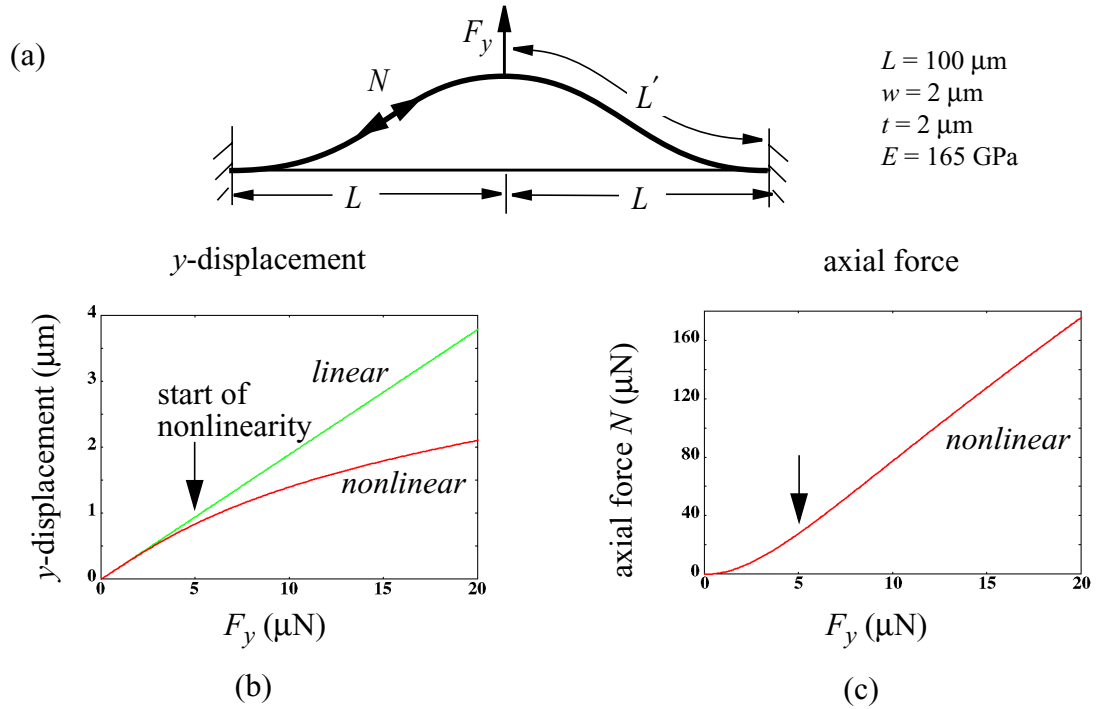
Figure 3.17: Beam stiffening due to large axial stress (a) a fixed-fixed beam in lateral bending (b) y-displacement of the center point (c) axial force in the beam

simulation in ABAQUS using linear and nonlinear beam elements, respectively. As indicated by the arrow, the 200 μm long fixed-fixed beam starts to behave nonlinearly and is stiffened at a very small displacement of less than 1 μm. Figure 3.17(c) explains why the nonlinearity happens at such a small deflection - the axial force generated in the beam is already as large as 30 μN at this small deflection! This example clearly verifies that the large axial stress is an important source of geometric nonlinearity, especially for structures such as MEMS switches, where fixed-fixed beams are widely used.

Variational Methods

The mechanics of nonlinear in-plane beam bending with axial stress has been studied extensively [43][63][66] and is given by the Euler-Bernoulli equation:

$$EI\frac{d^4 u_y}{dx^4} - N\frac{d^2 u_y}{dx^2} = 0 \qquad (3.84)$$

where the axial force, *N*, is the *external* force applied to the beam in its axial direction and only concentrated loads are applied. When there is no external axial forces, that is, $N = 0$, the equation simply reduces to

$$\frac{d^4 u_y}{dx^4} = 0 \qquad (3.85)$$

which is exactly the equation describing the mechanics of linear beam bending ((3.16)).

The Euler-Bernoulli equation has been directly used in many references as the basic governing equation for beam bending. To help understand the origin of this equation, in this section, we give the detailed derivation of (3.84) based on the variational methods [53][67].

The principle of virtual work, introduced by Bernoulli, is the most fundamental statement of the static equilibrium of mechanical systems [44][53]. D'Alembert's principle extends it to the dynamic case [53]. Hamilton's principle further extends D'Alembert's principle to reduce the dynamics problem to the problem of a scalar integral independent of the coordinate system being used. In stead of considering the variations of the general

coordinates, Hamilton's principle considers the entire motion of the system between two instants t_1 and t_2. The general format of Hamilton's principle is:

$$\delta \int_{t_1}^{t_2} L \, dt = 0 \tag{3.86}$$

where L is the Lagrangian and δ is a symbol introduced by Lagrange to represent the infinitesimal arbitrary variations in the coordinates of the system. When only the static equilibrium of the bending beam is considered, (3.86) reduces to:

$$\delta \int_{t_1}^{t_2} U \, dt = 0 \tag{3.87}$$

where U is the bending strain energy of the beam element given by [43]:

$$U = \frac{EA}{2} \int_0^L \left(\frac{\partial u_0}{\partial x} \right)^2 dx + \frac{EI}{2} \int_0^L \left(\frac{\partial^2 u_y}{\partial x^2} \right)^2 dx + \frac{EA}{2} \int_0^L \left(\frac{\partial u_0}{\partial x} \right) \left(\frac{\partial u_y}{\partial x} \right)^2 dx \tag{3.88}$$

E is the Young's Modulus, A is the cross-section area, I is the planar moment of inertia, $u_0(x, t)$ is the axial displacement in x-direction along the beam and $u_y(x, t)$ is the lateral displacement in y-direction along the beam. To be consistent with [43], u_y is used in the following derivation while it represents the same displacement function as $y(x)$ in the static equilibrium equations (3.84) and (3.85).

Substitute (3.88) into (3.87) and move the operator delta into the integral, we get:

$$\int_{t_1}^{t_2} \left[\frac{EA}{2} \int_0^L \delta \left(\frac{\partial u_0}{\partial x} \right)^2 dx + \frac{EI}{2} \int_0^L \delta \left(\frac{\partial^2 u_y}{\partial x^2} \right)^2 dx + \frac{EA}{2} \int_0^L \delta \left[\left(\frac{\partial u_0}{\partial x} \right) \left(\frac{\partial u_y}{\partial x} \right)^2 \right] dx \right] dt$$

$$= \int_{t_1}^{t_2} Q \, dt = 0 \tag{3.89}$$

Assuming operator δ and $\partial/\partial x$ are commutative, the integral for (3.89), Q, is rewritten as:

$$Q = \frac{EA}{2} \int_0^L 2 \left(\frac{\partial u_0}{\partial x} \right) \delta \left(\frac{\partial u_0}{\partial x} \right) dx + \frac{EI}{2} \int_0^L 2 \left(\frac{\partial^2 u_y}{\partial x^2} \right) \frac{\partial^2}{\partial x^2} (\delta u_y) dx$$

$$+ \frac{EA}{2} \int_0^L \left[\frac{\partial}{\partial x} (\delta u_0) \left(\frac{\partial u_y}{\partial x} \right)^2 + \left(\frac{\partial u_0}{\partial x} \right) 2 \left(\frac{\partial u_y}{\partial x} \right) \left(\frac{\partial}{\partial x} (\delta u_y) \right) \right] dx \tag{3.90}$$

Recall the theorem of integration by parts

$$\int_0^L u\,dv = uv\Big|_0^L - \int_0^L v\,du \tag{3.91}$$

function Q in (3.90) is rewritten as:

$$
\begin{aligned}
Q =\ & \frac{EA}{2}\left[2\left(\frac{\partial u_0}{\partial x}\right)(\delta u_0)\Big|_0^L - \int_0^L (\delta u_0)2\left(\frac{\partial^2 u_0}{\partial x^2}\right)dx\right] \\[2mm]
& + \frac{EI}{2}\left[2\left(\frac{\partial^2 u_y}{\partial x^2}\right)\frac{\partial}{\partial x}(\delta u_y)\Big|_0^L - \underbrace{\int_0^L \frac{\partial}{\partial x}(\delta u_y)2\left(\frac{\partial^3 u_y}{\partial x^3}\right)dx}\right] \\[2mm]
& + \frac{EA}{2}\left[\left(\frac{\partial u_y}{\partial x}\right)^2(\delta u_0)\Big|_0^L - \int_0^L (\delta u_0)2\left(\frac{\partial u_y}{\partial x}\right)\left(\frac{\partial^2 u_y}{\partial x^2}\right)dx\right] \\[2mm]
& + \frac{EA}{2}\left[2\left(\frac{\partial u_0}{\partial x}\right)\frac{\partial u_y}{\partial x}(\delta u_y)\Big|_0^L - \int_0^L (\delta u_y)\left[2\left(\frac{\partial^2 u_0}{\partial x^2}\right)\left(\frac{\partial u_y}{\partial x}\right) + 2\left(\frac{\partial^2 u_y}{\partial x^2}\right)\left(\frac{\partial u_0}{\partial x}\right)\right]dx\right]
\end{aligned}
\tag{3.92}
$$

in which the term in the large bracket can be further integrated by parts and rewritten as:

$$-\frac{EI}{2}\int_0^L \frac{\partial}{\partial x}(\delta u_y)2\left(\frac{\partial^3 u_y}{\partial x^3}\right)dx = -\frac{EI}{2}\left[2\left(\frac{\partial^3 u_y}{\partial x^3}\right)(\delta u_y)\Big|_0^L - \int_0^L (\delta u_y)2\left(\frac{\partial^4 u_y}{\partial x^4}\right)dx\right] \tag{3.93}$$

Next, we separate the terms with evaluation at boundaries from the terms with integrations and reorganize them, then substitute into the integration about time ((3.89)):

$$
\begin{aligned}
\int_{t_1}^{t_2}\Bigg\{ & EA\left[\left(\frac{\partial u_0}{\partial x}\right) + \frac{1}{2}\left(\frac{\partial u_y}{\partial x}\right)^2\right](\delta u_0)\Big|_0^L \\[2mm]
& -EA\left(\int_0^L\left[\frac{\partial^2 u_0}{\partial x^2} + \frac{\partial u_y}{\partial x}\frac{\partial^2 u_y}{\partial x^2}\right](\delta u_0)dx\right) \\[2mm]
& +\left[EI\frac{\partial^2 u_y}{\partial x^2}\frac{\partial}{\partial x}(\delta u_y) + EA\frac{\partial u_0}{\partial x}\frac{\partial u_y}{\partial x}(\delta u_y) - EI\frac{\partial^3 u_y}{\partial x^3}(\delta u_y)\right]\Bigg|_0^L \\[2mm]
& +\int_0^L\left[EI\frac{\partial^4 u_y}{\partial x^4} - EA\left(\frac{\partial^2 u_0}{\partial x^2}\frac{\partial u_y}{\partial x} + \frac{\partial^2 u_y}{\partial x^2}\frac{\partial u_0}{\partial x}\right)\right](\delta u_y)dx\Bigg\}dt = 0
\end{aligned}
\tag{3.94}
$$

(3.94) must be satisfied for arbitrary δu_0 and δu_y, therefore, we obtain the following

simultaneous equations:

$$EA\left[\left(\frac{\partial u_0}{\partial x}\right) + \frac{1}{2}\left(\frac{\partial u_y}{\partial x}\right)^2\right](\delta u_0)\Big|_0^L = 0 \tag{3.95}$$

$$\left[EI\frac{\partial^2 u_y}{\partial x^2}\frac{\partial}{\partial x}(\delta u_y) + EA\frac{\partial u_0}{\partial x}\frac{\partial u_y}{\partial x}(\delta u_y) - EI\frac{\partial^3 u_y}{\partial x^3}(\delta u_y)\right]\Bigg|_0^L = 0 \tag{3.96}$$

$$EA\left(\frac{\partial^2 u_0}{\partial x^2} + \frac{\partial u_y}{\partial x}\frac{\partial^2 u_y}{\partial x^2}\right) = 0 \tag{3.97}$$

$$EI\frac{\partial^4 u_y}{\partial x^4} - EA\left(\frac{\partial^2 u_0}{\partial x^2}\frac{\partial u_y}{\partial x} + \frac{\partial^2 u_y}{\partial x^2}\frac{\partial u_0}{\partial x}\right) = 0 \tag{3.98}$$

(3.95) and (3.96) represent the conditions that u_0 and u_y must satisfy at the boundaries

and (3.97) and (3.98) are the governing differential equations for the beam bending. Solve

(3.97) and (3.98) simultaneously, we get:

$$EI\frac{\partial^4 u_y}{\partial x^4} - EA\left(-\left(\frac{\partial u_y}{\partial x}\right)^2 + \frac{\partial u_0}{\partial x}\right)\frac{\partial^2 u_y}{\partial x^2} = 0 \tag{3.99}$$

Defining N as the axial force:

$$N = EA\left(-\left(\frac{\partial u_y}{\partial x}\right)^2 + \frac{\partial u_0}{\partial x}\right) \tag{3.100}$$

(3.99) is rewritten as:

$$EI\frac{\partial^4 u_y}{\partial x^4} - N\frac{\partial^2 u_y}{\partial x^2} = 0 \tag{3.101}$$

which has the same format as the Euler-Bernoulli equation given in (3.84). When

$$\left(\frac{\partial u_y}{\partial x}\right)^2 \ll \frac{\partial u_0}{\partial x} \tag{3.102}$$

the axial force N in (3.100) reduces to

$$N = EA\frac{\partial u_0}{\partial x} \tag{3.103}$$

which is the same as the external load defined in [43] and will be discussed in next section.

Geometric Stiffness Matrix

From (3.84), we see that the external axial load not only causes axial displacements, but also affects lateral displacements. The beam bends laterally without external lateral forces and bending moments. In another words, when beam geometric nonlinearity is considered, the displacement in *x* and *y* directions are no longer independent with each other, as was assumed in the linear beam modeling in section 3.3.1.1. Compressive axial loads cause bending in the beam, as shown in Figure 3.18. When the loads reach a critical value, the beam will buckle.

The geometric nonlinearity is modeled by a geometric stiffness matrix, $[k_G]$, which is derived using the energy methods [43], yielding a nonlinear force-displacement relation:

$$
\begin{bmatrix} F_{xa} \\ F_{ya} \\ M_{za} \\ F_{xb} \\ F_{yb} \\ M_{zb} \end{bmatrix} = \underbrace{\left(\underbrace{\begin{bmatrix} \frac{EA}{L} & & & & & \\ 0 & \frac{12EI_z}{L^3} & & \text{symmetric} & & \\ 0 & \frac{6EI_z}{L^2} & \frac{4EI_z}{L} & & & \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & & \\ 0 & -\frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & \\ 0 & \frac{6EI_z}{L^2} & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{4EI_z}{L} \end{bmatrix}}_{[k_0]} + N \underbrace{\overbrace{\begin{bmatrix} 0 & & & & & \\ 0 & \frac{6}{5L} & & \text{symmetric} & & \\ 0 & \frac{1}{10} & \frac{2L}{15} & & & \\ 0 & 0 & 0 & 0 & & \\ 0 & -\frac{6}{5L} & -\frac{1}{10} & 0 & \frac{6}{5L} & \\ 0 & \frac{1}{10} & -\frac{L}{30} & 0 & -\frac{1}{10} & \frac{2L}{15} \end{bmatrix}}^{[k_G]}}_{[k_{G0}]} \right)}_{[k]} \begin{bmatrix} x_a \\ y_a \\ \phi_{za} \\ x_b \\ y_b \\ \phi_{zb} \end{bmatrix}
$$

(3.104)
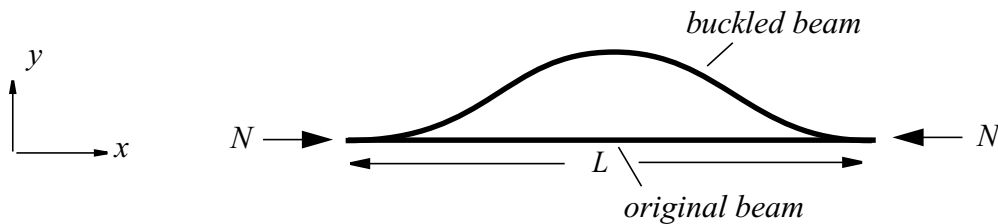
Figure 3.18: Beam bending due to axial stress

where k_0 is the linear stiffness matrix, $[k]$ is the entire nonlinear stiffness matrix which is the sum of the linear and geometric stiffness matrices:

$$\left[k\right] = \left[k_0\right] + \left[k_G\right] \tag{3.105}$$

In (3.104), the geometric stiffness matrix is defined as:

$$\left[k_G\right] = N\left[k_{G0}\right] \tag{3.106}$$

where $[k_{G0}]$ is a constant matrix determined by geometric sizes and N is the axial force which has a linear relationship with the node displacements in x-direction:

$$N = EA\frac{\partial u_0}{\partial x} = \frac{EA}{L}(x_b - x_a) \tag{3.107}$$

Examining (3.104) carefully, we notice that:

1. The lateral bending is affected by the axial loads. Different directions of the external axial load have different effects on the lateral stiffness. Compressive loads, as in Figure 3.18, give negative values of N, thus leading to beam softening; while tensile loads give positive values of N, thus leading to beam stiffening, as in Figure 3.17.

2. Matrix $[k_{G0}]$ has all zero elements for the rows and columns related to x-direction, implying that the axial displacement is still solely determined by the external axial loads, the same as in the linear beam model;

3. The axial force N is dependent on the x-displacements at beam ends. If x-displacements are zero, N will be zero, and the geometric stiffness matrix $[k_G]$ will be null.

Point 3 reveals a problem in the geometric stiffness matrix given in (3.104). Consider a fixed-fixed beam with a force F_y in the middle (Figure 3.17(a)). At least two beam elements must be used to compose the fixed-fixed beam in order to apply the force at the center. The ABAQUS simulation results (Figure 3.17(b) and (c)) indicate that the axial force is non-zero and the beam is stiffened, therefore $[k_G]$ should be a non-zero matrix. However, if we apply (3.107) and (3.104) directly to the fixed-fixed beam, we will see that due to the

symmetric boundary condition at the center of the beam, x displacements are forced to be zero, leading to a null axial force N and hence a null $[k_G]$, which is not physically true. (3.107) is correct only for cases where the axial forces are external loads causing displacements in x-direction. The equation has to be modified in order to correctly model the internal axial force.

Modified Definition of the Axial Force, N

As shown in Figure 3.19, when a beam is in tension, the actual beam length L' is longer than the original length L. Although there is no displacement in the x direction at the beam ends, the bending from tensile stress still generates a non-zero internal axial force. Therefore, we model N as:

$$N = \frac{EA}{L}\Delta L = \frac{EA}{L}(L' - L) \tag{3.108}$$

where L' is the actual length along the center line of the beam. We call L' the *effective beam length*.

With the modified model for N, the axial force is now a function of the variance in beam length, ΔL, rather than directly as a function of nodal x-displacements, Δx. The direction of the axial force is along the bent beam, rather than x-axis, the original axial direction. This is physically correct. The null $[k_G]$ problem caused by (3.104) in the case of fixed-fixed beam is solved by this new definition. The modified definition is also valid for cases where the axial force N is an external load causing displacements in x-direction. In those cases, ΔL is simply the same as Δx, for small displacements.
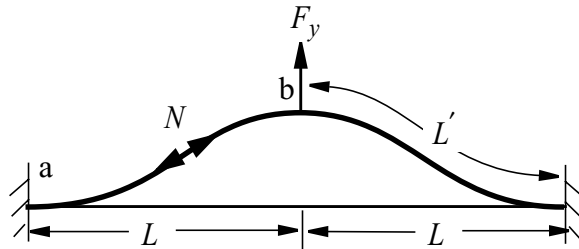
Figure 3.19: Fixed-fixed beam in lateral bending and axial tension

The effective beam length is calculated by integrating the arc length *ds* along the curved beam based on the cubic shape functions for small angle beam bending, as shown in Figure 3.20:

$$L^{'} = \int ds = \int_{x_a}^{L + x_b} \sqrt{1 + \left(\frac{du_y}{dx}\right)^2} \, dx \tag{3.109}$$

Now consider a beam with lateral forces in *y*-direction at the beam ends but no external axial forces in *x*-direction. From (3.104), the axial elements in $[k_G]$ (row 1 and row 4) are all zeros, thus *x*-displacements are solely determined by the axial loads, resulting in zero *x*-displacement. However, in fact, the beam is free to bend in *x*-direction and the beam length almost remains unchanged. Thus (3.104) must be changed to cover this case. Notice that the self-consistent solution for this case is that the nonlinear displacement in *x* comes from the geometric foreshortening of the beam, where *L'* = *L*. To include this effect, the force-balance equation for the axial direction in (3.104) is altered from

$$F_{xa} = -F_{xb} = \frac{EA}{L}(x_a - x_b) = \frac{EA}{L}\Delta x \tag{3.110}$$

to

$$F_{xa} = -F_{xb} = \frac{EA}{L}\Delta L = N \tag{3.111}$$

Notice that as the cubic beam bending shape functions are obtained based on the assumption of small-deflection, the calculation of L′ in (3.109) thus the new definition of *N* in (3.108) are only good for small lateral bending (y < 15% L). To deal with large lateral deflections, other measures have to be taken.
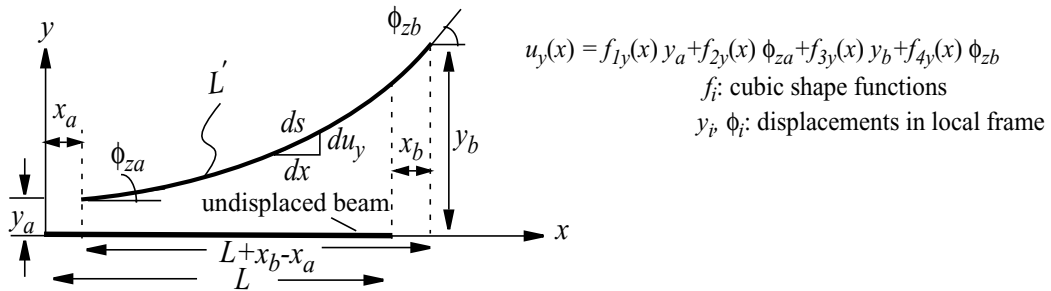
Figure 3.20: Calculation of the effective beam length
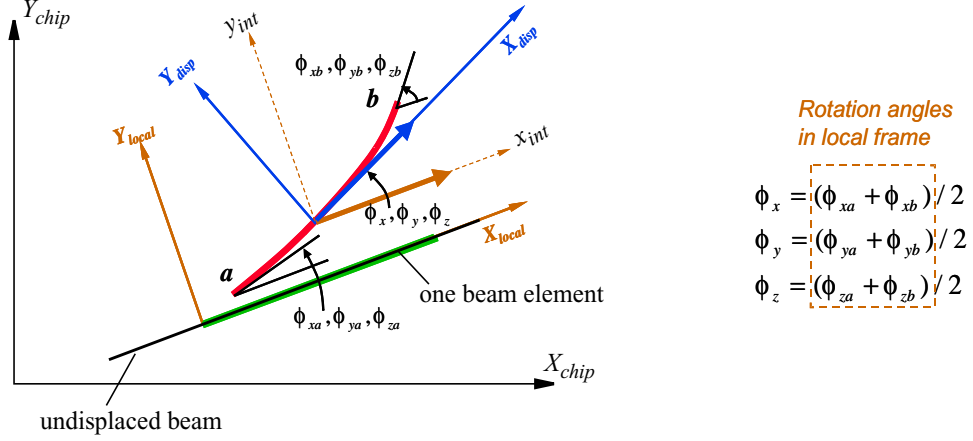
Figure 3.21: Dynamic coordinate transformation.

3.3.2.2 Large Geometric Deflection

This section will discuss the modeling of the other major source of geometric nonlinearity: large geometric deflection.

As mentioned previously, the cubic beam shape functions are only good for small deflection. To deal with large deflection accurately, a new frame of reference, called "displaced frame" is introduced, as illustrated in Figure 3.21. The coordinate transformation between the local frame and the displaced frame is a dynamic transformation based on an equivalent rotation vector $[\phi_x \ \phi_y \ \phi_z]^T$. The rotation vector is derived from the average of angular displacements at beam ends *a* and *b*, allowing the displacements at both ends to be taken account into the simulation. This is critical for the symmetry of the nonlinear beam model, to ensure that switching the loads and boundary conditions from one end of the beam to the other gives symmetric answers and turing the beam around by 180° gives the same answer. The model symmetry guarantees the model to work for every configuration and orientation cases, which is especially important for hierarchical designs where people usually design a sub-schematic and then rotate it around to compose other parts of the design with symmetric topologies.

Dynamic Coordinate Transformation

The first step of the transformation is a translation about the center of the displaced beam, leading to the intermediate frame of reference. The transformation is then followed by a 3D rotation. There are many different ways to describe a 3D rotation, such as the rotation matrix based on Euler angles [61] and the rotation matrix based on rotation about a single axis [68][69]. The latter is used in NODAS because of its unique representation and independence on the order of rotations. In this method, any 3D rotation, no matter how the rotation is implemented, is represented as an equivalent rotation about a rotation axis *p* by a rotation angle of ϕ, as shown in Figure 3.22. This method is commonly used in finite element packages [69].

The corresponding rotation matrix is:

$$[R] = \begin{bmatrix} up_x^2 + c & up_x p_y + sp_z & up_x p_z - sp_y \\ up_x p_y - sp_z & up_y^2 + c & up_y p_z + sp_x \\ up_x p_z + sp_y & up_y p_z - sp_x & up_z^2 + c \end{bmatrix} \tag{3.112}$$

where $p = [p_x\, p_y\, p_z]^T$, $c = cos\phi$, $s = sin\phi$, and $u = 1-cos\phi$. When $[R]$ is used to describe the rotation between the chip frame and the local frame for an in-plane beam element with an orientation angle Θ, $p = [0\ 0\ 1]^T$, and the rotation matrix simply reduces to:
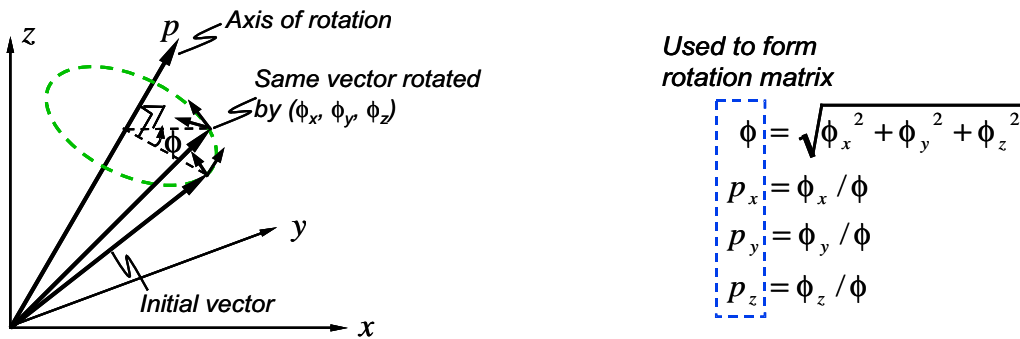
Figure 3.22: Equivalent 3D rotation about a single axis

$$[R] = \begin{bmatrix} \cos\Theta & \sin\Theta & 0 \\ -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.113}$$

By using this technique, as long as enough beam elements are used to compose the physical beam, the large displacements of each beam segment in the local frames will be transformed into displacements in the displaced frame which are small enough to satisfy the small-deflection assumption of the beam bending shape functions.

Notice that ϕ varies in time along with the beam bending dynamics, thus the coordinate transformation between the displaced frame and the local frame, $[R]$, is a dynamic transformation, unlike the static coordinate transformation between the chip frame and the local frame, $[T]$, which is only a function of initial orientation angles.

Effective Beam Length

The beam length integral, (3.109), must also be calculated in the displaced frame, because small-angle deflection is the assumption used in the derivation of the cubic shape functions. Both the displacements used in the shape functions and the limits of the integration must be measured in the displaced frame:

$$L^{'} = \int ds = \int_{\tilde{x}_a}^{L + \tilde{x}_b} \sqrt{1 + \left(\frac{d}{d\xi}\tilde{u}_y(\xi)\right)^2}\, d\xi \tag{3.114}$$

$$\tilde{u}_y(\xi) = f_{1y}(\xi)\tilde{y}_a + f_{2y}(\xi)\tilde{\phi}_{za} + f_{3y}(\xi)\tilde{y}_b + f_{4y}(\xi)\tilde{\phi}_{zb}$$

where $\tilde{y}_i$ and $\tilde{\phi}_{zi}$ are displacements in the displaced frame.

As the symbolic solution to (3.114) is not available, the explicit solution to the integral is implemented using a 1$^{\text{st}}$-order Taylor series expansion of the integral:

$$L^{'} = \int_{\tilde{x}_a}^{L + \tilde{x}_b} \sqrt{1 + (\tilde{u}_y{'}(\xi))^2}\, d\xi \approx \int_{\tilde{x}_a}^{L + \tilde{x}_b} \left(1 + \frac{1}{2}(\tilde{u}_y{'}(\xi))^2\right) d\xi \tag{3.115}$$

The symbolic expression of the effective beam length is obtained in Mathematica by substituting the shape functions into (3.115).

The geometric nonlinearity in out-of-plane bending is modeled in the same way as the in-plane bending, except that the moments of inertia are different. Notice that for a 12-DOF nonlinear beam element, the in-plane deflection and the out-of-plane deflection are no longer independent. They combine to the effective beam length of bending beam in space:

$$L' = \int_{\tilde{x}_a}^{L+\tilde{x}_b} \sqrt{1 + (\tilde{u}_y'(\xi))^2 + (\tilde{u}_z'(\xi))^2} \, d\xi \approx \int_{\tilde{x}_a}^{L+\tilde{x}_b} \left(1 + \frac{1}{2}((\tilde{u}_y'(\xi))^2 + (\tilde{u}_z'(\xi))^2)\right) d\xi \quad (3.116)$$

3.3.2.3 Algorithmic Structure of the Nonlinear Beam Model

Since the nonlinear stiffness matrix is based on the effective beam length, spring forces and moments should be calculated in the displaced frame based on variables in that frame, then be transformed back to the local frame. The inertial forces and moments must be calculated in the fixed local frame rather than the displaced frame, as the displaced frame is a non-inertial dynamic frame. Same principle applies to the calculation of damping forces and moments. These three types of forces/moments are then summed up in the local frame and transformed into the chip frame altogether to join the analysis of system matrix, as shown in Figure 3.23.

Because of the small-deflection assumption, an incremental loading method is needed to obtain accurate simulation results for large deflections. A nonlinear numeric method, such as Newton-Raphson, is also needed to solve the nonlinear equations. These techniques
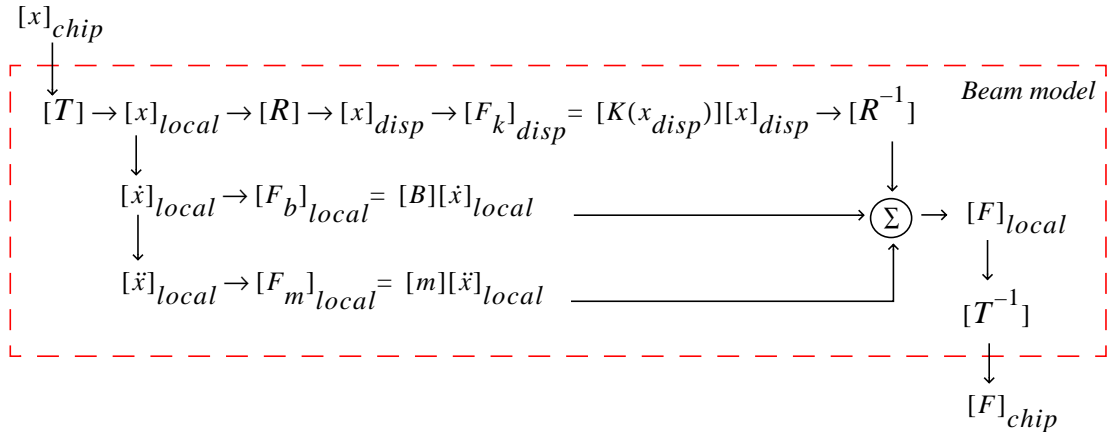
Figure 3.23: Algorithmic structure of nonlinear beam model

have to be included in the numeric solver, as done in commercial FEA tools and in behavioral simulation tools with self-maintained numeric solvers such as SUGAR. Since the NODAS models are embedded in Cadence, we take advantage of the incremental loading and Newton-Raphson methods already included in the Spectre simulator [34]. In transient analysis, if large loads are applied at the beginning of simulation or applied as a DC offset, then initial values for critical nodes should be set to aid convergence. In DC analysis with large loads, dcsweep of the load should be performed to obtain accurate nonlinear solution.

3.3.2.4 Verilog-A Code Implementation

Verilog-A code segment of the nonlinear beam model is given as follows. The fundamental structure is the same as the linear model. The dynamic coordinate transformation and calculation of forces and moments in different frames are highlighted as the specific characteristics of the nonlinear beam model.

```
 // include header files
`include "../../constants.h"
`include "../../discipline.h"
'include "../../process.h"
'include "../../design.h"
module beam_nl (xa, xb, phia, phib, va, vb);
// definition of bus pins
inout [0:2] xa;
kinematic [0:2] xa;
...
// user-specified parameters
parameter real l = 100e-6;
...
// definition of internal variables
real Iz, Iy, G, It, k_1_1, m_1_1
...
 // definition of internal variables used for nonlinear stiffness matrix
real lm, lp, l_eff, l_new, F_axial;
...
real k0_1_1;     // definition of coefficients of the linear stiffness matrix
real k1_1_1;     // definition of coefficients of the geometric nonlinear stiffness matrix
real k_1_1;      // definition of coefficients of the the entire nonlinear stiffness matrix
```

```
...
// begin of analog block
analog begin
// calculation of internal variables
area = t*w;
...
// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
...
// measure displacements in chip frame and transform into local frame
chip_xa = Pos(xa[0]);
local_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
...
// translated local frame
l_xa = local_xa - local_xa;
...
// dynamic rotation matrix from translated local frame to the displaced frame
rphix = (l_phixa+l_phixb)/2.0;
rx = rphix/rphi;
new_l1 = rt*rx*rx + rc;
...
// transform displacements into the displaced frame
new_l_xb = new_l1*(l_xb+l) + new_m1*l_yb + new_n1*l_zb - l;
...
// calculate effective beam length in the dispalced frame
lp = l+new_l_xb-new_l_xa;
l_eff = (lp*(15*pow(l_new,6)*(2+pow(new_l_phiya,2)...

// calculate the axial force
F_axial = ea*(l_eff-l)/l;

// linear stiffness matrix [k0], with shear deformation

// variables for shear deformation
Asy = 2.0/3.0*area; //effective shear area
...
k0_1_1 = ea/l;
k0_2_2 = 12.0*E*Iz/pow(l,3);
...
// geometric nonlinear stiffness matrix [kG], with shear deformation
k1_1_1 = 0;
k1_2_2 = F_axial*2.0*(6+10*Sy+5*pow(Sy,2))/(10*l*pow(1+Sy,2)); matrix
...
// sum up to get the entire nonlinear stiffness matrix [k]
```

```
k_1_1 = k0_1_1 + k1_1_1;
k_2_2 = k0_2_2 + k1_2_2;
...
// calculate the spring forces in the displaced frame
new_Fkxa = -F_axial;
new_Fkxb =  F_axial;
new_Fkya = k_2_1*new_l_xa+k_2_2*new_l_ya+k_3_2*new_l_za...
...
// transform spring forces  from the displaced frame back to the local frame
Fkxa=inv_new_l1*new_Fkxa+inv_new_m1*new_Fkya+inv_new_n1*new_Fkza;
...
// [m] matrix
m_1_1 = 1.0/6.0*m0;
m_2_2 = m0*13.0/35.0;
...
// [B] matrix
B_1_1 = 1.0/6.0*Bx;
B_2_2 = 13.0/35.0*By;
...
// calculate the inertial forces and damping forces in the local frame
Pos(Vxa) <+  ddt(local_xa);
Omega(Vphixa) <+  ddt(local_phixa);
Fmxa = m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
Fbxa = B_1_1*ddt(l_xa)+B_7_1*ddt(l_xb);
...
// sum up the forces in the local frame and transform forces/moments back to chip frame
chip_Fxa=inv_l1*(Fkxa+Fmxa+Fbxa)+inv_m1*(Fkya+Fmya+Fbya)+inv_n1*(Fkza+Fmza+Fbza
);
...
// forces/moments applied to the beam ends
F(xa[0]) <+ - chip_Fxa;
...
end         // end of analog block
endmodule  // end of module
```

3.3.2.5 Verification Simulation

To verify the accuracy of the nonlinear beam model, multiple spring topologies, including the cantilever beam, the fixed-fixed beam, the crab-leg spring and the folded-flexure spring, have been simulated and compared with FEA and exact analytical results.

Figure 3.24 shows the static analysis of a cantilever beam with a force applied in the *y*-direction, modeled with one, two and five beam elements respectively. The normalized
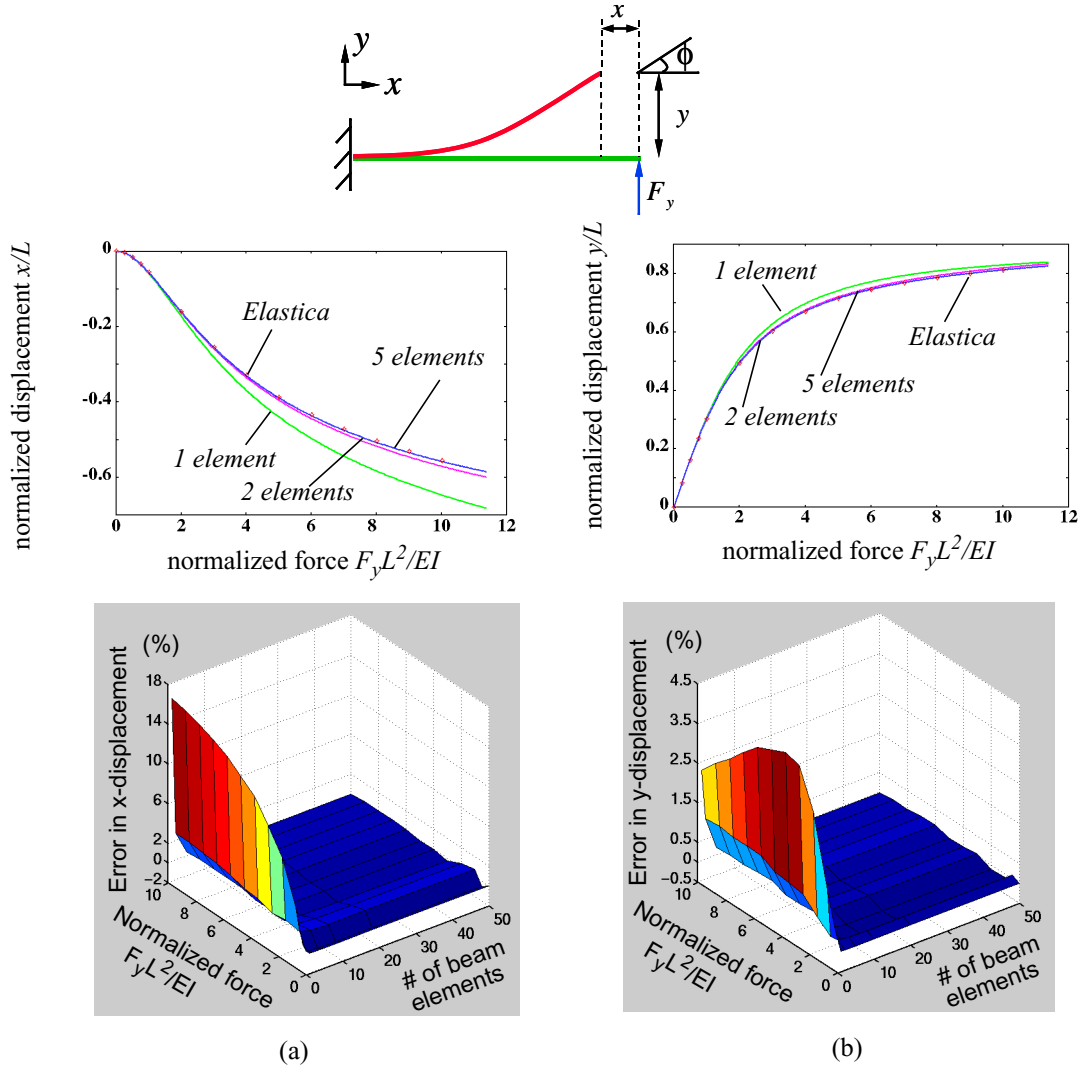
Figure 3.24: Static analysis of a cantilever beam with *Fy* applied to the free end (100 μm long, 2 μm wide, 2 μm thick, *E*=165GPa). (a) normalized *x* displacement and percentage error (b) normalized *y* displacement and percentage error.

displacements *vs.* normalized force *Fy* are plotted and compared to the elastica solution. With one beam element, the entire deflection range is modeled with an error less than 16%. Using two beam elements, the error is reduced to within 3%. With five beam elements further reduces the error to within 1%.

Figure 3.25 shows the static analysis of a cantilever beam with varying beam length from 50 μm to 250 μm. A uniform range of normalized force in *y*-direction is applied, to ensure that large deflection range is obtained in every case with different beam length. The
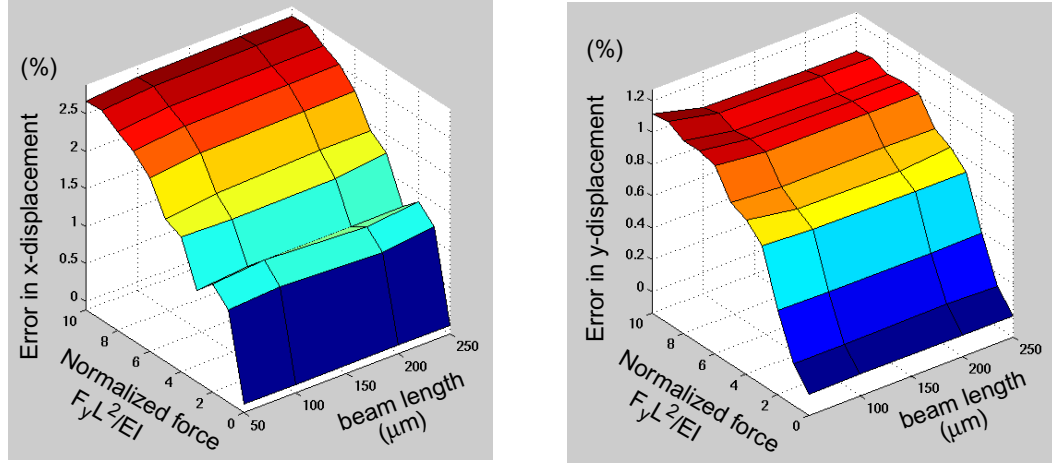
Figure 3.25: Static simulation of a cantilever beam using two beam elements (2 μm wide, 2 μm thick, *E*=165GPa, beam length varies from 50 μm to 250 μm) (a) percentage error in *x*-displacement (b) percentage error in *y*-displacement

beam is simulated using two beam elements for every set of values for force and beam length. The results shows that for all these cases, the error in *x* and *y* displacement are within 3% compared to Elastica solution.

Figure 3.26 shows the static analysis of a fixed-fixed beam with a central concentrated load, *Fy*, modeled with two, four and ten beam elements. Results are compared to FEA simulation in ABAQUS. Because the force is applied to the middle of the beam, at least two beam elements are needed for the entire simulation, one for each half. The normalized axial force in the fixed-fixed beam starts to be on the order of mN at a displacement of about 5 μm. The nonlinearity starts to be evident at a very small lateral displacement of about 1 μm. Displacement is very small even for a very large normalized lateral force of 27 (*i.e.*, 600 μN). As the lateral displacement is small, even two elements provide very accurate results with the error less than 6%. Using four and ten elements further reduce the error to within 3% and 1%, respectively.

AC simulations are also performed for the cantilever beam (Figure 3.24) and the fixed-fixed beam (Figure 3.26), respectively. The beams are stimulated by a force with DC bias F_y and a small AC amplitude of 1μN. In AC analysis, the system is linearized based on the DC
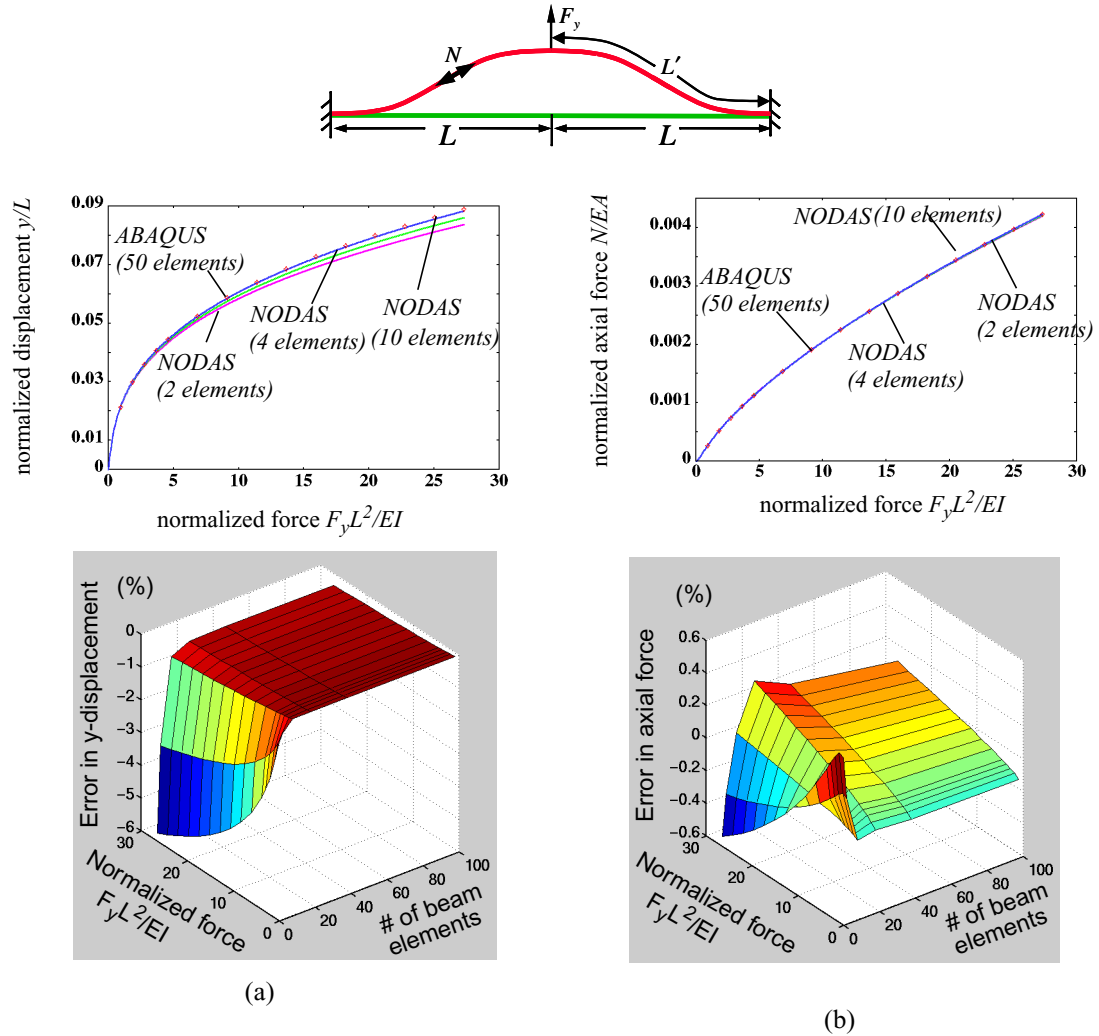
Figure 3.26: Static analysis of a fixed-fixed beam with *Fy* applied to the middle (100 μm long for each side, 2 μm wide, 2 μm thick, *E*=165GPa). (a) normalized *y* displacement and percentage error (b) normalized axial force and percentage error.

bias, therefore, varying F_y from small values to large values drives the beams from linear to nonlinear conditions, giving varying resonant frequencies. Figure 3.27 gives the resonant frequencies normalized by the linear resonant frequency f_0, *vs.* the normalized DC bias force. The results from NODAS are compared to the results from ABAQUS simulation. The simulation error in the resonant frequencies are within 10% when using 1 beam element for the cantilever and 2 elements for the fixed-fixed beam. The errors are reduces to within 1% when using 5 elements for the cantilever and 10 elements for the fixed-fixed beam.
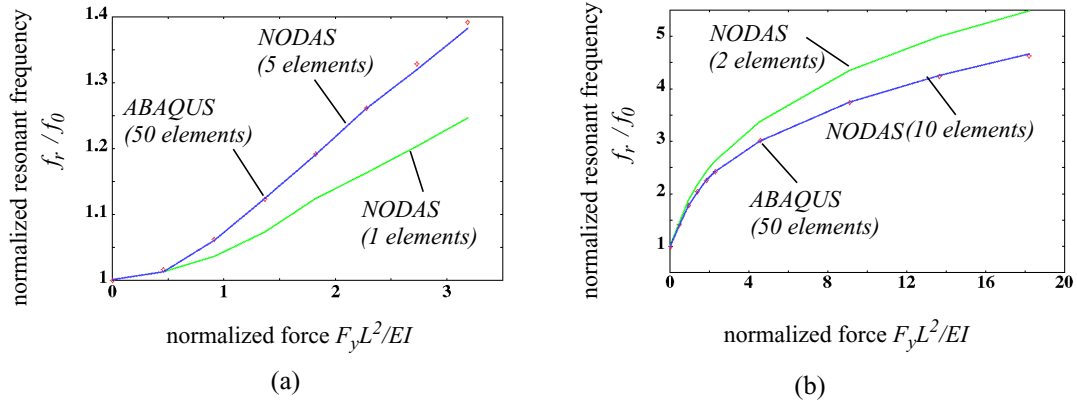
Figure 3.27: AC analysis of (a) a cantilever beam with Fy applied to the free end (100 μm long, 2 μm wide, 2 μm thick, *E*=165GPa) (b) a fixed-fixed beam, with *Fy* applied to the middle (100 μm long for each side, 2 μm wide, 2 μm thick, *E*=165GPa).

Similar simulation accuracies are obtained for crab-leg springs and folded-flexure springs. We see that the nonlinear beam model in NODAS is reusable and composable hence can be used as the building block for complicated systems. Using more beam elements gives better simulation accuracy, similar to finite element methods.

A folded-flexure resonator excited by a large sinusoidal force is simulated to verify existence of the Duffing effect, a well-known phenomenon caused by beam nonlinearity. The amplitude of the force is set to be such that the displacements are small at low frequencies but are large enough to cause the nonlinearity at frequencies near to the resonance. Figure 3.28 shows the steady-state envelope of the displacement magnitude
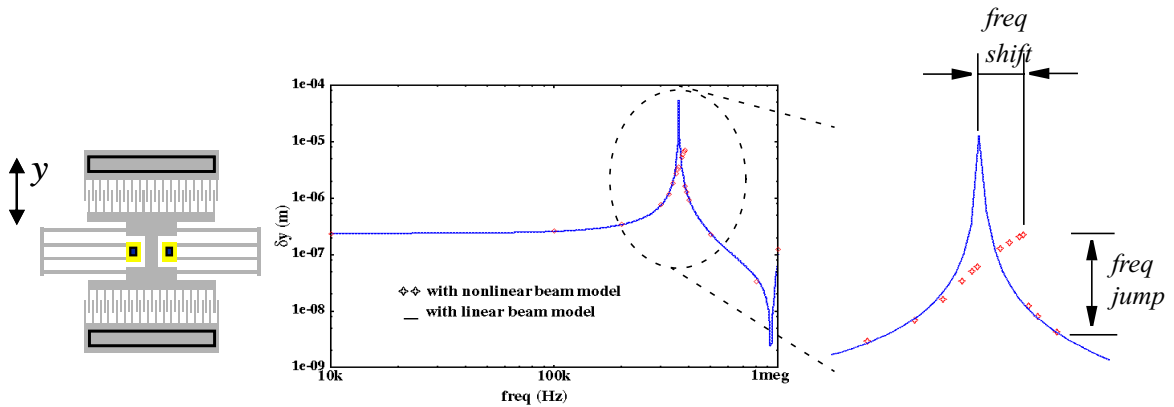
Figure 3.28: Duffing effect simulations using linear beam model and nonlinear beam model.

obtained from a series of transient analysis with varying excitation frequencies. Results given by the linear and nonlinear beam models are compared. The frequency shift due to the beam stiffening is clearly shown.

Summary of Beam Models

The linear beam model captures linear beam mechanics within small-deflection range. It covers axial compression/tension, in-plane and out-of-plane lateral bending, torsion about the longitudinal axis, inertia and viscous air damping. The nonlinear beam model captures the geometric nonlinearity caused by large axial stress and large geometric deflection. The accuracy of the models is verified through comparison to finite element analysis and analytical solutions. The excellent accuracy positively impacts the general applicability of composable design of MEMS using commercial electronic design tools.

Both models are reusable and composable. Using more beam elements to compose the physical beam provides a better fit to the exact shape of the displaced beam, thus gives a better simulation accuracy. For cases with analytical solution available, the simulation error can be analyzed by expanding the exact solution into Taylor Series then comparing to the polynomial fit in the beam models and estimating the percentage errors added by the higher order polynomials not covered by the cubic beam bending shape functions. For cases without analytical solutions, as the displacement shape function for the entire physical beam is formed by a stitching of the displacement shape functions for each individual constituent beam elements, the error can be estimated by examining the nodal displacements given by the simulation and calculating the error from the roughness (discontinuity in slopes) of the composite displacement shape function at the connection boundaries of beam elements.

How many beams should be used for a certain design to achieve accurate simulation results depends on the topology, the displacement range and the operation motion of the specific device. For example, if the beam has a displacement shape very similar to that of a

cantilever beam (for example: beams in crab-leg springs), then using one beam element is accurate enough for small displacement and using two beam elements is accurate enough for large deflections. If the beam is a structure similar to the fixed-fixed beam where the displacement is small but the internal axial stress is large, then nonlinear beam model must be used and two beam elements for each beam segment will be accurate enough. If the beam is operated at a motion which has very complicated displacement shape, then for each segment of the displacement shape which deviates from the cubic shape function, a beam element should be added in to better capture the exact shape. As NODAS simulations are not based on automatic remeshing, which helps adjusting the number of elements based on observation of the convergence in the simulation results [64][65], the user is expected to have a basic understanding of the physics involved in the structure to be simulated so that he or she can choose a trial number of beam element as a start of the simulation. He or she can then simply double the number of beams or change linear beams to nonlinear beams to see whether the changes make any difference to the results and repeat the process till a convergence is obtained. In the future, the NODAS simulation environment can also be improved by adding a feature to automatically report and make judgement on the displacement values at specified nodes and the internal axial stresses in specified beams so that the user can be notified when the number or the type of the beam elements need to be changed.

3.4 Plate Model

Beams are indispensable elements to suspended MEMS devices. The beams have finite mass and are hence subject to inertial forces, although their major function is as elastic suspensions. However, as the effective mass of the beam element is always coupled with the beam displacement motions, it's difficult to control the mass of the entire device and the compliance of the device simultaneously. Moreover, the mass of beams are usually very small due to the limit in sizes thus giving relatively high frequencies which are unwanted in
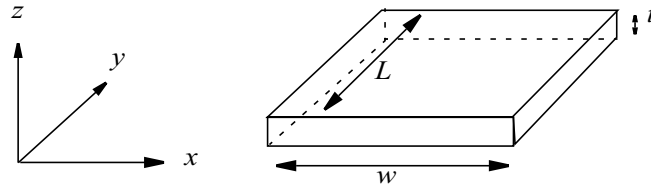
Figure 3.29: Basic geometric parameters for the plate element (length *L*, width *w* and thickness *t*)

many devices such as low-frequency resonators, accelerometers and gyroscopes. Therefore, many MEMS devices intentionally introduce to the device one or more *plates*, which act as dominating contributors to the mass of the entire device.

In NODAS, we currently model rectangular plates. The fundamental geometric parameters are length *L*, width *w* and thickness *t*, as shown in Figure 3.29. The layout orientation of the plate element is specified by Euler angles, same as in the beam model. The default orientation angles are $\alpha = 0$, $\beta = 0$ and $\gamma = 0$, that is, the width is aligned along x-axis, the length is aligned along y-axis, and the thickness is aligned along z-axis, as shown in Figure 3.29.

The stiffness of the plate element is determined by the aspect ratio and the material property. Unlike the beam element, which usually has $L \gg w$ and/or $L \gg t$, the plate element has a width and/or thickness which is comparable to the length *L*. To approximately estimate the strong dependence of stiffness on the aspect ratio, consider the in-plane lateral bending stiffness of a cantilever beam:

$$k_y = \frac{Etw^3}{4L^3} \tag{3.117}$$

(3.117) shows that k_y is proportional to the cubic of the aspect ratio *w/L*. If $w \approx L$, k_y will be 1000 times larger than that of a beam with $w = 0.1*L$. Therefore, structures with high aspect ratio of *w/L* or *t/L* should be modeled as a plate element, rather than a beam element. To accurately capture the elasticity of a plate, unlike in the beam element where the uniaxial

strain is proportional to the uniaxial stress by a constant of *E*, the Young's Modulus, the plane stresses in both *x* and *y* directions must be considered, A special case is the *biaxial plane stress*, in which the stress components in two directions equals each other and the biaxial strain is proportional to the biaxial stress by a constant of *E*/(1-ν), called *biaxial modulus* [66], where ν is the Poisson's ratio.

A plate element may have large stiffness in one, two or even all directions. According to the stiffness of the plate, we categorize the plate models into two types: rigid plates and elastic plates. In the rigid plate model, the plate is treated as a rigid body with infinite stiffness in every direction, transferring forces and moments with no deformation at all. In the elastic plate model, the plate can stretch in plane and bend out of plane, acting as a contributor of compliance to the entire system.

These two models will be presented in detail in the following sections, followed by simulation examples for accuracy verification.

3.4.1 Rigid Plate Model

3.4.1.1 User-specifiable Parameters

The geometric parameters *L* and *w* shown in Figure 3.29 are for solid plates. In fact, many suspended MEMS devices have perforated plates instead of solid ones. Figure 3.30 shows a typical crab-leg resonator with a perforated plate at the center. The perforations in
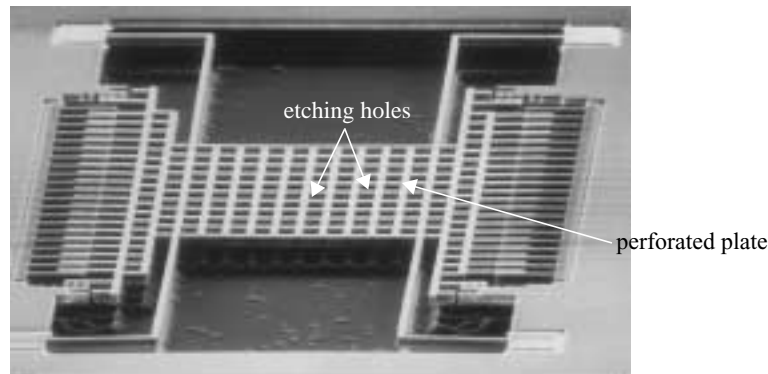
Figure 3.30: SEM picture of a typical crab-leg resonator with perforated plate (courtesy Xu Zhu)

the plates act as *etching holes* and are widely used to release large plates. The geometric parameters for the plate model are hence defined on basis of the perforated plate design and are illustrated in Figure 3.31.

The perforated plate is assumed to be composed of repetitive identical plate unit cells. Parameter *unitl* and *unitw* define the length and the width of the plate unit cell, respectively. The number of plate unit cells along x-axis is *unitnum_x* and the number of plate unit cells along y-axis is *unitnum_y*. The size of the entire plate is thus:

$$L = unitl \cdot unitnumy \tag{3.118}$$

$$w = unitw \cdot unitnumx \tag{3.119}$$

Parameter *fraction_holes* specifies the fraction of holes as:
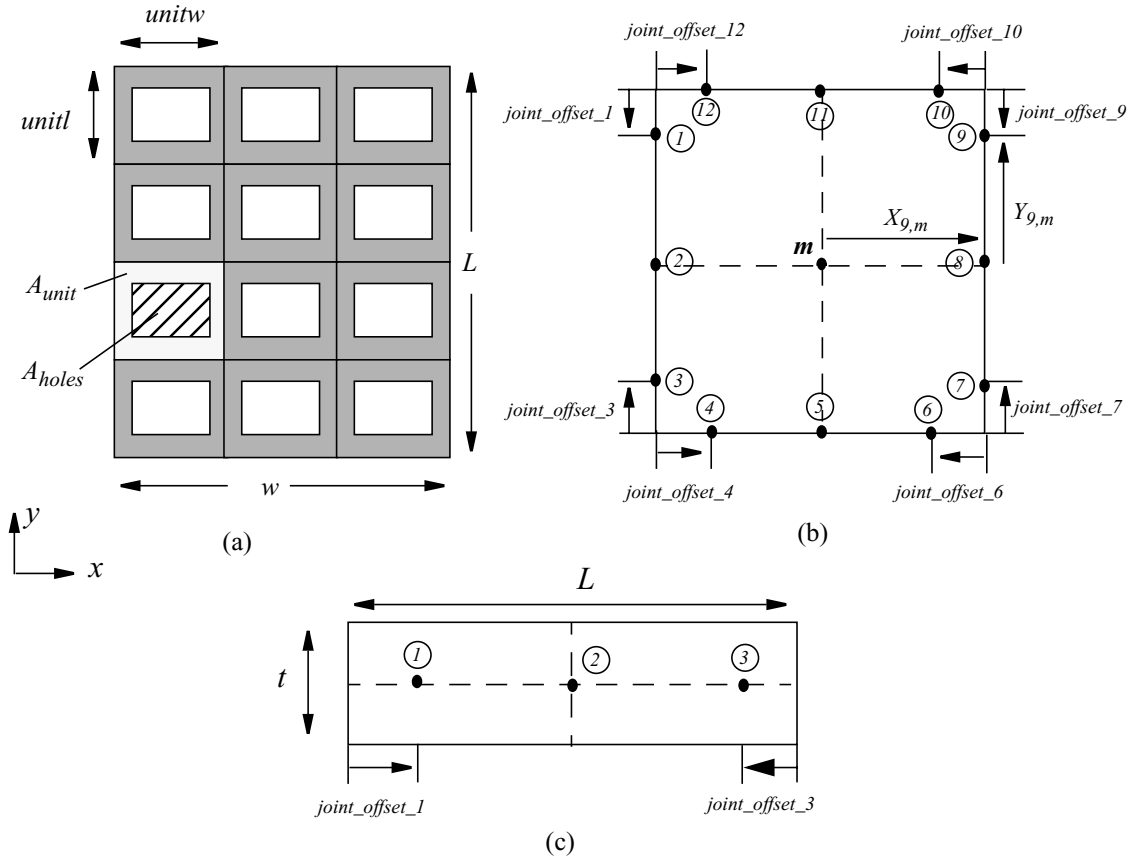
Figure 3.31: Geometric parameters for the perforated plate element (a) plate units (b) position of connection terminals (top view) (c) position of connection terminals (side view)

$$fractionholes = \frac{A_{holes}}{A_{unit}}, \tag{3.120}$$

where A_{holes} and A_{unit} are the areas of the shaded rectangles shown in Figure 3.31(a).

The rigid plate element has twelve connection terminals, numbered from 1 to 12 as in Figure 3.31(b). The terminals are defined in a counterclockwise order around the plate edges starting from the upper-left corner. Terminal 2,5,8 and 11 are fixed at the face-center of the side surfaces, as shown in Figure 3.31(c). Other terminals have default positions at the plate corners. Terminal 1 and 12 are at the upper-left corner, terminal 3 and 4 are at the lower-left corner, terminal 6 and 7 are at the lower-right corner and terminal 9 and 10 are at the upper-right corner. *Joint_offset* parameters are defined for the corner terminals, to be used for designs with connections that are offset laterally from the corners. Figure 3.31(b) defines the positions of the connection points with joint-offset. For instance, the terminal 1 is positioned with a move downwards with a positive value of *joint_offset_1* from the upper left corner of the plate.

Other user-specifiable parameters include the thickness, the mass density, the resistivity, the air gap between the plate bottom surface and the substrate, and the viscosity of the air.

3.4.1.2 Constraints in Rigid Plate Elements

A rigid plate elements is bounded by both constraints set by the force/moment balance equations and the constraints set by the fixed positions of the connection terminals on the rigid plate. These constraints are the physical equations implemented in the rigid plate model.

The static force and moment balance equations are as follows:

$$\sum_i F_{x,i} = 0, i = 1, ..., 12 \tag{3.121}$$

$$\sum_i F_{y,i} = 0, i = 1, ..., 12 \tag{3.122}$$

$$\sum_i F_{z,i} = 0, i = 1, \ldots, 12 \tag{3.123}$$

$$\sum_i M_{x,i} = 0, i = 1, \ldots, 12 \tag{3.124}$$

$$\sum_i M_{y,i} = 0, i = 1, \ldots, 12 \tag{3.125}$$

$$\sum_i M_{z,i} = 0, i = 1, \ldots, 12 \tag{3.126}$$

where $F_{x,i}$ is the force applied at the ith connection terminal in x-direction and $M_{x,i}$ is the moment applied at the ith connection terminal about x-axis. When dynamic equations are considered, inertial forces and damping forces should also be included.

The position constraints include the constraints on distances between the connections terminals along x, y and z directions and the constraints on the angular displacements of the connection terminals. Consider a rigid plate as shown in Figure 3.31, the distances between the connection terminals are fixed by the dimension of the plate and the joint_offset of each terminal, no matter how the plate moves and rotates in space. Due to the large number of connection terminals, the fixed distances between all these terminals are represented by the fixed distances of each terminal relative to the middle point of the plate, *m*, as illustrated in Figure 3.31 (b) and (c). For example, the distance between terminal 9 and the middle point along x, y and z directions are

$$X_{9,m} = \frac{w}{2} \tag{3.127}$$

$$Y_{9,m} = \frac{L}{2} - off_9 \tag{3.128}$$

$$Z_{9,m} = 0 \tag{3.129}$$

$X_{9,m}$ is positive as terminal 9 is located to the right of the middle point along x-axis, $Y_{9,m}$ is positive as terminal 9 is located to the upper side of the middle point along y-axis and $Z_{9,m}$ is zero because both terminal 9 is located on the same level as the middle point along z-axis. Position equations for other connection terminals are obtained in the same way, leading to altogether 36 position constraint equations.

As the plate is rigid, it's also subject to the constraint that the angular displacements at all the connection terminals are the same, i.e., the differences between the angular displacements are always zero. Again, these constraints are represented by the relations relative to the middle point, *m*:

$$\phi_{x,i} - \phi_{x,m} = 0, \ i \ = \ 1, \ \dots, \ 12 \tag{3.130}$$

$$\phi_{y,i} - \phi_{y,m} = 0, \ i \ = \ 1, \ \dots, \ 12 \tag{3.131}$$

$$\phi_{z,i} - \phi_{z,m} = 0, \ i \ = \ 1, \ \dots, \ 12 \tag{3.132}$$

where $\phi_{x,i}$ is the angular displacement at the ith connection terminal about *x*-axis and $\phi_{x,m}$ is the angular displacement at the middle point about *x*-axis.

Notice that although the distances between the connection terminals are fixed, the displacements of these terminals projected into the *x*, *y* and *z* directions are not. When the plate rotates, the displacements need to be obtained through coordinate transformations formed on basis of the rotation angles of the plate.

3.4.1.3 Coordinate Transformation

Similar to the beam model, there is a common frame of reference for every plate on the chip (called "chip frame"), and a frame of reference attached to each plate instance in the chip (called "local frame"). As shown in Figure 3.32, the plate forms a layout orientation angle of Θ relative to the *x*-direction of the chip frame. The *x*-axis of local frame is along the width of the plate and the *y*-axis of the local frame is along the length of the plate. The displaced frame is formed by a rotation from the local frame. The rotation angles, ϕ_x, ϕ_y, and ϕ_z, are determined by the angular displacements of the plate. The *x*-axis of displaced frame is always along the width of the displaced plate and the *y*-axis of the displaced frame is along the length of the displaced plate.

The transformation between the chip frame and the local frame is based on the layout orientation angles, α, β and γ. The transformation matrix, $[R]_{cl}$, is given in (3.81):
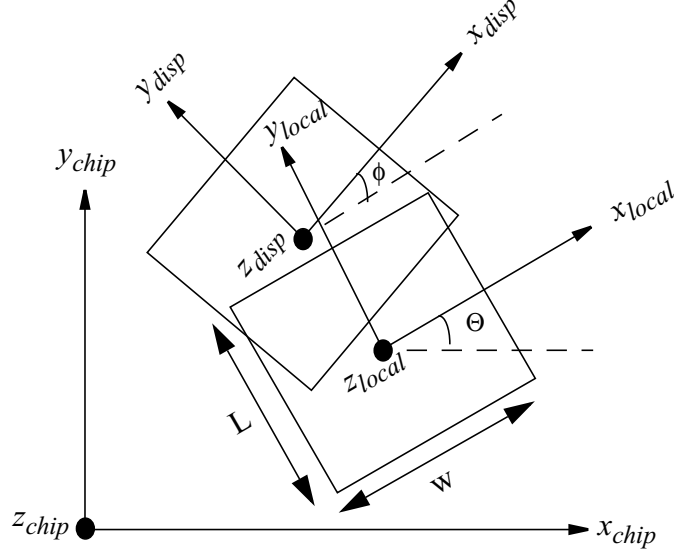
Figure 3.32: Frames of reference for the plate element

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_l = [R]_{cl} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_c = [R(\alpha, \beta, \gamma)]_{cl} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_c = \begin{bmatrix} l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \\ l_3 & m_3 & n_3 \end{bmatrix}_{cl} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_c \tag{3.133}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_c = [R]_{cl}^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_l \tag{3.134}$$

The transformation between the local frame and the displaced frame is formed based on the angular displacements in local frame. The angular displacements of the plate are first measured in chip frame then transformed into the local frame, based on (3.133):

$$\begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix}_l = [R]_{cl} \begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix}_c \tag{3.135}$$

Then the transformation matrix, $[R]_{ld}$, is obtained by replacing α, β and γ in $[R]_{cl}$ by the angular displacements in local frame, ϕ_{xl}, ϕ_{yl}, and ϕ_{zl}. The transformation between the local frame and the displaced frame are then formed based on $[R]_{ld}$:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_d = [R]_{ld} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_l = [R(\phi_{xl}, \phi_{yl}, \phi_{zl})]_{ld} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_l = \begin{bmatrix} l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \\ l_3 & m_3 & n_3 \end{bmatrix}_{ld} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_l \qquad (3.136)$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_l = [R]_{ld}^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_d \qquad (3.137)$$

3.4.1.4 Transfer of Forces, Moments and Position Constraints

One major function of the rigid plate model is to transfer the forces and moments applied at the connection points. As the plate is rigid, the forces and moments are calculated relative to the geometric center of the plate, *m*, as shown in Figure 3.31(b). The moment arm of each force on the plate is calculated according to the geometric position of the connection point where the force is applied. For example, in Figure 3.33, the length of the plate is along the *y*-axis, the width of the plate is along the *x*-axis and forces are applied in *x*, *y* and *z* directions at terminal 9 and terminal 12. The moments generated by these forces are:

$$M_{xm} = F_{z9}\left(\frac{L}{2} - off_9\right) + F_{z12}\frac{L}{2} \qquad (3.138)$$

$$M_{ym} = -F_{z9}\frac{w}{2} + F_{z12}\left(\frac{w}{2} - off_{12}\right) \qquad (3.139)$$

$$\text{and } M_{zm} = -F_{x9}\left(\frac{L}{2} - off_9\right) + F_{y9}\frac{w}{2} - F_{x12}\frac{L}{2} - F_{y12}\left(\frac{w}{2} - off_{12}\right) \qquad (3.140)$$

where $F_{\xi i}$ is the force in ξ-direction applied at the ith connection terminal and off_i is the joint_offset of the ith connection terminal. The moments due to forces applied at other connection terminals are derived in a similar way and added together for each direction.

For a plate element with rotation angles and/or non-zero layout orientation angles, the above equations need to be transformed into the local frame and the chip frame as the system equations are all formed in the chip frame.

Take terminal 9 as an example. The position of terminal 9 relative to the plate center
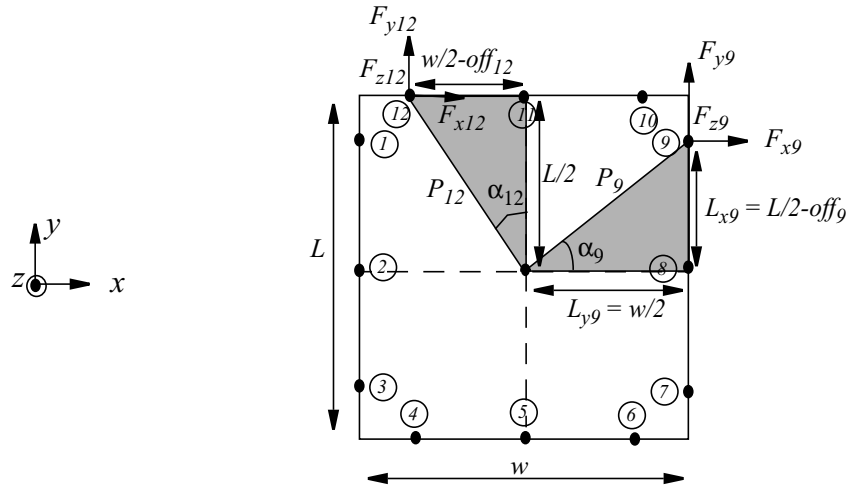
Figure 3.33: Moments generated by forces applied at connection terminals of the rigid plate

given in (3.127)-(3.129) is valid in the displaced frame. The difference between the displacement of terminal 9 and the displacement of the center point m in the local frame is thus:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{9,\,l} - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{m,\,l} = \begin{bmatrix} x_{9,\,m} \\ y_{9,\,m} \\ z_{9,\,m} \end{bmatrix}_l = [R]_{ld}^{-1} \begin{bmatrix} X_{9,\,m} \\ Y_{9,\,m} \\ Z_{9,\,m} \end{bmatrix} - \begin{bmatrix} X_{9,\,m} \\ Y_{9,\,m} \\ Z_{9,\,m} \end{bmatrix}
$$

(3.141)

where $[x\ y\ z]^{T}_{9,l}$ represents the displacements of terminal 9 in the local frame, $[x_{9,m}\ y_{9,m}\ z_{9,m}]^{T}_{l}$ represents the displacement difference between the displaced terminal 9 and point "m" in the local frame and $[X_{9,m}\ Y_{9,m}\ Z_{9,m}]^{T}$ represents the distance between terminal 9 and the middle point which is fixed as given in (3.127)-(3.129) and not changed by the rotation. The displacement difference in local frame is then transformed into the chip frame to serve as the rigid-body constraints used for solving the system matrix in the chip frame:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{9,\,c} - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{m,\,c} = [R]^{-1} \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{9,\,l} - \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{m,\,l} \right)
$$

(3.142)

As mentioned, for a rigid-body, the angular displacements at connection terminals are

all equal to the angular displacement at the center point. The difference in angular displacements is zero in every frame of reference. Thus, angular rigid-body constraints are directly specified in the chip frame as:

$$\begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix}_{i,c} - \begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix}_{m,c} = 0, i = 1, ..., 12 \tag{3.143}$$

For the convenience of moment arm calculations, the moments are first calculated in the local frame and then transformed back into the chip frame. The moment arms given in (3.138)-(3.140) are specified in the displaced frame, and are transformed into the local frame as:

$$\begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix}_{i,l} = [R]_{ld}^{-1} \begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix}_{i,d} , i = 1, ..., 12 \tag{3.144}$$

where $[L_x \, L_y \, L_z]_{i,d}$ are the moment arms about x, y and z axis in the displaced frame for the forces applied at the ith connection terminal, as illustrated in Figure 3.33. The forces are applied in the chip frame and thus they need to be transformed into the local frame to join the moment calculation:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_{i,l} = [R] \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_{i,c} , i = 1, ..., 12 \tag{3.145}$$

The moments in the local frame are then obtained based on the forces and moment arms in the local frame given by (3.144)-(3.145). The moments are finally transformed into the chip frame to join the system matrix analysis:

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}_{i,c} = [R]^{-1} \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}_{i,l} , i = 1, ..., 12 \tag{3.146}$$

3.4.1.5 Inertial Model

In addition to transferring moments, the plate elements are also major sources of inertia for suspended MEMS devices. For many designs, the plates remain rigid enough so that the sizing of the plates can be freely modified without affecting the stiffness of the entire device. Designers take advantage of this to independently achieve the desired resonant frequency.

The inertial forces and moments are calculated in the chip frame, based on the translational and angular accelerations of the plate as well as the distance from the plate center to the center of rotation. The Coriolis force and the centrifugal force are also captured in the model [70][71]. For an observer in the rotating chip frame, the effective force on the plate is given by

$$F_m = ma_r = ma_f - m\ddot{R}_f - m\omega \times (\omega \times r) - 2m\omega \times v_r \tag{3.147}$$

where m is the mass of the plate, v_r and a_r are the translational velocity and acceleration of the plate relative to the chip frame, a_f is the translational acceleration of the plate relative to the fixed global frame, $\ddot{R}_f$ is the translational acceleration of the chip relative to the fixed global frame, ω is the angular velocity of the chip frame, and r is distance from the plate to the chip layout origin. The first two terms are from Newton's equation, the term $-m\omega \times (\omega \times r)$ is for the centrifugal force and the last term is the Coriolis force.

3.4.1.6 Damping Model

The plate elements are major sources of damping as well. The damping in the plate includes the Couette damping, and is modeled in the chip frame as:

$$
\begin{bmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{bmatrix}_{B, c} = \begin{bmatrix} \dfrac{\mu(L+\delta)(w+\delta)}{\Delta}\dot{x}_m \\[2ex] \dfrac{\mu(w+\delta)(L+\delta)}{\Delta}\dot{y}_m \\[2ex] \dfrac{2\mu t(L+\delta+w+\delta)}{\Delta}\dot{z}_m \\[2ex] 2\dfrac{\mu}{\Delta}\dfrac{L^2+t^2}{12}\dot{\phi}_{xm} \\[2ex] 2\dfrac{\mu}{\Delta}\dfrac{w^2+t^2}{12}\dot{\phi}_{ym} \\[2ex] \dfrac{\mu}{\Delta}\dfrac{L^2+w^2}{12}\dot{\phi}_{zm} \end{bmatrix}_{B, c} \qquad (3.148)
$$

where δ is the edge factor which is 4 μm for damping surfaces of rectangular shape, Δ is the air gap between the plate bottom surface and the substrate and μ is the viscosity of the air. The translational velocities and angular velocities used in the calculation are those measured at the center of the plate. Similar to the beam model, a uniform air gap, Δ, is used for damping in every directions. In future improvement for the damping model, different air gaps should be used for different directions. In addition, this damping model is based on plates with solid surfaces thus is only an approximation for the real case when used for the simulation of perforated plates. More accurate damping model is available by including the squeeze film damping and Stokes flow [66] as well as the effects of plate etching holes [72].

3.4.1.7 Verilog-A Code Implementation

Code segment of the rigid plate model is given below to show the implementation of the model in Verilog-A.

```
'include "../../constants.h"
'include "../../discipline.h"
'include "../../process.h"
'include "../../design.h"
module plate3D_rigid(x1, v1, x2, v2, x3, v3, x4, v4, x5, v5, x6, v6, x7, v7, x8, v8, x9, v9, x10,
v10, x11, v11, x12, v12, phi, OMG_ext, ax_ext, t);
// definition of bus pins
inout [0:2] x1;
kinematic [0:2] x1;
```

```
    inout v1;
    electrical v1;
    inout [0:2] phi;
    rotational [0:2] phi;
    ...
    // external rotation rate
    inout [0:2] OMG_ext;
    rotational [0:2] OMG_ext;
    // external linear acceleration
    inout [0:2] ax_ext;
    kinematic  [0:2] ax_ext;
    inout t;
    thermal t;
    // definition of internal nodes for middle point of the plate
    kinematic xm, ym, zm;
    rotational phizm, phiym, phixm;
    electrical v_mid;
    // definition of user-specifiable parameters
    parameter real unitl = 'default_plate_unitl;
    parameter real unitw = 'default_plate_unitw;
    parameter real unitnum_x = 'default_plate_unitnum_x;
    parameter real unitnum_y = 'default_plate_unitnum_y;
    parameter real alpha = 0;
    parameter real beta  = 0;
    parameter real gamma  = 0;
    parameter real fraction_holes = 'default_plate_fraction_holes;
    parameter real joint_offset_1 = 'default_plate_joint_offset;
    ...
    parameter real joint_offset_12 = 'default_plate_joint_offset;
    parameter real Xc = 0;
    ...
    parameter real resistivity = 'default_resistivity;
    parameter real air_gap = 'default_air_gap;
    parameter real visc_air = 'default_visc_air;
    parameter real thickness = 'default_thickness;
    parameter real density = 'default_density;

    // definition of internal variables
    real cos_alpha, cos_beta, cos_gamma, sin_alpha, sin_beta, sin_gamma;
    real l1, m1, n1 ;
    real l2, m2, n2 ;
    real l3, m3, n3 ;
    real inv_l1, inv_m1, inv_n1 ;
    real inv_l2, inv_m2, inv_n2 ;
    real inv_l3, inv_m3, inv_n3 ;
    ...
```

```
real l_phixm, l_phiym, l_phizm;
real  fl_x1_xm, fl_x2_xm, fl_x3_xm, fl_x4_xm;
...
// beginning of analog block
analog begin

l= unitl*unitnum_y;
w= unitw*unitnum_x;
area = l*w*(1-fraction_holes);
resistance = resistivity * (l/w);
ms = density*thickness*area;
iiz = ms*(l*l+w*w)/12;
iiy = ms*(thickness*thickness+w*w)/12;
iix = ms*(thickness*thickness+l*l)/12;

dampx = visc_air/air_gap*((w+bloat)*(l+bloat));
dampy = visc_air/air_gap*((w+bloat)*(l+bloat));
dampz = visc_air/air_gap*((w+bloat)+(l+bloat))*2.0*thickness;
dampphix = visc_air/air_gap*(l*l+thickness*thickness)*2.0/12.0;
dampphiy = visc_air/air_gap*(w*w+thickness*thickness)*2.0/12.0;
dampphiz = visc_air/air_gap*(w*w+l*l)/12.0;

cos_alpha = cos(alpha /180*'M_PI);
cos_beta = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;
// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
```

```
        inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
        inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
        inv_n3 =  cos_alpha*cos_beta;


        // transfer angular displacements from chip fram to local frame
        l_phixm = 1e6*(l1*Theta(phixm) + m1*Theta(phiym) + n1*Theta(phizm));
        l_phiym = 1e6*(l2*Theta(phixm) + m2*Theta(phiym) + n2*Theta(phizm));
        l_phizm = 1e6*(l3*Theta(phixm) + m3*Theta(phiym) + n3*Theta(phizm));


        // transformation matrix from displaced frame to local frame, based on angular displacements
        cos_alpha_2 = cos(l_phixm);
        sin_alpha_2 = sin(l_phixm);
        ...
        euler_l1 =  cos_beta_2*cos_gamma_2;
        euler_l2 =  cos_beta_2*sin_gamma_2;
        euler_l3 = -sin_beta_2;
        euler_m1 =  sin_alpha_2*sin_beta_2*cos_gamma_2-cos_alpha_2*sin_gamma_2;
        euler_m2 =  sin_alpha_2*sin_beta_2*sin_gamma_2+cos_alpha_2*cos_gamma_2;
        euler_m3 =  sin_alpha_2*cos_beta_2;
        euler_n1 =  cos_alpha_2*sin_beta_2*cos_gamma_2+sin_alpha_2*sin_gamma_2;
        euler_n2 =  cos_alpha_2*sin_beta_2*sin_gamma_2-sin_alpha_2*cos_gamma_2;
        euler_n3 =  cos_alpha_2*cos_beta_2;


        // rigid-body constraints in the displaced frame
        x1_xm = -w/2;
        y1_ym = l/2-joint_offset_1;
        z1_zm = 0;
        ...
        x9_xm = w/2;
        y9_ym = l/2-joint_offset_9;
        z9_zm = 0;
        ...
        // rigid-body constraints in the local frame
        l_x9_xm = (euler_l1*x9_xm + euler_m1*y9_ym + euler_n1*z9_zm) - x9_xm;
        l_y9_ym = (euler_l2*x9_xm + euler_m2*y9_ym + euler_n2*z9_zm) - y9_ym;
        l_z9_zm = (euler_l3*x9_xm + euler_m3*y9_ym + euler_n3*z9_zm) - z9_zm;
        ...
        // inertial forces, centrifugal forces and Coriolis force are included in the part by Sita Iyer, please
refer to the Appendix for detail


        // damping force in local frame
        xg = l1*Pos(xm) + m1*Pos(ym) + n1*Pos(zm);
        yg = l2*Pos(xm) + m2*Pos(ym) + n2*Pos(zm);
        zg = l3*Pos(xm) + m3*Pos(ym) + n3*Pos(zm);
        fl_xm = dampx*ddt(xg);
        fl_ym = dampy*ddt(yg);
```

```
fl_zm = dampz*ddt(zg);

// damping forces transformed back to chip frame
fchip_xm = inv_l1*fl_xm + inv_m1*fl_ym + inv_n1*fl_zm;
fchip_ym = inv_l2*fl_xm + inv_m2*fl_ym + inv_n2*fl_zm;
fchip_zm = inv_l3*fl_xm + inv_m3*fl_ym + inv_n3*fl_zm;

// apply damping forces to the internal nodes for middle point of the plate
F(xm) <+ -fchip_xm;
F(ym) <+ -fchip_ym;
F(zm) <+ -fchip_zm;
...
// forces transfered from chip frame into local frame to calculate moments
fl_x9_xm = l1*F(x9[0],xm)+m1*F(x9[1],ym)+n1*F(x9[2],zm);
fl_y9_ym = l2*F(x9[0],xm)+m2*F(x9[1],ym)+n2*F(x9[2],zm);
fl_z9_zm = l3*F(x9[0],xm)+m3*F(x9[1],ym)+n3*F(x9[2],zm);
...
// moment arms in the displaced frame
L_x1_phiz = -(l/2-joint_offset_1);
L_y1_phiz = -w/2;
...
L_x9_phiz = -(l/2-joint_offset_9);
L_y9_phiz =  w/2;
...
L_x2_phiz =  0;
L_y2_phiz = -w/2;
...
L_z1_phix =  (l/2-joint_offset_1);
L_z1_phiy =  w/2;
...
L_z9_phix =  (l/2-joint_offset_9);
L_z9_phiy = -w/2;
...
L_z2_phix =  0;
L_z2_phiy =  w/2;
...
// moments calculated in local frame
tql_phizm =    fl_x1_xm*(L_x1_phiz*euler_l1+L_y1_phiz*euler_m1)
+ fl_y1_ym*(L_x1_phiz*euler_l2+L_y1_phiz*euler_m2)
+ fl_z1_zm*(L_x1_phiz*euler_l3+L_y1_phiz*euler_m3)
...   + fl_x9_xm*(L_x9_phiz*euler_l1+L_y9_phiz*euler_m1)
+ fl_y9_ym*(L_x9_phiz*euler_l2+L_y9_phiz*euler_m2)
+ fl_z9_zm*(L_x9_phiz*euler_l3+L_y9_phiz*euler_m3)
...   + fl_x2_xm*(L_x2_phiz*euler_l1+L_y2_phiz*euler_m1)
+ fl_y2_ym*(L_x2_phiz*euler_l2+L_y2_phiz*euler_m2)
+ fl_z2_zm*(L_x2_phiz*euler_l3+L_y2_phiz*euler_m3)
```

```
...
tql_phiym =    fl_x1_xm*(L_z1_phiy*euler_n1)
+ fl_y1_ym*(L_z1_phiy*euler_n2)
+ fl_z1_zm*(L_z1_phiy*euler_n3)
...+ fl_x9_xm*(L_z9_phiy*euler_n1)
+ fl_y9_ym*(L_z9_phiy*euler_n2)
+ fl_z9_zm*(L_z9_phiy*euler_n3)
...+ fl_x2_xm*(L_z2_phiy*euler_n1)
+ fl_y2_ym*(L_z2_phiy*euler_n2)
+ fl_z2_zm*(L_z2_phiy*euler_n3)

...
tql_phixm =    fl_x1_xm*(L_z1_phix*euler_n1)
+ fl_y1_ym*(L_z1_phix*euler_n2)
+ fl_z1_zm*(L_z1_phix*euler_n3)
...+ fl_x9_xm*(L_z9_phix*euler_n1)
+ fl_y9_ym*(L_z9_phix*euler_n2)
+ fl_z9_zm*(L_z9_phix*euler_n3)
...+ fl_x2_xm*(L_z2_phix*euler_n1)
+ fl_y2_ym*(L_z2_phix*euler_n2)
+ fl_z2_zm*(L_z2_phix*euler_n3)
...
// moments transfered from local frame back to chip frame
tqchip_phixm = inv_l1*tql_phixm + inv_m1*tql_phiym + inv_n1*tql_phizm;
tqchip_phiym = inv_l2*tql_phixm + inv_m2*tql_phiym + inv_n2*tql_phizm;
tqchip_phizm = inv_l3*tql_phixm + inv_m3*tql_phiym + inv_n3*tql_phizm;

// moments applied to the internal nodes for the middle point of plate
Tau(phizm) <+ -tqchip_phizm;
Tau(phiym) <+ -tqchip_phiym;
Tau(phixm) <+ -tqchip_phixm;

// damping moments applied to the internal nodes for the middle point of plate
Tau(phixm) <+ - (dampphix * 1e6*ddt(Theta(phixm)));
Tau(phiym) <+ - (dampphiy * 1e6*ddt(Theta(phiym)));
Tau(phizm) <+ - (dampphiz * 1e6*ddt(Theta(phizm)));

// rigid body constraints on displacements, transformed from local to chip frame
Pos(x9[0],xm) <+ (inv_l1*l_x9_xm + inv_m1*l_y9_ym + inv_n1*l_z9_zm);
Pos(x9[1],ym) <+ (inv_l2*l_x9_xm + inv_m2*l_y9_ym + inv_n2*l_z9_zm);
Pos(x9[2],zm) <+ (inv_l3*l_x9_xm + inv_m3*l_y9_ym + inv_n3*l_z9_zm);
...
// rigid body constraints on angular displacements, transformed from local to chip frame
Theta(phi[2],phizm) <+ 0 ;
Theta(phi[1],phiym) <+ 0 ;
Theta(phi[0],phixm) <+ 0 ;
...
```

```
// electrical equations
I(v1, v_mid) <+ V(v1, v_mid)/resistance;
...
end
endmodule
```

3.4.1.8 Verification

A crab-leg resonator, as shown in Figure 3.34(a), is simulated in NODAS using the rigid plate to model the shuttle mass. The resonant frequencies have good agreement with the results from finite element analysis in ABAQUS, as given in Figure 3.34(b).

Unlike the cases with beam model, where using more beam elements enables a better fit of the exact displacement shape thus gives better simulation accuracy, in the cases with rigid plate model, there is no such curve-fitting issues. The exact behavior of the plate is accurately captured in the model. Using more number of plate elements gives the same answer as using just one plate thus won't improve the accuracy and is not needed. The number of plate elements to be used in the simulation only depends on the shape of the proof mass and the complex of connections with other elements in the schematic.
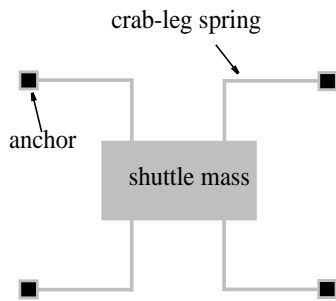
	NODAS	ABAQUS (C3D20)	error
f_x	350.75 kHz	353.2 kHz	0.7%
f_y	48.516 kHz	48.89 kHz	0.8%
f_z	39.355 kHz	40.18 kHz	2.1%
$f_{\theta x}$	97.724 kHz	101.3 kHz	3.5%
$f_{\theta y}$	64.565 kHz	67.03 kHz	3.7%
$f_{\theta z}$	331.13 kHz	331.6 kHz	0.1%

(a)

(b)

Figure 3.34: Simulation of a crab-leg resonator with rigid plate model (a) structure of crab-leg resonator (b) comparison of resonant frequencies

3.4.2 Elastic Plate Model

The rigid plate model given in the previous section can be used to model many suspended MEMS designs. However, if the plate is very thin or is made of a soft material, the stiffness of the plate will become non-negligible, so that the plate can no longer be treated as a rigid body. Therefore, an elastic plate model is included into NODAS, for two purposes. One, for designs where rigid plates are desired, the designer must be careful with the choice of geometric parameters and material properties of the plates to make them as stiff as possible. The elastic plate model can be used to aid in these designs. If the simulation with rigid plate model and with elastic plate model are very close to each other, it means that the plate can be treated as a rigid body, otherwise, the plates must be stiffened. Two, there exist many devices such as pressure sensors made of membranes, where compliant plates are employed on purpose. The elastic plate model is necessary for simulation of those cases.

The elastic plate model is developed based on the stiffness and mass matrices given by Przemieniecki [43]. The model captures the in-plane stretching and the out-of-plane bending of the plate.

3.4.2.1 In-Plane Stretching

The elastic plate has four connection terminals at the corners. For in-plane stretching, forces and translational displacements along x and y directions are considered, as shown in Figure 3.35.
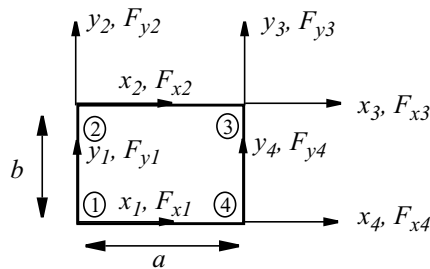
Figure 3.35: In-plane stretching of the elastic plate

There are two conventional methods of modeling the in-plane stiffness: the linear-edge-displacement assumption and the linear-stress assumption [43]. The former assumes that the boundary displacement functions vary linearly along the plate edge. This method guarantees the compatibility of the boundary displacement functions with the displacements on the adjacent elements. However, results from this method generally violates the stress equilibrium equations within the rectangular plate. The linear-stress assumption uses boundary stress functions that vary linearly along the plate edge. In this method, the stress equilibrium is guaranteed but the corresponding boundary displacement functions are not compatible with the adjacent elements. So neither of these two methods satisfies all the requirements. The linear-stress assumption is employed in NODAS elastic plate model because it has been proved that this method gives sufficient accuracy provided the size of the elements is small compared to the size of the stress variations [43].

Starting with the assumed linear stress distribution functions, the corresponding displacement distribution functions are derived. The unit-displacement theorem is then applied to determine the stiffness matrix [43]. The stiffness matrix $[k]_s$ for in-plane stretching of the elastic plate is given on page 105, where E is the Young's Modulus, ν is the Poisson ratio, t is the thickness, a is the width of the plate, b is the length of the plate, and β is the aspect ratio

$$\beta = \frac{b}{a} \tag{3.149}$$

The thinner the plate is and the softer the material is, the smaller is the stiffness of in-plane stretching.

The mass matrix for in-plane stretching is derived accordingly based on the displacement distribution function of the plate [43] and is given in (3.150), where ρ is the mass density of the plate and V is the volume of the plate:

stiffness matrix for in-plane stretching

$$[k]_s = \frac{Et}{12(1-v^2)} \begin{bmatrix}
(4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & & & & & & & \\
\frac{3}{2}(1+v) & (4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta & & & & \text{symmetric} & & \\
(2+v^2)\beta - \frac{3}{2}(1-v)\beta^{-1} & -\frac{3}{2}(1-3v) & (4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & & & & & \\
\frac{3}{2}(1-3v) & -(4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta & -\frac{3}{2}(1+v) & (4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta & & & & \\
-(2+v^2)\beta - \frac{3}{2}(1-v)\beta^{-1} & -\frac{3}{2}(1+v) & -(4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & -\frac{3}{2}(1-3v) & (4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & & & \\
-\frac{3}{2}(1+v) & -(2+v^2)\beta^{-1} - \frac{3}{2}(1-v)\beta & \frac{3}{2}(1-3v) & (2+v^2)\beta^{-1} - \frac{3}{2}(1-v)\beta & \frac{3}{2}(1+v) & (4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta & & \\
-(4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & \frac{3}{2}(1-3v) & -(2+v^2)\beta - \frac{3}{2}(1-v)\beta^{-1} & \frac{3}{2}(1+v) & (2+v^2)\beta - \frac{3}{2}(1-v)\beta^{-1} & -\frac{3}{2}(1-3v) & (4-v^2)\beta + \frac{3}{2}(1-v)\beta^{-1} & \\
-\frac{3}{2}(1-3v) & (2+v^2)\beta^{-1} - \frac{3}{2}(1-v)\beta & \frac{3}{2}(1+v) & -(2+v^2)\beta^{-1} - \frac{3}{2}(1-v)\beta & \frac{3}{2}(1-3v) & -(4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta & -\frac{3}{2}(1+v) & (4-v^2)\beta^{-1} + \frac{3}{2}(1-v)\beta
\end{bmatrix}$$

$$[m]_s = \frac{\rho V}{36} \begin{bmatrix} 4 & 2 & 1 & 2 & & & & \\ & 4 & 2 & 1 & 2 & & & \\ 2 & 4 & 2 & 1 & & & & \\ & 2 & 4 & 2 & & 1 & & \\ 1 & 2 & 4 & 2 & & & & \\ & 1 & 2 & 4 & & 2 & & \\ 2 & 1 & 2 & 4 & & & & \\ & 2 & 1 & 2 & 4 & & & \end{bmatrix} \tag{3.150}$$

3.4.2.2 Out-of-Plane Bending

The out-of-plane bending of the elastic plate includes the force in z-direction, the bending moments about x and y axes and the corresponding translational and angular displacements, as shown in Figure 3.36.

There are two different ways of modeling the displacement distribution of the plate in bending [43]. The first one is such that the boundary deflections on adjacent elements are compatible but the rotations of the element and thus the slopes at the boundaries are not compatible. The second one ensures both deflection and slope compatibility on adjacent elements [43], thus is taken to be implemented in NODAS.

Unit-displacement theorem is applied to derive the stiffness matrix. The entire stiffness matrix, $[k]_b$, is represented by submatrices:
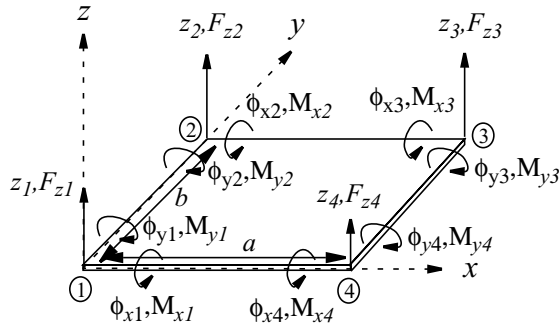
Figure 3.36: Out-of-plane bending of the 3D elastic plate

$$[k]_b = \begin{bmatrix} k_{I,I} & symmetric \\ k_{II,I} & k_{II,II} \end{bmatrix}$$

(3.151)

The submatices are given in page 108, page 109, and page 110, respectively.

The mass matrix for out-of-plane bending is derived accordingly based on the displacement distribution function with compatibility in both deflection and slope, as given below:

$$[m]_b = \frac{\rho abt}{176400} \begin{bmatrix}
24336 \\
3432b & 624b^2 \\
-3432a & -484ab & 624a^2 & & & & & & & symmetric \\
8424 & 2028b & -1188a & 24336 \\
-2028b & -468b^2 & 286ab & -3432b & 624b^2 \\
-1188a & -286ab & 216a^2 & -3432a & 484ab & 624a^2 \\
2916 & 702b & -702a & 8424 & -1188b & -2028a & 24336 \\
-702b & -162b^2 & 169ab & -1188b & 216b^2 & 286ab & -3432b & 624b^2 \\
702a & 169ab & -162a^2 & 2028a & -286ab & -468a^2 & 3432a & -484ab & 624a^2 \\
8424 & 1188b & -2028a & 2916 & -702b & -702a & 8424 & -2028b & 1188a & 24336 \\
1188b & 216b^2 & -286ab & 702b & -162b^2 & -169ab & 2028b & -468b^2 & 286ab & 3432b & 624b^2 \\
2028a & 286ab & -468a^2 & 702a & -169ab & -162a^2 & 1188a & -286ab & 216a^2 & 3432a & 484ab & 624a^2
\end{bmatrix}$$

3.4.2.3 Coordinate Transformation

Similar to the beam model and the rigid plate model, a coordinate transformation between the chip and local frames is needed to deal with non-zero layout orientation angles. Again, Euler angles α, β and γ are used to specify the orientation, following the same definition used in the other models. The transformation matrix is the same as the matrix given in (3.81).

The stiffness matrix and mass matrix given in previous sections are all formulated in the local frame of the elastic plate. The translational and rotational displacements are first measured in the chip frame, then transformed into the local frame where the spring, inertial and damping forces are calculated. Recall that the local frame is static (inertial). The forces

Stiffness matrix for rectangular plates in bending: submatrix $k_{I,I}$ based on compatible deflections

$$[k]_{I,I} = \frac{Et^3}{12(1-\nu^2)ab}\begin{bmatrix}
\frac{156}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & & & & & \\[4pt]
\left[\frac{22}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]b & \left[\frac{4}{35}\beta^2+\frac{52}{35}\beta^{-2}+\frac{8}{25}\right]b^2 & & \text{symmetric} & & \\[4pt]
-\left[\frac{78}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]a & -\left[\frac{11}{35}(\beta^2+\beta^{-2})+\frac{1}{50}(1+60\nu)\right]ab & \left[\frac{52}{35}\beta^2+\frac{4}{35}\beta^{-2}+\frac{8}{25}\right]a^2 & & & \\[4pt]
\frac{54}{35}\beta^2-\frac{156}{35}\beta^{-2}-\frac{72}{25} & \left(\frac{13}{35}\beta^2-\frac{78}{35}\beta^{-2}-\frac{6}{25}\right)b & \left[-\frac{27}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]a & \frac{156}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & & \\[4pt]
\left(-\frac{13}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}\right)b & \left[-\frac{3}{35}\beta^2+\frac{26}{35}\beta^{-2}-\frac{2}{25}\right]b^2 & \left[\frac{13}{70}\beta^2-\frac{11}{35}\beta^{-2}-\frac{1}{50}(1+5\nu)\right]ab & -\left[\frac{22}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]b & \left[\frac{4}{35}\beta^2+\frac{52}{35}\beta^{-2}+\frac{8}{25}\right]b^2 & \\[4pt]
\left[-\frac{27}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]a & \left[-\frac{13}{70}\beta^2+\frac{11}{35}\beta^{-2}+\frac{1}{50}(1+5\nu)\right]ab & \left[\frac{18}{35}\beta^2-\frac{4}{35}\beta^{-2}-\frac{8}{25}\right]a^2 & -\left[\frac{78}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5\nu)\right]a & \left[\frac{11}{35}(\beta^2+\beta^{-2})+\frac{1}{50}(1+60\nu)\right]ab & \left[\frac{52}{35}\beta^2+\frac{4}{35}\beta^{-2}+\frac{8}{25}\right]a^2
\end{bmatrix}$$

Stiffness matrix for rectangular plates in bending: submatrix $k_{II,I}$ based on compatible deflections

$$[k]_{II,I} = \frac{Et^3}{12(1-v^2)ab}
\begin{bmatrix}
-\frac{54}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & \left(-\frac{13}{35}\beta^2-\frac{27}{35}\beta^{-2}+\frac{6}{25}\right)b & \left(\frac{27}{35}\beta^2+\frac{13}{35}\beta^{-2}-\frac{6}{25}\right)a & -\frac{156}{35}\beta^2+\frac{54}{35}\beta^{-2}-\frac{72}{25} & \left[\frac{22}{35}\beta^2-\frac{27}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]b & \left(\frac{78}{35}\beta^2-\frac{13}{35}\beta^{-2}+\frac{6}{25}\right)a \\[2ex]
\left(\frac{13}{35}\beta^2+\frac{27}{35}\beta^{-2}-\frac{6}{25}\right)b & \left(\frac{3}{35}\beta^2+\frac{9}{35}\beta^{-2}+\frac{2}{25}\right)b^2 & \left[-\frac{13}{70}(\beta^2+\beta^{-2})+\frac{1}{50}\right]ab & \left[\frac{22}{35}\beta^2-\frac{27}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]b & \left(-\frac{4}{35}\beta^2+\frac{18}{35}\beta^{-2}-\frac{8}{25}\right)b^2 & \left[-\frac{11}{35}\beta^2+\frac{13}{70}\beta^{-2}-\frac{1}{50}(1+5v)\right]ab \\[2ex]
\left(-\frac{27}{35}\beta^2-\frac{13}{35}\beta^{-2}+\frac{6}{25}\right)a & \left[-\frac{13}{70}(\beta^2+\beta^{-2})+\frac{1}{50}\right]ab & \left(\frac{9}{35}\beta^2+\frac{3}{35}\beta^{-2}+\frac{2}{25}\right)a^2 & \left(-\frac{78}{35}\beta^2+\frac{13}{35}\beta^{-2}-\frac{6}{25}\right)a & \left[\frac{11}{35}\beta^2-\frac{13}{70}\beta^{-2}+\frac{1}{50}(1+5v)\right]ab & \left(\frac{26}{35}\beta^2-\frac{3}{35}\beta^{-2}-\frac{2}{25}\right)a^2 \\[2ex]
-\frac{156}{35}\beta^2+\frac{54}{35}\beta^{-2}-\frac{72}{25} & \left[-\frac{22}{35}\beta^2+\frac{27}{35}\beta^{-2}-\frac{6}{25}(1+5v)\right]b & \left(\frac{78}{35}\beta^2-\frac{13}{35}\beta^{-2}+\frac{6}{25}\right)a & -\frac{54}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & \left(\frac{13}{35}\beta^2+\frac{27}{35}\beta^{-2}-\frac{6}{25}\right)b & \left(\frac{27}{35}\beta^2+\frac{13}{35}\beta^{-2}-\frac{6}{25}\right)a \\[2ex]
\left[-\frac{22}{35}\beta^2+\frac{27}{35}\beta^{-2}-\frac{6}{25}(1+5v)\right]b & \left(-\frac{4}{35}\beta^2+\frac{18}{35}\beta^{-2}-\frac{8}{25}\right)b^2 & \left[\frac{11}{35}\beta^2-\frac{13}{70}\beta^{-2}+\frac{1}{50}(1+5v)\right]ab & \left(-\frac{13}{35}\beta^2-\frac{27}{35}\beta^{-2}+\frac{6}{25}\right)b & \left(\frac{3}{35}\beta^2+\frac{9}{35}\beta^{-2}+\frac{2}{25}\right)b^2 & \left[\frac{13}{70}(\beta^2+\beta^{-2})-\frac{1}{50}\right]ab \\[2ex]
\left(-\frac{78}{35}\beta^2+\frac{13}{35}\beta^{-2}-\frac{6}{25}\right)a & \left[-\frac{11}{35}\beta^2+\frac{13}{70}\beta^{-2}-\frac{1}{50}(1+5v)\right]ab & \left(\frac{26}{35}\beta^2-\frac{3}{35}\beta^{-2}-\frac{2}{25}\right)a^2 & \left(-\frac{27}{35}\beta^2-\frac{13}{35}\beta^{-2}+\frac{6}{25}\right)a & \left[\frac{13}{70}(\beta^2+\beta^{-2})-\frac{1}{50}\right]ab & \left(\frac{9}{35}\beta^2+\frac{3}{35}\beta^{-2}+\frac{2}{25}\right)a^2
\end{bmatrix}$$

Stiffness matrix for rectangular plates in bending: submatrix $k_{II,II}$ based on compatible deflections

$$[k]_{II,II} = \frac{Et^3}{12(1-v^2)ab} \begin{bmatrix} \frac{156}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & & & & & \\ -\left[\frac{22}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]b & \left(\frac{4}{35}\beta^2+\frac{52}{35}\beta^{-2}+\frac{8}{25}\right)b^2 & & \text{Symmetric} & & \\ \left[\frac{78}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]a & -\left[\frac{11}{35}(\beta^2+\beta^{-2})+\frac{1}{50}(1+60v)\right]ab & \left(\frac{52}{35}\beta^2+\frac{4}{35}\beta^{-2}+\frac{8}{25}\right)a^2 & & & \\ \frac{54}{35}\beta^2-\frac{156}{35}\beta^{-2}-\frac{72}{25} & \left(-\frac{13}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}\right)b & \left[\frac{27}{35}\beta^2-\frac{22}{35}\beta^{-2}-\frac{6}{25}(1+5v)\right]a & \frac{156}{35}(\beta^2+\beta^{-2})+\frac{72}{25} & & \\ \left(\frac{13}{35}\beta^2-\frac{78}{35}\beta^{-2}-\frac{6}{25}\right)b & \left(-\frac{3}{35}\beta^2+\frac{26}{35}\beta^{-2}-\frac{2}{25}\right)b^2 & \left[\frac{13}{70}\beta^2-\frac{11}{35}\beta^{-2}-\frac{1}{50}(1+5v)\right]ab & \left[\frac{22}{35}\beta^2+\frac{78}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]b & \left(\frac{4}{35}\beta^2+\frac{52}{35}\beta^{-2}+\frac{8}{25}\right)b^2 & \\ \left[\frac{27}{35}\beta^2-\frac{22}{35}\beta^{-2}-\frac{6}{25}(1+5v)\right]a & \left[-\frac{13}{70}\beta^2+\frac{11}{35}\beta^{-2}+\frac{1}{50}(1+5v)\right]ab & \left(\frac{18}{35}\beta^2-\frac{4}{35}\beta^{-2}-\frac{8}{25}\right)a^2 & \left[\frac{78}{35}\beta^2+\frac{22}{35}\beta^{-2}+\frac{6}{25}(1+5v)\right]a & \left[\frac{11}{35}(\beta^2+\beta^{-2})+\frac{1}{50}(1+60v)\right]ab & \left(\frac{52}{35}\beta^2+\frac{4}{35}\beta^{-2}+\frac{8}{25}\right)a^2 \end{bmatrix}$$

are summed up in the local frame and transformed back into the chip frame for system matrix analysis. The detailed transformation equations are the same as (3.80)-(3.82).

3.4.2.4 Other Issues about Elastic Plate Model

Other Degrees of Freedom

Rotation about *z*-axis is not included in the stiffness matrix and mass matrix for in-plane stretching and out-of-plane bending. For the completion of the model, corresponding equations for this degree of freedom, which are used in the rigid plate model, are employed in the elastic plate model as an approximation.

The reason why rotation about *z*-axis is not included in Przemieniecki's discussion is unclear. My conjecture is: 1. moment about *z*-axis is not the source of in-plane stretching and out-of-plane bending; 2. the mode shape of rotation about *z*-axis is different with the mode shapes for stretching and bending. Accurate analytical solution to this issue may exist but need further study.

Limited Connection Terminals

Unlike the rigid plate model, the elastic plate model has no connection terminals at the face centers and does not have joint-offset for connection terminals at the corners, *i.e.*, it only provides connection points at the four corners. The limited connection terminals limit schematic composition flexibility compared to rigid plate elements. The strain distributions are functions of the displacements at four corner connection terminals, derived with the assumption that only concentrated loads are applied, and only at the connection terminals. Terminals at locations other than the corners are not available. If a connection is needed at other positions, the physical plate has to be split into several pieces, each represented by an individual elastic plate element, then stitched together to form the entire physical plate. Perforated plates are not supported by the elastic plate model, as the holes in the plate will affect its elasticity.

Inertial and Damping Effects

Coriolis force and centrifugal force are not included either, because the formula used in the rigid plate model are valid for rigid bodies only. Theoretical solution to these forces can be derived based on the displacement distribution functions of the elastic plate, but the results are beyond the scope of this thesis. The damping of the elastic plate is currently modeled by introducing a damping matrix derived from the displacement distribution functions of the elastic plate. The damping matrix has a similar format as the mass matrix, same as the case in beam model.

How to Choose Between the Rigid Plate and the Elastic plate

There is no absolute criterion to choose between the rigid plate and the elastic plate. Which model to use depends on the aspect ratio and the material property of the specific plate element to be analyzed. For example, if the plate has $w \approx L$ and $t \approx L$, it's very stiff and can be treated as a rigid body. If the plate has $t << L$ and $t << w$, then using the elastic plate is necessary. For cases where there is no convincing prediction on the plate stiffness, the user can first run a simulation with the rigid plate elements, as the simulation with rigid plates is much faster than that with elastic plates. Then by changing the rigid plates into elastic plates, rerunning the simulation and comparing the simulation results, the user can determine if rigid plate model is adequate. If there is a big difference in the results, it means that the elasticity of the plate is non-negligible and the elastic plate element should be used to model this particular plate and other plates in the schematic with similar geometric and material properties.

3.4.2.5 Verilog-A Code Implementation

Code segment of the elastic plate model is given below to show the implementation of the model in Verilog-A.

```
// include header files
'include "../../constants.h"
```

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

module plate3D_elastic(x1, phi1, v1, x2, phi2, v2, x3, phi3, v3, x4, phi4, v4, t);
// definition of bus pins
inout [0:2] x1;
kinematic [0:2] x1;
inout [0:2] phi1;
rotational [0:2] phi1;
inout v1;
electrical v1;
...
// definition of user-specifiable parameters
parameter real unitl = 'default_plate_unitl;
parameter real unitw = 'default_plate_unitw;
parameter real unitnum_x = 'default_plate_unitnum_x;
parameter real unitnum_y = 'default_plate_unitnum_y;
parameter real alpha = 0;  // Euler angles
parameter real beta  = 0;
parameter real gamma  = 0;
parameter real E = 'default_E;  // Young's modulus of the elastic plate
parameter real thickness = 'default_thickness;
parameter real density = 'default_density;
parameter real Xc = 'default_plate_Xc;
parameter real Yc = 'default_plate_Yc;
parameter real resistivity = 'default_resistivity;

parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;

// definition of internal nodes
electrical vmid;
kinematic Vx1,Vy1,Vz1, Vx2,Vy2,Vz2, Vx3,Vy3,Vz3, Vx4,Vy4,Vz4;
rotational_omega Vphix1,Vphiy1,Vphiz1, Vphix2,Vphiy2,Vphiz2;
...
// definition of internal variables
real r;
real dampx,dampy,dampz, dampphix,dampphiy,dampphiz;
real cos_alpha, cos_beta, cos_gamma, sin_alpha, sin_beta, sin_gamma;
real l1, m1, n1 ;
real l2, m2, n2 ;
real l3, m3, n3 ;
...
real l_x1,l_y1,l_z1, l_x2,l_y2,l_z2, l_x3,l_y3,l_z3, l_x4,l_y4,l_z4;
real l_phix1,l_phiy1,l_phiz1, l_phix2,l_phiy2,l_phiz2;
```

113

```
real Fx1,Fy1,Fz1, Fx2,Fy2,Fz2, Fx3,Fy3,Fz3, Fx4,Fy4,Fz4;
real Tqx1,Tqy1,Tqx2,Tqy2,Tqx3,Tqy3,Tqx4,Tqy4;
real lwratio, nu,a,b,pp2,nn2,kb0;
// coefficients of stiffness matrix for out-of-plane bending
real kb11;
real kb21, kb22;
...
// coefficients of mass matrix for out-of-plane bending
real mb11;
real mb21, mb22;
...
// coefficients of stiffness matrix for in-plane stretching
real ks11;
real ks21,ks22;
...
// coefficients of mass matrix for in-plane stretching
real ms11;
real ms21,ms22;
...
// beginning of analog block
analog begin

b = unitl*unitnum_y;
a = unitw*unitnum_x;
r = resistivity * (b/a);

// Couette damping coefficients
dampx = visc_air/air_gap*((a+bloat)*(b+bloat));
dampy = visc_air/air_gap*((a+bloat)*(b+bloat));
dampz = visc_air/air_gap*((a+bloat)+(b+bloat))*2.0*thickness;
dampphix = visc_air/air_gap*(b*b+thickness*thickness)*2.0/12.0;
dampphiy = visc_air/air_gap*(a*a+thickness*thickness)*2.0/12.0;
dampphiz = visc_air/air_gap*(a*a+b*b)/12.0;

// [kb], stiffness matrix for out-of-plane bending
nu = 0.3;  // Possion_ratio;
lwratio = b/a;
pp2 = pow(lwratio,2);
nn2 = pow(lwratio,-2);
kb0 = E*thickness*thickness*thickness/(12.0*(1-nu*nu)*a*b);
kb11 = kb0*(156.0/35.0*(pp2+nn2)+72.0/25.0);
kb21 = kb0*( b*( 22.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
...
kb1212 = kb0*(a*a*(52.0/35.0*pp2+4.0/35.0*nn2+8.0/25.0));

// [mb], mass matrix for out-of-plane bending
```

```
mb0 = density*a*b*thickness;
mb11  =  24336.0/176400.0*mb0;
mb21  =  3432.0*b/176400.0*mb0;
...
mb1212 = 624.0*a*a/176400.0*mb0;

// [ks], stiffness matrix for out-of-plane bending
pp1 = pow(lwratio,1);
nn1 = pow(lwratio,-1);
ks0 = E*thickness/(12.0*(1.0-nu*nu));
//using linear-stress assumption
ks11 = ks0*( (4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks21 = ks0*( 3.0/2.0*(1.0+nu) );
...
ks88 = ks0*( (4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );

// [ms], mass matrix for in-plane stretching
// for x-axis related rows and columns, same matrix for y-axis
ms0 = density*a*b*thickness;
ms11=4.0/36.0*ms0;
ms31=2.0/36.0*ms0;
...
ms88=4.0/36.0*ms0;

// coordinate transformation matrix based on euler angles
cos_alpha = cos(alpha /180.0*'M_PI);
cos_beta = cos(beta /180.0*'M_PI);
cos_gamma = cos(gamma /180.0*'M_PI);
sin_alpha = sin(alpha /180.0*'M_PI);
sin_beta = sin(beta /180.0*'M_PI);
sin_gamma = sin(gamma /180.0*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
```

```
    inv_l3 = -sin_beta;
    inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
    inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
    inv_m3 =  sin_alpha*cos_beta;
    inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
    inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
    inv_n3 =  cos_alpha*cos_beta;


    // transform displacements from chip frame into local frame
    l_x1 = l1*Pos(x1[0]) + m1*Pos(x1[1]) + n1*Pos(x1[2]);
    l_y1 = l2*Pos(x1[0]) + m2*Pos(x1[1]) + n2*Pos(x1[2]);
    l_z1 = l3*Pos(x1[0]) + m3*Pos(x1[1]) + n3*Pos(x1[2]);
    ...
    l_phix1 = l1*Theta(phi1[0]) + m1*Theta(phi1[1]) + n1*Theta(phi1[2]);
    l_phiy1 = l2*Theta(phi1[0]) + m2*Theta(phi1[1]) + n2*Theta(phi1[2]);
    l_phiz1 = l3*Theta(phi1[0]) + m3*Theta(phi1[1]) + n3*Theta(phi1[2]);
    ...


    // Velocity in local frame
    Pos(Vx1) <+  ddt(l_x1);
    Pos(Vy1) <+  ddt(l_y1);
    Pos(Vz1) <+  ddt(l_z1);
    ...
    Omega(Vphix1) <+  ddt(l_phix1);
    Omega(Vphiy1) <+  ddt(l_phiy1);
    Omega(Vphiz1) <+  ddt(l_phiz1);


    // [F1]=[m][a]+[B][v], inertial and damping forces/moments in local frame for out-of-plane
bending
    fi_z1 = mb11*ddt(Pos(Vz1))+mb21*1e6*ddt(Omega(Vphix1))+mb31*1e6*ddt(Omega(Vphiy1))
    +mb41*ddt(Pos(Vz2))+mb51*1e6*ddt(Omega(Vphix2))+mb61*1e6*ddt(Omega(Vphiy2))
    +mb71*ddt(Pos(Vz3))+mb81*1e6*ddt(Omega(Vphix3))+mb91*1e6*ddt(Omega(Vphiy3))
    +mb101*ddt(Pos(Vz4))+mb111*1e6*ddt(Omega(Vphix4))+mb121*1e6*ddt(Omega(Vphiy4))
    +1.0/
mb0*(dampz*mb11*ddt(l_z1)+dampphix*mb21*1e6*ddt(l_phix1)+dampphiy*mb31*1e6*ddt(l_phiy1)
    +dampz*mb41*ddt(l_z2)+dampphix*mb51*1e6*ddt(l_phix2)+dampphiy*mb61*1e6*ddt(l_phiy2)
    +dampz*mb71*ddt(l_z3)+dampphix*mb81*1e6*ddt(l_phix3)+dampphiy*mb91*1e6*ddt(l_phiy3)

+dampz*mb101*ddt(l_z4)+dampphix*mb111*1e6*ddt(l_phix4)+dampphiy*mb121*1e6*ddt(l_phiy4))
;
    ...
    tqi_x1= mb21*ddt(Pos(Vz1))+mb22*1e6*ddt(Omega(Vphix1))+mb32*1e6*ddt(Omega(Vphiy1))
    +mb42*ddt(Pos(Vz2))+mb52*1e6*ddt(Omega(Vphix2))+mb62*1e6*ddt(Omega(Vphiy2))
    +mb72*ddt(Pos(Vz3))+mb82*1e6*ddt(Omega(Vphix3))+mb92*1e6*ddt(Omega(Vphiy3))
    +mb102*ddt(Pos(Vz4))+mb112*1e6*ddt(Omega(Vphix4))+mb122*1e6*ddt(Omega(Vphiy4))
    +1.0/
mb0*(mb21*ddt(l_z1)+dampphix*mb22*1e6*ddt(l_phix1)+dampphiy*mb32*1e6*ddt(l_phiy1)
```

```
    +dampz*mb42*ddt(l_z2)+dampphix*mb52*1e6*ddt(l_phix2)+dampphiy*mb62*1e6*ddt(l_phiy2)
    +dampz*mb72*ddt(l_z3)+dampphix*mb82*1e6*ddt(l_phix3)+dampphiy*mb92*1e6*ddt(l_phiy3)

+dampz*mb102*ddt(l_z4)+dampphix*mb112*1e6*ddt(l_phix4)+dampphiy*mb122*1e6*ddt(l_phiy4))
;

    tqi_y1= mb31*ddt(Pos(Vz1))+mb32*1e6*ddt(Omega(Vphix1))+mb33*1e6*ddt(Omega(Vphiy1))
    +mb43*ddt(Pos(Vz2))+mb53*1e6*ddt(Omega(Vphix2))+mb63*1e6*ddt(Omega(Vphiy2))
    +mb73*ddt(Pos(Vz3))+mb83*1e6*ddt(Omega(Vphix3))+mb93*1e6*ddt(Omega(Vphiy3))
    +mb103*ddt(Pos(Vz4))+mb113*1e6*ddt(Omega(Vphix4))+mb123*1e6*ddt(Omega(Vphiy4))
    +1.0/
mb0*(mb31*ddt(l_z1)+dampphix*mb32*1e6*ddt(l_phix1)+dampphiy*mb33*1e6*ddt(l_phiy1)
    +dampz*mb43*ddt(l_z2)+dampphix*mb53*1e6*ddt(l_phix2)+dampphiy*mb63*1e6*ddt(l_phiy2)
    +dampz*mb73*ddt(l_z3)+dampphix*mb83*1e6*ddt(l_phix3)+dampphiy*mb93*1e6*ddt(l_phiy3)

+dampz*mb103*ddt(l_z4)+dampphix*mb113*1e6*ddt(l_phix4)+dampphiy*mb123*1e6*ddt(l_phiy4))
;
    ...
    // [F2]=[k][x], spring forces and moments in local frame for out-of-plane bending
    Fz1 =   kb11*l_z1+kb21*1e6*l_phix1+kb31*1e6*l_phiy1
    +kb41*l_z2+kb51*1e6*l_phix2+kb61*1e6*l_phiy2
    +kb71*l_z3+kb81*1e6*l_phix3+kb91*1e6*l_phiy3
    +kb101*l_z4+kb111*1e6*l_phix4+kb121*1e6*l_phiy4;
    ...
    Tqx1=   kb21*l_z1+kb22*1e6*l_phix1+kb32*1e6*l_phiy1
    +kb42*l_z2+kb52*1e6*l_phix2+kb62*1e6*l_phiy2
    +kb72*l_z3+kb82*1e6*l_phix3+kb92*1e6*l_phiy3
    +kb102*l_z4+kb112*1e6*l_phix4+kb122*1e6*l_phiy4;

    Tqy1=   kb31*l_z1+kb32*1e6*l_phix1+kb33*1e6*l_phiy1
    +kb43*l_z2+kb53*1e6*l_phix2+kb63*1e6*l_phiy2
    +kb73*l_z3+kb83*1e6*l_phix3+kb93*1e6*l_phiy3
    +kb103*l_z4+kb113*1e6*l_phix4+kb123*1e6*l_phiy4;
    ...
    // [F1]=[m][a]+[B][v], inertial and damping forces/moments in local frame for in-plane stretching
    fi_x1 =  ms11*ddt(Pos(Vx1))+ms21*ddt(Pos(Vy1))
    +ms31*ddt(Pos(Vx2))+ms41*ddt(Pos(Vy2))
    +ms51*ddt(Pos(Vx3))+ms61*ddt(Pos(Vy3))
    +ms71*ddt(Pos(Vx4))+ms81*ddt(Pos(Vy4))
    +dampx*ddt(l_x1);

    fi_y1 =  ms21*ddt(Pos(Vx1))+ms22*ddt(Pos(Vy1))
    +ms32*ddt(Pos(Vx2))+ms42*ddt(Pos(Vy2))
    +ms52*ddt(Pos(Vx3))+ms62*ddt(Pos(Vy3))
    +ms72*ddt(Pos(Vx4))+ms82*ddt(Pos(Vy4))
    +dampy*ddt(l_y1);
    ...
```

// [F2]=[k][x], spring forces and moments in local frame for in-plane stretching
Fx1 = ks11*l_x1+ks21*l_y1+ks31*l_x2+ks41*l_y2
+ks51*l_x3+ks61*l_y3+ks71*l_x4+ks81*l_y4;

Fy1 = ks21*l_x1+ks22*l_y1+ks32*l_x2+ks42*l_y2
+ks52*l_x3+ks62*l_y3+ks72*l_x4+ks82*l_y4;
...
// spring forces/moments transformed from local frame back to chip frame
Fchipx1 = inv_l1*Fx1 + inv_m1*Fy1 + inv_n1*Fz1;
Fchipy1 = inv_l2*Fx1 + inv_m2*Fy1 + inv_n2*Fz1;
Fchipz1 = inv_l3*Fx1 + inv_m3*Fy1 + inv_n3*Fz1;
...
Tqchipx1 = inv_l1*Tqx1 + inv_m1*Tqy1 ;
Tqchipy1 = inv_l2*Tqx1 + inv_m2*Tqy1 ;
...
// inertial and damping forces/moments transformed from local frame back to chip frame
Fichipx1 = inv_l1*fi_x1 + inv_m1*fi_y1 + inv_n1*fi_z1;
Fichipy1 = inv_l2*fi_x1 + inv_m2*fi_y1 + inv_n2*fi_z1;
Fichipz1 = inv_l3*fi_x1 + inv_m3*fi_y1 + inv_n3*fi_z1;
...
Tqichipx1 = inv_l1*tqi_x1 + inv_m1*tqi_y1 ;
Tqichipy1 = inv_l2*tqi_x1 + inv_m2*tqi_y1 ;
...
// spring forces/moments applied to the ends of beam
F(x1[0]) <+ -Fchipx1;
F(x1[1]) <+ -Fchipy1;
F(x1[2]) <+ -Fchipz1;
...
Tau(phi1[0]) <+ -Tqchipx1;
Tau(phi1[1]) <+ -Tqchipy1;
...
// inertia & damping forces/moments applied to the ends of beam
F(x1[0]) <+ -Fichipx1;
F(x1[1]) <+ -Fichipy1;
F(x1[2]) <+ -Fichipz1;
...
Tau(phi1[0]) <+ -Tqichipx1;
Tau(phi1[1]) <+ -Tqichipy1;
...
// rigid-body approximation for the degree of freedom of twisting about z-axis
// internal variables related to the twisting about z-axis
rad = gamma* 'M_PI /180;
lngth_r = sqrt(pow(a/2,2)+pow(b/2,2));
arc_r = atan((b/2)/(a/2));
ii = density*thickness*b*a*(b*b+a*a)/12.0;
// phim is in Mega-radian

```
angmid = 1e6*Theta(phim);
phi_off_1_r = arc_r + (rad + angmid);
phi_off_2_r = arc_r - (rad + angmid);
// twisting moments about z-axis
Omega(angvmid) <+ ddt(angmid);
Tau(phim) <+ - (ii*ddt(Omega(angvmid))
        + dampphiz * Omega(angvmid)
+ lngth_r* ( sin(phi_off_1_r)*F(x1[0],x3[0])
+cos(phi_off_1_r)*F(x3[1],x1[1])
+sin(phi_off_2_r)*F(x4[0],x2[0])
+cos(phi_off_2_r)*F(x4[1],x2[1]) ));
// rigid-body contraints for twisting about z-axis
Theta(phi1[2],phim)  <+  0;
Theta(phi2[2],phim)  <+  0;
Theta(phi3[2],phim)  <+  0;
Theta(phi4[2],phim)  <+  0;

//*******equations for electrical***********
I(v1, vmid) <+ V(v1, vmid)/r ;
...
end
endmodule
```

3.4.2.6 Verification

The accuracy of the elastic plate model is tested by the simulation of a cantilever elastic plate, as shown in Figure 3.37 (a). The left edge of the plate is anchored. Terminal 3 and 4 on the right edge of the plate are connected to force sources to test the in-plane stretching and the out-of-plane bending. Figure 3.37(b) gives the simulation results of the resonant frequencies. Results from NODAS are compared to those from ABAQUS. The errors are within 10% when only one plate element is used, and the errors reduce to within 5% when four plate elements are used. We see that similar to the beam model, the elastic plate model is composable. Using more plate elements to compose the physical plate gives better simulation accuracy.

(a)

(b)

		ABAQUS	NODAS (1 plate element)	error	NODAS (4 plate elements)	error
bending	f_z	281.3 kHz	285.102 kHz	1.4%	283.465 kHz	0.8%
stretching	f_x	21.17 MHz	23.1739 MHz	9.5%	21.8776 MHz	3.3%
	f_y	8.817 MHz	9.33254 MHz	5.8%	9.23240 MHz	4.7%
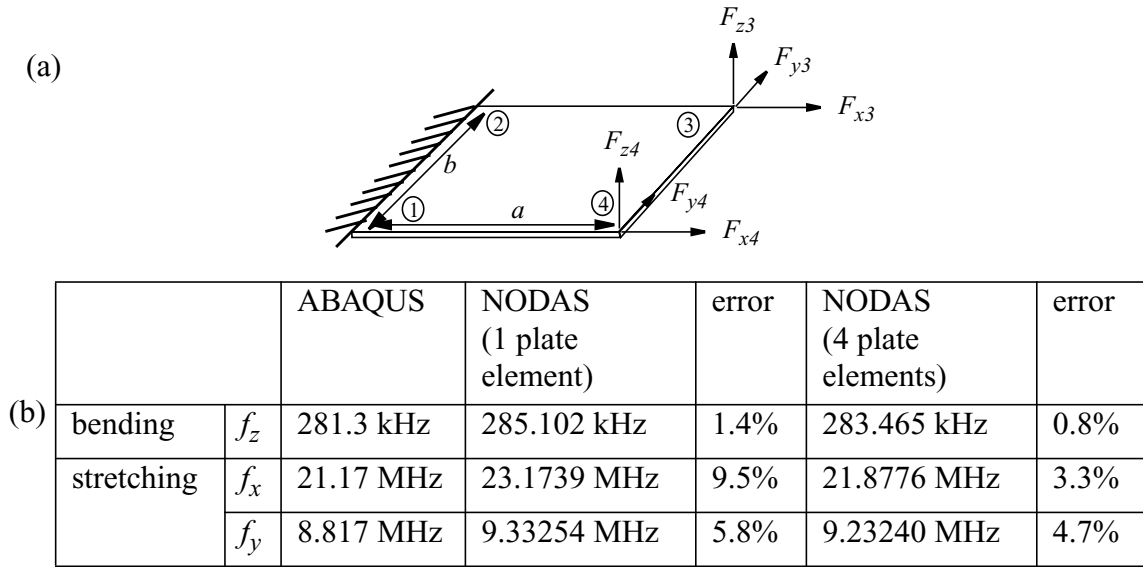
Figure 3.37: (a) A 3D cantilever elastic plate (b) comparison of NODAS and ABAQUS simulation results (a=100 μm, b=100 μm, t=2 μm, E=165GPa)

3.5 Electrostatic Gap Model

Electrostatic gaps are widely used in MEMS for capacitive actuation and sensing. Unlike the beam model and the plate model, which describes the behavior of the mechanical entities, the electrostatic gap model describes the interaction between mechanical elements - the electrodes which form the electrostatic field. Therefore, in order to analyze the electrostatic field, the gap model must "know" about the external elements, including their geometric shapes and sizes, material properties and relative positions. This information is either embedded in the model formulation or represented as parameters, as described in later sections.

The major function of the gap model is to capture the electrostatic effects between the electrodes. Figure 3.38 gives the algorithm structure of the gap model. The input displacements of the electrodes in the chip frame are first transformed to the local frame.

$[x]_{chip}$ V   beam

$$[T] \rightarrow [x]_{local} \rightarrow [F_e]_{local} = [F(x_{local}, V)] \rightarrow [T]^{-1} \longrightarrow [F_e]_{chip}$$
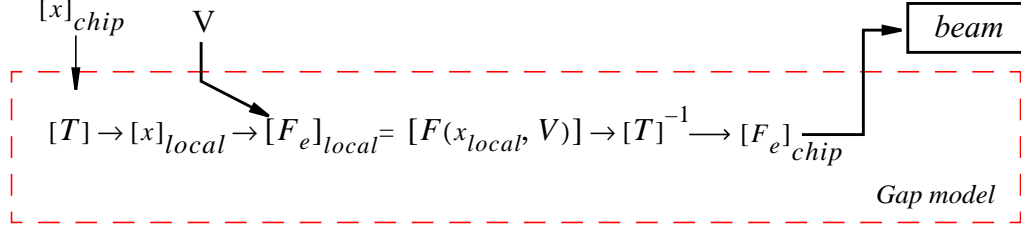
Gap model

Figure 3.38: Algorithmic structure of the electrostatic gap model

The transformed displacements, combined with the initial relative positions of the electrodes specified by parameters, are then used to estimate the instantaneous electrode positions hence the time-dependent shape of the electrostatic field. The electrostatic forces and moments are then calculated and fed back to the electrodes through the connection terminals for system analysis. When the gap is in normal operation, there exists a self-consistent solution between the mechanical domain and the electrostatic domain, which balances the spring restoring force and the electrostatic force. This self-consistent solution (equilibrium electrode positions) is obtained through the iterative system nodal analysis by the simulator.

Along with the increase of driving voltage, the electrodes move closer to each other and the gap decreases. The electrostatic force has a nonlinear relationship with the gap (for example, $F_e \propto 1/g^2$, for a 1D parallel plate gap), whereas the mechanical spring restoring force varies linearly with the gap ($F_k \propto -g$). Therefore, the system has two solutions: one at a larger gap, the other at a smaller gap. The former is a stable solution as the mechanical restoring forces increase with the descrease of the gap at a faster rate than the electrostatic force. The latter is a unstable solution. Once this gap is reached, the electrostatic force starts to increase at a much faster rate than the restoring force to cause the electrodes suddenly snap into each other. The driving voltage at this critical state is called the "pull-in voltage" [73]. The electrodes in contact will be a short-circuit, thus either an insulation layer that encloses the electrodes or mechanical "stops" are used to prevent the short-circuit. In

simulation, the snap-in has the potential to cause a non-physical situation where the gap could become negative valued. Therefore, in addition to the modeling of electrostatic field in normal operation, a contact model is also needed in the gap model to capture the contact physics.

When the electrodes move towards each other, squeeze film damping becomes the dominating air damping mechanism. Hence a squeeze film damping model is included in the gap model as well to help capture the dynamic behavior of the electrodes. The inertia of the electrodes are modeled by the elements forming the electrodes, not in the gap model.

In the following sections about the gap model, we first describe the interface of the gap model to the external elements, including the definitions of pins and parameters, then discuss the electrostatic, damping and contact models, and finally present several simulation examples to verify the model accuracy.

3.5.1 Definition of Pins and Parameters

The electrostatic gap model models the electrostatic field between two electrodes. There are infinite varieties of possible electrode types, varying from rectangular shape to irregular shape, from rigid body to compliant structures. Different types of electrodes forms different types of electrostatic fields with different dominating physical effects requiring different physical models. Due to the fact that the electrostatic field is highly dependent on the geometric shapes and relative positions of the electrodes, the current gap model in NODAS focuses on the gaps formed by two flexible beam electrodes, which is a common case used in many designs such as switches and electrostatic comb drives. In next section, we will prove that the beam bending shape functions are compatible with the rigid plate shape functions, thus the gap model based on beam electrodes are also capable to model the gaps formed by rigid plates, which allows a larger coverage of electrostatics including cases such as micro mirrors.
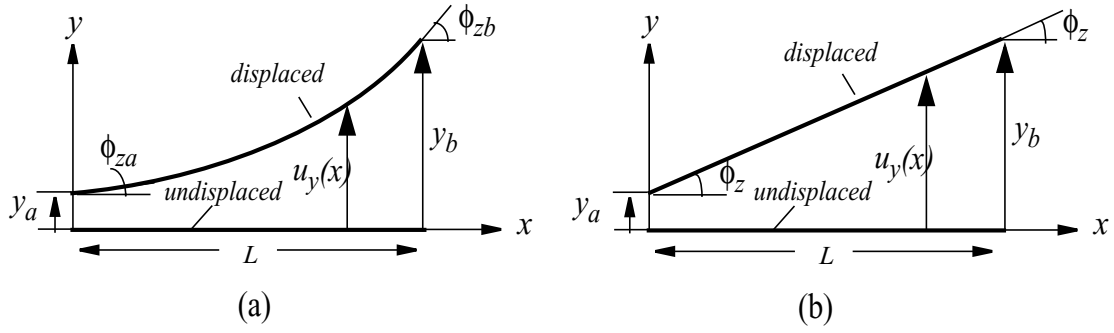
Figure 3.39: Shape functions of (a) compliant beam element and (b) rigid plate element.

3.5.1.1 Compatibility of Beam Bending Shape Functions

The following derivations shows that when the boundary conditions of rigid plates are applied, the beam bending shape functions given in previous sections about the beam model ((3.27)-(3.30)) simply reduce to the shape functions of the rigid plates, hence the gap model based on beam electrodes is also applicable to gaps formed by rigid plates. Substituting the beam bending shape functions into the displacement function, $u_y(x)$, we get:

$$
\begin{aligned}
u_y(x) &= f_{1y}(x)y_a + f_{2y}(x)\phi_{za} + f_{3y}(x)y_b + f_{4y}(x)\phi_{zb} \\
&= \left(1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3\right)y_a + \left(x - 2\left(\frac{x^2}{L}\right) + \left(\frac{x^3}{L^2}\right)\right)\phi_{za} \\
&\quad + \left(3\left(\frac{x}{L}\right)^2 - 2\left(\frac{x}{L}\right)^3\right)y_b + \left(-\left(\frac{x^2}{L}\right) + \left(\frac{x^3}{L^2}\right)\right)\phi_{zb}
\end{aligned}
\tag{3.152}
$$

where y_a and y_b are the translational displacements at beam ends a and b, and ϕ_{za} and ϕ_{zb} are the angular displacements at beam ends a and b, as shown in Figure 3.39(a).

If the beam is rigid and the translational and angular displacements are small, as illustrated in Figure 3.39(b), the boundary conditions are:

$$
\phi_{za} = \phi_{zb} = \phi_z
\tag{3.153}
$$

and

$$
y_b = y_a + L\phi_{za} = y_a + L\phi_z
\tag{3.154}
$$

Applying (3.153)-(3.154) to (3.152), the displacement function of the compliant beam elements becomes:

$$u_y(x) = \left(1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3\right)y_a + \left(x - 2\frac{x^2}{L} + \frac{x^3}{L^2}\right)\phi_z$$
$$+ \left(3\left(\frac{x}{L}\right)^2 - 2\left(\frac{x}{L}\right)^3\right)(y_a + L\phi_z) + \left(-\frac{x^2}{L} + \frac{x^3}{L^2}\right)\phi_z$$

(3.155)

Simplifying and rearranging the terms, we see that the beam bending displacement function reduces to

$$u_y(x) = y_a + x\phi_z,$$

(3.156)

which is exactly the displacement function for the rigid beam (or plate) shown in Figure 3.39(b). Therefore, the beam bending shape functions are fully compatible with that of rigid beams or plates. The gap model derived based on beam bending shape functions are applicable to rigid beams or plates as well.

3.5.1.2 Pin Definition

The gap model is designed to have four connection terminals, two for each of the opposing beam electrodes. Figure 3.40 gives the symbol view of the gap model and the connections to beam electrodes. Node *a* of the gap model is the connection terminal to node *a* of beam electrode 1, node *b* of the gap model is the connection terminal to node *b* of the beam electrode 1, node *c* of the gap model is the connection terminal to node *a* of the beam
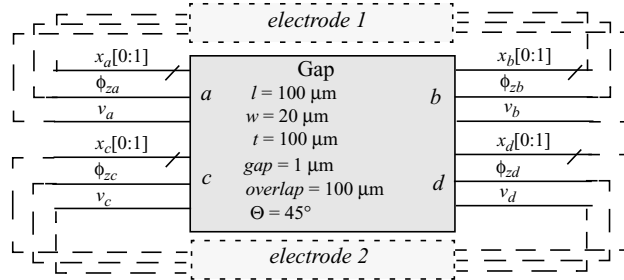
Figure 3.40: Gap model symbol and wiring to electrodes

electrode 2, and node *d* of the gap model is the connection terminal to node *b* of the beam electrode 2. Each of these connection terminals have a kinematic (translational) bus pin, a rotational pin and a voltage pin. Current gap model is a 2D gap model capturing the electrostatic field formed by beams bending in *x-y* plane and assuming the electrostatic field is uniform in *z*-direction, hence the translational pin have only two bus elements for *x* and *y* directions and the rotational pin is an individual pin for rotation about *z*-axis. This gap model deals with beam electrodes composed of a single conductor layer only, hence the voltage pin is just one individual electrical pin.

3.5.1.3 Parameter Definition

The user-specified parameters for the gap model include size parameters and position parameters, as illustrated in Figure 3.41, and material parameters (not shown in the figure).

Size parameters specify the sizes of the beam electrodes. L_1 and L_2 are the length of the upper beam (electrode 1) and the lower beam (electrode 2), respectively. w_1 and w_2 are the width of the beams. The two electrode are assumed to have the same thickness, *t*, for the 2D
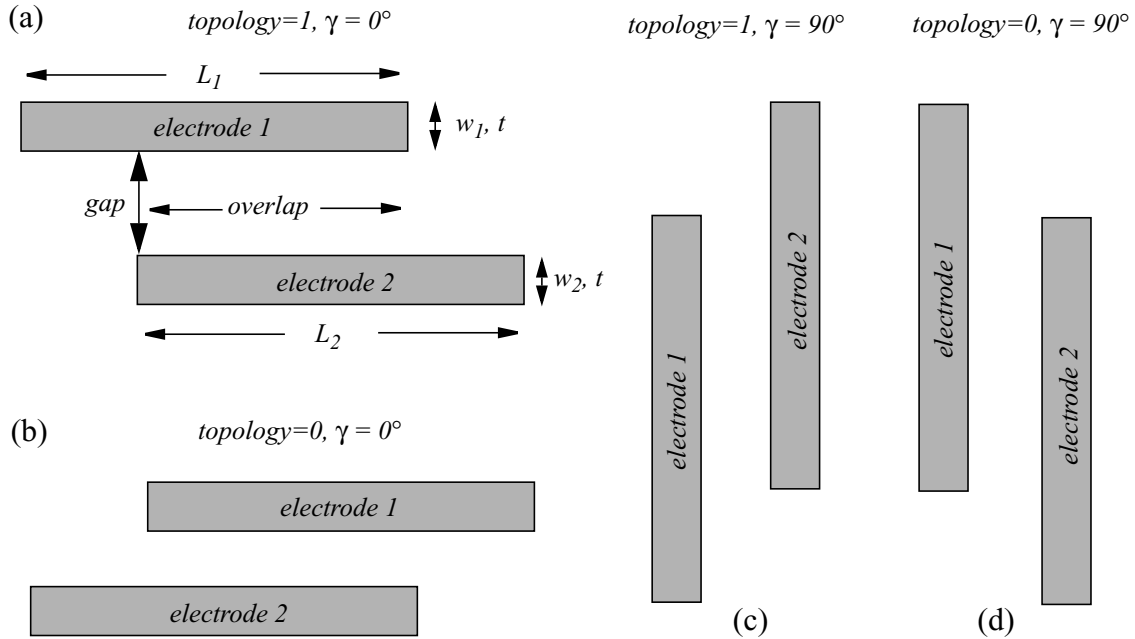
Figure 3.41: User-specified parameters for the gap model

gap model.

Material parameters specify the material properties of the electrodes and the gap, including the Young's modulus of the beams, the thickness of oxide insulator and the viscosity of the air. These parameters are used for the modeling of contact and air damping, which will be discussed in later sections.

Position parameters specify the initial layout orientation angle and the initial relative position of the electrodes. In the current gap model, beam electrodes are assumed to be positioned in parallel initially in the layout, although they are allowed to form a relative angle after being displaced. The initial layout orientation angle of the 2D gap is specified by the Euler angle, γ, as in the beam model and plate model. As shown in Figure 3.41, two horizontal beams in the layout form a gap with $\gamma = 0°$ and two vertical beams in the layout form a gap with $\gamma = 90°$. The gap is allowed to have arbitrary orientation angles, with the two beam electrodes having the same layout orientation angles in the chip frame.

Parameter *gap* is the initial gap between the electrodes. Parameter *overlap* is the initial overlap of the electrodes. Parameter *topology* specifies the relative position of the two electrodes. For horizontal beams as illustrated in Figure 3.41(a) and (b), *topology* = 1 when electrode 1 is to the left of electrode 2 and *topology* = 0 when electrode 1 is to the right of electrode 2. For vertical beams as illustrated in Figure 3.41(c) and (d), *topology* = 1 when electrode 1 is lower than electrode 2 and *topology* = 0 when electrode 1 is higher than electrode 2. The user must set the values of *topology* and γ consistently to ensure correct representation of the relative position of the electrodes. If the two beam electrodes have the same length and are completely overlapped with each other, then either topology 1 or 0 is valid. Cases with other types of electrode relative positions are not covered by current model and will be discuss in a later section about topology issues.

The topology information is used to determine which connection terminals are the reference points for calculation of the instantaneous overlap, the instantaneous shape of the bending beams and the distribution of electrostatic forces and moments. The parameter

topology only specifies the initial relative position of the electrodes. When the electrodes are displaced, the relative position may change and the polarity of the gap may change, depending on the magnitude and direction of the displacements. Thus the topology parameter is used in combination with the displacements to determine the actual instantaneous electrode positions.

The gap parameter *finger_number* is a "*multiplier*" used for simplified simulation of structures such as comb drives which are composed of multiple uniform gaps and beams. If each physical gap is represented by an individual gap element, then the parameter *finger_number* for every gap instances are all simply set to 1. If a single set of gap and beam elements is used to represent *n* sets of exactly the same combinations, which is the common way of schematic composition for comb drives with large number of fingers, then the parameter *finger_number* of the representative gap instance should be set to *n*. The mass of the uniform comb fingers are obtained by enlarging the mass of a single comb finger by *n* times. Compared to electrostatic comb drive model based on the assumption of rigid comb fingers [51][71], using the combination of gap and beam elements to compose the comb drive helps capturing the finite compliance and the distributed finite mass of comb fingers.

3.5.2 Electrostatic Model

3.5.2.1 Principle of Electrostatic Effect

Before deriving the 2D gap model based on bending beam electrodes, we first consider a simplified 1D gap model with rigid parallel-plate electrodes. This model forms a framework for the modeling of interactions between mechanical and electrostatic domains. It's a stepping stone to the development of the more complicated 2D gap model.

Consider a one-dimensional electrostatic gap for which the electrodes forming the gap are rigid plates in parallel, as shown in Figure 3.42. The initial gap is g_0. When the electrodes are displaced by an amount of y, the capacitance is
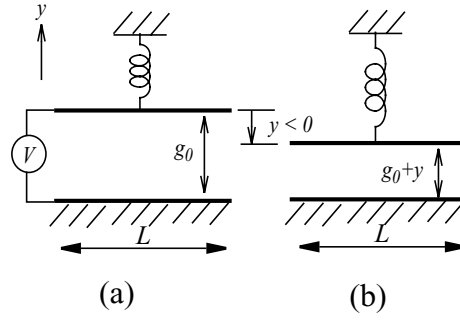
Figure 3.42: 1D Gap model with rigid parallel-plate electrodes (a) undisplaced (b) displaced

$$C = \frac{\varepsilon t L}{g_0 + y} \tag{3.157}$$

where ε_0 is the permittivity constant of the air, t is the thickness of the electrodes, L is the length of the overlapping region and y is the total electrode displacement. When a voltage is applied, an electrostatic force is generated and obtained as:

$$F_{ey} = \frac{d}{dy}\left(\frac{1}{2}C(y)V^2\right) = -\frac{1}{2}\frac{\varepsilon t L}{(g_0 + y)^2}V^2 \tag{3.158}$$

(3.158) indicates that the electrostatic force is an attractive force. Moreover, the electrostatic force has nonlinear relations with the displacement and the voltage.

In suspended MEMS, electrodes are normally suspended by springs, thus y represents the electrode displacement due to spring deformation. The displacement, y, is not an arbitrary value but a variable determined by the balance between the spring restoring force and the electrostatic force. The self-consistent solution between the mechanical and the electrostatic domains are obtained by solving the system matrix in the chip frame, as illustrated in the algorithmic structure of the gap model given in Figure 3.38.

The first generation of gap model is a 2D model based on rigid parallel plates [51][74]. It assumes the electrodes are rigid and only move in parallel with each other without any rotations. These assumptions are acceptable for cases such as comb drives with very stiff comb fingers, however is not acceptable for cases such as MEMS switches where the

electrodes are compliant and micro mirrors where the electrodes form an angle in between. Therefore, a second-generation 2D gap model with compliant beam electrodes is developed.

3.5.2.2 Electrostatic Field Between Bending Beams

Formation of Electrostatic Field

Figure 3.43 shows an improved gap model which captures the deflection of the flexible beam electrodes. The model is suitable for simulation of devices such as comb drives with non-rigid fingers and MEMS switches composed of beam electrodes. The model is also applicable for simulation of devices with rigid electrodes, as indicated by the compatibility between the shape functions of beams and rigid plates discussed in Section 3.5.1.1.

Cubic beam bending shape functions are employed to capture the non-uniform electrostatic field in the overlapping region. The gap varies along the bending beams:

$$g(x) \;=\; g_0 + u_y(x) \;=\; g_0 + u_{y1}(x) - u_{y2}(x) \tag{3.159}$$

where $u_{y1}(x)$ and $u_{y2}(x)$ are the displacement functions of the upper and lower beam electrodes, respectively, determined by the beam bending shape functions and the displacements at the beam ends. The beam bending shape functions are formed with the assumptions that the displacements are small and the variation in beam length from axial force is negligible.

$$u_{yi}(x) = f_{1y,i}(x)\, y_{ai} + f_{2y,i}(x)\, \phi_{zai} + f_{3y,i}(x)\, y_{bi} + f_{4y,i}(x)\, \phi_{zbi}$$

$f_{y,i}$: cubic shape functions

$y_i,\ \phi_{zi}$: displacements
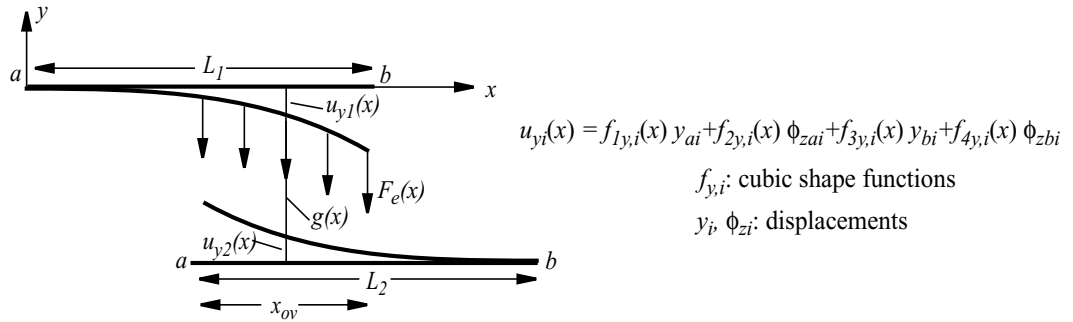
Figure 3.43: 2D Gap model based on beam bending shape functions

Shape Functions for Top and Bottom Beam Electrodes

As shown in Figure 3.43, we set the origin of the *x-y* coordinate system at node *a* of the top beam (electrode 1). The displacement function of the top beam is

$$u_{y1}(x) = f_{1y,\,1}(x)y_{a1} + f_{2y,\,1}(x)\phi_{za1} + f_{3y,\,1}(x)y_{b1} + f_{4y,\,1}(x)\phi_{zb1} \qquad (3.160)$$

where $f_{iy,\,1}$ are the beam bending shape functions of the top beam given as follows:

$$f_{1y,\,1}(x) = 1 - 3\left(\frac{x}{L_1}\right)^2 + 2\left(\frac{x}{L_1}\right)^3 \qquad (3.161)$$

$$f_{2y,\,1}(x) = x - 2\left(\frac{x^2}{L_1}\right) + \left(\frac{x^3}{L_1^2}\right) \qquad (3.162)$$

$$f_{3y,\,1}(x) = 3\left(\frac{x}{L_1}\right)^2 - 2\left(\frac{x}{L_1}\right)^3 \qquad (3.163)$$

$$f_{4y,\,1}(x) = -\left(\frac{x^2}{L_1}\right) + \left(\frac{x^3}{L_1^2}\right) \qquad (3.164)$$

These functions are the same as given in section 3.3.1.1. Notice that the two beam electrodes are allowed to have different length. The length of the top beam, L_1, is used in (3.161)-(3.164), as they only define $u_{y1}(x)$.

The displacement function of the bottom beam is

$$u_{y2}(x) = f_{1y,\,2}(x)y_{a2} + f_{2y,\,2}(x)\phi_{za2} + f_{3y,\,2}(x)y_{b2} + f_{4y,\,2}(x)\phi_{zb2} \qquad (3.165)$$

where $f_{iy,\,2}$ are the shape functions for the bottom beam. Notice that the basic beam bending shape functions given in (3.161)-(3.164) are derived by assuming that the coordinate origin is located at node *a* of beam 1. In order to model the bending of beam 2 in the same coordinate system, these basic shape functions must be modified before being applied to the bottom electrode. The boundary conditions for beam 2 are:

at $x = L_1 - x_{ov}$,

$$u_{y2}\big|_{x = L_1 - x_{ov}} = y_{a2} \qquad (3.166)$$

and

$$\frac{\partial u_{y2}}{\partial x}\bigg|_{x = L_1 - x_{ov}} = \phi_{za2} \tag{3.167}$$

at $x = L_1 + L_2 - x_{ov}$,

$$u_{y2}\big|_{x = L_1 + L_2 - x_{ov}} = y_{b2} \tag{3.168}$$

and

$$\frac{\partial u_{y2}}{\partial x}\bigg|_{x = L_1 + L_2 - x_{ov}} = \phi_{zb2} \tag{3.169}$$

where x_{ov} is the time-dependent overlap determined by the initial overlap, specified by the parameter *overlap*, and the displacements of electrodes in *x*-direction. The shape functions in (3.161)-(3.164) are transformed accordingly by being shifted along the *x*-axis by an amount of $L_1 - x_{ov}$ and by using the length of the bottom beam electrode, L_2, instead of L_1. The shape functions for the bottom beam electrode are:

$$f_{1y,2}(x) = 1 - 3\left(\frac{x - (L_1 - x_{ov})}{L_2}\right)^2 + 2\left(\frac{x - (L_1 - x_{ov})}{L_2}\right)^3 \tag{3.170}$$

$$f_{2y,2}(x) = (x - (L_1 - x_{ov})) - 2\left(\frac{(x - (L_1 - x_{ov}))^2}{L_2}\right) + \left(\frac{(x - (L_1 - x_{ov}))^3}{L_2^2}\right) \tag{3.171}$$

$$f_{3y,2}(x) = 3\left(\frac{x - (L_1 - x_{ov})}{L_2}\right)^2 - 2\left(\frac{x - (L_1 - x_{ov})}{L_2}\right)^3 \tag{3.172}$$

$$f_{4y,2}(x) = -\left(\frac{(x - (L_1 - x_{ov}))^2}{L_2}\right) + \left(\frac{(x - (L_1 - x_{ov}))^3}{L_2^2}\right) \tag{3.173}$$

Substitute (3.161)-(3.164) and (3.170)-(3.173) into (3.159), we obtain the gap varying along the bending beam electrodes.

Capacitance

The capacitance per unit length also varies along the bending beams:

$$\tilde{C}(x) = \frac{\varepsilon t}{g(x)} = \frac{\varepsilon t}{g_0 + u_{y1}(x) - u_{y2}(x)} = \frac{\varepsilon t}{g_0 + u_y(x)} \tag{3.174}$$

The total capacitance is obtained by integrating along x-direction in the overlapping region:

$$C(x) = \int_{(L_1 - x_{ov})}^{L_1} \tilde{C}(x)dx \tag{3.175}$$

Electrostatic Bending Forces and Moments

The electrostatic force per unit length in y-direction is obtained by taking the partial derivative of the energy about y, assuming the displaced beam electrodes move rigidly for a distance of $y(x)$:

$$\tilde{F}_{ey}(x) = \frac{\partial}{\partial u_y}\left(\frac{1}{2}\tilde{C}(x)V^2\right) = -\frac{1}{2}V^2\frac{\varepsilon t}{(g_0 + u_y(x))^2} \tag{3.176}$$

Similar to the damping force in the beam element, the electrostatic force is also a distributed force rather than a concentrated load, therefore, the electrostatic force is lumped based on beam bending shape functions into equivalent forces and moments applied at the ends of beam electrodes [54]. The equivalent loads for the upper and lower electrodes are:

$$F_{ey,a1} = \int_{(L_1 - x_{ov})}^{L_1} f_{1y,1}(x)\tilde{F}_{ey}(x)dx \tag{3.177}$$

$$M_{ez,a1} = \int_{(L_1 - x_{ov})}^{L_1} f_{2y,1}(x)\tilde{F}_{ey}(x)dx \tag{3.178}$$

$$F_{ey,b1} = \int_{(L_1 - x_{ov})}^{L_1} f_{3y,1}(x)\tilde{F}_{ey}(x)dx \tag{3.179}$$

$$M_{ez,b1} = \int_{(L_1 - x_{ov})}^{L_1} f_{4y,1}(x)\tilde{F}_{ey}(x)dx \tag{3.180}$$

$$F_{ey,a2} = \int_{(L_1 - x_{ov})}^{L_1} f_{1y,2}(x)\tilde{F}_{ey}(x)dx \tag{3.181}$$

$$M_{ez,a2} = \int_{(L_1 - x_{ov})}^{L_1} f_{2y,2}(x)\tilde{F}_{ey}(x)dx \tag{3.182}$$

$$F_{ey,b2} = \int_{(L_1 - x_{ov})}^{L_1} f_{3y,2}(x)\tilde{F}_{ey}(x)dx \tag{3.183}$$

$$M_{ez,b2} = \int_{(L_1 - x_{ov})}^{L_1} f_{4y,2}(x)\tilde{F}_{ey}(x)dx \tag{3.184}$$

As analytical solutions to these integrations are not available, 4th-order Taylor series

expansions of the integrand about the middle point of the overlapping region ($x_0 = (L_1 - x_{ov}/$

2)) are employed to obtain symbolic solutions to the integrations.

Electrostatic Force in x-direction

Unlike the distributed electrostatic force in *y*-direction, the electrostatic force in *x*-direction, F_{ex}, is treated as a concentrated load. The force is obtained by taking the partial derivative of the total force in *x*-direction about the overlap, x_{ov}, assuming the beam electrodes has an infinitesimal rigid move of x_{ov} along *x*-axis:

$$F_{ex} = \frac{\partial}{\partial x_{ov}}\left(\frac{1}{2}C(x)V^2\right) = \frac{1}{2}V^2\frac{\partial}{\partial x_{ov}}\left(\int_{(L_1 - x_{ov})}^{L_1}\frac{\varepsilon t}{g(x)}dx\right) \tag{3.185}$$

3.5.3 Damping Model

Since the electrodes move vertically toward each other, the squeeze-film damping in *y*-direction between the electrodes becomes dominant. The damping force is a distributed force along the bending beam electrodes. In current gap model, only the damping in the overlapping region is considered because the damping in the unengaged region is usually much smaller than that in the overlapping region. The squeeze-film damping force per unit length is modeled as:

$$\tilde{F}_{By}(x) = K_{By}(x_{ov}, t)\frac{\mu t^3}{g(x)^3}v(x) \tag{3.186}$$

where $K_{By}(x_{ov}, t)$ is the form factor introduced to account for the finite beam thickness, t, [83]. For beam electrode with $x_{ov} \gg t$, $K_{By}(x_{ov}, t) = 1$. μ is the viscosity of air, $g(x)$ is the gap along the bending beams as given in (3.159). $v(x)$ is the velocity along the bending beams defined as:

$$v(x) = \dot{u}_y(x) = \frac{d}{dt}(u_{y1}(x) - u_{y2}(x)) \tag{3.187}$$

A more accurate squeeze film damping model can be used instead of this one for improvement of the gap model in the future [57].

The damping force is then lumped to the beam ends into equivalent bending forces and moments, similarly to the electrostatic force in y-direction:

$$F_{By, a1} = \int_{(L_1 - x_{ov})}^{L_1} f_{1y, 1}(x)\tilde{F}_{By}(x)dx \tag{3.188}$$

$$M_{Bz, a1} = \int_{(L_1 - x_{ov})}^{L_1} f_{2y, 1}(x)\tilde{F}_{By}(x)dx \tag{3.189}$$

$$F_{By, b1} = \int_{(L_1 - x_{ov})}^{L_1} f_{3y, 1}(x)\tilde{F}_{By}(x)dx \tag{3.190}$$

$$M_{Bz, b1} = \int_{(L_1 - x_{ov})}^{L_1} f_{4y, 1}(x)\tilde{F}_{By}(x)dx \tag{3.191}$$

$$F_{By, a2} = \int_{(L_1 - x_{ov})}^{L_1} f_{1y, 2}(x)\tilde{F}_{By}(x)dx \tag{3.192}$$

$$M_{Bz, a2} = \int_{(L_1 - x_{ov})}^{L_1} f_{2y, 2}(x)\tilde{F}_{By}(x)dx \tag{3.193}$$

$$F_{By, b2} = \int_{(L_1 - x_{ov})}^{L_1} f_{3y, 2}(x)\tilde{F}_{By}(x)dx \tag{3.194}$$

$$M_{Bz, b2} = \int_{(L_1 - x_{ov})}^{L_1} f_{4y, 2}(x)\tilde{F}_{By}(x)dx \tag{3.195}$$

3.5.4 Rotated Parallel-Plate Approximation

As shown in Figure 3.43, the above derivations assume that the electrical field lines between the top and bottom beam electrodes are vertical straight lines, which is not physically true. Figure 3.44(a) shows the actual shape of the electrical field between the bending beam electrodes. The field lines must be perpendicular to the electrode surfaces therefore they are not vertical and not straight lines either when the beam electrodes are in bending. However, there is no analytical solution to the exact shape of the electrical field. The electrical field is thus approximated by a rotated parallel-plate electrical field in the overlapping region, which is a good approximation when the displacements are small.

As shown in Figure 3.44(b), the overlapping region of the beam electrodes are approximated by two rigid-plates forming an angle of ϕ_1 and ϕ_2, respectively, relative to the
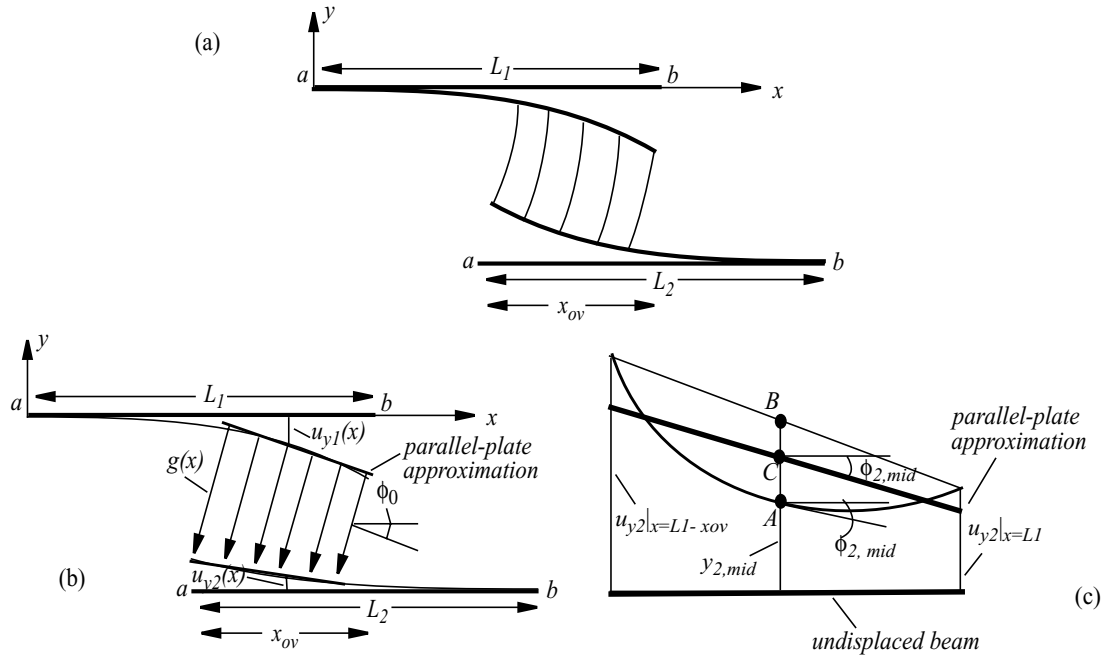
Figure 3.44: (a) Actual shape of electrical field lines of 2D gap with beam electrodes (b) rotated parallel-plate approximation in the overlapping region (c) formation of the approximation

x-axis. The electrical field lines between the rigid plates are approximated by straight lines, which form an angle of ϕ_0 relative to the *y*-axis. The rotation angles are defined as follows (taking the bottom beam electrode as an example), as shown in Figure 3.44(c):

1. Identify point *A*, the middle point of the part of bending beam electrode in the overlapping region. The *y*-displacement at point *A* is calculated from the displacement function $y_2(x)$ given in (3.165), which is derived based on the cubic beam bending shape functions:

$$y_{2, mid} = u_{y2}(x)\Big|_{x = L_1 - \frac{x_{ov}}{2}} \tag{3.196}$$

2. Draw a straight line across the bending beam in overlapping region. Find the center point *B*. The *y*-displacement at the left and right boundaries of the overlapping region are $u_{y2}|_{x=L1-xov}$ and $u_{y2}|_{x=L1}$, respectively.

3. Identify point *C*, the middle point on the straight line drawn from Point *A* to Point *B*.

Point C is the pivot for the rotation of the rigid plate used in the remaining steps.

4. Calculate the rotation angle $\phi_{2,\text{mid}}$, which is the angular displacement of the bottom beam electrode at the middle point A. $\phi_{2,mid}$ is obtained from the displacement function $u_{y2}(x)$ given in (3.165):

$$\phi_{2,\,mid} \;=\; \frac{d}{dx}u_{y2}(x)\bigg|_{x \,=\, L_1 \,-\, \frac{x_{ov}}{2}} \tag{3.197}$$

5. Draw a horizontal straight line in the overlapping region across the center point C, then rotate it by an angle of $\phi_{2,\text{mid}}$. Now we get the rigid plate approximation for the bottom beam.

6. The rigid plate approximation for the top beam electrode is obtained in the same way. $\phi_{1,mid}$, the counterpart for the top beam electrode, is obtained from the displacement function $u_{y1}(x)$ given in (3.160):

$$\phi_{1,\,mid} \;=\; \frac{d}{dx}u_{y1}(x)\bigg|_{x \,=\, L_1 \,-\, \frac{x_{ov}}{2}} \tag{3.198}$$

7. Calculate the rotation angle of the electrical field lines. The electrical field lines between the rigid plates are approximated by straight lines rotated from y-direction by an angle of ϕ_0, defined as:

$$\phi_0 \;=\; \frac{\phi_{1,\,mid} + \phi_{2,\,mid}}{2} \tag{3.199}$$

After obtaining the rigid plate approximation and electrical field lines, equations for the electrostatic field calculations are modified accordingly. First, the displacement functions $u_{y1}(x)$ and $u_{y2}(x)$ used for calculating the gap, capacitance and electrostatic forces are rewritten, based on the rigid-plate shape functions and the position of the center point C derived from the cubic beam bending shape functions:

$$u_{y1, rigid}(x) = \phi_{1, mid}\left[x - \left(L_1 - \frac{x_{ov}}{2}\right)\right] + \frac{y_{1, mid} + \frac{u_{y1}\big|_{x = L_1 - x_{ov}} + u_{y1}\big|_{x = L_1}}{2}}{2} \tag{3.200}$$

$$u_{y2, rigid}(x) = \phi_{2, mid}\left[x - \left(L_1 - \frac{x_{ov}}{2}\right)\right] + \frac{y_{2, mid} + \frac{u_{y2}\big|_{x = L_1 - x_{ov}} + u_{y2}\big|_{x = L_1}}{2}}{2} \tag{3.201}$$

The gap between the rigid plates is then rewritten as:

$$g(x) = (g_0 + u_{y1, rigid} - u_{y2, rigid})\cos\phi_0 \tag{3.202}$$

Notice that as shown in Figure 3.45, the original gap in the local frame is rotated by an angle of ϕ_0 into the rotated frame of reference defined along the rotated electrical field lines.

The capacitance is then calculated based on the modified gap equation following the same integration as given in (3.174)-(3.175). The electrostatic forces in x and y directions are first calculated in the rotated frame then transformed back to the local frame. The transformation matrix is:

$$\begin{bmatrix} F_{ex} \\ F_{ey} \end{bmatrix}_l = [R]^{-1}\begin{bmatrix} F_{ex} \\ F_{ey} \end{bmatrix}_r = \begin{bmatrix} \cos\phi_0 & -\sin\phi_0 \\ \sin\phi_0 & \cos\phi_0 \end{bmatrix}\begin{bmatrix} F_{ex} \\ F_{ey} \end{bmatrix}_r \tag{3.203}$$

3.5.5 Coordinate Transformation

Similar to the beam model and the plate model, coordinate transformations between the chip and local frames is also needed to deal with non-zero layout orientation angles. Again, Euler angle γ are used to specify the orientation, following the same definition as in the other models.
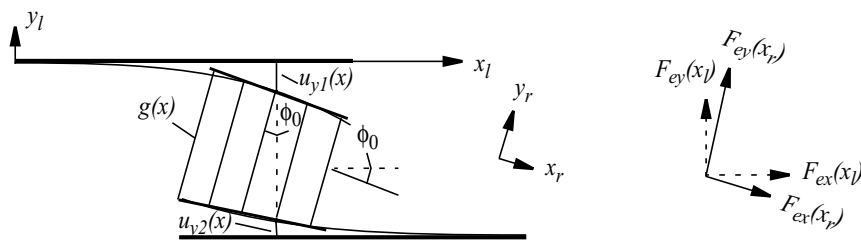
Figure 3.45: Rotated electrostatic field for the 2D gap with rigid parallel-plate approximation in the overlapping region

3.5.6 Contact Model

Another crucial physical phenomenon to be captured by the gap model is the snap-in effect due to the nonlinear relation between the displacement and the attractive electrostatic force. As discussed in Section 3.5.1.1, when the voltage is larger than the pull-in threshold, the electrostatic force becomes larger than the restoring bending force in the beams, leading to a physical instability. The two electrodes will suddenly snap into each other, causing potential convergence problems to the simulations. Hence it is very important to make a contact model to ensure the stable simulation even at the point of the physical instability.

As shown in Figure 3.46, in NODAS, the contact is modeled by a stiff spring deforming in the direction of snap-in formed by the contact region of the electrodes. The stiff spring arises from the existence of an oxide insulator coated on both electrodes. Normally, the electrodes are made of metals or silicon, which will oxidize in the air. The thickness of the oxide insulator is parameterized with a default setting of 20 nm.

When the gap is in normal operation (Figure 3.46 (a)), the two electrodes are separated from each other and the contact spring force is zero. When the electrodes start to abut against each other (Figure 3.46 (b)), a contact spring is formed by the parts of the electrodes
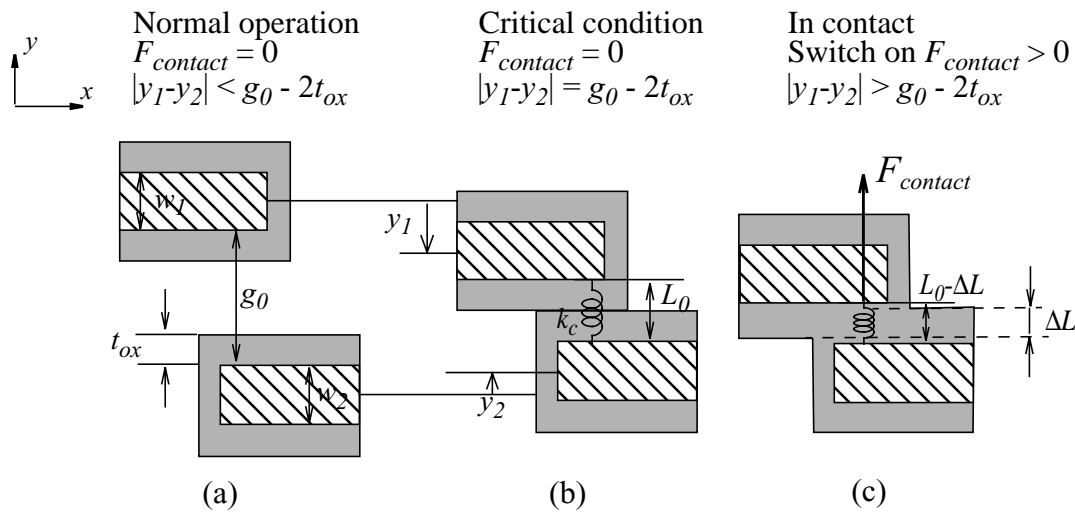
Figure 3.46: Contact model in the gap model

which are in contact. The contact spring deforms in the snap-in direction (y-direction). It is modeled as a linear spring with the spring constant defined as:

$$k_c = \frac{E \cdot x_c \cdot t}{L_0} \tag{3.204}$$

where E is the Young's Modulus of the electrodes, t is the thickness of the electrodes, x_c is the length of the contact region in x-direction, and L_0 is the original length of the contact spring which is around twice the thickness of the oxide insulator, t_{ox}. For bending beam electrodes, the contact length actually varies along with the bending process. It should be a function of the instantaneous contact point of the beam electrodes. However, as the beam behavior is lumped at the beam ends, only the displacements at terminal a and terminal b of the beam elements are available for the use in gap model. It's difficult to tell exactly which part of the bending beams are instantaneous in contact. Therefore, in current gap model, x_c is set to 1μm, as a rough approximation. Once the gap is in contact (Figure 3.46 (c)), the contact spring begins to deform and generate a large restoring force, $F_{contact}$, which works together with the restoring force given by the external elements to balance the large electrostatic force. The contact force is modeled as:

$$F_{contact} = k_c \cdot \Delta L = k_c \cdot (2t_{ox} - (g_0 + y_1 - y_2)), \tag{3.205}$$

where ΔL is the deformation of the contact spring.

3.5.7 Topology Issues

As mentioned in section 3.5.1.3, the parameter *topology* is used in combination with the node displacements to figure out the relative position of the beam electrodes. The relative position of the electrodes is critical for the definition of variables used in the calculation of gap, capacitance and electrostatic forces. In addition, as there exists a symmetry between topology 1 and 0, the topology parameter is used for switching of geometric and displacement variables so that the same set of equations can be used to model different topologies.

Definition of Topology

As presented in Section 3.5.1.3, current topology parameter in NODAS gap model has two possible values: 1 or 0. For a gap with horizontal beam electrodes, as illustrated in Figure 3.41(a) and (b), *topology* = 1 when beam 1 is to the left of beam2 and *topology* = 0 when beam 1 is to the right of beam2. Topology 1 is the case which is used for the physical equation derivations in previous sections.

Overlap Calculation

In the equations describing the electrostatic field, the instantaneous overlap, x_{ov}, is one of the key variables. x_{ov} is obtained based on the topology, the initial overlap and the displacements of the electrodes. First, we need to define the reference points. The reference points are the connection terminals of the beam electrodes which define boundaries of the overlapping region. The displacements at the reference points are the variables used in the calculation of instantaneous overlap, x_{ov}. As shown in Figure 3.47, for gaps with *topology* = 1, terminal *b* and *c* are the reference points, while for gaps with *topology* = 0, terminal *a* and *d* are the reference points. Relative displacement of the electrodes are calculated based on the displacements measured at the reference points:
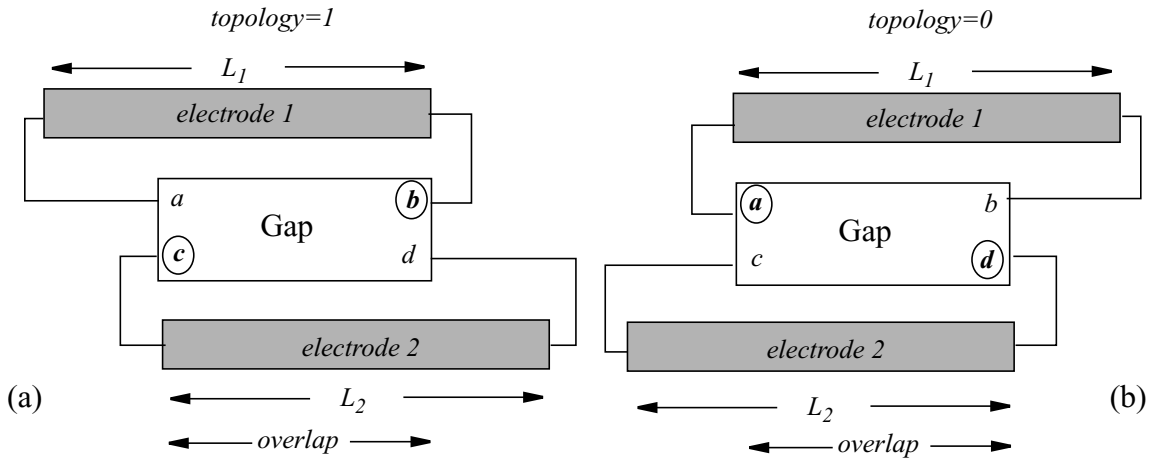
Figure 3.47: Reference points (circled connection terminals) for gaps with different topologies.

$$dx = topo \cdot (x_b - x_c) + (1 - topo) \cdot (x_d - x_a) \tag{3.206}$$

where *topo* represents the parameter *topology*. As *topology* is an integer parameter which is either 1 or 0, only one of the two terms in (3.206) is "switched-on" for certain topology.

The relative displacement, *dx*, is then combined with the initial overlap (parameter *overlap*) to figure out the instantaneous overlap, x_{ov}, as shown in Figure 3.48. Figure 3.48 illustrated the possible cases with different amount and direction of displacements in *x*-direction for a gap with *topology* = 1, $L_1 < L_2$ and electrode 2 fixed. x_c is hence 0 and *dx* = x_b. The overlap calculation for *topology* 0 can be illustrated in the similar way.

Examine Figure 3.48, we see that in Case I-III, the topology of the gap remains unchanged. The overlap increases, decreases or goes to zero according to the relative displacement of the electrodes in *x*-direction. Equations for the instantaneous overlap are different in different cases. In Cases IV-VI, the actual topology of the gap changes from 1 to 0 as indicated in the figure, while the value of the static parameter *topology* is still 1. The reference points should be switched to those for *topology* 0 accordingly, however, they are fixed by the value of the parameter *topology*. This indicates that the current way of topology definition needs to be improved.

This problem also has an adverse effect on the composibility of the gap model. Figure 3.49 illustrates several cases as examples of the composibility problems caused by the current definition of topology. Figure 3.49 (a) is a undisplaced beam with one gap element which has a topology of 1. Figure 3.49 (b) is a trial case to compose the physical gap with two gap elements (*A* and *B*), both have a topology of 1. However, in this case, part of the overlapping region between these two gaps, indicated by zone *C*, is not covered, leading to partial missing of electrical field and thus inaccurate results. Moreover, as shown in Figure 3.49 (c), if the top beam has a displacement of *dx* along the positive *x*-direction, gap *A* and *B* will get into a topology as given in Case IV in Figure 3.48, which is not covered by the current gap model. Figure 3.49 (d) gives another way of composing the physical gap with
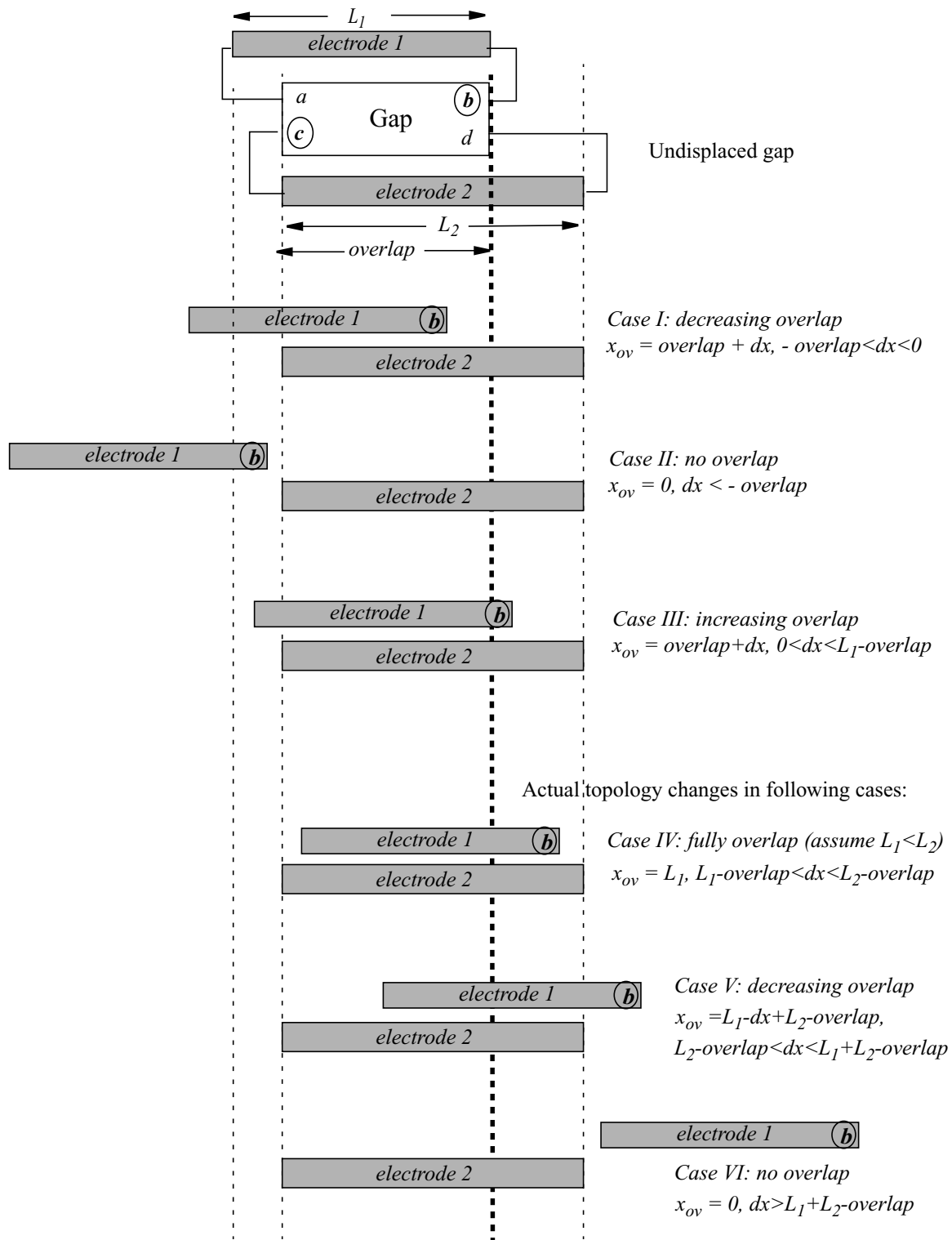
Figure 3.48: Different cases for overlap calculation (*topology* = 1, $L_1 < L_2$, and electrode 2 is fixed
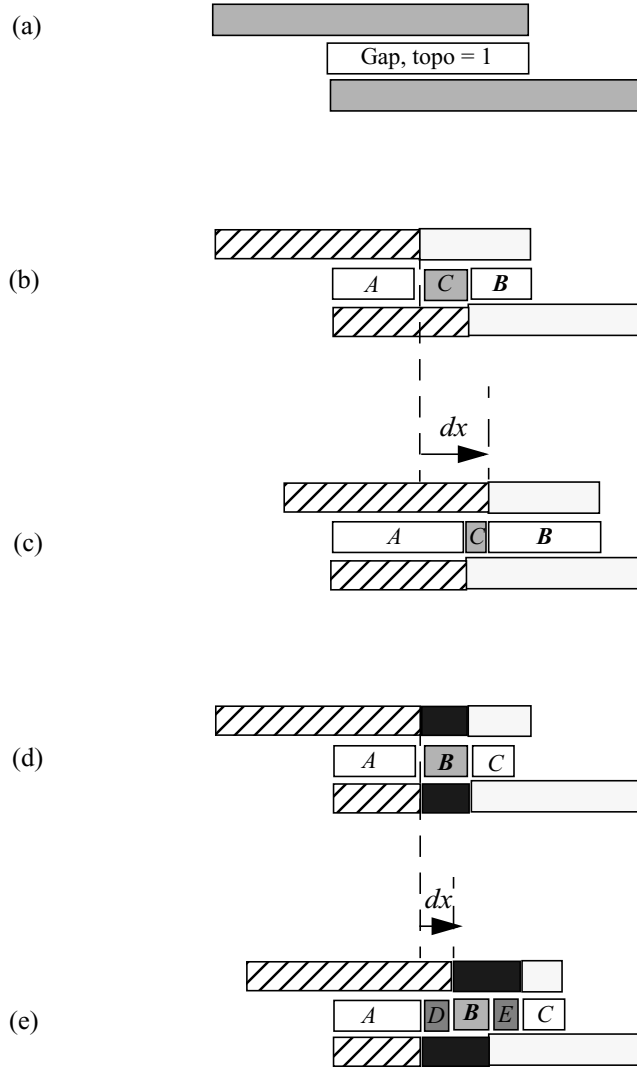
Figure 3.49: Composibility problems caused by the current definition of topology

multiple gap elements. When the beams are undisplaced, gap *A* and *C* have a topology of 1,

gap *B* can have a topology of either 1 or 0, and the entire overlapping region is covered.

However, once the beam electrodes start to move, problem happens. As shown in Figure

3.49 (e), if the top beam moves by *dx* along *x*-direction, gap *A* and *C* will turn into

uncovered Case IV of Figure 3.48, gap *B* turns into a topology of 0, and overlapping region

zone *D* and zone *E* become uncovered. Note that in the above discussion, the interaction

between the electrical fields in different zones at the boundaries are ignored, which In fact

also affect the accurate capture of the physics.

To solve these problems caused by the current definition of *topology*, either a new parameter should be introduced, or the *topology* should be defined as a dynamic variable which is a function of the displacements rather than being defined as a static parameter. However, as the displacements due to electrostatic effect are usually very small (in the amount of nm) compared to the length of the initial overlap and the length of the electrodes, most devices work in Case I and III where the topology doesn't change. Therefore, when the physical gap is composed by just one gap element, the current definition of topology is acceptable for many cases.

Topology Symmetry

As mentioned, the force and moment equations given in previous sections are all for the cases with *topology* = 1. For compactness of the model, the same set of equations is used for the cases with *topology* = 0, based on the similarity between these two kinds of topologies as shown in Figure 3.50. Coordinate system, electrode length and lumped electrostatic forces and moments are drawn in the figure for both cases. We see that if we regard the electrode 2 in *topology* 0 as the electrode 1 in *topology* 1 and regard the electrode 1 in *topology* 0 as the electrode 2 in *topology* 1, the equations for forces (F_{a1}, F_{a2}, *etc.*) and moments (M_{a1}, M_{a2}, *etc.*) will be the same except that the signs are reversed. Therefore, we did the following to enable the sharing of equations:
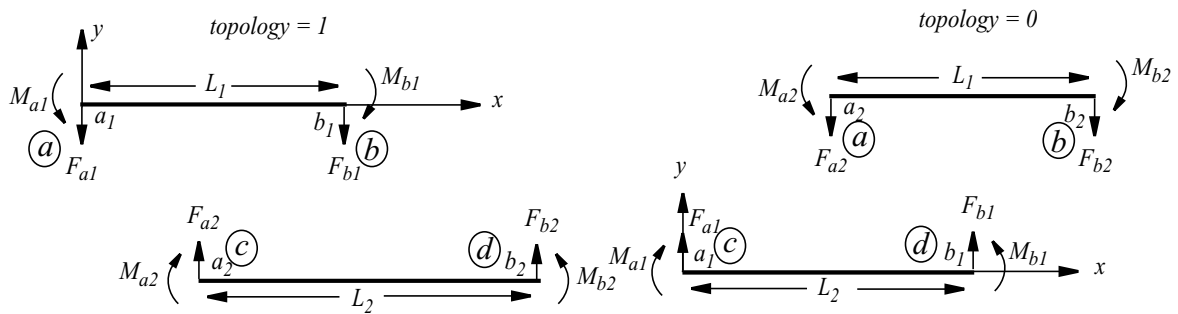
Figure 3.50: Similarity between topology 1 and topology 0.

1) Define the nodal displacements to be used in force /moments calculations, depending on the topology. For example:

$$
\begin{aligned}
x_{a1} &= x_a \cdot topo + x_c \cdot (1 - topo) \\
y_{a1} &= y_a \cdot topo - y_c \cdot (1 - topo) \\
\phi_{a1} &= \phi_a \cdot topo - \phi_c \cdot (1 - topo)
\end{aligned}
\tag{3.207}
$$

where x_{a1}, y_{a1} and ϕ_{a1} are the displacement variables applied to the force and moment equations, while x_a, y_a, ϕ_a, x_c, y_c and ϕ_c are the displacement measured at the connection terminals a and c of the gap. Point $a1$ resides on different electrode for different topologies. Notice that the sign of y-displacement and angular displacement is reversed for *topology* 0 due to the switching of the relative vertical position of electrode 1 and electrode 2 in y-direction. Other displacement variables are derived in the same way.

2) Define the electrode lengths depending on the topology:

$$
\begin{aligned}
L_1' &= L_1 \cdot topo + L_2 \cdot (1 - topo) \\
L_2' &= L_2 \cdot topo + L_1 \cdot (1 - topo)
\end{aligned}
\tag{3.208}
$$

where L_1' and L_2' are the electrode length used in the force and moment equations, while L_1 and L_2 are the actual length of electrode 1 and electrode 2 before the switching.

3) Switch the forces and moments according to the topology. For example:

$$
\begin{aligned}
F_a &= F_{a1} \cdot topo - F_{a2} \cdot (1 - topo) \\
M_a &= M_{a1} \cdot topo - M_{a2} \cdot (1 - topo)
\end{aligned}
\tag{3.209}
$$

where F_{a1}, M_{a1}, F_{a2} and M_{a2} are the forces and moments given by the force and moment equations, while F_a and M_a are the force and moment applied to the connectional terminal a of the gap. Notice that the sign of the forces and moments are reversed for *topology* 0 due to the switching of the relative vertical position of electrode 1 and electrode 2 in y-direction, as shown in Figure 3.50. Forces and moments at other connection terminals are derived in the same way.

3.5.8 Electrical Model

The electrostatic field not only generates electrostatic forces and moments, but also generates electrical currents when the electrodes are moving. The charges are distributed on the surface of the electrodes thus the currents need to be lumped to the beam ends, similar to the lumping of distributed electrostatic force. The currents generated due to the change in charge distribution enable the current sensing, one of the major sensing techniques provided by electrostatic devices.

The total current is calculated as:

$$I_{total} = \frac{dq}{dt} = \frac{d}{dt}(V \cdot C_{total}) \qquad (3.210)$$

where C_{total} is the total capacitance as given by (3.175) and V is the voltage difference between the reference points defined as:

$$V = (V_c - V_b) \cdot topo + (V_d - V_a) \cdot (1 - topo) \qquad (3.211)$$

(3.211) is an approximation of the real case because the voltage difference actually varies along the electrodes when there are currents flowing through. However, as the currents are usually very small (in the magnitude of nA), voltage difference between the two ends of the beam electrodes is negligible therefore (3.211) can be applied with acceptable accuracy. (3.210) is charge-conservative as it includes both the time-varying voltage and the time-varying capacitance into the differentiation.

The lumping method based on beam bending shape functions as used for the lumping of electrostatic force is not applicable to the lumping of the electrical currents. Instead, the total current is lumped based on the rigid-plate shape functions, as an approximation:

$$I_{tip} = I_{total} \cdot \left(1 - \frac{x_{ov}}{2L}\right)$$

$$I_{attach} = I_{total} \cdot \left(\frac{x_{ov}}{2L}\right) \qquad (3.212)$$

I_{tip} is the amount of current fed into the reference points of the gap and I_{attach} is the amount

of current fed into other connection terminals other than the reference points. The currents

at connection terminals are defined based on the topology:

$$\begin{aligned}
I_{b,c} &= I_{tip} \cdot topo + I_{attach} \cdot (1 - topo) \\
I_{a,d} &= I_{attach} \cdot topo + I_{tip} \cdot (1 - topo)
\end{aligned}$$

(3.213)

where $I_{b,c}$ is the current flowing from terminal *b* to terminal *c*, and $I_{c,d}$ is the current flowing

from terminal *c* to terminal *d*.

3.5.9 Verilog-A Code Implementation

Code segment of the electrostatic gap model is given below to show the

implementation in Verilog-A.

```
// include header files
'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);
// definition of bus pins
inout phia_b;
rotational phia_b;
inout va_b;
electrical va_b;
inout [0:1] xa_b;
kinematic [0:1] xa_b;
...
// topology = 0 if the top finger is located on the right of the bottom finger
// topology = 1 if the top finger is located on the left of the bottom finger
parameter integer topology = 1;
parameter real finger_w_t = 2e-6;
parameter real finger_w_b = 2e-6;
parameter real finger_l_t = 20e-6;
parameter real finger_l_b = 20e-6;
parameter real finger_number = 1;
parameter real gap = 2e-6;
parameter real overlap = 20e-6;
parameter real thickness = 2e-6;
parameter real E = 165e9;
parameter real angle = 0;
parameter real visc_air = 'default_visc_air;
```

```
parameter real contact_oxide_t = 'default_ntv_ox_t;
// internal nodes
kinematic V_dy;
// definition of internal variables
real ov;     // dynamic overlap along x-axis
real dy;     // change of gap in y-direction
real dx;     // change of overlap in x-direction
real vlt;    // applied voltage
real cap;    // capacitance
integer const2; // used for switch between normal and in-contact states
real q1, q2;  // charge
real q;  // charge
real slope;  // stiffness of contact spring
real Fa1_l_e, Fa2_l_e, Fb1_l_e, Fb2_l_e, Fa1_l_b, Fa2_l_b, Fb1_l_b, Fb2_l_b;
real Ma1_l_e, Ma2_l_e, Mb1_l_e, Mb2_l_e, Ma1_l_b, Ma2_l_b, Mb1_l_b, Mb2_l_b;
...
analog begin
// check the validity of the value of parameter topology
     @ (initial_step) begin
     if (topology < 0 | topology > 1) begin
     $display("Invalid topology, must be either '0' or '1', see 'help' for descriptions.");
      $finish;
     end
     end
// calculate internal variables
     rad = angle* 'M_PI /180;
     cos_dc = cos(rad);
     sin_dc = sin(rad);

// transform the displacement from chip frame into 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]);
...
// to use same set of equations for different topologies
// determine the variables used for different topologies
xm1 = xm1_l*topology+xm2_l*(1-topology);
ym1 = ym1_l*topology-ym2_l*(1-topology);
...
am1 = Theta(phia_t)*1e6*topology-Theta(phia_b)*1e6*(1-topology);
ap1 = Theta(phib_t)*1e6*topology-Theta(phib_b)*1e6*(1-topology);
...
vlt = (1-topology)*V(vb_b,va_t) + topology*V(va_b,vb_t);
vlt_sqrt = pow(vlt,2);

// parameter overlap is the initial overlap
// dx is used for calculation of dynamic overlap, ov
```

```
dx = xp1-xm2;
if(dx <=  -overlap) begin
//  beams are not overlapped to form a gap
ov = 0;
end
else if(dx <= (topology*(finger_l_b-overlap)+(1-topology)*(finger_l_t-overlap))) begin
//  beams have increasing overlap
ov = overlap + dx;
end
else if(dx <= (finger_l_t+finger_l_b - overlap)) begin
//  beams have decreasing overlap
ov = - dx + finger_l_t+finger_l_b - overlap;
end
else begin
//  beams are not overlapped to form a gap
ov = 0;
end


// L1 and L2 are switched for topology 1 and 0
L1 = finger_l_t*topology+finger_l_b*(1-topology);
L2 = finger_l_b*topology+finger_l_t*(1-topology);


// physical equations are formulated based on toplogy 1
// y1_mid is the y-disp of beam 1 at the inner boundary of overlapping region
y1_mid = ym1*(1-(3*pow(L1-ov,2))/pow(L1,2)+(2*pow(L1-ov,3))/pow(L1,3))+
   ap1*(-(pow(L1-ov,2)/L1)+pow(L1-ov,3)/pow(L1,2))+
   am1*(L1-(2*pow(L1-ov,2))/L1+pow(L1-ov,3)/pow(L1,2)-ov)+
   ((3*pow(L1-ov,2))/pow(L1,2)-(2*pow(L1-ov,3))/pow(L1,3))*yp1;
// y2_mid is the y-disp of beam 2 at the inner boundary of overlapping region
y2_mid = ym2*((3*pow(L2-ov,2))/pow(L2,2)-(2*pow(L2-ov,3))/pow(L2,3))-
   am2*(-(pow(L2-ov,2)/L2)+pow(L2-ov,3)/pow(L2,2))-
   ap2*(L2-(2*pow(L2-ov,2))/L2+pow(L2-ov,3)/pow(L2,2)-ov)+
   (1-(3*pow(L2-ov,2))/pow(L2,2)+(2*pow(L2-ov,3))/pow(L2,3))*yp2;


// dy is used to determine the operation status of the gap: normal or in-contact
dy = min((yp1-y2_mid),(y1_mid-ym2));
     if((gap + dy) > contact_oxide_t*2)    //normal gap
      const2 = 0;
     else                                 //in contact
      const2 = 1;


// calculation of cap, forces and moments, based on topology 1
lm = L1 - ov;
lp = L1;
// form the rigid-plate approximation for the beam electrodes
xmid = (lm+lp)/2.0;
```

ang1mid = am1*(1 - (2*(lm + lp))/L1 + (3*pow(lm + lp,2))/(4.0*pow(L1,2))) + ap1*(-((lm + lp)/L1) + (3*pow(lm + lp,2))/(4.0*pow(L1,2))) + ((-3*(lm + lp))/pow(L1,2) + (3*pow(lm + lp,2))/(2.0*pow(L1,3)))*ym1 + ((3*(lm + lp))/pow(L1,2) - (3*pow(lm + lp,2))/(2.0*pow(L1,3)))*yp1;

y1mid = am1*((lm + lp)/2.0 - pow(lm + lp,2)/(2.0*L1) + pow(lm + lp,3)/(8.0*pow(L1,2))) + ap1*(-pow(lm + lp,2)/(4.0*L1) + pow(lm + lp,3)/(8.0*pow(L1,2))) + (1 - (3*pow(lm + lp,2))/(4.0*pow(L1,2)) + pow(lm + lp,3)/(4.0*pow(L1,3)))*ym1 + ((3*pow(lm + lp,2))/(4.0*pow(L1,2)) - pow(lm + lp,3)/(4.0*pow(L1,3)))*yp1;

y1average = (am1*(lm - (2*pow(lm,2))/L1 + pow(lm,3)/pow(L1,2)) + ap1*(-(pow(lm,2)/L1) + pow(lm,3)/pow(L1,2)) +   am1*(lp - (2*pow(lp,2))/L1 + pow(lp,3)/pow(L1,2)) + ap1*(-(pow(lp,2)/L1) + pow(lp,3)/pow(L1,2)) +   (1 - (3*pow(lm,2))/pow(L1,2) + (2*pow(lm,3))/pow(L1,3))*ym1 +   (1 - (3*pow(lp,2))/pow(L1,2) + (2*pow(lp,3))/pow(L1,3))*ym1 +   ((3*pow(lm,2))/pow(L1,2) - (2*pow(lm,3))/pow(L1,3))*yp1 +   ((3*pow(lp,2))/pow(L1,2) - (2*pow(lp,3))/pow(L1,3))*yp1)/2.0;

...
// intermediate variables used in the expression of cap, forces and moments
c1 = ang1mid - ang2mid;
c2 = (y1average + y1mid)/2.0 + (-y2average - y2mid)/2.0;
c3 = c2 + gap + c1*(L1 - ov/2.0 - xmid);

// rotation angle of the rotated electrical field lines
theta0 = (ang1mid + ang2mid)/2.0;

intFy0 = 'eps0*thickness*vlt_sqrt/pow(cos(theta0),2)/2.0;

// total capacitance obtained through integration in overlapping region
cap = 'eps0*thickness*(ov*(240*pow(c3,4) + 20*pow(c1,2)*pow(c3,2)*pow(ov,2) + 3*pow(c1,4)*pow(ov,4))*cos(theta0))/(240.0*pow(c3,5));

// electrostatic force in x-direction in the rotated frame of reference
Fxd_l = -('eps0*thickness*vlt_sqrt*((ov*cos(theta0))/c3 + (pow(c1,2)*pow(ov,3)*cos(theta0))/(12.0*pow(c3,3)) + (pow(c1,4)*pow(ov,5)*cos(theta0))/(80.0*pow(c3,5))))/(2.0*ov);

// electrostatic forces and moments in y-direction in the rotated frame of reference
Fa1_l_e = (intFy0*pow(ov,3)*(560*pow(c3,4)*(2*L1 - ov) + 112*c1*pow(c3,3)*(5*L1 - 3*ov)*ov + 28*pow(c1,2)*pow(c3,2)*(12*L1 - 7*ov)*pow(ov,2) + 8*pow(c1,3)*c3*(21*L1 - 13*ov)*pow(ov,3) + 5*pow(c1,4)*(18*L1 - 11*ov)*pow(ov,4)))/(1120.0*pow(c3,6)*pow(L1,3));

Ma1_l_e = (intFy0*pow(ov,3)*(560*pow(c3,4)*(4*L1 - 3*ov) + 112*c1*pow(c3,3)*(10*L1 - 9*ov)*ov + 84*pow(c1,2)*pow(c3,2)*(8*L1 - 7*ov)*pow(ov,2) + 24*pow(c1,3)*c3*(14*L1 - 13*ov)*pow(ov,3) + 15*pow(c1,4)*(12*L1 - 11*ov)*pow(ov,4)))/(6720.0*pow(c3,6)*pow(L1,2));

...
// electrostatic forces and moments in y-direction transformed from rotated frame to local frame
Fb1x = cos(theta0)*Fxd_l-sin(theta0)*Fb1_l_e;
Fb1y_e = sin(theta0)*Fxd_l+cos(theta0)*Fb1_l_e;
...
// lumped damping forces and moments
intFy0_b = -0.6*visc_air*ov*finger_w_t*finger_w_t*pow(1.0/cos(theta0),3);

Fa1y_b = ddt((intFy0_b*pow(ov,3)*(560*pow(c3,5)*(2*L1-ov)+15*pow(c1,4)*gap*pow(ov,4)*(-

18*L1+11*ov)+8*pow(c1,2)*pow(c3,2)*pow(ov,2)*(-7*gap*(12*L1-7*ov)+c1*(21*L1-13*ov)*ov)+28*c1*pow(c3,3)*ov*(-6*gap*(5*L1-3*ov)+c1*(12*L1-7*ov)*ov)-112*pow(c3,4)*(5*gap*(2*L1-ov)+c1*ov*(-5*L1+3*ov))+5*pow(c1,3)*c3*pow(ov,3)*(c1*(18*L1-11*ov)*ov+gap*(-84*L1+52*ov))))/(1120.0*pow(c3,7)*pow(L1,3)));

    Ma1_l_b = ddt((intFy0_b*pow(ov,3)*(560*pow(c3,5)*(4*L1-3*ov)+45*pow(c1,4)*gap*pow(ov,4)*(-12*L1+11*ov)-112*pow(c3,4)*(5*gap*(4*L1-3*ov)+c1*ov*(-10*L1+9*ov))+84*c1*pow(c3,3)*ov*(c1*(8*L1-7*ov)*ov+gap*(-20*L1+18*ov))+24*pow(c1,2)*pow(c3,2)*pow(ov,2)*(c1*(14*L1-13*ov)*ov+gap*(-56*L1+49*ov))+15*pow(c1,3)*c3*pow(ov,3)*(c1*(12*L1-11*ov)*ov+gap*(-56*L1+52*ov))))/(6720.0*pow(c3,7)*pow(L1,2)));

```
    ...
    //sum up electrostatic and damping forces and moments
    Fa1y = Fa1y_e + Fa1y_b;
    Fb1y = Fb1y_e + Fb1y_b;
    ...
    Ma1 = Ma1_l_e + Ma1_l_b;
    Mb1 = Mb1_l_e + Mb1_l_b;
    ....
    // contact model
    slope = E*1e-6*thickness/(finger_w_t/2+finger_w_b/2);
    Fy_contact = const2*slope*(-dy-gap+contact_oxide_t*2);

    // sum up electrostatic, damping and contact forces
    Fy_tip_1    = Fb1y - Fy_contact;
    Fy_attach_1 = Fa1y ;
    Fy_tip_2    = Fa2y + Fy_contact;
    Fy_attach_2 = Fb2y ;

    // transform forces and moments from local frame back to chip frame
    F(xb_t[0]) <+ - (cos_dc*(topology*Fb1x+(1-topology)*Fb2x)-sin_dc*(topology*Fy_tip_1-(1-
topology)*Fy_attach_2))*finger_number;
    F(xb_t[1]) <+ - (sin_dc*(topology*Fb1x+(1-topology)*Fb2x)+cos_dc*(topology*Fy_tip_1-(1-
topology)*Fy_attach_2))*finger_number;
    Tau(phib_t) <+ - (topology*Mb1_l-(1-topology)*Mb2_l)*finger_number;
    ...
    // electrical currents
    q = vlt * cap;
    I_tip    = ddt(q)*(1-ov/2.0/finger_l_t) ;
    I_attach = ddt(q)*ov/2.0/finger_l_t ;
        I(v_b_t,v_a_b) <+ topology * I_tip+(1-topology) * I_attach;
        I(v_a_t,v_b_b) <+ (1-topology) * I_tip+topology * I_attach;
    end
    endmodule
```

3.5.10 Model Verification

Figure 3.51 shows the static and transient analyses of a 2D gap composed of a cantilever beam electrode on the top and a fixed electrode at the bottom. The cantilever beam is represented by one beam element in NODAS simulation. y-displacement at the tip of the cantilever beam, y_{tip}, is plotted versus the voltage applied across the electrodes.

The static analysis results from NODAS are compared to the data obtained from simulation in CoventorWare, a combined FEA/BEA tool which solves for the self-consistent solution through alternative mechanical and electrostatic analyses [76]. As the fringing field is included by CoventorWare but not in the NODAS gap model, the beam electrode is designed to be large ($L = 100\ \mu m$, $t = 20\ \mu m$, t is the electrode thickness in z-direction) and thin ($w = 2\ \mu m$, w is the electrode width in x-y plane) and the gap is designed to be very small ($g_0 = 0.5\ \mu m$) compared to the electrode dimensions, so that the fringing
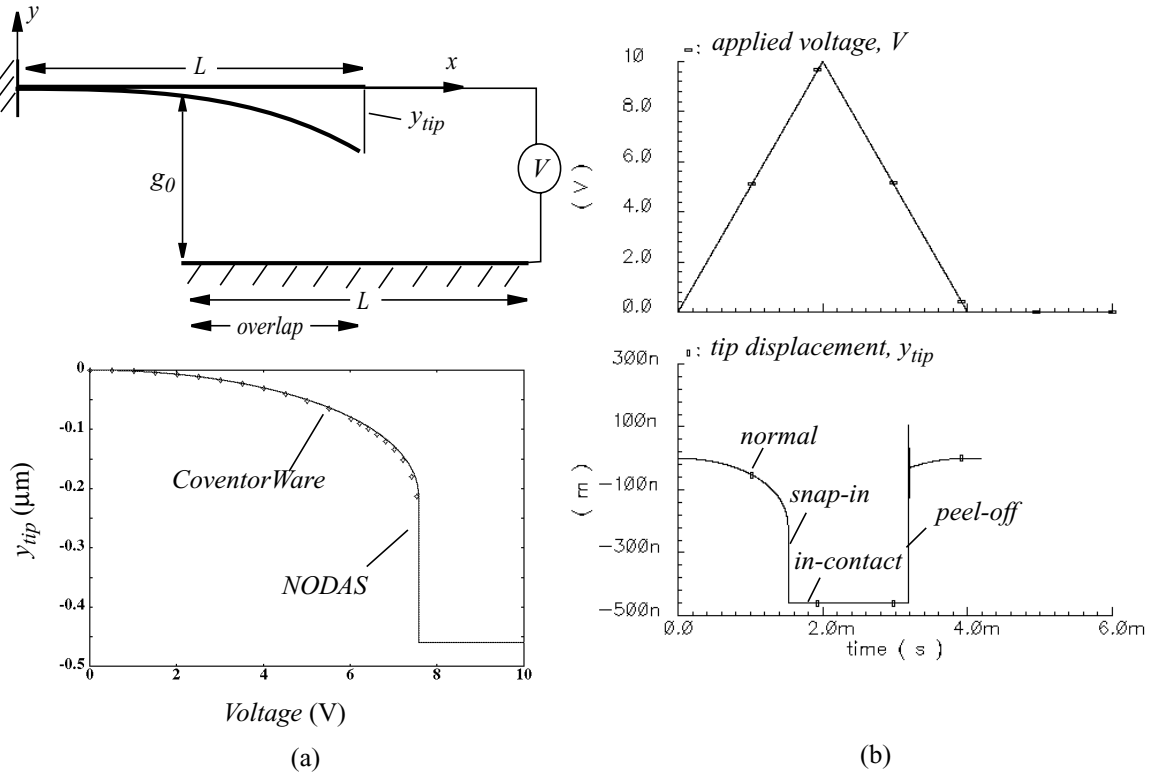
Figure 3.51: Simulation of a 2D gap composed of a cantilever beam electrode and a fixed electrode ($L = 100\ \mu m$, $w = 2\ \mu m$, $t = 20\ \mu m$, $E = 165$ GPa, $g_0 = 0.5\ \mu m$, *overlap* $= 50\ \mu m$) (a) Static analysis (b) transient analysis

effect is negligible. Comparison in Figure 3.51 (a) shows that NODAS simulation results match very well to that from CoventorWare. The error in the snap-in voltage is within 3%. The entire static analysis in NODAS takes about 1 min, while in self-consistent FEA/BEA, each data point requires about 20 min.

Transient analysis of the structure is also performed in NODAS, while not supported in CoventorWare. The voltage increases as a ramp from zero to a value larger than the pull-in threshold, then ramps back to zero. As shown in Figure 3.51(b), the gap goes through four stages during this process: normal operation, snap-in, in-contact and peel-off. The entire transient simulation in NODAS takes about 5 min.

Figure 3.52(a) illustrates a MEMS switch composed of a fixed-fixed beam on the top and a fixed electrode at the bottom. When the fixed-fixed beam is in its original position, the switch is off. When a voltage is applied, the beam bends down toward the bottom electrode and the switch is on. The fixed-fixed beam is represented by two beam elements, in order to access the displacement at the center point of the beam. Accordingly, the physical gap is also composed using two gap elements.
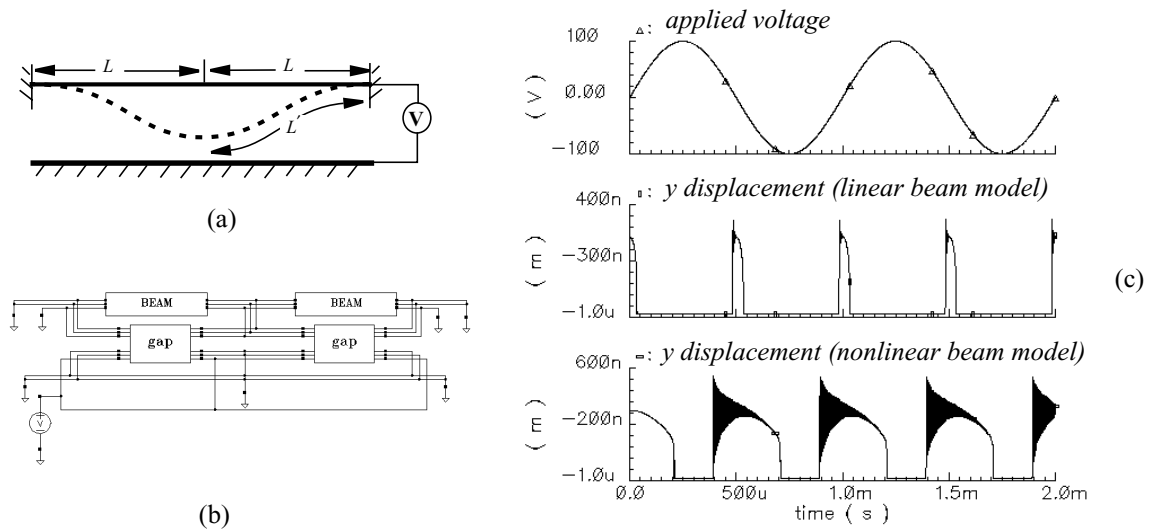
Figure 3.52: (a) Structure of a MEMS switch (L = 150 μm, w = 2 μm, t = 50 μm, gap = 1 μm) (b) NODAS schematic (c) transient analysis.

A sinusoidal voltage of 1 kHz is applied to the structure. Results from transient simulations with linear and nonlinear beam models are given in Figure 3.52(c). With the nonlinear beam model, beam stiffening is captured, which is critical and non-negligible in the fixed-fixed beam. The snap-in consequently happens at a voltage much higher than in the linear case due to the beam stiffening. The simulation also shows the nonlinearity in the relation between the electrostatic force and the applied voltage. Since the electrostatic force is proportional to V^2, the second harmonic is the dominating displacement term when V is a sinusoidal voltage.

Chapter 4   Extensibility of NODAS Modeling Methodology

A single cell library can not be expected to cover every desired feature. For a modeling methodology to be useful, it needs to be applicable to a wide range of applications and be extensible to new physical effects, new design styles, new processes and new physical domains. In this chapter, these four types of extensibility are discussed with supporting examples in NODAS and guidelines for extension.

4.1 New Physical Effects

All models are abstractions of the physical world and cannot capture all the physical effects involved in the actual system. Physical effects which are negligible in certain design spaces may become non-negligible in some other design spaces. For instance, slender beams are widely used in low-frequency MEMS designs, but, for recently emerging RF-MEMS designs, shorter beams are used for higher frequency operation. The shear effect in such beams thus becomes non-negligible. In this section, the linear and nonlinear beam models given in Chapter 3 are extended to include the shear effects, as an example of the extensibility to new physical effects.

4.1.1 Example: Shear Effect in Beam Element

4.1.1.1 Linear Beam Model with Shear

The linear stiffness matrix and mass matrix for beam elements with shear effect are given in [43], based on the assumption that the lateral deflection u_y on the beam subjected to shearing forces and associated moments is the sum of u_b, the lateral deflection due to bending strains and u_s, the additional deflection due to shear strains. The linear stiffness matrix with shear effect, $[k]_s$ is:

$$[k]_s = \begin{bmatrix}
\frac{Etw}{l} & & & & & & -\frac{Etw}{l} & & & & & \\[4pt]
& \frac{12EI_z}{l^3(1+\Phi_y)} & & & & \frac{6EI_z}{l^2(1+\Phi_y)} & & -\frac{12EI_z}{l^3(1+\Phi_y)} & & & & \frac{6EI_z}{l^2(1+\Phi_y)} \\[4pt]
& & \frac{12EI_y}{l^3(1+\Phi_z)} & & -\frac{6EI_y}{l^2(1+\Phi_z)} & & & & -\frac{12EI_y}{l^3(1+\Phi_z)} & & -\frac{6EI_y}{l^2(1+\Phi_z)} & \\[4pt]
& & & \frac{GJ_t}{l} & & & & & & -\frac{GJ_t}{l} & & \\[4pt]
& & -\frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(4+\Phi_z)EI_y}{l(1+\Phi_z)} & & & & \frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(2-\Phi_z)EI_y}{l(1+\Phi_z)} & \\[4pt]
& \frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(4+\Phi_y)EI_z}{l(1+\Phi_y)} & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(2-\Phi_y)EI_z}{l(1+\Phi_y)} \\[4pt]
-\frac{Etw}{l} & & & & & & \frac{Etw}{l} & & & & & \\[4pt]
& -\frac{12EI_z}{l^3(1+\Phi_y)} & & & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & \frac{12EI_z}{l^3(1+\Phi_y)} & & & & -\frac{6EI_z}{l^2(1+\Phi_y)} \\[4pt]
& & -\frac{12EI_y}{l^3(1+\Phi_z)} & & \frac{6EI_y}{l^2(1+\Phi_z)} & & & & \frac{12EI_y}{l^3(1+\Phi_z)} & & \frac{6EI_y}{l^2(1+\Phi_z)} & \\[4pt]
& & & -\frac{GJ_t}{l} & & & & & & \frac{GJ_t}{l} & & \\[4pt]
& & -\frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(2-\Phi_z)EI_y}{l(1+\Phi_z)} & & & & \frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(4+\Phi_z)EI_y}{l(1+\Phi_z)} & \\[4pt]
& \frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(2-\Phi_y)EI_z}{l(1+\Phi_y)} & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(4+\Phi_y)EI_z}{l(1+\Phi_y)}
\end{bmatrix} \tag{4.1}$$

Φ_y and Φ_z are *shear-deformation parameters* defined as

$$\Phi_y = \frac{12EI_z}{GA_{s_y}l^2} = 24(1+\nu)\frac{A}{A_{s_y}}\left(\frac{r_z}{l}\right)^2 \tag{4.2}$$

$$\Phi_z = \frac{12EI_y}{GA_{s_z}l^2} = 24(1+\nu)\frac{A}{A_{s_z}}\left(\frac{r_y}{l}\right)^2 \tag{4.3}$$

where r_z and r_y are the *radius of gyration* about z-axis and y-axis defined as

$$r_z = \sqrt{\frac{I_z}{tw}}, \; r_y = \sqrt{\frac{I_y}{tw}} \tag{4.4}$$

A is the cross-section area $A = tw$, used for shear in both y and z directions, A_{sy} and A_{sz} are the *effective shear areas* defined as

$$A_{s_y} = \frac{A}{\alpha_s}, \; A_{s_z} = \frac{A}{\alpha_s} \tag{4.5}$$

α_s is the *shear coefficient* which is 3/2 for a rectangular cross-section. Shear coefficient for other shapes of cross-section are given in [52].

Expressed in geometric and material parameters,

$$\Phi_y = \frac{E\alpha_s}{G}\left(\frac{w}{L}\right)^2 = 2\alpha_s(1+\nu)\left(\frac{w}{L}\right)^2, \Phi_z = \frac{E\alpha_s}{G}\left(\frac{t}{L}\right)^2 = 2\alpha_s(1+\nu)\left(\frac{t}{L}\right)^2 \tag{4.6}$$

The mass matrix with shear effect is given on page 158 and page 159 [43].

The stiffness and mass matrices for beams with shear show that the fundamental structure of the beam model stays unchanged. The only modifications to the matrices to include the shear effect are the addition of shear terms to the original matrix elements. Comparing the stiffness and mass matrices with and without shear is possible by considering (4.6) with $w \ll L$ and $t \ll L$ (i.e., slender beam). Both Φ_y and Φ_z reduce to zero for slender beams, and the above matrices accordingly reduce to the matrices without shear effect as given in Chapter 3. In addition, in the mass matrix with shear effect, a second term, as given on page 159, is added representing the rotary inertia of the beam, which was not considered in Chapter 3.

4.1.1.2 Nonlinear Beam Model with Shear

The matrices for the nonlinear beam model with shear are derived in NODAS with the aid of a math tool, Mathematica [77], by adding the shear energy due to shear deformation, U_s, to the strain energy caused by beam bending, U_b, following the same energy method as used in the derivation of nonlinear beam model without shear effects in Chapter 3.

The beam bending shape functions with shear deformation are [43]:

$$f_{1y} = 1 - 3\xi^2 + 2\xi^3 + (1-\xi)\Phi_y \tag{4.7}$$

$$f_{2y} = \left(\xi - 2\xi^2 + \xi^3 + \frac{(\xi - \xi^2)}{2}\Phi_y\right)L \tag{4.8}$$

$$f_{3y} = 3\xi^2 - 2\xi^3 + \xi\Phi_y \tag{4.9}$$

mass matrix with shear effect

$$[m] = \rho t w l \begin{bmatrix}
\frac{1}{3} & & & & & & & & & & & \\
\frac{\frac{13}{35}+\frac{7}{10}\Phi_y+\frac{1}{3}\Phi_y^{\ 2}}{(1+\Phi_y)^2} & & & & & & & & & & & \\
& \frac{\frac{13}{35}+\frac{7}{10}\Phi_z+\frac{1}{3}\Phi_z^{\ 2}}{(1+\Phi_z)^2} & & & & \text{\textit{symmetric}} & & & & & & \\
& & \frac{I_y+I_z}{3tw} & & & & & & & & & \\
& -\frac{\left(\frac{11}{210}+\frac{11}{120}\Phi_z+\frac{1}{24}\Phi_z^{\ 2}\right)l}{(1+\Phi_z)^2} & & \frac{\left(\frac{1}{105}+\frac{1}{60}\Phi_z+\frac{1}{120}\Phi_z^{\ 2}\right)l^2}{(1+\Phi_z)^2} & & & & & & & & \\
\frac{\left(\frac{11}{210}+\frac{11}{120}\Phi_y+\frac{1}{24}\Phi_y^{\ 2}\right)l}{(1+\Phi_y)^2} & & & & \frac{\left(\frac{1}{105}+\frac{1}{60}\Phi_y+\frac{1}{120}\Phi_y^{\ 2}\right)l^2}{(1+\Phi_y)^2} & & & & & & & \\
\frac{1}{6} & & & & & & \frac{1}{3} & & & & & \\
\frac{\frac{9}{70}+\frac{3}{10}\Phi_y+\frac{1}{6}\Phi_y^{\ 2}}{(1+\Phi_y)^2} & & & & \frac{\left(\frac{13}{420}+\frac{3}{40}\Phi_y+\frac{1}{24}\Phi_y^{\ 2}\right)l}{(1+\Phi_y)^2} & \frac{\frac{13}{35}+\frac{7}{10}\Phi_y+\frac{1}{3}\Phi_y^{\ 2}}{(1+\Phi_y)^2} & & & & & & \\
& \frac{\frac{9}{70}+\frac{3}{10}\Phi_z+\frac{1}{6}\Phi_z^{\ 2}}{(1+\Phi_z)^2} & & -\frac{\left(\frac{13}{420}+\frac{3}{40}\Phi_z+\frac{1}{24}\Phi_z^{\ 2}\right)l}{(1+\Phi_z)^2} & & & & \frac{\frac{13}{35}+\frac{7}{10}\Phi_z+\frac{1}{3}\Phi_z^{\ 2}}{(1+\Phi_z)^2} & & & & \\
& & \frac{I_y+I_z}{6tw} & & & & & & \frac{I_y+I_z}{3tw} & & & \\
& \frac{\left(\frac{13}{420}+\frac{3}{40}\Phi_z+\frac{1}{24}\Phi_z^{\ 2}\right)l}{(1+\Phi_z)^2} & & -\frac{\left(\frac{1}{140}+\frac{1}{60}\Phi_z+\frac{1}{120}\Phi_z^{\ 2}\right)l^2}{(1+\Phi_z)^2} & & & & \frac{\left(\frac{11}{210}+\frac{11}{120}\Phi_z+\frac{1}{24}\Phi_z^{\ 2}\right)l}{(1+\Phi_z)^2} & & \frac{\left(\frac{1}{105}+\frac{1}{60}\Phi_z+\frac{1}{120}\Phi_z^{\ 2}\right)l^2}{(1+\Phi_z)^2} & \\
-\frac{\left(\frac{13}{420}+\frac{3}{40}\Phi_y+\frac{1}{24}\Phi_y^{\ 2}\right)l}{(1+\Phi_y)^2} & & & & -\frac{\left(\frac{1}{140}+\frac{1}{60}\Phi_y+\frac{1}{120}\Phi_y^{\ 2}\right)l^2}{(1+\Phi_y)^2} & -\frac{\left(\frac{11}{210}+\frac{11}{120}\Phi_y+\frac{1}{24}\Phi_y^{\ 2}\right)l}{(1+\Phi_y)^2} & & & & & \frac{\left(\frac{1}{105}+\frac{1}{60}\Phi_y+\frac{1}{120}\Phi_y^{\ 2}\right)l^2}{(1+\Phi_y)^2}
\end{bmatrix}$$

mass matrix with shear effect, *cont.*

$$+\rho t w l \begin{bmatrix} 0 & & & & & & & & symmetric \\ \dfrac{\left(\frac{r_z}{l}\right)^2 \frac{6}{5}}{(1+\Phi_y)^2} & & & & & & & \\ & \dfrac{\left(\frac{r_y}{l}\right)^2 \frac{6}{5}}{(1+\Phi_z)^2} & & & & & & \\ & & 0 & & & & & \\ & \dfrac{\left(\frac{r_y}{l}\right)^2\left(\frac{1}{10}-\frac{1}{2}\Phi_z\right)l}{(1+\Phi_z)^2} & \dfrac{\left(\frac{r_y}{l}\right)^2\left(\frac{2}{15}+\frac{1}{6}\Phi_z+\frac{1}{3}\Phi_z{}^2\right)l^2}{(1+\Phi_z)^2} & & & & & \\ \dfrac{\left(\frac{r_z}{l}\right)^2\left(\frac{1}{10}-\frac{1}{2}\Phi_y\right)l}{(1+\Phi_y)^2} & & & \dfrac{\left(\frac{r_z}{l}\right)^2\left(\frac{2}{15}+\frac{1}{6}\Phi_y+\frac{1}{3}\Phi_y{}^2\right)l^2}{(1+\Phi_y)^2} & & & \\ 0 & & & & 0 & & & \\ -\dfrac{\left(\frac{r_z}{l}\right)^2 \frac{6}{5}}{(1+\Phi_y)^2} & & & \dfrac{\left(\frac{r_z}{l}\right)^2\left(-\frac{1}{10}+\frac{1}{2}\Phi_y\right)l}{(1+\Phi_y)^2} & \dfrac{\left(\frac{r_z}{l}\right)^2 \frac{6}{5}}{(1+\Phi_y)^2} & & \\ & -\dfrac{\left(\frac{r_y}{l}\right)^2 \frac{6}{5}}{(1+\Phi_z)^2} & \dfrac{\left(\frac{r_y}{l}\right)^2\left(-\frac{1}{10}+\frac{1}{2}\Phi_z\right)l}{(1+\Phi_z)^2} & & & \dfrac{\left(\frac{r_y}{l}\right)^2 \frac{6}{5}}{(1+\Phi_z)^2} & \\ & 0 & & & & & 0 & \\ & \dfrac{\left(\frac{r_y}{l}\right)^2\left(\frac{1}{10}-\frac{1}{2}\Phi_z\right)l}{(1+\Phi_z)^2} & \dfrac{\left(\frac{r_y}{l}\right)^2\left(-\frac{1}{30}-\frac{1}{6}\Phi_z+\frac{1}{6}\Phi_z{}^2\right)l^2}{(1+\Phi_z)^2} & & & \dfrac{\left(\frac{r_y}{l}\right)^2\left(-\frac{1}{10}+\frac{1}{2}\Phi_z\right)l}{(1+\Phi_z)^2} & \dfrac{\left(\frac{r_y}{l}\right)^2\left(\frac{2}{15}+\frac{1}{6}\Phi_z+\frac{1}{3}\Phi_z{}^2\right)l^2}{(1+\Phi_z)^2} & \\ \dfrac{\left(\frac{r_z}{l}\right)^2\left(\frac{1}{10}-\frac{1}{2}\Phi_y\right)l}{(1+\Phi_y)^2} & & & \dfrac{\left(\frac{r_z}{l}\right)^2\left(-\frac{1}{30}-\frac{1}{6}\Phi_y+\frac{1}{6}\Phi_y{}^2\right)l^2}{(1+\Phi_y)^2} & \dfrac{\left(\frac{r_z}{l}\right)^2\left(-\frac{1}{10}+\frac{1}{2}\Phi_y\right)l}{(1+\Phi_y)^2} & & & \dfrac{\left(\frac{r_z}{l}\right)^2\left(\frac{2}{15}+\frac{1}{6}\Phi_y+\frac{1}{3}\Phi_y{}^2\right)l^2}{(1+\Phi_y)^2} \end{bmatrix}$$

$$f_{4y} = \left(-\xi^2 + \xi^3 - \frac{(\xi - \xi^2)}{2}\Phi_y\right)L \tag{4.10}$$

where

$$\xi = \frac{x}{L} \tag{4.11}$$

Notice that when $\Phi_y = 0$, (4.7)-(4.10) reduces to the beam bending shape functions

without shear deformation as given in Chapter 3, (3.27)-(3.30).

The axial displacement u_x is:

$$u_x = x_a + (-x_a + x_b)\xi \tag{4.12}$$

The lateral displacements u_y is:

$$u_y = \frac{f_{1y}y_a + f_{2y}\phi_{za} + f_{3y}y_b + f_{4y}\phi_{zb}}{1 + \Phi_y} \tag{4.13}$$

Based on the shape functions with shear deformation (4.12)-(4.13), the strain energy

due to beam bending, U_b, is [43]:

$$U_b = \frac{EA}{2}\int_0^L \left(\frac{\partial u_x}{\partial x}\right)^2 dx + \frac{EI}{2}\int_0^L \left(\frac{\partial^2 u_y}{\partial x^2}\right)^2 dx + \frac{EA}{2}\int_0^L \frac{\partial u_x}{\partial x}\left(\frac{\partial u_y}{\partial x}\right)^2 dx \tag{4.14}$$

The third integral term is the origin of geometric nonlinearity. When considering shear

effects, in addition to the strain energy due to bending, the *shear strain energy, U_s,* should

also be included in the total strain energy. The shear strain energy is defined as [52]:

$$U_s = \int_0^L \frac{V^2}{2GA_s}dx \tag{4.15}$$

where V is the shear force which is proportional to the third derivative of the lateral

displacement, u_y:

$$V = \frac{dM}{dx} = \frac{d}{dx}\left(EI_z\frac{d^2 u_y}{dx^2}\right) = EI_z\frac{d^3 u_y}{dx^3} \tag{4.16}$$

Recall the definition of shear parameter Φ_y

$$\Phi_y = \frac{E\alpha_s}{G}\left(\frac{w}{L}\right)^2 \tag{4.17}$$

the shear strain energy is then rewritten as:

$$U_s = \frac{1}{2GA_{s_y}}\int_0^L\left(EI_z\frac{d^3u_y}{dx^3}\right)^2 dx = \frac{(EI_z)^2}{2GA_{s_y}}\int_0^L\left(\frac{d^3u_y}{dx^3}\right)^2 dx = \frac{EI_z\Phi_y L^2}{24}\int_0^L\left(\frac{d^3u_y}{dx^3}\right)^2 dx \tag{4.18}$$

The total strain energy with shear deformation, U_i, is:

$$U_i = U_b + U_s \tag{4.19}$$

Applying Castigliano's theorem to (4.19), the symbolic solution for the geometric stiffness matrix with shear effect is derived in Mathematica. The stiffness matrix for 12-DOF nonlinear beam elements with shear, $[k]_s$, is given on page 162, where N is the axial force as defined in Chapter 3:

$$N = \frac{EA}{L}\Delta L \tag{4.20}$$

For slender beam, $\Phi_y = 0$, the stiffness matrix simply reduces to the nonlinear stiffness matrix without shear effect as given in Chapter 3.

4.1.2 Guidelines for Extension

From the above discussion, we see that the beam model is extensible to new physical effects as long as the model framework required by the new effects, including the pin definition and the algorithm structure, is compatible with the original beam model. Similarly, other NODAS models are also extensible to new physical effects, given that this compatibility is satisfied. Only additional physics from the same domain can be added. If the desired physics involves interactions with a new physical domain, then new connection pins associated with new disciplines have to be created. For example, the curling effect in the composite beams caused by thermal expansion requires connection pins for the thermal domain. This leads to the issue of extensibility to new physical domains, which will be discussed in section 4.4.

$[k]_s$: nonlinear stiffness matrix with shear

$$[k]_S = \begin{bmatrix}
\frac{Etw}{l} & & & & & & -\frac{Etw}{l} & & & & & \\
& \frac{12EI_z}{l^3(1+\Phi_y)} & & & & \frac{6EI_z}{l^2(1+\Phi_y)} & & -\frac{12EI_z}{l^3(1+\Phi_y)} & & & & \frac{6EI_z}{l^2(1+\Phi_y)} \\
& & \frac{12EI_y}{l^3(1+\Phi_z)} & & -\frac{6EI_y}{l^2(1+\Phi_z)} & & & & -\frac{12EI_y}{l^3(1+\Phi_z)} & & -\frac{6EI_y}{l^2(1+\Phi_z)} & \\
& & & \frac{GJ_t}{l} & & & & & & -\frac{GJ_t}{l} & & \\
& & -\frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(4+\Phi_z)EI_y}{l(1+\Phi_z)} & & & & \frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(2-\Phi_z)EI_y}{l(1+\Phi_z)} & \\
& \frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(4+\Phi_y)EI_z}{l(1+\Phi_y)} & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(2-\Phi_y)EI_z}{l(1+\Phi_y)} \\
-\frac{Etw}{l} & & & & & & \frac{Etw}{l} & & & & & \\
& -\frac{12EI_z}{l^3(1+\Phi_y)} & & & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & \frac{12EI_z}{l^3(1+\Phi_y)} & & & & -\frac{6EI_z}{l^2(1+\Phi_y)} \\
& & -\frac{12EI_y}{l^3(1+\Phi_z)} & & \frac{6EI_y}{l^2(1+\Phi_z)} & & & & \frac{12EI_y}{l^3(1+\Phi_z)} & & \frac{6EI_y}{l^2(1+\Phi_z)} & \\
& & & -\frac{GJ_t}{l} & & & & & & \frac{GJ_t}{l} & & \\
& & -\frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(2-\Phi_z)EI_y}{l(1+\Phi_z)} & & & & \frac{6EI_y}{l^2(1+\Phi_z)} & & \frac{(4+\Phi_z)EI_y}{l(1+\Phi_z)} & \\
& \frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(2-\Phi_y)EI_z}{l(1+\Phi_y)} & & -\frac{6EI_z}{l^2(1+\Phi_y)} & & & & \frac{(4+\Phi_y)EI_z}{l(1+\Phi_y)}
\end{bmatrix}$$

$$+N\begin{bmatrix}
0 & & & & & & 0 & & & & & \\
& \frac{6+10\Phi_y+5\Phi_y^2}{5l(1+\Phi_y)^2} & & & & & & & & & & \\
& & \frac{6+10\Phi_z+5\Phi_z^2}{5l(1+\Phi_z)^2} & & & & & & & & & \\
& & & 0 & & & & & & 0 & & \\
& & -\frac{1}{10(1+\Phi_z)^2} & & \frac{8+10\Phi_z+5\Phi_z^2}{60(1+\Phi_z)^2}l & & & & & & & \\
& \frac{1}{10(1+\Phi_y)^2} & & & & \frac{8+10\Phi_y+5\Phi_y^2}{60(1+\Phi_y)^2}l & & & & & & \\
0 & & & & & & 0 & & & & & \\
& -\frac{6+10\Phi_y+5\Phi_y^2}{5l(1+\Phi_y)^2} & & & & -\frac{1}{10(1+\Phi_y)^2} & & \frac{6+10\Phi_y+5\Phi_y^2}{5l(1+\Phi_y)^2} & & & & \\
& & -\frac{6+10\Phi_z+5\Phi_z^2}{5l(1+\Phi_z)^2} & & \frac{1}{10(1+\Phi_z)^2} & & & & \frac{6+10\Phi_z+5\Phi_z^2}{5l(1+\Phi_z)^2} & & & \\
& & & 0 & & & & & & 0 & & \\
& & -\frac{1}{10(1+\Phi_z)^2} & & \frac{2+10\Phi_z+5\Phi_z^2}{60(1+\Phi_z)^2} & & & & \frac{1}{10(1+\Phi_z)^2} & & \frac{8+10\Phi_z+5\Phi_z^2}{60(1+\Phi_z)^2}l & \\
& \frac{1}{10(1+\Phi_y)^2} & & & & \frac{2+10\Phi_y+5\Phi_y^2}{60(1+\Phi_y)^2} & & -\frac{1}{10(1+\Phi_y)^2} & & & & \frac{8+10\Phi_y+5\Phi_y^2}{60(1+\Phi_y)^2}l
\end{bmatrix}$$

The basic steps for the extension of an existing NODAS model to include new physical effects are given below:

1. study the principle of the new physical effect;

2. examine whether the new physics is compatible with the existing model's physical domains, across and through variables, user-specified parameters, matrix formats and algorithm structures. The criterion is: the modification must be an "addition" rather than a "replacement".

3. If the new physics is not compatible, consider other extension types.

4. If compatible, implement the new physical effects in the original model framework, debug and verify the model accuracy. The new model must reduce to the original model when the new physical effect is neglected.

4.2 New Design Styles

As stated in Chapter 2, design styles come in innumerable varieties. For instance, while many devices use the Manhattan geometric style, curved design styles are employed in some design cases [78]. Therefore, the cell library should be extensible to new design styles. The modeling of beams with sidewall angles is described in this section as an example of this type of extensibility. A general extension routine is given to help create models for new-style elements based on the prototype models.

4.2.1 Example: Beam with Sidewall Angles

The beam models discussed in previous sections are all for beam elements with rectangular cross-sections. Although the rectangular cross-section is the most widely used in suspended MEMS designs because of its simplicity, other types of beam cross-section exist either as a specially wanted design style [79] or as an unwanted side-product of the fabrication process [80]. Sidewall angles caused by overetching is one such example.

Figure 4.1 shows a rectangular beam cross-section (a) and a trapezoidal beam cross-
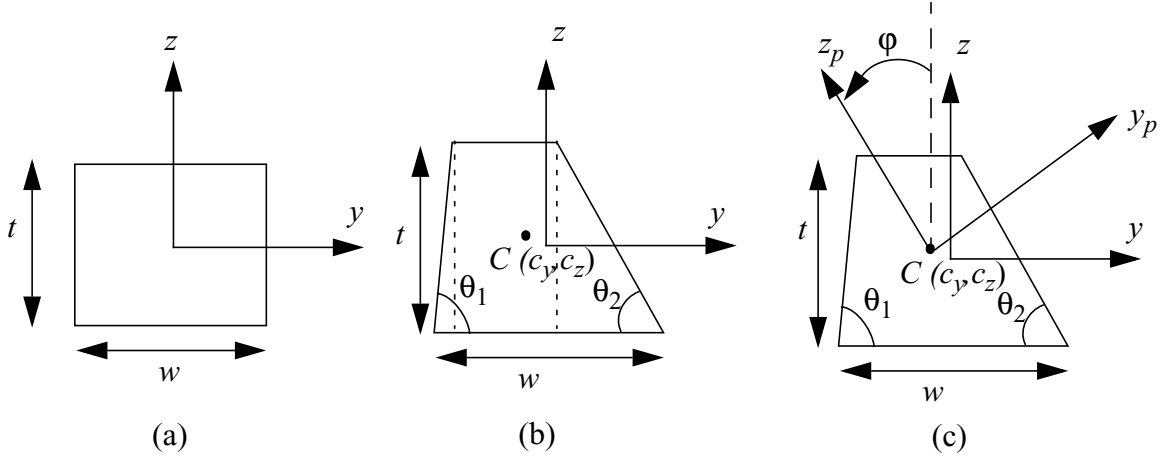
Figure 4.1: (a) Rectangular beam cross-section (b) centroid of trapezoidal beam cross-section (c) principle axes

section due to overetching (b-c). We use two sidewall angles, θ_1 and θ_2, to describe the trapezoidal cross-section. The sidewall angles could have multiple effects on the device performance depending on the specific location of this problem on the device. If it's on a beam working as part of the suspension spring, then the moment of inertia and thus the stiffness of the suspension will be affected; if it's on a beam working as a comb finger, then not only the rigidity of the finger but also the capacitance between the comb fingers will change. In this section, we will discuss the influence of the sidewall angles on the moment of inertia of the beam element.

There are two types of moment of inertia for a beam with *x*-direction as the longitudinal direction: the planar moment of inertia (I_z and I_y) and the torsional moment of inertia, J_t.

4.2.1.1 Planar Moment of Inertia

We know that for a beam with rectangular cross-section, the planar moment of inertia is simply [42]:

$$I_z = \frac{1}{12}tw^3 \tag{4.21}$$

and

$$I_y = \frac{1}{12}wt^3 \tag{4.22}$$

where t is the thickness and w is the width, as shown in Figure 4.1(a). This is the moment of inertia about the centroid in the principle-axes frame. For a rectangular cross-section, y and z axes are the principle axes and the centroid falls onto the y-axis and z-axis, therefore the moment of inertia about centroid in the principle-axes are the same as those about the coordinate axes. When non-90-degree sidewall angles are considered, the moments of inertia about the principle axes are no longer the same as those about the coordinate axes. To obtain the moment of inertia about the coordinate axes, which are the variables used in the stiffness matrix, the moment of inertia about the principle axes must first be calculated and then transformed into the inertia about coordinate axes according to the relation between the principle axes and the coordinate axes.

According to the definition of centroid, the coordinates of the centroid for the trapezoid, (c_y, c_z), are calculated by dividing the irregular trapezoidal area (called "composite area") into two triangles at sides and one rectangle at the center (called "component area"), as shown in Figure 4.1 (b), calculating the centroid for each component area and then obtain the centroid for the composite trapezoidal area [52]:

$$c_y = \frac{\sum y_i A_i}{\sum A_i} \tag{4.23}$$

and

$$c_z = \frac{\sum z_i A_i}{\sum A_i} \tag{4.24}$$

where A_i is the area of the ith component area with centroidal coordinates y_i and z_i:

The planar moments of inertia about the centroid are thus obtained [52]:

$$\bar{I}_y = \int (z - c_z)^2 dA \tag{4.25}$$

$$\bar{I}_z = \int (y - c_y)^2 dA \tag{4.26}$$

and

$$\overline{I_{yz}} = \int (y - c_y)(z - c_z)dA \tag{4.27}$$

As shown in Figure 4.1(b), with non-90-degree sidewall angles, the principle axes forms an angle, φ, with the coordinate axes. This angle are obtained from the moments of inertia in the principle-axes frame as follows [81]:

$$\varphi = \frac{1}{2}\operatorname{atan}\left(-2\frac{\overline{I_{yz}}}{\overline{I_y} - \overline{I_z}}\right) \tag{4.28}$$

Based on this angle, the planar moments of inertia in principle-axes frame are then transformed into the moments of inertia in the coordinate-axes frame:

$$I_y = \frac{\overline{I_y} + \overline{I_z}}{2} + \frac{\overline{I_y} - \overline{I_z}}{2}\cos(2\varphi) - \overline{I_{yz}}\sin(2\varphi) \tag{4.29}$$

$$I_z = \frac{\overline{I_y} + \overline{I_z}}{2} - \frac{\overline{I_y} - \overline{I_z}}{2}\cos(2\varphi) + \overline{I_{yz}}\sin(2\varphi) \tag{4.30}$$

When $\theta_1 = \theta_2 = 90°$, $\varphi = 0$, I_y reduces to $\overline{I_y}$, I_z reduces to $\overline{I_z}$ and $\overline{I_{yz}} = 0$.

4.2.1.2 Torsional Moment of Inertia

The torsional moment of inertia with sidewall angles is derived using the Griffith and Taylor method given in [82].

Step 1. The trapezoidal beam cross-section does not have long projections as shown in Figure 4.2(a) [82], thus the entire cross-section will be treated as one section without partition into multiple portions.

Step 2. Determine the area A_0 and the outer perimeter P_0.

As illustrated in Figure 4.2(b), the length of the top edge of the trapezoidal, q, is:

$$q = w - t\cot\theta_1 - t\cot\theta_2 \tag{4.31}$$

thus the area of the cross-section, A_0, is:

$$A_0 = \frac{1}{2}t(w + q) \tag{4.32}$$

and the outer perimeter, P_0, is:

$$P_0 = w + q + \frac{t}{\sin\theta_1} + \frac{t}{\sin\theta_2}$$

(4.33)

Step 3. Inscribe a circle of the largest possible radius R, which touches the original boundary of the cross-section at three (or more) points.

Since the sidewall angles being considered are caused by overetching, the top edge is always shorter than the bottom edge, thus the largest possible inscription circle must touch the bottom edge while not necessarily touching the top edge. Figure 4.2(b-e) shows the possible cases of the inscription. In (b), the circle only touches the bottom edge, while in (c) (d) and (e), the circle touches both the bottom edge and the top edge. For case (c), (d) and (e), the radius R is simply half of the beam thickness: $R = \frac{t}{2}$. For case (b), R is calculated as follows:
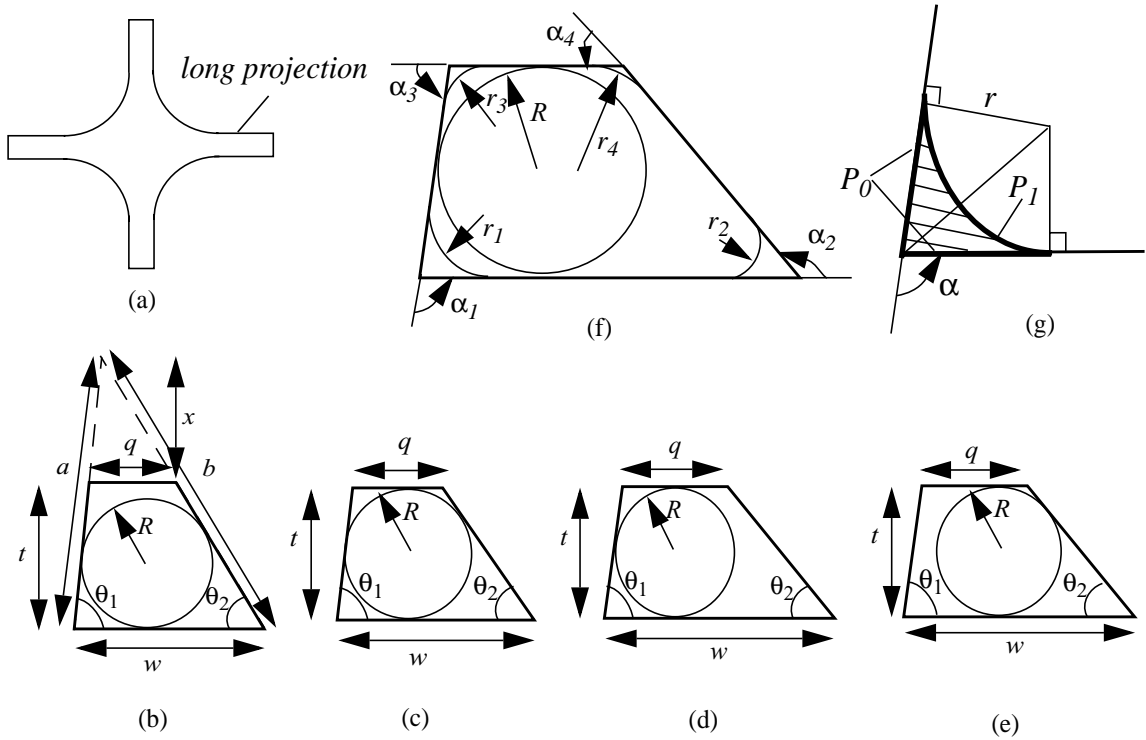
Figure 4.2: (a) cross-section with long projection (b) (c) (d) (e) possible cases of inscription for the trapezoidal beam cross-section without long projection (f) round off of corners (g) perimeter and area of the round-off corner

$$R = \sqrt{\frac{(s-a)(s-b)(s-w)}{s}} \tag{4.34}$$

where

$$x = \frac{qt}{w-q} \tag{4.35}$$

$$a = \frac{x+t}{\sin\theta_1} \tag{4.36}$$

$$b = \frac{x+t}{\sin\theta_2} \tag{4.37}$$

$$s = \frac{1}{2}(a+b+w) \tag{4.38}$$

Step 4. Round off the outer corners with arcs of circles of radius r.

A set of corner angle α, as shown in Figure 4.2 (f), is given in [82] with corresponding values of r/R. To obtain the general formula of r/R for arbitrary corner angles, polynomial curve fitting to the given set of values is performed in Matlab and results in:

$$\frac{r}{R} = 0.995 - 0.00208\alpha - 6.194 \times 10^{-5}\alpha^2 + 2.66 \times 10^{-7}\alpha^3, (0 < \alpha < \pi) \tag{4.39}$$

For the trapezoidal cross-section in Figure 4.2(f), the corner angles are: $\alpha_1 = \pi\text{-}\theta_1$, $\alpha_2 = \pi\text{-}\theta_2$, $\alpha_3 = \theta_1$, and $\alpha_4 = \pi\text{-}\theta_2$. Radius of the round-off arc of the circle for each corner, r_i, are subsequently obtained from (4.39).

Step 5. Determine the factor λ needed in the calculation of torsional moment of inertia.

A table of $RP_0/2A_0$ with corresponding values of λ is given in [82]. Similarly, formula of λ for arbitrary values of $RP_0/2A_0$ is obtained in Matlab from polynomial curve fitting to the given set of values:

$$\lambda = -0.422 + 2.83\frac{RP_0}{2A_0} - 1.22\left(\frac{RP_0}{2A_0}\right)^2 - 0.190\left(\frac{RP_0}{2A_0}\right)^3 \tag{4.40}$$

Step 6. Determine the perimeter P_1 and the area A_1 of the rounded-off figure.

As shown in Figure 4.2(g), P_1 is obtained by subtracting the length of the corners from P_0 and adding the length of the round-off arc:

$$P_1 \;=\; P_0 - 2\left(r_1\cot\left(\frac{\pi-\theta_1}{2}\right) + r_2\cot\left(\frac{\pi-\theta_2}{2}\right) + r_3\cot\frac{\theta_1}{2} + r_4\cot\frac{\theta_2}{2}\right)$$
$$+\, r_1\theta_1 + r_2\theta_2 + r_3(\pi-\theta_1) + r_4(\pi-\theta_2) \tag{4.41}$$

A_1 is obtained by subtracting the shaded area from the cross-section area A_0:

$$A_1 \;=\; A_0 - r_1{}^2\left(\cot\frac{\pi-\theta_1}{2} - \frac{\theta_1}{2}\right) - r_2{}^2\left(\cot\frac{\pi-\theta_2}{2} - \frac{\theta_2}{2}\right)$$
$$-\, r_3{}^2\left(\cot\frac{\theta_1}{2} - \frac{\pi-\theta_1}{2}\right) - r_4{}^2\left(\cot\frac{\theta_2}{2} - \frac{\pi-\theta_2}{2}\right) \tag{4.42}$$

Step 7. Calculate the torsional moment of inertia, J_t.

$$J_t \;=\; \frac{1}{2}A_1\lambda\left(\frac{2A_1}{P_1}\right)^2 \tag{4.43}$$

When $\theta_1 = \theta_2 = 90°$,

$$r_1 \;=\; r_2 \;=\; r_3 \;=\; r_4 \;=\; 0.5R\,,\; R = \min\,(t,w)/2 \tag{4.44}$$

$$P_1 \;=\; P_0 - 4\times2\times0.5\times1\times R + 4\times0.5\times\frac{\pi}{2}\times R \;=\; P_0 - (4-\pi)\times R \tag{4.45}$$

$$A_1 \;=\; A_0 - 4\times(0.5\times R)^2\times\left(\cot\frac{\pi}{4} - \frac{\pi}{4}\right) \;=\; A_0 - \left(1 - \frac{\pi}{4}\right)R^2 \tag{4.46}$$

As an example, if $t = w = 2$ μm, then $R = 1$ μm, $P_0 = 8$ μm, $A_0 = 4$ μm^2, $RP_0/2A_0 = 1$, $\lambda = 1$, $P_1 = 7.142$ μm, $A_1 = 3.785$ μm^2, so we have $J_t = 2.127$ μm^4 = 2.127×10^{-24} m^4. This value is about 5% different from the torsional moment of inertia for a rectangular cross-section of the same size given by the formula in [83][75], which is 2.248×10^{-24} m^4.

The torsional moment of inertia of beams with more complicated shapes of cross-sections can be obtained by partitioning the entire cross-section into several separate portions without projection [82], following the same methodology as described above to get the moment of inertia for each portion, then summing up to get the total moment of inertia.

For beams with composite layers such as CMOS-MEMS beams, normally only the top metal layer, which works as the etching mask, is subject to overetching and has non-90-degree sidewall angles. Other layers still have rectangular cross-sections. For those beams,

the planar and torsional moments of inertia for the top layer should first be calculated following the procedure given in this section and then approximate the top layer with a rectangle to find out the effective moment of inertia for the composite beam [84].

4.2.2 Guidelines for Extension

Depending on the extent of compatibility with the prototype models, extension to new styles could be easy or complicated.

1. First check whether the new style is compatible with the algorithm structure of the prototype model. If yes, go to step 2, if not, go to step3.

2. If compatible, (for example, the beam with sidewall angles follows the same shape functions as the beam with straight sidewalls. The only difference is the expression for the moments of inertia), then identify the different variables, parameters or matrix elements, figure out the new formula, add them into the prototype model, verify and check the model for consistency with simple cases.

3. If not compatible, then separate models have to be created for the new style. For example, curved beams have different shape functions from the straight beams, hence both the stiffness matrix and the mass matrix have to be rederived. However, since the fundamental mechanics stays the same for different design styles, there are still many things which can be inherited from the prototype models, such as the pin definition, the algorithm structure and the methodology for matrix derivation.

4.3 New Processes

The models mentioned in previous sections are all for single conductor layer processes. More complicated processes raise specific modeling issues. The models in this thesis should be extensible to new processes. In this section, we will take the beam models for CMOS-MEMS process as an example.

4.3.1 Example: Beam Model for CMOS-MEMS

CMOS-MEMS devices are obtained by post-process release following conventional CMOS IC fabrication [85]. Figure 4.3 shows the main steps of this process. In this process, the microstructures are fabricated simultaneously with the electronics. The top metal layer has three roles: as the structural mask during release, as a protection mask for electrical circuits outside of mechanical structures, and as the electrical wiring on top of the mechanical structures. Compared to many single structural layer processes, CMOS-MEMS has an advantage in the inherent compatibility with electronic circuitry.

The distinguishing characteristic of the CMOS-MEMS process is that the cross-sectional structure is a stack of metals and oxides, as shown in Figure 4.3. Take the beam element as an example. In a three-layer interconnect process, there are 14 types of beams with different combinations of the metal and oxide layers, as listed in Figure 4.4. The beam type is identified by a 4-digit number, with each digit representing one specific electrical conducting layer (metal3, metal2, metal1 or polysilicon). The value of each digit indicates
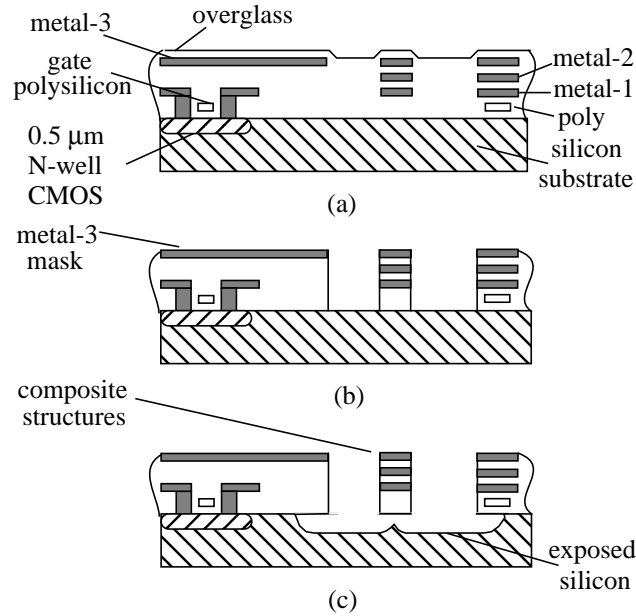
Figure 4.3: CMOS-MEMS process (a) CMOS chip after fabrication (b) anisotropic RIE removes dielectric (c) isotropic RIE undercuts silicon substrate

beam_1111   1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010

metal-3
metal-2
metal-1
poly
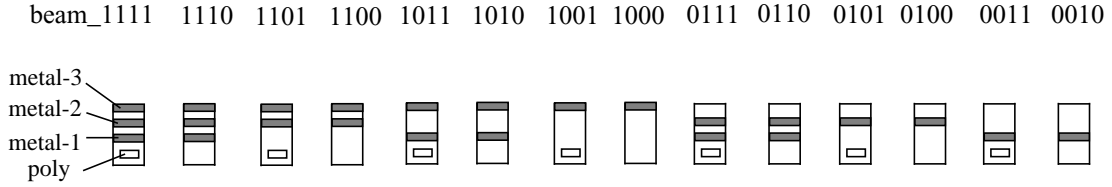
Figure 4.4: 14 types of beams in a three-layer interconnect CMOS-MEMS process

the existence of the corresponding layer: "1" means "yes" and "0" means "no". For instance, beam_1010 means the beam is composed of only metal3, metal1 and the corresponding oxide layers. The composite-layer structure of CMOS-MEMS arises several specific issues in modeling, which will be discussed in the following sections.

4.3.1.1 Pin Definition and Flag Parameters

As described in Chapter 2, bus pins are defined for each physical discipline: kinematic, rotational, electrical, etc. For CMOS-MEMS, we deal with the mechanics of the composite-layer structures by figuring out the equivalent geometric, process and material property parameters and then treat the composite-layer structure as a single-layer structure. The equivalent parameters are either theoretically calculated or experimentally extracted. Therefore, the definitions for mechanical pins stays the same as for single structural layer elements.

The definition for electrical pins is different, as each metal or polysilicon layer in CMOS-MEMS is a conductor and has to be treated uniquely. Thus, bus pins are also defined for the electrical terminals, as shown in Figure 4.5. The electrical pin has four dimensions,

Beam

$xyz_a[0:2]$

$\phi xyz_a[0:2]$

$v_a[0:3]$

a

$l = 100 \ \mu m$
$w = 2 \ \mu m$
$t = 2 \ \mu m$
$X_c = 50 \ \mu m$
$Y_c = 0 \ \mu m$
$\gamma = 0°$

b

$xyz_b[0:2]$

$\phi xyz_b[0:2]$

$v_b[0:3]$
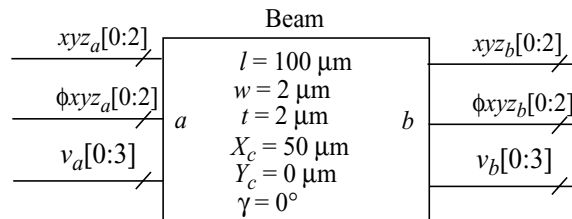
Figure 4.5: Bus pins for CMOS-MEMS beam element

each corresponding to one specific conducting layer.

To identify the type of the beam, user-specified flag parameters, *flag_m3*, *flag_m2*, *flag_m1*, and *flag_poly*, are defined. Their values are either "1" or "0", indicating the existence of the corresponding layers. These flag parameters are used for the calculation of equivalent mechanical parameters and for the conditional inclusion of resistances between node *a* and *b* on the corresponding layers.

4.3.1.2 Equivalent Parameters

The major equivalent parameters are the effective thickness, the effective mass density, and the effective Young's Modulus.

The effective thickness, t_{eff}, is calculated based on the layer combination and the process parameters for each layer. For example, in one specific foundry 0.5 μm CMOS process, there are three metal layers and one polysilicon layer, each having a fixed nominal value of thickness (and neglecting the manufacturing variation in the layer thickness). The oxide layer thickness varies with the presence or absence of specific conductor layers underneath due to the CMP process. For instance, the oxide between metal2 and metal1 is thinner when there is polysilicon underneath. The detailed layer thickness information is hardcoded into the header file *process.h*, which is shared by every NODAS model.

The effective thickness is calculated as:

$$
\begin{aligned}
t_{eff} = {} & f_{m3}t_{m3} + f_{m2}t_{m2} + f_{m1}t_{m1} + f_{poly}t_{poly} + f_{m3}f_{m2}t_{m3m2} \\
& + f_{m3}(1-f_{m2})f_{m1}f_{poly}t_{m3m1overpoly} + f_{m3}(1-f_{m2})f_{m1}(1-f_{poly})t_{m3m1overfield} \\
& + f_{m3}(1-f_{m2})(1-f_{m1})f_{poly}t_{m3poly} + f_{m3}(1-f_{m2})(1-f_{m1})(1-f_{poly})t_{m3sub} \\
& + f_{m2}f_{m1}f_{poly}t_{m2m1overpoly} + f_{m2}f_{m1}(1-f_{poly})t_{m2m1overfield} \\
& + f_{m2}(1-f_{m1})f_{poly}t_{m2poly} + f_{m2}(1-f_{m1})(1-f_{poly})t_{m2sub} \\
& + f_{m1}f_{poly}t_{m1poly} + f_{m1}(1-f_{poly})t_{m1sub} + f_{poly}t_{polysub}
\end{aligned}
\tag{4.47}
$$

where for simplicity, f_{m3} represents the flag parameter for metal 3, *flag_m3*, t_{m3m2} represents thickness of the oxide layer between metal3 and metal2, *t_m3_m2*, and so on.

Similarly, the effective mass density is calculated based on the density of the metal, polysilicon and oxide and the specific layer combination. Approximately, the layers are assumed to have a uniform vertical overetch which leads to a variation in the beam width but keeps the sidewalls straight. In addition, designers sometimes intentionally make the lower layers (metal2, metal1 or polysilicon layers) narrower than the top most metal3 layer (called "cut-in"), in order to make sure that the lower layers are safely enclosed by the oxide even if the layers are misaligned due to manufacturing variations.

The effective mass density, ρ_{eff}, is calculated as:

$$
\begin{aligned}
\rho_{eff} = (\\
\rho_{metal}(f_{m3}t_{m3}(w - \delta w) + f_{m2}t_{m2}((w - \delta w) - 2\Delta_{m2}) + f_{m1}t_{m1}((w - \delta w) - 2\Delta_{m1})) \\
+ \rho_{poly}f_{poly}t_{poly}((w - \delta w) - 2\Delta_{poly}) \\
+ \rho_{oxide}(f_{m2}t_{m2}(2\Delta_{m2}) + f_{m1}t_{m1}(2\Delta_{m1}) + f_{poly}t_{poly}(2\Delta_{poly})) \\
+ \rho_{oxide}(t - (f_{m3}t_{m3} + f_{m2}t_{m2} + f_{m1}t_{m1} + f_{poly}t_{poly}))(w - \delta w)) \\
/((w - \delta w)t_{eff})
\end{aligned}
\tag{4.48}
$$

where δw is the overetch, Δ_{m1} is the cut-in of metal1 layer, and so on.

The effective Young's Modulus, E_{eff}, can be derived from the Young's Modulus of each individual layer and the layer combination. However, since the exact value of the Young's Modulus of each individual layer is not available, experimentally extracted values of Young's Modulus for each type of beam is instead used in the model [86].

For design flexibility, the process parameters including the thickness of each layer and mass density of each type of material are provided as default values in *process.h* file, as well as user-specified instance parameters. The defaults will be overridden when a value is specified by the user and that value is specific to the edited instance. This allows the model to be used for designs in other similar processes. The user can adjust the process parameters and layer combinations properly to make equivalent models for the new process.

4.3.1.3 Submodules

There are two ways of manipulating the 14 types of CMOS-MEMS beams. One is to make a general beam model and use flag parameters to identify the beam type, as described above. The other is to make individual models for each types of beam. Since the beam mechanics is the same for all the 14 types of beams, submodules are used to ease the modeling and the maintenance of the models. In addition, as the electrical behavior of the conducting layers is similar and the electrical behavior is independent with the mechanical behavior, the model for the electrical behavior of the CMOS-MEMS beam element is also implemented as a submodule.

The mechanical submodule (*beam_mech*) contains no electrical behavior, thus only have mechanical nodes. It models single-layer beam mechanics. The identification of beam type and calculation of the equivalent geometric sizes and materials properties are all performed in the parent module, as shown in the Verilog-A code segment of the parent module given below. The equivalent parameters are then passed down to the mechanical submodule.

```
'include "../../process.h"    //header files
module beam (phia, phib, xa, xb, va, vb);  //both mechanical and electrical pins
inout [0:2] phia; // bus pin definition
rotational [0:2] phia;
// definition of flag parameters
parameter integer flag_m3 = 1, flag_m2 = 1, flag_m1 = 1, flag_poly = 1;
// calculation of equivalent parameters
parameter real E = flag_m3*flag_m2*flag_m1*flag_poly*'default_E_cmos_1111...
parameter real thickness = flag_m3*t_m3 + flag_m2*t_m2...
parameter real density = ('default_den_metal*( flag_m3*t_m3*(w-overetch)...
// call mechanical submodule, pin mapping and variable mapping
beam_mech # (.l(l), .w(w), .thickness(thickness), .E(E), .density(density)... )
      beam_mech_1 (phia[0:2], phib[0:2], xa[0:2], xb[0:2]); // only mechanical pins
```

The electrical submodule (*beam_elec*) defines the electrical behavior of the beam. As shown in Figure 4.6, the resistance of each existing conductor layer is modeled as a resistor between the corresponding electrical nodes, with the assumption of rectangular cross-

sections. For example, r_{m3} is the resistance between node v_a [3] and v_b [3], defined as

$$r_{m3} = R_{m3}\frac{L}{w} \tag{4.49}$$

where R_{m3} is the sheet resistance of metal3 layer. The overlap capacitance between adjacent layers are simply modeled as parallel-plate capacitors, based on the oxide thickness data provided by the foundry. For example, C_{m3_m2} is the capacitance between metal3 and metal2 layers defined as

$$C_{m3m2} = \frac{\varepsilon LW}{t_{m3m2}} \tag{4.50}$$

where ε is the permittivity constant of the oxide layer and t_{m3m2} is the thickness of the oxide. The capacitance is lumped to the beam ends by simply applying half of the capacitance to each end. Fringe effects are left as future development.

If a certain layer doesn't exist in a specific type of beam, physically there are no resistors or capacitors attached to the corresponding nodes, and thus there are no physical equations describing the relation between the across and through variables at these nodes. However, the Verilog-A syntax does not allow pins to have no associated equations. Therefore, a fictitious resistor of a very large resistance ($R_\infty = 10^{15}$ Ω) is applied to the nodes for non-existing layers, as an approximation of the infinite resistance.
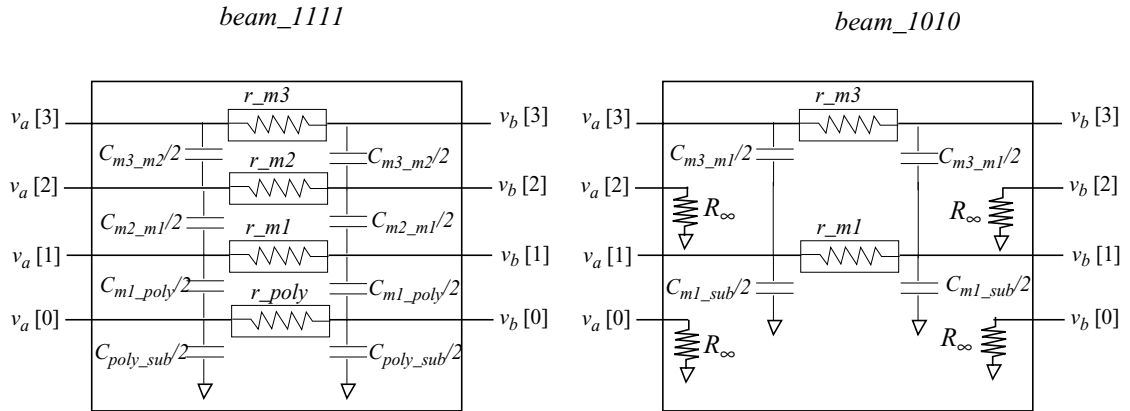
Figure 4.6: Electrical submodule for CMOS-MEMS beam elements

The moment of inertia of the composite CMOS-MEMS beam is obtained from the effective thickness and the experimentally extracted effective Young's Modulus, by treating the stacked-layer cross-section as a single-layer cross-section. More accurate formula could be obtained by considering the angled sidewall of the top layer. Since the top metal layer is normally only a small position of the entire stack, the error introduced by the approximation is acceptable. In addition, the misalignment of the multiple layers due to manufacturing variations also affects the moment of inertia. The misalignment is modeled by properly modifying the constructive variables and equations on basis of the prototype model [71].

4.3.2 Guidelines for Extension

The extension to new processes are restricted to processes which are compatible with the prototype process in the aspect of modeling. The prototype model library is based on single-structure-layer processes for suspended MEMS. As shown above, the models are extensible to CMOS-MEMS because it is one of the suspended MEMS processes thus it can be modeled by the same set of atomic elements and the special issues caused by the composite-layer structure can be solved by using equivalent single-layer parameters. The prototype models are also extensible to DRIE CMOS-MEMS process which is a combination of CMOS-MEMS and deep etching of silicon substrate [88]. However, for some other processes such as microfluidics processes, different sets of atomic models based on different physical principles are required, the prototype models in this thesis are thus not extensible to those processes. The parameterized lumped behavioral modeling methodology should still be extensible, though.

The guideline for extension to a new process compatible with the prototype process is given below:

1. Obtain the process data of the new process, including layer compositions, geometric parameters (layer thickness, design constraints (overetch, cut-in, etc.)), material properties (mass density, Young's modulus, resistivity, etc.). In cases where some of the data is not

available from the foundry, test structures have to be fabricated for experimental extraction. Parameter extraction examples for CMOS-MEMS are available from peer students' works at Carnegie Mellon University [86][87].

2. Examine the similarity and the difference between the new process and the prototype process, determine what can be inherited, what needs to be modified and what has to be created, including pin definitions, equivalent parameters, etc.

3. Calculate the new variables based on the process parameters of the new process.

4.4 New Physical Domains

Physical domains involved in MEMS are not restricted to mechanical, electrical and electrostatic domains. For instance, different thermal expansion coefficients between the layer stacks of CMOS-MEMS devices cause curling problems. The NODAS beam model has been successfully extended to the thermal domain to capture this effect [84]. This section will review the key steps in this extension and give a guideline for the extension to new physical domains.

4.4.1 Example: Thermal Effects in CMOS-MEMS Beam Elements

As the metal layer and the oxide layer have different TCEs, their deflections under the same temperature variation are different, leading to the curling in the CMOS-MEMS beams. For most capacitive actuation and sensing devices, the curling has negative effect on the device performance because it causes mismatch between the comb fingers thus reduces the actuation forces and sensing sensitivity. But for some devices, the curling effect is introduced on purpose to fulfill the device functionality [89][90]. In both cases, modeling the curling effect is very important in order to have a better design understanding of the curling and to account for its effect on performance.

First, a new pin associated with the thermal discipline is added to the beam, with *Temperature* as the across variable and *Power* as the through variable. This pin is where the

heat sources are applied and the temperatures are detected.

Next, the thermal and mechanical property of the composite layers are quantized. T_0 and TCEs of each layer are extracted from experimental data of beams with multiple layer combinations. Young's modulus of each individual layer is also obtained from curve-fitting on the experimental data of effective Young's modulus of 14 types of beams [84]. In the previous section, effective Young's modulus for each type is directly used in the prototype CMOS-MEMS beam model, because the composite-layer structures are treated as equivalent single-layer structures. However, when the curling effect is considered, the layers must be treated individually, as the different expansions between the layers is the source of the curling.

Stresses within each layer due to the thermal expansion are treated as axial loads applied to the two ends of each layer. Bending forces and moments for each layer are then derived and summed up under the constraints of force and moment balance, giving the total equivalent forces and moments of the composite beam. These equivalent forces and moments are then applied to the beam ends as extra external loads to join the beam dynamics equations and the system analysis [84].

Since analog HDLs inherently support co-simulations in multiple physical domains, the model library written in analog HDLs is inherently open to the modeling of effects in new physical domains.

4.4.2 Guidelines for Extension

As shown in the modeling of curling effect, the key steps for extension to new physical disciplines are:

1. If the new discipline is not defined in the existing discipine.h file, then add in the new definition.

Definition of a new discipline for Verilog-A models includes definition of the across (potential) and through (flow) variables, units and access name of the variables and the

simulation constraints such as abstol, blowup, etc. In Verilog-A, each physical discipline is allowed to have its own individual sets of tolerance options and blowup limits, which helps obtaining convergence for simulations which involve multiple disciplines with significantly different variable magnitudes. The definition of thermal discipline is given below:

```
// Thermal
// Temperature in Celsius
nature Temperature
     units    = "C";
     access   = Temp;
'ifdef TEMPERATURE_ABSTOL
     abstol   = 'TEMPERATURE_ABSTOL;
'else
     abstol   = 1e-4;
'endif
'ifdef TEMPERATURE_MAXDELTA
     maxdelta  = 'TEMPERATURE_MAXDELTA;
'endif
'ifdef TEMPERATURE_HUGE
     huge     = 'TEMPERATURE_HUGE;
'endif
'ifdef TEMPERATURE_BLOWUP
     blowup    = 'TEMPERATURE_BLOWUP;
'endif
endnature
// Power in Watts
nature Power
     units    = "W";
     access   = Pwr;
'ifdef POWER_ABSTOL
     abstol   = 'POWER_ABSTOL;
'else
     abstol   = 1e-9;
'endif
'ifdef POWER_MAXDELTA
     maxdelta  = 'POWER_MAXDELTA;
'endif
'ifdef POWER_HUGE
     huge     = 'POWER_HUGE;
'endif
'ifdef POWER_BLOWUP
     blowup    = 'POWER_BLOWUP;
'endif
endnature
```

```
// Conservative discipline
discipline thermal
      potential   Temperature;
      flow        Power;
enddiscipline
```

2. Add new terminal pins associated with the new discipline to both symbol and Verilog-A views.

3. Obtain the material properties for the new discipline either by existing handbook, theoretical calculation, or by experimental extraction.

4. Use the obtained material property parameters to model the new physics introduced by the new discipline. The new physics should be implemented in the form of equations describing the relations between the through and across variables of the new and other existing disciplines, in order to be compatible with the prototype model.

Chapter 5   Validation of the Composition Property of Suspended MEMS

As mentioned in Chapter 2, the composition property of suspended MEMS is the hypothesis of the simulation methodology of NODAS and several other MEMS CAD tools. In this chapter, simulations of a set of validation cases of this hypothesis will be presented. They are simulated in NODAS using the atomic-level elements modeled in the previous chapters. These validation examples cover a wide range of suspended MEMS applications, showing the proposed coverage ability of the small atomic-level element library. Besides validation of the composition property, the simulation of validation cases also verifies the accuracy of the atomic models, by comparing the simulation results in NODAS to results from FEA/BEA and available experimental results.

5.1 CMOS Bandpass Filter

5.1.1 Introduction

Bandpass filters are key components of transceivers in communication technology. Conventional physical implementations of bandpass filters are based on mechanical devices such as crystal resonators, or electronic devices such as transistor LC circuits. Mechanical resonators have high quality factors, but can only be interfaced with electronics at the board level, limiting miniaturization and transceiver performance. Electrical devices can be integrated, however, the performance of electronic filters is restricted by the limited quality factor of the electronics [91].

IC-compatible surface micromachining fabrication technology provides a potential solution to this problem. High-Q mechanical resonators can be implemented on-chip [91] and integrated with electronic interface circuits, forming miniaturized high-performance MEMS. To date, micro-mechanical bandpass filters, with center frequencies ranging from
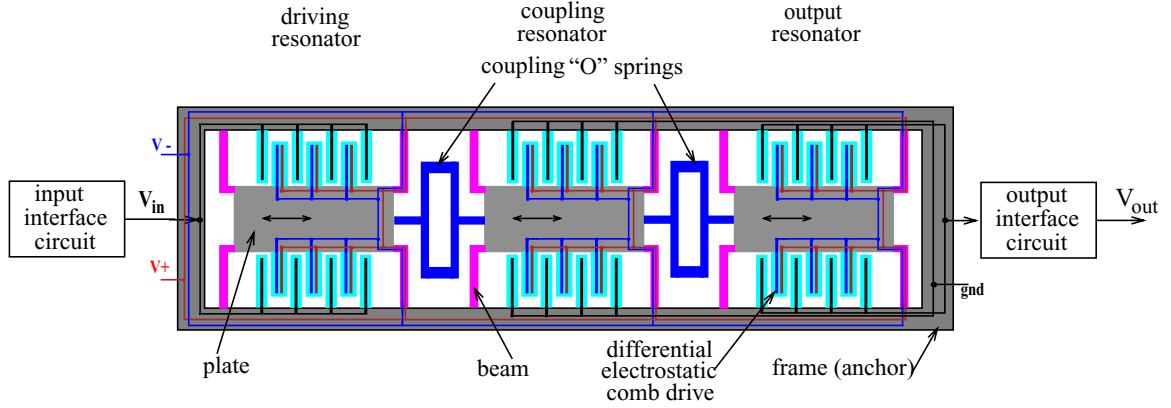
Figure 5.1: Structure of the CMOS micromechanical bandpass filter

300 kHz to 870 MHz, have been successfully implemented using a polysilicon IC-compatible surface micromachining fabrication technique [92]. This type of suspended MEMS devices has potential for revolutionizing the RF architects industry.

5.1.2 Device Topology and Working Principle

Our validation case is a CMOS micromechanical bandpass filter, first fabricated in the 0.5 μm HP CMOS process, then released at Carnegie Mellon [45].

As shown in Figure 5.1, the filter consists of three identical crab-leg resonators, coupled by "O"-shape springs. Figure 5.2 shows the operating principle of this type of mechanical filter. According to the analysis of coupled resonators [93], all the adjacent resonators vibrate in phase at the lowest natural frequency (Figure 5.2 (a)), and 180 degrees out of phase at the highest natural frequency (Figure 5.2 (c)). Resonances between the
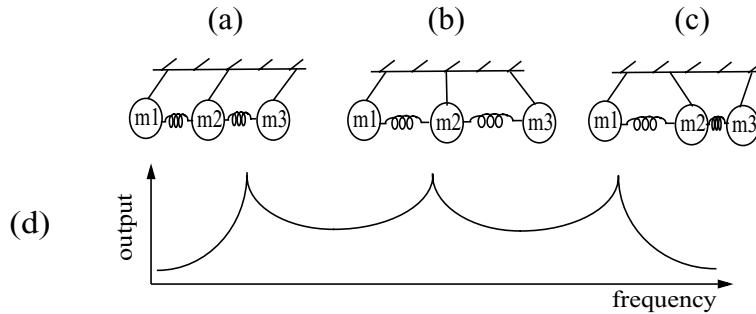
Figure 5.2: Three resonance modes (a-c) and peaks (d) of the mechanical filter system

lowest and the highest natural frequencies have displacement patterns where a resonator may be in phase, out of phase, or stationary with respect to its neighbor (Figure 5.2 (b)). The three resonant peaks of this filter scatter around the resonant frequency of the single resonator, forming a passband as shown in Figure 5.2(d). The center of the passband is determined by the resonant frequency of the constitutive resonators, and the shape of the passband is determined by the stiffness of the coupling springs. The entire system includes input and output electrical interface circuits.

In the bandpass filter design shown in Figure 5.1, each of the three resonators is composed of a center proof mass, crab-leg springs, and differential electrostatic comb drives (for actuating and sensing). The three resonators resonate in the *x*-direction, and are coupled at the center plates of each resonator. The coupling spring are composed of beams with metal-2 and metal-1 only, because thinner beams are softer and therefore give narrower bandwidth.

To take advantage of the flexibility in electrical connectivity through multiple conductive layers provided by CMOS-MEMS process, differential electrostatic comb drives are employed in this filter design. Compared to conventional single-finger interdigited comb drives, differential comb drives are more efficient in electrostatic driving and sensing. Figure 5.3 shows the electrical wiring of the driving resonator through differential comb drives. The stator fingers are connected to the input voltage V_{in} with DC bias at 0 V, and the
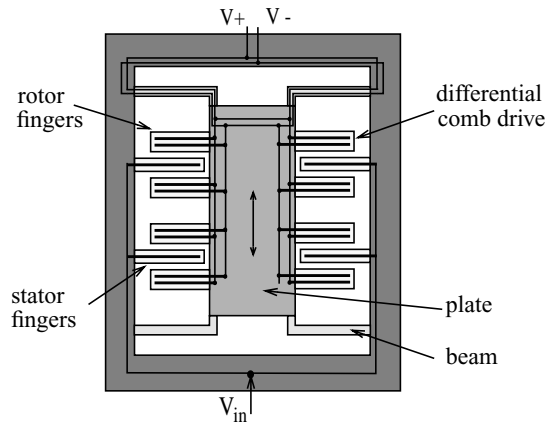
Figure 5.3: Electrical connectivity of the driving resonator.

rotor fingers are differentially DC biased at V+ and V-, therefore, the fingers theoretically will have no DC position offset. When a sinusoidal voltage is applied to the stator fingers, electrostatic forces will be generated to actuate the suspended microresonators. In this design, the comb fingers consist of a stack of three metal layers and a polysilicon layer. These layers are electrically connected to the same voltage.

The maximum resonant frequency achievable by micromechanical resonators is limited by many factors. For example, the effective finite mass of mechanical springs limits the maximum resonant frequency, and the minimum gap between fingers due to process design rules limits efficient electrostatic actuation of stiff microstructures. Thus, the application feasibility of micro-resonators in RF regions is still being studied. However, since microresonators working at 10 kHz to 1MHz are very stable and reliable, the IF region is a suitable application area. Our bandpass filter example is designed to be centered at 550 kHz, which is a typical value for IF applications.

5.1.3 Covered Physical Effects

Basic physical effects involved in this device include beam bending, moment transfer by rigid plates, and electrostatic capacitive transduction. Simulation using atomic NODAS elements will be performed to show the ability of handling these effects. The device will also be simulated at mixed-level with atomic-level element (beam) and functional-level element (comb drive) to show the interoperability of models at different levels and to show the necessity of using atomic-level models for capture of non-negligible physical effects. This validation case also serves as an example showing the extensibility of the NODAS simulation methodology to the CMOS-MEMS process.

5.1.4 Schematic Composition

The schematic of the bandpass filter created in Cadence using NODAS cells is given below. Both mechanical structures and interface circuitry are included.

The bandpass filter is modeled in two ways in NODAS. Figure 5.4 shows the first type of composition. It uses the function-level model, the comb drive model [51][71], to model the electrostatic comb drives of the resonators. In comb drive model, comb fingers are modeled as rigid bodies with infinite stiffness. The model captured the mass and the moment transfer through the comb drives. The rigid-body assumption is good for stiff comb fingers which are normally short and wide. The rest of the filter, the springs and proof masses, are modeled by atomic-level models including beams and plates.
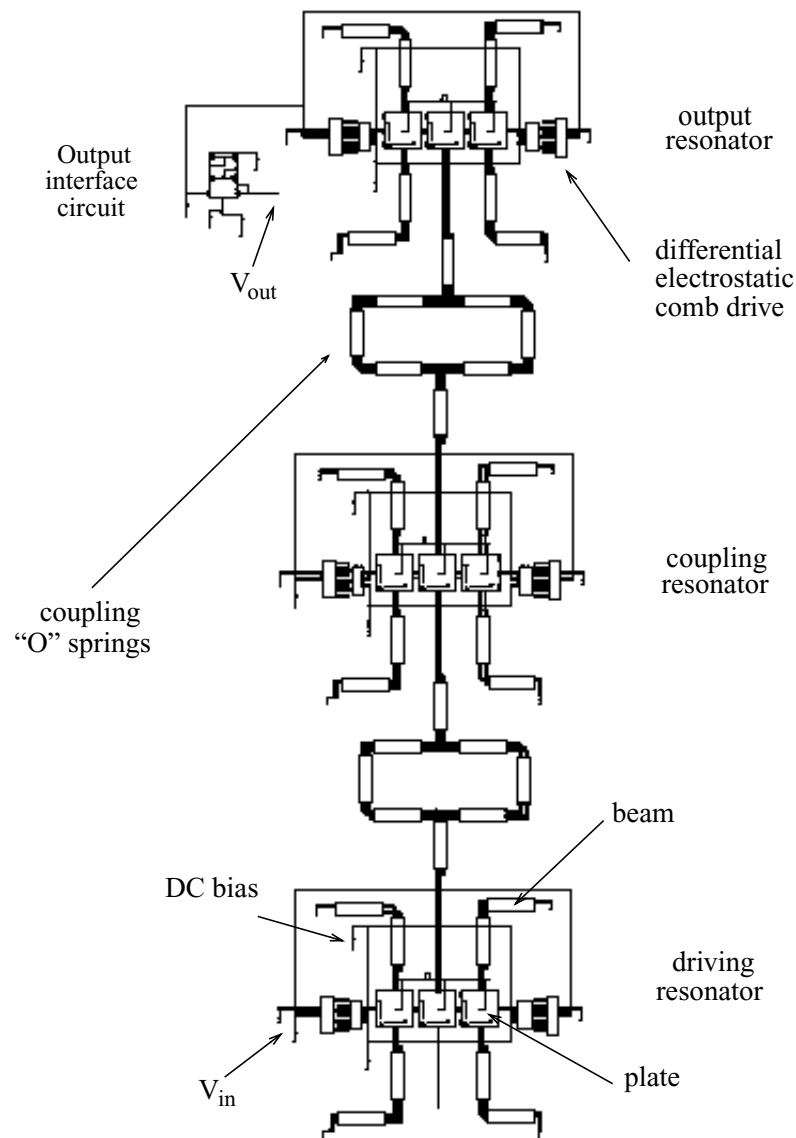
Figure 5.4: Schematic of the bandpass filter in NODAS with atomic-level elements and functional-level elements (comb drives)

Notice that in this representation, functional-level and atomic-level models co-exist in the same schematic. The mixed-level representation and simulation of the filter will be used as an example to verify the interoperability of the NODAS models between different levels.

Figure 5.5 shows the second type of composition. It uses purely atomic-level models. The electrostatic comb drives are composed by a combination of beams and electrostatic
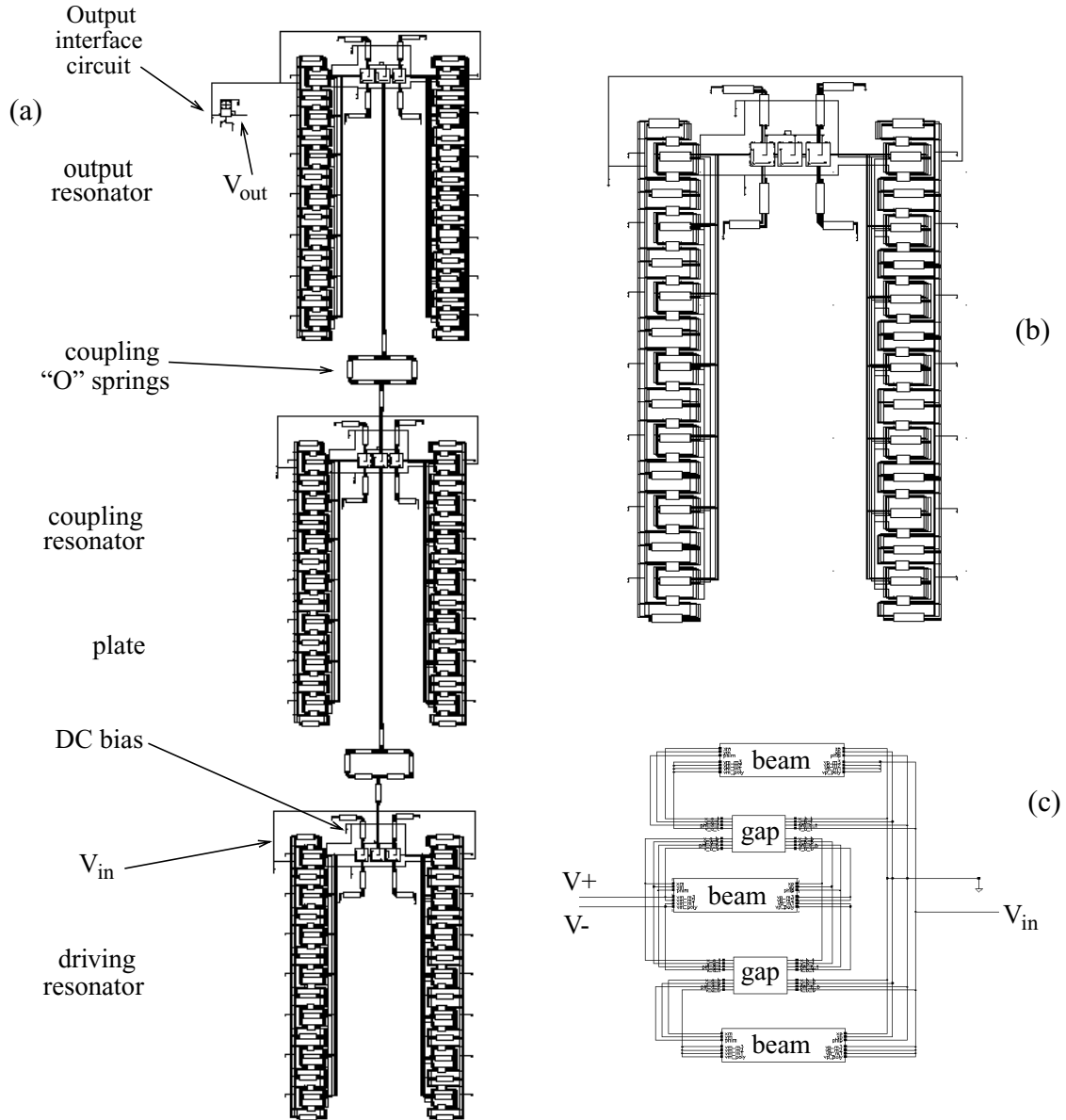
Figure 5.5: Schematic of bandpass filter in NODAS, with elements all at atomic-level. (a) schematic of the filter (b) schematic of the constitutional resonator (c) representation of the comb drive using beams and gaps (e.g. driving resonator).

gaps. As shown in Figure 5.5(c), each rotor finger is enclosed by two stator fingers, with two gap models in between to model the electrostatic fields on each side of the finger. The flat schematic looks cumbersome with a large number of gaps. Creating schematic cells for the comb drives and composing the system schematic hierarchically will lead to compact schematics. By using the atomic-level representation, the mechanical behavior of the comb fingers is captured by the beams and the electrostatic behavior of the comb fingers is captured by the gaps. A self-consistent solution to the electromechanical interaction will be solved by the simulator based on the system matrix.

The size of the system matrix gets bigger due to the introduction of a large number of nodes by the gap and beam models (the number of nodes increases from 574 to 947, the number of equations increases from 1953 to 3046), thus the simulation takes a bit longer (18 sec) than in the first case (5 sec). However, as the atomic-level models do not take rigid-body assumptions, they capture the physical behavior of the comb fingers with better accuracy. The difference becomes non-negligible when the comb fingers get softer.

5.1.5 Simulation and Verification

The mixed-level schematics (Figure 5.4) and the atomic-level schematics (Figure 5.5) are both simulated in NODAS and compared with the experimental data.

Figure 5.6 shows the SEM of the released CMOS-MEMS bandpass filter and the close-ups of the "O" coupling spring and the differential comb drive. The device is tested in air, with bias voltage at +/-20V.

Figure 5.7 shows the frequency response of the entire filter system (output voltage of interface circuit), with comparison to simulation. The comparison shows that the NODAS simulation with the functional-level comb drive macromodel and with the atomic-level gap and beam models both have very good agreement with the experimental data. The errors in the center frequency and bandwidth are within 3%. With gap and beam models, the frequency reduction due to the compliance of comb fingers is captured. Difference in the
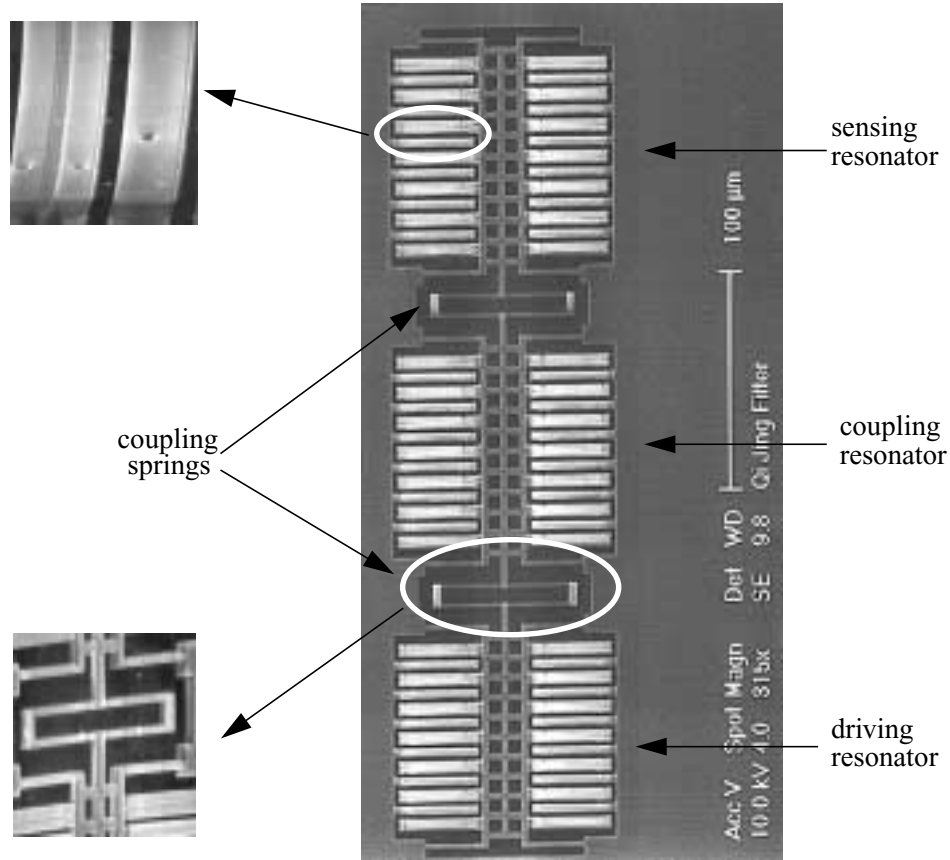
Figure 5.6: SEM picture of the CMOS-MEMS bandpass filter with close-ups of the "O"-shape coupling spring and the differential comb drives.

amplitude of resonant peaks are due to the difference in damping models. The squeeze film damping in the comb drive model is a more complicated and more accurate model compared to the simple squeeze film damping model in the gap [57][71], therefore, the simulation with comb drives gives a closer match to the experimental data in the resonant peak amplitudes.

5.1.6 Conclusions

Comparison of experimental data to simulation results validates the functionality and accuracy of the NODAS models for suspended MEMS design. The simulation with mixed-level models validates the interoperability of the NODAS models between different hierarchies.

The designed bandpass filter is oriented to application in the IF region. However,
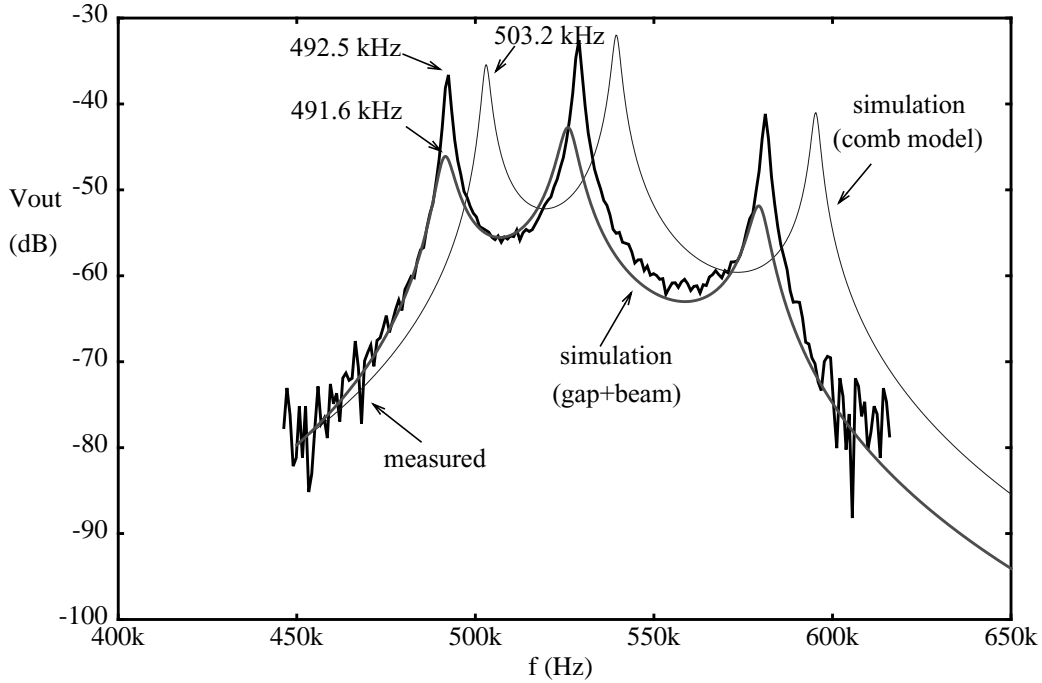
Figure 5.7: Measured output voltage of sensing interface circuit and output voltage from NODAS simulations (a. with functional-level comb drive macromodel and b. with atomic-level beam and gap models).

device performance such as bandwidth and passband ripple need to be improved for practical applications, which requires further exploration of effective Q-adjustment and frequency-tuning techniques.

5.2 RF Switch

5.2.1 Introduction

RF switches are used for switching RF or microwave signals. Compared to its electrical counterpart, RF switches fabricated in MEMS technology have advantages in electrical isolation and power consumption. As the electrodes of the MEMS RF switches are separated by air gaps when the switches are "off", there is no significant leakage or coupling between circuits at the two sides of the switches. Small power consumption is obtained due to the electrostatic actuation mechanism since dynamic currents flow through the switches only during the transition state.

5.2.2 Device Topology and Working Principle

RF switches can be implemented using thin beams or membranes in the topology of cantilever [46] or air bridge [5]. Figure 5.8 shows a typical RF switch composed by a fixed-fixed beam in air-bridge connection. The beam is actuated by the electrostatic attraction force generated by the pull-down electrodes underneath. When no pull-down voltage is applied, the beam stays flat. The big air gaps between the beam and pull-down electrodes electrically "shut off" the RF transmission line, leading to the "off" state of the switch. When a pull-down voltage is applied, the beam begins to bend down, moving closer to the RF transmission line. The air gap keeps decreasing and the electrostatic force keeps increasing during the actuation. When the voltage reaches a critical value (the pull-in threshold), the beam will snap to the underlying electrodes and stick to them, causing short circuits. To prevent the snap-in, the RF transmission line is designed to be slightly higher than the pull down electrodes, as shown in Figure 5.8. The beam electrode thus touches the RF transmission line before snapping to the pull down electrodes, leading to a safe "on" state of the switch. A thin dielectric layer between electrodes can also be used to prevent short circuit problems [5]. Switching time is the main figure of merit for RF switches. With a given amplitude of pull-down voltage, the switching speed is determined by the compliance of the beam and the air gap between the beam and the transmission line.
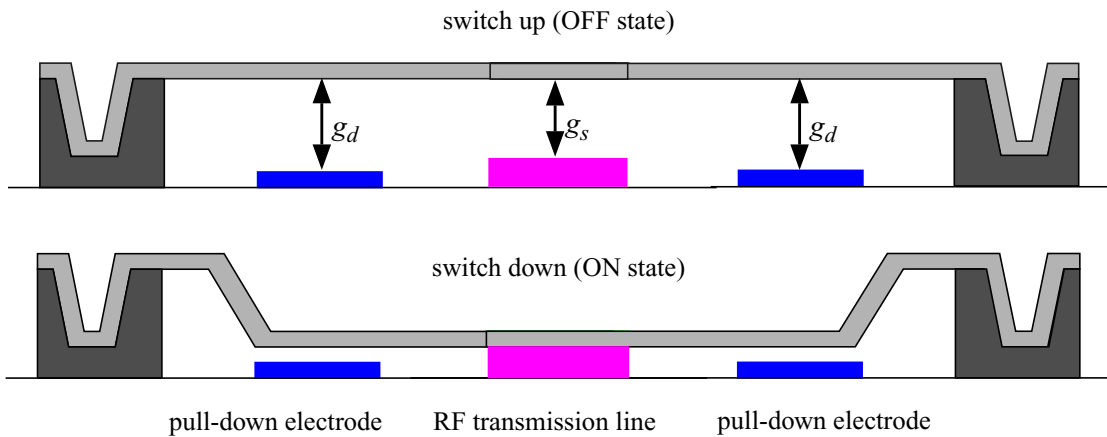
Figure 5.8: Typical structure of RF MEMS switch composed of fixed-fixed beam

5.2.3 Covered Physical Effects

Basic physical effects involved in this device include beam bending and electrostatic capacitive transduction. Simulation using atomic-level elements (beams and electrostatic gaps) will be performed in NODAS.

5.2.4 Schematic Composition

Figure 5.9 shows the NODAS schematic of the RF MEMS switch. As the fixed-fixed beam electrode at the top partially overlap with two pull-down electrodes and one RF transmission line, the beam must be discretized accordingly.

As shown in Figure 5.9(a), the fixed-fixed beam is split into four parts, each part represented by a beam element. Beam 1 and 4 model the two beam segments above the pull-down electrodes. They form a gap of g_d with the anchored driving electrodes at the bottom left and bottom right, respectively. For each pair of electrodes, a gap element is employed to
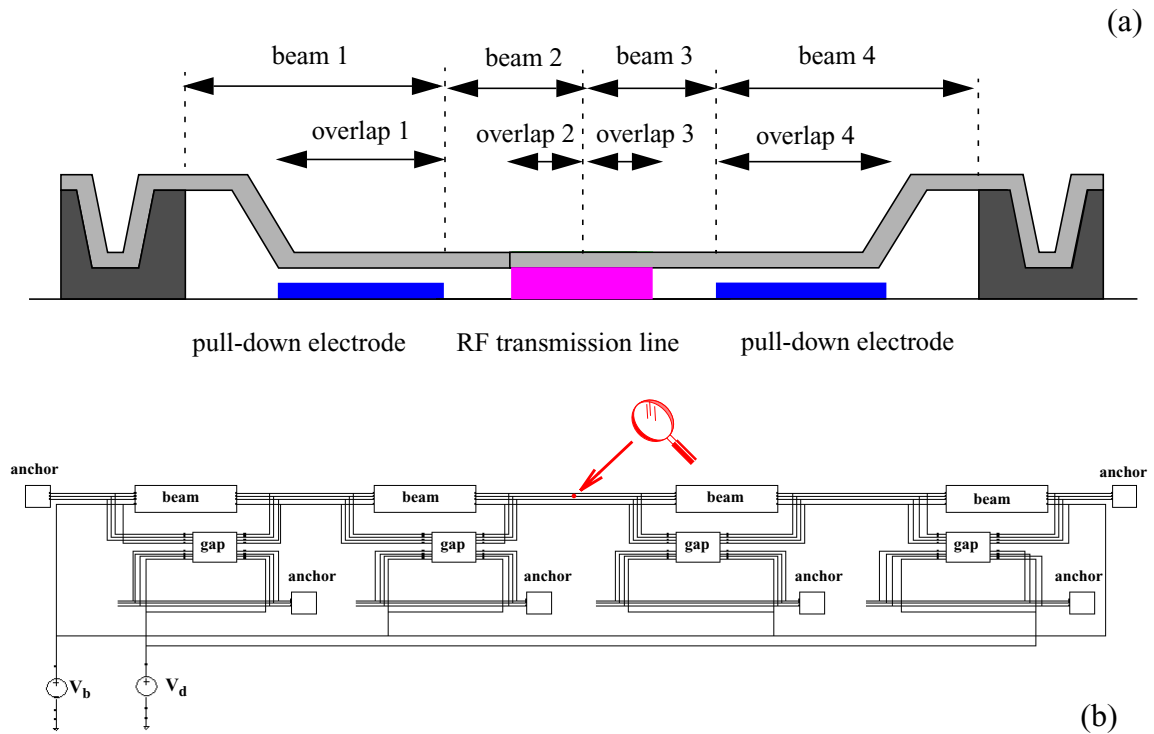
Figure 5.9: (a) Discretization of the fixed-fixed beam electrode of the RF MEMS switch (b) schematic in NODAS

capture the electrostatic transduction in between. The gap corresponding to beam 1 has a topology of 1 (top beam is to the left of bottom beam) and the gap corresponding to beam 4 has a topology of 0 (top beam is to the right of bottom beam).

The center beam segment above the RF transmission line is split into two parts too, modeled as beam 2 and beam 3. The split allows monitoring of the displacement at the center point of the fixed-fixed beam. In addition, as mentioned in the section about gap modeling in Chapter 3, case IV, which is the case for the gap between the fixed-fixed beam and the RF transmission line, is not covered by current gap model. Therefore, the split also allows the specific gap topology in this device to be correctly represented by current definition of the parameter *topology* in the gap model. As illustrated in Figure 5.9(a), the gap corresponding to beam 2 and the left half of the RF transmission line has a topology of 1 and the gap corresponding to beam 3 and the right half of the RF transmission line has a topology of 0. The initial gap for these two gap elements is g_s, which is designed to be smaller than g_d.

5.2.5 Simulation and Verification

Both static and transient analyses are performed for the RF switch. The fixed-fixed beam is electrically grounded. A driving voltage, v_d, is applied to the left and the right pull-down electrodes. We aim to simulate the driving process of the switch, therefore the RF transmission line is also electrically grounded so that there is no electrostatic transduction between the fixed-fixed beam and the RF transmission line, called "cold" switching [94]. As mentioned in the previous section, $g_s < g_d$, so the gap elements for beam 2 and 3 works as a mechanical stop for this simulation. The fixed-fixed beam bends down by the electrostatic force generated between the beam and the pull-down electrodes then stops and keeps in contact with the RF transmission line when the center of the beam touches the RF transmission line. Beam 1 and beam 4 have a length of 100 μm and a width of 2 μm. beam 2 and 3 have a length of 50 μm and a width of 2 μm. The overlap for the driving gaps is 80

μm and the overlap for the signal gaps is 25 μm. We know that the fringing effect is not included in the current NODAS gap model, but is included in the FEA/BEA simulation in CoventorWare. Therefore, in order to compare the NODAS simulation result to that from CoventorWare, the gaps should be set to small values and the thickness of the electrodes should be set to large values to make the fringing effect to be negligible. In the simulation given in Figure 5.10, g_d is 0.5 μm, g_s is 0.3 μm, and the thickness is 10 μm.

Figure 5.10(a) shows the static analysis results of the RF switch. The results from NODAS are compared to the results given by CoventorWare Analyzer simulation, showing very good agreement between the NODAS simulation and the self-consistent FEA/BEA simulation in Co-Solve [11]. The snap-in voltages match to within 1%.

Transient analysis is also performed in NODAS, while not supported by CoventorWare. A pulse driving voltage is applied, leading the RF switch through the transition from off-state to on-state, then back to the off-state. The entire process is shown in Figure 5.10(b). As the on and off times are the major specifications of RF switch design, the detail of the switch-on and switch-off transition processes are given in Figure 5.10(c) and (d) respectively and the switch times are measured. For a driving voltage with rise and fall steps of 1 μs, the switch-on time, t_{on}, is about 5.2 μs, and the switch-off time, t_{off}, is about 14 μs.

5.2.6 Conclusion

The verification simulation of the RF switch serves as another validation example of the functionality and simulation accuracy of the beam model and the electrostatic gap model. In addition, the NODAS schematic of this structure provides an example for the schematic composition using the gap elements.

5.3 Micromirror

5.3.1 Introduction

There are many designs of micro mirrors for applications in optical switching and beam

Figure 5.10: (a) Static analysis of the switch with comparison to CoventorWare simulation results (b) transient simulation in NODAS (c) switch-on process (d) switch-off process

steering [4][47][95]. Compared to projection display systems based on cathode-ray tube (CRT) and liquid crystal display (LCD), micromirror arrays are superior in cell uniformity, system brightness efficiency and contrast ratio.

5.3.2 Device Topology and Working Principle

Figure 5.11 shows a typical structure of micro mirrors [95][96][97]. The mirror is suspended by two torsional beams attached to support posts and electrically biased through the posts. It is actuated by electrostatic force generated between the mirror surface and the underneath electrodes and is constrained by the beam torsion torques. Voltages applied to the two underlying electrodes determine the direction of mirror rotation. After combining the mirror array with light source and projection optics, mirrors rotated in one direction steer the light into the lens, leading to the "on" state, and mirrors rotated in the opposite direction leads to the "off" state.

5.3.3 Covered Physical Effects

The basic physical effects involved in this device include beam torsion mechanics and torsional electrostatic actuation. Electrostatic gap elements will be used to simulate the capacitive actuation. The gap model is required to cover the torsional electrostatic effect to deal with the electrode angles induced by the mirror rotation.
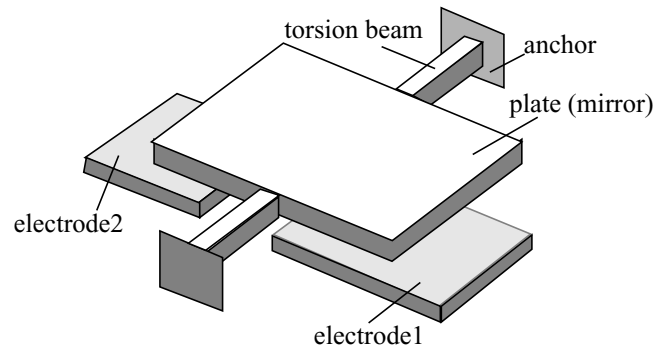
Figure 5.11: Structure of a micro mirror

5.3.4 Schematic Composition

Figure 5.12 shows the schematic of the micro mirror in NODAS. The torsion hinge is represented by two beams. The mirror surface is split into three parts, each modeled by a plate element. The plate element at the left represents the left part of the mirror, the plate element at the right represents the right part of the mirror and the center plate element represents the center part of the mirror connected to the hinges.

For each pair of electrodes (the mechanics of the driving electrodes are not included in the schematic, as both of their ends are anchored), a gap element is employed to model the electrostatic actuation. The gap at the left has a topology of 0 and the gap at the right has a topology of 1. As discussed in Chapter 4, the beam bending shape functions used for the derivation of gap model is fully compatible to the rigid-plate shape functions, therefore, the gap model based on bending beam electrodes is also suitable for the simulation of gaps formed by rigid plate electrodes. Notice that the splitter elements are used in the schematic,
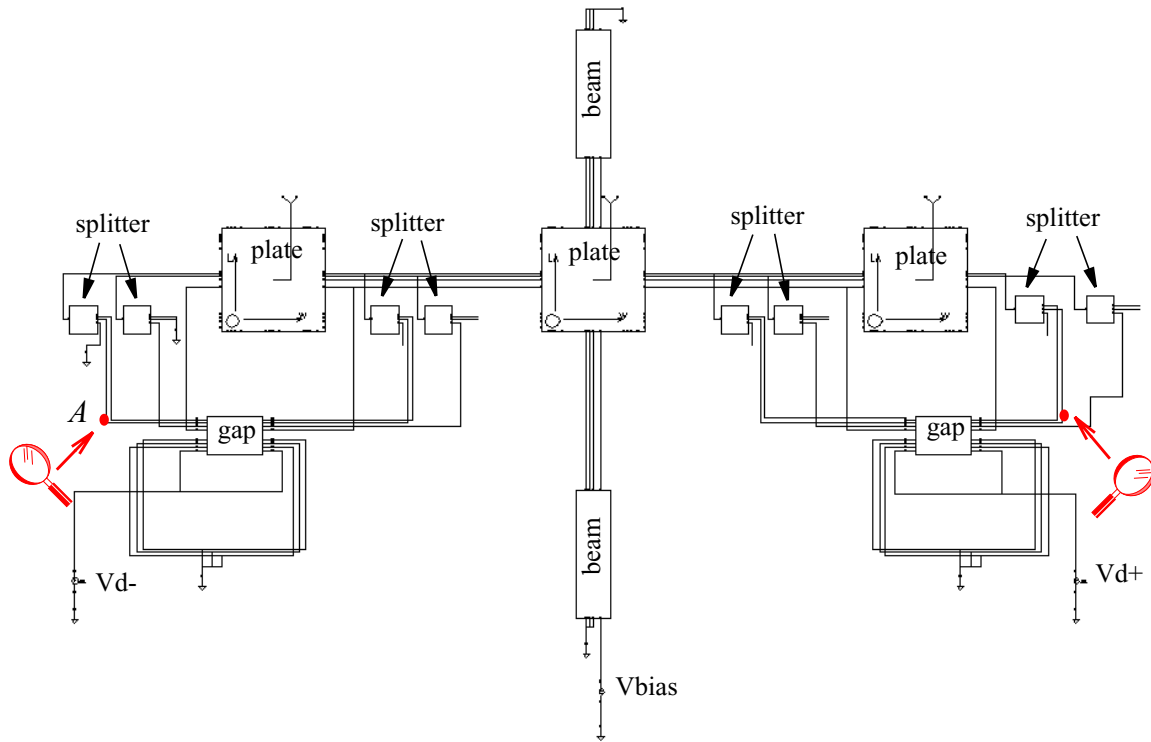
Figure 5.12: Schematic of the abstracted mirror in NODAS

because the torsion of the beams and the rotation of the plates are motions in space thus require 3D beam and plate elements while the current gap model is a 2D model. As mentioned in Chapter 2, bus pins are used for NODAS elements, leading to different bus dimensions between the 2D gap elements and the 3D beam elements. To solve the problem, splitters are employed to direct each bus element to its corresponding counterparts. In addition, the beams and the plates need to be rotated from their original positions. In Figure 5.11, the mirror moves in the *z*-direction and rotates about *y*-axis, however, the 2D gap model captures translations and rotations in-plane rather than out-of-plane. Therefore, the beams and plates are rotated in the schematic to translate out-of-plane motions into in-plane motions. According to the definition of Euler angles given in Chapter 3, the beam elements have $\alpha = 0$, $\beta = 90°$ and $\gamma = 90°$ and the plate elements have $\alpha = -90°$, $\beta = 0$ and $\gamma = 0$. In this way, the mirror moves along *y*-direction and rotates about *z*-axis so that they can work consistently with the 2D gap model to simulate the mirror. The displacements at the center of the far ends of the left and right mirror plates are the points where we monitor the displacements of the mirror.

5.3.5 Simulation and Verification

Static analysis for the abstracted mirror structure is performed in NODAS. A voltage, V_{d-}, is applied to the driving electrode at the left. The voltage of the driving electrode at the right (V_{d+}) and the voltage of the mirror plate (V_{bias}) are both grounded, so that the electrostatic field is concentrated between the electrode pair at the left. The results are compared to the data from CoventorWare Analyzer simulation. As mentioned in previous sections, the fringing effect is not included in current gap model but included by CoventorWare, therefore, the electrostatic gap is set to a small value of 0.5 μm to make the fringing effect negligible. The displacement at the far end of the left mirror plate, point *A*, as in Figure 5.12, is plotted. The comparison shows good agreement, as given in Figure 5.13(a). The static analysis verifies the accuracy of the beam model in capturing the beam
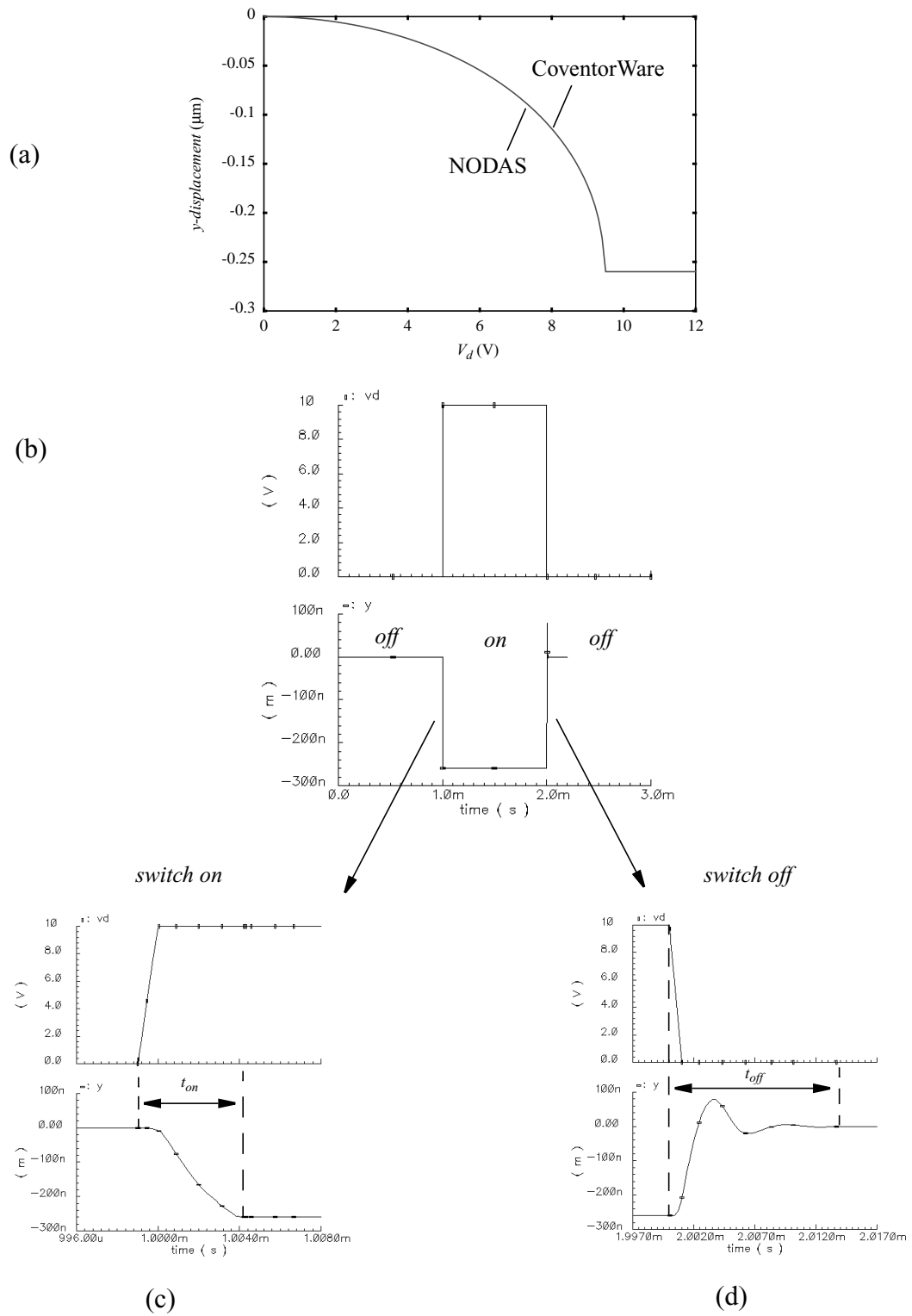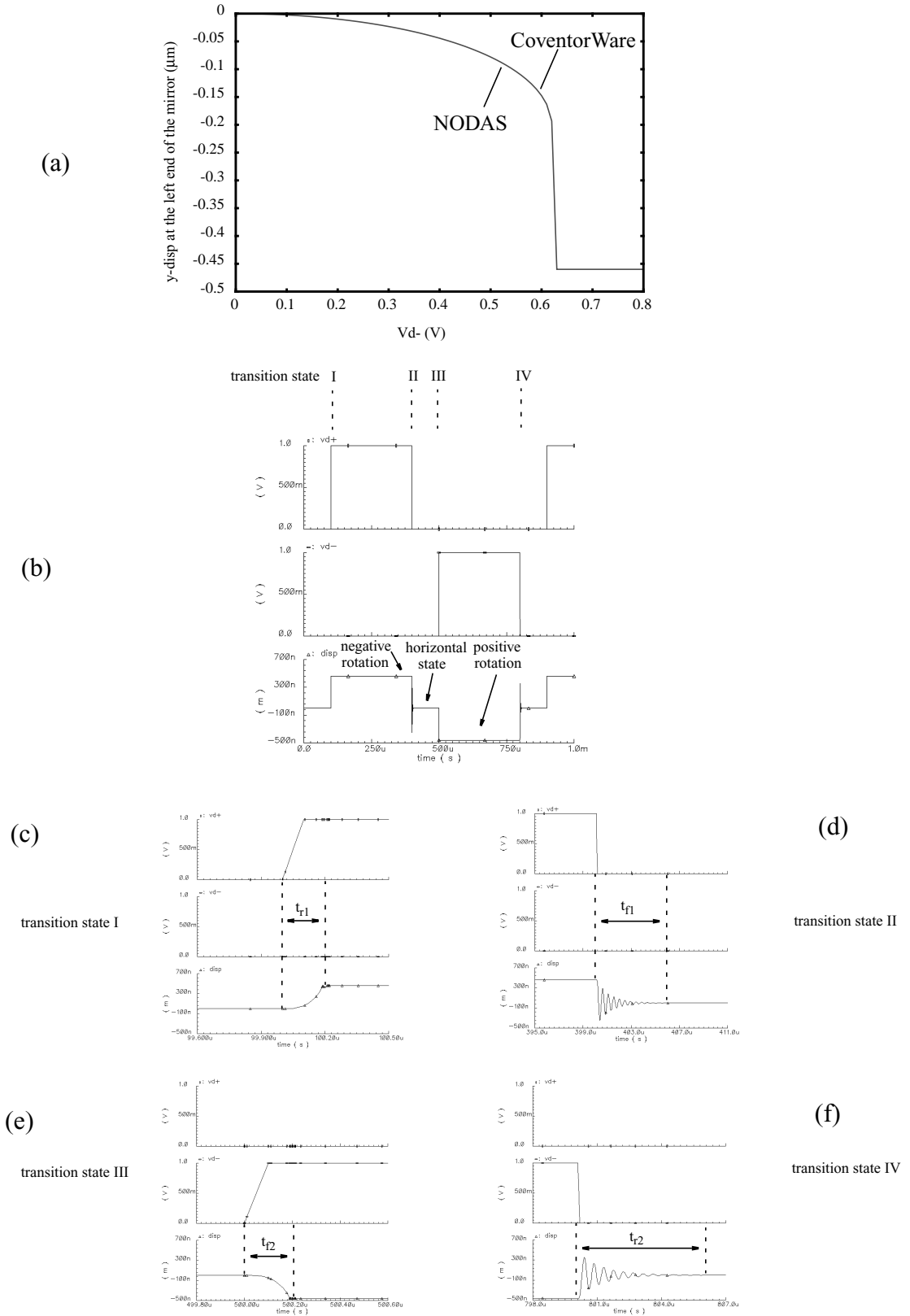
Figure 5.13: (a) Static analysis of the mirror (with comparison to CoventorWare simulation results) (b) Transient simulation in NODAS (c)-(f) Transition states

torsion and the accuracy of the gap model in capturing the electrostatic effect between plate electrodes with rotation angles. Notice that, as fringing effect thus the electrostatic force due to the fringing field is included in the CoventorWare simulation, the displacements given by CoventorWare are expected to be larger than those given by NODAS. However, the simulation results shows that they are actually smaller. This is due to the slightly higher structure stiffness given by the internal mechanical solver in CoventorWare, MechTool.

Figure 5.13(b) is the transient simulation in NODAS, which is not supported by self-consistent FEA/BEA. The simulation shows the positive rotation, the negative rotation and the horizontal state of the mirror. The displacement at the far end center of the left mirror plate, point *A*, is plotted. In this simulation, the mirror is electrically grounded. A positive V_{d+} is first applied to the driving electrode at the right side and V_{d-} is zero. Point A accordingly has a positive displacement in the *y*-direction and a negative rotation about the z-axis. A positive V_{d-} is then applied after V_{d+} is set back to zero. The mirror flips over along with the switch of the driving directions. When both driving voltages are zero, the mirror stays at the initial horizontal position. Figure 5.13(c)-(f) give the detail of the four transition states that the mirror goes through during the entire process: I - mirror rotates negatively around *z*-axis when V_{d+} is applied; II- mirror goes back to original position when V_{d+} goes back to zero; III - mirror rotates positively around *z*-axis when V_{d-} is applied; IV - mirror goes back to original position when V_{d-} goes back to zero. The rise and fall times of the driving voltages V_{d+} and V_{d-} are all set to 0.1 μs. As shown in the figures, transition I takes about 227 ns, transition II takes about 6.1 μs. Symmetrically, transition III takes about 227 ns and transition IV takes about 6.1 μs.

5.3.6 Conclusion

The static and transient analysis of the torsional micromirror validate the functionality and the simulation accuracy of the beam model in capturing the beam torsion mechanics and that of the gap model in capturing the electrostatic effects between rigid plate electrodes

with rotational displacements.

The mirror structure shown in Figure 5.11 is a simple structures which fulfills the fundamental function of torsional micro mirrors. Commercial produces, such the TI mirror arrays, have much more complicated design for the mirror surface, the driving electrodes and other mechanical parts such as the landing tip and hidden hinges [4]. The current NODAS model library only support rectangular plates and gaps formed by rigid rectangular plates or beams with rectangular/trapezoidal cross section, which limits the 100% accurate representation of such devices. However, as discussed in Chapter 5, as the NODAS models and modeling methodology is extensible to new design styles, the existing models are expected to be extensible to include these additional features.

5.4 Large-Stroke Actuator

5.4.1 Introduction

From the discussion about the gap model in Chapter 3, we know that for suspended MEMS devices under electrostatic actuation, the electrostatic force is usually very small as the allowed driving voltages are limited. In addition, snap-in happens when the electrode displacements are large enough to reach the pull-in threshold. Therefore, the displacement range achieved by electrostatic actuation is usually very limited.

In this section, a large-stroke actuator designed by Brennen [48], is reviewed and simulated as another validation case. The tangential drive employed in this specific design enables large amplitude tangential motion while keeping the driving voltage relatively low.

5.4.2 Device Topology and Working Principle

Figure 5.14 shows the structure of the large-stroke actuator. The force-generating part of the device consists two parallel bars separated by a small gap. One bar is fixed, and the other is attached to a parallelogram flexure suspension. when the suspension beams are not
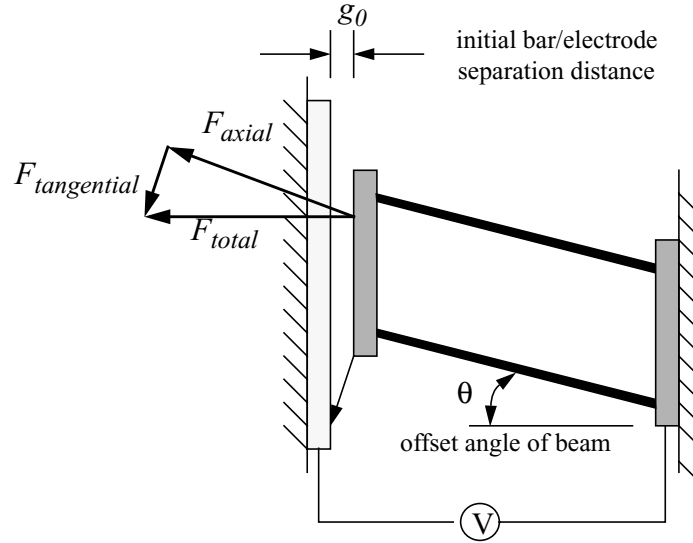
Figure 5.14: Structure of the tangential large-stroke actuator

parallel to the normal direction of the bars, the electrostatic force generated can be decomposed into two force components. One is parallel to the suspension beams, causing beam tension; the other is perpendicular to the beams, causing deflection of beams and lateral motion of the free bar. This structure, called tangential drive, converts electrostatic force between the fixed bar and the free bar into large amplitude tangential motion of the free bar.

5.4.3 Covered Physical Effects

This design is a combination of many physical effects due to its unique motion. It has a non-Manhattan topology. The beams not only bend but also have axial tension. The swing of the movable part involves both vertical and lateral electrostatic capacitive actuations. Thus, this device is a good comprehensive case to verify the capability of the NODAS cell library in dealing with non-Manhattan topology, composite beam mechanics, and mixed-mode capacitive actuation.

5.4.4 Schematic Composition

Figure 5.15 shows the NODAS schematic for the tangential large-stroke actuator.
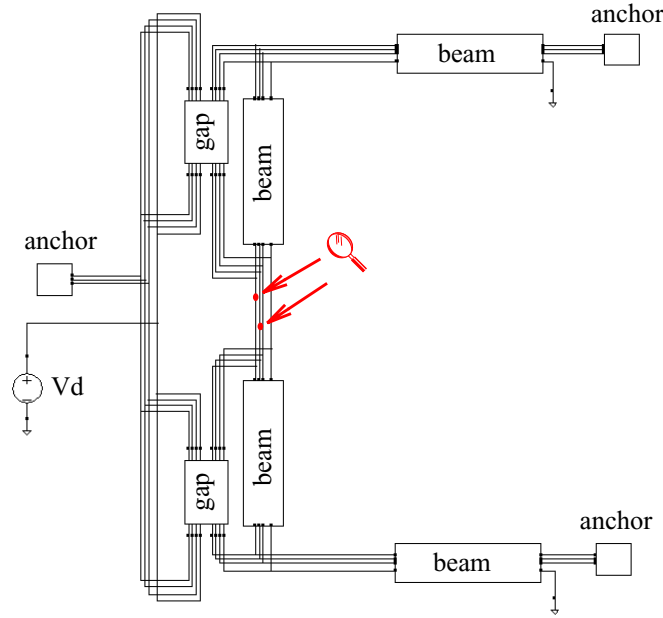
Figure 5.15: Schematic of the tangential large-stroke actuator

Notice that in this specific design, the moving bar is shorter than the fixed bar and is fully overlapped with the fixed bar. If only one gap element is used, neither topology 1 or topology 0 is suitable to be applied. This case is similar to the overlapping case IV which is illustrated in Section 3.5.2.7. Thus, we split the fixed bar and the moving bar into halves respectively, as shown in Figure 5.15. Each bar segment is represented as a beam element. The upper fixed and moving beams form an electrostatic gap with $\gamma = 90°$ and topology = 0. The lower fixed and moving beams form an electrostatic gap with $\gamma = 90°$ and topology = 1.

5.4.5 Simulation and Verification

Static analysis is performed for the large-stroke actuator. The offset angle of the parallelogram flexure suspension beams is set to -5.2°, a typical value given by [48]. The gap is set to a small value of 0.5 μm for the simulation, in order to make the fringing effect negligible. Figure 5.16 (a) and (b) gives the x and y displacements of the moving bar vs. the applied voltage. The comparison shows good agreement between NODAS and CoventorWare simulations. We see that the displacement in y-direction is about 10 times
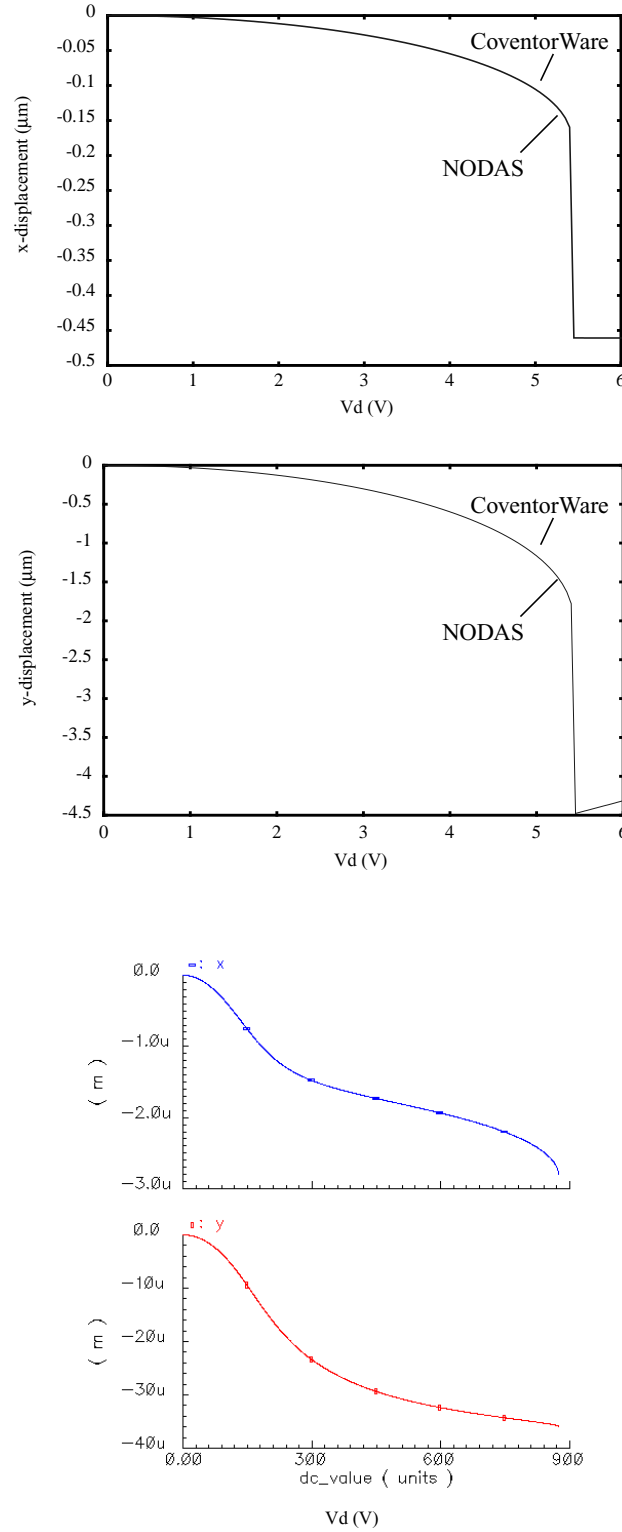
Figure 5.16: Static analysis of the tangential large-stroke actuator (a) *x*-displacement for the case with gap = 0.5 µm (b) *y*-displacement for the case with gap = 0.5 µm (c) *x* and *y* displacement for the case with gap = 4 µm

larger than the displacement in *x*-direction, which verifies that the tangential actuator can obtain large displacement. Although the maximum *y*-displacement given by this simulation is limited by the small gap, much larger displacements can be obtained by increasing the gap in real design [ref].

Figure 5.16 (c) gives the static simulation in NODAS of the large-stroke actuator with a much larger gap of 4 μm. The simulation is performed using nonlinear beam elements. It's shown that, unlike in the simulation with linear beam model, the electrodes don't snap in in the case when the actual gap approaches the snap-in threshold (about 1/3 of the initial gap). Instead, the actual gap becomes to decrease much slower after that. The nonlinear beam model is able to capture the large axial stress generated inside the beams and transform it into large tangential force to drive the flexible bar even further in *y*-direction. The large value of initial gap set for this simulation allows the flexible bar to have an extension in *x*-direction large enough to make large displacement in *y*-direction without touching the fixed electrode. The NODAS simulation results for this case are not accurate because fringing field is non-negligible in this case but is not covered in the gap model. However, this simulation verifies the design idea of the tangential large-stroke actuator, that is, using large axial stress in the beam to keep the structure from snapping in thus to obtain large tangential movement.

5.4.6 Conclusion

The simulation of the tangential large-stroke actuator validates the capability of NODAS in handling the non-Manhattan topology, the composite beam mechanics and the mixed lateral and vertical electrostatic effects. The model accuracy has been verified through the result comparison to FEA/BEA simulation in CoventorWare Analyzer.

The schematic composition of this device also validates the composition property of the suspended MEMS. Although the electrode topology of this device is not directly covered by the NODAS gap model, the device can still be simulated with satisfying

accuracy by decomposing the original device into several parts represented by the atomic-level NODAS elements.

5.5 Capacitive Pressure Sensor

5.5.1 Introduction

Capacitive pressure sensors employ pressure-sensitive capacitors to sense the variations of pressure. Compared to the piezoresistive counterpart, capacitive pressure sensors have superior characteristics in low temperature drift and high power efficiency [7].

5.5.2 Device Topology and Working Principle

Figure 5.17(a) shows a capacitive pressure sensor structure given in [7]. It's basically a variable capacitor with capacitance varying in response to the applied pressure. The membrane is the key element to the pressure sensor, implemented by the silicon etching
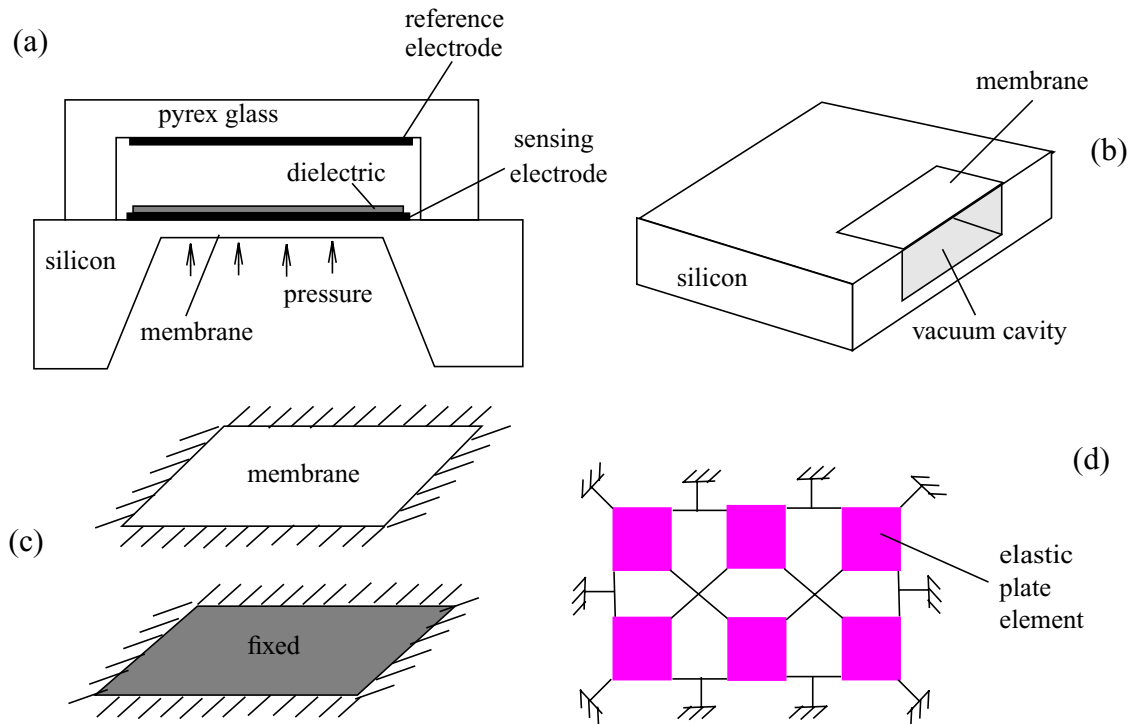
Figure 5.17: (a) A common capacitive pressure sensor (b) detail of the membrane structure (c) an abstraction of the device (d) a possible schematic-composition technique for the membrane

process. It deflects when subjected to pressure. The capacitor is formed by the sensing electrode mounted to the membrane, the fixed electrode attached to the pyrexglass, and the dielectric and air gap between them. When the membrane deflects, the sensing electrode moves toward the fixed electrode correspondingly, thus causing the capacitance variation.

Figure 5.17(b) shows the detail of the membrane structure. The membrane can be of various geometric shapes. We restrict it to be rectangular in this thesis work. Figure 5.17(c) is an abstraction of the real device. For the simplicity of simulation, we represent the pressure capacitive sensor as a rectangular membrane and a rectangular fixed sensing electrode.

The pressure sensor is a typical case where the elastic plate model should be employed to simulate the deflection of the membrane. As described in Section 3.4.2, the elastic plate model in NODAS only has connection terminals at corners and only accepts concentrated loads at these terminals. To deal with the pressure load, which is a distributed load, either new models need to be developed or an approximate schematic-composition method need to be explored. A possible solution is to represent the membrane as a network of elastic plate elements, as shown in Figure 5.17(d). Gap models are employed accordingly for the simulation of the nonlinear capacitance variance caused by the deflection of the membrane.

5.5.3 Covered Physical Effects

The physical effects covered by the capacitive pressure sensor include the elastic plate bending and the electrostatic capacitive sensing with bending electrodes.

As the current NODAS models only accept concentrated force and moment loads, the pressure sensor simulation also serves as a test case of the applicability of the NODAS models to simulations with distributed loads. In the simulation presented in the following section, the pressure load to the sensor is modeled as equivalent force loads evenly distributed along the membrane surface. The forces are applied to the connection terminals of the elastic plate elements in the plate network forming the membrane.

5.5.4 Schematic Composition

As mentioned in previous section, an important issue needed to be considered in the schematic composition is the extent of the discretization of the membrane, as we are simulating the continuous membrane under distributed pressure load using a finite number of elastic plate elements. Figure 5.18 shows several NODAS schematics for the membrane. The membrane is represented by networks with different numbers of elastic plate elements. Assume the membrane is L_0 long and L_0 wide. Figure 5.18(a) is a 2×2 network with totally 4 plates, each having a length of $L_0/2$ and a width of $L_0/2$; Figure 5.18(b) is a 4×4 network with totally 16 plates, each having a length of $L_0/4$ and a width of $L_0/4$; Figure 5.18(c) is a 8×8 network with totally 64 plates, each having a length of $L_0/8$ and a width of $L_0/8$; and Figure 5.18(d) is a 16×16 network with totally 256 plates, each having a length of $L_0/16$ and
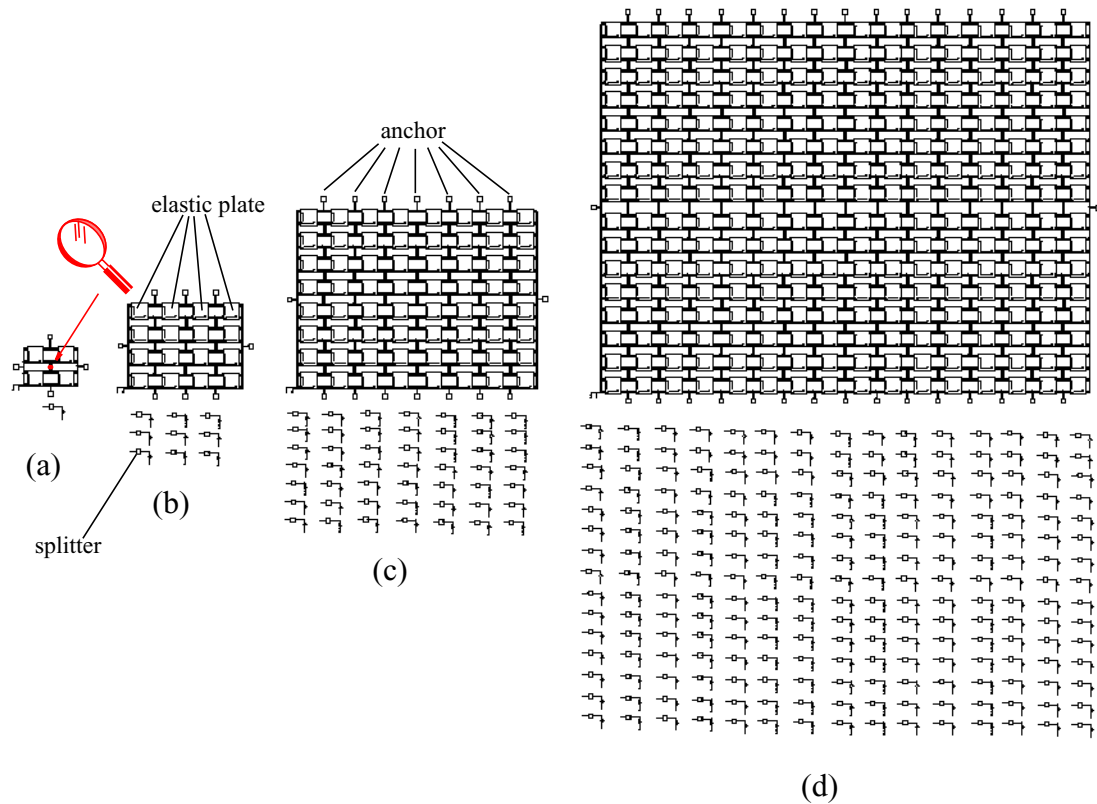
Figure 5.18: NODAS schematic of the membrane (a) a 2×2 network (b) a 4×4 network (c) a 8×8 network (d) a 16×16 network

a width of L_0/16. The four edges of the membrane are all anchored. The displacement at the center of the membrane, as shown Figure 5.18(a), is monitored. Equivalent forces are evenly applied to the "knots" of the networks. As shown in the figure, splitter elements are employed at each "knot" for the convenience of applying forces to the individual degrees of freedom.

The schematics given in Figure 5.18 are only the mechanical parts of the device. To simulate the entire function of the device, *i.e.*, the variance of the capacitance under the pressure load, networks of gap elements need to be created accordingly. When the numbers of elements get large, the effort required by manual schematic composition sharply increases. The schematic composition is expected to be eased by hierarchical schematic, which allows the designer to create a network with a small number of elements then use it repeatedly as the building block for larger networks, or by algorithmic schematic, which is expected to be created using Cadence SKILL code to allow parameterized automatic generation of schematic cell arrays, similar to the concept of Pcell for layout generation.

5.5.5 Simulation and Verification

Static analysis of the four mechanical networks given in Figure 5.18 are simulated in NODAS and compared to the data from CoventorWare. Figure 5.19(a) gives the displacement shape of the membrane. The displacements along the membrane surface are symmetric and reaches the maximum value at the center of the membrane. Figure 5.19(b) gives the percentage errors in the displacement at the center of the membrane for each network. The data shows that the errors decrease and converge along with the increase of the number of elements. When the total number of element increases to 256, the error reduces to 5.7%. However, as shown in Figure 5.20, the schematic with 256 plate elements greatly increases the total number of nodes in the system matrix hence sharply slows down the simulation, therefore is not a good solution of practical use.

(a) (b)
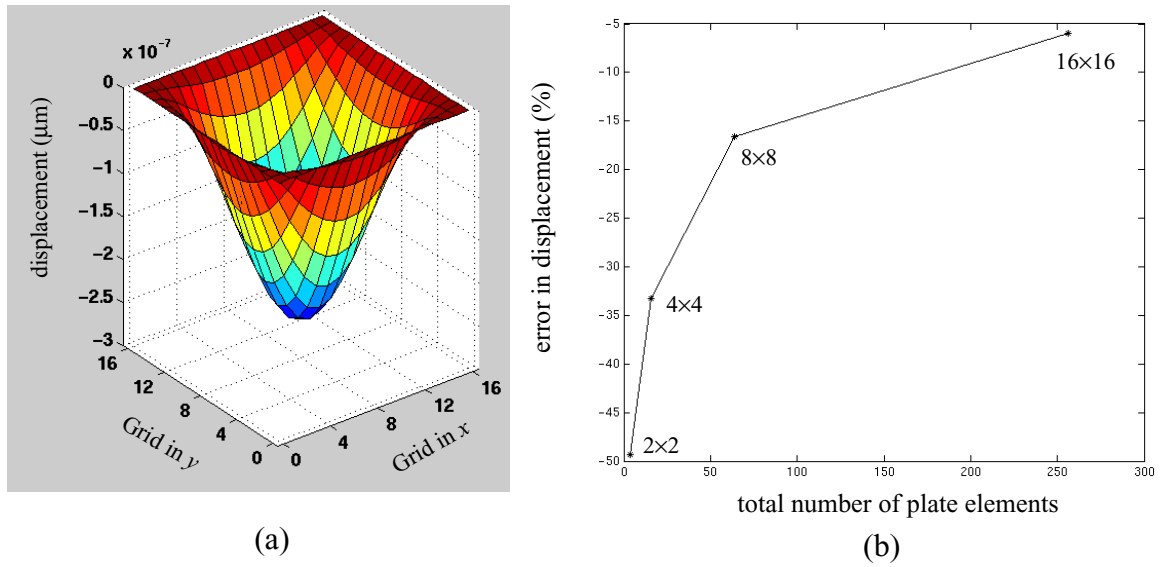
Figure 5.19: (a) displacement shape of the membrane (b) percentage errors in the displacement at the center of membrane (compared to data from CoventorWare simulation)
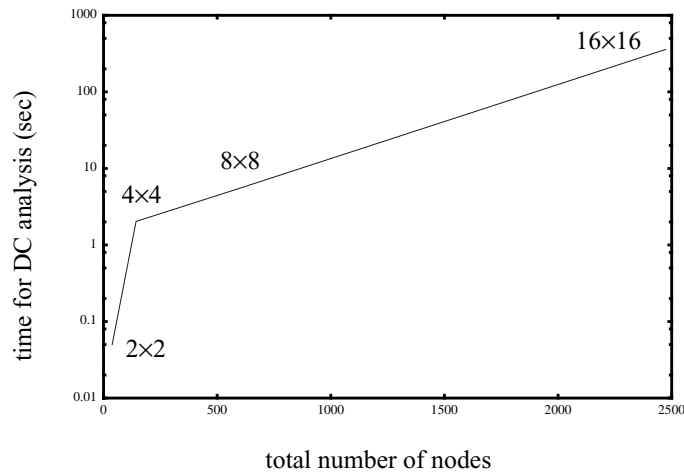
Figure 5.20: Effect of the total number of nodes in the plate network on the simulation time for the membrane

5.5.6 Conclusion

Different extent of discretization of the membrane is implemented and tested in NODAS. The result shows that using a very large number of elements may lead to acceptable simulation accuracy. However, the large number of nodes induced by large networks increases the simulation time thus implies that using the models based on

concentrated loads to approximate devices under distributed loads is not an efficient and practical methodology.

The simulation accuracies are expected to improve if the distributed pressure load could be lumped onto the connection terminals of elastic plate elements as equivalent forces and moments based on the elastic plate shape functions, similar to what is done in the electrostatic gap model and the beam model. However, in order to obtain symbolic solution to the lumped forces and moments, the exact function of the distributed load must be known and fixed. Unlike the electrostatic forces in the gap model or the inertial forces in the beam model, both of which are internal forces embedded inside the model, the distributed load to a pressure sensor is an unknown, external load to the model, except the model is aimed to deal with a specific type of load only. The feasibility of creating a general model good for arbitrary distributed load functions is limited.

Chapter 6   Summary and Future Work

This thesis is focused on NODAS, a modeling and simulation methodology for suspended MEMS. Several key issues are covered, including schematic representation, behavioral modeling, validation of composibility of suspended MEMS and extensibility of the modeling and simulation methodology to new physical effects, design styles, processes and physical domains.

6.1 Summary of NODAS Characteristics

NODAS is based on the composibility of suspended MEMS. It uses a small set of atomic elements to compose a large variety of suspended MEMS devices. The atomic-level elements include anchors, beams, plates and electrostatic gaps. Lumped models that describe the behavior of these atomic-level elements are implemented in analog HDLs. The models are parameterized by design geometry as well as process and material property parameters. The current version of NODAS is implemented in the design framework of Cadence, with models written in Verilog-A for simulation in Spectre. The major characteristics of NODAS have been discussed in detail in previous chapters, with comparison to other existing modeling and simulation methodologies. To sum up, as a CAD tool for the design of suspended MEMS, NODAS has the following characteristics:

- **Good accuracy**. In Chapter 3, the simulation accuracies of the atomic-level models are verified, either through comparison to analytical solutions or through comparison to FEA and/or BEA simulations. In Chapter 5, simulations of a set of validation cases are presented with comparison to experimental data or FEA/BEA simulations, further showing that NODAS gives simulation accuracy as good as FEA and/or BEA simulations.

- **Fast speed**. As NODAS models are lumped behavioral models, the system matrices are much smaller than those formed by FEA/BEA. NODAS hence offers faster simulation

speed, especially for transient analyses.

- **Ease of iterative evaluation**. Design iterations normally involve modifications of the element sizes and the device topologies. The geometric parameterization of NODAS elements allows the sizes of each element instance to be edited individually and easily, without the need to regenerate the layout and the mesh. Modeling and design are decoupled in NODAS as the models are independent with specific devices. Hence, in NODAS, the changes in device topology only require changes in element instances and/or wiring connections, without the need to recreate any models. In addition, NODAS uses schematics to represent the system. The editing of schematics is much easier and error-proof than the editing of the netlists. All the above features of NODAS greatly ease the iterative evaluations of suspended MEMS.

- **Ability to handle hierarchical design for large systems**. The implementation of NODAS in aHDLs allows the devices and systems to be represented at different levels (system level, device level or atomic level), as needed for the design. Elements at different levels can interact with each other freely using across and through variables shared at the common connection terminals. Hierarchical design for suspended MEMS is hence supported to enable the handling of large system complexity.

- **Ability to handle multi-physics analysis**. As discussed in Chapter 2, various physical disciplines are allowed by aHDLs to be represented in a uniform format: each physical discipline has its own sets of across and through variables; the physical equations are implemented as equations specifying relations between the across and through variables; a system matrix which contains the information from all of the involved physical disciplines is then formulated and solved by the simulator for a self-consistent solution. Therefore, NODAS is not only able to handle the simulation for individual physical disciplines, but also able to handle the interactions between multiple physical disciplines.

- **Ability to handle co-simulation with transistor-level electronics**. Simulators like Spectre allow co-simulation of MEMS devices based on Verilog-A models with transistor-

level electronics based on advanced SPICE models. This is a big advantage compared to MEMS CAD tools which do not support simulation of transistors or only support co-simulation with very simple transistor models. The inclusion of SPICE models into the co-simulation enables better simulation accuracy of the electronics as well as a better capture of the interactions between MEMS devices and electronics.

• **Coverage of a large variety of physical effects**. The model descriptions and verification simulations presented in Chapter 3, as well as the analyses and simulations of the validation cases presented in Chapter 5, shows that the small set of atomic-level elements is able to cover a large variety of suspended MEMS.

These atomic elements are chosen to be the building blocks for suspended MEMS as they capture and focus on different major types of physics. Compared to more generalized elements, such as the brick elements in finite element methods which can cover any 3D mechanics problem, these atomic-level models enable a better concentration on the dominating physics in specific designs. For example, if brick elements are used for the simulation of a slender structure, which has large aspect ratios, a very fine mesh will be required as the brick elements are accurate for small aspect ratios only. Instead, if using beam elements, the dominating beam elastic mechanics will be able to be captured with a much less number of elements as the beam model is derived based on slender beams and has the beam bending shapes embedded in. Similarly, if the proof mass in a design is known to be very stiff, then using the rigid plate element to model the proof mass will make the simulation much faster than using the general brick elements, while still being capable to cover the dominating physics. Therefore, although generalized elements provide larger coverage of physical effects and are hence indispensable for cases which can not be covered by the atomic-level elements such as beams and plates, the atomic elements are still included in finite element model libraries [9] and are chosen as the components for NODAS model library.

Despite the large coverage of the current NODAS model library, there are still cases

which are not yet covered, such as the hinge structures [98] and rotary motors with hubs [99]. According to the discussion about extensibility issues in Chapter 4, these uncovered cases are expected to be added as needed. For example, the 3D hinges could be modeled by adding a mechanical gap and contact model to capture the assembly process and adding a hinge model to capture the post-assembly fixed axle distance and relative rotation between the hinge and the gear [100]; the rotary motors could be modeled by adding an electrostatic stepper model to capture the actuation mechanism and adding a friction model to capture the sliding friction between the gear and the hub [101]. However, these extensions are expected to require lots of effort, as they are very different from the existing models hence there isn't much that can be inherited from the old models.

In summary, the current implementation of NODAS satisfies the targets outlined for MEMS CAD tools given in Chapter 1.

6.2 Thesis Contribution and Future Work

In this thesis work, the beam model has been extended from 2D to 3D, now covering the axial, lateral and torsional motions in space. Euler-angles are introduced for the specification of beam orientations and 3D rotation matrices are introduced for the coordinate transformations between the frames of reference. The linear beam model has been extended to the nonlinear beam model, which captures the geometric nonlinearity caused by large axial stress and large geometric deflection. The improvement greatly enlarges the application range of the beam element. Moreover, shear effects have been added to both linear and nonlinear beam models and the models have been extended to cover trapezoidal cross sections. The single-layer beam models have been extended to CMOS-MEMS beams with composite layer structures. Submodules are created for the core of mechanical and electrical models, respectively, and used in combination with parent modules to handle the multiple types of CMOS beams and to ease the maintenance of the model library. Future work on the beam element may consider expanding the straight beam

model to cover beams with curved shape, which is a design style used in some designs [78]. There are efforts on the modeling of beams with sine or cosine shapes [102], which split the curved beam into multiple straight beam elements scattered along the curved beam, then extract higher-level stiffness and mass matrices for the entire beam. This method is good for beams with well-known geometric shapes, but not of general use. However, similar efforts can be taken to expand the NODAS beam model to other specific curved shapes when needed.

For the rigid plate model, parameter *joint-offset* has been added to the connection terminals at the plate corners, allowing flexible connections to points which are offset from the corners. The plate model has been extended to 3D and CMOS-MEMS as well. An elastic plate model has been added to the NODAS model library. It covers the in-plane stretching and the out-of-plane bending of the plate. As discussed in the simulation for membranes under pressure loads in Chapter 5, the current elastic plate model, in which the loads are restricted to concentrated loads at connection terminals, can handle the distributed loads only when a large number of elements are used. However, the large amount of elements introduces large number of nodes to the system matrices and thus limiting the speed. It also degrades the circuit-level simulation to finite element analysis with fine meshes, hence is not a good solution of practical use. It may be a better option to create a specific elastic plate model compatible with distributed loads. As the model depends on the specific functions of the distributed loads, the model is expected to be a specific model for certain types of loads other than a general one.

The electrostatic gap model currently captures the 2D electrostatic effects between two beam electrodes. It uses two rotated rigid plates obtained from the beam bending shape functions to approximate the parts of bending beam electrodes in the overlapping region and then uses the rotated parallel-plate approximation to model the electrostatic field between the rotated rigid plates. The gap model includes a contact model which captures the physical touching between electrodes. Future work for the gap model includes improvement

on the composibility, extension to 3D, extension to large deflection, addition of the fringing effects and extension to CMOS-MEMS gaps. A dynamic definition of the gap topology based on both the initial layout positions and the electrode displacements is expected to make the gap composable. The extension to 3D requires definition and integration of the irregular electrostatic field in space. The extension to large electrode deflection requires capture of the beam displacement functions under large deflection and capture of the electrostatic field formed by two beam electrodes with large rotation angles. The addition of fringing fields requires capture of both the fringing field in the overlapping region [103] as well as the fringing field in the non-overlapping region [81], with the option to include ground plane or not. Extension of the gap model to CMOS-MEMS requires consideration of both the electrostatic fields formed by conductor layers facing each other and the interaction with the electrostatic fields formed by adjacent layers. As no analytical model is available for this case, a large amount of numerical simulations and corresponding curve-fittings are necessary for the model development [71].

References

[1] J. E. Vandemeer, M. S. Kranz, and G. K. Fedder, "Nodal Simulation of Suspended MEMS With Multiple Degrees of Freedom", in *Proceedings of ASME Winter Annual Conference (ASME WAM '97)*, Nov. 16-21, 1997, Dallas, TX, USA, pp. 113-118.

[2] W.C. Tang, T.-C.H. Nguyen, R.T. Howe, "Laterally Driven Polysilicon Resonant Microstructures", *Micro Electro Mechanical Systems, 1989, Proceedings*, 'An Investigation of Micro Structures, Sensors, Actuators, Machines and Robots', IEEE, Feb. 20-22, 1989, pp. 53 -59.

[3] K.H.-L. Chau, S.R. Lewis, Y. Zhao, R.T. Howe , *et.al.*, "An integrated force-balanced capacitive accelerometer for low-g applications", *Sensors and Actuators A* (Physical), A54, no. 1-3, June 1996, pp. 472-476.

[4] P. F. Van Kessel, L.J. Hornbeck, R.E. Meier, and M.R. Douglass, "A MEMS-Based Projection Display", IEEE proceedings, *Special Issue on Integrated Sensors, Microactuators and Microsystems*, Aug, 1998, pp. 1687-1704.

[5] C. Goldsmith, J. Randall, S. Eshelman, T.H. Lin, D. Denniston, S. Chen, B. Norvell, "Characteristics of Micromachined Switches at Microwave Frequencies", *IEEE MTT-S International Microwave Symposium Digest* 2, Jun 17-21 1996, pp. 1141-1144.

[6] K.Wang and C.T.-C. Nguyen, "High-Order Micromechanical Electronic Filters," *IEEE MEMS Workshop*, Japan, January 26-30, 1997, pp.25-30.

[7] H. Kim, Y. Jeong, K. Chun, "Improvement of the Linearity of a Capacitive Pressure Sensor Using an Interdigitated Electrode Structure", *Sensors and Actuators, A: Physical* v 62 n 1-3, July 1997, pp. 586-590.

[8] V. I. Zhulin, S. J. Owen, D. F. Ostergaard, "Finite Element Based Electrostatic-Structural Coupled Analysis with Automated Mesh Morphing", *Proceedings of the International Conference on Modeling and Simulation of Microsystems MSM 2000*, San Diego, CA, March 27-29, 2000.

[9] ABAQUS/Standard User's Manual, Hibbitt, Karlson and Sorenson Inc., Pawtucket RI.

[10] Maxwell User Manuals, Ansoft Corporation, Four Station Square, Pittsburgh, PA.

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

[12] A. Przekwas, "An Integrated Multi-Disciplinary CAD/CAE Environment for MEMS," *Proceedings of Design*

Test Integration and Packaging of MEMS/MOEMS, DTIP '99, Paris, France, March 30, 1999.

[13] F. Maseeh, "IntelliCAD: the CAD for MEMS", in *WESCON/95 Conference Record*, New York, NY, USA, 1995, pp. 320-324.

[14] S. Crary and Y. Zhang, "CAEMEMS: an Integrated Computer Aided Engineering Workbench for Micro-Electromechanical Systems", IEEE International Workshop on Microelectromechanical Systems, MEMS '90, Napa Valley, CA, 1990, pp. 113-114.

[15] J. G. Korvink, *et.al.*, "SESES: A Comprehensive MEMS Modeling System", *IEEE International Workshop on Microelectromechanical Systems, MEMS '94*, Oiso, Japan, 1994, pp. 22-27.

[16] J.M. Funk, J.G. Korvink, *et.al.*, "SOLIDIS: a Tool for Microactuator Simulation in 3-D", *Journal of Microelectromechanical Systems (IEEE)*6, no. 1, March 1997, pp. 70-82.

[17] http://www.ansys.com/ansys/mems/mems_key_features/key_feature_essolve.htm

[18] F. Wang, J. White, *et.al.*, "Automatic Model Order Reduction of a Microdevice Using the Arnoldi Approach", *Micro-Electro-Mechanical Systems (MEMS) - 1998. ASME International Mechanical Engineering Congress and Exposition*, New York, NY, USA, pp. 527-530.

[19] L.D. Gabbay, S.D. Senturia, "Automatic Generation of Dynamic Macro-Models Using Quasi-Static Simulations in Combination with Modal Analysis", in *Technical Digest. Solid-State Sensor and Actuator Workshop, 1998*, Cleveland, OH, USA, pp. 197-200.

[20] N. R. Swart, S. F. Bart, M. H. Zaman, *et.al.*, " AutoMM: Automatic Generation of Dynamic Macromodels for MEMS Devices", *MEMS '98*, Jan., 1998, pp. 178-183.

[21] Y. Chen, J. White, "A Quadratic Method for Nonlinear Model Order Reduction", *Proceedings of the International Conference on Modeling and Simulation of Microsystems, MSM'2000*, San Diego, CA, March 27-29, 2000.

[22] D.F. Ostergaard and M. Gyimesi, "Finite Element Based Reduced Order Modeling of Micro Electro Mechanical Systems (MEMS)", *the 2000 International Conference on Modeling and Simulation of Microsystems (MSM 2000)*, San Diego, CA, March 27-29, 2000, pp. 684-687.

[23] M. Varghese, S.D. Senturia, J.R. Gilbert, *et.al.*, "Automatic Reduced-Order Modeling in MEMCAD Using Modal Basis Functions", *11th International Conference on Solid-State Sensors and Actuators*, Berlin, Germany,

2001, Vol. 1, pp. 264-267.

[24] M.A. Maher and H.J. Lee, "MEMS Systems Design and Verification Tools," in *Proc. of the SPIE Smart Structures and Materials Conference*, San Diego, CA, March 1998, pp.40-48.

[25] M.A. Maher, "Design Flow from 3D Model into Layout", *CAD White Papers*, http://www.memscap.com/ products-c-whitepapers.html.

[26] 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.

[27] 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.

[28] T.P. Kurzweg, S.P. Levitan, *et.al.*, "A CAD Tool for Optical MEMS", in *Proceedings 1999 Design Automation Conference*, *DAC'99*, Piscataway, NJ, USA, pp. 879-884.

[29] D. Moulinier, *et.al.*, "MEMSMaster: A New Approach to Prototype MEMS", *DTIP'01*, April, 2001, pp.165-174.

[30] S. Iyer, H. Lakdawala, T. Mukherjee, G.K. Fedder, "Modeling Methodology for a CMOS-MEMS Electrostatic Comb", *Proceedings of SPIE - The International Society for Optical Engineering*, v 4655, 2002, pp. 114-125.

[31] ANSYS Tutorial on MEMS, *the 2002 International Conference on Modeling and Simulation of Microsystems (MSM 2002)*, San Juan, PR, April 22-25, 2002.

[32] http://www.synopsys.com/products/mixedsignal/hspice_ds.html

[33] K.S. Kundert, *The Designers Guide to SPICE and Spectre*, Kluwer Academic Publishers, Boston, 1995.

[34] http://www.cadence.com/datasheets/spectre_cir_sim.html

[35] G.K. Fedder, and Q. Jing, "A Hierarchical Circuit-Level Design Methodology Microelectromechanical Systems", in *IEEE Transactions on Circuits & Systems II (TCAS)*, Vol. 46, Issue 10, Oct. 1999, pp. 1309-1315.

[36] J. E. Vandemeer, M. S. Kranz, and G. K. Fedder, "Hierarchical Representation and Simulation of Micromachined Inertial Sensors", in *Technical Proceedings of the 1998 International Conference on Modeling and Simulation of Microsystems*, *Semiconductors, Sensors and Actuators (MSM' 98)*, Apr 6-8, 1998, Santa Clara, CA, pp. 540-545.

[37] Q. Jing, T. Mukherjee, and G. K. Fedder, "A Design Methodology for Micromechanical Bandpass Filters", in

Proceedings of 1999 IEEE/ACM/VIUF Intl. Workshop on Behavioral Modeling and Simulation (BMAS '99), October 4-6, Orlando, FL, USA.

[38] S. V. Iyer, and T. Mukherjee, "Numerical Spring Models for Behavioral Simulation of MEMS Inertial Sensors", in *Design, Test, Integration and Packaging of MEMS/MOEMS,* May 9-11, 2000, Paris, France, pp. 55-62.

[39] http://www.eda.org/verilog-ams/

[40] http://www.synopsys.com/products/mixedsignal/nanosim_wp.html

[41] http://www.mentor.com/ams/adms.html

[42] J.M. Gere and S.P. Timoshenko, *Mechanics of Materials*, 4th Ed., PWS Pub Co., Boston, MA, 1997.

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

[44] T.Y. Yang, *Finite Element Structural Analysis*, Prentice Hall College Div, Englewood Cliffs, NJ, 1986.

[45] Q. Jing, H. Luo, T. Mukherjee, L. R. Carley, and G. K. Fedder, "CMOS Micromechanical Bandpass Filter Design Using a Hierarchical MEMS Circuit Library", in *Proceedings of the 13th IEEE International Conference on Micro Electro Mechanical Systems (MEMS 2000)*, January 23-27, 2000, Miyazaki, Japan.

[46] E.R. Brown, "RF-MEMS Switches for Reconfigurable Integrated Circuits", *IEEE Transactions on Microwave Theory and Techniques*, Vol. 46, No. 11, Nov 1998, pp. 1868-1880.

[47] H. Toshiyoshi and H. Fujita, "An Electrostatically Operated Torsional Mirror for Optical Switching Device", *the 8th Inter. Conf. on Solid-State Sensors and Actuators*, Stockholm, Sweden, June 25-29, 1995, pp. 297-300.

[48] R.A. Brennen, M.G. Lim, A.P. Pisano and A.T. Chou, "Large Displacement Linear Actuator", *Solid-State Sensors and Actuators Workshop*, 1990, 4th Technical Digest, pp. 135-139.

[49] J.J.-Y. Gill, L.V. Ngo, P.R. Nelson, *et.al.*, "Elimination of Extra Spring Effect at the Step-up Anchor of Surface-Micromachined Structure", *Jour. of Microelectromechanical Systems*, Vol. 7, Issue 1, Mar 1998, pp 114-121.

[50] http://www.ece.cmu.edu/~mems/intranet/cadence/mumps_help/layout_generation.shtml

[51] J. E. Vandemeer, *Nodal Design of Actuators and Sensors (NODAS)*, M.S. Thesis, Carnegie Mellon Univ., 1998.

[52] J.M. Gere and S.P Timoshenko, *Mechanics of Materials*, 2nd Ed., Van Nostrand Reinhold Co., New York, 1972.

[53] L. Meirovitch, *Analytical Methods in Vibrations*, Macmillan Publishing Co., Inc., New York, NY, 1967.

[54] P. Mario, *Structural Dynamics, Theory and Computation*, Van Nostrand Reinhold, New York, 1980.

[55] X. Zhang and W.C. Tang, "Viscous Air Damping in Laterally Driven Microresonators", *Sensors and Materials*,

Vol.7, No. 6, 1995, pp. 415-430.

[56] J.B. Starr, "Squeeze Film Damping in Solid State Accelerometer", *Tech. Digest, IEEE Solid State Sensors and Actuators Workshop*, Hilton Head Island, SC, USA, June 1990, pp. 44-47.

[57] S. Vemuri, G.K. Fedder and T. Mukherjee, "Low-Order Squeeze Film Model for Simulation of MEMS Devices", in *Technical Proceedings of the Third International Conference on Modeling and Simulation of Microsystems (MSM 2000)*, pp. 205-208, March 27-29, 2000, San Diego, CA, USA.

[58] S. V. Iyer, T. Mukherjee, and G.K. Fedder, "Multi-mode Sensitive Layout Synthesis of Microresonator", in *Technical Proceedings of the 1998 International Conference on Modeling and Simulation of Microsystems (MSM 1998)*, pp. 392-397, April 6-8, 1998, Santa Clara, CA, USA.

[59] R.H., Parvin, *Inertial Navigation*, Contributing authors: William E. Shephard, W. Bruce Walker [and] David W. Geyer, Van Nostrand, Princeton, NJ, 1962.

[60] H., Goldstein, *Classical Mechanics*, 2nd ed., Reading, MA, Addison-Wesley, 1980.

[61] Resource library of Methematica, http://mathworld.wolfram.com/EulerAngles.html

[62] S.V. Iyer, Q. Jing, G.K. Fedder, and T. Mukherjee, "Convergence and Speed Issues in Analog HDL Model Formulation for MEMS", in *Technical Proceedings of the Fourth International Conference on Modeling and Simulation of Microsystems (MSM 2001)*, March 19-21, 2001, Hilton Head, SC, USA

[63] R. Frisch-Fay, *Flexible Bars*, Butterworths Scientific Publications, Washington, D.C., 1962.

[64] S.A. Canann, Y. C. Liu; A.V. Mobley, "Automatic 3D surface meshing to address today's industrial needs", *Finite Elements in Analysis and Design (Elsevier)* 25, no. 1-2, March 20, 1997, pp. 185-98.

[65] M. Bachtold, P. Ljung, "The Constrained Boundary Element Method, a Technique Allowing General Surface Meshes in MEMS Simulations", in *Solid-State Sensor and Actuator*, Cleveland, OH, USA, 1998, pp. 201-204.

[66] S.D. Senturia, *Microsystem Design*, Kluwer Academic Publishers, Boston, 2000.

[67] Personal Discussion with Prof. Gary K. Fedder.

[68] A. Glassner, *Graphics Gems*, Academic Press, Boston, 1990.

[69] ABAQUS/Standard User's Manual, V 5.6, Vol. II, Hibbitt, Karlson and Sorenson Inc., Pawtucket, RI, 1996.

[70] J. B. Marion and S.T. Thornton, *Classical Dynamics of Particles & Systems*, Harcourt Brace Jovanovich Inc.,

1988.

[71] S.V. Iyer, *Modeling and Simulation of Non-idealities in a Z-axis CMOS-MEMS Gyroscope*, PhD thesis, Carnegie Mellon University, 2003

[72] Eung-Sam Kim, Young-Ho Cho and Moon-Uhn Kim, "Effect of Holes and Edges on the Squeeze Film Damping of Perforated Micromechanical Structures", *Twelfth IEEE International Conference on Micro Electro Mechanical Systems, 1999 (MEMS '99)*, Jan 17-21, 1999, pp.296 -301.

[73] J.R. Gilbert, G.K. Ananthasuresh and S.D. Senturia, "3D Modeling of Contact Problems and Hysteresis in Coupled Electro-Mechanics", *MEMS' 96*, Feb.11-15, 1996, pp.127 -132.

[74] M. S.-C. Lu, and G. K. Fedder, "Parameterized Electrostatic Gap Models for Structured Design of Microelectromechanical Systems", in *Technical Proceedings of the 2nd International Conference on Modeling and Simulation of Microsystems (MSM 99)*, April 19-21, 1999, San Juan, PR, USA.

[75] G.K. Fedder, *Simulation of Microelectromechanical Systems*, PhD thesis, UCBerkeley, 1994.

[76] X. Cai, H. Yie, P. Osterberg, J. Gilbert, S. Senturia and J. White, "A Relaxation/Multiple Accelerated Scheme", Proc. ICCAD, IEEE, pp. 283-286, 1993.

[77] http://www.wolfram.com/products/mathematica/index.html

[78] F. Ayazi, K. Najafi, "Design and Fabrication of High-Performance Polysilicon Vibrating Ring Gyroscope", *MEMS'98*, Jan. 25-29, 1998, pp. 621 -626.

[79] H. Xie and G.K. Fedder, "A DRIE CMOS-MEMS Gyroscope", *Sensors, 2002. Proceedings of IEEE* , Vol.2 , 2002, pp. 1413 -1418.

[80] X. Zhu, D.W. Greve, G.K. Fedder, "Characterization of silicon isotropic etch by inductively coupled plasma etch in post-CMOS processing", *in Proceedings IEEE Thirteenth Annual International Conference on Micro ElectroMechanical Systems*, Piscataway, NJ, USA, 2000, pp. 568-573.

[81] G. Lorenz, *Network Simulation of Micromechanical Systems*, PhD Thesis, University of Bremen, 1999.

[82] E.J. Nestorides, *A Handbook on Torsional Vibration*, Cambridge [Eng.] University Press, 1958.

[83] S.P Timoshenko and J.N. Goodier, *Theory of Elasticity*, McGraw-Hill, New York, 3rd ed., 1970.

[84] S.V. Iyer, H. Lakdawala, G. K. Fedder and T. Mukherjee, "Macromodeling Temperature-Dependent Curl in CMOS Micromachined Beams", in *Technical Proceedings of the 4th International Conference on Modeling and*

Simulation of Microsystems (MSM 2001), March 19-21, 2001, Hilton Head, SC, USA.

[85] 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 Micro-structures in a Conventional CMOS Process", *Sensors and Actuators*, 1996, vol.A57, no. 2, pp. 103-110.

[86] M. Lu, X. Zhu, and G.K. Fedder , "Mechanical Property Measurement Of 0.5-mm CMOS Microstructures", in *Proceedings of Materials Research Society 1998 Spring Meeting*, April 13-17, 1998, San Francisco, CA, USA.

[87] H. Lakdawala, Temperature Control of CMOS Micromachined Sensors, PhD thesis, Carnegie Mellon Univ, 2002

[88] H. Xie, L. Erdmann, X. Zhu, K. J. Gabriel, and G. K. Fedder, "Post-CMOS Processing For High-aspect-ratio Integrated Silicon Microstructures", in *Technical Digest of the Solid-State Sensor and Actuator Workshop*, June 4-8, 2000, Hilton Head, SC, USA.

[89] A. Oz and G.K. Fedder, "RF CMOS-MEMS Capacitor Having Large Tuning Range", *2003 IEEE Transducers*, June 8-12, pp. 851-854.

[90] H. Xie and G.K. Fedder, "A CMOS Z-axis Capacitive Accelerometer with Comb-Finger Sensing", in *Proc. of the 13th IEEE Inter. Conf. on Micro Electro Mechanical Systems (MEMS 2000)*, Jan. 23-27, 2000, Miyazaki, Japan.

[91] C.T.-C. Nguyen, and R.T. Howe, "An Integrated CMOS Micromechanical Resonator High-Q Oscillator", *IEEE J. Solid-State Circuits*, vol. 34, No. 4, April 1999, pp.440-455.

[92] C.T.-C. Nguyen, "Transceiver Front-End Architectures Using Vibrating Micromechanical Signal Processors", *Silicon Monolithic Integrated Circuits in RF Systems, 2001*, Digest of Papers, pp. 23 -32.

[93] R.A. Johnson, *Mechanical Filters in Electronics*, John Wiley & Sons, New York, New York, 1983.

[94] D. Becher, et.al., "Reliability Study of Low-Voltage RF MEMS Switches", in *2002 GaAs MANTECH Conference*, St. Louis, MO, USA, GaAs MANTECH Inc, 2002, pp. 54-57.

[95] Y. Jang and Y. Kim, "Design and Fabrication of a Micromirror Using Silicon Bulk Micromachining for Out-of-Plane Right Angle Reflection", *Optical MEMs, 2002. Conference Digest. 2002 IEEE/LEOS International Conference on*, 2002, pp.79 -80.

[96] J. Lee, et.al., "SOI-Fabrication of Scanning Mirror for Laser Display", *Optical MEMs, 2002. Conference Digest. 2002 IEEE/LEOS International Conference on*, 2002, pp. 153 -154.

[97] H. Camon and F. Larnaudie, "Fabrication, Simulation and Experiment of a Rotating Electrostatic Silicon Mirror

with Large Angular Deflection", *MEMS 2000*, Jan 23-27, pp. 645 -650.

[98] E.E. Hui, R.T. Howe, M.S. Rodgers, "Single-step Assembly of Complex 3-D Microstructures", in *Proceedings IEEE Thirteenth Annual International Conference on Micro Electro Mechanical Systems*, Piscataway, NJ, USA, 2000, pp. 602-607.

[99] T.W. Krygowski, M.S. Rodgers, et.al., " Low-Voltage Rotary Actuator Fabricated Using a Five-Level Polysilicon Surface Micromachining Technology", in *International Electron Devices Meeting 1999,* Piscataway, NJ, USA, pp. 697-700.

[100] D.J. Keating, Yie He, N. Finch, et.al., "Numerical Simulation of Micro-assembly of MEMS Devices and Post Assembly Electromechanical Actuation", Proceedings of the SPIE - The International Society for Optical Engineering, 2000, pp. 63-70.

[101] S.L. Miller, J.J. Sniegowski, et.al., "Friction in surface micromachined microengines" *Proceedings of the SPIE - The International Society for Optical Engineering* 1996, pp. 197-204.

[102] J.V. Clark, K.S. Pister, et.al., "Addressing the Need for Complex MEMS Design", MEMS'02, Jan. 20-24, 2002, Las Vegas, NV, USA, pp 204-209.

[103] Johnson, et.al., "Electrophysics of Micromechanical Comb Actuators", *Journal of Microelectromechanical Systems* 4, no. 1, March 1995, pp 49-59.

Appendix

This appendix gives the Verilog-A codes for the atomic-level NODAS models presented in this thesis, including anchor model, layout_origin model, beam model, plate model and electrostatic gap model. Detailed comments are provided in each model, to clarify the notations and conventions used in coding and to help the readers link the codes to the theoretical equations given in the thesis.

A.1 Anchor Model

The anchors are the fixed points on the chip where the movable parts of the suspended MEMS devices are attached to. This is an ideal model which specifies zero translational and rotational displacements.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
module anchor3D(x, phi);
// definition of bus pins
inout [0:2] x; kinematic [0:2] x;
inout [0:2] phi; rotational [0:2] phi;
// definition of user-specifiable parameters
parameter real l = 0, w = 0;
// beginning of analog block
analog begin
Pos(x[0]) <+ 0; Pos(x[1]) <+ 0; Pos(x[2]) <+ 0; // zero translational displacements
Theta(phi[0]) <+ 0; Theta(phi[1]) <+ 0; Theta(phi[2]) <+ 0; // zero rotational displacements
end  // end of analog block
endmodule  // end of module
```

A.2 Layout_origin Model

As discussed in Chapter 3.2, the layout_origin specifies the origin of the chip layout. It's used for automatic layout generation and must be connected to one of the anchors in the schematic. Currently, it only supports zero values for parameter X_0 and Y_0.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
module layout_origin(x);
// definition of bus pins
inout [0:1] x;
kinematic [0:1] x;
// definition of user-specifiable parameters
parameter real X0 = 0;
parameter real Y0 = 0;
// beginning of analog
analog begin
            // empty analog block
end  // of analog block
endmodule  // of module
```

A.3 Linear Beam Model with Shear

The beam model given below is the 3D linear beam model, which captures the linear axial, lateral and torsional mechanics of beam elements with rectangular cross sections. This model is presented to show how the beam equations given in Chapter 3.3 are translated into Verilog-A codes and how the algorithm structure of the model given in Chapter 3.3 is implemented in Verilog-A.

Shear effects are also included in this model. A parameter *flag_shear* is introduced to switch on/off the shear effects. This parameter is a bi-value integer with possible values of 1 or 0. If *flag_shear* = 1, shear effects are included in the simulation; if *flag_shear* = 0, shear effects are not included in the simulation. In this way, the user can choose whether to include shear effects in the simulation according to the geometric sizes of the beam and the accuracy requirements of the simulation. As this section focuses on the implementation of the linear beam algorithm structure, features such as trapezoidal cross sections and multiple conductor layers are not included and will be presented in later sections.

The Verilog-A code for this model is:

```
// include header files
// constants.h -- physical constants
// design.h -- default parameter values
// discipline.h -- across/variable natures for multiple disciplines
// process.h -- process info including thickness, density, E, etc.

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

// model name and bus pin definition, multiple disciplines
module beam3D_lin(phia, phib, xa, xb, va, vb, ta, tb);
inout [0:2] phia;
rotational [0:2] phia;
inout [0:2] phib;
rotational [0:2] phib;
inout [0:2] xa;
kinematic [0:2] xa;
inout [0:2] xb;
kinematic [0:2] xb;
inout va, vb;
electrical va, vb;
// temperature pins are defined for inclusion of physical effects in thermal domain
inout ta;
thermal ta;
```

```
inout tb;
thermal tb;

// definition of user-specifiable parameters (with default values)
// Euler angles (alpha, beta and gamma, in degrees) are used
// to specify beam orientation and to form coordination transformation matrix
// Order of rotation:
// step1: rotate by gamma degree about z-axis
// step2: rotate by beta  degree about y-axis
// step3: rotate by alpha degree about x-axis
// default beam position:
// in plane, length is along x-axis, width is along y-axis, t is along z-axis
// node a is on the left side, node b is on the right side, with gamma=0
// angle values should always match with node positions, eg:
// to make a vertical beam, rotate the symbol counterclockwisely by 90 degree,
// node a should be at the bottom, node b should be on the top, give gamma=90

parameter integer flag_shear = 0;

parameter real l = 100e-6;
parameter real w = 2e-6;
parameter real thickness = 'default_thickness;
parameter real E = 'default_E;
parameter real density = 'default_density;

parameter real alpha = 0;
parameter real beta = 0;
parameter real gamma = 0;

parameter real resistivity = 'default_resistivity;
parameter real bloat = 4e-6;
parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real Poisson_ratio = 'default_Poisson_ratio;

parameter real Xc = 0;
parameter real Yc = 0;

// definition of internal variables

real Iz, Iy, G, It, m0y, m0z, ea, xt, xw, x, sum, area, l_corner, mass;
integer i;
real dampx, dampy, dampz, scale, resistance;

// variables for shear deformation
real Asy, Sy, Srz, Asz, Sz, Sry, m0y, m0z;

real cos_alpha, sin_alpha, cos_beta, sin_beta, cos_gamma, sin_gamma;
real l1, m1, n1, l2, m2, n2, l3, m3, n3;
real inv_l1, inv_m1, inv_n1, inv_l2, inv_m2, inv_n2, inv_l3, inv_m3, inv_n3;

real chip_xa, chip_ya, chip_za, chip_xb, chip_yb, chip_zb;
real l_xa, l_ya, l_za, l_xb, l_yb, l_zb;

real chip_phixa, chip_phiya, chip_phiza, chip_phixb, chip_phiyb, chip_phizb;
real l_phixa, l_phiya, l_phiza, l_phixb, l_phiyb, l_phizb;

real l_Fkxb, l_Fkxa, l_Fkyb, l_Fkya, l_Fkzb, l_Fkza;
real l_Mkxa, l_Mkya, l_Mkza, l_Mkxb, l_Mkyb, l_Mkzb;

real l_Fmxb, l_Fmxa, l_Fmyb, l_Fmya, l_Fmzb, l_Fmza;
real l_Mmxa, l_Mmya, l_Mmza, l_Mmxb, l_Mmyb, l_Mmzb;

real l_Fbxb, l_Fbxa, l_Fbyb, l_Fbya, l_Fbzb, l_Fbza;
real l_Mbxa, l_Mbya, l_Mbza, l_Mbxb, l_Mbyb, l_Mbzb;
```

```
real l_Fxb, l_Fxa, l_Fyb, l_Fya, l_Fzb, l_Fza;
real l_Mxa, l_Mya, l_Mza, l_Mxb, l_Myb, l_Mzb;
real chip_Fxb, chip_Fxa, chip_Fyb, chip_Fya, chip_Fzb, chip_Fza;
real chip_Mxa, chip_Mya, chip_Mza, chip_Mxb, chip_Myb, chip_Mzb;

kinematic Vxa, Vxb, Vya, Vyb, Vza, Vzb;
rotational_omega  Vphixa, Vphixb, Vphiya, Vphiyb, Vphiza, Vphizb;

real k_1_1;
real k_2_1, k_2_2;
real k_3_1, k_3_2, k_3_3;
real k_4_1, k_4_2, k_4_3, k_4_4;
real k_5_1, k_5_2, k_5_3, k_5_4, k_5_5;
real k_6_1, k_6_2, k_6_3, k_6_4, k_6_5, k_6_6;
real k_7_1, k_7_2, k_7_3, k_7_4, k_7_5, k_7_6, k_7_7;
real k_8_1, k_8_2, k_8_3, k_8_4, k_8_5, k_8_6, k_8_7, k_8_8;
real k_9_1, k_9_2, k_9_3, k_9_4, k_9_5, k_9_6, k_9_7, k_9_8, k_9_9;
real k_10_1,k_10_2,k_10_3,k_10_4,k_10_5,k_10_6,k_10_7,k_10_8,k_10_9,k_10_10;
real k_11_1,k_11_2,k_11_3,k_11_4,k_11_5,k_11_6,k_11_7,k_11_8,k_11_9,k_11_10,k_11_11;
real k_12_1,k_12_2,k_12_3,k_12_4,k_12_5,k_12_6,k_12_7,k_12_8,k_12_9,k_12_10,k_12_11,k_12_12;


real m_1_1;
real m_2_1, m_2_2;
real m_3_1, m_3_2, m_3_3;
real m_4_1, m_4_2, m_4_3, m_4_4;
real m_5_1, m_5_2, m_5_3, m_5_4, m_5_5;
real m_6_1, m_6_2, m_6_3, m_6_4, m_6_5, m_6_6;
real m_7_1, m_7_2, m_7_3, m_7_4, m_7_5, m_7_6, m_7_7;
real m_8_1, m_8_2, m_8_3, m_8_4, m_8_5, m_8_6, m_8_7, m_8_8;
real m_9_1, m_9_2, m_9_3, m_9_4, m_9_5, m_9_6, m_9_7, m_9_8, m_9_9;
real m_10_1,m_10_2,m_10_3,m_10_4,m_10_5,m_10_6,m_10_7,m_10_8,m_10_9,m_10_10;
real m_11_1,m_11_2,m_11_3,m_11_4,m_11_5,m_11_6,m_11_7,m_11_8,m_11_9,m_11_10,m_11_11;
real m_12_1,m_12_2,m_12_3,m_12_4,m_12_5,m_12_6,m_12_7,m_12_8,m_12_9,m_12_10,m_12_11,m_12_12;

analog begin

l_corner = l+0.3*w*(1-flag_shear);
resistance = resistivity*l_corner/w;

area = thickness*w;
mass = density*thickness*w*l_corner;

ea = E*area;  // coef used in [k] matrix

Iy = 1/12.0*w*thickness*thickness*thickness;
Iz = 1/12.0*thickness*w*w*w;

// torsional moment of inertia
    xt = min(thickness,w);
    xw = max(thickness,w);
    x  = xt/xw;
    sum = 0;
    for (i=1; i<20; i=i+2)
    sum = sum + tanh(i*'M_PI/x/2)/pow(i*1.0,5);
    It = pow(xt,3)*xw*(1-192/pow('M_PI,5)*x*sum)/3;

// shear modulus
G = E/2.0/(1+Poisson_ratio);

scale = 1e6;

dampx = visc_air*l_corner*(w+bloat)/air_gap;
dampy = visc_air*w*(l_corner+bloat)/air_gap;
```

```
dampz = visc_air*thickness*(l_corner+bloat)/air_gap;

//layout euler angles
cos_alpha = cos(alpha /180*'M_PI);
cos_beta  = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta  = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// translational displacement in chip frame
chip_xa = Pos(xa[0]);
chip_ya = Pos(xa[1]);
chip_za = Pos(xa[2]);
chip_xb = Pos(xb[0]);
chip_yb = Pos(xb[1]);
chip_zb = Pos(xb[2]);

// angular displacement in chip frame
chip_phixa = scale*Theta(phia[0]);
chip_phiya = scale*Theta(phia[1]);
chip_phiza = scale*Theta(phia[2]);
chip_phixb = scale*Theta(phib[0]);
chip_phiyb = scale*Theta(phib[1]);
chip_phizb = scale*Theta(phib[2]);

// transform from chip frame into local frame
l_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
l_ya = l2*chip_xa + m2*chip_ya + n2*chip_za;
l_za = l3*chip_xa + m3*chip_ya + n3*chip_za;
l_xb = l1*chip_xb + m1*chip_yb + n1*chip_zb;
l_yb = l2*chip_xb + m2*chip_yb + n2*chip_zb;
l_zb = l3*chip_xb + m3*chip_yb + n3*chip_zb;

l_phixa = l1*chip_phixa + m1*chip_phiya + n1*chip_phiza;
l_phiya = l2*chip_phixa + m2*chip_phiya + n2*chip_phiza;
l_phiza = l3*chip_phixa + m3*chip_phiya + n3*chip_phiza;
l_phixb = l1*chip_phixb + m1*chip_phiyb + n1*chip_phizb;
l_phiyb = l2*chip_phixb + m2*chip_phiyb + n2*chip_phizb;
l_phizb = l3*chip_phixb + m3*chip_phiyb + n3*chip_phizb;


// linear stiffness matrix [k], with shear deformation
```

```
// variables for shear deformation
Asy = 2.0/3.0*area; //effective shear area
Sy  = 1.0*12.0*E*Iz/G/pow(l_corner,2)/Asy * flag_shear; //shear deformation parameter
Srz = 1.0*pow(Iz/area, 0.5) * flag_shear; //radius of gyration about z-axis

Asz = 2.0/3.0*area; //effective shear area
Sz  = 1.0*12.0*E*Iy/G/pow(l_corner,2)/Asz * flag_shear; //shear deformation parameter
Sry = 1.0*pow(Iy/area, 0.5) * flag_shear; //radius of gyration about z-axis

k_1_1 = ea/l_corner;
k_2_1 = 0;
k_3_1 = 0;
k_4_1 = 0;
k_5_1 = 0;
k_6_1 = 0;
k_7_1 = -ea/l_corner;
k_8_1 = 0;
k_9_1 = 0;
k_10_1 = 0;
k_11_1 = 0;
k_12_1 = 0;

k_2_2 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_3_2 = 0;
k_4_2 = 0;
k_5_2 = 0;
k_6_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k_7_2 = 0;
k_8_2 = -12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_9_2 = 0;
k_10_2 = 0;
k_11_2 = 0;
k_12_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);


k_3_3 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_4_3 = 0;
k_5_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_6_3 = 0;
k_7_3 = 0;
k_8_3 = 0;
k_9_3 = -12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_10_3 = 0;
k_11_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_12_3 = 0;

k_4_4 = G*It/l_corner;
k_5_4 = 0;
k_6_4 = 0;
k_7_4 = 0;
k_8_4 = 0;
k_9_4 = 0;
k_10_4 = -G*It/l_corner;
k_11_4 = 0;
k_12_4 = 0;

k_5_5 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k_6_5 = 0;
k_7_5 = 0;
k_8_5 = 0;
k_9_5 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_10_5 = 0;
k_11_5 = (2.0-Sz)*E*Iy/l_corner/(1+Sz);
k_12_5 = 0;
```

```
k_6_6 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);
k_7_6 = 0;
k_8_6 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k_9_6 = 0;
k_10_6 = 0;
k_11_6 = 0;
k_12_6 = (2.0-Sy)*E*Iz/l_corner/(1+Sy);

k_7_7 = ea/l_corner;
k_8_7 = 0;
k_9_7 = 0;
k_10_7 = 0;
k_11_7 = 0;
k_12_7 = 0;

k_8_8 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_9_8 = 0;
k_10_8 = 0;
k_11_8 = 0;
k_12_8 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);

k_9_9 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_10_9 = 0;
k_11_9 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_12_9 = 0;

k_10_10 = G*It/l_corner;
k_11_10 = 0;
k_12_10 = 0;

k_11_11 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k_12_11 = 0;

k_12_12 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);


// [Fk]=[k][x], spring forces/moments in local frame

l_Fkxa = k_1_1*l_xa+k_2_1*l_ya+k_3_1*l_za
        +k_4_1*l_phixa+k_5_1*l_phiya+k_6_1*l_phiza
        +k_7_1*l_xb+k_8_1*l_yb+k_9_1*l_zb
        +k_10_1*l_phixb+k_11_1*l_phiyb+k_12_1*l_phizb;

l_Fkxb = k_7_1*l_xa+k_7_2*l_ya+k_7_3*l_za
        +k_7_4*l_phixa+k_7_5*l_phiya+k_7_6*l_phiza
        +k_7_7*l_xb+k_8_7*l_yb+k_9_7*l_zb
        +k_10_7*l_phixb+k_11_7*l_phiyb+k_12_7*l_phizb;

l_Fkya = k_2_1*l_xa+k_2_2*l_ya+k_3_2*l_za
        +k_4_2*l_phixa+k_5_2*l_phiya+k_6_2*l_phiza
        +k_7_2*l_xb+k_8_2*l_yb+k_9_2*l_zb
        +k_10_2*l_phixb+k_11_2*l_phiyb+k_12_2*l_phizb;

l_Fkyb = k_8_1*l_xa+k_8_2*l_ya+k_8_3*l_za
        +k_8_4*l_phixa+k_8_5*l_phiya+k_8_6*l_phiza
        +k_8_7*l_xb+k_8_8*l_yb+k_9_8*l_zb
        +k_10_8*l_phixb+k_11_8*l_phiyb+k_12_8*l_phizb;

l_Fkza = k_3_1*l_xa+k_3_2*l_ya+k_3_3*l_za
        +k_4_3*l_phixa+k_5_3*l_phiya+k_6_3*l_phiza
        +k_7_3*l_xb+k_8_3*l_yb+k_9_3*l_zb
        +k_10_3*l_phixb+k_11_3*l_phiyb+k_12_3*l_phizb;

l_Fkzb = k_9_1*l_xa+k_9_2*l_ya+k_9_3*l_za
```

Modeling and Simulation for Design of Suspended MEMS   233

```
            +k_9_4*l_phixa+k_9_5*l_phiya+k_9_6*l_phiza
            +k_9_7*l_xb+k_9_8*l_yb+k_9_9*l_zb
            +k_10_9*l_phixb+k_11_9*l_phiyb+k_12_9*l_phizb;

l_Mkxa = k_4_1*l_xa+k_4_2*l_ya+k_4_3*l_za
            +k_4_4*l_phixa+k_5_4*l_phiya+k_6_4*l_phiza
            +k_7_4*l_xb+k_8_4*l_yb+k_9_4*l_zb
            +k_10_4*l_phixb+k_11_4*l_phiyb+k_12_4*l_phizb;

l_Mkxb = k_10_1*l_xa+k_10_2*l_ya+k_10_3*l_za
            +k_10_4*l_phixa+k_10_5*l_phiya+k_10_6*l_phiza
            +k_10_7*l_xb+k_10_8*l_yb+k_10_9*l_zb
            +k_10_10*l_phixb+k_11_10*l_phiyb+k_12_10*l_phizb;

l_Mkya = k_5_1*l_xa+k_5_2*l_ya+k_5_3*l_za
            +k_5_4*l_phixa+k_5_5*l_phiya+k_6_5*l_phiza
            +k_7_5*l_xb+k_8_5*l_yb+k_9_5*l_zb
            +k_10_5*l_phixb+k_11_5*l_phiyb+k_12_5*l_phizb;

l_Mkyb = k_11_1*l_xa+k_11_2*l_ya+k_11_3*l_za
            +k_11_4*l_phixa+k_11_5*l_phiya+k_11_6*l_phiza
            +k_11_7*l_xb+k_11_8*l_yb+k_11_9*l_zb
            +k_11_10*l_phixb+k_11_11*l_phiyb+k_12_11*l_phizb;

l_Mkza = k_6_1*l_xa+k_6_2*l_ya+k_6_3*l_za
            +k_6_4*l_phixa+k_6_5*l_phiya+k_6_6*l_phiza
            +k_7_6*l_xb+k_8_6*l_yb+k_9_6*l_zb
            +k_10_6*l_phixb+k_11_6*l_phiyb+k_12_6*l_phizb;

l_Mkzb = k_12_1*l_xa+k_12_2*l_ya+k_12_3*l_za
            +k_12_4*l_phixa+k_12_5*l_phiya+k_12_6*l_phiza
            +k_12_7*l_xb+k_12_8*l_yb+k_12_9*l_zb
            +k_12_10*l_phixb+k_12_11*l_phiyb+k_12_12*l_phizb;


// mass matrix [m], with shear effects

m0y = density*thickness*w*l_corner/pow(1+Sy,2);
m0z = density*thickness*w*l_corner/pow(1+Sz,2);

m_1_1 = 1.0/3.0*density*thickness*w*l_corner;
m_2_1 = 0;
m_3_1 = 0;
m_4_1 = 0;
m_5_1 = 0;
m_6_1 = 0;
m_7_1 = 1.0/6.0*density*thickness*w*l_corner;
m_8_1 = 0;
m_9_1 = 0;
m_10_1 = 0;
m_11_1 = 0;
m_12_1 = 0;

m_2_2 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
m_3_2 = 0;
m_4_2 = 0;
m_5_2 = 0;
m_6_2 = m0y*l_corner*((11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));
m_7_2 = 0;
m_8_2 = m0y*((9.0/70.0+3.0/10.0*Sy+1.0/6.0*Sy*Sy)-pow(Srz/l_corner,2)*6.0/5.0);
m_9_2 = 0;
m_10_2 = 0;
m_11_2 = 0;
m_12_2 = m0y*l_corner*(-(13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));
```

```
m_3_3 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
m_4_3 = 0;
m_5_3 = -m0z*l_corner*((11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
m_6_3 = 0;
m_7_3 = 0;
m_8_3 = 0;
m_9_3 = m0z*((9.0/70.0+3.0/10.0*Sz+1.0/6.0*Sz*Sz)-pow(Sry/l_corner,2)*6.0/5.0);
m_10_3 = 0;
m_11_3 = -m0z*l_corner*(-(13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
m_12_3 = 0;

m_4_4 = (Iy+Iz)/3.0*density*l_corner;
m_5_4 = 0;
m_6_4 = 0;
m_7_4 = 0;
m_8_4 = 0;
m_9_4 = 0;
m_10_4 = (Iy+Iz)/6.0*density*l_corner;
m_11_4 = 0;
m_12_4 = 0;

m_5_5 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/3.0*Sz*Sz));
m_6_5 = 0;
m_7_5 = 0;
m_8_5 = 0;
m_9_5 = -m0z*l_corner*((13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
m_10_5 = 0;
m_11_5 = m0z*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/30.0-1.0/6.0*Sz+1.0/6.0*Sz*Sz));
m_12_5 = 0;

m_6_6 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/3.0*Sy*Sy));
m_7_6 = 0;
m_8_6 = m0y*l_corner*((13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));
m_9_6 = 0;
m_10_6 = 0;
m_11_6 = 0;
m_12_6 = m0y*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/30.0-1.0/6.0*Sy+1.0/6.0*Sy*Sy));

m_7_7 = 1.0/3.0*density*thickness*w*l_corner;
m_8_7 = 0;
m_9_7 = 0;
m_10_7 = 0;
m_11_7 = 0;
m_12_7 = 0;

m_8_8 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
m_9_8 = 0;
m_10_8 = 0;
m_11_8 = 0;
m_12_8 = m0y*l_corner*(-(11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));

m_9_9 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
m_10_9 = 0;
m_11_9 = -m0z*l_corner*(-(11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
m_12_9 = 0;

m_10_10 = (Iy+Iz)/3.0*density*l_corner;
m_11_10 = 0;
m_12_10 = 0;

m_11_11 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/
```

```
3.0*Sz*Sz));
     m_12_11 = 0;

     m_12_12 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/
3.0*Sy*Sy));


     Pos(Vxa) <+ ddt(l_xa);
     Pos(Vxb) <+ ddt(l_xb);
     Pos(Vya) <+ ddt(l_ya);
     Pos(Vyb) <+ ddt(l_yb);
     Pos(Vza) <+ ddt(l_za);
     Pos(Vzb) <+ ddt(l_zb);

     Omega(Vphixa) <+ ddt(l_phixa);
     Omega(Vphixb) <+ ddt(l_phixb);
     Omega(Vphiya) <+ ddt(l_phiya);
     Omega(Vphiyb) <+ ddt(l_phiyb);
     Omega(Vphiza) <+ ddt(l_phiza);
     Omega(Vphizb) <+ ddt(l_phizb);

     // [Fm]=[m][a], inertial forces/moments in local frame

     l_Fmxa = m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
     l_Fmxb = m_7_1*ddt(Pos(Vxa)) + m_7_7*ddt(Pos(Vxb));

     l_Fmya = m_6_2*ddt(Omega(Vphiza)) + m_12_2*ddt(Omega(Vphizb))
          + m_2_2*ddt(Pos(Vya)) + m_8_2*ddt(Pos(Vyb));
     l_Fmyb = m_8_6*ddt(Omega(Vphiza)) + m_12_8*ddt(Omega(Vphizb))
          + m_8_2*ddt(Pos(Vya)) + m_8_8*ddt(Pos(Vyb));

     l_Fmza = m_5_3*ddt(Omega(Vphiya)) + m_11_3*ddt(Omega(Vphiyb))
          + m_3_3*ddt(Pos(Vza)) + m_9_3*ddt(Pos(Vzb));
     l_Fmzb = m_9_5*ddt(Omega(Vphiya)) + m_11_9*ddt(Omega(Vphiyb))
          + m_9_3*ddt(Pos(Vza)) + m_9_9*ddt(Pos(Vzb));

     l_Mmxa = m_4_4*ddt(Omega(Vphixa)) + m_10_4*ddt(Omega(Vphixb));
     l_Mmxb = m_10_4*ddt(Omega(Vphixa)) + m_10_10*ddt(Omega(Vphixb));

     l_Mmya = m_5_5*ddt(Omega(Vphiya)) + m_11_5*ddt(Omega(Vphiyb))
     + m_5_3*ddt(Pos(Vza))+ m_9_5*ddt(Pos(Vzb));
     l_Mmyb = m_11_5*ddt(Omega(Vphiya)) + m_11_11*ddt(Omega(Vphiyb))
     + m_11_3*ddt(Pos(Vza)) + m_11_9*ddt(Pos(Vzb));

     l_Mmza = m_6_6*ddt(Omega(Vphiza)) + m_12_6*ddt(Omega(Vphizb))
     + m_6_2*ddt(Pos(Vya))+ m_8_6*ddt(Pos(Vyb));
     l_Mmzb = m_12_6*ddt(Omega(Vphiza)) + m_12_12*ddt(Omega(Vphizb))
     + m_12_2*ddt(Pos(Vya)) + m_12_8*ddt(Pos(Vyb));

     // [Fb]=[B][v], damping forces/moments in local frame
     //using damping matrix with shear effect

     l_Fbxa = dampx/mass*(m_1_1*ddt(l_xa) + m_7_1*ddt(l_xb));
     l_Fbxb = dampx/mass*(m_7_1*ddt(l_xa) + m_7_7*ddt(l_xb));

     l_Fbya = dampy/mass*(m_6_2*ddt(l_phiza) + m_12_2*ddt(l_phizb)
          + m_2_2*ddt(l_ya) + m_8_2*ddt(l_yb));
     l_Fbyb = dampy/mass*(m_8_6*ddt(l_phiza) + m_12_8*ddt(l_phizb)
          + m_8_2*ddt(l_ya) + m_8_8*ddt(l_yb));

     l_Fbza = dampz/mass*(m_5_3*ddt(l_phiya) + m_11_3*ddt(l_phiyb)
          + m_3_3*ddt(l_za) + m_9_3*ddt(l_zb));
     l_Fbzb = dampz/mass*(m_9_5*ddt(l_phiya) + m_11_9*ddt(l_phiyb)
          + m_9_3*ddt(l_za) + m_9_9*ddt(l_zb));
```

```
l_Mbxa = dampx/mass*(m_4_4*ddt(l_phixa) + m_10_4*ddt(l_phixb));
l_Mbxb = dampx/mass*(m_10_4*ddt(l_phixa) + m_10_10*ddt(l_phixb));

l_Mbya = dampz/mass*(m_5_5*ddt(l_phiya) + m_11_5*ddt(l_phiyb)
+ m_5_3*ddt(l_za)+ m_9_5*ddt(l_zb));
l_Mbyb = dampz/mass*(m_11_5*ddt(l_phiya) + m_11_11*ddt(l_phiyb)
+ m_11_3*ddt(l_za) + m_11_9*ddt(l_zb));

l_Mbza = dampy/mass*(m_6_6*ddt(l_phiza) + m_12_6*ddt(l_phizb)
+ m_6_2*ddt(l_ya)+ m_8_6*ddt(l_yb));
l_Mbzb = dampy/mass*(m_12_6*ddt(l_phiza) + m_12_12*ddt(l_phizb)
+ m_12_2*ddt(l_ya) + m_12_8*ddt(l_yb));

// sum up spring, inertial and damping forces
// transform from local frame back to the chip frame

chip_Fxb = inv_l1*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m1*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n1*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fyb = inv_l2*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m2*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n2*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fzb = inv_l3*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m3*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n3*(l_Fkzb+l_Fmzb+l_Fbzb);

chip_Fxa = inv_l1*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m1*(l_Fkya+l_Fmya+l_Fbya)+inv_n1*(l_Fkza+l_Fmza+l_Fbza);
chip_Fya = inv_l2*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m2*(l_Fkya+l_Fmya+l_Fbya)+inv_n2*(l_Fkza+l_Fmza+l_Fbza);
chip_Fza = inv_l3*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m3*(l_Fkya+l_Fmya+l_Fbya)+inv_n3*(l_Fkza+l_Fmza+l_Fbza);

chip_Mxb = inv_l1*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m1*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n1*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Myb = inv_l2*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m2*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n2*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Mzb = inv_l3*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m3*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n3*(l_Mkzb+l_Mmzb+l_Mbzb);

chip_Mxa = inv_l1*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m1*(l_Mkya+l_Mmya+l_Mbya)+inv_n1*(l_Mkza+l_Mmza+l_Mbza);
chip_Mya = inv_l2*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m2*(l_Mkya+l_Mmya+l_Mbya)+inv_n2*(l_Mkza+l_Mmza+l_Mbza);
chip_Mza = inv_l3*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m3*(l_Mkya+l_Mmya+l_Mbya)+inv_n3*(l_Mkza+l_Mmza+l_Mbza);

// forces and moments applied to the beam ends
F(xa[0]) <+ -chip_Fxa;
F(xa[1]) <+ -chip_Fya;
F(xa[2]) <+ -chip_Fza;
F(xb[0]) <+ -chip_Fxb;
F(xb[1]) <+ -chip_Fyb;
F(xb[2]) <+ -chip_Fzb;

Tau(phia[2]) <+ -chip_Mza;
Tau(phib[2]) <+ -chip_Mzb;
Tau(phia[1]) <+ -chip_Mya;
Tau(phib[1]) <+ -chip_Myb;
Tau(phia[0]) <+ -chip_Mxa;
Tau(phib[0]) <+ -chip_Mxb;

 //Calculate beam's electrical relation, I=V/R
 I(vb, va) <+ V(vb, va)/resistance;

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

A.4 Nonlinear Beam Model with Shear Effect

The model given below is the 3D nonlinear beam model, which captures the geometric beam nonlinearity caused by axial stress stiffening and large geometric deflections. The code shows the implementation in Verilog-A of the nonlinear beam theory and algorithm structure given in Chapter 3.3. The basic flow of the model is the same as in the linear beam model, while there are additional steps including dynamic rotation, calculation of effective beam length and axial force and modification of stiffness matrices. These steps specific to the nonlinear beam model are the focus of this section.

Again, shear effects are included in this model. A bi-value integer parameter *flag_shear* is used to switch on/off the shear effects. As this section focuses on the implementation of the nonlinear beam algorithm structure, features such as trapezoidal cross sections and multiple conductor layers are not included and will be presented in later sections.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
'include "../../design.h"
'include "../../process.h"

module beam3D_nlin(phia, phib, xa, xb, va, vb, ta, tb);

inout [0:2] phia;
rotational [0:2] phia;
inout [0:2] phib;
rotational [0:2] phib;
inout [0:2] xa;
kinematic [0:2] xa;
inout [0:2] xb;
kinematic [0:2] xb;
inout va, vb;
electrical va, vb;
inout ta;
thermal ta;
inout tb;
thermal tb;

parameter integer flag_shear = 0;

parameter real l = 100e-6;
parameter real w = 2e-6;
parameter real thickness = 'default_thickness;
parameter real E = 'default_E;
parameter real density = 'default_density;
```

```
parameter real alpha = 0;
parameter real beta = 0;
parameter real gamma = 0;

parameter real resistivity = 'default_resistivity;
parameter real bloat = 4e-6;
parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real Poisson_ratio = 'default_Poisson_ratio;

parameter real Xc = 0;
parameter real Yc = 0;

// definition of internal variables

real Iz, Iy, G, It, ea, xt, xw, x, sum, area, mass, l_corner;
integer i;

real dampx, dampy, dampz, scale, resistance;

// variables for shear deformation
real Asy, Sy, Srz, Asz, Sz, Sry, m0y, m0z;

real cos_alpha, sin_alpha, cos_beta, sin_beta, cos_gamma, sin_gamma;
real l1, m1, n1, l2, m2, n2, l3, m3, n3;
real inv_l1, inv_m1, inv_n1, inv_l2, inv_m2, inv_n2, inv_l3, inv_m3, inv_n3;

real chip_xa, chip_ya, chip_za, chip_xb, chip_yb, chip_zb;
real layout_xa, layout_ya, layout_za, layout_xb, layout_yb, layout_zb;
real l_xa, l_xb, l_ya,l_za, l_yb,l_zb;

real chip_phixa, chip_phiya, chip_phiza, chip_phixb, chip_phiyb, chip_phizb;
real layout_phixa, layout_phiya, layout_phiza, layout_phixb, layout_phiyb, layout_phizb;
real l_phixa, l_phixb, l_phiya,l_phiza, l_phiyb,l_phizb;

real chip_Fkxb, chip_Fkxa, chip_Fkyb, chip_Fkya, chip_Fkzb, chip_Fkza;
real chip_Mkxa, chip_Mkya, chip_Mkza, chip_Mkxb, chip_Mkyb, chip_Mkzb;
real layout_Fkxb, layout_Fkxa, layout_Fkyb, layout_Fkya, layout_Fkzb, layout_Fkza;
real layout_Mkxa, layout_Mkya, layout_Mkza, layout_Mkxb, layout_Mkyb, layout_Mkzb;

real chip_Fmxb, chip_Fmxa, chip_Fmyb, chip_Fmya, chip_Fmzb, chip_Fmza;
real chip_Mmxa, chip_Mmya, chip_Mmza, chip_Mmxb, chip_Mmyb, chip_Mmzb;
real layout_Fmxb, layout_Fmxa, layout_Fmyb, layout_Fmya, layout_Fmzb, layout_Fmza;
real layout_Mmxa, layout_Mmya, layout_Mmza, layout_Mmxb, layout_Mmyb, layout_Mmzb;

real chip_Fbxb, chip_Fbxa, chip_Fbyb, chip_Fbya, chip_Fbzb, chip_Fbza;
real chip_Mbxa, chip_Mbya, chip_Mbza, chip_Mbxb, chip_Mbyb, chip_Mbzb;
real layout_Fbxb, layout_Fbxa, layout_Fbyb, layout_Fbya, layout_Fbzb, layout_Fbza;
real layout_Mbxa, layout_Mbya, layout_Mbza, layout_Mbxb, layout_Mbyb, layout_Mbzb;

real layout_Fxb, layout_Fxa, layout_Fyb, layout_Fya, layout_Fzb, layout_Fza;
real layout_Mxa, layout_Mya, layout_Mza, layout_Mxb, layout_Myb, layout_Mzb;
real chip_Fxb, chip_Fxa, chip_Fyb, chip_Fya, chip_Fzb, chip_Fza;
real chip_Mxa, chip_Mya, chip_Mza, chip_Mxb, chip_Myb, chip_Mzb;

kinematic Vxa, Vxb, Vya, Vyb, Vza, Vzb;
rotational_omega  Vphixa, Vphixb, Vphiya, Vphiyb, Vphiza, Vphizb;
kinematic new_Fmxa, new_Fmxb, new_Fmya, new_Fmyb, new_Fmza, new_Fmzb;
rotational new_Mmxa, new_Mmxb, new_Mmya, new_Mmyb, new_Mmza, new_Mmzb;
kinematic new_Fbxa, new_Fbxb, new_Fbya, new_Fbyb, new_Fbza, new_Fbzb;
rotational new_Mbxa, new_Mbxb, new_Mbya, new_Mbyb, new_Mbza, new_Mbzb;

// add new variables for dynamatic rotation
real new_l1, new_m1, new_n1, new_l2, new_m2, new_n2, new_l3, new_m3, new_n3;
real new_inv_l1, new_inv_m1, new_inv_n1, new_inv_l2, new_inv_m2, new_inv_n2, new_inv_l3, new_inv_m3, new_inv_n3;
```

```
real rphix, rphiy, rphiz, rphi, rx, ry, rz, rs, rc, rt;

real new_l_phixa, new_l_phixb, new_l_phiya,new_l_phiza, new_l_phiyb,new_l_phizb;
real new_l_xa, new_l_xb, new_l_ya, new_l_yb, new_l_za, new_l_zb;
real new_Fkxa, new_Fkxb, new_Fkya, new_Fkyb, new_Fkza, new_Fkzb;
real new_Mkxa, new_Mkxb, new_Mkya, new_Mkyb, new_Mkza, new_Mkzb;

real lm, lp, l_eff, l_new, F_axial;

real k0_1_1;
real k0_2_1, k0_2_2;
real k0_3_1, k0_3_2, k0_3_3;
real k0_4_1, k0_4_2, k0_4_3, k0_4_4;
real k0_5_1, k0_5_2, k0_5_3, k0_5_4, k0_5_5;
real k0_6_1, k0_6_2, k0_6_3, k0_6_4, k0_6_5, k0_6_6;
real k0_7_1, k0_7_2, k0_7_3, k0_7_4, k0_7_5, k0_7_6, k0_7_7;
real k0_8_1, k0_8_2, k0_8_3, k0_8_4, k0_8_5, k0_8_6, k0_8_7, k0_8_8;
real k0_9_1, k0_9_2, k0_9_3, k0_9_4, k0_9_5, k0_9_6, k0_9_7, k0_9_8, k0_9_9;
real k0_10_1,k0_10_2,k0_10_3,k0_10_4,k0_10_5,k0_10_6,k0_10_7,k0_10_8,k0_10_9,k0_10_10;
real k0_11_1,k0_11_2,k0_11_3,k0_11_4,k0_11_5,k0_11_6,k0_11_7,k0_11_8,k0_11_9,k0_11_10,k0_11_11;
real k0_12_1,k0_12_2,k0_12_3,k0_12_4,k0_12_5,k0_12_6,k0_12_7,k0_12_8,k0_12_9,k0_12_10,k0_12_11,k0_12_12;

real k1_1_1;
real k1_2_1, k1_2_2;
real k1_3_1, k1_3_2, k1_3_3;
real k1_4_1, k1_4_2, k1_4_3, k1_4_4;
real k1_5_1, k1_5_2, k1_5_3, k1_5_4, k1_5_5;
real k1_6_1, k1_6_2, k1_6_3, k1_6_4, k1_6_5, k1_6_6;
real k1_7_1, k1_7_2, k1_7_3, k1_7_4, k1_7_5, k1_7_6, k1_7_7;
real k1_8_1, k1_8_2, k1_8_3, k1_8_4, k1_8_5, k1_8_6, k1_8_7, k1_8_8;
real k1_9_1, k1_9_2, k1_9_3, k1_9_4, k1_9_5, k1_9_6, k1_9_7, k1_9_8, k1_9_9;
real k1_10_1,k1_10_2,k1_10_3,k1_10_4,k1_10_5,k1_10_6,k1_10_7,k1_10_8,k1_10_9,k1_10_10;
real k1_11_1,k1_11_2,k1_11_3,k1_11_4,k1_11_5,k1_11_6,k1_11_7,k1_11_8,k1_11_9,k1_11_10,k1_11_11;
real k1_12_1,k1_12_2,k1_12_3,k1_12_4,k1_12_5,k1_12_6,k1_12_7,k1_12_8,k1_12_9,k1_12_10,k1_12_11,k1_12_12;

real k_1_1;
real k_2_1, k_2_2;
real k_3_1, k_3_2, k_3_3;
real k_4_1, k_4_2, k_4_3, k_4_4;
real k_5_1, k_5_2, k_5_3, k_5_4, k_5_5;
real k_6_1, k_6_2, k_6_3, k_6_4, k_6_5, k_6_6;
real k_7_1, k_7_2, k_7_3, k_7_4, k_7_5, k_7_6, k_7_7;
real k_8_1, k_8_2, k_8_3, k_8_4, k_8_5, k_8_6, k_8_7, k_8_8;
real k_9_1, k_9_2, k_9_3, k_9_4, k_9_5, k_9_6, k_9_7, k_9_8, k_9_9;
real k_10_1,k_10_2,k_10_3,k_10_4,k_10_5,k_10_6,k_10_7,k_10_8,k_10_9,k_10_10;
real k_11_1,k_11_2,k_11_3,k_11_4,k_11_5,k_11_6,k_11_7,k_11_8,k_11_9,k_11_10,k_11_11;
real k_12_1,k_12_2,k_12_3,k_12_4,k_12_5,k_12_6,k_12_7,k_12_8,k_12_9,k_12_10,k_12_11,k_12_12;

real m_1_1;
real m_2_1, m_2_2;
real m_3_1, m_3_2, m_3_3;
real m_4_1, m_4_2, m_4_3, m_4_4;
real m_5_1, m_5_2, m_5_3, m_5_4, m_5_5;
real m_6_1, m_6_2, m_6_3, m_6_4, m_6_5, m_6_6;
real m_7_1, m_7_2, m_7_3, m_7_4, m_7_5, m_7_6, m_7_7;
real m_8_1, m_8_2, m_8_3, m_8_4, m_8_5, m_8_6, m_8_7, m_8_8;
real m_9_1, m_9_2, m_9_3, m_9_4, m_9_5, m_9_6, m_9_7, m_9_8, m_9_9;
real m_10_1,m_10_2,m_10_3,m_10_4,m_10_5,m_10_6,m_10_7,m_10_8,m_10_9,m_10_10;
real m_11_1,m_11_2,m_11_3,m_11_4,m_11_5,m_11_6,m_11_7,m_11_8,m_11_9,m_11_10,m_11_11;
real m_12_1,m_12_2,m_12_3,m_12_4,m_12_5,m_12_6,m_12_7,m_12_8,m_12_9,m_12_10,m_12_11,m_12_12;

analog begin

l_corner = l+0.3*w*(1-flag_shear);
```

```
resistance = resistivity*l_corner/w;
area = thickness*w;
mass = density*thickness*w*l_corner;
ea = E*area;  // coef used in [k] matrix

ly = 1/12.0*w*thickness*thickness*thickness;
lz = 1/12.0*thickness*w*w*w;

// torsional moment of inertia
    xt = min(thickness,w);
    xw = max(thickness,w);
    x  = xt/xw;
    sum = 0;
    for (i=1; i<20; i=i+2)
    sum = sum + tanh(i*'M_PI/x/2)/pow(i*1.0,5);
    lt = pow(xt,3)*xw*(1-192/pow('M_PI,5)*x*sum)/3;
    // lt=2.253333e-24;

// shear modulus
G = E/2.0/(1+Poisson_ratio);

scale = 1e6;

// damping coefficients
dampx = visc_air*l_corner*(w+bloat)/air_gap;
dampy = visc_air*w*(l_corner+bloat)/air_gap;
dampz = visc_air*thickness*(l_corner+bloat)/air_gap;

cos_alpha = cos(alpha /180*'M_PI);
cos_beta  = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta  = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// translational displacement in chip frame
chip_xa = Pos(xa[0]);
chip_ya = Pos(xa[1]);
chip_za = Pos(xa[2]);
chip_xb = Pos(xb[0]);
chip_yb = Pos(xb[1]);
chip_zb = Pos(xb[2]);
```

```
// angular displacement in chip frame
chip_phixa = scale*Theta(phia[0]);
chip_phiya = scale*Theta(phia[1]);
chip_phiza = scale*Theta(phia[2]);
chip_phixb = scale*Theta(phib[0]);
chip_phiyb = scale*Theta(phib[1]);
chip_phizb = scale*Theta(phib[2]);

// transform displacements from chip frame into local frame
layout_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
layout_ya = l2*chip_xa + m2*chip_ya + n2*chip_za;
layout_za = l3*chip_xa + m3*chip_ya + n3*chip_za;
layout_xb = l1*chip_xb + m1*chip_yb + n1*chip_zb;
layout_yb = l2*chip_xb + m2*chip_yb + n2*chip_zb;
layout_zb = l3*chip_xb + m3*chip_yb + n3*chip_zb;

layout_phixa = l1*chip_phixa + m1*chip_phiya + n1*chip_phiza;
layout_phiya = l2*chip_phixa + m2*chip_phiya + n2*chip_phiza;
layout_phiza = l3*chip_phixa + m3*chip_phiya + n3*chip_phiza;
layout_phixb = l1*chip_phixb + m1*chip_phiyb + n1*chip_phizb;
layout_phiyb = l2*chip_phixb + m2*chip_phiyb + n2*chip_phizb;
layout_phizb = l3*chip_phixb + m3*chip_phiyb + n3*chip_phizb;

// translated local frame
l_xa = layout_xa - layout_xa;
l_ya = layout_ya - layout_ya;
l_za = layout_za - layout_za;
l_xb = layout_xb - layout_xa;
l_yb = layout_yb - layout_ya;
l_zb = layout_zb - layout_za;

l_phixa = layout_phixa;
l_phiya = layout_phiya;
l_phiza = layout_phiza;
l_phixb = layout_phixb;
l_phiyb = layout_phiyb;
l_phizb = layout_phizb;

// dynamic rotation matrix, from translated local frame to displaced frame

rphix = (l_phixa+l_phixb)/2.0;
rphiy = (l_phiya+l_phiyb)/2.0;
rphiz = (l_phiza+l_phizb)/2.0;
rphi  = sqrt(pow(rphix,2)+pow(rphiy,2)+pow(rphiz,2))+1e-36;

rx = rphix/rphi;
ry = rphiy/rphi;
rz = rphiz/rphi;
rc = cos(rphi);
rs = sin(rphi);
rt = 1 - cos(rphi);

new_l1 = rt*rx*rx + rc;
new_m1 = rt*rx*ry + rs*rz;
new_n1 = rt*rx*rz - rs*ry;
new_l2 = rt*rx*ry - rs*rz;
new_m2 = rt*ry*ry + rc;
new_n2 = rt*ry*rz + rs*rx;
new_l3 = rt*rx*rz + rs*ry;
new_m3 = rt*ry*rz - rs*rx;
new_n3 = rt*rz*rz + rc;

new_inv_l1 = rt*rx*rx + rc;
new_inv_l2 = rt*rx*ry + rs*rz;
new_inv_l3 = rt*rx*rz - rs*ry;
```

```
new_inv_m1 = rt*rx*ry - rs*rz;
new_inv_m2 = rt*ry*ry + rc;
new_inv_m3 = rt*ry*rz + rs*rx;
new_inv_n1 = rt*rx*rz + rs*ry;
new_inv_n2 = rt*ry*rz - rs*rx;
new_inv_n3 = rt*rz*rz + rc;

// displacements in the displaced frame
new_l_xa = 0;
new_l_ya = 0;
new_l_za = 0;
new_l_xb = new_l1*(l_xb+l_corner) + new_m1*l_yb + new_n1*l_zb - l_corner;
new_l_yb = new_l2*(l_xb+l_corner) + new_m2*l_yb + new_n2*l_zb;
new_l_zb = new_l3*(l_xb+l_corner) + new_m3*l_yb + new_n3*l_zb;

new_l_phixa = new_l1*l_phixa + new_m1*l_phiya + new_n1*l_phiza - rphix;
new_l_phiya = new_l2*l_phixa + new_m2*l_phiya + new_n2*l_phiza - rphiy;
new_l_phiza = new_l3*l_phixa + new_m3*l_phiya + new_n3*l_phiza - rphiz;
new_l_phixb = new_l1*l_phixb + new_m1*l_phiyb + new_n1*l_phizb - rphix;
new_l_phiyb = new_l2*l_phixb + new_m2*l_phiyb + new_n2*l_phizb - rphiy;
new_l_phizb = new_l3*l_phixb + new_m3*l_phiyb + new_n3*l_phizb - rphiz;

// calculate effective beam length
lm = 0;
lp = l_corner+new_l_xb-new_l_xa;
l_new=lp-lm;

l_eff = (lp*(15*pow(l_new,6)*(2+pow(new_l_phiya,2)+pow(new_l_phiza,2))-
30*pow(l_new,5)*lp*(2*pow(new_l_phiya,2)+new_l_phiya*new_l_phiyb+new_l_phiza*(2*new_l_phiza+new_l_phizb))+10*pow(l_new,
4)*lp*(lp*(11*pow(new_l_phiya,2)+11*new_l_phiya*new_l_phiyb+2*pow(new_l_phiyb,2)+11*pow(new_l_phiza,2)+11*new_l_phiza*n
ew_l_phizb+2*pow(new_l_phizb,2))-9*(new_l_phiza*(new_l_ya-new_l_yb)+new_l_phiya*(new_l_za-new_l_zb)))
    -
15*pow(l_new,3)*pow(lp,2)*(3*lp*(2*pow(new_l_phiya,2)+3*new_l_phiya*new_l_phiyb+pow(new_l_phiyb,2)+2*pow(new_l_phiza,2)+
3*new_l_phiza*new_l_phizb+pow(new_l_phizb,2))-4*(5*new_l_phiza*(new_l_ya-new_l_yb)+2*new_l_phizb*(new_l_ya-
new_l_yb)+(5*new_l_phiya+2*new_l_phiyb)*(new_l_za-new_l_zb)))
    +54*l_new*pow(lp,3)*(2*lp*(new_l_phiza*(new_l_ya-new_l_yb)+new_l_phizb*(new_l_ya-
new_l_yb)+(new_l_phiya+new_l_phiyb)*(new_l_za-new_l_zb))-5*(pow(new_l_ya,2)-
2*new_l_ya*new_l_yb+pow(new_l_yb,2)+pow(new_l_za-new_l_zb,2)))

+9*pow(l_new,2)*pow(lp,2)*(3*pow(lp,2)*(pow(new_l_phiya,2)+2*new_l_phiya*new_l_phiyb+pow(new_l_phiyb,2)+pow(new_l_phiza+
new_l_phizb,2))-5*lp*(7*new_l_phiza*(new_l_ya-new_l_yb)+5*new_l_phizb*(new_l_ya-
new_l_yb)+(7*new_l_phiya+5*new_l_phiyb)*(new_l_za-new_l_zb))+20*(pow(new_l_ya,2)-
2*new_l_ya*new_l_yb+pow(new_l_yb,2)+pow(new_l_za-new_l_zb,2)))
    +108*pow(lp,4)*(pow(new_l_ya,2)-2*new_l_ya*new_l_yb+pow(new_l_yb,2)+pow(new_l_za-new_l_zb,2))))/(30*pow(l_new,6));

// calculate axial force
F_axial = ea*(l_eff-l_corner)/l_corner;

// linear stiffness matrix [k0], with shear deformation

// variables for shear deformation
Asy = 2.0/3.0*area; //effective shear area
Sy  = 1.0*12.0*E*Iz/G/pow(l_corner,2)/Asy * flag_shear; //shear deformation parameter
Srz = 1.0*pow(Iz/area, 0.5) * flag_shear; //radius of gyration about z-axis

Asz = 2.0/3.0*area; //effective shear area
Sz  = 1.0*12.0*E*Iy/G/pow(l_corner,2)/Asz * flag_shear; //shear deformation parameter
Sry = 1.0*pow(Iy/area, 0.5) * flag_shear; //radius of gyration about z-axis

k0_1_1 = ea/l_corner;
k0_2_1 = 0;
k0_3_1 = 0;
k0_4_1 = 0;
k0_5_1 = 0;
k0_6_1 = 0;
```

```
k0_7_1 = -ea/l_corner;
k0_8_1 = 0;
k0_9_1 = 0;
k0_10_1 = 0;
k0_11_1 = 0;
k0_12_1 = 0;

k0_2_2 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k0_3_2 = 0;
k0_4_2 = 0;
k0_5_2 = 0;
k0_6_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k0_7_2 = 0;
k0_8_2 = -12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k0_9_2 = 0;
k0_10_2 = 0;
k0_11_2 = 0;
k0_12_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);

k0_3_3 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k0_4_3 = 0;
k0_5_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k0_6_3 = 0;
k0_7_3 = 0;
k0_8_3 = 0;
k0_9_3 = -12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k0_10_3 = 0;
k0_11_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k0_12_3 = 0;

k0_4_4 = G*It/l_corner;
k0_5_4 = 0;
k0_6_4 = 0;
k0_7_4 = 0;
k0_8_4 = 0;
k0_9_4 = 0;
k0_10_4 = -G*It/l_corner;
k0_11_4 = 0;
k0_12_4 = 0;

k0_5_5 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k0_6_5 = 0;
k0_7_5 = 0;
k0_8_5 = 0;
k0_9_5 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k0_10_5 = 0;
k0_11_5 = (2.0-Sz)*E*Iy/l_corner/(1+Sz);
k0_12_5 = 0;

k0_6_6 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);
k0_7_6 = 0;
k0_8_6 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k0_9_6 = 0;
k0_10_6 = 0;
k0_11_6 = 0;
k0_12_6 = (2.0-Sy)*E*Iz/l_corner/(1+Sy);

k0_7_7 = ea/l_corner;
k0_8_7 = 0;
k0_9_7 = 0;
k0_10_7 = 0;
k0_11_7 = 0;
k0_12_7 = 0;

k0_8_8 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
```

```
k0_9_8 = 0;
k0_10_8 = 0;
k0_11_8 = 0;
k0_12_8 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);

k0_9_9 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k0_10_9 = 0;
k0_11_9 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k0_12_9 = 0;

k0_10_10 = G*It/l_corner;
k0_11_10 = 0;
k0_12_10 = 0;

k0_11_11 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k0_12_11 = 0;

k0_12_12 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);

// geometric stiffness matrix [k1], with shear deformations

k1_1_1 = 0;
k1_2_1 = 0;
k1_3_1 = 0;
k1_4_1 = 0;
k1_5_1 = 0;
k1_6_1 = 0;
k1_7_1 = 0;
k1_8_1 = 0;
k1_9_1 = 0;
k1_10_1 = 0;
k1_11_1 = 0;
k1_12_1 = 0;

k1_2_2 = F_axial*2.0*(6+10*Sy+5*pow(Sy,2))/(10*l_corner*pow(1+Sy,2));
k1_3_2 = 0;
k1_4_2 = 0;
k1_5_2 = 0;
k1_6_2 = F_axial*l_corner/(10*l_corner*pow(1+Sy,2));
k1_7_2 = 0;
k1_8_2 = -F_axial*2.0*(6+10*Sy+5*pow(Sy,2))/(10*l_corner*pow(1+Sy,2));
k1_9_2 = 0;
k1_10_2 = 0;
k1_11_2 = 0;
k1_12_2 = F_axial*l_corner/(10*l_corner*pow(1+Sy,2));

k1_3_3 = F_axial*2.0*(6+10*Sz+5*pow(Sz,2))/(10*l_corner*pow(1+Sz,2));
k1_4_3 = 0;
k1_5_3 = -F_axial*l_corner/(10*l_corner*pow(1+Sz,2));
k1_6_3 = 0;
k1_7_3 = 0;
k1_8_3 = 0;
k1_9_3 = -F_axial*2.0*(6+10*Sz+5*pow(Sz,2))/(10*l_corner*pow(1+Sz,2));
k1_10_3 = 0;
k1_11_3 = -F_axial*l_corner/(10*l_corner*pow(1+Sz,2));
k1_12_3 = 0;

k1_4_4 = 0;
k1_5_4 = 0;
k1_6_4 = 0;
k1_7_4 = 0;
k1_8_4 = 0;
k1_9_4 = 0;
k1_10_4 = 0;
k1_11_4 = 0;
```

```
k1_12_4 = 0;

k1_5_5 = F_axial*l_corner*(8+10*Sz+5*pow(Sz,2))/(60*pow(1+Sz,2));
k1_6_5 = 0;
k1_7_5 = 0;
k1_8_5 = 0;
k1_9_5 = F_axial*6.0/(60*pow(1+Sz,2));
k1_10_5 = 0;
k1_11_5 = F_axial*l_corner*(-(2+10*Sz+5*pow(Sz,2)))/(60*pow(1+Sz,2));
k1_12_5 = 0;

k1_6_6 = F_axial*l_corner*(8+10*Sy+5*pow(Sy,2))/(60*pow(1+Sy,2));
k1_7_6 = 0;
k1_8_6 = -F_axial*6.0/(60*pow(1+Sy,2));
k1_9_6 = 0;
k1_10_6 = 0;
k1_11_6 = 0;
k1_12_6 = F_axial*l_corner*(-(2+10*Sy+5*pow(Sy,2)))/(60*pow(1+Sy,2));

k1_7_7 = 0;
k1_8_7 = 0;
k1_9_7 = 0;
k1_10_7 = 0;
k1_11_7 = 0;
k1_12_7 = 0;

k1_8_8 = F_axial*2.0*(6+10*Sy+5*pow(Sy,2))/(10*l_corner*pow(1+Sy,2));
k1_9_8 = 0;
k1_10_8 = 0;
k1_11_8 = 0;
k1_12_8 = -F_axial*l_corner/(10*l_corner*pow(1+Sy,2));

k1_9_9 = F_axial*2.0*(6+10*Sz+5*pow(Sz,2))/(10*l_corner*pow(1+Sz,2));
k1_10_9 = 0;
k1_11_9 = F_axial*l_corner/(10*l_corner*pow(1+Sz,2));
k1_12_9 = 0;

k1_10_10 = 0;
k1_11_10 = 0;
k1_12_10 = 0;

k1_11_11 = F_axial*l_corner*(8+10*Sz+5*pow(Sz,2))/(60*pow(1+Sz,2));
k1_12_11 = 0;

k1_12_12 = F_axial*l_corner*(8+10*Sy+5*pow(Sy,2))/(60*pow(1+Sy,2));

// entire nonlinear stiffness matrix [k]=[k0]+[k1]

k_1_1 = k0_1_1 + k1_1_1;
k_2_1 = k0_2_1 + k1_2_1;
k_3_1 = k0_3_1 + k1_3_1;
k_4_1 = k0_4_1 + k1_4_1;
k_5_1 = k0_5_1 + k1_5_1;
k_6_1 = k0_6_1 + k1_6_1;
k_7_1 = k0_7_1 + k1_7_1;
k_8_1 = k0_8_1 + k1_8_1;
k_9_1 = k0_9_1 + k1_9_1;
k_10_1 = k0_10_1 + k1_10_1;
k_11_1 = k0_11_1 + k1_11_1;
k_12_1 = k0_12_1 + k1_12_1;

k_2_2 = k0_2_2 + k1_2_2;
k_3_2 = k0_3_2 + k1_3_2;
k_4_2 = k0_4_2 + k1_4_2;
k_5_2 = k0_5_2 + k1_5_2;
```

```
k_6_2 = k0_6_2 + k1_6_2;
k_7_2 = k0_7_2 + k1_7_2;
k_8_2 = k0_8_2 + k1_8_2;
k_9_2 = k0_9_2 + k1_9_2;
k_10_2 = k0_10_2 + k1_10_2;
k_11_2 = k0_11_2 + k1_11_2;
k_12_2 = k0_12_2 + k1_12_2;

k_3_3 = k0_3_3 + k1_3_3;
k_4_3 = k0_4_3 + k1_4_3;
k_5_3 = k0_5_3 + k1_5_3;
k_6_3 = k0_6_3 + k1_6_3;
k_7_3 = k0_7_3 + k1_7_3;
k_8_3 = k0_8_3 + k1_8_3;
k_9_3 = k0_9_3 + k1_9_3;
k_10_3 = k0_10_3 + k1_10_3;
k_11_3 = k0_11_3 + k1_11_3;
k_12_3 = k0_12_3 + k1_12_3;

k_4_4 = k0_4_4 + k1_4_4;
k_5_4 = k0_5_4 + k1_5_4;
k_6_4 = k0_6_4 + k1_6_4;
k_7_4 = k0_7_4 + k1_7_4;
k_8_4 = k0_8_4 + k1_8_4;
k_9_4 = k0_9_4 + k1_9_4;
k_10_4 = k0_10_4 + k1_10_4;
k_11_4 = k0_11_4 + k1_11_4;
k_12_4 = k0_12_4 + k1_12_4;

k_5_5 = k0_5_5 + k1_5_5;
k_6_5 = k0_6_5 + k1_6_5;
k_7_5 = k0_7_5 + k1_7_5;
k_8_5 = k0_8_5 + k1_8_5;
k_9_5 = k0_9_5 + k1_9_5;
k_10_5 = k0_10_5 + k1_10_5;
k_11_5 = k0_11_5 + k1_11_5;
k_12_5 = k0_12_5 + k1_12_5;

k_6_6 = k0_6_6 + k1_6_6;
k_7_6 = k0_7_6 + k1_7_6;
k_8_6 = k0_8_6 + k1_8_6;
k_9_6 = k0_9_6 + k1_9_6;
k_10_6 = k0_10_6 + k1_10_6;
k_11_6 = k0_11_6 + k1_11_6;
k_12_6 = k0_12_6 + k1_12_6;

k_7_7 = k0_7_7 + k1_7_7;
k_8_7 = k0_8_7 + k1_8_7;
k_9_7 = k0_9_7 + k1_9_7;
k_10_7 = k0_10_7 + k1_10_7;
k_11_7 = k0_11_7 + k1_11_7;
k_12_7 = k0_12_7 + k1_12_7;

k_8_8 = k0_8_8 + k1_8_8;
k_9_8 = k0_9_8 + k1_9_8;
k_10_8 = k0_10_8 + k1_10_8;
k_11_8 = k0_11_8 + k1_11_8;
k_12_8 = k0_12_8 + k1_12_8;

k_9_9 = k0_9_9 + k1_9_9;
k_10_9 = k0_10_9 + k1_10_9;
k_11_9 = k0_11_9 + k1_11_9;
k_12_9 = k0_12_9 + k1_12_9;

k_10_10 = k0_10_10 + k1_10_10;
```

Modeling and Simulation for Design of Suspended MEMS

```
k_11_10 = k0_11_10 + k1_11_10;
k_12_10 = k0_12_10 + k1_12_10;

k_11_11 = k0_11_11 + k1_11_11;
k_12_11 = k0_12_11 + k1_12_11;

k_12_12 = k0_12_12 + k1_12_12;

// [Fk]=[k][x], spring forces and moments in displaced frame

new_Fkxa = -F_axial;
new_Fkxb =  F_axial;

new_Fkya = k_2_1*new_l_xa+k_2_2*new_l_ya+k_3_2*new_l_za
    +k_4_2*new_l_phixa+k_5_2*new_l_phiya+k_6_2*new_l_phiza
    +k_7_2*new_l_xb+k_8_2*new_l_yb+k_9_2*new_l_zb
    +k_10_2*new_l_phixb+k_11_2*new_l_phiyb+k_12_2*new_l_phizb;

new_Fkyb = k_8_1*new_l_xa+k_8_2*new_l_ya+k_8_3*new_l_za
    +k_8_4*new_l_phixa+k_8_5*new_l_phiya+k_8_6*new_l_phiza
    +k_8_7*new_l_xb+k_8_8*new_l_yb+k_9_8*new_l_zb
    +k_10_8*new_l_phixb+k_11_8*new_l_phiyb+k_12_8*new_l_phizb;

new_Fkza = k_3_1*new_l_xa+k_3_2*new_l_ya+k_3_3*new_l_za
    +k_4_3*new_l_phixa+k_5_3*new_l_phiya+k_6_3*new_l_phiza
    +k_7_3*new_l_xb+k_8_3*new_l_yb+k_9_3*new_l_zb
    +k_10_3*new_l_phixb+k_11_3*new_l_phiyb+k_12_3*new_l_phizb;

new_Fkzb = k_9_1*new_l_xa+k_9_2*new_l_ya+k_9_3*new_l_za
    +k_9_4*new_l_phixa+k_9_5*new_l_phiya+k_9_6*new_l_phiza
    +k_9_7*new_l_xb+k_9_8*new_l_yb+k_9_9*new_l_zb
    +k_10_9*new_l_phixb+k_11_9*new_l_phiyb+k_12_9*new_l_phizb;

new_Mkxa = k_4_1*new_l_xa+k_4_2*new_l_ya+k_4_3*new_l_za
    +k_4_4*new_l_phixa+k_5_4*new_l_phiya+k_6_4*new_l_phiza
    +k_7_4*new_l_xb+k_8_4*new_l_yb+k_9_4*new_l_zb
    +k_10_4*new_l_phixb+k_11_4*new_l_phiyb+k_12_4*new_l_phizb;

new_Mkxb = k_10_1*new_l_xa+k_10_2*new_l_ya+k_10_3*new_l_za
    +k_10_4*new_l_phixa+k_10_5*new_l_phiya+k_10_6*new_l_phiza
    +k_10_7*new_l_xb+k_10_8*new_l_yb+k_10_9*new_l_zb
    +k_10_10*new_l_phixb+k_11_10*new_l_phiyb+k_12_10*new_l_phizb;

new_Mkya = k_5_1*new_l_xa+k_5_2*new_l_ya+k_5_3*new_l_za
    +k_5_4*new_l_phixa+k_5_5*new_l_phiya+k_6_5*new_l_phiza
    +k_7_5*new_l_xb+k_8_5*new_l_yb+k_9_5*new_l_zb
    +k_10_5*new_l_phixb+k_11_5*new_l_phiyb+k_12_5*new_l_phizb;

new_Mkyb = k_11_1*new_l_xa+k_11_2*new_l_ya+k_11_3*new_l_za
    +k_11_4*new_l_phixa+k_11_5*new_l_phiya+k_11_6*new_l_phiza
    +k_11_7*new_l_xb+k_11_8*new_l_yb+k_11_9*new_l_zb
    +k_11_10*new_l_phixb+k_11_11*new_l_phiyb+k_12_11*new_l_phizb;

new_Mkza = k_6_1*new_l_xa+k_6_2*new_l_ya+k_6_3*new_l_za
    +k_6_4*new_l_phixa+k_6_5*new_l_phiya+k_6_6*new_l_phiza
    +k_7_6*new_l_xb+k_8_6*new_l_yb+k_9_6*new_l_zb
    +k_10_6*new_l_phixb+k_11_6*new_l_phiyb+k_12_6*new_l_phizb;

new_Mkzb = k_12_1*new_l_xa+k_12_2*new_l_ya+k_12_3*new_l_za
    +k_12_4*new_l_phixa+k_12_5*new_l_phiya+k_12_6*new_l_phiza
    +k_12_7*new_l_xb+k_12_8*new_l_yb+k_12_9*new_l_zb
    +k_12_10*new_l_phixb+k_12_11*new_l_phiyb+k_12_12*new_l_phizb;


// mass matrix [m], with shear effects
```

```
m0y = density*thickness*w*l_corner/pow(1+Sy,2);
m0z = density*thickness*w*l_corner/pow(1+Sz,2);

m_1_1 = 1.0/3.0*density*thickness*w*l_corner;
m_2_1 = 0;
m_3_1 = 0;
m_4_1 = 0;
m_5_1 = 0;
m_6_1 = 0;
m_7_1 = 1.0/6.0*density*thickness*w*l_corner;
m_8_1 = 0;
m_9_1 = 0;
m_10_1 = 0;
m_11_1 = 0;
m_12_1 = 0;

m_2_2 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
m_3_2 = 0;
m_4_2 = 0;
m_5_2 = 0;
m_6_2 = m0y*l_corner*((11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));
m_7_2 = 0;
m_8_2 = m0y*((9.0/70.0+3.0/10.0*Sy+1.0/6.0*Sy*Sy)-pow(Srz/l_corner,2)*6.0/5.0);
m_9_2 = 0;
m_10_2 = 0;
m_11_2 = 0;
m_12_2 = m0y*l_corner*(-(13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));

m_3_3 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
m_4_3 = 0;
m_5_3 = -m0z*l_corner*((11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
m_6_3 = 0;
m_7_3 = 0;
m_8_3 = 0;
m_9_3 = m0z*((9.0/70.0+3.0/10.0*Sz+1.0/6.0*Sz*Sz)-pow(Sry/l_corner,2)*6.0/5.0);
m_10_3 = 0;
m_11_3 = -m0z*l_corner*(-(13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
m_12_3 = 0;

m_4_4 = (Iy+Iz)/3.0*density*l_corner;
m_5_4 = 0;
m_6_4 = 0;
m_7_4 = 0;
m_8_4 = 0;
m_9_4 = 0;
m_10_4 = (Iy+Iz)/6.0*density*l_corner;
m_11_4 = 0;
m_12_4 = 0;

m_5_5 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/
3.0*Sz*Sz));
m_6_5 = 0;
m_7_5 = 0;
m_8_5 = 0;
m_9_5 = -m0z*l_corner*((13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
m_10_5 = 0;
m_11_5 = m0z*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/30.0-1.0/6.0*Sz+1.0/
6.0*Sz*Sz));
m_12_5 = 0;

m_6_6 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/
3.0*Sy*Sy));
m_7_6 = 0;
m_8_6 = m0y*l_corner*((13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));
```

```
    m_9_6 = 0;
    m_10_6 = 0;
    m_11_6 = 0;
    m_12_6 = m0y*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/30.0-1.0/6.0*Sy+1.0/
6.0*Sy*Sy));

    m_7_7 = 1.0/3.0*density*thickness*w*l_corner;
    m_8_7 = 0;
    m_9_7 = 0;
    m_10_7 = 0;
    m_11_7 = 0;
    m_12_7 = 0;

    m_8_8 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
    m_9_8 = 0;
    m_10_8 = 0;
    m_11_8 = 0;
    m_12_8 = m0y*l_corner*(-(11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));

    m_9_9 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
    m_10_9 = 0;
    m_11_9 = -m0z*l_corner*(-(11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
    m_12_9 = 0;

    m_10_10 = (Iy+Iz)/3.0*density*l_corner;
    m_11_10 = 0;
    m_12_10 = 0;

    m_11_11 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/
3.0*Sz*Sz));
    m_12_11 = 0;

    m_12_12 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/
3.0*Sy*Sy));

    // inertial and damping forces/moments calculated in the local frame

    Pos(Vxa) <+ ddt(layout_xa);
    Pos(Vxb) <+ ddt(layout_xb);
    Pos(Vya) <+ ddt(layout_ya);
    Pos(Vyb) <+ ddt(layout_yb);
    Pos(Vza) <+ ddt(layout_za);
    Pos(Vzb) <+ ddt(layout_zb);

    Omega(Vphixa) <+ ddt(layout_phixa);
    Omega(Vphixb) <+ ddt(layout_phixb);
    Omega(Vphiya) <+ ddt(layout_phiya);
    Omega(Vphiyb) <+ ddt(layout_phiyb);
    Omega(Vphiza) <+ ddt(layout_phiza);
    Omega(Vphizb) <+ ddt(layout_phizb);

    // [Fm]=[m][a], inertial forces/moments in local frame

    Pos(new_Fmxa) <+ m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
    Pos(new_Fmxb) <+ m_7_1*ddt(Pos(Vxa)) + m_7_7*ddt(Pos(Vxb));

    Pos(new_Fmya) <+ m_6_2*ddt(Omega(Vphiza)) + m_12_2*ddt(Omega(Vphizb))
        + m_2_2*ddt(Pos(Vya)) + m_8_2*ddt(Pos(Vyb));
    Pos(new_Fmyb) <+ m_8_6*ddt(Omega(Vphiza)) + m_12_8*ddt(Omega(Vphizb))
        + m_8_2*ddt(Pos(Vya)) + m_8_8*ddt(Pos(Vyb));

    Pos(new_Fmza) <+ m_5_3*ddt(Omega(Vphiya)) + m_11_3*ddt(Omega(Vphiyb))
        + m_3_3*ddt(Pos(Vza)) + m_9_3*ddt(Pos(Vzb));
    Pos(new_Fmzb) <+ m_9_5*ddt(Omega(Vphiya)) + m_11_9*ddt(Omega(Vphiyb))
        + m_9_3*ddt(Pos(Vza)) + m_9_9*ddt(Pos(Vzb));
```

```
Theta(new_Mmxa) <+ m_4_4*ddt(Omega(Vphixa)) + m_10_4*ddt(Omega(Vphixb));
Theta(new_Mmxb) <+ m_10_4*ddt(Omega(Vphixa)) + m_10_10*ddt(Omega(Vphixb));

Theta(new_Mmya) <+ m_5_5*ddt(Omega(Vphiya)) + m_11_5*ddt(Omega(Vphiyb))
+ m_5_3*ddt(Pos(Vza))+ m_9_5*ddt(Pos(Vzb));
Theta(new_Mmyb) <+ m_11_5*ddt(Omega(Vphiya)) + m_11_11*ddt(Omega(Vphiyb))
+ m_11_3*ddt(Pos(Vza)) + m_11_9*ddt(Pos(Vzb));

Theta(new_Mmza) <+ m_6_6*ddt(Omega(Vphiza)) + m_12_6*ddt(Omega(Vphizb))
+ m_6_2*ddt(Pos(Vya))+ m_8_6*ddt(Pos(Vyb));
Theta(new_Mmzb) <+ m_12_6*ddt(Omega(Vphiza)) + m_12_12*ddt(Omega(Vphizb))
+ m_12_2*ddt(Pos(Vya)) + m_12_8*ddt(Pos(Vyb));

// [Fb]=[B][v], damping forces/moments in local frame
// using damping matrix with shear effect

Pos(new_Fbxa) <+ dampx/mass*(m_1_1*ddt(layout_xa) + m_7_1*ddt(layout_xb));
Pos(new_Fbxb) <+ dampx/mass*(m_7_1*ddt(layout_xa) + m_7_7*ddt(layout_xb));

Pos(new_Fbya) <+ dampy/mass*(m_6_2*ddt(layout_phiza) + m_12_2*ddt(layout_phizb)
        + m_2_2*ddt(layout_ya) + m_8_2*ddt(layout_yb));
Pos(new_Fbyb) <+ dampy/mass*(m_8_6*ddt(layout_phiza) + m_12_8*ddt(layout_phizb)
        + m_8_2*ddt(layout_ya) + m_8_8*ddt(layout_yb));

Pos(new_Fbza) <+ dampz/mass*(m_5_3*ddt(layout_phiya) + m_11_3*ddt(layout_phiyb)
        + m_3_3*ddt(layout_za) + m_9_3*ddt(layout_zb));
Pos(new_Fbzb) <+ dampz/mass*(m_9_5*ddt(layout_phiya) + m_11_9*ddt(layout_phiyb)
        + m_9_3*ddt(layout_za) + m_9_9*ddt(layout_zb));

Theta(new_Mbxa) <+ dampx/mass*(m_4_4*ddt(layout_phixa) + m_10_4*ddt(layout_phixb));
Theta(new_Mbxb) <+ dampx/mass*(m_10_4*ddt(layout_phixa) + m_10_10*ddt(layout_phixb));

Theta(new_Mbya) <+ dampz/mass*(m_5_5*ddt(layout_phiya) + m_11_5*ddt(layout_phiyb)
+ m_5_3*ddt(layout_za)+ m_9_5*ddt(layout_zb));
Theta(new_Mbyb) <+ dampz/mass*(m_11_5*ddt(layout_phiya) + m_11_11*ddt(layout_phiyb)
+ m_11_3*ddt(layout_za) + m_11_9*ddt(layout_zb));

Theta(new_Mbza) <+ dampy/mass*(m_6_6*ddt(layout_phiza) + m_12_6*ddt(layout_phizb)
+ m_6_2*ddt(layout_ya)+ m_8_6*ddt(layout_yb));
Theta(new_Mbzb) <+ dampy/mass*(m_12_6*ddt(layout_phiza) + m_12_12*ddt(layout_phizb)
+ m_12_2*ddt(layout_ya) + m_12_8*ddt(layout_yb));

// spring forces/moments transformed from displaced frame back to the local frame
// sum up spring, inertial and damping forces/moments in local frame

layout_Fxb = Pos(new_Fmxb)+Pos(new_Fbxb) + new_inv_l1*new_Fkxb + new_inv_m1*new_Fkyb +new_inv_n1*new_Fkzb ;
layout_Fyb = Pos(new_Fmyb)+Pos(new_Fbyb) + new_inv_l2*new_Fkxb + new_inv_m2*new_Fkyb +new_inv_n2*new_Fkzb ;
layout_Fzb = Pos(new_Fmzb)+Pos(new_Fbzb) + new_inv_l3*new_Fkxb + new_inv_m3*new_Fkyb +new_inv_n3*new_Fkzb ;

layout_Fxa = Pos(new_Fmxa)+Pos(new_Fbxa) + new_inv_l1*new_Fkxa + new_inv_m1*new_Fkya +new_inv_n1*new_Fkza ;
layout_Fya = Pos(new_Fmya)+Pos(new_Fbya) + new_inv_l2*new_Fkxa + new_inv_m2*new_Fkya +new_inv_n2*new_Fkza ;
layout_Fza = Pos(new_Fmza)+Pos(new_Fbza) + new_inv_l3*new_Fkxa + new_inv_m3*new_Fkya +new_inv_n3*new_Fkza ;

layout_Mxb = Theta(new_Mmxb)+Theta(new_Mbxb) + new_inv_l1*new_Mkxb + new_inv_m1*new_Mkyb
+new_inv_n1*new_Mkzb ;
layout_Myb = Theta(new_Mmyb)+Theta(new_Mbyb) + new_inv_l2*new_Mkxb + new_inv_m2*new_Mkyb
+new_inv_n2*new_Mkzb ;
layout_Mzb = Theta(new_Mmzb)+Theta(new_Mbzb) + new_inv_l3*new_Mkxb + new_inv_m3*new_Mkyb
+new_inv_n3*new_Mkzb ;

layout_Mxa = Theta(new_Mmxa)+Theta(new_Mbxa) + new_inv_l1*new_Mkxa + new_inv_m1*new_Mkya
+new_inv_n1*new_Mkza ;
layout_Mya = Theta(new_Mmya)+Theta(new_Mbya) + new_inv_l2*new_Mkxa + new_inv_m2*new_Mkya
+new_inv_n2*new_Mkza ;
```

Modeling and Simulation for Design of Suspended MEMS   251

```
        layout_Mza = Theta(new_Mmza)+Theta(new_Mbza) + new_inv_l3*new_Mkxa + new_inv_m3*new_Mkya
+new_inv_n3*new_Mkza ;

        // forces/moments transformed from local frame back to chip frame

        chip_Fxb = inv_l1*layout_Fxb+inv_m1*layout_Fyb+inv_n1*layout_Fzb;
        chip_Fyb = inv_l2*layout_Fxb+inv_m2*layout_Fyb+inv_n2*layout_Fzb;
        chip_Fzb = inv_l3*layout_Fxb+inv_m3*layout_Fyb+inv_n3*layout_Fzb;

        chip_Fxa = inv_l1*layout_Fxa+inv_m1*layout_Fya+inv_n1*layout_Fza;
        chip_Fya = inv_l2*layout_Fxa+inv_m2*layout_Fya+inv_n2*layout_Fza;
        chip_Fza = inv_l3*layout_Fxa+inv_m3*layout_Fya+inv_n3*layout_Fza;

        chip_Mxb = inv_l1*layout_Mxb+inv_m1*layout_Myb+inv_n1*layout_Mzb;
        chip_Myb = inv_l2*layout_Mxb+inv_m2*layout_Myb+inv_n2*layout_Mzb;
        chip_Mzb = inv_l3*layout_Mxb+inv_m3*layout_Myb+inv_n3*layout_Mzb;

        chip_Mxa = inv_l1*layout_Mxa+inv_m1*layout_Mya+inv_n1*layout_Mza;
        chip_Mya = inv_l2*layout_Mxa+inv_m2*layout_Mya+inv_n2*layout_Mza;
        chip_Mza = inv_l3*layout_Mxa+inv_m3*layout_Mya+inv_n3*layout_Mza;

        // forces/moments applied to the beam ends
        F(xa[0]) <+ -chip_Fxa;
        F(xa[1]) <+ -chip_Fya;
        F(xa[2]) <+ -chip_Fza;
        F(xb[0]) <+ -chip_Fxb;
        F(xb[1]) <+ -chip_Fyb;
        F(xb[2]) <+ -chip_Fzb;

        Tau(phia[2]) <+ -chip_Mza;
        Tau(phib[2]) <+ -chip_Mzb;
        Tau(phia[1]) <+ -chip_Mya;
        Tau(phib[1]) <+ -chip_Myb;
        Tau(phia[0]) <+ -chip_Mxa;
        Tau(phib[0]) <+ -chip_Mxb;

        //Calculate beam's electrical relation, I=V/R
        I(vb, va) <+ V(vb, va)/resistance;

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

A.5 Linear Beam Model for CMOS-MEMS Process

In this section, the linear beam model for CMOS-MEMS process is presented to show the language implementation of models for composite-layer structures. This model is based on the HP 0.5um CMOS process, which provides three metal layers and one polysilicon layer. As described in Chapter 4.3, submodules are introduced for the modeling of CMOS-MEMS beams. The parent module, the mechanical submodule and the electrical submodule are given below, to show how the multiple types of composite-layer structures are modeled and maintained through one common set of modules and how the parameters are transferred between these modules.

Verilog-A code for the parent module is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
'include "../../process.h"
'include "../../design.h"

module beam3D_lin(phia, phib, xa, xb, va, vb, ta, tb);

inout [0:2] phia;
rotational [0:2] phia;
inout [0:2] phib;
rotational [0:2] phib;
inout [0:2] xa;
kinematic [0:2] xa;
inout [0:2] xb;
kinematic [0:2] xb;
inout [0:3] va;
electrical [0:3] va;
inout [0:3] vb;
electrical [0:3] vb;
inout ta;
thermal ta;
inout tb;
thermal tb;

parameter integer flag_shear = 0;

parameter real l = 'default_beam_l;
parameter real w = 'default_beam_w;

parameter real alpha = 0;
parameter real beta = 0;
parameter real gamma = 0;

// flag parameters for conductor layers
parameter integer flag_m3 = 1;
parameter integer flag_m2 = 1;
parameter integer flag_m1 = 1;
parameter integer flag_poly = 1;

parameter real Xc = 0;
```

```
parameter real Yc = 0;

parameter real cutin_poly = 'default_cutin_poly;
parameter real cutin_m1 = 'default_cutin_m1;
parameter real cutin_m2 = 'default_cutin_m2;

parameter real overetch = 'default_overetch;

parameter real resistivity_m3 = 'default_resistivity_m3;
parameter real resistivity_m2 = 'default_resistivity_m2;
parameter real resistivity_m1 = 'default_resistivity_m1;
parameter real resistivity_poly = 'default_resistivity_poly;

parameter real den_metal='default_den_metal;
parameter real den_poly='default_den_poly;
parameter real den_oxide='default_den_oxide;

// thickness of metal layers
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;

// thickness of oxide layers
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 real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real Poisson_ratio = 'default_Poisson_ratio;

// Young's modulus, thickness and mass density calculated based on layer composition

parameter real E = flag_m3*flag_m2*flag_m1*flag_poly*'default_E_cmos_1111
+flag_m3*flag_m2*flag_m1*(1-flag_poly)*'default_E_cmos_1110
+flag_m3*flag_m2*(1-flag_m1)*flag_poly*'default_E_cmos_1101
+flag_m3*flag_m2*(1-flag_m1)*(1-flag_poly)*'default_E_cmos_1100
+flag_m3*(1-flag_m2)*flag_m1*flag_poly*'default_E_cmos_1011
+flag_m3*(1-flag_m2)*flag_m1*(1-flag_poly)*'default_E_cmos_1010
+flag_m3*(1-flag_m2)*(1-flag_m1)*flag_poly*'default_E_cmos_1001
+flag_m3*(1-flag_m2)*(1-flag_m1)*(1-flag_poly)*'default_E_cmos_1000
+(1-flag_m3)*flag_m2*flag_m1*flag_poly*'default_E_cmos_0111
+(1-flag_m3)*flag_m2*flag_m1*(1-flag_poly)*'default_E_cmos_0110
+(1-flag_m3)*flag_m2*(1-flag_m1)*flag_poly*'default_E_cmos_0101
+(1-flag_m3)*flag_m2*(1-flag_m1)*(1-flag_poly)*'default_E_cmos_0100
+(1-flag_m3)*(1-flag_m2)*flag_m1*flag_poly*'default_E_cmos_0011
+(1-flag_m3)*(1-flag_m2)*flag_m1*(1-flag_poly)*'default_E_cmos_0010;

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 density = ('default_den_metal*
        ( flag_m3*t_m3*(w-overetch)
                 + flag_m2*t_m2*((w-overetch)-2.0*cutin_m2)
                 + flag_m1*t_m1*((w-overetch)-2.0*cutin_m1))
          + 'default_den_poly*flag_poly*t_poly
        *((w-overetch)-2.0*cutin_poly)
             + 'default_den_oxide*( flag_m2*t_m2*2.0*cutin_m2
                     + flag_m1*t_m1*2.0*cutin_m1
                     + flag_poly*t_poly*2.0*cutin_poly )
          + 'default_den_oxide*(thickness-(flag_m3*t_m3 + flag_m2*t_m2
          + flag_m1*t_m1 + flag_poly*t_poly)) *(w-overetch) )
        /((w-overetch)*thickness);

        // call mechanical beam submodule, map pins and pass parameters
        beam3D_mech_lin # (.l(l+0.3*w*(1-flag_shear)), .w(w-overetch), .thickness(thickness), .E(E), .density(density), .alpha(alpha),
.beta(beta), .gamma(gamma), .flag_shear(flag_shear),.air_gap(air_gap), .visc_air(visc_air), .Poisson_ratio(Poisson_ratio), .Xc(Xc),
.Yc(Yc) )
        beam3D_mech_lin_1 (phia[0:2], phib[0:2], xa[0:2], xb[0:2]);

        // call electrical beam submodule, map pins and pass parameters
        beam_elec # (.l(l+0.3*w*(1-flag_shear)), .w(w-overetch),
        .flag_m3(flag_m3), .flag_m2(flag_m2),
        .flag_m1(flag_m1), .flag_poly(flag_poly),
        .resistivity_m3(resistivity_m3), .resistivity_m2(resistivity_m2),
        .resistivity_m1(resistivity_m1), .resistivity_poly(resistivity_poly),
        .t_m3_m2(t_m3_m2), .t_m3_m1_overpoly(t_m3_m1_overpoly),
        .t_m3_m1_overfield(t_m3_m1_overfield), .t_m3_poly(t_m3_poly),
        .t_m3_sub(t_m3_sub), .t_m2_m1_overpoly(t_m2_m1_overpoly),
        .t_m2_m1_overfield(t_m2_m1_overfield), .t_m2_poly(t_m2_poly),
        .t_m2_sub(t_m2_sub), .t_m1_poly(t_m1_poly),
        .t_m1_sub(t_m1_sub), .t_poly_sub(t_poly_sub))
        beam_elec_1 (va[3], vb[3], va[2], vb[2], va[1], vb[1], va[0], vb[0]);

        endmodule  // end of parent module
```

Verilog-A code for the mechanical submodule is:

```
        // include header files
        'include "../../constants.h"
        'include "../../discipline.h"

        // only mechanical (kinematic and rotational) pins are defined in mechanical submodule
        // Internal variables and physical equations are the same as those in single-layer beam model

        module beam3D_mech_lin(phia, phib, xa, xb);

        inout [0:2] phia;
        rotational [0:2] phia;
        inout [0:2] phib;
        rotational [0:2] phib;
        inout [0:2] xa;
        kinematic [0:2] xa;
        inout [0:2] xb;
        kinematic [0:2] xb;
```

```
// all parameters are default to be 0
parameter real l = 0;
parameter real w = 0;
parameter real thickness = 0;
parameter real E = 0;
parameter real density = 0;

parameter real alpha = 0;
parameter real beta = 0;
parameter real gamma = 0;

parameter real bloat = 0;
parameter real air_gap = 0;
parameter real visc_air = 0;
parameter real Poisson_ratio = 0;

parameter real Xc = 0;
parameter real Yc = 0;

parameter integer flag_shear = 0;

// definition of internal variables

real Iz, Iy, G, It, ea, xt, xw, x, sum, area, mass;
integer i;

real dampx, dampy, dampz, scale;

// variables for shear deformation
real Asy, Sy, Srz, Asz, Sz, Sry, m0y, m0z;

real cos_alpha, sin_alpha, cos_beta, sin_beta, cos_gamma, sin_gamma;
real l1, m1, n1, l2, m2, n2, l3, m3, n3;
real inv_l1, inv_m1, inv_n1, inv_l2, inv_m2, inv_n2, inv_l3, inv_m3, inv_n3;

real chip_xa, chip_ya, chip_za, chip_xb, chip_yb, chip_zb;
real l_xa, l_ya, l_za, l_xb, l_yb, l_zb;
real chip_phixa, chip_phiya, chip_phiza, chip_phixb, chip_phiyb, chip_phizb;
real l_phixa, l_phiya, l_phiza, l_phixb, l_phiyb, l_phizb;

real l_Fkxb, l_Fkxa, l_Fkyb, l_Fkya, l_Fkzb, l_Fkza;
real l_Mkxa, l_Mkya, l_Mkza, l_Mkxb, l_Mkyb, l_Mkzb;
real l_Fmxb, l_Fmxa, l_Fmyb, l_Fmya, l_Fmzb, l_Fmza;
real l_Mmxa, l_Mmya, l_Mmza, l_Mmxb, l_Mmyb, l_Mmzb;
real l_Fbxb, l_Fbxa, l_Fbyb, l_Fbya, l_Fbzb, l_Fbza;
real l_Mbxa, l_Mbya, l_Mbza, l_Mbxb, l_Mbyb, l_Mbzb;
real l_Fxb, l_Fxa, l_Fyb, l_Fya, l_Fzb, l_Fza;
real l_Mxa, l_Mya, l_Mza, l_Mxb, l_Myb, l_Mzb;
real chip_Fxb, chip_Fxa, chip_Fyb, chip_Fya, chip_Fzb, chip_Fza;
real chip_Mxa, chip_Mya, chip_Mza, chip_Mxb, chip_Myb, chip_Mzb;

kinematic Vxa, Vxb, Vya, Vyb, Vza, Vzb;
rotational_omega  Vphixa, Vphixb, Vphiya, Vphiyb, Vphiza, Vphizb;

real k_1_1;
real k_2_1, k_2_2;
real k_3_1, k_3_2, k_3_3;
real k_4_1, k_4_2, k_4_3, k_4_4;
real k_5_1, k_5_2, k_5_3, k_5_4, k_5_5;
real k_6_1, k_6_2, k_6_3, k_6_4, k_6_5, k_6_6;
real k_7_1, k_7_2, k_7_3, k_7_4, k_7_5, k_7_6, k_7_7;
real k_8_1, k_8_2, k_8_3, k_8_4, k_8_5, k_8_6, k_8_7, k_8_8;
real k_9_1, k_9_2, k_9_3, k_9_4, k_9_5, k_9_6, k_9_7, k_9_8, k_9_9;
real k_10_1,k_10_2,k_10_3,k_10_4,k_10_5,k_10_6,k_10_7,k_10_8,k_10_9,k_10_10;
```

```
real k_11_1,k_11_2,k_11_3,k_11_4,k_11_5,k_11_6,k_11_7,k_11_8,k_11_9,k_11_10,k_11_11;
real k_12_1,k_12_2,k_12_3,k_12_4,k_12_5,k_12_6,k_12_7,k_12_8,k_12_9,k_12_10,k_12_11,k_12_12;

real m_1_1;
real m_2_1, m_2_2;
real m_3_1, m_3_2, m_3_3;
real m_4_1, m_4_2, m_4_3, m_4_4;
real m_5_1, m_5_2, m_5_3, m_5_4, m_5_5;
real m_6_1, m_6_2, m_6_3, m_6_4, m_6_5, m_6_6;
real m_7_1, m_7_2, m_7_3, m_7_4, m_7_5, m_7_6, m_7_7;
real m_8_1, m_8_2, m_8_3, m_8_4, m_8_5, m_8_6, m_8_7, m_8_8;
real m_9_1, m_9_2, m_9_3, m_9_4, m_9_5, m_9_6, m_9_7, m_9_8, m_9_9;
real m_10_1,m_10_2,m_10_3,m_10_4,m_10_5,m_10_6,m_10_7,m_10_8,m_10_9,m_10_10;
real m_11_1,m_11_2,m_11_3,m_11_4,m_11_5,m_11_6,m_11_7,m_11_8,m_11_9,m_11_10,m_11_11;
real m_12_1,m_12_2,m_12_3,m_12_4,m_12_5,m_12_6,m_12_7,m_12_8,m_12_9,m_12_10,m_12_11,m_12_12;


analog begin

area = thickness*w;
mass = density*thickness*w*l;

ea = E*area;  // coef used in [k] matrix
Iy = 1/12.0*w*thickness*thickness*thickness;
Iz = 1/12.0*thickness*w*w*w;

// torsional moment of inertia
     xt = min(thickness,w);
     xw = max(thickness,w);
     x  = xt/xw;
     sum = 0;
     for (i=1; i<20; i=i+2)
     sum = sum + tanh(i*'M_PI/x/2)/pow(i*1.0,5);
     It = pow(xt,3)*xw*(1-192/pow('M_PI,5)*x*sum)/3;
     // It=2.253333e-24;

// shear modulus
G = E/2.0/(1+Poisson_ratio);

scale = 1e6;

dampx = visc_air*l*(w+bloat)/air_gap;
dampy = visc_air*w*(l+bloat)/air_gap;
dampz = visc_air*thickness*(l+bloat)/air_gap;

//layout euler angles
cos_alpha = cos(alpha /180*'M_PI);
cos_beta  = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta  = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
```

```
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// translational displacement in chip frame
chip_xa = Pos(xa[0]);
chip_ya = Pos(xa[1]);
chip_za = Pos(xa[2]);
chip_xb = Pos(xb[0]);
chip_yb = Pos(xb[1]);
chip_zb = Pos(xb[2]);

// angular displacement in chip frame
chip_phixa = scale*Theta(phia[0]);
chip_phiya = scale*Theta(phia[1]);
chip_phiza = scale*Theta(phia[2]);
chip_phixb = scale*Theta(phib[0]);
chip_phiyb = scale*Theta(phib[1]);
chip_phizb = scale*Theta(phib[2]);

// transform displacements from chip frame into local frame
l_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
l_ya = l2*chip_xa + m2*chip_ya + n2*chip_za;
l_za = l3*chip_xa + m3*chip_ya + n3*chip_za;
l_xb = l1*chip_xb + m1*chip_yb + n1*chip_zb;
l_yb = l2*chip_xb + m2*chip_yb + n2*chip_zb;
l_zb = l3*chip_xb + m3*chip_yb + n3*chip_zb;

l_phixa = l1*chip_phixa + m1*chip_phiya + n1*chip_phiza;
l_phiya = l2*chip_phixa + m2*chip_phiya + n2*chip_phiza;
l_phiza = l3*chip_phixa + m3*chip_phiya + n3*chip_phiza;
l_phixb = l1*chip_phixb + m1*chip_phiyb + n1*chip_phizb;
l_phiyb = l2*chip_phixb + m2*chip_phiyb + n2*chip_phizb;
l_phizb = l3*chip_phixb + m3*chip_phiyb + n3*chip_phizb;

// linear stiffness matrix [k], with shear deformation

// variables for shear deformation
Asy = 2.0/3.0*area; //effective shear area
Sy  = 1.0*12.0*E*Iz/G/pow(l,2)/Asy * flag_shear; //shear deformation parameter
Srz = 1.0*pow(Iz/area, 0.5) * flag_shear; //radius of gyration about z-axis

Asz = 2.0/3.0*area; //effective shear area
Sz  = 1.0*12.0*E*Iy/G/pow(l,2)/Asz * flag_shear; //shear deformation parameter
Sry = 1.0*pow(Iy/area, 0.5) * flag_shear; //radius of gyration about z-axis

k_1_1 = ea/l;
k_2_1 = 0;
k_3_1 = 0;
k_4_1 = 0;
k_5_1 = 0;
k_6_1 = 0;
k_7_1 = -ea/l;
k_8_1 = 0;
k_9_1 = 0;
k_10_1 = 0;
k_11_1 = 0;
k_12_1 = 0;
```

```
k_2_2 = 12.0*E*Iz/pow(l,3)/(1+Sy);
k_3_2 = 0;
k_4_2 = 0;
k_5_2 = 0;
k_6_2 = 6.0*E*Iz/pow(l,2)/(1+Sy);
k_7_2 = 0;
k_8_2 = -12.0*E*Iz/pow(l,3)/(1+Sy);
k_9_2 = 0;
k_10_2 = 0;
k_11_2 = 0;
k_12_2 = 6.0*E*Iz/pow(l,2)/(1+Sy);

k_3_3 = 12.0*E*Iy/pow(l,3)/(1+Sz);
k_4_3 = 0;
k_5_3 = -6.0*E*Iy/pow(l,2)/(1+Sz);
k_6_3 = 0;
k_7_3 = 0;
k_8_3 = 0;
k_9_3 = -12.0*E*Iy/pow(l,3)/(1+Sz);
k_10_3 = 0;
k_11_3 = -6.0*E*Iy/pow(l,2)/(1+Sz);
k_12_3 = 0;

k_4_4 = G*It/l;
k_5_4 = 0;
k_6_4 = 0;
k_7_4 = 0;
k_8_4 = 0;
k_9_4 = 0;
k_10_4 = -G*It/l;
k_11_4 = 0;
k_12_4 = 0;

k_5_5 = (4.0+Sz)*E*Iy/l/(1+Sz);
k_6_5 = 0;
k_7_5 = 0;
k_8_5 = 0;
k_9_5 = 6.0*E*Iy/pow(l,2)/(1+Sz);
k_10_5 = 0;
k_11_5 = (2.0-Sz)*E*Iy/l/(1+Sz);
k_12_5 = 0;

k_6_6 = (4.0+Sy)*E*Iz/l/(1+Sy);
k_7_6 = 0;
k_8_6 = -6.0*E*Iz/pow(l,2)/(1+Sy);
k_9_6 = 0;
k_10_6 = 0;
k_11_6 = 0;
k_12_6 = (2.0-Sy)*E*Iz/l/(1+Sy);

k_7_7 = ea/l;
k_8_7 = 0;
k_9_7 = 0;
k_10_7 = 0;
k_11_7 = 0;
k_12_7 = 0;

k_8_8 = 12.0*E*Iz/pow(l,3)/(1+Sy);
k_9_8 = 0;
k_10_8 = 0;
k_11_8 = 0;
k_12_8 = -6.0*E*Iz/pow(l,2)/(1+Sy);

k_9_9 = 12.0*E*Iy/pow(l,3)/(1+Sz);
k_10_9 = 0;
```

```
k_11_9 = 6.0*E*Iy/pow(l,2)/(1+Sz);
k_12_9 = 0;

k_10_10 = G*It/l;
k_11_10 = 0;
k_12_10 = 0;

k_11_11 = (4.0+Sz)*E*Iy/l/(1+Sz);
k_12_11 = 0;

k_12_12 = (4.0+Sy)*E*Iz/l/(1+Sy);

// [Fk]=[k][x], spring forces and moments in local frame

l_Fkxa = k_1_1*l_xa+k_2_1*l_ya+k_3_1*l_za
     +k_4_1*l_phixa+k_5_1*l_phiya+k_6_1*l_phiza
     +k_7_1*l_xb+k_8_1*l_yb+k_9_1*l_zb
     +k_10_1*l_phixb+k_11_1*l_phiyb+k_12_1*l_phizb;

l_Fkxb = k_7_1*l_xa+k_7_2*l_ya+k_7_3*l_za
     +k_7_4*l_phixa+k_7_5*l_phiya+k_7_6*l_phiza
     +k_7_7*l_xb+k_8_7*l_yb+k_9_7*l_zb
     +k_10_7*l_phixb+k_11_7*l_phiyb+k_12_7*l_phizb;

l_Fkya = k_2_1*l_xa+k_2_2*l_ya+k_3_2*l_za
     +k_4_2*l_phixa+k_5_2*l_phiya+k_6_2*l_phiza
     +k_7_2*l_xb+k_8_2*l_yb+k_9_2*l_zb
     +k_10_2*l_phixb+k_11_2*l_phiyb+k_12_2*l_phizb;

l_Fkyb = k_8_1*l_xa+k_8_2*l_ya+k_8_3*l_za
     +k_8_4*l_phixa+k_8_5*l_phiya+k_8_6*l_phiza
     +k_8_7*l_xb+k_8_8*l_yb+k_9_8*l_zb
     +k_10_8*l_phixb+k_11_8*l_phiyb+k_12_8*l_phizb;

l_Fkza = k_3_1*l_xa+k_3_2*l_ya+k_3_3*l_za
     +k_4_3*l_phixa+k_5_3*l_phiya+k_6_3*l_phiza
     +k_7_3*l_xb+k_8_3*l_yb+k_9_3*l_zb
     +k_10_3*l_phixb+k_11_3*l_phiyb+k_12_3*l_phizb;

l_Fkzb = k_9_1*l_xa+k_9_2*l_ya+k_9_3*l_za
     +k_9_4*l_phixa+k_9_5*l_phiya+k_9_6*l_phiza
     +k_9_7*l_xb+k_9_8*l_yb+k_9_9*l_zb
     +k_10_9*l_phixb+k_11_9*l_phiyb+k_12_9*l_phizb;

l_Mkxa = k_4_1*l_xa+k_4_2*l_ya+k_4_3*l_za
     +k_4_4*l_phixa+k_5_4*l_phiya+k_6_4*l_phiza
     +k_7_4*l_xb+k_8_4*l_yb+k_9_4*l_zb
     +k_10_4*l_phixb+k_11_4*l_phiyb+k_12_4*l_phizb;

l_Mkxb = k_10_1*l_xa+k_10_2*l_ya+k_10_3*l_za
     +k_10_4*l_phixa+k_10_5*l_phiya+k_10_6*l_phiza
     +k_10_7*l_xb+k_10_8*l_yb+k_10_9*l_zb
     +k_10_10*l_phixb+k_11_10*l_phiyb+k_12_10*l_phizb;

l_Mkya = k_5_1*l_xa+k_5_2*l_ya+k_5_3*l_za
     +k_5_4*l_phixa+k_5_5*l_phiya+k_6_5*l_phiza
     +k_7_5*l_xb+k_8_5*l_yb+k_9_5*l_zb
     +k_10_5*l_phixb+k_11_5*l_phiyb+k_12_5*l_phizb;

l_Mkyb = k_11_1*l_xa+k_11_2*l_ya+k_11_3*l_za
     +k_11_4*l_phixa+k_11_5*l_phiya+k_11_6*l_phiza
     +k_11_7*l_xb+k_11_8*l_yb+k_11_9*l_zb
     +k_11_10*l_phixb+k_11_11*l_phiyb+k_12_11*l_phizb;

l_Mkza = k_6_1*l_xa+k_6_2*l_ya+k_6_3*l_za
```

Modeling and Simulation for Design of Suspended MEMS

```
                  +k_6_4*l_phixa+k_6_5*l_phiya+k_6_6*l_phiza
                  +k_7_6*l_xb+k_8_6*l_yb+k_9_6*l_zb
                  +k_10_6*l_phixb+k_11_6*l_phiyb+k_12_6*l_phizb;

l_Mkzb = k_12_1*l_xa+k_12_2*l_ya+k_12_3*l_za
                  +k_12_4*l_phixa+k_12_5*l_phiya+k_12_6*l_phiza
                  +k_12_7*l_xb+k_12_8*l_yb+k_12_9*l_zb
                  +k_12_10*l_phixb+k_12_11*l_phiyb+k_12_12*l_phizb;

// mass matrix [m], with shear effects

m0y = density*thickness*w*l/pow(1+Sy,2);
m0z = density*thickness*w*l/pow(1+Sz,2);

m_1_1 = 1.0/3.0*density*thickness*w*l;
m_2_1 = 0;
m_3_1 = 0;
m_4_1 = 0;
m_5_1 = 0;
m_6_1 = 0;
m_7_1 = 1.0/6.0*density*thickness*w*l;
m_8_1 = 0;
m_9_1 = 0;
m_10_1 = 0;
m_11_1 = 0;
m_12_1 = 0;

m_2_2 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l,2)*6.0/5.0);
m_3_2 = 0;
m_4_2 = 0;
m_5_2 = 0;
m_6_2 = m0y*l*((11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l,2)*(1.0/10.0-1.0/2.0*Sy));
m_7_2 = 0;
m_8_2 = m0y*((9.0/70.0+3.0/10.0*Sy+1.0/6.0*Sy*Sy)-pow(Srz/l,2)*6.0/5.0);
m_9_2 = 0;
m_10_2 = 0;
m_11_2 = 0;
m_12_2 = m0y*l*(-(13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l,2)*(1.0/10.0-1.0/2.0*Sy));

m_3_3 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l,2)*6.0/5.0);
m_4_3 = 0;
m_5_3 = -m0z*l*((11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l,2)*(1.0/10.0-1.0/2.0*Sz));
m_6_3 = 0;
m_7_3 = 0;
m_8_3 = 0;
m_9_3 = m0z*((9.0/70.0+3.0/10.0*Sz+1.0/6.0*Sz*Sz)-pow(Sry/l,2)*6.0/5.0);
m_10_3 = 0;
m_11_3 = -m0z*l*(-(13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l,2)*(1.0/10.0-1.0/2.0*Sz));
m_12_3 = 0;

m_4_4 = (Iy+Iz)/3.0*density*l;
m_5_4 = 0;
m_6_4 = 0;
m_7_4 = 0;
m_8_4 = 0;
m_9_4 = 0;
m_10_4 = (Iy+Iz)/6.0*density*l;
m_11_4 = 0;
m_12_4 = 0;

m_5_5 = m0z*l*l*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l,2)*(2.0/15.0+1.0/6.0*Sz+1.0/3.0*Sz*Sz));
m_6_5 = 0;
m_7_5 = 0;
m_8_5 = 0;
m_9_5 = -m0z*l*((13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l,2)*(-1.0/10.0+1.0/2.0*Sz));
```

```
m_10_5 = 0;
m_11_5 = m0z*l*l*(-(1.0/140.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l,2)*(-1.0/30.0-1.0/6.0*Sz+1.0/6.0*Sz*Sz));
m_12_5 = 0;

m_6_6 = m0y*l*l*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l,2)*(2.0/15.0+1.0/6.0*Sy+1.0/3.0*Sy*Sy));
m_7_6 = 0;
m_8_6 = m0y*l*((13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l,2)*(-1.0/10.0+1.0/2.0*Sy));
m_9_6 = 0;
m_10_6 = 0;
m_11_6 = 0;
m_12_6 = m0y*l*l*(-(1.0/140.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l,2)*(-1.0/30.0-1.0/6.0*Sy+1.0/6.0*Sy*Sy));

m_7_7 = 1.0/3.0*density*thickness*w*l;
m_8_7 = 0;
m_9_7 = 0;
m_10_7 = 0;
m_11_7 = 0;
m_12_7 = 0;

m_8_8 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l,2)*6.0/5.0);
m_9_8 = 0;
m_10_8 = 0;
m_11_8 = 0;
m_12_8 = m0y*l*(-(11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l,2)*(-1.0/10.0+1.0/2.0*Sy));

m_9_9 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l,2)*6.0/5.0);
m_10_9 = 0;
m_11_9 = -m0z*l*(-(11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l,2)*(-1.0/10.0+1.0/2.0*Sz));
m_12_9 = 0;

m_10_10 = (Iy+Iz)/3.0*density*l;
m_11_10 = 0;
m_12_10 = 0;

m_11_11 = m0z*l*l*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l,2)*(2.0/15.0+1.0/6.0*Sz+1.0/3.0*Sz*Sz));
m_12_11 = 0;

m_12_12 = m0y*l*l*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l,2)*(2.0/15.0+1.0/6.0*Sy+1.0/3.0*Sy*Sy));

Pos(Vxa) <+ ddt(I_xa);
Pos(Vxb) <+ ddt(I_xb);
Pos(Vya) <+ ddt(I_ya);
Pos(Vyb) <+ ddt(I_yb);
Pos(Vza) <+ ddt(I_za);
Pos(Vzb) <+ ddt(I_zb);

Omega(Vphixa) <+ ddt(I_phixa);
Omega(Vphixb) <+ ddt(I_phixb);
Omega(Vphiya) <+ ddt(I_phiya);
Omega(Vphiyb) <+ ddt(I_phiyb);
Omega(Vphiza) <+ ddt(I_phiza);
Omega(Vphizb) <+ ddt(I_phizb);

// [Fm]=[m][a], inertial forces/moments in local frame

I_Fmxa = m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
I_Fmxb = m_7_1*ddt(Pos(Vxa)) + m_7_7*ddt(Pos(Vxb));

I_Fmya = m_6_2*ddt(Omega(Vphiza)) + m_12_2*ddt(Omega(Vphizb))
    + m_2_2*ddt(Pos(Vya)) + m_8_2*ddt(Pos(Vyb));
I_Fmyb = m_8_6*ddt(Omega(Vphiza)) + m_12_8*ddt(Omega(Vphizb))
    + m_8_2*ddt(Pos(Vya)) + m_8_8*ddt(Pos(Vyb));

I_Fmza = m_5_3*ddt(Omega(Vphiya)) + m_11_3*ddt(Omega(Vphiyb))
    + m_3_3*ddt(Pos(Vza)) + m_9_3*ddt(Pos(Vzb));
```

Modeling and Simulation for Design of Suspended MEMS

```
l_Fmzb = m_9_5*ddt(Omega(Vphiya)) + m_11_9*ddt(Omega(Vphiyb))
      + m_9_3*ddt(Pos(Vza)) + m_9_9*ddt(Pos(Vzb));

l_Mmxa = m_4_4*ddt(Omega(Vphixa)) + m_10_4*ddt(Omega(Vphixb));
l_Mmxb = m_10_4*ddt(Omega(Vphixa)) + m_10_10*ddt(Omega(Vphixb));

l_Mmya = m_5_5*ddt(Omega(Vphiya)) + m_11_5*ddt(Omega(Vphiyb))
+ m_5_3*ddt(Pos(Vza))+ m_9_5*ddt(Pos(Vzb));
l_Mmyb = m_11_5*ddt(Omega(Vphiya)) + m_11_11*ddt(Omega(Vphiyb))
+ m_11_3*ddt(Pos(Vza)) + m_11_9*ddt(Pos(Vzb));

l_Mmza = m_6_6*ddt(Omega(Vphiza)) + m_12_6*ddt(Omega(Vphizb))
+ m_6_2*ddt(Pos(Vya))+ m_8_6*ddt(Pos(Vyb));
l_Mmzb = m_12_6*ddt(Omega(Vphiza)) + m_12_12*ddt(Omega(Vphizb))
+ m_12_2*ddt(Pos(Vya)) + m_12_8*ddt(Pos(Vyb));

// [Fb]=[B][v], inertial forces/moments in local frame
// using damping matrix with shear effect

l_Fbxa = dampx/mass*(m_1_1*ddt(l_xa) + m_7_1*ddt(l_xb));
l_Fbxb = dampx/mass*(m_7_1*ddt(l_xa) + m_7_7*ddt(l_xb));

l_Fbya = dampy/mass*(m_6_2*ddt(l_phiza) + m_12_2*ddt(l_phizb)
      + m_2_2*ddt(l_ya) + m_8_2*ddt(l_yb));
l_Fbyb = dampy/mass*(m_8_6*ddt(l_phiza) + m_12_8*ddt(l_phizb)
      + m_8_2*ddt(l_ya) + m_8_8*ddt(l_yb));

l_Fbza = dampz/mass*(m_5_3*ddt(l_phiya) + m_11_3*ddt(l_phiyb)
      + m_3_3*ddt(l_za) + m_9_3*ddt(l_zb));
l_Fbzb = dampz/mass*(m_9_5*ddt(l_phiya) + m_11_9*ddt(l_phiyb)
      + m_9_3*ddt(l_za) + m_9_9*ddt(l_zb));

l_Mbxa = dampx/mass*(m_4_4*ddt(l_phixa) + m_10_4*ddt(l_phixb));
l_Mbxb = dampx/mass*(m_10_4*ddt(l_phixa) + m_10_10*ddt(l_phixb));

l_Mbya = dampz/mass*(m_5_5*ddt(l_phiya) + m_11_5*ddt(l_phiyb)
+ m_5_3*ddt(l_za)+ m_9_5*ddt(l_zb));
l_Mbyb = dampz/mass*(m_11_5*ddt(l_phiya) + m_11_11*ddt(l_phiyb)
+ m_11_3*ddt(l_za) + m_11_9*ddt(l_zb));

l_Mbza = dampy/mass*(m_6_6*ddt(l_phiza) + m_12_6*ddt(l_phizb)
+ m_6_2*ddt(l_ya)+ m_8_6*ddt(l_yb));
l_Mbzb = dampy/mass*(m_12_6*ddt(l_phiza) + m_12_12*ddt(l_phizb)
+ m_12_2*ddt(l_ya) + m_12_8*ddt(l_yb));

// transform forces/moments from local frame back to the chip frame

chip_Fxb = inv_l1*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m1*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n1*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fyb = inv_l2*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m2*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n2*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fzb = inv_l3*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m3*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n3*(l_Fkzb+l_Fmzb+l_Fbzb);

chip_Fxa = inv_l1*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m1*(l_Fkya+l_Fmya+l_Fbya)+inv_n1*(l_Fkza+l_Fmza+l_Fbza);
chip_Fya = inv_l2*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m2*(l_Fkya+l_Fmya+l_Fbya)+inv_n2*(l_Fkza+l_Fmza+l_Fbza);
chip_Fza = inv_l3*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m3*(l_Fkya+l_Fmya+l_Fbya)+inv_n3*(l_Fkza+l_Fmza+l_Fbza);

chip_Mxb = inv_l1*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m1*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n1*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Myb = inv_l2*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m2*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n2*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Mzb = inv_l3*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m3*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n3*(l_Mkzb+l_Mmzb+l_Mbzb);

chip_Mxa = inv_l1*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m1*(l_Mkya+l_Mmya+l_Mbya)+inv_n1*(l_Mkza+l_Mmza+l_Mbza);
chip_Mya = inv_l2*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m2*(l_Mkya+l_Mmya+l_Mbya)+inv_n2*(l_Mkza+l_Mmza+l_Mbza);
chip_Mza = inv_l3*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m3*(l_Mkya+l_Mmya+l_Mbya)+inv_n3*(l_Mkza+l_Mmza+l_Mbza);


// forces/moments applied to the beam ends
```

```
F(xa[0]) <+ -chip_Fxa;
F(xa[1]) <+ -chip_Fya;
F(xa[2]) <+ -chip_Fza;
F(xb[0]) <+ -chip_Fxb;
F(xb[1]) <+ -chip_Fyb;
F(xb[2]) <+ -chip_Fzb;

Tau(phia[2]) <+ -chip_Mza;
Tau(phib[2]) <+ -chip_Mzb;
Tau(phia[1]) <+ -chip_Mya;
Tau(phib[1]) <+ -chip_Myb;
Tau(phia[0]) <+ -chip_Mxa;
Tau(phib[0]) <+ -chip_Mxb;

// no electrical equations in mechanical submodule

end // end of analog block
endmodule// end of mechanical submodule
```

Verilog-A code for the electrical submodule is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"

module beam_elec(vm_m3, vp_m3, vm_m2, vp_m2, vm_m1, vp_m1, vm_poly, vp_poly);

inout vm_m3;
electrical vm_m3;
inout vp_m3;
electrical vp_m3;
inout vm_m2;
electrical vm_m2;
inout vp_m2;
electrical vp_m2;
inout vm_m1;
electrical vm_m1;
inout vp_m1;
electrical vp_m1;
inout vm_poly;
electrical vm_poly;
inout vp_poly;
electrical vp_poly;

// all parameters are default to be zero
parameter real l = 0;
parameter real w = 0;
parameter integer flag_m3 = 0;
parameter integer flag_m2 = 0;
parameter integer flag_m1 = 0;
parameter integer flag_poly = 0;
parameter real resistivity_m3 = 0;
parameter real resistivity_m2 = 0;
parameter real resistivity_m1 = 0;
parameter real resistivity_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;

    analog begin

        // If a layer exist, a resistor is inserted between corresponding electrical nodes at beam ends; capacitors are inserted between
this layer and its upper and lower layers, respectively
        // If a layer doesn't exsit, very large resistors are inserted between corresponding electrical nodes at beam ends and the
electrical ground, respectively; capacitors are inserted between the upper and lower layers directly
        // overlapping capacitance is calculated based on the overlap area and oxide thickess, divided by half and then applied to beam
ends
        // there are al-together 14 beam layer-composition cases for CMOS-MEMS process with m3, m2, m1 and poly

        if (flag_m3==1 && flag_m2==1 && flag_m1==1 && flag_poly==1) begin
        I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
        I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
        I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);
        I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

        I(vm_m3, vm_m2) <+ ddt(V(vm_m3, vm_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vp_m3, vp_m2) <+ ddt(V(vp_m3, vp_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vm_m2, vm_m1) <+ ddt(V(vm_m2, vm_m1))*'eps0*l*w/2.0/t_m2_m1_overpoly;
        I(vp_m2, vp_m1) <+ ddt(V(vp_m2, vp_m1))*'eps0*l*w/2.0/t_m2_m1_overpoly;
        I(vm_m1, vm_poly) <+ ddt(V(vm_m1, vm_poly))*'eps0*l*w/2.0/t_m1_poly;
        I(vp_m1, vp_poly) <+ ddt(V(vp_m1, vp_poly))*'eps0*l*w/2.0/t_m1_poly;
        I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
        I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;
        end;


        if (flag_m3==1 && flag_m2==1 && flag_m1==1 && flag_poly==0) begin
        I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
        I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
        I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);

        I(vm_m3, vm_m2) <+ ddt(V(vm_m3, vm_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vp_m3, vp_m2) <+ ddt(V(vp_m3, vp_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vm_m2, vm_m1) <+ ddt(V(vm_m2, vm_m1))*'eps0*l*w/2.0/t_m2_m1_overfield;
        I(vp_m2, vp_m1) <+ ddt(V(vp_m2, vp_m1))*'eps0*l*w/2.0/t_m2_m1_overfield;
        I(vm_m1) <+ ddt(V(vm_m1))*'eps0*l*w/2.0/t_m1_sub;
        I(vp_m1) <+ ddt(V(vp_m1))*'eps0*l*w/2.0/t_m1_sub;

        // no capacitor between m1 and poly since poly doesn't exist in this case

        I(vm_poly) <+ V(vm_poly)/1e15;
        I(vp_poly) <+ V(vp_poly)/1e15;
        end;


        if (flag_m3==1 && flag_m2==1 && flag_m1==0 && flag_poly==1) begin
        I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
        I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
        I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

        I(vm_m3, vm_m2) <+ ddt(V(vm_m3, vm_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vp_m3, vp_m2) <+ ddt(V(vp_m3, vp_m2))*'eps0*l*w/2.0/t_m3_m2;
        I(vm_m2, vm_poly) <+ ddt(V(vm_m2, vm_poly))*'eps0*l*w/2.0/t_m2_poly;
        I(vp_m2, vp_poly) <+ ddt(V(vp_m2, vp_poly))*'eps0*l*w/2.0/t_m2_poly;
        I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
        I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

        I(vm_m1) <+ V(vm_m1)/1e15;
        I(vp_m1) <+ V(vp_m1)/1e15;
```

Modeling and Simulation for Design of Suspended MEMS

```
end;

if (flag_m3==1 && flag_m2==1 && flag_m1==0 && flag_poly==0) begin
I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);

I(vm_m3, vm_m2) <+ ddt(V(vm_m3, vm_m2))*'eps0*l*w/2.0/t_m3_m2;
I(vp_m3, vp_m2) <+ ddt(V(vp_m3, vp_m2))*'eps0*l*w/2.0/t_m3_m2;
I(vm_m2) <+ ddt(V(vm_m2))*'eps0*l*w/2.0/t_m2_sub;
I(vp_m2) <+ ddt(V(vp_m2))*'eps0*l*w/2.0/t_m2_sub;

I(vm_m1) <+ V(vm_m1)/1e15;
I(vp_m1) <+ V(vp_m1)/1e15;
I(vm_poly) <+ V(vm_poly)/1e15;
I(vp_poly) <+ V(vp_poly)/1e15;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==1 && flag_poly==1) begin
I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);
I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

I(vm_m3, vm_m1) <+ ddt(V(vm_m3, vm_m1))*'eps0*l*w/2.0/t_m3_m1_overpoly;
I(vp_m3, vp_m1) <+ ddt(V(vp_m3, vp_m1))*'eps0*l*w/2.0/t_m3_m1_overpoly;
I(vm_m1, vm_poly) <+ ddt(V(vm_m1, vm_poly))*'eps0*l*w/2.0/t_m1_poly;
I(vp_m1, vp_poly) <+ ddt(V(vp_m1, vp_poly))*'eps0*l*w/2.0/t_m1_poly;
I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

I(vm_m2) <+ V(vm_m2)/1e15;
I(vp_m2) <+ V(vp_m2)/1e15;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==1 && flag_poly==0) begin
I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);

I(vm_m3, vm_m1) <+ ddt(V(vm_m3, vm_m1))*'eps0*l*w/2.0/t_m3_m1_overfield;
I(vp_m3, vp_m1) <+ ddt(V(vp_m3, vp_m1))*'eps0*l*w/2.0/t_m3_m1_overfield;
I(vm_m1) <+ ddt(V(vm_m1))*'eps0*l*w/2.0/t_m1_sub;
I(vp_m1) <+ ddt(V(vp_m1))*'eps0*l*w/2.0/t_m1_sub;

I(vm_m2) <+ V(vm_m2)/1e15;
I(vp_m2) <+ V(vp_m2)/1e15;
I(vm_poly) <+ V(vm_poly)/1e15;
I(vp_poly) <+ V(vp_poly)/1e15;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==0 && flag_poly==1) begin
I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

I(vm_m3, vm_poly) <+ ddt(V(vm_m3, vm_poly))*'eps0*l*w/2.0/t_m3_poly;
I(vp_m3, vp_poly) <+ ddt(V(vp_m3, vp_poly))*'eps0*l*w/2.0/t_m3_poly;
I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

I(vm_m2) <+ V(vm_m2)/1e15;
I(vp_m2) <+ V(vp_m2)/1e15;
I(vm_m1) <+ V(vm_m1)/1e15;
I(vp_m1) <+ V(vp_m1)/1e15;
end;

if (flag_m3==1 && flag_m2==0 && flag_m1==0 && flag_poly==0) begin
I(vm_m3, vp_m3) <+ V(vm_m3, vp_m3)/(resistivity_m3 * l/w);
```

```
I(vm_m3) <+ ddt(V(vm_m3))*'eps0*l*w/2.0/t_m3_sub;
I(vp_m3) <+ ddt(V(vp_m3))*'eps0*l*w/2.0/t_m3_sub;

I(vm_m2) <+ V(vm_m2)/1e15;
I(vp_m2) <+ V(vp_m2)/1e15;
I(vm_m1) <+ V(vm_m1)/1e15;
I(vp_m1) <+ V(vp_m1)/1e15;
I(vm_poly) <+ V(vm_poly)/1e15;
I(vp_poly) <+ V(vp_poly)/1e15;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==1 && flag_poly==1) begin
I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);
I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

I(vm_m2, vm_m1) <+ ddt(V(vm_m2, vm_m1))*'eps0*l*w/2.0/t_m2_m1_overpoly;
I(vp_m2, vp_m1) <+ ddt(V(vp_m2, vp_m1))*'eps0*l*w/2.0/t_m2_m1_overpoly;
I(vm_m1, vm_poly) <+ ddt(V(vm_m1, vm_poly))*'eps0*l*w/2.0/t_m1_poly;
I(vp_m1, vp_poly) <+ ddt(V(vp_m1, vp_poly))*'eps0*l*w/2.0/t_m1_poly;
I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

I(vm_m3) <+ V(vm_m3)/1e15;
I(vp_m3) <+ V(vp_m3)/1e15;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==1 && flag_poly==0) begin
I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);

I(vm_m2, vm_m1) <+ ddt(V(vm_m2, vm_m1))*'eps0*l*w/2.0/t_m2_m1_overfield;
I(vp_m2, vp_m1) <+ ddt(V(vp_m2, vp_m1))*'eps0*l*w/2.0/t_m2_m1_overfield;
I(vm_m1) <+ ddt(V(vm_m1))*'eps0*l*w/2.0/t_m1_sub;
I(vp_m1) <+ ddt(V(vp_m1))*'eps0*l*w/2.0/t_m1_sub;

I(vm_m3) <+ V(vm_m3)/1e15;
I(vp_m3) <+ V(vp_m3)/1e15;
I(vm_poly) <+ V(vm_poly)/1e15;
I(vp_poly) <+ V(vp_poly)/1e15;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==0 && flag_poly==1) begin
I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);
I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

I(vm_m2, vm_poly) <+ ddt(V(vm_m2, vm_poly))*'eps0*l*w/2.0/t_m2_poly;
I(vp_m2, vp_poly) <+ ddt(V(vp_m2, vp_poly))*'eps0*l*w/2.0/t_m2_poly;
I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

I(vm_m3) <+ V(vm_m3)/1e15;
I(vp_m3) <+ V(vp_m3)/1e15;
I(vm_m1) <+ V(vm_m1)/1e15;
I(vp_m1) <+ V(vp_m1)/1e15;
end;

if (flag_m3==0 && flag_m2==1 && flag_m1==0 && flag_poly==0) begin
I(vm_m2, vp_m2) <+ V(vm_m2, vp_m2)/(resistivity_m2 * l/w);

I(vm_m2) <+ ddt(V(vm_m2))*'eps0*l*w/2.0/t_m2_sub;
I(vp_m2) <+ ddt(V(vp_m2))*'eps0*l*w/2.0/t_m2_sub;

I(vm_m3) <+ V(vm_m3)/1e15;
```

```
        I(vp_m3) <+ V(vp_m3)/1e15;
        I(vm_m1) <+ V(vm_m1)/1e15;
        I(vp_m1) <+ V(vp_m1)/1e15;
        I(vm_poly) <+ V(vm_poly)/1e15;
        I(vp_poly) <+ V(vp_poly)/1e15;
        end;

        if (flag_m3==0 && flag_m2==0 && flag_m1==1 && flag_poly==1) begin
        I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);
        I(vm_poly, vp_poly) <+ V(vm_poly, vp_poly)/(resistivity_poly * l/w);

        I(vm_m1, vm_poly) <+ ddt(V(vm_m1, vm_poly))*'eps0*l*w/2.0/t_m1_poly;
        I(vp_m1, vp_poly) <+ ddt(V(vp_m1, vp_poly))*'eps0*l*w/2.0/t_m1_poly;
        I(vm_poly) <+ ddt(V(vm_poly))*'eps0*l*w/2.0/t_poly_sub;
        I(vp_poly) <+ ddt(V(vp_poly))*'eps0*l*w/2.0/t_poly_sub;

        I(vm_m3) <+ V(vm_m3)/1e15;
        I(vp_m3) <+ V(vp_m3)/1e15;
        I(vm_m2) <+ V(vm_m2)/1e15;
        I(vp_m2) <+ V(vp_m2)/1e15;
        end;

        if (flag_m3==0 && flag_m2==0 && flag_m1==1 && flag_poly==0) begin
        I(vm_m1, vp_m1) <+ V(vm_m1, vp_m1)/(resistivity_m1 * l/w);

        I(vm_m1) <+ ddt(V(vm_m1))*'eps0*l*w/2.0/t_m1_sub;
        I(vp_m1) <+ ddt(V(vp_m1))*'eps0*l*w/2.0/t_m1_sub;

        I(vm_m3) <+ V(vm_m3)/1e15;
        I(vp_m3) <+ V(vp_m3)/1e15;
        I(vm_m2) <+ V(vm_m2)/1e15;
        I(vp_m2) <+ V(vp_m2)/1e15;
        I(vm_poly) <+ V(vm_poly)/1e15;
        I(vp_poly) <+ V(vp_poly)/1e15;
        end;


        end  // of analog block
        endmodule  // of electrical submodule
```

A.6 Linear Beam Model with Trapezoidal Cross Section

In this section, the linear beam model with trapezoidal cross section is presented. The theoretical equations are given in Chapter 4.2.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
'include "../../design.h"
'include "../../process.h"

module beam3D_lin(phia, phib, xa, xb, va, vb, ta, tb);

inout [0:2] phia;
rotational [0:2] phia;
inout [0:2] phib;
rotational [0:2] phib;
inout [0:2] xa;
kinematic [0:2] xa;
inout [0:2] xb;
kinematic [0:2] xb;
inout va, vb;
electrical va, vb;
inout ta;
thermal ta;
inout tb;
thermal tb;


parameter integer flag_shear = 0;

parameter real l = 100e-6;
parameter real w = 2e-6;
parameter real thickness = 'default_thickness;
parameter real E = 'default_E;
parameter real density = 'default_density;

parameter real alpha = 0;
parameter real beta = 0;
parameter real gamma = 0;

parameter real resistivity = 'default_resistivity;
parameter real bloat = 4e-6;
parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real Poisson_ratio = 'default_Poisson_ratio;

parameter real Xc = 0;
parameter real Yc = 0;

parameter real theta_left = 90;  //left-hand  sidewall angle in degrees
parameter real theta_right = 90; //right-hand sidewall angle in degrees

// definition of internal variables

real Iz, Iy, G, Ip, It, m0, l_corner, ea, mass;
real dampx, dampy, dampz, scale, resistance;
```

```
// variables for shear deformation
real Asy, Sy, Srz, Asz, Sz, Sry, m0y, m0z;

real cos_alpha, sin_alpha, cos_beta, sin_beta, cos_gamma, sin_gamma;
real l1, m1, n1, l2, m2, n2, l3, m3, n3;
real inv_l1, inv_m1, inv_n1, inv_l2, inv_m2, inv_n2, inv_l3, inv_m3, inv_n3;

real chip_xa, chip_ya, chip_za, chip_xb, chip_yb, chip_zb;
real l_xa, l_ya, l_za, l_xb, l_yb, l_zb;

real chip_phixa, chip_phiya, chip_phiza, chip_phixb, chip_phiyb, chip_phizb;
real l_phixa, l_phiya, l_phiza, l_phixb, l_phiyb, l_phizb;

real l_Fkxb, l_Fkxa, l_Fkyb, l_Fkya, l_Fkzb, l_Fkza;
real l_Mkxa, l_Mkya, l_Mkza, l_Mkxb, l_Mkyb, l_Mkzb;

real l_Fmxb, l_Fmxa, l_Fmyb, l_Fmya, l_Fmzb, l_Fmza;
real l_Mmxa, l_Mmya, l_Mmza, l_Mmxb, l_Mmyb, l_Mmzb;

real l_Fbxb, l_Fbxa, l_Fbyb, l_Fbya, l_Fbzb, l_Fbza;
real l_Mbxa, l_Mbya, l_Mbza, l_Mbxb, l_Mbyb, l_Mbzb;

real l_Fxb, l_Fxa, l_Fyb, l_Fya, l_Fzb, l_Fza;
real l_Mxa, l_Mya, l_Mza, l_Mxb, l_Myb, l_Mzb;
real chip_Fxb, chip_Fxa, chip_Fyb, chip_Fya, chip_Fzb, chip_Fza;
real chip_Mxa, chip_Mya, chip_Mza, chip_Mxb, chip_Myb, chip_Mzb;

kinematic Vxa, Vxb, Vya, Vyb, Vza, Vzb;
rotational_omega  Vphixa, Vphixb, Vphiya, Vphiyb, Vphiza, Vphizb;

real k_1_1;
real k_2_1, k_2_2;
real k_3_1, k_3_2, k_3_3;
real k_4_1, k_4_2, k_4_3, k_4_4;
real k_5_1, k_5_2, k_5_3, k_5_4, k_5_5;
real k_6_1, k_6_2, k_6_3, k_6_4, k_6_5, k_6_6;
real k_7_1, k_7_2, k_7_3, k_7_4, k_7_5, k_7_6, k_7_7;
real k_8_1, k_8_2, k_8_3, k_8_4, k_8_5, k_8_6, k_8_7, k_8_8;
real k_9_1, k_9_2, k_9_3, k_9_4, k_9_5, k_9_6, k_9_7, k_9_8, k_9_9;
real k_10_1,k_10_2,k_10_3,k_10_4,k_10_5,k_10_6,k_10_7,k_10_8,k_10_9,k_10_10;
real k_11_1,k_11_2,k_11_3,k_11_4,k_11_5,k_11_6,k_11_7,k_11_8,k_11_9,k_11_10,k_11_11;
real k_12_1,k_12_2,k_12_3,k_12_4,k_12_5,k_12_6,k_12_7,k_12_8,k_12_9,k_12_10,k_12_11,k_12_12;


real m_1_1;
real m_2_1, m_2_2;
real m_3_1, m_3_2, m_3_3;
real m_4_1, m_4_2, m_4_3, m_4_4;
real m_5_1, m_5_2, m_5_3, m_5_4, m_5_5;
real m_6_1, m_6_2, m_6_3, m_6_4, m_6_5, m_6_6;
real m_7_1, m_7_2, m_7_3, m_7_4, m_7_5, m_7_6, m_7_7;
real m_8_1, m_8_2, m_8_3, m_8_4, m_8_5, m_8_6, m_8_7, m_8_8;
real m_9_1, m_9_2, m_9_3, m_9_4, m_9_5, m_9_6, m_9_7, m_9_8, m_9_9;
real m_10_1,m_10_2,m_10_3,m_10_4,m_10_5,m_10_6,m_10_7,m_10_8,m_10_9,m_10_10;
real m_11_1,m_11_2,m_11_3,m_11_4,m_11_5,m_11_6,m_11_7,m_11_8,m_11_9,m_11_10,m_11_11;
real m_12_1,m_12_2,m_12_3,m_12_4,m_12_5,m_12_6,m_12_7,m_12_8,m_12_9,m_12_10,m_12_11,m_12_12;

// internal variables for trapezoidal cross-section
real qtop,xtop,aside,bside,A0,P0,ax,bx,ss,R,r1,r2,r3,r4,RP2A,lamda,P1,A1;
real theta1, theta2, t, area, cy, cz, Iyz, Iyc, Izc, Iyzc, prin_angle;

analog begin

theta1 = theta_left /180*'M_PI;
theta2 = theta_right/180*'M_PI;
```

```
l_corner = l+0.3*w*(1-flag_shear);
resistance = resistivity*l_corner/w;
t = thickness;
area = 0.5*t*(w+w-t/tan(theta1)-t/tan(theta2));
mass = density*area*l_corner;

ea = E*area;  // coef used in [k] matrix

// start of calculation of planar and torsional moment of inertia
// for beams with trapezoidal cross section

// calculation of centroid coordinates

cy = (6*w*w-6*w*t/tan(theta2)
+t*t/sin(theta1)/sin(theta1)/sin(theta2)/sin(theta2)
*(-cos(2*theta1)+cos(2*theta2)))/
(12*w-6*t/sin(theta1)/sin(theta2)*sin(theta1+theta2));

cz = t*(-3*w+2*t/tan(theta1)+2*t/tan(theta2))
/(-6*w+3*t/tan(theta1)+3*t/tan(theta2));

// planar moments of inertia about centroids in the principle-axes frame

Iyc = (t*t*t/sin(theta1)/sin(theta2)*
(-6*w*w-2*t*t+6*w*w*cos(2*theta1)-3*w*w*cos(2*(theta1-theta2))
+6*w*w*cos(2*theta2)-3*w*w*cos(2*(theta1+theta2))
+2*t*t*cos(2*(theta1+theta2))+6*w*t*sin(2*theta1)
+6*w*t*sin(2*theta2)-6*w*t*sin(2*(theta1+theta2))))
/(144*(t*cos(theta2)*sin(theta1)+t*cos(theta1)*sin(theta2)
-2*w*sin(theta1)*sin(theta2)));

Izc = (t*(-96*w*w*w*w-144*w*w*t*t/tan(theta1)/tan(theta1)
-288*w*w*t*t/tan(theta1)/tan(theta2)
+144*w*w*t*t/tan(theta2)/tan(theta2)
-96*w*w*t*t/pow(sin(theta1),2)/pow(sin(theta2),2)
+48*w*w*t*t*cos(2*(theta1-theta2))/pow(sin(theta1),2)/pow(sin(theta2),2)
+72*w*w*t*t*(cos(2*theta1)-cos(2*theta2))/pow(sin(theta1),2)/pow(sin(theta2),2)
+48*w*w*t*t*cos(2*(theta1+theta2))/pow(sin(theta1),2)/pow(sin(theta2),2)
+8 *t*t*t*t*cos(2*theta1)*cos(2*theta1)/pow(sin(theta1),4)/pow(sin(theta2),4)
-16*t*t*t*t*cos(2*theta1)*cos(2*theta2)/pow(sin(theta1),4)/pow(sin(theta2),4)
+8 *t*t*t*t*cos(2*theta2)*cos(2*theta1)/pow(sin(theta1),4)/pow(sin(theta2),4)
+18*w*t*t*t/pow(sin(theta1),3)/pow(sin(theta2),3)*(sin(theta1-3*theta2)+sin(3*theta1-theta2))
+48*w*t*t*t*(cos(2*theta1)+cos(2*theta2))/pow(sin(theta1),3)/pow(sin(theta2),3)*sin(theta1-theta2)
+192*w*w*w*t/sin(theta1)/sin(theta2)*sin(theta1+theta2)
+102*w*t*t*t/pow(sin(theta1),3)/pow(sin(theta2),3)*sin(theta1+theta2)
-24 *w*t*t*t/pow(sin(theta1),3)/pow(sin(theta2),3)*sin(theta1+theta2)*(cos(2*(theta1+theta2))+cos(2*(theta1-theta2)))
-9  *t*t*t*t/pow(sin(theta1),4)/pow(sin(theta2),4)*sin(theta1+theta2)*(sin(3*theta1-theta2)-sin(theta1-3*theta2))
+27 *t*t*t*t/pow(sin(theta1),4)/pow(sin(theta2),4)*pow(sin(theta1+theta2),2)
-6  *w*t*t*t/pow(sin(theta1),3)/pow(sin(theta2),3)*sin(3*(theta1+theta2))
+3  *t*t*t*t/pow(sin(theta1),4)/pow(sin(theta2),4)*sin(theta1+theta2)*sin(3*(theta1+theta2))))
/(576*(-2*w+t/tan(theta1)+t/tan(theta2)));


// Iyzc in terms of cy&cz
Iyzc =
(t/pow(sin(theta1),2)/pow(sin(theta2),2)*(-12*pow(w,2)*cz
+ 24*w*cy*cz + 6*pow(w,2)*t - 12*w*cy*t + 2*(pow(w,2)*(6*cz - 3*t)
+ (4*cz - 3*t)*pow(t,2) + 6*w*cy*(-2*cz + t))*cos(2*theta1)
- 3*w*(w - 2*cy)*(2*cz - t)*cos(2*(theta1 - theta2))
+ 12*pow(w,2)*cz*cos(2*theta2) - 24*w*cy*cz*cos(2*theta2)
- 6*pow(w,2)*t*cos(2*theta2) + 12*w*cy*t*cos(2*theta2)
- 8*cz*pow(t,2)*cos(2*theta2) + 6*pow(t,3)*cos(2*theta2)
- 6*pow(w,2)*cz*cos(2*(theta1 + theta2))
+ 12*w*cy*cz*cos(2*(theta1 + theta2))
```

```
    + 3*pow(w,2)*t*cos(2*(theta1 + theta2))
    - 6*w*cy*t*cos(2*(theta1 + theta2)) - 12*cy*cz*t*sin(2*theta1)
    + 8*cy*pow(t,2)*sin(2*theta1) + 6*w*cz*t*sin(2*(theta1 - theta2))
    - 4*w*pow(t,2)*sin(2*(theta1 - theta2))
    + 12*w*cz*t*sin(2*theta2) - 12*cy*cz*t*sin(2*theta2)
    - 8*w*pow(t,2)*sin(2*theta2) + 8*cy*pow(t,2)*sin(2*theta2)
    - 6*w*cz*t*sin(2*(theta1 + theta2)) + 12*cy*cz*t*sin(2*(theta1 + theta2))
    + 4*w*pow(t,2)*sin(2*(theta1 + theta2))
    - 8*cy*pow(t,2)*sin(2*(theta1 + theta2))))/96;

// the angle between the principle axes and the coordinate axes of local frame
prin_angle = atan(-2*Iyzc/(Iyc-Izc))/2.0;

// moments of inertia in the local frame
Iy = (Iyc+Izc)/2+(Iyc-Izc)/2*cos(2*prin_angle)-Iyzc*sin(2*prin_angle);
Iz = (Iyc+Izc)/2-(Iyc-Izc)/2*cos(2*prin_angle)+Iyzc*sin(2*prin_angle);

// polar moment of inertia
Ip = Iy+Iz;

// shear modulus
G = E/2/(1+Poisson_ratio);

// torsional moment of inertia with sidewall angles

qtop = w-t/tan(theta1)-t/tan(theta2);
xtop = qtop*t/(w-qtop);
aside = t/sin(theta1);
bside = t/sin(theta2);
A0 = t*(w+qtop)/2.0;
P0 = aside+bside+w+qtop;
ax = (xtop+t)/sin(theta1);
bx = (xtop+t)/sin(theta2);

ss = (ax+bx+w)/2.0;
R = sqrt((ss-ax)*(ss-bx)*(ss-w)/ss);
if (t <= 2.0*R) R = t/2.0;

r1 = R*( 0.9951049 -0.002079125*(theta1/'M_PI*180.0)
    - 6.194423e-05*pow(theta1/'M_PI*180.0,2)
    + 2.65795e-07*pow(theta1/'M_PI*180.0,3) );
r2 = R*( 0.9951049 -0.002079125*(theta2/'M_PI*180.0)
    - 6.194423e-05*pow(theta2/'M_PI*180.0,2)
    + 2.65795e-07*pow(theta2/'M_PI*180.0,3) );
r3 = R*( 0.9951049 -0.002079125*(('M_PI-theta1)/'M_PI*180.0)
    - 6.194423e-05*pow(('M_PI-theta1)/'M_PI*180.0,2)
    + 2.65795e-07*pow(('M_PI-theta1)/'M_PI*180.0,3) );
r4 = R*( 0.9951049 -0.002079125*(('M_PI-theta2)/'M_PI*180.0)
    - 6.194423e-05*pow(('M_PI-theta2)/'M_PI*180.0,2)
    + 2.65795e-07*pow(('M_PI-theta2)/'M_PI*180.0,3) );

RP2A = R*P0/2.0/A0;
lamda= -0.4220676 +2.832549 *RP2A -1.224242*pow(RP2A,2) -0.1895882*pow(RP2A,3);

P1 = P0 -2.0*(r1/tan(theta1/2.0)+r2/tan(theta2/2.0)
    +r3/tan(('M_PI-theta1)/2.0)+r4/tan(('M_PI-theta2)/2.0))
    +r1*('M_PI-theta1)+r2*('M_PI-theta2)+r3*theta1+r4*theta2;

A1 = A0 - r1*r1*(1.0/tan(theta1/2.0)-('M_PI-theta1)/2.0)
    - r2*r2*(1.0/tan(theta2/2.0)-('M_PI-theta2)/2.0)
    - r3*r3*(1.0/tan(('M_PI-theta1)/2.0)-theta1/2.0)
    - r4*r4*(1.0/tan(('M_PI-theta2)/2.0)-theta2/2.0);

It = 2.0*A0*lamda*pow(A1/P1,2);
```

Modeling and Simulation for Design of Suspended MEMS

```
// end of calculation of planar and torsional moments of inertia

scale = 1e6;

// damping coefficients
dampx = visc_air*l_corner*(w+bloat)/air_gap;
dampy = visc_air*w*(l_corner+bloat)/air_gap;
dampz = visc_air*thickness*(l_corner+bloat)/air_gap;

// Euler angles
cos_alpha = cos(alpha /180*'M_PI);
cos_beta  = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta  = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// translational displacement in chip frame
chip_xa = Pos(xa[0]);
chip_ya = Pos(xa[1]);
chip_za = Pos(xa[2]);
chip_xb = Pos(xb[0]);
chip_yb = Pos(xb[1]);
chip_zb = Pos(xb[2]);

// angular displacement in chip frame
chip_phixa = scale*Theta(phia[0]);
chip_phiya = scale*Theta(phia[1]);
chip_phiza = scale*Theta(phia[2]);
chip_phixb = scale*Theta(phib[0]);
chip_phiyb = scale*Theta(phib[1]);
chip_phizb = scale*Theta(phib[2]);

// transform displacements from chip frame into local frame
l_xa = l1*chip_xa + m1*chip_ya + n1*chip_za;
l_ya = l2*chip_xa + m2*chip_ya + n2*chip_za;
l_za = l3*chip_xa + m3*chip_ya + n3*chip_za;
l_xb = l1*chip_xb + m1*chip_yb + n1*chip_zb;
l_yb = l2*chip_xb + m2*chip_yb + n2*chip_zb;
l_zb = l3*chip_xb + m3*chip_yb + n3*chip_zb;

l_phixa = l1*chip_phixa + m1*chip_phiya + n1*chip_phiza;
l_phiya = l2*chip_phixa + m2*chip_phiya + n2*chip_phiza;
```

Modeling and Simulation for Design of Suspended MEMS   273

```
l_phiza = l3*chip_phixa + m3*chip_phiya + n3*chip_phiza;
l_phixb = l1*chip_phixb + m1*chip_phiyb + n1*chip_phizb;
l_phiyb = l2*chip_phixb + m2*chip_phiyb + n2*chip_phizb;
l_phizb = l3*chip_phixb + m3*chip_phiyb + n3*chip_phizb;

// linear stiffness matrix [k], with shear deformation

// variables for shear deformation
Asy = 2.0/3.0*area; //effective shear area
Sy  = 1.0*12.0*E*Iz/G/pow(l_corner,2)/Asy * flag_shear; //shear deformation parameter
Srz = 1.0*pow(Iz/area, 0.5) * flag_shear; //radius of gyration about z-axis

Asz = 2.0/3.0*area; //effective shear area
Sz  = 1.0*12.0*E*Iy/G/pow(l_corner,2)/Asz * flag_shear; //shear deformation parameter
Sry = 1.0*pow(Iy/area, 0.5) * flag_shear; //radius of gyration about z-axis

k_1_1 = ea/l_corner;
k_2_1 = 0;
k_3_1 = 0;
k_4_1 = 0;
k_5_1 = 0;
k_6_1 = 0;
k_7_1 = -ea/l_corner;
k_8_1 = 0;
k_9_1 = 0;
k_10_1 = 0;
k_11_1 = 0;
k_12_1 = 0;

k_2_2 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_3_2 = 0;
k_4_2 = 0;
k_5_2 = 0;
k_6_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k_7_2 = 0;
k_8_2 = -12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_9_2 = 0;
k_10_2 = 0;
k_11_2 = 0;
k_12_2 = 6.0*E*Iz/pow(l_corner,2)/(1+Sy);


k_3_3 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_4_3 = 0;
k_5_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_6_3 = 0;
k_7_3 = 0;
k_8_3 = 0;
k_9_3 = -12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_10_3 = 0;
k_11_3 = -6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_12_3 = 0;

k_4_4 = G*It/l_corner;
k_5_4 = 0;
k_6_4 = 0;
k_7_4 = 0;
k_8_4 = 0;
k_9_4 = 0;
k_10_4 = -G*It/l_corner;
k_11_4 = 0;
k_12_4 = 0;

k_5_5 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k_6_5 = 0;
```

```
k_7_5 = 0;
k_8_5 = 0;
k_9_5 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_10_5 = 0;
k_11_5 = (2.0-Sz)*E*Iy/l_corner/(1+Sz);
k_12_5 = 0;

k_6_6 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);
k_7_6 = 0;
k_8_6 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);
k_9_6 = 0;
k_10_6 = 0;
k_11_6 = 0;
k_12_6 = (2.0-Sy)*E*Iz/l_corner/(1+Sy);

k_7_7 = ea/l_corner;
k_8_7 = 0;
k_9_7 = 0;
k_10_7 = 0;
k_11_7 = 0;
k_12_7 = 0;

k_8_8 = 12.0*E*Iz/pow(l_corner,3)/(1+Sy);
k_9_8 = 0;
k_10_8 = 0;
k_11_8 = 0;
k_12_8 = -6.0*E*Iz/pow(l_corner,2)/(1+Sy);

k_9_9 = 12.0*E*Iy/pow(l_corner,3)/(1+Sz);
k_10_9 = 0;
k_11_9 = 6.0*E*Iy/pow(l_corner,2)/(1+Sz);
k_12_9 = 0;

k_10_10 = G*It/l_corner;
k_11_10 = 0;
k_12_10 = 0;

k_11_11 = (4.0+Sz)*E*Iy/l_corner/(1+Sz);
k_12_11 = 0;

k_12_12 = (4.0+Sy)*E*Iz/l_corner/(1+Sy);


// [Fk]=[k][x], spring forces/moments in local frame

l_Fkxa = k_1_1*l_xa+k_2_1*l_ya+k_3_1*l_za
        +k_4_1*l_phixa+k_5_1*l_phiya+k_6_1*l_phiza
        +k_7_1*l_xb+k_8_1*l_yb+k_9_1*l_zb
        +k_10_1*l_phixb+k_11_1*l_phiyb+k_12_1*l_phizb;

l_Fkxb = k_7_1*l_xa+k_7_2*l_ya+k_7_3*l_za
        +k_7_4*l_phixa+k_7_5*l_phiya+k_7_6*l_phiza
        +k_7_7*l_xb+k_8_7*l_yb+k_9_7*l_zb
        +k_10_7*l_phixb+k_11_7*l_phiyb+k_12_7*l_phizb;

l_Fkya = k_2_1*l_xa+k_2_2*l_ya+k_3_2*l_za
        +k_4_2*l_phixa+k_5_2*l_phiya+k_6_2*l_phiza
        +k_7_2*l_xb+k_8_2*l_yb+k_9_2*l_zb
        +k_10_2*l_phixb+k_11_2*l_phiyb+k_12_2*l_phizb;

l_Fkyb = k_8_1*l_xa+k_8_2*l_ya+k_8_3*l_za
        +k_8_4*l_phixa+k_8_5*l_phiya+k_8_6*l_phiza
        +k_8_7*l_xb+k_8_8*l_yb+k_9_8*l_zb
        +k_10_8*l_phixb+k_11_8*l_phiyb+k_12_8*l_phizb;
```

```
l_Fkza = k_3_1*l_xa+k_3_2*l_ya+k_3_3*l_za
       +k_4_3*l_phixa+k_5_3*l_phiya+k_6_3*l_phiza
       +k_7_3*l_xb+k_8_3*l_yb+k_9_3*l_zb
       +k_10_3*l_phixb+k_11_3*l_phiyb+k_12_3*l_phizb;

l_Fkzb = k_9_1*l_xa+k_9_2*l_ya+k_9_3*l_za
       +k_9_4*l_phixa+k_9_5*l_phiya+k_9_6*l_phiza
       +k_9_7*l_xb+k_9_8*l_yb+k_9_9*l_zb
       +k_10_9*l_phixb+k_11_9*l_phiyb+k_12_9*l_phizb;

l_Mkxa = k_4_1*l_xa+k_4_2*l_ya+k_4_3*l_za
       +k_4_4*l_phixa+k_5_4*l_phiya+k_6_4*l_phiza
       +k_7_4*l_xb+k_8_4*l_yb+k_9_4*l_zb
       +k_10_4*l_phixb+k_11_4*l_phiyb+k_12_4*l_phizb;

l_Mkxb = k_10_1*l_xa+k_10_2*l_ya+k_10_3*l_za
       +k_10_4*l_phixa+k_10_5*l_phiya+k_10_6*l_phiza
       +k_10_7*l_xb+k_10_8*l_yb+k_10_9*l_zb
       +k_10_10*l_phixb+k_11_10*l_phiyb+k_12_10*l_phizb;

l_Mkya = k_5_1*l_xa+k_5_2*l_ya+k_5_3*l_za
       +k_5_4*l_phixa+k_5_5*l_phiya+k_6_5*l_phiza
       +k_7_5*l_xb+k_8_5*l_yb+k_9_5*l_zb
       +k_10_5*l_phixb+k_11_5*l_phiyb+k_12_5*l_phizb;

l_Mkyb = k_11_1*l_xa+k_11_2*l_ya+k_11_3*l_za
       +k_11_4*l_phixa+k_11_5*l_phiya+k_11_6*l_phiza
       +k_11_7*l_xb+k_11_8*l_yb+k_11_9*l_zb
       +k_11_10*l_phixb+k_11_11*l_phiyb+k_12_11*l_phizb;

l_Mkza = k_6_1*l_xa+k_6_2*l_ya+k_6_3*l_za
       +k_6_4*l_phixa+k_6_5*l_phiya+k_6_6*l_phiza
       +k_7_6*l_xb+k_8_6*l_yb+k_9_6*l_zb
       +k_10_6*l_phixb+k_11_6*l_phiyb+k_12_6*l_phizb;

l_Mkzb = k_12_1*l_xa+k_12_2*l_ya+k_12_3*l_za
       +k_12_4*l_phixa+k_12_5*l_phiya+k_12_6*l_phiza
       +k_12_7*l_xb+k_12_8*l_yb+k_12_9*l_zb
       +k_12_10*l_phixb+k_12_11*l_phiyb+k_12_12*l_phizb;


// mass matrix [m], with shear effects

m0y = density*thickness*w*l_corner/pow(1+Sy,2);
m0z = density*thickness*w*l_corner/pow(1+Sz,2);

m_1_1 = 1.0/3.0*density*thickness*w*l_corner;
m_2_1 = 0;
m_3_1 = 0;
m_4_1 = 0;
m_5_1 = 0;
m_6_1 = 0;
m_7_1 = 1.0/6.0*density*thickness*w*l_corner;
m_8_1 = 0;
m_9_1 = 0;
m_10_1 = 0;
m_11_1 = 0;
m_12_1 = 0;

m_2_2 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
m_3_2 = 0;
m_4_2 = 0;
m_5_2 = 0;
m_6_2 = m0y*l_corner*((11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));
m_7_2 = 0;
```

```
    m_8_2 = m0y*((9.0/70.0+3.0/10.0*Sy+1.0/6.0*Sy*Sy)-pow(Srz/l_corner,2)*6.0/5.0);
    m_9_2 = 0;
    m_10_2 = 0;
    m_11_2 = 0;
    m_12_2 = m0y*l_corner*(-(13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(1.0/10.0-1.0/2.0*Sy));

    m_3_3 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
    m_4_3 = 0;
    m_5_3 = -m0z*l_corner*((11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
    m_6_3 = 0;
    m_7_3 = 0;
    m_8_3 = 0;
    m_9_3 = m0z*((9.0/70.0+3.0/10.0*Sz+1.0/6.0*Sz*Sz)-pow(Sry/l_corner,2)*6.0/5.0);
    m_10_3 = 0;
    m_11_3 = -m0z*l_corner*(-(13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(1.0/10.0-1.0/2.0*Sz));
    m_12_3 = 0;

    m_4_4 = (Iy+Iz)/3.0*density*l_corner;
    m_5_4 = 0;
    m_6_4 = 0;
    m_7_4 = 0;
    m_8_4 = 0;
    m_9_4 = 0;
    m_10_4 = (Iy+Iz)/6.0*density*l_corner;
    m_11_4 = 0;
    m_12_4 = 0;

    m_5_5 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/
3.0*Sz*Sz));
    m_6_5 = 0;
    m_7_5 = 0;
    m_8_5 = 0;
    m_9_5 = -m0z*l_corner*((13.0/420.0+3.0/40.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
    m_10_5 = 0;
    m_11_5 = m0z*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/30.0-1.0/6.0*Sz+1.0/
6.0*Sz*Sz));
    m_12_5 = 0;

    m_6_6 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/
3.0*Sy*Sy));
    m_7_6 = 0;
    m_8_6 = m0y*l_corner*((13.0/420.0+3.0/40.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));
    m_9_6 = 0;
    m_10_6 = 0;
    m_11_6 = 0;
    m_12_6 = m0y*l_corner*l_corner*(-(1.0/140.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/30.0-1.0/6.0*Sy+1.0/
6.0*Sy*Sy));

    m_7_7 = 1.0/3.0*density*thickness*w*l_corner;
    m_8_7 = 0;
    m_9_7 = 0;
    m_10_7 = 0;
    m_11_7 = 0;
    m_12_7 = 0;

    m_8_8 = m0y*((13.0/35.0+7.0/10.0*Sy+1.0/3.0*Sy*Sy)+pow(Srz/l_corner,2)*6.0/5.0);
    m_9_8 = 0;
    m_10_8 = 0;
    m_11_8 = 0;
    m_12_8 = m0y*l_corner*(-(11.0/210.0+11.0/210.0*Sy+1.0/24.0*Sy*Sy)+pow(Srz/l_corner,2)*(-1.0/10.0+1.0/2.0*Sy));

    m_9_9 = m0z*((13.0/35.0+7.0/10.0*Sz+1.0/3.0*Sz*Sz)+pow(Sry/l_corner,2)*6.0/5.0);
    m_10_9 = 0;
    m_11_9 = -m0z*l_corner*(-(11.0/210.0+11.0/210.0*Sz+1.0/24.0*Sz*Sz)+pow(Sry/l_corner,2)*(-1.0/10.0+1.0/2.0*Sz));
    m_12_9 = 0;
```

Modeling and Simulation for Design of Suspended MEMS   277

```
    m_10_10 = (Iy+Iz)/3.0*density*l_corner;
    m_11_10 = 0;
    m_12_10 = 0;

    m_11_11 = m0z*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sz+1.0/120.0*Sz*Sz)+pow(Sry/l_corner,2)*(2.0/15.0+1.0/6.0*Sz+1.0/
3.0*Sz*Sz));
    m_12_11 = 0;

    m_12_12 = m0y*l_corner*l_corner*((1.0/105.0+1.0/60.0*Sy+1.0/120.0*Sy*Sy)+pow(Srz/l_corner,2)*(2.0/15.0+1.0/6.0*Sy+1.0/
3.0*Sy*Sy));


    Pos(Vxa) <+  ddt(l_xa);
    Pos(Vxb) <+  ddt(l_xb);
    Pos(Vya) <+  ddt(l_ya);
    Pos(Vyb) <+  ddt(l_yb);
    Pos(Vza) <+  ddt(l_za);
    Pos(Vzb) <+  ddt(l_zb);

    Omega(Vphixa) <+  ddt(l_phixa);
    Omega(Vphixb) <+  ddt(l_phixb);
    Omega(Vphiya) <+  ddt(l_phiya);
    Omega(Vphiyb) <+  ddt(l_phiyb);
    Omega(Vphiza) <+  ddt(l_phiza);
    Omega(Vphizb) <+  ddt(l_phizb);

    // [Fm]=[m][a], inertial forces/moments

    l_Fmxa = m_1_1*ddt(Pos(Vxa)) + m_7_1*ddt(Pos(Vxb));
    l_Fmxb = m_7_1*ddt(Pos(Vxa)) + m_7_7*ddt(Pos(Vxb));

    l_Fmya = m_6_2*ddt(Omega(Vphiza)) + m_12_2*ddt(Omega(Vphizb))
        + m_2_2*ddt(Pos(Vya)) + m_8_2*ddt(Pos(Vyb));
    l_Fmyb = m_8_6*ddt(Omega(Vphiza)) + m_12_8*ddt(Omega(Vphizb))
        + m_8_2*ddt(Pos(Vya)) + m_8_8*ddt(Pos(Vyb));

    l_Fmza = m_5_3*ddt(Omega(Vphiya)) + m_11_3*ddt(Omega(Vphiyb))
        + m_3_3*ddt(Pos(Vza)) + m_9_3*ddt(Pos(Vzb));
    l_Fmzb = m_9_5*ddt(Omega(Vphiya)) + m_11_9*ddt(Omega(Vphiyb))
        + m_9_3*ddt(Pos(Vza)) + m_9_9*ddt(Pos(Vzb));

    l_Mmxa = m_4_4*ddt(Omega(Vphixa)) + m_10_4*ddt(Omega(Vphixb));
    l_Mmxb = m_10_4*ddt(Omega(Vphixa)) + m_10_10*ddt(Omega(Vphixb));

    l_Mmya = m_5_5*ddt(Omega(Vphiya)) + m_11_5*ddt(Omega(Vphiyb))
    + m_5_3*ddt(Pos(Vza))+ m_9_5*ddt(Pos(Vzb));
    l_Mmyb = m_11_5*ddt(Omega(Vphiya)) + m_11_11*ddt(Omega(Vphiyb))
    + m_11_3*ddt(Pos(Vza)) + m_11_9*ddt(Pos(Vzb));

    l_Mmza = m_6_6*ddt(Omega(Vphiza)) + m_12_6*ddt(Omega(Vphizb))
    + m_6_2*ddt(Pos(Vya))+ m_8_6*ddt(Pos(Vyb));
    l_Mmzb = m_12_6*ddt(Omega(Vphiza)) + m_12_12*ddt(Omega(Vphizb))
    + m_12_2*ddt(Pos(Vya)) + m_12_8*ddt(Pos(Vyb));


    // [Fb]=[B][V], damping forces/moments
    // using damping matrix with shear effect

    l_Fbxa = dampx/mass*(m_1_1*ddt(l_xa) + m_7_1*ddt(l_xb));
    l_Fbxb = dampx/mass*(m_7_1*ddt(l_xa) + m_7_7*ddt(l_xb));

    l_Fbya = dampy/mass*(m_6_2*ddt(l_phiza) + m_12_2*ddt(l_phizb)
        + m_2_2*ddt(l_ya) + m_8_2*ddt(l_yb));
    l_Fbyb = dampy/mass*(m_8_6*ddt(l_phiza) + m_12_8*ddt(l_phizb)
```

```
            + m_8_2*ddt(l_ya) + m_8_8*ddt(l_yb));

l_Fbza = dampz/mass*(m_5_3*ddt(l_phiya) + m_11_3*ddt(l_phiyb)
        + m_3_3*ddt(l_za) + m_9_3*ddt(l_zb));
l_Fbzb = dampz/mass*(m_9_5*ddt(l_phiya) + m_11_9*ddt(l_phiyb)
        + m_9_3*ddt(l_za) + m_9_9*ddt(l_zb));

l_Mbxa = dampx/mass*(m_4_4*ddt(l_phixa) + m_10_4*ddt(l_phixb));
l_Mbxb = dampx/mass*(m_10_4*ddt(l_phixa) + m_10_10*ddt(l_phixb));

l_Mbya = dampz/mass*(m_5_5*ddt(l_phiya) + m_11_5*ddt(l_phiyb)
+ m_5_3*ddt(l_za)+ m_9_5*ddt(l_zb));
l_Mbyb = dampz/mass*(m_11_5*ddt(l_phiya) + m_11_11*ddt(l_phiyb)
+ m_11_3*ddt(l_za) + m_11_9*ddt(l_zb));

l_Mbza = dampy/mass*(m_6_6*ddt(l_phiza) + m_12_6*ddt(l_phizb)
+ m_6_2*ddt(l_ya)+ m_8_6*ddt(l_yb));
l_Mbzb = dampy/mass*(m_12_6*ddt(l_phiza) + m_12_12*ddt(l_phizb)
+ m_12_2*ddt(l_ya) + m_12_8*ddt(l_yb));


// sum up spring, inertial and damping forces/moments
// transform from local frame back to the chip frame

chip_Fxb = inv_l1*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m1*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n1*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fyb = inv_l2*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m2*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n2*(l_Fkzb+l_Fmzb+l_Fbzb);
chip_Fzb = inv_l3*(l_Fkxb+l_Fmxb+l_Fbxb)+inv_m3*(l_Fkyb+l_Fmyb+l_Fbyb)+inv_n3*(l_Fkzb+l_Fmzb+l_Fbzb);

chip_Fxa = inv_l1*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m1*(l_Fkya+l_Fmya+l_Fbya)+inv_n1*(l_Fkza+l_Fmza+l_Fbza);
chip_Fya = inv_l2*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m2*(l_Fkya+l_Fmya+l_Fbya)+inv_n2*(l_Fkza+l_Fmza+l_Fbza);
chip_Fza = inv_l3*(l_Fkxa+l_Fmxa+l_Fbxa)+inv_m3*(l_Fkya+l_Fmya+l_Fbya)+inv_n3*(l_Fkza+l_Fmza+l_Fbza);

chip_Mxb = inv_l1*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m1*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n1*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Myb = inv_l2*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m2*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n2*(l_Mkzb+l_Mmzb+l_Mbzb);
chip_Mzb = inv_l3*(l_Mkxb+l_Mmxb+l_Mbxb)+inv_m3*(l_Mkyb+l_Mmyb+l_Mbyb)+inv_n3*(l_Mkzb+l_Mmzb+l_Mbzb);

chip_Mxa = inv_l1*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m1*(l_Mkya+l_Mmya+l_Mbya)+inv_n1*(l_Mkza+l_Mmza+l_Mbza);
chip_Mya = inv_l2*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m2*(l_Mkya+l_Mmya+l_Mbya)+inv_n2*(l_Mkza+l_Mmza+l_Mbza);
chip_Mza = inv_l3*(l_Mkxa+l_Mmxa+l_Mbxa)+inv_m3*(l_Mkya+l_Mmya+l_Mbya)+inv_n3*(l_Mkza+l_Mmza+l_Mbza);


// forces/moments applied to the beam ends
F(xa[0]) <+ -chip_Fxa;
F(xa[1]) <+ -chip_Fya;
F(xa[2]) <+ -chip_Fza;
F(xb[0]) <+ -chip_Fxb;
F(xb[1]) <+ -chip_Fyb;
F(xb[2]) <+ -chip_Fzb;

Tau(phia[2]) <+ -chip_Mza;
Tau(phib[2]) <+ -chip_Mzb;
Tau(phia[1]) <+ -chip_Mya;
Tau(phib[1]) <+ -chip_Myb;
Tau(phia[0]) <+ -chip_Mxa;
Tau(phib[0]) <+ -chip_Mxb;

//Calculate beam's electrical relation, I=V/R
I(vb, va) <+ V(vb, va)/resistance;

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

A.7 Rigid Plate Model

The rigid plate model given in this section covers the 3D motions of rigid plate elements with rectangular shape and single-layer structure. The plate model has twelve connection terminals, each having an individual joint-offset parameter, as illustrated in Chapter 3.4.1. The part about Coriolis force and centrifugal force is done by the peer student, Sita Iyer.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
'include "../../process.h"
'include "../../design.h"

module plate3D_rigid(x1, v1, x2, v2, x3, v3, x4, v4, x5, v5, x6, v6, x7, v7, x8, v8, x9, v9, x10, v10, x11, v11, x12, v12, phi,
OMG_ext, ax_ext, t);

inout [0:2] x1;
kinematic [0:2] x1;
inout v1;
electrical v1;

inout [0:2] x2;
kinematic [0:2] x2;
inout v2;
electrical v2;

inout [0:2] x3;
kinematic [0:2] x3;
inout v3;
electrical v3;

inout [0:2] x4;
kinematic [0:2] x4;
inout v4;
electrical v4;

inout [0:2] x5;
kinematic [0:2] x5;
inout v5;
electrical v5;

inout [0:2] x6;
kinematic [0:2] x6;
inout v6;
electrical v6;

inout [0:2] x7;
kinematic [0:2] x7;
inout v7;
electrical v7;

inout [0:2] x8;
kinematic [0:2] x8;
inout v8;
electrical v8;
```

```
    inout [0:2] x9;
    kinematic [0:2] x9;
    inout v9;
    electrical v9;

    inout [0:2] x10;
    kinematic [0:2] x10;
    inout v10;
    electrical v10;

    inout [0:2] x11;
    kinematic [0:2] x11;
    inout v11;
    electrical v11;

    inout [0:2] x12;
    kinematic [0:2] x12;
    inout v12;
    electrical v12;

    inout [0:2] phi;
    rotational [0:2] phi;

        inout [0:2] OMG_ext; // external rotation
        rotational [0:2] OMG_ext;

        inout [0:2] ax_ext; // external linear acceleration
        kinematic  [0:2] ax_ext;

        inout t;
        thermal t;

    // Internal nodes for middle point of the plate
    kinematic xm, ym, zm;
    rotational phizm, phiym, phixm;
    electrical v_mid;

    // internal nodes for Coriolis force and centrifugal force
        kinematic [0:2] vel;
        kinematic [0:2] a_int;
        rotational [0:2] dphidt;
        rotational [0:2] THT;
        rotational [0:2] dOMGdt;


    parameter real unitl = 'default_plate_unitl;
    parameter real unitw = 'default_plate_unitw;
    parameter real unitnum_x = 'default_plate_unitnum_x;
    parameter real unitnum_y = 'default_plate_unitnum_y;

    parameter real alpha = 0;
    parameter real beta  = 0;
    parameter real gamma  = 0;

    parameter real fraction_holes = 'default_plate_fraction_holes;

    parameter real joint_offset_1 = 'default_plate_joint_offset;
    parameter real joint_offset_3 = 'default_plate_joint_offset;
    parameter real joint_offset_4 = 'default_plate_joint_offset;
    parameter real joint_offset_6 = 'default_plate_joint_offset;
    parameter real joint_offset_7 = 'default_plate_joint_offset;
    parameter real joint_offset_9 = 'default_plate_joint_offset;
    parameter real joint_offset_10 = 'default_plate_joint_offset;
    parameter real joint_offset_12 = 'default_plate_joint_offset;
```

Modeling and Simulation for Design of Suspended MEMS   281

```
parameter real Xc = 0;
parameter real Yc = 0;
parameter real Zc = 0;

parameter real X_origin = 0;
parameter real Y_origin = 0;
parameter real Z_origin = 0;

parameter real resistivity = 'default_resistivity;
parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real bloat = 4u;
parameter real thickness = 'default_thickness;
parameter real density = 'default_density;

real l,w,area, ms, iiz, iiy, iix;
real dampx,dampy,dampz, dampphix,dampphiy,dampphiz;
real xg, yg, zg;

real cos_alpha, cos_beta, cos_gamma, sin_alpha, sin_beta, sin_gamma;
real l1, m1, n1 ;
real l2, m2, n2 ;
real l3, m3, n3 ;
real inv_l1, inv_m1, inv_n1 ;
real inv_l2, inv_m2, inv_n2 ;
real inv_l3, inv_m3, inv_n3 ;

real l_phixm, l_phiym, l_phizm;

real  fl_x1_xm, fl_x2_xm, fl_x3_xm, fl_x4_xm;
real  fl_x5_xm, fl_x6_xm, fl_x7_xm, fl_x8_xm;
real  fl_x9_xm, fl_x10_xm, fl_x11_xm, fl_x12_xm, fl_xm;
real  fl_y1_ym, fl_y2_ym, fl_y3_ym, fl_y4_ym;
real  fl_y5_ym, fl_y6_ym, fl_y7_ym, fl_y8_ym;
real  fl_y9_ym, fl_y10_ym, fl_y11_ym, fl_y12_ym, fl_ym;
real  fl_z1_zm, fl_z2_zm, fl_z3_zm, fl_z4_zm;
real  fl_z5_zm, fl_z6_zm, fl_z7_zm, fl_z8_zm;
real  fl_z9_zm, fl_z10_zm, fl_z11_zm, fl_z12_zm, fl_zm;

real fchip_xm,fchip_ym,fchip_zm;
real tql_phixm, tql_phiym, tql_phizm;
real tqchip_phixm, tqchip_phiym, tqchip_phizm;

real x1_xm, x2_xm, x3_xm, x4_xm;
real x5_xm, x6_xm, x7_xm, x8_xm;
real x9_xm, x10_xm, x11_xm, x12_xm;
real y1_ym, y2_ym, y3_ym, y4_ym;
real y5_ym, y6_ym, y7_ym, y8_ym;
real y9_ym, y10_ym, y11_ym, y12_ym;
real z1_zm, z2_zm, z3_zm, z4_zm;
real z5_zm, z6_zm, z7_zm, z8_zm;
real z9_zm, z10_zm, z11_zm, z12_zm;

real l_x1_xm, l_x2_xm, l_x3_xm, l_x4_xm;
real l_x5_xm, l_x6_xm, l_x7_xm, l_x8_xm;
real l_x9_xm, l_x10_xm, l_x11_xm, l_x12_xm;
real l_y1_ym, l_y2_ym, l_y3_ym, l_y4_ym;
real l_y5_ym, l_y6_ym, l_y7_ym, l_y8_ym;
real l_y9_ym, l_y10_ym, l_y11_ym, l_y12_ym;
real l_z1_zm, l_z2_zm, l_z3_zm, l_z4_zm;
real l_z5_zm, l_z6_zm, l_z7_zm, l_z8_zm;
real l_z9_zm, l_z10_zm, l_z11_zm, l_z12_zm;

real cos_alpha_2, sin_alpha_2, cos_beta_2, sin_beta_2, cos_gamma_2, sin_gamma_2;
```

```
    real euler_l1,euler_m1,euler_n1;
    real euler_l2,euler_m2,euler_n2;
    real euler_l3,euler_m3,euler_n3;

    kinematic xtop,ytop,ztop,xbot,ybot,zbot;
    rotational phixtop,phiytop,phiztop,phixbot,phiybot,phizbot;
    real l_xtop_xm,l_ytop_ym,l_ztop_zm,l_xbot_xm,l_ybot_ym,l_zbot_zm;
    real xtop_xm,ytop_ym,ztop_zm,xbot_xm,ybot_ym,zbot_zm;
    real fl_xtop_xm,fl_ytop_ym,fl_ztop_zm,fl_xbot_xm,fl_ybot_ym,fl_zbot_zm;

    real L_x1_phiz, L_y1_phiz, L_z1_phix, L_z1_phiy;
    real L_x2_phiz, L_y2_phiz, L_z2_phix, L_z2_phiy;
    real L_x3_phiz, L_y3_phiz, L_z3_phix, L_z3_phiy;
    real L_x4_phiz, L_y4_phiz, L_z4_phix, L_z4_phiy;
    real L_x5_phiz, L_y5_phiz, L_z5_phix, L_z5_phiy;
    real L_x6_phiz, L_y6_phiz, L_z6_phix, L_z6_phiy;
    real L_x7_phiz, L_y7_phiz, L_z7_phix, L_z7_phiy;
    real L_x8_phiz, L_y8_phiz, L_z8_phix, L_z8_phiy;
    real L_x9_phiz, L_y9_phiz, L_z9_phix, L_z9_phiy;
    real L_x10_phiz, L_y10_phiz, L_z10_phix, L_z10_phiy;
    real L_x11_phiz, L_y11_phiz, L_z11_phix, L_z11_phiy;
    real L_x12_phiz, L_y12_phiz, L_z12_phix, L_z12_phiy;

    real resistance;

    //variables for Coriolis force and centrifugal force, by Sita

      // These are the numbers for the
      // position of the chip w.r.t point about which the rotation is taking place
      // Function for computing the cross product of two vectors
    // function is changed to analog function to comply with Verilog-AMS OVI 2.0 standard
      analog function real crossprod;
        input v1x, v1y, v1z, v2x, v2y, v2z, component;
        real v1x, v1y, v1z, v2x, v2y, v2z, component;
         real yw, zw, z0;
        if (component == 0)
     crossprod = v1y*v2z - v1z*v2y;
         else if (component == 1)
     crossprod = v1z*v2x - v1x*v2z;
        else if (component == 2)
     crossprod = v1x*v2y - v1y*v2x;
         else
     crossprod = 0;
      endfunction // crossprod

      real  r[0:2];          // radius vector in chip frame
      real  ag[0:2];          // external accelerations after rotation
      real  acc[0:2];        // inertial acceleration vector in chip frame
      real  OMGr[0:2];    // crossprod of OMG and r in chip frame
      real  f0, f1, f2;       // For saving the inertial forces acting in each direction
      real  a0, a1, a2;

      real  f0_1, f1_1, f2_1; // For saving the forces acting in each direction

      // For rotation of the external acceleration to the chip frame
      // the elements of the rotation matrix have tht_ prefixed to them to indicate
      // that they are for rotation of the external acceleration to the chip frame

      real  cos_tht0, cos_tht1, cos_tht2, sin_tht0, sin_tht1, sin_tht2;
      real  tht_l1, tht_m1, tht_n1 ;
      real  tht_l2, tht_m2, tht_n2 ;
      real  tht_l3, tht_m3, tht_n3 ;
      real  tht_inv_l1, tht_inv_m1, tht_inv_n1 ;
      real  tht_inv_l2, tht_inv_m2, tht_inv_n2 ;
      real  tht_inv_l3, tht_inv_m3, tht_inv_n3 ;
```

Modeling and Simulation for Design of Suspended MEMS

```
analog begin

l= unitl*unitnum_y;
w= unitw*unitnum_x;
area = l*w*(1-fraction_holes);
resistance = resistivity * (l/w);

ms = density*thickness*area;
iiz = ms*(l*l+w*w)/12;
iiy = ms*(thickness*thickness+w*w)/12;
iix = ms*(thickness*thickness+l*l)/12;

dampx = visc_air/air_gap*((w+bloat)*(l+bloat));
dampy = visc_air/air_gap*((w+bloat)*(l+bloat));
dampz = visc_air/air_gap*((w+bloat)+(l+bloat))*2.0*thickness;
dampphix = visc_air/air_gap*(l*l+thickness*thickness)*2.0/12.0;
dampphiy = visc_air/air_gap*(w*w+thickness*thickness)*2.0/12.0;
dampphiz = visc_air/air_gap*(w*w+l*l)/12.0;

cos_alpha = cos(alpha /180*'M_PI);
cos_beta = cos(beta /180*'M_PI);
cos_gamma = cos(gamma /180*'M_PI);
sin_alpha = sin(alpha /180*'M_PI);
sin_beta = sin(beta /180*'M_PI);
sin_gamma = sin(gamma /180*'M_PI);

// transformation matrix from chip frame to local frame

l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 = cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame

inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// transfer angular displacements from chip fram to local frame
l_phixm = 1e6*(l1*Theta(phixm) + m1*Theta(phiym) + n1*Theta(phizm));
l_phiym = 1e6*(l2*Theta(phixm) + m2*Theta(phiym) + n2*Theta(phizm));
l_phizm = 1e6*(l3*Theta(phixm) + m3*Theta(phiym) + n3*Theta(phizm));

// transformation matrix from displaced frame to local frame, based on angular displacements

cos_alpha_2 = cos(l_phixm);
sin_alpha_2 = sin(l_phixm);
cos_beta_2  = cos(l_phiym);
sin_beta_2  = sin(l_phiym);
cos_gamma_2  = cos(l_phizm);
sin_gamma_2  = sin(l_phizm);
```

```
euler_l1 =  cos_beta_2*cos_gamma_2;
euler_l2 =  cos_beta_2*sin_gamma_2;
euler_l3 = -sin_beta_2;
euler_m1 =  sin_alpha_2*sin_beta_2*cos_gamma_2-cos_alpha_2*sin_gamma_2;
euler_m2 =  sin_alpha_2*sin_beta_2*sin_gamma_2+cos_alpha_2*cos_gamma_2;
euler_m3 =  sin_alpha_2*cos_beta_2;
euler_n1 =  cos_alpha_2*sin_beta_2*cos_gamma_2+sin_alpha_2*sin_gamma_2;
euler_n2 =  cos_alpha_2*sin_beta_2*sin_gamma_2-sin_alpha_2*cos_gamma_2;
euler_n3 =  cos_alpha_2*cos_beta_2;

// rigid-body constraints in the displaced frame

x11_xm = 0;
y11_ym = l/2;
z11_zm = 0;

x5_xm = 0;
y5_ym = -l/2;
z5_zm = 0;

x2_xm = -w/2;
y2_ym = 0;
z2_zm = 0;

x8_xm = w/2;
y8_ym = 0;
z8_zm = 0;

x1_xm = -w/2;
y1_ym = l/2-joint_offset_1;
z1_zm = 0;

x3_xm = -w/2;
y3_ym = -(l/2-joint_offset_3);
z3_zm = 0;

x7_xm = w/2;
y7_ym = -(l/2-joint_offset_7);
z7_zm = 0;

x9_xm = w/2;
y9_ym = l/2-joint_offset_9;
z9_zm = 0;

x4_xm = -(w/2-joint_offset_4);
y4_ym = -l/2;
z4_zm = 0;

x6_xm = w/2-joint_offset_6;
y6_ym = -l/2;
z6_zm = 0;

x12_xm = -(w/2-joint_offset_12);
y12_ym = l/2;
z12_zm = 0;

x10_xm = w/2-joint_offset_10;
y10_ym = l/2;
z10_zm = 0;

xtop_xm = 0;
ytop_ym = 0;
ztop_zm = thickness/2;
```

```
xbot_xm = 0;
ybot_ym = 0;
zbot_zm = -thickness/2;

// rigid-body constraints in the local frame

l_x11_xm = (euler_l1*x11_xm + euler_m1*y11_ym + euler_n1*z11_zm) - x11_xm;
l_y11_ym = (euler_l2*x11_xm + euler_m2*y11_ym + euler_n2*z11_zm) - y11_ym;
l_z11_zm = (euler_l3*x11_xm + euler_m3*y11_ym + euler_n3*z11_zm) - z11_zm;

l_x5_xm = (euler_l1*x5_xm + euler_m1*y5_ym + euler_n1*z5_zm) - x5_xm;
l_y5_ym = (euler_l2*x5_xm + euler_m2*y5_ym + euler_n2*z5_zm) - y5_ym;
l_z5_zm = (euler_l3*x5_xm + euler_m3*y5_ym + euler_n3*z5_zm) - z5_zm;

l_x2_xm = (euler_l1*x2_xm + euler_m1*y2_ym + euler_n1*z2_zm) - x2_xm;
l_y2_ym = (euler_l2*x2_xm + euler_m2*y2_ym + euler_n2*z2_zm) - y2_ym;
l_z2_zm = (euler_l3*x2_xm + euler_m3*y2_ym + euler_n3*z2_zm) - z2_zm;

l_x8_xm = (euler_l1*x8_xm + euler_m1*y8_ym + euler_n1*z8_zm) - x8_xm;
l_y8_ym = (euler_l2*x8_xm + euler_m2*y8_ym + euler_n2*z8_zm) - y8_ym;
l_z8_zm = (euler_l3*x8_xm + euler_m3*y8_ym + euler_n3*z8_zm) - z8_zm;

l_x1_xm = (euler_l1*x1_xm + euler_m1*y1_ym + euler_n1*z1_zm) - x1_xm;
l_y1_ym = (euler_l2*x1_xm + euler_m2*y1_ym + euler_n2*z1_zm) - y1_ym;
l_z1_zm = (euler_l3*x1_xm + euler_m3*y1_ym + euler_n3*z1_zm) - z1_zm;

l_x3_xm = (euler_l1*x3_xm + euler_m1*y3_ym + euler_n1*z3_zm) - x3_xm;
l_y3_ym = (euler_l2*x3_xm + euler_m2*y3_ym + euler_n2*z3_zm) - y3_ym;
l_z3_zm = (euler_l3*x3_xm + euler_m3*y3_ym + euler_n3*z3_zm) - z3_zm;

l_x4_xm = (euler_l1*x4_xm + euler_m1*y4_ym + euler_n1*z4_zm) - x4_xm;
l_y4_ym = (euler_l2*x4_xm + euler_m2*y4_ym + euler_n2*z4_zm) - y4_ym;
l_z4_zm = (euler_l3*x4_xm + euler_m3*y4_ym + euler_n3*z4_zm) - z4_zm;

l_x6_xm = (euler_l1*x6_xm + euler_m1*y6_ym + euler_n1*z6_zm) - x6_xm;
l_y6_ym = (euler_l2*x6_xm + euler_m2*y6_ym + euler_n2*z6_zm) - y6_ym;
l_z6_zm = (euler_l3*x6_xm + euler_m3*y6_ym + euler_n3*z6_zm) - z6_zm;

l_x7_xm = (euler_l1*x7_xm + euler_m1*y7_ym + euler_n1*z7_zm) - x7_xm;
l_y7_ym = (euler_l2*x7_xm + euler_m2*y7_ym + euler_n2*z7_zm) - y7_ym;
l_z7_zm = (euler_l3*x7_xm + euler_m3*y7_ym + euler_n3*z7_zm) - z7_zm;

l_x9_xm = (euler_l1*x9_xm + euler_m1*y9_ym + euler_n1*z9_zm) - x9_xm;
l_y9_ym = (euler_l2*x9_xm + euler_m2*y9_ym + euler_n2*z9_zm) - y9_ym;
l_z9_zm = (euler_l3*x9_xm + euler_m3*y9_ym + euler_n3*z9_zm) - z9_zm;

l_x10_xm = (euler_l1*x10_xm + euler_m1*y10_ym + euler_n1*z10_zm) - x10_xm;
l_y10_ym = (euler_l2*x10_xm + euler_m2*y10_ym + euler_n2*z10_zm) - y10_ym;
l_z10_zm = (euler_l3*x10_xm + euler_m3*y10_ym + euler_n3*z10_zm) - z10_zm;

l_x12_xm = (euler_l1*x12_xm + euler_m1*y12_ym + euler_n1*z12_zm) - x12_xm;
l_y12_ym = (euler_l2*x12_xm + euler_m2*y12_ym + euler_n2*z12_zm) - y12_ym;
l_z12_zm = (euler_l3*x12_xm + euler_m3*y12_ym + euler_n3*z12_zm) - z12_zm;

l_xtop_xm = (euler_l1*xtop_xm + euler_m1*ytop_ym + euler_n1*ztop_zm) - xtop_xm;
l_ytop_ym = (euler_l2*xtop_xm + euler_m2*ytop_ym + euler_n2*ztop_zm) - ytop_ym;
l_ztop_zm = (euler_l3*xtop_xm + euler_m3*ytop_ym + euler_n3*ztop_zm) - ztop_zm;

l_xbot_xm = (euler_l1*xbot_xm + euler_m1*ybot_ym + euler_n1*zbot_zm) - xbot_xm;
l_ybot_ym = (euler_l2*xbot_xm + euler_m2*ybot_ym + euler_n2*zbot_zm) - ybot_ym;
l_zbot_zm = (euler_l3*xbot_xm + euler_m3*ybot_ym + euler_n3*zbot_zm) - zbot_zm;

// inertial forces are now included in Sita's part

// damping forces in local frame
```

```
xg = l1*Pos(xm) + m1*Pos(ym) + n1*Pos(zm);
yg = l2*Pos(xm) + m2*Pos(ym) + n2*Pos(zm);
zg = l3*Pos(xm) + m3*Pos(ym) + n3*Pos(zm);

fl_xm = dampx*ddt(xg);
fl_ym = dampy*ddt(yg);
fl_zm = dampz*ddt(zg);

// damping forces transformed back to chip frame
fchip_xm = inv_l1*fl_xm + inv_m1*fl_ym + inv_n1*fl_zm;
fchip_ym = inv_l2*fl_xm + inv_m2*fl_ym + inv_n2*fl_zm;
fchip_zm = inv_l3*fl_xm + inv_m3*fl_ym + inv_n3*fl_zm;

// apply damping forces to the internal nodes for middle point of the plate
F(xm) <+ -fchip_xm;
F(ym) <+ -fchip_ym;
F(zm) <+ -fchip_zm;

// forces transfered from chip frame into local frame to calculate moments

fl_x11_xm = l1*F(x11[0],xm)+m1*F(x11[1],ym)+n1*F(x11[2],zm);
fl_y11_ym = l2*F(x11[0],xm)+m2*F(x11[1],ym)+n2*F(x11[2],zm);
fl_z11_zm = l3*F(x11[0],xm)+m3*F(x11[1],ym)+n3*F(x11[2],zm);

fl_x5_xm = l1*F(x5[0],xm)+m1*F(x5[1],ym)+n1*F(x5[2],zm);
fl_y5_ym = l2*F(x5[0],xm)+m2*F(x5[1],ym)+n2*F(x5[2],zm);
fl_z5_zm = l3*F(x5[0],xm)+m3*F(x5[1],ym)+n3*F(x5[2],zm);

fl_x2_xm = l1*F(x2[0],xm)+m1*F(x2[1],ym)+n1*F(x2[2],zm);
fl_y2_ym = l2*F(x2[0],xm)+m2*F(x2[1],ym)+n2*F(x2[2],zm);
fl_z2_zm = l3*F(x2[0],xm)+m3*F(x2[1],ym)+n3*F(x2[2],zm);

fl_x8_xm = l1*F(x8[0],xm)+m1*F(x8[1],ym)+n1*F(x8[2],zm);
fl_y8_ym = l2*F(x8[0],xm)+m2*F(x8[1],ym)+n2*F(x8[2],zm);
fl_z8_zm = l3*F(x8[0],xm)+m3*F(x8[1],ym)+n3*F(x8[2],zm);

fl_x1_xm = l1*F(x1[0],xm)+m1*F(x1[1],ym)+n1*F(x1[2],zm);
fl_y1_ym = l2*F(x1[0],xm)+m2*F(x1[1],ym)+n2*F(x1[2],zm);
fl_z1_zm = l3*F(x1[0],xm)+m3*F(x1[1],ym)+n3*F(x1[2],zm);

fl_x3_xm = l1*F(x3[0],xm)+m1*F(x3[1],ym)+n1*F(x3[2],zm);
fl_y3_ym = l2*F(x3[0],xm)+m2*F(x3[1],ym)+n2*F(x3[2],zm);
fl_z3_zm = l3*F(x3[0],xm)+m3*F(x3[1],ym)+n3*F(x3[2],zm);

fl_x4_xm = l1*F(x4[0],xm)+m1*F(x4[1],ym)+n1*F(x4[2],zm);
fl_y4_ym = l2*F(x4[0],xm)+m2*F(x4[1],ym)+n2*F(x4[2],zm);
fl_z4_zm = l3*F(x4[0],xm)+m3*F(x4[1],ym)+n3*F(x4[2],zm);

fl_x6_xm = l1*F(x6[0],xm)+m1*F(x6[1],ym)+n1*F(x6[2],zm);
fl_y6_ym = l2*F(x6[0],xm)+m2*F(x6[1],ym)+n2*F(x6[2],zm);
fl_z6_zm = l3*F(x6[0],xm)+m3*F(x6[1],ym)+n3*F(x6[2],zm);

fl_x7_xm = l1*F(x7[0],xm)+m1*F(x7[1],ym)+n1*F(x7[2],zm);
fl_y7_ym = l2*F(x7[0],xm)+m2*F(x7[1],ym)+n2*F(x7[2],zm);
fl_z7_zm = l3*F(x7[0],xm)+m3*F(x7[1],ym)+n3*F(x7[2],zm);

fl_x9_xm = l1*F(x9[0],xm)+m1*F(x9[1],ym)+n1*F(x9[2],zm);
fl_y9_ym = l2*F(x9[0],xm)+m2*F(x9[1],ym)+n2*F(x9[2],zm);
fl_z9_zm = l3*F(x9[0],xm)+m3*F(x9[1],ym)+n3*F(x9[2],zm);

fl_x10_xm = l1*F(x10[0],xm)+m1*F(x10[1],ym)+n1*F(x10[2],zm);
fl_y10_ym = l2*F(x10[0],xm)+m2*F(x10[1],ym)+n2*F(x10[2],zm);
fl_z10_zm = l3*F(x10[0],xm)+m3*F(x10[1],ym)+n3*F(x10[2],zm);

fl_x12_xm = l1*F(x12[0],xm)+m1*F(x12[1],ym)+n1*F(x12[2],zm);
```

```
fl_y12_ym = l2*F(x12[0],xm)+m2*F(x12[1],ym)+n2*F(x12[2],zm);
fl_z12_zm = l3*F(x12[0],xm)+m3*F(x12[1],ym)+n3*F(x12[2],zm);

fl_xtop_xm = l1*F(xtop,xm)+m1*F(ytop,ym)+n1*F(ztop,zm);
fl_ytop_ym = l2*F(xtop,xm)+m2*F(ytop,ym)+n2*F(ztop,zm);
fl_ztop_zm = l3*F(xtop,xm)+m3*F(ytop,ym)+n3*F(ztop,zm);

fl_xbot_xm = l1*F(xbot,xm)+m1*F(ybot,ym)+n1*F(zbot,zm);
fl_ybot_ym = l2*F(xbot,xm)+m2*F(ybot,ym)+n2*F(zbot,zm);
fl_zbot_zm = l3*F(xbot,xm)+m3*F(ybot,ym)+n3*F(zbot,zm);

// moment arms in the displaced frame

L_x1_phiz = -(l/2-joint_offset_1);
L_y1_phiz = -w/2;
L_x3_phiz =  (l/2-joint_offset_3);
L_y3_phiz = -w/2;

L_x4_phiz =  l/2;
L_y4_phiz = -(w/2-joint_offset_4);
L_x6_phiz =  l/2;
L_y6_phiz =  (w/2-joint_offset_6);

L_x7_phiz =  (l/2-joint_offset_7);
L_y7_phiz =  w/2;
L_x9_phiz = -(l/2-joint_offset_9);
L_y9_phiz =  w/2;

L_x10_phiz = -l/2;
L_y10_phiz =  (w/2-joint_offset_10);
L_x12_phiz = -l/2;
L_y12_phiz = -(w/2-joint_offset_12);

L_x2_phiz =  0;
L_y2_phiz = -w/2;
L_x8_phiz =  0;
L_y8_phiz =  w/2;

L_x5_phiz =  l/2;
L_y5_phiz =  0;
L_x11_phiz = -l/2;
L_y11_phiz =  0;

L_z1_phix =  (l/2-joint_offset_1);
L_z1_phiy =  w/2;

L_z3_phix = -(l/2-joint_offset_3);
L_z3_phiy =  w/2;

L_z4_phix = -l/2;
L_z4_phiy =  (w/2-joint_offset_4);

L_z6_phix = -l/2;
L_z6_phiy = -(w/2-joint_offset_6);

L_z7_phix = -(l/2-joint_offset_7);
L_z7_phiy = -w/2;

L_z9_phix =  (l/2-joint_offset_9);
L_z9_phiy = -w/2;

L_z10_phix =  l/2;
L_z10_phiy = -(w/2-joint_offset_10);

L_z12_phix =  l/2;
```

```
L_z12_phiy =  (w/2-joint_offset_12);

L_z2_phix =  0;
L_z2_phiy =  w/2;
L_z8_phix =  0;
L_z8_phiy = -w/2;

L_z5_phix = -l/2;
L_z5_phiy =  0;
L_z11_phix = l/2;
L_z11_phiy =  0;

// moments calculated in local frame

tql_phizm =    fl_x1_xm*(L_x1_phiz*euler_l1+L_y1_phiz*euler_m1)
+ fl_y1_ym*(L_x1_phiz*euler_l2+L_y1_phiz*euler_m2)
+ fl_z1_zm*(L_x1_phiz*euler_l3+L_y1_phiz*euler_m3)
  + fl_x3_xm*(L_x3_phiz*euler_l1+L_y3_phiz*euler_m1)
+ fl_y3_ym*(L_x3_phiz*euler_l2+L_y3_phiz*euler_m2)
+ fl_z3_zm*(L_x3_phiz*euler_l3+L_y3_phiz*euler_m3)
  + fl_x4_xm*(L_x4_phiz*euler_l1+L_y4_phiz*euler_m1)
+ fl_y4_ym*(L_x4_phiz*euler_l2+L_y4_phiz*euler_m2)
+ fl_z4_zm*(L_x4_phiz*euler_l3+L_y4_phiz*euler_m3)
  + fl_x6_xm*(L_x6_phiz*euler_l1+L_y6_phiz*euler_m1)
+ fl_y6_ym*(L_x6_phiz*euler_l2+L_y6_phiz*euler_m2)
+ fl_z6_zm*(L_x6_phiz*euler_l3+L_y6_phiz*euler_m3)
  + fl_x7_xm*(L_x7_phiz*euler_l1+L_y7_phiz*euler_m1)
+ fl_y7_ym*(L_x7_phiz*euler_l2+L_y7_phiz*euler_m2)
+ fl_z7_zm*(L_x7_phiz*euler_l3+L_y7_phiz*euler_m3)
  + fl_x9_xm*(L_x9_phiz*euler_l1+L_y9_phiz*euler_m1)
+ fl_y9_ym*(L_x9_phiz*euler_l2+L_y9_phiz*euler_m2)
+ fl_z9_zm*(L_x9_phiz*euler_l3+L_y9_phiz*euler_m3)
  + fl_x10_xm*(L_x10_phiz*euler_l1+L_y10_phiz*euler_m1)
+ fl_y10_ym*(L_x10_phiz*euler_l2+L_y10_phiz*euler_m2)
+ fl_z10_zm*(L_x10_phiz*euler_l3+L_y10_phiz*euler_m3)
  + fl_x12_xm*(L_x12_phiz*euler_l1+L_y12_phiz*euler_m1)
+ fl_y12_ym*(L_x12_phiz*euler_l2+L_y12_phiz*euler_m2)
+ fl_z12_zm*(L_x12_phiz*euler_l3+L_y12_phiz*euler_m3)
  + fl_x2_xm*(L_x2_phiz*euler_l1+L_y2_phiz*euler_m1)
+ fl_y2_ym*(L_x2_phiz*euler_l2+L_y2_phiz*euler_m2)
+ fl_z2_zm*(L_x2_phiz*euler_l3+L_y2_phiz*euler_m3)
  + fl_x8_xm*(L_x8_phiz*euler_l1+L_y8_phiz*euler_m1)
+ fl_y8_ym*(L_x8_phiz*euler_l2+L_y8_phiz*euler_m2)
+ fl_z8_zm*(L_x8_phiz*euler_l3+L_y8_phiz*euler_m3)
  + fl_x5_xm*(L_x5_phiz*euler_l1+L_y5_phiz*euler_m1)
+ fl_y5_ym*(L_x5_phiz*euler_l2+L_y5_phiz*euler_m2)
+ fl_z5_zm*(L_x5_phiz*euler_l3+L_y5_phiz*euler_m3)
  + fl_x11_xm*(L_x11_phiz*euler_l1+L_y11_phiz*euler_m1)
+ fl_y11_ym*(L_x11_phiz*euler_l2+L_y11_phiz*euler_m2)
+ fl_z11_zm*(L_x11_phiz*euler_l3+L_y11_phiz*euler_m3);


tql_phiym =    fl_x1_xm*(L_z1_phiy*euler_n1)
+ fl_y1_ym*(L_z1_phiy*euler_n2)
+ fl_z1_zm*(L_z1_phiy*euler_n3)
+ fl_x3_xm*(L_z3_phiy*euler_n1)
+ fl_y3_ym*(L_z3_phiy*euler_n2)
+ fl_z3_zm*(L_z3_phiy*euler_n3)
+ fl_x4_xm*(L_z4_phiy*euler_n1)
+ fl_y4_ym*(L_z4_phiy*euler_n2)
+ fl_z4_zm*(L_z4_phiy*euler_n3)
+ fl_x6_xm*(L_z6_phiy*euler_n1)
+ fl_y6_ym*(L_z6_phiy*euler_n2)
+ fl_z6_zm*(L_z6_phiy*euler_n3)
+ fl_x7_xm*(L_z7_phiy*euler_n1)
```

```
              + fl_y7_ym*(L_z7_phiy*euler_n2)
              + fl_z7_zm*(L_z7_phiy*euler_n3)
              + fl_x9_xm*(L_z9_phiy*euler_n1)
              + fl_y9_ym*(L_z9_phiy*euler_n2)
              + fl_z9_zm*(L_z9_phiy*euler_n3)
              + fl_x10_xm*(L_z10_phiy*euler_n1)
              + fl_y10_ym*(L_z10_phiy*euler_n2)
              + fl_z10_zm*(L_z10_phiy*euler_n3)
              + fl_x12_xm*(L_z12_phiy*euler_n1)
              + fl_y12_ym*(L_z12_phiy*euler_n2)
              + fl_z12_zm*(L_z12_phiy*euler_n3)
              + fl_x2_xm*(L_z2_phiy*euler_n1)
              + fl_y2_ym*(L_z2_phiy*euler_n2)
              + fl_z2_zm*(L_z2_phiy*euler_n3)
              + fl_x8_xm*(L_z8_phiy*euler_n1)
              + fl_y8_ym*(L_z8_phiy*euler_n2)
              + fl_z8_zm*(L_z8_phiy*euler_n3)
              + fl_x5_xm*(L_z5_phiy*euler_n1)
              + fl_y5_ym*(L_z5_phiy*euler_n2)
              + fl_z5_zm*(L_z5_phiy*euler_n3)
              + fl_x11_xm*(L_z11_phiy*euler_n1)
              + fl_y11_ym*(L_z11_phiy*euler_n2)
              + fl_z11_zm*(L_z11_phiy*euler_n3);


tql_phixm =    fl_x1_xm*(L_z1_phix*euler_n1)
              + fl_y1_ym*(L_z1_phix*euler_n2)
              + fl_z1_zm*(L_z1_phix*euler_n3)
              + fl_x3_xm*(L_z3_phix*euler_n1)
              + fl_y3_ym*(L_z3_phix*euler_n2)
              + fl_z3_zm*(L_z3_phix*euler_n3)
              + fl_x4_xm*(L_z4_phix*euler_n1)
              + fl_y4_ym*(L_z4_phix*euler_n2)
              + fl_z4_zm*(L_z4_phix*euler_n3)
              + fl_x6_xm*(L_z6_phix*euler_n1)
              + fl_y6_ym*(L_z6_phix*euler_n2)
              + fl_z6_zm*(L_z6_phix*euler_n3)
              + fl_x7_xm*(L_z7_phix*euler_n1)
              + fl_y7_ym*(L_z7_phix*euler_n2)
              + fl_z7_zm*(L_z7_phix*euler_n3)
              + fl_x9_xm*(L_z9_phix*euler_n1)
              + fl_y9_ym*(L_z9_phix*euler_n2)
              + fl_z9_zm*(L_z9_phix*euler_n3)
              + fl_x10_xm*(L_z10_phix*euler_n1)
              + fl_y10_ym*(L_z10_phix*euler_n2)
              + fl_z10_zm*(L_z10_phix*euler_n3)
              + fl_x12_xm*(L_z12_phix*euler_n1)
              + fl_y12_ym*(L_z12_phix*euler_n2)
              + fl_z12_zm*(L_z12_phix*euler_n3)
              + fl_x2_xm*(L_z2_phix*euler_n1)
              + fl_y2_ym*(L_z2_phix*euler_n2)
              + fl_z2_zm*(L_z2_phix*euler_n3)
              + fl_x8_xm*(L_z8_phix*euler_n1)
              + fl_y8_ym*(L_z8_phix*euler_n2)
              + fl_z8_zm*(L_z8_phix*euler_n3)
              + fl_x5_xm*(L_z5_phix*euler_n1)
              + fl_y5_ym*(L_z5_phix*euler_n2)
              + fl_z5_zm*(L_z5_phix*euler_n3)
              + fl_x11_xm*(L_z11_phix*euler_n1)
              + fl_y11_ym*(L_z11_phix*euler_n2)
              + fl_z11_zm*(L_z11_phix*euler_n3);

// moments transfered from local frame back to chip frame
tqchip_phixm = inv_l1*tql_phixm + inv_m1*tql_phiym + inv_n1*tql_phizm;
tqchip_phiym = inv_l2*tql_phixm + inv_m2*tql_phiym + inv_n2*tql_phizm;
```

```
tqchip_phizm = inv_l3*tql_phixm + inv_m3*tql_phiym + inv_n3*tql_phizm;

// moments applied to the internal nodes for the middle point of plate
Tau(phizm) <+ -tqchip_phizm;
Tau(phiym) <+ -tqchip_phiym;
Tau(phixm) <+ -tqchip_phixm;

// damping moments applied to the internal nodes for the middle point of plate
Tau(phixm) <+ - (dampphix * 1e6*ddt(Theta(phixm)));
Tau(phiym) <+ - (dampphiy * 1e6*ddt(Theta(phiym)));
Tau(phizm) <+ - (dampphiz * 1e6*ddt(Theta(phizm)));

// rigid body constraints on displacements, transformed from local to chip frame

Pos(x1[0],xm) <+ (inv_l1*l_x1_xm + inv_m1*l_y1_ym + inv_n1*l_z1_zm);
Pos(x1[1],ym) <+ (inv_l2*l_x1_xm + inv_m2*l_y1_ym + inv_n2*l_z1_zm);
Pos(x1[2],zm) <+ (inv_l3*l_x1_xm + inv_m3*l_y1_ym + inv_n3*l_z1_zm);

Pos(x2[0],xm) <+ (inv_l1*l_x2_xm + inv_m1*l_y2_ym + inv_n1*l_z2_zm);
Pos(x2[1],ym) <+ (inv_l2*l_x2_xm + inv_m2*l_y2_ym + inv_n2*l_z2_zm);
Pos(x2[2],zm) <+ (inv_l3*l_x2_xm + inv_m3*l_y2_ym + inv_n3*l_z2_zm);

Pos(x3[0],xm) <+ (inv_l1*l_x3_xm + inv_m1*l_y3_ym + inv_n1*l_z3_zm);
Pos(x3[1],ym) <+ (inv_l2*l_x3_xm + inv_m2*l_y3_ym + inv_n2*l_z3_zm);
Pos(x3[2],zm) <+ (inv_l3*l_x3_xm + inv_m3*l_y3_ym + inv_n3*l_z3_zm);

Pos(x4[0],xm) <+ (inv_l1*l_x4_xm + inv_m1*l_y4_ym + inv_n1*l_z4_zm);
Pos(x4[1],ym) <+ (inv_l2*l_x4_xm + inv_m2*l_y4_ym + inv_n2*l_z4_zm);
Pos(x4[2],zm) <+ (inv_l3*l_x4_xm + inv_m3*l_y4_ym + inv_n3*l_z4_zm);

Pos(x5[0],xm) <+ (inv_l1*l_x5_xm + inv_m1*l_y5_ym + inv_n1*l_z5_zm);
Pos(x5[1],ym) <+ (inv_l2*l_x5_xm + inv_m2*l_y5_ym + inv_n2*l_z5_zm);
Pos(x5[2],zm) <+ (inv_l3*l_x5_xm + inv_m3*l_y5_ym + inv_n3*l_z5_zm);

Pos(x6[0],xm) <+ (inv_l1*l_x6_xm + inv_m1*l_y6_ym + inv_n1*l_z6_zm);
Pos(x6[1],ym) <+ (inv_l2*l_x6_xm + inv_m2*l_y6_ym + inv_n2*l_z6_zm);
Pos(x6[2],zm) <+ (inv_l3*l_x6_xm + inv_m3*l_y6_ym + inv_n3*l_z6_zm);

Pos(x7[0],xm) <+ (inv_l1*l_x7_xm + inv_m1*l_y7_ym + inv_n1*l_z7_zm);
Pos(x7[1],ym) <+ (inv_l2*l_x7_xm + inv_m2*l_y7_ym + inv_n2*l_z7_zm);
Pos(x7[2],zm) <+ (inv_l3*l_x7_xm + inv_m3*l_y7_ym + inv_n3*l_z7_zm);

Pos(x8[0],xm) <+ (inv_l1*l_x8_xm + inv_m1*l_y8_ym + inv_n1*l_z8_zm);
Pos(x8[1],ym) <+ (inv_l2*l_x8_xm + inv_m2*l_y8_ym + inv_n2*l_z8_zm);
Pos(x8[2],zm) <+ (inv_l3*l_x8_xm + inv_m3*l_y8_ym + inv_n3*l_z8_zm);

Pos(x9[0],xm) <+ (inv_l1*l_x9_xm + inv_m1*l_y9_ym + inv_n1*l_z9_zm);
Pos(x9[1],ym) <+ (inv_l2*l_x9_xm + inv_m2*l_y9_ym + inv_n2*l_z9_zm);
Pos(x9[2],zm) <+ (inv_l3*l_x9_xm + inv_m3*l_y9_ym + inv_n3*l_z9_zm);

Pos(x10[0],xm) <+ (inv_l1*l_x10_xm + inv_m1*l_y10_ym + inv_n1*l_z10_zm);
Pos(x10[1],ym) <+ (inv_l2*l_x10_xm + inv_m2*l_y10_ym + inv_n2*l_z10_zm);
Pos(x10[2],zm) <+ (inv_l3*l_x10_xm + inv_m3*l_y10_ym + inv_n3*l_z10_zm);

Pos(x11[0],xm) <+ (inv_l1*l_x11_xm + inv_m1*l_y11_ym + inv_n1*l_z11_zm);
Pos(x11[1],ym) <+ (inv_l2*l_x11_xm + inv_m2*l_y11_ym + inv_n2*l_z11_zm);
Pos(x11[2],zm) <+ (inv_l3*l_x11_xm + inv_m3*l_y11_ym + inv_n3*l_z11_zm);

Pos(x12[0],xm) <+ (inv_l1*l_x12_xm + inv_m1*l_y12_ym + inv_n1*l_z12_zm);
Pos(x12[1],ym) <+ (inv_l2*l_x12_xm + inv_m2*l_y12_ym + inv_n2*l_z12_zm);
Pos(x12[2],zm) <+ (inv_l3*l_x12_xm + inv_m3*l_y12_ym + inv_n3*l_z12_zm);

Pos(xtop,xm) <+ (inv_l1*l_xtop_xm + inv_m1*l_ytop_ym + inv_n1*l_ztop_zm);
Pos(ytop,ym) <+ (inv_l2*l_xtop_xm + inv_m2*l_ytop_ym + inv_n2*l_ztop_zm);
Pos(ztop,zm) <+ (inv_l3*l_xtop_xm + inv_m3*l_ytop_ym + inv_n3*l_ztop_zm);
```

Modeling and Simulation for Design of Suspended MEMS

```
Pos(xbot,xm) <+ (inv_l1*l_xbot_xm + inv_m1*l_ybot_ym + inv_n1*l_zbot_zm);
Pos(ybot,ym) <+ (inv_l2*l_xbot_xm + inv_m2*l_ybot_ym + inv_n2*l_zbot_zm);
Pos(zbot,zm) <+ (inv_l3*l_xbot_xm + inv_m3*l_ybot_ym + inv_n3*l_zbot_zm);

// rigid body constraints on angular displacements, transformed from local to chip frame

Theta(phi[2],phizm) <+ 0 ;
Theta(phi[1],phiym) <+ 0 ;
Theta(phi[0],phixm) <+ 0 ;

Theta(phixtop,phixm) <+ 0 ;
Theta(phiytop,phiym) <+ 0 ;
Theta(phiztop,phizm) <+ 0 ;

Theta(phixbot,phixm) <+ 0 ;
Theta(phiybot,phiym) <+ 0 ;
Theta(phizbot,phizm) <+ 0 ;

//*******Coriolis force and centrifugal force******

    r[0] = Xc + X_origin + Pos(xm);
    // Xc is a constant for the given chip, X_origin will in general be time varying
    // because in general a car will be taking turns (i.e., rotating) about different axes
    // But for gyro simulation purposes, X_origin can be assumed constant.

    r[1] = Yc + Y_origin + Pos(ym);
    // Like, X_origin, Y_origin is again time varying in general but for simulation purposes
    // can be set to be a constant
    r[2] = Zc + Z_origin + Pos(zm);

    Pos(vel[0]) <+ ddt(Pos(xm));
    Pos(vel[1]) <+ ddt(Pos(ym));
    Pos(vel[2]) <+ ddt(Pos(zm));

    OMGr[0] = crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), r[0], r[1], r[2], 0);
    OMGr[1] = crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), r[0], r[1], r[2], 1);
    OMGr[2] = crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), r[0], r[1], r[2], 2);

    Theta(dphidt[0]) <+ ddt(Theta(phixm));
    Theta(dphidt[1]) <+ ddt(Theta(phiym));
    Theta(dphidt[2]) <+ ddt(Theta(phizm));

    Theta(dOMGdt[0]) <+ ddt(Theta(OMG_ext[0]));
    Theta(dOMGdt[1]) <+ ddt(Theta(OMG_ext[1]));
    Theta(dOMGdt[2]) <+ ddt(Theta(OMG_ext[2]));

    Theta(THT[0]) <+ idt(Theta(OMG_ext[0]),0);
    Theta(THT[1]) <+ idt(Theta(OMG_ext[1]),0);
    Theta(THT[2]) <+ idt(Theta(OMG_ext[2]),0);
cos_tht0 = cos(Theta(THT[0]));
cos_tht1 = cos(Theta(THT[1]));
cos_tht2 = cos(Theta(THT[2]));
sin_tht0 = sin(Theta(THT[0]));
sin_tht1 = sin(Theta(THT[1]));
sin_tht2 = sin(Theta(THT[2]));

tht_l1 =  cos_tht1*cos_tht2;
tht_m1 =  cos_tht1*sin_tht2;
tht_n1 = -sin_tht1;
tht_l2 =  sin_tht0*sin_tht1*cos_tht2-cos_tht0*sin_tht2;
tht_m2 =  sin_tht0*sin_tht1*sin_tht2+cos_tht0*cos_tht2;
tht_n2 =  sin_tht0*cos_tht1;
tht_l3 =  cos_tht0*sin_tht1*cos_tht2+sin_tht0*sin_tht2;
```

Modeling and Simulation for Design of Suspended MEMS   292

```
tht_m3 =  cos_tht0*sin_tht1*sin_tht2-sin_tht0*cos_tht2;
tht_n3 =  cos_tht0*cos_tht1;

ag[0] = tht_l1*Pos(ax_ext[0]) + tht_m1*Pos(ax_ext[1]) + tht_n1*Pos(ax_ext[2]);
ag[1] = tht_l2*Pos(ax_ext[0]) + tht_m2*Pos(ax_ext[1]) + tht_n2*Pos(ax_ext[2]);
ag[2] = tht_l3*Pos(ax_ext[0]) + tht_m3*Pos(ax_ext[1]) + tht_n3*Pos(ax_ext[2]);

// -------------- Following lines only for storing and display --------------//
    acc[0] = ag[0] + 2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]),
Pos(vel[2]), 0) +
        crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 0) +
        crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), OMGr[0], OMGr[1], OMGr[2], 0);

    acc[1] = ag[1] + 2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]),
Pos(vel[2]), 1) +
        crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 1) +
        crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), OMGr[0], OMGr[1], OMGr[2], 1);

    acc[2] = ag[2] + 2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]),
Pos(vel[2]), 2) +
        crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 2) +
        crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), OMGr[0], OMGr[1], OMGr[2], 2);

// ----------------- End of storing and display lines ------------------------//

    F(xm) <+ -ms*(ddt(Pos(vel[0])) + ag[0] +
    2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]), Pos(vel[2]), 0) +
    crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 0));
    F(ym) <+ -ms*(ddt(Pos(vel[1])) + ag[1] +
    2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]), Pos(vel[2]), 1) +
    crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 1));
    F(zm) <+ -ms*(ddt(Pos(vel[2])) + ag[2] +
    2*crossprod(Theta(OMG_ext[0]), Theta(OMG_ext[1]), Theta(OMG_ext[2]), Pos(vel[0]), Pos(vel[1]), Pos(vel[2]), 2) +
    crossprod(Theta(dOMGdt[0]), Theta(dOMGdt[1]), Theta(dOMGdt[2]), r[0], r[1], r[2], 2));

    Tau(phixm) <+ -(iix*(Theta(dOMGdt[0]) + ddt(Theta(dphidt[0]))) - (iiy-iiz)*Theta(OMG_ext[1])*Theta(OMG_ext[2]));
    Tau(phiym) <+ -(iiy*(Theta(dOMGdt[1]) + ddt(Theta(dphidt[1]))) - (iiz-iix)*Theta(OMG_ext[2])*Theta(OMG_ext[0]));
    Tau(phizm) <+ -(iiz*(Theta(dOMGdt[2]) + ddt(Theta(dphidt[2]))) - (iix-iiy)*Theta(OMG_ext[0])*Theta(OMG_ext[1]));

//*******end of Coriolis force and centrifugal force

// electrical equations

I(v5, v_mid) <+ V(v5, v_mid)/resistance;
I(v11, v_mid) <+ V(v11, v_mid)/resistance;
I(v2, v_mid) <+ V(v2, v_mid)/resistance;
I(v8, v_mid) <+ V(v8, v_mid)/resistance;

I(v1, v_mid) <+ V(v1, v_mid)/resistance;
I(v3, v_mid) <+ V(v3, v_mid)/resistance;
I(v4, v_mid) <+ V(v4, v_mid)/resistance;
I(v6, v_mid) <+ V(v6, v_mid)/resistance;
I(v7, v_mid) <+ V(v7, v_mid)/resistance;
I(v9, v_mid) <+ V(v9, v_mid)/resistance;
I(v10, v_mid) <+ V(v10, v_mid)/resistance;
I(v12, v_mid) <+ V(v12, v_mid)/resistance;

end
endmodule
```

Modeling and Simulation for Design of Suspended MEMS 293

A.8 Elastic Plate Model

The elastic plate model, as discussed in Chapter 3.4.2, is given in this section. It captures the in-plane stretching and the out-of-plane bending of rectangular plate elements.

The Verilog-A code for this model is:

```
// include header files
'include "../../constants.h"
'include "../../discipline.h"
'include "../../process.h"
'include "../../design.h"

module plate3D_elastic(x1, phi1, v1, x2, phi2, v2, x3, phi3, v3, x4, phi4, v4, t);

inout [0:2] x1;
kinematic [0:2] x1;
inout [0:2] phi1;
rotational [0:2] phi1;
inout v1;
electrical v1;

inout [0:2] x2;
kinematic [0:2] x2;
inout [0:2] phi2;
rotational [0:2] phi2;
inout v2;
electrical v2;

inout [0:2] x3;
kinematic [0:2] x3;
inout [0:2] phi3;
rotational [0:2] phi3;
inout v3;
electrical v3;

inout [0:2] x4;
kinematic [0:2] x4;
inout [0:2] phi4;
rotational [0:2] phi4;
inout v4;
electrical v4;

inout t;
thermal t;

parameter real unitl = 'default_plate_unitl;
parameter real unitw = 'default_plate_unitw;
parameter real unitnum_x = 'default_plate_unitnum_x;
parameter real unitnum_y = 'default_plate_unitnum_y;

parameter real alpha = 0;
parameter real beta  = 0;
parameter real gamma  = 0;

parameter real E = 'default_E;
parameter real thickness = 'default_thickness;
parameter real density = 'default_density;

parameter real Xc = 'default_plate_Xc;
```

```
parameter real Yc = 'default_plate_Yc;

parameter real resistivity = 'default_resistivity;
parameter real air_gap = 'default_air_gap;
parameter real visc_air = 'default_visc_air;
parameter real bloat = 4u;

electrical vmid;
kinematic Vx1,Vy1,Vz1, Vx2,Vy2,Vz2, Vx3,Vy3,Vz3, Vx4,Vy4,Vz4;
rotational_omega Vphix1,Vphiy1,Vphiz1, Vphix2,Vphiy2,Vphiz2;
rotational_omega Vphix3,Vphiy3,Vphiz3, Vphix4,Vphiy4,Vphiz4;

real r;
real dampx,dampy,dampz, dampphix,dampphiy,dampphiz;

real cos_alpha, cos_beta, cos_gamma, sin_alpha, sin_beta, sin_gamma;
real l1, m1, n1 ;
real l2, m2, n2 ;
real l3, m3, n3 ;
real inv_l1, inv_m1, inv_n1 ;
real inv_l2, inv_m2, inv_n2 ;
real inv_l3, inv_m3, inv_n3 ;

real l_x1,l_y1,l_z1, l_x2,l_y2,l_z2, l_x3,l_y3,l_z3, l_x4,l_y4,l_z4;
real l_phix1,l_phiy1,l_phiz1, l_phix2,l_phiy2,l_phiz2;
real l_phix3,l_phiy3,l_phiz3, l_phix4,l_phiy4,l_phiz4;
real Fx1,Fy1,Fz1, Fx2,Fy2,Fz2, Fx3,Fy3,Fz3, Fx4,Fy4,Fz4;
real Tqx1,Tqy1,Tqx2,Tqy2,Tqx3,Tqy3,Tqx4,Tqy4;
real fi_x1,fi_y1,fi_z1, fi_x2,fi_y2,fi_z2;
real fi_x3,fi_y3,fi_z3, fi_x4,fi_y4,fi_z4;
real tqi_x1,tqi_y1,tqi_x2,tqi_y2,tqi_x3,tqi_y3,tqi_x4,tqi_y4;
real Fchipx1,Fchipy1,Fchipz1, Fchipx2,Fchipy2,Fchipz2;
real Fchipx3,Fchipy3,Fchipz3, Fchipx4,Fchipy4,Fchipz4;
real Fichipx1,Fichipy1,Fichipz1, Fichipx2,Fichipy2,Fichipz2;
real Fichipx3,Fichipy3,Fichipz3, Fichipx4,Fichipy4,Fichipz4;
real Tqchipx1,Tqchipy1,Tqchipx2,Tqchipy2;
real Tqchipx3,Tqchipy3,Tqchipx4,Tqchipy4;
real Tqichipx1,Tqichipy1,Tqichipx2,Tqichipy2;
real Tqichipx3,Tqichipy3,Tqichipx4,Tqichipy4;

// coefficients of stiffness matrix for out-of-plane bending

real lwratio, nu,a,b,pp2,nn2,kb0;

real kb11;
real kb21, kb22;
real kb31, kb32, kb33;
real kb41, kb42, kb43, kb44;
real kb51, kb52, kb53, kb54, kb55;
real kb61, kb62, kb63, kb64, kb65, kb66;
real kb71, kb72, kb73, kb74, kb75, kb76, kb77;
real kb81, kb82, kb83, kb84, kb85, kb86, kb87, kb88;
real kb91, kb92, kb93, kb94, kb95, kb96, kb97, kb98, kb99;
real kb101,kb102,kb103,kb104,kb105,kb106,kb107,kb108,kb109,kb1010;
real kb111,kb112,kb113,kb114,kb115,kb116,kb117,kb118,kb119,kb1110,kb1111;
real kb121,kb122,kb123,kb124,kb125,kb126,kb127,kb128,kb129,kb1210,kb1211,kb1212;

// coefficients of mass matrix for out-of-plane bending

real mb0;

real mb11;
real mb21, mb22;
real mb31, mb32, mb33;
real mb41, mb42, mb43, mb44;
```

```
real mb51, mb52, mb53, mb54, mb55;
real mb61, mb62, mb63, mb64, mb65, mb66;
real mb71, mb72, mb73, mb74, mb75, mb76, mb77;
real mb81, mb82, mb83, mb84, mb85, mb86, mb87, mb88;
real mb91, mb92, mb93, mb94, mb95, mb96, mb97, mb98, mb99;
real mb101,mb102,mb103,mb104,mb105,mb106,mb107,mb108,mb109,mb1010;
real mb111,mb112,mb113,mb114,mb115,mb116,mb117,mb118,mb119,mb1110,mb1111;
real mb121,mb122,mb123,mb124,mb125,mb126,mb127,mb128,mb129,mb1210,mb1211,mb1212;

// coefficients of stiffness matrix for in-plane stretching

real pp1, nn1, ks0;

real ks11;
real ks21,ks22;
real ks31,ks32,ks33;
real ks41,ks42,ks43,ks44;
real ks51,ks52,ks53,ks54,ks55;
real ks61,ks62,ks63,ks64,ks65,ks66;
real ks71,ks72,ks73,ks74,ks75,ks76,ks77;
real ks81,ks82,ks83,ks84,ks85,ks86,ks87,ks88;

// coefficients of mass matrix for in-plane stretching

real ms0;

real ms11;
real ms21,ms22;
real ms31,ms32,ms33;
real ms41,ms42,ms43,ms44;
real ms51,ms52,ms53,ms54,ms55;
real ms61,ms62,ms63,ms64,ms65,ms66;
real ms71,ms72,ms73,ms74,ms75,ms76,ms77;
real ms81,ms82,ms83,ms84,ms85,ms86,ms87,ms88;

// internal nodes and variables for twisting about z-axis
rotational phim;
rotational_omega  angvmid;
real ii, angmid, rad, lngth_r, arc_r, phi_off_1_r, phi_off_2_r;

analog begin

b = unitl*unitnum_y;
a = unitw*unitnum_x;

r = resistivity * (b/a);

dampx = visc_air/air_gap*((a+bloat)*(b+bloat));
dampy = visc_air/air_gap*((a+bloat)*(b+bloat));
dampz = visc_air/air_gap*((a+bloat)+(b+bloat))*2.0*thickness;
dampphix = visc_air/air_gap*(b*b+thickness*thickness)*2.0/12.0;
dampphiy = visc_air/air_gap*(a*a+thickness*thickness)*2.0/12.0;
dampphiz = visc_air/air_gap*(a*a+b*b)/12.0;

// [kb], stiffness matrix for out-of-plane bending

////nu = Possion_ratio;
nu = 0.3;
lwratio = b/a;
pp2 = pow(lwratio,2);
nn2 = pow(lwratio,-2);
kb0 = E*thickness*thickness*thickness/(12.0*(1-nu*nu)*a*b);

kb11 = kb0*(156.0/35.0*(pp2+nn2)+72.0/25.0);
kb21 = kb0*( b*( 22.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
```

```
kb31 = kb0*(-a*( 78.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb41 = kb0*(54.0/35.0*pp2-156.0/35.0*nn2-72.0/25.0);
kb51 = kb0*( b*(-13.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0));
kb61 = kb0*( a*(-27.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));

kb22 = kb0*( b*b*( 4.0/35.0*pp2+52.0/35.0*nn2+8.0/25.0));
kb32 = kb0*(-a*b*( 11.0/35.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+60.0*nu)));
kb42 = kb0*(   b*( 13.0/35.0*pp2-78.0/35.0*nn2-6.0/25.0));
kb52 = kb0*( b*b*(- 3.0/35.0*pp2+26.0/35.0*nn2-2.0/25.0));
kb62 = kb0*( a*b*(-13.0/70.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+5.0*nu)));

kb33 = kb0*(a*a*( 52.0/35.0*pp2+ 4.0/35.0*nn2+8.0/25.0));
kb43 = kb0*(  a*(-27.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb53 = kb0*(a*b*( 13.0/70.0*pp2-11.0/35.0*nn2-1.0/50.0*(1.0+5.0*nu)));
kb63 = kb0*(a*a*( 18.0/35.0*pp2- 4.0/35.0*nn2-8.0/25.0));

kb44 = kb0*(156.0/35.0*pp2+156.0/35.0*nn2+72.0/25.0);
kb54 = kb0*(-b*(22.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb64 = kb0*(-a*(78.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));

kb55 = kb0*(b*b*( 4.0/35.0*pp2+52.0/35.0*nn2+8.0/25.0));
kb65 = kb0*(a*b*(11.0/35.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+60.0*nu)));

kb66 = kb0*(a*a*(52.0/35.0*pp2+4.0/35.0*nn2+8.0/25.0));

////

kb71  =  kb0*(- 54.0/35.0*pp2-54.0/35.0*nn2+72.0/25.0);
kb81  =  kb0*(b*( 13.0/35.0*pp2+27.0/35.0*nn2-6.0/25.0));
kb91  =  kb0*(a*(-27.0/35.0*pp2-13.0/35.0*nn2+6.0/25.0));
kb101 = kb0*(-156.0/35.0*pp2+54.0/35.0*nn2-72.0/25.0);
kb111 = kb0*(b*(-22.0/35.0*pp2+27.0/35.0*nn2-6.0/25.0*(1.0+5.0*nu)));
kb121 = kb0*(a*(-78.0/35.0*pp2+13.0/35.0*nn2-6.0/25.0));

kb72  =  kb0*(  b*(-13.0/35.0*pp2-27.0/35.0*nn2+6.0/25.0));
kb82  =  kb0*(b*b*( 3.0/35.0*pp2+ 9.0/35.0*nn2+2.0/25.0));
kb92  =  kb0*(a*b*(-13.0/70.0*pp2-13.0/70.0*nn2+1.0/50.0));
kb102 = kb0*(  b*(-22.0/35.0*pp2+27.0/35.0*nn2-6.0/25.0*(1.0+5.0*nu)));
kb112 = kb0*(b*b*(- 4.0/35.0*pp2+18.0/35.0*nn2-8.0/25.0));
kb122 = kb0*(a*b*(-11.0/35.0*pp2+13.0/70.0*nn2-1.0/50.0*(1.0+5.0*nu)));

kb73  =  kb0*(a*(27.0/35.0*pp2+13.0/35.0*nn2-6.0/25.0));
kb83  =  kb0*(a*b*(-13.0/70.0*pp2-13.0/70.0*nn2+1.0/50.0));
kb93  =  kb0*(a*a*( 9.0/35.0*pp2+ 3.0/35.0*nn2+2.0/25.0));
kb103 = kb0*(a*(78.0/35.0*pp2-13.0/35.0*nn2+6.0/25.0));
kb113 = kb0*(a*b*( 11.0/35.0*pp2-13.0/70.0*nn2+1.0/50.0*(1.0+5.0*nu)));
kb123 = kb0*(a*a*( 26.0/35.0*pp2- 3.0/35.0*nn2-2.0/25.0));

kb74  =  kb0*(-156.0/35.0*pp2+54.0/35.0*nn2-72.0/25.0);
kb84  =  kb0*(b*( 22.0/35.0*pp2-27.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb94  =  kb0*(a*(-78.0/35.0*pp2+13.0/35.0*nn2-6.0/25.0));
kb104 = kb0*(- 54.0/35.0*pp2-54.0/35.0*nn2+72.0/25.0);
kb114 = kb0*(b*(-13.0/35.0*pp2-27.0/35.0*nn2+6.0/25.0));
kb124 = kb0*(a*(-27.0/35.0*pp2-13.0/35.0*nn2+6.0/25.0));

kb75  =  kb0*(  b*(22.0/35.0*pp2-27.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb85  =  kb0*(b*b*(-4.0/35.0*pp2+18.0/35.0*nn2-8.0/25.0));
kb95  =  kb0*(a*b*(11.0/35.0*pp2-13.0/70.0*nn2+1.0/50.0*(1.0+5.0*nu)));
kb105 = kb0*(  b*(13.0/35.0*pp2+27.0/35.0*nn2-6.0/25.0));
kb115 = kb0*(b*b*( 3.0/35.0*pp2+ 9.0/35.0*nn2+2.0/25.0));
kb125 = kb0*(a*b*(13.0/70.0*pp2+13.0/70.0*nn2-1.0/50.0));

////

kb76  =  kb0*(  a*( 78.0/35.0*pp2-13.0/35.0*nn2+6.0/25.0));
```

```
kb86 = kb0*(a*b*(-11.0/35.0*pp2+13.0/70.0*nn2-1.0/50.0*(1.0+5.0*nu)));
kb96 = kb0*(a*a*( 26.0/35.0*pp2- 3.0/35.0*nn2-2.0/25.0));
kb106 = kb0*(  a*( 27.0/35.0*pp2+13.0/35.0*nn2-6.0/25.0));
kb116 = kb0*(a*b*( 13.0/70.0*pp2+13.0/70.0*nn2-1.0/50.0));
kb126 = kb0*(a*a*(  9.0/35.0*pp2+ 3.0/35.0*nn2+2.0/25.0));

kb77 = kb0*(156.0/35.0*pp2+156.0/35.0*nn2+72.0/25.0);
kb87 = kb0*(-b*(22.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb97 = kb0*( a*(78.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb107 = kb0*( 54.0/35.0*pp2-156.0/35.0*nn2-72.0/25.0);
kb117 = kb0*( b*(13.0/35.0*pp2-78.0/35.0*nn2-6.0/25.0));
kb127 = kb0*( a*(27.0/35.0*pp2-22.0/35.0*nn2-6.0/25.0*(1.0+5.0*nu)));

kb88 = kb0*( b*b*(  4.0/35.0*pp2+52.0/35.0*nn2+8.0/25.0));
kb98 = kb0*(-a*b*( 11.0/35.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+60.0*nu)));
kb108 = kb0*(   b*(-13.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0));
kb118 = kb0*( b*b*(- 3.0/35.0*pp2+26.0/35.0*nn2-2.0/25.0));
kb128 = kb0*( a*b*(-13.0/70.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+5.0*nu)));

kb99 = kb0*(a*a*(52.0/35.0*pp2+ 4.0/35.0*nn2+8.0/25.0));
kb109 = kb0*(  a*(27.0/35.0*pp2-22.0/35.0*nn2-6.0/25.0*(1.0+5.0*nu)));
kb119 = kb0*(a*b*(13.0/70.0*pp2-11.0/35.0*nn2-1.0/50.0*(1.0+5.0*nu)));
kb129 = kb0*(a*a*(18.0/35.0*pp2- 4.0/35.0*nn2-8.0/25.0));

kb1010 = kb0*(156.0/35.0*pp2+156.0/35.0*nn2+72.0/25.0);
kb1110 = kb0*(b*(22.0/35.0*pp2+78.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));
kb1210 = kb0*(a*(78.0/35.0*pp2+22.0/35.0*nn2+6.0/25.0*(1.0+5.0*nu)));

kb1111 = kb0*(b*b*( 4.0/35.0*pp2+52.0/35.0*nn2+8.0/25.0));
kb1211 = kb0*(a*b*(11.0/35.0*pp2+11.0/35.0*nn2+1.0/50.0*(1.0+60.0*nu)));

kb1212 = kb0*(a*a*(52.0/35.0*pp2+4.0/35.0*nn2+8.0/25.0));

// [mb], mass matrix for out-of-plane bending

mb0 = density*a*b*thickness;

mb11  =  24336.0/176400*mb0;
mb21  =  3432.0*b/176400.0*mb0;
mb31  = -3432.0*a/176400.0*mb0;
mb41  =  8424.0/176400.0*mb0;
mb51  = -2028.0*b/176400.0*mb0;
mb61  = -1188.0*a/176400.0*mb0;
mb71  =  2916.0/176400.0*mb0;
mb81  = - 702.0*b/176400.0*mb0;
mb91  =  702.0*a/176400.0*mb0;
mb101 =  8424.0/176400.0*mb0;
mb111 =  1188.0*b/176400.0*mb0;
mb121 =  2028.0*a/176400.0*mb0;

mb22  =  624.0*b*b/176400.0*mb0;
mb32  = - 484.0*a*b/176400.0*mb0;
mb42  =  2028.0*b/176400.0*mb0;
mb52  = - 468.0*b*b/176400.0*mb0;
mb62  = - 286.0*a*b/176400.0*mb0;
mb72  =  702.0*b/176400.0*mb0;
mb82  = - 162.0*b*b/176400.0*mb0;
mb92  =  169.0*a*b/176400.0*mb0;
mb102 =  1188.0*b/176400.0*mb0;
mb112 =  216.0*b*b/176400.0*mb0;
mb122 =  286.0*a*b/176400.0*mb0;

mb33  =  624.0*a*a/176400.0*mb0;
mb43  = -1188.0*a/176400.0*mb0;
mb53  =  286.0*a*b/176400.0*mb0;
```

```
mb63  =  216.0*a*a/176400.0*mb0;
mb73  = - 702.0*a/176400.0*mb0;
mb83  =  169.0*a*b/176400.0*mb0;
mb93  = - 162.0*a*a/176400.0*mb0;
mb103 = -2028.0*a/176400.0*mb0;
mb113 = - 286.0*a*b/176400.0*mb0;
mb123 = - 468.0*a*a/176400.0*mb0;

mb44  =  24336.0/176400.0*mb0;
mb54  = -3432.0*b/176400.0*mb0;
mb64  = -3432.0*a/176400.0*mb0;
mb74  =  8424.0/176400.0*mb0;
mb84  = -1188.0*b/176400.0*mb0;
mb94  =  2028.0*a/176400.0*mb0;
mb104 =  2916.0/176400.0*mb0;
mb114 =  702.0*b/176400.0*mb0;
mb124 =  702.0*a/176400.0*mb0;

mb55  =  624.0*b*b/176400.0*mb0;
mb65  =  484.0*a*b/176400.0*mb0;
mb75  = -1188.0*b/176400.0*mb0;
mb85  =  216.0*b*b/176400.0*mb0;
mb95  = - 286.0*a*b/176400.0*mb0;
mb105 = - 702.0*b/176400.0*mb0;
mb115 = - 162.0*b*b/176400.0*mb0;
mb125 = - 169.0*a*b/176400.0*mb0;

mb66  =  624.0*a*a/176400.0*mb0;
mb76  = -2028.0*a/176400.0*mb0;
mb86  =  286.0*a*b/176400.0*mb0;
mb96  = - 468.0*a*a/176400.0*mb0;
mb106 = - 702.0*a/176400.0*mb0;
mb116 = - 169.0*a*b/176400.0*mb0;
mb126 = - 162.0*a*a/176400.0*mb0;

mb77  =  24336.0/176400.0*mb0;
mb87  = -3432.0*b/176400.0*mb0;
mb97  =  3432.0*a/176400.0*mb0;
mb107 =  8424.0/176400.0*mb0;
mb117 =  2028.0*b/176400.0*mb0;
mb127 =  1188.0*a/176400.0*mb0;

mb88  =  624.0*b*b/176400.0*mb0;
mb98  = - 484.0*a*b/176400.0*mb0;
mb108 = -2028.0*b/176400.0*mb0;
mb118 = - 468.0*b*b/176400.0*mb0;
mb128 = - 286.0*a*b/176400.0*mb0;

mb99  =  624.0*a*a/176400.0*mb0;
mb109 =  1188.0*a/176400.0*mb0;
mb119 =  286.0*a*b/176400.0*mb0;
mb129 =  216.0*a*a/176400.0*mb0;

mb1010 = 24336.0/176400.0*mb0;
mb1110 = 3432.0*b/176400.0*mb0;
mb1210 = 3432.0*a/176400.0*mb0;

mb1111 = 624.0*b*b/176400.0*mb0;
mb1211 = 484.0*a*b/176400.0*mb0;

mb1212 = 624.0*a*a/176400.0*mb0;

// [ks], stiffness matrix for out-of-plane bending

pp1 = pow(lwratio,1);
```

```
nn1 = pow(lwratio,-1);
ks0 = E*thickness/(12.0*(1.0-nu*nu));

//using linear-stress assumption
ks11 = ks0*( (4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks21 = ks0*( 3.0/2.0*(1.0+nu) );
ks31 = ks0*( (2.0+nu*nu)*pp1-3.0/2.0*(1.0-nu)*nn1 );
ks41 = ks0*( 3.0/2.0*(1.0-3.0*nu) );
ks51 = ks0*(-(2.0+nu*nu)*pp1-3.0/2.0*(1.0-nu)*nn1 );
ks61 = ks0*(-3.0/2.0*(1.0+nu) );
ks71 = ks0*(-(4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks81 = ks0*(-3.0/2.0*(1.0-3.0*nu) );

ks22 = ks0*( (4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );
ks32 = ks0*(-3.0/2.0*(1.0-3.0*nu) );
ks42 = ks0*(-(4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );
ks52 = ks0*(-3.0/2.0*(1.0+nu) );
ks62 = ks0*(-(2.0+nu*nu)*nn1-3.0/2.0*(1.0-nu)*pp1 );
ks72 = ks0*( 3.0/2.0*(1.0-3.0*nu) );
ks82 = ks0*((2.0+nu*nu)*nn1-3.0/2.0*(1.0-nu)*pp1 );

ks33 = ks0*( (4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks43 = ks0*(-3.0/2.0*(1.0+nu) );
ks53 = ks0*(-(4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks63 = ks0*( 3.0/2.0*(1.0-3.0*nu) );
ks73 = ks0*(-(2.0+nu*nu)*pp1-3.0/2.0*(1.0-nu)*nn1 );
ks83 = ks0*( 3.0/2.0*(1.0+nu) );

ks44 = ks0*( (4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );
ks54 = ks0*(-3.0/2.0*(1.0-3.0*nu) );
ks64 = ks0*( (2.0+nu*nu)*nn1-3.0/2.0*(1.0-nu)*pp1 );
ks74 = ks0*( 3.0/2.0*(1.0+nu) );
ks84 = ks0*(-(2.0+nu*nu)*nn1-3.0/2.0*(1.0-nu)*pp1 );

ks55 = ks0*( (4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks65 = ks0*( 3.0/2.0*(1.0+nu) );
ks75 = ks0*( (2.0+nu*nu)*pp1-3.0/2.0*(1.0-nu)*nn1 );
ks85 = ks0*( 3.0/2.0*(1.0-3.0*nu) );

ks66 = ks0*( (4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );
ks76 = ks0*(-3.0/2.0*(1.0-3.0*nu) );
ks86 = ks0*(-(4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );

ks77 = ks0*( (4.0-nu*nu)*pp1+3.0/2.0*(1.0-nu)*nn1 );
ks87 = ks0*(-3.0/2.0*(1.0+nu) );

ks88 = ks0*( (4.0-nu*nu)*nn1+3.0/2.0*(1.0-nu)*pp1 );

// [ms], mass matrix for in-plane stretching
// for x-axis related rows and columns, same matrix for y-axis

ms0 = density*a*b*thickness;

ms11=4.0/36.0*ms0;
ms31=2.0/36.0*ms0;ms33=4.0/36.0*ms0;
ms51=1.0/36.0*ms0;ms53=2.0/36.0*ms0;ms55=4.0/36.0*ms0;
ms71=2.0/36.0*ms0;ms73=1.0/36.0*ms0;ms75=2.0/36.0*ms0;ms77=4.0/36.0*ms0;

ms22=4.0/36.0*ms0;
ms42=2.0/36.0*ms0;ms44=4.0/36.0*ms0;
ms62=1.0/36.0*ms0;ms64=2.0/36.0*ms0;ms66=4.0/36.0*ms0;
ms82=2.0/36.0*ms0;ms84=1.0/36.0*ms0;ms86=2.0/36.0*ms0;ms88=4.0/36.0*ms0;

// coordinate transformation matrix based on euler angles
cos_alpha = cos(alpha /180.0*'M_PI);
```

```
cos_beta = cos(beta /180.0*'M_PI);
cos_gamma = cos(gamma /180.0*'M_PI);
sin_alpha = sin(alpha /180.0*'M_PI);
sin_beta = sin(beta /180.0*'M_PI);
sin_gamma = sin(gamma /180.0*'M_PI);

// transformation matrix from chip frame to local frame
l1 =  cos_beta*cos_gamma;
m1 =  cos_beta*sin_gamma;
n1 = -sin_beta;
l2 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
n2 =  sin_alpha*cos_beta;
l3 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
m3 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
n3 =  cos_alpha*cos_beta;

// transformation matrix from local frame to chip frame
inv_l1 =  cos_beta*cos_gamma;
inv_l2 =  cos_beta*sin_gamma;
inv_l3 = -sin_beta;
inv_m1 =  sin_alpha*sin_beta*cos_gamma-cos_alpha*sin_gamma;
inv_m2 =  sin_alpha*sin_beta*sin_gamma+cos_alpha*cos_gamma;
inv_m3 =  sin_alpha*cos_beta;
inv_n1 =  cos_alpha*sin_beta*cos_gamma+sin_alpha*sin_gamma;
inv_n2 =  cos_alpha*sin_beta*sin_gamma-sin_alpha*cos_gamma;
inv_n3 =  cos_alpha*cos_beta;

// transform displacements from chip frame into local frame

l_x1 = l1*Pos(x1[0]) + m1*Pos(x1[1]) + n1*Pos(x1[2]);
l_y1 = l2*Pos(x1[0]) + m2*Pos(x1[1]) + n2*Pos(x1[2]);
l_z1 = l3*Pos(x1[0]) + m3*Pos(x1[1]) + n3*Pos(x1[2]);
l_x2 = l1*Pos(x2[0]) + m1*Pos(x2[1]) + n1*Pos(x2[2]);
l_y2 = l2*Pos(x2[0]) + m2*Pos(x2[1]) + n2*Pos(x2[2]);
l_z2 = l3*Pos(x2[0]) + m3*Pos(x2[1]) + n3*Pos(x2[2]);
l_x3 = l1*Pos(x3[0]) + m1*Pos(x3[1]) + n1*Pos(x3[2]);
l_y3 = l2*Pos(x3[0]) + m2*Pos(x3[1]) + n2*Pos(x3[2]);
l_z3 = l3*Pos(x3[0]) + m3*Pos(x3[1]) + n3*Pos(x3[2]);
l_x4 = l1*Pos(x4[0]) + m1*Pos(x4[1]) + n1*Pos(x4[2]);
l_y4 = l2*Pos(x4[0]) + m2*Pos(x4[1]) + n2*Pos(x4[2]);
l_z4 = l3*Pos(x4[0]) + m3*Pos(x4[1]) + n3*Pos(x4[2]);

l_phix1 = l1*Theta(phi1[0]) + m1*Theta(phi1[1]) + n1*Theta(phi1[2]);
l_phiy1 = l2*Theta(phi1[0]) + m2*Theta(phi1[1]) + n2*Theta(phi1[2]);
l_phiz1 = l3*Theta(phi1[0]) + m3*Theta(phi1[1]) + n3*Theta(phi1[2]);
l_phix2 = l1*Theta(phi2[0]) + m1*Theta(phi2[1]) + n1*Theta(phi2[2]);
l_phiy2 = l2*Theta(phi2[0]) + m2*Theta(phi2[1]) + n2*Theta(phi2[2]);
l_phiz2 = l3*Theta(phi2[0]) + m3*Theta(phi2[1]) + n3*Theta(phi2[2]);
l_phix3 = l1*Theta(phi3[0]) + m1*Theta(phi3[1]) + n1*Theta(phi3[2]);
l_phiy3 = l2*Theta(phi3[0]) + m2*Theta(phi3[1]) + n2*Theta(phi3[2]);
l_phiz3 = l3*Theta(phi3[0]) + m3*Theta(phi3[1]) + n3*Theta(phi3[2]);
l_phix4 = l1*Theta(phi4[0]) + m1*Theta(phi4[1]) + n1*Theta(phi4[2]);
l_phiy4 = l2*Theta(phi4[0]) + m2*Theta(phi4[1]) + n2*Theta(phi4[2]);
l_phiz4 = l3*Theta(phi4[0]) + m3*Theta(phi4[1]) + n3*Theta(phi4[2]);

// Velocity in local frame

Pos(Vx1) <+  ddt(l_x1);
Pos(Vy1) <+  ddt(l_y1);
Pos(Vz1) <+  ddt(l_z1);
Pos(Vx2) <+  ddt(l_x2);
Pos(Vy2) <+  ddt(l_y2);
Pos(Vz2) <+  ddt(l_z2);
Pos(Vx3) <+  ddt(l_x3);
```

```
Pos(Vy3) <+ ddt(l_y3);
Pos(Vz3) <+ ddt(l_z3);
Pos(Vx4) <+ ddt(l_x4);
Pos(Vy4) <+ ddt(l_y4);
Pos(Vz4) <+ ddt(l_z4);

Omega(Vphix1) <+ ddt(l_phix1);
Omega(Vphiy1) <+ ddt(l_phiy1);
Omega(Vphiz1) <+ ddt(l_phiz1);
Omega(Vphix2) <+ ddt(l_phix2);
Omega(Vphiy2) <+ ddt(l_phiy2);
Omega(Vphiz2) <+ ddt(l_phiz2);
Omega(Vphix3) <+ ddt(l_phix3);
Omega(Vphiy3) <+ ddt(l_phiy3);
Omega(Vphiz3) <+ ddt(l_phiz3);
Omega(Vphix4) <+ ddt(l_phix4);
Omega(Vphiy4) <+ ddt(l_phiy4);
Omega(Vphiz4) <+ ddt(l_phiz4);

// [F1]=[m][a]+[B][v],inertial and damping forces/moments in local frame for out-of-plane bending
// damping matrix added

fi_z1 = mb11*ddt(Pos(Vz1))+mb21*1e6*ddt(Omega(Vphix1))+mb31*1e6*ddt(Omega(Vphiy1))
+mb41*ddt(Pos(Vz2))+mb51*1e6*ddt(Omega(Vphix2))+mb61*1e6*ddt(Omega(Vphiy2))
+mb71*ddt(Pos(Vz3))+mb81*1e6*ddt(Omega(Vphix3))+mb91*1e6*ddt(Omega(Vphiy3))
+mb101*ddt(Pos(Vz4))+mb111*1e6*ddt(Omega(Vphix4))+mb121*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(dampz*mb11*ddt(l_z1)+dampphix*mb21*1e6*ddt(l_phix1)+dampphiy*mb31*1e6*ddt(l_phiy1)
+dampz*mb41*ddt(l_z2)+dampphix*mb51*1e6*ddt(l_phix2)+dampphiy*mb61*1e6*ddt(l_phiy2)
+dampz*mb71*ddt(l_z3)+dampphix*mb81*1e6*ddt(l_phix3)+dampphiy*mb91*1e6*ddt(l_phiy3)
+dampz*mb101*ddt(l_z4)+dampphix*mb111*1e6*ddt(l_phix4)+dampphiy*mb121*1e6*ddt(l_phiy4));

fi_z2 = mb41*ddt(Pos(Vz1))+mb42*1e6*ddt(Omega(Vphix1))+mb43*1e6*ddt(Omega(Vphiy1))
+mb44*ddt(Pos(Vz2))+mb54*1e6*ddt(Omega(Vphix2))+mb64*1e6*ddt(Omega(Vphiy2))
+mb74*ddt(Pos(Vz3))+mb84*1e6*ddt(Omega(Vphix3))+mb94*1e6*ddt(Omega(Vphiy3))
+mb104*ddt(Pos(Vz4))+mb114*1e6*ddt(Omega(Vphix4))+mb124*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb41*ddt(l_z1)+dampphix*mb42*1e6*ddt(l_phix1)+dampphiy*mb43*1e6*ddt(l_phiy1)
+dampz*mb44*ddt(l_z2)+dampphix*mb54*1e6*ddt(l_phix2)+dampphiy*mb64*1e6*ddt(l_phiy2)
+dampz*mb74*ddt(l_z3)+dampphix*mb84*1e6*ddt(l_phix3)+dampphiy*mb94*1e6*ddt(l_phiy3)
+dampz*mb104*ddt(l_z4)+dampphix*mb114*1e6*ddt(l_phix4)+dampphiy*mb124*1e6*ddt(l_phiy4));

fi_z3 = mb71*ddt(Pos(Vz1))+mb72*1e6*ddt(Omega(Vphix1))+mb73*1e6*ddt(Omega(Vphiy1))
+mb74*ddt(Pos(Vz2))+mb75*1e6*ddt(Omega(Vphix2))+mb76*1e6*ddt(Omega(Vphiy2))
+mb77*ddt(Pos(Vz3))+mb87*1e6*ddt(Omega(Vphix3))+mb97*1e6*ddt(Omega(Vphiy3))
+mb107*ddt(Pos(Vz4))+mb117*1e6*ddt(Omega(Vphix4))+mb127*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb71*ddt(l_z1)+dampphix*mb72*1e6*ddt(l_phix1)+dampphiy*mb73*1e6*ddt(l_phiy1)
+dampz*mb74*ddt(l_z2)+dampphix*mb75*1e6*ddt(l_phix2)+dampphiy*mb76*1e6*ddt(l_phiy2)
+dampz*mb77*ddt(l_z3)+dampphix*mb87*1e6*ddt(l_phix3)+dampphiy*mb97*1e6*ddt(l_phiy3)
+dampz*mb107*ddt(l_z4)+dampphix*mb117*1e6*ddt(l_phix4)+dampphiy*mb127*1e6*ddt(l_phiy4));

fi_z4 = mb101*ddt(Pos(Vz1))+mb102*1e6*ddt(Omega(Vphix1))+mb103*1e6*ddt(Omega(Vphiy1))
+mb104*ddt(Pos(Vz2))+mb105*1e6*ddt(Omega(Vphix2))+mb106*1e6*ddt(Omega(Vphiy2))
+mb107*ddt(Pos(Vz3))+mb108*1e6*ddt(Omega(Vphix3))+mb109*1e6*ddt(Omega(Vphiy3))
+mb1010*ddt(Pos(Vz4))+mb1110*1e6*ddt(Omega(Vphix4))+mb1210*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb101*ddt(l_z1)+dampphix*mb102*1e6*ddt(l_phix1)+dampphiy*mb103*1e6*ddt(l_phiy1)
+dampz*mb104*ddt(l_z2)+dampphix*mb105*1e6*ddt(l_phix2)+dampphiy*mb106*1e6*ddt(l_phiy2)
+dampz*mb107*ddt(l_z3)+dampphix*mb108*1e6*ddt(l_phix3)+dampphiy*mb109*1e6*ddt(l_phiy3)
+dampz*mb1010*ddt(l_z4)+dampphix*mb1110*1e6*ddt(l_phix4)+dampphiy*mb1210*1e6*ddt(l_phiy4));

tqi_x1= mb21*ddt(Pos(Vz1))+mb22*1e6*ddt(Omega(Vphix1))+mb32*1e6*ddt(Omega(Vphiy1))
+mb42*ddt(Pos(Vz2))+mb52*1e6*ddt(Omega(Vphix2))+mb62*1e6*ddt(Omega(Vphiy2))
+mb72*ddt(Pos(Vz3))+mb82*1e6*ddt(Omega(Vphix3))+mb92*1e6*ddt(Omega(Vphiy3))
+mb102*ddt(Pos(Vz4))+mb112*1e6*ddt(Omega(Vphix4))+mb122*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb21*ddt(l_z1)+dampphix*mb22*1e6*ddt(l_phix1)+dampphiy*mb32*1e6*ddt(l_phiy1)
+dampz*mb42*ddt(l_z2)+dampphix*mb52*1e6*ddt(l_phix2)+dampphiy*mb62*1e6*ddt(l_phiy2)
+dampz*mb72*ddt(l_z3)+dampphix*mb82*1e6*ddt(l_phix3)+dampphiy*mb92*1e6*ddt(l_phiy3)
```

+dampz*mb102*ddt(l_z4)+dampphix*mb112*1e6*ddt(l_phix4)+dampphiy*mb122*1e6*ddt(l_phiy4));

tqi_y1= mb31*ddt(Pos(Vz1))+mb32*1e6*ddt(Omega(Vphix1))+mb33*1e6*ddt(Omega(Vphiy1))
+mb43*ddt(Pos(Vz2))+mb53*1e6*ddt(Omega(Vphix2))+mb63*1e6*ddt(Omega(Vphiy2))
+mb73*ddt(Pos(Vz3))+mb83*1e6*ddt(Omega(Vphix3))+mb93*1e6*ddt(Omega(Vphiy3))
+mb103*ddt(Pos(Vz4))+mb113*1e6*ddt(Omega(Vphix4))+mb123*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb31*ddt(l_z1)+dampphix*mb32*1e6*ddt(l_phix1)+dampphiy*mb33*1e6*ddt(l_phiy1)
+dampz*mb43*ddt(l_z2)+dampphix*mb53*1e6*ddt(l_phix2)+dampphiy*mb63*1e6*ddt(l_phiy2)
+dampz*mb73*ddt(l_z3)+dampphix*mb83*1e6*ddt(l_phix3)+dampphiy*mb93*1e6*ddt(l_phiy3)
+dampz*mb103*ddt(l_z4)+dampphix*mb113*1e6*ddt(l_phix4)+dampphiy*mb123*1e6*ddt(l_phiy4));

tqi_x2= mb51*ddt(Pos(Vz1))+mb52*1e6*ddt(Omega(Vphix1))+mb53*1e6*ddt(Omega(Vphiy1))
+mb54*ddt(Pos(Vz2))+mb55*1e6*ddt(Omega(Vphix2))+mb65*1e6*ddt(Omega(Vphiy2))
+mb75*ddt(Pos(Vz3))+mb85*1e6*ddt(Omega(Vphix3))+mb95*1e6*ddt(Omega(Vphiy3))
+mb105*ddt(Pos(Vz4))+mb115*1e6*ddt(Omega(Vphix4))+mb125*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb51*ddt(l_z1)+dampphix*mb52*1e6*ddt(l_phix1)+dampphiy*mb53*1e6*ddt(l_phiy1)
+dampz*mb54*ddt(l_z2)+dampphix*mb55*1e6*ddt(l_phix2)+dampphiy*mb65*1e6*ddt(l_phiy2)
+dampz*mb75*ddt(l_z3)+dampphix*mb85*1e6*ddt(l_phix3)+dampphiy*mb95*1e6*ddt(l_phiy3)
+dampz*mb105*ddt(l_z4)+dampphix*mb115*1e6*ddt(l_phix4)+dampphiy*mb125*1e6*ddt(l_phiy4));

tqi_y2= mb61*ddt(Pos(Vz1))+mb62*1e6*ddt(Omega(Vphix1))+mb63*1e6*ddt(Omega(Vphiy1))
+mb64*ddt(Pos(Vz2))+mb65*1e6*ddt(Omega(Vphix2))+mb66*1e6*ddt(Omega(Vphiy2))
+mb76*ddt(Pos(Vz3))+mb86*1e6*ddt(Omega(Vphix3))+mb96*1e6*ddt(Omega(Vphiy3))
+mb106*ddt(Pos(Vz4))+mb116*1e6*ddt(Omega(Vphix4))+mb126*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb61*ddt(l_z1)+dampphix*mb62*1e6*ddt(l_phix1)+dampphiy*mb63*1e6*ddt(l_phiy1)
+dampz*mb64*ddt(l_z2)+dampphix*mb65*1e6*ddt(l_phix2)+dampphiy*mb66*1e6*ddt(l_phiy2)
+dampz*mb76*ddt(l_z3)+dampphix*mb86*1e6*ddt(l_phix3)+dampphiy*mb96*1e6*ddt(l_phiy3)
+dampz*mb106*ddt(l_z4)+dampphix*mb116*1e6*ddt(l_phix4)+dampphiy*mb126*1e6*ddt(l_phiy4));

tqi_x3= mb81*ddt(Pos(Vz1))+mb82*1e6*ddt(Omega(Vphix1))+mb83*1e6*ddt(Omega(Vphiy1))
+mb84*ddt(Pos(Vz2))+mb85*1e6*ddt(Omega(Vphix2))+mb86*1e6*ddt(Omega(Vphiy2))
+mb87*ddt(Pos(Vz3))+mb88*1e6*ddt(Omega(Vphix3))+mb98*1e6*ddt(Omega(Vphiy3))
+mb108*ddt(Pos(Vz4))+mb118*1e6*ddt(Omega(Vphix4))+mb128*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb81*ddt(l_z1)+dampphix*mb82*1e6*ddt(l_phix1)+dampphiy*mb83*1e6*ddt(l_phiy1)
+dampz*mb84*ddt(l_z2)+dampphix*mb85*1e6*ddt(l_phix2)+dampphiy*mb86*1e6*ddt(l_phiy2)
+dampz*mb87*ddt(l_z3)+dampphix*mb88*1e6*ddt(l_phix3)+dampphiy*mb98*1e6*ddt(l_phiy3)
+dampz*mb108*ddt(l_z4)+dampphix*mb118*1e6*ddt(l_phix4)+dampphiy*mb128*1e6*ddt(l_phiy4));

tqi_y3= mb91*ddt(Pos(Vz1))+mb92*1e6*ddt(Omega(Vphix1))+mb93*1e6*ddt(Omega(Vphiy1))
+mb94*ddt(Pos(Vz2))+mb95*1e6*ddt(Omega(Vphix2))+mb96*1e6*ddt(Omega(Vphiy2))
+mb97*ddt(Pos(Vz3))+mb98*1e6*ddt(Omega(Vphix3))+mb99*1e6*ddt(Omega(Vphiy3))
+mb109*ddt(Pos(Vz4))+mb119*1e6*ddt(Omega(Vphix4))+mb129*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb91*ddt(l_z1)+dampphix*mb92*1e6*ddt(l_phix1)+dampphiy*mb93*1e6*ddt(l_phiy1)
+dampz*mb94*ddt(l_z2)+dampphix*mb95*1e6*ddt(l_phix2)+dampphiy*mb96*1e6*ddt(l_phiy2)
+dampz*mb97*ddt(l_z3)+dampphix*mb98*1e6*ddt(l_phix3)+dampphiy*mb99*1e6*ddt(l_phiy3)
+dampz*mb109*ddt(l_z4)+dampphix*mb119*1e6*ddt(l_phix4)+dampphiy*mb129*1e6*ddt(l_phiy4));

tqi_x4= mb111*ddt(Pos(Vz1))+mb112*1e6*ddt(Omega(Vphix1))+mb113*1e6*ddt(Omega(Vphiy1))
+mb114*ddt(Pos(Vz2))+mb115*1e6*ddt(Omega(Vphix2))+mb116*1e6*ddt(Omega(Vphiy2))
+mb117*ddt(Pos(Vz3))+mb118*1e6*ddt(Omega(Vphix3))+mb119*1e6*ddt(Omega(Vphiy3))
+mb1110*ddt(Pos(Vz4))+mb1111*1e6*ddt(Omega(Vphix4))+mb1211*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb111*ddt(l_z1)+dampphix*mb112*1e6*ddt(l_phix1)+dampphiy*mb113*1e6*ddt(l_phiy1)
+dampz*mb114*ddt(l_z2)+dampphix*mb115*1e6*ddt(l_phix2)+dampphiy*mb116*1e6*ddt(l_phiy2)
+dampz*mb117*ddt(l_z3)+dampphix*mb118*1e6*ddt(l_phix3)+dampphiy*mb119*1e6*ddt(l_phiy3)
+dampz*mb1110*ddt(l_z4)+dampphix*mb1111*1e6*ddt(l_phix4)+dampphiy*mb1211*1e6*ddt(l_phiy4));

tqi_y4= mb121*ddt(Pos(Vz1))+mb122*1e6*ddt(Omega(Vphix1))+mb123*1e6*ddt(Omega(Vphiy1))
+mb124*ddt(Pos(Vz2))+mb125*1e6*ddt(Omega(Vphix2))+mb126*1e6*ddt(Omega(Vphiy2))
+mb127*ddt(Pos(Vz3))+mb128*1e6*ddt(Omega(Vphix3))+mb129*1e6*ddt(Omega(Vphiy3))
+mb1210*ddt(Pos(Vz4))+mb1211*1e6*ddt(Omega(Vphix4))+mb1212*1e6*ddt(Omega(Vphiy4))
+1.0/mb0*(mb121*ddt(l_z1)+dampphix*mb122*1e6*ddt(l_phix1)+dampphiy*mb123*1e6*ddt(l_phiy1)
+dampz*mb124*ddt(l_z2)+dampphix*mb125*1e6*ddt(l_phix2)+dampphiy*mb126*1e6*ddt(l_phiy2)
+dampz*mb127*ddt(l_z3)+dampphix*mb128*1e6*ddt(l_phix3)+dampphiy*mb129*1e6*ddt(l_phiy3)
+dampz*mb1210*ddt(l_z4)+dampphix*mb1211*1e6*ddt(l_phix4)+dampphiy*mb1212*1e6*ddt(l_phiy4));

// [F2]=[k][x], spring forces/moments in local frame for out-of-plane bending

Fz1 =   kb11*l_z1+kb21*1e6*l_phix1+kb31*1e6*l_phiy1
+kb41*l_z2+kb51*1e6*l_phix2+kb61*1e6*l_phiy2
+kb71*l_z3+kb81*1e6*l_phix3+kb91*1e6*l_phiy3
+kb101*l_z4+kb111*1e6*l_phix4+kb121*1e6*l_phiy4;

Fz2 =   kb41*l_z1+kb42*1e6*l_phix1+kb43*1e6*l_phiy1
+kb44*l_z2+kb54*1e6*l_phix2+kb64*1e6*l_phiy2
+kb74*l_z3+kb84*1e6*l_phix3+kb94*1e6*l_phiy3
+kb104*l_z4+kb114*1e6*l_phix4+kb124*1e6*l_phiy4;

Fz3 =   kb71*l_z1+kb72*1e6*l_phix1+kb73*1e6*l_phiy1
+kb74*l_z2+kb75*1e6*l_phix2+kb76*1e6*l_phiy2
+kb77*l_z3+kb87*1e6*l_phix3+kb97*1e6*l_phiy3
+kb107*l_z4+kb117*1e6*l_phix4+kb127*1e6*l_phiy4;

Fz4 =   kb101*l_z1+kb102*1e6*l_phix1+kb103*1e6*l_phiy1
 +kb104*l_z2+kb105*1e6*l_phix2+kb106*1e6*l_phiy2
 +kb107*l_z3+kb108*1e6*l_phix3+kb109*1e6*l_phiy3
 +kb1010*l_z4+kb1110*1e6*l_phix4+kb1210*1e6*l_phiy4;

Tqx1=   kb21*l_z1+kb22*1e6*l_phix1+kb32*1e6*l_phiy1
+kb42*l_z2+kb52*1e6*l_phix2+kb62*1e6*l_phiy2
+kb72*l_z3+kb82*1e6*l_phix3+kb92*1e6*l_phiy3
+kb102*l_z4+kb112*1e6*l_phix4+kb122*1e6*l_phiy4;

Tqy1=   kb31*l_z1+kb32*1e6*l_phix1+kb33*1e6*l_phiy1
+kb43*l_z2+kb53*1e6*l_phix2+kb63*1e6*l_phiy2
+kb73*l_z3+kb83*1e6*l_phix3+kb93*1e6*l_phiy3
+kb103*l_z4+kb113*1e6*l_phix4+kb123*1e6*l_phiy4;

Tqx2=   kb51*l_z1+kb52*1e6*l_phix1+kb53*1e6*l_phiy1
+kb54*l_z2+kb55*1e6*l_phix2+kb65*1e6*l_phiy2
+kb75*l_z3+kb85*1e6*l_phix3+kb95*1e6*l_phiy3
+kb105*l_z4+kb115*1e6*l_phix4+kb125*1e6*l_phiy4;

Tqy2=   kb61*l_z1+kb62*1e6*l_phix1+kb63*1e6*l_phiy1
+kb64*l_z2+kb65*1e6*l_phix2+kb66*1e6*l_phiy2
+kb76*l_z3+kb86*1e6*l_phix3+kb96*1e6*l_phiy3
+kb106*l_z4+kb116*1e6*l_phix4+kb126*1e6*l_phiy4;

Tqx3=   kb81*l_z1+kb82*1e6*l_phix1+kb83*1e6*l_phiy1
+kb84*l_z2+kb85*1e6*l_phix2+kb86*1e6*l_phiy2
+kb87*l_z3+kb88*1e6*l_phix3+kb98*1e6*l_phiy3
+kb108*l_z4+kb118*1e6*l_phix4+kb128*1e6*l_phiy4;

Tqy3=   kb91*l_z1+kb92*1e6*l_phix1+kb93*1e6*l_phiy1
+kb94*l_z2+kb95*1e6*l_phix2+kb96*1e6*l_phiy2
+kb97*l_z3+kb98*1e6*l_phix3+kb99*1e6*l_phiy3
+kb109*l_z4+kb119*1e6*l_phix4+kb129*1e6*l_phiy4;

Tqx4=   kb111*l_z1+kb112*1e6*l_phix1+kb113*1e6*l_phiy1
+kb114*l_z2+kb115*1e6*l_phix2+kb116*1e6*l_phiy2
+kb117*l_z3+kb118*1e6*l_phix3+kb119*1e6*l_phiy3
+kb1110*l_z4+kb1111*1e6*l_phix4+kb1211*1e6*l_phiy4;

Tqy4=   kb121*l_z1+kb122*1e6*l_phix1+kb123*1e6*l_phiy1
+kb124*l_z2+kb125*1e6*l_phix2+kb126*1e6*l_phiy2
+kb127*l_z3+kb128*1e6*l_phix3+kb129*1e6*l_phiy3
+kb1210*l_z4+kb1211*1e6*l_phix4+kb1212*1e6*l_phiy4;

// [F1]=[m][a]+[B][v], inertial and damping forces/moments in local frame for in-plane stretching
// damping matrix added

fi_x1 = ms11*ddt(Pos(Vx1))+ms21*ddt(Pos(Vy1))
+ms31*ddt(Pos(Vx2))+ms41*ddt(Pos(Vy2))
+ms51*ddt(Pos(Vx3))+ms61*ddt(Pos(Vy3))
+ms71*ddt(Pos(Vx4))+ms81*ddt(Pos(Vy4))
+ms11/ms0*dampx*ddt(l_x1)+ms21/ms0*dampy*ddt(l_y1)
+ms31/ms0*dampx*ddt(l_x2)+ms41/ms0*dampy*ddt(l_y2)
+ms51/ms0*dampx*ddt(l_x3)+ms61/ms0*dampy*ddt(l_y3)
+ms71/ms0*dampx*ddt(l_x4)+ms81/ms0*dampy*ddt(l_y4);

fi_y1 = ms21*ddt(Pos(Vx1))+ms22*ddt(Pos(Vy1))
+ms32*ddt(Pos(Vx2))+ms42*ddt(Pos(Vy2))
+ms52*ddt(Pos(Vx3))+ms62*ddt(Pos(Vy3))
+ms72*ddt(Pos(Vx4))+ms82*ddt(Pos(Vy4))
+ms21/ms0*dampx*ddt(l_x1)+ms22/ms0*dampy*ddt(l_y1)
+ms32/ms0*dampx*ddt(l_x2)+ms42/ms0*dampy*ddt(l_y2)
+ms52/ms0*dampx*ddt(l_x3)+ms62/ms0*dampy*ddt(l_y3)
+ms72/ms0*dampx*ddt(l_x4)+ms82/ms0*dampy*ddt(l_y4);

fi_x2 = ms31*ddt(Pos(Vx1))+ms32*ddt(Pos(Vy1))
+ms33*ddt(Pos(Vx2))+ms43*ddt(Pos(Vy2))
+ms53*ddt(Pos(Vx3))+ms63*ddt(Pos(Vy3))
+ms73*ddt(Pos(Vx4))+ms83*ddt(Pos(Vy4))
+ms31/ms0*dampx*ddt(l_x1)+ms32/ms0*dampy*ddt(l_y1)
+ms33/ms0*dampx*ddt(l_x2)+ms43/ms0*dampy*ddt(l_y2)
+ms53/ms0*dampx*ddt(l_x3)+ms63/ms0*dampy*ddt(l_y3)
+ms73/ms0*dampx*ddt(l_x4)+ms83/ms0*dampy*ddt(l_y4);

fi_y2 = ms41*ddt(Pos(Vx1))+ms42*ddt(Pos(Vy1))
+ms43*ddt(Pos(Vx2))+ms44*ddt(Pos(Vy2))
+ms54*ddt(Pos(Vx3))+ms64*ddt(Pos(Vy3))
+ms74*ddt(Pos(Vx4))+ms84*ddt(Pos(Vy4))
+ms41/ms0*dampx*ddt(l_x1)+ms42/ms0*dampy*ddt(l_y1)
+ms43/ms0*dampx*ddt(l_x2)+ms44/ms0*dampy*ddt(l_y2)
+ms54/ms0*dampx*ddt(l_x3)+ms64/ms0*dampy*ddt(l_y3)
+ms74/ms0*dampx*ddt(l_x4)+ms84/ms0*dampy*ddt(l_y4);

fi_x3 = ms51*ddt(Pos(Vx1))+ms52*ddt(Pos(Vy1))
+ms53*ddt(Pos(Vx2))+ms54*ddt(Pos(Vy2))
+ms55*ddt(Pos(Vx3))+ms65*ddt(Pos(Vy3))
+ms75*ddt(Pos(Vx4))+ms85*ddt(Pos(Vy4))
+ms51/ms0*dampx*ddt(l_x1)+ms52/ms0*dampy*ddt(l_y1)
+ms53/ms0*dampx*ddt(l_x2)+ms54/ms0*dampy*ddt(l_y2)
+ms55/ms0*dampx*ddt(l_x3)+ms65/ms0*dampy*ddt(l_y3)
+ms75/ms0*dampx*ddt(l_x4)+ms85/ms0*dampy*ddt(l_y4);

fi_y3 = ms61*ddt(Pos(Vx1))+ms62*ddt(Pos(Vy1))
+ms63*ddt(Pos(Vx2))+ms64*ddt(Pos(Vy2))
+ms65*ddt(Pos(Vx3))+ms66*ddt(Pos(Vy3))
+ms76*ddt(Pos(Vx4))+ms86*ddt(Pos(Vy4))
+ms61/ms0*dampx*ddt(l_x1)+ms62/ms0*dampy*ddt(l_y1)
+ms63/ms0*dampx*ddt(l_x2)+ms64/ms0*dampy*ddt(l_y2)
+ms65/ms0*dampx*ddt(l_x3)+ms66/ms0*dampy*ddt(l_y3)
+ms76/ms0*dampx*ddt(l_x4)+ms86/ms0*dampy*ddt(l_y4);

fi_x4 = ms71*ddt(Pos(Vx1))+ms72*ddt(Pos(Vy1))
+ms73*ddt(Pos(Vx2))+ms74*ddt(Pos(Vy2))
+ms75*ddt(Pos(Vx3))+ms76*ddt(Pos(Vy3))
+ms77*ddt(Pos(Vx4))+ms87*ddt(Pos(Vy4))
+ms71/ms0*dampx*ddt(l_x1)+ms72/ms0*dampy*ddt(l_y1)
+ms73/ms0*dampx*ddt(l_x2)+ms74/ms0*dampy*ddt(l_y2)
+ms75/ms0*dampx*ddt(l_x3)+ms76/ms0*dampy*ddt(l_y3)
+ms77/ms0*dampx*ddt(l_x4)+ms87/ms0*dampy*ddt(l_y4);

fi_y4 = ms81*ddt(Pos(Vx1))+ms82*ddt(Pos(Vy1))
+ms83*ddt(Pos(Vx2))+ms84*ddt(Pos(Vy2))

Modeling and Simulation for Design of Suspended MEMS   305

```
+ms85*ddt(Pos(Vx3))+ms86*ddt(Pos(Vy3))
+ms87*ddt(Pos(Vx4))+ms88*ddt(Pos(Vy4))
+ms81/ms0*dampx*ddt(l_x1)+ms82/ms0*dampy*ddt(l_y1)
+ms83/ms0*dampx*ddt(l_x2)+ms84/ms0*dampy*ddt(l_y2)
+ms85/ms0*dampx*ddt(l_x3)+ms86/ms0*dampy*ddt(l_y3)
+ms87/ms0*dampx*ddt(l_x4)+ms88/ms0*dampy*ddt(l_y4);

// [F2]=[k][x], spring forces/moments in local frame for in-plane stretching

Fx1 =  ks11*l_x1+ks21*l_y1+ks31*l_x2+ks41*l_y2
+ks51*l_x3+ks61*l_y3+ks71*l_x4+ks81*l_y4;

Fy1 =  ks21*l_x1+ks22*l_y1+ks32*l_x2+ks42*l_y2
+ks52*l_x3+ks62*l_y3+ks72*l_x4+ks82*l_y4;

Fx2 =  ks31*l_x1+ks32*l_y1+ks33*l_x2+ks43*l_y2
+ks53*l_x3+ks63*l_y3+ks73*l_x4+ks83*l_y4;

Fy2 =  ks41*l_x1+ks42*l_y1+ks43*l_x2+ks44*l_y2
+ks54*l_x3+ks64*l_y3+ks74*l_x4+ks84*l_y4;

Fx3 =   ks51*l_x1+ks52*l_y1+ks53*l_x2+ks54*l_y2
+ks55*l_x3+ks65*l_y3+ks75*l_x4+ks85*l_y4;

Fy3 =  ks61*l_x1+ks62*l_y1+ks63*l_x2+ks64*l_y2
+ks65*l_x3+ks66*l_y3+ks76*l_x4+ks86*l_y4;

Fx4 =  ks71*l_x1+ks72*l_y1+ks73*l_x2+ks74*l_y2
+ks75*l_x3+ks76*l_y3+ks77*l_x4+ks87*l_y4;

Fy4 =  ks81*l_x1+ks82*l_y1+ks83*l_x2+ks84*l_y2
+ks85*l_x3+ks86*l_y3+ks87*l_x4+ks88*l_y4;

// spring forces/moments transformed from local frame back to chip frame

Fchipx1 = inv_l1*Fx1 + inv_m1*Fy1 + inv_n1*Fz1;
Fchipy1 = inv_l2*Fx1 + inv_m2*Fy1 + inv_n2*Fz1;
Fchipz1 = inv_l3*Fx1 + inv_m3*Fy1 + inv_n3*Fz1;

Fchipx2 = inv_l1*Fx2 + inv_m1*Fy2 + inv_n1*Fz2;
Fchipy2 = inv_l2*Fx2 + inv_m2*Fy2 + inv_n2*Fz2;
Fchipz2 = inv_l3*Fx2 + inv_m3*Fy2 + inv_n3*Fz2;

Fchipx3 = inv_l1*Fx3 + inv_m1*Fy3 + inv_n1*Fz3;
Fchipy3 = inv_l2*Fx3 + inv_m2*Fy3 + inv_n2*Fz3;
Fchipz3 = inv_l3*Fx3 + inv_m3*Fy3 + inv_n3*Fz3;

Fchipx4 = inv_l1*Fx4 + inv_m1*Fy4 + inv_n1*Fz4;
Fchipy4 = inv_l2*Fx4 + inv_m2*Fy4 + inv_n2*Fz4;
Fchipz4 = inv_l3*Fx4 + inv_m3*Fy4 + inv_n3*Fz4;

Tqchipx1 = inv_l1*Tqx1 + inv_m1*Tqy1 ;
Tqchipy1 = inv_l2*Tqx1 + inv_m2*Tqy1 ;

Tqchipx2 = inv_l1*Tqx2 + inv_m1*Tqy2 ;
Tqchipy2 = inv_l2*Tqx2 + inv_m2*Tqy2 ;

Tqchipx3 = inv_l1*Tqx3 + inv_m1*Tqy3 ;
Tqchipy3 = inv_l2*Tqx3 + inv_m2*Tqy3 ;

Tqchipx4 = inv_l1*Tqx4 + inv_m1*Tqy4 ;
Tqchipy4 = inv_l2*Tqx4 + inv_m2*Tqy4 ;

// inertial and damping forces/moments transformed from local frame back to chip frame
```

Modeling and Simulation for Design of Suspended MEMS

```
Fichipx1 = inv_l1*fi_x1 + inv_m1*fi_y1 + inv_n1*fi_z1;
Fichipy1 = inv_l2*fi_x1 + inv_m2*fi_y1 + inv_n2*fi_z1;
Fichipz1 = inv_l3*fi_x1 + inv_m3*fi_y1 + inv_n3*fi_z1;

Fichipx2 = inv_l1*fi_x2 + inv_m1*fi_y2 + inv_n1*fi_z2;
Fichipy2 = inv_l2*fi_x2 + inv_m2*fi_y2 + inv_n2*fi_z2;
Fichipz2 = inv_l3*fi_x2 + inv_m3*fi_y2 + inv_n3*fi_z2;

Fichipx3 = inv_l1*fi_x3 + inv_m1*fi_y3 + inv_n1*fi_z3;
Fichipy3 = inv_l2*fi_x3 + inv_m2*fi_y3 + inv_n2*fi_z3;
Fichipz3 = inv_l3*fi_x3 + inv_m3*fi_y3 + inv_n3*fi_z3;

Fichipx4 = inv_l1*fi_x4 + inv_m1*fi_y4 + inv_n1*fi_z4;
Fichipy4 = inv_l2*fi_x4 + inv_m2*fi_y4 + inv_n2*fi_z4;
Fichipz4 = inv_l3*fi_x4 + inv_m3*fi_y4 + inv_n3*fi_z4;

Tqichipx1 = inv_l1*tqi_x1 + inv_m1*tqi_y1 ;
Tqichipy1 = inv_l2*tqi_x1 + inv_m2*tqi_y1 ;

Tqichipx2 = inv_l1*tqi_x2 + inv_m1*tqi_y2 ;
Tqichipy2 = inv_l2*tqi_x2 + inv_m2*tqi_y2 ;

Tqichipx3 = inv_l1*tqi_x3 + inv_m1*tqi_y3 ;
Tqichipy3 = inv_l2*tqi_x3 + inv_m2*tqi_y3 ;

Tqichipx4 = inv_l1*tqi_x4 + inv_m1*tqi_y4 ;
Tqichipy4 = inv_l2*tqi_x4 + inv_m2*tqi_y4 ;

// spring forces/moments applied to the ends of beam

F(x1[0]) <+ -Fchipx1;
F(x1[1]) <+ -Fchipy1;
F(x1[2]) <+ -Fchipz1;

F(x2[0]) <+ -Fchipx2;
F(x2[1]) <+ -Fchipy2;
F(x2[2]) <+ -Fchipz2;

F(x3[0]) <+ -Fchipx3;
F(x3[1]) <+ -Fchipy3;
F(x3[2]) <+ -Fchipz3;

F(x4[0]) <+ -Fchipx4;
F(x4[1]) <+ -Fchipy4;
F(x4[2]) <+ -Fchipz4;

Tau(phi1[0]) <+ -Tqchipx1;
Tau(phi1[1]) <+ -Tqchipy1;

Tau(phi2[0]) <+ -Tqchipx2;
Tau(phi2[1]) <+ -Tqchipy2;

Tau(phi3[0]) <+ -Tqchipx3;
Tau(phi3[1]) <+ -Tqchipy3;

Tau(phi4[0]) <+ -Tqchipx4;
Tau(phi4[1]) <+ -Tqchipy4;

// inertia & damping forces/moments applied to the ends of beam

F(x1[0]) <+ -Fichipx1;
F(x1[1]) <+ -Fichipy1;
F(x1[2]) <+ -Fichipz1;

F(x2[0]) <+ -Fichipx2;
```

```
F(x2[1]) <+ -Fichipy2;
F(x2[2]) <+ -Fichipz2;

F(x3[0]) <+ -Fichipx3;
F(x3[1]) <+ -Fichipy3;
F(x3[2]) <+ -Fichipz3;

F(x4[0]) <+ -Fichipx4;
F(x4[1]) <+ -Fichipy4;
F(x4[2]) <+ -Fichipz4;

Tau(phi1[0]) <+ -Tqichipx1;
Tau(phi1[1]) <+ -Tqichipy1;

Tau(phi2[0]) <+ -Tqichipx2;
Tau(phi2[1]) <+ -Tqichipy2;

Tau(phi3[0]) <+ -Tqichipx3;
Tau(phi3[1]) <+ -Tqichipy3;

Tau(phi4[0]) <+ -Tqichipx4;
Tau(phi4[1]) <+ -Tqichipy4;

// rigid-body approximation for the degree of freedom of twisting about z-axis

// internal variables related to the twisting about z-axis
rad = gamma* 'M_PI /180;
lngth_r = sqrt(pow(a/2,2)+pow(b/2,2));
arc_r = atan((b/2)/(a/2));
ii = density*thickness*b*a*(b*b+a*a)/12.0;

// phim is in Mega-radian
angmid = 1e6*Theta(phim);
phi_off_1_r = arc_r + (rad + angmid);
phi_off_2_r = arc_r - (rad + angmid);

// twisting moments about z-axis
Omega(angvmid) <+ ddt(angmid);

Tau(phim) <+ - (ii*ddt(Omega(angvmid))
        + dampphiz *Omega(angvmid)
+ lngth_r* ( sin(phi_off_1_r)*F(x1[0],x3[0])
+cos(phi_off_1_r)*F(x3[1],x1[1])
+sin(phi_off_2_r)*F(x4[0],x2[0])
+cos(phi_off_2_r)*F(x4[1],x2[1]) ));

// rigid-body contraints for twisting about z-axis
Theta(phi1[2],phim)  <+  0;
Theta(phi2[2],phim)  <+  0;
Theta(phi3[2],phim)  <+  0;
Theta(phi4[2],phim)  <+  0;


//*******equations for electrical***********
I(v1, vmid) <+ V(v1, vmid)/r ;
I(v2, vmid) <+ V(v2, vmid)/r ;
I(v3, vmid) <+ V(v3, vmid)/r ;
I(v4, vmid) <+ V(v4, vmid)/r ;

end
endmodule
```

A.9 Electrostatic Gap Model

The Verilog-A code for the electrostatic gap model discussed in Chapter 3.5 is given in this section. This model captures the electrostatic effects between two beam electrodes with in-plane lateral bending. Rotated rigid-plate approximation is applied to the overlapping region. The distributed electrostatic forces and damping forces are both lumped into equivalent forces and moments at the ends of the electrodes, based on the beam bending shape functions. The contact model is also included in this model.

The Verilog-A code for this model is:

```
// include header files
'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;

// topology = 0 if the top finger is located on the right of the bottom finger
// topology = 1 if the top finger is located on the left of the bottom finger
```

```
parameter integer topology = 1;

parameter real finger_w_t = 2e-6;
parameter real finger_w_b = 2e-6;

parameter real finger_l_t = 20e-6;
parameter real finger_l_b = 20e-6;

parameter real finger_number = 1;
parameter real gap = 2e-6;
parameter real overlap = 20e-6;
parameter real thickness = 2e-6;

parameter real E = 165e9;

parameter real angle = 0;

parameter real visc_air = 'default_visc_air;
parameter real contact_oxide_t = 'default_ntv_ox_t;

real ov;   // dynamic overlap along x-axis
real dy;     // change of gap in y-direction
real dx;     // change of overlap in x-direction

real vlt;    // applied voltage
real vlt_sqrt;  // v^2
real cap;     // capacitance
real q1, q2;  // charge
real q;  // charge

real slope;// stiffness of contact spring
integer const2; // used for switch between normal and in-contact states

real l_attach, l_tip;
real L1, L2, ym1, yp1, am1, ap1, lm, lp, ym2, yp2, am2, ap2;
real y1_mid, y2_mid;
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 xmid, y1mid, ang1mid, y1average, y2mid, ang2mid, y2average, theta0;

real c1, c2, c3, intFy0, intFy0_b, Fy_contact;
real Fxd_l, Fa1x, Fa1y, Fa2x, Fa2y, Fb1x, Fb1y, Fb2x, Fb2y;
real Fa1y_e, Fb1y_e, Fa2y_e, Fb2y_e, Fa1y_b, Fb1y_b, Fa2y_b, Fb2y_b;
real Fa1_l_e, Fa2_l_e, Fb1_l_e, Fb2_l_e;
real Ma1_l_e, Ma2_l_e, Mb1_l_e, Mb2_l_e, Ma1_l_b, Ma2_l_b, Mb1_l_b, Mb2_l_b;
real Ma1, Ma2, Mb1, Mb2;
real Fy_tip_1, Fy_tip_2, Fy_attach_1, Fy_attach_2;


analog begin

// check the validity of the value of parameter topology
     @ (initial_step) begin
     if (topology < 0 | topology > 1) begin
     $display("Invalid topology, must be either '0' or '1', see 'help' for descriptions.");
      $finish;
     end
     end


     rad = angle* 'M_PI /180;
     cos_dc = cos(rad);
     sin_dc = sin(rad);
```

```
// transform the displacement from chip frame into 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]);

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]);


// to use same set of equations for different topologies
// determine the variables used for different topologies
xm1 = xm1_l*topology+xm2_l*(1-topology);
ym1 = ym1_l*topology-ym2_l*(1-topology);
xm2 = xm2_l*topology+xm1_l*(1-topology);
ym2 = ym2_l*topology-ym1_l*(1-topology);
xp1 = xp1_l*topology+xp2_l*(1-topology);
yp1 = yp1_l*topology-yp2_l*(1-topology);
xp2 = xp2_l*topology+xp1_l*(1-topology);
yp2 = yp2_l*topology-yp1_l*(1-topology);

am1 = Theta(phia_t)*1e6*topology-Theta(phia_b)*1e6*(1-topology);
ap1 = Theta(phib_t)*1e6*topology-Theta(phib_b)*1e6*(1-topology);
am2 = Theta(phia_b)*1e6*topology-Theta(phia_t)*1e6*(1-topology);
ap2 = Theta(phib_b)*1e6*topology-Theta(phib_t)*1e6*(1-topology);

vlt = (1-topology)*V(vb_b,va_t) + topology*V(va_b,vb_t);
vlt_sqrt = pow(vlt,2);

// parameter overlap is the initial overlap
// dx is used for calculation of dynamic overlap, ov

dx = xp1-xm2;

if(dx <=  -overlap) begin
//  beams are not overlapped to form a gap
ov = 0;
end

else if(dx <= (topology*(finger_l_b-overlap)+(1-topology)*(finger_l_t-overlap))) begin
//  beams have increasing overlap
ov = overlap + dx;
end

else if(dx <= (finger_l_t+finger_l_b - overlap)) begin
//  beams have decreasing overlap
ov = - dx + finger_l_t+finger_l_b - overlap;
end

else begin
//  beams are not overlapped to form a gap
ov = 0;
end


// L1 and L2 are switched for topology 1 and 0
L1 = finger_l_t*topology+finger_l_b*(1-topology);
L2 = finger_l_b*topology+finger_l_t*(1-topology);

// physical equations are formulated based on toplogy 1
```

```
// y1_mid is the y-disp of beam1 at inner boundary of overlapping region
// y1_mid is the y-disp of beam2 at inner boundary of overlapping region

y1_mid = ym1*(1-(3*pow(L1-ov,2))/pow(L1,2)+(2*pow(L1-ov,3))/pow(L1,3))+
    ap1*(-(pow(L1-ov,2)/L1)+pow(L1-ov,3)/pow(L1,2))+
    am1*(L1-(2*pow(L1-ov,2))/L1+pow(L1-ov,3)/pow(L1,2)-ov)+
    ((3*pow(L1-ov,2))/pow(L1,2)-(2*pow(L1-ov,3))/pow(L1,3))*yp1;

y2_mid = ym2*((3*pow(L2-ov,2))/pow(L2,2)-(2*pow(L2-ov,3))/pow(L2,3))-
    am2*(-(pow(L2-ov,2)/L2)+pow(L2-ov,3)/pow(L2,2))-
    ap2*(L2-(2*pow(L2-ov,2))/L2+pow(L2-ov,3)/pow(L2,2)-ov)+
    (1-(3*pow(L2-ov,2))/pow(L2,2)+(2*pow(L2-ov,3))/pow(L2,3))*yp2;

// dy is used to determine the operation states: normal or in-contact

dy = min((yp1-y2_mid),(y1_mid-ym2));

    if((gap + dy) > contact_oxide_t*2)//normal gap
     const2 = 0;
    else//in contact
     const2 = 1;

// calculation of cap, forces and moments, based on topology 1

lm = L1 - ov;
lp = L1;

xmid = (lm+lp)/2.0;

ang1mid = am1*(1 - (2*(lm + lp))/L1 + (3*pow(lm + lp,2))/(4.0*pow(L1,2))) + ap1*(-((lm + lp)/L1) + (3*pow(lm + lp,2))/
(4.0*pow(L1,2))) + ((-3*(lm + lp))/pow(L1,2) + (3*pow(lm + lp,2))/(2.0*pow(L1,3)))*ym1 + ((3*(lm + lp))/pow(L1,2) - (3*pow(lm + lp,2))/
(2.0*pow(L1,3)))*yp1;

y1mid = am1*((lm + lp)/2.0 - pow(lm + lp,2)/(2.0*L1) + pow(lm + lp,3)/(8.0*pow(L1,2))) + ap1*(-pow(lm + lp,2)/(4.0*L1) + pow(lm
+ lp,3)/(8.0*pow(L1,2))) + (1 - (3*pow(lm + lp,2))/(4.0*pow(L1,2)) + pow(lm + lp,3)/(4.0*pow(L1,3)))*ym1 + ((3*pow(lm + lp,2))/
(4.0*pow(L1,2)) - pow(lm + lp,3)/(4.0*pow(L1,3)))*yp1;

y1average = (am1*(lm - (2*pow(lm,2))/L1 + pow(lm,3)/pow(L1,2)) + ap1*(-(pow(lm,2)/L1) + pow(lm,3)/pow(L1,2)) +   am1*(lp -
(2*pow(lp,2))/L1 + pow(lp,3)/pow(L1,2)) + ap1*(-(pow(lp,2)/L1) + pow(lp,3)/pow(L1,2)) +   (1 - (3*pow(lm,2))/pow(L1,2) +
(2*pow(lm,3))/pow(L1,3))*ym1 +   (1 - (3*pow(lp,2))/pow(L1,2) + (2*pow(lp,3))/pow(L1,3))*ym1 +   ((3*pow(lm,2))/pow(L1,2) -
(2*pow(lm,3))/pow(L1,3))*yp1 +   ((3*pow(lp,2))/pow(L1,2) - (2*pow(lp,3))/pow(L1,3))*yp1)/2.0;

ang2mid = ym2*((-6*(L1 + L2 + (-lm - lp)/2.0 - ov))/pow(L2,2) + (6*pow(L1 + L2 + (-lm - lp)/2.0 - ov,2))/pow(L2,3)) - am2*((2*(L1
+ L2 + (-lm - lp)/2.0 - ov))/L2 - (3*pow(L1 + L2 + (-lm - lp)/2.0 - ov,2))/pow(L2,2)) - ap2*(-1 + (4*(L1 + L2 + (-lm - lp)/2.0 - ov))/L2 -
(3*pow(L1 + L2 + (-lm - lp)/2.0 - ov,2))/pow(L2,2)) + ((6*(L1 + L2 + (-lm - lp)/2.0 - ov))/pow(L2,2) - (6*pow(L1 + L2 + (-lm - lp)/2.0 -
ov,2))/pow(L2,3))*yp2;

y2mid = ym2*((3*pow(L1 + L2 + (-lm - lp)/2.0 - ov,2))/pow(L2,2) - (2*pow(L1 + L2 + (-lm - lp)/2.0 - ov,3))/pow(L2,3)) - am2*(-
(pow(L1 + L2 + (-lm - lp)/2.0 - ov,2)/L2) + pow(L1 + L2 + (-lm - lp)/2.0 - ov,3)/pow(L2,2)) - ap2*(L1 + L2 + (-lm - lp)/2.0 - (2*pow(L1 +
L2 + (-lm - lp)/2.0 - ov,2))/L2 +   pow(L1 + L2 + (-lm - lp)/2.0 - ov,3)/pow(L2,2) - ov) + (1 - (3*pow(L1 + L2 + (-lm - lp)/2.0 - ov,2))/
pow(L2,2) + (2*pow(L1 + L2 + (-lm - lp)/2.0 - ov,3))/pow(L2,3))*yp2;

y2average = (ym2*((3*pow(L1 + L2 - lm - ov,2))/pow(L2,2) - (2*pow(L1 + L2 - lm - ov,3))/pow(L2,3)) -   am2*(-(pow(L1 + L2 - lm
- ov,2)/L2) + pow(L1 + L2 - lm - ov,3)/pow(L2,2)) +   ym2*((3*pow(L1 + L2 - lp - ov,2))/pow(L2,2) - (2*pow(L1 + L2 - lp - ov,3))/
pow(L2,3)) -   am2*(-(pow(L1 + L2 - lp - ov,2)/L2) + pow(L1 + L2 - lp - ov,3)/pow(L2,2)) -   ap2*(L1 + L2 - lm - (2*pow(L1 + L2 - lm -
ov,2))/L2 + pow(L1 + L2 - lm - ov,3)/pow(L2,2) - ov) -   ap2*(L1 + L2 - lp - (2*pow(L1 + L2 - lp - ov,2))/L2 + pow(L1 + L2 - lp - ov,3)/
pow(L2,2) - ov) +   (1 - (3*pow(L1 + L2 - lm - ov,2))/pow(L2,2) + (2*pow(L1 + L2 - lm - ov,3))/pow(L2,3))*yp2 +   (1 - (3*pow(L1 + L2 -
lp - ov,2))/pow(L2,2) + (2*pow(L1 + L2 - lp - ov,3))/pow(L2,3))*yp2)/2.0;

// intermediate variables used in the expression of cap, forces and moments
c1 = ang1mid - ang2mid;
c2 = (y1average + y1mid)/2.0 + (-y2average - y2mid)/2.0;
c3 = c2 + gap + c1*(L1 - ov/2.0 - xmid);

// rotation angle of the rotated electrical field lines
```

```
theta0 = (ang1mid + ang2mid)/2.0;

intFy0 = 'eps0*thickness*vlt_sqrt/pow(cos(theta0),2)/2.0;

// total capacitance obtained through integration in overlapping region

cap = 'eps0*thickness*(ov*(240*pow(c3,4) + 20*pow(c1,2)*pow(c3,2)*pow(ov,2) + 3*pow(c1,4)*pow(ov,4))*cos(theta0))/
(240.0*pow(c3,5));

// electrostatic force in x-direction in rotated frame of reference

Fxd_l = -('eps0*thickness*vlt_sqrt*((ov*cos(theta0))/c3 + (pow(c1,2)*pow(ov,3)*cos(theta0))/(12.0*pow(c3,3)) +
(pow(c1,4)*pow(ov,5)*cos(theta0))/(80.0*pow(c3,5))))/(2.0*ov);

// electrostatic forces/moments in y-direction in rotated frame of reference

Fa1_l_e = (intFy0*pow(ov,3)*(560*pow(c3,4)*(2*L1 - ov) + 112*c1*pow(c3,3)*(5*L1 - 3*ov)*ov +
28*pow(c1,2)*pow(c3,2)*(12*L1 - 7*ov)*pow(ov,2) + 8*pow(c1,3)*c3*(21*L1 - 13*ov)*pow(ov,3) + 5*pow(c1,4)*(18*L1 -
11*ov)*pow(ov,4)))/(1120.0*pow(c3,6)*pow(L1,3));

Ma1_l_e = (intFy0*pow(ov,3)*(560*pow(c3,4)*(4*L1 - 3*ov) + 112*c1*pow(c3,3)*(10*L1 - 9*ov)*ov +
84*pow(c1,2)*pow(c3,2)*(8*L1 - 7*ov)*pow(ov,2) + 24*pow(c1,3)*c3*(14*L1 - 13*ov)*pow(ov,3) + 15*pow(c1,4)*(12*L1 -
11*ov)*pow(ov,4)))/(6720.0*pow(c3,6)*pow(L1,2));

Fb1_l_e = (intFy0*ov*(-112*c1*pow(c3,3)*(5*L1 - 3*ov)*pow(ov,3) + 8*pow(c1,3)*c3*pow(ov,5)*(-21*L1 + 13*ov) +
560*pow(c3,4)*(2*pow(L1,3) - 2*L1*pow(ov,2) + pow(ov,3)) + 28*pow(c1,2)*pow(c3,2)*pow(ov,2)*(10*pow(L1,3) - 12*L1*pow(ov,2) +
7*pow(ov,3)) + 5*pow(c1,4)*pow(ov,4)*(14*pow(L1,3) - 18*L1*pow(ov,2) + 11*pow(ov,3))))/(1120.0*pow(c3,6)*pow(L1,3));

Mb1_l_e = -(intFy0*pow(ov,2)*(560*pow(c3,4)*(6*pow(L1,2) - 8*L1*ov + 3*pow(ov,2)) +
84*pow(c1,2)*pow(c3,2)*pow(ov,2)*(10*pow(L1,2) - 16*L1*ov + 7*pow(ov,2)) +  112*c1*pow(c3,3)*ov*(10*pow(L1,2) - 20*L1*ov +
9*pow(ov,2)) +  15*pow(c1,4)*pow(ov,4)*(14*pow(L1,2) - 24*L1*ov + 11*pow(ov,2)) +  24*pow(c1,3)*c3*pow(ov,3)*(14*pow(L1,2) -
28*L1*ov + 13*pow(ov,2))))/(6720.0*pow(c3,6)*pow(L1,2));

Fa2_l_e = -(intFy0*ov*(112*c1*pow(c3,3)*(5*L2 - 3*ov)*pow(ov,3) + 8*pow(c1,3)*c3*(21*L2 - 13*ov)*pow(ov,5) +
560*pow(c3,4)*(2*pow(L2,3) - 2*L2*pow(ov,2) + pow(ov,3)) +  28*pow(c1,2)*pow(c3,2)*pow(ov,2)*(10*pow(L2,3) - 12*L2*pow(ov,2) +
7*pow(ov,3)) +  5*pow(c1,4)*pow(ov,4)*(14*pow(L2,3) - 18*L2*pow(ov,2) + 11*pow(ov,3))))/(1120.0*pow(c3,6)*pow(L2,3));

Ma2_l_e = (intFy0*pow(ov,2)*(-560*pow(c3,4)*(6*pow(L2,2) - 8*L2*ov + 3*pow(ov,2)) -
84*pow(c1,2)*pow(c3,2)*pow(ov,2)*(10*pow(L2,2) - 16*L2*ov + 7*pow(ov,2)) + 112*c1*pow(c3,3)*ov*(10*pow(L2,2) - 20*L2*ov +
9*pow(ov,2)) - 15*pow(c1,4)*pow(ov,4)*(14*pow(L2,2) - 24*L2*ov + 11*pow(ov,2)) + 24*pow(c1,3)*c3*pow(ov,3)*(14*pow(L2,2) -
28*L2*ov + 13*pow(ov,2))))/(6720.0*pow(c3,6)*pow(L2,2));

Fb2_l_e = (intFy0*pow(ov,3)*(-560*pow(c3,4)*(2*L2 - ov) + 112*c1*pow(c3,3)*(5*L2 - 3*ov)*ov -
28*pow(c1,2)*pow(c3,2)*(12*L2 - 7*ov)*pow(ov,2) + 8*pow(c1,3)*c3*(21*L2 - 13*ov)*pow(ov,3) + 5*pow(c1,4)*pow(ov,4)*(-18*L2 +
11*ov)))/(1120.0*pow(c3,6)*pow(L2,3));

Mb2_l_e = (intFy0*pow(ov,3)*(560*pow(c3,4)*(4*L2 - 3*ov) - 112*c1*pow(c3,3)*(10*L2 - 9*ov)*ov +
84*pow(c1,2)*pow(c3,2)*(8*L2 - 7*ov)*pow(ov,2) + 15*pow(c1,4)*(12*L2 - 11*ov)*pow(ov,4) + 24*pow(c1,3)*c3*pow(ov,3)*(-14*L2 +
13*ov)))/(6720.0*pow(c3,6)*pow(L2,2));


// electrostatic forces/moments in y-direction
// transformed from rotated frame to local frame

Fa1x = cos(theta0)*0-sin(theta0)*Fa1_l_e;
Fa1y_e = sin(theta0)*0+cos(theta0)*Fa1_l_e;
Fb1x = cos(theta0)*Fxd_l-sin(theta0)*Fb1_l_e;
Fb1y_e = sin(theta0)*Fxd_l+cos(theta0)*Fb1_l_e;
Fa2x = cos(theta0)*(-Fxd_l)-sin(theta0)*Fa2_l_e;
Fa2y_e = sin(theta0)*(-Fxd_l)+cos(theta0)*Fa2_l_e;
Fb2x = cos(theta0)*0-sin(theta0)*Fb2_l_e;
Fb2y_e = sin(theta0)*0+cos(theta0)*Fb2_l_e;


// lumped damping forces and moments
```

Modeling and Simulation for Design of Suspended MEMS   313

```
intFy0_b = -0.6*visc_air*ov*finger_w_t*finger_w_t*pow(1.0/cos(theta0),3);

Fa1y_b = ddt((intFy0_b*pow(ov,3)*(560*pow(c3,5)*(2*L1-ov)+15*pow(c1,4)*gap*pow(ov,4)*(-
18*L1+11*ov)+8*pow(c1,2)*pow(c3,2)*pow(ov,2)*(-7*gap*(12*L1-7*ov)+c1*(21*L1-13*ov)*ov)+28*c1*pow(c3,3)*ov*(-6*gap*(5*L1-
3*ov)+c1*(12*L1-7*ov)*ov)-112*pow(c3,4)*(5*gap*(2*L1-ov)+c1*ov*(-5*L1+3*ov))+5*pow(c1,3)*c3*pow(ov,3)*(c1*(18*L1-
11*ov)*ov+gap*(-84*L1+52*ov))))/(1120.0*pow(c3,7)*pow(L1,3)));

Ma1_l_b = ddt((intFy0_b*pow(ov,3)*(560*pow(c3,5)*(4*L1-3*ov)+45*pow(c1,4)*gap*pow(ov,4)*(-12*L1+11*ov)-
112*pow(c3,4)*(5*gap*(4*L1-3*ov)+c1*ov*(-10*L1+9*ov))+84*c1*pow(c3,3)*ov*(c1*(8*L1-7*ov)*ov+gap*(-
20*L1+18*ov))+24*pow(c1,2)*pow(c3,2)*pow(ov,2)*(c1*(14*L1-13*ov)*ov+gap*(-
56*L1+49*ov))+15*pow(c1,3)*c3*pow(ov,3)*(c1*(12*L1-11*ov)*ov+gap*(-56*L1+52*ov))))/(6720.0*pow(c3,7)*pow(L1,2)));

Fb1y_b = ddt((intFy0_b*ov*(560*pow(c3,5)*(2*pow(L1,3)-2*L1*pow(ov,2)+pow(ov,3))-
15*pow(c1,4)*gap*pow(ov,4)*(14*pow(L1,3)-18*L1*pow(ov,2)+11*pow(ov,3))-112*pow(c3,4)*(c1*(5*L1-
3*ov)*pow(ov,3)+5*gap*(2*pow(L1,3)-2*L1*pow(ov,2)+pow(ov,3)))+28*c1*pow(c3,3)*pow(ov,2)*(6*gap*(5*L1-
3*ov)*ov+c1*(10*pow(L1,3)-12*L1*pow(ov,2)+7*pow(ov,3)))-8*pow(c1,2)*pow(c3,2)*pow(ov,2)*(c1*(21*L1-
13*ov)*pow(ov,3)+7*gap*(10*pow(L1,3)-12*L1*pow(ov,2)+7*pow(ov,3)))+5*pow(c1,3)*c3*pow(ov,4)*(4*gap*(21*L1-
13*ov)*ov+c1*(14*pow(L1,3)-18*L1*pow(ov,2)+11*pow(ov,3)))))/(1120.0*pow(c3,7)*pow(L1,3)));

Mb1_l_b = ddt(-(intFy0_b*pow(ov,2)*(560*pow(c3,5)*(6*pow(L1,2)-8*L1*ov+3*pow(ov,2))-
45*pow(c1,4)*gap*pow(ov,4)*(14*pow(L1,2)-24*L1*ov+11*pow(ov,2))- 112*pow(c3,4)*(c1*ov*(-10*pow(L1,2)+20*L1*ov-
9*pow(ov,2))+5*gap*(6*pow(L1,2)-8*L1*ov+3*pow(ov,2)))+ 84*c1*pow(c3,3)*ov*(gap*(-20*pow(L1,2)+40*L1*ov-
18*pow(ov,2))+c1*ov*(10*pow(L1,2)-16*L1*ov+7*pow(ov,2)))+ 15*pow(c1,3)*c3*pow(ov,3)*(c1*ov*(14*pow(L1,2)-
24*L1*ov+11*pow(ov,2)) -4*gap*(14*pow(L1,2)-28*L1*ov+13*pow(ov,2)))+ 24*pow(c1,2)*pow(c3,2)*pow(ov,2)*(-
7*gap*(10*pow(L1,2)-16*L1*ov+7*pow(ov,2))+c1*ov*(14*pow(L1,2)-28*L1*ov+13*pow(ov,2)))))/(6720.0*pow(c3,7)*pow(L1,2)));

Fa2y_b = ddt(-(intFy0_b*ov*(560*pow(c3,5)*(2*pow(L2,3)-2*L2*pow(ov,2)+pow(ov,3))-
15*pow(c1,4)*gap*pow(ov,4)*(14*pow(L2,3)-18*L2*pow(ov,2)+11*pow(ov,3))- 112*pow(c3,4)*(c1*pow(ov,3)*(-
5*L2+3*ov)+5*gap*(2*pow(L2,3)-2*L2*pow(ov,2)+pow(ov,3)))+ 28*c1*pow(c3,3)*pow(ov,2)*(6*gap*ov*(-
5*L2+3*ov)+c1*(10*pow(L2,3)-12*L2*pow(ov,2)+7*pow(ov,3)))+ 8*pow(c1,2)*pow(c3,2)*pow(ov,2)*(c1*(21*L2-13*ov)*pow(ov,3) -
7*gap*(10*pow(L2,3)-12*L2*pow(ov,2)+7*pow(ov,3)))+ 5*pow(c1,3)*c3*pow(ov,4)*(4*gap*ov*(-21*L2+13*ov)+c1*(14*pow(L2,3)-
18*L2*pow(ov,2)+11*pow(ov,3)))))/(1120.0*pow(c3,7)*pow(L2,3)));

Ma2_l_b = ddt(-(intFy0_b*pow(ov,2)*(560*pow(c3,5)*(6*pow(L2,2)-8*L2*ov+3*pow(ov,2))-
45*pow(c1,4)*gap*pow(ov,4)*(14*pow(L2,2)-24*L2*ov+11*pow(ov,2))- 112*pow(c3,4)*(5*gap*(6*pow(L2,2)-
8*L2*ov+3*pow(ov,2))+c1*ov*(10*pow(L2,2)-20*L2*ov+9*pow(ov,2)))+ 15*pow(c1,3)*c3*pow(ov,3)*(c1*ov*(14*pow(L2,2)-
24*L2*ov+11*pow(ov,2))+4*gap*(14*pow(L2,2)-28*L2*ov+13*pow(ov,2)))- 24*pow(c1,2)*pow(c3,2)*pow(ov,2)*(7*gap*(10*pow(L2,2)-
16*L2*ov+7*pow(ov,2))+c1*ov*(14*pow(L2,2)-28*L2*ov+13*pow(ov,2)))+ 84*c1*pow(c3,3)*ov*(c1*ov*(10*pow(L2,2)-
16*L2*ov+7*pow(ov,2))+gap*(20*pow(L2,2)-40*L2*ov+18*pow(ov,2)))))/(6720.0*pow(c3,7)*pow(L2,2)));

Fb2y_b = ddt((intFy0_b*pow(ov,3)*(-560*pow(c3,5)*(2*L2-ov)+15*pow(c1,4)*gap*(18*L2-
11*ov)*pow(ov,4)+8*pow(c1,2)*pow(c3,2)*pow(ov,2)*(7*gap*(12*L2-7*ov)+c1*(21*L2-13*ov)*ov)+112*pow(c3,4)*(5*gap*(2*L2-
ov)+c1*(5*L2-3*ov)*ov)+28*c1*pow(c3,3)*ov*(-6*gap*(5*L2-3*ov)+c1*ov*(-12*L2+7*ov))+5*pow(c1,3)*c3*pow(ov,3)*(c1*ov*(-
18*L2+11*ov)+gap*(-84*L2+52*ov))))/(1120.0*pow(c3,7)*pow(L2,3)));

Mb2_l_b = ddt((intFy0_b*pow(ov,3)*(560*pow(c3,5)*(4*L2-3*ov)+45*pow(c1,4)*gap*pow(ov,4)*(-
12*L2+11*ov)+15*pow(c1,3)*c3*pow(ov,3)*(gap*(56*L2-52*ov)+c1*(12*L2-11*ov)*ov)-112*pow(c3,4)*(5*gap*(4*L2-3*ov)+c1*(10*L2-
9*ov)*ov)+84*c1*pow(c3,3)*ov*(gap*(20*L2-18*ov)+c1*(8*L2-7*ov)*ov)+24*pow(c1,2)*pow(c3,2)*pow(ov,2)*(c1*ov*(-
14*L2+13*ov)+gap*(-56*L2+49*ov))))/(6720.0*pow(c3,7)*pow(L2,2)));


// sum up electrostatic and damping forces and moments
Fa1y = Fa1y_e + Fa1y_b;
Fb1y = Fb1y_e + Fb1y_b;
Fa2y = Fa2y_e + Fa2y_b;
Fb2y = Fb2y_e + Fb2y_b;
Ma1 = Ma1_l_e + Ma1_l_b;
Mb1 = Mb1_l_e + Mb1_l_b;
Ma2 = Ma2_l_e + Ma2_l_b;
Mb2 = Mb2_l_e + Mb2_l_b;

// contact model
slope = E*1e-6*thickness/(finger_w_t/2+finger_w_b/2);
Fy_contact = const2*slope*(-dy-gap+contact_oxide_t*2);
```

```
// sum up electrostatic, damping and contact forces
Fy_tip_1    = Fb1y - Fy_contact;
Fy_attach_1 = Fa1y ;
Fy_tip_2    = Fa2y + Fy_contact;
Fy_attach_2 = Fb2y ;


// transform forces/moments from local frame back to chip frame

    F(xb_t[0]) <+ - (cos_dc*(topology*Fb1x+(1-topology)*Fb2x)-sin_dc*(topology*Fy_tip_1-(1-
topology)*Fy_attach_2))*finger_number;
    F(xa_b[0]) <+ - (cos_dc*(topology*Fa2x+(1-topology)*Fa1x)-sin_dc*(topology*Fy_tip_2-(1-
topology)*Fy_attach_1))*finger_number;
    F(xa_t[0]) <+ - (cos_dc*(topology*Fa1x+(1-topology)*Fa2x)-sin_dc*(topology*Fy_attach_1-(1-
topology)*Fy_tip_2))*finger_number;
    F(xb_b[0]) <+ - (cos_dc*(topology*Fb2x+(1-topology)*Fb1x)-sin_dc*(topology*Fy_attach_2-(1-
topology)*Fy_tip_1))*finger_number;

    F(xb_t[1]) <+ - (sin_dc*(topology*Fb1x+(1-topology)*Fb2x)+cos_dc*(topology*Fy_tip_1-(1-
topology)*Fy_attach_2))*finger_number;
    F(xa_b[1]) <+ - (sin_dc*(topology*Fa2x+(1-topology)*Fa1x)+cos_dc*(topology*Fy_tip_2-(1-
topology)*Fy_attach_1))*finger_number;
    F(xa_t[1]) <+ - (sin_dc*(topology*Fa1x+(1-topology)*Fa2x)+cos_dc*(topology*Fy_attach_1-(1-
topology)*Fy_tip_2))*finger_number;
    F(xb_b[1]) <+ - (sin_dc*(topology*Fb2x+(1-topology)*Fb1x)+cos_dc*(topology*Fy_attach_2-(1-
topology)*Fy_tip_1))*finger_number;

    Tau(phib_t) <+ - (topology*Mb1-(1-topology)*Mb2)*finger_number;
    Tau(phia_b) <+ - (topology*Ma2-(1-topology)*Ma1)*finger_number;
    Tau(phia_t) <+ - (topology*Ma1-(1-topology)*Ma2)*finger_number;
    Tau(phib_b) <+ - (topology*Mb2-(1-topology)*Mb1)*finger_number;

    // electrical currents
    q = vlt * cap;

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

    I(vb_t,va_b) <+ topology * I_tip+(1-topology) * I_attach;
    I(va_t,vb_b) <+ (1-topology) * I_tip+topology * I_attach;


    end
    endmodule
```