Nodal Design of Actuators and Sensors (NODAS)

Technical Report

Jan E. Vandemeer

Department of Electrical and Computer Engineering
Carnegie Mellon University

May 7, 1998

Committee:
Dr. Gary K. Fedder, advisor
Dr. Tamal Mukherjee, second reader

ABSTRACT

A circuit-level methodology for simulating micromachined inertial sensors based on a hierarchical representation of microelectromechanical systems is presented. In the NODAS methodology (NOdal Design of Actuators and Sensors), various surface micromachined suspended microstructures are designed as netlists of general-purpose micromechanical beams, plates, electrostatic gaps, electrostatic comb-drives, joints, and anchors and evaluated using lumped-parameter behavioral models. NODAS provides the user with a one to one correspondence between layout and schematic, and the ability to simultaneously perform circuit level simulation on both the microelectromechanical components and the electronics in the schematic. The on-chip displacements and global position of each micromechanical component are separated in the netlist, enabling application of translation and rotation of the chip while simultaneously providing access to on-chip displacements for position sensing and electrostatic actuation. Each of the components is modeled with an Analog Hardware Description Language. Simulations of static displacements and modal frequencies of a cantilever beam, crab-leg flexure, folded-flexure resonator, capacitive accelerometer, and a vibratory-rate gyroscope are done using an ordinary differential equation solver. Simulation results agree to within 5% of finite-element analysis for displacements with small angles (less than 10˚). Simulation of a 16 kHz vibratory-rate gyroscope system with dual transresistance sense amplifiers, a demodulator and a filter illustrates the ability to perform system-level mixed-domain simulation with the NODAS methodology.

Table of Contents:

I. INTRODUCTION AND MOTIVATION

The increasing integration of microelectromechanical systems (MEMS) has pushed the demand for computer-aided design (CAD) tools to support rapid design of systems involving physical interactions between electrical, mechanical, magnetic, thermal, fluidic, and optical domains. For integrated circuits, design flow involves simulation and analysis at the system, circuit, and device (or physical) levels. Design is eased by abstracting physical layout into a schematic view which represents the circuit as an interconnected set of components. Presently, the design flow for MEMS does not include a schematic or "circuit" level analysis. This report describes a methodology for the NOdal Design of Actuators and Sensors (NODAS), which allows a designer to create MEMS designs in a schematic fashion. NODAS performs mixed-domain nodal simulation of MEMS by incorporating a hierarchical library of suspended MEMS components with in-plane motion. These models enable a novice MEMS designer to simulate complicated devices quickly and efficiently using schematic capture tools that are compatible with electrical circuit analysis.

At present, there are three methods of simulation used for MEMS: direct numerical simulation (*e.g.*, finite element analysis), signal flow analysis, and nodal analysis. Finite element analysis is both time consuming and computationally expensive for system design due to its low level of abstraction, lack of design hierarchy, and its inability to simulate multiple domains simultaneously. Signal flow analysis does not provide a one-to-one correspondence to layout due to its high level of abstraction. Previous work on higher-level MEMS simulation with nodal analysis has focused on behavioral simulation of individual devices (*e.g.*, microresonators) with abstract macromodels [1], or with eigenmode decomposition using single degree-of-freedom (DOF) elements [2][3]. This approach is suitable for evaluation of existing devices, but inhibits a top-down design flow for new devices.

Based on nodal analysis of mechanical systems [4], there has recently been an increasing amount of work to develop a design methodology using basic MEMS components modeled as sets of lumped parameter Ordinary Differential Equations (ODE's) to create schematics of MEMS [5][6][7]. In contrast to

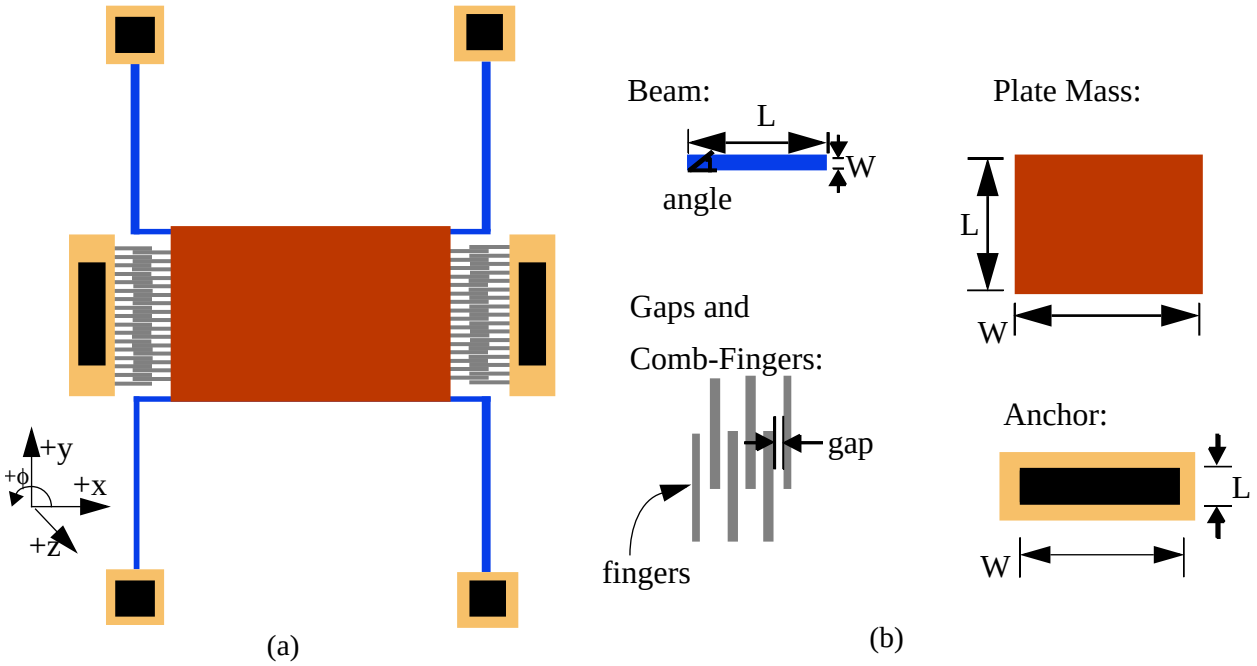


Figure 1. (a) MEMS design (capacitive accelerometer), (b) hierarchical set of MEMS components

finite element analysis, circuit-level schematics constrain simulation to local interactions between components. This enables a structured view for MEMS design using a hierarchical set of basic MEM components (*e.g.*, beam flexures, plate masses, electrostatic comb drives, electrostatic gaps, and anchors), which provide a direct linkage between the physical layout and behavioral simulation (Figure 1). In order to model the components with nonlinear ordinary differential equations in multiple degrees of freedom, and in various energy domains, an Analog Hardware Description Language (A-HDL) greatly eases model implementation (in this report, MAST [8] is used). A schematic capture package extracts a netlist from the schematic and a nonlinear ODE solver (SABER) [9] performs nodal analysis on the mixed-domain system. This methodology enables the user to perform the same types of analyses done on an electronic circuit simulator (static (DC), transient, AC, noise and Fourier). At present, implementation is restricted to suspended surface-micromachined MEMS. Suspended MEMS have no unconstrained mechanical elements, and include such devices as resonators, accelerometers, and gyroscopes.

Section II describes the general methodology of the MEMS schematic and nodal simulation,

including component descriptions and model implementations in MAST. In Section III, experiments in verification of the methodology are discussed, followed by examples of a folded-flexure resonator, capacitive accelerometer, and vibratory-rate gyroscope. The conclusion will discuss possible future directions for the research.

II. MODELING THEORY

This section discusses the theory behind modeling each of the components in the MEMS library (*e.g.,* beam flexures, plate-masses, joints, anchors, electrostatic gaps, and electrostatic comb-drives).

A. GENERAL THEORY AND CONVENTIONS

Static equilibrium dictates that the forces acting on a body must sum to zero. Likewise the sum of moments must also equal zero. These equations are called the “through” variable relations, and are analogous to Kirchoff’s Current Law in circuit theory. Nodal simulators solve for system variables by making the sum of the “through variables” flowing out of each node equal to zero. By defining component models (templates) using an A-HDL, one can create equations which relate multi-domain through variables in terms of across variables. The simulator is given an initial set of across variable values, then uses the “through” variable relations to determine the next set of across variable values. The present discussion is restricted to micromechanical simulation of in-plane motion (x, y, ϕ), with coupled electrostatic and electrical effects. Therefore in our current formulation, force in x and y, moment about ϕ, and current are through variables, while position in x and y, the angle of ϕ, and voltage are across variables.

MEMS components have a physical size and angular orientation associated with them (*e.g.*, beam flexures have length (L), width (W), and angle (Φ_{DC}), plate masses have length (L) and width (W)). These user-defined values are incorporated into the components to determine their positions. In this report “position” is defined as the position the element has with no on-chip forces applied to it (zero-force position) as shown for the crab-leg flexure in Figure 2a. Therefore, the position of a component is a reference point for any on-chip displacement that may occur. With no forces applied (thus no displacements), each component

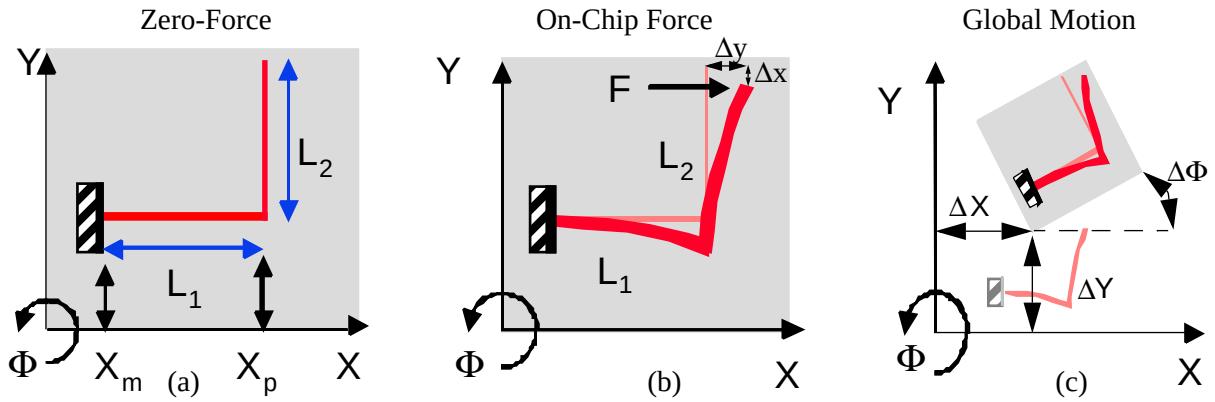


Figure 2. Physical systems of a crab-leg structure. (a) Zero-Force system determining initial positions, (b) On-Chip forces causing on-chip displacements, (c) Global motion of chip causing changes in position.

is treated as though it were a rigid body when determining its position. Simulation of inertial sensors (*e.g.*, accelerometers and gyroscopes), requires two frames of reference, the on-chip movable frame, and the global frame. The on-chip frame is needed to determine the displacements of the components relative to the chip (Figure 2b). The global frame is needed to simulate the external accelerations and rotations which cause translations and rotations of the chip relative to a stationary, or global, frame (this motion also effects the displacements of the components relative to the chip) (Figure 2c).

The convention used by nodal simulators is that when a through variable flows from the positive node to the negative node it is considered to have a positive value. Mechanical nodes associated with surfaces whose normal is directed along the +x, +y, or +z axes are considered positive (Figure 3a). The through variable going into a translational node corresponds to a force acting in the positive direction along the axis of the node. For example, the beam in tension in Figure 3b is represented by the schematic in Figure 3c. Similarly, a through variable flowing into a node is equivalent to a counterclockwise moment about the +z axis (right-hand rule).

Nodes corresponding to position are added into the components which are placed in a schematic, creating a rigid frame which defines the physical layout of the microelectromechanical system on the substrate. For example, when a horizontal beam is placed into a schematic, its left (minus) side has a rest posi-

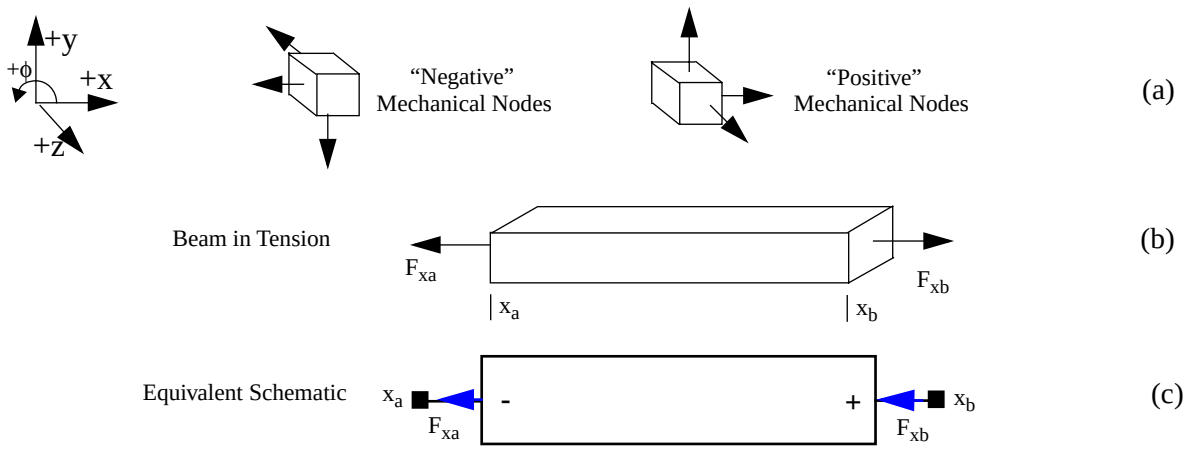


Figure 3. (a) Conventions for positive nodes on a mechanical element, (b) Physical beam undergoing tensile forces, (c) equivalent schematic representation using through variable conventions.

tion (X, Y, Φ), and its right (plus) side has a position (X + L, Y, Φ). This is also true for a vertical beam, except the beam position is oriented lengthwise from bottom to top (+y direction).

In MAST, when nodes are constrained by their through variables (*i.e.*, force sources), the template can express its "through" variable relation in terms of its across variables, or by a specified value. In contrast, when nodes are constrained by their across variables (*i.e.*, position sources), an extra variable needs to be defined to solve for the through variable between the nodes, such that the across variable is maintained. The constrained nodes then use this variable to represent their "through" variable relation from one node to the other. The problem with constraining nodes by their across variables is that each defining through variable adds an extra row and column to the Jacobian matrix, which slows simulation. Therefore, it is important to minimize number of nodes constrained this way when creating a design.

One example of a component constrained by its across variable is a voltage source (Figure 4a). To model the voltage source, a through variable, *i*, is defined such that the voltage across the source is maintained. This variable is then used to define the current flowing from one node to the other. Another example is a position source, as shown in Figure 4b. To model the "through" variable relation in the source, a through variable, *f*, is defined such that the position across the source is maintained. This variable is then used to define the force through the position source.

Figure 4. (a) Voltage source connected to a resistor, R. V defines the voltage across the source, while the extra through variable, *i*, defines the current through the source such that the voltage is maintained. (b) Position source connected to a spring, k. X defines the position across the source, while the extra through variable, *f*, defines the force through the source such that the position is maintained.

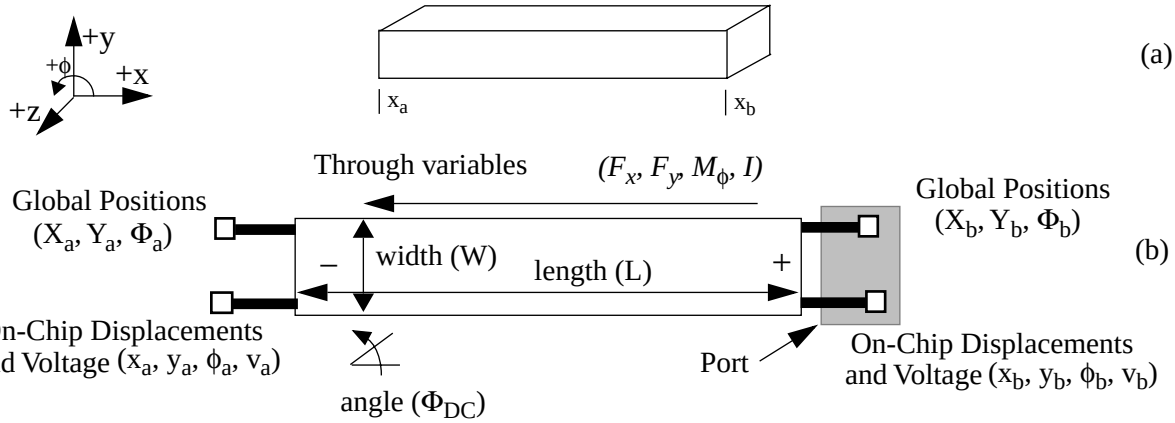


Figure 5. Beam component, (a) Physical model, (b) Schematic representation, showing global position nodes, and on-chip displacement nodes.

With two frames of reference, it is necessary to use two sets of variables to determine the complete motion of each component. One set represents the voltage and local displacement relative to the chip of the component denoted by (x, y, ϕ, v). The other set represents the global position relative to the outside environment of the component denoted by (X, Y, Φ). Global position is not affected by on-chip forces, only by external or global forces. Each set of variables is placed into a set of nodes. These sets of nodes are grouped together to create a port, (Figure 5b), which is used to connect the symbols together in a schematic. Figure 6 illustrates the physical and schematic representation of a mechanical beam subjected to an axial force F_x, a shearing force F_y, a bending moment M_ϕ, and an electrical current, I.

The remainder of this section goes through the modeling theory for each of the individual compo-

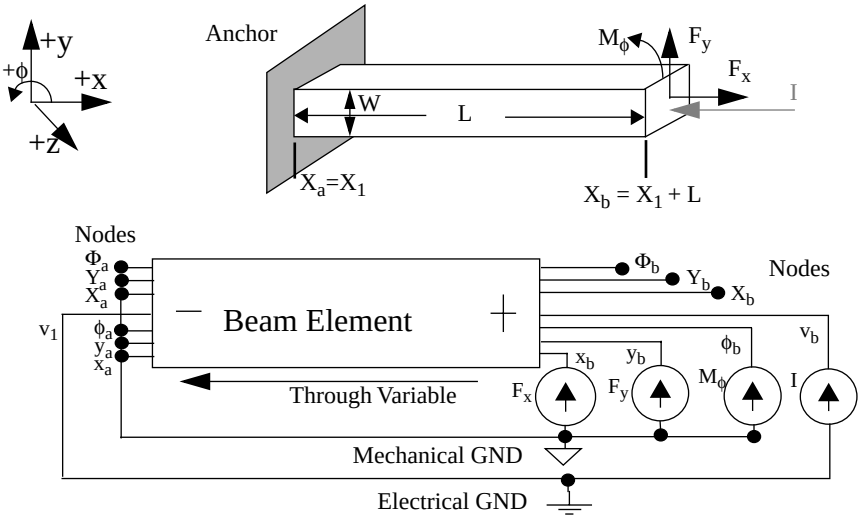


Figure 6. Mechanical element with axial tensile force, F_x, transverse force F_y, and bending moment M_ϕ, and equivalent schematic.

nents in the hierarchical library, starting with beams.

B. BEAMS

Beam flexures are placed together in a hierarchical fashion to create various flexures. Beams are parameterized by a length (*L*), width (*W*), and angle (Φ_{DC}) (Figure 5b). Their thickness, *T*, Young's modulus, *E*, and other material properties are stored in a technology file describing the process.

The rigid-body positions and angles of each end of a beam are related by

$$\boldsymbol{X}_a = \boldsymbol{X}_b + \boldsymbol{W}_G \boldsymbol{L}_b \tag{1}$$

where $\boldsymbol{X}_a$ and $\boldsymbol{X}_b$ are the position vectors at the end of the beam (in X, Y, and Φ), $\boldsymbol{W}_G$ is the global rigid-body rotation matrix, and $\boldsymbol{L}_b$ is the length vector of the beam.

$$\boldsymbol{X}_a = \begin{bmatrix} X_a \\ Y_a \\ \Phi_a \end{bmatrix}; \boldsymbol{X}_b = \begin{bmatrix} X_b \\ Y_b \\ \Phi_b \end{bmatrix}; \boldsymbol{W}_G = \begin{bmatrix} \cos\Phi_a & -\sin\Phi_a & 0 \\ \sin\Phi_a & \cos\Phi_a & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{L}_b = \begin{bmatrix} L \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

Bending and displacements are the result of an applied force or torque on the beam flexure. These displacements are measured with respect to the substrate or chip. By altering the angle parameter, beams can be placed into a schematic at any orientation. Calculating displacements across a beam is eased by transforming them from the chip frame of reference to a local frame that is independent of orientation. The

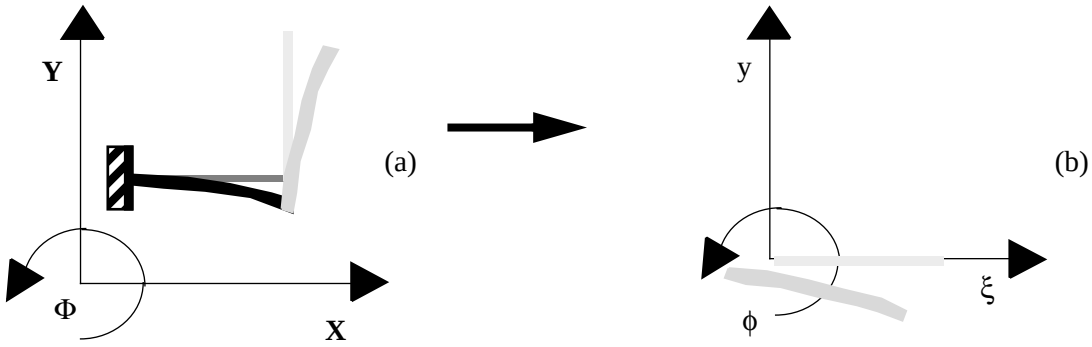


Figure 7. (a) Crab-leg flexure in chip frame of reference, (b) single beam in local frame of reference.

beam's local y - ξ frame of reference is a frame whose ξ-axis is parallel to the length of the beam (Figure 7).

The transformation of the flexure displacements from the chip frame of reference ($\boldsymbol{u}_{c,k}(t)$) to the local frame of reference, $\boldsymbol{u}_L(t)$, is

$$\boldsymbol{u}_L(t) = \boldsymbol{W}_L^{-1} \cdot \boldsymbol{u}_{c,k}(t) \tag{3}$$

where W_L is the local rotation matrix,

$$\boldsymbol{u}_{c,k}(t) = \left[x_a(t)\ y_a(t)\ \phi_a(t)\ x_b(t)\ y_b(t)\ \phi_b(t)\right]^T \tag{4}$$

$$\boldsymbol{u}_L(t) = \left[x_{al}(t)\ y_{al}(t)\ \phi_{al}(t)\ x_{bl}(t)\ y_{bl}(t)\ \phi_{bl}(t)\right]^T \tag{5}$$

$$\boldsymbol{W}_L = \begin{bmatrix} \cos\Phi_{DC} & -\sin\Phi_{DC} & 0 & 0 & 0 & 0 \\ \sin\Phi_{DC} & \cos\Phi_{DC} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\Phi_{DC} & -\sin\Phi_{DC} & 0 \\ 0 & 0 & 0 & \sin\Phi_{DC} & \cos\Phi_{DC} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Forces in the local frame are related to displacements by

$$\boldsymbol{F}_L = -\boldsymbol{k}\boldsymbol{u}_L(t) \tag{7}$$

where $\boldsymbol{k}$ is the stiffness matrix, and $\boldsymbol{F}_L$ is the force vector in the local frame.

$$\boldsymbol{F}_L = \left[F_{xa}\ F_{ya}\ M_{\phi a}\ F_{xb}\ F_{yb}\ M_{\phi b}\right]^T \tag{8}$$

If small angle approximations are made, then the displacements in the axial (ξ) direction of the

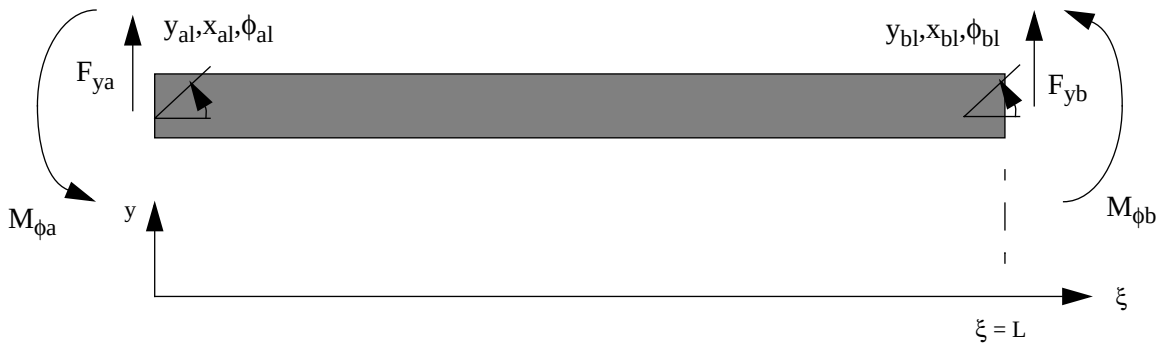


Figure 8. General beam in local frame with shearing forces and bending moments.

local frame are assumed to be independent of torsion or shear, and only dependent on axial loading. The resultant force-displacement relation is,

$$-F_{xa} = F_{xb} = \frac{EA}{L}(x_b(t) - x_a(t)) \tag{9}$$

where A is the cross-sectional area of the beam.

The shearing forces (F_{ya}, F_{yb}) and bending moments ($M_{\phi a}$, $M_{\phi b}$) that cause displacements in the local frame can be solved through fundamental beam bending theory [10], which states that for a beam undergoing bending (Figure 8), the displacements, $y_L(\xi)$, are related to the position along the beam (ξ) by,

$$\frac{d^4 y_L(\xi)}{d\xi^4} = -\left(\frac{q}{EI}\right) \tag{10}$$

$$EI\frac{d^3 y_L(\xi)}{d\xi^3} = F_s \tag{11}$$

$$EI\frac{d^2 y_L(\xi)}{d\xi^2} = M_b \tag{12}$$

$$I = \frac{Tw^3}{12} \tag{13}$$

where q is the distributed load, F_s is the shearing force, M_b is the bending moment, T is the thickness of the beam, w is the width of the beam, and I is the moment of inertia of the beam in ϕ.

For the present models, forces are constrained to being concentrated at the ends of the beam (q is zero). Solving the differential equation in (10), and including the displacements caused by axial loading,

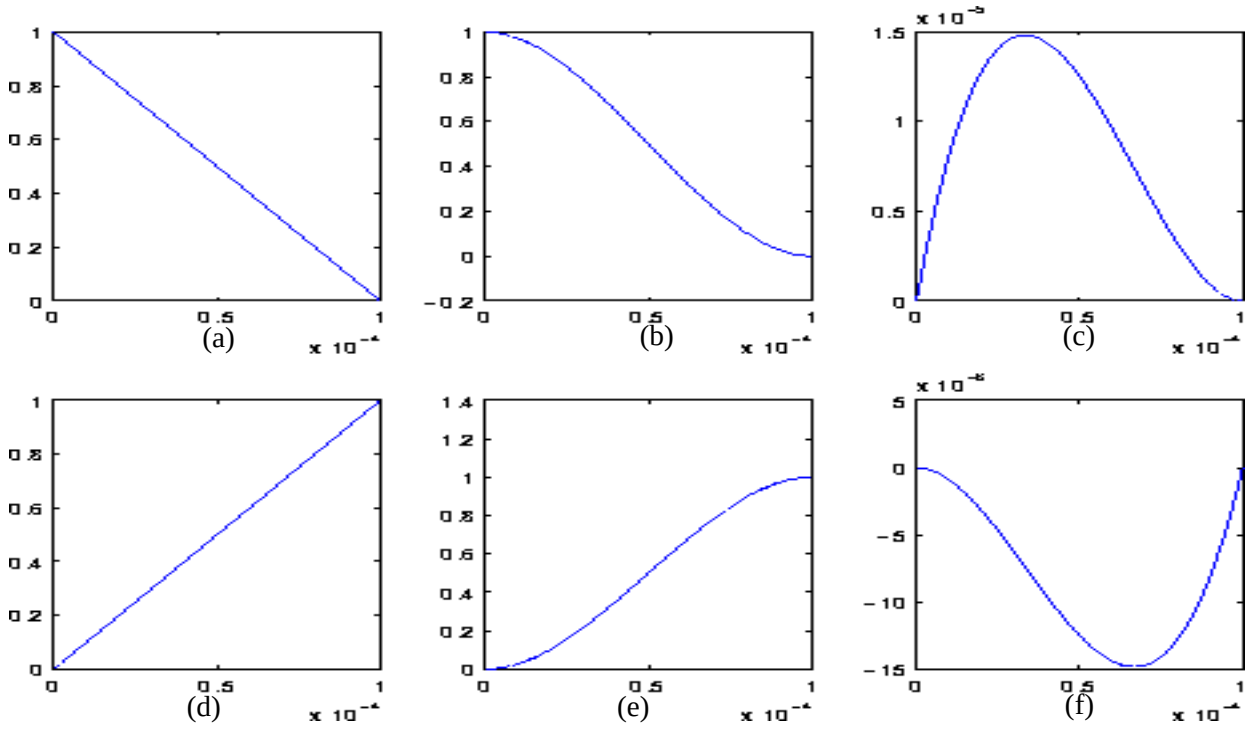


Figure 9. Shape functions for unit displacements in the x, y, and ϕ axes vs. position along the length of the beam, (a) x_a displaced, (b) y_a displaced, (c) ϕ_a displaced, (d) x_b displaced, (e) y_b displaced, (f) ϕ_b displaced.

yields the distributed displacements along the beam, $\boldsymbol{s}_L(\xi,t)$, such that,

$$\boldsymbol{s}_L(\xi, t) = \boldsymbol{a}^T(\xi)\boldsymbol{u}_L(t) \tag{14}$$

where,

$$\boldsymbol{s}_L(\xi, t) = \left[x_L(\xi, t)\ y_L(\xi, t)\right]^T \tag{15}$$

and $a(\xi)$ is a matrix of shape functions [4].

$$\boldsymbol{a} = \begin{bmatrix} 1-\frac{\xi}{L} & 0 & 0 & \frac{\xi}{L} & 0 & 0 \\ 0 & \left(1+\frac{2\xi^3}{L^3}-\frac{3\xi^2}{L^2}\right) & \left(\xi+\frac{\xi^3}{L^2}-\frac{2\xi^2}{L}\right) & 0 & \left(\frac{3\xi^2}{L^2}-\frac{2\xi^3}{L}\right) & \left(\frac{\xi^3}{L^2}-\frac{\xi^2}{L}\right) \end{bmatrix} \tag{16}$$

The shape functions (Figure 9) define the shape of the beam for different displacement boundary conditions at the end of the beam. The stiffness coefficients are determined through the force-displacement rela-

tions in (9), (11) and (12) and rearranged as the stiffness matrix, $\boldsymbol{k}$.

$$\boldsymbol{k} = \frac{EI}{L^3}\begin{bmatrix} \frac{AL^2}{I} & 0 & 0 & -\frac{AL^2}{I} & 0 & 0 \\ 0 & 12 & 6L & 0 & -12 & 6L \\ 0 & 6L & 4L^2 & 0 & -6L & 2L^2 \\ -\frac{AL^2}{I} & 0 & 0 & \frac{AL^2}{I} & 0 & 0 \\ 0 & -12 & -6L & 0 & 12 & -6L \\ 0 & 6L & 2L^2 & 0 & -6L & 4L^2 \end{bmatrix} \tag{17}$$

Displacements at the nodes of the beam component in the schematic are relative to the chip, and not relative to the local coordinates of the beam. Therefore, the forces and moments are transformed back into the chip frame of reference by multiplying the local force vector $\boldsymbol{F}_L$ by the local rotation matrix, $\boldsymbol{W}_L$.

$$\boldsymbol{F}_{c,k} = \boldsymbol{W}_L \cdot \boldsymbol{F}_L \tag{18}$$

where,

$$\boldsymbol{F}_{c,k} = \begin{bmatrix} F_{c,k,xa} & F_{c,k,ya} & M_{c,k,\phi a} & F_{c,k,xb} & F_{c,k,yb} & M_{c,k,\phi b} \end{bmatrix}^T \tag{19}$$

Each of the forces and moments are then added to their respective displacement nodes in the component (i.e., $F_{c,k\,xa}$ is subtracted from node x_a).

In addition to having a stiffness, beams also have an effective mass associated with them. Effective mass is included in the beam component by solving for the equivalent element mass matrix, $\boldsymbol{m}$, of a uniform beam and multiplying by local accelerations, $\ddot{\boldsymbol{u}}_L$(t), at the ends of the beam.

$$\boldsymbol{F}_{L,m} = -\boldsymbol{m}\ddot{\boldsymbol{u}}_L(t) \tag{20}$$

Accelerations along the beam are required to solve for the equivalent element mass matrix. Assuming a static mode shape across the beam, and neglecting rotary inertia and shear deformation, the shape functions in (16) are used to approximate acceleration along the beam.

$$\ddot{\boldsymbol{s}}_L(\xi, t) = \frac{d^2}{dt^2}(\boldsymbol{a}^T(\xi)\boldsymbol{u}_L(t)) \tag{21}$$

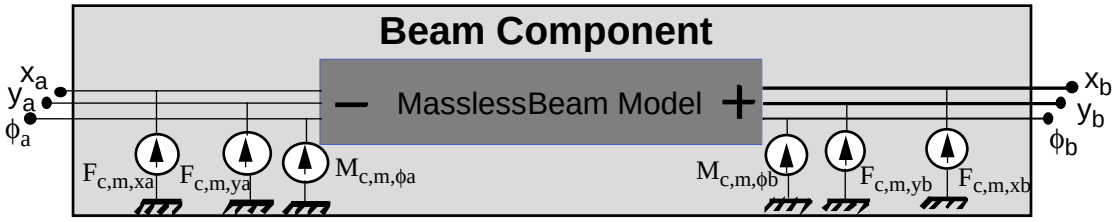


Figure 10. Final beam component model with inertial effects included.

From principles of Lagrangian dynamics,

$$\boldsymbol{F}_{L,\,m} = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{\boldsymbol{s}}_{L,\,i}}\right)\sum_{i=1}^{n}\int_{0}^{L}\frac{1}{2}\boldsymbol{m}(\boldsymbol{s}_L(\dot{\xi},\,t))^2\,d\xi \tag{22}$$

where n is the number of rows in the distributed displacement matrix, $\boldsymbol{s}_L(\xi)$, leads to [4]

$$\boldsymbol{m} = \int_{0}^{L}\rho A\boldsymbol{a}^T(\xi)\boldsymbol{a}(\xi)\,d\xi \tag{23}$$

where ρ is the material density of the beam. Substituting (16) into (23), the mass matrix is

$$\boldsymbol{m} = \frac{\rho AL}{420}\begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22L & 0 & 54 & -13L \\ 0 & 22L & 4L^2 & 0 & 13L & -3L^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13L & 0 & 156 & -22L \\ 0 & -13L & -3L^2 & 0 & -22L & 4L^2 \end{bmatrix} \tag{24}$$

The above mass matrix is then substituted into (20) to determine the local inertial forces acting on the beam. Inertial forces in the chip frame of reference are derived via the local transformation matrix, $\boldsymbol{W}_L$.

$$\boldsymbol{F}_{c,\,m} = \boldsymbol{W}_L \cdot \boldsymbol{F}_{L,\,m} \tag{25}$$

Inertial forces and moments in the chip frame of reference are then inserted at the ends of the beam to provide an effective mass and moment of inertia as shown in Figure 10.

C. PLATE MASSES

Plate masses are modeled as rigid-body elements. Flexural displacements within the structure are not modeled. Plate masses contain numerous ports, and may be configured in many ways. For brevity, we

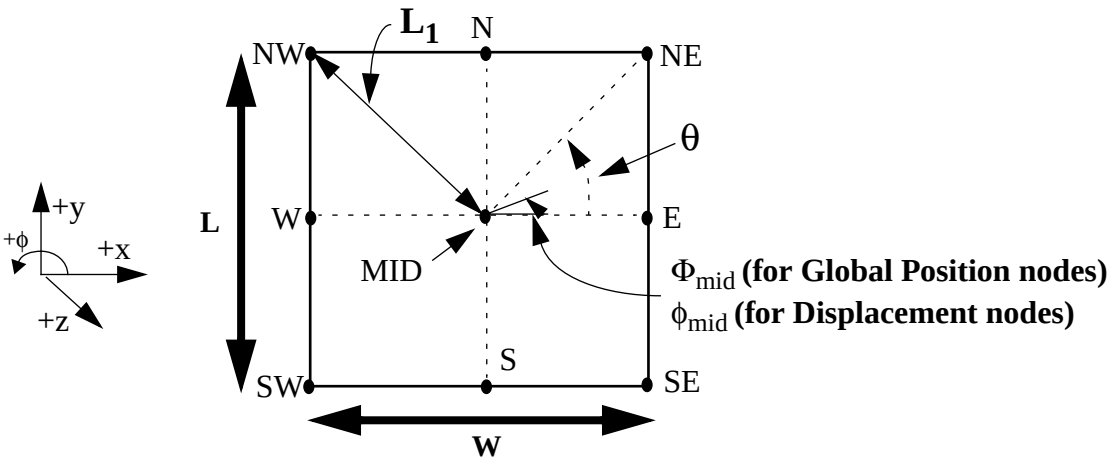


Figure 11. Various values for dimensions in calculating the displacements and positions across the plate-mass component.

will only discuss the plate-mass model with 3 ports on each side, plus one port on the top and bottom of the plate (Figure 11). Each model contains nodes at the center of mass to provide inertial and damping forces in *x*, *y*, and ϕ.

The method for determining the global position and angle at each node in the plate-mass component is similar to the method used in the beam component, except that currently, there is no angle parameter. Figure 11 illustrates the values and parameters used in the plate-mass component to constrain the positions and angles across the plate mass. For example, the trigonometric equation,

$$X_{nw} - X_{mid} = -L_1 * cos(\theta - \Phi_{mid}) \quad (26)$$

constrains the positions along the x-axis between the NW and MID nodes (X_{nw}, and X_{mid} respectively). Positions and angles between the midpoint and every other node are constrained in a similar manner.

Calculations for determining displacements at each node in a plate-mass model are simpler than in the beam model. This is due to the current plate-mass model not having an angle parameter, which restricts its orientation to a default value, and eliminates the need for a local frame of reference. As with the global position nodes, each local displacement node is constrained to maintain a defined orientation from the center of mass. The main difference between defining the displacements and positions in a plate mass is that nodes have zero displacement when there is no force acting on the plate-mass, since displacements are rel-

ative to the position in the system. For example, by examining Figure 11, the equation,

$$x_{nw} - x_{mid} = -(L/2)*cos(\theta - \phi_{mid}) + L_1*cos(\theta) \tag{27}$$

constrains the rigid-body displacements along the x-axis between the NW and MID nodes (x_{nw} and x_{mid} respectively), other displacements are constrained in a similar manner.

Inertial effects from both external and on-chip motion are included in the plate-mass model. All motions are referred to the global frame of reference to calculate the inertial force. Therefore, when transforming back to the chip frame of reference, Coriolis effects and centrifugal forces are modeled properly. Damping in the plate mass is determined by multiplying a damping factor, *B*, by local velocities. The damping models for each component are discussed in section II.H.

When determining the inertial effects of the plate mass, the on-chip displacements, $\boldsymbol{x}_{mid}$, are transformed into the global frame through $\boldsymbol{W}_G$, a global transformation matrix.

$$\boldsymbol{x}_g = \boldsymbol{W}_G \boldsymbol{x}_{mid} \tag{28}$$

where $\boldsymbol{x}_g$ is a vector representing the global displacements.

$$\boldsymbol{x}_g = \begin{bmatrix} x_{midg} \\ y_{midg} \\ \phi_{midg} \end{bmatrix}, \boldsymbol{x}_{mid} = \begin{bmatrix} x_{mid} \\ y_{mid} \\ \phi_{mid} \end{bmatrix}, \boldsymbol{W}_G = \begin{bmatrix} \cos\Phi_{mid} & -\sin\Phi_{mid} & 0 \\ \sin\Phi_{mid} & \cos\Phi_{mid} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{29}$$

Global acceleration is found by taking the second time derivative of the sum of global displacement and position. This acceleration is transformed back into the chip frame of reference and multiplied by a mass matrix, $\boldsymbol{M}$, to determine the on-chip inertial force $\boldsymbol{F}_{c,mp}$, which is included in the force equation for the nodes at the center of mass.

$$\boldsymbol{F}_{c,mp} = \boldsymbol{M}((\boldsymbol{W}_G)^{-1}(\ddot{\boldsymbol{x}}_g + \ddot{\boldsymbol{X}}_{mid})) \tag{30}$$

where,

$$\boldsymbol{M} = \begin{bmatrix} m_p & 0 & 0 \\ 0 & m_p & 0 \\ 0 & 0 & I_p \end{bmatrix} \tag{31}$$

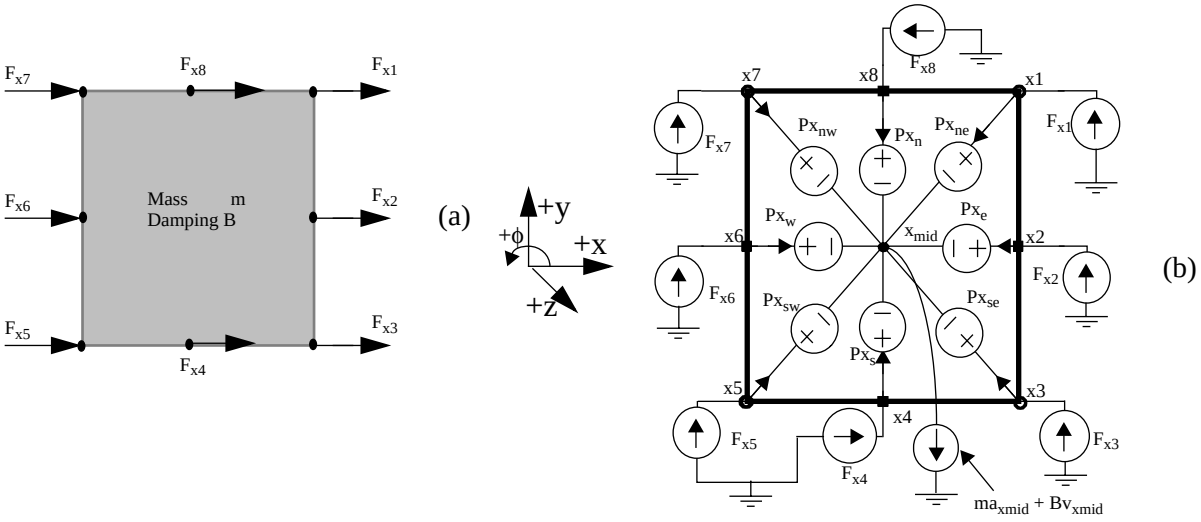


Figure 12.(a) Mechanical element with applied forces, F_{xi}, i = 1 to 8, (b) Equivalent schematic with position sources.

and

$$m_p = \rho TWL\,,\; I_p = m_p\frac{(L^2 + W^2)}{12} \tag{32}$$

In the plate mass, *L* is the length of the plate along the y-axis, *W* is the width of the plate along the x-axis, and *T* is the thickness of the plate. For rotational inertial effects, it is important to subtract the moments created by the forces along the edges of the mass.

Figure 12a shows a physical plate element with the forces applied in the +x direction. A single degree of freedom model of the plate mass is shown in Figure 12b. The inertial and damping forces leading from the center of mass must equal the sum of all the forces acting on the center of mass from the external nodes.

D. JOINTS

Joints perform two important functions. One function is to act as an angle source in the global frame of reference between two connecting components, enabling the simulator to properly measure the global angle throughout the chip. A input parameter, Θ, in the joint model determines the angle between connected components. Figure 13a illustrates the convention and formulas used by the joint to determine

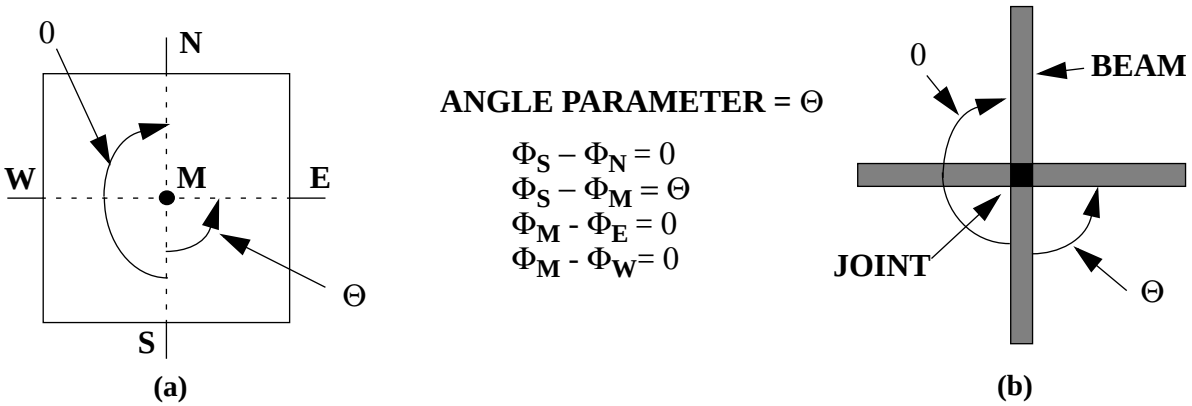


Figure 13. (a) Global angle conventions in joint model, (b) Physical structure using joint.

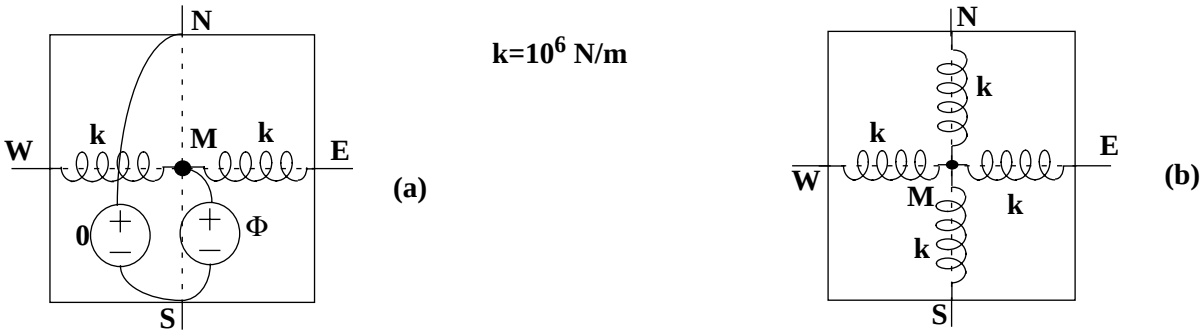


Figure 14. (a) Global angle equivalent "circuit", (b) global positions (x, and y) equivalent "circuit".

the angle. Figure 13b shows a physical structure of beams utilizing a joint. Any variation in angle shows up between the east port and south port.

The other function of the joint component is to prevent a singular Jacobian matrix from forming when other components are placed in a closed loop configuration. In all other components, the global position nodes are constrained by position or angle sources. If a schematic is created with a loop of these sources, the through variable will be undefined throughout the subcircuit, which causes the simulator to fail due to a singular Jacobian matrix. To prevent this, stiff springs are placed between global nodes in the joint model (Figure 14). This creates a small amount of compliance in the global frame of reference, which contributes a negligible error to the simulation. It is critical not to make the spring constants too large (over 10^6 N/m), to avoid an ill-conditioned matrix, which causes inaccuracies in the simulation.

The joint is modeled as a rigid body, so displacement nodes are connected directly together at the

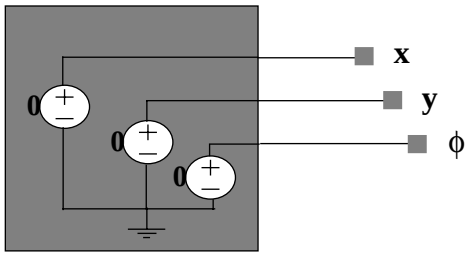


Figure 15. Anchor model, x, y, and ϕ displacements shorted to ground.

center point of this body. This is accomplished by modeling the displacement nodes as zero-value position and angle sources (i.e., short circuits), from the midpoint to the four faces of the joint. To model the electrical characteristics, the joint is assumed to have a square layout. Using Van der Pauw's method [11], the current through any two adjacent faces is equal to 4.53 times the voltage across the opposite faces.

E. ANCHORS

Anchors are used to define points where suspended structures are connected to the substrate. The displacements of connected components are zero, independent of any force or moment applied. This is accomplished by modeling the anchor as a "short circuit" to ground at each displacement node (*x*, *y*, and ϕ) (Figure 15). The anchor contains no connections for the global position, since anchors only constrain on-chip displacements. In addition, the anchor is modeled as a mechanical reference, not an electrical one, therefore there is no electrical node on the anchor. This allows the user to connect any electrical components to the micromechanical design.

F. ELECTROSTATIC GAPS

Many MEMS use electrostatic force as their main source of actuation. In addition, electrostatic forces can provide the designer with unwanted actuation or capacitance. For these reasons, it is important to model and simulate these forces. The electrostatic gap model relates the electrostatic force to the displacements between two components. Mechanical contact between two elements is also modeled.

The electrostatic gap model is a four-port device, as shown in Figure 16. Two ports on top connect to each end of a beam segment, and two ports on the bottom connect to an overlapping beam segment.

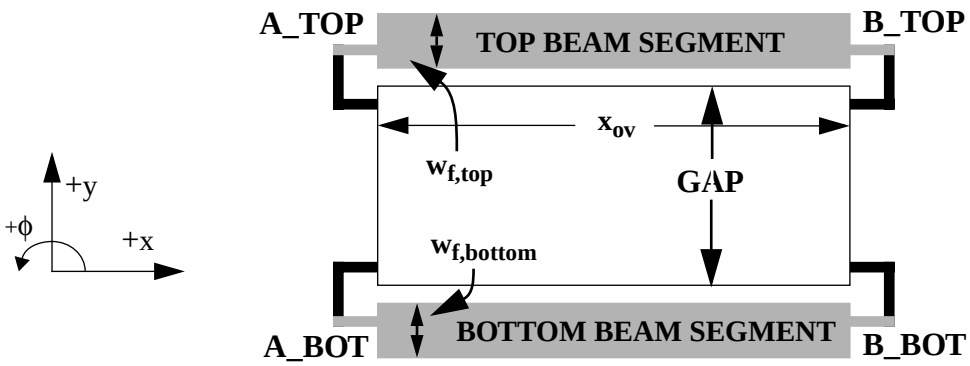


Figure 16. Symbol version of Electrostatic Gap Component. Includes all input parameters.

There are four input parameters used in determining the electrostatic gap size: the widths of the overlapping elements ($w_{f,top}$, and $w_{f,bottom}$), the distance between elements, *gap*, and the overlap of the beams, x_{ov}.

Electrostatic force is given by [12]:

$$F_{ex} = \frac{(V^2)}{2}\frac{\partial C}{\partial x} \tag{33}$$

$$F_{ey} = \frac{(V^2)}{2}\frac{\partial C}{\partial y} \tag{34}$$

where C is the capacitance between two segments, and V is the voltage between the segments. Using the parallel-plate approximation, the capacitance between two segments is,

$$C = \frac{x_{ov} T \varepsilon_0}{d} \tag{35}$$

where ε_0 is the permittivity of air, T is the thickness of the segments, whose values are defined in a technology file referred to in the models, and

$$d = gap + y_{bot} - y_{top} \tag{36}$$

is the space between the two overlapping segments. y_{top} is the y displacement at the overlapping end of the top element, and y_{bot} is the y displacement at the overlapping end of the bottom element. Via substitution of the capacitance into (33) and (34), the electrostatic forces (F_{ex} and F_{ey}) are defined.

$$F_{ex} = \frac{V^2 T \varepsilon_0}{2d} \tag{37}$$

$$F_{ey} = \frac{V^2 x_{ov} T \varepsilon_0}{2d^2} \tag{38}$$

In an electrostatic gap, when electrostatic forces become greater than the restoring spring forces acting on overlapping segments, an instability is reached causing the segments to crash into one another. The segments release again when the electrostatic forces become smaller than the restoring spring forces. This instability causes the simulator to fail due to the apparent singularity in force at contact ($d = 0$). To remedy this problem, a separate equation must be created to handle contact between the elements.

Our model approach is to treat the electrostatic gap as a very stiff spring when the distance separating the two segments becomes smaller than a threshold. To prevent the separation from becoming zero (which would cause a divide-by-zero error), an assumption is made that each element is surrounded by a thin insulating layer (perhaps modeling a native layer oxide on the surface). This thickness, T_{ox}, is used as the threshold value after which the "gap" is modeled as a stiff spring. The complete force equations are

$$F_{ex} = \frac{V^2 T \varepsilon_0}{2(c_1 d + (1 - c_1) T_{ox})} \tag{39}$$

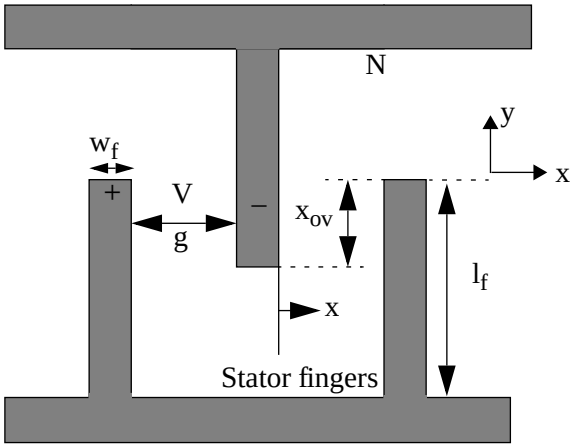
$$F_{ey} = \frac{V^2 x_{ov} T \varepsilon_0}{2((c_1 \cdot d) + (1 - c_1) T_{ox})^2} - c_2 (T_{ox} - d) \tag{40}$$

where T_{ox} is set to a thickness of 20nm. In MAST, values c_1 and c_2 are used in the force equations through the gap. These values are altered by crossing the threshold. When the electrostatic gap is not in contact, c_1 is one, while c_2 is zero. After the threshold is crossed, and the gap is modeled as a stiff spring, c_1 is set to zero, and c_2 is set to $E^* x_{ov} T/(w_{f,top}/2 + w_{f,bottom}/2)$, where E is the Young's modulus of polysilicon.

G. ELECTROSTATIC COMB-DRIVES

Electrostatic comb drives may be built from combinations of beams with interleaved electrostatic gaps. However, a macromodel of the comb drive is developed to speed simulation. To properly determine the capacitance and forces, one must input the number of rotor fingers, N, finger length, l_f, finger width, w_f, gap, g, and the overlap between the fingers, x_{ov} (Figure 17).

The capacitance is defined using parallel-plate approximations between the rotor finger and its surrounding stator fingers. The total capacitance is,



N = Number of rotor fingers
V = Voltage across fingers
x_{ov} = Overlap between fingers
l_f = length of fingers
w_f = width of fingers
g = distance between fingers

Figure 17. Interdigitated Comb-Finger. Use N of these to create an electrostatic comb-drive.

$$C = TN\varepsilon_0(x_{ov} + \Delta y)\left(\frac{1}{(g - \Delta x)} + \frac{1}{(g + \Delta x)}\right) \tag{41}$$

where Δx and Δy are the displacements in x and y. Electrostatic forces are then found using (33) and (34).

$$F_{ex} = \frac{(V^2 TN\varepsilon_0(x_{ov} + \Delta y))}{2}\left(\frac{1}{(g - \Delta x)^2} - \frac{1}{(g + \Delta x)^2}\right) \tag{42}$$

$$F_{ey} = \frac{1.12(V^2 TN\varepsilon_0 b)}{2}\left(\frac{1}{(g - \Delta x)} + \frac{1}{(g + \Delta x)}\right) \tag{43}$$

With a stator finger on both sides, the rotor finger is free to move in either the positive or negative x direction depending on the initial position of the finger. Ideally, the rotor finger is equidistant from both of the stator fingers ($\Delta x = 0$), which cancels out the electrostatic forces and motion in x. The force in y is multiplied by an empirical value of 1.12, which improves the accuracy of the force equation [13], assuming the geometry of the comb drive is that shown in Figure 17 with $w_f = g = T$. The electrical current flowing through the comb drive is equivalent to the time derivative of the charge, q.

$$I = \frac{d}{dt}(CV) = \frac{dq}{dt} \tag{44}$$

If the rotor fingers are offset from the center of the stator fingers, they are forced to move towards the closest finger. An instability is created when the electrostatic force becomes greater than the restoring force of the flexure connected to the fingers. At this point, the rotor fingers will crash into stator fingers. As mentioned in the electrostatic gap model, this instability causes simulation failure. To alleviate this prob-

lem, two separate, but continuous, functions are needed to model the comb drive.

As with the electrostatic gap model, the electrostatic comb drive is modeled as a stiff spring when the finger displacements cross a threshold (when the distance between the fingers is less than the native oxide thickness, T_{ox}). Values are used to modify the force equations to model two separate functions without creating a discontinuity.

$$F_{ex} = \frac{V^2 TN\varepsilon_0(X_{ov} + \Delta y)}{2}\left(\frac{1}{(g-(c_1\Delta x + x_{ox}))^2} - \frac{1}{(g + c_1\Delta x + x_{ox})^2}\right) - c_2(\Delta x - x_{ox}) \quad (45)$$

$$F_{ey} = \frac{1.12(V^2 TN\varepsilon_0)}{2}\left(\frac{1}{\left(g-\left(c_1\Delta x + \frac{x_{ox}}{T_{ox}}\right)\right)} + \frac{1}{\left(g+\left(c_1\Delta x + \frac{x_{ox}}{T_{ox}}\right)\right)}\right) \quad (46)$$

Similarly, the capacitance is also modified.

$$C = TN\varepsilon_0(x_{ov} + \Delta y)\left(\frac{1}{(g-(c_1\Delta x + x_{ox}))} + \frac{1}{(g+(c_1\Delta x + x_{ox}))}\right) \quad (47)$$

where c_1, c_2, s, and x_{ox} are values. T_{ox} is set to 20nm, and s is set to either positive or negative one, depending on which direction the rotor finger is moving (+x, or -x). When the comb-fingers are separated by an air gap, c_1 is set to one, while x_{ox} and c_2 are set to zero. When the fingers snap together, c_1 is set to zero, c_2 is set to $E^*x_{ov}T/w_f$, and x_{ox} is $s^*(g\text{-}T_{ox})$.

H. DAMPING MODELS FOR COMPONENTS

Each component has a model for determining the damping forces within them. Each model multiplies a damping factor, B, by a directional velocity. It is unnecessary to transform velocities into the global frame, since damping is related to the flow of air surrounding the system. Assuming the chip is packaged, the air flow surrounding the MEMS device is unaffected by the global air flow.

The current components do not model out of plane motion. Therefore, squeeze-film damping is neglected.

To determine the damping factor, we will assume that the damping forces are dominated by the Coutte flow underneath the surface of the component, which results in

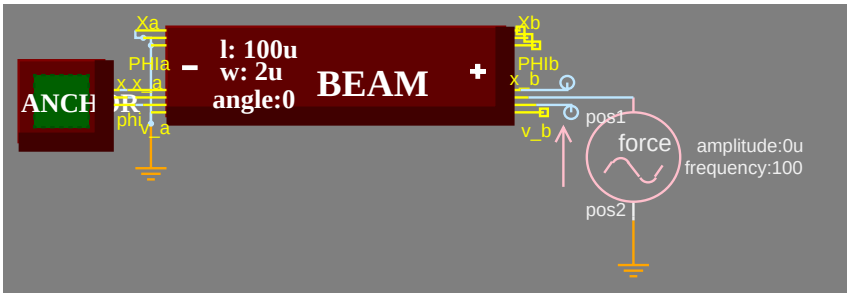


Figure 18. Schematic representation of 100μm long by 2μm wide cantilever beam with AC force source actuating it in the y-direction.

$$B = \mu \frac{A_s}{g_a} \tag{48}$$

where μ is the viscosity of air (17.9 μPa-s), A_s represents the layout areas for the component, and g_a is the air gap between the suspended structure and the substrate. For the comb-drive model, air flow between the fingers is also included, giving a damping factor of

$$B = \mu\left(\frac{A_s}{g_a} + \frac{A_c}{g}\right) \tag{49}$$

where A_c is the cross-sectional area of the fingers. To take into account edge and finite size effects, A_s and A_c are calculated after extending the widths of each component by 4μm [13][14], which approximately triples the damping factor of comb fingers, doubles the damping factor of slightly thicker beams, and does little to the damping of large plate masses.

III. SIMULATION EXPERIMENTS AND RESULTS

Various experiments were performed to verify the accuracy and speed of the components and designs created in NODAS using SABER [9] and SaberSketch [15]. The first experiment in this section compares resonant frequencies of cantilever beam models designed in both NODAS and a finite element analysis tool, ABAQUS [16]. Other experiments compare deflections in a crab-leg flexure designed in NODAS and ABAQUS, and the time taken to simulate a beam and plate-mass schematic with varying numbers and types of beam components. Another set of experiments show simulation accuracy, and ability

to simulate external accelerations and electrostatically induced comb crashing.

A. EXPERIMENTS IN ACCURACY

The first experiment regarding accuracy compares the resonant frequencies of cantilever beams created in NODAS to those created using finite element analysis, both when the cantilever is unconstrained, and when it is constrained in ϕ (*i.e.*, guided-end condition). Figure 18 shows the schematic representation of an unconstrained cantilever beam. Table 1 shows results of NODAS simulations as compared to the ABAQUS finite element simulations of 10 beam elements of type B32H (3-node quadratic beam elements in space, with hybrid formulation). Experiments were done where the beam was split into multiple smaller beams of appropriate length such that the total length of the cantilever structure is 100μm long by 2μm wide. If more than one beam component is used to create the cantilever schematic, the results are accurate to within 0.04%.

The next experiment compares the *x* and *y* displacements of a crab-leg flexure consisting of two 100μm long by 2μm wide beams actuated by a force source in the *x* direction (Figure 19). Figure 20 shows the results of the displacements in *x* and *y* for both ABAQUS and NODAS. The NODAS components are linear models, thus the force-displacement curve is a straight line. The force-displacement curve in ABAQUS exhibits a nonlinearity at large deflections. The graph shows that the NODAS simulations match within 7% when the x-deflections are less than 10μm, and 4% with deflections less than 1μm.

Table 1: Comparison of Resonant Frequencies in Cantilever Beam Models.

DESIGN	ω_{ry} (Hz) Unconstrained	ω_{ry} (Hz) Constrained in ϕ
ABAQUS	271.8kHz	432.2kHz
NODAS, 1-100μm long beam	273.0kHz	439.2kHz
NODAS, 2-50μm long beams	271.9kHz	433.9kHz
NODAS, 4-25μm long beams	271.9kHz	432.1kHz

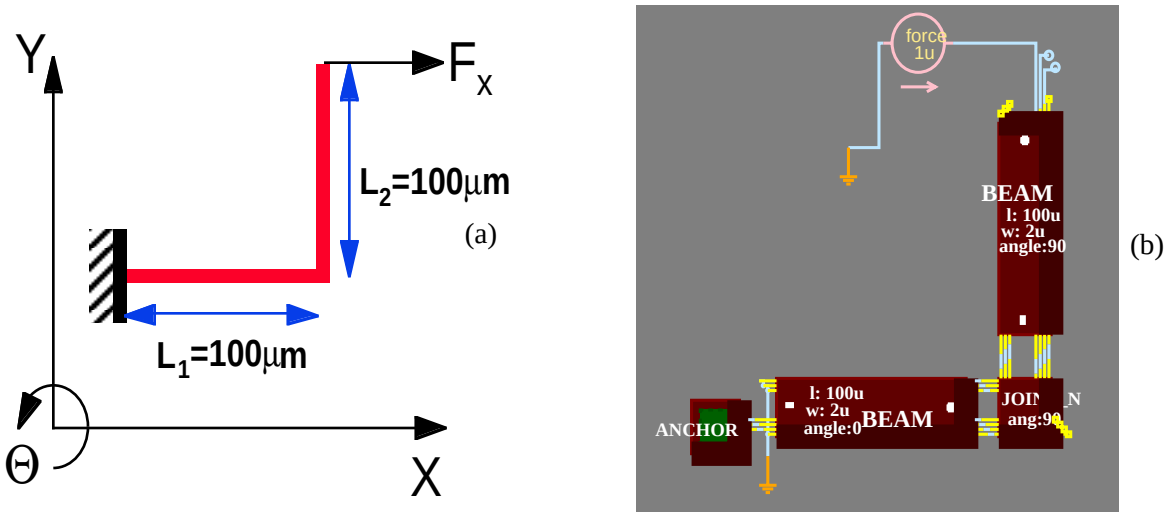


Figure 19.Crab-Leg Flexure, with force in the positive x direction, (a) Layout View, (b) SABER schematic view.

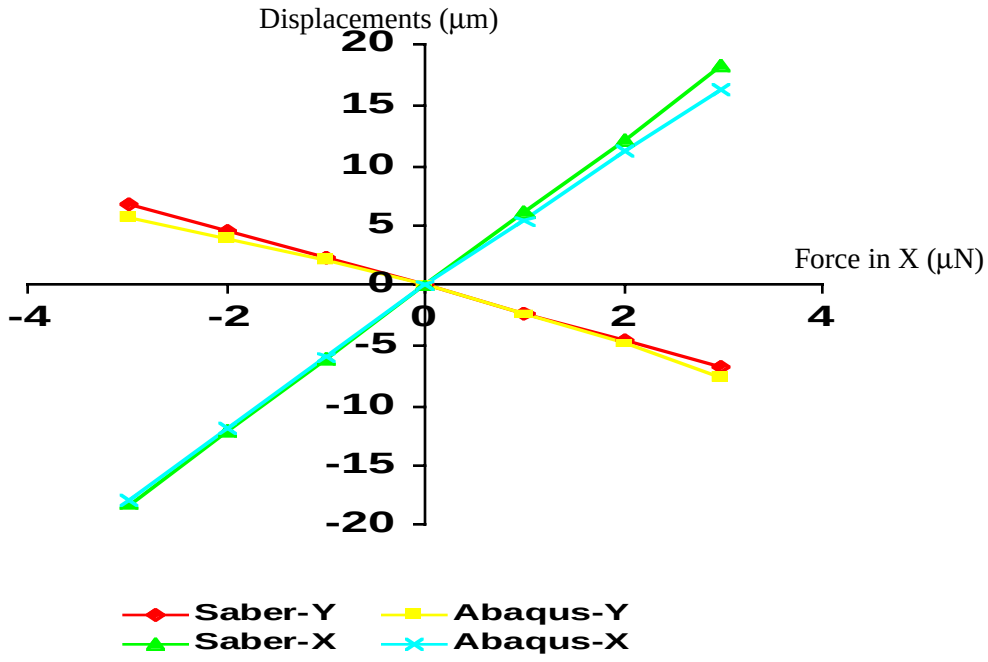


Figure 20. Graph comparing displacements vs. force for Crab-Leg structure in SABER and ABAQUS.

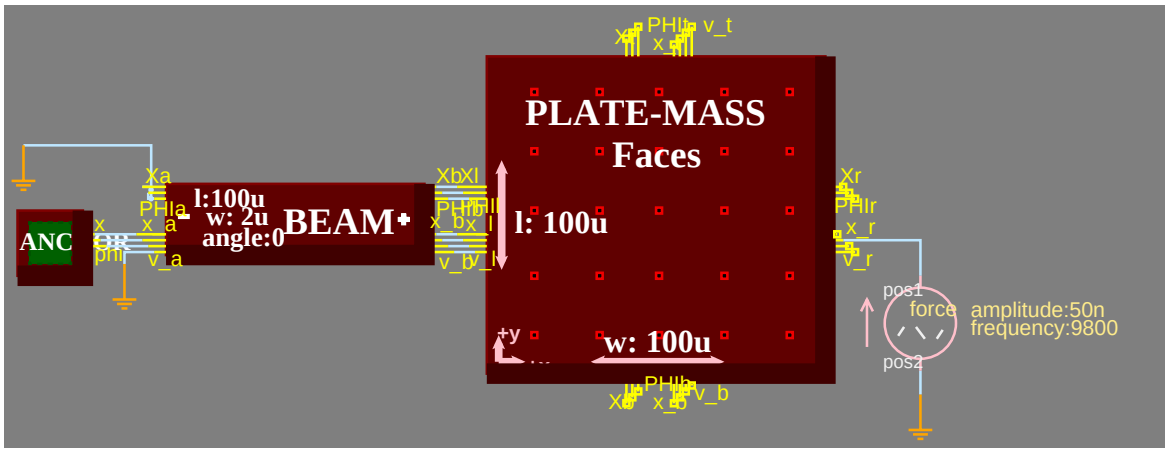


Figure 21. Schematic Representation of a Cantilever Beam Structure.

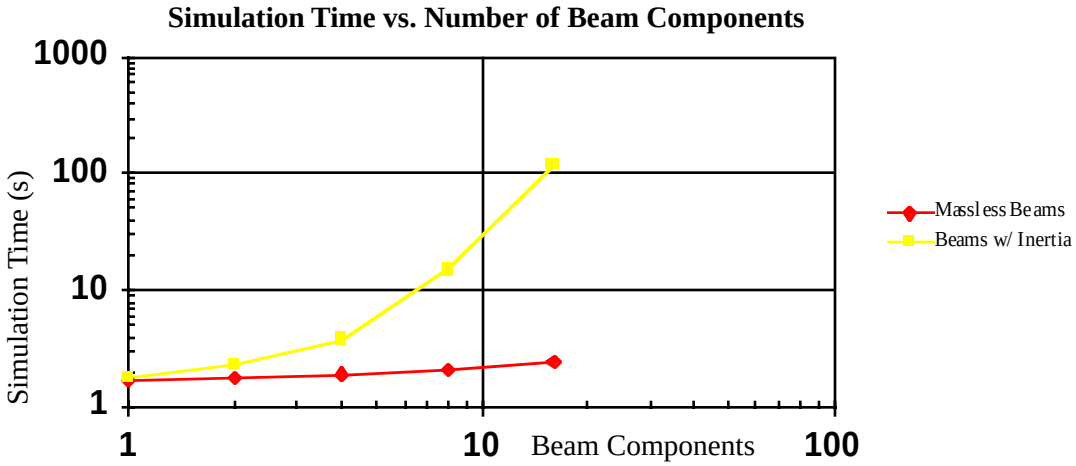


Figure 22. Graph of Simulation Time versus Beam Components in a Cantilever Beam Structure.

B. EXPERIMENTS IN SPEED OF SIMULATION

This experiment examines the simulation time of a 100μs long transient analysis for a cantilever beam structure connected to a plate mass (Figure 21). The cantilever beam consists of a varying number of beam components with various lengths such that the total length of the cantilever beam is 100μm. Transient simulation times are compared for each design in Figure 22. The simulations were run on a 300 MHz UltraSparc2 with 512MB of memory. The simulation times for cantilever structures made of beams with inertia rise rapidly with the number of components due to the extra velocity and acceleration variables solved in the simulation.

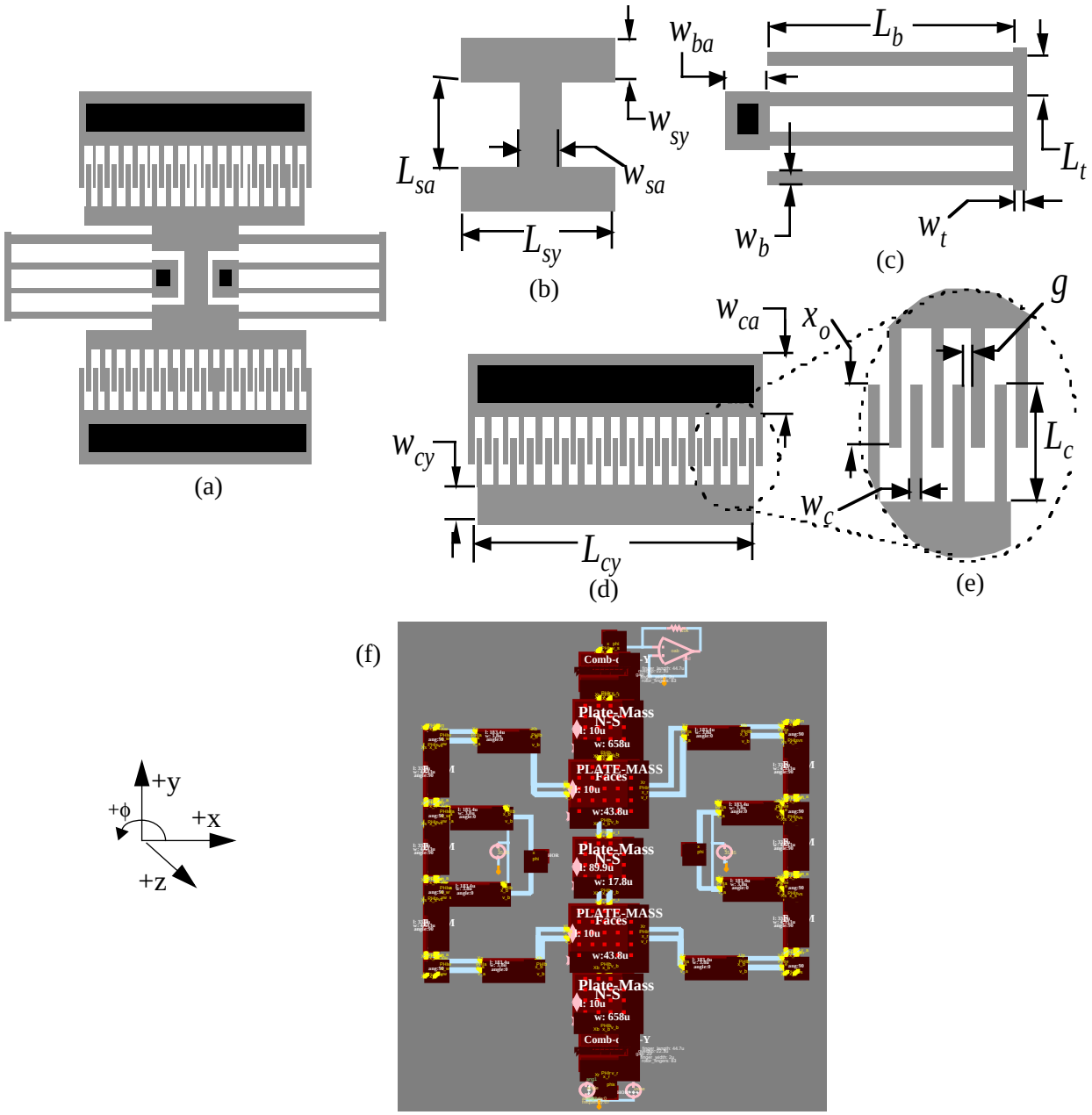


Figure 23. Folded-Flexure resonators, (a) Layout view, parameterized microresonator elements (b) shuttle mass, (c) folded flexure, (d) comb-drive with N movable rotor fingers, (e) close-up view of comb-fingers (f) schematic view.

C. FOLDED-FLEXURE RESONATOR

This experiment demonstrates the accuracy of the NODAS components by comparing the values of dominant (y-direction) resonant frequencies, and spring constants of folded-flexure resonator designs [17] calculated with NODAS to an equivalent simulation with finite element analysis (Figure 23).

Table 2: Parameters Used For Altering the Folded-Flexure Resonator Dimensions.

Case	w_b μm	L_b μm	L_t μm	w_t μm	w_{sa} μm	w_{sy} μm	L_{sy} μm	w_{cy} μm	L_{cy} μm	L_f μm	w_f μm	g μm	x_{ov} μm	N
A	2	327.7	46.8	3.276	12.1	10	38.1	687.5	46.6	73.5	2	4.2	36.8	56
B	3.3	320.9	46.8	5.711	20.9	10	46.9	578.6	10	72.2	2	2.4	36.1	66
C	4	238.7	38.9	6.282	22.9	10	48.9	690	10	55.7	2	2	27.9	87
D	3.8	183.4	33.3	4.533	17.8	10	43.8	658	10	44.7	2	2	22.3	83
E	2.6	71.8	21.5	7.177	22.1	10	48.1	466	10	22.4	2	2	11.2	59
F	2	34.5	16.3	7.030	19.8	10	45.8	210	10	12.6	2	2	6.3	27

Table 3: Comparison of Resonant Frequencies and Spring Constants of Folded-Flexure Resonators found in SABER and ABAQUS

Case	ω_{ry} Analytical	ω_{ry} SABER	ω_{ry} ABAQUS	k_y Analytical	k_y SABER	k_y ABAQUS
A	3.030kHz	3.040kHz	3.066kHz	0.1466 (N/m)	0.1462 (N/m)	0.1468 (N/m)
B	9.691kHz	10.02 kHz	10.17kHz	0.7029 (N/m)	0.6997 (N/m)	0.5619 (N/m)
C	19.66kHz	20.24 kHz	20.14kHz	3.0126 (N/m)	2.9859 (N/m)	2.9599 (N/m)
D	29.23kHz	29.95 kHz	29.77kHz	5.438 (N/m)	5.3846 (N/m)	5.3436 (N/m)
E	96.40kHz	97.76 kHz	95.38kHz	31.013 (N/m)	30.813 (N/m)	29.931 (N/m)
F	296.6kHz	300.8 kHz	291.9kHz	127.53 (N/m)	126.80 (N/m)	119.70 (N/m)

Table 2 shows the values given to parameters (Figure 23b - Figure 23e) to give the resonator a desired resonant frequency. Table 3 compares the resonant frequencies and spring constants found in ABAQUS and NODAS to analytical calculations [18] for various cases of the folded-flexure topology. A DC voltage of 50V was applied across the comb-drive to calculate the force and displacements of the resonator in NODAS. The results show that SABER simulations match to within 2% of the resonant frequencies and 20% of the spring constants found using finite element analysis. It was found that NODAS results matched much more closely to the analytically calculated values for the spring constants. The difference in the results may have come from mass in the joints, or flexibility in the plate masses that are not included in the MAST modules.

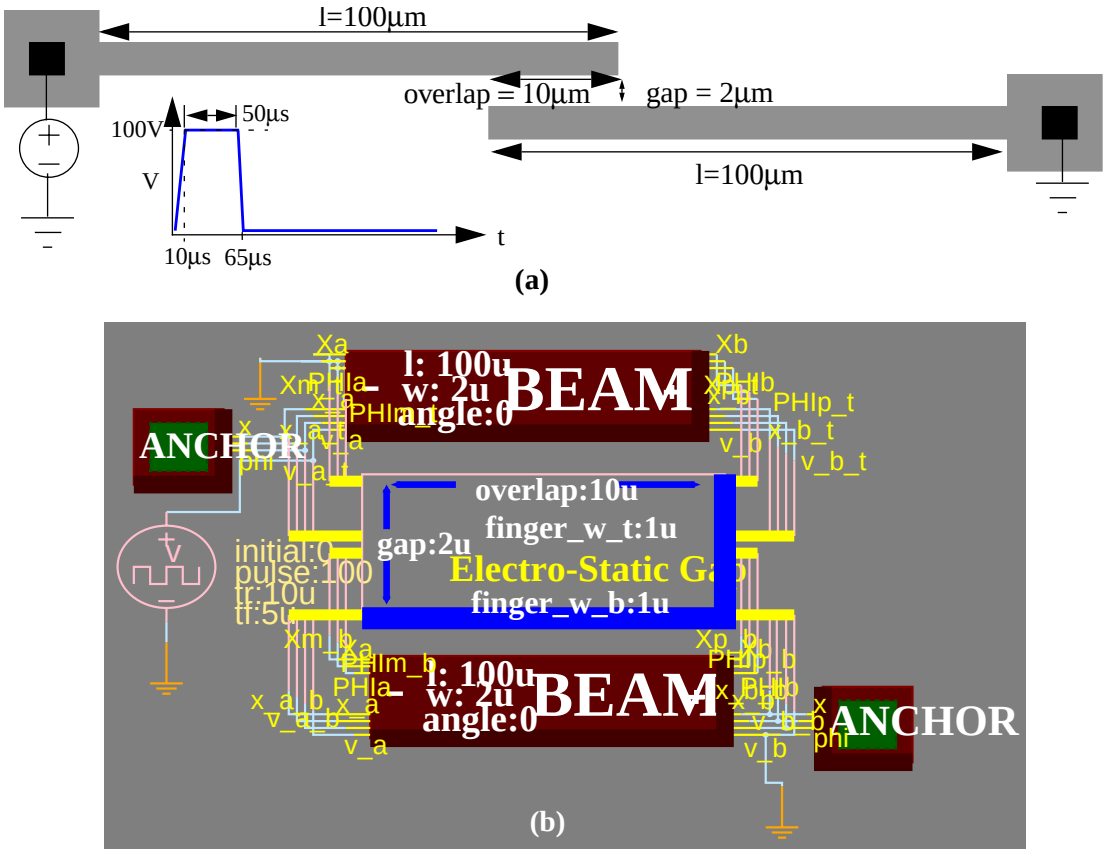


Figure 24. Electrostatic comb-finger models, (a) layout view, (b) schematic view.

D. ELECTROSTATIC COMB-FINGERS

A schematic was created to simulate comb-finger crashing between two fingers caused by electro-static attraction. In this design, one beam is anchored on its left side, and one beam is anchored on the right. Between the beams, is an air gap of 2µm. The ends of the beams overlap each other by 10µm. A pulse voltage that ramps from 0V to 100V in 10µs, with a pulse width of 50µs, and a fall time of 5µs is applied to the top beam (Figure 24).

The threshold voltage is defined as the maximum voltage applied to the beams before they snap together. At this voltage, the electrostatic force of the gap is equal to the restoring forces in the beams. Based on this condition, and assuming a parallel-plate approximation at the end of the beams, and ignoring

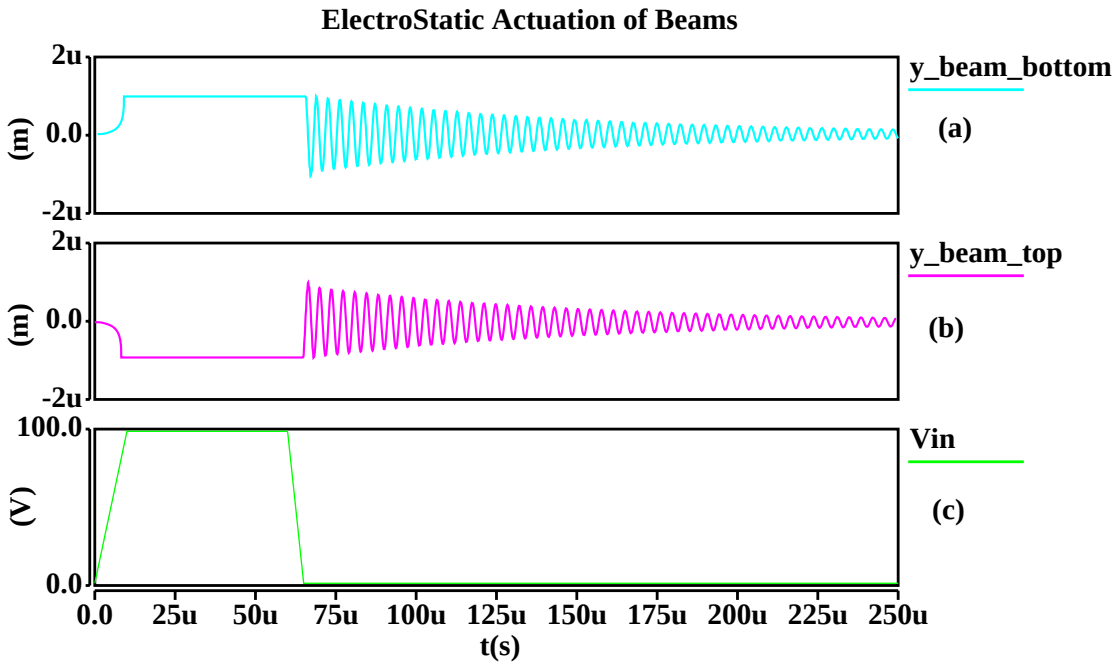


Figure 25.Transient analysis of comb finger snapping and release due to electrostatic actuation, (a) y-displacement at the left corner of the bottom beam, (b) y-displacement at the right corner of the top beam, (c) input voltage.

fringe-field effects, the threshold voltage between the beams is,

$$V_{th} = \sqrt{\frac{4kg^3}{27\varepsilon_0 T x_{ov}}} \tag{50}$$

where k is the spring constant in the flexure. The simulation results found the threshold voltage to be 66.47V, which matches the analytically calculated value exactly. Similarly, the voltage needed to release the beams after they have snapped together is,

$$V = \sqrt{\frac{k\left(g - \frac{T_{ox}}{2}\right)(T_{ox})^2}{\varepsilon_0 T x_{ov}}} \approx \sqrt{\frac{kg(T_{ox})^2}{\varepsilon_0 T x_{ov}}} \tag{51}$$

where T_{ox} is 20nm in this case. This equation yields a release voltage of 1.722V. NODAS simulations show a release voltage of approximately 1.72V.

A transient analysis with a period of 250μs was performed on the schematic in Figure 24. The displacement due to electrostatic actuation at the unconstrained ends of the beams is shown in Figures 25a and 25b. When the voltage crosses the threshold at t = 8.264μs, the beams snap together, and remain so

until the voltage falls below the release threshold. When the beams release at t = 65μs, they undergo a damped oscillation at their resonance frequency of 276.8kHz.

E. CAPACITIVE ACCELEROMETER

A capacitive accelerometer shown in Figure 26a is used to demonstrate NODAS's ability to simulate external accelerations. The accelerometer schematic uses crab-leg flexures made from beam components to suspend a rectangular plate mass. Electrostatic comb drives are placed on the left and right side of the plate mass to sense x-axis motion differentially. Transresistance amplifiers are placed at the outputs of the comb drives to sense current created by motion (Figure 26)

A sinusoidal position source, X_{in}, with amplitude 1nm and frequency ω_c drives the lower left anchor node in the global frame of reference nodes to emulate external lateral motion of the chip in the x direction. This corresponds to an external acceleration of amplitude (1nm)ω_c^2 giving the transfer function of the system a high-pass characteristic.

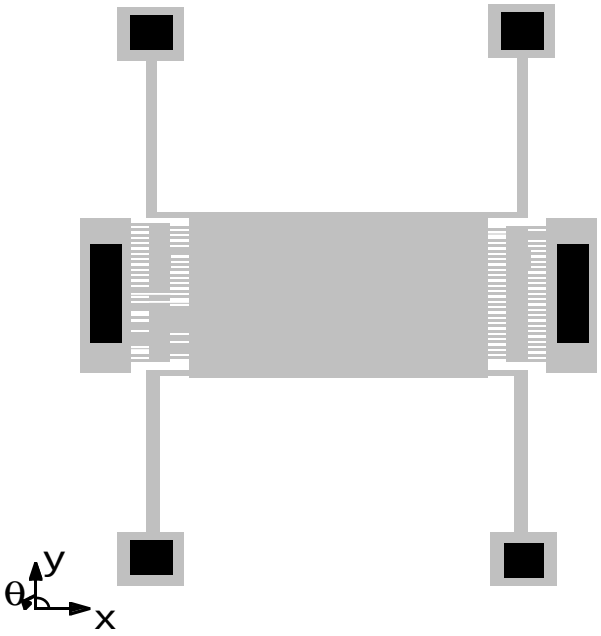
$$X_{out}(\omega_c) = X_{in}\frac{m\omega_c^2}{\sqrt{1-\left(\frac{m\omega_c}{k}\right)^2+\left(\frac{B\omega_c}{k}\right)^2}} \tag{52}$$

where m represents the effective mass of the system, B is the damping coefficient, and k is the effective spring constant. The ac analysis results in Figure 27 display a resonant frequency of 10.24 kHz. The resonant frequency was calculated as 10.39 kHz using analytical equations, and as 10.38 kHz by using finite element methods. These values match to within 2%.

F. VIBRATORY-RATE GYROSCOPE

A vibratory-rate gyroscope was simulated to demonstrate the ability to analyze relatively complex inertial microsystems using NODAS. The vibratory rate gyroscope shown in Figure 28 consists of a system of beams and masses which are designed with three-fold symmetry to match mechanical modes in the

(a)



(b)

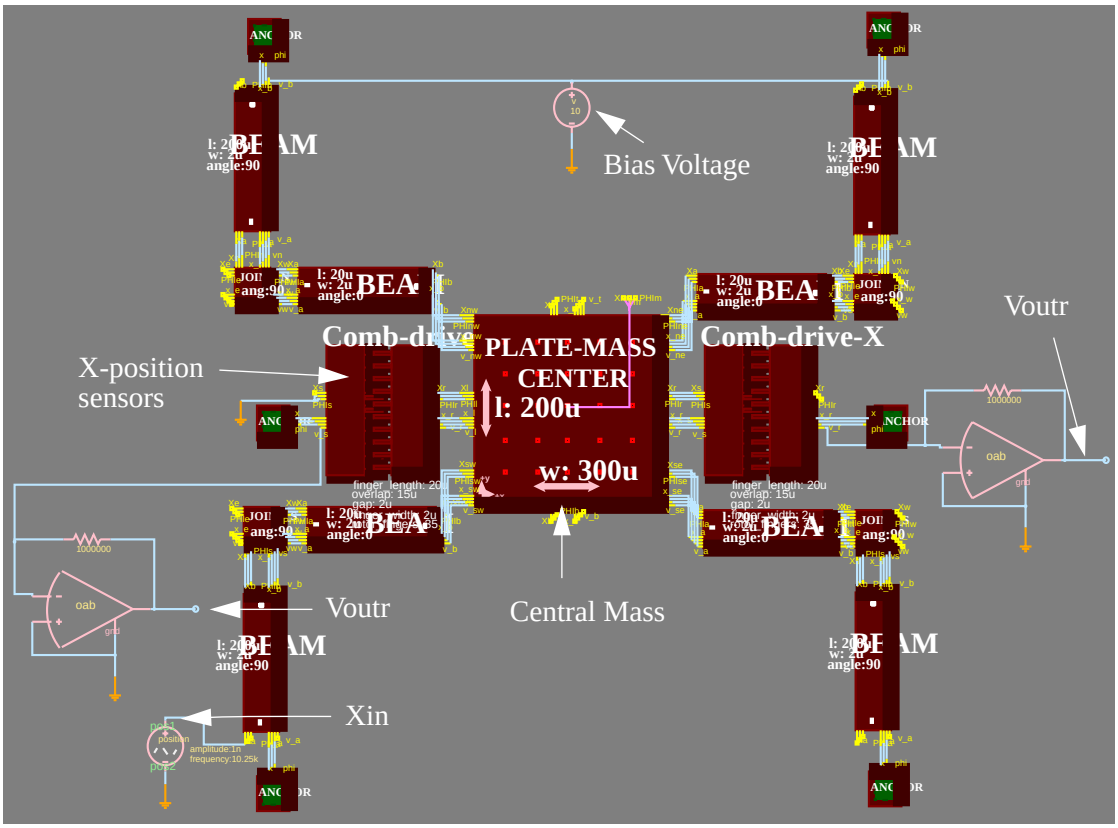


Figure 26. Crab-leg accelerometer (a) Layout view, (b) Schematic view.

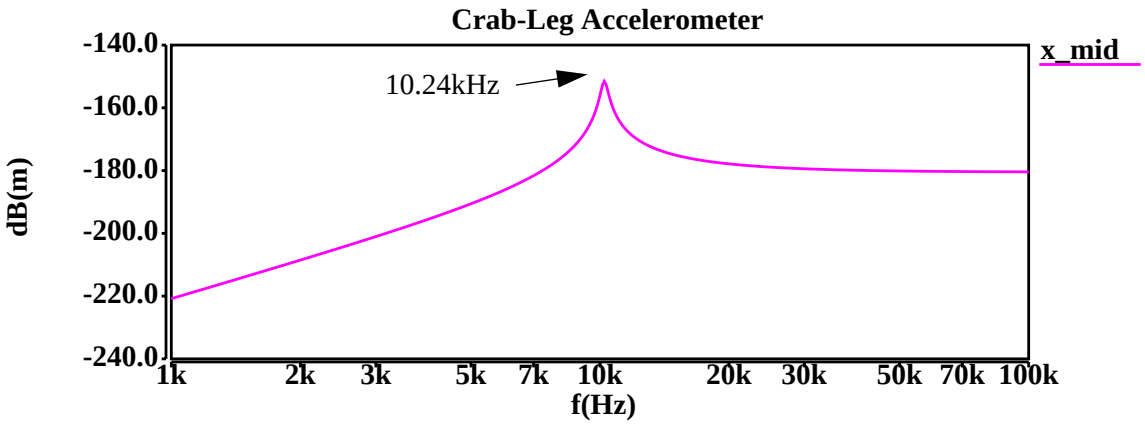


Figure 27. AC Analysis of the crab-leg accelerometer.

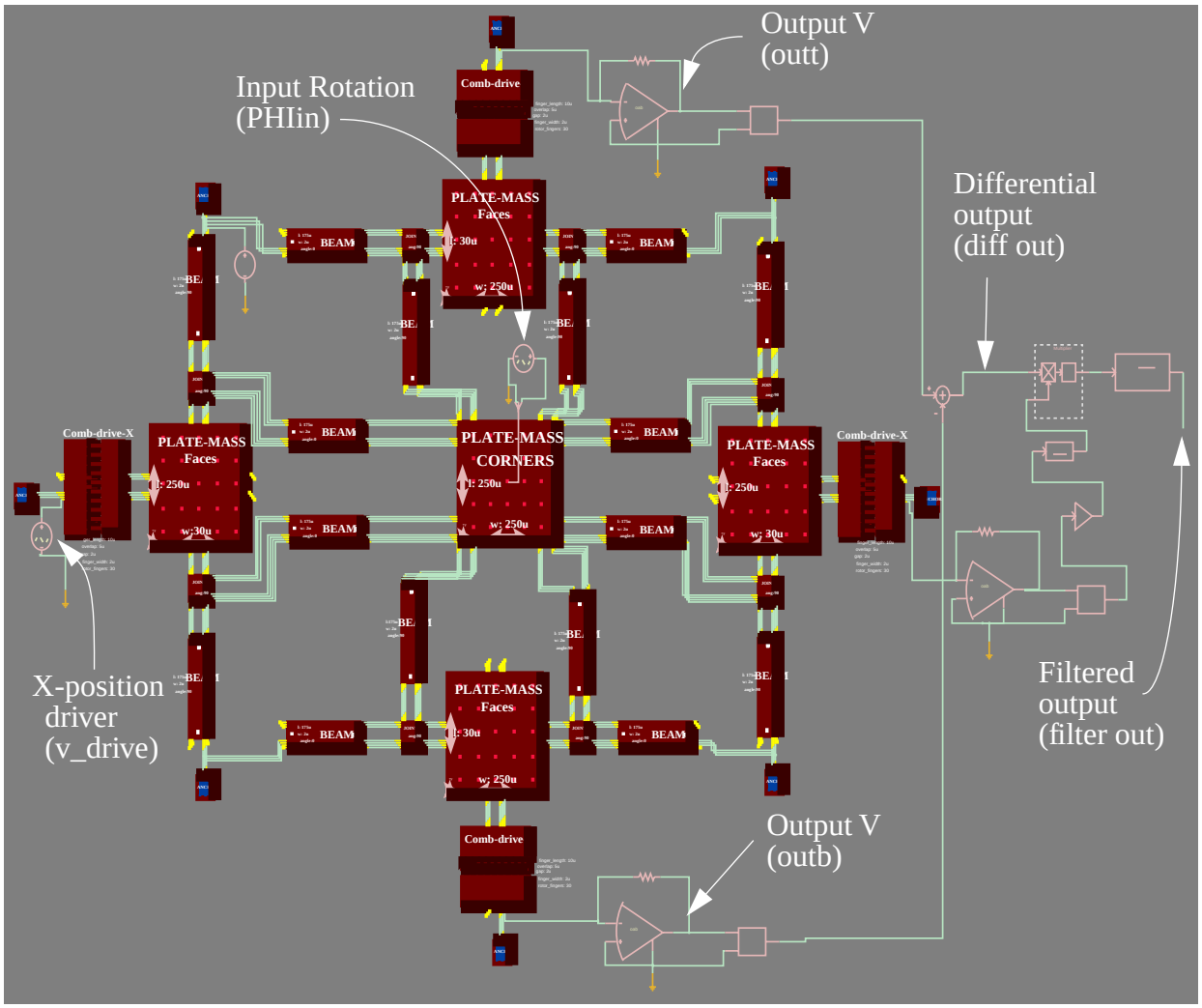
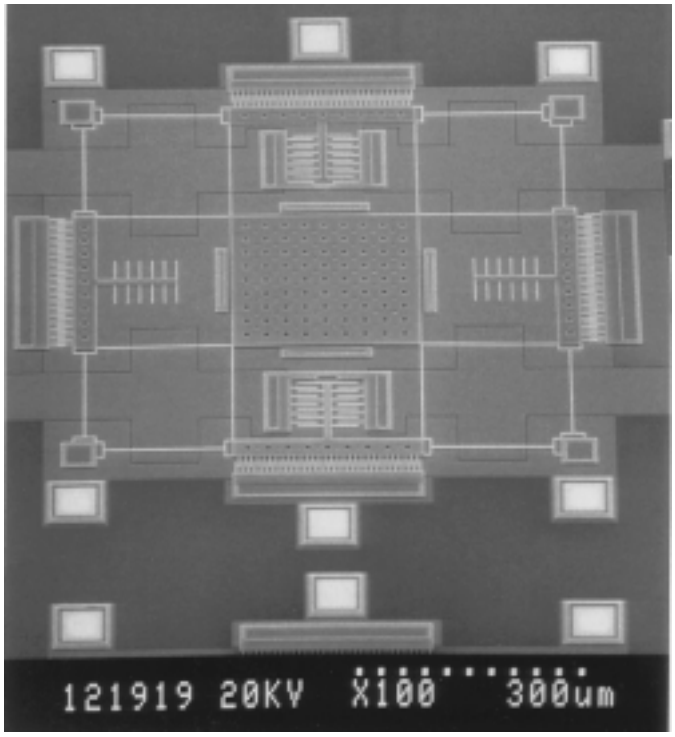


Figure 29.Equivalent schematic view of vibratory rate gyroscope.

(a)



(b)

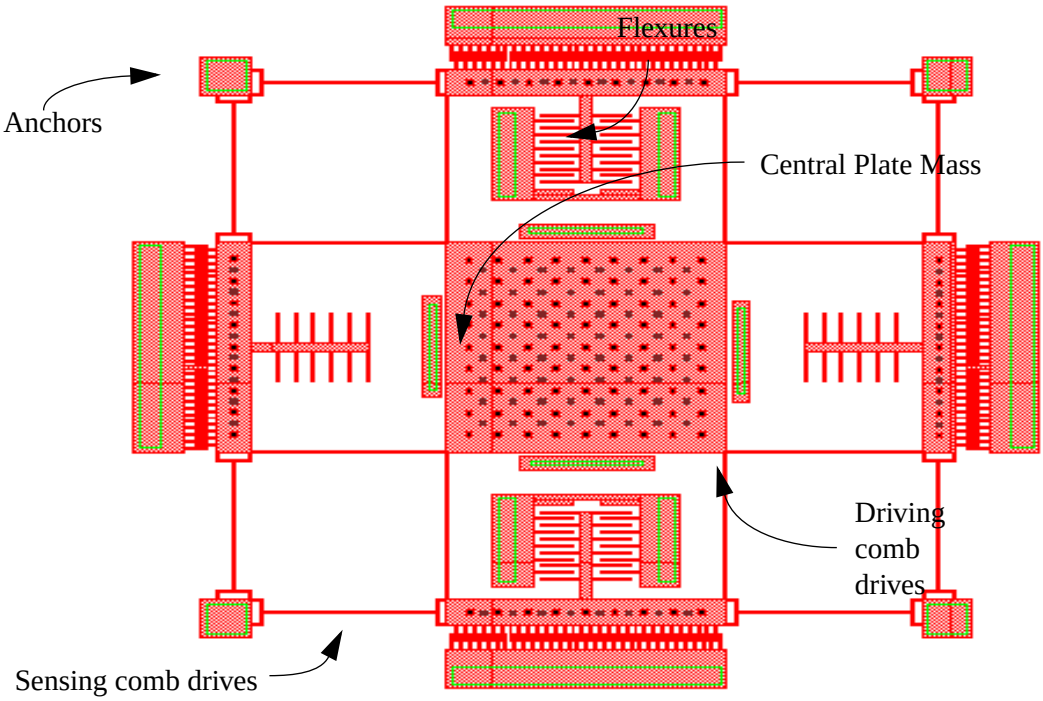


Figure 28. Vibratory Rate Gyroscope (a) SEM of polysilicon gyroscope, (b) layout.

x and *y* directions [19]. Further details of this gyroscope design and simulation can be found in [20]. The gyroscope uses these two degrees of motion to detect rotation. One direction (in this case, *x*) is chosen to be the direction in which the gyroscope is actuated. It is driven at the resonant frequency of the system to maximize displacements and increase sensitivity. External rotation in Φ generates a Coriolis force acting in the orthogonal direction (in this case, y) that is proportional to rotational rate.

An equivalent schematic of the vibratory-rate gyroscope, shown in Figure 29, was created using the beam, plate, joint, anchor, and comb-drive components. The entire movable structure is connected to a dc bias source. Differential electrostatic comb drives are located in the *x* and *y* axis. The left-hand x-axis comb actuator is driven with a sinusoidal input voltage (v_drive) operating at the mechanical resonant frequency. The right-hand x-axis comb in conjunction with a transresistance amplifier is used to detect the sinusoidal drive velocity. This signal is then used to demodulate the differential output signal.

The comb drives and transresistance amplifiers mounted on the y-axis sense motion due to the Coriolis force. An angle source (PHIin) in the global frame of reference was attached at the center of mass of the system to simulate external rotation of the chip. To verify the accuracy of the mechanical models, an AC analysis was first performed with NODAS and compared to a modal analysis done with finite elements. As shown in Table 4, the analyses match to within 1.5%. The symmetry of the design is also shown in the ac analysis by the equivalence of the *x* and *y* resonant frequencies (16.23kHz).

A transient analysis of the gyroscope is shown in Figure 30. The simulation had a period of 0.022s, with a step time of 2μs. The x-axis drive voltage (v_drive) was assigned an amplitude of 5 Vpp and a fre-

Table 4: Comparison of Methods Used to Find Resonant Frequencies of a Vibratory Rate Gyroscope.

Resonant Frequencies	SABER	ABAQUS (FEA)
ω_{rx} (Hz)	16.23k	16.00k
ω_{ry} (Hz)	16.23k	16.00k
$\omega_{r\phi}$ (Hz)	56.95k	57.1k

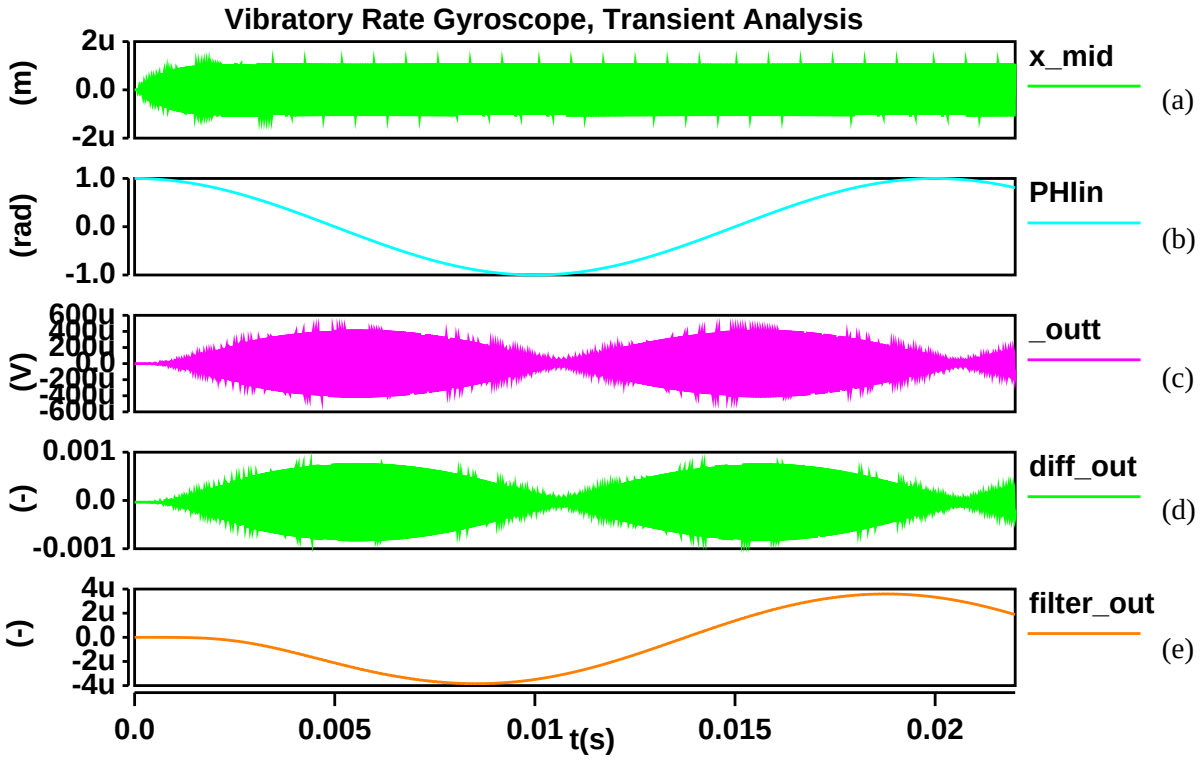


Figure 30.Transient analysis of vibratory rate gyroscope, (a) X axis motion due to input driving voltage (v_drive), (b) external rotation (PHIin), (c)top combdrive's electrical output (outt), (d) differential output signal (diff_out), (e) demodulated, and filtered output (filter_out).

quency of 16.23kHz. An external angle source (PHIin) was set to rotate the chip with an amplitude of 1 radian at a frequency of 50 Hz. A 40V dc bias was set on the movable structure to enable a motional current in the y-axis comb sensors. Figure 30a shows the envelope of the x-axis motion of the system caused by the sinusoidal input drive voltage. It took approximately 1410 seconds to run the simulation.

When the input rotation (PHIin) is applied (Figure 30b), motional current in the *y* direction arising from the Coriolis force is sensed by the upper and lower transresistance amplifiers, as seen in the output voltage in Figure 30c. In order to eliminate common-mode disturbances, the difference of the top and bottom output signals are computed and shown in Figure 30d. Finally, this signal is demodulated with the electrical output from the drive signal (v_drive) and fed through a low-pass filter to eliminate the drive harmonics (Figure 30e). The phase shift in the final output is an artifact of the low-pass filter.

IV. CONCLUSIONS

This paper describes a methodology for Nodal Design of Actuators and Sensors (NODAS) which uses a hierarchical representation of MEMS components implemented in an analog hardware description language and simulated with a nodal simulator. The NODAS design methodology will allow a designers to quickly and easily create complex MEMS designs and build up a hierarchical parts library for reuse in future designs. An example of the power of this methodology is the ability to design a fully customized microelectromechanical system with a control system and test it in a global frame of reference. In previous design methodologies it was nearly impossible to create and simulate a fully customized MEMS design with a control system at the circuit level of abstraction. NODAS also enables the user to perform all the analyses that can be done in a circuit simulator.

There are many options for future work on the components. Modeling of other degrees of freedom, *z* translation, rotation about *x* and rotation about *y,* is necessary. Component models may also include other energy domains such as thermal energy. The comb-drive and electrostatic gap components can include force-displacement and torque-rotation relations for fingers that overlap at angles relative to one another. Another advancement in the models is including parameters for stress and buckling to let the designer know if the design is manufacturable. Work may also be done to transform the entire NODAS methodology from SABER into other HDL simulators. Research can also be done on using external variables to represent the accelerations and rotations acting on the chip [21][22], which would replace the “position” nodes, thus cutting the size of the Jacobian matrix by approximately one half, speeding up simulation.

V. ACKNOWLEDGEMENTS

This research effort is sponsored by NSF CAREER Award MIP-9625471, and by the Defense Advanced Research Projects Agency (DARPA) and U. S. Air Force Research Laboratory, under agreement number F30602-96-2-0304. The U. S. Government is authorized to reproduce and distribute reprints for governmental purposes not with standing any copyright notation thereon. The views and conclusions con-

tained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U. S. Air Force Research Laboratory, or the U. S. Government.

I personally would like to thank my fiancée, Tina and the rest of my family for supporting me. I would like to thank Gary and Tamal for being patient with me, and helping me out. I would also like to thank Mike Kranz, Sitaraman Iyer, and the rest of the MEMS group. Finally, I would like to thank Darrell Teegarden and the rest of the people at ANALOGY for helping with SABER and MAST.

VI. REFERENCES

[1] J.M. Karam, B. Courtois, K. Hofmann, A. Poppe, M. Rencz, and M. Glesner, "Microsystems Modeling at System Level," *APCHDL '96*, Bangalore, India, 8-10 January, 1996.

[2] E. C. Berg, N. R. Lo, J. N. Simon, H. J. Lee, and K. S. J. Pister, "Synthesis and Simulation for MEMS Design", *ACM SIGDA Physical Design Workshop*, Reston VA, April 1996, pp. 66-70.

[3] J. Scholliers, T. Yli-Pietilä, "A SPICE-based Library for Mechatronic Systems," *Proc. IEEE Intl. Conference on Robotics and Automation*, Nagoya, Japan, 21-27 May 1995, vol. 3, pp.2847-52.

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

[5] J. E. Vandemeer, M. S. Kranz, G. K. Fedder, "Nodal Simulation of Suspended MEMS With Multiple Degrees of Freedom", *ASME Winter Annual Conference*, Dallas, TX, 16-21 November, 1997.

[6] J. E. Vandemeer, M.S. Kranz and G.K. Fedder, "Hierarchical Representation and Simulation of Micromachined Inertial Sensors", *Proc. Modeling and Simulation of Microsystems Workshop*, Santa Clara, CA, April 6-8, 1998.

[7] J. Clark, N. Zhou, S. Brown and K. S. J. Pister, "Nodal Analysis for MEMS Simulation and Design", *Proc. Modeling and Simulation of Microsystems Workshop*, Santa Clara, CA, April 6-8, 1998.

[8] MAST Reference Manual, Release 4.2, Analogy Inc., Beaverton OR, 1997.

[9] Analyzing Designs Using Saber Designer for the Cadence Design Framework II, Release 4.2, Analogy Inc., Beaverton OR, 1997.

[10] S. P. Timoshenko and J.M. Gere, *Mechanics of Materials, 2nd ed.*, Wadsworth, Belmont, 1984.

[11] R. C. Jaeger, *Introduction to Microelectronic Fabrication* (*Modular Series on Solid State Devices; volume 5)*, Addison-Wesley Publishing, Reading, Massachusetts, 1993.

[12] G. K. Fedder, *Simulation of Microelectromechanical Systems*, PhD thesis, Dept. of Electrical Engineering and Computer Sciences, University of California at Berkeley, September 1994.

[13] S. Iyer, T. Muhkerjee, and G. Fedder, "Multi-mode Sensitive Layout Syntheses of Microresonators," *Modeling and Simulation of Microsystems*, Santa Clara, CA, April 6-8, 1998.

[14] X. Zhang and W. C. Tang, "Viscous Air Damping in Laterally Driven Microresonators," *Sensors and Materials*, v. 7, no. 6, 1995, pp. 415-430.

[15] SaberDesigner Reference, SaberSketch Reference, Release 4.2, Analogy Inc., Beaverton OR, 1997.

[16] ABAQUS/Standard User's Manual, Version 5.6, Volume II, Hibbitt, Karlson and Sorenson Inc., Pawtucket RI, 1996.

[17] W. C. Tang, T.-C. H. Nguyen, M. W. Judy, and R. T. Howe, "Electrostatic Comb Drive of Lateral Pol-

ysilicon Resonators," *Sensors and Actuators A*, vol.21, no. 1-3, pp. 328-31, Feb. 1990.

[18] G. K. Fedder, and Tamal Muhkerjee, "Physical Design for Surface Micromachined MEMS", Proceedings of the 5th ACM/SIGDA Physical Design Workshop, Reston, VA, April 15-17, 1996, pp. 53-60.

[19] M. Kranz, G. K. Fedder "Micromechanical Vibratory Rate Gyroscopes Fabricated in Conventional CMOS", *Symposium Gyro Technology 1997*, Stuttgart, Germany, pp. 3.0-3.8.

[20] M. Kranz, *Design, Simulation, and Implementation of Two Novel Micromechanical Vibratory-Rate Gyroscopes*, M.S. Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1998.

[21] G. Lorenz, R. Neul "Network-Type Modeling of Micromachined Sensor Systems", *Proc. Modeling and Simulation of Microsystems Workshop*, Santa Clara, CA, April 6-8, 1998.

[22] Darrell Teegarden, G. Lorenz, R. Neul "Designing Inertial Microsensor Systems", *IEEE Spectrum*, Summer 1998.

[23]SaberDesigner Reference, Customizing SaberDesigner Using AIM, Release 4.2, Analogy Inc., Beaverton OR, 1997.

[24]SaberDesigner Reference, SaberScope Reference, Release 4.2, Analogy Inc., Beaverton OR, 1997.

APPENDIX A: SCHEMATIC IMPLEMENTATION

This appendix describes how to generate and simulate schematics using SABER [9] and SaberSketch [15]. The first part of this appendix describes what each of the nodes on the symbols represent, and how to connect them together. This is followed by an explanation of NODAS design rules, and how to create schematics in accordance with them. The final part of this section describes how to optimize schematics, and how to efficiently simulate and analyze them.

A. CREATING A MEMS SCHEMATIC IN SABER SKETCH

As mentioned earlier (refer to section II.A), each symbol contains ports consisting of two sets of nodes that interconnect with other symbols. Since MAST is not case sensitive, the labels for these nodes on the symbols and templates can not be distinguished by the letter case alone. To solve this problem, global position nodes are characterized by capital letters (X, Y, PHI) followed immediately by a descriptive term (*e.g.* a, b...). On-chip displacement and voltage nodes are characterized by a lower case letter (x, y, phi, v) followed by an underscore, and a descriptive term (*e.g.* _a, _b...) (Figure 31).

Symbols are connected together at the ports, with each set of nodes connecting to the appropriate set (*e.g.* global position to global position). The symbols were specifically designed to make this process as painless as possible, however one must be certain that the snap spacing variable in SaberSketch's [15] schematic preferences menu is set to a value less than or equal to four in order to snap the symbols together. When using beams in a schematic, the angle parameter must be equivalent to the angle the beam

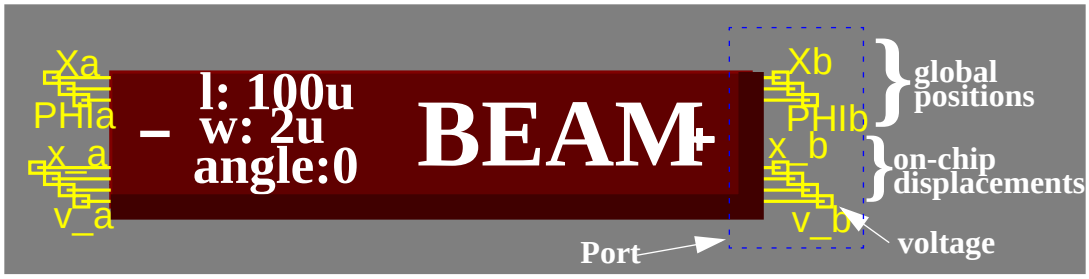


Figure 31. Beam Symbol, with ports, global position nodes, chip displacement nodes, and electrical nodes.

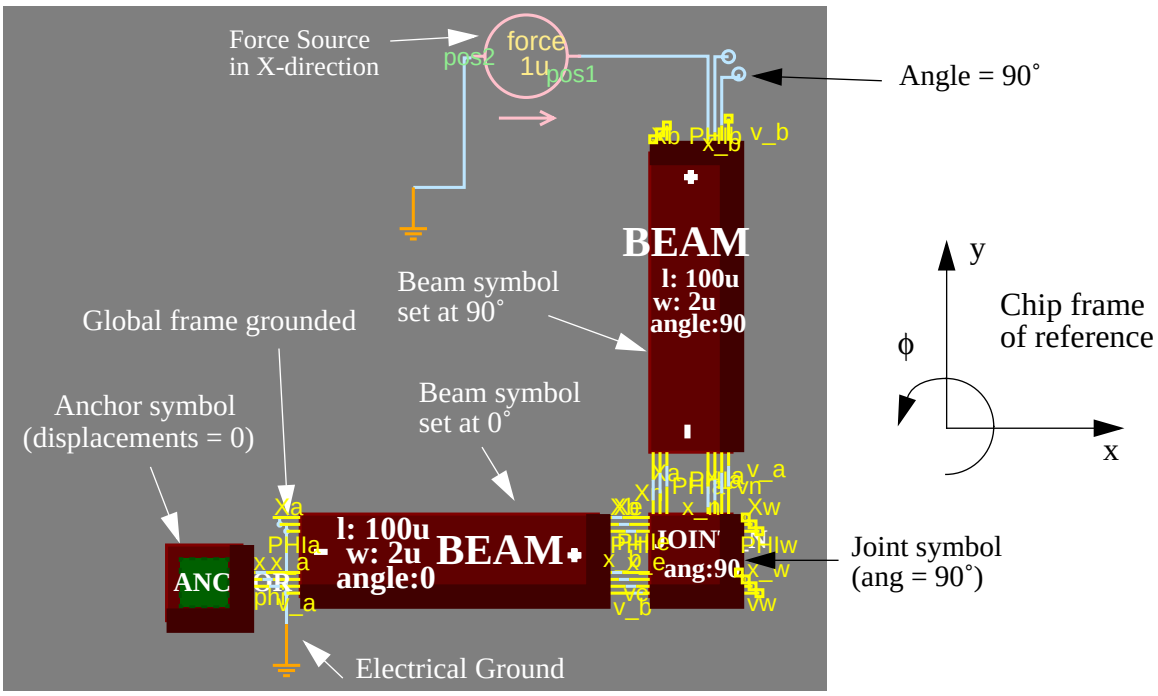


Figure 32. Schematic of Crab-leg structure. Angles must be set according to the chip frame of reference.

has relative to the chip frame of reference. In addition, when joints are used in a design, the angle parameter given to them must be consistent with adjacent elements. For example, when a rotated beam is connected to the south port on a joint, and the joint has an angle parameter of 90˚, the east and west ports contain a global angle of 0˚, thus a port with a global angle of 0˚ must be connected to those ports to make the schematic consistent (Figure 32). It is important to remember that while angle parameters are defined in degrees, the angle across the nodes is measured in radians, making it necessary to put external angle sources in terms of radians.

Currently, plate masses and comb drives do not contain an angle parameter, and therefore can not be placed in a rotated orientation on the chip. Plate-masses contain many ports at both 0˚ and 90˚ relative to the chip. It is the users decision to determine connections to the plate-mass. Their are two separate models for the comb drive, one vertical and one horizontal. Actuation in the x direction, is accomplished with a horizontal comb drive (combdrive_x), while y actuation is accomplished with a vertical comb drive (combdrive_y).

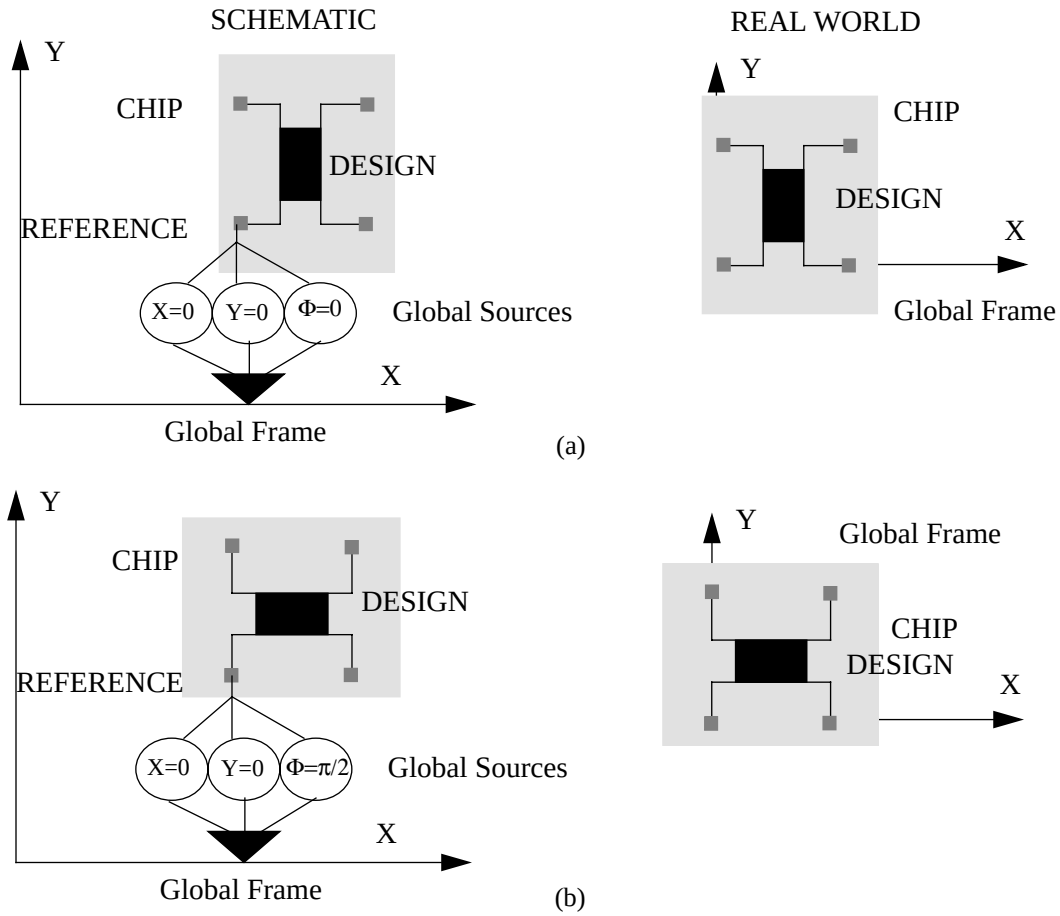


Figure 33.Comparison of schematic views and real chips in the global frame of reference, (a) Schematic with global angle reference=0, (b) Schematic with global reference angle = $\pi/2$.

The user must make certain that each schematic has a reference to the global frame on one, and only one, of their components (Figure 33). If one wishes to make a design that is stationary, a constant source must be used at the global reference node. Another important characteristic about the global frame of reference is that when elements loop together, one must be certain that the positions across all the components are defined so that the total position around the loop is zero. This is equivalent to a Kirchhoff Voltage Loop. If the positions are incorrectly defined, then the design will not simulate correctly. Finally, when referencing the chip to the global coordinate system, be certain to determine the values of the global angle and positions according to the global reference frame, not the chip's reference frame (Figure 33). There are no incorrect input values for the global position and angle references, however, evaluation may be difficult if the chip is not at the same angle and position the user thinks it is.

In the chip displacement nodes, the anchor acts as a reference node, therefore there is no need to connect them to reference. The electrical nodes must also have a reference node. If the electrical circuitry is not connected to a reference, the circuit will not be complete, resulting in a singular Jacobian matrix.

B. SIMULATING AND ANALYZING A MEMS DESIGN IN SABER

Once a MEMS schematic has been completed, the next step is simulating the design. While the process of simulating the design is simple, it can also be rather time consuming. This section will describe methods for efficient design, simulation and data analysis.

The first step in reducing simulation time is creating an efficient schematic. There are numerous methods for optimizing a schematic. One way is to make certain the schematic uses joints and plate-masses with the minimum number of connections needed to complete a design. Ports that are not connected to other components add to the size of the Jacobian matrix, but provide no functionality. Another method for reducing simulation time is to use less advanced models when the functionality of the design does not rely heavily on those components. For example, if a design has a large plate mass, the mass added by beams would not be significant, therefore if the user replaced the beam models with “massless” beams, the design still operates similarly, while the simulation time is reduced. When needed, the user can also use plate models without global acceleration effects. Simulation time is also reduced by routing all anchored ports into one anchor, which reduces the number of nodes, and sacrifices only aesthetics. An additional method for reducing simulation time is using “cosinusoidal” sources (adding a phase of 90° to sinusoidal signals). Using a cosine waveform to input sources eliminates the initial impulses in force at the first time step. This allows the simulator to start at a more accurate value, and continue the simulation more quickly.

Once the design is optimized there are some methods for reducing the simulation time via the simulator control panel [15]. If the user sets the “Monitor Progress” option on the simulator control panel to a non-zero value, he or she can monitor the simulation of their design. If the simulator is having trouble, the user can hit the “Update Probes” button on the SaberSketch menu to see the waveforms being produced by the simulator. If they do not look correct, the user can stop the simulation and examine the design, instead

of waiting for the simulation to finish. Another method used to expedite simulation time is the “Min Time Step” option under the “Integration Control” menu of the control panel. If this value is set between five and twenty times smaller than the initial time step, it prevents the simulator from taking time steps that are too small, thereby reducing the simulation time significantly. If one has a design or group of designs that needs to be simulated or analyzed, an AIM script file can be written [23]. Using a script allows the user to run a sequence of operations on multiple designs. In addition, scripts enable the user to simulate designs in batch mode. Finally, these results were obtained by setting the “Method” option in the “Integration Control” menu of the transient analysis control panel to “TRAP” instead of “GEAR”. When the “GEAR” method of integration was used, unexplained inaccuracies were encountered.

There are three ways to analyze the data collected after a simulation, through measurements, probing, and using SaberScope [24]. When the “Batch Measures” option in the Analyses menu in SaberSketch window is used, measurements can be taken on the variables calculated in the simulation of the current design. The results from these measurements appear in the SaberGuide Transcript window, they are also stored in a “.out” file which can be read by other programs. Probes can be used for a quick display of the across variable values at each node. No calculations can be done with the waveforms in the probes. The advantage of using the probes is that they give the user a quick overview of the design. A third, and most powerful, option for analyzing designs is viewing waveforms of a plotfile in SaberScope [24]. SaberScope allows the user to perform calculations and measurements (as with the Batch Measures option) on your waveforms in addition to letting the user view the waveform (as with the probes). The disadvantage of using SaberScope is that it requires the user to call up a separately running program.

APPENDIX B: MAST FILES

This appendix contains the MAST files used to model the components.

A. ANCHOR

```
#MAST Code for an Anchor
#with respect to the x, y and phi directions
#
#Jan E. Vandemeer, M. Kranz
#Carnegie Mellon University
#1/23/98
#
#***********************************************

element template anchor x y phi

translational_posy,  #reference position of x and y
                                                                    x

rotational_angphi #angle that will be maintained at reference

{
var frc_N fx, fy
var tq_Nm tphi

equations{
fx:       pos_m(x) = 0
fy:       pos_m(y) = 0
tphi:     ang_rad(phi) = 0

frc_N(x) += fx
frc_N(y) += fy
tq_Nm(phi) += tphi
}

}
```

B. BEAM

```
#       MAST Code for a Simple flexible BEAM (small deflections)
#       with respect to the x, y and phi directions
#       This model is broken up into position and displacement
#
#       Jan E. Vandemeer and Michael Kranz
#       Carnegie Mellon University. Copyright 1998, All rights reserved
#       4/24/98
#
#***********************************************

element template beam x_a x_b Xa Xb y_a y_b Ya Yb \
                              phi_a phi_b PHIa PHIb v_a v_b = l, w, angle

translational_pos       x_a, #x-displacement at port "x_a"
                        x_b,      #x-displacement at port "x_b"
                        y_a,      #y-displacement at port "y_a"
                         y_b,      #y-displacement at port "y_b"
                        Xa,       #x-position at the a port
                        Xb,       #x-position at the b port
                        Ya,       #y-position at the a port
                         Yb        #y-position at the b port

rotational_ang          phi_a,  #angle displacement at port "phi_a"
                        phi_b,  #angle displacement at port "phi_b"
                       PHIa,  #Angle of beam at a node
                       PHIb  #Angle of beam at b node

electrical             v_a,#Voltage at a node
                       v_b   #Voltage at b node

number          w,      #width of beam
                 l,      #length of beam
               angle    #Angle of beam (about z-axis)
{

#Include tech files
<tech_file.sin

number  ms,   #mass of beam (density*volume)
          I,      #moment of inertia about z-axis
          r,       #reistance through beam
          rad, #Convert degrees to radians
          B     #Damping coefficient

numberc12, c6, ea, c4, c2#Coefficients for spring constants
numbercos_dc, sin_dc     #Trig functions of the "static" angle
numberk1, k2, k3, k4       #Coefficients for effective mass
```

```
valpos_m px, py             #Position values (x,y)
valang_radphi               #Position values (phi)
valpos_mdx, dy              #Displacements across the beam in local frame
valpos_mchip_x, chip_y      #Chip frame displacements
valpos_mchip_xa, chip_ya    #Chip frame displacements at a
valpos_mchip_xb, chip_yb    #Chip frame displacements at b
valpos_ml_xa, l_ya          #Local frame displacements at a
valpos_ml_xb, l_yb          #Local frame displacements at b
valfrc_NFxda, Fyda          #Values for the displacement forces (at node a)
valfrc_NFxdb, Fydb          #Values for the displacement forces (at node b)
valfrc_NFchipxa, Fchipya    #Values for on-chip displacement forces
valfrc_NFchipxb, Fchipyb    #Values for on-chip displacement forces
valtq_NmTqa, Tqb            #Values for the torques
valfrc_NFichipxa, Fichipya  #Values for the inertial forces
valfrc_NFichipxb, Fichipyb  #Values for the inertial forces
valpos_mfil_yb, fil_ya      #Local frame inertial forces
valpos_mfil_xb, fil_xa      #Local frame inertial forces
valpos_mtql_b, tql_a        #Local frame torques

#Helpful, time cutting value
val ang_rad chip_phi  #Sum of angles across the beam

var frc_N Fxp, Fyp              #Variables for the position forces
var tq_NmTpp                    #Variables fot the angular torques
varvel_mpsVyda, Vydb            #Local Velocity in x and y
varaccl_mps2Ayda, Aydb          #Local Acceleration in x and y
varvel_mpsVxda, Vxdb            #Local Velocity in x and y
varaccl_mps2Axda, Axdb          #Local Acceleration in x and y
varw_radpsVphia, Vphib          #Angular velocity
vardw_radps2Aphia, Aphib        #Angular acceleration

parameters{
        ms = poly1_t*w*l*poly_den
          I = (poly1_t*(w**3))/12
          r = poly1_rho*(l/w)
          B = visc_air*(l*(w+4u))/(air_gap)     #Neglect skin depth
          c12 = 12*E*I/(l**3)
          c6 = 6*E*I/(l**2)
          ea = E*poly1_t*w/l
          c4  = 4*E*I/l
          c2 = 2*E*I/l

          rad = angle*PI/180
          cos_dc = cos(rad)
          sin_dc = sin(rad)
          k1 = ms/420
          k2 = ms/6
          k3 = l*k1
          k4 = 156/420
}
```

```
values{
        px = pos_m(Xb,Xa)
        py = pos_m(Yb,Ya)
        phi = ang_rad(PHIb,PHIa)
        chip_x = pos_m(x_b,x_a)
        chip_y = pos_m(y_b,y_a)
        chip_xa = pos_m(x_a)
        chip_ya = pos_m(y_a)
        chip_xb = pos_m(x_b)
        chip_yb = pos_m(y_b)

        chip_phi = ang_rad(phi_b) + ang_rad(phi_a)

#rotate into local frame
        l_xa = cos_dc*chip_xa + sin_dc*chip_ya
        l_ya = cos_dc*chip_ya - sin_dc*chip_xa
        l_xb = cos_dc*chip_xb + sin_dc*chip_yb
        l_yb = cos_dc*chip_yb - sin_dc*chip_xb
        dx = cos_dc*chip_x + sin_dc*chip_y
        dy = cos_dc*chip_y - sin_dc*chip_x

#Solve for the inertial forces and moments in the local frame
        fil_xa = k2*(2*Axda + Axdb) + B*Vxda
        fil_xb = k2*(Axda + 2*Axdb) + B*Vxdb
        fil_ya = k1*(l*(22*Aphia-13*Aphib)+ 156*Ayda +54*Aydb) +B*Vyda
        fil_yb = k1*(l*(13*Aphia-22*Aphib)+ 54*Ayda +156*Aydb) +B*Vydb
        tql_a = k3*(l*(4*Aphia-3*Aphib) + 22*Ayda +13*Aydb)
        tql_b = k3*(l*(-3*Aphia+4*Aphib) - 13*Ayda -22*Aydb)

#Solve for the forces and torques
        Fxda =  -ea*dx
        Fxdb =  ea*dx
        Fyda =  -c12*dy + c6*chip_phi
        Fydb =  c12*dy - c6*chip_phi
        Tqa = -c6*dy + c4*ang_rad(phi_a) + c2*ang_rad(phi_b)
        Tqb = -c6*dy + c4*ang_rad(phi_b) + c2*ang_rad(phi_a)

#Rotate back from local frame to CHIP frame
        Fchipxb = Fxdb*cos_dc - Fydb*sin_dc
        Fchipyb = Fydb*cos_dc + Fxdb*sin_dc
        Fchipxa = Fxda*cos_dc - Fyda*sin_dc
        Fchipya = Fyda*cos_dc + Fxda*sin_dc

#Inertial forces (in chip frame)
        Fichipxb = fil_xb*cos_dc - fil_yb*sin_dc
        Fichipyb = fil_xb*sin_dc + fil_yb*cos_dc
        Fichipxa = fil_xa*cos_dc - fil_ya*sin_dc
        Fichipya = fil_xa*sin_dc + fil_ya*cos_dc
}
```

```
equations {
#Current through the beam
        i(v_b->v_a) += v(v_b,v_a)/r

#Forces at the end of the beams
        frc_N(x_b) -= Fchipxb
        frc_N(x_a) -= Fchipxa
        frc_N(y_b) -= Fchipyb
        frc_N(y_a) -= Fchipya

#Moment at end of beam
        tq_Nm(phi_a) -= Tqa
        tq_Nm(phi_b) -= Tqb


#Effective Mass and damping
        frc_N(y_a) -= fichipya
        frc_N(y_b) -= fichipyb
        frc_N(x_a) -= fichipxa
        frc_N(x_b) -= fichipxb
        tq_Nm(phi_a) -= tql_a
        tq_Nm(phi_b) -= tql_b

#Beams positions
        frc_N(Xa->Xb) += Fxp
        frc_N(Ya->Yb) += Fyp
        tq_Nm(PHIa->PHIb) += Tpp

        Fxp:    px = l*cos(ang_rad(PHIa))
        Fyp:    py = l*sin(ang_rad(PHIa))
        Tpp:    phi = 0

#Velocities
        Vyda: Vyda = d_by_dt(l_ya)
        Vydb: Vydb = d_by_dt(l_yb)
        Vxda: Vxda = d_by_dt(l_xa)
        Vxdb: Vxdb = d_by_dt(l_xb)
        Vphia: Vphia = d_by_dt(ang_rad(phi_a))
        Vphib: Vphib = d_by_dt(ang_rad(phi_b))

#Accelerations
        Ayda: Ayda = d_by_dt(Vyda)
        Aydb: Aydb = d_by_dt(Vydb)
        Axda: Axda = d_by_dt(Vxda)
        Axdb: Axdb = d_by_dt(Vxdb)
        Aphia: Aphia = d_by_dt(Vphia)
        Aphib: Aphib = d_by_dt(Vphib)
        }
}
```

C. PLATE MASS

```
#MAST Code for a solid-plate mass
#with respect to the x, y and phi directions
#Its split into displacement and position
#
#Jan E. Vandemeer, Michael Kranz
#Carnegie Mellon University
#5/1/98
#
#***********************************************

element template plate_mass y_l y_r y_b y_t y_ne y_nw y_se y_sw \
                            Yl Yr Yb Yt Yne Ynw Yse Ysw \
                            x_l x_r x_b x_t x_ne x_nw x_se x_sw \
                            Xl Xr Xb Xt Xne Xnw Xse Xsw \
                            phi_l phi_r phi_b phi_t \
                            phi_ne phi_nw phi_se phi_sw \
                            PHIl PHIr PHIb PHIt PHIne PHInw PHIse PHIsw \
                            v_l v_r v_b v_t v_ne v_nw v_se v_sw = l, w

translational_pos      y_l, y_r, y_b, y_t,     #Y-displacements
                       y_ne, y_nw, y_se, y_sw

translational_pos      Yl, Yr, Yb, Yt, Yne, Ynw, Yse, Ysw  #Y-positions

translational_pos      x_l, x_r, x_b, x_t,     #X-displacements
                       x_ne, x_nw, x_se, x_sw

translational_pos      Xl, Xr, Xb, Xt, Xne, Xnw, Xse, Xsw  #X-positions

electrical             v_l, v_r, v_b, v_t,     #Voltages
                       v_ne, v_nw, v_se, v_sw

rotational_ang         phi_l, phi_r, phi_b, phi_t,     #Change in angle
                       phi_ne, phi_nw, phi_se, phi_sw

rotational_ang         PHIl, PHIr, PHIb, PHIt,         #Angles
                       PHIne, PHInw, PHIse, PHIsw

number w,  #width of mass plate
         l    #length of mass plate

{
#Internal nodes at the midpoint
translational_pos Ym, y_mid, Xm, x_mid
electricalv_mid
rotational_angPHIm, phi_mid

#System variables to solve force in the y-direction
```

```
var frc_N f_l_mid, f_r_mid, f_b_mid, f_t_mid
var frc_N f_ne_mid, f_nw_mid, f_se_mid, f_sw_mid

var frc_N F_nw, Fx_nw, F_t, Fx_t, F_ne, Fx_ne,
          F_r, Fx_r, F_se, Fx_se, F_b, Fx_b,
          F_sw, Fx_sw, F_l, Fx_l

#System variables to solve force in the x-direction
var frc_N f_x_l_mid, f_x_r_mid, f_x_b_mid, f_x_t_mid

var frc_N f_x_ne_mid, f_x_nw_mid, f_x_se_mid, f_x_sw_mid

#System variables to find the torque about the mid-point
var tq_Nm t_l_mid, t_r_mid, t_b_mid, t_t_mid
var tq_Nm t_ne_mid, t_nw_mid, t_se_mid, t_sw_mid

var tq_Nm T_nw, T_sw, T_t, T_ne, T_se, T_r, T_b, T_l

#System variables for global velocity and acceleration
var vel_mps   Vx, Vy

var accl_mps2 Ax, Ay

<tech_file.sin

number  ms,#mass of plate
          I,     #moment of inertia
          r,     #Resistance of plate
          arc,  #Angle between midpoint, and corner
          sinarc,     #Sine of that angle
          cosarc,     #Cosine of that angle
          Lngth,     #hypotenuse between midpoint and corner
          By     #Damping coefficient

valpos_mpos_mid     #value of y at the center of mass
valpos_mpos_x_mid     #value of x at the center of mass
valang_rad ang_mid    #value of phi at the center of mass

valang_rad ANG     #Overall angle of the center mass

varw_radpsangv_mid     #angular velocity of plate
vardw_radps2  anga_mid     #angular acceleration of plate
valang_radphi_off_1     #offset angle between midpoint and corners
valang_rad phi_off_2 #tan-1(l/w) - phi_mid
valpos_msin_off_1,     #values to help simplify the equations,
          sin_off_2,     #trigonemtric equations
          cos_off_1,
          cos_off_2,
          l_cos_mid,
          w_cos_mid,
```

```
        l_sin_mid,
        w_sin_mid

valpos_mLcos, Lsin, Wcos, Wsin,     #Trigonemetric vals for position
        PHIoff1, PHIoff2,
        Sinoff1, Sinoff2,
        Cosoff1, Cosoff2

valpos_mxg, yg     #Global displacements

parameters{
ms = poly1_t*w*l*poly_den
I = ms*(l**2 + w**2)/12
r = poly1_rho*(l/w)
Lngth = sqrt((w/2)**2 + (l/2)**2)
arc = atan(l/w)
sinarc = sin(arc)*Lngth
cosarc = cos(arc)*Lngth
By = visc_air*(l*w)/(air_gap)
}

values{
pos_mid = pos_m(y_mid)
pos_x_mid = pos_m(x_mid)
ang_mid = ang_rad(phi_mid)
ANG = ang_rad(PHIm)

phi_off_1 = arc + ang_mid
phi_off_2 = arc - ang_mid
sin_off_1 = Lngth*sin(phi_off_1)
sin_off_2 = Lngth*sin(phi_off_2)
cos_off_1 = Lngth*cos(phi_off_1)
cos_off_2 = Lngth*cos(phi_off_2)
l_cos_mid = (l/2)*cos(ang_mid)
w_cos_mid = (w/2)*cos(ang_mid)
l_sin_mid = (l/2)*sin(ang_mid)
w_sin_mid = (w/2)*sin(ang_mid)

PHIoff1 = arc + ANG
PHIoff2 = arc - ANG

Lcos = (l/2)*cos(ANG)
Lsin = (l/2)*sin(ANG)
Wcos = (w/2)*cos(ANG)
Wsin = (w/2)*sin(ANG)
Sinoff1 = Lngth*sin(PHIoff1)
Sinoff2 = Lngth*sin(PHIoff2)
Cosoff1 = Lngth*cos(PHIoff1)
Cosoff2 = Lngth*cos(PHIoff2)
```

```
#Place displacements in global frame
xg = pos_x_mid*cos(ANG) - pos_mid*sin(ANG)
yg = pos_x_mid*sin(ANG) + pos_mid*cos(ANG)
}

equations {

#Inertial forces, translational
frc_N(x_mid) -= ms*(cos(ANG)*(Ax)+sin(ANG)*(Ay))+ d_by_dt(By*pos_x_mid)
frc_N(y_mid) -= ms*(-sin(ANG)*(Ax)+cos(ANG)*(Ay))+ d_by_dt(By*pos_mid)

#Rotational inertial forces
tq_Nm(phi_mid) -= I*anga_mid + \
w_cos_mid*(f_r_mid - f_l_mid) + l_sin_mid*(f_b_mid - f_t_mid) + \
w_sin_mid*(f_x_l_mid - f_x_r_mid) + l_cos_mid*(f_x_b_mid - f_x_t_mid)+\
Lngth*(sin(phi_off_1)*(f_x_sw_mid - f_x_ne_mid) + \
cos(phi_off_1)*(f_ne_mid - f_sw_mid) + \
sin(phi_off_2)*(f_x_se_mid - f_x_nw_mid) + \
cos(phi_off_2)*(f_se_mid - f_nw_mid))

#Solving for the through variables at each node
frc_N(y_b->y_mid) += f_b_mid
frc_N(y_l->y_mid) += f_l_mid
frc_N(y_r->y_mid) += f_r_mid
frc_N(y_t->y_mid) += f_t_mid
frc_N(y_ne->y_mid) += f_ne_mid
frc_N(y_nw->y_mid) += f_nw_mid
frc_N(y_se->y_mid) += f_se_mid
frc_N(y_sw->y_mid) += f_sw_mid

frc_N(x_b->x_mid) += f_x_b_mid
frc_N(x_l->x_mid) += f_x_l_mid
frc_N(x_r->x_mid) += f_x_r_mid
frc_N(x_t->x_mid) += f_x_t_mid
frc_N(x_ne->x_mid) += f_x_ne_mid
frc_N(x_nw->x_mid) += f_x_nw_mid
frc_N(x_se->x_mid) += f_x_se_mid
frc_N(x_sw->x_mid) += f_x_sw_mid

tq_Nm(phi_b->phi_mid) += t_b_mid
tq_Nm(phi_l->phi_mid) += t_l_mid
tq_Nm(phi_r->phi_mid) += t_r_mid
tq_Nm(phi_t->phi_mid) += t_t_mid
tq_Nm(phi_ne->phi_mid) += t_ne_mid
tq_Nm(phi_nw->phi_mid) += t_nw_mid
tq_Nm(phi_se->phi_mid) += t_se_mid
tq_Nm(phi_sw->phi_mid) += t_sw_mid

frc_N(Ynw->Ym) += F_nw
frc_N(Yt->Ym) += F_t
```

```
frc_N(Yne->Ym) += F_ne
frc_N(Yr->Ym) += F_r
frc_N(Yse->Ym) += F_se
frc_N(Yb->Ym) += F_b
frc_N(Ysw->Ym) += F_sw
frc_N(Yl->Ym) += F_l

frc_N(Xnw->Xm) += Fx_nw
frc_N(Xt->Xm) += Fx_t
frc_N(Xne->Xm) += Fx_ne
frc_N(Xr->Xm) += Fx_r
frc_N(Xse->Xm) += Fx_se
frc_N(Xb->Xm) += Fx_b
frc_N(Xsw->Xm) += Fx_sw
frc_N(Xl->Xm) += Fx_l

tq_Nm(PHIb->PHIm) += T_b
tq_Nm(PHIr->PHIm) += T_r
tq_Nm(PHIt->PHIm) += T_t
tq_Nm(PHIne->PHIm) += T_ne
tq_Nm(PHInw->PHIm) += T_nw
tq_Nm(PHIse->PHIm) += T_se
tq_Nm(PHIsw->PHIm) += T_sw
tq_Nm(PHIl->PHIm) += T_l

#Current through the mass
i(v_l->v_mid) += v(v_l,v_mid)/r
i(v_r->v_mid) += v(v_r,v_mid)/r
i(v_b->v_mid) += v(v_b,v_mid)/r
i(v_t->v_mid) += v(v_t,v_mid)/r
i(v_ne->v_mid) += v(v_ne,v_mid)/r
i(v_nw->v_mid) += v(v_nw,v_mid)/r
i(v_se->v_mid) += v(v_se,v_mid)/r
i(v_sw->v_mid) += v(v_sw,v_mid)/r

#Solving for the system variables

T_nw:   ang_rad(PHInw,PHIm) = 0
T_sw:   ang_rad(PHIsw,PHIm) = 0
T_t:    ang_rad(PHIt,PHIm) = PI/2
T_b:    ang_rad(PHIb,PHIm) = PI/2
T_ne:   ang_rad(PHIne,PHIm) = 0
T_se:   ang_rad(PHIse,PHIm) = 0
T_r:    ang_rad(PHIr,PHIm) = 0
T_l:    ang_rad(PHIl,PHIm) = 0

F_nw:   pos_m(Ynw,Ym) = Sinoff2
Fx_nw: pos_m(Xnw,Xm) = -Cosoff2
F_t:    pos_m(Yt,Ym) = Lcos
Fx_t:   pos_m(Xt,Xm) = -Lsin
```

```
F_ne:    pos_m(Yne,Ym) = Sinoff1
Fx_ne:  pos_m(Xne,Xm) = Cosoff1
F_r:     pos_m(Yr,Ym) = Wsin
Fx_r:    pos_m(Xr,Xm) = Wcos
F_se:    pos_m(Yse,Ym) = -Sinoff2
Fx_se:  pos_m(Xse,Xm) = Cosoff2
F_b:     pos_m(Yb,Ym) = -Lcos
Fx_b:    pos_m(Xb,Xm) = Lsin
F_sw:   pos_m(Ysw,Ym) = -Sinoff1
Fx_sw: pos_m(Xsw,Xm) = -Cosoff1
F_l:     pos_m(Yl,Ym) = -Wsin
Fx_l:    pos_m(Xl,Xm) = -Wcos

f_b_mid:pos_m(y_b,y_mid) = -l_cos_mid + l/2
f_l_mid:pos_m(y_l,y_mid) = -w_sin_mid
f_r_mid:pos_m(y_r,y_mid) = w_sin_mid
f_t_mid:pos_m(y_t,y_mid) = l_cos_mid - l/2
f_ne_mid:pos_m(y_ne,y_mid) = sin_off_1 - sinarc #Lngth*sin(arc)
f_nw_mid:pos_m(y_nw,y_mid) = sin_off_2 - sinarc #Lngth*sin(arc)
f_se_mid:pos_m(y_se,y_mid) = sinarc - sin_off_2 #Lngth*sin(arc)
f_sw_mid:pos_m(y_sw,y_mid) = sinarc - sin_off_1 #Lngth*sin(arc)

f_x_b_mid :pos_m(x_b,x_mid) = l_sin_mid
f_x_l_mid :pos_m(x_l,x_mid) = -w_cos_mid + w/2
f_x_r_mid :pos_m(x_r,x_mid) = w_cos_mid - w/2
f_x_t_mid :pos_m(x_t,x_mid) = -l_sin_mid
f_x_ne_mid :pos_m(x_ne,x_mid) = cos_off_1 - cosarc #Lngth*cos(arc)
f_x_nw_mid :pos_m(x_nw,x_mid) = cosarc - cos_off_2 #Lngth*cos(arc)
f_x_se_mid :pos_m(x_se,x_mid) = cos_off_2 - cosarc #Lngth*cos(arc)
f_x_sw_mid :pos_m(x_sw,x_mid) = cosarc - cos_off_1 #Lngth*cos(arc)

t_l_mid : ang_rad(phi_l,phi_mid) = 0
t_r_mid :ang_rad(phi_r,phi_mid) = 0
t_b_mid :ang_rad(phi_b,phi_mid) = 0
t_t_mid :ang_rad(phi_t,phi_mid) = 0
t_ne_mid :ang_rad(phi_ne,phi_mid) = 0
t_nw_mid :ang_rad(phi_nw,phi_mid) = 0
t_se_mid :ang_rad(phi_se,phi_mid) = 0
t_sw_mid :ang_rad(phi_sw,phi_mid) = 0

angv_mid :angv_mid = d_by_dt(ang_mid+ANG)
anga_mid :anga_mid = d_by_dt(angv_mid)

#Velocities and accelerations
Vx:      Vx = d_by_dt(pos_m(Xm)+xg)
Vy:      Vy = d_by_dt(pos_m(Ym)+yg)
Ax:      Ax = d_by_dt(Vx)
Ay:      Ay = d_by_dt(Vy)
         }
}
```

D. Y-AXIS COMB DRIVE

```
#       MAST Code for a Comb-Drive Actuator
#       with respect to the x, y and phi directions
#
#       Jan E. Vandemeer and Michael Kranz
#       Carnegie Mellon University
#       4/17/97
#
#*********************************************

element template combdrive_y x_r Xr x_s Xs y_r Yr y_s Ys \
                              phi_r PHIr phi_s PHIs v_r v_s =  \
                              rotor_fingers, overlap, gap, finger_width, finger_length

translational_pos       x_r, y_r, #displacement of the rotors "r"
                        x_s, y_s     #and stators "s"

translational_pos       Xr, Yr, #Position of the rotors "r"
                        Xs, Ys     #and stators "s"

rotational_ang          phi_r, PHIr, phi_s, PHIs #Angle and angle displacement

electricalv_r, v_s     #voltage

number          finger_width,#width of fingers
                finger_length, #length of fingers
                overlap,  #initial overlap of fingers
                gap,   #gap between fingers
                rotor_fingers   #number of rotor fingers

{
<tech_file.sin

number ntv_ox_t = 20n#Native oxide thickness

number  ms,             #mass of rotor fingers
        I,              #moment of inertia
        B,              #Damping factor
        Area            #Cross-sectional area of comb-fingers

val c     cap           #capacitance between comb-fingers
val v     vlt           #voltage across fingers.
val v     vltg          #voltage squared, times area divided by eps0.
valq      q             #Charge in coulombs

valpos_mpx, py          #Position across comb-drive
valpos_mdx, dy          #Displacement across comb-drive

valpos_movrlp           #overlap just incase combs crash in y.
```

```
valnu    sign          #If dx is + or -

valang_radphi        #angle across comb-drive
valang_rad dphi      #Angle displacement in comb-drive

val nu   const,        #on or off-alternates force fcn
         const2,       #the spring constant of poly-Si (after crashing)
         intcpt1

varvel_mps  velxr, velyr       #Velocity of rotor
varvel_mps  velxs, velys       #Velocity of stator

varnu    dvlt                  #Derivative of the voltage

varfrc_NFx, Fy       #Forces through comb-drive (position nodes)
valfrc_NFxd, Fyd     #Forces through comb-drive
vartq_NmTphi         #Torque accross comb-drive

valpos_mskin_depth     #Skin depth for damping

#whenever you have the two beams crash!! Make sure the simulator is accurate
when(threshold((gap-abs(pos_m(x_r,x_s))),ntv_ox_t)){
schedule_next_time(time)
}

parameters{
Area = rotor_fingers*poly1_t
ms = rotor_fingers*finger_length*finger_width*poly1_t*poly_den
B = visc_air*((finger_length+4u)*(finger_width+4u)*\
   rotor_fingers/air_gap + (finger_length+4u)*(poly1_t)/gap)
}

values{
dphi = ang_rad(phi_r,phi_s)
dx = pos_m(x_r,x_s)
dy = pos_m(y_r,y_s)

px = pos_m(Xr,Xs)
py = pos_m(Yr,Ys)
phi = ang_rad(PHIr,PHIs)

if(dc_domain){
         skin_depth = sqrt((visc_air/PI)*0.001*air_den)
}
else if(time_domain){
#Approximation for time domain
         skin_depth = sqrt((visc_air/PI)*30k*air_den)
}
else{
         skin_depth = sqrt((visc_air/PI)*(1+freq_mag)*air_den)
```

```
}

#Check for crashing in y-direction

if(dy > overlap){
        ovrlp = -overlap
}
else if(dy < (overlap - finger_length)){
        ovrlp = -overlap + finger_length
}
else{
        ovrlp = -dy
}

#check for sign value
if(dx < 0){
        sign = -1
}
else{
        sign = 1
}

if((gap-abs(pos_m(x_r,x_s))) < ntv_ox_t){
        const = 0
        const2 = E*poly1_t*overlap/finger_width
        intcpt1 = sign*(gap-ntv_ox_t)
}
else{
        const = 1
        const2 = 0
        intcpt1 = 0
}

vlt = v(v_r,v_s)
vltg = (vlt**2)*Area*eps0/2

cap= Area*eps0*(overlap+ovrlp)*((1/(gap-(const*dx + intcpt1))) + \
    (1/(gap+(const*dx + intcpt1))))

Fyd =   1.12*vltg*((1/(gap-(const*dx + intcpt1/ntv_ox_t)))+ \
        (1/(gap+(const*dx + intcpt1/ntv_ox_t))))

Fxd = (vltg*(overlap+ovrlp))*((1/((gap - (const*dx + intcpt1))**2)) - \
     (1/((gap + (const*dx + intcpt1))**2))) - const2*(dx-intcpt1)

q = cap*vlt

}

equations {
```

```
dvlt:    dvlt = d_by_dt(vlt)
velxr:   velxr = d_by_dt(pos_m(x_r))
velyr:   velyr = d_by_dt(pos_m(y_r))
velxs:   velxs = d_by_dt(pos_m(x_s))
velys:   velys = d_by_dt(pos_m(y_s))

i(v_r->v_s) += d_by_dt(q)

frc_N(y_r->y_s) += Fyd
frc_N(x_r->x_s) += Fxd

frc_N(y_r) -= d_by_dt(ms*velyr) + (B*velyr*(1 + air_gap/skin_depth))
frc_N(y_s) -= d_by_dt(ms*velys) + (B*velys*(1 + air_gap/skin_depth))
frc_N(x_r) -= d_by_dt(ms*velxr) + (B*velxr*(1 + air_gap/skin_depth))
frc_N(x_s) -= d_by_dt(ms*velxs) + (B*velxs*(1 + air_gap/skin_depth))

#Rotational stuff (none thus far!)
tq_Nm(phi_r->phi_s) += 1meg*dphi

frc_N(Xr->Xs) += Fx
frc_N(Yr->Ys) += Fy
tq_Nm(PHIr->PHIs) += Tphi

Fx:      px = (-2*finger_length + overlap)*cos(ang_rad(PHIr))
Fy:      py = (-2*finger_length + overlap)*sin(ang_rad(PHIr))
Tphi:    phi = 0
}
}
```

E. X-AXIS COMB DRIVE

```
#       MAST Code for a Comb-Drive Actuator
#       with respect to the x, y and phi directions
#
#       Jan E. Vandemeer and Michael Kranz
#       Carnegie Mellon University
#       4/17/97
#
#*********************************************

element template combdrive_x x_r Xr x_s Xs y_r Yr y_s Ys \
                                phi_r PHIr phi_s PHIs v_r v_s =  \
                                rotor_fingers, overlap, gap, finger_width, finger_length

translational_pos       x_r, y_r, #displacement of the rotors “r”
                        x_s, y_s     #and stators “s”

translational_pos       Xr, Yr,  #Position of the rotors “r”
                        Xs, Ys     #and stators “s”

rotational_ang          phi_r, PHIr, phi_s, PHIs #Angle and angle displacement

electricalv_r, v_s     #voltage

number          finger_width,#width of fingers
                finger_length, #length of fingers
                overlap,          #initial overlap of fingers
                gap,              #gap between fingers
                rotor_fingers    #number of rotor fingers

{

<tech_file.sin

number ntv_ox_t = 20n

number  ms,             #mass of rotor fingers
          I,              #moment of inertia
          B,              #Damping factor
          Area            #Cross-sectional area of comb-fingers

val c     cap     #capacitance between comb-fingers
val v     vlt     #voltage across fingers.
val v     vltg    #voltage squared, times area divided by eps0.
valq      q       #Charge in coulombs

valpos_mpx, py     #Position across comb-drive
valpos_mdx, dy     #Displacement across comb-drive
```

```
valpos_movrlp     #overlap just incase combs crash in y.
valnu    sign       #If dx is + or -

valang_radphi     #angle across comb-drive
valang_rad dphi   #Angle displacement in comb-drive

val nu   const,      #on or off-alternates force fcn
         const2,     #the spring constant of poly-Si (after crashing)
         intcpt1     #constant offset for after crashing

varvel_mps  velxr, velyr     #Velocity of rotor
varvel_mps  velxs, velys     #Velocity of stator

varnu    dvlt                #Derivative of the voltage

varfrc_NFx, Fy         #Forces through comb-drive (position nodes)
valfrc_NFxd, Fyd       #Forces causing displacement in comb-drive
vartq_NmTphi           #Torque accross comb-drive

valpos_mskin_depth    #Skin depth for damping

#whenever you have the two beams crash!!  Adjust the force function
when(threshold((gap-abs(pos_m(y_r,y_s))),ntv_ox_t)){
schedule_next_time(time)
}
parameters{
Area = rotor_fingers*poly1_t
ms = rotor_fingers*finger_length*finger_width*poly1_t*poly_den
B = visc_air*((finger_length+4u)*(finger_width+4u)*\
   rotor_fingers/air_gap+ (finger_length+4u)*(poly1_t)/gap)
}

values{
dphi = ang_rad(phi_r,phi_s)
dx = pos_m(x_r,x_s)
dy = pos_m(y_r,y_s)

px = pos_m(Xr,Xs)
py = pos_m(Yr,Ys)
phi = ang_rad(PHIr,PHIs)

if(dc_domain){
         skin_depth = sqrt((visc_air/PI)*0.001*air_den)
}
else if(time_domain){
         #Approximation for time domain
         skin_depth = sqrt((visc_air/PI)*30k*air_den)
}
else{
         skin_depth = sqrt((visc_air/PI)*(1+freq_mag)*air_den)
```

```
}

#Check for crashing in y-direction

if(dx > overlap){
        ovrlp = -overlap
}
else if(dx < (overlap - finger_length)){
        ovrlp = -overlap + finger_length
}
else{
        ovrlp = -dx
}

#check for sign value
if(dy < 0){
        sign = -1
}
else{
        sign = 1
}

if((gap-abs(pos_m(y_r,y_s))) < ntv_ox_t){
        const = 0
        const2 = E*poly1_t*overlap/finger_width
        intcpt1 = sign*(gap-ntv_ox_t)
}
else{
        const = 1
        const2 = 0
        intcpt1 = 0
}

vlt = v(v_r,v_s)
vltg = (vlt**2)*Area*eps0/2

cap= Area*eps0*(overlap+ovrlp)*\
    ((1/(gap-(const*dy + intcpt1)))+(1/(gap+(const*dy + intcpt1))))

Fxd = 1.12*vltg*((1/(gap-(const*dy + intcpt1/ntv_ox_t)))+ \
(1/(gap+(const*dy + intcpt1/ntv_ox_t))))

Fyd =   vltg*(overlap+ovrlp)*((1/((gap - (const*dy + intcpt1))**2))-\
        (1/((gap + (const*dy + intcpt1))**2))) - const2*(dy-intcpt1)

q = cap*vlt
}

equations {
dvlt:   dvlt = d_by_dt(vlt)
```

```
velxr:    velxr = d_by_dt(pos_m(x_r))
velyr:    velyr = d_by_dt(pos_m(y_r))
velxs:    velxs = d_by_dt(pos_m(x_s))
velys:    velys = d_by_dt(pos_m(y_s))

i(v_r->v_s) +=d_by_dt(q)

frc_N(x_r->x_s) += Fxd
frc_N(y_r->y_s) += Fyd
frc_N(y_r) -= d_by_dt(ms*velyr) + (B*velyr*(1 + air_gap/skin_depth))
frc_N(y_s) -= d_by_dt(ms*velys) + (B*velys*(1 + air_gap/skin_depth))
frc_N(x_r) -= d_by_dt(ms*velxr) + (B*velxr*(1 + air_gap/skin_depth))
frc_N(x_s) -= d_by_dt(ms*velxs) + (B*velxs*(1 + air_gap/skin_depth))

tq_Nm(phi_r->phi_s) += 10meg*dphi

#Rotational stuff (none thus far!) Assumed stiff!!

frc_N(Xr->Xs) += Fx
frc_N(Yr->Ys) += Fy
tq_Nm(PHIr->PHIs) += Tphi

Fy:       py = (-2*finger_length + overlap)*cos(ang_rad(PHIr))
Fx:       px = (-2*finger_length + overlap)*sin(ang_rad(PHIr))
Tphi:     phi = 0
}
}
```

F. ELECTROSTATIC GAP

```
#       MAST Code for an electrostatic gap in the x-direction
#
#       Jan E. Vandemeer and Michael Kranz
#MAST Model for an electrostatic gap!
#       Carnegie Mellon University
#       5/2/98
#
#*********************************************

element template es_gap Xm_t Xm_b Xp_t Xp_b Ym_t Ym_b Yp_t Yp_b \
                        x_a_t x_b_t x_a_b x_b_b y_a_t y_a_b y_b_t y_b_b \
                        PHIm_t PHIm_b PHIp_t PHIp_b \
                        phi_a_t phi_a_b phi_b_t phi_b_b \
                        v_a_t v_a_b v_b_t v_b_b \
                        = finger_w_t, finger_w_b, gap, overlap

translational_pos       x_a_t,      #x-displacement and force at port “x_a_t”
                        y_a_t,      #y-displacement and force at port “y_a_t”
                        x_a_b,      #x-displacement and force at port “x_a_b”
                        y_a_b,      #y-displacement and force at port “y_a_b”
                        x_b_t,      #x-displacement and force at port “x_b_t”
                        y_b_t,      #y-displacement and force at port “y_b_t”
                        x_b_b,      #x-displacement and force at port “x_b_b”
                        y_b_b,      #y-displacement and force at port “y_b_b”
                        Xm_t,       #X-position at “Xm_t”
                        Xm_b,       #X-position at “Xm_b”
                        Ym_t,       #Y-position at “Ym_t”
                        Ym_b,       #Y-position at “Ym_b”
                        Xp_t,       #X-position at “Xp_t”
                        Xp_b,       #X-position at “Xp_b”
                        Yp_t,       #Y-position at “Yp_t”
                        Yp_b        #Y-position at “Yp_b”

rotational_ang          phi_a_t,  #Angle displacement about z angle and torque
                        phi_a_b,  #Angle displacement about z angle and torque
                        phi_b_t,  #Angle displacement about z angle and torque
                        phi_b_b,  #Angle displacement about z angle and torque
                        PHIm_t,  #Rotation about z angle
                        PHIm_b,
                        PHIp_t,  #Rotation about z angle
                        PHIp_b

electricalv_a_t, v_a_b, v_b_t, v_b_b     #input & output voltage

number          finger_w_t = 2u#width of the top finger
number          finger_w_b = 2u#width of the bottom finger
number          gap = 2u            #Gap between the two beams
number          overlap = 10u       #X-overlap between the beams
```

```
{

<tech_file.sin

varfrc_Npyp, pxm     #Position source for y and x

number  slope = E*overlap*poly1_t/(finger_w_t/2 + finger_w_b/2),
ntv_ox_t = 20n     #Native oxide thickness around the fingers

valfrc_Nfyd1, fyd2   #Electrostatic force in y
valfrc_Nfxd1, fxd2   #Electrostatic force in x

valpos_movrlp         #overlap of beams

valpos_mppt, ppb, pmt, pmb #Positions at +/- on top and bottom

valpos_mdy      #The displacements in y
number ydc       #The DC value of y position.

valang_radphi, dphi
valv      vlt

valarea_m2Area     #Cross sectional Area of fingers
val c     cap           #capacitance between comb-fingers
valv      vltg
valnu     caparea
val nu    test

val nu    const,      #state, digital on or off-alternates force function
          const2     #the spring constant (after crashing)

#whenever you have the two beams crash, shorten time steps to help simulator
when(threshold((gap - dy),(ntv_ox_t))){ #,before,after)){
schedule_next_time(time)
}

#Parameters section
parameters{
ydc = gap + finger_w_t/2 + finger_w_b/2
}

#values section
values{
ppt = pos_m(Xp_t)
ppb = pos_m(Xp_b)
pmt = pos_m(Xm_t)
pmb = pos_m(Xm_b)

test = 0
```

```
if ((ppt > ppb) & (ppb > pmt) & (pmt > pmb)) {
        test = 1
        ovrlp = ppb - pmt
        dy = pos_m(y_b_b,y_a_t)
        vlt = v(v_b_b,v_a_t)
}
else{
        test = 2
        ovrlp = ppt - pmb
        dy = pos_m(y_a_b,y_b_t)
        vlt = v(v_a_b,v_b_t)
}

if((gap - dy) > ntv_ox_t){
        const = 1
        const2 = 0
}
else{
        const = 0
        const2 = slope
}

phi = ang_rad(PHIp_t,PHIm_b)

# Find coeff. in cap. eq.

Area = (ovrlp)*poly1_t
vltg = (vlt)**2
caparea = Area*eps0
cap = caparea/((1-const)*ntv_ox_t + const*(gap - dy))

if (pmt < pmb) {
dphi = ang_rad(phi_b_t,phi_a_b)

fxd1 = (vltg*eps0*poly1_t)/(2*(const*(gap - dy) + (1 - const)*ntv_ox_t)) - \
       const2*pos_m(x_b_t,x_a_b)

fxd2 = 0

fyd1 = (vltg*(caparea) /(2*(const*(gap - dy) +(1-const)*ntv_ox_t)**2))  - const2*(-(gap - dy) + \
      ntv_ox_t)

fyd2 = 0
}
else{
dphi = ang_rad(phi_a_t,phi_b_b)

fxd1 = 0
```

```
fxd2 = -(vltg*eps0*poly1_t)/(2*(const*(gap - dy) + (1-const)*ntv_ox_t)) - \
        const2*pos_m(x_a_t,x_b_b)

fyd2 = (vltg*(caparea) /(2*(const*(gap - dy) +(1-const)*ntv_ox_t)**2)) - const2*(-(gap - dy) + \
        ntv_ox_t)

fyd1 = 0
         }
}

equations {
          i(v_a_t->v_a_b) += d_by_dt(v(v_a_t,v_a_b)*cap)
          i(v_b_t->v_b_b) += d_by_dt(v(v_b_t,v_b_b)*cap)

          frc_N(x_b_t->x_a_b) -= fxd1
          frc_N(x_a_t->x_b_b) -= fxd2

          frc_N(y_b_t->y_a_b) -=  fyd1
          frc_N(y_a_t->y_b_b) -=  fyd2

          tq_Nm(phi_b_t->phi_b_b) -= 0
          tq_Nm(phi_a_t->phi_a_b) -= 0

#Positions

tq_Nm(PHIp_b->PHIm_t) += 1meg*ang_rad(PHIp_b,PHIm_t)
tq_Nm(PHIp_t->PHIm_b) += 1meg*ang_rad(PHIp_t,PHIm_b)

frc_N(Yp_t,Yp_b) += pyp
frc_N(Yp_t,Ym_t) += 1meg*pos_m(Yp_t,Ym_t)
frc_N(Yp_b,Ym_b) += 1meg*pos_m(Yp_b,Ym_b)

frc_N(Xm_b,Xm_t) += pxm
frc_N(Xp_t,Xm_t) += 1meg*pos_m(Xp_t,Xm_t)
frc_N(Xp_b,Xm_b) += 1meg*pos_m(Xp_b,Xm_b)
frc_N(Xp_t,Xp_b) += 1meg*pos_m(Xp_t,Xp_b)

pyp:     pos_m(Yp_t,Yp_b) = gap+(finger_w_t+finger_w_b)/2
pxm:     pos_m(Xm_b,Xm_t) = pos_m(Xp_t) - overlap
}
}
```

G. JOINT

```
#MAST Code for a rotational joint
#with respect to the x, y and phi directions
#
#Jan E. Vandemeer, M. Kranz
#Carnegie Mellon University,
#Copyright 1998, All rights reserved.
#5/1/98
#
#***********************************************

element template jointx_w x_e x_n x_s Xw Xe Xn Xs \
                       y_w y_e y_n y_s Yw Ye Yn Ys \
                       phi_w phi_e phi_n phi_s PHIw PHIe PHIn PHIs \
                       vw ve vn vs = ang

translational_posy_w, y_s, y_n, y_e,  #Y displacement
                  x_w, x_s, x_n, x_e      #X displacement

translational_posYw, Ys, Yn, Ye,#Y position
                   Xw, Xs, Xn, Xe  #X position

rotational_angphi_w, phi_s, phi_n, phi_e,     #Angular displacement
                PHIw, PHIs, PHIn, PHIe        #Global Angle

electricalvw, vn, ve, vs

number ang#Angle between the east and south ports (in degrees)

{

rotational_angPHIm

<tech_file.sin

number  offset

#system variables for position (and displacement) sources
var frc_N xSbend, ySbend, xNbend, yNbend, xEbend, yEbend

#system variables for angle sources
var tq_Nm phiSbend, phiNbend, phiEbend
var tq_Nm PHIsb, PHInb

parameters{
  offset = ang*PI/180#From degrees to radians
}

equations {
```

```
#Voltage sources (short circuits)
      i(vn->vw) += 4.53*v(vs,ve) #/poly1_rho
      i(vs->vw) += 4.53*v(vn,ve) #/poly1_rho
      i(ve->vs) += 4.53*v(vw,vn) #/poly1_rho
      i(ve->vn) += 4.53*v(vw,vs) #/poly1_rho

#Damping from joint
      frc_N(Xw) -= d_by_dt(1n*pos_m(Xw))
      frc_N(Yw) -= d_by_dt(1n*pos_m(Yw))
      tq_Nm(PHIm) -= d_by_dt(1n*ang_rad(Phim))

#Actual solutions for the forces at nodes
frc_N(x_w->x_s) += xSbend
frc_N(y_w->y_s) += ySbend
frc_N(Xw->Xs) += 1meg*pos_m(Xw,Xs)
frc_N(Yw->Ys) += 1meg*pos_m(Yw,Ys)
frc_N(x_w->x_n) += xNbend
frc_N(y_w->y_n) += yNbend
frc_N(Xw->Xn) += 1meg*pos_m(Xw,Xn)
frc_N(Yw->Yn) += 1meg*pos_m(Yw,Yn)
frc_N(x_w->x_e) += xEbend
frc_N(y_w->y_e) += yEbend
frc_N(Xw->Xe) += 1meg*pos_m(Xw,Xe)
frc_N(Yw->Ye) += 1meg*pos_m(Yw,Ye)
tq_Nm(PHIw->PHIm) += 1meg*ang_rad(PHIw,PHIm)
tq_Nm(phi_w->phi_s) += phiSbend
tq_Nm(PHIm->PHIs) += PHIsb
tq_Nm(phi_w->phi_n) += phiNbend
tq_Nm(PHIs->PHIn) += PHInb
tq_Nm(phi_w->phi_e) += phiEbend
tq_Nm(PHIm->PHIe) += 1meg*ang_rad(PHIm,PHIe)

#Solution for system variables

xSbend:pos_m(x_w,x_s) = 0
ySbend:pos_m(y_w,y_s) = 0
xNbend:pos_m(x_w,x_n) = 0
yNbend:pos_m(y_w,y_n) = 0
xEbend:pos_m(x_w,x_e) = 0
yEbend:pos_m(y_w,y_e) = 0

phiSbend: ang_rad(phi_w,phi_s) = 0
PHIsb:    ang_rad(PHIm,PHIs) = -offset
phiNbend: ang_rad(phi_w,phi_n) = 0
PHInb:    ang_rad(PHIs,PHIn) = 0
phiEbend: ang_rad(phi_w,phi_e) = 0
}
}
```