UNIVERSITY OF CINCINNATI

DATE: May 8, 2000

I, **Dennis Gibson** **,**

hereby submit this as part of the requirements for the degree of:

MASTER OF SCIENCE

in:

Dept. of Electrical & Computer Engineering & Computer Science

It is entitled:

Integrating Behavioral Modeling & Simulation

for MEMS Components into CAD for VLSI

Approved by:

Carla Purdy

Harold W. Carter

Robert Ewing

Integrating Behavioral Modeling & Simulation for MEMS Components into CAD for VLSI

A thesis submitted to the

Division of Research and Advanced Studies
Of University of Cincinnati

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In the Department of Electrical & Computer Engineering & Computer Science
of the College of Engineering

2000

by

Dennis Gibson

B.S.E.E., University of Kentucky, 1988

Committee Chair: Dr. Carla Purdy
Committee Members: Dr. Hal Carter
Dr. Robert Ewing

Abstract

The goal of this project is to show that the use of behavioral modeling, in conjunction with current VLSI CAD tools, will aid in the rapid prototyping of Micro-Electro-Mechanical Systems (MEMS) and also in the area of design automation. A behavioral model of the cantilever beam (a basic MEMS device) is used as an instrument to demonstrate how such models may be utilized.

The first example demonstrates how to extend finite element analysis capabilities to an existing design tool. Second, an example including the cantilever beam model is used to demonstrate the usefulness of modeling in the area of simulation, which is of great importance for the rapid prototyping of systems. Finally an example is given to show how the cantilever beam model may be utilized to aid in the area of design automation.

Acknowledgements

First and foremost, I would like to thank my advisor and committee chair, Dr. Carla Purdy. This thesis would not have been possible without the constant support, guidance, patience and encouragement that she gave. Not only did I receive the benefits of her mentoring, but I also gained a great deal of knowledge from the courses that she taught.

I would also like to thank Dr. Robert Ewing and Dr. Hal Carter for the help that I have received from them and also for being members of my thesis defense committee.

In addition, I would like to thank the Wright Labs, the State of Ohio Dayton Area Graduate Studies Institute (DAGSI) for their generous support.

Finally, I would like to thank the University of Cincinnati Summer Fellows program for its contributions to my research.

Table of Contents

List of figures

List of tables

Chapter 1
Introduction

This thesis attempts to show the benefits that behavioral modeling and simulation provide in the area of Microelectromechanical Systems (MEMS) design. This work will give examples to demonstrate these benefits. This work has strictly limited itself to behavioral modeling and does not consider the structural or physical modeling domains.

Chapter 2 contains a discussion on the background and the significance of MEMS. A description of what MEMS devices are and what innovative applications are being devised for these devices is included.

Chapter 3 gives an overview of the work that has been performed and the goals of this research project.

Chapter 4 contains an overview of current MEMS manufacturing techniques. This chapter will explain why silicon is the predominant material used in MEMS fabrication. The chapter also characterizes the major techniques currently being used, such as the Excimer Laser, LIGA, bulk micromachining and surface micromachining. This project will mainly be involved with silicon micromachining.

Chapter 5 gives a listing and description of the tools and materials that were used in this project. This chapter is broken into two main components: tools used in the design of

MEMS devices, and tools used for simulation. The design tools and materials discussed include L-Edit, CIF format, and MUMPS. The simulation tools discussed include PSPICE, SEAMS (VHDL-AMS), ANSYS, MATHEMATICA, and MechanicsExplorers.

Chapter 6 details the model of a cantilever beam. The theory and behavior of a cantilever beam are discussed and relevant mathematical equations are stated. A simpler model is suggested to replace the standard cantilever beam model to aid the simplification of simulation.

Chapter 7 discusses extending finite element analysis capabilities to VHDL-AMS. The model of the cantilever beam is used to demonstrate.

Chapter 8 begins with our first application for the modeling of MEMS devices. In this chapter a technique is given for extracting the behavioral data from physical layout files to useful information for various simulators. A process for recognizing certain types of cantilever beams is given. In addition, the process for translation of the behavioral data to the various simulators used is shown.

Chapter 9 gives a second application for mathematical modeling of MEMS devices. This chapter discusses the benefits mathematical modeling may have in the area of design automation. A process for design automation is suggested. The chapter discusses how our model of the cantilever beam may also be extended to automate the design of similar MEMS devices. In our case, this includes the micro-mirror.

Chapter 10 states the conclusions derived from the research in this project. Future work needed is also discussed.

The appendices I through VI include the code and sample simulation results from this project.

Chapter 2
Background:
Significance of MEMS

Since 1965 the electronics industry has been keeping pace with Moore's Law that states that the number of transistors on a microchip will double every 18 months [INTEL]. We now see millions of microscopic circuit elements in the same area where just a few years ago we only saw a few thousand (see Figure 2.1). This technology

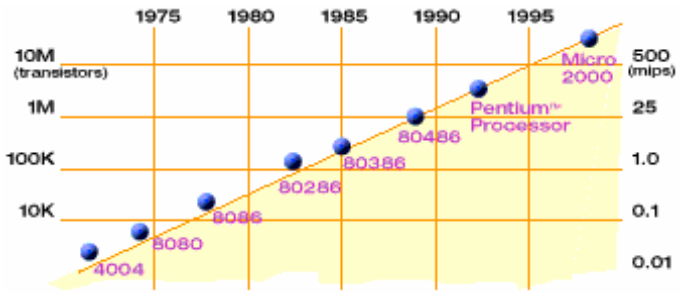


Figure 2.1 Transistor densities over last 30 years [INTEL]

is available for mass production and is commonplace in our everyday lives. Pocket telephones, beepers and pagers, caller ID's, and high-powered desktop computers are all examples of this incredible technology.

The processes for manufacturing microelectronics are very advanced. Now researchers are using the materials and processes of microelectronics to build a variety of microscopic mechanisms such as beams, pits, membranes, gears and motors [Peterson]. The size of these mechanisms is measured in microns.

Many believe that this structural engineering on silicon dies will have as great an impact on society as did the electronic miniaturization revolution [Gabriel]. Micromechanical devices will allow us to sense and control motion, light, sound, and other physical forces.

By putting together or coupling mechanical, fluidic and electrical systems, rapid advances will occur in many engineering areas. Microelectromechanical systems, or MEMS, is the title given to the combining of miniaturized mechanical and electrical components. MEMS are also referred to as mechatronics by some researchers. MEMS devices are made using similar manufacturing processes to that of building electronic components.

MEMS devices are in use today. For example, MEMS sensors are commonly used in air bags to determine if the force of an impact is great enough for the air bag to be deployed. This type of sensor employs beams manufactured with surface micromachining. "It changes the position of suspended parallel beams that make up an electrical capacitor, thus altering the amount of stored charge [Gabriel]." When an automobile rapidly decelerates (as is the case in the event of a crash), the beams' positions are affected, and thus the bag is deployed. Silicon Designs, Inc. has created an accelerometer based on this technology. This is shown in Figure 2.2.

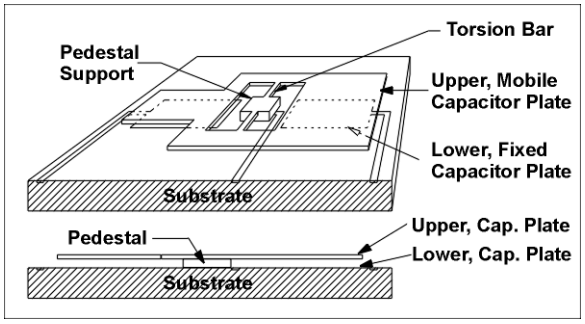


Figure 2.2 Example of accelerometer [Silicon Designs].

Another example of an innovative MEMS application comes from Texas Instruments [TI]. Texas Instruments has built an electronic display in which the picture elements, or pixels, that make up the image are controlled by microelectromechanical devices. "Each pixel consists of a 16-micron wide aluminum mirror that can reflect pulses of colored light onto a screen. The pixels are turned off or on when an electric field causes the mirrors to tilt 10 degrees to one side or the other. In one direction, a light beam is reflected onto the screen to illuminate the pixel. In the other, it scatters away from the screen, and the pixel remains dark [Gabriel]." What this design allows is the ability for a large screen to have a very high degree of brightness and resolution. It is common knowledge that other technologies have a problem with creating large screens with an adequate level of brightness and detail.

These two examples are not the only applications of MEMS devices, but are just a small sample. Several manufacturers have marketed MEMS based pressure transducers that have been manufactured commercially for over 10 years. As the manufacturing processes for MEMS become more sophisticated, such that thousands, or even millions, of devices are easily put on a single chip, more and more potential uses will be realized.

The future of MEMS can be seen from looking at the types of research that are being performed. In these research projects innovative and exciting technologies are being developed. One such exciting innovative technology is in the area of alternate data storage methods. Magnetic recording is the dominant data storage technology as of today. Compare capacities of typical hard drives from just three years ago to those of today and it is obvious that rapid advances are being made in data density. However, in the future the laws of physics will take over. It is expected that at some point, bit instability, due to super-paramagnetism, will limit the achievable data density. This limit is believed to be about 100 Gbits/in^2 [IBM, a]. IBM has been working on a MEMS based storage system. "Under more practical conditions of room temperature and atmospheric pressure, atomic-force microscopy (AFM) provides a means to write and read information at densities between 40-300 Gbits/in^2. This method relies on using a sharp tip mounted on a micromechanical cantilever. Data is written thermomechanically by heating the tip while it is in contact with a plastic disk substrate. The combination of heating and tip pressure causes a small indentation to be formed in the surface of the plastic. Readback of the data is achieved by monitoring the motion of the cantilever as the tip rides over the tiny indentations on the disk. As a comparison, an average CD holds 620 Mbytes in 23 in^2, whereas with the MEMS technology, IBM claims 50 Gbits in the same area" [IBM,a]. However, the 50Gbits is only the beginnings since with this technology, the limit would be 2,000-3,000 Gbits/in^2 [IBM, a]. In addition, since a large array of tips is used, high degrees of reliability and parallelism are possible. An example of how the tip works is portrayed in Figure 2.3.

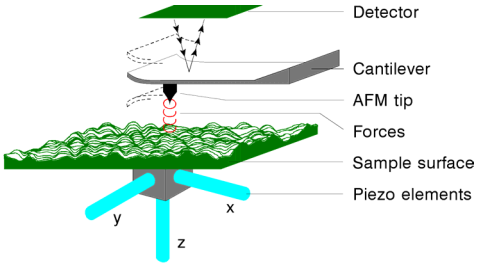


Figure 2.3 MEMS read/write head [IBM, b]

MEMS are also expected to make tremendous breakthroughs in other engineering disciplines, including aviation. A team of engineers at UCLA and the California Institute of Technology wants to show how MEMS may eventually influence aerodynamic design. These engineers have outlined how the large moving surfaces of the wings that control the turning, ascent and descent of the aircraft might be replaced. "It plans to line the surface of a wing with thousands of 150 micron-long plates that, in their resting position, remain flat on the wing surface. When an electrical voltage is applied, the plates rise from the surface at up to a 90-degree angle. Thus activated, they can control vortices of air that form across selected areas of the wing. Sensors can monitor the currents of air rushing over the wing and send a signal to adjust the position of the plates" [Gabriel]. It is expected that an aircraft with such technology would turn more quickly, stabilize against turbulence, and burn less fuel because of greater flying efficiency.

In conclusion, it is evident that the uses for MEMS devices are many and varied. As the technologies for manufacturing such devices matures and the complexity of devices increases, a need for simulation of such systems will be necessary, just as in VLSI

systems. Research will emphasize microelectromechanical systems design and analyses at the circuit level of abstraction, building on lowerlevels of abstraction involving device (finite element) and fabrication (process). “The objective is to align micro-electro-mechanical systems design with micro-electrical systems design, recognizing that there is already a significant technology base for very large scale integration (VLSI) micro-electronics that can serve as a useful precedent for advancing the emerging field of micro-electromechanical devices” [Dewey].

The work described in this paper will demonstrate that this technology base can serve to advance MEMS design.

Chapter 3
Overview of research project

In the last two decades, the concept of CAD for MEMS has been primarily associated with process modeling and finite element analysis (FEA). While a great deal of progress has been made in these areas, little has been done to benefit the system designer, who typically must start from scratch with each new project. One goal of researchers is to develop tools which allow MEMS designers to easily produce fully-simulated complex designs with reasonable confidence in the system's predicted performance. These tools would use a standard process (such as MCNC's MUMPS [Koester]) and would allow the designer to utilize a library of pre-simulated components to quickly build up fully simulated MEMS structures.

MEMS devices cannot be characterized as purely electrical or mechanical in nature but have properties from both domains. These domains are not independent. A change in the mechanical state of a MEMS device may affect the electrical state of the same device, and the converse is also true. There exist a number of well-developed electrical simulators and mechanical simulators for these separate domains. In an effort not to "re-invent the wheel", many researchers are incorporating these simulators into CAD packages to better enable the MEMS design process. But it may be the case that better mixed domain simulations would result from a closer integration of electrical and mechanical simulation techniques.

The work described here deals with the problems of modeling and simulation for MEMS devices. The eventual goal is to produce a top-down MEMS design system similar to today's powerful VLSI design systems. In this work we have chosen a fixed fabrication process (MUMPS) as a target and a fixed layout tool (L-Edit). Thus, our efforts are concentrated on reliable simulation of designs and eventual comparison with actual fabricated parts. Initially we focus on two main problems:

1) definition of the interface between a layout file and a variety of simulators which may be designed for MEMS or extendable to MEMS components, and
2) automatic translation of a device level description into its simulation description for these specific simulators.

By solving these problems in some specific cases, we expect to derive both insight into how to do MEMS simulation effectively and understanding of how to model MEMS components for simulation. Eventually we will develop a hierarchy of MEMS component models, for common devices such as beams, cantilever beams, and membranes, similar to the hierarchy of transistor models used in SPICE, as well as a library of more complex parts and parameterized descriptions. Using models from this hierarchy a designer will be able to choose the level of detail most appropriate for various phases of the design process.

In chapter 4 an overview of basic MEMS fabrication techniques is given. Chapter 5 gives a brief introduction into all the tools and materials used in this project. Chapter 6 discusses the model used in this project to represent the cantilever beam. Chapter 7 gives examples of how finite element analysis may be incorporated into VHDL-AMS models.
In chapter 8 we describe the various ways in which a simple MEMS device, the cantilever beam, can be built in the MCNC MUMPS process, and how appropriate data can be automatically extracted from the L-Edit [Tanner] design files to enable simulation in four different simulators: SPICE [Tuinenga], MATHEMATICA [Wolfram], ANSYS [ANSYS], and SEAMS [SEAMS] (which partially implements the VHDL-AMS Language Reference Manual (LRM) [Standards97]). For the SPICE simulations, we perform domain translations to allow the electrical simulators in this package to use equivalent equations to simulate mechanical behavior. VHDL-AMS allows other domain definitions. ANSYS was used as a control and was used mainly to compare these models with the standard FEA method to measure deflection in cantilever beams.

Once models are designed, these models may be used to automate the design of a particular component, the cantilever beam, based upon its behavioral description and given physical parameters. This gives the designer more freedom from the low-level details of the design just as in VLSI design. In chapter 9 this is demonstrated by a case study. The design of a micro-mirror is automated using a program written in MATHEMATICA.

Chapter 4
Overview of MEMS fabrication techniques

4.1 Silicon as a mechanical material

The basis for most micro-machines, silicon, in their usual role as an electrical material, has already been exploited due to an advanced microfabrication technology that has been developed over the last few decades. Silicon can also be exploited as a high-precision, high-strength, and high-reliability mechanical material. Silicon is especially useful in applications where miniature mechanical devices and components must be integrated or interfaced with electronics such as in the examples given in Chapter 2.

Silicon appears to be the most successful material employed in the pursuit of miniaturization. Four factors have played crucial roles in this phenomenal success [Petersen]:

1. Silicon is abundant, inexpensive, and can be produced and processed to meet high standards of purity and perfection;
2. Silicon processing is based on very thin deposited films which are highly amenable to miniaturization;
3. Definition and reproduction of the device shapes and patterns are performed using photographic techniques which are capable of high precision and amenable to miniaturization;
4. Silicon microelectronic circuits are batch-fabricated.

It is becoming clear that these same four factors that have been responsible for the rise of silicon microelectronics can be exploited in the design and manufacture of a large number of miniature mechanical devices and components.

Any consideration of mechanical devices made with silicon must take into account the mechanical behavior and properties of single-crystal silicon (SCS) which can be seen in Table 4.1 [Petersen].

	Yield Strength (10^{10} dyne/cm^2)	Knoop Hardness (kg/mm^2)	Young's Modulus (10^{12} dyne/cm^2)	Density (gr/cm^3)	Thermal Conductivity (W/cm°C)	Thermal Expansion (10^{-6}/°C)
*Diamond	53	7000	10.35	3.5	20	1.0
*SiC	21	2480	7.0	3.2	3.5	3.3
*TiC	20	2470	4.97	4.9	3.3	6.4
*Al_2O_3	15.4	2100	5.3	4.0	0.5	5.4
*Si_3N_4	14	3486	3.85	3.1	0.19	0.8
*Iron	12.6	400	1.96	7.8	0.803	12
SiO_2						
SiO_2 (fibers)	8.4	820	0.73	2.5	0.014	0.55
*Si	7.0	850	1.9	2.3	1.57	2.33
Steel (max strength)	4.2	1500	2.1	7.9	0.97	12
W	4.0	485	4.1	19.3	1.78	4.5
Stainless Steel	2.1	600	2.0	7.9	0.329	17.3
Mo	2.1	275	3.43	10.3	1.38	5.0
Al	0.17	130	0.70	2.7	2.36	25

Table 4.1 : Comparison of several materials. *single crystal values. [Petersen]

Although SCS is a brittle material, it is not fragile. From the table it can be seen that the Young's modulus has a value approaching those of stainless steel and nickel, and well above that of quartz. Silicon's hardness is close to quartz, just below chromium, and almost twice as high as nickel and iron. SCS crystals have a tensile yield strength of $6.9x10^{10}$ dyne/cm^2, which is at least 3 times higher than stainless steel wire [Petersen].

Although high-quality SCS is intrinsically strong, the apparent strength of a particular mechanical component or device will depend on its crystallographic orientation and geometry, the number and size of surface, edge, and bulk imperfections, and the stressed induced and accumulated during growth, polishing, and subsequent processing. When these considerations have been properly accounted for, one can hope to obtain mechanical components with strengths exceeding that of the highest strength alloy steels [Petersen].

4.2 MEMS fabrication techniques

Microengineering refers to the practice of making three dimensional structures and devices on the micrometer scale [Banks]. There are two main techniques for microengineering: microelectronics and micromachining. Microelectronics, producing electronic circuitry on silicon chips, is a very well developed technology. Micromachining is the name for the techniques used to produce the structures and moving parts of microengineered devices [Banks].

One of the main goals of microengineering is to be able to integrate microelectronic circuitry into micromachined structures, to produce completely integrated systems. Such

systems could have the same advantages of low cost, reliability and small size as silicon chips produced in the microelectronics industry.

The remainder of this chapter introduces three of the micromachining techniques that are in use or under development: the Excimer Laser, LIGA, and silicon micromachining. Silicon micromachining is given the most prominence, since this is one of the better developed micromachining techniques.

4.2.1 Excimer laser micromachining

Excimer lasers produce relatively wide beams of ultraviolet laser light. One interesting application of these lasers is their use in micromachining organic materials (plastics, polymers, etc). This is because the Excimer laser doesn't remove material by burning or vaporizing it, unlike other types of laser, so the material adjacent to the area machined is not melted or distorted by heating effects.

When machining organic materials the laser is pulsed on and off, removing material with each pulse. The amount of material removed is dependent on "the material itself, the length of the pulse, and the intensity of the laser light. Below a certain threshold intensity, dependent on the material, the laser light has no effect. As the intensity is increased above the threshold, the depth of material removed per pulse is also increased" [Banks]. It is possible to accurately control the depth of the cut by counting the number of pulses. Quite deep cuts (hundreds of microns) can be made using the Excimer laser.

A chrome on quartz mask, like the masks produced for photolithography, controls the shape of the structures produced. In the simplest system the mask is placed in contact with the material being machined, and the laser light is projected through it. Excimer lasers can be employed for example, as part of the LIGA process described in the next section.

4.2.2 LIGA

The acronym LIGA comes from the German name for the process (Lithographie, Galvanoformung, Abformung) [Banks]. LIGA uses lithography, electroplating, and molding processes to produce microstructures. It is capable of creating very finely defined microstructures of up to 1000μm high.

In the process as originally developed, a special kind of photolithography using X-rays (X-ray lithography) is used to produce patterns in very thick layers of photoresist. The X-rays from a synchrotron source are shone through a special mask onto a thick photoresist layer (sensitive to X-rays) which covers a conductive substrate. This resist is then developed.

The pattern formed is then electroplated with metal. The metal structures produced can be the final product, but it is common to produce a metal mold. This mold can then be filled with a suitable material, such as a plastic, to produce the finished product in that material.

As the synchrotron source makes LIGA expensive, alternatives are being developed. These include high voltage electron beam lithography which can be used to produce structures on the order of 100μm high, and Excimer lasers capable of producing structures up to several hundred microns high [Banks].

Electroplating is not limited to use with the LIGA process, but may be combined with other processes and more conventional photolithography to produce microstructures.

4.2.3 Silicon micromachining

The techniques for depositing and patterning thin films can be used to produce quite complex microstructures on the surface of silicon. Electrochemical etching techniques are being investigated to extend the set of basic silicon micromachining techniques. Silicon bonding techniques can also be utilized to extend the structures produced by silicon micromachining techniques into multilayer structures.

4.2.3.1 Basic techniques

There are three basic techniques associated with silicon micromachining. These are the deposition of thin films of materials, the removal of material (patterning) by wet chemical etchants, and the removal of material by dry etching techniques (bulk micromachining). Another technique that is utilized is the introduction of impurities into the silicon to change its properties (i.e., doping).

4.2.3.2 Thin films

There are a number of different techniques that facilitate the deposition or formation of very thin films (on the order of micrometers, or less) of different materials on a silicon wafer. These films can then be patterned using photolithographic techniques and suitable etching techniques. Common materials include silicon dioxide (oxide), silicon nitride (nitride), polycrystalline silicon (polysilicon or poly), and aluminum.

4.2.3.3 Surface micromachining

Bulk micromachining involves forming microstructures by etching away the bulk of the silicon wafer to achieve the desired result. On the other hand, surface micromachining techniques build up the structure in layers of thin films on the surface of the silicon wafer (or any other suitable substrate). Surface micromachining will be discussed at this point since it is the process that MCNC's MUMPS uses.

The process typically employs films of two different materials, a structural material (commonly polysilicon) and a sacrificial material (oxide). These are deposited and dry etched in sequence. Finally the sacrificial material is wet etched away to release the structure. The more layers, the more complex the structure, and the more difficult it becomes to fabricate.

A simple surface micromachined cantilever beam is shown in Figure 4.1 [Banks]. A sacrificial layer of oxide is deposited on the surface of the wafer. A layer of polysilicon is then deposited, and patterned to a beam with an anchor pad (Figure 4.1a). The wafer is then wet etched to remove the oxide layer under the beam, freeing it (Figure 4.1b). The

anchor pad has been under etched. However, the wafer was removed from the etch bath before all the oxide was removed from under the pad leaving the beam attached to the wafer.

An advantage of this approach is that it closely resembles the process from which VLSI circuits are developed. Thus, by using a similar process it may be possible to use or extend current VLSI development tools to have MEMS development capabilities. This approach is obviously more favorable than having to "re-invent the wheel".

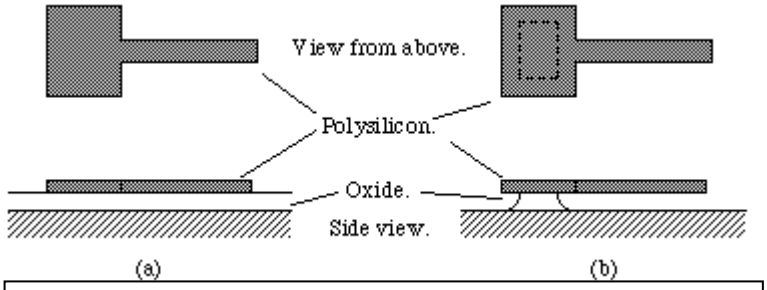


Figure 4.1 Micro-machined cantilever beam [Banks]

Chapter 5
Description of tools and materials used in project

5.1 Design

This project has limited itself to examining the process of surface micromachining. The tools and materials chosen for this project were chosen due to their being readily available and useful for this process.

5.1.2 L-Edit

Many layout tools for VLSI design are available. These layout tools are adequate for MEMS design, but lack features that would make them more valuable. For example, MAGIC [MAGIC] and LASI [LASI] are two common VLSI mask layout design tools. However, MAGIC lacks a sophisticated user interface. Thus, it has a large learning curve. LASI is similar to MAGIC except that it has a basic user interface included. However, it is purely a two dimensional tool. These two tools, although being freeware, are very limited in their abilities and lack features that will make them more useful to MEMS design.

L-EDIT [Tanner] is an integrated layout editor packaged with automatic design rule checker, and automatic place and route tools. It allows for output in three formats: its own binary format and two widely accepted formats, GDSII and CIF (Caltech Intermediate Form) [Mead]. It has a well-developed user interface and is easy to learn.

One feature that is useful for MEMS design is its cross-sectional viewer. However, it lacks any 3-D capabilities. To use this layout editor with the MUMPS process, it is necessary to edit the setup to use the MUMPS technology file that is available from Tanner Tools. To use the cross-sectional viewer for MUMPS designs it is necessary to obtain the MUMPS cross-sectional view file from Tanner also. One drawback to L-Edit is that it is proprietary.

5.1.2 Caltech Intermediate Form (CIF)

Storing VLSI and MEMS layouts can be done in a variety of forms. Many tools have a binary format available, such as L-Edit's own binary format (TBF). However, a binary format is designed for system efficiency, but not for human readability.

The CIF form is a means of describing graphic items (mask features) of interest in VLSI and MEMS designs. Its purpose is to serve as a standard machine-readable representation from which other forms can be constructed for specific output devices such as plotters, video displays, and in our case, various simulators. It is not intended as a symbolic layout language. Nevertheless, CIF files are fairly readable.

The basic idea of the form is to specify every geometric object in the design using high precision. Some advantages of using this form are that it provides design groups easy access to output devices other than their own, enables sharing designs with others, allows combining several designs to form a larger chip, and the like. Table 5.1 lists the major commands and their forms:

Command	Form
Polygon with a path	P path
Box with length, width. Center, and direction (direction defaults to (1.0) if omitted)	B integer integer point point
Round trash with diameter and center Wire with width and path	R integer point
Wire with width and path	W integer path
Layer specification	L shortname
Start svmbol definition with index, a, *b* (a and b both default to 1 if omitted)	DS integer integer integer
Finish symbol definition	DF
Delete symbol definitions	DD integer
Call symbol	C integer transformation
User extension	digit userText
Comments with arbitrary text	(commentText)
End marker	E

Table 5.1 Common Commands in CIF format [Mead]

5.1.2.1 Semantics

The underlying task of this intermediate form is to describe without confusion the pattern geometries for VLSI circuits and MEMS designs. It is necessary for all writers and users of CIF files to have the same understanding of how the file is to be understood.

CIF form uses a right-handed coordinate system, with x increasing to the right and y increasing upward. The units of distance measurements are hundredths of a micron. Instead of using angles, CIF uses a pair of integers to specify a direction vector. This helps avoid the need for trigonometric functions and avoids the problem of choosing units of angular measure. The first integer is the component of the direction vector along the x-axis, and the second integer along the y-axis.

5.1.2.2 Geometric primitives

CIF form has four built in geometric primitives: the box, polygon, flash, and wire. Each primitive geometry element must be labeled with the name of a fabrication mask on which it belongs. The layer names are given according to the process that will be used to fabricate the design. In our case, the MUMPS process will define the layer names.

5.2 Target physical process: MUMPS

Just as in VLSI design, MEMS design is dependent upon the fabrication technology used. Therefore, the layout description is dependent upon the fabrication technology since it will be described as elements of specific layers used in the chosen technology to create masks. It is imperative to choose a widely accepted and well-defined process so that any work done would be usable by the greatest number of MEMS designers. In addition, choosing an obscure process would not only benefit just a few designers, but would lead to rapid obsolescence if the process is not widely supported. The possible processes are a MOSIS CMOS [MOSIS] process and the MUMPS [MUMPS] process.

CMOS is the standard process used for making VLSI circuits. MOSIS is a foundry which supports this process. MOSIS offers several CMOS processes today, with feature sizes from 1.50 microns to 0.18 microns. By selecting the appropriate process, designers can access five metal layers, two polysilicon layers, NPN transistors, linear capacitors, and MEMS devices [MOSIS]. MCNC's Multi-User MEMS Process or MUMPS has been developed in a DARPA-supported program and is used by industry, government, and academic institutions. Either of these processes would mesh well with the other tools

chosen. However, since we had access to a chip fabricated with the MUMPS process, MUMPS was chosen instead of a CMOS process.

MUMPS is a three-layer polysilicon surface micromachining process. In this standard process:

1. polysilicon is used as the structural material,
2. deposited oxide PSG is used as the sacrificial layer, and
3. silicon nitride is used as electrical isolation between the polysilicon and the substrate.

The process is different from most customized surface micromachining processes in that it is designed to be as general as possible, and to be capable of supporting many different designs on a single silicon wafer. Since the process was not optimized with the purpose of fabricating any one specific device, the thicknesses of the structural and sacrificial layers were chosen to suit most users, and the layout design rules were chosen conservatively to guarantee the highest yield possible.

Figure 5.1 demonstrates how the layers of the MUMPS process are ordered. The sacrificial layers, oxide 1 and 2, have not been removed yet. When the sacrificial layers are removed, a micro-motor has been created. This can be seen in Figure 5.2.

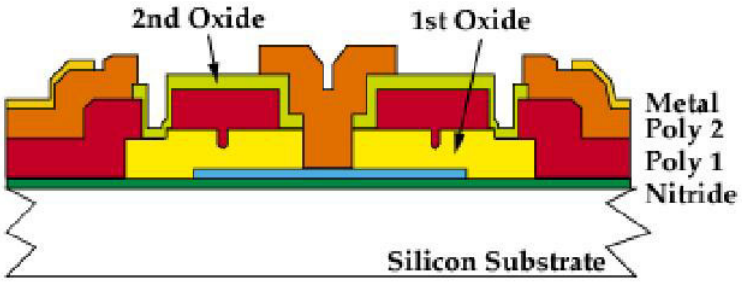


Figure 5.1 Layers of MUMPS process with sacrificial layers still present [Koester]

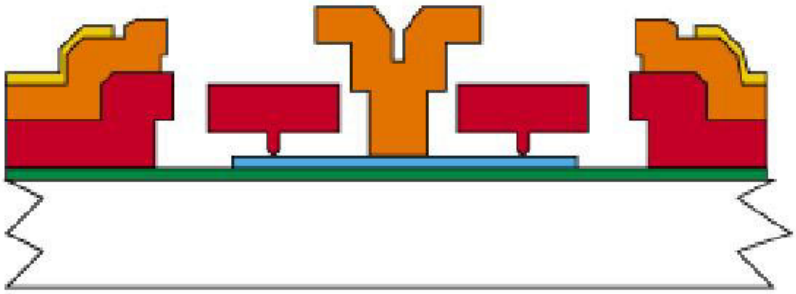


Figure 5.2 MUMPS process with sacrificial layers removed [Koester]

5.3 Simulation tools

5.3.1 SPICE

SPICE is a general-purpose electric-circuit simulation program. The name stands for Simulation Program with Integrated Circuit Emphasis. The allowed components are resistors, capacitors, inductors, mutual inductances, independent dc and ac sources, dependent sources, transmission lines, diodes, and transistors [Tuinenga].

SPICE is capable of performing three main types of analysis. It can determine the dc behavior of selected output voltages with respect to changes in input voltages. A second type of analysis that is usually required in order to fully determine a circuit's behavior is called a transient analysis. Transient analyses calculate circuit voltages and currents with respect to time. This assumes that there is a time-dependent object that causes an effect on the rest of the circuit. The third type of analysis that SPICE can perform is called an ac analysis. This type is also referred to as a sinusoidal steady-state analysis. Voltages and currents are calculated as a function of frequency. In an ac analysis output variable changes are calculated in response to changes in the amplitude, frequency or phase of sinusoidal input voltage or current sources.

The advantage of SPICE is that it has a well-developed, built-in simultaneous equation solver. However, these equations must be input in a circuit description. This means that to do a mechanical simulation, a domain translation must be performed.

5.3.2 VHDL-AMS & SEAMS

VHDL is the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL is an international standard specification language for describing digital hardware used by industry worldwide. VHDL enables hardware modeling from the gate to system level. VHDL provides a mechanism for digital design and reusable design documentation. VHDL is the outcome of a U.S. Government request for a new means of describing digital hardware. The need for a common language to describe and

communicate digital design was clear [Ashenden]. VHDL has numerous advantages [VHDL]:

1) Design Methodology: VHDL supports many different design methodologies (top-down, bottom-up, delay of detail) and is very flexible in its approach to describing hardware.
2) Technology Independence: VHDL is independent of any specific technology or process. However, VHDL code can be written and then targeted at many different technologies.
3) Wide Range of Descriptions: VHDL can model hardware from a high level to a low level. VHDL can describe hardware from the standpoint of a "black box" to the gate level. VHDL also allows for different descriptions of the same component and allows the designer to mix behavioral descriptions with gate level descriptions.
4) Standard Language: The use of a standard language allows for easier documentation and the ability to run the same code in a variety of environments. Communication among designers and among design tools is enhanced by a standard language as well.
5) Design Management: Use of VHDL constructs, such as packages and libraries, means that common elements can be shared among members of a design group.
6) Flexible Design: VHDL can be used to model digital hardware and other types of systems.

The IEEE has standardized VHDL. The current standard is VHDL 1076-1993 [Standards97]. VHDL-AMS (where the AMS stands for Analog and Mixed Signal) is an

effort to standardize an extension of VHDL 1076 to support the description and the simulation of analog and mixed-signal circuits and systems. Formally, VHDL-AMS is known as VHDL 1076.1. VHDL-AMS is a strict superset of VHDL 1076-1993 [Standards97].

Where VHDL 1076 deals with the discrete domain, VHDL-AMS extends to the continuous domain. Since VHDL-AMS continuous models are based on differential algebraic equations (DAE's), any domain that can be expressed with DAE's can be modeled in VHDL-AMS. Thus, differential equations that describe the state variable solutions of systems in the mechanical, thermal, etc., domains (many of which are of the same form as the differential equations in the electrical domain) can be included.

An advantage of VHDL-AMS models is that domain translations are not necessary for mixed systems. Their behaviors may be specified using algebraic equations or ordinary differential equations. A disadvantage is that VHDL-AMS does not have symbolic calculation capabilities. It is limited to numerical solving. Also, it cannot solve PDE's directly, because there is only one independent variable, which is time.

SEAMS stands for ***S****imulation* ***E****nvironment for VHDL-****AMS.*** This is an analog and mixed signal simulator. The system takes as an input a VHDL/VHDL-AMS description and goes through the following stages: parsing, elaboration, code generation and simulation. The entire system rests on a Time Warp system developed at the University of Cincinnati [SEAMS]. Optimistic Synchronisation Protocols [Frey] are used to

implement this mixed signal simulator. SEAMS partially implements the Language Reference Manual for VHDL-AMS [Standards97].

5.3.3 ANSYS

The response of most real-world engineering systems to applied actions is usually difficult, if not impossible, to determine by a closed-form mathematical solution. The finite element method offers a convenient way of obtaining approximate solutions to just about any engineering problem.

The name *finite element* summarizes the basic concept of the method: the transformation of an engineering system with an *infinite* number of unknowns (the response at every location in a system) to one that has a finite number of unknowns related to each other by elements of finite sizes.

The unknowns, called degrees of freedom, represent the responses to applied actions. The degrees of freedom and the actions are related by a set of basic equations. The purpose of the finite element method is to determine the solution of these equations across the entire engineering system being analyzed. The simplest form of a basic equation is as follows [Ansys]:

$$[K]\{d\} = \{A\}$$

where {d} is the degree of freedom vector, {A} is the action vector, and [K] is the matrix relating {d} to {A} and is often called the stiffness or coefficient matrix. In general, [K] and {A} are known, and {d} is initially unknown.

The type of analysis being performed determines the actual form of a basic equation. For example, in a static structural analysis as in Figure 5.3, the equation is:

$$[K]\{u\} = \{F\}$$

where [K] is the structural stiffness matrix, {u} is the displacement vector, and {F} is the force vector. In the spring mass system, K is constant since the model is one-dimensional. u represents the distance the mass is displaced from its original position. F represents the force applied to the mass to displace the mass distance u.

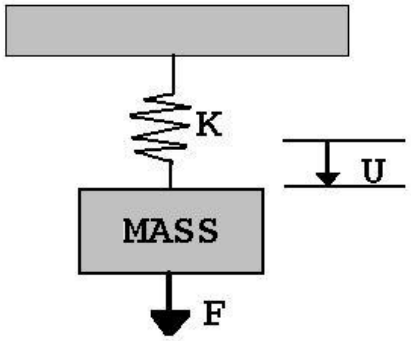


Figure 5.3 Static spring mass system

To obtain solution data for the entire system being analyzed, the [K] matrices for the individual elements are assembled into a global [K] matrix. This task is not difficult since the elements are connected to each other mathematically by their nodes. The resulting global set of simultaneous equations can then be solved for the unknowns or degrees of freedom.

ANSYS uses a frontal (wavefront) equation solver which performs the assembly and solution steps in parallel. Once the degrees of freedom are determined, derived results are calculated within each element using its shape functions. Stress and strain would be examples of derived results for a structural element.

ANSYS is considered an industry standard for mechanical simulation. ANSYS was chosen in this project to be a control in which to compare the mechanical results derived from the VLSI tools. One disadvantage of ANSYS is that it does not have any built in electrical capabilities. Thus, for the electrical part of simulation, a domain translation is necessary. ANSYS allows input from files, which allows for the creation of a template input file in which parameters for our cantilever beams can be introduced.

5.3.4 MATHEMATICA

MATHEMATICA is a useful tool for those who do quantitative analysis, symbolic calculations and manipulations, as well as for those who want to visualize functions or data [Wolfram]. With it one can calculate, model, prototype, and analyze results.

MATHEMATICA is an interpreted language. This means that it reads an expression, evaluates the result, and then prints it out. MATHEMATICA is programmable. One is able to create functions on one's own. MATHEMATICA has built into the language many of the primitives and constructs found in C, FORTRAN, and Pascal. In addition to

procedural programming, MATHEMATICA supports rule-based programming using pattern matching.

Mathematica performs three basic types of computation: numerical, symbolic, and graphical. It works with numbers of arbitrary magnitude and precision, as well as with polynomials, power series expansions, matrices, and graphs. Mathematica provides standard symbolic operations of algebra and calculus, including integration and differentiation.

Version 3.01 was the version used for this project. It has the advantage that it has an intuitive user interface resulting in ease of use. This tool does not require that models be input using domain translation and is not restricted to only algebraic equations and ordinary differential equations. One disadvantage of MATHEMATICA is that it does not have any built in electrical or mechanical capabilities that may be exploited.

This package was used to create a template file that contained the equations for the model of the cantilever beam. This file accepts the parameters of the beam that are extracted from the layout file.

5.3.5 MechanicsExplorers: Add on package for MATHEMATICA [Kaufmann]

This is a user-friendly program for the bending of beams. This package uses the Euler-Bernoulli theory for small deflections of thin elastic straight beams. The basic function of the package is *SolveBeam.* It calculates the shear force S_y, the bending moment M_z,

and the deflection d and slope s for beams with given loads (discrete forces, distributed forces, and discrete moments), supports (fixed or simple) and hinges. The bending stiffness EI_z can be described by arbitrary functions. E is Young's modulus, and I_z is the main moment of inertia about the z-axis.

The solution is calculated by integrating the following well-known differential equations where f_z is the distributed force in z-direction:

$$\frac{\delta S_y(x)}{\delta x} == -f_z(x)$$

$$\frac{\delta M_z(x)}{\delta x} == -S_y(x)$$

$$\frac{\delta^2 d(x)}{\delta x^2} == \frac{M(x)}{EI_z}$$

The advantage of this tool is that it extends MATHEMATICA to having built in mechanical solving capabilities. However, it only extends it to one mechanical device, the cantilever beam.

Using this package a template was generated which allows us to enter the parameters that are extracted for the cantilever beam.

Chapter 6
Model of cantilever beam

For this project, we have limited our analyses to the deflection of the free end of a cantilever beam (with uniform cross section), which is fixed at one end, with a force applied to the tip of the free end of the beam. For this analysis we assume that the deflection at the fixed end is for all practical purposes zero. There are two types of analyses that can be performed: static and dynamic. The static case is not of much interest since it examines the deflection of a beam under a constant load. The dynamic case is of more interest to the MEMS designer since many applications require controlled motion of the cantilever beam. Both types of analyses will be mentioned below.

6.1 Static beam analysis

If we are given a constant force applied to the free end of the beam, we can analyze its static deflection. For this problem the standard equation for a maximum deflection of a cantilever beam of constant cross section is given as [Blake]:

$$D = \frac{WL^3}{3EI}$$

where D is the deflection of the free end of the beam, W is the force applied to the free end, L is the length of the beam, E is Young's modulus and I is the moment of inertia of the beam.

6.2 Dynamic beam analysis

Of practical interest to us is the case of a beam with a force (which is permitted to change) applied at the free end. This force may be mechanical in nature, or it may be that

the force is the result of the electrical attraction between two structural elements in the device. The beam has length L, being fixed rigidly at x = 0, and a force of the form Fcosωt applied at x = L. The equation of motion is given as [McCallion, p.89]:

$$\frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\rho}{E}\frac{\partial^2 u(x,t)}{\partial t^2} = 0$$

where u(x,t) is the deflection at position x at time t. This equation applies to all cross-sections of the beam, and as the end cross-section at x=L is acted upon by a force of period 2π/ω, we look for the steady-state solution of the form:

$$u(x,t) = U(x)\{C\sin\omega t + D\cos\omega t\}$$

By substitution and meeting the end conditions, the vibratory motion of the beam may be expressed as [McCallion]:

$$u(x,t) = \frac{F\sin\alpha x\cos\omega t}{AE\alpha\cos\alpha L}$$

6.3 Alternative dynamic beam analysis

Since we have simplified the problem, and are only concerned with the deflection at the free end of the beam, we can approximate the deflection of the cantilever beam tip if we model the tip as a dampened, spring-mass mechanical system as in Figure 6.1.

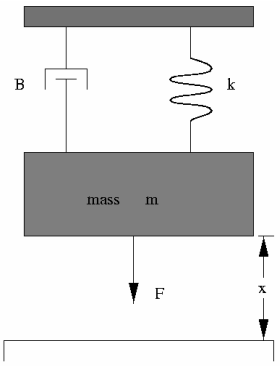


Figure 6.1 Dampened spring mass system

The mass of the system would be:

$$m = wht\rho$$

where *w, h, t,* are the width, height, and thickness of the beam, and ρ is the density of the material of which the beam is composed. The spring constant, *k*, for a cantilever beam is given as:

$$k = \frac{Ea^3 t}{l^3}$$

where *l* is the length of the beam. The differential equation obeyed by the mass *M* in Figure 6.1 is given by [Lo]:

$$F_{ext} = M\frac{d^2x}{dt^2} + B\frac{dx}{dt} + kx$$

In modeling the cantilever beam in this fashion, we have reduced the problem to a much simpler one. This equation can be used to create parameterized template files for the various simulators mentioned for this research. The simulation results of the various simulators for deflection can be compared with an ANSYS beam model and with actual devices for comparison. It is hoped that the results will be close enough to enable design automation of applications which would use the cantilever beam.

The above equations have been used to simulate cantilever beam behavior in MATHEMATICA, ANSYS, VHDL-AMS and SPICE. The results are summarized in Chapter 8.

Chapter 7
Extending VHDL-AMS to finite element analysis

7.1 Introduction

Since VHDL-AMS is an ordinary differential equation solver, it is ideal for solving systems of equations. We demonstrate its effectiveness and ease by modeling the cantilever beam (with the assumption that the beam is uniform). In this case, we only deal with a mechanical beam in which a constant load is applied. Therefore, we are examining the static behavior of the beam. Finite element analysis (FEA) on the beam can be done by breaking the beam into multiple elements using the equation:

$$F = KX$$

where F is the vector of forces applied to the assigned elements, K is the stiffness matrix for the beam, and X is the vector of displacements. The K matrix is actually the combination of all the stiffness matrices for each individual element of the beam. Since the endpoints of each element interact with the endpoints of an adjacent element, the K matrix takes this point into consideration. Consider a two-element beam with the elements, A and B, where A has endpoints 1 and 2 and B has endpoints 2 and 3. A has its matrix shown as

$$K^A = \begin{bmatrix} K^A{}_{11} & K^A{}_{12} \\ K^A{}_{21} & K^A{}_{22} \end{bmatrix}$$

And B has its corresponding matrix shown as

$$K^B = \begin{bmatrix} K^B{}_{11} & K^B{}_{12} \\ K^B{}_{21} & K^B{}_{22} \end{bmatrix}$$

K^A_{ij} represents the stiffness coefficient for the corresponding $F=KU$ equation at the corresponding node. For example, for element A of Figure 7.1, the stiffness coefficient K^A_{11} is the coefficient for the equation which goes with the force applied at node 1 and the displacement at node 1. K^B_{ij} is analogous to K^A_{ij} , but for element B. Therefore, the corresponding stiffness matrix for a beam with two elements will be

$$K = \begin{bmatrix} K^A{}_{11} & K^A{}_{12} & 0 \\ K^A{}_{21} & K^A{}_{22} + K^B{}_{11} & K^B{}_{12} \\ 0 & K^B{}_{21} & K^B{}_{22} \end{bmatrix}$$

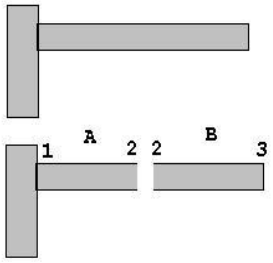


Figure 7.1 Two element

7.2 Modeling beam in VHDL-AMS with FEA

We have written a program (see Appendix III) that generates a VHDL-AMS model for a cantilever beam given the dimensions of the beam and the number of elements. The

models are generated rapidly using this program. The program concentrates on filtering out useless terms, such as ignoring the 0 terms in the K matrix and using the boundary condition which assumes that the deflection at the fixed end is 0. In Figure 7.2 we show a model generated for a beam modeled with only one element.

The model in Figure 7.2 was reduced from a two by two matrix to just one element due to the boundary condition that deflection at Node 0 is 0. So column 1 and row 1 of the matrix are eliminated from consideration. Thus the deflection at node 1, called V1, is dependent only on the stiffness of the single element.

```
-----------------------------------
entity FEABEAM is
end entity FEABEAM;
-----------------------------------
architecture behavior of FEABEAM is
  constant F0: real:=1.0e-5;
  constant  L: real := 80.0e-6;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := 3.0*EI/L3;
  constant L2 :real := L*L;
  quantity V1 : real;
begin
    V1 == F0/k;
end behavior;
-----------------------------------
```

Figure 7.2 FEA model of beam with one element

Figure 7.3 describes the model generated for a beam partitioned into five finite elements. The model in Figure 7.3 results in a system of five equations and five unknowns.

FEA may be incorporated into VHDL-AMS designs to aid in the rapid prototyping of systems. In the future, the program will be modified so that the models created will input the parameters via input file. This will aid the designer since the designer will not have to continue to recompile the models each time a set of beam parameters or different number of elements is chosen. As FEA is incorporated into VHDL-AMS, greater flexibility will be given to MEMS designers in the future.

```
-----------------------------------
entity FEABEAM is
end entity FEABEAM;
-----------------------------------
architecture behavior of FEABEAM is
  constant F5: real:=1.0e-5;
  constant  L: real := 80.0e-6/5.0;
  constant  W: real := 20.0e-6;
  constant  H: real := 2.0e-6;
  constant EZ: real := 170.0e9;
  constant IZ: real := (W*H*H*H)/12.0;
  constant EI: real := EZ*IZ;
  constant L3 :real := L*L*L;
  constant k : real := EI/(16.0*L3);
  constant L2 :real := L*L;

  quantity V1: real;
  quantity V2: real;
  quantity V3: real;
  quantity V4: real;
  quantity V5: real;

begin
     V1 == (2.0*V2- V3);
     V2 == 2.0*V1;
     V2 == -2.0*V3 + V4;
     V3 == 2.0*V4 - V5;
     V4 == (k*V5 - F5)/k;
end behavior;
---------------------------------
```

Figure 7.3 FEA model of 5 element beam

7.3 Results of FEA on cantilever beam

The FEA models generated were simulated on the SEAMS [SEAMS] simulator. The results were compared to ANSYS FEA models of an elastic 1-D static beam with the same number of elements. ANSYS was used for the comparison since it has been shown to be highly reliable for these types of measurements. Tables 7.1 and 7.2 show results for a beam with dimensions 80x20x2 μ^3 with a constant load applied to the free end of the beam for both the 3 element and 5 element cases.

	VHDL-AMS	ANSYS	Difference
Node	microns	microns	%
0	0.0000	0.0000	0.0000
1	0.2509804	0.11155	124.93
2	0.5019608	0.39041	28.57
3	0.7529412	0.75294	0.00547

Table 7.1 Results for 3 element beam comparison

Node	VHDL-AMS (microns)	ANSYS (microns)	Difference %
0	0.0000	0.0000	0.0000
1	0.05782588	0.42165	37.1419
2	0.1156518	0.15661	26.1466
3	0.1734776	0.32527	46.6665
4	0.4626071	0.53007	12.7271
5	0.7517365	0.75294	0.1598

Table 7.2 Results for 5 element beam comparison

The errors for the FEA model are somewhat high for beams with fewer elements. However, the error at the end of the beam is negligible. Some of the error may have been introduced with some simplifying assumptions. One may create a more detailed model for FEA; this should be able to reduce the errors at each node to an acceptable level.

7.4 Advantages of this approach

The above examples illustrate how VHDL-AMS can be used for system design involving multiple energy domains. The language gives a unified approach to dealing with multiple domains. VHDL-AMS also allows for the definition of physical types. For example, "time" is a standard VHDL-AMS type. This feature facilitates understanding of domain interactions and also simplifies translations of units between energy domains. VHDL-AMS encourages concentration on system rather than component considerations, encapsulates low-level information, encourages hierarchical, evolutionary design and reuse, and provides component designers with concrete specifications for their work. It is compatible with the use of component libraries which are already being developed. In addition, it provides a comfortable path into the MEMS design area for electrical and computer engineers. It also encourages the development of MEMS tools which interface well with current hardware / software design tools. It encourages decoupling of system design from low-level physical considerations, while providing support for simulations, which take physical behavior into account. It should be possible to provide VHDL-AMS interfaces to powerful MEMS simulation systems [Senturia, b] already under development. VHDL-AMS supports simulation of dynamic system behavior and can model both continuous and discrete events, thereby providing support for the simulation of complex physical systems.

7.5 Possible drawbacks

Some limitations of the approach we have described here include the effort required to interface VHDL-AMS to existing simulators, the present lack of graphical interfaces for

VHDL-AMS modeling and simulation, and the lack of symbolic computation. Some flaws in VHDL itself have been pointed out [Ghosh], which may necessitate some redesign both of VHDL and of VHDL-AMS. But we believe that the advantages listed above are more than sufficient to justify our methodology.

Chapter 8
Process for extracting behavioral data

8.1 Construction of cantilever beams in MUMPS

There appear to be multiple ways to construct cantilever beams using the MUMPS process. Once can divide the set of possible cantilever beams into two categories. The first are those that are intended to be purely mechanical in nature. In this case it is not necessary to take into account the separation of the ground plane and the source plane. However, the cantilever beams that are of interest are the ones that are formed such that the layer of polysilicon for the structural part of the beam is electrically isolated from the ground plane (a lower level of polysilicon), and some distance lies between the beam and the ground plane. Therefore, every cantilever beam's base will be contained in the nitride layer which serves as an electrical insulator from the substrate. Thus, a cantilever beam, for our purposes, is a layer of polysilicon which is isolated from a lower level of polysilicon, the lower level of polysilicon is contained in the nitride layer. The nitride layer and the lower layer of polysilicon will not reside beneath the beam except for the minimal distance required to properly anchor the beam. The upper layer of polysilicon will extend beyond the anchored base. The following four cases have been considered (Figure 8.1):

1) Layer Poly 1 extends beyond the base, and is isolated from Poly 0.
2) Layer Poly 2 extends beyond the base, and is isolated from Poly 0.
3) Layer Poly 2 extends beyond the base, and is isolated from Poly 1.

4) Layers Poly 1 and 2 are acting as one layer by the Thin Oxide Cut and extend beyond the base, and are isolated from Poly 0.

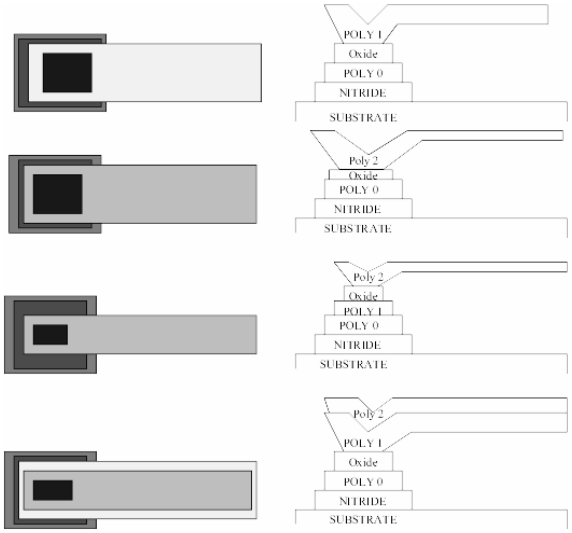


Figure 8.1 Types of recognizable beams

The four cases also have the following constraints:

- The lower layer of polysilicon in the base resides completely in the nitride layer.
- All bases of cantilever beams are contained in a nitride rectangle which only undercuts the beam by the minimal spacing required to sufficiently anchor the beam to the base layers.

8.2 Extraction of cantilever beams from CIF files

With the understanding that these four types of beams can be categorized, then a procedure can be produced which can translate a physical layout design into useful information for various simulators. This procedure is given in diagram form in Figure 8.2.

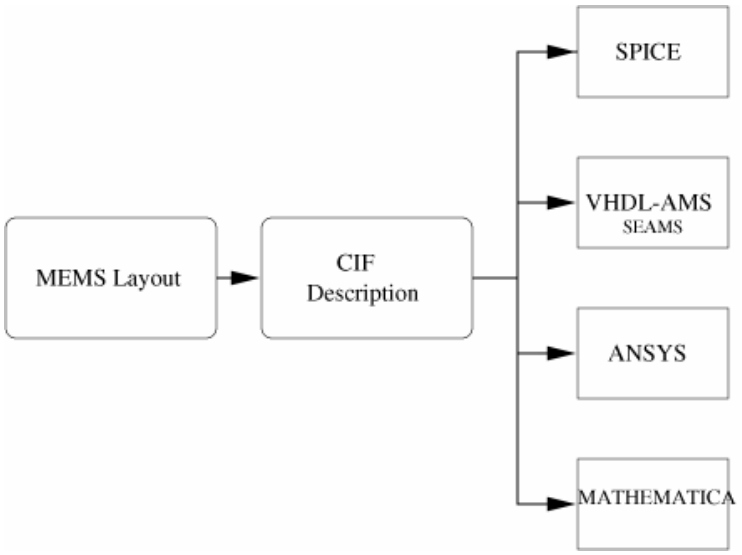


Figure 8.2 Process to extract behavioral data for simulation

A program in C++ (see Appendix I) has been written to read CIF files and to detect cantilever beams present in the layout which correspond to the four cases above with the corresponding restrictions. An array of linked lists is created. Each index of the array represents the layer number in the MUMPs process. Each index holds a list of all rectangles of the corresponding layer in the layout. An array of Boolean values is also created. The contents of each index indicates whether there are any occurrences of the corresponding layer in the layout. If the layers of a particular case are in the layout, then the lists for those layers are scanned to see if they match the conditions and restrictions. If a cantilever beam is found, it is stored in a list of cantilever beams with all pertinent information (such as length, height, thickness, width) for simulation. The following is a high-level pseudocode description:

```
//entry point for program
void main()
{
	Layer Layer_Lists[14]; //declare list for each layer.
	Beam Beam_List[4]; //declare list for each beam type
	Read(CIF File);
	//Create a separate list for each layer type in process
	//Each list created contains all boxes of that layer type
	//in the layout.
	Generate_Layer_Lists(Layer_Lists);
	//Create a list of all beams found of each type.
	Beam_List_1 = Scan_Beam_Type_1(Layer_Lists);
	Beam_List_2 = Scan_Beam_Type_2(Layer_Lists);
	Beam_List_3 = Scan_Beam_Type_3(Layer_Lists);
	Beam_List_4 = Scan_Beam_Type_4(Layer_Lists);
	//Extract parameters for each beam found into an output file.
	Write_Beam_Parameters_To_files(Beam_List[0]);
	Write_Beam_Parameters_To_files(Beam_List[1]);
	Write_Beam_Parameters_To_files(Beam_List[2]);
	Write_Beam_Parameters_To_files(Beam_List[3]);
}
Generate_Layer_Lists(Layer Layer_Lists[])
{
	//This function reads a CIF file, the list for each layer in MUMPS process
	//and separates each box in file to proper layer list.
	//An array of the lists created is returned.
	CIFobject C;
	while (not end of CIF file)
	{
		C = get_CIF_object(current file pointer);
		If (C is a box)
		{
			layer_name=get_box_layer_name();
			add_box_to_layer(Layer_Lists);
		}
	}
}
Beam_List* Scan_Beam_Type_1(Layer Layer_Lists[])
{
//This function takes the lists of layers and scans them, looking for beams of type 1.
//It returns a list of all beams of type 1 that were found.
	For(each element of Poly_1 that is contained in element Poly_0)
	{
		for (each element of Poly_2 that has width contained in Poly_1)
		{
			for(each element of First_Oxide_Cut contained in intersection)
			{
				Write_Parameters_To_Beam_List(1);
			}
		}
	}
}
```

8.3 Translation of cantilever beams for simulation

Currently, work is being done in studying how to translate layout data into data for simulation. Dimensional analysis is a future area of exploration for the translation of data. Since the simulations require units from both the electrical and mechanical domains, tedious conversions were done by hand to convert data into common reference units. For this work, this was acceptable, but for a systems approach, this step must be automated.

At this point the capabilities of various simulators are being explored. The simulators that are currently being explored are SPICE, SEAMS, ANSYS, and MATHEMATICA. Since SPICE is an electrical simulator, its input file requires that the input be in terms of electrical devices. The cantilever beam behaves electrically like a parallel plate capacitor in which the distance between the two plates is allowed to vary. Mechanically, the cantilever beam behaves like a mass connected to a spring such that the mass is able to move up or down in the vertical plane. It is also possible to convert this mechanical domain description into an equivalent electrical domain description. Thus, the cantilever beam can be simulated electrically and mechanically with SPICE. This is advantageous since SPICE tends to be easy to use and not time consuming, whereas FEA simulation is generally time consuming, although generally more accurate. The mechanical simulation results can be compared to the mechanical simulation results from FEA simulators such as ANSYS. It is hoped that, when this project is finished, an appropriate level of abstraction for behavioral extraction will be determined.

8.4 Format for SPICE input file

All elastic parameters are stored in the data structure when the cantilever beam is extracted. These are inserted into the SPICE program. The parallel RLC circuit description for a single cantilever which is purely mechanical in nature is as follows:

```
*comment or circuit description
R1 0 1 1/B
C1 0 1 m
L1 0 1 1/k
I1 1 0 (...) with parameters for current source.
Simulation data inserted here.
.END
```

This format can be repeated for each beam found in the layout with its own specific B, m and k values in the same input file. It will consider data after the .END statement as a new simulation.

For a beam that is actuated by a voltage source, the model must include a feedback loop. A mechanical force exerted at the end of the beam models the force of attraction due to the capacitance. Since the force is related to the deflection, the force needs to be recalculated at each time step for the new distance of the beam. However, this step was ignored for SPICE and only the mechanical beam was simulated. Figure 8.3 shows the response for a sample beam. The following is the actual SPICE code to generate the graph in Figure 8.3:

```
*circuit description
VIN     1       0       1volts
R20     2       0       255089
R23     2       3       10
Rmeas   1       2       1e5
L30     3       0       1.88235e-2      IC=0
C20     2       0       28.928e-12      IC=0
.TRAN   1e-7    300e-7 UIC
.PRINT  TRAN V(1)       I(R20)   I(C20) I(L30)
.PROBE
.OPTIONS NOPAGE
.END
```

where *Rmeas* has been added to aid in calculating the input current (or Force in energy domain translation, since $I_{in} = V_{Rmeas}/Rmeas$). Resistor R20 has been added to correct the reactive time for the inductor L30.

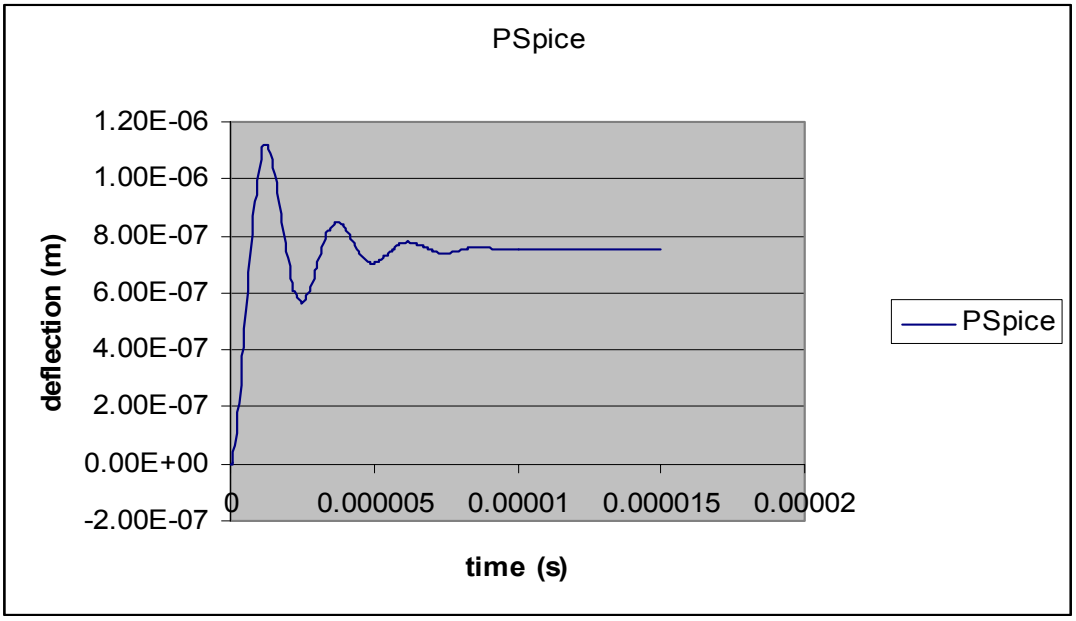


Figure 8.3 80x20x2 μ^3 beam with constant force of 1.0×10^{-5} N applied

8.5 MATHEMATICA modeling

Just as with the SPICE modeling, the same parameters are inserted into a model which solves for the deflection. MATHEMATICA has the capability to solve the second order differential equation for the deflection using the DSOLVE command. Therefore, no domain translation is necessary. The following code is a sample file generated for a 80x20x2 $(\mu m)^3$cantilever beam and the output can be seen in Figure 8.4:

```
F[t_] = 1/10^5;
Clear[S,t,x]
S[t_,x_] = m*x''[t] +B*x'[t] + k*x[t]
answer=DSolve[{S[t,x]==F[t],x[0]==0,x'[0]==0},x[t],t]
Clear[soln];
soln[t\_] = x[t]/.answer[[1]];
CC = 2*Sqrt[m*k];
B = (2/10)*CC;
soln[t];
L = 80.0/10^6;
W = 20.0/10^6;
H = 2.0/10^6;
P = 2.26*10^(3);
Y = 170*10^9;
IZ = (W*H^3)/12;
Rigidity = Y*IZ;
m = P*W*L*H;
k = N[(3*Rigidity)/L^3,30];
soln[t];
Plot[soln[t],{t,0,1.0 10^(-5)}];
```

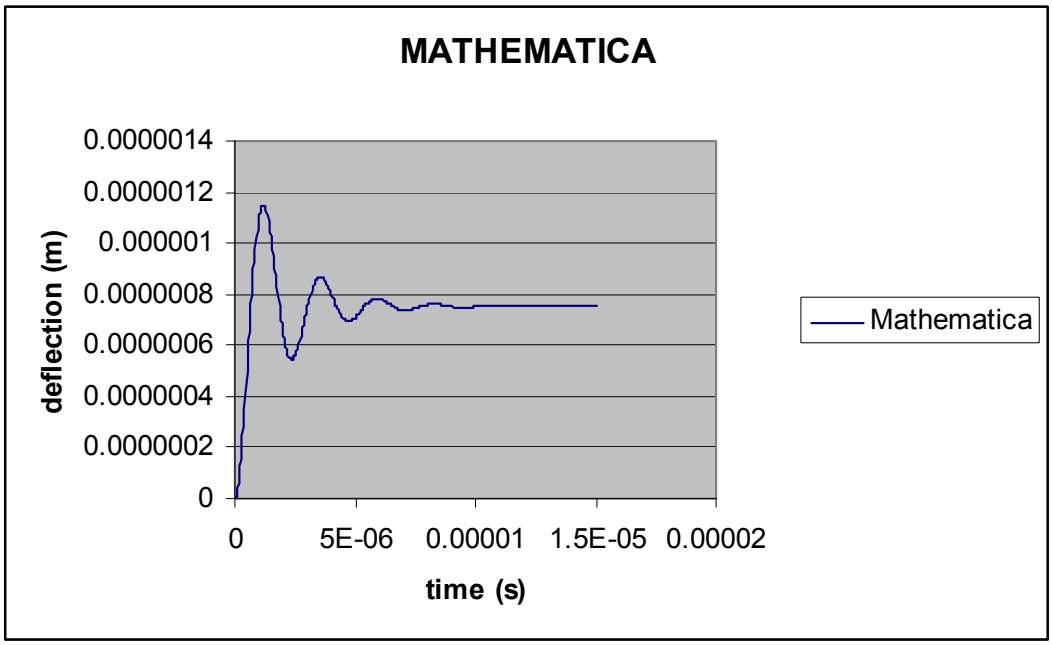


Figure 8.4: 80x20x2 μ^3 beam with constant force of $1x10^{-5}$ N applied

Section 8.6 VHDL-AMS modeling

Although SPICE has differential equation solvers built into it, VHDL-AMS offers the same advantage that MATHEMATICA does, at least for ordinary differential equations. For SPICE it has been shown that parameters have to be input as electrical quantities for SPICE to run. However, VHDL-AMS has a differential equation solver and there is no restriction as to the type of quantities that must be input. Therefore, this eliminates the translation of the mechanical system into an electrical system (which should reduce any errors due to approximations). The equations for the mechanical system can be entered into VHDL-AMS directly. This allows for the description to be done in a high level manner. For instance, the cantilever beam could be described as:

```
entity cantbeam is
end entity cantbeam;

architecture simple of cantbeam is
        quantity x: real;
        quantity v: real;
        quantity a: real;
        quantity F: real;
        constant L: real := 20.0E-6;
        constant W: real := 10.0E-6;
        constant H: real := 1.5E-6;
        constant P: real := 2.26E3;
        constant M: real := L*W*H*P;
        constant Y: real := 170.0E9;
        constant IZ: real:= (W*H*H*H)/12.0;
        constant Rigidity: real := Y*IZ;
        constant k: real := (3.0*Rigidity)/(L*L*L);
        constant B: real := 0.4*sqrt(M*K);
begin
        b1: break v => 0.0, a => 0.0, x => 0.0, F => 0.0;
        force: F == F1 + F2 + F3;
        force1: F1 == M*a;
        force2: F2 == k*x;
        force3: F3 == B*v;
        vel: v == x'dot;accel: a == v'dot;
end architecture simple;
```

The declared constants represent data that is either extracted or properties derived from extracted data. In this model the values of the dimensions of the mechanical cantilever beam appear as constants. However, this is not the most efficient way since this requires that a new model be created for each beam with different dimensions. The time to create an executable for simulation is extensive. It is better to use variables and read the values of the dimensions at run-time. The following model is for a cantilever beam that is actuated by a voltage source. This model incorporates reading the dimensions of the beam from an input file. This allows an executable to be created once, and only an input file needs to be generated by the extraction program.

```
USE std.textio.ALL;
entity cantbeamactuator is
        port(Vin,Length,Width: in real;
        deflection,Force: out real);
end entity cantbeamactuator;

architecture simple of cantbeamactuator is
        quantity x,Volt,v,a,F,W,H,M,IZ,Rigidity,K: real;
        quantity k: real;
        constant P: real := 2.26E3;
        constant Y: real := 170.0E9;
        constant Eo: real:= 8.85E-12;
        constant H: real := 1.5E-6;
begin
        b1: break Volt => 0.0, a => 0.0, x => 0.0, F => 0.0;
        InputTestBench: Process
                File Infile : text OPEN READ_MODE IS "cantbeam.in";
                VARIABLE linebuf : line;
                VARIABLE vtemp,ltemp,wtemp,htemp :real;
                BEGIN
                        WHILE (NOT (endfile(Infile))) LOOP
                                readline(Infile,linebuf);
                                read(linebuf,vtemp);
                                Vin <= vtemp;
```

```
                    read(linebuf,ltemp);
                    Length <= ltemp;
                    read(linebuf,wtemp);
                    Width <= wtemp;
                    WAIT FOR 100 ns;
                END LOOP;
            END process;
        Initialize: Process (Vin,Length,Width)
            VARIABLE MASS,MOMENT,RIG,SpringK: real;
            BEGIN
                Volt := Vin;
                L := Length;
                W := Width;
                MASS := L*W*H*P;
                M := MASS;
                MOMENT := (W*H*H*H)/12.0;
                IZ := MOMENT;
                RIG := Y*IZ;
                Rigidity := RIG;
                B := 0.4*sqrt(M*K);
                SpringK := (3.0*Rigidity)/(L*L*L);
                k := SpringK;
            END process;
        Calc_Force: Process
            VARIABLE Area, Perm,AttractForce: real;
            BEGIN
                Area := L*W;
                Perm := Eo;
                AttractForce := (Area*Perm*Volt*Volt)/(2.0*x*x);

                F := AttractForce;
            END process;
        Force_applied: F == F1 + F2 + F3;
        force1: F1 == M*a;
        force2: F2 == k*x;
        force3: F3 == B*v;
        vel: v == x'dot;
        accel: a == v'dot;
        deflection = x;
        Force = F;
end architecture simple;
```

Once a description for the beam has been created, simulations can be done using SEAMS [Seams]. SEAMS implements the IEEE 1076.1 VHDL-AMS standard [Standards97]. Figure 8.5 gives the output of a SEAMS simulation on an 80x20x2 μm^3 beam.

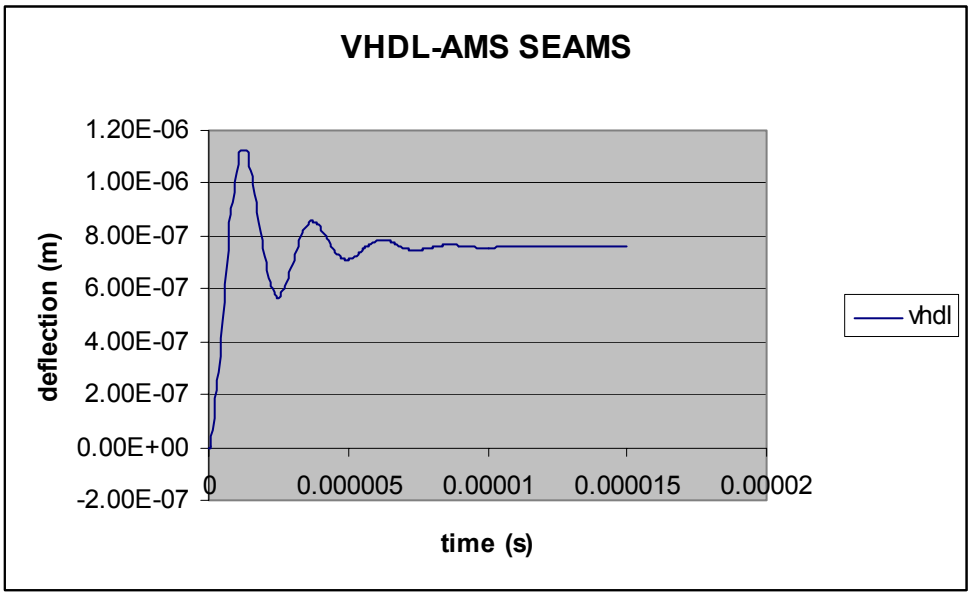


Figure 8.5: 80x20x2 m3 beam with a constant force 1.0x10-5 N applied

Section 8.7 FEA analysis ANSYS

Interfacing with the ANSYS FEA program is not a difficult task [ANSYS]. The format of the source program contains a sequence of statements in 4 groups: Header, Preprocessor, Solution, and Postprocessor. The Header section allows the user to specify BATCH mode and a title. In the Preprocessor section, /PREP7, three tables are created describing the element properties: Element type (ET), Real constants (R), and Material Properties, (MP). After the ET, R, and MP lines are defined, the nodes and the elements are defined. In the Solution section /SOLUTION, the loads and constraints are defined. Loads are defined with the F, SFBEAM and SF commands. The element type in ANSYS for a cantilever beam is 3. The Real constants are entered from the data structure which holds

relevant information on the beam such as length, width, height. The Material Properties are also entered into the file using the elastic properties of polysilicon which are also stored from the cantilever beam extraction program. The values extracted are then written to the template input log file as variables at the beginning. Then the variable names are used thereafter in the model. The model was set up using a built-in model in ANSYS for a cantilever beam fixed at one end. The following is an example of the source program for a cantilever beam with values written in for the variables:

```
/BATCH
/* comment: Define the beam dimensions
length = 100e-6
width = 20e-6
depth = 2e-6
xsect = (width*depth)
inertiaz = (width*depth**3)/12
Force =10e-6
Youngs = 170e9
density = 2.26e3
dampcoef = .2
!*
/NOPR
/PMETH,OFF
KEYW,PR_SET,1
KEYW,PR_STRUC,1
KEYW,PR_THERM,0
KEYW,PR_FLUID,0
KEYW,PR_ELMAG,0
KEYW,MAGNOD,0
KEYW,MAGEDG,0
KEYW,MAGHFE,0
KEYW,MAGELC,0
KEYW,PR_MULTI,0
KEYW,PR_CFD,0
/GO
!*
/COM,
/COM, Structural
!*
/PREP7
!* comment: Begin assigning  the beams properties to variables
```

```
ET,1,BEAM3
!*
!*
R,1,xsect,inertiaz,depth, , , ,
!*
!*
UIMP,1,EX, , ,Youngs
UIMP,1,DENS, , ,density
UIMP,1,ALPX, , , ,
UIMP,1,REFT, , , ,
UIMP,1,NUXY, , , ,
UIMP,1,PRXY, , , ,
UIMP,1,GXY, , , ,
UIMP,1,MU, , , ,
UIMP,1,DAMP, , ,dampcoef,
UIMP,1,KXX, , , ,
UIMP,1,C, , , ,
UIMP,1,ENTH, , , ,
UIMP,1,HF, , , ,
UIMP,1,EMIS, , , ,
UIMP,1,QRATE, , , ,
UIMP,1,MURX, , , ,
UIMP,1,MGXX, , , ,
UIMP,1,RSVX, , , ,
UIMP,1,PERX, , , ,
UIMP,1,VISC, , , ,
UIMP,1,SONC, , , ,
!*
K,1,0,0,0
K,2,length,0,0
/pnum, kp,1
kplot
LSTR,1,2
ESIZE,length/10,0,
LMESH, 1
/PNUM,KP,0
/PNUM,LINE,0
/PNUM,AREA,0
/PNUM,VOLU,0
/PNUM,NODE,0
/PNUM,SVAL,0
/NUM,0
!*
/PNUM,ELEM,1
/REPLOT
!*
```

SAVE,,,
FINISH
/SOLU
*!**
ANTYPE,0
FLST,2,1,1,ORDE,1
FITEM,2,1
D,P51X, , , , , ,ALL
FLST,2,1,1,ORDE,1
FITEM,2,2
F,P51X,FY,-Force

This model is used as an input file to ANSYS, and the type of simulation that is desired can be run, whether it be a static or dynamic. Figure 8.6 shows the output generated when a dynamic solution is chosen.

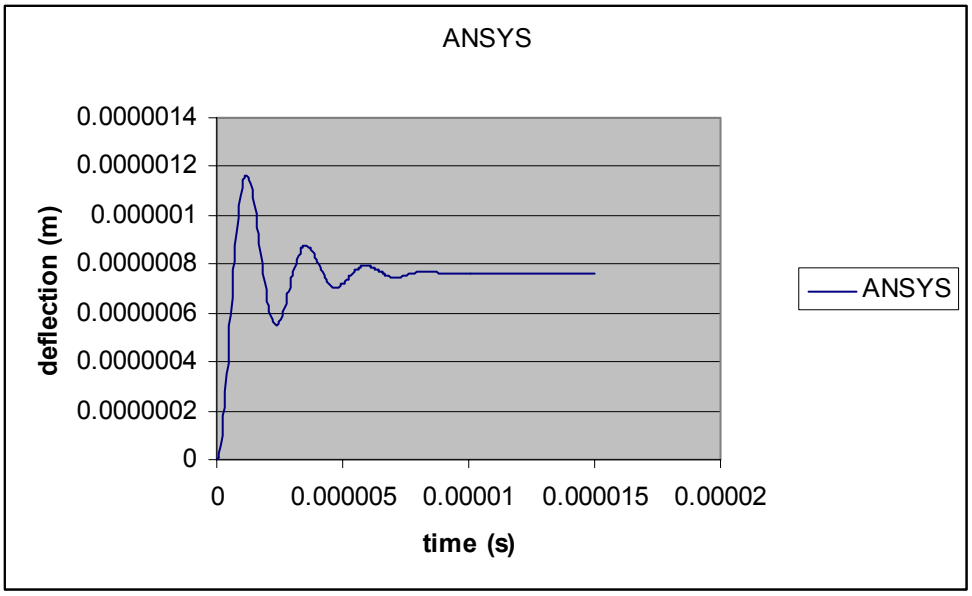


Figure 8.6: 80x20x2 m3 beam with a constant force 1.0x10-5 N applied

Section 8.8 Results for extracting behavioral data

Simulations were made on various beam sizes for each of the simulators with various forces applied. The beam dimensions that were simulated ranged in length from 20 to 100 micrometers, and widths for each of these beams ranged from 5 to 20 micrometers. For each of these beams, forces were applied in increments until the maximum deflection of the beam occurred. Each of the simulators gave similar results for the input given. For each simulator, an input file was generated with the dimensions of the cantilever beam and other pertinent information. For the simulations, static and dynamic forces were applied. Each of the simulators were able to show dampened oscillatory motion for the deflection when a force was applied. The results for the deflection of the largest and smallest beams tested with a constant force applied are shown in Tables 8.1 and 8.2. Appendix V gives the graphs of all the beams. The deflection given is the value after it has ceased to oscillate. The % error is calculated using the following formulas:

$$SPICE\ \%\ error = \frac{ABS(ANSYS - SPICE)}{ANSYS}$$

and

$$MATHEMATICA\ \%\ error = \frac{ABS(ANSYS - MATHEMATICA)}{ANSYS}$$

Deflection of 20x5x2 μm cantilever beam					
Force (10^{-4} N)	ANSYS (μm)	SPICE (μm)	SPICE % error	MATHEMATICA (μm)	MATHEMATICA % error
1.0	0.47059	0.4705	0.009	0.4704	.019
2.0	0.94118	0.941176	0.0004	0.941176	0.0004
3.0	1.4118	1.411768	0.0032	1.411768	0.0032
4.0	1.8824	1.88235	0.005	1.88235	0.005

Table 8.1 Sample deflection comparison (20x5x2 µm^3 beam)

Deflection of 100x20x2 μm cantilever beam					
Force (10^{-6} N)	**ANSYS (μm)**	**SPICE (μm)**	**SPICE % error**	**MATHEMATICA (μm)**	**MATHEMATICA % error**
1.0	0.14706	0.147101	0.0041	0.14710	0.004
2.0	0.29412	0.294202	0.0082	0.29420	0.008
3.0	1.4118	1.411768	0.0032	1.411768	0.0032
4.0	0.44118	0.441303	00123	0.44130	0.012

Table 8.2 Sample deflection comparison (8020x2 µm^3 beam)

Chapter 9
Design automation of MEMS systems using behavioral modeling

If there is to be automation for the design of MEMS systems, there must be a separation between the designer and the low-level design of components. So, it is imperative that there exist a set of primitive devices which have parameters (such as dimensions and specifications of criteria) from which the high-level designer may choose. The designer should be able to automatically generate these devices from a behavioral description into "cells" for a layout design. Thus, the designer is enabled to create the layout of the circuit geometry at a higher, more abstract level. In addition, by using these parameterized components, which have been tested for performance, the high-level designer will have added confidence in the final performance of the system (Figure 9.1).

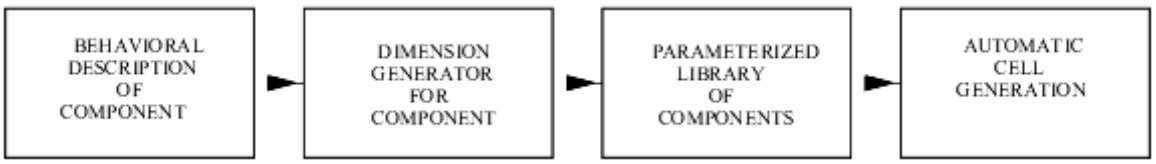


Figure 9.1 Flow for design automation

In order to create this set of primitive MEMS devices, it is necessary to create models for each primitive device. One way currently being explored [MCNC] to create a parameterized device is to create a mathematical model for the device. The user can enter the dimensions of the device. At this point, the model may be simulated. The user can

change the parameters until the desired result is attained. Such parameterized libraries are being created by Tanner [Tanner, a], and by MCNC [MCNC]. For instance, MCNC maintains the Consolidated Micromechanical Element Library, CaMEL. The library consists of two independent parts; the nonparameterized cell database and the parameterized microelectromechanical element library, PME. Their objective is to provide MEMS cell libraries that are useful not only to novice MEMS designers, but also to experienced designers.. Both libraries are intended to assist the user in the design and layout of MEMS devices by providing an initial layout for components of a MEMS system. It is assumed that the user will modify these elements and customize them as desired and assemble designs using a suitable mask layout editor. However, we concede that this is still an iterative process which would be very time consuming.

Another more interesting way is our proposal to allow the user to enter the device by behavioral modeling. In this method the user would choose a component from a library of parts, and specify criteria, such as maximum area the device may use and the voltage range at which the device must operate to obtain the desired result. In this case the mask layout of the component should be automated and the high-level designer should only be provided the layout cell that meets the specified constraints.

We illustrate our approach by a case study of an actual MEMS system designed and constructed at the University of Cincinnati. This chapter will address some of the problems encountered with the current state of MEMS system design, suggested solutions to some of these problems, and what was actually achieved. The design of one MEMS

primitive, a micro-mirror, which was used in the system, is used as an example of how the process of automating MEMS design may be accomplished. We now begin our discussion of the application of the micro-mirror.

9.1 Basics of the micro-mirror

The micro-mirror device that we are considering is basically a cantilever beam with a rectangular plate connected to the end of the beam (Figure 9.2). In addition, directly beneath the plate is another plate, which is electrically isolated from the upper plate. This lower plate is used as a drive for the upper one. When a voltage is applied between the two plates, they act as a parallel plate capacitor. Thus, when a voltage is applied deflection of the beam occurs due to the force of attraction between the two plates. This voltage can be used to actuate the mirror [Osterberg]. We can measure the deflection of the beam for any voltage, but we are especially interested in the case of the "pull-in" voltage (the voltage at which the plates are brought together, or to a stopper which keeps the two plates from touching [Gilbert]. At pull-in and higher voltages the mirror would be considered in the down position. The mirror can be returned to the up position by reducing the voltage. Therefore, the mirror could be controlled with an input voltage to oscillate between the up and down positions. In this case the mirror could be used as a "light switch" to send optical signals to a receiver [Hare].

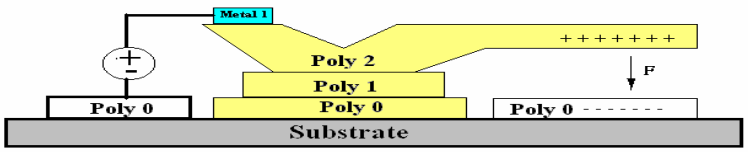


Figure 9.2 Cantilever beam actuator

9.2 Concentration of the research

Our research concentrates on design automation of the micro-mirror. The actual construction of the micro-mirror was done manually using AUTOCAD as a layout mask editor and drawn by hand [Hare]. Many micro-mirrors were used in the system and many additional mirrors were included only for testing purposes. In addition, some of the mirrors incorporated a unique technique of "corrugating" the beam in an effort to reduce the actuation voltage of the micro-mirror. These micro-mirrors were of varying dimensions, and thus took a great deal of time and effort to construct. In addition, because the design was custom built, problems in the design were not detected until after the system had been fabricated and was being tested. It is believed that such a system could have been constructed in less time and avoided the design problems, if a design system for automating the design had been used.

9.3 Method for automating the design of MEMS devices

Four things are necessary for the automation of MEMS system design:

1) A well-developed library of parameterized components needs to exist.
2) Using this library, each component needs to have a mathematical model which describes the behavior of the component given specific dimensions for the component.
3) Algorithms are needed to generate the correct dimensions of the component which will meet the behavioral specification given.
4) From the dimensions calculated, a cell needs to be automatically generated by inserting the calculated dimensions into the parameterized library component.

It must be noted that these four steps are process dependent. This research bases its work on MCNC's MUMPS process [Koester]. Even though problems with the MUMPS process were encountered, it can still be used to discover what types of behavior can be handled automatically by the design tools. Currently, the library of parameterized components contains only the design for the micro-mirror. MATHEMATICA [Wolfram] was utilized to simulate the behavioral equations for the micro-mirror. This model enabled dimensions of a micro-mirror to be entered, and returned the deflection of the mirror for a given voltage. A program was written in MATHEMATICA (see Appendix II) implementing an incremental algorithm which adjusted the dimensions of the micro-mirror until the specified behavior was met. In this case, it was when the mirror deflected into the down position at a specified actuation voltage. This program could be enhanced if a binary search algorithm replaced the incremental algorithm. Once the dimensions were calculated for such a micro-mirror, the dimensions were used to create a layout mask file (CIF) of the micro-mirror. At this point, the micro-mirror layout mask file may be input into a layout editor such as L-EDIT [Tanner, b], which does accept CIF [Mead] format.

9.4 Results for design automation

Our behavioral model for the cantilever beam modeled as a dampened spring-mass system was compared using various simulators on multiple dimensioned beams. The deflection from the behavioral model compared well with that given by simulators such as ANSYS [ANSYS], and MECHANICS EXPLORER [Kaufmann]. Given the added confidence that the model worked with the classical mechanical results, the results were compared with an actual fabricated micro-mirror. The results for this mirror are recorded in Table 9.1. Dimensions for beams with pull-in voltages from 10 to 25 volts are found in Appendix VI. Our model predicted that the actuating voltage for a mirror with the given dimensions would be 30 volts. However, the actual voltage for pull-in on the beam occurred at 22.5 volts. The mirrors in the system are currently undergoing testing, and at this point results for the other fabricated mirrors are not available. The fabricated mirror for which the results were available may not give an accurate picture of the behavioral model's prediction due to the fact that it was one of the special beams which incorporated the "corrugation" that was mentioned previously. We are currently

Micro Mirror Results			
	Actual Dimensions (μm)	**Predicted Dimensions (μm)**	**% error**
Mirror Length	70	59	15.714
Mirror Width	70	59	15.714
Beam Length	30	30	0.0
Beam Width	30	30	0.0
Pull in voltage (V)	22.5	-	-
Behavioral Predicted Pull in voltage (V)	30	22.5	25.0

Table 9.1 Results of predicted dimensions vs. actual dimensions for pull in voltage

investigating how to account for the corrugation both in the MATHEMATICA model and in the standard mechanical simulators. However, it stands to reason that the actual pull-in voltage would be lower than the predicted voltage in this case, since that is the reason for the corrugation in the beam. What can be gleaned from this example is that the voltage is at least in the correct range. Table 9.1 also gives the dimensions of the mirror that have a predicted pull-in voltage at 22.5 volts.

The % error in Table 9.1 was calculated using the following equation:

$$\%\ \text{error} = \frac{ABS(Actual - \text{Pr}edicted)}{Actual}$$

Problems in the testing of many of the fabricated mirrors occurred. Many of the mirrors, once actuated, would not release due to a "sticking" effect with the oxide layer. However, the MUMPS process can be altered to avoid this problem. This is a good example of a detail which should be incorporated into the design tool and hidden from the designer. In addition, measuring the slope of the mirror was problematic in that the layers were not flat enough to get good measurements.

Chapter 10
Conclusions & future work

10.1 Extending VHDL-AMS to finite element analysis

The results in chapter 7 demonstrate that it is viable to incorporate FEA into models in VHDL-AMS. Hence, tools such as VHDL-AMS, which were designed with circuitry applications in mind, may be useful in MEMS design when mechanical components may be modeled using a traditional FEA method.

10.2 Extracting behavioral data

To aid in CAD for MEMS, a set of rules for building cantilever beams has been developed for the MUMPS process. These rules will aid in the creation of a parameterized standard cell for cantilever beams in a library. A program to extract parameters for cantilever beams from a CIF file format has been developed. Using these parameters, it has been demonstrated how these parameters may be used to automate input for various simulators, such as SPICE, SEAMS, ANSYS and MATHEMATICA. SPICE requires that a mapping of mechanical properties to electrical properties be performed. It appears that dimensional analysis would be practical to help in these mappings. These mappings can be input to SPICE or SEAMS, although VHDL-AMS and MATHEMATICA do not require the domain translation. In addition, it seems that SPICE is rather limited in its abilities, due to necessary domain translations. However, VHDL-AMS and MATHEMATICA do show promise. Both have the capability to solve ordinary differential equations, which allowed the model proposed to be demonstrated in a

straightforward, high level manner. However, MATHEMATICA seems to be the most useful. It has the capability to solve not only ordinary differential equations, but also partial differential equations. Also, changing a MATHEMATICA model and simulating is not as time consuming as it is in the VHDL-AMS simulator, SEAMS. Although traditional methods for CAD, such as FEA, will probably always be necessary, this work hopes to demonstrate that other methods for specific problems may be useful to the system designer as MEMS technology matures.

10.3: Design automation

Since insufficient measurements have been gathered, it is not clear whether the behavioral model for the micro-mirror is adequate for the parameters being determined. If it turns out that the model is inadequate, then certain simplifying assumptions must be eliminated from the model. However, it is clear that, even with an overly simplified model of the problem, the results are in the correct range.

Once adequate behavioral models and parameterized component libraries are developed, MEMS design can be taken to a higher level of abstraction. The designer can be removed from the low level mask descriptions and deal with components as in VLSI design. In addition, issues such as the problem of "sticking" should be handled by the design tools automatically and not involve the designer. Thus, once such a paradigm for MEMS design is accepted, a system, such as the optical processor in this case study, could be designed at a higher level of abstraction, and the problems that were encountered could have been eliminated by the design tools.

10.4 Future work

We have presented examples showing how existing tools such as SPICE, MATHEMATICA, ANSYS, and VHDL-AMS can be used to further the development of commercially viable MEMS systems. One method [Lo] is to use one tool such as SPICE, and translate all the models to a specific domain. In this case all models would be converted to the electrical domain. However, this is a great deal of work and is not always straightforward. Another method mentioned is to incorporate all these domain specific solvers together into one package such as MEMCAD [MEMCAD]. However, there seems to be a large amount of overhead interfacing all these tools.

It appears that there is the most hope in tools such as VHDL-AMS and MATHEMATICA. These two tools have mathematical equation solvers built in. The ability to describe models with behavioral equations is an advantage since it gives the ability to solve different energy domains within a single model. MATHEMATICA may excel in that it has a more advanced equation solver and is not limited to solving just ordinary differential equations. However, VHDL-AMS has the advantage that it has a large amount of VLSI circuit descriptions built in. From this project, it would seem that VHDL-AMS could be a valuable tool for MEMS design in the future. This research has resulted in three publications which are listed as references [Gibson, a], [Gibson, b] and [Gibson, c].

In the future we intend to develop a library of parts like those presented here. In particular, we are developing a model of interacting arrays of cantilever beams and beam

models in which more complicated behaviors (such as fractures, e.g.) can be simulated. In addition, we will expand the FEA capabilities in our programs. Future research will also include extending our modeling techniques to other domains, including the fluidic domain, and interfacing our VHDL-AMS descriptions with domain-specific simulators.

References

[Ansys] Ansys **Primer for Stress Analysis for revision 4.4**, Swanson Analysis Systems, Inc., P.O. Box 65, Johnson Rd., Houston, PA, 15342-0065, copyright 1991.

[Ashenden] P. Ashenden, "*The VHDL Cookbook, First Edition*", copyright 1990, Dept. Computer Science, University of Adelaide, South Australia.
[Banks] http://www.ee.surrey.ac.uk/Personal/D.Banks/ueng.html, "Introduction to Microengineering", copyright 1996.

[Blake] Alexander Blake, **Practical Stress Analysis in Engineering Design**, Marcel Dekker Inc., copyright 1990, pp. 211-221.

[Clough] Ray W. Clough, Joseph Penzien, **Dynamics of Structures**, McGraw-Hill, Inc., copyright 1975, pp.18-77.

[Dewey] A. Dewey, and E. Icoz, "*Visual Modeling and Design of Microelectromechanical Systems (MEMS) Transducers*," Modeling and Simulation of Microsystems, Semiconductors, Sensors, and Actuators (MSM), April 1999.

[Frey] P. Frey and R. Radhakrishnan and H. Carter and P. Wilsey, *"Optimistic Synchronization of Mixed-Mode Simulators"*, Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, pp. 694-700, ISBN 0-8186-8403-8, March 30-April 3, IEEE Computer Society, Los Alamitos, 1998.

[Gabriel] Kaigham J. Gabriel, **"***Engineering Microscopic Machines*", Scientific American, Sept. 1995, pp. 150-153.

[Gibson, DA] D. Gibson, A. Hare, F. Beyette, Jr., and C. Purdy**,** *"Design Automation of MEMS Systems using Behavioral Modeling*", Ninth Great Lakes Symposium on VLSI, Ann Arbor Mich. (ed. R.J. Lomax and P. Mazumder), March 1999, pp. 266-269.

[Gibson, a] D. Gibson, A. Hare, F. Beyette, Jr., and C. Purdy, "*Design Automation of MEMS Systems using Behavioral Modeling*",Ninth Great Lakes Symposium on VLSI, Ann Arbor, MI (ed. R. J. Lomax and P. Mazumber), March 1999, pp. 266-269.

[Gibson, b] D. Gibson and C. Purdy, "*Extracting Behavioral Data from Physical Descriptions of MEMS for Simulation*" Analog Integrated Circuits and Signal Processing 20, 1999, 227-238.

[Gibson, c] D. Gibson, H. Carter, and C. Purdy, "*The use of hardware description languages in the development of microelectromechanical systems"*, *Journal of Analog Integrated Circuits and Signal Processing* 28 (2), Aug. 2001, 173-180.

[Gilbert] J. R. Gilbert, G. K. Ananthasuresh, and S. D. Senturia, "*3D Modeling of Contact Problems and hysteresis in Coupled Electro-Mechanics*", IEEE 1996, pp. 127-132.

[Ghosh] S. Ghosh and N. Giambiasi, "***Language*** *barriers in hardware design?*", Circuits and Devices, Sept. 1999, 25-40.

[Hare] Alva E. Hare and F. R. Beyette Jr., "*Deflectable Micro-Mirror Arrays for Implementation of a Recirculating Folded Perfect Shuffle Processor*", IEEE/LEOS Summer Topical Meetings on Smart Pixels, July 1998, Monterey CA, pp. 83-84.

[IBM, a] http://www.research.ibm.com/topics/deep/storage/, Atomic Force Microscopy (AFM)- How It Works, July 2000.

[IBM, b], http://www.zurich.ibm.com/Technology/Atomic/atomic.force.html, Atomic Force Microscopy, July 2000.

[INTEL] http://www.intel.com/intel/museum/25anniv/Hof/moore.htm, Intel Museum Home Page, July 2000.

[Itoh] Toshihiro Itoh, Takahiro Ohashi, and Tadatomo Suga, "*PiezoElectric Cantilever Array For Multiprobe Scanning Force Microscopy*", Micro Electro Mechanical Systems '96, 1996, pp. 451-455.

[Kaufmann] Stephan Kaufmann, **MechanicsExplorers**, http://www.ifm.ethz.ch/kaufmann/1997.

[Koester] David A. Koester, Ramaswammy Mahadevan, Alex Shishkoff, and Karen W**.** Markus, **SmartMUMPS Design handbook including MUMPS Introduction and Design Rules (rev. 4)**, MEMS Technology Applications Center, MCNC, 3021 Cornwallis Road, Research Trangle Park, NC 27709, copyright 1996 by MCNC.

[LASI] http://cmosedu.com/cmos1/book.htm, July 2000.

[Lo] Nanping R. Lo,Eric C. Berg, , Scott R. Quakkelaar, Jonathan N. Simon, Mark Tachiki, Hee-Jung Lee, and S. J. Pister, ``*Parameterized Layout Synthesis, Extraction, and SPICE Simulation for MEMS*", ISCAS 96, May 1996, 481-484.

[MAGIC] http://www.rsl.ukans.edu/~mlinhart/magic/magic.html, July 2000.

[MATLAB] http://www.mathworks.com/products/matlab/, July 2000.

[McCallion] H. McCallion**, Vibration of Linear Mechanical Systems**, Halsted Press, copyright 1973, pp. 107-113.

[MCNC] http://mems.mcnc.org/camel.html , MCNC Camel Library, July 2000.

[Mead] Carver Mead and Lynn Conway, **Introduction to VLSI Systems**, Addison-Wesley Publishing Company, Inc., copyright 1980, pp. 115-127.

[MEMCAD] Microcosm homepage http://www.memcad.com, July 2000.

[MOSIS] MOSIS homepage. http://www.mosis.org/Aboutmosis/Tour/tour2.html, July 2000.

[Osterberg] P. M. Osterberg, R. K. Gupta, J. R. Gilbert, S. D. Senturia, "*Quantitative Models for the Measurement of Residual Stress, Poisson's Ratio, and Young's Modulus Using Electrostatic Pull-in of Beams and Diaphragms*," *1994 Solid-State Sensor and Actuator Workshop,* Hilton Head, SC, June 1994.

[Petersen] K. E. Petersen, *"Silicon as a Mechanical Material"*, IEEE Proceedings, 70 (5), may 1982, 420-457.

[Seams] University of Cincinnati, ECECS Department, Distributed Processing Laboratory, **SEAMS Simulator project,** http://www.ececs.uc.edu/hcarter/ , July 2000.

[Senturia, a] S. D. Senturia, A. Aluru, J. White, ``*Simulating the Behavior of MEMS Devices: Computational Methods and Needs*". IEEE Computational Science and Enginering 4(1), 1997, 3-13.

[Senturia, b] S. D. Senturia*, "Simulation and Design of Microsystems: A 10 Year Perspective",* Sensors and Actuators A67, 1998, 1-7.

[Silicon Designs] http://www.silicondesigns.com/tech.html, Silicon Designs, Inc. Technology Report Page, July 2000.

[Standards97] Design Automation Standards Committee, IEEE Computer Society, **IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS Changes)**, Std. 1076.1, IEEE, 1997.

[Steidel] Robert Steidel, **An Introduction to Mechanical Vibrations**, John Wiley & Sons, copyright 1989, pp. 397-416.

[SUGAR] K. Pister, SUGAR 1.0, http://bsac.eecs.berkeley.edu/~cfm/mainpage.html, July 2000.

[Tanner, a] **Tanner Tools User's Manual**, Tanner Research, Inc., 1996.

[Tanner, b] http://www.tanner.com/, Tanner Research Home, July 2000.

[TI] http://www.ti.com/corp/docs/press/company/1994/447easc.shtml, Texas Instruments press release, July 2000.

[Tuinenga] Paul W.Tuinenga, **SPICE, A Guide to Circuit Simulation and Analysis Using PSPICE**, Prentice Hall, Inc.,copyright 1992, pp. 1-6, 115-132, 147-152.

[VHDL] http://vhdl.org/vi/analog/wwwpages/tutorial/ppframe.htm, *"Analog and Mixed-Signal Extensions to VHDL Through Examples*", July 2000.

[Wolfram] Stephen Wolfram, **Mathematica: A System for Doing Mathematics by Computer**, Addison-Wesley Publishing Co., Inc., copyright 1991.

Appendix I
Code to extract behavioral data from cantilever beams

The following code corresponds to chapter 8 and extracts cantilever beams from an input file in CIF format. This code compiles and runs on Microsoft Visual C++ 6.0

```
#include"list.cpp"
#include"box.h"
#include<iostream.h>
#include<fstream.h>
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include"string.h"
#include<conio.h>

const int SIZE=80;
void Read_Input_File(LinkedList<Box> B[],bool L[]);
void Get_Input_File_Name(char F[]);
void Open_Input_File(ifstream& In,char F[]);
bool Check_If_Input_File_Exists(char F[]);
void Get_Layer_Name(char info[],char name[]);
int Mark_Layer_Used(bool Layers[],char LayerName[]);
void Add_Box_To_Layer(char lineofinput[],LinkedList<Box> BL[],int n);
void Scan_For_Cantilever_Beams(LinkedList<Box> BL[],
                                                    bool
Layersused[],LinkedList<CantBeam>&CL);
void Find_CB(LinkedList<Box> BL[],LinkedList<CantBeam>& CL, int n);
bool Anchored_In(Box A,Box B,int & case_number);
bool Contained_In(Box A,Box B);
int GetCharNum(char A[],int& Index);



void main()
{

        LinkedList<Box> Mumps_Layers[11];
        LinkedList<CantBeam> CB;
        bool Layers_Used[11]={false};
```

```
	Read_Input_File(Mumps_Layers,Layers_Used);

	Scan_For_Cantilever_Beams(Mumps_Layers,Layers_Used,CB);
	CB.Display();

}

int GetNumFromString(char A[],int& Index)
{
	char Temp[100];
	int x=Index;
	int count=0;
	while(A[x]<'0' || A[x]>'9')
		x++;
	while(A[x]>='0' && A[x] <='9')
	{
		Temp[count]=A[x];
		count++;
		x++;
	}
	Temp[count]='\0';
	Index=x;
	return atoi(Temp);
}

void Get_Input_File_Name(char F[])
{
	//Ask user to enter input file name, name of cif file
	cout<<"\nEnter Input file (including path): ";
	cin>>F;
}
bool Check_If_Input_File_Exists(char F[])
{
	//Checks whether input file exists or not
	ifstream Input;
	Input.open(F,ios::nocreate);
	if(Input.bad())
	{
		Input.clear();
		return false;
	}
	else
	{
		Input.close();
		return true;
```

```
        }
}


void Open_Input_File(ifstream& In,char F[])
{
        //checks for error on opening input CIF file.
        if (Check_If_Input_File_Exists(F))
        {
                In.open(F);
                cout<<"\nInput file "<<F<<"opened successfully"<<endl<<endl;
        }
        else
        {
                cout<<"\nerror occurred finding file.  Make sure complete path.";
                cout<<endl<<endl;
                exit(0);
        }
}

void Get_Layer_Name(char info[],char name[])
{
        //Given a string, stores the string as the layer name

        for(int i=0;info[i]!='C';i++); //moves to name in string

        name[0] = 'C';
        name[1] = info[i+1];
        name[2] = info[i+2];
        name[3] = '\0';
}

void Read_Input_File(LinkedList<Box> B[],bool LayerUsed[])
{
        //Reads a CIF file as input. Creates list of layers
        char filename[SIZE];
        char lineofinput[SIZE];
        ifstream Input;
        char Layer_Name[4]="\0";
        char Current_Layer[4]="\0";
        int Layer_Number;

        Get_Input_File_Name(filename);
        Open_Input_File(Input,filename);

        while(!Input.eof())
```

```
	{

		Input.getline(lineofinput,';');
		if(lineofinput[0]=='L')
		{
			Get_Layer_Name(lineofinput,Layer_Name);
			strcpy(Current_Layer,Layer_Name);
			Layer_Number = Mark_Layer_Used(LayerUsed,Layer_Name);
		}
		else if (lineofinput[0]=='B')
		{
		Add_Box_To_Layer(lineofinput,B,Layer_Number);  //add box
		}
	}
	Input.close();
}

int Mark_Layer_Used(bool Layers[],char LayerName[])
{
	//Marks the used array as true if layer used.
	//Used to speed up searching for beams.
	//If a layer is not used, then certain types of beams
	//need not be searched for.
	if(strcmp(LayerName,"CSN")==0)
	{
		Layers[0] = true;
		return 0;
	}
	else if(strcmp(LayerName,"CPZ")==0)
	{
		Layers[1]=true;
		return 1;
	}
	else if(strcmp(LayerName,"COF")==0)
	{
		Layers[2]=true;
		return 2;
	}
	else if(strcmp(LayerName,"COS")==0)
	{
		Layers[3]=true;
		return 3;
	}
	else if(strcmp(LayerName,"CPS")==0)
	{
		Layers[4]=true;
```

```
		return 4;
	}
	else if(strcmp(LayerName,"COT")==0)
	{
		Layers[5]=true;
		return 5;
	}
	else if(strcmp(LayerName,"COL")==0)
	{
		Layers[6]=true;
		return 6;
	}
	else if(strcmp(LayerName,"CPT")==0)
	{
		Layers[7]=true;
		return 7;
	}
	else if(strcmp(LayerName,"CCM")==0)
	{
		Layers[8]=true;
		return 8;
	}
	else if(strcmp(LayerName,"CHO")==0)
	{
		Layers[9]=true;
		return 9;
	}
	else if(strcmp(LayerName,"CHT")==0)
	{
		Layers[10]=true;
		return 10;
	}
	else
	{
		return -1;
	}
}

void Add_Box_To_Layer(char lineofinput[],LinkedList<Box> BL[],int Layer_Number)
{
	//Adds a box of a certain layer to corresponding layer list

	Box B;

	int commacount=0;
	int currentposition=1;
```

```
//check to see if box has direction given or not.
//this is done by counting if there are 2 commas or 1.
for(unsigned i=0;i<strlen(lineofinput);i++)
{
	if(lineofinput[i]==',')
		commacount++;
}
B.SetLength(GetNumFromString(lineofinput,currentposition));
B.SetWidth(GetNumFromString(lineofinput,currentposition));
B.SetCenterX(GetNumFromString(lineofinput,currentposition));
B.SetCenterY(GetNumFromString(lineofinput,currentposition));
if(commacount==1)
{

	B.SetDirX(1);
	B.SetDirY(0);

}
else
{
	B.SetDirX(GetNumFromString(lineofinput,currentposition));
	B.SetDirY(GetNumFromString(lineofinput,currentposition));
}

if(B.GetDirX()==0 && B.GetDirY()==0)
{
	B.SetDirX(1);
	B.SetDirY(0);
	B.SetAngle(0);
}
else if(B.GetDirX()==0)
{
	B.SetDirY(1);
	B.SetAngle(90);
}
else if (B.GetDirY()==0)
{
	B.SetDirX(1);
	B.SetAngle(0);
}
else if(B.GetDirX()==B.GetDirY())
{
	B.SetDirX(1);
	B.SetDirY(1);
	B.SetAngle(45);
}
```

```
        else
        {

        B.SetAngle(atan2((double)B.GetDirY(),(double)B.GetDirX())*180*7.0/22);
        }
        BL[Layer_Number].AddToFront(B);
}

void Scan_For_Cantilever_Beams(LinkedList<Box> BL[],
                bool LayersUsed[],LinkedList<CantBeam>&CL)
{
        //check to see if all layers used in type of beam
        //are used.  If so, begin searching for that
        //type of beam.  Otherwise, sip and look for other types.
        if(LayersUsed[1] && LayersUsed[2] && LayersUsed[4]) //case1:
        {
                Find_CB(BL,CL,1);
        }
        if(LayersUsed[1] && LayersUsed[6] && LayersUsed[7]) //case2:
        {
                Find_CB(BL,CL,2);
        }
        if(LayersUsed[4] && LayersUsed[5] && LayersUsed[7]) //case 3:
        {
                Find_CB(BL,CL,3);
        }
        if(LayersUsed[4] && LayersUsed[6] && LayersUsed[7]) //case 4:
        {
                Find_CB(BL,CL,4);
        }
}

void Find_CB(LinkedList<Box> BL[],LinkedList<CantBeam> &CL, int n)
{
        //Searches layer list for 4 types of beams
        int box_case;
        Box B1,B2,B3;     //Store boxes retrieved from lists.
        int p1,p2,p3;    //represent layer numbers for type.
        CantBeam Temp;
        bool F1,F2,F3;  //check to see if end of each list.
        int i,j,k;
        i=j=k=1;

        if(n==1)  //case 1,set p's to 1,2, and 4
        {
                p1=1;p2=2;p3=4;
```

```
}
else if (n==2) //case 2
{
        p1=1;p2=6;p3=7;
}
else if(n==3)
{
        p1=4;p2=5;p3=7;
}
else if(n==4)
{
        p1=4;p2=6;p3=7;
}
else
{
        cout<<"error in case number in Find_CB procedure"<<endl;
        exit(0);
}
//Check for conditions of each type of beam.
//See if boxes of each layer meet rules for beam.
F1 = BL[p1].GetInfo(i,B1);     //retrieves position i of list BL
                                                        //and stores in B1.
while(F1)
{
        F2 = BL[p2].GetInfo(j,B2);
        while(F2)
        {
                F3=BL[p3].GetInfo(k,B3);
                while(F3)
                {
                        if(Contained_In(B1,B3) &&
                        Anchored_In(B3,B2,box_case))
                        {
                                Temp.SetCaseNumber(n);
                                Temp.SetCenterX(B1.GetCenterX());
                                Temp.SetCenterY(B1.GetCenterY());
                                if(box_case==1)
                                {
                                Temp.SetLength(B3.GetWidth()-B2.GetWidth()-1);
                                        Temp.SetWidth(B3.GetLength()-2);
                                }
                                else if(box_case==2)
                                {
                        Temp.SetLength(B3.GetLength()-B2.GetLength()-1);
                                        Temp.SetWidth(B3.GetWidth()-2);
                                }
```

```
                                        else if(box_case==3)
                                        {
                                        Temp.SetLength(B3.GetWidth()-B2.GetWidth()-1);
                                                Temp.SetWidth(B3.GetLength()-2);
                                        }
                                        else if(box_case==4)
                                        {
                                Temp.SetLength(B3.GetLength()-B2.GetLength()-1);
                                                Temp.SetWidth(B3.GetWidth()-2);
                                        }
                                        Temp.SetBeam(B3);
                                        Temp.SetAnchor(B2);
                                        Temp.SetDirX(B3.GetDirX());
                                        Temp.SetDirY(B3.GetDirY());
                                        Temp.SetAngle(B3.GetAngle());
                                        bool Test = CL.AddToFront(Temp);
                                }
                                k++;
                                F3=BL[p3].GetInfo(k,B3);
                        } //end while F3
                        j++;
                        F2 = BL[p2].GetInfo(j,B2);
                } //end while F2
                i++;
                F1= BL[p1].GetInfo(i,B1);
        }  //end while F1
}

bool Anchored_In(Box A,Box B,int & box_case)
{
        //Determine if A is anchored in B
        double T1[4];
        double T2[4];

        T1[0] = A.GetCenterX()-A.GetLength()/2.0;
        T1[1] = A.GetCenterX()+A.GetLength()/2.0;
        T1[2] = A.GetCenterY()-A.GetWidth()/2.0;
        T1[3] = A.GetCenterY()+A.GetWidth()/2.0;

        T2[0] = B.GetCenterX()-B.GetLength()/2.0;
        T2[1] = B.GetCenterX()+B.GetLength()/2.0;
        T2[2] = B.GetCenterY()-B.GetWidth()/2.0;
        T2[3] = B.GetCenterY()+B.GetWidth()/2.0;

        if(Contained_In(A,B) && fabs(A.GetAngle()-B.GetAngle())<.1)
        {
```

```
                if(abs(T1[0]-T2[0])<=1.0 && abs(T1[1]-T2[1])<=1.0 &&
                        abs(T1[2]-T2[2])<=1.0&& abs(T1[3]-T2[3])>1.0)
                {
                        box_case=1;
                }
                else if(abs(T1[0]-T2[0])<=1 && abs(T1[2]-T2[2])<=1 &&
                        abs(T1[3]-T2[3])<=1&& abs(T1[1]-T2[1])>1)
                {
                        box_case=2;
                }
                else if(abs(T1[0]-T2[0])<=1 && abs(T1[1]-T2[1])<=1 &&
                        abs(T1[3]-T2[3])<=1 && abs(T1[2]-T2[2])>1)
                {
                        box_case=3;
                }
                else if(abs(T1[1]-T2[1])<=1 && abs(T1[2]-T2[2])<=1 &&
                        abs(T1[3]-T2[3])<=1&& abs(T1[0]-T2[0])>1)
                {
                        box_case=4;
                }
                return true;
        }
        else
        {
                return false;
        }

}

bool Contained_In(Box A,Box B)
{
        //returns true if box A is contained in Box B, otherwise false
        if( (B.GetCenterX()-B.GetLength()/2 > A.GetCenterX()-A.GetLength()/2) &&
                (B.GetCenterX()+B.GetLength()/2 < A.GetCenterX()+A.GetLength()/2)
                &&
                (B.GetCenterY()-B.GetWidth()/2  > A.GetCenterY()- A.GetWidth()/2)
                &&
                (B.GetCenterY()+B.GetWidth()/2  < A.GetCenterY()+ A.GetWidth()/2))
                return true;
        else
                return false;
}
```

The following are the Box and CantBeam classes used to store the data in the extraction program:

```
#ifndef BOX_H
#define BOX_H
#include<iostream.h>
#include<fstream.h>

class Box
{
        friend ostream &operator<<(ostream& Out, Box& B){Out<<"Length:
"<<B._Length<<endl<<"Width: "<<B._Width<<endl
                                                        <<"Center:
("<<B._Centerx<<","<<B._Centery<<")"<<endl
                                                        <<"Direction:
("<<B._Dirx<<","<<B._Diry<<")"<<endl
                                                        <<"Angle:
"<<B._Angle<<endl;return Out;}
        friend istream &operator>>(istream& In, Box&
B){In>>B._Length>>B._Width>>B._Centerx
                >>B._Centery>>B._Dirx>>B._Diry>>B._Angle;return In;}
public:
        Box(){_Length=0;_Width=0;_Centerx=0;_Centery=0;_Dirx=0;_Diry=0;_Angle=
0;}
//      Box(const Box&
B){_Length=B._Length;_Width=B._Width;_Centerx=B._Centerx;
//
        _Centery=B._Centery;_Dirx=B._Dirx;_Diry=B._Diry;_Angle=B._Angle;}
        //accessors
        int GetLength(){return _Length;}
        int GetWidth(){return _Width;}
        int GetCenterX(){return _Centerx;}
        int GetCenterY(){return _Centery;}
        int GetDirX(){return _Dirx;}
        int GetDirY(){return _Diry;}
        double GetAngle(){return _Angle;}
        int CalcArea(){return _Length*_Width;}
        //mutators
        void SetLength(int L){_Length=L;}
        void SetWidth(int W){_Width=W;}
        void SetCenterX(int X){_Centerx=X;}
        void SetCenterY(int Y){_Centery=Y;}
        void SetDirX(int X){_Dirx=X;}
        void SetDirY(int Y){_Diry=Y;}
        void SetAngle(double A){_Angle=A;}
```

```
        Box operator=(const Box &
B){_Length=B._Length;_Width=B._Width;_Centerx=B._Centerx;

        _Centery=B._Centery;_Dirx=B._Dirx;_Diry=B._Diry;_Angle=B._Angle;return
*this;}
        void DisplayBox(ostream Out){Out<<"Length: "<<_Length<<endl<<"Width:
"<<_Width<<endl
                                                        <<"Center:
("<<_Centerx<<","<<_Centery<<")"<<endl
                                                        <<"Direction:
("<<_Dirx<<","<<_Diry<<")"<<endl
                                                        <<"Angle:
"<<_Angle<<endl; }

private:
        int _Length;
        int _Width;
        int _Centerx;
        int _Centery;
        int _Dirx;
        int _Diry;
        double _Angle;
};

class CantBeam
{
        friend ostream &operator<<(ostream& Out, CantBeam& C){Out<<"Case
Number: "<<C._CaseNumber<<endl
                <<"Anchor: "<<endl<<C._Anchor<<endl<<"Beam: "<<C._Beam<<endl
                <<"Length: "<<C._Length<<endl<<"Width: "<<C._Width<<endl
                                                        <<"Center:
("<<C._Centerx<<","<<C._Centery<<")"<<endl
                                                        <<"Direction:
("<<C._Dirx<<","<<C._Diry<<")"<<endl
                                                        <<"Angle:
"<<C._Angle<<endl;return Out;}
        friend istream &operator>>(istream& In, CantBeam&
C){In>>C._CaseNumber>>C._Anchor>>C._Beam
                >>C._Length>>C._Width>>C._Centerx
                >>C._Centery>>C._Dirx>>C._Diry>>C._Angle;return In;}
public:
        CantBeam(){_CaseNumber=0;_Length=0;_Width=0;_Area=0;_Centerx=0;_Cent
ery=0;_Dirx=0;
                                _Diry=0;_Angle=0;}
//      CantBeam(const CantBeam&
C){_CaseNumber=C._CaseNumber;_Anchor=C._Anchor; _Beam=C._Beam;
```

```
//
        _Length=C._Length;_Width=C._Width;_Centerx=C._Centerx;
//
        _Centery=C._Centery;_Dirx=C._Dirx;_Diry=C._Diry;_Angle=C._Angle;}
        //accessors
        int GetCaseNumber(){return _CaseNumber;}
        Box GetAnchor(){return _Anchor;}
        Box GetBeam(){return _Beam;}
        int GetLength(){return _Length;}
        int GetWidth(){return _Width;}
        int GetCenterX(){return _Centerx;}
        int GetCenterY(){return _Centery;}
        int GetDirX(){return _Dirx;}
        int GetDirY(){return _Diry;}
        double GetAngle(){return _Angle;}
        int GetArea(){return _Area;}
        //mutators
        void SetCaseNumber(int CS){_CaseNumber=CS;}
        void SetAnchor(Box B){_Anchor = B;}
        void SetBeam(Box B){_Beam = B;}
        void SetLength(int L){_Length=L;}
        void SetWidth(int W){_Width=W;}
        void SetCenterX(int X){_Centerx=X;}
        void SetCenterY(int Y){_Centery=Y;}
        void SetDirX(int X){_Dirx=X;}
        void SetDirY(int Y){_Diry=Y;}
        void SetAngle(double A){_Angle=A;}
        CantBeam operator=(const CantBeam&
C){_CaseNumber=C._CaseNumber;_Anchor=C._Anchor; _Beam=C._Beam;

        _Length=C._Length;_Width=C._Width;_Centerx=C._Centerx;

        _Centery=C._Centery;_Dirx=C._Dirx;_Diry=C._Diry;_Angle=C._Angle; return
*this;}


private:
        int _CaseNumber;
        Box     _Anchor;
        Box _Beam;
        int _Length;
        int _Width;
        int _Area;
        int _Centerx;
        int _Centery;
        int _Dirx;
```

```
        int _Diry;
        double _Angle;
};

#endif
```

The following is the code for the Linked Lists used to store the boxes and cantilever beams

```
#include<iostream.h>
#include<assert.h>
#include<stdlib.h>
#include"string.cpp"

template<class ElementType> class LinkedList;  //forward declaration

template< class ElementType>
class Node
{
        friend class LinkedList<ElementType>;
public:
        Node();
        ~Node();
        void SetInfo(ElementType E);
        ElementType GetInfo();
private:
        ElementType _Info;
        Node<ElementType> * _Next;
};

template< class ElementType>
Node<ElementType>::Node()
{
        _Next = NULL;  // _Next = 0;
}

template< class ElementType>
Node<ElementType>::~Node()
{}

template< class ElementType>
void Node<ElementType>::SetInfo(ElementType E)
{
        _Info = E;   //assumption:overloaded = for element type object
}

template< class ElementType>
ElementType Node<ElementType>::GetInfo()
{
        return _Info;
}
//*****************************************************
```

```
template< class ElementType>
class LinkedList
{
public:
	LinkedList();
	LinkedList(const LinkedList<ElementType> &L);
	~LinkedList();
	void Display();
	bool AddToFront(ElementType E);
	bool RemoveFromFront();
	bool AddToEnd(ElementType E);
	bool RemoveFromEnd();
	bool Insert(int Position,ElementType E);
	bool Remove(int Position);
	bool IsEmpty();
	bool IsFull();
	void DeleteList();
	int Maximum(ElementType &M);  //returns index where Max is found
	LinkedList<ElementType> Sort();
	LinkedList<ElementType> & operator=(const LinkedList<ElementType> & L);
	int GetListSize();
	bool operator==(const LinkedList<ElementType> &L);
	bool GetInfo(int Position,ElementType &E);

protected:
	Node<ElementType> * _Head;
	Node<ElementType> * _Tail;
	int _ListSize;
};

template< class ElementType>
LinkedList<ElementType>::LinkedList()
{
	_ListSize=0;
	_Head = _Tail=NULL;
}

template< class ElementType>
LinkedList<ElementType>::~LinkedList()
{
	bool Check;
	int Size= _ListSize;
	for(int i=1;i<=Size;i++)
	{

		Check=RemoveFromFront();
```

```
                if(!Check)
                {
                        cout<<"Error in destructor, trying to remove from front"<<endl;
                        exit(0);
                }
        }
//      cout<<"In destructor listsize = "<<_ListSize<<endl;
}

template< class ElementType>
void LinkedList<ElementType>::DeleteList()
{
        bool Check;
        int Size= _ListSize;
        for(int i=1;i<=Size;i++)
        {

                Check=RemoveFromFront();
                if(!Check)
                {
                        cout<<"Error in delete list, trying to remove from front"<<endl;
                        exit(0);
                }
        }
}

template< class ElementType>
LinkedList<ElementType>::LinkedList(const LinkedList<ElementType> &L)
{
        _Head=NULL;
        _Tail=NULL;
        _ListSize=0;
        Node<ElementType>* Temp;
        Temp = L._Head;
        for(int i = 1 ;i<=L._ListSize;i++)
        {
                AddToEnd(Temp->_Info);
                Temp = Temp->_Next;
        }

}
template< class ElementType>
bool LinkedList<ElementType>::AddToFront(ElementType E)
{
        if (_ListSize==0)
        {
```

```
                _Head = new Node<ElementType>;
                if(_Head == NULL)
                        return false;
                _Head->_Info = E;
                _Tail = _Head;
                _Head->_Next = NULL;
                _ListSize++;
                return true;
        }
        else if(_ListSize==1)
        {
                _Head = new Node<ElementType>;
                if(_Head == NULL)
                        return false;
                _Head->_Info = E;
                _Head->_Next = _Tail;
                _ListSize++;
                return true;
        }
        else if (_ListSize>=2)
        {
                Node<ElementType>* Temp = new Node<ElementType>;
                if(Temp == NULL)
                        return false;

                Temp->_Info = E;
                Temp->_Next = _Head;
                _Head = Temp;
                _ListSize++;
                return true;
        }
        else
        {
                cout<<"Listsize < 0, list is corrupt";
                exit(0);
        }
}
template< class ElementType>
void LinkedList<ElementType>::Display()
{
        if (_ListSize==0)
        {
                cout<<"List is empty."<<endl;
        }
        else if (_ListSize>0)
        {
```

```
                Node<ElementType>* Temp = _Head;
                for (int i=1;i<=_ListSize;i++)
                {
                        cout<<Temp->_Info<<" ";
                        Temp = Temp->_Next;
                }
                cout<<endl;
        }
}
template< class ElementType>
bool LinkedList<ElementType>::RemoveFromFront()
{
        if (_ListSize== 0)
        {
                return false;
        }
        else if(_ListSize==1)
        {
                delete _Head;
                _Head = _Tail = NULL;
                _ListSize--;
                return true;
        }
        else if(_ListSize>1)
        {
                Node<ElementType>* Temp = _Head;
                _Head = Temp->_Next;
                Temp->_Next = NULL;
                delete Temp;
                _ListSize--;
                return true;
        }
        else
        {
                cout<<"Error trying to remove from neg. sized list"<<endl;
                exit(0);
        }

}
template< class ElementType>
bool LinkedList<ElementType>::AddToEnd(ElementType E)
{
        if (_ListSize==0)
        {
                return AddToFront(E);
        }
```

```
	else if (_ListSize>0)
	{
		Node<ElementType> * Temp = new Node<ElementType>;
		if (Temp == NULL)
		{
			return false;
		}
		else
		{
			Temp->_Info = E;
			_Tail->_Next = Temp;
			_Tail=Temp;
			_ListSize++;
			return true;
		}
	}
	else
	{
		cout<<"Error in Listsize trying to add to end"<<endl;
		exit(0);
	}
}
template< class ElementType>
bool LinkedList<ElementType>::RemoveFromEnd()
{
	if (_ListSize==0)
	{
		return false;
	}
	else if (_ListSize == 1)
	{
		return RemoveFromFront();
	}
	else if(_ListSize >1)
	{
		Node<ElementType> * Temp = _Head;
		for (int i=1;i<=_ListSize-2;i++)
		{
			Temp = Temp->_Next;
		}
		delete _Tail;
		_Tail = Temp;
		_Tail->_Next = NULL;
		_ListSize--;
		return true;
	}
```

```
	else
	{
		cout<<"Error in removeing from end, listsize is negative"<<endl;
		exit(0);
	}
}
template< class ElementType>
int LinkedList<ElementType>::GetListSize()
{
	return _ListSize;
}
template< class ElementType>
bool LinkedList<ElementType>::IsEmpty()
{
	if (_ListSize==0)
		return true;
	else
		return false;
}
template< class ElementType>
bool LinkedList<ElementType>::IsFull()
{
	Node<ElementType> * Temp = new Node<ElementType>;
	if (Temp == NULL)
		return true;
	else
	{
		delete Temp;
		return false;
	}
}
template< class ElementType>
bool LinkedList<ElementType>::Insert(int Position,ElementType E)
{
	if (Position >0 && Position<=_ListSize+1)
	{
		if (Position == 1)
			return AddToFront(E);
		else if (Position == _ListSize+1)
			return AddToEnd(E);
		else
		{
			Node<ElementType>* Temp = new Node<ElementType>;
			if (Temp == NULL)
			{
				return false;
```

```
                        }
                        else
                        {
                                Temp->_Info = E;
                                Node<ElementType> * Previous= _Head;
                                for(int i=1;i<=Position-2;i++)
                                {
                                        Previous = Previous->_Next;
                                }
                                Temp->_Next = Previous->_Next;
                                Previous->_Next = Temp;
                                _ListSize++;
                                return true;
                        }

                }
        }
        else
        {
                return false;
        }
}
template< class ElementType>
bool LinkedList<ElementType>::Remove(int Position)
{
        if (Position<1 || Position>_ListSize)
        {
                return false;
        }
        else
        {
                if (Position == 1)
                {
                        return RemoveFromFront();
                }
                else if(Position == _ListSize)
                {
                        return RemoveFromEnd();
                }
                else
                {
                        Node<ElementType>* Previous = _Head;
                        Node<ElementType> * Temp;
                        for (int i=1;i<=Position-2;i++)
                        {
                                Previous = Previous->_Next;
```

```
                        }
                        Temp = Previous->_Next;
                        Previous->_Next = Temp->_Next;
                        Temp->_Next = NULL;
                        delete Temp;
                        _ListSize--;
                        return true;
                }
        }
}
template< class ElementType>
int LinkedList<ElementType>::Maximum(ElementType &M)
{
        if (_ListSize ==0)
        {
                return 0;
        }
        else
        {
                int MaxIndex = 1;
                M = _Head->_Info;
                Node<ElementType> *  Temp = _Head;
                for (int i = 1; i<=_ListSize;i++)
                {
                        if (Temp->_Info>M)
                        {
                                M = Temp->_Info;
                                MaxIndex = i;
                        }
                        Temp = Temp->_Next;
                }
                return MaxIndex;
        }
}

template< class ElementType>
LinkedList<ElementType> & LinkedList<ElementType>::operator=(const
LinkedList<ElementType> & L)
{
        Node<ElementType> * Temp = L._Head;
        if(this != &L)
        {
                DeleteList();
                for(int i=1;i<=L._ListSize;i++)
                {
                        AddToEnd(Temp->_Info);
```

```
                        Temp = Temp->_Next;
                }
        }
        return *this;
}
template< class ElementType>
bool LinkedList<ElementType>::operator==(const LinkedList<ElementType> &L)
{
        bool flag=true;
        Node<ElementType>* T1;
        Node<ElementType> *T2;

        if (_ListSize!=L._ListSize)
        {
                return false;
        }
        else
        {
                T1 = L._Head;
                T2 = _Head;
                for(int i=1;i<=_ListSize;i++)
                {
                        if(T1->_Info!=T2->_Info)
                        {
                                flag=false;
                                break;
                        }
                        T1=T1->_Next;
                        T2=T2->_Next;
                }
                return flag;
        }
}
template< class ElementType>
bool LinkedList<ElementType>::GetInfo(int Position,ElementType &E)
{
        if(Position>=1 && Position <=_ListSize)
        {
                Node<ElementType> *Temp;
                Temp = _Head;
                for(int i=1;i<Position;i++)
                {
                        Temp=Temp->_Next;
                }
                E = Temp->_Info;
                return true;
```

```
        }
        else
        {
                return false;
        }
}
template< class ElementType>
LinkedList<ElementType> LinkedList<ElementType>::Sort()
{
        LinkedList<ElementType> L,Copy;
        int Position_M;
        ElementType E;

        Copy = *this;
        for(int i=1;i<=_ListSize;i++)
        {
                Position_M = Copy.Maximum(E);
                L.AddToFront(E);
                Copy.Remove(Position_M);
        }
        return L;
}
```

Appendix II
MATHEMATICA code
for
design automation

The following code corresponds to chapter 9 and generates the dimensions of a beam given a desired pull-in voltage.

```
(*comment:  Define all necessary equations for the beam *)
mass:=MirrorArea*P*H;
springconstant:=(3*Rigidity)/L^3;
CriticalFrequency := Sqrt[4*m*k];
DampingCoefficient := (2.0/10.0)*CC;
MomentZ := (W*H^3)/12.0;
RigidityFactor := N[Y*IZ];
BeamLength := Ceiling[(1.0/2.0)*MirrorLength*10^6]/10^6;
BeamWidth := BeamLength;

Area := MirrorLength*MirrorWidth;     (*mirror length is 20 microns*)
MirrorLength = 20/10^6;               (*mirror width  is 20 microns*)
MirrorWidth = 20/10^6;

MirrorArea = Area;
Voltage= 22.0;                (*value of desired pull-in voltage*)

H =1.5/10^6;                  (*enter constant beam characteristics*)
P = 2.26*10^(3);
Y =170.0*10^9;
Beamheight= .75/10^6;
Permitivity = 8.85/10^12;
deltaT= 1/10^7;
 Stopper = .05/10^6;
 StickPoint = Beamheight - Stopper;
 Clear[S,t,x];
S [t_,x_]:=m*x''[t] +B*x'[t] + k*x[t];          (*define equation of motion*)
answer =DSolve[{S [t,x]== F,x[0]==0,x'[0]==0 }, x[t],t]; (*solve equation*)
Clear[soln,t];
soln[t_] = x[t]/.answer[[1]];

 L = BeamLength;
 W =BeamWidth;
 m = mass;
 IZ = MomentZ;
```

```
 Rigidity = RigidityFactor;
 k = springconstant;
 CC = CriticalFrequency;
 B = DampingCoefficient;
 Deflection = 0;
While
                [ Deflection < StickPoint,
                t=0;
                F = (MirrorArea*Permitivity*Voltage^2)/(2*(Beamheight)^2) ;
                n = 100;
                (*Calculate the pull-in voltage*)
                Do [ t= t+deltaT; Deflection = Re[soln[t]];
                        If[ Deflection >1.0/3.0*Beamheight, PullinVoltage = Voltage;
                                Deflection = StickPoint;
                                Break[];
                    ];
                F = (MirrorArea*Permitivity*Voltage^2)/(2*(Beamheight-Deflection)^2);
                ,
                {i,1,n}
                ];

        If[ Deflection < StickPoint,
                                (*if did not pull in then increase length and width*)
                                (*of mirror and recalculate parameters and start *)
                                (*over*)
                MirrorLength = MirrorLength + 1/10^6;
                MirrorWidth = MirrorLength;
                MirrorArea = Area;
                L = BeamLength;
                W =BeamWidth;
                 m = mass;
                IZ = MomentZ;
                 Rigidity = RigidityFactor;
                 k = springconstant;
                CC = CriticalFrequency;
                B = DampingCoefficient
           ]
]

Print[N[PullinVoltage]];

t=0;
Deflection = Re[soln[t]];
While[ Deflection < StickPoint,
        Deflection = Re[soln[t]];
        F =  (MirrorArea*Permitivity*Voltage^2)/(2*(Beamheight-Deflection)^2);
```

```
        t = t+deltaT;
        SuperStressVoltage = Voltage;
        Voltage = Voltage + 1;
       ]
```
(*The following commands give the dimensions of desired micro mirror*)
```
N[MirrorLength]
N[MirrorWidth]
N[L]
N[W]
SuperStressVoltage
```

Appendix III
Extending FEA
To
VHDL-AMS

The following code applies to chapter 7. this code will generate VHDL-AMS FEA models up to 10 elements from user specified input.

```
#include<iostream.h>
#include<fstream.h>
#include<math.h>
#include<string.h>
#include<iomanip.h>

const int MAX=10;
const int STRSIZE=20;

struct BeamType
{
        char Length[MAX];
        char Width[MAX];
        char Height[MAX];
        int number_elements;
        char applied_force[MAX];

};
struct StringType
{
        char S[STRSIZE];
};

struct StiffnessType
{
        StringType K[2*MAX][2*MAX];
        int number_elements;
};

BeamType Get_Beam_Info();
void Write_Beam_Info(BeamType B);
void Write_Entity(ofstream& Out);
void Write_Architecture(ofstream &Out,BeamType B);
StiffnessType Create_Stiffness_Matrix(StiffnessType& k,BeamType B);
```

```
void Init_K(StiffnessType &K1);
void Init_Stiffness(StiffnessType& K);
void Write_Constants(ofstream& Out,BeamType B);
void Write_Quantities(ofstream& Out,int number_elements);
void Write_Simultaneous_Equations(ofstream& Out,StiffnessType K);
void Offset(StiffnessType &K2,int Start_Pos);
void Concat_K1_To_K(StiffnessType K1,StiffnessType&K);

//*********************************************************************

void main()
{

        ofstream Out;
        Out.open("FeaBeam.vhd");

        BeamType BeamValues;

   BeamValues = Get_Beam_Info();
        Write_Beam_Info(BeamValues);
        Write_Entity(Out);
        Write_Architecture(Out,BeamValues);

  Out.close();
}
//*********************************************************************

BeamType Get_Beam_Info()
{
//Gets beam info from user.
        BeamType BeamValues;
        cout<<"Enter Beam Length: ";
        cin>>BeamValues.Length;
        cout<<"Enter Beam Width: ";
        cin>>BeamValues.Width;
        cout<<"Enter Beam Height: ";
        cin>>BeamValues.Height;
        cout<<"Enter number of finite elements to break beam into (1-10): ";
        cin>>BeamValues.number_elements;
        cout<<"Enter applied force: ";
        cin>>BeamValues.applied_force;
        return BeamValues;
}
//*********************************************************************
```

```
void Write_Beam_Info(BeamType B)
{
//displays info stored in beam.
        cout<<"Length : "<<B.Length<<endl;
        cout<<"Width  : "<<B.Width<<endl;
        cout<<"Height : "<<B.Height<<endl;
        cout<<"elmts  : "<<B.number_elements<<endl;
        cout<<"Force  : "<<B.applied_force<<endl;
}
//*********************************************************************

void Write_Entity(ofstream &Out)
{
//Writes the entity section of VHDL-AMS model to output file.
        Out<<"entity FEABEAM is"<<endl<<"end entity FEABEAM;"<<endl<<endl;

}
//*********************************************************************


void Write_Stiffness_Matrix(StiffnessType k)
{
//Generates the stiffness matrix for a given k.
        for(int i=0;i<=2*k.number_elements+1;i++)
        {
                for(int j=0;j<=2*k.number_elements+1;j++)
                {
                        cout<<setw(9)<<k.K[i][j].S;
                }
                cout<<endl;
        }
}
//*********************************************************************

void Offset(StiffnessType &K,int Start_Pos)
{

                strcpy(K.K[Start_Pos][Start_Pos].S,"24.0");
                strcpy(K.K[Start_Pos][Start_Pos+1].S,"0.0");
                strcpy(K.K[Start_Pos+1][Start_Pos].S,"0.0");
                strcpy(K.K[Start_Pos+1][Start_Pos+1].S,"8.0*L2");

}
//*********************************************************************
**********
```

```
void Concat_K1_To_K(StiffnessType K1,StiffnessType&K)
{
//Create stiffness matrix affected by a given stiffness matrix.
	int start=2;
	int m=0;
	int n=0;

	for(int i=1;i<K.number_elements;i++)
	{   m=0;
		for(int j=start;j<start+4;j++)
		{
			n=0;
			for(int k=start;k<start+4;k++)
			{
				strcpy(K.K[j][k].S,K1.K[m][n].S);
				n++;
			}
			m++;
		}
		start+=2;
	}
	for(i=1;i<K.number_elements;i++)
	{
		Offset(K,2*i);
	}
		Write_Stiffness_Matrix(K);
}
//*************************************************************************

StiffnessType Create_Stiffness_Matrix(StiffnessType& K1,BeamType B)
{
//Generate the stiffness matrix for all sub stiffness matrices.
	StiffnessType K;
	K.number_elements=K1.number_elements=B.number_elements;
	Init_Stiffness(K1);
	Init_Stiffness(K);
	Init_K(K1);
	Init_K(K);
	Concat_K1_To_K(K1,K);


	return K;
}

//*************************************************************************
```

```
void Init_Stiffness(StiffnessType& K)
{
//initializes stiffness matrix to all 0's.
        for (int i=0;i<=2*K.number_elements+1;i++)
        {
                for(int j=0;j<=2*K.number_elements+1;j++)
                {
                        strcpy(K.K[i][j].S,"0.0");
                }
        }
}
//*************************************************************************

void Init_K(StiffnessType &K1)
{
//Generate initial values for stiffness matrix for a beam.
        strcpy(K1.K[0][0].S,"12.0");
        strcpy(K1.K[0][1].S,"6.0*L");
        strcpy(K1.K[0][2].S,"-12.0");
        strcpy(K1.K[0][3].S,"6.0*L");
        strcpy(K1.K[1][0].S,"6.0*L");
        strcpy(K1.K[1][1].S,"4.0*L2");
        strcpy(K1.K[1][2].S,"-6.0*L");
        strcpy(K1.K[1][3].S,"2.0*L2");
        strcpy(K1.K[2][0].S,"-12.0");
        strcpy(K1.K[2][1].S,"-6.0*L");
        strcpy(K1.K[2][2].S,"12.0");
        strcpy(K1.K[2][3].S,"-6.0*L");

        strcpy(K1.K[3][0].S,"6.0*L");
        strcpy(K1.K[3][1].S,"2.0*L2");
        strcpy(K1.K[3][2].S,"-6.0*L");
        strcpy(K1.K[3][3].S,"4.0*L2");
}
//*************************************************************************

void Write_Constants(ofstream& Out,BeamType B)
{

        Out<<"\tconstant L:real :="<<B.Length<<";"<<endl;
        Out<<"\tconstant W:real :="<<B.Width<<";"<<endl;
        Out<<"\tconstant H:real :="<<B.Height<<";"<<endl;
        Out<<"\tconstant EZ:real:= 170.0e9;"<<endl;
        Out<<"\tconstant IZ:real:= (W*H*H*H)/12.0;"<<endl;
        Out<<"\tconstant EI:real:= EZ*IZ;"<<endl;
        Out<<"\tconstant L3:real:=L*L*L;"<<endl;
```

```
	Out<<"\tconstant L2:real:=L*L;"<<endl;
	Out<<"\tconstant Fn"<<B.number_elements<<":real:=
"<<B.applied_force<<";"<<endl;
	Out<<"\tconstant Fn0:real:= -Fn"<<B.number_elements<<";"<<endl;
	Out<<"\tconstant Vn0:real:= 0.0;"<<endl;
}

void Write_Constants(ofstream& Out,BeamType B)
{

	Out<<"\tconstant L:real :=
	("<<B.Length<<")/"<<B.number_elements<<".0;"<<endl;
	Out<<"\tconstant W:real :="<<B.Width<<";"<<endl;
	Out<<"\tconstant H:real :="<<B.Height<<";"<<endl;
	Out<<"\tconstant EZ:real:= 170.0e9;"<<endl;
	Out<<"\tconstant IZ:real:= (W*H*H*H)/12.0;"<<endl;
	Out<<"\tconstant EI:real:= EZ*IZ;"<<endl;
	Out<<"\tconstant L3:real:=L*L*L;"<<endl;
	Out<<"\tconstant L2:real:=L*L;"<<endl;
	Out<<"\tconstant K:real := (3.0*EI)/L3;"<<endl;
	Out<<"\tconstant F"<<B.number_elements<<":real:=
"<<B.applied_force<<";"<<endl;
	Out<<"\tconstant V0:real:= 0.0;"<<endl;
}
//*********************************************************************

void Write_Quantities(ofstream& Out,int number_elements)
{
	for (int i=0;i<=2*number_elements+1;i++)
	{
		Out<<"\tquantity F"<<i<<": real;"<<endl;
	}
	cout<<endl;
	for (i=0;i<=number_elements;i++)
	{
		Out<<"\tquantity V"<<i<<":real;"<<endl;
		Out<<"\tquantity Theta"<<i<<":real;"<<endl;
	}
}

void Write_Quantities(ofstream& Out,int number_elements)
{
	cout<<endl;
	for (int i=1;i<=number_elements;i++)
	{
		Out<<"\tquantity V"<<i<<":real;"<<endl;
```

```
	}
}
//*********************************************************************

void Write_Simultaneous_Equations(ofstream& Out,StiffnessType K)
{

	for (int i=0;i<=2*K.number_elements+1;i++)
	{
		Out<<"F"<<i<<" == (EI/L3)*(";
		for(int j=0;j<=2*K.number_elements+1;j++)
		{
			if( strcmp(K.K[i][j].S,"0.0")!=0)
			{
				Out<<"("<<K.K[i][j].S<<"*";
				if (j%2 == 0)
				{
					Out<<"V"<<j/2;
				}
				else
				{
					Out<<"Theta"<<j/2;
				}
				Out<<")";

				if (j<2*K.number_elements+1 &&
strcmp(K.K[i][j+1].S,"0.0")!=0)
					Out<<" + ";
			}

		}
		Out<<");"<<endl;
	}
	Out<<"V0 == Vn0;"<<endl;
	Out<<"F0 == Fn0;"<<endl;
	Out<<"F"<<K.number_elements<<"== Fn"<<K.number_elements<<";"<<endl;
}

void Write_Simultaneous_Equations(ofstream& Out,StiffnessType K)
{
	if(K.number_elements==1)
	{
		Out<<"V1 == F1/K"<<endl;
	}
	else
```

```
        {
                Out<<"V1 == V2/2.0;"<<endl;
                for (int i=1;i<K.number_elements-1;i++)
                {
                        Out<<"V"<<i<<" == 2.0*V"<<i+1<<" - V"<<i+2<<";"<<endl;
                }
                Out<<"V"<<K.number_elements-1<< " ==
(K*V"<<K.number_elements<<" - F"
                        <<K.number_elements<<")/K;"<<endl;
        }
}
//*********************************************************************

void Write_Architecture(ofstream &Out,BeamType B)
{
        StiffnessType K1;
        StiffnessType K;

        K=Create_Stiffness_Matrix(K1,B);

        Out<<"architecture behavior of FEABEAM is"<<endl;
        Write_Constants(Out,B);
        Write_Quantities(Out,K.number_elements);

        Out<<"begin"<<endl;

        Write_Simultaneous_Equations(Out,K);

        Out<<"end behavior;"<<endl;


}
//**********************************The end of file
```

The following is an example output file generated by this program for the given inputs:
Length = 80
Width = 20
Height = 2
And the number of elements = 5

```
entity FEABEAM is
end entity FEABEAM;

architecture behavior of FEABEAM is
        constant L:real := (80)/5.0;
        constant W:real :=20;
        constant H:real :=2;
        constant EZ:real:= 170.0e9;
        constant IZ:real:= (W*H*H*H)/12.0;
        constant EI:real:= EZ*IZ;
        constant L3:real:=L*L*L;
        constant L2:real:=L*L;
        constant K:real := (3.0*EI)/L3;
        constant F5:real:= 10;
        constant V0:real:= 0.0;
        quantity V1:real;
        quantity V2:real;
        quantity V3:real;
        quantity V4:real;
        quantity V5:real;
begin
V1 == V2/2.0;
V1 == 2.0*V2 - V3;
V2 == 2.0*V3 - V4;
V3 == 2.0*V4 - V5;
V4 == (K*V5 - F5)/K;
end behavior;
```

Appendix IV
Results for extracting cantilever beams from CIF files

The following results correspond to chapter 8. The program in Appendix I was used on the following CIF input files and generated the following results. A CIF file corresponding to each case of cantilever beam is given.

CIF FILE	Output generated
( ) ( ) DS 1 100 2; 9 Cell0; L CSN; B 50 50 25,25; L CPZ; B 50 50 25,25; B 50 50 25,25; L COF; B 36 46 30,25; B 36 46 30,25; L CPS; B 48 48 25 25; B 48 48 25 25; DF; C 1; E	Case Number: 1 Anchor: Length: 36 Width: 46 Center: (30,25) Direction: (1,0) Angle: 0 Beam: Length: 48 Width: 48 Center: (25,25) Direction: (1,0) Angle: 0 Length: 11 Width: 46 Center: (25,25) Direction: (1,0) Angle: 0

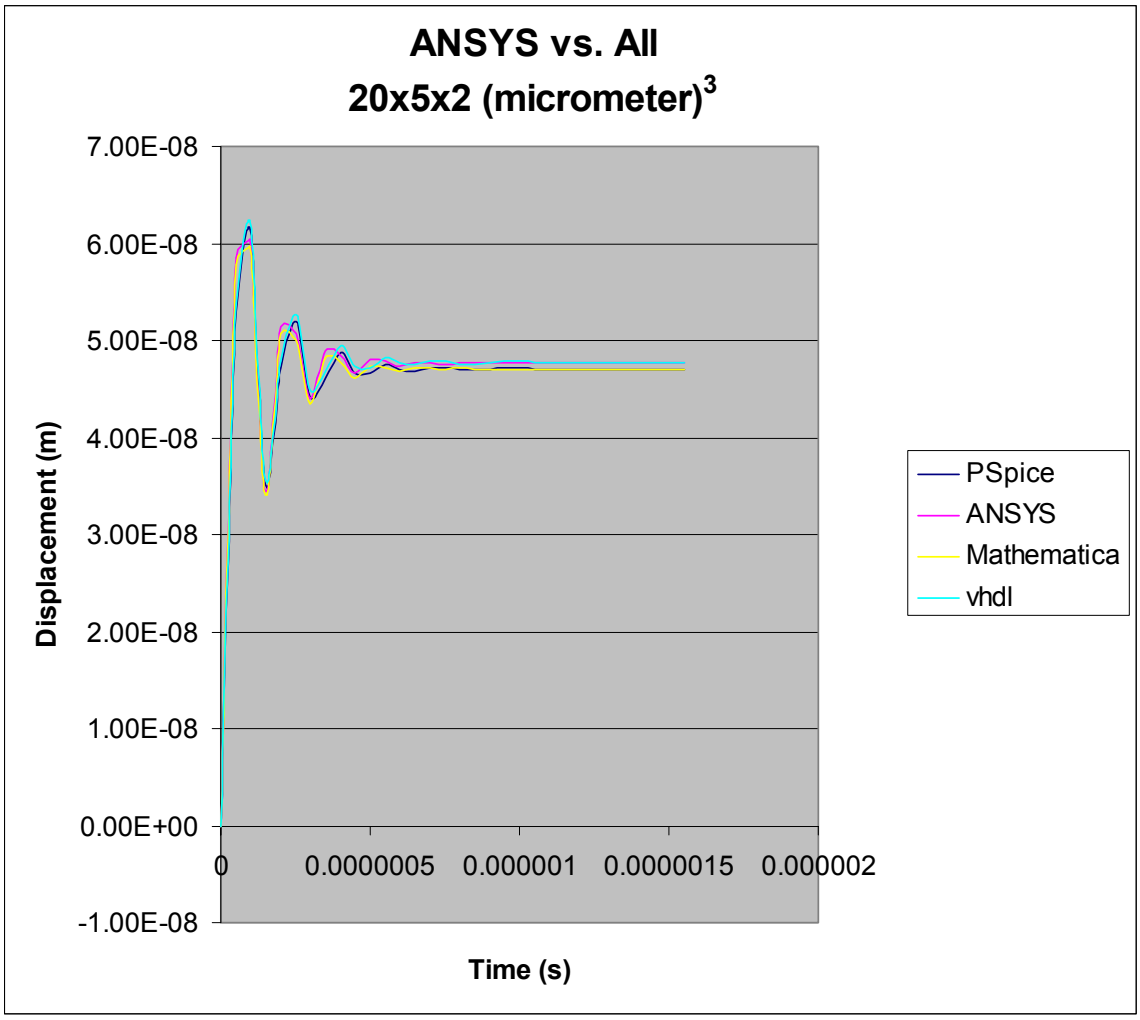
CIF FILE	Output generated
( ) ( ) DS 1 100 2; 9 Cell0; L CSN; B 50 50 25,25; L CPZ; B 50 50 25,25; L COL; B 36 46 30,25; L CPT; B 48 48 25,25; DF; C 1; E	Case Number: 2 Anchor: Length: 36 Width: 46 Center: (30,25) Direction: (1,0) Angle: 0 Beam: Length: 48 Width: 48 Center: (25,25) Direction: (1,0) Angle: 0 Length: 11 Width: 46 Center: (25,25) Direction: (1,0) Angle: 0

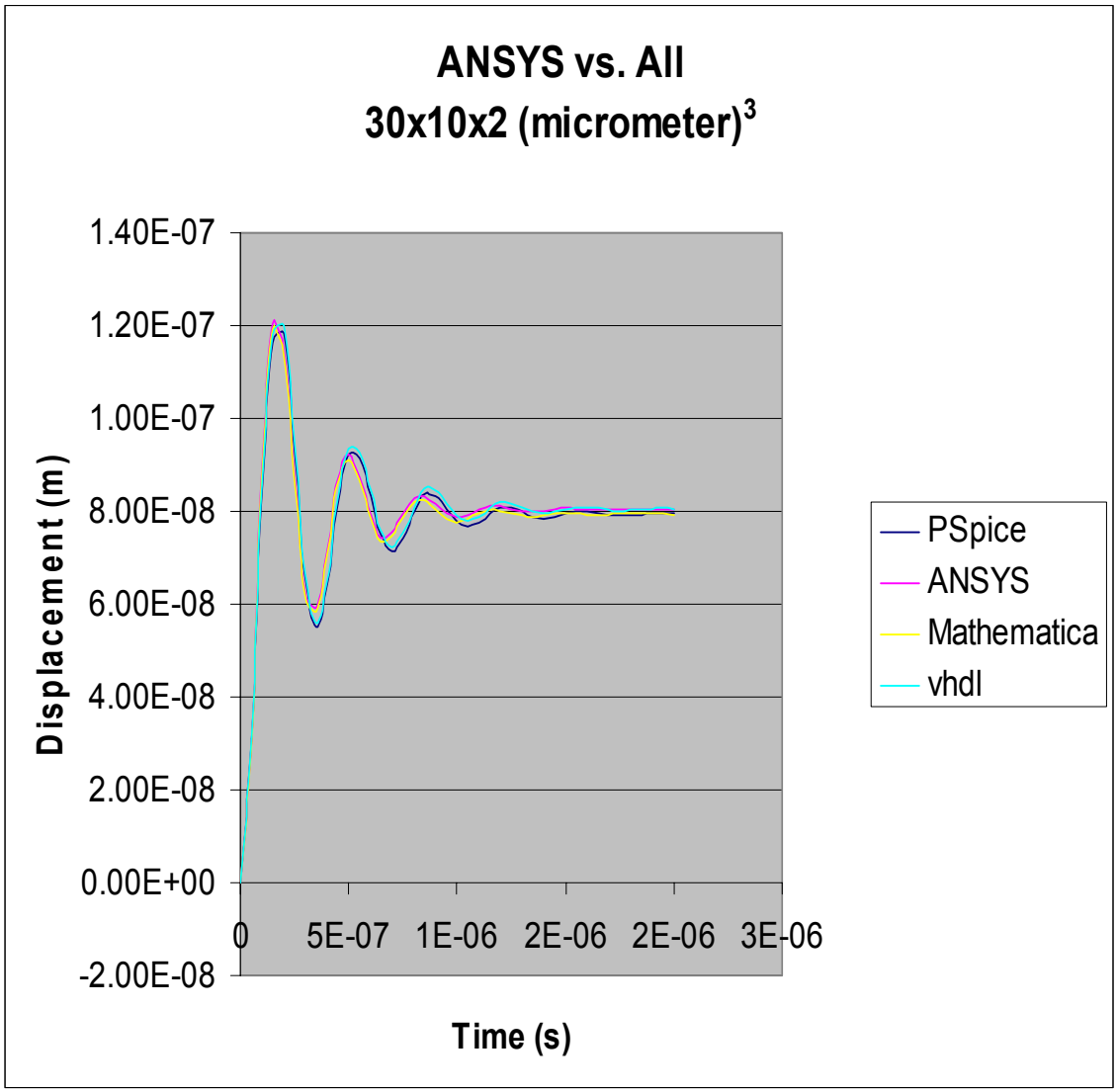
CIF FILE	Output generated
( ) ( ) DS 1 100 2; 9 Cell0; L CSN; B 50 50 25,25; L CPS; B 50 50 0,0; L COT; B 8 38 -5,0; L CPT; B 20 40 0,0; DF; C 1; E	Case Number: 3 Anchor: Length: 8 Width: 38 Center: (5,0) Direction: (1,0) Angle: 0 Beam: Length: 20 Width: 40 Center: (0,0) Direction: (1,0) Angle: 0 Length: 11 Width: 38 Center: (0,0) Direction: (1,0) Angle: 0

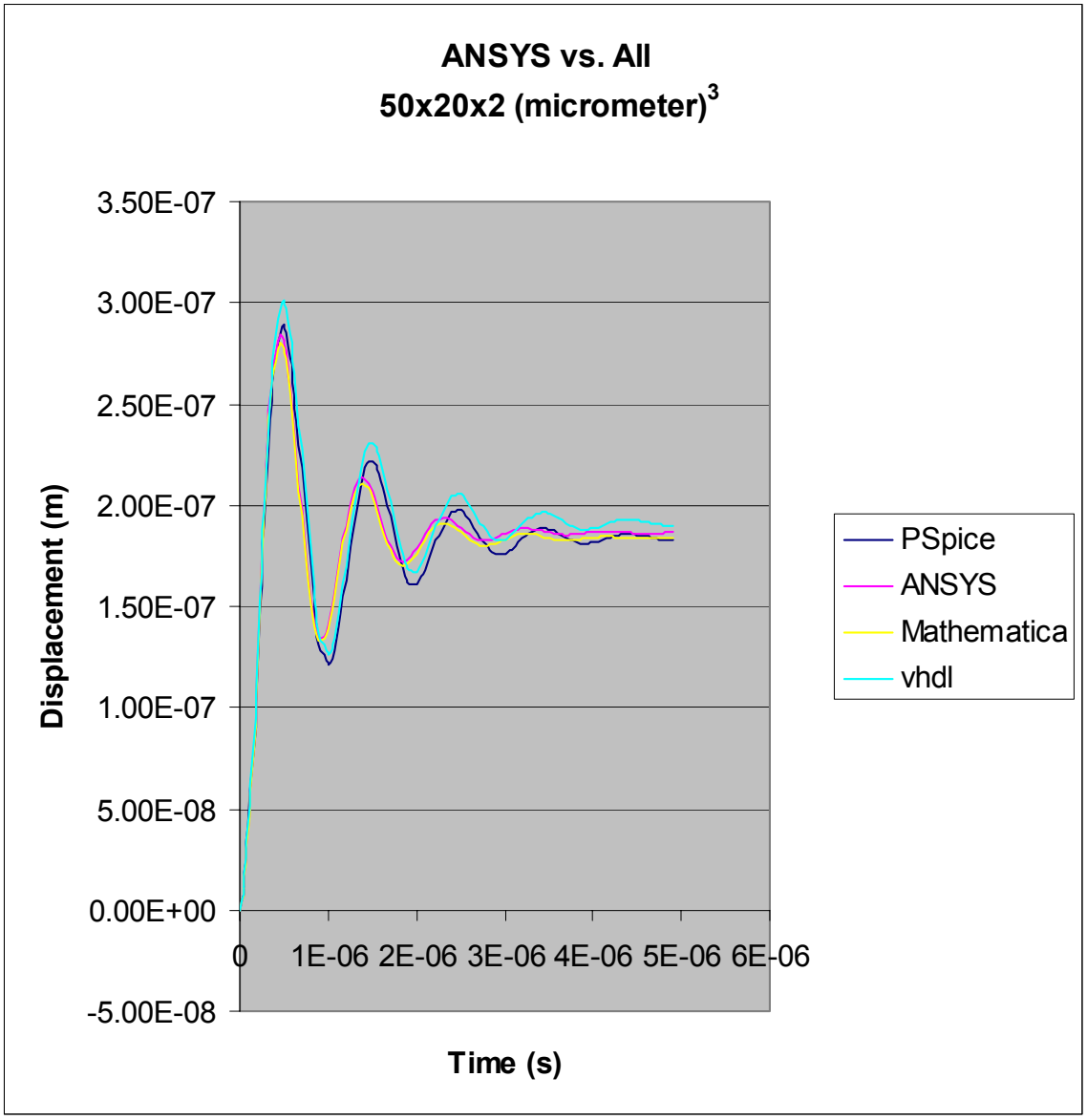
CIF FILE	Output generated
( ) ( ) DS 1 100 2; 9 Cell0; L CSN; B 50 50 25,25; L CPS; B 50 50 0,0; L COL; B 8 38 -5,0; L CPT; B 20 40 0,0; DF; C 1; E	Case Number: 4 Anchor: Length: 8 Width: 38 Center: (5,0) Direction: (1,0) Angle: 0 Beam: Length: 20 Width: 40 Center: (0,0) Direction: (1,0) Angle: 0 Length: 11 Width: 38 Center: (0,0) Direction: (1,0) Angle: 0

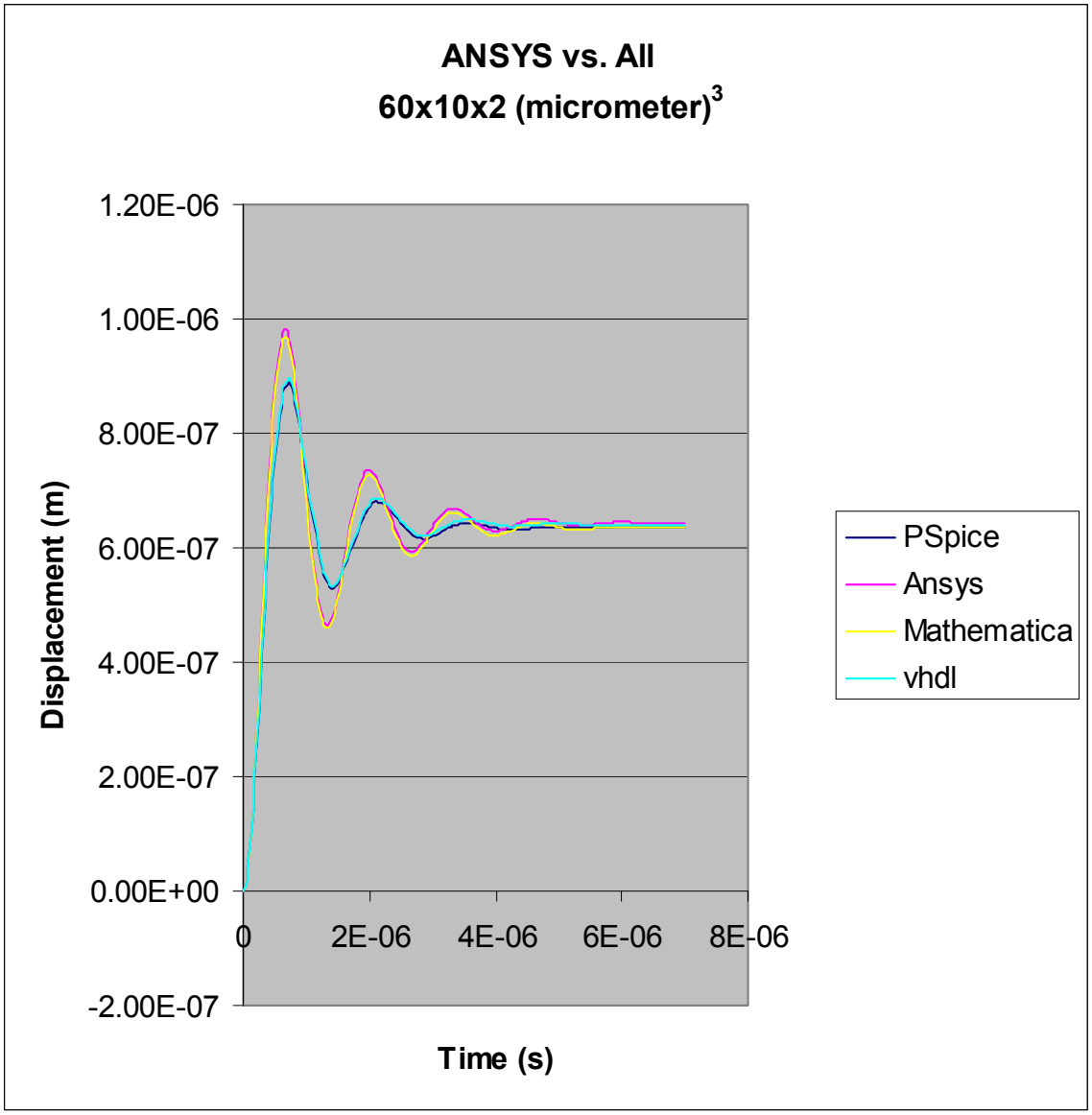
Appendix V
Results for simulations on different beams

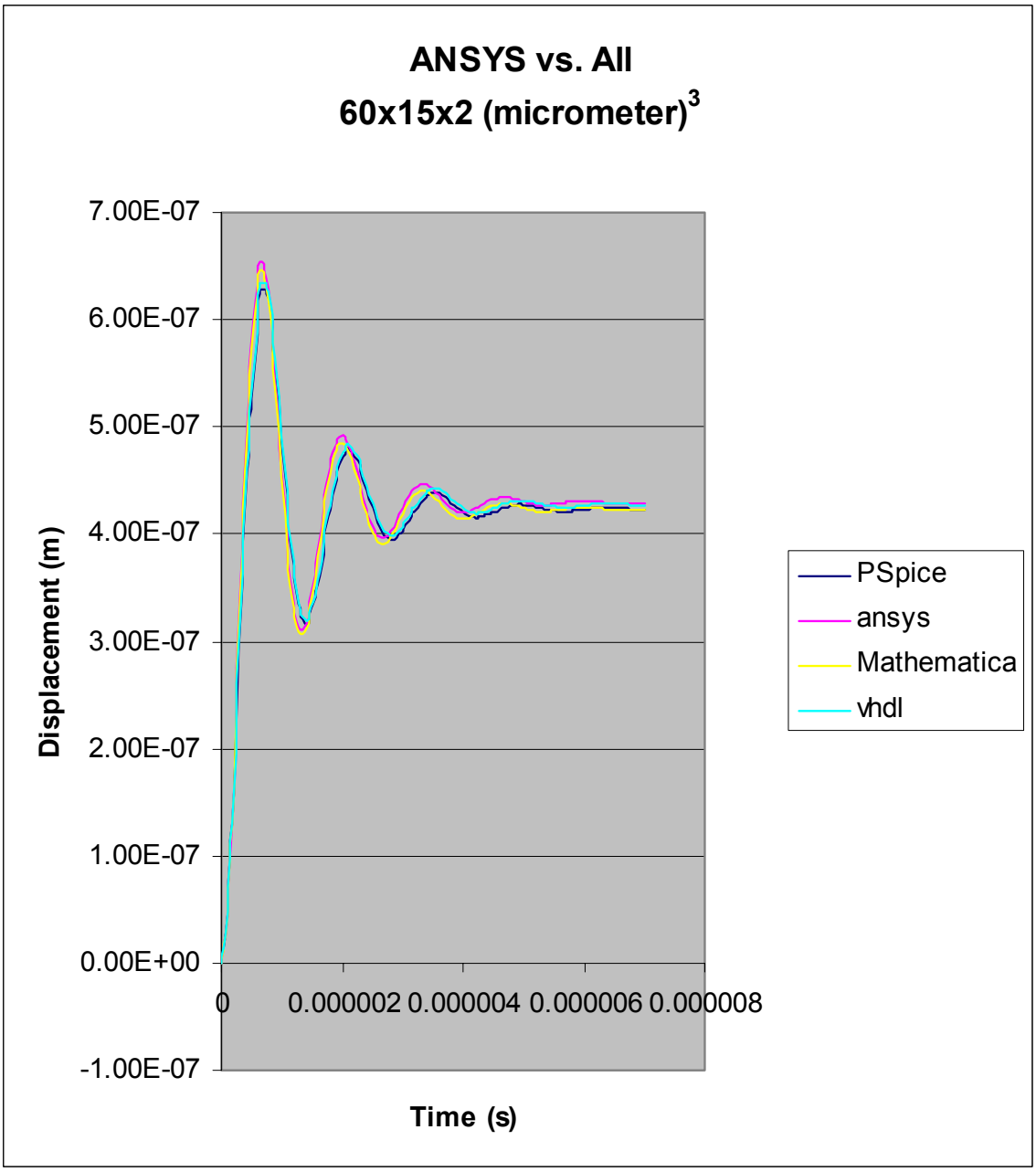
The following graphs correspond with chapter 8. All graphs demonstrate the dampened oscillation of a cantilever beam with a constant 1E-5 N force applied. Complete data files available at http://www.ececs.uc.edu/~cpurdy.

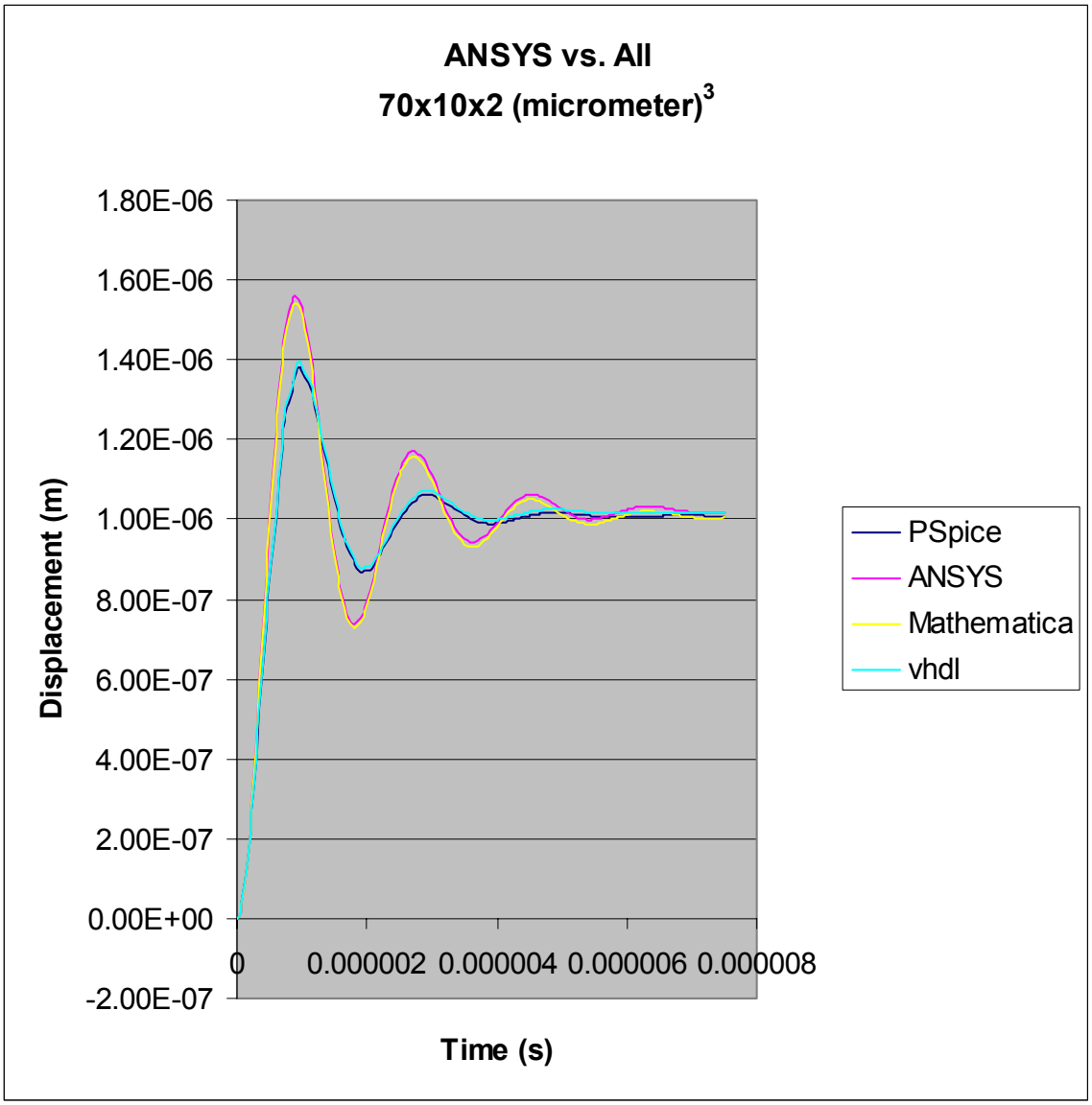


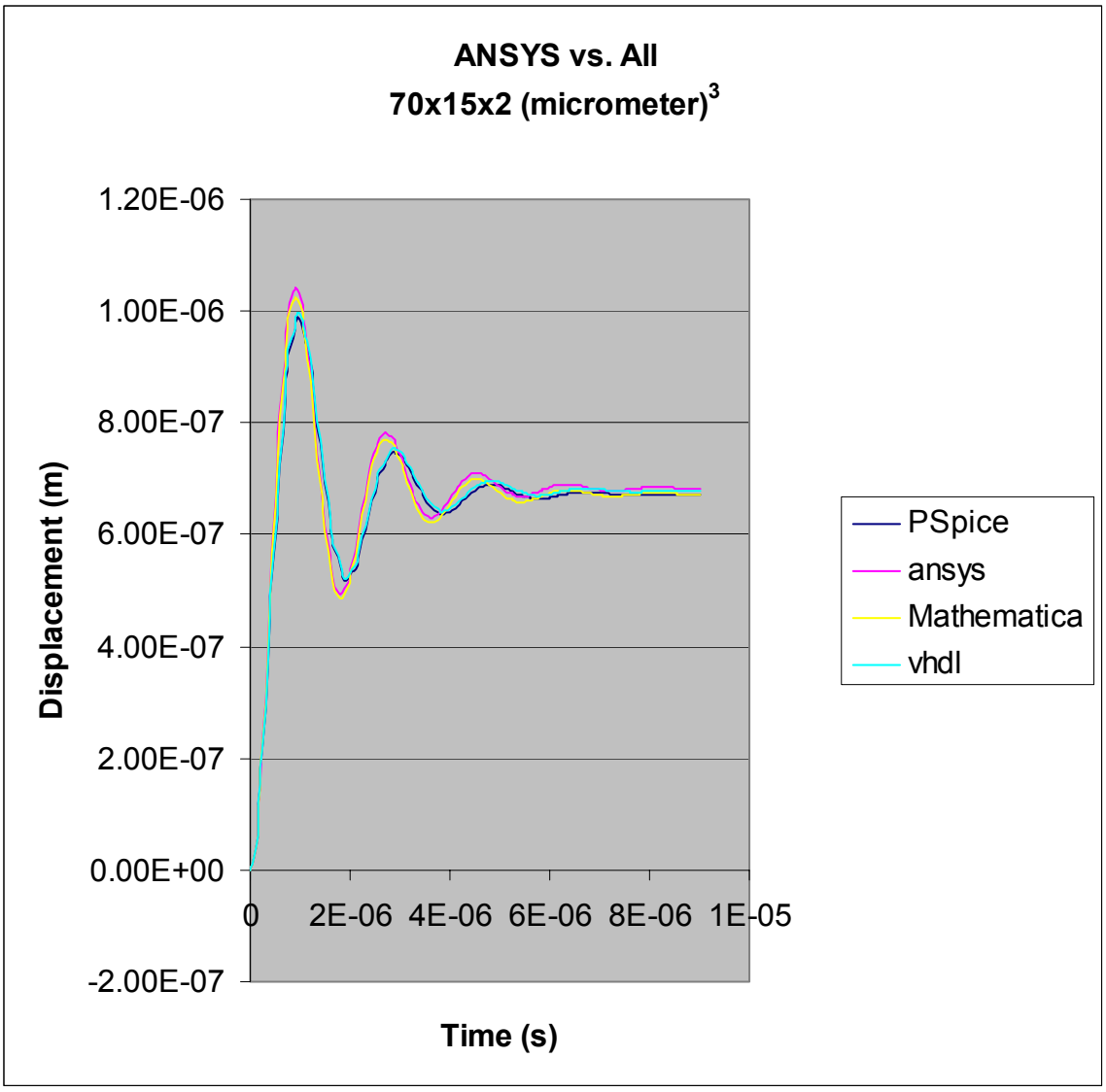

ANSYS vs. All
30x10x2 (micrometer)3
1.40E-07
1.20E-07
1.00E-07
8.00E-08
6.00E-08
4.00E-08
2.00E-08
0.00E+00
-2.00E-08
Displacement (m)
0
5E-07
1E-06
2E-06
2E-06
3E-06
Time (s)
PSpice
ANSYS
Mathematica
vhdl

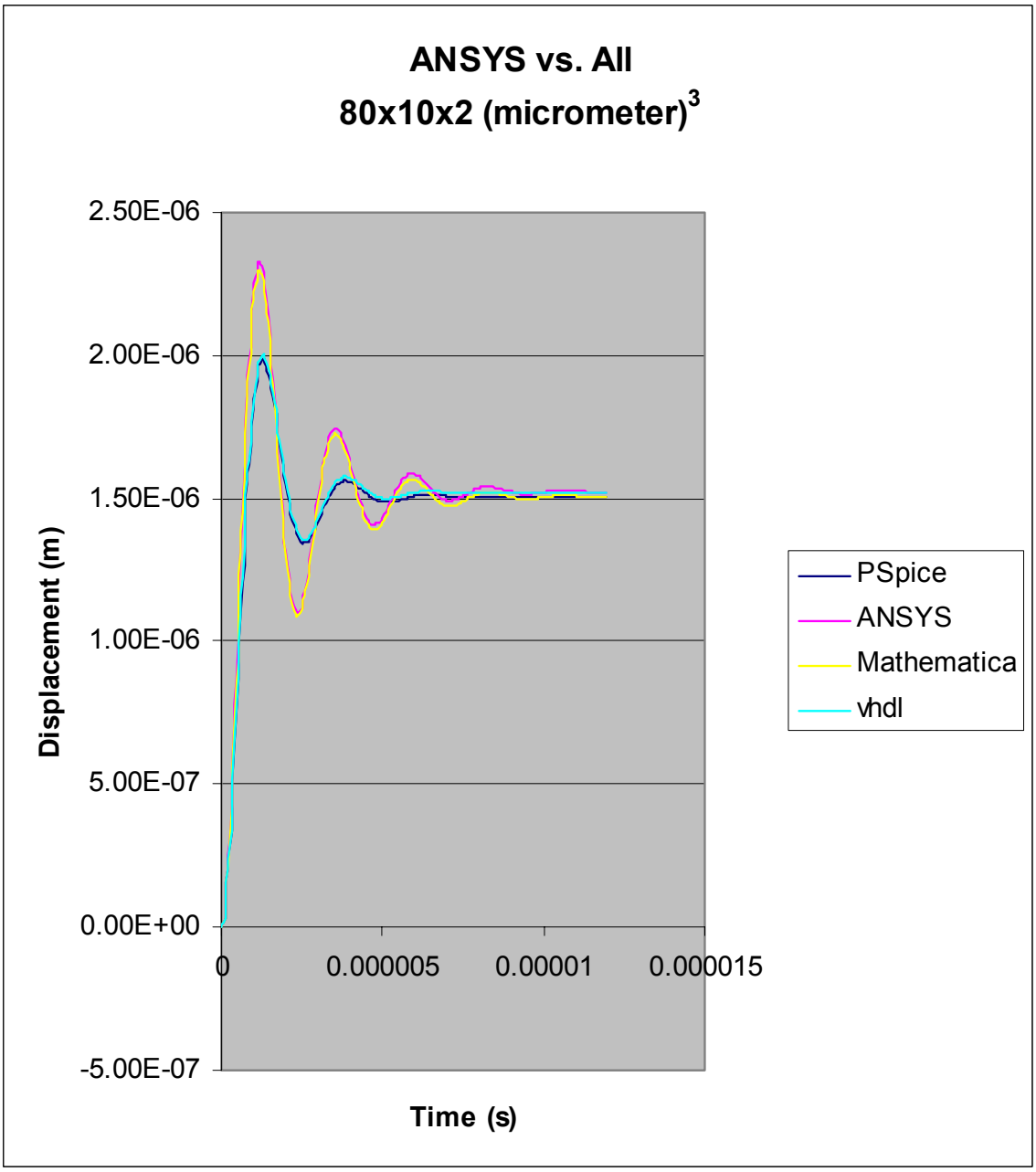

ANSYS vs. All
50x20x2 (micrometer)3
3.50E-07
3.00E-07
2.50E-07
2.00E-07
1.50E-07
1.00E-07
5.00E-08
0.00E+00
-5.00E-08
Displacement (m)
0
1E-06
2E-06
3E-06
4E-06
5E-06
6E-06
Time (s)
PSpice
ANSYS
Mathematica
vhdl

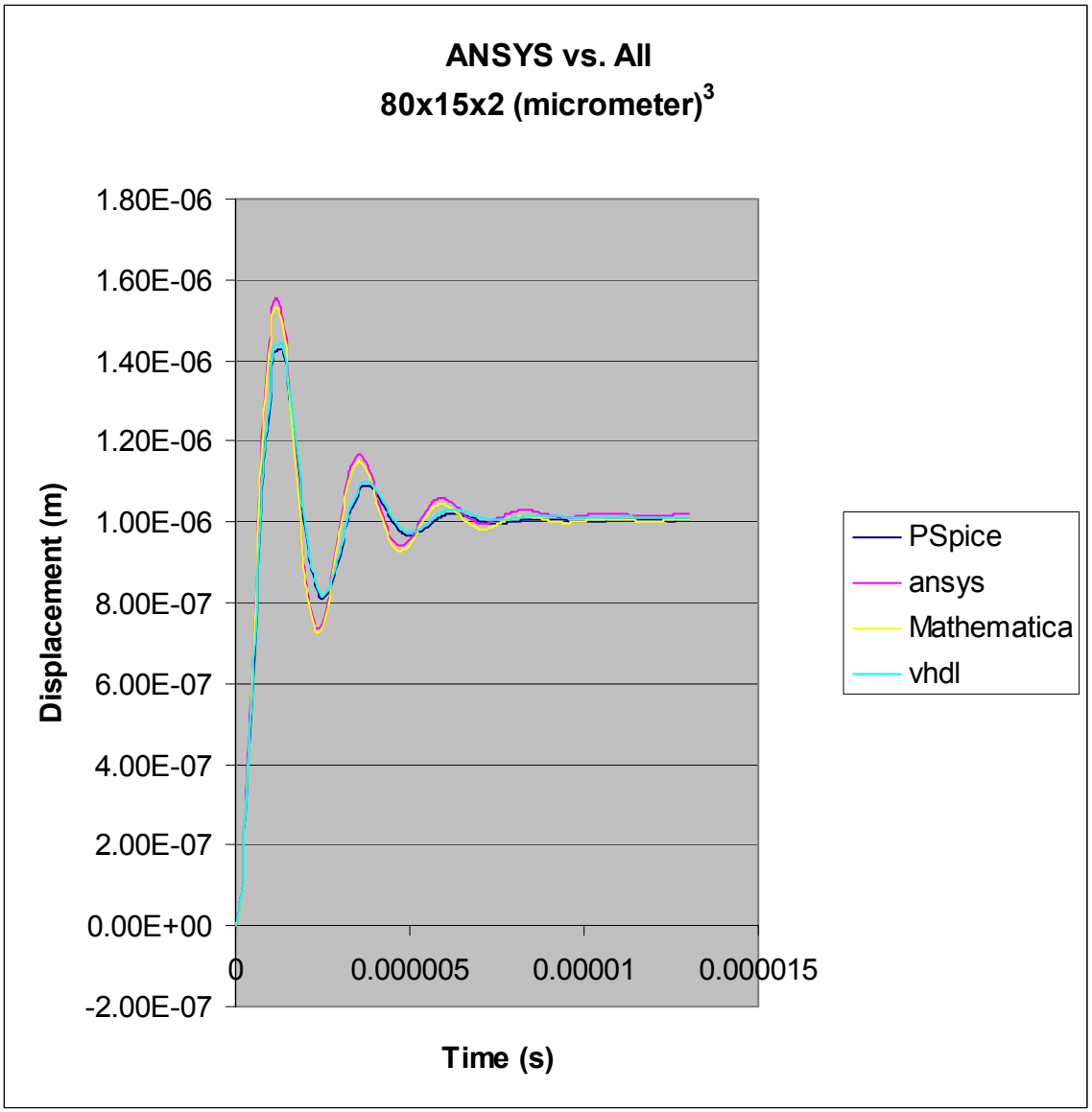

ANSYS vs. All
60x10x2 (micrometer)3
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
2E-06
4E-06
6E-06
8E-06
Time (s)
PSpice
Ansys
Mathematica
vhdl

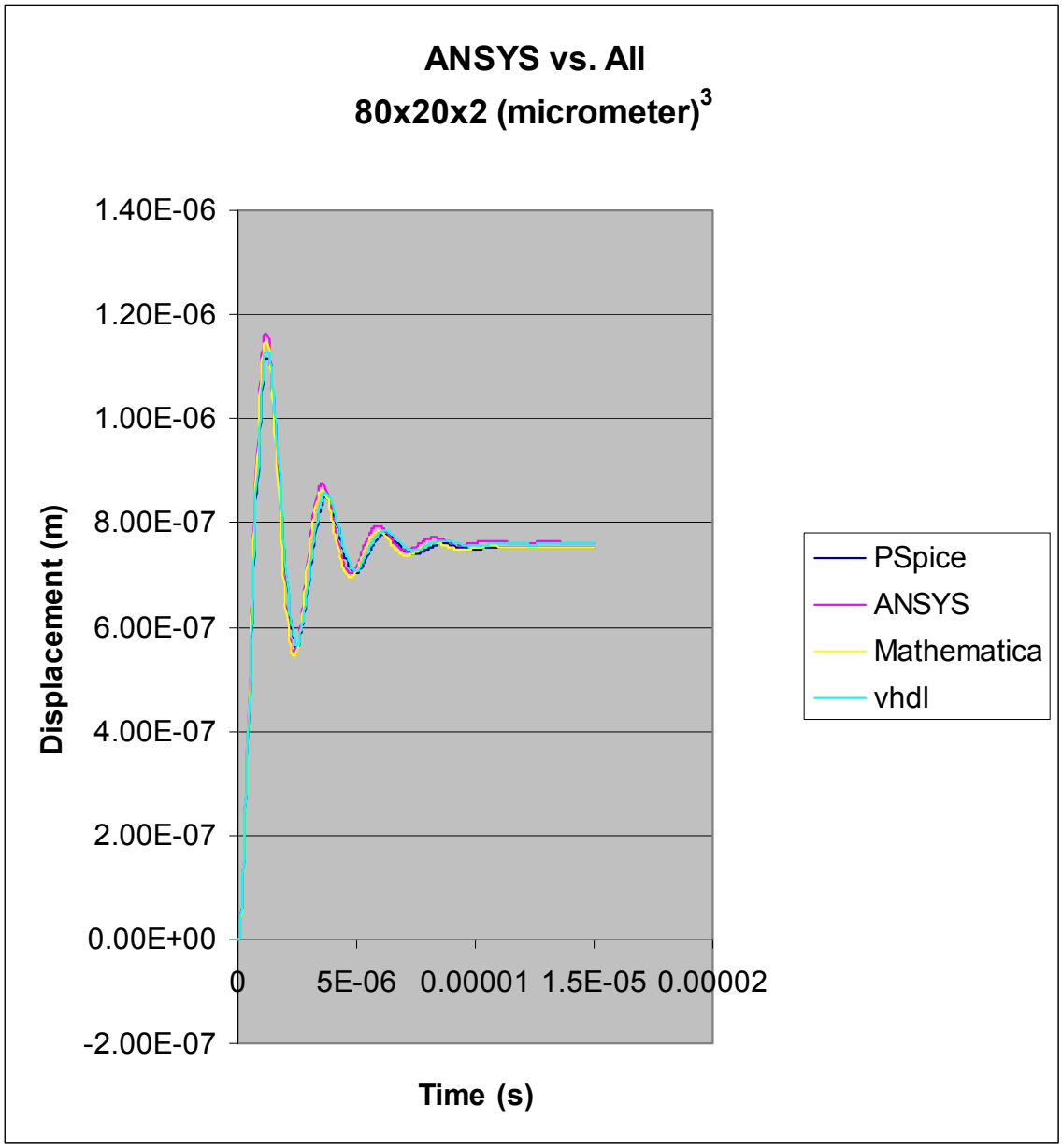

ANSYS vs. All
60x15x2 (micrometer)3
7.00E-07
6.00E-07
5.00E-07
4.00E-07
3.00E-07
2.00E-07
1.00E-07
0.00E+00
-1.00E-07
Displacement (m)
0
0.000002
0.000004
0.000006
0.000008
Time (s)
PSpice
ansys
Mathematica
vhdl

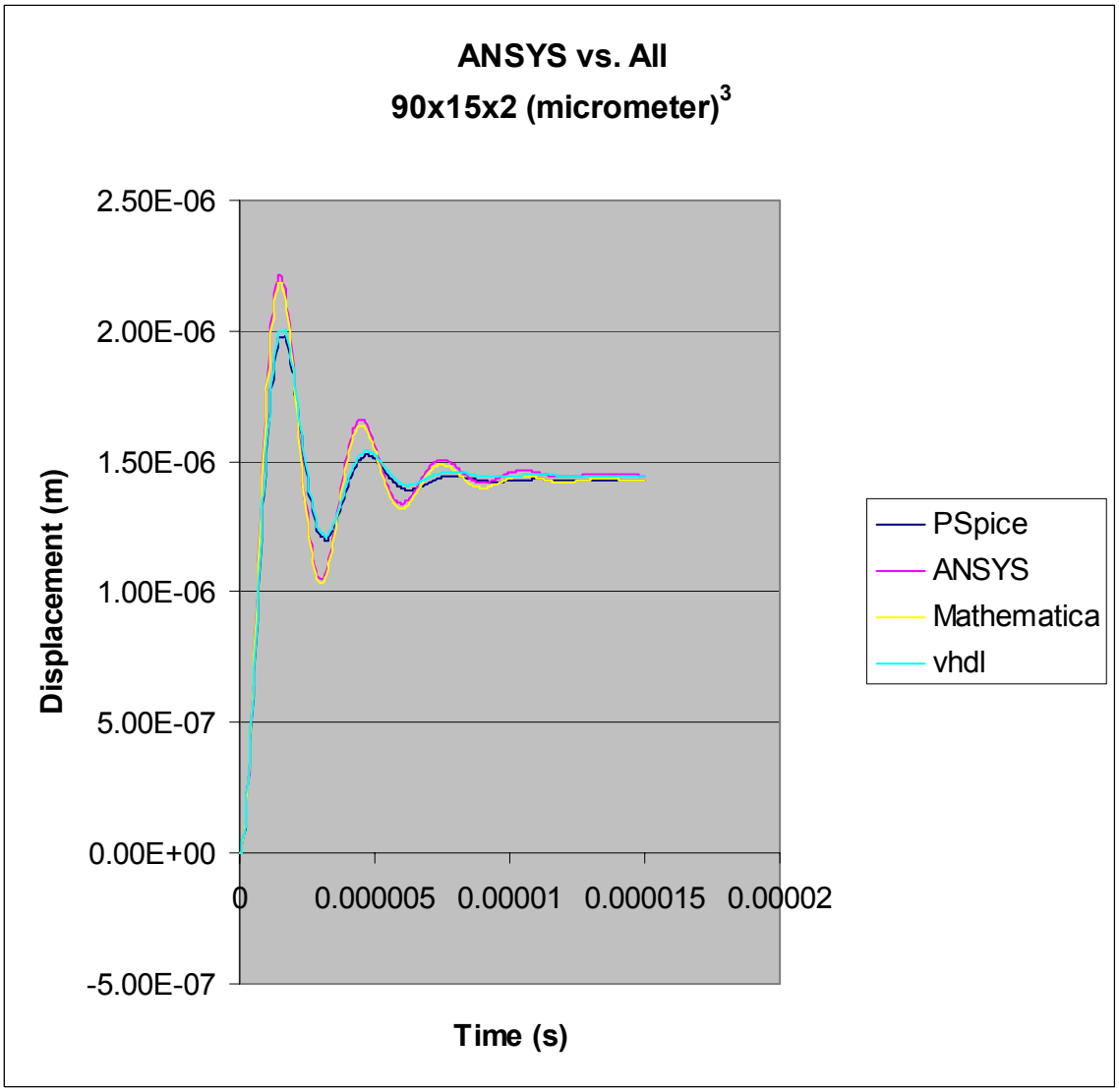

ANSYS vs. All
70x10x2 (micrometer)3
1.80E-06
1.60E-06
1.40E-06
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
0.000002
0.000004
0.000006
0.000008
Time (s)
PSpice
ANSYS
Mathematica
vhdl

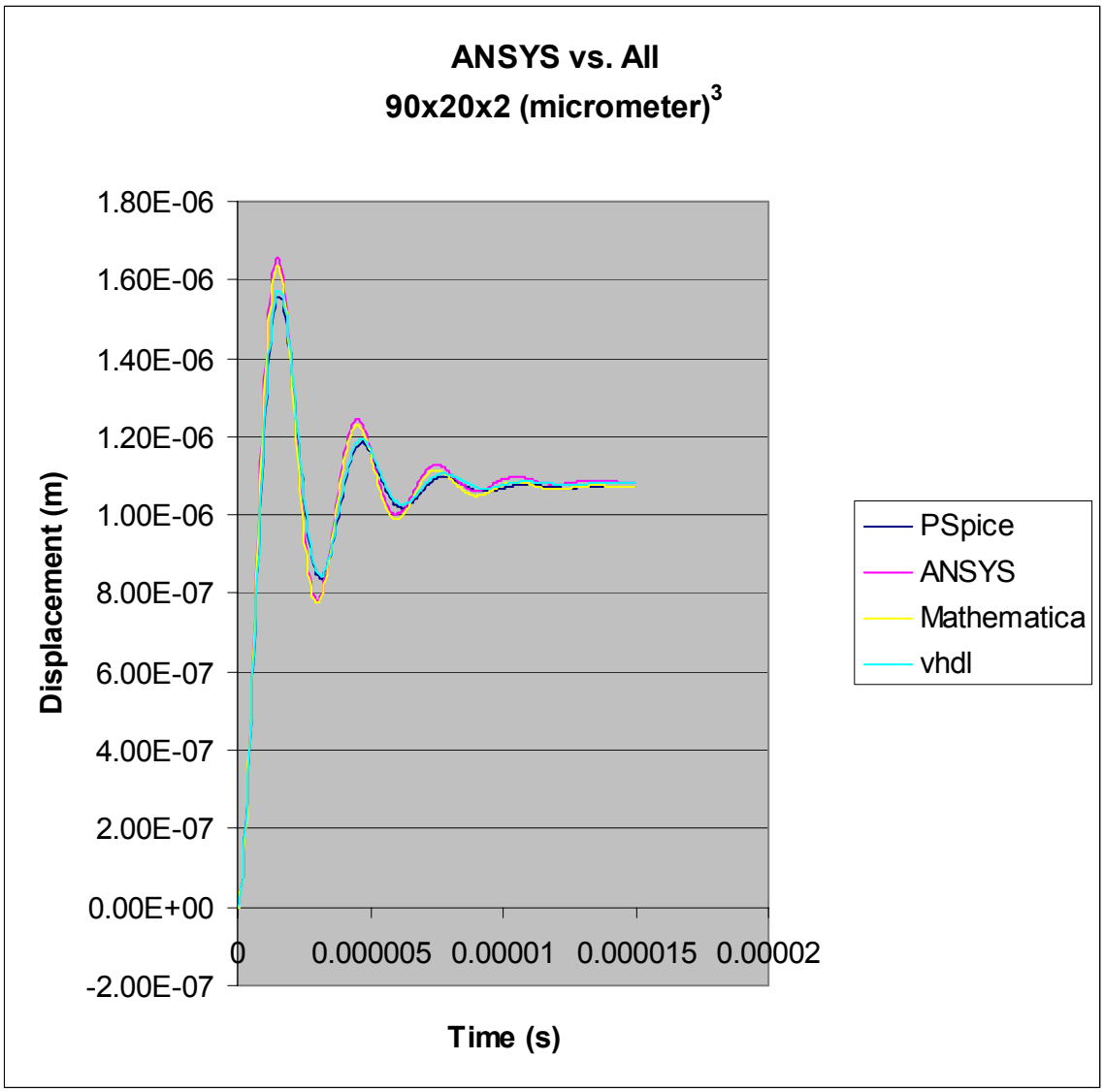

ANSYS vs. All
70x15x2 (micrometer)3
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
2E-06
4E-06
6E-06
8E-06
1E-05
Time (s)
PSpice
ansys
Mathematica
vhdl

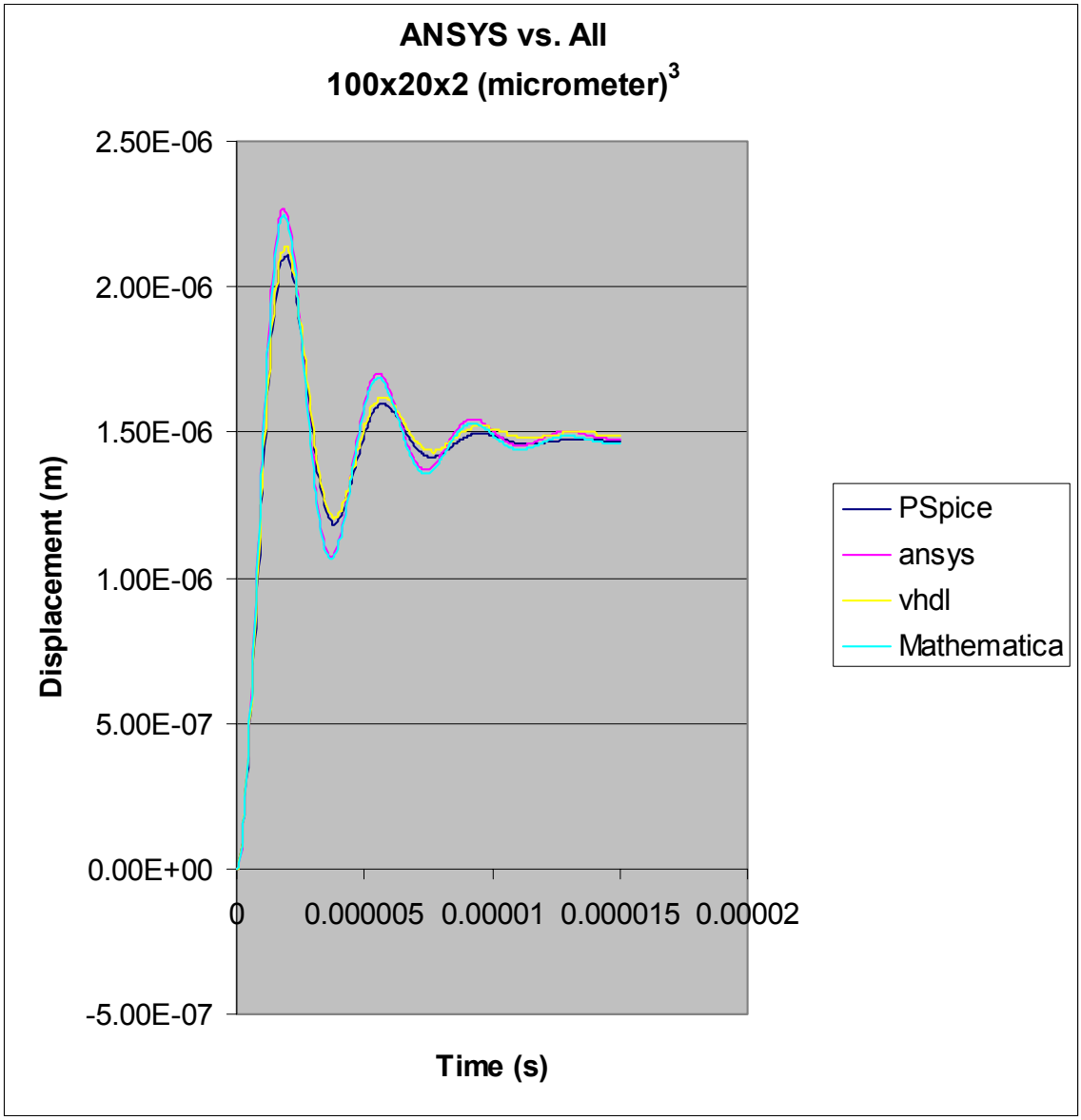

ANSYS vs. All
80x10x2 (micrometer)3
2.50E-06
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
-5.00E-07
Displacement (m)
0
0.000005
0.00001
0.000015
Time (s)
PSpice
ANSYS
Mathematica
vhdl

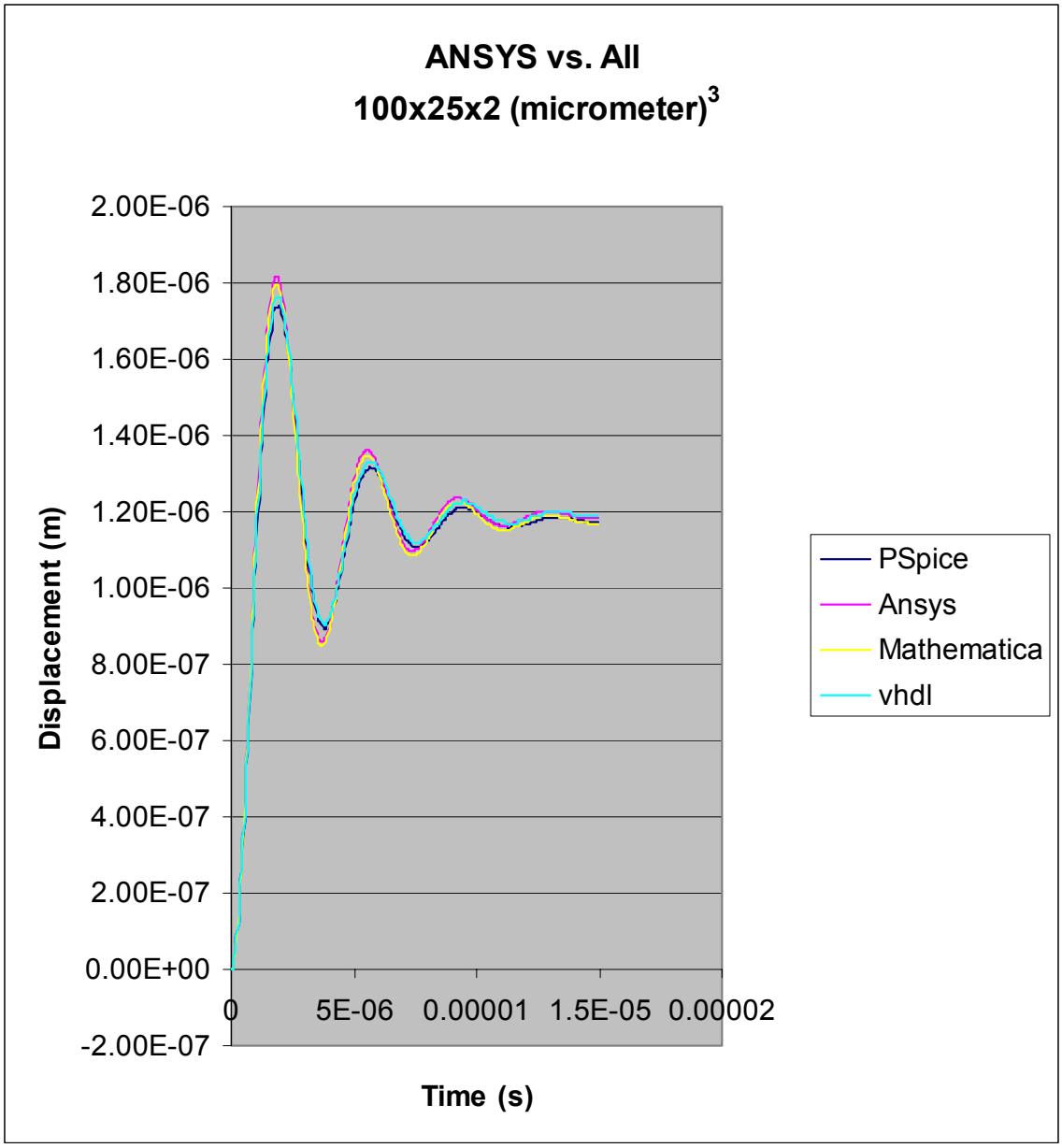

ANSYS vs. All
80x15x2 (micrometer)3
1.80E-06
1.60E-06
1.40E-06
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
0.000005
0.00001
0.000015
Time (s)
PSpice
ansys
Mathematica
vhdl


ANSYS vs. All
80x20x2 (micrometer)3
1.40E-06
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
5E-06
0.00001
1.5E-05
0.00002
Time (s)
PSpice
ANSYS
Mathematica
vhdl


ANSYS vs. All
90x15x2 (micrometer)3
2.50E-06
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
-5.00E-07
Displacement (m)
0
0.000005
0.00001
0.000015
0.00002
Time (s)
PSpice
ANSYS
Mathematica
vhdl


ANSYS vs. All
90x20x2 (micrometer)3
1.80E-06
1.60E-06
1.40E-06
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
0.000005
0.00001
0.000015
0.00002
Time (s)
PSpice
ANSYS
Mathematica
vhdl


ANSYS vs. All
100x20x2 (micrometer)3
2.50E-06
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
-5.00E-07
Displacement (m)
0
0.000005
0.00001
0.000015
0.00002
Time (s)
PSpice
ansys
vhdl
Mathematica


ANSYS vs. All
100x25x2 (micrometer)3
2.00E-06
1.80E-06
1.60E-06
1.40E-06
1.20E-06
1.00E-06
8.00E-07
6.00E-07
4.00E-07
2.00E-07
0.00E+00
-2.00E-07
Displacement (m)
0
5E-06
0.00001
1.5E-05
0.00002
Time (s)
PSpice
Ansys
Mathematica
vhdl

Appendix VI
Results for design automation using MATHEMATICA

The following results correspond to chapter 9 and the code in Appendix II.

Pull- In Voltage (V)	**Mirror Length (microns)**	**Mirror Width (microns)**	**Beam Length (microns)**	**Beam Width (microns)**
10	87	87	44	44
11	83	83	42	42
12	79	79	40	40
13	77	77	39	39
14	73	73	37	37
15	71	71	36	36
16	69	69	35	35
17	67	67	34	34
18	65	65	33	33
19	63	63	32	32
20	62	62	31	31
21	61	61	31	31
22	59	59	30	30
23	58	58	29	29
24	57	57	29	29
25	56	56	28	28