MODELING OF AND EXPERIMENTS CHARACTERIZING ELECTROMIGRATION-INDUCED FAILURES IN INTERCONNECTS

by

VAIBHAV K. ANDLEIGH

B.S. Materials Science and Engineering
Cornell University, 1995

Submitted to the Department of Materials Science and Engineering
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
in ELECTRONIC MATERIALS
at the

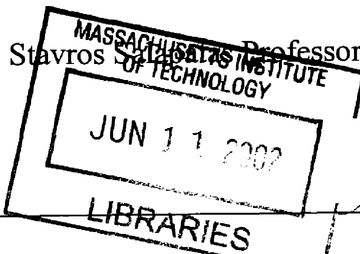
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

© Massachusetts Institute of Technology, 2001
All Rights Reserved

Signature of Author ______________________________
Materials Science & Engineering
May 4, 2001

Certified by ______________________________
Carl V. Thompson
Stavros Salapatas Professor of Materials Science and Engineering
Thesis Supervisor



Accepted by ______________________________
Harry L. Tuller
Chairman, Departmental Committee on Graduate Studies

ARCHIVES

MODELING OF AND EXPERIMENTS CHARACTERIZING ELECTROMIGRATION-INDUCED FAILURES IN INTERCONNECTS

by

VAIBHAV K. ANDLEIGH

Submitted to the Department of Materials Science & Engineering on May 4, 2001 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electronic Materials

ABSTRACT

As interconnect linewidths continue to scale downward, a detailed knowledge of stress evolution and void growth processes enables a determination of the electromigration-induced failure times of these interconnects. This thesis provides this knowledge through the development of an electromigration simulation MIT/EmSim and through experiments.

The stress effect on diffusivity and alloying effects were incorporated into the MIT/EmSim model for Al-based interconnects, demonstrating Cu transport effects consistent with experiments by IBM. The void nucleation and growth process was modeled in long and short lines, and current density exponents for Black's equation determined for several failure conditions. Interconnects can also be immortal, either without void nucleation or by resistance saturation. This complex reliability behavior can be catalogued in the form of failure mechanism maps using simulation or analytical approximations, plotting failure or immortality mode as a function of current density and line length. To extend the MIT/EmSim model to Cu interconnects, experimental observation of void phenomenology and measurement of Cu-based electromigration parameters were performed using Cu/Ta interconnects specifically designed and fabricated at MIT. The knowledge gained from these experiments was used to develop an electromigration model for Cu, and incorporated into MIT/EmSim. Simulations comparing void growth in Al and Cu interconnects indicate that differences in failure times of these interconnects observed in accelerated tests may not necessarily be apparent at service conditions. A failure mechanism map constructed for damascene Cu demonstrates the absence of immortality by resistance saturation due to the shunt structure. Finally, proposed damascene designs eliminating the diffusion barrier at the studs may be expected to have an adverse effect on interconnect reliability due to the loss of short length effects.

The test structure developed in this thesis provides a simple means of testing the effects of new shunt and barrier layer technologies on the reliability of Cu-based interconnects.Through the use of the simulation, an accurate methodology for predicting the reliability of Al- and Cu-based interconnects in semiconductor chips has been developed. MIT/EmSim is now being used by Motorola and LSI Logic for evaluating interconnect reliability during the design of future Cu interconnects, and has also been used by numerous SRC-companies and universities through EmSim-Web for electromigration research.

Thesis Supervisor: Carl V. Thompson
Title: Stavros Salapatas Professor of Materials Science and Engineering

To My Parents and GrandParents,
Whose Direction and Support Led Me
To Complete This Degree

Acknowledgments

My completion of this degree would not have been possible without significant contributions from my colleagues and friends.

I wish to thank Prof. Carl Thompson, whose guidance and support as my thesis advisor helped shape this research into a complete dissertation important to the semiconductor industry.

Additional members of my thesis committee, Prof. Suresh and Prof. Antoniadis, also provided valuable suggestions on improving the content of this thesis.

The research performed in this thesis was supported by the Semiconductor Research Corporation, and their support is appreciated. The experiments performed in this thesis could not have been completed without the aid of the staff and facilities of the Microsystems Technology Laboratories at MIT. In particular, Kurt Broderick and Vicky Diadiuk were very helpful in the development of my fabrication process.

My research group has also been very helpful during my research. Brett was a wonderful mentor. Dr. Fayad has always been a good friend and colleague, and I already miss our daily discussions. I have also cherished my friendship with Dr. Kobrinsky, and wish him the best in his future career aspirations. I am certain he will reach the top of his field, and will be rewarded with a very well-deserved cubicle. I also would like to thank Rob, Felix, Steve, Srikar, and Cody for their friendship and favors over the years. Now that Felix and I have graduated, I count on Zung to carry on the spirit of the Big Red.

My friends helped make my time at MIT more enjoyable. I've enjoyed having Jason and Matt as roommates, and wish them well in their future careers (and advise them to steer clear of those proximity mines!). I hope they come and visit me often after graduation, and Jason, don't forget the jelly-filled munchkins. Over the years, I have always looked forward to meeting with Jason, Mike, Kevin, Phil, Jeev, and Matt for our weekly discussions over a plate of General Gau's, and the occasional pitcher of mango. Certainly, our Kart bouts and movies added to the entertainment. I also wish to thank Mayank("the king"), Tom, Gianni, Sajan, and Chris, along with rest of Prof. Fitzgerald's group, for all the good times. Finally, I also wish to thank Dave, Katherine, Joanne, Li-li, Anton, Rajeev, Manish, Paul, and John for their years of close friendship and support. These friendships have always been very dear to me, and helped influence my decision to remain in Boston.

Most of all, I thank my family for their constant encouragement and support for as long as I can remember, and in particular, for their career guidance during my studies.

Finally, I am grateful to ACUNIA for providing me a very worthy opportunity, and to also thank them for their patience.

Table of Contents

List of Figures 9

List of Tables 14

1 Introduction 15

1.1 Interconnects & Interconnect Scaling 15
1.2 Electromigration and EM-Induced Failure 19
1.3 Electromigration Drift Experiments 20
1.4 Electromigration in Interconnects 22
1.5 Effects of Alloying and Microstructure 27
1.6 Black's Equation 38
1.7 Summary 40
1.8 Thesis Organization 41

2 Modeling of Pure and Alloyed Interconnects 43

2.1 Introduction 43
2.2 Alloy Modeling 44
2.3 Alloy Effects in Homogeneous Structures 55
2.4 Alloy Effects in Polygranular Clusters 58
2.5 Conclusions 61

3 Void Nucleation & Growth in Long Lines 62

3.1 Introduction 62
3.2 Modeling of Void Nucleation & Growth 63
3.3 Void Evolution in Long Lines 65
3.4 Scaling Test Data For Long Lines 67
3.5 Conclusions 71

4 Void Nucleation & Growth in Short Lines **73**
4.1 Introduction 73
4.2 Resistance Saturation & Immortality 75
4.3 Scaling Test Data For Short Lines 78
4.4 Conclusions 81

5 Failure Mechanism Maps **83**
5.1 Introduction 83
5.2 Generation of Al Failure Mechanism Maps 84
5.3 Construction of Al-Cu Failure Map 92
5.4 Analytical Construction of Failure Maps 94
5.5 Conclusions 98

6 Measuring EM in Pure Cu Interconnects **100**
6.1 Introduction 100
6.2 Experimental 104
6.3 Characterization & Test Results 117
6.4 Conclusions 140

7 Simulation of Pure Cu Interconnects **143**
7.1 Introduction 143
7.2 Model Parameters 144
7.3 Void Nucleation & Growth 144
7.4 Current Density Scaling 145
7.5 Cu Failure Map 149
7.6 Future of Electromigration in Context 152
7.7 Conclusions 157

8 Summary and Future Research **159**

8.1 Summary 159

8.2 Future Research 164

Appendix A – MIT/EmSim Description and Source Code **168**

A.1 MIT/EmSim Help Manual 168

A.2 MIT/EmSim Source Code 177

A.3 EmSim-Web Description 198

A.4 MIT/EmSim-ELM Extension Source Code 200

A.5 MIT/EmSim Verification Tools 217

Appendix B – Helper Applications **221**

B.1 subline Code Generation Tool 221

B.2 Theracore Input Tool 232

B.3 find_ttf Analysis Tool 238

B.4 vgline Analysis Tool 246

B.5 calc_res Analysis Tool 251

B.6 MIT/EmSim Movie-Making Utility 255

B.7 YearSecondCalc Utility 255

B.8 Mask-Layout Utility 261

Appendix C – Analytical Methodologies **319**

C.1 Void Saturation Calculation 319

C.2 Grain Orientation Effects on Cu Reliability 321

References **326**

List of Figures

1-1 The hierarchy of interconnect levels allow each metal level varying functionality. Once fabricated, semiconductor chips are placed in environmentally-protective packaging. [Based on Lin 98] 16

1-2 As transistors and interconnects scale downward, signal transmission through interconnects becomes the major source of delay. [Lin 98] 18

1-3 The Blech drift structure is composed of Al strips deposited on TiN. Application of current results in the removal of Al at the cathode and the formation of hillocks at the anode. [Ble 76, courtesy of Kno_2 97] 21

1-4 Drift rate of four Al strips with varying lengths after application of 3.7E5 A/cm^2. 22

1-5 Interconnect metal is surrounded by a conducting shunt material, with blocking boundaries at either end. 23

1-6 The tensile stress at the cathode rises until the back-stress begins to suppress electromigration. At steady-state, the maximum stress is given by the expression in Eq. 1-2. The current density applied was 1.0 MA/cm^2. 25

1-7 An electromigration test of an Al-0.5wt%Cu interconnect results in void nucleation and growth until saturation. [From Fil 95] 27

1-8 Interconnects with varying linewidths and equivalent grain sizes can exhibit very different microstructures, leading to different bamboo and polygranular cluster length distributions. 29

1-9 This figure illustrates the complex dependence of median-time-to-failure (MTTF) of a population of interconnects as a function of linewidth-to-grain-size ratio. 30

1-10 A tensile stress enhances the diffusivity while a compressive stress reduces the diffusivity. 31

1-11 Stress versus position for an interconnect with electrons proceeding to the right for the case of (a) stress effect on diffusivity neglected and (b) included. Including this effect results in a compressive stress twice as large. 32

1-12 Lifetime data measured at accelerated conditions is generally scaled to service conditions using Black's equation assuming n=2. At higher densities, Joule heating can significantly alter the interconnect lifetimes. [Courtesy of Kon 97] 39

1-13 The failure times measured for a population of 100 μm long, Al-0.5 wt% Cu lines do not suggest a constant current density exponent. [Courtesy of Fil 95] 40

2-1 Implementation involves dividing interconnect into a series of cells and calculating the flux between cells. 45

2-2 Stress (a) and copper migration 9b) profiles for an Al-0.25% stud-to-stud line with all-bamboo microstructure at 1 MA/cm^2. 57

2-3 Time-to-failure results for a 200 μm interconnect with varying Cu alloying content. Increasing Cu contents results in improved lifetimes. 58

2-4 Stress (a) and copper migration (b) profiles for Al-0.25%Cu stud-to-stud lines with a 21 um polygranular cluster at the center of the line. 60

3-1 The nucleation of a void results in the formation of a stress relaxation zone adjacent to the void. 64

3-2 Stress evolution until void nucleation, followed by growth until steady-state. 66

3-3 The current density for void nucleation failure in Al and Al-Cu lines is n=2. The exponent for void growth failure is n=1 in Al and Al-Cu lines. 69

3-4 At high j, void-growth-limited failure is observed, resulting in an n=1 scaling exponent. At low j, failure is nucleation-limited, requiring an n=2 exponent. 70

4-1 Stress evolution (a) until void nucleation (b) followed by void growth until (c) a steady-state stress gradient develops. 74

4-2 After the incubation time, the void nucleates causing an increase in the line resistance. The void continues growing until void saturation is reached, resulting in no further increase in resistance. 78

4-3 Times-to-failure as a function of current density for several void length failure conditions. 80

5-1 A line length vs. current density failure mechanism map illustrating regions of immortality without void nucleation(NV), immortality due to void saturation (RS), failures with impeded void growth with n>1 (IVF), and unimpeded void growth-limited failures with n=1 (UGF) assuming a constant void failure length of 0.50 μm. 85

5-2 Line length vs. current density failure mechanism map based on a void volume failure condition of 1.0 μm. Resistance saturation occurs at longer lines for this failure condition relative to the more conservative failure criterion in Fig. 5-1. 88

5-3 Line length vs. current density failure mechanism map assuming a resistance failure condition of 25%. No void saturation region is present. 89

5-4 Line length vs. current density failure mechanism map assuming a resistance failure condition of 50%. A small void saturation region is present for all line lengths. 91

5-5 Failure map assuming that a 1.0% resistance increase causes failure, illustrating regions of immortality without void nucleation (NV), impeded void failure (IVF), void-nucleation-limited failure (VNL), and unimpeded mixed failure (UMF). 92

5-6 Failure mechanism map for bamboo Al-0.5wt%Cu, assuming a resistance failure condition of 25%, constructed for a service temperature of 100°C. The corresponding map for bamboo pure Al is identical to this map. A regime for anode failure (AF) is indicated on this map. 94

5-7 Failure mechanism map assuming a constant-void-length failure condition of 0.5 μm. Results from the use of the analytical model for the transition current density are shown as points on this plot, and indicate a good correlation with the simulation results. 98

6-1 Subtractive etch fabrication process for patterning Al-based interconnects. [Based on Gro 99] 101

6-2 Cu interconnects are fabricated by the damascene process, which requires the patterning of the dielectric followed by metal deposition and subsequent CMP. 102

6-3 Top and side view of probe pad interconnect structures. Cu is in white, Ta in black, and nitride in grey. 105

6-4	Die layout (a) composing of three mask layers. Alignment structures (b) and etch indicators are also shown.	106
6-5	Thicknesses of Cu / Ta films deposited by sputtering for all wafers fabricated.	109
6-6	Schematic of an electromigration test station.	114
6-7	Optical microscope image of several interconnects.	118
6-8	SEM image of an interconnect structure (a). A close-up of the interconnect is shown in (b). The bright white fuzziness is presumably due to the Ta.	119
6-9	Interferometric profilometry image of the interconnect test structure.	120
6-10	Joule heating tests demonstrate that the temperature rise in the interconnects is not significant.	121
6-11	Temperature profile of the fabricated based on joule heating simulations.	122
6-12	Typical resistance curves (a,b) observed for resistance saturation in interconnects. The resistance remains constant until a void nucleates, after which it grows linearly until reaching resistance saturation.	127
6-13	Typical resistance profile for an interconnect with dielectric failure, leading to a steep increase in resistance.	129
6-14	Profilometer image of failed interconnect, illustrating the formation of a hillock after dielectric failure.	130
6-15	Drift velocities for the Cu/Ta structure along with those reported for other Blech strips and damascene interconnects. Open data points represent unpassivated (NP) structures while closed data points represent passivated interconnects. Damascene structures are denoted by DP. Open and closed triangles represent the unpassivated and Si_3N_4 passivated Cu/Ta structures, respectively, used in this study.	136
6-16	Effective diffusivity plotted against the inverse of time yields the activation energy and the corresponding diffusivity coefficient.	137

7-1 After the initial development of a tensile and compressive stress (t=100 hrs), a void nucleates and grows (t=5,000 hrs) until void growth saturation is reached (t=100,000 hrs). 145

7-2 Current density scaling behavior of Cu interconnects is compared to that of bamboo Al-based technologies. Al data is replotted from Fig. 4-3. 147

7-3 The reliability of Cu interconnects at low current densities is significantly reduced upon the removal of the Si_3N_4 diffusion barriers. 149

7-4 Failure mechanism map of Cu interconnects assuminga 25% resistance failure criterion. Data obtained by simulation is marked by data points, while analytical approximations are represented by solid lines. 150

A-1 View of EmSim-Web data entry page. 199

C-1 Lognormal plot of normalized bamboo grain length, L/w, distributions in bamboo structures resulting from post-patterning annealing of polygranular strips with different widths. The overlapping curves for different w/D_0 show that the distribution of L/w is line-width-independent. Solid lines represent the best-fitting lognormal distribution. 322

C-2 Distribution of interface diffusivities as a function of bamboo grain orientation. The grain orientation is assigned randomly, and the diffusivity for a given orientation θ is given by $D=D_{fac}(1+ A \theta \ln(\theta) - B \theta)$ where $D_{fac} = 2.2*10^{-16}$ m^2/s at 392°C, A = 1.8, and B=0.55. 323

C-3 Lognormal plot showing the simulated variations in lifetimes resulting from interface diffusivity variations in lines with bamboo grain structures. 324

List of Tables

5-I	Legends Used in Failure Map Figures	86
6-I	Comparison of (jL) Product Values	131
6-II	Comparison of Average Nucleation Times	132
6-III	Comparison of z* Values	133
6-IV	Comparison of Drift Velocities	135
7-I	Subset of SIA Technology Roadmap	153

Chapter 1

Introduction

The completion of this thesis comes at a time of great change in interconnect and process technology. For over forty years, aluminum and its alloys have served as the primary interconnect material for connecting transistors on semiconductor chips. Today, the industry is in the midst of a transformation to copper-based interconnects fabricated by the damascene process, requiring swift modeling and characterization of these new technologies for rapid adoption into manufacturing. This thesis serves to characterize the reliability of both Al and future Cu interconnect technologies through simulation and experiments.

1.1. Interconnects & Interconnect Scaling

A typical semiconductor chip consists of a high purity, singular orientation, planar silicon wafer on which transistor devices are fabricated, as illustrated in Figure 1-1. The processing associated with fabricating the active device (transistor) area is referred to as front-end processing. Above the transistors are several layers of metal interconnects separated by insulating dielectric material such as silica. The interconnects are connected to the transistors and other interconnects on the metal level immediately above or below by vias. The metal levels closest to the transistor (typically referred to as metal 1 for the first metal level, and metal 2 for the second metal level, and so on) are the smallest and

most densely packed, approaching 0.13 um in linewidth at the time of this writing. On each of the higher metal levels, the linewidth gets larger , approaching 1 micron at the upper-most metal levels (now approaching 6-8 levels of metal) and the spacing between interconnects increases significantly. The fabrication of the interconnects and dielectric materials is collectively referred to as back-end processing. The chip is encapsulated in a relatively thick, mechanically- and environmentally-protective coating known as passivation. The semiconductor piece, or die, is then mounted in chip packaging.

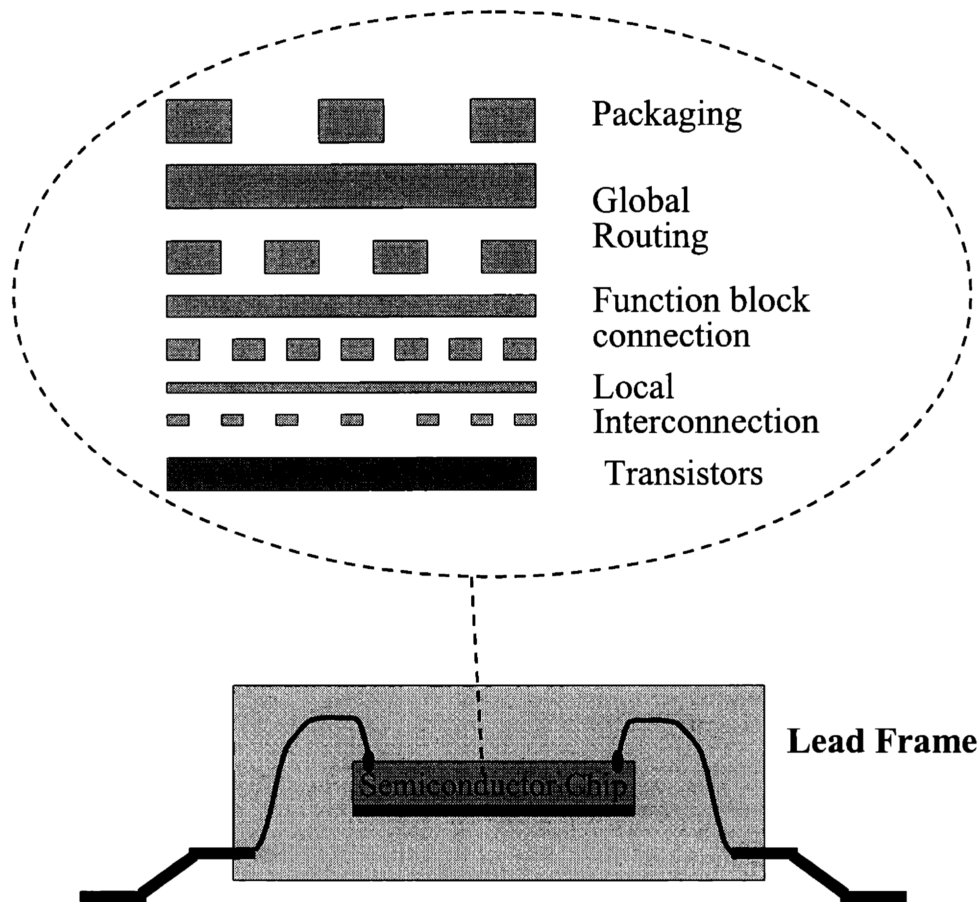


Figure 1-1. The hierarchy of interconnect levels allow each metal level varying functionality. Once fabricated, semiconductor chips are placed in environmentally-protective packaging. [Based on Lin 98]

The interconnects on different levels perform different functions. The metal 1 and metal 2 layers link the transistors to form the basic circuit elements while the middle layer allows for local signal propagation. The upper most metal layers provide electrical power and long-range signal propagation, such as clock distribution. In addition, these interconnects also make contact with the outside world through the chip packaging.

Historically, transistor size has been decreasing dramatically every year to exploit the overwhelming technological and financial benefits associated with reduced feature size [Boh 96]. Smaller transistors may be switched at a faster rate, resulting in improved microprocessor performance. They also require less power to operate, resulting in reduced thermal generation and power consumption. In addition, smaller transistors favor the fabrication of significantly more transistors per wafer, enabling either smaller chip (die) sizes or additional processing power per chip. With revenue per wafer increasing at a far greater rate than cost per wafer, there has been a great drive to build advanced fab facilities with ever smaller minimum feature size capabilities and larger wafer diameters, even as the fab construction costs climb to near $3 billion at the time of this writing [Lin 00]. As transistor size scaled downward, the interconnects that connect the individual transistors to other transistors and the outside world also scaled similarly downward to exploit the overwhelming technological and financial advantages associated with die shrinkage.

As of late, the semiconductor community has been hampered in its quest for higher performance through interconnect scaling. As interconnect linewidths on the lower metal levels have scaled downward, their resistance has increased (resistance is inversely related to linewidth), leading to increased concerns about the propagation delay associated with transmitting a signal through an interconnect, also known as the RC delay. As the RC delay of interconnects have increased relative to transistor switching time, signal propagation has threatened to impose a ceiling on the maximum microprocessor speed. In essence, the maximum performance of a semiconductor chip is

now more limited by delays from signal transmission through the interconnects rather than the transistor switching time. (See Fig. 1-2)

To compensate for this, the semiconductor community eased the interconnect scaling requirements by adding additional metal levels to accommodate the high density of interconnects necessary near the transistors. This, however, increases process complexity and costs of interconnect fabrication. To address this, over the last two years, industry has abandoned its dependence on aluminum alloys and has begun to fabricate pure copper interconnects. Pure copper interconnects offer a technological advantage of a 30% reduction in resistivity, reducing the RC delay by 30%. By switching to Cu

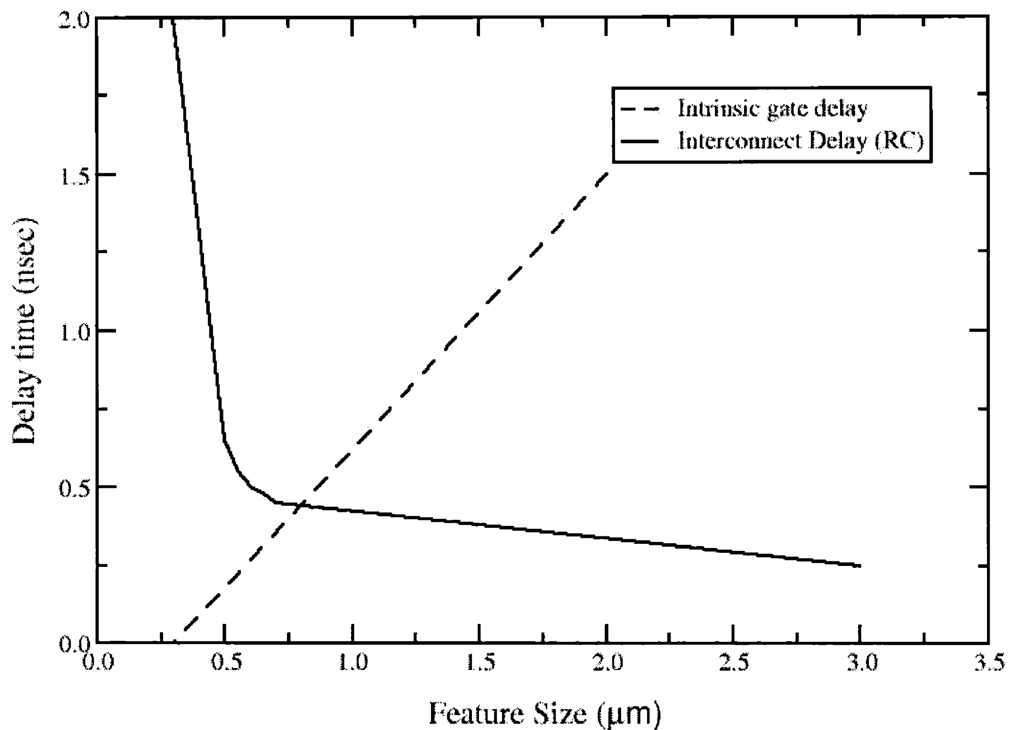


Figure 1-2. As transistors and interconnects scale downward, signal transmission through interconnects becomes the major source of delay. [Lin 98]

interconnects, which may be processed at finer linewidths at lower costs for a given amount of RC delay as compared to Al, the rate of increase in interconnect metal levels may be reduced. Coupled with the changeover in interconnect metal is a move toward new insulator materials encapsulating the interconnects with a lower dielectric constant, known as low-k dielectrics. These new dielectrics reduce the RC delay associated with the capacitance of the dielectric. Processing pure Cu interconnects, using the damascene process, is also significantly simpler and cheaper than aluminum alloys, providing a strong financial impetus to rapidly implement pure Cu interconnects into standard chip fabrication processes. With this rapid introduction of Cu interconnect technology, there lies a need to study the performance and processability of pure Cu, and in particular, interconnect reliability.

So far, the introduction of Cu has been limited to the upper-most interconnect levels due to contamination concerns, fearing that Cu can act as a deep-level trap if present in the active region of a transistor, thereby decreasing recombination times and altering device behavior. For this reason, the semiconductor community has avoided placing Cu interconnects near the transistor region, relegating Cu to the higher metal levels instead. Metal 1 and 2 continue to be aluminum-based alloys in most chips, although this trend may not hold in the future. At the time of this writing, there have already been recent industrial announcements describing all-Cu interconnect implementations in both microprocessors and DRAM.

1.2. Electromigration & EM-Induced Failure

Interconnect reliability is governed by the damage associated with electromigration. Electromigration is the diffusion of atoms in an interconnect due to an electron momentum transfer arising from an applied current. [Mal 97] Electromigration was first discovered to reduce the reliability of Al interconnects in 1966. [Ble 66] The electromigration force is referred to as the "electron wind", and the atomic flux, J_a,

resulting from this force is given by:

$$J_a = \frac{D_a C_a}{kT}(eZ^* \rho j), \tag{1.1}$$

where D_a is the appropriate atomic diffusivity, k is Boltzmann's constant, T is the test temperature (in Kelvin), e is the fundamental electron charge, Z^* is an effective valence of the atoms, j is the current density, and ρ is the metal resistivity [Hun 61]. Electromigration reduces the lifetime of an interconnect when there is a divergence in the atomic flux at a specific point along the interconnect, leading to evolving mechanical stresses in confined interconnects, and, in turn, to the nucleation and growth of voids and hillocks which result in failure [Ble 76, Mal 97, Tho 93].

1.3. Electromigration Drift Experiments

Shortly after electromigration was first observed in Al interconnects, an experimental technique for measuring the electromigration drift ratewas developed by Blech. [Ble 76, Ble_2 76, Ble 77] These experiments, commonly referred to as drift experiments, were performed by applying a current through strips of Al deposited on TiN (see Fig. 1-3). Because Al has a higher conductivity, the majority of the current flows through the Al, leading to vacancy drift to the cathode. Because most metals diffuse by a vacancy exchange mechanism, there will be a corresponding atomic drift of Al to the anode. [She 63] This leads to a depletion of Al at the cathode, and its subsequent accumulation at the anode leading to the formation of hillocks.

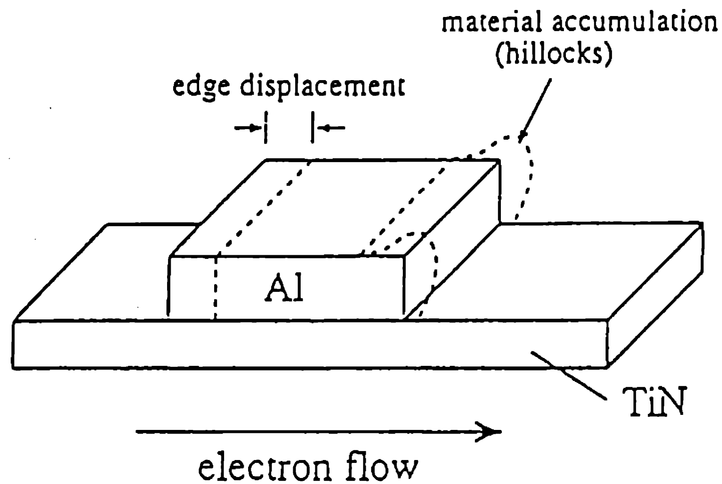


Figure 1-3. The Blech drift structure is composed of Al strips deposited on TiN. Application of current results in the removal of Al at the cathode and the formation of hillocks at the anode. [Ble 76, Kno_2 97]

Blech demonstrated that the drift was inversely related to the strip length, as shown in Fig. 1-4. This very important result suggests that the transport of Al leads to the formation of a back-stress balancing the electromigration force. [Ble 76]. From Fig. 1-4, it is apparent that there is a minimum stripe length for which no electromigration drift occurs at this current density, often described as the *Blech length*. The product of the current density and strip length, known as the (jL) product, is also known as the Blech criterion. From these results, it is apparent that drift experiments conducted under conditions exhibiting a sufficiently high (jL) product will undergo mass transport, forming voids and hillocks. Experimental conditions favoring a low (jL) product will result in the development of a back-stress opposing electromigration, and no mass transport of metal will be observed. Drift experiments will be revisited in the experiments described in Chap. 6 of this thesis.

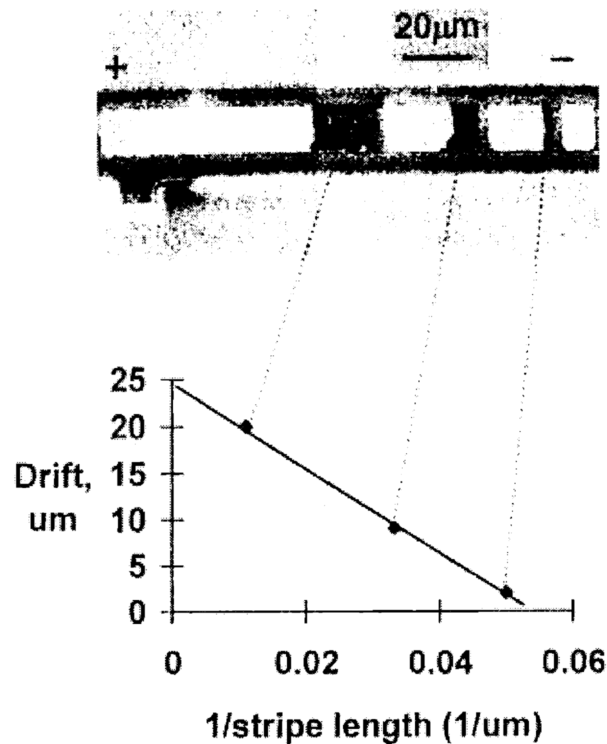


Figure 1-4. Drift of four Al strips with varying lengths after application of 3.7E5 A/cm^2. [Ble 76]

1.4. Electromigration in Interconnects

Electromigration behavior in interconnects is more difficult to characterize due to the increased complexity of the interconnect structure and impaired visual observation because interconnects are generally passivated.

On-chip interconnects consist of metal or alloy (Al or Cu based) lines, encapsulated in a rigid oxide matrix, and clad with, or surrounded by, refractory metal layers (such as Ta, TiN, or Al_3Ti). These lines often terminate at contacts or vias filled with refractory metal layers or plugs (see Fig. 1-5). While Al-based and Cu-based metals electromigrate under conditions of practical interest, the refractory metals do not. For this reason, the studs generally act as blocking boundaries at either end of the interconnect.

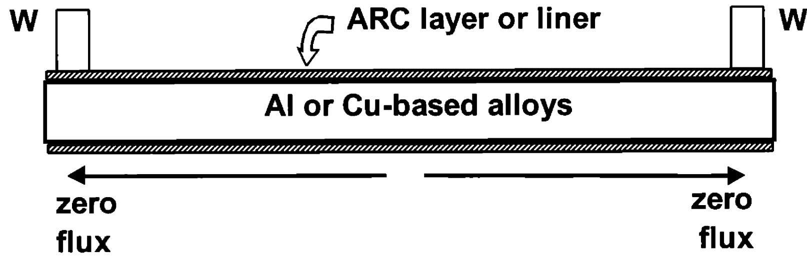


Figure 1-5. Interconnect metal is surrounded by a conducting shunt material, with blocking boundaries at either end.

Al and Cu have relatively high diffusivities and electromigration rates when compared to the materials used for liners and anti-reflection coatings, as well as materials such as W which are used to fill vias and contacts. Therefore, when an interconnect terminates at low diffusivity barriers at contacts or vias (collectively referred to here as studs), these diffusion barriers will also lead to atomic flux divergences during electromigration. A tensile stress will develop at the cathode end of the line and a compressive stress (also known as back-stress) will develop at the anode end. A stress gradient will develop, and will suppress the electromigration flux as given by

$$J_a = \frac{D_a C_a}{kT}\left(eZ^* \rho j + \nabla\mu\right), \tag{1.2}$$

where $\nabla\mu$ is the gradient in chemical potential. [Kor 93] This gradient in chemical potential is generally associated with "downhill diffusion", although it can also include chemical potentials of individual elements of alloys, which will be treated in the next chapter.

If the stress in the interconnect exceeds the critical stress for failure, the line will fail (in the simplest case) and the interconnect reliability will be a strong function of the critical stress for failure. This critical stress is associated with the nucleation of a void spanning the width of the interconnect, resulting in an open circuit and the interconnect is said to fail by a tensile failure. Several more complex failure mechanisms will be described and characterized in Chapter 5.

Immortality

Interconnects in service are generally passivated by a rigid dielectric to provide electrical isolation and mechanical stability. This dielectric prevents the formation of hillocks more commonly observed in drift structures assuming excessive stresses are not developed. When a current is applied, metal atoms are transported downwind, the metal at the cathode is depleted while atoms accumulate at the anode. Because of the stiff dielectric, reduced atom density at the cathode results in the development of a hydrostatic tensile stress while similarly, an increased atom density at the anode results in the development of a compressive stress. For sufficiently short interconnects, the stresses can evolve and interact with each other. For an interconnect of length L stressed at a sufficiently low current density, the back-stress will increase until the resulting stress-induced gradient in chemical potential exactly opposes the electromigration wind force, assuming the interconnect does not fail. (See Fig. 1-6) If the diffusivity is taken to be independent of stress, the maximum stress (σ_{max}) within an interconnect at steady-state with a stud-to-stud length L is

$$\sigma_{\max} = \frac{jLZ^{*}e\rho}{2\Omega}, \tag{1.3}$$

where Ω is the atomic volume [Ble 76].

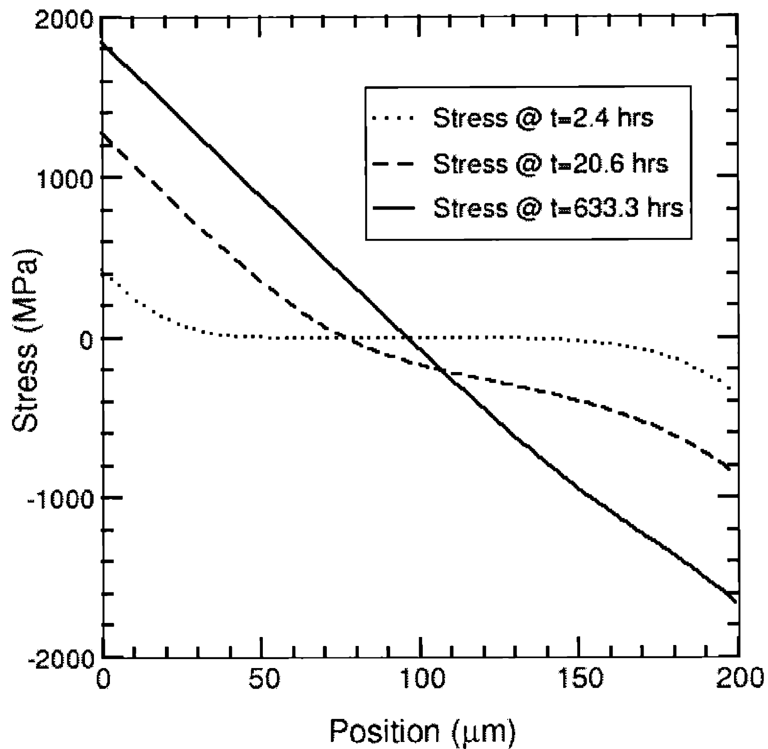


Fig. 1-6. The tensile stress at the cathode rises until the back-stress begins to suppress electromigration. At steady-state, the maximum stress is given by the expression in Eq. 1-2. The current density applied was 1.0 MA/cm^2.

From Eq. 1.3, it is apparent that stress evolution is a strong function of line length and current density. However, the generation of stress alone does not always result in interconnect failure. If the maximum stress in an interconnect at steady-state is less than some critical stress associated with failure, the interconnect will never fail and is considered *immortal*. For failures by the simplest failure mechanism, inwhich void nucleation results in failure of the interconnect, Equation 1.3 is often rewritten in the form,

$$(jL) = \frac{2\sigma_{crit}\Omega}{Z^* e\rho}. \qquad (1.4)$$

where σ_{crit} is the critical stress for failure. This expression is commonly referred to as the *jL product*. Interconnects with a sufficiently low (jL) product will be immortal, while others will fail by the nucleation of a void (in the simplest case).

Void Nucleation & Growth

If the tensile stress at the cathode in the interconnect reaches some critical stress (σ_{crit}), a void will nucleate and grow. A number of studies have been conducted to examine void nucleation and growth behavior through experiments [Rie 96, Arz 94, Mar 95, Mar 95] and simulations [Bor 92, Kra95]. Depending on the nature of the specific interconnect materials, several different modes of failure in interconnects may result. In lines which do not have refractory layers (underlayers and ARC layers, or liners in Cu-based interconnects), once a void nucleates, it can quickly grow to span the width of the line and cause a void-nucleation-limited open circuit failure. In lines with refractory electromigration-resistant layers, current can shunt around voids, so that voids can span the width of the lines and continue to grow for long times after nucleation. For short interconnects, the back-stress could interact with void, and hamper void growth. For short lines with sufficiently low current densities, void growth could cease, leading to void growth saturation as first predicted by Korhonen. [Kor 93] This has been observed experimentally by Filippi [Fil 95, Fil 96], as shown in Fig. 1-7. If the resistance of an interconnect at void growth saturation is below the failure resistance, the interconect will also be immortal. Void behavior will be investigated in substantially more detail in Chaps. 3-5.

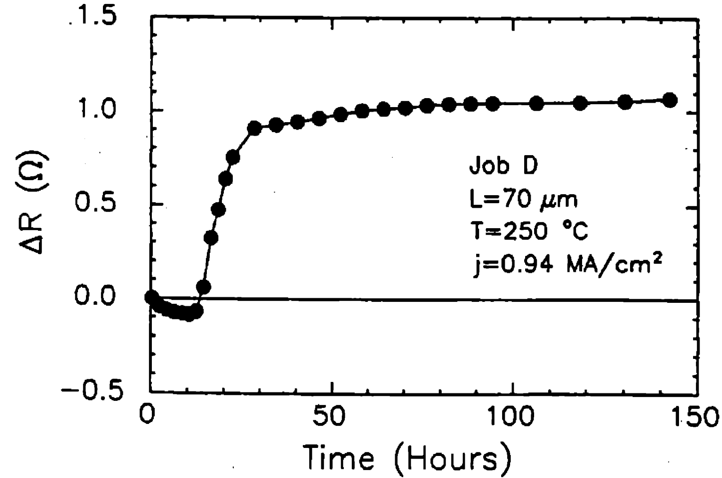


Figure 1-7. An electromigration test of an Al-0.5wt%Cu interconnect results in void nucleation and growth until saturation. [From Fil 95]

Electromigration is primarily a concern on the upper-most levels of metal associated with power distribution and long-range signal propagation. This is due to the high currents and long line lengths involved at these levels. Electromigration is less of a concern on the lower metal levels. Since these lower level interconnects are used primarily for local communication, the line lengths are shorter, reducing the likelihood of interconnect failure. Also, since the interconnects on the lower levels have not been scaled at the same rate as the higher levels, the linewidths have not decreased as rapidly, slowing the increases in service current densities. Because electromigration continues to be of concern, especially at the upper metal levels, significant studies have been conducted to explore different methods of extending interconnect lifetimes to improve interconnect reliability. Some of these methods will be summarized in the next sub-section.

1.5. Effect of Alloying and Microstructure

The reliability of interconnects is significantly affected by alloying additions and microstructure. Control of the interconnect composition and microstructure allows the

reduction in electromigration-induced damage, and thereby improved lifetimes of interconnects.

Alloy Effects

The use of alloys has been shown to have a dramatic effect on extending interconnect lifetimes. It has been shown that the reliability of Al-based interconnects increases by over an order of magnitude with 1at% Cu alloying additions to Al [Ame 70, How 71, Aga 72, Ble 77, Hu 93]. In Al-Cu alloys, copper atoms segregate to the grain boundaries and slow the diffusion of aluminum along the otherwise high diffusion paths such as grain boundary. This reduced Al diffusivity slows the evolution of mechanical stress and consequently increases the lifetime and reliability of the interconnect. Several models have been proposed to characterize EM-induced diffusion in alloys, and this will be described in further detail in Chap. 2.

Microstructure Effects

The nature of the microstructures of interconnect materials has also been shown to profoundly affect their reliability [Vai 80, Cho 89, Wal 91, Wal 92, Tho 93]. In Al-based interconnects, lines with bamboo microstructures, in which all grain boundaries lie in planes perpendicular to the axis of the interconnect, have been shown to have drastically increased lifetimes because of the absence of high-diffusivity grain boundary paths along the length of the lines. Interconnects with near-bamboo microstructures, have a mixture of lengths along which there are grain boundary paths for rapid electromigration (polygranular clusters) and lengths in which there are only bamboo grains (bamboo clusters). [Joo 94] Interconnects with near-bamboo structures have reliabilities which are strongly coupled to the detailed statistical characteristics of the grain structure. [Joo 94, Kno 97, Kno_2 97] Sites of flux divergence are created at the transitions between low diffusivity bamboo regions and high diffusivity polygranular regions. This results in the depletion of atoms at the upwind edge of the cluster, thereby leading to a tensile stress. Similarly, an accumulation of atoms at the downwind edge of the cluster results in the

formation of a compressive stress. The magnitude of the developed stresses depends on the bamboo and polygranular cluster length distributions, which in turn depends on the linewidth and mean grain size (see Fig. 1-8). [Joo 94, Tho 93, Tho 95, Kno 97, Fie 97]

The median-time-to-failure (MTTF) has been demonstrated to be a complex function of the linewidth-to-grain-size ratio (w/d), as shown in Fig. 1-9. Above a w/d ratio of 5, the MTTF is relatively constant (corresponding to the upper interconnect in Fig. 1-8). Below w/d=5, the MTTF drops as a result of the distribution of, and complex interactions of bamboo lengths and polygranular clusters (see lower interconnect in Fig. 1-8). [Cho 94] As the linewidth becomes increasingly small relative to the grain size, the length of, and number of polygranular clusters decreases, leading to an increase in the reliability of the interconnect. Deviations in the time-to-failure are shown to increase as the linwidth-to-grain size ratio decreases. [Cho 94] Knowlton *et al.* further demonstrated that failure times of test populations of interconnects are bimodal when a critical stress failure criterion is used. Failure can either be grain boundary diffusion-induced or transgranular. The effects of microstructure on Cu interconnects will be discussed in Appendix C.2.

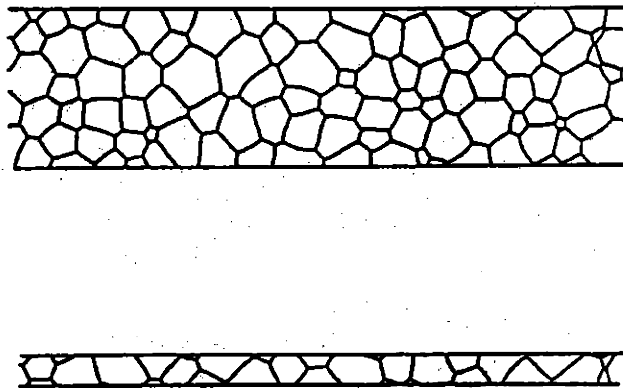


Figure 1-8. Interconnects with varying linewidths and equivalent grain sizes can exhibit very different microstructures, leading to different bamboo and polygranular cluster length distributions. [Tho 93]

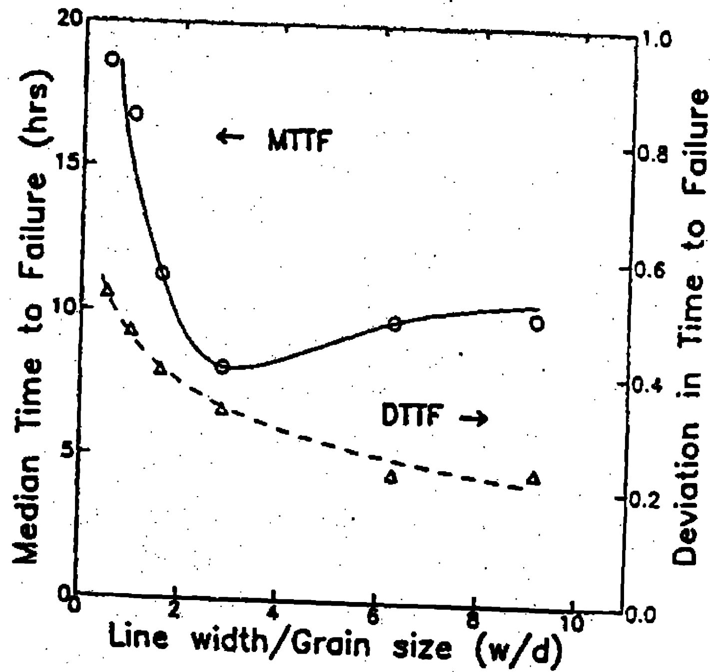


Figure 1-9. This figure illustrates the complex dependence of median-time-to-failure (MTTF) of a population of interconnects as a function of linewidth-to-grain-size ratio. [Cho 94]

Stress Effect on Diffusivity

It has been shown that the lifetime of interconnects with near-bamboo microstructure are strongly affected by the stress effect on diffusivity. [Par 97] Stresses in confined interconnects affect the diffusivity by altering the vacancy concentration and lattice density. In particular, tensile stresses increase the vacancy concentration and reduce the lattice density, resulting in an exponential increase of the diffusivity. Conversely, compressive stresses will decrease the diffusivity (see Fig. 1-10). The phenomenological origins and theory associated with the curves in Fig. 1-10 will be discussed in further detail in the next chapter.

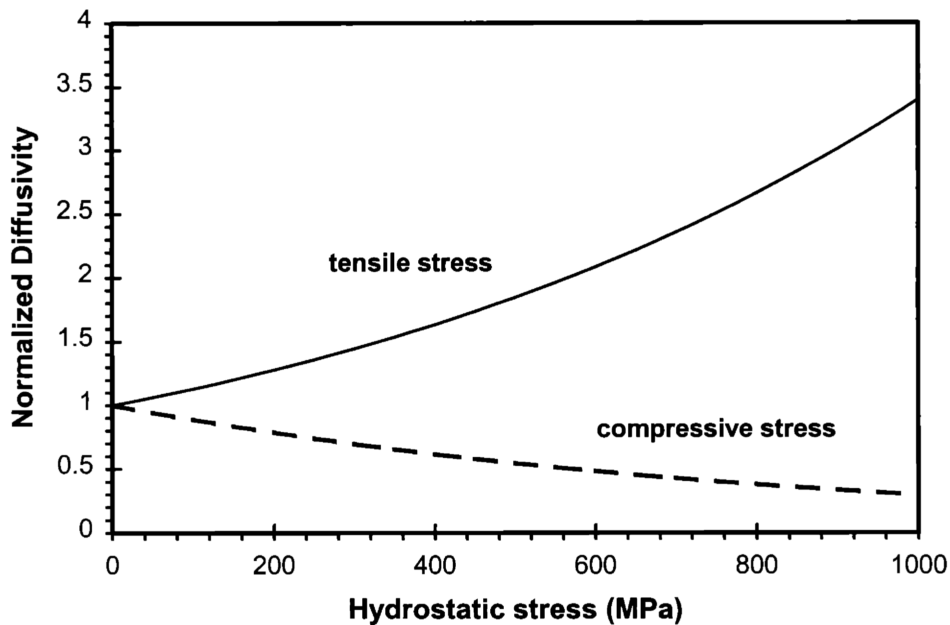


Figure 1-10. A tensile stress enhances the diffusivity while a compressive stress reduces the diffusivity.

The effects of the stress effect on diffusivity are most pronounced in polygranular clusters in near-bamboo lines, in which the resulting steady-state compressive stress in the cluster is twice as large as compared to when this force is neglected. (See Fig. 1-11) This results in an addition of a coefficient of 2 in equation 1.3 to account for this artifact [Par 97].

Electromigration in such structures can be modeled using a one-dimensional diffusion-drift equation, whose solution requires knowledge of the initial conditions, the boundary conditions, the transport parameters of the atomic species involved, the geometry of the structure, and the test conditions. If the spatial and temporal evolution of the electromigration-induced stress is known, the reliability of interconnects can be estimated by associating a critical failure criterion with each mode of failure.

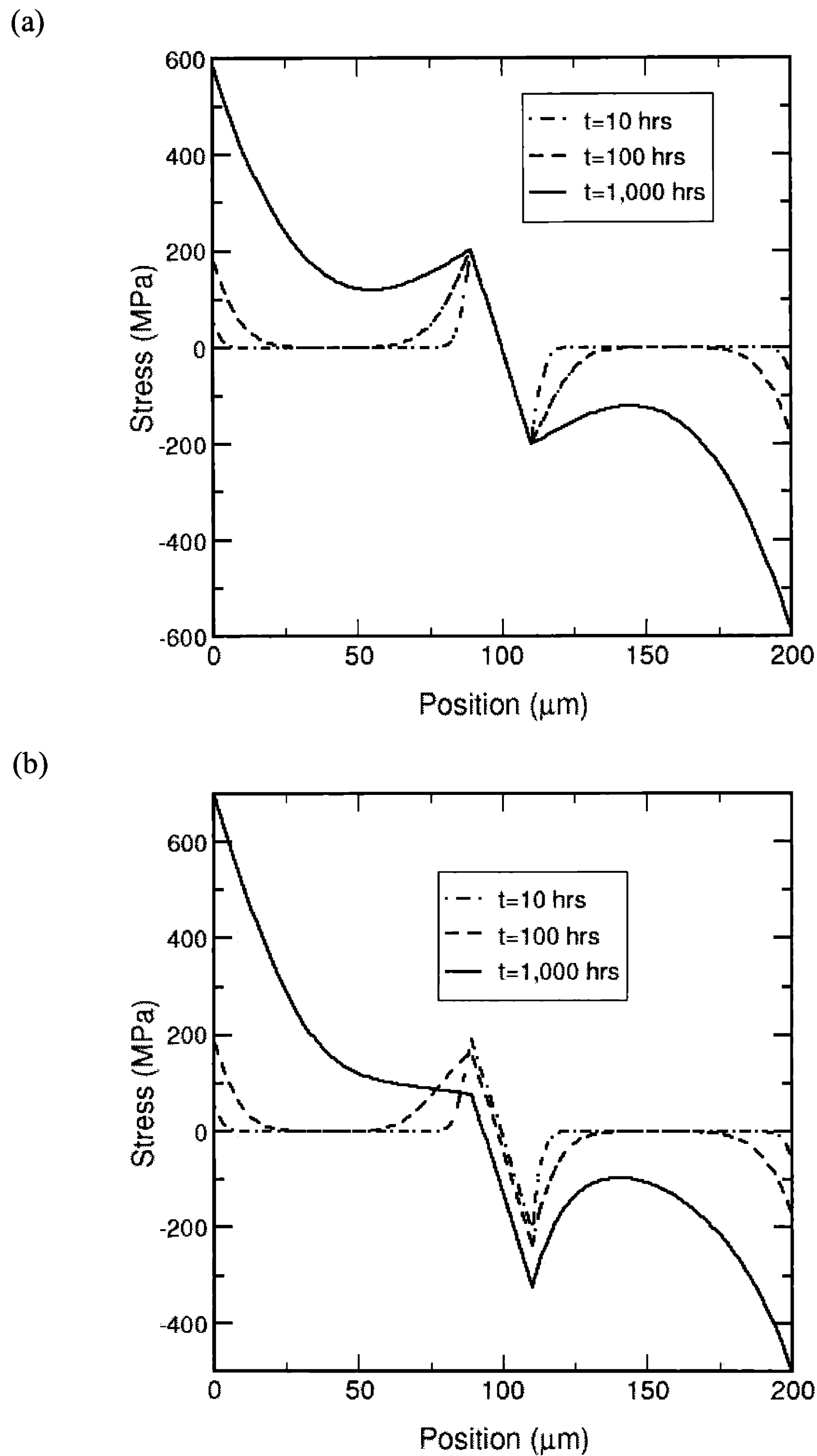


Figure 1-11. Stress versus position for an interconnect with electrons proceeding to the right for the case of (a) stress effect on diffusivity neglected and (b) included. Including this effect results in a compressive stress twice as large.

Electromigration Modeling

The need for accurate predictions of interconnect lifetimes has lead to considerable research in developing models of electromigration. The more commonly accepted model today was developed by Korhonen *et al.* [Kor 93, Kor_2 93], which models the stress evolution by balancing the wind force with a chemical-potential-induced back-stress. This model was used by the Cornell group to study stress evolution of polygranular clusters. [Bro 95] Knowlton complemented the Korhonen model with a front-tracking grain growth simulator [Wal_2 91, Wal 92, Kno 97, Kno_2 97] to simulate electromigration in interconnects with varying distributions of polygranular and bamboo segments. Shatzkes and Lloyd solve the diffusion equations analytically by approximating flux divergences as blocking boundaries assuming a constant vacancy source. Lloyd and Kitchin use a LaPlace transform in their analytic solution to explore current density scaling (see Sec. 1.6) while assuming a constant vacancy source. [Llo 94] Kircheim and Kaeber incorporate a sink-source term in their model and simulations to explore its effect on the vacancy supersaturation limit associated with void nucleation. Several other electromigration models for pure metal have also been proposed recently. [Dwy 96, Lo 97, Liu 98] To model electromigration in Al-Cu alloys, several models have been suggested, and this will be discussed in more detail in Chap. 2. [Ros 72, Kor 95, Dek 97] None of the models discussed above incorporate the stress effect on diffusivity.

To model void nucleation and growth, several different approaches have been developed. Void nucleation is generally assumed to occur when the hydrostatic stress in the interconnect exceeds a specific critical stress for void nucleation. While this model may seem simplistic in light of the 3D stress state that may be expected to occur at interface site for heterogenous void nucleation, this hydrostatic stress is relatively simple to measure in and compare with experiments. Korhonen assumes that the size of the void can be calculated by integrating the stress curve along an interconnect. [Kor_2 93]

Clement uses a similar model for simulating electromigration under dc and ac conditions. [Cle_2 97] Several other void models have also been proposed. [Bor 92, Kra 95]

Korhonen Model

The Korhonen stress evolution model [Kor 93] is increasingly gaining acceptance as a means to model electromigration in dielectric-encapsulated interconnects. This model tracks atomic transport based on Eqn. 1.2, whereby these atoms are assumed either alter the vacancy concentration or to accumulate at lattice dislocations or grain boundaries. The dielectric (and shunt) are modeled as a perfectly rigid confinement, in which thermally-induced and EM-induced strains lead to the formation of hydrostatic stresses. Others have studied the case in which the confinement is not perfectly rigid, leading to a triaxial stress state. [Fli 93, Sau 92, Kor 91] In Korhonen's stress evolution model, the lateral stresses are assumed to be equal to maintain a constant chemical potential. Because interconnects lengths are much larger than their thickness and linewidths, a 1D electromigration analysis is sufficient to model the stress evolution along the length of the interconnect.

By calculating the flux of atoms between adjacent volume units, the corresponding change in stress can be determined by assuming that climbing grain boundary or other dislocations alter the number of lattice sites, and thereby the lattice density by:

$$\frac{\partial C}{C} = -\frac{\partial \sigma}{B}. \tag{1.5}$$

where B is the appropriate modulus. To determine this modulus, Korhonen modeled an encapsulated interconnect as an elliptic cylinder Eshelby inclusion encapsulated by an infinitely large Si matrix. From the uniform free strains, a uniform stress state is formed in an interconnect. [Kor 93] It has been shown that the simplification of a rectangular-prismatic interconnect as an elliptic cylinder does not alter the dilational stresses

significantly. [Kor 92] Three-dimensional finite element studies have been performed at Brown to study the shape evolution and migration of voids due to electromigration and strain-induced surface diffusion. [Zha 99] The results from these simulations were compared to other two-dimensional finite element simulations to determine the conditons under which three dimensional simulations are necessary. However, the results from these types of simulations have been difficult to compare to experiment because of the obstacles associated with measuring stresses in three dimensions at a sub-micron scale. In summary, the Korhonen model assumes that electromigration-induced fluxes lead to the generation of hydrostatic stresses by assuming the interconnect is an Eshelby inclusion in a rigid matrix.

Preview of MIT/EmSim Model

In this thesis, a new model (MIT/EmSim) is proposed and developed (Chaps. 2-5, 7). The purpose of the modeling component of this thesis is to develop a physically-based, circuit-level electromigration simulation tool that accurately models the the electromigration-induced mass transport in interconnects interconnects with varying microstructures, compositions, and geometries in a reasonably short period of time. This will enable accurate predictions of stress evolution and electromigration-induced failure times of large populations of interconnects, allowing the investigation of conditions leading to interconnect failure, and for the generation of failure statistics of realistic interconnect structures. Further development of this circuit-level electromigration tool for the rapid prototyping of the interconnect lifetime and reliability of individual elements of *entire* mask levels will support the eventual incorporation of MIT/EmSim into industrial reliability CAD tools.

The MIT/EmSim model is an extension of the Korhonen model [Kor 93] as developed by Clement and Thompson. [Cle 95] The assumptions and simplifications [Kor 93, Cle 95] present in the Korhonen stress evolution model are maintained in the MIT/EmSim model. Because Al interconnects can be assumed to be significantly longer

in length than they are wide or thick, mass transport in interconnectscan be modeled using a one-dimensional finite element representation of the interconnect. The interconnects are assumed to terminate at tungsten studs, which act as diffusion barriers to Al transport leading to flux divergences. The interconnects are assumed to be encapsulated by a rigid oxide dielectric that adheres well to the interconnect, and atomic transport within this rigid matrix can lead to the generation of stresses. The vacancy concentration is assumed to remain in equilibrium with the stresses through the creation and annihilation of vacancies by dislocation climb at sources and sinks, which are located at grain boundary and lattice dislocations, shunt interfaces, and diffusion barrier interfaces. [Cle 95] The majority of net flux will be taken up by climbing grain boundary dislocations and lattice dislocations, with a significantly smaller amount of vacancies consumed by stress generation. [Kor 93] Because the vacancy concentration is significantly smaller than the lattice density, dislocations climb need only occur over relatively short distances to maintain vacancy equilibrium. As an initial condition for interconnects, the vacancy concentration and lattice density are assumed to be constant throughout the interconnect. [Kor 93, Cle 95] The determination of the stresses generation due to atomic transport in a confined medium is based on the Eshelby theory of inclusions, in which the interconnect is modeled as an elliptic cylinder in an infinite Si matrix. The stresses can be determined based on the uniform, dilational free strains arising from atomic transport assuming an effective modulus, which depends on the mechanical properties of the metal and dielectric, and on the metal aspect ratio and interconnect geometry. The critical stress for void nucleation depends on the mechanical strength of the oxide glass and assuming a homogenous metal structure, the critical stress will remain constant over the length of an interconnect. Contamination of the metal interfaces can lead to the contrary, however. [Mar 95, Fli 95] Because of the rigid confinement, coupled with the difficulty of experimentally measuring triaxial stress states in interconnects, the magnitude of stresses generated and the critical stress for failure are tracked assuming a hydrostatic stress state. The magnitude of the critical stress for void nucleation has been reported to vary between 100 and 1000 MPa. [Kno 97] A discussion

of assumptions inherent for several different failure conditions is detailed in Chaps. 2-5. By assuming a hydrostatic stress state in a one-dimensional interconnect model, electromigration-induced mass transport and stress evolution can be simulated using the MIT/EmSim model to predict the electromigration-induced failure times of Al interconnects. The assumptions listed above form the basis for the MIT/EmSim model, and further modifications of this model is described in this Thesis.

Park modified this model to incorporate the stress effect on diffusivity. [Par 97] Park [Par 99] also assisted in the implementation of alloys, employing the Rosenberg model for Cu-trapping in alloys [Ros 72]. Gradients in chemical potential are calculated based on polynomial expressions developed by Murray [Mur 85]. In this thesis, alloying effects on stress evolution, electromigration-induced failure times, and precipitate formation are investigated and characterized. The effects of microstructure in Al-based are treated in a companion program to MIT/EmSim, EmSimGen, developed by Fayad. [Fay 98, Fay 00] A novel approach to the modeling of voids will be presented in this thesis in Chap. 3, and used to study void nucleation and growth in the context of void phenomenology, current density scaling, and the construction of failure mechanism maps. Based on experimental results from Chap. 6, a revised model for simulating electromigration in Cu interconnects is presented in Chap. 7. In appendix C.2, a model for predicting the deviation in mean-time-to-failure (MTTF) is presented for Cu interconnect based flux divergences associated with the varying grain orientation microstructure. [Fay 01] The major contributions of this thesis to the MIT/EmSim model include: investigations of simulations using the MIT/EmSim alloying model, the model development and in-depth treatment of void nucleation and growth to study current density scaling in interconnects, and the adaptation of the MIT/EmSim model to Cu interconnects based on experimental observation and measurements conducted using Cu/Ta interconnects that were fabricated in the experimental component of this thesis.

1.6. Black's Equation

Experimental characterization of interconnect reliability is carried out through accelerated tests on populations of lines to cause failure to occur in a reasonable period of time (typically under a thousand hours), since electromigration is a slow and stochastic process. Current density and temperature are used as acceleration factors, and test data must be scaled to service temperature and current densities, and Black's equation [Bla 69] is almost universally used to scale accelerated test data to service conditions. This equation relates the median time-to-failure (t_{50}) of a population of interconnects to the current density and temperature

$$t_{50} = Aj^{-n} \exp\left(\frac{Q}{kT}\right), \tag{1.6}$$

where A is a current density and temperature independent constan, Q is the activation energy for the rate-limiting atomic-level process required for failure, usually associated with diffusion, and the current density exponent, n, which is generally taken to be 2 (see Fig. 1-12). It has been shown that an exponent of 2 is consistent with nucleation-limited failures in models based on the Korhonen analysis [Kor 93]. The rate of unconstrained void growth should be proportional to j, so that the current density exponent for void-growth-limited failure is expected to be 1 [Kir 91]. Oates has observed current density exponents close to n=1 in Al lines with a TiN shunt layer [Oat 95]. However, current density exponents of 2 have been found in experiments on stud-to stud structures which might also have been expected to fail through void-growth-limited mechanisms [Ata 94]. It has been suggested that Cu alloy additions may be responsible for observations of n=2 in stud-to-stud structures [Llo 96]. Knowlton observed that the void-nucleation-based failure times of Al interconnects composed of a distribution of polygranular clusters and bamboo regions exhibit n=2 scaling for certain ranges of current densities. In some

experiments, the current density exponent does not appear to be constant (see Fig. 1-13). [Fil 95] Knowledge of the appropriate value of n is critical in scaling test results to service conditions, since in changing n from 1 to 2, the projected in-service reliability can change by orders of magnitude. This current density scaling behavior will be explored in significantly more detail in Chaps. 3-5 and 7.

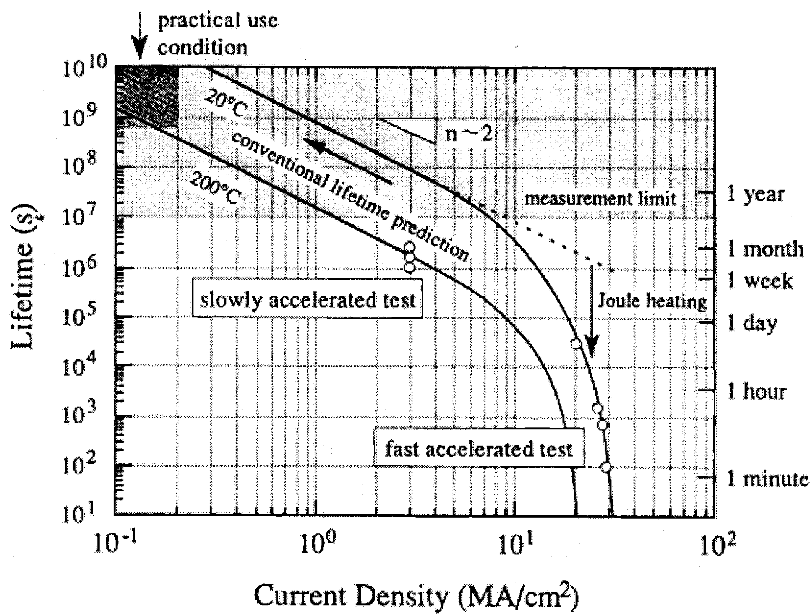


Figure 1-12. Lifetime data measured at accelerated conditions is generally scaled to service conditions using Black's equation assuming n=2. At higher densities, Joule heating can significantly alter the interconnect lifetimes. [Kon 97]

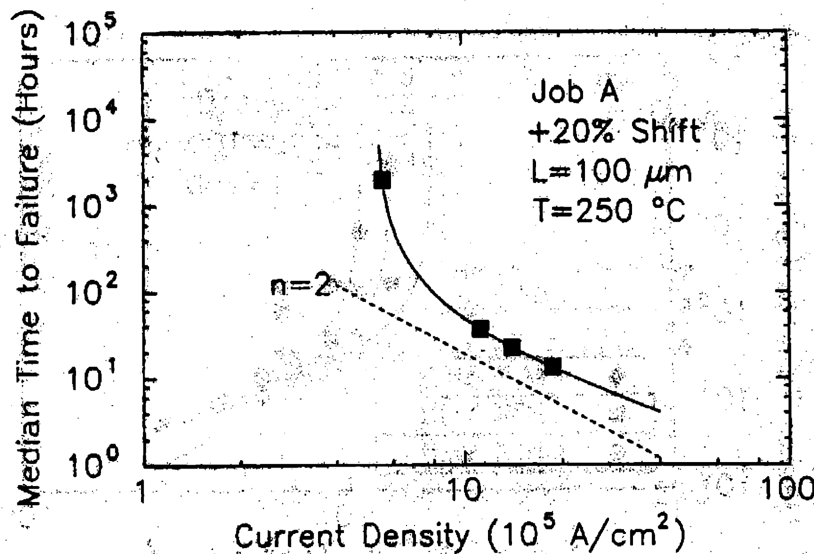


Figure 1-13. The failure times measured for a population of 100 μm long, Al-0.5 wt% Cu lines do not suggest a constant current density exponent. [Fil 95]

1.7. Summary

In summary, the demand for microprocessor performance has driven the need for transistor and interconnect scaling. One result of this intensive pace of scaling has been increased current densities and concern about interconnect reliability, specifically electromigration. As industry reached the limits of Al-based interconnects, fabs have begun the transition to Cu-based technology to alleviate EM concerns and reduce RC delay. The rapid introduction of Cu interconnects has resulted in the need for detailed characterization of interconnect reliability.

A review of the knowledge base about electromigration reveals that the immortality of an interconnect is a strong function of line length and current density. The maximum stress generated in the interconnect relative to the critical stress for void

nucleation will determine whether a line is immortal or fails due to a tensile failure. Effects of microstructure and the stress effect on diffusivity must be included to accurately model the electromigration-based failure processes. Although Black's equation is commonly used to scale accelerated test data, the value of the current density exponent is not well understood. An accurate reliability assessment requires a knowledge of the detailed dependence of the time evolution of stress and the resulting failures based on interconnect geometry, choice of metallization materials, and the effects of interconnect processing.

1.8. Thesis Organization

It is the purpose of this thesis to develop an understanding of interconnect reliability in a materials-generic manner, whose models and results will be applicable to both Al-based and Cu interconnects. In this work, we first develop reliability models for Al-based interconnects (Chap. 2), and compare the results to existing data on Al-Cu reliability. These models will be further advanced to study the void nucleation in growth process in *long* lines (Chap. 3). Electromigration-induced void nucleation and growth in *short* interconnects will be characterized in Chap. 4. The complex reliability behavior of both long and short lines will then be catalogued in the form of failure mechanism maps (Chap. 5). Chapter 6 details experimental work geared toward measuring important materials parameters associated with Cu reliability and enable development of the Cu electromigration models. These models, as adapted for pure Cu, are then used to offer insight and input to MIT/EmSim, and to characterize Cu interconnect reliability (Chap. 7). The results of the thesis are summarized in Chap. 8, with references included after the appendices.

Appendix A contains a description and instruction manual to the source code to the MIT/EmSim electromigration model developed as a part of this thesis. It contains the source code to EmSim-Web, the online version of this electromigration simulator.

Appendix B contains a collection of the many helper applications and shell scripts written for data analysis for MIT/EmSim and analysis of experimental measurements. Appendix C contains a theoretical derivation associated with electromigration void phenomena.

Chapter 2

Modeling EM in Pure and Alloyed Interconnects

2.1. Introduction

It has been shown that the reliability of Al-based interconnects increases by over an order of magnitude with 1at% Cu alloying additions to Al [Hu 93, Ble 77]. In Al-Cu alloys, copper atoms segregate to the grain boundaries and slow the diffusion of aluminum along the otherwise high diffusion paths such as a grain boundary. This reduced Al diffusivity slows the evolution of mechanical stress and consequently increases the lifetime and reliability of the interconnect. Several models have been proposed to quantitatively account for this reduced Al diffusivity in the presence of Cu (in solution): Korhonen *et al.* [Kor 95] have assumed that the diffusivity decreases ten-fold when the Cu concentration was depleted below a threshold concentration of 0.1%;while Lloyd and Clement [Llo 96] assumed that the diffusivity of Al drops to zero above an unspecified threshold Cu concentration. Rosenberg developed a trapping model [Ros 72] to predict a continuous dependence of the Al diffusivity with Cu concentration. This trapping model provides the relatively drastic yet continuous change of Al diffusivity with Cu concentration necessary to explain experimental results while also providing a reasonable physical explanation of the phenomena. We will use this model to describe the effects of Cu alloy additions on Al electromigration, and as a prototype for treatment of the effects of other alloy additions to Al or alloy additions to pure Cu.

The use of alloys also leads to a chemical driving force for diffusion which can affect electromigration, in addition to the electrical and mechanical driving forces. To incorporate the effects of the chemical driving force, Korhonen *et al.* [Kor 95] adopted an ideal solution model for the Gibbs free energy of Al-Cu alloys. In our analysis reported below, we have used a polynomial representation of the free energy of Al-Cu alloys, as given by Murray [Mur 85], for determining the chemical driving forces for electromigration. This polynomial representation was confirmed by Murray through comparisons with experimentally obtained phase diagrams for Al-Cu alloys.

The time to failure is affected by the atomic scale diffusive processes which govern the evolution of stress along an interconnect. Modeling of the temporal and spatial evolution of stress throughout the failure process in both pure and alloyed systems therefore provides the basis for failure modeling and reliability simulations, and is the focus of this chapter.

2.2. Model Formulation

A number of theoretical formulations have been developed to describe electromigration [Sha 86, Kor 93, Kir 91, Cle 95]. We have developed a simulation of stress evolution based on the model of Korhonen *et al.* [Kor 93], as modified by Clement and Thompson [Cle 95]. We have found results which are similar to those obtained by Korhonen *et al.* [Kor 93] in examining the effects of microstructure on stress evolution, but have also found that it is important to take into account the effects of stress on diffusion during electromigration [Par 97]. A tensile stress increases the vacancy concentration aiding diffusion while a compressive stress will depress the vacancy concentration, hampering diffusion. Not only does the stress state affect diffusivities through effects on the concentration of vacancies, but stress also affects the activation volume, increasing it and enhancing diffusion in tensile stress states, and decreasing it and suppressing diffusion in compressive stress states. The stress state can therefore

significantly alter the diffusivity of metals, leading to significant effects on stress evolution and failure predictions for interconnects. Earlier analyses of electromigration in alloys [Kor 95, Llo 96] have assumed that diffusivities are independent of stress, and have also been based on very simple models for the effects of Cu on the chemical potentials and diffusivities of Al and Cu (as described in the introduction). The mathematical formulation used in the analysis described earlier [Par 97] and extended below, has been developed into an electromigration simulation, MIT/EmSim. [Ems 96]

The Atomic Flux

We use a formulation based on the conservation of atoms [Par 97], in which the flux of atoms through incremental volumes along the interconnect length is calculated as time is incremented, and the time-dependent evolution of the Al and Cu concentration in constrained volumes and the resulting stress state are determined. (see Fig. 2-1) The atomic flux(J) is written as the product of the velocity(v) and atom concentration(C)

$$J = v \times C. \tag{2.1}$$

The velocity can, in turn, be written as the product of the change in the free energy associated with atomic motion (ΔG) and the atomic mobility (M)

$$v = \Delta G \times M. \tag{2.2}$$

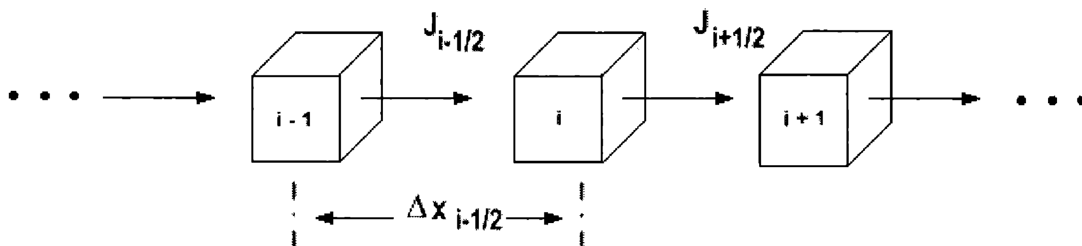


Figure 2-1. Implementation involves dividing interconnect into a series of cells and calculating the flux between cells.

The change in the free energy can be described as the sum of all the partial derivatives of the chemical potentials involved

$$\Delta G = -\nabla \mu_{elect} - \nabla \mu_{mech} - \nabla \mu_{chem} , \tag{2.3}$$

where $\nabla \mu_{elect}$ is the electric-current-induced driving force due to the momentum transfer from the electron wind, $\nabla \mu_{mech}$ is the mechanical driving force or back-stress caused by stress gradients, and $\nabla \mu_{chem}$ is the chemical driving force due to gradients in the concentration of Al and/or Cu. The first two driving forces can be calculated using [Tho 93]

$$\Delta \mu_{elect} = Z^{*} eE = Z^{*} e \rho j \tag{2.4}$$

and

$$\Delta \mu_{mech} = -\sigma \cdot \Omega_{Al/Cu} , \tag{2.5}$$

where Z* is the effective charge, e is the electron charge, ρ is the resistivity, j is the current density, σ is the stress, and $\Omega_{Al/Cu}$ is the atomic volume. In this analysis, for simplicity, we assume that Al and Cu have the same substitutional atomic volumes. Murray [Mur 85] has given two polynomial expressions for the chemical potentials of Al and Cu which we use to calculate the Gibbs free energy for Al and Cu in Al-Cu FCC alloys. They are

$$\mu_{chem}^{Al} = \mu_{o}^{Al} + kT \ln x_{Al} + \frac{\left[B - C + 2(3C - B)x_{Al} - (9C - B)x_{Al}^{2} + 4Cx_{Al}^{3}\right]}{N_{A}} \tag{2.6a}$$

and

$$\mu_{chem}^{Cul} = \mu_{o}^{Cu} + kT \ln x_{Cu} + \frac{\left[B - C - 2(3C + B)x_{Cu} + (9C + B)x_{Cu}^{2} - 4Cx_{Cu}^{3}\right]}{N_{A}} , \tag{2.6b}$$

where

$$B = -28353 - 13.5780T$$

and (2.6c)

$$C = 22364 + 12.0517T\,, \qquad (2.6d)$$

and where μ_o^{Al} and μ_o^{Cu} are the chemical potentials for pure Al and Cu, x_{Al} and x_{Cu} are the mole fractions of Al and Cu respectively, and N_A is Avogadro's number. Note that in this formulation, all the copper atoms are assumed to be in solution and the effects of Al_2Cu precipitates are neglected for simplicity.

The mobility M in Eq. (2.2) is related to the diffusivity (D) by the Einstein relation

$$M = \frac{D}{kT}\,. \qquad (2.7)$$

The diffusivity is dependent upon three important factors:

(1) temperature affects the thermal energy of the atoms, and therefore the probability of successful jumps over the activation barrier,
(2) stress affects the activation volume and the concentration of vacant sites for substitutional diffusion, and
(3) the concentration of Cu affects the degree of copper-vacancy trapping.

As usual, thermal effects are accounted for by assuming that the diffusivity has an Arrhenius form

$$D' = D_0 \exp\left[-\frac{\Delta H}{kT}\right], \tag{2.8}$$

where ΔH is the activation enthalpy for diffusion. The effect of stress on diffusivity is taken to have the form [Cle 95]

$$D = D' \exp\left[\left(\frac{\Omega_a}{kT} + \frac{1}{b}\right)\sigma\right], \tag{2.9}$$

where Ω_a is the atomic volume of the Al-Cu alloy and b is the appropriately defined modulus for the metal-dielectric composite. [Kor 93] The first term in the exponential in Eq. (2.9) refers to the change in vacancy concentration due to the applied stress while the second term results from a change in lattice density (and consequent change in the activation volume) due to the applied stress.

We use the trapping model of Rosenberg to account for the reduction in diffusivity caused by copper-vacancy trapping at the grain boundary. [Ros 72] In the absence of copper atoms, all vacancies are assumed to be mobile and lead to a diffusivity of Al atoms at grain boundaries given by $D_{Al,gb}^{pure}$. In the presence of copper, however, it is assumed that some fraction of the vacancies become bound to the copper atoms, limiting the diffusivity of those trapped vacancies to that of the copper in the grain boundary ($D_{Al,gb}^{sat}$). Hence the overall Al diffusivity ($D_{Al,gb}$) depends on the fraction of mobile vacancies (f_{mob}) diffusing at a fast rate ($D_{Al,gb}^{pure} f_{mob}$) and the fraction of immobile (bound) vacancies (f_{imm}) diffusing at the slower rate of ($D_{Al,gb}^{sat} f_{imm}$):

$$D_{Al,gb} = D_{Al,gb}^{pure} f_{mob} + D_{Al,gb}^{sat} f_{imm}. \tag{2.10}$$

Using the condition for a mass balance, it can be shown that the fraction of mobile and immobile vacancies can be given by [Ros 72]

$$f_{mob} = \frac{1}{1 + [Cu_{Al}] S \exp\left(\frac{\Delta H_B}{kT}\right)}, \tag{2.11a}$$

where

$$f_{imm} = 1 - f_{mob}, \tag{2.11b}$$

and where $[Cu_{Al}]$ is the mole fraction of copper atoms at the grain boundary, S is the lattice coordination number at the grain boundary, and ΔH_B is the copper-vacancy binding energy. The values of $D^{o}_{pure}, D^{o}_{sat}$, ΔH^{pure}, and ΔH^{sat} can be obtained from experimental results and S may be assumed to be 12 for FCC materials. However, the exact values of $[Cu_{Al}]$ and ΔH_B are more difficult to determine. $[Cu_{Al}]$ is related to the Cu concentration in the bulk Al grains, through the segregation coefficient. Recent work shows that the segregation coefficient for Cu in Al grain boundaries can vary drastically with temperature and grain boundary structure [Fea 93]. Therefore, for simplicity, $[Cu_{Al}]$ is assumed to be equivalent to the average Cu concentration in this analysis. In addition, the dependence of the Al diffusivity on the Cu concentration is also very highly sensitive to ΔH_B. As ΔH_B increases, the Al diffusivity decreases rapidly and reaches the Cu-saturated value. Since it has been experimentally shown that the Al diffusivity increases sharply near a Cu concentration of about 0.1 at% [Kor 95], a value of 0.25eV for ΔH_B seems appropriate and is adopted in this analysis.

In bamboo regions, grain boundaries are not available for rapid electromigration of Cu and Al. Experiments on single crystal Al interconnects suggest that when grain boundaries are not present, the transport path for both Al and Cu is along the Al/oxide interface [Joo 94, Sri_2 99]. In the absence of evidence to the contrary, we will assume

that Cu plays a similar role in reducing the rate of Al electromigration at the Al/dielectric or Al/liner interfaces as it does at grain boundaries, so that that the effects of Cu on electromigration in bamboo elements can be accounted for in a way analogous to the one outlined above for diffusion and electromigration in polygranular clusters.

Using Eqs. (2.8-2.10), the diffusivities for Al and Cu in bamboo and poly-granular (grain boundary) regions may be written as

$$D_{Al,bam} = D^{pure}_{Al,bam,0} \exp\left(-\frac{\Delta H^{pure}_{Al}}{kT_{test}}\right) \exp\left[\left(\frac{\Omega_a}{kT_{test}} + \frac{1}{b}\right)\sigma\right], \tag{2.12a}$$

$$D_{Al,gb} = \left\{D^{pure}_{Al,gb,0} \exp\left(-\frac{\Delta H^{pure}_{Al}}{kT_{test}}\right) f_{mob}\right\} \exp\left[\left(\frac{\Omega_a}{kT_{test}} + \frac{1}{b}\right)\sigma\right] + \left\{D^{sat}_{Al,gb,0} \exp\left(-\frac{\Delta H^{sat}_{Al}}{kT_{test}}\right) f_{imm}\right\} \exp\left[\left(\frac{\Omega_a}{kT_{test}} + \frac{1}{b}\right)\sigma\right], \tag{2.12b}$$

$$D_{Cu,bam} = D^{pure}_{Cu,bam,0} \exp\left(-\frac{\Delta H^{pure}_{Cu}}{kT_{test}}\right) \exp\left[\left(\frac{\Omega_a}{kT_{test}} + \frac{1}{b}\right)\sigma\right], \tag{2.12c}$$

and

$$D_{Cu,gb} = D^{pure}_{Cu,gb,0} \exp\left(-\frac{\Delta H^{pure}_{Cu}}{kT_{test}}\right) \exp\left[\left(\frac{\Omega_a}{kT_{test}} + \frac{1}{b}\right)\sigma\right]. \tag{2.12d}$$

The value of $Z^*_{Cu} D_{Cu,gb} / Z^*_{Al} D_{Al,gb}$, in this model changes with the Cu concentration. For example, $Z^*_{Cu} D_{Cu,gb} / Z^*_{Al} D_{Al,gb}$ varies from 0.85 to 20 as the Cu concentration is increased from 0 to 0.5at%. In contrast, in the model of Korhonen *et al.* this ratio retains a constant value of 5 [Kor 95]. Considering the fact that Al grain boundary diffusivities are more sensitive to Cu concentration than Cu grain boundary diffusivities, we feel that the model adopted here leads to more realistic characteristics. This difference is very significant in regions in which Cu is depleted. If the Cu is

sufficiently depleted, the electromigration of Al is faster than that of Cu, thereby leading to significant Al depletion prior to the removal of all of the Cu. This behavior is consistent with the experimental results of Hu *et al.* [Hu 92, Hu 93], who observed that significant electromigration did not occur until the Cu was depleted, whereupon the Al immediately migrated.

Using the atomic diffusivities of Al and Cu in grain boundaries, the apparent diffusivity in the poly-granular cluster region was obtained by considering the ratio of the grain boundary width to the linewidth. In this analysis, the ratio was assumed to be 4×10^{-5}. While the atomic flux occurs by fast grain boundary diffusion in the poly-granular regions [Llo 96], the atomic flux occurs via relatively slow interface diffusion within the bamboo regions. For this analysis, diffusion in bamboo regions is assumed to be 200 times slower than that in the poly-granular cluster regions.

Using Eqs. (2.1-2.3) with Eq. (2.7), an overall expression for the flux is given by

$$J = \frac{DC}{kT_{test}}\left(-\nabla\mu_{elec} + -\nabla\mu_{mech} + -\nabla\mu_{chem}\right). \quad (2.13)$$

Further substitution of Eqs. (2.4-2.6) results in expressions for the flux of Al and Cu atoms:

$$\begin{aligned} J_{Al} = & \frac{D_{Al}C_{Al}}{kT_{test}}\{Z^* e\rho j + \Omega\nabla\sigma - kT_{test}\nabla \ln x_{Al} \\ & - \frac{D_{Al}C_{Al}}{kT_{test}}\left\{\frac{\nabla}{A}\left[B - C + 2(3C - B)x_{Al} - (9C - B)x_{Al}^2 + 4Cx_{Al}^3\right]\right\} \end{aligned} \quad (2.14a)$$

and

$$J_{Cu} = \frac{D_{Cu}C_{Cu}}{kT_{test}}\{Z^* e\rho j + \Omega\nabla\sigma - kT_{test}\nabla \ln x_{Cu}$$
$$- \frac{D_{Cu}C_{Cu}}{kT_{test}}\left\{\frac{\nabla}{A}\left[B + C - 2(3C + B)x_{Cu} + (9C - B)x_{Cu}^2 - 4Cx_{Cu}^3\right]\right\}, \quad (2.14b)$$

where either the bamboo or grain boundary diffusivity of Al and Cu is used in these equations, depending on which is relevant at a given location.

Vacancy Concentration

In the formulation described above, the vacancy concentration is assumed to remain in equilibrium with the stress. The vacancy concentration is dependent upon two effects:

(1) the balance between the enthalpy and the entropy of formation, and
(2) the effects of stress.

In balancing the enthalpic and entropic terms, the vacancy concentration (C_v) is given by

$$C_v = C_{v,0} \exp\left(-\frac{E_v^F}{kT}\right), \quad (2.15)$$

where E_v^F is the energy of formation of a vacancy and $C_{v,0}$ is the vacancy concentration in the unstressed lattice. The coupling between the vacancy concentration and stress can be described by [Cle 95]

$$\partial C_v = \frac{\Omega C_v}{kT}\partial\sigma, \quad (2.16)$$

where Ω is the atomic volume. Integration of Eq. (2.16) relates the vacancy concentration directly to stress as shown below:

$$C_v = C_{v,0} \exp\left(\frac{\Omega\sigma}{kT}\right) . \quad (2.17)$$

A change in atom concentration over time, $\left(\frac{\partial c}{\partial t}\right)$, will result in a change in stress over time, $\left(\frac{\partial \sigma}{\partial t}\right)$, given by

$$\frac{\partial \sigma}{\partial t} = -b\Omega \frac{\partial c}{\partial t} . \quad (2.18)$$

Integration of Eq. (2.18) leads to an expression relating atomic concentration (C) to the stress as shown below

$$C = C_0 \exp\left(-\frac{\sigma}{b}\right), \quad (2.19)$$

where C_0 is the atomic concentration under zero stress, or simply the inverse of the atomic volume ($1/\Omega$). Differentiating Eq. (2.17) with respect to time and substituting into Eqs. (2.7,2.15, and 2.19) results in an expression for the source/sink term for the change in vacancy concentration with time(γ) [Cle 95]

$$\gamma = -\frac{b\Omega}{kT_{test}} \frac{C_{v,0}}{C_0} \exp\left[\frac{\Omega\sigma}{kT} + \frac{\sigma}{b} - \frac{E_v}{kT}\right] \frac{\partial C}{\partial t} . \quad (2.20)$$

Stress Evolution

Using the continuity equation, the change in atom concentration over time ($\partial C/\partial t$) is accompanied by a vacancy source/sink term (γ) and the divergence of the flux (∇J) such that

$$\frac{\partial c}{\partial t} = \gamma - \nabla J . \quad (2.21)$$

The atoms that accumulate due to the flux divergence will be accommodated through an increase in the number of lattice sites, either through dislocation climb, or through plating out of new planes at grain boundaries or interfaces (climb of grain boundary or interface 'dislocations'). The increase in the number of lattice sites in a volume constrained by the dielectric will result in a compressive increase in the stress, and it is assumed that the vacancy concentration will change to remain in equilibrium with the local stress. Substituting Eq. (2.14) and (2.20) into the continuity equation (2.21) gives an expression for the change in atom concentrations for Al and Cu [Par 97]

$$\frac{\partial C}{\partial t} = \frac{\nabla J}{1 + \frac{b\Omega}{kT_{test}} \frac{C_{v,0}}{C_0} \exp\left[\frac{\Omega\sigma}{kT_{test}} + \frac{\sigma}{b} - \frac{E_v}{kT_{test}} \right]} \quad . \tag{2.22}$$

Integrating Eqs. (2.18) and (2.22) over each time step yields the atom concentrations and stress profiles as a function of time. Alternatively, rearranging Eq. (2.19) gives an expression which can be used for the direct calculation of the stress given the concentration of atoms

$$\sigma = -b \ln\left(\frac{C}{C_0} \right) . \tag{2.23}$$

Equations (2.22) and (2.23) form the basis of our electromigration simulation tool.

In simulating the temporal and spatial evolution of stress along the length of an interconnect line, we consider volumes of incremental lengths along the line axis (which span the full cross-section of the line) and track the flux in and out of these volumes during incremental time steps. Net increases in the atomic concentration are then related to stress changes and changes in the vacancy concentration. Diffusivites into or out of a volume element are taken to be a function of the nature of the microstructure at either

boundary of the volume (polygranular or bamboo), and the Cu concentration and the stress are tracked for calculation of the relevant diffusivities. (see Fig. 2-1)

A number of topics were studied using the approach described above, and as embodied in the tool MIT/EmSim[Ems 96]. Areas of investigation included analyses of the following:

(1) the effects of copper on the lifetime of all-bamboo lines, and
(2) copper migration in Al-based lines with near-bamboo structures.

Simulations

In the simulations to be described below, the following input values were used: Z^*_{Al}=4, Z^*_{Cu}=12, $D^{pure}_{Al,bam,0}$=9.5E-8m^2/s, $D^{pure}_{Al,gb,0}$=1.9E-5m^2/s [Has 74], ΔH^{pure}_{Al}=0.6eV [Has 74], $D^{sat}_{Al,bam,0}$=9.5E-8m^2/s, $D^{sat}_{Al,gb,0}$=1.9E-5m^2/s, ΔH^{sat}_{Al}=0.8eV, $D^{pure}_{Cu,bam,0}$=4.9E-4m^2/s [Ho 74], $D^{pure}_{Cu,gb,0}$=9.8E-2m^2/s, ΔH^{pure}_{Cu}=1.0eV [Sim 60], ΔH_B=0.25eV, E^F_v=0.75eV [Sim 60], b=50GPa, $\Omega=\Omega_a$=1.66E-29m^3/atom, S=12, and ρ=5 $\mu\Omega$-cm, as first selected by Y.J. Park *et al.* [Par 97] In all the cases reported below, we have also assumed that the test was conducted at 200°C.

2.3. Copper Effects in All-Bamboo Interconnects:

The effects of copper on 200 μm long stud-to-stud, all bamboo, Al-0.25%Cu structures were investigated at a current density of 1 MA/cm^2. The time to nucleation-limited failure was found to increase by over an order of magnitude when compared to pure Al. Figure 2-2 shows a series of stress and copper-concentration profiles at different simulation times for this structure. The generation of a tensile stress at the cathode or 'up-wind' end of the line begins only after the electromigration of most of the copper from that end of the line. This implies that the stress will be generated at a rate dependent

upon the Cu removal rate. This observation that the rate of electromigration of Al is higher than Cu, once the Cu is sufficiently depleted, is in agreement with the experimental results of Hu *et al.* [Hu 92]. Therefore, failure in Al-Cu lines will be limited by the removal of copper at the sites of flux divergence. It is also noted that a significant build-up of a copper 'front' near the middle of the line is observed after about 200 hours of testing. This would presumably be observed as the formation of Al_2Cu precipitates, as the growth of pre-existing precipitates, or as a measurable increase in the concentration of Cu in solution. This prediction of the formation of an additional precipitate at the Cu front is also consistent with the observations of Hu *et al.*, [Hu 93] in which an Al_2Cu precipitate was observed to grow at a distance of only about 3 μm's from the cathode end of a 10 μm test stripe, in addition to the precipitate growth at the anode end.

Effects of varying copper-alloying additions were also investigated by simulation. A current density of 1.0 MA/cm^2 was simulated in interconnects of length 200 μm with uniform microstructure and composition varying from pure Al to Al-1.2at%Cu. A deterministic time-to-failure was determined for each interconnect assuming a critical stress for failure of 400 MPa. Figure 2-3 shows the time-to-failure results for each of these interconnects. Pure Al interconnects failed at very early times, and larger Cu alloying additions were shown to improve lifetime. In particular, an 0.90at% Cu alloying addition increased lifetime over pure Al by a factor of 27x. This is consistent with results of Hu *et al.*, who observed that a similar alloying addition increased lifetime by a factor of 25x [Hu 93]. Finally, it is noted that although the effect of Cu-alloying additions appears to level off above 1.0 at%Cu, the composition of Cu in Al will rarely exceed 1.0 at% in experiment since Cu will precipitate out of solution into Al_2Cu precipitates first.

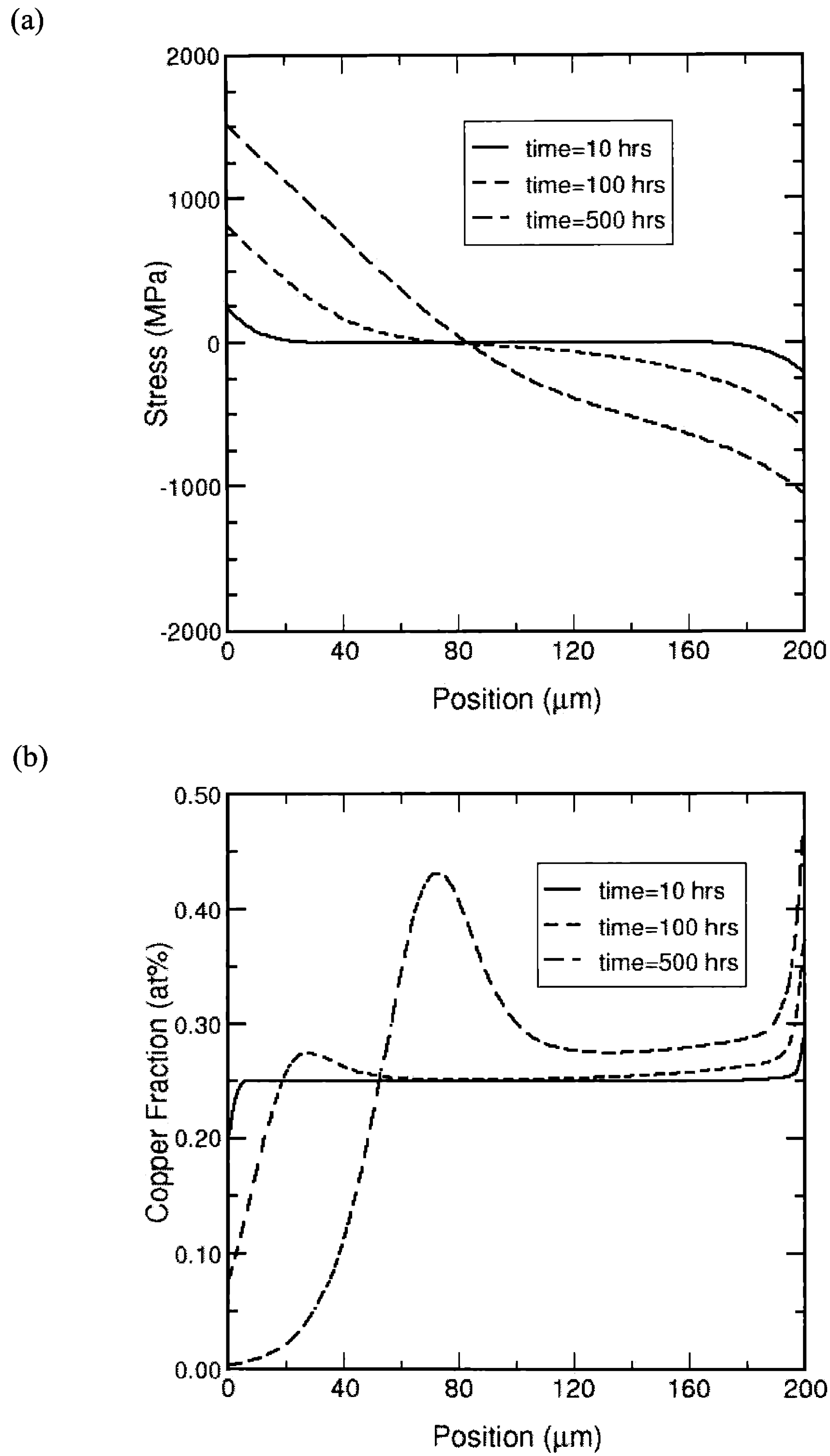


Figure 2-2. Stress (a) and copper migration (b) profiles for an Al-0.25% stud-to-stud line with all-bamboo microstructure at 1 MA/cm^2.

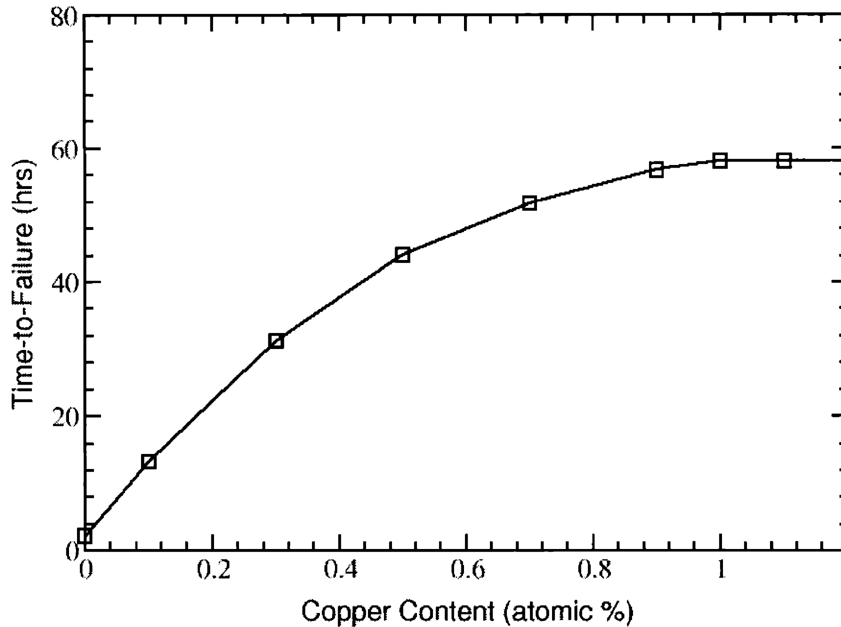


Figure 2-3. Time-to-failure results for a 200 μm interconnect with varying Cu alloying content. Increasing Cu contents results in improved lifetimes.

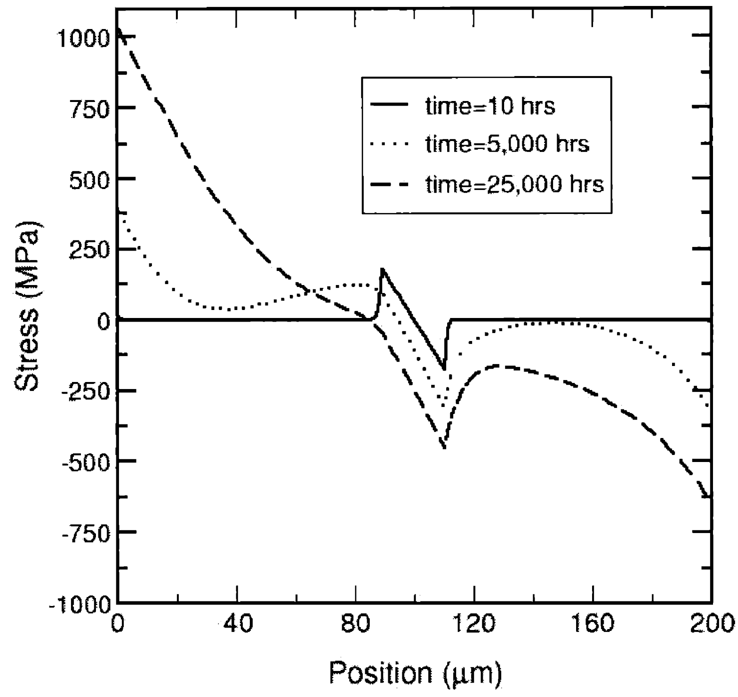
2.4. Copper Effects in Near-Bamboo Interconnects:

The effects of copper on a 200 μm stud-to-stud Al-0.25%Cu structure with a centered 21 μm-long polygranular cluster were simulated at a current density of 1 MA/cm^2. Figure 2-4 shows a series of stress and copper-concentration profiles for different simulation times. At early times, Cu electromigrates from the upwind end of polygranular cluster to the downwind end. The stress profile follows a similar evolution, since, again, the Al rapidly electromigrates only after the Cu is sufficiently depleted. This motion of the Cu is consistent with the report of Kim *et al.* [Kim 93], who observed a sweeping of Cu and Al_2Cu precipitates from the polygranular regions and the formation of large Cu-rich precipitates at the downwind end of polygranular clusters.

At intermediate times, Cu depletion (and development of a significant tensile stress) begins to occur in the bamboo region at the cathode, while the tensile stress at the upwind edge of the polygranular cluster begins to decrease due to transport in the anode-end bamboo region. However, at the downwind edge of the cluster, the compressive stress continues to build due to the reduced diffusivity resulting from the compressive stress. The copper content at this edge is also shown to rise significantly because of the effect of stress on the diffusivity. At later times, the bamboo region dominates in determining the stress profile and a significant copper depletion region is observed at the cathode.

As in the case when no alloy additions are present, the most rapid stress evolution is associated with the polygranular cluster. However, a quasi-steady state stress profile develops within the cluster, with corresponding maximum and minimum stresses which are a function of the cluster length and the current density [Kno 97]. Only after the longer times associated with slower transgranular (bamboo) diffusion, are higher stresses reached, in this case, at the zero flux ends of the lines. The effect of Cu then is to govern the rate of this evolution, without significantly changing the basic phenomenology. These results suggest though that Cu accumulation (most likely in the form of precipitates) can occur either at the Cu 'front', which is observed with or without microstructural heterogeneities, or at the downwind end of a polygranular cluster, even when the void nucleation site is not associated with that cluster. This behavior is consistent with the results of Hu *et al.* [Hu 93] and Kim *et al.* [Kim 93]. It should also be noted that precipitates could act as Cu sources, so that the presence of precipitates initially located at the cathode stud or the upwind end of the polygranular cluster, would lead to a significant increase in the lifetime.

(a)



(b)

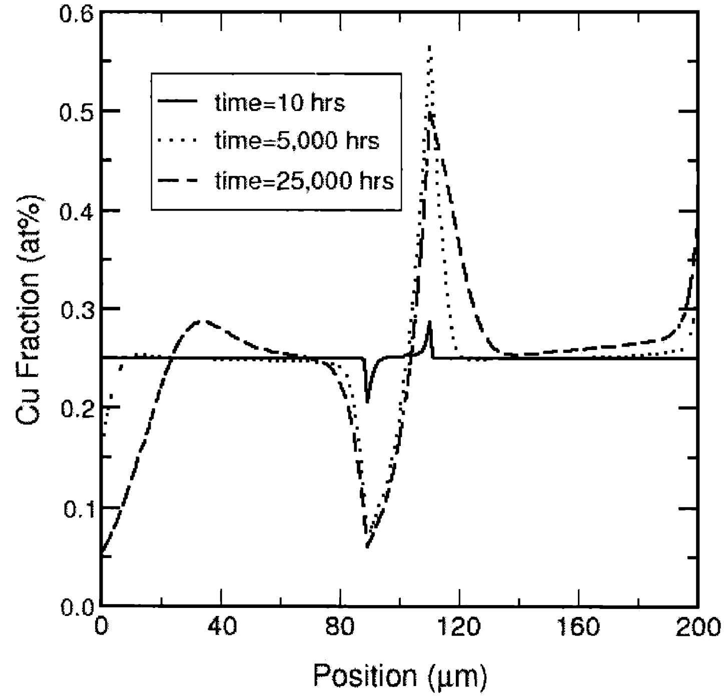


Figure 2-4. Stress (a) and copper migration (b) profiles for Al-0.25%Cu stud-to-stud lines with a 21 um polygranular cluster at the center of the line.

2.5. Summary

An electromigration simulation technique has been described and the technique has been embodied in a new software tool called MIT/EmSim, a version of which can be used over the web. [Ems 96] Extension of this simulation approach to account for the effects of alloy additions was described in detail. The evolution of the mechanical stress along the length of an interconnect is determined by simulating and tracking atomic fluxes due to electron wind forces, and back fluxes due to stress gradients. Atomic fluxes in alloys are simulated by including the effects of chemical concentration gradients on the driving force for diffusion and by including a solute-vacancy trapping model to account for kinetic effects. In simulating electromigration-induced failure, it was assumed that there is a critical stress required for nucleation of voids, and voids were allowed to grow in order to simulate failure in lines in which there are current shunts around voids. Failure can be defined as the time for void nucleation, or as the time required to reach a void length (or volume) which corresponds to an unacceptably high resistance increase.

Simulations of Al-Cu stud-to-stud lines with all-bamboo microstructures revealed that the evolution of stress in the line was limited by the removal of copper, and suggested that there should be an accumulation of Cu at a significant distance ahead of the site for void nucleation, but not necessarily at the anode stud. Increases in Cu alloying additions also resulted in improved interconnect lifetimes. Simulations of near-bamboo stud-to-stud lines indicated a similar dependence of the evolution of stress on the removal rate of Cu and suggested that downwind ends of polycrystalline clusters should be sites for Cu accumulation, even if the site of void nucleation is not associated with the cluster.

Chapter 3

Void Nucleation & Growth in Long Lines

3.1. Introduction

Stress evolution caused by electromigration in pure or alloyed interconnects leads to a process of void nucleation and growth. If the tensile stress in the interconnect reaches some critical stress (σ_{crit}), a void will nucleate and grow. A number of studies have been conducted to examine void nucleation and growth behavior through experiments [Rie 96, Arz 94, Mar 95, Mar 95] and simulations [Bor 92, Kra95]. Depending on the nature of the specific interconnect materials, several different modes of failure in interconnects may result. In lines which do not have refractory layers (underlayers and ARC layers, or liners in Cu-based interconnects), once a void nucleates, it can quickly grow to span the width of the line and cause a void-nucleation-limited open circuit failure. In lines with refractory electromigration-resistant layers, current can shunt around voids, so that voids can span the width of the lines and continue to grow for long times after nucleation. In this case, failure is defined as the time at which voiding has lead to unacceptably high line resistances. If the time for growth of voids is long compared to the time required to reach the void nucleation condition, the failure processes is void-growth limited. Lines which have been passivated by application of a surrounding rigid dielectric are subject to different mechanical constraints than unpassivated lines. Significant compressive stresses may build until the dielectric fractures (dielectric failure), leading to rapid extrusion of metal [Llo 83], and consequently to rapid nucleation and growth of fatal voids. In this case, failure is limited

by dielectric failure. Whether failure is nucleation-limited, growth-limited or limited by the mechanical properties of the dielectric, the time to failure is affected by the atomic scale diffusive processes which govern the evolution of stress along an interconnect. Modeling of the temporal and spatial evolution of stress throughout the failure process in both pure and alloyed systems therefore provides the basis for failure modeling and reliability simulations, and is the focus of this chapter.

3.2. Void Model Formulation

In further developing the MIT/EmSim model for the void nucleation and growth process, we have assumed zero flux boundary conditions, as would apply in the stud-to-stud test line illustrated in Figure 1-5. For Al-based interconnects, the W filling the vias at either end of the interconnect acts as a diffusion barrier. Similarly, for Cu-based lines, Ta and Si_3N_4 act as diffusion barriers at the studs. Lines are also assumed to have refractory layers which can shunt current around voids.

During interconnect reliability testing, tensile stresses will develop at the up wind end of polygranular clusters, and at the stud at the cathode end of the line. As in previous work [Kno 97], we assume that voids will nucleate once a critical tensile stress is reached anywhere along a line. If the microstructure is homogeneous (all polycrystalline or all bamboo), and if the critical stress required for nucleation of voids is constant along the line length, void nucleation will always occur at the cathode stud. In our one-dimensional simulation, the void is assumed to span the width and thickness of the interconnect. Since the total number of atoms is conserved, the initial void size at nucleation can be assumed to be equal to the volume necessary to relax the stress some small but finite distance from the void (defined as the stress relaxation zone). The stress relaxation zone length will be defined as L_{relax} (see Figure 3-1).

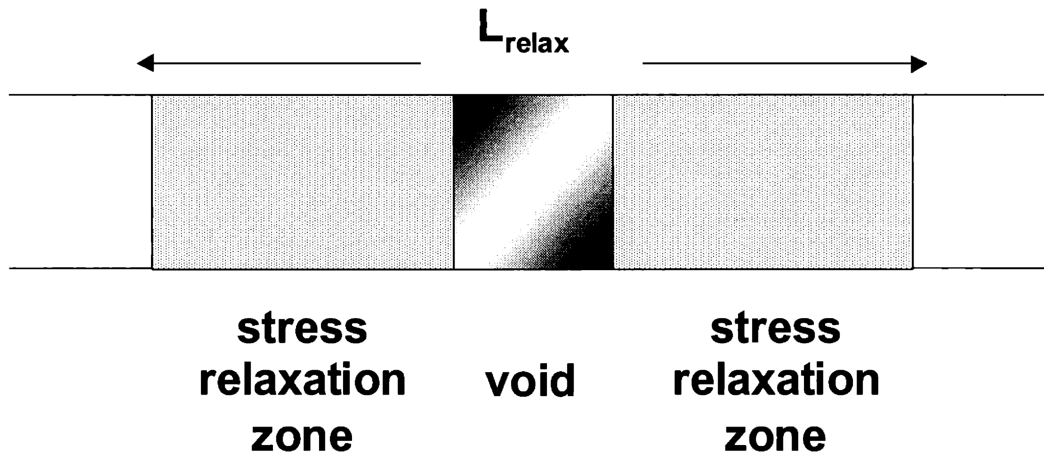


Figure 3-1. The nucleation of a void results in the formation of a stress relaxation zone adjacent to the void.

The initial void length (L^{o}_{void}) is then given by

$$L^{0}_{void} = L_{relax} - \frac{(N_{Al} + N_{Cu})\Omega}{Wh}, \tag{3.1}$$

where N_{Al} and N_{Cu} are the number of Al and Cu atoms located within the relaxation zone, and where W is the linewidth and h is the line thickness.

Once the void has nucleated, the void size will change when there is a net atomic flux in or out of the void. A net atomic flux out of the void results in void growth, and a net atomic flux into the void results in shrinkage. A flux into the void when coupled with a flux out of the void, will result in void motion, even when there is no net change in the void size. The void size is adjusted such that the length of the stress relaxation zone on either side of the void is constant as it changes size and/or moves. In keeping with the one-dimensional representation of the interconnect, the enlargement or shrinkage of the void (Δl_{void}) is assumed to follow a mass transport model and may be related to the flux by the following

$$\Delta l_{void} = \frac{J\Omega}{WT} \cdot \Delta t, \tag{3.2}$$

where Δt is the time step. Previous models for void growth include the integration method, in which the void size was calculated by integrating the stress throughout the entire line [Kor_2 93]. However, it is unrealistic to assume that void growth is affected by distant stresses. A fundamental assumption of the mass transport model presented here is that only local stresses and fluxes drive void growth. In all the cases reported below, we have also assumed that the test was conducted at 200°C with lines taken to have heights and widths of 1 μm.

3.3. Void Nucleation and Growth in Long Lines

Electromigration in an all-bamboo Al lines with studs at both ends was simulated to observe the effects of void nucleation and growth. A critical tensile stress of 400 MPa was assumed for void nucleation, and a current density of 1.0 MA/cm^2 was applied. At early times, the stress profile appears similar to that of the stud-to-stud simulations described earlier (compare Figure 1-6 with the schematic diagram of Figure 3-2a). However, once the upwind end of the line reaches the critical tensile stress for void nucleation, a void is nucleated and the stress in the vicinity of the void rapidly drops to near zero as the stress in the nearby region is relaxed (Fig. 3-2b). As time goes on, the void grows in size and the region of tensile stress completely vanishes due to stress relaxation through transport of vacancies to the growing void. As electromigration continues, a stress gradient develops ahead of the growing void, changing from zero stress near the void, to the maximum compressive stress at the downwind end of the line. In the case of long lines, in which the void does not interact with the back-stress, void growth would continue until a resistance failure or dielectric failure occurred. The effects

of this complex interaction on interconnect reliability will be investigated in much further detail in Chapters 5 and 6.

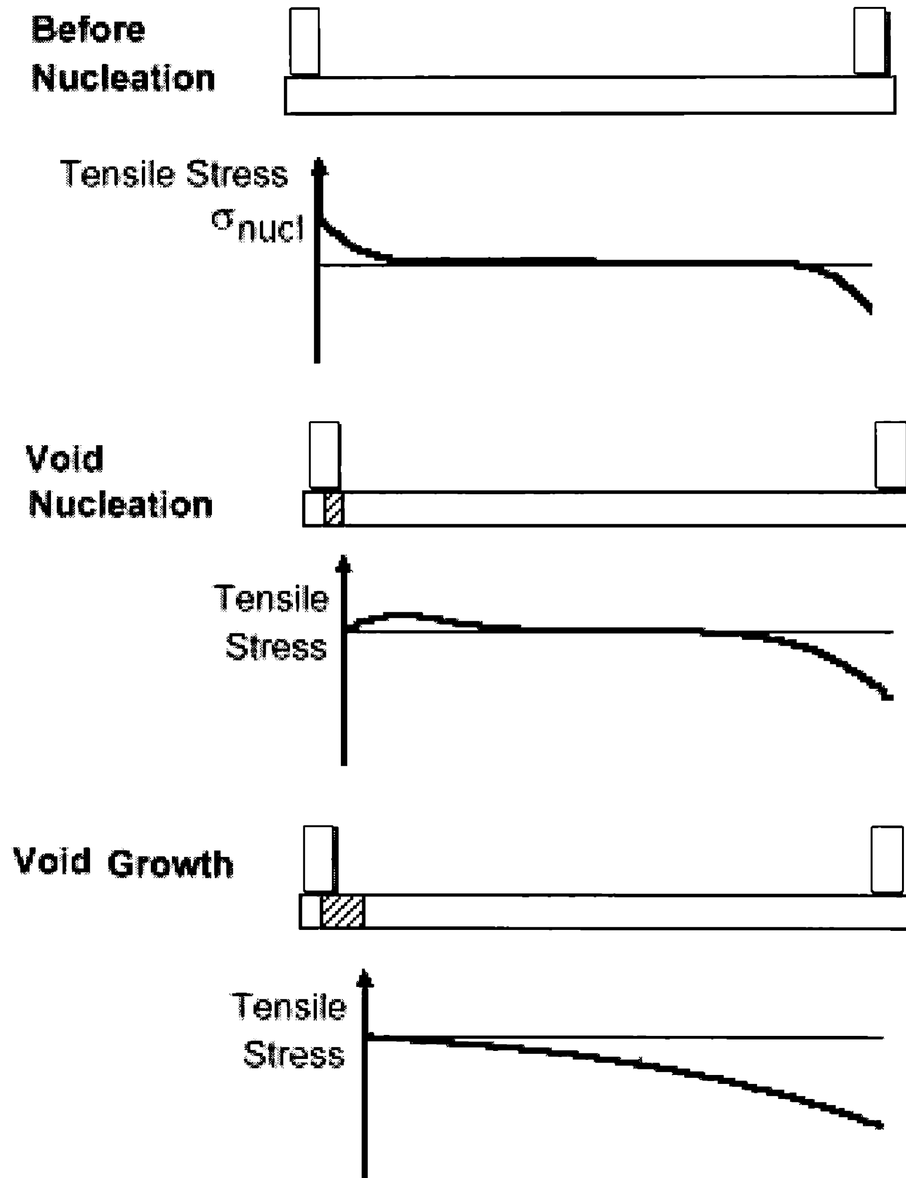


Figure 3-2. Stress evolution until void nucleation, followed by growth until steady-state.

3.4. Current Density Scaling Laws:

Experimental characterization of interconnect reliability is carried out through accelerated tests on populations of lines to cause failure to occur in a reasonable period of time (typically under a thousand hours), since electromigration is a slow and stochastic process. Current density and temperature are used as acceleration factors, and test data must be scaled to service temperature and current densities, and Black's equation [Bla 69] is almost universally used to scale accelerated test data to service conditions. This equation relates the median time-to-failure (t_{50}) of a population of interconnects to the current density and temperature (restated from Eqn. 1.6)

$$t_{50} = Aj^{-n} \exp\left(\frac{Q}{kT}\right), \tag{3.3}$$

where A is a current density and temperature independent constan, Q is the activation energy for the rate-limiting atomic-level process required for failure, usually associated with diffusion, and the current density exponent, n, which is generally taken to be 2. It has been shown that an exponent of 2 is consistent with nucleation-limited failures in models based on the Korhonen analysis [Kor 93]. The rate of unconstrained void growth should be proportional to j, so that the current density exponent for void-growth-limited failure is expected to be 1 [Kir 91]. Oates has observed current density exponents close to n=1 in Al lines with a TiN shunt layer [Oat 95]. However, current density exponents of 2 have been found in experiments on stud-to stud structures which might also have been expected to fail through void-growth-limited mechanisms [Ata 94]. It has been suggested that Cu alloy additions may be responsible for observations of n=2 in stud-to-stud structures [Llo 96]. Knowledge of the appropriate value of n is critical in scaling test results to service conditions, since in changing n from 1 to 2, the projected in-service reliability can change by orders of magnitude. Also, when Black's equation is used to extrapolate test results to service conditions, it is implicitly assumed that the failure

mechanism does not change through the range of extrapolation, and it is essential to know if this condition is met.

We have used the simulation described above to determine the current density scaling laws for void-nucleation-limited and void-growth-limited failure in pure Al and Cu-doped Al lines with uniform microstructures. These simulations were performed using long lines (5,000 μm), to prevent the compressive stress region at the anode end of the line from interacting with the tensile stress region or the growing void at the cathode end of the line. Stress-dependent diffusivities were used in this set of simulation runs. Void growth saturation in shorter lines will be treated in the next chapter.

To determine the current-density-dependence of void-nucleation-limited failure, electromigration in an Al stud-to-stud line, electromigration in lines with all-bamboo structure was simulated assuming a void nucleation stress of 400 MPa. As shown in Figure 3-3, a current density exponent of 2 was observed for void-nucleation-limited failure for both pure Al and Al-1.0%Cu. Similar simulations for Al-1.0%Cu resulted in a time-to-failure increase of a factor of 27, and yielded a current density exponent of 2. This set of runs was repeated neglecting the stress effect on diffusivity, and the same current density exponents were observed.

In the investigation of the current density exponent for void-growth-limited failures, it was assumed that a small void already existed at the cathode end of the line. Void growth was observed until the void reached a critical void length of 0.15 μm. A current density exponent of 1 was found for both pure Al and Cu-doped Al (see Figure 3-3). The time to void-growth-limited failure of Al-1.0%Cu lines was found to be a factor of 27 times higher than that of pure Al lines. Since the void was already pre-existing at the upwind stud, the times-to-failure for the Al-Cu lines was increased over those of Al lines by the difference in diffusivity at the test conditions. It is also important to note that no Cu front is formed near the upwind stud since the flux is constant along

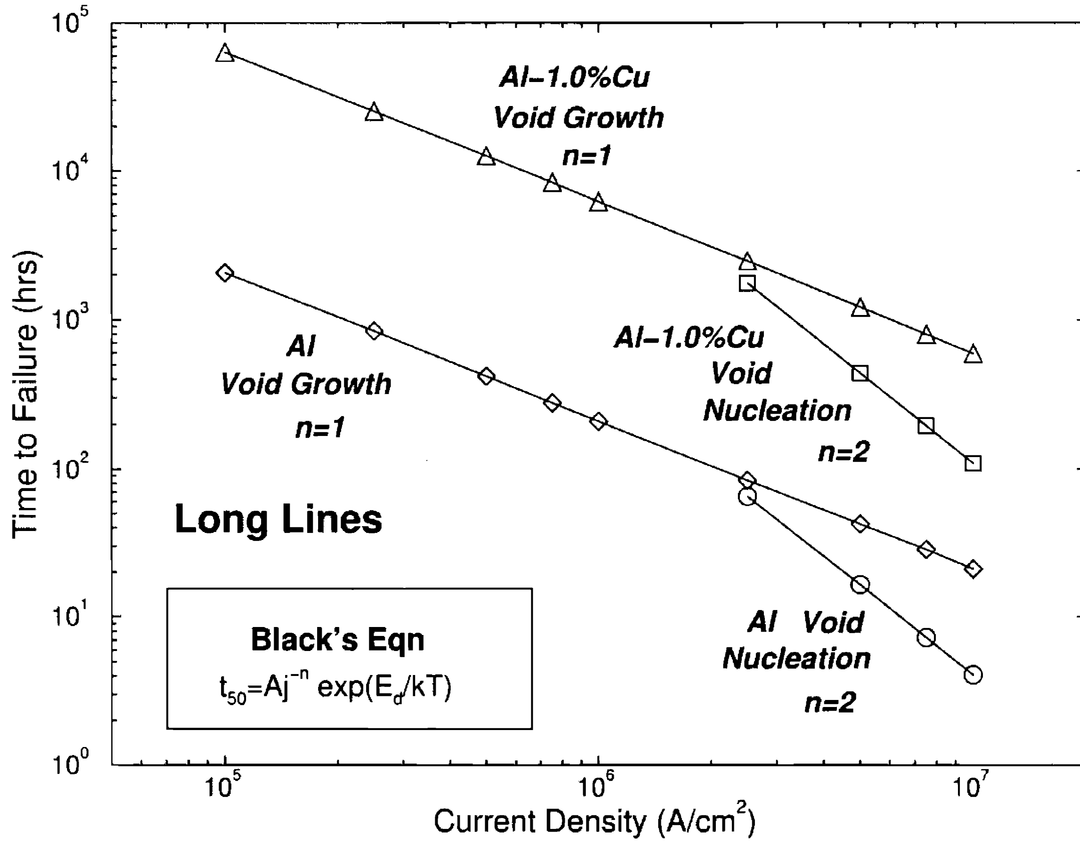


Figure 3-3. The current density for void nucleation failure in Al and Al-Cu lines is n=2. The exponent for void growth failure is n=1 in Al and Al-Cu lines.

the entire line; there are no flux divergences except at the downwind stud. Repeating these runs without the stress effect on diffusivity included, resulted in a slight reduction in failure times but revealed no change in the current density exponent.

Finally, a set of 5,000 μm all-bamboo lines without pre-existing voids was tested to failure assuming a 0.15 μm failure void length. For Al lines, a failure mechanism transition was observed for both pure Al and Cu-alloyed interconnects (Figure 3-4). At high current densities, an n=1 exponent is observed. This is readily explained since at high current densities, the tensile stress quickly reaches the critical value for nucleation,

and most of the time-to-failure(ttf) is consumed during void growth, resulting in void-growth-limited failures. At low current densities, an n=2 exponent is observed since most of the ttf is the time required for void nucleation. Once the void nucleates, the adjacent tensile peak that remains after the void nucleates greatly aids void growth toward failure, resulting in void-nucleation-limited failures. Black's equation must therefore be used prudently, so as to ensure that the failure mechanism does not change from void-growth limited to void-nucleation limited in scaling from accelerated test conditions to field conditions. It is also observed that no significant change in the transition current density is apparent with Cu-alloying additions. In addition, as the void failure length is decreased to 0.04 μm, no significant change in the transition current density is observed.

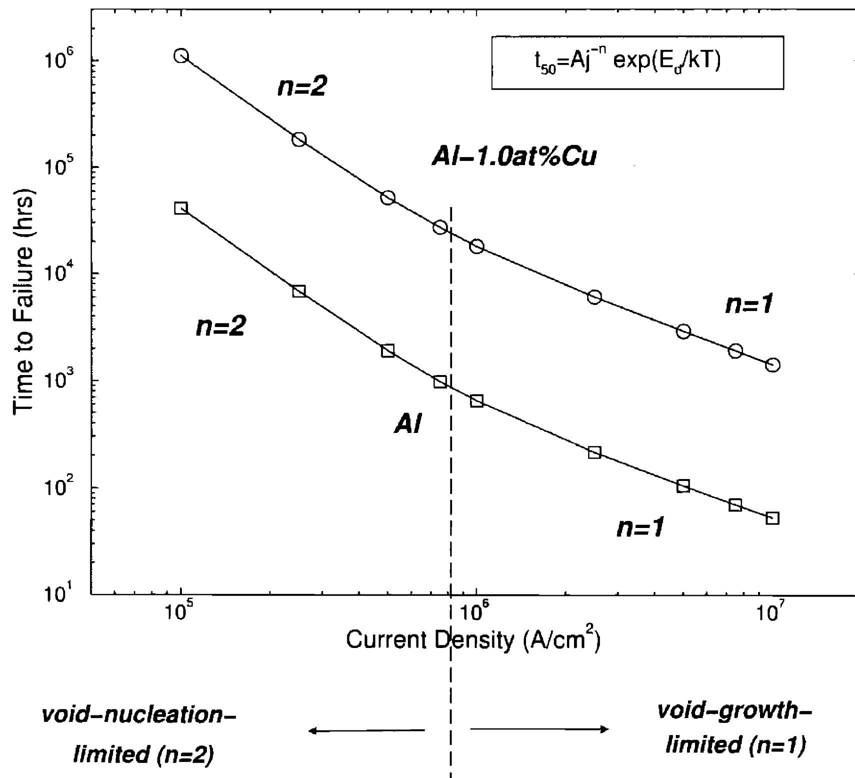


Figure 3-4. At high j, void-growth-limited failure is observed, resulting in an n=1 scaling exponent. At low j, failure is nucleation-limited, requiring an n=2 exponent.

The times to void-nucleation-limited failures are therefore seen to scale with the current density with an exponent of n=2 while void-growth-limited failure times scale j with an exponent n=1. It was observed that the current density exponent changes from n=1 to n=2 when scaling from accelerated test conditions to field conditions due to a change in the failure mechanism. Because this could lead to errors in the estimation of interconnect reliability by several orders of magnitude, care must be taken to verify that Black's equation applies, and if it does not, alternative scaling procedures, e.g., based on simulations, must be used. It was also observed that the current density at which this failure mechanism transition occurs was not significantly affected by Cu alloying additions or a change in the void length taken to lead to failure.

3.5. Summary

Void nucleation and growth was simulated for stud-to-stud pure Al and Al-Cu lines with all-bamboo microstructures. The nucleation of a void was shown to rapidly relax all of the tensile stress in its vicinity. Void growth would then continue until the back-stress due to the stress gradient associated with the compressive peak at the downwind end of the line compensated the electromigration driving force. Current density scaling laws for failure times were investigated for electromigration in longer lines. For void-nucleation-limited failure, a current density exponent of 2 was observed for both Al and Al-1%Cu lines. For void-growth-limited failure with a pre-existing void at the cathode end, a current density exponent of 1 was observed for both Al and Al-1%Cu lines. For resistance failures without a pre-existing void, void-nucleation-limited failures are observed at low current densities and void-growth-limited failures are observed at higher current densities. This demonstrates that current density scaling of failure times based on the use of Black's equation must be performed with care since this equation is valid only when the dominant failure mode is constant throughout the range of extrapolation. Neither the addition of Cu nor a change in the void failure length leads

to a change in the rate-limiting failure mechanism or the appropriate current density scaling relationship.

Finally, it is noted that the simulation techniques demonstrated here with simulations for Al and Al-Cu alloys can be applied to other materials as well. It is expected that the basic phenomenology described here for Al and Al-Cu will also occur for pure Cu and Cu-alloy interconnects, and is the subject of Chapter 7. Materials-dependent parameters such as the diffusivity in bamboo and polygranular regions and the mechanical properties will change, resulting in changes in the absolute interconnect failure times. However, the general current density scaling behavior described here should be preserved.

Chapter 4

Void Nucleation & Growth in Short Lines

4.1. Introduction

While the previous chapter investigated the void nucleation and growth process in long interconnects, this chapter will focus on the more complex behavior of voids in short lines. In short lines, after a void nucleates (Fig. 4.1a), the stress near the void drops to zero and the void begins to grow (see Fig. 4-1b) as described in Section 3.2. In short lines, however, after some time, the back-stress will extend back to the cathode, interacting with the void and decreasing the electromigration force (Fig. 4-1c). As the back-stress near the void becomes larger and larger, the electromigration wind near the void continues to decrease, slowing void growth. After a sufficiently long period of time, void growth will cease, resulting in void growth saturation. The void will be at a steady-state, and the stresses in the interconnect will no longer evolve. Behavior of this sort has been demonstrated by Filippi *et al.* [Fil 96] for stud-to-stud lines with shunt layers.

An additional complexity arising in the void nucleation and growth behavior of short lines occurs when relatively low current densities are applied, resulting in the interaction of the tensile peak with the compressive stress well before void nucleation. This leads to the possibility that the compressive stress will interact with the void immediately after nucleation, significantly increasing both the time for void nucleation and the time for void growth to failure. A second complication in short lines, as discussed in Section 1.4, occurs if the maximum stress in a line is lower than the critical

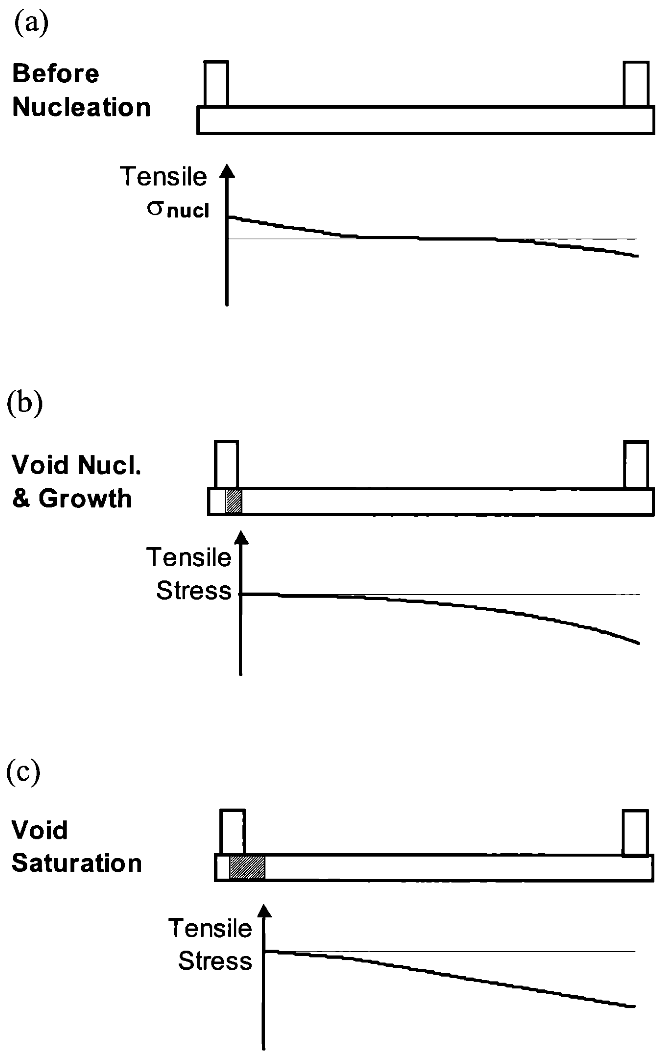


Figure 4-1. Stress evolution (a) until void nucleation (b) followed by void growth until (c) a steady-state stress gradient develops.

stress for void nucleation, in which case the interconnect will not fail and will be 'immortal'.

In summary, interconnects can evolve to one of the following final states: immortality (either due to absence of void nucleation, or resistance saturation), failure at

the cathode due to void nucleation and growth, and compressive-stress failure at the anode. The void nucleation and growth process in short lines is observed to be markedly different from that of long lines, and requires further study to characterize the complex reliability behavior of short lines. In this chapter, the resistance saturation and current density scaling behavior of short lines will be investigated using the MIT/EmSim model as detailed in Chapters 2 and 3.

4.2. Resistance Saturation & Immortality

In chip design, it is customary to characterize failure based on an unacceptable increase in resistance (as opposed to void size, or tensile stress, or some other metric). If it is assumed that current can shunt around the void, the void length changes with time. Because the shunt layers are of higher resistivity, as a void nucleates and begins to grow, the resistance will also increase. For short lines, as void growth saturates to a volume V_{ail},, the resistance, too, will stop increasing, resulting in resistance saturation. The saturation void volume V_{sat} is given by [Fil 96, Kor_2 93, Suo 98, And_2 99]:

$$V_{sat} = \frac{e\rho z * jL^2}{2\Omega b}, \tag{4.1}$$

where each variable was defined in Chapter 1. The resistance failure criterion corresponds to a void volume corresponding to failure, V_{fail}. If V_{sat} exceeds V_{fail} , the line will also be immortal. The size of the void at saturation L_{sat} will be larger for longer lines, and the corresponding saturation resistance larger. The saturation resistance will also depend on the liner resistivity and the void shape and location. The failure time can be defined as the time required to reach a given resistance R_{fail}, corresponding to a void volume V_{fail} or length L_{fail}. Depending on the resistance failure criterion, a line will be immortal if the critical failure resistance is greater than the resistance at saturation. Korhonen *et al.* have determined an analytic expression for determining the size of the

void at saturation in clusters [Kor_2 93]. Suo also derived an expression for approximating the increase in resistance at void saturation by studying the effects of the back-stress on the electromigration driving force [Suo 98]. A more precise expression for resistance saturation as a function of void length is derived in Appendix C.1 for two different failure criteria.

Given information about the resistivity of the shunt layers, and the geometry of the void, such data can be converted to give the line resistance as a function of time. To calculate the resistance of a line, an interconnect is modeled as a resistor network. In the regions to the left and right of the void, three resistors in parallel are assumed for the over-layer, line, and under-layer. For the regions at the void, two resistors in parallel are assumed for the over-layer and under-layer. The three regions are assumed to be in series. The resistance R is related to the resistivity ρ by

$$R = \frac{\rho}{Wh} L \ , \tag{4.2}$$

where W is the linewidth and h is the line thickness. From basic circuit theory, an effective resistivity ρ_s can be calculated for the resistor networks corresponding to the shunt structure, assuming the total shunt thickness as the effective thickness of the shunt (t_s). For the shunt structure specified above, this results in an effective resistivity of 82 $\mu\Omega$-cm and effective thickness of 110 nm. The resistance for the entire interconnect, defined as R_o, can then be calculated based on the effective shunt layer resistivity and metal resistivity (ρ). In the presence of a void, the normalized increase in resistance, $\Delta R/R_o$, of the line can be calculated as:

$$\frac{\Delta R}{R_o} = \frac{L_v \rho_s h}{L \rho t_s} \ , \tag{4.3}$$

where L_v is the void length. This expression can be used to determine the critical void size that results in failure according to the resistance failure criteria. Equation (4.3) may be rearranged as a function of the critical void length L_v^c to give

$$L_v^c = \frac{\rho\, t_s}{\rho_s h} \frac{\Delta R_f}{R_o} L \ , \qquad (4.4)$$

where ΔR_f is the resistance change defined as corresponding to failure. From Eq. (4.4), it is apparent that the critical void size is a function of three terms: a materials term M_c given by $\rho t_s/\rho_s h$, a failure-condition term ΔR_f, and a geometrical term L. As expected, an increase in the line length or resistance failure criteria will result in a (linear) increase in the critical void size for failure. For the shunt structure specified above, the materials term is equal to 0.023.

Electromigration was simulated in a 200 μm Al interconnect of uniform microstructure to observe void nucleation and void growth to saturation. Using the materials parameters described above, a resistance profile through time was determined (see Fig. 4-2). From this figure, it is apparent that the selection of the failure resistance will determine the lifetime of the interconnect, or whether the line is immortal. If the line fails, as in Fig. 4-2, the failure may be either void-nucleation-limited or void-growth-limited. If L_{sat} is small compared to L_{fail}, such that the time to failure is largely composed of the time required to reach the critical stress for void nucleation, failure is nucleation-limited. On the other hand, if L_{fail} is small compared to L_{sat}, and most of the time to failure occurs after void nucleation, failure is void-growth limited. The dominating failure mechanism may also be determined from Figure 4-2 as the greater of either the incubation time (t_N) or the void growth time (t_G).

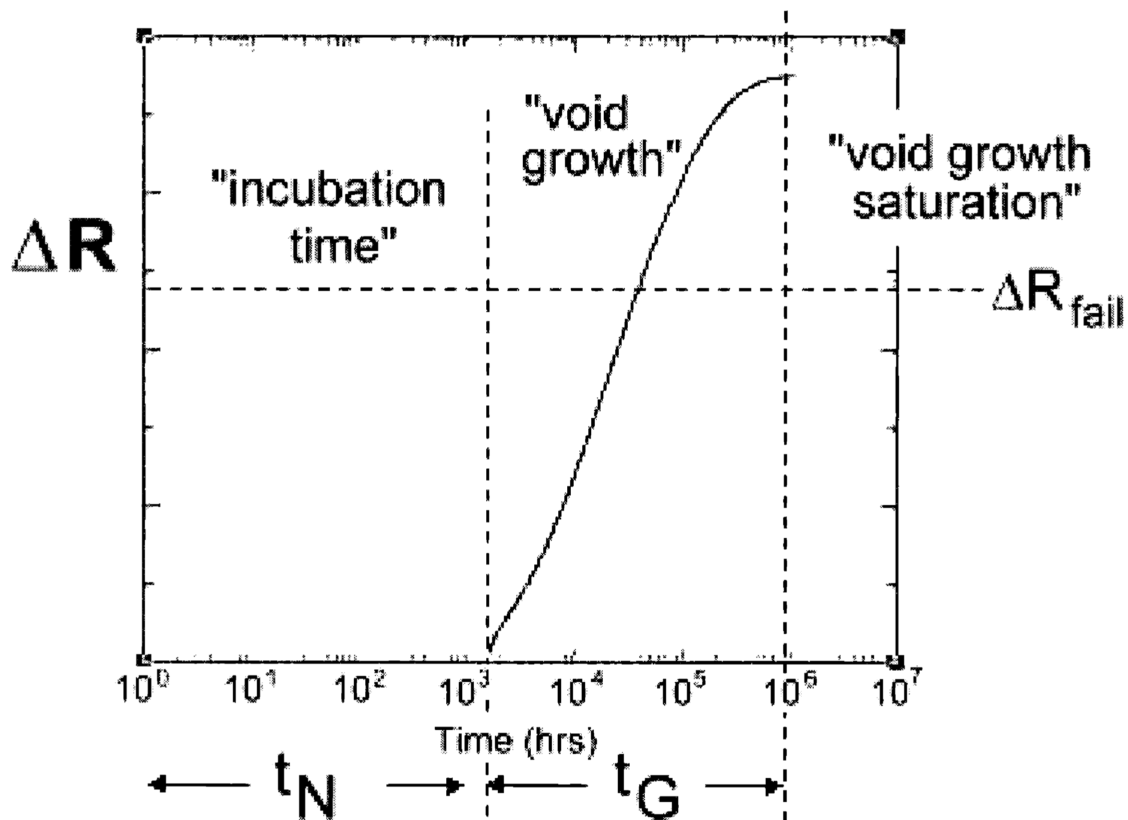


Figure 4-2. After the incubation time, the void nucleates causing an increase in the line resistance. The void continues growing until void saturation is reached, resulting in no further increase in resistance.

In summary, failures in short lines may be void-nucleation-limited or void-growth-limited, and this may readily determined by comparing the incubation time and growth time. The relation of these failure mechanisms to the current density exponent will now be discussed.

4.3. Current Density Scaling in Short Lines

As described in Chapter 3, there has been disagreement in the literature regarding the proper value of the current density exponent n for scaling of data for lines terminating in diffusion barriers using Black's equation [Bla 69]. A wide disparity of exponents have

been observed, attributed to either the failure mechanism (void-nucleation-limited or void-growth-limited), alloy composition, and joule heating. It was demonstrated in Section 3.4 that in long lines in which the compressive peak does not interact with the void or tensile peak, the current density exponent has been shown by simulations to be n=2 for void nucleation failures and n=1 for void growth failures (with a pre-existing void) for both pure metal and alloy interconnects. More importantly, the current density exponent was shown to increase from n=1 at accelerated conditions to n=2 at service conditions when failure is defined by void nucleation *and* growth to a specific void volume [Par 99]. In short lines, however, in which the compressive peak may interact with the evolving tensile stress to affect the rate of void growth, the current density scaling is more complex, and will be investigated in this section..

Electromigration in a pure Al 200 μm stud-to-stud, all-bamboo line without pre-existing voids was simulated assuming a critical void nucleation stress of 400 MPa and critical failure void lengths of 0.01, 0.05, and 0.10 μm (the voids are assumed to span the line). When the failure criterion was defined as a void length of 0.05 μm (L_v^c/L=0.025%), a current-density exponent of unity was observed at high current densities (see Fig. 4-3). This is readily explained, since at high current density, the tensile stress quickly reaches the critical value for nucleation, and most of the time-to-failure (ttf) is the time for void growth, resulting in void-growth-limited failures. As the current density is lowered, the exponent is observed to increase, maintaining a constant value of two between current densities ranging from 0.6 to 1.5 MA/cm^2. This increase in the exponent is due to the transition from void-growth-limited failures to void-nucleation-failures as the void growth time, t_G, becomes lower than the time for void nucleation or incubation time, t_N. In this case, once the void nucleates, the tensile stress in the region adjacent to the void greatly aids void growth to failure, resulting in void-nucleation-limited failures. As the current density is further lowered, a sharp increase in the ttf is observed due to the interaction of the compressive peak with the tensile peak, resulting in dramatically slowed stress and void evolution. Ultimately, at low enough current densities, resistance

saturation results in line immortality. If the maximum stress at steady-state is lower than the critical tensile stress for void nucleation, the line is also immortal.

For simulations involving a more conservative failure criterion, with a void failure length of 0.01 μm (with L_v^c/L=0.005%), the void-nucleation-limited regime is observed to span a much greater range of current densities, extending from approximately 0.3 to 7 MA/cm^2. Changing the void failure criterion to 0.10 μm (with L_v^c/L=0.05%), the void-growth-limited regime extends over a broader range, from 2.5 MA/cm^2 and above. In addition, there is no clearly distinguishable n = 2 region.

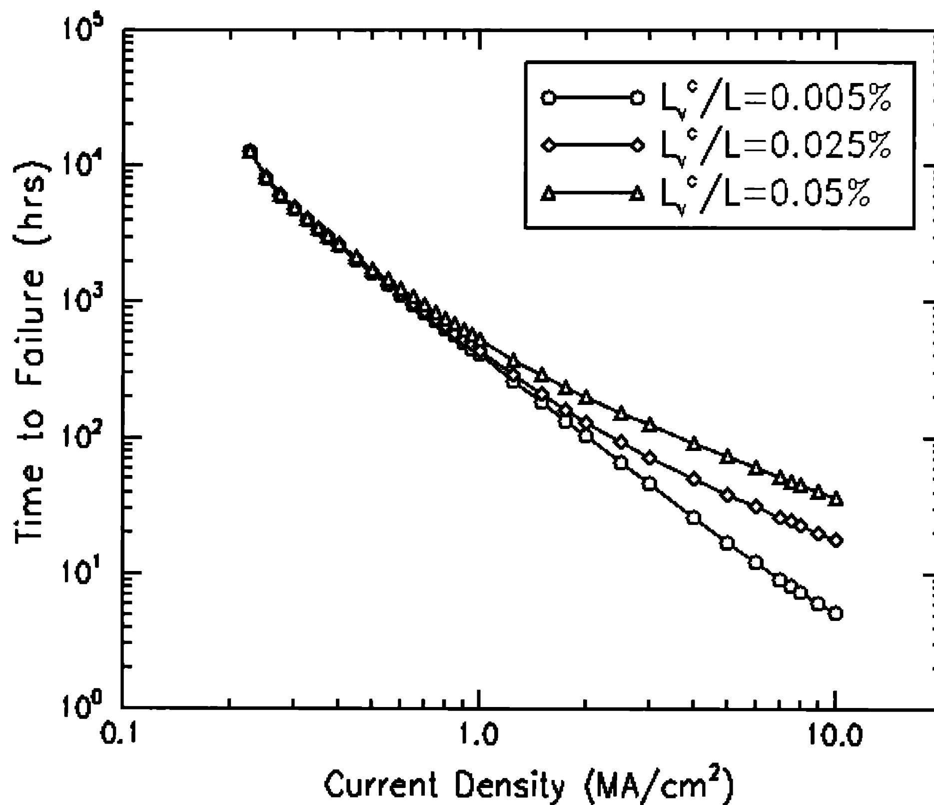


Figure 4-3. Times-to-failure as a function of current density for several void length failure conditions.

In summary, failure is void-nucleation-limited (n=2) for conservative void failure criteria in the absence of back-stress effects. Failure is void-growth-limited for relaxed void failure criteria, fitting a j^{-1} current density dependence when back-stresses do not interact with the tensile peak and growing void. When the back-stress interacts with a void in lines with void-growth-limited failure, the exponent is observed to increase above unity. Similarly, when failure is void-nucleation-limited, an exponent of $n \geq 2$ is observed. The void failure length criteria of 0.01, 0.05, and 0.10 μm correspond to resistance failures of 0.217%, 1.08%, and 2.17% for the specified shunt structure. It is therefore apparent that the lifetime data reported using more relaxed failure criteria will tend to be void-growth-limited and likely dominated by back-stress effects, yielding an exponent of $n>1$. Only lifetime tests conducted at sufficiently high current densities under these failure conditions should yield an exact n=1 dependence. The transition current density (j_{tr}), which defines the lowest current density limit at which an n=1 exponent is apparent, was observed to increase with the selection of more conservative failure criteria. Since the dominant failure mode of the lines changes with current density, the current density exponent in Black's equation also changes depending on the current density. To accurately scale test results using Black's equation, not only must the failure mechanism remain the same, but also the back-stress must *not* interact with the tensile peak or void. A reliable and precise method for calculating j_{tr} is therefore desirable, and will be considered in the next chapter.

4.4. Summary

Void nucleation and growth was simulated for *short*, stud-to-stud lines with all-bamboo microstructures. The nucleation of a void was shown to rapidly relax all of the tensile stress in its vicinity. Because interconnects are surrounded by conducting shunt layers, current will continue to flow and the void will grow. Void growth then continues until the back-stress due to the stress gradient associated with the compressive peak at the downwind end of the line compensated the electromigration driving force, resulting in

void saturation. The higher resistivity of the shunt results in an increasing resistance as the void grows, ultimately reaching resistance saturation in short lines. If the saturation resistance is less than the failure resistance, the line is immortal. Conversely, if the resistance associated with void growth exceeds the failure resistance, the line is said to fail. Depending on the relative magnitude of the incubation time and growth time, the failure may be void-nucleation-limited or void-growth-limited.

Investigation of the current density exponent in short lines relating to Black's equation revealed a intricate dependence upon the line length, current density, and failure condition. In the absence of back-stress effects, failures at low current densities were void-nucleation-limited exhibiting $n=2$ scaling while at higher current densities, failures were void-growth-limited with $n=1$ scaling. In the presence of back-stress interactions, the exponent was shown to vary with $n>2$ and $n>1$ scaling respectively. The current density domains over which these dependencies were observed varied with failure condition. Only at the highest current densities was the exponent observed to remain constant with $n=1$ behavior. This suggests that the application of Black's equation to scale accelerated test data to service conditions is accurate in short lines only at higher current densities to ensure that the failure mechanism remains the same. A methodology to determine the current density satisfying this will be discussed in the next chapter.

Chapter 5

Failure Mechanism Maps

5.1. Introduction

In the preceding chapters, a variety of failure mechanisms and immortality modes have been discussed in detail. As a current is applied in a line, the back-stress may interact with the tensile stress leading to a steady-state, resulting in immortality without void nucleation (Chap. 1). When a higher current density is applied, a void will nucleate and grow. If the back-stress interacts with this void and suspends void growth, resistance saturation results. When the saturation resistance is below the failure resistance, immortality due to void saturation results. Increasing the current density further results in a resistance failure that may be void-nucleation-limited or void-growth-limited depending on the relative magnitudes of the incubation time and growth time. When the back-stress does not interact with the void, void-nucleation-limited failures are characterized by $n=2$ scaling in Black's equation while void-growth-limited failures exhibit $n=1$ scaling. When the back-stress interacts with the void, incubation time and growth time both increase, and the current density exponent becomes variable at $n>2$ and $n>1$ scaling respectively. Changes in line length and current density lead to changes in failure times as well as in failure modes. This drastically complicates efforts to accurately estimate reliability at service conditions based on accelerated data. Cataloging of simulation results allows assessment of the validity of Black's equation, allows more accurate scaling of test results to service conditions, and allows the layout of interconnects to take advantage of immortality or slower failure modes. One convenient method for such cataloging is the

construction of *failure mechanism maps*. Kraft *et al.* [Kra 98] have developed electromigration damage maps, focussing on defining domains for different mechanisms for diffusion in near-bamboo polycrystalline interconnects, an idea also explored by Thompson *et al.*[Tho 93] In this work, we demonstrate the means of construction of failure mechanism maps indicating domains for fundamentally differing failure mechanisms and lifetime scaling as a function of line length and current density. We also discuss how these maps may be used to assess interconnect reliability. Finally, an analytical means of constructing failure maps for simple geometries and compositions is discussed, and compared to simulation results.

5.2. Failure Mechanism Maps

It was demonstrated over the last few chapters that the failure mechanism of an interconnect is strongly dependent upon the current density. The failure mechanism is also strongly dependent on the line length, because the line length influences the time for interaction of the void or tensile stress with the compressive stress at the anode end of the line. Resistance saturation was also shown to be dependent upon both the line length and current density. A condensed overview of the significance of the current density and line length on the reliability of interconnects may be gained by developing a failure mechanism map. A failure map displays, as functions of line length and current-density, the regions of dominance of different failure modes: line immortality without void nucleation, immortality due to void saturation, impeded void-nucleation-limited failure ($n>2$), unimpeded void-nucleation-limited failure ($n=2$), impeded void-growth-limited failure ($n>1$), unimpeded void growth-limited failure ($n=1$), and anodic or dielectric failure due to compressive stress build-up [And 99]. A subset of these failure modes is illustrated in Fig. 5-1.

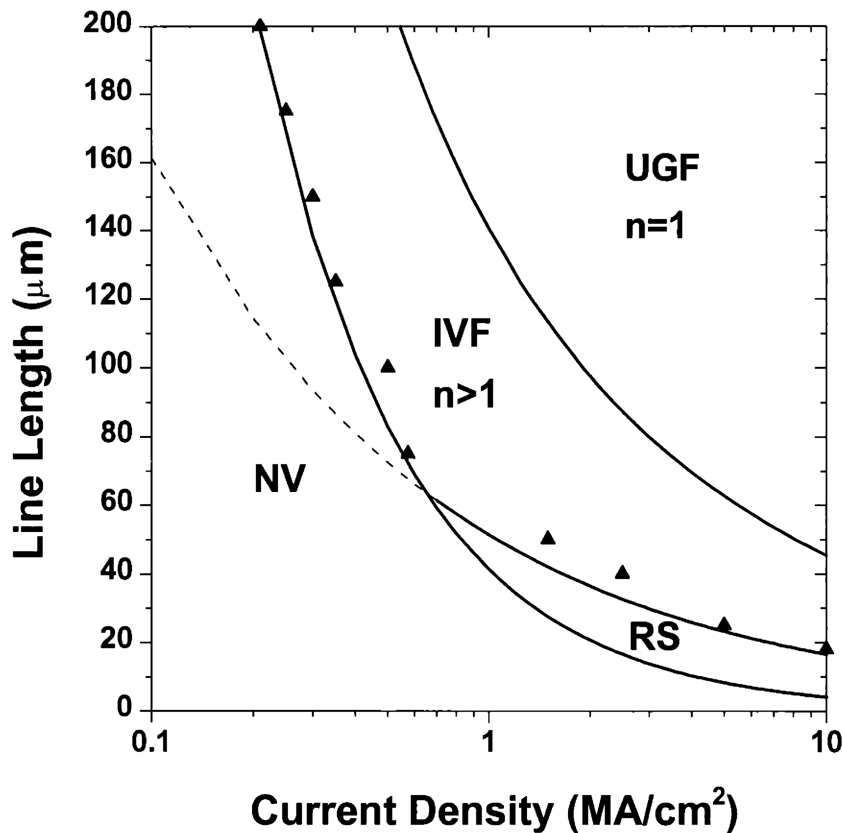


Figure 5-1. A line length vs. current density failure mechanism map illustrating regions of immortality without void nucleation(NV), immortality due to void saturation (RS), failures with impeded void growth with n>1 (IVF), and unimpeded void growth-limited failures with n=1 (UGF) assuming a constant void failure length of 0.50 μm.

The failure maps in this chapter were constructed by simulating electromigration in a population of Al lines for a wide range of line lengths and current densities. The line lengths included: 10, 18, 25, 40, 50, 75, 100, 125, 150, 175, and 200 μm. Failure in each line was simulated at the following current densities: 0.1, 0.25, 0.50, 0.75, 1.0, 2.5, 5.0, 7.5, and 10.0 MA/cm^2. Additional current densities were sometimes investigated to better define transitions in behavior. The simulation model was used as defined in Chap. 2 and 3. The lines were assumed to have widths of 1.0 μm and thicknesses of 0.3 μm. The shunt structure was as specified above. Failure was defined by void growth to a critical void length defined by various failure criteria.

Figure 5-1 shows a failure map constructed assuming a failure criterion of constant-void length of 0.5 μm (or a void volume of 0.15 μm^3). Four distinct regions can be seen: line immortality without void nucleation (NV), immortality with resistance saturation (RS), impeded void failure (IVF) with n>1, and unimpeded void-growth-limited failure (UGF) with n=1 (see Table 5-I). Equations (1.3) and (4.1) were used to represent the boundaries of the regions NV and RS respectively, and match the simulated data well. In the simulations, an interconnect was considered to belong in the NV region if the tensile stress at steady state was lower than the critical tensile stress for void nucleation. Similarly, when the void size was lower than the failure size at steady-state, the interconnect was considered to belong to the RS region. Depending on the choice of failure conditions, it is possible that the RS region will fall within, or intersect, the NV region. In the former case, the line will saturate immediately upon nucleation of the void, as verified using the simulations. The triangles in Fig. 5-1, which represent the simulation results marking the boundary of the immortality regime, agree with the theoretical treatment, being located close to the jL product line.

Table 5-I. Legends Used in Failure Map Figures

Legend	Explanation
NV	Immortality without void nucleation
RS	Immortality due to resistance saturation
IVF	Failure with impeded void growth (n > 1)
UGF	Unimpeded void growth failure (n = 1)
AF	Anode failure due to compressive stresses

In determining the line separating the n>1 impeded void-growth-limited region from the n=1 unimpeded void-growth-limited region, the simulated ttf was plotted as a function of current-density for each line length. The transition current density was defined to be the one at which the average current-density exponent exceeded 1.05. It is

noted that no distinct void-nucleation-limited regime ($n \geq 2$) was observed using this failure criterion.

A number of observations may be made from the constant void volume failure map in Fig. 5-1:

(1) In scaling from accelerated test conditions back to field conditions, the possibility of a transition in failure mechanism is much greater for longer lines (i.e. 150 μm or above) than for shorter lines.
(2) For the shorter lines (less than 50 um), if failure occurs, scaling is difficult since the current density exponent will change throughout the impeded void growth regime.
(3) For low current densities (less than 1 MA/cm^2), lines will either be immortal or fail by impeded void growth ($n>1$); and unimpeded void growth-limited failures with $n=1$ scaling are unlikely, except at very long line lengths (at low j), limiting the accuracy of Black's equation.
(4) Higher current densities are more likely to lead to unimpeded void growth-limited failures since the void nucleates very quickly; most of the failure time is consumed growing the void to the void failure length, without any interaction of the compressive back-stress with the void or tensile stress.
(5) Similarly, higher line lengths will also lead to unimpeded void growth-limited failures since the compressive stress is less likely to affect void nucleation and void growth.
(6) The shortest of lines will remain immortal, regardless of the current density (within the given range of the failure map).

Figure 5-2 illustrates a failure map constructed using a failure condition of void-length of 1.0 μm. The size of the void saturation regime is observed to increase while the

span of the unimpeded void-growth-limited region is decreased. These simulation results are also seen to be in agreement with the theoretical calculations for void nucleation, void saturation, and unimpeded void-growth-limited failure.

While the failure maps above were constructed assuming a constant void volume failure criterion (V-criterion), failure is more often defined by a resistance criterion (R-criterion). For this reason, several new failure maps were constructed using different resistance failure criteria.

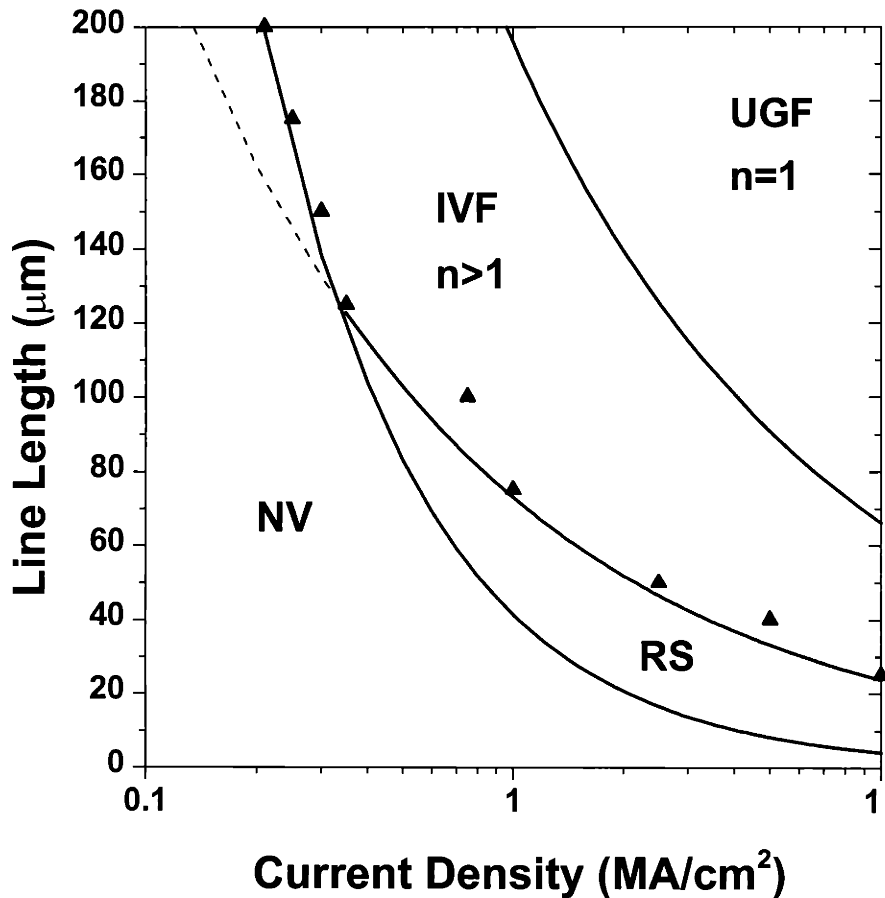


Figure 5-2. Line length vs. current density failure mechanism map based on a void volume failure condition of 1.0 μm. Resistance saturation occurs at longer lines for this failure condition relative to the more conservative failure criterion in Fig. 5-1.

In Fig. 5-3, the failure criterion used to construct the failure map was defined as $\Delta R/R_0$=25%. For this resistance criterion, the calculated failure void size will scale linearly with line length, in contrast to the constant void volume used above. This will lead to increased failures in short lines, and also to a diminished void saturation immortality region, compared to the V-criterion failure map. It should be noted that the subdivision of the immortal region without void nucleation will *not* be affected by this change in failure criterion since it is defined by the jL product. According to the simulation results (see Fig. 6), no void saturation region exists, consistent with the predictions of Eq. A6. The slope of the line marking the unimpeded void-growth-limited region (n=1) was shown to be much steeper for the 25% R-criterion than for the 1.0 μm V-criterion, resulting in an enlarged n=1 region and a greater range of applicability for Black's equation.

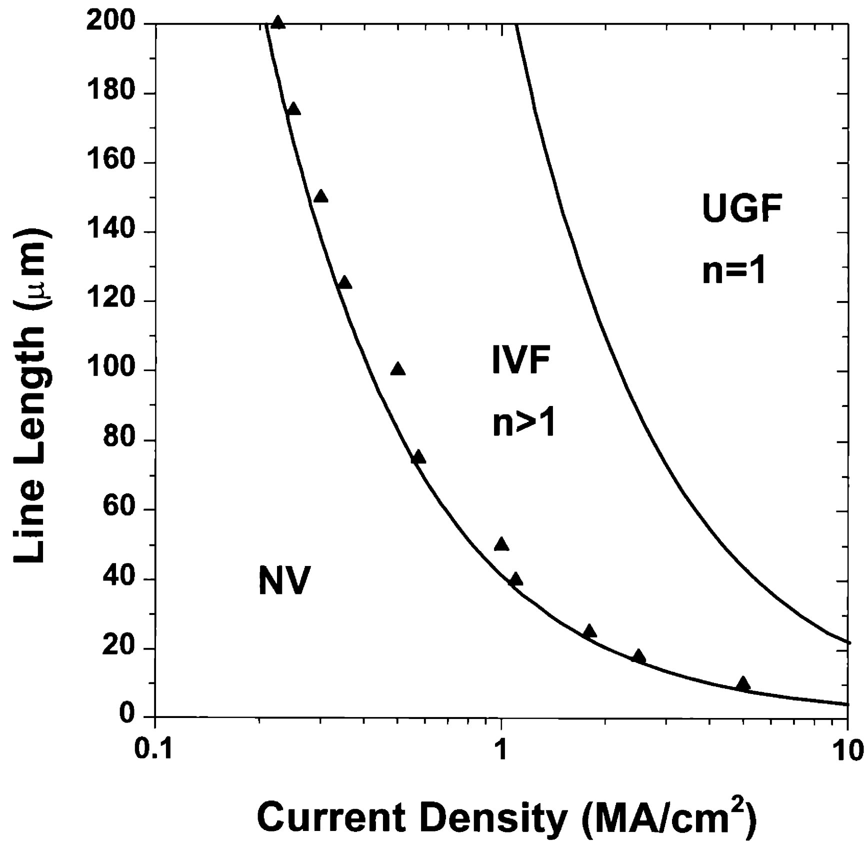


Figure 5-3. Line length vs. current density failure mechanism map assuming a resistance failure condition of 25%. No void saturation region is present.

The failure map in Fig. 5-4 was constructed using a failure criterion defined by $\Delta R/R_o$=50%, and displays a region of immortality due to void growth saturation. In addition, the size of the n=1 void-growth-limited region was observed to decrease. It is also noted that according to Eq. 4.4, halving the resistance failure condition in going from Fig. 5-4 to Fig. 5-3 is equivalent to reducing the shunt-layer thickness by half while maintaining a constant resistance-failure condition. Consequently, Fig. 5-4 also corresponds to a failure map using a 50% resistance failure condition with a 50% thinner shunt structure. Therefore, for a structure with a 50% thinner shunt layer and unadjusted metal structure, defined by TiN(20nm) / Ti(5nm) / Al_3Ti(20nm) / Al(300nm) / TiN(5nm) / Ti(5nm), a decrease in the void saturation regime may be expected relative to the initial shunt structure, since the reduced shunt layer thickness results in a larger resistance increase for a given void size, leading to higher likelihood of failure before void saturation. The decrease in the dimensions of shunt layers, especially in the case of Cu-based interconnects, could therefore lead to reliability concerns. Using Eq. 4.4, additional effects of new materials and modified metal and shunt structures can be quantitatively determined. For instance, an increase in the resistivity of the shunt layer with a simultaneous decrease in metal resistivity will result in reduced void failure lengths, resulting in a diminished void saturation region while also reducing the span of unimpeded (n=1) void-growth-limited failure regime.

In Fig. 5-5, which uses a relatively low resistance failure criterion of $\Delta R/R_o$=1%, four regimes of behavior are observed; an immortality regime due to the absence of void nucleation(NV), a regime of void failures with n>2 scaling, a regime of void-nucleation-limited failures (VNL) with n=2 scaling, and a regime with n<2 scaling. The VNL region on the failure map was defined by first plotting the time-to-failures vs. current density for each line length and identifying the current density range in which the current density exponent was between 1.95 and 2.05. It should also be noted that no regime of immortality due to void-growth saturation was observed. This is in agreement with Eqs. 1 and 2, which indicate that the void nucleation condition is more constraining than the

condition for void growth saturation. Under these conditions, once void nucleation occurs, local stress relaxation rapidly leads to growth of a void large enough to cause a resistance failure, without the need for further long-range diffusion.

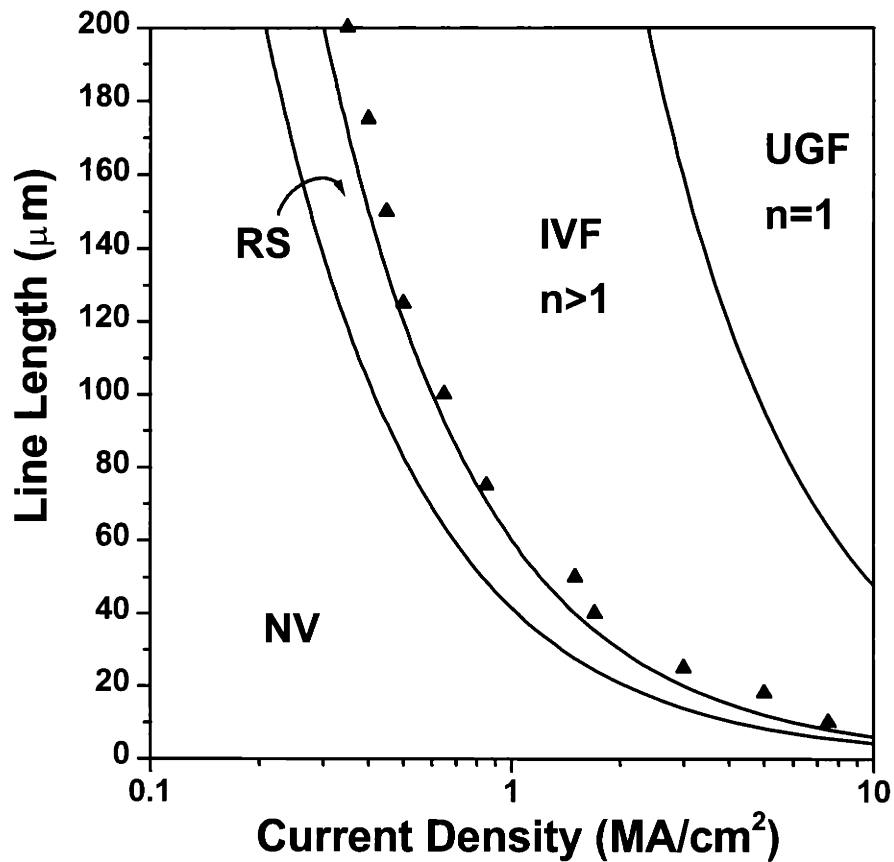


Figure 5-4. Line length vs. current density failure mechanism map assuming a resistance failure condition of 50%. A small void saturation region is present for all line lengths.

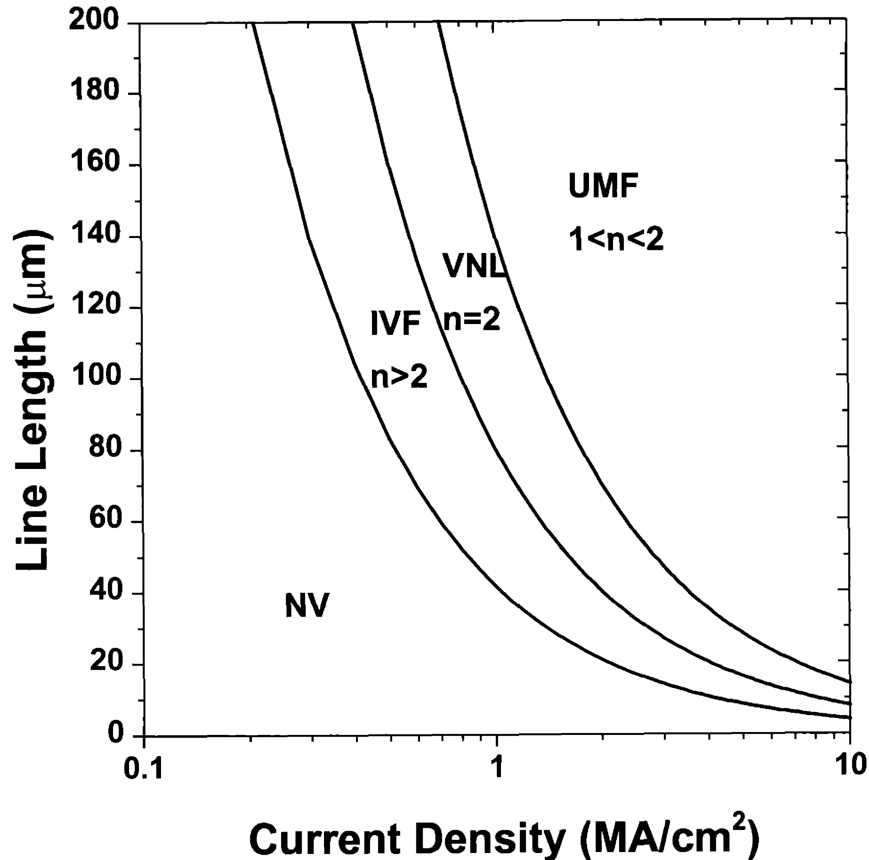


Figure 5-5. Failure map assuming that a 1.0% resistance increase causes failure, illustrating regions of immortality without void nucleation (NV), impeded void failure (IVF), void-nucleation-limited failure (VNL), and unimpeded mixed failure (UMF).

5.3. Failure mechanism map for Al-Cu bamboo interconnects

As an example of failure mechanism maps for *alloy* interconnects, a map was constructed for bamboo Al (0.5wt%Cu) interconnects. While it is known that the addition of Cu has a dramatic effect on the electromigration reliability of polygranular Al interconnects, increasing the times to failure by over an order of magnitude, it is unclear whether Cu has a similar effect on bamboo Al interconnects.

The representative structures chosen were bamboo Al and Al (0.5 wt% Cu) lines, with a 1 μm square cross-section and length L (varied in the range 10 – 200 μm). The lines were clad with Al_3Ti capping layers of total thickness 0.2 μm. The current-density was varied in the range 0.1 –7 MA/cm^2, and the temperature was assumed to be 100 °C. The effective interface diffusivity of Al was assumed to be 100 times the corresponding value through the lattice (and unaffected by the presence of the Al_3Ti layers) [Sri 98], and the effective valence of the Al was assumed to be –2. The corresponding transport properties for Cu in Al were obtained from recent experimental measurements [Sri 99-JAP]. The critical stresses for nucleation of damage were defined to be 400 MPa in tension and 3 GPa in compression, and failure was defined as a 25% increase in electrical resistance. The details of the thermodynamic and kinetic model for electromigration in Al-Cu alloy interconnects are presented in Chap. 2.

Figure 5-6 is the failure mechanism map for Al (0.5wt% Cu) at 100°C. The different domains were obtained as explained in the previous section, with the exception of the line separating regions where the current-density exponent is equal to one from the region in which it is greater than unity. For alloy interconnects, the theoretical approximation of the earlier section does not apply, and this line was obtained by fitting the simulated data to a curve of the form jL = constant. In addition, the anodic or dielectric failure regime (AF) is calculated based on a jL product and is illustrated in the alloy failure map.

The corresponding map for pure Al was found to be identical to the alloy failure map in Fig. 5-6. Numerical simulations of the transport of Cu in such interconnects at *test* conditions show that all the Cu is removed from the cathode regions in times much shorter than experimentally observed failure times [Sri 99]. This result suggests that the presence or absence of Cu has no impact on the topology of the failure mechanism map for bamboo Al interconnects.

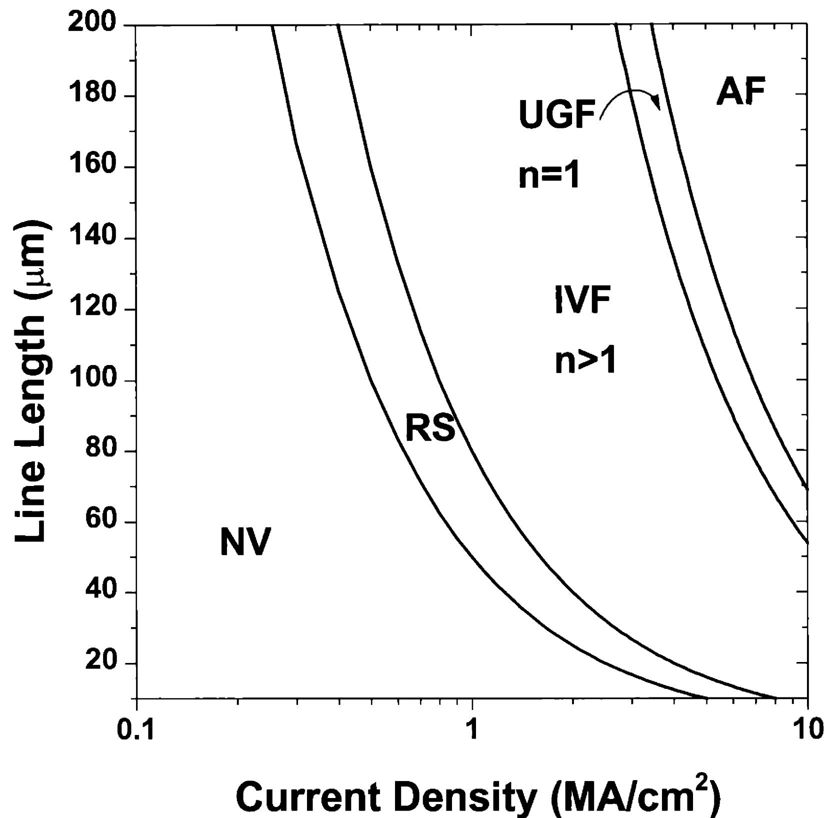


Figure 5-6. Failure mechanism map for bamboo Al-0.5wt%Cu, assuming a resistance failure condition of 25%, constructed for a service temperature of 100°C. The corresponding map for bamboo pure Al is identical to this map. A regime for anode failure (AF) is indicated on this map.

5.4. Analytical Approximations for Failure Map Construction

While the failure mechanism maps above were constructed for simple interconnect geometries and microstructures based on simulation, failure maps can also be constructed by a model based on analytical approximations. These approximate failure maps can be compared to maps constructed by simulations results, or to estimate reliability behavior by the rapid construction of failure maps for similarly simple interconnect structures.

As discussed in Chap. 1, the expression for void nucleation is given by the jL product in Eq. (1.4), and can be used to delimit the failure map subdivision of line immortality without void nucleation. Similarly, the line immortality with void saturation regime may be delineated using the void saturation analysis in Appendix C.1, specifically Eq. (C5) or (C6) depending on the failure criterion.

To determine bounds of the unimpeded void-growth-limited failure regime, a more complex analysis is necessary to determine the transition current density, j_{tr}, as a function of line length and failure criteria. The significance of the transition current density is that it defines the lowest current density at which Black's equation is accurate, assuming a current density exponent of 1. While only a simulation can accurately determine j_{tr}, it is possible to develop an approximate value for j_{tr} analytically. To obtain an n=1 dependence, it is necessary that failure be void-growth-limited, in which case $t_N < t_G$. In addition, the back-stress must not interact with the tensile peak or void.

The time to nucleation can be calculated using the following expression, [Kor 93]

$$t_N = \left(\frac{\sigma_{crit}}{2G}\right)^2 \frac{\pi}{\kappa}, \tag{5.1}$$

where $G = e\rho j z^*/\Omega$ and $\kappa = D_a B\Omega/kT$ and D_a is the atomic diffusivity in the dominant diffusion path. To calculate the void growth time, we start with an expression for the flux given by [Cle 95]

$$J_a = \frac{D_a \Omega C}{kT}\left(\frac{\partial \sigma}{\partial x} + \frac{z^* j\rho}{\Omega}\right), \tag{5.2}$$

where C is the atomic concentration, or the inverse of the atomic volume in stress-free metal. To deplete atoms occupying a volume equal to the failure volume, V_{fail}, atoms must diffuse through a cross-sectional area, A, for a period of time equal to the growth time given by

$$V_{fail} = J_a A t_g \Omega . \tag{5.3}$$

Since there exists a stress relaxation zone near the void, the stress gradient term may be eliminated from Eq. (5.2), leading to an approximate expression for the growth time:

$$t_g = \frac{V_{fail} kT}{D_a G \Omega} , \tag{5.4}$$

assuming unit cross-sectional area. Therefore, t_N and t_G must be calculated using Eqs. (5.1) and (5.4) to verify that the incubation time is less than the growth time to ensure an n=1 exponent.

Lines whose failures are nucleation-limited are assumed to have interaction of the compressive and tensile stress regions at approximately the mid-point of the interconnect. For lines whose failures are growth-limited, the back-stress is assumed to interact with the void at the cathode. Using the solution of Korhonen *et al.* [Kor 93] for the stress profile as a function of position, x, and time,

$$\sigma = G\left[\sqrt{\frac{4\kappa t}{\pi}} \exp\left(-\frac{x^2}{4\kappa t}\right) - x \cdot erfc\left(\frac{x}{\sqrt{4\kappa t}}\right)\right], \tag{5.5}$$

where t is the total time, and is equal to the sum of t_N and t_G given by Eqs. (5.1) and (5.4). For void nucleation-limited failures, x is one-half the length of the line while x is approximated as the full length of the line for void-growth-limited failures. Because the

calculation of the stress in this equation assumes that the diffusivity is not dependent on stress (unlike the simulation tool), Eq. (5.5) is an approximate solution of the stress profile through time [Par 97]. If an interaction is defined by assuming that the compressive stress reaches 5 MPa at the interaction point, the critical current density for a given line length can be approximated by simultaneously solving Eqs. (5.1, 5.4, and 5.5). In this work, the solution to these equations is determined by successive approximations.

In Fig. 5-7, a failure mechanism map for Al is constructed based on simulation results (shown as triangles and squares)assuming a void volume failure condition of 0.5 μm similar to Fig. 5-1. The analytical expressions for line immortality without void nucleation and void saturation subdivisions are overlaid (shown as solid lines), showing very close correlation with simulation. This is expected since the simulation and analytical expressions share a common set of assumptions. Also overlaid on Fig. 5-7 is the analytical expression for the transition current density delineating the n=1 scaling regime. The correlation for this expression with the simulation results is also good, even though they each have different assumptions. The assumptions for the analytical expression for j_{tr} differ from the simulation in that it neglects the back-stress and assumes a non-stress-dependent diffusivity.

The analytical expressions discussed above can be useful for the rapid construction of failure mechanism maps. However, the use of these expressions is limited to simple cases. Besides the limitations discussed above, these analytical expressions are restricted to simple geometries and compositions and to lines with homogeneous microstructures. More complex structures, such as interconnect trees, near-bamboo or polygranular microstructures, and alloys require simulation to develop failure mechanism maps for an accurate assessment of interconnect reliability.

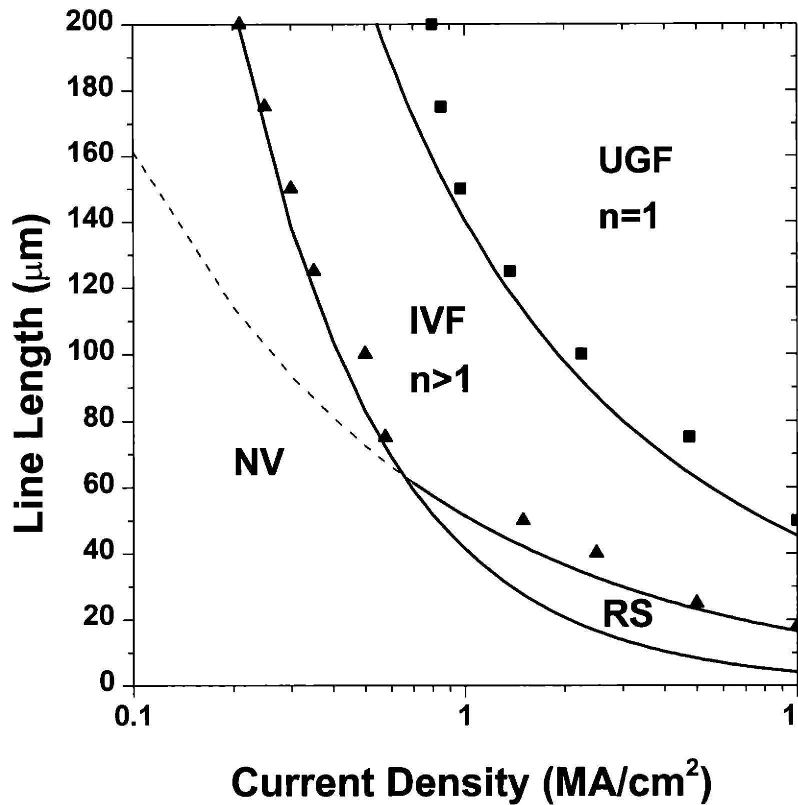


Figure 5-7. Failure mechanism map assuming a constant-void-length failure condition of 0.5 μm. Results from the use of the analytical model for the transition current density are shown as points on this plot, and indicate a good correlation with the simulation results.

5.5. Conclusions

Simulation of electromigration-induced stress evolution is a useful technique for estimation of interconnect reliability. Associating a critical stress with each mode of failure, we can estimate times to failure at test and service conditions. Much of the uncertainty associated with scaling data from accelerated lifetime tests can hence be removed.

Failure modes of interconnects are strongly dependent on, among other factors, line length and current density. A convenient method of cataloging this information is though the construction of failure mechanism maps. These have axes of current density and line length, and illustrate domains of dominance of different failure modes. These domains include immortality, void-nucleation-limited failure, void-growth-limited-failure, and compressive-stress failures. More relaxed failure criteria led to further incidence of void saturation regimes. In addition, in void-volume-based failure criteria, void saturation was only observed at longer line lengths, while resistance-based failure criteria exhibited a resistance saturation regime over all line lengths for moderate failure criteria. When sufficiently strict failure criteria are used, a resistance saturation regime is not observed. Theoretical expressions were determined for delineating some of these regimes in alloyed interconnects, and were shown to agree with simulation results.

Failure mechanism maps provide an overview of the reliability of interconnects, and can be used to scale both current density and line-length. In addition, the effect of changes in materials, geometry and processing can be assessed using such maps. Failure mechanism maps or the information contained in them is crucial in interpreting the relevance of results obtained in accelerated lifetime tests to expected behavior in service conditions. Failure mechanism maps delineate the narrow conditions under which conventional scaling of test results, based on the use of Black's equation, are valid.

Chapter 6

Electromigration Experiments with Cu

6.1. Introduction

As described in the last three chapters, MIT/EmSim has been used to model and characterize the behavior of electromigration-induced failures in pure and alloyed aluminum interconnects. Recent industry announcements have signaled a move toward the use of Cu-based interconnects, necessitating the adaptation of the MIT/EmSim model to incorporate the simulation of copper-based interconnects. Cu-based interconnects offer lower resistivity for reduced interconnect RC delay, which accounts for up to 50% of the total signal path delay. [Obe 88] Cu interconnects are also believed to provide superior resistance to electromigration-induced damage.

To accommodate the introduction of Cu interconnects, a metamorphosis of the interconnect processing methodology has been required. The fabrication of Al-based interconnects has typically been performed using subtractive etch, in which films of the aluminum metal along with shunt material are deposited followed by a patterning and etch step, and dielectric deposition (See Fig. 6-1).

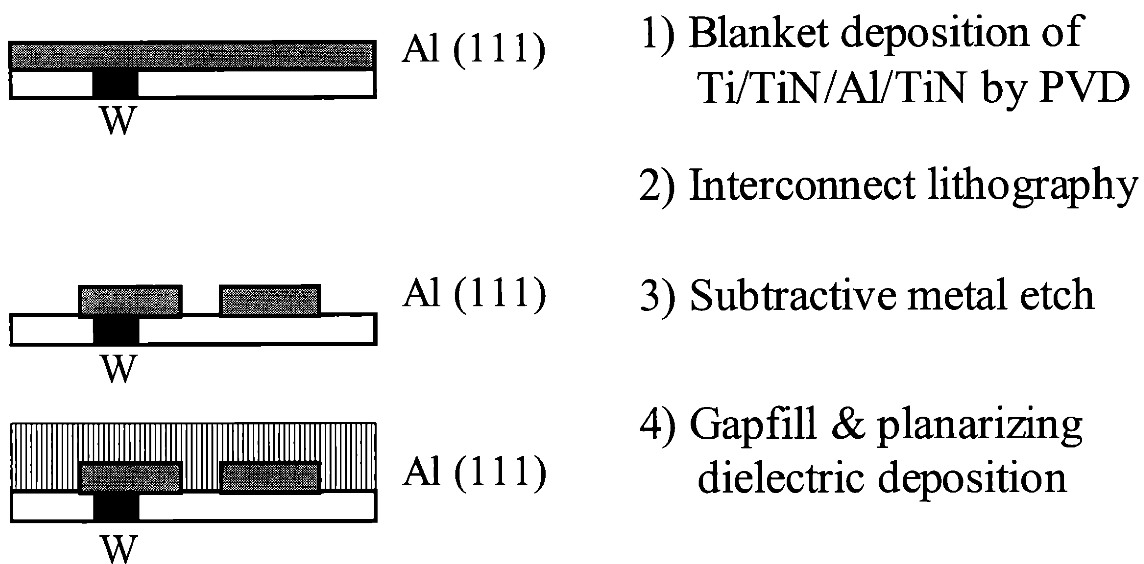


Figure 6-1. Subtractive etch fabrication process for patterning Al-based interconnects. [Based on Gro 99]

Cu interconnect fabrication is customarily performed by the damascene process. In this process, the inter-layer dielectric (ILD) is first deposited and patterned, followed by shunt material and Cu seed layer deposition (see Fig. 6-2). Additional Cu is then electroplated, and planarized by chemical-mechanical polishing (CMP). Finally, silicon nitride is deposited to serve as a barrier layer. This process is then repeated for the Cu vias. In dual damascene, the ILD is deposited and patterned twice, once for the interconnect level and once again for the via level. This is then followed by Cu deposition and planarization as described above. In the work to be described this chapter, a subtractive-etch process was developed for Cu interconnect fabrication.

Relative to Al technology, the understanding of electromigration in Cu interconnects is still in its infancy. Numerous studies have been conducted measuring the failure times of wide Al, indicating that wide Al lines traditionally failed due to the presence of a network of high-diffusivity grain boundaries [e.g., see review by Mal 97].

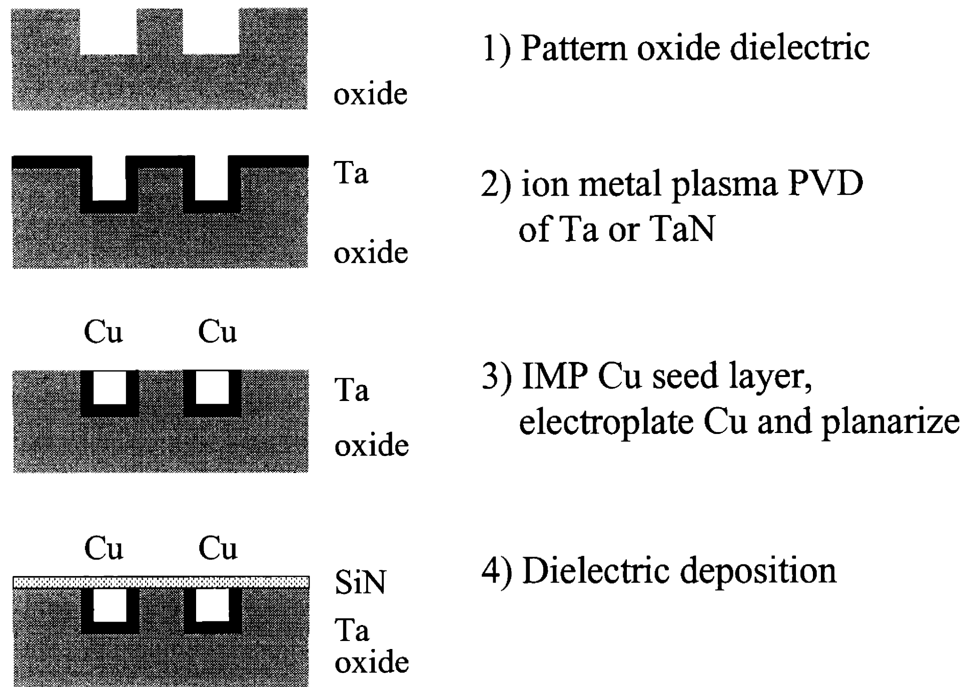


Figure 6-2. Cu interconnects are fabricated by the damascene process, which requires the patterning of the dielectric followed by metal deposition and subsequent CMP.

Early comparisons of Al failure times in wide lines with failure times in wide Cu interconnects demonstrated the superior reliability of Cu-based interconnects by a factor of 100. [e.g., Ari 94, Hu 98] However, Cu self-diffusion at grain boundaries is over 1000 times slower compared to Al self-diffusion [Fro 82]. Furthermore, the activation energy for electromigration failure was often found to be less than 0.9 eV, even though the activation energy for grain boundary diffusion in Cu is closer to 1.2 eV. [Llo 95] Because Cu lattice diffusion has an activation of 2.2 eV, lattice diffusion could not provide the dominant diffusive pathway over any reasonable test temperatures. [Fro 82] It was surmised that the dominant path for Cu diffusion was the interfaces, which typically exhibited smaller activation energies. [Llo 95, Hu 97, Par 91]. If the dominant path for Cu self-diffusion is at the interfaces, influence of microstructure on

electromigration-induced failures could be expected to be reduced because of the smaller impact of grain boundary diffusion. [Hu 97] Assuming that the interface was kinetically dominating atomic transport, two possible sources were hypothesized, the Cu-Ta interface and the Cu-Si_3N_4 interface, although it was not known which dominates. Initial studies suggested that the dominant transport path in Cu interconnects is at the Cu-Si_3N_4 interface. [Hu 97]. However, if the diffusivity at the interface could be suppressed in Cu, grain boundary diffusion would become the dominant diffusivity path as in Al, leading to improved Cu reliability. Recent experiments have demonstrated that a dominant diffusivity might be present at the grain boundaries of wide lines. [Hu 99] An accurate prediction of electromigration-induced failure times for Cu interconnects requires further understanding into, and measurement of, the absolute and relative magnitudes of the diffusivities and z*'s. These issues presented here are the motivation for the work to be presented in this chapter.

In order to model electromigration in Cu interconnects accurately, the kinetic parameters including diffusivity and z* must be precisely measured for the overall interconnect, and if possible, individually for both the Cu-Ta interface and the Cu-Si_3N_4 interface. The observation of phenomenology consistent with the predictions of MIT/EmSim would also help support the MIT/EmSim electromigration model as it has been extended for Cu interconnects. To complete these measurements, a test structure was developed that would contain both Cu-Ta and Cu-Si_3N_4 interfaces, and allow for the determination of the effective diffusivity and possibly diffusivities at each interface by the measurement of void growth rates for interconnects of varying linewidths [similar to Hu 97]. The determination of z* would be based on the measurement of resistance saturation. The next few sections detail the test structure, its fabrication, and electromigration testing methods and results.

6.2. Interconnect Design & Fabrication

The design of the Cu interconnect test structures require the presence of Cu metal with non-diffusive contacts at the ends of each line to provide a diffusion barrier necessary for stress evolution. Shunting of the Cu with Ta supports a void nucleation and growth process. The interconnect structure must also exhibit void saturation to enable the determination of an effective z*', requiring that the interconnect and shunt structure be encapsulated in a rigid Si_3N_4 dielectric to permit the formation of a back-stress to suppress further electromigration. This dielectric, along with Ta, must interface with the Cu metal permit the measurement of effective and potentially individual diffusivities at these interfaces. Finally, the structure must be simple enough to be feasibly and economically fabricated in the portion of the MIT fab allocated for Cu-contaminated wafers within a reasonable amount of time. Because no Cu-CMP process was available in the MIT fab at the time of this work, a damascene process could not be developed, necessitating the development of a subtractive etch process for the fabrication of Cu interconnects.

Test Structures

The general form of the test structures is a single metal level interconnect composed of Cu on top of Ta (Fig 6-3). The Cu has a thickness of 1.5 μm while the Ta has a thickness of 0.375 μm. These dimensions were selected to ensure that the Ta could allow for significant void growth without either excessive heating or electromigrating, while also to provide reasonable failure times and the possibility of void saturation based on MIT/EmSim simulations performed using estimated input values for Cu. To provide a diffusion barrier at the end of each interconnect similar to those present at studs in damascene, a short segment of the Cu layer is etched at the end of each interconnect structure. Since Ta does not electromigrate under the test conditions intended for Cu EM

testing, the Ta provides the function of a diffusion barrier. The interconnect test structures are electrically isolated from the wafer by an LPCVD nitride. Further encapsulation above and on the sides of each interconnect is provided by a PECVD nitride.

The primary interconnect structures tested were probe pad structures, which consisted of interconnect structures connected by very short runners (less than 20 um long) to either 100x100 or 75x75 um probe pads. Half of these structures shared probe pads to increase the number of interconnect structures present; half were designed as isolated. This set of test structures is shown with a top and side view in Fig. 6-1.

Bond pad interconect structures, in which long interconnect structures were connected to bond pads located around the perimeter of the die, were placed throughout a 1cm x 1cm die (Fig. 6-4) while probe pad structures were nestled between the bond pad structures. Alignment structures bearing crosses of multiple sizes were placed at the center while etch indicators were placed at the center and four corners of each die. There were near 1,000 test structures per die of varying linewidths (1-10 microns) and varying linewidths (25-750 microns), and a interconnect ID was etched into the upper nitride adjacent to each test structure (e.g., L100W8).

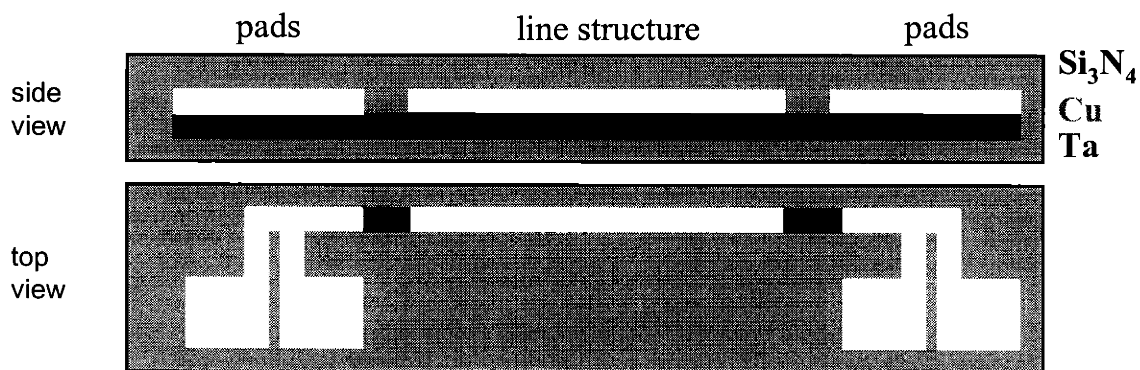
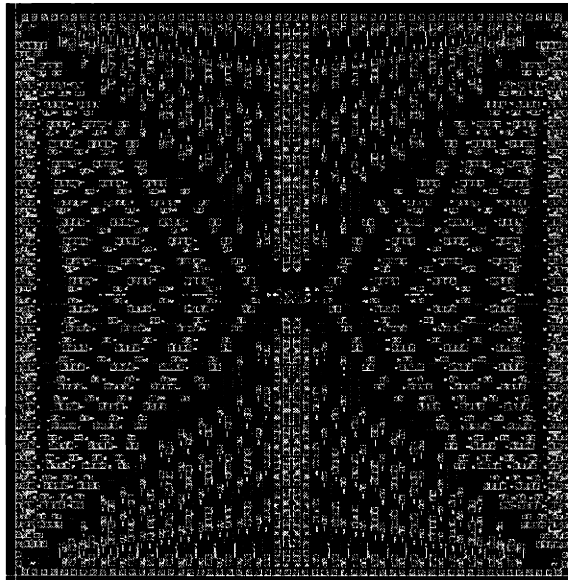


Figure 6-3. Top and side view of probe pad interconnect structures. Cu is in white, Ta in black, and nitride in grey.

Fabrication Sequence

The interconnect test structures were fabricated using MIT microfabrication facilities: Integrated Circuits Laboratory (ICL) and the Technology Research Laboratory (TRL). Only initial wafer processing was performed in ICL due to stringent bans of Cu-contaminated wafers.

(a)



(b)

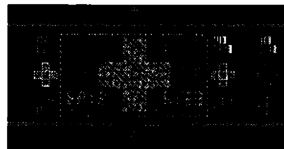


Figure 6-4. Die layout (a) composing of three mask layers. Alignment structures (b) and etch indicators are also shown.

The interconnect test structures were fabricated on 4" <100> n-type Si test wafers with 1-20 Ω-cm resistivity. Since the selection of the Si wafer was irrelevant with regards to the design and processing of the interconnect test structures, these wafers were chosen for economy rather than technology.

LPCVD Wafer Encapsulation

Upon receiving the wafers, an RCA clean in ICL was performed to remove any metallic, organic, and inorganic contaminants in compliance with ICL regulations. Next, an LPCVD silicon nitride of thickness 725+/-4 angstrom (KLA-Tencor Prometrix UV-1250SE profilometer using Si_3N_4 recipe) was deposited on both sides of the wafer using Tube B2 in ICL with the standard LPCVD nitride recipe. The deposition rate was measured to be 28 A/min. The presence of silicon nitride on the back-side of the wafer is important later in the process as it will serve as an etch mask for the backside of the Si wafer.

Metal Sputtering

The Ta-Cu metal bi-layer were then deposited in a three target RF sputtering tool (Materials Research Corporation 8620 Sputtering System, Orangeburg, New York, currently in possession of Prof. Smith) of approximately 30 years of age. This system had been previously modified from its original construction by the substitution of an improved cryogenic pump for achieving improved (initial) vacuum with shorter pumping times. For depositions, after inserting a wafer, the chamber was evacuated to better than $3x10^{-6}$ torr (approximately 3.5 hrs for this pump-down). An argon flow of 20.0 sccm was used and the base pressure during sputtering was 20 mtorr. For most wafers, a power of 350 watts was used, with an RF peak voltage ranging from 1.7 to 1.8 KV. These power and gas flow settings were selected for their: stability in maintaining the plasma, avoidance of standing wave switching, and minimization of reflectance (less than 4 watts). The 5" diameter Ta target was of 99.99999% purity while the Cu target was an OHFC Cu with about two nines purity (metallurgical grade). Prior to each sputtering, a

pre-sputtering period of 6-8 minutes with a lobe shutter covering the target (and the wafer situated on the other side of the chamber) was used to remove any surface contamination of the target. The pre-sputtering period also afforded an opportunity to adjust RF tuning to optimize power and reflectance settings. The tuning varied from run to run based on the material coating the sidewalls of the chamber, although a specific power and reflectance could always be consistently achieved.

Initial calibration runs were performed by depositing uni-layer films for each target and patterning to allow measurement of film thickness based on step heights using a surface profilometer (Dektak II profilometer in TRL). Wafers 11 and 12 were used for the initial calibration runs of Ta and Cu sputtering rates. These thickness measurements revealed a Ta deposition rate of 124 A/min and Cu deposition rate of 252 A/min at the above power and tuning settings. Figure 6-5 displays the film thickness determined by the deposition rates and confirmed by profilometer measurements for the entire range of wafers fabricated in this experiment. For wafers allocated to test structure production, film thickness of both the Ta and Cu were varied for several different wafers, although only one thickness combination yielded satisfactory etching results (see below). Subsequent etching of sputtered wafers along with profilometer measurements demonstrated good uniformity (< 500 A variance over 6 cm) in the center 6 centimeters of the wafer, with significantly declining uniformity along the perimeter of the wafer. The first nine wafers were fabricated with a Ta/Cu/Ta layer structure while all wafers fabricated subsequent to wafer 13 employed a Ta/Cu structure. This modification was made due to difficulties with contacting microprobes to Ta pads, possibly due to Ta oxidation.

After sputtering, the wafers were brought to TRL and a solvent clean was performed, both to remove any contaminants from the wafers and to comply with TRL regulations regarding wafer entry. This solvent clean consisted of a 4 minute dip in each of acetone, methanol, and 2-isopropanol. The wafers were rapidly washed under a de-

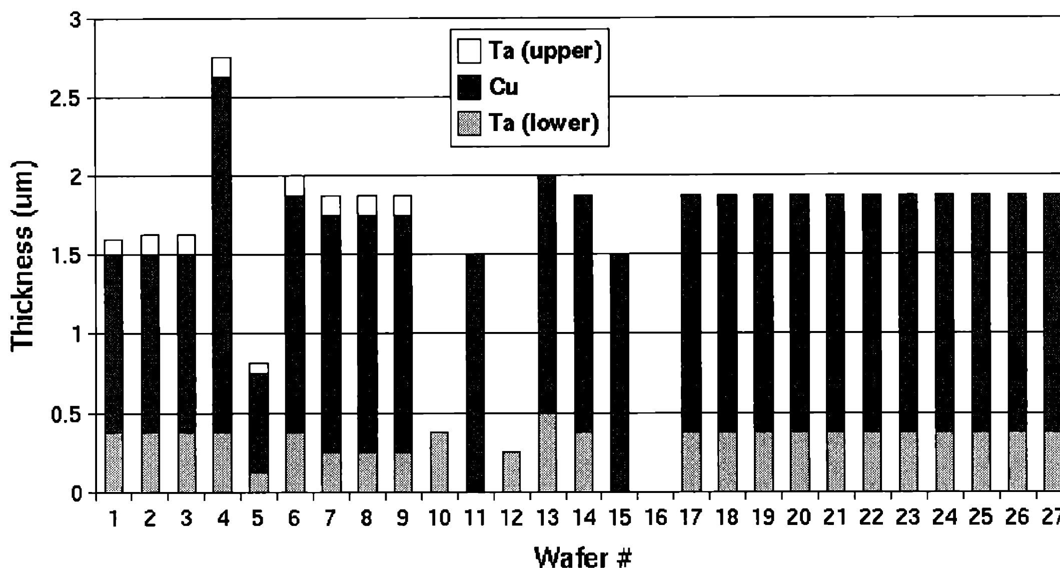


Figure 6-5. Thicknesses of Cu / Ta films deposited by sputtering for all wafers fabricated.

ionized (DI) water fountain and inserted into the automatic wafer cleaner for one normal cycle to avoid any possibility of solvent residue, as confirmed by microscopic observation.

Patterning Level 1

The next step was the first mask layer patterning of the interconnect test structures, pads, and connecting runners. The first level mask was a 5" chrome-plated bright field mask fabricated by Microtronics. Conventional patterning procedures as recommended by TRL were used. After a normal HMDS cycle to promote resist adhesion to the metal film, standard positive resist (OCG-825-20) was spun on using a coater. The spin cycle on the coater was composed of 6 second dispense, 6 second spread, and 30 second spin at 3000 rpm yielding a post-spin thickness of 1.0 microns. This thickness was confirmed by both TRL calibration charts and the profilometer measurements. After a 30 minute pre-bake at 90°C, the wafers were then exposed in

vacuum contact using a contact printer (Karl Suss MA4 contact aligner mounted on air table) of wavelength 350-405, intensity of 5-6 mW/cm^2 and exposure time of 55.0 seconds. The mask, which beared a 1 μm minimum feature size, was a 5" soda-lime, chrome bright-field master mask fabricated by Microtronics using a 0.25 μm spot size. The patterns were then developed using a gentle swirl in OCG 934 1:1 positive resist developer (Arch Chemicals) for 70 seconds. Both the exposure time and development time were optimized using several wafers and an optical microscope. After developing, a post-bake was performed for 30 minutes at 120°C. Visual inspection of the wafer, along with microscopic inspection using an optical microscope was not suitable for photoresist examination. Because photoresist may be observed using a fluoroscope, fluoroscopic inspection was instead performed on each wafer. It was observed that at low power, the metal reflected some of the fluoroscope's light, resulting in a diffuse red glow while high power clearly defined the photoresist regions.

The next step was to perform a wet etch of the Cu and Ta. Dry etching was avoided due to the lack of a good dry etch for Cu -- chlorine chemistry-based dry etches can result in the formation of a non-volatile $CuCl_2$, which is corrosive especially as it hydrolyzes upon exposure toair. [Gha 94] Considerable efforts were geared toward discovering and optimizing wet etches for Cu, and in particular, Ta. While it is commonly known that nitric acid etches Cu, concentrated HNO_3 also attacks and removes photoresist. After experimenting with varying dilutions of nitric acid with deionized water, a 1:10 HNO3:H2O etchant was used for Cu, providing a good balance between etch rate and minimization of photoresist damage. Etch times of 30 seconds were sufficient for Cu films of 1.5 um thickness. Nitric acid, and its dilutions, were shown to have no effect on Ta.

Many sources [e.g., Gha 94] list HF as an etchant for Ta; however tests with HF and dilutions thereof suffered from either photoresist attack/removal or lack of any etching of Ta. Buffered oxide etch 7:1 (NH_4F buffer with HF) along with its dilutions

were also unsuccessful at etching Ta. Finally, as noted by Ghandhi [Gha 94], 1:1:2 HF:HNO_3:H_2O is an adequate etchant for Ta. Further dilutions of this etchant, even as small as 1:1:2.5 ratios, resulted in the lack of any Ta etching while reduced deionized water proportions resulted in considerable photoresist damage and removal. Etch times of Ta were about 15-20 seconds for films of 0.3 um thickness. This etchant also etches Cu at an even faster rate of 0.15 μm/sec, allowing the use of a single etchant to etch both the Ta and Cu metal for the first level mask process. The etchant solution was always used within ten minutes of its preparation to minimize weakening of the solution. Each wafer was etched in a fresh, 2400 ml etchant solution to provide best etching results -- etchant solutions of 1000 ml resulted in uneven etches and excessively long etch times as the solution weakened over time while 4000 ml etchant solutions were shown to attack the photoresist. After etching, the wafers were cleaned in an automatic wafer cleaner. Photoresist was removed by placing the wafers in an acetone bath for 30 minutes in an ultrasonic cleaner followed by cleaning in the automatic wafer cleaner.

Patterning Level 2

The line endstops were then patterned at the ends of each interconnect test structure using a second mask (dark-field). Similar procedures as described above were used for patterning in this second mask. Alignment structures bearing a cross (on the wafer) within a slightly large cross (on the mask) were used for wafer-mask alignment. Exposure times and development times were the same, followed by inspection by microscopy and optical fluorscopy.

To etch the line endstops, the upper layer of Ta (if present) and Cu must be removed, leaving the lower layer of Ta intact. This was accomplished by using the 1:1:2 HF:HNO_3:H_2O etchant described above for 8-10 seconds to etch through the thin (0.1 um) Ta upper layer and part of the Cu. After a dip in DI water (to ensure that etching ceases), the Cu was selectively etched using the 1:10 diluted nitric acid for 60 seconds, ensuring complete removal of the Cu. Wafers which only required removal of the Cu

layer were only etched with dilute nitric acid. Inspection using the optical microscope revealed a gold colored Ta metal at the line ends with a blue perimeter indicating the silicon nitride, enabling a determination of whether the etching was sufficient. Wafers were then washed in the automatic wafer cleaner and acetone removed as before.

Passivation

To passivate the interconnect structure (and provide a silicon nitride interface on all exposed surfaces of the Cu interconnect), a PECVD nitride was deposited using a PlasmaQuest Series II ECR Enhanced RIE and PECVD tool. The standard recipe for silicon nitride was followed using gases of SiH_4 (113 sccm)and dry N_2 (10 sccm) at a power of 210 watts at 80°C. The RF tuning was continually adjusted to maintain a constant power and minimize reflectance, which did not exceed 0.5%. The deposition time was 21 minutes at a deposition rate of 405 A/min resulting in a final thickness of 0.85 um as measured by profilometer. An additional batch of wafers was processed without the PECVD nitride deposition to allow a comparison of electromigration test data with and without the nitride passivation.

Patterning Level 3

The third level mask (dark-field) was then patterned, providing contact cuts through the nitride to the pads. The procedure followed was as described for the second level mask, using an identical set of alignment marks on another portion of the wafer. Fluoroscopy was used to verify the photoresist patterning.

The PECVD silicon nitride was etched using buffered oxide etch(BOE) 7:1. Buffered oxide etch is commonly used to evaluate the quality of silicon nitride depositions [Gha 94], and was chosen as an etchant in this process. An etch rate of approximately 100 A/sec was observed. Since BOE does not etch Cu or Ta, this wet etch is selective to nitride only and is conveniently used for contact cuts through the nitride. After visual inspection, the wafer was cleaned in the automatic wafer cleaner, and the

photoresist was removed by an acetone soak in the ultrasonic cleaner. The wafer was then sectioned into dies using an automatic die saw equipped with a diamond-impregnated stainless steel rotary blade. An adhesive cutting tape was applied to the back of each wafer prior to cutting to secure the freed dies to the stage.

Testing Procedure & Equipment

After processing, the die was placed on a electromigration probe station equipped with a heated chuck. The chuck temperature was controlled by a temperature controller, and varied within one degree of the setpoint. A second thermocouple contacting the chuck surface was used to verify that the temperature at the surface remained at the setpoint. Interconnects were tested at 200, 250, and 300°C.

Four micropositioners were contacted to the four pads on each test structure, employing a four-point resistance measurement technique. An external power supply supplied a constant current, and the voltage was measured continuously by voltmeter and at one second intervals by computerized data recording. The ammeter was switched into the circuit as necessary. All of the computerized DAQ equipment, multimeters, and power supplies were placed on an electrical circuit equipped with line noise suppressors and a uninterruptible power supply (UPS) to minimize electrical supply noise. The set-up is detailed in Fig. 6-6.

Joule heating tests were conducted on select interconnects to ensure that the test temperature was not being significantly altered due to resistive heating. A temperature increase is undesired during testing since diffusivity is an exponential of the inverse of temperature, which could lead to significant errors in the measurement of electromigration properties. [Dhe 78] A brief overview of Joule heating effects on fabricated interconnect structures is presented here. Lloyd et al. indicate that a current density less than 2.0 MA/cm^2 is insufficient to lead to a temperature gradient capable of

affecting electromigration-induced stress evolution. [Llo 99] Arnaud et al. demonstrate through experiments using Al-Cu interconnects encapsulated in oxide that joule heating does not affect the validity of electromigration tests below 5.0 MA/cm^2 for DC currents. [Arn 99, Ove 98, Jon 96] In the early 1970's, it was demonstrated in experiments the role of thermal gradients on void motion was relatively small. [Ho 70] Recent experiments have shown that Al-4%Cu interconnects *without* a shunt suffer from Joule heating when the formation of a void reduces the local metal width by 75-90%, resulting in localized heating due to current crowding. [Jon 96] Analytic studies by Ru confirm this general conclusion, relegating thermomigration to affect void growth to open circuit failure during the late stages of electromigration, in which the void spans a majority of the linewidth for unshunted interconnects. [Ru 99] Although AC currents were *not* used in this thesis experiment, it has been proposed that Joule heating from AC current loads could lead to a temperature cycling of the metal and dielectric, leading to mechanical weakness based on the difference in coefficients of thermal expansion. [Wal 98] Because the Ta thickness is considerably larger than the shunts in the literature described above, Joule heating would not seem to be of significant concern in this interconnect structure. For this structure, most of the joule heating would

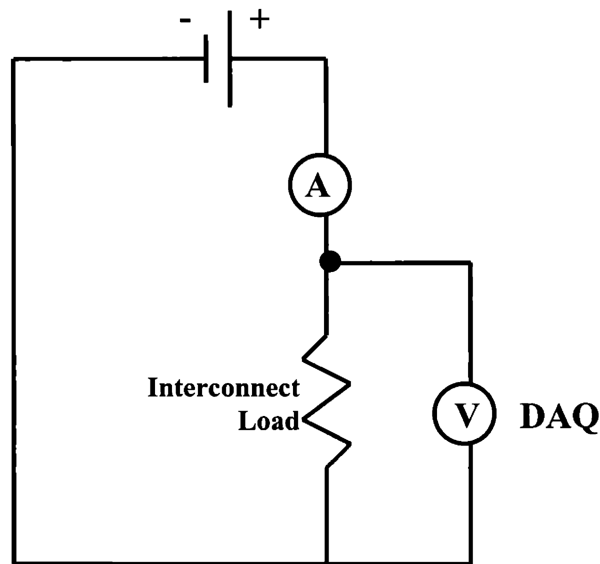


Figure 6-6. Schematic of an electromigration test station.

be expected to occur at the Ta diffusion barriers at the line ends, where stress evolution is most significant. This necessitates a measurement of joule heating at experimental conditions. The increase in temperature due to Joule heating ΔT can be expressed as:

$$\Delta T = \frac{1}{\alpha}\left(\frac{R}{R_o} - 1\right). \tag{6.1}$$

where (R-R_o) is the resistance increase from joule heating and α is the temperature coefficient of resistivity (0.0033 /K for Cu). By monitoring the voltage while successively increasing the current, the resistance is given by Ohm's law and an estimate of the joule heating present at a particular experimental condition may be determined by Eq. 6.1. The results of the joule heating tests are described in the next section.

Finite-element simulations of the joule heating in this interconnect structure were also performed to ensure that there was no significant localized heating in the Cu adjacent to the Ta diffusion barriers (artificial studs). Because of the significantly higher resistivity of Ta along with its smaller dimensions relative to Cu, the majority of joule heating is expected to occur at the Ta artificial studs. A gradient in temperature could lead a thermally-induced atomic flux, and could also be expected to affect the diffusivity along the length of the gradient. [e.g., Kir 91, Llo 99, Ru 99] The geometry and dimensions of the interconnect structure were defined according to the fabricated structure defined above. The interconnect was divided into a 100 element, 1D mesh of one micron long cells, and the fluxes of heat between them determined by

$$J_T = -\kappa_T \frac{\partial T}{\partial x}, \tag{6.2}$$

where J_T is the thermal flux, and κ_T is the thermal conductivity. The pads are assumed to act as perfect heat sinks, which seems appropriate given the experimentally-observed effectiveness of the micropositioner probe tips on reducing hot chuck temperatures when

placed in direct contact with the specimen stage. During each time step, the joule heating was calculated based on I^2R, and the heat fluxes between adjacent cells calculated based on Eq. 6.2. Additional heat losses from the metal to the dielectric across all four Cu-nitride interfaces were also calculated based on Eq. 6.2. In recent literature, it has been demonstrated through simulations and analytic methods that for interconnects whose linewidths are larger than the underlying oxide, the majority of heat transfer is to the Si wafer. [Kel 99] Because the nitride dielectric layer isolating the interconnect from the Si wafer in this thesis experiment is relatively thin, approximately 700 nm thick, the dielectric and Si wafer can be assumed to be perfect heat sinks. [Kel 99] It is noted that all perfect heat sinks were assumed to retain the hot chuck temperature. Once the net flux of energy (heat) into a cell is determined (and weighted) for a particular time step, the increase in temperature of the cell is determined by dividing this energy by the heat capacity of the material in the cell. For most metals, and Cu in particular, this can be approximated by 3*R where R is 8.314 J/ (mol K). This calculation is repeated for each of the cells, and the new temperature profile of the interconnect is determined. By repeating this heat generation and heat flux calculation for each iteration of time step, the temperature profile through time can be determined. The thermal conductivities of Cu, Ta, and Si_3N_4 used in this simulation were 400, 57, and 30 watts / (m K) respectively. The chuck temperature was defined as 200°C, which will lead to the largest and most conservative temperature gradients. The results of this simulation will be discussed in the next sub-section.

Prior to each run, the sample dies were heated at 300°C for six hours to aid in densifying the PECVD nitride, and to provide a stable microstructure in the Cu interconnects. After this pre-anneal, dies were placed on the hot chuck of the appropriate EM test station for one hour to allow the temperature to equilibrate throughout the die. Interconnects were subjected to either (jL) product testing or lifetime testing. For (jL) product testing, a current density of 0.1 MA/cm^2 was applied, and raised every 48 hours until a resistance increase indicating void nucleation occurred. For lifetime testing, a

constant current density was applied throughout the test, which ended either by the observation of resistance saturation or massive resistance increases indicating dielectric failure.

6.3. Characterization & Test Results

The process outlined above was successfully used to fabricate Ta(0.375μm) / Cu (1.5μm) / Ta (0.125μm) and Ta (0.375μm) / Cu (1.5μm) interconnects encapsulated in SiN in MIT's microfabrication facilities using a subtractive-etch approach. Cu strips were etched away at the end of line structures to serve as non-diffusive, EM-resistant studs analagous to the diffusive barrier present in damascene structures. The most difficult processing step in this approach is etching the Ta, which requires a vigorous etchant. Linewidths ranging from 6-10 microns and line lengths of 25 and 50 ums have been demonstrated. However, the Ta/Cu/Ta structures were found to be unsuitable for EM testing because of excessively high contact resistances, presumably due to oxidation of the Ta. Subsequent wafer processing employed the Ta/Cu structure (see Fig. 6-3), for which contacting was not a problem. No difficulties were encountered in fabricating Cu/Ta structures without Si_3N_4 passivation.

<u>Topographical Characterization</u>

The dies were inspected using both optical microscopy (Nikon Nomarski Microscope), scanning electron microscopy (Hitachi), and interferometric profilometry (Wyko). An optical microscopy view of a cluster of interconnect structures is shown in Fig. 6-7. In this figure, the interconnects and runners are shown to maintain a constant linewidth. Measurements of the linewidth using occular gradations were performed using another Nikon optical microscope (not equipped with a camera), and indicated that most 8 and 10 μm wide lines had been patterned to the proper linewidths, although some 6 μm wide lines had irregularities. For this reason, EM testing was performed only on lines

with a linewidth of 8 or 10 microns. The removal of the Cu layer at the line ends (pseudo-studs) was also verified for several dies on each wafer using the second optical microscope.

For the dies used in scanning electron microscopy imaging, the PECVD nitride passivation, if present, was etched away. A conductive chromium layer was sputtered on prior to imaging to minimize surface charging. Images from an SEM of a structure and a close-up of the interconnect are shown in Fig. 6-8a and b. The pads appear a dark grey, with the lower Ta layer appearing a very bright white at the periphery. The interconnects and connecting runners appear to maintain the defined linewidth, although the blurring from the Ta layer make this difficult. In fact, for all the SEM images, the interconnects appeared a fuzzy white, and all attempts to obtain improved images failed. Presumably, the high atomic number of Ta is resulting in increased electron capture and release from the scanning e^- beam, making imaging difficult.

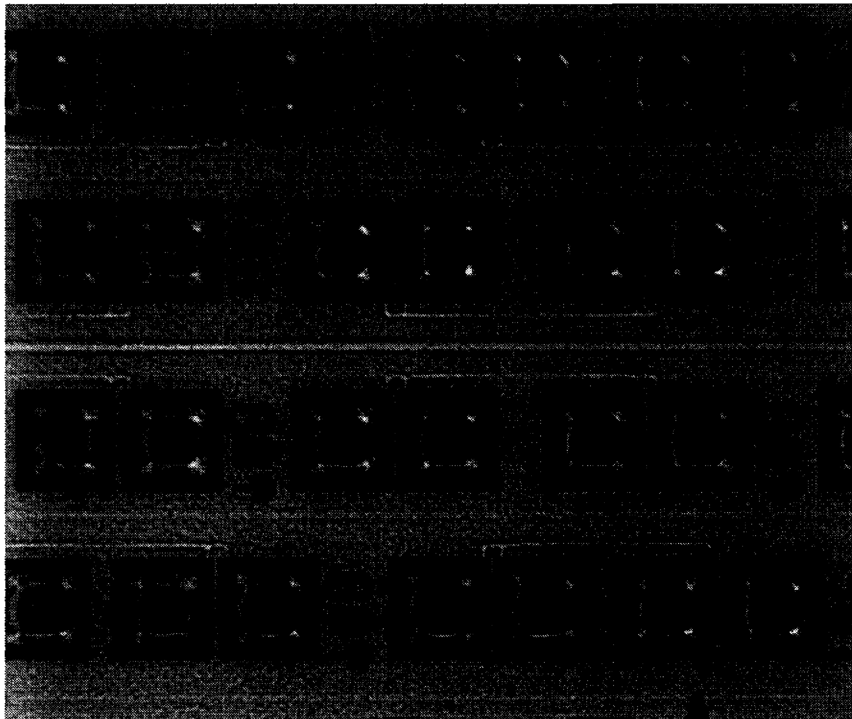
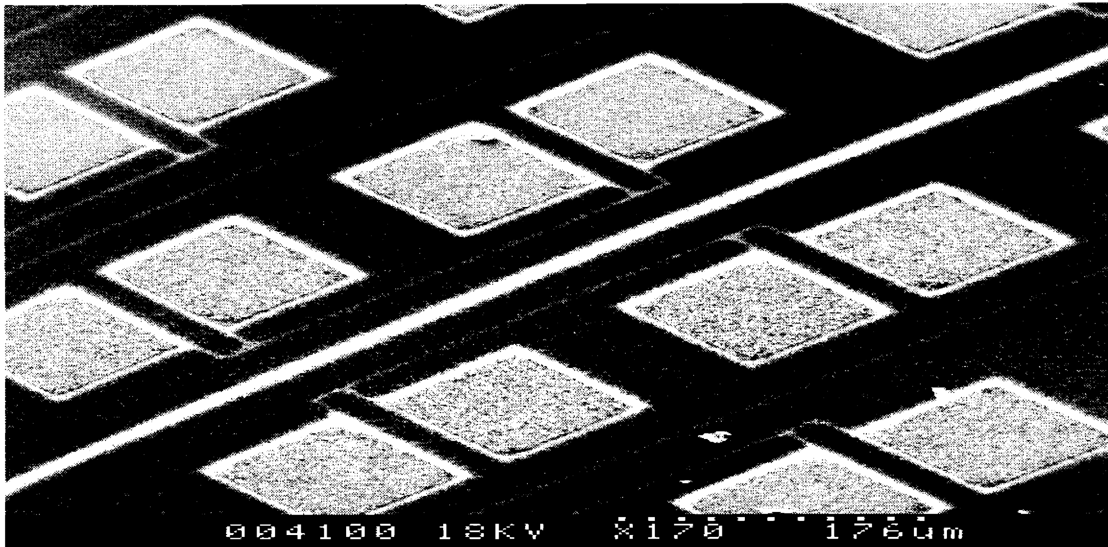


Figure 6-7. Optical microscope image of several interconnects.

(a)



(b)

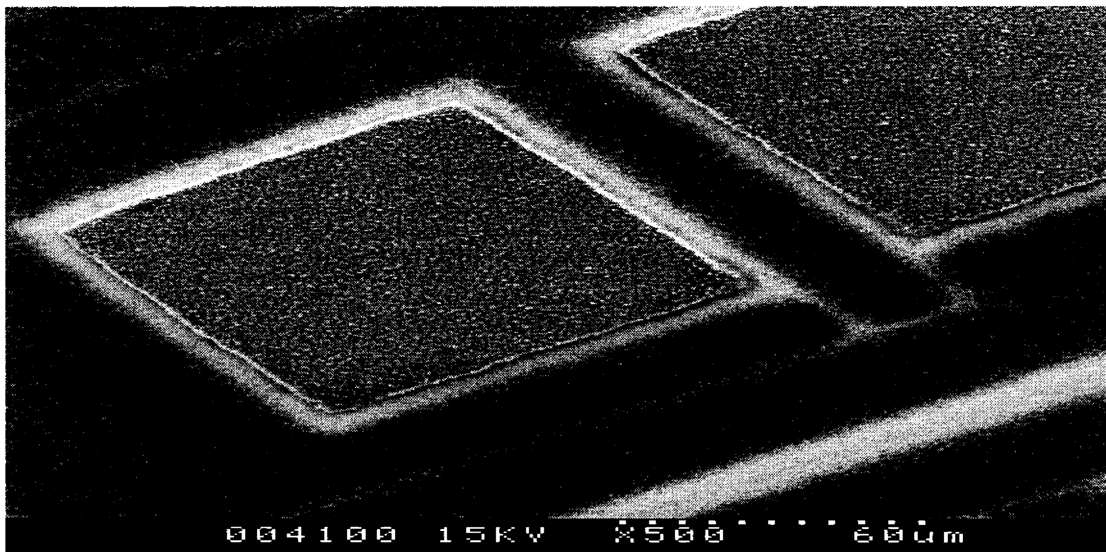


Figure 6-8. SEM image of an interconnect structure (a). A close-up of the interconnect is shown in (b). The bright white fuzziness is presumably due to the Ta.

For a final verification of the initial structure, the Wyko profilometer was used to image the interconnects. (see Fig. 6-9). The interconnect topography is apparent from this image, and supports the precision of the fabrication process. The "vertical bars" present above the pads, which represent the signal return of the specimen surface, are due to the surface roughness of the pads. Some facets along the surface of the pads return a signal to the detector while others reflect the light away from the detector, resulting in the formation of the "vertical bars".

Joule Heating Tests

Joule heating tests were conducted on every wafer to ensure adequate temperature control. Test results demonstrated joule heating contributed less than a five degree increase for a broad range of current densities (see Fig. 6-10). This seems reasonable given the small size of the interconnects and their proximity to significantly larger pads (100μm x 100 μm) which act as radiators to the probe tips and the air. Similar results were observed for unpassivated structures.

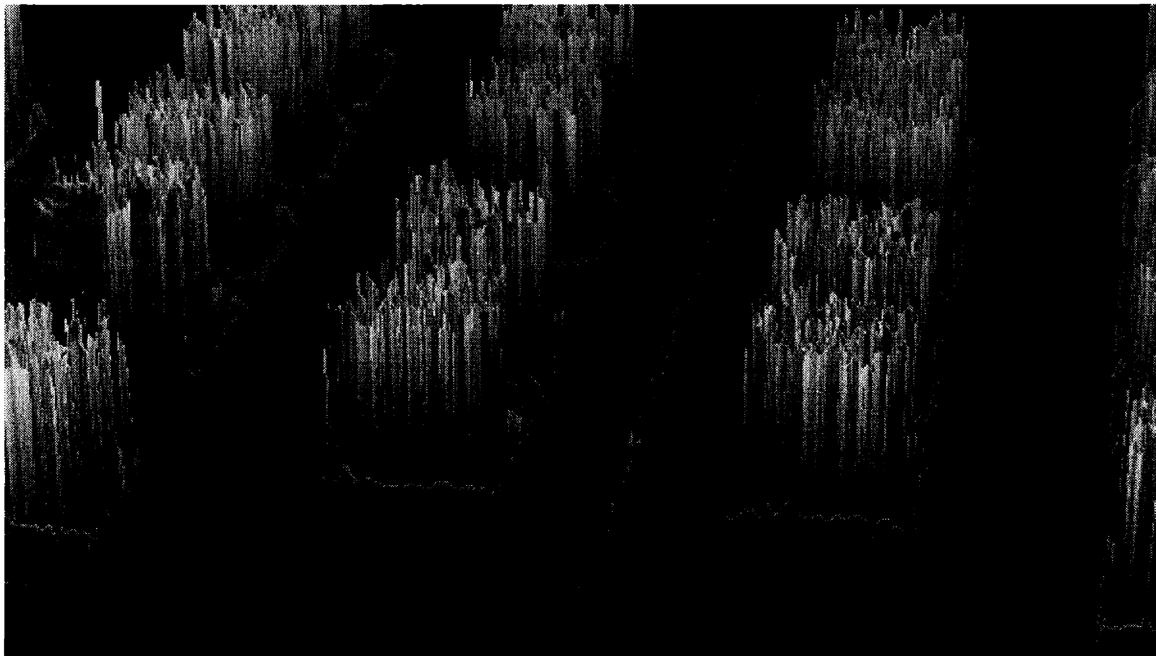


Figure 6-9. Interferometric profilometry image of the interconnect test structure.

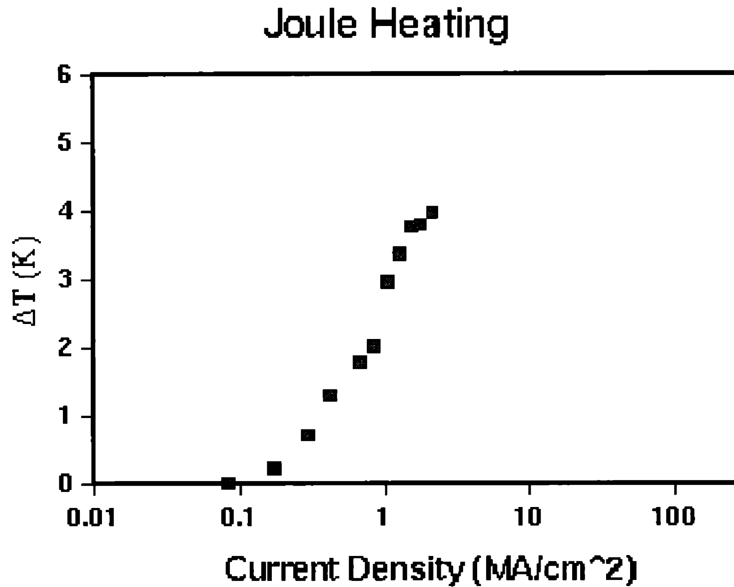


Figure 6-10. Joule heating tests demonstrate that the temperature rise in the interconnects is not significant.

Joule heating simulations were also conducted to investigate the effects of joule heating on the temperature profile of the interconnect. The simulation details are described in the previous sub-section. The Ta artificial studs occur at positions between 15 and 25 μm, and between 75 and 85 μm. The Cu interconnect is 50 μm long, and is situated between the Ta studs. The dimensions of the metals runners on either side of the Ta artifical studs correspond exactly to those defined in the fabricated structures. The simulation results indicate a 4.8°C overall increase in temperature, which is reasonably close to the joule heating test results in Fig. 6-10. The temperature profile for an interconnect near steady-state is shown in Fig. 6-11. These results show a fifteen degree increase in the temperature of the Ta artificial studs, where the majority of joule heating occurs due to geometric and electrical limitations of the Ta layer. However, because of the high thermal conductivity of Cu (which is expected given its high electrical

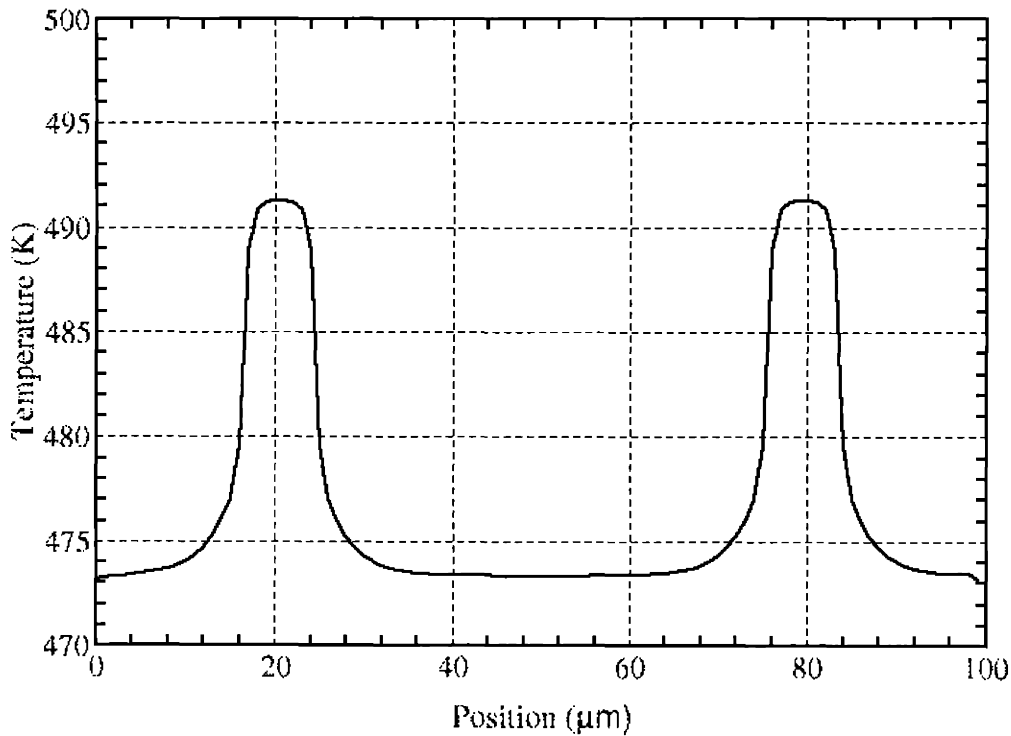


Figure 6-11. Temperature profile of the fabricated test structure determined by joule heating simulations.

conductivity), the heat is rapidly transported away to the heat sinks. At each stud, a temperature increase of no more than four degrees is present, which should not alter the diffusivity significantly. The maximum temperature gradient in the metal is 0.75 °C/µm, which will aid diffusion at the cathode and suppress electromigration at the anode. In comparison, the electromigration-induced stress gradient that forms at the anode and cathode suppress the electron wind.

To further assess the impact of Joule heating on the electromigration results, the experimental test conditions and results of the Joule heating simulations can be compared to the extensive studies in literature Schwarzenberger *et al.* studied the effects of localized increases in temperature on diffusivity and flux. [Sch 88, Ros 89] Specifically, because the resistivity of metals increases linearly for small increases in temperature, this

will cancel with the temperature term in the denominator of the electromigration flux equation (see Eq. 2.13). Therefore, the the dominant effect of temperature increases on flux divergence can be expected to relate to the effective diffusivity through an Arrhenius relationship with temperature, and the temperature dependence is dominated by the inverse of the temperature in the exponential term. Simulations by this group demonstrated that for 300 μm-long Al interconnects at 170°C, a rise in temperature by 10°C in a 150 μm-long zone centered along the length of the interconnect will result in a flux divergence-induced stresses of 25 MPa. However, the line lengths of the test structures used in our experiment were 25 and 50 μms, which is smaller than one-sixth that of Schwarzenberger. Because the maximum stress formed in an interconnect is linearly related to the line length due to back-stress effects [Cle 94], the additional flux divergence and stress evolution induced by the temperature gradient in our experiment can be expected to be small. Furthermore, the temperature gradient technique of Ross *et al.* [Ros 89] can be used to evaluate the effects of Joule heating on electromigration drift in Blech drift structures. In this work, it is suggested that an increase in temperature could lead to changes in the jL product measured in drift structures. As will be discussed later in this sub-section, the jL products measured in this experiment did not vary with test temperature ranging from 200°C and 300°C. From the work of Schwarzenberger, it is clear that a comprehensive analysis of the effects of Joule heating-induced temperature gradients requires modeling of the stress evolution resulting from these temperature gradients.

This analysis was performed by Clement *et al.*, who modified the Korhonen equation to account for a temperature profile that varies along the length of the interconnect such that $T(x)=T_o+\delta T(x)$. [Cle 94] The effects of temperature gradients in the Korhonen equation were isolated into a term, β, resulting in a stress evolution equation as follows:

$$\frac{\partial \sigma}{\partial t} = \frac{\partial}{\partial x}\left[\frac{D_o B \Omega}{kT_o}\left(\frac{\partial \sigma}{\partial x} + \frac{jez^* \rho}{\Omega}\right)\beta\right], \tag{6.3}$$

where

$$\beta(x) = \frac{1}{1+\delta T/T_o}\exp\left[\frac{\Delta H}{kT_o}\left(\frac{\delta T/T_o}{1+\delta T/T_o}\right)\right], \tag{6.4}$$

where ΔH is the activation energy of the dominant diffusion mechanism. The derivation of β is based on the substitution of D_a by D_o and the substitution of T by T_o in the denominator of the electromigration flux equation, resulting in the expression given in Eq. 6.4. It is noted that in this analysis, the resistivity was assumed to remain constant with increasing temperature, contrary to the work by Schwarzenberger. This assumption will be revisited later in this analysis. By solving Eq. 6.3 for several different temperature gradients and l_o/L ratios, where l_o is the length of the interconnect at raised temperature and L is the line length, the effects of thermal gradients can be determined. Because a step function is the worst-case scenario in terms of temperature-gradient-induced flux divergence, simulations were performed using dimensionless variables by Clement *et al.* to characterize the set of conditions at which temperature gradients could lead to significant stress build-up. Based on the temperature gradient and temperature increase determined by Joule heating simulations in this experiment, the β is determined to be 0.85. It is noted that in the calculation of β, the temperature gradient effects associated with diffusivity are shown to be far more significant than the temperature term in the denominator of the electromigration flux equation. In the work by Clement, the profiles of the maximum increase in stress are plotted as a function of l_o/L for several different values of β, with β=2 being the lowest. Because the gradient-induced stress decreases with decreasing β, the use of β=2 in the calculation performed using the Joule heating simulation data obtained in this thesis is conservative. For interconnects

terminating at zero flux boundary conditions such as studs, Clement's study of the maximum temperature-gradient-induced stress occurs at a l_o/L of 0.7, and drops rapidly to zero as l_o/L approaches either zero or one. From the Joule heating simulations performed in this thesis, it is apparent that the l_o/L ratio is 0.85. Based on these parameters, the maximum increase in stress assuming a step profile is determined to be 10.7 MPa. As will be shown later in this sub-section, tensile stresses in the interconnects tested in this experiment reach an average of 270 MPa. Therefore, it can be safely assumed that the additional stress that develops due to the Joule heating-induced temperature gradients are insignificant with regard to the measurements to be made in this experiment.

In the analysis by Clement, the resistivity was assumed to remain constant over the range of temperatures considered. [Cle 94] However, in other temperature gradient-induced electromigration flux models, the resistivity has been assumed to increase linearly with increasing temperature. [Sch 88]. The Clement model will now be modified to incorporate a temperature-dependent resistivity, such that $\rho(T)=\rho_o+\delta\rho(T)$ where $\delta\rho(T)$ is a linear function of temperature over the range of temperatures considered. Assuming a linear temperature gradient as before, the stress evolution expression in Eq. 6.3 can be modified to the following:

$$\frac{\partial\sigma}{\partial t}=\frac{\partial}{\partial x}\left[\frac{D_o B\Omega}{kT_o}\left(\frac{\partial\sigma}{\partial x}+\frac{jez^*\rho_o\left(1+\delta T/T_o\right)}{\Omega}\right)\beta\right], \tag{6.5}$$

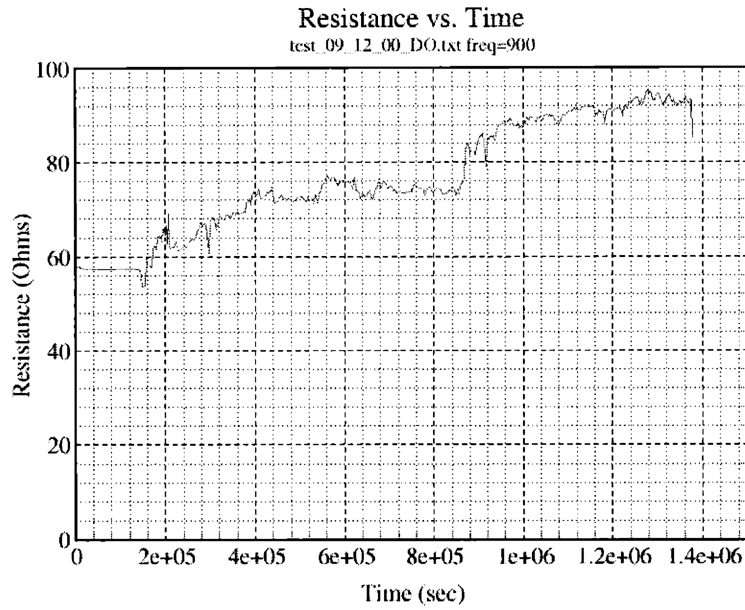
where β is defined as it was in Eq. 6.4. Using the results of the Joule heating simulations described earlier in this thesis, the $(1+\delta T/T_o)$ term has a value of 1.02. This indicates that a change in resistivity due to Joule heating in this experiment will increase the electromigration flux by only 2%. For comparison, a 2% variation in the lithographically-defined interconnect length is one micron, which is also within the

lithographic and etch error of the patterning process. Furthermore, based on an interpolation of β in the Clement results, a 2% error in β results in a error of stress estimation by less than 0.5 MPa. Clearly, the assumption of a non-temperature-dependent resistivity by Clement is valid, and that the effects of a temperature gradients on stress evolution is dominated by the exponential dependence on temperature of the effective diffusivity. Furthermore, based on the conservative analysis performed here, the temperature gradient in the test structure fabricated in this thesis results in a maximum increase in the stress evolution of 10 MPa, which is well below the stress that occurs near the artificial stud.

Electromigration Testing

Electromigration testing was initially conducted to observe and characterize the failure behavior. In most cases, when current is applied, the resistance remained constant for some incubation time, after which a void nucleated. The resistance was then observed to increase as the void grew, and eventually saturate for tests with sufficiently low current densities (0.5 MA/cm^2 or lower). Figure 6-12 illustrates this behavior for two such interconnects. Figure 6-12a shows the resistance measurement for a 10 μm wide interconnect with a length of 50 μm tested at 0.5 MA/cm^2 at a test temperature of 200°C. The resistance remains constant for an incubation period of 1.6E5 seconds (44.5 hours), after which the resistance increases due to void nucleation and growth. The void grows at a linear rate for another 8E5 seconds (222 hours) until void growth saturation occurs. In Fig. 6-12b, the resistance data is contained for a 8 μm-wide interconnect of length 25 μms is shown for a test conducted at 0.25 MA/cm^2 and 200°C. This data shows that the resistance begins to increase just after the current is applied, implying the presence of a pre-existing void at the cathode. The resistance increases at a linear rate until void growth saturation begins to occur at about 2E6 seconds (555 hrs). It is noted that the majority of efforts aimed at reducing the noise in the measured signal were unsuccessful. Line conditioners and uninterruptible power supplies were used to minimize the noise in the current applied, and all power supply and data acquisition equipment was connected

(a)



(b)

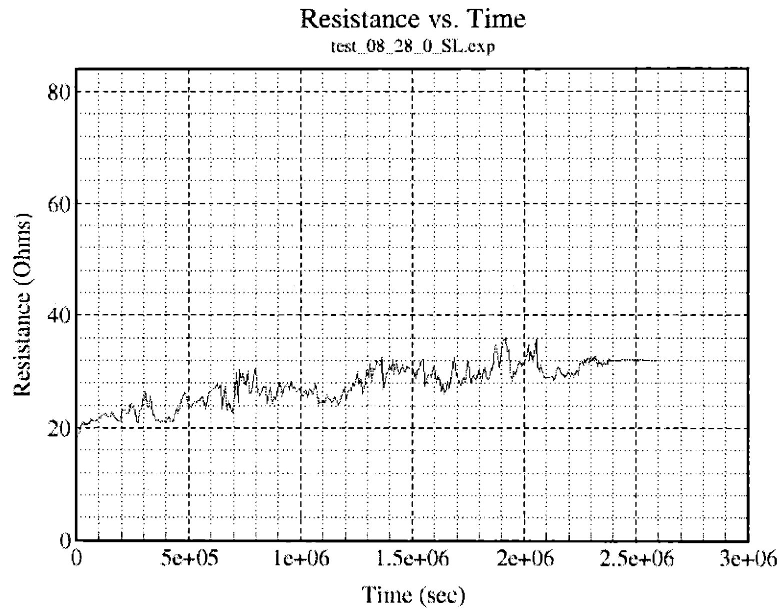


Figure 6-12. Typical resistance curves (a,b) observed for resistance saturation in interconnects. In (a), the resistance remains constant until a void nucleates, after which it grows linearly until reaching resistance saturation. In (b), void nucleation and growth begin at the start of the test, and continues until resistance saturation.

on a separate circuit from other noisy power consumption devices like hot chucks and vacuum equipment. Even voltage measurements conducted by directly contacting the probe tips to the highly conductive brass plate on the stage of the hot chuck also resulted in a similarly noisy data measurements, suggesting that at least some of the noise in the measured signal is not sample-related. In short, after using several approaches to reduce the noise in the measured signal, any remaining noise in the acquired signal is due to a combination of input line noise and possibly electromigration localized in the vicinity of the probe tip in the pads of the interconnect structure.

Higher current densities resulted in a spike in resistance associated with the failure of the SiNx dielectric (Fig 6-13). Although the resistance increases very rapidly at a near vertical rate, post-EM test resistance measurements using an ohmmeter showed that the resistance ultimately levels off in the range of 500-7,000 ohms in about half the interconnects. Because of limitations in automated voltage DAQ measurement, the resistance profile above 500 ohms in Fig. 6-13 could not be plotted. The massive increase in measured interconnect resistance that occurs after Cu failure arises because the Ta has a resistivity over two orders of magnitude higher than Cu, coupled with a thickness 75% smaller than that of the Cu. This rapid increase in resistance associated with rapid void growth in the Cu is presumed to occur due to dielectric failure. If the dielectric fails, the compressive stress at the anode is no longer constrained, leading to a reduction in the back-stress and possibly the extrusion of hillocks. Finally, it is noted that no analysis of resistance data above 750 ohms is performed in this analysis. For safety and experiment-related reasons, the power supply was programmed to discontinue any attempts to maintain constant-current conditions once the resistance increased above 750 ohms. Besides avoiding excessively high voltage conditions that could lead to electrical arc-ing, this programming limits further damage to the post-mortem interconnects from Joule heating. For these reasons, any data acquired after the resistance of the interconnect under test increases above 750 ohms is suspect, and is not considered in this analysis.

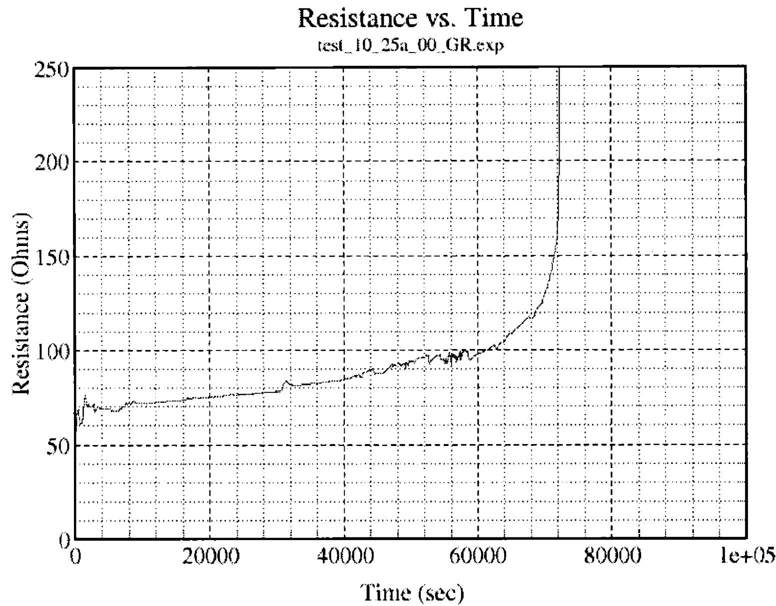


Figure 6-13. Typical resistance profile for an interconnect with dielectric failure, leading to a steep increase in resistance due to the high resistivity and small thickness of the Ta.

A interferometric profilometer image of a dielectric failure in one interconnect is shown in Fig. 6-14. A hillock is seen to form just to the right of the four pads at the left of the image. As the dielectric failed, the stress relief and continuing electromigration force resulted in the movement of Cu metal atoms to form this hillock.

Electromigration tests were performed under accelerated conditions at a range of temperatures of 200-300°C and current densities of 0.2-0.7 MA/cm^2 on diced wafers with controlled thermal histories. Over half of the unpassivated lines exhibited no net drift. The remainder of the unpassivated interconnects were observed to undergo void nucleation and growth, although void growth saturation was never observed. The outcome of the series of tests was not temperature dependent, nor was it dependent on thermal history. Possible causes for this behavior include the formation of an oxide on the exposed Cu and Ta, or the simultaneous drift of both ends of an interconnect and/or its connecting runners.

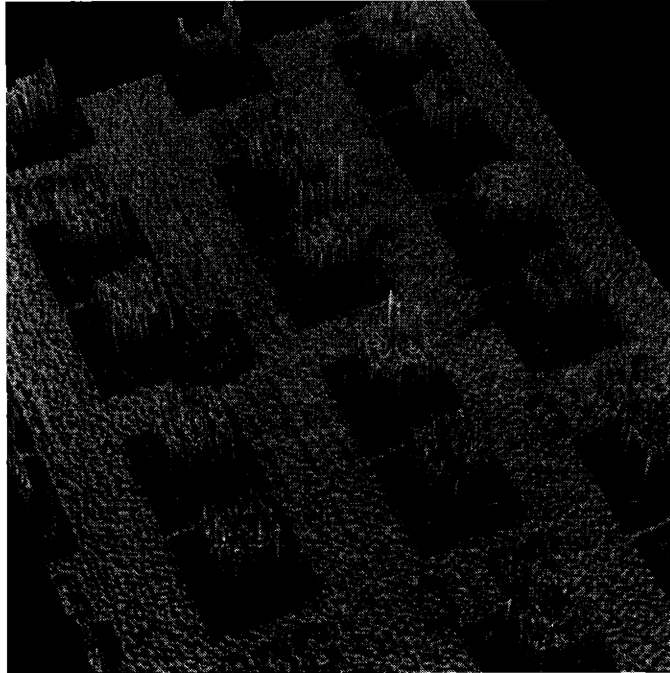


Figure 6-14. Profilometer image of failed interconnect, illustrating the formation of a hillock after dielectric failure.

In the (jL) product tests, a current density of 0.2 MA/cm^2 was sufficient to cause void nucleation and growth in 50 μm-long lines, but insufficient in 25 μm-long passivated lines, even at 300°C. The product of the current density and line length results in a (jL) product of 500 A/cm, which compares well with partially passivated Ta(20nm) / Cu(300nm) / Ta(80nm) strips of Lee who observed a (jL) product of 525 A/cm in strips 4 μm-wide with length 25 μms at 400°C. [Lee 95] A 100 nm thick layer of PECVD nitride was deposited above the metal strips (deposited on a W underlayer), followed by an RIE etch of the nitride and upper Ta layer to act as a mask for later ion milling of the Cu for strip length definition. The sidewalls of the metal remained covered by the nitride. These tests were conducted *in situ* in an high resolution SEM. Although the strips of Lee were conducted at 400°C compared to the 200-300°C used in this experiment, the jL product is not known to vary significantly with temperature in the range compared here (see Eq. 1.3). In another set of tests by Frankovic, unpassivated Cu tests strips on a TiN conductor in vacuum exhibited a (jL) product of 1200 A/cm at

180°C (see Table 6-I). In unpassivated lines tested in this experiment, drift was shown to occur in 25 μm-long, unpassivated interconnects at a current density of 0.2 MA/ cm^2. This indicates a (jL) product below 500 A/cm for unpassivated Cu strips. Because the tests by Frankovic were conducted in vacuum, which reduces the effects of Cu oxidation at the exposed Cu strip surfaces, it is not surprising that Frankovic observes a higher value of jL product compared the results of the unpassivated strips from this experiment. Using Eq. 1.3, which is restated below, the critical stress for void nucleation may be determined from the (jL) product and other experimental and materials parameters given in

$$\sigma_{max} = \frac{jLZ^{*}e\rho}{2\Omega}, \tag{6.6}$$

Table 6-I. Comparison of (jL) Product Values

Source	(jL) product in A/cm
This Study, Passivated by PECVD nitride, T=200 and 300°C	500
This Study, Unpassivated T=200°C	less than 500
K.L. Lee *et al.* [Lee 95] Ta/Cu/Ta strips partially passivated by PECVD nitride, T=400°C	525
R. Frankovic *et al.* [Fra 97] Unpassivated Cu/TiN strips in vacuum, T=180°C	1200

where Ω is the atomic volume for Cu. The atomic volume is determined by dividing the density of Cu by its atomic mass, and is equal to $1.19\text{x}10^{-29}$ /m^3. The resistivity for Cu was measured to be 2.7 Ω-cm based on a four point probe test. From Eq. 6.3, the critical stress for void nucleation in Cu is shown to be 270 MPa. Although this is less than that of Al, the higher modulus associated with Cu coupled with its lower adhesivity interface ease the nucleation of a void.

In lifetime tests, passivated interconnects were tested with current densities ranging between 0.25 and 0.75 MA/cm^2. For the three temperatures tested, incubation times ranged from 1.1-21.1 hrs. It was shown that incubation times decreased as temperature was increased (see Table 6-II). The author was unable to find any reported values of nucleation times in Cu, which is generally considered closely guarded information by most microfabrication facilities.

Interconnects tested at sufficiently low current densities exhibited void growth to saturation. The normalized increase in resistance at saturation has been theoretically derived to be: [Kor_2 93, Suo 98, And_2 99]

$$\frac{\Delta R}{R_o} = \frac{\rho_{sh}/A_{sh}}{\rho/A}\left(\frac{z_{eff}^{*} e\rho}{2\Omega B}\right) jL \ , \qquad (6.7)$$

Table 6-II. Comparison of Average Nucleation Times

Source	Ave. Nucl. Time in hrs
This Study, 200°C	21.1
This Study, 250°C	2.0
This Study, 300°C	1.1

where A is the cross-sectional area of the metal, A_{sh} is the cross-sectional area of the shunt, and z^*_{eff} is the effective z* for the overall interconnect. From Eq. 6.7, the effective z* for the Cu interconnects can be determined based on experimental measurements of the saturation resistance. The effective z* was calculated for the three different test temperatures, and the data is summarized in Table 6-III. Generally speaking, the effective z was observed to be about 4-7. The partially passivated Cu/Ta test strips of Lee demonstrated a z* of 6.4 while Gutmann observed a z* of 3.7-4.3 in passivated Cu. The test strips by Lee were described in detail in the discussion about jL products. Finally, it was noted that in this test structure, a pre-annealing step at 300°C seemed to increase the probability of dielectric failure relative to resistance saturation. This is believed to be due dielectric weakening from the difference in CTEs between Cu and nitride. Because resistance saturation was not observed in the unpassivated lines examined in this study, the effective z* for unpassivated lines could not be determined.

Table 6.III. Comparison of z* Values

Source	Z*
This Study, Passivated by PECVD nitride, 200°C	4.3 ± 1.2
This Study, Passivated by PECVD nitride, 250°C	6.1 ± 1.1
This Study, Passivated by PECVD nitride, 300°C	7.1 ± 2.2
K.L. Lee *et al.* Ta/Cu/Ta strips partially passivated by PECVD nitride,[Lee 95]	6.4
R.J. Gutmann *et al.* Passivated, [Gut 95]	3.7-4.3

The drift velocity and effective diffusivity may also be determined from lifetime test experiments. In Section 4.2, an equation for the normalized increase in resistance at saturation was defined in Eq. 4.3 as the product of a materials constant and normalized void size, and is restated here as:

$$\frac{\Delta R}{R_o} = \frac{\Delta L_v \rho_s h}{L \rho t_s} \quad . \tag{6.8}$$

Dividing both sides by Δt and letting Δt approach zero, Eq. 6.8 can be represented in terms of differentials. Further substitution of Eq. 4.2 for R_o, the void growth rate, $\partial L_v / \partial t$, may be determined. This is equivalent to the drift velocity, v_d, and is given by [Oat 91]

$$v_d = \left(\frac{\partial R}{\partial t} \right) \frac{2 h_{sh} W}{\rho_{sh}} \quad . \tag{6.9}$$

It is noted that the drift velocity is equal to the electromigration drift in the absence of a back-stress. Measurement of the resistance growth rate associated with void growth enabled the determination of the EM drift rate assuming full-spanning voids. Drift velocities measured for passivated and unpassivated interconnects are shown in Table 6-IV. These results do not suggest a significant difference in the drift velocities of unpassivated and passivated Cu interconnects, which is consistent with the those in the existing literature (see Fig. 6-15).

In Fig. 6-15, the literature drift velocities were normalized to a current density of 0.5 MA/cm^2 assuming a theoretically-derived current density exponent of n=1. Two sets of data [Hu 97, Hu 99] employ a resevoir-type structure and pertain to lines tested without passivation in vacuum (open circles and squares respectively). In the first set of structures, evaporated Cu lines were patterned by an e-beam lithography-based lift-off process, and situated above a tungsten resevoir section using Ti as an adhesion layer and

Table 6.IV. Comparison of Drift Velocities in This Study

Temperature	Drift Rate (μm/hr), Passivated Lines	Drift Rate (μm/hr), Unpassivated Lines
200°C	0.018	0.061
250°C	0.090	0.180
300°C	0.406	0.250

for reduced contact resistance. The Ti(10 nm) / Ta(15 nm) / Cu(170 or 270 nm) / Ta(10 nm) structures had linewidths of 0.25, 0.5, and 1.0 μms and tested in a nitrogen ambient until failure, which was defined as a 30% increase in resistance. In the second set of samples, the linewidth of Ti(10 nm) / Ta(15 nm) / Cu(300 nm) structures ranged from 0.15 to 10 μms, and the void front was tracked by resistance measurements and SEM. Testing was performed in a 15 torr chamber pressure of high purity nitrogen, and failure was defined by an increase in resistance of 10%. In another batch of experiments by Lee, tests stripes of a Ta/Cu/Ta using a minimal 100 nm passivation are plotted as open diamonds. [Lee 95] The processing and test conditions for these test stripes was described in the discussion about jL products. In summary, all three sets of data for unpassivated lines exhibit a slightly slower drift rate than the experimental values measured in this thesis, approximately a factor of five. Also plotted on this figure are the data of Cu-polyimide damascene interconnects [Hu 95, Hu 98], which are fully passivated, and are represented by filled circles and squares respectively. In the first set of samples, two micron-wide interconnects had a length of 300 μm and structure Ta(20 nm) / Cu(300 nm) / Ta(20 nm) passivated in polyimide, in which the Cu was deposited by PVD and CVD. Testing was preceded by an annealing at 500°C for four hours in a nitrogen ambient while the test itself is presumed to have occurred in a vacuum environment (the exact test environment is never stated). In the second set of samples, the Cu dual-damascene interconnect structure was Ta(30 nm) / Cu(1000 nm) / Si_3N_4(100

nm) with a polyimide passivation. Testing was performed in a 20-50 torr environment of helium or forming gas, and drift rates measured based on resistance measurements. Finally, passivated test stripes were tested by Proost, and are plotted as filled diamonds in Fig. 6-15. [Pro 00] To fabricate these structures, a damascene process was used. Specifically, trenches were patterned in PECVD SiO_2, followed by an 80 nm sputtering of Cu onto 150 nm TaN and Ta. Cu was then electroplated one micron of Cu, and etched and planarized by CMP. The Blech structures were passivated by a 100 nm PECVD Si_3N_4 dielectric. In general, all of the passivated interconnects exhibited significant variability in drift rates amonst themselves, and were statisically indistiguishable from the unpassivated interconnect structures. In summary, the results from the experiments performed in this study do not confirm an effect of passivation on the drift rates, which is consistent with the existing literature. The drift velocities in this experiment were slightly faster but well within the range of error of those in the literature.

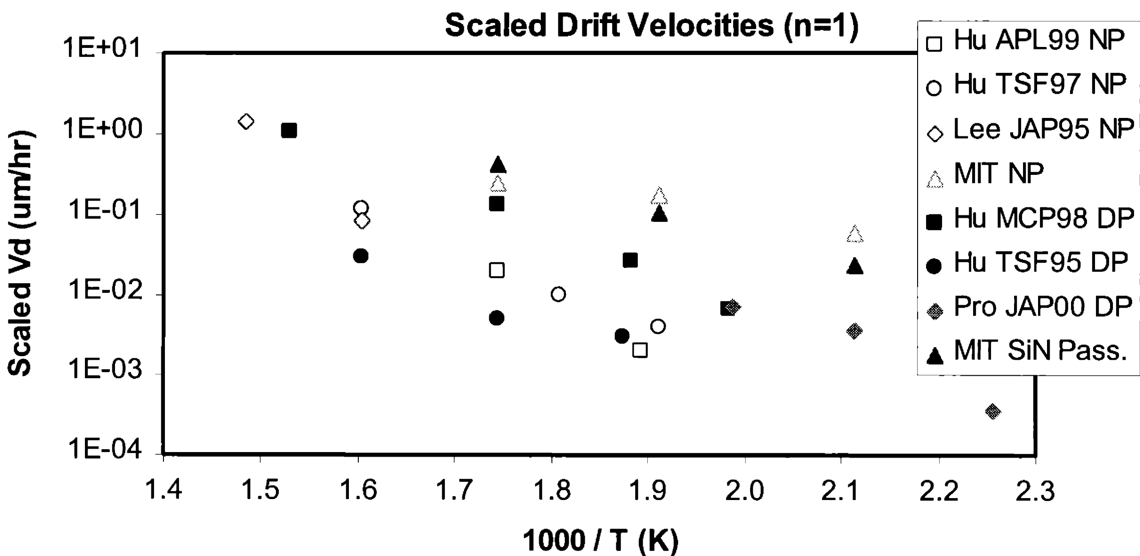


Figure 6-15. Drift velocities for the Cu/Ta structure along with those reported for other Blech strips and damascene interconnects. Open data points represent unpassivated (NP) structures while closed data points represent passivated interconnects. Damascene structures are denoted by DP. Open and closed triangles represent the unpassivated and Si_3N_4 passivated Cu/Ta structures, respectively, used in this study.

Given the drift velocity and effective z* at a particular temperature, the corresponding effective diffusivity, D_{eff}, may be determined. In the Korhonen model [Kor 93], the drift velocity is related to effective diffusivity by:

$$v_d = \frac{D_{eff}}{kT}\left(z^*_{eff} ej\rho\right) . \tag{6.10}$$

The effective diffusivities at each temperature were calculated using this equation, and are plotted in Fig. 6-16. Because diffusion is an activated process, a plot of the effective diffusivity versus the inverse of temperature yields the activation energy and diffusivity coefficient. For this experiment, an activation energy of 0.72 eV and a pre-exponential of 1.53×10^{-6} m^2/sec were extracted for passivated interconnects from this effective diffusivity data. This activation energy is consistent with literature, where 0.67-0.81 eV for surface/interface transport [Hu 95, Lee 95, Hu 97, Hu 98, Hu 99] and 0.95-1.2 for GB transport [Sur 94, Hu 96, Hu 98] has been observed., In some cases, it has observed

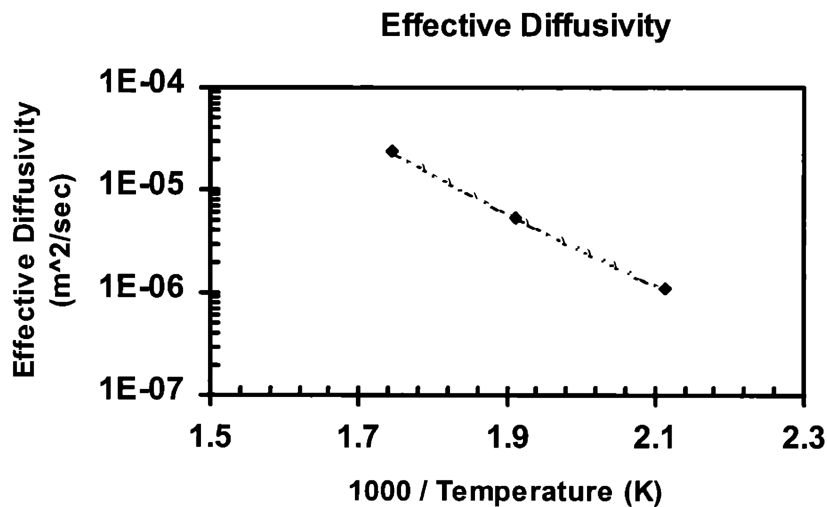


Figure 6-16. Effective diffusivity plotted against the inverse of time yields the activation energy and the corresponding diffusivity coefficient.

an activation energy of 1.1 eV in drift experiments. [Pro 00] The activation energy for bulk diffusion for Cu has been reported as 2.19 eV. [Llo 95, Gut 95] These drift velocities and diffusivities observed in this experiment are slightly higher than that of literature, and this is presumed to be due to this experiment's lower quality SiNx and its interface with the Cu interconnects. Finally, it is noted that the data for unpassivated interconnects was inadequate for determining the corresponding effective diffusivity data.

Hu previously detailed that the effective diffusivity of an interconnect was the summation of its weighted component diffusivity products, including the lattice, GB, surface, and (multiple) interfaces. [Hu 95] These diffusivity products, formed as the product of the effective diffusivity and z*, are weighted by their cross-sectional area of each diffusive area. For this thesis, Hu's analysis was extended to the test structures used in this experiment, based on the summation of the mass transport along the lower Ta interface and SiN interfaces at the top and sides. Grain boundary diffusion is neglected given the impurity levels present in this Cu. Similarly, lattice diffusion is also neglected based on the activation energy and test temperature. This results in an effective diffusivity-z* product of:

$$\left(D_{eff} z_{eff}^{*}\right) = \left(D_{SiN} z_{SiN}^{*}\right)\frac{\delta_{SiN}}{h} + 2\left(D_{SiN} z_{SiN}^{*}\right)\frac{\delta_{SiN}}{w} + \left(D_{Ta} z_{Ta}^{*}\right)\frac{\delta_{Ta}}{h}, \quad (6.11)$$

where D_{SiN} is the diffusivity at the nitride interface, z_{SiN}^{*} is the effective charge at the nitride interface, δ_{SiN} is the nitride interface thickness, D_{Ta} is the diffusivity at the Ta interface, z_{Ta}^{*} is the effective charge at the Ta interface, and δ_{Ta} is the Ta interface thickness. Although there are multiple unknowns, the individual $Dz^{*}\delta$ products associated with the Ta and SiN interfaces may be determined if the linewidth (or interconnect thickness) is varied by plotting the effective diffusivity product versus the

inverse of the linewidth. In MIT/EmSim simulations, only the product of the diffusivity, effective charge, and thickness of the interface is ever used, negating the need to determine these parameters separately. For this experiment, EM tests on 8 and 10 micron interconnects were used to attempt to isolate the individual Dz*δ products. By plotting the effective diffusivity-z* product versus the inverse of the linewidth, the slope yields:

$$\text{slope} = 2\left(D_{SiN} z^{*}_{SiN} \delta_{SiN}\right) . \tag{6.12}$$

and the intercept is given by:

$$\text{intercept} = \left(D_{SiN} z^{*}_{SiN} \delta_{SiN}\right)\frac{1}{h} + 2\left(D_{Ta} z^{*}_{Ta} \delta_{Ta}\right)\frac{1}{h} . \tag{6.13}$$

At 200°C, the Dz*δ product for SiN was calculated to be 1.5E-12 and the corresponding Dz*δ product for Ta was 1.2E-12. This implies a faster interface at the nitride, although the confidence in their relative magnitudes is limited because the difference is relatively small. At 250°C and 300°C, no such analysis could be performed because of excessive variation of data and limited data points. We are not aware of any published results reporting values for these component diffusivity products.

In summary, the critical stress for nucleation, effective z*, and the effective diffusivity for Cu interconnects have been determined from this work. Efforts to further define the Dz*δ individually for the Ta and Si_3N_4 interface were inconclusive, and require further redesign of the test structure (see Sec. 8.2 about Future Research). The resistance profiles measured in this experiment verify that much of the phenomenology present in Cu interconnects is similar to that in Al-based interconnects. The Cu interconnect input values obtained from this experiment allow MIT/EmSim to be adapted to accurately model electromigration in pure Cu interconnects.

With this Cu electromigration model, MIT/EmSim will be used in the next chapter to explore the interconnect reliability phenomenology of Cu interconnects, and ultimately, to generate a failure mechanism map for cataloging pure Cu interconnect reliability. While the Cu electromigration parameters can be expected to vary between different microfab because the Cu interfaces are very sensitive to minor process variations, the electromigration parameters measured in this experiment will permit an investigation of the fundamental Cu reliability behavior and void phenomenology, and the essential determination of the accuracy of scaling methodologies currently being extended from Al to Cu. The fundamental Cu electromigration phenomonelogy to be demonstrated in Chap. 7 will also be observed in industrial fabs; and in the future, industrial fabs can substitute Cu electromigration parameters corresponding to their own process to obtain accurate electromigration-induced failure time predictions. Finally, the Cu electromigration parameters measured in this experiment will enable a comparison of Al- and Cu-based technologies, and more importantly, the limits of such comparisons.

6.4. Conclusions

A viable process for fabricating Cu/Ta interconnect structures encapsulated in silicon nitride suitable for electromigration testing has been demonstrated using MTL's microrabrication facilities at MIT. A subtractive-etch process, rather than the more conventional damascene process, was used to define the interconnects because of limitations in the availability of a Cu-CMP process capability. Initial surface characterization was performed to inspect and verify the Cu interconnect structure. These techniques included optical microscopy, scanning electron microscopy, and interferometric profilometry. The latter was also used to examine the formation of hillocks in interconnects exhibiting a dielectric failure. Joule heating tests were conducted prior to electromigration testing to measure the amount of resistive heating in the interconnect, which was generally less than five degrees Celsius. Joule heating simulations demonstrated that the majority of heating occurred at the Ta artificial studs,

with minimal heating of the Cu. Furthermore, the temperature rise and temperature gradient present in the Cu was insufficient to have a significant effect on Cu electromigration, which is in agreement with the significant studies in the literature.

Electromigration tests were conducted at temperatures ranging from 200-300°C at current densities between 0.25 and 0.75 MA/cm^2. In these tests, a constant current was applied, and the voltage drop over the interconnect recorded at one second intervals. During testing, the resistance of passivated interconnects was observed to increase after a specific incubation time, where incubation time was observed to decrease with increasing temperature. After a void nucleates, the resistance increased until the resistance saturation was reached. For sufficiently high current densities, the dielectric failed, indicated by a sudden increase in resistance. This phenomenology is similar to that observed in the modeling and experiments involving Al-based technologies. More than half of the unpassivated Cu/Ta interconnects in this study did not drift while the remainder exhibited a void nucleation and growth process without void growth saturation.

From (jL) product tests, a (jL) product of 500 A/cm was measured, which compared well with the 525 A/cm reported in literature. [Fra 97]. A critical stress for void nucleation in Cu was determined to be 270 MPa. Resistance saturation tests demonstrated a z* value ranging from 4-7. Void growth tests were conducted to determine drift rates and diffusivities. The experimental results reported in this study do not confirm an effect of passivation on drift rate, which is consistent with the existing literature. Drift velocities measured in this experiment were shown to be about a factor of five times faster than that in the literature, but well within the error of the literature. The activation energy for the effective diffusivity was measured to be 0.72 eV, and its corresponding coefficient was 1.5×10^{-6} m^2/sec. Attempts directed at determining the $Dz^*\delta$ individually for the Ta and Si_3N_4 interface were inconclusive, and will require some redesign in the next generation of the test structure. The Cu interconnect input

values measured in this experiment, including critical stress for void nucleation, effective diffusivity, and effective z*, will assist in the adaptation of MIT/EmSim to accurately predict interconnect reliability in pure Cu interconnect systems. The MIT/EmSim model as revised for Cu will enable an investigation of fundamental Cu reliability behavior. While industrial microfabs will measure different Cu electromigration parameters due to the sensitivity of the Cu interface to process variations, this fundamental behavior will still be observed in industrial microfabs. Furthermore, in the future, industrial microfabs can substitute their own Cu electromigration parameters into MIT/EmSim to obtain accurate predictions of Cu interconnect lifetimes.

Chapter 7

Simulation of Pure Cu Interconnects

7.1. Introduction

In Chapters 2-5, the MIT/EmSim model was developed to simulate stress evolution and the void nucleation and growth processes in pure Al and Al-Cu alloys. The current density scaling behavior in interconnects was characterized for lines for different failure conditions based on line length and current density. In lines with a sufficiently small length and current density, void growth saturation was observed. This complex reliability behavior was catalogued in the form of a failure mechanism map. While failure maps for Al and Al-Cu were constructed using MIT/EmSim, no failure map could be determined for Cu interconnects until a Cu electromigration model is developed. Specifically, the MIT/EmSim model must be extended to include Cu interconnects by obtaining vital materials parameters and ensuring experimental verification of this new model. The experiments detailed in Chapter 6 led to a measurement of several of these parameters, and serve to help build the MIT/EmSim electromigration model for pure Cu interconnects. This Cu electromigration model will be used in this chapter to investigate the reliability of Cu interconnects, and to compare the reliability of Al and Cu interconnects. Finally, the effects on reliability of removing the diffusion barriers of Cu damascene interconnects will be studied.

7.2. Model Parameters

Inherent to the MIT/EmSim model are several materials-specific parameters. These include the: atomic volume, effective modulus, resistivity, critical stress for void nucleation, effective diffusivity, effective z*. In addition, based on the test structure, input of the shunt resistivity and shunt thickness are necessary.

The atomic volume is determined by dividing the density of Cu by its atomic mass. For Cu, this yields a value of 1.19×10^{-29} /m^3. The effective modulus can be estimated based on the analysis by Korhonen to be 120 Gpa. [Kor 93, Gar 95] The resistivity of damascene Cu after a post-deposition anneal to promote stagnation of the microstructure has been reported to be 2.3 Ω-cm [Hu 95]. From jL product tests in Chapter 6, the critical stress for void nucleation was measured to be 270 MPa. Similarly, the effective diffusivity was measured in Chapter 6 to have an activation energy of 0.72 eV and an coefficient of 7.1x10-6 m^2/sec. Also, z* was measured to be 5. For typical shunt structures in damascene Cu, the shunt thickness is 20 nm and the shunt resistivity (Ta) is 200 Ω-cm. [Hu 95] These materials input values were entered into the MIT/EmSim model for Cu, and the dominant (only) diffusive species was set to Cu atoms and vacancies. Because current damascene technology retains a rigid oxide inter-layer dielectric, the rigid matrix assumption inherent in the MIT/EmSim model is still valid. As a side note, during simulation runs, other Al-based technology portions of MIT/EmSim code were switched off to promote calculation efficiency.

7.3. Void Nucleation & Growth

The void nucleation and growth process was simulated in a 200 um Cu interconnect at a current density of 1×10^{6} A/cm^2. After the current is applied, a tensile stress at the cathode and the compressive stress at the anode increase in magnitude (t=100 hrs in Fig 7-1). Once the critical stress is reached, a void nucleates and grows (t=5,000

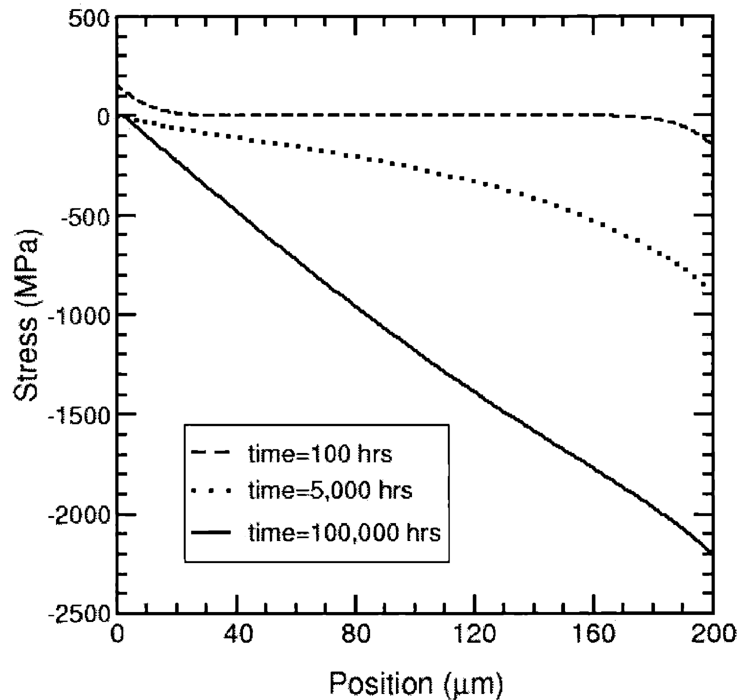


Figure 7-1. After the initial development of a tensile and compressive stress (t=100 hrs), a void nucleates and grows (t=5,000 hrs) until void growth saturation is reached (t=100,000 hrs).

hrs in Fig 7-1). For sufficiently short interconnects tested at an adequately low current density, void growth slows and eventually saturates (t=100,000 hrs in Fig7-1). This behavior is similar to that of Al-based interconnects, although the lifetimes and stresses achieved are different.

7.4. Current Density Scaling

In Section 4.3, the currrent density scaling behavior applicable to Black's equation was examined for Al-based interconnects. It was demonstrated that at high current densities, failure was void-growth-limited exhibiting n=1 scaling while at lower

current densities, n=2 scaling consistent with void-nucleation-limited failures was observed. The current density scaling behavior of Cu interconnects is now studied by simulations using MIT/EmSim as extended for Cu technologies.

Electromigration testing of a populations of 200 μm long Cu interconnects was simulated over a range of current densities between 0.1 and 10.0 MA/cm^2. Failure was defined to occur when thenormalized void length L_v/L reaches 0.025%, and the lifetimes of each interconnect determined. As shown by Fig. 7-2, at high current densities, Cu interconnects exhibit void-growth-limited failures consistent with n=1 scaling while at lower current densities, failures are void-nucleation-limited exhibiting n=2 scaling. This current density scaling behavior of Cu interconnects is identical to that observed for bamboo Al-based technologies (Al data replotted from Fig. 4-3). A study of the relative lifetimes of each technology in Fig. 7-2 illustrates that at higher current densities, at which accelerated electromigration tests are conducted, Cu interconnects exhibit an increase in reliability compared to Al, which is consistent with what others have reported. [Ari 94, Hu 98]. However, at lower current densities, the benefits of Cu over bamboo Al interconnects are not as apparent. The difference in lifetimes between Al and Cu interconnects changes because Cu interconnects exhibit a lower transition current density of 0.7 MA/cm^2 relative to Al, whose j_{tr} occurs at 3.0 MA/cm^2. This results in the current density exponent of Cu increasing to n=2 at a lower current density than Al, leading to a reduction in the difference in lifetimes. Once again, this demonstrates the importance of ensuring the current density exponent does not change when using Black's equation to scale accelerated lifetime data to service conditions. More importantly, Fig. 7-2 highlights the observation that although Cu has been reported to exhibit enhanced reliability over Al at accelerated conditions, this may not be the case at service conditions depending on the value of the relative transition current densities between failure modes for each metal. As the failure mode changes, the current density exponent also changes, resulting in a change in the relative lifetimes of Al- and Cu-based interconnects. The current density at which the failure mechanism changes is a function of the line length,

current density, and failure resistance. For this reason, the difference in lifetimes between Al and Cu interconnects at accelerated and service conditions will also depend on the line length, current density, and failure resistance. This suggests that comparisons of the reliability of Cu and Al at accelereated conditions cannot be extrapolated to service conditions without additional simulation or analytic techniques. That is, comparisons of accelerated electromigration test data of Al and Cu interconnects are insufficient to estimate similar comparisons at service conditions.

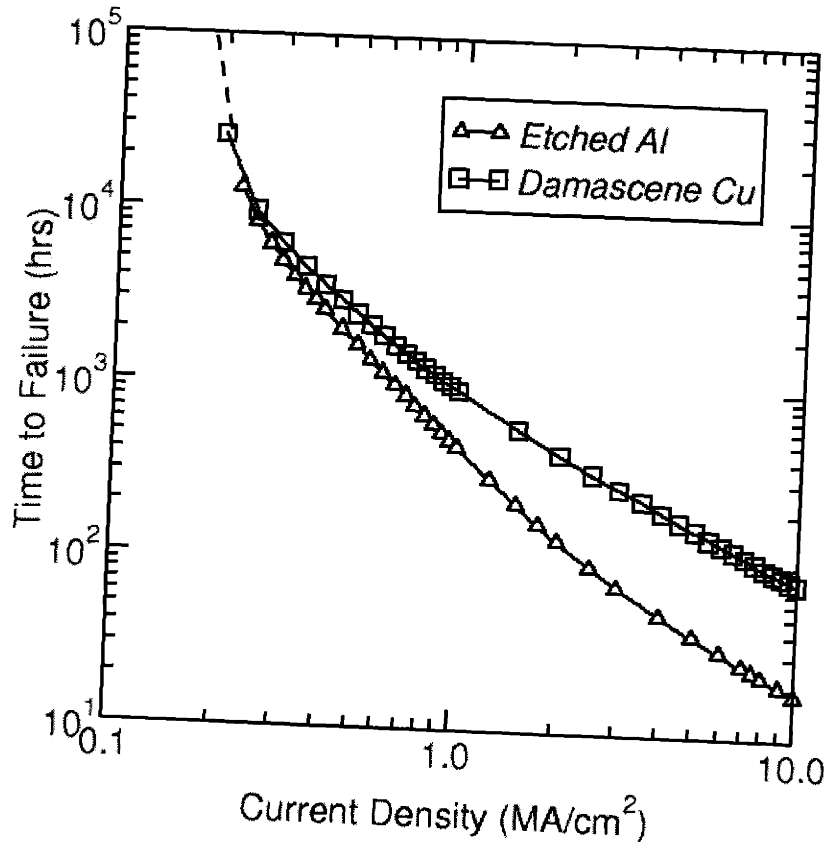


Figure 7-2. Current density scaling behavior of Cu interconnects is compared to that of bamboo Al-based technologies. Al data is replotted from Fig. 4-3.

Recent research in damascene processing has alluded to the removal of the Si_3N_4 diffusion barriers situated at both ends of Cu interconnect (see Fig. 6-2). In current damascene design, this Si_3N_4 layer serves to limit Cu migration into undesired areas and to act as an etch stop for dielectric patterning. Removal of this barrier would lead to improved fill of Cu studs and adhesion of higher metal levels, in addition to a slightly reduced resistance. However, the Si_3N_4 barrier serves a vital role in limiting electromigration-induced damage, and ultimately Cu interconnect reliability. From Fig. 7-2, it is observed that interconnect lifetimes rapidly increase at low current densities (approximately 0.2 MA/cm^2) as an interconnect becomes immortal, either without void nucleation or by void growth saturation. These interconnect immortality fates occur as a direct result of the compressive back-stress hampering the electron wind driving stress evolution and/or void growth. Without the Si_3N_4 diffusion barriers acting as blocking boundaries at either end of the line, interconnects will more likely suffer void growth to failure. In effect, the line lengths of Cu interconnects would be drastically increased, in many cases eradicating the beneficial immortality fate. In Fig. 7-3, the Cu lifetime data from the previous figure is replotted, extrapolating lifetimes at lower current densities for the case in which interconnects are no longer immortal. Comparing Fig. 7-2 to Fig. 7-3, it is observed that at low current densities, the reliability of Cu interconnects has been significantly reduced because of the removal of the diffusion barrier. In summary, short length effects are beneficial for promoting Cu interconnect reliability, and the removal of Si_3N_4 diffusion barriers will sacrifice the benefit of interconnect immortality.

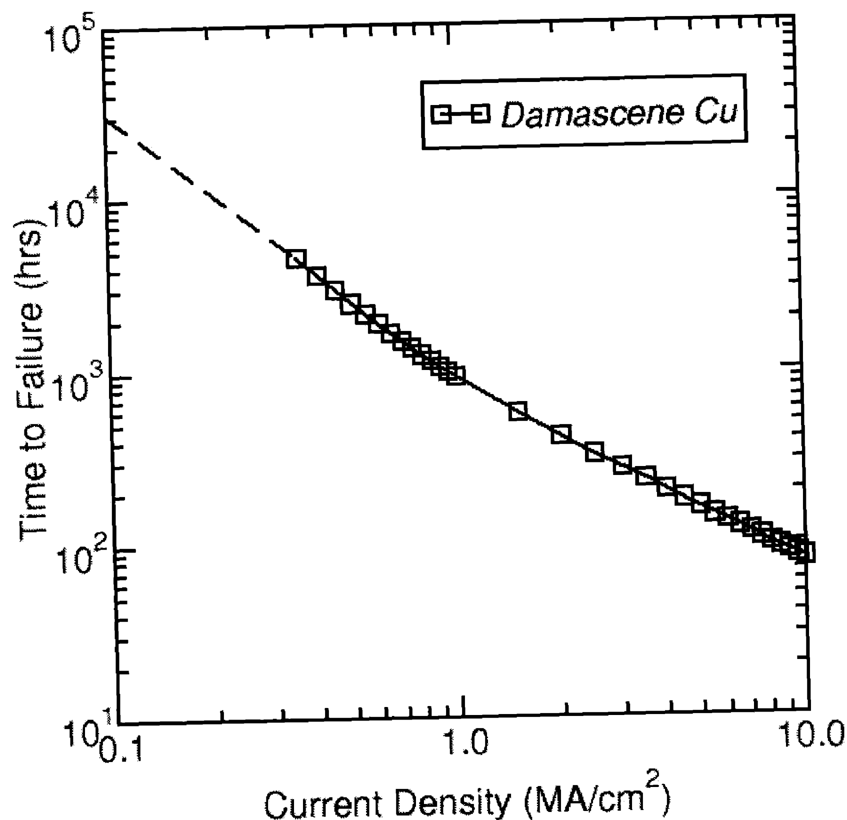


Figure 7-3. The reliability of Cu interconnects at low current densities is significantly reduced upon the removal of the Si_3N_4 diffusion barriers.

7.5. Cu Failure Mechanism Maps

Copper interconnects ranging in length from 25-200 μms at a test temperature of 200°C were simulated at current densities ranging between 0.1 and 10.0 MA/cm^2. A conventional damascene test structure was assumed, with a Ta shunt present below and at the sides of the Cu and a Si_3N_4 layer above. The thickness of the Ta layer is assumed to be 20 nm, and its resistivity is 200 Ω-cm. [Hu 95] Assuming a resistance failure criterion of 25%, the lifetime of each interconnect was determined and catalogued in the form of a failure mechanism map (Fig. 7-4). From this map, it can be observed that the span of the

immortality regime without void nucleation (triangles in Fig. 7-4) is slightly smaller than that of Al at the same failure condition (see Fig. 5-3), balancing the competing effects of copper's decreased critical stress for nucleation, increased effective modulus, and reduced resistivity. No resistance saturation regime is observed, which was also the case with the Al technology. Also, the span of the void-growth-limited regime exhibiting n=1 scaling (squares in Fig. 7-4), for which Black's equation is valid, is observed to be larger than that in Al technologies. This is primarily due to the increased effective modulus of Cu and the lower critical stress for nucleation, which is countered by copper's higher resistance shunt structure. As discussed in Chap. 5, this map can be used to evaluate other interconnect structures since the map in Fig. 7-4 also represents a failure map assuming a 50% resistance failure condition with a shunt layer 50% thinner based on Eq. 4.4.

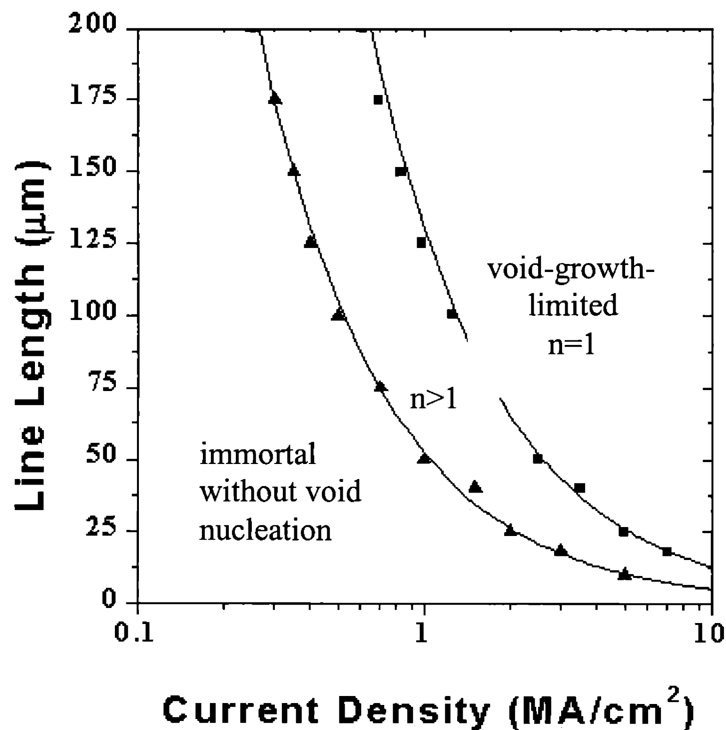


Figure 7-4. Failure mechanism map of Cu interconnects assuminga 25% resistance failure criterion. Data obtained by simulation is marked by data points, while analytical approximations are represented by solid lines.

In Section 5.4, a means of constructing failure mechanism maps for simple-geometry, pure Al interconnects was demonstrated based on analytical approximations. In this section, this analysis is extended to Cu interconnect technologies. In the experimental results discussed in Sections 6.4 and 7.2, along with published research [Hu 95], the void nucleation and growth until process has been repeatedly demonstrated for Cu technologies. The theoretical (jL) product associated with immortality without void nuceation, defined by Eq. (1.4), is overlaid on Fig. 7-4 (solid line) and is shown to be consistent with simulation. The void saturation regime, defined by Eq. (C6), also was observed to be situated directly above the (jL) pruduct line, negating the possibility of a void saturation regime. This is consistent with the simulation results shown in Fig. 7-4. The line indicating the void-growth-limited regime, defined by solving Eqs. (5.1, 5.4, 5.5), is plotted in Fig. 7-4 and is reasonably consistent with simulation. The closeness of fit of this line is better for Cu interconnects than Al technologies, which is presumably due to the lower critical stress for void nucleation in Cu interconnects. This results in a smaller tensiles stresses present in Cu, and less error associated with the analytical approximation neglecting the stress effects on diffusivity.

The appearance of a failure mechanism map will also be significantly altered by the removal of Si_3N_4 diffusion barriers. Specifically, the immortality region will no longer be present -- instead, a void-nucleation-limited region with n=2 scaling will occur in its place. Short lines tested at low current densities, which are immortal in the presence of diffusion barriers at either ends of an interconnect, will now fail by void-nucleation-limited mechanism. This family of interconnects will now require significant additional reliability analysis by both accelerated electromigration tests. Furthermore, as indicated in the previous sub-section, comparisons of Al and Cu reliability at accelerated conditions are difficult to extrapolate to low current densities, necessitating additional simulation and analytic techniques while also reducing the confidence in lifetime predictions. The removal of the Si_3N_4 diffusion barriers will significantly complicate

reliability predictions of short lines tested at low current densities, and will also be difficult to determine based on extrapolations of accelerated electromigration test data. Finally, the evaluation of interconnect reliability will be required to transcend multiple metal levels at once to adequately divide the multiple levels into multi-level tress, the revised fundamental unit of reliability.

In sumary, a failure mechanism map for Cu/Ta technology has been constructed by simulation. The use of analytical approximations discussed in Sec. 5.4 has been utilized for the rapid construction of failure maps for simple Cu interconnect structures. Cu failure maps exhibit a smaller immortaliy regime, while the void-growth-limited regime spans a larger span of test conditions. As new interconnect technologies such as Cu alloys and Ta-based shunts and diffusion barriers are developed, an understanding and prediction of the reliability behavior will be necessary. This can be obtained by the investigation of new Cu failure mechanism maps, constructed by either simulation or analytical means, to correspond to these advanced interconnect technologies.

7.6. Future of Electromigration in Context

This thesis has focussed on developing models for predicting electromigration-induced failure times, for the design and execution of experiments to measure relevant electromigration parameters, and for the evaluation of the Black's equation design rules for interconnect layouts. However, as interconnect linewidths continue to scale downward, there exist additional issues that may limit future scaling including Joule heating and RC delay. In Table 7-I, select elements of the Semiconductor Industry Association (SIA) Technology Roadmap are listed. [SIA 99] This data will be used in this sub-section to estimate the relative importance of electromigration, Joule heating, and RC delay on interconnect design. Although the SIA roadmap projects technology requirements well beyond 2004, the accuracy of projections beyond five years is suspect, and therefore will not be considered in this analysis.

Table 7-I. Subset of SIA Technology Roadmap

Element	Y2000	Y2001	Y2002	Y2003	Y2004
Technology	180 nm	180 nm	150 nm	150 nm	150 nm
# of metal levels	6-7	7	7-8	8	8
J_{max} at 105°C (MA/cm^2)	0.71	0.80	0.96	1.10	1.30
I_{max} at 105°C (mA)	0.36	0.33	0.32	0.29	0.27
Local wiring pitch (nm)	450	405	365	330	295
Intermediate wiring pitch (nm)	575	520	465	420	375
Global wiring pitch (nm)	945	850	765	690	620
Local wiring A/R in Cu	1.4	1.5	1.5	1.6	1.6
Intermediate wiring A/R in Cu	2.3	2.4	2.5	2.6	
Global wiring A/R in Cu	2.3	2.4	2.5	2.6	2.7
Conductor resistivity (μΩ-cm)	2.2	2.2	2.2	2.2	2.2
Barrier cladding thickness (nm)	16	14	13	12	11
Effective dielectric constant	3.5-4.0	2.7-3.5	2.7-3.5	2.2-2.7	2.2-2.7

Note: A/R = Aspect Ratio. Adapted from SIA Technology Roadmap [SIA 99]

The RC delay of an interconnect is given by the product of the resistance of the metal and capacitance of the dielectric (see Chap. 1). In most semiconductor chips, the spacing between interconnects is almost an order of magnitude larger for global interconnects over local interconnects, although Table 7-I indicates a *minimum* pitch comparison differing by a factor of two. Furthermore, the linewidths of local interconnects are also about an order of magnitude narrower than global wiring. [Boh 96]

For these reasons, RC delay is primarily a concern for local interconnects, and of minimum consequence for global wiring with the exception of very long interconnects used for clock distribution. In fact, scaling of linewidths of local interconnects has been slowed because of concerns about RC delay, and for this reason, the remainder of this analysis of RC delay focuses on local interconnects. Based on Table 7-I, the resistivity will remain constant while the linewidth decreases 15% over the next five years, leading to an increased resistance. Although the spacing between interconnects will decrease by 25%, the dielectric constant decreases by about 40%, leading to a decrease in capacitance of 20%. The overall RC delay for local interconnects can then be projected to decrease by approximately 5%.

Joule heating is determined by the product of the current squared and the resistance. While the resistance is expected to increase by 15%, the maximum current will decrease by 25%, leading to a net decrease of Joule heating by 35%. The effects of Joule heating also depend on the efficiency of heat transport away from the interconnects. For simplicity, this Joule heating analysis assumes that the heat generated in the active device region due to transistor switching is conducted away by the Si wafer to the exterior packaging. The Si wafer can also be expected to draw away heat from the first level of local interconnects, since these interconnects are electrically connected to the active device region by either tungsten or Cu vias, which are both effective heat conductors. Similarly, the uppermost level of interconnects is also electrically connected to the external chip interconnects through ball grid array or lead frame packaging, which act as a heat sink for this level of interconnects. For these reasons, Joule heating would appear to be of greatest concern for the intermediate-level interconnects (which are increasing in number as indicated in Table 7-I). The transport of heat from these interconnects will depend on the thermal conductivity of the shunt, barrier layer, and inter-layer and intra-layer dielectric. Because the thermal conductivity of Ta is twice that of Si_3N_4 (see Chap. 6) coupled with its higher interface area, the majority of heat removal will occur through the Ta shunt. It is noted that the Ta cladding interfaces with the

bottom and sidewalls of the Cu, which given the aspect ratios for intermediate-level interconnects in Table 7-I, results in the Ta-Cu interface covering 85% of the Cu surface area. It is also noted that the Ta cladding thickness projections decrease by 30% in the next five years. Ultimately, the heat must be dissipated into the dielectric, and the efficiency of this heat tranport will depend on the thermal conductivity of the dielectric to the electronic packaging. As is apparent from Table 7-I, the industry is planning to replace oxide by low-k dielectrics in the near future. These candidate low-k dielectrics include fluorinated oxides, polmer-based dielectrics such as polymides, aerogels, and air cavities (listed in order of decreasing thermal conductivity). The effects of the (reduced) Joule heating in the future will depend on the choice of low-k dielectric and the thermal transport efficiency of this medium, although in general, it can be expected that Joule heating will become of increased concern in five years.

Electromigration has typically been of greater concern in global interconnects in semiconductor chips (see Chap. 1) than intermediate-level or local interconnects. As stated above, the scaling of linewidths of local interconnects has slowed in order to minimize increases in RC delay, limiting the maximum current density applied to these lines. Furthermore, local interconnects are generally short in length, leading to increased likelihood of interconnect immortality without void nucleation due to short length effects (see Sec. 7.4 and 7.5). If the damascene structure in present use is retained, electromigration in local interconnects will continue to be of minimal concern. The technological and economical drive for reducing linewidths in intermediate-level and global interconnects to help slow the increase in the number of metal levels leads to the application of higher current densities, which in turn, will lead to increased concern about electromigration. As current density increases, lifetimes can be expected to decrease according to the results shown in Fig. 7-2. In the simplest case, in which failure is void-nucleation-limited with a current density exponent of 2, the lifetimes will scale as j^{-2}. The benefits of Cu over Al-based technologies are not always apparent (see Sec. 7.4), which could result in increased difficulties in meeting the current density requirements of

the Technology Roadmap. Furthermore, electromigration-induced mass transport is exponentially-sensitive to temperature, and the net change in temperature due to Joule heating is uncertain, depending upon the thermal conductivities of future low-k dielectrics. These candidate low-k dielectrics are also mechanically weaker than oxide, which will hamper the positive effects of the back-stress, minimizing conditions supporting interconnect immortality while possibly leading to increased void growth. Finally, the alteration of the damascene structure by the removal of the Si_3N_4 diffusion barriers at the studs will also decrease the possibility of immortality (see Sec. 7.4), leading to increased concerns about electromigration at *local* interconnects in addition to intermediate-level and global interconnects. Futhermore, analysis of interconnect reliability will be required to transcend multiple levels, leading to the need for detailed electromigration analysis of numerous multi-level tree units. In summary, in the next five years, electromigration will be of significantly higher concern in *global* and *intermediate-level* interconnects as current densities are increased; furthermore, the removal of the Si_3N_4 diffusion barriers will reduce reliability, and further complicate the design of, and reliability estimates of, Cu interconnects.

In this sub-section, the future influences of RC delay, Joule heating, and electromigration on future semiconductor chip design over the next five years has been evaluated based on technology projections defined in the SIA roadmap. The RC delay, which is most significant in and limits linewidth scaling in local interconnects, will not change significantly. Joule heating will decrease 35%, although this will be very dependent upon the thermal conductivity of the future low-k dielectrics employed. Electromigration-induced damage will increase in intermediate-level and global interconnects as current densities rise, and will continue to be of minor concern in local interconnects. However, if mechanically-weaker low-k dielectrics are employed, electromigration will be of increased concern in global and intermediate-level interconnects. Finally, if the diffusion barriers are removed from the damascene structure, interconnect lifetimes will decrease as conditons favoring immortality are

reduced. In addition, interconnect design will become more complex, requiring the division of multiple metal levels into numerous multi-level tree units for individual reliability analysis.

7.7. Summary

The MIT/EmSim model has been extended to simulated the stress evolution and void behavior of Cu interconnects based on the experimental results of Chap. 6. The void nucleation and growth process until void growth saturation was simulated, and shown to be similar to that of Al-based technologies. Current density scaling phenomenology was examined in Cu interconnects by simulation, demonstrating void-growth-limited failures exhibiting n=1 scaling at high current densities. At lower current densities, void-nucleation-limited failures with n=2 scaling was observed. Because the transition in failure modes occurs at different current densities for Cu and bamboo Al interconnects, the superior failure times measured at accelerated conditions in Cu interconnects relative to bamboo Al may not necessarilybe evident at service conditions.

A failure mechanism map for damascene Cu/Ta interconnects was constructed based on lifetime data obtained from an array of simulations varying the current density and line length. The immortality regime for Cu interconnects was smaller than that of Al-based technologies due to the difference in their critical stresses for void nucleation. The void-growth-limited regime was larger for Cu technologies, indicating the wider range of applicability of Black's equation. The method for constructing failure mechanism maps based on analytical means was extended for Cu, and the error of the analytical approximations was smaller for Cu due to the smaller tensile stresses involved. Copper failure maps can be used to 1) serve as an overview of Cu interconnect reliability, 2) illustrate domain of accuracy of conventional current density scaling methodologies, 3) aid in the design of accelerated test conditions, and 4) to influence the design of, and evaluate the reliability of interconnect layouts. As new barrier and shunt technologies are

developed for Cu, failure mechanism maps may be constructed by either simulation or analytical means for these new interconnect technologies.

If the Si_3N_4 diffusion barriers are removed from the Cu studs at the line ends, the lifetimes of Cu interconnects will be significantly reduced to the loss of short line effects. In particular, interconnects will no longer be immortal, but rather will fail by void-nucleation-limited failures with n=2 scaling. Futhermore, because the extrapolation of comparisons of Al and Cu reliability at accelerated conditions to low current densities is uncertain without simultaneous simulation or analytical methodologies, increased uncertainty in lifetime predictions and overall reduced Cu interconnect reliability will result. Finally, the evaluation of the reliability of multiple metal levels will become more complex, requiring their subdivision into numerous multi-level tree units for individual reliability analysis.

The influence of RC delay, Joule heating, and electromigration on future interconnect design were evaluated and compared based on projections defined in the SIA Technology Roadmap. Over the past few years, RC delay has limited interconnect linewidth scaling of local interconnects, and this will continue to be the case in the future. Joule heating will be of greatest concern in the intermediate-level interconnects, and is expected to decrease by 35%. However, the substitution of low-k dielectrics in the next few years could alter this depending on thermal conductivity of the candidate dielectric relative to oxide. Electromigration is of greatest concern in the global and intermediate-level interconnects because of their long lengths and the drive for linewidth scaling as a means to slow the increase in the number of metal levels. Use of mechanically-weaker low-k dielectric materials will futher increase the influence of electromigration in interconnect design. Finally, the removal of the Si_3N_4 diffusion barrier will drastically increase concerns about interconnect reliability as the conditions favoring interconnect immortality are hindered.

Chapter 8

Summary and Future Research

8.1. Summary

As interconnect linewidths have been scaled downward, the effects of electromigration-induced damage on interconnect reliability has increasingly been of concern . A detailed knowledge of stress evolution and void growth processes enables a determination of the electromigration-induced failure times of these interconnects. This thesis provides this knowledge through the development of an electromigration simulation MIT/EmSim and through experiments.

Stress evolution is simulated by tracking atomic fluxes based on the electron wind force and back fluxes due to concentration gradients, and has been developed into a grain-structure sensitive simulation tool, MIT/EmSim. The stress effect on diffusivity was represented by an exponential expression relating stress effects on lattice density and vacancy concentration. For alloy modeling, the effects of chemical concentration gradients on driving force and the kinetic effects associated with Cu trapping of vacancies was modeled. In Al-Cu alloys, it was demonstrated in this thesis that Cu atoms significantly retard the atomic transport associated with electromigration through Cu trapping, limiting stress evolution to occur after the Cu has been swept away by the electron wind. Al-Cu alloy simulations also predicted the counter-intuitive formation of precipitates in the center of the length of an interconnect, which was observed experimentally by IBM.

The void nucleation and growth process was simulated using a mass conservation and mass transport model. Because of time constraints, electromigration tests are conducted at accelerated conditions by increasing either of the current density or temperature. Failure times from accelerated tests are commonly scaled to service conditions using Black's equation, which relates the time-to-failure to the current density to the exponent -n. The value for this current density exponent n was not well understood, with experimental evidence providing a wide range of values between 1 and 4. MIT/EmSim was used to study current density scaling in the context of void nucleation and growth. In long lines in which failure occurred by nucleation and growth to a specific void volume (or resistance failure), failures of interconnects tested at high current densities were void-growth-limited, and exhibited n=1 scaling. At lower current densities, failure was void-nucleation-limited, leading to scaling with an n=2 exponent. A transition in the current density exponent was observed at intermediate current densities, suggesting that Black's equation is not valid for scaling failure times when there is a change in the failure mechanism, and thereby, the current density exponent.

In short lines, stress evolution and void behavior are more complex. In sufficiently short lines, the back-stress that develops at the anode could interact with the tensile stress at the cathode, slowing further increases in the stress until a steady-state stress profile is reached, and the interconnect is immortal without void nucleation. Alternatively, in lines of relatively longer length, the back-stress will be delayed in its interaction with the tensile stress, and void nucleation will occur. For interconnects in which the back-stress hampers void growth until it ceases, void growth saturation occurs. If the resistance at void growth saturation is less than the failure resistance, the interconnect will never fail and is immortal due to void saturation. Conversely, if the void grows until the resistance increases beyond the resistance failure criterion, the line will fail either by a void-nucleation-limited failure or a void-growth-limited failure. This complex reliability behavior can be catalogued in the form of failure mechanism maps.

Failure mechanism maps illustrate the failure mode or immortality mode of an interconnect as a function of current density and line length. Failure maps can be constructed by using simulations such as MIT/EmSim. For pure interconnects with simple geometries, failure maps can also be constructed by means of analytical approximations. In general, these maps show that short interconnects tested at low current densities result in immortality while long interconnects tested at high current densities exhibit void-growth-limited failures with $n=1$ scaling. Simulations demonstrate that Black's equation is accurate over a relatively short range of test conditions. While several failure maps were constructed for Al-based interconnects, failure maps for pure Cu interconnects required the development of an electromigration model for Cu.

To aid in the development of a Cu EM model, Cu/Ta interconnect structures were fabricated in Microsystems Technology Laboratories at MIT. These structures consisted of a patterned Cu/Ta bilayer interconnect encapsulated in Si_3N_4. Artificial diffusion barriers were created by the removal of strips of Cu at the end of each interconnect. Electromigration tests were conducted using these structures to explore void phenomenology in Cu, and to measure key Cu electromigration parameters. A process of void nucleation and growth until saturation was observed for Cu/Ta interconnects, and found to be consistent with that of Al-based interconnects. The (jL) product tests indicated a critical stress for void nucleation of 270 MPa. Resistance saturation tests demonstrated an effective valence z^* of 4 to 7. Drift rates were shown to be a factor of five faster than that reported in the literature. Void growth tests conducted between 200 and 300°C demonstrated an activation energy for the effective diffusivity of 0.72 eV and coefficient of 1.5×10^{-6} m^2/sec. Attempts directed at determining the $Dz^*\delta$ individually for the Ta and Si_3N_4 interface were inconclusive, and will require some redesign in the next generation of the test structure. These experiments provide an understanding of void nucleation and growth phenomenology in Cu interconnects along with experimental measurement of key Cu electromigration parameters necessary to serve as a foundation

for a Cu-based electromigration model. While values of the Cu electromigration parameters may be expected to vary from microfab to microfab because Cu interfaces are very sensitive to variations in processing, the Cu parameters measured in this thesis are sufficient to develop a fundamental understanding of Cu interconnect reliability. In the future, industrial microfabs could substitute Cu electromigration parameters measured for interconnects fabricated in their own microfab to yield accurate predictions of Cu interconnect lifetimes.

Based on the Cu/Ta experiments, MIT/EmSim was adapted to model electromigration in pure Cu interconnects. Void nucleation and growth was simulated, and current density scaling consistent with Al-based technologies observed. Because of differing values of transition current densities, increased lifetimes of Cu at accelerated conditions may not necessarily be apparent at service conditions. These extrapolations must be complemented by additional simulation or analytic methodologies. Simulations were also conducted to construct a failure mechanism map for Cu damascene interconnects. Compared to an Al map at similar test conditions, the Cu map exhibited a reduced region of immortality and an enlarged region for void-growth-limited failure. Failure maps can be constructed to assess the reliabilty of Cu interconnects, evaluate the accuracy of extrapolating accelerated test data to service conditions using Black's equation, aid in the design of future accelerated tests, and to predict the reliability of future Cu interconnect metal and shunt technologies. Furthermore, short line effects are instrumental in extending the reliability of Cu interconnects, and suggested future designs of Cu interconnects eliminating the diffusion barrier at both studs will have an adverse effect on interconnect reliability. Besides the loss of interconnect immortality, predictions of Cu interconnect lifetimes will become more uncertain as accelerated electromigration test data are extrapolated to low current densities. Furthermore, evaluation of interconnect reliability will become significantly more complex, requiring the subdivison of multiple metal levels into numerous multi-level tree units for individual reliability analysis.

The influence of RC delay, Joule heating, and electromigration on future interconnect design were evaluated and compared based on projections defined in the SIA Technology Roadmap. Over the past few years, RC delay has limited interconnect linewidth scaling of local interconnects, and this will continue to be the case in the future. Joule heating will be of greatest concern in the intermediate-level interconnects, and is expected to decrease by 35%. However, the substitution of low-k dielectrics in the next few years could alter this depending on thermal conductivity of the candidate dielectric relative to oxide. Electromigration is of greatest concern in the global and intermediate-level interconnects because of their long lengths and the drive for linewidth scaling as a means to slow the increase in the number of metal levels. Use of mechanically-weaker low-k dielectric materials will futher increase the influence of electromigration in interconnect design. Finally, the removal of the Si_3N_4 diffusion barrier will drastically increase concerns about interconnect reliability as the conditions favoring interconnect immortality are hindered.

The test structure developed in this thesis provides a simple means of testing the effects of new shunt and barrier layer technologies on the reliability of Cu-based interconnects. Furthermore, this test structure can also be used to evaluate the effects of alternative inter-layer dielectrics on the interface diffusivities for varying Cu-ILD interfaces, and ultimately Cu interconnect reliability. Through the use of the simulation, an accurate methodology for predicting the reliability of Al- and Cu-based interconnects in semiconductor chips has been developed. MIT/EmSim is now being used by Motorola and LSI Logic for evaluating interconnect reliability during the design of future Cu interconnects, and has also been used by numerous SRC-companies and universities through EmSim-Web for electromigration research.

8.2. Future Research

Although the simulations and experiments provide an accurate methodology for evaluating interconnect reliability of Al-based and Cu interconnect technologies, further improvements will enable the study of future Cu technologies and further refine the accuracy of the models and measurements.

Several second order effects are not included in the current MIT/EmSim model, and their inclusion would further increase the accuracy of the simulation. For instance, while current crowding effects are not significant in a 1D simulation tool, use of MIT/EmSim in 2D mode would require the incorporation of current crowding. This would entail the simulation of the effects of current crowding at interconnect tree junctions and at width transitions on the local electron wind force. As another example, a relatively primitive model for partially-spanning voids has been developed for MIT/EmSim. Further refinement of this model and simulation using a 2D version of MIT/EmSim would provide a more accurate determination of the temporary effects of partially-spanning voids, if any, on interconnect reliability. In particular, a characterization of the conditions that favor the transition of a partially-spanning void to a reliability-relevant, fully-spanning void would be of interest.

Because of the tremendous drive for decreasing interconnect linewidths and increasing interconnect performance, new interconnect and dielectric technologies are being developed and introduced at a rapid rate. Modification of MIT/EmSim to simulate these advanced technologies would be beneficial to maintaining this tool at the cutting-edge. For instance, current Cu technologies suggest the domination of a single interface for atomic transport. The possibility of changes in diffusion barrier or shunt material will lead to changes in the dominant interface, or possibly, even to several simultaneous dominant interfaces, and will require the incorporation of a summation of multiple diffusivity paths in place of the current effective diffusivity model. If modifications to

remove the Si_3N_4 diffusion barrier from the current damascene structure are implemented, it would be useful and relatively simple to extend MIT/EmSim to study the reliability of fundamental tree units spanning multiple levels (see Chap. 7). New advances in low-k dielectric materials necessary for RC delay reduction are expected tolead to the replacement of silicon oxide. Some of the candidate dielectric materials include fluorinated oxides, polyimides, aerogels, and patterned air pockets in oxide. These alternative dielectrics are generally of lower mechanical strength than current silicon oxide, and will lead to increased concerns about creep and fracture of the material surrounding an interconnect. Mechanically softer dielectrics will invalidate the current rigid box condition inherent to the MIT/EmSim model, necessitating a replacement of the hydrostatic stress model by a new formulation for a component stress model. Softer dielectrics will also require further study into the stress evolution between adjacent interconnects when the interconnect spacing is on the order of the linewidth. Soft dielectrics will also introduce the possibility of multiple failure modes associated with shunt failure, diffusion barrier failure, intra-layer dielectric failure, and inter-layer dielectric [Hau 00]. Finally, the use of Cu alloying elements such as Sn and Cr has been investigated experimentally [Hu 95], and could be incorporated into MIT/EmSim with minimal modification.

While the Cu/Ta interconnect structures designed and fabricated were sufficient to meet the objectives of the experimental components of this thesis, a number of design suggestions are recommended for the second generation of test structures.

Several suggestions are offered for the redesign of the current mask set. To prevent probe tip damage to the interconnects during inadvertent tip motions during contacting, pads should be placed at least 100 μms away from the interconnect test structures. Pads should also be larger than 100 μm at an edge, and coated with Al to prevent the oxidation of Cu at higher temperatures. The addition of several super-wide lines with linewidths greater than 25 μms will enable more accurate determinations of

individual Dz*δ products. Resistance saturation and other length effects could be better measured with the addition of more line structures between 50 and 200 microns long. Finally, to facilitate the identification of structures tested without a upper nitride layer, it is recommended that test structure labeling also occur on the first level mask.

Modifications in the process flow can also be made to measure the electromigration effects in new diffusion barrier and shunt technologies. While the current fabrication sequence used sputtered Cu, the fabrication of interconnects with electroplated Cu or even a sputtered Cu alloy would provide an interesting comparison. The effects of alternative shunts materials such as TaN or TaON on the void nucleation behavior and shunt diffusivity would provide useful inputs to MIT/EmSim. The feasibility of this experiment will depend on the existence and quality of a wet etch for the shunt. Variations in the design of the process flow and interconnect and shunt structure could permit the use of a dry etch, which would enable the fabrication of finer linewidths beneficial for determining the individual Dz*δ products of each interface. The fabricaton of a Cu interconnect completely surrounded by Ta, requiring the design of one additional mask with interconnect linewidths slightly wider than the Cu interconnects, will allow a direct measurement of the diffusivity at the Cu-Ta interface. As MTL fabrication equipment is further advanced through equipment replacement and additions, the preparation of improved test structures will be possible. Once the Cu CMP process is fully developed and characterized, the fabrication of Cu interconnects through a damascene process will be possible. When new equipment for depositing silicon nitride films of significantly higher quality than current PECVD reactor is installed (rumored to be around Jun 01), the effects a more mechanically rigid dielectric can be studied. Furthermore, as additional deposition materials for inter-layer dielectrics at MTL become available, it will be possible to measure the effects of alternative ILDs on the Cu interface diffusivity and Cu interconnect reliability. The study of more advanced low-k dielectrics such as polyimide on interconnect failure could also be studied as the equipment becomes available. It would also be interesting to investigate the effects of thermal cycling on any

of: the Cu and shunt interface for new shunt materials, Cu and diffusion barrier for films other than nitride, and on any weakening effects on the Si_3N_4 dielectric or low-k dielectric. Given its relatively high dielectric constant, the use of Si_3N_4 as a diffusion barrier for damascene Cu in the future appears uncertain given the importance of reducing RC delay. The test structure used in this experiment offers the opportunity of surveying other alternatives for the diffusion barrier as candidate barrier materials become evident.

The experimental suggestions listed above will allow the measurement and determination of input parameters critical to MIT/EmSim for new technologies associated with the metal, shunt, diffusion barrier, and dielectric. These experimental results, coupled with the simulation modifications listed above, will provide even more accurate predictions for current and future Cu-based interconnects. The use of these more advanced versions of MIT/EmSim in conjuction with the interconnect layout tool MAJIC and grain growth simulator GGSim provides a metal level reliability design tool for interconnect engineers.

Appendix A

MIT/EmSim Manual and Source Code

This Appendix contains a description of running and modifying the source code of MIT/EmSim. Further compilations of EmSim-Web and the Elf Extension are also contained. Finally, the verification of MIT/EmSim using PROMIS and Matlab are also described.

A.1. MIT/EmSim User Manual

MIT/EmSim Introduction

EmSim, or more accurately known as MIT/EmSim, is a finite-element-based electromigration simulation tool useful for accurately predicting the lifetime of pure and alloyed interconnects of varying geometry, thermal history, etc. as a function of line length, current density, and test temperature. A version of EmSim has been installed on the web, and is known as EmSim-Web. MIT/EmSim has also been coupled with a graphical interconnect layout tool, MAJIC and interconnect network analysis tool, ERNI, and a grain growth simulator tool, GGSIM. EmSim was also integrated into Motorola's reliability tool, ELF. EmSim is extremely versatile, both in feature-rich qualities and its interactions with other simulation tools and the World Wide Web. This, however, can lead to some growing pains and panic for new users becaue of the wide variety of features/run-modes thrown at a new user all at once. Let me also note that no GUI is available specifically for EmSim, since MAJIC is expected to provide that function.

MIT/EmSim Features

EmSim currently supports the following:

- Aluminum, binary Al-Cu lines, or pure Copper lines
- Realistic microstructures (when coupled with EmSimGen or GGSim)
- Varying geometries, such as width transitions
- Complex interconnect trees
- Partial- and Full-spanning void nucleation & growth (except at junctions in trees)
- Precipitates (only the data structure, no science here yet)

MIT/EmSim Model

MIT/EmSim uses a modified-Korhonen model. Please see the publication by Park [Par 99] for all the details of the model. Some highlights of the modifications beyond the Korhonen model include:

- The stress effect on diffusivity alters the diffusivity based on the local stress-state, specifically by changing the local vacancy concentration with relation to the stress and by a smaller effect, namely altering the lattice density with the stress.
- For alloys, chemical potentials are defined for Al-Cu metal based on the work of Murray. From these chemical potentials, the divergence in the chemical potentials is calculated, and used to determine atom fluxes in the presence of alloys.
- In alloys, the diffusivity is reduced by Cu-trapping effects based on the model of R. Rosenberg. In this model, a fraction of the vacancies are rendered immobile due to binding with Cu atoms present at the grain boundary. This fraction is calculated based on a mass balance argument. Korhonen's step function diffusivity is also available.
- Various tree structures may also be tested using EmSim. EmSim handles these calculations very similar to the way in which standard stud-to-stud structures are calculated. For tree structures, additional fluxes are calculated at the junctions for each adjacent strand based on the model presented here.

- Two types of void models are included: full-spanning and partial-spanning. The full-spanning void model, in which voids span the full width of the interconnect, is the most matured void model. The partiall-spanning void model was only recently introduced. In both cases, voids may either nucleate at a user-specified critical tensile stress or they be assumed to be pre-existing at a user-specified location. Multiple voids per line are allowed, and void-void interactions generally don't occur due to the physics.
- In the full-spanning void model, once a void nucleates, its initial void size is calculated by relaxing all the stress in the vicinity (relaxation distance) of the void. Void walls are then moved based on fluxes to and away from the void. A net flux to the void results in void shrinkage while a net flux away from the void results in void growth. If the flux of atoms is in the same direction on both sides of the void, void migration results. Void growth continues until a user-defined failure is reached. Failure may either be specified by open circuit (tensile failure), dielectric failure (compressive stress failure), or resistance failure. In the case of resistance failure, the user may define failure based on either a critical void length or a critical resistance percentage increase.

MIT/EmSim Implementation

EmSim employs a finite element model, dividing a test structure into a series of cells and calculating the fluxes of atoms between cells based on the divergence in total potentials (chemical potential of Al and Cu, stress gradient, and electromigration driving current) and the diffusivity between the two cells associated with the microstructure at that point of the interconnect. A detailed summary of the data structures is also given.

Configuring & Compiling MIT/EmSim

This help page assumes you are already knowledgeable in basic unix commands and working with source code (in C). If you aren't, go visit a book on pc/linux/unix machines and learn!

To configure EmSim prior to starting a run, first switch to the source code directory, and then type "make clean". This removes the old object files, output data files, and gets EmSim ready for a new run. Next, configure EmSim for your run by editing the file named switches.h. This file allows the user to select which features of the EmSim model to use, along with other model and formatting options. Select/disable the options you desire to use. You may view an in-depth page describing the switches in detail to help in configuring EmSim for your runs. The Park publication [Par 99] gives the fine details of the EmSim model. It is recommended you leave the switches "FUNC_COMPLETE" and "PARAMS" enabled. All of the switches after that are primarily debugging and unfinished features so you should leave those disabled. Once you have set/un-set all the switches appropriately, save the switches.h file and close it.

Now you can compile the emsim code with your configuration by typing "make emsim". The included Makefile will complete all necessary compiling and linking operations for you. Note that EmSim has no directory-specific code in it -- that is, EmSim can be easily shifted from directory to directory without any modification or need for recompiling. This allows for easier file and version management of EmSim for both new and experienced users. One word of caution: Using a "make clean" command with the Makefile provided with EmSim will clear all EmSim data files from the current working directory. Always be sure to remove any desired output files from the current directory prior to cleaning and compiling in the future. EmSim was written platform-neutral ANSI C, avoiding even the slightest shred of platform-specific code. This greatly benefits the user in that EmSim compiles on every platform attempted without warnings or errors, including: linux, DEC Unix, HPUX, Solaris, Ultrix, and Windows. Other platforms have not been attempted, but are not expected to offer any problems assuming an ANSI C compiler is used. For new users, it is recommended you use Gnu C Compiler (gcc).

MIT/EmSim Input

EmSim requires 3 input files and they must be in the same directory as the executable. These three files detail the simulation conditions, interconnect geometry, and interconnect microstructure of the simulated EM test. The fourth input parameter is the name of the output file containing the time-to-failure data. The first input file is the simulation test conditions (default name of emsim.inp), which stores simulation input parameters. The second input file (default name of geometry) gives interconnect geometry information, strand-specific information, and time_to_print information
The third input file (default name of diffdata), stores the microstructure-specific information indicating bamboo regions and poly-granular cluster regions
The fourth input parameter (default name of ttf.dat), defines the name of the output file to write the failure information to.

A sample copy of each input file, along with very detailed comments about the input parameters, is contained in the attached diskette. The files are each named using the default names specified above.

Testing of tree structures is significantly more complex, and has been summarized (coupled with an example) by S. Hau-Riege. [Hau 00]

Running MIT/EmSim

In the command line itself, EmSim requires four parameters (in addition to the executable, argv[0]). The first three inputs correspond to input filenames while the last corresponds to the name of the primary output file.

Sample input:
emsim120o emsim.inp geometry diffdata ttf.dat
argv[0].........argv[1]....argv[2]......argv[3]......argv[4]

- the first input parameter is the simulation test conditions (default name of emsim.inp), which stores simulation input parameters.
- the second input parameter (default name of geometry) gives interconnect geometry information, strand-specific information, and time_to_print information
- the third input parameter (default name of diffdata), stores the microstructure-specific information indicating bamboo regions and poly-granular cluster regions
- the fourth input parameter (default name of ttf.dat), defines the name of the output file to write the failure information to.

To retain all of the default input names, you can simply type the name of the binary or "make run":

Sample input:
emsim120s
argv[0]

EmSim will automatically use the default names specified above. For single runs, this procedure saves time while for larger batches of runs, it is preferable to enter in the full command line. Before running EmSim, you must first write the contents of the input files as detailed in the links above. Samples of the input files are provided in the descriptions and links above. It is suggested that you use these samples as a template for forming your own input files.

Since most simulation runs will take anywhere from minutes to hours to possibly days, it is suggested you run them in background. Use the "nohup" command to do this, and pipe the output to a file named "output.dat" .

Once a simulation run is started, you will see a "PROGRAM BEGIN" message followed by a series of `function complete' messages conveying the progress of EmSim

in setting up and configuring the data structures corresponding to the user-defined settings in the input files. Finally, a "STARTING CALCUATION" message will signify that EmSim has begun calculating fluxes between cells for each time step to determine stress evolution. EmSim will periodically flash the number of time steps traversed (and corresponding simulation time) on screen (or to the output file if you used nohup). Eventually, assuming EmSim reaches the user-defined failure criterion, EmSim will display failure information, and a closing "PROGRAM COMPLETE" message. If you like, take a look at the example output file. If you want, you can see the fine intricate details of the actual steps involved in the simulation (as recorded in the output file).

Be wary of simply walking away from the simulation run once begun. It is best to periodically observe the progress of the EmSim run, not only to verify that EmSim is simulation what you intended, but also to determine when an "unreacheable" failure condition has been realized (such as setting a failure stress above the steady-state failure stress with the jL product detector disabled). Use the output files described in the next section to observe the progress of the simulation.

MIT/EmSim Output

MIT/EmSim writes a number of simulation output and debugging output files in each run. All of the output files are written in the same directory as the executable, and always end with a .dat extension. With the exception of the first output file listed below, ttf.dat (see notes above), all of the output files use the standard names listed below. The contents of the files are described in the text and links below:

- ttf.dat, whose name is set by the user, contains the simulation time at which the line failed, or a "-1" flag if the line survived
- maxstress.dat determines the maximum tensile stress and the maximum compressive stress along the line at user-defined times and writes them along with their locations to file

- xprofile.dat contains the initial cell data structure information such as cell location, size, current density, and # atoms in each cell
- params.dat gives a quick summary of the input conditions used for the defined line
- cellsize.dat gives dynamic information regarding the cell data structure similar to xprofile.dat; however the cellsize.dat is periodically updated during simulation to reflect changes, additions, and deletions of cells during the run
- (userfilename)_(time).dat uses a user-defined name given in geometry to store data regarding atom concentrations and stress profiles along the length of the line
- void(void#)_(void location) gives void-specific data including void position and size, along with a resistivity depending on the passivation
- erni_info.dat gives the stress output from EmSim at several user-specified times for use by by the metal layout analysis tool, ERNI.
- output.dat is traditionally used for piping output to the screen to this file

Modifying MIT/EmSim

In revising EmSim over the last few years, I have been using a combination of numerical and alphabetic increments to track each new version of the simulation. Minor modifications receive an alphabetic increment while major modifications (new void model, new precipitate model, addition of Cu, etc.) receive a numerical upgrade (i.e., from 1.19 to 1.20). All of the revisions to each version are detailed in a file named filedescript. I avoided the use of RCS and similar programs because of the difficulty in going back to and transferring old versions of the simulation to PCs (where I did quite a bit of development work).

For details on the structure of the code and coding styles (EmSim mimics an object-oriented style, although it isn't truly object-oriented), you can take a look at the coder's page to see the general layout of the EmSim code. Note that the coder's manual only broadly covers EmSim since the source code is extremely well documented line-by-line.

Y2K Compliance

EmSim does not use any year 2000 information in any calculations and is therefore Y2K-compliant. After the passing into the new millenium, no known Y2K bugs of any kind have surfaced. Similarly, no Y2035 bugs associated with 32-bit Unix system date rollovers are anticipated for the same reasons.

Credits

- The EmSim model incorporating the stress effect on diffusivity and the treatment of Al-Cu alloys was developed by Y.J. Park.
- Coding of this model and the development of the void models was performed by V.K. Andleigh.
- Addition of models determining the value of thermal stresses was performed by V.K. Andleigh.
- The addition of the tree model was completed jointly by S.P. Hau-Riege and V.K. Andleigh. The help notes associated with trees was written by S.P. Hau-Riege.
- In the web version of EmSim, the CGI framework was developed by M. Verminski with further modifications completed by V.K. Andleigh.
- W. Fayad authored EmSimGen, which is used to provide realistic microstructures to EmSim and EmSim-Web.
- W. Fayad performed a major rewrite of PolySeg to enable realistic microstructure of user-specified tree structures as well.
- Y. Chery created ERNI, which is used to provide tree structures from mask layouts to MIT/EmSim, as well as to illustrate the results.
- This help page was authored by V.K. Andleigh.

A.2. MIT/EmSim Source Code

MIT/EmSim Code Overview

EmSim is written in C, and is composed of about 250 pages of code (15,000 lines of code!) contained in about 30 files. Only two short files are included in the print version of this appendix. The main file is emsim.c, and all the main data structures and useful definitions are stored in struct_emsim.h. All of the "science" switches in EmSim are stored in switches.h (such as pure copper, trees, voids, etc.)

Most of EmSim uses doubly-linked lists as the primary data structure. Specifically, every interconnect is broken up into one/series of strands (doubly-linked list) which in turn are divided up into a series of cells (also a doubly-linked list). Voids are stored in the form of a doubly-linked list. The order in the lists are such that the previous and next nodes are the previous and next neighbors of the simulated structure.

MIT/EmSim Data Structures

A tree is divided into a series of strands connected at junctions. Strands allow for boundary conditions at each end of the strand -- this offers EmSim tremendous flexibility in the type of structures it can test, some of which may not be realistic in real interconnects. Junctions, which allow for boundary conditions on all four sides and also enables 2D and 3D modeling if the user so desires. Each strand in turn is divided into a series of cell data structures. Partial-spanning voids are defined "around" a specific cell, while full-spanning voids are delineated simply by stating full-spanning void boundary conditions. Precipitates are defined within a specific cell.

- strands - strands are the fundamental unit of the simplest tree structures. They are composed of a head (beginning) and a tail (end). Boundary conditions, which could links to other junctions, are defined at the head and tail. Pointers at the head

and tail refer to the cells located at each end. The strands data structure is a doubly-linked list. Strands also contain current (Amps) information.

- junctions - junctions join the various strands together to form the tree structure. Junctions have four boundary conditions (N, E, S, W), which may be strands (for standard tree structure) or other junctions (for more complex trees, 2D simulations). Junctions are defined as a doubly-linked list. Junctions also contain geometry and # of atoms information.
- cells - cells are the fundamental unit into which interconnects are broken down. The cells form the finite elements in the simulation, and fluxes are tracked between cells (and junctions) over each time step to determine the stress evolution over time. Each set of cells within a strand are a doubly-linked-list. Cells contain geometry information and # of atom information, along with flux calculation parameters such as current density and diffusivity.
- voids - voids are defined between/around cells depending on whether they are full-spanning/partial-spanning respectively. Voids are tracked primarily based on geometry information (the outer bounds of the void). A switch in the voids data structure indicates whether the void is partial- or fully-spanning. The data structure for a void is a doubly-linked-list.

MIT/EmSim File Structure

MIT/EmSim contains three main files, numerous supporting C files, and numerous header files associated with them.

<u>Main</u>

The following three files dominate the general heirarchical design of EmSim. The functions included in these files are the program main(), data structure specification, and scientific and debugging optional switches for customizing EmSim's behavior to your desire. These files include

- emsim.c - declares and initializes all the linked lists, initializes calculation inputs, calls all the major functions, contains the main repeating while loop of the simulation, and performs closing functions.
- struct_emsim.h - declares all of the data structures used in EmSim, as well as the #define formulas and the like. Also contains key on abbreviations used in the the EmSim code.
- switches.h - contains all of the scientific and debugging switches that are in EmSim. Composed of #define switches that allow features such as voids, trees, etc., as well as debugging commands such as cellsize.dat output files, etc. Also contains switch to enable EmSim-Web version of code only.

Supporting C Files

These file descriptions are in no particular order. Note that some files do have interdependencies on others, as evidenced by the file imports at the beginning of each file. These include:

- biemsim_calc.c - contains calculation of fluxes for alloys only.
- emsim_files.c - contains code allowing the writing of ALL of of EmSim's output files.
- emsim_resistance.c - given a void size and line/shunt geometry information, calculates the resistance of the line using several different models.
- emsim_graphics.c - writes output for Java graphics output of EmSim. Not fully developed yet.
- emsim_save.c - saves all of the EmSim data values to a file. May be used in future for continuing a stopped run.
- emsim_calc.c - contains calculation of fluxes for pure elements (Al, or Cu) only.
- emsim_initialize.c - initializes the input values of test conditions, diffusivities, shunt structure, etc.

- emsim_small_voids.c - defines and runs the small (partial-spanning) void model including void nucleation, void growth, and void conversion to full-spanning void
- emsim_define_cells.c - defines the cell linked-list for cells within a strand, including geometry and microstructure information, along with current density, etc. Almost every component of the cell data structure is defined here.
- emsim_node_fcns.c - allows object-oriented editing of linked-lists such as inserting or deleting cell/node, splitting or combining nodes, as well as updating nodes.
- emsim_trees.c - contains all the code for defining tree structure into EmSim, and then performing the calculation of fluxes for both pure and alloyed materials.
- emsim_end.c - determines whether/how line failed, and reports results to user on-screen and in output files.
- emsim_numerics.c - experimental testbed for evaluating faster integration schemes in EmSim. No numerical calculating scheme could be adapted to MIT/EmSim to provide an equivalently accurate, stable, and fast calculation as the forward Euler.
- emsim_voids.c - defines full-spanning void model, allowing for user-defined specification of voids, nucleation, and growth of voids. Also prints void geometry results.
- emsim_error.c - composes error messages from EmSim to the user either on-screen or in a separate file as specified by the user.
- emsim_ppts.c - defines precipitate location, nucleation, and growth of precipitates. Although the code for handling the precipitate data structure is there, no science has been implemented yet for precipitates.

Support Header Files

These files simply include the function prototypes of the files listed above. No other functions are performed in these files. These header files include:
biemsim_calc.h, emsim_files.h, emsim_resistance.h, emsim.h, emsim_graphics.h, emsim_save.h, emsim_calc.h, emsim_initialize.h, emsim_small_voids.h,

emsim_define_cells.h, emsim_node_fcns.h, emsim_trees.h, emsim_end.h, emsim_numerics.h, emsim_voids.h, emsim_error.h, and emsim_ppts.h

Run Synopsys

When a run is started, EmSim goes through the following steps: definining the input variables, building the data structures, main simulation loop, and data dump and cleanup. The listing of emsim.c, which performs the operations listed in this Run Synopsys, is included at the end of this sub-section.

Defining Input Variables

The first steps in an EmSim run are to read in basic constants like filenames, mathematical and general scientific symbols, and variables associated with the thermodynamic and kinetic parameters in electromigration. These steps are:

1. EmSim first displays a "BEGIN PROGRAM" greeting and prints a listing of the SWITCHES options enabled/disabled.
2. EmSim then assigns input/output filenames based on the text entered in the command line.
3. Builds the strand linked-list, and allocates the appropriate memory
4. Reads in the simulation conditions contained in emsim.inp and enters the values into the appropriate variables
5. Defines constants relating to mechanical properties of the metal and shunt, chemical potentials, diffusivities, and vacancy parameters.
6. Calculates the thermal stress based on the thermal stress parameters and the processing temperature and test temperature differential
7. Defines constants associated with precipitation and Cu segregation at the grain boundary

Building Data Structures

The main data structures to build in EmSim are strands (and within them cells), trees, precipitates, and voids. After building the data structures, the failure condition is defined. The steps for this are:

1. Using the information contained in the geometry file, information regarding the strand location (what are its neighbors? -- stud, junction, etc.) along with the current(I) flowing through the strand, definining a sequential series (linked-list) of cells for each strand with the position information (coordinate along x or y axis), assigning other dimensional information to the cells as a function of the coordinate position along the interconnect, initializing cells with an appropriate number of atoms consistent with the thermal stress and cell geometry, and setting flags regarding when to print the stress information in the strand to file. This is done for all the strands present.
2. Diffusivity factors are assigned to the cells based on the microstructure present as a function of coordinate position along the interconnect as defined in the diffdata file.
3. Based on the current flowing in a strand (defined earlier), the current density present in each cell is calculated for each cell.
4. Quick jL product test to see if interconnect is certain to be immortal based on the (short) length of an interconnect.
5. Trees are next defined based on the information contained in the geometry file. The junction data structure is defined and appropriate memory allocated. Then the boundary conditions (N, E, S, W) of each junction is determined and the appropriate links made. Finally, based on the dimensions given in the geometry file, the dimensional information is written to the junction structure and the appropriate number of atoms determined based on the thermal stress and junction dimensions.
6. Precipitate data structure is then created and filled in. This part of the code is still a work in progress, and likely to be changed in the future.

7. Next the voids data structure is created, and pre-existing voids are then nucleated at locations specified in the geometry file.
8. An initial value for the resistance is calculated (if a void is present, the resistance will be higher than that without void nucleation).
9. The failure condition is determined from emsim.inp, and the corresponding flags in EmSim set to stop simulating once this condition is reached.

<u>Main Simulation Loop</u>

The next series of functions are the heart and soul of EmSim. This series of functions will be repeated anywhere from thousands to billions of times depending on the simulation conditions set by the users. The steps include:

1. 1.Prior to starting the repeating loop, a "STARTING CALCULATION" greeting is displayed to inform the user that EmSim has begun calculations.
2. First, fluxes between each cell are calculated, followed by fluxes in between cells and junctions. Then the atoms are transferred, and the new atom concentrations and stresses are calculated. To maximize the efficiency of the calculation, these steps are separated for pure Al, pure Cu, and alloys. ("if" loops cost a lot of computational power)
3. After the concentrations have been updated, an update on precipitates is conducted: first to grow/shrink existing precipitates, and then to see if new precipitates have nucleated.
4. Next time index and total time are updated, followed by a function to adjust the time step based on the change in atom concentration (not too fast, not too slow).
5. Then voids are updated: first the walls of existing voids are adjusted left or right (or down for small voids) based on atoms flows, and then a check to see if new voids nucleate is conducted. Finally, the resistance of the interconnect is updated based on the new void sizes.

6. At user-defined intervals, the data obtained during the simulation (stresses, atom concentrations, etc.) are written to the various output files.
7. The values for the stress and atom concentrations from the current step are copied over to that of the previous time step (to prepare for the next time step). Note the purpose of storing the old values of stress and concentration is that it allows us to backtrack a time step if the atom concentration changes are too rapid, allowing the possibility of excessive error being introduced into the calculation.
8. Now the failure condition is compared to current simulation parameters, and a "get_out" variable flagged if failure has been reached.
9. We then go back to step 2 of this section and keep repeating until failure is reached.

Data Dump and Cleanup

In this final section, the simulation conditions are output to all the output files, simulation conclusions messages, and memory cleanup exercises are completed. These steps are:

1. Output data is written to all the output files, including max/min stresses, x_profile, resistance, voids, celldata, ttf, save_all_data, and various plot data files. Files, including params.dat, are closed cleanly.
2. Function write_ttf determines the failure condition that was reached, and prints associated failure data as requested by the user.
3. Further file clean-up and memory clean-up is conducted, and simulation complete greeting is displayed. A value of zero is returned by the simulation when the end is successfully reached.

Programming Notes

- When variables are passed from one function to another, a "_" is appended to the beginning of the variable name to distinguish it as a local variable. For the main data structures detailed above, this specification is maintained.

- When adding new sections of code, try to retain the abbreviations as maintained in struct_emsim.h to make the code easier to use. Also, try to use the object-oriented functions given in emsim_node_fcns.c to make the code more stable and easier to modify.
- Do not use any machine-specific code in EmSim -- EmSim compliles on all platforms without errors or warning. Please help keep it that way!
- Do not add any directory-specific code into EmSim. Given the wide array of platforms along with wide variety of software interactions (ELF, ERNI, EmSim-Web, etc.) and difficulties in file and version management of EmSim, directory-specific information is not permitted in EmSim.
- When adding new features, be sure they work with the rest of EmSim. Otherwise, it becomes difficult to track what new features work with different parts of the code. If your addition to the EmSim source code breaks other features in EmSim, I won't incorporate it. For example, if a feature you add doesn't work with trees, I won't insert it into the EmSim code base.
- Keep your code additions heavily commented. Not only is this useful for you when you look at your code a year-and-a-half later, it makes it possible for other users to use/modify your code. I refused to incorporate poorly commented code additions.
- Test your code additions before you incorporate them into MIT/EmSim. It's taken years to get EmSim has stable as it is right now. If you make an addition to the code, you're responsible for fixing the bugs in it.

Raw Source Code

Because the inclusion of the entire MIT/EmSim source code would make this thesis unreasonably long (the source code is 250 pages even at a ten-point font), only two files are listed here. The first is the Makefile, which instructs the compiler what files and options are to be used during compilation. In addition, the Makefile also offers the standard run commands for MIT/EmSim to the user. After the Makefile, the source to emsim.c, which serves as the program Main, is listed. MIT/EmSim is protected by a

GNU Public License, and is the property of the original copyright holders. The source code is reprinted here with the permission of the original copyright holders, and no ownership rights of any kind are transferred to MIT in the course of its inclusion in this thesis.

Here is the Makefile for MIT/EmSim.

```
########################################################################
# Makefile for EMSIM
#
# Vaibhav Andleigh
# 1-Dec-00
#
# (c) copyright 1996-2001   GPL License
#                           All rights reserved.
########################################################################

BIN   = emsim120r

CC    = gcc
FLAGS = -O3
INCL  = -I/.
LIBS  = -lm

# for optimized runs, use gcc with flag -O3 only
# for debugging, use cc with flags -g -Wall which eliminates
optimization

OBJS  = emsim.o \
        biemsim_calc.o \
        emsim_calc.o \
        emsim_files.o \
        emsim_initialize.o \
        emsim_define_cells.o \
        emsim_node_fcns.o \
        emsim_resistance.o \
        emsim_voids.o \
        emsim_small_voids.o \
        emsim_ppts.o \
        emsim_trees.o \
        emsim_end.o \
        emsim_graphics.o \
        emsim_save.o \
        emsim_error.o

OBJS_C      = emsim.o \
        biemsim_calc.o \
        emsim_calc.o \
        emsim_files.o \
        emsim_initialize.o \
```

```
        emsim_define_cells.o \
        emsim_node_fcns.o \
        emsim_resistance.o \
        emsim_voids.o \
        emsim_small_voids.o \
        emsim_ppts.o \
        emsim_trees.o \
        emsim_end.o \
        emsim_graphics.o \
        emsim_save.o \
        emsim_error.o

OUTPUT      = params.dat \
        maxstress.dat \
        xprofile.dat \
        fluxes.dat \
        ttf.dat \
        output.dat \
        all_sim_info.dat \
        cellinfo.dat \
        junction.dat \
        gbplot.dat \
        error.dat

#######################################################################

.c.o: ; $(CC) -c $(FLAGS) $(INCL) $*.c

#######################################################################

go:   clean rm_dat emsim go_bg

goc:  rm_dat emsim go_fg

run:    rm_dat go_fg

runi: rm_dat go_fgi

runc:   clean rm_dat emsim go_fg

debug:      clean rm_dat emsim
      gdb $(BIN)

gdb:  rm_dat
      gdb $(BIN)

emsim:      $(OBJS)
      $(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

go_bg:      rm_dat
      nohup $(BIN) emsim.inp geometry diffdata ttf.dat > output.dat&

go_fg:      rm_dat
      $(BIN) emsim.inp geometry diffdata ttf.dat
```

```
go_fgi:     rm_dat
      $(BIN) emsim.inp geometry diffdata ttf.dat 3e5


clean:      rm_dat
      rm -f core *~ $(BIN) $(OUTPUT) $(OBJS_C)

rm_dat:
      rm -f *.dat
```

The following is a listing of emsim.c, the primary execution file for MIT/EmSim.

```
/***********************************************************************
 *                                                                     *
 *   EmSim 1.20  Elemental & Alloy Electromigration with Trees,        *
 *               Void Nucleation & Growth, and Precipitates(some)      *
 *               with Save All Data Feature, Graphics Output,          *
 *               Small Voids and Error File                            *
 *                                                                     *
 *    Vaibhav Andleigh                                                 *
 *    Young Joon Park                                                  *
 *    Stefan Riege                                                     *
 *    Prof. Carl Thompson                                              *
 *                                                                     *
 *    1-Dec-00                                                         *
 *                                                                     *
 *    (c) copyright 1996-2001,  GPL License                            *
 *                              All rights reserved.                   *
 *                                                                     *
***********************************************************************

NOTE ON VARIABLE NAMES: variables in functions differ from those in
main only in that the ones in functions have an '_' in front of them.
See structemsim.h for common abreviations used throughout the code.

Help file is located at:  http://nirvana.mit.edu/~vab/emsimhelp/

Input Summary:
  emsim120 emsim.inp geometry diffdata ttf.dat
or
  emsim1.20
for default input files (listed above)

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

/*** libraries ***/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*** switches and data structures ***/
```

```
#include "switches.h"           /* turns on WEB, TREES, VOIDS,
STRESS...  */
#include "structemsim.h"        /* structemsim.h includes data
structures */

/*** function prototypes ***/
#include "emsim_initialize.h"
#include "emsim_define_cells.h"
#include "emsim_node_fcns.h"
#include "emsim_files.h"
#include "emsim_calc.h"
#include "biemsim_calc.h"
#include "emsim_resistance.h"
#include "emsim_trees.h"
#include "emsim_small_voids.h"
#include "emsim_voids.h"
#include "emsim_ppts.h"
#include "emsim_end.h"
#include "emsim_graphics.h"
#include "emsim_save.h"
#include "emsim_error.h"

/*** global output FILES ***/
FILE *fparams;                  /* stores user values for the run   */

/*** global variables ***/
int num_cells=1;                /* stores highest id# +1 of cells   */

/***==<< EMSIM VARIABLE DECLARATIONS>>=============================***/

int main(int argc, char *argv[])
{
/*** SIMULATION parameters ***/
  double run_time,             /* how long to run simulation in sec  */
         delta_t,              /* time step between calculations     */
         dt_old,               /* previous time step value           */
         total_time = 0.0;     /* simulation run time (in seconds)   */
  char *input_fn,              /* simulation input file name string  */
       *diffusivity_fn,        /* diffusivity data filename string   */
       *cellsize_fn,           /* xstep, width, thick, ttp file name */
       *output_fn,             /* user-entered ttf output file name  */
       *jbatch_txt;            /* j entered at command line/override */

  double Q_vac_fm,             /* heat of formation for a vacancy    */
         Q_bind_alcu,          /* binding energy of Al, Cu atoms     */
         cu_segr_gb,           /* copper GB segregation factor       */
         vac_diff,             /* vacancy diffusivity incl. neighbor */
         coord_num;            /* coordination # for GB atoms        */

  int t_step_index = 0,        /* time step index (# not value)      */
      t_step_size = 1;         /* do we really need this?            */
#ifndef WEB
  int print_maxstress = 0;     /* if changes, print maxstresses vs.t */
#endif
```

```
  /*** thermal mismatch parameters ***/
  int therm_stress_model;      /* which model to calc thermal stress */
  double pass_temp,            /* passivation temperature in Kelvin  */
         test_temp,            /* test temperature in Kelvin         */
         init_stress;          /* initial stress in line due to CTE  */

  /*** define Al/Cu DATA STRUCTURES here ***/
  Alcutype q;                  /* q = effective charge z*      */
  Alcutype elec_coeff;         /* elec_coeff = q*e*rho         */
  Chem_2coeff_type chem_coeff; /* chemical potential of Al,Cu  */
  At_frac_type at_frac;        /* atomic fraction initial, max */
  Difftype diff;               /* diffusivity of pure Al at GB */

  /*** define STRAND VARIABLES here ***/
  Strandnode *strand,          /* strand data structure pointer*/
             *strandptr;       /* pointer to strand reference  */

  /*** define TREE VARIABLES here ***/
  Junction *junction=NULL;     /* junction structure pointer   */

  /*** some FAILURE VARIABLES here ***/
  int get_out = 0,             /* boolean to see if stress exceed(1) */
      fail_mode = 0;           /* user-defined failure mode for run  */
  double maxstress=0.0,        /* maximum tensile stress along line  */
         minstress=0.0,        /* minimum compressive stress in line */
         max_clus_len=0.0,     /* maximum cluster length in the line */
         ten_sigma_crit,       /* tensile critical stress            */
         cmp_sigma_crit,       /* compressive critical stress        */
         crit_resist=0.0;      /* critical resistance for failure    */

  /*** define VOID VARIABLES here ***/
  int void_model;              /* which model to nucl/gro voids*/
  double max_void_len=0.0;     /* size of largest void         */
  double crit_void_len=0.0;    /* void length for failure      */
  double void_zs_len=7E-7;     /* stress-free zone about a void*/
  Voidnode *voids=NULL;        /* voids data struct. ptr       */

  /*** define SHUNT LAYER VARIABLES here ***/
  double rho_shunt,            /* resistivity of shunt layer   */
         shunt_thick,          /* thickness of shunt layer     */
         init_resist,          /* initial resistance of line   */
         resistance=0.0;       /* resistance of entire line    */

  /*** define PRECIPITATE VARIABLES here ***/
  double cu_solubility,        /* copper solubility in Al */
         ppt_density,          /* density of atoms in ppt */
         crit_num_cu_at;       /* critical # of Cu atoms  */
#ifndef WEB
  Pptnode *ppt=NULL;           /* precipitate struct. ptr */
#endif

  /*** CONSTANTS declarations ***/
  double rho,                  /* rho = resistivity of metal*/
```

```
        b,                          /* b = bulk modulus of metal */
        poisson,                    /* Poisson's ratio for metal */
        omega;                      /* omega = atomic volume met.*/

  double kT,                        /* kT = thermal energy       */
        kappa;                      /* kappa = B*omega / k*T     */

#define k 1.38E-23                  /* k = Boltzmann's constant  */
#define e 1.6E-19                   /* e = charge of electron    */
#define AVOG 6.02E23                /* AVOG = Avogadero's number */

/***====== EMSIM PROGRAM MAIN BEGINS
======================================***/

  /*** initialization greeting ***/
#ifndef WEB
  printf("\n\nPROGRAM BEGIN... \n\n");

  /*** print switches listing to screen ***/
  print_switches();

  /*** set-up default command line arguments, or use user-input values
***/
  if ( (argc < 5) && (argc != 1) )
    printf("\n\n  You fucked up the command line, dumbass! \n\n");
  fflush(NULL);
#endif
  if ( argc == 1 ) {
    input_fn = "emsim.inp";
    cellsize_fn = "geometry";
    diffusivity_fn = "diffdata";
    output_fn = "ttf.dat";
  }
  else {
    input_fn = argv[1];
    cellsize_fn = argv[2];
    diffusivity_fn = argv[3];
    output_fn = argv[4];
  }
#ifdef JBATCH
  if ( argc == 6 )
    jbatch_txt = argv[5];
  else
    printf("\n\n Insert the current density into the command
line\n\n");
  fflush(NULL);
#endif

  /*** initialize strand pointer, formed linked list if necessary ***/
  strand = malloc(sizeof(Strandnode));
  define_strand_list(cellsize_fn, strand);

  /*** Enter input parameters ***/
```

```
  user_inputs(input_fn, &delta_t, &run_time, &test_temp, &pass_temp,
&therm_stress_model, &fail_mode, strand, &ten_sigma_crit,
&cmp_sigma_crit, &at_frac.init.cu, &void_model);

  /*** calculate constants for diffusion equation  ***/
  kT = k*test_temp;
  Q_vac_fm = 0.75 * e;          /* heat of formation for vacancy      */
  Q_bind_alcu = 0.25 * e;       /* binding energy of Al, Cu atoms     */
  coord_num = 12;               /* coordination # for GB atoms        */
  vac_diff = coord_num * exp ( Q_bind_alcu / kT );  /* vacancy
diffusivity*/

  /*** define metal properties ***/
  define_metal_props(cellsize_fn, at_frac.init.cu, &rho, &b, &poisson,
&omega, &q, &elec_coeff);

  /*** define shunt layer properties ***/
  define_shunt_props(cellsize_fn, &rho_shunt, &shunt_thick);

  /*** define thermodynamic quantities (chemical potential) ***/
  define_chem_potl(&chem_coeff, test_temp, kT);

   /*** define Al and Cu diffusivities ***/
  define_diffusivity(&diff, kT);

  /*** calculate initial thermal stress ***/
  if( therm_stress_model == PLASTIC_COMPL )
    read_plastic_stress(&init_stress, cellsize_fn, test_temp,
pass_temp);
  else
    define_init_stress(&init_stress, at_frac.init.cu,
therm_stress_model, test_temp, pass_temp, b, poisson);

  /*** define precipitate constants ***/
  cu_solubility = test_temp * exp(0) / test_temp * 0.01 * 1E25;
  ppt_density = 1 / omega;
  crit_num_cu_at = PI * square(1E-9) * ppt_density;     /* 1nm crit
nucl.rad */

  /*** define copper segregation constants ***/
  cu_segr_gb = 1.0;             /* Cu GB segregation factor      */
  at_frac.max.cu = at_frac.init.cu * cu_segr_gb;
  at_frac.init.al = 1.0 - at_frac.init.cu;  /* maximum Al atomic
fraction    */
  at_frac.max.al = 1.0;         /* maximum Al content is pure Al */

  /*** read in strand info (start_x,length,current), all strand if
trees ***/
  if( strand->next == NULL )
    define_first_strand(cellsize_fn, strand, jbatch_txt);
  else
    define_strand_info(cellsize_fn, strand);

  /*** open data output files and write file headers ***/
```

```
  kappa = b*omega/(k*test_temp);
  initialize_files(delta_t, run_time, strand, pass_temp, test_temp, q,
b, omega, elec_coeff, at_frac, kappa, ten_sigma_crit, cmp_sigma_crit,
init_stress);

  /*** loop through all the strands ***/
  for( strandptr = strand; strandptr != NULL; strandptr = strandptr-
>next ) {

    /*** read in strand location and orientation ***/
    set_strand_location(cellsize_fn, strandptr);

    /*** read in xstep resolution and define cells ***/
    set_cell_resolution(cellsize_fn, strandptr);

    /*** read in dimensions and set initial values of cells ***/
    enter_dimensions(cellsize_fn, strandptr, b, omega);

    initialize_conc(strandptr, at_frac, init_stress, omega, kT);

    /*** read in first time_to_print, increment ttp counter ***/
    setup_time_to_print(strandptr, cellsize_fn);
  }

  /*** read in diffusivity file data ***/
  enter_diff_ratio(diffusivity_fn, strand, diff, b, omega,
&max_clus_len);

  /*** calculate current density, write initial cell data structure
***/
  calc_current_density(cellsize_fn, strand);

#ifdef CELLDATA
  /*** celldata output ***/
  print_celldata(strand, total_time);
#endif

#ifdef jL_TEST
  /*** see if line never fails by jL product (assumes no trees) ***/
  if( strand->next == NULL )
    see_if_line_fails(strand, max_clus_len, ten_sigma_crit,
cmp_sigma_crit, &get_out, init_stress, rho, q, omega);
#endif

#ifdef TREES
  /*** setup tree structure if necessary ***/
  if( strand->next != NULL ) {
    /*** initialize junction pointer ***/
    junction = malloc(sizeof(Junction));

    /*** setup junction linked list ***/
    define_junctions(cellsize_fn, diffusivity_fn, junction, strand);

    /*** define boundary conditions for the strands ***/
```

```
    initialize_tree_bc(cellsize_fn, junction, strand);

    /*** determine #Al,Cu atoms, stress for junction cell ***/
    initialize_junctions(junction, strand, at_frac, b, omega,
init_stress, kT);
  }
#endif

#ifdef PRECIPITATES
  /*** read in user-defined precipitates ***/
  ppt = malloc(sizeof(Pptnode));
  define_ppts(cellsize_fn, ppt, ppt_density, strand, crit_num_cu_at);
#endif

  /*** initialize voids, read in user-defined *full-size* voids &
nucleate ***/
  voids = malloc(sizeof(Voidnode));
  voids->id = NONE;
  voids->next = NULL;
#ifdef VOIDS
  define_voids(cellsize_fn,voids,strand,ten_sigma_crit,omega, b,
void_model);
  if( fail_mode != TENSILE_FAIL )
    nucl_which_void(voids, strand, ten_sigma_crit, void_zs_len, omega,
b, void_model);
#endif

  /*** calc initial resistance for line w/ or w/o presence of voids
***/
  init_resist = calc_resistance(voids, strand, shunt_thick, rho_shunt,
rho);

  /*** read in user-defined failure mode ***/
  define_failure(input_fn,fail_mode, void_model,
&crit_resist,init_resist, &crit_void_len);

#ifdef CELLDATA
  /*** celldata output ***/
  print_celldata(strand, total_time);
#endif

  /*** announce START OF CALCULATION ***/
#ifndef WEB
  printf("\n\nSTARTING CALCULATION ..... \n\n");
  fflush(NULL);
#endif

  /*** GIANT WHILE LOOP that performs time iterations ***/
  while (total_time <= run_time) {

    /*** increment total time and step index ***/
    t_step_index += t_step_size;

    /*** perform DIFFUSION CALCULATION ***/
```

```
    if ( at_frac.init.cu == 0.0 ) {
      /*** calculate fluxes for each branch for Al or Al/Cu case ***/
      calc_al_flux(strand, diff, elec_coeff, Q_vac_fm, kappa, omega,
kT, b);

#ifdef TREES
      /*** calculate fluxes at junctions ***/
      calc_junction(junction, diff, chem_coeff, cu_segr_gb, delta_t,
vac_diff, elec_coeff, Q_vac_fm, kappa, omega, kT, b, &maxstress,
&minstress);
#endif

      /*** calc. new Al,Cu concentrations at cell/int, find max/min
stress ***/
      calc_al_conc(strand, diff, elec_coeff, Q_vac_fm,  delta_t, kappa,
omega, kT, b, &maxstress, &minstress);
    }

#ifdef PURE_CU


    else
      /*** calculate pure copper diffusion ***/
      if( at_frac.init.cu == 1.0 ) {
      calc_cu_flux(strand, diff, elec_coeff, Q_vac_fm, kappa, omega,
kT, b);
      /*** calc new Al,Cu concentrations at cell/int, find max/min
stress **/
      calc_cu_conc(strand, diff, elec_coeff, Q_vac_fm,  delta_t, kappa,
omega, kT, b, &maxstress, &minstress);
       }
#endif

    else {
      /*** calculate alloy diffusion ***/
      calc_bi_flux(strand, diff, chem_coeff, elec_coeff, Q_vac_fm,
vac_diff, cu_segr_gb, kappa, omega, kT, b);
#ifdef TREES
      calc_junction(junction, diff, chem_coeff, cu_segr_gb, delta_t,
vac_diff, elec_coeff, Q_vac_fm, kappa, omega, kT, b, &maxstress,
&minstress);
#endif
      calc_bi_conc(strand, diff, chem_coeff, elec_coeff, Q_vac_fm,
vac_diff, cu_segr_gb, delta_t, kappa, omega, kT, b, &maxstress,
&minstress);
    }

#ifdef PRECIPITATES
    /*** see if nucleate precipitate, also determine growth of ppt ***/
    nucleate_ppt(ppt, cu_solubility, strand);
    if( ppt->next != NULL)
      alter_ppt_size(ppt, cu_solubility, strand);
#endif
```

```
    /*** update total time ***/
    total_time += delta_t;
    dt_old = delta_t;

    /*** see if we should increase delta_t ***/
    alter_time_step_size(junction, strand, &total_time, &delta_t);

#ifdef VOIDS
    /*** calculate growth of VOIDS ***/
    if( voids->id != NONE )
      grow_which_void(voids,strand,dt_old,void_zs_len,omega,b,
&max_void_len);
#endif

    /*** see if a void nucleates ***/
#ifdef ONE_VOID
    if( voids->id == NONE )
      if( fail_mode != TENSILE_FAIL )
      nucl_which_void(voids, strand, ten_sigma_crit, void_zs_len,
omega, b, void_model);
#endif
#ifdef MULT_VOIDS
    /*** see if additional voids nucleate ***/
    if( fail_mode != TENSILE_FAIL )
      nucl_which_void(voids, strand, ten_sigma_crit, void_zs_len,
omega, b, void_model);
#endif

    /*** update resistance value with voids ***/
    if( t_step_index % PRINT_TO_FILE == 0 )
      resistance = calc_resistance(voids,strand,shunt_thick, rho_shunt,
rho);

#ifndef WEB
    /*** write profile data to file, is the 2nd condition needed???
***/
    if ((t_step_index % PRINT_TO_FILE == 0) || (strand-
>time_to_print[strand->ttp_counter] <= total_time))
      write_to_files(junction, voids, ppt, strand, delta_t, total_time,
t_step_index, &print_maxstress, resistance);
#endif

    /*** copy concentrations and stresses over, find maximum stresses
***/
    copy_for_next_time_step(junction, strand, &maxstress, &minstress);

    /*** determine FAILURE MODE ***/
    /*** check for compressive failure ***/
    get_out = ( (minstress < cmp_sigma_crit) ? (COMP_FAILURE) :
(get_out) );

    /*** check for tensile failure if no void formation ***/
    if( fail_mode == TENSILE_FAIL )
```

```
      get_out = ( (maxstress > ten_sigma_crit) ? (TENS_FAILURE) :
(get_out) );
#ifdef VOIDS
    /*** check for resistance failure if appropriate ***/
    if( fail_mode == RESISTANCE_FAIL )
      get_out = ( (resistance >= crit_resist) ? (RES_FAILURE) :
(get_out) );

    /*** check for void failure if appropriate ***/
    if( fail_mode == VOID_LEN_FAIL )
      get_out = ( (max_void_len >= crit_void_len) ? (RES_FAILURE) :
(get_out));
#endif

    /*** if critical stress exceeded...get out of giant WHILE loop ***/
    if ( get_out != 0 )
      break;

  }    /*** end of GIANT WHILE loop ***/

  /*** write data before EXITING PROGRAM ***/
#ifndef WEB
  print_maxminstress(junction, strand, delta_t, total_time);
  print_x_profile(strand, total_time);
#endif
  /*** write final resistance data ***/
  resistance = calc_resistance(voids,strand,shunt_thick, rho_shunt,
rho);

  /*** write final void size data ***/
#ifdef VOIDS
  print_void_size(voids, strand, total_time, resistance);
#endif

#ifdef CELLDATA
  /*** celldata output ***/
  print_celldata(strand, total_time);
#endif

  /*** print simulation completion message, write ttf to fp_out ***/
  write_ttf(output_fn, get_out, total_time, fail_mode, crit_void_len,
crit_resist, maxstress, minstress);

#ifdef SAVE_DATA
  /*** save all the simulation data output to file to reload later ***/
  save_simulation_info(delta_t, test_temp, pass_temp, run_time,
ten_sigma_crit, cmp_sigma_crit, junction, strand, voids, ppt);
#endif

#ifdef PLOT_GRAPHICS
  /*** write plot output data ***/
  write_plot_file(voids, ppt, strand, total_time);
#endif
```

```
#ifdef PARAMS
  /*** close output files ***/
  fclose(fparams);
#endif

#ifndef WEB
  /*** write simulation termination message ***/
  printf("\n\n ========================================");
  printf("\n\n PROGRAM COMPLETE... \n\n");
#endif

  /*** exit program successfully ***/
  return(0);

}      /*** end of main ***/

/***===============================================================***/
```

A.3. EmSim-Web Description

To ease technology transfer to the semiconductor industry, MIT/EmSim was installed on web server in our lab (see Fig. A-1). Accounts are created for SRC and university users using standard CGI scripts (adduser command followed by password). Currently, the EmSim-Web is mounted on nirvana.mit.edu and is located in the /usr/local/www/httpd/cgi-bin/emsim/ directory. The emsim.pl perl script in EmSim-Web takes the responses from a user regarding interconnect structure microstructure, thermal history, and test conditions using a standard CGI web form. After performing user authentication routines and units handling, emsim.pl forwards all of the necessary userid and EM parameters to another perl script, emsim_background.pl. The emsim_background.pl script writes all the necessary input files for EmSimGen and MIT/EmSim and runs each program sequentially. It then sits in a wait state until the run is completed, after which it accepts the failure time from MIT/EmSim and generates a custom lognormal plot of failure times embedded in a custom web page detailing user inputs. The custom web page and plot are stored in the ~/Results directory indefinitely until the admin chooses to delete them. The emsim_background script is also responsible for sending the user and admin email when runs are started, and sending an email to the user when the results are available.

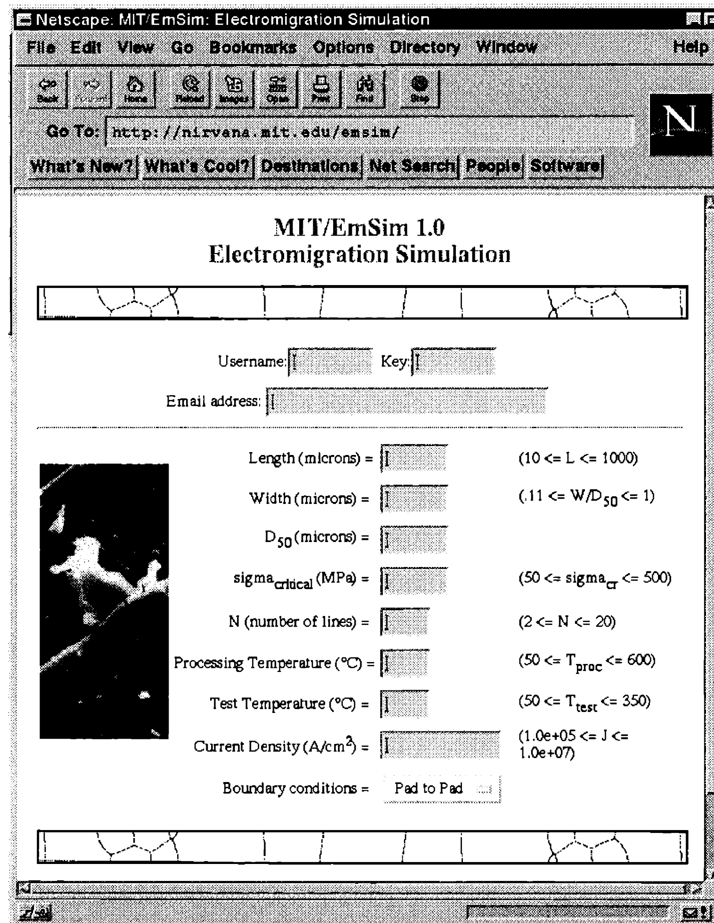


Figure A-1. View of EmSim-Web data entry page.

Modifying the web page is relatively simple, and instructions for this are provided at the top of each perl script. It is noted that neither EmSimGen nor MIT/EmSim is allowed to print any text to the standard output or the perl scripts will crash (a weakness of Perl, not of the script). The source code for emsim.pl and emsim_background.pl is approximately 25 pages long, and is located at http://nirvana.mit.edu/emsim/.

A.4. MIT/EmSim-ELM Extension

As part of a collaboration with Motorola Semiconductor Products Sector in Mesa, AZ, a graphical user interface was developed for MIT/EmSim so that it may be integrated into Motorola's reliability CAD tools. This code was developed in Tcl/Tk.

```
#!/usr/local/bin/tclsh7.6
package require Tk
#package require Tclx
#//////////////////////////////////////////////////////////////////////
#!/usr/lanl/bin/wish -f
#//////////////////////////////////////////////////////////////////////
#package require Tk

tk appname EmSim
option add *Button.disabledForeground     gray
option add *Menubutton.disabledForeground gray
option add *Menu.disabledForeground       gray
#
set auto_path [linsert $auto_path 0 .]

#//////////////////////////////////////////////////////////////////////
#
# Fonts.
#
#//////////////////////////////////////////////////////////////////////
set helvetica "-linotype-helvetica-medium-r-normal--*-120-*"
set times     "-adobe-times-medium-r-normal--*-180-*"

#//////////////////////////////////////////////////////////////////////
#
# Colors
#
#//////////////////////////////////////////////////////////////////////
set workareacolor  LightSkyBlue1
set buttoncolor    LightSkyBlue1
set workareawidth  800
set workareaheight 800

#//////////////////////////////////////////////////////////////////////
#
# Root window.
#
#//////////////////////////////////////////////////////////////////////
wm minsize   . $workareawidth $workareaheight
#wm aspect   . 1  1  1  1
#wm title    . "MIT/EmSim Electromigration Simulator"
wm iconname . "Vab"
wm geometry . +0+0

#//////////////////////////////////////////////////////////////////////
#
```

```
# Global variables.
#
#//////////////////////////////////////////////////////////////////////

global workareacolor

#//////////////////////////////////////////////
#
# EmSim Layout
#
#//////////////////////////////////////////////


#///////////////////////////////////////////////////////
# Global variables.
#///////////////////////////////////////////////////////
global pt Pt
global temp Temp
global current Current j J
global cuFrac CuFrac
global length Length
global d50 D50
global width Width
global n N
global lbc Lbc
global rbc Rbc
global FailureCriterion critStress CritStress
global critres CritRes
global Msg
global plottf
global ScrollInfoField InfoFieldLineNumber
global DesignRuleField Th th Wdr wdr Im im Tk tk

set Msg "Enter input parameters...."
set InfoFieldLineNumber 0

#///////////////////////////////////////////////////////
# Preliminaries
#///////////////////////////////////////////////////////
#destroy .logo
#.menu.mod   configure -state disabled
#.menu.analy configure -state disabled
#.menu.tools configure -state disabled
#.menu.lib   configure -state disabled
#.menu.option.a entryconfigure 1 -state normal
#.menu.option.a.list entryconfigure 1 -state normal
#.menu.option.a.list entryconfigure 2 -state normal
#.menu.option.a.list entryconfigure 3 -state normal
#.menu.option.a.list entryconfigure 4 -state normal
#.menu.option.a.list invoke 1

#///////////////////////////////////////////////////////
# Frame for the model.
#///////////////////////////////////////////////////////
```

```
frame .w -background $workareacolor
pack  .w -side top -fill both -expand yes

#//////////////////////////////////////////////////////
# Title
#//////////////////////////////////////////////////////
label  .w.title  \
  -text "MIT/EmSim 1D Model" \
  -fg Maroon -bg OldLace
pack  .w.title -side top -fill x

#//////////////////////////////////////////////////////
# MIT/EmSim logo
#//////////////////////////////////////////////////////
canvas .w.emSimLogo -bg OldLace -height 32m
.w.emSimLogo create bitmap 55m 13m -bitmap @bitmaps/emsim5bw.xbm \
 -foreground Maroon
pack .w.emSimLogo -side top -fill x

#//////////////////////////////////////////////////////
# Menu button bar.
#//////////////////////////////////////////////////////
frame .w.menu -relief raised -borderwidth 1 -background $workareacolor
pack  .w.menu -side top -fill x

button .w.menu.quit \
  -relief raised -text "Quit" -foreground Maroon \
  -activebackground AntiqueWhite \
  -bg $buttoncolor \
  -command  {destroy .}
pack .w.menu.quit -side left -padx 1m -pady 1m

menubutton .w.menu.option          \
  -menu .w.menu.option.a           \
  -text "Options"  -underline 0   -relief raised              \
  -foreground Maroon -activebackground AntiqueWhite \
  -background $buttoncolor

menu .w.menu.option.a

  .w.menu.option.a  add cascade    \
  -label "Failure Time Definition" \
  -menu .w.menu.option.a.list

menu .w.menu.option.a.list

  .w.menu.option.a.list add radiobutton    \
  -label "First Void Nucleation" \
  -variable FailureCriterion -value FirstVoidFail \
  -command FirstVoidFail

  .w.menu.option.a.list add radiobutton    \
  -label "Open" \
  -variable FailureCriterion -value Open \
```

```
  -command OpenFail

  .w.menu.option.a.list add radiobutton    \
  -label "Resistance Change" \
  -variable FailureCriterion -value ResistanceChange \
  -command ResistanceFail

  .w.menu.option.a.list add radiobutton    \
  -label "Critical Stress" \
  -variable FailureCriterion -value CriticalStress \
  -command CriticalStressFail

  .w.menu.option.a.list add radiobutton    \
  -label "Critical Void Length" \
  -variable FailureCriterion -value VoidLengthFail \
  -command VoidLengthFail
pack .w.menu.option -side left -padx 1m -pady 1m


button .w.menu.run \
  -relief raised -text "Run"  -foreground Maroon \
  -activebackground AntiqueWhite \
  -bg $buttoncolor
pack .w.menu.run -side left -padx 1m -pady 1m

button .w.menu.ref \
  -relief raised -text "References" -foreground Maroon \
  -activebackground AntiqueWhite \
  -bg $buttoncolor
pack .w.menu.ref -side left -padx 1m -pady 1m

button .w.menu.help \
  -relief raised -text "Help"  -foreground Maroon \
  -activebackground AntiqueWhite \
  -bg $buttoncolor
pack .w.menu.help -side left -padx 1m -pady 1m


#////////////////////////////////////////////////////
# Map out Entry and Message Fields.
#////////////////////////////////////////////////////
frame .w.entry -bg $workareacolor -relief flat
frame .w.msg   -bg $workareacolor -relief flat
label .w.msg.l -bg $workareacolor -textvariable $Msg -fg black -relief
sunken -width 35 -anchor w

#frame .w.listbox -bg $workareacolor -relief flat
frame .w.radiobutton -bg $workareacolor -relief flat

#////////////////////////////////////////////////////
# Define scrollable information field
#////////////////////////////////////////////////////
frame .w.info -relief flat -bg $workareacolor
set ScrollInfoField .w.info.sif
```

```
frame $ScrollInfoField -relief flat -bg $workareacolor
set c $ScrollInfoField.c
canvas $c \
  -bg AntiqueWhite \
  -scrollregion { 0 0 500 1000 } \
  -xscrollcommand "$ScrollInfoField.hscroll set" \
  -yscrollcommand "$ScrollInfoField.vscroll set" \
  -relief sunken -width 414 -height 500
scrollbar $ScrollInfoField.vscroll  -relief sunken -command "$c yview"
-bg LightGray
scrollbar $ScrollInfoField.hscroll -orient horiz -relief sunken -
command "$c xview"  -bg LightGray
bind $c <2> "$c scan mark %x %y"
bind $c <B2-Motion> "$c scan dragto %x %y"

pack .w.entry .w.msg .w.info -side top -anchor nw
pack .w.msg.l -side left -anchor nw

#///////////////////////////////////////////////////////
# Design rules field --  currently unused
#///////////////////////////////////////////////////////
set DesignRuleField .w.info.dr
frame $DesignRuleField -relief flat -bg $workareacolor

pack $ScrollInfoField $DesignRuleField -side left -anchor nw -padx 3
pack $ScrollInfoField.vscroll -side right -fill y
pack $ScrollInfoField.hscroll -side bottom -fill x
pack $c -side right -fill y

#///////////////////////////////////////////////////////
# Setup failure criterion region
#///////////////////////////////////////////////////////
frame .w.msg.r -bg $workareacolor -relief groove -bd 2
label .w.msg.r.l -bg $workareacolor -relief flat -text "Failure
Criterion: $FailureCriterion" -anchor w -width 38 -fg Maroon

pack .w.msg.r   -side left -anchor nw -padx 6
pack .w.msg.r.l -side top  -anchor nw
FirstVoidFail

#///////////////////////////////////////////////////////
# Setup 5 columns of frames
#///////////////////////////////////////////////////////
frame .w.entry.l -bg $workareacolor -relief flat
frame .w.entry.m -bg $workareacolor -relief flat
frame .w.entry.p -bg $workareacolor -relief flat
frame .w.entry.q -bg $workareacolor -relief flat
#frame .w.listbox.s -bg $workareacolor -relief flat
frame .w.entry.r -bg $workareacolor -relief groove -bd 2
pack .w.entry.l .w.entry.m .w.entry.p .w.entry.q .w.entry.r  -side left
-fill y -pady 5 -padx 5

frame .w.entry.q.top -bg $workareacolor -relief raised
frame .w.entry.q.bot -bg $workareacolor -relief raised
```

```
pack .w.entry.q.top .w.entry.q.bot -side top -fill y -pady 5 -padx 5

#///////////////////////////////////////////////////////
# Define user-entry textboxes + radiobuttons
#///////////////////////////////////////////////////////

# Cu concentration
label .w.entry.l.mat  -text "Atomic% Cu"     -bg $workareacolor -fg
black \
    -anchor w -width 15
entry .w.entry.m.mat -textvariable cuFrac    -bg LightGray      -fg
black \
    -relief sunken   -width 10
set cuFrac 0.1

# line length
label .w.entry.l.length  -text "Length (um)" -bg $workareacolor -fg
black \
    -anchor w -width 15
entry .w.entry.m.length -textvariable length -bg LightGray      -fg
black \
    -relief sunken   -width 10
set length 200

# median grain size
label .w.entry.l.d50  -text "D50 (um)"       -bg $workareacolor -fg
black \
    -anchor w -width 15
entry .w.entry.m.d50 -textvariable d50       -bg LightGray      -fg
black \
    -relief sunken   -width 10
set d50 0.5

# linewidth
label .w.entry.l.width  -text "Width (um)"
      -bg $workareacolor -fg black -anchor w -width 10
entry .w.entry.m.width -textvariable width   -bg LightGray
      -fg black -relief sunken   -width 10
set width 0.4

# processing temperature (to calc. thermal stress)
label .w.entry.l.pt  -text "Proc Temp (C)"   -bg $workareacolor
      -fg black -anchor w -width 15
entry .w.entry.m.pt -textvariable pt         -bg LightGray
      -fg black -relief sunken   -width 10
set pt 200

# test temperature
label .w.entry.l.temp -text "Test Temp (C)"  -bg $workareacolor
      -fg black -anchor w  -width 15
entry .w.entry.m.temp -textvariable temp -bg LightGray -fg black \
    -relief sunken   -width 10
set temp 200
set Temp $temp
```

```
bind .w.entry.m.temp <Leave> {
   global Temp temp
   set Temp $temp
}
bind .w.entry.m.temp <Return> {
   global Temp temp
   set Temp $temp
}

# current density (later converted to current for MIT/EmSim)
label .w.entry.p.j  -text "j (MA/cm^2)"      -bg $workareacolor -fg
black \
    -anchor w -width 10
entry .w.entry.q.top.j  -textvariable j -bg LightGray -fg black \
    -relief sunken  -width 9
set j 1
set Jvals $j
set Jprint [expr $j * 1e6]
bind .w.entry.q.top.j <Leave> {
   global Jvals j
   set Jvals $j
}
bind .w.entry.q.top.j <Return> {
   global Jvals j
   set Jvals $j
}


# Number of lines to test
label .w.entry.p.n  -text "\# of lines" -bg $workareacolor -fg black \
    -anchor w -width 10
entry .w.entry.q.top.n  -textvariable n -bg LightGray -fg black \
    -relief sunken  -width 9
set n 5
set N $n

# left boundary condition
label .w.entry.p.lbc  -text "Left BC" -bg $workareacolor -fg black \
    -anchor w -width 10
radiobutton .w.entry.q.bot.lpad -text "Pad" -variable lbc -value "Pad"
    -bg LightGray -fg black -relief sunken -width 6
radiobutton .w.entry.q.bot.lstud -text "Stud" -variable lbc
      -value "Stud" -bg LightGray -fg black -relief sunken -width 6
set lbc "Stud"

#right boundary condition
label .w.entry.p.rbc  -text "Right BC" -bg $workareacolor -fg black \
    -anchor w -width 10
radiobutton .w.entry.q.bot.rpad  -text "Pad"  -variable rbc
      -value "Pad" -bg LightGray -fg black -relief sunken -width 6
radiobutton .w.entry.q.bot.rstud -text "Stud" -variable rbc
      -value "Stud" -bg LightGray -fg black -relief sunken   -width 6
set rbc "Stud"
```

```
#////////////////////////////////////////////////////////
# Define plot option checkboxes
#////////////////////////////////////////////////////////
checkbutton .w.entry.r.p1 -text "Plot failure times" \
   -bg $workareacolor -activebackground $workareacolor -fg black
      -relief flat -highlightbackground $workareacolor -anchor w \
      -variable plottf -onvalue 1 -offvalue 0

checkbutton .w.entry.r.p2 -text "Plot t50 vs j" \
   -bg $workareacolor -activebackground $workareacolor -fg black
      -relief flat -highlightbackground $workareacolor -anchor w \
      -variable plott50vsj -onvalue 1 -offvalue 0

checkbutton .w.entry.r.p3 -text "Plot t50 vs 1/T" \
   -bg $workareacolor -activebackground $workareacolor -fg black
      -relief flat -highlightbackground $workareacolor -anchor w \
      -variable plott50vstinv -onvalue 1 -offvalue 0


#////////////////////////////////////////////////////////
# Pack data input boxes and buttons onto frame
#////////////////////////////////////////////////////////
# Material, Geometry & Microstructure
pack .w.entry.l.mat .w.entry.l.length .w.entry.l.width .w.entry.l.d50 \
    -side top -pady 5 -anchor nw -fill x
pack .w.entry.m.mat .w.entry.m.length .w.entry.m.width .w.entry.m.d50 \
    -side top -pady 3 -anchor nw -fill x

# Processing and Test Temperature
pack .w.entry.l.pt -side top -pady 5 -anchor nw -fill x
pack .w.entry.m.pt -side top -pady 3 -anchor nw -fill x

pack .w.entry.l.temp -side top -pady 5 -anchor nw -fill x
pack .w.entry.m.temp -side top -pady 3 -anchor nw -fill x

# Current density
pack .w.entry.p.j    -side top -pady 5 -anchor nw -fill x
pack .w.entry.q.top.j    -side top -pady 3 -anchor nw -fill x

# Number of lines
pack .w.entry.p.n    -side top -pady 10 -anchor nw -fill x
pack .w.entry.q.top.n    -side top -pady 1 -anchor nw -fill x

# Boundary Conditions
pack .w.entry.p.lbc  -side top -pady 5 -anchor nw -fill x
pack .w.entry.q.bot.lstud .w.entry.q.bot.lpad \
     -side top -pady 0 -anchor nw -fill x

pack .w.entry.p.rbc  -side top -pady 30 -anchor nw -fill x
pack .w.entry.q.bot.rpad .w.entry.q.bot.rstud \
     -side bottom -pady 0 -anchor nw -fill x

# Plotting option checkboxes
pack .w.entry.r.p1  .w.entry.r.p2  .w.entry.r.p3  -side top -pady 3 \
```

```
    -anchor nw -fill x

#////////////////////////////////////////////////////////
# Operations after Run button is pressed
#////////////////////////////////////////////////////////
bind .w.menu.run  <ButtonPress> \
{
    global plottf ScrollInfoField InfoFieldLineNumber
    global Msg
    global Temp Jvals
    global J I

    # do display labels on scrollbar
    set line $InfoFieldLineNumber
    if { $line > 0 } {
      DisplayInfo line $ScrollInfoField.c ""
    }


    # run simulations for range of temps and current densities
    foreach T $Temp {
      foreach J $Jvals {
          # convert current density to current
          set I [expr $J * $width / 100]

          # run EmSimGen + MIT/EmSim
          runEmSim pt T length width d50 cuFrac I lbc rbc n tf line

          # organize output
          SaveTf T J n tf

#         set data [T50andSigma $n]
          set t50   [lindex $data 0]
          set sigma [lindex $data 1]
          if { $sigma != "infinity" } {
            set t01   [expr $t50 * exp (-2.326 * $sigma)]
          } else {
            set t01 infinity
          }
          DisplayInfo line $ScrollInfoField.c " "
          DisplayInfo line $ScrollInfoField.c \
            "T      J          t50     Sigma              t01"
          DisplayInfo line $ScrollInfoField.c \
            "$T   $J    $t50   $sigma          $t01"
          set T5($T,$J) $t50
          set Sigma($T,$J) $sigma

          puts "temp=$T j=$J"
      }
    }

    set f [open t01 w]
    puts $f "0 $t01"
    puts $f "1 $t01"
```

```
    close $f

#    if { $plottf        } { PlotTf Temp Jvals t01              }
#    if { $plott50vsj    } { PlotT50vsJ Temp Jvals T5 Sigma     }
#    if { $plott50vstinv } { PlotT50vsTinv Temp Jvals T5 Sigma }

    set InfoFieldLineNumber $line
    puts ""
    puts "Finished MIT/EmSim Model script..."
    puts ""
}

#///////////////////////////////////////////////////////////////////////
# Run EmSimGen and EmSim code
#///////////////////////////////////////////////////////////////////////

proc runEmSim { Pt temp Length Width D50 CuFrac Current Lbc Rbc N TF
Line} {

   upvar $Pt             pt
   upvar $temp           tt
   upvar $CuFrac         cu
   upvar $Length         L
   upvar $Current        I
   upvar $Lbc            lbc
   upvar $Rbc            rbc
   upvar $Width          W
   upvar $N              n
   upvar $TF             tf
   upvar $D50            d50
   upvar $Line           line
   global ScrollInfoField
   global CriticalStress CritRes VoidLen
   global FailureCriterion

   # write emsim.inp file for MIT/EmSim
   set fileEmSimInp [open emsim.inp w 0600]
   puts $fileEmSimInp "PASSIVATION_TEMPERATURE\n$pt\n"
   puts $fileEmSimInp "TEST_TEMPERATURE\n$tt\n"
   puts $fileEmSimInp "THERMAL_STRESS_MODEL\n1\n"
   puts $fileEmSimInp "RUN_TIME\n3e5\n"

   # list failure-dependent text in scrolling box + write to emsim.inp
    DisplayInfo line $ScrollInfoField.c ""
    if { [string compare $FailureCriterion CriticalStress] == 0 } {
      CriticalStressFail
      DisplayInfo line $ScrollInfoField.c "#   tf (hrs)  Failure
Stress(MPa)"
      puts $fileEmSimInp "FAILURE_MODE\n1\n"
    }
    if { [string compare $FailureCriterion FirstVoidFail] == 0 } {
      FirstVoidFail
      DisplayInfo line $ScrollInfoField.c "#   tf (hrs)  Nucl. Stress
(MPa)"
```

```
      puts $fileEmSimInp "FAILURE_MODE\n1\n"
    }
    if { [string compare $FailureCriterion ResistanceChange] == 0 } {
      ResistanceFail
      DisplayInfo line $ScrollInfoField.c "#   tf (hrs)  D R / Ro(%)"
      puts $fileEmSimInp "FAILURE_MODE\n2\n"
      puts $fileEmSimInp "RESISTANCE_FAILURE_PCT\n$CritRes\n"
    }
    if { [string compare $FailureCriterion VoidLengthFail] == 0 } {
      VoidLengthFail
      DisplayInfo line $ScrollInfoField.c "#   tf (hrs)  Void Length
(um)"
      puts $fileEmSimInp "FAILURE_MODE\n3\n"
      puts $fileEmSimInp "VOID_FAILURE_LENGTH\n$VoidLen\n"
    }
   set crs $CriticalStress
   puts $fileEmSimInp "CRIT_TENSILE_STRESS\n$crs\n"
   puts $fileEmSimInp "CRIT_COMPRESSIVE_STRESS\n$crs\n"
   puts $fileEmSimInp "PERCENT_COPPER\n$cu\n"
   puts $fileEmSimInp "LENGTH\n$L\n"
   puts $fileEmSimInp "CURRENT\n$I\n"

   puts $fileEmSimInp "BOUNDARY_CONDITIONS"
   if { [string compare $lbc Stud]==0 && [string compare $rbc Stud]==0
} {
       puts $fileEmSimInp "-1, -1"
   }
   if { [string compare $lbc Stud]==0 && [string compare $rbc Pad]==0 }
{
       puts $fileEmSimInp "-1, -2"
   }
   if { [string compare $lbc Pad]==0 && [string compare $rbc Stud]==0 }
{
       puts $fileEmSimInp "-2, -1"
   }
   if { [string compare $lbc Pad]==0 && [string compare $rbc Pad]==0 }
{
       puts $fileEmSimInp "-2, -2"
   }
   # finished writing to emsim.inp
   close $fileEmSimInp

   # loop through simulation for each line microstructure
   set q 1
   while { $q <= $n } {
       # run EmSimGen and EmSim, increment q
       puts "Running EmSimGen ..."
       catch {exec models/emsim_gen $L $W $d50 emsimgen.out} error
       puts "Running EmSim \#$q ..."
#       catch {exec models/emsim119ae emsim.inp geometry emsimgen.out
ttf.dat} output
       set output [expr $tt + 100 * $q * $I]
       puts $output
       set t $output
```

```
       if { $t > 0 } {
         if { [string compare $FailureCriterion FirstVoidFail] == 0 } {
             DisplayInfo line $ScrollInfoField.c "$q   $t   "
         }
         if { [string compare $FailureCriterion CriticalStress] == 0 }
{
             DisplayInfo line $ScrollInfoField.c "$q   $t   "
         }
         if { [string compare $FailureCriterion ResistanceChange] == 0
} {
             DisplayInfo line $ScrollInfoField.c "$q   $t   "
         }
         if { [string compare $FailureCriterion VoidLengthFail] == 0 }
{
             DisplayInfo line $ScrollInfoField.c "$q   $t   "
         }
         if { [string compare $FailureCriterion OpenFail] == 0 } {
             DisplayInfo line $ScrollInfoField.c "$q   $t   "
         }
         set tf($q) $t
         set Msg "Failure time for line $q: $tf($q) hours"
       } else {
         set Msg "Failure time for line $q is infinity"
       }
       .w.msg.l configure -text $Msg
       update
       incr q
   }
#   set n [expr $q - 1]

   set Msg "Done"; .w.msg.l configure -text $Msg; update idletasks
   after 1000
   set Msg ""; .w.msg.l configure -text $Msg; update idletasks
}

#//////////////////////////////////////////////////////////////////////
# Data Handling and Plot Generation
#//////////////////////////////////////////////////////////////////////

proc SaveTf { Temp Current N TF } {

   upvar $Temp      T
   upvar $Current   J
   upvar $N         n
   upvar $TF        tf

   set list ""
   for {set i 1} {$i <= $n} {incr i} {lappend list $tf($i)}
   set sortedlist [lsort -real -increasing $list]

   set file [open EmSim.out w]
   for {set i 0} {$i < $n} {incr i} {
      set time [lindex $sortedlist $i]
      if { $time > 0 } { puts $file $time }
```

```
   }
   close $file
puts "$T"
   exec cp -f EmSim.out EmSim.$T.$J.out

}

proc PlotTf { Temp Current T01 } {

   upvar $Temp      T
   upvar $Current   J
   upvar $T01       t01

   set Msg "Plotting ..."; .w.msg.l configure -text $Msg; update
idletasks

   set i 0
   set files ""
   exec cp -f lognormal.xmgr.parms.t temp
   foreach temp $T {
      foreach cur $J {
         exec View -data EmSim.$temp.$cur.out -ln > tf.$temp.$cur
         set files [concat $files tf.$temp.$cur]

         exec sed "s/FILM$i/tf-$temp-$cur/g" temp > xmgr.parms
         exec mv -f xmgr.parms temp
         incr i
      }
   }
   exec sed "s/YLABEL/Failure Time (Hours)/g" temp > xmgr.parms

   catch {eval exec xmgr $files -p xmgr.parms -autoscale y} error

   exec rm -f temp xmgr.parms
   set Msg ""; .w.msg.l configure -text $Msg; update idletasks

}

proc PlotT50vsJ { Temp Current T5 Sigma } {

   upvar $Temp      T
   upvar $Current   J
   upvar $T5        t50
   upvar $Sigma     sigma

   set file [open t50vsj w]
   foreach j $J {
      foreach t $T {
        set newj [expr 1e6 * $j]
         if { ! [string match $t50($t,$j) infinity] } {
            puts $file "$newj $t50($t,$j)"
         }
      }
   }
```

```
   close $file

   catch {exec xmgr t50vsj -p t50vsj.parms -autoscale y} error
}

proc PlotT50vsTinv { Temp Current T5 Sigma } {

   upvar $Temp      T
   upvar $Current   J
   upvar $T5        t50
   upvar $Sigma     sigma

   set file [open t50vstinv w]
   foreach j $J {
      foreach t $T {
         if { ! [string match $t50($t,$j) infinity] } {
            set tinv [expr 1.0e3/$t]
            puts $file "$tinv $t50($t,$j)"
         }
      }
   }
   close $file

   catch {exec xmgr t50vstinv -p t50vstinv.parms -autoscale y} error
}


proc MaxTf { N TF Imax Tfmax } {

   upvar $N     n
   upvar $TF    tf
   upvar $Imax  imax
   upvar $Tfmax tfmax

   set imax  0
   set tfmax 0
   for {set i 1} {$i <= $n} {incr i} {
      if { $tf($i) > $tfmax } { set tfmax $tf($i); set imax $i }
   }
}

#proc T50 { } {
#   set lnt50 [exec Moments -data EmSim.out -nm 0 -ln]
#   set t50 [expr exp ($lnt50)]
#   return $t50
#}
#
#proc T50andSigma { n } {
#   if {$n > 1} {
#      exec Moments -data EmSim.out -nm 1 -ln > Moments.out
#      set file [open Moments.out r]
#      gets $file lnt50
#      gets $file m2
#      close $file
```

```
#      set t50 [expr exp ($lnt50)]
#      set sigma [expr sqrt ($m2)]
#      set data [concat $t50 $sigma]
#      return $data
#   } else {
#      set data "infinity infinity"
#      return $data
#   }
#}

proc DisplayInfo { N c msg } {

   upvar $N n

   set dy 15
   set y [expr 5 + $dy * $n]
   $c create text 5 $y -anchor nw -text $msg
   incr n
}


#///////////////////////////////////////////////////////////////////////
# Failure Criteria
#///////////////////////////////////////////////////////////////////////
proc CriticalStressFail {} {
   global FailureCriterion
   global CriticalStress criticalstress
   global workareacolor

    # get rid of previous stress textbox
    if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }

   puts "Failure Criterion: Critical Stress"
   if { [winfo exists .w.msg.r.l] } {
      .w.msg.r.l configure -text "Failure Criterion: $FailureCriterion"
      frame .w.msg.r.lv -relief flat -bg $workareacolor
      label .w.msg.r.lv.l -text "Stress (MPa)" -bg $workareacolor -fg
black
      entry .w.msg.r.lv.v -textvariable criticalstress -bg LightGray -
fg black -relief sunken   -width 20
      pack .w.msg.r.lv -side top -anchor nw
      pack .w.msg.r.lv.l .w.msg.r.lv.v -side left -anchor w
#      bind .w.msg.r.lv.v <Return> {
#         set CriticalStress $criticalstress
#      }
#      bind .w.msg.r.lv.v <Leave> {
#         set CriticalStress $criticalstress
#      }
#      set criticalstress 600
      set CriticalStress $criticalstress
   }
}
```

```
   if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }

proc FirstVoidFail {} {
   global FailureCriterion
   global CriticalStress critstress
   global workareacolor
    # get rid of previous stress textbox
    if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }
   puts "Failure Criterion: First Void Nucleation"
   if { [winfo exists .w.msg.r.l] } {
      .w.msg.r.l configure -text "Failure Criterion: $FailureCriterion"
      frame .w.msg.r.lv -relief flat -bg $workareacolor
      label .w.msg.r.lv.l -text "Stress (MPa)" -bg $workareacolor -fg
black
      set critstress 400
      puts "0.1critstress = $critstress"
      entry .w.msg.r.lv.v -textvariable critstress -bg LightGray \
        -fg black -relief sunken -width 20 -state normal
      puts "1critstress = $critstress"
      bind .w.msg.r.lv.v <Return> {
        global CriticalStress critstress
        set CriticalStress $critstress
      }
      bind .w.msg.r.lv.v <Leave> {
        global CriticalStress critstress
        set CriticalStress $critstress
      }
      puts "2critstress = $critstress"
      set CriticalStress $critstress
      puts "99CriticalStress = $CriticalStress"
#       set critstress $CriticalStress
      pack .w.msg.r.lv -side top -anchor nw
      pack .w.msg.r.lv.l .w.msg.r.lv.v -side left -anchor w
   }
}

proc OpenFail {} {
   global FailureCriterion
   global workareacolor
    # get rid of previous stress textbox
    if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }

   puts "Failure Criterion: Open"
   if { [winfo exists .w.msg.r.l] } {
      .w.msg.r.l configure -text "Failure Criterion: $FailureCriterion"
      frame .w.msg.r.lv -relief flat -bg $workareacolor
```

```
      label .w.msg.r.lv.l -text "Stress (MPa)" -bg $workareacolor -fg
black
      entry .w.msg.r.lv.v -textvariable criticalstress -bg LightGray -
fg black -relief sunken   -width 20
      pack .w.msg.r.lv -side top -anchor nw
      pack .w.msg.r.lv.l .w.msg.r.lv.v -side left -anchor w
      bind .w.msg.r.lv.v <Return> {
         set CriticalStress $criticalstress
      }
      bind .w.msg.r.lv.v <Leave> {
         set CriticalStress $criticalstress
      }
      set criticalstress 500
      set CriticalStress $criticalstress
   }
}

proc VoidLengthFail {} {
    global FailureCriterion
    global VoidLen voidlen
    global workareacolor
    # get rid of previous stress textbox
    if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }
   puts "Failure Criterion: Critical Void Length"
   if { [winfo exists .w.msg.r.l] } {
      .w.msg.r.l configure -text "Failure Criterion: $FailureCriterion"
      frame .w.msg.r.lv -relief flat -bg $workareacolor
      label .w.msg.r.lv.l -text "Length (um)" -bg $workareacolor -fg
black
      entry .w.msg.r.lv.v -textvariable critres -bg LightGray -fg black
            -relief sunken   -width 20
      pack .w.msg.r.lv -side top -anchor nw
      pack .w.msg.r.lv.l .w.msg.r.lv.v -side left -anchor w
      bind .w.msg.r.lv.v <Return> {
         set VoidLen $voidlen
      }
      bind .w.msg.r.lv.v <Leave> {
         set VoidLen $voidlen
      }
      set voidlen 1
      set VoidLen $voidlen
   }
}

proc ResistanceFail {} {
    global FailureCriterion
    global CritRes critres
    global workareacolor
    # get rid of previous stress textbox
    if { [winfo exists .w.msg.r.lv] } {
      destroy .w.msg.r.lv
   }
```

```
   puts "Failure Criterion: Resistance Change"
   if { [winfo exists .w.msg.r.l] } {
      .w.msg.r.l configure -text "Failure Criterion: $FailureCriterion"
      frame .w.msg.r.lv -relief flat -bg $workareacolor
      label .w.msg.r.lv.l -text "D R / Ro (%)" -bg $workareacolor -fg
black
      entry .w.msg.r.lv.v -textvariable critres -bg LightGray -fg black
-relief sunken   -width 20
      pack .w.msg.r.lv -side top -anchor nw
      pack .w.msg.r.lv.l .w.msg.r.lv.v -side left -anchor w
      bind .w.msg.r.lv.v <Return> {
         set CritRes $critres
      }
      bind .w.msg.r.lv.v <Leave> {
         set CritRes $critres
      }
      set critres 10
      set CritRes $critres
   }
}
```

A.5. MIT/EmSim Verification Tools

In order to ensure that the integration routines in MIT/EmSim were calculating the stress evolution accurately, stress profiles in space and time were compared to that of PROMIS, a blah blah balh. The source code input to PROMIS is described in detail in the Ph.D. thesis of B.D. Knowlton [Kno_2 98]. Software tools created to generate these PROMIS input files for populations of lines with varying microstructures is discussed in Appendix B. Overlays of the stress profiles from MIT/EmSim and PROMIS were considered sufficient to verify the accuracy of the MIT/EmSim.

A further verification was performed by developing a version of MIT/EmSim in Matlab limited to pure metal and the incorporation of the stress effect on diffusivity. Overlays of the stress profiles was once again used to ensure the accuracy of MIT/EmSim. While the Matlab environment offered significantly faster calculation time due to its use of extremely advanced integration schemes, limitations in the data structure creation aspects of Matlab prevented further incorporation of the MIT/EmSim model. The source code for the Matlab m-file scripts are contained below.

The first file, runsim2.m, controls the calculation iterations of the script, as well as plotting aspects.

```
::::::::::::::
runsim2.m
::::::::::::::
function [t,y,stress] = runsim2(method)
% RUNSIM2   Run the EnSim simulation
%
% [T,Y,S] = runsim2('ode45') will run EnSim with ode45 (modified R-K).
% T is a vector where each element is the time corresponding to each
% row in Y which holds the solution matrix for concentration.
% S is a matrix of same size as Y holding the stresses for each time
% step.
%
% Use plot(S(end,:)) to plot the final stress versus cell.
% Use surf(S(1:step:end,:)) to plot a surface for every 'step'
timestep.
% Use rotate3d to rotate the surface plot
%

global prev_t;

prev_t = .001;

load physcon;
load data1;

if nargin < 1
  method = 'ode23s';
end

options = odeset('initialStep', 1e-5, 'RelTol',1e-5,'AbsTol',1e+17);
[t,y] = feval(method, 'emsim2',[0 5e+7], c0, options, Dratio, bc, pc,
c0);
c0 = c0';
for i = 1:length(t)
   stress(i,:) = -pc.b * log(y(i,:)./c0);
end
h = surf(s);
set(h, 'ydata',t);
shading flat;
lighting p;
material metal;
set(gca,'view',[-138 34]);
l = light('pos',[200 1e+011 1e+010]);
colormap(gray(255));
xl=xlabel('Cell Number');
yl=ylabel('Time (s)');
zl=zlabel('Stress (Pa)');
set([xl;yl;zl], 'units','norm');
set(gca,'yscale','log');
```

The file, emsim2.m, contains the electromigration calculation engine of the Matlab EM script.

```
::::::::::::::
emsim2.m
::::::::::::::
function F = emsim2(t, c, flag, Dratio, bc, pc, c0)

global prev_t;
global edf;
global kT;
global T;
global j;
global exp_const;


if (t == 0)
  T = 473;
  % calculate j
  j = 1e+10;
  % electromigration driving force
  edf  = pc.q*pc.e*pc.rho*j;
  kT = pc.k * T;
  exp_const = (pc.omega / kT + 1/pc.b);
  tic;
elseif (log(t) > (log(prev_t) + 1))
  et = toc;
  fprintf('deltaT: %g, elapsed time: %g\n', t, et);
  prev_t =t;
end

% calulate length of c and stress
cn = length(c);
stress = -pc.b * log(c./c0);


% normally per cell.
dx = 1e-6;

% calculate ave stress
ave_stress = [0; (stress(2:end) + stress(1:end-1))/2];

% stress driving force
sdf = [0; ((stress(2:end) - stress(1:end-1)) * pc.omega)/dx];

% calculate j
j = 1e+10;

% electromigration driving force
edf  = pc.q*pc.e*pc.rho*j;
T = 473;
kT = pc.k * T;
```

```
% calculate diffusivity
diff_calc = pc.diffu * exp((pc.omega / kT + 1/pc.b) .* ave_stress) .*
Dratio;

flux_in = zeros(n,1);
for i = 2:n
   flux_in(i) = ((c(i-1)/dx) * diff_calc(i) * (sdf(i) + edf)) / kT;
end
flux_out = [flux_in(2:end); 0];

if (bc.left == 0)
   flux_in(1) = flux_in(2);
else
   flux_in(1) = 0;
end

if (bc.right == 0)
   flux_out(end) = flux_in(end-1);
else
   flux_out(end) = 0;
end

denom = 1 + (pc.b * pc.omega / kT) * exp( (pc.omega / kT + 1 / pc.b) .*
stress - pc.qvac / kT );

F = (flux_in - flux_out) ./ denom;
```

This concludes the collection of code associated with MIT/EmSim. Appendix B contains utilities created to develop and analyze MIT/EmSim data.

Appendix B

Helper Applications

In completing this thesis, a number of tools and utilities were written to ease code generation, data generation, data analysis, and movie-making abilities. These tools and utilities are described below.

B.1. subline Code Generation Tool

During the initial verification stages of MIT/EmSim development, simple and more complex input data sets were used to ensure the accuracy of MIT/EmSim. This required frequent runs with lines of varying microstructures originating from GGSim to be simulated in both PROMIS and MIT/EmSim. To ease the creation of input files for PROMIS and MIT/EmSim, the subline tool was written. This tool is especially convenient for generating input files for populations of lines to be simulated under the same test conditions, but exhibiting different microstructures. This tool is composed of three files, and the associated Makefile. To run the code, the user runs the binary executable and follows the on-screen instructions. The Makefile and source code to subline is contained below:

```
::::::::::::::
Makefile
::::::::::::::
CC       = gcc
FLAGS    = -O3 -g
INCL     = -I/.
LIBS     = -lm

OBJS     = subline.o

.c.o:    ; $(CC) -c $(FLAGS) $(INCL) $*.c
```

```
subline:      $(OBJS)
        $(CC) $(FLAGS) $(INCL) $(OBJS) -o subline $(LIBS)

clean:
        rm -f subline.o core subline.c~ Makefile~ subline
promisbottom.inp~ promistop.inp~
```

Below is the code for subline.c, the primary source file for subline.

```
/***  subline.c

This version of Subline reads the output given by the
grain growth simulator, extract the endpoint values of high
diffusivity, and write these to a new file, endpoint.dat.

This program will then find the length of the line (maximum
value for length) and use this to determine an equi-spaced
sampling of "sub-lines" (usually ten) to be entered into
Promis for statistical purposes.

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "sublineheader.h"            /* holds structures, const */

/*** function prototypes ***/
void initialize(FILE *_fggrowth, int *_numfilelines, int
*_doinputfile);

void queryinfo(int *_numsublines, double *_sublinelength, int
*_boundarycond, float *_lowdiff, float *_highdiff);

void getendpoints(FILE *_fggrowth, struct diffrange *_xval, int
_numfilelines);

void setuparray(struct diffrange *_xval, double *_linelength, int
_numsublines, double  *_spacing, int _numfilelines);

void writepromistop(FILE *_fsubline, int _currsubline, char *_prefix,
double _startsubline, double _endsubline, float _lowdiff, float
_highdiff);

void writelines(FILE *_fsubline, struct diffrange *_xval, double
_numfilelines, double _startsubline, double _endsubline, int
_doinputfile);

void writepromisbottom(FILE *_fsubline, int _boundarcond);
```

```
/***==========================================================***/
int main(int argc, char *argv[])
{
  /*** variable declarations ***/

  int numfilelines,    /* how many lines in ggrowth data file    */
    numsublines,       /* number of sub-lines entered by user    */
    currsubline,       /* keeps track of which sub-line we're on */
    boundarycond;      /* 0 for zero flux, 1 for const stress    */

  float startsubline,  /* starting x value for current sub-line  */
    endsubline;        /* ending x value for current sub-line    */

  double spacing,      /* spacing between the lines              */
    linelength,        /* overall length of the line in microns  */
    sublinelength;     /* user entered length of each sub-line   */

  float lowdiff=1.55E-18,   /* bamboo diffusivity                */
    highdiff=3.1E-16;       /* cluster diffusivity               */

  int doinputfile=0;        /* set to 0=promis, set to 1=emsim     */

  char prefix[FILELEN];   /* filename w/o extension of ggrowth file */
  char suffix[3];           /* holds suffix for ggrowth file        */
  char suffixprom[3];       /* holds suffix for promis input file   */
  char fn_ggrowth[FILELEN]; /* filename of grain growth data        */
  char fn_subline[FILELEN]; /* filename for sub-line output data    */

  struct diffrange *xval;   /* parallel array holds endpoint values */

  /*** File pointer declarations ***/
  FILE *fggrowth;           /* grain growth simulator output file    */
  FILE *fsubline;           /* local file referring to each sub-line */

/*** MAIN code starts ***/

  /*** open userfilename and assign as *fggrowth ***/
  if (argc < 2 )
    printf("Must indicate input data file...\n\n");
  strcpy(prefix, argv[1]);
  strcpy(suffix, ".dat");                   /* for ggrowth file */
  strcpy(suffixprom, ".inp");               /* for promis file  */
  sprintf(fn_ggrowth, "%s%s", prefix, suffix);
  if ( (fggrowth = fopen(fn_ggrowth, "r")) == NULL )
    printf("Can't open grain growth file, you idiot\n");

  /*** allocate memory for xval struct array ***/
  initialize(fggrowth, &numfilelines, &doinputfile);
  xval = malloc(sizeof(struct diffrange)*((int)(numfilelines+1)));

  /*** close and re-open file to start at top of file ***/
  close(fggrowth);
  if ( (fggrowth = fopen(fn_ggrowth, "r")) == NULL )
```

```
    printf("Can't open grain growth file, you idiot\n");

  /*** get user input settings ***/
  queryinfo(&numsublines, &sublinelength, &boundarycond, &lowdiff,
&highdiff);

  /*** get endpoints from grain growth data file ***/
  getendpoints(fggrowth, xval, numfilelines);

  /*** determine sub-line spacing, put x values in order ***/
  setuparray(xval, &linelength, numsublines, &spacing, numfilelines);

  /*** write data for each sub-line in Promis syntax at proper spacing
***/
  for ( currsubline=1; currsubline<=numsublines; currsubline++) {
    /*** name and open current sub-line file ***/
    sprintf(fn_subline, "%i%s%s", currsubline, prefix, suffixprom);
    if ( (fsubline=fopen(fn_subline, "w")) == NULL )
      printf("Error in writing to the sub-line file %s\n", fn_subline);

    /*** advance currsubline + 1/2 spacing ***/
    startsubline =  ( (currsubline - 0.5)*spacing - (sublinelength)/2);
    endsubline = startsubline + sublinelength;
    printf("startsub=%f endsub=%f\n", startsubline, endsubline);

    /*** write PROMIS input file if necessary ***/
    if ( doinputfile == 0 ) {
      writepromistop(fsubline, currsubline, prefix, startsubline,
endsubline, lowdiff, highdiff);
      writelines(fsubline, xval, numfilelines, startsubline,
endsubline, doinputfile);
      writepromisbottom(fsubline, boundarycond);
      close(fsubline);
    }
    /*** write EmSim input file if necessary ***/
    if (doinputfile == 1 ) {
      fprintf(fsubline, "%e, %e \n", lowdiff, highdiff);
      writelines(fsubline, xval, numfilelines, startsubline,
endsubline, doinputfile);
    }

  }   /* end of for loop */

  /*** close remaining files ***/
  close(fggrowth);

  printf("\nProgram complete....\n");

}   /** end of MAIN **/

/***=========================================================***/
void initialize(FILE *_fggrowth, int *_numfilelines, int *_doinputfile)
/***
Determines number of file lines in grain growth data file
```

```
so we can allocate xval array at that length.  Also
determines whether to form EmSim or PROMIS input file.
***/
{
  int f=1;                  /* number of file lines */
  char temptext[TEXTLEN];   /* dummy string for getting numfilelines */

  /*** count # of file lines with f and record for array size ***/
  while ( fgets(temptext, TEXTLEN, _fggrowth) != NULL)
    f++;
  *_numfilelines = f;

  /*** determine whether to create PROMIS or EmSim input file ***/
  printf("\nSelect input file to form: \n\n");
  printf("\t (0) PROMIS input file \n");
  printf("\t (1) EmSim input file \n\n");
  printf("Enter your choice: ");
  scanf("%i", _doinputfile);

  printf("\nDone with initialize...\n");

}  /** end of initialize **/

/***------------------------------------------------------------***/
void queryinfo(int *_numsublines, double *_sublinelength, int
*_boundarycond, float *_lowdiff, float *_highdiff)
     /***
       Queries the user for various input parameters listed below.
***/
{
  char *diffchoice;           /* letter choice for changing diffusivity
*/

  /*** find out # of sub-lines to form ***/
  printf("\nEnter number of sub-lines: ");
  scanf("%i", _numsublines);
  printf("numsublines=%i", *_numsublines);

  /*** find out sub-line length ***/
  printf("\nEnter sub-line length (in microns): ");
  scanf("%lg", _sublinelength);
  printf("\nsublinelength=%f", *_sublinelength);

  /*** get diffusivity values ***/
  printf("\n\nBamboo (low) diffusivity is set at:   %e \n", *_lowdiff);
  printf("Cluster (high) diffusivity is set at:   %e \n\n",
*_highdiff);
  printf("Press (a) to accept or (c) to change : ");
  scanf("%s", diffchoice);
  if ( *diffchoice == 'c' ) {
    printf("\nEnter bamboo (low) diffusivity: ");
    scanf("%e", _lowdiff);
    printf("\nEnter cluster (high) diffusivity: ");
    scanf("%e", _highdiff);
```

```
    printf("\n\nBamboo (low) diffusivity is set at:   %e \n",
*_lowdiff);
    printf("Cluster (high) diffusivity is set at:   %e \n\n",
*_highdiff);
  }

  /*** find out boundary conditions ***/
  printf("\nSelect boundary conditions: \n\n");
  printf("\t (0) Zero flux \n ");
  printf("\t (1) Constant stress \n\n");
  printf("Enter your choice: ");
  scanf("%i", _boundarycond);
  printf("boundarycond=%i", *_boundarycond);

printf("\nDone with queryinfo...\n");

}  /** end of queryinfo **/

/***-----------------------------------------------------***/
void getendpoints(FILE *_fggrowth, struct diffrange *_xval, int
_numfilelines)
/***
Takes the grain growth output file and removes the
extranneous text leaving two columns containing the
endpoints of high diffusivity.  Writes data to parallel
arrays in *_xval, starting at array index = 1.  Makes
use of function getx to perform the actual extraction
for each line.
***/
{
  int  f;                    /*  index for finding # of file lines */
  char textline[TEXTLEN];    /* stores text line from ggrowth file */
  char *textptr;
  int xmarker;

  /*** extract *_xvals from each do loop ***/
  for ( f=1; (fgets(textline, TEXTLEN, _fggrowth)!= NULL); f++ ) {
    textptr = &textline[0];
    xmarker=0;

    /*** segregate first xval number ***/
    while ( isdigit(*textptr++) == 0 ) {     /* indicates not a digit
*/
      xmarker++;
    }
    textptr = &textline[xmarker];
    _xval[f].start= atof(textptr);

    /*** segregate second xval number ***/
    while ( isdigit(*textptr++) != 0 ) {
      xmarker++;
    }
    xmarker += 3;                          /* gets past first number */
    textptr = &textline[xmarker];
```

```
    while ( isdigit(*textptr++) == 0 ) {  /* indicates not a digit  */
      xmarker++;
    }
    textptr = &textline[xmarker];
    _xval[f].end = atof(textptr);
    /* printf("\n Loop %i: start=%e end=%e", f,_xval[f].start,
_xval[f].end); */
  }    /* end of giant for loop */

  printf("\n\nDone with getendpoints! \n");

}  /** end of getendpoints **/

/***----------------------------------------------------***/
void setuparray(struct diffrange *_xval, double *_linelength, int
_numsublines, double  *_spacing, int _numfilelines)
/***
First determines the length of the entire line, and given
the number of statistical sub-lines to use, it determines
the proper spacing to use for sampling points for sub-lines.
Finally, having noted when the x values loop back to zero,
it rearranges the file at this point so the endpoints are
in logical order.
***/

{
  /*** variables ***/
  int i,j,          /* scales for xval and tempx respectively */
    index,          /* index used to find length of line */
    maxxindex;      /* index in struct containing max x */
  double maxx=0.0;   /* max value for x */
  struct diffrange *tempx;    /* temp struct for holding x vals in loop
*/


  /*** allocate temp memory for tempx ***/
  tempx = malloc(sizeof(struct diffrange)*((int)(_numfilelines+1)));

  /*** find length of entire interconnect line (maxx), copy array ***/
  for ( index=1; index<=(_numfilelines); index++) {
    if (_xval[index].end > maxx) {
      maxx = _xval[index].end;
      maxxindex = index;
    }
    tempx[index].start = _xval[index].start;
    tempx[index].end = _xval[index].end;
  }
  *_linelength = maxx;

  /*** find spacing ***/
  *_spacing = (*_linelength/_numsublines);
  printf("\nlinelength=%f sublines=%i spacing=%f\n", *_linelength,
_numsublines, *_spacing);
  printf("maxxindex=%i maxx=%.1f \n", maxxindex, maxx);
```

```
  /*** loop this file back on itself so x starts at zero ***/
  for ( i=1; i<(_numfilelines-maxxindex); i++) {
    _xval[i].start = tempx[i+maxxindex].start;
    _xval[i].end = tempx[i+maxxindex].end;
  }
  for ( i=(_numfilelines-maxxindex),j=1; i<=_numfilelines,j<=maxxindex;
i++,j++) {
    _xval[i].start = tempx[j].start;
    _xval[i].end = tempx[j].end;
  }
  printf("\nDone with setuparray\n");

}    /** end of setuparray **/

/***-------------------------------------------------------***/
void writepromistop(FILE *_fsubline, int _currsubline, char *_prefix,
double _startsubline, double _endsubline, float _lowdiff, float
_highdiff)
/***
Writes the top half of the Promis input file for this
particular sub-line
***/
{
  int c;                        /* holds temp character        */
  float gridpts,                /* # of gridpoints for promis  */
        dratio;                 /* diffusivity ratio, low/high */
  char fn_promistop[FILELEN];   /* filename for top input data */
  FILE *fpromistop;             /* file pointer for top promis */

 /*** write personalized syntax for promis file ***/
  gridpts = (_endsubline-_startsubline) * 10 + 1;
  fprintf(_fsubline, "CO EQ=1 NV=%.0f NL=3 CF=", gridpts);
  fprintf(_fsubline, "'%i%s.CNTRL'\n", _currsubline, _prefix);
  fprintf(_fsubline, "+  TX='GP=%.0f P=5.0 tm=1000000.0 j=10.0 T=473.0'
\n", gridpts);
  fprintf(_fsubline, "+  PF='%i%s.vac' \n* \n", _currsubline, _prefix);
  fprintf(_fsubline, "* Boundary conditions: S(0,t)=0 and S(L,t)=0
\n");
  fprintf(_fsubline, "GR EV=%.0f \n* QU PO=1 EL=VACANCY \n", gridpts-
2);
  fprintf(_fsubline, "QU PO=1 EL=ARBITRARY ID=100 LI=OFF \n");
  fprintf(_fsubline, "AR EB=1. WB=-1. NB=%.0f SB=%.0f \n",
_startsubline, _endsubline+1);

  /*** open and assign promistop.inp ***/
  strcpy(fn_promistop, "/user2/vab/bin/promistop.inp");
  if ( (fpromistop=fopen(fn_promistop, "r")) == NULL )
    printf("Missing file promistop.inp");

  /*** copy rest of data over into sub-line file ***/
  while ( (c=fgetc(fpromistop)) != EOF )
    fputc(c, _fsubline);
```

```
  /*** write diffusivity ratio ***/
  dratio = _lowdiff / _highdiff;
  fprintf(_fsubline, "      Dratio = %.5f \n", dratio);

  fclose(fpromistop);
  printf("\nDone with writepromistop\n");

}  /** end of writepromistop **/

/***----------------------------------------------------------***/
void writelines(FILE *_fsubline, struct diffrange *_xval, double
_numfilelines, double _startsubline, double _endsubline, int
_doinputfile)
/***
Given start and end (last) x values of the line to sample
from, this function reads the high diffusivity range values
from the two columns and writes them in Promis syntax to
a specific output file.
***/
{
  int j;                /* tells what element in array we're at */

  /*** write xvals for length of sub-line ***/
  for ( j=1; j<=_numfilelines-1; j++) {
    if ( (_xval[j].start>_startsubline) && (_xval[j].end<_endsubline)
&& (_xval[j].start<_endsubline) && (_xval[j].end>_startsubline) ) {
      if (_doinputfile == 0) {
      fprintf(_fsubline, "      IF ((POSX .GE. %.1f) .AND. ",
_xval[j].start);
      fprintf(_fsubline, "(POSX .LE. %.1f)) Dratio=1. \n",
_xval[j].end);
      }
      else if (_doinputfile == 1)
      fprintf(_fsubline, "%.2f, %.2f \n", _xval[j].start,
_xval[j].end);
    }
  }

}  /** end of writelines **/

/***----------------------------------------------------------***/
void writepromisbottom(FILE *_fsubline, int _boundarycond)
/***
Writes the bottom half of the Promis input file for this
particular sub-line
***/
{
  int c;                          /* holds temp character        */
  char fn_promisbottom[FILELEN];  /* filename for top input data */
  FILE *fpromisbottom;            /* file pointer for top promis */

  /*** open and assign promistop.inp ***/
  strcpy(fn_promisbottom, "/user2/vab/bin/promisbottom.inp");
  if ( (fpromisbottom=fopen(fn_promisbottom, "r")) == NULL )
```

```
    printf("Missing file promisbottom.inp");

 /*** write portion relating to alpha ***/
  fprintf(_fsubline, "      IC(1) = 10 00 10 00 \n");
  fprintf(_fsubline, "      CA(1,1) = -1.*Dratio \n");
  fprintf(_fsubline, "C  CC(1,1) = Z*epjL/kT \n");
  fprintf(_fsubline, "      CC(1,1) = -.15*Dratio \n\n      END\n");

  /*** copy data over into sub-line file ***/
  while ( (c=fgetc(fpromisbottom)) != EOF )
    fputc(c, _fsubline);

  /*** write rest of input file, include B.C. ***/
  if ( _boundarycond == 0 ) {
    fprintf(_fsubline, "      IB(1) = 10 00 \n");
    fprintf(_fsubline, "      BB(1,1) = 1. \n\n");
  }
  if ( _boundarycond == 1 ) {
    fprintf(_fsubline, "      IB(1) = 00 11 \n");
    fprintf(_fsubline, "      BF(1) = W(1)-VALUE \n");
    fprintf(_fsubline, "      SBF(1) = MAX(ABS(W(1)),VALUE) \n");
    fprintf(_fsubline, "      DBF(1,1) = 1. \n");
  }

  /*** finish off promis file ***/
  fprintf(_fsubline, "      ELSEIF (BS) THEN \n");
  fprintf(_fsubline, "      IB(1) = 00 11 \n");
  fprintf(_fsubline, "      BF(1) = W(1)-VALUE \n");
  fprintf(_fsubline, "      SBF(1) = MAX(ABS(W(1)),VALUE) \n");
  fprintf(_fsubline, "      DBF(1,1) = 1. \n      ENDIF \n \n      END
\n");

  /*** close promis data file ***/
  fclose(fpromisbottom);

  printf("\nDone with writepromisbottom\n");

}  /** end of writepromisbottom **/

/***-------------------------------------------------------***/
```

The header file for subline, sublineheader.h, is contained below.

```
::::::::::::::
sublineheader.h
::::::::::::::
/***  subline.h ***/

/***
Contains structures for use in program ready4promis.c .
***/

struct diffrange {
```

```
  double start;            /* gives starting x for high diff */
  double end;              /* gives ending x for high diff   */
} ;

#define MAXLEN 1E5
#define FILELEN 75
#define TEXTLEN 80
#define NUMLEN 10
```

The following is the upper half of the PROMIS input file, which contains the text input information for the top half of PROMIS input files, and for my runs was written as follows:

```
::::::::::::::
promistop.inp
::::::::::::::
IN IR=/LINIR/ TE=473.
* PR TX='DC TEST CASE: IC'
TR TD=/LINTD/ JV=/LINJ/ BD=/LINBD/ LS=/ONEDIM/
+  ST=0. ET=1000000. DT=1.E-5 TS=LOG TE=473. DS=3
+  TV=/VALUES/ PL=0.
*SV(1:3)=(0.,0.5,1.) NV(1:2)=(499,499)
*PR TX='TEMP GRAD; C(0)=Co, C(l)=Co: T=0.01'
**TR ET=0.001 CO=ON
*TR ET=0.01 CO=ON
*TR ET=0.1 CO=ON
*TR ET=1.0 CO=ON
*TR ET=10. CO=ON
*PR TX='TEMP GRAD; C(0)=Co, C(l)=Co: T=10.'
PR QU=TV TX='Vacancy Conc vs Reduced Time; S(0,T)=0, S(L,T)=0'
*SA TX='TEMP GRAD; J(0)=0, C(l)=Co' FN='TG_S2_C0_C0_B2D20.SAVE' ST=RE
*SA QU=TV TX='STRESS VS TIME,POSITION' FN='TG_DATA2.SAVE' ST=RE
EN
      SUBROUTINE LINIR(
     D                NEQU,NPTX,NPTY,NAUXF,NUSP,
     V                W,
     I                POSX,POSY,TEMP,TIME,AUXF,USPAR,NELE)
      INCLUDE '$PROMISINC:TYPE.INC'
      DIMENSION W(NEQU,NPTY,NPTX)

      DO 20 IPTX=1,NPTX
      DO 10 IPTY=1,NPTY
       W(1,IPTY,IPTX) = 1.
10    CONTINUE
20    CONTINUE

      END
      SUBROUTINE LINTD(
     D                NEQU,NAUXF,NAUXP,NUSP,
```

```
     O                ITD,TD,STD,DTD,
     I                W,POSX,POSY,TEMP,TIME,
     I                AUXF,AUXPAR,USPAR)
      INCLUDE '$PROMISINC:TYPE.INC'
      DIMENSION ITD(NEQU),TD(NEQU,NEQU)

      ITD(1) = 10
C  The value of TD(1,1) is supposed to be = CC(1,1)**2 where
C  CC(1,1) is defined in LINJ below.
      TD(1,1) = 0.0225

      END
      SUBROUTINE LINJ(
     D                NEQU,NAUXF,NAUXP,NUSP,
     O                IC,CA,CB,CC,CD,SCA,SCB,SCC,SCD,
     O                DCA,DCB,DCC,DCD,D1CA,D1CB,D1CC,D1CD,IPSI,
     I                W,W1,POSX,POSY,POSX1,POSY1,TEMP,TIME,
     I                AUXF,AUXF1,AUXPAR,USPAR)
      INCLUDE '$PROMISINC:TYPE.INC'
      DIMENSION IC(NEQU),CA(NEQU,NEQU),CC(NEQU,NEQU)
```

The following is the text contents of promisbottom.inp, which contains the PROMIS input file information for the bottom half of the PROMIS input file.

```
::::::::::::::
promisbottom.inp
::::::::::::::
       SUBROUTINE LINBD(
     D                NEQU,NAUXF,NAUXP,NUSP,
     O                IB,BB,BF,SBB,SBF,DBF,D1BF,
     I                W,W1,D1,POSX,POSY,TEMP,TIME,
     I                BN,BS,BE,BW,
     I                AUXF,AUXPAR,USPAR)
      INCLUDE '$PROMISINC:TYPE.INC'
      LOGICAL BN,BS,BE,BW
      DIMENSION IB(NEQU)
      DIMENSION BB(NEQU,NEQU),BF(NEQU),SBF(NEQU),DBF(NEQU,NEQU)
      DIMENSION W(NEQU),AUXPAR(NAUXP)

      VALUE = 1.
      IF (BN) THEN
```

B.2. thermacore Input Tool

To incorporate the Suresh model for estimating initial thermal stresses in interconnects, a stress estimator based on curve-fits was written to provide stress inputs to MIT/EmSim. This tool, thermacore, accepts user information about geometry and

thermal history to determine the stress. This tool is easily incorporated into MIT/EmSim, or may be run as a stand-alone program. The Makefile and source code for thermacore is contained below:

```
#######################################################################
# Makefile for thermacore
#
# Vaibhav Andleigh
# 24-Apr-98
#
# (c) copyright 1999, 2000, Massachusetts Institute of Technology
#                           All rights reserved.
#######################################################################

BIN    = thermacore

CC     = gcc
FLAGS  = -O3
INCL   = -I/.
LIBS   = -lm

# for optimized runs, use gcc with flag -O3 only
# for debugging, use cc with flags -g -Wall which eliminates
optimization

OBJS   = thermacore.o \
         thermacore_prog.o

OBJS_C       = thermacore.o \
         thermacore_prog.o

OUTPUT       = error.dat

#######################################################################

.c.o: ; $(CC) -c $(FLAGS) $(INCL) $*.c

#######################################################################

go:    clean rm_dat thermacore go_bg

gdb:   rm_dat
       gdb $(BIN)

run:   rm_dat
       $(BIN)

emsim:       $(OBJS)
       $(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

therm:       $(OBJS)
```

```
	$(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

clean:		rm_dat
	rm -f core *~ $(BIN) $(OUTPUT) $(OBJS_C)

rm_dat:
	rm -f *.dat
```

The code for the main thermacore file, thermacore.c is contained below.

```
::::::::::::::
thermacore.c
::::::::::::::
/***
 ** This version of thermacore computes
 ** the thermal stress based on a variety of mechanical parameters
 ** already contained in the code.  The thermacore.c file simply calls
 ** the actual thermal computation program.  This is done to allow
 ** thermacore to function both as a stand-alone program as well as to
 ** allow its incorporation into EmSim as well.

Command line:     thermacore

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/*** define constants and data structures ***/
#include "thermacore.h"

/*** function prototypes ***/
#include "thermacore_prog.h"

/***=====================================================***/
int main(int argc, char *argv[])
{
  /*** variable declarations ***/
  double thermal_stress;           /* final value of thermal stress */

  /*** ---------------------------------------------------------- ***/

  /*** MAIN code starts ***/
  printf("\nThermacore Program Begins...\n");

  /*** call thermal stress function ***/
  thermal_stress = calc_thermal_stress();

  /*** print result ***/
```

```
  printf("thermal stress is %e\n", thermal_stress);

  /*** closing statement ***/
  printf("\nProgram Finished...\n\n");

  /*** exit program successfully ***/
  return(0);

}   /** end of MAIN **/
```

The source code to a support file, thermacore_prog.c, which performs the actual stress computation, is contained below:

```
/***   thermacore.c support file   ***/

/*** libraries ***/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*** define data structures ***/
#include "thermacore.h"

/*** function prototypes ***/
#include "thermacore_prog.h"

/*** function listing **********

double calc_thermal_stress()

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

/***-------------------------------------------------------------------***
 * Takes no input but calculates the thermal stress based on the
mechanical
 * behaviors.  Returns the value of the thermal stress in the end.
/***===================================================================***/
double calc_thermal_stress()
{
  /*** dependent variables ***/
  double sigma1,            /* stress in normal direction     */
         sigma2,            /* stress in some other direction */
         sigma3,            /* stress in a third direction    */
         sigmaH,            /* hydrostatic stress (desired)   */
         M,                 /* some complex fraction          */
         N,                 /* some complex fraction          */
         P,                 /* some complex fraction          */
         Q,                 /* some complex fraction          */
         R,                 /* some complex fraction          */
         Y,                 /* some complex fraction          */
         Z;                 /* some complex fraction          */
```

```
/*** independent variables ***/
double E_p = 2e9,
       E_l = 3e9,
       nu_p = 0.31,
       nu_l = 0.33,
       f_p = 0.5,
       f_l = 0.3,
       alpha_p = 0.4,
       alpha_l = 0.6,
       alpha_s = 0.9;

/*** my variables ***/
double nu_by_el,            /* equals 1+nu divided by E for l */
       nu_by_ep,            /* equals 1+nu divided by E for p */
       temp;

/*** define my variables ***/
nu_by_el = (1 + nu_l) / E_l;
nu_by_ep = (1 + nu_p) / E_p;

/*** define M and N ***/
M  = -1 * nu_by_el * f_l;;
M /=      nu_by_ep * f_p;
N = 1 - M;

/*** define Y and Z ***/
Y = alpha_s - f_l * alpha_l - f_p * alpha_p;
Z = alpha_p - alpha_l;

/*** define P ***/
P  = f_l/E_l + f_p/E_p * M * Y;
P /= -1 * (nu_l * f_l/E_l + nu_p * f_l/E_p);

/*** define Q ***/
Q  = -1*nu_l * f_l / E_l + f_p / E_p * N - nu_p * f_p / E_p;
Q /=  -1 * (nu_l * f_l/E_l + nu_p * f_l/E_p);

/*** define R ***/
R  = -1 * (nu_by_el - nu_by_el*P - nu_by_ep*M + nu_by_ep*P* f_l /
f_p);
R /= -1 * nu_by_el*Q - nu_by_ep*N + nu_by_ep*Q* f_l / f_p;

/*** define sigma1 ***/
temp  = (1-P*nu_l)/E_l + 1/E_p * (f_l/f_p * nu_by_ep / nu_by_el);
temp += R*(-1*nu_l/E_l*(1+Q) + 1/E_p*(nu_p - N + nu_p * f_l * Q) );
temp /= alpha_p - alpha_l;
sigma1 = 1 / temp;

/*** define sigma2 ***/
sigma2 = R * sigma1;

/*** define sigma3 ***/
sigma3 = P * sigma1  +  Q * R * sigma1;
```

```
  /*** define hydrostatic stress ***/
  sigmaH = (sigma1 + sigma2 + sigma3) / 3;

  /*** return function value ***/
  return(sigmaH);

}  /** end of function calc_thermal_stress() **/

/***----------------------------------------------------------------***/

/***-================================================================-***/
```

The header files to thermacore are contained below.

```
/*** thermacore.h support file ***/

/***
Contains data structures used throughout thermacore.h
***/

/*** constants ***/
#define MAXLEN 1E5
#define FILELEN 75
#define TEXTLEN 90
#define NUMLEN 10

/*** definitions ***/

/*** debugging functions ***/
#define PRINT_ID(x)      printf(#x "->id = %i\n\n", x->id)
#define PRINT_NUM(x)     printf(#x " = %e\n\n", x)
#define PRINT_INT(x)     printf(#x " = %i\n\n", x)
#define TEST(i)          printf("\n Made it here to %i ! \n\n", i)
/*============================================*/


/***  thermacore_prog.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in thermacore_prog.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef THERMACORE_PROG_H
#define THERMACORE_PROG_H

/*** function prototypes ***/

double calc_thermal_stress();

#endif  /* THERMACORE_PROG_H */
```

B.3. find_ttf Analysis Tool

MIT/EmSim and PROMIS provides a wealth of information regarding the evolving stresses and atom concentrations as a function of time and position in an interconnect. Because of the large size of these data files, the find_ttf tool was developed to analyze PROMIS and MIT/EmSim output files for several different failure criterion including tensile failure, dielectric failure, void nucleation failure, resistance failure, and void-volume-based failure. Finally, find_ttf may also be used to track stresses or void sizes at a particular instant of simulated time. To run this tool, run the binary and follow the on-screen instructions. The Makefile and source code are contained below:

```
::::::::::::::
Makefile
::::::::::::::
CC      = gcc
FLAGS   = -O3 -g
INCL    = -I/.
LIBS    = -lm

OBJS    = find_ttf.o

.c.o:   ; $(CC) -c $(FLAGS) $(INCL) $*.c

go:     clean find_ttf

find_ttf:     $(OBJS)
        $(CC) $(FLAGS) $(INCL) $(OBJS) -o /vab/bin/find_ttf $(LIBS)

clean:
        rm -f find_ttf.o core find_ttf.c~ Makefile~ find_ttf
```

The source code to find_ttf.c is listed below. No other files are required for find_ttf.

```
/*** find_ttf.c

This version of find_ttf reads in the output files from
emigdata (*.max, *.ten, *.cmp) and finds the time to
failure for each line according to a given critical stress.
These times are written to an outputfile, ttf*.dat .  A
population of lines numbered by the user are handled by this
program.  Alternatively, this program may be used to determine
void growth failures based on a certain void size.
```

```
Command line:     find_ttf
               or find_ttf <option> <filename>

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/*** function prototypes ***/
void write_to_promis_file(char *_inputfile, FILE *_outputfile);

/*** constants ***/
#define MAXLEN 1E5
#define FILELEN 75
#define TEXTLEN 90
#define NUMLEN 10

/***=====================================================***/
int main(int argc, char *argv[])
{
  /*** variable declarations ***/

  int numsublines,         /* number of sub-lines entered by user    */
    currsubline,           /* keeps track of which sub-line we're on */
    first_subline,         /* numerical value for first subline      */
    last_subline,          /* numerical value of last subline        */
    do_emsim,              /* if equals 1, opens maxstress.dat       */
    maxstress_type;        /* type of maxstress.dat file w/ or w/o 1 */

  char *choice;            /* text equivalent of do_emsim for command*/
  float time,              /* temporary var for time from input file */
        stress,            /* temp var for stress from input file    */
        ttf,               /* time to failure for current line       */
        ttf_cmp,           /* time to failure for tensile stress     */
        ttf_ten,           /* time to failure for compressive stress */
        length,            /* length of void at the critical time    */
        crit_stress,       /* critical stress for all the lines      */
        crit_len,          /* critical length of void until failure  */
        crit_time,         /* critical time to get void length       */
        current_density;   /* current density for this line          */

  float dummy1,            /* dummy variable for holding delta t     */
        dummy2,            /* dummy variable for holding max_ten_x   */
        dummy3,            /* dummy variable for holding max_cmp_x   */
        dummy4,            /* dummy variable for holding strand id   */
        dummy5,            /* dummy variable for holding strand id   */
        dummy6,            /* dummy variable for holding clock_sec   */
        cmp_stress,        /* hold compressive stress val from emsim */
        ten_stress,        /* hold tensile stress value from emsim   */
```

```
      void_len;         /* hold void length (delta_x) from emsim  */

  char prefix[FILELEN];  /* file name w/o extension of input file  */
  char suffix[3];        /* holds suffix for input stress file     */
  char label[6];         /* holds text label for the line (set)    */
  char fn_input[FILELEN]; /* filename of grain growth data         */
  char fn_ttf[FILELEN];  /* filename for sub-line output data      */
  char fn_crit_stress[FILELEN]; /* filename for critical stress data*/
  char fn_all_dat[FILELEN]; /* filename for all ttf data output    */
  char fn_voids[FILELEN];   /* filename for void input file        */
  char fn_crit_len[FILELEN];/* filename for voidcritical length info*/
  char fn_crit_time[FILELEN]; /* filename for critical time info   */

  /*** File pointer declarations ***/
  FILE *finput;             /* grain growth simulator output file   */
  FILE *fttf;               /* local file referring to each sub-line*/
  FILE *fcritstress;        /* local file holding critical stresses */
  FILE *f_all_dat;          /* file to contain all data for a run   */
  FILE *fvoids;             /* local file holding void growth data  */
  FILE *fcrit_len;          /* file in bin dir with critical lengths*/
  FILE *fcrit_time;         /* file in bin dir with critical times  */

/*** MAIN code starts ***/

  /*** only query input if not given in command line ***/
  if( argc > 1 ) {
    choice = argv[1];
    sscanf(choice, "%i", &do_emsim);
    strcpy(prefix, argv[2]);
    strcpy(suffix, "dat");
    strcpy(label, argv[3]);
    choice = argv[4];
    sscanf(choice, "%i", &maxstress_type);
  }
  else {
    /*** determine if from Emsim data or from Emigdata ***/
    printf("\nWhere did your data come from: \n\n");
    printf("\t(0) Emigdata TTF results from a PROMIS run \n");
    printf("\t(1) EmSim TTF results \n");
    printf("\t(2) EmSim TTF results with maxstress.dat \n");
    printf("\t(3) EmSim TTF results with void length failure \n\n");
    printf("\t(4) EmSim length results with void growth \n");
    printf("\t(5) EmSim TTF tensile results with maxstress.dat \n");
    printf("\t(6) EmSim TTF compressive results with maxstress.dat
\n\n");
    printf("Enter your choice (0,1,2,3,4,5 or 6):  ");
    scanf("%i", &do_emsim);

    /*** open userfilename and assign as *fggrowth ***/
    if ( (do_emsim == 2) || ((do_emsim>=5) && (do_emsim<=6)) ) {
      strcpy(prefix, "maxstress");
      strcpy(suffix, "dat");
    }
    else if ( (do_emsim == 3) || (do_emsim == 4) ) {
```

```
      strcpy(prefix, "void1_0.050");
      strcpy(suffix, "dat");
    }
    else {
      printf("\nEnter name of file (w/o suffix):  ");
      scanf("%s", prefix);
      if ( do_emsim == 0 ) {
      printf("\nEnter the type of file (max, ten, cmp):  ");
      scanf("%s", suffix);
      }
      else
      strcpy(suffix, "dat");
    }
  } /* end of query input */

  /*** open file for all ttf data output ***/
  if ( (do_emsim >= 2) && (do_emsim <= 6) ) {
    if( do_emsim == 2 )
      strcpy(fn_all_dat, "../all_ttf.dat");
    if( (do_emsim == 3) && (maxstress_type == 1) )
      strcpy(fn_all_dat, "../resvoid_ttf.dat");
    if( (do_emsim == 3) && (maxstress_type == 2) )
      strcpy(fn_all_dat, "../voidlen_ttf.dat");
    /* strcpy(fn_all_dat, "../resvoidnucl_ttf.dat"); */
    if( do_emsim == 4 )
      strcpy(fn_all_dat, "../void_len_at_time.dat");
    if( do_emsim == 5 )
      strcpy(fn_all_dat, "../ten_ttf.dat");
    if( do_emsim == 6 )
      strcpy(fn_all_dat, "../cmp_ttf.dat");
    if ( (f_all_dat = fopen(fn_all_dat, "a")) == NULL )
      printf("Can't open all ttf output file, you idiot\n");
  }
  /*** ------------------------------------------------------------ ***/
  /*** analyze voids data for ttf ***/
  if ( do_emsim == 3 ) {
    /*** open void critical length failure file ***/
    if(maxstress_type == 2)
      strcpy(fn_crit_len, "/vab/common/crit_len.dat");
    else
      strcpy(fn_crit_len, "/vab/common/critres_len.dat");
    if ( (fcrit_len = fopen(fn_crit_len, "r")) == NULL )
      printf("Can't open critical length void input file, you
idiot\n");

    /*** put text label into f_all_dat file ***/
    fprintf(f_all_dat, "%s \t", label);
    printf("\n Line being tested:  %s\n", label);

    /*** find void failure data for each critical length ***/
    fscanf(fcrit_len, "%e", &crit_len);
    while ( crit_len != -1 ) {
      /*** write to all_data file ***/
      ttf = -1;
```

```
      /*** open file for all voids data input ***/
      sprintf(fn_voids, "%s.%s", prefix, suffix);
      if ( (fvoids = fopen(fn_voids, "r")) == NULL )
      printf("Can't open voids input file, you idiot\n");
      while( fscanf(fvoids, "%e %e %e %e %e %e %e", &time, &dummy1,
&dummy2, &dummy3, &void_len, &dummy4, &dummy5) != EOF )
      if ( void_len >= crit_len ) {
        ttf = time;
        while( fscanf(fvoids, "%e %e %e %e %e %e %e", &time, &dummy1,
&dummy2, &dummy3, &void_len, &dummy4, &dummy5) != EOF )
          ;      /* advance to end of file */
        fclose(fvoids);
        fprintf(f_all_dat, "%e \t", ttf);
      }
      printf("\ncritical length = %e um \n", crit_len);
      printf("ttf = %e hrs \n", ttf);
      fscanf(fcrit_len, "%e", &crit_len);
    }  /* end of crit_len while loop */
    crit_stress = -1;
  }  /* end of option 3 */

 /*** ------------------------------------------------------------ ***/
  /*** analyze voids data for length data at ttf ***/
  if ( do_emsim == 4 ) {
    /*** open critical time file ***/
    strcpy(fn_crit_time, "/vab/common/crit_time.dat");
    if ( (fcrit_time = fopen(fn_crit_time, "r")) == NULL )
      printf("Can't open critical time input file, you idiot\n\n");

    /*** put text label into f_all_dat file ***/
    fprintf(f_all_dat, "%s \t", label);
    printf("\n Line being tested:  %s\n", label);

    /*** find void length data for each critical time ***/
    fscanf(fcrit_time, "%e", &crit_time);
    while ( crit_time != -1 ) {
      /*** write to all_data file ***/
      ttf = -1;

      /*** open file for all voids data input ***/
      sprintf(fn_voids, "%s.%s", prefix, suffix);
      if ( (fvoids = fopen(fn_voids, "r")) == NULL )
      printf("Can't open voids input file, you idiot\n\n");

      /*** make sure void has formed first ***/
      fscanf(fvoids, "%e %e %e %e %e %e", &time, &dummy1, &dummy2,
&dummy3, &void_len, &dummy4);
      if( time > crit_time )
      fprintf(f_all_dat, "%e \t", 0.000001);
      else
      while( fscanf(fvoids, "%e %e %e %e %e %e", &time, &dummy1,
&dummy2, &dummy3, &void_len, &dummy4) != EOF )
        if( time >= crit_time ) {
```

```
          length = void_len;
          while( fscanf(fvoids, "%e %e %e %e %e %e", &time, &dummy1,
&dummy2, &dummy3, &void_len, &dummy4) != EOF )
             ;      /* advance to end of file */
          fclose(fvoids);
          fprintf(f_all_dat, "%e \t", length);
        }
      printf("\ncritical time = %e hrs\n", crit_time);
      printf("length = %e um \n", length);
      fscanf(fcrit_time, "%e", &crit_time);
    }  /* end of crit_len while loop */
    crit_stress = -1;
  }   /* end of option 4 */

 /*** ------------------------------------------------------------ ***/

  /*** get user input settings ***/
  if ( do_emsim == 0 ) {    /* only for PROMIS lines */
    printf("\nEnter starting value for the lines: ");
    scanf("%i", &first_subline);
    printf("\nEnter the ending value for the lines: ");
    scanf("%i", &last_subline);
  }
  else {   /* for EmSim lines */
    first_subline = 1;
    last_subline = 1;
  }

  if ( (do_emsim == 2) || ((do_emsim>=5) && (do_emsim<=6)) ) {
    /*** open file and write critical stress column headers ***/
    if( (do_emsim == 2) || (do_emsim==5) )
      strcpy(fn_crit_stress, "/vab/common/crit_stress.dat");
    if( do_emsim == 6 )
      strcpy(fn_crit_stress, "/vab/common/cmp_crit_stress.dat");
    if ( (fcritstress = fopen(fn_crit_stress, "r")) == NULL )
      printf("Can't open critical stress output file, you idiot\n");

    /*** put text label into f_all_dat file ***/
    fprintf(f_all_dat, "%s \t", label);
    if( do_emsim == 2 )
      printf("\n Line being tested:  %s\n", label);

    /*** read in first critical stress ***/
    fscanf(fcritstress, "%e", &crit_stress);
    printf("Critical stress for the line (MPa):  %.1f\n", crit_stress);
  }
  else if ( (do_emsim != 3) && (do_emsim != 4) ) {
    printf("\nEnter critical stress for the lines (MPa):  ");
    scanf("%f", &crit_stress);
  }

  /*** find ttf for each critical_stress until exit value (-1) ***/
  while ( crit_stress != -1 ) {
```

```
    /*** open fbatch file ***/
    if(do_emsim != 5) {
      sprintf(fn_ttf, "ttf%.2f_%s.%s", crit_stress, prefix, suffix);
      if ( (fttf = fopen(fn_ttf, "w")) == NULL )
      printf("Can't open ttf output file, you idiot\n");
    }

    /*** read in each file, write appropriate PROMIS file ***/
    for ( currsubline=first_subline; currsubline<=last_subline;
currsubline++) {
      /*** name and open input data files ***/
      if( do_emsim == 2 )
      printf("\nProcessing line %i... \n", currsubline);
      if ( do_emsim == 0 )
      sprintf(fn_input, "%i%s.%s", currsubline, prefix, suffix);
      else
      sprintf(fn_input, "%s.%s", prefix, suffix);
      if ( (finput = fopen(fn_input, "r")) == NULL ) {
      printf("Can't open the input file, you idiot\n");
      fprintf(f_all_dat, "\n");
      exit(1);
      }

      /*** find time_to_failure ***/
      ttf = -1;
      if ( do_emsim == 0 ) {
      while ( fscanf(finput, "%e %e", &time, &stress) != EOF )
        if ( stress >= crit_stress ) {
          ttf = time;
          while ( fscanf(finput, "%e %e", &time, &stress) != EOF )
            ;                              /* advance to end of file */
        }
      }
      else /* only do_emsim = 1,2,5,6 do this procedure */
      while ( fscanf(finput, "%e %e %e %e %e %e %e", &time, &dummy1,
&dummy2, &ten_stress, &dummy3, &cmp_stress, &dummy4/* , &dummy5,
&dummy6 */) != EOF ) {
        /*** finish reading line if necessary ***/
        if(maxstress_type == 1)
          fscanf(finput, "%e %e", &dummy5, &dummy6);

        /*** define stress based on type of failure ***/
        if( (do_emsim == 1) || (do_emsim == 2) )
          stress = (ten_stress > cmp_stress) ? (ten_stress) :
(cmp_stress);
        if( do_emsim == 5 )
          stress = ten_stress;
        if( do_emsim == 6 )
          stress = cmp_stress;
        /*** failed, so advance to the end of the file ***/
        if ( stress >= crit_stress ) {
          ttf = time;
```

```
          while ( fscanf(finput, "%e %e %e %e %e %e %e %e %e", &time,
 &dummy1, &dummy2, &ten_stress, &dummy3, &cmp_stress, &dummy4, &dummy5,
 &dummy6) != EOF )
            ;                               /* advance to end of file */
        }
      }
      /*** write to batch file ***/
      if(do_emsim != 5)
      fprintf(fttf, "%e\n", ttf);

      /*** write to main data file if appropriate ***/
      /* if( do_emsim == 2 )
      fprintf(f_all_dat, "%e \t", ttf); */

      /*** write acknowledgement of run ***/
      if ( ttf == -1 )
      printf("\n  Line SURVIVED ... -1 written to file\n\n\n");
      else
      printf("\n  Line FAILED at time=%.4f hrs\n\n\n", ttf);

      /*** close files ***/
      fclose(finput);

    }   /* end of for loop */

    /*** get next critical stress value ***/
    if ( (do_emsim == 2) || ((do_emsim>=5) && (do_emsim<=6))) {
      fscanf(fcritstress, "%e", &crit_stress);
      printf("Critical stress for the line (MPa):  %.1f\n",
crit_stress);
      if ( ttf != -1 )
      fprintf(f_all_dat, "%.4e \t", ttf);
    }
    else {
      printf("\nEnter critical stress for the lines (-1 to exit):  ");
      scanf("%f", &crit_stress);
    }
  }

  /*** end and close files ***/
  if( (do_emsim >= 2) && (do_emsim <= 6) ) {
    fprintf(f_all_dat, "\n");
    fclose(f_all_dat);
  }

  if( (do_emsim != 3) && (do_emsim != 4) && (do_emsim !=5) )
    fclose(fttf);
  printf("\n Data extraction complete for....%s\n", label);

}   /** end of MAIN **/
```

B.4. vgline Analysis Tool

In Section 5.4, an analytical means of constructing failure maps for simple structures of pure interconnects was discussed. A method of successive approximations was suggested for solving for the failure map line exhibiting void-growth-limited failures. A tool was written to perform this calculation, and determine this line's line lengths over a broad range of current densities. This program, vgline, may be run by simply executing the binary. The Makefile and source code for vgline is contained below.

```
::::::::::::::
Makefile
::::::::::::::
########################################################################
# Makefile for vgline
#
# Vaibhav Andleigh
# 24-Apr-98
#
# (c) copyright 1996, 1997, Massachusetts Institute of Technology
#                            All rights reserved.
########################################################################

BIN   = vgline

CC    = gcc
FLAGS = -O3
INCL  = -I/.
LIBS  = -lm

# for optimized runs, use gcc with flag -O3 only
# for debugging, use cc with flags -g -Wall which eliminates
optimization

OBJS  = vgline.o

OBJS_C      = vgline.o

OUTPUT      = error.dat

########################################################################

.c.o: ; $(CC) -c $(FLAGS) $(INCL) $*.c

########################################################################

go:   clean rm_dat emsim go_bg

gdb:  rm_dat
      gdb $(BIN)
```

```
vgline:     $(OBJS)
      $(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

clean:      rm_dat
      rm -f core *~ $(BIN) $(OUTPUT) $(OBJS_C)

rm_dat:
      rm -f *.dat
```

The file vgline.c below contains all of the code associated with vgline.

```
::::::::::::::
vgline.c
::::::::::::::
/***
This version of vgline calculates an approximate length vs
current density data regarding where we would expect to see
a current density exponent of n=1.

Command line:     vgline
               or vgline <option> <filename>

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/*** function prototypes ***/
void calclj(double *_j, double _L, double _LvFail);

/*** constants ***/
#define MAXLEN 1E5
#define FILELEN 75
#define TEXTLEN 90
#define NUMLEN 10

#define ERROR 1e3

/*** handy debugging functions ***/
#define PRINT_NUM(x) printf(#x " = %e\n\n", x)
#define PRINT_INT(x) printf(#x " = %i\n\n", x)
#define TEST(i)      printf("\n Made it here to %i ! \n\n", i)
#define average(x,y) ( (x)+(y) )/2             /* average of x and y */
#define square(x)    (x)*(x)                   /* equals x squared   */

/***=====================================================***/
int main(int argc, char *argv[])
```

```
{
  /*** variable declarations ***/
  char *choice;             /* text equivalent of do_emsim for
command*/

  double maxL,             /* max line length of any interconnect    */
         L,                /* length of the interconnect             */
         j,                /* critical current density for n=1 line  */
         MatProp=0.0229365,/* materials property constant             */
         FailPct,          /* failure % for DR / Ro                  */
         LvFail;           /* void length for failure of interconnect*/

  /*** File name variables ***/
  char output_fn[FILELEN];/* filename for all the L j data output   */

  /*** File pointer declarations ***/
  FILE *foutput;           /* output file for all the L j data       */

  /*** ---------------------------------------------------------- ***/

  /*** MAIN code starts ***/
  printf("\nProgram Begins...\n");

  /*** only query input if not given in command line ***/
  if( argc > 1 ) {
    choice = argv[1];
    sscanf(choice, "%lf", &maxL);
    choice = argv[2];
    sscanf(choice, "%lf", &FailPct);
  } else {
    printf("\nEnter max line length(maxL): ");
    scanf("%lf", &maxL);
    printf("\nEnter failure percent(DR/Ro): ");
    scanf("%lf", &FailPct);
  }

  /*** convert units ***/
  maxL *= 1e-6;
  FailPct *= 0.01;

  PRINT_NUM(maxL);
  PRINT_NUM(FailPct);

  /*** open output file ***/
  sprintf(output_fn, "Lj_neq1.dat");
  foutput = fopen(output_fn, "w");

#ifdef INIT_TEST
  printf("\nTest\n");
  calclj(&j, 2.5e-5, 5.734e-9);
  printf("jcrit = %e\n", j);
#endif

  /*** loop thru all the L values and calc L j data, record it ***/
```

```
  for(L=2.5e-5; L<=(maxL+1e-6); L += 5e-6) {
    /*** calc LvFail ***/
    LvFail = MatProp * FailPct * L;

    /*** calc j for our L ***/
    calclj(&j, L, LvFail);

    /*** write this data to file ***/
    printf("%.2e %.4e ", L, j*1e-4);
    fprintf(foutput, "%.2e %.4e ", L, j*1e-4);

    /*** calc LvFail, 2nd iteration ***/
    LvFail = MatProp * 2.0 * FailPct * L;
    calclj(&j, L, LvFail);
    printf("%.4e ", j*1e-4);
    fprintf(foutput, "%.4e ", j*1e-4);

    /*** calc LvFail, 3rd iteration ***/
    LvFail = MatProp * 4.0 * FailPct * L;
    calclj(&j, L, LvFail);
    printf("%.4e ", j*1e-4);
    fprintf(foutput, "%.4e ", j*1e-4);

    /*** calc LvFail, 4th iteration ***/
    LvFail = MatProp * 0.01 * L;
    calclj(&j, L, LvFail);
    printf("%.4e ", j*1e-4);
    fprintf(foutput, "%.4e ", j*1e-4);

    /*** calc LvFail, 5th iteration ***/
    LvFail = 5e-7;
    calclj(&j, L, LvFail);
    printf("%.4e ", j*1e-4);
    fprintf(foutput, "%.4e ", j*1e-4);

    /*** calc LvFail, 6th iteration ***/
    LvFail = 1e-6;
    calclj(&j, L, LvFail);
    printf("%.4e\n", j*1e-4);
    fprintf(foutput, "%.4e \n", j*1e-4);
    fflush(NULL);
  }

  printf("\nProgram Finished...\n");
  fclose(foutput);
  return(1);
}   /** end of MAIN **/

/*** ------------------------------------------------------------ ***/
/* Given info, calculates L for a given j value. */
void calclj(double *_j, double _L, double _LvFail)

{
  /*** variable declarations ***/
```

```
double pi=3.14159,           /* math pi              */
       b=5e10,              /* bulk modulus         */
       omega=1.66e-29,      /* atomic volume        */
       k=1.38e-23,          /* Boltzmann's constant */
       tempK=473.15,        /* temperature in K     */
       Da=1.57e-18,         /* atomic diffusivity   */
       kappa,               /* K=b*omega/kT         */

       e=1.6e-19,           /* electron charge      */
       rho=5e-8,            /* resisitivity         */
       z=4,                 /* effective charge     */
       G,                   /* G=e*rho*z/omega w/o j*/
     Lg,                  /* length at which meet */
       minstress=5e6,       /* min significant stres*/

       sigma_crit=4e8,      /* crit stress for nucl */
       tn,                  /* nucleation time      */
       tg,                  /* void growth time     */
       t_total,             /* total time= tn + tg  */

       jold,                /* value to store old j */
       jnew,                /* value to guess new j */
       diff,                /* difference btwn eqns */
       factor,              /* mult factor in succes*/
       temp,                /* temp holder for var  */
       fourKt,              /* fourKt = 4 * K * T   */
     breadth,             /* breadth = Lg^2/4kT   */
       term1,               /* 1st term in stress eqn */
       term2;               /* 2nd term in stress eqn */

/*** do successive approximations until solve j to within error ***/
jnew = 1.13e10;
while( ( (jnew-jold) > ERROR) || ((jold-jnew) > ERROR) ) {

  /*** guess initial value of j ***/
  jold = jnew;

  /*** calculate standard constants ***/
  kappa = Da * b * omega / (k * tempK);
  G = e * rho * z * jold / omega;

  /*** calculate void nucl. and growth times ***/
  tn = square(0.5 * sigma_crit / G) * pi / kappa;
  tg = _LvFail * k * tempK / (Da * G * omega);
  t_total = tn + tg;

  /*** use half of Lg as Lg-width if no void nucleation time
dominates ***/
  if(tn >= tg)
    Lg = 0.5 * _L;
  else
    Lg = _L;

  /*** calculate calculation constants ***/
```

```
        fourKt = 4 * kappa * t_total;
        breadth = Lg * Lg / fourKt;

        /*** calculate stress expression ***/
        term1 = sqrt(fourKt/pi) * exp(-1*breadth);
        term2 = Lg * erfc(sqrt(breadth));

        /*** solve for right hand side of eqn ***/
        temp = G * (term1 - term2);
        diff = minstress - temp;
        factor  = fabs(diff) / 1e10;

        /*** adjust jnew value appropriately ***/
        if( minstress < temp)
          jnew = jold * (1+factor);
        else
          jnew = jold * (1-factor);
  /* PRINT_NUM(jnew);   */
    }  /* end of WHILE loop */
    if(tn >= tg)
      printf("N ");
    *_j = jnew;

  /* PRINT_NUM(tn); */
  /* PRINT_NUM(tg); */


  }  /* end of function calclj */
  /***=======================================================***/
```

B.5. calc_res Analysis Tool

While conducting experiments, the voltage across the interconnect was measured as a function of time for interconnects of differing test conditions and geometries. Using Ohm's Law, the resistance as a function of time may then be calculated assuming a constant current. Because the data acquisition rate and data format varied from one test to the next, a tool was developed for analyzing the raw data to determine the resistance vs. time profile. This tool was named calc_res, and may be run by executing the binary without any arguments. The Makefile and source code for this tool are included below:

```
##########################################################################
# Makefile for Calc_Res
#
# Vaibhav Andleigh
# 17-Jun-2000
#
```

```
# (c) copyright 2000        Massachusetts Institute of Technology
#                           All rights reserved.
########################################################################

BIN     = calc_res

CC      = gcc
FLAGS   = -O3 -g
INCL    = -I/.
LIBS    = -lm

OBJS    = calc_res.o

.c.o:   ; $(CC) -c $(FLAGS) $(INCL) $*.c

go:     clean calc_res

emsim:          clean calc_res

calc_res:       $(OBJS)
        $(CC) $(FLAGS) $(INCL) $(OBJS) -o $(BIN) $(LIBS)

run:
        $(BIN)

clean:
        rm -f calc_res.o core calc_res.c~ Makefile~ $(BIN)
```

All of the code for calc_res is contained in one file, calc_res.c, shown below.

```
::::::::::::::
calc_res.c
::::::::::::::
/***
This version of calc_res reads in the experimentally-created
data files from Labview and EM Test Setup (Hades) and
transforms the data to two file formats.  The first indexes
the time axis and prints the voltage data.  The second
uses the current (in Amps) input to calculate the resistance
of the line versus time.  The file extensions associated
with these files are *.vlt and *.res respectively.  For
output condensed into one file, the file extension
is *.exp .

Command line:     calc_res
               or find_ttf <input filename> <current>

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>
#include <math.h>

/*** function prototypes ***/

/*** constants ***/
#define FILELEN 75
#define TEXTLEN 90
#define NUMLEN 10
#define INPLEN 20
#define SMOOTH 10

/*** switches ***/
#define CONDENSED

/***=====================================================***/
int main(int argc, char *argv[])
{ /* MAIN */
  /*** variable declarations ***/
  int index=1,            /* index variable for counting time (sec) */
      num_pts;            /* # of data points for data smoothing    */

  double voltage,         /* voltage in Volts in input file         */
         current,         /* current in Amps applied on the line    */
         resistance;      /* resistance in ohms calculated as V/I   */

  char prefix[FILELEN];   /* file name w/o extension of input file  */
  char suffix[3];         /* holds suffix for input stress file     */
  char fn_input[FILELEN]; /* filename of voltage input data         */
  char fn_volt[FILELEN];  /* filname for voltage output data        */
  char fn_res[FILELEN];   /* filename for resistance output data    */
  char fn_both[FILELEN];  /* filename for voltage and resistance dat*/

  char *dtemp;            /* dummy text for reading command line    */
  char *d1;               /* dummy text variable for reading input  */
  char *d2;               /* dummy text variable for reading input  */
  char *d3;               /* dummy text variable for reading input  */

  double dt[SMOOTH];      /* dummy array for holding averaging vals */

  /*** File pointer declarations ***/
  FILE *finput;           /* voltage input data file                */
  FILE *fvolt;            /* voltage output data file               */
  FILE *fres;             /* resistance output data file            */
  FILE *fboth;            /* voltage and resistance output data file*/
  /*** MAIN code starts ***/

  /*** program begin greeting ***/
  printf("\n\nPROGRAM STARTING \n");

  /*** only query input if not given in command line ***/
  if( argc > 1 ) {
    strcpy(prefix, argv[1]);
```

```
    strcpy(suffix, "txt");
    d1 = argv[2];
    sscanf(d1, "%lf", &current);
  }
  else {
    /*** determine if from Emsim data or from Emigdata ***/
    printf("\nEnter the input filename (w/o extension: ");
    scanf("%s", &prefix);
    strcpy(suffix, "txt");
    printf("\nEnter the current (in Amps):  ");
    scanf("%lf", &current);
    printf("\n");
  } /* end of query input */

  /*** open input file ***/
  sprintf(fn_input, "%s.%s", prefix, suffix);
  if ( (finput = fopen(fn_input, "r")) == NULL )
    printf("Can't open voltage INPUT file\n");

  /*** open file for voltage and resistance output file ***/
  sprintf(fn_volt, "%s.vlt", prefix);
  sprintf(fn_res, "%s.res", prefix);
  sprintf(fn_both, "%s.exp", prefix);
#ifndef CONDENSED
  if ( (fvolt = fopen(fn_volt, "a")) == NULL )
      printf("Can't open VOLTAGE output file\n");
  if ( (fres = fopen(fn_res, "a")) == NULL )
      printf("Can't open RESISTANCE output file\n");
#endif
  if ( (fboth = fopen(fn_both, "a")) == NULL )
      printf("Can't open BOTH output file\n");

  /*** initialize smoothing ***/


  /*** while loop for reading input data ***/
  printf("Performing data retrieval and write operations...\n\n");
  while( fscanf(finput, "%s %s %s %lf", &d1, &d2, &d3, &voltage) != EOF
) {
    /*** write voltage and resistance to appropriate files ***/
    resistance = voltage / current;
#ifndef CONDENSED
    fprintf(fvolt, "%i \t %e\n", index, voltage);
    fprintf(fres, "%i \t %e\n", index, resistance);
#endif
    fprintf(fboth, "%i \t %e \t %e\n", index, voltage, resistance);
    index++;
    printf(".");
  }

  /*** close all files ***/
  fclose(finput);
#ifndef CONDENSED
  fclose(fvolt);
```

```
  fclose(fres);
#endif
  fclose(fboth);

  /*** program exit greeting ***/
  printf("\n\nPROGRAM FINISHED\n\n");
  fflush(NULL);

}   /** end of MAIN **/
```

B.6. MIT/EmSim Movie-Making Utility

One of the most important capabilities of MIT/EmSim is to track stress evolution along the interconnect through time. This space-dependent and time-dependent information may be conveniently analyzed by the construction of MIT/EmSim stress evolution movies. For convenience, a utility was written for converting textual stress information from all the recorded times into a graphical, plotted stress profile gif movie. This utility (for Linux systems) requires that the program whirlgif be installed, and should be run from the bash shell. In addition, to simplify the naming conventions, the movie switch in MIT/EmSim must be enabled to ensure that the stress profiles from each time are sequentially numbered. For brevity, this source is not included here, although is available upon request.

B.7. YearSecondCalc Utility

Lifetimes in interconnects can range anywhere from hours to decades, or even longer as in the case of immortal lines. When failure times are reported (by MIT/EmSim for instance), they are typically given in hours. For convenience, a java-based utility was written to convert hours into more meaningful units, and vice-versa. An image of the GUI of this utility, YearSecondCalc, is shown below.

The source code to YearSecondCalc is contained below. This code was written to compile to JDK 1.2.2. The first file is the automatically generated html file for operating this program's Java class file in a web browser.

```
:::::::::::::
autogen_YearSecond.html
::::::::::::::
<HTML>
<HEAD>
<TITLE>Autogenerated HTML</TITLE>
</HEAD>
<BODY>
<APPLET CODE="YearSecondCalc.class" WIDTH=426 HEIGHT=288></APPLET>
</BODY>
</HTML>
```

The file below, YearSecondCalc.java, contains the actual Java code for YearSecondCalc.

```
::::::::::::::
YearSecondCalc.java
::::::::::::::
/*
    A basic extension of the java.applet.Applet class
 */

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class YearSecondCalc extends Applet {
      void calcSeconds_MouseClick(java.awt.event.MouseEvent event) {
            // read in values of textboxes
            numYear = years.getText();
            numDay = days.getText();
            numHour = hours.getText();
            numMin = minutes.getText();
            numSec = seconds.getText();
            // increment numSeconds for each value
            int numSeconds = 0;
            numSeconds += Integer.parseInt(numSec);
            numSeconds += Integer.parseInt(numMin) * 60;
            numSeconds += Integer.parseInt(numHour) * 60*60;
            numSeconds += Integer.parseInt(numDay) * 60*60*24;
            numSeconds += Integer.parseInt(numYear)*60*60*24*365;
            totalSeconds.setText(String.valueOf((int)numSeconds));
      }

      void calcYears_MouseClick(java.awt.event.MouseEvent event) {
            // calculate years, then days, etc.
            int numYear = 0;
            int numDay = 0;
            int numHour = 0;
            int numMin = 0;
            int numSec = 0;
            int workingSeconds =
      Integer.parseInt(totalSeconds.getText());
            while( workingSeconds >= (365*24*60*60) ) {
```

```
            numYear++;
            workingSeconds -= 365*24*60*60;
        }
        while( workingSeconds >= (24*60*60) ) {
            numDay++;
            workingSeconds -= 24*60*60;
        }
        while( workingSeconds >= (60*60) ) {
            numHour++;
            workingSeconds -= 60*60;
        }
        while( workingSeconds >= (60) ) {
            numMin++;
            workingSeconds -= 60;
        }
        numSec = workingSeconds;
        years.setText(String.valueOf((int)numYear));
        days.setText(String.valueOf((int)numDay));
        hours.setText(String.valueOf((int)numHour));
        minutes.setText(String.valueOf((int)numMin));
        seconds.setText(String.valueOf((int)numSec));
  }


String numSec, numMin, numHour, numDay, numYear, numSeconds;

  void years_EnterHit(java.awt.event.ActionEvent event) {
        // read in value
        numYear = years.getText();
  }


  void days_EnterHit(java.awt.event.ActionEvent event) {
        // read in value
        numDay = days.getText();
  }


  void hours_EnterHit(java.awt.event.ActionEvent event) {
        // read in value
        numHour = hours.getText();
  }


  void minutes_EnterHit(java.awt.event.ActionEvent event) {
        // read in value
        numMin = minutes.getText();
  }


  void seconds_EnterHit(java.awt.event.ActionEvent event) {
        // read in value
        numSec = seconds.getText();
  }
```

```
	public void init() {
		// Call parents init method.
		super.init();
	}

	public void addNotify() {
			// Take out this line if you don't use
symantec.itools.net.RelativeURL
		//
symantec.itools.lang.Context.setDocumentBase(getDocumentBase());

		// Call parents addNotify method.
		super.addNotify();

	// This code is automatically generated by Visual Cafe when you add
	// components to the visual environment. It instantiates and
	// initializes the components. To modify the code, only use code
	// syntax that matches
	// what Visual Cafe can generate, or Visual Cafe may be unable to
	// back parse your Java file into its visual environment.
			//{{INIT_CONTROLS
			setLayout(null);
			resize(426,288);
			setBackground(new Color(65535));
			totalSeconds = new java.awt.TextField();
			totalSeconds.setText("0");
			totalSeconds.reshape(324,120,84,24);
			totalSeconds.setBackground(new Color(16777215));
			add(totalSeconds);
			label6 = new java.awt.Label("Days");
			label6.reshape(48,84,57,24);
			add(label6);
			label5 = new java.awt.Label("Hours");
			label5.reshape(48,120,57,24);
			add(label5);
			label4 = new java.awt.Label("Minutes");
			label4.reshape(48,156,57,24);
			add(label4);
			label3 = new java.awt.Label("Seconds");
			label3.reshape(48,192,57,24);
			add(label3);
			label2 = new java.awt.Label("Seconds");
			label2.reshape(252,120,57,24);
			add(label2);
			label1 = new java.awt.Label("Years");
			label1.reshape(48,48,57,24);
			add(label1);
			calcSeconds = new java.awt.Button("---->");
			calcSeconds.reshape(228,240,51,26);
			calcSeconds.setFont(new Font("Dialog", Font.BOLD, 16));
			calcSeconds.setBackground(new Color(16756655));
```

```
        add(calcSeconds);
        calcYears = new java.awt.Button("<----");
        calcYears.reshape(156,240,51,26);
        calcYears.setFont(new Font("Dialog", Font.BOLD, 16));
        calcYears.setBackground(new Color(16756655));
        add(calcYears);
        days = new java.awt.TextField();
        days.setText("0");
        days.reshape(120,84,44,24);
        days.setBackground(new Color(16777215));
        add(days);
        hours = new java.awt.TextField();
        hours.setText("0");
        hours.reshape(120,120,44,24);
        hours.setBackground(new Color(16777215));
        add(hours);
        minutes = new java.awt.TextField();
        minutes.setText("0");
        minutes.reshape(120,156,44,24);
        minutes.setBackground(new Color(16777215));
        add(minutes);
        seconds = new java.awt.TextField();
        seconds.setText("0");
        seconds.reshape(120,192,44,24);
        seconds.setBackground(new Color(16777215));
        add(seconds);
        years = new java.awt.TextField();
        years.setText("0");
        years.reshape(120,48,44,24);
        years.setBackground(new Color(16777215));
        add(years);
        title = new java.awt.Button("VabWare Creations");
        title.reshape(96,12,241,26);
        title.setFont(new Font("Dialog", Font.BOLD, 16));
        title.setForeground(new Color(16777215));
        title.setBackground(new Color(16711808));
        add(title);
        //}}

        //{{REGISTER_LISTENERS
        //Action lAction = new Action();
        //seconds.addActionListener(lAction);
        //minutes.addActionListener(lAction);
        //hours.addActionListener(lAction);
        //days.addActionListener(lAction);
        //years.addActionListener(lAction);
        Mouse lMouse = new Mouse();
        calcYears.addMouseListener(lMouse);
        calcSeconds.addMouseListener(lMouse);
        //}}
}

//{{DECLARE_CONTROLS
java.awt.TextField totalSeconds;
```

```
    java.awt.Label label6;
    java.awt.Label label5;
    java.awt.Label label4;
    java.awt.Label label3;
    java.awt.Label label2;
    java.awt.Label label1;
    java.awt.Button calcSeconds;
    java.awt.Button calcYears;
    java.awt.TextField days;
    java.awt.TextField hours;
    java.awt.TextField minutes;
    java.awt.TextField seconds;
    java.awt.TextField years;
    java.awt.Button title;
    //}}

    /*class Action implements java.awt.event.listener {
          public void actionPerformed(java.awt.event.ActionEvent
event) {
                Object object = event.getSource();
                if (object == seconds)
                      seconds_EnterHit(event);
                else if (object == minutes)
                      minutes_EnterHit(event);
                else if (object == hours)
                      hours_EnterHit(event);
                else if (object == days)
                      days_EnterHit(event);
                else if (object == years)
                      years_EnterHit(event);
          }
    }*/

    class Mouse implements java.awt.event.MouseListener {
          public void mouseReleased(java.awt.event.MouseEvent event){
          }

          public void mousePressed(java.awt.event.MouseEvent event) {
          }

          public void mouseEntered(java.awt.event.MouseEvent event) {
          }

          public void mouseExited(java.awt.event.MouseEvent event) {
          }

          public void mouseClicked(java.awt.event.MouseEvent event) {
                Object object = event.getSource();
                if (object == calcYears)
                      calcYears_MouseClick(event);
                else if (object == calcSeconds)
                      calcSeconds_MouseClick(event);
          }
    }
```

```
}
```

B.8. Mask-Layout Utility

A utility was written to aid in the tri-level mask layout of the interconnect structures for the experiment described in Chapter 6. Because of the four-fold symmetry and size scaling of the test structures for varying linewidths and line lengths, it was deemed simpler to create a mask layout utility in lieu of using a GUI-based mask-layout tool such as KIC. The Makefile and supporting files for this utility, mask, are included below:

```
########################################################################
# Makefile for mask
#
# Vaibhav Andleigh
# 24-Apr-98
#
# (c) copyright 1999, 2000, Massachusetts Institute of Technology
#                           All rights reserved.
########################################################################

BIN    = mask_new

CC     = gcc
FLAGS  = -O3
INCL   = -I/.
LIBS   = -lm

# for optimized runs, use gcc with flag -O3 only
# for debugging, use cc with flags -g -Wall which eliminates
optimization

OBJS   = mask.o \
         mask_bond.o \
         mask_probe.o \
         mask_short.o \
         mask_alphnum.o \
         mask_align.o

OBJS_C = mask.o \
         mask_bond.o \
         mask_probe.o \
         mask_short.o \
         mask_alphnum.o \
         mask_align.o
```

```
OUTPUT      = error.dat

########################################################################

.c.o: ; $(CC) -c $(FLAGS) $(INCL) $*.c

########################################################################

go:   clean rm_dat mask go_bg

gdb:  rm_dat
      gdb $(BIN)

run:  rm_dat
      $(BIN)

emsim:      $(OBJS)
      $(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

mask: $(OBJS)
      $(CC) $(FLAGS) $(INCL) $(OBJS_C) -o $(BIN) $(LIBS)

clean:      rm_dat
      rm -f core *~ $(BIN) $(OUTPUT) $(OBJS_C)

rm_dat:
      rm -f *.dat
```

The following file, mask.c, is the main program file for the mask-layout utility.

```
/***
This version of mask reads ...

Command line:     mask
               or mask <option> <filename>

Variable names in functions are distinguished with a "_"
in front of them
***/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_bond.h"
#include "mask_probe.h"
#include "mask_short.h"
```

```
#include "mask_alphnum.h"
#include "mask_align.h"

/***====================================================***/
int main(int argc, char *argv[])
{
  /*** variable declarations ***/

  double pad_spacing,       /* spacing between the pads              */
         pad_len,           /* side length of square pad             */
         line_len,          /* length of test structure line         */
         line_width,        /* width of test structure line          */
         strand_width,      /* width of strands connecting line topad*/
         link_len_pi,       /* length of strand connecting I pad     */
         link_len_pv,       /* length of strand connecting V pad     */
         strand_len_li,     /* length of long strand to I pad        */
         strand_len_lv,     /* length of long strand to V pad        */
         link_pi_offset,    /* offset from edge for link to I pad    */
         lv_overhang,       /* overhang of long strand to V pad      */
         run_len,           /* length of runner to test struc andbond*/
         pr_pad_len,        /* length of pad in probe test structure */
         pr_pad_spacing,    /* spacing between pad in probe test truc*/
         pr_link_len,       /* length of link connect probe pad 2line*/
         pr_str_overhang,   /* overhang of probe strand beyond links */
         pr_strand_len,     /* length of entire strand in probe truct*/
         min_feat_size,     /* minimum feature size of the ask(scale)*/
         bwidth;            /* width of bond pad test structure      */

  Point offset_v,           /* x and y offset for each vert structure*/
        offset_h,           /* x and y offset for each horiz struct  */
        obj_loc_v,          /* location of vert substructure location*/
        obj_loc_h,          /* location of horiz substructurelocation*/
        probe_loc_h,        /* location of horiz probe structure     */
        probe_loc_v,        /* location of vertical probe structure  */
        alphnum_loc_h,      /* location of alphanumeric character str*/
        alphnum_loc_v,      /* location of alphanumeric character str*/
        align_loc,          /* location of alignment structures      */
        align_text_loc,     /* location of alignment text structures */
        text_loc,           /* location of final text segments       */
        short_loc,          /* location of very short test structures*/
        short_end_loc,      /* end location of very short structures */
        short_text_loc;     /* location of alphnumeric text labels   */
  int num_structs=0,        /* number of structures to place onto ask*/
      orient,               /* orientation of structures             */
      set_num,              /* set id number of test structures      */
      bwidth_num=1;         /* width bin # of bond pad test structure*/

  /*** Filename and File pointer declarations ***/
  char out_fn1[FILELEN],    /* file name with extension of mask1 file*/
       out_fn2[FILELEN],    /* file name with extension of mask2 file*/
       out_fn3[FILELEN];    /* file name with extension of mask3 file*/

  FILE *fp_mask1,           /* output file for interconnect structs  */
       *fp_mask2,           /* output file for line end-stops        */
```

```
      *fp_mask3;           /* output file for pad definition        */

  /*** ------------------------------------------------------------ ***/

  /*** MAIN code starts ***/

  printf("\nProgram Begins...\n");

  /*** define basic dimensions for bond pad structure ***/
  min_feat_size  = 1.0;
  pad_len        = 100.0 * min_feat_size;
  pad_spacing    = 75.0  * min_feat_size;
  strand_width   = 10.0  * min_feat_size;
  link_len_pi    = 20.0  * min_feat_size;
  link_len_pv    = 10.0  * min_feat_size;
  link_pi_offset = 20.0  * min_feat_size;
  lv_overhang    = 20.0  * min_feat_size;
  run_len        = 20.0  * min_feat_size;
  strand_len_li  = (pad_len + link_len_pv - link_pi_offset) *
min_feat_size;
  strand_len_lv  = (pad_len + pad_spacing - link_len_pi + lv_overhang);
  strand_len_lv  *= min_feat_size;

  /*** define additional dimensions for probe pad structure ***/
  pr_pad_len       = 75.0 * min_feat_size;
  pr_pad_spacing   = 10.0 * min_feat_size;
  pr_link_len      = 10.0 * min_feat_size;
  pr_str_overhang  = 20.0 * min_feat_size;
  pr_strand_len    = 2*pr_str_overhang + 2*strand_width +
pr_pad_spacing;
  pr_strand_len   *= min_feat_size;

  /*** define initial offsets ***/
  offset_v.x1 = 0.0;
  offset_v.y1 = pad_len + link_len_pv + strand_width;
  offset_h.x1 = pad_len + link_len_pv + strand_width + run_len +
100.0;;
  offset_h.y1 = pad_len + link_len_pv + strand_width;

  /*** define thefilename, and open file ***/
  sprintf(out_fn1, "mask1.kic");
  if ( (fp_mask1 = fopen(out_fn1, "w")) == NULL)
    printf("\nCould not open the  %s input file!!!\n", out_fn1);
  sprintf(out_fn2, "mask2.kic");
  if ( (fp_mask2 = fopen(out_fn2, "w")) == NULL)
    printf("\nCould not open the  %s input file!!!\n", out_fn2);
  sprintf(out_fn3, "mask3.kic");
  if ( (fp_mask3 = fopen(out_fn3, "w")) == NULL)
    printf("\nCould not open the  %s input file!!!\n", out_fn3);
  fflush(NULL);

  /*** print initial file header data ***/
  fprintf(fp_mask1, "(Symbol %s);\n", out_fn1);
  fprintf(fp_mask1, "9 %s;\nDS 0 1 1;\n", out_fn1);
```

```
  fprintf(fp_mask1, "L GPE;\n");
  fprintf(fp_mask2, "L RES;\n");
  fprintf(fp_mask3, "L CAP;\n");

  /*** loop through all the line lengths layer 1 ***/
  line_width = 4;
  line_len = 100.0;
  for(set_num = 1; set_num <= 14; set_num++) {

    /*** define VERTICAL BOND pad structure ***/
    define_bond_struct(offset_v, fp_mask1, fp_mask2, fp_mask3, VERT,
pad_len, pad_spacing, strand_width, link_len_pi, link_len_pv,
strand_len_li, strand_len_lv, link_pi_offset, lv_overhang);

    /*** define HORIZONTAL BOND pad structure ***/
    define_bond_struct(offset_h, fp_mask1,  fp_mask2, fp_mask3, HORIZ,
pad_len, pad_spacing, strand_width, link_len_pi, link_len_pv,
strand_len_li, strand_len_lv, link_pi_offset, lv_overhang);

    /*** define object location for test structure lines ***/
    obj_loc_v.x1 = offset_v.x1 + pad_len + link_len_pv + strand_width;
    obj_loc_v.y1 = offset_v.y1;
    obj_loc_h.x1 = offset_h.x1;
    obj_loc_h.y1 = offset_h.y1;
    probe_loc_h.x1 = obj_loc_v.x1 + 20.0;
    probe_loc_h.y1 = obj_loc_v.y1 + strand_width + 20.0;
    alphnum_loc_h.x1 = offset_v.x1 + 120.0;
    alphnum_loc_h.y1 = offset_v.y1 - 60.0;
    alphnum_loc_v.x1 = offset_h.x1 - 160.0;
    alphnum_loc_v.y1 = offset_h.y1 + 100.0 + 6 * 50.0;

    /*** define connecting runner ***/
    define_line(obj_loc_v, fp_mask1, run_len, strand_width);
    obj_loc_v.x1 += run_len;

    /*** define LINE STRUCTURE between bond pads ***/
    bwidth = get_linewidth(bwidth_num);
    define_centered_line(obj_loc_v, fp_mask1, HORIZ, line_len, bwidth);
    define_end_stops(obj_loc_v, fp_mask2, HORIZ, line_len);
    obj_loc_v.x1 += line_len;
    num_structs++;

    /*** define backwards L shape using two offset values ***/
    define_L_shape(obj_loc_v, obj_loc_h, fp_mask1, strand_width);

    /*** skip first and last set of VERTICAL PROBE test structures ***/
    if( (set_num != 1) && (set_num != 14) ) {
      /*** define vertical repeating probe test structure (free) ***/
      probe_loc_v.x1 = obj_loc_h.x1 - 105.0;
      probe_loc_v.y1 = obj_loc_v.y1 + strand_width - 30.0;
      probe_loc_v.y1 = pad_len + link_len_pv + 3*strand_width;
      repeat_probe_struct(probe_loc_v, obj_loc_v, fp_mask1, fp_mask2,
fp_mask3, VERT, line_len, set_num, pr_pad_len, pr_pad_spacing,
strand_width, pr_strand_len, pr_str_overhang, pr_link_len, JOINED);
```

```
      /*** redefine probe coordinates for 2nd vertical probe set ***/
      probe_loc_v.x1 += strand_len_lv - 4*strand_width;
      probe_loc_v.y1 += 20.0;

      /*** def. 2nd vertical probe teststructures(enveloped, large)***/
      repeat_probe_struct(probe_loc_v, obj_loc_v, fp_mask1, fp_mask2,
fp_mask3, VERT, line_len, set_num, pad_len, pr_pad_spacing,
strand_width, pr_strand_len, pr_str_overhang, pr_link_len, ISOLATED);
    }

    /*** skip first set of HORIZONTAL PROBE test structures ***/
    if( set_num != 1 ) {
      /*** def horiz repeating probetest structure(enveloped,large)***/
      repeat_probe_struct(probe_loc_h, obj_loc_h, fp_mask1, fp_mask2,
fp_mask3, HORIZ, line_len, set_num, pad_len, pr_pad_spacing,
strand_width, pr_strand_len, pr_str_overhang, pr_link_len, ISOLATED);

      /*** redef probe coordinates for second horizontal probe set ***/
      probe_loc_h.x1 = offset_v.x1+ pad_len + link_len_pv +
strand_width;
      probe_loc_h.y1 = obj_loc_v.y1 + strand_len_lv + 10.0;
      /* probe_loc_h.y1 += 3 * strand_width; */

      /*** define second horizontal probe test structures (free)***/
      repeat_probe_struct(probe_loc_h, obj_loc_h, fp_mask1, fp_mask2,
fp_mask3, HORIZ, line_len, set_num, pr_pad_len, pr_pad_spacing,
strand_width, pr_strand_len, pr_str_overhang, pr_link_len, JOINED);
    }

    /*** skip first set of TEXT LABELING structures ***/
    if( set_num > 2 ) {
      /*** skip last vert. test label structure(25 and 50 instead) ***/
      if( set_num != 14 )
      /*** write L and numbers, advance accordingly for VERTICAL ***/
      parse_length(&alphnum_loc_v, fp_mask3, VERT, line_len);

      /*** write L and numbers, advance accordingly for HORIZONTAL ***/
      parse_length(&alphnum_loc_h, fp_mask3, HORIZ, line_len);
      parse_width(&alphnum_loc_h, fp_mask3, HORIZ, bwidth_num);
    }

    /*** adjust object location for back half route ***/
    obj_loc_h.x1 += strand_len_lv - strand_width;
    obj_loc_v.y1 += strand_len_lv - strand_width;

    /*** define backwards L shape to go back to vert bond structure***/
    define_L_shape(obj_loc_v, obj_loc_h, fp_mask1, strand_width);

    /*** define second LINE STRUCTURE ***/
    obj_loc_v.x1 -= line_len;
    bwidth = get_linewidth(bwidth_num);
    define_centered_line(obj_loc_v, fp_mask1, HORIZ, line_len, bwidth);
    define_end_stops(obj_loc_v, fp_mask2, HORIZ, line_len);
```

```
    bwidth_num++;
    num_structs++;

    /*** define connecting runner ***/
    obj_loc_v.x1 -= run_len;
    define_line(obj_loc_v, fp_mask1, run_len, strand_width);

    /*** increment vertical bond structure y offsets ***/
    offset_h.x1 += 2 * (pad_spacing + pad_len);
    offset_v.y1 += 2 * (pad_spacing + pad_len);

    /*** increment line lengths ***/
    line_len += 50.0;

  }

  /*** print # of structures produced ***/
  printf("\nNumber of structures = %d\n", num_structs);

  /*** draw large,small alignment and developing structs on mask 1 ***/
  align_loc.x1 = 4965.0;
  align_loc.y1 = 5000.0;
  align_text_loc.x1 = 4740.0;
  align_text_loc.y1 = 4900.0;
  draw_red_cross(align_loc, fp_mask1, 80, 1);
  parse_align_num(align_text_loc, fp_mask1, 2, 6);
  align_text_loc.x1 = 5030.0;
  parse_align_num(align_text_loc, fp_mask1, 3, 4);
  align_loc.x1 -= 300.0;
  draw_red_cross(align_loc, fp_mask1, 25, 1);
  align_loc.y1 -= 100.0;
  draw_diag_squares(align_loc, fp_mask1, 30, 1);
  align_loc.x1 -= 30.0;
  align_loc.y1 += 200.0;
  draw_bar_code(align_loc, fp_mask1, 1);

  /*** draw window,small alignment and developing struct on mask 2 ***/
  align_loc.x1 = 4715.0;
  align_loc.y1 = 4865.0;
  align_loc.x2 = 5215.0;
  align_loc.y2 = 5135.0;
  write_led_bar(fp_mask2, align_loc, 2);
  align_loc.x1 = 4965.0;
  align_loc.y1 = 5000.0;
  align_loc.x1 -= 300.0;
  draw_bigger_cross(align_loc, fp_mask2, 27, 2);
  align_loc.x1 += 600.0;
  draw_red_cross(align_loc, fp_mask1, 25, 1);
  align_loc.y1 -= 100.0;
  draw_diag_squares(align_loc, fp_mask2, 30, 2);
  align_loc.x1 -= 30.0;
  align_loc.y1 += 200.0;
  draw_bar_code(align_loc, fp_mask2, 2);
```

```
/*** draw window,small alignment and developing struct on mask 3 ***/
align_loc.x1 = 4715.0;
align_loc.y1 = 4865.0;
align_loc.x2 = 5215.0;
align_loc.y2 = 5135.0;
write_led_bar(fp_mask3, align_loc, 3);
align_loc.x1 = 4965.0;
align_loc.y1 = 5000.0;
align_loc.x1 -= 300.0;
align_loc.x1 += 600.0;
draw_bigger_cross(align_loc, fp_mask3, 27, 3);
align_loc.x1 += 150.0;
draw_red_cross(align_loc, fp_mask1, 25, 1);
align_loc.y1 -= 100.0;
draw_diag_squares(align_loc, fp_mask3, 30, 3);
align_loc.x1 -= 30.0;
align_loc.y1 += 200.0;
draw_bar_code(align_loc, fp_mask3, 3);
align_loc.x1 = 4515.0;
align_loc.y1 = 5000.0;
draw_red_cross(align_loc, fp_mask3, 25, 3);

/*** draw developing structures on the four corners of each mask ***/
align_loc.x1 = 200.0;
align_loc.y1 = 200.0;
draw_diag_squares(align_loc, fp_mask1, 30, 1);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask2, 30, 2);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask3, 30, 3);
align_loc.x1 = 2 * HALF_DIE_LEN/100.0 - align_loc.x1 - 170.0;
align_loc.y1 = 2 * HALF_DIE_LEN/100.0 - align_loc.y1 - 100.0;
draw_diag_squares(align_loc, fp_mask1, 30, 1);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask2, 30, 2);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask3, 30, 3);
align_loc.x1 = 200.0;
draw_diag_squares(align_loc, fp_mask1, 30, 1);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask2, 30, 2);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask3, 30, 3);
align_loc.x1 = 2 * HALF_DIE_LEN/100.0 - align_loc.x1 - 170.0;
align_loc.y1 = 200.0;
draw_diag_squares(align_loc, fp_mask1, 30, 1);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask2, 30, 2);
align_loc.x1 += 70.0;
draw_diag_squares(align_loc, fp_mask3, 30, 3);

/*** draw vertical probe VERY SHORT 50 um test structures ***/
short_loc.x1 = 4820.0;
short_loc.y1 = 240.0;
```

```
  short_end_loc.x1 = short_loc.x1;
  short_end_loc.y1 = 4900.0;
  line_len = 50.0;
  repeat_short_struct(short_loc, short_end_loc, fp_mask1, fp_mask2,
fp_mask3, VERT, line_len, set_num, pad_len, pr_pad_spacing,
strand_width, pr_link_len, ISOLATED);
  short_text_loc.x1 = 4820.0;
  short_text_loc.y1 = 160.0;
  parse_length(&short_text_loc, fp_mask3, HORIZ, line_len);

  /*** draw vertical probe VERY SHORT 25 um test structures ***/
  line_len = 25.0;
  short_loc.x1 = 4650.0;
  short_end_loc.x1 = short_loc.x1;
  short_end_loc.y1 = 4600.0;
  repeat_short_struct(short_loc, short_end_loc, fp_mask1, fp_mask2,
fp_mask3, VERT, line_len, set_num, pad_len, pr_pad_spacing,
strand_width, pr_link_len, ISOLATED);
  short_text_loc.x1 = 4650.0;
  short_text_loc.y1 = 160.0;
  parse_length(&short_text_loc, fp_mask3, HORIZ, line_len);

  /*** write text amusements ***/
  text_loc.x1 = 1500.0;
  text_loc.y1 = 4980.0;
  write_emsim(text_loc, fp_mask3);
  text_loc.x1 = 3000.0;
  write_andleigh(text_loc, fp_mask3);
  text_loc.x1 = 9000.0;
  write_bites(text_loc, fp_mask3);
  text_loc.x1 = 7000.0;
  write_microsoft(text_loc, fp_mask3);

  /*** print file footer data ***/
  fprintf(fp_mask3, "DF; \nE");

  /*** close file ***/
  fclose(fp_mask1);
  fclose(fp_mask2);
  fclose(fp_mask3);

  /*** closing statement ***/
  printf("\nProgram Finished...\n\n");

  /*** exit program successfully ***/
  return(0);

}   /** end of MAIN **/


/***   mask_align.c support file   ***/

/*** libraries ***/
#include <math.h>
```

```
#include <stdio.h>
#include <stdlib.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_align.h"
#include "mask_alphnum.h"

/*** function listing **********

draw_red_cross()

draw_bigger_cross()

draw_diag_squares()

draw_square()

draw_bar_code()

draw_rectangle()

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

/***=============================================================***/
void draw_red_cross(Point loc, FILE *fp_out, double width, int _level)
/***
Takes input location (CENTER!) and draws 5 squares in the
shape of the red cross symbol for the alignment structure.
***/
{
  Point north,                       /* position of square top     */
        east,                        /* position of square right   */
        south,                       /* position of square bottom  */
        west,                        /* position of square left    */
        center;                      /* position of square center  */

  /*** assign coordinates to all the squares ***/
  center.x1 = loc.x1 - width/2.0;
  center.y1 = loc.y1 - width/2.0;
  north.x1 = center.x1;
  north.y1 = center.y1 - width;
  south.x1 = center.x1;
  south.y1 = center.y1 + width;
  east.x1 = center.x1 + width;
  east.y1 = center.y1;
  west.x1 = center.x1 - width;
  west.y1 = center.y1;

  /*** draw the 5 squares to form the red cross symbol ***/
  draw_square(center, fp_out, width, _level);
  draw_square(north, fp_out, width, _level);
```

```
  draw_square(east, fp_out, width, _level);
  draw_square(south, fp_out, width, _level);
  draw_square(west, fp_out, width, _level);

}  /** End of function draw_red_cross() **/

/***------------------------------------------------------------------***/
void draw_bigger_cross(Point loc, FILE *fp_out, double width, int
_level)
/***
Takes input location (CENTER!) and draws 5 squares in the
shape of the red cross symbol for the alignment structure.
***/
{
  double new_width;                  /* decrement width by 2.0     */
  Point north,                       /* position of square top     */
        east,                        /* position of square right   */
        south,                       /* position of square bottom  */
        west,                        /* position of square left    */
        center;                      /* position of square center  */

  /*** assign new_width ***/
  new_width = width - 2.0;

  /*** assign coordinates to all the squares ***/
  center.x1 = loc.x1 - width/2.0;
  center.y1 = loc.y1 - width/2.0;
  north.x1 = center.x1;
  north.y1 = center.y1 - new_width;
  north.x2 = center.x1 + width;
  north.y2 = center.y1;
  south.x1 = center.x1;
  south.y1 = center.y1 + width;
  south.x2 = center.x1 + width;
  south.y2 = center.y1 + width + new_width;
  east.x1 = center.x1 - new_width;
  east.y1 = center.y1;
  east.x2 = center.x1;
  east.y2 = center.y1 + width;
  west.x1 = center.x1 + width;
  west.y1 = center.y1;
  west.x2 = center.x1 + width + new_width;
  west.y2 = center.y1 + width;

  /*** draw the 5 squares to form the red cross symbol ***/
  draw_square(center, fp_out, width, _level);
  write_led_bar(fp_out, north, _level);
  write_led_bar(fp_out, east, _level);
  write_led_bar(fp_out, south, _level);
  write_led_bar(fp_out, west, _level);


}  /** End of function draw_bigger_cross() **/
```

```
/***-----------------------------------------------------------------***/
void draw_diag_squares(Point loc, FILE *fp_out, double width, int
_level)
/***
Takes input location (CENTER!!) and draws two squares with
corner touching at diagonal.
***/
{
  Point left_sq,                   /* stores coords of left square */
        right_sq;                  /* stores coords of right square*/

  /*** assign coordinates for the squares ***/
  left_sq.x1 = loc.x1 - width;
  left_sq.y1 = loc.y1 - width;
  right_sq.x1 = loc.x1;
  right_sq.y1 = loc.y1;

  /*** draw both the squares ***/
  draw_square(left_sq, fp_out, width, _level);
  draw_square(right_sq, fp_out, width, _level);

}  /** end of function draw_diag_squares() **/

/***-----------------------------------------------------------------***/
void draw_square(Point loc, FILE *fp_out, double width, int _level)
/***
Draws square with edges width (length) positioning left corner of
square at loc.
***/
{
  Point square;                    /* stores coordinates of square */

  /*** assign coordinates for square ***/
  square.x1 = loc.x1;
  square.y1 = loc.y1;
  square.x2 = loc.x1 + width;
  square.y2 = loc.y1 + width;

  /*** draw square ***/
  write_led_bar(fp_out, square, _level);

}  /** end of function draw_square() **/

/***-----------------------------------------------------------------***/
void draw_bar_code(Point loc, FILE *fp_out, int _level)
/***
Draws barcode structure at the location given, varying
lengths between 1 and 10 and maintaining a length of 25 microns.
***/
{
  int i;                           /* index variable               */
  double width,                    /* width of each rectangle      */
         height,                   /* height of all the rectangles */
         spacing;                  /* spacing between rectangles   */
```

```
  Point rect_loc,                   /* stores coordinates of rect   */
        rect2_loc;                  /* stores coords of 2nd rect    */

  /*** define rectangle dimensions ***/
  width = 10.0;
  height = 25.0;
  spacing = 2.0;

  /*** define initial coordinates ***/
  rect_loc.x1 = loc.x1;
  rect_loc.y1 = loc.y1 + spacing;
  rect2_loc.x1 = loc.x1;
  rect2_loc.y1 = loc.y1 - height - spacing;

  /*** draw rectangles and adjust coordinates (w=10,8,6,4,2) ***/
  for( i=0; i<5; i++ ) {
    draw_rectangle(rect_loc, fp_out, _level, width, height);
    draw_rectangle(rect2_loc, fp_out, _level, width, height);
    rect_loc.x1 += width + spacing;
    rect2_loc.x1 += 2 * width;
    width -= 2.0;
  }

  /*** draw final rectangle for w=1 ***/
  width = 1.0;
  draw_rectangle(rect_loc, fp_out, _level, width, height);
  draw_rectangle(rect2_loc, fp_out, _level, width, height);

}  /** end of function draw_bar_code() **/

/***---------------------------------------------------------------***/
void draw_rectangle(Point loc, FILE *fp_out, int _level,
                double width, double height)
/***
Draws rectangle with width and height, positioning left corner of
rectangle at loc.
***/
{
  Point rect;                       /* stores coordinates of square */

  /*** assign coordinates for rect ***/
  rect.x1 = loc.x1;
  rect.y1 = loc.y1;
  rect.x2 = loc.x1 + width;
  rect.y2 = loc.y1 + height;

  /*** draw rect ***/
  write_led_bar(fp_out, rect, _level);

}  /** end of function draw_rect() **/

/***-============================================================-***/
```

The following file, mask_alphnum.c, enables numeric and alphabetic character writing on the mask.

```
/***   mask_alphnum.c support file   ***/

/*** libraries ***/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_alphnum.h"
#include "mask_bond.h"
#include "mask_probe.h"

/*** function listing **********

void write_andleigh()

void write_emsim()

void write_bites()

void parse_length()

void parse_width()

void define_letter()

void define_num()

void write_alphnum()

void write_led_bar()

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

/***==============================================================***/
void write_andleigh(Point _alphnum_loc, FILE *fp_out)
/***
Writes V.K. Andleigh text.
***/
{
  Point loc;                              /* temp var for loc  */

  /*** define loc temp variable ***/
  loc.x1 = _alphnum_loc.x1;
  loc.y1 = _alphnum_loc.y1;
```

```
  /*** write V.K. ***/
  define_letter(loc, fp_out, HORIZ, 18);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 16);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 17);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 16);
  loc.x1 += HLET_SPAC;
  loc.x1 += HLET_SPAC;

  /*** write Andleigh ***/
  define_letter(loc, fp_out, HORIZ, 8);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 13);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, 0);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 1);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 11);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, 6);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 14);
  loc.x1 += HLET_SPAC;


  /*** write ph.d. ***/
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 15);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 14);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 16);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, 0);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 16);
  loc.x1 += HLET_SPAC;
}  /** End of function write_andleigh() **/

/***-----------------------------------------------------------------***/
void write_emsim(Point _alphnum_loc, FILE *fp_out)
/***
Writes MIT/EmSim text.
***/
{
  Point loc;                              /* temp var for loc  */

  /*** define loc temp variable ***/
  loc.x1 = _alphnum_loc.x1;
```

```
  loc.y1 = _alphnum_loc.y1;

  /*** write MIT ***/
  define_letter(loc, fp_out, HORIZ, 7);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 10);
  loc.x1 += HLET_SPAC;

  /*** write EmSim ***/
  define_letter(loc, fp_out, HORIZ, 11);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 7);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 12);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 7);
  loc.x1 += HLET_SPAC;

  /*** write rules ***/
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 19);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 18);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 1);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 11);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 12);
  loc.x1 += HLET_SPAC;

}  /** End of function write_emsim() **/

/***--------------------------------------------------------------***/
void write_bites(Point _alphnum_loc, FILE *fp_out)
/***
Writes MIT/EmSim text.
***/
{
  Point loc;                              /* temp var for loc  */

  /*** define loc temp variable ***/
  loc.x1 = _alphnum_loc.x1;
  loc.y1 = _alphnum_loc.y1;

  /*** write MIT ***/
  define_letter(loc, fp_out, HORIZ, 7);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
```

```
  define_letter(loc, fp_out, HORIZ, 10);
  loc.x1 += HLET_SPAC;
  loc.x1 += HLET_SPAC;

  /*** write bites ***/
  define_num(loc, fp_out, HORIZ, 8);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 10);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 11);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 12);
  loc.x1 += HLET_SPAC;

}  /** End of function write_emsim() **/
/***-------------------------------------------------------------------***/
void write_microsoft(Point _alphnum_loc, FILE *fp_out)
/***
Writes MIT/EmSim text.
***/
{
  Point loc;                          /* temp var for loc  */

  /*** define loc temp variable ***/
  loc.x1 = _alphnum_loc.x1;
  loc.y1 = _alphnum_loc.y1;

  /*** write microsoft ***/
  define_letter(loc, fp_out, HORIZ, 7);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 20);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 19);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, 0);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 12);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, 0);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 21);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 10);
  loc.x1 += HLET_SPAC;
  loc.x1 += HLET_SPAC;

  /*** write bites ***/
  define_num(loc, fp_out, HORIZ, 8);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 9);
```

```
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 10);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 11);
  loc.x1 += HLET_SPAC;
  define_letter(loc, fp_out, HORIZ, 12);
  loc.x1 += HLET_SPAC;

}  /** End of function write_microsoft() **/

/***-------------------------------------------------------------***/
void parse_length(Point *alphnum_loc, FILE *fp_out, int _orient,
              double _length)
/***
Takes input length and writes "L" and corresponding digits for the
line length.
***/
{
  int hundreds=0,                        /* hundreds digit    */
      tens=0,                            /* tens digit        */
      units=0,                           /* units digit       */
      len;                               /* modifiable length */
  Point loc;                             /* temp var for loc  */

  /*** recast length as int and parse length into digits ***/
  len = (int) (_length + 0.1);
  while( len >= 100 ) {
    hundreds++;
    len -= 100;
  }
  while( len >= 10 ) {
    tens++;
    len -= 10;
  }
  units = len;

  /*** define loc temp variable ***/
  loc.x1 = alphnum_loc->x1;
  loc.y1 = alphnum_loc->y1;

  /*** if vertical orientation, write text vertically ***/
  if( _orient == VERT ) {
    /*** write full L text only if large enough, not for 25, 50 ***/
    if( _length > 50.0 ) {
      /*** write arrow and L ***/
      define_4_letter(loc, fp_out, VERT, 4, 4);
      loc.y1 -= VLET_SPAC;
      define_4_letter(loc, fp_out, VERT, 1, 2);
      loc.y1 -= VLET_SPAC;
      /*** write length ***/
      define_4_num(loc, fp_out, VERT, hundreds, 0);
      loc.y1 -= VLET_SPAC;
      define_4_num(loc, fp_out, VERT, tens, -2);
      loc.y1 -= VLET_SPAC;
```

```
      define_4_num(loc, fp_out, VERT, units, -4);
      loc.y1 -= VLET_SPAC;
    } else {
      define_4_letter(loc, fp_out, VERT, 1, 0);
      loc.y1 -= VLET_SPAC;
      define_4_num(loc, fp_out, VERT, tens, -2);
      loc.y1 -= VLET_SPAC;
      define_4_num(loc, fp_out, VERT, units, -4);
      loc.y1 -= VLET_SPAC;
    }
  }

  /*** if horizontal orientation, write text horizontally ***/
  if( _orient == HORIZ ) {
    /*** write full L text only if large enough, not for 25, 50 ***/
    if( _length > 50.0 ) {
      /*** write arrow and L ***/
      define_4_letter(loc, fp_out, HORIZ, 5, 0);
      loc.x1 += HLET_SPAC;
      define_4_letter(loc, fp_out, HORIZ, 1, -3);
      loc.x1 += HLET_SPAC;
      /*** write length ***/
      define_4_num(loc, fp_out, HORIZ, hundreds, -1);
      loc.x1 += HLET_SPAC;
      define_4_num(loc, fp_out, HORIZ, tens, 1);
      loc.x1 += HLET_SPAC;
      define_4_num(loc, fp_out, HORIZ, units, 3);
      loc.x1 += HLET_SPAC;
    } else {
      define_4_letter(loc, fp_out, HORIZ, 1, -2);
      loc.x1 += HLET_SPAC;
      define_4_num(loc, fp_out, HORIZ, tens, 0);
      loc.x1 += HLET_SPAC;
      define_4_num(loc, fp_out, HORIZ, units, 2);
      loc.x1 += HLET_SPAC;
    }
  }

  /*** write var changes ***/
  alphnum_loc->x1 = loc.x1;
  alphnum_loc->y1 = loc.y1;

}  /** End of function parse_length() **/

/***-------------------------------------------------------------***/
void parse_width(Point *alphnum_loc, FILE *fp_out, int _orient,
             double _width_bin)
/***
Takes input width_bin and writes "W" text along with integer
value of width given the initial coordinates.
***/
{
  int width;                            /* int value of width */
  Point loc;                            /* temp var for loc   */
```

```
  /*** define loc temp variable ***/
  loc.x1 = alphnum_loc->x1;
  loc.y1 = alphnum_loc->y1;

  /*** write W ***/
  define_4_letter(loc, fp_out, VERT, 2, 0);

  /*** determine linewidth ***/
  width = width_chart(_width_bin);

  /*** if vertical orientation, write text vertically ***/
  if( _orient == VERT ) {
    /*** advance to next character ***/
    loc.y1-= VLET_SPAC;
    /*** write width ***/
    if( width < 10 )
      define_4_num(loc, fp_out, VERT, width, 0);
    else
      if( width == 10 ) {
       define_4_num(loc, fp_out, VERT, 1, 1);
       loc.y1 -= VLET_SPAC;
       define_4_num(loc, fp_out, VERT, 0, -1);
      }
    loc.y1 -= VLET_SPAC;
  }

  /*** if horizontal orientation, write text horizontally ***/
  if( _orient == HORIZ ) {
    /*** advance to next character ***/
    loc.x1 += HLET_SPAC;
    /*** write width ***/
    if( width < 10 )
      define_4_num(loc, fp_out, HORIZ, width, 0);
    else
      if( width == 10 ) {
       define_4_num(loc, fp_out, HORIZ, 1, -1);
       loc.x1 += HLET_SPAC;
       define_4_num(loc, fp_out, HORIZ, 0, 1);
      }
    loc.x1 += HLET_SPAC;
  }

  /*** write var changes ***/
  alphnum_loc->x1 = loc.x1;
  alphnum_loc->y1 = loc.y1;

}  /** End of function parse_width() **/

/***----------------------------------------------------------------***/
void parse_align_num(Point _alphnum_loc, FILE *fp_out, int _level,
                 int _arrow_dir)
/***
Takes input location and writes arrow and A 1-<_level> to signify
```

```
which mask levels are being aligned.
***/
{
  Point loc;                              /* temp var for loc   */

  /*** define loc temp variable ***/
  loc.x1 = _alphnum_loc.x1;
  loc.y1 = _alphnum_loc.y1;

  /*** write arrow if on left side ***/
  if( _arrow_dir == 6 )
    define_letter(loc, fp_out, VERT, 6);
  loc.x1 += HLET_SPAC;

  /*** write A ***/
  define_letter(loc, fp_out, VERT, 8);

  /*** advance to next character ***/
  loc.x1 += HLET_SPAC;

  /*** write lower mask level ***/
  define_num(loc, fp_out, HORIZ, 1);
  loc.x1 += HLET_SPAC;
  define_num(loc, fp_out, HORIZ, _level);

  /*** write arrow if on right side ***/
  loc.x1 += HLET_SPAC;
  if( _arrow_dir == 4 )
    define_letter(loc, fp_out, VERT, 4);

}  /** End of function parse_align_num() **/

/***------------------------------------------------------------***/
void define_4_letter(Point alphnum_loc, FILE *fp_out, int _orient,
                 int _letter_struct, int displ)
/***
Takes the input request to write a letter, and resends the request
to write the letter in each of the other four quadrants.
***/
{
  int high_struct,                          /* higher type of strt */
      low_struct;                           /* lower type of struct*/
  double die_len,                           /* local version       */
         text_shift_h,                      /* horiz text shift    */
         text_shift_v;                      /* vert text shift     */
  Point quad1,                              /* write to quadrant 1 */
        quad2,                              /* write to quadrant 2 */
        quad3,                              /* write to quadrant 3 */
        quad4;                              /* write to quadrant 4 */

  /*** re-adjust die_len for mask coordinates, not fine coordinates
***/
  die_len = HALF_DIE_LEN / 100.0;
```

```
  /*** determine vertical and horizontal text shifts ***/
  text_shift_h = HLET_SPAC;
  text_shift_v = VLET_SPAC;

  /*** determine coordinates in other quadrants ***/
  quad1.x1 = alphnum_loc.x1;
  quad1.y1 = alphnum_loc.y1;
  if( _orient == HORIZ )
    quad2.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0 +
displ*text_shift_h;
  else
    quad2.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0;
  quad2.y1 = quad1.y1;
  if( _orient == HORIZ )
    quad3.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0 +
displ*text_shift_h;
  else
    quad3.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0;
  if( _orient == VERT )
    quad3.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0+ displ
*text_shift_v;
  else
    quad3.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0;
  quad4.x1 = quad1.x1;
  if( _orient == VERT )
    quad4.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0+ displ
*text_shift_v;
  else
    quad4.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0;

  /*** determine high/low letter companions ***/
  high_struct = _letter_struct + 2;
  low_struct  = _letter_struct - 2;

  /*** write letters for L and W ***/
  if( (_letter_struct==1) || (_letter_struct==2) ) {
    define_letter(quad1, fp_out, _orient, _letter_struct);
    define_letter(quad2, fp_out, _orient, _letter_struct);
    define_letter(quad3, fp_out, _orient, _letter_struct);
    define_letter(quad4, fp_out, _orient, _letter_struct);
  }

  /*** write opposing arrows for each sector where appropriate ***/
  define_letter(quad1, fp_out, _orient, _letter_struct);
  /*** down arrow ***/
  if( _letter_struct == 3 ) {
    define_letter(quad2, fp_out, _orient, _letter_struct);
    define_letter(quad3, fp_out, _orient, high_struct);
    define_letter(quad4, fp_out, _orient, high_struct);
  }
  /*** right arrow ***/
  if( _letter_struct == 4) {
    define_letter(quad2, fp_out, _orient, high_struct);
    define_letter(quad3, fp_out, _orient, high_struct);
```

```
    define_letter(quad4, fp_out, _orient, _letter_struct);
  }
   /*** up arrow ***/
  if( _letter_struct == 5) {
    define_letter(quad2, fp_out, _orient, _letter_struct);
    define_letter(quad3, fp_out, _orient, low_struct);
    define_letter(quad4, fp_out, _orient, low_struct);
  }
  /*** left arrow ***/
  if( _letter_struct == 6) {
    define_letter(quad2, fp_out, _orient, low_struct);
    define_letter(quad3, fp_out, _orient, low_struct);
    define_letter(quad4, fp_out, _orient, _letter_struct);
  }

}  /** End of function define_4_letter() **/

/***-------------------------------------------------------------***/
void define_4_num(Point alphnum_loc, FILE *fp_out, int _orient,
              int _num, int displ)
/***
Takes the input request to write a number, and resends the request
to write the letter in each of the other four quadrants.
***/
{
  double die_len,                            /* local version       */
         text_shift_h,                       /* horiz text shift    */
         text_shift_v;                       /* vert text shift     */
  Point quad1,                               /* write to quadrant 1 */
        quad2,                               /* write to quadrant 2 */
        quad3,                               /* write to quadrant 3 */
        quad4;                               /* write to quadrant 4 */

  /*** re-adjust die_len for mask coordinates, not fine coordinates
***/
  die_len = HALF_DIE_LEN / 100.0;

  /*** determine vertical and horizontal text shifts ***/
  text_shift_h = HLET_SPAC;
  text_shift_v = VLET_SPAC;

  /*** determine coordinates in other quadrants ***/
  quad1.x1 = alphnum_loc.x1;
  quad1.y1 = alphnum_loc.y1;
  if( _orient == HORIZ )
    quad2.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0 +
displ*text_shift_h;
  else
    quad2.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0;
  quad2.y1 = quad1.y1;
  if( _orient == HORIZ )
    quad3.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0 +
displ*text_shift_h;
  else
```

```
    quad3.x1 = 2 * die_len - quad1.x1 - TEXT_WIDTH -174.0;
  if( _orient == VERT )
    quad3.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0+ displ
*text_shift_v;
  else
    quad3.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0;
  quad4.x1 = quad1.x1;
  if( _orient == VERT )
    quad4.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0+ displ
*text_shift_v;
  else
    quad4.y1 = 2 * die_len - quad1.y1 - TEXT_HEIGHT -95.0;

  /*** write numbers ***/
  define_num(quad1, fp_out, _orient, _num);
  define_num(quad2, fp_out, _orient, _num);
  define_num(quad3, fp_out, _orient, _num);
  define_num(quad4, fp_out, _orient, _num);

}  /** End of function define_4_letter() **/

/***------------------------------------------------------------------***/
void define_letter(Point alphnum_loc, FILE *fp_out, int _orient,
               int _letter_struct)
/***
Defines the coordinates for the given letter structure and writes them
to the mask output file.
***/
{
  /*** write L ***/
  if( _letter_struct == 1 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 7);
  }

  /*** write W ***/
  if( _letter_struct == 2 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
    write_alphnum(alphnum_loc, fp_out, 12);
    write_alphnum(alphnum_loc, fp_out, 13);
  }

  /*** to write arrow down ***/
  if( _letter_struct == 3 ) {
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 6);
```

```
  write_alphnum(alphnum_loc, fp_out, 7);
}

/*** to write arrow right ***/
if( _letter_struct == 4 ) {
  write_alphnum(alphnum_loc, fp_out, 3);
  write_alphnum(alphnum_loc, fp_out, 4);
  write_alphnum(alphnum_loc, fp_out, 6);
  write_alphnum(alphnum_loc, fp_out, 11);
  write_alphnum(alphnum_loc, fp_out, 13);
}

/*** to write arrow up ***/
if( _letter_struct == 5 ) {
  write_alphnum(alphnum_loc, fp_out, 8);
  write_alphnum(alphnum_loc, fp_out, 9);
  write_alphnum(alphnum_loc, fp_out, 1);
  write_alphnum(alphnum_loc, fp_out, 2);
  write_alphnum(alphnum_loc, fp_out, 3);
}

/*** to write arrow left ***/
if( _letter_struct == 6 ) {
  write_alphnum(alphnum_loc, fp_out, 2);
  write_alphnum(alphnum_loc, fp_out, 4);
  write_alphnum(alphnum_loc, fp_out, 5);
  write_alphnum(alphnum_loc, fp_out, 10);
  write_alphnum(alphnum_loc, fp_out, 12);
}

/*** write M ***/
if( _letter_struct == 7 ) {
  write_alphnum(alphnum_loc, fp_out, 2);
  write_alphnum(alphnum_loc, fp_out, 3);
  write_alphnum(alphnum_loc, fp_out, 5);
  write_alphnum(alphnum_loc, fp_out, 6);
  write_alphnum(alphnum_loc, fp_out, 8);
  write_alphnum(alphnum_loc, fp_out, 9);
  write_alphnum(alphnum_loc, fp_out, 10);
  write_alphnum(alphnum_loc, fp_out, 11);
}

/*** write A ***/
if( _letter_struct == 8 ) {
  write_alphnum(alphnum_loc, fp_out, 1);
  write_alphnum(alphnum_loc, fp_out, 2);
  write_alphnum(alphnum_loc, fp_out, 3);
  write_alphnum(alphnum_loc, fp_out, 4);
  write_alphnum(alphnum_loc, fp_out, 5);
  write_alphnum(alphnum_loc, fp_out, 6);
}
/*** write I ***/
if( _letter_struct == 9 ) {
  write_alphnum(alphnum_loc, fp_out, 8);
```

```
    write_alphnum(alphnum_loc, fp_out, 9);
  }

  /*** write T ***/
  if( _letter_struct == 10 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
  }

  /*** write E ***/
  if( _letter_struct == 11 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 7);
  }

  /*** write S ***/
  if( _letter_struct == 12 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }

  /*** write N ***/
  if( _letter_struct == 13 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 10);
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
    write_alphnum(alphnum_loc, fp_out, 13);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 3);
  }

  /*** write H ***/
  if( _letter_struct == 14 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 6);
  }

  /*** write P ***/
  if( _letter_struct == 15 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
```

```
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
  }

  /*** write (.) period ***/
  if( _letter_struct == 16 ) {
    write_alphnum(alphnum_loc, fp_out, 12);
  }

  /*** write K ***/
  if( _letter_struct == 17 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 14);
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
    write_alphnum(alphnum_loc, fp_out, 11);
    write_alphnum(alphnum_loc, fp_out, 13);
  }

  /*** write U ***/
  if( _letter_struct == 18 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 7);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 6);
  }

  /*** write R ***/
  if( _letter_struct == 19 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 9);
  }

  /*** write C ***/
  if( _letter_struct == 20 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 7);
  }

  /*** write F ***/
  if( _letter_struct == 21 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
  }
```

```
}  /** End of function define_letter() **/

/***----------------------------------------------------------------***/
void define_num(Point alphnum_loc, FILE *fp_out, int _orient, int _num)
/***
Given a specific number to write, defines coordinates for this number
and writes it to the file accordingly.
***/
{
  /*** to write 1 ***/
  if( _num == 1 ) {
    write_alphnum(alphnum_loc, fp_out, 8);
    write_alphnum(alphnum_loc, fp_out, 9);
  }
  /*** to write 2 ***/
  if( _num == 2 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 7);
  }
  /*** to write 3 ***/
  if( _num == 3 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }
  /*** to write 4 ***/
  if( _num == 4 ) {
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 6);
  }
  /*** to write 5 ***/
  if( _num == 5 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }
  /*** to write 6 ***/
  if( _num == 6 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
```

```
  }
  /*** to write 7 ***/
  if( _num == 7 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 6);
  }
  /*** to write 8 ***/
  if( _num == 8 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }
  /*** to write 9 ***/
  if( _num == 9 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 4);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }
  /*** to write 0 ***/
  if( _num == 0 ) {
    write_alphnum(alphnum_loc, fp_out, 1);
    write_alphnum(alphnum_loc, fp_out, 2);
    write_alphnum(alphnum_loc, fp_out, 3);
    write_alphnum(alphnum_loc, fp_out, 5);
    write_alphnum(alphnum_loc, fp_out, 6);
    write_alphnum(alphnum_loc, fp_out, 7);
  }

}  /** end of function define_num() **/

/***-----------------------------------------------------------------***/
void write_alphnum(Point alphnum_loc, FILE *fp_out, int led_num)
/***
Given which element of the LED structure to represent, determines
coordinates of this structure and writes them to file.
Note that to make these match up with kic, I had to flip the
paper map of led positions upside down to correspond to my reverse
x-y axis.
***/
{
  Point led_bar;                          /* corresponding LED bar    */

  /*** structure for top bar ***/
  if( led_num == 7 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 0.0  + alphnum_loc.y1;
```

```
    led_bar.x2 = 20.0 + alphnum_loc.x1;
    led_bar.y2 = 5.0  + alphnum_loc.y1;
  }
  /*** structure for upper left bar ***/
  if( led_num == 5 ) {
    led_bar.x1 = 0.0  + alphnum_loc.x1;
    led_bar.y1 = 5.0  + alphnum_loc.y1;
    led_bar.x2 = 5.0  + alphnum_loc.x1;
    led_bar.y2 = 20.0 + alphnum_loc.y1;
  }
  /*** structure for upper right bar ***/
  if( led_num == 6 ) {
    led_bar.x1 = 20.0 + alphnum_loc.x1;
    led_bar.y1 = 5.0  + alphnum_loc.y1;
    led_bar.x2 = 25.0 + alphnum_loc.x1;
    led_bar.y2 = 20.0 + alphnum_loc.y1;
  }
  /*** structure for middle bar ***/
  if( led_num == 4 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 20.0 + alphnum_loc.y1;
    led_bar.x2 = 20.0 + alphnum_loc.x1;
    led_bar.y2 = 25.0 + alphnum_loc.y1;
  }
  /*** structure for lower left bar ***/
  if( led_num == 2 ) {
    led_bar.x1 = 0.0  + alphnum_loc.x1;
    led_bar.y1 = 25.0 + alphnum_loc.y1;
    led_bar.x2 = 5.0  + alphnum_loc.x1;
    led_bar.y2 = 40.0 + alphnum_loc.y1;
  }
  /*** structure for lower right bar ***/
  if( led_num == 3 ) {
    led_bar.x1 = 20.0 + alphnum_loc.x1;
    led_bar.y1 = 25.0 + alphnum_loc.y1;
    led_bar.x2 = 25.0 + alphnum_loc.x1;
    led_bar.y2 = 40.0 + alphnum_loc.y1;
  }
  /*** structure for lower bar ***/
  if( led_num == 1 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 40.0 + alphnum_loc.y1;
    led_bar.x2 = 20.0 + alphnum_loc.x1;
    led_bar.y2 = 45.0 + alphnum_loc.y1;
  }
  /*** structure for upper middle w bar ***/
  if( led_num == 9 ) {
    led_bar.x1 = 11.0 + alphnum_loc.x1;
    led_bar.y1 = 5.0  + alphnum_loc.y1;
    led_bar.x2 = 14.0 + alphnum_loc.x1;
    led_bar.y2 = 20.0 + alphnum_loc.y1;
  }
  /*** structure for lower middle w bar ***/
  if( led_num == 8 ) {
```

```
    led_bar.x1 = 11.0 + alphnum_loc.x1;
    led_bar.y1 = 25.0 + alphnum_loc.y1;
    led_bar.x2 = 14.0 + alphnum_loc.x1;
    led_bar.y2 = 40.0 + alphnum_loc.y1;
  }
  /*** structure for left part of top bar ***/
  if( led_num == 10 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 40.0 + alphnum_loc.y1;
    led_bar.x2 = 10.0 + alphnum_loc.x1;
    led_bar.y2 = 45.0 + alphnum_loc.y1;
  }
  /*** structure for right part of top bar ***/
  if( led_num == 11 ) {
    led_bar.x1 = 15.0 + alphnum_loc.x1;
    led_bar.y1 = 40.0 + alphnum_loc.y1;
    led_bar.x2 = 20.0 + alphnum_loc.x1;
    led_bar.y2 = 45.0 + alphnum_loc.y1;
  }
  /*** structure for left part of bottom bar ***/
  if( led_num == 12 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 0.0  + alphnum_loc.y1;
    led_bar.x2 = 10.0 + alphnum_loc.x1;
    led_bar.y2 = 5.0  + alphnum_loc.y1;
  }
  /*** structure for left part of bottom bar ***/
  if( led_num == 13 ) {
    led_bar.x1 = 15.0 + alphnum_loc.x1;
    led_bar.y1 = 0.0  + alphnum_loc.y1;
    led_bar.x2 = 20.0 + alphnum_loc.x1;
    led_bar.y2 = 5.0  + alphnum_loc.y1;
  }
  /*** structure for left portion of middle bar ***/
  if( led_num == 14 ) {
    led_bar.x1 = 5.0  + alphnum_loc.x1;
    led_bar.y1 = 20.0 + alphnum_loc.y1;
    led_bar.x2 = 10.0 + alphnum_loc.x1;
    led_bar.y2 = 25.0 + alphnum_loc.y1;
  }

  /*** write the led bar ***/
  write_led_bar(fp_out, led_bar, 1);

}  /** end of function write_alphnum() **/

/***---------------------------------------------------------------***/
void write_led_bar(FILE *fp_out, Point p_coord, int _level)
/***
Takes the coordinate p_coordure and writes the four data
points in the format of KIC.  Only one quadrant is drawn.
***/
{
```

```
  double xc,                          /* center position of x      */
         yc,                          /* center position of y      */
         width,                       /* width of block            */
         height;                      /* height of block           */

  /*** calculate center coordinates ***/
  xc = (p_coord.x1 + p_coord.x2) / 2.0 * 100.0;
  yc = (p_coord.y1 + p_coord.y2) / 2.0 * 100.0;

  /*** calculate width, height ***/
  width  = (p_coord.x2 - p_coord.x1) * 100.0;
  height = (p_coord.y2 - p_coord.y1) * 100.0;

  /*** upper left quadrant ***/
  /*** write block label ***/
  fprintf(fp_out, "B ");

  /*** write width, height ***/
  fprintf(fp_out, "%.0f %.0f ", width, height);

  /*** write center coordinates ***/
  fprintf(fp_out, "%.0f %.0f; \n", xc, yc);

}  /** end of function write_led_bar **/

/***-==============================================================-***/
```

The following file, mask_bond.c, lays out the bond pad structures spread around the edges of each die. The interconnects associated with these bond pads are connected using the snake structure.

```
/***   mask_bond.c support file   ***/

/*** libraries ***/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_bond.h"

/*** function listing **********

void define_struct()

void define_strands()

void define_pads()
```

```
void define_links()

void define_centered_line()

void define_line()

void define_tung()

void define_end_stops()

void write_coords()

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

/***=============================================================***/
void define_bond_struct(Point _offset,
                  FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                  int _orient, double pad_len,
                  double pad_spacing, double strand_width,
                  double link_len_pi, double link_len_pv,
                  double strand_len_li, double strand_len_lv,
                  double link_pi_offset, double lv_overhang)
/***
Defines the repeating current, voltage pad structure without test lines
given the offset and orientation.
***/
{
  /*** define pads ***/
  define_pads(_offset, fp_mask1, fp_mask3, _orient, pad_len,
pad_spacing, strand_width, strand_len_li, link_len_pi, link_pi_offset,
lv_overhang);

  /*** define links ***/
  define_links(_offset, fp_mask1, _orient, pad_len, pad_spacing,
strand_width, strand_len_li, link_len_pi, link_len_pv, link_pi_offset,
lv_overhang);

  /*** define strands ***/
  define_strands(_offset, fp_mask1, _orient, pad_len, pad_spacing,
strand_width, link_len_pi, link_len_pv, strand_len_li, strand_len_lv,
link_pi_offset, lv_overhang);

}  /** End of function define_struct() **/

/***-------------------------------------------------------------***/
void define_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3, int
_orient,
             double _pad_len, double _pad_spacing, double
_strand_width,
             double _strand_len_li, double _link_len_pi,
             double _link_pi_offset, double _lv_overhang)
/***
Determine coordinates of the five pads and write them to
```

```
the output file.
***/
{
  Point padi,                          /* current pad                  */
        padv,                          /* voltage pad                  */
        coord;                         /* holds true coordinate data   */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    padv.x1 = 0.0;
    padv.y1 = _lv_overhang;
    padi.x1 = 0.0;
    padi.y1 = padv.y1 + _pad_len + _pad_spacing;
  } else {   /* HORIZ */
    padi.x1 = -1 * _pad_len - _link_len_pi;
    padi.y1 = -1 * _strand_width - _strand_len_li - _link_pi_offset;
    padv.x1 = _pad_spacing - _link_len_pi;
    padv.y1 = padi.y1;
  }

  /*** step through each pad and determine coordinates with offsets
***/

    /*** add offsets, determine x2,y2 coordinates padi ***/
    coord.x1 = padi.x1 + offset.x1;
    coord.y1 = padi.y1 + offset.y1;
    coord.x2 = coord.x1 + _pad_len;
    coord.y2 = coord.y1 + _pad_len;

    /*** write data to file ***/
    write_coords(fp_mask1, coord, 1);
    write_coords(fp_mask3, coord, 1);

    /*** add offsets, determine x2,y2 coordinates padv ***/
    coord.x1 = padv.x1 + offset.x1;
    coord.y1 = padv.y1 + offset.y1;
    coord.x2 = coord.x1 + _pad_len;
    coord.y2 = coord.y1 + _pad_len;

    /*** write data to file ***/
    write_coords(fp_mask1, coord, 1);
    write_coords(fp_mask3, coord, 1);

}  /** end of function define_pads() **/

/***-----------------------------------------------------------------***/
void define_links(Point offset, FILE *fp_out, int _orient,
              double _pad_len, double _pad_spacing,
              double _strand_width, double _strand_len_li,
              double _link_len_pi, double _link_len_pv,
              double _link_pi_offset, double _lv_overhang)
/***
Determine coordinates of the five links and write them to
the output file.
```

```
***/
{
  Point link_pi,                          /* coords for link to I pad    */
        link_pv,                          /* coords for link to V pad    */
        coord;                            /* holds true coordinate data  */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    link_pi.x1  = _link_pi_offset;
    link_pi.y1  = _lv_overhang + _pad_len + _pad_spacing -
_link_len_pi;
    link_pv.x1  = _pad_len;
    link_pv.y1  = _lv_overhang + 0.5 * (_pad_len - _strand_width);
  } else {   /* HORIZ */
    link_pi.x1  = -1* _link_len_pi;
    link_pi.y1  = -1* _strand_width -_strand_len_li;
    link_pv.x1  = _pad_spacing - _link_len_pi;
    link_pv.x1 += 0.5 * (_pad_len - _strand_width);
    link_pv.y1  = -1 * _strand_width - _link_len_pv;
  }

  /*** step through each link and determine coordinates with offsets
***/

  /*** add offsets, determine x2,y2 coordinates, link_pi ***/
  coord.x1 = link_pi.x1 + offset.x1;
  coord.y1 = link_pi.y1 + offset.y1;
  if( _orient == VERT ) {
    coord.x2 = coord.x1 + _strand_width;
    coord.y2 = coord.y1 + _link_len_pi;
  } else {   /* HORIZ */
    coord.x2 = coord.x1 + _link_len_pi;
    coord.y2 = coord.y1 + _strand_width;
  }

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

  /*** add offsets, determine x2,y2 coordinates, link_pv ***/
  coord.x1 = link_pv.x1 + offset.x1;
  coord.y1 = link_pv.y1 + offset.y1;
  if( _orient == VERT ) {
    coord.x2 = coord.x1 + _link_len_pv;
    coord.y2 = coord.y1 + _strand_width;
  } else {   /* HORIZ */
    coord.x2 = coord.x1 + _strand_width;
    coord.y2 = coord.y1 + _link_len_pv;
  }

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

} /** end of function define_links() **/
```

```
/***-------------------------------------------------------------***/
void define_strands(Point offset, FILE *fp_out, int _orient,
                double _pad_len, double _pad_spacing, double
_strand_width,
                double _link_len_pi, double _link_len_pv,
                double _strand_len_li, double _strand_len_lv,
                double _link_pi_offset, double _lv_overhang)
/***
Determine coordinates of the five links and write them to
the output file.
***/
{
  int i;                         /* index to step thru array    */
  Point strand_li,
        strand_lv,
        coord;                   /* holds true coordinate data  */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    strand_li.x1  = _link_pi_offset;
    strand_li.y1  = _lv_overhang + _pad_len + _pad_spacing -
_link_len_pi;
    strand_li.y1 -= _strand_width;
    strand_lv.x1  = _pad_len + _link_len_pv;
    strand_lv.y1  = 0.0;
  } else {   /* HORIZ */
    strand_li.x1  = 0.0;
    strand_li.y1  = -1 * _strand_width - _strand_len_li;
    strand_lv.x1  = 0.0;
    strand_lv.y1  = -1 * _strand_width;
  }

  /*** adjust for offsets ***/
  strand_li.x1  += offset.x1;
  strand_li.y1  += offset.y1;
  strand_lv.x1  += offset.x1;
  strand_lv.y1  += offset.y1;

  /*** write opposing corner coordinates ***/
  if( _orient == VERT ) {
    strand_li.x2  = strand_li.x1 + _strand_len_li;
    strand_li.y2  = strand_li.y1 + _strand_width;
    strand_lv.x2  = strand_lv.x1 + _strand_width;
    strand_lv.y2  = strand_lv.y1 + _strand_len_lv;
  } else {   /* HORIZ */
    strand_li.x2  = strand_li.x1 + _strand_width;
    strand_li.y2  = strand_li.y1 + _strand_len_li;
    strand_lv.x2  = strand_lv.x1 + _strand_len_lv;
    strand_lv.y2  = strand_lv.y1 + _strand_width;
  }

  /*** write data to file for each strand ***/
  write_coords(fp_out, strand_li, 1);
  write_coords(fp_out, strand_lv, 1);
```

```
}  /** end of function define_links() **/

/***------------------------------------------------------------***/
void define_centered_line(Point loc, FILE *fp_out, int _orient,
                    double _line_len, double _line_width)
/***
Given upper left location of line structure, determines new
coordinates such that line is centered between 10 um connectors.
Also responsible for writing data to file.
***/
{
  Point offset,                      /* offset on each line edge    */
        line;                        /* test line to be tested      */

  /*** determine offset necessary to center line ***/
  offset.x1 = 0.0;
  offset.y1 = 0.0;
  if( _orient == HORIZ )
    offset.y1 = (10.0 - _line_width) / 2.0;
  else  /* VERT */
    offset.x1 = (10.0 - _line_len) / 2.0;

  /*** write initial (upper left) coords ***/
  line.x1 = loc.x1 + offset.x1;
  line.y1 = loc.y1 + offset.y1;

  /*** define lower right coordinate ***/
  line.x2 = line.x1 + _line_len;
  line.y2 = line.y1 + _line_width;

  /*** write data to file ***/
  write_coords(fp_out, line, 1);

}  /** end of function define_centered_line() **/

/***------------------------------------------------------------***/
void define_line(Point loc, FILE *fp_out,
             double _line_len, double _line_width)
/***
Determine coordinates of the actual test structure line itself.
***/
{
  Point line;                        /* test line to be tested      */

  /*** write initial (upper left) coords ***/
  line.x1 = loc.x1;
  line.y1 = loc.y1;

  /*** define lower right coordinate ***/
  line.x2 = line.x1 + _line_len;
  line.y2 = line.y1 + _line_width;

  /*** write data to file ***/
```

```
  write_coords(fp_out, line, 1);

}  /** end of function define_line() **/

/***-----------------------------------------------------------------***/
void define_L_shape(Point left, Point right, FILE *fp_out,
                double _strand_width)
/***

Receives the coordinates of where to draw the L shape after the
actual test structure line and the horizontal bond pad structure.
Takes this data and computes the coordinates for the L shape object.
***/
{
  Point vert_box,                /* vertical box part of the L  */
        horiz_box;               /* horizontal box part of L    */

  /*** write initial coords ***/
  horiz_box.x1 = left.x1;
  horiz_box.y1 = left.y1;
  horiz_box.x2 = right.x1 + _strand_width;
  horiz_box.y2 = horiz_box.y1 + _strand_width;

  vert_box.x1 = right.x1;
  vert_box.y1 = right.y1;
  vert_box.x2 = horiz_box.x2;
  vert_box.y2 = left.y1;

  /*** write data to file ***/
  write_coords(fp_out, horiz_box, 1);
  write_coords(fp_out, vert_box,  1);

}  /** end of function define_tungs() **/

/***-----------------------------------------------------------------***/
void define_end_stops(Point line, FILE *fp_out, int _orient,
                  double length)
/***
Takes initial left coordinate of where line structure will be drawn
and places 10 by 10 blocks on mask to delineate the line structures.
***/
{
  double stop_thick,                   /* stop thickness           */
         stop_width,                   /* stop width rel. to line  */
         extra_width;                  /* overshoot of width align */
  Point start,                         /* coords for start stop    */
        end;                           /* coords for end stop      */

  /*** define stop thickness ***/
  extra_width = 4.0;
  stop_thick = 10.0;
  stop_width = RUN_WIDTH + extra_width;
```

```
  /*** do for horizontal and vertical structures structures ***/
  if( _orient == HORIZ ) {
    /*** determine start-stop coordinates ***/
    start.x1 = line.x1 - stop_thick;
    start.y1 = line.y1 - extra_width/2.0;
    start.x2 = line.x1;
    start.y2 = start.y1 + stop_width;
    /*** determine end-stop coordinates ***/
    end.x1 = line.x1 + length;
    end.y1 = line.y1 - extra_width/2.0;
    end.x2 = end.x1 + stop_thick;
    end.y2 = end.y1 + stop_width;
  } else {  /* VERT */
    /*** determine start-stop coordinates ***/
    start.x1 = line.x1 - extra_width/2.0;
    start.y1 = line.y1 - stop_thick;
    start.x2 = start.x1 + stop_width;
    start.y2 = line.y1;
    /*** determine end-stop coordinates ***/
    end.x1 = line.x1 - extra_width/2.0;
    end.y1 = line.y1 + length;
    end.x2 = end.x1 + stop_width;
    end.y2 = end.y1 + stop_thick;
  }

  /*** draw start and end stops ***/
  write_coords(fp_out, start, 1);
  write_coords(fp_out, end, 1);

} /** end of function define_end_stops() **/

/***---------------------------------------------------------------***/
void write_coords(FILE *fp_out, Point p_coord, int _level)
/***
Takes the coordinate p_coordure and writes the four data
points in the format of KIC.
***/
{

  double xc,                          /* center position of x     */
         yc,                          /* center position of y     */
         width,                       /* width of block           */
         height;                      /* height of block          */

  double die_len = 5.05e5;            /* die length on each side  */

  /*** calculate center coordinates ***/
  xc = (p_coord.x1 + p_coord.x2) / 2.0 * 100.0;
  yc = (p_coord.y1 + p_coord.y2) / 2.0 * 100.0;

  /*** calculate width, height ***/
  width  = (p_coord.x2 - p_coord.x1) * 100.0;
  height = (p_coord.y2 - p_coord.y1) * 100.0;
```

```
  /*** upper left quadrant ***/
  /*** write block label ***/
  fprintf(fp_out, "B ");

  /*** write width, height ***/
  fprintf(fp_out, "%.0f %.0f ", width, height);

  /*** write center coordinates ***/
  fprintf(fp_out, "%.0f %.0f; \n", xc, yc);

  /*** upper right quadrant ***/
 /*** write block label ***/
  fprintf(fp_out, "B ");

  /*** write width, height ***/
  fprintf(fp_out, "%.0f %.0f ", width, height);

  /*** write center coordinates ***/
  fprintf(fp_out, "%.0f %.0f; \n", 2*die_len - xc - 1.7e4, yc);

 /*** lower left quadrant ***/
 /*** write block label ***/
  fprintf(fp_out, "B ");

  /*** write width, height ***/
  fprintf(fp_out, "%.0f %.0f ", width, height);

  /*** write center coordinates ***/
  fprintf(fp_out, "%.0f %.0f; \n", xc, 2*die_len - yc - 0.95e4);

  /*** lower right quadrant ***/
  /*** write block label ***/
  fprintf(fp_out, "B ");

  /*** write width, height ***/
  fprintf(fp_out, "%.0f %.0f ", width, height);

  /*** write center coordinates ***/
  fprintf(fp_out, "%.0f %.0f; \n", 2*die_len - xc - 1.7e4, 2*die_len -
yc - 0.95e4);

}  /** end of function write_coords **/
/***-============================================================-***/
```

This file, mask_probe.c, defines the probe pad structures interspersed between the bond pad snake structures.

```
/***   mask_probe.c support file   ***/

/*** libraries ***/
#include <math.h>
#include <stdio.h>
```

```
#include <stdlib.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_bond.h"
#include "mask_probe.h"
#include "mask_alphnum.h"

/*** function listing **********

void repeat_probe_struct()

void define_probe_struct()

void define_probe_pads()

void define_probe_links()

void define_probe_strand()

double get_linewidth()

int width_chart()

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

/***==============================================================***/
void repeat_probe_struct(Point probe_loc, Point end,
                   FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                   int _orient, double line_len, int _set_num,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_strand_len,
                   double pr_str_overhang, double pr_link_len,
                   int _struct_type)
/***
Repeats the probe and line structure given the offset and orientation
up to the end-bounds marker.
***/
{
  int width_num=1,         /* incremented width variable for bins    */
      width_int;           /* width in integer format for labeling   */
  double width,            /* linewidth of line test structure       */
         tot_offset;       /* temp var to add total length of struct */
  Point probe_line_loc,    /* location of test line in probe struct  */
        text_loc;          /* location of text label of linewidth    */

  /*** for HORIZONTAL structures ***/
  if( _orient == HORIZ ) {

    /*** define initial horizontal probe test structures ***/
```

```
    define_probe_struct(probe_loc, fp_mask1, fp_mask3, HORIZ,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);
    probe_loc.x1 += pr_pad_len + pr_pad_spacing +
strand_width+pr_str_overhang;

    /*** until we reach end of spare region, repeat struct ***/
    while( (probe_loc.x1 + line_len) < end.x1 ) {

      /*** define line structure and text label ***/
      probe_line_loc.x1  = probe_loc.x1;
      probe_line_loc.y1  = probe_loc.y1 + pr_pad_len + pr_link_len;
      width = get_linewidth(width_num);
      define_centered_line(probe_line_loc, fp_mask1, HORIZ, line_len,
width);
      define_end_stops(probe_line_loc, fp_mask2, HORIZ, line_len);
  /*     text_loc.x1 = probe_loc.x1 + 10.0 - 2* pr_pad_len -
pr_pad_spacing; */
      text_loc.x1 = probe_loc.x1 - pr_str_overhang + pr_pad_len + 20.0;
      text_loc.y1 = probe_loc.y1 + 10.0;
      if( _set_num > 3 )
      parse_width(&text_loc, fp_mask3, HORIZ, width_num);
      width_num++;

      /*** define probe test structures ***/
      probe_loc.x1 += line_len - pr_pad_len + pr_str_overhang +
strand_width;
      define_probe_struct(probe_loc, fp_mask1, fp_mask3, HORIZ,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);

      /*** account for whether structures are joined or isolated ***/
      if( _struct_type == ISOLATED ) {
      /*** adjust coordinates for next probe structure ***/
      probe_loc.x1 += 2 * pr_pad_len + 2 * pr_pad_spacing;
      /*** if there is room to fit another test structure, draw it ***/
      if( (probe_loc.x1+ line_len) < end.x1)
        define_probe_struct(probe_loc, fp_mask1, fp_mask3, HORIZ,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);
      }
      /*** adjust coordinates for line test structure ***/
      probe_loc.x1 += pr_pad_len + pr_pad_spacing + strand_width;
      probe_loc.x1 += pr_str_overhang;
    }
  } else { /* VERT */

    /*** define intial vertical probe test structures ***/
    tot_offset  = line_len + 3 * pr_pad_len + pr_pad_spacing +
strand_width;
    tot_offset += pr_str_overhang;
    if( (probe_loc.y1 + tot_offset) < end.y1 )
```

```
      define_probe_struct(probe_loc, fp_mask1, fp_mask3, VERT,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);
    probe_loc.y1 += pr_pad_len + pr_pad_spacing +
strand_width+pr_str_overhang;

    /*** until we reach end of spare region, repeat struct ***/
    while( (probe_loc.y1 + line_len + 2*pr_pad_len) < end.y1 ) {

      /*** define line structure ***/
      probe_line_loc.x1 = probe_loc.x1 + pr_pad_len + pr_link_len;
      probe_line_loc.y1 = probe_loc.y1;
      width = get_linewidth(width_num);
      define_centered_line(probe_line_loc, fp_mask1, VERT, width,
line_len);
      define_end_stops(probe_line_loc, fp_mask2, VERT, line_len);
      probe_loc.y1 += line_len - pr_pad_len + pr_str_overhang +
strand_width;
      text_loc.x1 = probe_loc.x1 + 25.0;
      text_loc.y1 = probe_loc.y1 + 10.0 -  pr_pad_len - pr_pad_spacing;
      if( _struct_type != ISOLATED ) {
      if( _set_num > 3 )
        parse_width(&text_loc, fp_mask3, VERT, width_num);
      } else /* ISOLATED */
      if( _set_num > 4 )
        parse_width(&text_loc, fp_mask3, VERT, width_num);
      width_num++;

      /*** define probe test structures ***/
      define_probe_struct(probe_loc, fp_mask1, fp_mask3, VERT,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);

      /*** account for whether structures are joined or isolated ***/
      if( _struct_type == ISOLATED ) {
      /*** adjust coordinates for next probe struct ***/
      probe_loc.y1 += 2 * pr_pad_len + 2 * pr_pad_spacing;
      /*** if there is room to fit another test structure, draw it ***/
      if( (probe_loc.y1 + tot_offset) < end.y1)
        define_probe_struct(probe_loc, fp_mask1, fp_mask3, VERT,
pr_pad_len, pr_pad_spacing, strand_width, pr_strand_len,
pr_str_overhang, pr_link_len);
      }
      /*** adjust coordinates for line test structure ***/
      probe_loc.y1 += pr_pad_len + pr_pad_spacing + strand_width;
      probe_loc.y1 += pr_str_overhang;
    }
  }

}  /** End of function repeat_probe_struct() **/

/***-------------------------------------------------------------***/
void define_probe_struct(Point _offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double pr_pad_len,
```

```
                   double pr_pad_spacing,
                   double strand_width, double pr_strand_len,
                   double pr_str_overhang, double pr_link_len)
/***
Defines the current, voltage pad structure without test lines
given the offset and orientation.
***/
{
  /*** define probe pads ***/
  define_probe_pads(_offset, fp_mask1, fp_mask3, _orient, pr_pad_len,
pr_pad_spacing);

  /*** define probe links ***/
  define_probe_links(_offset, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_link_len);

  /*** define probe strand ***/
  define_probe_strand(_offset, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_strand_len, pr_str_overhang,
pr_link_len);

}  /** End of function define_probe_struct() **/

/***---------------------------------------------------------------***/
void define_probe_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double _pr_pad_len, double
_pr_pad_spacing)
/***
Determine coordinates of the five pads and write them to
the output file.
***/
{
  Point padi,                          /* current pad                 */
        padv,                          /* voltage pad                 */
        coord;                         /* holds true coordinate data  */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    padi.x1 = 0.0;
    padi.y1 = 0.0;
    padv.x1 = 0.0;
    padv.y1 = _pr_pad_len + _pr_pad_spacing;
  } else {   /* HORIZ */
    padi.x1 = 0.0;
    padi.y1 = 0.0;
    padv.x1 = _pr_pad_len + _pr_pad_spacing;
    padv.y1 = 0.0;
  }

  /*** step through each pad and determine coordinates with offsets
***/

    /*** add offsets, determine x2,y2 coordinates padi ***/
    coord.x1 = padi.x1 + offset.x1;
```

```
    coord.y1 = padi.y1 + offset.y1;
    coord.x2 = coord.x1 + _pr_pad_len;
    coord.y2 = coord.y1 + _pr_pad_len;

    /*** write data to file ***/
    write_coords(fp_mask1, coord, 1);
    write_coords(fp_mask3, coord, 1);

    /*** add offsets, determine x2,y2 coordinates padv ***/
    coord.x1 = padv.x1 + offset.x1;
    coord.y1 = padv.y1 + offset.y1;
    coord.x2 = coord.x1 + _pr_pad_len;
    coord.y2 = coord.y1 + _pr_pad_len;

    /*** write data to file ***/
    write_coords(fp_mask1, coord, 1);
    write_coords(fp_mask3, coord, 1);

}  /** end of function define_probe_pads() **/

/***------------------------------------------------------------------***/
void define_probe_links(Point offset, FILE *fp_out, int _orient,
                  double _pr_pad_len, double _pr_pad_spacing,
                  double _strand_width, double _pr_link_len)
/***
Determine coordinates of the five links and write them to
the output file.
***/
{
  Point link_pi,                      /* coords for link to I pad    */
        link_pv,                      /* coords for link to V pad    */
        coord;                        /* holds true coordinate data  */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    link_pi.x1  = _pr_pad_len;
    link_pi.y1  = _pr_pad_len - _strand_width;
    link_pv.x1  = link_pi.x1;
    link_pv.y1  = link_pi.y1 + _strand_width + _pr_pad_spacing;
  } else {   /* HORIZ */
    link_pi.x1  = _pr_pad_len - _strand_width;
    link_pi.y1  = _pr_pad_len;
    link_pv.x1  = link_pi.x1 + _strand_width + _pr_pad_spacing;
    link_pv.y1  = link_pi.y1;
  }

  /*** step through each link and determine coordinates with offsets
***/

  /*** add offsets, determine x2,y2 coordinates, link_pi ***/
  coord.x1 = link_pi.x1 + offset.x1;
  coord.y1 = link_pi.y1 + offset.y1;
  if( _orient == VERT ) {
    coord.x2 = coord.x1 + _pr_link_len;
```

```
    coord.y2 = coord.y1 + _strand_width;
  } else {   /* HORIZ */
    coord.x2 = coord.x1 + _strand_width;
    coord.y2 = coord.y1 + _pr_link_len;
  }

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

  /*** add offsets, determine x2,y2 coordinates, link_pv ***/
  coord.x1 = link_pv.x1 + offset.x1;
  coord.y1 = link_pv.y1 + offset.y1;
  if( _orient == VERT ) {
    coord.x2 = coord.x1 + _pr_link_len;
    coord.y2 = coord.y1 + _strand_width;
  } else {   /* HORIZ */
    coord.x2 = coord.x1 + _strand_width;
    coord.y2 = coord.y1 + _pr_link_len;
  }

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

}  /** end of function define_probe_links() **/

/***------------------------------------------------------------***/
void define_probe_strand(Point offset, FILE *fp_out, int _orient,
                   double _pr_pad_len, double _pr_pad_spacing,
                   double _strand_width, double _pr_strand_len,
                   double _pr_str_overhang, double _pr_link_len)
/***
Determine coordinates of the five links and write them to
the output file.
***/
{
  int i;                           /* index to step thru array    */
  Point pr_strand,
        coord;                     /* holds true coordinate data  */

  /*** write initial coords ***/
  if( _orient == VERT ) {
    pr_strand.x1  = _pr_pad_len + _pr_link_len;
    pr_strand.y1  = _pr_pad_len - _strand_width - _pr_str_overhang;
  } else {   /* HORIZ */
    pr_strand.x1  = _pr_pad_len - _strand_width - _pr_str_overhang;
    pr_strand.y1  = _pr_pad_len + _pr_link_len;
  }

  /*** adjust for offsets ***/
  pr_strand.x1  += offset.x1;
  pr_strand.y1  += offset.y1;

  /*** write opposing corner coordinates ***/
  if( _orient == VERT ) {
```

```
    pr_strand.x2  = pr_strand.x1 + _strand_width;
    pr_strand.y2  = pr_strand.y1 + _pr_strand_len;
  } else {   /* HORIZ */
    pr_strand.x2  = pr_strand.x1 + _pr_strand_len;
    pr_strand.y2  = pr_strand.y1 + _strand_width;
  }

  /*** write data to file for each strand ***/
  write_coords(fp_out, pr_strand, 1);

}  /** end of function define_probe_strands() **/

/***------------------------------------------------------------------***/
double get_linewidth(int width_bin_num)
/***
Receives linewidth bin label and is assigned a linewidth based on
a table given below.  Returns this linewidth in the function.
***/
{
  double width;                               /* width for given bin */

  /*** go to width chart and recast int to double ***/
  width = (double) (width_chart(width_bin_num));

  /*** return linewidth ***/
  return(width);

}    /** end of function get_linewidth **/

/***------------------------------------------------------------------***/
int width_chart(int width_bin_num)
/***
Takes the width bin and gives the corresponding linewidth as an
integer.  It is up to the calling to function to convert this
to a double if necessary.
***/
{
  int width;                                  /* width for given bin */
  int width_bin,                              /* width num after mod */
      max_bins=6;                             /* max # of width bins */

  /*** artificial mod function ***/
  width_bin = width_bin_num;
  if( width_bin > max_bins )
    while( width_bin > max_bins )
      width_bin -= max_bins;

  if( width_bin > 6 ) {
    printf("Error with width bin");
    PRINT_INT(width_bin);
  }

  /*** assign linewidth to bin ***/
  if( width_bin == 1 )
```

```
    width = 2;
  if( width_bin == 2 )
    width = 4;
  if( width_bin == 3 )
    width = 6;
  if( width_bin == 4 )
    width = 8;
  if( width_bin == 5 )
    width = 10;
  if( width_bin == 6 )
    width = 1;

  /*** return linewidth ***/
  return(width);

}    /** end of function width_chart **/

/***-============================================================-***/
```

The mask_short.c file is reponsible for defining the 25 and 50 micron probe pad structures along the center vertical of the die.

```
/***   mask_short.c support file   ***/

/*** libraries ***/
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/*** define data structures ***/
#include "mask.h"

/*** function prototypes ***/
#include "mask_bond.h"
#include "mask_short.h"
#include "mask_alphnum.h"

/*** function listing **********

void repeat_short_struct()

void define_short_struct()

void define_short_pads()

void define_short_links()

void define_short_strand()

double get_linewidth()

int width_chart()
```

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

/***=================================================================***/
void repeat_short_struct(Point short_loc, Point end,
                   FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                   int _orient, double line_len, int _set_num,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_link_len,
                   int _struct_type)
/***
Repeats the short and line structure given the offset and orientation
up to the end-bounds marker.
***/
{
  int width_num=1,         /* incremented width variable for bins    */
      width_int;           /* width in integer format for labeling   */
  double width;            /* linewidth of line test structure       */
  Point short_line_loc,    /* location of test line in short struct  */
        text_loc;          /* location of text label of linewidth    */

  /*** until we reach end of spare region, repeat struct ***/
  while( (short_loc.y1 + line_len + 4*pr_pad_len) < end.y1 ) {

    /*** define initial vertical short test structures ***/
    define_short_struct(short_loc, fp_mask1, fp_mask3, VERT, line_len,
pr_pad_len, pr_pad_spacing, strand_width, pr_link_len);
    short_loc.y1 += 2 * pr_pad_len + pr_pad_spacing;

    /*** define line structure ***/
    short_line_loc.x1 = short_loc.x1 + pr_pad_len + pr_link_len;
    short_line_loc.y1 = short_loc.y1;
    width = get_linewidth(width_num);
    define_centered_line(short_line_loc, fp_mask1, VERT, width,
line_len);
    define_end_stops(short_line_loc, fp_mask2, VERT, line_len);
    short_loc.y1 += line_len;

    /*** adjust coordinates to end of this line test structure ***/
    short_loc.y1 += 2 * pr_pad_len + pr_pad_spacing + strand_width;

    /*** write text label for last text structure ***/
    text_loc.x1 = short_loc.x1 - 10.0;
    text_loc.y1 = short_loc.y1 + 10.0;
    if( _set_num > 3 ) {
      define_4_letter(text_loc, fp_mask3, HORIZ, 3, 0);
      text_loc.x1 += HLET_SPAC;
      parse_width(&text_loc, fp_mask3, HORIZ, width_num);
    }
    width_num++;

    /*** adjust coordinates for next line structure ***/
    short_loc.y1 += 65.0;
  }
```

```
}  /** End of function repeat_short_struct() **/

/***---------------------------------------------------------------***/
void define_short_struct(Point _offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double line_len,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_link_len)
/***
Defines the current, voltage pad structure without test lines
given the offset and orientation.
***/
{
  Point sec_set;                       /* second set of pads         */

  /*** define short pads ***/
  sec_set.x1 = _offset.x1;
  sec_set.y1 = _offset.y1;
  define_short_pads(sec_set, fp_mask1, fp_mask3, _orient, pr_pad_len,
pr_pad_spacing);

  /*** set coords and define other set of pads ***/
  sec_set.y1 = _offset.y1 + 2 * pr_pad_len + pr_pad_spacing + line_len;
  define_short_pads(sec_set, fp_mask1, fp_mask3, _orient, pr_pad_len,
pr_pad_spacing);

  /*** define short links ***/
  sec_set.y1 = _offset.y1 + pr_pad_len - strand_width;
  define_short_links(sec_set, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_link_len);

  /*** set coords and define other set of links ***/
  sec_set.y1  = _offset.y1 + 3 * pr_pad_len + pr_pad_spacing +
line_len;
  sec_set.y1 -= strand_width;
  define_short_links(sec_set, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_link_len);

  /*** define short strand ***/
  sec_set.y1 = _offset.y1 + pr_pad_len - strand_width;
  define_short_strand(sec_set, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_link_len);

  /*** set coords and define other strand ***/
  sec_set.y1 = _offset.y1 + 2 * pr_pad_len + pr_pad_spacing + line_len;
  define_short_strand(sec_set, fp_mask1, _orient, pr_pad_len,
pr_pad_spacing, strand_width, pr_link_len);


}  /** End of function define_short_struct() **/

/***---------------------------------------------------------------***/
void define_short_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3,
```

```
                    int _orient, double _pr_pad_len, double
_pr_pad_spacing)
/***
Determine coordinates of the five pads and write them to
the output file.
***/
{
  Point padi,                        /* current pad                */
        padv,                        /* voltage pad                */
        coord;                       /* holds true coordinate data  */

  /*** write initial coords ***/
  padi.x1 = 0.0;
  padi.y1 = 0.0;
  padv.x1 = 0.0;
  padv.y1 = _pr_pad_len + _pr_pad_spacing;

  /*** step through each pad and determine coordinates with offsets
***/

  /*** add offsets, determine x2,y2 coordinates padi ***/
  coord.x1 = padi.x1 + offset.x1;
  coord.y1 = padi.y1 + offset.y1;
  coord.x2 = coord.x1 + _pr_pad_len;
  coord.y2 = coord.y1 + _pr_pad_len;

  /*** write data to file ***/
  write_coords(fp_mask1, coord, 1);
  write_coords(fp_mask3, coord, 1);

  /*** add offsets, determine x2,y2 coordinates padv ***/
  coord.x1 = padv.x1 + offset.x1;
  coord.y1 = padv.y1 + offset.y1;
  coord.x2 = coord.x1 + _pr_pad_len;
  coord.y2 = coord.y1 + _pr_pad_len;

  /*** write data to file ***/
  write_coords(fp_mask1, coord, 1);
  write_coords(fp_mask3, coord, 1);

}  /** end of function define_short_pads() **/

/***-------------------------------------------------------------***/
void define_short_links(Point offset, FILE *fp_out, int _orient,
                  double _pr_pad_len, double _pr_pad_spacing,
                  double _strand_width, double _pr_link_len)
/***
Determine coordinates of the five links and write them to
the output file.
***/
{
  Point link_pi,                     /* coords for link to I pad    */
        link_pv,                     /* coords for link to V pad    */
        coord;                       /* holds true coordinate data  */
```

```
  /*** write initial coords ***/
  link_pi.x1  = _pr_pad_len;
  link_pi.y1  = 0.0;
  link_pv.x1  = link_pi.x1;
  link_pv.y1  = link_pi.y1 + _strand_width + _pr_pad_spacing;

  /*** step through each link and determine coordinates with offsets
***/

  /*** add offsets, determine x2,y2 coordinates, link_pi ***/
  coord.x1 = link_pi.x1 + offset.x1;
  coord.y1 = link_pi.y1 + offset.y1;
  coord.x2 = coord.x1 + _pr_link_len;
  coord.y2 = coord.y1 + _strand_width;

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

  /*** add offsets, determine x2,y2 coordinates, link_pv ***/
  coord.x1 = link_pv.x1 + offset.x1;
  coord.y1 = link_pv.y1 + offset.y1;
  coord.x2 = coord.x1 + _pr_link_len;
  coord.y2 = coord.y1 + _strand_width;

  /*** write data to file ***/
  write_coords(fp_out, coord, 1);

}  /** end of function define_short_links() **/

/***--------------------------------------------------------------***/
void define_short_strand(Point offset, FILE *fp_out, int _orient,
                   double _pr_pad_len, double _pr_pad_spacing,
                   double _strand_width, double _pr_link_len)
/***
Determine coordinates of the five links and write them to
the output file.
***/
{
  Point pr_strand,
        coord;                        /* holds true coordinate data  */

  /*** write initial coords ***/
  pr_strand.x1  = _pr_pad_len + _pr_link_len;
  pr_strand.y1  = 0.0;

  /*** adjust for offsets ***/
  pr_strand.x1  += offset.x1;
  pr_strand.y1  += offset.y1;

  /*** write opposing corner coordinates ***/
  pr_strand.x2  = pr_strand.x1 + _strand_width;
  pr_strand.y2  = pr_strand.y1 + _strand_width + _pr_pad_spacing+
_pr_pad_len;
```

```
  /*** write data to file for each strand ***/
  write_coords(fp_out, pr_strand, 1);

}  /** end of function define_short_strand() **/

/***-============================================================-***/
```

All of the following files comprise the header files associated with the mask program files listed above.

```
/*** mask.h support file ***/

/***
Contains data structures used throughout mask.h
***/

/*** constants ***/
#define MAXLEN    1E5
#define FILELEN   75
#define TEXTLEN   90
#define NUMLEN    10

/*** definitions ***/
#define HORIZ    0
#define VERT     1
#define YES      1
#define NO       0
#define JOINED   1
#define ISOLATED 0

/*** critical dimensions ***/
#define HALF_DIE_LEN 5.05e5      /* twice of this gives true die len */
#define TEXT_WIDTH       25      /* width  of LED text in gridpts    */
#define TEXT_HEIGHT      45      /* height of LED text in gridpts    */
#define HLET_SPAC 35.0           /* width  and inter-letter Hspacing */
#define VLET_SPAC 55.0           /* height and inter-letter Vspacing */
#define RUN_WIDTH 10.0

/*** define Point data structure ***/
  typedef struct {
    double x1;            /* bottom left x-coordinate of point      */
    double y1;            /* bottom left y-coordinate of point      */
    double x2;            /* upper right x-coordinate of point      */
    double y2;            /* upper right y-coordinate of point      */
} Point;

typedef struct Point Pointnode; /* defines 2 coordinate Point struct */

/*** debugging functions ***/
#define PRINT_ID(x)     printf(#x "->id = %i\n\n", x->id)
#define PRINT_NUM(x)    printf(#x " = %e\n\n", x)
```

```
#define PRINT_INT(x)    printf(#x " = %i\n\n", x)
#define TEST(i)         printf("\n Made it here to %i ! \n\n", i)



/***  mask_align.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in mask_align.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef MASK_ALIGN_H
#define MASK_ALIGN_H

/*** function prototypes ***/

void draw_red_cross(Point loc, FILE *fp_out, double width, int _level);

void draw_bigger_cross(Point loc, FILE *fp_out, double width, int
_level);

void draw_diag_squares(Point loc, FILE *fp_out, double width, int
_level);

void draw_square(Point loc, FILE *fp_out, double width, int _level);

void draw_bar_code(Point loc, FILE *fp_out, int _level);

void draw_rectangle(Point loc, FILE *fp_out, int _level,
                double width, double height);

#endif  /* MASK_ALIGN_H */



/***  mask_alphnum.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in mask_alphnum.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef MASK_ALPHNUM_H
#define MASK_ALPHNUM_H

/*** function prototypes ***/

void write_andleigh(Point _alphnum_loc, FILE *fp_out);
```

```
void write_emsim(Point _alphnum_loc, FILE *fp_out);

void write_bites(Point _alphnum_loc, FILE *fp_out);

void write_microsoft(Point _alphnum_loc, FILE *fp_out);

void parse_length(Point *alphnum_loc, FILE *fp_out, int _orient,
              double _length);

void parse_width(Point *alphnum_loc, FILE *fp_out, int _orient,
             double _width_bin);

void parse_align_num(Point _alphnum_loc, FILE *fp_out, int _level,
                 int _arrow_dir);

void define_4_letter(Point alphnum_loc, FILE *fp_out, int _orient,
                 int _letter_struct, int displ);

void define_4_num(Point alphnum_loc, FILE *fp_out, int _orient,
              int _num, int displ);

void define_letter(Point alphnum_loc, FILE *fp_out, int _orient,
               int _letter_struct);

void define_num(Point alphnum_loc, FILE *fp_out, int _orient, int
_num);

void write_alphnum(Point alphnum_loc, FILE *fp_out, int led_num);

void write_led_bar(FILE *fp_out, Point p_coord, int _level);


#endif  /* MASK_ALPHNUM_H */


/***  mask_bond.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in mask_bond.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef MASK_BOND_H
#define MASK_BOND_H

/*** function prototypes ***/

void define_bond_struct(Point _offset,
                  FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                  int _orient, double pad_len,
                  double pad_spacing,double strand_width,
                  double link_len_pi, double link_len_pv,
```

```
                  double strand_len_li, double strand_len_lv,
                  double link_pi_offset, double lv_overhang);

void define_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3,
             int _orient,
             double _pad_len, double _pad_spacing,
             double _strand_width,
             double _strand_len_li, double _link_len_pi,
             double _link_pi_offset, double _lv_overhang);

void define_links(Point offset, FILE *fp_out, int _orient,
              double _pad_len, double _pad_spacing,
              double _strand_width, double _strand_len_li,
              double _link_len_pi, double _link_len_pv,
              double _link_pi_offset, double _lv_overhang);

void define_strands(Point offset, FILE *fp_out, int _orient,
                double _pad_len, double _pad_spacing,
                double _strand_width,
                double _link_len_pi, double _link_len_pv,
                double _strand_len_li, double _strand_len_lv,
                double _link_pi_offset, double _lv_overhang);

void define_centered_line(Point loc, FILE *fp_out, int _orient,
                    double _line_len, double _line_width);

void define_line(Point loc, FILE *fp_out,
             double _line_len, double _line_width);

void define_L_shape(Point left, Point right, FILE *fp_out,
                double _strand_width);

void define_end_stops(Point line, FILE *fp_out, int _orient,
                  double length);

void write_coords(FILE *fp_out, Point p_coord, int _level);

#endif  /* MASK_BOND_H */


/***  mask_probe.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in mask_probe.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef MASK_PROBE_H
#define MASK_PROBE_H

/*** function prototypes ***/
```

```
void repeat_probe_struct(Point probe_loc, Point end,
                   FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                   int _orient, double line_len, int _set_num,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_strand_len,
                   double pr_str_overhang, double pr_link_len,
                   int _joined);

void define_probe_struct(Point _offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double pr_pad_len,
                   double pr_pad_spacing,
                   double strand_width, double pr_strand_len,
                   double pr_str_overhang, double pr_link_len);

void define_probe_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double _pr_pad_len,
                   double _pr_pad_spacing);

void define_probe_links(Point offset, FILE *fp_out, int _orient,
                  double _pr_pad_len, double _pr_pad_spacing,
                  double _strand_width, double _pr_link_len);

void define_probe_strand(Point offset, FILE *fp_out, int _orient,
                   double _pr_pad_len, double _pr_pad_spacing,
                   double _strand_width, double _pr_strand_len,
                   double _pr_str_overhang, double _pr_link_len);

double get_linewidth(int width_bin_num);

int width_chart(int width_bin);

#endif  /* MASK_PROBE_H */


/***  mask_short.h support file ***/

/***
 * This file contains all the function prototypes
 * for all the functions used in mask_short.c
 * They are listed in the order in which they appear
 * in this files
***/

#ifndef MASK_SHORT_H
#define MASK_SHORT_H

/*** function prototypes ***/

void repeat_short_struct(Point short_loc, Point end,
                   FILE *fp_mask1, FILE *fp_mask2, FILE *fp_mask3,
                   int _orient, double line_len, int _set_num,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_link_len,
                   int _joined);
```

```
void define_short_struct(Point _offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double line_len,
                   double pr_pad_len, double pr_pad_spacing,
                   double strand_width, double pr_link_len);

void define_short_pads(Point offset, FILE *fp_mask1, FILE *fp_mask3,
                   int _orient, double _pr_pad_len,
                   double _pr_pad_spacing);

void define_short_links(Point offset, FILE *fp_out, int _orient,
                  double _pr_pad_len, double _pr_pad_spacing,
                  double _strand_width, double _pr_link_len);

void define_short_strand(Point offset, FILE *fp_out, int _orient,
                   double _pr_pad_len, double _pr_pad_spacing,
                   double _strand_width, double _pr_link_len);

#endif  /* MASK_SHORT_H */
```

Appendix C

Analytical Methodologies

This appendix contains the derivation of an analytical expression for the construction of failure mechanism maps for interconnects in which failure occurs by void growth to a specific void volume.

C.1. Void Saturation Calculation

After a void nucleates, the void will continue to grow until the compressive stress opposes the electromigration driving force, reaching a steady-state and void growth saturation with a linear increase in the compressive stress along the line. Since the electromigration driving force is given by eρjz* and is opposed by the back-stress with a force $\Omega \frac{\partial \sigma}{\partial x}$ (where x is the coordinate along the line length), the stress can be represented as follows:

$$\int_{L_v}^{L} \frac{e\rho jz^*}{\Omega} \partial x = \int_0^{\sigma} \partial \sigma \quad . \tag{C1}$$

Integration yields an expression for stress as a function of position given by

$$\sigma(x) = \frac{e\rho jz^*}{\Omega}\left(x - L_v^{sat}\right) , \tag{C2}$$

where L_v^{sat} is the saturation void length. The volumetric strain is related to the saturation void size and may be determined by dividing the total stress present in the line by the appropriate modulus as given by:

$$L_v^{sat} = \frac{1}{B} \int_{L_v^{sat}}^{L} \sigma(x) \partial x \, . \tag{C3}$$

After substituting Eq. C2 into Eq. C3 and integrating, the following expression for the void saturation length may be determined:

$$L_v^{sat} = \frac{e \rho j z^*}{2B\Omega} \left(L - L_v^{sat} \right)^2 \, . \tag{C4}$$

Rearranging Eq. C4 results in the following relationship between the line length, the saturated void length, and the current density:

$$L = \sqrt{\frac{2 L_v^{sat} B\Omega}{e \rho j z^*}} + L_v^{sat} \, . \tag{C5}$$

Equation C5 represents an expression for void saturation based on a void volume failure condition. For resistance-based failure conditions, Eq. 4.4 can be substituted into Eq. C5, which after rearranging, results in a relationship between the line length, the current density, and $\Delta R/R_o$:

$$L = \frac{2 M_c \frac{\Delta R}{R_o} B\Omega}{e \rho j z^* \left(1 - M_c \frac{\Delta R}{R_o} \right)^2} \, . \tag{C6}$$

This expression reduces to the expression of Suo [Suo 98] if the squared term is approximated as equal to unity. Equation C6 will be used in Section 5.4.

C.2. Grain Orientation Effects on Cu Reliability

As discussed in Chap. 6, in Cu interconnects, the dominant interface for diffusivity is the Cu-Si_3N_4 interface. It is believed that the diffusivity at an interface will be a strong function of the orientation of the grain. A model for the linewidth-dependent grain structure statistics in bamboo interconnects is presented. It will then be shown, using MIT/EmSim, that grain orientation-dependent interface diffusivities constitute a likely mechanism for the variabilities in lifetimes observed experimentally.

It has been shown elsewhere [Fay 99] that 2D normal grain growth in thin films leads to a uniquely defined grain structure, evolving in a geometrically and statistically self-similar fashion, with an average grain area increasing linearly with both time and the grain boundary mobility μ (assuming μ is constant and uniform). Under these conditions, the grain structure evolution in lines of width w, etched from films with an average grain area much smaller than w^2, will obey geometric and kinetic scaling [Wal 92, Fay_2 00]: the duration of the transformation will scale with w^2/μ and the resulting bamboo grain structure statistics in lines with different widths will only differ due to geometric magnification. In particular, we expect the distribution of the bamboo grain lengths normalized by the line width, (L/w), to be width-independent. Figure C-1 shows that such a distribution is well fit by a lognormal distribution function. When plotting ln(L/w) as a function of Φ^{-1}(F(L/w)), where F(L/w) is the proportion of grains with normalized length smaller than L/w and Φ is the Gaussian function, data that falls on a straight line fits a lognormal distribution. For the simulated lines with different widths, the curves overlap, as expected with the geometric scaling outlined above. The unique lognormal distribution function that fits these results is characterized by a median value of 1.9 and a

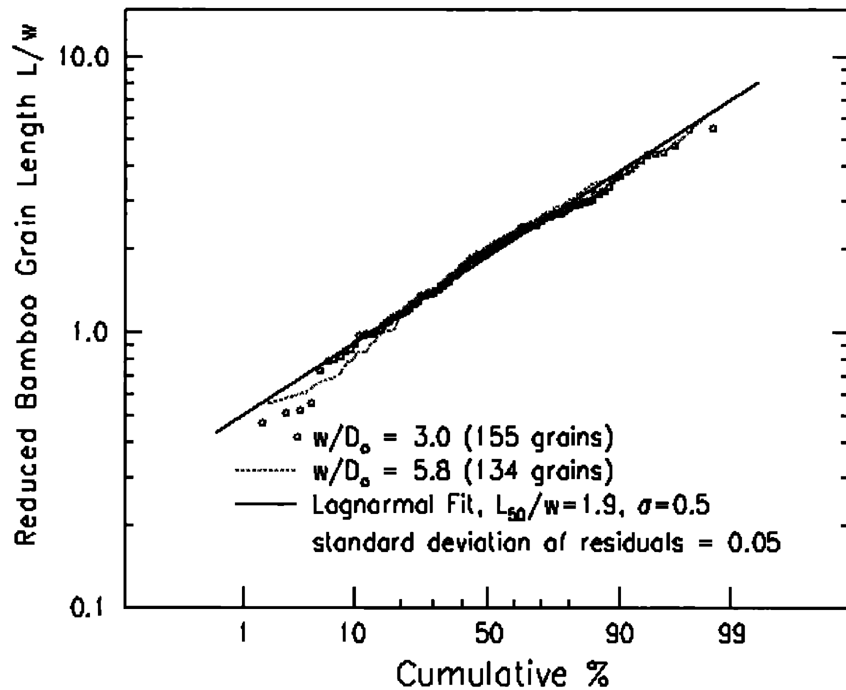


Figure C-1. Lognormal plot of normalized bamboo grain length, L/w, distributions in bamboo structures resulting from post-patterning annealing of polygranular strips with different widths. The overlapping curves for different w/D_0 show that the distribution of L/w is line-width-independent. Solid lines represent the best-fitting lognormal distribution.

lognormal deviation σ = 0.5. The average bamboo grain length associated with such a distribution is about twice (2.1 times) the line width.

Annealing-induced grain structure evolution in metal interconnects is not only governed by curvature-driven growth. It also depends on a number of other factors, such as elastic anisotropy [Car 96], variable grain boundary energy [Fro 94], surface grooving or other boundary pinning effects [Fro 90, Fro_2 94]. However, the 2D normal growth simulation and the model to be presented here provide a good first order evaluation of the resulting bamboo grain structure. The use of the compact model discussed above (based on the generation of structures using the fitted lognormal distribution function) or the use of GGSim enables the direct generation of appropriately varying grain lengths for

simulations of electromigration in lines with bamboo structures. MIT/EmSim is then used to determine failure times associated with void nucleation or resistance failures.

To simulate the effects of grain-structure-related variations in the interface diffusion, which is the dominant diffusion mechanism in the case of bamboo interconnection [Joo 97, Sri 98], it is assumed that the interface diffusivity of Cu varies with grain orientation in accordance with a Read-Shockley model [Rea 54], and is fitted using published values for interface diffusion on (100) and (111) Cu interfaces [Cho 62], which are expected to be the minimum and the maximum diffusivities, respectively. D_{111} and D_{100} were evaluated with Arrhenius relationships with activation energies of 2.10 eV and 2.26 eV respectively, and pre-exponential factors of 1.8 m^2/s and 4.0 m^2/s respectively [Cho 62]. This description can also be implemented in MIT/EmSim so that interface diffusivities can be assigned to bamboo grains, assuming a random grain orientation, according to a Read-Shockley function, as depicted in Fig. C-2.

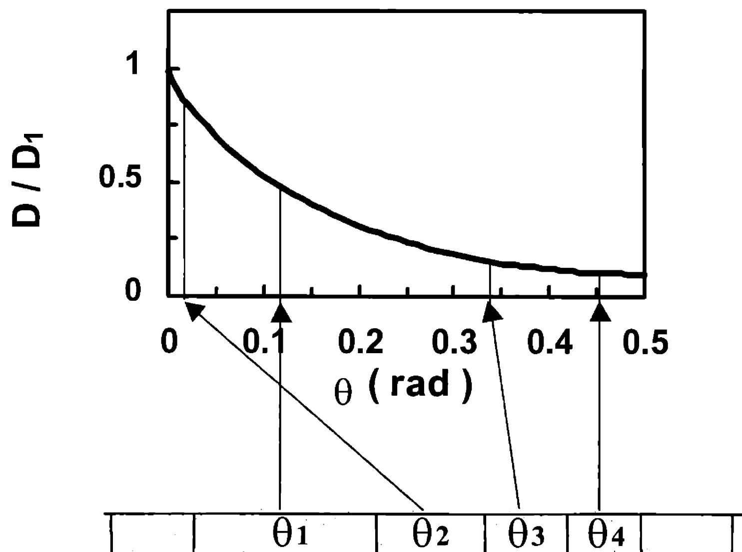


Figure C-2. Distribution of interface diffusivities as a function of bamboo grain orientation. The grain orientation is assigned randomly, and the diffusivity for a given orientation θ is given by $D=D_{fac}(1+ A\,\theta \ln(\theta) - B\,\theta)$ where $D_{fac} = 2.2*10^{-16}$ m^2/s at 392°C, A = 1.8, and B=0.55.

When simulations of the reliability of bamboo structures were carried out using these models, results such as those shown in Fig. C-3 were obtained. Figure C-3 is a lognormal plot of EM-induced failure times in a population of 10 simulated bamboo lines with a width of 1.4 μm and a length of 300 μm. A current density of $2*10^6$ A/cm^2 and a temperature of 392°C were used. The lifetime variations observed in the simulations are very similar to those observed in experiments [Hu 97]. Assuming a critical stress for failure of 300 MPa, we obtained a lognormal deviation in the times to failure of about 0.3. This important result supports the expectation that lifetime variations in lines for which interface diffusion is dominant (e.g. Cu-lines or Al-lines with bamboo structures), are the result of grain-structure related variations in interface diffusivities, and the associated atomic flux divergences.

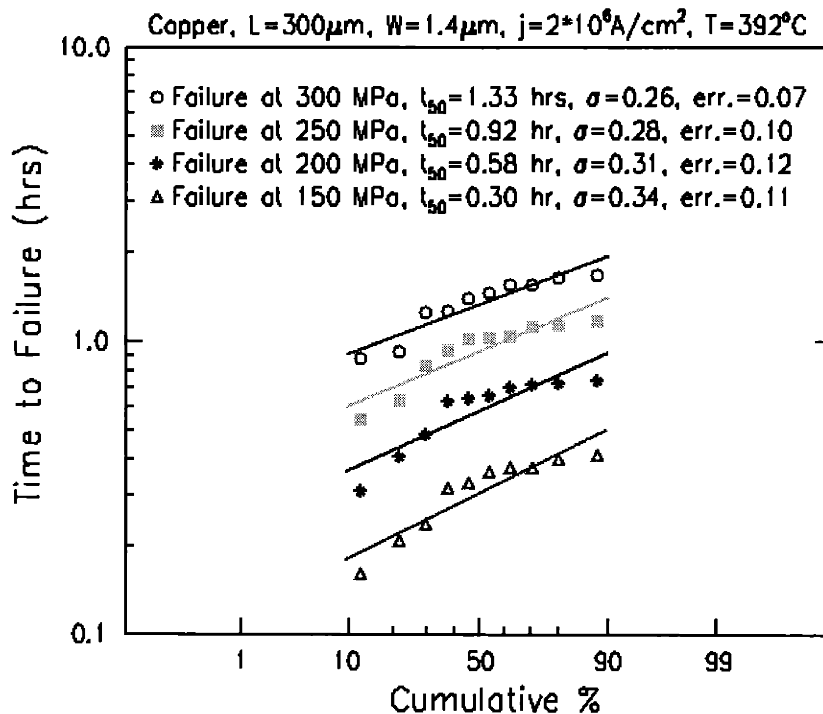


Figure C-3. Lognormal plot showing the simulated variations in lifetimes resulting from interface diffusivity variations in lines with bamboo grain structures.

In summary, a model for grain length statistics in bamboo interconnects with structures that have evolved from an initially polygranular structure has been presented. The average grain length scales with the line width, and the deviation in normalized length is independent of line-width. Assuming grain-orientation-dependent interface diffusivities allowed us to successfully simulate the experimentally observed variations in EM-lifetimes in bamboo Cu interconnects.

REFERENCES

[Aga 72] B.N. Agarwala, B. Patnaik, and R. Schnitzel. *J. Appl. Phys.* **43**, 1487 (1972).

[Ame 70] I. Ames, F.M. d'Heurle, and R.E. Horstmann. *IBM J. Res. Dev.* **14**, 461 (1970).

[And 98] V. K. Andleigh, Y. J. Park, and C. V. Thompson. *SRC TechCon'98 Symp. Proc.* Paper 8.4, Las Vegas, NV, (Sept. 1998).

[And 99] V. K. Andleigh, Y. J. Park, and C. V. Thompson. *Mater. Res. Soc. Symp. Proc.* **563**, 59 (1999).

[And$_2$ 99] V. K. Andleigh, V. T. Srikar, Y.-J. Park, and C. V. Thompson. *J. Appl. Phys.* **86**, 6737 (1999).

[Ari 94] Y. Arita, N. Awaya, K. Ohno, and M. Sato. *MRS Bulletin.* **19**, 68 (1994).

[Arn 99] L. Arnaud, G. Reimbold, and P. Waltz. *Microelectron. Reliab.* **39**, 773 (1999).

[Arz 94] E. Arzt, O. Kraft, W. D. Nix, and J. E. Sanchez. *J. Appl. Phys.* **76**, 1563 (1994).

[Ata 94] E. M. Atakov, J. J. Clement, and B. Miner. *Int. Reliab. Phys. Symp.* **32**, 231 (1994).

[Bla 69] J. R. Black. *IEEE Trans. Electron Devices.* **ED-16**, 338 (1969).

[Ble 66] I.A. Blech and H. Sello. *Phys. Failure in Electronics.* **5**, 496 (1966).

[Ble 76] I. A. Blech. *J. Appl. Phys.* **47**, 1203 (1976).

[Ble$_2$ 76] I. A. Blech and C. Herring. *Appl. Phys. Lett.* **29**, 131 (1976).

[Ble 77] I. A. Blech. *J. Appl. Phys.*, **48**, p.43 (1977).

[Boh 96] M. T. Bohr. *Solid State Technol.* **39**, 106 (1996).

[Bor 92] P. Bϕrgesen, M. A. Korhonen, and C.-Y. Li. *Thin Solid Films.* **220**, 8 (1992).

[Bro 95] D.D. Brown, J.E. Sanchez, Jr., M.A. Korhonen, and C.-Y. Li. *Appl. Phys. Lett.* **67**, 439 (1995).

[Car 96] R. Carel, C.V. Thompson, and H.J. Frost. *Acta Mat.* **44**, 2479 (1996).

[Cho 62] J.Y. Choi and P.G. Shewmon. *Trans. AIME.* **224**, 589 (1962).

[Cho 89] J. Cho, C.V. Thompson. *Appl. Phys. Lett.* **54**, 2577 (1989).

[Cle 94] J.J. Clement, C.V. Thompson, and A. Enver. *Mater. Res. Soc. Symp. Proc.* **338**, 353 (1994).

[Cle 95] J. J. Clement and C. V. Thompson. *J. Appl. Phys.* **78**, 900 (1995).

[Cle 97] J. J. Clement and T. S. Sriram, Digital Equipment Corporation, personal communication (1997).

[Cle_2 97] J.J. Clement. *J. Appl. Phys.* **82**, 5991 (1997).

[Dek 97] J.P. Dekker, A. Lodder, and J. van Ek. *Phys. Rev. B.* **56**, 167 (1997).

[Dhe 78] F.J. d'Heurle and P.S. Ho. Thin Films -- Interdiffusion and Reactions, edited by M. Poate, K.N. Tu, and J.W. Mayer. Electrochemical Society. Wiley, New York, 1978.

[Dwy 96] V.M. Dwyer, M.J. Kearney, and P.C. Bressloff. *J. Appl. Phys.* **80**, 3792 (1996).

[Ems 96] V. K. Andleigh, W. Fayad, M. Verminski, and C. V. Thompson. EmSim-Web, http://nirvana.mit.edu/emsim/index.html (1996).

[Fay 98] W. Fayad, V. K. Andleigh, and C.V. Thompson. *Mat. Rel. in Microelec VIII. Symp. Mater. Res. Soc.* **516**, 159 (1998).

[Fay 99] W. Fayad, H.J. Frost, and C.V. Thompson. *Scripta Mat.* **40**, 1199 (1999).

[Fay 00] W.R. Fayad. Ph.D. Thesis. Massachusetts Institute of Technology (2000).

[Fay_2 00] W.R. Fayad, M.J. Kobrinsky, and C.V. Thompson. *Phys. Rev. B.* **62**, 5221 (2000).

[Fay 01] W.R. Fayad, V.K. Andleigh, and C.V. Thompson. *J. Mater. Res.* **16**, in publication (2001).

[Fea 93] D. R. Fear, J. R. Michael, and A. D. Romig, Jr.. *Mat. Res. Soc. Proc.* **309**, 359 (1993).

[Fie 97] D. P. Field, J. E. Sanchez, Jr., P. R. Besser, and D. J. Dingley. *J. Appl. Phys.* **82**, 2383 (1997).

[Fil 95] R.G. Filippi, G.A. Biery, and R. A. Wachnik. *J. Appl. Phys.* **78**, 3756 (1995).

[Fil 96] R. G. Filippi, R. A. Wachnik, H. Aochi, J. R. Lloyd, and M.A. Korhonen. *Appl. Phys. Lett.* **69**, 2350 (1996).

[Fli 93] P.A. Flinn, A.S Mack, PR. Besser, and T. Marieb. *MRS Bulletin.* **93**, 26 (1993).

[Fli 95] P. Flinn. *MRS Bulletin.* **20**, 70 (1995).

[Fra 97] R. Frankovic and G.H. Bernstein. *J. Appl. Phys.* **81**, 1604 (1997).

[Fro 82] H.J. Frost and M.F. Ashby. Deformation Mechanism Maps: The Plasticity and Creep of Metal and Ceramics. Pergamon Press: Oxford and others, 1982.

[Fro 90] H.J. Frost, C.V. Thompson, and D.T. Walton. *Acta Met.* **38**, 1455 (1990).

[Fro 94] H.J. Frost, Y. Hayashi, C.V. Thompson, and D.T. Walton. *MRS Symp. Proc.* **317**, 485 (1994).

[Fro_2 94] H.J Frost, Y. Hayashi, C.V. Thompson, and D.T Walton. *MRS Symp. Proc.* **317**, 431 (1994).

[Gar 95] D.S. Gardner, J. Onuki, K. Kudoo, Y. Misawa, Q.T. Vu. *Thin Sol. Films.* **262**, 104 (1995).

[Gha 94] S.K. Ghandhi. VLSI Fabrication Principles: Silicon and Gallium Arsenide. John Wiley and Sons, Inc: New York and others, 1994.

[Gro 99] M.E. Gross, C. Lingk, W.L. Brown, and R. Drese. *Sol. State. Tech.* **42**, 48 (1999).

[Gut 95] R.J. Gutmann, T.P. Chow, A.E. Kaloyeros, W.A. Lanford, S.P. Murarka. *Thin Sol. Films.* **262**, 177 (1995).

[Has 74] A. Haβner, *Krist. Technik.* **9**, 1974, p. 1371 (1974).

[Hau 00] S.P. Hau-Riege. Ph.D. Thesis. Massachusetts Institute of Technology, (2000).

[Ho 70] P.S. Ho. *J. Appl. Phys.* **41**, 64 (1970).

[Ho 74] P. S. Ho and J. K. Howard, *J. Appl. Phys.* **45**, 3229 (1974).

[How 71] J.K. Howard and R.F. Ross. *Appl. Phys. Lett.* **18**, 344 (1971).

[Hu 92] C-K. Hu, P. S. Ho, and M. B. Small. *J. Appl. Phys.* **72**, 291 (1992).

[Hu 93] C. K. Hu, M. B. Small, and P. S. Ho. *J. Appl. Phys.* **74**, 969 (1993).

[Hu 95] C.-K. Hu, B. Luther, F.B. Kaufman, J. Hummel, C. Uzoh, and D.J. Pearson. *Thin Sol. Films.* **262**, 84 (1995).

[Hu 96] C.-K. Hu, K.Y. Lee, D. Gupta, P. Blauner. *Mater. Res. Soc. Symp. Proc.* **428**, 43 (1996).

[Hu 97] C.-K. Hu, K.Y. Lee,, L. Gignac, R. Carruthers. *Thin Sol. Films.* **308-309**, 443 1997).

[Hu 98] C.-K. Hu and J. M. E. Harper, *Mat. Chem. and Phys.* **52**, 5 (1998).

[Hu 99] C.-K. Hu, R. Rosenberg, and K.Y. Lee. *Appl. Phys. Lett.* **74**, 2945 (1999).

[Hun 61] H.B. Huntington and A.R. Grone. *J. Chem. Phys. Solids.* **20**, 76 (1961).

[Jon 96] B.K. Jones, Y.Z. Xu, and P. Zobb. *Microelectron. Reliab.* **36**, 1051 (1996).

[Joo 94] Y.-C. Joo and C. V. Thompson. *J. Appl. Phys.* **76**, 7339 (1994).

[Joo 97] Y.-C. Joo and C.V. Thompson. *J. Appl. Phys.* **81**, 6062 (1997).

[Kel 99] N. Kelaidis, A. Scorzoni, M. Impronta, and I. De Munari. *Microelectron. Reliab.* **39**, 627 (1999).

[Kim 93] C. Kim and J.W. Morris, Jr. *J. Appl. Phys.* **73**, 4885 (1993).

[Kir 91] R. Kircheim and U. Kaeber. *J. Appl. Phys.* **70**, 172 (1991).

[Kno 97] B. D. Knowlton, J. J. Clement, and C. V. Thompson. *J. Appl. Phys.* **81**, 6073 (1997).

[Kno_2 97] B.D. Knowlton, Ph.D Thesis. Massachusetts Institute of Technology (1997).

[Kno 98] B. D. Knowlton and C. V. Thompson. *J. Mater. Res.* **13**, 1164 (1998).

[Kon 97] S. Kondo, O. Deguchi, and K. Hinode. *Jap. J. Appl. Phys.* **36**, 2077 (1997).

[Kor 91] M.A. Korhonen, C. A. Paszkiet, and C-Y. Li. *J. Appl. Phys.* **69**, 8083 (1991).

[Kor 92] M.A. Korhonen, P. Bϕrgesen, and C.-Y. Li. Thermal Stress and Strain in Microelectronics Packaging. Van Nostrand Reinhold, New York (1992).

[Kor 93] M. A. Korhonen, P. Bϕrgesen, K. N. Tu, and C.-Y. Li. *J. Appl. Phys.* **73**, 3790 (1993).

[Kor_2 93] M. A. Korhonen, P. Bϕrgesen, D. D. Brown, and C.-Y. Li. *J. Appl. Phys.* **74**, 4995 (1993).

[Kor 95] M. A. Korhonen, T. Liu, D. D. Brown, and C.-Y. Li, *Mat. Res. Soc. Symp. Proc.* **391**, 411 (1995).

[Kra 95] O. Kraft and E. Arzt. *Appl. Phys. Lett.* **66**, 2063 (1995).

[Kra 98] O. Kraft and E. Arzt. *Acta mater.* **46**, 3733 (1998).

[Lee 95] K.L. Lee, C.K. Hu, K.N. Tu. *J. Appl. Phys.* **78**, 4428 (1995).

[Lin 98] X.W. Lin and D. Pramanik. *Sol. State. Tech.* **41**(10), 63 (1998).

[Liu 98] Y. Liu, C.L. Cox, R.J. Diefendorf. *J. Appl. Phys.* **83**, 3600 (1998).

[Llo 83] J. R. Lloyd and P. M. Smith. *J. Vac. Sci. Technol. A* **1**, 455 (1983).

[Llo 94] J.R. Lloyd and J. Kitchin. *J. Mater. Res.* **9**, 563 (1994).

[Llo 95] J.R. Lloyd and J.J. Clement. *Thin Sol. Films.* **262**, 135 (1995).

[Llo 96] J. R. Lloyd and J. J. Clement. *Appl. Phys. Lett.* **69**, 2486 (1996).

[Llo 99] J.R. Lloyd, J. Clemens, and R. Snede. *Microelectron. Reliab.* **39**, 1595 (1999).

[Lo 97] V.C. Lo and X.T. Dam. *Modelling Simul. Sci. Eng.* **5**, 563 (1997).

[Mal 97] D. W. Malone and R. E. Hummel. *Crit. Rev. in Solid State and Mat. Sci.* **22**, 199 (1997).

[Mar 94] T. Marieb, J. C. Bravman, P. Flinn, D. S. Gardner, and M. Madden. *Appl. Phys. Lett.* **64**, 2424 (1994).

[Mar 95] T. Marieb, J. C. Bravman, P. Flinn, D. S. Gardner, and M. Madden. *J. Appl. Phys.* **78**, 1026 (1995).

[Mur 85] J. L. Murray. *Int. Met. Rev.* **30**, 211 (1985).

[Obe 88] A.S. Oberai. *AIP Conf. Proc.* **167**, 146 (1988).

[Oat 91] A.S. Oates. *J. Appl. Phys.* **70**, 5369 (1991).

[Oat 95] A. S. Oates. *Appl. Phys. Lett.* **66**, 1475 (1995).

[Ove 98] D. Overhauser, J.R. Lloyd, S. Rochel, G. Steele, and S.Z. Hussain. *Microelectron. Reliab.* **38**, 851 (1998).

[Par 91] C.W. Park and R.W Vook. *Appl. Phys. Lett.* **59**, 175 (1991).

[Par 97] Y. J. Park and C. V. Thompson. *J. Appl. Phys.* **82**, 4277 (1997).

[Par 99] Y. J. Park, V. K. Andleigh, and C. V. Thompson. *J. Appl. Phys.* **85**, 3546 (1999).

[Pro 00] J. Proost, T. Hirato, T. Furuhara, K. Maex, J.-P. Celis. *J. Appl. Phys.* **87**, 2792 (2000).

[Rea 54] W.T. Read Jr. and W. Shockley. Dislocations in Metals. Inst. of Metals, Div. AIME, New York, 1954.

[Rie 96] S. P. Riege, J. A. Prybyla, and A. W. Hunt. *Appl. Phys. Lett.* **69**, 2367 (1996).

[Ros 72] R. Rosenberg -- *J. Vac. Sci. Technol.* **9**, 263 (1972).

[Ros 89] C.A. Ross, R.E. Somekh, and J.E. Evetts. *Thin Solid Films.* **173**, L129 (1989).

[Ru 99] C.Q. Ru. *Acta mater.* **47**, 3571 (1999).

[Sau 92] A.I. Sauter and W.D. Nix. *IEEE Trans. Comp. Hybrids, and Mfg.* **15**, 594 (1992).

[Sch 88] A.P. Schwarzenberger, C.A. Ross, J.E. Evetts, and AL. Greer. *J. Elec. Mater.* **17**, 473 (1988).

[Sha 86] M. Shatzkes and J. R. Lloyd. *J. Appl. Phys.* **59**, 3890 (1986).

[She 63] P.G. Shewmon. Diffusion in Solids. McGraw-Hill, New York, 1963.

[SIA 99] Semiconductor Industry Association Technology Roadmap, http://www.itrs.net/1999_SIA_Roadmap/Home.htm.

[Sim 60] R. O. Simmons and R. W. Balluffi. *Phys. Rev.* **117**, 52 (1960).

[Sri 98] V. T. Srikar and C. V. Thompson. *Appl. Phys. Lett.* **72**, 2677 (1998).

[Sri 99] V. T. Srikar, Ph.D. Thesis, Massachusetts Institute of Technology (1999).

[Sri_2 99] V. T. Srikar and C. V. Thompson, *Appl. Phys. Lett.* **74**, 37 (1999).

[Suo 98] Z. Suo. *Acta. mater.* **46**, 3725 (1998).

[Sur 94] T. Surholt, Y.M. Mishin, C. Herzig. *Phys. Rev. B.* **50**, 3577 (1994).

[Sur 97] T. Surholt and C. Herzig. *Acta Mater.* **45**, 3817 (1997).

[Tho 93] C. V. Thompson and J. R. Lloyd. *MRS Bulletin.* **18**, 19 (1993).

[Tho 95] C. V. Thompson, Y. C. Joo, and B. D. Knowlton. *Mat. Res. Soc. Symp. Proc.* **391**, 163 (1995).

[Tho 99] C. V. Thompson, S. P. Hau-Riege, and V. K. Andleigh. *Amer. Inst. of Phys. Conf. Proc.* **491**, 62 (1999).

[Vai 80] S. Vaidya, T.T Sheng, and A.K. Sinha. *Appl. Phys. Lett.* **36**, 464 (1980).

[Wal 91] D.T. Walton, J.J. Frost, and C.V. Thompson. *MRS Symp. Proc.* **225**, 219 (1991).

[Wal_2 91] D.T. Walton. Masters Thesis. Massachusetts Institute of Technology (1991).

[Wal 92] D.T. Walton, J.J. Frost, and C.V. Thompson. *Appl. Phys. Lett.* **61**, 40 (1992).

[Wal 98] P. Waltz, L. Arnaud, G. Lormand, and G. Tartavel. *Microelectron. Reliab.* **38**, 1531 (1998).

[Zha 99] Y.W. Zhang, A.F. Bower, L. Xia, and C.F. Shih. *J. Mech. and Phys. Sol.* **47**, 173 (1999).