Fall 2006

Approximating Interfacial Adhesion Energies of Thermal Barrier Coatings

Richard Foukes

Approximating Interfacial Adhesion Energies of Thermal Barrier Coatings

A Dissertation

Presented to the Bayer School

of Natural and Environmental Sciences

Duquesne University

In partial fulfillment of

the requirements for

the degree of Doctor of Philosophy

by

Richard Foukes

December 2006

Name	**Richard Foukes**
Dissertation Title	**Approximating Interfacial Adhesion Energies of Thermal Barrier Coatings**
Degree	Doctor of Philosophy
Date	December 2006

RICHARD FOUKES (Ph.D., Chemistry)

APPROXIMATING INTERFACIA ADHESION ENERGIES OF THERMAL BARRIER COATINGS (December 2006)

Abstract of a dissertation at Duquesne University

Dissertation supervised by Professor and Chairman Jeffry D. Madura

No. of pages in text: 179

This Ph.D. dissertation is centered on approximating the interfacial adhesion energies of a thermal barrier coating (TBC) within a density functional theory (DFT) framework. The strategy is to develop and validate a computational protocol to study systems which enhance the interfacial adhesion of the thermally grown oxide (TGO) to the Ni (100) substrate. The open source DFT code known as DACAPO, supported through the Technical University of Denmark, was used in this study. The investigation begins with reproducing the model system (SiO_2/Ni Complex) energetic values in the DACAPO DFT platform. These results are compared to earlier work which was carried out using the VASP (Vienna AB Initio Simulation Package) platform. The variables investigated in this study included: TGO thickness (since this species grows over time in the field), TGO phase (alpha and beta), lattice mis-match, and thermal expansion stressors. The results from this study highlight a potentially new TGO base material which does not exhibit the shift in electron density (as seen in currently used Al_2O_3) and provides a more stable TGO network within the thermal barrier coating (TBC). Finally, the protocol mapped out in this investigation can be applied quickly to screen alternative materials in the design of a new TBC system.

To My Loving Wife and Best Friend

Katie

ACKNOWLEDGEMENTS

I would like to thank Jeffry D. Madura on his dedication and focus over the years regarding this effort. His guidance and wisdom has been greatly appreciated and respected. I especially appreciate his patience with me, given my complex work demands and personal responsibilities to my family. Without his (and the committee's) understanding, I would not be here writing this.

I would also like to thank my committee, who have challenged and guided me on this evolution of knowledge. I especially appreciate and respect the diversity of expertise within the committee, as I believe it has made me a better and more knowledgeable individual.

I thank the Technical University of Denmark, especially Lars Hansen, on working with me on challenges and solutions in DACAPO Density Functional Theory platform.

Finally, the support of my wife throughout this journey is exemplary. For someone to be so supportive for so long goes beyond words of value. I feel blessed to be married to such a model person and she is the primary reason I have accomplished this goal in my life.

Vita

The author was born in Detroit Michigan on March 11, 1965 and adopted by his loving parents, Ellen and John Foukes, prior to his first birthday. He graduated from Eisenhower High School in 1983 and moved on to earn his BS in Chemistry from Central Michigan University in 1987. At that time he began his career as a chemist at Standard Products Company in Dearborn, Michigan. After 2 years, he moved to BASF Corporation as a Development Chemist in Coatings and Colorants at their Southfield, Michigan Research Center. In 1995, he accepted a position at DaimlerChrysler (DCX) in Advanced Manufacturing Engineering, within the Paint and Energy Management Department (PEM). While working at DCX in Auburn Hills, Michigan, the author earned his Master Degree in Physical Chemistry from Oakland University in 1997. After being transferred to Toronto Canada in 1997 to work at the Bramalea Assembly Plant as the Senior Manager of Paint and Sealing, he accepted a position with PPG Industries in their Coatings Research Center in Allison Park, Pennsylvania in late 2000. Currently, he resides in Marshall Twp., Pennsylvania with his loving wife Kathaleen, and their two children Megan and Brian.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Managing thermal transfer in industry is a never ending challenge. In combustion driven devices (i.e. engines) minimizing the amount of heat that is introduced in the supporting and/or adjacent substrates is highly desired. Any reduction of heat in the combustion chamber typically impacts the efficiency of the device. As the desire to reduce weight to increase overall efficiency continues, the sensitivity to extreme heat baths on a substrate is magnified. If the thickness of the walls in the chamber are reduced while the original structural integrity is required, then reducing the thermal input on the material will help maintain the structural integrity. Lower density alloys (i.e. metal blends) are more sensitive to elevated temperatures, therefore lower thermal exposure is needed to maintain structural integrity. A prime example is the environment present in gas turbine engines used in the aerospace industry. Of all industrial combustion engines, there is no other area that is more sensitive to weight and efficiency than jet turbine engines. Recognizing that the industry has identified optimum super alloys (i.e. Rene' 5N from GE), the emphasis now is on increasing efficiency of the turbine chamber. One significant effort is to enable higher temperature combustion environments[1].

1.1 Thermal Barrier Coatings

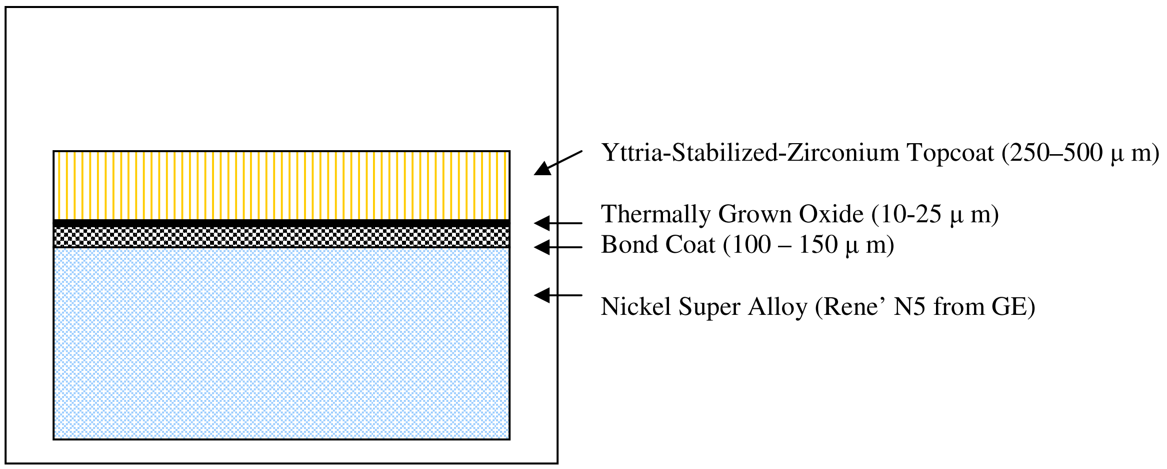
1.1.1 Purpose in Industry

Gas turbine engines utilize super alloys which possess a melting range of 1230°C to 1315°C. This super alloy operates (is in contact with) in a combustion chamber where the ambient temperature can reach 1370°C or more, noticeably above the melting point of the super alloy. To avoid structural failure by softening and/or melting, the airfoils (turbine blades made from Ni super alloy) are made hollow and discharged air is circulated through them to remove any heat that is transferred into the material (cooled) from the back side of the air foil and support wall. The industry has, and continues, to find more innovative ways (i.e. alternative geometries, cooling holes, etc.) to provide maximum cooling with minimal air consumption since any discharge air used for cooling takes away from the efficiency (i.e. thrust) of the turbine engine[2].

A second source of thermal protection for the super alloy substrate is the use of a thermal barrier coating (TBC) on the surface that is exposed to the combustion chamber. The use of thermal barrier coatings on the air foil surface provides heat reduction to the air foil with minimal impact on weight and unnecessary consumption of discharged air (both which negatively effect efficiency). The insulative nature of the TBC can reduce the turbine airfoil temperature by as much as 167°C, improving the efficiency by more than 1 percent[3]. A secondary advantage TBC provides is enhanced component durability (due to reducing corrosion by acting as a high temperature oxygen barrier to the super alloy).[1]

1.1.2 Material Construction

A TBC consists of three primary layers that cover the combustion engine super alloy which is exposed to the combustion chamber and subsequent high temperature discharge regions. The primary layer is a yttrium stabilized zirconium topcoat (YSZ) which provides the thermal protection, followed by a thermally grown oxide (TGO) layer to minimize corrosion, and a metal alloy bond coat to provide adhesion to the alloy[4] as shown in Figure 1.


Yttria-Stabilized-Zirconium Topcoat (250–500 μ m)
Thermally Grown Oxide (10-25 μ m)
Bond Coat (100 – 150 μ m)
Nickel Super Alloy (Rene' N5 from GE)

[Figure 1] **Thermal Barrier Coating Layering**

The structure of the zirconium topcoat [zirconium oxide (ZrO_2)] when heated above 1000°C changes from monoclinic to tetragonal; the accompanying 4 – 6% volume increase can result in severe spalling (delamination) of the ceramic layer. Therefore, the zirconium oxide is partially stabilized with 6 – 8% by wt. of Y_2O_3, to prevent the expansion[5]. The TGO is formed from the growth of Alumina (Al_2O_3) on the bond coat due to the presence of oxygen at elevated temperatures during application and use.

It is important to note that the TGO layer will continue to grow over time during the use of the turbine engine, which is a critical point for the efforts of this thesis. Finally, the bond coat layer comprises of a blend of Ni with Al added for TGO growth, Cr added for corrosion resistance and a trace amount of Y added to form a dense, well adhered, protective layer which enhances adhesion to the above layers throughout the heat cycling of the engine[6].

1.1.3 Failure Modes and Mechanisms

There are many failure modes of high temperature structural materials (super alloys) used in gas turbine engines, see Table 1 for the top examples[7]. The dominant failure mode experienced in the industry is spallation of the TBC from the alloy. Spallation is when two different layers of material are in stable contact (i.e. adhesion) with each other, separate at the boundary layer.

Failure Mode	**Definition**
Solid State Diffusion	Reduction of the aluminum content of the coating because of interdiffusion with the base metal
Spallation	Loss of protective oxide layer at the coating/alloy interface
Hot Corrosion	Electrochemical reaction between metal and molten salts in the presence of Oxygen at elevated temperatures
High Cycle Fatigue	Microstructural damage that results from small stress amplitude cyclic loading
Creep	Time dependent, thermally activated inelastic deformation of material. The rate of Creep increases as the temperature increases for constant stress

[Table 1] **Failure Modes of Thermal Barrier Coatings**

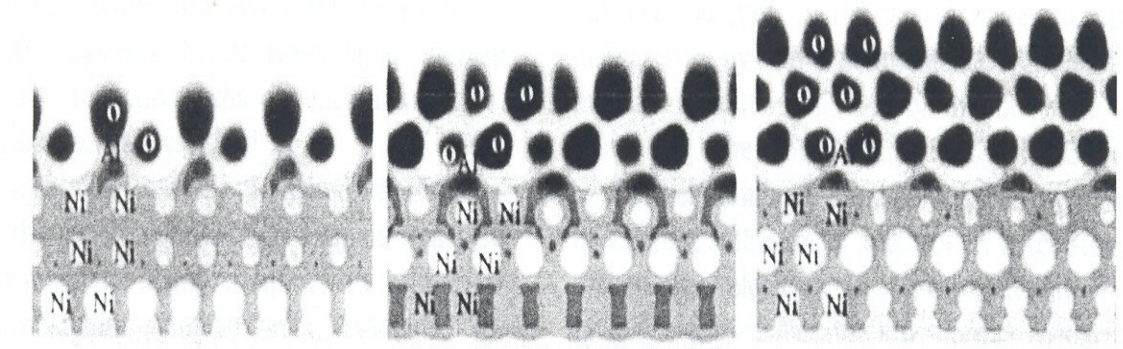
Once the TBC layer is compromised (e.g. complete removal, fracture in coating, or weak region), the corrosion of the alloy is accelerated, expediting the failure of the turbine engine. It is important to understand the mechanisms of spallation within a typical TBC.

It is known when the Al_2O_3 (TGO) layer grows over time, the internal stress increase due to the thermal expansion coefficient being lower than that of YSZ. But, the spallation failure interface experienced in the industry is at the Ni/Al_2O_3 boundary[4]. This prompted additional studies into the characteristics of Al_2O_3 as it grows over time. Carter et al found (via density functional theory simulations) that the Al_2O_3 interfacial adhesion energy decreased by an order of magnitude from 2 layers to 3 layers, see Table 2. The adhesion value for Ni/Al_2O_3 interface is far below the ideal cleavage energies and considered fairly weak for ceramic/metal systems[4]. In addition, they observed a shift in electron density from the interface atoms in Al_2O_3 to the interior bulk atoms.

Substrate + coating	**One layer coating mJ/m^2**	**Two layer coating mJ/m^2**	**Three layer coating mJ/m^2**
Al_2O_3 + ZrO_2	1,142	1,256	1,189
Ni + ZrO_2	2,011	1,308	995
Ni + Al_2O_3	618	943	456

[Table 2] **Interfacial Adhesion Energies (W_{ad})**

This shift in electron density clearly compromised the interfacial adhesion energy at the Ni/Al_2O_3 interface, promoting spallation of the coating (see Figure 2).



[Figure 2] **Electron Density for 0.5, 1.0, 2.0 layers of Al_2O_3 on Ni.** Courtesy of Emily Carter and HPC and National Security, March/April 2002

In this case, the weakening interfacial adhesion trends can be explained largely by the strength of nickel-aluminum interactions. As additional layers of Al_2O_3 are deposited on itself during the TGO growth process, the Al_2O_3 monolayer at the Ni surface rearranges favoring the aluminum-oxygen interactions. This electron density shift weakens the nickel-aluminum bonds, decreasing the interfacial adhesion energy[4].

Numerous attempts at reducing the growth of the Al_2O_3 layer over time have not much success. A radical approach could be the replacement of the Al_2O_3 with an alternative oxide which does not exhibit the shift in electron density from the interface atoms to the interior bulk, which is a key weakness of Al_2O_3 as the TGO. It is known that Alumina is a highly ionic, closed shell species and does not possess enough covalent content. A more covalent oxide product might exhibit enhanced interfacial adhesion through more covalent bonding[4]. SiO_2 (silicon dioxide or quartz) is a more covalent species compared to Al_2O_3. Preliminary DFT investigations show that SiO_2 may demonstrate increased adhesion energies independent of thickness. (see Table 3)

SiO_2 thickness on Ni	**1 layer (5 Å)**	**2 layer (10 Å)**	**3 layer (15 Å)**
Interfacial Adhesion Energy (W_{ad}) (mJ/ m^2)	1,292	1,374	1,342

[Table 3] **Interfacial Adhesion Energy (W_{ad}) of Ni/SiO_2 Complex**

This investigation was not exhaustive and many factors were not considered. However, the data is encouraging.

1.2 Objective of Thesis

We have clearly reviewed the deficiencies of Al_2O_3 TGO within thermal barrier coatings with respect to spallation tendencies. The weakening of the interface bonds between the Al and Ni are a systemic issue within the complex. Attempts to minimize the growth of the TGO have not resolved the problem to the satisfaction of the aerospace industry. Therefore, since the reduction of interfacial adhesion energy is at the atomic level in the boundary between Ni and Al atoms of the super alloy and TGO respectfully, a realistic solution can only be obtained using quantum chemical methods.

<u>The thesis is centered on 2 main aspects:</u>

1) Using the existing literature work on Ni/SiO_2 complex as a reference point, develop a density functional theory simulation protocol using the open source code of DACAPO from CAMP (Center for Atomic-scale Materials Physics).[8] This will allow any researcher to impose specific orientations, environments, element and/or material options, etc. to the system and determine any energetic changes. These energetic measurements can then be used to calculate the cohesive, surface free, and interfacial adhesion energies of the complex, all which play key roles in the spallation.

2) Determine if SiO_2 is a viable replacement TGO material for Al_2O_3. This investigation will center on the beta phase of SiO_2 since it is the predominant phase present at the combustion temperatures in the gas turbine engines. The previous literature study on SiO_2 only considered the alpha phase, which is only stable up to 847 °K (574 °C). The combustion temperature range for a gas turbine engine reaches > 1300 °C, where the beta phase is dominant. Also, the present study will understand the influence, if any, of the alpha – beta phase transition which will occur during use, since the TGO will be at an ambient temperature when not in use and an elevated temperature (1300 °C) when fired. To accomplish this, we will study the potential energy surface (PES) of the alpha phase as it approaches the inversion point (during heat cycle) and after, where the β phase is predominant. The energy values will be used to determine cohesive, surface free, and interfacial adhesion energies. Finally, we ensure that reasonable lattice mis-match from either different coefficient of thermal expansions or deposition conditions, do not generate elevated internal stress at the boundary layer, compromising the adhesion of the Ni/SiO_2 interface.

1.3 References

(1) Sheffler, K., Gupta, D., *Current Status and Future Trends in Turbine Application of Thermal Barrier Coatings*, ASME Journal of Engineering for Gas Turbines and Power, Vol. 10, p. 605 – 609 **(1998)**

(2) Meier, S., Gupta, D., *The Evolution of Thermal Barrier Coatings in Gas Turbine Engine Applications*, AMSE, Vol. 116, Jan **(1994)**

(3) Miller, R., Lowell, C., *Failure Mechanisms of Thermal Barrier Coatings Exposed to Elevated Temperature*, Thin Solids Films, Vol. 99, p. 265 **(1982)**

(4) Jarvis, E., Carter, E., *The Role of Reactive Elements in Thermal Barrier Coatings,* HPC and National Security, March/April, pp. 33 - 41 **(2002)**

(5) Duvall, D., Ruckle, D., *Ceramic Thermal Barrier Coatings for Turbine Engine Components*, ASME paper 82-GT-332, **(1982)**

(6) Beele, W., Marijnissen, G., van Lieshout, A., *The evolution of thermal barrier coatings – status and upcoming solutions for today's key issues*, Surface Coatings Tech, 120 – 121, pp. 61 – 67 **(1999)**

(7) National Materials Advisory Board, *Coatings for High Temperature Structural Materials*, National Academy Press, **(1996)**

(8) Bahn, S., Jacobsen, K., *An Object-Oriented Scripting Interface to a Legacy Electronic Structure Code*, Programming Languages, May/June, pp.56 – 66, **(2002)**

Chapter 2

Electronic Structure Background

Electronic structure methods use the postulates of quantum mechanics as the basis of their development. Quantum mechanics states that the energy of an atom or molecule can be obtained by solving the Schrödinger equation [1],

$$H\psi = E\psi \qquad [1]$$

where H is the Hamiltonian operator, Ψ is the wave function, and E is the total energy.[1]

The only atomic system which can be solved analytically is a single proton and one electron system. As a result, mathematical approximations are used to determine a solution. There are 3 primary classes of electronic structure methods:

1) Semi-empirical, simplifies the solution to the Schrödinger equation by replacing some terms with experimental values.[2]

2) Density functional theory (DFT) solves an "idealized" many-electron problem as opposed to the full N-electron wave function problem as does the Hartree-Fock ab initio approach. As a result, the many-electron effects (i.e. electron correlation) are taken into account explicitly. The central idea within DFT is that there is a relationship between the total electronic energy and the overall electronic density.[3] Practically, DFT only attempts to calculate the total electronic energy and the overall electronic density distribution. This will be discussed in greater detail later.

3) Ab initio methods, unlike semi-empirical and DFT above, use no experimental parameters in their computations. Instead, these computations are primarily based on the postulates of quantum mechanics as well as the Born-Oppenheimer approximation and an independent electron approximation.[2]

2.1 The Schrödinger Equation

Quantum mechanics begins with the full, time dependent Schrödinger equation [2].

$$\left\{-\frac{\hbar^2}{2m}\nabla^2+V\right\}\boldsymbol{\psi}(\mathbf{r}',t)=\frac{i\hbar\partial\boldsymbol{\psi}(\mathbf{r}',t)}{\partial t} \qquad [2]$$

Where $\boldsymbol{\psi}$ is the wave function, m is the mass of the particle, $\hbar$ is Planck's constant, *V* is the potential in which the particle is moving, and ∇^2 is the partial differentiation with respect to x, y, and z components known as "del-squared".
The energy and many other properties of a particle (e.g. electron, proton, etc.) can be obtained by solving [2] for $\boldsymbol{\psi}$. The many different wave functions which are solutions to [2], correspond to the different stationary states of the system.[2] If *V* is not a function of time, the Schrödinger equation can be simplified using separation of variables and is reduced to the form

$$H\psi(r')=E\psi(r') \qquad [3]$$

Where E is the eigenenergy of the particle and H is the Hamiltonian operator, equal to:

$$H = \frac{-\hbar^2}{2m}\nabla^2 + V \qquad [4]$$

Note that equation [3] is an eigenvalue equation in which the operator acting on a function produces the identical wave function and an eigenvalue associated with it.[2]

2.2 Born-Oppenheimer Equation

The Schrödinger equation can further be simplified if one takes advantage of the significant difference in size between the proton and electron in an atomic species. The mass of a proton is approximately 1800 times more massive than the corresponding electron. For example, the nucleus of carbon is 20,000 times larger than the electrons.[4] Born-Oppenheimer used this fact to assume that (from an extreme point of view) the electrons move in a field of fixed nuclei. It is this assumption that lends to the famous Born-Oppenheimer approximation. As a result, since the nuclei are fixed in space, their kinetic energy is zero and the potential energy due to nucleus-nucleus repulsion becomes a constant. This approximation is used to separate the Hamiltonian and wave function into nuclear and electronic terms. The total Hamiltonian of an atomic species of *M* nuclei and *N* electrons[4] is

$$H = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \frac{1}{2}\sum_{A=1}^{M}\frac{1}{M_A}\nabla_A^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}} + \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}} \qquad [5]$$

Where Z is nuclear charge, R_{AB} is distance between nuclei, and r_{ia} is distance between nuclei and electrons. [5] can then be reduced to the electronic Hamiltonian

$$H^{elec} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}} \qquad [6]$$

The solution of the Schrödinger equation using H^{elec}, is the electronic wave function $\boldsymbol{\psi}^{elec}$, and the total electronic eigenenergy E^{elec}, and the equation takes the form[4]

$$H\boldsymbol{\psi}^{elec} = E\boldsymbol{\psi}^{elec} \qquad [7]$$

It is critical to realize that the wave function itself is not observable. A physical interpretation can only be associated with the square of the wave function where

$$\left|\psi(x_1, x_2, \ldots\ldots x_n)\right|^2 dx_1, dx_2, \ldots\ldots dx_n \qquad [8]$$

is the probability that electrons 1,2,…n are found simultaneously in the volume elements dx_1,dx_2,…..dx_n.[4] The probability interpretation of the wave function is the integral of [8] over the full range of variables and is equal to one. In other words, the probability of finding *n* electrons anywhere in space must be unity

$$\int_{-\infty}^{\infty} \psi(x_1, x_2, \ldots\ldots x_n)^2\, dx_1, dx_2, \ldots\ldots dx_n = 1 \qquad [9]$$

and the wave function that satisfies [9] is said to be normalized.[4]

2.3 Density Functional Theory

Density functional theory (DFT) was established by the efforts of Hohenberg and Kohn[5] and Kohn and Sham[6] in the sixties and has become one of the most successful electronic structure methods for condensed matter physics. DFT is based on the observation that the ground state electronic properties of a system are functionals of the ground state electron density. Also, Kohn and Sham demonstrated that the minimization of the total energy functional can be reduced to a set of single particle Schrödinger-like equations with all the many body effects captured in the exchange correlation term.[7]

2.3.1 Hohenberg and Kohn Theorems

Hohenberg and Kohn theorems form the basis of DFT. They consider a system of N electrons in an external potential $V_{\text{ext}}(\mathbf{r})$ where there is a universal functional $F\{n(\mathbf{r})\}$ of electron density $n(\mathbf{r})$. The ground state electron density is minimized to form

$$E\{n(\mathbf{r})\} = \int d\mathbf{r} V_{ext}(\mathbf{r}) n(\mathbf{r}) + F\{n(\mathbf{r})\} \qquad [10]$$

where

$$\int d\mathbf{r}n(\mathbf{r}) = N \qquad [11]$$

and the minimum of $E\{n(\mathbf{r})\}$ gives the ground state total energy.[7] Unfortunately, the form of $F\{n(\mathbf{r})\}$ is unknown and the Hohenberg and Kohn theorems by themselves are not enough for any practical application.[5] Therefore, reasonable approximations, as the ones developed by Kohn-Sham[5], are necessary.

2.3.2 Kohn-Sham Equations

The unknown functional $F\{n(\mathbf{r})\}$ in equation [10] includes the contributions from the kinetic energy and the electron-electron interactions. Kohn-Sham represented the total energy functional in the following manner.

$$E\{n(\mathbf{r})\} = T_0\{n(\mathbf{r})\} + \frac{1}{2}\int d\mathbf{r}n(\mathbf{r})\mathbf{\Phi}(\mathbf{r}) + F\{n(\mathbf{r})\} + \int d\mathbf{r}n(\mathbf{r})V_{ext(r)} + E_{xc}\{n(\mathbf{r})\} \qquad [12]$$

Where T_0 is the kinetic energy functional with density $n(\mathbf{r})$, followed by the classical electrostatic energy, with $\mathbf{\Phi}(\mathbf{r})$ being the coulomb potential for electrons and is represented by

$$\mathbf{\Phi}(\mathbf{r}) = \int d\mathbf{r}' \frac{n(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}, \qquad [13]$$

The last term in [12], $E_{\mathrm{xc}}\{n(\mathbf{r})\}$, is the exchange-correlation energy functional.[6] Kohn-Sham showed that the density that minimizes [12] can be represented by

$$n(\mathbf{r}) = \sum_{i=1}^{N} |\boldsymbol{\psi}_i(\mathbf{r})|^2 \qquad [14]$$

where $\boldsymbol{\psi}_i(\mathbf{r})$ are solutions of the Schrödinger equation for a system of N electrons and can be represented by the famous Kohn-Sham expression [15], which is constructed along the lines of the classic Schrödinger equation.

$$\left[-\frac{1}{2}\nabla^2 + V_{eff}(\mathbf{r})\right]\boldsymbol{\psi}(\mathbf{r}) = \varepsilon\boldsymbol{\psi}(\mathbf{r}) \qquad [15]$$

where V_{eff} is the effective single electron potential defined as

$$V_{eff}(\mathrm{r}) = V_{ext}(\mathbf{r}) + \boldsymbol{\Phi}(\mathbf{r}) + V_{xc}(\mathbf{r}) \qquad [16]$$

and $V_{xc}(\mathbf{r})$ is the exchange correlation functional, which is further defined in the next section. A set of single-electron Kohn-Sham equations in [15] are orders of magnitude easier to solve than the original many body expression [12]. The final step required is an approximation for the exchange correlation functional $V_{xc}(\mathbf{r})$.[7]

2.3.3 Exchange Correlation Approximations

The simplest approximation for the exchange-correlation functional is the local density approximation (LDA). In LDA it is assumed that the contribution to the exchange correlation energy from each point **r** with the local electron density $n(\mathbf{r})$ is the same as in the uniform electron gas with the corresponding electron density n(**r**).[8] The exchange correlation functional takes the form

$$E_{xc}^{LDA}[n] = \int d\mathbf{r} n(\mathbf{r}) \varepsilon_{xc}^{unif}[n(\mathbf{r})] \qquad [17]$$

where $\varepsilon_{xc}{}^{unif}[n(\mathbf{r})]$ is the exchange correlation energy per electron ($\mathbf{r}$) in a homogeneous electron gas, with the density n. It must be recognized that there are many situations where the LDA leads to unacceptably large over estimates of the bonding strength in most transition metals, leading to inaccurate ground state structure and energies due to the fact that the electron density is anything but uniform.[10] This can be improved by mapping out the density gradients in the approximation, which is done within the generalized gradient approximation (GGA). The GGA gradient expansion is represented by the following generalized functionals of density gradients[11]

$$E_{xc}^{GGA}[n] = \int d\mathbf{r} n(\mathbf{r}) \varepsilon_{xc}^{GGA}[n, \nabla_n] \qquad [18]$$

Where $\nabla = \frac{\partial}{\partial x_n}\hat{i} + \frac{\partial}{\partial y_n}\hat{j} + \frac{\partial}{\partial z_n}\hat{k}$. The most popular GGA used in DFT is that of Perdew and Wang 1991 (i.e. PW91-GGA) which has been shown to correct the serious deficiencies of LDA for transition metals, providing accurate descriptions of their structural and cohesive properties.[12] This thesis utilizes the proven PW91-GGA, exclusively.

2.4 Applications to Solids

One can apply the Kohn-Sham equation [15] to a periodic system (solids) by utilizing Bloch's Theorem[13] which states that the solutions of [15] can be written in the form

$$\boldsymbol{\psi}(\mathbf{r}) = \exp(i\mathbf{k}\cdot\mathbf{r})u_{\mathbf{k}}(\mathbf{r}) \qquad [19]$$

where $u_{\mathbf{k}}(\mathbf{r})$ has the periodicity of the system with respect to the $\mathbf{k}$ vector for electron $\mathbf{r}$. Substituting [19] into [15] leads to the following expression for $u_{\mathbf{k}}(\mathbf{r})$.

$$\left[-\frac{1}{2}(\nabla + i\mathbf{k})^2 + V_{eff}(\mathbf{r})\right]u_k(\mathbf{r}) = \varepsilon_k u_k(\mathbf{r}) \qquad [20]$$

Using Bloch's theorem one only needs to consider the unit cell in solving equation [20]. Finally, the electron density $n(\mathbf{r})$ takes the form of

$$n(\mathbf{r}) = 2\sum \int d\mathbf{k} f\left(\varepsilon_{ik} - \varepsilon_F\right)\left|u_{ik}(\mathbf{r})\right|^2 \qquad [21]$$

where *i* is the band index (energy levels) for the different eigenstates, to yield the corresponding eigenvalues, $\varepsilon_{i\mathbf{k}}$.[13]

2.5 Summary

Density functional theory was established when Kohn-Sham identified the link between ground state electronic energy and the ground state electron density. Considering this postulate, one could work within electron density (applicable to large, solid systems) and approximate the electronic energy of a solid system with confidence. Using a generalized gradient approximation to represent the exchange correlation functional in a non-homogeneous electron density was critical in accurately approximating transition metal complexes.

2.6 References

(1) Ratner, M., Schatz, G., *Introduction to Quantum Mechanics in Chemistry*, Prentice Hall, **(2001)**

(2) Foresman, J., Frisch, A., *Exploring Chemistry with Electronic Structure Methods*, Gaussian, Inc., **(1996)**

(3) Leach, A., *Molecular Modeling, Principles and Applications*, Prentice Hall, **(2001)**

(4) Koch, Wolfram., Holthausen, Max., *A Chemist's Guide to Density Functional Theory*, Wiley-VCH, **(2001)**

(5) Hohenberg, P., Kohn, W., *Inhomogeneous electron gas*, Phys. Rev. 136, B864 **(1964)**

(6) Kohn, W., Sham, L., *Self Consistent equations including exchange and correlation effects*, Phys. Rev. 140, A1133 **(1965)**

(7) Dudiy, S., *Microscopic Theory of Wetting and Adhesion in Metal-Carbonitride Systems,* Chalmers University of Technology, Göteborg University, **(2002)**

(8) Gunnarsson, O., Lundqvist, B., *Exchange and correlation in atoms, molecules, and solids by the spin-density-functional formalism*, Phys. Rev. B **13**, 4274 **(1976)**

(9) Barth, J., Hedin, L., *A local exchange-correlation potential for the spin polarized case. i*, J. Phys. C **5**, 1629 **(1972)**

(10) Moruzzi, V., Janak, J., Williams, A., *Calculated Electronic Properties of Metals*, (Pergamon, New York, **1978**)

(11) Langreth, D., Mehl, M., *Easily implementable nonlocal exchange-correlation energy functional*, Phys. Rev. Lett. **47**, 446 **(1981)**

(12) Perdew, J., Chevary, J., Vosko, S., Jackson, K., Pederson, M., Singh, D., Fiolhais, C., *Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation*, Phys. Rev. B **46**, 6671 **(1996)**

(13) Ashcroft, N., Mermin, N., *Solid State Physics* (Sounders College Publishing, Fort Worth, **1976**)

Chapter 3

Computational Method

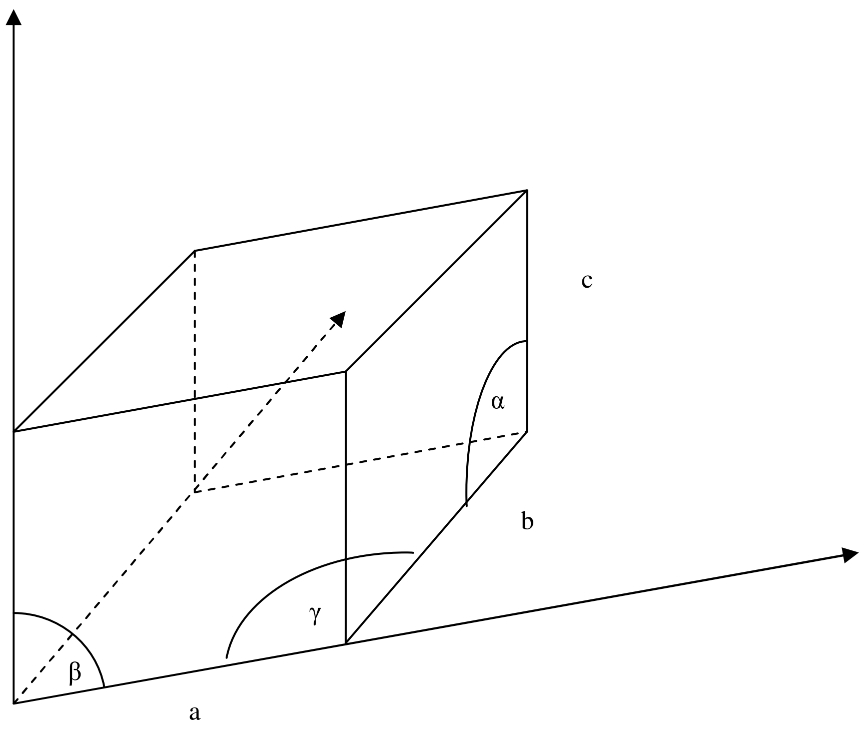
The density functional theory discussed in Chapter 2 has become a much desired (and appreciated) tool in the quantum mechanical investigation of electronic properties in a condensed matter system. DFT nicely distills down to solving a set (very large at times) of coupled single particle Kohn-Sham equations [15] and [16]. It must be recognized that the solutions to the Kohn-Sham equations still require a sizable computational effort and for many years the practical application of the first principles DFT were somewhat limited to the size of the system that could be undertaken in a reasonable amount of time. This section describes the computational platform and key elements of its construction. Chapter 2 reviewed how DFT was discovered in the sixties by Hohenberg and Kohn, followed by the practical equations of Kohn-Sham. This chapter will focus on the specific functions, planewaves, for basis sets and the use of pseudopotentials as a means to simplify the inclusion of core electrons. A review of DACAPO[1], our computational platform, utilized in the investigation of thermal barrier coatings, is covered.

3.1 Bravais Lattice and k-points

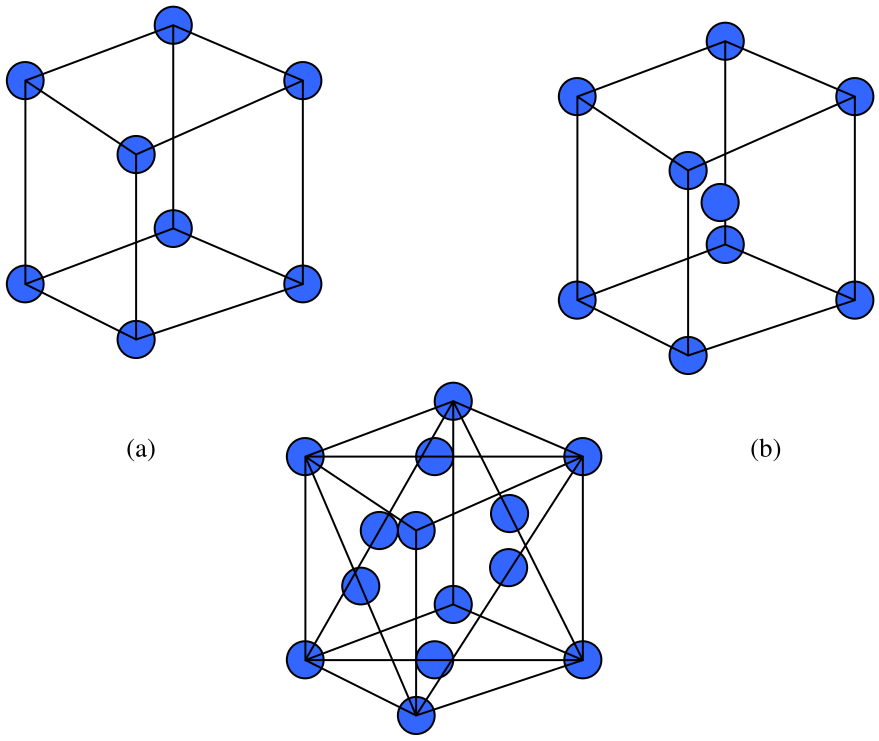
The quantum mechanical investigations of solids are different than those traditionally utilized for individual molecules or isolated complexes. A crystalline system can be constructed by infinitely stacking copies of some repeating unit (unit cell) in a systematic fashion without gaps[7]. The unit cell can be characterized by three lattice vectors **a**, **b**, **c**, and the angles between them α, β, γ, shown in Figure 4. The general vector **r** can be expressed by the fractional coordinates

$$\mathbf{r} = (\alpha\mathbf{a}, \beta\mathbf{b}, \gamma\mathbf{c}) \qquad [22]$$

There are fourteen different types of unit cells called Bravais Lattices[2]. Common ones include cubic, body centered cubic, face centered cubic (nickel), and hexagonal (Silica). Refer to Figure 5 for illustrations and additional examples.



[Figure 3] **Lattice Vectors of a Unit Cell**

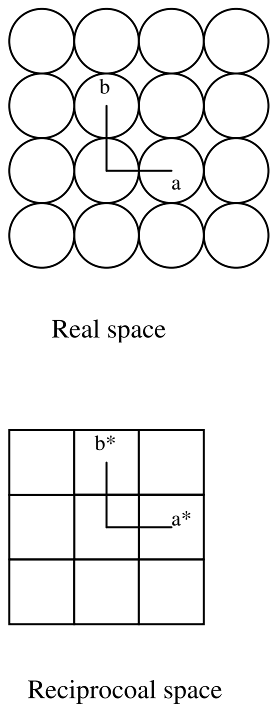


[Figure 4] **Typical Unit Cells "Bravais Lattice".** (a) Simple Cubic, (b) Body Centered Cubic, (c) Face Centered Cubic. Courtesy of Pearson Education Limited; Leach, A., *Molecular Modeling, Principles and Applications*, Prentice Hall, (2001)

Another concept that is critical when working with lattice structures is the reciprocal lattice. X-ray crystallographers use a reciprocal lattice defined by three vectors $\mathbf{a}^*$, $\mathbf{b}^*$, $\mathbf{c}^*$, where $\mathbf{a}^*$ is perpendicular to $\mathbf{b}$ and $\mathbf{c}$ and is scaled so that the scalar product of $\mathbf{a}^*$ and **a** equals 1[2], $\mathbf{b}^*$ and $\mathbf{c}^*$ are defined the same. See equation [23]

$$\mathbf{a}^* = \frac{b \times c}{a \cdot b \times c} \qquad \mathbf{b}^* = \frac{a \times c}{b \cdot a \times c} \qquad \mathbf{c}^* = \frac{a \times b}{c \cdot a \times b} \qquad [23]$$

Recognizing that the denominator is equal to the volume of the unit cell, it forces $\mathbf{a}^*$, $\mathbf{b}^*$, and $\mathbf{c}^*$, to have units of 1/length. This is the representation of reciprocal space and reciprocal lattice[2]. An illustrative example of reciprocal space is that of 2D square lattice where the vectors **a** and **b** are orthoganol and of length equal to lattice spacing, **a**. Here **a*** and **b*** are directed along the same directions as **a** and **b** respectively and have a length 1/**a**. see Figure 6.



[Figure 5] **2 Dimensional Reciprocal and Real Space Representation.**

This reciprocal lattice is known as the Brillouin zone and will be referred to as such, throughout the remainder of this thesis.

After successfully defining the Brillouin zone (unit cell and reciprocal space), it is important to understand where we sample (points) the space to produce a realistic approximation of the bulk properties. Recognizing that it is not practical to sample every point in the Brillouin zone (BZ), we need to determine a representative sampling matrix we can trust. Many calculations in crystals involve the averaging of a set of points (**k**) over the BZ. These special **k**-points were developed by Chadi-Cohen[2] and avoid the use of interpolation in the calculation of averages[3],as opposed to Monkhorst and Pack. If one considers a face centered cubic Bravais lattice (Nickel) a good starting point for $\mathbf{k} = (k_x, k_y, k_z)$ is $k_1 = (½, ½, ½)$ with units of $2\pi/a$, where a is the lattice constant.[3] 2π is an expansion constant conveniently used in crystallographic investigations.[2] An example of a **k**-point series (not comprehensive) is represented in Figure 6.

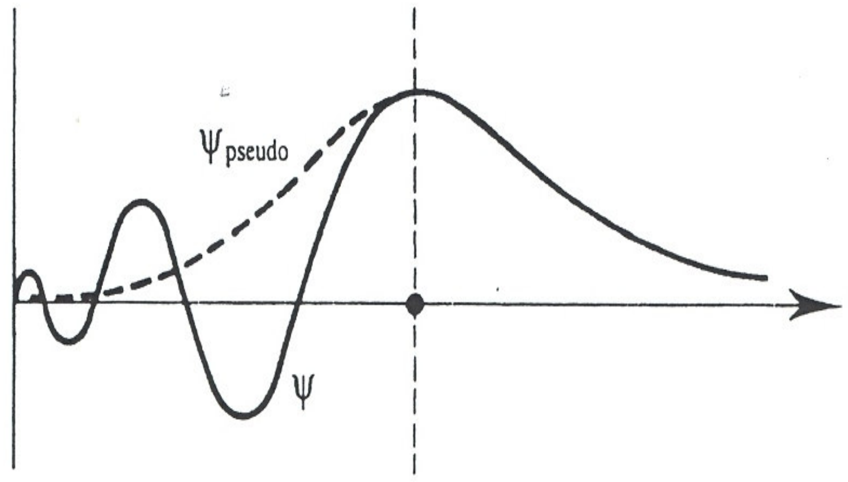
k-point vector	**Weight factor**	**k-point vector**	**Weight factor**
k_1 = (7/8, 3/8, 1/8)	α_1 = 3/16	k_2 = (7/8, 1/8, 1/8)	α_2 = 3/32
k_3 = (5/8, 5/8, 1/8)	α_3 = 3/32	k_4 = (5/8, 3/8, 3/8)	α_4 = 3/32
k_5 = (5/8, 3/8, 1/8)	α_5 = 3/16	k_6 = (5/8, 1/8, 1/8)	α_6 = 3/32
k_7 = (3/8, 3/8, 3/8)	α_7 = 1/32	k_8 = (3/8, 3/8, 1/8)	α_8 = 3/32
k_9 = (3/8, 1/8, 1/8)	α_9 = 3/32	k_{10} = (1/8, 1/8, 1/8)	α_{10} = 1/32

[Figure 6] **Face-Centered-Cubic Bravais Lattice**

3.2 Pseudopotentials

A major problem in DFT calculations based on planewave basis sets is the description of the core electrons in the atom. This is because the wave functions of these core electrons are rapidly oscillating in space. To accurately represent all these oscillations would require a prohibitively large set of planewaves, quickly becoming impractical. This problem is solved within the pseudopotential approximation.[4]

In atomic level interactions, the valence electrons of the atom are of most interest since they are largely responsible for bonding and electronic properties.[2] The core electron contributions are minimized from the overwhelming electrostatic potential the protons have on the electrons, cancelling out their kinetic energy[2] [(refer back to the Born-Oppenheimer Approximation in Section 2.2)]. Attempting to represent the core electrons with planewaves, capturing all the rapid oscillations that are present in this region, is a formidable task described earlier. Also, transition metal systems which possess many core electrons, instill an even greater computational demand. To deal with this, the true potential of the core electrons is replaced with a much weaker potential called a pseudopotential.[2] A pseudopotential is a potential function that gives the exact behavior of the electron outside the core region, beyond the core radius, smoothing out the core region with minimal nodes, illustrated in Figure 7. This has an effect of reducing the number of terms required for the plane wave expansion of the wave function, which drastically reduces the computational demand of the calculation.



[Figure 7] **Schematic of Pseudopotential Wave Function.** Courtesy of Pearson Education Limited; Leach, A., *Molecular Modeling, Principles and Applications*, Prentice Hall, **(2001)**

Pseudopotentials are constructed from a nearly all electron atomic calculations with the valence electrons being represented exactly to reproduce the behavior and properties of the valence electrons.[2] Therefore, the pseudopotentials are the wave functions in which the core electron terms are represented with a simplified term and the valence electrons are explicitly represented to reduce the complexity of calculations.

3.3 Exchange Correlation Functional: GGA

One of the critical differences in DFT calculations is the exchange correlation functional, discussed earlier. Originally, the functional was based on a hypothetical assumption of a uniform electron gas (i.e. density). This is a system where the electrons move on a positive background charge distribution and is electrically neutral.[5] This assumption is reasonable with small, tightly bound electron densities but is unrealistic with larger atoms (especially transition metals) due to the rapidly varying densities. Kohn-Sham provided the most complete correlation functional based on non uniform electron densities (section 2.3.3)[5] and were widely accepted when investigating transition metal complexes. Within that, the simplest approximation for E_{xc} was the local density approximation (LDA)

$$E_{xc}^{LDA}[n] = \int d\mathbf{r}\, n(\mathbf{r})\, \varepsilon_{xc}^{unif}[n(\mathbf{r})] \qquad [24]$$

which has a strong tendency to over estimate ground state energies. This was correct by utilizing the generalized gradient approximation for the exchange correlation functional that captures the density gradients within the cloud.

$$E_{xc}^{GGA}[n] = \int d\mathbf{r} n(\mathbf{r}) \varepsilon_{xc}^{GGA}[n, \nabla_n] \qquad [25]$$

This approximation favors density inhomogeneity more than LDA does. Perdew-Wang 1991 provides an analytical fit to this numerical GGA[5] and is exclusively used in the first principle electronic investigations (DACAPO) in this thesis.

3.4 Planewaves as Basis Set

Density functional theory method represents the Kohn-Sham wave functions as discrete sets of numbers through expanding the wave functions in some basis set within a finite number of terms, based on the system of choice. The choice of basis functions is critical to the accuracy and efficiency of the approximation.[6] There are many choices available within DFT like: linearized-muffin-tin-orbitals[7] (LMTO), full potential linearized augmented plane wave[8] (FLAPW), and the plane wave psuedopotential[9] (PWPP) method. This thesis utilized the PWPP method exclusively, which is explained later in section 3.2.

Planewaves are based on the fundamental wave function

$$\Phi^{pw}(\mathbf{r}) = \exp[i\mathbf{k}\cdot\mathbf{r}] \qquad [26]$$

where **k** is a lattice vector.[4] An important condition on the expansion of the wave function [26] is the periodicity of the system and the ability to generate a finite set of points that can represent the infinite (bulk) system. Since, our system is periodic in nature we can utilize Bloch's Theorem [19], discussed in section 2.4. Bloch's Theorem, for periodic systems, allows one to represent the wave functions in terms of $u_{\mathbf{k}}$ that retain the periodicity of the system. Whilst, enabling us to generate an approximation of the electronic properties of the bulk, infinite system of choice.

Expanding the functions $u_{\mathbf{k}}$ [19] in the plane wave basis set [26], one can represent the wave functions for each k-point (i.e. designated points in unit cell which in total represents the complete cell) as

$$\psi_{nk}(\mathbf{r}) = \sum_{G} c_{nk}(\mathbf{G})\exp[i(\mathbf{k}+\mathbf{G})\mathbf{r}] \qquad [27]$$

where the summation is over all reciprocal-space vectors **G**, and $c_{n\mathbf{k}}(\mathbf{G})$ are the expansion coefficients. Reciprocal space is introduced here to to completely fill the space without overlapping and without any gaps.[6] DFT and DACAPO take advantage of the fact that not all planewave contribute significantly to the total energy. In other words, there is a point where the kinetic energy of the planewaves plateau, minimizing the contribution to the total electronic energy. This allows DFT to employ a planewave cutoff E_c condition where the approximation only utilize planewaves that have energies less than the designated cutoff (E_c).

$$\frac{|\mathbf{k}+\mathbf{G}|^2}{2} < E_c \qquad [28]$$

Ec is typically determined by running a ladder of values, plotting the corresponding energies, and determining the value. This allows for a more efficient calculation, without compromising the accuracy of the approximated electronic energy.

3.5 DACAPO

DACAPO density functional theory code was developed at the Center for Atomic-scale Materials Physics (CAMP) at the Technical University of Denmark to describe atomic system structure and dynamics.[10] Using quantum mechanical descriptions of electronic motion, DACAPO generates the electronic energy and forces for a system of interest. The quantum mechanical postulates that govern electronic behavior are the same for all atomic systems, allowing DACAPO to cover a broad spectrum ranging from molecular dynamics and/or structural relaxation simulations involving reactivity and diffusion on metal surfaces, surface energies, cohesive energies, and overall electronic energies.[11] DACAPO uses planewaves for the valence electrons and describes the core electron interactions with pseudopotentials. The program performs self-consistent calculations for both LDA and GGA exchange correlations potentials (reviewed in 2.3.3), using state of the art algorithms.[12]

The DACAPO code originated in the 1980's and was written in Fortran 77. CAMP researchers have implemented several iteration schemes and they continue to improve and update the code.[13] DACAPO requires a large number of input parameters to effectively approximate the electronic energies and behaviors of a many body atomic system. These include: unit cell shape and size, atomic positions and species, planewave cutoff (described earlier), k-point sampling, exchange correlation functional, and minimization scheme[1].

3.6 Electronic Minimization via Block Davidson Algorithm

First principles quantum mechanical ground state electronic energy calculations require large minimization sequences for convergence. DACAPO utilizes an efficient method (iterative Davidson algorithm) to determine self consistent field (SCF) solution of large eigenvalue problems.[14] It has been found that the best efficiency occurs when one employs small block size iterations, allowing the algorithm to be less demanding on memory,[14] speeding up the calculation. This approach is the Block Davidson Algorithm exclusively employed by DACAPO.

In Density Functional Theory the ground state total energy E_0 can be obtained as the global minimum of the energy functional $E[\rho]$.

$$E[\rho] = T[\rho] + V_{cc} + \int \rho(\mathbf{r}) \upsilon_{ext}(\mathbf{r}) d\mathbf{r} + \gamma_{ewald} \qquad [29]$$

Where $\mathbf{r}$ is the position vector in space, $\rho(\mathbf{r})$ is the charge density of electrons satisfying

$$\int \rho(\mathbf{r}) d\mathbf{r} = N \qquad [30]$$

with N representing the number of electrons in the system.[14] $\upsilon_{ext}(\mathbf{r})$ is the external potential function acting on the electrons from the nuclei. The kinetic energy $T[\rho]$ and the electron-electron interaction energy $V_{cc}[\rho]$ are functionals of the density ρ and γ_{Ewald} is the electrostatic repulsion energy between the nuclei.[14]

Kohn-Sham proved that the charge density $\rho(\mathbf{r})$ can be represented by

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} \left|\boldsymbol{\psi}_i(\mathbf{r})\right|^2 \qquad [31]$$

where $\Psi_i(\mathbf{r})$ are discrete quantum mechanical wave functions of the electrons. The kinetic energy functional T_s of the system is represented as

$$T_s[\rho] = -\frac{1}{2}\sum_{i=1}^{N}\int \boldsymbol{\psi}_i^*(\mathbf{r})\nabla^2\boldsymbol{\psi}_i(\mathbf{r})d\mathbf{r} \qquad [32]$$

Allowing equation [29] to be re-written as

$$E[\rho] = T[\rho] + E_{Hartree}[\rho] + E_{xc}[\rho] + \int \rho(\mathbf{r}) \upsilon_{ext}(\mathbf{r}) d\mathbf{r} + \gamma_{ewald} \qquad [33]$$

Where $E_{Hartree}[\rho]$ is the classical electrostatic energy of the electrons and $E_{xc}[\rho]$ is the exchange correlation energy functional.[14] The Kohn-Sham equation takes on the form of

$$H\boldsymbol{\psi}_i(\mathbf{r}) = \left[-\frac{1}{2}\nabla^2 + \upsilon_{eff}(\mathbf{r}) \right] \boldsymbol{\psi}_i(\mathbf{r}) \qquad [34]$$

where H is the Hamiltonian operator and the effective potential $\upsilon_{eff}(\mathbf{r})$ is the sum of the external, Hartree (electrostatic), and exchange correlation potentials[14] and is represented by

$$\upsilon_{eff}(\mathbf{r}) = \upsilon_{ext}(\mathbf{r}) + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' + \upsilon_{xc}(\mathbf{r}) \qquad [35]$$

One can see that the effective potential, $\upsilon_{\text{eff}}(\mathbf{r})$, depends on the charge density $\rho(\mathbf{r})$ defined in equation [31] as the sum of the squared wavefunctions that are determined as eigensolutions of [34].[14]

To more clearly understand the flow of iterative calculations to generate the self-consistent solutions of the equations above, one starts with a guess (estimate) of the charge density $\rho(\mathbf{r})$

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} \left|\boldsymbol{\psi}_i(\mathbf{r})\right|^2 \qquad [36]$$

and determine the effective potential $\upsilon_{eff}(\mathbf{r})$ using external potential ($\upsilon_{ext}(\mathbf{r})$) and the exchange correlation approximation ($\upsilon_{xc}(\mathbf{r})$).

$$\upsilon_{eff}(\mathbf{r}) = \upsilon_{ext}(\mathbf{r}) + \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + \upsilon_{xc}(\mathbf{r}) \qquad [37]$$

To determine the new wave function $\Psi_i(\mathbf{r})$ it is necessary to calculate the Kohn-Sham eigenvalue equation using $\upsilon_{eff}(\mathbf{r})$ from [37].

$$H\boldsymbol{\psi}_i(\mathbf{r}) = \left[-\frac{1}{2}\nabla^2 + \upsilon_{eff}(\mathbf{r})\right]\boldsymbol{\psi}_i(\mathbf{r}) \qquad [38]$$

Once the new wave function $\Psi_i(\mathbf{r})$ is determined, the new charge density $\rho(\mathbf{r})$

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} |\boldsymbol{\psi}_i(\mathbf{r})|^2 \qquad [39]$$

and kinetic energy $T_s[\rho]$

$$T_s[\rho] = -\frac{1}{2}\sum_{i=1}^{N}\int \boldsymbol{\psi}_i^*(\mathbf{r})\nabla^2\boldsymbol{\psi}_i(\mathbf{r})d\mathbf{r} \qquad [40]$$

can be calculated. Using the above terms the ground state total energy functional $E[\rho]$ can be solved.[14]

$$E[\rho] = T[\rho] + V_{cc} + \int \rho(\mathbf{r})\upsilon_{ext}(\mathbf{r})d\mathbf{r} + \gamma_{ewald} \qquad [41]$$

$E[\rho]$ is minimized by repeating the sequence above starting at equation [36] through [41] until the lowest value is determined.

3.7 Sample Input File

DACAPO utilizes an input file to assemble all the parameters to perform an approximation of the electronic energy. It is important to review a basic input file to show how DACAPO receives all the input values to generate a total energy output, including relaxation dynamics to further optimize the geometric configuration of the complex. The system investigated in this thesis is far more complex than the CO dimer example in this section. But, the CO example will be sufficient to illustrate the fundamental framework of an input file. A more comprehensive review of DACAPO is found in Appendix I.

One of the simplest calculations is the total energy of a CO molecule. The minimal specifications that must be provided are a unit cell, geometry, planewave energy cutoff, and the number of electronic bands. Below is an example script from DACAPO of such a calculation followed by an explanation for each step.[15]

```
1    #!/usr/bin/env python
2
3    ##################################
4    #                                #
5    #     CO dimer in a box          #
6    #                                #
```

```
7         ###################################
8
9         from Simulations.Dacapo import *
10
11        sim=Simulation()
12
13        sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
14                                [0.0,   1.0,   0.0],
15                                [0.0,   0.0,    1.0]])
16
17        sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
18                               Atom(type=O, position=Vector([5.1, 4,    4]))],
19                               unitcell=8.0*sc_latt
20
21        sim.plancut = PlaneWaveCutoff(350)  #  in eV
22        sim.eband = ElectronicBands(10)
23
24        sim.Execute(outfile=”CO_in_a_box.nc”, ascii=CO_in_a_box.txt”)
```

Line 1 enables the user to run python as a shell language anywhere. Other than the first line, the “#” tells the system to the rest of the line is a comment enabling the user to place comments after statements, as shown in line 21. Line 9 calls out the functions from ‘Simulations.Dacapo’. Line 11 creates a python object ‘sim’ of data type Simulation. Line 13 is similar and the data type ‘Bravais Lattice’ represents a Bravais lattice. The initial value given to the Bravais Lattice is the unit matrix [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]. The default units are Å for length and fs for time. Line 17 sets up the atoms and their respective coordinates. Line 19 sets up the periodic boundary conditions, in the example above we apply a factor of 8 to the simple cubic. Another technique is setting the Bravais lattice dimensions (line 13) to [[8.0, 0.0, 0.0], [0.0, 8.0, 0.0], [0.0, 0.0, 8.0]]. Line 21 sets the plane wave basis kinetic energy cut off at 300 eV and line

22 sets the number of electronic valance bands in the calculation. Finally, line 24 is a function call, acting on the variable ‘sim’, that first causes all the attached variables to ‘sim’ to be written to an input file. The output file names called out in line 24 are of 2 types. The first is a binary netCDF formatted file (.nc) containing all important physical data (energies, forces, charge density, magnetic moments, etc.) and the second output file is an ASCII formatted readable log file (.txt) describing the course of the just performed calculation[15].

Typically, an investigator is interested in the total energy of the optimized geometry of a system. This can be done in DACAPO by adding a single line of code, as illustrated below by DACAPO in line 48.

```
25      #!/usr/bin/env python
26
27      ##################################
28      #                                #
29      #      CO dimer in a box         #
30      #                                #
31      ###################################
32
33      from Simulations.Dacapo import *
34
35      sim=Simulation()
36
37      sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
38                              [0.0,   1.0,   0.0],
39                              [0.0,   0.0,    1.0]])
40
41      sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
42                             Atom(type=O, position=Vector([5.1, 4,    4]))],
43                             unitcell=8.0*sc_latt
44
45      sim.plancut  = PlaneWaveCutoff(350)  #  in eV
```

```
46      sim.eband    = ElectronicBands(10)
47
48      sim.dyn       = Dynamics()
49
50      sim.Execute(outfile=”CO_relaxed.nc”)
```

Line 48 introduced the subroutine molecular dynamics which is employed to search for the minimum total energy configuration of the complex. Refer to Appendix I for more examples.

3.8 References

(1) Bahn, R., Jacobsen, K., *An object-oriented scripting interface to a legacy electronic structure code*, Computing in Science and Engineering, May/June, **(2002)**

(2) Leach, A., *Molecular Modeling, Principles and Applications*, Prentice Hall, **(2001)**

(3) Chadi, D., Cohen, M., *Special points in the Brillouin Zone*, Phys. Rev. B 12, vol. 8, pp. 5747 – 5188 **(1973)**

(4) Koch, Wolfram., Holthausen, Max., *A Chemist’s Guide to Density Functional Theory*, Wiley-VCH, **(2001)**

(5) Perdew, J., Burke, K., Ernzerhof, M., *Generalized Gradient Approximation Made Simple*, Phys. Rev. Lett. 18, vol 77, October **(1996)**

(6) Koch, Wolfram., Holthausen, Max., *A Chemist's Guide to Density Functional Theory*, Wiley-VCH, **(2001)**

(7) Andersen, O., *Linear methods in band theory*, Phys. Rev. B 12, 3060 **(1975)**

(8) Krakauer, H., Posternak, M., Freeman, A., *Linearized augmented plane wave method for the electronic band structure of thin films*, Phys. Rev. B 19, 1706 **(1979)**

(9) Payne, M., Teter, M., Allan, D., et al, *Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients*, Rev. Mod. Phys. 64, 1045 **(1992)**

(10) Bahn, R., Jacobsen, K., *An object-oriented scripting interface to a legacy electronic structure code*, Computing in Science and Engineering, May/June, **(2002)**

(11) Hammer, B., Norskov, J., *Why Gold is the noblest of all the metals*, Nature, vol . 376, no. 6537, pp. 238 – 240 **(1995)**

(12) Bendtsen, C., Nielsen, O., Hansen, L., *Solving large non-linear generalized eigenvalue problems from density functional theory calculations in parallel*, Applied Numerical Mathematics, vol. 37, nos. 1-2, p. 189 **(2001)**

(13) Hohenberg, P., Kohn, W., *Inhomogenous electron gas*, Phys. Rev. 136, B864 **(1964)**

(14) Bendtsen, C., Nielsen, O., Hansen, L., *Solving large non-linear generalized eigenvalue problems from density functional theory calculations in parallel*, Applied Numerical Mathematics, vol. 37, nos. 1-2, p. 189 **(2001)**

(15) Foukes, R., *Dacapo Tutorial*, Dec **(2004)**

Chapter 4

Approximating Interfacial Adhesion Energies

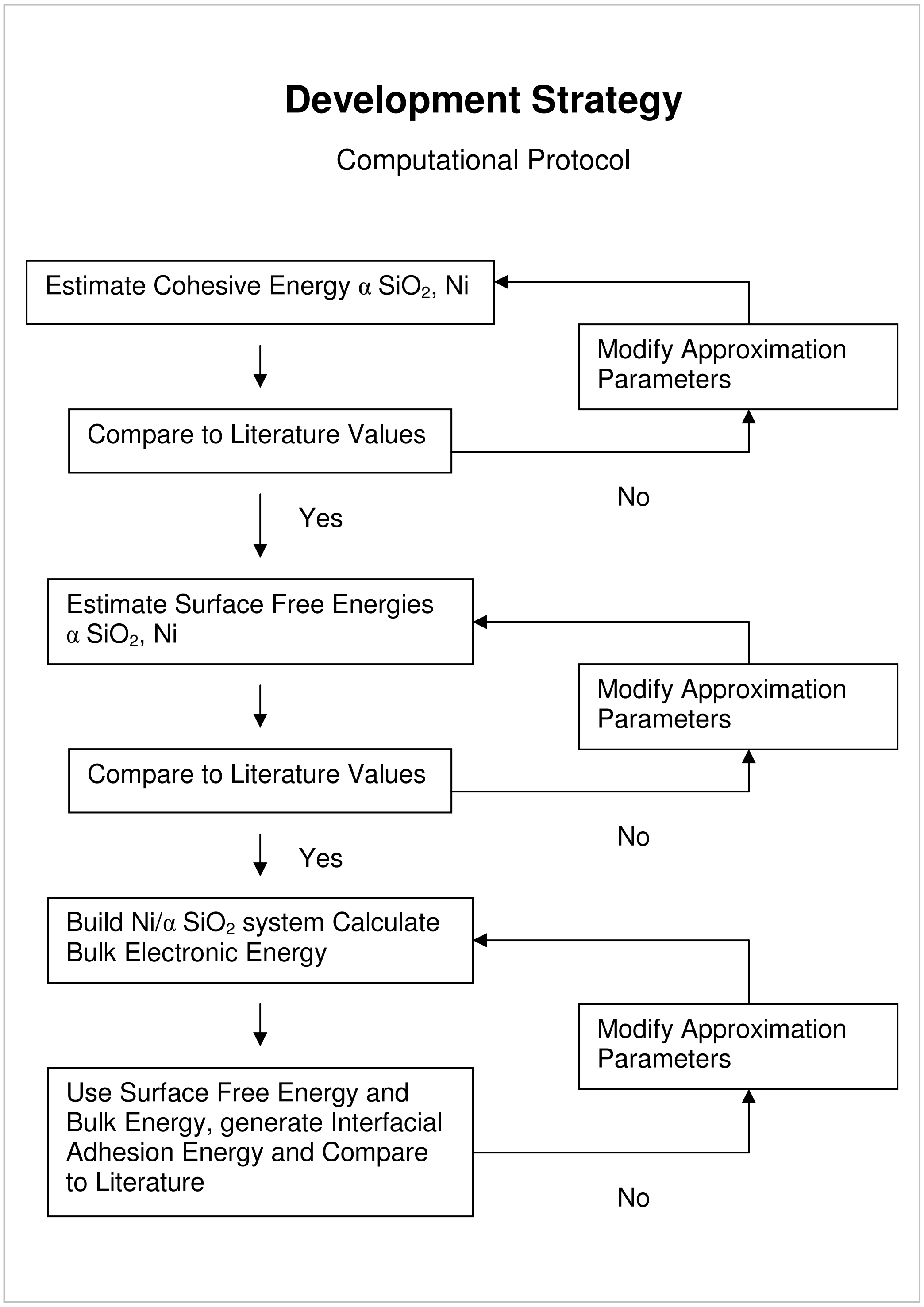
The DACAPO platform, which performs spin-polarized density functional theory calculations and expands the valence electron density using a planewave basis set[1], can be used to estimate the total electronic energy and corresponding cohesive properties. Generalized Gradient Approximation is used for the exchange correlation functional to best capture any changes in electron density. This is especially important, recognizing that we are studying transition metals and large oxides, which have a tendency to possess variations in electron density. First principles DFT calculations are ideal to characterize the atomic level behaviors of a metallic/oxide interface of a specific system.[2] DFT enables the inclusion many transition metals in the approximation by focusing on valence electrons only. This type of investigation can lead to understanding the atomic level causes of spallation at the nickel-oxide boundary layer. Since the exact structure of the interface is not well characterized experimentally,[2] an ideal approach will be very useful in identifying key elements that can enhance and/or weaken the interfacial adhesion energy between the Ni/TGO in the thermal barrier coating system.

The interfacial adhesion energy calculations consist of three steps:

1) **Bulk material calculations to approximate the bulk electronic energy and generate the cohesive energy.**[1] This is a critical step, as all further calculations build on the bulk electronic energy used to predict the cohesive energy. Also, the cohesive energy will be the reference value (experimentally and first principles) used to validate the computational protocol. The final protocol will be validated by comparing the interfacial adhesion energies experimentally known against the ones determined in this investigation (at the end of this section).

2) **Surface energy calculations, which use the bulk electronic energies from cohesive energy**[2]. Surface free energy is a primary input parameter in the determination of the interfacial adhesion energy and uses the bulk electronic energy (validated in the cohesive energy calculation in step 1) in it's generation.

3) **Interfacial adhesion energy produced from the surface free energy and bulk energy.**[1] This is illustrated in flow chart in Figure 8.


Development Strategy
Computational Protocol
Estimate Cohesive Energy α SiO2, Ni
Modify Approximation Parameters
Compare to Literature Values
No
Yes
Estimate Surface Free Energies α SiO2, Ni
Modify Approximation Parameters
Compare to Literature Values
No
Yes
Build Ni/α SiO2 system Calculate Bulk Electronic Energy
Modify Approximation Parameters
Use Surface Free Energy and Bulk Energy, generate Interfacial Adhesion Energy and Compare to Literature
No

[Figure 8] **Development Strategy Computational Protocol**

4.1 Cohesive Energy

It is clear density functional theory has progressed beyond the qualitative method and into the quantitative method with the introduction of gradient corrections to the local density approximation.[3] Perdew and Wang showed that the Generalized Gradient Approximation affords an accuracy average of < 4.0 kcal/mol with hydrocarbon and/or transition metal species.[3] This provides sound encouragement in accurately estimating the cohesive energies of Ni (Nickel) and α SiO_2 (Silica) and the complex thereof.

The cohesive energy is the first fundamental value reproduced, prior to moving on to surface free and interfacial adhesion energy calculations. Exhaustive effort/iterations went into approximating the cohesive energy of Ni, α SiO_2, and Ni/α SiO_2 complex. The general formula for cohesive energy[3] (E_{coh}) is captured in equation [42]

$$E_{coh} = -\left(E_{atom/mol} - \frac{E_{bulk}}{N}\right) \qquad [42]$$

Where $E_{atom/mol}$ is the total energy of one free atom or molecule, depending on how the system is structured. Silica (SiO_2) possesses 2 repeating units within a unit cell and allows the formula to be broken up within that framework. E_{bulk} is the total electronic energy of the complex and N is the number of atoms/molecules in our system.[3] Final cohesive results are shown in Table 4 below.

	Cohesive Energy (+/- 0.20 eV)	Cohesive Energy Literature (eV)
α SiO_2	26.93	22.42[a]
Ni	4.07	4.52[b]

[Table 4] **Cohesive Energy Approximation** (0.20 eV variance established from intentional bond length fluctuations) a = reference 4; b = reference 5

The cohesive energies are within 15% of literature values, consistent with findings of Carter et al.[1] This was very encouraging and initially supports the computational framework used in DACAPO. In addition, the cohesive values provide confidence in the bulk electronic energy, critical for the surface free energy.

4.2 Surface Free Energy

Typically, surface free energy (γ) of a solid surface is determined experimentally from the contact angle of dispersive and non dispersive liquids

$$\gamma = \gamma^{dispersive} + \gamma^{non-dispersive} \qquad [43]$$

where γ is the surface energy derived from the contact angle of the respective liquids. This widely accepted technique is somewhat empirical in nature, though proven very reliable.[1] To understand what type of atomic level contributions influence the surface free energy, one must resort to a first principles investigation. Surface free energy[3] (E_{surf}) can be defined as

$$E_{surf} = \left[\frac{\left(E_{slab} - N \times E_{fcc}\right)}{2 \cdot Area} \right] \qquad [44]$$

where E_{slab} is the bulk electronic energy of the slab (validated in the cohesive energy study in the previous section), N is the number of atoms/molecule, E_{fcc} is the pure electronic energy of the atom/molecule, and *Area* is the square of the Nickel lattice constant for the Ni (100) surface. Notice the factor of 2 in the denominator, this is present because when DFT calculates the bulk energy, it is representing the top and bottom surfaces. There is only one surface that comes in contact at the boundary layer for each material. Therefore, the area is divided by a factor of 2. Table 5 contains the result of α SiO_2 and Ni.

	E_{surf} Surface Free Energy (+/- 15 mJ/m^2)	E_{surf} Surface Free Energy Literature (mJ/m^2)
α SiO_2	176	179 - 191[a]
Ni	1360	2066[b]

[Table 5] **Surface Free Energy** (0.20 eV variance established from intentional bond length fluctuations) a = reference 15; b = reference 16

There is good agreement with α SiO_2 and slightly less with Ni. Experimentally measuring a pure, perfect Ni surface is very difficult due to the oxidation tendency of Ni. The next step is to move on to estimating the interfacial adhesion energy of the complex and validating against literature work done by Carter et al.[1]

4.3 Interfacial Adhesion Energy

The energetics of the physical interactions of two materials can be described by the work of adhesion, given by the Dupré equation

$$W_{ad} = \gamma_c + \gamma_m - \gamma_{cm} \qquad [45]$$

where γ_c, γ_m are the surface free energies of the ceramic (i.e. SiO_2) and metal (i.e. Ni) and γ_{cm} is the combined total energy of the complex. Applying this equation to a Ni/SiO_2 system results in:

$$W_{ad} = \frac{\left(E_{Ni:surf} + E_{SiO_2:surf} - E_{NiSiO_2}\right)}{Area} \qquad [46]$$

where $E_{Ni:surf}$ and $E_{SiO2:surf}$ are the surface free energies of Ni and α SiO_2 respectively, $E_{NI/SiO2}$ is the bulk electronic energy of the total system, and *Area* is the lattice constant squared.[1] An additional two layers of SiO2 was added to further test the computational protocol, see Table 6.

	1 layer SiO_2 (+/- 15 mJ/m^2)	2 layers SiO_2 (+/- 15 mJ/m^2)
Ni/α SiO_2 (This Study)	1220	1413
Ni/α SiO_2 (Carter et al)	1292[a]	1374[a]

[Table 6] **Interfacial Adhesion Energy** (0.20 eV variance established from intentional bond length fluctuations) a = reference 1

The data is consistent with the literature values generated through previous first principle investigations using DFT and GGA platform of VASP. These results provided confidence, especially when adding a second layer of α SiO_2 on the first layer was compatible with the established protocol. The next steps to consider are the different thermal expansions and α to β transition.

4.4 References

(1) Carter, E., Jarvis, E., The *role of reactive elements in thermal barrier coatings*, HPC and Nat. Sec., Mar/Apr, pp. 33 – 41 **(2002)**

(2) Jarvis, E,. Christensen, A., Carter, E., *Weak bonding of alumina coatings on Ni(111)*, Surf Sci, 487, pp. 55 – 76 **(2001)**

(3) Hanson et al., CAMPOS/DACAPO/Manual, Technical Univ. Sweden, **(2002)**

(4) Fowkes, F., J. Ind. Eng. Chem. *Attractive forces at interfaces*, 56 (12) pg 40 **(1964)**

(5) van Oss, C., Chaudhury, M., Good, R., *Interfacial lifshitz-vanderwals and polar interactions in macroscopic systems*, Chem. Rev. 88, pg 927 **(1988)**

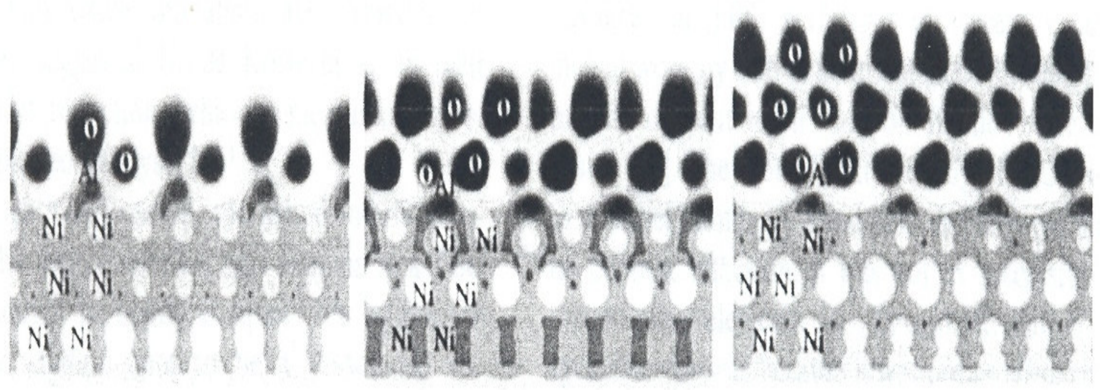
(6) Janczuk, B., Zdziennicka, A., *A study on the components of surface free energy of quartz from contact angle measurements*, J. Mat. Sci., 29 pp. 3559 – 3564 **(1994)**

(7) Vitos, L., Ruban, A., Skriver, H., Kollar, J., *The surface energy of metals*, Surf. Sci., 411 pp. 186 – 202 **(1998)**

Chapter 5

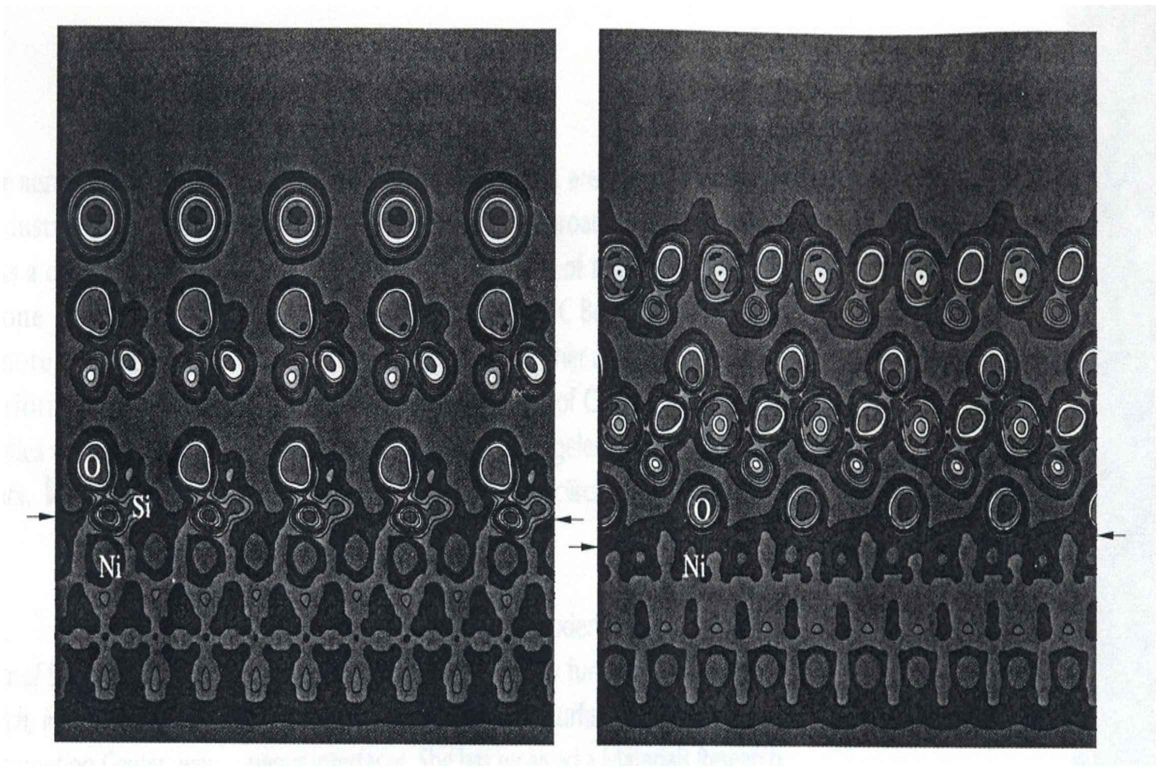
Ni/TGO Interfacial Adhesion Energy

The previous section established a first principles density functional theory protocol and initially validated by reproducing the literature values of Ni/α SiO_2 complex by Carter et al.[1] It is important to study the electronic tendencies of the Ni/TGO complex in greater detail with respect to the existing TGO (Al_2O_3) and the potential replacement (SiO_2). Recall, the primary weakness with Al_2O_3 as the TGO occurs when multiple layers are present and the electron density shifts from the surface atoms to the interior, bulk atoms.[2] As a result the interfacial adhesion energy is substantially reduced, initiating spallation at the interface. This is illustrated in Figure 9.



[Figure 9] **Electron Density for 0.5, 1.0, 2.0 layerss of Al_2O_3 on Ni.** Courtesy of Emily Carter and HPC and National Security, March/April 2002

Alternatively, SiO_2 interfacial adhesion energies remained unchanged from one to three layers, suggesting that it should not demonstrate the shift in electron density seen with Al_2O_3.[1] Figure 10 supports the stability of the electron density.



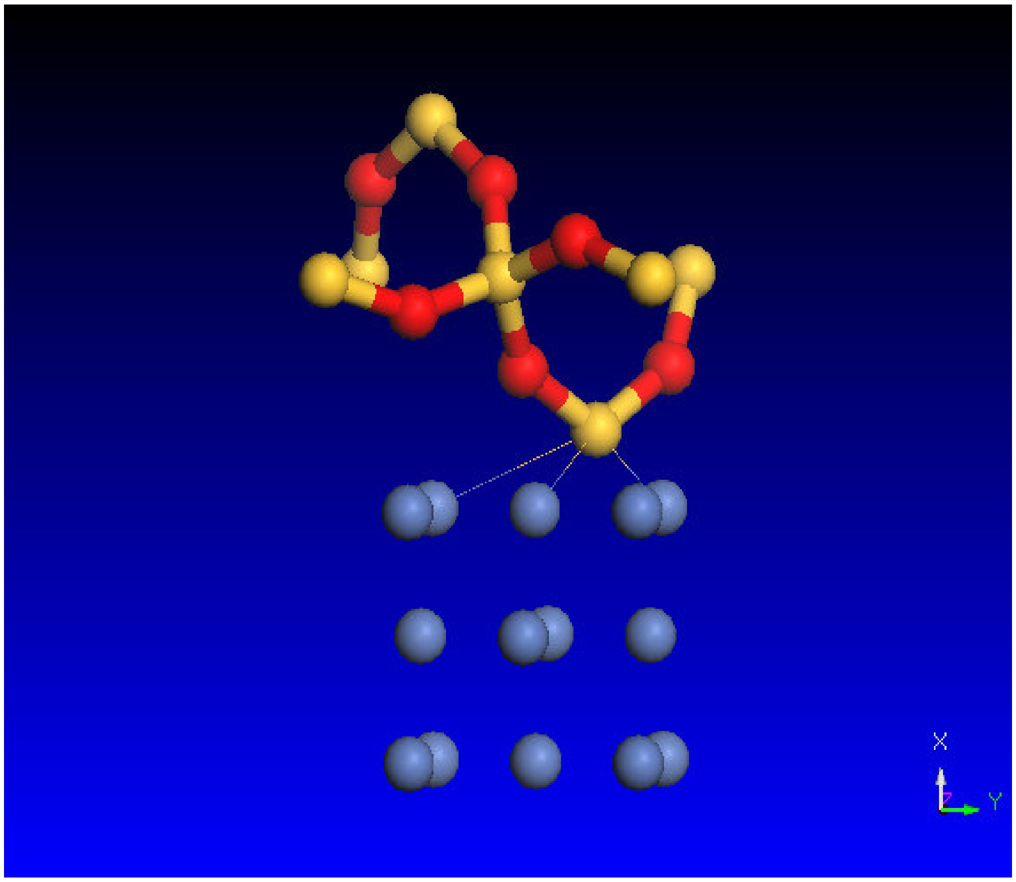
$\longrightarrow$ Represents the bonding region between Ni/SiO_2 interface. Notice the additional electron density apparent in the Ni/Si region compared to Ni/O region.

[Figure 10] **Electron Density Profile of 2 layers of SiO_2 on Ni.** Courtesy of Emily Carter and HPC and National Security, March/April 2002

The above Figures 9 and 10 illustrate the fundamental weakness in Al_2O_3 and an encouraging signal for the use of α SiO_2 as the thermally grown oxide (TGO).

5.1 Ni/TGO Complex

The system (Ni/SiO_2) studied focused on α SiO_2 and was somewhat flawed from the presence of the β phase of SiO_2 present. The target system of choice is β SiO_2 on Ni surface, illustrated in Figure 11 below.



[Figure 11] **2 layers of SiO_2 on Ni(100) Surface** Atoms: Red: Oxygen; Yellow: Silicon; Blue: Ni

This complex is slightly different than the system investigated by Carter et al, covered earlier in the introduction, and is considered the model system to investigate due to the predominate phase present during fuel combustion and operation.

5.2 Al_2O_3 vs. SiO_2

It is important to study the energetic properties of the three TGO's in this study. Comparing the interfacial adhesion energies, it is clearly evident that α and β SiO_2 are electronically more stable than Al_2O_3 in a multi-layer complex, shown in Table 7.

	1st layer (+/- 15 mJ/m^2)	2nd layer (+/- 15 mJ/m^2)
Ni/Al_2O_3	934	456
Ni/α SiO_2	1220	1413
Ni/β SiO_2	1041	1316

[Table 7] **Interfacial Adhesion Energy** (variance of 15 mJ/m^2 was determined by intentional bond length changes)

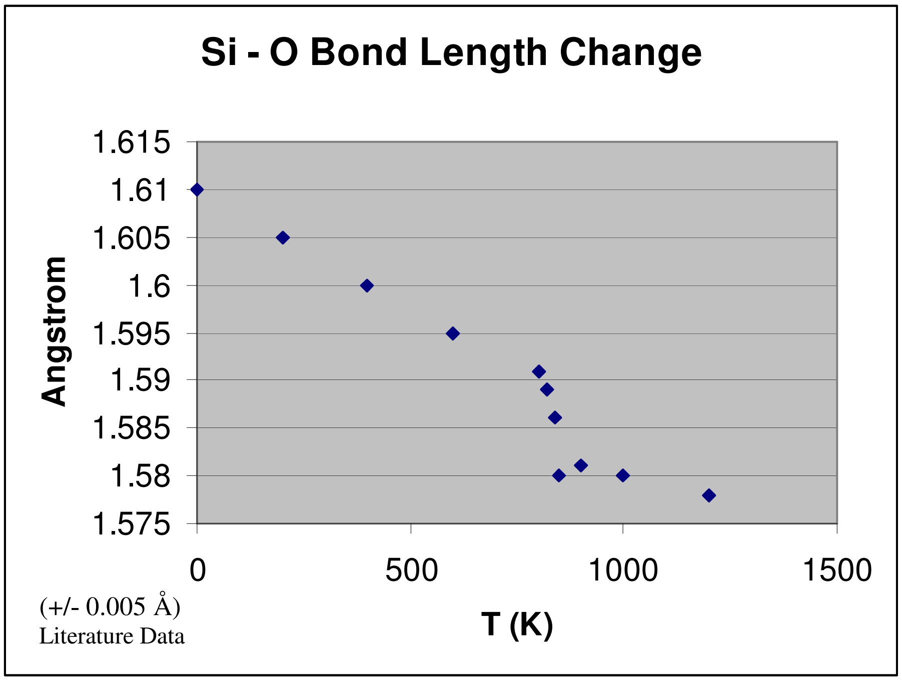
It is important to consider additional factors which can impact the TGO when in operation. These include the phase change between α, β SiO_2 and thermal expansion. Both can impact the electronic properties of the complex, influencing the interfacial adhesion energy.

5.3 SiO_2 Concerns

SiO_2 and the Ni/SiO_2 interface possess some unique characteristics which include: geometric inversion between the alpha and beta phases,[3] lattice mis-match influence, and thermal expansion. These three factors must be studied to properly access the viability of utilizing SiO_2 as the TGO.

5.3.1 α – β Transition

One of the key issues to consider in SiO_2 is the phase change that occurs within the TGO due to the elevated temperatures present in the gas turbine engine. The TGO (i.e. SiO_2) will be subjected to temperatures ranging from 25 to 1300 °C, forcing the SiO_2 to transition from α to β phase. It is understood that the inversion point of the phase change ("hop point") is relatively unknown and poorly characterized due to the instantaneous solid phase change.[3] Understanding the impact of the bond length change of Si – O in SiO_2 complex as it approach's the "hop point" is critical for mimicking the behavior. Also, understanding the electronic tendencies of β phase SiO_2 at elevated temperatures is just as important. Figure 12 illustrates the bond length changes with respect to temperature for alpha and beta SiO_2.



[Figure 12] **Si – O Bond Length Change** data provided by reference 3.

The Si – O bond length in α SiO_2 possesses a gradual decrease in length as the temperature approaches the "hop point". Conversely, the beta phase bond length is relatively unchanged. Therefore, studying the change experienced in the alpha phase to the "hop point" and the behavior of beta SiO_2, should provide a good understanding of any influences which can impact the interfacial adhesion energy through the phase transition.

5.3.2 Thermal Expansion

Thermal expansion is a factor to seriously consider. This is accomplished by expanding the complex identical to the bond length changes experienced with increasing temperature (illustrated in figure 12) and plotting the potential energy surface (PES). From the potential energy surface, the cohesive, surface free, and interfacial adhesion energies can be determined. Investigation into the thermal expansion coefficients showed less difference between the Ni and SiO_2 compared to Al_2O_3, shown in Table 8.

	500 °C (10^{-6}, °C^{-1})	1000 °C (10^{-6}, °C^{-1})
Ni	16.3[a]	17.5[a]
Al_2O_3	9.2[c]	10.2[a]
α SiO_2	19.4[b]	NA
β SiO_2	NA	14.6[b]

[Table 8] **Thermal Expansion Coefficient** a,=reference 4, b=reference 5

If two adjacent layers possess different coefficients of thermal expansion, the interface between the layers can be subjected to increased stress, potentially reducing the interfacial adhesion energy.[2]

5.4 References

(1) Carter, E., Jarvis, E., The *role of reactive elements in thermal barrier coatings*, HPC and Nat. Sec., Mar/Apr, pp. 33 – 41 **(2002)**

(2) Jarvis, E,. Christensen, A., Carter, E., *Weak bonding of alumina coatings on Ni(111)*, Surf Sci, 487, pp. 55 – 76 **(2001)**

(3) Tsuneyuki, S., Aoki, H., Tsukada, M., *Molecular-Dynamics Study of the α to β Structural Phase Transition of Quartz*, Phy. Rev. Lett. **64**, 7, pp. 776 – 779 **(1990)**

(4) Hidnert, P., Krider, H., National Bureau of Standards, **(1998)**

(5) Shaffer, P., Hausner, H., No 1 Materials Index, Plenum Press, NY, **(1964)**

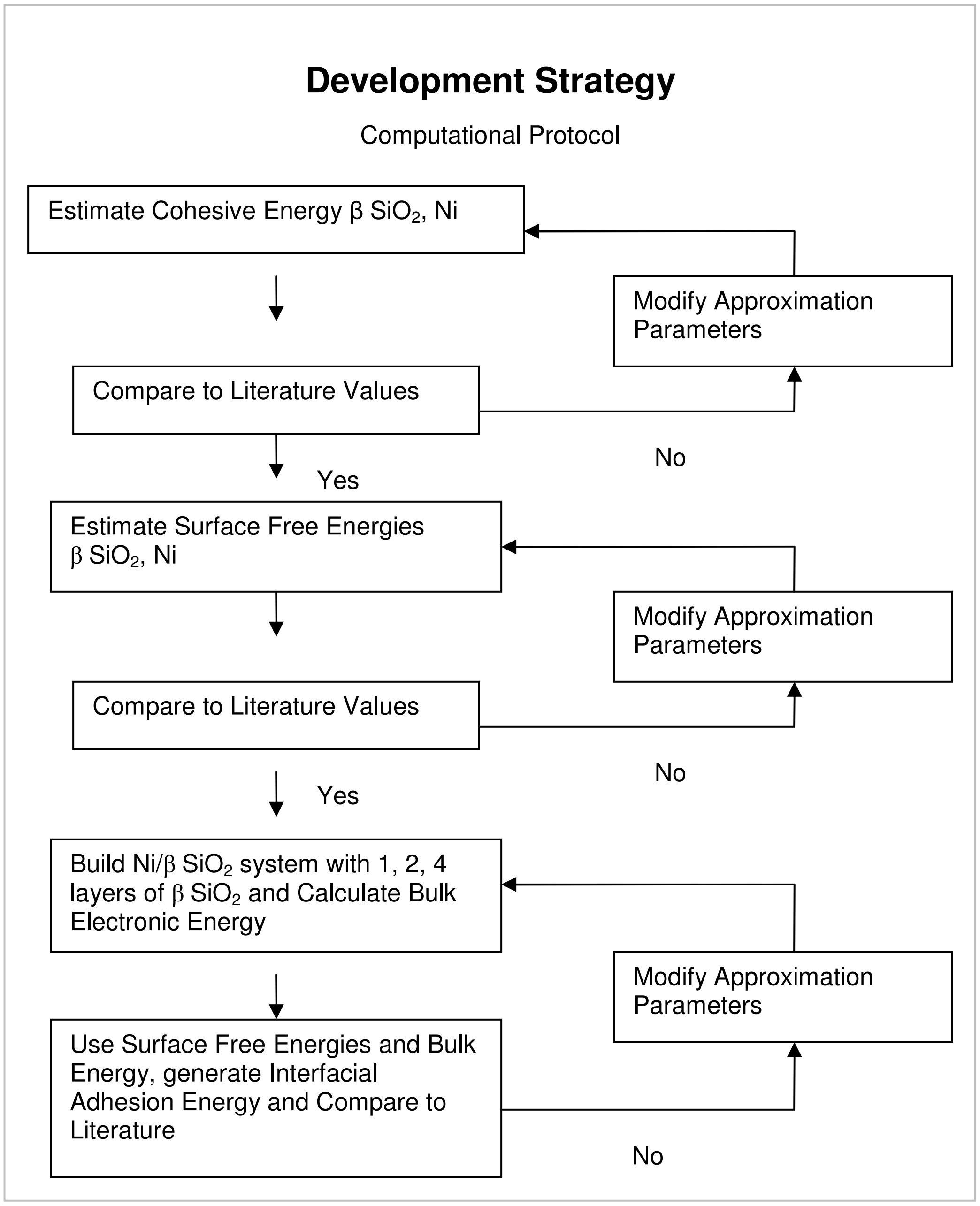
Chapter 6

Interfacial Adhesion Energy of Model Ni/SiO_2

The model TGO system, comprising of α, β SiO_2 on Ni (100) surface, is investigated using the protocol outlined in section 4 including the thought map to further refine the computational platform and parameters to ensure all atomic combinations are compatible. Within the protocol, the cohesive energy is approximated to establish a sound fundamental base for all energetic calculations, leading up to interfacial adhesion energy.[1] This section also captures the computational elements which are critical to generating valid, applicable electronic energies.[2]

6.1 Computational Protocol

To determine the finalized computational platform for our energetic calculations, which will include alternative geometric orientations, multi-layer oxide layer complexes, and expanded systems, a computational protocol is required. Figure 13 illustrates the computational protocol utilized for the model system of α, β SiO_2 on Ni (100) surface.


Development Strategy
Computational Protocol
Estimate Cohesive Energy β SiO_2, Ni
Modify Approximation Parameters
Compare to Literature Values
No
Yes
Estimate Surface Free Energies β SiO_2, Ni
Modify Approximation Parameters
Compare to Literature Values
No
Yes
Build Ni/β SiO_2 system with 1, 2, 4 layers of β SiO_2 and Calculate Bulk Electronic Energy
Modify Approximation Parameters
Use Surface Free Energies and Bulk Energy, generate Interfacial Adhesion Energy and Compare to Literature
No

[Figure 13] **Development Strategy Computational Protocol**

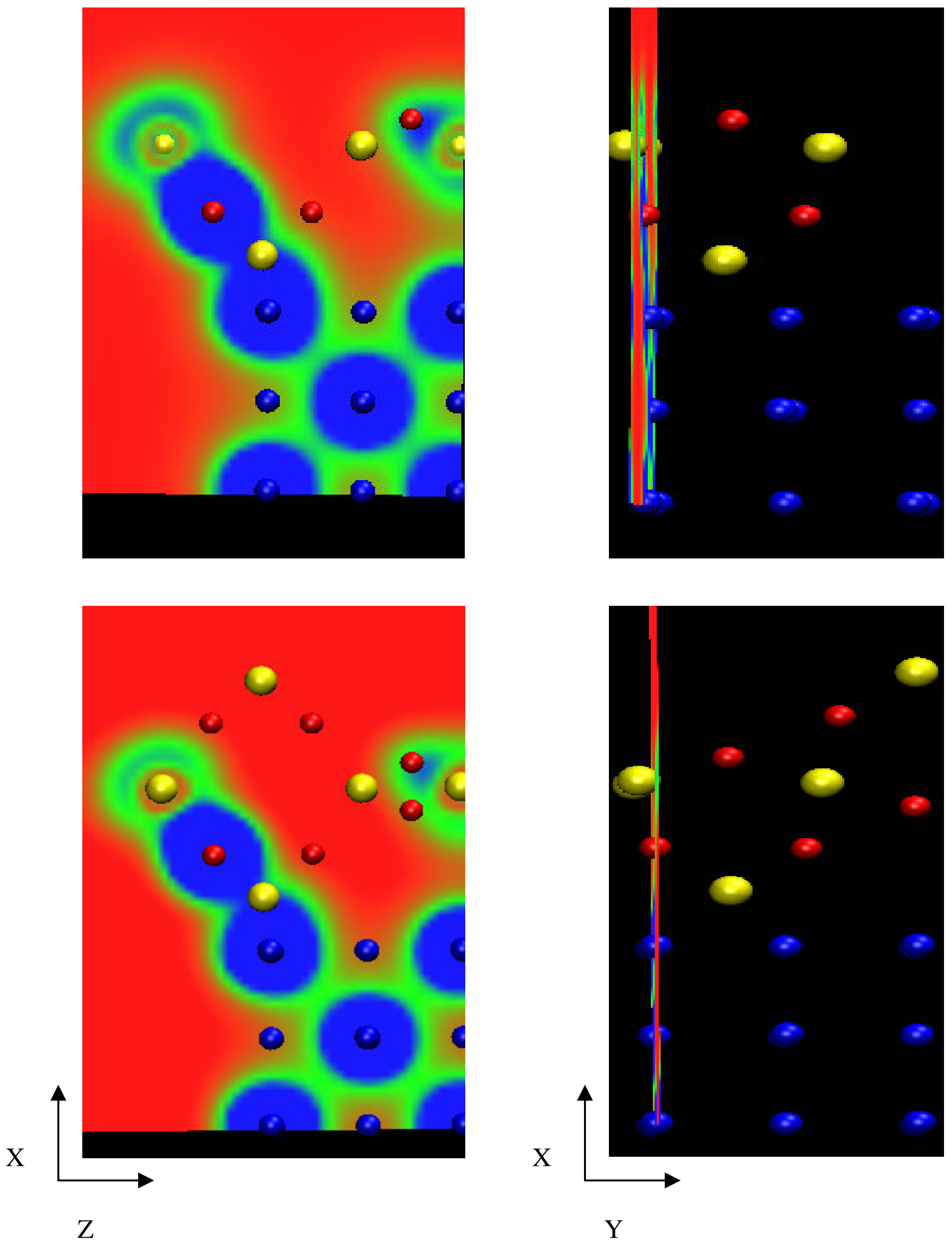
6.1.1 Results

The interfacial adhesion energies of multi layer complexes of β SiO_2 were successfully approximated using the protocol outline in the previous section. See appendix II for the specific input files. α SiO_2 was calculated to validate the changes to the computational framework (i.e. input file). The energetic data provided confidence in the computational construction, displayed in Table 9.

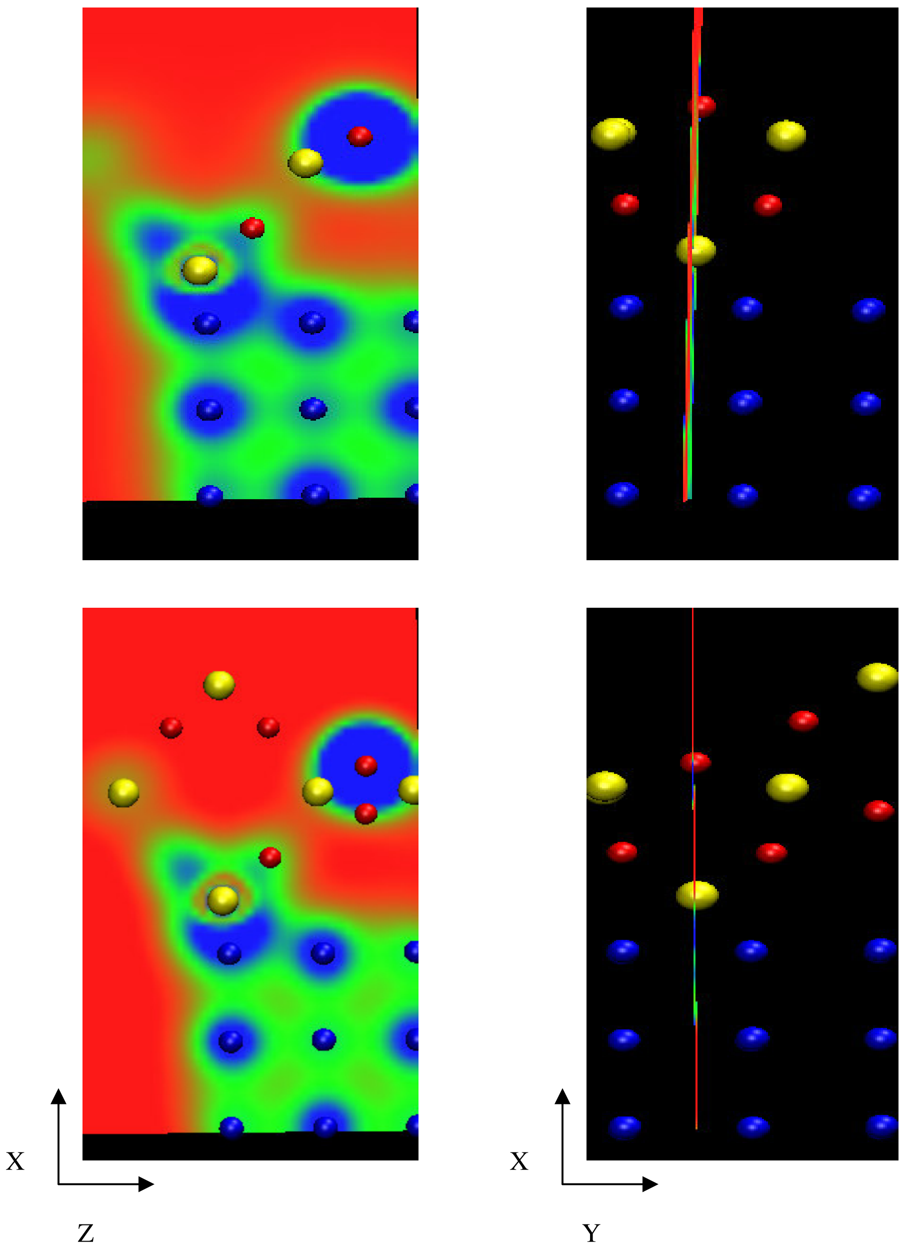
	1st layer (+/- 15 mJ/m^2)	**2nd layer** (+/- 15 mJ/m^2)	**4th layer** (+/- 15 mJ/m^2)	**r** (correlation coefficient)
Ni/α SiO_2	1218	1409	1487	0.9706
Ni/β SiO_2	1040	1314	1388	0.9482

[Table 9] **Interfacial Adhesion Energy** (variance of 15 mJ/m^2 was determined by intentional bond length changes)

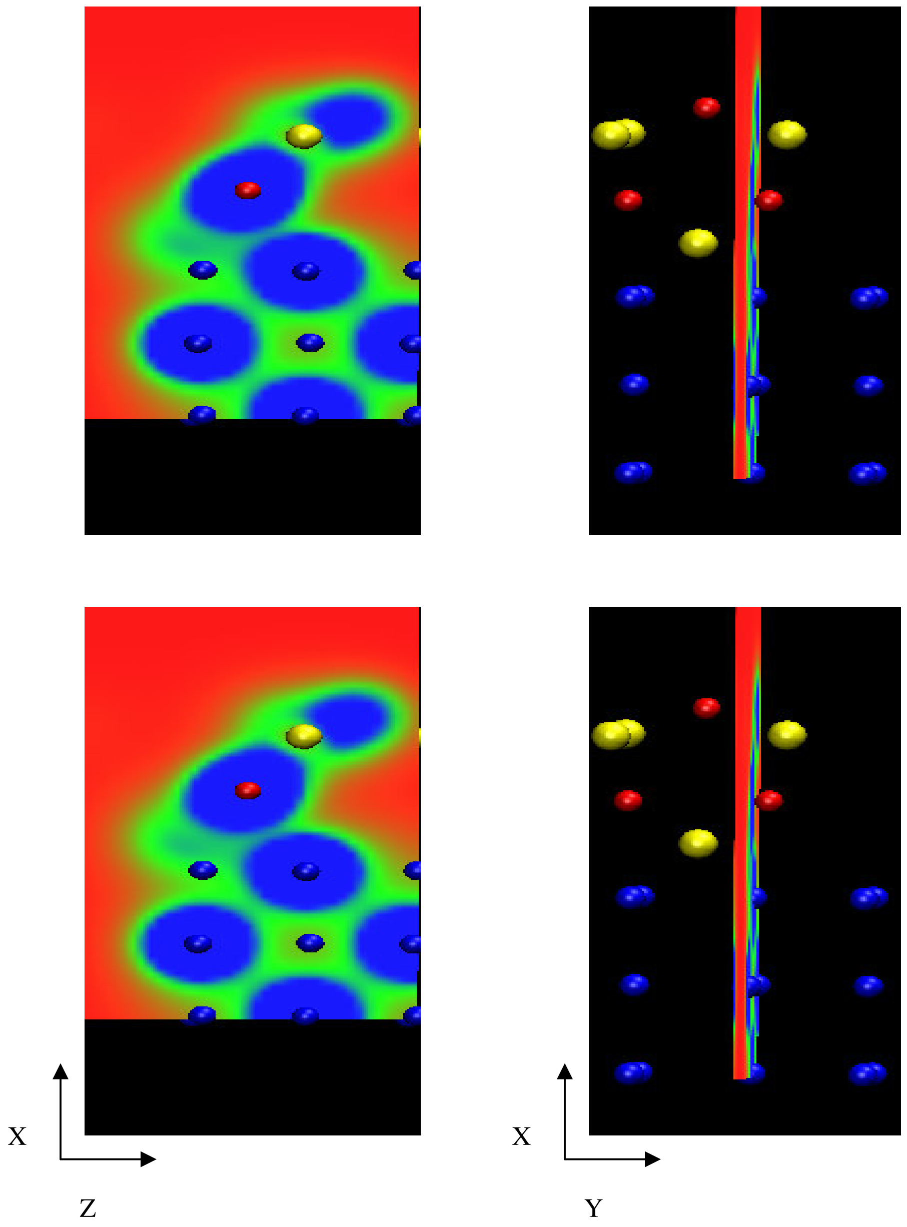
The energy data is consistent with earlier results for the interfacial adhesion energy and a degree of stabilization progressing through the 4th layer of SiO_2 on Ni. The data above suggests that the electron density of β SiO_2 does not shift from the interface atoms to the interior bulk atoms with multiple layers present, as opposed to the Al_2O_3 TGO. Plotting the electron densities of the Ni/β SiO_2 will allow for a more comprehensive evaluation of any change in electron density.



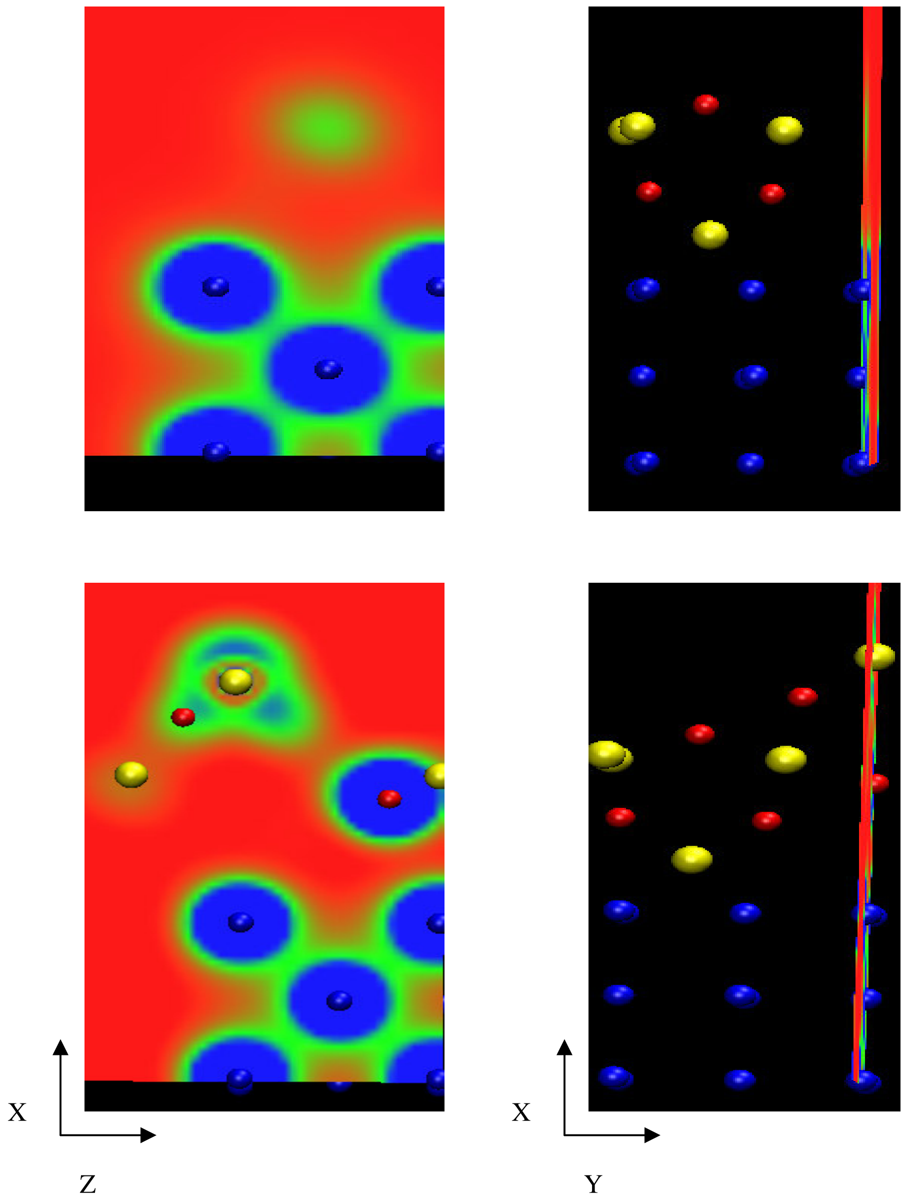
[Figure 14] **XZ-plane volume slice (0.40 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel
Density Color Scale: 0 = Red, 400 = Green, 800 = Blue



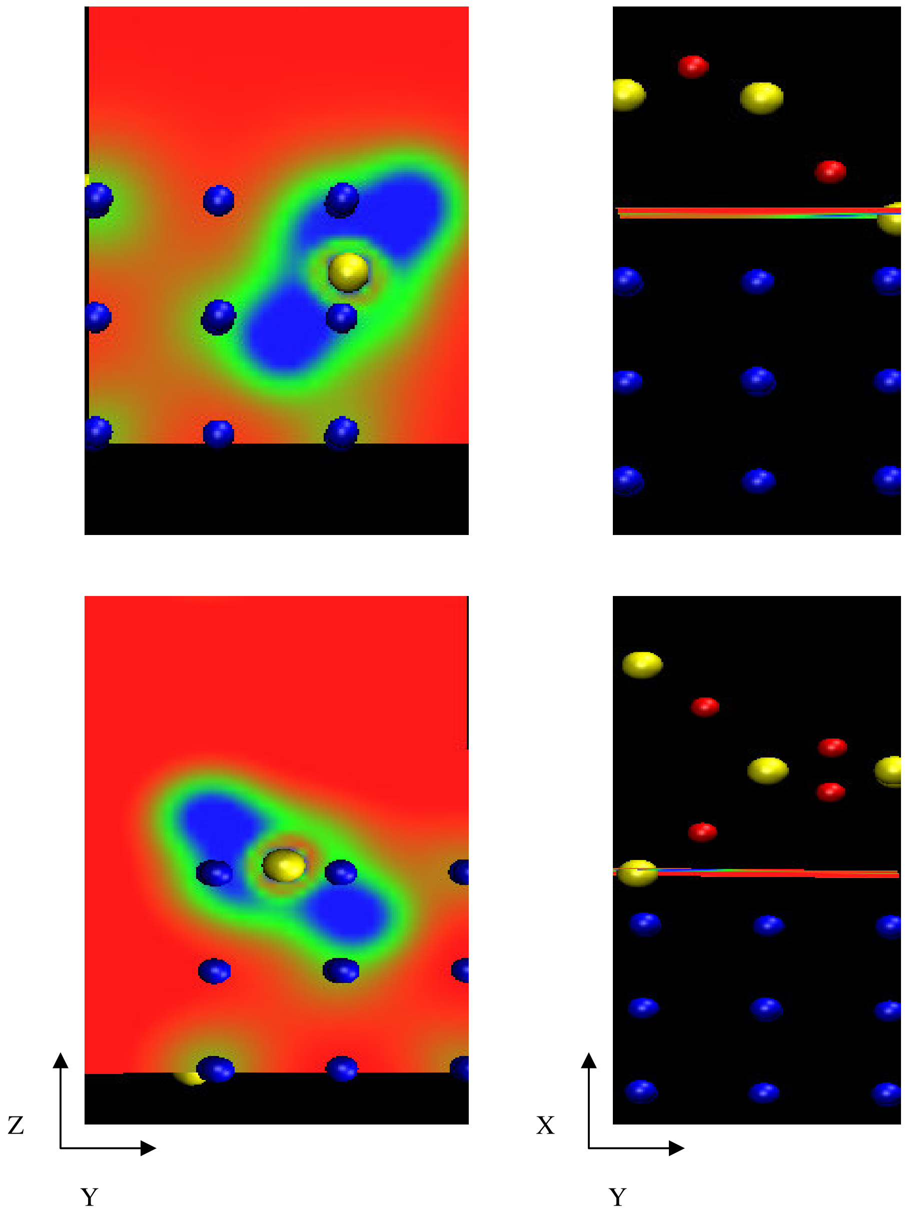
[Figure 15] **XZ-plane volume slice (0.25 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel Density Color Scale: 0 = Red, 400 = Green, 800 = Blue



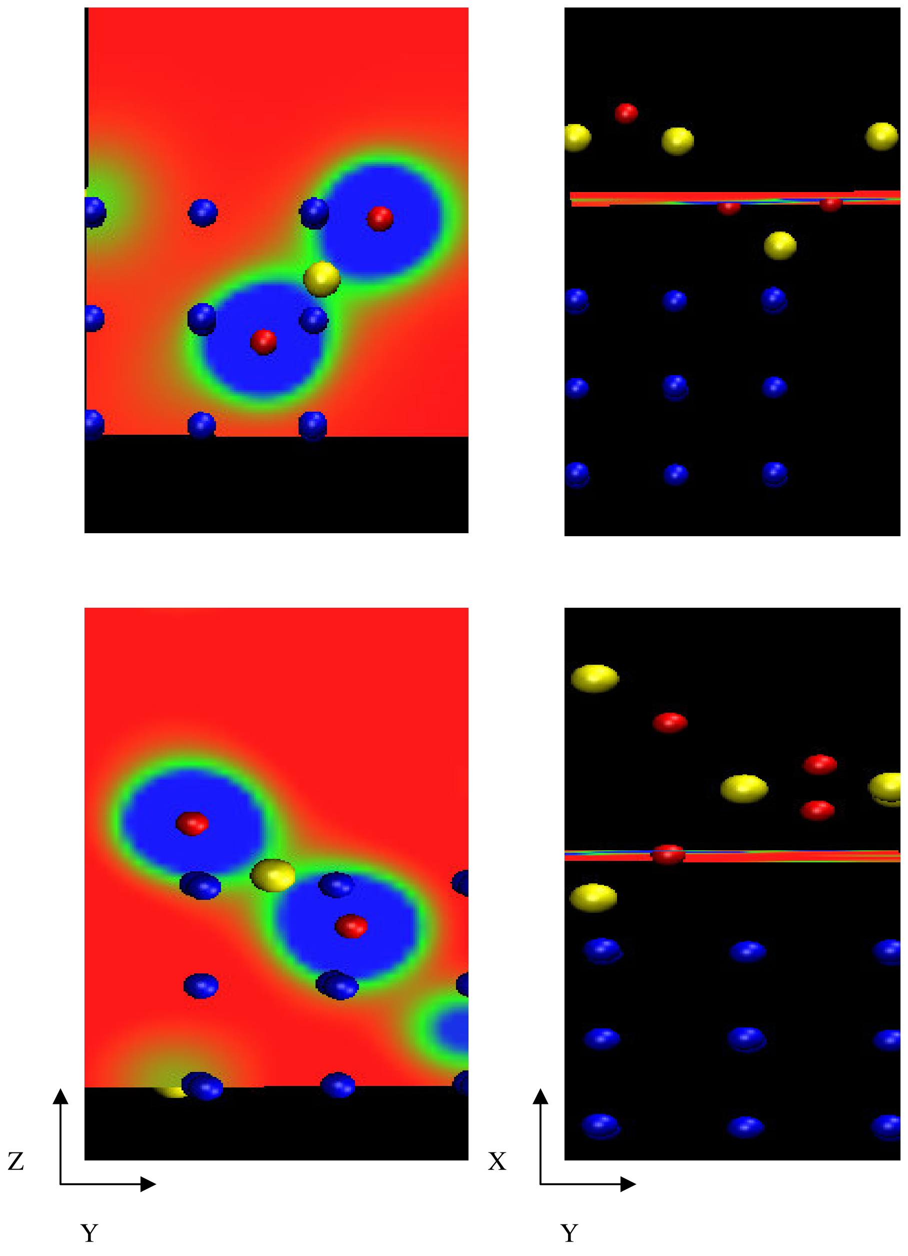
[Figure 16] **XZ-plane volume slice (0.18 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel Density Color Scale: 0 = Red, 400 = Green, 800 = Blue



[Figure 17] **XZ-plane volume slice (0.00 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel
Density Color Scale: 0 = Red, 400 = Green, 800 = Blue



[Figure 18]   **YZ-plane volume slice (0.40 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel
Density Color Scale: 0 = Red, 400 = Green, 800 = Blue



[Figure 19] **YZ-plane volume slice (0.34 offset)** Top figures are 1 layer SiO_2 on Nickel and bottom figures are 2 layers SiO_2. Right frames are slice orientation, left frames are density projection of plane slice. Spheres: Red – Oxygen; Yellow – Silicon; Blue - Nickel
Density Color Scale: 0 = Red, 400 = Green, 800 = Blue

The electron density plots in Figures 14 -19 were generated from DACAPO cube files which possessed the electron density of the unit cell. The electron density distributions were illustrated using VMD (Visualization Molecular Dynamics) version 1.8.5 from the University of Illinois. Figures 14 - 19, support the stability of the electron density in β SiO_2, building more confidence in SiO_2 as a potential replacement TGO. Figure 14 shows the electron density in the XZ-plane slice at an offset of 0.4, which is positioned on the side surface of the face centered cube of Ni. The slice intersects the 1st of the two lowest oxygen atoms directly above the nickel surface. The electron density overlap between the Ni and oxygen atoms is clearly visible and unchanged between the 1 and 2 layers of SiO_2. Figure 15, the XZ-plane slice is positioned at an offset of 0.28, where it intersects the lowest silicon atom. Again, no appreciable change in electron density overlap from 1 to 2 layers of SiO_2 is apparent. Figure 16, the 2nd lowest oxygen atom is intersected at an offset of 0.18 with significant density overlap between the Ni and oxygen atoms. The last XZ-plane slice (Figure 17) is at the origin of the lattice (offset 0.0) and showing no electron density interaction between the Ni surface and oxygen atom due to the increased distance (> 2.0 Ǻ) between the surface and oxygen atom. Figures 18 and 19 are YZ-plane slice and show the interaction between the silicon and oxygen atoms within the SiO_2 complex.

The electron density stability of the complex is illustrated in all the figures above (Figures 14 – 19) showing little change from one and two layers of β SiO_2 at the Ni (100) interface.

6.2 Computational Construction Elements

The finalized computational parameters used in determining the electron density plots were validated through the cohesive energy calculations and further supported with the surface free energy values. The parameters established via the computational protocol can be used for additional studies in our system. These elements include bravais lattice, planewave cutoff, k-points, pseudopotential, exchange-correlational functional, electronic minimization, electronic bands, and convergence control. See Table 10.

Parameter	**Value**
Bravais Lattice	10 – 20 X 8 X 8 Å (+/- 1)
Planewave Cutoff	377 eV (+/- 20)
k-points	54 (+/- 9) Chadi – Cohen
Pseudopotential	Perdew Wang 1991
Exchange-Correlation Functional	PW91 (Perdew Wang 1991 GGA-parameterization)
Electronic Minimization (Eigen Solution)	Block Davidson Algorithm
Electronic Bands	10 – 20 greater than 50% of total electrons in system
Convergence Control	< 0.05 eV/atom

[Table 10] **DACAPO Parameters**

The parameters in Table 10 were critical components in the energetic calculations of the cohesive, surface free, and interfacial adhesion energies.

6.3 Summary

It was successfully shown that the interfacial adhesion energy between the Ni(100) and α, β SiO_2 are relatively unchanged from the 1st layer to the 4th layer, mimicking the growth of the SiO_2 layer (TGO) over time. Recall this was the fundamental flaw with Al_2O_3 at the TGO, catalyzing spallation (delamination) of the Oxide layer from the Ni alloy. Furthermore, the electron density was stable at the interface atoms of the β SiO_2/Ni complex in mono and bi-layers of SiO_2, supporting the interfacial adhesion energy trends. It is important to consider additional factors experienced in the thermal barrier coating system prior to any definitive conclusions on the viability of utilizing SiO_2 as the TGO in a TBC.

6.4 References

(1) Bahn, R., Jacobsen, K., *An object-oriented scripting interface to a legacy electronic structure code*, Computing in Science and Engineering, May/June, **(2002)**

(2) Bendtsen, C., Nielsen, O., Hansen, L., *Solving large non-linear generalized eigenvalue problems from density functional theory calculations in parallel*, Applied Numerical Mathematics, vol. 37, nos. 1-2, p. 189 **(2001)**

Chapter 7

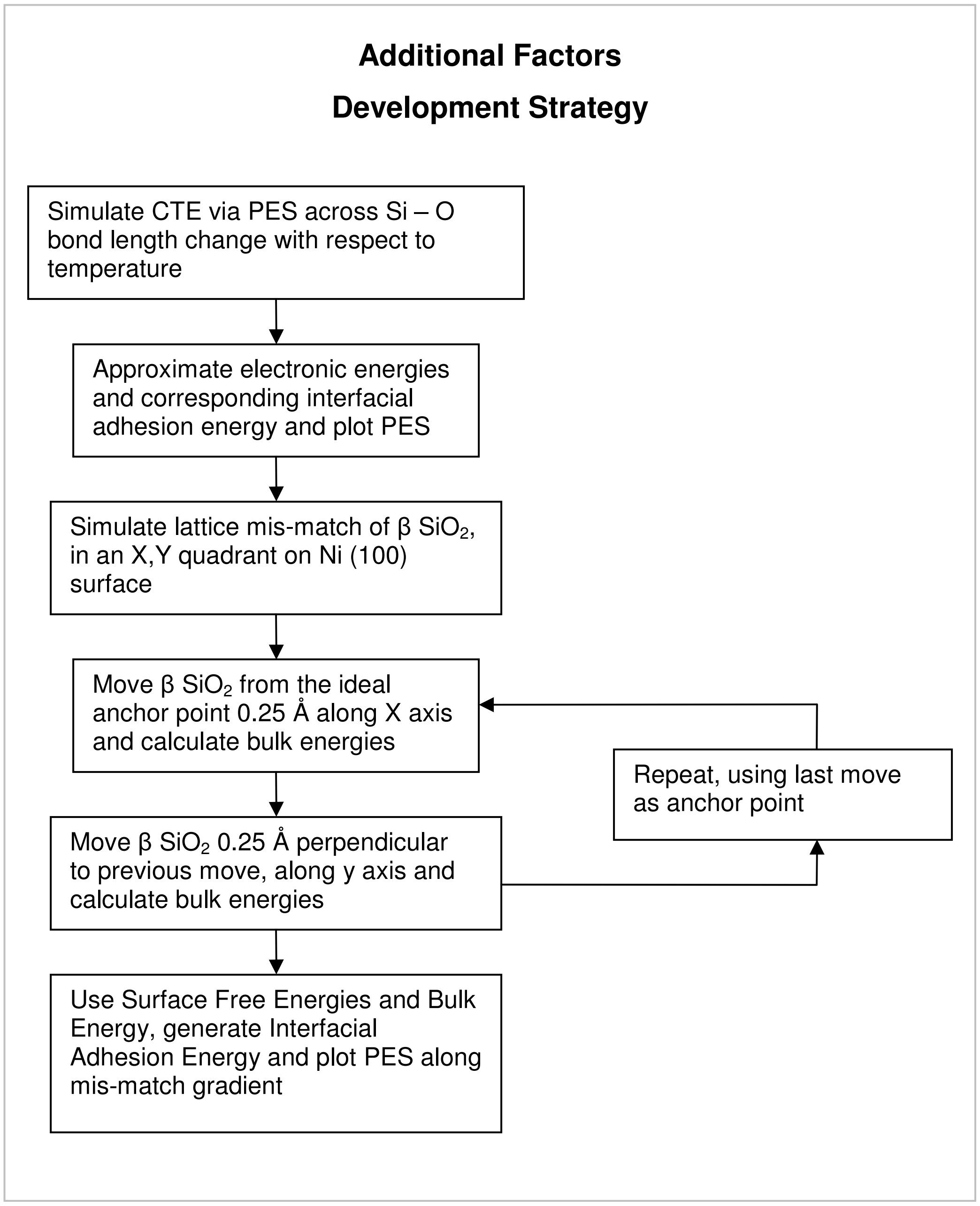
Additional Factors to Consider

Chapter 6 produced evidence in supporting the use of silica (SiO_2) as a TGO replacement to Al_2O_3. Based upon electron density we showed the stability of the interface increased going from one to four layers of SiO_2 on Ni (100) surface. This data was an incomplete assessment of SiO_2 considering the many scenarios which can impact the assembly/orientation of the interface atoms. These include the coefficient of thermal expansion (CTE) between the two adjacent materials (Ni/SiO_2), α - β phase transition for SiO_2, and lattice mis-match. Literature studies on thermal barrier coatings (Carter el al´) did not take these elements into consideration.[1]

Thermal expansion can be studied by progressively calculating the cohesive, surface free, and interfacial adhesion energy along the known Si – O bond length change.[2] Using the data one can plot the potential energy surface (PES) of these values to see if there are any deleterious barriers or changes that can negatively affect the interfacial adhesion energy. Understanding the impact of the α, β phase change is accomplished by studying the behavior of each phase before and after the "hop point".[2] Finally, understanding how the two materials (Ni/SiO_2) match up at the boundary layer is accomplished by considering the lattice mis-match conditions and mapping out the PES.

7.1 Protocol

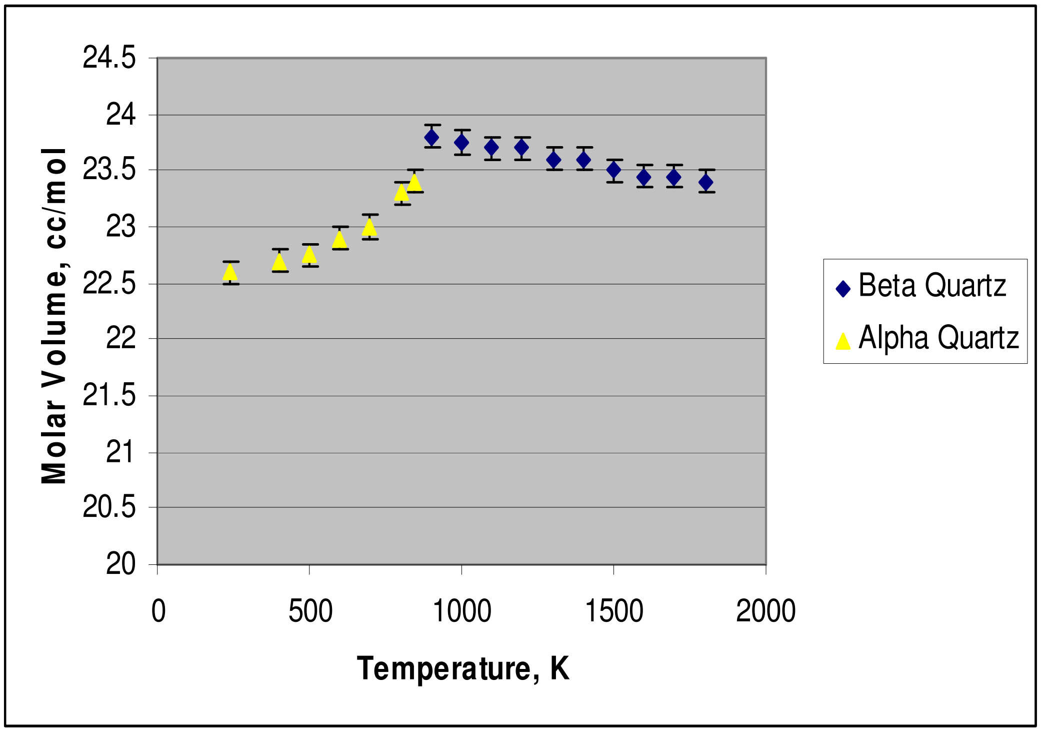
The computational protocol/strategy is adjusted to study how thermal expansion, SiO_2 phase change, and lattice mis-match influence the interfacial adhesion energy. Figure 20 is a consolidated flow chart removing the redundancy from the previous flow charts (Figures 8 and 13).


Additional Factors
Development Strategy
Simulate CTE via PES across Si – O bond length change with respect to temperature
Approximate electronic energies and corresponding interfacial adhesion energy and plot PES
Simulate lattice mis-match of β SiO_2, in an X,Y quadrant on Ni (100) surface
Move β SiO_2 from the ideal anchor point 0.25 Å along X axis and calculate bulk energies
Move β SiO_2 0.25 Å perpendicular to previous move, along y axis and calculate bulk energies
Repeat, using last move as anchor point
Use Surface Free Energies and Bulk Energy, generate Interfacial Adhesion Energy and plot PES along mis-match gradient

[Figure 20] **Additional Factors Development Strategy**

7.2 Coefficient of Thermal Expansion

One technique that captures thermal expansion is the change in molar volume of a material. Figure 21 shows the behavior of α, β SiO_2 from 200 - 2000 °C.

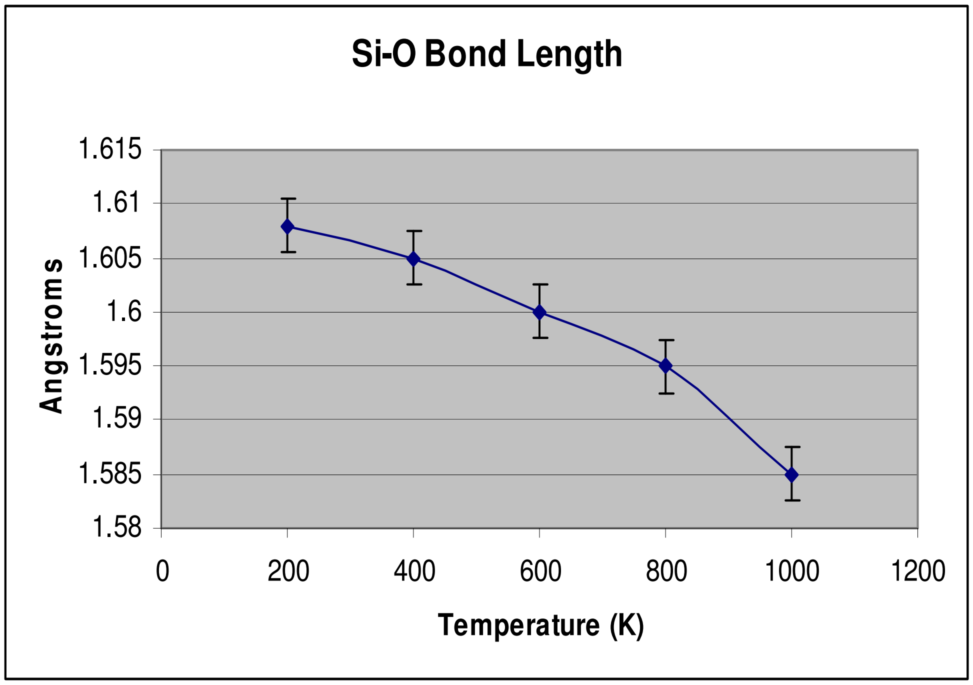


[Figure 21] **Expansion of α, β SiO_2 (Quartz)** Literature data, reference 2 (Quartz in this figure is identical to SiO_2 in this thesis)

The volume change in the α phase from 250 – 800 °C is substantial. This equates to approximately a 7% unit volume change[3] which must be considered in our thermal barrier coatings system. Alternatively, the β SiO_2 is relatively stable with respect to volume change from 800 – 1800 °C. To accurately study the influence of this unit cell expansion, it is important to characterize the change in Si – O bond length with respect to temperature in the alpha configuration.

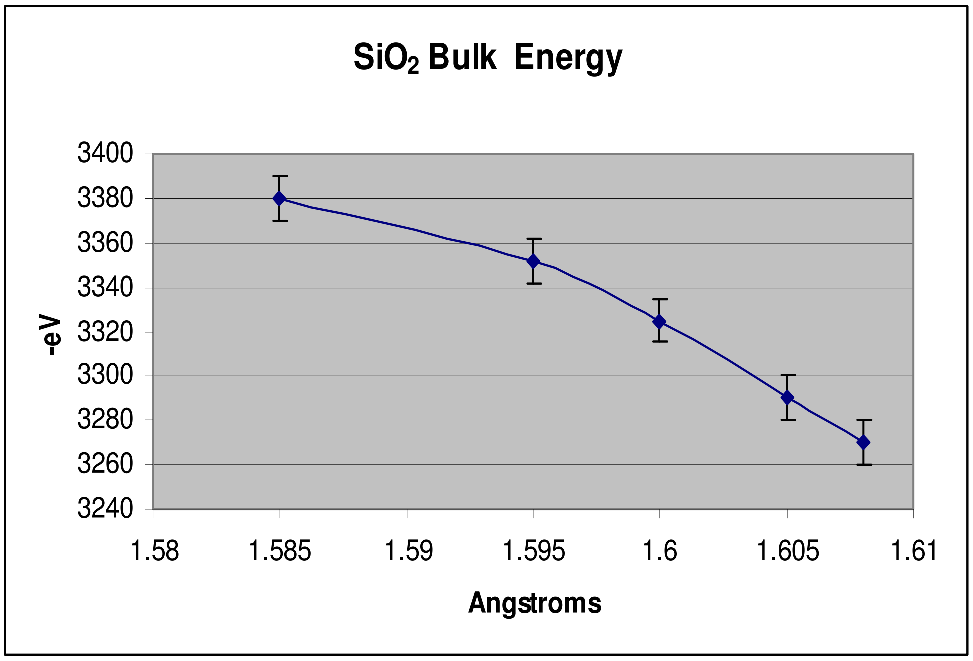
7.2.1 PES along Si — O Bond Length Change

The Si – O bond length change in α SiO_2 is practically linear with increasing temperature[5] up to the α – β inversion point, shown in Figure 22. To understand the PES it is necessary to calculate the electronic energies at each points along the progression of the bond length change as the temperature changes.



[Figure 22] **Si – O Bond Length Change** (Data plotted from reference 3, with a **r** of 0.97)

Incorporating these bond lengths into the α SiO_2 complex to generate the bulk electronic energies allows us to understand the electronic energy trend as the temperature is increased. The unit cell was electronically minimized at each bond length to produce the PES. Figure 23 shows that the highest energy is at the shortest bond length while the lowest energy is at the largest bond length. Suggesting the electronic properties of the system are changing with increasing temperature.



[Figure 23] **SiO_2 Bulk Energy (r** = 0.98)

The cohesive, surface free, and interfacial adhesion energies can be calculated and their relationship to temperature change and bond length can be analyzed. Table 11 shows the highest and lowest electronic energy for α SiO_2 is at 1.585 Å and 1.608 Å respectfully. Recall, that the bond length for β SiO_2 does not appreciably change from 800 to 1800 °C, therefore it is reasonable to use the control bond length (1.582 Å) to represent all temperatures in the beta phase. Using the bulk energy values from Figure 19, the PES of cohesive, surface free, and interfacial adhesion energies are produced to understand any influences the changing Si – O bond length has when the temperature is increased in the combustion chamber. Table 11 nicely illustrates the PES with respect to the bond length changes.

The data shows the cohesive energy decreasing as the Si – O bond length reduces from 1.608 Å (control bond length) to 1.585 Å. At 1.585 Å the α SiO_2 has a cohesive energy of -3.02 eV, very unstable, as a result the complex rearranges it's atomic configuration to form the β $SiO_{2.}$ At essentially the same bond length the beta configuration possesses a cohesive energy of -26.48 eV, many orders of magnitude higher than the alpha configuration. Throughout all the bond length changes, the surface free and interfacial adhesion energies are minimally impacted, suggesting a stable interface boundary layer during the heat cycle of the system. It is also important to note that the correlation coefficient of the data series was comfortably above 0.95.

Si – O Bond Length [Å]	Cohesive Energy [-eV] (STDEV: 0.63) (Sig F: 0.024) (+/- 0.2)	Surface Free Energy [-eV] (STDEV: 0.63) (Sig F: 0.004) (+/- 0.2)	Interfacial Adhesion Energy [mj/m^2] (STDEV: 5.5) (Sig F: 0.029) (+/- 15)
1.608 (α *)	26.9	161.0	1413
1.605 (α)	26.4	161.5	1363
1.600 (α)	13.5	162.2	1351
1.595 (α)	5.1	162.9	1338
1.585 (α)	3.0	163.5	1327
r^2	0.86	0.95	0.84
1.582 (β*)	26.48	164.56	1316

* Ground state energy bond length

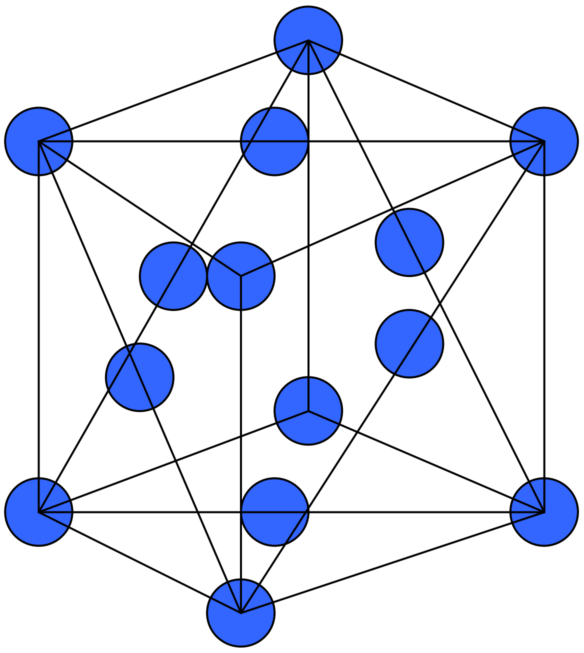
[Table 11] **Interfacial Energies relative to Si – O bond length Change** (r2 in table for each data array is with respect to bond length change, the standard deviation of fit 0.63, variance of +/- 0.2, and a significance of 0.004 – 0.029)

7.3 Lattice Mis-Match

It is commonly known when two materials come together, the tendency to not position all interface atoms ideally is more common than not, especially when you have thermal expansion of one layer different than the other.[5] Up until now, the electronic studies have been based on ideal positioning of α and β SiO_2 on Ni surface. Intentionally shifting the anchor point atoms that directly connect to the Ni surface can closer mimic the "real" interface. This is accomplished by shifting the SiO_2 complex along the *X*, *Y*, axis on the surface of Ni face, keeping the Z axis position constant. Next, uniformly shifting the whole complex in 0.25 Å increments along the *X* axis followed by another 0.25 Å move 90 ° along the *Y* axis and repeating the sequence again should allow the PES to be mapped out.

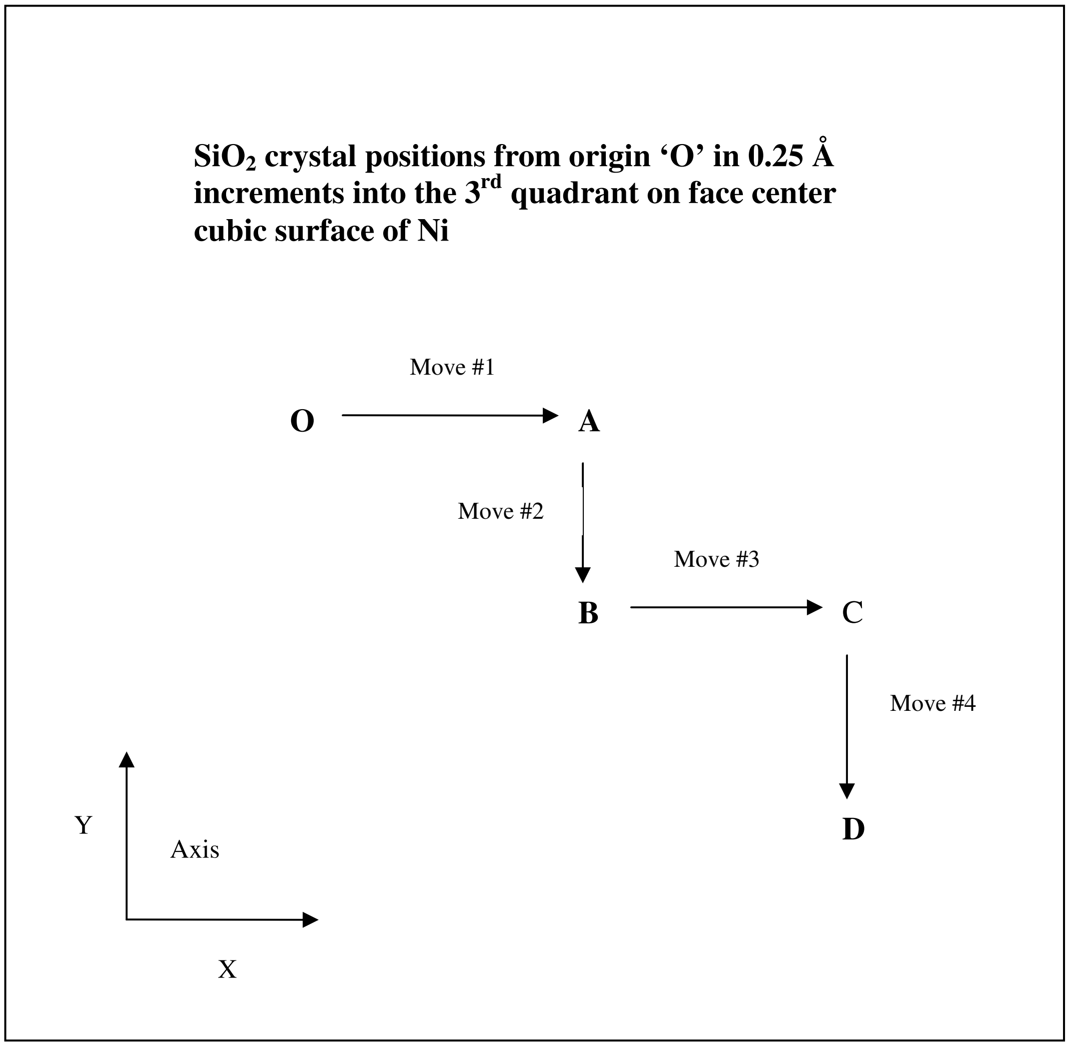
7.3.1 PES across Ni surface

Nickel crystal possesses a face center cubic orientation and the (100) surface possesses four symmetrical quadrants[8], illustrated in Figure 20.



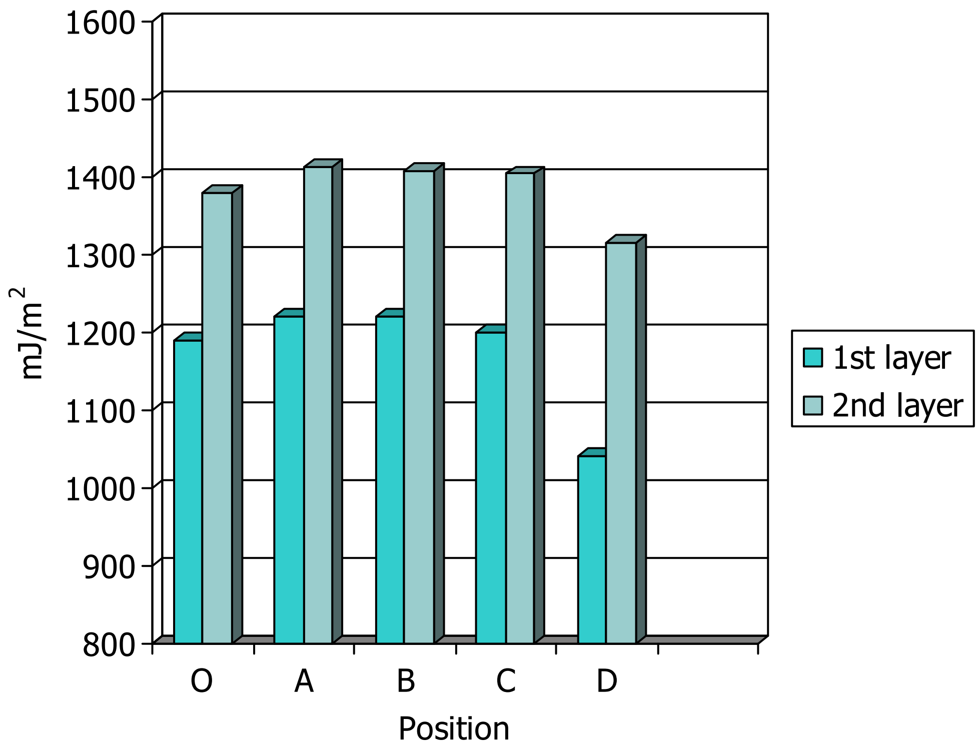
[Figure 24] **Face Centered Cubic Configuration**

Recognizing the symmetrical nature of the Ni surface (four identical quadrants), it is reasonable to reduce the electronic energy calculations to a single quadrant. Each quadrant has a size of 1.762 Å x 1.762 Å and using the pathagoreom theorem, the greatest distance from the center point to the corner of the quadrant to be 2.492 Å. If we move our complex in 0.25 Å increments as illustrated in Figure 25, the complex will travel 0.707 Å linear distance toward the corner of the face, which is approximately 30% of the total diagonal length.


SiO₂ crystal positions from origin 'O' in 0.25 Å increments into the 3rd quadrant on face center cubic surface of Ni
Move #1
O
A
Move #2
Move #3
B
C
Move #4
D
Y
Axis
X

[Figure 25] **SiO_2 Crystal Positions**

Figure 26 plots the interfacial adhesion energy (PES) and shows a relatively stable energy profile for most positions moving away from the idea anchor position (origin). The last position (farthest away) shows a drop off in interfacial adhesion energy as expected, if the relaxed orientation is at a minimum well.



[Figure 26] **Interfacial Adhesion Energy Across Ni Face** (data possessed variance of +/- 15 mJ/m^2 and a STDEV of 5.5)

The 2nd layer of SiO_2 was included to ensure the electronic energy trends are consistent in multi-layer complex. The interfacial adhesion energies between the 1st and 2nd layers are consistent, supporting a stable complex.

7.4 Summary

This chapter investigated the influence of thermal expansion, phase change inversion from α to β, and lattice mis-match between SiO_2 and Ni in the thermal barrier coating complex. The PES of the cohesive, surface free, and interfacial adhesion energy of α SiO_2 on Ni clearly showed the surface free and interfacial adhesion energy of the complex are relatively unchanged up to, and after, the α – β SiO_2 transition point.

The change in the cohesive energy with respect to the Si – O bond length change was found to be substantial in the α SiO_2, driving the system to rearrange itself into the β SiO_2 orientation.

Finally, the lattice mis-match investigation revealed the SiO_2 crystal positions on Ni (100) surface are relatively stable out to 0.707 Ǻ from the origin. Suggesting the Ni/SiO_2 thermal barrier system is tolerant of some degree of "non-perfect" crystal positioning on the Ni substrate.

7.5 References

(1) Carter, E., Jarvis, E., The *role of reactive elements in thermal barrier coatings*, HPC and Nat. Sec., Mar/Apr, pp. 33 – 41 **(2002)**

(2) Tucker, M., Keen, D., Dove, M., *A detailed structural characterization of quartz on heating through the α – β phase transition*, Mineralogical Magazine, 65, pp. 489 – 507 Aug, **(2001)**

(3) Swamy, V., Saxena, S., *A thermodynamic assessment of silica phase diagram*, J. Geophysical Res. 99, pp. 11,787 – 11,794, June **(1994)**

(4) Welche, P., Heine, V., Dove, M., *Negative Thermal Expansion in beta-quartz*, Phys. Chem. Minerals, 26, pp. 63 – 77 **(1998)**

(5) Muser, M., Binder, K., *Molecular dynamics study of the α – β transition in quartz: elastic, finite size effects, and hysteresis in the local structure*, Phys. Chem, Minerals, 28, pp. 746 – 755 **(2001)**

Chapter 8

Conclusion

The primary objectives were outlind in the beginning of this thesis, comprising of a theoretical protocol and an alternative TGO material. The first principles density functional theory protocol was identified to assess the electronic properties of a thermal barrier coating system. The DFT platform of DACAPO was used to successfully approximate the interfacial adhesion energy between the nickel and thermally grown oxide layers, where spallation occurs. Secondly, an alternative oxide layer (SiO_2 as opposed to Al_2O_3) was identified to reduce the tendency of spallation, as the TGO grows over time. Once it was established that SiO_2 should be more stable than Al_2O_3, in a multi-layering scenario, the phase change which occurs with SiO_2 ($\alpha - \beta$ @ 847 K) had to be investigated to ensure that it was not detrimental to the interfacial adhesion energy. Finally, the interface of the 2 layers (Ni/SiO_2) will naturally possess some degree of lattice mis-match, driving the investigation into understanding the lattice mis-match relationship to the interfacial adhesion energy.

8.1 Protocol

To accurately approximate the electronic energies, a density functional theory platform was developed in DACAPO. This consisted of identifying the optimum bravais lattice, planewave cutoff, k-points, pseudopotential wave function, and exchange correlation functional. The final protocol was capable of handling single and multi-layer complexes of SiO_2 on Ni (100) surface and outlined in Table 10, Section 6.2. This approach should allow any researcher to quickly impose changes to the complex and understand the influence on the electronic properties. Also, one can insert alternative materials in the system to understand if they should be considered or avoided in subsequent research, potentially expediting the development of an improved system for industry by quickly providing atomic level insight into the target complex.

8.2 Ni/SiO_2 Viability

It was demonstrated that β SiO_2 remains electronically stable from the mono-layer to a multi-layer complex. The approximations showed a minimal change of 250 mJ/m^2 in interfacial adhesion energy from the 1st to the 4th layer. Also, the interfacial adhesion energy levels off after the 2nd layer, relatively unchanged from there. This suggested that the growth of SiO_2 (TGO), during the combustion cycle of the turbine engine, will not induce spallation as opposed to the current Al_2O_3 TGO system.

To understand the impact of the phase change for SiO_2 at 847 K, the potential energy surface of the Si – O bond length change was plotte within the operating heat cycle. The PES of interfacial adhesion energy showed the highest value at the control bond length and slight decrease, < 100 mJ/m^2, at the inversion point to the beta phase. It was very interesting (and expected) to witness the significant reduction in cohesive energy of α SiO_2 at the Si – O bond length changed at elevated temperatures. At a bond length of 1.585 Å, the cohesive energy was reduced from 26.9 eV to 3.0 eV. At this point, SiO_2 inverts to the β phase configuration with the identical Si – O bond length (1.582 Å) and the cohesive energy jumps to 26.5 eV, creating a cohesively stable network. The beta phase possessed greater thermal stability with minimal Si – O bond length change from 847 to 2000 K. This strongly supported the use of SiO_2 as the TGO in the thermal barrier coating. One last element investigated was the realistic presence of lattice mis-match between the Ni/TGO materials. To understand the influences of lattice mis-match on interfacial adhesion energy, the anchor point of SiO_2 was intentionally shifted along the *X* and *Y* axis to create a series of mis-matches over the Ni (100) surface. As a result, the interfacial adhesion energy remained unchanged until the farthest position, 0.707 Å from origin, further supporting the use of SiO_2 as the TGO.

Considering all the constraints and stressors we imposed on the Ni/SiO_2 complex and the relatively stable interfacial adhesion energies, the study can confidently state that SiO_2 should be considered a viable replacement candidate for Al_2O_3, from a first principles electronic energy approach.

8.3 Future Considerations

This thesis mimicked the behavior of α SiO_2 from ambient conditions up to 847 K and β SiO_2 from 847 to 2000 K. Unfortunately, the actual solid – solid phase transformation was not reproduced, due to the documented difficulty in simulating this solid state instantaneous inversion.[1] Just recently, some literature surfaced on a potential first principles protocol to analyze the transition states and minimum energy paths for a solid – solid phase transformation.[1] As knowledge in this area increases one might consider including this type of investigation for an even more thorough understanding.

8.4 References

(1) Caspersen, K., Carter, E., *Finding transition states for crystalline solid-solid phase transformations*, PNAS, May, 102, pp. 6738 - 6743. **(2005)**

Appendix I

DACAPO Tutorial

Installation

The following installation instructions are centered on the proven version of Dacapo 2.7 on a Redhat Linux system. There is a new version of software (upgrade) being developed by Campos called ASE2, these instructions are geared to the proven version of Dacapo 2.7.

1) Create directory: /dacapo_rpm
2) Go to web site ftp://ftp.fysik.dtu.dk/pub/Campos/bin/Linux/
3) Download the following rpm's into /dacapo_rpm
    a. Campos-1.1-1.i386
    b. Dacapo_2.7.3.run
    c. Distutils-1.0.1-3.noarch.rpm
    d. f2c-19991109-2.i386.rpm
    e. fftw-2.1.3-1.i386.rpm
    f. netcdf-3.5b3-.i386.rpm
    g. numeric-17.3.0.tar.gz
    h. python-netcdf-1.03-1.i386.rpm
    i. python-numpy-1.11-2.i386.rpm
    j. ScientificPython-2.0-1.i386.rpm
    k. wxGTK-2.2.2-1mdk.i586.rpm
    l. wxGTK-gl-2.2.2-1mdk.i586.rpm
    m. wxPython-2.2.2-1-py15.i386.rpm
    n. Rasmol modules
4) Open rpm's c,d,e,f,I,k,l,m (rpm –Uvh filename)
5) Force open h (rpm –Uvh --nodeps python-netcdf-1.03-1.i386.rpm)
6) Open j
7) Open a
8) Expand Tar file g (tar –xzvf numeric-17.3.0.tar.gz)
9) Change mode for b (chmod 777 dacapo-2.7.3.run)
10) Change mode for n as done in step 9
11) Break Shell
12) Open New Shell
13) Ready to go.

CO Molecule

One of the simplest calculations is the total energy of a CO molecule. The minimal specifications that must be provided are a unit cell, geometry, planewave energy cutoff, and the number of electronic bands. Below is and example script of such a calculation followed by an explanation for each step[5].

```
51        #!/usr/bin/env python
52
53        ###################################
54        #                                 #
55        #      CO dimer in a box          #
56        #                                 #
57        ####################################
58
59        from Simulations.Dacapo import *
60
61        sim=Simulation()
62
63        sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
64                                [0.0,   1.0,   0.0],
65                                [0.0,   0.0,    1.0]])
66
67        sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
68                               Atom(type=O, position=Vector([5.1, 4,    4]))],
69                               unitcell=8.0*sc_latt
70
71        sim.plancut = PlaneWaveCutoff(300)  #  in eV
72        sim.eband = ElectronicBands(10)
73
74        sim.Execute(outfile=”CO_in_a_box.nc”, ascii=CO_in_a_box.txt”)
```

Line 1 enables the user to run python as a shell language anywhere. Other than the first line, the “#” means that the rest of the line is a comment enabling the user to place comments after statements, as shown in line 21. Line 9 is where you make function calls to Simulations.Dacapo possible. Line 11 allows you to create a python object sim of data type Simulation. Line 13 is similar and the data type

BravaisLattice represents a Bravais lattice. The initial value given to the Bravais Lattice is the unit matrix [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]. The default units are Å for length and fs for time. Line 17 sets up the atoms and their respective coordinates. Line 19 sets up the periodic boundary conditions. In the example above we apply a factor of 8 to the simple cubic. This can also be done by setting the Bravais lattice dimensions (line 13) to [[8.0, 0.0, 0.0], [0.0, 8.0, 0.0], [0.0, 0.0, 8.0]]. Line 21 sets the plane wave basis kinetic energy cut off at 300 eV and line 22 sets the number of electronic valance bands in the calculation. Finally, Line 24 is a function call, acting on the variable 'sim', that first causes all the attached variables to sim to be written to an input file. The output file names called out in line 24 are of 2 types. The first is a binary netCDF formatted file (.nc) containing all important physical data (energies, forces, charge density, magnetic moments, etc.) and the second output file is an ASCII formatted readable log file (.txt) describing the course of the just performed calculation[2].

Geometric Optimization

Typically, an investigator is interested in the total energy of the optimized geometry of a system. This can be done in Dacapo by adding a single line of code, as illustrated below in line 48.

```
75      #!/usr/bin/env python
76
77      ####################################
78      #                                  #
79      #       CO dimer in a box          #
80      #                                  #
81      ####################################
82
83      from Simulations.Dacapo import *
84
85      sim=Simulation()
86
87      sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
88                              [0.0,   1.0,   0.0],
89                              [0.0,   0.0,   1.0]])
90
91      sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
92                             Atom(type=O, position=Vector([5.1, 4,    4]))],
93                             unitcell=8.0*sc_latt
94
95      sim.plancut  = PlaneWaveCutoff(300)  #  in eV
96      sim.eband    = ElectronicBands(10)
97
98      sim.dyn      = Dynamics()
99
100     sim.Execute(outfile=”CO_relaxed.nc”)
```

Atom Projected DOS

To produce a projection of Density of States (DOS), the following script is required.

```
101     #!/usr/bin/env python
102
103     ##################################
104     #                                #
105     #       CO dimer in a box        #
106     #                                #
107     ###################################
108
109     from Simulations.Dacapo import *
110     from Simulations.Dacapo.AtomProjectedDOS    import AtomProjectedDOS
111
112     sim=Simulation()
113
114     sim.atoms    =  ListOfAtoms()
115     sim.UpdateFromNetCDFFile("CO_relaxed.nc")
116
117     sim.plancut = PlaneWaveCutoff(300)  #  in eV
118     sim.eband = ElectronicBands(10)
119
120     sim.ados    = AtomProjectedDOS()
121
122     sim.Execute(outfile="CO_relaxed_ados.nc")
```

Line 60 loads the necessary DOS module in the working directory. Line 65 reloads the latest binary .nc file, which contains all atomic information of the complex. Notice that it is the output of the previous calculation. This is done intentionally to show how easily you can link one investigation with another. Line 70 is the key line in this, tagging the DOS to a specific name.

Visualizing Geometry, DOS, Potential Energy

```
123     #!/usr/bin/env python
124
125     from Simulations.Dacapo import *
126     from Simulations.Dacapo.EigenState              import EigenState
127     from Simulations.Dacapo.ListOfEigenStates       import ListOfEigenStates
128
129     sim                     = Simulation()
130     sim.atoms               = ListOfAtoms()
131     sim.all                 = ListOfEigenStates()
132
133     sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
134                             [0.0,   1.0,   0.0],
135                             [0.0,   0.0,    1.0]])
136
137     sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
138                               Atom(type=O, position=Vector([5.1, 4,    4]))],
139                               unitcell=8.0*sc_latt
140
141     sim.plancut  = PlaneWaveCutoff(300)  #  in eV
142     sim.eband    = ElectronicBands(10)
143
144     sim.dyn        = Dynamics()
145
146     sim.Execute(outfile=”CO_relaxed.nc”, ascii=”CO_relaxed.txt”)
147
148     sim.UpdateFromNetCDFFile(“CO_relaxed.nc”)
149
150     dosplot         = sim.all.GetDOS(smoothfactor=0.2).GetPlot()
151     raw_input (“press Enter to continue”)
152     dosplot.SaveAsPS(“CO_relaxed_plot.ps”)
153
154     p=sim.atoms.GetPlot()
155     raw_input (“press Enter to continue”)
156
157     print ‘The Energy is %2.4f eV’% sim.atoms.GetTotalPotentialEnergy()
```

Lines 76, 77 load the EigenState Modules into the working directory. Line 98 reloads the latest binary .nc file into the working directory to extract all necessary data for visualization. Lines 100 – 107 are the visualization commands. Imbedded in these lines are raw input commands to allow the researcher the ability to review the plot/display at their discretion. The last line (107) prints the Total Potential Energy on the screen, within the shell. In the upcoming examples, we will demonstrate how Magnetic Moment, Bond Length, and Atomic Forces are displayed.

Combining all in 1 script

Applying all the 4 elements described above (Total Energy, Geometry Optimization, Atom Projected DOS, and Visualization) into 1 script can be done as follows.

```
#!/usr/bin/env python

##################################
#                                #
#       CO dimer in a box        #
#                                #
###################################

from Simulations.Dacapo import *
from Simulations.Dacapo.AtomProjectedDOS     import AtomProjectedDOS
from Simulations.Dacapo.EigenState           import EigenState
from Simulations.Dacapo.ListOfEigenStates    import ListOfEigenStates

sim                  = Simulation()
sim.all              = ListOfEigenStates()

sc_latt=BravaisLattice([[1.0,   0.0,   0.0],
                        [0.0,   1.0,   0.0],
                        [0.0,   0.0,   1.0]])

sim.atoms=ListOfAtoms([Atom(type=C, position=Vector([4,    4,    4])),
                       Atom(type=O, position=Vector([5.1, 4,    4]))],
                       unitcell=8.0*sc_latt

sim.plancut = PlaneWaveCutoff(300)  #  in eV
sim.eband   = ElectronicBands(10)

sim.ados     = AtomProjectedDOS()
sim.dyn      =  Dynamics()

sim.Execute(outfile="CO_relaxed.nc", ascii=CO_in_a_box.txt")

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

```

```
192     sim.UpdateFromNetCDFFile("CO_relaxed.nc")
193
194     dosplot         = sim.all.GetDOS(smoothfactor=0.2).GetPlot()
195     raw_input ("press Enter to continue")
196     dosplot.SaveAsPS("CO_relaxed_plot.ps")
197
198     p=sim.atoms.GetPlot()
199     raw_input ("press Enter to continue")
200
201     print sim.atoms.GetTypes()
202
203     postitions = sim.atoms.GetCartesianPositions()
204     print positions
205
206     C_pos=positions[0]
207     O_pos=positions[1]
208
209     print 'Bondlength = ',(Vector(C_pos)-Vector(O_pos)).Length()
210
211     print 'Forces: '
212     print sim.atoms.GetCartesianForces()
213
214     magneticmoment = sim.all.GetMagneticMoment()
215     print 'Magnetic Moment = ', magneticmoment
216
217     print 'The Energy is %2.4f eV'% sim.atoms.GetTotalPotentialEnergy()
```

Line 151 prints the atoms being studied. Line 153 and 154 display the positions of each atom while line 156 and 157 label each atom of interest. The Bond Length is then calculated in Line 159, which calls on the specific atoms of interest. The Atomic Forces are printed in line 162 and the Magenetic Moment is shown in line 165.

Oxygen Example

Below is a script designed to calculate the magnetic moment of Oxygen and print out the resulting approximation. Included in the script is an alternative way to represent the Bravais Lattice.

```
#!/usr/bin/env python

from Simulations.Dacapo import *
from Simulations.Dacapo.AtomProjectedDOS    import AtomProjectedDOS
from Simulations.Dacapo.EigenState          import EigenState
from Simulations.Dacapo.ListOfEigenStates   import ListOfEigenStates

sim                 =Simulation()
sim.all             =ListOfEigenStates()

#  Lattice Vector Definitions
a1=[1.0,0.0,0.0]
a2=[0.0,1.0,0.0]
a3=[0.0,0.0,1.0]

ucell=10*BravaisLattice([a1,a2,a3])

sim.loa=ListOfAtoms([Atom(type=O, position=Vector([4,    4,    4])),
                          magneticmoment = 2.0)],
                          unitcell=ucell)

sim.ExcFunc=ExcFunctional('RPBE')

sim.bands=ElectronicBands(10)
sim.bands.OccupationStatistics_FermiTemperature=0.001
sim.bands.SpinPolarization=2

sim.plancut = PlaneWaveCutoff(300)  #  in eV

sim.Execute('O.nc', 'O.txt')

print 'Forces: '
print sim.atoms.GetCartesianForces()

magneticmoment = sim.all.GetMagneticMoment()
print 'Magnetic Moment = ', magneticmoment

print 'The Energy is %2.4f eV'% sim.atoms.GetTotalPotentialEnergy()
```

Line 186 is an initial guess at the magnetic moment and will serve as a starting point for all iterations. This value directly impacts the Exchange/Correlation Functional in Line 189. Fermi Temperature and Spin Polarization are set in Lines 192, 193 respectfully. The remaining lines are carried over from previous examples.

An alternative way to retrieve the magnetic moment from the .txt file can be done by utilizing the “grep” command as follows:

```
grep –A 2 MOM O.txt
```

You will see many lines that look like.

```
MOM    1.999999991           (4.000000000003  2.0000000000012)
```

The first number (1.99999991 is the net magnetic moment of the atom. The numbers in parentheses refer to the number of spin up and spin down electrons.

Oxygen Molecule

let's look at the O_2 molecule and combine the elements in the O atom study in the previous section. Also, we will allow the molecule to relax (dynamics) to an optimized configuration and then determine the magnetic moments and energies.

```
256     #!/usr/bin/env python
257
258     from Simulations.Dacapo import *
259     from Numeric import *
260
261     sim=Simulation()
262
263     #  Lattice Vector Definitions
264     a1=[1.0,0.0,0.0]
265     a2=[0.0,1.0,0.0]
266     a3=[0.0,0.0,1.0]
267
268     ucell=10*BravaisLattice([a1,a2,a3])
269
270     sim.loa=ListOfAtoms([
271         Atom(type=O, position=Vector([4.4, 5.0, 5.0]),magneticmoment=1.0),
272         Atom(type=O, position=Vector([5.61, 5.0, 5.0]),magneticmoment=1.0)],
273         unitcell=ucell)
274
275     NumberOfElectrons=(10)
276
277     sim.bands=ElectronicBands(NumberOfElectrons/2+5)
278     sim.bands.OccupationStatistics_FermiTemperature = 0.001
279     sim.bands.SpinPolarization = 2
280
281     sim.ExcFunc=NetCDF.Entry('ExcFunctional' , 'RPBE')
282
283     sym.dynamics=Dynamics()
284
285     sim.plancut = PlaneWaveCutoff(400)
286
287     sim.Execute('O2_relaxed.nc', 'O2_relaxed.txt')
```

Lines 225 and 227 work together to automatically determine the number of electronic bands. This is a useful technique in determining the # of bands. Line

225 determines the total number of valance electrons and Line 227 calls on this number and divides it by 2 and adds 5. Dacapo has established the following working principle:

# of Electronic Bands must be $\geq$ ½ the total # of Valence electrons

You can see in Line 233 we invoke the dynamics step to allow the molecule to relax (converge) to the lowest energy state.

Cadmium Sulfide (1 unit Cell)

In this section we will calculate the bulk energy of a F43m CdS unit cell. We will add new elements to the input file. These include more involved definition of Dynamics (relaxation), k-point sampling, and the importance of the Unit Cell size.

```
288     #!/usr/bin/env python
289
290     from Simulations.Dacapo import *
291     from Numeric import *
292
293     sim=Simulation()
294
295     #  Lattice Vector Definitions
296     a1=[1.0,0.0,0.0]
297     a2=[0.0,1.0,0.0]
298     a3=[0.0,0.0,1.0]
299
300     ucell=8*BravaisLattice([a1,a2,a3])
301
302     sim.loa=ListOfAtoms([
303         Atom(type=Cd, position=Vector([5.832,   0.000,   5.832])),
304         Atom(type=Cd, position=Vector([5.832,   0.000,   0.000])),
305          Atom(type=Cd, position=Vector([0.000,   0.000,    0.000])),
306          Atom(type=Cd, position=Vector([0.000,   0.000,    5.832])),
307          Atom(type=Cd, position=Vector([5.832,   5.832,    5.832])),
308          Atom(type=Cd, position=Vector([0.000,   5.832,    5.832])),
309          Atom(type=Cd, position=Vector([5.832,   5.832,    0.000])),
310          Atom(type=Cd, position=Vector([0.000,   5.832,    0.000])),
311          Atom(type=Cd, position=Vector([2.916,   0.000,    2.916])),
312          Atom(type=Cd, position=Vector([2.916,   2.916,    5.832])),
313          Atom(type=Cd, position=Vector([5.832,   2.916,    5.832])),
314          Atom(type=Cd, position=Vector([2.916,   2.916,    0.000])),
315          Atom(type=Cd, position=Vector([0.000,   2.916,    2.916])),
316          Atom(type=Cd, position=Vector([2.916,   5.832,    2.916])),
317          Atom(type=S, position=Vector([4.374,   1.458,    4.374])),
318          Atom(type=S, position=Vector([1.458,   1.458,    1.458])),
319          Atom(type=S, position=Vector([1.458,   4.374,    4.374])),
320          Atom(type=S, position=Vector([4.374,   4.374,    4.374]))],
321          unitcell=ucell)
322
323      NumberOfElectrons=(192)
324
325      sim.plancut = PlaneWaveCutoff(400)
326
327      sim.bands=ElectronicBands(NumberOfElectrons/2+5)
328
329      sim.dynamics= NetCDF.Entry(name="Dynamics", value="");
330      sim.dynamics.Type = “Relaxation”
331      sim.dynamics.Method = “ConjugateGradient”
332      sim.dynamics.Step = 1.2000
333
334      sim.eigensolver = NetCDF.Entry(name="electonicMinimization", value="");
335      sim.eigensolver.Method = “eigsolve”
336      sim.eigensolver.DiagonalizationPerBand = 2
337
338      sim.conv = NetCDF.Entry(name="ConvergenceControl")
339      sim.conv.MaxNumberOfSteps = 100000
340
341      sim.kpoints = NetCDF.Entry(name="kpointSetup", value=[15,15,15])
342
343      sim.Execute(‘CdS_bulk.nc’, ‘CdS_bulk.txt’)
344      sim.UpdateFromNetCDFFile(“CdS_bulk.nc)
```

In the above script, we allow all atoms to relax in Line 279. In subsequent lines we call out the specific Type, Method, and Step (Lines 280 – 282). Our selections call the calculation to use a Complex Conjugate Algorithm with a 1.2 fs time integration for the ionic minimization. In Line 284 and 286 we set up the electronic minimization subroutine (Method) as “eigsolve” which refers to the Block Davidson algorithm. Line 260 outlines 2 diagonalizations per band. This applies to Method “resmin” (Power method, Lennart Bentson, and can only handle k-point parallelization). Line 288 and 289 is a way to control the number on steps. This may be valuable in the screening of a script or computational approach to minimize time. Finally, Line 294 updates the binary .nc file automatically.

It important to note when studying bulk species (and others) the number of planewaves used in the calculation is dependent on the size of the unit cell (ie. Bravais Lattice). If one would like to know how many planewaves are being used. Use the following command line in a shell.

```
grep –A 3 “# of PW” CdS_bulk.txt
ncdump –v NumberPlaneWavesKpoint CdS_bulk.nc
ncsum –d CdS_bulk.nc
```

If your planewave cutoff is high enough, the influence of the Unit Cell dimensions is minimized.

k-Point Determination

There is no automated way to determine the optimum # of k-points for a given calculation. Most individuals begin with an estimated amount of k-points, usually low, and run quick energy calculations. They change the # of k-points in a controlled fashion, usually increasing until the energy begins to converge. At that point, the optimum k-points can be identified. You can also measure the lattice constant of a relaxed complex if desired. Both techniques work.

Bulk Energy with Constraints on designated Atoms

Below is the same script used in section 5.1 but we apply motion constraints on designated atoms and all others are allowed to relax.

```
#!/usr/bin/env python

from Simulations.Dacapo import *
from Numeric import *

sim=Simulation()

#  Lattice Vector Definitions
a1=[1.0,0.0,0.0]
a2=[0.0,1.0,0.0]
a3=[0.0,0.0,1.0]

ucell=8*BravaisLattice([a1,a2,a3])

sim.loa=ListOfAtoms([
    Atom(type=Cd, position=Vector([5.832, 0.000, 5.832]), constraints='123'),
    Atom(type=Cd, position=Vector([5.832, 0.000, 0.000]), constraints='123'),
    Atom(type=Cd, position=Vector([0.000,    0.000,     0.000])),
    Atom(type=Cd, position=Vector([0.000,    0.000,     5.832])),
    Atom(type=Cd, position=Vector([5.832,    5.832,     5.832])),
    Atom(type=Cd, position=Vector([0.000,    5.832,     5.832])),
    Atom(type=Cd, position=Vector([5.832,    5.832,     0.000])),
    Atom(type=Cd, position=Vector([0.000,    5.832,     0.000])),
    Atom(type=Cd, position=Vector([2.916,    0.000,     2.916])),
    Atom(type=Cd, position=Vector([2.916,    2.916,     5.832])),
    Atom(type=Cd, position=Vector([5.832,    2.916,     5.832])),
    Atom(type=Cd, position=Vector([2.916,    2.916,     0.000])),
    Atom(type=Cd, position=Vector([0.000,    2.916,     2.916])),
    Atom(type=Cd, position=Vector([2.916,    5.832,     2.916])),
    Atom(type=S, position=Vector([4.374, 1.458, 4.374]), constraints='123'),
    Atom(type=S, position=Vector([1.458, 1.458, 1.458]), constraints='123'),
    Atom(type=S, position=Vector([1.458,    4.374,     4.374])),
    Atom(type=S, position=Vector([4.374,    4.374,     4.374]))],
    unitcell=ucell)
```

```
379
380     NumberOfElectrons=(180)
381
382     sim.plancut = PlaneWaveCutoff(400)
383
384     sim.bands=ElectronicBands(NumberOfElectrons/2+5)
385
386     sim.dynamics= NetCDF.Entry(name=”Dynamics”, value=””);
387     sim.dynamics.Type = “Relaxation”
388     sim.dynamics.Method = “ConjugateGradient”
389     sim.dynamics.Step = 1.2000
390
391     sim.eigensolver = NetCDF.Entry(name=”electonicMinimization”, value=””);
392     sim.eigensolver.Method = “eigsolve”
393     sim.eigensolver.DiagonalizationPerBand = 2
394
395     sim.conv = NetCDF.Entry(name=”ConvergenceControl”)
396     sim.conv.MaxNumberOfSteps = 100000
397
398     sim.kpoints = NetCDF.Entry(name=”kpointSetup”, value=[15,15,15])
399
400     sim.Execute(‘CdS_bulk_constraints.nc’, ‘CdS_bulk_constraints.txt’)
401     sim.UpdateFromNetCDFFile(“CdS_bulk_constraints.nc)
```

Constraints are applied to 2 Cd and 2 S atoms (Lines 310, 311 and 324, 325) in the complex. This freezes the geometric position within the unit cell and allows all other atoms to move to a low energy position. Some times it is desired to freeze a slab atoms position and allow a lone atomic species to find the lowest energy state on said slab surface. This minimizes the computational demand, speeding up the calculation, and allows insight into the mechanics/electrical properties of the adsorbate on a frozen slab. We will discuss this in more detail in the next section. As illustrated in the previous examples, you can add the commands to display any information you want after line 351. We did not include it here to minimize space.

Complex/Adsorbate on the surface of a CdS slab

The script below utilizes an existing bulk Cadmium Sulfide, F43m space group (CdS_bulk.nc) file for the slab “list of atoms” position and illustrates how an adsorbate species can be added to the surface for electronic approximations. The adsorbate atoms will be allowed to relax through the dynamic parameters outlined in the input file.

If the .nc file does not exist, it must be created and placed in the working directory to execute the calculation. Generating the lowest energy state (relaxed configuration) binary ‘.nc’ file was demonstrated many times throughout this tutorial.

```
#!/usr/bin/env python

from Simulations.Dacapo import *
from Numeric import *
from RandomArray import random

cds_sim=Simulation()
cds_sim.loa=ListOfAtoms()
cds_sim.UpdateFromNetCDFFile(‘CdS_bulk.nc’)

for atom in CdS_bulk.nc:              #  only required if constraints not present
        atom.SetConstraints(‘123’)  #  only required if constraints not present

cds_sim.loa.append(Atom(C,Vector([10,10,10])))
cds_sim.loa.append(Atom(H,Vector([10,11,10])))
cds_sim.loa.append(Atom(H,Vector([10,11,11])))
cds_sim.loa.append(Atom(H,Vector([10,12,11])))

p=cds_sim.loa.GetPlot()
raw_input (“press Enter to continue”)

NumberOfElectrons=(180)

sim.bands=ElectronicBands(NumberOfElectrons/2+5)

sim.kpoints = NetCDF.Entry(name=”kpointSetup”, value=[15,15,15])
```

```
428
429     sim.dip=NetCDF.Entry('DipoleCorrection')
430
431     sim.ExcFunc=NetCDF.Entry('ExcFunctional','RPBE')
432
433     sim.symmetry=NetCDF.Entry(name="UseSymmetry",value="Maximum")
434
435     sim.dynamics=NetCDF.Entry("Dynamics")
436
437     sim.Execute('CdS+ch3.nc', 'CdS+ch3.txt')
438     sim.UpdateFromNetCDFFile('CdS+ch3.nc')
```

In the beginning we load the necessary modules in Dacapo as done before. Lines 358, 359, and 360 load the required binary (.nc) file for the slab complex. If the file resides in an alternative directory/location, then a route command must be used to direct the program to the right location to retrieve the file (or just make sure the file is in the working directory). Lines 362, 363 are present to place constraints on the slab atoms. This allows the adsorbate complex to relax to the lowest energy configuration in relation to the slab and will help expedite the calculation. If your slab already possesses constraints in the ListOfAtoms (loa), then these commands are not required. Lines 365 – 368 add the adsorbate atoms (species) on the slab through the "append" command. We added visualization step in 370 & 371 to allow the researcher to verify the location of the newly added atoms (species). This is critical in building an appropriate configuration. A raw input command is installed after the species is viewed to allow the researcher the ability to abort the calculation and modify the configuration. This new complex is used in the remaining electronic calculations as done in previous scripts. Notice that we have implemented Dynamics (Relaxation) on the complex and since we have introduced constraints on the slab atoms, only the adsorbate species will be manipulated to the lowest energy state in relation to a frozen slab.

ASCII Output File

The ASCII output can either be read through the ascii file with an editor, or you can use the grep command (grep keyword ASCIIoutfile) to search for lines containing a key word. Below is a list of keywords in the ASCII output file from Dacapo.

<u>TOT and DFT</u>

The total energy for the different exchange-correlation functional is given for each iteration. The headline tells you if the energy is self-consistent or calculated non-self-consistent for the given density.

<u>STEP</u>

For ionic relaxation, monitors ionic step length, residual forces on non-constrained degreees of freedom and the total energy.

<u>MOM</u>

For spin polarization calculations, monitors the evolution of the magnetic moment for each electronic iteration.

<u>STRUCTURE</u>

This gives the unit cell and the atomic coordinates for the calculation. Also the constraints on the atoms is given here.

SYM

Gives information about the point group operations the symmetry module has found.

KPT

The KPT keyword gives information about the k-point given as input, either by specifying a predefined k-points set (Monkhorst-Pack/Chadi-Cohen) in the variable KpointSetup or directly in the variable BZKpoints. The symmetry reduced set of k-points in the irreducible Brillouin zone is also listed. There are many other key words and one can search the CAMPOS web site for further details[6]. At the end of the ASCII file, there is a breakdown on the computational time required to complete a successful run.

References

1 Leach A., 2001. Molecular Modeling Principles and Applications Prentice Hall, Ch. 2; pgs 26 – 41.

2 Ratner M., Schatz G., Introduction to Quantum Mechanics in Chemistry Prentice Hall, pgs 148 – 156.

3 Hoffmann R. 1988. Solids and Surfaces, A Chemist's View of Bonding in Extended Structures WILEY-VCH, pgs.3 –32.

4 Burdett J., 1995. Chemical Bonding in Solids Oxford University Press, Ch 1 – 3.

5 Christensen A. et al. CAMPOS/Dacapo, Technical University of Sweden, (2002)

6 Kitchen, J., Dacapo Tutorial April 2004

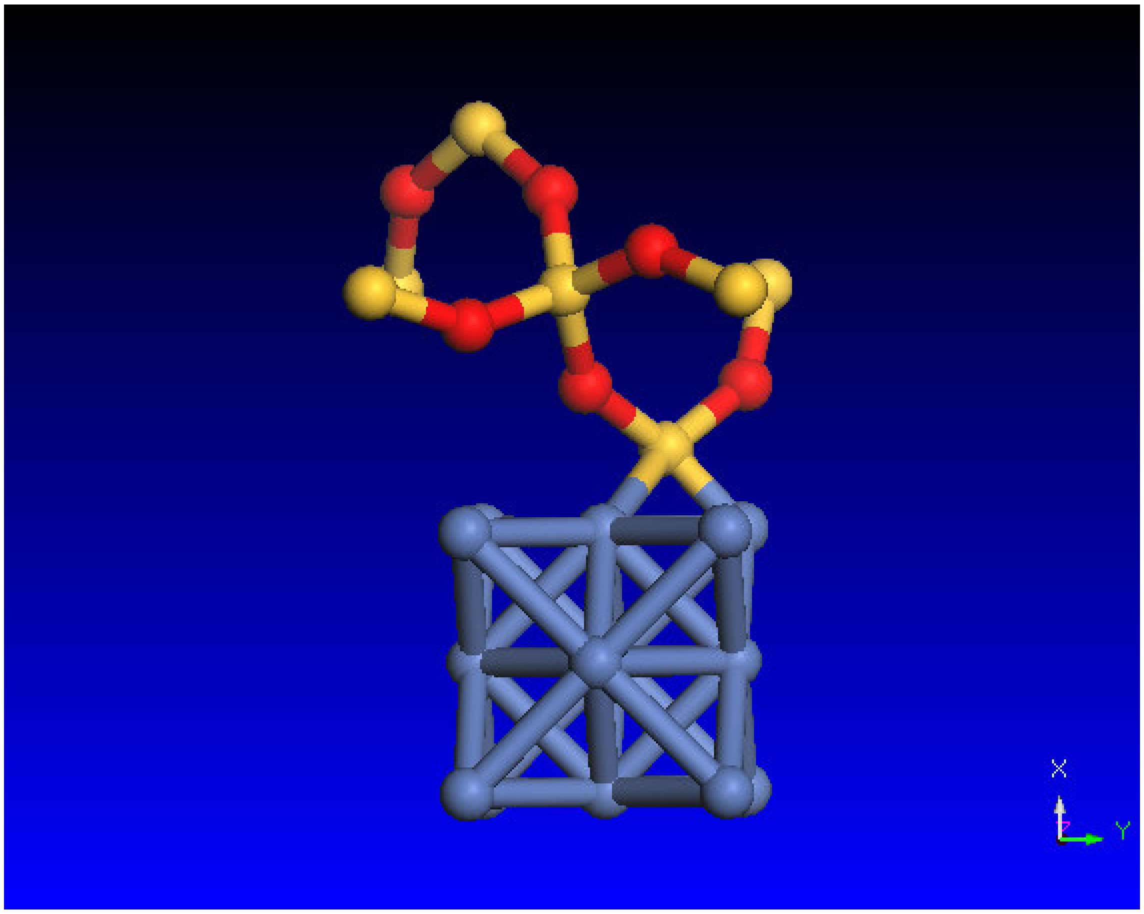
Appendix II

Input Files and Unit Cell Configurations

This Appendix contains the DACAPO input files used to generate the bulk free energy of the Ni/SiO_2 complexes, which involve one and two layers α, β SiO_2 on Ni complex. These bulk energies were used to construct the cohesive, surface free, and interfacial adhesion energies. The input files were further utilized to produce the Potential Energy Surface by reducing the Si – O bond length as described earlier. Ni surface and complex remained unchanged in all studies in this thesis. After each input file, the geometric configuration is illustrated for reference.

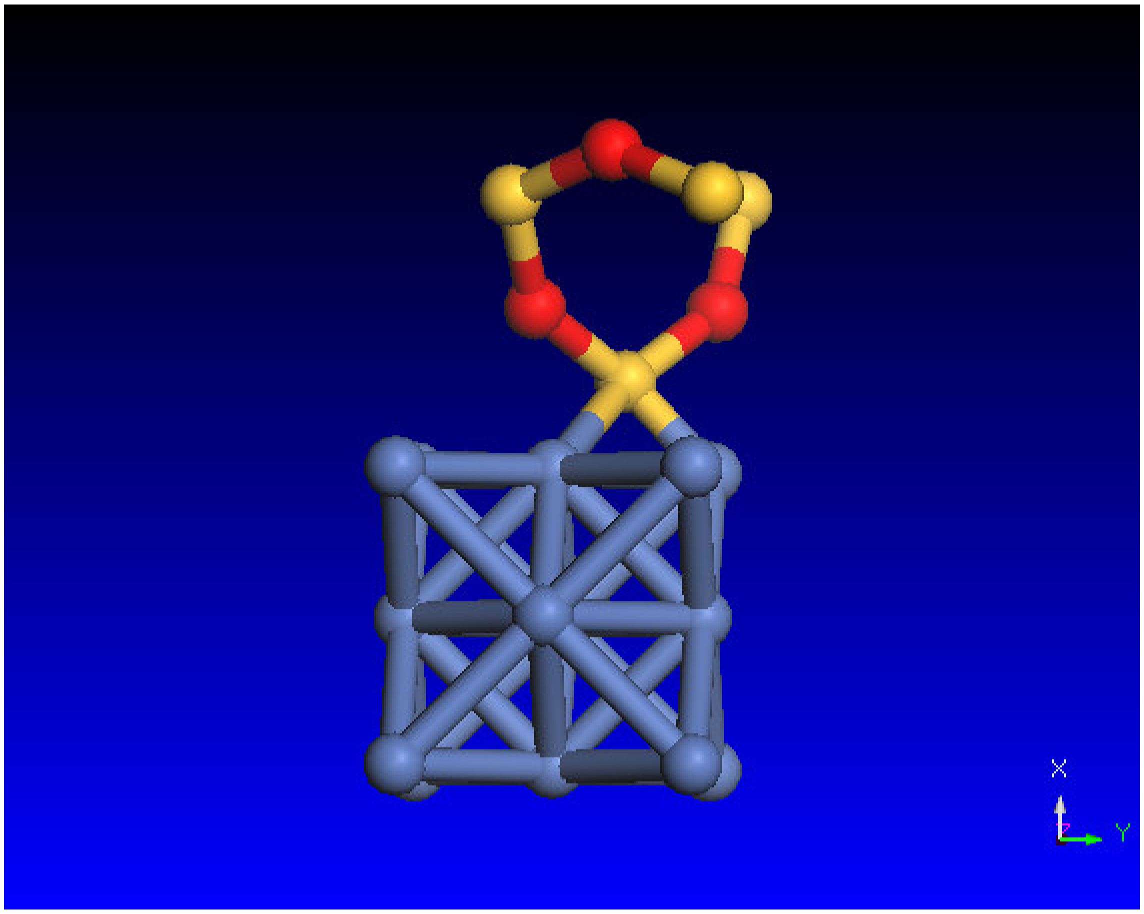
Ni/β SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python

###########################################################
#                                                         #
#    NiSiO2_beta3_1layer_control  input file for 2 layer of SiO2 on Ni   #
#                                                         #
###########################################################



from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

#########################################################
#                                                       #
# The Ni - O bond length is 1.93 A                      #
# Ni Lattice constant is 3.524  A                       #
# Therefore, add 4.614 A is lattice offset for Si and O atoms   #
# This offset is added to the X axis for all Si and O atoms     #
# Refer to Control input file for original X positions  #
# Orig file name:  SiO2_beta3_1layer_relax2             #
#                                                       #
#########################################################


sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.757,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.505,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.757,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 1.252,
1.823], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([8.953, 0.000,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.253,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.253,
0.000], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.530,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 1.480,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, -1.025,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, 1.025,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([6.323, 0.000,
0.912], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.505,
0.912], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000
sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_beta3_1layer_controla.nc",
ascii="NiSiO2_beta3_1layer_controla.txt")

sim.UpdateFromNetCDFFile("NiSiO2_beta3_1layer_control.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```



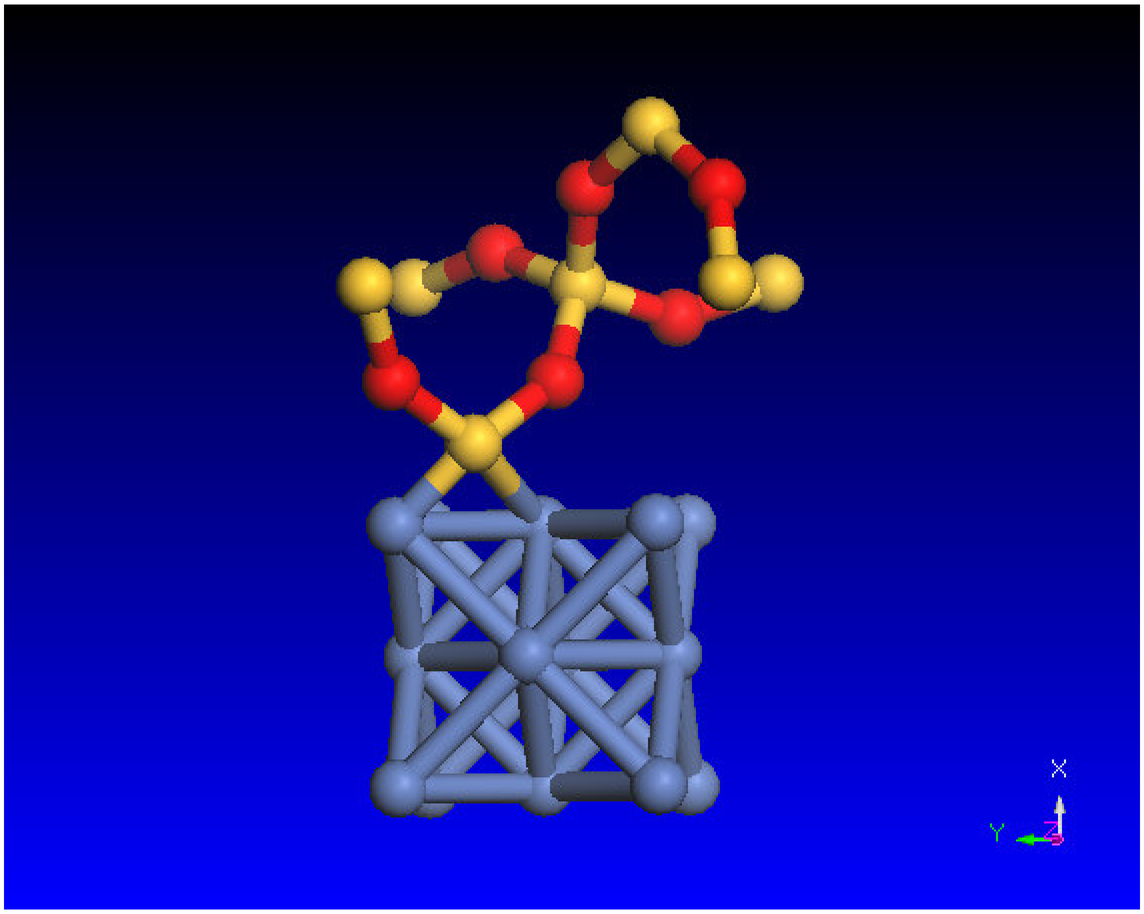
Ni/β SiO_2 Complex input file
1 layer of SiO_2

```
#!/usr/bin/env python

#####################################################
#                                                   #
# NiSiO2_beta-halflayer  input file for 1 layer of SiO2 on Ni     #
#                                                   #
#####################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##############################################################
#                                                            #
# The Ni - O bond length is 1.93 A                           #
# Ni Lattice constant is 3.524  A                            #
# Therefore, add 4.614 A is lattice offset for Si and O atoms  #
# This offset is added to the X axis for all Si and O atoms   #
# Refer to Control input file for original X positions        #
# Orig file name:  SiO2_beta3_1layer_relax2                   #
#                                                            #
##############################################################


sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000, 0.000],
ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000, 3.524],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([0.000, 3.524, 3.524],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([0.000, 3.524, 0.000],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([3.524, 3.524, 0.000],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([3.524, 3.524, 3.524],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([3.524, 0.000, 0.000],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([3.524, 0.000, 3.524],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([1.762, 3.524, 1.762],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([0.000, 1.762, 1.762],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([1.762, 1.762, 3.524],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([3.524, 1.762, 1.762],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([1.762, 0.000, 1.762],
ucell),constraints='123'),
                    Atom(type=Ni, position=Vector([1.762, 1.762, 0.000],
ucell),constraints='123'),
                    Atom(type=Si, position=Vector([6.783, 3.757, 5.470],
ucell),constraints='123'),
                    Atom(type=Si, position=Vector([4.614, 2.505, 3.647],
ucell),constraints='123'),
                    Atom(type=Si, position=Vector([6.783, 3.757, 0.000],
ucell),constraints='123'),
                    Atom(type=Si, position=Vector([6.783, 1.252, 1.823],
ucell),constraints='123'),
#                   Atom(type=Si, position=Vector([8.953, 0.000, 3.647],
ucell),constraints='123'),
#                   Atom(type=Si, position=Vector([6.783, -1.253, 5.470],
ucell),constraints='123'),
#                   Atom(type=Si, position=Vector([6.783, -1.253, 0.000],
ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.530, 4.558],
ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 1.480, 2.735],
ucell),constraints='123'),
#                             Atom(type=O, position=Vector([8.098, -1.025, 4.558],
ucell),constraints='123'),
#                             Atom(type=O, position=Vector([8.098, 1.025, 2.735],
ucell),constraints='123'),
#                             Atom(type=O, position=Vector([6.323, 0.000, 0.912],
ucell),constraints='123')],
                              Atom(type=O, position=Vector([7.306, 2.505, 0.912],
ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(120)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization", value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000
sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_beta_halflayer1.nc",
ascii="NiSiO2_beta_halflayer1.txt")

#sim.UpdateFromNetCDFFile("NiSiO2_beta_halflayer.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```



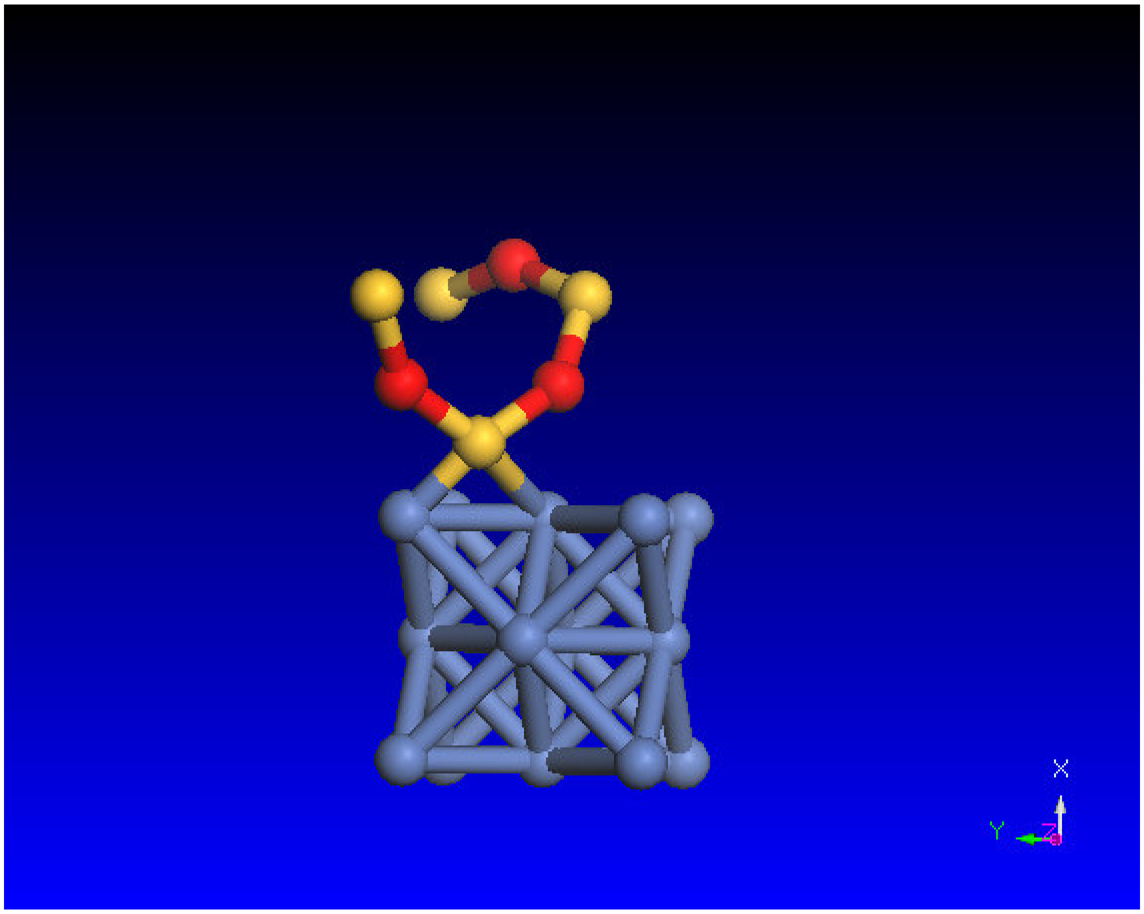
Ni/α SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python

##########################################################
#                                                        #
#   NiSiO2_alpha_1layer_control  input file for 2 layer of SiO2 on Ni  #
#                                                        #
##########################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

########################################################
#                                                      #
# The Ni - O bond length is 1.93 A                     #
# Ni Lattice constant is 3.524  A                      #
# Therefore, add 4.614 A is lattice offset for Si and O atoms  #
# This offset is added to the X axis for all Si and O atoms    #
# Refer to Control input file for original X positions         #
# Orig file name:  SiO2_beta3_1layer_relax2            #
#                                                      #
########################################################


sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                                Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([6.727, -1.220,
0.000], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([6.727, 1.220,
1.803], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([8.840, -0.000,
3.607], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([6.727, -1.220,
5.410], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([6.727, 3.660,
5.410], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([4.614, 2.440,
3.607], ucell),constraints='123'),
                                Atom(type=Si, position=Vector([6.727, 3.660,
0.000], ucell),constraints='123'),
                         Atom(type=O, position=Vector([6.279, -0.000,
0.902], ucell),constraints='123'),
                         Atom(type=O, position=Vector([8.008, 0.998,
2.705], ucell),constraints='123'),
                         Atom(type=O, position=Vector([8.008, -0.998,
4.508], ucell),constraints='123'),
                         Atom(type=O, position=Vector([5.447, 3.438,
4.508], ucell),constraints='123'),
                         Atom(type=O, position=Vector([5.447, 1.442,
2.705], ucell),constraints='123'),
                         Atom(type=O, position=Vector([7.175, 2.440,
0.902], ucell),constraints='123')],
                         unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000
sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_beta3_1layer_controla.nc",
ascii="NiSiO2_beta3_1layer_controla.txt")

sim.UpdateFromNetCDFFile("NiSiO2_beta3_1layer_control.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```



Ni/α SiO_2 Complex input file
1 layer of SiO_2

```
#!/usr/bin/env python

##############################################################
#                                                            #
#    NiSiO2_alpha_halflayer_control  input file for 2 layer of SiO2 on Ni #
#                                                            #
##############################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

############################################################
#                                                          #
# The Ni - O bond length is 1.93 A                         #
# Ni Lattice constant is 3.524  A                          #
# Therefore, add 4.614 A is lattice offset for Si and O atoms #
# This offset is added to the X axis for all Si and O atoms #
# Refer to Control input file for original X positions     #
# Orig file name:  SiO2_beta3_1layer_relax2                #
#                                                          #
############################################################


sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.727, -1.220,
0.000], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([6.727, 1.220,
1.803], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([8.840, -0.000,
3.607], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([6.727, -1.220,
5.410], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.727, 3.660,
5.410], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.440,
3.607], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.727, 3.660,
0.000], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([6.279, -0.000,
0.902], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([8.008, 0.998,
2.705], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([8.008, -0.998,
4.508], ucell),constraints='123'),
                               Atom(type=O, position=Vector([5.447, 3.438,
4.508], ucell),constraints='123'),
                               Atom(type=O, position=Vector([5.447, 1.442,
2.705], ucell),constraints='123'),
                               Atom(type=O, position=Vector([7.175, 2.440,
0.902], ucell),constraints='123')],
                               unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000
sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_beta3_1layer_controla.nc",
ascii="NiSiO2_beta3_1layer_controla.txt")

sim.UpdateFromNetCDFFile("NiSiO2_beta3_1layer_control.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```



Appendix III

Input Files for Lattice Mis-Match

This appendix contains the significant input files for the lattice mis-match that simulated the presence of a non-ideal marriage of the Ni and SiO_2 complex. In the beginning of the input file, the offset along the *X* and *Z* axis is called out. The geometric illustrations were left out due to the minimal differences between them with respect to the control configurations displayed in Appendix I.

Ni/β SiO_2 Complex input file
1 layer of SiO_2

```
#!/usr/bin/env python

############################################################
#                                                          #
#   NiSiO2_-5Yaxis_halflayer_silica  input file for 1 layer of SiO2   #
#   offset by 0.5 Angstroms from origin along the Yaxis on Ni face  #
#                                                          #
############################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

############################################################
#                                                          #
# The Ni - O bond length is 1.93 A                         #
# Ni Lattice constant is 3.524  A                          #
# Therefore, add 4.614 A is lattice offset for Si and O atoms  #
# This offset is added to the X axis for all Si and O atoms    #
# Refer to Control input file for original X positions         #
# Orig file name:  SiO2_beta3_1layer_relax2                    #
#                                                          #
############################################################

############################################################
#                                                          #
# -.5 Y axis lattice miss match evaluation                 #
#                                                          #
############################################################
```

```
sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.257,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.005,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.257,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 0.752,
1.823], ucell),constraints='123'),
#                               Atom(type=Si, position=Vector([8.953, -0.500,
3.647], ucell),constraints='123'),
#                               Atom(type=Si, position=Vector([6.783, -1.753,
5.470], ucell),constraints='123'),
#                               Atom(type=Si, position=Vector([6.783, -1.753,
0.000], ucell),constraints='123'),
                                Atom(type=O, position=Vector([5.469, 3.030,
4.558], ucell),constraints='123'),
                                Atom(type=O, position=Vector([5.469, 0.980,
2.735], ucell),constraints='123'),
#                               Atom(type=O, position=Vector([8.098, -1.505,
4.558], ucell),constraints='123'),
#                               Atom(type=O, position=Vector([8.098, 0.525,
2.735], ucell),constraints='123'),
#                               Atom(type=O, position=Vector([6.323, -0.500,
0.912], ucell),constraints='123'),
                                Atom(type=O, position=Vector([7.306, 2.005,
0.912], ucell),constraints='123')],
                                unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(120)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Yaxis_halflayer_silica.nc",
ascii="NiSiO2_-5Yaxis_halflayer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Yaxis_halflayer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python

##########################################################
#                                                        #
#    NiSiO2_-5Yaxis_1layer_silica  input file for 2 layer of SiO2        #
#    offset by 0.5 Angstroms from origin along the Y axis on Ni face  #
#                                                        #
##########################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##########################################################
#                                                        #
# The Ni - O bond length is 1.93 A                       #
# Ni Lattice constant is 3.524  A                        #
# Therefore, add 4.614 A is lattice offset for Si and O atoms    #
# This offset is added to the X axis for all Si and O atoms      #
# Refer to Control input file for original X positions   #
# Orig file name:  SiO2_beta3_1layer_relax2              #
#                                                        #
##########################################################

##########################################################
#                                                        #
# -.5 Y axis lattice miss match evaluation               #
#                                                        #
##########################################################
```

```
sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                         Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                         Atom(type=Si, position=Vector([6.783, 3.257,
5.470], ucell),constraints='123'),
                         Atom(type=Si, position=Vector([4.614, 2.005,
3.647], ucell),constraints='123'),
                         Atom(type=Si, position=Vector([6.783, 3.257,
0.000], ucell),constraints='123'),
                         Atom(type=Si, position=Vector([6.783, 0.752,
1.823], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([8.953, -0.500,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
0.000], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.030,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 0.980,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, -1.505,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, 0.525,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([6.323, -0.500,
0.912], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.005,
0.912], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Yaxis_1layer_silica.nc",
ascii="NiSiO2_-5Yaxis_1layer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Yaxis_1layer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file

1 layer of SiO_2

```
#!/usr/bin/env python

##################################################################
#                                                                #
#    NiSiO2_-5Y-5Zaxis_halflayer_silica  input file for 1 layer of SiO2   #
#    offset by 0.5/0.5 Angstroms from origin along the Y/Z axis on Ni face
#
#                                                                #
##################################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                       [ 0.00000,  9.010000, -2.40000],
                       [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##################################################################
#                                                                #
#  The Ni - O bond length is 1.93 A                              #
#  Ni Lattice constant is 3.524  A                               #
#  Therefore, add 4.614 A is lattice offset for Si and O atoms   #
#  This offset is added to the X axis for all Si and O atoms     #
#  Refer to Control input file for original X positions          #
#  Orig file name:  SiO2_beta3_1layer_relax2                     #
#                                                                #
##################################################################

##################################################################
#                                                                #
#  -.5 Y and -.5 Z axis lattice miss match evaluation            #
#                                                                #
```

```
##########################################################

sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 3.524,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 0.000,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
-0.500], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 3.257,
4.970], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([4.614, 2.005,
3.147], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 3.257,
-0.500], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 0.752,
1.323], ucell),constraints='123'),
#                           Atom(type=Si, position=Vector([8.953, -0.500,
3.147], ucell),constraints='123'),
#                           Atom(type=Si, position=Vector([6.783, -1.753,
4.970], ucell),constraints='123'),
#                           Atom(type=Si, position=Vector([6.783, -1.753,
-0.500], ucell),constraints='123'),
                            Atom(type=O, position=Vector([5.469, 3.030,
4.058], ucell),constraints='123'),
                            Atom(type=O, position=Vector([5.469, 0.980,
2.235], ucell),constraints='123'),
#                           Atom(type=O, position=Vector([8.098, -1.505,
4.058], ucell),constraints='123'),
#                           Atom(type=O, position=Vector([8.098, 0.525,
2.235], ucell),constraints='123'),
#                           Atom(type=O, position=Vector([6.323, -0.500,
0.412], ucell),constraints='123'),
                            Atom(type=O, position=Vector([7.306, 2.005,
0.412], ucell),constraints='123')],
                            unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(120)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Y-5Zaxis_halflayer_silica.nc",
ascii="NiSiO2_-5Y-5Zaxis_halflayer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Y-5Zaxis_halflayer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python

##############################################################
#                                                            #
#    NiSiO2_-5Y-5Zaxis_1layer_silica  input file for 2 layer of SiO2
#
#    offset by 0.5/0.5 Angstroms from origin along the Y/Z axis on Ni face
#
#                                                            #
##############################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##########################################################
#                                                        #
# The Ni - O bond length is 1.93 A                       #
# Ni Lattice constant is 3.524  A                        #
# Therefore, add 4.614 A is lattice offset for Si and O atoms    #
# This offset is added to the X axis for all Si and O atoms      #
# Refer to Control input file for original X positions           #
# Orig file name:  SiO2_beta3_1layer_relax2                      #
#                                                        #
##########################################################

##########################################################
#                                                        #
# -.5 Y and -.5 Z axis lattice miss match evaluation             #
#                                                           #
#############################################################

sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
-0.500], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 0.000,
3.024], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.024], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
-0.500], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
-0.500], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.024], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
-0.500], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.024], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.262], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.262], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.024], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.262], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.262], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
-0.500], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.257,
4.970], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.005,
3.147], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.257,
-0.500], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 0.752,
1.323], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([8.953, -0.500,
3.147], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
4.970], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
-0.500], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.030,
4.058], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 0.980,
2.235], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, -1.505,
4.058], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, 0.525,
2.235], ucell),constraints='123'),
                              Atom(type=O, position=Vector([6.323, -0.500,
0.412], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.005,
0.412], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Y-5Zaxis_1layer_silica.nc",
ascii="NiSiO2_-5Y-5Zaxis_1layer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Y-5Zaxis_1layer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
1 layer of SiO_2

```
#!/usr/bin/env python

##############################################################
#                                                            #
#    NiSiO2_-5Y-25Zaxis_halflayer_silica  input file for 1 layer of SiO2 #
#    offset by 0.5/0.25 Angstroms from origin along the Y/Z axis on Ni
face #
#                                                                #
#################################################################



from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##############################################################
#                                                            #
# The Ni - O bond length is 1.93 A                           #
# Ni Lattice constant is 3.524  A                            #
# Therefore, add 4.614 A is lattice offset for Si and O atoms  #
# This offset is added to the X axis for all Si and O atoms    #
# Refer to Control input file for original X positions       #
# Orig file name:  SiO2_beta3_1layer_relax2                  #
#                                                            #
##############################################################

##############################################################
#                                                            #
# -.5 Y and -.25 Z axis lattice miss match evaluation        #
#                                                              #
################################################################

sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 3.524,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 0.000,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
-0.500], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 3.257,
5.220], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([4.614, 2.005,
3.397], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 3.257,
-0.250], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 0.752,
1.573], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([8.953, -0.500,
3.397], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([6.783, -1.753,
5.220], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([6.783, -1.753,
-0.250], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.030,
4.308], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 0.980,
2.485], ucell),constraints='123'),
#                             Atom(type=O, position=Vector([8.098, -1.505,
4.308], ucell),constraints='123'),
#                             Atom(type=O, position=Vector([8.098, 0.525,
2.485], ucell),constraints='123'),
#                             Atom(type=O, position=Vector([6.323, -0.500,
0.662], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.005,
0.662], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(120)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Y-25Zaxis_halflayer_silica.nc",
ascii="NiSiO2_-5Y-25Zaxis_halflayer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Y-25Zaxis_halflayer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python

############################################################
#                                                          #
#    NiSiO2_-5Y-25Zaxis_1layer_silica  input file for 2 layer of SiO2
 #
#    offset by 0.5/0.25 Angstroms from origin along the Y/Z axis on Ni
face #
#                                                          #
############################################################

from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

############################################################
#                                                          #
# The Ni - O bond length is 1.93 A                         #
# Ni Lattice constant is 3.524  A                          #
# Therefore, add 4.614 A is lattice offset for Si and O atoms
#
# This offset is added to the X axis for all Si and O atoms
#
# Refer to Control input file for original X positions
#
# Orig file name:  SiO2_beta3_1layer_relax2
#
#                                                          #
############################################################
```

```
################################################################
#                                                              #
#  -.5 Y and -.25 Z axis lattice miss match evaluation          #
#                                                              #
################################################################

sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 3.524,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
-0.500], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 0.000,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 3.524,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([0.000, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
3.024], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([3.524, 1.762,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 0.000,
1.262], ucell),constraints='123'),
                        Atom(type=Ni, position=Vector([1.762, 1.762,
-0.500], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([6.783, 3.257,
5.220], ucell),constraints='123'),
                        Atom(type=Si, position=Vector([4.614, 2.005,
3.397], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.257,
-0.250], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 0.752,
1.573], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([8.953, -0.500,
3.397], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
5.220], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.753,
-0.250], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.030,
4.308], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 0.980,
2.485], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, -1.505,
4.308], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, 0.525,
2.485], ucell),constraints='123'),
                              Atom(type=O, position=Vector([6.323, -0.500,
0.662], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.005,
0.662], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-5Y-25Zaxis_1layer_silica.nc",
ascii="NiSiO2_-5Y-25Zaxis_1layer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-5Y-25Zaxis_1layer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
1 layer of SiO_2

```
#!/usr/bin/env python

################################################################
#                                                              #
#    NiSiO2_-25Yaxis_halflayer_silica  input file for 1 layer of SiO2    #
#    offset by 0.25 Angstroms from origin along the Y axis on Ni face    #
#                                                              #
################################################################

from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

##############################################################
#                                                            #
# The Ni - O bond length is 1.93 A                           #
# Ni Lattice constant is 3.524  A                            #
# Therefore, add 4.614 A is lattice offset for Si and O atoms  #
# This offset is added to the X axis for all Si and O atoms    #
# Refer to Control input file for original X positions         #
# Orig file name:  SiO2_beta3_1layer_relax2                    #
#                                                            #
##############################################################

##############################################################
#                                                            #
# -.25 Y axis lattice miss match evaluation                    #
#                                                            #
##############################################################
```

```
sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.507,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.255,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.507,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 1.002,
1.823], ucell),constraints='123'),
#                             Atom(type=Si, position=Vector([8.953, -0.250,
3.647], ucell),constraints='123'),
#                              Atom(type=Si, position=Vector([6.783, -1.503,
5.470], ucell),constraints='123'),
#                              Atom(type=Si, position=Vector([6.783, -1.503,
0.000], ucell),constraints='123'),
                               Atom(type=O, position=Vector([5.469, 3.280,
4.558], ucell),constraints='123'),
                               Atom(type=O, position=Vector([5.469, 1.230,
2.735], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([8.098, -1.275,
4.558], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([8.098, 0.775,
2.735], ucell),constraints='123'),
#                              Atom(type=O, position=Vector([6.323, -0.250,
0.912], ucell),constraints='123'),
                               Atom(type=O, position=Vector([7.306, 2.255,
0.912], ucell),constraints='123')],
                               unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(120)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000

sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-25Yaxis_halflayer_silica.nc",
ascii="NiSiO2_-25Yaxis_halflayer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-25Yaxis_halflayer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```

Ni/β SiO_2 Complex input file
2 layers of SiO_2

```
#!/usr/bin/env python


##############################################################
#                                                            #
#    NiSiO2_-25Yaxis_1layer_silica  input file for 2 layer of SiO2      #
#    offset by 0.25 Angstroms from Origin on Ni                   #
#                                                             #
##############################################################


from Simulations.Dacapo import*
sim=Simulation()
from os import environ

Latticeconstant = 1.0000
cell = BravaisLattice ([[ 14.01000,  0.000000,  0.0000000],
                        [ 0.00000,  9.010000, -2.40000],
                        [ 0.00000,  0.00000000,  9.47000000]])

ucell = Latticeconstant*cell

############################################################
#                                                          #
#  The Ni - O bond length is 1.93 A                        #
#  Ni Lattice constant is 3.524  A                         #
#  Therefore, add 4.614 A is lattice offset for Si and O atoms     #
#  This offset is added to the X axis for all Si and O atoms      #
#  Refer to Control input file for original X positions          #
#  Orig file name:  SiO2_beta3_1layer_relax2                #
#                                                          #
############################################################

############################################################
#                                                          #
#  -.25 Y axis lattice miss match evaluation                #
#                                                         #
```

```
#######################################################

sim.atoms=ListOfAtoms ([Atom(type=Ni, position=Vector([0.000, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 3.524,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
0.000], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 0.000,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 3.524,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([0.000, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
3.524], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([3.524, 1.762,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 0.000,
1.762], ucell),constraints='123'),
                              Atom(type=Ni, position=Vector([1.762, 1.762,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.507,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([4.614, 2.255,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 3.507,
0.000], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, 1.002,
1.823], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([8.953, -0.250,
3.647], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.503,
5.470], ucell),constraints='123'),
                              Atom(type=Si, position=Vector([6.783, -1.503,
0.000], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 3.280,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([5.469, 1.230,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, -1.275,
4.558], ucell),constraints='123'),
                              Atom(type=O, position=Vector([8.098, 0.775,
2.735], ucell),constraints='123'),
                              Atom(type=O, position=Vector([6.323, -0.250,
0.912], ucell),constraints='123'),
                              Atom(type=O, position=Vector([7.306, 2.255,
0.912], ucell),constraints='123')],
                              unitcell=ucell)


sim.plancut = PlaneWaveCutoff(440)

#N = 3
#M = 9
#layers = 2
#elecperatom = 3

#sim.eband = ElectronicBands(10 + (N*M*layers*elecperatom)/2 +3)
sim.eband = ElectronicBands(154)

sim.dynamics = NetCDF.Entry(name="Dynamics", value="")
sim.dynamics.Type = "Relaxation"
sim.dynamics.Method = "ConjugateGradient"
sim.dynamics.Step = 1.20000
sim.eigensolver = NetCDF.Entry(name="electronicMinimization",
value="");
sim.eigensolver.Method = "eigsolve"
sim.eigensolver.DiagonalizationPerBand = 2

sim.conv=NetCDF.Entry(name="ConvergenceControl")
sim.conv.MaxNumberOfSteps=10000

sim.kpoints=NetCDF.Entry(name="kpointSetup",value=[15,15,15])

sim.Execute(outfile="NiSiO2_-25Yaxis_1layer_silica.nc",
ascii="NiSiO2_-25Yaxis_1layer_silica.txt")

sim.UpdateFromNetCDFFile("NiSiO2_-25Yaxis_1layer_silica.nc")
#print "Total Potential Energy = ", sim.atoms.GetTotalPotentialEnergy()
#print "Forces on Atoms        = ", sim.atoms.GetCartesianForces()
```