Semi-Empirical Tight-Binding Model Applied to Deep Traps in Gallium Arsenide

By Kurtis Wickey

We developed a program to explore the electronic structure of Gallium Arsenide with impurities placed in the crystal. We used the sp3s* tight binding method (also called LCAO). We considered spin-orbit coupling as well as central cell potential due to the presence of an impurity. We are in the process of considering d-orbitals and long range Coulomb potential. The information generated by this program will come in handy when experiments begin, sometime this fall.

We are attempting to describe the energy bands of Gallium Arsenide doped with impurities using the sp^3s* tight binding model. The reason for calculating these energy bands is to explore different dopant ions that could act as qubit [5] in a semiconductor-based quantum computer. There is currently an effort at Michigan State University to build a quantum computer based on deep impurities in GaAs. My project was to develop software to calculate the electronic structure of deep impurities. This knowledge is important for the experimental implementation since will allow us to screen different impurities for an optimal design of the quantum computer. We first calculated the bands for the un-doped Gallium Arsenide, and then compared them to the bands for the doped case. There are extra energy levels introduced by the impurity that form impurity bands. One impurity band in particular we are interested in, and that is the one located between the valence bands and the conduction bands. The impurity band can represent either a deep trap or a shallow trap. They are both relevant for quantum computing implementations.

The model we are using is semi-empirical. This means we are using parameters from literature, such as overlap integrals (orbital interaction between atoms), atomic energies, and spin orbit parameters that have been fitted from the experiment. We assume the lattice has periodic boundary conditions (where the last atom wraps around and connects to the first). This assumption is valid if we are considering a very large
lattice, and it makes the math much easier (no edge effects). In the tight binding model, we are assuming that wavefunction can be represented by basis functions:

$$\Phi(\vec{r}) = \sum_{m=1}^{N} \varphi_m u_m(\vec{r})$$

(1)

$N$ is the number of basis functions. $\varphi_m$ is the coefficient. $u_m$ is the basis function. We can reduce the size of the Hamiltonian if we choose our basis functions to look like our expected wavefunction [1]. With this in mind, we choose our basis functions to be atomic orbitals. These are known as Bloch functions [2]. Our atomic orbitals would look like:

$$u_m(\vec{r}) = \frac{1}{\sqrt{N}} \sum_{k} e^{i\vec{k} \cdot \vec{R}_i} \phi_n(\vec{r} - \vec{R}_i)$$

(2)

With $R_i$ being the position of the atom, and $k$ our wave vector. $N$ is the number of atoms in the lattice. This method gives rise to the alternate name for the tight binding model; Linear Combination of Atomic Orbitals (LCAO) [2]. If we consider more orbitals, then our corresponding Hamiltonian will make a better approximation of the system. In our model, we are considering 5 orbitals as our Bloch functions; the valence s, p, and an additional orbital, s*, which acts like a ‘and the rest’ orbital. Ideally, we would include d-orbitals to obtain a better approximation, but the s* approximation works fairly well. We are, however, accounting only for first neighbor interactions (where the norm of the position vector equals $\sqrt{3}/4$), neglecting the second neighbor interactions. Our Hamiltonian is simplified without a significant loss of accuracy. We are also discounting lattice distortions created by an impurity as they are difficult to model, and make only small changes.
Using the tight-binding model, we need a way to sum the orbitals over all the atoms in the semi-conductor. Fortunately, we can take advantage of symmetries in the lattice and do not need to sum up each atom individually. Gallium Arsenide is composed of Gallium atoms spaced regularly in a face centered cubic (fcc) structure. The Arsenic atoms are also spaced regularly in a fcc and are displaced from the Gallium atoms by the vector $[1/4, 1/4, 1/4]*a$ ($a$ is the length of one side of the simple cell). The Ga lattice is periodic in nature, and it is a called a Bravais lattice [4]. We can construct the crystal lattice (right now excluding As) by a set of three vectors, formally called primitive vectors: $[.5,0,.5]$ $[0,.5,.5]$ $[.5,.5,0]$. To take full advantage of the periodicity, we need to choose the smallest possible cell, known as the unit cell, which repeats.

![Diamond lattice](image)

Fig. 1. A diamond lattice, the light atoms can be thought of as Ar, and the dark as Ga. The light atoms can be thought of as a face centered cubic lattice offset from the dark atoms, another fcc lattice.

In this case, the unit cell turns out to be only one Ga atom and its corresponding As atom (displaced by the aforementioned vector) [1]. The resulting Hamiltonian is 10 by 10 (5 orbitals per atom, two atoms). In this Hamiltonian, we will need to take account for the fact that there are four nearest neighbor interactions between a Ga and the four As
surrounding it in a tetrahedral. However, since we only have one As atom in our unit cell, we will need to multiply every off-diagonal term by 4 to compensate.

There is a problem with this approach, namely, it assumes that every atom in the lattice is Ga and every offset atom is As. What we would like to do is to introduce impurities in the system. This reduces the periodicity of the lattice, and as a result, we can not use this simple unit cell any longer. We are forced to use a larger cell; one that will require different treatment.

We now consider a cubic cell of arbitrary length $a$. If $a$ is 1, then the lattice will have a total of 4 Ga atoms and 4 As atoms. (This is discounting three of the faces of the cube. We are counting the $x=0$, $y=0$, and $z=0$ faces only). Using the primitive vectors, we can calculate the position of the Ga atoms, as well as all 4 of their nearest neighbor As atoms. The Hamiltonian of the system is now 40 by 40 (5 orbitals, 8 atoms). Now instead of merely multiplying every off-diagonal element by 4 to account for the nearest neighbor As atoms not in the cell, we are going to use a tau vector that relates a Ga atom on an edge to As atoms inside the cell by translating the Ga atom onto the other side and finding its nearest neighbors. These plus the nearest neighbors from the other side will equal four. This allows us to recognize which entries in the Hamiltonian will have non-zero bonding terms. Formally, this looks like:

$$\langle \Psi_\alpha | H | \Psi_\beta \rangle = \sum_n e^{ik \cdot (\tau_n + d_\alpha - d_\beta)}$$  \hspace{1cm} (3)

$\tau_n$ is the tau vector, and $d_\alpha$, $d_\beta$ are the position vectors for the atoms in question.

It is necessary to restrict the values of the wave vector $k$ due to assumption of periodic boundary conditions and finite lattice size. The wave vector is periodic, with values of $k \cdot a$ differing by $2\pi$ being non-distinct ($a$ being the length of the unit cell) [1].
We restrict the values of $k$ from –half period to half period. This is formally known as the 1st Brillouin zone. We can determine this Brillouin zone by first constructing the reciprocal lattice.

The reciprocal lattice vectors are defined as:

$$
\vec{b}_1 = 2\pi \cdot \frac{\vec{a}_2 \times \vec{a}_3}{\vec{a}_1 \cdot (\vec{a}_2 \times \vec{a}_3)} \quad \vec{b}_2 = 2\pi \cdot \frac{\vec{a}_3 \times \vec{a}_1}{\vec{a}_1 \cdot (\vec{a}_2 \times \vec{a}_3)} \quad \vec{b}_3 = 2\pi \cdot \frac{\vec{a}_1 \times \vec{a}_2}{\vec{a}_1 \cdot (\vec{a}_2 \times \vec{a}_3)}
$$

(4)

The original lattice vectors are $a_1$, $a_2$, $a_3$ [4]. The wave vector $k$ can range through the parallelepiped defined by the reciprocal lattice vectors. However, it is customary to graph certain points of interest (the $\Gamma$ point, $k=0$, and the $X$ point, $k=[1,0,0]*2\pi/a$) in a certain way.

**Energy Bands for Gallium Arsenide**

![Energy Bands](image-url)
Fig. 2. Calculated eigenvalues of the Hamiltonian are plotted against the ratio of K to the L point, and the X point. Notice that there are only 8 bands, and we expect 10 eigenvalues from the size of the Hamiltonian. This means that two are degenerate. The top valence band is degenerate, and the third conduction band is degenerate.

At the $\Gamma$ point ($k=0$), the valence bands (typically near zero and below) reach their maximum energy. Also at the $\Gamma$ point, the conduction bands (those above zero) reach their minimum value [3]. We can see then that the smallest band gap (between the conduction bands and the valence bands) is located at the $\Gamma$ point. This value is about 1.54 eV for GaAs.

The top of the valence band is not represented very well in this model. We have neglected electron spins and spin-orbital coupling. Spin-orbital coupling arises from the fact that electrons have two possible spins. They have a magnetic moment, and they are also traveling around the nucleus. Since electrons carry a charge and they are moving, there is an effective magnetic field. The magnetic moment aligns with this effective magnetic field, and this effect is called spin-orbital coupling. The coupling depends on the angular momentum of the electron. This means that we will have coupling terms only for the p orbitals (since s orbital electrons angular momentum is zero). We can correct for this by adding the matrix $H_{so}$ to our Hamiltonian. $H_{so}$ is defined as:

$$H_{so} = \frac{q\hbar}{4m^2c^2}\sigma \cdot (\vec{E} \times \vec{p})$$

(5)

$\sigma$ is the Pauli spin matrices defined as:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

(6)

This correction matrix accounts for spin-orbital coupling between spin up, and spin down electrons on the same atom. This effect does not take place between atoms [1]. Since we
are differentiating between up spin and down spin, the size of the Hamiltonian is doubled with the top left quadrant and the bottom right quadrant being the same as our original Hamiltonian. The off diagonal quadrants are zero, until $H_{so}$ is added.

![Spin-Orbital Coupling](image)

Fig. 3. Calculated eigenvalues are plotted against the ratio of $k$. Notice the splitting of the top valence band. No longer is this band degenerate when the spin-orbital correction is present. Also, the top of the valence band increases by a small fraction.

The top valence band is behaving precisely what we would expect. If the electrons were experiencing an effective magnetic field, then their energy levels would change, either up or down depending on their magnetic moments.

We are now ready to introduce the impurity. Essentially, all we are doing is to pluck a Ga atom out of the lattice, and replace it with a different element. In our model, we are considering a short range central cell potential, which is the difference in atomic
energies between the original atom and the one that is replacing it. This can be easily done by adding a $V_0$ term to the diagonal elements of the Hamiltonian where the impurity is [6]. $V_0$ is defined as:

$$V_0 = \varepsilon_{\text{imp}}^h - \varepsilon_{\text{Ga}}^h$$ \hspace{1cm} (7)

The $E$ is the atomic energy for each orbital, not the average as others have done (Hjalmarson). We found that the results fit better with our approximation then by the average method. The off-diagonal hopping terms are also affected, but we do not take them into account. The main contribution is from the diagonal term (since the diagonal terms is always larger than the off-diagonal terms). An approximation is being made, but we believe a valid one. We are also adjusting the spin-orbit parameters in the $H_{so}$ matrix since the impurity is going to have different splitting effects than the original Ga atom.

We have set up the model in such a way that not only does one get to choose the number of impurities in the cell, but also their location. It turns out that the energy bands are independent of the impurities position. (Which is as we expect, logically we are replacing an atom in a very large periodic lattice. For every atom, there exists another atom exactly like the first.)
Fig. 4. Calculated eigenvalues are plotted against k (from the Γ point to the X point). Notice the extra band in the Band Gap introduced by Zinc.

Since the extra band is located well into the Band gap, it is known as a deep trap. A deep trap is defined as being more than 0.025 eV from either the conduction or valence bands [6]. Otherwise, it is a shallow trap. This is fortunate for our model, since with deep traps, central cell potential is the dominating force. We can safely neglect the long range Coulomb potential introduced when an impurity has either more or less electrons than the original atom. However, with shallow traps, such an assumption is not valid, since central cell potential is not the dominating force, but rather the Coulomb force. Notice that the X point has changed from [1,0,0]*2*pi to [1,0,0]*pi. This is because our Brillouin zone has changed. When introducing impurities into the lattice, it becomes necessary to adopt a larger size cell; otherwise the effects of the impurity become too dominant.
The next step for this model is to include the Coulomb potential in order to successfully model the shallow traps. We are also planning on incorporating d-orbitals sometime in the near future. Lattice distortions, however, are unlikely to be added due to their complexity. At some point efficiency outweighs accuracy.

We wrote our program using FORTRAN 90/95. We compiled it on a Linux machine using the compiler ifort. We linked the libraries lapack, and g2c to our program for our diagonalization routine (see attached code). We found it necessary to manually increase the stack size as we were surpassing the default memory limit.

In conclusion, we have developed a program that calculates the eigenvalues of the Hamiltonian for GaAs with and without an impurity present in the crystal. We generated the Hamiltonian using the semi-empirical tight-binding model. We used the atomic orbitals as basis vectors in Hilbert space. We then wrote the wavefunction as a linear combination of the basis vectors. The eigenvalues generated can be plotted against the wave vector k to obtain the energy bands. Looking at these energy bands, we can observe whether an impurity creates a deep or shallow trap. This knowledge will be useful when the experiments begin sometime in the fall.
Acknowledgements

With special thanks to:

Carlo Piermarocchi
Guillermo Quinteiro
George Perkins
Michael Katkov
References


program gallium

type position
  real*8, dimension(3) :: vector
end type position

type(position), allocatable :: GA(:,), AR(:,)
integer*8 :: n,i,j,o,l,INFO,number_of_loops,p,size,IL,IU,M1,dopants
real*8, dimension(3) :: k,max_k
complex*16, dimension(:,), allocatable :: Hamiltonian1,Z
complex*16, dimension(:,), allocatable :: Work
real*8, dimension(:,), allocatable :: RWORK,W
integer, dimension(:,), allocatable :: IWORK,IFAIL,dope
real*8 :: a,pi,norm,VL,VU,C,Eds,Edp,EdsX
real :: t1,t2
external fill,find

open(11,file='eigenvalues2.dat', status='replace')

pi=3.141592653589793; C=0

print *, 'how long is one side of the cube?'
read (*,*) size
print *, 'please enter the number of intervals: '
read (*,*) number_of_loops
print *, 'how many impurities?'
read(*,*) dopants

n=80*size**3; m=2*n-1
allocate(Z(n,n)); allocate(Hamiltonian1(n,n)); allocate(W(n)); allocate(IWORK(5*n)); allocate(IFAIL(n)); allocate(dope(dopants))
allocate(Work(m)); allocate(RWORK(7*n)); allocate(GA(4*size**3)); allocate(AR(4*size**3))

print *, 'where is the impurity? (an integer from 1-4*size**3)'
read (*,*) dope
print *, 'enter in Atomic energy for dopant (s)'
read (*,*) Eds
print *, 'enter in Atomic energy for dopant (p)'
read(*,*) Edp
print *, 'enter in Atomic energy for dopant (s*)'
read(*,*) EdsX

max_k(1)=(pi*2.0)/size

call cpu_time(t1)

call find(GA,AR,size)

do p=1,number_of_loops
write(*,*) 'loop number :', p

k=0
k(1)=(pi*2.0)/(size*(number_of_loops-1))*(p-1)
write(*,*) 'filling hamiltonian'

call fill(Hamiltonian1,GA,AR,size,k,dopants,dope,Eds,Edp,EdsX)

write(*,*) 'k = ', k(1)
write(*,*) 'finding eigenvalues'

call ZHEEVX('N','A','U',n,Hamiltonian1,n,VL,VU,IL,IU,C,M1,W,Z,n,Work,m,RWORK,INFO)

do i=1,n
   write (11,*) k(1)/max_k(1), W(i)
end do
end do

call cpu_time(t2)

write(*,*) 'time taken was',t2-t1,'seconds'

deallocate(Z); deallocate(Hamiltonian1); deallocate(W); deallocate(IWORK);
deallocate(IFAIL);deallocate(dope)
deallocate(Work); deallocate(RWORK); deallocate(GA); deallocate(AR)
close(11)
end program gallium

subroutine fill(Hamiltonian1,GA,AR,size,k,dopants,dope,Eds,Edp,EdsX)
type position
   real*8, dimension(3) :: vector
end type position

integer, intent(in) :: size, dopants
complex*16, intent(inout), dimension(80*size**3,80*size**3) :: Hamiltonian1
type(position), intent(in), dimension(4*size**3) :: GA,AR
real*8, intent(in), dimension(3) :: k
real*8, intent(in) :: Eds,Edp,EdsX
integer, intent(out) :: shift

integer, dimension(size) :: AR_index
real*8, dimension(3) :: temp, dif_tau
complex*16 :: imag
type(position), dimension(4) :: difference

external nearest_neighbor1, nearest_neighbor2, orbitals

! 1 is for type position, 2 is for scalar type
! dif_tau is the tau m vector plus the difference of the position vectors

imag=cmplx(0,1); shift=20*size**3; Hamiltonian=0; Hso=0; Hamiltonian1=0

Esa=-8.3431; Esc=-2.6569; Epa=1.0414; Epc=3.6686; EsXa=8.5914; EsXc=6.7386

E_s=-11.55; E_p=-5.67; E_sX=-2.6; Dc=.013/3.0; Da=.38/3.0

! for closest neighbor (s orbital) interaction
do j=1,4*size**3

p=j

! for origin
if (GA(j)%vector(1)==0 .and. GA(j)%vector(2)==0 .and. GA(j)%vector(3)==0) then

! s orbitals
  call nearest_neighbor1(GA,AR,size,AR_index,p,difference)
  call orbitals(difference(1)%vector,Hamiltonian,size,k,dif_tau,AR_index(1),j)
  temp=GA(j)%vector+[size,size,0]
  call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
  dif_tau=[size,size,0]+difference(1)%vector
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)
  temp=GA(j)%vector+[size,0,size]
  call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
  dif_tau=[size,0,size]+difference(1)%vector
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)
  temp=GA(j)%vector+[0,size,size]
  call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
  dif_tau=[0,size,size]+difference(1)%vector
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

  ! for the z-edge
else if (GA(j)%vector(1)==0 .and. GA(j)%vector(2)==0) then

  call nearest_neighbor1(GA,AR,size,AR_index,p,difference)
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)
  temp=GA(j)%vector+[size,size,0]
  call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
  dif_tau=[size,size,0]+difference(1)%vector
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)
  temp=GA(j)%vector+[size,0,0]
  call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
  dif_tau=[size,0,0]+difference(1)%vector
  temp=difference(1)%vector
  call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

end if
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[size,0,0]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,size,0]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,size,0]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

! for the y-edge
else if (GA(j)%vector(1)==0 .and. GA(j)%vector(3)==0) then

call nearest_neighbor1(GA,AR,size,AR_index,p,difference)
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[size,0,0]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[size,0,0]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[size,0,size]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[size,0,size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

! for the x edge
else if (GA(j)%vector(2)==0 .and. GA(j)%vector(3)==0) then

call nearest_neighbor1(GA,AR,size,AR_index,p,difference)
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,size,0]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,size,0]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,0,size]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,0,size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,0,size]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,0,size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,0,size]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,0,size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,0,size]
call nearest_neighbor2(temp,AR,size,AR_index,p,difference)
dif_tau=[0,0,size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(1),j)

temp=GA(j)%vector+[0,0,size]
call nearest_neighbor2(temp, AR, size, AR_index, p, difference)
dif_τ=[0, size, size]+difference(1)%vector
temp=difference(1)%vector
call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(1), j)

! for the y-z face
else if(GA(j)%vector(1)==0) then

call nearest_neighbor1(GA, AR, size, AR_index, p, difference)
do i=1,2
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

temp=GA(j)%vector+[size, 0, 0]
call nearest_neighbor2(temp, AR, size, AR_index, p, difference)
do i=1,2
   dif_τ=[size, 0, 0]+difference(i)%vector
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

! for the x-z face
else if(GA(j)%vector(2)==0) then

call nearest_neighbor1(GA, AR, size, AR_index, p, difference)
do i=1,2
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

temp=GA(j)%vector+[0, size, 0]
call nearest_neighbor2(temp, AR, size, AR_index, p, difference)
do i=1,2
   dif_τ=[0, size, 0]+difference(i)%vector
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

! for the x-y face
else if(GA(j)%vector(3)==0) then

call nearest_neighbor1(GA, AR, size, AR_index, p, difference)
do i=1,2
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

temp=GA(j)%vector+[0, 0, size]
call nearest_neighbor2(temp, AR, size, AR_index, p, difference)
do i=1,2
   dif_τ=[0, 0, size]+difference(i)%vector
   temp=difference(i)%vector
   call orbitals(temp, Hamiltonian, size, k, dif_τ, AR_index(i), j)
end do

else

call nearest_neighbor1(GA,AR,size,AR_index,p,difference)
do i=1,4
    temp=difference(i)%vector
    call orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index(i),j)
end do
end if
end do

! the diagonal

do j=1,dopants
    do i=1,2*shift

        if (i==dope(j)) then
            Hamiltonian(i,i)=(Eds-E_s+Esc)
        else if(i<=4*size**3) then
            Hamiltonian(i,i)=Esc
        else if (i==(dope(j)+4*size**3) .or. i==(dope(j)+8*size**3) .or. i==(dope(j)+12*size**3)) then
            Hamiltonian(i,i)=(Edp-E_p+Epc)
        else if(i>4*size**3 .and. i<=16*size**3) then
            Hamiltonian(i,i)=Epc
        else if(i==(dope(j)+16*size**3)) then
            Hamiltonian(i,i)=(EdsX-E_sX+EsXc)
        else if (i>16*size**3 .and. i<=20*size**3) then
            Hamiltonian(i,i)=EsXc
        else if (i>20*size**3 .and. i<=24*size**3) then
            Hamiltonian(i,i)=Esa
        else if (i>2*shift-4*size**3) then
            Hamiltonian(i,i)=Epa
        else if (i>2*shift-4*size**3) then
            Hamiltonian(i,i)=EsXa
        end if

    end do
end do

! assigning Hamiltonian to the 1st and 4th quadrants of Hamiltonian1

do i=1,2*shift
    do j=1,2*shift
        if(i>j) then
            Hamiltonian(i,j)=conjg(Hamiltonian(j,i))
        end if
    end do
end do

assigning Hamiltonian to the 1st and 4th quadrants of Hamiltonian1

do i=1,2*shift
    do j=1,2*shift
        if(i>j) then
            Hamiltonian1(i,j)=Hamiltonian(i,j)
            Hamiltonian1(i+2*shift,j+2*shift)=Hamiltonian(i,j)
            if(Hamiltonian1(i,j+2*shift)/=0 .or. Hamiltonian1(i+2*shift,j)/=0) then
                write(*,*),'error: non-zero entries in Hso'
end if
end do
end do

! calculate Hso

do i=1,4*shift
do j=1,4*shift
if (j>i) then

! Gallium spin orbital coupling

if(i==4*size**3+1 .and. j==8*size**4+1) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-imag*Dc
end do
else if (i==4*size**3+1 .and. j==(2*shift+12*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=Dc
end do
else if (i==8*size**3+1 .and. j==(2*shift+12*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-imag*Dc
end do
else if (i==12*size**3+1 .and. j==(2*shift+4*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-Dc
end do
else if (i==12*size**3+1 .and. j==(2*shift+8*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=imag*Dc
end do
else if (i==(2*shift+4*size**3+1) .and. j==(2*shift+8*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=imag*Dc
end do

! Arsenic spin orbital coupling (p-orbitals)

else if (i===(shift+4*size**3+1) .and. j===(shift+8*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=imag*Da
end do
else if (i===(shift+4*size**3+1) .and. j===(3*shift+12*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=Da
end do
else if (i===(shift+8*size**3+1) .and. j===(3*shift+12*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-imag*Da
end do
else if (i===(shift+12*size**3+1) .and. j===(3*shift+12*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-imag*Da
end do
else if (i===(shift+12*size**3+1) .and. j===(3*shift+8*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=Da
end do
else if (i===(shift+12*size**3+1) .and. j===(3*shift+4*size**3+1)) then
do p=0,4*size**3-1
   Hso(i+p,j+p)=-Da
end do
else if (i===(shift+12*size**3+1) .and. j===(3*shift+8*size**3+1)) then
end do
end do
end if
do p=0,4*size**3-1
    Hso(i+p,j+p)=imag*Da
end do
else if (i==((3*shift+4*size**3+1) .and. j==((3*shift+8*size**3+1)) then
    do p=0,4*size**3-1
        Hso(i+p,j+p)=imag*Da
    end do
end if
end if
end do
end do

do i=1,4*shift
    do j=1,4*shift
        if(i>j) then
            Hso(i,j)=conjg(Hso(j,i))
        end if
    end do
end do

Hamiltonian1=Hamiltonian1+Hso
return
end subroutine fill

!locates the position of the atoms
subroutine find(GA,AR,size)

type position
    real*8, dimension(3) :: vector
end type position

integer, intent(inout) :: size

type(position), intent(inout), dimension(4*size**3) :: GA,AR
real*8, dimension(3) :: v1,v2,v3,temp,a1
integer :: i,j,k,l,n

v1=[.5,.5,0.0]; v2=[.5,0.0,.5]; v3=[0.0,.5,.5]; n=0; a1=[.25,.25,.25]; m=0
l=2*size

do i=1,m
    GA(i)%vector=0
    AR(i)%vector=0
end do

do i=-l,l
    do j=-l,l
        do k=-l,l
            temp=i*v1+j*v2+k*v3
            if (temp(1)<size .and. temp(2)<size .and. temp(3)<size .and. temp(1)>=0 .and. temp(2)>=0 .and. temp(3)>=0) then
                n=n+1
                GA(n)%vector=temp
                AR(n)%vector=GA(n)%vector+a1
            end if
        end do
    end do
end do
end do
end do

return
end subroutine find

subroutine nearest_neighbor1(GA,AR,size,AR_index,n,difference)
type position
    real*8, dimension(3) :: vector
end type position

integer, intent(inout) :: size,n
type(position), intent(in), dimension(4*size**3) :: GA,AR
type(position), intent(out), dimension(4) :: difference
integer, intent(inout), dimension(4) :: AR_index
real*8, dimension(3) :: temp
real*8 :: norm
integer :: i,k

k=0
do i=1,4*size**3
    temp=GA(n)%vector-AR(i)%vector
    if(abs(norm(temp))==(DSQRT(dble(3.0))/dble(4.0))) then
        k=k+1
        AR_index(k)=i
        difference(k)%vector=temp
    end if
end do
return
end subroutine nearest_neighbor1

subroutine nearest_neighbor2(temp,AR,size,AR_index,n,difference)
type position
    real*8, dimension(3) :: vector
end type position

integer, intent(inout) :: size,n
real*8, intent(in), dimension(4*size**3) :: temp
type(position), intent(in), dimension(4*size**3) :: AR
type(position), intent(out), dimension(4) :: difference
integer, intent(inout), dimension(4) :: AR_index
real*8, dimension(3) :: temp1
real*8 :: norm
integer :: i,k

k=0

do i=1,4*size**3
    temp1=temp-AR(i)%vector
    if(abs(norm(temp1))==(DSQRT(dble(3.0))/dble(4.0))) then
        k=k+1
        AR_index(k)=i
    end if
end do
return
end subroutine nearest_neighbor2
difference(k)%vector=temp1
end if
end do
return
end subroutine nearest_neighbor2

subroutine orbitals(temp,Hamiltonian,size,k,dif_tau,AR_index,j)
integer, intent(in) :: size,j,AR_index
complex*16, intent(inout), dimension(40*size**3,40*size**3) :: Hamiltonian
real*8, intent(inout),dimension(3) :: temp,k,dif_tau
real*8 :: Ess,Esapc,Exx,Epasc,EsXapc,Exy,EpasXc
complex :: imag
integer :: shift,test1,test2,test3

Ess=-6.4513/4; Esapc=4.48/4; Exx=1.9546/4; Epasc=-5.7839/4; EsXapc=4.8422/4; Exy=5.0779/4;
EpasXc=-4.8077/4
imag=cmplx(0,1); shift=20*size**3

Hamiltonian(j,AR_index+shift)=Ess*exp(imag*dot_product(k,dif_tau))
Hamiltonian(j+4*size**3,AR_index+shift+4*size**3)=Exx*exp(imag*dot_product(k,dif_tau)) !x-orbital
Hamiltonian(j,AR_index+shift+4*size**3)=test1(temp)*Epasc*exp(imag*dot_product(k,dif_tau)) !s-orbital, x-orbital
Hamiltonian(j+4*size**3,AR_index+shift)=test1(temp)*Esapc*exp(imag*dot_product(k,dif_tau)) !x-orbital, s-orbital
Hamiltonian(j+4*size**3,AR_index+shift+8*size**3)=test1(temp)*test2(temp)*Exy*exp(imag*dot_product(k,dif_tau))!x-orbital, y-orbital
Hamiltonian(j+8*size**3,AR_index+shift+8*size**3)=Exx*exp(imag*dot_product(k,dif_tau))!y-orbital, y-orbital
Hamiltonian(j+8*size**3,AR_index+shift+4*size**3)=test1(temp)*test2(temp)*Exy*exp(imag*dot_product(k,dif_tau))!y-orbital, s-orbital
Hamiltonian(j+12*size**3,AR_index+shift+4*size**3)=test1(temp)*test3(temp)*Exy*exp(imag*dot_product(k,dif_tau))!z-orbital, x-orbital
Hamiltonian(j+12*size**3,AR_index+shift+12*size**3)=Exx*exp(imag*dot_product(k,dif_tau))!z-orbital, z-orbital
Hamiltonian(j+4*size**3,AR_index+shift+12*size**3)=test1(temp)*test3(temp)*Exy*exp(imag*dot_product(k,dif_tau))!x-orbital, z-orbital
Hamiltonian(j+12*size**3,AR_index+shift+8*size**3)=test2(temp)*test3(temp)*Exy*exp(imag*dot_product(k,dif_tau))!z-orbital, y-orbital
Hamiltonian(j+8*size**3,AR_index+shift+12*size**3)=test3(temp)*Epasc*exp(imag*dot_product(k,dif_tau)) !s-orbital, z-orbital
Hamiltonian(j,AR_index+shift+12*size**3)=test3(temp)*Esapc*exp(imag*dot_product(k,dif_tau)) !z-orbital, s-orbital
Hamiltonian(j,AR_index+shift+12*size**3)=test3(temp)*EpasXc*exp(imag*dot_product(k,dif_tau)) !s-orbital, z-orbital
Hamiltonian(j+16*size**3,AR_index+shift+12*size**3)=test3(temp)*EpasXc*exp(imag*dot_product(k,dif_tau))!*s*-orbital, z-orbital

Hamiltonian(j+12*size**3,AR_index+shift+16*size**3)=test3(temp)*EsXapc*exp(imag*dot_product(k,dif_tau))!z-orbital, s*-orbital

Hamiltonian(j+16*size**3,AR_index+shift+4*size**3)=test1(temp)*EpasXc*exp(imag*dot_product(k,dif_tau))!*s*-orbital, x-orbital

Hamiltonian(j+4*size**3,AR_index+shift+16*size**3)=test1(temp)*EsXapc*exp(imag*dot_product(k,dif_tau))!x-orbital,s*-orbital

Hamiltonian(j+16*size**3,AR_index+shift+8*size**3)=test2(temp)*EpasXc*exp(imag*dot_product(k,dif_tau))!*s*-orbital,y-orbital

Hamiltonian(j+8*size**3,AR_index+shift+16*size**3)=test2(temp)*EsXapc*exp(imag*dot_product(k,dif_tau))!y-orbital,s*-orbital

    return
end subroutine orbitals

real*8 function norm(f)
  real*8, dimension(3):: f
  norm=sqrt(f(1)**2+f(2)**2+f(3)**2)
return
end function norm

integer function test1(difference)
  real*8, dimension(3) :: difference
  if (difference(1)<0) then
    test1=-1
  else
    test1=1
  end if
return
end function test1

integer function test2(difference)
  real*8, dimension(3) :: difference
  if (difference(2)<0) then
    test2=-1
  else
    test2=1
  end if
return
end function test2

integer function test3(difference)
  real*8, dimension(3) :: difference
  if (difference(3)<0) then
    test3=-1
  else
    test3=1
  end if
return
end function test3