UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

FACULTAD DE CIENCIAS

Cálculos de estructura electrónica de la superficie del Germanio reconstruida en su cara (111)

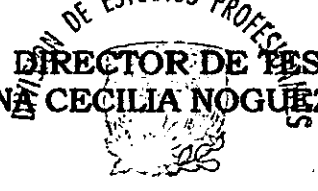
T E S I S

QUE PARA OBTENER EL TITULO DE:

FÍSICO

P R E S E N T A :

MARTÍN SOLÍS PÉREZ



DIRECTOR DE TESIS:
DRA. ANA CECILIA NOGUEZ GARRIDO

FACULTAD DE CIENCIAS
SECCION ESCOLAR
2000

280684

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

MAT. MARGARITA ELVIRA CHÁVEZ CANO
Jefa de la División de Estudios Profesionales
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"Cálculos de estructura electrónica de la superficie del Germanio reconstruída en su cara (111)"

realizado por SOLIS PEREZ MARTIN

Con número de cuenta 7012562-6 , pasante de la carrera de Física.

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director de tesis
Propietario
DRA. ANA CECILIA NOGUEZ GARRIDO
Propietario
DR. RUBEN GERARDO BARRERA PEREZ
Propietario
DR. CHUMIN WANG CHEN
Suplente
DRA. ROSA MARIA MENDEZ VARGAS
Suplente
DR. GUILLERMO MONSIVAIS GALINDO

Consejo Departamental de Física

DRA. PATRICIA GOLDSTEIN MENACHE
Coordinadora de Licenci

FACULTAD DE CIENCIAS
DEPARTAMENTO DE FISICA

Resumen

El propósito de este trabajo es el calcular la estructura electrónica de la superficie de Germanio reconstruida en su cara (111). Para llevar a cabo esta tarea vamos a emplear el modelo cuántico de Enlace Fuerte o *Tight Binding*. Para simplificar los cálculos utilizaremos la aproximación a primeros vecinos, y se usará la base extendida sp^3s^*, en donde s y s^* son orbitales atómicos con simetría esférica, el símbolo (*) denota un estado excitado, y además p representa orbitales atómicos con números cuánticos (l = 1).

Durante el presente trabajo se desarrollaron una serie de programas en lenguaje de programación C++ para que realicen el trabajo computacional que aparece al aplicar el método de Enlace Fuerte. Para verificar la ejecución correcta de los programas, se calculará primeramente la estructura electrónica del cristal cúbico de germanio, y de su superficie (111)-1×1 no reconstruida. Se comparan los resultados obtenidos con los datos teóricos y experimentales publicados por otros autores.

Finalmente se calcula la estructura electrónica de la superficie Ge(111)-2×1 reconstruida. Se analizan los datos obtenidos en el presente trabajo, y se comparan con los resultados teóricos y experimentales publicados por otros autores.

A Estela, Damián y Adolfo
mi esposa y mis pequeños hijos
quienes dan calor a mi vida.

CONTENIDO

	INTRODUCCIÓN	1
1	FUNCIONES DE BLOCH Y ESTADOS DE SUPERFICIES	4
1.1	Teorema de Bloch	7
1.2	La condición a la frontera de Born-Von Karman	8
1.3	Observaciones generales sobre el Teorema de Bloch	10
1.4	Estados de Superficie	11
1.5	Relajación y Reconstrucción	18
2	EL MÉTODO DE ENLACE FUERTE	21
2.1	Sumas de Bloch y el Método de Enlace Fuerte	23
2.2	Cálculo de la estructura electrónica del cristal de germanio	29
2.3	Formulación del método de Enlace Fuerte para superficies	37
2.4	Cálculo de la estructura electrónica de la superficie Ge(111)-1×	39
3	ESTRUCTURA ELECTRONICA DEL Ge(111)-2×1	44
3.1	Método de cálculo	45
3.2	Resultados y Discusión	48
3.3	Densidad de Estados	52
4	CONCLUSIONES	54
	LISTADOS DE PROGRAMAS	56
	Referencias y Bibliografía	81

Introducción

El germanio es un elemento químico cuya existencia fue predicha por Mendeleev en 1871. Él predijo que el elemento desconocido germanio debería tener sus propiedades químicas muy parecidas a las del silicio. Mendeleev sugirió llamarlo *Ekasilicon* (símbolo Es). Sus predicciones se acercaron mucho a la realidad. El germanio fue descubierto en un mineral llamado *argyrodite* en el año de 1886 por Clemens Alexander Winkler. El germanio es un semimetal de color gris claro, y en su estado puro es un cristal brillante que conserva su lustre a temperatura ambiente. El germanio es un material semiconductor muy importante, y sus usos principales son los siguientes:

- Se utiliza en transistores
- Se usa en lámparas fosforescentes y fluorescentes
- Como catalizador
- Se utiliza en equipo óptico, como son los detectores infrarrojos ultrasensibles, lentes de cámaras y objetivos de microscopios
- Se utiliza en detectores de radiación gama.

Por lo anterior, vemos que es importante estudiar las propiedades físicas del germanio, y como veremos en el presente trabajo, el estudio de la superficie del Ge(111)-2×1 reconstruida ha sido un tema que ha llamado la atención de muchos investigadores teóricos y experimentales. Aunque ha habido un progreso considerable, la estructura y las propiedades de esta superficie aún no se conocen detalladamente.

La superficie de un sólido cristalino en el vacío se define generalmente como las capas atómicas exteriores del sólido cuyas posiciones atómicas difieren de las posiciones que tendrían en el interior o sustrato del mismo. La superficie puede estar totalmente limpia o puede tener átomos extraños depositados sobre ella (adsorbatos).

Si la superficie está limpia, las capas atómicas superiores pudiesen estar reconstruidas, o a veces, no reconstruidas. En superficies no reconstruidas el ordenamiento atómico es coherente con la del sustrato del cristal, excepto por una variación de la separación entre capas en la superficie, a este fenómeno se le conoce como relajación. En la relajación, los átomos mantienen su estructura en el plano superficial como lo estaba de acuerdo con la proyección de la celda del seno del material sobre dicha superficie; sólo varía su distancia al seno del material. En la mayoría de las superficies metálicas se presenta la relajación superficial. En cambio, en la reconstrucción, los átomos de las capas superficiales forman estructuras en las cuales los átomos no están en concordancia con los átomos que forman el sustrato del cristal. La reconstrucción superficial puede ser una consecuencia de una redistribución de enlaces covalentes o iónicos rotos en la superficie.

En estas condiciones los átomos en la superficie se agrupan en filas con separaciones alternativamente mayores y menores que en el sustrato del cristal; la creación de una superficie dejaría enlaces no saturados que cuelgan hacia el espacio exterior, a estos enlaces se les conoce como enlaces colgantes o sueltos.

El estudio moderno de las superficies de semiconductores empezó en 1947 con la construcción de un dispositivo fabricado con silicio llamado transistor de emisión de campo (FET de sus siglas en inglés) en los Laboratorios Bell por William Shockley, John Bardeen, y Walter Brattian. Inmediatamente después de 1947 los investigadores se dieron cuenta que además de las propiedades del sustrato de cristal, el estudio de la superficie juega un papel muy importante en la caracterización de las propiedades del semiconductor.

En 1957 ya se tenía la tecnología para producir alto vacío (presiones inferiores a 10^{-9} torr) por lo que se podían llevar a cabo experimentos a estas presiones tan bajas. A través de los años se desarrollaron varios métodos para obtener superficies atómicas limpias: crecimiento epitaxial molecular, cortes mecánicos, bombardeo iónico, y técnicas de calentamiento, etc..

En la actualidad con la técnica de Ultra Alto Vacío (10^{-10} torr) se ha logrado producir superficies muy limpias en el laboratorio, y con la ayuda de las espectroscopías de superficie (Electrones de Baja Energía, Espectroscopía Electrónica de Auger) se ha logrado identificar a los átomos presentes en la superficie del sólido, así como a sus posiciones. Por otra parte, el estudio teórico de la Física de Superficies no tendría el nivel de conocimiento actual sin la ayuda de las computadoras modernas, debido a que es necesario realizar una gran cantidad de cálculos muy complejos.

Un tópico central en la Física moderna de superficies es el desarrollo de un modelo que permita comprender en forma detallada la estructura electrónica de una superficie. En la parte teórica, el método general es similar al que se emplea en el sustrato del cristal: en esencia se utiliza la aproximación de un electrón y se intenta resolver la ecuación de Schrödinger para un electrón cerca de la superficie. Una gran variedad de métodos aproximados pueden entonces ser utilizados para que tomen en cuenta el efecto de los demás electrones. El formalismo matemático para estudiar las superficies es más complicado respecto al del sustrato, debido a que en la superficie sólo existe simetría traslacional en las direcciones del plano de la superficie, y en la dirección perpendicular a la superficie la simetría se rompe. Un cálculo completo de la estructura electrónica de la superficie requiere conocer las nuevas posiciones de los átomos debido a que se presenta la relajación o reconstrucción de la superficie, y por lo tanto, los átomos son desplazados de sus posiciones ideales que ellos ocuparían si el sustrato del cristal se truncara en dos partes.

Por la gran importancia tecnológica del germanio, en el presente trabajo vamos a calcular la estructura electrónica de la superficie en la cara (111) del germanio con una reconstrucción 2×1. Para llevar a cabo dicha tarea, vamos a utilizar el método de Enlace Fuerte (*Tight Binding*) utilizando una base de orbitales atómicos sp^3s^*. Los programas computacionales que se desarrollaron para llevar a cabo los cálculos correspondientes, se escribieron en lenguaje de programación C++ que corre en un ambiente LINUX Red Hat 5.x o mayor. Se eligió al lenguaje C++ por ser orientado a objetos; el Hamiltoniano es una matriz, y toda matriz puede ser representada en el programa como un objeto que se crea (toma recursos del sistema operativo dinámicamente) o se destruye (libera los recursos utilizados dinámicamente) a petición del programa. C++ permite una implementación muy sencilla de los algoritmos que involucran cálculo matricial, como es la obtención de los valores y vectores propios de una matriz. Además es un lenguaje de programación que ha tenido una gran aceptación en el mundo científico.

Los cálculos obtenidos los vamos analizar en función de la estructura atómica del sistema y los vamos a comparar con la estructura electrónica medida en los experimentos, y con otros resultados teóricos. Nuestro interés en realizar cálculos de la estructura de bandas de la superficie Ge(111)-2x1 es comparar nuestros resultados, usando la aproximación de Enlace Fuerte en donde se utiliza una base de orbitales atómicos, con los resultados teóricos obtenidos por otros autores que en su mayoría se realizaron usando una base de ondas planas. Dada la naturaleza del problema, es de suponerse que una base de orbitales atómicos es más eficiente que la base de ondas planas para encontrar estados electrónicos de superficie como se discutirá más adelante.

CAPITULO I

FUNCIONES DE BLOCH Y ESTADOS DE SUPERFICIE

En un cristal perfecto los iones están colocados en un arreglo periódico, esto nos lleva a considerar el problema de un electrón sujeto a un potencial periódico $U(\mathbf{r})$ que tiene una periodicidad igual al de la red de Bravais [1], es decir,

$$U(\mathbf{r}+\mathbf{R}) = U(\mathbf{r}), \tag{1.1}$$

para todos los vectores $\mathbf{R}$ de la red de Bravais. Como la escala en la periodicidad del potencial $U(\mathbf{r})$ es aproximadamente 10^{-10} m, es esencial utilizar a la mecánica cuántica para conocer los efectos del potencial sobre el electrón.

Debemos reconocer que la periodicidad perfecta es una idealización. Los sólidos en realidad no son absolutamente puros, y en una vecindad alrededor de los átomos de impureza el cristal cambia respecto a otras regiones dentro del mismo sólido. Para encontrar las soluciones al problema, lo dividimos en dos partes:

i) Se considera al sólido como un cristal ideal, esto es, el potencial es periódico

ii) Las desviaciones en las propiedades de un cristal real respecto al ideal, debidas a que la periodicidad no es perfecta se tratan como perturbaciones pequeñas.

El problema de un electrón en un sólido es en realidad un problema de muchos electrones que interactúan con los iones, ya que el Hamiltoniano del sólido contiene además de los potenciales entre electrones y iones, los potenciales que describen las interacciones electrón-electrón.

La ecuación de Schrödinger para un cristal es

$$\mathbf{H} = \sum_{i=1}^{n} \frac{\mathbf{p}_i^2}{2m_e} + \sum_{i=1}^{N} \frac{\mathbf{P}_I^2}{2M} + \sum_{i>j} \frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{I>J} \frac{e^2}{|\mathbf{R}_I - \mathbf{R}_J|} - \sum_{i,I} \frac{e^2}{|\mathbf{r}_i - \mathbf{R}_I|}, \tag{1.2}$$

en donde el primer término es la energía cinética de los electrones, el segundo término es la energía cinética de los iones, el tercer término es la interacción entre electrones, el cuarto término es la interacción entre iones, y finalmente el quinto término es la interacción entre los iones y los electrones.

Para simplificar los cálculos, nos apoyaremos en la aproximación adiabática que considera que los iones se mantienen fijos junto con los electrones más ligados a ellos (coraza), esta suposición se sustenta en que el cociente entre la masa del electrón entre la masa del núcleo es mucho menor que 1, es decir, $\frac{m_e}{M} << 1$. Es conveniente en este caso descomponer al Hamiltoniano en dos partes, el Hamiltoniano debido a los iones ($\mathbf{H}_I$) más el Hamiltoniano debido a los electrones ($\mathbf{H}_e$):

$$\mathbf{H} = \mathbf{H}_I + \mathbf{H}_e , \tag{1.3}$$

en donde,

$$\mathbf{H}_I = \sum_{i=1}^{N} \frac{\mathbf{P}_I^2}{2M} + \sum_{I \neq J} \frac{e^2}{|\mathbf{R}_I - \mathbf{R}_J|}, \tag{1.4}$$

$$\mathbf{H}_e = \sum_{i=1}^{n} \frac{\mathbf{p}_i^2}{2m_e} + \sum_{i>j} \frac{e^2}{\mathbf{r}_i - \mathbf{r}_j} - \sum_{i,I} \frac{e^2}{\mathbf{r}_i - \mathbf{R}_I} = \mathrm{T}(\mathbf{p}_i) + \mathrm{U}(\mathbf{r}_i, \mathbf{r}_j) . \tag{1.5}$$

Aún con esta separación del Hamiltoniano, seguimos teniendo un problema de muchos cuerpos. Lo simplificamos sustituyendo el potencial $\mathrm{U}(\mathbf{r}_i, \mathbf{r}_j)$ por un potencial efectivo $\mathrm{U}_{eff}(\mathbf{r}_i)$ de un sólo cuerpo que siente de manera efectiva cada uno de los electrones y que es determinado utilizando distintos esquemas de aproximación, tales como el Hartree, Hartree-Fock o funcional de la densidad.

Como se puede ver de esta expresión, si el arreglo de los iones es periódico, también lo es el potencial efectivo. Con esta aproximación, denominada de electrón independiente, para cada uno de los electrones se puede escribir un Hamiltoniano $\mathbf{h}_i$ de la forma:

$$\mathbf{h}_i = \frac{\hbar^2}{2m_e} \nabla_i^2 + \mathrm{U}_{eff}(\mathbf{r}_i) , \tag{1.6}$$

en donde m_e es la masa del electrón y $\mathrm{U}_{eff}(\mathbf{r}_i)$ es un potencial periódico debido a los iones y a los electrones ligados a ellos. Finalmente el Hamiltoniano que tenemos que resolver para el cristal lo podemos escribir como,

$$\mathbf{H}_e = \sum_i \mathbf{h}_i \tag{1.7}$$

Como podemos ver, se considera que cada electrón obedece a una ecuación de Schrödinger con un potencial periódico, a estos electrones se les conoce como electrones independientes o de Bloch.

Los estados estacionarios de los electrones de Bloch tienen la siguiente propiedad como una consecuencia de la periodicidad del potencial U.

1.1 Teorema de Bloch.

Los estados propios ψ del Hamiltoniano para un electrón $\mathbf{H} = -\hbar^2/2m\nabla^2 + \mathrm{U}(\mathbf{r})$, en donde $\mathrm{U}(\mathbf{r} + \mathbf{R}) = \mathrm{U}(\mathbf{r})$ para todas las $\mathbf{R}$ en una red de Bravais, toman la forma de una onda plana multiplicados por una función que tiene la misma periodicidad de la red de Bravais:

$$\psi_{\alpha\mathbf{k}}(\mathbf{r}) = e^{i\mathbf{k}\cdot\mathbf{r}} u_{\alpha\mathbf{k}}(\mathbf{r}), \tag{1.8}$$

en donde $\mathbf{k}$ es el vector de onda, y $\alpha \in \mathbf{N}$ (conjunto de los números naturales), además

$$u_{\alpha\mathbf{k}}(\mathbf{r}+\mathbf{R}) = u_{\alpha\mathbf{k}}(\mathbf{r}), \tag{1.9}$$

para todas las $\mathbf{R}$ en la red de Bravais.

De las ecuaciones (1.8) y (1.9) tenemos que

$$\psi_{\alpha\mathbf{k}}(\mathbf{r}+\mathbf{R}) = e^{i\mathbf{k}\cdot\mathbf{R}} \psi_{\alpha\mathbf{k}}(\mathbf{r}), \tag{1.10}$$

para cualquier $\mathbf{R}$ de la red de Bravais.

1.2 La condición a la frontera de Born-Von Karman.

Imponiendo condiciones a la frontera apropiadas sobre las funciones de onda podemos demostrar que el vector de onda **k** deber ser real, y por lo tanto sus valores quedarán restringidos. Consideremos el volumen infinito generado por una celda primitiva de la red de Bravais fundamental, e impongamos entonces las llamadas condiciones periódicas a la frontera, de la siguiente manera,

$$\psi(\mathbf{r} + N_i\mathbf{a}_i) = \psi(\mathbf{r}), \quad i = 1, 2, 3, \tag{1.11}$$

En donde los $\mathbf{a}_i$ son los vectores primitivos del cristal, y los N_i son números enteros del orden $N^{1/3}$, con N el número total de celdas primitivas $N = N_1 \cdot N_2 \cdot N_3$.

Aceptaremos estas condiciones a la frontera suponiendo que $N >> 1$, y que las propiedades del sustrato del sólido no dependerán de la elección de las condiciones a la frontera. Aplicando el teorema de Bloch (1.10) a la condición a la frontera (1.11) llegamos a que

$$\psi_{n\mathbf{k}}(\mathbf{r} + N_i\mathbf{a}_i) = \exp(iN_i\mathbf{k}\cdot\mathbf{a}_i)\,\psi_{n\mathbf{k}}(\mathbf{r}), \qquad i = 1, 2, 3, \tag{1.12}$$

lo cual requiere que

$$\exp(iN_i\mathbf{k}\cdot\mathbf{a}_i) = 1, \qquad i = 1, 2, 3, \tag{1.13}$$

Si **k** tiene la forma $\mathbf{k} = x_1\mathbf{b}_1 + x_2\mathbf{b}_2 + x_3\mathbf{b}_3$, en donde los $\mathbf{b}_i$ son los vectores de la red recíproca, entonces la ecuación (1.13) requiere que

$$\exp(i2\pi \cdot N_i x_i) = 1, \tag{1.14}$$

y consecuentemente debemos tener,

$$x_i = \frac{m_i}{N_i}, \qquad \text{con } m_i \text{ un entero.} \tag{1.15}$$

Entonces la forma general para los vectores de onda de Bloch permitidos es,

$$\mathbf{k} = \sum_{i=1}^{3} \frac{m_i}{N_i}\mathbf{b}_i, \qquad \text{con } m_i \text{ un entero.} \tag{1.16}$$

Ahora bien, el volumen Δk en el espacio-**k** por valor permitido de **k**, es el volumen del paralelogramo con ejes $\frac{\mathbf{b}_i}{N_i}$:

$$\Delta k = \frac{\mathbf{b}_1}{N_1} \cdot \left(\frac{\mathbf{b}_2}{N_2} \times \frac{\mathbf{b}_3}{N_3} \right) = \frac{1}{N} \mathbf{b}_1 \cdot (\mathbf{b}_2 \times \mathbf{b}_3). \tag{1.17}$$

Por lo tanto, *el número de vectores de onda permitidos en una celda primitiva de la red recíproca es igual al número de unidades en el cristal.*

El volumen de la celda primitiva de la red recíproca es $\frac{(2\pi)^3}{\upsilon}$, en donde $\upsilon = \frac{V}{N}$, entonces la ecuación (1.17) puede escribirse como,

$$\Delta \mathrm{k} = \frac{(2\pi)^3}{V} \tag{1.18}$$

1.3 Observaciones generales sobre el teorema de Bloch.

1. El teorema de Bloch define un vector **k**, que es muy importante para resolver el problema de movimiento de un electrón en un potencial periódico. Debemos ver a **k** como un número cuántico que caracteriza a la simetría traslacional de un potencial periódico.

2. El vector de onda **k** que aparece en el teorema de Bloch, puede ser confinado a la primera zona de Brillouin. Esto es, porque cualquier vector de onda **k'**, que no esté en la primera zona de Brillouin puede ser escrito como $\mathbf{k'} = \mathbf{k} + \mathbf{K}$, donde **K** es un vector en la red recíproca y **k** pertenece a la primera zona de Brillouin. Como $\exp(i\mathbf{K}\cdot\mathbf{R}) = 1$ para cualquier vector de la red recíproca, entonces si el teorema de Bloch se cumple para **k'**, también se cumple para **k**, esto se deduce a partir de la ecuación (1.11). Este procedimiento se conoce como proyectar (*mapping*) la banda de energía (en el punto 4 se define banda de energía) en el esquema de zona reducida de Brillouin.

3. El índice n que aparece en el teorema de Bloch es consecuencia de que para una **k** dada tenemos muchas soluciones a la ecuación de Schrödinger. Por lo que n nos proporciona un índice de banda de energía.

4. Aunque el conjunto completo de niveles se describe con **k** restringida a la primera zona de Brillouin, frecuentemente es útil permitir a **k** que tome todos los valores posibles en el espacio **k**, a pesar de que la descripción sea redundante. Como el conjunto de todas las funciones de onda y los niveles de energía para dos valores de **k** que difieren por un vector de la red recíproca deben ser idénticos, podemos asignar el índice n a los niveles de tal manera que para una n dada, los autoestados y autovalores sean funciones periódicas de **k** en la red recíproca:

$$\begin{aligned} \psi_{n,\mathbf{k}+\mathbf{K}}(\mathbf{r}) &= \psi_{n\mathbf{k}}(\mathbf{r}) \\ \varepsilon_{n,\mathbf{k}+\mathbf{K}} &= \varepsilon_{n\mathbf{k}} \end{aligned} \qquad (1.19)$$

   Esto nos lleva a describir a los niveles de energía de un electrón en un potencial periódico en términos de una familia de funciones continuas $\varepsilon_n(\mathbf{k})$, cada una con la periodicidad de la red recíproca. La información contenida en estas funciones se conoce como la estructura de banda del sólido. Para cada n, el conjunto de niveles electrónicos especificados por $\varepsilon_n(\mathbf{k})$ se llama *banda de energía*, porque están separadas por regiones de energía en las que no existen orbitales electrónicos. A estas regiones no permitidas se les denomina bandas prohibidas o bandas de energía prohibidas y son el resultado de la interacción de las ondas de los electrones de conducción con los núcleos del cristal.

1.4 Estados de Superficie

Dado que la superficie es la terminación del sustrato, los átomos en la superficie tienen un menor número de vecinos que los que tienen los átomos del sustrato. Parte de los enlaces químicos que constituyen la estructura del sustrato se rompen para formar a la superficie, por lo que la formación de la superficie tiene un costo energético. Por lo tanto, la estructura electrónica cerca de la superficie es diferente respecto a la del sustrato. Es más, aún en la superficie ideal con sus átomos conservando las posiciones del sustrato al ser truncado, muestran nuevos niveles energéticos debidos al rompimiento de los enlaces químicos. Muchos efectos macroscópicos en las superficies están relacionados con este cambio en la estructura electrónica, por ejemplo, la energía superficial, y las fuerzas de adhesión.

Para ilustrar esto último, vamos a calcular los estados de energía de superficie para un cristal seminfinito. Los átomos con coordenada $z < 0$ forman el sustrato, en $z = 0$ están los átomos de nuestra superficie, y para $z > 0$ está el vacío; además vamos a utilizar el modelo de electrón casi libre. Dentro del plano de la superficie asumamos que tenemos una periodicidad bidimensional. En la dirección perpendicular (z), la simetría traslacional se rompe en la superficie del cristal. Las coordenadas de la superficie del cristal están dadas por el vector $\mathbf{r}_{\|}$. La función de onda ϕ_{ss} (los índices (ss) son una abreviatura de "*surface states*", estados de superficie) más general para un electrón cuyos estados de energía están localizados cerca de la superficie tiene el carácter de una onda plana (o de Bloch) en el plano paralelo a la superficie,

$$\varphi_{ss}(\mathbf{r}_{\|}, z) = \mathrm{u}_{k_{\|}}(\mathbf{r}_{\|}, z)\exp(i\mathbf{k}_{\|}\cdot\mathbf{r}_{\|}), \tag{1.20}$$

en donde $\mathbf{k}_{\|} = (k_x, k_y)$ es un vector de onda paralelo a la superficie. La función de modulación $\mathrm{u}_{k_{\|}}$ tiene la periodicidad de la superficie y se etiqueta con el vector de onda $k_{\|}$. Por simplicidad tomemos como modelo un cristal en una dimensión seminfinito para $z < 0$ de átomos idénticos que represente la periodicidad del sustrato del cristal. El final ($z = 0$) representa la superficie. Utilicemos el modelo del electrón casi libre, y supongamos que el electrón se encuentra en un potencial representado por una función coseno para z negativas,

$$\mathrm{V}(z) = 2\hat{V}\cos(\frac{2\pi z}{a}), \quad \text{para } z < 0, \tag{1.21}$$

a lo largo de la superficie, ver figura 1.1.

La superficie ($z = 0$) es modelada por un escalón de potencial V_0 que es una simplificación de la realidad.

Resolvamos la ecuación de Schrödinger

$$\left[-\frac{\hbar^2}{2m}\frac{d^2}{dz^2}+\mathrm{V}(z)\right]\psi(z)=E\,\psi(z), \tag{1.22}$$

usando el potencial definido por la ecuación (1.21) para $V(z)$.

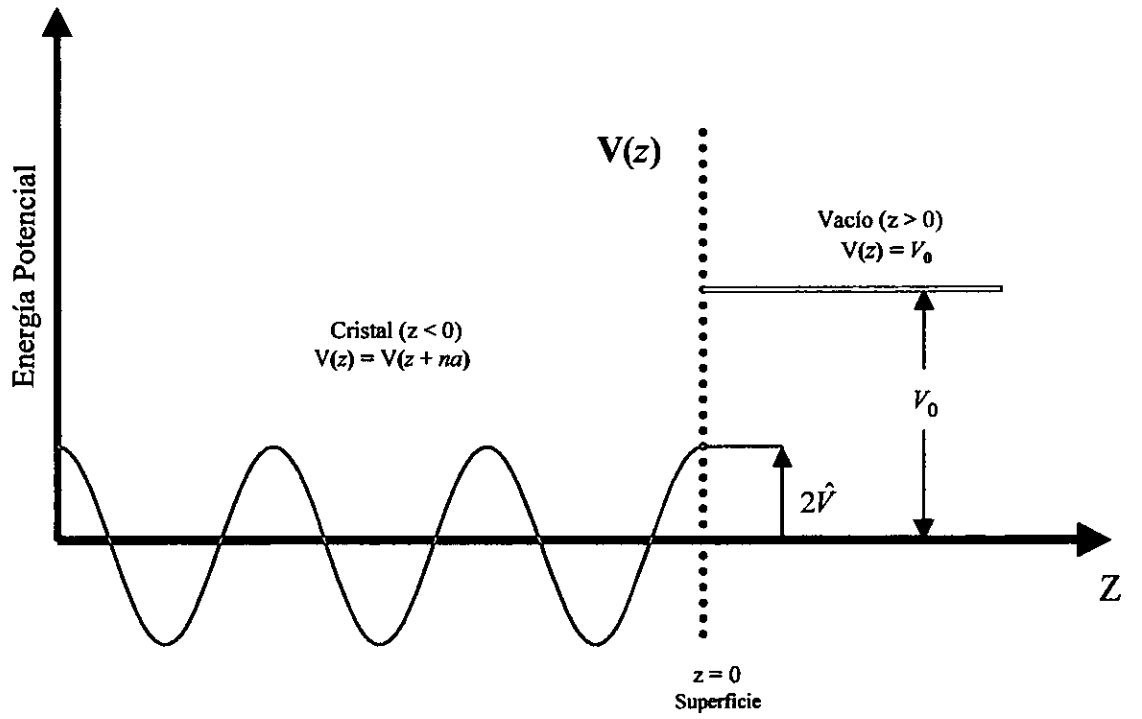


Figura 1.1. Representación de la interfaz entre el vacío y el cristal. En la superficie (z = 0) tenemos la presencia de una energía potencial.

Fijémonos en las regiones muy profundas en el sustrato ($z << 0$), lejos de la superficie (z = 0). En esta región podemos considerar al sustrato con tamaño infinito y despreciar los efectos debidos a la superficie. Lejos de las fronteras de la primera zona de Brillouin $\mathbf{k}_\perp = \pm \pi/a$ ($\mathbf{k}_\perp$ es un vector de onda normal a la superficie) los estados electrónicos tienen el carácter de una onda plana y sus bandas de energías tienen la forma de la parábola del electrón libre (Fig. 1.2). Cerca de la frontera de la zona, la separación de las bandas se presenta debido a que la función de onda debe ser tomada dentro de la aproximación de menor orden, como una superposición de dos ondas planas. En la frontera de la zona un electrón es dispersado del estado $k_\perp = \pi/a$ en el estado $k_\perp = -\pi/a$. Para valores de $k_\perp$ cercanos a π/a = G/2 (G es un vector de la red recíproca) la función de onda con esta aproximación es:

$$\psi(z) = A\exp(ik_\perp) + B\exp(i[k_\perp - \frac{2\pi}{a}]z)\,. \qquad (1.23)$$

Usando el potencial dado por la ecuación (1.21) y sustituyendo la ecuación anterior en la ecuación de Schrödinger, llegamos a la siguiente ecuación matricial :

$$\begin{bmatrix} \frac{\hbar^2}{2m}k_\perp^2 - E(k_\perp) & \hat{V} \\ \hat{V} & \frac{\hbar^2}{2m}\left(k_\perp - \frac{2\pi}{a}\right)^2 - E(k_\perp) \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = 0\,. \qquad (1.24)$$

La solución no trivial se encuentra igualando a cero su determinante.

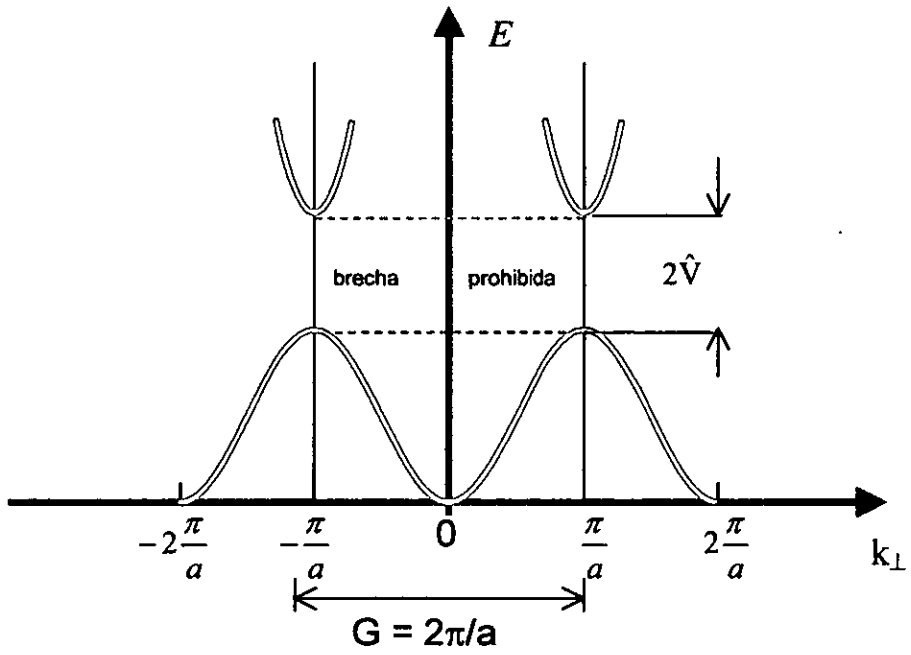


Figura 1.2. Bandas de energía $E(k_\perp)$ para un electrón del sustrato.

Nos interesan soluciones cercanas a la frontera de la zona de Brillouin, es decir, cerca de $k_\perp = \pm G/2 = \pm\pi/a$. Hagamos $k_\perp = \kappa + \pi/a$, donde los valores pequeños del número real κ corresponden a los $k_\perp$ que nos interesan; los valores propios de la energía que se obtienen al resolver la ecuación (1.24) están dados por la siguiente expresión,

$$E=\frac{\hbar^2}{2m}\left[\frac{\pi}{a}+\kappa\right]^2+\left|\hat{V}\right|\left[\frac{-\hbar^2\pi\kappa}{ma\left|\hat{V}\right|}\pm\sqrt{\left(\frac{\hbar^2\pi\kappa}{ma\left|\hat{V}\right|}\right)^2+1}\right]. \qquad (1.25)$$

La función de onda ψ_i para regiones dentro del cristal y lejanas de la superficie ($z << 0$) se obtienen utilizando el resultado de la ecuación (1.25), y resolviendo el determinante de la ecuación (1.24) para A, B, y finalmente introduciendo el resultado en la expresión para la función de onda (1.23):

$$\psi_i = C\exp(i\kappa z)\left\{\exp(i\pi\frac{z}{a})+\frac{\hat{V}}{\hat{V}}\left[\frac{\hbar^2\pi\kappa}{ma\hat{V}}\pm\sqrt{\left(\frac{\hbar^2\pi\kappa}{ma\hat{V}}\right)^2+1}\right]\exp(-i\pi\frac{z}{a})\right\}. \qquad (1.26)$$

Aquí C es la constante de normalización. Para regiones muy internas en el cristal ($z << 0$), los niveles de energía electrónicos toman la forma que se presenta en la figura 1.2.
La parábola del "electrón libre" se divide cerca de la frontera de la zona $k_\perp = \pm\pi/a$, y se presentan las bandas de energía permitidas y prohibidas. La dependencia de $E(k_\perp = \kappa + \pi/a)$ es parabólica cerca de $\pm\pi/a$ como se puede ver en la ecuación (1.25), y la anchura de la banda prohibida es $2\hat{V}$ de acuerdo con esta misma relación.

Veamos ahora las soluciones de la ecuación de Schrödinger (1.22) cerca de la superficie del sólido ($z = 0$). Dichas soluciones deben estar compuestas por una parte que es compatible con la energía potencial constante en el vacío $E_{vacío} = V_0$ ($z > 0$) y de otra parte que resuelve a la ecuación de Schrödinger (1.22) en el sustrato del cristal con un potencial dado por la ecuación (1.21). Las dos soluciones junto con sus derivadas $\frac{\partial\psi}{\partial z}$ para los casos $z > 0$ y $z < 0$ tienen que ser igualadas en la superficie ($z = 0$) . Cualquier solución ψ_o en un potencial constante V_0 en el lado del vacío ($z > 0$) con $E < V_0$, debe tener un decaimiento exponencial de la siguiente forma,

$$\psi_0 = D\exp\left[-\sqrt{\frac{2m}{\hbar^2}(V_0-E)}z\right], \quad E < V_0. \qquad (1.27)$$

Como la ecuación (1.27) para la función de onda con $z > 0$, no tiene parte compleja como la ecuación (1.26) para la función de onda con $z < 0$ dada por $\exp(i\kappa z)$, las ψ_i del tipo (1.26) dentro del cristal pueden ser igualadas sí y sólo sí en la superficie se comportan como una onda estacionaria, entonces

$$\psi_0(z=0) = \alpha\psi_i(z=0,\kappa) + \beta\psi_i(z=0,-\kappa), \tag{1.28}$$

con ψ_0 y ψ_i dadas por (1.27) y (1.26) respectivamente. La ecuación (1.28) y la relación correspondiente para las derivadas pueden cumplirse para cualquier valor de la energía E dentro de la banda permitida. Entonces algunas soluciones en la superficie del cristal son funciones de Bloch estacionarias dentro del cristal que decaen exponencialmente en el vacío (Fig. 1.3a). Los niveles de energía correspondientes no se ven muy diferentes respecto a los estados del sustrato del cristal.

Además se pueden tener soluciones adicionales si permitimos vectores de onda complejos. Permitamos entonces que κ sea imaginario

$$\kappa = -iq\,, \text{ con } q \text{ un número real,} \tag{1.29}$$

y definamos a

$$\gamma = i\sin(2\delta) = -i\frac{\hbar^2\pi q}{ma\left|\hat{V}\right|}. \tag{1.30}$$

Entonces la función de onda electrónica para κ imaginario dentro del cristal a partir de la ecuación (1.26) y utilizando las relaciones (1.29) y (1.30) tenemos que

$$\psi_i(z\le 0) = F\exp(qz)\left\{\exp\left[i\left(\frac{\pi}{a}z\pm\delta\right)\right]\mp\exp\left[-i\left(\frac{\pi}{a}z\pm\delta\right)\right]\right\}\exp(\mp i\delta), \tag{1.31}$$

donde F es una constante de normalización. Esto es esencialmente una onda estacionaria con una amplitud que tiene un decaimiento exponencial (fig. 1.3b). Los valores propios de la energía los obtenemos a partir de las relaciones (1.25) y (1.28), y están dados por la siguiente expresión,

$$E = \frac{\hbar^2}{2m}\left[\left(\frac{\pi}{a}\right)^2 - q^2\right] \pm \left|\hat{V}\right| \sqrt{1-\left(\frac{\hbar^2\pi q}{ma\left|\hat{V}\right|}\right)^2}\,. \tag{1.32}$$

Los valores de E son reales y ψ no diverge para valores muy negativos de z siempre y cuando q tome valores $0 < q < q_{\max} = \dfrac{ma\left|\hat{V}\right|}{\hbar^2\pi}$. Para estos valores de q; todas las energías caen dentro de la banda prohibida de la estructura electrónica de bandas del sustrato del cristal. Para encontrar las constantes de normalización (1.31) para el problema de la superficie, tenemos que igualar

dentro y fuera del sustrato las funciones de onda (1.31) con la función de onda (1.27). Esto es, la condición anterior requiere que las funciones de onda junto con sus derivadas tomen los mismos valores en la frontera del cristal,

$$\psi_0(z=0)=\psi_i(z=0) \quad \text{y} \quad \frac{d\psi_0}{dz}\Big|_{z=0}=\frac{d\psi_i}{dz}\Big|_{z=0}. \tag{1.33}$$

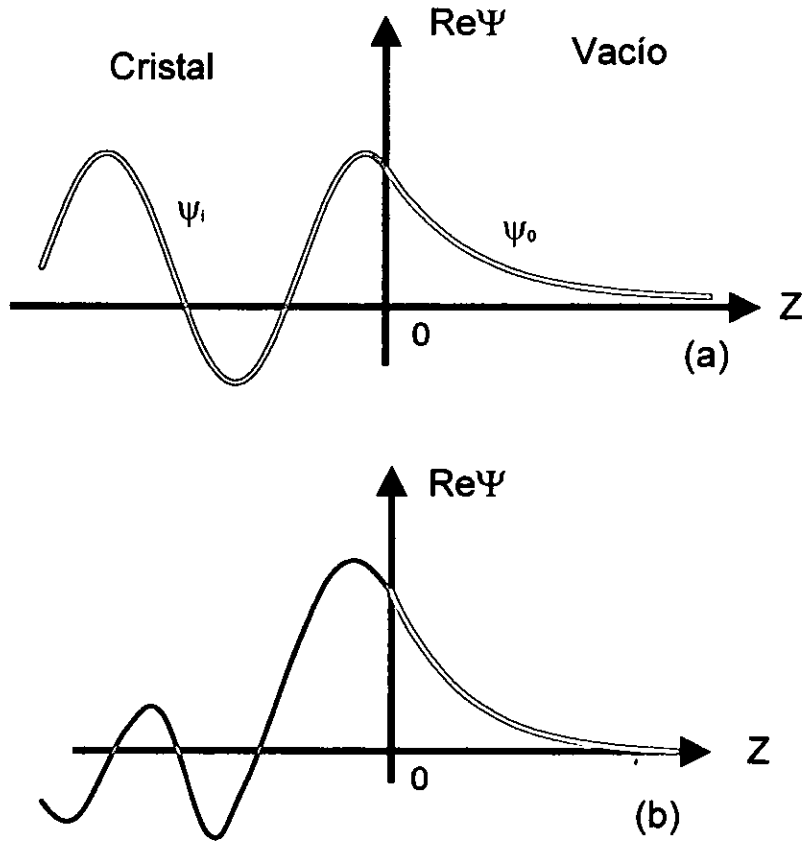


Figura 1.3. Parte real de la función de onda para un electrón, Re{ψ}, cuando (a) tenemos una función de Bloch estacionaria (ψ_i) que se iguala con una función exponencial (ψ_o) que decae en el vacío. Y cuando (b) tenemos una función de onda para un estado localizado en la superficie ($z = 0$).

La función de onda resultante se muestra gráficamente en la figura 1.3b. Su amplitud se anula para los valores $\pm z$ lejanas a la superficie. Si $|\hat{V}|$ es pequeño, es decir, de algunos eV, los electrones en estos estados energéticos están localizados a pocos Åmstrongs de la superficie. Otra consecuencia importante que se obtiene a partir de las condiciones a la frontera, ecuación (1.33), es la restricción en los valores permitidos para E. Dentro de la banda prohibida de energía del sustrato del cristal, aparece un nivel de energía E que está fijo y cumple con las condiciones a la frontera dadas por la relación (1.33).

El presente cálculo para la superficie seminfinita muestra entonces, que hay un estado electrónico de superficie que está localizado en alguna parte dentro de la banda prohibida de energía de los estado de sustrato del cristal.

En resumen, para la superficie seminfinita existen estados localizados en la superficie que se encuentran en alguna región dentro de la banda prohibida de los estados electrónicos del sustrato del cristal.

1.5 Relajación y Reconstrucción.

Ahora vamos a estudiar la estructura atómica de una superficie. Debido a la ausencia de átomos vecinos en la parte superior de una superficie, las fuerzas interatómicas en las capas superiores se modifican considerablemente. Las condiciones de equilibrio para los átomos en la superficie son diferentes con respecto al sustrato; se espera que las posiciones atómicas se alteren, y en consecuencia la estructura de la superficie no concuerda con la estructura del sustrato. Por lo que una superficie no es sólo truncar el sustrato del cristal. La distorsión en la estructura de la superficie va a depender del material, va a ser diferente en metales y en semiconductores. El germanio es un semiconductor que cristaliza con enlaces tetraédricos, por lo que los enlaces direccionales están presentes. Ver figura 1.4.

La contribución principal a la energía superficial de un semiconductor es la energía necesaria para romper los enlaces atómicos; las superficies con índices de Miller (111) de cristales cúbicos se ven favorecidas energéticamente, porque tienen la menor densidad de enlaces comparada con otras caras del cristal. Entonces el plano de corte natural para el cristal del germanio es la superficie (111).

Si las capas exteriores de átomos de la superficie se mueven únicamente en una dirección normal a la superficie decimos que tenemos una **relajación.** En este caso, la red bidimensional es periódica en una dirección paralela a la superficie, es decir, tiene la misma periodicidad que tendría la proyección de la celda unitaria del sustrato en la superficie del cristal.

Se pueden tener cambios más dramáticos en la estructura de la superficie si los átomos se mueven en una dirección paralela a la superficie, como resultado cambian la periodicidad paralela a la superficie. La celda unitaria tiene diferentes dimensiones respecto a la celda unitaria del sustrato proyectada sobre la superficie. A este tipo de reacomodo se le conoce como **reconstrucción.** En la figura 1.5 se muestra un modelo de reconstrucción del germanio que se ha encontrado para la superficie Ge(111). Los átomos de germanio en la superficie se acomodan de una manera tal que los enlaces colgantes de los átomos vecinos forman cadenas en zigzag. Los átomos de las capas adyacentes, de las cuatro primeras capas atómicas, cambian sus posiciones, y también participan en la reconstrucción. Cálculos realizados usando la técnica de dinámica molecular cuántica muestran que este reacomodo cumple con los requerimientos de minimizar la energía total respecto a otras configuraciones atómicas. En una superficie cristalina, tenemos simetría traslacional en los planos paralelos a ella.

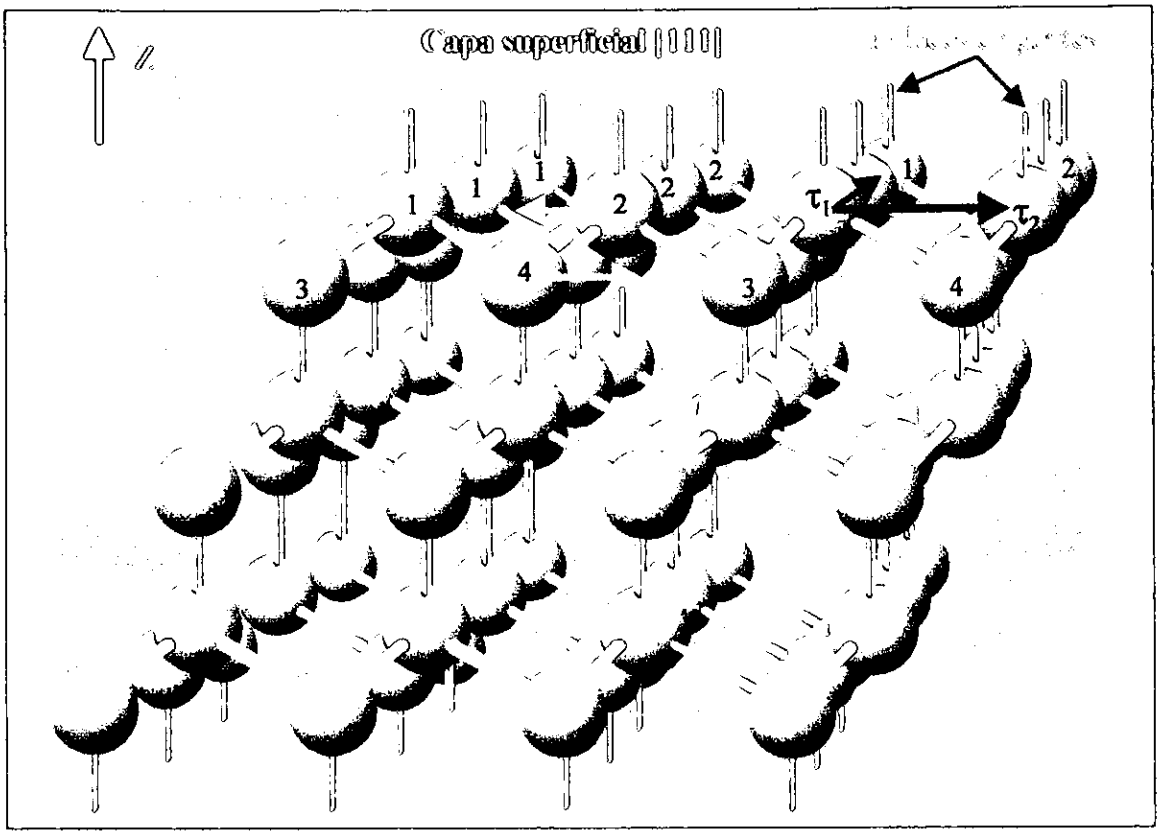


Figura 1.4. Enlaces colgantes de la superficie ideal (111) de la estructura cúbica del germanio con enlaces covalentes [3]. En la parte superior derecha de la presente figura se muestra a los vectores base τ_1 y τ_2 de la proyección de la celda unitaria del sustrato en la superficie. Al formarse la superficie reconstruida del Ge(111)-2×1 los átomos con etiquetas 2 y 4 cambian sus posiciones como lo indican las flechas amarilla y verde respectivamente, dando lugar a una recomposición atómica como se muestra en la figura 1.5.

Sean τ_1 y τ_2 los vectores base de la proyección de la celda unitaria del sustrato en la superficie. Estudios de difracción de electrones de baja energía (LEED por su siglas en inglés) revelan que tipo de simetría traslacional tiene la superficie [2]. Por ejemplo, puede informar que las traslaciones primitivas se generan con $2\tau_1$ y τ_2 , entonces se dice que la superficie tiene una reconstrucción 2×1. Similarmente una simetría traslacional generada por $m\tau_1$ y $n\tau_2$ es nombrada una reconstrucción $m \times n$.

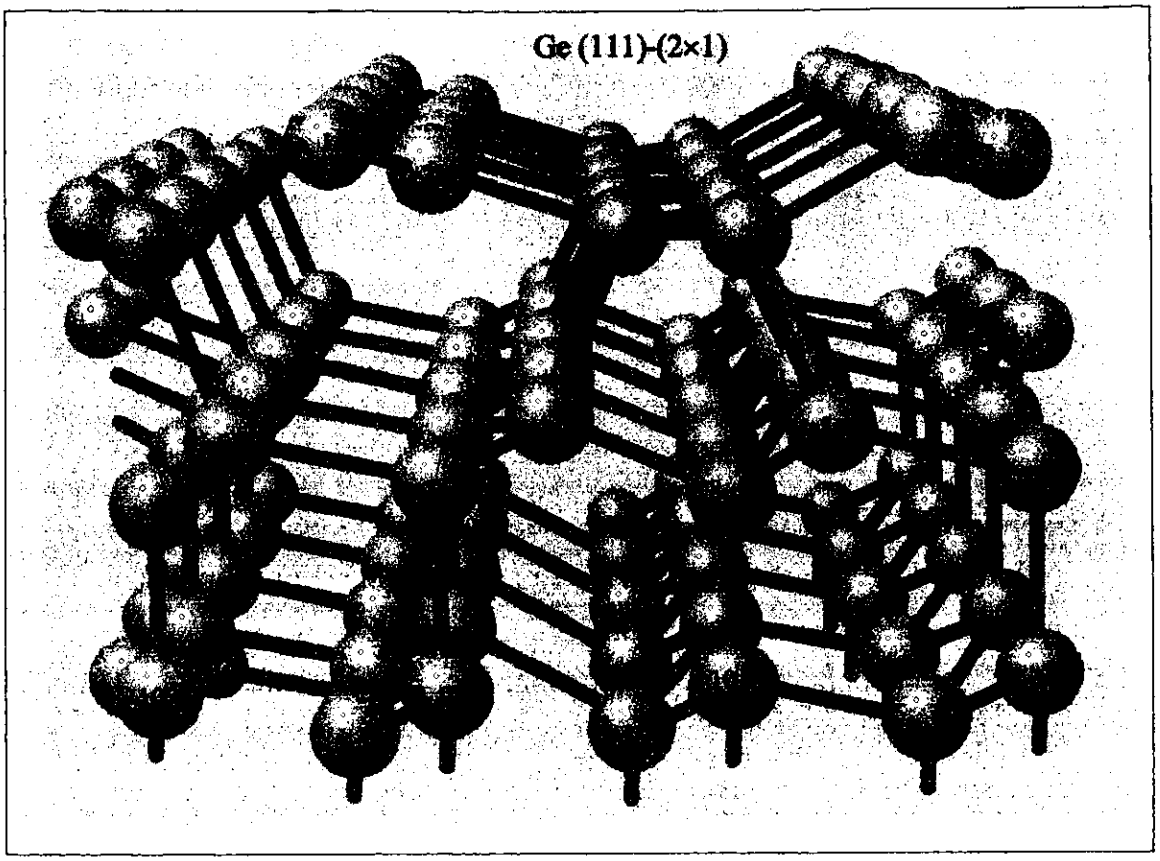


Figura 1.5. Vista esquemática de la superficie reconstruida del Ge(111)-2×1. Los átomos 1 y 2, difieren en su desplazamiento paralelo a la normal de la superficie por 0.5 a_B (radio de Bohr). En la superficie ideal estos dos átomos están a la misma altura. La longitud del enlace entre átomos 1 y 2 en la superficie es de $4.37\,a_B$, que es significativamente más pequeña que la longitud del enlace entre los átomos del sustrato que es de $4.62\,a_B$ [4].

Nuestro propósito en este trabajo es encontrar la estructura electrónica de la superficie reconstruida del Ge(111)-2x1. Dada las características de la superficie es necesario un modelo teórico menos simplificado que el que se discutió para encontrar los estados electrónicos de la superficie monoatómica seminfinita. Para esto, vamos a utilizar el método de Enlace Fuerte, que describiremos en el siguiente capítulo.

CAPITULO II

EL método de Enlace Fuerte

En este capítulo vamos a explicar en que consiste el método de Enlace Fuerte. Usaremos este método para calcular las propiedades electrónicas tanto del sustrato como la superficie de un cristal ideal de germanio Ge(111)-1×1. Como veremos más adelante es este capítulo, se desarrollaron una serie de programas en lenguaje C++ que implementan el método de Enlace Fuerte. Para verificar que nuestros programas trabajan correctamente, vamos a comparar los resultados arrojados por los programas con los datos publicados por otros autores.

El método de Enlace Fuerte es uno de los más usados para resolver los problemas con potenciales periódicos presentes en la teoría de los movimientos electrónicos en sólidos, y se le conoce también con los nombres de LCAO (*Linear Combination of Atomic Orbitals*), o *Tight Binding*[5]. Este método originalmente fue propuesto por Bloch, y consiste en llevar a cabo combinaciones lineales de los orbitales atómicos localizados en los diferentes átomos que forman al cristal, los coeficientes que parecen en la combinación lineal, son los valores de la onda plana $\exp(i\mathbf{k}\cdot\mathbf{R})$ en las diferentes posiciones $\mathbf{R}$ de los átomos en el cristal. Del método LCAO se obtienen soluciones que muestran correctamente las propiedades de simetría de las bandas de energía, y además facilita el cálculo de las bandas de energía en un punto arbitrario de la zona de Brillouin.

Al resolver la ecuación de Schrödinger con el método LCAO, los resultados que se obtienen están dados por una matriz Hamiltoniana, cuyas entradas se resuelven integrando el operador Hamiltoniano con las funciones de onda inherentes al método. Nosotros vamos a despreciar muchos términos de la matriz Hamiltoniana que presentan dificultad para hacer los cálculos matemáticos exactos, por lo que vamos a retener sólo aquellos términos que sean relevantes para llevar a cabo los cálculos. Además, en lugar de evaluar de manera analítica o numérica las integrales de la matriz Hamiltoniana que aparecen durante el proceso de cálculo, vamos a usar constantes que sustituyen a estas integrales (ver la ecuación (2.6)), y cuyos valores se van a elegir a partir de otros métodos más exactos, como el método celular, o el método de ondas planas ortogonalizadas, que arrojan resultados más precisos en los puntos de alta simetría de la zona de Brillouin[8]. En el caso de que las constantes no hayan sido calculadas por métodos analíticos más exactos, en su lugar, se utilizaran valores obtenidos a partir de mediciones experimentales[7].

Por otro lado, se simplifican mucho los cálculos si se utilizan orbitales con una dirección preestablecida, como los orbitales atómicos. Por ejemplo, en el germanio, la disposición de los enlaces atómicos forma un tetraedro, como se

puede ver en la figura 2.1. Por lo que, si se utilizara una base de ondas planas, se requeriría un número mucho mayor de funciones de onda para lograr la orientación que tienen los enlaces atómicos, lo que a su vez haría mucho más grande la matriz Hamiltoniana, debido a que nuestra base de funciones aumenta, y como consecuencia, el cálculo numérico sería más complejo.

2.1 Sumas de Bloch y el Método de Enlace Fuerte

A las funciones de onda espaciales que representen el estado cuántico de un electrón en un átomo, las llamaremos orbitales atómicos[5]. Consideremos un orbital atómico con números cuánticos simbolizados por el subíndice α, y situado en un átomo en la posición $\mathbf{R}_i$, denotado por $\chi_\alpha(\mathbf{r}-\mathbf{R}_i)$. Construimos las sumas de Bloch para cada tipo de orbital atómico α, y utilizando el Teorema de Bloch (ecuación (1.9)) llegamos a que,

$$\varphi_\alpha(\mathbf{k},\mathbf{r}) = \frac{1}{\sqrt{N}}\sum_i \exp(i\mathbf{k}\cdot\mathbf{R}_i)\chi_\alpha(\mathbf{r}-\mathbf{R}_i), \tag{2.1}$$

en donde la suma se hará sobre todos los átomos en las posiciones equivalentes dentro de todas las N celdas unitarias del cristal.

Podemos obtener una solución aproximada del problema del potencial periódico de la siguiente manera: tomemos un conjunto finito de orbitales atómicos para cada uno de los átomos de la celda unitaria, los electrones que están dentro de la coraza del átomo no se tomarán en cuenta, porque su potencial de interacción no contribuye en gran medida a las propiedades del material que nos interesa, como es la conductividad o la respuesta óptica. Supondremos también que las corazas, compuestas por los iones y los electrones ligados a ellas, permanecen fijos. Empecemos con los estados de energía más bajos hasta llegar con los niveles más altos ocupados en el cristal. Es decir, α va a tomar todos los números cuánticos posibles de los orbitales atómicos ocupados por electrones de valencia. Con estos orbitales atómicos construyamos las sumas de Bloch. Para un valor dado de $\mathbf{k}$, podemos construir funciones de onda que son combinaciones lineales de todas las sumas de Bloch, llamadas funciones de Bloch.

Si empezamos con los orbitales atómicos $\chi_\alpha(\mathbf{r}-\mathbf{R}_i)$ ortogonales en cada átomo de la celda unitaria, y construimos las sumas de Bloch con estos orbitales, vamos a encontrar que las funciones de Bloch no son mutuamente ortogonales. La razón de esto, es que las funciones de Bloch φ_α, que son combinaciones lineales de los orbitales de los átomos, no son ortoganales entre sí. Podemos eliminar esta dificultad gracias al teorema de Lödwin[8], que nos permite construir nuevos orbitales atómicos, que son combinaciones lineales de los originales, de tal forma que sean ortogonales entre sí mismos, y conserven la simetría de los antiguos orbitales. Asumiremos que esto ya se hizo, y los vamos a utilizar para calcular la función de onda del electrón que se construye como la combinación lineal de sumas de Bloch de la manera siguiente. Cada función de Bloch corresponde a cada uno de los orbitales atómicos necesarios para describir los electrones de valencia del sistema. Por ejemplo, para el silicio (Si)

se tiene cuatro electrones de valencia, dos de ellos en el orbital $3s$ y los otros dos en el orbital $3p$. Estos nos definen una base mínima de orbitales, llamada sp^3, con las que se construye las funciones de Bloch,

$$\Psi(\mathbf{k},\mathbf{r}) = \sum_{\alpha} U_{\alpha}\varphi_{\alpha}(\mathbf{k},\mathbf{R}), \tag{2.2}$$

o explícitamente,

$$\Psi(\mathbf{k},\mathbf{r}) = \sum_{\alpha,j} U_{\alpha}\exp(i\mathbf{k}\cdot\mathbf{R}_j)\chi_{\alpha}(\mathbf{r}-\mathbf{R}_j), \tag{2.3}$$

donde ahora $\chi_{\alpha}(\mathbf{r}-\mathbf{R}_j)$ son los orbitales de Lödwin, y las funciones de Bloch φ_{α} son ortogonales[8]. El método de Enlace Fuerte lo aplicaremos al cristal de germanio infinito, y como veremos en la siguiente sección, es un cristal formado por dos redes cúbicas de caras centradas interpenetradas, por lo que tenemos que tomar condiciones a la frontera periódicas, y suponemos que el número de celdas unitarias que forman el cristal es N. Entonces las sumas de Bloch se tienen que hacer sobre las celdas unitarias que forman al cristal. La función de Bloch va a estar formada por un orbital de Lödwin tipo atómico χ_{α} por celda unitaria. Si tenemos varios átomos por celda unitaria, vamos a usar diferentes sumas de Bloch para cada orbital de cada uno de los átomos en la celda unitaria. Es decir, α además de representar los números cuánticos del orbital también tiene un índice que etiqueta al número de átomo en la celda unitaria.

La ecuación de Schrödinger independiente del tiempo es

$$\mathbf{H}\Psi_n(\mathbf{k},\mathbf{r}) = E_n(\mathbf{k})\Psi_n(\mathbf{k},\mathbf{r}), \tag{2.4}$$

en donde $\mathbf{H}$ es el Hamiltoniano del cristal en la aproximación de un electrón. Sustituyendo la ecuación 2.3 en la ecuación de Schrödinger, nos resulta un sistema de ecuaciones algebraicas lineales para los coeficientes U_{α}, de la forma

$$\sum_{\beta,i} U_{\beta}\exp(i\mathbf{k}\cdot\mathbf{R}_i)\mathbf{H}\chi_{\beta}(\mathbf{r}-\mathbf{R}_i) = E_n(\mathbf{k})\sum_{\beta,i} U_{\beta}\exp(i\mathbf{k}\cdot\mathbf{R}_i)\mathbf{H}\chi_{\beta}(\mathbf{r}-\mathbf{R}_i). \tag{2.5}$$

Multiplicando a la ecuación anterior por el complejo conjugado de la función de Bloch φ_{α}^{*} e integrando sobre el volumen que ocupa el cristal, obtenemos los elementos de matriz

$$\mathbf{H}_{\alpha\beta} = \langle\varphi_{\alpha}\,|\,\mathbf{H}\,|\,\varphi_{\beta}\rangle = \frac{1}{N}\sum_{i,j}\exp(-i\mathbf{k}\cdot(\mathbf{R}_j-\mathbf{R}_i))\int_{V}\chi_{\alpha}^{*}(\mathbf{r}-\mathbf{R}_j)\mathbf{H}\chi_{\beta}(\mathbf{r}-\mathbf{R}_i)d\mathbf{r}, \tag{2.6}$$

en donde se ha tomando en cuenta la condición de ortogonalidad de los orbitales atómicos χ_α, $\int_V \chi_\alpha^*(\mathbf{r}-\mathbf{R}_i)\chi_\beta(\mathbf{r}-\mathbf{R}_i)d\mathbf{r} = \delta_{\alpha,\beta}$, y que no hay traslape entre orbitales atómicos situados en las posiciones $\mathbf{R}_i$ y $\mathbf{R}_j$ con $\mathbf{R}_i \neq \mathbf{R}_j$, es decir,

$$\int \chi_\alpha^*(\mathbf{r}-\mathbf{R}_i)\,\chi_\beta(\mathbf{r}-\mathbf{R}_j)d\mathbf{r} = 0, \quad \text{si } \mathbf{R}_i \neq \mathbf{R}_j. \tag{2.7}$$

La integral que aparece en la ecuación (2.6) y que está dada por la siguiente expresión,

$$\mathbf{M}_{\alpha\beta}(\mathbf{R}_i,\mathbf{R}_j) = \int_V \chi_\alpha^*(\mathbf{r}-\mathbf{R}_j)\mathbf{H}\chi_\beta(\mathbf{r}-\mathbf{R}_i)dr \tag{2.8}$$

nos define una matriz $\mathbf{M}_{\alpha\beta}$ que llamaremos *matriz de interacción*. En nuestro ejemplo del Si, los índices α, β que identifican a los elementos de la matriz de interacción toman los valores s, p_x, p_y y p_z.

Para obtener una solución aproximada, tomaremos únicamente cierto conjunto de orbitales atómicos para cada uno de los átomos de la celda unitaria. En las sumas de la ecuación (2.6) de la matriz Hamiltoniana, se está tomando en cuenta las interacciones entre los orbitales atómicos de todos los átomos que forman al cristal. Se puede hacer una aproximación de la siguiente manera: las interacciones se pueden restringir a interacciones entre átomos que guardan una distancia de primeros vecinos, segundos o incluso hasta terceros vecinos. Para elegir que aproximación es la mejor, ésta se escoge de tal manera que reproduzca la mayoría de las tendencias de las bandas de energía que estén de acuerdo con resultados experimentales o teóricos más precisos. En el caso de la interacción a primeros vecinos usando una base $sp^3 = \{s, p_x, p_y, p_z\}$ en semiconductores cúbicos, los resultados difieren a la realidad observada, aunque en este caso los cálculos son sencillos pues se tienen que realizar únicamente nueve integrales independientes. Sin embargo, considerando interacciones hasta segundos vecinos, los resultados que se obtienen son muy buenos, pero el número de variables crece a 23, lo que implica un enorme trabajo numérico. Como el modelo sp^3 a primeros vecinos no reproduce la brecha indirecta que se presenta para la mayoría de los semiconductores cúbicos, y por consiguiente tampoco para el Ge, vamos aumentar nuestra base sp^3 a una base extendida $sp^3s^* = \{s, p_x, p_y, p_z, s^*\}$, y también consideraremos interacciones sólo a primeros vecinos. Aquí s^* es un orbital atómico excitado con simetría esférica. El estado excitado s^* repele a la banda de energía más baja de los átomos vecinos. En particular, empuja hacía abajo el mínimo relativo e indirecto de la banda de energía de conducción. Tomaremos la interacción entre dos orbitales s^* igual a cero; y como una aproximación adicional, también supondremos que los orbitales s^* y p tienen interacción sólo si los átomos están adyacentes (son primeros vecinos). Con estas aproximaciones se obtienen resultados satisfactorios utilizando un número pequeño de parámetros

independientes, lo cual simplifica los cálculos numéricos del método de Enlace Fuerte.

A partir de los elementos de la matriz de la expresión (2.8), usando las aproximaciones mencionadas anteriormente, haciendo el cambio de variable $\mathbf{r}-\mathbf{R}_j \rightarrow \mathbf{r}'$ (j=1,...,N), y renombrando $\mathbf{r}'$ como $\mathbf{r}$, obtenemos que

$$\mathbf{M}_{\alpha\beta}(\mathbf{d}_j) = \langle \chi_\alpha(\mathbf{r}) \mid \mathbf{H} \mid \chi_\beta(\mathbf{r}-\mathbf{d}_j) \rangle , \tag{2.9a}$$

en donde $\mathbf{d}_j = \mathbf{R}_j - \mathbf{R}_i$, por lo que la ecuación (2.6) toma la forma,

$$\mathbf{H}_{\alpha\beta} = \sum_j \exp(i\mathbf{k}\cdot\mathbf{d}_j)\mathbf{M}_{\alpha\beta}(\mathbf{d}_j) \tag{2.9b}$$

El factor N^{-1} se cancela debido a que tenemos N términos idénticos de la matriz definida por la relación (2.6).

Por otra parte, a la matriz de interacción $\mathbf{M}_{\alpha\beta}$ definida por la ecuación (2.9a) la podemos separar en dos términos matriciales que están dados por,

$$\mathbf{M}_{\alpha\beta} = \underbrace{\mathbf{M}_{\alpha\beta}(\mathbf{k},\mathbf{R}_i = \mathbf{R}_j)}_{\mathbf{M}_1} + \underbrace{\mathbf{M}_{\alpha\beta}(\mathbf{k},\mathbf{R}_i \neq \mathbf{R}_j)}_{\mathbf{M}_2}, \tag{2.9c}$$

en donde $\mathbf{M}_1$ es la matriz de interacción en el mismo átomo, y $\mathbf{M}_2$ es la matriz de interacción en átomos distintos.

La matriz de interacción $\mathbf{M}_{\alpha\beta}$ en nuestro problema tendrá una dimensión de 5×5, porque estamos tomando una base con cinco orbitales atómicos $\{s, p_x, p_y, p_z, s^*\}$. En el capítulo siguiente veremos que el germanio tiene dos átomos por celda unitaria, por lo que la matriz Hamiltoniana tendrá una dimensión de 10×10. El diagrama de bloques de la matriz Hamiltoniana es el siguiente,

$$\mathbf{H} = \begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix} \tag{2.9d}$$

La matrices M_1 y M_4 son las matrices de interacción definidas anteriormente, y se calculan a partir de la ecuación (2.9a). Como los dos átomos de la celda unitaria son de la misma especie, germanio en nuestro caso, las dos matrices son iguales. Estas matrices son diagonales y sus elementos son los siguientes,

$$M_1 = \begin{bmatrix} E_{ss} & 0 & 0 & 0 & 0 \\ 0 & E_{px} & 0 & 0 & 0 \\ 0 & 0 & E_{py} & 0 & 0 \\ 0 & 0 & 0 & E_{pz} & 0 \\ 0 & 0 & 0 & 0 & E_{s^*s^*} \end{bmatrix} \tag{2.9e}$$

Los elementos de la diagonal se calculan como ya mencionamos, a partir de la ecuación (2.9a), haciendo $\mathbf{d}_j = 0$ ($\mathbf{R}_i = \mathbf{R}_j$), y están dados por,

$$E_{ss} = M_{ss} = <\chi_s^* | \mathbf{H} | \chi_s>, \tag{2.9f}$$

$$E_{px} = M_{px} = <\chi_{px}^* | \mathbf{H} | \chi_{px}>, \tag{2.9g}$$

$$E_{px} = M_{py} = <\chi_{py}^* | \mathbf{H} | \chi_{py}>, \tag{2.9h}$$

$$E_{pz} = M_{pz} = <\chi_{pz}^* | \mathbf{H} | \chi_{pz}>, \tag{2.9i}$$

$$E_{s^*} = M_{s^*s^*} = <\chi_{s^*}^* | \mathbf{H} | \chi_{s^*}> \tag{2.9j}$$

Las matrices $\mathbf{M}_2$ y $\mathbf{M}_3$ son la matrices de interacción del átomo, que en la figura 2.1 está coloreado en azul (rojo) con sus primeros vecinos, que en la misma figura están coloreados en rojo (azul), y también son iguales. La expresión analítica de los elementos de estas dos matrices se obtienen también a partir de la ecuación (2.9a) tomando $\mathbf{d}_j \neq 0$ ($\mathbf{R}_i \neq \mathbf{R}_j$), y puede consultarse en la referencia [5, páginas 76 y 77, tabla 3-1]. En nuestro caso, como veremos más adelante, se desarrolló una rutina en lenguaje C, que realiza los cálculos de manera automatizada.

Por lo tanto, la solución al problema, para encontrar los coeficientes U_α y la relación de dispersión de la energía en función del vector de onda $\mathbf{k}$, está dada por la siguiente ecuación secular

$$Det \mid \mathbf{H}_{\alpha\beta}(\mathbf{k},\mathbf{r}) - E(\mathbf{k})\delta_{\alpha\beta} \mid = 0, \tag{2.10}$$

en donde $\mathbf{H}_{\alpha\beta}(\mathbf{k}, \mathbf{r})$ está dado por la ecuación (2.6).

En el formalismo de Enlace Fuerte semiempírico, en lugar de realizar las integrales de la ecuación (2.9a) los elementos de la matriz Hamiltoniana entre orbitales atómicos se consideran como parámetros que se obtienen ajustando las bandas de energía en ciertos puntos $\mathbf{k}$ de alta simetría a otros resultados experimentales o teóricos también en los puntos de alta simetría [7].

En general, si n es el número de orbitales por cada uno de los átomos de la celda unitaria, y m es el número por celda unitaria, entonces se tendrán a lo más $n \times m$ raíces diferentes de la ecuación secular (2.10) para cada valor diferente de $\mathbf{k}$. Las raíces de la ecuación secular que tengan el valor más pequeño formarán la primera banda de energía, aquellas que tengan la segunda energía más baja, formarán la segunda banda, y así sucesivamente. En ciertos puntos de alta simetría, es posible que varias bandas de energía puedan estar degeneradas, es decir, tienen el mismo valor.

2.2 Cálculo de la estructura electrónica del cristal de germanio.

En el presente trabajo, para calcular la estructura electrónica de la superficie del germanio en su cara (111) se desarrollaron un conjunto de programas en lenguaje C++ [6]. En los programas *matrix.cpp, vector.cpp* y *eigen.cpp* se implementaron los algoritmos que se encargan de manipular a las matrices (suma, resta, transpuesta, multiplicación, norma, cálculo de valores y vectores propios, etc.) que se generan durante el proceso del cálculo de la estructura electrónica con el método de Enlace Fuerte. Para probar que nuestros programas trabajan correctamente vamos a calcular la estructura electrónica del cristal de germanio utilizando una base sp^3, y una base sp^3s^*. También con este cálculo se demostrará la ventaja de la base sp^3s^* sobre la sp^3.

La red espacial del germanio consiste en dos redes cúbicas de caras centradas interpenetradas, lo que da lugar a una estructura cristalina tipo diamante. La base primitiva tiene dos átomos idénticos en las posiciones $(0,0,0)$ y $(\frac{a}{4},\frac{a}{4},\frac{a}{4})$ asociados con cada punto de la red con constante a. Como se puede ver en la figura 2.1, cada átomo tiene cuatro vecinos más próximos y doce vecinos siguientes a los próximos.

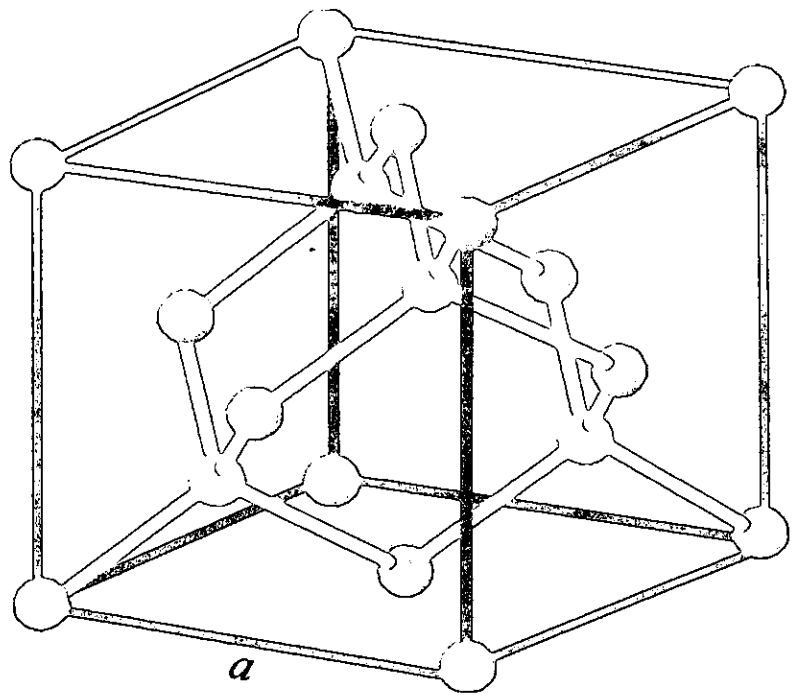


Figura 2.1. Estructura cristalina del germanio, mostrando la disposición de los enlaces tetraédricos, y los primeros y segundos vecinos. Como puede observarse cada átomo de germanio en rojo (azul) tiene cuatro primeros vecinos en azul (rojo), y segundos vecinos en azul (rojo).

La constante de red del germanio es a = 5.658 Å[1]. Mientras que la distancia a primeros vecinos en una red tipo diamante está dada por $|\mathbf{d}| = \frac{\sqrt{3}}{4}a$.

Los elementos de la matriz Hamiltoniana para átomos adyacentes de acuerdo a la ecuación (2.9) tienen la forma $\langle\alpha|\mathbf{H}|\beta\rangle$. Para los orbitales entre átomos adyacentes vamos a construir un vector **d** que parte desde el núcleo del átomo cuyo orbital es $|\alpha\rangle$ (átomo izquierdo) al núcleo del átomo con orbital $|\beta\rangle$ (átomo derecho). Vamos a utilizar un sistema de coordenadas esféricas con el eje Z paralelo al vector **d**, y con el origen en el centro de un átomo. La forma angular de los orbitales es $Y_l^m(\theta,\varphi)$ para el orbital izquierdo y $Y_q^p(\theta',\varphi)$ para el orbital derecho. Los factores angulares dependen de φ y son de la forma $e^{i(m-p)\varphi}$ (recordemos que $\langle\alpha|$ es el complejo conjugado de $|\alpha\rangle$). La integración sobre φ da cero a menos que $m = p$. Entonces todos los elementos de la matriz $\langle\alpha|\mathbf{H}|\beta\rangle$ se anulan a menos que $m = p$. A estos elementos de matriz se le etiqueta con los símbolos σ, π, o δ, en analogía con s, p, d, para m = 0, 1, ó 2, respectivamente.

En la figura 2.2 se muestran las posibles interacciones entre átomos adyacentes. En la parte inferior de la figura 2.2 se muestra una interacción entre un orbital s en el origen y un orbital p_z en un átomo vecino, cuyo enlace forma un ángulo θ con el vector **d** que está paralelo al eje Z. Podemos expresar al orbital p_z como una combinación lineal de orbitales p en un nuevo sistema de coordenadas. Nuestro nuevo sistema de coordenadas denotado por (') tendrá su eje Z' paralelo al vector **d**, y el eje X' será perpendicular a Z' y al eje de rotación. El orbital p_z puede entonces ser expresado como una combinación lineal de los orbitales p_z' y p_x', es decir, $p_z = p_z'\cos\theta + p_x'\sin\theta$, como se muestra en la parte inferior de la figura 2.2. El orbital s en el origen es el mismo en el sistema de coordenadas anterior y nuevo. Por lo tanto, el elemento de la matriz Hamiltoniana entre el orbital s en el origen y el orbital p_z' es $V_{sp\sigma}$, mientras que la interacción del orbital s con el orbital p_x' es igual a cero (no hay traslape entre los orbitales). En realidad lo que estamos haciendo es descomponer al orbital p_z en dos componentes paralela y perpendicular al eje de enlace **d**.

Similarmente, cuando consideremos la interacción entre dos orbitales p entre átomos vecinos vamos a descomponer a los orbitales p en tres componentes, que serán paralelas a los ejes coordenados X, Y, y Z (paralelo al vector **d**). Cada componente de los orbitales p se calcula a partir de los cosenos directores del vector **d**. Entonces un orbital p en términos de sus componentes respecto a los ejes coordenados se puede expresar como la siguiente combinación lineal,

$$p = \frac{d_x}{|\mathbf{d}|}p_x + \frac{d_y}{|\mathbf{d}|}p_y + \frac{d_z}{|\mathbf{d}|}p_z = al\cdot p_x + am\cdot p_y + an\cdot p_z \text{ y } |\mathbf{d}| = \sqrt{d_x^2 + d_y^2 + d_z^2},$$

en donde suponemos que el vector **d** tiene coordenadas (d_x, d_y, d_z), y *al*, *am* y *an* son los cosenos directores del vector **d**. Al lector que le interese conocer la expresión analítica de las integrales de la ecuación (2.9a) para una base de orbitales ampliada puede consultar el artículo de J. C. Slater y G. F. Koster[8].

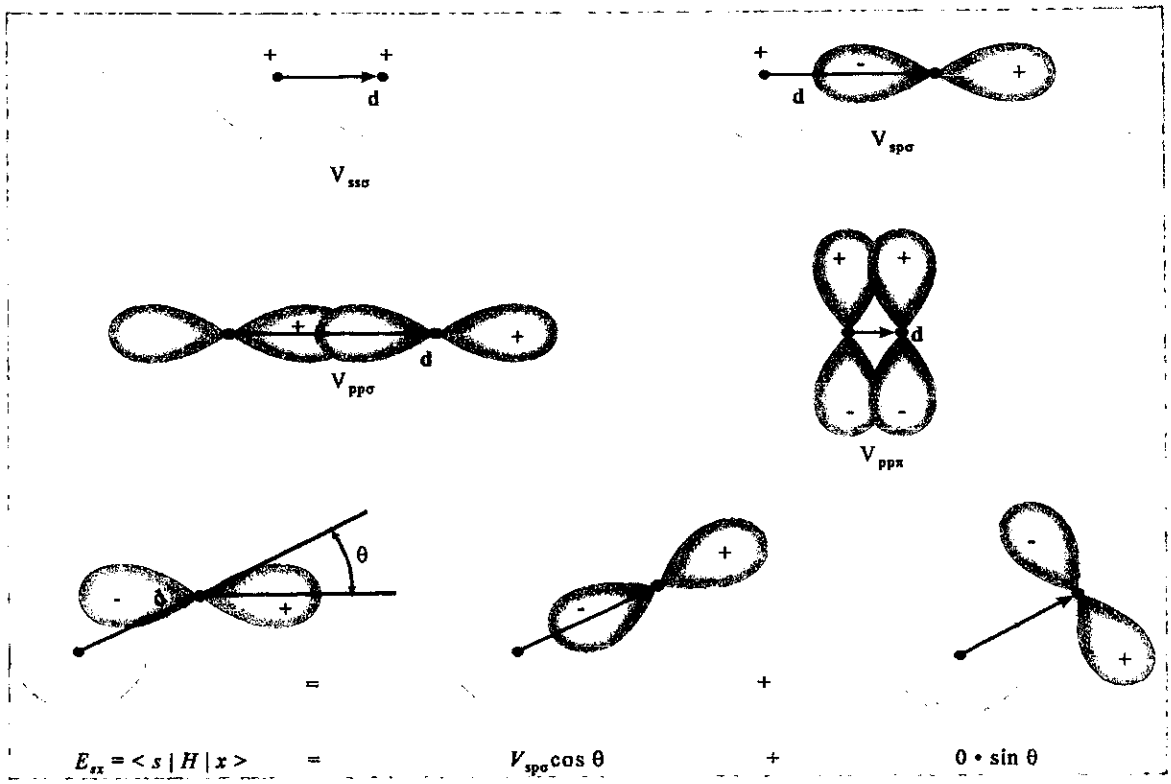


Figura 2.2. En esta figura podemos observar las cuatro posibles interacciones atómicas entre átomos adyacentes (primeros vecinos) con la base sp^3. Cuando los orbitales p no están orientados como se muestra en los diagramas superiores, ellos pueden ser descompuestos geométricamente como vectores para poder evaluar las componentes de la matriz, como se muestra en el diagrama inferior. Las componentes de los vectores están dadas por las proyecciones (cosenos directores) del vector sobre los ejes coordenados.

A continuación vamos a describir el programa que realiza los cálculos de la estructura electrónica del germanio. Los listados de todos los programas que se utilizaron en el presente trabajo se muestran en el apéndice A.

En el programa *fcc2.cpp* se calcula la estructura electrónica del germanio utilizando una base sp^3 y la interacción a primeros vecinos. Sin pérdida de generalidad, podemos adimensionalizar las longitudes con la constante de red del cristal. El algoritmo es el siguiente:

i) La función *GetData*() lee las energías E_{ss}, E_{px}, E_{py}, E_{pz}, definidas por las ecuaciones (2.9f a la 2.9i), V_{ss}, V_{sp}, V_{ps}, $V_{pp\pi}$, y $V_{pp\sigma}$. En la figura 2.2 se muestra esquemáticamente como se calculan las interacciones entre cualquier par de átomos a una distancia de primeros vecinos. En la tabla

2.1 se muestran los valores que se utilizaron para generar la matriz Hamiltoniana.

ii) La función *GetR*() calcula los primeros vecinos de los átomos base de la celda unitaria del cristal de germanio. La función toma a los vectores base de la red **x**, **y**, **z**, y se forma la combinación lineal $\mathbf{v} = l\mathbf{x} + m\mathbf{y} + n\mathbf{z}$. Si el vector **v** está a una distancia igual a la distancia a primeros vecinos entonces **v** es un primer vecino.

Parámetro	Valor (eV), para la base sp^3s	Valor (eV), para la base sp^3s^*
(α=β=s) E_{ss} (R=0)	-5.8800	-5.8800
(α=s, β=p_x, p_y, p_z) E_{px}, E_{py}, E_{pz} (R=0)	2.1000	1.6100
(α=β=s^*) E_{s^*} (R=0)	0	6.3900
(α=β=s) V_{ss} (R=d)	-1.7900	-1.6950
(α=s, β=p_x, p_y, p_z) V_{sp} (R=d)	2.3600	2.3664
(α=p_x, p_y, p_z, β=s) V_{ps} (R=d)	-2.3600	-2.3664
(α=p_x, p_y, p_z, β=s) $V_{pp\pi}$ (R=d)	-1.0500	-0.8225
(α=p_x, p_y, p_z, β=s) $V_{pp\sigma}$ (R=d)	4.0700	2.8525
(α=s^*, β=p_x, p_y, p_z) V_{s^*p} (R=d)	0	2.260

Tabla 2.1. Parámetros empíricos que se utilizan para generar los elementos de la matriz Hamiltoniana usando la base sp^3s [5] y sp^3s^*[7], respectivamente. La primera columna representa a las integrales $\langle \chi_\alpha(\mathbf{r}) | \mathbf{H} | \chi(\mathbf{r}-\mathbf{d}_j) \rangle$ definidas en la ecuación (2.9a) para diferentes valores de α y β.

iii) La función *GetMatInt*(), calcula la matriz de interacción utilizando las integrales de energía que se muestran en la tabla 2.1. En nuestra función, se utiliza un arreglo con cinco componentes a los que se les asigna los siguientes valores: $V[0] = V_{ss\sigma}$, $V[1] = V_{sp\sigma}$, $V[2] = -V[1]$, $V[3] = V_{pp\sigma}$, y $V[4] = V_{pp\pi}$. Los cosenos directores *al*, *am*, y *an* son calculados en la función *GetMatHam*() y son los cosenos directores del vector **d** respecto a los ejes X, Y, y Z respectivamente.

iv) La función *GetMatHam*(), realiza las sumas de las funciones de Bloch y genera la matriz Hamiltoniana. Recordemos que tenemos dos átomos por celda unitaria y estamos tomando una base ya sea de cuatro (sp^3) o cinco (sp^3s^*) orbitales por átomo, por lo que la matriz Hamiltoniana tendrá 8×8 ó 10×10 elementos, respectivamente.

v) La función *join*() genera a la matriz Hamiltoniana, juntando su parte real con su parte imaginaria en una sola matriz, para poder aplicarle la función *GetEigen*() que calcula los valores y vectores propios de la matriz Hamiltoniana.

vi) La función *GetEnerB*(), es la función principal del programa y se encarga de calcular la estructura electrónica del cristal al recorrer la primera zona de Brillouin en las direcciones que unen a los puntos de alta simetría (L, Γ, X, K, Γ). En la figura 2.3 se muestra la zona de Brillouin y los puntos de alta simetría.

vii) La función *GetEigen*(), calcula los valores propios y vectores propios de la matriz Hamiltoniana (generada por la función *join*()). Al terminar de ejecutarse esta función, los valores propios los deja en un arreglo, y los vectores propios los deja en los renglones de la matriz que recibió como parámetro.

viii) La función *WriteEner*() escribe en el archivo *GeEB.dat* las energías, que en realidad son los valores propios de la matriz Hamiltoniana, correspondientes a los puntos de alta simetría.

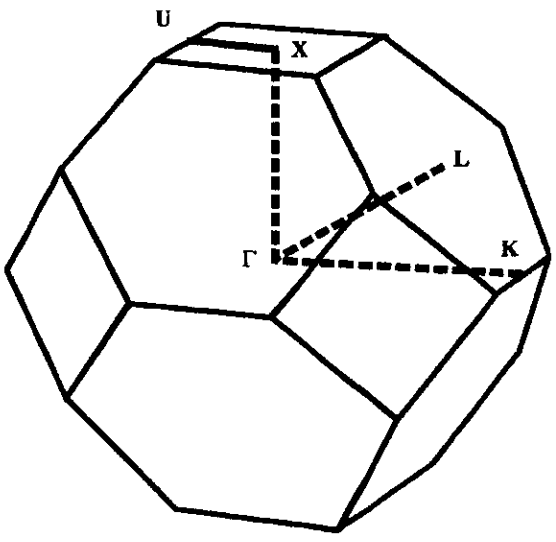


Figura 2.3. Zona de Brillouin para la red cristalina del germanio mostrando los puntos de alta simetría. Las bandas mostradas en la figura 2.4 han sido graficadas a lo largo de **L** a Γ, de Γ a **X**, de **X** a **U**, y de **K** a Γ.

El programa *fcc2s.cpp* calcula la estructura electrónica del cristal de germanio utilizando una base sp^3s^*. La estructura de este programa es similar a la del programa *fcc2.cpp*. Sólo que en este caso, la matriz de interacción toma en cuenta al orbital excitado s^*. Al correr los programas se obtienen los siguientes resultados:

1. La base sp^3 reproduce correctamente los estados de energía en la banda de valencia, sin embargo, en la banda de conducción, los estados energéticos no reproducen lo medido experimentalmente, como se verá más adelante.

2. La base sp^3s^* preserva los estados de energía de la banda de valencia, reproduce exitosamente los estados energéticos que se encuentran en la banda de conducción, y también muestra correctamente la brecha indirecta del germanio.

3. El estado excitado s^* repele a los estados de la banda de conducción, y como consecuencia el mínimo relativo en la banda de conducción disminuye.

En la figura 2.4, se muestra gráficamente la estructura electrónica del germanio, los datos graficados son los resultados obtenidos al correr los programas correspondientes.

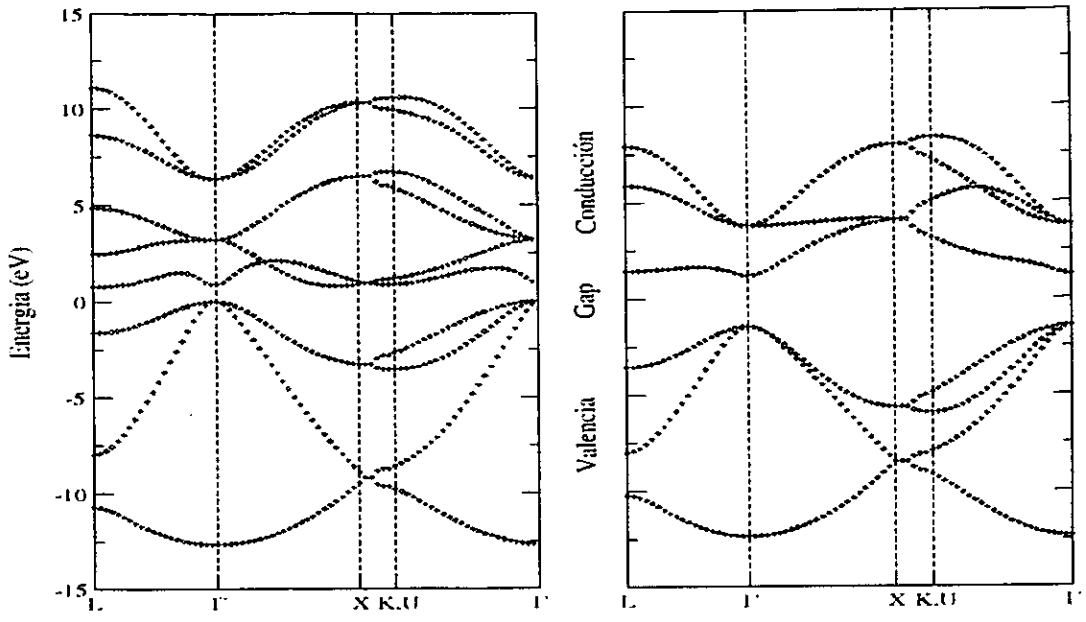


Figura 2.4. En el cuadro de la izquierda se muestran las bandas de energía calculadas utilizando la base sp^3s^* para el germanio. En el cuadro que está a la derecha se muestra la estructura electrónica del germanio utilizando una base sp^3. En ésta figura se puede ver que la brecha disminuye y la banda de valencia no cambia considerablemente al usar la base extendida sp^3s^*.

Los datos obtenidos usando la base sp^3s^* coinciden con los resultados publicados por P. Volg y Harold P. Hjalmarson[7]. Y los datos obtenidos utilizando la base sp^3 coinciden con los publicados por Walter Harrinson[5]. Ver figura 2.5. Aunque en ambas figuras 2.4 y 2.5 no se aprecia a simple vista el mínimo indirecto, al consultar los valores que se obtuvieron al correr el programa *fcc2s* se puede ver que el valor de la energía en el punto de alta simetría L es de 0.7649 eV menor al valor 0.9 eV en el punto de alta simetría Γ. Por lo que en el punto de alta simetría L tenemos un mínimo indirecto.

En la parte izquierda de la figura 2.5 [8] se presentan en color azul los valores calculados utilizando una base sp^3s^*, la línea punteada en color rojo representa a los valores experimentales que se obtuvieron utilizando la técnica de Fotoemisión Inversa[16]. Por lo que podemos concluir que la base sp^3s^* reproduce de manera más aproximada a los valores experimentales que caen dentro de la banda de valencia y conducción, obteniendo así la brecha indirecta entre los puntos L y Γ del Ge. Por otro lado, la gráfica de la derecha muestra que la base sp^3 sólo nos puede dar una descripción cualitativa de las bandas. En conclusión la base extendida sp^3s^* es mejor que la base sp^3.

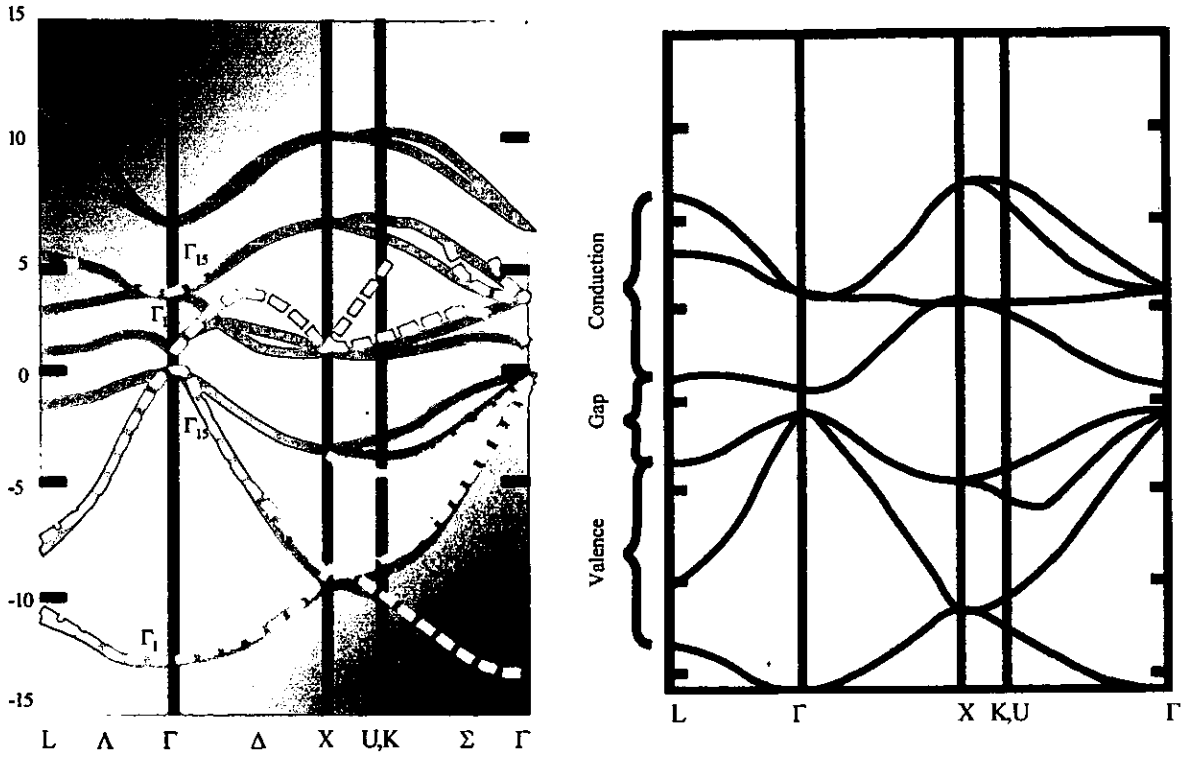


Figura 2.5. En la parte derecha se presentan las bandas de energía del Ge publicadas por Walter Harrinson[5] usando una base sp^3. Mientras que en la parte izquierda se presentan las bandas de energía del Ge publicadas por Volg y sus colaboradores usando una base sp^3s^*[7].

2.3 Formulación del método de Enlace Fuerte para superficies

El paso siguiente es adaptar esta formulación de Enlace Fuerte a las superficies cristalinas. Usemos la descripción del cristal seminfinito en términos de una red bidimensional (2D) paralela al plano de la superficie y con celdas unitarias que se extienden infinitamente en dirección perpendicular a la superficie. Denotemos los vectores de la red 2D por $\mathbf{r}$ y los de la base de los átomos de cada celda unitaria por τ_{pm}, en donde p indica el número de átomo de la celda unitaria y m el número de plano. El vector de posición $\mathbf{R}_{pm} = \mathbf{r} + \tau_{pm}$ corresponde al p-ésimo átomo del plano m de la celda unitaria localizada en $\mathbf{r}$. Escribamos el vector de posición del p-ésimo átomo del plano m en sus componentes paralela y perpendicular a la superficie,

$$\mathbf{R}_{pm} = \mathbf{r} + \boldsymbol{\tau}^{p}_{m_\perp} + \boldsymbol{\tau}^{p}_{m_\parallel}. \tag{2.11}$$

Los estados electrónicos estacionarios del cristal con superficie se pueden describir como funciones de Bloch $\varphi_{\mathbf{q}}(\mathbf{x})$ en donde $\mathbf{q}$ son vectores de onda de 2D en la primera ZB (zona de Brillouin) de la superficie. Escribamos a estas funciones de Bloch en 2D $\varphi^{pm}_{\alpha\mathbf{q}}(\mathbf{x})$ como,

$$\varphi^{pm}_{\alpha\mathbf{q}}(\mathbf{x}) = \frac{1}{\sqrt{N_\parallel}} \sum_{\mathbf{r}} \exp[i\mathbf{q}\cdot(\mathbf{r} + \boldsymbol{\tau}^{p}_{m_\parallel})] \cdot \chi_\alpha(\mathbf{x} - \mathbf{r} - \boldsymbol{\tau}^{p}_{m_\perp} - \boldsymbol{\tau}^{p}_{m_\parallel}), \tag{2.12}$$

en donde $N_\parallel$ es el número de celdas unitarias de superficie, α toma los valores dados por los números cuánticos $\{s, p_x, p_y, p_z, s^*\}$, y la suma se hace sobre los vectores unitarios de la celda bidimensional paralela al plano de la superficie. Estas funciones de Bloch son conocidas como *orbitales de plano*. Con este conjunto de ecuaciones se explota la simetría bidimensional del sistema. Entonces la función de onda del electrón en función de esta base de funciones de Bloch se obtiene como la combinación lineal de las funciones de Bloch para cada uno de los orbitales $\varphi^{pm}_{\alpha\mathbf{q}}(\mathbf{x})$,

$$\chi_{\mathbf{q}}(\mathbf{x}) = \sum_{\alpha} \sum_{p,m} d^{pm}_{\alpha} \varphi^{pm}_{\alpha\mathbf{q}}(\mathbf{x}), \tag{2.13}$$

con d^{pm}_{α}, los coeficientes de la combinación lineal. Sustituyendo esta función de onda en la ecuación de Schrödinger con el Hamiltoniano de un electrón encontramos la ecuación de valores propios,

$$\sum_{\beta,p'm'} \{\mathbf{H}_{\alpha\beta}^{pmp'm'}(\mathbf{q}) - E\,\delta_{\alpha\beta}\delta_{pp'}\delta_{mm'}\} d_{\beta}^{p'm'}(\mathbf{q}) = 0, \tag{2.14}$$

en donde los elementos de la matriz Hamiltoniana son,

$$\mathbf{H}_{\alpha\beta}^{pmp'm'}(\mathbf{q}) = \frac{1}{N_{\|}} \sum_{\mathbf{r},\mathbf{r}'} \exp[-i\mathbf{q}\cdot(\mathbf{r}+\boldsymbol{\tau}_{m_{\|}}^{p} - \mathbf{r}' - \boldsymbol{\tau}_{m'_{\|}}^{p'})]\mathbf{H}_{\alpha\beta}(\mathbf{r}+\boldsymbol{\tau}_{m_{\|}}^{p}, \mathbf{r}'+\boldsymbol{\tau}_{m'_{\|}}^{p'}), \tag{2.15a}$$

con $\mathbf{H}_{\alpha\beta}$ igual a

$$\mathbf{H}_{\alpha\beta}(\mathbf{R}_i, \mathbf{R}_j) = \int \chi_{\alpha}^{*}(\mathbf{r}-\mathbf{R}_i)\mathbf{H}\chi_{\beta}(\mathbf{r}-\mathbf{R}_j)d^3r, \tag{2.15b}$$

y usando los parámetros correspondientes a la base sp^3s^*. Nótese que las integrales que aparecen en la ecuación (2.15b) son exactamente iguales a las integrales que aparecen en la ecuación (2.8), por lo que la relación (2.15b) nos define la matriz de interacción para la superficie del cristal. En consecuencia, no es necesario calcular más parámetros para conocer la matriz Hamiltoniana de nuestra superficie, con los parámetros del sustrato es suficiente. Lo único que cambia es la fase de los orbitales.

Esto nos lleva en principio a resolver una matriz de orden infinito, sin embargo es posible aproximar la interacción sólo a primeros vecinos, es decir, $\mathbf{H}_{\alpha\beta}^{mm'}(\mathbf{q})$ sólo es diferente de cero en el plano cuando $m = m'$ o $m = m \pm 1$, esto es, cuando son primeros vecinos. Como la celda unitaria se extiende infinitamente en la dirección perpendicular a la superficie, debemos simular nuestro cristal seminfinito mediante un sistema con un número finito de planos conocido como placa, lámina o *slab*. Estos sistemas deben de reproducir los efectos debidos a la superficie así como los del sustrato. Para lograr esto, por lo regular las placas tienen entre seis y treinta planos atómicos, dependiendo de la superficie que se quiera simular y la capacidad de cómputo disponible. Hay dos tipos diferentes de placas, aquellas con dos superficies separadas por un número suficiente de planos atómicos con los átomos en sus posiciones de sustrato, el número de planos atómicos se escoge de manera que no exista interacción entre las dos superficies. El segundo tipo de placa tiene una sola superficie y algunos planos que simulan el sustrato, en donde el último de ellos se fija o satura de manera que simule el cristal seminfinito.

En este trabajo se utilizará el primer tipo de placa. En la siguiente sección vamos a ilustrar como se implementa un cálculo dentro del modelo de Enlace Fuerte en superficies. Para esto vamos a presentar el problema más simple posible, el de la superficie de Ge(111)-1×1 sin reconstruir.

2.4 Cálculo de la estructura electrónica de la superficie Ge(111)-1x1.

La superficie del germanio (111)-1×1 es modelada utilizando una lámina de 24 capas de átomos de germanio. El grosor de la lámina es suficiente adecuado para desacoplar los estados de superficie en la parte superior e inferior de la lámina. Tomemos el eje X a lo largo de la dirección cristalográfica $[\bar{1}\,\bar{1}\,2]$, el eje Y a lo largo de $[1\,\bar{1}\,0]$ y el eje Z en la dirección perpendicular a la superficie. En la figura 2.6 se muestra la vista superior de la superficie junto con la celda unitaria que contiene un sólo átomo.

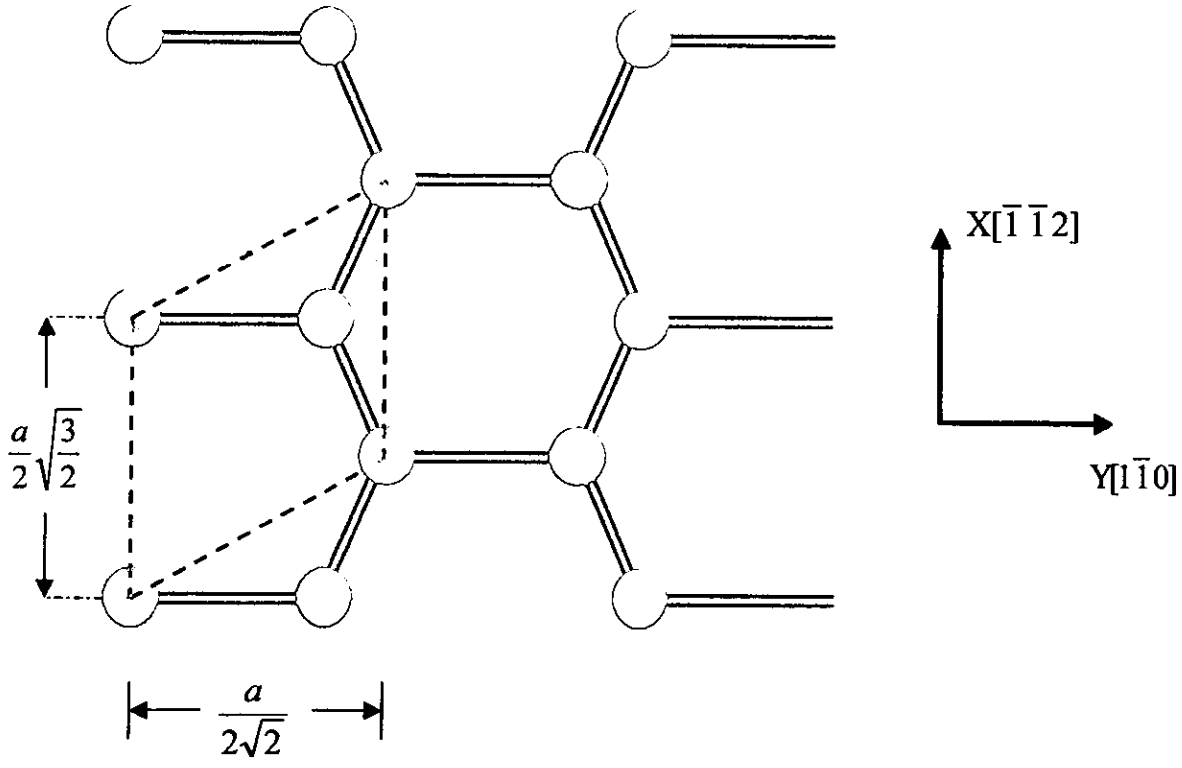


Figura 2.6. Vista superior de la superficie de Ge(111)-1x1. La celda unitaria que se muestra en color rojo, contiene un átomo de germanio por capa atómica. Los círculos verdes representan a los átomos que están en la primera capa τ_p=1, 6; mientras que los círculos en azul representan a los átomos en la segunda y tercera capa τ_p =2, 3; y finalmente los círculos amarillos representan a los átomos en la cuarta y quinta capas τ_p=4, 5;.

El programa que calcula los estados de superficie se llama *surf1x1* y su algoritmo es el siguiente:

i) La función *GetData*() lee los parámetros correspondientes a las energías. Y como vimos en esta sección, únicamente se necesitan los parámetros que se utilizaron para el sustrato. Por lo tanto, se utilizan los valores dados en la tabla 2.1.

ii) Vamos a utilizar la aproximación a primeros vecinos. La función *GetR()* calcula las posiciones de los átomos en la lámina y los almacena en los arreglos de vectores **R**, y **vref**. El primer elemento del arreglo **vref** contiene la posición del primer átomo en la capa 1, y los tres primeros elementos del arreglo **R**, contienen las posiciones de los primeros vecinos. El segundo elemento de **vref** contiene la posición del primer átomo que está en la segunda capa, y los siguientes tres elementos del arreglo **R**, contienen las posiciones de los primeros vecinos de este último átomo, y así sucesivamente.

iii) La función que calcula las posiciones de los átomos en las 24 capas de la lámina se llama *GetAtomPos*(). En la figura 2.7 sólo se muestran las primeras seis capas de la superficie del Ge(111)-1×1.

iv) Ya que tenemos las posiciones de los átomos en la lámina, el siguiente paso es recorrer los puntos de alta simetría en la zona irreducible de Brillouin en dos dimensiones. En la figura 2.8 se muestra la primera zona de Brillouin y su parte irreducible. El programa *surf1x1* recorre los puntos de alta simetría de la zona irreducible, de Γ a M, de M a K y de K a Γ.

v) Las funciones GetMatHam se encarga de formar la matriz Hamiltoniana. Como estamos utilizando para modelar la superficie 24 capas con un átomo por celda unitaria, y una base $\{s, p_x, p_y, p_z, s^*\}$ de 5 orbitales por átomo, la matriz a diagonalizar tendrá una dimensión de 120×120 elementos. La función GetEigen() calcula los valores y vectores propios del Hamiltoniano, y la función WriteEner() escribe en un archivo los resultados.

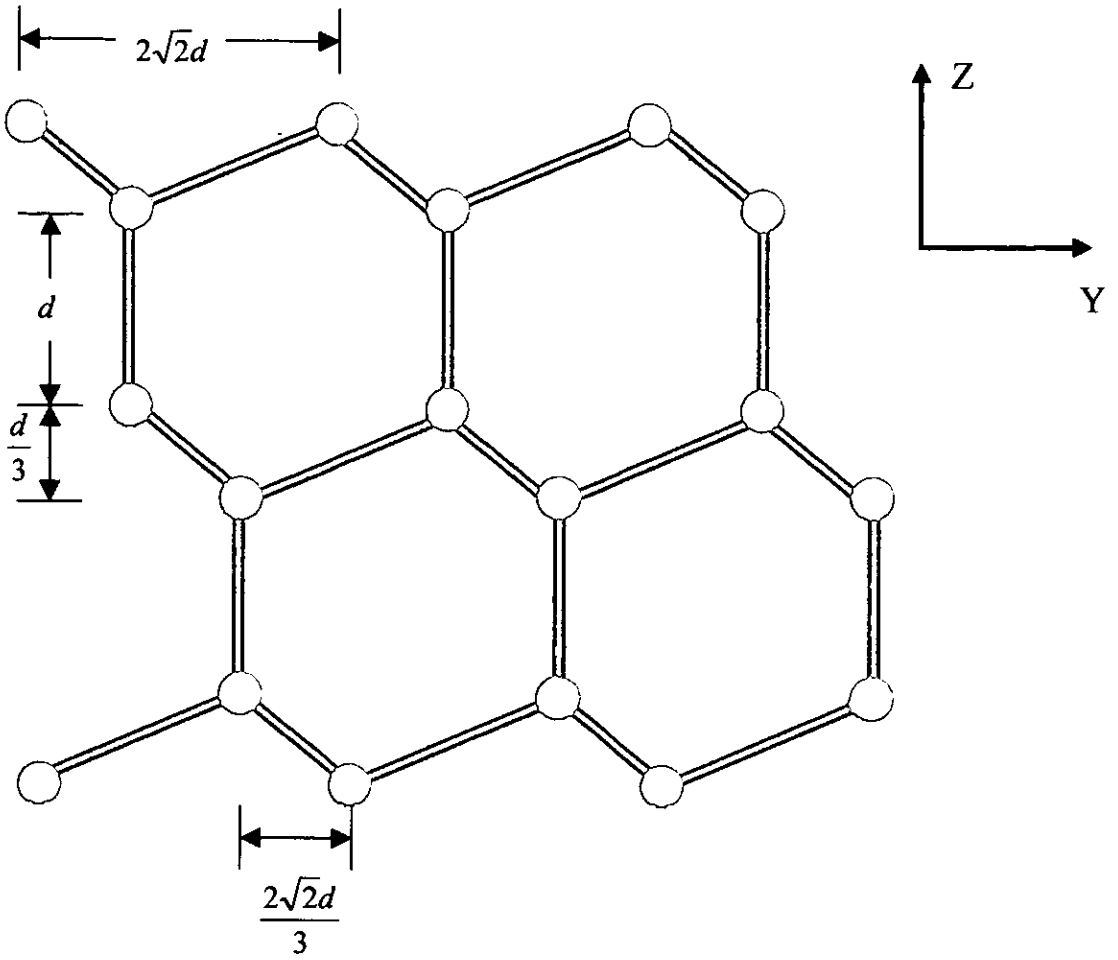


Figura 2.7. Vista lateral de la superficie del Ge(111)-1x1 con las seis capas superiores de la lámina. El valor de la variable *d* es igual a $\frac{\sqrt{3}}{4}a$, donde *a* es el parámetro de la red.

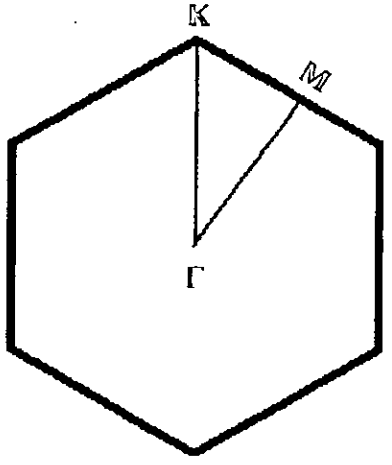


Figura 2.8. Zona de Brillouin de la superficie Ge(111)-1x1 en la que se muestra la parte irreducible junto con los puntos de alta simetría en color rojo.

La densidad de estados $\rho(E)dE$ se define como el número de estados comprendidos en el intervalo de energía que va de E a $E + dE$. Se puede demostrar que la densidad de estados por átomo en el espacio-k está dada por la siguiente integral de superficie [17, página 91]

$$\rho(E_0) = \left(\frac{a}{2\pi}\right)^3 \int_{E=E_0} \frac{dS(\mathbf{k})}{|\nabla_{\mathbf{k}} E(\mathbf{k})|}. \tag{2.16}$$

El parámetro a es la constante del cristal, y la integral se realiza en el espacio-k, sobre la superficie $E = E_0$. En la sección 3.3 vamos a estudiar en forma más detallada la densidad de estado en superficies cristalinas.

En la figura 2.9 se muestra la estructura electrónica de la superficie del Ge(111)-1×1. Del lado derecho también se muestra la densidad de estados electrónicos correspondiente. Nótese la aparición de los estados de superficie dentro de la brecha, que dan lugar a una distribución continua de los estados electrónicos dando un carácter metálico a la superficie. Es decir, la brecha original del sustrato desaparece.

Como podemos ver, arriba de la banda de energía (d) existe una banda de energía sin dispersión, es decir, es totalmente plana. También aproximadamente en –3.0 eV aparece otra banda de energía totalmente plana. Si la banda es casi plana se tiene poca dispersión, por lo que estas bandas dan una gran contribución a la densidad de estados $\rho(E)$. En nuestro ejemplo la banda (d) muestra dispersión por lo que no contribuye significativamente a la densidad de estados. Mientras que en la banda de conducción, ligeramente arriba de la banda (d) se tiene muy poca dispersión, es decir, $|\nabla_{\mathbf{k}} E(\mathbf{k})| \approx 0$ a lo largo de la zona de Brillouin lo cual da una contribución grande a la densidad de estados $\rho(E)$ según la ecuación (2.16), y como podemos observar en la figura 2.9. Por otro lado, en las bandas de valencia, entre 0 y –3.0 eV se tiene una gran dispersión $|\nabla_{\mathbf{k}} E(\mathbf{k})| >> 0$, y aunque hay un gran número de estados con esta energía, no contribuyen tanto a la densidad de estados (2.16) como lo hacen las bandas con poca dispersión.

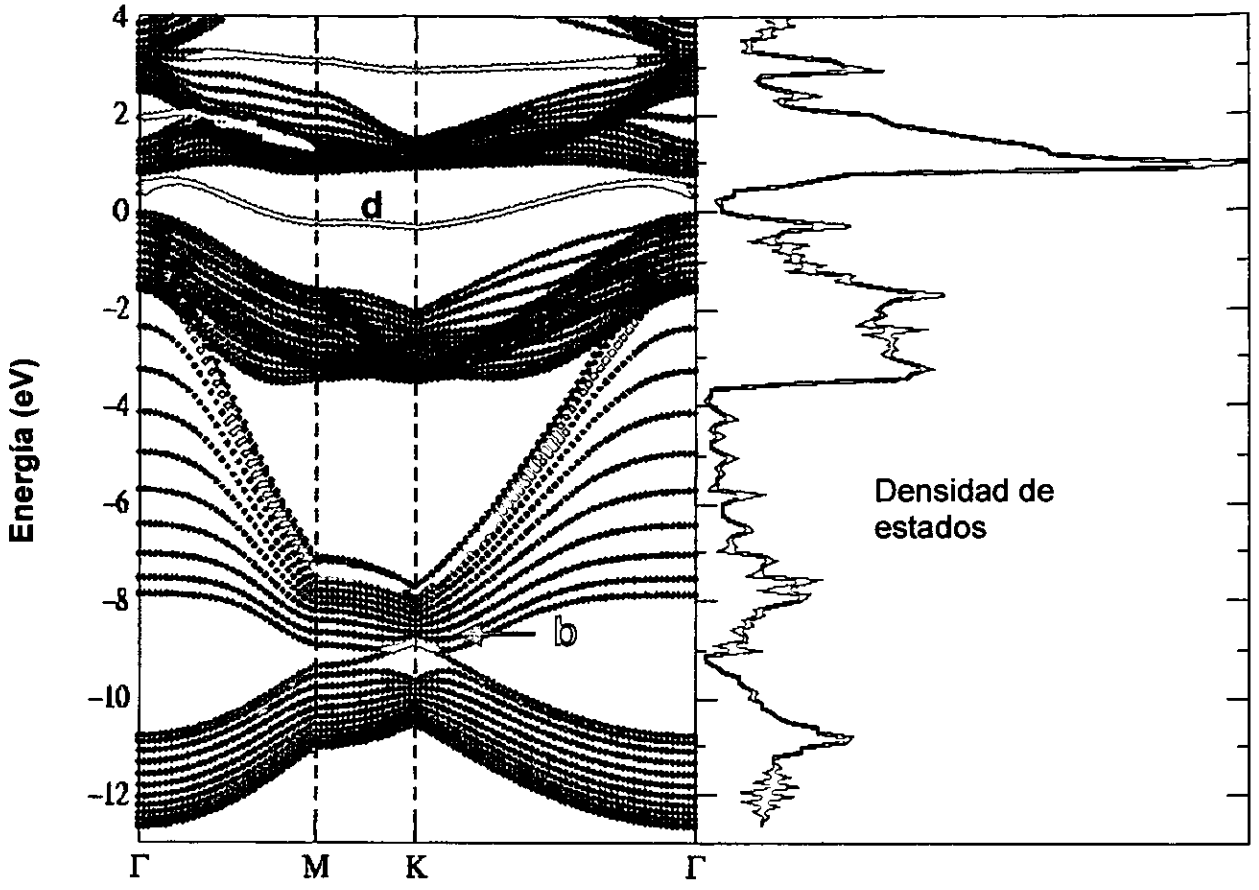


Figura 2.9. Estructura electrónica de la superficie Ge(111)-1x1 junto con su densidad de estados de energía. La banda de energía (d) representa a los estados de superficie de los enlaces colgantes. En la banda de energía (b) están los enlaces llamados de *backbond.*

Se puede observar en la figura 2.9, que la estructura electrónica del Ge(111)-1×1 que calculamos coincide con la estructura electrónica publicada por H. Lüth[2]. La banda etiquetada con la letra (d) corresponde a los estados de superficie de los enlaces colgantes. Mientras que la banda etiquetada con la letra (b) corresponde a los estados llamados de *backbond.* Por los resultados obtenidos con estos dos ejemplos, podemos concluir sin temor a equivocarnos, que las rutinas que calculan los valores propios y vectores propios están correctas. En el siguiente capítulo vamos a calcular la estructura electrónica de la superficie del Ge(111)-2×1 reconstruida utilizando las mismas rutinas que usamos en este último ejemplo.

CAPITULO III

Estructura electrónica del Ge(111)-2×1

La superficie (111) del germanio muestra una reconstrucción (2×1) cuando se corta mecánicamente a temperatura ambiente. Aunque la naturaleza de la reconstrucción (2×1) es desconocida, un candidato posible es la geometría de cadenas con enlaces π propuesta por Pandey[9] para la superficie Si(111)-2×1. Esta hipótesis se sustenta experimentalmente en diferentes espectros de los estados de superficie del Ge(111)-2×1 que se observan en medidas de Fotoemisión Electrónica [10, 11], de Reflectancia Diferencial [12], y de Espectroscopía de Tunelaje [13]. Por otro lado, cálculos teóricos de primeros principios han encontrado que este modelo de Pandey es energéticamente favorable [4, 14].

En este capítulo se calculará la estructura electrónica de la superficie reconstruida del Ge(111)-2×1, ver figura 3.2. Se va a utilizar el método de Enlace Fuerte descrito en el capítulo anterior y los resultados se van comparar con las medidas experimentales y cálculos teóricos, reportados con anterioridad.

3.1 Método de Cálculo.

Para modelar a la superficie del Ge(111)-2×1 vamos a considerar una lámina de 24 capas atómicas, que contiene en total 48 átomos de Ge. En el plano de la superficie se toman condiciones a la frontera periódicas. El espesor de la lámina es lo suficientemente grande para desacoplar los estados de superficie que se presentan tanto en la parte superior como en la inferior. En la figura 3.1 se muestra una vista frontal de la superficie, y su celda unitaria que contiene dos átomos de germanio por capa atómica.

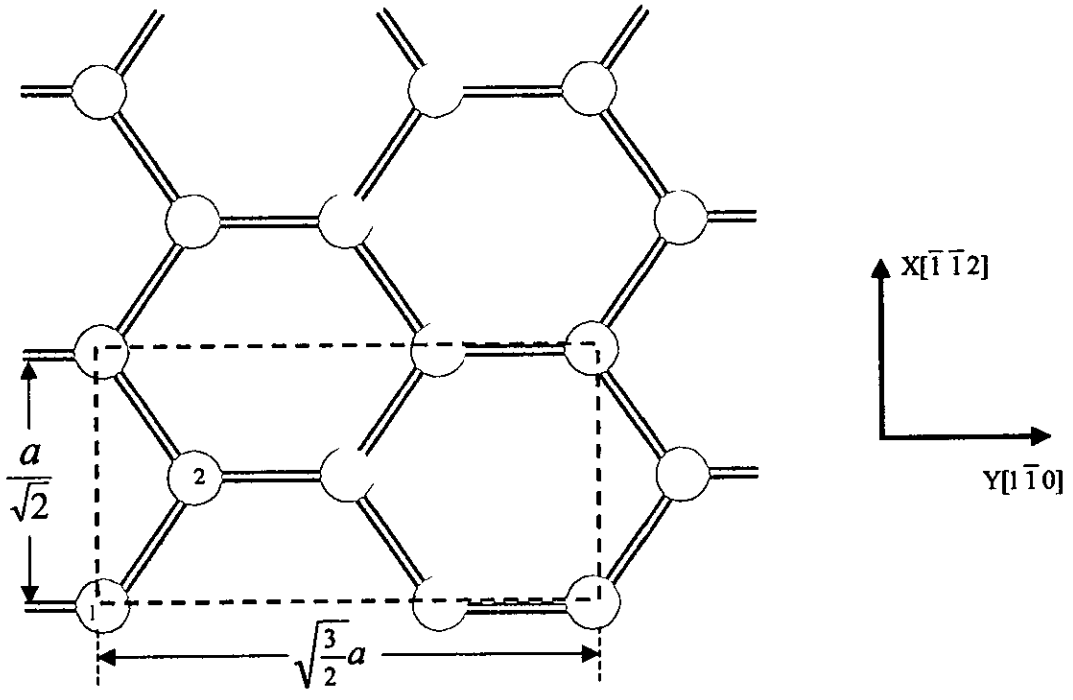


Figura 3.1. Vista frontal de la superficie Ge(111)-2×1, la línea punteada en negro muestra a la celda unitaria de la superficie, cuyas dimensiones se encuentran en términos de la constante de red *a* del sustrato. Los átomos de la celda unitaria están etiquetados con los símbolos "1" y "2" respectivamente.

Para calcular los estados de superficie se utilizó el método de Enlace Fuerte con una base de orbitales sp^3s^*, descrito en el capítulo anterior. Como vimos esta base nos da una buena descripción de la banda de conducción de los cristales semiconductores con geometría cúbica. Se desarrollaron una serie de programas de cómputo que se utilizaron para calcular la estructura electrónica de la superficie reconstruida. Las posiciones de los átomos se tomaron de los resultados teóricos de primeros principios obtenidos por Noboru Takeuchi y colaboradores[14], y que son similares a los calculados por Northrup y cohen[4] también usando un método de primeros principios.

Esta reconstrucción se caracteriza por la formación de cadenas en "zigzag" a lo largo del eje X en la superficie como se muestra en las figuras 3.1 y 3.2.

Otra característica importante de esta reconstrucción, es que los átomos en la superficie que forman la cadena muestran un desplazamiento en la dirección vertical o normal a la superficie. En el modelo de enlace π, en donde π se refiere al carácter del enlace según se explicó en la sección 2.2, que encontró Pandey[9] para la superficie de Si(111)-2×1 y que es muy parecida a la de Ge(111)-2×1, estos átomos que forman la cadena se encontrarían con la misma posición en Z. La longitud del enlace entre los átomos que forman a la cadena en la superficie es 5.351 Å, la cual es ligeramente menor que la longitud del enlace en el sustrato 5.658 Å. Por otro lado, el desplazamiento vertical entre los átomos que forman la cadena en la superficie es de 0.264 Å. En las cuatro capas atómicas adyacentes a la superficie también se encuentran desplazamientos atómicos importantes respectos a las posiciones que guardarían en el sustrato sin reconstruir.

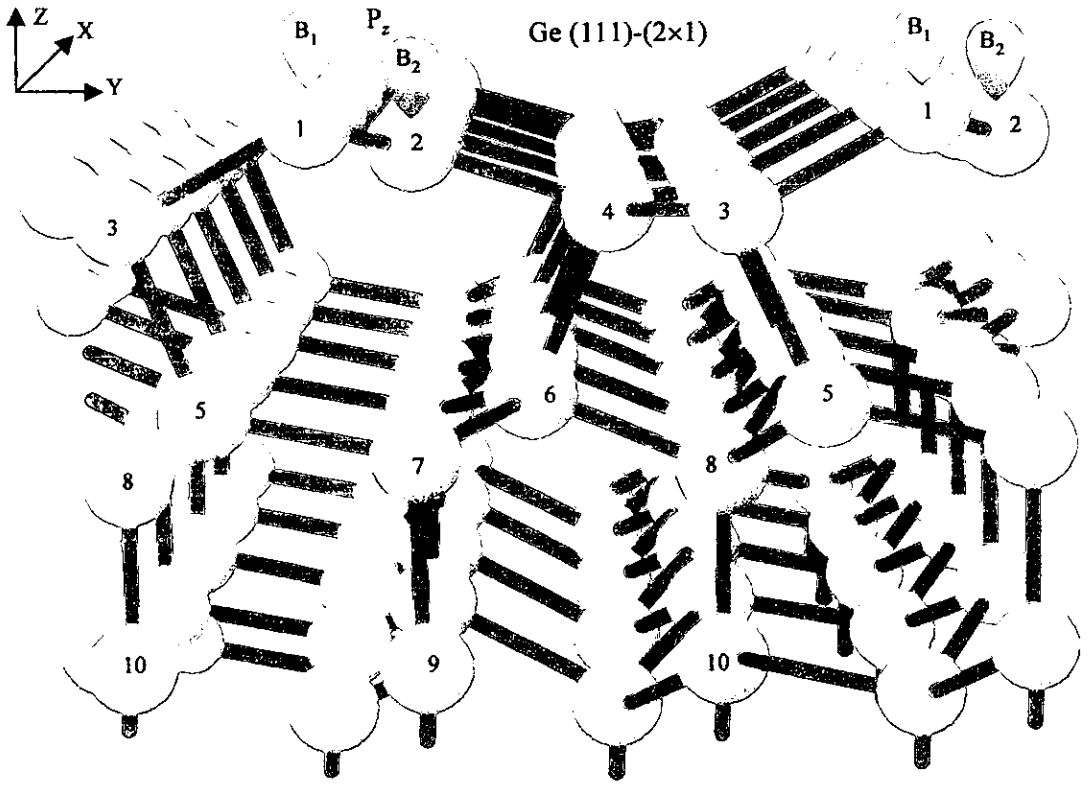


Figura 3.2. Modelo de la reconstrucción atómica de la superficie Ge(111)-2×1. Los lóbulos etiquetados con B_1 y B_2 son orbitales con carácter P_z. Como podemos observar, a lo largo del eje X tenemos cadenas de átomos etiquetados con "1", con orbitales colgantes ocupados, y con átomos etiquetados con "2" que tienen orbitales colgantes desocupados.

Con las posiciones atómicas descritas en el párrafo anterior, ahora vamos a calcular la estructura electrónica correspondiente. A continuación se describe el algoritmo que se utilizó para este fin:

i) **surf2x1:** Llama a la función GetData() para leer las energías E_{ss}, E_{px}, E_{py}, E_{pz}, $E_{s^*s^*}$, V_{ss}, V_{sp}, V_{ps}, $V_{pp\pi}$, $V_{pp\sigma}$ y $V_{s^*p\sigma}$ que se utilizan para generar a la matriz Hamiltoniana. En el caso de la superficie reconstruida se leen las nuevas posiciones atómicas de los átomos de las primeras diez capas. Las capas atómicas restantes conservan las posiciones que tienen en el sustrato. Como las energías de la matriz de interacción V_{ss}, V_{sp}, V_{ps}, $V_{pp\pi}$, y $V_{pp\sigma}$, se obtuvieron para el sustrato, en el caso de la superficie se utiliza la regla de Harrison [5] para su extrapolación. Es decir, los nuevos parámetros se ajustan por la cantidad $\frac{|\mathbf{r}|^2}{d^2}$, donde **r** es la distancia entre los átomos primeros vecinos de la superficie reconstruida y $d = \frac{\sqrt{3}}{4}a$, es la distancia entre dos átomos vecinos en el sustrato. A continuación se recorre la primera zona de Brillouin a lo largo de los puntos de alta simetría, Γ-J-K-J'-Γ. Este programa llama a los subprogramas GetMatHam(), GetEigen() y WriteEner(), que se describen a continuación.

ii) **GetMatHam**: Calcula las posiciones de los átomos y la distancia entre átomos que son primeros vecinos, cuyas interacciones son las únicas que se consideran. También construye a la matriz de interacción (*GetMatInt*), calcula las sumas de Bloch (*GetMatHRl*) y construye a la matriz Hamiltoniana tanto con su parte real como imaginaria (*join*).

iii) **GetEigen**: Calcula los valores y vectores propios del Hamiltoniano. Ya diagonalizada la matriz, los vectores propios se encuentran en los renglones de la matriz Hamiltoniana.

iv) **WriteEner**: Etiqueta los estados electrónicos distinguiendo entre los de superficie y de sustrato. Como sabemos los estados de superficie se encuentran localizados en una vecindad de la superficie. Basados en este hecho, se escoge un parámetro δ, tal que, cuando los estados cuya probabilidad en las primeras capas atómicas de la lámina es mayor que δ, se etiquetan como estados de superficie. Mientras que aquellos estados cuya probabilidad en las primeras capas atómicas es menor que δ se etiquetan como estados proyectados del sustrato.

3.2 Resultados y discusión

Los estados de superficie y la proyección de los estados del sustrato calculados teóricamente se presentan en la figura 3.3. En la estructura de bandas de la superficie del Ge(111)-2×1 reconstruida encontramos que los estados de superficie etiquetados con B_2 tienen un carácter P_z, es decir, son enlaces colgantes caracterizados con un orbital tipo p en la dirección normal a la superficie (z). Esta banda también se caracteriza por un enlace de carácter π entre orbitales tipo P a lo largo de las direcciones del plano XY. Por este motivo el enlace y antienlace de los átomos 1 y 2 forma las cadenas llamadas π (ver figura 2.2), en analogía al modelo de Pandey para la superficie del Si(111)-2×1. Estos estados B_2 y cuyo carácter es P_z sé encuentran localizados en el átomo etiquetado como 2 en la figura 3.2 y que pertenecen a la primera capa de la superficie. Esta banda B_2 se encuentra por arriba del nivel de Fermi, y por lo tanto, se encuentra desocupada. En el punto Γ, la banda vacía B_2 se encuentra alrededor de 0.56 eV por arriba del máximo de la banda de valencia, y alcanza su mínimo en el punto de alta simetría K y tiene un valor de 0.18 eV, que es el tamaño de la brecha indirecta como se observa en la gráfica de la estructura de bandas. Por otro lado, el máximo de B_2 se encuentra alrededor del punto J' y tiene un valor de 0.9 eV. Además, la dispersión que se observa entre los puntos K-J' que corresponde en el espacio real a la dirección X, se debe a que los electrones se pueden mover de manera más libre a lo largo de los enlaces que forman las cadenas-π de los átomos en la superficie. Mientras que en la dirección J-K la dispersión es mucho menor, y esto se debe a la poca interacción de los electrones que se encuentran en cadenas adyacentes, es decir, en la dirección correspondiente a Y en el plano de la superficie. Nótese que la distancia entre cadenas adyacentes es del tamaño $\sqrt{\frac{3}{2}}a > d$.

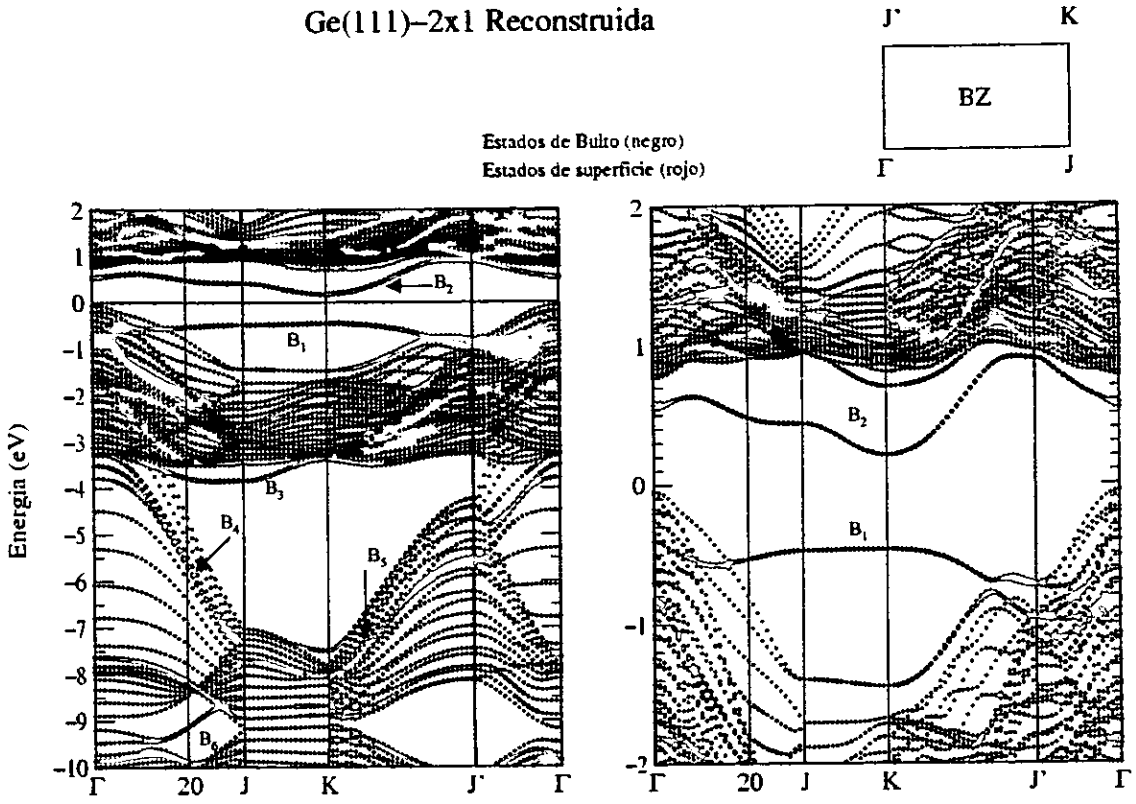


Figura 3.3. Estructura electrónica de la superficie Ge(111)-2×1 reconstruida. En el cuadro de la parte izquierda se presentan las bandas de energía entre los valores –10.0 eV a 2.0 eV. Como las bandas B_1 y B_2 muestran un carácter P_z se les conoce como estados energéticos π. Mientras que a los estado energéticos de la banda B_6 por tener carácter S, se les llama estados σ. En el cuadro derecho se da una vista de los estados de superficie en el rango de –2.0 a 2.0 eV. Como puede observarse, la banda de energía B_2 muestra gran dispersión.

Por debajo del nivel de Fermi, se encuentra una banda de estados electrónicos de superficie etiquetada como B_1 que también tienen carácter P_z y que se encuentran localizados en los átomos etiquetados como "1" en la figura 3.2. Esta banda B_1 también se debe al enlace π entre los átomos que forman la cadena. Se encuentra por debajo del nivel de Fermi, y por lo tanto, se encuentra ocupada.

En el punto Γ, la banda ocupada B_1 se localiza aproximadamente a 0.69 eV por debajo del máximo de la banda de valencia, y alcanza su mínimo en el punto de alta simetría J' y tiene un valor de –0.73 eV. Por otro lado, el máximo de B_1 se encuentra alrededor del punto de alta simetría K y tiene un valor de -0.46 eV. Como podemos observar en la figura 3.3, la banda de energía B_1 muestra una dispersión menor que la banda desocupada B_2. En los puntos de alta simetría J y K la banda es bastante plana. También podemos observar que B_1 se mete dentro de los estados proyectados del sustrato cuando se acerca al punto Γ.

El mínimo de la banda B_2 y el máximo de la banda B_1 nos dan una brecha directa en el punto de alta simetría K de aproximadamente 0.65 eV. Experimentalmente usando técnicas de Reflectancia Diferencial, Espectroscopía de Fotoemisión y de Espectroscopía de Tunelaje para la superficie Ge(111)-(2×1), se reportaron valores de la brecha que van desde 0.45 eV hasta 0.75 eV [12], de la referencia [14] ver la tabla 5. Mientras que otros cálculos teóricos basados en métodos de primeros principios encontraron valores muy pequeños de la brecha, de aproximadamente 0.25 eV[4] y de 0.56 eV[14]. Estos cálculos teóricos de primeros principios se basan en la Teoría de la Funcional de la Densidad. Esta teoría [15], no describe de manera adecuada estados electrónicos excitados y por esto, la brecha que se encuentra siempre está subestimada. Sin embargo, se puede corregir en una primera aproximación[15]. Se encontró que haciendo correcciones en la Teoría de la Funcional de la Densidad [15] la brecha que se obtiene es de 0.67 eV, es muy parecida a la que nosotros calculamos (0.65 eV). En nuestros cálculos se encontró que la energía de Fermi (E_F) es igual a 0.04 eV. Los cálculos de primeros principios dan un valor de 0.1 eV, muy parecido al nuestro [14].

La banda de energía B_2 que encontramos muestra una dispersión entre los puntos que van de Γ a K, de 0.69 eV, muy cercano al valor calculado por Northrup y Cohen [4] de 0.8 eV y por Zhu y Louie[15]. Mientras que la banda de energía B_1 en nuestros cálculos presenta una dispersión más pequeña, de 0.27 eV en los mismos puntos de simetría. La banda B_2 es totalmente plana de K a J, y esto se debe a las cadenas de átomos que se forman a lo largo del eje X, y a que los electrones en las diferentes cadenas (dirección Y) no interaccionan entre sí. Los estados de superficie de las bandas B_1 y B_2 se les llaman estados π, por tener un fuerte carácter de enlace y antienlace, respectivamente, entre estados P_x y P_y.

También de la figura 3.3 se observa una banda de estados de superficie etiquetada por B_3 que se encuentra por debajo del nivel de Fermi a energías entre 3.0 y 4.0 eV. Estos estados se deben principalmente al enlace entre los átomos de la primera capa atómica y la segunda. En particular, debido al enlace entre el átomo "2" y el átomo "4". A energías más bajas también se encuentran estados de superficie etiquetados con B_4 con el mismo origen físico.

Los estados de superficie de la banda B_4 también se presentan tanto en la superficie ideal del Ge(111)-2×1 como en la superficie reconstruida. Muestran un carácter S, P_y, muy fuerte en las capas 4 y 5, mientras que en la superficie ideal su carácter es P_x, P_y. Por lo que el proceso de reconstrucción al formar cadenas de átomos a lo largo de X cambia el carácter de los estados de energía en la superficie reconstruida.

Las bandas B_5, y B_6 en la superficie reconstruida muestran un carácter S, es decir, son estados de superficie debido a enlaces de tipo σ de los átomos involucrados en la reconstrucción. En la superficie ideal muestran un carácter S

en la primera capa, y P_x, P_y, en la segunda capa. Al reconstruirse la superficie Ge(111)-2×1 ideal, la banda de energía B_6 cambia a un carácter S únicamente. A los estados de superficie de las bandas B_5 y B_6 por su carácter S se les llama estados σ. La banda B_6 se encuentra en una brecha que está a –9.00 eV de la banda de valencia, entre los puntos Γ y J, Northrup y Cohen[4] también en sus cálculos encontraron a esta banda de estados de superficie situada a –8.00 eV. La dispersión de esta banda es creciente y es de 1.00 eV. Estos estados energéticos están asociados a los enlaces σ entre los átomos involucrados en la reconstrucción de la superficie.

Los resultados teóricos publicados por diferentes autores que aplicaron el mismo método (Densidad Funcional) no coinciden como puede verse al consultar las referencias [4, 14 y 15]. Una situación similar se presenta con los resultados experimentales, por ejemplo, los experimentos de Espectroscopía de Fotoemisión hechos por Nicholls y sus colaboradores[10] muestran una banda ocupada altamente dispersiva con un ancho de banda de 0.8 eV, en tanto que, F. Solal y sus colaboradores [11] muestran una dispersión de sólo 0.2 eV, cuyo valor es muy similar al que obtuvimos en nuestros resultados teóricos de 0.27 eV, ver figura 2 de la referencia [15].

En conclusión, sería necesario realizar más cálculos y medidas experimentales de otras propiedades físicas como la pérdida de energía de electrones, de reflectancia diferencial, etc., para poder aclarar cual método de cálculo está más cercano a la realidad, y si el modelo teórico propuesto corresponde o no a la reconstrucción de ésta superficie.

3.3 Densidad de estados

Otra cantidad importante en el análisis de la estructura electrónica de materiales es la densidad local de estados de superficie (DOS) por capa atómica m, que podemos definir para cada plano del sistema como,

$$\rho^m(E) = \sum_{\alpha,j,\mathbf{k}} |\chi_\alpha^{jm}(\mathbf{k})|^2 \, \delta(E - E(\mathbf{k})), \tag{3.1}$$

donde se suma sobre todos los puntos de la ZB de la superficie, los orbitales α y los átomos α de la celda unitaria. Usando la función de onda definida por la ecuación (2.13) encontramos que la densidad de estados se puede reescribir en función de los coeficientes de la expansión lineal,

$$\rho^m(E) \propto \sum_{\alpha,j,\mathbf{k}} |d_\alpha^{jm}(\mathbf{k})|^2 \, \delta(E - E(\mathbf{k})) \tag{3.2}$$

que se obtienen al diagonalizar la matriz de la ecuación (2.14). Finalmente, la densidad de estados total del sistema será igual a la suma de las densidades parciales por plano,

$$\rho(E) = \sum_m \rho^m(E), \tag{3.3}$$

En la figura 3.4, se graficó la densidad de estados total y para las tres primeras capas atómicas de la superficie del Ge(111)-2×1. En las gráficas se pueden ver claramente los estados de superficie al comparar las densidades de los primeros planos con la densidad total del sistema. En el primer plano se observan principalmente los estados dentro de la brecha asociados a los enlaces sueltos y las cadenas. Mientras nos alejamos de la superficie estas estructuras desaparecen y la densidad de estados se parece mas a la densidad de estados del bulto. Esto puede observarse claramente en la figura 3.4. La contribución principal a la densidad de estados para las bandas de energía etiquetadas con B_1 y B_2 esta dada por los átomos de la primera, segunda y tercera capas atómicas, en ese orden. Esto confirma que los estados energéticos que pertenecen a estas bandas son estados de superficie, porque están localizados en las primeras capas atómicas.

También podemos observar en la figura 3.4, que la densidad de estados total entre los −3.0 y 2.0 eV es similar con la densidad de estados de la superficie Ge(111)1×1 (ver figura 2.9), por lo que se puede concluir que ambas tienen el mismo origen que se discutió con anterioridad en el capítulo II.

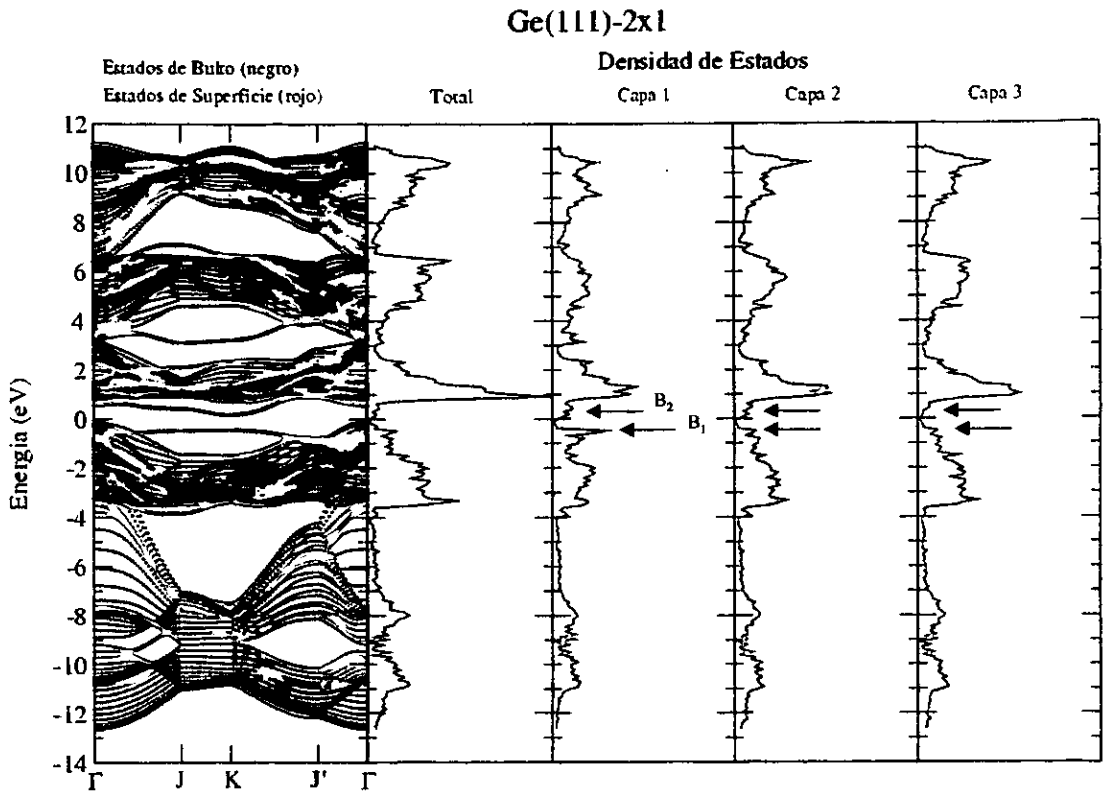


Figura 3.4. Densidad de estados de superficie del Ge(111)-2×1 reconstruida. En el cuadro que está más a la izquierda se muestra la estructura electrónica de la superficie Ge(111)- 2×1 reconstruida. Después se muestra la densidad de estados total del sistema. En los cuadros siguientes se muestra la densidad de estados de la primera, segunda y tercera capa atómica de la superficie reconstruida.

CONCLUSIONES

En este trabajo se calculó la estructura electrónica de la superficie Ge(111)-2×1 reconstruida. Para llevar a cabo ésta tarea, se desarrollaron una serie de programas en lenguaje C++ que implementaron algoritmicamente el método de Enlace Fuerte con la aproximación a primeros vecinos, se utilizó una base de orbitales atómicos sp^3s^*, en donde, s y s^* son orbitales con simetría esférica, s^* es un orbital excitado, y los orbitales p corresponden a los números cuánticos con (l = 1).

Para verificar la ejecución correcta de nuestros programas, primeramente se calculó la estructura electrónica del cristal de Ge, y se compararon los datos obtenidos con los resultados publicados por otros autores. Un resultado importante que obtuvimos es que la base sp^3s^* preserva los estados de energía de la banda de valencia, reproduce exitosamente los estados energéticos que se encuentran en la banda de conducción, y también muestra correctamente la brecha indirecta del germanio. Nuestros programas también se pueden utilizar para calcular la estructura electrónica de otros cristales semiconductores (C, Si, Sn-α, GaAs, Ga-c, etc.).

En nuestro cálculo de la estructura electrónica de la superficie Ge(111)-2x1 reconstruida, se observó que en la brecha hay dos bandas B_1 y B_2 debido a la reconstrucción, y la dispersión calculada por nuestro modelo teórico de la banda de estados de superficie B_1 con carácter π coincide cualitativamente con la dispersión observada experimentalmente por Nicholls[10].

Nuestros resultados, al igual que los obtenidos por Northrup y Cohen[4] predicen la existencia de una banda de estados de superficie con carácter σ en una brecha dentro de la estructura electrónica de la banda de valencia aproximadamente a 9.0 eV abajo del máximo de la banda de valencia.

Un resultado importante que se obtuvo de nuestro modelo teórico es el tamaño de la brecha directa de la superficie Ge(111)-2×1 reconstruida con un valor aproximado de 0.65 eV, que cae dentro del rango de valores (0.45, 0.75 eV) reportados por otros autores que utilizaron técnicas experimentales como la Reflectancia Diferencial, Fotoemisión, y Espectroscopía de Tunelaje.

En resumen, la base sp^3s^*, hace posible el cálculo con una buena aproximación, minimizando el número de parámetros empíricos que se requieren para calcular la estructura electrónica de la superficie Ge(111)-2×1 reconstruida. Esto nos permite conocer las propiedades físicas de la superficie, y que podrían utilizarse para el desarrollo de nuevas tecnologías.

Hace falta explorar tanto teóricamente como experimentalmente las propiedades físicas de la superficie Ge(111)-2x1 reconstruida para conocer con certeza su estructura atómica real y el método teórico más adecuado.

APENDICE

LISTADOS DE PROGRAMAS

```
/*************************************************************************
*                Archivo vecmat.h                                        *
*                Definición de clases Matriz y Vector                    *
*************************************************************************/
#include <iostream.h>
enum  state { ITERATING, SUCCESS, WONTSTOP, NEARZERO, SAMESIGN } ;
class matrix ;

class vector {
friend matrix ;
private:
   int size ;
   double *vec ;
public:
   vector (int) ;
   vector (const vector&) ;
   vector (const double *, int ) ;
   vector (const char *) ;
   ~vector() { delete vec ; } ;
   double& operator[] (int i) { return vec[i] ; } ;
   vector& operator+() { return *this ; } ;
   vector& operator=(const vector&) ;
   vector& operator+=(const vector&) ;
   vector& operator-=(const vector&) ;
   vector& operator*=(double) ;          // Multiplicado por un double
   vector& operator/=(double) ;          // Dividido por un double
   int getsize () { return size ; } ;
   void swap (int, int) ;
   friend double norm (const vector&) ;              // Norma Euclidiana
   friend double norminf (const vector&) ;           // Norma Infinito
   friend vector operator-(const vector&) ;
   friend vector operator+(const vector&, const vector&) ;
   friend vector operator-(const vector&, const vector&) ;
   friend vector operator*(const vector&, double) ;
   friend vector operator*(double, const vector&) ;
   friend vector operator/(const vector&, double) ;
   friend double scalar(const vector&, const vector&) ; // Producto Escalar
   friend vector operator*(const matrix&, const vector&) ; // v = A.x
   friend vector operator*(const vector&, const matrix&) ; // v = b.B
   friend matrix operator*(vector&, vector&) ; // A = v.x
   friend matrix operator*(const matrix&, const matrix&) ; // C = A.B
   friend double norm (const matrix&) ;              // Norma Euclides
   friend double norminf (const matrix&) ;           // Norma Infinita
   friend ostream& operator<<(ostream&, const vector&) ;
   friend istream& operator>>(istream&, vector&) ;
   friend ostream& operator<<(ostream&, const matrix&) ;
   friend istream& operator>>(istream&, matrix&) ;
} ;

class matrix {
private:
   int numrows ;
   int numcols ;
   vector **mat ;
public:
   matrix (int, int) ;        // Matriz Rectangular
   matrix (int) ;             // Matriz Cuadrada
   matrix (const matrix&) ;
   matrix (const char *) ;
   ~matrix() ;
   vector& operator[](int i) { return *mat[i] ; }
   matrix& operator+() { return *this ; }
```

```
   matrix& operator=(const matrix&) ;
   matrix& operator+=(const matrix&) ;
   matrix& operator-=(const matrix&) ;
   matrix& operator*=(double) ;
   matrix& operator/=(double) ;
   friend  int join(matrix&, matrix&, matrix&) ;
   int getnumrows() { return numrows ; } ;
   int getnumcols() { return numcols ; } ;
   int getsize() ;
   void swap (int, int) ;
   matrix transpose() ;
   friend matrix operator-(const matrix&) ;              // Menos unario
   friend matrix operator+(const matrix&, const matrix&) ; // A = B + C
   friend matrix operator-(const matrix&, const matrix&) ; // A = B - C
   friend matrix operator*(const matrix&, const matrix&) ; // A = B * C
   friend matrix operator*(const matrix&, double) ;      // A = c * B
   friend matrix operator*(double, const matrix&) ;      // A = B * c
   friend matrix operator/(const matrix&, double) ;      // A = B / c
   friend vector operator*(const matrix&, const vector&) ; // v = A * x
   friend vector operator*(const vector&, const matrix&) ; // v = b * B
   friend matrix operator*(vector&, vector&) ;           // A = v * x
   friend double norm (const matrix&) ;                  // Norma Euclides
   friend double norminf (const matrix&) ;               // Norma Infinita
   friend int insmat (matrix&, matrix&, int, int) ;
   friend int insmat (matrix&, int, int, matrix&, int, int, int) ;
   friend ostream& operator<<(ostream&, const matrix&) ;
   friend istream& operator>>(istream&, matrix&) ;
} ;

inline vector operator*(double d, const vector& v)
{
   return v * d ;
}
inline matrix operator*(double d, const matrix& m)
{
   return m * d ;
}

/*****************************************************
*  Archivo: vector.cpp                               *
*  Rutinas para el manejo de vectores                *
*****************************************************/
#include   <fstream.h>
#include   <math.h>
#include   <stdlib.h>
#include   "vecmat.h"

inline void fatal (const char *str)
{
   cerr << str ;    exit (1) ;
}

vector::vector (int n)
{
   size = n ;
   vec  = new double [size] ;
   if ( !vec )  fatal ("No hay memria para crear vector\n") ;
   for (int i = 0 ; i < size ; ++i )
      vec[i] = 0 ;
}
```

```
vector::vector (const double *a, int n)
{
   size = n ;
   vec  = new double [size] ;
   if ( !vec )
      fatal ("No hay memoria para crear vector\n") ;
   for (int i = 0 ; i < size ; ++i )
      vec[i] = a[i] ;
}

vector::vector (const vector &v)
{
   size = v.size ;
   vec  = new double [size] ;
   if ( !vec )
      fatal ("No hay memoria para crear vector\n") ;
   for (int i = 0 ; i < size ; ++i )
      vec[i] = v.vec[i] ;
}

vector::vector (const char *path)
{
   int      n ;
   ifstream inpdata (path) ;
   if ( !inpdata )
      fatal ("No pude abrir al archivo de datos: ") ;
   inpdata >> n ;
   size = n ;
   vec  = new double [size] ;
   for (int i = 0; i < n; i++ )
      inpdata >> vec[i] ;
}

vector& vector::operator=(const vector& v)
{
   if ( v.size != size )
      cerr << "Los vectores son de tamaños diferentes\n" ;
   for (int i = 0; i < size; i++)
      vec[i] = v.vec[i] ;
   return *this ;
}

vector& vector::operator+=(const vector &v)
{
   if ( v.size != size )
      cerr << "Los vectores son de tamaños diferentes\n" ;
   for (int i = 0; i < size; i++)
      vec[i] += v.vec[i] ;
   return *this ;
}

vector& vector::operator-=(const vector &v)
{
   if ( v.size != size )   cerr << "Los vectores son de tamaños diferentes\n" ;
   for (int i = 0; i < size; i++)
      vec[i] -= v.vec[i] ;
   return *this ;
}

vector& vector::operator*=(double x)
{
   for (int i = 0; i < size; i++)
```

```
      vec[i] *= x ;
   return *this ;
}

vector& vector::operator/=(double x)
{
   for (int i = 0; i < size; i++)
      vec[i] /= x ;
   return *this ;
}

void vector::swap (int i, int j)
{
   double tmp = vec[i] ;
   vec[i] = vec[j] ;
   vec[j] = tmp ;
}

vector operator-(const vector &v)
{
   int    n = v.size ;
   vector u(n) ;
   for (int i = 0; i < n; ++i)
      u.vec[i] = -v.vec[i] ;
   return u ;
}

vector operator+(const vector& v1, const vector& v2)
{
   int    n = v1.size ;
   if (v1.size != v2.size )
      cerr << "No puedo sumar dos vectores de diferente tamaño\n" ;
   vector v(n) ;
   for (int i = 0; i < n; i++ )
      v.vec[i] = v1.vec[i] + v2.vec[i] ;
   return (v) ;
}

vector operator-(const vector& v1, const vector& v2)
{
   int    n = v1.size ;
   if (v1.size != v2.size )
      cerr << "No puedo sumar dos vectores de diferente tamaño\n" ;
   vector v(n) ;
   for (int i = 0; i < n; i++ )
      v.vec[i] = v1.vec[i] - v2.vec[i] ;
   return (v) ;
}

vector operator*(const vector& v, double d)
{
   vector vd = v ;
   vd *= d ;
   return vd ;
}

inline vector operator*(double d, vector& v)
{
   return v * d ;
}
```

```
vector operator/(const vector& v, double d)
{
   vector vd = v ;
   retrun (vd /= d ) ;
}

double scalar (const vector& u, const vector& v)
{
   double t = 0 ;
   int n = v.size ;
   if (v.size != u.size )  cerr << "No puedo hacer el producto escalar dos vectores " <<  "con diferente
tamaño\n" ;
   for (int i = 0; i < n; i++)
      t+= u.vec[i] * v.vec[i] ;
   return t ;
}

double norm(const vector &v)
{
   double t = 0.0 ;
   for (int i = 0; i < v.size; i++) {
      double vi = v.vec[i] ;
      t += vi * vi ;
   }
   return sqrt(t) ;
}

double norminf(const vector &v)
{
   double t = 0.0 ;
   for (int i = 0; i < v.size; i++ ) {
      double vi = fabs (v.vec[i]) ;
           if ( vi > t ) t = vi ;
  }
  return t ;
}

ostream& operator<<(ostream &s, const vector &v)
{
   int n = v.size ;
   s << "( " ;
   for (int i = 0; i < n - 1; ++i)
      s << v.vec[i] << ", " ;
   s << v.vec[n - 1] << " )" ;
   return s ;
}

istream& operator>>(istream &s, vector &v)
{
   int n = v.size ;
   cout << "Teclee los " << n << " componentes del vector\n" ;
   for (int i = 0; i < n; i++) {
      cout << "v[" << i << "] = " ;
      s >> v.vec[i] ;
   }
   return s ;
}

/*****************************************************
*  Archivo: matrix.cpp                              *
*  Rutinas para el manejo de matrices               *
*****************************************************/
```

```
#include   <fstream.h>
#include   <iomanip.h>
#include   <stdlib.h>
#include   <math.h>
#include   "vecmat.h"

inline void fatal (const char *str)
{
   cerr << str ;
   exit (1) ;
}

matrix::matrix (int n, int m)
{
   numrows = n ;
   numcols = m ;
   mat     = new vector* [numrows] ;
   if ( ! mat )  fatal ("no hay memoria para crear matriz\n") ;
   for (int i = 0 ; i < numrows ; ++i ) {
      mat[i] = new vector(numcols) ;
          if ( ! mat[i] )  fatal ("no hay memoria para crear matriz\n") ;
   }
}

matrix::matrix (int n)            // matriz cuadrada
{
   numcols = numrows = n ;
   mat     = new vector* [numrows] ;
   if ( ! mat )  fatal ("no hay memoria para crear matriz\n") ;
   for (int i = 0 ; i < numrows ; ++i ) {
      mat[i] = new vector(numcols) ;
          if ( ! mat[i] )  fatal ("no hay memoria para crear matriz\n") ;
   }
}

matrix::matrix (const matrix &m)
{
   int  i ;
   numrows = m.numrows ;
   numcols = m.numcols ;
   mat     = new vector* [numrows] ;
   if ( ! mat )  fatal ("no hay memoria para crear matriz\n") ;
   for (i = 0 ; i < numcols ; ++i ) {
      mat[i] = new vector(numcols) ;
          if ( ! mat[i] )  fatal ("no hay memoria para crear matriz\n") ;
   }
   for ( i = 0; i < numrows; i++ )
      *mat[i] = *m.mat[i] ;
}

matrix::matrix (const char *path)
{
   int      i, n, m ;
   ifstream  inpdata (path) ;
   if ( !inpdata)  fatal ("no pude abrir al archivo de datos\n") ;
   inpdata >> n ;
   numrows = n ;
   inpdata >> m ;
   numcols = m ;
   mat     = new vector* [numrows] ;
   if ( ! mat )  fatal ("no hay memoria para crear matriz\n") ;
   for (i = 0; i < numrows; i++)
```

```
      mat[i] = new vector (numcols) ;
   for ( i = 0; i < n; i++ ) {
      for (int j = 0; j < m; j++)
         inpdata >> mat[i]->vec[j] ;
   }
   inpdata.close() ;
}

matrix::~matrix()
{
   for (int i = numrows; i > 0; --i)
      delete mat[i - 1] ;
   delete mat ;
}

matrix& matrix::operator=(const matrix& m)
{
   if ( m.numrows != numrows || m.numcols != numcols )  fatal ("las matrices son de tamaños diferentes\n") ;
   for (int i = 0; i < numrows; i++)
      *mat[i] = *m.mat[i] ;
   return *this ;
}

int matrix::getsize()
{
   if (numrows != numcols )   cerr << "getsize es válido sólo para matrices cuadradas\n" ;
   return numrows ;
}

matrix& matrix::operator+=(const matrix &m)
{
   if ( m.numcols != numcols || m.numrows != numrows )  fatal ("las matrices son de tamaños diferentes\n") ;
   for (int i = 0; i < m.numrows; i++)
      for (int j = 0; j < numcols; j++)
         mat[i]->vec[j] += m.mat[i]->vec[j] ;
   return *this ;
}

matrix& matrix::operator-=(const matrix &m)
{
   if ( m.numcols != numcols || m.numrows != numrows )  fatal ("las matrices son de tamaños diferentes\n") ;
   for (int i = 0; i < m.numrows; i++)
      for (int j = 0; j < numcols; j++)
         mat[i]->vec[j] -= m.mat[i]->vec[j] ;
   return *this ;
}

matrix& matrix::operator*=(double x)
{
   for (int i = 0; i < numrows; i++)
      for (int j = 0; j < numcols; j++)
         mat[i]->vec[j] *= x ;
   return *this ;
}

matrix& matrix::operator/=(double x)
{
   for (int i = 0; i < numrows; i++)
      for (int j = 0; j < numcols; j++)
         mat[i]->vec[j] /= x ;
   return *this ;
}
```

```
void matrix::swap (int i, int j)
{
   vector *tmp = mat[i] ;
   mat[i] = mat[j] ;
   mat[j] = tmp ;
}

matrix matrix::transpose()
{
   int p = numrows, q = numcols ;
   matrix mt (q, p) ;
   for (int i = 0; i < q; i++) {
      for (int j = 0; j < p; ++j)
            mt.mat[i]->vec[j] = mat[j]->vec[i] ;
   }
   return mt ;
}

matrix operator-(const matrix &m)
{
   int    n = m.numrows, r = m.numcols ;
   matrix u(n, r) ;
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < r; ++j)
         u.mat[i][j] = -m.mat[i][j] ;
   return u ;
}

matrix operator+(const matrix& m1, const matrix& m2)
{
   if ( m1.numrows != m2.numrows || m1.numcols != m2.numcols )  fatal ("las matrices son de tamaños
diferentes\n") ;
   matrix mt = m1 ;
   mt += m2 ;
   return mt ;
}

matrix operator-(const matrix& m1, const matrix& m2)
{
   if ( m1.numrows != m2.numrows || m1.numcols != m2.numcols )  fatal ("las matrices son de tamaños
diferentes\n") ;
   matrix mt = m1 ;
   mt -= m2 ;
   return mt ;
}

vector operator*(const matrix &m, const vector &v)
{
   int nr = m.numrows ;
   if (m.numcols != v.size )  fatal ("la matriz no puede aplicarse al vector diff en tamaño\n") ;
   vector u(nr) ;
   for (int i = 0; i < nr; i++)
      u[i] = scalar (*m.mat[i], v) ;
   return u ;
}

vector operator*(const vector &v, const matrix &m)
{
   int nr = m.numrows, nc = m.numcols ;
   if ( v.size != nr )  fatal ("diferentes tamaños entre vector y matriz\n") ;
   vector u(nc) ;
```

```
  for (int i = 0; i < nc; ++i) {
     double t = 0.0 ;
          for (int j = 0; j < nr; j++)
             t += v.vec[j] * m.mat[j]->vec[i] ;
     u.vec[i] = t ;
  }
  return u ;
}

matrix operator*(vector &u, vector &v)
{
  int nr = u.size, nc = v.size ;
  matrix m(nr, nc) ;
  for (int i = 0; i < nr; i++) {
     for (int j = 0; j < nc; j++ )
          m.mat[i]->vec[j] = u[i] * v[j] ;
  }
  return m ;
}

matrix operator*(const matrix &m1, const matrix &m2)
{
  if ( m1.numcols != m2.numrows)  fatal ("no se puede multiplicar a las matrices\n") ;
  matrix m(m1.numrows, m2.numcols) ;
  for (int i = 0; i < m1.numrows; i++ )
     for (int j = 0; j < m2.numcols; j++) {
             m.mat[i]->vec[j] = 0.0 ;
             for (int k = 0; k < m1.numcols; k++ )
                m.mat[i]->vec[j] += m1.mat[i]->vec[k] * m2.mat[k]->vec[j] ;
     }
  return m ;
}

matrix operator*(const matrix &m, double c)
{
  matrix m1 = m ;
  return (m1 *= c) ;
}

double norm(const matrix &m)
{
  double t = 0.0 ;
  for (int i = 0; i < m.numrows; i++) {
     for (int j = 0; j < m.numcols; j++ ) {
        double mij = m.mat[i]->vec[j] ;
        t += mij * mij ;
      }
  }
  return sqrt(t) ;
}
int insmat (matrix& r, matrix& v, int m, int n)
{
  int i, j, lim, l ;
  lim = r.getsize() ;
  l  = v.getsize() ;
  if ( (m + l) > lim || (n + l) > lim ) {
     cout << "Error al insertar mat[" << l << "][" << l << "]" ;
     cout << " se exeden los limites\n" ;
     return (0) ;
  }
  for ( i = m; i < m + l; i++ )
     for (j = n; j < n + l; j++)
```

```
            r[i][j] = v[i - m][j - n] ;
    return (1) ;
}

int insmat (matrix& r, int m, int n, matrix& v, int p, int q, int l)
{
    int i, j, lim ;
    lim = r.getsize() ;
    if ( (m + l) > lim || (n + l) > lim ) {
        cout << "Error en destino: mat[" << lim << "][" << lim << "]" ;
        cout << " se exeden sus limites\n" ;
        return (0) ;
    }
    lim = v.getsize() ;
    if ( (p + l) > lim || (q + l) > lim ) {
        cout << "Error en fuente: mat[" << lim << "][" << lim << "]" ;
        cout << " se exeden sus limites\n" ;
        return (0) ;
    }
    for ( i = m; i < m + l; i++ )
        for (j = n; j < n + l; j++)
            r[i][j] = v[i - m + p][j - n + q] ;
    return (1) ;
}

double norminf(const matrix &m)
{
    double t = 0.0 ;
    for (int i = 0; i < m.numrows; i++ ) {
        double s = 0 ;
            for (int j = 0; j < m.numcols; j++)
                s += fabs(m.mat[i]->vec[j]) ;
        if ( s > t )
                t = s ;
    }
    return t ;
}

ostream& operator<<(ostream &s, const matrix &m)
{
    for (int i = 0; i < m.numrows; ++i) {
        for (int j = 0; j < m.numcols; j++)
            s << setprecision(4) << setiosflags(ios::fixed) << m.mat[i]->vec[j] << " " ;
        s << "\n" ;
    }
    return s ;
}

istream& operator>>(istream &s, matrix &m)
{
    int nr = m.numrows ;
    cout << "teclee los " << nr << " renglones de la matriz\n" ;
    for (int i = 0; i < nr; i++) {
        for (int j = 0; j < m.numcols; j++) {
            cout << "m[" << i << "][" << j << "] = " ;
            s >> m.mat[i]->vec[j] ;
        }
        cout << "\n" ;
    }
    return s ;
}
```

```
int join (matrix &a, matrix& b, matrix& mat)
{
   int n = a.getsize() ;
   int m = mat.getsize() ;
   if ( m == 2 * n) {
      for (int i = 0 ; i < n; i++) {
         for (int j = 0; j < n; j++)  {
               mat[i + n][j + n] = mat[i][j]  = a[i][j] ;
               mat[i][j + n] = -b[i][j] ;
               mat[i + n][j] = b[i][j] ;
         }
      }
      return (0) ;
   }
   return (-1) ;
}

/*******************************************************
*  Archivo: eigen.h                                    *
*  Definicion de varianles para eigen.cpp              *
*******************************************************/
#include  <math.h>
#include  "vecmat.h"
#ifndef   M_PI
#define   M_PI       3.14159265358979323846
#endif
#define   MAXITER    50

inline sign (double a)
{
         return a >= 0.0 ? 1 : - 1 ;
}
void   house (matrix&, vector&, vector&) ;
void   francis(vector&, vector&, matrix&) ;

inline GetEigen (matrix& C, vector& d, vector& e)
{
   house (C, d, e) ;
   francis (d, e, C) ;
}

/*****************************************************************************
*   Programa eigen.cpp                                                       *
*   Programa que diagonaliza una matriz,                                     *
*   Los vectores propios quedan en las columnas de la matriz                 *
*   Los valores propios quedan en un arreglo                                 *
*****************************************************************************/
#include <stdlib.h>
#include "eigen.h"

inline void fatal (const char *str)
{
   cerr << str ;   exit (1) ;
}

void house (matrix &a, vector &d, vector &e)
{
   int    n = a.getsize() ;
   int    i, j, k, l ;
   double  scale, hh, h, g, f ;
   for (i = n - 1; i >= 1; i--) {
      l = i - 1 ;
```

```
      h = scale = 0.0 ;
      if (l > 0 ) {
         for (k = 0; k <= l; k++)
            scale += fabs(a[i][k]) ;
         if ( scale == 0.0)
            e[i] = a[i][l] ;
         else {
            for (k = 0; k <= l; k++) {
               a[i][k] /= scale ;
               h += a[i][k] * a[i][k] ;
            }
            f = a[i][l] ;
            g = ( f >= 0.0 ? -sqrt(h) : sqrt(h)) ;
            e[i] = scale * g ;
            h -= f * g ;
            a[i][l] = f - g ;
            f = 0.0 ;
            for (j = 0; j <= l; j++) {
               a[j][i] = a[i][j] / h ;
               g = 0.0 ;
               for (k = 0; k <= j; k++)
                  g += a[j][k] * a[i][k] ;
               for (k = j + 1 ; k <= l; k++)
                  g += a[k][j] * a[i][k] ;
               e[j] = g / h ;
               f += e[j] * a[i][j] ;
            }
            hh = f / (h + h) ;
            for (j = 0; j <= l; j++) {
               f = a[i][j] ;
               e[j] = g = e[j] - hh * f ;
               for (k = 0; k <= j; k++ )
                  a[j][k] -= (f * e[k] + g * a[i][k]) ;
            }
         }
      } else
          e[i] = a[i][l] ;
       d[i] = h ;
   }
   d[0] = 0.0 ;
   e[0] = 0.0 ;
   for ( i = 0; i < n; i++) {
      l = i - 1 ;
      if ( d[i] ) {
         for (j = 0; j <= l; j++) {
            g = 0.0 ;
            for (k = 0; k <= l; k++)
               g += a[i][k] * a[k][j] ;
            for (k = 0; k <= l; k++)
               a[k][j] -= g * a[k][i] ;
         }
      }
      d[i] = a[i][i] ;
      a[i][i] = 1.0 ;
      for (j = 0; j <= l; j++) a[j][i] = a[i][j] = 0.0 ;
   }
}

void francis (vector& d, vector& e, matrix& a)
{
   int    n = a.getsize() ;
   int    l, m, iter, i, k ;
```

```
  double  s, r, p, g, f, dd, c, b ;
  for (i = 1; i < n; i++) e[i - 1] = e[i] ;
  e[n - 1] = 0.0 ;
  for (l = 0; l < n; l++) {
     iter = 0 ;
     do {
        for (m = l; m < n - 1; m++) {
           dd = fabs (d[m]) + fabs(d[m + 1]) ;
           if ( fabs(e[m]) + dd == dd ) break ;
        }
        if ( m != l ) {
           if ( iter++ == MAXITER ) fatal ("No hay convergencia\n") ;
           g = ( d[l + 1] - d[l] ) / (2.0 * e[l]) ;
           r = sqrt (g * g + 1.0) ;
           g = d[m] - d[l] + e[l] / ( g + r * sign(g)) ;
           s = c = 1.0 ;
           p = 0.0 ;
           for ( i = m - 1; i >= l; i--) {
              f = s * e[i] ;
              b = c * e[i] ;
              e[i + 1] = ( r = sqrt( f * f + g * g)) ;
              if ( r == 0.0 ) {
                 d[i + 1] -= p ;
                 e[m] = 0.0 ;
                 break ;
              }
              s = f / r ;
              c = g / r ;
              g = d[i + 1] - p ;
              r = ( d[i] - g ) * s + 2.0 * c * b ;
              d[i + 1] = g + (p = s * r) ;
              g = c * r - b ;
              for (k = 0; k < n; k++) {
                 f = a[k][i + 1] ;
                 a[k][i + 1] = s * a[k][i] + c * f ;
                 a[k][i] = c * a[k][i] - s * f ;
              }
           }
           if ( r == 0.0 && i >= l ) continue ;
           d[l] -= p ;
           e[l]  = g ;
           e[m] = 0.0 ;
        }
     } while ( m != l ) ;
  }
  a = a.transpose() ;
}

/*****************************************************************
 *    Archivo: fss2s.h                                           *
 *    Definicion de constantes y variables para                  *
 *    fcc2s.cpp                                                  *
 *****************************************************************/
#include   <fstream.h>
#include   <math.h>
#include   <stdlib.h>
#include   "..\\matrix\\eigen.h"
#define    DIM   10     // Dimension de la matriz de Interaccion
struct _E {             // Densidad de Estados
   double n ;
   double e ;
} ;
```

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

```
double    GetData   (const char*, double*, double*) ;
void      GetEnerB  (const char*, double, double*, double*) ;
void      GetDos    (const char*, double, double*, double*) ;
void      GetMatInt (double, double, double, double*, matrix&) ;
int       GetR      (matrix&, double) ;
void      GetMatHam (vector&, double*, double*, matrix&, matrix&) ;
void      WriteEner (ofstream&, vector&, double) ;
void      WriteDos  (ofstream&, vector&, matrix&) ;

inline void fatal (const char *str)
{
  cerr << str ;  exit (1) ;
}

/*
 *   Archivo: fss2s.cpp
 *   Esta funcion la constante del cristal, y sus energias.
 *   a, Ess, Epx, Epy, Epz, Ess, Epx, Epy, Eps, Vss, Vsp, Vps, Vpp, Vppi
 */
#include "fcc2s.h"
double GetData (const char *path, double *OnSiteEner, double *V)
{
  double   a, sr3 = sqrt (3.0) ;
  double   Vss, Vxx, Vxy, Vsp, Vs2p ;
  ifstream  inpdata(path) ;
  if ( ! inpdata )  fatal ("No puedo leer los parametros del crsital\n") ;
  inpdata >> a ;   // *** Parametro de la red cristalina
  for ( int i = 0 ; i < DIM ; i++)   // *** On-site energies  Es, Ep, Es*, Es, Ep, Es*
     inpdata >> OnSiteEner[i] ;
  inpdata >> Vss  ;            // V(s,s)
  inpdata >> Vxx  ;            // V(x,x)
  inpdata >> Vxy  ;            // V(x,y)
  inpdata >> Vsp  ;            // V(s,p)
  inpdata >> Vs2p ;            // V(s*,p)
  V[0] = 0.25 * Vss ;
  V[1] = Vsp * sr3 * 0.25 ;  V[2] = -V[1] ;
  V[3] = (Vxx + 2 * Vxy) * 0.25 ;   // Vpps
  V[4] = (Vxx - Vxy) * 0.25 ;       // Vppi
  V[5] = 0.0 ;                  // Vs*s*
  V[6] = Vs2p * sr3 * 0.25 ;
  V[7] = -V[6] ;
  inpdata.close() ;
  return (a) ;
}
/*
 *  Funcion que obtiene en un arreglo los vecinos mas proximos
 *  En el algoritmo hacemos a / 4 = 1.  x, y, z : Vectores base del cristal FCC
 *  tao: Lista las posiciones de los dos atomos de la celda unitaria
 */
int GetR (matrix& R, double a)
{
  int    nR = 0 ;
  double  sr3 = sqrt(3.0) + 0.01 ;
  vector  x(3), y(3), z(3) ;
  matrix  tao(2, 3) ;
  x[0] = x[1] = 2.0 ; x[2] = 0 ;   // *** Vectores base del cristal FCC
  y[0] = 0.0 ; y[1] = y[2] = 2.0 ;
  z[0] = 2.0 ; z[1] = 0.0 ; z[2] = 2.0 ;
  tao[0][0] = tao[0][1] = tao[0][2] = 0.0 ; // *** Dos atomos por celda unitaria: (0, 0, 0) y (a/4, a/4, a/4)
  tao[1][0] = tao[1][1] = tao[1][2] = 1.0 ;
  for (int l = -2; l < 3; l++) {
```

```
      for (int m = -2; m < 3; m++) {
         for (int n = -2; n < 3; n++) {
            for (int h = 0; h < 2; h++) {
               vector r = l * x + m * y + n * z + tao[h] ;
               if ( norm (r) < sr3 )
                  R[nR++] = r * a / 4.0 ;   // Vecino mas proximo
            }
         }
      }
   }
   return nR ;
}
/*
 *  Generamos la matriz de Interaccion entre los atomos vecinos
 */
void GetMatInt (double al, double am, double an, double *V, matrix& MatInt)
{
      MatInt[0][0] = V[0] ;
      MatInt[0][1] = V[1] * al ;
      MatInt[0][2] = V[1] * am ;
      MatInt[0][3] = V[1] * an ;
      MatInt[0][4] = 0.0 ;        // SS2
      MatInt[1][1] = al * al * V[3] + (1.0 - al * al) * V[4] ;
      MatInt[1][2] = al * am * (V[3] - V[4]) ;
      MatInt[1][3] = al * an * (V[3] - V[4]) ;
      MatInt[1][4] = V[7] * al ;   // PS2 * al
      MatInt[2][2] = am * am * V[3] + (1.0 - am * am) * V[4] ;
      MatInt[2][3] = am * an * (V[3] - V[4]) ;
      MatInt[2][4] = V[7] * am ;
      MatInt[3][3] = an * an * V[3] + (1.0 - an * an) * V[4] ;
      MatInt[3][4] = V[7] * an ;
      MatInt[4][4] = V[5] ;
      MatInt[3][0] = V[2] * an ;
      MatInt[2][0] = V[2] * am ;
      MatInt[1][0] = V[2] * al ;
      MatInt[2][1] = MatInt[1][2] ;
      MatInt[3][1] = MatInt[1][3] ;
      MatInt[3][2] = MatInt[2][3] ;
      MatInt[4][1] = V[6] * al ;          // S2P * al
      MatInt[4][2] = V[6] * am ;          // S2P * am
      MatInt[4][3] = V[6] * an ;          // S2P * an
}
/*
 *    Generamos a la matriz Hamiltoniana del cristal
 *    HR: Parte real del Hamiltoniano. HI: Parte imaginaria del Hamiltoniano
 *     R: Es una lista de los vectores de posicion de los vecinos mas proximos, incluyenod al (0, 0, 0).
 */
void GetMatHam (vector& k, double *OnSiteEner, double *V, matrix& R, matrix& C)
{
   int   nR = DIM / 2 ;
   matrix HR(DIM, DIM), HI(DIM, DIM), MatInt(nR, nR) ;
   for (int i = 0; i < nR; i++) {
      for (int j = 0; j < nR; j++) {
         for (int m = 0; m <= 4; m++) {
            if ( i == j ) {
               HR[i][j] = OnSiteEner[i] ;
               HR[i+nR][j+nR] = OnSiteEner[i+nR] ;
            }
            if ( norm (R[m]) > 0 ) {
               double phase = scalar (k, R[m] ) ;
               double r = norm (R[m]) ;
               double cosine = cos (phase) ;
```

```
                double sine   = sin (phase) ;
                double al = R[m][0] / r ;
                double am = R[m][1] / r ;
                double an = R[m][2] / r ;
                GetMatInt (al, am, an, V, MatInt) ;
                HR[i][j + nR] += MatInt[i][j] * cosine ;
                HR[j + nR][i] = HR[i][j+nR] ;
                HI[i][j+nR] += MatInt[i][j] * sine ;
                HI[j + nR][i] = -HI[i][j+nR] ;
              }
            }
          }
        }
        join (HR, HI, C) ;     // Generamos martiz compleja
}

void WriteEner (ofstream& outdata, vector& d, double pt)
{
   int    nv = 0, i, j, esta = 0 ;
   int    dim = d.getsize() ;
   vector  vp(DIM) ;
   for (i = 0; i < dim; i++) {    // *** Unicamente valores propios diferentes  ***
      for (j = 0 ; j < nv; j++ ) {
         if ( fabs (vp[j] - d[i]) < 0.00000001 ) {
            esta = 1 ;
            break ;
          }
      }
      if ( ! esta ) vp[nv++] = d[i] ;
      esta = 0 ;
   }
   for (i = 0; i < nv; i++ )    //  *** Escribimos energias de salida  ***
      outdata << pt << " " << vp[i] << "\n" ;
}

void WriteDos(ofstream& dos, vector& d, matrix& C)
{
   int       i, j, esta = 0, nv = 0, total = 0 ;
   int       dim = DIM * 2 ;
   struct _E  den[1024] ;
   vector    vp(DIM) ;
   dos.setf (ios::fixed) ;
   dos.precision(2) ;
   for (i = 0; i < dim; i++) {    // *** Unicamente valores propios diferentes  ***
      for (j = 0 ; j < nv; j++ ) {
         if ( fabs (vp[j] - d[i]) < 0.000001 ) {
            den[j].n += norm (C[i]) ;
            esta = 1 ;
            break ;
          }
      }
      if ( ! esta )  {
         den[nv].e = vp[nv] = d[i] ;
         den[nv++].n = norm (C[i]) ;
         total++ ;
      }
      esta = 0 ;
   }
   for (i = 0; i < total; i++)
      dos << den[i].e << " " << den[i].n / 2.0 << "\n" ;
}
```

```
void main(int argc, char **argv)
{
   double  OnSiteEner [DIM] ;  // Es, Epx, Epy, Epz, Es, Epx, Epy, Epz
   double  V[DIM] ;            // Vss, Vsp, Vps, Vpps, Vppi, Vs*s*, Vs*p, Vps*
   double  a ;                 // Constante del cristal
   if ( argc != 3 )  fatal ("Usar: diamante Entrada Salida\n") ;
   if ( !strcmp (argv[1], argv[2]) )  fatal ("El archivo de entrada deber ser diferente al de salida\n\n") ;
   a = GetData (argv[1], OnSiteEner, V) ;   // *** Leemos energias y constante del cristal
   GetEnerB (argv[2], a, OnSiteEner, V) ;
}

void GetEnerB (const char* path, double a, double *OnSiteEner, double *V)
{
   int      dim = 2 * DIM ;
   double   limit, inc, pt = 0 ;
   vector   k(3), d(dim), e(dim), vp(DIM) ;
   matrix   R(5, 3), C (dim, dim) ;
   ofstream  outdata ;           // Datos de salida
   outdata.open (path) ;
   if ( !outdata )  fatal ("No se puede abrir archivo de datos de salida\n") ;
   GetR (R, a) ;
   cout << "Calculando estructura electrónica\n" ;   // *** L - GAMA ***
   limit = M_PI / a ;
   inc = limit / 20.0 ;
   for (k[1] = k[2] = k[0] = limit; k[0] > 0.0; k[0] -= inc ) {
      k[2] = k[1] = k[0] ;
      GetMatHam (k, OnSiteEner, V, R, C) ;
      GetEigen (C, d, e) ;
      WriteEner (outdata, d, pt) ;
      pt += 0.10 ;
   }
   cout.flush() << "--> L - GAMA" ;
   limit = 2 * M_PI / a ;   // *** GAMA - X ***
   inc = limit / 20.0 ;
   k[1] = k[2] = 0.0 ;
   for (k[0] = 0.0; k[0] < limit; k[0] += inc ) {
      GetMatHam (k, OnSiteEner, V, R, C) ;
      GetEigen (C, d, e) ;
      WriteEner (outdata, d, pt) ;
      pt += 0.10 ;
   }
   cout.flush() << " - X" ;
   k[0] = limit ;   // *** X - W ***
   k[2] = 0.0 ;
   limit =  M_PI / a ;
   inc = limit / 2.0 ;
   for ( k[1] = 0.0 ; k[1] < limit; k[1] += inc ) {
      GetMatHam (k, OnSiteEner, V, R, C) ;
      GetEigen (C, d, e) ;
      WriteEner (outdata, d, pt) ;
      pt += 0.10 ;
   }
   cout.flush() << " - W" ;
   k[0] = 2.0 * M_PI / a ;   // *** W - K ***
   k[1] = limit ;
   k[2] = 0.0 ;
   limit = 1.5 * M_PI / a ;        // 3pi entre 2a
   inc = (0.5 * M_PI) / a ;
   inc /= 2.0 ;
   for ( ; k[0] > limit; k[0] -= inc ) {
      k[1] = -k[0] + 3 * M_PI / a ;
```

```
      GetMatHam (k, OnSiteEner, V, R, C) ;
      GetEigen (C, d, e) ;
      WriteEner (outdata, d, pt) ;
      pt += 0.10 ;
    }
    cout.flush() << " - K" ;
    k[0] = k[1] = 1.5 * M_PI / a ;   // *** K - GAMA ***
    inc = k[0] / 20.0 ;
    limit = 0.0 ;
    for ( ; k[0] >= 0.0; k[0] -= inc ) {
      k[1] = k[0] ;
      GetMatHam(k, OnSiteEner, V, R, C) ;
      GetEigen (C, d, e) ;
      WriteEner (outdata, d, pt) ;
      pt += 0.10 ;
    }
    cout << " - GAMA <--\n" ;
}

/*********************************************************************
*   Archivo surf.h                                                   *
*   Definicion de variables y constantes para surf2x1.cpp            *
*********************************************************************/
#include   <fstream.h>
#include   <math.h>
#include   <stdlib.h>
#include   "eigen.h"
#define    BASE      5    // Dimension de las sub-matrices Hamiltonianas
#define    NLAYER    24
#define    DELTA     0.000000000001    // ¡¡ Cero !!
struct _E {
   int   nv ;
   int   *m ;
   double *d ;
} ;
double GetData   (const char*, double*,   double*) ;
void   GetEnerB  (double,  double*, double*) ;
void   GetAtomPos (vector&, matrix&) ;
void   GetDos    (const char*, double, double*, double*) ;
void   GetMatInt (double,  double,  double,  double*, matrix&) ;
void   GetMatInt (double,  double,  double,  double,  double*, matrix&) ;
void   GetMatHRI (const int, vector&, matrix&, matrix&, matrix&, matrix&,  double*) ;
void   GetMatHam  (double*, double*, vector&, matrix&, matrix&) ;
void   GetBulk2x1 (matrix&) ;
void   WriteEner  (int, matrix&, vector&, int) ;
void   WriteEigenV (matrix&, struct _E&, int, int) ;
void   WrDosLayer (int, double, vector&) ;
void   ordena     (vector&, int, int) ;

inline void fatal (const char *str)
{
   cerr << str ;  exit (1) ;
}

/*********************************************************************
*     Archivo: hamil2x1.cpp                                          *
*     Rutinas que construyen el hamiltoniano del cristal             *
*     Ge(111)-2x1                                                    *
*********************************************************************/
#include   <stdio.h>
#include   "surf.h"
extern ofstream  fBulk, fSurf, fDos[4], fEigen ;
```

```
extern double   a, EFermi ;
/*
 *  Generamos la matriz de Interaccion entre los atomos vecinos
 */
void GetMatInt (double r, double al, double am, double an, double *V, matrix& MatInt)
{
      const double h = (3.0 / 16.0) / (r * r) ;
      MatInt[0][0] = V[0] ;
      MatInt[0][1] = V[1] * al ;
      MatInt[0][2] = V[1] * am ;
      MatInt[0][3] = V[1] * an ;
      MatInt[0][4] = 0.0 ;          // SS2
      MatInt[1][1] = al * al * V[3] + (1.0 - al * al) * V[4] ;
      MatInt[1][2] = al * am * (V[3] - V[4]) ;
      MatInt[1][3] = al * an * (V[3] - V[4]) ;
      MatInt[1][4] = V[7] * al ;    // PS2 * al
      MatInt[2][2] = am * am * V[3] + (1.0 - am * am) * V[4] ;
      MatInt[2][3] = am * an * (V[3] - V[4]) ;
      MatInt[2][4] = V[7] * am ;
      MatInt[3][3] = an * an * V[3] + (1.0 - an * an) * V[4] ;
      MatInt[3][4] = V[7] * an ;
      MatInt[4][4] = V[5] ;           // S*S*
      MatInt[3][0] = V[2] * an ;
      MatInt[2][0] = V[2] * am ;
      MatInt[1][0] = V[2] * al ;
      MatInt[2][1] = MatInt[1][2] ;
      MatInt[3][1] = MatInt[1][3] ;
      MatInt[3][2] = MatInt[2][3] ;
      MatInt[4][0] = 0.0 ;
      MatInt[4][1] = V[6] * al ;          // S2P * al
      MatInt[4][2] = V[6] * am ;          // S2P * am
      MatInt[4][3] = V[6] * an ;          // S2P * an
      MatInt *= h ;
}

/*
 *    Generamos a la matriz Hamiltoniana del cristal
 *    HR: Parte real del Hamiltoniano. HI: Parte imaginaria del Hamiltoniano
 *     R: Es una lista de los vectores de posicion de los vecinos mas proximos, incluyenod al (0, 0, 0).
 */
void GetMatHRI (const int nNb, vector& q, matrix& Pos, matrix& R1,  matrix& HR, matrix& HI, double* V)
{
   int    i, j, m ;
   double  phase, r, cosine, sine, al, am, an ;
   matrix  MatInt(BASE) ;
   for (m = 0; m < nNb; m++) {
      r     = norm (Pos[m]) ;
      phase  = scalar (q, R1[m]) ;
      cosine = cos (phase) ;
      sine   = sin (phase) ;
      al    = Pos[m][0] / r ;
      am    = Pos[m][1] / r ;
      an    = Pos[m][2] / r ;
      for (i = 0; i < BASE; i++) {
         for (j = 0; j < BASE; j++) {
            GetMatInt (r, al, am, an, V, MatInt) ;
            HR[i][j] += MatInt[i][j] * cosine ;
            HI[i][j] += MatInt[i][j] * sine ;
         }
      }
   }
}
```

```
void GetMatHam (double* V, double* OnSiteEner, vector& q,  matrix& tao, matrix& C)
{
  int    i, j, k, l, m, n, n1 = 0 ;
  int    nBase = 2 * NLAYER, nBulk = 2 * BASE * NLAYER ;
  vector Rx(3), Ry(3), R(3), v(3) ;
  matrix HR1(BASE), HI1(BASE), R1(BASE, 3), Pos(BASE, 3) ;
  matrix HRT(BASE), HIT(BASE) ;
  matrix HR(nBulk), HI(nBulk), MatInt(BASE) ;
  double xcut = 0.5 ;
  Rx[0] = sqrt(3.0 / 2.0) ; Rx[1] = Rx[2] = 0.0 ;
  Ry[1] = sqrt(1.0 / 2.0) ; Ry[0] = Ry[2] = 0.0 ;
  for (k = 0; k <= nBase; k++) {
    for (l = k; l <= k + 3; l++) {
      if ( l == nBase ) break ;
      for ( m = -1; m <= 1; m++) {
        for ( n = -1; n <= 1; n++) {
          R = m * Rx + n * Ry ;
          v = tao[l] - tao[k] + R ;
          if ( norm (v) <= xcut ) {
            R1[n1] = R ;
              Pos[n1++] = v ;
          }
        }
      }
      if ( k == l ) {
        for (i = 0; i < BASE; i++ ) HR1[i][i] = OnSiteEner[i] ;
        insmat (HR, k * BASE, k * BASE, HR1, 0, 0, 5) ;
      } else if ( n1 > 0 ) {
        GetMatHRI (n1, q, Pos, R1, HR1, HI1, V) ;
        insmat (HR, k * BASE, l * BASE, HR1, 0, 0, 5) ;
        insmat (HI, k * BASE, l * BASE, HI1, 0, 0, 5) ;
        HRT = HR1.transpose() ;
        insmat (HR, l * BASE, k * BASE, HRT, 0, 0, 5) ;
        HIT = -1.0 * HI1.transpose() ;
        insmat (HI, l * BASE, k * BASE, HIT, 0, 0, 5) ;
      }
      n1 = 0 ;
      for (i = 0; i < BASE; i++)
        for (j = 0; j < BASE; j++)
          HR1[i][j] = HI1[i][j] = 0.0 ;
    }
  }
  join (HR, HI, C) ;     // Generamos martiz compleja
}

void WriteEner (int bDos, matrix& C, vector& d, int pt)
{
  int    i, j, k, nv = 0, esta = 0, nL = 2 * NLAYER ;
  double xcut ;
  int    dim = d.getsize(), *idx = new int[dim] ;
  matrix Re(12, 5), Im(12, 5) ;
  struct _E ener ;
  ener.d = new double[dim] ;
  ener.m = new int[dim] ;
  ener.nv = 0 ;
  for (i = 0; i < dim; i++) {
    for (j = 0 ; j < ener.nv; j++ ) {
      if ( fabs (ener.d[j] - d[i]) < 0.000001 ) {
        esta = 1 ;
        ener.m[j]++ ; break ;
      }
    }
```

```
      if ( ! esta ) {
         ener.d[ener.nv]   = d[i] ;
         ener.m[ener.nv] = 1 ;
         idx[ener.nv++] = i ;
      }
      esta = 0 ;
   }
   k = 0 ;   // Densidad de Estados Total y por capa 1, 2, y 3
   for ( i = 0; i < ener.nv; i++) {
      if ( i != idx[i] ) C[i] = C[idx[i]] ;
      if ( bDos ) {
         for (j = 0; j < ener.m[i] / 2; j++)
            d[k + j] = ener.d[i] ;
         k += j ;
         fDos[0] << ener.d[i] << " " << 0.5 * ener.m[i] << "\n" ;
         WrDosLayer (ener.m[i], ener.d[i], C[i]) ;
      }
   }
   ordena (d, 0, k - 1) ;
   EFermi += ((d[95] + d[96]) / 2.0) ;
   for (i = 0; i < ener.nv; i++ ) {
      xcut = 0.0 ;
      for (j = 0; j < 12; j++) {            // Celda 2x1
         for (k = 0; k < 5; k++) {
            Re[j][k] = (j < 6 ) ? C[i][k + 5 * j] : C[i][k + 5 * (j + 36)] ;
            Im[j][k] = (j < 6 ) ? C[i][k + 5 * (j + nL)] :
                                  C[i][k + 5 * (j + 36 + nL)] ;
         }
         xcut += (norm (Re[j]) + norm (Im[j])) ;
      }
      if ( xcut >= 2.5 ) {
         fSurf << pt << " " << ener.d[i] << "\n" ;
         if ( ener.d[i] <= 2.0 ) {
            if (pt == 1 || pt == 20 || pt == 32 || pt == 50 || pt == 82)
               WriteEigenV (C, ener, i, pt) ;
         }
      } else   fBulk << pt << " " << ener.d[i] << "\n" ;
   }
}
 void WriteEigenV (matrix& C, struct _E& ener, int i, int pt)
{
   int   j, k, nDim, nc = 1, na = 1, l = 0 ;
   char  *msg[4] = {"Gama", "J", "K", "j\'"} ;
   vector Re(5), Im(5) ;
   nDim = C.getsize() ;
   switch (pt) {
      case 0:
      case 1:
      case 20:  l = 0 ;  break;
      case 32: l = 1 ;   break ;
      case 50: l = 2 ;   break ;
      case 82: l = 3 ;   break ;
   }
   fEigen << "Energia: " << ener.d[i] << ": " << msg[l] << "(" << pt << ")\n" ;
   for (j = 0; j <= 235; j += 5) {
      if ( j > 35 && j < 200) continue ;
      if ( j == 200 ) {
         nc = 21 ;
         na = 41 ;
      }
      if ( ! (j % 10) )
         fEigen << "Capa: " << nc++ << "\n" ;
```

```
      for (k = 0; k < 5; k++) {
         Re[k] = C[i][j + k] ;
         Im[k] = C[i][j + k + 240] ;
      }
      fEigen << "Atomo: " << na++ << "\n" << Re << " + I" << Im << " : " ;
      fEigen << norm(Re) + norm(Im) << "\n" ;
   }
}

void WrDosLayer (int m, double d, vector& C)
{
   int   j, k ;
   double norma = 0 ;
   vector Re(10), Im(10) ;
   for (j = 0; j < 30; j += 10) {
      for (k = 0; k < 10; k++) {
         Re[k] = C[j + k] ;
         Im[k] = C[j + k + 240] ;
         norma = norm(Re) + norm(Im) ;
      }
      norma = norm(Re) + norm(Im) ;
      fDos[j / 10 + 1] << d << " " <<  0.5 * norma * m << "\n" ;
   }
}

void ordena (vector& item, int izq, int der)   // Ordenamos Energias para calcular el nivel de Fermi  */
{
   int   i, j ;
   double x, y ;
   i = izq ;  j = der ;
   x = item[ (izq + der) / 2] ;
   do {
      while (item[i] < x && i < der) i++ ;
      while (x < item[j] && j > izq) j-- ;
      if ( i <= j ) {
         y = item[i] ;
         item[i] = item[j] ;
         item[j] = y ;
         i++ ; j-- ;
      }
   } while ( i <= j ) ;
   if ( izq < j )  ordena (item, izq, j) ;
   if ( i < der )  ordena (item, i, der) ;
}

/*****************************************************************
 *     Archivo: surf2x1.cpp                                      *
 *     Programa que calcula la estructura electrónica de la      *
 *     superficie del Ge(111)-2x1  reconstruida                  *
 *****************************************************************/
#include      <string.h>
#include      <stdio.h>
#include    "surf.h"
int      bDos ;
char     strDos[80] ;
double   a, EFermi = 0 ;
ofstream  fBulk, fSurf, fEigen ;  // Estados de Bulto y Superficie
ofstream  fDos[4] ;               // 0:Total, 1:L1, 2:L2, 3:L3

void main(int argc, char** argv)
{
   double  OnSiteEner[BASE * 2] ;  // Es, Epx, Epy, Epz, Es, Epx, Epy, Epz
```

```
   double  V[BASE * 2] ;  // Vss, Vsp, Vps, Vpps, Vppi, Vs*s*, Vs*p, Vps*
   if ( argc <= 2 )  fatal ("Usar: surf2x1 Energies EBulk ESurf [EDos]\n") ;
   if ( argc == 5 ) {
      bDos = 1 ;
      strcpy (strDos, argv[4]) ;
   }
   fBulk.open (argv[2]) ;
   if ( !fBulk )  fatal ("No se puede el abrir archivo de estados del bulto\n") ;
   fSurf.open (argv[3]) ;
   if ( !fSurf )  fatal ("No se puede abrir el archivo de estados de superficie\n") ;
   fEigen.open ("Eigenv.dat") ;
   if ( !fEigen )  fatal ("No se puede abrir el archivo de vectores propios\n") ;
   if ( !strcmp (argv[1], argv[2]) )  fatal ("El archivo de entrada deber ser diferente al de salida\n\n") ;
   a = GetData (argv[1], OnSiteEner, V) ;   // *** Leemos energias y constante del cristal
   GetEnerB (a, OnSiteEner, V) ;
}

// Funcion que calcula los estados de energia tanto del bulto como de la superficie del cristal
// en el plano (111).
//  En fBulk guardamos los estados de Bulto.  En fSurf guradamos los estados de Superficie.
void GetEnerB (double a, double* OnSiteEner, double* V)
{
   int      dim = 2 * BASE * NLAYER, i, *mult = new int[2 * dim] ;  // Dos átomos por celda
   double   qx0, qy0, dqx, dqy ;
   char     str[80] ;
   ofstream fBZ ;
   vector   q(3), d(2 * dim), e(2 * dim), vp(2 * dim), v(3) ;
   matrix   tao(2 * NLAYER, 3), C(2 * dim, 2 * dim), tao2(10, 3) ;
   if ( bDos ) {
      fDos[0].open (strDos) ;
      strDos[strlen(strDos) - 4] = '\0' ;
      for (i = 1; i < 4; i++) {
         sprintf (str, "%sL%1d.dat", strDos, i) ;
         fDos[i].open (str) ;
         if ( !fDos[i] )  fatal ("No puedo generar archivo densidad de estados\n") ;
      }
   }
   cout << "\n\tInicia cálculo de Estados de Superficie Electrónica\n" ;
   cout << "\n\tCalculando las posiciones atómicas del Bulto Ge(111)\n" ;
   GetBulk2x1 (tao) ;
   /* Psociones de los atomos después de la  recosntrucción */
   /* Valores dados por Takeuchi  */
   tao2[0][0] = -1.817515 ; tao2[0][1] = 0.0  ; tao2[0][2] =  0.081413 ;
   tao2[1][0] = -0.793638 ; tao2[1][1] = 0.353553 ; tao2[1][2] = -0.041568 ;
   tao2[2][0] = -1.459902 ; tao2[2][1] = 0.0      ; tao2[2][2] = -0.178398 ;
   tao2[3][0] = -1.206373 ; tao2[3][1] = 0.353553 ; tao2[3][2] = -0.162757 ;
   tao2[4][0] = -1.612987 ; tao2[4][1] = 0.0      ; tao2[4][2] = -0.588889 ;
   tao2[5][0] = -1.036138 ; tao2[5][1] = 0.353553 ; tao2[5][2] = -0.575634 ;
   tao2[6][0] = -0.815685 ; tao2[6][1] = 0.0      ; tao2[6][2] = -0.701474 ;
   tao2[7][0] = -1.430505 ; tao2[7][1] = 0.353553 ; tao2[7][2] = -0.753436 ;
   tao2[8][0] = -0.815685 ; tao2[8][1] = 0.0      ; tao2[8][2] = -1.146615 ;
   tao2[9][0] = -1.428873 ; tao2[9][1] = 0.353553 ; tao2[9][2] = -1.174323 ;
   for (i = 0; i <= 9; i++) {
      v = tao2[i] - tao[i] ;
      tao[i] = tao2[i] ;
      tao[2 * NLAYER - i - 1] = tao[2 * NLAYER - i - 1] -v ;
   }
   ofstream fTao ;
   fTao.open ("Ge2x1r.dat") ;
   for (i = 0; i < 2 * NLAYER; i++)
      fTao << tao[i] << "\n" ;
   fTao.close() ;
```

ESTA TESIS NO DEBE SALIR DE LA BIBLIOTECA

```
  fBZ.open ("SBZone.dat") ;
  q[2] = 0.0 ;     // qz = 0
  cout.flush() << "\n\tGAMA " ;   // *** GAMA - J ***
  qx0  = 0.0 ;  dqx  = 0.0 ;  qy0  = 0.0 ;
  dqy  = M_PI * sqrt (2.0) / 32.0 ;
  for ( i = 0; i <= 32; i++) {
     q[0] = qx0 + dqx * i ;
     q[1] = qy0 + dqy * i ;
     GetMatHam (V, OnSiteEner, q, tao, C) ;
     GetEigen (C, d, e) ;
     WriteEner (bDos, C, d, i) ;
     if ( ! (i % 4) ) cout.flush() << "-" ;
     fBZ << q[0] << " " << q[1] << "\n" ;
  }
  cout.flush() << "> J " ;
  qx0   = 0.0 ;   // *** J - K ***
  dqx   = M_PI * sqrt(2.0 / 3.0) / 18.0 ;
  qy0   = M_PI * sqrt(2.0) ;
  dqy   = 0.0 ;
  for (i = 1; i <= 18; i++) {
     q[0] = qx0 + dqx * i ;
     q[1] = qy0 + dqy * i ;
     GetMatHam (V, OnSiteEner, q, tao, C) ;
     GetEigen (C, d, e) ;
     WriteEner (bDos, C, d, i + 32) ;
     if ( ! (i % 4) ) cout.flush() << "-" ;
     fBZ << q[0] << " " << q[1] << "\n" ;
  }
  cout.flush() << "> K " ;
  qx0   = M_PI * sqrt(2.0 / 3.0) ;   // *** K - J' ***
  dqx   = 0.0 ;
  qy0   = M_PI * sqrt (2.0) ;
  dqy   = qy0 / 32.0 ;
  for ( i = 1; i <= 32; i++) {
     q[0] = qx0 - dqx * i ;
     q[1] = qy0 - dqy * i ;
     GetMatHam (V, OnSiteEner, q, tao, C) ;
     GetEigen (C, d, e) ;
     WriteEner (bDos, C, d, i + 50) ;
     if ( ! (i % 4) ) cout.flush() << "-" ;
     fBZ << q[0] << " " << q[1] << "\n" ;
  }
  cout.flush() << "> J' " ;
  qx0   = M_PI * sqrt(2.0 / 3.0) ;   // *** K - J' ***
  dqx   = qx0 / 18 ;
  qy0   = 0.0 ;  dqy   = 0.0 ;
  for ( i = 1; i <= 18; i++) {
     q[0] = qx0 - dqx * i ;
     q[1] = qy0 - dqy * i ;
     GetMatHam (V, OnSiteEner, q, tao, C) ;
     GetEigen (C, d, e) ;
     WriteEner (bDos, C, d, i + 82) ;
     if ( ! (i % 4) ) cout.flush() << "-" ;
     fBZ << q[0] << " " << q[1] << "\n" ;
  }
  cout << "> GAMA\n" ;
  cout << "\n\tEnergía de Fermi: " << EFermi / 100.0 << "\n" ;
}
```

Referencias y Bibliografía.

[1] Charles Kittel, *Introducción a la FISICA DE ESTADO SÓLIDO*, 3ª edición, Reverté 1995.

[2] H. Lüth, Surfaces and Interfaces of Solid Materials, Springer-Verlag, 1997.

[3] M. Prutton, Surface of Physics, Claredon, 1975.

[4] John E. Northrup and Marvin L. Cohen, *Atomic geometry and surface-state spectrum for Ge(111)-(2×1)*, PHYSICAL REVIEW B, VOLUME **27**, NUMBER 10, pag. 6553, 15 MAY 1983.

[5] Walter A. Harrinson, *ELECTRONIC STRUCTURE AND THE PROPERTIES OF SOLIDS*, Dover Publications, 1989.

[6] Bjarne Stoustrup, *El lenguaje de Programación C++*, Addison-Wesley, 1991.

[7] P. Vogl, Harold P. Hjalmarson and John D. Dow, J. Phys. Chem. Solids, *A SEMI-EMPIRICAL TIGHT-BINDING THEORY OF THE ELECTRONIC STRUCTURE OF SEMICONDUCTORS*, Vol. 44, No. **5**, pag. 365-378, 1983

[8] J. C. Slater and G. F. Koster, *Simplified LCAO Method for the Periodic Potencial Problem*, Physical Review, Volume **94**, pag. 1498, 1954.

[9] K. C. Pandey, Phys. Rev. Lett. **47**, pag. 1913 , 1981.

[10] J. M. Nicholls, G. V. Hanson, R. I. Uhrberg and S. A. Foldström, *Experimemtal dangling-bond on Ge(111)-2×1 surface*, Physical Review B, Volume **27**, number 4, pag. 2594, 1983.

[11] F. Solal, G. Jezequel, A Barski, P. Steiner, R. Pinchaux, E. Y. Petroff, Phys. Rev. Lett. **52**, pag 360, 1984.

[12] S. Nannarone, P. Chiaradia, F. Ciccacci, R. Memeo, P. Sassaroli, S. Selci, and G. Chiarotti, Solid State Commun., **33**, pag. 593, 1981.

[13] R. M. Feenstra, Physical Review B, Vol. **44**, pag. 13791, 1991

[14] Noburu Takeuchi, A. Selloni, A. I. Shkrebtii, E. Tosatti, *Structural and electronic properties of (111)2×1 surface of Ge from first-principles calculations*, Physical Review B, Volume **44**, Number **24**, pag. 13611, 15 Dec 1991-II

[15] Xuejun Zhu and Steven G. Louie, *Quasiparticle surface band structure and photoelectric threshold of Ge*(111)-2×1, PHYSICAL REVIEW B, VOLUME **43**, NUMBER 14, pag 12146, 15 MAY 1991

[16] Chelikowski J. R. and Cohen M. L., Phys. Rev. **15**, 556 (1976)

[17] Adrian P. Sutton, *Electronic Structure of Materials*, Oxford Science Publications, 1993.

Bibliografía General

- Solid-State Physics, Harald Ibach, Hans Lüth, Springer, 1995
- Semiconductor Surfaces and Interfaces, Friedhelm Bechstedt, Rolf Enderlein, AKADEMIE-VERLAG BERLIN, 1988.
- Basic Theory of Surface States, Sydney G. Davison and Maria Steslicka, Oxford Science Publications, 1992.
- Ana Cecilia Noguez Garrido, *TESIS DOCTORAL*, FACULTAD DE CIENCIAS UNAM,1995