Electron Spin Dynamics in Semiconductors

by

Mehmet Fatih Yanik

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

Author ....
Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by.... .....
Rajeev Ram
Associate Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Electron Spin Dynamics in Semiconductors

by

Mehmet Fatih Yanik

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2000, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

Spin dynamics offers a new dimension for microelectronic devices both in information storage and processing. The recognition of the spins of carriers can enable a new generation of devices, and promising steps have been taken in the recent years. The most crucial factor in spin effect devices is the preservation of spin over long enough time and length scales, and this issue has been controversial in the literature. This MEng thesis involves the study of electron spin dynamics in bulk III-V semiconductors by Monte Carlo simulations and optical pump-probe measurement methods. The agreement between the theoretical calculations and the experiments are better than the ones previously reported in the literature within the temperature ranges considered, and results clearly show the relative strength of different spin relaxation processes.

Thesis Supervisor: Rajeev Ram
Title: Associate Professor

Acknowledgements

I would like to thank to my supervisor, Prof. Rajeev Ram, for many things. He taught me much of what I know about how to do experiments, how to model experiments, and how to use theory and experiment in an effective way. He gave me great flexibility in defining and conducting my research. I also want to thank to my previous supervisor Prof. Tomas Arias, and my advisor Prof. Stephen Senturia, who supported me on every issue during my undergraduate years and masters.

I would like to thank to all of my friends in the Semiconductor Laser Group. Without their support, I could never finish this thesis on time. I am very greatful to my friend Herry Lee, a great experimentalist, for helping me in generating plots and in latex, and also for machining parts of the femtosecond setup at the machine shop. I want to thank to my classmate Erwin Lau, Kevin Pipe and Steve Patterson for their support, and to Farhan Rana for his patience to my never ending questions. I wish them a great future, and expect to see them at great positions both in academia and in industry.

I benefited greatly from my discussions with my friend Mehmet Ozgur Oktel at the Condensed Matter Theory Group of MIT. I wish him a great career after his Ph.D.

Many of my friends gave me great support; my best friend Cagri Savran, my roommate Ramazan Demir, Ertugrul Ozbudak a great biologist with whom I hope to work with in the future, Hale Ozsoy whom I wish a great future in finance, Oguz Silahtar a man who takes 4 H-level graduate courses in computer science and still finds time for night clubs, Namik K. Yilmaz, Danny Seth, two freshmen biologist friends of mine Aydin Albayrak and Gokhan Demirkan, Tanya Zelevinsky from the Harvard Physics Department, Zoe Teagarden from the MIT Media Lab, Aykut Firat, Suleyman Kachani president of MIT graduate students, Selim Temizer and many other friends I too numerous to count here. I want to thank them all for making this world an exciting place to live. I am thankful to my family for supporting me during every stage of my education.

I enjoyed working on the problem of electron spin dynamics; it is simple enough so that I obtained experimental and theoretical results in less than 8 months including the time I spent on learning C++ and constructing femtosecond pump-probe setup with its electronics and optics, and it is rich enough that I applied many ideas from quantum/statistical physics and sometimes came up with new ones, ranging from simple quasiparticle scattering problems to the creation of correlated/decorrelated electronic states, and spin dependent coherent/incoherent exchange processes. Simulations agreed with experiments very well, which is quite encouraging for me in the early stages of my career. I am indebted to my supervisor Prof. Rajeev Ram for giving me the chance to work on this problem and for his guidance.

Contents

List of Figures

Chapter 1

Introduction

The study of spin dynamics is becoming important because of the emerging field of spintronics which aims at using the electron's spin to store, manipulate and transport information [13]. Because of the fundamental nature of spin dependent processes, the study of spin dynamics might also lead to a better understanding of other dynamic quantum processes in interacting systems. In this thesis femtosecond optical methods are used to probe spin dynamics in semiconductors. Theoretical analyses of the experiments and modelling of spin dynamics in semiconductors are done via density matrix, Greens function and Monte Carlo methods. Sufficiently long spin lifetimes are necessary for the operation of efficient spin effect devices, and this thesis concentrates on the analysis of the dominant spin dephasing mechanisms in semiconductors.

The first chapter provides background and motivation for the thesis. The second chapter explains the dominant spin dephasing mechanisms in semiconductors. It presents various mechanisms suggested in the literature, and models in detail the ones relevant to our experiments and spin effect devices. The spin splitting in the bandstructures of III-V semiconductors is discussed in this section. The third chapter describes in detail the Monte Carlo simulations conducted to simulate spin dynamics in semiconductors. This section explains the relevant details of the Monte Carlo methods used, compares simulation results with the experiments to understand the essence of spin dephasing mechanisms in semiconductors.

The fourth chapter explains the optical experiments. For semiconductors, optical

methods make it possible to study electron spin dynamics despite the background of other magnetic processes (eg. scattering with holes, magnetic impurities). Optical techniques enable us to study various factors that effect spin lifetimes, including impurity and electron concentration, temperature, interface, magnetic field, spin orbit coupling and particle interactions. This chapter discusses the construction of an optical pump-probe setup used in optical measurements; its electronics, control, optical setup and alignment, and signal detection. The fifth chapter is the conclusion part of the thesis. Appendix A discusses nonequilibrium spin injection into micron scale metal structures. Appendix B includes the source code for Monte-Carlo simulations.

1.1 Background

Conventional electronics has ignored the spin of the electron. In the conventional silicon transistor, carriers are distinguished by their charges but no use of the fact that some carriers are spin-up and others are spin-down is made. The recognition of the spins of carriers can enable a new generation of devices, and promising steps have been taken in the recent years. The first spin dependent transport effect was discovered by N. F. Mott in 1936 [1]. Mott realized that in ferromagnetic materials electron conduction is carried out by two distinct conduction channels, one being spin parallel to the magnetization axis and the other being antiparallel. Mott's theoretical thesis was confirmed by the experiments of A. Fert and I. A. Campbell in 1968 [2] [3]. Their discovery confirmed the fact that electrons of one-spin polarization direction is more heavily scattered than the others. This difference in the scattering times of different spins is the one of the key ideas of spintronics. In 1988, independently Fert and Grunberg [4] [5] discovered the first spin electronic device, a passive two terminal device called giant magnetoresistive multilayers. Current passing through metal layers sandwiched between ferromagnets experiences a resistance depending on the relative alignment of the magnetic axis of ferromagnets. The resistance arises because of the difference in the mobility of carriers with different spin polarizations relative to the magnetic axis of ferromagnets. Next, the first three terminal device,

an all-metal transistor, was invented by Mark Johnson in 1993 [6]. Johnson's studies showed spin relaxation times of $10ps - 1ns$ with spin relaxation lengths of $1 - 10\mu m$ in aluminum at room temperature.

Although Johnson in 1994 suggested a mechanism [7] to obtain sufficient current gain, it hasn't been tested experimentally. Metals do not have the drawback of semiconductors when scaled and can be scaled down to dimensions as small as $10nm$. In semiconductors, surface states start to dominate and degrade as the dimensions decrease but metals do not suffer from that. Thus, in all metal transistor could be a good candidate for scaling down transistors if the problem of current gain can be solved.

In 1995, D. J. Monsma et al. suggested a magnetic field sensor called the Spin-Valve Transistor [12]. It consists of a base of a magnetic multilayer metal stack sandwiched between collector and emitter Si. The transmission is highly sensitive to the relative alignment of magnetic axis of ferromagnets, due to the spin polarization dependent conductivity of each layer. Although the device does not have enough gain to be a transistor, it can control the energies of hot electrons injected into the base region with a range from 0.2 to 3eV. Thus, this makes it possible to observe the dependence of spin relaxation mechanisms on the carrier energies. To our knowledge, the Spin-Valve Transistor is the only semiconductor spin transistor that has been experimentally demonstrated. However, it does not function as a logic element, instead it serves as a sensitive magnetic field sensor.

Another semiconductor spin transistor that is proposed by Datta [9] has been considered the most promising candidate for spin logic transistors for integrated circuits. It operates similarly to an optical modulator. A ferromagnetic emitter injects spin polarized electrons into the 2D electron gas formed at the heterojunction between InAlAs and InGaAs. The channel has very high mobility and is free of spin-flip scattering events. From the base, carriers are injected into the collector ferromagnet. Depending on the relative polarization of the spin of the carriers and the magnetic alignment of the ferromagnet, the carriers experience a resistance. Ideally when there is no spin scattering at the interfaces, the resistance should go to infinity when the

carrier's spin polarization is antiparallel to the polarization of the ferromagnetic collector. Thus improvement of interface spin scattering is crucial. This spin dependent resistance can be controlled by rotating spins by changing the spin orbit coupling of electrons in the 2D gas. Spin orbit coupling can be controlled by charge accumulation across the base. Thus a base voltage may control the resistance of the device. Other mechanisms may also be implemented to control the spin rotation in the base (eg. by applying a switchable magnetic field). The feasibility of Datta's proposal and the integration of spin effect devices in semiconductors require the preservation of spin phase over sufficiently long length and time scales. The proposal of Datta [9] requires a mean free path of $0.67\mu m$ for 2D gas.

Optical experiments towards understanding spin dynamics in semiconductors have been performed by several groups in the past, the most impressive ones being that of J. M. Kikkawa and D.D. Awschalom et. al. Kikkawa's and Awschalom's work includes the femto-second excitation of spin polarized carriers in bulk and in quantum structures of GaAs [15], [14]. They measured spin relaxation times of order $1 - 100ns$ in n-type bulk GaAs. Their results are encouraging for the construction of semiconductor spin devices with base transport times shorter than a few nanoseconds. Kikkawa and Awschalom studied the effect of magnetic fields on spin rotation and relaxation by Faraday Rotation method with magnetic fields of order of tens of Orsted. Although, their experiments showed long spin dephasing times, the mechanism of dominant spin dephasing process has not been analyzed, and this thesis provides such an analysis.

Chapter 2

Spin relaxation mechanisms and Modelling Non-Equilibrium Spin Dynamics

Spin relaxation refers to the conversion of an unbalanced population of spin polarized particles into a thermal equilibrium population. Spin relaxation mechanisms in semiconductors can arise due to spin scattering because of impurities, electron-electron collisions, electron-phonon collisions, spin-orbit coupling, internal inhomogenous magnetic fields, electron-hole exchange effect or due to the inhomogeneity in the spin factor g of electrons. In early work, inhomogenous broadening due to finite Δg was shown to be only a minor effect in GaAs [15]. In this section, the spin relaxation mechanisms of electrons are discussed. The discussion of holes is ignored because spin-orbit coupling is so strong for holes that spin polarized holes do not have practical significance.

There are three spin scattering mechanisms accepted in the literature for semiconductors: Elliott-Yafet, D'yakonov-Perel', and Bir-Aronov-Pikus. The Elliott-Yafet (EY) mechanism [20] arises because in semiconductors Bloch states are not spin eigenstates due to spin orbit coupling. At any given crystal momentum, spin up and spin down states are mixed with the spatial angular momentum of electrons such that it is not possible to diagonalize eigenstates in spin subspace. Usual spin independent

scattering from impurities [19] and carriers [17] can connect spin up states with spin down states. Thus, the spin scattering rate is proportional to momentum scattering rate in the Elliott-Yafet mechanism. Phonon-modulated spin-orbit interaction also causes similar spin scattering [27]. The coupling between the spin and orbital angular momentum is inversely proportional to the bandgab in semiconductors, and Elliott-Yafet mechanism is believed to be dominant in small band gab semiconductors such as InSb [24].

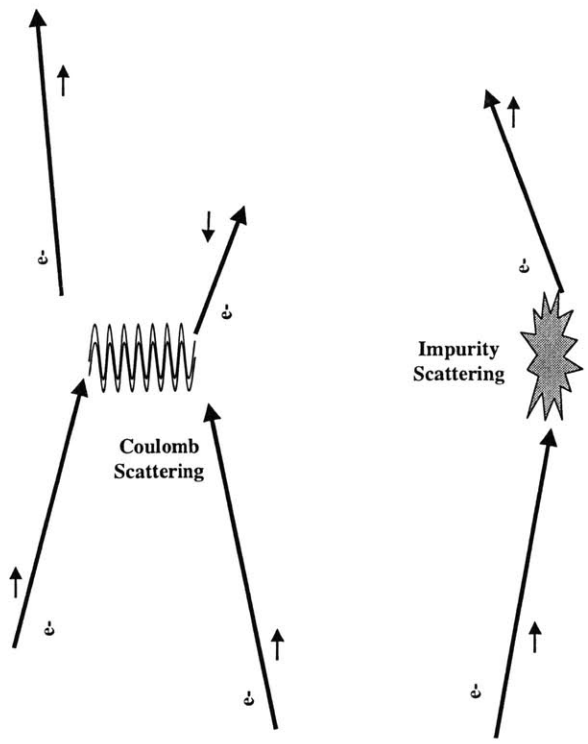

Figure 2-1: Elliott-Yafet mechanism: Coulomb and impurity/phonon scattering couples spin eigenstates

In crystals that lack spin inversion symmetry (such as zincbelende III-V semiconductors), the spin orbit interaction lifts the spin degeneracy [16]. However, at any momentum state, the spin eigenspace is diagonalizable unlike in the case of Elliott-Yafet mechanism. The spin eigenstates and their splitting become momentum dependent. A collective excitation consisting of electrons at different momentum states dephases due to the relative difference in the spin precession axis and precession rate at different momentum states. This momentum dependent spin splitting effect can

be described by a momentum dependent magnetic field $B(\overrightarrow{l})$. Momentum scattering causes a change in this effective magnetic field as the carriers scatter different momentum states. If the momentum scattering rate is fast enough, electrons are scattered between momentum states before they can precess appreciably in any momentum state. Since the precession direction changes as the electron is being scattered, the average precession rate decreases. This causes the spin scattering rate to become inversely proportional to the momentum scattering rate, and is called "motional narrowing". This effect is called the D'yakonov-Perel (DP) mechanism, and is believed to be the dominant mechanism for large band semiconductors such as GaAs. However, it is shown in this thesis that EY and DP are both significant at all temperatures in determining the order of magnitude of spin lifetime contrary to predictions reported in the literatur

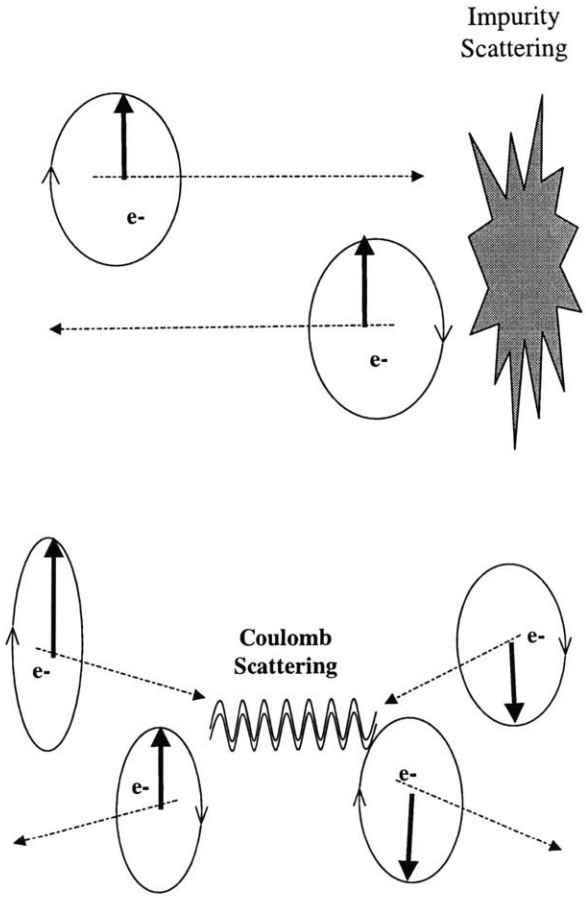

Figure 2-2: D'yakonov-Perel mechanism: spin precession direction changes due to collisions causing "motional narrowing"

Another source of spin relaxation arises due to spin scattering from holes and is called the Bir-Aronov-Pikus mechanism [25]. Energies of both bound and free electron-hole pairs split due to the spin dependent exchange effect which acts as an effective magnetic field on electrons. If the hole spin states are scattered at a much faster rate than the electron spin precession rate in this effective magnetic field, motional narrowing results as in the case of D'yakonov-Perel mechanism. The Bir-Aronov-Pikus mechanism is effective only in semiconductors with significant overlap of electron and hole wave functions. It can be suppressed either by keeping hole concentration small enough or by the decreasing overlap of hole and electron wave functions as in modulation doped heterostructures [26].

In the literature, the longest spin lifetimes are obtained with the absence of Bir-Aronov-Pikus mechanism by decreasing the interaction time of holes with electrons. The modeling in this thesis concentrates on Elliyott-Yafet and D'yakonov-Perel mechanism, and it neglects Bir-Aronov-Pikus mechanism since it can be suppressed. In the experiments considered for this thesis, Bir-Aronov-Pikus mechanism is suppressed by annihilating holes in short time scales before they influence the conduction band electrons via exchange spin scattering mechanism: Spin polarized holes and electrons are generated by polarized pump pulse. Since hole relaxation times are much shorter due to strong spin orbit coupling, the spin of the hole population dephases much faster (a few picoseconds) compared with the electron spin dephasing rate. After the scattering of hole spins, electron-hole recombination takes a few nanoseconds. Since the hole spins are randomized in a much shorter period of time than the electron-hole recombination time, collective hole spin polarization has vanished and recombination of holes and electrons leaves an excess amount of collective spin polarization in the conduction band. The recombination of holes zeros out Bir-Aronov-Pikus electron-hole exchange spin scattering mechanism.

The following section explains the spin energy splitting in the conduction band which results in D'yakonov-Perel scattering mechanism. D'yakonov-Perel (DP) mechanism has great generality because such a spin scattering mechanism can arise in any system whenever there exists a momentum or position dependent spin splitting. Scat-

tering in momentum and position space modifies the scattering rate significantly. In the second section, another spin orbit coupling effect is discussed which leads to the Elliott-Yafet scattering mechanism.

2.1 Spin Splitting in the Conduction Band of III-V semiconductors

In this section tight binding and $k \cdot p$ methods are used to show the origin of spin splitting effects in III-V semiconductors. In the III-V semiconductors, because of the lack of inversion symmetry, the spin orbit coupling splits spin eigenstates as a function of the crystal momentum. Spin orbit coupling arises due to the effective magnetic field that an electron feels in its moving reference frame with respect to the crystal atoms. The crystal hamiltonian H takes the following form

$$H = \frac{p^2}{2m} + V(\vec{r}) + \frac{1}{2mc}\vec{\nabla}V \times \vec{p} \cdot \vec{\sigma} \tag{2.1}$$

Due to the periodicity of the crystal, the effective crystal potential that an electron feels has a periodic dependence:

$$V(\vec{r}) = V(\vec{r} + \vec{R}) \tag{2.2}$$

where $\vec{R}$ is the crystal lattice vector that leaves crystal potential invariant under translation by $\vec{R}$. Such a crystal potential has Fourier components in momentum space at discrete reciprocal vectors $(\vec{g_1}, \vec{g_2}, \vec{g_3})$ and couples any two free space eigenstates $e^{i\vec{k}\cdot\vec{r}}$ and $e^{i\vec{k'}\cdot\vec{r}}$ whenever k and k' are different by $n * \vec{g_1} + m * \vec{g_2} + l * \vec{g_3}$ where l, m, n are integers. This implies that the eigenstates of the crystal hamiltonian can be written as

$$\psi_{n\sigma\vec{k}}(\vec{r}) = e^{i\vec{k}\cdot\vec{r}} u_{n\sigma\vec{k}}(\vec{r}) \tag{2.3}$$

where $u_{n\vec{k}}(\vec{r})$ is called a Bloch function and has periodicity of the crystal lattice. $n\sigma$ denotes spin eigenstate σ at band n with eigenenergy $E_{n\sigma\vec{k}}$. In the conduction and valence bands of III-V semiconductors, the lowest energy states occur near the Γ $(k = 0)$ point.

In order to understand the effect of spin orbit coupling as a perturbation on spin eigenstates, the symmetry of the conduction band and valence bands can be used. Since the states near the Γ point resemble atomic states of the crystal atoms, an intuitive way to find symmetries of conduction and valence bands is to use the tight binding approximation which expands eigenstates of the crystal in terms of the superposition of atomic states $\phi_\alpha(\vec{r} + \vec{R})$ at lattice point $\vec{R}$. A form of superposition that satisfies the translational invariance of crystal (Bloch Theorem) is

$$\chi_{\alpha\vec{k}} = \frac{1}{\sqrt{N}} \sum_{\vec{R}} e^{i\vec{k}\cdot\vec{r}} \phi_\alpha(\vec{r} + \vec{R}) \tag{2.4}$$

where spin degeneracy is assumed and the spin label σ is dropped. The eigenstates can be expressed as

$$|\psi_{n\vec{k}}\rangle = \sum_{\alpha'\vec{k'}} c_{\alpha'}(\vec{k'})|\chi_{\alpha'\vec{k'}}\rangle \tag{2.5}$$

To obtain the coefficients $c_{\alpha'}(\vec{k'})$, eigenstate equation is multiplied by $\langle\chi_{\alpha\vec{k}}|$ resulting in the eigenvalue equation

$$\sum_{\alpha'\vec{k'}} \langle\chi_{\alpha\vec{k}}|H|\chi_{\alpha'\vec{k'}}\rangle c_{\alpha'}(\vec{k'}) = Ec_\alpha(\vec{k}) \tag{2.6}$$

$'s'$ and $'p'$ atomic orbitals are frequently used in tight binding calculations of semiconductors

$$\phi_{s\uparrow\downarrow}, \phi_{p_x\uparrow\downarrow}, \phi_{p_y\uparrow\downarrow}, \phi_{p_z\uparrow\downarrow} \tag{2.7}$$

When only nearest neighbor interactions are taken into account among the atomic

orbitals at points

$$\pm\frac{a}{2} < 1, \pm 1, 0 >, \pm\frac{a}{2} < 1, 0, \pm 1 >, \pm\frac{a}{2} < 0, 1, \pm 1 > \tag{2.8}$$

$8X8$ Hamiltonian matrix results which is $4X4$ block diagonal due to spin degeneracy. Eigenstates of this Hamiltonian at Γ point have the symmetries of atomic orbitals: Conduction Band Eigenstates with energy E_s are $|S_{\downarrow\uparrow}\rangle$, and Valence Band Eigenstates with energy E_p are $|X_{\downarrow\uparrow}\rangle$, $|Y_{\downarrow\uparrow}\rangle$, $|Z_{\downarrow\uparrow}\rangle$, where (S, X, Y, Z) represent the symmetry of atomic orbitals (S, P_x, P_y, P_z) respectively.

Since the eigenstates at $k = 0$ are complete, $\vec{k} \neq 0$ states can be expanded in terms of $k = 0$ states and spin orbit coupling can be included as a perturbation. Let's substitute finite $\vec{k}$ eigenstates to the Hamiltonian by including spin orbit coupling

$$\{\frac{p^2}{2m} + V(\vec{r}) + \frac{\hbar}{4m^2c^2}[\vec{\nabla}V \times \vec{p}] \cdot \vec{\sigma}\}\psi_{n\vec{k}}(\vec{r}) = E_n(\vec{k})\psi_{n\vec{k}}(\vec{r}) \tag{2.9}$$

substituting $\psi_{n\sigma\vec{k}}(\vec{r}) = e^{i\vec{k}\cdot\vec{r}}u_{n\sigma\vec{k}}(\vec{r})$ results in

$$\{\tfrac{p^2}{2m} + V(\vec{r}) + \tfrac{\hbar}{m}\vec{k}\cdot\vec{p} + \tfrac{\hbar}{4m^2c^2}[\vec{\nabla}V \times \vec{p}] \cdot \vec{\sigma} + \tag{2.10}$$

$$\tfrac{\hbar^2}{4m^2c^2}[\vec{\nabla}V \times \vec{k}] \cdot \vec{\sigma}\}u_{n\vec{k}}(\vec{r}) = E_n(\vec{k})u_{n\vec{k}}(\vec{r}) \tag{2.11}$$

The first two terms in the brackets give the Hamiltonian of $k = 0$ eigenstates with no spin orbit coupling. The last three terms in the brackets are the perturbation terms for finite $\vec{k}$ and finite spin orbit coupling. The fifth term can be ignored because it is much weaker compared with the fourth term. The reason is that when the gradient of the crystal momentum becomes strongest near the center of lattice atoms, the actual momentum $\vec{p}$ gets much larger with respect to the crystal momentum $\vec{k}$. The simplified hamiltonian becomes

$$\{H_o + \frac{\hbar}{m}\vec{k}\cdot\vec{p} + \frac{\hbar}{4m^2c^2}[\vec{\nabla}V \times \vec{p}] \cdot \vec{\sigma}\}u_{n\vec{k}}(\vec{r}) = E_n(\vec{k})u_{n\vec{k}}(\vec{r}) \tag{2.12}$$

where H_o is the unperturbed Hamiltonian of states at $k = 0$. Finite k eigenstates can

be expanded in terms of $k = 0$ eigenstates

$$u_{n\vec{k}}(\vec{r}) = \sum_{n'} a_{n'} u_{n'0}(\vec{r}) \tag{2.13}$$

Before substituting this expression into the Hamiltonian making a basis transformation in the degenerate eigenspaces of $k = 0$ greatly simplifies the calculation yielding

$$|iS\downarrow\rangle, |\frac{X - iY}{\sqrt{2}}\uparrow\rangle, |Z\downarrow\rangle, |-\frac{X + iY}{\sqrt{2}}\uparrow\rangle \tag{2.14}$$

and

$$|iS\downarrow\rangle, |\frac{X - iY}{\sqrt{2}}\uparrow\rangle, |Z\downarrow\rangle, |-\frac{X + iY}{\sqrt{2}}\uparrow\rangle \tag{2.15}$$

where the first subset of states are not coupled to the second subset of states by the perturbing Hamiltonian. This leads to an $8X8$ Hamilatonian with $4X4$ block diagonals.

$$H_{8X8} = \begin{bmatrix} H_{4X4} & 0_{4X4} \\ 0_{4X4} & H_{4X4} \end{bmatrix} \tag{2.16}$$

where

$$H_{4X4} = \begin{bmatrix} E_s & 0 & kP & 0 \\ 0 & E_p - \frac{\Delta}{3} & \frac{\sqrt{2}}{3}\Delta & 0 \\ kP & \frac{\sqrt{2}}{3}\Delta & E_p & 0 \\ 0 & 0 & 0 & E_p + \frac{\Delta}{3} \end{bmatrix} \tag{2.17}$$

with

$$P = -i\frac{\hbar}{m}\langle S|p_z|Z\rangle, \Delta = \frac{3\hbar i}{4m^2c^2}\langle X|\frac{\partial V}{\partial x}p_y - \frac{\partial V}{\partial y}p_x|Y\rangle \tag{2.18}$$

The valence band can be selected to be ground state by setting $E_p + \Delta/3 = 0$ and

the Hamiltonian becomes

$$H_{4X4} = \begin{bmatrix} E_g & 0 & kP & 0 \\ 0 & -\frac{2}{3}\Delta & \frac{\sqrt{2}}{3}\Delta & 0 \\ kP & \frac{\sqrt{2}}{3}\Delta & -\frac{\Delta}{3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.19}$$

Elements of the above 8X8 Hamiltonian are calculated by using the symmetry properties of $k = 0$ basis states:

$$H_{11} = \langle iS \downarrow | H_o + \frac{\hbar}{m}\vec{k} \cdot \vec{p} + \frac{\hbar}{4m^2c^2}\vec{\nabla} V \times \vec{p} \cdot \vec{\sigma} | iS \downarrow \rangle \tag{2.20}$$

The second $k{\cdot}p$ term is zero because of the parity of basis. In the third term, σ_x and σ_y operator terms give zero contribution because they change the down spin to up spin. The σ_z term

$$\langle S \downarrow | \frac{\hbar}{4m^2c^2}\sigma_z(\frac{\partial V}{\partial x}p_y - \frac{\partial V}{\partial y}p_x) | S \downarrow \rangle \tag{2.21}$$

is also zero since exchange of (x, y) changes the sign of this term, yielding $H_{11} = H_o$

$$H_{13} = \langle iS \downarrow | H_o + \frac{\hbar}{m}\vec{k} \cdot \vec{p} + \frac{\hbar}{4m^2c^2}\vec{\nabla} V \times \vec{p} \cdot \vec{\sigma} | Z \downarrow \rangle \tag{2.22}$$

can be simplified by similar symmetry operations to

$$H_{13} = 0 - i\frac{\hbar}{m}k\langle S|p_z|Z\rangle + 0 = kP \tag{2.23}$$

Similarly

$$H_{22} = \langle \frac{X - iY}{\sqrt{2}} \uparrow | H_o + \frac{\hbar}{m}\vec{k} \cdot \vec{p} + \frac{\hbar}{4m^2c^2}\vec{\nabla} V \times \vec{p} \cdot \vec{\sigma} | \frac{X - iY}{\sqrt{2}} \uparrow \rangle \tag{2.24}$$

can be simplified to

$$H_{22} = E_p + \frac{1}{2}\langle X - iY | \frac{\hbar}{4m^2c^2}\vec{\nabla} V \times p \cdot \sigma_z | X - iY \rangle = E_p - \frac{\Delta}{3} \tag{2.25}$$

The other elements of the Hamiltonian can be calculated by similar symmetry properties yielding to

$$H_{33} = E_p \tag{2.26}$$

$$H_{44} = E_p + \tfrac{\Delta}{3} \tag{2.27}$$

$$H_{23} = H_{32} = \langle Z \downarrow | \tfrac{\hbar}{4m^2c^2} \vec{\nabla} V \times \vec{p} \cdot \vec{\sigma} | \tfrac{X-iY}{2} \uparrow \rangle = \tfrac{\sqrt{2}}{3}\Delta \tag{2.28}$$

The eigenstates are easily calculated for the upper $4X4$ part of the Hamiltonian

$$\phi_{hh\alpha} = | - \frac{X+iY}{\sqrt{2}} \uparrow\rangle, \phi_{n\alpha} = a_n|iS \downarrow\rangle + b_n|\frac{X-iY}{\sqrt{2}} \uparrow\rangle + c_n|Z \downarrow\rangle \tag{2.29}$$

and for the lower $4X4$ part of the Hamiltonian

$$\phi_{hh\beta} = |\frac{X-iY}{\sqrt{2}} \downarrow\rangle, \phi_{n\beta} = a_n|iS \uparrow\rangle + b_n| - \frac{X+iY}{\sqrt{2}} \downarrow\rangle + c_n|Z \uparrow\rangle \tag{2.30}$$

where hh and $n = so, lh, c$ indicates respectively the heavy hole, split hole, light hole and conduction bands with coefficients

$$a_n = \frac{kP(E'_n + 2\Delta/3)}{norm}, b_n = \frac{\sqrt{2}\Delta/3(E'_n - E_g)}{norm}, c_n = \frac{(E'_n - E_g)(E'_n + 2\Delta/3)}{norm} \tag{2.31}$$

The $norm$ is such that ${a_n}^2 + {b_n}^2 + {c_n}^2 = 1$. To this order the spin eigenstates $\phi_{c\downarrow}$ and $\phi_{c\uparrow}$ of the conduction band are degenerate in addition to the two fold degeneracy of the valence band eigenstates. The spin energy splitting in the conduction band arises due to scattering processes to the higher lying energy bands induced by the perturbation term of the Hamiltonian

$$V_{m,n} = \langle m|\frac{\hbar}{m}k \cdot p|n\rangle \tag{2.32}$$

where m, n are band indices. Coupling to the higher lying energy bands is given by

the self energy term

$$\Sigma_{m,n}(\omega) = \sum_{i,j}^{\Gamma'_{15}} V_{m,i} G_{i,j}(\omega) V_{j,n} \tag{2.33}$$

where i, j are the higher lying band indices, Γ'_{15} represents the symmetry of summation over all bands excluding the 4 bands, and $G_{i,j}$ is the propagator of the higher lying bands given by

$$G_{i,j}(\omega) = \frac{1}{\omega - E_i} \delta_{i,j} \tag{2.34}$$

Due to the smallness of the coupling, one can make the approximation $\omega \simeq \frac{E_c + E_v}{2}$ and add the approximate self-energy term to the $8X8$ Hamiltonian derived above and diagonalize to obtain the spin splitting energy

$$E_{c\alpha} - E_{c\beta} = \frac{4\Delta PT[k^2(k^2{}_x k^2{}_y + k^2{}_x k^2{}_z + k^2{}_y k^2{}_z) - 9k^2{}_x k^2{}_y k^2{}_z]^{1/2}}{3(E_c - E_v)(E_c - E_v + \Delta)} \tag{2.35}$$

where

$$T = \frac{2\hbar^2}{m^2} \sum_j^{\Gamma'_{15}} \frac{\langle S|p_x|u_j\rangle\langle u_j|p_y|Z\rangle}{(E_c + E_v)/2 - E_j} k_x k_y \tag{2.36}$$

and eigenstates are polarized along the direction of crystal momentum $\overrightarrow{k}$.

In this section, we have shown that scattering of the tight binding orbitals

$$\phi_{s\uparrow\downarrow}, \phi_{p_x\uparrow\downarrow}, \phi_{p_y\uparrow\downarrow}, \phi_{p_z\uparrow\downarrow} \tag{2.37}$$

to the higher lying energy bands cause spin splitting in the conduction band. In the section on Monte Carlo simulations, this splitting is used to model electron spin rotation between consecutive collisions.

2.2 Higher Order Spin Orbit Coupling Effects

Due to higher order spin coupling effects other than described in the previous section, Bloch eigenstates are no longer spin eigenstates. Spin orbit coupling mixes spin eigenstates with spatial angular momentum, and Bloch eigenstates can no longer be diagonalized in spin subspace. In the section on electron-electron collisions, the consequences of these higher order spin orbit coupling effects is discussed. It is no longer possible to separate energy eigenstates into purely spin up $|\vec{k}\uparrow\rangle$ and spin down $|\vec{k}\downarrow\rangle$ states but eigenstates can be classified as up $|\vec{k}+\rangle$ or down $|\vec{k}-\rangle$ polarized according to how much spin up or down state contributes to their wavefunctions. Spin orbit coupling mixes the spin eigenstates in such a way that up and down polarized Bloch eigenstates $e^{\vec{k}} u_{\vec{k},\vec{\sigma}}$ at different momentum states have a finite overlap [17] which would be absent otherwise,

$$\eta_{k,k'} = \frac{1}{\Omega}\int_\Omega u^*_{\vec{k}+}(\vec{r})u_{\vec{k'}-}(\vec{r})d^3r \cong \frac{\gamma}{2}\frac{\hbar^2}{\sqrt{2}m_{eff}E_g}(k_-k'_z - k_zk'_-) \tag{2.38}$$

$$\equiv (\vec{k}+|\vec{k'}-) \tag{2.39}$$

where $k_- = k_x - ik_y$, $\gamma = \frac{2\Delta(\Delta+2E_g)}{(\Delta+E_g)(2\Delta+3E_g)}$, and Δ, E_g are the spin orbit splitting and band gab energies. Note that there is an error in reference [17] corrected in equation 2.38 which is derived from the Bloch wavefunctions calculated in reference [18].

This overlap gives finite Coulomb scattering matrix element between up and down polarized states due to mixing in spin up and spin down states,

$$V_{+-}(\vec{q}) = \langle(\vec{k}-\vec{q})+,(\vec{k}'+\vec{q})\sigma|V_{coulomb}|\vec{k}+,\vec{k}'\sigma\rangle \tag{2.40}$$

$$= V_{coulomb}(\vec{q})\times(\vec{k}+|\vec{k}'-)(\vec{q})\times(\vec{k}\sigma|\vec{k}'\sigma) \tag{2.41}$$

Similarly in the electron impurity and phonon scattering processes, such an overlap mixes spin eigenstates. In the section on Monte Carlo Simulations, the EY spin scattering induced by momentum scattering processes is discussed.

Chapter 3

Monte Carlo Many Particle Simulations of Spin Dynamics

Electron spin dynamics in the literature have been modeled analytically under various approximations. In order to obtain analytical expressions, scattering processes have been simplified considerably and carriers have been considered in an equilibrium Fermi distribution [21][22]. Most importantly, the effect of electron-electron scattering is completely neglected which results in orders of magnitude different spin lifetimes as shown in the next sections. Optical experiments conducted in the literature result in non equilibrium distributions of carriers during the initial carrier thermalization stage. Although one can ignore the effects of this short time non-equilibrium dynamics on long time electron spin dynamics, there is another non-equilibrium situation that is maintained for long time scales. This non equilibrium arises due to the high level excitation of spin polarized carriers that results out of equilibrium within the spin up and down polarized carriers. This non-equilibrium lasts for time scales on the order of spin dephasing time, and cannot be considered as negligible. Since the carriers from one spin polarization direction are converted to the opposite spin polarization with a rate proportional to their momentum vectors, a dynamic non-equilibrium Fermi distribution is created as the carriers dephase, and evolves towards equilibrium by fast electron-electron collisions. In addition to these, for a fair comparison of the effectiveness of DP and EY mechanisms, both mechanisms have to be modeled under

the same assumptions on carrier scattering processes. Monte Carlo simulations are well suited to include all these effects which are not considered in the literature, and the results of this thesis lead to results in better agreement with experiments than earlier calculations reported in the literature. The Monte Carlo (MC) Method tracks the dynamics of a large number of carriers in a classical phase space. In this thesis, quantum effects are also introduced into the MC simulations within quasiparticle picture by spin dependent single particle self-energy, spin density matrix, and screening factors in collisions.

3.1 Hamiltonian

The dynamics of electrons and phonons in our system is governed by the following Hamiltonian:

$$\sum_{k,\sigma} \frac{\hbar^2 k^2}{2m} c^\dagger_{k,\sigma} c_{k,\sigma} + \sum_k T_{\sigma,\sigma'}(k) c^\dagger_{k\sigma'} c_{k\sigma} + H_{Coulomb} + H_{phonon} + H_{imp} + H_{el-phonon} \quad (3.1)$$

The terms in the Hamiltonian represent respectively electron kinetic energy, spin splitting due to crystal lattice potential, electron-electron Coulomb interaction energy, phonon energy, electron accoustic/optical phonon and electron impurity interaction energies.

3.2 Electron Impurity and Phonon Scattering

The electron phonon interaction in second quantization is

$$H_{el-phonon} = \sum_q M_q c^\dagger_{k+q} c_k (a_q + a^\dagger_{-q}) \quad (3.2)$$

and the electron impurity interaction is

$$H_{imp} = \sum_{p,q,\{R_\alpha\}} \frac{V_q}{\Omega} e^{-i\vec{q}\cdot\vec{R_\alpha}} c^\dagger_{p+q} c_p \quad (3.3)$$

If the impurity concentration is not too high or electron coherence length is small enough and if the phonon electron coupling is not too strong unlike superconductors, the electron impurity and phonon scattering processes can be represented by scattering rates via Fermi's Golden Rule. This is a valid case for the semiconductor systems when electron localization length λ_T due to thermal energy is much smaller that mean particle separation,

$$\lambda_T = \frac{\hbar}{\sqrt{2m_e kT}} \ll N_e^{-1/3} \tag{3.4}$$

The scattering rates are then used in the Monte Carlo (MC) simulation to select scattering processes as described in the next sections on the implementation of MC simulations. In the rate equations, the non-parabolic shape of the conduction band is ignored to gain from the computation time in MC simulations, and the results are in good agreement with experiments.

For the electron accoustic-phonon scattering in GaAs, the scattering process can be approximated as elastic and the scattering rate is given by [33]

$$\Gamma_{el-acph}(\overrightarrow{k}) = \frac{2\pi D_{ac}^2 k_B T N_E}{\hbar \rho v_s^2} \tag{3.5}$$

where ρ is the mass density of GaAs, v_s is the sound velocity in GaAs, D_{ac} is electron accoustic deformation potential and $N_E = \frac{mk}{\pi^2\hbar^2}$ is the density of states per unity energy with N_e as the electron density.

Due to the large acoustic-phonon momentum, the scattering direction is independent of the carrier's original momentum vector and the scattering angles can be taken as uniformly distributed

$$\theta = \arccos(1 - 2r) \tag{3.6}$$

$$\phi = 2\pi r' \tag{3.7}$$

where r and r' are random variables uniformly distributed between 0 and 1.

Optical phonon scattering involves emission and absorption of an optical phonon

(coherent multiple phonon emission is unlikely for the excitation energies considered in this thesis). Assuming that the phonon distribution does not significantly deviate from equilibrium, the emission and absorption rates are given by [33]

$$\Gamma_{el-opph}^{emission}(\overrightarrow{k}) = \frac{q^2 m}{4\pi\hbar^2 k}\omega_{op}(\frac{1}{\epsilon_\infty} - \frac{1}{\epsilon_0})\ln(|\frac{k+\sqrt{k^2-2m\omega_{op}/\hbar}}{k-\sqrt{k^2-2m\omega_{op}/\hbar}}|)(N_{opph}+1) \quad (3.8)$$

$$\Gamma_{el-opph}^{absorption}(\overrightarrow{k}) = \frac{q^2 m}{4\pi\hbar^2 k}\omega_{op}(\frac{1}{\epsilon_\infty} - \frac{1}{\epsilon_0})\ln(|\frac{k+\sqrt{k^2+2m\omega_{op}/\hbar}}{k-\sqrt{k^2+2m\omega_{op}/\hbar}}|)N_{opph} \quad (3.9)$$

where $N_{opph} = \frac{1}{\exp(\hbar\omega_{op}/k_B T)-1}$ is the optical phonon occupancy factor. The scattering angles θ and ϕ in optical phonon scattering are selected as a function of random variables r and r' uniformly distributed between 0 and 1

$$f = 2k\frac{\sqrt{k^2+2m\omega_{op}/\hbar}}{(k-\sqrt{k^2+2m\omega_{op}/\hbar})^2} \quad (3.10)$$

$$\theta = \arccos(\frac{1+f-(1+2f)^r}{f}) \quad (3.11)$$

$$\phi = 2\pi r' \quad (3.12)$$

The scattering rate from impurities is given by [33]

$$\Gamma_{imp}(\overrightarrow{k}) = \frac{\pi}{8\hbar}(\frac{Zq^2}{4\pi\epsilon})^2 k^4 N_{imp} N_E (\frac{2k}{l_D})^2 \frac{1}{1+(\frac{l_D}{2k})^2} \quad (3.13)$$

where the Debye screening length is $l_D = \sqrt{\frac{N_e q^2}{\epsilon k_B T}}$. The electron scattering from impurities is rather anisotropic, and the scattering angles can be expressed in terms of random variables r and r' uniformly distributed between 0 and 1 [33]

$$\theta = \arccos(1 - \frac{2(1-r)}{1+4k^2 r/l_D^2}) \quad (3.14)$$

$$\phi = 2\pi r' \quad (3.15)$$

3.3 Quasi-particle Picture with Dynamic/Static Screening

The bare coulomb scattering is divergent for small momentum transfers q when treated by Fermi's Golden Rule. One has to take into account the collective response and screening of the electron gas in the electron-electron interaction. The collective behaviour of the electron gas can be seen clearly by a unitary transformation of the Coulomb interaction [40] within the random phase approximation (RPA)

$$H_{Coulomb} = \sum_{\substack{q>q_c, k_1, k_2, \\ \sigma_1\sigma_1'\sigma_2\sigma_2'}} V_{\substack{q,\sigma_1\sigma_1' \\ \sigma_2\sigma_2'}} c^\dagger_{k_1-q,\sigma_1'} c^\dagger_{k_2+q,\sigma_2'} c_{k_2,\sigma_2} c_{k_1,\sigma_1} \tag{3.16}$$

$$+\sum_k \hbar\omega_p(a_k^\dagger a_k + \frac{1}{2}) + H_{el-plasmon} \tag{3.17}$$

The first term represents the interaction among quasi-particles, the second term represents the free energy of collective excitations-plasmons. The last term represents the interaction between the quasi-particles and plasmons. The scattering term among the quasi-particles have a lower cutoff momentum transfer q_c, and such a cutoff can be represented by a dynamic dielectric constant within the RPA plasmon pole approximation leading to a screened and short range coulomb interaction

$$\widetilde{V}_{q,\sigma_1\sigma_1'\sigma_2\sigma_2'} = \frac{V_{q,\sigma_1\sigma_1'\sigma_2\sigma_2'}}{\epsilon_{RPA}(q)} \tag{3.18}$$

with

$$\epsilon_{RPA}(q) = 1 + \frac{\omega_{pl}^2}{(\omega+i\delta)^2-\omega_q^2} \tag{3.19}$$

$$\omega_q^2 = \omega_{pl}^2(1+\frac{q^2}{\kappa^2}) \tag{3.20}$$

$$\kappa = \sqrt{\frac{4\pi q^2}{\epsilon}\frac{\partial n}{\partial \mu}} \tag{3.21}$$

Such a screened short range interaction is a good approximation if the electron gas is not too dilute [41]. This is the case in our experiments. The screened Coulomb

interaction between quasiparticles can be treated by scattering rates via Fermi's Golden Rule, when the electron gas is dilute enough, etc., the thermal wavelength $\lambda_{th} = \frac{\hbar}{\sqrt{2m_c k_B T}}$is much larger than electron mean free path $n^{-1/3}$. This is valid for the temperatures and densities studied in this thesis. The scattering rate without any exchange process and Pauli exclusion for two oppositely polarized carriers is then given by [35]

$$\lambda(|\vec{g}|) = \frac{mq^2 k_B T}{4\pi\hbar^3 \epsilon\epsilon_o} \frac{g}{(g^2+\beta^2)} \tag{3.22}$$

where $\vec{g} = \vec{k}_1 - \vec{k}_2$ is the relative wave vector of carriers before the collision.The scattering angle at the center of mass frame can be selected as a function of random variables r and r' uniformly distributed between 0 and 1

$$\theta = 1 - \frac{2r}{1+\frac{g^2}{\beta^2(1-r)}} \tag{3.23}$$

$$\phi = 2\pi r' \tag{3.24}$$

which specifies the relative wave vector of carriers after collision $\vec{g}'$. The momentum vectors after the collision become

$$\vec{k_2}' = \frac{\vec{k_1}+\vec{k_2}-\vec{g}'}{2} \tag{3.25}$$

$$\vec{k_1}' = \vec{g}' + \vec{k_2} \tag{3.26}$$

The total scattering rate for a particle is equal to the sum of the rate of scattering from all other particles,

$$\Gamma_{e-e}(\vec{k_o}) = \sum_{otherparticles} \lambda(|\vec{k} - \vec{k_o}|) \tag{3.27}$$

For the MC simulations, instead of calculating this rate at each scattering process, a self scattering approach is used as described in the section on MC method which requires a tight upper bound for the maximum scattering rate. Such a tight bound can be obtained by substituting $g/(g^2+\beta^2)$ with its maximum value $1/2\beta$ in equation

3.22 [36]

$$\Gamma_{e-e,max}(\overrightarrow{k_o}) = \frac{mq^4 N_e}{8\pi(\epsilon\epsilon_o)^2\hbar^3\beta^3} \tag{3.28}$$

In the next section, carrier-carrier scattering for arbitrarily spin polarized carriers with spin orbit coupling effects is considered.

3.4 Spin Polarized Carrier-Carrier Scattering and Exchange Effects

In the MC simulations, each electron's spin degree of freedom is treated via a two by two density matrix. The Coulomb scattering rate between two carriers depends on the spin polarity of scattering carriers due to anti-symmetrization of the electronic wavefunctions, often called exchange interaction. The Coulomb interaction also induces correlation among the spins of the scattering carriers. The Coulomb interaction including the spin-orbit effects mentioned in the previous sections is

$$\begin{aligned}
H_{Coulomb} &= \sum_{q,k_1,k_2,\sigma_1\sigma_1'\sigma_2\sigma_2'} \tilde{V}_{q,\sigma_1\sigma_1'\sigma_2\sigma_2'} c^\dagger_{k_1-q,\sigma_1'} c^\dagger_{k_2+q,\sigma_2'} c_{k_2,\sigma_2} c_{k_1,\sigma_1} && (3.29)\\
&= \sum_{q,k_1,k_2,\sigma_1\sigma_2} \tilde{V}_q [c^\dagger_{k_1-q,\sigma_1} c^\dagger_{k_2+q,\sigma_2} c_{k_2,\sigma_2} c_{k_1,\sigma_1} + && (3.30)\\
&\quad +\eta_{k_2,k_2+q} c^\dagger_{k_1-q,\sigma_1} c^\dagger_{k_2+q,\overline{\sigma}_2} c_{k_2,\sigma_2} c_{k_1,\sigma_1} + && (3.31)\\
&\quad +\eta_{k_1,k_1-q} c^\dagger_{k_1-q,\overline{\sigma}_1} c^\dagger_{k_2+q,\sigma_2} c_{k_2,\sigma_2} c_{k_1,\sigma_1}] && (3.32)
\end{aligned}$$

where two spin flip processes are ignored and the over lines on spin variables σ indicate opposite spin polarity. The factor η factor arises due to spin orbit splitting as shown in the previous section on "Higher Order Spin Orbit Coupling Effects" and repeated here

$$\eta_{k,k'} = \frac{1}{\Omega}\int_\Omega u^*_{\overrightarrow{k}_+}(\overrightarrow{r}) u_{\overrightarrow{k'}_-}(\overrightarrow{r}) d^3r \cong \frac{\gamma}{2}\frac{\hbar^2}{\sqrt{2}m_{eff}E_g}(k_- k'_z - k_z k'_-) \tag{3.33}$$

where $k_- = k_x - ik_y$, $\gamma = \frac{2\Delta(\Delta+2E_g)}{(\Delta+E_g)(2\Delta+3E_g)}$, and Δ, E_g are the spin orbit splitting and band gab energies.

The initial state of two uncorrelated electrons incident with momentum vectors k_1 and k_2 can be represented in the occupancy space as

$$|\psi_o\rangle = (\alpha_1 c^\dagger_{k_1,\uparrow} + \beta_1 c^\dagger_{k_1,\downarrow})(\alpha_2 c^\dagger_{k_2,\uparrow} + \beta_2 c^\dagger_{k_2,\downarrow})|vac\rangle \tag{3.34}$$

where the anti-symmetrization is taken into account by the commutation property of creation and annihilation operators. Operating with the Coulomb term on this state yields the state after the collision

$$\begin{aligned} H|\psi_o\rangle = \sum_q \widetilde{V}_q \quad [\quad & (\alpha_1\alpha_2 + \eta_{k_2,k_2+q}\alpha_1\beta_2 + \eta_{k_1,k_1+q}\beta_1\alpha_2) c^\dagger_{k_1-q,\uparrow} c^\dagger_{k_2+q,\uparrow} & (3.35)\\ + \quad & (\beta_1\beta_2 + \eta_{k_1,k_1+q}\alpha_1\beta_2 + \eta_{k_2,k_2+q}\beta_1\alpha_2) c^\dagger_{k_1-q,\downarrow} c^\dagger_{k_2+q,\downarrow} & (3.36)\\ + \quad & (\alpha_1\beta_2 + \eta_{k_2,k_2+q}\alpha_1\alpha_2 + \eta_{k_1,k_1-q}\beta_1\beta_2) c^\dagger_{k_1-q,\uparrow} c^\dagger_{k_2+q\downarrow} & (3.37)\\ + \quad & (\alpha_2\beta_1 + \eta_{k_1,k_1+q}\alpha_1\alpha_2 + \eta_{k_2,k_2-q}\beta_1\beta_2) c^\dagger_{k_1-q,\downarrow} c^\dagger_{k_2+q\uparrow}] & (3.38) \end{aligned}$$

Projecting the state space to two particles propagating in directions $\overrightarrow{k_1} - \overrightarrow{q_o}$ and $\overrightarrow{k_2} + \overrightarrow{q}_o$ yield

$$\begin{aligned} H|\psi_o\rangle &= A c^\dagger_{k_1-q,\uparrow} c^\dagger_{k_2+q,\uparrow} + B c^\dagger_{k_1-q,\downarrow} c^\dagger_{k_2+q,\downarrow} & (3.39)\\ &+ C c^\dagger_{k_1-q,\uparrow} c^\dagger_{k_2+q\downarrow} + D c^\dagger_{k_1-q,\downarrow} c^\dagger_{k_2+q\uparrow} & (3.40) \end{aligned}$$

where the coefficients A, B, C, D are

$$\begin{aligned} A &= (\widetilde{V}_{q_o} - \widetilde{V}_{q_o+k_2-k_1})(\alpha_1\alpha_2 + \eta_{k_2,k_2+q}\alpha_1\beta_2 + \eta_{k_1,k_1+q}\beta_1\alpha_2) & (3.41)\\ B &= (\widetilde{V}_{q_o} - \widetilde{V}_{q_o+k_2-k_1})(\beta_1\beta_2 + \eta_{k_1,k_1+q}\alpha_1\beta_2 + \eta_{k_2,k_2+q}\beta_1\alpha_2) & (3.42)\\ C &= \widetilde{V}_{q_o}(\alpha_1\beta_2 + \eta_{k_2,k_2+q}\alpha_1\alpha_2 + \eta_{k_1,k_1-q}\beta_1\beta_2) + & (3.43)\\ & -\widetilde{V}_{q_o+k_2-k_1}(\alpha_2\beta_1 + \eta_{k_1,k_1+q}\alpha_1\alpha_2 + \eta_{k_2,k_2-q}\beta_1\beta_2) & (3.44)\\ D &= \widetilde{V}_{q_o}(\alpha_2\beta_1 + \eta_{k_1,k_1+q}\alpha_1\alpha_2 + \eta_{k_2,k_2-q}\beta_1\beta_2) + & (3.45)\\ & -\widetilde{V}_{q_o+k_2-k_1}(\alpha_1\beta_2 + \eta_{k_2,k_2+q}\alpha_1\alpha_2 + \eta_{k_1,k_1-q}\beta_1\beta_2) & (3.46) \end{aligned}$$

The density matrix after scattering is in pure state

$$\widehat{\rho} = \frac{H|\psi_o\rangle\langle\psi_o|H^\dagger}{\langle\psi_o|H^\dagger H|\psi_o\rangle} \tag{3.47}$$

As seen above, the spin degrees of freedom of the scattering carriers is correlated after the collision. In the MC simulation with a large number of carriers, only the single particle spin density matrix can be tracked, and spin correlations among two particles have to be projected to obtain two uncorrelated single particle density matrices. In order to obtain uncorrelated spin density matrix for the carrier with wavevector $\overrightarrow{k_1} - \overrightarrow{q_o}$, the spin degree of freedom of the carrier with wavevector $\overrightarrow{k_2} + \overrightarrow{q}_o$ has to be traced

$$\widehat{\rho}_1 = tr_{k_2+q_o}\widehat{\rho} \tag{3.48}$$

with the matrix elements calculated by

$$\langle\sigma_1|\widehat{\rho}_1|\sigma_1'\rangle = \sum_{\sigma_2}\langle vac|c_{k_1-q_o,\sigma_1}c_{k_2+q_o,\sigma_2}\widehat{\rho}c^\dagger_{k_2+q_o,\sigma_2}c^\dagger_{k_1-q_o,\sigma_1'}|vac\rangle \tag{3.49}$$

leading to

$$\widehat{\rho}_1 = \frac{1}{A^*A + C^*C + B^*B + D^*D}\begin{bmatrix} A^*A + C^*C & AD^* + CB^* \\ A^*D + CB^* & B^*B + D^*D \end{bmatrix} \tag{3.50}$$

and analogous operation yields the spin density matrix for the second particle after collision

$$\widehat{\rho}_2 = \frac{1}{A^*A + C^*C + B^*B + D^*D}\begin{bmatrix} A^*A + D^*D & C^*A + B^*D \\ A^*C + D^*B & B^*B + C^*C \end{bmatrix} \tag{3.51}$$

The scattering rate S for arbitrarily polarized carriers is given in the Born Approximation by Fermi's Golden Rule

$$S = \frac{2\pi}{\hbar}\,|\langle\psi_{final}|H|\psi_o\rangle|^2 \tag{3.52}$$

with

$$\psi_{final} = \frac{H|\psi_o\rangle}{\sqrt{|\langle\psi_o|H^\dagger H|\psi_o\rangle|}} \tag{3.53}$$

results in

$$S = \frac{2\pi}{\hbar}(|A|^2 + |B|^2 + |C|^2 + |D|^2) \tag{3.54}$$

for the scattering rate. As shown above in order to calculate the scattering rates and final states, the single particle density matrix has to be decomposed into pure states $|\psi_w\rangle$ with corresponding propabilities f_w,

$$\widehat{\rho} = \sum f_w |\psi_w\rangle\langle\psi_w| \tag{3.55}$$

From a computational perspective, it is efficient to use a single pure state for each electron instead of a density matrix to store and propagate spin. Such a pure state could be obtained, from a knowledge of f_w by a propababilistic selection rule in Monte Carlo simulation. However, decomposition of density matrix to its pure states and corresponding probability factors f_w is not unique. One has to resort to physical intuition to select a pure state. The tracing operation on one of the particles degrees of freedom physically represent the process of that particle being coupled to a reservoir with a continuous spectrum. If the correlated two-particle state is coupled to such a reservoir via the second particle's spin degree of freedom by a Hamiltonian, the final spin state evolves towards one of the eigenstates of the spin hamiltonian.

The most general two particle final state can be expressed as

$$|\psi_{final}\rangle = (a_{1\uparrow}c^\dagger_{k_1-q,\uparrow} + a_{1\downarrow}c^\dagger_{k_1-q,\downarrow})(a_{2\uparrow}c^\dagger_{k_2+q,\uparrow} + a_{2\downarrow}c^\dagger_{k_2+q,\downarrow}) + \tag{3.56}$$

$$+(b_{1\uparrow}c^\dagger_{k_1-q,\uparrow} + b_{1\downarrow}c^\dagger_{k_1-q,\downarrow})(b_{2\uparrow}c^\dagger_{k_2+q,\uparrow} + b_{2\downarrow}c^\dagger_{k_2+q,\downarrow})|vac\rangle \tag{3.57}$$

where the second particle's states

$$(a^{\dagger}_{2\uparrow}c_{k_2+q,\uparrow} + a^{\dagger}_{2\downarrow}c_{k_2+q,\downarrow})|vac\rangle \tag{3.58}$$

$$(b_{2\uparrow}c^{\dagger}_{k_2+q,\uparrow} + b_{2\downarrow}c^{\dagger}_{k_2+q,\downarrow})|vac\rangle \tag{3.59}$$

can be selected to be the eigenstates of the spin Hamiltonian of the second particle and to be orthogonal to each other resulting in the constraint

$$\langle vac|(a_{2\uparrow}c_{k_2+q,\uparrow} + a_{2\downarrow}c_{k_2+q,\downarrow})(b_{2\uparrow}c^{\dagger}_{k_2+q,\uparrow} + b_{2\downarrow}c^{\dagger}_{k_2+q,\downarrow})|vac\rangle = \tag{3.60}$$

$$a^{*}_{2\uparrow}b_{2\uparrow} + a^{*}_{2\downarrow}b_{2\downarrow} = 0 \tag{3.61}$$

However, the single particle states with momentum $\overrightarrow{k}_1 - \overrightarrow{q}$ cannot be forced to be orthogonal at the same time. When the second electron's spin degree of freedom is coupled to a reservoir with spectrum of eigenstates $|R_i\rangle$, the new state would evolve to

$$|\psi_{final}\rangle = (a_{1\uparrow}c^{\dagger}_{k_1-q,\uparrow} + a_{1\downarrow}c^{\dagger}_{k_1-q,\downarrow})(a_{2\uparrow}c^{\dagger}_{k_2+q,\uparrow} + a_{2\downarrow}c^{\dagger}_{k_2+q,\downarrow})|vac\rangle \otimes |R_a\rangle + \tag{3.62}$$

$$+(b_{1\uparrow}c^{\dagger}_{k_1-q,\uparrow} + b_{1\downarrow}c^{\dagger}_{k_1-q,\downarrow})(b_{2\uparrow}c^{\dagger}_{k_2+q,\uparrow} + b_{2\downarrow}c^{\dagger}_{k_2+q,\downarrow})|vac\rangle \otimes |R_b\rangle \tag{3.63}$$

The correlated two-particle state in the above expression is a superposition of two uncorrelated two-particle states 3.56 and 3.57. Reduction to one of the reservoir states would lead to one of the two uncorrelated two-particle states 3.56, 3.57. The single particle state of the first electron would be one of the following states

$$(a_{1\uparrow}c^{\dagger}_{k_1-q,\uparrow} + a_{1\downarrow}c^{\dagger}_{k_1-q,\downarrow})|vac\rangle \tag{3.64}$$

$$(b_{1\uparrow}c^{\dagger}_{k_1-q,\uparrow} + b_{1\downarrow}c^{\dagger}_{k_1-q,\downarrow})|vac\rangle \tag{3.65}$$

The difficulty with the methodology described above is that the hamiltonian that couples the second electron to the reservoir of other electrons is uncomputable in practice, thus it is not possible to identify the eigenstates of such a hamiltonian as it is used above. However, since there is no preferred direction in spin space and

since the coupling hamiltonian in a collision is expected to be uncorrelated with the coupling hamiltonian in the next collisions (due to the fast e-e scattering rate), the eigenstates of the spin hamiltonian in the Monte Carlo simulation can be oriented along a random direction after each collision by using two random variables uniformly distributed between 0 and 1

$$\theta = acos(1-2r) \tag{3.66}$$

$$\phi = 2\pi r' \tag{3.67}$$

$$\vec{v} = (sin(\theta)cos(\phi), sin(\theta)sin(\phi), cos(\theta)) \tag{3.68}$$

$$S = \vec{v} \cdot \vec{\sigma}_{pauli} \tag{3.69}$$

$$\begin{pmatrix} a_{2\uparrow} \\ a_{2\downarrow} \end{pmatrix} = Normalized\, 1^{st}\, Eigenstate\, of\, S \tag{3.70}$$

$$\begin{pmatrix} b_{2\uparrow} \\ b_{2\downarrow} \end{pmatrix} = Normalized\, 2^{nd}\, Eigenstate\, of\, S \tag{3.71}$$

Once the state coefficients of the second particle are selected, the first particle's state coefficients can be determined by equating equation 3.56- 3.57 with the normalized form of final state in the expression 3.39- 3.40

$$a_{1\uparrow}a_{2\uparrow} + b_{1\uparrow}b_{2\uparrow} = A \cdot norm \tag{3.72}$$

$$a_{1\downarrow}a_{2\downarrow} + b_{1\downarrow}b_{2\downarrow} = B \cdot norm \tag{3.73}$$

$$a_{1\uparrow}a_{2\downarrow} + b_{1\uparrow}b_{2\downarrow} = C \cdot norm \tag{3.74}$$

$$a_{1\downarrow}a_{2\uparrow} + b_{1\downarrow}b_{2\uparrow} = D \cdot norm \tag{3.75}$$

with

$$norm = |A|^2 + |B|^2 + |C|^2 + |D|^2 \tag{3.76}$$

and solving these equations for the first particle's state coefficients yield

$$a_{1\uparrow} = norm \cdot \frac{Bb_{2\uparrow} - Db_{2\downarrow}}{a_{2\downarrow}b_{2\uparrow} - a_{2\uparrow}b_{2\downarrow}} \tag{3.77}$$

$$a_{1\downarrow} = norm \cdot \frac{Ab_{2\downarrow} - Cb_{2\uparrow}}{a_{2\uparrow}b_{2\downarrow} - a_{2\downarrow}b_{2\uparrow}} \tag{3.78}$$

$$b_{1\uparrow} = norm \cdot \frac{Ba_{2\uparrow} - Da_{2\downarrow}}{b_{2\downarrow}a_{2\uparrow} - b_{2\uparrow}a_{2\downarrow}} \tag{3.79}$$

$$b_{1\downarrow} = norm \cdot \frac{Aa_{2\downarrow} - Ca_{2\uparrow}}{b_{2\uparrow}a_{2\downarrow} - b_{2\downarrow}a_{2\uparrow}} \tag{3.80}$$

To identify which reservoir state the final state will be projected, the amplitude coefficients are compared to a random number r uniformly distributed between 0 and 1,

$$\begin{aligned} if\ ((|a_{1\uparrow}|^2 + |a_{1\downarrow}|^2) &> r(|a_{1\uparrow}|^2 + |a_{1\downarrow}|^2 + |b_{1\uparrow}|^2 + |b_{1\downarrow}|^2)) \\ |\psi_{k_1-q_o}\rangle &= (a_{1\uparrow}c^\dagger_{k_1-q_o,\uparrow} + a_{1\downarrow}c^\dagger_{k_1-q_o,\downarrow})|vac\rangle \\ else\quad |\psi_{k_1-q_o}\rangle &= (b_{1\uparrow}c^\dagger_{k_1-q_o,\uparrow} + b_{1\downarrow}c^\dagger_{k_1-q_o,\downarrow})|vac\rangle \end{aligned}$$

The approach described above is not used in the MC simulations because it is not possible to integrate analytically the scattering rate S over scattering angle to obtain an expression for the scattering rate between two carriers as a function of their momentum, and a numerical evaluation during the code execution requires tremendous amount of time. Instead, a great simplification is possible if we ignore the terms $\widetilde{V}_{q_o+k_2-k_1}$ representing the electron-electron exchange processes. Then the carriers' spin are not correlated after scattering. Since the spin flip factor $\eta_{k,k+q_o} << 1$ is quite small, the scattering rate can be approximated by the rate given in the previous section for two classical particle scattering rate. Since, without exchange, spin flip process does not correlate spins, both particles' spins can be updated independently

$$|\phi_1\rangle = norm \cdot ((\alpha_i + \eta_{k_i,k_i\pm q_o}\beta_i)c^\dagger_{k_i\pm q_o,\uparrow} + (\beta_i + \eta_{k_i,k_i\pm q_o}\alpha_i)c^\dagger_{k_i\pm q_o,\downarrow})|vac\rangle \tag{3.81}$$

Neglecting the exchange process leads to an overestimation of the scattering rate

because the exchange process would reduce the spin scattering rate by negative interference. Since the electron-electron scattering rate is overestimated, the electron spin flip rate during collisions is also overestimated. However, neglecting exchange processes did not result in significant differences from experiments.

This section has covered how to calculate two particle scattering rates and uncorrelated single particle states of carriers after scattering.

3.5 Pauli Exclusion in Spin polarized Electron Gas

The Pauli exclusion cannot be treated in a spin coherent electron gas as it is treated in a thermalized electron gas. It is not possible to identify indiviual momentum states as being occupied by a number of spin up or spin down carriers. Suppose a carrier is scattered to a state with momentum $\overrightarrow{k}$ and the carrier's spin state after scattering would be $|\phi_k\rangle$ if this state were unoccupied. Suppose also that the momentum state at $\overrightarrow{k}$ is occupied with probability $f_{w,k}$ with electrons in spin states $|w_k\rangle$

$$\widehat{\rho}_k = \sum f_{w,k}|w_k\rangle\langle w_k| \tag{3.82}$$

The overlap between the wavefunctions of two carriers determines the orthogonality of their states, and the likelihood that the electron scattering to this state will take place is modified by

$$1 - \sum_w f_{w,k}|\langle\phi_k|w_k\rangle|^2 = 1 - \langle\phi_k|\widehat{\rho}_k|\phi_k\rangle \tag{3.83}$$

In MC simulations, the smallest resolution Δk used to store fillings in the momentum space is limited by the memory requirements. For a given carrier density n and number of simulated electrons N, the maximum occupancy of a momentum space of volume Δk^3 is given by

$$N_{max} = \Delta k_x \cdot \Delta k_y \cdot \Delta k_z \frac{V}{(2\pi)^3} \tag{3.84}$$

$$V = \frac{N_{simulation}}{N_e} \tag{3.85}$$

When the ensemble spin density matrix of electrons ρ_k at a given momentum state $\overrightarrow{k}$ with volume Δk^3 is computed, each carrier's spin density matrix at that momentum state is given a weight $\frac{1}{N_{max}}$.

Another issue arising due to Pauli Exclusion is that the excited spin polarized electrons form a layer of gas above the occupied energy levels of the unpolarized electrons at low enough temperatures. In the experiments, the percentage of electrons excited by the optical pulse is kept small relative to the background electron density in order to generate a small number of holes and annihilate them in a short period of time. The number of Coulomb scattering processes per unit time scale as N^2 and these processes determine the computation time in the simulations. Assigning a large number of carriers to represent background distribution is not feasible. Instead background carriers are considered to have an unperturbed distribution, and only their influence in the momentum space occupancy, Pauli exclusion, and their screening effect is taken into account in the MC simulations. Inclusion of these particular effects do not require any additional particles, but results in much better agreement in the distribution of excited polarized carriers with the experiments. A fermi level is obtained for the equilibrium distribution of carriers. The following rejection technique is used after each collision to implement Pauli exclusion due to background carriers

$$\begin{aligned} f_e(E) &= \tfrac{1}{exp((E-E_f)/k_BT)+1} \\ \textbf{if}(f_e < r) &\quad \text{scatter particle} \\ \textbf{else} &\quad \text{do not scatter} \end{aligned}$$

where r is a random number uniformly distributed between 0 and 1. At low enough temperatures where scattering by excitation of carriers from the well defined Fermi surface becomes important, this approximation might be inaccurate.

3.6 Modelling Spin Dynamics of coherent processes in momentum space: Electron-Electron Coulomb Scattering

So far, the incoherent interaction effects in momentum space are considered in terms of Fermi's Golden rule. Coherent processes in momentum space such as the exchange process results in a self consistent spin dependent potential. To see this, consider a non interacting electron Greens function

$$G^o_{\alpha\beta} = \frac{1}{w - \frac{p^2}{2m} + i\eta}\delta_{\alpha\beta} \tag{3.86}$$

where α and β are spin indices. The exchange term results in the following expression for the propagator via Dyson's equation under the spin dependent scattering potential $T_{\gamma_1\gamma_2}$ diagonal in momentum eigenspace

$$G_{\alpha\beta}(k) = G^o_{\alpha\beta}(k) + \sum_{\gamma_1\gamma_2} G^o_{\alpha\gamma_1}(k)(T_{\gamma_1\gamma_2}(k) + \int d^3k V_{q=0} G^{Ro}_{\gamma_1\gamma_2}(k))G_{\gamma_2\beta}(k) \tag{3.87}$$

which shows that the effect of exchange interaction leads to a spin dependent potential of the form

$$\widetilde{V}_{\gamma_1\gamma_2}(k) = T_{\gamma_1\gamma_2}(k) + V_{q=0}\rho_{\gamma_1\gamma_2} \tag{3.88}$$

where $\rho_{\gamma_1\gamma_2}$ is the spin density matrix. This spin dependent potential can also be included in the MC simulations. It acts similar to a magnetic field due to internal spin polarization, and increases spin lifetime. Since the number of spin polarized carriers is relatively small with respect to the background carries in the experiments, this effect is quite small.

3.7 Implementation of Monte Carlo Algorithm

This section discusses some details of the algorithm assuming a basic knowledge of Monte Carlo Methods. The reader is referred to many excellent reviews on Monte Carlo Simulation techniques in the literature for other implementation details [33] [42] [43].

3.7.1 Distribution of Carriers in momentum and spin space

Given the doping concentration and density of excited carriers in the input file, the code distributes carriers due to dopants according to a Fermi Distribution and the optically excited carriers according to a Gaussian distribution with peak and mean given by the energy and linewidth of laser pulse,

$$f_{exc.}(E) = norm \cdot \exp(\frac{E - E_{peak}}{2\Gamma_{pulse}}) \tag{3.89}$$

where $norm$ is calculated such that

$$N_{exc} \equiv Total\, number\, of\, excited\, carriers = \sum_{\{E_i\}} f_{exc.}(E_i) \tag{3.90}$$

and $\{E_i\}$'s are the energy grid elements. The carriers due to dopants are distributed to yield a zero ensemble polarization, while the excited carriers are initially polarized along a specified direction. If the background carrier density is much larger than the number of excited carriers, as mentioned in the section on Pauli exclusion, the background carriers do not participate directly in collisions and are just included to model Pauli exclusion and screening. In the expression for screening, the density of electrons is equal to the sum of the background and excited electron density.

3.7.2 Ensemble Averages

In order to generate a zero mean ensemble momentum average initially, the mean carrier momentum is subtracted from each carrier's momentum. The code calculates

the ensemble spin density matrix and average energy at specified intervals simply by summing over these over all particles.

$$\rho_{\sigma\sigma'} = \sum_{i=1..N} c_{i,\sigma} c^*_{i,\sigma'} \tag{3.91}$$

3.7.3 Propagation of Spin Between Two Consecutive collisions

The carrier spin rotates due to spin-orbit coupling between the two consecutive collisions. The spin eigenstates are polarized along the wave vector of the carrier as shown in the section on spin splitting. This yields the following 2X2 Hamiltonian

$$H = \Delta E_s \overrightarrow{k} \cdot \overrightarrow{\sigma}_{pauli} \tag{3.92}$$

where

$$\Delta E_s = \delta \cdot [k^2(k^2{}_x k^2{}_y + k^2{}_x k^2{}_z + k^2{}_y k^2{}_z) - 9k^2{}_x k^2{}_y k^2{}_z]^{1/2} \tag{3.93}$$

and $\delta = 20.9, 9.0 eV\mathring{A}^3$ for GaAs and InP respectively ??. The eigenvalues of the 2X2 Hamiltonian are

$$E_{1,2} = \frac{1}{2}(H_{11} + H_{22} \pm \sqrt{(H_{11} + H_{22})^2 - 4(H_{11}H_{22} - |H_{12}|^2)}\,) \tag{3.94}$$

and the corresponding eigenstates are computed by

$$v_{1,2} = \begin{pmatrix} 1 \\ \frac{H_{21}}{e_{1,2} - H_{22}} \end{pmatrix} \tag{3.95}$$

$$v_{1,2} = \frac{v_{1,2}}{v^\dagger_{1,2} \cdot v_{1,2}} \tag{3.96}$$

The projections of spin state c of electron to the eigenstates are calculated. The projection amplitudes propagate simply by a phase factor, and evolved spin state can

be constructed from the projection amplitudes simply by

$$c_{t_{final}} = (c^{\dagger}_{t_{initial}} \cdot v_1)e^{-i\frac{E_1(t_{final}-t_{initial})}{\hbar}}v_1 + (c^{\dagger}_{t_{initial}} \cdot v_2)e^{-i\frac{E_2(t_{final}-t_{initial})}{\hbar}}v_2 \quad (3.97)$$

3.7.4 Particle-Particle Scattering Times

The upper bound on the collision rate of two carriers was calculated in the section on electron-electron interaction, and it is repeated here,

$$\Gamma_{e-e,max}(\overrightarrow{k_o}) = \frac{mq^4 N_e}{8\pi(\epsilon\epsilon_o)^2\hbar^3\beta^3} \quad (3.98)$$

The rate that a collision occurs in an ensemble of N electrons is bounded by this rate times $N(N-1)/2$.

$$\Gamma_{e-e,ensemble} = \frac{N(N-1)}{2}\Gamma_{e-e,max} \quad (3.99)$$

An ensemble e-e scattering time is computed from this rate simply by

$$t_{e-e} = -\frac{\ln(r)}{\Gamma_{e-e,ensemble}} \quad (3.100)$$

where r is a random number uniformly distributed between 0 and 1. At the collision time, two carriers are selected at random from the ensemble of N electrons, and the collision process is accepted if

$$r'\Gamma_{e-e,max} < \lambda(|\overrightarrow{k}_1 - \overrightarrow{k}_2|) \quad (3.101)$$

where, r' is again a random number uniformly distributed between 0 and 1, and $\lambda(|\overrightarrow{k}_1 - \overrightarrow{k}_2|)$ is the scattering rate between the electrons with k_1, k_2 wavevectors. If the collision process is accepted the spins of both carriers are propagated until the collision time and the wave vectors are updated as described in the section on electron-electron interactions.

3.7.5 Impurity, Acoustic and Optical Phonon Scattering Times

The single particle scattering times t_e for each particle are independent of the ensemble e-e scattering rate, and are selected at random according to the maximum rate $\Gamma_{max} = 1e^{-15} sec$ that a carrier would be scattered by a scatterer

$$t_e = -\frac{\ln(r)}{\Gamma_{max}} \tag{3.102}$$

A scattering time for each particle is stored in a linked list. The list is ordered according to the scattering times of carriers. At the time given by first element of the list, the spin state of the first particle of the list is propagated in time. The first particle's scattering rate for all processes is computed as described in the section on scattering processes. A process is selected at random

if $(\Gamma_{impurity} > \frac{r}{t_e})$
select impurity scattering
else if $((\Gamma_{impurity} + \Gamma_{ac-phonon}) > \frac{r}{t_e})$
select acoustic phonon scattering
else if $((\Gamma_{impurity} + \Gamma_{ac-phonon} + \Gamma_{op-phonon}) > \frac{r}{t_e})$
select optical phonon scattering
else no self scattering

The particle's momentum is updated as described in the section on impurity and phonon scattering processes. If the self scattering process is selected, no momentum update is done. The scattering list is updated by assigning the scattered particle a new collision time, and by re-inserting it to the list according to its new scattering time.

3.7.6 Object Oriented Programming

Object Oriented Programming is used in order to decrease the amount of time spent on coding, and to increase the modularity of the code. In a short period of time, the code can be extended to include many variety of situations such as spatial dependencies, time dependent electric fields, and self-consistent potentials. The following classes are used, "vector3D, counter.hpp, complex.hpp, matrix.hpp, electrongas.hpp, scatlist.hpp". The vector3D, complex, and matrix classes handle 3 dimensional vector, complex number and matrix operations. The class electrongas.hpp methods include propagating spin state, selecting scattering process and scattering particle, spin flip process during e-e collisions. The scatlist.hpp class methods include ordering its elements according to their scattering time, returning its first element and its scattering time, and updating the scattering list by assigning a new time to the first particle.

3.8 Simulations

3.8.1 Spin Splitting Energy as a Function of Momentum

The spin splitting energy is an anisotropic function of momentum as shown in the section on the band structure. Some momentum states have negligible amount of spin splitting and the spin precession rate is slow at these states, and some momentum states have large spin splitting, and the spin precession rate is fast at these states. At a given energy, the amount of spin splitting as a function of momentum vector is shown in the 3D plot 3-1.

3.8.2 Initial Excitation Conditions

The initial excitation energy in simulations is selected to be as that of J. M. Kikkawa and D. D. Awschalom's experiments [15], $E_{exc} = 14meV$. The bandwidth is taken as $BW = 7meV$. The influence of the excitation energy and the bandwidth is appreciable only during the short thermalization period in the simulations, and the higher the excitation energy the faster the initial spin polarization decay rate becomes.

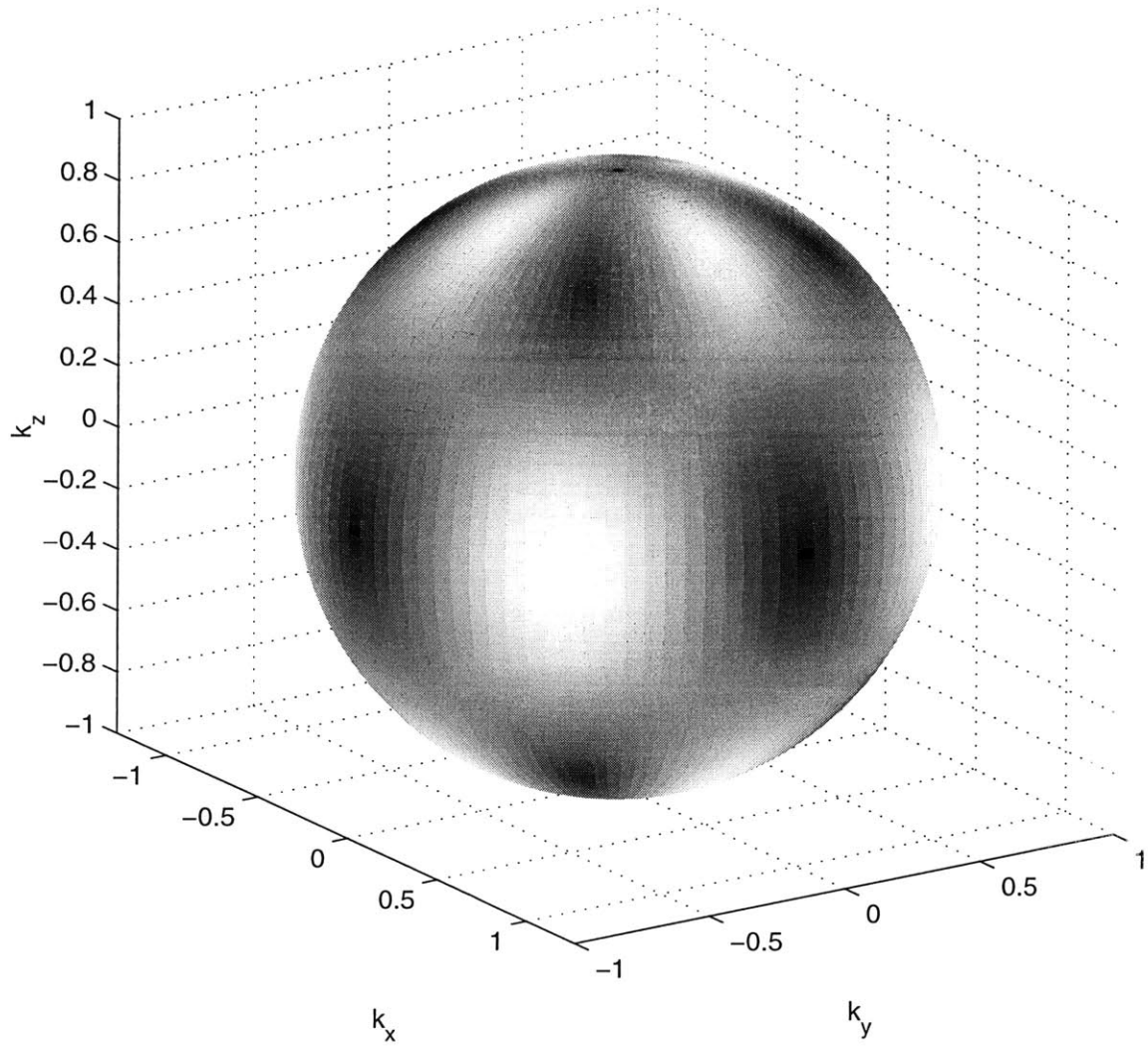

Figure 3-1: Spin splitting as a function of the direction of momentum vector at constant energy. The amount of spin splitting increases from dark to lighter regions.

3.8.3 Electron Thermalization and Energy Relaxation

Electron thermalization takes place in subpicosecond time scale, and leads to a Fermi distribution of carriers as shown in figure 3-2. The initially excited electron gas equilibrates with the lattice via optical phonon scattering in hundreds of picoseconds as shown in figure 3-3.

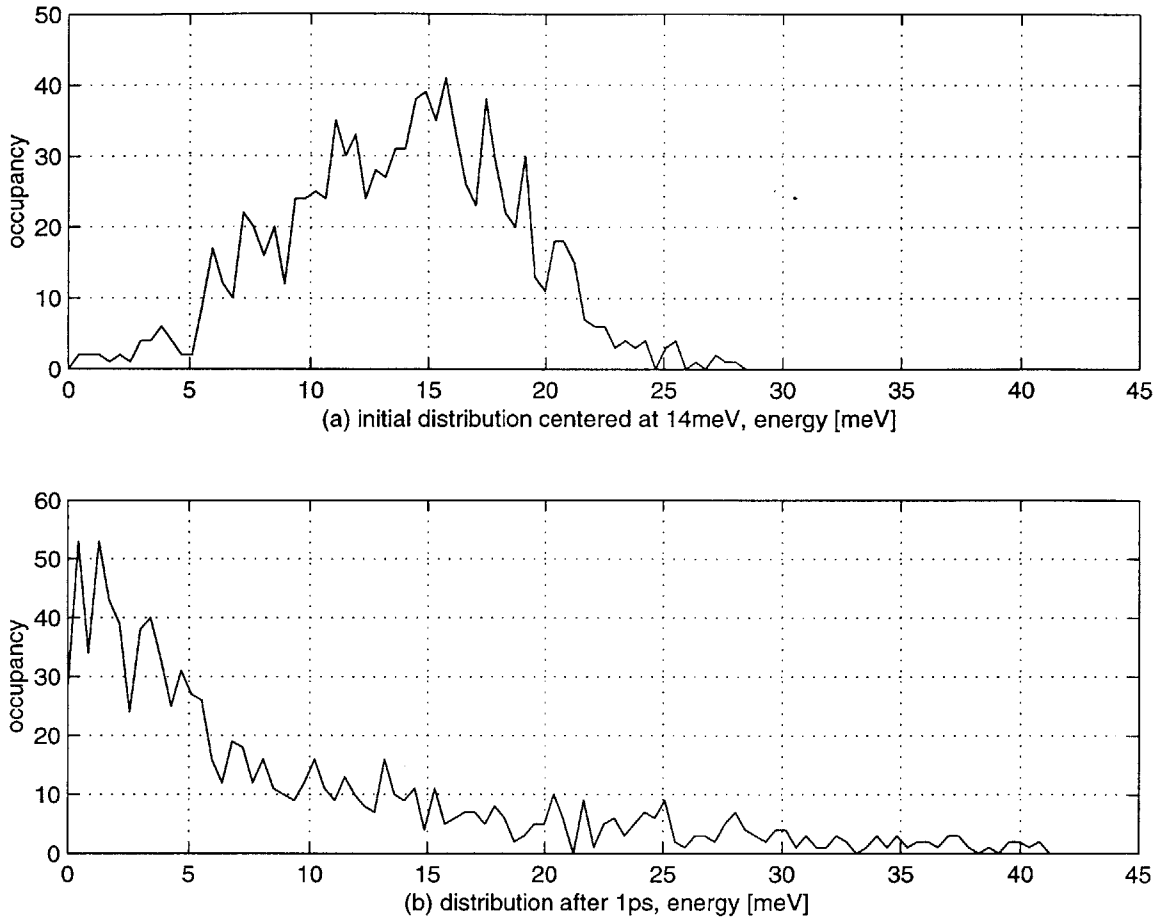

Figure 3-2: Thermalization takes place in subpicoseconds, 1000 electrons simulated at 100K, initial excitation energy 14meV, bandwidth of pulse 7meV

3.8.4 Simulations and Comparison with experiments

The spin lifetime is calculated from MC simulations as a function of the lattice temperature and compared with the results of Kikkawa and Awschalom's experiments [15] in figure 3-4. At low temperatures ($T < 30K$), the excitations close to the fermi surface that were ignored by assuming a static background of carriers become significant, and the simulation underestimates the number of collisions resulting in much shorter lifetime due to underestimated rate of DP mechanism. However, at higher temperatures ($T > 30K$), the simulation agrees well with experiments since these excitations are negligible. Due to the lack of the available experimental data, the higher temperature ($T > 150K$) spin lifetimes are not comparable with experiments, but the agreement is expected to be good due to the expected absence of any

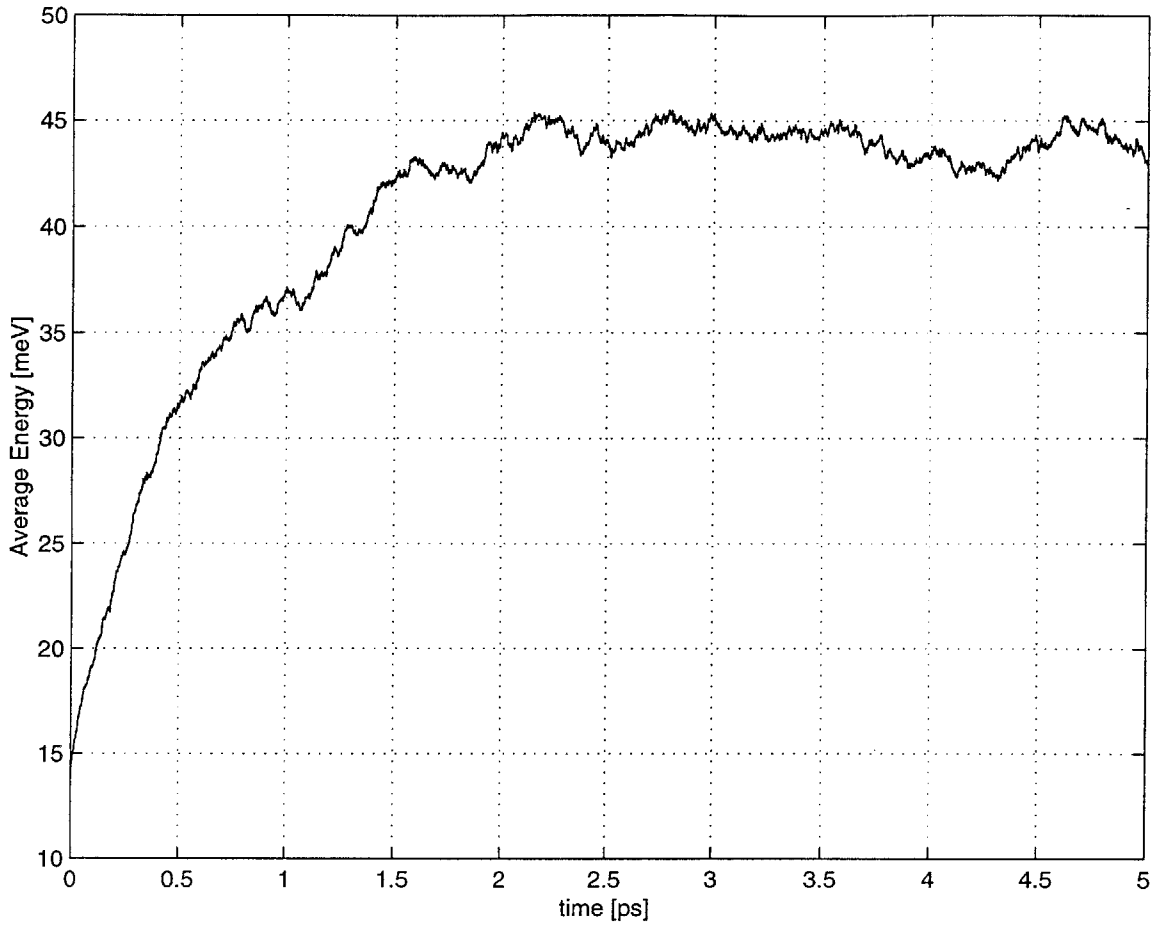

Figure 3-3: The mean kinetic energy of electron gas evolves from the excitation peak energy at $E_{exc} = 14meV$ towards that of thermal equilibrium electron gas $3k_BT/2 \sim 40meV$ (a little more than $3k_BT/2$ because of Pauli exclusion) ; $N = 1000$ electrons simulated

new artifacts/processes that could appear at higher temperatures. All simulations are done with $N = 300$ electrons, but a few points are calculated with larger N, and it is observed that the agreement between the simulations and experiments is getting better as N increases.

3.8.5 Efficiency of various processes

At high temperatures the carriers on average have larger momentum vectors than that of carriers at the lower temperatures. Since spin splitting is large for high momentum vectors, the average spin precession rate is faster at higher temperatures than that at

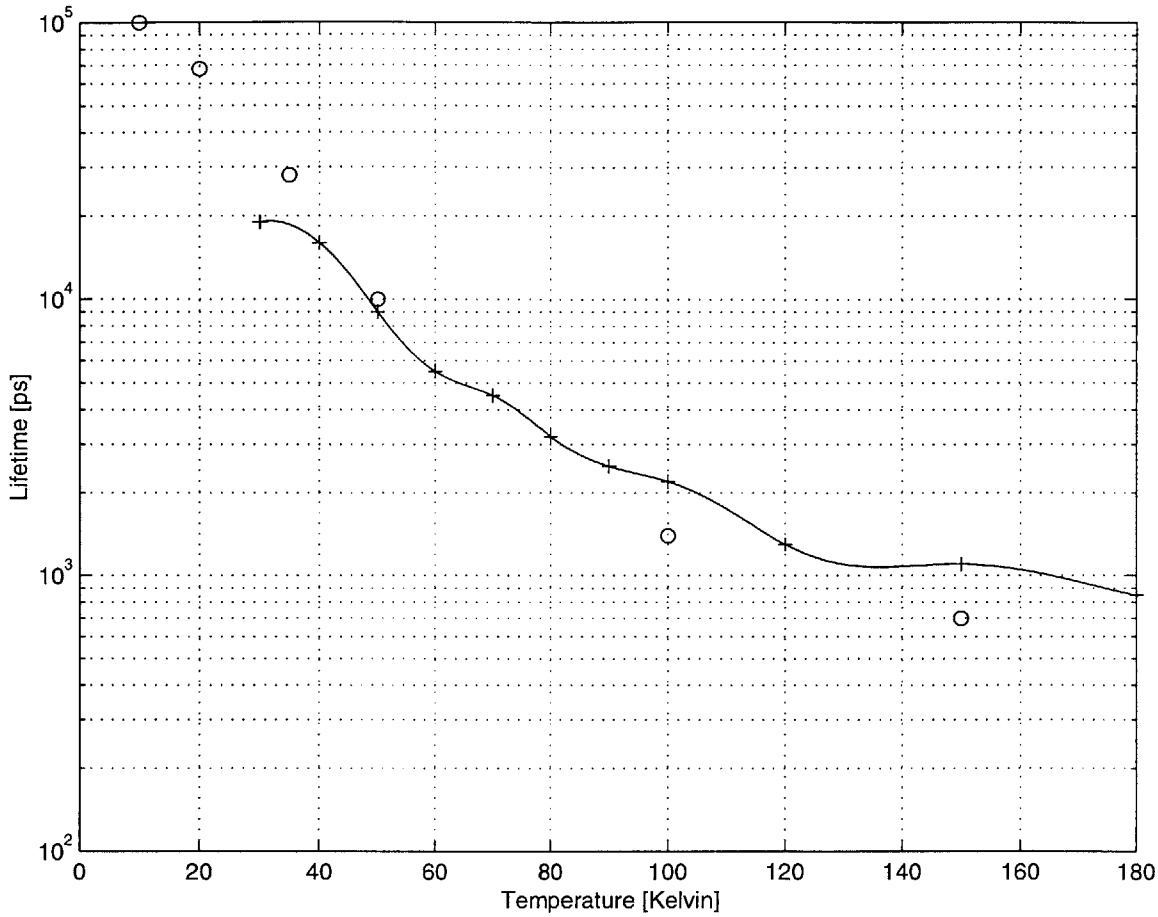

Figure 3-4: Temperature versus spin lifetime: simulation '+' and experimental points 'o'

lower temperatures (figure 3-5).

For comparison, some scattering processes and spin dephasing mechanisms are turned off and the spin lifetimes are plotted as a function of temperature in figure 3-6.

All the simulations reported in the other sections assume a non-magnetic impurity concentration due to the dopants, thus the number of impurities is equal to the number of dopants. The non-magnetic impurity scattering effects the spin dephasing rate via by both EY and DP mechanism. However, inclusion of both impurity and phonon scattering without electron-electron scattering (compare plots (b) and (c)) underestimates the spin lifetimes significantly in contradiction with the predictions made in the literature. Electron-electron scattering is the most significant scattering

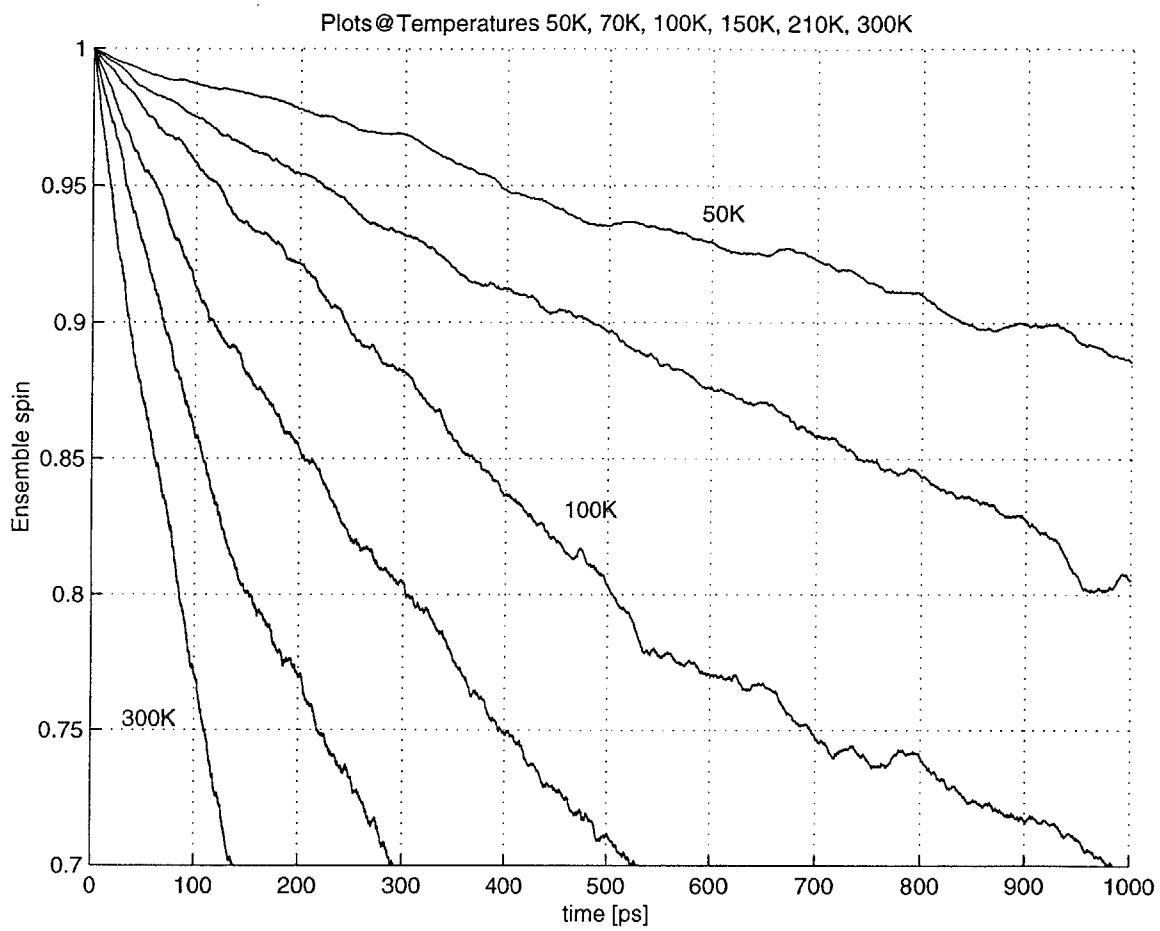

Figure 3-5: Decay of ensemble spin polarization at different lattice temperatures

mechanism, and dominates the dependencies of spin lifetimes both at high and at low temperatures. There are four major ways that electron density influences spin dynamics. Increasing electron density:

◇ increases electron-electron momentum scattering rate, and via DP mechanism decreases spin dephasing rate

◇ increases spin-orbit scattering rate via EY mechanism, leading to faster spin dephasing rate

◇ decreases the scattering rates due to Pauli exclusion

◇ increases the hole annihilation rate, decreasing hole-electron exchange spin coupling leading to lower spin dephasing rate

In order to understand the relative strength of EY and DP mechanism, EY mech-

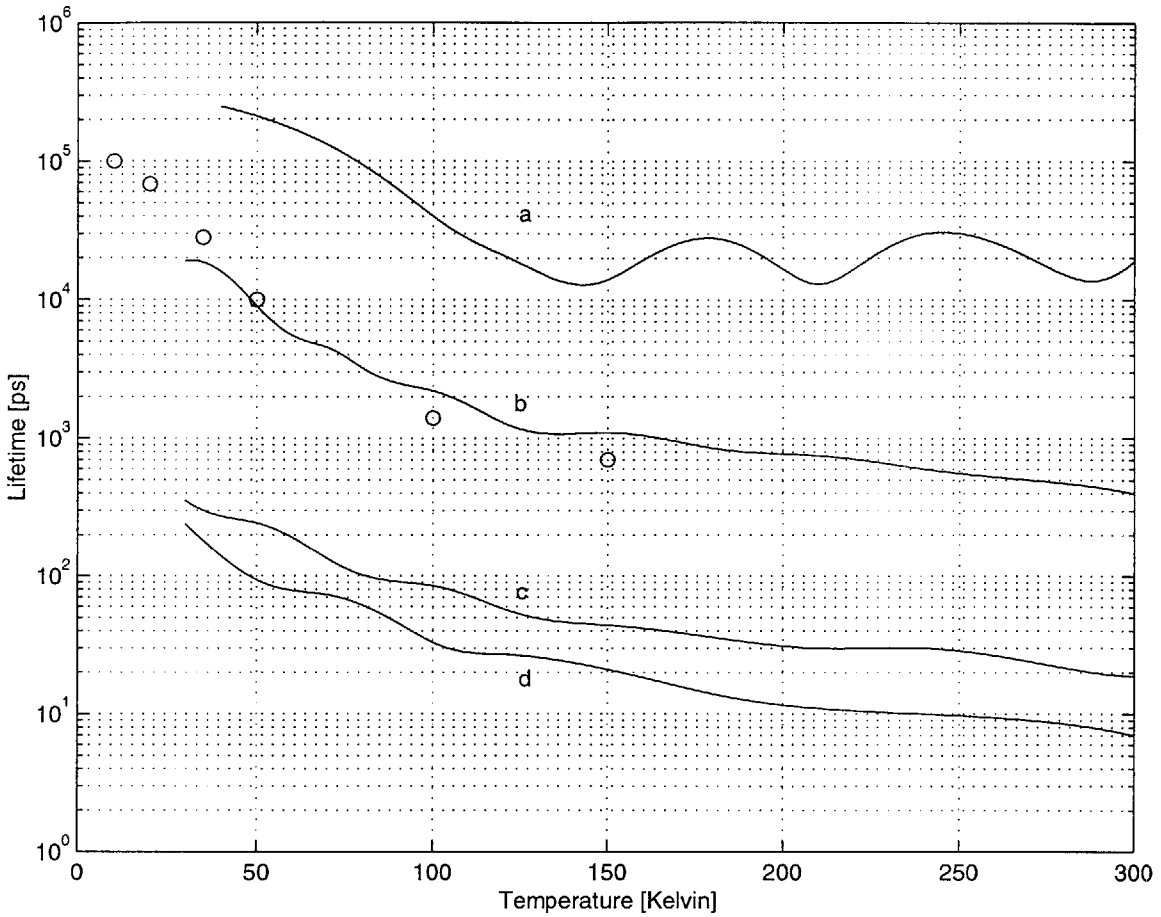

Figure 3-6: Plots of spin lifetime versus temperature: (a) spin scattering during collision is ignored (no EY), (b) best fit to experiments due to inclusion of all EY and DP mechanisms, (c) only impurity and phonon scattering is active, no electron-electron scattering, (d) no scattering processes, spin polarization decays due to the difference in the precession rate of carriers at different momentum vectors

anism is turned off by setting the η Bloch function overlap factor to zero. The temperature dependence of spin lifetime without the EY mechanism is shown in plot (a) of figure 3-6. The EY mechanism is not negligible at any temperature, neither the DP mechanism. Both mechanisms influence the order of magnitude values of spin lifetimes (compare plots (b) and (c) of figure 3-6). The oscillatory artifacts in plot (a) are because of the small signal to noise ratio in the MC simulations when the spin lifetime is very long (as is the case when EY is removed in plot (a)), and can be improved by using more carriers in the simulations.

Chapter 4

Probing Optically Electron Spin Dynamics

For semiconductors optical methods make it possible to study electron spin dynamics despite the background of other magnetic processes (eg. scattering with holes, magnetic impurities). Optical techniques enable the study of various factors that effect spin lifetimes, including impurity and electron concentration, temperature, interface, magnetic field, spin orbit coupling and particle interactions. Previous optical studies [15] [14] have concentrated on electron spin dynamics in GaAs bulk and heterostructures. In bulk semiconductors, spin lifetimes are sufficiently long for the operation of possible spin effect devices. In heterostructures, because of the enhanced spin orbit coupling due to interfaces [24] [28], spin lifetimes are significantly shorter with respect to that of bulk.

Photons carry angular momentum equivalent to twice the spin angular momentum of the electron. They have two angular momentum eigenstates directed along the propagation direction of electromagnetic wave, and are called left and right circular polarized states. Photons can exist in a linear superposition of these two eigenstates such as vertical or horizontal linearly polarized states. To convert a circularly polarized photon to a linearly polarized photon requires a linear polarizer or a quarter wave plate, and the reverse requires a quarter wave plate. Two quarter wave plates or a half wave plate can rotate a linearly polarized photon to an orthogonal linearly

polarized photon.

Due to the angular momentum conservation, when a photon is absorbed/emitted by an electron, photon's angular momentum $\hbar/2$ is also added/subtracted to/from that of the electron. The heavy and light hole valence bands of III-V semiconductors is degenerate at the Γ point. An optical pulse excites carriers from heavy and light hole valence bands with a ratio of 3:1. Due to angular momentum conservation, a circularly polarized photon with angular momentum $\hbar\hat{z}$ can excite an electron from the **heavy** hole valence band with $-\frac{\hbar}{2}\hat{z}$ angular momentum to the conduction band with $\frac{\hbar}{2}\hat{z}$ angular momentum. However, circularly polarized photon with angular momentum $\hbar\hat{z}$ can excite an electron from the **light** hole valence band with angular momentum of either $-\frac{3\hbar}{2}\hat{z}$ or $-\frac{\hbar}{2}\hat{z}$ to the conduction band state of angular momentum $-\frac{\hbar}{2}\hat{z}$ or $\frac{\hbar}{2}\hat{z}$ respectively. When an electron relaxes back to the valence band, it emits a photon circularly polarized in the direction parallel to the electron's initial spin polarization, and the spin polarization of the electron changes by $\frac{\hbar}{2}$.

Thus, one can selectively excite spin up or down polarized electrons to the conduction band from the valence band by left or right circularly polarized light. In the pump-probe experiments performed in this thesis, a polarized pump pulse is used to excite spin polarized electrons to the conduction band, and a polarized probe beam is used to probe the spin polarization states of electrons. When the polarized probe beam interacts with the sample, its transmission is modified due to three processes:

◇ The probe pulse can be amplified by stimulated emission of photons polarized in the same direction as that of the probe. The stimulated emission of polarized photons depends on the existence of electrons in the conduction band with the appropriate spin polarization direction as that of probe pulse.

◇ The absorption of the probe pulse can be diminished if no electrons in the valence band with opposite spin polarization exist. In other words, existence of holes with same spin polarization direction with probe enhances the transmission of probe pulse.

◇ The absorption of the probe pulse can be diminished if electrons with same spin polarization in the conduction band block the excitation of new carriers from the

valence band.

All three processes increase the transmission rate of the probe pulse. However, the first process depends on the phase of the spins of electrons in the conduction band and the second depends on the phase of the spin of holes in the valence band. Since the spin dephasing time of holes is much faster than that of electrons, on sufficiently long time scales the change in the transmission of the probe pulse is entirely due to the dynamics of the ensemble spin polarization of the electrons in the conduction band. The third process is also very weak because of the small number of carriers excited and their quick thermalization to form a nondegenerate gas. Thus the change in the transmission is dominated by the first process.

The time resolution in the pump-probe experiments is limited by the width of the pump and the probe pulses which can be selected to be either order of tens of picoseconds or hundreds of femtoseconds. The 5 picosecond probe width used in the experiments is short enough to probe the evolution of spin dynamics in bulk semiconductors and the pulse shape can be approximated as a delta function in time, however femtosecond probe pulses are necessary to probe the spin dynamics in quantum well structures because spin dephasing occurs on picosecond timescales [28].

To measure the spin lifetime of carriers in semiconductors, the transmission rate of the polarized probe pulse is obtained as a function of the delay between the polarized pump and probe pulses. When the probe pulse passes through the sample, its transmission rate depends on the present state of the carriers. The delay between the pump and the probe is adjusted by a mechanical translation stage and the maximum delay is about one nanoseconds. The spin lifetimes of electrons can be as short as picoseconds and as long as nanoseconds. For short lifetime processes the range of mechanical delay is enough to probe the spin dynamics, but for long lifetime processes Faraday rotation technique [15] can be employed to extract lifetimes.

4.1 Lifetimes

The rate of transmission of probe pulse is recorded as a function of the delay between the probe and pump pulses. The initial discontinuity in the transmission rate is the point of zero delay. In order to extract each lifetime, one can model the dynamics in terms of rate equations to obtain a functional form for the transmission curve, and then one can extract lifetimes of individual processes. In the experiments and timescales conducted to observe spin dynamics, the transmission is governed by two processes, electron-hole recombination and momentum dependent spin rotation. Since the excited carrier density is quite low and produces a small number of holes, the recombination process does not influence strongly the population of spin polarized carriers.

$$\frac{1}{\tau_{rec}} \approx 10^{-10} cm^{-3} s^{-1} \sqrt{N_{exc} P_{exc}} \tag{4.1}$$

$$\approx 10^{4} s^{-1} \tag{4.2}$$

Hence the spin polarized probe transmission only depends on the spin dynamics of carriers, and when the lifetime of the spin polarized carriers is long enough, the transmission curve can be fit simply to a linear curve as a function of delay,

$$e^{-\frac{t}{\tau_{spin}}} \simeq 1 - \frac{t}{\tau_{spin}} \tag{4.3}$$

If the recombination process influences the number of spin polarized carriers due to a large enough background hole distribution, but the hole distribution is stationary, the tranmission can be modeled by two linear rate equations,

$$\frac{dN_{\uparrow}}{dt} = \frac{N_{\downarrow}}{\tau_{spin}} - (\frac{1}{\tau_{e-h}} + \frac{1}{\tau_{spin}})N_{\uparrow} \tag{4.4}$$

$$\frac{dN_{\downarrow}}{dt} = \frac{N_{\uparrow}}{\tau_{spin}} - (\frac{1}{\tau_{e-h}} + \frac{1}{\tau_{spin}})N_{\downarrow} \tag{4.5}$$

The two equations can be combined to obtain an equation for $N_\uparrow$

$$\frac{d^2N_\uparrow}{dt^2} = -2(\frac{1}{\tau_{e-h}} + \frac{1}{\tau_{spin}})\frac{dN_\uparrow}{dt} - (\frac{1}{\tau_{e-h}^2} + \frac{2}{\tau_{spin}\tau_{e-h}})N_\uparrow \tag{4.6}$$

Substituting a solution of the form

$$N_\uparrow = c_1 e^{-\lambda_1 t} + c_2 e^{-\lambda_2 t} \tag{4.7}$$

with $\lambda_1 > \lambda_2$ yields

$$\frac{1}{\tau_{spin}} = \frac{\lambda_1 - \lambda_2}{2} \tag{4.8}$$

$$\frac{1}{\tau_{e-h}} = \lambda_2 \tag{4.9}$$

The electron hole recombination lifetime τ_{e-h} can be determined independent of the spin relaxation time by linearly polarized pump-probe experiment. Given τ_{e-h} the following equation is fit to the experimental curve to obtain τ_{spin}

$$\text{Transmission} \sim c_1 e^{-\frac{t}{\tau_{e-h}}} + c_2 e^{-t(\frac{1}{\tau_{e-h}} + \frac{2}{\tau_{spin}})} \tag{4.10}$$

by adjusting c_1, c_2 and τ_{spin}.

4.2 Observation of Slow Spin Dynamics

For long lifetime processes, the range of mechanical delay is not enough to probe the dynamics of carriers. To observe long spin lifetimes Faraday rotation technique can be used where an external magnetic field B applied transverse to the initial spin excitation direction causes the spins of the excited electrons to rotate [15]. Since the probe field is also polarized, its amplification while passing through the sample depends on the relative spin states of excited electrons. A pump probe delay of τ will cause amplification in the probe beam by an amount $Ae^{-\tau/T_2}cos(g\mu_B B\tau/\hbar)$. However, because of the repetition of laser pulses and long spin lifetime, excited spins

do not relax appreciably between two pump signals and one has take into account the spin memory effect due to previous pump excitations. In order to obtain the highest faraday effect, the magnetic field can be adjusted such that electron spins rotate exactly an integer multiple of full rotations of 2π in one repetition time of t_{rep}. This results in the constructive addition of spins excited by multiple pump signals. It enables measurement of long spin dephasing lifetimes. The signal appears as [15]:

$$\sum_n \Theta(\tau + nt_{rep}) A e^{-(\tau_{spin}+nt_{rep})/T_2} cos(g\mu_B B(\tau + nt_{rep})/\hbar) \tag{4.11}$$

where n, g and μ_B respectively is an integer number, electron spin factor and Bohr Magneton. The data can be fit by adjusting τ_{spin}, A, and g. In the experiments conducted for this thesis, Faraday rotation technique is not used.

4.3 Design of Measurement Apparatus

The setup shown in the figure 4-1 is constructed and used in pump-probe experiments [11]. The system works as follows: A tunable Ti/Sapphire laser provides $5ps$ pulses with repetition times of ${t_{rep}}^{-1} = 80MHz$. Pulses are split into two by a beam splitter having ultrafast response time and pump pulse passes through a computer controlled translation stage to introduce delay with a spatial resolution of $0.1\mu m$ corresponding to $0.4fs$ temporal resolution. The probe pulse passes through a manual translation stage with a spatial resolution of $1\mu m$. Both pulses are polarized by quarter wave plates having ultrafast response times. In order to increase the signal-to-noise ratio, a phase locked detection scheme is used. The Ti/Sapphire laser has noise components up to $9Mhz$ due to oscillations in its Argon ion pump source. Pump pulses are chopped by an anti reflection coated acousto-optical modulator at $10Mhz$ in order to prevent intrinsic laser noise. Due to the heating effects at the sample, low frequency noise below $1kHz$ is added to the probe signal. In order to cancel this noise, the probe pulse is chopped by a low frequency mechanical chopper. The pump and probe pulses are incident onto the sample surface at slightly different angles so that the

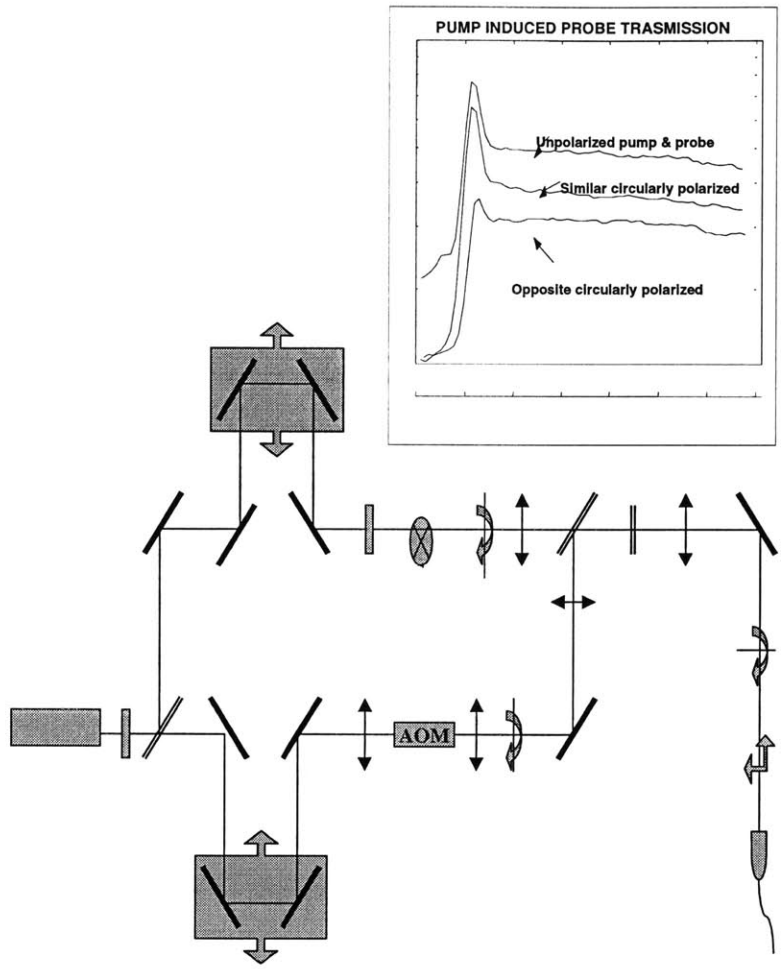

Figure 4-1: Pump-probe setup used to study electron spin dynamics, inset shows a measurement done on a GaAs/AlGaAs 80Åwidth n-type Quantum Well

probe beam can be measured without a background due to pump. The probe pulse is filtered by the combination of circular and linear polarizers. A low noise photo detector with $125MHz$ response time is used to detect the probe pulse. Computer control of translation stage provides a fast scan to prevent any noise due to laser drift. Reference frequencies are supplied to the lock-in amplifier from both the mechanical and the acousto-optical chopper. The lock-in amplifier takes the ratio of the two signals to obtain a reference signal at $10MHz + 1KHz$. At data sample points corresponding to the positions of the translation stage, a trigger signal is sent from the computer board to the lock-in amplifier to sample data from the photo detector. After one full scan of the position stage, sample points from the lock-in amplifier are

downloaded for further analysis. The lenses plus the mirrors are aligned to introduce no significant change in the direction and overlap of the pump and probe beams on the detector over the full motion of position stage.

The position stage is calibrated with respect to the zero delay point. The zero delay position can be obtained from two-photon absorption in a GaAs sample or from initial carrier bleaching. A few experiments are conducted on quantum wells with varying width, and results are found to be in good agreement with previously reported measurements [28]. No measurements are yet conducted on bulk semiconductors due to the limited amount of time available for the completion of thesis, and Monte-Carlo simulations are compared with those of earlier bulk spin lifetime measurements reported in the literature [15].

Chapter 5

Conclusion

In this thesis, electron spin dynamics in III-V semiconductors is studied by femtosecond pump-probe spectroscopy and Monte-Carlo simulations. It is found that the major spin dephasing mechanism is due to the momentum dependent spin splitting in the conduction band, and this spin dephasing mechanism is suppressed via D'yakanov-Perel (DP) mechanism by "motional narrowing". It is found that DP mechanism by itself is not sufficient to estimate measured lifetimes, and Elliott-Yafet (EY) mechanism should also be included at all temperatures. It is also shown that impurity and phonon scattering processes play a minor role with respect to the electron-electron scattering processes. Simulation results agreed with the experiments better than those reported in the literature within the temperature ranges considered. These results show that both EY and DP are significant at all temperatures, and electron-electron scattering is most significant scattering process for the long spin lifetimes measured in experiments [15].

Appendix A

All-Metal Spin Transistors

Metal spin transistors were first demonstrated by Mark Johnson of Bell Labs [6]. Their operation principle is different than proposed semiconductor spin transistors. As shown in figure 3b, a ferromagnetic emitter injects spin polarized current into the paramagnetic metal base. Since the current injected from the ferromagnetic emitter is spin polarized, inequilibrium is maintained between the fermi levels of up and down spin polarized electrons in the base. A ferromagnetic collector acts like a spin filter and probes only the fermi level of the spin population aligned parallel to the magnetization axis of the collector. Thus the output voltage depends on direction of the magnetization axis of the collector. A current controlled voltage source can be made by modulating the collector magnetization by another controlled current carrier. It is also possible possible to control collector voltage by modulating the emitter-base current. A simple relationship between the measured signal V_o and the input emitter-base current I_e can be obtained [8]. Let's pick the emitter's polarization axis as upwards for simplicity. Then I_e is up spin polarized. The interfacel conductivities in the ferromagnets are given by $g_{\uparrow} \neq g_{\downarrow}$ where $\uparrow, \downarrow$ are the spin polarization directions. The relative Fermi levels of ferromagnets are in equilibrium $E_{F;f\uparrow} = E_{F;f\downarrow} = E_{F;f}$. The spin-up and spin-down fermi levels of paramagnet are not in equilibrium $E_{F;p\uparrow} \neq E_{F;p\downarrow}$.

The electric and magnetic current densities injected from the emitter, assuming a

potential of V_e applied between the emitter and base, are given by

$$J_e = 1/e[g_\uparrow(E_{F;f} - E_{F;p}) + g_\downarrow(E_{F;f} - E_{F;p})] = (g_\uparrow + g_\downarrow)V_e \tag{A.1}$$

$$J_M = 1/e[g_\uparrow(E_{F;f} - E_{F;p}) - g_\downarrow(E_{F;f} - E_{F;p})] = (g_\uparrow - g_\downarrow)V_e \tag{A.2}$$

The ratios of the current densities are then given by

$$J_M/J_e = (g_\uparrow - g_\downarrow)/(g_\uparrow + g_\downarrow) = \eta \tag{A.3}$$

Johnson assumed that the dominant mechanism for the decay of magnetic current density is the spin relaxation in the base, because the lateral dimensions of his transistors was much larger than spin relaxation length. This is not accurate for transistors with small base volumes. The escape of spin polarized carriers out of the base region before they relax is the dominant mechanism, and the spin relaxation in the base area is negligible in the case of base dimensions much smaller than the spin relaxation length. A simple relationship between the difference in fermi energies of up and down spin polarized carriers and the injected current I_e in the base can be obtained for the limiting cases of very small base area and very large base area. We first calculate this relationship for the case of small area. The magnetic current will be proportional to the number of excess spin polarized carriers δn in the base

$$J_M = e < v_{base} > \delta n \tag{A.4}$$

where e is electron's charge and $< v_{base} >$ is the average component of fermi velocity in the direction of current flow. The number of excess spin polarized carriers is related to the difference in the fermi levels of spin up and down polarized electrons. Assuming a uniform density of states near Fermi level gives

$$\delta n = D_\uparrow(E_F)eV_d \tag{A.5}$$

where $D_{\uparrow}(E_F)$ is the density of spin-up states at the fermi energy E_F in the paramagnet, and V_d is the difference in the fermi energies of up and down spin polarized electrons in the base.

Thus, in the case of small base area, the energy difference V_d is related to the injected current density by

$$V_d = \eta/(g_{\uparrow p})J_e/A_e ff \tag{A.6}$$

where $g_{\uparrow p}$ is the conductivity of the up-spin channel in the paramagnetic base.

For the large base area as Johnson's, the magnetic current injected into the base is the source term for decaying number of excess spin polarized electrons and no magnetic current escapes out of base area

$$I_M = \delta n \Omega / T_2 \tag{A.7}$$

where Ω is the volume of base. Then, the difference in the fermi levels of up and down spin electrons in the base is given in terms of I_e as

$$V_d = T_2 \eta/(D_{\uparrow}(E_F)e)I_e/\Omega \tag{A.8}$$

Next, we calculate the relationship betwee the voltages V_d and V_o. The current density in the collector is given by

$$J_e = 1/e[g_{\uparrow}(E_{F;p\uparrow} - E_{F;f}) + g_{\downarrow}(E_{F;p\downarrow} - E_{F;f})] \tag{A.9}$$

Using $E_{F;p\uparrow} + E_{F;p\downarrow} = E_{F;p}$ and $E_{F;p\uparrow} - E_{F;p\downarrow} = eV_d$ gives

$$J_e = 1/e[(g_{\uparrow} + g_{\downarrow})(E_{F;p} - E_{F;f}) + (g_{\uparrow} - g_{\downarrow})eV_d] \tag{A.10}$$

If the collector side has high impedance as in voltage measurements, the collector

current is zero. Identifying η and the output voltage V_o as $E_{F;p} - E_{F;f}$ gives

$$V_o = \eta V_d \tag{A.11}$$

Thus the output voltage as a function of injected current density for small base area is

$$V_o = \eta^2/(g_{\uparrow p})J_e \tag{A.12}$$

and for the large base area

$$V_o = T_2\eta^2/(D_\uparrow(E_F)e)I_e/\Omega \tag{A.13}$$

The relationship between the collector voltage and injected current shows that scaling the volume of the base with fixed emitter current increases the collector voltage.

The previously fabricated devices of Johnson had quite large areas, on the order of $0.01mm^2$ [6]. Since the output voltage gets stronger as the device dimensions shrink, devices with a wide range of area will be fabricated and tested. The smallest devices that can be fabricated will have an area of $1\mu m^2$. This should increase the output signal 10^4 times with respect to the experiments of Johnson [6], reaching signal levels above 10^{-4} Volts under an emitter current I_e of few mA's. Since the area of the base will be much smaller, spin relaxation in the base will not be completed. This implies even higher output signals.

Bibliography

[1] N.F. Mott, Proc. R. Soc. 153 (1936), 699

[2] A. Fert, I.A.Campbell, Phys. Rev. Lett. 21 (1968), 1190

[3] A. Fert, I.A. Campbell, J. Phys. F: Metal Phys. 6 (1975) 849

[4] M. N. Baibich, J.M. Broto, A. Fert, F. Nguyen Van Dau, F. Petroff, Phys. Rev. Lett. 61 (1988), 2472

[5] G. Binasch et al., Phys. Rev. B 39 (1989), 4824

[6] M. Johnson, Science Vol. 260 (1993), 320

[7] M. Johnson, IEEE Spectrum May 1994

[8] M. Johnson, R. H. Silsbee, J. Appl. Phys. 63 (8) (1988), 3934

[9] S. Datta, B. Das, Appl. Phys. Lett. 56 (7) (1990), 665

[10] M. Roukes, submitted paper

[11] Siegfried B. Fleischer, PhD. Thesis MIT (1997)

[12] D.J. Monsma, J.C. Lodder, J.A. Popma, B. Dieny, Phys. Rev. Lett. 74 (1995), 5260

[13] J. Gregg et.al., J. Magn. and Magn. Materials 175 (1997), 1-9

[14] S.A. Crooker, D.D. Awschalom, J.J. Baumberg, F. Flack, N. Samarth, Phys. Rev. B 56 (12) (1997), 7574

[15] J.M. Kikkawa, D.D. Awschalom, Phys. Rev. Lett. 80 (19) (1998), 4313

[16] G. Dresselhaus, Phys. Rev. 100 (1955), 580

[17] P. Boguslawski, Solid State Commun. 33 (1980), 389

[18] W. Zawadzki, W. Szymanska, Phys. Stat. Sol. (b) 45 (1971), 415

[19] R. J. Elliott, Phys. Rev. 96 (2) (1954), 266

[20] Elliott-Yafet

[21] M. I. D'yakonov, V. I. Perel', Sov. Phys. JETP, 33 (1971), 1053

[22] M. I. D'yakonov, V. I. Perel', Sov. Phys.-Solid State 13 (12) (1972), 3023

[23] H. Mayer, U. Rossler, 87 (2) (1993), 81

[24] J. Fabian, S. Das Sarma, J. Vac. Sci. Technol. B 17(4) (1999), 1708

[25] G. L. Bir, A. G. Aronov, G. E. Pikus, Sov. Phys. JETP 42 (1976), 705

[26] J. Wagner et al., Phys. Rev. B 47 (1993), 4786

[27] A. W. Overhauser, Phys. Rev. 89 (4) 1953, 689

[28] M. I. D'yakonov, V. Yu. Kachorovskii, Sov. Phys. Semicond. 20(1) (1986), 110

[29] R. S. Britton et al., Appl. Phys. Lett. 73 (15) (1998), 2140

[30] J. Goree, Rev. Sci. Instrum. 56 (8) (1985), 1662

[31] P. Bado, S. B. Wilson, K. R. Wilson, Rev. Sci. Instrum., 53 (1982), 706

[32] L. Andor et al., Rev. Sci. Instrum 55 (1984), 64

[33] Jasprit Singh, Physics of Semiconductors and their Heterostructures, Ch. 13, p. 432

[34] A. Matulionis et al., Solid State Commun., 16 (1975), 1133

[35] P. Lugli, D. K. Ferry, Physics 117B-118B (1983), 251

[36] R. Brunetti et al., Physica 134B (1985), 369

[37] M. Mosko, A. Moskova, Phys. Rev. B, 44 (1991), 10794

[38] M. Mosko, A. Moskova, Semicond. Sci. Technol. 9 (1994), 478

[39] A. Moskova, M. Mosko, Phys. Rev. B, 49 (1994), 7443

[40] D. Bohm, D. Pines, Phys. Rev., 92 (1953), 609

[41] V. Cambel, M. Mosko, Semicond. Sci. Technol. 9 (1994), 474

[42] C. Jacoboni, L. Reggiani, Rev. Mod. Phys. 55 (3) (1983), 645

[43] C. Jacoboni, P. Lugli, The Monte Carlo Method for Semiconductor Device Simulation, Springer-Verlag Pub.

APPENDIX B:

SAMPLE INPUT:

N	300
tspan	1000
kT	25.5
Ndop	1e16
Nexc	2e14
BW	7.0
Eexc	14
tsample	0.1

SAMPLE OUTPUT:

```
%N 300
%tspan 999.999972 ps
%kT 15.300000 meV
%Ndop 1.000000e+016 cm^-3
%Nexc 2.000000e+014 cm^-3
%BW 7.000000 meV
%Eexc 14.000000 meV
%tsample 0.100000 ps
%Emax 152.999997 meV, kmax 5.183500e+008 [m^-1]
%Debye length =3.307601e-008 [m]
%Scattering lifetimes: single carrier 1.000000e-015 carrier-carrier 6.062423e-019 [s]
%Smallest time step 6.062423e-019; Total number of steps 1.649506e+009
%Number of energy grids 199
%Excitation polarity:
 % 1.000000+i0.000000
 % 0.000000+i0.000000
%Background Fermi Level [eV] -0.061185
%Initial ensemble averaged momentum components:
%kx=-1.112620e-008 ky=3.327926e-009 kz=3.849467e-009
%Maximum momentum in each direction 5.183500e+008
%Size of each cell in momentum space 5.183500e+007
%Number of cells 1000.000000
%Maximum occupancy of each momentum cell 16.513940

%Time vs Collective spin polarization & Average Energy [eV]:
0.000000e+000    1.000000e+000  14.044171
9.999966e-014    9.998994e-001  14.277504
1.999993e-013    9.998074e-001  15.560838
2.999990e-013    9.997064e-001  16.610838
3.999986e-013    9.995931e-001  17.894171
4.999983e-013    9.994544e-001  19.294171
5.999980e-013    9.993291e-001  20.344171
6.999976e-013    9.992009e-001  21.277504
7.999973e-013    9.990642e-001  21.277504
8.999970e-013    9.989090e-001  21.394171
9.999966e-013    9.987506e-001  19.294171
1.099996e-012    9.986642e-001  19.060838
1.199996e-012    9.984895e-001  19.294171
1.299996e-012    9.983374e-001  20.810838
1.399995e-012    9.981577e-001  19.877504
1.499995e-012    9.979979e-001  19.644171
1.599995e-012    9.978772e-001  18.827504
1.699994e-012    9.977367e-001  18.827504
1.799994e-012    9.975736e-001  19.177504

.....
```

GLOBAL VARIABLES:

globals.hpp

```
#ifndef GLOBALS_HPP
#define GLOBALS_HPP


#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

#define pi 3.14159265358979
const double q=1.6e-19;
const double mo=9.109534e-31; //kg
const double mc=0.067*mo; //for GaAs, Corzine p12
const double hbar=1.0545887e-34; //Js
const double sqrhbar=hbar*hbar;
const double epsio=8.854e-12;
const double epsi=epsio*13.18;
const double normrnd=1/double(RAND_MAX);
double kmax;

#include "complex.hpp"
#include "vector3D.hpp"
#include "counter.hpp"
#include "scatlist.hpp"
#include "matrix.hpp"


#endif
```

MAIN FUNCTION:

manyparticle.cpp

```
//ON UNIX COMPILE: g++ manyparticle.cpp -lm
//ON WINDOWS COMPILE: cl manyparticle.cpp -O3 -G6
//USE: cl -? to see other machine dependent options

#include "globals.hpp"
#include "electrongas.hpp"

void print_occE(FILE *fp, double dE, double *occE, int sizeE, counter time, double dt);
void distribute(FILE *fdist, double Ndop, double& Ef, double excitation, double BW, double Eexc, complex *polarity, double kT,
double dE, int sizeE, const int N, electrongas *particle, long int Tsample, double kmax);
void eescatter(electrongas &particle1, electrongas &particle2, double eerate, double eefactor, double Ef, double sqrbeta, double kT,
counter tpresent, double dt);
void compute_occE(double dE, double *occE, int sizeE, int N, electrongas particle[]);

void main(void)
{
 srand((unsigned)time( NULL ) ); //seed the random number generator: DO NOT USE IF YOU WANT TO USE SAME SEQUENCE
OF RANDOM VARIABLES FOR EACH RUN
 long int i, j, l;

 int N=1;
 float tspan; //s
 float kT; //eV
 float Ndop;//1e22 1/m^3
 float Nexc; //1/m^3
 float BW; //eV
 float Eexc; //eV
 float tsample; //s

 FILE *finput;
 char inputfile[40];
 printf("\nEnter input file name:\n");
 scanf("%s", inputfile);
 finput=fopen(inputfile, "r");
 fscanf(finput, "N %d\n", &N);
 printf("N=%d\n", N);
 fscanf(finput, "tspan %f\n", &tspan);
 printf("tspan=%f ps\n", tspan);
 tspan=tspan*1e-12;
 fscanf(finput, "kT %f\n", &kT);
 printf("kT=%f meV\n", kT);
 kT=kT*1e-3;
 fscanf(finput, "Ndop %f\n", &Ndop);
 printf("Ndop=%e cm^-3\n", Ndop);
 Ndop=Ndop*1e6;
 fscanf(finput, "Nexc %f\n", &Nexc);
 printf("Nexc=%e cm^-3\n", Nexc);
 Nexc=Nexc*1e6;
 fscanf(finput, "BW %f\n", &BW);
 printf("BW=%f meV\n", BW);
 BW=BW*1e-3;
 fscanf(finput, "Eexc %f\n", &Eexc);
 printf("Eexc=%f meV\n", Eexc);
 Eexc=Eexc*1e-3;
 fscanf(finput, "tsample %f\n", &tsample);
 printf("tsample=%f ps\n", tsample);
 tsample=tsample*1e-12;
 fclose(finput);
 // printf("N=%d, tspan=%e, kT=%e, Ndop=%e, Nexc=%e, BW=%e, Eexc=%e, tsample=%e\n", N, tspan, kT, Ndop, Nexc, BW,
Eexc, tsample);

 //PART 1:
 FILE *fstatus;
 char outputfile[40];
 printf("\nEnter output file name:\n");
```

```
 scanf("%s", outputfile);
 fstatus=fopen(outputfile, "w");
 clock_t start_time = clock();
 fprintf(fstatus, "%%N %d\n", N);
 fprintf(fstatus, "%%tspan %f ps\n", tspan*1e12);
 fprintf(fstatus, "%%kT %f meV\n", kT*1e3);
 fprintf(fstatus, "%%Ndop %e cm^-3\n", Ndop*1e-6);
 fprintf(fstatus, "%%Nexc %e cm^-3\n", Nexc*1e-6);
 fprintf(fstatus, "%%BW %f meV\n", BW*1e3);
 fprintf(fstatus, "%%Eexc %f meV\n", Eexc*1e3);
 fprintf(fstatus, "%%tsample %f ps\n", tsample*1e12);
 float Emax;
 Emax=10*kT+30e-3+sqrhbar*pow(3*pi*pi*Ndop,2/3)/2/mc/q; //eV
 if (Emax < 5*BW+Eexc)
          Emax=5*BW+Eexc;
 kmax=sqrt(2*mc*Emax*q)/hbar;
 fprintf(fstatus, "%%Emax %f meV, kmax %e [m^-1]\n", Emax*1e3, kmax);

 //PART 2:
 electrongas *particle;
 particle=new electrongas[N];

 //PART 3: SCATTERING TIME CONSTANTS
 float Ne=Ndop+Nexc; //Total electron concentration
 const double sqrbeta=q*Ne/(epsi*kT); //square of inverse debye length : CARE kT is in eV's: CHECKED
 fprintf(fstatus, "%%Debye length =%e [m]\n", 1/sqrt(sqrbeta));
 const double eefactor=mc*q*q*(q*kT)/(4*pi*pow(hbar,3)*epsi);
 const double eerate=Ne*mc*pow(q,4)/(8*pi*pow(epsi,2)*pow(hbar*sqrt(sqrbeta),3)); //TO DEBUG: remove ee scattering, set
eerate=1e10 and N=1000
 const double te=1e-15; //Single electron scattering lifetime
 double tee;
 if (N!=1)
  tee=1/(eerate*N*(N-1)/2); //electron electron scattering lifetime
 else
  tee=1/eerate;

 double dt; //seconds:time resolution
 if (tee < te)
  dt=tee;
 else
  dt=te;

 const long int Te=int(te/dt); //single carrier scattering time
 const long int Tee=int(tee/dt); //carrier-carrier scattering time

 //PART 4: SIMULATION STEPS
 const long Tsample=int(tsample/dt); //Calculate individual spin state and COMPUTE total density matrix at Tsample rate
 if (Tsample == 0)
  assert("Sampling time steps are smaller than program time steps");
 float Tspan;
 Tspan=tspan/dt;
 counter Tend=counter(0, int(Tspan/Tsample), Tsample);
 printf("\nScattering lifetimes:\n single carrier %e\n carrier-carrier %e [s]\n", te, tee);
 printf("Smallest time step %e;\nTotal number of steps %e\n", dt, Tspan);
 fprintf(fstatus, "%%Scattering lifetimes: single carrier %e carrier-carrier %e [s]\n", te, tee);
 fprintf(fstatus, "%%Smallest time step %e; Total number of steps %e\n", dt, Tspan);
 counter tpresent=counter(0, 0, Tsample);

 //PART 5: INITIALIZE SPIN DENSITY MATRIX:
 complex (*SDM)[2][2]=new( complex[Tend.block][2][2] ); //SPIN DENSITY MATRIX
 for (i=0; i<Tend.block; i++)
  for (j=0; j<2; j++)
   for (l=0; l<2; l++)
          SDM[i][j][l]=complex(0,0);

 //PART 6: INITIALIZE ENERGY SPACE OCCUPANCIES

 double dE=0.05*kT;
 const int sizeE=int(Emax/dE);
 fprintf(fstatus, "%%Number of energy grids %d\n", sizeE);
```

```
  double *occE=new(double[sizeE]);
  double Eavg;

  double Ef;
  for (i=0; i<sizeE; i++)
    occE[i]=0;
  complex polarity[2]={complex(1,0), complex(0,0)}; //INITIAL EXCITATION POLARITY
  fprintf(fstatus, "%%Excitation polarity: \n %% %f+i%f\n %% %f+i%f\n", real(polarity[0]), imag(polarity[0]), real(polarity[1]),
imag(polarity[1]));
  distribute(fstatus, Ndop, Ef, Nexc, BW, Eexc, polarity, kT, dE, sizeE, N, particle, Tsample, kmax);

  /*
  FILE *fdist;
  fdist=fopen("dist_initial", "w");
  compute_occE(dE, occE, sizeE, N, particle);
  print_occE(fdist, dE, occE, sizeE, counter(0, 0, Tsample), 0);
  fclose(fdist);
 */

  //IN MATLAB:load energy_dist; d=energy_dist; plot(d(:,1), d(:,2), '-*')

  vector3D avk=vector3D(0, 0, 0);
  for (i=0; i<N; i++)
    avk+=particle[i].momentum;
  avk=avk/N;
  fprintf(fstatus, "%%Initial ensemble averaged momentum components:\n%%kx=%e ky=%e kz=%e\n", avk.x, avk.y, avk.z);

  //PART 7: INITIALIZE MOMENTUM SPAPE OCCUPANCIES
  const int numk=10; //number of momentum cells
  complex occK[2*numk+1][2*numk+1][2*numk+1][2][2]; //class complex set all occupancies to zero
  const double dk=kmax/numk; //lattice sizeE in k space to calculate PAULI blocking factors
  fprintf(fstatus, "%%Maximum momentum in each direction %e \n%%Size of each cell in momentum space %e\n", kmax, dk);
  const double Nmax=(pow(dk/pi,3)/8*N/(Ndop+Nexc));
  fprintf(fstatus, "%%Number of cells %f \n%%Maximum occupancy of each momentum cell %f\n", pow(numk,3),  Nmax);
  /*
  for (i=0; i<N; i++)
    for (j=0; j<2; j++)
      for (l=0; l<2; l++)
            occK[numk+int(particle[i].momentum.x/dk)][numk+int(particle[i].momentum.y/dk)][numk+int(particle[i].momentum.z/dk
)][j][l]+=conj(particle[i].spin[l])*particle[i].spin[j];
  */
  //PART 8: PREPARE INITIAL SINGLE PARTICLE SCATTERING TIMES
  scatlist escatlist;
  escatlist.initialize(N, Te, counter(0, 0, Tsample));

  //PART 9: START ITERATION IN TIME
  int N1, N2;
  fprintf(fstatus, "\n%%Time vs Collective spin polarization & Average Energy [meV]:\n");

  clock_t loopenter_time=clock();
  for (tpresent=0; tpresent < Tend; tpresent++)
    {

      if(N!=1)
            {
             N1=int(rand()*normrnd*(N-1));
             N2=int(rand()*normrnd*(N-1));
             //assert(N-1); //PARTICLE_PARTICLE SCATTERING NOT POSSIBLE WHEN N==1
             while (N1 == N2)
               N2=int(rand()*normrnd*(N-1));
             //printf("two particle scattering:\n");
             //printf("N1=%d, N2=%d\n", N1, N2);
             eescatter(particle[N1], particle[N2], eerate, eefactor, Ef, sqrbeta, kT, tpresent, dt);
            };

      //LOG OR EXP!!!! TAKING TOO LONG to calculate lifetimes: INSTEAD USING ONE ENSEMBLE SCATTERING
PROCESS EXACTLY AT EACH STEP

      while (tpresent>=escatlist.tsmallest())
            {
```

```
            //printf("particle %d, scattering time=%d, tpresent=%d \n", escatlist.firstparticle(),
(escatlist.tsmallest().sub+escatlist.tsmallest().block*escatlist.tsmallest().period), (tpresent.sub+tpresent.block*tpresent.period));
            particle[escatlist.firstparticle()].scatter(Te, Ne, tpresent, dt);
            escatlist.update(Te, tpresent);
            //          printf("Next single particle collision time %e\n",
(escatlist.tsmallest().block*escatlist.tsmallest().period+escatlist.tsmallest().sub)*dt);
          };

    // PART 10: SAMPLE SPIN DENSITY MATRIX AT "Tsample" RATE
    if ( tpresent.sub == 0 )
          {
            if (tpresent.block%200==1)
              {
               printf("\nExpected time to finish %f mins\n", double((clock()-loopenter_time))/CLOCKS_PER_SEC*(Tend.block-
tpresent.block)/60/tpresent.block);
               printf("%d Blocks of %d total blocks completed\n", tpresent.block, Tend.block);
               printf("Current time=%e, Start time=%e\n", double(clock())/CLOCKS_PER_SEC,
double(loopenter_time)/CLOCKS_PER_SEC);
              }
            Eavg=0;
            for (int i=0; i<N; i++)
              {
               particle[i].propagate(tpresent, dt);
               for (int j=0; j<2; j++)
                      for (int l=0; l<2; l++)
                        SDM[tpresent.block][j][l]+=conj(particle[i].spin[l])*particle[i].spin[j];
                      Eavg+=sqrhbar*pow(mag(particle[i].momentum),2)/(2*mc);
              }
            Eavg=Eavg/N/q;
            //printf("Average energy %e \n", Eavg);
            //PART 11: PRINT OUT TIME vs SPIN POLARIZATION, AVERAGE ENERGY
            fprintf(fstatus, "%e   %e   %f\n", tpresent.block*tpresent.period*dt, real(SDM[tpresent.block][0][0]-
SDM[tpresent.block][1][1])/double(N), Eavg*1e3);
            // fprintf(foo, "%e %e %e \n", tpresent.block*dt*tpresent.period, real(conj(particle[0].spin[0])*particle[0].spin[0]),
real(conj(particle[0].spin[1])*particle[0].spin[1]));
          }
  }

 printf("\n");
 clock_t stop_time = clock();
 fprintf(fstatus, "%%Time taken = %d secs\n", (stop_time-start_time)/CLOCKS_PER_SEC);

 avk=vector3D(0, 0, 0);
 for (i=0; i<N; i++)
  avk+=particle[i].momentum;
 avk=avk/N;
 fprintf(fstatus, "%%Final ensemble averaged momentum components:\n%%kx=%e ky=%e kz=%e kmax=%e\n", avk.x, avk.y, avk.z,
kmax);

 //PART 12: PRINT OUT FINAL ENERGY DISTRIBUTION

 /*
 fdist=fopen("dist_final", "w");
 compute_occE(dE, occE, sizeE, N, particle);
 print_occE(fdist, dE, occE, sizeE, Tend, dt);
 fclose(fdist);
 */

 //MATLAB: load energy_dist_initial; di=energy_dist_initial; load energy_dist_final; df=energy_dist_final; plot(di(:,1), di(:,2), 'b-*',
df(:,1), df(:,2), 'g-*');
 printf("Output file: %s\n", outputfile);
 fclose(fstatus);
}


inline void distribute(FILE *fdist, double Ndop, double& Ef, double Nexc, double BW, double Eexc, complex *polarity, double kT,
double dE, int sizeE, const int N, electrongas *particle, long int Tsample, double kmax)
{
 double Nc=2.51e25*pow(mc/mo,3/2)*pow(kT/25e-3,3/2); //effective density of states in m^-3 Corzine 415
 double a1=1/sqrt(8);
```

```
double a2=-4.95009e-3;
double a3=1.48386e-4;
double a4=-4.42563e-6;
Ef=kT*(log(Ndop/Nc)/(1-pow(Ndop/Nc,2)) + pow(3*sqrt(pi)/4*Ndop/Nc, 2/3)/(1+pow(0.24+1.08*pow(3*sqrt(pi)/4*Ndop/Nc, 2/3)
, -2)) );     //Corzine 417
fprintf(fdist, "%%Background Fermi Level [eV] %f\n", Ef);

double Ne=(Ndop+Nexc);
        /*
int Nbg=int(N/Ne*Ndop);
int Npol=N-Nbg;

if (Nbg!=0 && Npol !=0)
  fprintf(fdist, "%%Number of excited carriers %d background carriers %d\n", Npol, Nbg);
else
  {
   if (Nbg==0)
        fprintf(fdist, "%%No background carriers, excited carriers %d\n", Npol);
   else
        fprintf(fdist, "%%No excited carriers, background carriers %d\n", Nbg);
  }
*/
int Nbg=0;
int Npol=N;

int i, j;
double prob, s1r, s1i, s2r, s2i, s;
complex *spin=new(complex[2]);

double *norm1=new(double[sizeE]);
norm1[0]=0;
for (i=1; i< sizeE; i++)
  norm1[i]=norm1[i-1]+sqrt(dE*i)/(exp((dE*i-Ef)/kT)+1); //3D DOS multiplied by fermi function
for (i=1; i< sizeE; i++)
  norm1[i]=norm1[i]/norm1[sizeE-1];

// CHECK PROPER FERMI NORMALIZATION!!!
//      FILE *fermidist;
//      fermidist=fopen("fermidist", "w");
//      fprintf(fermidist, "%%sizeE %d=\n", sizeE);
//      for (i=0; i< sizeE; i++)
//              fprintf(fermidist, "%f\n", norm[i]);

double teta, phi;
vector3D momentum;
for (j=0; j< Nbg; j+=2)
  {
   prob=rand()*normrnd;
   i=0;
   while (prob > norm1[i])
        i++;

   s1r=rand()*normrnd-0.5;
   s1i=rand()*normrnd-0.5;
   s2r=rand()*normrnd-0.5;
   s2i=rand()*normrnd-0.5;
   s=sqrt(s1r*s1r+s1i*s1i+s2r*s2r+s2i*s2i);

   spin[0]=complex(s1r, s1i)/s;
   spin[1]=complex(s2r, s2i)/s;

   teta=acos(1-2*rand()*normrnd);
   phi=2*pi*rand()*normrnd;
   momentum=vector3D(sin(teta)*cos(phi), sin(teta)*sin(phi), cos(teta))*sqrt(2*mc*dE*i*q)/hbar;  //Randomly oriented vector

   particle[j].initialize(spin, momentum, Ef, kT, counter(0, 0, Tsample));

   if (j+1 < Nbg)
                {
                 //GENERATE ORTHOGONAL SPIN COMPONENT:
```

```
                        spin[0]=complex(s2r, -s2i)/s;
                        spin[1]=complex(-s1r, s1i)/s;
                        particle[j+1].initialize(spin, momentum, Ef, kT, counter(0, 0, Tsample));
                        };
  }

 vector3D avk_bg=vector3D(0,0,0);
 for (i=0; i<Nbg; i++)
  avk_bg+=particle[i].momentum;
 avk_bg=avk_bg/double(Nbg);
 for (i=0; i<Nbg; i++)
  particle[i].momentum-=avk_bg;

 //distribute excited carriers

 int grid=Npol+1;
 double dev=3*BW/grid;
 assert( dE*sizeE > dev*grid+Eexc); //Excitation energy much larger than grid range
 if ( Eexc-dev*grid < 0)
          dev=Eexc/(grid+1);
 double *norm2=new(double[grid]);
 norm2[0]=0;

 for (i=1; i<grid; i++)
  norm2[i]=norm2[i-1]+exp(-pow(dev*i/BW,2)); //3D DOS multiplied by fermi function
 for (i=1; i<grid; i++)
  norm2[i]=norm2[i]/norm2[grid-1];

 for (j=Nbg; j<N; j++)
  {
   prob=rand()*normrnd;
   i=0;
   while ((prob > norm2[i]) && (i< grid-1))
                    i++;

   teta=acos(1-2*rand()*normrnd);
   phi=2*pi*rand()*normrnd;
   if (rand()*normrnd < 0.5)
          {
           momentum=vector3D(sin(teta)*cos(phi), sin(teta)*sin(phi), cos(teta))*sqrt(2*mc*(dev*i+Eexc)*q)/hbar;  //Randomly
oriented vector
           particle[j].initialize(polarity, momentum, Ef, kT, counter(0, 0, Tsample));
          }
   else
          {
           momentum=vector3D(sin(teta)*cos(phi), sin(teta)*sin(phi), cos(teta))*sqrt(2*mc*(-dev*i+Eexc)*q)/hbar;
           particle[j].initialize(polarity, momentum, Ef, kT, counter(0, 0, Tsample));
          }
  }

 vector3D avk_exc=vector3D(0, 0, 0);
 for (i=Nbg; i<N; i++)
  avk_exc+=particle[i].momentum;
 avk_exc=avk_exc/double(N);
 for (i=Nbg; i<N; i++)
                    particle[i].momentum-=avk_exc;

 printf("\nEf=%e meV\n", Ef*1e3);
 printf("Carriers distributed!\n");

}

inline void eescatter(electrongas &particle1, electrongas &particle2, double eerate, double eefactor, double Ef, double sqrbeta, double
kT, counter tpresent, double dt)
{
 double costeta, sinteta, phi, rnd, gmag, mag1, mag2;
 vector3D momentum1, momentum2, g, gnew;
 g=particle2.momentum-particle1.momentum;
 gmag=mag(g);
```

```
 //printf("rand*eerate= %e, eefactor*gmag/(g=mag*gmag+sqrbeta)%e\n", rand()*normrnd*eerate ,
eefactor*gmag/(gmag*gmag+sqrbeta));
 if ( rand()*normrnd*eerate < eefactor*gmag/(gmag*gmag+sqrbeta))
  {
   rnd=rand()*normrnd;
   costeta=(1-2*rnd/(1+gmag*gmag*(1-rnd)/sqrbeta));
   sinteta=sqrt(1-costeta*costeta);
   phi=2*pi*rand()*normrnd;
   gnew=gmag*vector3D(sinteta*cos(phi), sinteta*sin(phi), costeta);
   momentum1=particle1.momentum-0.5*(gnew-g);
   momentum2=particle2.momentum+0.5*(gnew-g);
            mag1=mag(momentum1);
            mag2=mag(momentum2);
   if ( (1/(exp( (sqrhbar*mag1*mag1/(2*mc*q)-Ef)/kT ) + 1) < rand()*normrnd) && (1/(exp( (sqrhbar*mag2*mag2/(2*mc*q)-
Ef)/kT ) + 1) < rand()*normrnd))
                        if (( mag1 < kmax ) && ( mag2 < kmax ) ) //PARTICLE MOMENTUM SHOULD BE LESS THAN
MAXIMUM MOMENTUM
                       {
                        //          printf("Old momentum %f new momentum %f\n", particle1.momentum.x, momentum1.x);
                        particle1.scatter(momentum1, tpresent, dt);
                       particle2.scatter(momentum2, tpresent, dt); //PARTICLE SCATTERING INCLUDES SPIN FLIP PROCESS
                       }
  };
}


inline void print_occE(FILE *fp, double dE, double *occE, int sizeE, counter time, double dt)
{
 int i;
 fprintf(fp, "%%OCCUPANCIES AS A FUNCTION OF ENERGY[%d] eV: TIME %f sec\n", sizeE,
time.period*dt*time.block+time.sub);
 for (i=0; i<sizeE; i++)
  fprintf(fp, "%f %f\n", dE*i, occE[i]);
}

inline void compute_occE(double dE, double *occE, int sizeE, int N, electrongas particle[])
{
 int i, j;
 for (i=0; i<sizeE; i++)
  occE[i]=0;
 for (i=0; i<N; i++)
  {
   j=int(sqrhbar*sqr(particle[i].momentum)/(2*mc*dE*q));
   if (j<sizeE)
          occE[j]++;
  };
}
```

CLASSES:

electrongas.hpp

```
#include "globals.hpp"

class electrongas
{
public:
        complex spin[2];
        vector3D momentum;
        counter tpresent;

        electrongas(void)
        {
         // delta=9.0*pow(10, -30); // InP
                delta=20.9*pow(10, -30); //for GaAs
                Vo=pow(10, -23);     //EXCHANGE EFFECT IS TAKEN INTO ACCOUNT AS A SELF CONSISTENT
POTENTIAL:IMPLEMENT LATER
                w_op=q*35e-3/hbar;
                op_factor=q*q*mc/sqrhbar*w_op/4/pi;

                split=0.2; //Valence band spin-orbit split
                Eg=1.54; //Band gab
                //gamma=0;
                gamma=sqrhbar/(sqrt(2)*mc*Eg*q)*split*(split+2*Eg)/ ( (split+Eg)*(2*split+3*Eg) ); //USING THIS
ONE!!!!!!!
                //MINE: gamma=sqrhbar/(6*mc*Eg*q)*split*split/ ( (split+Eg)*(2*split+3*Eg) );
        }

        void initialize(complex *s, const vector3D k, double Efermi, double kTeV, counter time);

 void propagate(counter tfinal, double dt);
 void scatter(long int Te, double density, counter tfinal, double dt);
 void scatter(const vector3D kfinal, counter tfinal, double dt);
 void spinflip(const vector3D momentum, const vector3D kfinal);

private:
        double Ef;
        double kT;

        double delta;
        double Vo;

 double split;
 double Eg;
 double gamma;
 complex eta;

        double dEs, kx, ky, kz, H11, H22, e1, e2, norm, teta1, teta2;
        complex H12, H21, eig1[2], eig2[2], prj1, prj2;
        int i;

        double knew;
        double teta;
        double phi;
        //SCATTERING RATES
        double Z;
        double Nimp;
        double NE;
        double lamb;
        double F;
        double imp;
        double Dac;
        double rho;
        double vs;
        double acph;
```

```
        double Nopph;
        double w_op;
        double kappa_infi;
        double kappa_zero;
        double op_factor;
        double opph_ab;
        double opph_em;
        double f;

        double r;
        double prob;
        double tetarot, phirot;

};

inline void electrongas::initialize(complex *s, const vector3D k, double Efermi, double kTeV, counter time)
{
        tpresent=time;
        Ef=Efermi;
        kT=kTeV;

        for (i=0; i<2; i++)
                spin[i]=s[i];

        momentum=k;
}

inline void electrongas::propagate(counter tfinal, double dt)
{
        double k;
        k=mag(momentum);

        kx=momentum.x;
        ky=momentum.y;
        kz=momentum.z;

        dEs=q*delta*sqrt(k*k*((kx*kx*ky*ky)+(kx*kx*kz*kz)+(ky*ky*kz*kz))-9*(kx*kx*ky*ky*kz*kz)); //in JOULES

        if (k < 1e-16*kmax)
        {
                H11=0;
                H22=0;
                H12=complex(0,0);
                H21=complex(0,0);
        }
        else
        {
                H11=dEs*kz/k;
                H22=-dEs*kz/k;
                H12=dEs*complex(kx/k, -ky/k);
                H21=conj(H12);
        }

        e1=0.5*(H11+H22-sqrt( (H11+H22)*(H11+H22) - 4*(H11*H22- real(H12*H21) ) ) );
        e2=0.5*(H11+H22+sqrt( (H11+H22)*(H11+H22) - 4*(H11*H22- real(H12*H21) ) ) );

        if (fabs(e1-H22) < fabs(e1)*1e-10)
        {
                eig1[0]=0;
                eig1[1]=1;
        }
        else
        {
                eig1[0]=1;
                eig1[1]=H21/(e1-H22);
                norm=sqrt( real(eig1[0]*conj(eig1[0]) + eig1[1]*conj(eig1[1])) );
                eig1[0]=eig1[0]/norm;
                eig1[1]=eig1[1]/norm;
        }
```

```
		if (fabs(e2-H22) < fabs(e2)*1e-10)
		{
			eig2[0]=0;
			eig2[1]=1;
		}
		else
		{
			eig2[0]=1;
			eig2[1]=H21/(e2-H22);
			norm=sqrt( real(eig2[0]*conj(eig2[0]) + eig2[1]*conj(eig2[1])) );
			eig2[0]=eig2[0]/norm;
			eig2[1]=eig2[1]/norm;
		}

		//PROPAGATE FROM tpresent till tfinal

		//	printf("point1:tfinal.period=%d tfinal.sub=%d tfinal.block=%d\n", tfinal.period, tfinal.sub, tfinal.block);
		//	printf("point1:tpresent.period=%d tpresent.sub=%d tpresent.block=%d\n", tpresent.period, tpresent.sub,
tpresent.block);

		// if( tpresent > tfinal)
	//{
		// printf("tfinal.period=%d tfinal.sub=%d tfinal.block=%d\n", tfinal.period, tfinal.sub, tfinal.block);
		// printf("tpresent.period=%d tpresent.sub=%d tpresent.block=%d\n", tpresent.period, tpresent.sub, tpresent.block);
		// }

		teta1=e1*(tfinal-tpresent)*dt/hbar;
		prj1=(conj(eig1[0])*spin[0]+conj(eig1[1])*spin[1])*complex(cos(teta1), -sin(teta1));
		teta2=e2*(tfinal-tpresent)*dt/hbar;
		prj2=(conj(eig2[0])*spin[0]+conj(eig2[1])*spin[1])*complex(cos(teta2), -sin(teta2));

		spin[0]=prj1*eig1[0]+prj2*eig2[0];
		spin[1]=prj1*eig1[1]+prj2*eig2[1];

		//printf("%f %f	%f %f\n", spin[1].re, spin[1].im, prj1.re, prj1.im);

		norm=sqrt(real(spin[0]*conj(spin[0]) + spin[1]*conj(spin[1])) );
		spin[0]=spin[0]/norm;
		spin[1]=spin[1]/norm;

		//		printf("point2: tfinal.period=%d tfinal.sub=%d tfinal.block=%d\n", tfinal.period, tfinal.sub, tfinal.block);
		//		printf("point2: tpresent.period=%d tpresent.sub=%d tpresent.block=%d\n", tpresent.period, tpresent.sub,
tpresent.block);
		tpresent=tfinal;

}

inline void electrongas::scatter(long int Te, double density, counter tfinal, double dt)
{
		//FIRST PROPAGATE
		propagate(tfinal, dt);
		//THEN Select single particle scattering event:

		double k=mag(momentum);
		vector3D kfinal;
		//SCATTERING RATES

		//IONIZED IMPURITY SCATTERING PARAMETERS:
		Z=1;
		Nimp=density;
		NE=mc*k/(pi*pi*sqrhbar);
		lamb=sqrt(density*q/epsi/kT);
		F=1/hbar*pow((Z*q*q/(4*pi*epsi)),2)/32/pow(k,4)*Nimp*NE;
		imp=4*pi*F*pow(2*k/lamb,2)*(1/(1+pow(lamb/2/k, 2))); //IMPURITY SCATTERING RATE

		//OPTICAL PHONON SCATTERING PARAMETERS: PAGE 47 of C. Jacobini,THE MONTE CARLO METHOD FOR
DEVICE SIMULATION
		Nopph=1/(exp(hbar*w_op/(kT*q))-1);
		kappa_infi=epsio*10.60*(1+9e-5*kT*300/25.9e-3); //MADELUNG; SPRINGER VERLAG
		kappa_zero=epsio*12.40*(1+1.2e-4*kT*300/25.9e-3);
```

```
	opph_ab=op_factor/k*(1/kappa_infi-1/kappa_zero)*log(fabs( (k+sqrt(k*k+2*mc*w_op/hbar))/(k-
sqrt(k*k+2*mc*w_op/hbar)) ))*Nopph;
	if ( (k*k-2*mc*w_op/hbar) > 0 )
		opph_em=op_factor/k*(1/kappa_infi-1/kappa_zero)*log(fabs( (k+sqrt(k*k-2*mc*w_op/hbar))/(k-sqrt(k*k-
2*mc*w_op/hbar)) ))*(Nopph+1);
	else
		opph_em=0;
	//RATE IS HIGH BY less than 1 ORDER OF MAGNITUDE!!!!!!!!

	//FOR NO ENERGY LOSS, SET opph_ab=0;opph_em=0;

	//ACCOUSTIC PHONON SCATTERING PARAMETERS
	Dac=7.0*q; //Joules
	rho=5.36e3; //kg/m^3
	vs=4e3;//m/s
	acph=2*pi*Dac*Dac*(kT*q)*NE/(hbar*rho*vs*vs);	//ACCOUSTIC PHONON SCAT RATE

	//	printf("total scattering rate= %e, imp=%e, acph=%e, opph_ab=%e, opph_em=%e\n",
opph_em+opph_ab+acph+imp, imp, acph, opph_ab, opph_em); //CHECK PAGE 48 of JACOBINI
	prob=rand()*normrnd;

	assert(Te*(opph_em+opph_ab+acph+imp)*dt < 1);
	if (prob < acph*Te*dt)
	{
		teta=acos(1-2*rand()*normrnd);
		phi=2*pi*rand()*normrnd;
		knew=k;
	}
	else
		if (prob < (imp+acph)*Te*dt)
		{
			r=rand()*normrnd;
			teta=acos(1- 2*(1-r)/(1+4*k*k/(lamb*lamb)*r));
			phi=2*pi*rand()*normrnd;
			knew=k;
		}
		else
			if (prob < (opph_ab+acph+imp)*Te*dt)
			{	//HAVE TO THINK ABOUT OPTICAL PHONON SCATTERING?????
				r=rand()*normrnd;
				f=2*k*sqrt(k*k+2*mc*w_op/hbar)/pow(k-sqrt(k*k+2*mc*w_op/hbar),2);
				if (fabs(f) <1e-16)
					teta=acos(1-2*r); //IN CASE f is close to zero
				else
					teta=acos( ((1+f)-pow(1+2*f,r))/f ); //PAGE 117 of JACOBINI:
				//printf("teta=%e\n", 180*teta/pi);
				phi=2*pi*rand()*normrnd;
				knew=sqrt(k*k+2*mc*w_op/hbar);
			//	printf("optical phonon absorption\n");
			}
			else
				if(prob < (opph_em+opph_ab+acph+imp)*Te*dt)
				{
					r=rand()*normrnd;
					f=2*k*sqrt(k*k-2*mc*w_op/hbar)/pow(k-sqrt(k*k-2*mc*w_op/hbar),2);
					if (fabs(f) <1e-16)
						teta=acos(1-2*r); //IN CASE f is close to zero
					else
						teta=acos( ((1+f)-pow(1+2*f,r))/f ); //PAGE 117 of JACOBINI
					phi=2*pi*rand()*normrnd;
					knew=sqrt(k*k-2*mc*w_op/hbar);
			//		printf("Optical phonon emission\n");
				}
				else
				{
					teta=0;
					phi=0;
					knew=k;
				}
```

```
		if (1/(exp( (sqrhbar*knew*knew/(2*mc*q)-Ef)/kT ) + 1) < rand()*normrnd)
		{
			tetarot=atan2(sqrt(momentum.x*momentum.x+momentum.y*momentum.y), momentum.z);
			//DON'T USE atan(), USE atan2() instead ********************************************
			phirot=atan2(momentum.y, momentum.x);
			/*
			matrix Y(3,3);
			Y(0,0)=cos(tetarot); Y(0,1)=0; Y(0,2)=sin(tetarot);
			Y(1,0)=0; Y(1,1)=1; Y(1,2)=0;
			Y(2,0)=-sin(tetarot); Y(2,1)=0; Y(2,2)=cos(tetarot);

			matrix Q(3,3);
			Q(0,0)=cos(phirot); Q(0,1)=-sin(phirot); Q(0,2)=0;
			Q(1,0)=sin(phirot); Q(1,1)=cos(phirot); Q(1,2)=0;
			Q(2,0)=0; Q(2,1)=0; Q(2,2)=1;

			matrix W(3,3); //ANALTYTICAL EXPRESSION FOR Q*Y
			W(0,0)=cos(tetarot)*cos(phirot); W(0,1)=-sin(phirot); W(0,2)=sin(tetarot)*cos(phirot);
			W(1,0)=cos(tetarot)*sin(phirot); W(1,1)=cos(phirot); W(1,2)=sin(tetarot)*sin(phirot);
			W(2,0)=-sin(tetarot); W(2,1)=0; W(2,2)=cos(tetarot);
			*/
			//vector3D new_momentum;
			kfinal.x=knew*(sin(teta)*cos(phi)*cos(tetarot)*cos(phirot)-
sin(teta)*sin(phi)*sin(phirot)+cos(teta)*sin(tetarot)*cos(phirot));

		kfinal.y=knew*(sin(teta)*cos(phi)*cos(tetarot)*sin(phirot)+sin(teta)*sin(phi)*cos(phirot)+cos(teta)*sin(tetarot)*sin(phirot)
);
			kfinal.z=knew*(-sin(tetarot)*sin(teta)*cos(phi)+0+cos(tetarot)*cos(teta));
			spinflip(momentum, kfinal);
			momentum=kfinal;
			//vector3D g=momentum-new_momentum;
			//g=momentum-(Q*Y)*vector3D(knew*sin(teta)*cos(phi), knew*sin(teta)*sin(phi), knew*cos(teta));
			//printf("difference in momentum %e %e %e\n", g.x, g.y, g.z);
		}
}
inline void electrongas::scatter(const vector3D kfinal, counter tfinal, double dt)
{
		//FIRST PROPOGATE
		propagate(tfinal, dt);
		//THEN update momentum
		spinflip(momentum, kfinal);
		momentum=kfinal;
}

inline void electrongas::spinflip(const vector3D momentum, const vector3D kfinal)
{
 complex newspin[2];
 eta=gamma*( complex(momentum.x, -momentum.y)*kfinal.z - momentum.z*complex(kfinal.x, -kfinal.y)) ;
 //Solid State Communications, Vol.33, pp.389-391, 1980
 //Phys. stat. sol. (b) 45, 415 (1971)

 newspin[0]=spin[0]+eta*spin[1];
 newspin[1]=spin[1]+conj(eta)*spin[0];
 norm=1/real(conj(newspin[0])*newspin[0]+conj(newspin[1])*newspin[1]);
 spin[0]=newspin[0]*norm;
 spin[1]=newspin[1]*norm;
}
```

scatlist.hpp

```
#ifndef SCATLIST_HPP
#define SCATLIST_HPP

#include "globals.hpp"

struct listelem
{
        long int particle;
        counter stime;
        int next;
};

class scatlist
{
public:
  int first, end;
  scatlist(): first(0) { };

  ~scatlist() {delete list;}
  void initialize(long int N, long int Te, counter tpresent);
  void update(long int Te, counter tpresent); //assigns first particle a new scattering time and orders list
  counter tsmallest() {return list[first].stime;}
  int firstparticle() {return first;}
  void print();

private:
  listelem* list;
  long int N;
};

void scatlist::print()
{
  int next=first;
  //printf("first=%d at %d\n", first, list[first].stime.sub);
  //printf("second=%d at %d\n", list[first].next, list[list[first].next].stime.sub );
  //printf("third=%d at %d\n", list[list[first].next].next, list[list[list[first].next].next].stime.sub);
  //printf("fourth=%d at %d\n", list[list[list[first].next].next].next, list[list[list[list[first].next].next].next].stime.sub);
  //  printf("end=%d\n", end);
  while (next != end )
    {
     printf("particle %d at scattime %d\n", next, list[next].stime.sub+list[next].stime.period*list[next].stime.block );
     next=list[next].next;
    }
  printf("particle %d at scattime %d\n", next, list[next].stime.sub+list[next].stime.period*list[next].stime.block );

};

void scatlist::initialize(long int num, long int Te, counter tpresent)
{
  int i;
  N=num;
  srand((unsigned)time( NULL ) );
  list=new listelem[N];

  for (i=0; i<N; i++)
    list[i].stime = counter(0, 0, tpresent.period);

  for (i=0; i<N-1; i++)
    list[i].next = i+1;

  first=0; end=N-1;
  for (i=0; i<N; i++)
    update(Te, tpresent);
}

void scatlist::update(long int Te, counter tpresent)
{
  int runner, pivot;
```

```
 // printf("----------------------------\n");
  list[first].stime=int(-Te*log(rand()*normrnd))+tpresent;

  if (first != end)
    if (list[first].stime >= list[list[first].next].stime)
    {
      runner=first;
      first=list[runner].next;

      pivot=first;
   //   printf("runner=%d\n", runner);
   //   printf("pivot before=%d\n",pivot);
      while (list[list[pivot].next].stime <= list[runner].stime)
                        {
                                        pivot=list[pivot].next;
//                                      printf("i=%d\n", i);
                                        if (pivot==end)
                                          break;
                        }
    //  printf("pivot after=%d\n",pivot);
      if (pivot==end)
                        end=runner;
      list[runner].next=list[pivot].next;
      list[pivot].next=runner;
    }
}
#endif
```

counter.hpp

```
#ifndef COUNTER_HPP
#define COUNTER_HPP

#include <math.h>

class counter
{
public:
  long int sub;
  long int block;
  long int period;

  friend counter operator+(int x, const counter &v);
  friend counter operator+(const counter &u, int x);
  friend counter operator+(const counter &u, const counter &v);
  friend counter operator-(int x, const counter &v);
  friend counter operator-(const counter &u, int x);
  friend counter operator-(const counter &u, const counter &v);
  friend int operator>(const counter &u, const counter &v);
  friend int operator<(const counter &u, const counter &v);
  friend int operator>=(const counter &u, const counter &v);
  friend int operator<=(const counter &u, const counter &v);
  friend int operator==(const counter &u, const counter &v);

  friend double operator*(const counter &u, double x);
  friend double operator*(double x, const counter &u);

  counter &operator=(int x);
  counter &operator=(const counter &z);
  counter &operator+=(int x);
  counter &operator+=(const counter &z);
  counter &operator++(int);

  counter(void) { };

  counter(int newperiod)
    {
     sub=0;
     block=0;
     period=newperiod;
    }

  counter(int newsub, int newblock, int newperiod)
    {
     sub=newsub%newperiod;
     block=newblock+newsub/newperiod;
     period=newperiod;
    }
};

inline counter operator+(int x, const counter &v)
{
 return counter( (v.sub+x)%v.period, (v.sub+x)/v.period+v.block, v.period);
}

inline counter operator+(const counter &u, int x)
{
 if (x >= 0)
   return counter( (u.sub+x)%u.period, (u.sub+x)/u.period+u.block, u.period );
 else
   return (u-abs(x));
}

inline counter operator+(const counter &u, const counter &v)
{
 assert(u.period == v.period);

 return counter( (u.sub+v.sub)%u.period, (u.sub+v.sub)/u.period+u.block+v.block, u.period);
```

```
}

inline counter operator-(int x, const counter &v)
{
 return (counter(x, 0, v.period)-v);
}

inline counter operator-(const counter &u, int x)
{
            long int j, b;
            j=u.sub-x;
 b=u.block;
 while (j <0)
  {
   b--;
   j=j+u.period;
   assert( !((j<0) && (b==0)) );
  }

 b+=j/u.period; //IN CASE x is negative
 j=j%u.period;

 return counter(j, b, u.period);
}

inline counter operator-(const counter &u, const counter &v)
{
 //printf("u.period=%d v.period=%d\n", u.period, v.period);
            long int b;
 assert(u.period == v.period);

 assert( (u.block-v.block) >= 0 );
 b=u.block-v.block;

 return ( counter(0, b, u.period)+(u.sub-v.sub) );
}

inline int operator>(const counter &u, const counter &v)
{
 assert(u.period == v.period);

 if (u.block == v.block)
  {
   if (u.sub > v.sub)
            return 1;
   else
            return 0;
  }
 else
  {
   if (u.block > v. block)
            return 1;
   else
            return 0;
  };
};

inline int operator<(const counter &u, const counter &v)
{
 return (v > u);
};

inline int operator>=(const counter &u, const counter &v)
{
 assert(u.period == v.period);
 if ( (u.block == v.block) && (u.sub == v.sub) )
 return 1;
 else
 return (u>v);
};
```

```
inline int operator<=(const counter &u, const counter &v)
{
  return (v >= u);
};

inline int operator==(const counter &u, const counter &v)
{
  return ((u.sub==v.sub) && (u.block==v.block));
};

inline double operator*(const counter &u, double x)
{
  return( (double(u.block)*double(u.period)+double(u.sub)) * x);
};
inline double operator*(double x, const counter &u)
{
  return( (double(u.block)*double(u.period)+double(u.sub)) * x);
};

inline counter &counter::operator=(int x)
{
  sub=(x%period);
  block= x/period;
  return *this;
}

inline counter &counter::operator=(const counter &z)
{
  period=z.period;
  sub=z.sub;
  block=z.block;
  return *this;
}

inline counter &counter::operator+=(int x)
{
  sub+=x;
  block+=sub/period;
  sub=sub%period;
  return *this;
}

inline counter &counter::operator+=(const counter &z)
{
  assert(period == z.period);
  sub+=z.sub;
  block+=z.block+(sub/period);
  sub=(sub%period);
  return *this;
}

inline counter &counter::operator++(int)
{
  sub++;
  block+=sub/period;
  sub=sub%period;
  return *this;
}

#endif
```

matrix.hpp

```
#ifndef COUNTER_HPP
#define COUNTER_HPP

#include <math.h>

class counter
{
public:
  long int sub;
  long int block;
  long int period;

  friend counter operator+(int x, const counter &v);
  friend counter operator+(const counter &u, int x);
  friend counter operator+(const counter &u, const counter &v);
  friend counter operator-(int x, const counter &v);
  friend counter operator-(const counter &u, int x);
  friend counter operator-(const counter &u, const counter &v);
  friend int operator>(const counter &u, const counter &v);
  friend int operator<(const counter &u, const counter &v);
  friend int operator>=(const counter &u, const counter &v);
  friend int operator<=(const counter &u, const counter &v);
  friend int operator==(const counter &u, const counter &v);

  friend double operator*(const counter &u, double x);
  friend double operator*(double x, const counter &u);

  counter &operator=(int x);
  counter &operator=(const counter &z);
  counter &operator+=(int x);
  counter &operator+=(const counter &z);
  counter &operator++(int);

  counter(void) { };

  counter(int newperiod)
    {
     sub=0;
     block=0;
     period=newperiod;
    }

  counter(int newsub, int newblock, int newperiod)
    {
     sub=newsub%newperiod;
     block=newblock+newsub/newperiod;
     period=newperiod;
    }
};

inline counter operator+(int x, const counter &v)
{
 return counter( (v.sub+x)%v.period, (v.sub+x)/v.period+v.block, v.period);
}

inline counter operator+(const counter &u, int x)
{
 if (x >= 0)
   return counter( (u.sub+x)%u.period, (u.sub+x)/u.period+u.block, u.period );
 else
   return (u-abs(x));
}

inline counter operator+(const counter &u, const counter &v)
{
 assert(u.period == v.period);
```

```
 return counter( (u.sub+v.sub)%u.period, (u.sub+v.sub)/u.period+u.block+v.block, u.period);
}

inline counter operator-(int x, const counter &v)
{
 return (counter(x, 0, v.period)-v);
}

inline counter operator-(const counter &u, int x)
{
            long int j, b;
            j=u.sub-x;
 b=u.block;
 while (j <0)
  {
   b--;
   j=j+u.period;
   assert( !((j<0) && (b==0)) );
  }

 b+=j/u.period; //IN CASE x is negative
 j=j%u.period;

 return counter(j, b, u.period);
}

inline counter operator-(const counter &u, const counter &v)
{
 //printf("u.period=%d v.period=%d\n", u.period, v.period);
            long int b;
 assert(u.period == v.period);

 assert( (u.block-v.block) >= 0 );
 b=u.block-v.block;

 return ( counter(0, b, u.period)+(u.sub-v.sub) );
}

inline int operator>(const counter &u, const counter &v)
{
 assert(u.period == v.period);

 if (u.block == v.block)
  {
   if (u.sub > v.sub)
            return 1;
   else
            return 0;
  }
 else
  {
   if (u.block > v. block)
            return 1;
   else
            return 0;
  };
};

inline int operator<(const counter &u, const counter &v)
{
 return (v > u);
};

inline int operator>=(const counter &u, const counter &v)
{
 assert(u.period == v.period);
 if ( (u.block == v.block) && (u.sub == v.sub) )
 return 1;
 else
 return (u>v);
```

```
};

inline int operator<=(const counter &u, const counter &v)
{
 return (v >= u);
};

inline int operator==(const counter &u, const counter &v)
{
 return ((u.sub==v.sub) && (u.block==v.block));
};

inline double operator*(const counter &u, double x)
{
 return( (double(u.block)*double(u.period)+double(u.sub)) * x);
};
inline double operator*(double x, const counter &u)
{
 return( (double(u.block)*double(u.period)+double(u.sub)) * x);
};

inline counter &counter::operator=(int x)
{
 sub=(x%period);
 block= x/period;
 return *this;
}

inline counter &counter::operator=(const counter &z)
{
 period=z.period;
 sub=z.sub;
 block=z.block;
 return *this;
}

inline counter &counter::operator+=(int x)
{
 sub+=x;
 block+=sub/period;
 sub=sub%period;
 return *this;
}

inline counter &counter::operator+=(const counter &z)
{
 assert(period == z.period);
 sub+=z.sub;
 block+=z.block+(sub/period);
 sub=(sub%period);
 return *this;
}

inline counter &counter::operator++(int)
{
 sub++;
 block+=sub/period;
 sub=sub%period;
 return *this;
}

#endif
```

complex.hpp

```
#ifndef COMPLEX_HPP
#define COMPLEX_HPP

#include <iostream.h>
#include <math.h>

class complex
{
        friend complex operator+(double x, const complex &v);
        friend complex operator+(const complex &u, double x);
        friend complex operator+(const complex &u, const complex &v);
        friend complex operator-(double x, const complex &v);
        friend complex operator-(const complex &u, double x);
        friend complex operator-(const complex &u, const complex &v);

        friend complex operator*(double x, const complex &v);
        friend complex operator*(const complex &u, double x);
        friend complex operator*(const complex &u, const complex &v);
        friend complex operator/(const complex &u, double x);

        friend double real(const complex &z);
        friend double imag(const complex &z);
        friend double mod(const complex &z);
        friend complex conj(const complex &z);
        friend complex exp(const complex &z);

public:
        double &real(void);
        double &imag(void);

        complex(void){re=0.0; im=0.0;}
        complex(double r, double i);
        complex(const complex &z);

        complex &operator=(double x);
        complex &operator=(const complex &z);

        complex &operator+=(double x);
        complex &operator+=(const complex &z);

        complex operator-() const;

        double re, im;
};

//friend functions:
inline complex operator+(double x, const complex &v)
{
        return complex( x + v.re, v.im);
}

inline complex operator+(const complex &u, double x)
{
        return complex(u.re + x, u.im);
}

inline complex operator+(const complex &u, const complex &v)
{
        return complex(u.re + v.re, u.im + v.im);
}

inline complex operator-(double x, const complex &v)
{
        return complex( x - v.re, -v.im);
}

inline complex operator-(const complex &u, double x)
{
```

```
        return complex(u.re - x, u.im);
}

inline complex operator-(const complex &u, const complex &v)
{
        return complex(u.re - v.re, u.im - v.im);
}

inline complex operator*(double x, const complex &v)
{
        return complex( x * v.re, x * v.im);
}

inline complex operator*(const complex &u, double x)
{
        return complex(u.re * x, u.im * x);
}

inline complex operator*(const complex &u, const complex &v)
{
        return complex(u.re * v.re - u.im * v.im, u.im * v.re + u.re * v.im);
}

inline complex operator/(const complex &u, double x)
{
        return complex(u.re/x, u.im/x);
}

inline double real(const complex &z)
{
        return z.re;
}

inline double imag(const complex &z)
{
        return z.im;
}

inline double mod(const complex &z)
{
        return(sqrt(z.re * z.re + z.im * z.im));
}

inline complex conj(const complex &z)
{
        return complex(z.re, -z.im);
}

inline complex exp(const complex &z)
{
        double temp;
        temp=exp(z.re);
        return complex(temp * cos(z.im), temp * sin(z.im));
}

//member functions and operators
inline double &complex::real(void)
{
        return re;
}

inline double &complex::imag(void)
{
        return im;
}

inline complex::complex(double r, double i)
{
        re=r;
        im=i;
```

```
}

inline complex::complex(const complex &z)
{
        re=z.re;
        im=z.im;
}

inline complex &complex::operator=(double x)
{
        re=x;
        im=0.0;
        return *this;
}

inline complex &complex::operator=(const complex &z)
{
        re=z.re;
        im=z.im;
        return *this;
}

inline complex &complex::operator+=(double x)
{
        re +=x;
        return *this;
}

inline complex &complex::operator+=(const complex &z)
{
        re += z.re;
        im += z.im;
  return *this;
}

inline complex complex::operator-() const
{
        return complex(-re, -im);
}

#endif //COMPLEX.HPP
```

vector3D.hpp

```
#ifndef VECTOR3D_HPP
#define VECTOR3D_HPP

#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

class vector3D
{
        friend vector3D operator+(const vector3D &u, const vector3D &v);
        friend vector3D operator-(const vector3D &u, const vector3D &v);
        friend vector3D operator*(double r, const vector3D &v);
        friend vector3D operator*(const vector3D &u, double r);
        friend double operator*(const vector3D &u, const vector3D &v);
        friend vector3D operator/(const vector3D &u, double r);
        friend ostream& operator<<(ostream& out, vector3D v);

        friend double mag(const vector3D &z);
        friend double sqr(const vector3D &z);
public:
        vector3D(void){x=0.0; y=0.0; z=0.0;}
        vector3D(double x, double y, double z);
        vector3D(const vector3D &z);

        vector3D &operator=(const vector3D &z);

        vector3D &operator+=(const vector3D &z);

        vector3D &operator-=(const vector3D &z);

        vector3D operator-() const;

        double x, y, z;
};

//friend functions:

inline vector3D operator+(const vector3D &u, const vector3D &v)
{
        return vector3D(u.x+v.x, u.y + v.y, u.z+v.z);
}

inline vector3D operator-(const vector3D &u, const vector3D &v)
{
        return vector3D(u.x-v.x, u.y - v.y, u.z-v.z);
}

inline vector3D operator*(double r, const vector3D &v)
{
        return vector3D( r * v.x, r * v.y, r * v.z);
}

inline vector3D operator*(const vector3D &v, double r)
{
        return vector3D( r * v.x, r * v.y, r * v.z);
}

inline double operator*(const vector3D &u, const vector3D &v)
{
        return (u.x * v.x+u.y * v.y+u.z * v.z);
}

inline vector3D operator/(const vector3D &v, double r)
{
        return vector3D( v.x/r, v.y/r, v.z/r);
}
```

```
inline double mag(const vector3D &u)
{
        return(sqrt(u.x*u.x + u.y*u.y + u.z*u.z));
}

inline double sqr(const vector3D &u)
{
        return(u.x*u.x + u.y*u.y + u.z*u.z);
}

//member functions and operators

inline vector3D::vector3D(double e1, double e2, double e3)
{
        x=e1;
        y=e2;
        z=e3;
}

inline vector3D::vector3D(const vector3D &u)
{
        x=u.x;
        y=u.y;
        z=u.z;
}

inline vector3D &vector3D::operator=(const vector3D &u)
{
        x=u.x;
        y=u.y;
        z=u.z;
        return *this;
}

inline vector3D &vector3D::operator+=(const vector3D &u)
{
        x += u.x;
        y += u.y;
        z += u.z;
        return *this;
}

inline vector3D &vector3D::operator-=(const vector3D &u)
{
        x -= u.x;
        y -= u.y;
        z -= u.z;
        return *this;
}

inline vector3D vector3D::operator-() const
{
        return vector3D(-x, -y, -z);
}

inline ostream& operator<<(ostream& out, vector3D v)
{
        out << "(" << v.x << "," << v.y << "," << v.z << ")";
        return out;
}
#endif
```