Theoretical Study and Monte Carlo Simulation of III-V Compound Low-Noise Photodiodes

Dissertation

Presented to the Faculty of the Graduate School
at University of Virginia

In Partial Fulfillment
Of the Requirements for the Degree
of Doctor of Philosophy

By
Wenlu Sun

Advisor: Prof. Joe C. Campbell

December 2014

APPROVAL SHEET

The dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

AUTHOR: 

The dissertation has been read and approved by the examining committee:

Joe C. Campbell (advisor)

Robert M. Weikle (Chair)

Stephen G. Wilson (ECE)

Andreas Beling (ECE)

Olivier Pfister (PHYS)

Accepted for the School of Engineering and Applied Science:

Dean, School of Engineering and Applied Science

December 2014

Acknowledgements

I would like to thank my advisor, Dr. Joe C. Campbell, for his great guidance and support throughout these years. I have learned a great deal from him about not only how to perform good scientific research, but also how to become a person with thoughts and ideas. Without his insights many ideas presented in this dissertation would not have been realized.

Next, I would like to thank my colleague, Dr. Xiaoguang Zheng, who has provided lots of great ideas and help in all the simulation and experiment. With his wealthy knowledge in photonic area, I have truly learned a lot.

I will also thank Dr. Seth R. Bank, and Scott Maddox at University of Texas at Austin, with their hard work and trust, we truly achieved great collaboration and made significant progress. I also would like to thank Dr. Franklin, Quinlan at NIST for his great help during my simulation work. His advanced mathematical skills really impressed me and encouraged me to obtain better ideas and results.

And I would like to thank all my colleagues and university staff members for their valuable discussion whenever I have questions and problems. Thank you all!

Theoretical Study and Monte Carlo Simulation of III-V Compound Low-Noise Photodiodes

Wenlu Sun

Advisor: Joe C. Campbell

Department of Electrical and Computer Engineering,

University of Virginia

A Monte Carlo simulation tool is developed to theoretically study the characterisics of avalanche photodiodes (APDs) and high-power photodiodes and to design new structures with enhanced performance. Consistency is achieved between experiments and simulations.

With respect to APDs, I have studied the noise of an InAlAs/InAlGaAs tandem APD structure. My simulations reproduced previous experimental results. I then used the model to modify the structure in order to achieve lower excess noise. The new device has been fabricated and confirms my prediction of lower noise. Recently it has been reported that InAs APDs can achieve extremely low noise. In this dissertation, two InAs APD structures are designed and fabricated. They exhibit low dark current and low excess noise factors. An AlAsSb blocking layer is used in order

to further improve performance. Lower dark current and significantly higher gain have been demonstrated. Monte Carlo simulation is also used to investigate the bandwidth enhancement effect in APDs at high photocurrent levels. The non-linear behavior of these photodiodes is studied by considering the APD as a mixer and characterizing the conversion efficiencies.

Photonic microwave signal generation has been used to generate microwave signals with extremely low phase noise. It has been observed in experiments that the total shot noise contributes unequally to phase noise and amplitude noise. An analytical model developed at NIST explains the imbalance effect but cannot replicate the observed saturation behavior of the phase noise level. Through Monte Carlo simulation I have shown that the randomness in carrier transport contributes to excess shot noise, which significantly affects the phase noise level and explains the saturation phenomenon.

Table of Contents

List of Figures

1 Introduction

1.1 Avalanche photodiodes and their applications

Avalanche photodiodes (APDs) have been widely used to detect and amplify weak optical signals in high-bit-rate, long-haul optical fiber communications systems. The internal gain provided by the impact ionization process can significantly improve the receiver sensitivities, see Fig. 1.1, sensitivity is defined as the noise equivalent optical power level. Compared to photomultiplier tubes (PMTs) APDs are smaller, require lower bias voltage, and cost less. They have also been used in short-wave and mid-wave infrared detection systems, such as imaging and LIDAR detection. A key consideration for these applications is the APD noise level.

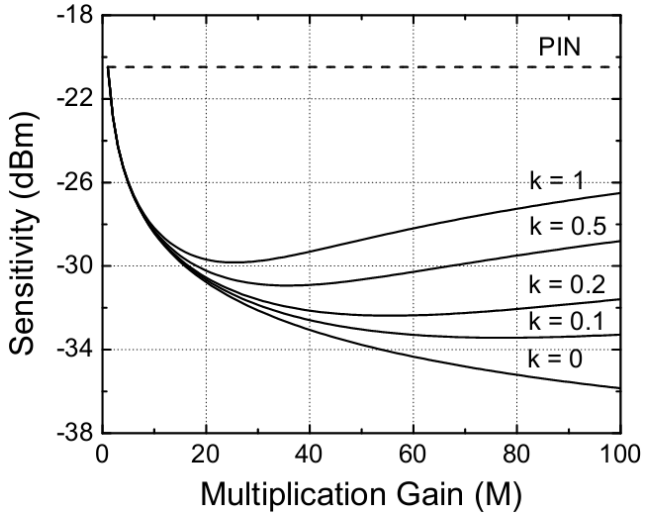


Figure 1.1. Typical Comparison of sensitivity between P-I-N and APD photodetector, the k value is ratio of ionization coefficients in APDs, see details below.

Avalanche multiplication is always accompanied by excess shot noise that is caused by the statistical nature of the impact ionization process, which is closely

related to the band structure. This results in significant variation in the multiplication gain of avalanche photodiode for each carrier injected into the multiplication region. In semiconductors, both electrons and holes are capable of initiating impact ionization events. Key parameters are the electron and hole ionization coefficient, α and β, respectively, which are defined as the inverse of the mean distance between successive ionization events.

The impact ionization events (both electron and hole) occur with certain randomness that contributes to excess shot noise in the generated photocurrent. McIntyre theoretically studied the noise power spectral density, S, of an APD in linear mode operation, i.e. below breakdown, and derived the following equation [1]

$$S = 2qI\langle M^2\rangle = 2qI\langle M\rangle^2 F(M) \tag{1.1}$$

$$F(M) = \langle M^2\rangle / \langle M\rangle^2 \tag{1.2}$$

where q is the fundamental charge, I is the average APD current, $\langle M\rangle$ is the average multiplication gain, and F(M) is defined as the excess noise factor. If there is no multiplication gain, the noise is given by the conventional shot noise expression and there is no excess noise. The excess noise factor F(M), caused by randomness of carrier transport on the microscopic level, is determined by several carrier scattering mechanisms, such as phonon scattering, impurity scattering, and impact ionization scattering. McIntyre derived the expression for the excess noise factor in the case of an uniform electric field:

$$F(M) = kM + (2-1/M)(1-k) \tag{1.3}$$

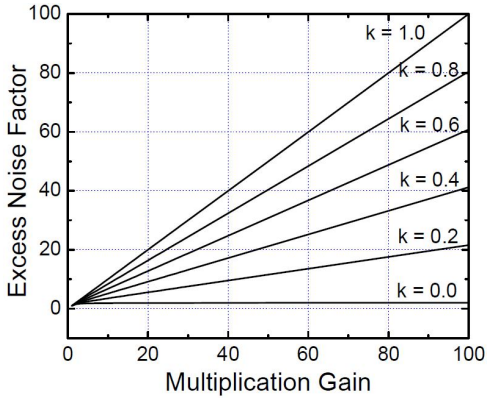


Figure 1.2. Excess noise factor as a function of multiplication gain at different k values.

where k is defined as the ratio of the ionization coefficients. Figure 1.2 shows the excess noise versus gain. It is clear that the k value should be minimized in order to achieve low noise. Another APD performance characteristic that is closely related to excess noise is bandwidth. The APD bandwidth is limited by both carrier transit-time and the RC-time constant. For APDs the carrier transit-time may be affected by the same factors that influence the excess noise properties.

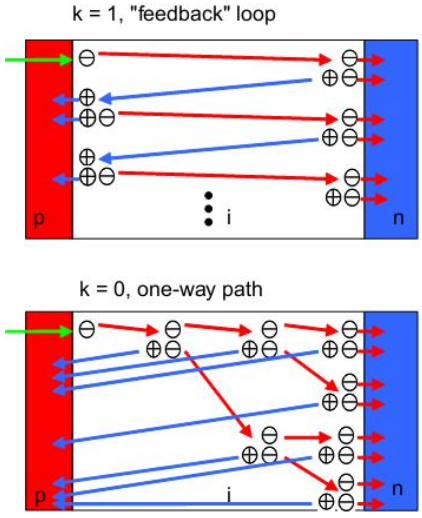


Figure 1.3. Illustration of APD bandwidth degradation due to impact ionization process.

As shown in Fig. 1.3, if k is larger than 0, both electrons and holes can initiate impact ionization. Therefore the multiplication process in an APD will form a feedback loop of impact ionization, which will result in an increase in the time required for the gain to build up. The loop will disappear if k equals to 0. Figure 1.4 shows the bandwidth as a function of multiplication gain for different k values as calculated using the local-field model [2]. The bandwidth is relatively independent of multiplication gain when k is close to zero. Therefore, in order to achieve APDs with low noise and high speed, it is beneficial to have k value as low as possible. The next section introduces several types of materials with low k factors, and structural designs that have been used to achieve low excess noise.

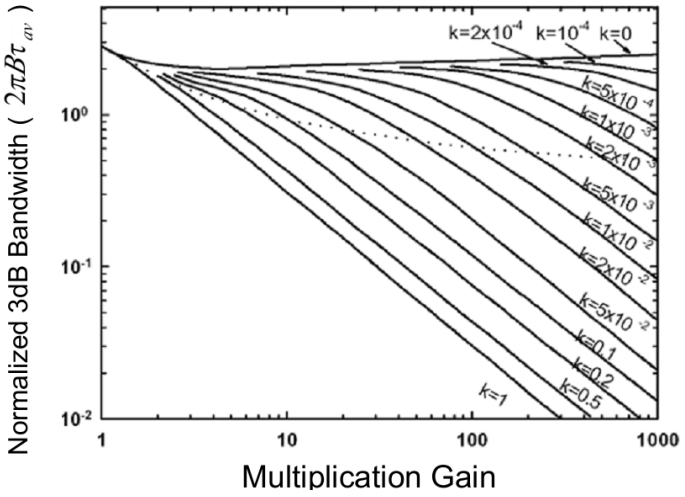


Figure 1.4. Bandwidth as a function of multiplication gain at different k values.

1.2 Low noise avalanche photodiode

The separate-absorption-charge-multiplication (SACM) APD structure (Fig. 1.5) has been widely deployed for fiber optic communication systems. A small band-gap material, which is typically $In_{0.53}Ga_{0.47}As$, is used for the absorption layer. The multiplication layer, where essentially all impact ionization events occur, consists of a large band-gap material that does not exhibit a significant tunneling component of the dark current at high electric field. The most widely used SACM APD structure for telecommunications uses InP for the multiplication region. While the InP/InGaAsP/InGaAs SACM APDs have achieved excellent receiver sensitivities up to 2.5 Gb/s [3]-[5] the relatively low gain-bandwidth product (<100 GHz) restricts the frequency response. The gain-bandwidth product and high excess noise are consequences of the reasonably unfavorable ionization coefficients of InP [6]-[8]. Recently owing to its lower k value, $In_{0.52}Al_{0.48}As$ has become the material of choice for the multiplication region in the highest performance APDs. The electric field in the multiplication layer should be high enough to generate the required gain. In the absorption layer, on the other hand, the field should be low enough to suppress tunneling and high enough to maintain carrier saturation velocity in order to minimize the transit time and prevent degradation of the bandwidth. Sandwiched between the absorption and multiplication regions is the charge layer, which through its doping density can achieve the desired electric field distribution. To achieve low k values, it is necessary that the electron ionization coefficient and hole ionization

coefficient in the multiplication region be as different as possible.

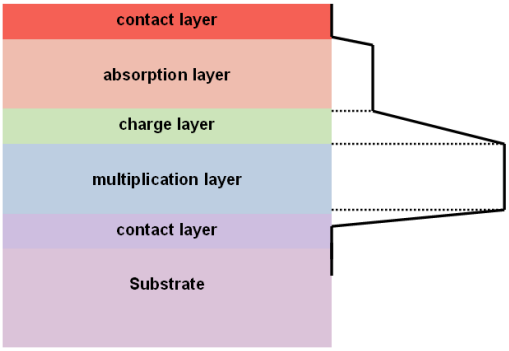


Figure 1.5. Separate-absorption-charge-multiplication (SACM) structure and the field distribution.

The k values of some typical materials are included in table 1.1. Most material types have k value larger than 0.2. Silicon has a very low k value (~ 0.02), however, it has very low responsivity at telecommunication wavelengths (1300 nm to 1600 nm). InAs has a k value of 0, making it very attractive for low noise APDs, but the small band-gap results in high dark current at room temperature, which degrades the overall performance of InAs for many applications. This will be discussed in detail in Chapter 2. An alternative to using material with a low k value, is to design a device structure that suppresses the excess noise.

Material	k value	Band-gap (eV)	Dark current	Substrate
GaAs	0.6	1.4	10 pA ~ 1 nA	GaAs
InP	0.4	1.41	10 ~ 100 pA	InP
InGaAs	0.2	0.79	1 ~ 10 μA	InP
InAlAs	0.2	1.5	1 ~ 10 nA	InP
Si	0.02	1.2	1 ~ 10 pA	Si
InAs	0	0.4	10 ~ 100 μA	InAs

Table 1.1. Typical k values for semiconductor materials.

For most materials both the electron and hole ionization coefficients increase with electric field strength and eventually become comparable [9]. Therefore the k value approaches unity as the field increases. In APDs with a thin multiplication layer, the electric field has to be higher than that for a thick layer in order to achieve the same gain, thus, one would expect the excess noise characteristics of APDs with thin multiplication layers to mimic materials with high k values. However, the opposite case is observed. It has been shown for a wide range of materials including InP [10]-[11], GaAs [12]-[15], InAlAs [16], AlGaAs [17]-[18], and SiC [19] that lower k value is achieved with thinner multiplication regions. This is caused by the non-local effect [14]. Carriers that initiate an impact ionization event have to travel a certain distance before they can gain sufficient energy from the electric field to initiate an impact ionization. This distance is referred to as the dead space. When the dead space is an appreciable fraction of the multiplication thickness, it is less likely for a carrier to initiate gain greatly in excess of the average gain. The result is lower statistical variance in the impact ionization process and thus lower excess noise. Thin layers of AlInAs have been incorporated into the multiplication region of SACM APDs. Ning Li et al. [8] reported that mesa-structure undepleted-absorber InAlAs APDs with 180 nm-thick multiplication regions exhibited excess noise equivalent to $k = 0.15$ and gain-bandwidth product of 160 GHz.

Another approach that has been very effective in reducing the excess noise is impact-ionization engineering (I^2E) [20]-[22]. I^2E utilizes hetero-junctions to provide greater localization of impact ionization than can be achieved in spatially uniform

structures, which renders impact ionization events more deterministic and results in lower excess noise factor. A typical InAlAs/InAlGaAs I^2E APD structure and the corresponding electric field distribution are shown in Fig. 1.6. The noise measurement in Fig. 1.7 indicates that the InAlAs/InAlGaAs I^2E structure has excess noise that corresponds to k ~ 0.1 if F(M) is plotted versus gain using McIntyre's local-field model. It should be noted that the local-field model is not valid when non-local effects dominate (see chapter 5). Nevertheless, the local-field graphs of F(M) for various k values provide a convenient reference. Another structure used to achieve low noise APD is tandem APD structure. This approach is described in chapter 2.

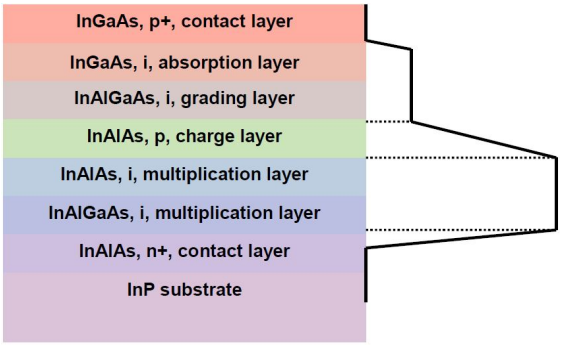


Figure 1.6. Schematic structure of typical InAlAs/InAlGaAs I^2E APD with electric field distribution.

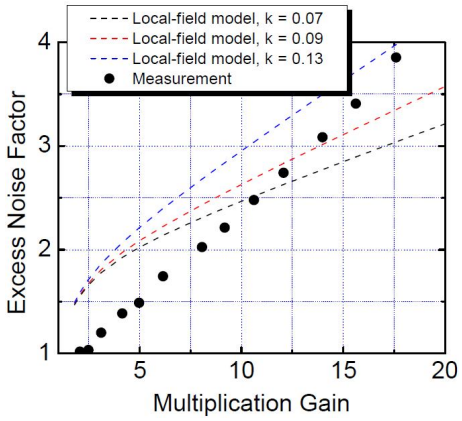


Figure 1.7. The measured excess noise factor of an nLight InAlAs/InAlGaAs I^2E APD.

1.3 Noise Measurement Methods

In our laboratory excess noise is typically measured using an HP8970 noise figure meter. A bias tee is used to provide DC bias and couple the noise signal through a low-noise amplifier into the noise figure meter, as shown in Fig. 1.8. Optical signal from a laser is illuminated onto the APD under test. It is important to ensure that the light spot is focused on the central area of mesa, otherwise part of light will be absorbed by the mesa edge, which results in mixed-injection and higher excess noise. Local field theory indicates that the noise power spectral density can be expressed as:

$$S = 2eIM^2F(M)R \tag{1.4}$$

where R is the total impedance of the APD and the measurement system. The first step is bias the APD slightly above the punch-through voltage. The punch-through voltage is the voltage as which the edge of the depletion reaches the absorbing layer. This measurement is used to determine the unity gain, i.e., M = 1 at which point F(M) = 1, and S = 2eIR. Using an attenuator to change the optical power, we measure the noise power versus photocurrent curve. Linear fitting of this curve provides a good estimation of 2eR, which I will refer to as α. The second step is to fix the optical power at a low level and measure the noise power as the bias is increased. At higher bias:

$$S = 2eIM^2F(M)R = \alpha M^2F(M). \tag{1.5}$$

This permits determination of F(M) at different voltages and gain values, M.

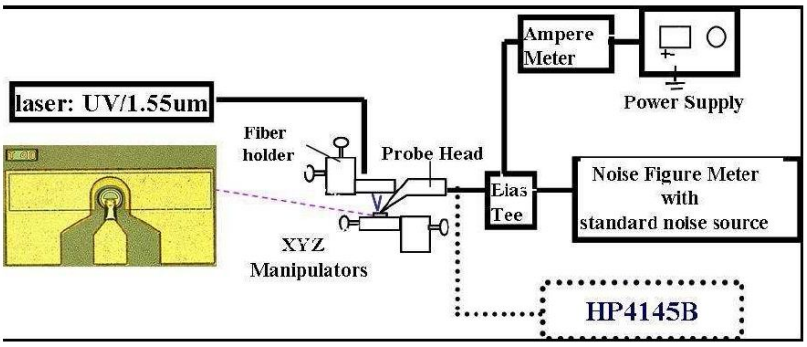


Figure 1.8. Block diagram of the experimental setup for noise measurement.

This noise measurement is unambiguous if the gain at the punch-through voltage is unity, however, this is not always the case. If the gain at punch-through is greater than unity, it is necessary to determine that gain value by another method. Assume the punch-through gain is M_0. It follows that the noise power at punch-through voltage is:

$$S_0 = 2eIM_0^2F(M_0)\,. \tag{1.6}$$

The noise power at higher voltage is:

$$S = 2eIM^2F(M) \tag{1.7}$$

and the noise power ratio is

$$\frac{S}{S_0} = \frac{M^2}{M_0^2}\frac{F(M)}{F(M_0)}\,. \tag{1.8}$$

The excess noise factor F(M) typically follows F(M) = kM + (1-k). This is particularly true for small gain and will be discussed in Chapter 4. Therefore:

$$F(M_0) = kM_0 + (1-k)$$
$$F(M) = kM + (1-k) \tag{1.9}$$

$$\frac{F(M)}{F(M_0)} = \frac{kM + (1-k)}{kM_0 + (1-k)} = \frac{k\frac{M}{M_0} + \frac{(1-k)}{M_0}}{k + \frac{(1-k)}{M_0}} = \frac{kM_{\exp} + \frac{(1-k)}{M_0}}{k + \frac{(1-k)}{M_0}} \tag{1.10}$$

where M_{exp} is the experimentally measured gain, and M is the real gain at a certain bias. Finally, we have:

$$(\frac{S}{S_0})_{\exp} = \frac{M^2}{M_0^2}\frac{F(M)}{F(M_0)} = M_{\exp}^2 \frac{kM_{\exp} + \frac{(1-k)}{M_0}}{k + \frac{(1-k)}{M_0}} \quad . \tag{1.11}$$

By measuring S, S_0, and M_{exp}, we can plug the raw data into the above equation and use k and M_0 as two fitting parameters to extract the punch-through gain M_0.

Another frequently used measurement method employs the phase-sensitive detection system shown in Fig. 1.9. The light signal from the laser is chopped at 200 Hz. This imposes 200 Hz modulation on the photo-current and photo-noise (fluctuation within photo-current). Two lock-in amplifiers are used to measure the photo-current and the photo-noise signals. This noise measurement setup can be used to measure APD devices with high dark current, since the dark current is not chopped it is rejected by the lock-in amplifier. All the noise measurements performed on InAs APDs in chapter 3 used this phase-sensitive detection setup.

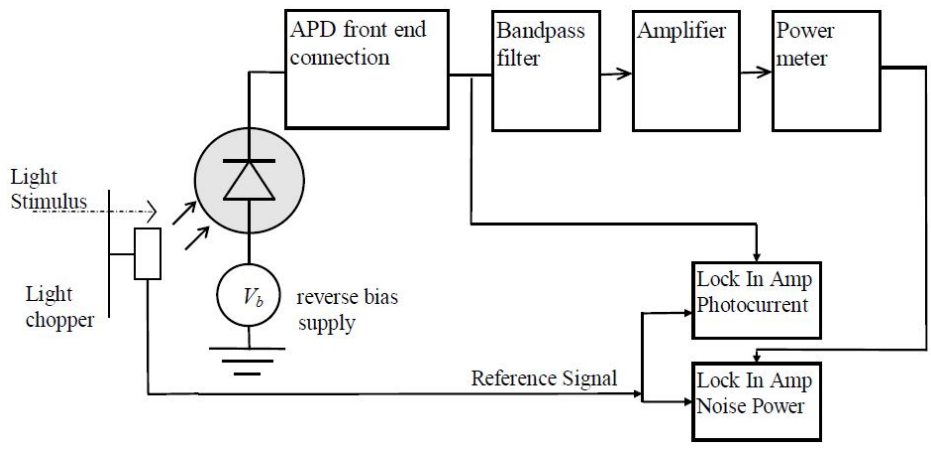


Figure 1.9. Noise measurement system with phase-sensitive detection system.

1.4 Monte Carlo Simulation of APDs

Monte Carlo simulation of multiplication in APDs has been reported for a wide range of materials and structures. Schichijo and Hess [27] performed an early Monte Carlo simulation of high-field electron transport and impact ionization in GaAs using a full-band model that was generated with pseudo-potentials. The full-band approach has the advantage of accurate bands throughout the Brillouin zone at the cost of long computation time. It has been shown that Monte Carlo simulation with a simplified parabolic band structure can achieve very accurate results with greatly reduced computational resources [23]-[26].

The Monte Carlo model used in this work employs spherical non-parabolic bands for both the conduction band and the valence band. The conduction band structure includes Γ, X, and L valleys and heavy hole, light hole, and split-off valance bands, as illustrated in Fig. 1.10. For carrier transport, phonon scattering, alloy scattering, and

impurity scattering of both electrons and holes are incorporated in the model. Phonon scattering includes both inter-valley and intra-valley scattering. Formulas for scattering rates were derived using Fermi's golden rule. The impact ionization scattering rate was calculated by the Keldysh formula [28]. When carriers initiate impact ionization events, newly generated electron and hole are in the Γ valley and the heavy hole band, respectively. The original carrier, together with these newly generated carriers will continue carrier transport until reaching the metal contact.

The simulation program is written in C++, and all the simulations are performed by parallel computing on UVA Linux Clusters. The first step of simulation includes establishing a grid and reading parameters. As shown in Fig. 1.11, all the necessary parameters are read from input files. The APD structure is divided into a one-dimensional grid. The mesh size for the spatial resolution is dictated by the charge variations. Hence, we have to choose the mesh size to be smaller than the smallest wavelength of the charge variations. The smallest wavelength is approximately equal to the Debye length, λ_D, given as:

$$\lambda_D = \sqrt{\frac{\varepsilon_s k_B T}{e^2 n}} \tag{1.12}$$

Where k_B is the Boltzmann constant, T is the carrier temperature, e is fundamental electronic charge, and n is the carrier density. The highest carrier density specified in the model must be used to estimate λ_D from the stability criterion. The mesh size must be chosen to be smaller than λ_D. Typically, mesh size is chosen around 1 – 5 nm.

The program sets the bias and calculates the electric field in each grid region

using Poisson's equation. Carrier drift and scattering processes for each electron or hole are then simulated according to all the read-in parameters and physics models. After simulation, the gain of each individual photo-generated carrier is extracted to calculate the excess noise factor. To aid structural design, in addition to the total output photo-generated current, the spatial distribution of the carrier impact ionization events can be calculated.

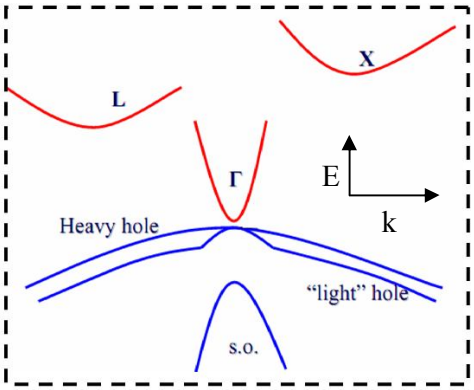


Figure 1.10. Simplified non-parabolic band structure used in Monte Carlo program.

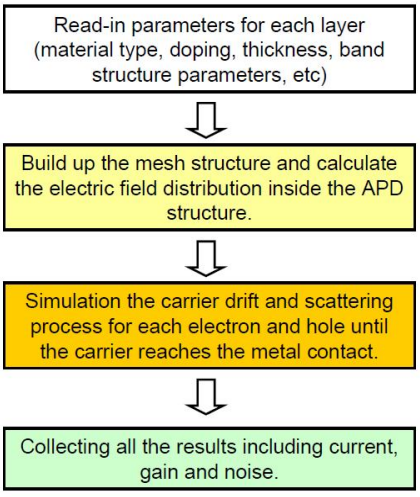


Figure 1.11. Structure flow of the simulation program.

1.5 Applications for high-power photodiodes

To transmit microwave signals through fibers, an analog optical link is required to convert the microwave signals into an optical format that propagates through the fiber and finally convert the optical signal back into the microwave domain in the receiver. The most basic transmission unit in Radio Frequency (RF) optoelectronics is an analog fiber optic link such as that shown in Fig 1.12. On the input side, a laser and modulator are used to impose the RF signals onto the optical carrier. An optical fiver is the transmission medium, and a photodetector detects the RF signal.

It is beneficial to reduce the loss of a transmission link. Due to the low-loss nature of optical fiber, the loss of an analog optical link is primarily dependent on the RF/optical conversion efficiencies of the modulators and photodiodes. At the photodiode side, the capability to handle very high photocurrent levels while maintaining the corresponding RF power output is essential to improve the loss and dynamic range of the analog optical link. In addition to high-power capability, the linearity performance of photodiodes is also important for high-performance analog optical links because it may be the limiting factor for the spur-free dynamic range of the link.

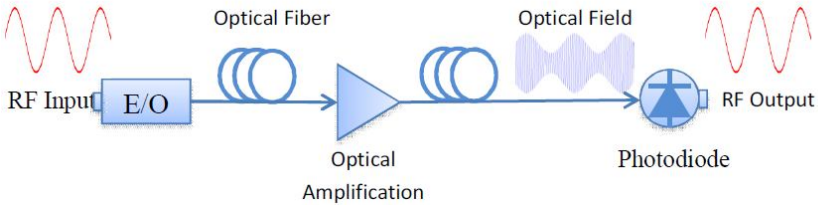


Figure 1.12. A simplified illustration of an analog optical link.

Another application of high-power photodiode is radio-over-fiber systems that operate at millimeter-wave frequencies. In conventional electronic systems an electronic mixer, which combines the local-oscillator (LO) and the intermediate-frequency (IF) signals to create an up-converted radio frequency (RF) signal, is used. Photodiodes, when operated under low bias, can exhibit very nonlinear behavior. This enables them to be used as optoelectronic mixers to directly up-convert IF signals. Compared to traditional electrical domain mixing methods, optical mixing in a nonlinear photodiode not only eliminates the complex electrical mixing circuit but also provides the advantage of low loss optical transmission from central station to base stations through the low-loss optical fiber. It should be noted that the photodiodes operated in nonlinear mode as optoelectronic mixers have to provide high power and high conversion efficiency. These requirements challenge the power handling capability and the strength of nonlinearity of photodiodes.

1.6 Figures of Merit for high-power photodiode

The important figures of merit to characterize high-power photodiodes include responsivity, bandwidth, RF output power, saturation current, and linearity.

Responsivity

The responsivity is used to describe how effectively a photodiode converts light into electrical current, defined as the ratio of the output photocurrent to the

incident optical power in the units of amperes per watt (A/W). For a vertically illuminated photodiode, the responsivity can be calculated by:

$$\mathrm{R} = \frac{I_{ph}}{P_{opt}} = \frac{\eta_{ext}\lambda[\mu m]}{1.241} \tag{1.13}$$

$$\eta_{ext} = (1 - R_{surface})(1 - e^{-\alpha d}) \tag{1.14}$$

where I_{ph} is the photocurrent, P_{opt} is optical power, η_{ext} is the external quantum efficiency. λ is light wavelength, $R_{surface}$ is the surface reflectivity, and α and d are the absorption coefficient and thickness of the absorber, respectively. Methods to enhance responsivity include using thicker absorption region, anti-reflection (AR) coating, and optical resonant-cavities.

Bandwidth

The 3-dB bandwidth, is defined as the frequency at which the output RF power decreases by 3dB compared to its low frequency value. The bandwidth of a photodiode is primarily limited by its RC circuit response time and the carrier transit time, as shown by:

$$\begin{aligned} f_{3dB} &\approx \sqrt{\frac{1}{\frac{1}{f^2{}_{RC}} + \frac{1}{f^2{}_{tr}}}} \\ f_{RC} &= \frac{1}{2\pi R_{total} C_{PD}} \\ f_{tr} &\approx \frac{3.5v}{2\pi d} \end{aligned} \tag{1.15}$$

where R_{total} is the total resistance including the series resistance of photodiode and the load resistance (typically 50 Ω), C_{PD} is the junction capacitance of the photodiode,

and v is the averaged carrier drift velocity in the depletion layer (note that equation 1.15 only applies to P-I-N photodiode structures). For more complicated structures, the transit-time limited bandwidth cannot be easily calculated.

Output RF power

Typically, the power output refers to the RF power delivered to a 50 Ω load, which is given by:

$$P_{load} = \frac{1}{2} I^2{}_{AC} R_{load} \quad . \tag{1.16}$$

The typical RF power characteristics of high-power photodiodes is plotted in Fig. 1.13. At a fixed frequency, the output RF power increases linearly with input optical power until a certain power level at which the RF power starts to deviate from the linear curve. Saturation refers to the point at which the deviation is 1 dB. The corresponding RF output power and average photocurrent are referred to as saturation

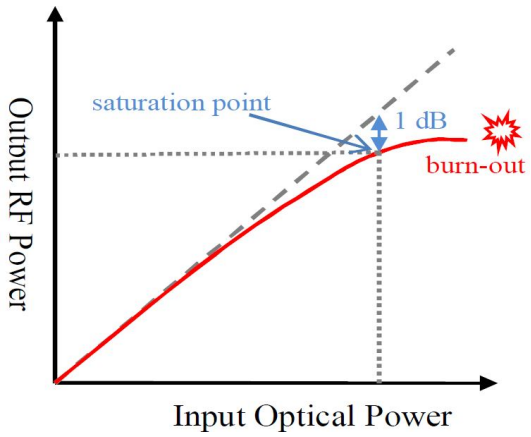


Figure 1.13. Illustration of saturation of output RF power and saturation point definition.

power and saturation current, respectively. The space-charge effect is one of the primary effects that cause saturation of output RF power. An illustration of the space-charge effect is shown in Fig. 1.14. At high photocurrent, the photo-generated electrons and holes induce a space-charge field that works against the electric field of the bias voltage. As a result, at certain positions in the photodiode, the total electric field starts to decrease, which, in turn, decreases the carrier velocity and saturation of the RF output power.

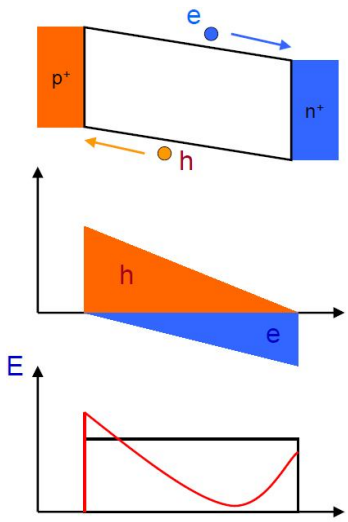


Figure 1.14. Band diagram, carrier distribution, and field distribution of a P-I-N structure.

1.7 Monte Carlo simulation of high-power photodiode

As discussed in section 1.3, the Monte Carlo program will first set the bias and calculate the electric field, and then simulate carrier transport process of each photo-generated carrier, i. e., the simulation operates in a loop of carriers. However, this is

not suitable for simulation of high-power photodiodes. Due to the space-charge effect, the electric field inside a high-power photodiode will change with time. Therefore, the program should run in a time loop. At each time interval, carrier the transport process is simulated for each single carrier. Then the new charge distribution and electric field are calculated. In the next time interval, all carriers will transported under the new electric field, as shown in Fig. 1.15. The time interval Δt must be chosen to be related to the plasma frequency,

$$\omega_p = \sqrt{\frac{e^2 n}{\varepsilon_s m}} \tag{1.17}$$

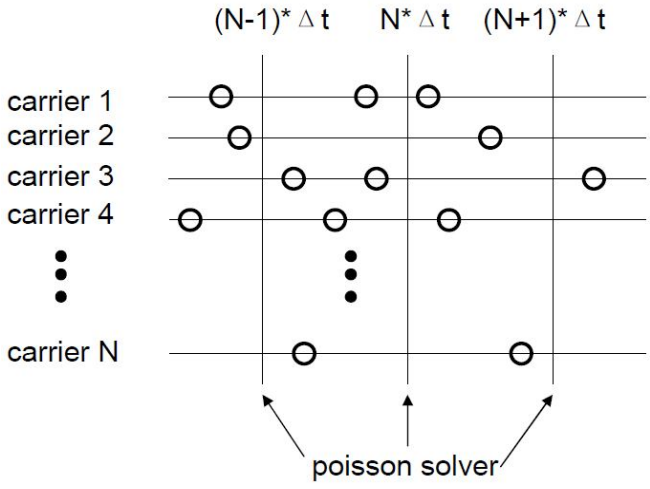


Figure 1.15. An illustration of simulation running in loop of time.

where e is the magnitude of the electronic charge, ε_s is the dielectric constant of the material, m is the effective mass of carrier, and n is the carrier density. From the viewpoint of the stability criterion, Δt must be much smaller than the inverse of the

plasma frequency, i.e., $\Delta t << 1/\omega_p$. Typically, Δt is chosen to be ~ 0.02 ps. Instead of Figure 1.11, the flowchart for simulation of high-power photodiode is shown in Figure 1.16.

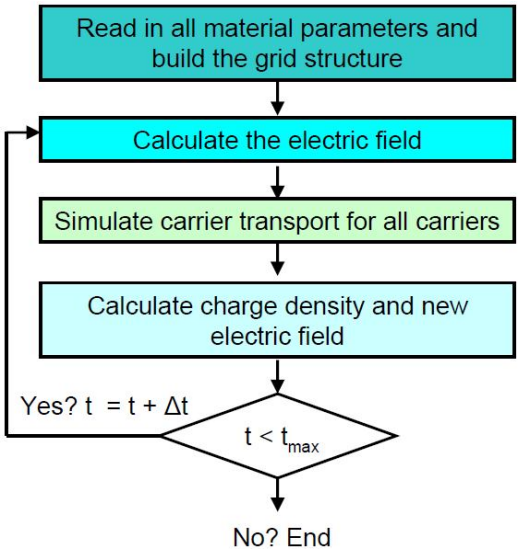


Figure 1.16. Flowchart of Monte Carlo program for simulation of high-power photodiode.

1.8 Thesis Organization

The main focus of this work is to theoretically study noise properties of avalanche photodiodes and high-power photodiodes using Monte Carlo simulation techniques. Chapter 2 presents the Monte Carlo simulation of an InAlAs/InAlGaAs tandem APD structure. A modified tandem structure is proposed to achieve even lower excess noise. Chapter 3 presents a systematic study of InAs avalanche photodiodes. Extremely low noise behavior is confirmed for those APDs. In chapter 4, to further improve the InAs APD performance, two new InAs APD structures are designed.

Lower dark current and much higher gain are demonstrated. Chapter 5 presents a theory on the linear relationship between excess noise factor and gain when non-local effect becomes significant. The Monte Carlo simulation is used to verify mathematical conclusions. In chapter 6, Monte Carlo simulation of APD performance under high-power condition is performed. Non-linear properties of APDs and a bandwidth enhancement effect are studied. Chapter 7 focuses on phase noise in photonically generated microwave signals. Monte Carlo simulation is used to study the imbalance effect and saturation effect in phase noise. Finally, the dissertation is summarized with proposed future research plans in chapter 8.

2. Three-Stage InAlAs/InAlGaAs Tandem Avalanche Photodiode

2.1 Introduction

As discussed in chapter 1, a promising approach for low-noise APDs is the cascade or tandem structure, which consists of a series of multiplication regions, all operated at relatively low gain [29], [30]. The interconnection of two adjacent multiplication cells can significantly enhance impact ionization of the carrier type with higher ionization rate, and suppress impact ionization of the carrier with lower ionization rate. Recently, W. R. Clark, et al., have successfully demonstrated a 3-stage tandem APD. They report a low k factor of 0.1 [31]. With heterojunctions inside each multiplication cell, most impact ionization events occur in low band-gap and high ionization rate layers (InAlGaAs). It is possible to ensure electrons are injected into high ionization rate layers from the InAlAs region with high electric field so that they have high energy, while holes are injected from the InAlAs region with low electric field.

A detailed description of the tandem APD structure is presented in Ref. [31]. It consists of a 1 μm-thick InGaAs absorption layer lattice-matched to InP, followed by three sequential multiplication cells. Fig. 2.1 shows the layer structure of a single multiplication cell, which contains a 100 nm InAlAs hole relaxation layer, a 100 nm p-type InAlAs layer in which the electric field increases, a 200 nm InAlAs "acceleration" layer, a 10 nm InAlAs layer with decreasing electric field, and a 100

nm InAlGaAs (band-gap = 0.92 eV) impact ionization layer.

InAlAs	uid	100 nm	1
InAlAs	p	100 nm	2
InAlAs	uid	200 nm	3
InAlAs	n	10 nm	4
InAlGaAs	uid	100 nm	5

Figure 2.1. Layer structure of one multiplication cell.

2.2 Monte Carlo Simulation and Structure Re-design

The Monte Carlo simulation model described in chapter 1 has been used to simulate InAlAs/InAlGaAs tandem avalanche photodiodes. To establish the input file for the tandem APD structure, the material parameters of InAlAs and InGaAs were obtained from simulations of homojunction p-i-n structure APDs. Some important parameters are listed in Table 2.1. Parameters for the InAlGaAs multiplication layer were obtained by linear interpolation of InAlAs and InGaAs using the algorithm in Ref. [32]. For InAlAs p-i-n structures with different thicknesses of the i-region, Fig. 2.2 shows that the simulation results agree well with experimental measurements of gain versus voltage (Fig. 2.2(a)) and excess noise versus gain (Fig. 2.2(b)) reported in Ref. [33]. Figure 2.3 shows that similar simulations on InGaAs p-i-n APDs are consistent with gain and excess noise measurements in Ref. [34]. Impact ionization coefficients (Fig. 2.4) are also calculated by averaging the distance travelled between every two successive impact ionization events initiated by one single carrier.

	InGaAs	InAlAs	InAlGaAs
Conduction band-gap (eV)			
E_{Γ}	0.79	1.53	0.92
E_L	1.32	1.77	1.44
E_X	1.30	1.82	1.51
Valence band-gap (eV)			
E_{hh}	0.0	0.0	0.0
E_{lh}	0.05	0.05	0.05
E_{SO}	0.34	0.34	0.34
Impact Ionization parameter			
Electron E_{th} (eV)	1.246	1.76	1.322
Hole E_{th} (eV)	1.246	1.76	1.322
Electron C_{ii} (s^{-1})	2.00×10^{13}	2.00×10^{15}	4.95×10^{14}
Hole C_{ii} (s^{-1})	1.98×10^{13}	5.20×10^{11}	1.53×10^{13}

Table 2.1. Band parameters and impact ionization parameters of InGaAs, InAlAs, and InAlGaAs.

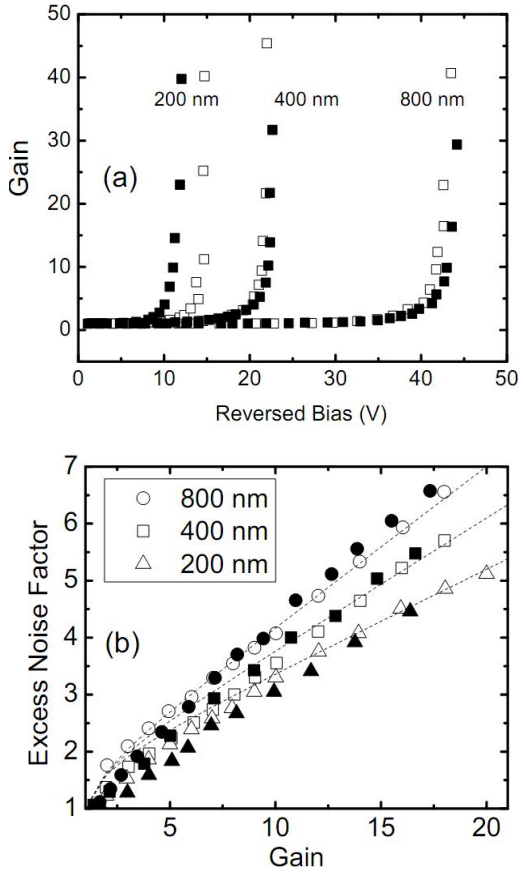


Figure 2.2. Simulation (open symbols) and experimental data (filled symbols) of (a) gain versus bias voltage and (b) excess noise versus gain for InAlAs homojunction p-i-n structures with i-region thickness of 200 nm, 400 nm, and 800 nm.

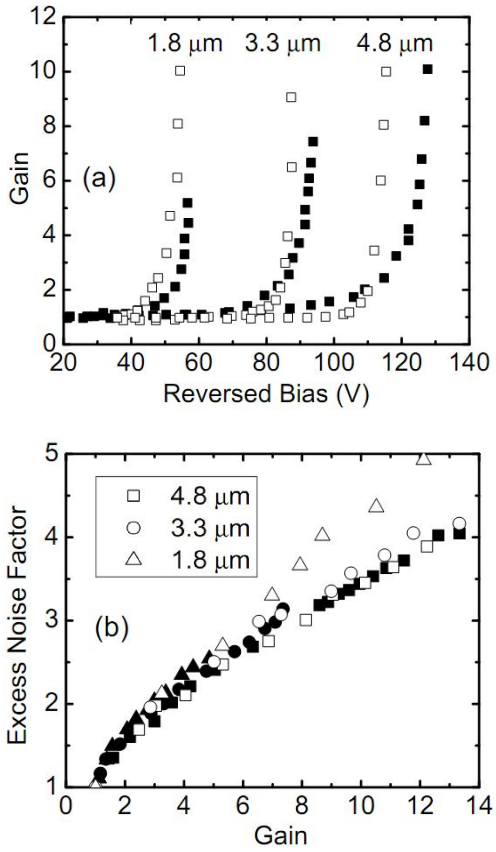


Figure 2.3. Simulation (open symbols) and experimental data (closed symbols) of gain versus bias voltage and excess noise versus gain for InGaAs homojunction p-i-n structures with i-region thickness of 1.8 μm, 3.3 μm, and 4.8 μm.

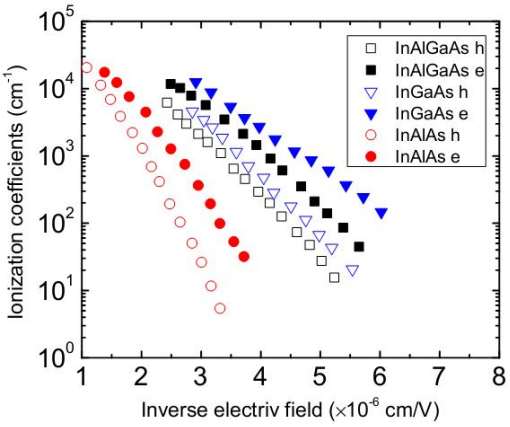


Figure 2.4. Calculated ionization coefficients of InGaAs, InAlAs, and InAlGaAs.

The energy band diagram of one multiplication cell is shown in Fig. 2.5. The intrinsically doped quaternary layer and 10 nm n-doped InAlAs layer form a large valence band discontinuity that presents a barrier to hole transport. Since InAlGaAs has a smaller band-gap than InAlAs and, thus, lower threshold energy for impact ionization, high impact ionization rates are anticipated for carriers injected into the InAlGaAs layer. The electric field was calculated by solving Possion's equation. The field at unity-gain is plotted in Fig. 2.6. Hole transport at heterojunctions was treated quantum mechanically by solving Schrodinger's equation. It is calculated that when the energy of a hole is smaller than the barrier height, the tunneling probability is almost zero for a 10 nm or thicker barrier. Hence, we assume that holes can only surmount heterojunction barriers by thermal emission, otherwise, elastic reflection is assumed. For the case that a carrier is unable to surmount a heterojunction barrier, we record number of times, N, that it is reflected. If this number is larger than a predefined threshold value, T, the carrier is assumed to be "trapped" and is no longer tracked. Empirically, it was found that T = 12 yielded good convergence.

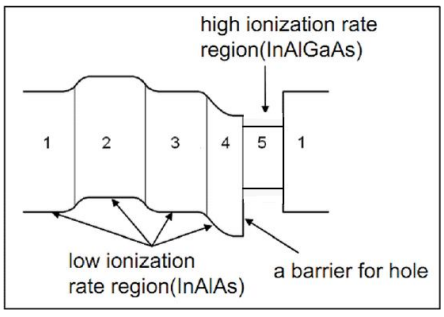


Figure 2.5. Band diagram of one multiplication cell.

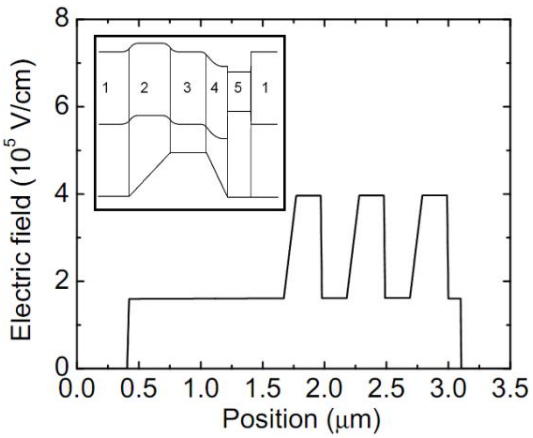


Figure 2.6. Electric field profile of a three-stage tandem APD at unity gain. The inset shows the electric field in each layer of one multiplication cell (not to scale).

Figure 2.7 shows the simulated (open symbols) and measured (closed symbols) [35] excess noise versus gain, M, of the tandem APD structure. The simulation does not yield as low noise below gain of 10 as the measured values but agrees well at higher gain. Both simulation and measurement show an increase in the k factor near M = 10. This effect is caused by the valance band discontinuity (labeled "barrier for

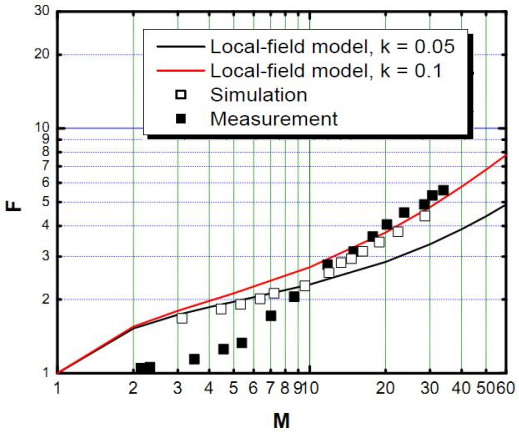


Figure 2.7. Simulated (open symbols) and experimental (closed symbols) data of three-cell tandem InAlAs/InAlGaAs APD.

hole" in Fig. 2.5). At low electric field, the barrier effectively blocks hole transport owing to the relatively low energy of holes. With increasing field, more holes can surmount the barrier and initiate impact ionization events. This is illustrated by the spatial distribution of ionization events plotted in Fig. 2.8. As gain increases from 10

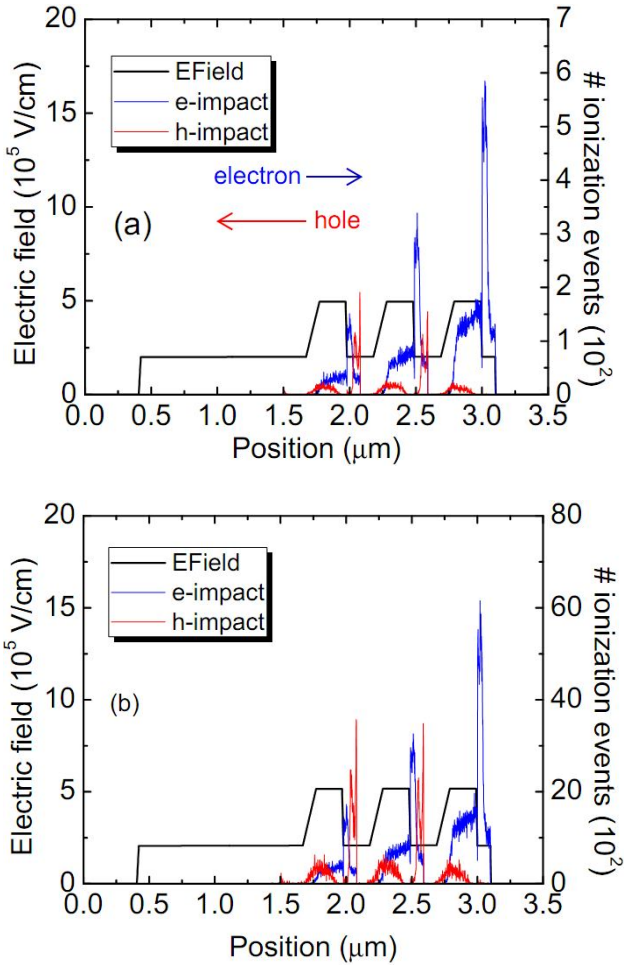


Figure 2.8. Simulated spatial impact ionization distribution of tandem APD at a) gain = 10 and b) gain = 100.

to 100, there is a significant increase in hole initiated impact ionization as a result of

the fact that an increasing number of holes can cross the barrier and impact ionize. This will cause an increase in the excess noise. To confirm this hypothesis, we have calculated the ratio of reflection to transmission at the hole barrier denoted in Fig. 2.8. The ratio drops significantly with increasing gain. Let n be the label for the n^{th} multiplication region in the tandem structure. The impact ionization distributions in Fig. 2.8 indicate that holes exiting from the $n+1^{th}$ multiplication cell enter the n^{th} cell and can impact ionize in its InAlGaAs layer (see Fig. 2.10). To suppress hole-initiated ionization, we have investigated increasing the thickness of the relaxation layer in the multiplication cells [36]. This has the benefit of allowing the holes to cool before they enter the InAlGaAs multiplication layer. For the excess noise simulation in Fig. 2.11, the thickness was increased from 100 nm to 500 nm. The k factor for the thicker hole relaxation layer is less than 0.05 for gain values up to ~ 100.

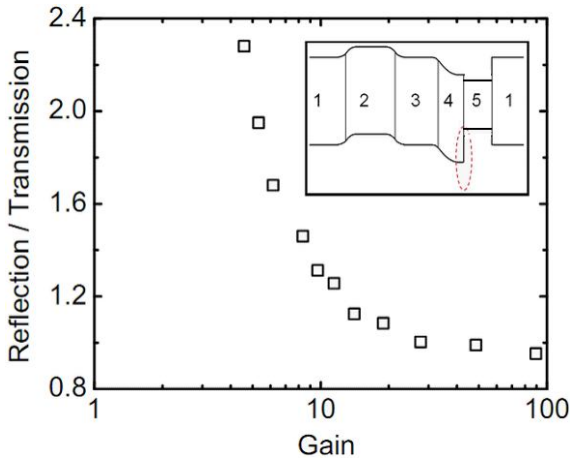


Figure 2.9. Calculated ratio of reflection to transmission versus gain. Inset illustrates the barrier.

The simulation also shows that the average energy of holes is < 0.12 eV before they enter the InAlGaAs layer, which effectively reduces hole-initiated impact ionization. This is illustrated by the spatial ionization distribution plotted in Fig. 2.12. Comparing it with Fig. 2.8(a) shows significantly reduced hole ionization.

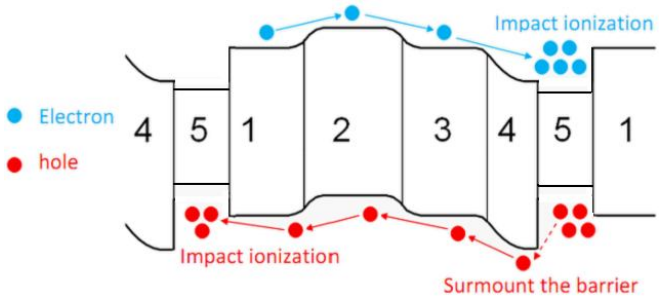


Figure 2.10. Illustration of hole impact ionization.

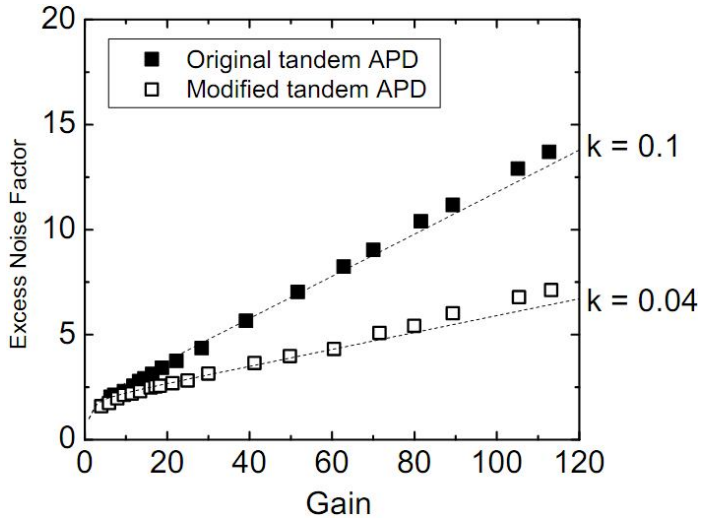


Figure 2.11. Simulated excess noise of tandem APD with a 500 nm relaxation layer.

The 3-dB bandwidth was determined by Fourier transform of the impulse response. Compared to a conventional APD, the tandem APD exhibits significant

decrease in the gain-bandwidth product as shown in Fig. 2.13. The simulation also indicates that the tandem APD structure with a longer hole relaxation layer has an even lower gain-bandwidth product of 50 GHz. With a longer relaxation layer and the concomitant reduction in hole energy results in longer times for the holes to surmount the valance band barrier.

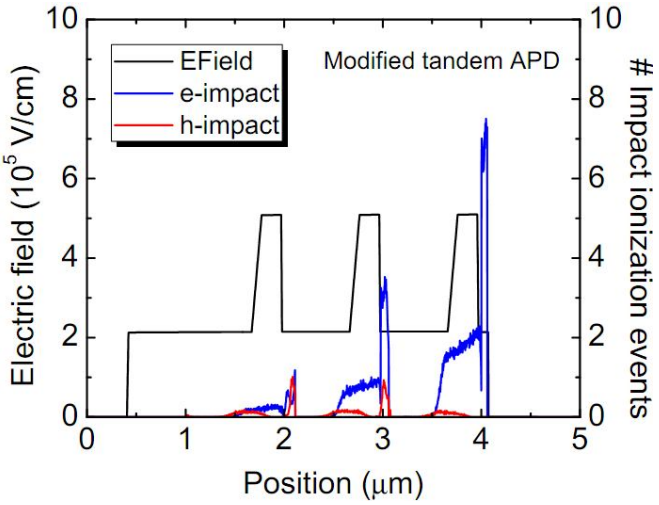


Figure 2.12. Simulated spatial impact ionization distribution with 500 nm relaxation layer at gain = 10.

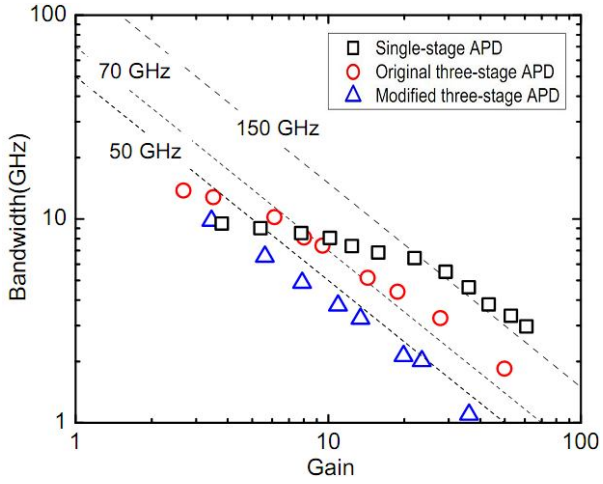


Figure 2.13. Simulated gain-bandwidth product of one-stage and three-stage APDs.

2.3 Experimental InAlAs/InAlGaAs Tandem APD

Recently, Optogration Inc. has grown this modified structure and fabricated the wafer into devices. The measured current-voltage characteristics and multiplication gain curve are shown in Fig. 2.14 and Fig. 2.15, respectively. Our noise measurement is shown in Fig. 2.16. The excess noise factor is indeed smaller than the original tandem APD structure, however, not as low as the simulation results, which might be due to the discrepancy between the wafer and the design since this is a very complex structure that requires precise control of the doping, composition, and thickness of many epitaxial layers. To illustrate this, we have simulated the noise performance of this tandem APD structure with certain "error" in doping concentration. As demonstrated in section 2.2, the electric field distribution at each stage should be very similar; otherwise, some stages are not active and total noise performance is degraded. This requires that the product of doping and thickness of p layer should roughly equal to that of the n layer in each stage. Simulation of excess noise with different mismatch level between p and n layer for each stage is shown in Fig. 2.17. Note that, 5% mismatch means a difference between the product of doping and thickness of p layer and n layer of 5% of the averaged value of the p and n layers. Clearly, the mismatch, even 20%, could result in significant degradation of excess noise performance.

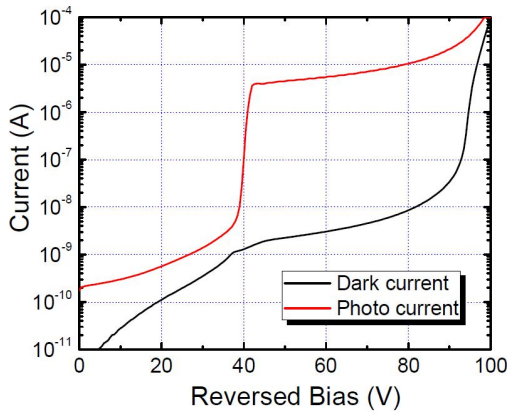


Figure 2.14. The measured current-voltage characteristics of a modified tandem APD structure.

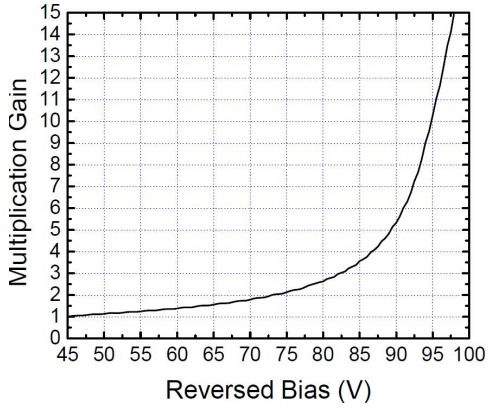


Figure 2.15. The measured multiplication gain versus reversed bias of modified tandem APD structure.

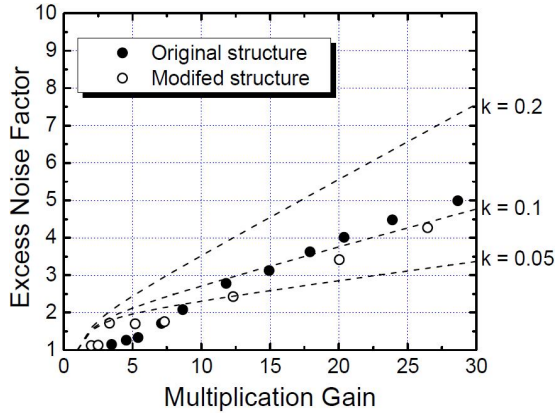


Figure 2.16. Comparison between noise measurement of original tandem APD structure and modified structure.

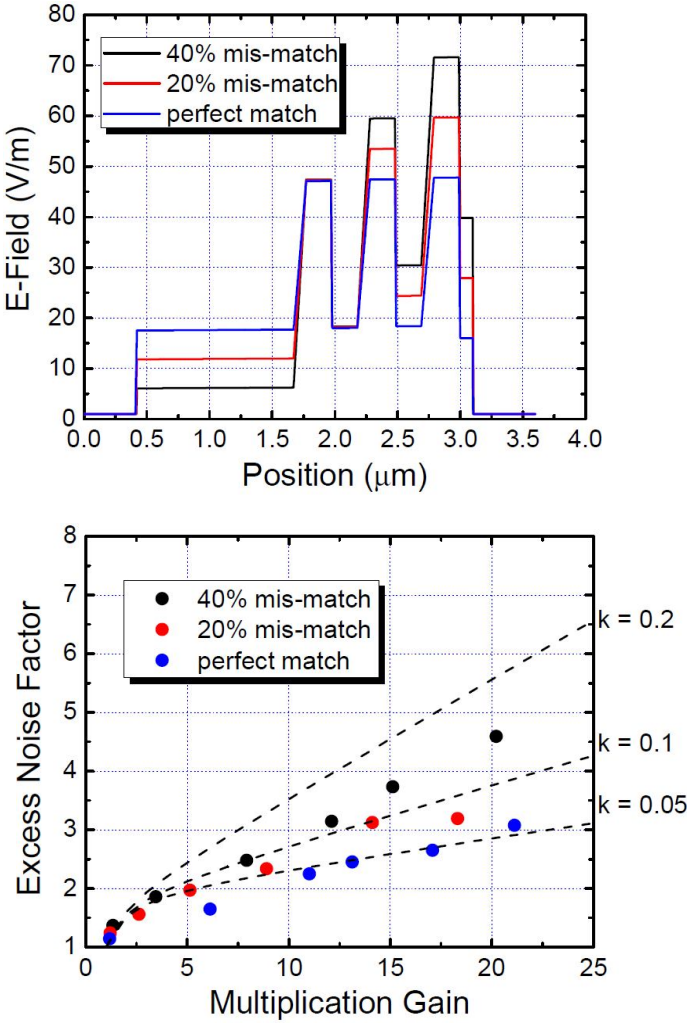


Figure 2.17. (Top) Simulation of electric field profile with different mismatch percentage. (Bottom) Simulation of the excess noise with different mismatch percentage.

2.4 Chapter summary

In this chapter, Monte Carlo simulation is performed on an InAlAs/InAlGaAs tandem APD structure that was reported to have low excess noise factor. Consistency is achieved between our simulation and experimental data. A modified structure is

proposed with a longer relaxation layer to suppress hole impact ionization events. Simulation indicates that this structure has k value as low as 0.05. Optogration Inc. has grown and tested this new structure and the noise performance indeed improves, however not as much as expected. One possible reason is that it is a difficult structure to grow and small changes in the doping and thickness of the epitaxial layers can degrade performance, as illustrated by further Monte Carlo simulations.

3. Low-Noise and High Gain InAs Avalanche Photodiode

3.1 Introduction

By taking advantage of the non-local effect or using the I^2E technique, we can achieve low excess noise. However, as stated in the previous section, the complex I^2E structures require high-quality growth and precise control of doping concentrations. It has been demonstrated that $Hg_{0.7}Cd_{0.3}Te$ and InAs have k ~ 0, which enables use of simple p-i-n structures to achieve low excess noise. $Hg_{0.7}Cd_{0.3}Te$, has demonstrated excellent APDs characteristics [37-38]. The exponential increase in gain with bias and extremely low measured excess noise (F(M) ~ 1 for M up to 100) suggest that the ratio of the hole and electron ionization coefficients, is ~ 0. However, HgCdTe remains inaccessible to many and challenging to grow and fabricate. Also, the extremely small bandgap of HgCdTe (0.29 eV) results in relatively high dark current making it unattractive for APD receivers at room temperature. Recently, InAs APDs have also demonstrated k ~ 0 with moderately low dark current at room temperature [37]-[40]. I have used my Monte Carlo model to better understand the device physics of the APDs. Good agreement is obtained between the simulations and measurements. The Monte Carlo model is described in chapter 1. Some important model parameters for InAs are listed in Table 3.1. To test my model and the parameters, I simulated electron drift velocities in bulk InAs at different electric field strengths. The results agree well with those reported in [41], as shown in Fig. 3.1. The

decrease of the electron drift velocity at higher field is attributed to inter-valley scattering. The saturation velocity of electrons in InAs is demonstrated to be close to 1×10^5 m/s, which is faster than that of InGaAs.

Material parameters	Γ	L	X
Energy band-gap (eV)	0.417	1.133	1.433
Electron effective mass (m^*/m_0)	0.026	0.072	0.224
Threshold energy of impact ionization (eV)	0.82		
Longitudinal sound velocity (m/s)	4620		
Static dielectric constant	15.15		
High frequency dielectric constant	12.3		
Accoustic phonon energy (meV)	11.0		
Optical phonon energy (meV)	24.3		
Intervalley phonon energy (meV)	13.2		
Intervalley deformation potential (eV/m)	1.07×10^{11}		

Table 3.1. Parameters used in Monte Carlo simulation of InAs APDs.

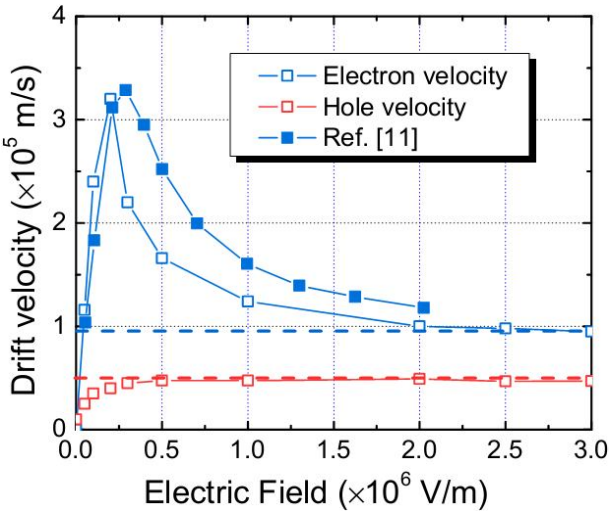


Figure 3.1. Simulated electron drift velocity of InAs (open symbol), compared to data in [41] (closed symbol).

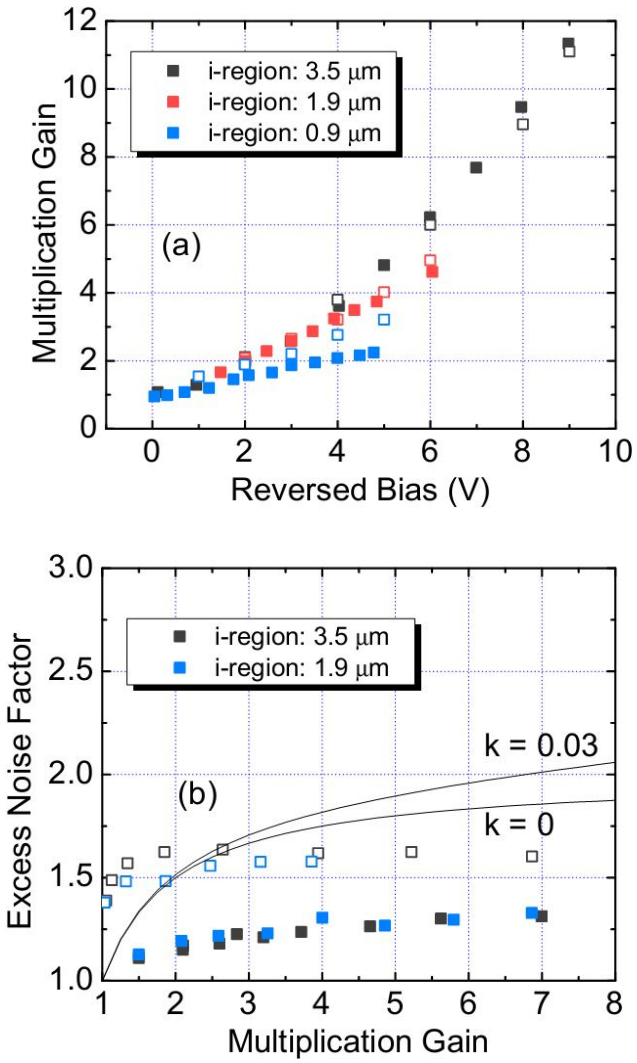


Figure 3.2. Simulated (closed symbols) (a) gain and (b) excess noise factor of InAs p-i-n structures with i-region thicknesses of 0.9 μm, 1.9 μm, and 3.5 μm, compared to measurements in [42] (open symbols).

The scattering parameters were determined by simulating the p-i-n structures with different i-region thicknesses reported in [42]. Pure electron injection is assumed for the simulations with i-region thicknesses of 0.9 μm, 1.9 μm, and 3.5 μm. The

simulated gain agrees with the experimental data for all three thicknesses, as shown in Fig. 3.2a. The excess noise simulations are consistent with previous measurements on InAs APDs and are consistent with pure electron ionization. Figure 3.3 shows the calculated occupancy percentage of electrons and holes in different valleys of the conduction band. We observe that a significant fraction of electrons populate the satellite valleys (L and X valleys) at low gain, which indicates that electrons in InAs can quickly gain energy and initiate inter-valley scattering or impact ionization. On the other hand, few holes occupy states outside the heavy-hole valance band and their energy increases marginally even at high electric fields. It follows that the ionization coefficient for holes is much lower than that of electrons and therefore k is close to 0, which is consistent with the observation of very low excess noise in InAs APDs. It is also shown by both experiment and simulation in Fig. 3.2a that for a given bias higher gain is achieved with thicker multiplication region, which suggests that a longer depletion region is preferred for high gain InAs APDs. Also a thicker depletion region can effectively suppress the tunneling component of the dark current owing to the thicker effective barrier and lower electric field. Low background doping is crucial to achieve thick depletion regions and gain enhancement. Figure 3.4 shows the Monte Carlo simulated gain for a p-i-n structure with 6 μm-thick depletion region at different background doping levels. It is clear that lower doping results in significantly higher gain.

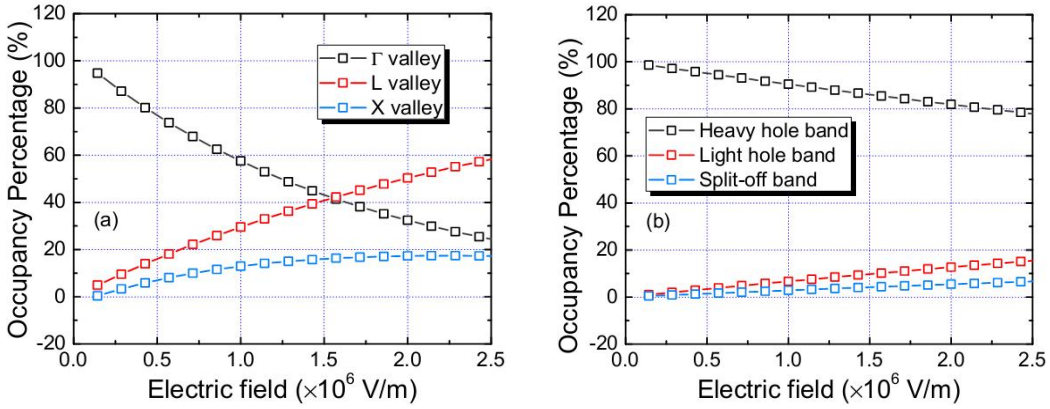


Figure 3.3. Simulated occupancy percentage for (a) electrons in Γ, L, and X valleys, and (b) holes in heavy-hole, light-hole, and split-off bands at different electric fields.

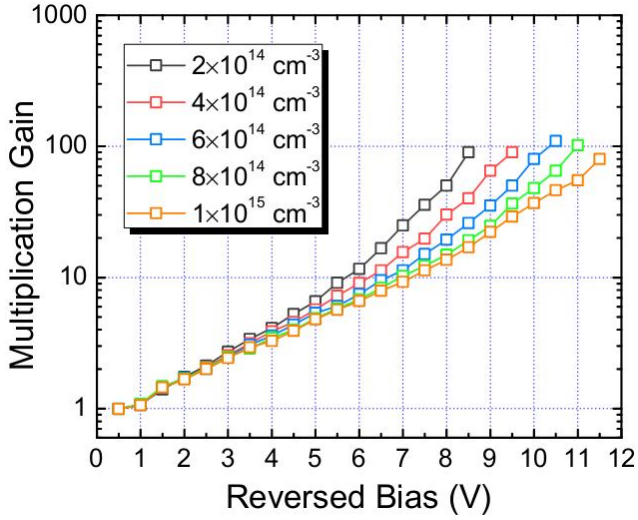


Figure 3.4. Simulated gain of a p-i-n structure InAs APD with 6 μm-thick i-region at different background doping levels.

3.2 Design and Measurement of InAs Avalanche Photodiodes

Two InAs APD structures with i-region as thick as 6 μm were grown using MBE techniques by Prof. Seth Bank's group at University of Texas at Austin. The

schematic cross sections are shown in Fig. 3.5. Beryllium and silicon were used as acceptors and donors, respectively. Using Hall effect measurement, the extracted n-type background doping concentration in the i-region was below 1×10^{15} cm^{-3}. For the graded p-doped structure, to further reduce the background doping the Be cell temperature was ramped in order to form graded p-type doping in the first 2 μm of the depletion region to compensate the n-type background doping. Mesa structure devices with diameters from 50 μm to 500 μm were defined by wet etching using 1:1:1 (phosphoric acid, hydrogen peroxide, de-ionized water), followed by 30 seconds etching in 1:8:80 (sulphuric acid, hydrogen peroxide, de-ionized water). This recipe has been shown to suppress surface leakage current [43]. The etched mesa sidewalls were immediately covered with SU-8 passivation to minimize surface degradation [44]. Due to the small band-gap of InAs, good ohmic contact is easily formed by depositing Ti/Au (20/150nm) without annealing.

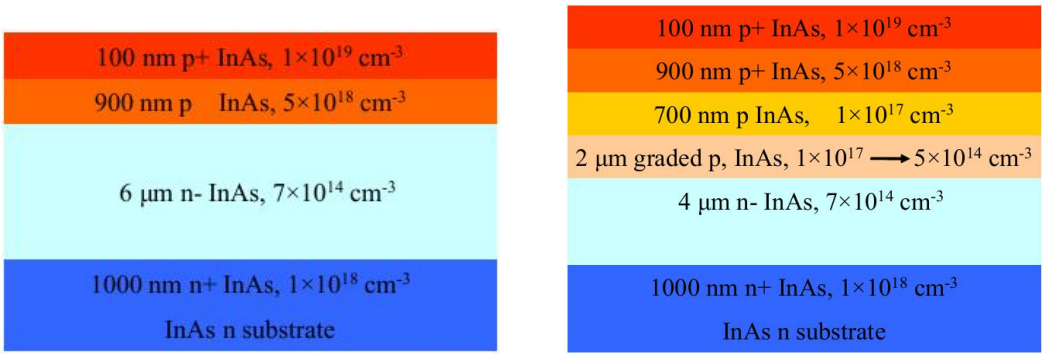


Figure 3.5. Layer structure of (top) Unintentionally-doped (UID) structure, and (bottom) Graded p-doped structure (graded).

The dark currents were measured using a HP 4145B semiconductor parameter analyzer. For gain and excess noise measurement, a phase-sensitive detection method described in [45] was used. The light source is chopped at 200 Hz, then AC photo-current and photo-noise are measured using two lock-in amplifiers. An advantage of this measurement technique is that it enables characterization of photodiodes with high dark current. Figure 3.6 shows the measured dark currents of 100 μm-diameter devices for both structures. The dark current of the UID structure is as low as that reported in [46], and even lower dark current is achieved on the graded structure due to reduced diffusion current. The dark currents at -10 V bias of devices with different areas are plotted in Fig. 3.7. We see that the dark current scales with the active area for both structures, which attests to the efficacy of the SU-8 surface passivation.

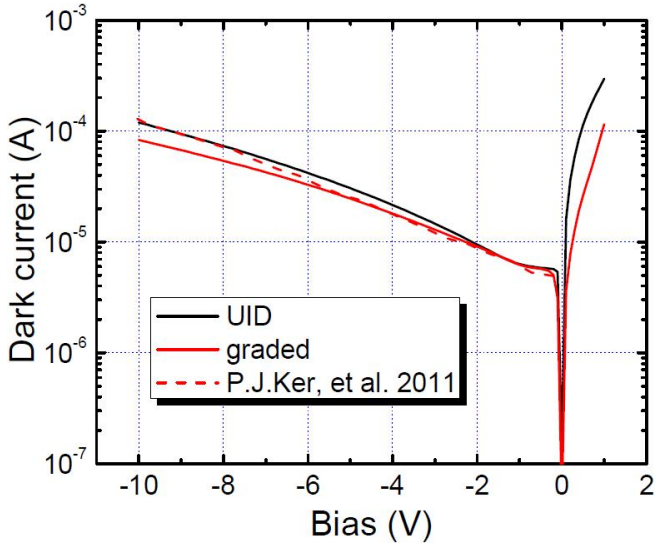


Figure 3.6. Measured dark currents of UID and graded structure devices with 100 μm diameter, compared to data in [46].

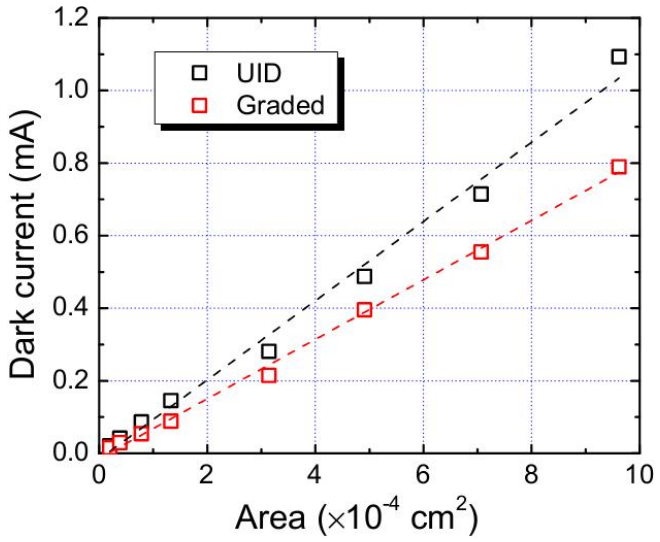


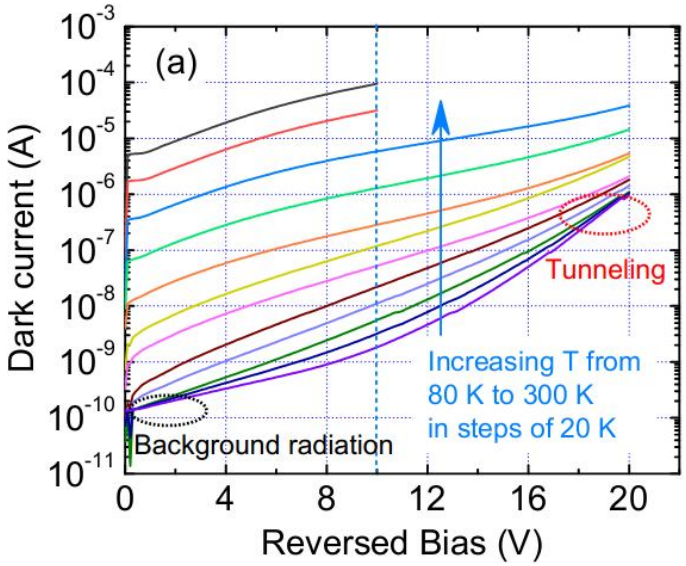
Figure 3.7. Dark currents of devices with different areas at bias = -10 V for both structures.

Figure 3.8a shows the dark currents of a 100 μm-diameter graded structure device at different temperatures. The tunneling current does not become significant until the bias exceeds 16 V, which confirms that the 6 μm-thick i-region suppresses the tunneling component of the dark current. The saturation of dark current below 100 K is due to background radiation. Unlike the work in [47], the size-dependent study of dark current in Fig. 3.8b indicates that the dark current of the graded structure InAs APDs is primarily bulk-dominated at all temperatures. Using Eq. 3.1 an activation energy, E_a, has been extracted from the dark current versus

$$I_d \propto T^2 \exp(\frac{-E_a}{kT}) \tag{3.1}$$

temperature after subtracting the background induced current. In eq. 3.1 k is the

Boltzmann constant and T is the absolute temperature. As shown in Fig. 3.9 when T is between 300K and 220K, E_a is ~ 0.42 eV, which is close to the band-gap of InAs. In this temperature range the dark current appears to be diffusion current. From 220K to 140K, the diffusion current decreases rapidly and can be ignored; the fitted E_a in this temperature range is ~ 0.19 eV close to mid-gap, an indication that generation-recombination inside the depletion region becomes dominate. The UID structure devices exhibit the same trend except dark current is slighter higher at all temperatures. Since dark currents for both structures are dominated by diffusion current, one reasonable explanation for the ~ 20% lower dark current of the graded structure is the slightly thicker p-region due to intentionally graded p-doping. According to the calculation of diffusion current in [48], a longer p-layer has smaller diffusion current if surface recombination is significant [48], which may be the case since the top surface is not passivated with SU8.



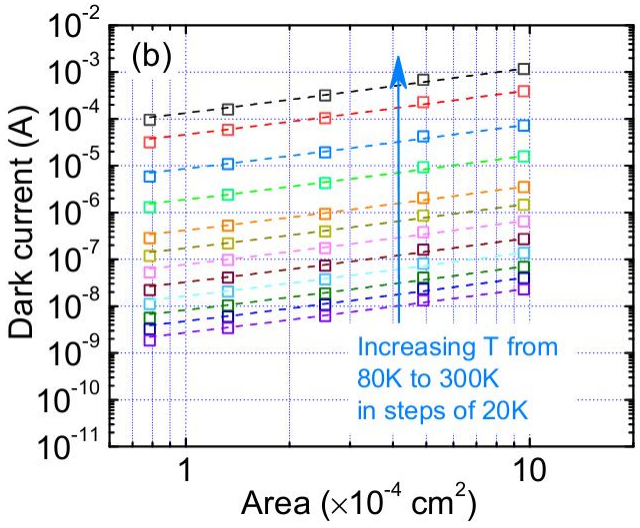


Figure 3.8. (a) Dark currents of a 100 μm-diameter graded structure device at different temperatures. (b) Dark currents of devices with different areas at different temperatures, and bias = -10 V.

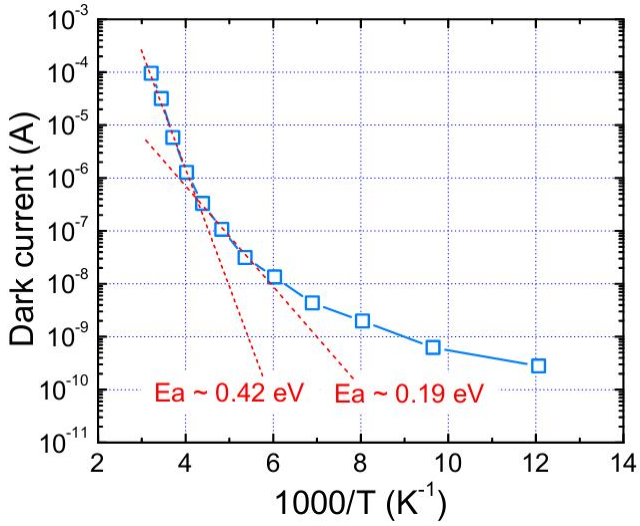


Figure 3.9. Dark current at bias = -10 V versus 1/kT for a 100 μm-diameter graded structure device.

Figure 3.10 shows that the measured gain is almost the same for both structures and increases exponentially with bias, showing no sign of breakdown, a

signature of k ~ 0. The highest gain that can be measured is limited by the device dark current, which will degrade the measurement accuracy above a certain level (~ 0.1 mA). For the UID structure the reversed bias was increased up to 10 V before reaching the measurement limit. The highest gain obtained is ~ 30, while for the graded structure, the slightly lower dark current enables higher bias voltages. The maximum gain, ~100, is achieved at 12 V reversed bias. The InAs APDs in Ref. [48] follow the gain curve of the APDs reported here except the maximum gain is 20. Figure 3.11 shows that the depletion width for a given bias is almost the same for the UID and graded structures. At high reverse bias, the depletion widths are ~ 6 μm. We note that comparable depletion width was also reported for the InAs APDs in Ref. [48]. This explains the similarity of the gain curves. On the other hand, the InAs APD in Ref. [49] has a depletion region of approximately 3 μm and the gain increases slowly, i.e., the gain is lower for the same bias, as shown in Fig. 3.10. Fig. 3.12 shows the measured and simulated excess noise factor versus gain for the UID and graded structures. These results are also consistent with k ~ 0. The measured gain versus bias voltage of the 6 μm i-region InAs APD in [47] and the graded structure reported here at 77K are plotted in Fig. 3.13. Similar to the report in [40], for a given bias voltage the multiplication gain decreases significantly at 77K, which is opposite to most III-V semiconductors. The differences between the gain in [47] and the graded structure are larger at 300K than at 77K because the gain of InAs APDs becomes insensitive to the depletion width at 77K [50]. To investigate the origin of smaller gain at 77K, Monte Carlo simulations were performed. Equation 3.2 [51] was used to model the

temperature dependence of the InAs band-gap. The energies of the L and X minima, E_L and E_X, respectively, are assumed to have the same

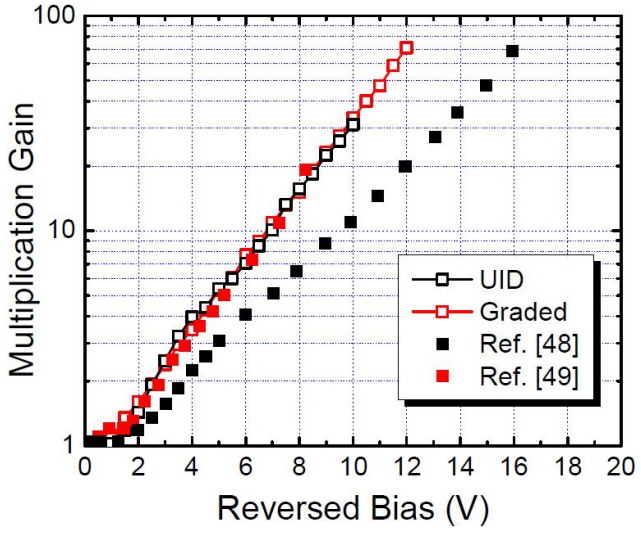


Figure 3.10. Measured gain (open symbols) of UID and graded InAs APDs compared to data in [48], [49] (closed symbols).

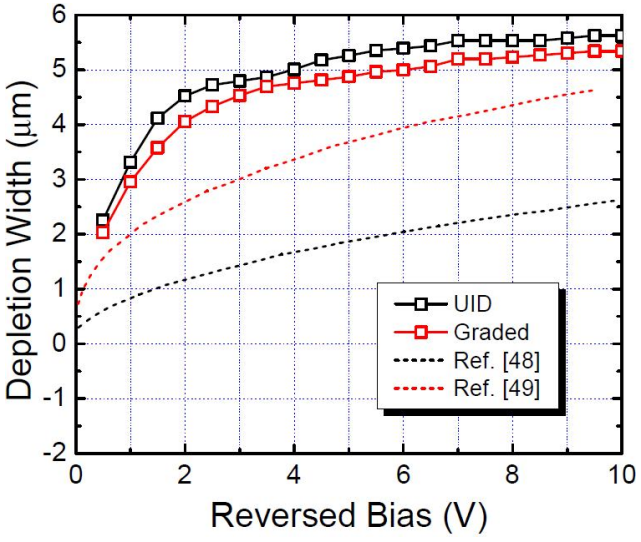


Figure 3.11. Depletion region thicknesses of UID and graded InAs APDs compared to data in [48], [49] (dash lines).

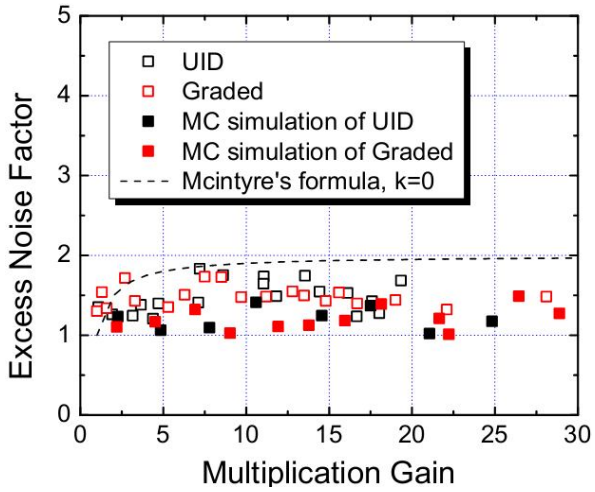


Figure 3.12. Measured (open symbols) and simulated (closed symbols) excess noise factor of the UID and graded InAs APDs.

temperature dependence as that of the Γ conduction band minimum, E_Γ [52]. First the model was calibrated by simulation of the InAs p-i-n structure with 3.5 μm thick i-region reported in [53]. As shown in Fig. 3.14, the simulated gain at both 300K and 77K agree well with the measurement in [53]. With the same set of parameters, simulation was then performed on the graded APD structure and, again, there is good agreement between experiment and simulation, see Fig. 3.13.

$$Eg = 0.415 - 2.76 \times 10^{-4} \frac{T^2}{T + 83} eV \tag{3.2}$$

As the temperature decreases, the total carrier scattering rates decrease significantly. Consequently, the carriers will experience less phonon scattering and gain energy faster, however, the bangap, and thus, the threshold energy for impact

ionization increase at lower temperature, making it harder for carriers to initiate impact ionization events. For typical III-V materials, such as InP, InAlAs, and GaAs the first mechanism is more pronounced and higher gain is expected with decreasing temperature. On the other hand, InAs exhibits the opposite. If we look at the simulated electron scattering rates (Fig. 3.15) for different scattering mechanisms, there are basically two regions: 1) when the electron energy is < 1 eV, intervalley scattering due to non-polar optical phonons is marginal and electrons will primarily stay in the Γ valley. In this energy regime, impurity and polar optical phonon scattering dominate. Since polar optical phonon scattering involves emission or absorption of optical phonons, we focus on phonon scattering for this case.

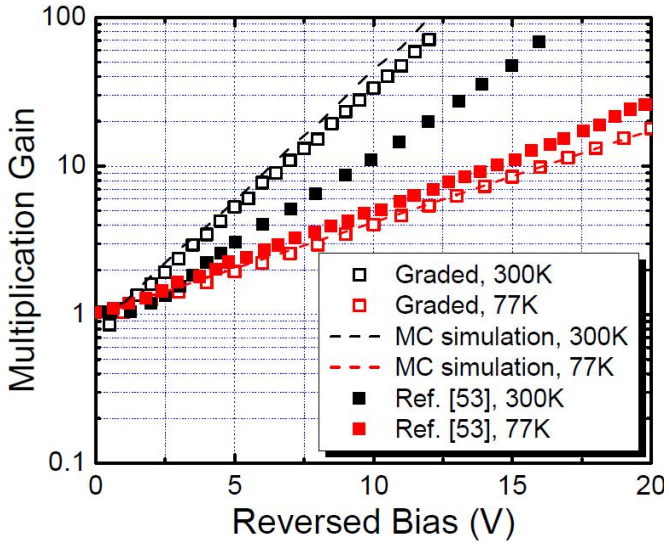


Figure 3.13. Measured (open symbols) and simulated (dash lines) gain of the graded structure InAs APD at 77K and 300K, compared to data in [53].

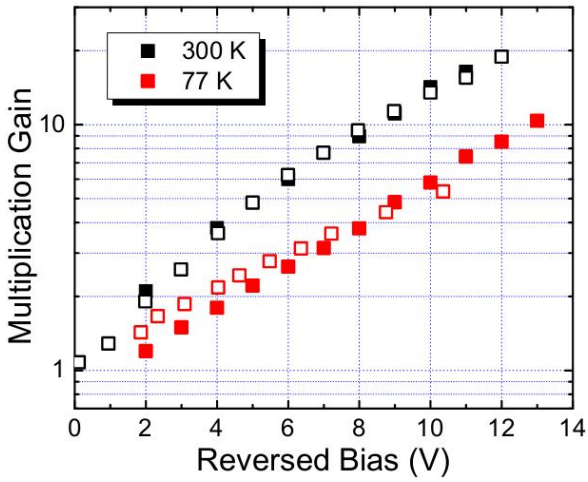


Figure 3.14. Measured gain versus bias voltage reported in [53] (open symbols) and simulated (closed symbols) gain of InAs p-i-n structure with 3.5 μm thick i-region at both 300K and 77K.

$$< n >= \frac{1}{\exp(\frac{\hbar\omega}{kT}) - 1} \tag{3.3}$$

2) On the other hand, when the electron energy is > 1 eV, intervalley scattering and impact ionization rates increase rapidly. For an APD, the impact ionization coefficients depend on how fast a carrier can attain threshold energy, and, for electrons, the two most important processes are 1) phonon-scattering at energy < 1eV, which determines the rate of increase of energy and 2) the impact ionization scattering rate, which determines the threshold energy and the electron ionization rate, or simply the "destination" of electron energy. In Fig. 3.15, when the electron energy is < 1 eV, although impurity scattering is smaller at 77K than that at 300K, we note that the optical phonon emission rate remains almost unchanged. Thus, in InAs the electrons still experience numerous phonon scattering events at 77K. In addition, the absorption

of optical phonons decreases dramatically due to the significantly decreased number of phonons according to the Bose-Einstein distribution (see Eq. 3.3). Hence it becomes harder for electrons to gain energy. The influence of higher threshold energy that results from the larger band-gap at 77K causes the ionization coefficient of the electron to decrease, resulting in the much smaller gain observed in Fig. 3.14 at 77K. The low gain at 77K appears to restrict the operation of these APDs to the temperature range 200K to 300K, which can be achieved with two- or three-stage thermoelectric coolers.

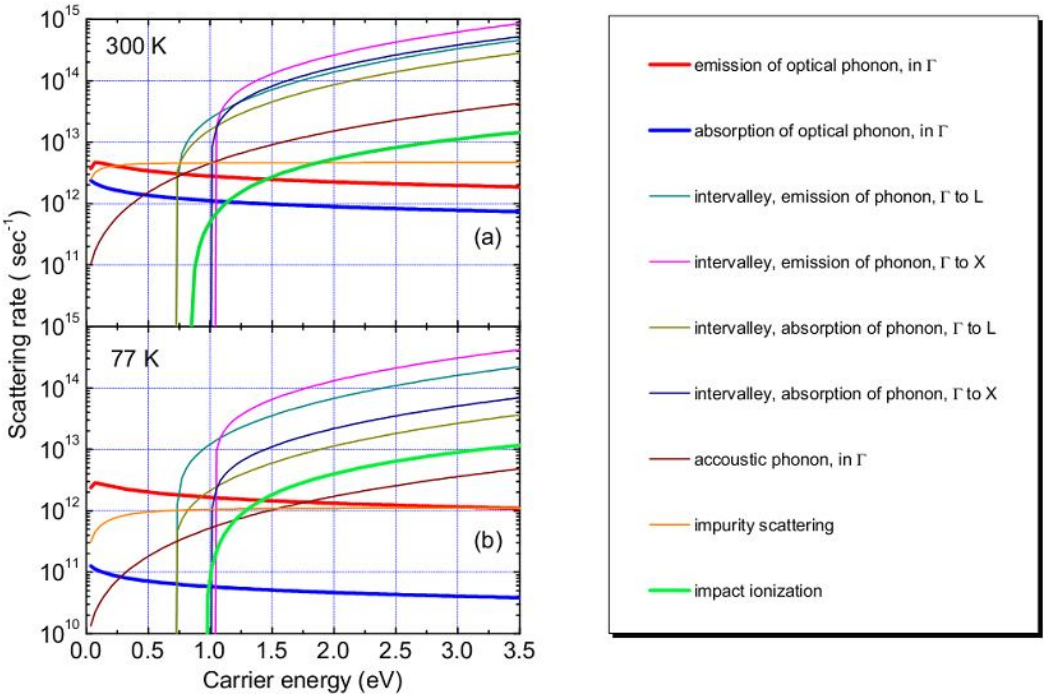


Figure 3.15. Calculated electron scattering rates in an InAs APD at (a) 300 K and (b) 77K.

The bandwidth was measured with a HP 20 GHz lightwave component analyzer. Figure 3.16 shows the measured frequency response for various bias voltages (open symbols) of a 70 μm-diameter graded structure device that was bonded to a 50 Ω G-S-G contact pad. The measured bandwidth is in the range 2 GHz to 3 GHz independent of bias voltage. This is less than the 15 GHz RC bandwidth determined from measured S-parameters, which indicates that the speed of these APDs is transit-time limited. Monte Carlo simulation using the method described in [54] was employed to verify the transit-time response. Impulse responses are plotted in Fig. 3.17. The pulse widths at different bias are nearly the same. The frequency responses, shown as solid lines in Fig. 3.16, are the Fourier transform of the impulse responses. Since the speed is limited by the transit time, a shorter depletion region is required to speed up carrier transport, however, as indicated by the simulated gain curves in Fig. 3.2a a shorter depletion region has lower gain for a given voltage. Increasing the voltage to recover the gain will result in higher dark current, particularly if tunneling becomes significant. Although the bandwidths of the thick depletion layer APDs may be too restrictive for telecommunication applications, they may be sufficient for imaging systems, where bandwidths of several GHz are widely used. The measured and simulated bandwidths versus gain are plotted in Fig. 3.18. Consistent with the measurements reported in [49] there is no evidence of a finite gain-bandwidth product, as would be expected for $k \sim 0$.

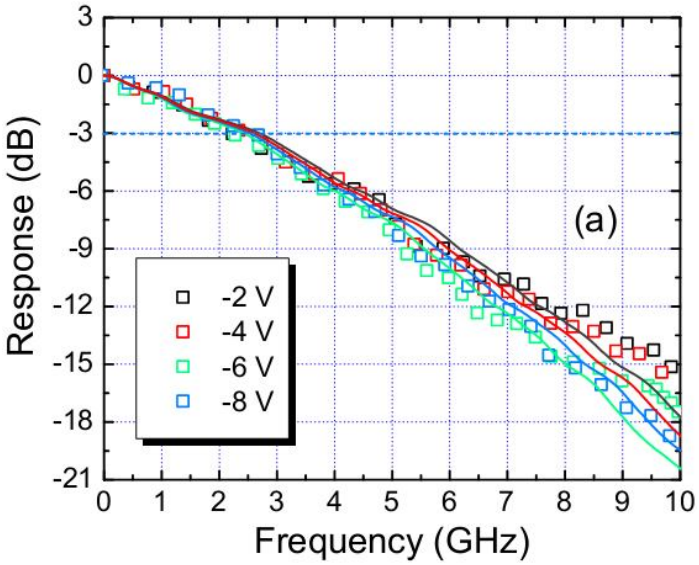


Figure 3.16. Measured (dots) and simulated (lines) frequency responses of the graded structure InAs APD at bias = -2, -4, -6, and -8 V.

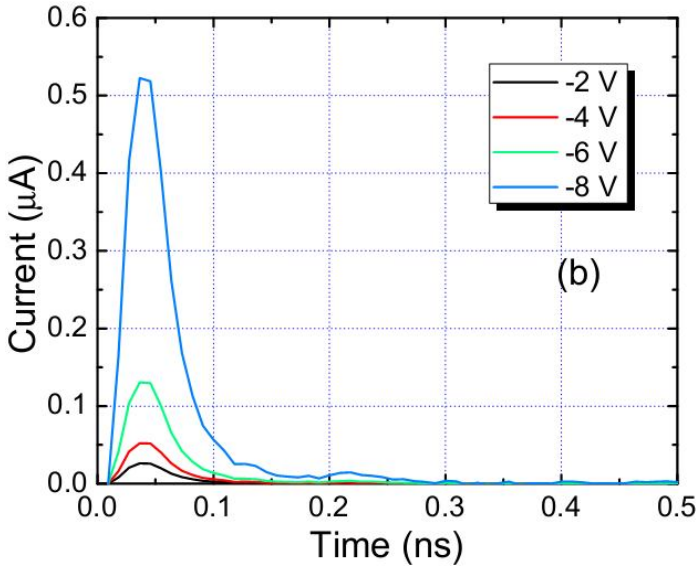


Figure 3.17. Simulated impulse responses of the graded structure InAs APD.

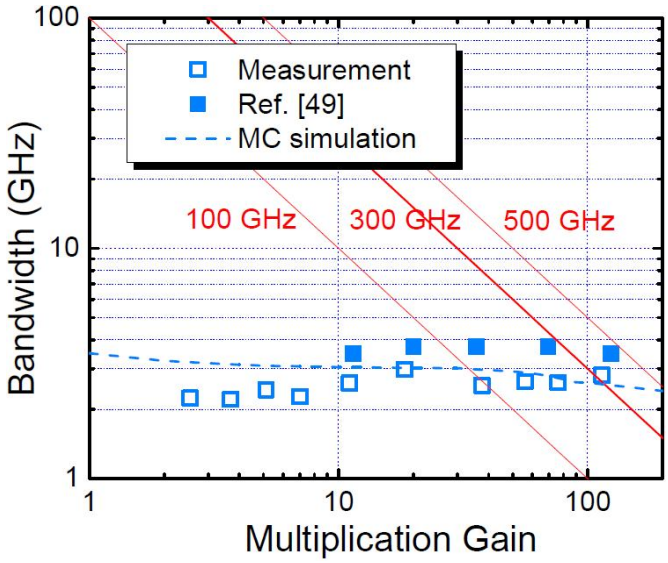


Figure 3.18. Measured (open symbol) and simulated (dash line) bandwidths of the graded structure InAs APD, compared to data in Ref. [49].

3.3 Chapter Summary

In this chapter, Monte Carlo simulation of InAs APDs is performed and consistency is achieved between simulations and previously reported experimental data. Then two InAs APD structures with 6 μm-thick depletion region are designed, fabricated and characterized. The dark currents of those APDs are comparable to currently reported record data, as well as measured multiplication gain. Noise measurements confirmed that the k value is close to zero at gain as high as 30. The bandwidth measurement demonstrates the trend of unlimited gain-bandwidth product, another signature of $k \sim 0$ for those InAs APDs.

4. InAs Avalanche Photodiodes with AlAsSb Blocking Layer

4.1 Design of InAs APDs with AlAsSb blocking layer

In order to achieve even lower dark current and high gain at low bias, our collaborators, Prof. Seth Bank's group, have designed and grown the epitaxial wafers for an InAs APD structure with a 6 μm-thick i-region thickness. The structure is shown in Fig. 4.1. An AlAsSb layer has been inserted to block electrons from diffusing into the depletion region and suppress the electron diffusion current. This significantly decreased the total dark current, which is dominated by diffusion current as confirmed in chapter 3.

One difficulty related to this structure is etching the AlAsSb layer. I tried several etching techniques on the wafer. The first recipe I tried is 1:1:1 H_3PO_4:H_2O_2:H_2O, which etches all the way down to the substrate. SEM image in Fig. 4.2 shows that the sidewall is seriously damaged. My second approach was to use 1:1:10 (H_2SO_4:H_2O_2:H_2O) to etch the AlAsSb layer, and 1:1:1 H_3PO_4:H_2O_2:H_2O to etch all InAs layers. Although the sidewall, looks better the surface leakage current remained unacceptable. A third approach used 1:1:9 HCl:H_2O_2:H_2O to etch the AlAsSb layer. The sidewall improved. The comparison of all the dark currents in Fig. 4.3 also confirms that the HCl acid etching is optimal.

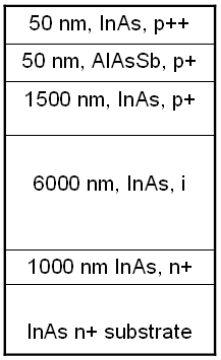


Figure 4.1. Schematic structure of the InAs APD with 6 μm-thick i-region.

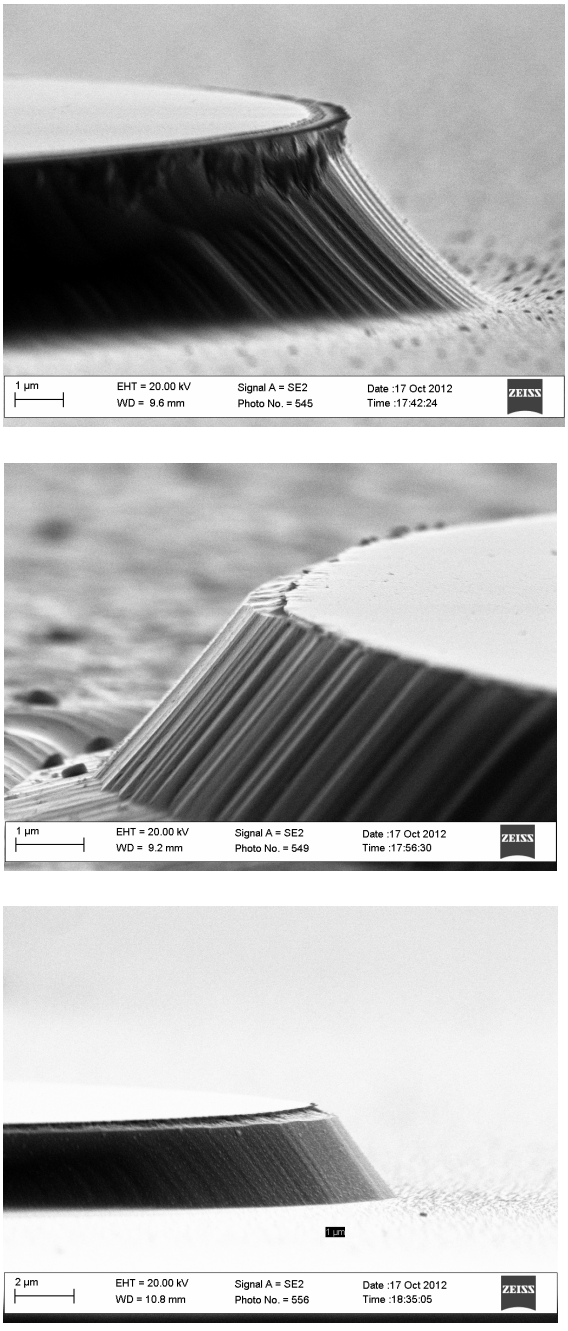


Figure 4.2. SEM results of the sidewall for all three etching recipes, top: 1:1:1 H_3PO_4:H_2O_2:H_2O, middle: 1:1:10 (H_2SO_4:H_2O_2:H_2O), bottom: 1:1:9 HCl:H_2O_2:H_2O.

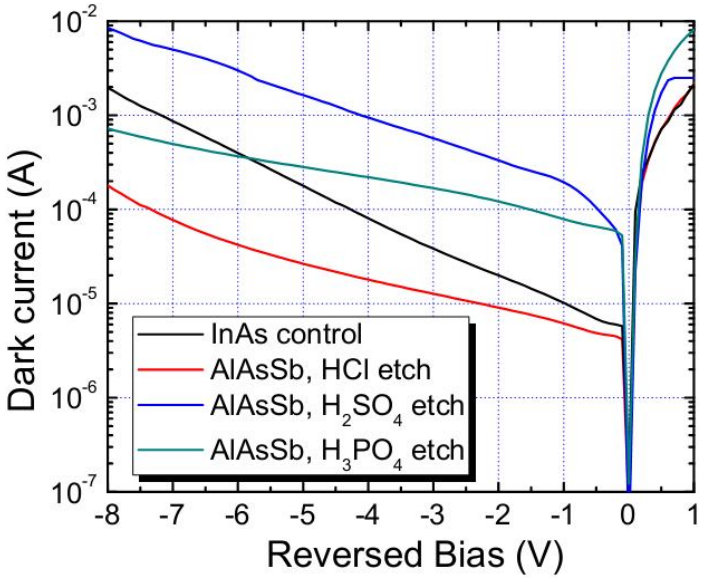


Figure 4.3. Comparison of dark currents for all etching recipes.

Mesa structure devices with diameter from 50 µm to 500 µm were defined by wet etching using 1:1:1 H_3PO_4:H_2O_2:H_2O, followed by 30 seconds etching in 1:8:80 (H_2SO_4:H_2O_2:H_2O). This recipe has been shown to suppress surface leakage current [55, 46]. The etched mesa sidewalls were covered with SU-8 passivation to minimize surface degradation. The measured dark current on a 100 μm-diameter device is shown in Fig. 4.4. We have also measured the dark current on a control structure without the blocking layer. The dark current with the blocking layer is two times lower than without the blocking layer. The gain measurement at room temperature is shown in Fig. 4.5a. Gain as high as 200 were achieved at only 13 V reverse bias, an indication of full depletion of the i-region. As shown in Fig. 4.5b, the calculated depletion width using capacitance measurement also confirmed this.

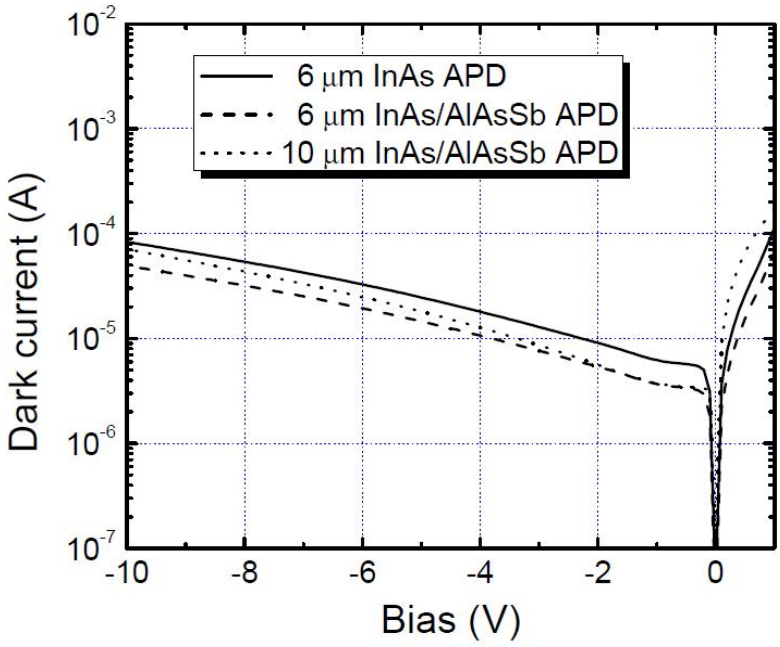


Figure 4.4. Measured room temperature dark current versus bias.

In order to further increase the gain at low bias, we fabricated another structure with a 10 μm-thick un-doped region. It also contained the AlAsSb blocking layer. As shown in Fig. 4.4, the dark current was only slightly higher than the 6-μm structure with the AlAsSb blocking layer; however, the thicker multiplication region achieved significantly higher gain. Figure 4.5b shows that the depletion width was as thick as 8 μm, which is likely limited by unintentional surface segregation of silicon during the molecular beam epitaxial growth. Low dark current is essential in order to operate at high reverse bias. Figure 4.6 shows the gain versus bias for a 20 μm-diameter device having the thicker depletion thickness. At 15 V bias the gain was 300 at room temperature. This is ~2× higher than previous reports in [57]. Noise measurement was also performed on the InAs/AlAsSb APD with 10 μm-thick un-doped region. As shown in Fig. 4.7, the excess noise factor F(M) is < 2 for gain up to 60, a signature of k ~ 0.

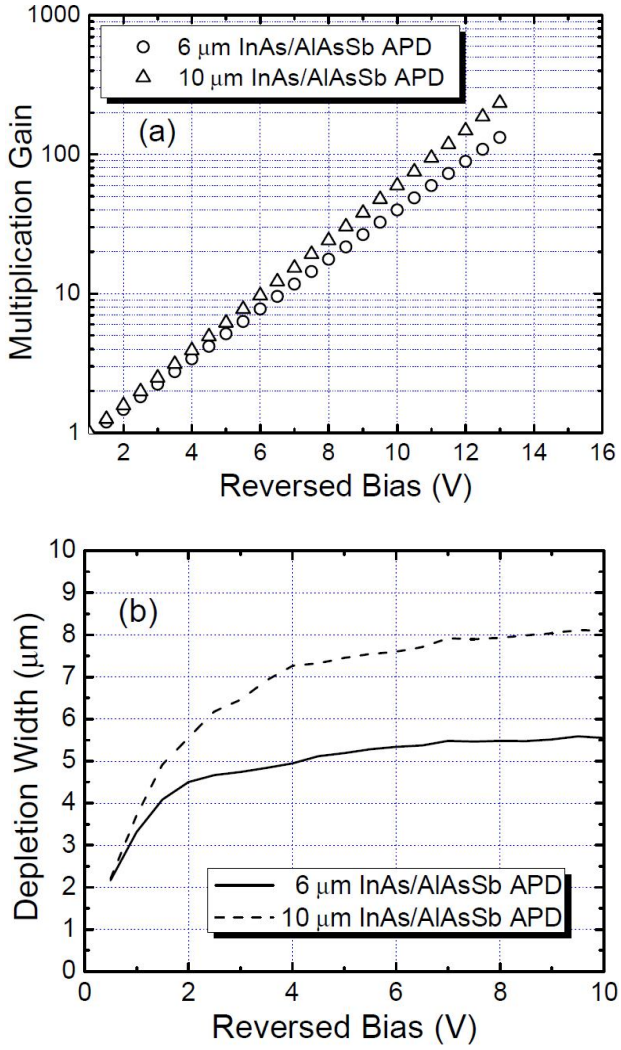


Figure 4.5. Measured (a) room temperature gain and (b) depletion region width of 6 μm-thick and 10 μm-thick i-region APDs.

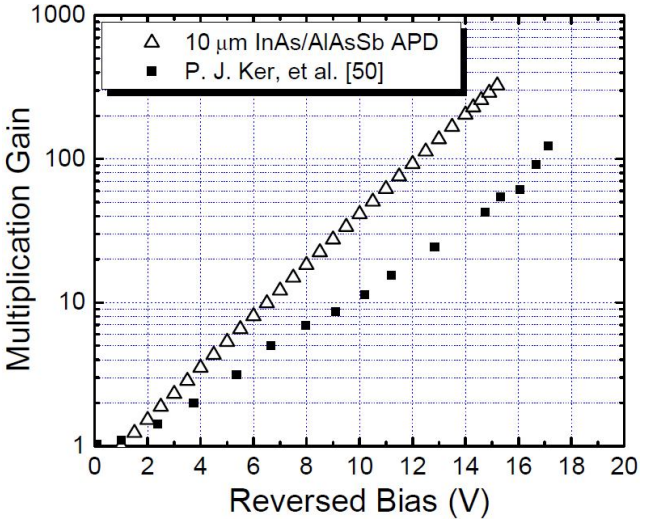


Figure 4.6. Measured multiplication gain for a 20 μm-diameter device for the 10 μm-thick i-region APD.

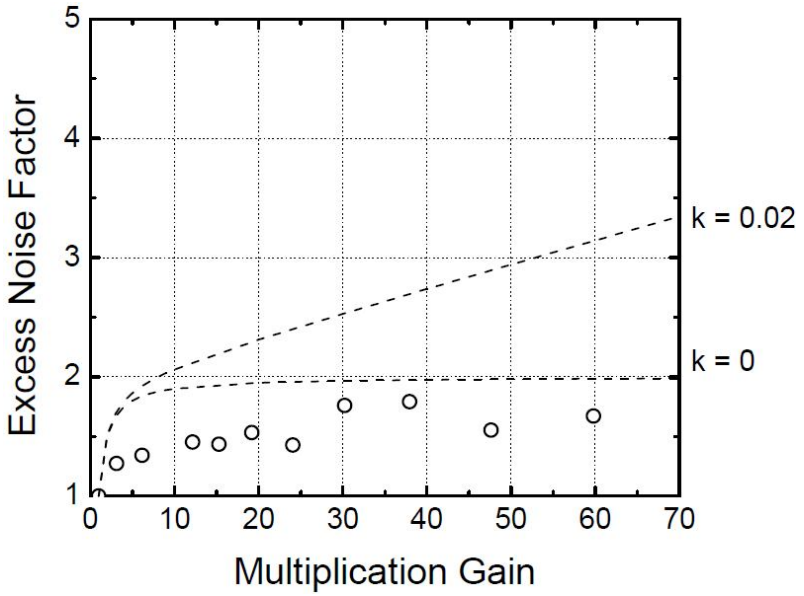


Figure 4.7. Noise measurement on the 10 μm-thick i-region InAs/AlAsSb APD.

4.2 Chapter summary

In this chapter, we have designed two InAs APD structures to further suppress dark current and enhance multiplication gain. Etching techniques have been tested for InAs APDs with an AlAsSb blocking layer. Measurements show that the dark current decreased almost by half owning to suppressed diffusion current. The multiplication gain is also enhanced by employing a 10 μm i-region. Gain as high as 300 was observed at only 15 V, which is significantly higher than previously reported. Excess noise measurement shows that k ~ 0 at gain as high as 60.

5 Study of Excess Noise Factor under Non-Local Effect in APDs

5.1 Introduction

The local-field model indicates that the excess noise factor can be expressed in terms of $<M>$ and the ratio of the electron and hole ionization coefficients, k, as $F(M) = <M^2>/<M>^2 = k \bullet <M> + (1-k)(2-1/<M>)$. This formula has been widely used and verified in noise measurements of APDs with thick multiplication regions. However, for APDs with a thin multiplication layer, it fails to describe the relationship between F(M) and M as a result of non-local effects. In these APDs, the electric field has to be higher than that for a thick layer in order to achieve the same gain, and the ratio of the electron ionization coefficient to that of the hole tends to unity as the electric field increases. Thus, one would expect the excess noise characteristics of APDs with thin multiplication layers to mimic materials with high k values. However, as mentioned in chapter 1, the opposite case is observed. It has been shown for a wide range of materials including InP, GaAs, InAlAs, AlGaAs, and SiC that the excess noise factor for thinner multiplication regions corresponds to lower effective k value. This is caused by the non-local effect. Note that the actual ratio of ionization coefficients is close to unity but the excess noise factor is that which would be obtained for materials with low k values when the local-field model applies. In this case, k is simply an indirect figure of merit. Carriers that initiate an impact ionization event have to travel a certain distance before they can gain sufficient energy from the electric filed to initiate another impact ionization, the dead space. When the dead space is an appreciable fraction of the multiplication thickness, it is less likely for a carrier to initiate gain in excess of the average gain. The result is lower statistical variance in the impact ionization process

and thus lower excess noise. The non-local effect not only suppresses the excess noise, but also changes the relationship between excess noise factor and multiplication gain. In the local-field model *F(M)* follows a non-linear relationship with M. However, it tends toward linear at very high gain. When the non-local effect applies, i.e., the APD utilizes a thin multiplication region, it has been observed that *F(M)* is linear in *M* for a wide range of materials [15-19]. The same phenomenon also exists with I^2E APDs [20-22], which use appropriately designed heterojunction structures to tailor impact ionization. The way that the non-local effect affects excess noise behavior has been studied in [58-61]. J. S. Ng, et al. have used a random path length (RPL) model to study how the dead-space effect modifies the multiplication probability distribution function (pdf) [61], and it was found that dead space tends to narrow the distribution when dead space becomes significant. In this chapter 1 present a study of the shape of the multiplication pdf that yields a linear relationship between *F(M)* and *M*.

5.2 Mathematics and Monte Carlo Simulation

When non-local effects are significant, it has been observed that F(M) is given by the relation:

$$F(M) = k_{eff}(M-1)+1 = k_{eff}M + (1-k_{eff}). \tag{5.1}$$

Notice that the k_{eff} is not the same as in the local-field model; it is defined as the slope of *F(M)* versus *M*. However the primary definition of *F(M)* remains

$$F(M) = \frac{<M^2>}{<M>^2}. \tag{5.2}$$

Therefore, if we know the probability density function of *M*, then $<M>$, $<M^2>$, and *F(M)*

can be calculated. Assume x is the total number of impact ionization events initiated by one carrier, and $f(x)$ is the pdf of x. Then $M = 1+x$. It follows that:

$$\begin{aligned} &< M >\approx \int_0^\infty (1+x)\, f(x)dx = \int_0^\infty f(x)dx + \int_0^\infty xf(x)dx = 1 + \int_0^\infty xf(x)dx \\ &< M >^2 \approx (1 + \int_0^\infty xf(x)dx)^2 = 1 + 2\int_0^\infty xf(x)dx + (\int_0^\infty xf(x)dx)^2 \end{aligned} \tag{5.3}$$

$$\begin{aligned} < M^2 > &\approx \int_0^\infty (1+x)^2 f(x)dx = \int_0^\infty f(x)dx + 2\int_0^\infty xf(x)dx + \int_0^\infty x^2 f(x)dx \\ &= 1 + 2\int_0^\infty xf(x)dx + \int_0^\infty x^2 f(x)dx \end{aligned} . \tag{5.4}$$

Now, according to the linear relationship of $F(M)$ and M, we have:

$$\frac{< M^2 >}{< M >^2} = 1 + k(< M > -1) . \tag{5.5}$$

and

$$\frac{1 + 2\int_0^\infty xf(x)dx + \int_0^\infty x^2 f(x)dx}{1 + 2\int_0^\infty xf(x)dx + (\int_0^\infty xf(x)dx)^2} = 1 + k\int_0^\infty xf(x)dx . \tag{5.6}$$

One solution to this integral-equation of $f(x)$ is a log-normal distribution, i. e.:

$$\begin{aligned} &f(x,\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}, x > 0 \\ &< M >= 1 + \int_0^\infty xf(x,\mu,\sigma)dx = 1 + e^{\mu + \frac{\sigma^2}{2}} \end{aligned} \tag{5.7}$$

In order to confirm that the pdf of impact ionization events follows a log-normal distribution, a Monte Carlo simulation can be used to calculate the number of impact ionization events for each injected carrier. By repeating the calculation for a large number

of injected carriers the pdf, *f(x)*, can be extracted from the simulation results.

The Monte Carlo tool that was used in this work is described in chapter 1. The model uses a simplified non-parabolic band structure that includes the Γ, L, and X valleys in the conduction band, and heavy-hole, light-hole, and split-off valence bands. For carrier scattering, we include impurity scattering, acoustic and optical phonon scattering, and more importantly, impact ionization scattering. Figure 5.1 shows the calculated excess noise factor versus gain, M, for InP, $In_{0.52}Ga_{0.48}As$, $Al_{0.2}Ga_{0.8}As$, and GaAs p-i-n structure APDs with multiplication thickness of 280 nm, 190 nm, 90 nm, and 100 nm, respectively. The calculations agree well with measurements reported in [62-64]. To simulate the distribution of impact ionization events, we approach *f(x)* as N(x)/N, where N(x) is the number of carriers that initiate x impact ionization events and N is total number of carriers simulated. In this work N is 10,000. Figure 5.2 shows the calculated distribution for the four structures for $<M>$ = 10 and 20.

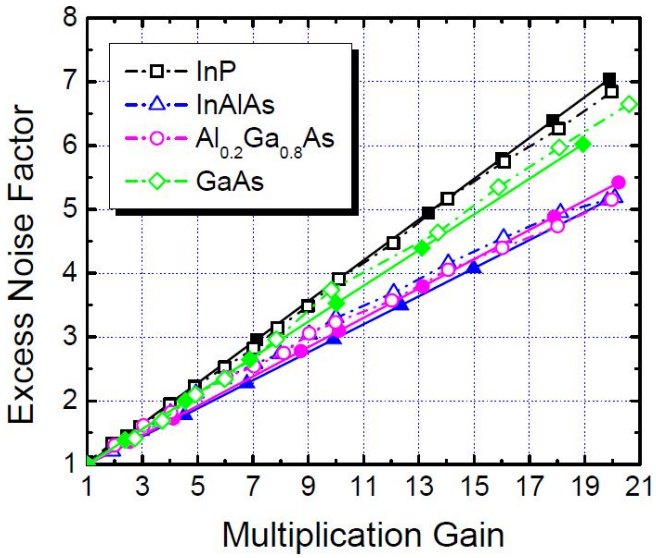


Figure 5.1. Monte Carlo simulation of excess noise (dash line) for all four p-i-n APD structures, compared to measured data (solid line).

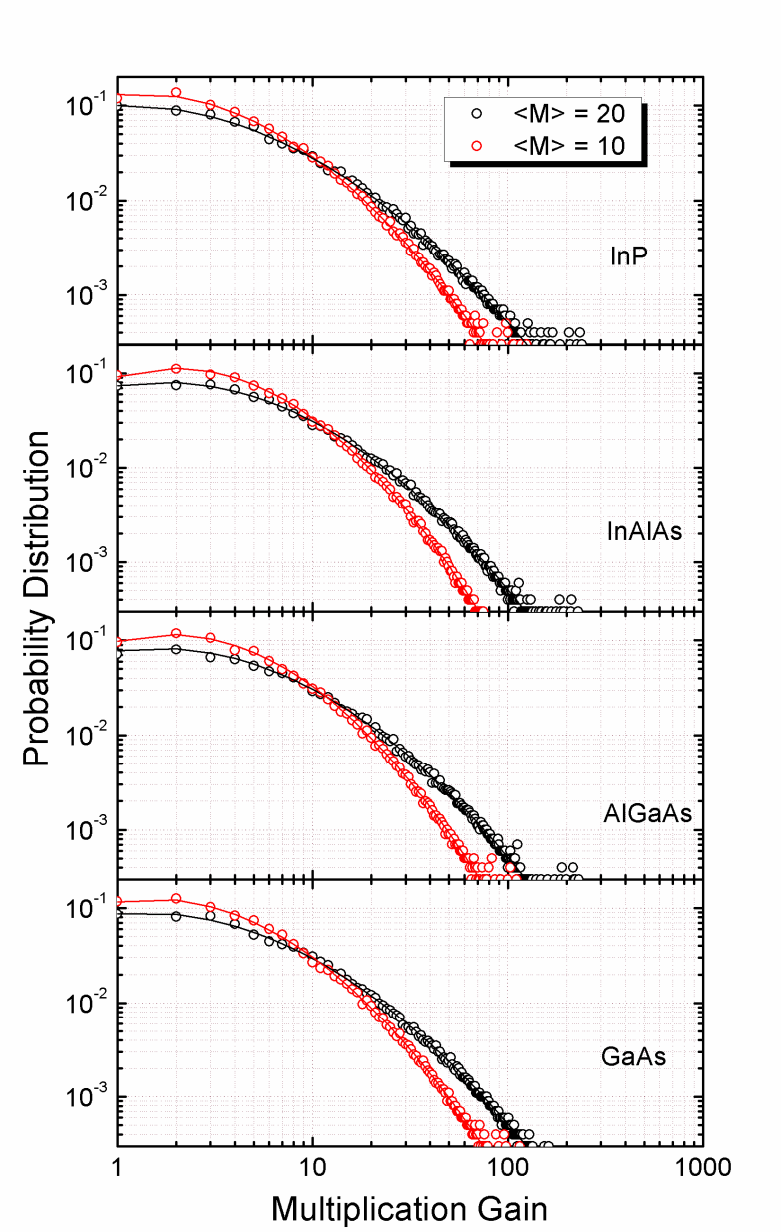


Figure 5.2. Simulated probability distribution function (dots) of multiplication gain for all four p-i-n APD structures for <*M*> = 10, 20. Lines represent the fitting curve using a log-normal distribution.

For all four structures, the simulated probability distribution function fits quite well with a log-normal distribution, therefore integral equation 5.6 is satisfied and the linear relationship between F(M) and M is elucidated.

Non-local effects are also important in I^2E APD structures. Figure 5.3 shows the

simulated and measured excess noise factor versus gain for an InAlAs/InAlGaAs I^2E APD structure, which has a multiplication region that incorporates an InAlAs/InAlGaAs heterojunction [65]. Again, the simulation agrees well with measurement and both exhibit a linear relationship between F(M) and M. The simulated pdfs for this structure at gains of 10 and 20 are plotted in Fig. 5.4. The simulated distribution again follows the log-normal distribution.

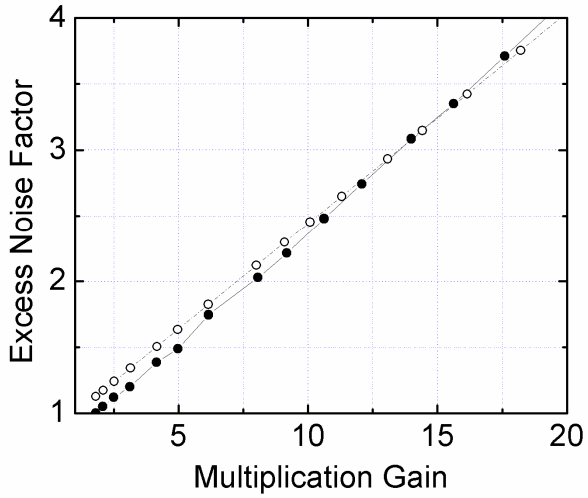


Figure 5.3. Simulated (open symbols) and measured (closed symbols) excess noise factor F(M) versus M for our InAlAs/InAlGaAs I^2E APD.

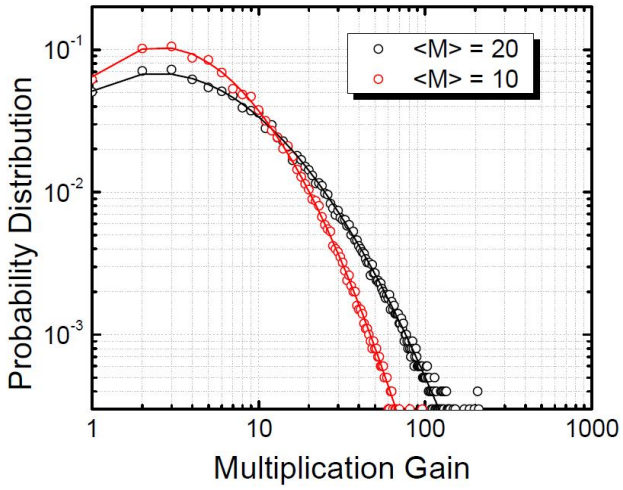


Figure 5.4. Simulated probability distribution function (dots) of multiplication gain for our InAlAs/InAlGaAs I^2E APD at gain = 10, 20. Lines represent the fitting curve using log-normal distribution.

5.3 Chapter Summary

This chapter presents a theoretical study of the linear relationship between gain and excess noise factor when the non-local effect is significant. The gain-noise relationship is different from the classic non-linear relationship given by the local-field model. It was mathematically derived that if the impact ionization distribution follows a log-normal distribution, the linear relationship will hold. Monte Carlo simulation was performed on APDs with different materials and structures, and good fitting is achieved using log-normal distribution for the simulated statistical distribution of impact ionization events.

6. Non-Linear Effect in Avalanche Photodiode under High Power Condition

6.1 Introduction

As stated in chapter 1, an important figure-of-merit for APDs is the gain-bandwidth product. The avalanche process in an APD requires a certain amount of time to build up or decay. The higher the gain, the longer the buildup time, which gives rise to the gain-bandwidth product. Typically, APDs are employed at low signal levels. At high input power levels, however, APDs have not been analyzed as thoroughly. Gyungock Kim, et al. operated the InGaAs/InAlAs APD using high illumination levels with photocurrent in the range of 0.1 ~ 1 mA. Enhanced frequency response and bandwidth were observed [66]. Bandwidth enhancement was also observed with Si/SiGe and Ge/Si APDs close to breakdown for photocurrents in the range of 1~10 mA [67-70]. Daoxin Dai, et al. used carrier transport equations and small signal analysis to derive an expression for the impedance of APD and concluded that an inductance should be included to complete the circuit model. The enhanced frequency response was described in terms of an LC resonance. This circuit model approach provides good fits to the measured impedances and bandwidths [71] but does not yield insight into the physical carrier transport mechanisms that can be obtained with Monte Carlo simulations. In this chapter, a simulation tool based on a self-consistent Monte Carlo technique is developed to calculate the frequency responses of InGaAs/InAlAs SACM APDs at high photocurrent levels.

6.2 Simulation Method

The model of carrier transport used in this work is similar to the simple model

described in chapter 1. The impact ionization parameters of InGaAs and InAlAs materials are the same as those in [72], which were determined by fitting the measured gain of p-i-n structure avalanche photodiodes with different i-region thicknesses. This model has demonstrated good agreement between simulations and measurements of the gain and bandwidth of InGaAs/InAlAs APDs [73].

The typical circuit model of a photodiode is shown in Fig. 6.1, where $I_{diode}(t)$ is the photo-generated conduction current, C is the junction capacitance, R_S is the series resistance, and R_L is the load impedance. When photo-current is high enough, the voltage drop on the external load is superimposed on the photodiode and can not be ignored. This can induce a temporal bias variation that gives rise to a displacement current.

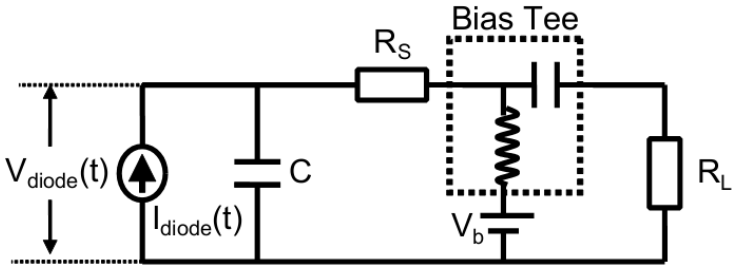


Figure 6.1. Equivalent circuit model of photodiode.

The photodiode bias can be written as:

$$V_{diode}(t) = -V_b + V_s + V(t) \tag{6.1}$$

where $-V_b$ is the DC reversed bias, V_S is the DC voltage drop on R_S, and V(t) is the AC component of photodiode bias, which is also the AC voltage drop on the branch of R_S and R_L. Therefore the AC current through the load is:

$$I_{load}(t) = \frac{V(t)}{R_S + R_L} \quad . \tag{6.2}$$

The displacement current, which is the variation of the charge on the junction capacitance C, is

$$I_D(t) = C\frac{dV_{diode}(t)}{dt} = C\frac{dV(t)}{dt} \quad . \tag{6.3}$$

Here we assume the junction capacitance C is constant, since for an SACM APD the depletion region changes very little with bias once it is fully depleted. According to the continuity equation for current, the AC element of the conduction current from the photodiode is:

$$I_C^{AC}(t) = I_D(t) + I_{load}(t) = C\frac{dV(t)}{dt} + \frac{V(t)}{R_S + R_L} \quad . \tag{6.4}$$

Taking the derivative of equation 6.4, yields:

$$\begin{aligned} \frac{dI_C^{AC}(t)}{dt} &= C\frac{d^2V(t)}{dt^2} + \frac{1}{R_S + R_L}\frac{dV(t)}{dt} \\ &= \left[\frac{i2\pi f(R_S + R_L)C + 1}{R_S + R_L}\right]\frac{dV(t)}{dt} \end{aligned} \tag{6.5}$$

where f is the modulation frequency. Finally, we obtain

$$\frac{dV_{diode}(t)}{dt} = \frac{R}{i2\pi fRC + 1}\frac{dI_{diode}(t)}{dt} \tag{6.6}$$

using the following expressions:

$$\begin{aligned} &\frac{dI_C^{AC}(t)}{dt} = \frac{d(I_C^{AC}(t) + I_C^{DC})}{dt} = \frac{dI_{diode}(t)}{dt} \\ &\frac{dV(t)}{dt} = \frac{d(-V_b + Vs + V(t))}{dt} = \frac{dV_{diode}(t)}{dt} \\ &R = R_S + R_L. \end{aligned} \tag{6.7}$$

Equation 6.6 indicates that the photodiode bias is directly related to the conduction

current, and the conduction current can be calculated using Monte Carlo simulation of carrier transport. It is also noted that the photodiode bias is not in phase with the conduction current due to the junction capacitance, and this phase delay between V_{diode} and I_{diode} varies with frequency.

The time-varying photodiode bias will significantly affect the impact ionization process owing to the resulting variation of the electric field. Hence, in order to accurately simulate APD performance under high power conditions, where both current and voltage are modulated, an accurate description of impact ionization process is needed. Daoxin Dai, et al. used the local-field model to include the multiplication process [74], which assumes ionization coefficients are only functions of the local value of the electric field. The local-field model works well with relatively thick gain layers, however, thin multiplication regions are usually preferred for high-speed and low-noise performance of APDs. In thin multiplication regions non-local effects become significant, which means the ionization process depends on the history of carrier motion [75]. This is especially important when the electric field is modulated. Since the Monte Carlo technique directly simulates carrier drift and scattering processes on the microscopic level, the non-local effect is automatically included. Under high optical illumination, the significant space charge density will redistribute the electric field in the APD. To account for this, the electric field profile is updated by solving Poisson's equation at each time step to make the simulation self-consistent. The mesh size and time step are set as 1 nm and 10 fs, respectively. The flow chart of the simulation is shown in Fig. 6.2. The simulation begins with the time-varying injection rate of photo-generated carriers at a certain modulation frequency. The conduction current and photodiode bias are iteratively calculated with an open loop time-step Monte Carlo calculation, which stops once the current stabilizes.

Then the RF signal power delivered to the load is calculated using Fourier transform and the simulation starts again at the next frequency. The final result is a plot of the frequency response.

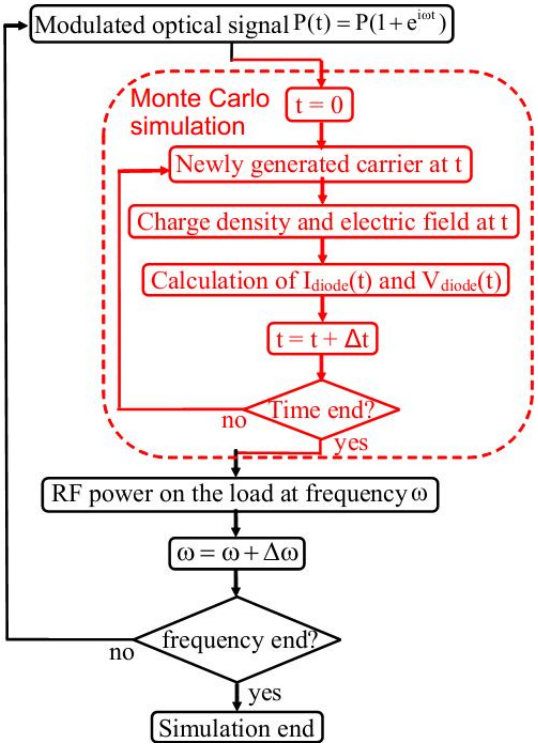


Figure 6.2. Algorithm flow chart of simulation.

6.3 Simulation Results

The model was tested by simulating a typical InGaAs/InAlAs SACM APD structure. A schematic cross section is shown in Fig. 6.3. Starting from an n^+ InP substrate, the growth sequence consists of a 200 nm unintentionally-doped InAlAs multiplication layer, a 270 nm InAlAs charge layer, a 200 nm unintentionally-doped InGaAs absorption layer, and 50 nm highly p-doped InGaAs that serves as the contact layer. Figure 6.4 shows the simulated DC gain of a 50-μm-diameter device at different optical power levels. As expected, the multiplication gain decreases with increasing optical power. This is the well-known gain saturation effect. To simulate the frequency response, we assume the

series resistance, R_S = 20 Ω, which is reasonable with a good ohmic contact on the InGaAs contact layer. The junction capacitance C_j is considered as a simple parallel plate capacitor and the calculated value is ~ 0.62 pF for a 50-μm-diameter device. An example of simulated APD current and bias is shown in Fig. 6.6, where the bias is 30 V. The modulation frequencies are 1 GHz and 6 GHz and the optical power is 0.3 mW. Both the APD current and bias are modulated, and there is a phase delay between them similar to that reported in [66-68], the time-varying APD bias would directly affect the impact ionization process so that the instantaneous gain also changes with time.

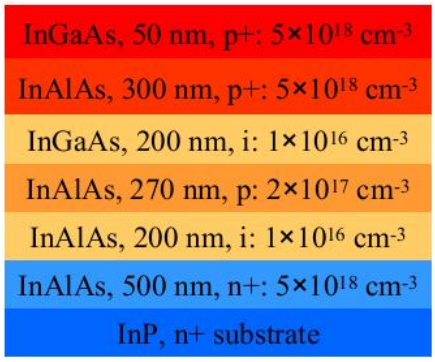


Figure 6.3. Layer structure of the InGaAs/InAlAs APD.

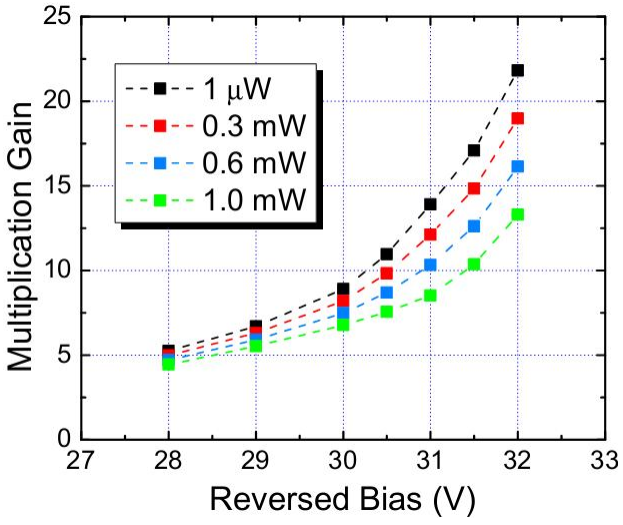


Figure 6.4. Simulated gain versus reversed bias at four different optical power levels.

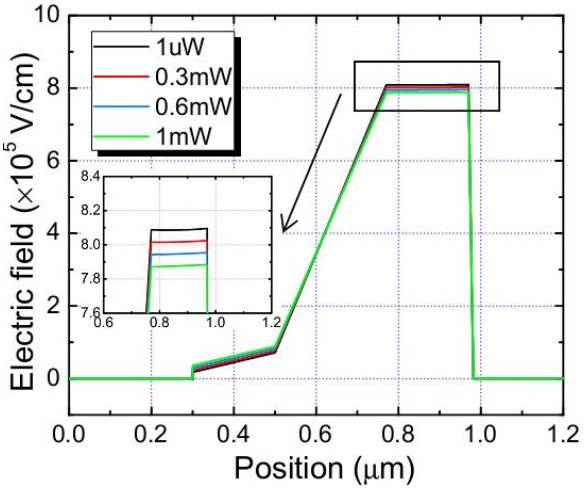


Figure 6.5. Simulated electric fields (reverse bias = 30 V) at four different optical power levels.

This influence will vary with modulation frequency and DC bias because the modulation frequency changes the phase delay and the DC bias changes the build-up time of the gain. At a certain frequency, the modulated gain is in phase with the modulated injection rate of carriers into multiplication region, which means that when the carrier injection rate is at its highest value, the instantaneous gain that the injected carriers experienced is also at its highest value. For this frequency a larger amplitude of AC current or "internal gain" is expected. Figure 6.6 illustrates that a much higher AC current is generated at 6 GHz than at 1 GHz modulation frequency. Figure 6.7 includes the simulated frequency responses at different optical power levels after calculating the power of the AC signals delivered to the load. With an input optical power of 1 μW, a typical APD frequency response is observed and the 3dB bandwidth is approximately 10 GHz; with higher optical power the frequency response exhibits a peak. This "RF enhancement" phenomenon has the effect of increasing the bandwidth well beyond 10 GHz. For 1 mW incident power, the bandwidth more than doubles, approaching 25 GHz. The enhanced bandwidth clearly originates from the voltage swing effect due to the feedback of the load, as there is no such effect when the incident power is 1 μW for which the voltage swing across the load is negligible.

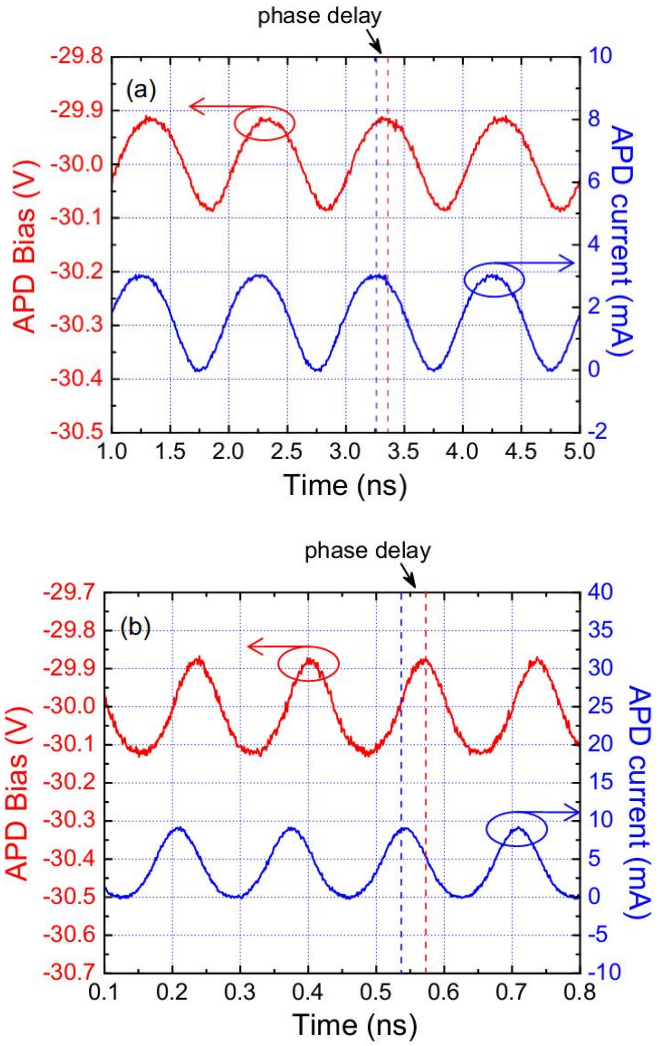


Figure 6.6. Simulated APD current and bias versus time at DC bias = 30 V, optical power = 0.3 mW, (a) modulation frequency = 1 GHz. (b) modulation frequency = 6 GHz.

This enhancement phenomenon is similar to the effect of an internal inductance in the circuit model of APDs, which in conjunction with the capacitance results in an LC resonance that causes the phase delay and the peak in the frequency response. Using the circuit model with internal inductance shown in Fig. 6.8 and equation 6.8, where I_{SC} represents the APD short-circuit current and L_i and R_i are the internal inductance and resistance of APD, respectively, we have extracted the inductance values that give the best fitting to the simulated frequency responses. As shown in Fig. 6.9, the inductance decreases with DC reverse bias, and also with optical power level. This is consistent with

the work in [66-68], where it was found that the inductance is inversely proportional to the injected current, which increases with bias and optical power.

$$I_L = I_{SC}\frac{Z}{Z+(R_S+R_L)}$$

$$\frac{1}{Z} = \frac{1}{R_i + j\omega L_i} + \frac{1}{1/j\omega C} \tag{6.8}$$

The calculated gain-bandwidth product is plotted in Fig. 6.10. Although the multiplication gain decreases somewhat due to the space-charge effect at high power levels, the bandwidth is significantly enhanced. Therefore, compared to low optical power, the gain-bandwidth product is much higher and reaches almost 300 GHz.

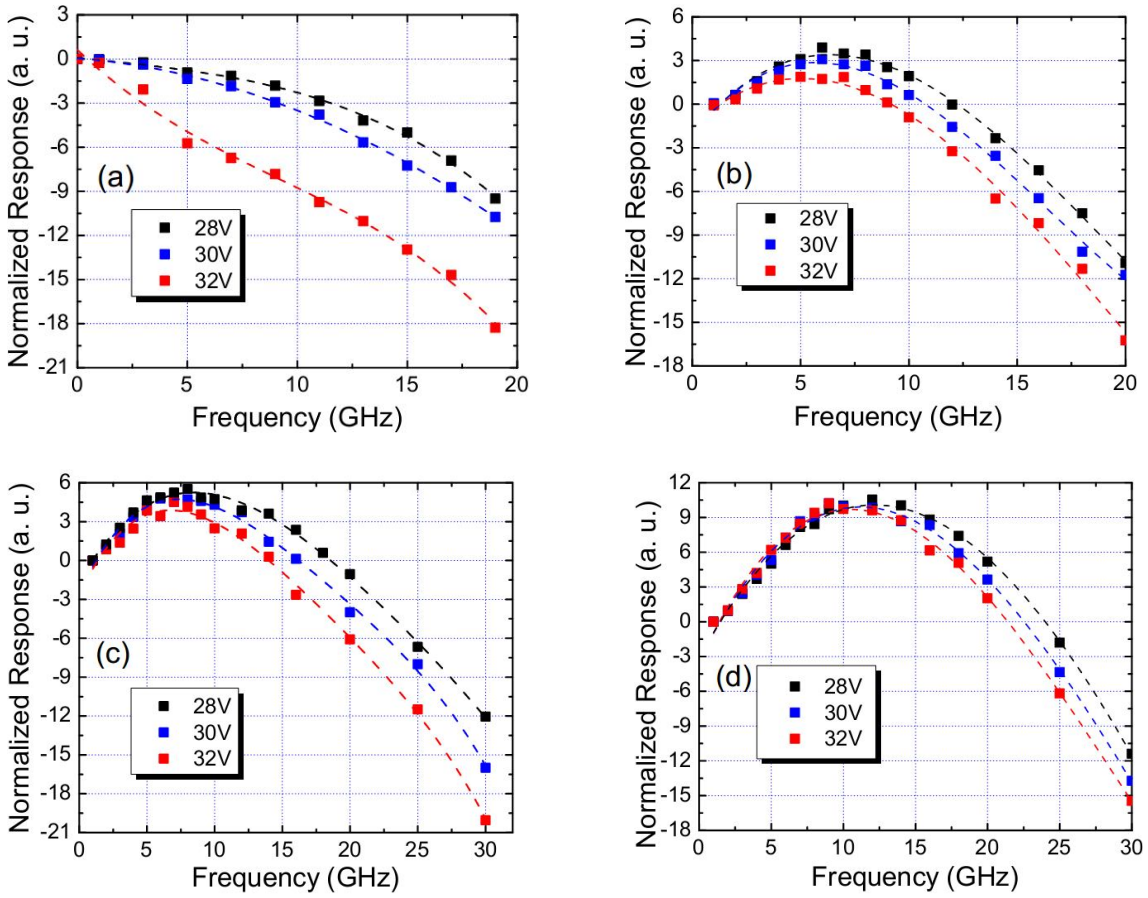


Figure 6.7. Simulated frequency responses at reversed bias = 28 V, 30 V, and 32 V with (a) optical power = 1μW, (b) optical power = 0.3 mW, (c) optical power = 0.6 mW, (d) optical power = 1 mW.

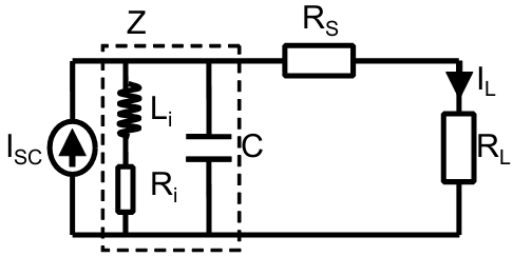


Figure 6.8. APD circuit model with internal inductance included.

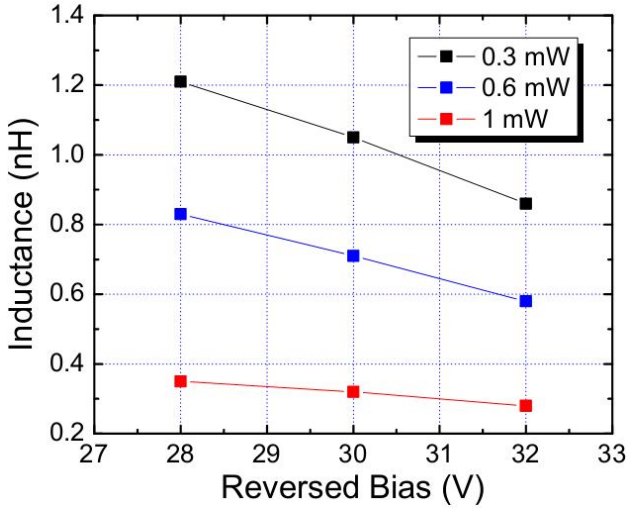


Fig. 6.9. Extracted inductances at different bias and optical power levels.

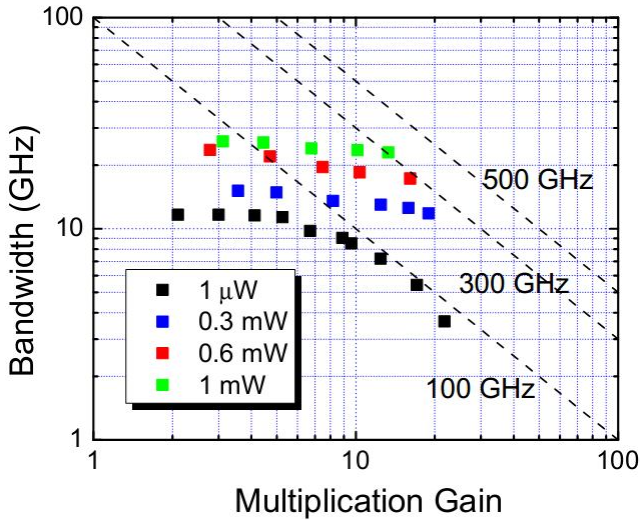


Fig. 6.10. Simulated bandwidth versus gain at different optical power levels.

6.4 Characterization of Non-Linear Effect

As discussed above, at high power levels modulation of the photocurrent and bias can increase the speed performance of APDs. However, this time-varying APD bias means that the instantaneous multiplication gain, as well as the responsivity will be modulated. It follows that the response will become highly nonlinear. In order to study this nonlinear effect caused by voltage swing at high power levels, we characterized the performance of an InGaAs/InAlAs SACM APD [76] as an RF mixer. Previously, APDs have been used as optoelectronic mixers by modulating the bias voltage [77-79]. The apparatus employed in this work is shown in Fig. 6.11. Two 1550-nm lasers with $LiNbO_3$ modulators were the optical sources for the intermediate-frequency (IF) and local oscillator (LO) signals. Large-signal modulation (100%) frequencies of 15 MHz and 40 MHz were used for the IF signal and the LO signal, respectively. The optical IF and LO signals were combined through a coupler followed by an EDFA. The optical signal was attenuated to the desired power level and coupled into the photodiode with a lensed fiber. The APD was biased through a bias tee. The IF, LO, and up-converted powers were measured with a spectrum analyzer. During the measurement it was observed that at different biases and power levels the highest up-converted RF signal was obtained when the LO optical power was 40 ~ 45% of total optical power, which is consistent with the report in [77] that the IF and LO optical power should be approximately equal to achieve the highest RF signal power for a given total optical power, Therefore the percentage of LO signal power was fixed at 45% for these measurements.

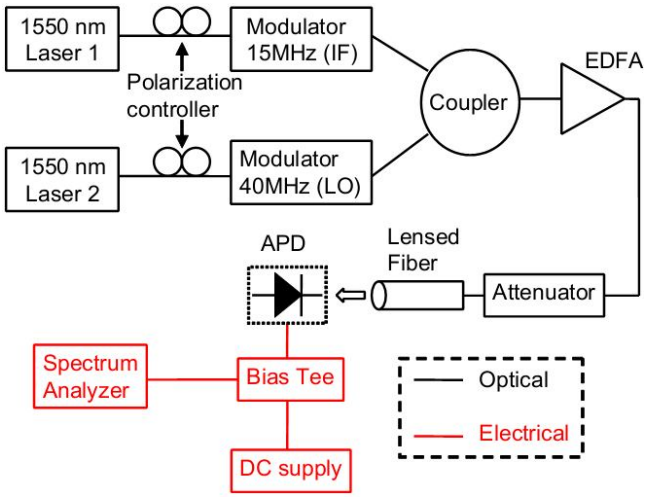


Figure 6.11. Block diagram of the setup for mixing in APDs.

The measured gain at different optical power levels is shown in Fig. 6.12. Consistent with the simulations in Fig. 6.4, significant gain saturation is observed. Figure 6.13 shows the measured electrical power of the LO signal at 40 MHz for different bias voltages and optical power levels. For low bias (45 V and 48 V), the LO electrical power increases linearly with optical power, however, at 51 V and 54 V, the LO electrical power eventually saturates at high power levels due to gain saturation. A measure of the nonlinear behavior of the APD is the conversion efficiency, P_{RF}/P_{LO}, the ratio of measured up-converted RF electrical power to LO electrical power in dB. Strong nonlinearity will manifest in higher conversion efficiency. Figure 6.14 shows the conversion efficiency versus the incident optical power. For a given optical power level, the conversion efficiency increases with bias because at higher bias the AC signal power is also higher due to higher multiplication gain (as shown in Fig. 6.13). The amplitude of the voltage swing is high enough to cause strong nonlinearity, and relatively high conversion efficiency is expected. Similarly, for fixed bias, the conversion efficiency increases with optical power, however, instead increasing linearly the conversion

efficiency saturates at high power levels; at 51 V and 54 V reverse bias the conversion efficiency saturates at ~ -10 dB. To better understand this, we performed a numerical simulation of the conversion efficiency similar to the model in [95]. Using this approach equation 6.4 is transformed into a discrete-time differential equation, equation 6.9. The

$$I_k = \frac{C(V_k) - C(V_{k-1})}{t_k - t_{k-1}} V_k + C(V_k) \frac{V(t_k) - V(t_{k-1})}{t_k - t_{k-1}} + \frac{V_k}{R_S + R_L}$$
$$k = 1,\ 2,\ 3\ ...,\ N. \tag{6.9}$$

$$I_k = R(V_k)(\sin(2\pi f_{IF} t_k) + \sin(2\pi f_{LO} t_k)) \tag{6.10}$$

current, I(t), is calculated using equation 6.10. Both I(t) and V(t) are solved iteratively with an open-loop time step calculation. The voltage dependent capacitance C(V) and reponsivity R(V) are obtained from measured data. After stabilization of I(t) and V(t), the power spectral density is calculated by performing a fast Fourier transform of the calculated I(t) and then the conversion efficiency is extracted. Using this numerical technique, the calculated conversion efficiency is shown in Fig. 6.14 (dashed lines). The results are consistent with measurements. At high reverse bias and high power level both show saturation of the conversion efficiency. Therefore, although the induced nonlinearity of APDs at high power levels may degrade the APD performance, the sidebands caused by the nonlinearity will eventually saturate. Therefore, APDs may be candidates for high-speed applications at high power levels.

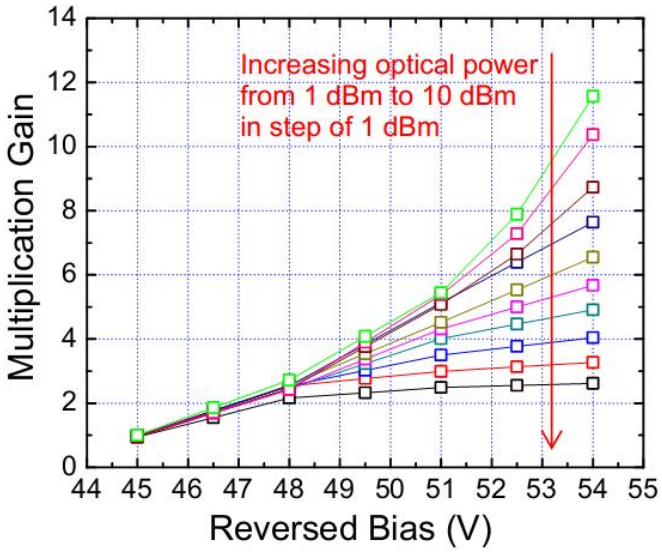


Figure 6.12. Measured multiplication gain vs. reversed bias at different optical power levels.

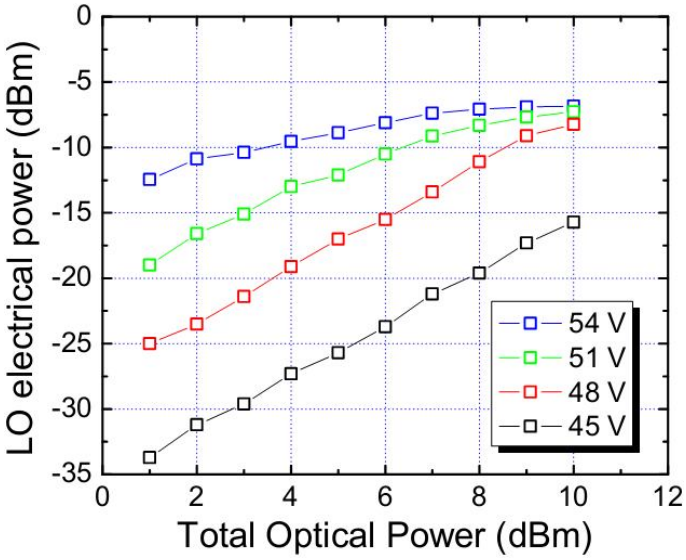


Figure 6.13. Measured LO electrical power at different biases and optical power levels.

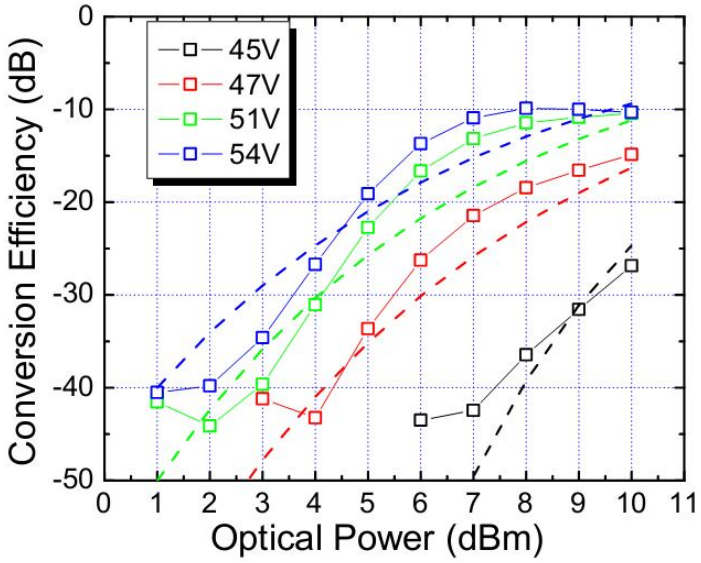


Figure 6.14. Measured (dot-solid line) and simulated (dash line) conversion efficiencies at different biases and optical power levels.

6.5 Chapter summary

This chapter presents a theoretical study of the nonlinear behavior of APDs at high-power condition. Monte Carlo simulation is used to explain the saturation of gain, and more importantly, the bandwidth enhancement effect. It was found that for high input power, both the bias voltage and the multiplication gain of APDs are modulated. The phase delay between modulated gain and modulated input carrier number plays an important role in the bandwidth enhancement. Under high-power operation, significant non-linear properties suggest APDs as possible mixers. Using a two-tone setup, the conversion efficiencies of our InAlAs APD are characterized.

7. Broadband Noise Limit in the Photodetection of Ultra-low Jitter Optical Pulses

7.1 Introduction

Generating and distributing low-phase-noise microwave signals is important for a number of scientific applications such as microwave atomic clocks [95], synchronization at large-scale facilities [96], and radar systems [97]. Several photonic systems, including optical delay-line oscillators, whispering-gallery-mode parametric oscillators and dual-mode lasers have been investigated for the generation of low-noise microwave signals. A photonics approach to low-noise microwave generation developed in NIST has enabled 10 GHz signals with close-to-carrier absolute phase noise less than -100 dBc/Hz, which is 40 dB lower than the best room-temperature electronic oscillators. In this technique, the microwave signals to date have been generated via high-speed photodetection of an ultralow-jitter optical pulse train, as illustrated in Fig. 7.1. Here an optical frequency comb produced by a femtosecond mode-locked laser is locked to an ultrastable optical frequency reference, transferring its fractional frequency stability to the pulse repetition rate [87]. The low-jitter optical pulse train illuminates a high-speed photodetector and is transformed into an electrical pulse train. The spectrum of such a pulse train approximates a series of Dirac-delta functions separated by the repetition rate. A band-pass filter centered at the desired frequency is used to select the microwave signal. In this manner, 10 GHz signals have been

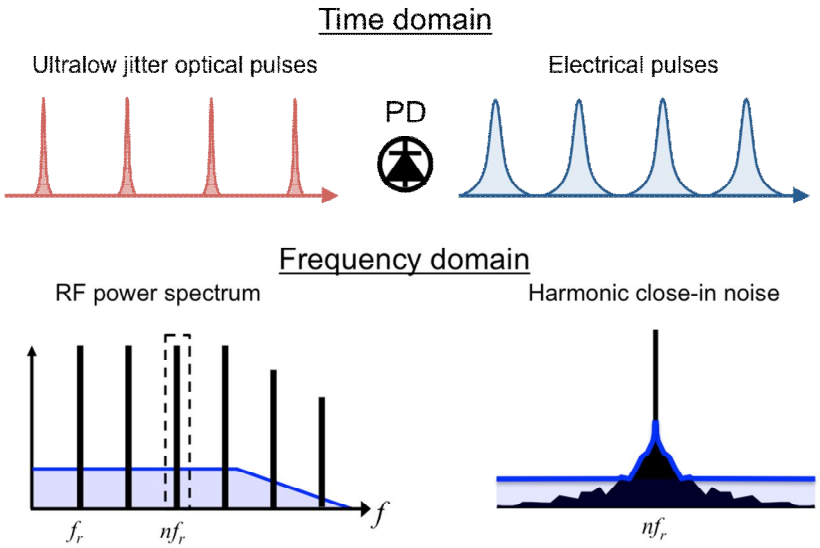


Figure. 7.1. Schematic diagram of photonic microwave generation via the detection of ultralow jitter optical pulses. In the lower right, the black noise sidebands represent the noise from the optical pulse train, and the blue line is the total noise after photodetection.

generated with phase noise less than -100 dBc/Hz at 1 Hz offset from the carrier [88], closely following the noise level expected from the optical reference. This is more than 40 dB below the best room-temperature microwave oscillators and comparable to the best cryogenic microwave oscillators. The corresponding 10 GHz timing jitter, obtained by extrapolating and integrating the phase noise spectrum to the Nyquist frequency (5 GHz), is < 3 fs for time scales up to 1 second.

The majority of the 10 GHz timing jitter can be attributed to the phase noise for offset frequencies 1 MHz and greater, and phase noise floors as low as -179 dBc/Hz have been demonstrated [89]. Previously, it was believed that this phase noise is contributed by the total shot noise in optical pulse trains, which is introduced by the randomness in the arrival time of photons. An analytical model developed by Quinlan, et al mathematically derived the percentage of total shot noise that goes to phase noise of the generated microwave signal [90]. However, The measured phase noise level is ~25 dB above what analysis indicates. Thus there remains a disparity between analysis and what has been

demonstrated in the phase noise far from carrier.

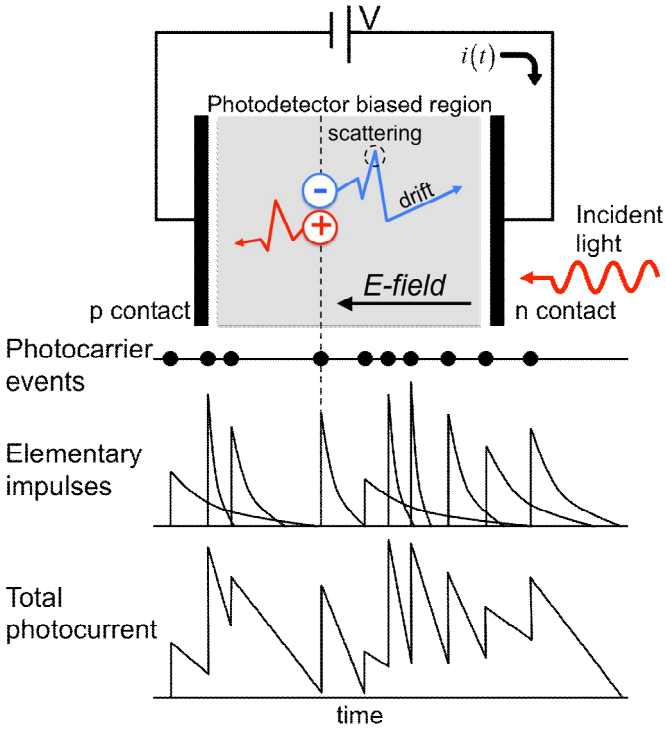


Figure. 7.2. Simplified schematic illustration of photo-carrier transport and resulting photocurrent from a reverse-biased photodiode. Both the randomness in the timing of the photoelectron events and the shape of the elementary impulses give rise to noise in the photocurrent.

To resolve the discrepancy between experiment and analysis, here we consider detection noise from sources not normally accounted for in photodiodes (PDs), namely photo-carrier scattering and distributed absorption of photons within the detector. A Monte Carlo simulation has been developed that indicates that the contribution to the phase noise of this excess noise can be orders of magnitude above that of the shot noise, and can easily limit the phase stability of microwaves derived from the lowest jitter pulse trains. This constitutes an important, previously unrecognized noise source inherent in the optical to microwave conversion of ultralow jitter pulse trains.

7.2 Simulation Method

Shot noise and photo-carrier transport. – Shot noise may be described as the

quantum noise arising from randomness in the photon arrival, described with Poisson statistics, that is transformed into fluctuations in the photocurrent during the photodetection process. As shown in Fig. 7.2, absorption of photons generates photo-carriers that, when subject to an electric field, create current impulses [91]. For detectors without gain, each impulse contains the charge of a single electron, and the sum of these impulses becomes the measured photocurrent. For average photocurrents above a few mA, thermal noise is smaller than shot noise, and the photocurrent noise spectral density can be represented as

$$S_i(f) = 2qI_{avg}\left|H(f)\right|^2 \quad \left(A^2/Hz\right) \tag{7.1}$$

where q is the fundamental charge, I_{avg} is the average photocurrent, and $H(f)$ is the transfer function associated with the impulse response of the photodiode. As illustrated in Fig. 7.2, PDs will additionally have randomness in the impulse response, since the exact shape and width of every current impulse will depend on where in the absorptive region the photo-carrier is generated, as well as on the random path taken by each photo-carrier. Equation (7.1) can be adapted to include these effects by allowing the impulse response to vary randomly, as is often used in detectors with gain. This yields

$$S_i(f) = 2qI_{avg}\left|\left\langle H(f)\right\rangle\right|^2 F_H \tag{7.2}$$

where the brackets indicate ensemble average, and F_H is the excess noise factor defined by

$$F_H = \left\langle\left|H(f)\right|^2\right\rangle \Big/ \left|\left\langle H(f)\right\rangle\right|^2 . \tag{7.3}$$

By definition $F_H \geq 1$. Whereas the excess noise contributes less to the total photocurrent than does the shot noise, the situation is drastically changed when considering the phase quadrature of microwaves generated by detecting ultrashort optical pulses. Here the shot noise contribution is dependent on the optical pulse width [90]. For a train of

Gaussian-shaped optical pulses, the shot noise contribution to the phase noise on the n^{th} harmonic of the repetition rate is given by

$$S_\phi(f) = \frac{2qI_{avg}\left|H(nf_r)\right|^2 R}{P_\mu(nf_r)}\left[1-\exp\left\{-\left(2\pi nf_r\tau_G\right)^2\right\}\right] \quad \left(rad^2/Hz\right) \tag{7.4}$$

where f_r is the pulse repetition rate, R is the system impedance, P_μ is the power in the n^{th} harmonic, and τ_G is related to the optical pulse full width at half maximum τ_p by $\tau_p = 2\sqrt{\ln(2)}\tau_G$. For a detector with unity quantum efficiency, this represents the quantum limit of the optical pulse timing jitter [90]. The term in brackets in Eq. 7.4 is the optical pulse width-dependent improvement in the phase noise, and represents the deviation in the phase noise as compared to Eq. 7.1. For a 10 GHz microwave carrier generated by the detection of 1 ps duration pulses, the predicted phase noise deviation is nearly -30 dB. With the shot noise contribution to the phase noise so low, the conclusion that the excess noise can be ignored in PDs without gain needs to be revisited. Assessing the impact of the excess noise depends on calculating the random path of the photo-carriers, making a quantitative analytical approach intractable. However, the problem is well suited to Monte Carlo simulation. As described below, a Monte Carlo simulation of a high-speed PD has been developed to study the impact of the excess noise, leading to good agreement with experimental results.

The Monte Carlo tool used in this work is similar to the model used in chapter 5. The simulated device is a modified uni-traveling carrier detector (MUTC) with 50 μm active area, the structure of which is described in [92]. Compared to standard p-i-n detectors, the MUTC structure improves the microwave saturation power while maintaining high speed and high linearity, since only high-drift-speed electrons are employed as active

carriers in the drift/collection region. The 3dB bandwidth is close to 20 GHz, limited by the RC time constant given by the device capacitance and the circuit's resistive load. The carrier transit time is nominally 10 ps at 18V and 15 mA average photocurrent, but will vary according to the applied bias voltage and photo-carrier density. This detector, along with a 2 GHz repetition rate optical pulse train with center wavelength of 1 μm, was used in order to best compare to experimental results in [89].

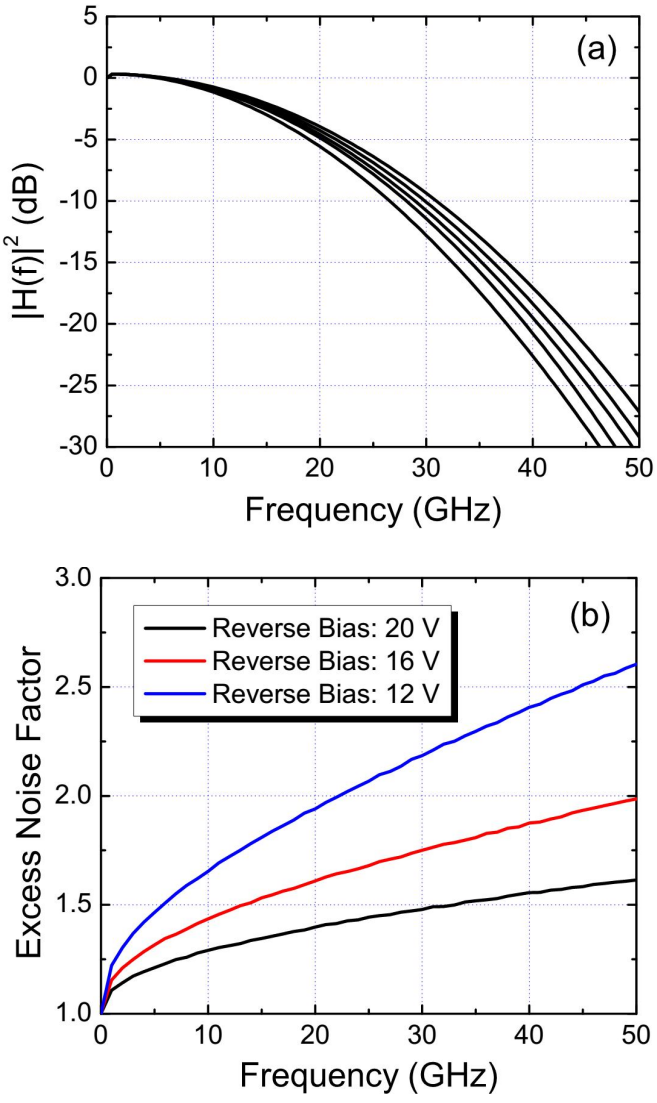


Figure 7.3. (a) $|H(f)|^2$ for a few representative current impulses for 16 V bias, I_{avg} of 15 mA, and pulse width of 15 ps. (b) Calculated F_H at average current 15 mA and pulse width of 15 ps.

The PD structure is divided into a one-dimensional grid. A bias voltage is established across the detector, and the electric field is calculated in each grid region using Poisson's equation. Carrier drift and scattering processes are then simulated in the structure. In each small time interval Δt the probability of photon detection is proportional to the optical intensity in that time interval. For each detected photon, the absorption depth in the photodiode is chosen randomly with probability determined by the absorption profile of the detector material. The photon-generated electrons and holes transit the undepleted and depleted regions by diffusion and drift, respectively. Carrier transport in the drift regions contribute to an impulse response. The electrical impulse responses are summed to form the total output photocurrent, i. e., the electrical pulse train.

Representative transfer functions of simulated current impulses are shown in Fig. 7.3(a). All the transfer functions have the same value at zero frequency since each impulse response has the same total charge. At frequencies of 10 GHz and above, variations in the transfer function become apparent, indicating variations in the photo-carrier transit time. Simulations of F_H are shown in Fig. 7.3(b) for different PD bias voltages and fixed 15 mA average current and 15 ps optical pulse width. The increase of F_H with frequency is expected from the frequency dependent variation in the transfer function. For large photocurrents, the number of photo-carriers is large enough that the electric field they generate is a significant fraction of the applied bias field, known as the space charge effect [93]. This causes changes in the electric field experienced by the photo-carriers, impacting their transit time. The F_H dependence on bias voltage is due to the fact that, for lower bias voltages, space charge effects are more significant, leading to a larger variation in the photo-carrier transit time. F_H is unity at zero frequency, as it must be when each impulse has the same total charge. This is the distinguishing feature between the excess noise factor

of PDs with and without internal gain.

Near 10 GHz, excess noise factor dependence on average photocurrent and optical pulse width were also explored, shown in Fig. 7.4. As mentioned above, the space charge effect is less significant at lower average photocurrent, therefore excess noise factor also tends to be lower. The optical pulse width dependence may be understood as a consequence of dynamic changes in the space charge. The narrower the optical pulse, the more likely all photo-carriers experience the same space charge field, leading to a smaller variation in transit time and lower excess noise.

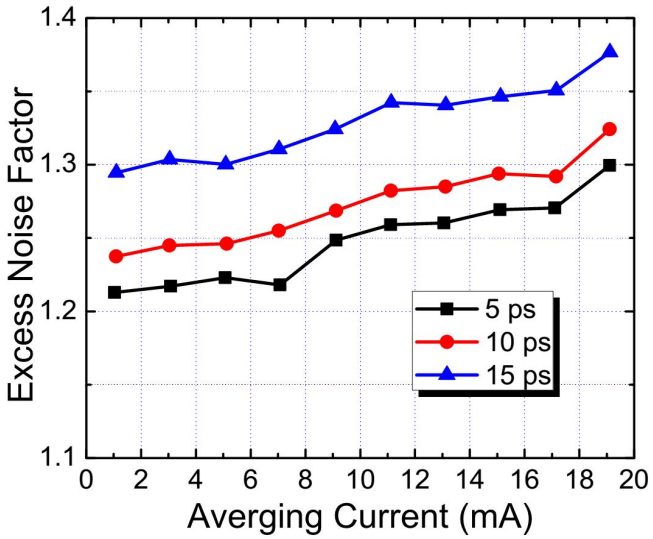


Figure. 7.4. Average photocurrent and optical pulse width dependence of F_H at 10 GHz.

The results in Fig. 7.3 and Fig. 7.4 show that $F_H < 2$ within the bandwidth of the detector. At 10 GHz and 18 V bias, the excess noise factor is only a 1 to 1.5 dB correction to the shot noise limit of Eq. 7.1, depending on the optical pulse width. At lower frequencies the correction is even smaller. However, when considering phase noise, the contribution of the excess noise can be more than an order of magnitude above the shot noise limit predicted by Eq. 7.4.

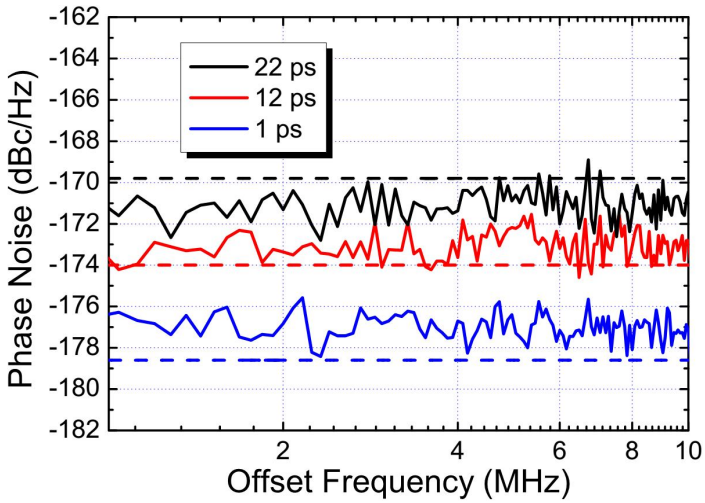


Figure. 7.5. Simulated single sideband phase noise level at three selected pulse widths. Dotted lines are the corresponding experimental results from [89].

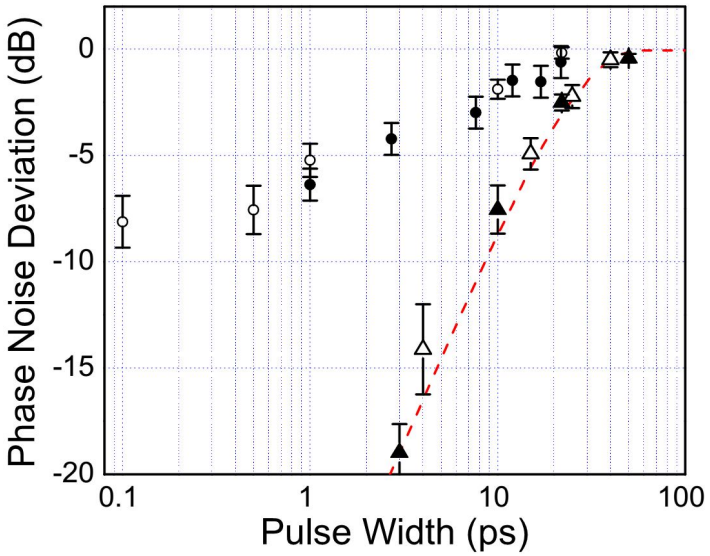


Figure. 7.6. Phase noise deviation from the long pulse limit. Red dash curve is analytical calculation from [90]; closed circle symbols are the experimental measurement from [89] with error bars; open triangle symbols are Monte Carlo simulation of shot noise only; closed triangle symbols are Monte Carlo simulation of shot noise and distributed absorption only; open circle symbols are Monte Carlo simulation of shot noise, photon absorption and carrier scattering.

Phase noise simulation. – As illustrated in Fig. 7.2, the detector's excess noise is related to randomness in the shape of each current impulse. This results in randomness in the "center of mass" of each impulse, contributing to the timing jitter of the electrical pulse train and to the phase noise of any harmonic of f_r. This is in addition to the timing jitter due

to the shot noise of Eq. 7.4. Since variations in the impulse shape give rise to the frequency dependence of F_H, it is this variation with frequency, not the nominal value that indicates added phase noise. The phase noise/timing jitter of a photonically generated microwave was determined by multiplying the simulated pulse train with a sinusoidal reference signal whose zero-crossings overlap with the arrival of the pulses, followed by a low-pass filter. Fourier transformation then yields the phase noise power spectrum. Fig. 7.5 shows the simulated phase noise level for three selected pulse widths. Dotted lines are the corresponding experimental results from [89]. The simulated phase noise levels are all within 1-2 dB of the experimental results.

The impact of the various noise terms was analyzed by removing them individually and recalculating the resulting phase noise. These calculations are shown in Fig. 7.6, where the phase noise deviation vs. optical pulse width is plotted. In the absence of any excess noise from the PD, only the shot noise is present, and the Monte Carlo simulation should reproduce the analytical model of [90]. The agreement between the Monte Carlo simulation and the analytical calculation is excellent, and provides an external consistency check of the model.

For MUTC devices, the impact of distributed photon absorption is minor, as it only slightly increases the noise above the shot noise limit. The dominant noise term is photo-carrier scattering, placing the phase noise more than 20 dB above the shot noise limit for 1 ps pulses. Moreover, the noise does not continue to decrease with decreasing pulse width, but reaches a floor nearly 9 dB below the long pulse phase noise. Also plotted is the experimental phase noise deviation from [89], with an updated value for 1 ps pulses. The dominance of photo-carrier scattering in the simulated phase noise combined with the close agreement of the phase noise level with experimental results leads to the conclusion that

randomness in carrier scattering is the limit of the lowest phase noise floors yet generated by the detection of ultrashort optical pulses.

7.3 Chapter Summary

While contributing little to the total photocurrent noise, excess noise in reverse-biased photodiodes can dominate the phase noise floor of microwave signals generated by detecting ultralow-jitter optical pulse trains. By developing a Monte Carlo model of carrier transport, we have evaluated the impact of photo-carrier scattering and distributed absorption for high power, high linearity MUTC detectors. It was determined that scattering results in phase noise several orders of magnitude above the shot noise limit. While the impact of distributed absorption was minimal in the device studied, other detector structures, such as waveguide detectors [94], may be more sensitive to distributed absorption. As with shot noise, reducing the impact of the excess noise can be achieved by increasing the photodetected microwave power with higher power optical pulses. Additionally, the transit time variance has a smaller impact at lower microwave carrier frequencies, thus operating further from the transit time limit should lead to improved phase noise performance.

8. Future Work

8.1 High Gain InAs avalanche photodiode

As mentioned in the discussion of InAs APDs in chapter 3, it is desirable to have gain as high as possible at low bias, which means the depletion region needs to be as thick as the crystal growth technology will permit. While further decrease of background doping is difficult, a tandem structure to cascade several stages of multiplication region can be employed. The electric field distribution of a single stage homojunction PIN structure with a 12 μm-thick i-region is shown in Fig. 8.1. The peak electric field is extremely high, which will significantly increase the tunneling current and degrade performance. However, a two-stage PIN structure with a 6 μm-thick i-region in each stage circumvents this problem. Hence the multi-stage tandem InAs structure could reach higher gain while maintaining dark current comparable to single-stage APDs.

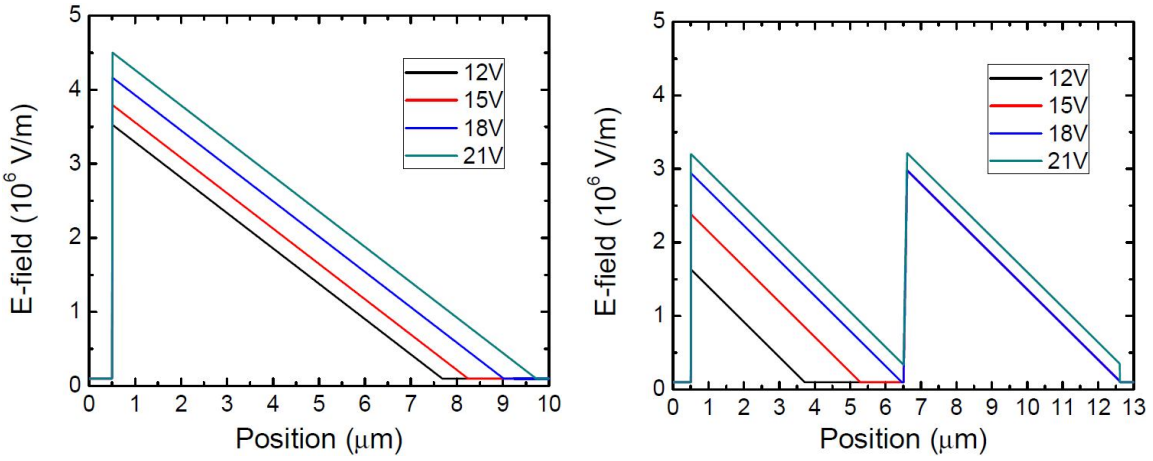


Figure 8.1. Electric field distribution of (left) single-stage APD with a 12 μm-thick i-region, and (right) two-stage PIN structure with a 6 μm-thick i-region in each stage.

We have both experimentally and theoretically confirmed that k is close to 0 for InAs APDs, which makes them possible candidates for low-noise infrared APD receivers. APD receiver performance is usually characterized by sensitivity that measures how bit-

error rate changes with power level. Fig. 8.2 demonstrates the origin of bit errors [98]. I_1 is the output current of an APD receiver in response to an optical signal, representing bit "1", while I_0 is that when there is no optical signal, representing bit "0". Both I_1 and I_0 fluctuate as a result of thermal and shot noise. I_D is the threshold decision level. Therefore, if measured I_1 is below I_D, a bit-error will be introduced, as well as when measured I_0 is above I_D. Hence the bit-error rate can be quantified as:

$$BER = p(1)p(0/1) + p(0)p(1/0) = 1/2[p(0/1) + p(1/0)]. \quad (8.2)$$

Where p(1) is the probability of transmitted bit "1", and p(0/1) is the probability that bit "1" is decoded as "0", similarly for p(0) and p(1/0).

If we assume Gaussian statistics for the current (I_0 and I_1) then we will have:

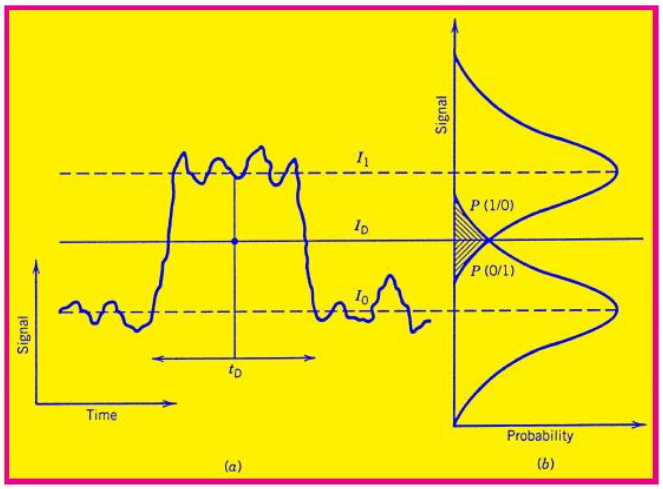


Figure 8.2. Illustration of bit-error rate in APD receivers.

$$BER = \frac{1}{4}[erfc(\frac{I_1 - I_D}{\sigma_1\sqrt{2}}) + erfc(\frac{I_D - I_0}{\sigma_0\sqrt{2}})]. \quad (8.3)$$

Where, σ_1, and σ_0 are the standard deviation of current I_1, and I_0, respectively. Choosing the threshold level, I_D, to minimize BER, we have:

$$I_D = \frac{\sigma_0 I_1 + \sigma_1 I_0}{\sigma_0 + \sigma_1} \quad (8.4)$$

$$Q = \frac{I_1 - I_0}{\sigma_1 + \sigma_0} \tag{8.5}$$

$$BER = \frac{1}{2} erfc(\frac{Q}{\sqrt{2}}) \quad . \tag{8.6}$$

Return-to-zero binary signal means I_0 should be zero as there are no incident photons during the 0 bits. Therefore, σ_0 is simply caused by shot noise of dark current and thermal noise. It can be expressed as:

$$\sigma_0^{\ 2} = 2qI_{dark}M^2F(M)B + 4kTB/R \, . \tag{8.7}$$

On the other hand, noise in I_1 is caused by shot noise of total current and also thermal noise. Therefore:

$$\sigma_1^{\ 2} = 2q(I_{photo} + I_{dark})M^2F(M)B + 4KTB/R \quad . \tag{8.8}$$

An estimation using the parameters of typical InAs APDs showed that only ~10 μW optical power is needed to achieve Q ~ 6 (see Fig. 8.3), which means the BER will be around 10^{-12} by eq. 8.6.

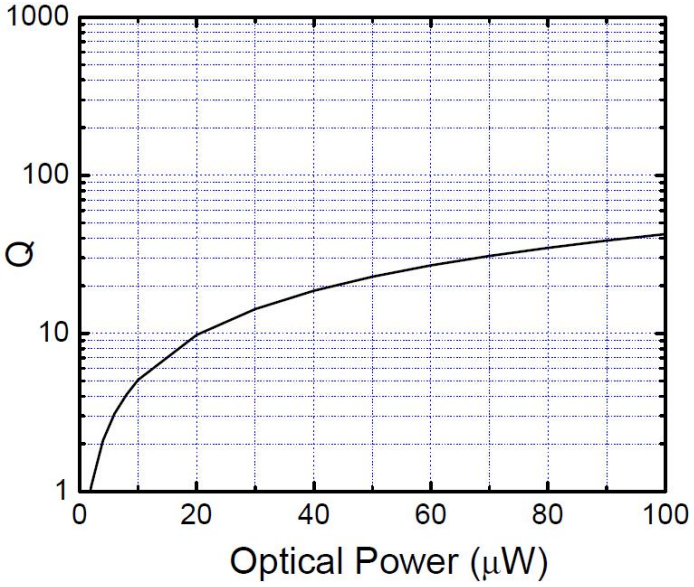
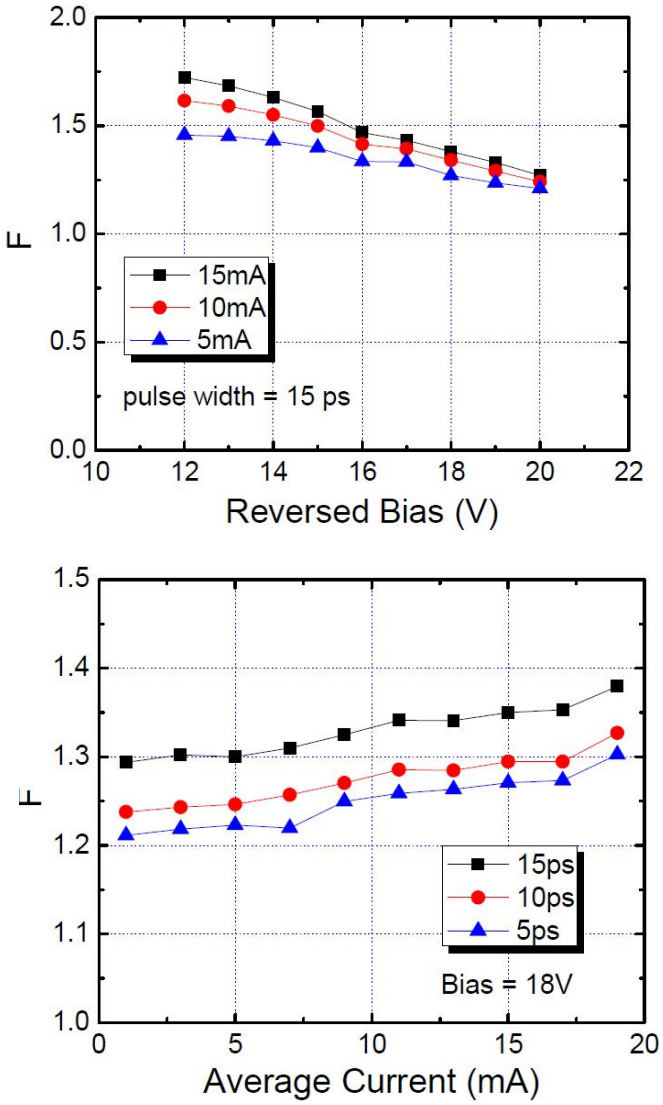


Figure 8.3. Calculation of Q as a function of optical power for InAs APD receiver.

8.2 Minimize phase noise in a photonic microwave signal

In chapter 7, we concluded that the shot noise from carrier transport limits the imbalance effect and results in higher phase noise. Therefore, it will be beneficial to study the details of carrier transport. In order to do this, we have studied the relationship between the excess noise factor $F_{H(f)}$ and the operating conditions of the photodiode. Figure 8.4 shows some initial simulation results.



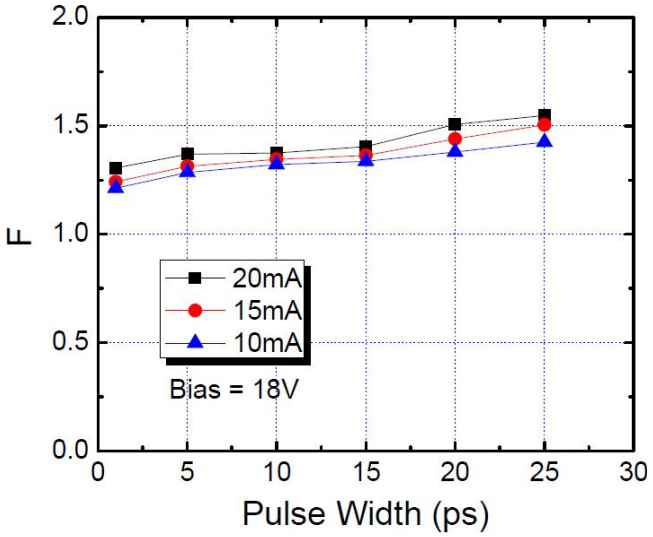


Figure 8.4. Calculated excess noise factor at different bias voltage, average current, and pulse-width.

With higher reverse bias, lower average photo-current, or shorter pulse width, the excess noise factor is smaller. With further systematic study, we believe that it is possible to find the optimized operating conditions so that the variance will be minimized.

Monte Carlo simulation is also performed to investigate how the phase noise level changes with temperature. At low temperature, some scattering mechanisms, such as phonon scattering, tend to have lower scattering rate. This will make the carrier transport slightly more deterministic and reduce the variance of the photodiode impulse response. Simulation results are shown in Fig. 8.5. Bias voltage of the photodiode is set at -5 V to make sure that there is no excess noise introduced by impact ionization, especially at low temperature; the pulse repetition rate is 2 GHz and the average photocurrent is ~ 15 mA. As we can see, the phase noise level still exhibits saturation due to variance of the impulse response, but as the temperature is reduced, it will decrease slightly.

Simulation of different photodiode structures also seems essential as we may find a relationship between the variance factor and aspects of the device structure, such as layer thickness and doping level, which will help to optimize future designs.

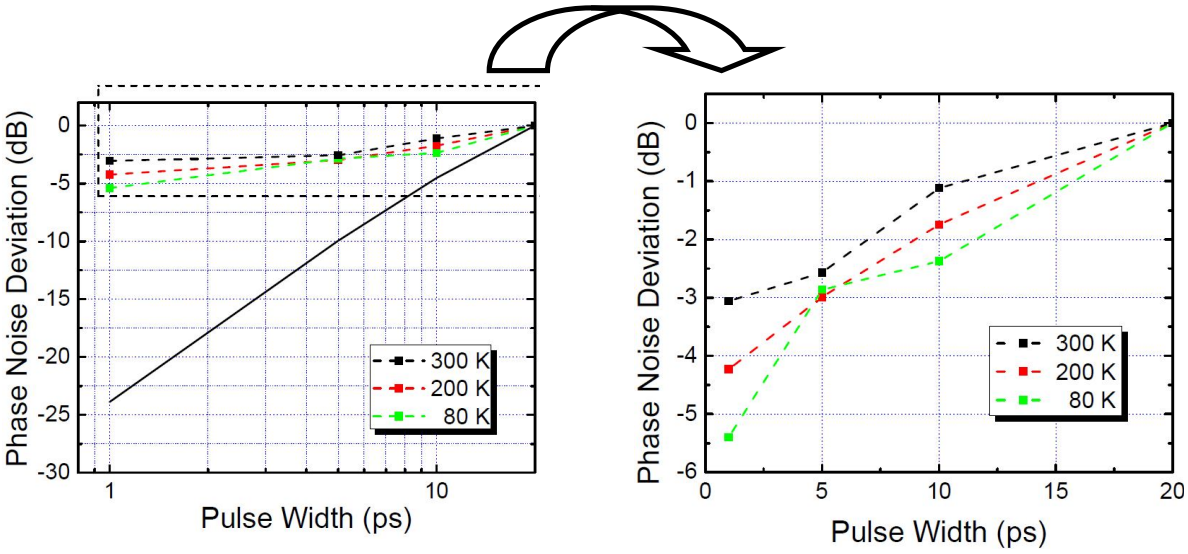


Figure 8.5. Simulation of phase noise level at different temperatures.

References:

[1] R.J. McIntyre, "Multiplication noise in uniform avalanche diodes," IEEE Trans. Electron. Devices, vol. ED-13, no. 1, pp. 164-168, Jan. 1966.

[2] R. B. Emmons, "Avalanche Photodiode Frequency Response", Journal of Applied Physics, vol. 38, no. 9, Aug. 1967.

[3] C. Y. Park, K. S. Hyun, S. K. Kang, M. K. Song, T. Y. Yoon, H. M. Kim, H. M. Park, S.-C. Park, Y. H. Lee, C. Lee, and J. B. Yoo, "High-performance InGaAs/InP photodiode for 2.5 Gb/s optical receiver," Optical and Quantum Elec. Vol. 24, no. 5, 1995.

[4] Ghulam Hasnain, Wayne G. Bi, S. Song, John T. Anderson, Nick Moll, Chung-yi Su, James N. Hollenhorst, Nicholas D. Baynes, I. Athroll, Sean Amos, and R. M. Ash, "Buried-Mesa Avalanche Photodiodes", IEEE Journal of Quantum Electronics, vol. 34, no. 12, 1998.

[5] M. A. Itzler, C. S. Wang, S. McCoy, N. Codd, N. Komaba, "Planar bulk InP avalanche photodiode design for 2.5 and 10 Gb/s applications", Proceedings of ECOC'98 – 24th European Conference on Optical Communication, vol. 1, Sep. 1998.

[6] C. A. Amiento and S. H. Groves, "Impact ionization in (100)-, (110)-, and (111)-oriented InP avalanche photodiodes," Applied Physics Letters, vol. 43, no. 2, 1983.

[7] L. W. Cook, G. E. Bulman, and G. E. Stillman, "Electron and hole ionization coefficients in InP determined by photomultiplication measurements," Applied Physics Letters, vol. 40, no. 7, 1982.

[8] Ning Li, Rubin Sidhu, Xiaowei Li, Feng Ma, Xiaoguang Zheng, Shuling Wang, Gauri Karve, Stephane Demiguel, Archie L. Holmes, Jr., and Joe C. Campbell, "InGaAs/InAlAs avalanche photodiode with undepleted absorber," Applied Physics Letters, vol. 82, 2003.

[9] J. P. R. David, C. H. Tan, "Material Considerations for Avalanche Photodiodes", IEEE Journal of Selected topics in quantum electronics, vol. 14, no. 4, 2008.

[10] K. F. Li, S. A. Plimmer, J. P. R. David, R. C. Tozer, G. J. Rees, P. N. Robson, C. C. Button, and J. C. Clark, "Low avalanche noise characteristics in thin InP p -i-n

diodes with electron initiated multiplication," *IEEE Photon. Tech. Lett.*, vol. 11, pp. 364–366, Mar. 1999.

[11] P. Yuan, C. C. Hansing, K. A. Anselm, C. V. Lenox, H. Nie, A. L. Holmes, Jr., B. G. Streetman, and J. C. Campbell, "Impact ionization characteristics of III-V semiconductors for a wide range of multiplication region thicknesses," *IEEE J. Quantum Electron.*, vol. 36, pp. 198–204, Feb. 2000.

[12] K. F. Li, D. S. Ong, J. P. R. David, G. J. Rees, R. C. Tozer, P. N. Robson, and R. Grey, "Avalanche multiplication noise characteristics in thin GaAs p -i-n diodes," IEEE Trans. Electron Dev., vol. 45, pp. 2102–2107, Oct. 1998.

[13] C. Hu, K. A. Anselm, B. G. Streetman, and J. C. Campbell, "Noise characteristics of thin multiplication region GaAs avalanche photodiodes," Appl. Phys. Lett., vol. 69, no. 24, pp. 3734–3736, 1996.

[14] D. S. Ong, K. F. Li, G. J. Rees, G. M. Dunn, J. P. R. David, and P. N. Robson, "A Monte Carlo investigation of multiplication noise in thin p-i-n GaAs avalanche photodiodes," IEEE Trans. Electron Dev., vol. 45, pp. 1804–1810, Aug. 1998.

[15] S. A. Plimmer, J. P. R. David, D. C. Herbert, T.-W. Lee, G. J. Rees, P. A. Houston, R. Grey, P. N. Robson, A. W. Higgs, and D. R. Wight, "Investigation of impact ionization in thin GaAs diodes," IEEE Trans. Electron Dev., vol. 43, pp. 1066–1072, July 1996.

[16] C. Lenox, P. Yuan, H. Nie, O. Baklenov, C. Hansing, J. C. Campbell, and B. G. Streetman, "Thin multiplication region InAlAs homojunction avalanche photodiodes," Appl. Phys. Lett., vol. 73, pp. 783–784, 1998.

[17] P. Yuan, C. C. Hansing, K. A. Anselm, C. V. Lenox, H. Nie, A. L. Holmes, Jr., B. G. Streetman, and J. C. Campbell, "Impact ionization characteristics of III-V semiconductors for a wide range of multiplication region thicknesses," IEEE J. Quantum Electron., vol. 36, pp. 198–204, Feb. 2000.

[18] M. A. Saleh, M. M. Hayat, P. O. Sotirelis, A. L. Holmes, J. C. Campbell, B. Saleh, and M. Teich, "Impact-ionization and noise characteristics of thin III-V avalanche photodiodes," IEEE Trans. Electron Devices, vol. 48, pp. 2722–2731, Dec. 2001.

[19] B. K. Ng, J. P. R. David, R. C. Tozer, G. J. Rees, Y. Feng, J. H. Zhao, and M. Weiner, "Nonlocal effects in thin 4H-SiC UV avalanche photodiodes," IEEE Trans. Electron Devices, vol. 50, pp. 1724–1732, Aug. 2003.

[20] P. Yuan, S. Wang, X. Sun, X. G. Zheng, A. L. Holmes, Jr., and J. C. Campbell, “Avalanche photodiodes with an impact-ionization-engineered multiplication region,” IEEE Photon. Technol. Lett., vol. 12, pp. 1370–1372, Oct. 2000.

[21] S. Wang, R. Sidhu, X. G. Zheng, X. Li, X. Sun, A. L. Holmes, Jr., and J. C. Campbell, “Low-noise avalanche photodiodes with graded impact-ionization-engineered multiplication region,” IEEE Photon. Technol. Lett., vol. 13, pp. 1346–1348, Dec. 2001.

[22] S. Wang, F. Ma, X. Li, R. Sidhu, X. G. Zheng, X. Sun, A. L. Holmes, Jr., and J. C. Campbell, “Ultra-low noise avalanche photodiodes with a “centered-well” multiplication region,” IEEE J. Quantum Electron., vol. 39, pp. 375–378, Feb. 2003.

[23] G. M. Dunn, R. Ghin, G. J. Rees, J. P. R. David, S. Plimmer, and D. C. Herbert, "Monte Carlo simulation of high-field transport and impact ionization in AlGaAs p+in+ diodes," Semicond. Sci. Technol. 14, pp. 994-1000, 1999.

[24] G. M. Dunn, G. J. Rees, J. P. R. David, S. Plimmer, and D. C. Herbert, "Monte Carlo simulation of impact ionization and current multiplication in short GaAs p+in+ diodes," Semicond. Sci. Technol. 12, pp. 111-120, 1997.

[25] Feng Ma, Xiaowei Li, and Joe C. Campbell, "Monte Carlo simulations of Hg0.7Cd0.3Te avalanche photodiodes and resonance phenomenon in the multiplication noise," Appl. Phys. Lett, Vol. 83, No. 4, Jul. 2003.

[26] C. K. Chia, "Numerical simulation of impact ionization in Ge/AlxGa1-xAs avalanche photodiode," Appl. Phys. Lett, Vol. 97, No. 7, 2010.

[27] H. Shichijo and K. Hess, "Band-structure-dependent transport and impact ionization in GaAs," Phys. Rev. B, vol. 38, No. 8, 1981.

[28] L. V. Keldysh, "Kinetic theory of impact ionization in semiconductors," Soviet Phys. JETP, vol. 37, No. 10, 1960.

[29] J. P. Gordon, R. E. Nahory, M. A. Pollack, and J. M. Worlock, "Low-Noise Multistage Avalanche Photodetector," IEEE Electronic Lett., vol. 15, No. 17, 1979.

[30] S. Rakshit and N. B. Charkraborti, "Multiplication noise in multi-heterostructure avalanche photodiodes," Solid State Electronics, vol. 26, No. 10, 1983.

[31] W. R. Clark, K. Vaccaro, and W. D. Waters, "InAlAs-InGaAs based avalanche photodiodes for next generation eye-safe optical receivers," Proc. Of SPIE Vol. 6796, 67962H, 2007.

[32] I. Vurgaftman and J. R. Meyer, "Band parameters for III- V compound semiconductors and their alloys," Journal of Appl. Phys., vol. 89, No. 11, 2001.

[33] C. Lenox, P. Yuan, H. Nie, O. Baklenov, C. Hansing, J. C. Campbell, A. L. Holmes, Jr., and B. G. Streetman, "Thin multiplication region InAlAs homojunction avalanche photodiodes," Appl. Phys. Lett., vol. 73, No. 6, 1998.

[34] Y. L. Goh, J. S. Ng, C. H. Tan, W. K. Ng, and J. P. R. David, "Excess Noise Measurement in In0.53Ga0.47As," IEEE Photonics Tech. Lett., vol. 17, No. 11, 2005.

[35] W. R. Clark, K. Vaccaro, and W. D. Waters, "InAlAs-InGaAs based avalanche photodiodes for next generation eye-safe optical receivers," Proc. Of SPIE Vol. 6796, 67962H, 2007.

[36] A. Huntington, M. Compton, S. Coykendall, G. Soli, and G. M. Williams, "Lnear-Mode Single-Photon-Sensitive Avalanche Photodiodes for GHz-Rate Near-Infrared Quantum Communications," Military Communications Conference, pp. 1-6, 2008.

[37] A. R. J. Marshall, C. H. Tan, M. J. Steer, and J. P. R. David, “extremely low excess noise in InAs electron avalanche photodiodes”, IEEE Photon. Tech. lett. , vol. 21, no. 13, pp. 866-868, July, 2009.

[38] A. R. J. Marshall, J. P. R. David, and C. H. Tan, “Impact ionization in InAs electron avalanche photodiodes”, IEEE Trans. Electron. Devices, vol. 57, no. 10, pp 2631-2638, 2010.

[39] P. J. Ker, A. R. J. Marshall, J. P. R David, and C. H. Tan, “low noise high responsivity InAs electron avalanche photodiode for infrared sensing”, Phys. Status. Solidi C, vol. 9, no. 2, 2011.

[40] A. R. J. Marshall, P. J. Ker, A. Krysa, J. P. R. David, and C. H. Tan, “High speed InAs electron avbalanche photodiode overcome the conventional gain-bandwidth product limit”, Optical express, vol. 19, no. 23, 2011.

[41] H. Arabshahi, S. Golafrooz, "Monte Carlo Based Calculation of Electron Transport Properties in Bulk InAs, AlAs and InAlAs", Bulg. J. Phys. 37, 215-222, 2010.

[42] A. R. J. Marshall, J. P. R. David, and C. H. Tan, "Impact Ionization in InAs Electron Avalanche Photodiodes", IEEE Trans. On Elec. Devices, vol. 57, No. 10, 2010.

[43] A. R. J. Marshall, C. H. Tan, J. P. R.David, M. J. Steer, and M. Hopkinson,

"Growth and Fabrication of InAs APDs", 4th EMRS DRC Tech. Conferences, Edinburgh, 2007.

[44] H. S. Kim, E. Plis, A. Khoshakhlagh, S. Myers, N. Gautam, Y. D. Sharma, L. R. Dawson, S. Krishna, S. J. Lee, and S. K. Noh, "Performance Improvement of InAs/GaSb strained layer super-lattice detectors by reducing surface leakage current with SU-8 passivation", Appl. Phys. Lett. Vol. 96, 2010.

[45] K.S.Lau, C. H. Tan, B. K. Ng, K. F. Li, R. C. Tozer, J. P. R. David, and G. J. Rees, "Excess noise measurement in avalanche photodiodes using a transimpedance amplifier front-end", Meas. Sci. Technol. vol. 17, 2006.

[46] Pin Jern Ker, et al, "Temperature Dependence of Leakage Current in InAs Avalanche Photodiodes", IEEE Journal of Quantum Electronics, vol. 47, no. 8, 2011.

[47] P. J. Ker, A. R. J. Marshall, J. P. R. David, C. H. Tan, "Low Noise High Responsivity InAs Electron Avalanche Photodidoes for Infrared Sensing", Phys. Status. Solidi C. No. 2, pp. 310-313, 2011.

[48] A. R. J. Marshall, A. Krysa, S. Zhang, A. S. Idris, S. Xie, J. P. R. David, and C. H. Tan, "High Gain InAs Avalanche Photodiodes", 6th EMRS DTC Tech. Conf. Edinburgh, 2009.

[49] A. R. J. Marshall, P. J. Ker, A. Krysa, J. P. R. David, and C. H. Tan, "High Speed InAs Electron Avalanche Photodiodes Overcome the Conventional Gain-bandwidth Product Limit", Optics Express, Vol. 19, No. 23, 2011.

[50] A. R. J. Marshall, P. Vines, P. J. Ker, J. P. R. David, and C. H. Tan, "Avalanche Multiplication and Excess Noise in InAs Electron Avalanche Photodiodes at 77K", IEEE Journal of Quantum Electronics. Vol. 47, No. 6, 2011.

[51] Fang, Z. M., K. Y. Ma, D. H. Jaw, R. M. Cohen, and G. B. Stringfellow, "Photoluminescence of InSb, InAs, and InAsSb grown by organometallic vapor phase epitaxy", J. Appl. Phys. 67, 11 (1990) 7034-7039.

[52] C. H. Tan, G. J. Rees, P. A. Houston, J. S. Ng, W. K. Ng, and J. P. R. David, "Temperature Dependence of Electron Impact Ionization in In0.53Ga0.47As", Appl. Phys. Lett. Vol. 84, No. 13, 2004.

[53] A. R. J. Marshall, P. Vines, P. J. Ker, J. P. R. David, and C. H. Tan, "Avalanche Multiplication and Excess Noise in InAs Electron Avalanche Photodiodes at 77K",

IEEE Journal of Quantum Electronics. Vol. 47, No. 6, 2011.

[54] W. Sun, X. Zheng, Z. Lu, J. C. Campbell, "Monte Carlo Simulation of InAlAs/InAlGaAs Tandem Avalanche Photodiodes", IEEE Journal of Quantum Electronics. Vol. 48, No. 4, 2012.

[55] A. R. J. Marshall, J. P. R. David, and C. H. Tan, "Impact Ionization in InAs Electron Avalanche Photodiodes", IEEE Trans. On Elec. Devices, vol. 57, No. 10, 2010.

[56] P. J. Ker, A. R. J. Marshall, J. P. R. David, C. H. Tan, "Low Noise High Responsivity InAs Electron Avalanche Photodidoes for Infrared Sensing", Phys. Status. Solidi C. No. 2, pp. 310-313, 2011.

[57] K.S.Lau, C. H. Tan, B. K. Ng, K. F. Li, R. C. Tozer, J. P. R. David, and G. J. Rees, "Excess noise measurement in avalanche photodiodes using a transimpedance amplifier front-end", Meas. Sci. Technol. vol. 17, 2006.

[58] M. M. Hayat, B. E. A. Saleh, and M. C. Teich, "Effect of dead space on gain and noise of double-carrier multiplication avalanche photodiodes," IEEE Trans. Electron Devices, vol. 39, pp.546-552, 1992.

[59] M. M. Hayat, O-H. Kwon, Y. Pan, P. Sotirelis, J. C. Campbell, B. E. A. Saleh, and M. C. Teich, ``Gain-bandwidth characteristics of thin avalanche photodiodes," IEEE Trans. Electron Devices, vol.49, pp. 4037-4039, May 2002.

[60] S. A. Plimmer, J. P. R. David, and D. S. Ong, “The merits and limitations of local impact ionization theory”, IEEE Trans. Electron. Dev., vol. 47, No. 5, pp. 1080-1088, MAY 2000.

[61] J. S. Ng, C. H. Tan, B. K. Ng, P. J. Hambleton, J. P. R. David, G. J. Rees, A. H. You, and D. S. Ong, “Effect of dead space on avalanche speed”, IEEE Trans. Electron. Dev., vol. 49, no. 4, pp. 544- 549, 2002.

[62] K. F. Li, S. A. Plimmer, J. P. R. David, R. C. Tozer, G. J. Rees, P. N. Robson, C. C. Button, and J. C. Clark, “Low avalanche noise characteristics in thin InP p -i-n diodes with electron initiated multiplication,” IEEE Photon. Tech. Lett., vol. 11, pp. 364–366, Mar. 1999.

[63] P. Yuan, C. C. Hansing, K. A. Anselm, C. V. Lenox, H. Nie, A. L. Holmes, Jr., B. G. Streetman, and J. C. Campbell, “Impact ionization characteristics of III-V semiconductors for a wide range of multiplication region thicknesses,” IEEE J.

Quantum Electron., vol. 36, pp. 198–204, Feb. 2000.

[64] K. F. Li, D. S. Ong, J. P. R. David, G. J. Rees, R. C. Tozer, P. N. Robson, and R. Grey, “Avalanche multiplication noise characteristics in thin GaAs p -i-n diodes,” IEEE Trans. Electron Dev., vol. 45, pp. 2102–2107, Oct. 1998.

[65] S. Wang, R. Sidhu, X. G. Zheng, X. Li, Sun, A. L. Holmes, Jr., and J. C. Campbell, “Low-noise avalanche photodiodes with graded impact-ionization-engineered multiplication region,” IEEE Photon. Tech. Lett., vol. 13, p. 1346 (2001)

[66] G. Kim, I. G. Kim, J. H. Baek, and O. K. Kwon, "Enhanced frequency response associated wtih negative photoconductance in an InGaAs/InAlAs avalanche photodetector", Appl. Phys. Lett., vol. 83, no. 6, 2003.

[67] J. W. Shi, Y. S. Wu, Z. R. Li, and P. S. Chen, "Impact-Ionization-Induced Bandwidth-Enhancement of a Si-SiGe-Based Avalanche Photodiode Operating at a Wavelength of 830 nm With a Gain-Bandwidth Product of 428 GHz", IEEE, Photonics Tech. Lett., vol. 19, no. 7, 2007.

[68] J. W. Shi, F. M. Kuo, F. C. Hong, and Y. S. Wu, "A Si/SiGe Based Impact Ionization Avalanche Transit Time Photodiode with Ultra-high Gain-Bandwidth Product (690GHz) for 10-Gb/s Fiber Communication", 2009 OSA/OFC/NFOEC.

[69] D. Dai, J. E. Bowers, "Simple Matrix-Method Modeling for Avalanche Photodetectors With Arbitrary Layer Structures and Absorption/Multiplication Coefficients", IEEE, Journal of Lightwave Technology, vol. 28, no. 9, 2010.

[70] D. Dai, M. J. Rodwell, J. E. Bowers, Y. Kang, M. Morse, "Derivation of the Small Signal Response and Equivalent Circuit Model for a Separate Absorption and Multiplication Layer Avalanche Photodetector", IEEE, Journal of Selected Topics in Quantum Electronics, vol. 16, no. 5, 2010.

[71] D. Dai, H. W. Chen, J. E. Bowers, Y. Kang, M. Morse, M. J. Paniccia, "Resonant normal-incidence separate-absorption-charge-multiplication Ge/Si avalanche photodiodes", Optics Express, vol. 17, no. 19, 2009.

[72] W. Sun, X. Zheng, Z. Lu, J. C. Campbell, "Monte Carlo Simulation of InAlAs/InAlGaAs Tandem Avalanche Photodiodes", IEEE Journal of Quantum Electronics. Vol. 48, No. 4, 2012.

[73] F. Ma, N. Li, J. C. Campbell, "Monte Carlo Simulation of the Bandwidth of InAlAs Avalanche Photodiodes", IEEE, Trans. Elec. Devices, vol. 50, no. 11, 2003.

[74] D. Dai, J. E. Bowers, "Simple Matrix-Method Modeling for Avalanche Photodetectors With Arbitrary Layer Structures and Absorption/Multiplication Coefficients", IEEE, Journal of Lightwave Technology, vol. 28, no. 9, 2010.

[75] R. J. McIntyre, " A New Look at Impact Ionization Part 1: A Theory of Gain, Noise, Breakdown Probability, and Frequency Response", IEEE Trans. Electon Devices, vol. 46, no. 8, 1999.

[76] X. Bai, P. Yuan, P. McDonald, J. Boisvert, J. Chang, R. Sudharsanan, M. Krainak, G. Yang, X. Sun, W. Liu, Z. Lu, Q. Zhou, W. Sun, J. C. Campbell, "Development of low excess noise SWIR APDs", SPIE proceedings, vol. 8353, Baltimore, 2012.

[77] W. K. Kulczyk, Q. V. Davis, "The avalanche photodiode as an electronic mixer in an optical receiver", IEEE Trans. Elec. Devices, vol. 19, no. 11, 1972.

[78] D. Dupuy, M. Lescure, "Improvement of the FMCW Laser Range-Finder by an APD Working as an Optoelectronic Mixer", IEEE, Trans. Inst. Meas., vol. 51, no. 5, 2002.

[79] M. J. Lee, H. S. Kang, K. H. Lee, W. Y. Choi, "IEEE Trans. Microwave Theory and Tech., vol. 56, no. 12, 2008.

[80] T. Kessler, C. Hagemann, C. Grebing, T. Legero, U. Sterr, F. Riehle, M. J. Martin, L. Chen, and J. Ye, "A sub-40-mHz-linewidth laser based on a silicon single-crystal optical cavity," Nature Photonics 6, 687-692 (2012).

[81] N. Hinkley, J. A. Sherman, N. B. Phillips, M. Schioppo, N. D. Lemke, K. Beloy, M. Pizzocaro, C. W. Oates, and A. D. Ludlow, "An Atomic Clock with 10(-18) Instability," Science 341, 1215-1218 (2013).

[82] B. J. Bloom, T. L. Nicholson, J. R. Williams, S. L. Campbell, M. Bishof, X. Zhang, W. Zhang, S. L. Bromley, and J. Ye, "An optical lattice clock with accuracy and stability at the 10(-18) level," Nature 506, 71-+ (2014).

[83] G. Santarelli, C. Audoin, A. Makdissi, P. Laurent, G. J. Dick, and A. Clairon, "Frequency stability degradation of an oscillator slaved to a periodically interrogated atomic resonator," IEEE Trans. Ultrason. Ferroelectr. Freq. Control 45, 887-894 (1998).

[84] J. Kim, J. A. Cox, J. Chen, and F. X. Kartner, "Drift-free femtosecond timing synchronization of remote optical and microwave sources," Nature Photonics 2, 733-736 (2008).

[85] J. F. Cliche and B. Shillue, "Precision timing control for radioastronomy - Maintaining femtosecond synchronization in the atacama large millimeter array," IEEE Control Syst. Mag. 26, 19-26 (2006).

[86] P. Ghelfi, F. Laghezza, F. Scotti, G. Serafino, A. Capria, S. Pinna, D. Onori, C. Porzi, M. Scaffardi, A. Malacarne, V. Vercesi, E. Lazzeri, F. Berizzi, and A. Bogoni, "A fully photonics-based coherent radar system," Nature 507, 341-345 (2014).

[87] S. A. Diddams, A. Bartels, T. M. Ramond, C. W. Oates, S. Bize, E. A. Curtis, J. C. Bergquist, and L. Hollberg, "Design and control of femtosecond lasers for optical clocks and the synthesis of low-noise optical and microwave signals," IEEE J. Sel. Top. Quantum Electron. 9, 1072-1080 (2003).

[88] T. M. Fortier, M. S. Kirchner, F. Quinlan, J. Taylor, J. C. Bergquist, T. Rosenband, N. Lemke, A. Ludlow, Y. Jiang, C. W. Oates, and S. A. Diddams, "Generation of ultrastable microwaves via optical frequency division," Nat. Photon. 5, 425-429 (2011).

[89] F. Quinlan, T. M. Fortier, H. Jiang, A. Hati, C. Nelson, Y. Fu, J. C. Campbell, and S. A. Diddams, "Exploiting shot noise correlations in the photodetection of ultrashort optical pulse trains," Nature Photonics 7, 290-293 (2013).

[90] F. Quinlan, T. M. Fortier, H. F. Jiang, and S. A. Diddams, "Analysis of shot noise in the detection of ultrashort optical pulse trains," J. Opt. Soc. Am. B-Opt. Phys. 30, 1775-1785 (2013).

[91] B. E. A. Saleh and M. C. Teich, Fundamentals of Photonics, Wiley Series in Pure and Applied Optics (John Wiley & Sons, Inc., New York, 1991), p. 966.

[92] Z. Li, H. P. Pan, H. Chen, A. Beling, and J. C. Campbell, "High-Saturation-Current Modified Uni-Traveling-Carrier Photodiode With Cliff Layer," IEEE J. Quantum Electron. 46, 626-632 (2010).

[93] K. J. Williams and R. D. Esman, "Nonlinearities in p-i-n Microwave Photodetectors," Journal of Lightwave Technology 14, 84-96 (1996).

[94] S. M. Madison, J. Klamkin, D. C. Oakley, A. Napoleone, J. J. Plant, and P. W. Juodawlkis, "Limits to Maximum Absorption Length in Waveguide Photodiodes," Ieee Photonics J 3, 676-685 (2011).

[95] G. Santarellim, C. Audoin, A. Makdissi, P. Laurent, G. J. Dick, and A. Clairon, "Frequency stability degradation of an oscillator slaved to a periodically interrogated atomic resonator", IEEE Trans. Ultrason. Ferroelectr. Freq. Control. 45, 1998.

[96] J. Kim, J. A. Cox, J. Chen, and F. X. Kartner, "Drift-free femtosecond timing synchronization of remote optical and microwave sources", Nat. Photon. 2. 2008.

[97] J. A. Scheer and J. L. Kurtz, "Coherent Radar Performance estimation (Artech House, 1993).

[98] Govind P. Agrawal, "Optical Communication Systems", University of Rochester, 2007.

Publications:

[1] **Wenlu Sun**, Franklyn Quinlan, Tara M. Fortier, Jean-Daniel Deschenes, Yang Fu, Scott A. Diddams, and Joe C. Campbell, "Broadband noise limit in the photodetection of ultralow jitter optical pulses", Accepted by Physics Review Letters.

[2] **Wenlu Sun**, Xiaoguang Zheng, Zhiwen Lu, Joe C Campbell, "Monte Carlo simulation of InAlAs/InAlGaAs tandem avalanche photodiodes", IEEE Journal of Quantum Electronics. Vol 48, no. 4. April. 2012.

[3] **Wenlu Sun**, Yang Fu, Zhiwen Lu, Joe Campbell, "Study of bandwidth enhancement and non-linear behavior in avalanche photodiodes under high power condition", Journal of Applied Physics, vol. 113, no. 4. Jan. 2013.

[4] **Wenlu Sun**, Xiaoguang Zheng, J Campbell, "Study of Excess Noise Factor under Non-Local Effect in Avalanche Photodiodes", IEEE Photonic Technology Letters. Vol. 26, no. 21. Aug, 2014.

[5] **Wenlu Sun**, Xiaoguang Zheng, Zhiwen Lu, Joe C Campbell, "Monte Carlo Simulation of AlGaAs Avalanche Photodiode", IEEE Journal of Quantum Electronics. Vol. 47, no. 12. Dec. 2011.

[6] **Wenlu Sun**, Xiaoguang Zheng, Zhiwen Lu, Baile Chen, Archie L Holmes, Joe C Campbell, "Numerical simulation of InAlAs/InAlGaAs tandem avalanche photodiodes", IEEE Photonics Conference. Oct. 2011.

[7] **Wenlu Sun**, Zhiwen Lu, Xiaoguang Zheng, JC Campbell, Scott J Maddox, Hari P Nair, Seth R Bank, "Charge-Compensated High Gain InAs Avalanche Photodiodes", Photonics Conference (IPC), 2012 IEEE, Sep. 2012.

[8] **Wenlu Sun**, Scott Maddox, Seth Bank, Joe C Campbell, "Record high gain from InAs avalanche photodiodes at room temperature", Device Research Conference (DRC), 2014 72nd Annual.

[9] Mohan Chen, **Wenlu Sun**, Guang-Can Guo andLixin He, "Substrate induced bandgap in multilayer epitaxial graphene on the 4H-SiC surface", basic solid state physics, vol. 248, no. 7, July 2011.

[10] Zhiwen Lu, **Wenlu Sun**, Joe C Campbell, Xudong Jiang, Mark A Itzler, "Pulsed gating with balanced InGaAs/InP single photon avalanche diodes", IEEE Journal of Quantum Electronics. Vol. 49, no. 5, May, 2013.

[11] Zhiwen Lu, **Wenlu Sun**, Xiaoguang Zheng, Joe Campbell, Xudong Jiang, Mark A Itzler, "Balanced InGaAs/InP avalanche photodiodes for single photon detection", SPIE NanoScience+ Engineering, Oct. 2012.

[12] Zhiwen Lu, **Wenlu Sun**, Qiugui Zhou, Joe Campbell, Xudong Jiang, Mark A Itzler, "Improved sinusoidal gating with balanced InGaAs/InP single photon avalanche diodes", Optics Express, vol. 21, no. 14. July, 2013.

[13] Zhiwen Lu, **Wenlu Sun**, Chong Hu, Archie Holmes, Joe C Campbell, Yimin Kang, Han-Din Liu, "Ge on Si and InP/InGaAs single photon avalanche diodes", SPIE Optical Engineering+ Applications, Sep. 2011.

[14] Zhiwen Lu, **Wenlu Sun**, Joe Campbell, Xudong Jiang, Mark A Itzler, "Balanced detection in single photon counting", SPIE Defense, Security, and Sensing, May. 2013.

[15] Baile Chen, **Wenlu Sun**, Joe C Campbell, Jr AL Holmes, "Quantum efficiency modeling of PIN photodiodes with InGaAs/GaAsSb quantum wells absorption region", Photonics Conference (PHO), 2011 IEEE, Oct. 2011.

[16] Scott J Maddox, **Wenlu Sun**, Zhiwen Lu, HP Nair, Joe C Campbell, Seth R Bank, "InAs Avalanche Photodiode with Improved Electric Field Uniformity", Device Research Conference (DRC), 2012 70th Annual, June, 2012.

[17] Scott J Maddox, **Wenlu Sun**, Zhiwen Lu, HP Nair, Joe C Campbell, Seth R Bank, "Enhanced low-noise gain from InAs avalanche photodiodes with reduced dark current and background doping", Applied Physics Letters. Vol. 101, no. 15, Oct. 2012.

[18] Joe C Campbell, **Wenlu Sun**, Zhiwen Lu, Mark A Itzler, Xudong Jiang. "Common-mode cancellation in sinusoidal gating with balanced InGaAs/InP single photon counting avalanche diodes", IEEE Journal of Quantum Electronics, vol. 48, no. 12. Dec. 2012.

[19] Min Ren, J Chen, **Wenlu Sun**, Xiao Jie Chen, Erik B Johnson, James F Christian, Joe C Campbell, "High-Detection-Sensitivity Al0. 8Ga0. 2As Avalanche Photodiodes", IEEE Photonic Technology Letters. Vol. pp, no. 99, Sep. 2014.

[20] Min Ren, Yan Liang, **Wenlu Sun**, Guang Wu, J Campbell, Heping Zeng, "Timing Response of Sinusoidal-Gated Geiger Mode InGaAs/InP APD", IEEE Photonic Technology Letters. Vol. 26, no. 17, Aug. 2014.

[21] Seth R Bank, Scott J Maddox, **Wenlu Sun**, Zhiwen Lu, HP Nair, Joe C Campbell, "Recent advances in InAs avalanche photodiodes", Photonics Conference (IPC), 2013 IEEE, Sep. 2013.

[22] Zhiwen Lu, Xiaoguang Zheng, **Wenlu Sun**, Joe Campbell, Xudong Jiang, Mark A Itzler, "InGaAs/InP Single Photon Avalanche Diodes", ECS Transactions, April. 2013.

[23] Zhiwen Lu, Xiaoguang Zheng, **Wenlu Sun**, Joe Campbell, Xudong Jiang, Mark A Itzler, "InGaAs/InP Single Photon Avalanche Diodes", ECS meeting, 2012.

[24] Zhiwen Lu, Xiaoguang Zheng, **Wenlu Sun**, Joe Campbell, Xudong Jiang, Mark A Itzler, "Characterization of sinusoidal gating of InGaAs/InP single photon avalanche diodes", SPIE Defense, Security, and Sensing, May. 2012.

[25] Qiugui Zhou, Dion C McIntosh, Yaojia Chen, **Wenlu Sun**, Zhi Li, Joe C Campbell, "Nanosphere natural lithography surface texturing as anti-reflective layer on SiC photodiodes", Optics Express. Vol. 19, no. 24, Nov. 2011.

[26] Kejia Li, Lijun Li, Petr P Khlyabich, Beate Burkhart, **Wenlu Sun**, Zhiwen Lu, Barry C Thompson, Joe C Campbell, "Breakdown mechanisms and reverse current-voltage characteristics of organic bulk heterojunction solar cells and photodetectors", Journal of

Applied Physics. Vol. 115, no. 22, June, 2014.

[27] Baile Chen, Qiugui Zhou, DC McIntosh, Jinrong Yuan, Yaojia Chen, **Wenlu Sun**, JC Campbell, AL Holmes, "Natural lithography nano-sphere texturing as antireflective layer on InP-based pin photodiodes" , Electronics letters, vol. 48, no. 21. Oct. 2010.

[28] Mark David Hammig, Xiao Jie Chen, Joseph C Campbell, Taehoon Kang, **Wenlu Sun**, Erik B Johnson, Kyusang Lee, James F Christian, "Development of AlGaAs photodiodes for use in wide band-gap solid-state photomultiliers", IEEE Transactions on Nuclear Science, vol. 60, April. 2013.

[29] Xiaogang Bai, Ping Yuan, Paul McDonald, Joseph Boisvert, James Chang, Rengarajan Sudharsanan, Michael Krainak, Guangning Yang, Xiaoli Sun, Wei Lu, Zhiwen Lu, Qiugui Zhou, **Wenlu Sun**, Joe Campbell, "Development of low excess noise SWIR APDs", SPIE Defense, Security, and Sensing, May. 2012.

Appendix I. Description of Source Code

apd.h:	header file, definitions of constants, structures and subroutines.
main.C:	read-in parameters, step up the bias, run the simulation and generate output files.
LoadParams.C:	read-in parameters from input file, calculate and store all scattering rates.
FindJunction.C:	find the location of pn junction.
BiasLoop.C:	loop through many bias.
InitArrays.C:	initialize all the arrays for storing of calculated data.
GetParams.C:	look up table for parameters for each grid.
SetBias.C:	set up the desired bias on the photodiode.
BuildEFTable.C:	calculate the electric field in the grids.
CalcBias.C:	used by SetBias.C to achieve accurate bias value.
FirstCarrierLoop.C:	generate all the photo-absorbed carriers, calculate the incoming time, absorption position, initial energy and velocity.
CarrierLoop.C:	loop through all the carriers to simulate carrier transport at each time step.
AbsorbPhoton.C:	calculate the absorption position for each incoming photon.
GetAbs.C:	look up the absorption position.
StartElectron.C:	execute drift and scattering for electron carrier.
ElectronDrift.C:	calculate the drift process of electron during each flight time.
ElectronScat.C:	simulate the scattering process of electron at each scattering event, calculate the new energy, velocity, etc.
StartHole.C:	execute drift and scattering for hole carrier.
HoleDrift.C:	calculate the drift process of hole during each flight time.
HoleScat.C:	simulate the scattering process of hole at each scattering event, calculate the new energy, velocity, etc.
DataToFile.C:	generate output files for gain, noise, current, gain distribution, distribution of impact ionization, etc.

Apd.h

```
//needed header files

#include <iostream>
#include <iomanip>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <time.h>
#include <stdio.h>
using namespace std;  //program constants
const int NUMX=3000;  //number of sections into which to divide structure

const int MAX_CHAR=31;  //maximum characters in a string
const unsigned long NUM_BIN=1000; //number of bins for time response
histogram.
const int MAX_COUNT=30000; //limit gain to prevent infinite loops
const int STEP_LIMIT=19; //size to limit number of steps
const float RANDOMBASE=float(RAND_MAX); //resolution of random number
const int MAXLAYERS=42; //maximum number of allowed layers in structure


//physical constants
const double q=1.60219e-19; //charge of an electron const double
const double e=2.71828183;
const double eps0=8.85419e-12; //dioelectric constant in vaccum in units
of F/m
const double ep0=8.85419e-12;  //ep0 is the same as eps0
const double pi=3.141592653589793;
const double h=1.05459e-34;     // h-bar
const double am0=9.10953e-31;
const double bk=1.38066e-23;

const double device_diameter = 50.0e-6;

const float tem=300.0;

const float de=0.035;    //energy increment
const int iemax=100;  //max energy steps

const int ivalley=2;       //number of conduction band valleys
const int iscat=11;         //number of scattering mechanisms

//device structures
struct layer {
  float Thickness, Doping;
  char Material[MAXCHARS]; //name of the material
  float Ae, Ah, Eacte, Eacth, ce, ch; //the parameters Alex used
  float EacteH, EacthH;  //hard threshold, above which definitely ionize
  float Abs; //the absorption coefficient in inverse meters
  float Ec[ivalley+1], Ev[ivalley+1]; //information about band edge
alignment, [1] for Gamma-valley; Ec[2] for L valley;  will extend to more
valleys.
    float eps,epf,ep; // relative dielectric constant at static and high
```

```
frequency
    float eg;  //band gap
    float me[ivalley+1], mh[ivalley+1], hs; //  effective masses; ONLY
consider heavy hole band for now.  hs is the hole scattering rate scaling
factor, to increase the hole scattering rate, e.g., supposingly to light
hole and so bands.
    float gm; //total scattering rate in this layer, as a function of
energy.
   float swk[ivalley+1][iscat+1][iemax+1];    //individual scattering
rate as a function of [iv][n][iemax]
    float af[ivalley+1],af2[ivalley+1],af4[ivalley+1];  //band
parameters.

    float smh[ivalley+1],hhm[ivalley+1],hm[ivalley+1];  //composite
constants made of am, q, h
   float rou,sv,hwo,hwij,hwe,hwaco;
   float qd2;

  float x;  //Al fraction, used for alloy scattering estimate
  float as; //alloy scattering coefficient.

  //a parrallel set of parameters for holes

   float gmH; //total scattering rate for holes in this layer
   float swkH[ivalley+1][iscat+1][iemax+1];

   float afH[ivalley+1],af2H[ivalley+1],af4H[ivalley+1];

   float smhH[ivalley+1],hhmH[ivalley+1],hmH[ivalley+1];
  //  float rouH,svH;   rou and sv seem not to depend on e or h.
}; //end of layer structure

struct device { //stores values relevant to device structure
  int NumLayers;
  layer DeviceLayers[MAXLAYERS];
  float acu_thickness[MAXLAYERS];
  float TotalThickness;
  float DepletionEnd;
  float DepletionStart;
  float charge_density[NUMX+1];
  float EFTable[NUMX+1];        //speed up calculations,
  float Potential[NUMX+1];    //potential energy as a func of position,
  float EFMax;                //peak electric field
  float Vapp,Vbi;
    int index[NUMX];
};

//data structure
struct stats{  //keeps track of ionization statistics

  int ImpactDist_e[NUMX+1];   //spatial dis. of electron impacts
  int ImpactDist_h[NUMX+1];   //spatial dis. of hole impacts
  int AbsDist[NUMX+1];        //distribution of absorption events
  int GainDist[MAX_COUNT+1];  //gain distribution

  int max_steps;              //number of biases estimated
  int stat_bin;               //bin interval for statistcs of
                      //SINGLE carrier gain (m).
```

```
  float gain[STEP_LIMIT];     //variables for gain and bias
  float Vapp[STEP_LIMIT];     //max_step isn't constant, so use huge
arrays
  float EFMax[STEP_LIMIT];    //track the peak e-field
  float sq_gain[STEP_LIMIT];  //to calculate the noise

  float max_time;             //longest time in device

  float Current[NUM_BIN+1];  //conduction current defined as instananeous
velocity x e charge
  float AC_bias[NUM_BIN+1];
  long impacte[NUM_BIN+2];
  long impacth[NUM_BIN+2];
  long total_count;
  long total_sq_count;
  long Ne;
  long Nh;


  float EnergyDist_e[NUMX+1];  //total energy of electron in this
position
  float VelocityDist_e[NUMX+1];  //total energy of electron in this
position
  float CountDist_e[NUMX+1];
  float EnergyDist_h[NUMX+1];  //total energy of hole in this position
  float VelocityDist_h[NUMX+1];  //total energy of hole in this position
  float CountDist_h[NUMX+1];
  int GDist_e[NUMX+1];      //Gamma valley electron
  int LDist_e[NUMX+1];      //L valley electron
  int GDist_h[NUMX+1];      //heavy hole
  int LDist_h[NUMX+1];      //light hole


}; //end of structure stats


struct params{  //parameters to determine calculation limits
  float tem;    //temperature
  float photonev; //incident photon energy
  float impfac; //impurity scattering rate scaling factor, try 10.
  float DeltaEg; //Eg temperature coefficient, ~0.05
  int I0;       //number of initial e-h pairs
  int avg;      //number of simulations over which final output is
averaged
  double max_time;  //max time checked =1 ns??
  float  min_time;  //minimum time scale. If smaller than this, set =
this.
  float delta_t;    //time step to update electric field.
  float printing_time;
  float temp_time;
  float Vapp;   //starting bias
  float Vinc;   //step to incrememt the bias
  float Vbi;    //built in potential
  float max_gain; //Max gain before terminating and saving
  int max_steps; //max number of electric fields to try
  int stat_bin;              //bin interval for statistcs of
```

```
                                //SINGLE carrier gain (m).
  double DC_current;
  char structfilename[MAX_CHAR];
  char gainfilename[MAX_CHAR];
  char distrfilename[MAX_CHAR];
  char freqfilename[MAX_CHAR];
  char infilename[MAX_CHAR];
  char carrierfilename[MAX_CHAR];
  char swk0filename[MAX_CHAR];
  char swk1filename[MAX_CHAR];
  char swk2filename[MAX_CHAR];
  char swkHfilename[MAX_CHAR];

  int mode;
  double quasi_charge;
  double frequency;
  //  char Enefilename[MAX_CHAR];
  //  char Enhfilename[MAX_CHAR];

};  //end of structure params

struct electron {

  float kx,ky,kz;  // components of wave vector
  //float sk;
  int iv; // valley index. iv=1 for Gamma-valley; iv=2 for L-Valley.

  //float ki;
  float ve;  //carrier average velocity during a flight time
  float e;   //carrier initial energy during each free flight.
  //  float ft[ivalley+1];
  float position, ft;

  int impact;
  int count;

  int parent;

};  //end of structure carriers

struct hole {

  float kx,ky,kz;  // components of wave vector
  //float sk;
    int iv; // valley index. iv=1 for heavy hole; iv=2 for light hole; 0
for so.

  //float ki;
  float ve;  //carrier average velocity during a flight time
  float e;   //carrier initial energy during each free flight.
  //  float ft[ivalley+1];
  float position, ft;

  int impact;
  int count;

  int parent;
};  //end of structure carriers
```

```
//function declarations

//to set up the device parameters
void SetBias(device&, params&, stats&, float JunctionLocation1);
float CalcBias(device&, float JunctionLocation1);
void BuildEFTable(device&, float JunctionLocation1);
float FindJunction(device&);
float GetDoping(device&, float position);
float GetAbs(device&, float position);
layer GetParams(device&, float position);

//for generating electrons and holes
void StartElectron(device&, params&, electron*, hole*, stats&, int
e_number);
void StartHole(device&, params&, hole*, electron*, stats&, int e_number);
float AbsorbPhoton(device&, stats&);

//working with data
void InitArrays(stats&);
void LoadParams(device&, params&);
void DataToFile(device&, stats&, params&);

//functions for program flow
void BiasLoop(device&, params&, stats&, electron*, hole*,
       float JunctionLocation1);
void CarrierLoop(device&, params&, stats&, electron*, hole*);
void FirstCarrierLoop(device&, params&, stats&, electron*, hole*);


//functions for physical processes
void ElectronDrift(electron&,device&,params&, stats& APDresults, float
tau);
void HoleDrift(hole&,device&,params&, stats& APDresults, float tau);

void ElectronScat(device&,params&,electron&);
void HoleScat(device&,params&,hole&);

void realft (float data[], unsigned long n, int isign);
void four1(float data[],unsigned long n, int isign);
```

Main.C

```
#include "apd.h"

int main(){

      char infilename[MAX_CHAR] = "input.prm";

    params APDparams;  //list of parameters
```

```
    device APDdevice;
    stats APDresults;  //calculation data

      strcpy(APDparams.infilename,infilename);

    srand(time(NULL));
      LoadParams(APDdevice, APDparams);  //input the parameters from
*.prm

      struct electron* NewElectron = (struct
electron*)malloc((APDparams.I0*int(APDparams.max_gain))*sizeof(struct
electron));
      struct hole* NewHole = (struct
hole*)malloc((APDparams.I0*int(APDparams.max_gain))*sizeof(struct hole));

      float JunctionLocation1=FindJunction(APDdevice);

      BiasLoop(APDdevice, APDparams, APDresults, NewElectron, NewHole,
JunctionLocation1);

      DataToFile(APDdevice, APDresults, APDparams);
      cout<<"\n                    # data saved, enjoy!"<<endl;
      return 0;
}
```

LoadParams.C

```
#include "apd.h"

/********************
Function to get parameters from input datafile
***************************/

void LoadParams(device& APDdevice, params& APDparams)
  {
```

```
char param_name[MAX_CHAR]="/";
int i=0,j=1,ie=0;  //counter variable
float temp=0.0;
float dx;  //spacing of grid
ifstream in_file(APDparams.infilename);  //open file


if(in_file.fail()) {exit(1);}

do { //step through file
  in_file.getline(param_name,MAX_CHAR,'\n');
  //      in_file>>param_name;
  i++;
  if(i>10000)
  {cout<<"there is an error in the parameter input" << endl;}
  else if (strncmp(param_name,"/",1)==0)
  {
     in_file.getline(param_name,MAX_CHAR,'\n');
   cout<<"skip the / part"<<endl;}
  else if (strncmp(param_name,"tem",3)==0)
  {in_file>>APDparams.tem;}
  else if (strncmp(param_name,"photonev",8)==0)
  {in_file>>APDparams.photonev;}
  else if (strncmp(param_name,"impfac",6)==0)
  {in_file>>APDparams.impfac;}
  //scaling factor for impurity scattering rate
  else if (strncmp(param_name,"DeltaEg",7)==0)
  {in_file>>APDparams.DeltaEg;}
  //temperature coefficient of Eg, ~0.05.

  else if (strncmp(param_name,"I0",2)==0)
  {in_file>>APDparams.I0;}
  else if (strncmp(param_name,"max_time",8)==0)
  {in_file>>APDparams.max_time;}
  else if (strncmp(param_name,"min_time",8)==0)
  {in_file>>APDparams.min_time;}
    else if (strncmp(param_name,"delta_t",7)==0)
  {in_file>>APDparams.delta_t;}
  else if (strncmp(param_name,"printing_time",13)==0)
    {in_file>>APDparams.printing_time;}
  else if (strncmp(param_name,"Vapp",4)==0)
  {in_file>>APDparams.Vapp;}
  else if (strncmp(param_name,"Vinc",4)==0)
  {in_file>>APDparams.Vinc;}
  else if (strncmp(param_name,"Vbi",3)==0)
  {in_file>>APDparams.Vbi;}
  else if (strncmp(param_name,"max_gain",8)==0)
  {in_file>>APDparams.max_gain;}
  else if (strncmp(param_name,"max_steps",9)==0)
  {in_file>>APDparams.max_steps;}
  else if (strncmp(param_name,"stat_bin",8)==0)
  {in_file>>APDparams.stat_bin;}
  else if (strncmp(param_name,"mode",4)==0)
    {in_file>>APDparams.mode;}
  else if (strncmp(param_name,"quasi_charge",12)==0)
    {in_file>>APDparams.quasi_charge;}
  else if (strncmp(param_name,"frequency",9)==0)
    {in_file>>APDparams.frequency;}
```

```
      else if (strncmp(param_name,"DC_current",10)==0)
        {in_file>>APDparams.DC_current;}
      else if (strncmp(param_name,"structfilename",14)==0)
      {in_file>>APDparams.structfilename;}
      else if (strncmp(param_name,"gainfilename",12)==0)
      {in_file>>APDparams.gainfilename;}
      else if (strncmp(param_name,"distrfilename",12)==0)
      {in_file>>APDparams.distrfilename;}
      else if (strncmp(param_name,"freqfilename",12)==0)
      {in_file>>APDparams.freqfilename;}
      else if (strncmp(param_name,"carrierfilename",15)==0)
      {in_file>>APDparams.carrierfilename;}
      else if (strncmp(param_name,"swk0filename",12)==0)
      {in_file>>APDparams.swk0filename;}
      else if (strncmp(param_name,"swk1filename",12)==0)
      {in_file>>APDparams.swk1filename;}
      else if (strncmp(param_name,"swk2filename",12)==0)
      {in_file>>APDparams.swk2filename;}
      else if (strncmp(param_name,"swkHfilename",12)==0)
      {in_file>>APDparams.swkHfilename;
         }
      else if (strncmp(param_name,"structure",9)==0) //
      {in_file>>APDdevice.DeviceLayers[j].Material;

      while(strncmp(APDdevice.DeviceLayers[j].Material,"endstructure",12)
!=0)
        {
          in_file>>APDdevice.DeviceLayers[j].Thickness;
          in_file>>APDdevice.DeviceLayers[j].Doping;
          in_file>>APDdevice.DeviceLayers[j].Abs;

          in_file>>APDdevice.DeviceLayers[j].Eacte;
          in_file>>APDdevice.DeviceLayers[j].EacteH;
          in_file>>APDdevice.DeviceLayers[j].Eacth;
          in_file>>APDdevice.DeviceLayers[j].EacthH;


                in_file>>APDdevice.DeviceLayers[j].ce;
                //power index, usually =4.0
                  in_file>>APDdevice.DeviceLayers[j].ch;

          in_file>>APDdevice.DeviceLayers[j].Ae;
          in_file>>APDdevice.DeviceLayers[j].Ah;


          float Ec1,Ec2,Ec0,Ev1,Ev2,Ev0;
          in_file>>Ec1;
          in_file>>Ec2;
          in_file>>Ec0;
          in_file>>Ev1;
          in_file>>Ev2;
          in_file>>Ev0;

            APDdevice.DeviceLayers[j].Ec[1]=Ec1-Ec1*(1.0-
APDparams.tem/300)*APDparams.DeltaEg;
            APDdevice.DeviceLayers[j].Ec[2]=Ec2-Ec2*(1.0-
APDparams.tem/300)*APDparams.DeltaEg;
            APDdevice.DeviceLayers[j].Ec[0]=Ec0-Ec0*(1.0-
```

```
APDparams.tem/300)*APDparams.DeltaEg;

            APDdevice.DeviceLayers[j].Ev[1]=Ev1-Ev1*(1.0-
APDparams.tem/300)*APDparams.DeltaEg;
            APDdevice.DeviceLayers[j].Ev[2]=Ev2-Ev2*(1.0-
APDparams.tem/300)*APDparams.DeltaEg;
            APDdevice.DeviceLayers[j].Ev[0]=Ev0-Ev0*(1.0-
APDparams.tem/300)*APDparams.DeltaEg;

          //temperature dependent Eg;

          in_file>>APDdevice.DeviceLayers[j].me[1];
          in_file>>APDdevice.DeviceLayers[j].me[2];
          in_file>>APDdevice.DeviceLayers[j].me[0];

          in_file>>APDdevice.DeviceLayers[j].mh[1];  //mhh
          in_file>>APDdevice.DeviceLayers[j].mh[2];  //mlh
          in_file>>APDdevice.DeviceLayers[j].mh[0];  //mso
            in_file>>APDdevice.DeviceLayers[j].hs;
          //hole scattering rate scaling factor, should be larger than 1


          in_file>>APDdevice.DeviceLayers[j].eps; //eps=12.90*ep0 for
GaAs
          in_file>>APDdevice.DeviceLayers[j].epf; //epf=10.92*ep0 for
GaAs
                APDdevice.DeviceLayers[j].ep
  =1.0/(1.0/APDdevice.DeviceLayers[j].epf-
1.0/APDdevice.DeviceLayers[j].eps);


   //******************band and scattering
parameters************************


            in_file>>APDdevice.DeviceLayers[j].eg;  //eg=1.424 for GaAs.
          //probably won't use this parameter because we'd always use
          //Ec[i]+Ev[i].

              in_file>>APDdevice.DeviceLayers[j].x; //Al composition,
              //here used for alloy scattering rate
              in_file>>APDdevice.DeviceLayers[j].as;//alloy scattering
              //rate scalling factor--should be relatively small
              //compared with total phonon scattering rate at room
              //temperature.




          in_file>>APDdevice.DeviceLayers[j].rou; //rou=5360 for GaAs
          in_file>>APDdevice.DeviceLayers[j].sv;  //sv=5240 for GaAs

              float cl=APDdevice.DeviceLayers[j].rou
                      *APDdevice.DeviceLayers[j].sv
                          *APDdevice.DeviceLayers[j].sv;

              float z2=4.0;  //L equivallent valley number
              float z0=3.0;  //X equivallent valley number
```

```
      float da=7.0*q;
      float dij=1.0e11*q;
      float deq=1.0e11*q;

      float hwaco300;
      float hwo300;
      float hwij300;

      in_file>>hwaco300;  //hwaco300=0.002 eV for GaAs at 300K
      in_file>>hwo300; //hwo=0.03536 for GaAs at 300K
      in_file>>hwij300; //hwij=0.03 for GaAs//hwij=0.03 for GaAs



APDdevice.DeviceLayers[j].hwaco=0.9*hwaco300+0.1*hwaco300*APDparams.tem/3
00.0;

APDdevice.DeviceLayers[j].hwo=0.9*hwo300+0.1*hwo300*APDparams.tem/300.0;

APDdevice.DeviceLayers[j].hwij=0.9*hwij300+0.1*hwij300*APDparams.tem/300.
0;
      APDdevice.DeviceLayers[j].hwe
      =APDdevice.DeviceLayers[j].hwij;  //Temperature coefficient of
      //phonon energy

      //**************correcponding hole************************
      APDdevice.DeviceLayers[j].afH[1]
     =pow((1.0-APDdevice.DeviceLayers[j].mh[1]),2.0)
     /(APDdevice.DeviceLayers[j].Ev[1]+APDdevice.DeviceLayers[j].Ec[1]);

      APDdevice.DeviceLayers[j].af[1]
     =pow((1.0-APDdevice.DeviceLayers[j].me[1]),2.0)
     /(APDdevice.DeviceLayers[j].Ec[1]+APDdevice.DeviceLayers[j].Ev[1]);

      APDdevice.DeviceLayers[j].af2[1]
    =2.0*APDdevice.DeviceLayers[j].af[1];
      APDdevice.DeviceLayers[j].af4[1]
     =4.0*APDdevice.DeviceLayers[j].af[1];

      //**************correcponding hole************************
      APDdevice.DeviceLayers[j].af2H[1]
    =2.0*APDdevice.DeviceLayers[j].afH[1];
      APDdevice.DeviceLayers[j].af4H[1]
     =4.0*APDdevice.DeviceLayers[j].afH[1];
     //**************correcponding hole************************

      APDdevice.DeviceLayers[j].af[2]
     =pow((1.0-APDdevice.DeviceLayers[j].me[2]),2.0)

/(APDdevice.DeviceLayers[j].Ec[2]+APDdevice.DeviceLayers[j].Ev[1]);

APDdevice.DeviceLayers[j].af2[2]=2.0*APDdevice.DeviceLayers[j].af[2];

APDdevice.DeviceLayers[j].af4[2]=4.0*APDdevice.DeviceLayers[j].af[2];

      APDdevice.DeviceLayers[j].afH[2]
     =pow((1.0-APDdevice.DeviceLayers[j].mh[2]),2.0)
```

```
/(APDdevice.DeviceLayers[j].Ev[2]+APDdevice.DeviceLayers[j].Ec[1]);

APDdevice.DeviceLayers[j].af2H[2]=2.0*APDdevice.DeviceLayers[j].afH[2];

APDdevice.DeviceLayers[j].af4H[2]=4.0*APDdevice.DeviceLayers[j].afH[2];



      APDdevice.DeviceLayers[j].af[0]
     =pow((1.0-APDdevice.DeviceLayers[j].me[0]),2.0)

/(APDdevice.DeviceLayers[j].Ec[0]+APDdevice.DeviceLayers[j].Ev[1]);

APDdevice.DeviceLayers[j].af2[0]=2.0*APDdevice.DeviceLayers[j].af[0];

APDdevice.DeviceLayers[j].af4[0]=4.0*APDdevice.DeviceLayers[j].af[0];


      APDdevice.DeviceLayers[j].afH[0]
     =pow((1.0-APDdevice.DeviceLayers[j].mh[0]),2.0)

/(APDdevice.DeviceLayers[j].Ev[0]+APDdevice.DeviceLayers[j].Ec[1]);

APDdevice.DeviceLayers[j].af2H[0]=2.0*APDdevice.DeviceLayers[j].afH[0];

APDdevice.DeviceLayers[j].af4H[0]=4.0*APDdevice.DeviceLayers[j].afH[0];


      float bktq=bk*APDparams.tem/q;
      float qh=q/h;

      APDdevice.DeviceLayers[j].smh[1]
     =sqrt(2.0*APDdevice.DeviceLayers[j].me[1]*am0)*sqrt(q)/h;
      APDdevice.DeviceLayers[j].smh[2]
     =sqrt(2.0*APDdevice.DeviceLayers[j].me[2]*am0)*sqrt(q)/h;
      APDdevice.DeviceLayers[j].smh[0]
     =sqrt(2.0*APDdevice.DeviceLayers[j].me[0]*am0)*sqrt(q)/h;



      APDdevice.DeviceLayers[j].hhm[1]
     =h/(APDdevice.DeviceLayers[j].me[1]*am0)/qh/2.0;
      APDdevice.DeviceLayers[j].hhm[2]
     =h/(APDdevice.DeviceLayers[j].me[2]*am0)/qh/2.0;
      APDdevice.DeviceLayers[j].hhm[0]
     =h/(APDdevice.DeviceLayers[j].me[0]*am0)/qh/2.0;


      APDdevice.DeviceLayers[j].hm[1]
     =h/(APDdevice.DeviceLayers[j].me[1]*am0);
      APDdevice.DeviceLayers[j].hm[2]
     =h/(APDdevice.DeviceLayers[j].me[2]*am0);
      APDdevice.DeviceLayers[j].hm[0]
     =h/(APDdevice.DeviceLayers[j].me[0]*am0);

      //**************correcponding hole************************
      APDdevice.DeviceLayers[j].smhH[1]
     =sqrt(2.0*APDdevice.DeviceLayers[j].mh[1]*am0)*sqrt(q)/h;
```

```
      APDdevice.DeviceLayers[j].smhH[2]
     =sqrt(2.0*APDdevice.DeviceLayers[j].mh[2]*am0)*sqrt(q)/h;
      APDdevice.DeviceLayers[j].smhH[0]
     =sqrt(2.0*APDdevice.DeviceLayers[j].mh[0]*am0)*sqrt(q)/h;


//cout<<"APDdevice.DeviceLayers[j].smhH"<<APDdevice.DeviceLayers[j].smhH<
<endl;

      APDdevice.DeviceLayers[j].hhmH[1]
     =h/(APDdevice.DeviceLayers[j].mh[1]*am0)/qh/2.0;
      APDdevice.DeviceLayers[j].hhmH[2]
     =h/(APDdevice.DeviceLayers[j].mh[2]*am0)/qh/2.0;
      APDdevice.DeviceLayers[j].hhmH[0]
     =h/(APDdevice.DeviceLayers[j].mh[0]*am0)/qh/2.0;


      APDdevice.DeviceLayers[j].hmH[1]
     =h/(APDdevice.DeviceLayers[j].mh[1]*am0);
      APDdevice.DeviceLayers[j].hmH[2]
     =h/(APDdevice.DeviceLayers[j].mh[2]*am0);
      APDdevice.DeviceLayers[j].hmH[0]
     =h/(APDdevice.DeviceLayers[j].mh[0]*am0);
      //**************correcponding hole************************


      //these are not distinguishable to electrons or holes
      float waco=APDdevice.DeviceLayers[j].hwaco*q/h;
      float wo=APDdevice.DeviceLayers[j].hwo*q/h;
      float wij=APDdevice.DeviceLayers[j].hwij*q/h;
      float we=APDdevice.DeviceLayers[j].hwe*q/h;
      float naco=1.0/(exp(APDdevice.DeviceLayers[j].hwaco/bktq)-1.0);
      float no=1.0/(exp(APDdevice.DeviceLayers[j].hwo/bktq)-1.0);
      float nij=1.0/(exp(APDdevice.DeviceLayers[j].hwij/bktq)-1.0);
      float ne=1.0/(exp(APDdevice.DeviceLayers[j].hwe/bktq)-1.0);
      float
dos1=pow((sqrt(2.0*APDdevice.DeviceLayers[j].me[1]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi;
      //      cout<<"dos1="<<dos1<<endl;
      float
dos2=pow((sqrt(2.0*APDdevice.DeviceLayers[j].me[2]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi;
      float
dos0=pow((sqrt(2.0*APDdevice.DeviceLayers[j].me[0]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi;
      //**************correcponding hole************************
      float
dosH1=pow((sqrt(2.0*APDdevice.DeviceLayers[j].mh[1]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi*APDdevice.DeviceLayers[j].hs;
      float
dosH2=pow((sqrt(2.0*APDdevice.DeviceLayers[j].mh[2]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi*APDdevice.DeviceLayers[j].hs;
      float
dosH0=pow((sqrt(2.0*APDdevice.DeviceLayers[j].mh[0]*am0)*sqrt(q)/h),
                3.0)/4.0/pi/pi*APDdevice.DeviceLayers[j].hs;

     float poe=q/8.0/pi/(APDdevice.DeviceLayers[j].ep*ep0)*q*wo*(no+1.0);
      float poa=poe*no/(1.0+no);
```

```
      float aco=2.0*pi*da/q*da*bktq/h*q/cl
/(bktq/APDdevice.DeviceLayers[j].hwaco)*naco;

      float ope=pi*dij/wij*dij/APDdevice.DeviceLayers[j].rou/q*(nij+1.0);
      float opa=ope*nij/(1.0+nij);
      float eqe=pi*deq/we*deq/APDdevice.DeviceLayers[j].rou/q*(ne+1.0);
      float eqa=eqe*ne/(1.0+ne);

      float cimp=fabs(APDdevice.DeviceLayers[j].Doping);
      if((APDparams.tem<100)&&(cimp>1e23)) //below 100K, for ionized
impurities
      //surrounded by mobile carriers, they recombine; however, note that
in
      //i-regions mobil carriers are depleted and this does not happen
(or
        //maybe the E-field is not strong enough to ionize the impurity?
Does
      //it depend on add voltage first or cool the device first??)
      {
       cimp=cimp*(1.0-1.0/(1.0+2.0*exp(-20.0/APDparams.tem)));
      }
      float qd=sqrt(q*cimp
               /bktq/(APDdevice.DeviceLayers[j].eps*ep0));
      APDdevice.DeviceLayers[j].qd2=qd*qd;
      float bimp=2.0*pi*cimp*q*q/h*q/(APDdevice.DeviceLayers[j].eps*ep0)
                /(APDdevice.DeviceLayers[j].eps*ep0);

 for(ie=1;ie<=iemax;ie++)
    {
  float ei=de*float(ie);
  float sei=sqrt(ei);

  //alloy scattering, prop. to sqrt(E)
  float ef=ei;   //alloy scattering elastic? need to look up...
  //for now just treat it like impurity scattering.....
         APDdevice.DeviceLayers[j].swk[1][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
    *pow(APDdevice.DeviceLayers[j].me[1],float(2.5))*pow(ei,float(0.5));

         APDdevice.DeviceLayers[j].swk[2][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
    *pow(APDdevice.DeviceLayers[j].me[2],float(2.5))*pow(ei,float(0.5));

         APDdevice.DeviceLayers[j].swk[0][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
    *pow(APDdevice.DeviceLayers[j].me[0],float(2.5))*pow(ei,float(0.5));

         APDdevice.DeviceLayers[j].swkH[1][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
    *pow(APDdevice.DeviceLayers[j].mh[1],float(2.5))*pow(ei,float(0.5));
         APDdevice.DeviceLayers[j].swkH[2][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
```

```
    *pow(APDdevice.DeviceLayers[j].mh[2],float(2.5))*pow(ei,float(0.5));
          APDdevice.DeviceLayers[j].swkH[0][0][ie]
    =APDdevice.DeviceLayers[j].as*APDdevice.DeviceLayers[j].x*(1-
APDdevice.DeviceLayers[j].x)
    *pow(APDdevice.DeviceLayers[j].mh[0],float(2.5))*pow(ei,float(0.5));


         //Gamma valley polar optical phonon

  //emission of phonon, within Gamma
  ef=ei-APDdevice.DeviceLayers[j].hwo;
      if(ef>0.0)
        {
        float sef=sqrt(ef);
        float qmax=sef+sei;
        float qmin=sei-sef;
        APDdevice.DeviceLayers[j].swk[1][1][ie]
         =poe*APDdevice.DeviceLayers[j].smh[1]*sei/ei/q*log(qmax/qmin)
          +APDdevice.DeviceLayers[j].swk[1][0][ie];

        //hole
        APDdevice.DeviceLayers[j].swkH[1][1][ie]
         =poe*APDdevice.DeviceLayers[j].smhH[1]*sei/ei/q*log(qmax/qmin)
       +APDdevice.DeviceLayers[j].swkH[1][0][ie];
        APDdevice.DeviceLayers[j].swkH[2][1][ie]
         =poe*APDdevice.DeviceLayers[j].smhH[2]*sei/ei/q*log(qmax/qmin)
       +APDdevice.DeviceLayers[j].swkH[2][0][ie];
        APDdevice.DeviceLayers[j].swkH[0][1][ie]
         =poe*APDdevice.DeviceLayers[j].smhH[0]*sei/ei/q*log(qmax/qmin)
       +APDdevice.DeviceLayers[j].swkH[0][0][ie];
      }
      else
        {

APDdevice.DeviceLayers[j].swk[1][1][ie]=APDdevice.DeviceLayers[j].swk[1][
0][ie];

        //hole, emission of optical phonon, within each band.

APDdevice.DeviceLayers[j].swkH[1][1][ie]=APDdevice.DeviceLayers[j].swkH[1
][0][ie];

APDdevice.DeviceLayers[j].swkH[2][1][ie]=APDdevice.DeviceLayers[j].swkH[2
][0][ie];

APDdevice.DeviceLayers[j].swkH[0][1][ie]=APDdevice.DeviceLayers[j].swkH[0
][0][ie];
        }


      //absorption of optical phonon, within Gamma
      ef=ei+APDdevice.DeviceLayers[j].hwo;
      float sef=sqrt(ef);
      float qmax=sef+sei;
      float qmin=sef-sei;
      APDdevice.DeviceLayers[j].swk[1][2][ie]
     =APDdevice.DeviceLayers[j].swk[1][1][ie]
     +poa*APDdevice.DeviceLayers[j].smh[1]*sei/ei/q*log(qmax/qmin);
```

```
      //hole, absorption of optical phonon, within each band
      APDdevice.DeviceLayers[j].swkH[1][2][ie]
     =APDdevice.DeviceLayers[j].swkH[1][1][ie]
     +poa*APDdevice.DeviceLayers[j].smhH[1]*sei/ei/q*log(qmax/qmin);
      APDdevice.DeviceLayers[j].swkH[2][2][ie]
     =APDdevice.DeviceLayers[j].swkH[2][1][ie]
     +poa*APDdevice.DeviceLayers[j].smhH[2]*sei/ei/q*log(qmax/qmin);
      APDdevice.DeviceLayers[j].swkH[0][2][ie]
     =APDdevice.DeviceLayers[j].swkH[0][1][ie]
     +poa*APDdevice.DeviceLayers[j].smhH[0]*sei/ei/q*log(qmax/qmin);


      //intervalley emission, from 1-2,  Gamma-->L
      //seen from the energy change, intervalley phonon energy is an
average
      // of large k optical and accoustic phonon and is close to hwo.
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[1]
        -APDdevice.DeviceLayers[j].Ec[2];
      if(ef>0.0)
      {
        float sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
        APDdevice.DeviceLayers[j].swk[1][3][ie]
         =APDdevice.DeviceLayers[j].swk[1][2][ie]
         +z2*ope*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);
        //note that using af[2] for the FINAL state (L).
      }
      else
        {
        APDdevice.DeviceLayers[j].swk[1][3][ie]
         =APDdevice.DeviceLayers[j].swk[1][2][ie];
      }


      // HH to LH, emission. 1-->2

      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[1]
        -APDdevice.DeviceLayers[j].Ev[2];
      if(ef>0.0)
      {
        float sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
        APDdevice.DeviceLayers[j].swkH[1][3][ie]
         =APDdevice.DeviceLayers[j].swkH[1][2][ie]
         +ope*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
      } //note that no z2 here because no equivallent valleys for holes
      //ope depends on nij, which intern depends strongly on temperature
      //as hwij is smaller than hwo.
      else
        {
        APDdevice.DeviceLayers[j].swkH[1][3][ie]
         =APDdevice.DeviceLayers[j].swkH[1][2][ie];
      }

      //intervalley emission, from 1-0,  Gamma-->X
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[1]
```

```
        -APDdevice.DeviceLayers[j].Ec[0];
      if(ef>0.0)
      {
        float sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
        APDdevice.DeviceLayers[j].swk[1][4][ie]
         =APDdevice.DeviceLayers[j].swk[1][3][ie]
         +z0*ope*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);
      }
      else
        {
        APDdevice.DeviceLayers[j].swk[1][4][ie]
         =APDdevice.DeviceLayers[j].swk[1][3][ie];
      }


      //hole, emission, from 1-0,  HH --> SO
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[1]
        -APDdevice.DeviceLayers[j].Ev[0];
      if(ef>0.0)
      {
        float sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
        APDdevice.DeviceLayers[j].swkH[1][4][ie]
         =APDdevice.DeviceLayers[j].swkH[1][3][ie]
         +ope*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);
      }
      else
        {
        APDdevice.DeviceLayers[j].swkH[1][4][ie]
         =APDdevice.DeviceLayers[j].swkH[1][3][ie];
      }


      //intervalley absorption, Gamma to L

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[1]
          -APDdevice.DeviceLayers[j].Ec[2];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
        APDdevice.DeviceLayers[j].swk[1][5][ie]
         =APDdevice.DeviceLayers[j].swk[1][4][ie]
         +z2*opa*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[1][5][ie]
         =APDdevice.DeviceLayers[j].swk[1][4][ie];
      }


      //hole, absorption, HH to LH, 1-->2

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[1]
          -APDdevice.DeviceLayers[j].Ev[2];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
```

```
        APDdevice.DeviceLayers[j].swkH[1][5][ie]
         =APDdevice.DeviceLayers[j].swkH[1][4][ie]
         +opa*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[1][5][ie]
         =APDdevice.DeviceLayers[j].swkH[1][4][ie];
      }



      //intervalley absorption, Gamma to X

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[1]
           -APDdevice.DeviceLayers[j].Ec[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
        APDdevice.DeviceLayers[j].swk[1][6][ie]
         =APDdevice.DeviceLayers[j].swk[1][5][ie]
         +z0*opa*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[1][6][ie]
         =APDdevice.DeviceLayers[j].swk[1][5][ie];
      }


      //hole, absorption, HH-->SO,  1-->0

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[1]
           -APDdevice.DeviceLayers[j].Ev[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
        APDdevice.DeviceLayers[j].swkH[1][6][ie]
         =APDdevice.DeviceLayers[j].swkH[1][5][ie]
         +opa*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[1][6][ie]
         =APDdevice.DeviceLayers[j].swkH[1][5][ie];
      }


//acoustic phonon in Gamma Valley, little change in energy
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
      APDdevice.DeviceLayers[j].swk[1][7][ie]
     =APDdevice.DeviceLayers[j].swk[1][6][ie]
     +aco*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].af[1]*ef);


      //heavy hole, acoustic phonon,
      ef=ei;
```

```
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
      APDdevice.DeviceLayers[j].swkH[1][7][ie]
     =APDdevice.DeviceLayers[j].swkH[1][6][ie]
     +aco*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].afH[1]*ef);

      //impurity scattering
      ef=ei;   //elastic, no change in energy
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
      float ak=APDdevice.DeviceLayers[j].smh[1]*sef;
      float qq=APDdevice.DeviceLayers[j].qd2
              *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);
      float
wk=APDparams.impfac*bimp/qq*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].a
f[1]*ef);
      if(wk>1.0e15){wk=1.0e15;}
      //upper limit for impurity scattering to avoid divergence at low
energy


      //the above is for screened ionized impurity. For unscreened
      //impurity (set the criterion to be bimp>10^{23}, even though
strictly the
      //criterion should be that the "position" is in the depletion
region rather
      //than judging from the Layer "j".

      if(cimp<1e23)
      {
        float bmax=1.0/(2.0*pow(cimp,float(0.333)));
          //float
bmaxek=8.0*pi*APDdevice.DeviceLayers[j].eps*ep0*bmax*ef/q;
        //double costhetamin=1.0-2.0/(1.0-bmaxek*bmaxek);
        //the above costhetamin formula from Tomizawa seems problemic.
        //try using different formula from Yang Fu Jia in the following:

        double aruth=q/(4.0*pi*APDdevice.DeviceLayers[j].eps*ep0*ef);
        double a4b=aruth*aruth/4.0/bmax/bmax;
        double costhetamin=(1.0-a4b)/(1.0+a4b);

wk=APDparams.impfac*bimp/2.0*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].
af[1]*ef)/(4.0*ak*ak)/((1.0-costhetamin)*2.0*ak*ak)*(1.0+costhetamin);
        //it seems that this wk increases with ef, in contrast to wk with
screened; in this unscreened case the scat rate does not depend on T.
        //case---the reason, here costhetamin dominates as (1-
costhetamin) increases sharply
        //with ef as thetamin approaches zero.

      }

      //cout<<"j= "<<j<<"ef="<<ef<<"wk="<<wk<<endl;

      APDdevice.DeviceLayers[j].swk[1][8][ie]
      =APDdevice.DeviceLayers[j].swk[1][7][ie]+wk;

      //impurity scattering of hole, in HH band
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
      ak=APDdevice.DeviceLayers[j].smhH[1]*sef;
      qq=APDdevice.DeviceLayers[j].qd2
```

```
              *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);

wk=APDparams.impfac*bimp/qq*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].
afH[1]*ef);
      if(wk>1.0e15){wk=1.0e15;}

      //float temp=APDdevice.DeviceLayers[j].afH;

     APDdevice.DeviceLayers[j].swkH[1][8][ie]
      =APDdevice.DeviceLayers[j].swkH[1][7][ie]+wk;

     //APDdevice.DeviceLayers[j].afH=temp;  //value jumps to
APDdevice.DeviceLayers[j].swkH[6][ie]
     //try to see if this works.


      //L valleys polar optical phonon emission,
      ef=ei-APDdevice.DeviceLayers[j].hwo;
      if(ef>0.0)
      {
        sef=sqrt(ef);
        qmax=sef+sei;
        qmin=sei-sef;
        APDdevice.DeviceLayers[j].swk[2][1][ie]
         =poe*APDdevice.DeviceLayers[j].smh[2]*sei/ei/q*log(qmax/qmin)
         +APDdevice.DeviceLayers[j].swk[2][0][ie];
      }
      else
      {
      APDdevice.DeviceLayers[j].swk[2][1][ie]=APDdevice.DeviceLayers[j].s
wk[2][0][ie];
      }

      //HH polar optical phonon emission,
      ef=ei-APDdevice.DeviceLayers[j].hwo;
      if(ef>0.0)
      {
        sef=sqrt(ef);
        qmax=sef+sei;
        qmin=sei-sef;
        APDdevice.DeviceLayers[j].swkH[2][1][ie]
         =poe*APDdevice.DeviceLayers[j].smhH[2]*sei/ei/q*log(qmax/qmin)
         +APDdevice.DeviceLayers[j].swkH[2][0][ie];
      }
      else
      {
      APDdevice.DeviceLayers[j].swkH[2][1][ie]=APDdevice.DeviceLayers[j].
swkH[2][0][ie];
      }


      //X valleys polar optical phonon emission,
      ef=ei-APDdevice.DeviceLayers[j].hwo;
      if(ef>0.0)
      {
        sef=sqrt(ef);
        qmax=sef+sei;
        qmin=sei-sef;
```

```
        APDdevice.DeviceLayers[j].swk[0][1][ie]
         =poe*APDdevice.DeviceLayers[j].smh[0]*sei/ei/q*log(qmax/qmin)
         +APDdevice.DeviceLayers[j].swk[0][0][ie];
      }
      else
      {
      APDdevice.DeviceLayers[j].swk[0][1][ie]=APDdevice.DeviceLayers[j].s
wk[0][0][ie];
      }

      //SO optical phonon emission,
      ef=ei-APDdevice.DeviceLayers[j].hwo;
      if(ef>0.0)
      {
        sef=sqrt(ef);
        qmax=sef+sei;
        qmin=sei-sef;
        APDdevice.DeviceLayers[j].swkH[0][1][ie]
         =poe*APDdevice.DeviceLayers[j].smhH[0]*sei/ei/q*log(qmax/qmin)
         +APDdevice.DeviceLayers[j].swkH[0][0][ie];
      }
      else
      {
      APDdevice.DeviceLayers[j].swkH[0][1][ie]=APDdevice.DeviceLayers[j].
swkH[0][0][ie];
      }

      //L valleys polar optical phonon absorption,
      ef=ei+APDdevice.DeviceLayers[j].hwo;
      sef=sqrt(ef);
      qmax=sef+sei;
      qmin=sef-sei;
      APDdevice.DeviceLayers[j].swk[2][2][ie]
     =APDdevice.DeviceLayers[j].swk[2][1][ie]
     +poa*APDdevice.DeviceLayers[j].smh[2]*sei/ei/q*log(qmax/qmin);


      //LH polar optical phonon absorption,
      ef=ei+APDdevice.DeviceLayers[j].hwo;
      sef=sqrt(ef);
      qmax=sef+sei;
      qmin=sef-sei;
      APDdevice.DeviceLayers[j].swkH[2][2][ie]
     =APDdevice.DeviceLayers[j].swkH[2][1][ie]
     +poa*APDdevice.DeviceLayers[j].smhH[2]*sei/ei/q*log(qmax/qmin);

      //X valleys polar optical phonon absorption,
      ef=ei+APDdevice.DeviceLayers[j].hwo;
      sef=sqrt(ef);
      qmax=sef+sei;
      qmin=sef-sei;
      APDdevice.DeviceLayers[j].swk[0][2][ie]
     =APDdevice.DeviceLayers[j].swk[0][1][ie]
     +poa*APDdevice.DeviceLayers[j].smh[0]*sei/ei/q*log(qmax/qmin);


      //SO polar optical phonon absorption,
      ef=ei+APDdevice.DeviceLayers[j].hwo;
```

```
      sef=sqrt(ef);
      qmax=sef+sei;
      qmin=sef-sei;
      APDdevice.DeviceLayers[j].swkH[0][2][ie]
     =APDdevice.DeviceLayers[j].swkH[0][1][ie]
     +poa*APDdevice.DeviceLayers[j].smhH[0]*sei/ei/q*log(qmax/qmin);


      //L valley Nonpolar optical phonon emission
      ef=ei-APDdevice.DeviceLayers[j].hwe;
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
        APDdevice.DeviceLayers[j].swk[2][3][ie]
         =APDdevice.DeviceLayers[j].swk[2][2][ie]
         +(z2-
1.0)*eqe*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[2][3][ie]
         =APDdevice.DeviceLayers[j].swk[2][2][ie];
      }

      //LH Nonpolar optical phonon emission
      ef=ei-APDdevice.DeviceLayers[j].hwe;
      if(ef>0.0)
      {
          APDdevice.DeviceLayers[j].swkH[2][3][ie]
         =APDdevice.DeviceLayers[j].swkH[2][2][ie];


      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[2][3][ie]
         =APDdevice.DeviceLayers[j].swkH[2][2][ie];
      }

      //X valley Nonpolar optical phonon emission, equivallent valleys
      ef=ei-APDdevice.DeviceLayers[j].hwe;
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
        APDdevice.DeviceLayers[j].swk[0][3][ie]
         =APDdevice.DeviceLayers[j].swk[0][2][ie]
         +(z0-
1.0)*eqe*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[0][3][ie]
         =APDdevice.DeviceLayers[j].swk[0][2][ie];
      }


      //SO Nonpolar optical phonon emission
      ef=ei-APDdevice.DeviceLayers[j].hwe;
```

```
      if(ef>0.0)
      {
        APDdevice.DeviceLayers[j].swkH[0][3][ie]
         =APDdevice.DeviceLayers[j].swkH[0][2][ie];
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[0][3][ie]
         =APDdevice.DeviceLayers[j].swkH[0][2][ie];
      }


      //L valley hwe phonon absorption, interequivallent valley
scattering
      ef=ei+APDdevice.DeviceLayers[j].hwe;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
      APDdevice.DeviceLayers[j].swk[2][4][ie]
     =APDdevice.DeviceLayers[j].swk[2][3][ie]
     +(z2-1.0)*eqa*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);


      //LH Nonpolar optical phonon absorption
      //   ef=ei+APDdevice.DeviceLayers[j].hwe;
      //  sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
      APDdevice.DeviceLayers[j].swkH[2][4][ie]
      =APDdevice.DeviceLayers[j].swkH[2][3][ie];
      //   +eqa*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
      //no (z2-1.0) term because there are no equivallent valleys for
holes


      //X valley Nonpolar optical phonon absorption
      ef=ei+APDdevice.DeviceLayers[j].hwe;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
      APDdevice.DeviceLayers[j].swk[0][4][ie]
     =APDdevice.DeviceLayers[j].swk[0][3][ie]
     +(z0-1.0)*eqa*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);


      //SO Nonpolar optical phonon absorption
      // ef=ei+APDdevice.DeviceLayers[j].hwe;
      //sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
      APDdevice.DeviceLayers[j].swkH[0][4][ie]
      =APDdevice.DeviceLayers[j].swkH[0][3][ie];
      //+eqa*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);



      //L --> Gamma, emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[2]
        -APDdevice.DeviceLayers[j].Ec[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
        APDdevice.DeviceLayers[j].swk[2][5][ie]
         =APDdevice.DeviceLayers[j].swk[2][4][ie]
         +ope*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].af[1]*ef);
```

```
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[2][5][ie]
         =APDdevice.DeviceLayers[j].swk[2][4][ie];
      }


      //hole, 2--1, emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[2]
        -APDdevice.DeviceLayers[j].Ev[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
        APDdevice.DeviceLayers[j].swkH[2][5][ie]
         =APDdevice.DeviceLayers[j].swkH[2][4][ie]
         +ope*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].afH[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[2][5][ie]
         =APDdevice.DeviceLayers[j].swkH[2][4][ie];
      }


      //L --> Gamma, absorption

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[2]
        -APDdevice.DeviceLayers[j].Ec[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
        APDdevice.DeviceLayers[j].swk[2][6][ie]
         =APDdevice.DeviceLayers[j].swk[2][5][ie]
         +ope*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].af[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[2][6][ie]
         =APDdevice.DeviceLayers[j].swk[2][5][ie];
      }


      //hole, 2--1,  absorption

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[2]
        -APDdevice.DeviceLayers[j].Ev[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
        APDdevice.DeviceLayers[j].swkH[2][6][ie]
         =APDdevice.DeviceLayers[j].swkH[2][5][ie]
         +ope*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].afH[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[2][6][ie]
```

```
         =APDdevice.DeviceLayers[j].swkH[2][5][ie];
      }

      //X --> Gamma, emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[0]
        -APDdevice.DeviceLayers[j].Ec[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
        APDdevice.DeviceLayers[j].swk[0][5][ie]
         =APDdevice.DeviceLayers[j].swk[0][4][ie]
         +ope*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].af[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[0][5][ie]
         =APDdevice.DeviceLayers[j].swk[0][4][ie];
      }

      //H, 0--1,  emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[0]
        -APDdevice.DeviceLayers[j].Ev[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
        APDdevice.DeviceLayers[j].swkH[0][5][ie]
         =APDdevice.DeviceLayers[j].swkH[0][4][ie]
         +ope*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].afH[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[0][5][ie]
         =APDdevice.DeviceLayers[j].swkH[0][4][ie];
      }

      //X --> Gamma, absorption

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[0]
        -APDdevice.DeviceLayers[j].Ec[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[1]*ef));
        APDdevice.DeviceLayers[j].swk[0][6][ie]
         =APDdevice.DeviceLayers[j].swk[0][5][ie]
         +ope*sef*dos1*(1.0+2.0*APDdevice.DeviceLayers[j].af[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[0][6][ie]
         =APDdevice.DeviceLayers[j].swk[0][5][ie];
      }


      //H, 0--1,  absorption

ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[0]
```

```
        -APDdevice.DeviceLayers[j].Ev[1];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
        APDdevice.DeviceLayers[j].swkH[0][6][ie]
         =APDdevice.DeviceLayers[j].swkH[0][5][ie]
         +ope*sef*dosH1*(1.0+2.0*APDdevice.DeviceLayers[j].afH[1]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[0][6][ie]
         =APDdevice.DeviceLayers[j].swkH[0][5][ie];
      }



      //L --> X, emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[2]
        -APDdevice.DeviceLayers[j].Ec[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
        APDdevice.DeviceLayers[j].swk[2][7][ie]
         =APDdevice.DeviceLayers[j].swk[2][6][ie]
         +z0*ope*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[2][7][ie]
         =APDdevice.DeviceLayers[j].swk[2][6][ie];
      }


      //2--0, hole,  emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[2]
        -APDdevice.DeviceLayers[j].Ev[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
        APDdevice.DeviceLayers[j].swkH[2][7][ie]
         =APDdevice.DeviceLayers[j].swkH[2][6][ie]
         +ope*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[2][7][ie]
         =APDdevice.DeviceLayers[j].swkH[2][6][ie];
      }


      //X --> L, emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[0]
        -APDdevice.DeviceLayers[j].Ec[2];
      if(ef>0.0)
      {
```

```
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
        APDdevice.DeviceLayers[j].swk[0][7][ie]
         =APDdevice.DeviceLayers[j].swk[0][6][ie]
         +z2*ope*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[0][7][ie]
         =APDdevice.DeviceLayers[j].swk[0][6][ie];
      }

      //0--2, hole,  emission
      ef=ei-
APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[0]
        -APDdevice.DeviceLayers[j].Ev[2];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
        APDdevice.DeviceLayers[j].swkH[0][7][ie]
         =APDdevice.DeviceLayers[j].swkH[0][6][ie]
         +ope*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[0][7][ie]
         =APDdevice.DeviceLayers[j].swkH[0][6][ie];
      }

      // L --> X, absorption
      ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[2
]
          -APDdevice.DeviceLayers[j].Ec[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
        APDdevice.DeviceLayers[j].swk[2][8][ie]
         =APDdevice.DeviceLayers[j].swk[2][7][ie]
         +z0*opa*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[2][8][ie]
         =APDdevice.DeviceLayers[j].swk[2][7][ie];
      }

      // hole, 2--0,  absorption
      ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[2
]
          -APDdevice.DeviceLayers[j].Ev[0];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[1]*ef));
        APDdevice.DeviceLayers[j].swkH[2][8][ie]
         =APDdevice.DeviceLayers[j].swkH[2][7][ie]
         +opa*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);
      }
      else
      {
```

```
        APDdevice.DeviceLayers[j].swkH[2][8][ie]
         =APDdevice.DeviceLayers[j].swkH[2][7][ie];
      }

      // X --> L, absorption
      ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ec[0
]
          -APDdevice.DeviceLayers[j].Ec[2];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
        APDdevice.DeviceLayers[j].swk[0][8][ie]
         =APDdevice.DeviceLayers[j].swk[0][7][ie]
         +z2*opa*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swk[0][8][ie]
         =APDdevice.DeviceLayers[j].swk[0][7][ie];
      }



      // 0--2, hole, absorption
      ef=ei+APDdevice.DeviceLayers[j].hwij+APDdevice.DeviceLayers[j].Ev[0
]
          -APDdevice.DeviceLayers[j].Ev[2];
      if(ef>0.0)
      {
        sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
        APDdevice.DeviceLayers[j].swk[0][8][ie]
         =APDdevice.DeviceLayers[j].swk[0][7][ie]
         +opa*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
      }
      else
      {
        APDdevice.DeviceLayers[j].swkH[0][8][ie]
         =APDdevice.DeviceLayers[j].swkH[0][7][ie];
      }


////////////////////////////////////////////////////////////////////////
/////////////////

      //acoustic phonon  L valley
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
      APDdevice.DeviceLayers[j].swk[2][9][ie]
     =APDdevice.DeviceLayers[j].swk[2][8][ie]
     +aco*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].af[2]*ef);


      //acoustic phonon  hole, 2
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
      APDdevice.DeviceLayers[j].swkH[2][9][ie]
     =APDdevice.DeviceLayers[j].swkH[2][8][ie]
     +aco*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].afH[2]*ef);
```

```
      //acoustic phonon  X valley
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
      APDdevice.DeviceLayers[j].swk[0][9][ie]
     =APDdevice.DeviceLayers[j].swk[0][8][ie]
     +aco*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].af[0]*ef);


      //acoustic phonon  hole, 0,
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
      APDdevice.DeviceLayers[j].swkH[0][9][ie]
     =APDdevice.DeviceLayers[j].swkH[0][8][ie]
     +aco*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].afH[0]*ef);




      //impurity scattering L valley
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[2]*ef));
      ak=APDdevice.DeviceLayers[j].smh[2]*sef;
      qq=APDdevice.DeviceLayers[j].qd2
        *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);

wk=APDparams.impfac*bimp/qq*sef*dos2*(1.0+2.0*APDdevice.DeviceLayers[j].a
f[2]*ef);
      if(wk>1.0e15){wk=1.0e15;}
      APDdevice.DeviceLayers[j].swk[2][10][ie]
     =APDdevice.DeviceLayers[j].swk[2][9][ie]+wk;


      //impurity scattering LH
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[2]*ef));
      ak=APDdevice.DeviceLayers[j].smhH[2]*sef;
      qq=APDdevice.DeviceLayers[j].qd2
        *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);

wk=APDparams.impfac*bimp/qq*sef*dosH2*(1.0+2.0*APDdevice.DeviceLayers[j].
afH[2]*ef);
      if(wk>1.0e15){wk=1.0e15;}
      APDdevice.DeviceLayers[j].swkH[2][10][ie]
     =APDdevice.DeviceLayers[j].swkH[2][9][ie]+wk;


      //impurity scattering X valley
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].af[0]*ef));
      ak=APDdevice.DeviceLayers[j].smh[0]*sef;
      qq=APDdevice.DeviceLayers[j].qd2
        *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);

wk=APDparams.impfac*bimp/qq*sef*dos0*(1.0+2.0*APDdevice.DeviceLayers[j].a
f[0]*ef);
      if(wk>1.0e15){wk=1.0e15;}
      APDdevice.DeviceLayers[j].swk[0][10][ie]
```

```
     =APDdevice.DeviceLayers[j].swk[0][9][ie]+wk;

      //impurity scattering SO, 0
      ef=ei;
      sef=sqrt(ef*(1.0+APDdevice.DeviceLayers[j].afH[0]*ef));
      ak=APDdevice.DeviceLayers[j].smhH[0]*sef;
      qq=APDdevice.DeviceLayers[j].qd2
        *(4.0*ak*ak+APDdevice.DeviceLayers[j].qd2);

wk=APDparams.impfac*bimp/qq*sef*dosH0*(1.0+2.0*APDdevice.DeviceLayers[j].
afH[0]*ef);
      if(wk>1.0e15){wk=1.0e15;}
      APDdevice.DeviceLayers[j].swkH[0][10][ie]
     =APDdevice.DeviceLayers[j].swkH[0][9][ie]+wk;

  //Impact Ionization Rate, labeled as the last scattering mechanism.


if(ei>(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[1]+AP
Ddevice.DeviceLayers[j].Ev[1]))
    {
  APDdevice.DeviceLayers[j].swk[1][9][ie] = APDdevice.DeviceLayers[j].Ae

*pow((ei/(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[1]
+APDdevice.DeviceLayers[j].Ev[1])-
1.0),double(APDdevice.DeviceLayers[j].ce))
    +APDdevice.DeviceLayers[j].swk[1][8][ie];
  //note that swk is accumulative....

  //the rate depends only on energy, not the valleys the carriers are in?
  //now improve the treatment here: in each valley once the carrier
energy is
  //above the Eacte+Eg, then impact ionization occurs.
    }
  else
    {
    APDdevice.DeviceLayers[j].swk[1][9][ie]
   = APDdevice.DeviceLayers[j].swk[1][8][ie];
    }



if(ei>(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[1]+AP
Ddevice.DeviceLayers[j].Ev[1]))
    {
  APDdevice.DeviceLayers[j].swk[2][11][ie] = APDdevice.DeviceLayers[j].Ae

*pow((ei/(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[2]
+APDdevice.DeviceLayers[j].Ev[1])-
1.0),double(APDdevice.DeviceLayers[j].ce))
    +APDdevice.DeviceLayers[j].swk[2][10][ie];
    }
     else
  {
    APDdevice.DeviceLayers[j].swk[2][11][ie]
   = APDdevice.DeviceLayers[j].swk[2][10][ie];
  }
```

```
if(ei>(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[1]+AP
Ddevice.DeviceLayers[j].Ev[1]))
    {
  APDdevice.DeviceLayers[j].swk[0][11][ie] = APDdevice.DeviceLayers[j].Ae

*pow((ei/(APDdevice.DeviceLayers[j].Eacte+APDdevice.DeviceLayers[j].Ec[0]
+APDdevice.DeviceLayers[j].Ev[1])-
1.0),double(APDdevice.DeviceLayers[j].ce))
    +APDdevice.DeviceLayers[j].swk[0][10][ie];
    }
  else
    {
    APDdevice.DeviceLayers[j].swk[0][11][ie]
   = APDdevice.DeviceLayers[j].swk[0][10][ie];
    }



if(ei>(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]+AP
Ddevice.DeviceLayers[j].Ev[1]))
    {
  APDdevice.DeviceLayers[j].swkH[1][9][ie] = APDdevice.DeviceLayers[j].Ah

*pow((ei/(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]
+APDdevice.DeviceLayers[j].Ev[1])-
1.0),double(APDdevice.DeviceLayers[j].ch))
    +APDdevice.DeviceLayers[j].swkH[1][8][ie];
    }
  else
    {
    APDdevice.DeviceLayers[j].swkH[1][9][ie]
   = APDdevice.DeviceLayers[j].swkH[1][8][ie];
    }



if(ei>(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]+AP
Ddevice.DeviceLayers[j].Ev[2]))
    {
  APDdevice.DeviceLayers[j].swkH[2][11][ie] =
APDdevice.DeviceLayers[j].Ah

*pow((ei/(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]
+APDdevice.DeviceLayers[j].Ev[2])-
1.0),double(APDdevice.DeviceLayers[j].ch))
    +APDdevice.DeviceLayers[j].swkH[2][10][ie];
    }
     else
  {
    APDdevice.DeviceLayers[j].swkH[2][11][ie]
   = APDdevice.DeviceLayers[j].swkH[2][10][ie];
  }



if(ei>(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]+AP
```

```
Ddevice.DeviceLayers[j].Ev[0]))
    {
  APDdevice.DeviceLayers[j].swkH[0][11][ie] =
APDdevice.DeviceLayers[j].Ah

*pow((ei/(APDdevice.DeviceLayers[j].Eacth+APDdevice.DeviceLayers[j].Ec[1]
+APDdevice.DeviceLayers[j].Ev[0])-
1.0),double(APDdevice.DeviceLayers[j].ch))
    +APDdevice.DeviceLayers[j].swkH[0][10][ie];
    }
  else
    {
    APDdevice.DeviceLayers[j].swkH[0][11][ie]
   = APDdevice.DeviceLayers[j].swkH[0][10][ie];
    }


    }//matches for....


  //total scattering rate

  APDdevice.DeviceLayers[j].gm=APDdevice.DeviceLayers[j].swk[1][9][1];

  APDdevice.DeviceLayers[j].gmH=APDdevice.DeviceLayers[j].swkH[1][7][1];

      for(ie=1;ie<=iemax;ie++)
{

if(APDdevice.DeviceLayers[j].swk[1][9][ie]>APDdevice.DeviceLayers[j].gm)

{APDdevice.DeviceLayers[j].gm=APDdevice.DeviceLayers[j].swk[1][9][ie];}


if(APDdevice.DeviceLayers[j].swkH[1][9][ie]>APDdevice.DeviceLayers[j].gmH
)

{APDdevice.DeviceLayers[j].gmH=APDdevice.DeviceLayers[j].swkH[1][9][ie];}


if(APDdevice.DeviceLayers[j].swk[2][11][ie]>APDdevice.DeviceLayers[j].gm)

{APDdevice.DeviceLayers[j].gm=APDdevice.DeviceLayers[j].swk[2][11][ie];}


if(APDdevice.DeviceLayers[j].swkH[2][11][ie]>APDdevice.DeviceLayers[j].gm
H)

{APDdevice.DeviceLayers[j].gmH=APDdevice.DeviceLayers[j].swkH[2][11][ie];
}


if(APDdevice.DeviceLayers[j].swk[0][11][ie]>APDdevice.DeviceLayers[j].gm)

{APDdevice.DeviceLayers[j].gm=APDdevice.DeviceLayers[j].swk[0][11][ie];}


if(APDdevice.DeviceLayers[j].swkH[0][11][ie]>APDdevice.DeviceLayers[j].gm
```

```
H)

{APDdevice.DeviceLayers[j].gmH=APDdevice.DeviceLayers[j].swkH[0][11][ie];
}

}

      if(j==2)
      {
      ofstream swk1_out_file;
      swk1_out_file.open(APDparams.swk1filename);

      ofstream swkH_out_file;
      swkH_out_file.open(APDparams.swkHfilename);

 for(ie=1;ie<=iemax;ie++)
  {

   swk1_out_file<<ie*de;
    swkH_out_file<<ie*de;

    for(i=0;i<=9;i++)
      {
      swk1_out_file<<"  "<<APDdevice.DeviceLayers[j].swk[1][i][ie];
      swkH_out_file<<"  "<<APDdevice.DeviceLayers[j].swkH[1][i][ie];
      }
    swk1_out_file<<endl;
    swkH_out_file<<endl;
  }

 swk1_out_file.close();
 swkH_out_file.close();


      ofstream swk0_out_file;
      swk0_out_file.open(APDparams.swk0filename);

      ofstream swk2_out_file;
      swk2_out_file.open(APDparams.swk2filename);

 for(ie=1;ie<=iemax;ie++)
  {
    swk0_out_file<<ie*de;
    swk2_out_file<<ie*de;


    for(i=0;i<=11;i++)
      {

      swk2_out_file<<"  "<<APDdevice.DeviceLayers[j].swk[2][i][ie];
      swk0_out_file<<"  "<<APDdevice.DeviceLayers[j].swk[0][i][ie];
      }

    swk0_out_file<<endl;
    swk2_out_file<<endl;

  }
 swk0_out_file.close();
```

```
 swk2_out_file.close();
      }

 for(ie=1;ie<=iemax;ie++)
  {

    for(i=0;i<=9;i++)
      {

      APDdevice.DeviceLayers[j].swk[1][i][ie]
       =APDdevice.DeviceLayers[j].swk[1][i][ie]
       /APDdevice.DeviceLayers[j].gm;


      APDdevice.DeviceLayers[j].swkH[1][i][ie]
       =APDdevice.DeviceLayers[j].swkH[1][i][ie]
       /APDdevice.DeviceLayers[j].gmH;
            }
  }

 for(ie=1;ie<=iemax;ie++)
  {

    for(i=0;i<=11;i++)
      {
      APDdevice.DeviceLayers[j].swk[2][i][ie]
       =APDdevice.DeviceLayers[j].swk[2][i][ie]
       /APDdevice.DeviceLayers[j].gm;

      APDdevice.DeviceLayers[j].swkH[2][i][ie]
       =APDdevice.DeviceLayers[j].swkH[2][i][ie]
       /APDdevice.DeviceLayers[j].gmH;

      APDdevice.DeviceLayers[j].swk[0][i][ie]
       =APDdevice.DeviceLayers[j].swk[0][i][ie]
       /APDdevice.DeviceLayers[j].gm;

      APDdevice.DeviceLayers[j].swkH[0][i][ie]
       =APDdevice.DeviceLayers[j].swkH[0][i][ie]
       /APDdevice.DeviceLayers[j].gmH;
       }

  }

//************************end band and scattering
parameters*****************************//

          j++;
          in_file>>APDdevice.DeviceLayers[j].Material;
          APDdevice.NumLayers=j;

        }//end while load structure
      }// end if
    }//end do-while
    while (strncmp(param_name,"end",3)!=0);
         in_file.close();

         APDdevice.TotalThickness=0;
```

```
          for(j=1;j<=APDdevice.NumLayers;j++){

          APDdevice.TotalThickness+=APDdevice.DeviceLayers[j].Thickness;
          APDdevice.acu_thickness[j] = APDdevice.TotalThickness;


          }//end for

          //put layer information into arrays, e.g.,
//using PositionIndex j rather than
//layer # to label the paprameters in each layer.

          dx=APDdevice.TotalThickness/NUMX;
          int layer_index;
          for(j=0;j<=NUMX;j++)
            {
           for(layer_index = 1;(j*dx-
APDdevice.acu_thickness[layer_index])>0;layer_index++);
           APDdevice.index[j] = layer_index;


          }//end for
          APDdevice.Vbi=APDparams.Vbi;

          //prevent overwriting end of arrays
          if(APDparams.max_steps>=STEP_LIMIT)
          {APDparams.max_steps=STEP_LIMIT-1;}

          //initialize the parameters that need initializing
          APDdevice.EFMax=1e7; //a guess

          return;
}//end LoadParams
```

```
#include "apd.h"

float FindJunction(device& MyDevice)
  {
    float JunctionPosition=MyDevice.DeviceLayers[1].Thickness;
    int oldsign,newsign;

    if (MyDevice.DeviceLayers[1].Doping>0) {oldsign=1;}
    else                                   {oldsign=-1;}

    for(int i=2; i<=MyDevice.NumLayers; i++) //starting point of i
changed
      {
      if (MyDevice.DeviceLayers[i].Doping>0) {newsign=1;}
      else {newsign=-1;}
      if (newsign==oldsign)
        {JunctionPosition+=MyDevice.DeviceLayers[i].Thickness;}
      else
        {i=MyDevice.NumLayers+1;}  //leave the loop
      }//end for
    return JunctionPosition;
  }
```

BiasLoop.C:

```
#include "apd.h"

void BiasLoop(device& APDdevice, params& APDparams, stats& APDresults,
              electron* NewElectron, hole* NewHole,float
JunctionLocation1)
{

      int i=0,j=0;
      int timing_steps = 0;
      APDresults.total_count=0;
      APDresults.total_sq_count=0;
      float dx=APDdevice.TotalThickness/float(NUMX);
      int JunctionIndex = int(JunctionLocation1/dx);

    APDresults.Ne = APDparams.I0;
    APDresults.Nh = APDparams.I0;

    InitArrays(APDresults);  //initialize loop variables
    for (j=0;j<=NUMX;j++) {APDdevice.charge_density[j] = 0;}
    SetBias(APDdevice, APDparams, APDresults, JunctionLocation1);

    timing_steps = int(APDparams.max_time/APDparams.delta_t);


cout<<"\n#####################################################################
#########################################\n"<<endl;
    cout<<"                    # Program Start:\n"<<endl;
    cout<<"                    # device_bias =
"<<APDparams.Vapp<<"\n"<<endl;
    cout<<"                    # Total_charge =
"<<APDparams.I0<<"\n"<<endl;
    cout<<"                    # Quasi_charge =
"<<APDparams.quasi_charge<<"\n"<<endl;
    if (APDparams.mode==2){cout<<"                    # Device work at DC
mode\n"<<endl;}
    if (APDparams.mode==1){cout<<"                    # Device work at
pulse mode\n"<<endl;}
    APDparams.temp_time = APDparams.delta_t;
    FirstCarrierLoop(APDdevice, APDparams, APDresults, NewElectron,
NewHole);

    for(i=2;i<=timing_steps;i++){ //cout<<"# completed
"<<int(i*APDparams.delta_t/APDparams.max_time*100)<<"%\r";
                                  SetBias(APDdevice, APDparams,
APDresults, JunctionLocation1);
                                  for (j=0;j<=NUMX;j++)
{APDdevice.charge_density[j] = 0;}
                                  APDparams.temp_time =
i*APDparams.delta_t;
                                  CarrierLoop(APDdevice, APDparams,
APDresults, NewElectron, NewHole);
                                  }

      for (j=0;j<APDresults.Ne;j++) {APDresults.total_count =
APDresults.total_count + NewElectron[j].count;}
      int* parent_sum = (int*)malloc((APDparams.I0+10)*sizeof(int));
      for (j=0;j<APDparams.I0;j++) {parent_sum[j] = 0;}
      for (j=0;j<APDresults.Ne;j++)
```

```
{parent_sum[NewElectron[j].parent]+=NewElectron[j].count;}
      for (j=0;j<APDparams.I0;j++) {APDresults.total_sq_count =
APDresults.total_sq_count + parent_sum[j]*parent_sum[j];}
      APDresults.gain[0]=
float(APDresults.total_count)/(float(APDparams.I0));
      APDresults.sq_gain[0]=
float(APDresults.total_sq_count)/(float(APDparams.I0));
      ofstream distribution("gain_distribution.txt");
      for(i=0;i<APDparams.I0;i++) {distribution<<i<<"
"<<parent_sum[i]<<endl;}

      cout<<"\n                    # calculation completed:"<<endl;
      cout<<"                      # Bias                  Gain
      F(M)"<<endl;
      cout<<"                      # "<<APDresults.Vapp[0]<<"          "
          <<APDresults.gain[0]<<"         "
          <<APDresults.sq_gain[0]/
        APDresults.gain[0]/APDresults.gain[0]<<endl;

      ofstream current("current.txt");
      for(i=0;i<NUM_BIN;i++) {current<<i*APDparams.max_time/NUM_BIN<<"
      "<<APDresults.Current[i]<<endl;}
      return;
} //end Bias Loop
```

InitArrays.C

```
#include "apd.h"

/***********
Initialize many of the arrays
***********/
```

```
void InitArrays(stats& APDresults)
  {
    int i=0;

    for(i=0;i<=NUM_BIN;i++)
      {
      APDresults.Current[i]=0.0;
        APDresults.impacte[i]=0;
      APDresults.impacth[i]=0;
      }
    for(i=0;i<=MAX_COUNT;i++)
    {APDresults.GainDist[i]=0;}
    for(i=0;i<=NUMX;i++)
    {
      APDresults.ImpactDist_h[i]=0;
      APDresults.ImpactDist_e[i]=0;
      APDresults.AbsDist[i]=0;

      APDresults.EnergyDist_e[i]=0.0;
      APDresults.EnergyDist_h[i]=0.0;
      APDresults.VelocityDist_e[i]=0.0;
      APDresults.VelocityDist_h[i]=0.0;
      APDresults.CountDist_e[i]=1.0;
      APDresults.CountDist_h[i]=1.0;

      APDresults.LDist_e[i]=0;
      APDresults.GDist_e[i]=0;
      APDresults.LDist_h[i]=0;
      APDresults.GDist_h[i]=0;

    }
    return;
  }
```

SetBias.C

```
#include "apd.h"

  /***************************
   Sets the bias for the device
   This requires detemining the electric field profile and
   tabulating it in the EFTable
```

```
  ***************************/

void SetBias(device& MyDevice, params& APDparams, stats& APDresults,
float JunctionLocation1)
 {
    float Error=1.00;
    const float TOLERANCE=0.001;  //temporarily increase the tolerance to
0.001.
    double resistance = 50;
    double capacitance = 0.2e-12;
    double impedance;
    impedance =
resistance/sqrt(1+(4*3.1416*APDparams.frequency*resistance*capacitance)*(
4*3.1416*APDparams.frequency*resistance*capacitance));
    double delay_time;
    double current_time;
    double period = 1/APDparams.frequency;
    //Get the best estimate of the peak EField
    int N1,N2;
    delay_time = 3.1416-
atan(4*3.1416*resistance*capacitance*APDparams.frequency);
    current_time = APDparams.temp_time - delay_time/(2*3.1416)*period;
    N1 = int(APDparams.temp_time/(APDparams.max_time/NUM_BIN));
    N2 = int(current_time/(APDparams.max_time/NUM_BIN));

    if(N2<0) N2 = 0;
    if(N1>0) N1 = N1 - 1;

    float DesiredBias;
    APDresults.AC_bias[N1] = APDparams.Vapp + 0*(APDresults.Current[N2]-
APDparams.DC_current)*impedance;
    DesiredBias = APDresults.AC_bias[N1];
    while (fabs(Error)>TOLERANCE)
      {
      Error=CalcBias(MyDevice, JunctionLocation1)-DesiredBias;

      MyDevice.EFMax-=(Error/MyDevice.TotalThickness);
      } //end while
    MyDevice.Vapp = CalcBias(MyDevice, JunctionLocation1);

    //Calculate the table of E-fields
    BuildEFTable(MyDevice,JunctionLocation1);

    return;
  }  //end of Set Bias
```

BuildEFTable.C

```
#include "apd.h"

/***The algorithm starts at the junction and calculates to both ends of
the
device from there. Uses the value of EFMax set in other routines
******************/
```

```
void BuildEFTable(device& MyDevice, float JunctionLocation1)
  {

    int JunctionIndex1;
    int i,j;
    float dx=MyDevice.TotalThickness/float(NUMX);
    MyDevice.DepletionStart=0.0;
    MyDevice.DepletionEnd=0.0;

    JunctionIndex1 = int(JunctionLocation1/dx);


    for(i=0;i<=JunctionIndex1;i++)
      {MyDevice.EFTable[i]=0.0;}

    MyDevice.EFTable[JunctionIndex1]=MyDevice.EFMax;

    //build table forward from junction
    for(int i=JunctionIndex1+1; i<=NUMX; i++)
      {
      MyDevice.EFTable[i]=((q*dx)/(eps0*MyDevice.DeviceLayers[MyDevice.in
dex[i]].eps))*
        (MyDevice.DeviceLayers[MyDevice.index[i]].Doping -
MyDevice.charge_density[i])+MyDevice.EFTable[(i-1)];


      if (MyDevice.EFTable[i]<1e6)
        {MyDevice.EFTable[i]=1e6;}
      else
      {MyDevice.DepletionEnd=i*dx;}   //note end of space charge region
      } //end Make Table forwards from junction
    for (int j=JunctionIndex1-1; j>=0; j--)
       { //build table backwards from junction
      MyDevice.EFTable[j]=MyDevice.EFTable[(j+1)]-
((q*dx)/(eps0*MyDevice.DeviceLayers[MyDevice.index[j]].eps))*(MyDevice.De
viceLayers[MyDevice.index[j]].Doping - MyDevice.charge_density[j]);
      if (MyDevice.EFTable[j]<1e6)
        {MyDevice.EFTable[j]=1e6;}
      else{MyDevice.DepletionStart=j*dx;}  //note start of depletion
region
       }  //end calculate EF backwards from junction


}//end BuildEFTable
```

CalcBias.C

```
#include "apd.h"

/*********************************
Calculate the Bias accross the device given the set Max Efield
The table of E-field values needs to be tabulated since the maximum
E-field may have changed.
*********************************/
```

```
float CalcBias(device& MyDevice, float JunctionLocation1)
  {
    float dx=MyDevice.TotalThickness/(float(NUMX));

    MyDevice.Vapp=0.0;
    MyDevice.Potential[0]=0.0;
    BuildEFTable(MyDevice,JunctionLocation1);   //build a table of
EFields based on EFMax set elsewhere
    for(int i=1; i<NUMX; i++)
      {MyDevice.Potential[i]=MyDevice.Potential[i-
1]+MyDevice.EFTable[i]*dx; }
    MyDevice.Vapp=MyDevice.Potential[NUMX-1];
    //    cout<<"In CalcBias, MyDevice.Vapp="<<MyDevice.Vapp<<endl;

    return MyDevice.Vapp-=MyDevice.Vbi;  //There is a build in bias to
consider
  }//end CalcBias
```

GetParams.C

```
#include "apd.h"
/*********************
Steps through the structure until the layer containing
the desired position is found, and returns the doping in that layer.
**********************/

layer GetParams(device& MyDevice, float position)
  {
    float CurrentThickness=0.0;
```

```
    int i;
      //CurrentLayer: i
    for (i=1;(position>CurrentThickness)&&(i<=MyDevice.NumLayers);i++)
      {CurrentThickness+=MyDevice.DeviceLayers[i].Thickness;} //end for
    if (i<=1){ //change i<=0 to i<=1 to accormodate the
      //problem that MyDevice.DeviceLayers[0].Thickness always jumps to
0.
      return MyDevice.DeviceLayers[1];}
    else{
      return MyDevice.DeviceLayers[i-1];}
  }

/****************
Finds a position in the device where there is a junction
****************/

///The 2nd time finding junction (from BuildEFTable) got problems. seems
// related to the problem of APDdevice.DeviceLayers[0].Thickness jumps
back to 0.
```

FirstCarrierLoop.C

```
#include "apd.h"

void FirstCarrierLoop(device& APDdevice, params& APDparams,
    stats& APDresults, electron* NewElectron, hole* NewHole)
{
  float position = 0.0;
  float* ft = (float*)malloc((APDparams.I0+100)*sizeof(float));
  float time=0.0;      //time of event
  double p=1.0/e;      //poisson probability
  int i=0,j=0;
  double randomnumber, randomnumber1;
```

```
  float dx=APDdevice.TotalThickness/float(NUMX);
  //int count=0;       //number of carriers collected
  double device_area = pi*(device_diameter/2)*(device_diameter/2);
  double device_volume = dx*device_area;
  double bin,sum=0,A=0;
  int temp_j;
  int carrier_dis[NUM_BIN+1];
  bin = APDparams.max_time/NUM_BIN;
  for(i=1;i<=NUM_BIN;i++){sum = sum +
(1+sin(2*3.1416*APDparams.frequency*i/NUM_BIN*APDparams.max_time));}
  A = APDparams.I0/sum;
  for(i=1;i<=NUM_BIN;i++){carrier_dis[i] =
int(A*(1+sin(2*3.1416*APDparams.frequency*i/NUM_BIN*APDparams.max_time)))
;}
  ofstream carrier_distri("distribution.txt");
  for(i=1;i<=NUM_BIN;i++){carrier_distri<<carrier_dis[i]<<endl;}
  temp_j = 1;

  for(i=1;i<=NUM_BIN;i++){
        for(j = temp_j;j<=temp_j+carrier_dis[i]-1;j++){ft[j] = i*bin;}
        temp_j = j;
  }
  ofstream time_pointer("time_distribution.txt");
  for(i=1;i<=APDparams.I0;i++)time_pointer<<ft[i]<<endl;


  for(i=0;(i<APDparams.I0);i++)
    {  //loop through many carriers

            time=0;
            NewElectron[i].count=0;
            NewHole[i].count=0;
            NewElectron[i].parent=i;
            NewHole[i].parent=i;
            NewElectron[i].ft=ft[i];
            if(APDparams.mode==1){NewElectron[i].ft=0;}
            NewElectron[i].position=AbsorbPhoton(APDdevice, APDresults);
            NewHole[i].position=NewElectron[i].position;
            float dx=APDdevice.TotalThickness/float(NUMX);
            int PosIndex=int(NewElectron[i].position/dx);

            float bktq=bk*APDparams.tem/q;


if(NewElectron[i].ft>APDparams.max_time){NewElectron[i].ft=APDparams.max_
time;}

if(NewElectron[i].ft<APDparams.min_time){NewElectron[i].ft=APDparams.min_
time;}

          NewHole[i].ft=NewElectron[i].ft;
          if(NewElectron[i].ft<APDparams.temp_time)
            {
                  NewElectron[i].count=1;
                  NewHole[i].count=1;
            }
            NewElectron[i].ve=0.0;
            NewHole[i].ve=0.0;
```

```
            NewElectron[i].iv=1;
            NewHole[i].iv=1;

            NewElectron[i].e=(APDparams.photonev-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[NewElectron[i].iv]-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[NewHole[i].iv])*APDd
evice.DeviceLayers[APDdevice.index[PosIndex]].mh[NewHole[i].iv]/(APDdevic
e.DeviceLayers[APDdevice.index[PosIndex]].mh[NewHole[i].iv]+APDdevice.Dev
iceLayers[APDdevice.index[PosIndex]].me[NewElectron[i].iv]);

            NewHole[i].e=APDparams.photonev-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[NewElectron[i].iv]-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[NewHole[i].iv]-
NewElectron[i].e;

            float
ki=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[NewElectron[i].i
v]*sqrt(NewElectron[i].e*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosI
ndex]].af[NewElectron[i].iv]*NewElectron[i].e));

            randomnumber=float(rand())/float(RAND_MAX);

      float cs=0.5*randomnumber; //here assuming electrons being injected

      float sn=sqrt(1.0-cs*cs);
      randomnumber=float(rand())/float(RAND_MAX);
      float fai=2.0*pi*randomnumber;
      NewElectron[i].kx=ki*cs;
      NewElectron[i].ky=ki*sn*cos(fai);
      NewElectron[i].kz=ki*sn*sin(fai);


      ki=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[1]

*sqrt(NewHole[i].e*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]]
.afH[1]
        *NewHole[i].e));

      randomnumber=float(rand())/float(RAND_MAX);
      cs=1.0-2.0*randomnumber;
      sn=sqrt(1.0-cs*cs);
      randomnumber=float(rand())/float(RAND_MAX);
      fai=2.0*pi*randomnumber;
      NewHole[i].kx=ki*cs;
      NewHole[i].ky=ki*sn*cos(fai);
      NewHole[i].kz=ki*sn*sin(fai);

    if(NewElectron[i].ft<APDparams.temp_time){
                                            if
(NewElectron[i].count==0){NewElectron[i].count=1;}

StartElectron(APDdevice,APDparams,NewElectron,NewHole,APDresults,i);
                                            }

    if(NewHole[i].ft<APDparams.temp_time){
                                          if
(NewHole[i].count==0){NewHole[i].count=1;}
```

```
StartHole(APDdevice,APDparams,NewHole,NewElectron,APDresults,i);
                                                }



} //end for I0

  delete [] ft;

    for (j=0;j<APDresults.Ne;j++)
{APDdevice.charge_density[int(NewElectron[j].position/dx)]-
=APDparams.quasi_charge*NewElectron[j].count/device_volume;}
    for (j=0;j<APDresults.Nh;j++)
{APDdevice.charge_density[int(NewHole[j].position/dx)]+=APDparams.quasi_c
harge*NewHole[j].count/device_volume;}

  return;
}  //end CarrierLoop
```

CarrierLoop.C

```
#include "apd.h"

void CarrierLoop(device& APDdevice, params& APDparams,
    stats& APDresults, electron* NewElectron, hole* NewHole)
{
  int i=0;
  double randomnumber, randomnumber1;
  float dx=APDdevice.TotalThickness/float(NUMX);
  float bktq=bk*APDparams.tem/q;
  double device_area = pi*(device_diameter/2)*(device_diameter/2);
  double device_volume = dx*device_area;

  for(i=0;(i<APDresults.Ne);i++)
    {
    if(NewElectron[i].ft<APDparams.temp_time){
                                                if
(NewElectron[i].count==0){NewElectron[i].count=1;}
```

```
StartElectron(APDdevice,APDparams,NewElectron,NewHole,APDresults,i);
                                                    }
    }
  for(i=0;(i<APDresults.Nh);i++)
    {
    if(NewHole[i].ft<APDparams.temp_time){
                                                  if
(NewHole[i].count==0){NewHole[i].count=1;}

StartHole(APDdevice,APDparams,NewHole,NewElectron,APDresults,i);
                                                  }
    }
    for (i=0;i<APDresults.Ne;i++)
{APDdevice.charge_density[int(NewElectron[i].position/dx)]-
=APDparams.quasi_charge*NewElectron[i].count/device_volume;}
    for (i=0;i<APDresults.Nh;i++)
{APDdevice.charge_density[int(NewHole[i].position/dx)]+=APDparams.quasi_c
harge*NewHole[i].count/device_volume;}

  return;
}  //end CarrierLoop
```

AbsorbPhoton.C

```
#include "apd.h"

/****************
Generate an e h pair based on the absorption coefficients
of the different material layers
Returns the position of the e-h pair
*****************/

float AbsorbPhoton(device& APDdevice, stats& APDResults)
  {
    float randomnumber;
    float position=0;     //position for the absorption event
    float dx=APDdevice.TotalThickness/NUMX;  //size of a position step
inside device
    int i;               //counter
    while(position==0){
      for (i=2;i<NUMX-1; i++)
      {randomnumber=float(rand())/float(RAND_MAX);
```

```
        if ((GetAbs(APDdevice,i*dx)*dx)>randomnumber)
         { //if there is an abs event
        position=i*dx;      //note the position of the event

        APDResults.AbsDist[i]++;
        i=int(APDdevice.TotalThickness/dx);      //terminate loop

       } //end if
      } //end for
    } //end while
    return position;
  }
```

GetAbs.C

```
#include "apd.h"


/**********************
Steps through the structure until the layer containing
the desired position is found, and returns the absorption in that layer.
**********************/
float GetAbs(device& MyDevice, float position)
  {
    float CurrentThickness=0.0;
    int i;
    //CurrentLayer:i
    for (i=1;(position>CurrentThickness)&&(i<MyDevice.NumLayers);i++)
      {CurrentThickness+=MyDevice.DeviceLayers[i].Thickness;} //end for
    if (i<=1)
      {return MyDevice.DeviceLayers[1].Abs;}
    else
      {return MyDevice.DeviceLayers[i-1].Abs;}
  }
```

StartElectron.C

```
#include "apd.h"

/**************
This function gets a position and a time at which an e starts to move
after an event.
If there is another impact ionization, it starts a new e and h at the new
time and place.
If the count exceeds a max value, it will not start any more carriers
If the carrier is collected, it adds to the count and updates the time-
response histogram
It passes the pos and time as references to the function
NextElectronImpact,
which contains all the structure and physics details
***************/

void StartElectron(device& APDdevice, params& APDparams,electron*
NewElectron, hole* NewHole
              ,stats & APDresults, int e_number )
{
  float dx;       //size of section
```

```
  float randomnumber;
  int bin1, bin, temp_int;
  int PosIndex,NewPosIndex;
  float ei,ef,sq,sk,skx,sky,skz,tau,fai;
  float temp, temp1;
  dx=APDdevice.TotalThickness/float(NUMX);

      PosIndex=int(NewElectron[e_number].position/dx);
      NewElectron[e_number].impact=0;

  while ((PosIndex>1)
        &&(NewElectron[e_number].impact==0)
        &&(PosIndex<NUMX-
1)&&(NewElectron[e_number].ft<APDparams.temp_time))
    {


   do{
     do{randomnumber =
float(rand())/float(RAND_MAX);}while(randomnumber==0);

      tau=-
log(randomnumber)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].gm;
   if(tau<APDparams.min_time)
     {tau=APDparams.min_time;}
   NewPosIndex =
int((NewElectron[e_number].position+NewElectron[e_number].ve*tau)/dx);
   if(NewPosIndex<0) NewPosIndex =0;
   if(NewPosIndex>NUMX) NewPosIndex = NUMX;
   }
  while(abs(APDdevice.index[PosIndex]-APDdevice.index[NewPosIndex])>1);



 ElectronDrift(NewElectron[e_number],APDdevice,APDparams,APDresults,tau);


   bin1=int((NUM_BIN*NewElectron[e_number].ft)/APDparams.max_time);
   NewElectron[e_number].ft=NewElectron[e_number].ft+tau;
   bin=int((NUM_BIN*NewElectron[e_number].ft)/APDparams.max_time);

  PosIndex=int(NewElectron[e_number].position/dx);

        temp1=tau/(APDparams.max_time/NUM_BIN);

 if((PosIndex<NUMX-1)&&
    (PosIndex>1))
   {
    if ((bin<=NUM_BIN)&&(bin>1)) {
     if(APDdevice.EFTable[PosIndex]>1e6)
       {
      if(bin1==bin)
      {APDresults.Current[bin]+=APDparams.quasi_charge*q/(APDdevice.Deple
tionEnd-APDdevice.DepletionStart)*NewElectron[e_number].ve*temp1;}
      else
      {
      temp=(NewElectron[e_number].ft-bin*APDparams.max_time/NUM_BIN)/tau;
```

```
APDresults.Current[bin]+=APDparams.quasi_charge*q/(APDdevice.DepletionEnd
-APDdevice.DepletionStart)*NewElectron[e_number].ve*temp*temp1;

APDresults.Current[bin1]+=APDparams.quasi_charge*q/(APDdevice.DepletionEn
d-APDdevice.DepletionStart)*NewElectron[e_number].ve*temp1*(1-temp);
    }
       }
                                  }


    ElectronScat(APDdevice,APDparams,NewElectron[e_number]);
  }
 } //end while



  if (NewElectron[e_number].impact==1)
     {
       temp_int =
int(NewElectron[e_number].ft/APDparams.max_time*NUM_BIN);
       APDresults.impacte[temp_int]++;
       cout<<"e impact"<<endl;
       APDresults.Ne ++;
       NewElectron[APDresults.Ne].iv = 1;
       NewElectron[e_number].iv=1;
       NewElectron[APDresults.Ne].count = 1;

NewElectron[e_number].kx=NewElectron[e_number].kx/(2.0+APDdevice.DeviceLa
yers[APDdevice.index[PosIndex]].mh[1]/APDdevice.DeviceLayers[APDdevice.in
dex[PosIndex]].me[NewElectron[e_number].iv]);
       NewElectron[APDresults.Ne].kx = NewElectron[e_number].kx;
       ei=NewElectron[e_number].e;
       NewElectron[e_number].e=(ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[NewElectron[e_number
].iv]-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1])/(2.0+APDdevice.D
eviceLayers[APDdevice.index[PosIndex]].mh[1]/APDdevice.DeviceLayers[APDde
vice.index[PosIndex]].me[NewElectron[e_number].iv]);
       NewElectron[APDresults.Ne].e = NewElectron[e_number].e;
       ef=NewElectron[e_number].e;

sq=ef*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2[NewElectron[e
_number].iv]+1.0;
       sk=(sq*sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4[NewElectron[e_
number].iv]/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhm[NewElec
tron[e_number].iv];
       skx=NewElectron[e_number].kx*NewElectron[e_number].kx;
       randomnumber=float(rand())/float(RAND_MAX);
       fai=2.0*pi*randomnumber;
       NewElectron[e_number].ky=sqrt(fabs(sk-skx))*cos(fai);
       NewElectron[e_number].kz=sqrt(fabs(sk-skx))*sin(fai);
       NewElectron[APDresults.Ne].ky = NewElectron[e_number].ky;
       NewElectron[APDresults.Ne].ky = NewElectron[e_number].ky;
       NewElectron[APDresults.Ne].parent = NewElectron[e_number].parent;
       NewElectron[APDresults.Ne].ft = NewElectron[e_number].ft;
```

```
	NewElectron[APDresults.Ne].position =
NewElectron[e_number].position;



	APDresults.Nh ++;
	NewHole[APDresults.Nh].parent = NewElectron[e_number].parent;
	NewHole[APDresults.Nh].position = NewElectron[e_number].position;
	NewHole[APDresults.Nh].ft = NewElectron[e_number].ft;
	NewHole[APDresults.Nh].count = 1;
	NewHole[APDresults.Nh].iv=1;

NewHole[APDresults.Nh].kx=NewElectron[e_number].kx*APDdevice.DeviceLayers
[APDdevice.index[PosIndex]].mh[1]/APDdevice.DeviceLayers[APDdevice.index[
PosIndex]].me[NewElectron[e_number].iv];
	NewHole[APDresults.Nh].e=ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[NewElectron[e_number
].iv]-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]-
NewElectron[e_number].e*2.0;
	ef=NewHole[APDresults.Nh].e;

sq=ef*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2H[NewHole[APDr
esults.Nh].iv]+1.0;
	sk=(sq*sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4H[NewHole[APDre
sults.Nh].iv]/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhmH[NewH
ole[APDresults.Nh].iv];
	skx=NewHole[APDresults.Nh].kx*NewHole[APDresults.Nh].kx;
	randomnumber=float(rand())/float(RAND_MAX);
	fai=2.0*pi*randomnumber;
	NewHole[APDresults.Nh].ky=sqrt(fabs(sk-skx))*cos(fai);
	NewHole[APDresults.Nh].kz=sqrt(fabs(sk-skx))*sin(fai);



NewElectron[e_number].ve=sqrt(2.0*NewElectron[e_number].e/APDdevice.Devic
eLayers[APDdevice.index[PosIndex]].me[NewElectron[e_number].iv]);
	NewElectron[APDresults.Ne].ve = NewElectron[e_number].ve;

NewHole[APDresults.Nh].ve=sqrt(2.0*NewHole[APDresults.Nh].e/APDdevice.Dev
iceLayers[APDdevice.index[PosIndex]].mh[1]);



	if ((PosIndex<=NUMX)&&(PosIndex>=1))
		{
		APDresults.ImpactDist_e[PosIndex]++;
		}


StartHole(APDdevice,APDparams,NewHole,NewElectron,APDresults,APDresults.N
h);

StartElectron(APDdevice,APDparams,NewElectron,NewHole,APDresults,APDresul
ts.Ne);

StartElectron(APDdevice,APDparams,NewElectron,NewHole,APDresults,e_number
);
```

```
    }  //end if there is an impact event

  return;
} //end StartElectron
```

ElectronDrift.C

```
#include "apd.h"

void ElectronDrift(electron& DriftElectron,
           device& APDdevice, params& APDparams, stats& APDresults, float
tau)

{
  float randomnumber;

  float qh=q/h;


  float dx=APDdevice.TotalThickness/float(NUMX);
  int PosIndex=int(DriftElectron.position/dx);

  float fx=APDdevice.EFTable[PosIndex];

  float ei=DriftElectron.e;
  float dkx=qh*fx*tau;  //change of kx due to field fx

  DriftElectron.kx=DriftElectron.kx+dkx;

  float skx=DriftElectron.kx*DriftElectron.kx;
  float sky=DriftElectron.ky*DriftElectron.ky;
  float skz=DriftElectron.kz*DriftElectron.kz;
  float sk=skx+sky+skz;
```

```
  float
sq=sqrt(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4[DriftEl
ectron.iv]

*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhm[DriftElectron.iv]*
sk);

  float ef=(sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2[DriftElectron.
iv];



  DriftElectron.e=ef;

  DriftElectron.impact=0;

 //cout<<"in ElectronDrift.C, ef="<<ef<<endl;


  DriftElectron.ve=(ef-ei)/fx/tau;
if(DriftElectron.ve>3e7) {DriftElectron.ve = 3e7;}
if(DriftElectron.ve<-3e7) {DriftElectron.ve = -3e7;}


  //******reflection at heterojunction*****************************
  int PosIndexL=int(DriftElectron.position/dx);
    if((PosIndexL>1)&&(PosIndexL<NUMX-1))
      {
  DriftElectron.position=DriftElectron.position+DriftElectron.ve*tau;
  if(DriftElectron.position<0)
    {DriftElectron.position=0.0;}
  if(DriftElectron.position>APDdevice.TotalThickness)
    {DriftElectron.position=APDdevice.TotalThickness;}
    PosIndex=int(DriftElectron.position/dx);


APDresults.EnergyDist_e[PosIndex]=APDresults.EnergyDist_e[PosIndex]+Drift
Electron.e;



  if(DriftElectron.iv==1)
    {
  APDresults.GDist_e[PosIndex]=APDresults.GDist_e[PosIndex]+1;
    }
  else if(DriftElectron.iv==2)
    {
  APDresults.LDist_e[PosIndex]=APDresults.LDist_e[PosIndex]+1;
    }


  APDresults.VelocityDist_e[PosIndex]=
  APDresults.VelocityDist_e[PosIndex]+DriftElectron.ve;
  APDresults.CountDist_e[PosIndex]=
```

```
  APDresults.CountDist_e[PosIndex]+1;

    if((PosIndex>1)&&(PosIndex<NUMX-
1)&&(APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].me[DriftElectron.
iv]!=
      APDdevice.DeviceLayers[APDdevice.index[PosIndex]].me[DriftElectron.
iv]))
    {

      float EL=DriftElectron.e;
      float ER=DriftElectron.e-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[DriftElectron.iv]+

APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].Ec[DriftElectron.iv];

     float
kL=sqrt(2*APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].me[DriftElec
tron.iv]*am0
          *EL)/h;
     float
kR=sqrt(2*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].me[DriftElect
ron.iv]*am0
          *fabs(ER))/h;

     float
Reflection=(kL/APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].me[Drif
tElectron.iv]
                 -
kR/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].me[DriftElectron.iv]
)

/(kL/APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].me[DriftElectron.
iv]

+kR/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].me[DriftElectron.iv
]);

     //assume no reflection

          Reflection=Reflection*Reflection;
          randomnumber=float(rand())/float(RAND_MAX);

  if(randomnumber<Reflection)
    {
  DriftElectron.e=EL;
  DriftElectron.kx=-DriftElectron.kx;
  DriftElectron.position=PosIndexL*dx;
  DriftElectron.ve=-DriftElectron.ve;

    }
          else
    {
      if(DriftElectron.ve<0)
      {DriftElectron.position=(PosIndex-0.5)*dx;}
      else
        {DriftElectron.position=(PosIndex+0.5)*dx;}
  DriftElectron.e=ER;
  if(DriftElectron.e<0)
```

```
    {
      randomnumber=float(rand())/float(RAND_MAX);
      float bktq=bk*APDparams.tem/q;
      DriftElectron.e=-bktq*log(randomnumber)*1.5;
    }
      float
ki=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[DriftElectron.iv
]*sqrt(DriftElectron.e*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosInd
ex]].af[DriftElectron.iv]*DriftElectron.e));
      float kyz=sqrt(fabs(ki*ki-DriftElectron.kx*DriftElectron.kx));
      randomnumber=float(rand())/float(RAND_MAX);
      float fai=2.0*pi*randomnumber;

      DriftElectron.ky=kyz*cos(fai);
      DriftElectron.kz=kyz*sin(fai);

    }
    }
      }

      return;
}
```

ElectronScat.C

```
#include "apd.h"

void ElectronScat(device& APDdevice, params& APDparams,
   electron& ScatElectron)

{

  float x1,x2,x3,r1,r2,cs,sn,cf,sf;
    float skk,a11,a12,a13,a21,a22,a23,a31,a32,a33;
    float ef,kf,cb;
    float f,randomnumber,sb,fai;



    // int idum=-2;

  float ei=ScatElectron.e;
  float dx=APDdevice.TotalThickness/float(NUMX);
  int PosIndex=int(ScatElectron.position/dx);

  ScatElectron.impact=0;


 if(ei==0.0){return;}
 //   {// bracket all the way to the end.    point *1*
 // it seems that no need of else statement when there is
 // return in if.

      float sk=ScatElectron.kx*ScatElectron.kx
            +ScatElectron.ky*ScatElectron.ky
      +ScatElectron.kz*ScatElectron.kz;
```

```
      float ki=sqrt(sk);
      int ie=int(ei/de)+1;
      if(ie>iemax){ie=iemax;}


      if(ScatElectron.iv==1)
      {     // bracket the 1000 block     point *1000*


  //alloy scattering,
         r1=float(rand())/float(RAND_MAX);


if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][0][ie])
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
         }

  //emission of phonon, within Gamma
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][1][ie])
        {
          float ef=ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}
          //goto 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]
```

```
*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);
          if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);

          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;

            //30 block following 20 block
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

           cf=cos(fai);
           sf=sin(fai);
           skk=sqrt(ScatElectron.kx*ScatElectron.kx
                +ScatElectron.ky*ScatElectron.ky);
           a11=ScatElectron.ky/skk;
           a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
           a13=ScatElectron.kx/ki;
           a21=-ScatElectron.kx/skk;
           a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
           a23=ScatElectron.ky/ki;
           a32=-skk/ki;
           a33=ScatElectron.kz/ki;
           x1=kf*sb*cf;
           x2=kf*sb*sf;
           x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;



        }
      //absorption of phonon, within Gamma
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][2][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
          if(f<=0.0){return;}
           randomnumber=float(rand())/float(RAND_MAX);

          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;

            //30 block following 20 block
         //         goto 30
         sb=sqrt(1.0-cb*cb);
```

```
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

      //intervalley emission, from 1-2,  Gamma-->L
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][3][ie])
      {
      ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
        +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]
        -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2];
      if(ef<=0.0){return;}
      ScatElectron.iv=2;

      //goto 40
      // 40 block


      kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectr
on.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
      return;}

      //intervalley emission, from 1-0,  Gamma-->X
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][4][ie])
```

```
              {
             ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]
                    -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0];
             if(ef<=0.0){return;}
             ScatElectron.iv=0;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatElectron.kx=kf*cs;
   ScatElectron.ky=kf*sn*cos(fai);
   ScatElectron.kz=kf*sn*sin(fai);
   ScatElectron.e=ef;
   return;

            }

      //intervalley absorption, Gamma to L
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][5][ie])
              {
             ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]
                    -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2];
             if(ef<=0.0){return;}
             ScatElectron.iv=2;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatElectron.kx=kf*cs;
   ScatElectron.ky=kf*sn*cos(fai);
   ScatElectron.kz=kf*sn*sin(fai);
   ScatElectron.e=ef;
   return;

            }
```

```
      //intervalley absorption, from 1-0,  Gamma-->X
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][6][ie])
              {
            ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]
                  -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0];
            if(ef<=0.0){return;}
            ScatElectron.iv=0;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatElectron.kx=kf*cs;
   ScatElectron.ky=kf*sn*cos(fai);
   ScatElectron.kz=kf*sn*sin(fai);
   ScatElectron.e=ef;
   return;
            }

//acoustic phonon in Gamma, little change in energy
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][7][ie])
        {
         ef=ei;
         kf=ki;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatElectron.kx=kf*cs;
   ScatElectron.ky=kf*sn*cos(fai);
   ScatElectron.kz=kf*sn*sin(fai);
   ScatElectron.e=ef;
   return;
       }

      //impurity scattering
      else
```

```
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][8][ie])
      {
        ef=ei;

         r2=float(rand())/float(RAND_MAX);

   ScatElectron.e=ef;
   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;

         }


       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[1][9][ie])|
|(ScatElectron.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Eacte
H+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]+APDdevice.Devic
eLayers[APDdevice.index[PosIndex]].Ev[1])))
      {

        ScatElectron.impact=1;

          return;

         }
      }
      else if(ScatElectron.iv==2)  //  2000 block
{  //alloy scattering,
         r1=float(rand())/float(RAND_MAX);


if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][0][ie])
```

```
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
         }


            //L valleys polar optical phonon emission,
    else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][1][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}
          //          goto 20
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);

        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                 //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
```

```
          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

      //L valleys polar optical phonon absorption,
      else
if(r1<APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][2][ie])
        {
          ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
```

```
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

      //L valley Nonpolar optical phonon emission
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][3][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          if(ef<=0.0){return;}
          //     goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //L valley Nonpolar optical phonon absorption
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][4][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          //goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //L --> Gamma, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][5][ie])
```

```
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1];
        if(ef<=0.0){return;}
        ScatElectron.iv=1;
                    //    goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //L --> Gamma, absorption
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][6][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1];
        if(ef<=0.0){return;}
        ScatElectron.iv=1;
                    //goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
        }

      //L --> X, emission
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][7][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
```

```
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0];
          if(ef<=0.0){return;}
          ScatElectron.iv=0;
                      //    goto 40
            //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
            randomnumber=float(rand())/float(RAND_MAX);
            cs=1.0-2.0*randomnumber;
            sn=sqrt(1.0-cs*cs);
              randomnumber=float(rand())/float(RAND_MAX);
            fai=2.0*pi*randomnumber;
            ScatElectron.kx=kf*cs;
            ScatElectron.ky=kf*sn*cos(fai);
            ScatElectron.kz=kf*sn*sin(fai);
            ScatElectron.e=ef;
            return;
              }

        //L --> X, absorption
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][8][ie])
          {
          ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
              +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2]
              -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0];
          if(ef<=0.0){return;}
          ScatElectron.iv=0;
                      //goto 40
            //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
            randomnumber=float(rand())/float(RAND_MAX);
            cs=1.0-2.0*randomnumber;
            sn=sqrt(1.0-cs*cs);
              randomnumber=float(rand())/float(RAND_MAX);
            fai=2.0*pi*randomnumber;
            ScatElectron.kx=kf*cs;
            ScatElectron.ky=kf*sn*cos(fai);
            ScatElectron.kz=kf*sn*sin(fai);
            ScatElectron.e=ef;
            return;
          }

        //acoustic phonon  L valley
           else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][9][ie])
          {
            ef=ei;
            kf=ki;
            //  goto 40
```

```
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;

         }

      //impurity scattering L valley
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][10][ie])
        {
          ef=ei;
           r2=float(rand())/float(RAND_MAX);
           cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
           kf=ki;
         //    goto 30
                 //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

  //Impact Ionization
```

```
       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[2][11][ie])
||(ScatElectron.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Eact
eH+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2]+APDdevice.Devi
ceLayers[APDdevice.index[PosIndex]].Ev[1])))
        {
          ScatElectron.impact=1;
          return;
        }
}// ...iv==2

     //X valley*********************************************

     else  //iv==0
   {
         r1=float(rand())/float(RAND_MAX);

        //alloy scattering,

if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][0][ie])
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
         }

            //X valleys polar optical phonon emission,
     else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][1][ie])
        {
```

```
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}
          //          goto 20
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);

        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

      //X valleys polar optical phonon absorption,
      else
if(r1<APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][2][ie])
        {
          ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
```

```
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

      //X valley Nonpolar optical phonon emission
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][3][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          if(ef<=0.0){return;}
          //     goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //X valley Nonpolar optical phonon absorption
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][4][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          //goto 40
          //40 ...
```

```
kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //X --> Gamma, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][5][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1];
        if(ef<=0.0){return;}
        ScatElectron.iv=1;
                    //    goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //X --> Gamma, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][6][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1];
        if(ef<=0.0){return;}
        ScatElectron.iv=1;
                    //goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]
```

```
*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
        }

      //X --> L, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][7][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2];
        if(ef<=0.0){return;}
        ScatElectron.iv=2;
                    //    goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
            }

      //X --> L, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][8][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[2];
        if(ef<=0.0){return;}
        ScatElectron.iv=2;
                    //goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
```

```
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;
        }

      //acoustic phonon  X valley
         else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][9][ie])
        {
          ef=ei;
          kf=ki;
          //  goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smh[ScatElectron.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af[ScatEl
ectron.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatElectron.kx=kf*cs;
          ScatElectron.ky=kf*sn*cos(fai);
          ScatElectron.kz=kf*sn*sin(fai);
          ScatElectron.e=ef;
          return;

         }

      //impurity scattering X valley
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][10][ie])
        {
          ef=ei;
           r2=float(rand())/float(RAND_MAX);
           cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
           kf=ki;
         //    goto 30
                 //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatElectron.kx*ScatElectron.kx
               +ScatElectron.ky*ScatElectron.ky);
          a11=ScatElectron.ky/skk;
```

```
          a12=ScatElectron.kx*ScatElectron.kz/skk/ki;
          a13=ScatElectron.kx/ki;
          a21=-ScatElectron.kx/skk;
          a22=ScatElectron.ky*ScatElectron.kz/skk/ki;
          a23=ScatElectron.ky/ki;
          a32=-skk/ki;
          a33=ScatElectron.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatElectron.kx=a11*x1+a12*x2+a13*x3;
          ScatElectron.ky=a21*x1+a22*x2+a23*x3;
          ScatElectron.kz=a32*x2+a33*x3;
          ScatElectron.e=ef;
          return;
        }

  //Impact Ionization

       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swk[0][11][ie])
||(ScatElectron.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Eact
eH+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[0]+APDdevice.Devi
ceLayers[APDdevice.index[PosIndex]].Ev[1])))
        {
          ScatElectron.impact=1;
          return;
        }
   }// X valley block
}//void...
```

StartHole.C

```
#include "apd.h"

/**************
This function gets a position and a time at which an e starts to move
after an event.
If there is another impact ionization, it starts a new e and h at the new
time and place.
If the count exceeds a max value, it will not start any more carriers
If the carrier is collected, it adds to the count and updates the time-
response histogram
It passes the pos and time as references to the function
NextElectronImpact,
which contains all the structure and physics details
***************/

void StartHole(device& APDdevice, params& APDparams,
                  hole* NewHole, electron* NewElectron
             ,stats& APDresults, int h_number)
{
  float ei,ef,sq,sk,sky,skz,skx,fai,tau;

  float dx;       //size of section
  float randomnumber;
  int bin, bin1, temp_int;
```

```
  int PosIndex,NewPosIndex;
  float temp1, temp;

  dx=APDdevice.TotalThickness/float(NUMX);

      PosIndex=int(NewHole[h_number].position/dx);
      NewHole[h_number].impact=0;

  while ((PosIndex>2)&&(PosIndex<NUMX-1)&&
        (NewHole[h_number].impact==0)
       &&(NewHole[h_number].ft<APDparams.temp_time))
    {

do {
   do{randomnumber =
float(rand())/float(RAND_MAX);}while(randomnumber==0);

   tau=-
log(randomnumber)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].gmH;
   if(tau<APDparams.min_time)
     {tau=APDparams.min_time;}
   NewPosIndex =
int((NewHole[h_number].position+NewHole[h_number].ve*tau)/dx);
   if(NewPosIndex<0) NewPosIndex = 0;
   if(NewPosIndex>NUMX) NewPosIndex = NUMX;
   }while(abs(APDdevice.index[PosIndex]-APDdevice.index[NewPosIndex])>1);


 HoleDrift(NewHole[h_number],APDdevice,APDparams,APDresults,tau);


  bin1=int((NUM_BIN*NewHole[h_number].ft)/APDparams.max_time);
   NewHole[h_number].ft=NewHole[h_number].ft+tau;
    bin=int((NUM_BIN*NewHole[h_number].ft)/APDparams.max_time);
  PosIndex=int(NewHole[h_number].position/dx);

        temp1=tau/(APDparams.max_time/NUM_BIN);

 if((PosIndex<NUMX-1)&&(PosIndex>1))
 {
    if ((bin<=NUM_BIN)&&(bin>2))
      {
        if(APDdevice.EFTable[PosIndex]>1e6)
        {
      if(bin1==bin)
      {
    APDresults.Current[bin]-
=APDparams.quasi_charge*q/(APDdevice.DepletionEnd-
APDdevice.DepletionStart)*NewHole[h_number].ve*temp1;
    }
      else
      {
          temp=(NewHole[h_number].ft-bin*APDparams.max_time/NUM_BIN)/tau;
        APDresults.Current[bin]-
=APDparams.quasi_charge*q/(APDdevice.DepletionEnd-
APDdevice.DepletionStart)*NewHole[h_number].ve*temp*temp1;
        APDresults.Current[bin1]-
=APDparams.quasi_charge*q/(APDdevice.DepletionEnd-
```

```
APDdevice.DepletionStart)*NewHole[h_number].ve*temp1*(1-temp);
          } }
          }

   HoleScat(APDdevice,APDparams,NewHole[h_number]);

 }    //end if
}//end while



  if (NewHole[h_number].impact==1)
     {
       temp_int = int(NewHole[h_number].ft/APDparams.max_time*NUM_BIN);
       APDresults.impacth[temp_int]++;
       cout<<"h impact"<<endl;
       APDresults.Nh ++;
       NewHole[APDresults.Nh].iv = 1;
       NewHole[h_number].iv=1;
       NewHole[APDresults.Nh].count = 1;

NewHole[h_number].kx=NewHole[h_number].kx/(2.0+APDdevice.DeviceLayers[APD
device.index[PosIndex]].me[1]/APDdevice.DeviceLayers[APDdevice.index[PosI
ndex]].mh[NewHole[h_number].iv]);
       NewHole[APDresults.Nh].kx = NewHole[h_number].kx;
       ei=NewHole[h_number].e;
       NewHole[h_number].e=(ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[NewHole[h_number].iv
])/(2.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].mh[NewHole[h_nu
mber].iv]/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].me[1]);
       NewHole[APDresults.Nh].e = NewHole[h_number].e;
       ef=NewHole[h_number].e;

sq=ef*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2H[NewHole[h_nu
mber].iv]+1.0;
       sk=(sq*sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4H[NewHole[h_num
ber].iv]/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhmH[NewHole[h
_number].iv];
       skx=NewHole[h_number].kx*NewHole[h_number].kx;
       randomnumber=float(rand())/float(RAND_MAX);
       fai=2.0*pi*randomnumber;
       NewHole[h_number].ky=sqrt(fabs(sk-skx))*cos(fai);
       NewHole[h_number].kz=sqrt(fabs(sk-skx))*sin(fai);
       NewHole[APDresults.Nh].ky = NewHole[h_number].ky;
       NewHole[APDresults.Nh].kz = NewHole[h_number].kz;
       NewHole[APDresults.Nh].parent = NewHole[h_number].parent;
       NewHole[APDresults.Nh].position = NewHole[h_number].position;
       NewHole[APDresults.Nh].ft = NewHole[h_number].ft;


       APDresults.Ne ++;
       NewElectron[APDresults.Ne].parent = NewHole[h_number].parent;
       NewElectron[APDresults.Ne].position = NewHole[h_number].position;
       NewElectron[APDresults.Ne].ft = NewHole[h_number].ft;
       NewElectron[APDresults.Ne].count = 1;
```

```
       NewElectron[APDresults.Ne].iv=1;

NewElectron[APDresults.Ne].kx=NewHole[h_number].kx*APDdevice.DeviceLayers
[APDdevice.index[PosIndex]].me[1]/APDdevice.DeviceLayers[APDdevice.index[
PosIndex]].mh[NewHole[h_number].iv];
       NewElectron[APDresults.Ne].e=ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ec[1]-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[NewHole[h_number].iv
]-NewHole[h_number].e*2.0;
       ef=NewElectron[APDresults.Ne].e;

sq=ef*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2[NewElectron[A
PDresults.Ne].iv]+1.0;
       sk=(sq*sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4[NewElectron[AP
Dresults.Ne].iv]/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhm[Ne
wElectron[APDresults.Ne].iv];
       skx=NewElectron[APDresults.Ne].kx*NewElectron[APDresults.Ne].kx;
       randomnumber=float(rand())/float(RAND_MAX);
       fai=2.0*pi*randomnumber;
       NewElectron[APDresults.Ne].ky=sqrt(fabs(sk-skx))*cos(fai);
       NewElectron[APDresults.Ne].kz=sqrt(fabs(sk-skx))*sin(fai);



      NewHole[h_number].ve=-
sqrt(2.0*NewHole[h_number].e/APDdevice.DeviceLayers[APDdevice.index[PosIn
dex]].mh[NewHole[h_number].iv]);
      NewHole[APDresults.Nh].ve = NewHole[h_number].ve;
      NewElectron[APDresults.Ne].ve=-
sqrt(2.0*NewElectron[APDresults.Ne].e/APDdevice.DeviceLayers[APDdevice.in
dex[PosIndex]].me[1]);


       if ((PosIndex<=NUMX)&&(PosIndex>=1))
                {
                APDresults.ImpactDist_h[PosIndex]++;
               }


StartElectron(APDdevice,APDparams,NewElectron,NewHole,APDresults,APDresul
ts.Ne);

StartHole(APDdevice,APDparams,NewHole,NewElectron,APDresults,APDresults.N
h);

StartHole(APDdevice,APDparams,NewHole,NewElectron,APDresults,h_number);

     }  ////end if there is an impact event

return;
} //end StartHole
```

HoleDrift.C

```
#include "apd.h"

void HoleDrift(hole& DriftHole,
           device& APDdevice, params& APDparams, stats& APDresults, float
tau)

{

  float randomnumber;
  float qh=q/h;

  float dx=APDdevice.TotalThickness/float(NUMX);

  int PosIndex=int(DriftHole.position/dx);

  float fx=APDdevice.EFTable[PosIndex];

  float ei=DriftHole.e;

  float dkx=-qh*fx*tau;  //change of kx due to field fx

  float tmp_ve = DriftHole.ve;


  DriftHole.kx=DriftHole.kx+dkx;

  float skx=DriftHole.kx*DriftHole.kx;
  float sky=DriftHole.ky*DriftHole.ky;
  float skz=DriftHole.kz*DriftHole.kz;
  float sk=skx+sky+skz;
  float
```

```
sq=sqrt(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af4H[DriftH
ole.iv]

*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hhmH[DriftHole.iv]*sk)
;

  float ef=(sq-
1.0)/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af2H[DriftHole.iv]
;
  DriftHole.e=ef;

  DriftHole.impact=0;



  DriftHole.ve=-(ef-ei)/fx/tau;
if(DriftHole.ve>3e7)    {DriftHole.ve = 3e7;}
if(DriftHole.ve<-3e7)    {DriftHole.ve = -3e7;}


  //******reflection at heterojunction*****************************

        int PosIndexL=int(DriftHole.position/dx);
    if((PosIndexL>1)&&(PosIndexL<NUMX-1))
      {

  DriftHole.position=DriftHole.position+DriftHole.ve*tau;

  if(DriftHole.position<0)
    {DriftHole.position=0.0;}
  if(DriftHole.position>APDdevice.TotalThickness)
    {DriftHole.position=APDdevice.TotalThickness;}

    PosIndex=int(DriftHole.position/dx);



  APDresults.EnergyDist_h[PosIndex]=
  APDresults.EnergyDist_h[PosIndex]+DriftHole.e;

  if(DriftHole.iv==1)
    {
  APDresults.GDist_h[PosIndex]=APDresults.GDist_h[PosIndex]+1;
    }
  else if(DriftHole.iv==2)
    {
  APDresults.LDist_h[PosIndex]=APDresults.LDist_h[PosIndex]+1;
    }

  APDresults.VelocityDist_h[PosIndex]=
  APDresults.VelocityDist_h[PosIndex]+DriftHole.ve;
  APDresults.CountDist_h[PosIndex]=
  APDresults.CountDist_h[PosIndex]+1;

    if((PosIndex>1)&&(PosIndex<NUMX-
1)&&(APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].mh[DriftHole.iv]!
=
      APDdevice.DeviceLayers[APDdevice.index[PosIndex]].mh[DriftHole.iv])
```

```
)
    {

      float EL=DriftHole.e;
      float ER=DriftHole.e-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[DriftHole.iv]+
  APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].Ev[DriftHole.iv];

     float
kL=sqrt(2*APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].mh[DriftHole
.iv]*am0
          *EL)/h;
     float
kR=sqrt(2*APDdevice.DeviceLayers[APDdevice.index[PosIndex]].mh[DriftHole.
iv]*am0
          *fabs(ER))/h;

     float
Reflection=(kL/APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].mh[Drif
tHole.iv]
                  -
kR/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].mh[DriftHole.iv])

/(kL/APDdevice.DeviceLayers[APDdevice.index[PosIndexL]].mh[DriftHole.iv]

+kR/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].mh[DriftHole.iv]);

          Reflection=Reflection*Reflection;
          float randumnumber=float(rand())/float(RAND_MAX);
  if(randumnumber<Reflection)
    {
  DriftHole.e=EL;
  DriftHole.kx=-DriftHole.kx;
  DriftHole.position=PosIndexL*dx;
  DriftHole.ve=-DriftHole.ve;

    }
          else
    {
      if(DriftHole.ve<0)
      {DriftHole.position=(PosIndex-0.5)*dx;}
      else
        {DriftHole.position=(PosIndex+0.5)*dx;}
  DriftHole.e=ER;
    if(DriftHole.e<0)
    {
      randomnumber=float(rand())/float(RAND_MAX);
      float bktq=bk*APDparams.tem/q;
      DriftHole.e=-bktq*log(randomnumber)*1.5;
    }
      float
ki=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[DriftHole.iv]*s
qrt(DriftHole.e*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].af
H[DriftHole.iv]*DriftHole.e));
      float kyz=sqrt(fabs(ki*ki-DriftHole.kx*DriftHole.kx));
      randomnumber=float(rand())/float(RAND_MAX);
      float fai=2.0*pi*randomnumber;
      DriftHole.ky=kyz*cos(fai);
```

```
      DriftHole.kz=kyz*sin(fai);

    }
    }
      } //matches if(..>2 ..< NUMX-1...)...

      return;
}
```

HoleScat.C

```
#include "apd.h"

void HoleScat(device& APDdevice, params& APDparams,
   hole& ScatHole)

{

  float x1,x2,x3,r1,r2,cs,sn,cf,sf;
    float skk,a11,a12,a13,a21,a22,a23,a31,a32,a33;
    float ef,kf,cb;
    float f,randomnumber,sb,fai;



    // int idum=-2;

  float ei=ScatHole.e;
  float dx=APDdevice.TotalThickness/float(NUMX);
  int PosIndex=int(ScatHole.position/dx);

  ScatHole.impact=0;

 if(ei==0.0){return;}
 //   {// bracket all the way to the end.    point *1*
 // it seems that no need of else statement when there is
 // return in if.

      float sk=ScatHole.kx*ScatHole.kx
            +ScatHole.ky*ScatHole.ky
      +ScatHole.kz*ScatHole.kz;

      float ki=sqrt(sk);
      int ie=int(ei/de)+1;
      if(ie>iemax){ie=iemax;}
```

```
      if(ScatHole.iv==1)
      {     // bracket the 1000 block     point *1000*

  //alloy scattering,
         r1=float(rand())/float(RAND_MAX);


if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][0][ie])
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
         }

  //emission of phonon, within Gamma
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][1][ie])
        {
          float ef=ei-
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);
          if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
```

```
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;

         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

           cf=cos(fai);
           sf=sin(fai);
           skk=sqrt(ScatHole.kx*ScatHole.kx
                +ScatHole.ky*ScatHole.ky);
           a11=ScatHole.ky/skk;
           a12=ScatHole.kx*ScatHole.kz/skk/ki;
           a13=ScatHole.kx/ki;
           a21=-ScatHole.kx/skk;
           a22=ScatHole.ky*ScatHole.kz/skk/ki;
           a23=ScatHole.ky/ki;
           a32=-skk/ki;
           a33=ScatHole.kz/ki;
           x1=kf*sb*cf;
           x2=kf*sb*sf;
           x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
         }
       //absorption of phonon, within Gamma
       else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][2][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
          if(f<=0.0){return;}
           randomnumber=float(rand())/float(RAND_MAX);

          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;

         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
```

```
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

      //intervalley emission, from 1-2,
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][3][ie])
      {
      ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
        +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]
        -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2];
      if(ef<=0.0){return;}
      ScatHole.iv=2;

      kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.
iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
      return;}

      //intervalley emission, from 1-0,
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][4][ie])
              {
            ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]
                  -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0];
            if(ef<=0.0){return;}
            ScatHole.iv=0;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);
```

```
 fai=2.0*pi*randomnumber;
   ScatHole.kx=kf*cs;
   ScatHole.ky=kf*sn*cos(fai);
   ScatHole.kz=kf*sn*sin(fai);
   ScatHole.e=ef;
   return;

            }

      //intervalley absorption, 1--2
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][5][ie])
              {
             ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]
                    -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2];
             if(ef<=0.0){return;}
             ScatHole.iv=2;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatHole.kx=kf*cs;
   ScatHole.ky=kf*sn*cos(fai);
   ScatHole.kz=kf*sn*sin(fai);
   ScatHole.e=ef;
   return;

            }

      //intervalley absorption, from 1-0,  Gamma-->X
          else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][6][ie])
              {
             ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij

+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]
                    -
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0];
             if(ef<=0.0){return;}
             ScatHole.iv=0;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
```

```
 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatHole.kx=kf*cs;
   ScatHole.ky=kf*sn*cos(fai);
   ScatHole.kz=kf*sn*sin(fai);
   ScatHole.e=ef;
   return;
            }

//acoustic phonon in Gamma, little change in energy
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][7][ie])
        {
         ef=ei;
         kf=ki;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);

 cs=1.0-2.0*randomnumber;
 sn=sqrt(1.0-cs*cs);
          randomnumber=float(rand())/float(RAND_MAX);

 fai=2.0*pi*randomnumber;
   ScatHole.kx=kf*cs;
   ScatHole.ky=kf*sn*cos(fai);
   ScatHole.kz=kf*sn*sin(fai);
   ScatHole.e=ef;
   return;
       }

      //impurity scattering
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][8][ie])
      {
        ef=ei;

         r2=float(rand())/float(RAND_MAX);

   ScatHole.e=ef;
   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
```

```
            +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;

         }

       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[1][9][ie])
||(ScatHole.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].EacthH+A
PDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1]+APDdevice.DeviceLa
yers[APDdevice.index[PosIndex]].Ec[1])))
      {
        ScatHole.impact=1;
          return;
         }
      }
      else if(ScatHole.iv==2)  //  2000 block
{  //alloy scattering,
         r1=float(rand())/float(RAND_MAX);


if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][0][ie])
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         //         goto 30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
```

```
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
         }
            //L valleys polar optical phonon emission,
    else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][1][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);

        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

      //L valleys polar optical phonon absorption,
      else
if(r1<APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][2][ie])
        {
          ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
```

```
kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

      //L valley Nonpolar optical phonon emission
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][3][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          if(ef<=0.0){return;}

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }
```

```
      //L valley Nonpolar optical phonon absorption
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][4][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //L --> Gamma, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][5][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1];
        if(ef<=0.0){return;}
        ScatHole.iv=1;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //L --> Gamma, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][6][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1];
        if(ef<=0.0){return;}
        ScatHole.iv=1;
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]
```

```
*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
        }

      //L --> X, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][7][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0];
        if(ef<=0.0){return;}
        ScatHole.iv=0;

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //L --> X, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][8][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0];
        if(ef<=0.0){return;}
        ScatHole.iv=0;
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
```

```
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
        }

      //acoustic phonon  L valley
         else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][9][ie])
        {
          ef=ei;
          kf=ki;
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;

         }

      //impurity scattering L valley
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][10][ie])
        {
          ef=ei;
           r2=float(rand())/float(RAND_MAX);
           cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
           kf=ki;
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
```

```
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

  //Impact Ionization
       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[2][11][ie]
)||(ScatHole.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].EacthH+
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2]+APDdevice.DeviceL
ayers[APDdevice.index[PosIndex]].Ec[1])))
        {
          ScatHole.impact=1;
          return;
        }
}// ...iv==2

     //X valley*******************************************

     else  //iv==0
   {
         r1=float(rand())/float(RAND_MAX);

        //alloy scattering,

if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][0][ie])
         {    ef=ei;

         r2=float(rand())/float(RAND_MAX);

   cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
         kf=ki;
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
```

```
          return;
         }

            //X valleys polar optical phonon emission,
     else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][1][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          if(ef<=0.0){return;}
          //          goto 20
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          f=2.0*ki*kf/(ki-kf)/(ki-kf);

        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

      //X valleys polar optical phonon absorption,
      else
if(r1<APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][2][ie])
        {
          ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwo;
          // 20

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
```

```
ole.iv]*ef));
        f=2.0*ki*kf/(ki-kf)/(ki-kf);
        if(f<=0.0){return;}
          randomnumber=float(rand())/float(RAND_MAX);
          cb=(1.0+f-pow((1.0+2.0*f),double(randomnumber)))/f;
                  //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
               +ScatHole.ky*ScatHole.ky);
          a11=ScatHole.ky/skk;
          a12=ScatHole.kx*ScatHole.kz/skk/ki;
          a13=ScatHole.kx/ki;
          a21=-ScatHole.kx/skk;
          a22=ScatHole.ky*ScatHole.kz/skk/ki;
          a23=ScatHole.ky/ki;
          a32=-skk/ki;
          a33=ScatHole.kz/ki;
          x1=kf*sb*cf;
          x2=kf*sb*sf;
          x3=kf*cb;
          ScatHole.kx=a11*x1+a12*x2+a13*x3;
          ScatHole.ky=a21*x1+a22*x2+a23*x3;
          ScatHole.kz=a32*x2+a33*x3;
          ScatHole.e=ef;
          return;
        }

      //X valley Nonpolar optical phonon emission
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][3][ie])
        {
          ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          if(ef<=0.0){return;}
          //     goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //X valley Nonpolar optical phonon absorption
```

```
      else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][4][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwe;
          //goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //X --> Gamma, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][5][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1];
        if(ef<=0.0){return;}
        ScatHole.iv=1;
                    //    goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //X --> Gamma, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][6][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[1];
```

```
        if(ef<=0.0){return;}
        ScatHole.iv=1;
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
        }

      //X --> L, emission
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][7][ie])
        {
        ef=ei-APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2];
        if(ef<=0.0){return;}
        ScatHole.iv=2;
                    //    goto 40
          //40 ...

kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
            }

      //X --> L, absorption
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][8][ie])
        {
        ef=ei+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].hwij
            +APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0]
            -APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[2];
        if(ef<=0.0){return;}
        ScatHole.iv=2;
                    //goto 40
          //40 ...
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]
```

```
*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;
        }

      //acoustic phonon  X valley
         else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][9][ie])
        {
          ef=ei;
          kf=ki;
          //  goto 40
          //40 ...
   kf=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].smhH[ScatHole.iv]

*sqrt(ef*(1.0+APDdevice.DeviceLayers[APDdevice.index[PosIndex]].afH[ScatH
ole.iv]*ef));
          randomnumber=float(rand())/float(RAND_MAX);
          cs=1.0-2.0*randomnumber;
          sn=sqrt(1.0-cs*cs);
            randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;
          ScatHole.kx=kf*cs;
          ScatHole.ky=kf*sn*cos(fai);
          ScatHole.kz=kf*sn*sin(fai);
          ScatHole.e=ef;
          return;

         }

      //impurity scattering X valley
        else
if(r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][10][ie])
        {
          ef=ei;
           r2=float(rand())/float(RAND_MAX);
           cb=1.0-r2/(0.5+(1.0-
r2)*sk/APDdevice.DeviceLayers[APDdevice.index[PosIndex]].qd2);
           kf=ki;
         //    goto 30
                 //30
         sb=sqrt(1.0-cb*cb);
          randomnumber=float(rand())/float(RAND_MAX);
          fai=2.0*pi*randomnumber;

          cf=cos(fai);
          sf=sin(fai);
          skk=sqrt(ScatHole.kx*ScatHole.kx
```

```
              +ScatHole.ky*ScatHole.ky);
            a11=ScatHole.ky/skk;
            a12=ScatHole.kx*ScatHole.kz/skk/ki;
            a13=ScatHole.kx/ki;
            a21=-ScatHole.kx/skk;
            a22=ScatHole.ky*ScatHole.kz/skk/ki;
            a23=ScatHole.ky/ki;
            a32=-skk/ki;
            a33=ScatHole.kz/ki;
            x1=kf*sb*cf;
            x2=kf*sb*sf;
            x3=kf*cb;
            ScatHole.kx=a11*x1+a12*x2+a13*x3;
            ScatHole.ky=a21*x1+a22*x2+a23*x3;
            ScatHole.kz=a32*x2+a33*x3;
            ScatHole.e=ef;
            return;
          }

  //Impact Ionization
       else
if((r1<=APDdevice.DeviceLayers[APDdevice.index[PosIndex]].swkH[0][11][ie]
)||(ScatHole.e>(APDdevice.DeviceLayers[APDdevice.index[PosIndex]].EacthH+
APDdevice.DeviceLayers[APDdevice.index[PosIndex]].Ev[0]+APDdevice.DeviceL
ayers[APDdevice.index[PosIndex]].Ec[1])))
          {
            ScatHole.impact=1;
            return;
          }
   }// X valley block
}//void...
```

DataToFile.C

```
#include "apd.h"
/*****************************************
Output the data
*****************************************/
void DataToFile(device& APDdevice, stats& APDresults, params& APDparams)
  {
    int i,j;
    // output data that varies as a function of position
```

```
       ofstream out_file;
       out_file.open(APDparams.structfilename);
       if (out_file.fail())
              {
                cout << "no output filename?" << endl;
                    exit(1);}
       cout << endl << "                         # Saving data, be
patient!" << endl;
       out_file <<"position      "<<"abs(Doping)     "<<"EField    "
              <<"e-impact       "<<"h-impact     "
                  <<"          "<<"Absorption   "
              <<"Ec " << "Ev " << endl;
       for (i=0; i<=NUMX; i++) {//loop through different positions

         out_file  <<i*APDdevice.TotalThickness/NUMX<<"     "

<<log10(fabs(APDdevice.DeviceLayers[APDdevice.index[i]].Doping)+1.0)<< "
"
                     <<APDdevice.EFTable[i] <<"         "
                   <<APDresults.ImpactDist_e[i] <<"        "
                   <<APDresults.ImpactDist_h[i] <<"        "
                     <<APDresults.AbsDist[i] <<"          "
      <<APDdevice.Vapp-
APDdevice.Potential[i]+APDdevice.DeviceLayers[APDdevice.index[i]].Ec[1]
      << "  "
         <<APDdevice.Vapp-APDdevice.Potential[i]-
APDdevice.DeviceLayers[APDdevice.index[i]].Ev[1];
         out_file  <<endl;
       }//end for
       out_file.close();

//output data that vary with the bias --gain, gain^2, and bw
 ofstream gain_out_file;
 gain_out_file.open(APDparams.gainfilename);
if(gain_out_file.fail())
  {exit(1);}
 gain_out_file<<setprecision(4);
 gain_out_file<<"          "<<"bias    "
            <<"          "<<"gain    "
              <<"          "<<"F(M)    "<<endl;
 for (int step=1;step<=APDresults.max_steps;step++){

   gain_out_file <<APDresults.Vapp[step]<<"  "
      //<<APDdevice.DepletionEnd-APDdevice.DepletionStart<<" "
          <<APDresults.gain[step]<<"  "
          <<APDresults.sq_gain[step]/
        APDresults.gain[step]/APDresults.gain[step]<<"  "<<endl;
 }//end for
 gain_out_file.close();

 ofstream distr_out_file;
 distr_out_file.open(APDparams.distrfilename);

 //save the histogram of the distribution of gain
 distr_out_file<< endl << "  m   # of carriers     log10(#)"<<endl;

 float temp;
```

```
 for (j=1;j<=MAX_COUNT/APDparams.stat_bin;j++)
   {
     temp=0.0;
      for (i=1;i<=MAX_COUNT;i++)
      {
         if(i<=j*APDparams.stat_bin&&i>(j-1)*APDparams.stat_bin)
        {temp=temp+APDresults.GainDist[i];}
        }

      distr_out_file<<" "<<j*APDparams.stat_bin<<"    "<<temp
                     <<"                "<<log10(temp)<<endl;
   }//end for j
 distr_out_file.close();

 ofstream carrier_out_file;
 carrier_out_file.open(APDparams.carrierfilename);
 //save the (average)carrier travelling through the device, energy and
velociy

 for(j=0;j<=NUMX;j++)
 {
   //cout<<"j="<<j<<endl;
  carrier_out_file<<" "<<j*APDdevice.TotalThickness/float(NUMX)<<"  "<<
   APDresults.EnergyDist_e[j]/APDresults.CountDist_e[j]<<
    "  "<<APDresults.VelocityDist_e[j]/APDresults.CountDist_e[j]<<"
"<<APDresults.EnergyDist_h[j]/APDresults.CountDist_h[j]<<"
"<<APDresults.VelocityDist_h[j]/APDresults.CountDist_h[j]<<"
"<<float(APDresults.GDist_e[j])/APDresults.CountDist_e[j]<<"
"<<float(APDresults.LDist_e[j])/APDresults.CountDist_e[j]<<" "
              <<1.0-
float(APDresults.GDist_e[j])/APDresults.CountDist_e[j]-
    float(APDresults.LDist_e[j])/APDresults.CountDist_e[j]<<" "
        <<float(APDresults.GDist_h[j])/APDresults.CountDist_h[j]<<"
"<<float(APDresults.LDist_h[j])/APDresults.CountDist_h[j]<<" "
              <<1.0-
float(APDresults.GDist_h[j])/APDresults.CountDist_h[j]-
float(APDresults.LDist_h[j])/APDresults.CountDist_h[j]
              <<endl;
  }
 carrier_out_file.close();

 return;
  }// end DataToFile
```