COMPUTATIONAL STUDIES OF ELECTRON TRANSPORT IN NANODEVICES
by
Taran Villoch Harman
A thesis project submitted in partial fulfillment of the requirements for the degree of
Bachelor of Arts, Honors
Ball State University
2001
Approved by ______________________
Advisor
Program Authorized to Offer Degree
Date

______________________________
Ball State University

Abstract

CONDENSED MATTER
RESEARCH

by Taran Villoch Harman

Faculty Advisor: Professor Dr. R.M. Cosby
Department of Physics and Astronomy

This thesis project is the culmination of three years of research in Condensed Matter Physics, under the guidance of Dr. Cosby, with the support of an Honors Undergraduate Fellowship. The thesis project is a portfolio of presentations on the research, and a guide to a future research assistant in this area. The project includes a brief overview of the theory used to describe electron transport through nanometer-scale systems, as well as extensive descriptions of the computer programs used to calculate conductance, and a brief journal outlining the progress of this research and this researcher over the past three years.
ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Ronald Cosby who has been a teacher and a mentor to me since my very first day at Ball State University. His advice and encouragement made all of this work possible.

I would also like to thank my family and my husband, Doug, for love and support through the past four years and in all things in life.

Thanks to Dr. Ober, Jana Stockum, Kevin Tajkowski, and everyone in the Department of Physics and Astronomy for making this a fun place to work, an easy place to learn, and a good place to grow.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Glossary</strong></td>
<td>iv</td>
</tr>
<tr>
<td><strong>Chapter 1: INTRODUCTION</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Goals and Objectives</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Theory and Modeling</td>
<td>1</td>
</tr>
<tr>
<td><strong>Chapter 2: PROGRAMS</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Setting up a Project using Digital Visual Fortran</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Input Data Acquisition</td>
<td>5</td>
</tr>
<tr>
<td>2.2.1 ProcessNanoInput</td>
<td>5</td>
</tr>
<tr>
<td>2.2.2 ProcessParaInput</td>
<td>7</td>
</tr>
<tr>
<td>2.3 ComputeConductance</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Data Interpretation Program</td>
<td>33</td>
</tr>
<tr>
<td><strong>Chapter 3: PRESENTATIONS</strong></td>
<td></td>
</tr>
<tr>
<td>3.1 Butler Undergraduate Research Conference 1999</td>
<td>35</td>
</tr>
<tr>
<td>3.2 Ball State University Student Symposium 1999</td>
<td>38</td>
</tr>
<tr>
<td>3.3 Ball State University Student Symposium 2000</td>
<td>39</td>
</tr>
</tbody>
</table>
3.4 American Association of Physics Teachers Meeting 2000 39

3.5 Texas American Physical Society Meeting 2000 42

Chapter 4: PROJECT JOURNAL 44

Bibliography 60

Index of Data Files 62
GLOSSARY

2DEG - the two-dimensional electron gas at the interface of a GaAs/AlGaAs structure through which electrons can propagate.

AlGaAs - Aluminum Galium Arsenide

Angstrom (A) - $10^{-10}$ meters

Conductance (G) - the inverse of resistance. Resistance is the tendency of physical properties of a material to inhibit current flow through the material. Conductance is directly proportional to transmittance.

Fermi Energy - a measure of the density of electrons. As electrons are added to an energy band, they will fill all available states, starting in the lowest band and going up in energy. The Fermi energy is the maximum state that the electron fill up to at absolute zero.

GaAs - Galium Arsenide

Nanometer (nm) - $10^{-9}$ meters
INTRODUCTION

1.1 Goals and Objectives
As the need for faster and smaller technology increases, it has become necessary to investigate electronic structures that have dimensions so small that the size of the electron wavelength can no longer be ignored. In this case, electron transport must be modeled using quantum mechanics. It is the goal of this project to investigate electron transport in semiconductor nanostructures. To do this, the devices are modeled using computer programs designed and adapted for this project. After investigation of numerous parameters, the device that shows the most interesting and useful results is the Notched Electronic Stub Tuner (NEST). It has become an objective to study the electron flow pattern throughout the two dimensional structure, and to that end, extensive modifications to the computer program have been made.

1.2 Theory and Modeling
The system studied is a model of the two-dimensional electron gas (2DEG) at the interface of an AlGaAs/GaAs heterostructure. We look at the electron transmission through the 2DEG in order to learn the quantum impacts on electronic device operation and develop futuristic electronic devices. The computer program is called ComputeConductance, and it is written in FORTRAN computer code. The program uses a tight-binding, recursive Green's function method to compute the device conductance as a function of electron energy, or device parameters. Figure 1.1 shows the theoretical AlGaAs/GaAs
structure. The 2DEG is formed at the interface of the two materials. If we put a metallic base on the structure and metal contacts on the top, we can negatively bias the top contacts and create a depletion zone in the 2DEG below, thereby shaping the structure. The computer program assumes a hard-wall case, where no electrons stray below the contacts.
Chapter 2

PROGRAMS

2.1 Setting up a Project using Digital Visual Fortran

This program is the environment in which one can use the ComputeConductance and related programs. The purpose of this document is to provide some helpful information to anyone who has never used the FORTRAN Developer Studio.

The first thing to do is to create a project workspace. This is not a difficult process. From the File menu, choose, New. A dialog box pops up. Click the Workspaces tab, name the workspace, and choose its location. Then click OK.

Now add projects to the workspace. ComputeConductance requires three projects, one for ComputeConductance, one for ProcessNanoInput, and one for ProcessParaInput. To insert projects into a workspace, go back to File/New, and this time choose the Projects tab. Choose the type of project desired, name it, choose its location and be sure to click Add to current workspace.

Once the three projects are created, one can begin adding files to them. Under the Project menu, choose Set Active Project, and set the project where the files need to go as the “active project”. One can also right click on the project in the File View window and choose Set as Active Project from that menu. Then go to Project/Add to Project/Files. Browse for the program files and insert them into the project. Check in the File View window that all of the necessary external dependencies are satisfied (ComputeConductance needs ParaDataStruct, NanoDataStruct, and Constants). These are usually attached to the FORTRAN program file, but if not,
just add them like the program file. Once the three projects have all the necessary files, the next step is to build the projects. Change the active project to the one to build. Then from the Build menu, choose Build. This will compile and link the file and make it ready to execute. If there is an attempt to execute without building, DVF will just send up a dialog box, saying, “one or more of these projects are out of date or do not exist. Would you like to build them?” Clicking Yes will build the program and then execute it. If it is likely that the program has errors, or was recently changed, then choose Compile first, and make sure that there are no compiler errors before moving on to linking and executing. There are shortcut buttons on the toolbar for most of the Build menu. (If they are not, one can right-click the tool bar and get them.)

Compiling tips: At the bottom of the screen is the Build window. This is where DVF says out how many errors the program has, and what they are. Once the program finishes compiling and the error information is there, scroll up to see details about each error, including the line number where they occur. Double-clicking on the line number makes an arrow point to the line in the text window (the largest window). If there are unfamiliar terms in the error description, highlight a term and press F1, and the Help will look up the term. The Help dialog box will pop up automatically with the result of the search.

Once all of the projects have been built, they are ready to execute.

If there are run-time errors when the program executes, use the debugging feature of DVF. Insert breakpoints in the program (places where the program stops running during the debug) by going to the line in the program in the text window and clicking the little hand icon on the build toolbar, or choosing Insert Breakpoint from the Build menu. Then, when the breakpoints are set, go to the Build menu and choose Debug/Go. The program will run to the first breakpoint
and stop. The window on the bottom left of the screen shows the values for the variables in the program, this is helpful if there is an array bounds or variable type error. The values that were changed most recently in the program are in red. To continue, click Go on the toolbar or on the Debug menu to proceed to the next breakpoint, or you can use the Step features to execute the program line-by-line.

2.2 Input Data Acquisition

2.2.1 ProcessNanoInput
This program is easy to operate, just choose from the menus and name the file when prompted. Make sure that there is a new name for each file, this program will not write over an existing file. To fix a file, change things using the “Modify an existing file” feature, but then save the file as something different, and then delete the old file and rename the new one in Windows Explorer. The following is the information this programs requires.

The effective Mass Parameter will be read into the file as NanoData.MassPara, which is a double-precision (real) value. The effective mass for the AlGaAs/GaAs heterostructure is .067 (in units of the electron mass), which is the value used in all files in this thesis.

The Tight-binding Lattice Parameter is read into the file as NanoData.TBLatPara, which is also a double-precision value. It is in units of Angstroms \((10^{-10} \text{ meters})\) and it has been found that the data converges at a value of .1 A for the electron energies used in the notched electronic stub tuner project. Keep this value in mind when deciding the number of lattice sites for a particular slice. At this value, 100 sites makes the width (x-dimension) of the slice 10 A, or 1 nanometer.
The **Number of Slices** is read into the file as NanoData.NumOfSlices and must be an integer. This is simply the number of slices for the structure. A typical structure has a slice for the semi-infinite lead, one to five slices for the structure and another semi-infinite slice. The number of slices can be much higher, however, and is limited only by array dimensioning in the program.

For each slice, enter the following:

The number of **Lattice Sites** in the slice. This is read as NanoData.Slice(SliceIndex).NumOfSites. This must be an integer between 1 and 9999 (determined by an array dimension in the program). Keep the Lattice Parameter in mind when declaring this value.

The **Hard Wall Minimum Y-value** for the slice. This is read into the file as NanoData.Slice(SliceIndex).YMin and is a double-precision value, in Angstroms. This is the lower bound of the slice. The **Hard wall Maximum Y-value** is the upper bound of the slice.

Next, the program will ask if there is a base potential in the slice, and if there is, enter the **Normalized Base Potential** (NanoData.Slice(SliceIndex).NormBasePotential). The default value for this is zero.

When asked, enter the number of impurities in the slice. If there are none (enter 0), the program will move on to the next slice. If there are impurities, then initialize the following information about the impurity.

The **Impurity Minimum Y-value** will be stored in the program as NanoData.Slice(SliceIndex).Imp(ImpurityIndex).ImpYMn. (In the Compute Conductance Program as it stands, only one Impurity is allowed, so the Impurity
Index will always be 1 in that program.) This is the minimum value (in Angstroms) of the impurity in the Y-direction; this should be greater than the minimum Y-value for the slice that contains this impurity (and there is a check for this in the ComputeConductance program).

The Impurity Maximum Y-Value is stored in the program as NanoData.Slice(SliceIndex).Imp(ImpurityIndex).ImpYMax. This is also in Angstroms and should be less than the maximum y-value for the slice that contains this impurity.

The Normalized Impurity Potential is stored as NanoData.Slice(SliceIndex).Imp(ImpurityIndex).NormImpPotential. If the impurity is attractive, this will be a value less than zero. As it stands, the program does not do multiple impurities and it does not do any repulsive impurities, other than an infinite potential impurity.

This is the end of the input file. Name the file, and the program will exit.

2.2.2 ProcessParaInput

The opening menu for this program is similar to the one for ProcessNanoInput. The most common choice is to create a new parameter data file. The program will ask first what kind of calculations the user wants it to do. The Conductance Vs. Geometry part does not work yet, so the default choice and the only one that works is choice 1, Conductance Vs. Fermi Energy. When prompted, input the following parameters for the structure.

The Number of Data Points to be considered is entered into the program with the variable name ParaData.NumOfDataPts. In most cases, this value is set to at least 500 data points. If the user is changing the information about a slice and wants to
look at only one energy level, part of the program manipulation is to choose 1 for
the number of Data Points, and run the program through 200 or so iterations.

Next is the number of Transverse Modes. This is stored in the program as ParaData.
NumOfModes, and is important in all of the calculations in ComputeConductance. It was found that the data converges at 60 modes, so in
most cases, this is what is used.

Then, enter the Number of Propagating Modes, which becomes
ParaData.NumOfPropModes. This is (as the program says) the number of
conductance plateaus wanted. The number of propagating modes should be
much less than the number of transverse modes. Depending on the
nanostructure and the project, one to twelve propagating modes has been a
typical number.

Next, the program asks if there is a particular Fermi Energy Interval to be
considered. If yes, enter the values desired, and they will be stored in the program
as ParaData.NormMinEf and ParaData.NormMaxEf. All of the conductance
calculations will be performed on this Fermi Energy interval if specified. The
default is to start at zero and do as many conductance plateaus as there are
propagating modes.

Next, enter the X-Tolerance and then the F-Tolerance. These are stored as
respectively. In this work, the values were left at 0.1 D-19 for both of them. (That
is how the program reads zero point one times ten to the negative nineteenth.)
The last thing prompted for is the Maximum Number of Iterations. Again, the only
value used for this was 125. This is stored in the program as
ParaData.NumericalPara.MaxItr. The last three discussed parameters are used in
the solution of transcendental equations for finding transverse mode energy
eigenvalues when an attractive impurity is present. Thus, these parameters were not important in this thesis work.

After all of the data is entered, name the file, as in Process NanoInput, and the program returns to the main menu where you can exit the program.

Once the input data files are created, a few things must be done in order for ComputeConductance to recognize them. Using Windows Explorer, look in the folder that contains the selected DVF project workspace. The files, as they were named in the Process____Input Programs, are in the folders for the Process____Input programs. These need to be moved or copied to the folder for ComputeConductance, and then renamed (the one from ProcessParaInput) “ParaData.Input”, and (the one from ProcessNanoInput) “NanoData.Input”. Once this is done, return to the DVF environment and execute the program ComputeConductance. After the ComputeConductance program finishes executing, you can find the output file in the folder for the ComputeConductanc project. Windows Explorer is a valuable tool in this process and it is worthwhile to learn to use it effectively.

2.3 ComputeConductance

The ComputeConductance program does not require any more input from the user. The output to the monitor consists of three statements saying what it is doing, “READING the nanostructure information from the file ‘NanoData.Input’”, “READING the executions parameters from the file ‘ParaData.Input’”, and “COMPUTING the conductance as a function of Fermi
energies...". There are several versions of the program, this is the information on the basic, original program. The following description of the program may be more detailed than necessary, it is written for someone who might have to make changes to the FORTRAN code.

The main program `ComputeConductance` calls four subroutines, which then call subroutines, which then call subroutines, which call subroutines, which call subroutines and functions. The general organization is in that order: main program, what the author will call first-call subroutines (in the order they are called), second-call subroutines (in the order they are called), third-call subroutines (in the order they are called), fourth-call subroutines (in the order they are called), then there is one fifth-call subroutine and then the functions. One subroutine, `WARN`, is called from more than one level; it is written between the second- and third-call subroutines. This subroutine is an infinite loop that forces the user to abort the program because a fatal error has occurred.

This section is written as an attempt at a guide to the program. For that purpose, this will be written in the order that the program runs, as if the reader is a piece of information going through the program.

**ComputeConductance**

The very first line is "IMPLICIT NONE", which gets rid of some assumptions that are left over from older versions of FORTRAN. This is followed by some declaration and include statements. All of the subroutines will begin this way. `ComputeConductance` "includes" `NanoDataStruct` and `ParaDataStruct`, which deal with the input files. CC declares the new data file name "NormG(NormEf).DATA" which will be the output file. It then sets the variables X and Y equal to zero.
The next statement is a CALL statement for the NanoData input file, which has to be named "NanoData.Input" to be recognized by the program.

*ReadFromNanoDataFile*

**INPUT:** NanoData (input file)

**OUTPUT:** none

This subroutine takes the file "NanoData.Input" and reads the information stored in that file. It begins with the usual statements. Then it prints to the screen “READING the nanostructure information from the file ‘NanoData.Input’”. Then it reads the data from the file. It reads the number of slices, the effective mass, and the tight binding lattice parameter according to the format line 110. Then it has a do loop that runs for the number of slices, in order to read the slice information. The slice data is read according to format line 210. The important thing to remember on the format lines is to be sure that it reads an integer when it says to read an integer. The whole program will stop and have to be aborted if, in the input program, the user accidentally puts a decimal in the number of data points for example. This is the place where those mistakes get caught, not in the input programs. Format line 110 expects an integer, then two decimal numbers. Format line 210 expects three decimals, then two integers. There is a third format line (310) in the case that the slice has an impurity. Then, the subroutine closes the data file and ends. All programs and subroutines in Fortran have to end with an END statement. Attention goes back to the main program. It next calls the subroutine ReadFromParaDataFile.

*ReadFromParaDataFile*

**INPUT:** ParaData (input file)

**OUTPUT:** none
This subroutine is located after ReadFromNanoDataFile in the code. This subroutine reads the parameter input file, which must be named "ParaData.Input". It is so similar to ReadFromNanoDataFile that it is not worthwhile to go through it.

Back in Compute Conductance, the subroutine ComputeConductanceVersusFermiEnergy is called. This subroutine is the data-generating program. The only subroutine called after it is the subroutine that writes to the data file.

ComputeConductanceVersusFermiEnergy

INPUT: NanoData, ParaData

OUTPUT: NormEF (normalized Fermi Energy), NormG (normalized conductance)

This subroutine follows ReadFromParaInputFile in the code. It starts with IMPLICIT NONE and INCLUDE statements, then declares REAL*8 and COMPLEX*16 variables. Then it prints a message to the screen. Then calls its second-call subroutines. The first is ComputeEfNormFactor.

ComputeEfNormFactor

INPUT: NanoData

OUTPUT: EfNormFactor (Fermi energy normalization factor)

As a second-call subroutine, this subroutine follows WriteToGDataFile (the last first-call subroutine) in the code. In this subroutine, a Do Loop iterates over the number of slices in the structure to determine the slice with the smallest width (in
the y direction). It then computes the normalization factor based on this minimum slice width. The code looks like this:

\[ Ef\text{NormFactor} = \frac{\text{DACOS}(-1.0D0) / \text{MinSliceWidth}^{**2} / (\text{NanoData.MassPara} * \alpha \text{prime})}{\text{DACOS}(-1.0D0) / \text{MinSliceWidth}^{**2} / (\text{NanoData.MassPara} * \alpha \text{prime})} \]

The decimal Arccosine of -1 is \(\pi\). In the comment above this line of code, we see that the Ef Norm Factor is given by the following equation:

\[ Ef\text{NormFactor} = \frac{\pi}{\text{MinSliceWidth}^{**2}} \]

The value for \(\alpha\) can be found in “Constants”

After the EfNormFactor is computed, the program goes back to \(\text{ComputeConductanceVersusFermiEnergy}\), which then calls \(\text{ComputeTransverseEnergy}\).

\(\text{ComputeTransverseEnergy}\)

**INPUT:** NanoData, ParaData, EfNormFactor  
**OUTPUT:** E1 (transverse energy), k1(wave vector), k2(wave vector)  
This subroutine declares the variables E1, k1 and k2, then initializes them all to zero. It has a Do Loop that iterates over the number of slices. For each slice, it determines the number of impurities in the slice. It then calls its third-call subroutines to compute the transverse energy for the case with no impurities (TransverseEnergyCase1), and the case with a single attractive impurity (TransverseEnergyCase2a). This set-up leaves allowance for a case with a single repulsive impurity.

If the number of impurities in the slice equals zero, the subroutine TransverseEnergyCase1 is called.
TransverseEnergyCase1

INPUT: NanoData, SliceIndex (number of the current slice), EfNormFactor
OUTPUT: E1, k1
As a third-call subroutine, this subroutine is after the subroutine WARN in the code. WARN is a second- and third-call subroutine. Within the subroutine TransverseEnergyCase1, the program makes sure the base potential is a positive real number. If it is not, WARN is called. Then, it sets the temporary variable “temp” as Pi over the slice width. Then, for each mode (a Do Loop), k1 is set as the mode number (made into a decimal value by the function DFLOAT) times Pi over the slice width (temp). Then the transverse energy is calculated by the code:

\[ E1(SliceIndex,:) = \frac{k1(SliceIndex,:)^2}{(NanoData.MassPara * alpha prime) + EfNormFactor * NanoData.Slice(SliceIndex).NormBasePotential^2} \]

Where the colon in the parentheses indicates that the entire declared range for the dimension is used in the array. In this case, the colon represents the number of modes. The equation for this is

\[ E1 = \frac{(k1)^2}{(Mass)(\alpha')} + (EfNormFactor)(NormBasePotential)^2 \]

which is the same as

\[ E1 = \frac{k1^2}{\alpha} + BasePotential \]

If the number of impurities in the slice is one, TransverseEnergyCase2a is called.

TransverseEnergyCase2a

INPUT: NanoData, ParaData, SliceIndex, EfNormFactor
OUTPUT: E1, k1, k2
In this case, there are considerably more variables to consider. At the beginning of the subroutine, they are divided into three sections. Under “Input” are the parameters from the input files. Under “Intermediary” are the variables that are determined for the slice in question. These are: \(a, c, d, e, \alpha, \beta, \theta_1, \theta_2, \theta_3\), and the Mode. Under “Output” are \(E_1, k_1, \) and \(k_2\). After the declaration statements, the variables are initialized.

\(a\) is the minimum \(y\) value of the slice.

\(c\) is the minimum \(y\) value of the single impurity.

\(d\) is the maximum \(y\) value of the impurity.

\(e\) is the maximum \(y\) value for the slice.

It is assumed that \(a < c < d < e\), and there is a check for this. The thetas are determined from \(a, c, d, e, \) and \(\alpha\) is determined from MassPara (from input files) and \(\alpha\) prime (which is in “Constants”). \(\beta\) is \(\alpha\) times the \(E_{fNORMFACTOR}\) times the impurity potential squared.

With all this information, the subroutine calls the fourth-call subroutine \textit{SolveEqns}.

\textit{SolveEqns}

\textbf{INPUT:} ParaData, \(\beta\) (related to a constant, calculated in TECase2a), \(\theta_1, \theta_2, \theta_3\) (these are regions of the slice, based on dimensions, calculated in TECase2a)

\textbf{OUTPUT:} \(k_1\) (called \textit{Roots})

This subroutine is located after \textit{ComputeTransmittance} in the code. It is the first of the fourth-call subroutines. After the requisite IMPLICIT NONE statement is a paragraph of comments regarding the parameters in this subroutine. The user can read those. Following those comments are the Parameter, Include, and Declaration statements. This is the first place where logical variables are used.
The RootIndex is initialized to 1, and the MoreRoots variable is initialized to TRUE. The first thing we want to do is solve the function $f \ E_\ n$ in order to get $nk_1$, which is the wave vector of a bound state. We set FCNTyp to $-1$, $Stop \ n$ to the square root of $\beta$ ($\beta$ is defined above), and $Start$ to 0.0. Then we want to set the step size for $n$. $Stop \ n - Start$ is the zone $n$ containing all of the bound states. So, the step size is set as: $Step \ n_1 = \frac{\text{the zone } n}{\text{NumIntvl } n}$ (defined as a parameter in the code as 107). Then we refine the step size for the function when necessary. So, $Step \ n_2$ is defined as whichever is smaller, $Step \ n_1$, or the ratio $FineZone \ n / FineZoneNumIntvl \ n$ (defined as a parameter in the code as 0.0157/214). Then the variable $Right$ is defined as $Stop \ n - Step \ n_2$. After this, the variable $jR$ gets the result of the function $f \ E_\ n$ (which is given $Right$, $\beta$, and the $\theta$es.)

\[ f \ E_\ n \]

This is the first function, located after the final subroutine, ModifiedRegulaFalsi. According to the comments in the program, the tow function $f \ E_\ n$ and $f \ E_\ p$ define the transcendental equations in the case of the single finite impurity. The function is given as follows:

\[ nk_2 = \sqrt{(-nk_1)^2 + \beta} \]

\[ f \ E_\ n = \left(2.0 * (nk_1)^2 - \beta\right) \cdot \cosh(nk_1 * \theta_2) + \beta \cdot \cosh(nk_1 * \theta_1) \cdot \sin(nk_2 * \theta_3) - 2.0 * nk_1 \cdot nk_2 \cdot \sinh(nk_1 * \theta_2) \cdot \cos(nk_2 * \theta_3) \]

This is sent back to SolveEqns.
SolveEqns (continued)

It was called with the value Right as the \( nk1 \) value, so \( jR \) gets the result of this function with those values. The variable \( \text{SIGN}jR \) is 1 if \( jR \) is positive and \(-1\) if \( jR \) is negative. The variable \( \text{Left} \) equals \( \text{Right} \) minus \( \text{Step n1} \). Then there is a Do While \( \text{MoreRoots} \) is TRUE and \( \text{Left} \) is greater than \( \text{Start} \). Within this loop, \( jL \) gets the result of \( fEn \) with \( \text{Left} \) as \( nk1 \). \( \text{SIGN}jL \) is 1 if \( jL \) is positive and \(-1\) if \( jL \) is negative. Then the Boolean variable \( \text{SignChange} \) is TRUE if \( jR \) and \( jL \) have different signs and FALSE if they have the same sign. If \( \text{SignChange} \) is TRUE, then a probable \( a\text{Root} \) is discovered. If this is the case, \( \text{Left} \) is redefined as \( a \), \( \text{Right} \) is redefined as \( b \), \( jL \) as \( fa \), and \( jR \) as \( fb \). Then \( \text{SolveEqns} \) calls \( \text{ModifiedRegulaFalsi} \) with those variables, as well as others from this subroutine and from input data.
**ModifiedRegulaFalsi**

This is the only fifth-call subroutine, so it is the last one before the functions in the code. According to the comments in the code, this subroutine uses a Modified Regula Falsi Algorithm to solve a nonlinear equation. The variable `FCNType` is a flag indicating which function should be used. The -1 is for the function \( f_E \) and +1 is for \( f_E \). \( x \) gets \( a \) and \( f_x \) gets \( f_a \). For iterations 1 through the maximum number given in the input file, the following Do Loop is executed. `SIGNjPRVSx` gets 1 if \( f_x \) is positive and -1 if \( f_x \) is negative. Then there is a check to see that the interval is small enough. `XERROR` gets the absolute value of \( a-b \). If this is less than the \( X\)-Tolerance specified in the input, then the value is returned to `SolveEqns`. `FERROR` gets the absolute value of \( f_x \). If this is less than the specified \( F\)-Tolerance, then the value is returned to `SolveEqns`. If we're still in this subroutine after those checks, \( x = \frac{fb \cdot a - fa \cdot b}{fb - fa} \), and \( f_x \) gets the value returned by calling the function \( f_E \) and \( f_E \) depending on `FCNType`. Then if \( f_a \) times the sign of the new \( f_x \) is less than zero, \( b \) gets \( x \), \( fb \) gets \( f_x \), and if `SIGNjPRVSx` times `SIGNf_x` is greater than zero, then \( fa \) is divided by two. If \( f_a \) times `SIGNf_x` not less than zero, then \( a \) gets \( x \) and \( fa \) gets \( f_x \); and if `SIGNjPRVSx` times `SIGNf_x` is greater than zero, then \( fb \) is divided by two. This is the end of the Do Loop and the end of the subroutine. So, we go back to `SolveEqns`.

**SolveEqns (cont.)**

Now in this subroutine, \( aRoot \) is TRUE if \( x \) is less than the \( X\)-tolerance, and \( x \) is less than \( Stop \), which would mean that \(-Vb < E < 0\). Then `Roots(RootIndex)` gets \(-x\) (which is \( nkf \)), and `RootIndex` is increased by 1. This is the end of the If statement that started with if `SignChange` is TRUE, but still part of the While Loop. Then `Right` gets the value for `Left` and `fR` gets the value for \( f_L \), and new left
values are determined. This is the end of the While Loop. Once that is finished, the whole thing is done again for the unbound (p) states. This time, the function \( f \) \( E \) \( p \) is called.

\[ f \ E \ p \]

\[ pk_2 = \sqrt{pk_1^2 + beta} \]

\[ f\text{En} = ((2 \cdot pk_1^2 + beta) \cdot \cos(pk_1 \cdot \theta_2) - beta \cdot \cos(pk_1 \cdot \theta_1)) \cdot \sin(pk_2 \cdot \theta_3) - 2 \cdot pk_1 \cdot pk_2 \cdot \sin(pk_1 \cdot \theta_2) \cdot \cos(pk_2 \cdot \theta_3) \]

That is all that is in this function, and that covers all of \( SolveEqs \), so the program returns to \( ComputeTransverseEnergyCase2a \)

\( ComputeTransverseEnergyCase2a \) (cont.)

When all of the values return from \( SolveEqs \), there is a Do Loop for the number of modes that performs the following:

If \( k_1 \) for that Slice and mode is less than zero, then \( k_2 \) (for that slice and mode) gets the square root of \( beta - k_1 \) squared, and \( E_1 \) for that slice and mode gets \( -k_1 \) squared over \( alpha \). If \( k_1 \) for that slice and mode is not less than zero, \( k_2 \) gets the same thing, but \( E_1 \) gets \( +k_1 \) squared over \( alpha \). That ends the If statement and the Do Loop and the subroutine. This sends the program back to \( ComputeTransverseEnergy \).

\( ComputeTransverseEnergy \) (cont.)
There is another statement in the If statement that calls the cases, for parameters that don't belong to either of the cases. In this case, the subroutine _WARN_ is called and the program goes into an infinite loop and has been aborted. This is the end of _ComputeTransverseEnergy_, so the program goes back to _ComputeConductanceVsFermiEnergy_.

 científico

**ComputeConductanceVsFermiEnergy (cont.)**

The next statement in this subroutine calls _ComputeWaveAmplitude_.

**ComputeWaveAmplitude**

**INPUT:** NanoData, k1, k2, NumOfModes (number of transverse modes, renamed from ParaData.NumOfModes)

**OUTPUT:** ampltdA, ampltdB, ampltdC, ampltdD (wave amplitudes)

This subroutine declares the variables ampltdA, ampltdB, ampltdC, and ampltdD and initializes them all to zero. Then it computes the wave amplitudes for each slice. There is a Do Loop for this that iterates over the number of slices. The first statement in the loop checks to see if there are impurities in the slice. If there are no impurities, the subroutine calls _WaveAmplitudeCase1_.

**WaveAmplitudeCase1**

**INPUT:** NanoData, NumOfModes, SliceIndex

**OUTPUT:** ampltdA
This subroutine is located after \textit{TransverseEnergyCase2a} in the code. It determines the slice width by subtracting the minimum y-value for the slice from the maximum y-value for the slice. Then \( a_{pltdA} \) for that slice and all the modes is:

\[
 a_{pltdA} = \sqrt{\frac{2}{\text{Slice Width}}}.
\]

That is the end of \textit{WaveAmplitudeCase1}.

\textit{ComputeWaveAmplitude (cont.)}

If there is a single attractive impurity in the slice, then this subroutine calls \textit{WaveAmplitudeCase2a}.

\textit{WaveAmplitudeCase2a}

\textbf{INPUT:} NanoData, NumOfModes, SliceIndex, \( k_1 \), \( k_2 \)
\textbf{OUTPUT:} \( a_{pltdA} \), \( a_{pltdB} \), \( a_{pltdC} \), \( a_{pltdD} \)

In this subroutine, some physical parameters have to be defined. \( c \) is defined as the min y-value for the impurity in the slice, \( d \) is defined as the max y-value for the impurity in the slice, \textit{region1} is the part of the slice between the min y-value for the slice and \( c \), \textit{region2} is the impurity, and \textit{region3} is the region between the max of the impurity and the max of the slice.

Then a Do Loop iterates over the number of modes. IF \( k_1 \) for that slice and mode is less than zero, \textit{nWaveAmplitude} is called.
**nWaveAmplitude**

**INPUT:** k1, k2, region1, c, region2, d, region3 (defined above)

**OUTPUT:** ampltdA, ampltdB, ampltdC, ampltdD

\[
nl_1 = \frac{\sinh(2 * nk_1 * region_1) - region_1}{4 * nk_1}
\]

\[
nl_2 = \frac{region_2}{2} + \frac{(\sin(2 * k_2 * d) - \sin(2 * k_2 * c))}{4 * k_2}
\]

\[
nl_3 = \frac{(\cos(2 * k_2 * c) - \cos(2 * k_2 * d))}{2 * k_2}
\]

\[
nl_4 = \frac{\sinh(2 * nk_1 * region_3) - region_3}{4 * nk_1}
\]

\[
n\alpha = \frac{nk_1}{k_2 \cdot \text{tanh}(nk_1 \cdot \text{region}_1)}
\]

\[
n\beta = \frac{\cos(k_2 \cdot c) - n\alpha \cdot \sin(k_2 \cdot c)}{\sin(k_2 \cdot c) + n\alpha \cdot \cos(k_2 \cdot c)}
\]

\[
n\epsilon = \frac{\sinh(nk_1 \cdot \text{region}_1)}{\sin(k_2 \cdot c) + n\beta \cdot \cos(k_2 \cdot c)}
\]

\[
n\epsilon = \frac{\sin(k_2 \cdot d) + n\beta \cdot \cos(k_2 \cdot d)}{\sinh(-nk_1 \cdot \text{region}_3)}
\]

\[
nA = \left( nl_1 + nl_3^2 * (region_2 + (n\beta^2 - 1) * nl_2 + n\beta * nl_3 + n\epsilon^2 * nl_4) \right)^{\frac{1}{2}}
\]
\[ nB = nZeta \times nA \]
\[ nC = nBeta \times nB \]
\[ nD = nEpsilon \times nB \]

That is all of \( nWaveAmplitude \).

**WaveAmplitudeCase2a (cont.)**

If \( k_1 \) is greater than zero, \( pWaveAmplitude \) is called.

**PWaveAmplitude**

**INPUT:** \( k_1, k_2, \text{region}_1, c, \text{region}_2, d, \text{region}_3 \) (defined above)

**OUTPUT:** \( \text{ampltd}_A, \text{ampltd}_B, \text{ampltd}_C, \text{ampltd}_D \)

\[ p11 = \frac{\text{region}_1}{2} - \frac{\sin(2 \times pk_1 \times \text{region}_1)}{4 \times pk_1} \]

\[ p12 = \frac{\text{region}_2}{2} + \frac{\left(\sin(2 \times k_2 \times d) - \sin(2 \times k_2 \times c)\right)}{4 \times k_2} \]

\[ p13 = \frac{\left(\cos(2 \times k_2 \times c) - \cos(2 \times k_2 \times d)\right)}{2 \times k_2} \]

\[ p14 = \frac{\text{region}_3}{2} - \frac{\sinh(2 \times pk_1 \times \text{region}_3)}{4 \times pk_1} \]

\[ pAlpha = \frac{pk_1}{k_2 \times \tan(pk_1 \times \text{region}_1)} \]

\[ pBeta = \frac{\cos(k_2 \times c) - pAlpha \times \sin(k_2 \times c)}{\sin(k_2 \times c) + pAlpha \times \cos(k_2 \times c)} \]
\[ pZeta = \frac{\sin(pk1 \cdot \text{region}1)}{\sin(k2 \cdot c) + pBeta \cdot \cos(k2 \cdot c)} \]

\[ pEpsilon = \frac{\sin(k2 \cdot d) + pBeta \cdot \cos(k2 \cdot d)}{\sin(-pk1 \cdot \text{region}3)} \]

\[ pA = \left( pI1 + pZeta^2 \cdot \text{region}2 + (pBeta^2 - 1) \cdot pI2 + pBeta \cdot pI3 + pEpsilon^2 \cdot pI4 \right)^{\frac{1}{2}} \]

\[ pB = pZeta \cdot pA \]
\[ pC = pBeta \cdot pB \]
\[ pD = pEpsilon \cdot pB \]

This is the end of \( p\text{WaveAmplitude} \).

This is also the end of \( \text{WaveAmplitudeCase2a} \), and if we return to \( \text{ComputeWaveAmplitude} \), we see that that is all there is to that one too, so we go back to \( \text{ComputeConductanceVsFermiEnergy} \).

**ComputeConductanceVsFermiEnergy (cont.)**

This subroutine calls \( \text{ComputeHoppingMatrix} \) next.

**ComputeHoppingMatrix**

**INPUT:** NanoData, k1, k2, ampltdA, ampltdB, ampltdC, ampltdD, NumOfModes

**OUTPUT:** t (hopping energy term), Vqp1q(hopping matrix from slice q to slice q plus 1), Vqqp1(hopping matrix from slice q plus 1 to slice q)
This subroutine follows a similar pattern to the transverse energy and wave amplitude subroutines in that it breaks into cases based on the number and nature of impurities in each slice. The slice in question is given the slice index variable \textit{Left} and the next slice is \textit{Right}. If there are no impurities in the left and there are no impurities in the right, then the subroutine calls \textit{Hopping Case 1}.

\textbf{HoppingCase1}

\textbf{INPUT:} NanoData, \textit{Left} (the slice index for the slice in question), \textit{k1}, NumOfModes, ampltdA

\textbf{OUTPUT:} V (temporary value for the hopping matrix)

This subroutine is located after \textit{WaveAmplitudeCase2a} in the code. The interaction matrix for two slices is in general

\[
\hat{V}_{21} = t \int_{y_{\text{min}}}^{y_{\text{max}}} \Psi_1(y)\Psi_2(y) dy
\]

In this particular case, we have two impurity-free slices. This subroutine, like the one before it, calls the slice in question \textit{Left} and the next slice \textit{Right}. It finds the maximum of the \textit{YMin} values for the two slices and the minimum of the \textit{YMax} values (to get the smallest width). The amplitude of the left slice, already calculated in \textit{ComputeWaveAmplitude}, and the right slice are given to the variables \textit{ampltdLeft} and \textit{ampltdRight}, respectively. The interaction matrix is calculated by a set of nested Do Loops. The first Do is over the number of modes in the left slice and the second Do is over the number of modes in the right slice, so that the matrix has the dimension \((\text{NumOfModes})^2\). For each place in the matrix, the program does the integral above, putting in the pieces of the wave functions that have already been calculated. For this case, the integral is \textit{Integral1}. 

25
\textit{Integral 1}

\textit{Integral 1} is a function and is located after $f E p$ in the code. The integral performed is

\[ \int_{Lower}^{Upper} \{ampltd1 \sin(k1(y - a1))\} \cdot \{ampltd2 \sin(k2(y - a2))\} \, dy\]

Where the 1's correspond to the values for the left slice and the 2's correspond to the values for the right slice. The value of the integral is then send back to the Hopping case.

\textit{HoppingCase1 (cont.)}

One the interaction matrix is calculated for the whole range of modes, the value is returned to the original Hopping subroutine.

\textit{ComputeHoppingMatrix (cont.)}

If there is an attractive impurity in the left slice and no impurity in the right slice, then this subroutine calls HoppingCase2a. If there is no impurity in the left slice and an attractive impurity in the right slice, then this subroutine still calls HoppingCase2a, but it trades the labels for the left and right slices, because it is the exact opposite case.

\textit{HoppingCase2a}

INPUT: NanoData, Left, Right (the slice index for the next slice), $k1$, $k2$, ampltdA, ampltdB, ampltdC, ampltdD, NumOfModes

26
This is, of course, right after \emph{HoppingCase1} in the code. The idea is the same, which is to compute the integral $\hat{V}_{21} = t \int_{\min}^{\max} \Psi_1(y)\Psi_2(y) \, dy$ for the interaction matrix. The difference is that the impurity complicates the wave function, more than the simple \emph{Integral1} is required to calculate the interaction matrix. No work that I have done deals with this case, and I am not experienced enough with the theory to explain this case or the next one.

\textit{ComputeHoppingMatrix (cont.)}

If there are attractive impurities in both of the adjacent slices, then the subroutine calls \emph{HoppingCase3a}. Again, I do not have any experience with the attractive impurities. Once the case subroutines calculate the \texttt{tempV}, \textit{ComputeHoppingMatrix} has the lines

\begin{verbatim}
Vqqp1(SliceIndex, :, :) = DCMPLX(t*tempV, 0.0D0)
Vqp1q(SliceIndex, :, :) = TRANSPOSE(Vqqp1(SliceIndex,:,:))
\end{verbatim}

The first is the hopping matrix from left to right ($V$ for $q$ to $q$ plus one) and the second is the hopping matrix from right to left ($V$ for $q$ plus one to $q$), which is the transpose of the left to right matrix. This is the end of \textit{ComputeHoppingMatrix}.

\textit{ComputeConductanceVsFermiEnergy (cont.)}
This subroutine calls ComputeEjPara next.

ComputeEjPara

INPUT: ParaData, NumOfSlices (the number of slices in the structure, renamed from NanoData.NumOfSlices), E1, EfNormFactor

OUTPUT: MinEf (the minimum Fermi energy), EfIncrmnt (the Fermi energy increment)

The input for this subroutine is the parameter data, \( E1 \) and \( Ef\)NormFactor. The output is the energy increment, \( Ef\)Incrmnt, and the minimum Fermi energy, \( MinEf \). If no particular Fermi energy level is specified in the ParaData.Input file, then ComputeEjPara uses the default \( MinEf \), which is 0.0D0. Otherwise, the \( MinEf \) is the \( Ef\)NormFactor (Fermi energy normalization factor) times the specified minimum Fermi energy, ParaData.NormMinEf squared.

If there is no specified Fermi energy level, then \( MaxEf \) is the largest value of \( E1 \) within the propagating range. If there is a specified Fermi energy, then \( MaxEf \) is the \( Ef\)NormFactor times ParaData.MaxEf squared. Then the \( Ef\)Incrmnt is the difference between \( MaxEf \) and \( MinEf \) divided by the number of data points.

ComputeConductanceVsFermiEnergy (cont.)

The final subroutine called is ComputeGVsEf.

ComputeGVsEf
INPUT: NanoData, NumOfDataPts (the number of data points to be calculated, renamed from ParaData.NumOfDataPts), NumOfModes, EfNormFactor, MinEf, EfIncrmnt, E1, t, Vqqp1, Vqp1q

OUTPUT: NormG (the normalized conductance), NormEf (the normalized Fermi energy)

We have been waiting for this subroutine. It takes all of the data from all of the calculations from all of the other subroutines and actually calculates the conductance as a function of the normalized Fermi Energy.

$Ef$ is initialized to zero. Then there is a Do Loop over the number of data points. For each point, $Ef$ gets the $MinEf$ plus the $EfIncrmnt$ times the current data point minus one. Then this subroutine calls ComputeGreensPropagator.

**ComputeGreensPropagator**

INPUT: NanoData, Ef (temporary variable calculated in ComputeGVsEf), E1, t, NumOfModes, Vqqp1, Vqp1q

OUTPUT: Gjp (the greens propagators)

This subroutine takes the Fermi energy that was just calculated and the transverse energy, and finds the Green’s function propagators from right to left through the nanostructure. The math and physics in this part of the program is at a higher level than most undergraduate physicists. Looking at this subroutine, one sees why we have a computer program to do this, and do not do it by hand.
`ComputeGreensPropagator` calls the following subroutines:

- `ComputeGB SI` - computes the Green's propagators for the rightmost semi-infinite slice.
- `ComputeGA F` - computes the Green's propagators for a left finite slice
- `ComputeGApB` - computes the Green's propagators for a composite slice of two separate slices
- `ComputeGA SI` - computes the Green's propagators for the leftmost semi-infinite slice.

The above subroutines are after `pWaveAmplitude` in the code.

`ComputeGVsEf (cont.)`

Once the Green's propagators are calculated, the next step is to find the transmittance.

`ComputeTransmittance`

**INPUT:** NanoData, t, Ef, Ei, Gji (Gjp), NumOfModes

**OUTPUT:** TnmSqr (the transmittance)
The basic equation for transmittance is:

\[
\text{Transmittance} = |t_{n,m}|^2 = t_{n,m}^* t_{n,m}
\]

In this subroutine, there is a Do Loop over the number of modes nested into another Do Loop over the number of modes. Inside both Do Loops, there are cases that depend on the type of mode, propagating or evanescent. Propagating modes are modes where the energy is below the Fermi energy and evanescent modes are modes where the energy is above the Fermi energy. The calculation changes slightly when one or both of the modes are evanescent. The matrix \( T_{nm} \) is \( |t_{n,m}|^2 \), the transmittance.

**ComputeGVsEf (cont.)**

The normalized conductance for the two-terminal (the nanostructure begins and ends with a semi infinite lead) is the trace of the transmittance matrix, so \( \text{NormG} \) for each data point is the sum of the transmittance matrix for that point.

**ComputeConductanceVsFermiEnergy (cont.)**

\( \text{ComputeGVsEf} \) was the last subroutine called from this subroutine, so it returns to the main program.

**ComputeConductance (cont.)**

The main program calls its last subroutine, \( WriteToG DataFile \).
**WriteToG DataFile**

**INPUT:** X (the normalized Fermi energy), Y (the normalized conductance), NanoData, NumOfDataPts

**OUTPUT:** writes X and Y to an output file

In the original version of the program, the format statement in the Write subroutine simply writes two real numbers with a space between them. For each data point, it writes the normalized Fermi energy and then the conductance. Then it returns to the main program.

**ComputeConductance (cont.)**

The main program prints "writing the results to the file... NormG(NormEf).Data" to the screen, and then closes the output data file. Then it prints to the screen "Exit PROGRAM ComputeConductance" and the user presses "Enter" to exit the program.

**2.3.2 Changes to Compute Conductance**

In most versions of the ComputeConductance program, the main program contains a Do Loop that causes the program to run multiple times, each time with one or two parameters of the NanoData file changed. There are several versions of this, not all permanently stored. The Do Loop must be changed, and the file "rebuilt", for each type of output that is desired. The versions can be put in three categories: "L", "Tab Y", and "Imp".
The "L" versions change the distance between the notch and the stub of the Notched Electronic Stub Tuner (NEST). The desired output is the whole conductance vs. Fermi energy curve for each value of L, the distance between the notch and the stub. The Do Loop for this comes after the call statement for ReadFromParaDataFile. It reads:

Do I = 1, (the number of values of L that are desired, e.g. 22)

Then it reads the call statement for ComputeConductanceVsFermiEnergy, then the call statement for WriteToGDataFile, then the statements changing the parameters, e.g. NanoData.Slice(the slice between the notch slice and the stub slice).NumOfSites = NanoData.Slice(the slice between the notch slice and the stub slice).NumOfSites +(the number of sites that the user wants to add)

2.4 Data Interpretation Program

QuattroPro

Efforts have been made to automate the process of graphing the output of Compute conductance, but they always take a lower priority to fixing up the programs and getting data. The result is, it is necessary to use an outside program such as Quattro Pro or Microsoft Excel to graph the output. Quattro Pro has been used in all data reduction for this thesis.

The output data file from ComputeConductance can be read as a notepad or WordPad file. The data appears to be in two columns, but it is not recognized as such in QP. Just copy and paste each set of data points into QP, and the go to
Notebook/Parse. The parse tool is easy to use. In the input box, type the addresses of the cells where the data is (e.g. A1...A551 the first column, rows 1 through 551), in the output box, type the addresses of the cells where the first column should be (usually the same address). Then click Create, and the format line appears, and the output values are shifted down one row to compensate. You should see something like this: »V»»»»»»V. Where the V's are at the two columns where the data will go. Then click OK. The data should be parsed into two columns. The format line and any other lines can be removed using the buttons on the toolbar. Tips for using QP: 1) label all of the data, and name the graphs before entering data values, because it is difficult to go back and name them later. 2) Yellow is a horrible color for a line on a white piece of paper, and yet it is the fourth color QP picks for lines. Use the mouse to select any of the lines and go up to Properties/Current Object on the toolbar. In this window, it will be possible to change many of the attributes of the lines and the graph.
Chapter 3

PRESENTATIONS

3.1 Butler Undergraduate Research Conference 1999

Link to presentation file

High Energy Electron Transport in Semiconductor Nanostructures

This is the documentation for the paper presented at the Butler Undergraduate Research conference in April of 1999. An Undergraduate Honors Fellowship supported the work. The report includes work done by the Theoretical Condensed Mater Research Group at Ball State University. The faculty and students in that group at the time of the presentation are credited on the title page of the presentation.

The first page gives an overview of the presentation. The report includes an introduction to the project, verification of the computer modeling programs, and discussion of modifications that need to be made to the programs to facilitate future work on the project.

The next page contains project goals and objectives. The goals are to determine if electrostatic potential reflectors may be used to control energetic electrons in a nanodevice, and to find out how they would affect the conductance resonance structure of a quantum well, or stub. To reach these goals, we need to verify the computer program results, generate data for the stub structure and the reflector
structure individually, determine at what energies the electros should be and what parameters the reflector should have for optimum results, and to modify the computer program to provide information on the conductance as a function of some physical parameter.

The next page shows some of the general information about the project. The field of study is in the electrical fields of nanometer scale devices. The system we study is a model of the two-dimensional electron gas (2DEG) at the interface of an AlGaAs/GaAs heterostructure. We look at the electron transmission through the 2DEG in order to learn the quantum impacts on electronic device operation and develop futuristic electronic devices.

The next page gives an overview of the electronic transport model that the computer program uses, and mentions some of the assumptions made. We assume that there is electron transport through the device in the longitudinal direction. We also assume hard wall conditions, that is, that there are no electrons outside the walls we determine for the structure. The computer program uses a nearest-neighbor tight-binding approximation and an iterative Green's function method in its calculation of the conductance through the device.

The next page shows a diagram of the AlGaAs/GaAs structure and its 2DEG channel. It is easy to see in the top picture that the contacts on the top of the structure are negatively biased with respect to the base of the structure. This causes a depletion zone in the 2DEG. The shape of the contacts will determine that shape of the channel that the electrons are able to go through. The bottom picture shows a top view of how a channel through the 2DEG could be structured.
The next page shows the quantization of conductance for two simple structures. The graphs show the conductance as a function of the normalized Fermi energy. The bottom one is a straight channel. The conductance curve has a stair-step shape that shows the quantization. In the top graph, the channel is constricted like the example on the page before. Scattering off the sharp edges of the constriction gives rise to the imperfect stair-steps. They show organ-pipe type resonance structure from the constriction.

On the next page are the graphs that show verification of the computer program. They are from two different sources that used different computer programs, and our graphs (shown) match the published graphs almost perfectly. Slight differences are most likely due to resolution. The top graph is for the structure of the electron stub tuner by F. Sols et al, and the bottom one is for a structure with a narrow slit by Y.S. Joe, et al.

The next page shows the structure with the reflector and stub. The reflector is at an angle of 45 degrees. It is good to note here that the reflector is not smooth, but is actually eleven steps, which had to be manually done in the input file. A suggestion for change in the program would be to include a routine that would automatically create these steps, making it possible to have hundreds, and thereby a smoother slope for the reflector. At the top are a schematic diagram and a graph of the conductance versus normalized Fermi energy for a structure with a stub that has an x-dimension of twenty nanometers. It is easy to see that there seems to be very little effect of the stub. The graph looks much like one for a straight wire. We figure here that the conductance affects displayed are due mainly to the reflector. It looks like the electrons are unable to set up standing waves in the stub, so they are unaffected by it. For the graph on the bottom, we doubled the x-dimension of the stub, and found a greater affect on the
conductance. We saw from this that the x-dimension of the stub is an important part of the structure. This has to do with the electron wavelength, which can be calculated from the Fermi energy. We found that for a normalized Fermi energy of ten in the wire, the electron wavelength is 4nm. The first standing wave requires only half the wavelength, so the x-dimension of the stub should be 2nm.

When we ran the program for the structure with the x-dimension of the stub nearing and at 2nm, we found something interesting. The next page in the presentation shows the results. The standing waves cause resonance structures in the conductance. As the value of the stub x-dimension approaches 2nm, the oscillations approach ten for the normalized Fermi Energy. You can see that for narrower stubs, the oscillations start at a higher energy, (the energy where the stub x-dimension coincides with half the electron wavelength.)

In conclusion, we learned that our program works, that is, it agrees with other similar programs. In our study of the effects of a reflector in a nanostructure, we learned that we must first fully understand the effects of the stub. "Future work" after this project was to change the computer program to allow for computation of conductance vs. geometry. Work was completed on the input program, but not on the main program to allow for this. New developments in the nature of the stub and the idea of a notch, or barrier in the structure led us down a different path.

3.2 Ball State University Student Symposium 1999
This was a poster presentation over the same material as the Butler paper.
3.3 Ball State University Student Symposium 2000
This is a poster presentation over the same material as the AAPT meeting paper.

3.4 American Association of Physics Teachers Meeting 2000
Link to presentation file

Optimizing a nanoelectronic device:
the notched electronic stub tuner

This is the documentation for the presentation given in April of 2000 at the AAPT meeting in Richmond, Indiana. The work for this report was supported by an Honors Undergraduate Fellowship and by a Research Fellowship from the Center for Energy Research, Education, and Service at Ball State University.

The first page is an overview of the presentation. The first thing to do is to discuss the goal of the project. Then, background information is presented. Next is the description of the Notched Electron Stub Tuner (NEST).

At the top of the next page is the goal, which is to optimize the NEST using expected electron streamlines. On the lower part of this page is the description of the electron streamlines. They are described using Bohn's formalism of quantum mechanics in which the system is described by wave functions and particle trajectories. The electron velocity is given as a function of the probability current density, and the trajectories are a function of the electron position vector.

The next page is a figure from a reference by Wu and Sprung, in which one can see the electron flow pattern through a narrow wire with a very narrow notch.
The top picture shows a "bouncing ball" behavior for electrons at high energies, and the bottom picture shows the trajectories for electrons at lower energies. The energies we are dealing with are in the range of the lower picture, but the top one is useful for reference. It is easy to see that the darker areas of the picture are the places where we expect a high probability density of electrons and the lighter areas are those places where we don't expect to see many electrons. The idea for our structure is to take advantage of this behavior and use a notch to set up a predictable flow pattern, and to place a stub tuner at a location where we expect a high probability density of electrons.

The next page shows design examples for the structure. In the left picture, the stub is "upstream" from the wire. In the right picture, it is "downstream". Much of the data was taken for both cases, with no difference. The notch sets up the flow pattern upstream and downstream. This stub in these cases is very narrow, so the streamlines should be the same as for a notched quantum wire without a stub. The stub samples the probability current density near the hard wall of the wire. The stub-induced conductance resonances should depend on the probability current density near the wall.

The next page shows a graph of conductance vs. normalized Fermi energy for a quantum wire with a stub (no notch). The stub (a quantum well) attached to the wire introduces antiresonances as seen in this graph. These antiresonances result in conductance oscillations as the stub is elongated (increase the y-dimension). We hope to improve these effects by placing the stub in a location where there are more electrons.

The next page shows a series of offset conductance vs. normalized Fermi energy curves. We see the antiresonance feature as in the previous graph. The difference
in the structure for each curve is the distance between the stub and the notch. As
the stub is moved away from the notch, we see differences in the antiresonance
structure. There is a minimum value for the conductance in each curve near the
normalized Fermi energy of 5.5. If we plot this minimum value of conductance
as a function of the notch-stub separation (L), we get the graph on the next page.

The oscillation of this graph is consistent with the Bohm streamline pattern. A
low conductance value here means that more electrons are being trapped in the
stub, so this would correspond to an area where there is a high probability
density of electrons near the wire wall at this location. A high value on this
graph means that there are few electrons to be trapped in the stub (the
antiresonance feature is shallow). This would coincide with a lighter area on the
Wu and Sprung figure. The lowest point on this graph is the location of the
stub where we expect the highest probability density of electrons at the wall.
The maximum on this graph coincides with a minimum of electrons at the wall.
The next step is to look at the resonance features as the stub is elongated (in the
y-direction) in these two locations.

The blue curve on the next graph shows shallow oscillations for the location
where we do not expect the electrons. The green graph shows an oscillation of
nearly one conductance unit for the stub location where we expect the highest
probability density of electrons. These two curves could represent “on” and
“off” states for this device.

If we assume that the electrons are following a “bouncing ball” pattern, we
expect to find a high probability density of electron at the top hard wall at the
same L value as we found a low probability density at the bottom hard wall, and
vice versa. If we keep the same curve color for the same $L$ value and put the stub on the top of the wire instead, we get the graph on the next page.

The green line has the shallow oscillations this time, as expected. The blue line does not show as big a conductance range as the green one did in the case with the stub on the bottom. This is most likely due to stub-notch interactions because they are on the same side of the wire.

The next page is a graph that is just the green curves form the previous two pages. It shows more clearly that when there is a high probability density of electrons at the bottom wall, there is a low probability density of electrons at the top wall. This is consistent with Bohm's streamlines.

The final page is the summary of the presentation. We saw that we can use Bohm's streamlines to give an optimum placement of the stub. As the stub length changes, the conductance values by nearly one conductance unit when there is a high probability current density at the wall, and the stub is on the opposite wall as the notch. It varies by a half conductance unit if the stub is on the opposite side of the wall and there is a high probability current density at that wall. We took data for the "upstream" and the "downstream" case and found the same results.

3.5 Texas American Physical Society Meeting 2000

Link to presentation file

Conductance characteristics of a modified electronic stub tuner
This presentation is so similar to the AAPT presentation that the documentation for that is sufficient except at the very end. The only difference is in the second to last page and in the summary, where we discuss future work. It has been shown that having multiple stubs in a series will enhance the conductance effects of the electron stub tuner. We have shown that the correct placement of a notch in the wire will enhance the conductance effects in a different way. If we could place notches and stubs in series in a wire, with optimum positions for the Bohm streamline pattern, we expect to see the conductance oscillations and a broadening of those oscillations, to make them appear more like a square-wave. In order to determine optimum distances, it will be necessary to know the electron flow pattern throughout the wire, not just at the hard walls. One proposed method of determining this flow pattern is to add a very small infinite potential barrier that would act as a scatterer. Like the stub, the scatterer would have a greater affect on the conductance of the device when placed in an area of high probability density of electrons. When we compare the conductance without the scatterer to the conductance with the scatterer in many locations in the device, we will be able to determine where the electrons are, and thereby the flow pattern through the wire. The best way to do this would be to “scan” the scatterer through the structure, determining the flow pattern at every location on a grid whose size is determined by the size of the scatterer. To do this, we need to adapt the computer program we have been using to allow for the “scanning” process and for the infinite barrier.
This project started in spring of 1998. I learned the basics of electronic conduction in nanostructure devices. I applied for an Honors College Undergraduate Fellowship for fall 1998 and spring 1999, and received it. Also during this time, I learned how to use the ProcessNanoInput, ProcessParaInput, and ComputeConductance programs to generate data. The test device was a constricted channel with wide reservoirs of the 2DEG on either side.

The device that it was the original goal of the project to explore was a "mesoscopic pinball machine". It was a channel with a tab or stub (quantum well) on one side and a reflector opposite. The idea was that the reflector could be used to alter the conductance effects of the tab.

Work on the target device began in the fall of 1998. I ran data for the device at low normalized Fermi energies (0-4). Initial data was taken for the device without the tab so we could see the results of the reflector alone on the conductance through the channel. According to an optical approach, the ideal angle of the reflector from the horizontal would be 45 degrees if we want to reflect the electron waves into a tab opposite the reflector. The y-height of the channel was 40nm on the left and 20nm on the right, coming down to form the reflector.
ran data for a reflector at 45 and 90 degrees with respect to the horizontal. The graph of Conductance vs. Normalized Fermi Energy showed little effect of the reflector.

At this point, I think I should explain how the reflector could be formed within the limits of the computer program. I was running this data in August 1999 well before we started modifications on the program. There was no way to make diagonal lines for the device model within the program, so the reflector was jagged. There were eleven steps from the 40nm height to the 20nm height. Any structure in the conductance graph was easily attributable to the jagged nature of the reflector. Also, we found there was very little structure to the graph.

The next step was to add the reflector. The reflector was 20nm by 20nm and across from a 45-degree reflector. The graph for this had a great deal of structure but in no recognizable pattern. We decided to see the effects of moving the reflector to the right or left (x direction) and leaving the tab where it was. For this data, we used a reflector of 90 degrees. The results were similar. Next, we ran the same data for a plain channel with a tab (no reflector). We used 40nm and 20nm for the heights of the channels. These results were similar to the previous ones, which made us question the effects of the reflector at all.

We decided to move the reflector downstream (to the right) of the tab. We could change the properties of the reflector without changing the tab when we modified the input files in this way. The optical model requires a 60-degree angle for reflection into the tab at the downstream location. The graph still showed little effect of either the tab or the reflector.
Verification: Dr. Cosby decided it would be a good idea to verify the program again, although it had been done before. I ran the program using data for structures that were used in published papers. I ran data for a wire with slits by Y.S. Joe et al. and for a wire with a tab by F. Sols et al. It worked fine for the paper by Joe, but I had a little trouble matching the data by Sols. The problem turned out to be that I was using a different effective mass for the electron, and eventually we got it to match perfectly.

Now we have data for a wire with a reflector and a tab that was run using a verified program, but there are almost no effects of the reflector or the tab on the conductance. We tried changing the lattice parameter and found that the data converged. This means our value for the lattice parameter was not affecting the graph very much. The next thought was to widen the tab (to make it bigger in the x direction). In running the data for a tab at 20nm, 30nm, and 40nm, we saw a significant increase in the effects of the tab as the width of the tab increased. This was when we realized that the width of the tab is significant to the conductance through the wire. We did calculations to determine what the electron wavelength was at that energy. It was too long to fit in the width of the tab.

At higher energies, the wavelength of the electron is much shorter, so the next step in the project was to take all we learned up to higher energies. We started running data for devices at high energies (9-11) in October 1998. The first was a plain wire 20nm high. We got the characteristic graph for the quantized conductance. We calculated the electron wavelength at the tenth energy plateau and found that it was 4nm. We ran the program for a wire 20nm high with a tab 20nm deep and 4nm wide. There were great but unpredictable effects on the conductance. We decided to try the organ-pipe resonance model, where it is the
half-wavelength that is the critical length. We ran the program again for the same structure but shortened the width of the tab to 2nm. This was on October 14. The graph for this showed orderly resonance that began on the tenth plateau.

Next, we experimented with the width of the tab. We made it smaller by increments and it is clear that the resonance appears as soon as the half wavelength will fit into the tab. Although the results for this were fascinating, they were hard to come by. The process involved rerunning the program for slightly varying data a number of times. We were starting to develop real need for the program modification to allow us to run the program to find the conductance vs. the structure geometry, rather than the Fermi energy.

In the meantime, I made analyses if the minima for the graphs with the Fano-type resonance, comparing the energy level at the minima with the wavelength and the size of the tab. This analysis took up more than half of November, with no real hypotheses resulting from it.

In December, I started work modifying the computer programs. The first to fix was the ProcessParaInput program. This program is the data-file generation program for the parameters of the structure, but not the device. The input for the ProcessParaInput Program includes the Fermi energy levels, the lattice parameter, and the effective mass of the electron, along with other things. It was in this program that it seemed most logical to allow for the choice between conductance vs. Fermi energy and conductance vs. geometry. Modification of this program was left mostly to me, although the direction was from Dr. Cosby. I had to learn a great deal of Fortran90, of which I am still in the process. Dr. Cosby began work on modifying the Compute Conductance program while I finished the ParaInput program. The ProcessParaInput modifications were finished by March.
1 1999. The ProcessParaInput program now allows the user to decide between Fermi energy and geometry, allows the user to specify a particular Fermi energy to analyze, and allows for an incrementing of the y-dimension of the structure. The ProcessNanoInput program remains the same. The ComputeConductance revisions have not been completed at the time I write this (April 20, 1999).

On March 23, 1999 I presented a poster at the Ball State University Student Symposium entitled “Electronic Conduction in a Mesoscopic Device”. There, I had an opportunity there to talk with people who had no training in condensed matter research about the project. You have to know what you are talking about to make other people understand what you do, and that what you do is interesting and worthwhile.

On April 9, 1999, I gave an oral presentation at the Butler Undergraduate Research Conference entitled “High Energy Electron Transport in Semiconductor Nanostructures”, that was similar to the poster, but for obvious reasons, more in-depth. This was my first experience with presenting research findings in front of other Physics researchers. It went well.

After the Butler conference, I went back to work in the computing lab working on the programs. While Dr. Cosby continued to work on the ComputeConductance program, I began looking at a graphing program that we may be able to add to the ComCon program. It will graph the results, instead of just giving data points that have to be imported to Quattro Pro. The graphing program is Scigraph and is included with our version of Fortran PowerStation, although there is some assembly required. Along with the other modifications to the ComCon program, the output has to be written into a two by two array to allow the Scigraph file to read it.
On April 26, 1999, the last week of the semester before finals, I successfully added a Scigraph Subroutine to the compute conductance program. It has begun as an unnecessary appendage. It takes no data from the program; it just presents a blank graph when the program runs. We still have to rewrite the output of the program into arrays of acceptable size for Scigraph data, and get those into the subroutine. I hope this can be finished by the end of this week.

It seems at this point that I will have to leave for the summer while on the verge of finishing the Scigraph Subroutine in the experimental ComputeConductance program. With my limited knowledge of the programming language Fortran90, I am unable to reshape the output arrays from the WriteTo subroutine to fit the data array for Scigraph.

I will return to work on this project in August, perhaps on this very problem.

At the start of a new semester, I am trying to figure out where I left off. In trying to incorporate the Scigraph subroutine, I have put more bugs in the Compute Conductance program. I am beginning to think we were better off importing the data into Quattro Pro.

I finally got the programs to work the way they used to. I learned a few things. First, that “build” and “rebuild” are entirely different things. Once all of the files are in the projects, if you make any changes you have to “rebuild” the project. In addition, we did some modifying to the Compute Conductance program, so the revised version must be used with the revised ProcessParaInput. I am back to trying to see if Scigraph is really worth the trouble. The next step for the project is incorporating a new graduate student as Hsiu-Lien Hu is leaving soon. I am going to teach Kevin the ins and outs of the Fortran PowerStation programs, how to set up a project workspace and get it running.
On September 22, 1999, I began work on a new structure. Dr. Cosby and a graduate student had been working on a structure with a potential barrier notch as a wave-guide and a small tab as a probe for the conductance. I first started to work on a structure with two notches and a tab between them. What we wanted to look at was the trend of the conductance as the tab moved across from right next to one tab to right next to the other, for different values of $D$, the separation on the notches. This presented a problem with programs working the way they did. The compute conductance program calculated the conductance for one position of the tab at a time. We instituted a "Do Loop" in the program to allow it run several times with different values of the x-dimensions of the structure. This calculated the conductance for 551 points in and energy range per structure, for 22 variations on structure.

We ran into a few obstacles with the computers. The 450 MHz computer requires seven minutes to run through the program for one structure, about three hours to run the whole thing. I then went through the data, transferring the files to Quattro Pro and graphing the conductance curves. With program modification and heaps of data to sort through, this work took me to November 1.

On November 1, 1999, I began working on a one-notch structure. Hsiu Lien Hu, a graduate student, had been working on a structure with a notch followed by a tab. It was my task to find the properties of a structure where the tab came first, was upstream from the notch. Hu had found an interesting pattern to the conductance minima as the notch and the tab were separated. She was seeing something like a sine wave that was dissipating as the distance between the two features increased. Dr. Cosby wanted to investigate the validity of using Bohm streamlines to describe the electron flow pattern, using this data.
With the modified program we already had developed, it was not difficult to get the data. Again, the program was running on a fast computer for several hours at a time. We wanted data for intervals of the tab-notch separation distance \( L \) of one Angstrom, starting with the notch and the tab right next to each other and moving the tab to 100 Angstroms away from the notch. We narrowed the range of energy, but not the number of data points, in order to get an accurate value for the conductance minima. We were looking at normalized Fermi energy between 4.25 and 4.75. We realized that the conductance minimum we were following moves across this range and outside of this range as \( L \) increases. For this reason, we have not been using the data beyond 80 angstroms for \( L \) until we expand the energy range.

On the graph of these conductance minima, there is a minimum at \( L=45 \)A and a maximum at \( L=69 \)A. The value halfway between these is \( L=54 \)A. The next step was to investigate the conductance properties when the energy and \( L \) are held constant and the \( Y \) dimension of the tab, \( S \), is changed. This required another program modification. I began working on that on November 16.

The work was done for a structure with the tab downstream of the notch. There were several ways to look at this data, after running the program for each of the different \( L \) values at each of the different energies. For this program, the energy is constant, and the \( S \) value changes. The difficulty is that the conductance minima for the three values of \( L \) I was using occur at slightly different energies. That is why I ran the data for each of the energies. It turns out it does not make much of a difference in the features on the graph, the difference in conductance between \( L=69 \) and \( L=45 \) was drastic at all of the energy level. To support Dr. Cosby's
hypothesis about Bohm streamlines, we only had to show that there is a difference.

On November 29, 1999, I wrote out a guide to the variations on the L-varying program. There are four variations: one that works when the notch and tab are overlapping, and one that works when they are not. Two of those programs work when the tab is downstream, and two work when the tab is upstream.

In December, Dr. Cosby started preparing for a January seminar. I helped get the data together and into a presentable form. I got Hsiu Lien's data for the conductance minima curve when the tab is upstream, and combined it with mine to get the whole upstream picture. I also worked during this time on the case when the tab and the notch are overlapping. This might be another case where our energy range is too small to get the particular conductance minimum that we want to investigate.

In late December and early January, I was working with a new (to me) graphing program, Psi Plot. This program makes it possible to generate publication-quality graphs. I was able to generate graphs of the conductance minima, the conductance vs. normalized Fermi energy for several values of L, and the conductance vs. S for specific values of L. On each of these, I was able to draw a schematic diagram of the structure and the parameters that we varied.

I was happy to find on my return this semester that the programs on the old Fortran PowerStation are still working. None of my program modifications were lost to the Y2K bug.
Each of the investigations we make into these structures seems to lead to at least ten more. Now that we have the programs for L variation and Y variation, there are so many more things to investigate in each structure.

At the end of January, Dr. Cosby and I sat down and made a plan for what we want to look into this semester. The first thing is the structure as before but with the notch and the tab on the top. I ran the programs, acquired data for conductance vs. S (tab length) as the tab is moved along the top and the bottom of the structure, and compared the two. As expected, when there was a deep resonance with the tab on the bottom, there was a shallow resonance with the tab on the top for the same value of L. As L was varied from 45 to 69 with the tab on top, the resonances went from shallow to deep. As L was varied from 45 to 69 with the tab on the bottom, the resonances went from deep to shallow. The effect is considerably more pronounced with the tab on the bottom.

I had some of the answers to the questions we were posing with our plan by mid-February. For the structure with the tab and the notch on the same side, I ran the program and found the Gmin vs. L plot. It was not very interesting. It was considerably less pronounced than the Gmin vs. L plot for the tab on the bottom. Dr. Cosby thought that this might be from standing waves that could be set up in the space between the edge of the tab and the edge of the notch when they are on the same side. This effect would not occur with the notch on the bottom. I also ran the program in which we vary the depth of the tab for a certain energy and L value. This showed what we were expecting to happen. The behavior of the tab on the top was opposite that of the tab on the bottom. The magnitude was, of course, different. Again, this is probably due to those standing waves.
In late February and early March I worked on the question of whether $L_n=20$ is really the best value for the notch width. It turns out it is. The first thing I did was to run a program adapted from the L-vary program that allowed me to increment the tab width from 10nm to 40nm. This structure had no tab, so we could see what the effects were on the resonances due to just the notch width. There did not seem to be much effect. Therefore, I ran the long L-vary program for $L_n=30$ and $L_n=10$. I generated a $G_{\text{min}}$ vs. L plot for 30 that looked very much like the one we got for 20. I then went back to the data to make sure I was following the same resonance all the way. It turns out I was not. The resonance I thought I was following moves out of the energy range at $L=67$, which was a turning point on the $G_{\text{min}}$ vs. L plot. I ran the program again for the higher values of L at a lower energy and picked up the resonance. When this data was added to the $G_{\text{min}}$ vs. L plot, I saw that the pattern fizzes out after 67. There really is not much there that we could use. This led me to wonder if 10 was even better than 20, since we manage to maintain the pattern in the $G_{\text{min}}$ vs. L plot for 20 using the one resonance. I know that the decision to use $L_n=20$ was made after some calculations. When I ran the L-vary program with $L_n=10$ and generated the $G_{\text{min}}$ vs. L plot, I saw the same features as the $L_n=30$ plot. The resonance leaves the picture at about $L=67$. After all this, we see that we were right to begin with, which is always nice to know.

On March 15, 2000, I am working on upcoming presentations. I will present again this year at the Ball State Student Symposium. The presentation is a poster on the NEST: Notched Electronic Stub Tuner. Then in April, I will present a paper on a similar topic at the AAPT meeting.

The information presented at the Ball State Student Symposium on March 28 and at the AAPT meeting on April 15 included a reference to a paper by Wu and
Sprung. This is a graph that shows the electron streamlines through a wire with a very narrow potential barrier (notch). I explained how this barrier affected the conductance through the wire. Without the notch, the electron streamlines would be parallel, and there would be no differences in the probability current density along the wall. The notch sets up the pattern, which we used to enhance the effects of the stub on the wire. I showed information on the effects of the stub without the notch, which was a conductance graph showing the resonance that causes oscillations in the conductance as the stub is elongated. I had a graph of these conductance curves for the NEST, showing how the resonance chances in depth as the notch-stub separation distance changes. I showed the plot of the conductance minima as a function of the notch-stub separation. I indicated that where there was a deeper resonance, this was where there was a greater effect of the stub on the structure, which coincides with a greater probability current density near the wall. Where the minima were shallow, this coincides with a low probability current density near the wall. These graphs support the electron streamline picture we were using. The presentations also included plots of the conductance oscillations caused but the elongation of the stub. If the stub is positioned where there is a high probability current density near the wall, the oscillations have amplitude of nearly a conductance unit. If the stub is positioned at low probability current density, the oscillations have very small amplitude. We also had the plot for the case where the notch and the stub are on the same side of the wire, and a plot that showed the possible "on" "off" states for the stub at the same L-value on the bottom and the top of the wire. I thought the presentations went rather well. I have a good understanding of the information.

I do not know if I will continue work on this structure next year or not. Upcoming events include a paper by Dr. Cosby on the NanoSpace 2000
information (I have already made the black and white plots for publication), and the end of the semester.

Before we got busy presenting, I had been working on the stub directly under the notch structure. I ran into a problem in the computer program. The program automatically calculates the Ef normalization factor for the narrowest space in the wire. For all of our other calculations, this was always the same because it was the space under the notch. When the stub is under the notch, this is the smallest space until S gets to be greater than the width of the wire. In addition, it changes every time the program goes through the loop, because that dimension changes. We have not decided yet the future of these particular calculations.

At the start of the Fall 2000 semester, we upgraded to a new developer studio for the programs. I moved all of the programs to the new Digital Visual Fortran and got them working the way they were in the old environment. It is pretty much the same program, but with a few new helpful features and a few charming idiosyncrasies. The first problem I ran into was with the stack size. The default value for temporary memory the computer allocates for the program was insufficient in the new program, and that has to be changed every time a new workspace is created.

On August 25, 2000, I ran the program for a few simple verification structures and all went well. On August 30, Dr. Cosby and I discussed a new idea for the structure. In the past we had been probing the electron density at the wall with a stub (tab, well). The new ideas was to probe the electron density everywhere in the two dimensional structure using a small infinite barrier, or scatterer. We can calculate the conductance for the structure without the scatterer and with the scatter and plot the difference at each point using Array Viewer. In order to do
this, we needed to modify the Compute Conductance program to allow for the infinite scatter. This turned out to be a huge job.

On September 13, I wrote the code that allowed us to put all of the needed new subroutines into the structure of the existing program. In the last week of September I learned how to use the Array Viewer program. For the rest of the semester I was up to my elbows in quantum theory, trying to find all of the necessary parameters and the wave equations for the infinite barrier. Also this semester, I presented at the Texas APS meeting in Houston and got to see some of the leading Hispanic Physicists speak. Knowing that I would be doing program modification next semester, I spent finals week and part of my Christmas break going through the Compute Conductance program and finding out what happens in every step of the program.

In the Spring 2001 semester, I am working the ten hours a week for the fellowship, and I am taking an independent study with Dr. Cosby and working on my thesis. That means that I am in the lab in the basement of Cooper from ten or eleven until at least five every day. This semester we want to get the program together to run the scatterer data. This means that three subroutines have to be changed and four had to be written and added to the program.

On February 5, 2001, I finished the changes to the transverse energy part of the program. I changed the existing ComputeTransverseEnergy subroutine to allow for the case when the slice has an infinite barrier, and then I added the subroutine that calculates the transverse energy in this case, which needed a separate subroutine that sorts the energies.
On February 7, I ran test data and was happy to see that the subroutines were doing what I wanted. I then proceeded to change the SortEnergies every two days or so until I had it just right. As I went on making changes to other parts of the program, I realized that there was more information that I needed that I could get from SortEnergies. On February 9th, I finished the modifications to the WaveAmplitude part of the program. When I tested this, it all looked right. Then came the hard part. I started working on the part of the program that calculates the hopping matrix.

On Feb 15th-19th I worked on the hopping matrix subroutines. As far as I can tell, the program is doing what I told it to do. The code is all right, but the program doesn't work right. In the last part of February, I tried to verify the program, using published data from Dr. Joe, but it doesn't work. Since the problem is most likely in the physics or the math, Dr. Cosby is working on it as well. Spring break was the first full week in March and we are still looking for the problem. I have been working on getting things organized for my thesis project and catching up on this report.

On a miraculous day in the second week of April, after re-writing the hopping case for when the right slice has an impurity and the left slice does not, the program finally worked right. For the next several days, I ran data to verify our results with previously published results, and all seemed well. Then I started working on automating the program. As it was, when you wanted information about how the conductance changes at a set energy as a function of some physical parameter of the device, all of that information had to be hard-coded into the program. Now, the information can be entered into the ParaData.Input program, and ComputeConductance can be left alone. There are still some major flaws in this version. It will change a specific value of the device every time it runs though, but in only one slice of the structure. And, it has no idea what the
original value or the final value are, so it is up to the user to remember that. This is remarkably difficult for a user (specifically me) to do. The ultimate version of the program will run an impurity through a scan of the whole device—all slices—without stopping. This gets into whole new dimensions of difficulty. I have to finish my thesis project as soon as possible so that I can spend the final two weeks of my undergraduate career preparing the way for the next person who has this job. Good luck. —T.V.H.


Plain Channel

This structure is a simple quantum wire. The conductance graph shows the characteristic stair-step pattern of quantization.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\PC2heND</td>
<td>Data\GEf\pc2heGEf</td>
<td>Data\Graphs\Pc2heplot</td>
</tr>
</tbody>
</table>

Wire with stub on top

This structure was used in early verification of the Compute Conductance program. It is a simple wire with a potential well on the top hard wall.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\SL2ND</td>
<td>Data\GEf\SL2GEf</td>
<td>Data\Graphs\SL2plot</td>
</tr>
<tr>
<td>Sols2ND</td>
<td>Sols2GEf</td>
<td>Sols2plot</td>
</tr>
<tr>
<td>Sols3aND</td>
<td>Sols3aGEf</td>
<td>Sols3apl</td>
</tr>
<tr>
<td>Sols3bND</td>
<td>Sols3bGEf, Sols3b2GEf</td>
<td>Sols3bplot</td>
</tr>
<tr>
<td>Sols3ND</td>
<td>Sols3GEf</td>
<td>Sols3plo</td>
</tr>
<tr>
<td>SolsND</td>
<td>SolsGEf</td>
<td>Solsplot</td>
</tr>
</tbody>
</table>
Wire with reflector

This was a preliminary study for the original "quantum pinball" device. We looked at the conductance features due to the reflector at different angles, before adding the potential well. The numbers at the beginning of the filenames are the Angles (A).

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\15degND</td>
<td>Data\GEf\15degGEf</td>
<td>Not graphed</td>
</tr>
<tr>
<td>45degND</td>
<td>45degGEf</td>
<td>Data\Graphs\45degplo</td>
</tr>
<tr>
<td>90degND</td>
<td>90degGEf</td>
<td>90degplo</td>
</tr>
</tbody>
</table>

Wire with reflector and well, or "Quantum Pinball Machine"

This is the first device for this project. The idea was to use the reflector to increase the quantity of electrons getting trapped in the well, thereby enhancing the effects of the well. The first number in the filenames is the angle (A) in degrees. The "lp" in some of them is from an investigation in altering the Lattice Parameter. "tw" and "tab" are followed by the width of the tab (Ls) in nanometers.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\45tabND</td>
<td>45tabGEf</td>
<td>Data\Graphs\45tabplo</td>
</tr>
<tr>
<td>60tabND</td>
<td>60tabGEf</td>
<td>60tabplo</td>
</tr>
<tr>
<td>90tabND</td>
<td>90tabGEf</td>
<td>90tabplo</td>
</tr>
<tr>
<td>90tab2ND</td>
<td>90tab2GEf</td>
<td>90t2plot</td>
</tr>
<tr>
<td>90tab3ND</td>
<td>90tab3GEf</td>
<td>90t3plot</td>
</tr>
<tr>
<td>60t1ND</td>
<td>60t1GEf</td>
<td>60t1plot</td>
</tr>
<tr>
<td>60t3ND</td>
<td>60t3GEf</td>
<td>60t3plot</td>
</tr>
<tr>
<td>60lp1ND</td>
<td>60lp1GEf</td>
<td>60lp1plo</td>
</tr>
<tr>
<td>60lp2ND</td>
<td>60lp2GEf</td>
<td>60lp2plo</td>
</tr>
<tr>
<td>60lp'5ND</td>
<td>60lp'5GEf</td>
<td>60lp'5pl</td>
</tr>
<tr>
<td>60lp'1ND</td>
<td>60lp'1GEf</td>
<td>60lp'1pl</td>
</tr>
<tr>
<td>60tw3ND</td>
<td>60tw3GEf</td>
<td>60tw3plo</td>
</tr>
<tr>
<td>60tw4ND</td>
<td>60tw4GEf</td>
<td>60tw4plo</td>
</tr>
<tr>
<td>45tw4ND</td>
<td>45tw4GEf</td>
<td>45tw4plo</td>
</tr>
</tbody>
</table>
Wire with Stub (also called potential well, or tab)

This is very similar to the Sols structure we used for verification in the first stages, except this time the well is on the bottom hard wall. The conductance graphs look the same. The names here refer to the dimensions. "p2" means the channel is 200A wide, "t4" means the stub (tab, well) is 40A wide (Ls).

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\p2t4heND</td>
<td>Data\GEf\p2t4heGEf</td>
<td>Data\Graphs\P2t4hepl</td>
</tr>
<tr>
<td>P2t4heND</td>
<td>p2t4n5GEf</td>
<td>P2t4n5pl</td>
</tr>
<tr>
<td></td>
<td>p2t4n6GEf</td>
<td>P2t4n6pl</td>
</tr>
<tr>
<td>P2t1heND</td>
<td>p2t1n5GEf</td>
<td>P2t1n5pl</td>
</tr>
<tr>
<td>P2t2heND</td>
<td>p2t2e0-11GEf</td>
<td>P2t2p11p</td>
</tr>
<tr>
<td></td>
<td>p2t2n6GEf</td>
<td>P2t2n6pl</td>
</tr>
<tr>
<td></td>
<td>t2e9-11dpGEf</td>
<td>T2e9-11p</td>
</tr>
<tr>
<td></td>
<td>t2n6e4-6Gef</td>
<td>T2n6e4-6</td>
</tr>
<tr>
<td>P4t4heND</td>
<td>p4t4heGEf</td>
<td></td>
</tr>
<tr>
<td>P2t2&quot;heND</td>
<td>p2&quot;e4-5GEf</td>
<td></td>
</tr>
<tr>
<td>P2t2&quot;heND</td>
<td>p2&quot;e4-6GEf</td>
<td>P2&quot;e4-6p</td>
</tr>
<tr>
<td>P2t125heND</td>
<td>p2t125GEf</td>
<td>P2t125</td>
</tr>
<tr>
<td>P2t150heND</td>
<td>p2t150GEf</td>
<td>P2t150</td>
</tr>
<tr>
<td>P2t175heND</td>
<td>p2t175GEf</td>
<td>P2t175</td>
</tr>
<tr>
<td>P2t180heND</td>
<td>p2t180GEf</td>
<td>P2t180</td>
</tr>
<tr>
<td>P2t185heND</td>
<td>p2t185GEf</td>
<td>P2t185</td>
</tr>
<tr>
<td>P2t190heND</td>
<td>p2t190GEf</td>
<td>P2t190</td>
</tr>
<tr>
<td>P2t195heND</td>
<td>p2t195GEf</td>
<td>P2t195</td>
</tr>
<tr>
<td>P2t225heND</td>
<td>p2t225GEf</td>
<td>P2t225</td>
</tr>
<tr>
<td>P2t250heND</td>
<td>p2t250GEf</td>
<td>P2t250</td>
</tr>
<tr>
<td>P2t275heND</td>
<td>p2t275GEf</td>
<td>P2t275</td>
</tr>
<tr>
<td>P2t300heND</td>
<td>p2t300GEf</td>
<td>P2t300</td>
</tr>
<tr>
<td>PCtabND</td>
<td>PCtabGEf</td>
<td>Pctabplo</td>
</tr>
<tr>
<td>PCtab2ND</td>
<td>PCtab2GEf</td>
<td>Pctab2plot</td>
</tr>
<tr>
<td>Data\Tab</td>
<td>Data\Tab</td>
<td></td>
</tr>
<tr>
<td>Y\NanoData\SonlyVar</td>
<td>Y\GEf\G_SonlyVar</td>
<td>Data\Tab</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y\Graphs\SonlyVar</td>
</tr>
</tbody>
</table>
The Notched Electron Stub Tuner (NEST)

The NEST is the most extensively studied structure, and was the reason for several revisions of the CC program. In these files, either L or S was varied with iterations of the program.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEfs)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\L\NanoData\N_N10_L1_T</td>
<td>Data\L\GEfs\G_N10_L1_20_T</td>
<td>N10_lv_t</td>
</tr>
<tr>
<td>N_N10_L20_T</td>
<td>G_N10_L20_41</td>
<td></td>
</tr>
<tr>
<td>N_N10_L40_T</td>
<td>G_N10_L40_61</td>
<td></td>
</tr>
<tr>
<td>N_N10_L60_T</td>
<td>G_N10_L60_81</td>
<td></td>
</tr>
<tr>
<td>N_N20_L50_T</td>
<td>G_N20_L50_70_T</td>
<td>N20_L50</td>
</tr>
<tr>
<td>N_N20_L71_T</td>
<td>G_N20_L71_81</td>
<td></td>
</tr>
<tr>
<td>N_N30_L1_t</td>
<td>G_N30_E49_53</td>
<td>N30_lowe</td>
</tr>
<tr>
<td>N_N30_L20_T</td>
<td>G_N30_L20_41_T</td>
<td>N30_lv_t</td>
</tr>
<tr>
<td>N_N30_L40_T</td>
<td>G_N30_L40_61_T</td>
<td></td>
</tr>
<tr>
<td>N_N30_L60_T</td>
<td>G_N30_L60_81_T</td>
<td></td>
</tr>
<tr>
<td>N_nt_L1</td>
<td></td>
<td>Not graphed</td>
</tr>
<tr>
<td>N_wide10_T</td>
<td>G_N_wide10_T</td>
<td>N_wide_t</td>
</tr>
<tr>
<td>N_wide30_T</td>
<td>G_N_wide30_T</td>
<td></td>
</tr>
<tr>
<td>Data\Tab Y\NanoData\N_L45_nt</td>
<td>Data\Tab Y\GEfs\G_L45,E45 and 69 files\ G_L45_E45,G_L45_E54,G_L45_E69</td>
<td>Data\Tab Y\Graphs\L45_tabv</td>
</tr>
<tr>
<td>N_L54_nt</td>
<td>G_L54_E45, G_L54_E54</td>
<td>L54_tabv</td>
</tr>
<tr>
<td>N_L69_nt</td>
<td>G_L69_E45, G_L69_E54</td>
<td>L69_tabv</td>
</tr>
<tr>
<td>Data\Tab Y\NanoData\notch tab bottom files\ntb_L45</td>
<td>Data\Tab Y\GEfs\ntb files\ntb_L45_E45</td>
<td>L45_tabv, E45_tabv</td>
</tr>
<tr>
<td>ntb_L475</td>
<td>ntb_L475_E45</td>
<td>E45_tabv, Ntb_ls</td>
</tr>
<tr>
<td>ntb_L50</td>
<td>ntb_L50_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L525</td>
<td>ntb_L525_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L55</td>
<td>ntb_L55_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L575</td>
<td>ntb_L575_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L60</td>
<td>ntb_L60_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L625</td>
<td>ntb_L625_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L65</td>
<td>ntb_L65_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L675</td>
<td>ntb_L675_E45</td>
<td></td>
</tr>
<tr>
<td>ntb_L69</td>
<td>ntb_L69_E45</td>
<td></td>
</tr>
<tr>
<td>Data\NanoData\N_nt_L1</td>
<td>Data\GEfs\G_nt_L1_22</td>
<td>Not Graphed</td>
</tr>
<tr>
<td>N_nt_L90</td>
<td>G_nt_L90-98</td>
<td></td>
</tr>
</tbody>
</table>
NEST (stub “upstream” from notch)

This was the first NEST device studied. The notch is used to set up a predictable electron flow pattern through the wire, the tab samples the probability current density at the bottom hard wall.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\Tab Y\NanoData\N_L45_tn</td>
<td>Data\Tab Y\GEf\G_L45,54, and 69 files\G_L45_E45_tn</td>
<td>Not Graphed</td>
</tr>
<tr>
<td>N_L69_tn</td>
<td>G_L69_E45_tn</td>
<td></td>
</tr>
<tr>
<td>N_tn_Lit</td>
<td>G_e4_Lit22</td>
<td>Data\Graphs\Q_tn_e5</td>
</tr>
<tr>
<td></td>
<td>G_e5_Lit22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G_tn_L1_22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G_tn_L20_42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G_tn_L40_62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G_tn_L60_82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G_tn_L80_102</td>
<td></td>
</tr>
</tbody>
</table>
NEST (Stub “downstream”, top)

In this NEST device, the stub samples the current density along the top hard wall. The graphs show that there is a low current density at the top wall for values of L that correspond to high current density at the bottom wall, as expected.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT (GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\L\NanoData\N_nnt_L1</td>
<td>Data\L\GEf\G_nnt_L1_22</td>
<td>Not graphed</td>
</tr>
<tr>
<td>N_nnt_L20</td>
<td>G_nnt_L20_42</td>
<td></td>
</tr>
<tr>
<td>N_nnt_L40</td>
<td>G_nnt_L40_62</td>
<td></td>
</tr>
<tr>
<td>N_nnt_L60</td>
<td>G_nnt_L60_82</td>
<td></td>
</tr>
<tr>
<td>Data\Tab Y\NanoData\notch tab top files\nttop_L45</td>
<td>Data\Tab Y\GEf\nttop files\nttop_L45_E45</td>
<td>Data\Tab Y\Graphs\Ntt_Ls</td>
</tr>
<tr>
<td>ntttop_L47.5</td>
<td>ntttop_L47.5_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L50</td>
<td>ntttop_L50_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L52.5</td>
<td>ntttop_L52.5_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L55</td>
<td>ntttop_L55_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L57.5</td>
<td>ntttop_L57.5_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L60</td>
<td>ntttop_L60_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L62.5</td>
<td>ntttop_L62.5_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L65</td>
<td>ntttop_L65_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L67.5</td>
<td>ntttop_L67.5_E45</td>
<td></td>
</tr>
<tr>
<td>ntttop_L69</td>
<td>ntttop_L69_E45</td>
<td></td>
</tr>
</tbody>
</table>
NEST (Stub "upstream", top)

This data was taken to verify that the stub behaves the same way in the "upstream" case as it did in the "downstream" case.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\Tab Y\NanoData\N_L45_ttn</td>
<td>Data\Tab Y\GEf\G_L45,54, and 69</td>
<td>Not graphed</td>
</tr>
</tbody>
</table>

Notched Wire

This structure was used as a comparison with the NEST to see which conductance features were due to the stub, and which were due to the notch.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\L\NanoData\N_only</td>
<td>Data\L\GEf\G_Nonly_vary</td>
<td>N_wide</td>
</tr>
</tbody>
</table>
Double-Notched Wire

This structure was investigated before the NEST, and is an object of future work. If a predictable electron flow pattern can be determined using the NEST, then multiple notches (and multiple stubs) can be used to enhance desired behavior.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\notchwireN</td>
<td>Data\GEf\Gefnotchwire</td>
<td>Data\Graphs\60ntchwr</td>
</tr>
<tr>
<td>14notchwireN</td>
<td>14Gefnotchwire</td>
<td>14notwir</td>
</tr>
</tbody>
</table>

Double-Notched wire with stub (DNEST)

In this data, effects of the widths of the notches and their separation distances are studied.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\N_nwt_L2</td>
<td>Data\GEf\G_nwt_e4_L2</td>
<td>Data\Graphs\Q_nwt_l2</td>
</tr>
<tr>
<td>N_nwt_L5</td>
<td>G_nwt_L5</td>
<td>Q_e4_L5</td>
</tr>
<tr>
<td>N_nwt_Lit</td>
<td>G_nwt_Lit_e4</td>
<td>Q_e4_lit, Aq_e4_li, Bq_e4_li, Cq_e4_li, Dq_e4_li, Eq_e4_li, Fq_e4_li</td>
</tr>
</tbody>
</table>
Verification Structure

This is a structure that Dr. Y.S. Joe and the condensed matter research group at Ball State University had previously studied. The data taken for a published paper on the structure was taken using a different program. That information was used in early verification of the ComputeConductance program.

<table>
<thead>
<tr>
<th>NANO DATA</th>
<th>OUTPUT(GEf)</th>
<th>GRAPH FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data\NanoData\091498ND</td>
<td>Data\GEf\091498GEf</td>
<td>Data\Graphs\091498pl</td>
</tr>
</tbody>
</table>