Untersuchung dünner Si_3N_4 - Schichten mit Ellipsometrie und RBS

von

Tanja Dessauvagie

Diplomarbeit in Physik
angefertigt im Institut für Strahlen- und Kernphysik

vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

im September 1997

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent : Priv. Doz. Dr. R. Vianden

Koreferent: Prof. Dr. K. Maier

Inhaltsverzeichnis

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent : Priv. Doz. Dr. R. Vianden

Koreferent: Prof. Dr. K. Maier

1 Einleitung

Das Studium von Siliziumnitrid-Schichten hat in den letzten Jahren immer größere Bedeutung erlangt. Siliziumnitrid besitzt Eigenschaften, die für Anwendungen in der Halbleiterindustrie sehr wertvoll sind, z.B. hoher elektrischer Widerstand, große Härte, Oxidationsstabilität auch bei hohen Temperaturen und gute Abnutzungsresistenz. Dies führt zu einer Vielzahl von verschiedenen Anwendungen, wie z.B. die Passivierung von Siliziumoberflächen oder die Verwendung als Oxidationsmaske für planare Strukturen.
In dieser Arbeitsgruppe werden mit Hilfe der gestörten $\gamma\gamma$-Winkelkorrelation (PAC = engl.: Perturbed Angular Correlation) unter anderem Defekte und Defektkomplexe in ionenimplantierten Galliumarsenid untersucht. Bei der Ionenimplantation mit stabilen und radioaktiven Isotopen werden die oberen Kristallschichten durch den Einschuß der Sondenatome zerstört. Diese Proben werden deswegen mit einer sogenannten Rapid-Thermal-Annealing-Anlage (RTA) ausgeheilt. Dabei werden die Proben solange erhitzt, bis der Kristall aus den intakten tieferen Schichten wieder nachgewachsen ist. Bei den III-V-Halbleitern hat man jedoch das Problem, daß die Elemente der V. Hauptgruppe aufgrund ihres hohen Dampfdrucks bei der Ausheiltemperatur von 800-1000°C aus der Oberfläche der Probe ausdampfen [Mar90]. Eine der Standardmethoden um das zu verhindern ist es, die Proben mit einer dünnen, amorphen Siliziumnitridschicht zu überziehen, die dann als sogenanntes „Cap“ fungiert und das Ausdampfen verhindert. Die Zusammensetzung der Schicht und die Schichtdicke sind ausschlaggebend dafür, ob das Cap seine Aufgabe erfüllt oder nicht. Ist die Schicht zu dünn, können Löcher in der Schicht entstehen, die das Ausdampfen der Elemente der V. Hauptgruppe erlauben. Bei zu dicken Schichten ist die Gefahr vorhanden, daß thermische Spannungen zwischen Schicht und Substrat auftreten und das Cap aufbricht. Ist die stöchiometrische Zusammensetzung nicht gegeben, kann es sein, daß die Schicht nicht mehr für andere Komponenten dicht ist. Wenn die Schicht z.B. zu viel Sauerstoff enthält, dann dampft Gallium aus.
Um dünne Schichten oder Filme zu untersuchen, stehen eine Vielzahl von Methoden zur Verfügung. Zwei dieser Methoden wurden in dieser Arbeit verwendet, um die Qualität der hergestellten Siliziumnitridschichten zu untersuchen. Die erste Methode, die Ellipsometrie, konnte mit dem von M.Mendel [MEN96] gebauten Ellipsometer im Haus durchgeführt werden. Sie erlaubt eine genaue Untersuchung der Schichtdicke und des Brechungsindex der Schicht. Bisher war es nur möglich anhand einer Farbskala die Schichtdicke auf ca. 100Å genau einzuschätzen. Das Ellipsometer liefert erstens wesentlich genauere Werte für die Schichtdicke und hat außerdem im Brechungsindex einen Indikator auf die Zusammensetzung der Schicht. Er ist deswegen ein wichtiger Hinweis auf die Qualität des aufgedampften Siliziumnitrids. Da mit dem von M.Mendel gebauten Ellipsometer bisher nur einige Testmessungen durchgeführt wurden, sollte im Rahmen dieser Arbeit das Ellipsometer ausführlicher getestet werden.
Die Programme, mit denen die Ellipsometermeßwerte bisher ausgewertet wurden, sind recht umständlich in der Handhabung. Außerdem basiert die Arbeitsweise dieser Programme auf einem iterativen Verfahren, das seine Anfangswerte aus einem Ψ-Δ-Diagramm bezieht. Dieses Verfahren hat mehrere Nachteile, die das

Programm XEllip, das im Rahmen dieser Arbeit geschrieben wurde, zu umgehen versucht, indem ein analytischer Ansatz gewählt wird.
Die Rutherford-Rückstreu-Experimente wurden im ITN (Instituto Tecnológico e Nuclear) in Sacavem durchgeführt. Diese RBS-Messungen geben Aufschlüsse auf die Zusammensetzung und - bedingt - auch auf die Schichtdicke.

2 Ellipsometrie

2.1 Einführung in die Ellipsometrie

2.1.1 Aufbau und Meßmethode

Die Ellipsometrie ist eine zerstörungsfreie Methode zur Messung von Schichtdicke und Brechungsindex dünner Schichten. Das Prinzip besteht darin, die Änderung des Polarisationszustandes von Licht bei Reflexion oder Transmission zu messen. Ein Polarisationszustand ist gegeben durch die Phasen- und Amplitudenbeziehung zwischen zwei Komponenten des elektrischen Feldvektors, in die eine Lichtwelle zerlegt werden kann. Üblicherweise sind dies die senkrechte (s) und parallele (p) Komponente des Feldvektors in bezug auf die Einfallsebene des Lichtes. Bei einer Reflexion oder Transmission ändern diese beiden Komponenten in der Regel ihre Phasenlage zueinander und ihre Amplituden. Das von uns verwendete Ellipsometer ist ein sogenanntes Reflexions-Nullellipsometer. Die Polarisationsrichtung des linear polarisierten Lichtes nach der Reflexion wird dadurch bestimmt, daß das Licht durch einen Polarisationsfilter (den Analysator) zur Auslöschung gebracht wird.
Bei einem Ellipsometersystem sind immer zwei Polarisationsrichtungen dadurch ausgezeichnet, daß sich die Polarisationsrichtung vor und nach der Reflexion nicht ändert. Man nennt sie auch Eigenpolarisationen. Im Fall eines Reflexionsellipsometers sind diese Eigenpolarisationen das senkrecht und parallel polarisierte Licht in Bezug auf die Einfallsebene.
Die elektrischen Feldvektoren des einfallenden (i) und reflektierten (r) Lichtstrahls werden im folgenden mit E_{ip}, E_{rp}, E_{is} und E_{rs} bezeichnet (siehe auch Abbildung 2.1).

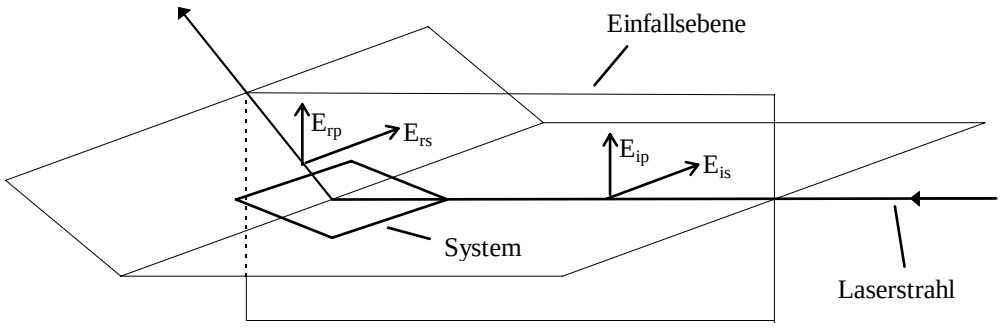


Abbildung 2.1: Der einlaufende und reflektierte Laserstrahl mit ihren jeweiligen Komponenten des Polarisationszustandes parallel (p) und senkrecht (s) zur Einfallsebene.

Der prinzipielle Aufbau eines Reflexions-Ellipsometers ist in Abbildung 2.2 zu sehen. Ein Helium-Neon-Laser sendet linear polarisiertes Licht der Wellenlänge λ = 632,8 nm aus. Danach kommt ein λ/4-Plättchen, dessen optische Achse um 45° gegenüber der Polarisationsebene des Lasers gedreht ist, um zirkular polarisiertes Licht herzustellen. Damit wird gewährleistet, daß das Licht nach dem folgenden Polarisator (einem Polarisationsfilter) unabhängig von seiner Winkelstellung immer die gleiche Amplitude hat. Nun wird mit einem weiteren λ/4-Plättchen elliptisch polarisiertes Licht hergestellt, das dann unter einem Winkel von 70° auf die Probe fällt und reflektiert wird. Nach der Reflexion wird die Lichtintensität mit einem Analysator (2. Polarisationsfilter) und einem Detektor (einer Siliziumphotodiode) gemessen.
Das Meßverfahren nutzt aus, daß die Vorgänge der Reflexion und Brechung an einer dünnen Schicht die Elliptizität der Lichtwelle verändern. Man stellt die Elliptizität mit dem Polarisator so ein, daß nach der Reflexion an der Probe linear polarisiertes Licht entsteht. Dieses kann man dann mit Hilfe des Analysators zur Auslöschung bringen, indem die Polarisationsebene senkrecht zur Schwingungsebene des Lichtes eingestellt wird. Im Idealfall ist so eine vollständige Lichtauslöschung zu erreichen. Daher stammt auch die Bezeichnung Nullellipsometrie.

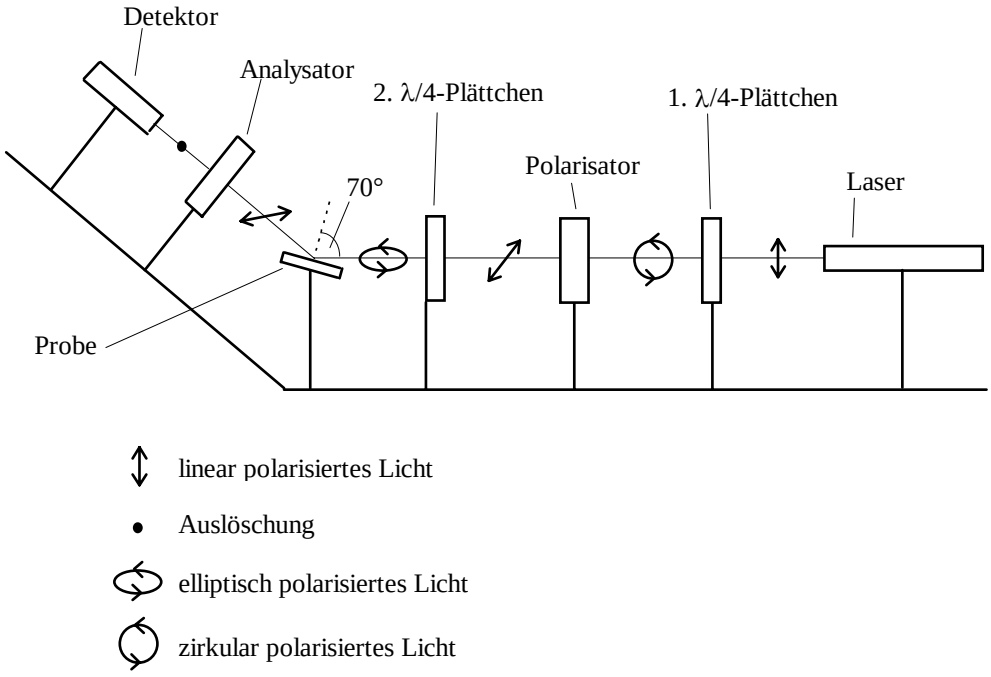


Abbildung 2.2 : Prinzipieller Aufbau eines Ellipsometers. Als polarisierte Lichtquelle dient ein He-Ne-Laser. Ein Polarisator und Kompensator (2. λ/4-Plättchen) erzeugen elliptisch polarisiertes Licht. Durch Reflexion am zu untersuchenden System wird die Elliptizität der Lichtwelle verändert und diese Änderung wird mit Analysator und Detektor nachgewiesen.

Praktisch geht man so vor, daß man Polarisator und Analysator so lange abwechselnd verstellt bis im Detektor ein Minimum angezeigt wird. Die Meßgrößen sind die Azimutwinkel von Polarisator und Analysator, β und α. Die Stellung des 2. $\lambda/4$-Plättchens ist durch den Winkel γ gegeben, um den die optische Achse des Plättchens gegen die Einfallsebene geneigt ist.

2.1.2 Theorie der Ellipsometrie

2.1.2.1 Das ellipsometrische System

Das System wird eindeutig durch zwei komplexe Größen R_p und R_s beschrieben. R_p (bzw. R_s) ist gegeben als das Verhältnis der Größe E_p (bzw. E_s) nach und vor der Reflexion, also

$$R_p = \frac{E_{rp}}{E_{ip}} \qquad ,bzw. \qquad R_s = \frac{E_{rs}}{E_{is}},$$

wobei p und s die Lage in Bezug auf die Einfallsebene bezeichnen. Jede elliptisch polarisierte Welle kann als Überlagerung von E_{ip} und E_{is} gesehen werden. Beide Wellen erfahren beim Durchgang durch das System unabhängig voneinander eine Änderung in Amplitude und Phase. Nach der Reflexion werden sie wieder überlagert. Es ist also möglich, die Elliptizität der Welle so zu wählen, daß nach der Reflexion kein Phasenunterschied mehr vorhanden ist, d.h. linear polarisiertes Licht vorliegt. Stellt man dann die Polarisationsebene des Analysators senkrecht hierzu, so erhält man eine völlige Auslöschung der Lichtwelle.
In dieser Auslöschungsstellung haben die Azimutwinkel der optischen Komponenten des Systems β, α, γ sowie der komplexen Zahl ρ, die definiert ist als das Verhältnis der Größen R_p und R_s, folgende Beziehung zueinander [AZZ77] :

$$\rho = \frac{R_p}{R_s} = -\tan\alpha \left(\frac{\tan\gamma - i\tan(\beta-\gamma)}{1 + i\tan\gamma\tan(\beta-\gamma)} \right) \qquad (2.1)$$

ρ wird auch als ellipsometrisches Verhältnis bezeichnet. Man erkennt, daß der Polarisatorwinkel β nur in Differenz mit γ auftritt, d.h. die Elliptizität des Lichtes hängt ausschließlich von der relativen Lage des Polarisators zum zweiten $\lambda/4$-Plättchen ab. Die Lage der „Ellipse“ im Raum wird durch den Term $\tan\gamma$ festgelegt. Der Winkel γ des zweiten $\lambda/4$-Plättchens ist also frei wählbar und wird auf 45° festgelegt. Dies vereinfacht die Gleichung (2.1) zu:

$$\rho = -\tan\alpha \; e^{-2i\left(\beta - \frac{\pi}{4}\right)} \qquad (2.2)$$

Im Experiment findet man zwei Winkelpaare, die diese Gleichung erfüllen. Werden Polarisator und Analysator so eingestellt, daß der Detektor das erste Minimum anzeigt, erfüllen die Azimutwinkel β_1 und α_1 die Gleichung (2.2). Für das Winkelpaar $\beta_2 = \beta_1 + \frac{\pi}{2}$ und $\alpha_2 = 2\pi - \alpha_1$ ändert sich die Gleichung nicht. An diesen Winkeln findet man im Experiment das zweite Intensitätsminimum.
Mit diesen gemessenen Winkeln kann man die sogenannten ellipsometrischen Winkel Ψ und Δ einführen:

$$\Psi = \frac{|\alpha_1| + |\alpha_2|}{2}, \tag{2.3}$$

$$\Delta = 2\pi - (\beta_1 + \beta_2). \tag{2.4}$$

Dies sind die beiden Meßgrößen der Ellipsometrie. Sie erfüllen die Auslöschungsbedingung (2.2) und bestimmen damit die Eigenschaften unseres optischen Systems in der sogenannten ellipsometrischen Gleichung:

$$\rho = \frac{R_p}{R_s} = \tan\Psi e^{i\Delta} \tag{2.5}$$

Die Äquivalenz zwischen (2.2) und (2.5) läßt sich durch Einsetzen beweisen, wenn man beachtet, daß α_1 zwischen 0° und 90° liegt, also positiv ist.

2.1.2.2 Fresnelsche Formeln und das Snellius´sche Brechungsgesetz

Wenn eine ebene Welle auf ein absorbierendes Medium trifft, läßt sich ihre Ausbreitung in diesem Medium mit Hilfe eines komplexen Brechungsindex $N = n - ik$ beschreiben. Hierbei ist *n* der reelle Brechungsindex und k der Absorptionskoeffizient.
Trifft diese ebene Welle auf die Grenzfläche zwischen zwei isotropen Medien (0) und (1) (mit den jeweiligen Brechungsindizes n_1 und n_2) mit dem Winkel φ_0 auf, so wird ein Teil unter dem Winkel φ_1 gebrochen. Die Beziehung zwischen diesen vier Größen wird im sogenannten Snellius´schen Brechungsgesetz beschrieben:

$$n_0 \sin\varphi_0 = n_1 \sin\varphi_1 \tag{2.6}$$

Für den Winkel der reflektierten Welle gilt: Einfallswinkel = Ausfallswinkel.
Das Verhalten der ebenen Welle an dieser Grenzfläche läßt sich auch beschreiben durch seine einfallenden, E_{ip} und E_{is}, seine reflektierten, E_{rp} und E_{rs}, und seine gebrochenen Komponenten des elektrischen Feldvektors, E_{tp} und E_{ts} (t = transmittiert). Wenn man die Stetigkeitsbedingungen der Tangentialkomponenten des

Lichtvektors und die vollständige Theorie der Reflexion, Brechung und Polarisation isotroper Medien berücksichtigt, dann erhält man die sogenannten Fresnel´schen Formeln. Diese geben die komplexwertigen Reflexions- und Brechungskoeffizienten für die p- und s-Polarisationsrichtungen an [POH83] :

$$r_{p01} = \frac{E_{rp}}{R_{ip}} = \frac{n_1 \cos\varphi_0 - n_0 \cos\varphi_1}{n_1 \cos\varphi_0 + n_0 \cos\varphi_1} \tag{2.7}$$

$$r_{s01} = \frac{E_{rs}}{R_{is}} = \frac{n_0 \cos\varphi_0 - n_1 \cos\varphi_1}{n_0 \cos\varphi_0 + n_1 \cos\varphi_1} \tag{2.8}$$

$$t_{p01} = \frac{E_{tp}}{E_{ip}} = \frac{2n_0 \cos\varphi_0}{n_1 \cos\varphi_0 + n_0 \cos\varphi_1} \tag{2.9}$$

$$t_{s01} = \frac{E_{ts}}{E_{is}} = \frac{2n_0 \cos\varphi_0}{n_0 \cos\varphi_0 + n_1 \cos\varphi_1} \tag{2.10}$$

2.1.2.3 Das beschichtete Substrat

Bei einem beschichteten Substrat hat man zwei solcher Grenzflächen: die erste zwischen der Umgebung und der Schicht; die zweite zwischen der Schicht und dem Substrat. Die Schichten haben die jeweiligen Brechungsindizes n_0, n_1 und n_2 (siehe Abbildung 2.3).

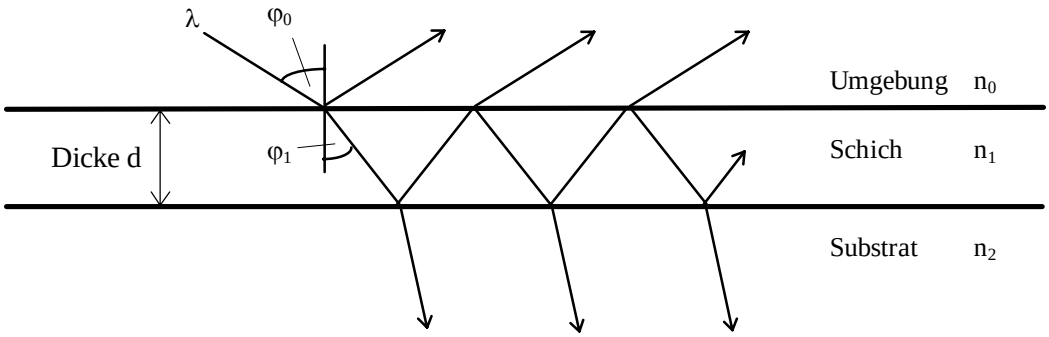


Abbildung 2.3: Verlauf einer ebenen Welle in einem beschichteten Substrat. λ: Wellenlänge; d: Schichtdicke; φ_0: Einfallswinkel = Ausfallswinkel; φ_1: Brechungswinkel; n_0, n_1, n_2: Brechungsindizes von Umgebung, Schicht und Substrat.

Eine ebene Welle der Wellenlänge λ kommt aus dem umgebenden Medium (0) und wird an der ersten Grenzfläche aufgespalten: sie wird dort teilweise reflektiert und teilweise in die Schicht hinein gebrochen. Der gebrochene Strahl erfährt in

der Schicht Mehrfachreflexion, an den Grenzflächen (1-0) und (1-2) wird allerdings immer ein kleiner Teil aus der Schicht herausgebrochen.
Die Reflexions- und Brechungsvorgänge werden, wie oben bereits erwähnt, beschrieben durch die Fresnelkoeffizienten r_{01}, t_{01}, r_{12}, t_{12}, r_{10} und t_{10}, wobei der erste Index für das Medium steht, aus dem der Strahl kommt, der zweite Index bezeichnet das Medium an dessen Grenzfläche die Reflexion (r) oder Brechung (t (engl.:) transmission) stattfindet.
Die mehrfach reflektierte Welle erfährt bei jedem Durchqueren der Schicht eine Phasenänderung von

$$\delta = 2\pi \frac{d}{\lambda} n_1 \cos\varphi_1. \qquad (2.11)$$

Die in das Medium (0) auslaufenden Teilstrahlen können als Produkt der Fresnelkoeffizienten, die die Reflexions- und Brechungsvorgänge dieses Teilstrahls charakterisieren, und dem Ausdruck $e^{-i\delta}$, für die auftretenden Phasenverschiebungen δ, beschrieben werden. Das bedeutet, daß ab der zweiten Teilwelle jeder Teilstrahl den zusätzlichen Ausdruck $r_{10}r_{12}e^{-2i\delta}$ erhält: für die zwei Reflexionen und die zwei Phasenverschiebungen in der Schicht.

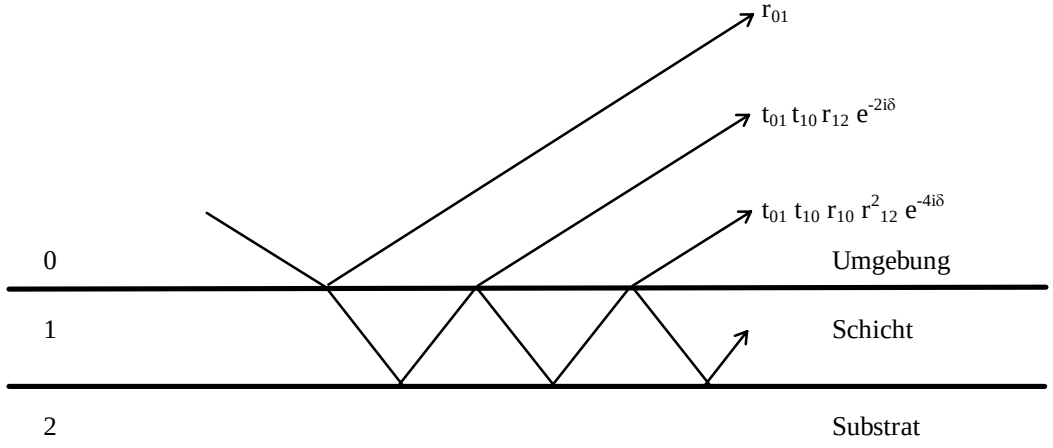


Abbildung 2.4: Die auslaufenden Wellen beschrieben durch die beteiligten Fresnelkoeffizienten. Der erste Index steht für das Medium, aus dem der Strahl kommt, der zweite Index bezeichnet das Medium an dessen Grenzfläche die Reflexion (r) oder Brechung (t (engl.:) transmission) stattfindet. Ab der zweiten Teilwelle erhält jede Teilwelle einen zusätzlichen Faktor $r_{10}r_{12}e^{-2i\delta}$ für die zwei Reflexionen und die zwei Phasenverschiebungen in der Schicht.

Die ersten drei auslaufenden Teilstrahlen ergeben sich damit zu r_{01}, $t_{01}t_{10}r_{12}e^{-2i\delta}$ und $t_{01}t_{10}r_{10}r^2{}_{12}e^{-4i\delta}$. Die Summe aller Partialwellen erzeugt die resultierende reflektierte Welle R.

$$R = r_{01} + t_{01}t_{10}r_{12}e^{-2i\delta} + t_{01}t_{10}r_{10}r_{12}^2e^{-4i\delta} + \cdots \qquad (2.12)$$

Mit Hilfe der unendlichen geometrischen Reihe kann man dies vereinfachen zu

$$R = r_{01} + \frac{t_{01}t_{10}r_{12}e^{-2i\delta}}{1 - r_{10}r_{12}e^{-2i\delta}} = \frac{r_{01} + r_{12}e^{-2i\delta}}{1 - r_{10}r_{12}e^{-2i\delta}} \qquad (2.13)$$

Das zweite Gleichheitszeichen gilt auf Grund der folgenden Beziehungen zwischen den Fresnel´schen Koeffizienten, die sich durch Einsetzen zeigen lassen.

$$1. \quad r_{10} = -r_{01} \qquad \text{und} \qquad 2. \quad t_{01}t_{10} = 1 - r_{10}^2 \qquad (2.14a,b)$$

Die Gleichung (2.13) ist allgemeingültig und darf deswegen auch getrennt für s- und p-polarisiertes Licht angewandt werden:

$$R_p = \frac{r_{01p} + r_{12p}e^{-2i\delta}}{1 - r_{10p}r_{12p}e^{-21\delta}} \qquad (2.15a)$$

$$R_s = \frac{r_{01s} + r_{12s}e^{-2i\delta}}{1 - r_{10s}r_{12s}e^{-21\delta}} \qquad (2.15b)$$

In diesen beiden Variablen ist jetzt die gesamte Information der Reflexions- und Brechungsvorgänge enthalten.

2.1.2.4 Schichtdicke und Brechungsindex

Wenn man die beiden eben hergeleiteten Größen R_p und R_s in die Gleichung (2.5) einsetzt erhält man:

$$\tan\Psi e^{i\Delta} = \rho = \frac{R_p}{R_s} = \frac{r_{01p} + r_{12p}e^{-21\delta}}{1 - r_{10p}r_{12p}e^{-2i\delta}} \cdot \frac{1 - r_{10s}r_{12s}e^{-2i\delta}}{r_{01s} + r_{12s}e^{-2i\delta}} \qquad (2.16)$$

Diese Gleichung verbindet die ellipsometrischen Winkel Ψ und Δ mit den Eigenschaften des optischen Systems, bestehend aus Umgebung / Film / Substrat. Diese Eigenschaften sind durch die folgenden Größen gegeben:

- die komplexen Brechungsindizes der Umgebung n_0, der Schicht n_1, und des Substrats n_2;
- die Schichtdicke d;
- die Vakuumwellenlänge des benutzten Laserlichts λ;
- den Einfallswinkel des Lichtes auf die Probe φ_0;

Man kann die funktionale Abhängigkeit der ellipsometrischen Winkel also folgendermaßen schreiben

$$\tan\Psi e^{i\Delta} = \rho\left(\lambda, \varphi_0, d, n_0, n_1, n_2\right), \tag{2.17}$$

wobei ρ durch die rechte Seite der Gleichung (2.16) gegeben ist.
Diese komplexwertige Gleichung kann in zwei reelle Gleichungen für Ψ und Δ separiert werden:

$$\Psi = \tan^{-1}\left|\rho\left(\lambda, \varphi_0, d, n_0, n_1, n_2\right)\right| \tag{2.18}$$

$$\Delta = \arg\left(\rho\left(\lambda, \varphi_0, d, n_0, n_1, n_2\right)\right) \tag{2.19}$$

$|\rho|$ und $\arg(\rho)$ stehen hier jeweils für den absoluten Betrag und das Argument der komplexen Funktion ρ [WIN55]. Für eine Ellipsometermessung lassen sich aus den gemessenen Werten β_1, α_1, β_2, α_2 die ellipsometrischen Winkel Ψ und Δ nach (2.3) und (2.4) berechnen. Weiterhin sind in unserem Fall die Größen λ, φ_0, n_0 und n_2 bekannt. Damit lassen sich im Prinzip aus den obigen Gleichungen (2.18) und (2.19) die beiden gesuchten Größen d und n_1 bestimmen. Hierzu wird üblicherweise ein Computerprogramm benutzt, das die Lösungen iterativ berechnet.

Abbildung 2.5: Δ-Ψ-Diagramm für eine mit SiO_2 beschichtete Siliziumprobe. Der Brechungsindex n_1 bestimmt die Kurven. Die Pfeile zeigen in Richtung der wachsenden Schichtdicke d. So entspricht z.B. das mit (Ψ_0,Δ_0) eingezeichnete Lösungspaar einer Dicke von 1000Å und einem Brechungsindex von 1,46 der gemessenen Siliziumoxidschicht.

Graphisch kann man den Zusammenhang auch in einem Δ-Ψ-Diagramm darstellen. Ψ nimmt Werte an zwischen 0° und 90°, Δ liegt zwischen 0° und 360°. In Abbildung 2.5 ist ein solches Δ-Ψ-Diagramm für eine Siliziumoxidschicht auf Silizium zu sehen [ANA75]: Der verwendete He-Ne-Laser hat die Wellenlänge λ = 632,8 nm, der Einfallswinkel φ_0 beträgt 70° und der komplexe Brechungsindex von Silizium ist $n_2 = 3{,}86 - i\cdot 0{,}02$. Die Lösungspaare n_1 und d liegen auf geschlossenen Kurven. Zu jeder Kurve gehört ein fester Brechungsindex. Auf diesen Kurven wird die Schichtdicke gegen den Uhrzeigersinn durchlaufen. Kurven, die zu unterschiedlichen Brechungsindizes gehören schneiden sich untereinander nicht, aber allen Kurven ist der Punkt mit der Dicke d = 0 gemeinsam. Geschlossene Kurven sind es eigentlich nur in der Projektion auf die Δ-Ψ-Ebene. Wenn man eine dritte Dimension hinzufügt, die der Schichtdicke zugeordnet ist, dann erhält man Spiralen. Dies bedeutet, daß für einen festen Brechungsindex die zugehörige Dicke d nur bis auf eine Periode D eindeutig bestimmt. D entspricht dem Dickenzuwachs innerhalb eines Durchlaufs auf der „geschlossenen“ Kurve in Abbildung 2.5. Rechnerisch kann man sie auch aus Gleichung (2.11) unter Zuhilfenahme des Snellius´schen Brechungsgesetzes erhalten.

$$D = d(\delta = \pi) = \frac{\lambda}{2}\frac{1}{\sqrt{n_1^2 - n_0^2 \sin^2 \varphi_0}} \qquad (2.20)$$

Als Beispiel: Die Periodendicke eines GaAs-Substrats, das mit Si_3N_4 beschichtet wurde, beträgt bei einem Einfallswinkel von $\varphi_0 = 70°$ und einer Wellenlänge von $\lambda = 632{,}8nm$ $D = 1758Å$.

2.2 Lösung der ellipsometrischen Gleichung

2.2.1 Einleitung

Eine andere Möglichkeit die Lösungspaare (n_1,d) zu erhalten wurde von Drolev [DRO94] entwickelt. Diese Methode führt die ellipsometrische Gleichung auf ein Polynom 5. Grades zurück. Das Problem ist dann darauf reduziert, die Nullstellen (oder Wurzeln) dieses Polynoms zu finden. Die Koeffizienten dieses Polynoms werden, wie in der oben erwähnten numerischen Methode auch, durch den Einfallswinkel φ_0, den reellen Brechungsindex des umgebenden Mediums (meistens Luft), den komplexen Brechungsindex des Substrats und das gemessene ellipsometrische Verhältnis ρ bestimmt.
Bei den numerischen (klassischen) Methoden hat man das Problem, daß die Lösung nur sehr langsam konvergiert oder überhaupt nicht erreicht wird. Dies hängt vor allem von der Wahl der Anfangswerte der gesuchten Parameter ab. Ein weiteres Problem stellt die Tatsache dar, daß es normalerweise mehr als eine mathematische oder physikalische Lösung für eine Messung des Systems gibt, d.h. es gibt mehrere Lösungen, die die gleichen Werte für Ψ und Δ ergeben. Man muß alle physikalisch sinnvollen Lösungen kennen, um zwischen den erhaltenen Lösung zu entscheiden, welche für das System die Zutreffende ist. Mit numerischen Methoden ist es schwierig alle Lösungen zu einem Problem zu finden oder im voraus zu wissen, wieviele Lösungen es gibt, da die Anzahl von System zu System verschieden sein kann. Es gibt also keine Garantie, daß alle Lösungen gefunden werden.
Bei der analytischen Methode wird ausgenutzt, daß die Lösung in zwei Schritte unterteilt werden kann: im ersten Schritt löst man eine Gleichung, die nur vom Brechungsindex abhängt, im zweiten Schritt wird die Dicke der Schicht aus diesem Brechungsindex berechnet. Man kann zeigen, daß sich die Gleichung für den Brechungsindex, die auf den ersten Blick recht kompliziert aussieht (ein Polynom 24. Grades), auf ein Polynom 5. Grades zurückführen läßt. Obwohl diese Methode nicht vollkommen analytisch ist (mindestens eine Nullstelle dieses Polynoms muß iterativ gefunden werden), ist sie direkt und komplett in dem Sinn, daß man von Anfang an weiß, daß es maximal fünf Nullstellen gibt (Polynom 5. Grades). Außerdem kann man von diesen fünf Lösungen gut erkennen, welche Lösungen sinnvoll sind und welche nicht. Die einzige Vorausetzung, die gemacht werden muß, ist die, daß der relative Brechungsindex $n = n_1/n_0$ reell, d.h. die Schicht transparent sein soll.

2.2.2 Herleitung des Polynoms 5. Grades

Um das Polynom 5. Grades herzuleiten, beginnt man mit einer Umformung von ρ in Gleichung (2.16). Man führt eine zusätzliche komplexe Variable X ein, alle anderen Variablen bezeichnen die üblichen Werte: Schichtdicke d, Brechungsindizes der Umgebung, Schicht, Substrat n_0, n_1, n_2, Wellenlänge λ, Einfallswinkel φ_0, und die Fresnel´schen Reflexionskoeffizienten r_{ij} für die Reflexion an der ij. Grenzschicht.

$$\rho = \frac{\left(Xr_{p12} + r_{p01}\right)\left(Xr_{s12}r_{s01} + 1\right)}{\left(Xr_{p12}r_{p01} + 1\right)\left(Xr_{s12} + r_{s01}\right)} \tag{2.21}$$

$$X = \exp\left(-\frac{4\pi i(\cos\varphi_1)\ d\ n_1}{\lambda}\right) \tag{2.22}$$

$$r_{p01} = \frac{-n_0\cos\varphi_1 + n_1\cos\varphi_0}{n_0\cos\varphi_1 + n_1\cos\varphi_0} \tag{2.23}$$

$$r_{p12} = \frac{-n_1\cos\varphi_2 + n_2\cos\varphi_1}{n_1\cos\varphi_2 + n_2\cos\varphi_1} \tag{2.24}$$

$$r_{s01} = \frac{-n_1\cos\varphi_1 + n_0\cos\varphi_0}{n_1\cos\varphi_1 + n_0\cos\varphi_0} \tag{2.25}$$

$$r_{s01} = \frac{-n_2\cos\varphi_2 + n_1\cos\varphi_1}{n_2\cos\varphi_2 + n_1\cos\varphi_1} \tag{2.26}$$

mit
$$\cos\varphi_j = \frac{\left[n_j^2 - n_0^2\left(\sin\varphi_0\right)^2\right]^{1/2}}{n_j} \tag{2.27}$$

Als ersten Schritt formt man Gleichung (2.21) in ein Polynom für X um:

$$aX^2 + bX + c = 0 \tag{2.28}$$

wobei die Koeffizienten wie folgt gegeben sind:

$$a = \left(-r_{s01} + r_{p01}\rho\right)r_{p12}r_{s12} \tag{2.29}$$

$$b = -r_{p12} - r_{p01}r_{s12}r_{s01} + \rho\left(r_{s12} + r_{p12}r_{p01}r_{s01}\right) \tag{2.30}$$

$$c = -r_{p01} + r_{s01}\rho \tag{2.31}$$

Jetzt benutzt man die Tatsache, daß $X^\dagger = 1/X$ wenn der relative Index $n = n_1/n_0$ reell ist und stellt die komplex Konjugierte der Gleichung (2.28) auf:

$$c^\dagger X^2 + b^\dagger X + a^\dagger = 0 \tag{2.32}$$

Mit den Gleichungen (2.28) und (2.32) kann man X^2 eliminieren und erhält einen Ausdruck für *X*, der keine Wurzeln enthält:

$$X = \frac{aa^\dagger - cc^\dagger}{-ab^\dagger + bc^\dagger} \tag{2.33}$$

Man muß betonen, daß Gleichung (2.33) nur *eine* Lösung ergibt, anstelle der zwei, die man auf dem üblichen Weg mit der p/q-Formel erhalten würde.
Man benutzt wieder $X^\dagger = 1/X$ und erhält eine Gleichung, die nur noch vom relativen Brechungindex n abhängt und unabhängig ist von der Dicke d. Diese muß für n gelöst werden :

$$\left(-ab^\dagger + bc^\dagger\right)^\dagger \left(-ab^\dagger + bc^\dagger\right) - \left(aa^\dagger - cc^\dagger\right)^2 = 0 \tag{2.34a}$$

oder

$$-|a|^4 + |a|^2|b|^2 - |c|^4 + 2|a|^2|c|^2 + |b|^2|c|^2 - ac\left(b^\dagger\right)^2 - b^2a^\dagger c^\dagger = 0. \tag{2.34b}$$

Jetzt werden die folgenden Substitutionen durchgeführt:

$$t = \frac{n_1 \cos\varphi_1}{n_0 \cos\varphi_0} \tag{2.35}$$

$$x = -\tan\varphi_0 \tan\varphi_2 \tag{2.36}$$

$$y = -\frac{n_2 \cos\varphi_2}{n_0 \cos\varphi_0} \tag{2.37}$$

$$z = -\frac{\rho - 1}{\rho + 1} \tag{2.38}$$

Im zu lösenden System hat man eine reelle Unbekannte n_1 und fünf unabhängig bekannte Parameter (eine für den Einfallswinkel φ_0, zwei für den komplexen Brechungsindex des Substrates n_2 und zwei für das komplexe ellipsometrische Verhältnis ρ). Mit den Substitutionen (2.35) bis (2.38) hat man scheinbar eine reelle unbekannte t und sechs Parameter (Real- und Imaginärteile von x, y und z). x und y sind aber durch die Beziehung $xy = x^\dagger y^\dagger = tan^2\varphi$ verbunden, d.h. es sind immer noch fünf freie Parameter vorhanden.
Mit diesen Definitionen werden die Fresnelkoeffizienten und ρ zu:

$$r_{p01} = \frac{(t-1)(t-xy)}{(t+1)(t+xy)} \tag{2.39}$$

$$r_{p12} = -\frac{(t+x)(t+y)}{(t-x)(t-y)} \tag{2.40}$$

$$r_{s01} = -\frac{t-1}{t+1} \tag{2.41}$$

$$r_{s12} = \frac{t+y}{t-y} \tag{2.42}$$

$$\rho = -\frac{z-1}{z+1} \tag{2.43}$$

und die Koeffizienten a, b und c von Gleichung (2.28) können geschrieben werden als:

$$a = -(t+1)(t-1)(t+x)(t+y)^2(t+xyz)C \tag{2.44}$$

$$b = 2(t+y)(t-y)\left[t^4 + (2xy + x^2yz + 2xz + 1)t^2 + x^2yz\right]C \tag{2.45}$$

$$c = -(t+1)(t-1)(t-x)(t-y)^2(t-xyz)C \tag{2.46}$$

$$C = \frac{\rho+1}{(t+1)^2(t-x)(t-y)^2(t+xy)} \tag{2.47}$$

wobei C ein gemeinsamer Faktor ist, der aus den Gleichungen (2.14) herausgekürzt werden kann. Wenn man nun die ausgeschriebene Form von Gleichung (2.14b) nimmt, führt jeder Term zu einem Polynom 24. Grades für t. Glücklicherweise kann man wegen einer Vorzeichensymmetrie in a und c vieles vereinfachen.
Nachdem man (2.44) bis (2.46) in (2.14b) eingesetzt hat, läßt sich ein weiterer gemeinsamer Faktor $(t-1)^2(t+1)^2$ herauskürzen, sodaß nun ein Polynom 20. Grades vorliegt:

$$\left(-a_1 b_1^\dagger + b_1 c_1^\dagger\right)^\dagger \left(-a_1 b_1^\dagger + b_1 c_1^\dagger\right) - \left(a_1 a_1^\dagger - c_1 c_1^\dagger\right)^2 \left(t^2 - 1\right)^2 = 0 \tag{2.48}$$

wobei hier die Koeffizienten definiert sind als

$$a_1 = -(t+x)(t+y)^2(t+xyz) \tag{2.49}$$

$$b_1 = 2(t+y)(t-y)\left[t^4 + (2xy + x^2yz + 2xz + 1)t^2 + x^2yz\right] \tag{2.50}$$

$$c_1 = -(t-x)(t-y)^2(t-xyz). \tag{2.51}$$

Nun kann man diese Terme ausmultiplizieren und so allmählich den Grad des Polynoms reduzieren. Gleichungen (2.49) bis (2.50) werden zu

$$a_1 = -t^4 - \alpha_3 t^3 - \alpha_2 t^2 - \alpha_1 t - \alpha_0 \tag{2.52}$$

$$b_1 = 2\left(t^6 + \alpha_4 t^4 + \alpha_5 t^2 - \alpha_0\right) \tag{2.53}$$

$$c_1 = -t^4 + \alpha_3 t^3 - \alpha_2 t^2 + \alpha_1 t - \alpha_0 \tag{2.54}$$

mit den Koeffizienten:

$$\alpha_0 = x^2y^3z$$

$$\alpha_1 = xy^2 + xy^3z + 2x^2y^2z$$

$$\alpha_2 = y^2 + 2xy + 2xy^2z + x^2yz$$

$$\alpha_3 = x + 2y + xyz$$

$$\alpha_4 = -y^2 + 2xy + 2xz + x^2yz + 1$$

$$\alpha_5 = -2xy^3 - y^2 - x^2y^3z - 2xy^2z + x^2yz \tag{2.55}$$

Die Summanden in Gleichung (2.48) werden getrennt betrachtet. Im ersten Teil des zweiten Summanden läßt die Symmetrie zwischen a_1 und c_1 die geraden Potenzen von t verschwinden, von der 8. Potenz bis zum konstanten Term. Übrig bleibt das folgende:

$$a_1 a_1^\dagger - c_1 c_1^\dagger = 2\left(g_6 t^6 + g_4 t^4 + g_2 t^2 + g_0\right)t \tag{2.56}$$

Die verbleibenden Koeffizienten g sind alle reell:

$$g_0 = \alpha_0 \alpha_1^\dagger + \alpha_1 \alpha_0^\dagger$$

$$g_2 = \alpha_0\alpha_3^\dagger + \alpha_1\alpha_2^\dagger + \alpha_2\alpha_1^\dagger + \alpha_3\alpha_0^\dagger$$

$$g_4 = \alpha_1 + \alpha_2\alpha_3^\dagger + \alpha_3\alpha_2^\dagger + \alpha_1^\dagger$$

$$g_6 = \alpha_3 + \alpha_3^\dagger \qquad (2.57)$$

Quadrieren der Gleichung (2.56) führt dazu, daß man nur Terme mit geraden Potenzen von t erhält, und die Multiplikation mit $\left(t^2-1\right)^2$ führt auf ein Polynom 18. Grades mit ausschließlich reellen Koeffizienten.
Der erste Summand von Gleichung (2.48) wird folgendermaßen behandelt: die Ausmultiplizieren von $c_1^\dagger b_1 - a_1 b_1^\dagger$ läßt ebenfalls die höchste Potenz und den konstanten Term herausfallen. Die restlichen Koeffizienten sind abwechselnd reell und imaginär. Es fällt auf, daß der erste und der letzte Term die gleichen Koeffizienten haben wie Gleichung (2.56):

$$c_1^\dagger b_1 - a_1 b_1^\dagger = 2\left(g_6 t^8 + if_7 t^7 + f_6 t^6 + if_5 t^5 + f_4 t^4 + if_3 t^3 + f_2 t^2 + if_1 t - g_0\right)t \qquad (2.58)$$

Die Koeffizienten f sind alle reell:

$$f_1 = \left(-\alpha_0\alpha_5^\dagger - \alpha_0\alpha_2^\dagger + \alpha_5\alpha_0^\dagger + \alpha_2\alpha_0^\dagger\right)i$$

$$f_2 = \alpha_1\alpha_5^\dagger - \alpha_0\alpha_3^\dagger + \alpha_5\alpha_1^\dagger - \alpha_3\alpha_0^\dagger$$

$$f_3 = \left(-\alpha_0 - \alpha_2\alpha_5^\dagger - \alpha_0\alpha_4^\dagger + \alpha_5\alpha_2^\dagger + \alpha_4\alpha_0^\dagger + \alpha_0^\dagger\right)i$$

$$f_4 = \alpha_3\alpha_5^\dagger + \alpha_1\alpha_4^\dagger + \alpha_5\alpha_3^\dagger + \alpha_4\alpha_1^\dagger$$

$$f_5 = \left(\alpha_5 - \alpha_0 - \alpha_5^\dagger - \alpha_2\alpha_4^\dagger + \alpha_4\alpha_2^\dagger + \alpha_0^\dagger\right)i$$

$$f_6 = \alpha_1 + \alpha_3\alpha_4^\dagger + \alpha_4\alpha_3^\dagger + \alpha_1^\dagger$$

$$f_7 = \left(\alpha_4 - \alpha_2 - \alpha_4^\dagger + \alpha_2^\dagger\right)i \qquad (2.59)$$

Wenn die Gleichung (2.58) mit ihrem komplex konjugierten multipliziert wird, lassen die alternierenden reellen und imaginären Koeffizienten die ungeraden Potenzen von t verschwinden und man erhält ein weiteres Polynom 18. Grades mit reellen Koeffizienten.
Die zwei Polynome - Gleichung (2.56) quadriert und Gleichung (2.58) multipliziert mit seinem komplex konjugierten - enthalten beide den Term $4t^2$, den man herauskürzen kann. Dies reduziert den Grad des Polynoms von 18 auf 16.

Es sind jetzt nur noch gerade Potenzen von t vorhanden und es bietet sich an eine weitere Variable s einzuführen mit $s = t^2$. Damit hat man nun ein Polynom 8. Grades in *s*, das definiert ist als

$$s = \frac{n_1^2\left(\cos\varphi_1\right)^2}{n_0^2\left(\cos\varphi_0\right)^2} \qquad \text{oder} \qquad s = \left(xy+1\right)\varepsilon - xy \tag{2.60a}$$

oder $$\varepsilon = s\,\left(\cos\varphi_0\right)^2 + \left(\sin\varphi_0\right)^2 \tag{2.60b}$$

wobei $\varepsilon = \left(n_1 / n_0\right)^2$ die relative dielektrische Konstante und $xy = \tan^2\varphi$ ist.
Wenn die Gleichung (2.48) wieder zusammengesetzt wird, erkennt man, daß durch die Differenz die höchste und niedrigste Potenz herausfallen, da die jeweiligen Koeffizienten übereinstimmen. Das resultierende Polynom ist ein Polynom 7. Grades ohne konstanten Term, also ein Polynom 6. Grades, nachdem die triviale Nullstelle herausgekürzt wurde. Dieses Polynom hat nun folgende Form:

$$j_6 s^6 + j_5 s^5 + j_4 s^4 + j_3 s^3 + j_2 s^2 + j_1 s + j_0 = 0 \tag{2.61}$$

wobei die j alle reell sind und mit den f und g wie folgt zusammenhängen:

$$j_0 = f_1^2 + 2g_0^2 - 2f_2 g_0 - 2g_2 g_0$$

$$j_1 = f_2^2 + 2f_3 f_1 - g_2^2 - g_0^2 - 2f_4 g_0 - 2g_4 g_0 + 4g_2 g_0$$

$$j_2 = f_3^2 + 2f_4 f_2 + 2f_5 f_1 + 2g_2^2 - 2g_4 g_2 - 2f_6 g_0 - 2g_6 g_0 + 2g_4 g_0 - 2g_2 g_0$$

$$j_3 = f_4^2 + 2f_5 f_3 + 2f_6 f_2 + 2f_7 f_1 - g_4^2 - g_2^2 - 2g_6 g_2 + 4g_4 g_2 + 2g_6 g_0 - 2g_4 g_0$$

$$j_4 = f_5^2 + 2f_6 f_4 + 2f_7 f_3 + 2f_2 g_6 + 2g_4^2 - 2g_6 g_4 + 4g_6 g_2 - 2g_4 g_2 - 2g_6 g_0$$

$$j_5 = f_6^2 + 2f_7 f_5 - g_6^2 + 2f_4 g_6 - g_4^2 + 4g_6 g_4 - 2g_6 g_2$$

$$j_6 = f_7^2 + 2g_6^2 + 2f_6 g_6 - 2g_6 g_4 \tag{2.62}$$

Man könnte das Polynom auch mit der physikalischen Variable ε statt mit s darstellen, indem man Gleichung (2.60a) benutzt. Dies führt zu einem Polynom mit dem konstanten Term

$$j_6 x^6 y^6 - j_5 x^5 y^5 + j_4 x^4 y^4 - j_3 x^3 y^3 + j_2 x^2 y^2 - j_1 xy + j_0 \tag{2.63}$$

Dies ist Gleichung (2.61) mit $s = -xy$. Es ist möglich zu zeigen, daß dieser Term immer 0 ist. Um dies zu zeigen muß man die j´s direkt in x, y und z ausdrücken, also die Gleichungen (2.55), (2.57) und (2.59) in (2.63) einsetzen. Man erhält einen Ausdruck, der insgesamt 2047 Terme enthält, die sich alle wegheben. Das Polynom hat demnach auch keinen konstanten Term und man kann wieder die triviale Nullstelle herausdividieren. Letztendlich bleibt ein Polynom 5. Grades für die relative dielektrische Konstante übrig :

$$k_5\varepsilon^5 + k_4\varepsilon^4 + k_3\varepsilon^3 + k_2\varepsilon^2 + k_1\varepsilon + k_0 = 0 \qquad (2.64)$$

mit

$$k_0 = j_1 - 6j_6x^5y^5 + 5j_5x^4y^4 - 4j_4x^3y^3 + 3j_3x^2y^2 - 2j_2xy$$

$$k_1 = \left(j_2 + 15j_6x^4y^4 - 10j_5x^3y^3 + 6j_4x^2y^2 - 3j_3xy\right)\left(xy+1\right)$$

$$k_2 = \left(j_3 - 20j_6x^3y^3 + 10j_5x^2y^2 - 4j_4xy\right)\left(xy+1\right)^2$$

$$k_3 = \left(j_4 + 15j_6x^2y^2 - 5j_5xy\right)\left(xy+1\right)^3$$

$$k_4 = \left(j_5 - 6j_6xy\right)\left(xy+1\right)^4$$

$$k_5 = j_6\left(xy+1\right)^5 \qquad (2.65)$$

Das Polynom in Gleichung (2.64) hat nur reelle Koeffizienten und deswegen entweder eine, drei oder fünf reelle Lösungen. Nur die reellen Lösungen, die $\varepsilon \geq \sin^2\varphi_0$ erfüllen, führen zu $XX^\dagger = 1$, während alle anderen Nullstellen diese erste Einschränkung, die man für die Herleitung gemacht hat, nicht erfüllen. In den Fällen, die $\varepsilon < \sin^2\varphi_0$ erfüllen, findet Totalreflexion in der Schicht statt. Dies ist ein Fall, der hier nicht weiter besprochen werden soll, da dies den Rahmen dieser Einführung sprengen würde.
Für jede gültige Lösung erhält man die Schichtdicke, indem man Gleichung (2.22) nach d auflöst:

$$d = \left(\frac{1}{4}\frac{\lambda}{\pi\ \left(\cos\varphi_1\right)\ n_1}\right) i\ln X + \frac{1}{2}\frac{m\lambda}{\left(\cos\varphi_1\right)\ n_1} \qquad (2.66)$$

Die Schichtdicke ist nur bekannt bis auf ein ganzzahlig Vielfaches von $\lambda / 2\left(\cos\varphi_1\right)n_1$ bekannt. Dies ist ein Wert der die Weglänge des Laserlichtes in der Schicht um eine Wellenlänge ändert, *m* ist eine ganze Zahl. Um *X* zu berechnen könnte man die Gleichungen (2.29) bis (2.31) verwenden und die

Reflexionskoeffizienten ausrechnen. Eine andere Möglichkeit besteht darin die rechte Seite von Gleichung (2.33) zu benutzen und die Substitutionen einzusetzen. So kann man X mit den bereits bekannten Koeffizienten ausdrücken :

$$\frac{1}{X} = \frac{g_6 t^8 + f_6 t^6 + f_4 t^4 + f_2 t^2 - g_0 + i\left(f_7 t^6 + f_5 t^4 + f_3 t^2 + f_1\right)t}{\left(t^2 - 1\right)\left(g_6 t^6 + g_4 t^4 + g_2 t^2 + g_0\right)} \qquad (2.67a)$$

oder auch (da $1/X = X^{\dagger}$ gilt):

$$X = \frac{g_6 t^8 + f_6 t^6 + f_4 t^4 + f_2 t^2 - g_0 - i\left(f_7 t^6 + f_5 t^4 + f_3 t^2 + f_1\right)t}{\left(t^2 - 1\right)\left(g_6 t^6 + g_4 t^4 + g_2 t^2 + g_0\right)} \qquad (2.67b)$$

In diesem Fall wird Gleichung (2.67b) benutzt. Der Term $i \ln X$ in (2.66) ist die negative Phase von X [1)]. Wenn man die Phase nach $\arctan\left[(\mathrm{Im}\, X)/(\mathrm{Re}\, X)\right]$ berechnen will, ist Gleichung (2.67) gut zu verwenden, da Real- und Imaginärteil bereits getrennt vorliegen (*i* ist der einzige komplexe Term). Man muß allerdings aufpassen, daß man den richtigen Quadranten - bestimmt durch die Vorzeichen von Real- und Imaginärteil - für die Phase wählt.

Man könnte den Term $(1+xy)^n$ von den k_n trennen und das Polynom für $(1+xy)\varepsilon$ lösen, aber die *k´*s sind ausgewogener, wenn sie nach (2.65) berechnet werden. Die Berechnung der polynomialen Koeffizienten ist direkt mit den Gleichungen (2.35)-(2.38), (2.55), (2.57), (2.59), (2.62) und (2.65). Algorithmen um Nullstellen eines Polynoms zu berechnen lassen sich in diverser Literatur finden, so daß man leicht die möglichen physikalischen Lösungen findet.

Diese Herleitung mag etwas kompliziert erscheinen, aber sie besteht lediglich aus der expliziten Ausmultiplikation und Aufsummierung der Polynome. Hier ist eine kurze Zusammenfassung der Herleitung:

Das quadratische Polynom in *X* (Gleichung (2.28)) hat nur dann eine Lösung mit Betrag 1, wenn man die Bedingung (2.34a) für *a, b* und *c* aufstellt. Nachdem man den gemeinsamen Faktor $|C|^4$ herausgekürzt hat, erhält man ein Polynom in *t*. Nach Einsetzen von *a, b, c* in (2.14a), läßt sich weiterhin der Term $(t+1)^2(t-1)^2$ herauskürzen und man erhält Gleichung (2.28). Die Expansion des Polynoms benötigt nur reines Ausmultiplizieren und Aufaddieren der diversen Koeffizienten und führt schlußendlich auf ein Polynom 5. Grades für die dielektrische Konstante der Schicht.

[1)] Es gilt $\ln z = \ln\left(|r|e^{i\varphi}\right) = \ln|r| + i\varphi$ für komplexe Zahlen. Für *X* gilt aber $XX^{\dagger} = |X| = 1$, d.h. der Logarithmus des Betrages $\ln|r|$ ist in diesem Fall gleich 0. Hieraus folgt: $i \cdot \ln X = i \cdot i\varphi = -\varphi$.

2.3 Erklärungen zum Quellcode des Programms

Das Programm wurde erstellt mit MS Visual C++, das einen Großteil der Windowsoberfläche automatisch generiert, d.h. eine „leere“ Windows-Anwendung bereits zur Verfügung stellt, die nur noch „gefüllt“ werden muß. Die folgenden Dateien sind Teil des Projekts XEllip:

- calc.h / calc.cpp
- complex.h / complex.cpp
- dmwerte.h / dmwerte.cpp
- utility.h / utility.cpp
- mainfrm.h / mainfrm.cpp
- stdafx.h / stdafx.cpp
- xellidoc.h / xellidoc.cpp
- xellip.h / xellip.cpp
- xellivw.h / xellivw.cpp

Der Quellcode dieser Dateien befindet sich im Anhang.
Jeder der neun Teile besteht aus zwei Dateien. Erstens aus einer sogenannten Header-Datei (*.h), in der die Prozeduren und Funktionen deklariert werden und zweitens aus einer CPP-Datei (*.cpp) in der die Routinen dann implementiert werden.
Die letzten fünf Dateien sind die Dateien, die von Visual C++ generiert wurden, um die Windowsoberfläche zu verwalten. Die anderen vier beziehen sich auf die Berechnung der gesuchten Werte. Dmwerte.h und Dmwerte.cpp werden ebenfalls zu einem großen Teil von Visual C++ generiert und verwalten den Dialog, der kreiert wurde, um die Werte ein- und auch auszugeben. Die interne Verwaltung muß allerdings noch „per Hand“ geschrieben werden. Utility.h und Utility.cpp enthalten nützliche Konstanten und Routinen, die von der Rechnung genutzt werden. In Complex.h und Complex.cpp ist eine Klasse für komplexe Zahlen auf Objektorientierter Basis (OOP : Objektorientierte Programmierung) geschrieben worden, die später auch für andere Programme als Bibliothek dienen kann. In Calc.h und Calc.cpp ist dann die Rechnung aus dem vorigen Kapitel realisiert worden, die im einzelnen die folgenden Schritte beinhaltet:

1. Die komplexen Zahlen *x*, *y* und *z* werden aus n_0, n_2, φ_0 und dem gemessenen ρ berechnet (Gleichungen (2.36) bis (2.38)).
2. Die komplexen α werden aus *x*, *y* und *z* berechnet. (Gleichungen (2.55)).
3. Die reellen *g* werden aus den α berechnet. (Gleichungen (2.57)).
4. Die reellen *f* werden aus den α berechnet. (Gleichungen (2.59)).
5. Die reellen *j* werden aus den *f* und *g* berechnet (Gleichungen (2.62)).
6. Die reellen *k* werden aus den *j* berechnet (Gleichungen (2.65)).
7. Die Nullstellen des Polynoms werden gesucht (Gleichung (2.64)).
8. Für jede gültige Nullstelle $\varepsilon>0$:

- Der Brechungsindex $n_1 = +n_0\sqrt{\varepsilon}$ und *t* werden berechnet (Gleichung (35)).
- Die Variable *X* wird berechnet (Gleichung (67)).
- Die Schichtdicke *d* wird berechnet (Gleichung (66)).

Spezielle Fälle werden von diesem Program bisher noch nicht bearbeitet, wie z.B. die Reflexion bei Winkeln, die größer sind als der kritische Winkel, oder der Fall, daß totale interne Reflexion in der Schicht auftritt.
Hier ein Beispiel für die Werteeingabe im Programm:

2.4 Messungen

2.4.1 Herstellung der Proben

Die Proben zur Untersuchung mit Hilfe der Ellipsometrie wurden mit einer LPCVD-Anlage (Low-Pressure-Chemical-Vapour-Deposition) hergestellt, die auch als RTA-Anlage (Rapid-Thermal-Annealing) genutzt werden kann. Diese Anlage wurde von G. Marx gebaut [MAR90]. Benutzt wurden ca. $1 \times 1 cm^2$ große Stücke aus Silizium und aus Galliumarsenid, die dann unter leicht unterschiedlichen Bedingungen mit einer Siliziumnitridschicht (engl.: cap) bedampft wurden. Die Variationen bestanden in der Cap-Dauer und leicht geänderten Temperaturen. In der Tabelle sind außer dem Namen der Probe, der Temperatur und der Cap-Dauer noch die Einstellungen der Anlage angegeben.

Name	T[°C]	Dauer[s]	US	SW	OS	RSp[V]	Vakuum[10^{-5}mbar]
GaAs 2/1.	660	30	671	680	682	3,35	4,2
GaAs 2/2.	660	35	671	680	682	3,35	4,3
GaAs 2/3.	660	40	671	680	682	3,35	3,8
GaAs 2/4.	660	45	672	680	682	3,35	4,1
GaAs 2/5.	660	50	672	680	682	3,35	4,8
GaAs 2/6.	660	55	672	680	682	3,35	4,3
GaAs 2/7.a	660	60	672	680	682	3,35	4,5
GaAs 2/7.b	660	60	672	680	682	3,35	4,5
GaAs 2/8.a	660	65	672	680	682	3,35	4,5
GaAs 2/8.b	660	65	672	680	682	3,35	4,5
GaAs 2/9.a	660	70	672	680	682	3,35	5,5
GaAs 2/9.b	660	70	672	680	682	3,35	5,5
GaAs 2/11.	660	55	672	680	682	3,35	5,6
GaAs 2/12.	660	50	672	680	682	3,35	5,2

Tabelle 2.1 : Cap-Bedingungen der GaAs-Proben. Die Temperatur ist in °C angegeben, ebenso wie die Einstellungen des Temperaturreglers: US : untere Schwelle, SW : Sollwert, OS : obere Schwelle. Die Dauer ist in Sekunden und die Regelspannung (Rsp) in Volt dargestellt. Das Vakuum vor dem Füllen der Kammer mit den Gasen ist in 10^{-5}mbar gemessen.

Die Abkürzungen US, SW und OS stehen hierbei jeweils für untere Schwelle, Sollwert und obere Schwelle. Diese Werte werden an der Temperatursteuereinheit eingestellt, die von A. Möller [MÖL92] gebaut wurde. Diese regelt den Graphitstreifen, auf dem die Probe liegt, auf die gewünschte Temperatur ein. Man muß beachten, daß der Sollwert nicht die Temperatur angibt. In der Arbeit von G. Marx sind Tabellen angegeben, die ein Umrechnung erlauben. Der Vakuumwert ist der Wert, der vor dem Einlassen der Cap-Gase in der Kammer vorliegt.

Name	T[°C]	Dauer[s]	US	SW	OS	RSp[V]	Vakuum[10^{-5}mbar]
Si 2/1.	660	60	665	680	682	3,35	4,0
Si 2/2.	660	50	665	680	682	3,35	4,0
Si 2/3.	660	40	669	680	682	3,35	3,3
Si 2/4.	660	30	671	680	682	3,35	4,2
Si 2/5.	660	70	671	680	682	3,35	3,8
Si 2/6.	660	65	671	680	682	3,35	3,1
Si 2/7.	660	55	671	680	682	3,35	4,1
Si 2/8.	660	45	671	680	682	3,35	3,7
Si 2/9.	660	35	671	680	682	3,35	3,5
Si 3/1.	670	30	682	690	692	3,40	3,8
Si 1/1.	650	30	658	669	671	3,35	3,5
Si 1/3.	650	50	658	669	671	3,35	4,0
Si 1/4.	650	50	658	669	671	3,35	4,7
Si 1/5.	650	90	658	669	671	3,35	4,0
Si 1/6.	650	50	658	669	671	3,35	3,9

Tabelle 2.2 : Cap-Bedingungen der Si-Proben. Die Temperatur ist in °C angegeben, ebenso wie die Einstellungen des Temperaturreglers: US : untere Schwelle, SW : Sollwert, OS : obere Schwelle. Die Dauer ist in Sekunden und die Regelspannung (RSp) in Volt dargestellt. Das Vakuum vor dem Belüften der Kammer mit den Gasen ist in 10^{-5}mbar gemessen.

2.4.2 Meßergebnisse

2.4.2.1 Bonner Ellipsometer

Die ersten Ellipsometermessungen wurden mit dem Bonner Ellipsometer gemacht, das von M.Mendel [MEN96] gebaut wurde. In der Tabelle sind die zwei Paare von Polarisator- und Analysatorwinkel β_1, α_1 und α_2, β_2 und die mit dem Programm XEllip berechneten Werte für Schichtdicke (in Å) und Brechungsindex angegeben. Die Proben wurden alle an den dicksten Stelle ihrer Caps gemessen, soweit dies an der Farbe zu erkennen war. In [MEN96] sind Tabellen angegeben, die eine sehr grobe Abschätzung (auf ca. 100Å) der Schichtdicke anhand der Farbe erlauben. Die Einstellung der Winkel im Polarisator und Analysator sind mit Hilfe eines Nonius auf 0,08° genau. Ebenfalls in [MEN96] ist eine ausführliche Fehlerdiskussion aufgeführt, die den Fehler in der Schichtdickenmessung auf ca. 5 Å und den Fehler im Brechungsindex auf etwa als 0.0005 beziffert.

Name	β_1	α_1	β_2	α_2	n	d
GaAs 2/1.	179,83	161,75	273,25	201,08	2,003	357,301
GaAs 2/2.	109,33	28,41	196,66	334,66	2,108	561,411
GaAs 2/3.	113,58	30,66	201,0	331,83	2,109	615,601
GaAs 2/4.	119,5	33,75	207,33	329,08	2,120	677,129
GaAs 2/5.	124,41	36,33	212,16	326,16	2,105	729,468
GaAs 2/6.	107,0	28,66	194,91	335,0	2,063	552,040
GaAs 2/7.a	124,66	36,0	213,5	324,58	2,097	739,976
GaAs 2/7.b	138,0	36,58	229,08	323,58	2,117	838,312
GaAs 2/8.a	154,66	28,16	249,0	330,66	2,149	984,352
GaAs 2/8.b	151,33	31,16	245,66	328,58	2,127	959,325
GaAs 2/9.a	146,41	32,41	239,33	326,25	2,142	899,467
GaAs 2/9.b	157,33	26,83	252,25	332,25	2,149	1020,908
GaAs 2/11.	132,25	35,08	222,0	325,16	2,148	776,113
GaAs 2/12.	134,83	37,58	225,0	322,33	2,098	821,545

Tabelle 2.3 : Meßdaten des Bonner Ellipsometers für die GaAs-Proben. Aufgeführt werden die zwei Meßpaare von Polarisator- und Analysatorwinkel in Grad, Brechungsindex und Schichtdicke in Å, beide berechnet mit dem Programm XEllip.

Name	β_1	α_1	β_2	α_2	n	d
Si 2/1.(1.)	125,66	36,66	214,83	325,16	2,01	766,871
Si 2/1.(2.)	119325	34,75	209,25	327,41	2,094	717,296
Si 2/2.	108,08	29,25	196,16	334,33	2,065	587,466
Si 2/3.	189,08	158,25	281,08	205,41	2,040	491,793
Si 2/4.	64,16	12,33	151,91	347,91	1,121	272,313
Si 2/5.	168,08	165,0	261,66	197,16	1,774	260,878
Si 2/6.	132,75	36,66	221,83	324,33	2,119	811,093
Si 2/7.	115,33	32,83	204,08	330,75	2,01	667,291
Si 2/8.	101,25	25,33	188,75	338,33	2,042	490,047
Si 2/9.	102,83	26,0	188,25	337,08	2,009	509,246
Si 3/1.	105,16	27,66	193,66	336,16	2,063	548,735
Si 1/1.	162,0	166,25	256,08	194,75	1,706	205,360
Si 1/3.	176,75	162,0	271,83	200,16	1,945	356,355
Si 1/4.	173,66	164,16	265,83	198,91	1,903	303,698
Si 1/5.(1.)	112,25	34,33	205,0	328,08	2,032	692,048
Si 1/5.(2.)	197,33	152,41	291,91	211,5	2,052	628,563
Si 1/6.	174,5	163,0	268,66	198,66	1,948	320,721

Tabelle 2.4 : Meßdaten des Bonner Ellipsometers für die Si-Proben. Aufgeführt werden die zwei Meßpaare von Polarisator- und Analysatorwinkel in Grad, Brechungsindex und Schichtdicke in Å, beide berechnet mit dem Programm XEllip.

Manche Proben, z.B. Si 2/1., sind mehrmals gemessen worden, wenn durch sichtbare Beschädigung der Oberfläche eine genaue Messung an der Maximumstelle nicht zu erwarten war. Andere Proben sind ganz aus der Messreihe herausgenommen worden, als eine versuchte Ellipsometermessung aufzeigte, daß aufgrund eines Fehlers bei der Capherstellung, kein Cap vorhanden war (z.B. GaAs 2/10.).

2.4.2.2 Kommerzielles Ellipsometer in Surrey

Mit dem kommerziellen Ellipsometer in Surrey wurden 13 dieser Proben noch einmal gemessen um die Messungen des Bonner Ellipsometers zu überprüfen. Bei dem Ellipsometer handelt es sich um ein Plasmos SD 6.21, das sogenannte Scanmessungen macht, d.h. es fährt einen eingebenen Bereich der Probe ab und führt die vorher angebene Anzahl von Messungen durch, z.B. wurde die Probe GaAs 2/6. auf einem Bereich von 3×4 mm mit 5 mal 5 Meßpunkten abgedeckt.
In der folgenden Tabelle sind die Ergebnisse einer dieser Scanmessungen, die der Probe GaAs 2/1, aufgeführt. Außerdem sind in der Tabelle noch die Größe des abgedeckten Bereichs in mm und die Anzahl der Messungen, die auf diesem Bereich gemacht worden sind, angegeben.

Name: GaAs 2/1.	Bereich: 3×4mm	Anzahl: 5×5
Nummer	d	n
1	290,33	2,0626
2	288,30	2,0603
3	284,20	2,0594
4	278,96	2,0527
5	274,81	2,0439
6	320,08	2,0648
7	322,68	2,0602
8	319,05	2,0608
9	311,95	2,0613
10	316,23	2,0229
11	344,07	2,0630
12	347,27	2,0610
13	348,73	2,0553
14	348,49	2,0483
15	361,65	2,0074
16	359,78	2,0478
17	361,62	2,0550
18	359,81	2,0617
19	359,02	2,0591
20	**371,12**	**2,0226**
21	352,94	2,0753
22	356,30	2,0742
23	355,19	2,0717
24	349,09	2,0695
25	343,21	2,0717

Tabellen 2.5 : Ellipsometermessungen, die in Surrey gemacht wurden an der Probe GaAs 2/1. Der maximale Wert wurde hervorgehoben, da ungefähr an dieser Stelle auch mit dem Bonner Ellipsometer gemessen wurde.

2.4.3 Beobachtungen

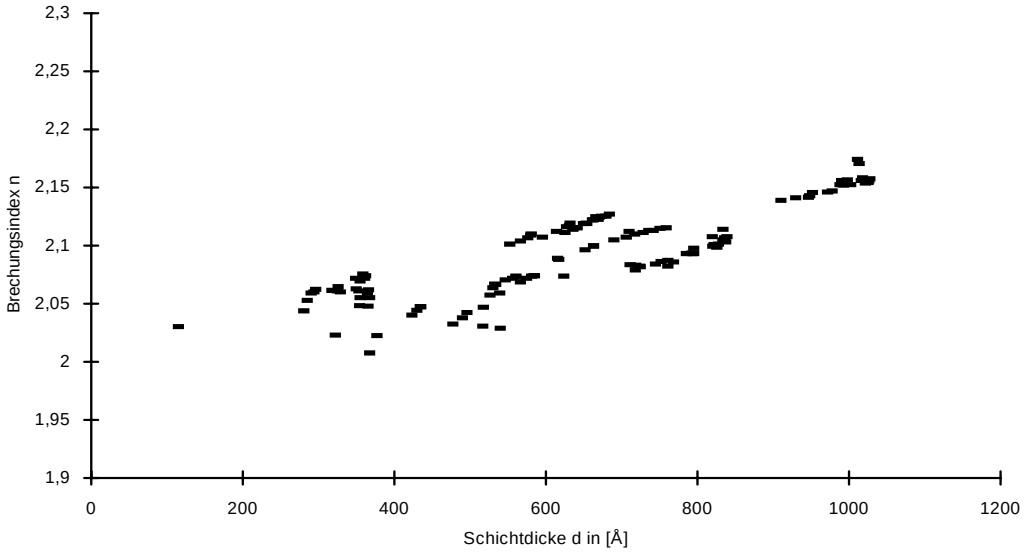
Die erste Beobachtung, die man machen kann, ist die, daß die Messungen des Bonner Ellipsometers mit denen aus Surrey sehr gut übereinstimmen (siehe Tabelle 2.18). Verglichen wurde immer der maximale Wert der Meßreihe, die durch das Surreyellipsometer an einer Probe durchgeführt wurde, mit dem in Bonn gemessenen Wert. Nur bei einer Probe (Si 2/4) wurde nicht der maximale, sondern der zweitgrößte Wert verglichen, da der gemessene maximale Wert wegen einer Beschädigung an der Probenoberfläche eindeutig falsch war. Die Schichtdicken in der Tabelle sind wieder in Å angegeben. Zusätzlich ist relative prozentuale Abweichung der beiden Werte in % angegeben.

Name	eigenes E.	Surrey-E.	Δ[%]
GaAs 2/1.	357,30	371,12	1,9
GaAs 2/4.	677,13	678,29	0,09
GaAs 2/5.	729,47	753,25	1,6
GaAs 2/6.	552,04	579,02	2,39
GaAs 2/9b.	1020,91	1021,41	0,02
GaAs 2/12.	821,55	833,28	0,71
Si 1/1.	205,36	203,50	0,45
Si 1/6.	320,72	338,99	2,77
Si 2/1.	766,87	775,12	0,54
Si 2/3.	491,79	489,37	0,25
Si 2/4.	272,31	121,66	-
Si 2/5.	260,88	280,54	3,63
Si 2/8.	490,05	496,61	0,66

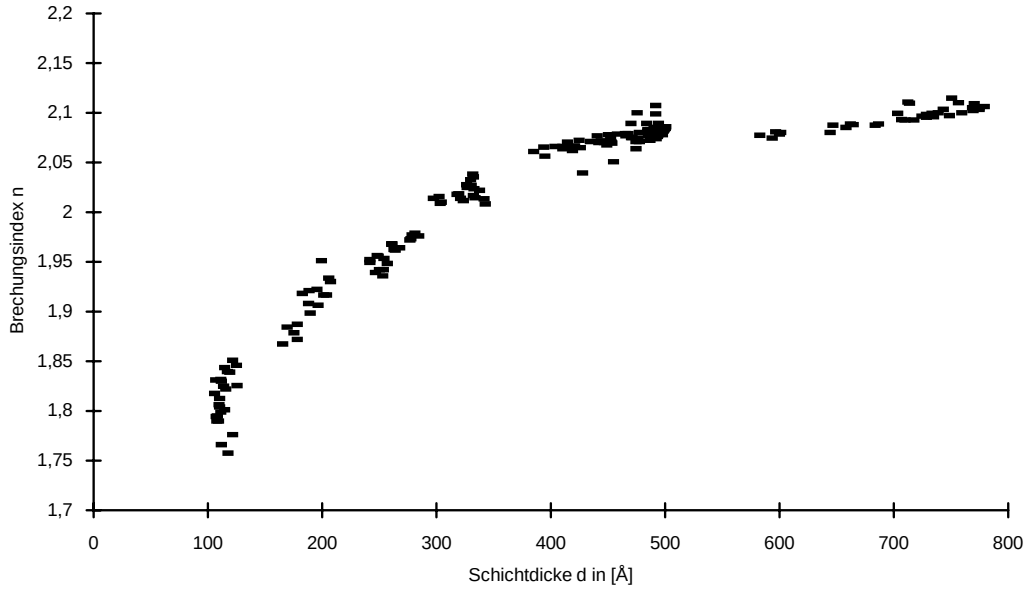
Tabelle 2.18 : Vergleich der Schichtdickenmessungen des Bonner Ellipsometers mit den Messungen, die in Surrey gemacht wurden. Die Angaben der Schichtdicke wurden in Å gemacht.

Bei der Probe Si 2/4 wurde auf die Berechnung der Abweichung, da offensichtlicherweise einer der beiden Messungen an einer beschädigten Stelle gemacht wurde. Die Abweichungen, die im Durchschnitt etwas mehr als 1% betragen, sind darauf zurückzuführen, daß die Messungen nicht ganz genau an den gleichen Stellen gemacht wurden. Eine Verschiebung der Meßstelle um etwa 0,5mm in beliebiger Richtung kann auf einen Unterschied von bis zu 50Å in der Messung führen.
Die zweite Beobachtung, die man macht, ist, daß der Brechungsindex von der Schichtdicke abhängig ist. Wenn man alle Werte der Surrey-Messungen (weil diese in größerer Zahl vorhanden sind) in ein Diagramm einträgt, in dem

Brechungsindex in Abhängigkeit von der Schichtdicke dargestellt wird, erhält man die Graphen 2.1 und 2.2 für die Galliumarsenid- und Silizium-Proben. Die verschiedenen Gruppen von Punkten in den Kurven gehören zu den einzelnen Proben. Es ist ein deutlicher Anstieg des Brechungsindex mit der Schichtdicke zu erkennen. Speziell bei Silizium ist zuerst ein scharfer Anstieg bis zu einer Dicke von etwa 350 Å, danach wird die Kurve flacher. Ein Grund für den Anstieg könnte sein, daß die Siliziumnitridschichten, die aufgedampft werden, keine reinen Si_3N_4-Schichten sind, sondern aus einer Mischung aus Si und Si_3N_4 bestehen.



Graph 2.1 : Brechungsindex in Abhängigkeit der Schichtdicke (n(d)-Diagramm) für die GaAs-Proben. Die verschiedenen Gruppen von Punkten stammen von den verschiedenen Proben. Es ist ein deutlicher Anstieg im Brechungsindex mit der Dicke zusehen.



Graph 2.2 : Brechungsindex in Abhängigkeit der Schichtdicke (n(d)-Diagramm) für die Si-Proben. Die verschiedenen Gruppen von Punkten stammen von den verschiedenen Proben. Es ist ein deutlicher Anstieg im Brechungsindex mit der Dicke zusehen.

In mehreren Artikeln ([ZHA92], [DEH95], [GRI96]) wird erwähnt, daß Si_3N_4-Schichten immer zusätzliches Silizium enthalten. Es ist nur nicht sicher in welcher Form, denn es gibt mehrere Möglichkeiten, die in Frage kämen. Die erste ist, daß man amorphes Silizium mit gleichmäßig verteiltem Stickstoff vorliegen hat. Die anderen beiden Möglichkeiten bestehen aus einer Mischung von Si_3N_4 und amorphen Silizium, wobei die eine überwiegend aus Si_3N_4 mit einzelnen eingebauten Si-Inseln besteht und die andere aus Silizium mit Si_3N_4-Inseln aufgebaut ist. Die erste und die letzte Möglichkeit kann man dadurch ausschließen, daß der Brechungsindex unserer Schichten immer noch in der Nähe des Brechungsindex von Si_3N_4 (n = 2,04) liegt. Der (reelle) Brechungsindex von Silizium liegt bei etwa n = 4. Es liegen also wahrscheinlich Schichten vor, die aus Si_3N_4 mit eingebautem Silizium bestehen. Den Anstieg im Brechungsindex kann man dann dadurch erklären, daß diese „Inseln“ aus Silizium mit steigender Schichtdicke anwachsen und so den Brechungsindex erhöhen. Der Wert für den Brechungsindex wird aber wahrscheinlich nie sehr weit von seinem „Sollwert“ abweichen, da wir immer ein Übergewicht von Si_3N_4 gegenüber von Silizium in der Schicht haben werden. Speziell im Fall von Si_3N_4 auf Siliziumsubstrat kann man noch eine weitere Vermutung anstellen. Der gemessene Brechungsindex der Schicht beginnt bei ca. 1,75. Dies kann man folgendermaßen erklären: Der Brechungsindex von Siliziumoxid liegt bei 1,46. Es ist bekannt [Men96], daß der Brechungsindex sich kontinuierlich von diesem Wert auf 2,04 ändert, wenn die Schicht zuerst aus reinem Siliziumoxid besteht und dann über Siliziumoxynitridverbindungen auf eine reine Siliziumnitridschicht geändert wird. Es ist außerdem bekannt, daß sich auf dem Siliziumwafer eine Oxidschicht bildet. Diese wird nach und nach von der Siliziumnitridschicht überdeckt, aber bei sehr dünnen Schichten wird sie vom Ellipsometer über den Brechungsindex noch detektiert. Ab einer Dicke von ca. 350 Å ist dann diese Schicht überdeckt und die Kurve flacht ab.

Um herauszufinden, ob sich die Theorie des überschüssigen Siliziums in der Schicht bestätigt, wurden an einigen Proben RBS-Messungen durchgeführt. Diese werden in Kapitel 3 ausführlich beschrieben. Die Vermutung, daß sich eine Siliziumoxidschicht an der Oberfläche des Siliziumwafers gebildet haben könnte, kann man mit RBS-Messungen leider nicht nachprüfen, da die Konzentration von Sauerstoff in dieser vermutlich sehr dünnen Siliziumoxidschicht zu gering ist, um detektiert zu werden.

3 Rutherford Rückstreuung (RBS)

3.1 Einführung in die RBS

Die Rutherford-Rückstreuung (**R**utherford **B**acks**c**attering) beruht auf dem Prinzip der elastischen Streuung von Projektilen an Targetkernen. Diese zerstörungsfreie Methode zur Untersuchung von Festkörpern macht sich die kinematischen Verhältnisse zunutze, wie sie auch beim Billiardspielen auftreten. Die Energie des gestreuten Projektils ist abhängig von der Masse des streuenden Kerns. Dies erlaubt eine Unterscheidung der Atome in der untersuchten Probe.

3.1.1 Kinematischer Faktor

Wenn zwei Atomkerne zusammenstoßen, die nicht so große Energie haben, daß Kernkräfte auftreten, wechselwirken diese über die Coulomb-Kraft. Die Verhältnisse beim Stoß sind in Abbildung 3.1 dargestellt.

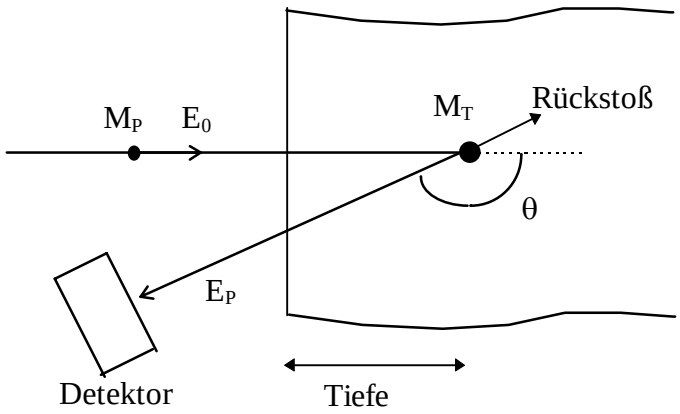


Abbildung 3.1: Schematische Anordnung bei der Rutherford-Rückstreung. Eingezeichnet sind: M_P : Projektil, M_T : Targetatom, θ : Rückstreuwinkel, E_0 : Anfangsenergie des Projektils, E_P : Energie des zurückgestreuten Projektils. Der Index P steht für Projektil, T für Target.

Das Verhältnis der Projektilenergie nach dem Stoß E_P zur Energie unmittelbar vor dem Stoß E_0 wird kinematischer Faktor genannt.

$$K := \frac{E_P}{E_0} \tag{3.1}$$

Aus den Energie- und Impulserhaltungssätzen für den elastischen Stoß läßt sich dieser Faktor leicht berechnen:

$$K = \left[\frac{\sqrt{1 - \left[\left(M_P \,/\, M_T \right) \sin\theta \right]^2} + \left(M_P \,/\, M_T \right) \cos\theta}{1 + \left(M_P \,/\, M_T \right)} \right]^2 \tag{3.2}$$

Wie man sieht, ist der kinematische Faktor nur abhängig vom Verhältnis der beteiligten Massen $M_P \,/\, M_T$ und vom Streuwinkel θ. Sind der Streuwinkel und das Projektil bekannt, kann man aus der Energie des rückgestreuten Projektils feststellen, welches Targetatom das Projektil getroffen hat, da der kinematische Faktor für jedes Element spezifisch ist.
Um den Energieverlust des Projektils optimal bestimmen zu können, soll *K* bei einem gegebenem $M_P \,/\, M_T$ Verhältnis und fester Einschußenergie E_0 möglichst klein sein. Dies ist der Fall, wenn Rückstreuung vorliegt, also $\theta = 180°$ ist, weil das Projektil dann die meiste Energie an das Targetatom überträgt. Für *K* gilt dann:

$$K(\theta = 180°) = \left[\frac{1 - \left(M_P \,/\, M_T \right)}{1 + \left(M_P \,/\, M_T \right)} \right]^2 \tag{3.3}$$

Man muß allerdings die Bedingung stellen, daß $M_P < M_T$, da nur dann eine Rückstreuung stattfinden kann.
Die vorzuziehende Detektorstellung ist also bei $\theta = 180°$ gegeben. Hiervon leitet sich auch der Name Rutherford-Rückstreuung ab. In der Praxis läßt sich $\theta = 180°$ natürlich nicht realisieren, da der einfallende Teilchenstrahl nicht behindert werden darf. Mit modernen Ringdetektoren, durch dessen Zentrum der Strahl hindurchgeht, ist es heute aber möglich, sehr dicht an die „perfekten 180°“ heranzukommen.

3.1.2 Wirkungsquerschnitt für die Rutherford-Streuung

Wenn man die RBS-Methode anwendet, ist es wichtig, die Wahrscheinlichkeit des Streuprozesses für die einzelnen Atomsorten zu kennen. Diese Wahrscheinlichkeit *W* für die Reaktion eines Streuprozesses ist durch den Wirkungquerschnitt σ und die Anzahl der Streuzentren N pro bestrahlter Targetfläche (Strahldurchmesser) *A* bestimmt.

$$W = \frac{N}{A}\sigma \tag{3.4}$$

Man kann sich die Größe σ anschaulich vorstellen, wenn man Abbildung 3.2 zu Hilfe nimmt. Man nimmt an, daß die Teilchen, die in den Bereich der Fläche σ kommen, gestreut werden, die anderen jedoch ungehindert und vor allem unabgelenkt hindurchfliegen. Die Größe σ gibt dann die (gedachte) Fläche an, innerhalb derer eine Streuung stattfindet. Diese Vorstellung und Gleichung (3.4) gelten aber nur dann, wenn die Targetzone sehr dünn ist, sich die Trefferflächen also nicht überlappen. Die Anzahl der Streuereignisse pro Zeit läßt sich damit aus

$$I = I_0 \cdot W = \frac{I_0 N}{A}\sigma \tag{3.5}$$

berechnen. I_0 steht hier für die Anzahl der einfallenden Teilchen pro Zeiteinheit.

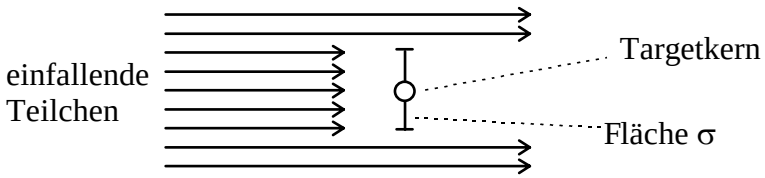


Abbildung 3.2 : Anschauliche Darstellung des Wirkungsquerschnitts. Die Teilchen, die in den Bereich der Fläche σ treffen, erfahren eine Wechselwirkung (Streuung). Die anderen fliegen ungehindert - und ohne ihre Richtung zu verändern - hindurch.

Als den differentiellen Wirkungsquerschnitt $d\sigma / d\Omega$ definiert man analog zu Gleichung (3.5):

$$\frac{d\sigma}{d\Omega} = \frac{A}{I_0 N}\frac{dI}{d\Omega} \tag{3.6}$$

Diese Größe ist ein Maß für die Anzahl der Streuereignisse pro Zeiteinheit, die im Raumwinkelelement $d\Omega$ beobachtet werden.
Bei der Rutherford-Streuung rechnet man mit einer reinen Coulomb-Streuung der beteiligten Atomkerne. Abweichungen treten nur auf, wenn die Energie der Teilchen so groß ist, daß die Coulomb-Barriere der Kerne überwunden wird und Kernkräfte auftreten. Dieser Fall soll hier aber nicht weiter diskutiert werden.

Im Falle der reinen Coulomb-Wechselwirkung lautet der differentielle Wirkungsquerschnitt für Rutherford-Streuung im Laborsystem:

$$\frac{d\sigma}{d\Omega} = \left(\frac{Z_P Z_T e^2}{16\pi\varepsilon_0 E}\right)^2 \frac{4}{\sin^4\theta} \frac{\left[\sqrt{1-\left[\left(M_P / M_T\right)\sin\theta\right]^2} + \cos\theta\right]^2}{\sqrt{1-\left[\left(M_P / M_T\right)\sin\theta\right]^2}} \qquad (3.7)$$

Z_P und Z_T sind hier die Kernladungszahlen des Projektils und des Targetatoms. Für die Näherung, daß $M_P << M_T$ (das Laborsystem ist dann identisch mit dem Schwerpunktsystem) erhalten wir die sogenannte Rutherford-Formel:

$$\frac{d\sigma}{d\Omega} = \left(\frac{Z_P Z_T e^2}{16\pi\varepsilon_0 E}\right)^2 \frac{1}{\sin^4(\theta/2)} \qquad (3.8)$$

Aus dieser Formel kann man zwei wichtige Tatsachen herauslesen. Erstens ist der Wirkungquerschnitt proportional zu $\left(Z_P Z_T\right)^2$, d.h. es ist immer günstiger, wenn die Targetatome bereits bekannt sind, Projektile mit höherer Kernladungszahl zu wählen, z.B. ^{4}He statt ^{1}H. Die zweite Tatsache ist, daß der Wirkungsquerschnitt mit E^{-2} anwächst. Das bedeutet, daß die Streurate bei höheren Einschußenergien sehr nachläßt.

3.1.3 Energieverlust in Materie

Geladene Teilchen erfahren beim Durchgang durch Materie einen Energieverlust. Wenn die Energien nicht zu klein sind ist der wichtigste Prozeß das sogenannte elektronische Bremsen, die Streuung an Elektronen der Targetatome. Hierbei werden die Atome angeregt oder sogar ionisiert. Sehr viel seltener tritt die Rutherford-Streuung an den Targetatomkernen auf. Beim Energieverlust durch den Prozeß der Vielfachstreuung entsteht auch ein Energie- und Winkelauffächerung (engl.: energy straggling, angular straggling) des Strahls.
Den Energieverlust pro Weglänge bezeichnet man als Bremsvermögen (engl.: stopping power).

$$-\frac{dE}{dx} := -\lim_{\Delta x \to 0} \frac{\Delta E}{\Delta x} \qquad (3.9)$$

Von Bethe und Bloch stammt ein Zusammenhang mit dem man das elektronische Bremsen berechnen kann:

$$-\frac{dE}{dx} = n\frac{Z_P^2 Z_T e^4}{4\pi\varepsilon_0^2 v^2 m_e}\left[\ln\frac{2m_e v^2}{\langle I\rangle} - \ln\left(1-\frac{v^2}{c^2}\right) - \frac{v^2}{c^2}\right] \qquad (3.10)$$

Dabei ist Z_P die Kernladungszahl, v die Geschwindigkeit der einfallenden Teilchen und n die Anzahldichte der Targetatome mit der Kernladungszahl Z_T. Die Größe $\langle I\rangle$ beschreibt das mittlere Ionisationspotential der Elektronen, das man mit $\langle I\rangle = 11{,}5\cdot Z_T(eV)$ abschätzen kann.
Da bei den üblichen Ionenstrahltechniken die Projektilgeschwindigkeiten sehr viel geringer sind als die Lichtgeschwindigkeit ($v<<c$) kann man die letzten beiden Terme in Gleichung (3.10) vernachlässigen. Für kleine Energien gewinnt der erste logarithmische Term an Bedeutung. Bei kleineren Energien gibt es einen weiteren Prozeß, der zum Energieverlust beiträgt, nämlich die Wechselwirkung des Projektil mit schwach gebundenen Elektronen. Bei diesen (kleineren) Energien tritt auch das nukleare Bremsen verstärkt auf. Diese beiden Effekte führen zu Abweichungen in der Bethe-Bloch-Formel.
Die Diskussion bezog sich bisher auf Substanzen, die aus einem einzigen Element bestehen. Für Targets, die aus mehreren Elementen zusammengesetzt sind, gibt es eine Näherungsformel, die eine Additivität des auf die Atomdichte normierten Bremsvermögens zugrunde legt. Dies ist die sogenannte Bragg-Kleeman-Regel. Sind die Elemente mit *A* und *B* bezeichnet und die relativen Anteile in der Substanz α und β, dann gilt für das Bremsvermögen des zusammengesetzten Materials

$$\frac{dE}{dx}\left(A_\alpha B_\beta\right) = n\left(A_\alpha B_\beta\right)\left[\frac{\alpha}{n(A)}\frac{dE}{dx}(A) + \frac{\beta}{n(B)}\frac{dE}{dx}(B)\right] \qquad (3.11)$$

$n(A)$ und $n(B)$ sind die jeweiligen Anzahldichten der Targetatome.
Im Tabellenwerk von J.Ziegler [Zie77] ist das Bremsvermögen für verschiedene Projektil-Target-Kombinationen aufgeführt.

3.1.4 RBS-Messungen an dünnen Schichten

Die Untersuchung von dünnen Schichten ist ein sehr wichtiges Anwendungsgebiet der Rutherford-Rückstreuung.
Für diese Art von Experimenten muß man sich den Zusammenhang zwischen der gemessenen Energie der gestreuten Projektile und der Tiefe, in der die Streuung stattgefunden hat, klarmachen. In Abbildung 3.3 ist dieser Sachverhalt schematisch dargestellt.
Wenn man die Energie E_p des Projektils nach einer Streuung in der Tiefe *x* berechnen will, ist es nützlich, eine Näherung zu machen: Das Bremsvermögen dE/dx soll für den Hinweg $\left(l_1 = x/\sin\alpha_1\right)$ des Projektils zum Streuzentrum und für den Rückweg $\left(l_2 = x/\sin\alpha_2\right)$ zur Oberfläche als konstant angenommen werden. Diese Näherung ist für kurze Wege, also kleine Schichtdicken, sehr gut erfüllt, da das Bremsvermögen nur wenig mit der Energie variiert. Für die Energie E_x des Teilchens kurz vor der Streuung gilt dann:

$$E_x = E_0 - \left.\frac{dE}{dx}\right|_{E_0} \frac{x}{\sin\alpha_1} \qquad (3.12)$$

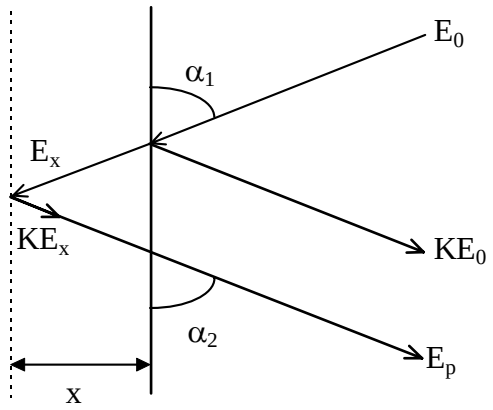


Abbildung 3.3 : Energieverhältnisse bei der Streuung eines Projektilteilchens an der Oberfläche und in einer Tiefe x. Ein- und ausfallender Strahl schließen die Winkel α_1 und α_2 zur Probenoberfläche ein.

d.h. ursprüngliche Energie E_0 minus dem Energieverlust durch die Schicht. Das gleiche gilt für den Rückweg, nur daß hier die ursprüngliche Energie KE_x beträgt:

$$E_P = KE_x - \left.\frac{dE}{dx}\right|_{KE_x} \frac{x}{\sin\alpha_2} \qquad (3.13)$$

Mit Hilfe dieser beiden Gleichungen ist es möglich den Endergieunterschied ΔE zwischen der Streuung an der Oberfläche und der Streuung in der Tiefe x anzugeben.

$$\Delta E = KE_0 - E_P = \left[\frac{K}{\sin\alpha_1} \left.\frac{dE}{dx}\right|_{E_0} + \frac{1}{\sin\alpha_2} \left.\frac{dE}{dx}\right|_{KE_x} \right] \cdot x \qquad (3.14)$$

Bei einem unbekanntem x kann man aber E_x, das zur Berechnung des Bremsvermögens nötig ist, nicht angeben. Um dieses Problem zu lösen, macht man sich wieder die relativ schwache Abhängigkeit des Bremsvermögens von der Energie zunutze und setzt

$$\left.\frac{dE}{dx}\right|_{KE_x} = \left.\frac{dE}{dx}\right|_{KE_0} \tag{3.15}$$

Damit erhält man

$$\Delta E = \left[\frac{K}{\sin\alpha_1} \left.\frac{dE}{dx}\right|_{E_0} + \frac{1}{\sin\alpha_2} \left.\frac{dE}{dx}\right|_{KE_0} \right] \cdot x \tag{3.16}$$

Es ergibt sich ein linearer Zusammenhang zwischen der Rückstreutiefe und der Rückstreuenergie.

Abbildung 3.4 : Ideales Rückstreuspektrum einer zweielementigen Schicht auf einem leichteren Substrat. (aus [MOD04])

Ein Beispiel eines idealen RBS-Spektrums einer dünnen zweielementigen Schicht auf einem leichteren Substrat ist in Abbildung 3.4 zu sehen. Die Kanalnummer ist direkt proportional zur Rückstreuenergie. Da die Rückstreuenergie aber um so größer ist, je schwerer das streuende Element war, ist die Kanalnummer ebenso proportional zur Masse der streuenden Atoms, d.h. je schwerer das Element desto weiter rechts steht es im Spektrum. So kann man in diesem Beispiel erkennen, daß die zwei Elemente A und B der Schicht beide schwerer sind als das Substrat. Die rechte Seite der abgeflachten Peaks entsprechen den Projektilen, die an der Oberfläche der Schicht gestreut wurden und haben die Energie $K_A E_0 = E_1^A$ bzw. $K_B E_0 = E_1^B$. Die linke Seite der Peaks entspricht den Projektilen, die an der Rückseite der Schicht gestreut wurden. Diese haben zusätzliche Energie verloren, weil sie die Schicht noch zweimal (hin und zurück) durchqueren mußten. Die Breite der abgeflachten Peaks ist also ein Maß für die Schichtdicke. Mit dem Energieunterschied der Vorder- und Rückseite läßt sich die Schichtdicke ausrechnen. Aus der Höhe der Peaks läßt sich der relative Anteil der jeweiligen Elemente und damit die Stöchiometrie der Schicht bestimmen. Je mehr von einem

Element in der Schicht vorhanden ist desto größer ist die Wahrscheinlichkeit des Projektils an diesem Element zu streuen und desto größer werden die Zählraten in den Kanälen, die den Energien dieses Elements zugeordnet sind. Der leichte Anstieg zu den niedrigen Energien hin liegt an der Tatsache begründet, daß der Wirkungsquerschnitt für die Rutherford-Streuung mit sinkender Teilchenenergie zunimmt.
Das Substrat taucht in den Spektren als flaches Band auf, das bis zu den niedrigen Energien reicht, denn das Substrat ist zu dick als daß die an der Rückseite gestreuten Projektile noch zur Oberfläche zurück und in den Detektor gelangen könnten. Der Anstieg der Zählrate bei niedrigen Energien, der in allen RBS-Spektren vorhanden ist, hat seinen Grund in der sogenannten Vielfachstreuung (engl.: multiscattering). In diesem Fall wurden die Projektile nicht nur einmal gestreut sondern mehrfach, sie verlieren also mehr Energie und führen so zu einem Anstieg bei Zählraten der niedrigen Energien.
Zu Abweichungen von dieser idealen Form kann es durch sogenannte Channelling-Effekte kommen. Mit Channelling oder auch Gitterführung bezeichnet man im allgemeinen den Fall, daß der einkommende Strahl durch das Gitter des Substrats „geführt“ wird, keine Streuung erfährt und so für das Rückstreuspektrum verloren geht. In diesem Fall sind die Zählraten des gemessenen Spektrums geringer als man sie erwarten würde. Ein gegenteiliger Effekt mit fast gleicher Ursache führt zu überhöhten Zählraten im Spektrum. Weicht der Strahl nämlich wenig von einer gitterführenden Richtung ab, so führt dies zu einer verstärkten Streuung an den „Tunnelwänden“ und die Zählraten steigen an. Der sogenannte „pile-up“-Effekt kann ebenfalls zu höheren Zählraten beitragen. Dabei handelt es sich um mehrere Projektile geringer Energie, die fast gleichzeitig im Detektor ankommen und als ein einziges hochenergetisches Teilchen registriert werden.

3.2 Messungen

Die Rückstreuexperimente wurden am ITN (Instituto Tecnológico e Nuclear) in Sacavem in der Nähe von Lissabon durchgeführt.

Die Messungen wurden mit $^{4}He^{+}$ bei 2MeV Energie unter unterschiedlichen Winkeln durchgeführt. Es gibt dort zwei Kammern mit insgesamt drei Silizium-Oberflächenschicht-Detektoren. Da die Kammern hintereinander aufgebaut sind, nutzen beide Kammern den sogenannten „annular"- Detektor. Dieser ist als Ring geformt und um den einfallenden Strahl gebaut, so daß die rückgestreuten Teilchen unter einem Winkel von fast 180° detektiert werden. In der ersten Kammer, die „small chamber", ist ein Detektor in einer Cornellgeometrie eingebaut, der die Teilchen detektiert, die im Winkel von 140° zurückgestreut werden. In der „universal chamber" detektiert der dritte Detektor die unter 160° zurückgestreuten Teilchen. Dieser steht in einer sogenannten IBM-Geometrie. Bei dieser Geometrie liegen einfallender Strahl, Probennormale und detektierter Strahl in einer Ebene, in diesem Fall waagerecht. Im Gegensatz dazu bilden bei der Cornell-Geometrie einfallender Strahl und detektierter Strahl eine senkrechte Ebene, die von der waagerechten Ebene, die vom einfallenden Strahl und Probennormale gebildet wird, senkrecht geschnitten wird.

Die Detektoren werden im folgenden mit annular-Detektor (180°), standard-Detektor (160°) und 140°-Detektor (140°) bezeichnet.

3.3 Spektren und Simulationen

Die genauen Spezifikationen der einzelnen Messungen sind bei den einzelnen Spektren im Kapitel 3.3 mit angegeben. Die Vorgehensweise war bei allen Spektren gleich. Zuerst wurde eine Energieeichung durchgeführt mit Hilfe der rechten Kanten der Silizium- und Stickstoffpeaks, um die Kanalnummern ihrer Energie zuzuordnen. Dies ist bei den Spektren mit „conv“ (für engl.: conversion = Umwandlung) angegeben: der erste Wert gibt die Energie pro Kanal an, der zweite ist die Energie im Kanal 0. Danach wurden die Details der Messung angegeben: Energie, Ladung (engl.: current), die Winkel θ und φ, und die Werte für die Detektoren : die Halbwertsbreite FWHM (Full Width Half Maximum) in keV und der Raumwinkel Ω in msr (Millisteradiant). Für den annular-Detektor liegen diese Werte bei FWHM = 15 keV, Omega = 18 msr, für den standard-Detektor bei FWHM = 12 keV, Omega = 1,45 msr, für den 140°-Detektor bei FWHM = 13 keV, Omega = 3,4 msr. Der Winkel θ steht für den Winkel zwischen einfallendem Strahl und der Flächennormalen der Probe. Der Winkel φ ist der Streuwinkel (der Winkel zwischen einfallendem und detektierten Strahl). Die Bestimmung der Gesamtladung ist der unsicherste Parameter. Deswegen geht man so vor, daß die Ladung (engl.: charge) solange variiert wird, bis das Niveau der simulierten mit den gemessenen Spektren übereinstimmt. Simuliert wurden die Spektren mit Hilfe des Programms RUMP. Die Simulation besteht immer aus zwei Schichten (engl.: layers), von denen nur die erste, die Angaben für die Siliziumnitridschicht, im Spektrum mit aufgeführt wurde. Die zweite beinhaltet die Angaben für das Substrat. Für jede Schicht muß eine Dicke angegeben werden in der Einheit */cm2*, die für die natürliche Einheit $10^{15} at / cm^2$ als Kürzel eingesetzt wird. Diese Einheit ist eine sogenannte Flächendichte, die unabhängig vom untersuchten Material ist. Die Dichte wird später mit berücksichtigt (siehe unten). Danach werden die Elemente, die in der Schicht vorhanden sein sollen, in relativen Einheiten angegeben. Für Si_3N_4 schreibt man dann : si 3 n 4; für GaAs : ga 1 as 1.
Zu jeder Probe gibt es zwei Spektren (eine für den annular-Detektor und eine für den standard-Detektor oder 140°-Detektor, je nachdem in welcher Kammer gemessen wurde) mit jeweils zwei Simulationen. Die erste Simulation ist immer eine reine Siliziumnitrid-Schicht (Si_3N_4), die zweite ist eine Simulation mit überschüssigem Silizium zum Vergleich. Bei den GaAs-Proben sind nur die reinen Siliziumnitrid-Schichten simuliert, da diese Simulationen schon ausreichend waren.
Als zweite Auswertung wurden die Energien, wenn möglich, der Vorder- und Rückseite der Silizium- und Stickstoff-Peaks aus den Spektren abgelesen, um eine Schichtdickenberechnung durchzuführen. Dazu ist es nötig, den Bremswirkungsquerschnitt und das Bremsvermögen zu berechnen. Da diese von der Energie, der Projektil- und Targetart abhängig sind, werden sie hier für $^4He^+$ mit der Energie 2MeV in Si_3N_4 berechnet.
Das Bremsvermögen von Si_3N_4 mit der Einfallsenergie 2MeV wird nach der Bragg-Kleeman-Regel mit der folgenden Formel berechnet:

$$\varepsilon_{in}^{Si_3N_4} = \left(3 \cdot \varepsilon_{in}^{Si} + 4 \cdot \varepsilon_{in}^{N}\right)$$

wobei ε_{in}^{Si} und ε_{in}^{N} durch Näherungsformeln aus dem Tabellenwerk von Ziegler [ZIE77] gegeben sind (Anhang 5.2).

$$\varepsilon_{in}^{Si} = 47{,}9 \cdot 10^{-15}\, eVcm^2$$

$$\varepsilon_{in}^{N} = 30{,}9 \cdot 10^{-15}\, eVcm^2$$

$$\Rightarrow \qquad \varepsilon_{in}^{Si_3N_4} = 267{,}3 \cdot 10^{-15}\, eVcm^2$$

Jetzt braucht man noch die Werte für die Energien, nachdem das Projektil an entweder Silizium $(K(Si) = 0{,}5633)$ oder Stickstoff $(K(N) = 0{,}3806)$ gestreut worden ist, d.h. das Projektil hat entweder die Energie $K(Si) \cdot E_0 = 1126{,}6keV$ oder $K(N) \cdot E_0 = 617{,}2keV$.

$$\varepsilon_{out,Si}^{Si} = 60{,}7 \cdot 10^{-15}\, eVcm^2$$

$$\varepsilon_{out,Si}^{N} = 39{,}9 \cdot 10^{-15}\, eVcm^2$$

$$\Rightarrow \varepsilon_{out,Si}^{Si_3N_4} = \left(3 \cdot \varepsilon_{out,Si}^{Si} + 4 \cdot \varepsilon_{out,Si}^{N}\right) = 341{,}7 \cdot 10^{-15}\, eVcm^2$$

und

$$\varepsilon_{out,N}^{Si} = 71{,}3 \cdot 10^{-15}\, eVcm^2$$

$$\varepsilon_{out,N}^{N} = 41{,}8 \cdot 10^{-15}\, eVcm^2$$

$$\Rightarrow \varepsilon_{out,N}^{Si_3N_4} = \left(3 \cdot \varepsilon_{out,N}^{Si} + 4 \cdot \varepsilon_{out,N}^{N}\right) = 381{,}1 \cdot 10^{-15}\, eVcm^2$$

Mit diesen Werten kann man dann den Bremswirkungsquerschnitt ausrechnen:

$$\left[\varepsilon_0\right]_{Si}^{Si_3N_4} = K(Si) \cdot \varepsilon_{in}^{Si_3N_4} \cdot \frac{1}{\cos\alpha_1} + \varepsilon_{out,Si}^{Si_3N_4} \cdot \frac{1}{\cos\alpha_2}$$

$$= 150{,}7 \cdot \frac{1}{\cos\alpha_1} + 341{,}7 \frac{1}{\cos\alpha_2} \qquad \text{(a)}$$

$$\left[\varepsilon_0\right]_{N}^{Si_3N_4} = K(N) \cdot \varepsilon_{in}^{Si_3N_4} \cdot \frac{1}{\cos\alpha_1} + \varepsilon_{out,N}^{Si_3N_4} \cdot \frac{1}{\cos\alpha_2}$$

$$= 82{,}5 \cdot \frac{1}{\cos\alpha_1} + 381{,}1 \frac{1}{\cos\alpha_2} \qquad \text{(b)}$$

Diese Werte gelten allerdings nur bei einem Winkel von 180°, denn wir begannen die Rechnung mit den kinematischen Faktoren $K(Si)$ und $K(N)$ und die sind, wie in Kapitel 3.1 erwähnt wird, winkelabhängig. Die Werte für die Streuwinkel 140° und 160° sind zusammen mit denen von 180° in der Tabelle 3.1 angegeben.

	180°	160°	140°
K(Si)	0,5633	0,5731	0,6022
K(N)	0,3086	0,3194	0,3531
$K(Si)\cdot E_0$ in keV	1126,6	1146,1	1204,4
$K(N)\cdot E_0$ in keV	617,2	638,8	706,2
$\varepsilon_{out,Si}^{Si}$	60,7	60,3	59,2
$\varepsilon_{out,Si}^{N}$	39,9	39,7	39,1
$\Rightarrow \varepsilon_{out,Si}^{Si_3N_4}$	341,7	339,7	334,0
$\varepsilon_{out,N}^{Si}$	71,3	70,9	69,6
$\varepsilon_{out,N}^{N}$	41,8	41,9	42,1
$\Rightarrow \varepsilon_{out,N}^{Si_3N_4}$	381,1	380,3	377,2

Tabelle 3.1 : Die kinematischen Faktoren, die Energie unmittelbar nach dem Stoß in keV und das berechnete Bremsvermögen für die Winkel 140°, 160° und 180° in $10^{-15} eVcm^2$.

Zu den Formeln (a) und (b) muß man noch eine weitere Bemerkung machen. Die angegebenen Winkel α_1 und α_2 sind die absoluten Winkel von ein- und ausfallendem Strahl gegenüber der Probennormalen. In unserem Fall muß man die verschiedenen Geometrien der Detektoren mit einbeziehen. Dann erhält man die folgenden Terme:

für 180°:

$$\left[\varepsilon_0\right]_{Si}^{Si_3N_4} = 150{,}6 \cdot \frac{1}{\cos\theta} + 341{,}7 \frac{1}{\cos\theta}$$

$$\left[\varepsilon_0\right]_{N}^{Si_3N_4} = 82{,}5 \cdot \frac{1}{\cos\theta} + 381{,}1 \frac{1}{\cos\theta}$$

für 160°:

$$\left[\varepsilon_0\right]_{Si}^{Si_3N_4} = 153{,}2 \cdot \frac{1}{\cos\theta} + 339{,}7 \frac{1}{\cos 20°} \frac{1}{\cos\theta}$$

$$\left[\varepsilon_0\right]_{N}^{Si_3N_4} = 85{,}4 \cdot \frac{1}{\cos\theta} + 380{,}3 \frac{1}{\cos 20°} \frac{1}{\cos\theta}$$

für 140°:

$$\left[\varepsilon_0\right]_{Si}^{Si_3N_4} = 161{,}0 \cdot \frac{1}{\cos\theta} + 334{,}0 \frac{1}{\cos(40° - \theta)}$$

$$\left[\varepsilon_0\right]_{N}^{Si_3N_4} = 94{,}4 \cdot \frac{1}{\cos\theta} + 377{,}2 \frac{1}{\cos(40° - \theta)}$$

wobei θ den Winkel beschreibt, um den Probe gedreht wurde und der in den Spektren ebenfalls als θ angegeben ist.
Wenn man nun noch die molekulare Dichte von Si_3N_4 ausrechnet

$$N^{Si_3N_4} = \frac{3{.}1\, g/cm^3 \cdot N_A}{140\, g/mol} = 1{,}329 \cdot 10^{22} \frac{Si_3N_4}{cm^3}$$

ist es mit der folgenden Formel möglich, die Schichtdicke auszurechnen.

$$x = \frac{\Delta E_{Si/N}}{\left[\varepsilon_0\right]_{Si/N}^{Si_3N_4} \cdot N^{Si_3N_4}}$$

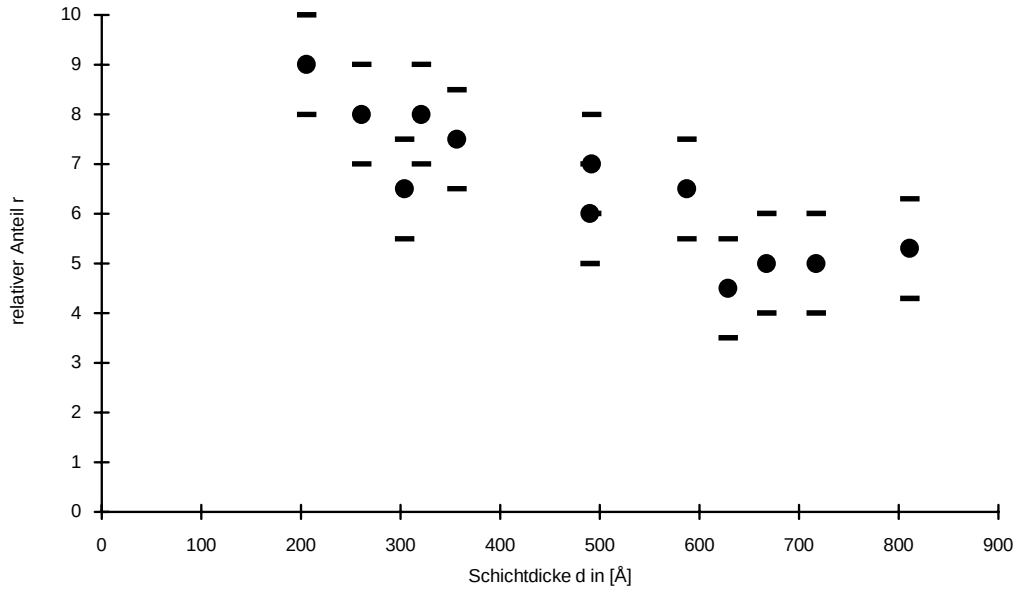
Diese berechnete Schichtdicke muß allerdings mit einer gewissen Skepsis betrachtet werden. Die Gründe hierfür sind in Kapitel 3.4 am Schluß erwähnt.

Auf den folgenden Seiten sind die gemessenen Spektren mit den zugehörigen Simulationen zu sehen.

3.4 Beobachtungen

3.4.1 Diskussion des relativen Anteils von Silizium in der Schicht

Bei allen Spektren der Proben, die Silizium als Substrat haben, ist zu sehen, daß der gemessene relative Anteil von Stickstoff zu hoch ist, wenn die Simulation nur die ursprünglich erwarteten Anteile Si : N = 3 : 4 ansetzt. Wird der relative Anteil von Silizium erhöht, paßt sich die simulierte Kurve in allen Fällen besser an das Spektrum an. Das Augenmerk lag bei der Simulation fast immer auf der Höhe des Stickstoffpeaks, da - wegen der geringen Dicke der Schicht - die Höhe der Siliziumkante kaum zu erkennen ist.
Im folgenden Graph ist der relative Anteil gegenüber der Schichtdicke (gemessen mit dem Bonner Ellipsometer) für die Siliziumproben aufgetragen.



Graph 3.1 : relativer Anteil von Silizium aufgetragen gegenüber der Schichtdicke für die Proben Si3N4 auf Silizium. Der relative Anteil fällt mit der Schichtdicke.

Die Werte für die relativen Anteile sind aus den Simulationen der RBS-Spektren entnommen. Da man jeweils zwei Werte hat, einen für das annular- und einen für das Standardspektrum bzw. 140°-Spektrum, wurde hier der Mittelwerte der beiden Werten genommen. Als Fehler ist für den relativen Anteil ±1 in das Diagramm eingetragen. Für diesen Wert zeigte sich in den Simulationen eine deutliche Abweichung zu den gemessenen Spektren.
Man erkennt einen Abfall im relativen Anteil des Siliziums. Man könnte dies dadurch erklären, daß auf dem Siliziumwafer zuerst bevorzugt Silizium aufwächst aus dem sich später dann Siliziumnitrid bildet. Diese Bevorzugung von Silizium könnte auch für das Wachstum der Siliziuminseln in der Siliziumnitridschicht

verantwortlich sein (siehe Kapitel 2.4). Daß man dieses Wachstum in den Spektren nicht erkennen kann liegt daran, daß die RBS-Methode hierauf nicht empfindlich genug ist. Die Ellipsometrie dagegen reagiert sehr sensitiv auf Änderungen der Zusammensetzung über den Brechungsindex.
Die Spektren der Galliumarsenid-Proben lassen sich außnahmslos durch reine Si_3N_4-Schichten simulieren. Aus diesem Grund wurde auf eine Darstellung in einem Graphen verzichtet. Die RBS-Methode ist also auch bei diesen Proben nicht empfindlich genug ein Ansteigen des Siliziumanteils zu detektieren.

3.4.2 Schichtdickenmessung mit RBS

In der folgenden Tabelle ist ein Vergleich angestellt worden zwischen den Schichtdicken, die aus den RBS-Spektren berechnet wurden und den Werten, die mit dem Bonner-Ellipsometer gemessen wurden.

	x_{RBS} [Å]	x_{ell} [Å]
Si 1/1.	345,560 ± 115,539	205,360
Si 1/3.	341,580 ± 65,006	356,355
Si 1/4.	538,192 ± 167,147	303,698
Si 1/5.	681,503 ± 191,927	692,048
Si 1/6.	337,453 ± 115,539	320,721
Si 2/1.	567,386 ± 8,019	717,296
Si 2/2.	481,284 ± 48,843	587,466
Si 2/3.	421,061 ± 99,419	491,793
Si 2/5.	294,287 ± 11,914	260,878
Si 2/6.	838,499 ± 287,799	811,093
Si 2/7.	737,331 ± 302,464	667,291
Si 2/8.	560,192 ± 98,690	490,047
GaAs 2/4.	629,496 ± 166,418	677,129
GaAs 2/5.	531,441 ± 53,521	729,468
GaAs 2/6.	428,754 ± 140,447	552,040
GaAs 2/8a.	748,301 ± 100,873	984,352
GaAs 2/8b.	685,711 ± 216,179	959,325
GaAs 2/9a.	655,264 ± 37,927	899,467
GaAs 2/9b.	789,675 ± 45,211	1020,908
GaAs 2/12.	608,834 ± 37,798	821,545

Tabelle 3.2 : Vergleich der Schichtdicke gemessen mit RBS und dem Bonner-Ellipsometer. Die Angaben sind in Å gemacht.

Die RBS-Werte sind Durchschnittswerte aus den bis zu vier Werten, die man von einer Probe erhalten kann, nämlich aus jedem der beiden Spektren ein Wert aus dem Siliziumpeak und ein Wert aus dem Stickstoffpeak. Der angegebene Fehler ist die Standardabweichung dieser Werte. In den zwei Fällen, in denen nur ein Wert zur Verfügung stand, wurde als Fehler der Durchschnittswert der anderen Fehler berechnet (115,539Å).
Man kann erkennen, daß die berechneten Schichtdicken größenordnungsmäßig mit den Ellipsometerwerten übereinstimmen. Man muß aber anmerken, daß diese

berechnete Schichtdicke nicht nur wegen des großen Fehlers großer Vorsicht zu behandeln ist. Zuallererst sind die Formeln für das Bremsvermögen aus dem Tabellenwerk von Ziegler [ZIE77] nach den Angaben des Autors für gasförmige Elemente in festen Stoffen (in diesem Fall Stickstoff als Nitrid) kaum durch Experimente gestützt. Zweitens muß man bei zusammengesetzten Stoffen die Bragg-Kleeman-Regel anwenden, die ebenso eine Näherung darstellt, da das Bremsverhalten von Ionen in Festkörpern immer noch nicht sehr gut verstanden ist. Ein weiterer Unsicherheitsfaktor ist in den Spektren zu sehen. Da die untersuchten Proben nur sehr dünne Schichten aus Siliziumnitrid haben, ist es häufig kaum zu erkennen, wo die Peaks von Silizium und Stickstoff anfangen und enden. Vor allem die Stickstoff-Peaks in den standard-Spektren gehen häufig in der geringen Statistik der Spektren unter.
Aus den oben bereits genannten Gründen und aus der Streuung der bis zu vier Werte, die man für eine Schicht hat, kann man den Fehler für die Schichtdickenmessung mit RBS auf etwas mehr als 100Å einschätzen.

4 Zusammenfassung

Nach einem Vergleich der Messungen, die mit dem Bonner-Ellipsometer gemacht wurden, mit denen, die in Surrey gemacht wurden, kann man sagen, daß das von M.Mendel gebaute Ellipsometer genau so gut mißt wie ein kommerziell gekauftes. Das Programm, das im Rahmen dieser Diplomarbeit geschrieben, verbessert die Berechnung der Schichtdicke und des zugehörigen Brechungsindex aus den gemessenen Polarisator- und Analysatorwinkeln.
Die Schichten, die untersucht wurden, zeigten eine Abhängigkeit des Brechungsindex von der Schichtdicke. Bei den Schichten, die auf Silizium aufgedampft wurden, vollzieht sich diese Abhängigkeit in zwei Schritten. Im ersten steigt der Brechungindex bis zu einer Schichtdicke von ca. 300Å von 1,7 bis etwa 2 steil an. Im zweiten Teil flacht die Kurve dann ab und steigt nur noch wenig mit der Schichtdicke. Bei geringen Schichtdicken registriert das Ellipsometer noch die Siliziumoxidschicht, die auf allen unbeschichteten Siliziumproben vorhanden ist. Diese Siliziumoxidschicht wird dann nach und nach von Siliziumnitrid verdeckt. Ab einer Schichtdicke von ca. 300Å geht die Siliziumoxid-Schicht unter und das Ellipsometer registriert nur noch den Brechungsindex von Siliziumnitrid. Ab da kann man vermuten, daß sich Siliziuminseln im Siliziumnitrid bilden, die zusammen mit der Schichtdicke anwachsen. Bei den Galliumarsenid-Proben erkennt man nur den zweiten Teil dieser Entwicklung.
Die RBS-Methode stellte sich als ungeeignet heraus, um diesen Anstieg des Siliziums in den Schichten zu erkennen. Im Gegenteil: bei den Silizium-Proben kann man erkennen, daß wesentlich mehr Silizium in den Schichten vorhanden ist als angenommen wurde. Je dünner die Schicht desto größer ist der Anteil von Silizium. Deswegen kann man annehmen, daß bei Silizium als Substrat zuerst bevorzugt Silizium aufwächst. Das Wachstum von Siliziuminseln in Siliziumnitrid geht in diesen Messungen unter. Auch bei den Galliumarsenidproben ist keine Zunahme des Silizium in der Schicht zu sehen. Diese Schichten bestehen aus Siliziumnitrid in der richtigen stöchiometrischen Zusammensetzung. Sauerstoff oder andere Verunreinigungen sind nicht in den Spektren zu sehen. Zumindest nicht in den Konzentrationen, die eine Rolle spielen könnten. Für die leichten Elemente, wie z.B. Wasserstoff müßte man allerdings noch gesonderte ERDA-Messungen machen.
Auch um die Schichtdickenmessungen zu kontrollieren ist die RBS-Meßmethode nicht geeignet. Die Größenordnungen stimmen zwar überein, aber der Fehler stellt sich als zu groß heraus, als daß man die Werte wirklich vergleichen könnte.
Zusammenfassend kann man sagen, daß die Schichten, die in der kombinierten LPCVD / RTA-Anlage hergestellt werden, wirklich nur aus Siliziumnitrid bestehen. Die Siliziumproben enthalten bei sehr dünnen Schichten zuviel Silizium, aber Sauerstoff oder andere Verunreinigungen haben sich nicht gezeigt.

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent : Priv. Doz. Dr. R. Vianden

Koreferent: Prof. Dr. K. Maier

5 Anhang

5.1 Quellcode des Programms XEllip

5.1.1 Header-Files

//calc.h : Headerfile für die Berechnungsroutinen

```
// rho                :      ellipsometric ratio = tan(psi)*exp(i*delta)
// psi                :      = (alpha1 + alpha2)/2
// delta              :      = (360 - (beta1-beta2))
// alpha1/2           :      Analysatorwinkel
// beta1/2            :      Polarisatorwinkel
// n0, n1, n2         :      refractive indices of ambient, layer, substrate
(respectively)
// lambda             :      wavelength
// phi0               :      angle of incidence
// phi1, phi2         :      angles in layer and substrate (respectively)
// n                  :      relative index = n1/n0
// d                  :      layerthickness
// x, y, z, t,  bigx  :      temporary variables
// a[6], g[7], f[8], j[7], k[6] :    dito


#ifndef _COMPLEX_H
#include "complex.h"
#endif

#ifndef _UTILITY_H
#include "utility.h"
#endif

#ifndef _CALC_H
#define _CALC_H

void step1(long double phi0,
                  complex        phi2,
                  long double n0,
                  complex        n2,
                  complex        rho,
                  complex     &x,
                  complex        &y,
                  complex        &z);

void step2(complex x,
                  complex y,
                  complex z,
                  complex a[6]);

void step3(complex a[6], long double g[7]);

void step4(complex a[6], long double f[8]);

void step5(long double f[8],
                  long double g[7],
                  long double j[7]);

void step6(long double j[7],
                  complex x,
                  complex y,
                  complex z,
                  long double k[6]);
void step7(long double k[6],
                  long double rt[6],
                  int &anzahl);                                    //polynomial
roots

void step81(long double root,
                       long double n0,
                       long double &phi1,
                       long double phi0,
                       long double &n1,
                       long double &t);

void step82(long double t,
                       long double f[8],
```

```
                      long double g[7],
                      complex &bigx);

void step83(long double lambda,
                      long double phi1,
                      long double n1,
                      complex bigx,
                      long double &d,
                      long double &d1);

void Calculation(long double alpha1,
                              long double alpha2,
                              long double beta1,
                              long double beta2,
                              long double lambda,
                              long double phi0,
                              long double n0,
                              complex       n2,
                              long double *n1,
                              long double *d,
                              int           &anzahl,
                              long double *d1,
                              long double *root);

BOOL Calculation2(long double phi0,
                               long double n0,
                               long double n1,
                               complex        n2,
                               long double lambda,
                               long double psi,
                               long double delta,
                               long double d);

#endif // _CALC_H
```

//complex.h : Klasse der komplexen Zahlen

```
#ifndef _COMPLEX_H
#define _COMPLEX_H

#ifndef _UTILITY_H
#include "utility.h"
#endif

class complex
{
protected:
private:
//Instanzvariablen
        long double real;
        long double imag;

//Funktionen
public:

        complex(long double a=0.0, long double b=0.0);             //Konstruktor
        complex(long double a,                      double b);
        complex(long double a,                      int    b);
        complex(        double a,        long double b);
        complex(        double a,                   double b);
        complex(        double a,                   int    b);
        complex(        int    a,        long double b);
        complex(        int    a,                   double b);
        complex(        int    a,                   int    b);
        //~complex();                                //Destruktor

    complex* operator= (complex* a);
        complex* operator= (long double* a);
        complex* operator= (double *a);

        complex operator+ (complex b);
        friend complex operator+ (      complex a, long double b);
        friend complex operator+ (      complex a,     double b);
        friend complex operator+ (      complex a,     int    b);
        friend complex operator+ (long double a, complex b);
        friend complex operator+ (       double a, complex b);
        friend complex operator+ (              int a, complex b);

        complex operator- (complex b);
        friend complex operator- (      complex a, long double b);
        friend complex operator- (      complex a,     double b);
```

```
        friend complex operator- (     complex a,     int    b);
        friend complex operator- (long double a, complex b);
        friend complex operator- (      double a, complex b);
        friend complex operator- (             int a, complex b);

        complex operator* (complex b);
        friend complex operator* (     complex a, long double b);
        friend complex operator* (     complex a,     double b);
        friend complex operator* (     complex a,     int    b);
        friend complex operator* (long double a, complex b);
        friend complex operator* (      double a, complex b);
        friend complex operator* (             int a, complex b);

        complex operator/ (complex b);
        friend complex operator/ (     complex a, long double b);
        friend complex operator/ (     complex a,     double b);
        friend complex operator/ (     complex a,     int    b);
        friend complex operator/ (long double a, complex b);
        friend complex operator/ (      double a, complex b);
        friend complex operator/ (             int a, complex b);

        complex operator^ (int p);

        friend complex sinl(complex a);
        friend complex cosl(complex a);
        friend complex tanl(complex a);
        friend complex cotl(complex a);

        friend complex sinhl(complex a);
        friend complex coshl(complex a);
        friend complex tanhl(complex a);
        friend complex cothl(complex a);

        friend complex asinl(complex a);
        friend complex acosl(complex a);
        friend complex atanl(complex a);
        friend complex acotl(complex a);

        friend complex asinhl(complex a);
        friend complex acoshl(complex a);
        friend complex atanhl(complex a);
        friend complex acothl(complex a);

        friend complex ln(complex a);
        friend long double abs(complex a);
        friend long double sign(long double a);
        friend complex cc(complex a);
        friend complex sqrt(complex a);
        friend complex exp(complex a);
        friend complex sqr(complex a);
        friend complex cube(complex a);

        BOOL operator== (complex b);

        friend long double  Re(complex a) {return a.real;};
        friend long double  Im(complex a) {return a.imag;};


}; //complex


#endif  // _COMPLEX_H
```

// dmwerte.h : header file für den Dialog

```
// DMwerte dialog

class DMwerte : public CDialog
{
// Construction
public:
        double analysator1;
        double analysator2;
        double polarisator1;
        double polarisator2;
        double lambda;
        double n2real;
        double n2imag;
        double phi;


        DMwerte(CWnd* pParent = NULL);       // standard constructor
```

```
// Dialog Data
        //{{AFX_DATA(DMwerte)
        enum { IDD = IDD_MWERTEDLG };
        double  m_ana1;
        double  m_ana2;
        double  m_lambda;
        double  m_n2imag;
        double  m_n2real;
        double  m_phi;
        double  m_pol1;
        double  m_pol2;
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        // Generated message map functions
        //{{AFX_MSG(DMwerte)
        virtual BOOL OnInitDialog();
        afx_msg void OnSetfocusAnalysator1();
        afx_msg void OnKillfocusAnalysator1();
        afx_msg void OnKillfocusAnalysator2();
        afx_msg void OnSetfocusAnalysator2();
        afx_msg void OnKillfocusPolarisator1();
        afx_msg void OnSetfocusPolarisator1();
        afx_msg void OnKillfocusPolarisator2();
        afx_msg void OnSetfocusPolarisator2();
        virtual void OnOK();
        afx_msg void OnKillfocuslambda();
        afx_msg void OnSetfocuslambda();
        afx_msg void OnSetfocusn2imag();
        afx_msg void OnKillfocusn2imag();
        afx_msg void OnKillfocusn2real();
        afx_msg void OnSetfocusn2real();
        afx_msg void OnSetfocusphi();
        afx_msg void OnKillfocusphi();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

//Utility.h : Utility headerfile für nützliche Routinen und Konstanten

```
#ifndef _COMPLEX_H
#include "complex.h"
#endif

#ifndef _UTILITY_H
#define _UTILITY_H


#define PI            3.141592654
#define RAD    (PI / 180.0)
#define I             (complex(0.0, 1.0))
#define FALSE  0
#define TRUE   1
#define NanoM  (pow(10.0, -9.0))
#define MykroM (pow(10.0, -6.0))
#define MilliM (pow(10.0, -3.0))

typedef int BOOL;

long double cot(long double a);
long double coth(long double a);


#endif // _UTILITY_H
```

// mainfrm.h : interface of the CMainFrame class

```
class CMainFrame : public CMDIFrameWnd
{
        DECLARE_DYNAMIC(CMainFrame)
public:
        CMainFrame();

// Attributes
public:
```

```
// Operations
public:

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
	CStatusBar  m_wndStatusBar;
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	//{{AFX_MSG(CMainFrame)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
		// NOTE - the ClassWizard will add and remove member functions
here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

// stdafx.h : include file for standard system include files,

```
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions (including VB)
```

// xellidoc.h : interface of the CXellipDoc class

```
class CXellipDoc : public CDocument
{
protected: // create from serialization only
	CXellipDoc();
	DECLARE_DYNCREATE(CXellipDoc)

// Attributes
public:
// Operations
public:

// Implementation
public:
	virtual ~CXellipDoc();
	virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
	virtual BOOL OnNewDocument();

// Generated message map functions
protected:
	//{{AFX_MSG(CXellipDoc)
	afx_msg void OnExtrasMesswerteingabe();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

// xellip.h : main header file for the XELLIP application

```
#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CXellipApp:
// See xellip.cpp for the implementation of this class
```

```
//

class CXellipApp : public CWinApp
{
public:
	CXellipApp();

// Overrides
	virtual BOOL InitInstance();

// Implementation

	//{{AFX_MSG(CXellipApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions
here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

// xellivw.h : interface of the CXellipView class

```
class CXellipView : public CView
{
protected: // create from serialization only
	CXellipView();
	DECLARE_DYNCREATE(CXellipView)

// Attributes
public:
	CXellipDoc* GetDocument();

// Operations
public:

// Implementation
public:
	virtual ~CXellipView();
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

	// Printing support
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Generated message map functions
protected:
	//{{AFX_MSG(CXellipView)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in xellivw.cpp
inline CXellipDoc* CXellipView::GetDocument()
   { return (CXellipDoc*)m_pDocument; }
#endif
```

5.1.2 Cpp-Files

//calc.cpp : Berechnungsroutinen der Ellipsometrie

```
#include "calc.h"

#ifndef _MATH_H
#include <math.h>
#endif

#ifndef _UTILITY_H
#include "utility.h"
#endif
```

```
#ifndef _COMPLEX_H
#include "complex.h"
#endif

#ifndef _AFX_H
#include "afx.h"
#endif

void step1(long double phi0,
                  complex        phi2,
                  long double n0,
                  complex n2,
                  complex rho,
                  complex &x,
                  complex &y,
                  complex &z)
{
       x = (-1) * tanl(phi0) * tanl(phi2);
       y = ( n2 * cosl(phi2) );
       y = (-1) * y;
       y = y / ( n0 * cosl(phi0) );
       z = (-1) * ( rho - 1 ) / ( rho + 1 );
}

void step2(complex x,
                  complex y,
                  complex z,
                  complex a[6])
{
       a[0] = (x^2)*(y^3)*z;
       a[1] = x*(y^2)                      + x*(y^3)*z + 2*(x^2)*(y^2)*z;
       a[2] = (y^2)                + 2*x*y        + 2*x*(y^2)*z         +
(x^2)*y*z;
       a[3] = x                            + 2*y          + x*y*z;
       a[4] = (-1)*(y^2)           + 2*x*y        + 2*x*z                      +
(x^2)*y*z      + 1;
       a[5] = (-2)*x*(y^3)   - (y^2)        - (x^2)*(y^3)*z       - 2*x*(y^2)*z  +
(x^2)*y*z;
}

void step3(complex a[6], long double g[7])
{
       g[0] = Re( a[0]*cc(a[1]) + a[1]*cc(a[0]) );
       g[2] = Re( a[0]*cc(a[3]) + a[1]*cc(a[2])    + a[2]*cc(a[1]) + a[3]*cc(a[0])
);
       g[4] = Re( a[1]                      + a[2]*cc(a[3])      + a[3]*cc(a[2])
+         cc(a[1]) );
       g[6] = Re( a[3]                      +         cc(a[3]) );
}

void step4(complex a[6], long double f[8])
{
       f[1] = Re( I * ( (-1)*a[0]*cc(a[5]) - a[0]*cc(a[2]) + a[5]*cc(a[0]) +
a[2]*cc(a[0]) ) );
       f[2] = Re(              a[1]*cc(a[5]) - a[0]*cc(a[3]) + a[5]*cc(a[1]) -
a[3]*cc(a[0]) );
       f[3] = Re( I * ( (-1)*a[0]                  - a[2]*cc(a[5]) - a[0]*cc(a[4])
+ a[5]*cc(a[2]) + a[4]*cc(a[0]) + cc(a[0]) ) );
       f[4] = Re(              a[3]*cc(a[5]) + a[1]*cc(a[4]) + a[5]*cc(a[3]) +
a[4]*cc(a[1]) );
       f[5] = Re( I * (        a[5]                - a[0]                -
cc(a[5]) - a[2]*cc(a[4]) + a[4]*cc(a[2]) + cc(a[0]) ) );
       f[6] = Re(                        a[1]               + a[3]*cc(a[4]) +
a[4]*cc(a[3]) +          cc(a[1]) );
       f[7] = Re( I * (        a[4]                - a[2]                -
cc(a[4]) +       cc(a[2]) ) );
}

void step5(long double f[8],
                  long double g[7],
                  long double j[7])
{
       j[0] = f[1]*f[1] + 2*g[0]*g[0] - 2*f[2]*g[0] - 2*g[2]*g[0];
       j[1] = f[2]*f[2] + 2*f[3]*f[1] -   g[2]*g[2] -   g[0]*g[0] - 2*f[4]*g[0] -
2*g[4]*g[0] + 4*g[2]*g[0];
       j[2] = f[3]*f[3] + 2*f[4]*f[2] + 2*f[5]*f[1] + 2*g[2]*g[2] - 2*g[4]*g[2] -
2*f[6]*g[0] - 2*g[6]*g[0] + 4*g[4]*g[0] - 2*g[2]*g[0];
       j[3] = f[4]*f[4] + 2*f[5]*f[3] + 2*f[6]*f[2] + 2*f[7]*f[1] -   g[4]*g[4] -
g[2]*g[2] - 2*g[6]*g[2] + 4*g[4]*g[2] + 2*g[6]*g[0] - 2*g[4]*g[0];
       j[4] = f[5]*f[5] + 2*f[6]*f[4] + 2*f[7]*f[3] + 2*f[2]*g[6] + 2*g[4]*g[4] -
2*g[6]*g[4] + 4*g[6]*g[2] - 2*g[4]*g[2] - 2*g[6]*g[0];
       j[5] = f[6]*f[6] + 2*f[7]*f[5] -   g[6]*g[6] + 2*f[4]*g[6] -   g[4]*g[4] +
4*g[6]*g[4] - 2*g[6]*g[2];
```

```
       j[6] = f[7]*f[7] + 2*g[6]*g[6] + 2*f[6]*g[6] - 2*g[6]*g[4];
}

void step6(long double j[7],
                  complex x,
                  complex y,
                  complex z,
                  long double k[6])
{
       k[5] = Re(                              j[1] - 2*j[2]*x*y +
3*j[3]*(x^2)*(y^2) -  4*j[4]*(x^3)*(y^3) +  5*j[5]*(x^4)*(y^4) -
6*j[6]*(x^5)*(y^5));
       k[4] = Re(  (x*y+1)    * ( j[2] - 3*j[3]*x*y +  6*j[4]*(x^2)*(y^2) -
10*j[5]*(x^3)*(y^3) + 15*j[6]*(x^4)*(y^4) ));
       k[3] = Re( ((x*y+1)^2) * ( j[3] - 4*j[4]*x*y + 10*j[5]*(x^2)*(y^2) -
20*j[6]*(x^3)*(y^3) ));
       k[2] = Re( ((x*y+1)^3) * ( j[4] - 5*j[5]*x*y + 15*j[6]*(x^2)*(y^2) ));
       k[1] = Re( ((x*y+1)^4) * ( j[5] - 6*j[6]*x*y ));
       k[0] = Re( ((x*y+1)^5) * ( j[6] ));
}

void step7(long double k[6],
                  long double rt[6],
                  int           &anzahl)  //polynomial roots
{
    // k[0] ist der Koeffizient der hoechsten Potenz
    // k[5] ist die Konstante am Ende

       long double a[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};//       = {1.0, -7.0, -
3.0, 79.0, -46.0, -120.0};
       long double b[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
       long double c;
       long double y;
       int            r, s, zahl, anz;                    // Zaehlvariablen
       long double genauigkeit = powl(10, -15);
       int                   N = 5;                // Grad des Polynoms

       anz    = 1;
       rt[1]  = 4.0;
       for (zahl = 0; zahl < 6; zahl++)     a[zahl] = k[zahl]/k[0];

       for (s = 1; s < 6; s++)                                    // Hauptschleife
       {
              for (zahl = 1; zahl < 101; zahl++)
              {
                  // falls der Grad N nur 1 betraegt
                     if (N == 1)
                            rt[anz] = - a[1] / a[0];
                     b[0] = a[0];

                     // f(x) auswerten
                     for (r = 1; r<=N; r++)
                     {
                            b[r] = b[r-1] * rt[anz] + a[r];
                     }

                     // f´(x) auswerten
                     c = b[0];
                     for (r = 1; r<N; r++)
                     {
                            c = c * rt[anz] + b[r];
                     }

                     // Newton´sche Formel
                     y = b[N] / c;
                     rt[anz] -= y;
                     if (fabsl(y) < genauigkeit)   // falls Nullstelle gefunden
                            {
                                   zahl = 101;
                                   anz++;
                                   if (anz != 6)
                                   {
                                          rt[anz] = rt[anz-1];

                                   }
                                   N--;
                                   for(r = 1; r <= N; r++)      a[r] = b[r];
                            }
              }
              if (fabsl(y) > genauigkeit) s = 6;
       }

       if ( (anz==2)||(anz==4) )
```

```
        {
                anzahl = anz-1;
        }
        else anzahl = anz;
}

void step81(long double root,
                        long double n0,
                        long double &phi1,
                        long double phi0,
                        long double &n1,
                        long double &t)
{
        n1              = n0 * sqrtl(root);
        phi1    = n1*n1 - n0*n0*sinl(phi0)*sinl(phi0);
        phi1    = sqrtl( n1*n1 - n0*n0*sinl(phi0)*sinl(phi0) ) / n1;
        phi1    = acosl( sqrtl( n1*n1 - n0*n0*sinl(phi0)*sinl(phi0) ) / n1 );
        long double phi12 = (asinl( n0 / (n1) * sinl(phi0) ));   // angle in layer

        t               = ( n1*cosl(phi1) ) / ( n0*cosl(phi0) );
}

void step82(long double t,
                        long double f[8],
                        long double g[7],
                        complex &bigx)
{
        complex zaehler, nenner;
        nenner = (t*t - 1.0) * ( g[6]*powl(t,6) + g[4]*powl(t,4) + g[2]*powl(t,2) +
g[0] );
        zaehler = g[6]*powl(t,8);
        zaehler = g[6]*powl(t,8) + f[6]*powl(t,6);
        zaehler = g[6]*powl(t,8) + f[6]*powl(t,6) + f[4]*powl(t,4);
        zaehler = g[6]*powl(t,8) + f[6]*powl(t,6) + f[4]*powl(t,4) +
f[2]*powl(t,2);
        zaehler = g[6]*powl(t,8) + f[6]*powl(t,6) + f[4]*powl(t,4) + f[2]*powl(t,2)
- g[0];
        zaehler = g[6]*powl(t,8) + f[6]*powl(t,6) + f[4]*powl(t,4) + f[2]*powl(t,2)
- g[0];
        zaehler = zaehler + I*( f[7]*powl(t,7) + f[5]*powl(t,5) + f[3]*powl(t,3) +
f[1]*t );
        bigx    = nenner/zaehler;
}

void step83(long double lambda,
                        long double phi1,
                        long double n1,
                        complex        logbigx,
                        long double &d,
                        long double &d1)
{
    complex d2;
    //d2 = (I*ln(bigx)*lambda) / (4.0*PI*cosl(phi1)*n1) ;
    d2 = (0.25 * lambda / (PI*cosl(phi1)*n1)) * I * logbigx;
    d  = Re(d2);
    d1 = 0.5*(1.0*lambda) / (cosl(phi1)*n1);
    if  ( d>d1 ) d-=d1;
}

void Calculation(long double alpha1, long double alpha2,
                                long double beta1,  long double beta2,
                                long double lambda, long double phi0,
                                long double n0,     complex   n2,
                                long double *n1,    long double *d,
                                int             &anzahl,long double *d1,
                                long double *root)
{
        //alpha1/2      :     Analysatorwinkel
        //beta1/2               :       Polarisatorwinkel
        //lambda                :       wavelength
        //phi0                  :       angle of incidence
        //n0                    :       refractive index of ambient
        //n2                    :       refractive index of substrate
    //needed to calculate:
        //n1                    :       refractive index of layer
        //d                             :       layerthickness

        // temporary variables:
        complex         x, y, z;

        complex         bigx[6];

        complex         a[6];
```

```
        long double g[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        long double f[8] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        long double j[7] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        long double k[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        int           n;
        // Zaehlvariablen
        int           pizahl = 0;
        BOOL          weiter;

        long double   phi1[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        long double   t[6]   = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};


    if (alpha1>180)          alpha1 = 360 - alpha1;
    if (alpha2>180)          alpha2 = 360 - alpha2;
        long double   psi           = RAD * (alpha1 + alpha2)/2;
        long double   delta  = RAD * (360 - (beta1+beta2));
        complex       rho           = tanl(psi)*exp(I*delta);
        // ellipsometric ratio
        complex       phi2    = (asinl( n0 * sinl(phi0) / n2 ));  // angle
in substrate
        complex       logbigx;

        step1(phi0, phi2, n0, n2, rho, x, y, z);
        step2(x, y, z, a);
        step3(a, g);
        step4(a, f);
        step5(f, g, j);
        step6(j, x, y, z, k);
        step7(k, root, anzahl);             //polynomial roots

        for (n = 1; n <= anzahl; n++)
        {
               if ( root[n] > (sinl(phi0)*sinl(phi0)) )
               {
                      // calculation of n1, phi1, t
                      step81(root[n], n0, phi1[n], phi0, n1[n], t[n]);
                      // calculation of bigx
                      step82(t[n], f, g, bigx[n]);

                      // calculation of d, d1

                      do
                      {
                          logbigx = ln(bigx[n]) - pizahl * I * PI;
                             step83(lambda, phi1[n], n1[n], logbigx, d[n], d1[n]);
                             pizahl++;
                             weiter = (Calculation2(phi0, n0, n1[n], n2, lambda,
psi, delta, d[n]) == FALSE) && (pizahl<2);
                      }
                      while (weiter);
               }
        }
}

BOOL Calculation2(long double phi0,
                              long double n0,
                              long double n1,
                              complex       n2,
                              long double lambda,
                              long double psi,
                              long double delta,
                              long double d)
{
        long double cosphi0 = cosl(phi0);
        long double sinphi0 = sinl(phi0);
        complex       cosphi1 = sqrtl(n1*n1 - n0*n0*sinphi0*sinphi0) / n1;
        complex       cosphi2 = sqrt( n2*n2 - n0*n0*sinphi0*sinphi0) / n2;
        complex       bigx = exp(I*(-4)*PI*cosphi1*d*n1/lambda);
        complex       rp01 = (-n0*cosphi1 + n1*cosphi0)/(n0*cosphi1 + n1*cosphi0);
        complex       rp12 = (-n1*cosphi2 + n2*cosphi1)/(n1*cosphi2 + n2*cosphi1);
        complex       rs01 = (-n1*cosphi1 + n0*cosphi0)/(n1*cosphi1 + n0*cosphi0);
        complex       rs12 = ((-1)*n2*cosphi2 + n1*cosphi1)/(n2*cosphi2 +
n1*cosphi1);
        complex       rho1 = ( (bigx*rp12 + rp01)*(bigx*rs12*rs01 + 1.0) )/(
(bigx*rp12*rp01 + 1.0)*(bigx*rs12 + rs01) );

        complex       rho2   = tanl(psi)*exp(I*delta);

        if (rho1 == rho2) return TRUE;
               else return FALSE;
}
```

// complex.cpp file

```
#include "complex.h"

#ifndef _MATH_H
#include <math.h>
#endif

#ifndef _UTILITY_H
#include "utility.h"
#endif

//Konstruktor
complex::complex(long double a, long double b)
{
	real = a;
	imag = b;
}
complex::complex(long double a, double b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(long double a, int b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(double a, long double b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(double a, double b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(double a, int b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(int a, long double b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(int a, double b)
{
	real = (long double)a;
	imag = (long double)b;
}
complex::complex(int a, int b)
{
	real = (long double)a;
	imag = (long double)b;
}
/*
//Destruktor
complex::~complex()
{

}
*/


//operator =
complex* complex::operator= (complex *a)
{
	complex *res;

	res  =  new complex;
	res->real = a->real;
	res->imag = a->imag;
	return res;
}
complex* complex::operator= (long double *a)
{
	complex *res;

	res  =  new complex;
```

```
        res->real = (long double)(*a);
        res->imag = 0.0;
        return res;
}
complex* complex::operator= (double *a)
{
        complex *res;

        res  =  new complex;
        res->real = (long double)(*a);
        res->imag = 0.0;
        return res;
}


//operator +
complex complex::operator+ (complex b)
{
        complex res;

        res.real = real + b.real;
        res.imag = imag + b.imag;
        return res;
}
complex operator+ (long double a, complex b)
{
        complex res((long double)a, 0.0);
        res = res + b;
        return res;
}
complex operator+ (complex a, long double b)
{
        complex res((long double)b, 0.0);
        res = a + res;
        return res;
}
complex operator+ (double a, complex b)
{
        complex res((long double)a, 0.0);
        res = res + b;
        return res;
}
complex operator+ (complex a, double b)
{
        complex res((long double)b, 0.0);
        res = a + res;
        return res;
}
complex operator+ (complex a, int b)
{
        complex res((long double)b, 0.0);
        res = a + res;
        return res;
}
complex operator+ (int a, complex b)
{
        complex res((long double)a, 0.0);
        res = res + b;
        return res;
}


//operator -
complex complex::operator- (complex b)
{
        complex res;

        res.real = real - b.real;
        res.imag = imag - b.imag;
        return res;
}
complex operator- (long double a, complex b)
{
        complex res((long double)a, 0.0);
        res = res - b;
        return res;
}
complex operator- (complex a, long double b)
{
        complex res((long double)b, 0.0);
        res = a - res;
        return res;
}
```

```
complex operator- (double a, complex b)
{
	complex res((long double)a, 0.0);
	res = res - b;
	return res;
}
complex operator- (complex a, double b)
{
	complex res((long double)b, 0.0);
	res = a - res;
	return res;
}
complex operator- (int a, complex b)
{
	complex res((long double)a, 0.0);
	res = res - b;
	return res;
}
complex operator- (complex a, int b)
{
	complex res((long double)b, 0.0);
	res = a - res;
	return res;
}


//operator *
complex complex::operator* (complex b)
{
	complex res;

	res.real = real*b.real - imag*b.imag;
	res.imag = real*b.imag + b.real*imag;
	return res;
}
complex operator* (long double a, complex b)
{
	complex res((long double)a, 0.0);
	res = res * b;
	return res;
}
complex operator* (complex a, long double b)
{
	complex res((long double)b, 0.0);
	res = a * res;
	return res;
}
complex operator* (double a, complex b)
{
	complex res((long double)a, 0.0);
	res = res * b;
	return res;
}
complex operator* (complex a, double b)
{
	complex res((long double)b, 0.0);
	res = a * res;
	return res;
}
complex operator* (int a, complex b)
{
	complex res((long double)a, 0.0);
	res = res * b;
	return res;
}
complex operator* (complex a, int b)
{
	complex res((long double)b, 0.0);
	res = a * res;
	return res;
}


//operator /
complex complex::operator/ (complex b)
{
	complex res;
	long double factor;

	factor  = b.real*b.real + b.imag*b.imag;
	res.real = (real*b.real + imag*b.imag)/factor;
	res.imag = (b.real*imag - real*b.imag)/factor;
	return res;
```

```
}
complex operator/ (long double a, complex b)
{
	complex res((long double)a, 0.0);
	res = res / b;
	return res;
}
complex operator/ (complex a, long double b)
{
	complex res((long double)b, 0.0);
	res = a / res;
	return res;
}
complex operator/ (double a, complex b)
{
	complex res((long double)a, 0.0);
	res = res / b;
	return res;
}
complex operator/ (complex a, double b)
{
	complex res((long double)b, 0.0);
	res = a / res;
	return res;
}
complex operator/ (int a, complex b)
{
	complex res((long double)a, 0.0);
	res = res / b;
	return res;
}
complex operator/ (complex a, int b)
{
	complex res((long double)b, 0.0);
	res = a / res;
	return res;
}


// potenzieren
complex complex::operator^ (int p)
{
	complex res(1.0, 0.0);
	complex a(real, imag);
	//res = exp( p * ln(res) );
	for (int i = 1; i <= p ; i++)
		{
			res = res * a;
		}
	return res;
}

// sinl, cosl, tanl, cotl
complex sinl(complex a)
{
	complex res;
	complex z;

	z		= a ;
	res.real = (::sinl(z.real)) * (::coshl(z.imag));
	res.imag = (::cosl(z.real)) * (::sinhl(z.imag));
	return res;
}
complex cosl(complex a)
{
	complex res;
	complex z;

	z		= a;
	res.real =	 (::cosl(z.real)) * (::coshl(z.imag));
	res.imag = (-1.0) * (::sinl(z.real)) * (::sinhl(z.imag));
	return res;
}
complex tanl(complex a)
{
	complex res;
	complex z;

	z   = a;
	res = sinl(z)/cosl(z);
	return res;
}
complex cotl(complex a)
```

```
{
        complex res;
        complex z;

        z   = a;
        res = cosl(z)/sinl(z);
        return res;
}

// sinh, cosh, tanlh, coth
complex sinhl(complex a)
{
        complex res;
        complex z;

        z              = a;
        res.real = (::sinhl(z.real)) * (::cosl(z.imag));
        res.imag = (::coshl(z.real)) * (::sinl(z.imag));
        return res;
}
complex coshl(complex a)
{
        complex res;
        complex z;

        z              = a;
        res.real = (::coshl(z.real)) * (::cosl(z.imag));
        res.imag = (::sinhl(z.real)) * (::sinl(z.imag));
        return res;
}
complex tanhl(complex a)
{
        complex res;
        complex z;

        z   = a;
        res = sinhl(z)/coshl(z);
        return res;
}
complex cothl(complex a)
{
        complex res;
        complex z;

        z   = a;
        res = coshl(z)/sinhl(z);
        return res;
}

// asinl, acosl, atanl, acotl
complex asinl(complex a)
{
        complex res;
        //getestet!!

        res = - 1.0 * I * ln( I * a + sqrt(1.0 - a * a) );
        return res;
}
complex acosl(complex a)
{
        complex res;

        res = - 1.0 * I * ln( a + sqrt(a * a - 1.0) );
        if (a.real*a.imag < 0)     //warum???
                {
                        res = res * (-1);
                }
        return res;
}
complex atanl(complex a)
{
        complex res;
        //stimmt dies???

        res = - 0.5 * I * ln( (1.0 + I * a) / (1.0 - I * a) );
        return res;
}
complex acotl(complex a)
{
        complex res;
        //stimmt dies???

        res = 0.5 * I * ln( (I * a + 1.0) / (I * a - 1.0) );
```

```
        return res;
}

// asinhl, acoshl, atanhl, acothl
complex asinhl(complex a)
{
        complex res;

        res = ln( a + sqrt(a * a + 1.0) );
        return res;
}
complex acoshl(complex a)
{
        complex res;

        res = ln( a + sqrt(a * a - 1.0) );
        return res;
}
complex atanhl(complex a)
{
        complex res;

        res = 0.5 * ln( (1.0 + a) / (1.0 - a) );
        return res;
}
complex acothl(complex a)
{
        complex res;

        res = 0.5 * ln( (a + 1.0) / (a - 1.0) );
        return res;
}


// ln, abs, conj, sqrt, exp
complex ln(complex a)
{
        complex res;

        res.real = a.real*a.real + a.imag*a.imag;
        res.real = 0.5 * logl(a.real*a.real + a.imag*a.imag);
        res.imag = (::atanl(a.imag/a.real));
        return res;
}
long double abs(complex a)
{
        long double res;

        res = ::sqrtl(a.real*a.real + a.imag*a.imag);
        return res;
}
long double sign(long double a)
{
        long double res;
        res = 1.0;
        if (a < 0) res = -1.0;
        return res;
}
complex cc(complex a)
{
        complex res;

        res.real = a.real;
        res.imag = -a.imag;
        return res;
}
complex sqrt(complex a)
{
        complex res;

        res.real =                              ::sqrtl( (  a.real + abs(a))/2.0 );
        res.imag = sign(a.imag) * ::sqrtl( (- a.real + abs(a))/2.0 );
        return res;
 }
complex exp(complex a)
{
        complex res;

        res.real = (::expl(a.real))*(::cosl(a.imag));
        res.imag = (::expl(a.real))*(::sinl(a.imag));
        return res;
}
complex sqr(complex a)
```

```
{
	complex res;

	res = a^2;
	return res;
}
complex cube(complex a)
{
	complex res;

	res = a^3;
	return res;
}
BOOL complex::operator== (complex b)
{
	long double genau = 0.000001;
	complex res;

	res.real = real - b.real;
	if (res.real<0.0) res.real *= -1;
	res.imag = imag - b.imag;
	if (res.imag<0.0) res.imag *= -1;

	if ( ( res.real <= genau) && (res.imag <= genau) )
	return TRUE;
	else return FALSE;
}
```

// dmwerte.cpp : implementation file

```
#include "stdafx.h"
#include "xellip.h"
#include "dmwerte.h"

#ifndef _CALC_H
#include "calc.h"
#endif
#ifndef _UTILITY_H
#include "utility.h"
#endif
#ifndef _MATH_H
#include "math.h"
#endif

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// DMwerte dialog


DMwerte::DMwerte(CWnd* pParent /*=NULL*/)
	: CDialog(DMwerte::IDD, pParent)
{
	//{{AFX_DATA_INIT(DMwerte)
	m_ana1 = 0;
	m_ana2 = 0;
	m_lambda = 0;
	m_n2imag = 0;
	m_n2real = 0;
	m_phi = 0;
	m_pol1 = 0;
	m_pol2 = 0;
	//}}AFX_DATA_INIT
}

void DMwerte::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(DMwerte)
	DDX_Text(pDX, IDC_ANA1, m_ana1);
	DDV_MinMaxDouble(pDX, m_ana1, 0., 360.);
	DDX_Text(pDX, IDC_ANA2, m_ana2);
	DDV_MinMaxDouble(pDX, m_ana2, 0., 360.);
	DDX_Text(pDX, IDC_lambda, m_lambda);
	DDX_Text(pDX, IDC_n2imag, m_n2imag);
	DDX_Text(pDX, IDC_n2real, m_n2real);
	DDX_Text(pDX, IDC_phi, m_phi);
	DDV_MinMaxDouble(pDX, m_phi, 0., 90.);
	DDX_Text(pDX, IDC_POL1, m_pol1);
```

```
        DDV_MinMaxDouble(pDX, m_pol1, 0., 360.);
        DDX_Text(pDX, IDC_POL2, m_pol2);
        DDV_MinMaxDouble(pDX, m_pol2, 0., 360.);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DMwerte, CDialog)
        //{{AFX_MSG_MAP(DMwerte)
        ON_EN_SETFOCUS(IDC_ANA1, OnSetfocusAnalysator1)
        ON_EN_KILLFOCUS(IDC_ANA1, OnKillfocusAnalysator1)
        ON_EN_KILLFOCUS(IDC_ANA2, OnKillfocusAnalysator2)
        ON_EN_SETFOCUS(IDC_ANA2, OnSetfocusAnalysator2)
        ON_EN_KILLFOCUS(IDC_POL1, OnKillfocusPolarisator1)
        ON_EN_SETFOCUS(IDC_POL1, OnSetfocusPolarisator1)
        ON_EN_KILLFOCUS(IDC_POL2, OnKillfocusPolarisator2)
        ON_EN_SETFOCUS(IDC_POL2, OnSetfocusPolarisator2)
        ON_EN_KILLFOCUS(IDC_lambda, OnKillfocuslambda)
        ON_EN_SETFOCUS(IDC_lambda, OnSetfocuslambda)
        ON_EN_SETFOCUS(IDC_n2imag, OnSetfocusn2imag)
        ON_EN_KILLFOCUS(IDC_n2imag, OnKillfocusn2imag)
        ON_EN_KILLFOCUS(IDC_n2real, OnKillfocusn2real)
        ON_EN_SETFOCUS(IDC_n2real, OnSetfocusn2real)
        ON_EN_SETFOCUS(IDC_phi, OnSetfocusphi)
        ON_EN_KILLFOCUS(IDC_phi, OnKillfocusphi)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// DMwerte message handlers

BOOL DMwerte::OnInitDialog()
{
        CDialog::OnInitDialog();

        m_ana1 = 0.0;
        m_ana2 = 0.0;
        m_pol1 = 0.0;
        m_pol2 = 0.0;
        m_lambda = 632.8;
        m_n2real = 3.85;
        m_n2imag = -0.02;
        m_phi   = 70.0;

    UpdateData(FALSE);

        analysator1    = m_ana1;
        analysator2    = m_ana2;
        polarisator1   = m_pol1;
        polarisator2   = m_pol2;
        lambda                 = m_lambda;
        n2real                 = m_n2real;
        n2imag                 = m_n2imag;
        phi                    = m_phi;

        return TRUE;  // return TRUE  unless you set the focus to a control
}

/*void DMwerte::WriteDefaultParameter();
{

}*/

void DMwerte::OnSetfocusAnalysator1()
{
        UpdateData();
        m_ana1 = analysator1;
        UpdateData();
}
void DMwerte::OnSetfocusAnalysator2()
{
        UpdateData();
        m_ana2 = analysator2;
        UpdateData();
}
void DMwerte::OnSetfocusPolarisator1()
{
        UpdateData();
        m_pol1 = polarisator1;
        UpdateData();
}
void DMwerte::OnSetfocusPolarisator2()
{
```

```
	UpdateData();
	m_pol2 = polarisator2;
	UpdateData();
}
void DMwerte::OnSetfocuslambda()
{
	UpdateData();
	m_lambda = lambda;
	UpdateData();
}
void DMwerte::OnSetfocusn2real()
{
	UpdateData();
	m_n2real = n2real;
	UpdateData();
}
void DMwerte::OnSetfocusn2imag()
{
	UpdateData();
	m_n2imag = n2imag;
	UpdateData();
}
void DMwerte::OnSetfocusphi()
{
	UpdateData();
	m_phi = phi;
	UpdateData();
}
void DMwerte::OnKillfocusAnalysator1()
{
	UpdateData();
	analysator1 = m_ana1;
	UpdateData();
}
void DMwerte::OnKillfocusAnalysator2()
{
	UpdateData();
	analysator2 = m_ana2;
	UpdateData();
}
void DMwerte::OnKillfocusPolarisator1()
{
	UpdateData();
	polarisator1 = m_pol1;
	UpdateData();
}
void DMwerte::OnKillfocusPolarisator2()
{
	UpdateData();
	polarisator2 = m_pol2;
	UpdateData();
}
void DMwerte::OnKillfocuslambda()
{
	UpdateData();
	lambda = m_lambda;
	UpdateData();
}
void DMwerte::OnKillfocusn2imag()
{
	UpdateData();
	n2imag = m_n2imag;
	UpdateData();
}
void DMwerte::OnKillfocusn2real()
{
	UpdateData();
	n2real = m_n2real;
	UpdateData();
}
void DMwerte::OnKillfocusphi()
{
	UpdateData();
	phi = m_phi;
	UpdateData();
}


void DMwerte::OnOK()
{
	char    str[2000];
	CString buf, buf2;
	//double       wurzel[6];
```

```
	int	zahl;// ende;
	//double	c[6] = {1.0, 2.0, -13.0, 4.0, -30.0, 0.0};

	//long double	lambda = 632.8 * pow(10, -9);//546.1*pow(10, -9);
	long double	phi0	= 70.0 * RAD;
	long double	n0		= 1.0;
	//complex		n2	= complex(3.856, -0.196);// complex(4.08, -
0.028);
	long double	n1[6]	= {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
	long double	d[6]	= {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
	long double	d1[6]	= {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
	long double	root[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0};		// array
for roots of the polynomial
	int			anzahl;

	OnKillfocusAnalysator1();
	OnKillfocusAnalysator2();
	OnKillfocusPolarisator1();
	OnKillfocusPolarisator2();
	OnKillfocuslambda();
	OnKillfocusn2real();
	OnKillfocusn2imag();
	OnKillfocusphi();


	complex n2 = complex((long double)n2real, (long double)n2imag); //n2real +
I*n2imag;
	lambda = lambda*pow(10, -9);

	sprintf (str, "Polarisator1 : %.2lf \n", polarisator1);
	buf += str;
	sprintf (str, "Polarisator2 : %.2lf \n", polarisator2);
	buf += str;
	sprintf (str, "Analysator1 : %.2lf \n", analysator1);
	buf += str;
	sprintf (str, "Analysator2 : %.2lf \n", analysator2);
	buf += str;
	AfxMessageBox(buf);

	Calculation(analysator1, analysator2, polarisator1, polarisator2, lambda,
phi0, n0, n2, n1, d, anzahl, d1, root);
	//Calculation2();

	for (zahl = 1; zahl <= anzahl; zahl++)
	{
		if ( (root[zahl] > 0)&&(n1[zahl] > 1.0) )
		{
			sprintf (str, "n  : %.5f \n", (double)n1[zahl]);
			buf2 += str;
			sprintf (str, "d  : %.5f A\n", (double)d[zahl]*pow(10,10));
		    buf2 += str;
			sprintf (str, "d1 : %.5f A\n", (double)d1[zahl]*pow(10,10));
		    buf2 += str;
		}
	}
	AfxMessageBox(buf2);

	CDialog::OnOK();
}
```

// Utility.cpp : Utility file

```
#include "utility.h"

#ifndef _MATH_H
#include <math.h>
#endif

#ifndef _COMPLEX_H
#include "complex.h"
#endif



long double cotl(long double a)
{
	long double res;

	res = 1.0 / tanl(a*RAD);
	return res;
}
```

```
long double cothl(long double a)
{
	long double res;

	res = 1.0 / tanhl(a*RAD);
	return res;
}
```

// mainfrm.cpp : implementation of the CMainFrame class

```
#include "stdafx.h"
#include "xellip.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
	// same order as in the bitmap 'toolbar.bmp'
	ID_FILE_NEW,
	ID_FILE_OPEN,
	ID_FILE_SAVE,
		ID_SEPARATOR,
	ID_EDIT_CUT,
	ID_EDIT_COPY,
	ID_EDIT_PASTE,
		ID_SEPARATOR,
	ID_FILE_PRINT,
	ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
		!m_wndToolBar.SetButtons(buttons,
		  sizeof(buttons)/sizeof(UINT)))
```

```
	{
		TRACE("Failed to create toolbar\n");
		return -1;	// fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE("Failed to create status bar\n");
		return -1;	// fail to create
	}

	return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

// stdafx.cpp : source file that includes just the standard includes

```
//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

// xellidoc.cpp : implementation of the CXellipDoc class

```
#include "stdafx.h"
#include "xellip.h"

#include "xellidoc.h"

#ifndef _DMWERTE_H
#include "DMwerte.h"
#endif

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CXellipDoc

IMPLEMENT_DYNCREATE(CXellipDoc, CDocument)

BEGIN_MESSAGE_MAP(CXellipDoc, CDocument)
	//{{AFX_MSG_MAP(CXellipDoc)
	ON_COMMAND(ID_EXTRAS_MESSWERTEINGABE, OnExtrasMesswerteingabe)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CXellipDoc construction/destruction

CXellipDoc::CXellipDoc()
{
	// TODO: add one-time construction code here
}

CXellipDoc::~CXellipDoc()
{
}
```

```
BOOL CXellipDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CXellipDoc serialization

void CXellipDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}
}

/////////////////////////////////////////////////////////////////////////////
// CXellipDoc diagnostics

#ifdef _DEBUG
void CXellipDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CXellipDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CXellipDoc commands

void CXellipDoc::OnExtrasMesswerteingabe()
{
	DMwerte dlg;
	if (dlg.DoModal() != IDOK )
		return;
	UpdateAllViews(NULL);
}
```

// xellip.cpp : Defines the class behaviors for the application.
//

```
#include "stdafx.h"
#include "xellip.h"

#include "mainfrm.h"
#include "xellidoc.h"
#include "xellivw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CXellipApp

BEGIN_MESSAGE_MAP(CXellipApp, CWinApp)
	//{{AFX_MSG_MAP(CXellipApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CXellipApp construction

CXellipApp::CXellipApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CXellipApp object

CXellipApp NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
// CXellipApp initialization

BOOL CXellipApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

	SetDialogBkColor();	// Set dialog background color to gray
	LoadStdProfileSettings();  // Load standard INI file options (including
MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CMultiDocTemplate* pDocTemplate;
	pDocTemplate = new CMultiDocTemplate(
		IDR_XELLIPTYPE,
		RUNTIME_CLASS(CXellipDoc),
		RUNTIME_CLASS(CMDIChildWnd),	// standard MDI child frame
		RUNTIME_CLASS(CXellipView));
	AddDocTemplate(pDocTemplate);

	// create main MDI Frame window
	CMainFrame* pMainFrame = new CMainFrame;
	if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
		return FALSE;
	m_pMainWnd = pMainFrame;

	// create a new (empty) document
	OnFileNew();

	if (m_lpCmdLine[0] != '\0')
	{
		// TODO: add command line processing here
	}

	// The main window has been initialized, so show and update it.
	pMainFrame->ShowWindow(m_nCmdShow);
	pMainFrame->UpdateWindow();

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

// Implementation
protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CXellipApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CXellipApp commands
```

// xellivw.cpp : implementation of the CXellipView class

```
#include "stdafx.h"
#include "xellip.h"

#include "xellidoc.h"
#include "xellivw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CXellipView

IMPLEMENT_DYNCREATE(CXellipView, CView)

BEGIN_MESSAGE_MAP(CXellipView, CView)
	//{{AFX_MSG_MAP(CXellipView)
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CXellipView construction/destruction

CXellipView::CXellipView()
{
	// TODO: add construction code here
}

CXellipView::~CXellipView()
{
}

/////////////////////////////////////////////////////////////////////////////
// CXellipView drawing

void CXellipView::OnDraw(CDC* pDC)
{
	CXellipDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	// TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CXellipView printing
```

```
BOOL CXellipView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CXellipView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CXellipView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CXellipView diagnostics

#ifdef _DEBUG
void CXellipView::AssertValid() const
{
	CView::AssertValid();
}

void CXellipView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CXellipDoc* CXellipView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CXellipDoc)));
	return (CXellipDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CXellipView message handlers
```

5.2 Berechnung des Bremsvermögens

Diese Formel zur Berechnung von Bremsvermögen ε von $^4He^+$ in verschiedenen Elementen stammt aus dem Tabellenwerk von Ziegler [ZIE77].

$$\varepsilon = \left(\frac{1}{S_{LOW}} + \frac{1}{S_{HIGH}} \right)^{-1}$$

mit

$$S_{LOW} = A_1 E_0^{A_2}$$

$$S_{HIGH} = \left(\frac{A_3}{E_0 / 1000} \right) \cdot \ln \left[1 + \frac{A_4}{E_0 / 1000} + \frac{A_5 E_0}{1000} \right]$$

Für die Koeffizienten gelten

	Si	N
A_1	2,1	2,51
A_2	0,65	0,4752
A_3	49,34	38,26
A_4	1,788	13,02
A_5	4,133	1,892

E_0 muß in keV angegeben werden.

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent : Priv. Doz. Dr. R. Vianden

Koreferent: Prof. Dr. K. Maier

6 Literatur

[Ana75] K.V. Anand, S.K. Momodu: *Applications of ellipsometry in semiconductor technology*. Electronic Engineering, 51-53, (1976)

[Azz77] R.M.A. Azzam, N.M. Bashara: *Ellisometry and Polarized Light*. North-Holland Publishing Company, New York, (1977)

[Deh95] E.Dehan, P.Temple-Boyer, R.Henda, J.J.Pedroviejo, E.Scheid : *Optical and structural properties of SiO_x and SiN_x materials*. Thin Solid Films 266 (1995), p.14-19

[Dro94] J.-P. Drolet, S.C. Russev, M.I. Boyanov, R.M. Leblanc: *Polynomial inversion of the single transparent layer problem in ellipsometry*. J. Opt. Soc. Am. A/Vol. 11, No12/December 1994, p.3284-91

[Gri96] V.A.Gritsenko, A.D.Milov: *Wigner crystallization of electrons and holes in amorphous silicon nitride. Antiferromagnetic ordering of localized electrons and holes as a result of a resonance exchange interaction*. JETP Letters, Vol.64, No.7, 10.Oct.1996

[Mar90] G.Marx : *Aufbau und Test einer Kurzzeit-Temper-Anlage*. Diplomarbeit, Universität Bonn, (1990)

[Men96] M.Mendel : *Aufbau und Test eines Ellipsometers zu Bestimmung der optischen Eigenschaften dünner dielektrischer Schichten*. Diplomarbeit, Universität Bonn, (1996)

[Mod04] *Handbook of Modern Ion Beam Materials Analysis*. Materials Research Society, Pittsburgh, Pennsylvania

[Möl92] A.Möller : *Aufbau und Test eines elektronischen Temperaturreglers für die RTA-Kurzzeit-Temper-Anlage, Diplomarbeit*, Universität Bonn (1992)

[Zha92] S.-L.Zhang, J.-T.Wang, W.Kaplan, M.Östling : *Silicon nitride films deposited from SiH_2Cl_2-NH_3 by low pressure chemical vapor deposition: kinetics, thermodynamics, composition and structure*. Thin Solid Films, 213 (1992), p.182-191

[Zie77] J.F.Ziegler : *The Stopping and Ranges of Ions in Matter, Vol. 4 : Helium, Stopping Powers and Ranges in All Elements*, Pergamon Press 1977

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Referent : Priv. Doz. Dr. R. Vianden

Koreferent: Prof. Dr. K. Maier

7 Danksagung

Bei Herrn Priv.Doz. Dr. Vianden möchte ich mich ganz herzlich für die interessante Aufgabe und die gute Betreuung bedanken.
Bei Prof. Dr. Maier möchte ich mich für die Übernahme des Koreferats für diese Diplomarbeit bedanken.
Ganz besonders bedanke ich mich bei Prof. Soares und seinen Mitarbeitern in Lissabon für die herzliche Aufnahme. Speziell bei Edouardo Alves, der mir alles gezeigt hat. Außerdem bei Carlos, Rita, Carlos, Filomena, Theresa und Jhongzin, die zu sehr guten Freunden geworden sind.
Mein größter Dank gilt meinen Gruppenmitgliedern. Insbesondere bedanke ich mich bei Christoph Nathusius und Jörn Bartels, die einige Messungen für mich durchgeführt haben, und bei Arndt Dürr für die ausführlichen Diskussionen.
Zum Schluß möchte ich mich noch bei meinen Eltern für ihre große Geduld bedanken.